

# COMPUTER ORGANIZATION AND ARCHITECTURE

İşlevlilik için Tasarım

Tenth Edition



WILLIAM HAMILTON NG'S Yİ OYALIYOR

# **BİLGİSAYAR ORGANİZASYONU VE MİMARİSİ *PERFORMANS İÇİN TASARIM***

**ONUNCU BASKI**

*Bu sayfa kasıtlı olarak boş bırakılmıştır*

# **BİLGİSAYAR ORGANİZASYONU VE MİMARİSİ *PERFORMANS İÇİN TASARIM***

**ONUNCU BASKI**

**William Stallings**

Peter Zeno'nun katkılarıyla  
*Bridgeport Üniversitesi*

Chris Jesshope'un Önsözü ile  
*Profesör (emeritus) Amsterdam Üniversitesi*

**PEARSON**

Boston - Columbus - Hoboken - Indianapolis - New York - San Francisco Amsterdam - Cape Town - Dubai  
- Londra - Madrid - Milano - Münih - Paris - Montreal  
Toronto - Delhi - Mexico City - São Paulo - Sidney - Hong Kong - Seul - Singapur - Taipei - Tokyo

Başkan Yardımcısı ve Yayın Direktörü, ECS: *Marcia J. Horton*  
Genel Yayın : *Tracy Johnson (Dunkelberger)*  
Editör Asistanı: *Kelsey Loanes*  
Program Yöneticisi: *Carole Snyder*  
Ürün Yönetimi Direktörü: *Erin Gregg* Ürün Yönetimi Takım  
Lideri: *Scott Disanno* Proje Yöneticisi: *Robert Engelhardt*  
Medya Ekibi Lideri: *Steve Wright*  
R&P Yöneticisi: *Rachel Youdelman*  
R&P Kudemli Proje Yöneticisi: *Timothy Nicholls*  
Satin Alma Müdürü: *Mary Fischer*  
Kudemli Uzman, Program Planlama ve Destek:  
*Maura Zaldivar-Garcia*

Envanter Müdürü: *Bruce Boundy*  
Pazarlama Başkan Yardımcısı: *Christy Lesko*  
Saha Pazarlama Direktörü: *Demetrius Hall* Ürün Pazarlama  
Müdürlü: *Bram van Kempen* Pazarlama Asistanı: *Jon Bryant*  
Kapak Tasarımcısı: *Marta Samsel* Kapak  
Resmi: © *anderm / Fotolia* Tam Hizmet  
Proje Yönetimi:  
*Mahalatchoumy Saravanan, Jouve Hindistan*  
Yazıcı / Ciltçi: *Edwards Brothers Malloy*  
Kapak Yazıcısı: *Lehigh-Phoenix Color/Hagerstown*  
Yazı karakteri: *Times Ten LT Std 10/12*

**Telif Hakkı© 2016, 2013, 2010 Pearson Education, Inc, Hoboken, NJ 07030.** Tüm hakları saklıdır. Amerika Birleşik Devletleri'nde üretilmiştir.  
Bu yayın Telif Hakkı ile korunmaktadır ve elektronik, mekanik, fotokopi, kayıt veya benzer şekilde herhangi bir biçimde veya herhangi bir yolla yasaklanmış herhangi bir çoğaltma, bir erişim sisteminde depolama veya iletimden önce yayından izin alınmalıdır. Bu çalışmada materyalleri kullanmak üzere izin(ler) almak için lütfen Pearson Higher Education, Permissions Department, 221 River Street, Hoboken, NJ 07030 adresine yazılı bir talep gönderin.

Üreticiler ve satıcılar tarafından ürünlerini ayırt etmek için kullanılan tanımlamaların birçoğu ticari marka olarak iddia edilmektedir. Bu tanımlamaların bu kitapta yer aldığı ve yayınının ticari marka iddiasından haberدار olduğu durumlarda, tanımlamalar ilk büyük harflerle veya büyük harflerle basılmıştır. Diğer kaynaklardan alınan ve bu ders kitabında izin çoğaltılan referanslar ve teşekkür yazıları 833. sayfada yer almaktadır.

Bu kitabın yazarı ve yayıncısı, bu hazırlarken ellerinden gelen tüm çabayı göstermişlerdir. Bu çabalardan etkinliklerini belirlemek için teorilerin ve programların geliştirilmesini, araştırmasını ve test edilmesini içerir. Yazar ve yayıncı, bu programlar veya içerdikleri belgeler ile ilgili olarak açık veya zimni hiçbir garanti vermemektedir.  
bu kitapta. Yazar ve yayıncı, bu programların sağlanması, performansı veya kullanımı ile ilgili veya bunlardan kaynaklanan arzı veya dolaylı zararlarından hiçbir şekilde sorumlu olmayacaktır.

Pearson Education Ltd., Londra  
Pearson Education Australia Pty. Ltd., Sydney Pearson  
Education Singapore, Pte. Ltd.  
Pearson Education North Asia Ltd, Hong Kong Pearson  
Education Canada, Inc, Toronto Pearson Education de  
Mexico, S.A. de C.V. Pearson Education-Japan, Tokyo  
Pearson Education Malaysia, Pte. Ltd. Pearson  
Education, Inc., Hoboken, New Jersey

#### **Kongre Kütüphanesi Yayın Verileri Kataloglaması**

Stallings, William.

Bilgisayar organizasyonu ve mimarisi : performans için tasarım / William Stallings. - Onuncu baskı. sayfa cm

Biblioografik referanslar ve indeks içerir.

ISBN 978-0-13-410161-3 - ISBN 0-13-410161-8 1. BILGISAYAR ORGANİZASYONU. Bilgisayar organizasyonu. 2. Bilgisayar mimarisi.

I. Başlık. QA76.9.C643S73

2016 004.2'2-dc23

2014044367

10 9 8 7 6 5 4 3 2 1



[www.pearsonhighered.com](http://www.pearsonhighered.com)

ISBN-10: 0-13-410161-8  
ISBN-13: 978-0-13-410161-3

*Tricia'ya*

*Sevgili eşim, en nazik ve en kibar  
insan*

*Bu sayfa kasıtlı olarak boş bırakılmıştır*

# İÇİNDEKİLER

Önsöz xiii Önsöz xv

Yazar Hakkında xxiii

## **BİRİNCİ BÖLÜM GİRİŞ 1**

### **Bölüm 1 Temel Kavramlar ve Bilgisayarın Evrimi 1**

- 1.1** Organizasyon ve Mimari 2
- 1.2** Yapı ve İşlev 3
- 1.3** Bilgisayarların Kısa Tarihi 11
- 1.4** Intel x86 Mimarisinin Evrimi 27
- 1.5** Gümülü Sistemler 29
- 1.6** Kol Mimarisi 33
- 1.7** Bulut Bilişim 39
- 1.8** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 42

### **Bölüm 2 Performans Sorunları 45**

- 2.1** Performans için Tasarım 46
- 2.2** Çok Çekirdekli, Mikrofonlar ve GPGPU'lar 52
- 2.3** İçgörü Sağlayan İki Yasa: Ahmdahl Yasası ve Little Yasası 53
- 2.4** Bilgisayar Performansının Temel Ölçütleri 56
- 2.5** Ortalamanın Hesaplanması 59
- 2.6** Benchmarklar ve Teknik Özellikler 67
- 2.7** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 74

## **İKİNCİ BÖLÜM BİLGİSAYAR SİSTEMİ 80**

### **Bölüm 3 Bilgisayar İşlevi ve Ara Bağlantılarına Üst Düzey Bir Bakış 80**

- 3.1** Bilgisayar Bileşenleri 81
- 3.2** Bilgisayar İşlevi 83
- 3.3** Ara Bağlantı Yapıları 99
- 3.4** Otobüs Ara Bağlantısı 100
- 3.5** Noktadan Noktaya Ara Bağlantı 102
- 3.6** PCI Express 107
- 3.7** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 116

### **Bölüm 4 Önbellek 120**

- 4.1** Bilgisayar Bellek Sistemine Genel Bakış 121
- 4.2** Önbellek Bellek İlkeleri 128
- 4.3** Önbellek Tasarımının Unsurları 131
- 4.4** Pentium 4 Önbellek Organizasyonu 149
- 4.5** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 152  
Ek 4A İki Seviyeli Belleklerin Performans Özellikleri 157

## viii İÇİNDEKİLER

### Bölüm 5 Dahili Bellek 165

- 5.1** Yarı İletken Ana Bellek 166
- 5.2** Hata Düzeltme 174
- 5.3** DDR DRAM 180
- 5.4** Flash Bellek 185
- 5.5** Yeni Uçucu Olmayan Katı Hal Bellek Teknolojileri 187
- 5.6** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 190

### Bölüm 6 Harici Bellek 194

- 6.1** Manyetik Disk 195
- 6.2** RAID 204
- 6.3** Katı Hal Sürücüler 212
- 6.4** Optik Bellek 217
- 6.5** Manyetik Bant 222
- 6.6** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 224

### Bölüm 7 Giriş/Çıkış 228

- 7.1** Harici Cihazlar 230
- 7.2** I/O Modülleri 232
- 7.3** Programlanmış I/O 235
- 7.4** Kesme Tahraklı I/O 239
- 7.5** Doğrudan Bellek Erişimi 248
- 7.6** Doğrudan Önbellek Erişimi 254
- 7.7** G/Ç Kanalları ve İşlemciler 261
- 7.8** Harici Arabağlantı Standartları 263
- 7.9** IBM zEnterprise EC12 G/Ç Yapısı 266
- 7.10** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 270

### Bölüm 8 İşletim Sistemi Desteği 275

- 8.1** İşletim Sistemine Genel Bakış 276
- 8.2** Çizelgeleme 287
- 8.3** Bellek Yönetimi 293
- 8.4** Intel x86 Bellek Yönetimi 304
- 8.5** Arm Bellek Yönetimi 309
- 8.6** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 314

## ÜÇÜNCÜ BÖLÜM ARİTMETİK VE MANTIK 318

### Bölüm 9 Sayı Sistemleri 318

- 9.1** Ondalık Sistem 319
- 9.2** Konumsal Sayı Sistemleri 320
- 9.3** İkili Sistem 321
- 9.4** Binary ve Decimal Arasında Dönüşüm 321
- 9.5** Onaltılık Gösterim 324
- 9.6** Anahtar Terimler ve Sorunlar 326

### Bölüm 10 Bilgisayar Aritmetiği 328

- 10.1** Aritmetik ve Mantık Birimi 329
- 10.2** Tamsayı Gösterimi 330
- 10.3** Tamsayı Aritmetiği 335

- 10.4** Kayan Nokta Gösterimi 350
- 10.5** Kayan Noktalı Aritmetik 358
- 10.6** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 367

**Bölüm 11 Dijital Mantık 372**

- 11.1** Boole Cebiri 373
- 11.2** Gates 376
- 11.3** Kombinasyonel Devreler 378
- 11.4** Sıralı Devreler 396
- 11.5** Programlanabilir Mantık Aygıtları 405
- 11.6** Anahtar Terimler ve Sorunlar 409

**DÖRDÜNCÜ BÖLÜM MERKEZİ İŞLEM BİRİMİ 412****Bölüm 12 Komut Kümeleri: Özellikler ve İşlevler 412**

- 12.1** Makine Komutu Özellikleri 413
- 12.2** İşlenen Türleri 420
- 12.3** Intel x86 ve ARM Veri Türleri 422
- 12.4** Operasyon Türleri 425
- 12.5** Intel x86 ve ARM İşlem Türleri 438
- 12.6** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 446 Ek 12A Küçük, Büyük ve İki Kızılderili 452

**Bölüm 13 Komut Kümeleri: Adresleme Modları ve Biçimleri 456**

- 13.1** Adresleme Modları 457
- 13.2** x86 ve ARM Adresleme Modları 463
- 13.3** Talimat Formatları 469
- 13.4** x86 ve ARM Komut Formatları 477
- 13.5** Assembly Dili 482
- 13.6** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 484

**Bölüm 14 İşlemci Yapısı ve İşlevi 488**

- 14.1** İşlemci Organizasyonu 489
- 14.2** Kayıt Kuruluşu 491
- 14.3** Talimat Döngüsü 496
- 14.4** Komut Ardişık Dizilimi 500
- 14.5** x86 İşlemci Ailesi 517
- 14.6** ARM İşlemci 524
- 14.7** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 530

**Bölüm 15 Azaltılmış Komut Kümeli Bilgisayarlar 535**

- 15.1** Komut Yürütme Özellikleri 537
- 15.2** Büyük Bir Kayıt Dosyasının Kullanımı 542
- 15.3** Derleyici Tabanlı Kayıt Optimizasyonu 547
- 15.4** Azaltılmış Komut Kümesi Mimarisi 549
- 15.5** RISC Pipelining 555
- 15.6** MIPS R4000 559
- 15.7** SPARC 565
- 15.8** RISC ve CISC Tartışması 570
- 15.9** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 571

## **X İÇİNDEKİLER**

### **Bölüm 16 Komut Düzeyinde Paralellik ve Süperkalar İşlemciler 575**

- 16.1** Genel Bakış 576
- 16.2** Tasarım Sorunları 581
- 16.3** Intel Core Mikro Mimari 591
- 16.4** ARM Cortex-A8 596
- 16.5** ARM Cortex-M3 604
- 16.6** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 608

### **BEŞİNCİ BÖLÜM PARALEL ÖRGÜTLENME 613**

#### **Bölüm 17 Paralel İşleme 613**

- 17.1** Çoklu İşlemci Kuruluşları 615
- 17.2** Simetrik Çoklu İşlemciler 617
- 17.3** Önbellek Tutarlılığı ve MESI Protokolü 621
- 17.4** Multithreading ve Çip Çoklu İşlemciler 628
- 17.5** Kümeler 633
- 17.6** Tekdüze Olmayan Bellek Erişimi 640
- 17.7** Bulut Bilişim 643
- 17.8** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 650

#### **Bölüm 18 Çok Çekirdekli Bilgisayarlar 656**

- 18.1** Donanım Performans Sorunları 657
- 18.2** Yazılım Performans Sorunları 660
- 18.3** Çok Çekirdekli Organizasyon 665
- 18.4** Heterojen Çok Çekirdekli Organizasyon 667
- 18.5** Intel Core i7-990X 676
- 18.6** ARM Cortex-A15 MPCore 677
- 18.7** IBM zEnterprise EC12 Mainframe 682
- 18.8** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 685

#### **Bölüm 19 Genel Amaçlı Grafik İşleme Birimleri 688**

- 19.1** Cuda Temelleri 689
- 19.2** GPU'ya karşı CPU 691
- 19.3** GPU Mimarisi Genel Bakış 692
- 19.4** Intel'in Gen8 GPU'su 701
- 19.5** GPU Ne Zaman Yardımcı İşlemci Olarak Kullanılır 704
- 19.6** Anahtar Terimler ve Gözden Geçirme Soruları 706

### **ALTINCI BÖLÜM KONTROL ÜNITESİ 707**

#### **Bölüm 20 Kontrol Ünitesinin Çalışması 707**

- 20.1** Mikro-Operasyonlar 708
- 20.2** İşlemcinin Kontrolü 714
- 20.3** Kablolu Uygulama 724
- 20.4** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 727

#### **Bölüm 21 Mikro Programlı Kontrol 729**

- 21.1** Temel Kavramlar 730
- 21.2** Mikro Komut Sıralaması 739

<b>21.3</b>	Mikro Komut Yürütme 745
<b>21.4</b>	TI 8800 755
<b>21.5</b>	Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 766

**Ek A Bilgisayar Organizasyonu ve Mimarisi Öğretmek için Projeler 768**

<b>A.1</b>	İnteraktif Simülasyonlar 769
<b>A.2</b>	Araştırma Projeleri 771
<b>A.3</b>	Simülasyon Projeleri 771
<b>A.4</b>	Assembly Dili Projeleri 772
<b>A.5</b>	Okuma/Rapor Ödevleri 773
<b>A.6</b>	Yazma Ödevleri 773
<b>A.7</b>	Test Bankası 773

**Ek B Assembly Dili ve İlgili Konular 774**

<b>B.1</b>	Assembly Dili 775
<b>B.2</b>	Montajcılar 783
<b>B.3</b>	Yükleme ve Bağlama 787
<b>B.4</b>	Anahtar Terimler, Gözden Geçirme Soruları ve Problemler 795

**Referanslar 800**

**Dizin 809**

**Krediler 833**

**ÇEVİRİMİÇİ EKLER<sup>1</sup>**

<b>Ek C</b>	Sistem Otobüsleri
<b>Ek D</b>	Protokoller ve Protokol Mimarileri Ek E
	Karıştırma
<b>Ek F</b>	Mağdur Önbellek Stratejileri
<b>Ek G</b>	Aralıklı Bellek
<b>Ek H</b>	Uluslararası Referans Alfabesi Ek I
	Yığınlar
<b>Ek J</b>	Thunderbolt ve Infiniband
<b>Ek K</b>	Sanal Bellek Sayfa Değiştirme Algoritmaları Ek L
	Karma Tablolar
<b>Ek M</b>	Özyinelemeli Prosedürler
<b>Ek N</b>	Ek Talimat Boru Hattı Konuları Ek O
	Zamanlama Diyagramları
<b>Sözlük</b>	

<sup>1</sup> Çevrimiçi bölümler, ekler ve diğer belgeler Premium İçeriktir ve bu kitabın önündeki erişim kartı aracılığıyla edinilebilir.

*Bu sayfa kasıtlı olarak boş bırakılmıştır*



## ÖNSÖZ

---

*tarafindan Chris Jesshope*

*Profesör (emeritus) Amsterdam Üniversitesi*

*Paralel Bilgisayarlar kitabının yazarı (R W Hockney ile birlikte), 1981 & 1988*

Uzun yıllardır bilgisayar organizasyonu ve mimarisi alanında faaliyet gösteren William Stallings'in bu konudaki kapsamlı kitabının yeni baskısı için bu önsözü yazmak benim için bir zevktir. Bunu yaparken, kendimi bu konuya ilgilendığım süre boyunca konudaki eğilimler ve değişiklikler üzerine düşünürken buldum. Ben de bilgisayar mimarisyle önemli yeniliklerin ve bozulmaların yaşandığı bir dönemde ilgilenmeye başladım. Bu bozulma sadece teknolojideki ilerlemelerle değil, belki de daha önemlisi bu teknolojiye erişimle . VLSI buradaydı ve VLSI tasarımları sınıflarda öğrencilerin kullanımına açtı. Bunlar heyecan verici zamanlardı. Anabilgisayar tarzı bir bilgisayarn tek bir silikon çip üzerine entegre edilebilmesi bir dönüm noktasıydı, ancak bunun akademik bir araştırma ekibi tarafından başarılı olmuş olmasının başarıyı oldukça benzersiz kılıyordu. Bu dönem, bilgisayar mimarisinde yenilik ve çeşitlilik ile karakterize edildi ve ana trendlerden biri paralellik alanındaydı. 1970'lerde, bilgisayar mimarisinde açık paralellliğin erken bir örneği olan ve tesadüfen tüm yarı iletken belleklere öncülüük eden Illiac IV ile ilgili uygulamalı deneyimim oldu. Bu etkileşim, ki kesinlikle öyledi, bilgisayar mimarisi ve organizasyonuna, özellikle de bilgisayar mimarisindeki açık paralelliğe olan ilgimi başlattı.

1980'ler boyunca ve 1990'ların başında bu alanda araştırmalar gelişti ve büyük bir kısmı üniversitelerde girişimleri aracılığıyla pazara sunulan çok sayıda yenilik ortaya çıktı. Ancak ironik bir şekilde, bu eğilimi tersine çeviren de aynı teknoloji oldu. Çeşitlilik yerini yavaş yavaş bilgisayar sistemlerinde neredeyse monokültüre bıraktı ve sadece birkaç yönere seti mimarisinde ilerleme kaydedildi. Endüstrinin kılavuz çizgisi haline gelen ve kendi kendini gerçekleştiren bir öngörü olan Moore yasası, temel aygit hızlarının ve entegrasyon yoğunluklarının her ikisinin katlanarak arttığı ve ikincisinin her 18 ayda bir ikiye katlandığı anlamına . Hız artışı bilgisayar mimarları için meşhur bedava öğle yemeğiydi ve entegrasyon seviyeleri mikro mimari seviyesinde daha fazla karmaşıklığa ve yeniliğe izin veriyordu. Bu bedava öğle yemeğinin elbette bir malîyeti vardı: Moore yasasını yerine getirmek için gereken sermaye yatırıminin katlanarak büyümesi ve bunun da en son teknolojilere erişimi bir kez daha sınırlaması. Dahası, çoğu kullanıcı, tuzakları ve zorluklarıyla birlikte paralel bilgisayarlardaki yeniliklere yatırım yapmaktansa bir sonraki nesil ana akım işlemciyi daha kolay buldu. Bunun istisnaları, en üst düzeyde performans gerektiren birkaç büyük kurumdu; iklim modellemesi gibi büyük ölçekli bilimsel simülasyonlar ve ayrıca kod kırmaya yönelik güvenlik hizmetlerimiz bunun iki güncel örneğidir. İçin

Diger herkes için oyunun adı uyumluluktu ve bundan yararlanan iki komut seti mimarisi x86 ve ARM idi; ikincisi gömülü sistemlerde, ilki ise hemen hemen diğer her şeyle. Paralellik bu ISA'ların uygulanmasında hala vardı, sadece örtüktü, onu yönlendiren komut akışında değil mimari tarafından kullanılıyordu.

1990'ların sonu ve 2000'lerin başı boyunca, tek çekirdekli bilgisayar sistemlerinde dolaylı olarak konpara biriminden yararlanmaya yönelik bu yaklaşım gelişti. Ancak, mantık yoğunluğunun üstel büyümeye rağmen, bu dönemi sona erdiren, kullanılan tekniklerin maliyeti olmuştur. Süperskalar işlemcilerde mantık maliyetleri sorun genişliği ile doğrusal olarak artmazken (par-allelizm), bazı bileşenler sorun genişliğinin karesi ve hatta küpü kadar artmaktadır. Mantıktaki üstel büyümeye bu sürekli gelişimi sürdürübilese de, iki büyük tuzak vardı: eşzamanlılığı zorunlu programlardan dolaylı olarak ortaya çıkarmak giderek zorlaşıyordu ve bu nedenle komut sorunu yuvalarının kullanımındaki verimlilik azaldı. Belki de daha önemlisi, teknoloji performans kazanımlarının önündeki yeni bir engelle, yani güç dağılımıyla karşı karşıyaydı ve birkaç süperskalar geliştirme, işlerindeki silikon çok sıcak olacak için durduruldu. Bu kısıtlamalar, uyumluluk zorluklarına rağmen açık paralelliğin kullanılmasını zorlunu kılmıştır. Öyle görünüyor ki yenilik ve çeşitlilik bu alanı yeni araştırmalara açıyor.

Belki de 1980'lerden bu yana bu alanda çalışmak bu kadar ilginç olmamıştı. Farklılığın ekonomik bir gerçeklik olduğu, ana akım işlemcilerdeki sayı genişliğinin azalması (örtük paralellik) ve çekirdek sayısının artması (açık) ile görülebilir. Ancak asıl soru, hem uygulama hem de sistem düzeyinde bundan nasıl yararlanılacağıdır. Burada hala çözülmeli gerekken önemli zorluklar vardır. Superscalar işlemciler, tek bir komut akışından paralelliği çıkarmak için işlemciye güvenmektedir. Peki ya vurguya değiştirirsek ve maksimum paralelliğe sahip bir komut akışı sağlarsak, farklı seviyelerde açık paralellik gerektiren farklı konfigürasyonlarda ve/veya işlemci nesillerinde bundan nasıl faydalananız? Bu nedenle, ISA'da yakalanan bu maksimum eşzamanlılığı çekirdeklerin mevcut konfigürasyonuna uyacak şekilde sıralayan ve programlayan bir mikro mimariye sahip olmak mümkün müdür, böylece açık paralellik dünyasında aynı uyumluluğu elde edebilir miyiz? Bu, verimlilik için silikon içinde işletim sistemleri gerektirir mi?

Bunlar bugün karşı karşıya olduğumuz sorulardan sadece bazları. Bu soruları ve daha fazlasını yanıtlamak için bilgisayar organizasyonu ve mimarisi konusunda sağlam bir temel gerekiyor ve William Stallings'in bu kitabı çok zamanında ve kapsamlı bir temel sağlıyor. Oldukça karmaşık olabilecek konuları görünür bir basitleştirme ele alarak, gerekli temel bilgilere eksiksiz bir giriş yapıyor. Ayrıca, bu alanda geçmişte ve şu anda yeniliklerin gerçekleştiği daha yeni gelişmeleri de ele alıyor. Örnekler süperskalar ve açıkça paralel çoklu çekirdeklerdir. Dahası, bu son baskı, genel amaçlı kullanım için GPU'ların tasarımını ve kullanımını ve bulut bilişimdeki en son trendler gibi her ikisi de yakın zamanda ana akım haline gelen iki çok yeni konuya içermektedir. Kitap, ele alınan teorik konuları vurgulamak için baştan sona örneklerden iyi bir şekilde yararlanıyor ve bu örneklerin çoğu en yaygın kullanılan iki ISA'daki, yani x86 ve ARM'deki gelişmelerden alınıyor. Tekrarlamak gerekirse, bu kitap eksiksizdir ve okuması bir zevktir ve umarım daha fazla genç araştırmacıyı benim son 40 yılda keyif aldığım aynı yolda ilerletir!



# ÖNSÖZ

## ONUNCU BASIMDAKİ YENİLİKLER

Bu kitabın dokuzuncu baskısının yayına, alan sürekli yeniliklere ve gelişmelere sahne oldu. Bu yeni baskıda, tüm alanı geniş ve kapsamlı bir şekilde ele almayı sürdürken bu değişiklikleri yakalamaya çalıştım. Bu revizyon sürecine başlamak için, bu kitabın dokuzuncu baskısı, konuyu öğreten bir dizi profesör ve alanda çalışan profesyoneller tarafından kapsamlı bir şekilde gözden geçirildi. Sonuç olarak, birçok yerde anlatım netleştirilmiş, sıklaştırılmış ve resimler iyileştirilmiştir.

Pedagojiyi ve kullanım kolaylığını geliştirmeye yönelik bu iyileştirmelerin ötesinde, kitap boyunca önemli değişiklikler yapılmıştır. Aşağı yukarı aynı bölüm organizasyonu korumus, ancak materyallerin çoğu gözden geçirilmiş ve yeni materyaller eklenmiştir. En kayda değer değişiklikler aşağıdaki gibidir:

- **GPGPU |Grafik İşleme Birimlerinde (GPU'lar) Genel Amaçlı Hesaplama|:** Son yıllarda en önemli yeni gelişmelerden biri, büyük veri dizileri içeren çok çeşitli uygulamaları ele almak için geleneksel CPU'larla koordineli olarak çalışmak üzere GPGPU'ların geniş çapta benimsenmesi olmuştur. GPGPU'lar konusuna yeni bir bölüm ayrılmıştır.
- **Heterojen çok çekirdekli işlemciler:** Çok çekirdekli mimarideki en son gelişme heterojen çok çekirdekli işlemcidir. Çok çekirdekli işlemcilerle ilgili bölümde yer alan yeni bir bölüm, çeşitli heterojen çok çekirdekli işlemci türlerini incelemektedir.
- **Gömülü sistemler:** Bölüm 1'deki gömülü sistemlere genel bakış, gömülü teknolojinin mevcut durumunu yansıtacak şekilde önemli ölçüde revize edilmiş ve genişletilmiştir.
- **Mikrodenetleyiciler:** Sayısal olarak bakıldığından, şu anda kullanılan bilgisayarların neredeyse tamamı gömülü mikro denetleyicilerdir. Bölüm 1'de gömülü sistemlerin ele alımı artık mikrodenetleyicileri de kapsamaktadır. ARM Cortex-M3 mikrodenetleyicisi metin boyunca örnek sistem olarak kullanılmaktadır.
- **Bulut bilişim:** Bu baskında yeni olan, Bölüm 1'de genel bir bakış ve Bölüm 17'de daha ayrıntılı bir şekilde ele alınan bulut bilişim tartışmasıdır.
- **Sistem performansı:** Sistem performansı konuları gözden geçirilmiş, genişletilmiş ve daha net ve kapsamlı bir şekilde ele alınmak üzere yeniden düzenlenmiştir. Bölüm 2 bu konuya ayrılmıştır ve sistem performansı konusu kitabı boyunca yer almaktadır.

- **Flaş bellek:** Flaş bellek kapsamı güncellenmiş ve genişletilmiştir ve artık dahili bellek (Bölüm 5) ve harici bellek (Bölüm 6) için flaş bellek teknolojisi ve organizasyonu hakkında bir tartışma içermektedir.
- **Uçucu Olmayan RAM:** Bu baskında bellek hiyerarşisinde farklı konumlarda yer alan üç önemli yeni uçucu olmayan katı hal RAM teknolojisi ele alınmaktadır: STT-RAM, PCRAM ve ReRAM.
- **Doğrudan önbellek erişimi (DCA):** Çok yüksek hızlı ağ bağlantılarının protokol işleme taleplerini karşılamak için Intel ve diğer üreticiler, geleneksel doğrudan bellek erişimi (DMA) yaklaşımlarından çok daha fazla verim sağlayan DCA teknolojilerini geliştirmiştir. Bu baskında yeni olan Bölüm 7, DCA'yı biraz ayrıntılı olarak incelemektedir.
- **Intel Çekirdek Mikro Mimarisi:** Önceki baskında olduğu gibi, Intel x86 ailesi tüm baskı boyunca ana örnek sistem olarak kullanılmıştır. Yeni Intel sistemlerini, özellikle de hem PC hem de sunucu ürünlerinde kullanılan Intel Core Microarchitecture'ı yansıtacak şekilde güncellenmiştir.
- **Ev ödevi problemleri:** Öğrencilerin pratik yapabilmeleri için çözümü ek ev ödevi problemlerinin sayısı artırılmıştır.

## ACM/IEEE BİLGİSAYAR BİLİMLERİ MÜFREDATININ DESTEKLENMESİ 2013

Kitap hem akademik hem de profesyonel bir kitleye yöneliktir. Bir ders kitabı olarak, bilgisayar bilimleri, bilgisayar mühendisliği ve elektrik mühendisliği bölümleri için bir veya iki dönemlik bir lisans dersi olarak tasarlanmıştır. Bu baskı ACM/IEEE Bilgisayar Bilimleri Müfredatı 2013 (CS2013) önerilerini desteklemek tasarlanmıştır. CS2013 tüm ders çalışmalarını üç kategoriye ayırmaktadır: Çekirdek-Kademe 1 (tüm konular müfredata edilmelidir); Çekirdek-Kademe-2 (tüm veya neredeysse tüm konular dahil edilmelidir); ve Seçmeli (genişlik ve derinlik sağlamak için arzu edilir). Mimari ve Organizasyon (AR) alanında CS2013, her biri bir dizi alt konuya sahip beş Kademe-2 konusu ve üç Seçmeli konu içermektedir. Bu metin CS2013 tarafından listelenen sezik konunun tamamını kapsamaktadır. Tablo P.1, bu ders kitabında AR Bilgi Alanı için sağlanan desteği göstermektedir.

**Tablo P.1** CS2013 Mimari ve Organizasyon (AR) Bilgi Alanının Kapsamı

IAS Bilgi Birimleri	Konular	Ders Kitabı Kapsamı
<b>Sayısal Mantık ve Sayısal Sistemler (Seviye 2)</b>	<ul style="list-style-type: none"> <li>● Bilgisayar mimarisine genel bakış ve tarihçesi</li> <li>● Kombinasyonel ve sıralı mantık / Temel bir kombinasyonel sıralı mantık yapı taşı olarak alan programlanabilir kapı dizileri</li> <li>● Çoklu temsiller/yorumlama katmanları (donanım sadece başka bir katmandır)</li> <li>● Fiziksel kısıtlamalar (kapı gecikmeleri, fan girişi, fan çıkışı, enerji/güç)</li> </ul>	-Bölüm 1 -Bölüm 11
<b>Verilerin Makine Seviyesinde Temsili (Kademe 2)</b>	<ul style="list-style-type: none"> <li>● Bitler, baytlar ve kelimeler</li> <li>● Sayısal veri gösterimi ve sayı tabanları</li> <li>● Sabit ve kayan noktalı sistemler</li> <li>● İşaretli ve iki tümleyenli gösterimler</li> <li>● Sayısal olmayan verilerin gösterimi (karakter kodları, grafiksel veriler)</li> </ul>	-Bölüm 9 -Bölüm 10

IAS Bilgi Birimleri	Konular	Ders Kitabı Kapsamı
<b>Montaj Seviyesi Makine Organizasyonu (Kademe 2)</b>	<ul style="list-style-type: none"> <li>● Von Neumann makinesinin temel organizasyonu</li> <li>● Kontrol birimi; komut getirme, kod çözme ve yürütme</li> <li>● Komut setleri ve türleri (veri manipülasyonu, kontrol, )</li> <li>● Assembly/makine dili programlama</li> <li>● Talimat formatları</li> <li>● Adresleme modları</li> <li>● Alt yordam çağrı ve dönüş mekanizmaları (çapraz referans PL/Dil Çeviri ve Yürütme)</li> <li>● G/C ve kesmeler</li> <li>● Paylaşımlı bellek çoklu işlemciler/çok çekirdekli organizasyon</li> <li>● SIMD ile MIMD ve Flynn Taksonomisine Giriş</li> </ul>	<ul style="list-style-type: none"> <li>-Bölüm 1</li> <li>-Bölüm 7</li> <li>-Bölüm 12</li> <li>-Bölüm 13</li> <li>-Bölüm 17</li> <li>-Bölüm 18</li> <li>-Bölüm 20</li> <li>-Bölüm 21</li> <li>-Ek A</li> </ul>
<b>Bellek Sistemi Organizasyonu ve Mimarisi (Kademe 2)</b>	<ul style="list-style-type: none"> <li>● Depolama sistemleri ve teknolojileri</li> <li>● Bellek hiyerarşisi: zamansal ve mekansal yerellik</li> <li>● Ana bellek organizasyonu ve işlemleri</li> <li>● Gecikme, döngü süresi, bant genişliği ve serpiştirme</li> <li>● Önbellek bellekleri (adres eşleştirme, blok boyutu, değiştirme ve saklama politikası)</li> <li>● Çok işlemcili tutarlılığı/Çekirdekler arası senkronizasyon için bellek sisteminin kullanılması/atomik bellek işlemleri</li> <li>● Sanal bellek (sayfa tablosu, TLB)</li> <li>● Arıza yönetimi ve güvenilirlik</li> </ul>	<ul style="list-style-type: none"> <li>-Bölüm 4</li> <li>-Bölüm 5</li> <li>-Bölüm 6</li> <li>-Bölüm 8</li> <li>-Bölüm 17</li> </ul>
<b>Arayüz ve (Kademe 2)</b>	<ul style="list-style-type: none"> <li>● G/C temelleri: el süküşma, arabelleğe alma, programlanmış G/C, kesme güdümlü G/C</li> <li>● Kesme yapıları: vektörel ve öncelikli, kesmeler arası onaylama</li> <li>● Harici depolama, fiziksel organizasyon ve sürücüler</li> <li>● Veri yolları: veri yolu protokollerı, tahlkim, doğrudan bellek erişimi (DMA)</li> <li>● RAID mimarileri</li> </ul>	<ul style="list-style-type: none"> <li>-Bölüm 3</li> <li>-Bölüm 6</li> <li>-Bölüm 7</li> </ul>
<b>Fonksiyonel Organizasyon (Seçmeli)</b>	<ul style="list-style-type: none"> <li>● Komut ardışık düzlenmesi, tehlike tespiti ve çözümü dahil olmak üzere basit veri yollarının uygulanması</li> <li>● Kontrol ünitesi: Kablolu gerçekleştirmeye ve mikroprogramlı gerçekleştirmeye</li> <li>● Komut ardışık dizilimi</li> <li>● Komut düzeyinde paralelliğe (ILP) giriş</li> </ul>	<ul style="list-style-type: none"> <li>-Bölüm 14</li> <li>-Bölüm 16</li> <li>-Bölüm 20</li> <li>-Bölüm 21</li> </ul>
<b>Çoklu İşlem ve Alternatif Mimariler (Seçmeli)</b>	<ul style="list-style-type: none"> <li>● Örnek SIMD ve MIMD komut setleri ve mimarileri</li> <li>● Ara bağlantı ağları</li> <li>● Paylaşımlı çok işlemcili bellek sistemleri ve bellek tutarlılığı</li> <li>● Çok işlemcili önbellek tutarlılığı</li> </ul>	<ul style="list-style-type: none"> <li>-Bölüm 12</li> <li>-Bölüm 13</li> <li>-Bölüm 17</li> </ul>
<b>Performans Artırıcılar (Seçmeli)</b>	<ul style="list-style-type: none"> <li>● Süperskalar mimarı</li> <li>● Dallanma tahmini, Spekulatif yürütme, Sıra dışı yürütme</li> <li>● Prefetching</li> <li>● Vektör işlemciler ve GPU'lar</li> <li>● Çoklu iş parçasığı için donanım desteği</li> <li>● Ölçeklenebilirlik</li> </ul>	<ul style="list-style-type: none"> <li>-Bölüm 15</li> <li>-Bölüm 16</li> <li>-Bölüm 19</li> </ul>

## HEDEFLER

Bu kitap bilgisayarların yapısı ve işlevi hakkındadır. Amacı, günümüz bilgisayar sistemlerinin doğasını ve özelliklerini olabildiğince açık ve eksiksiz bir şekilde sunmaktır.

Bu görev birkaç nedenden dolayı zordur. Birincisi, birkaç dolara mal olan tek çipli mikroişlemcilerden on milyonlarca dolara mal olan süper bilgisayarlara kadar bilgisayar adını haklı olarak talep edebilecek muazzam çeşitlilikte ürünler bulunmaktadır. Çeşitlilik sadece maliyet açısından değil, boyut, performans ve uygulama açısından da kendini göstermektedir. İkinci olarak, bilgisayar teknolojisini her zaman karakterize eden hızlı değişim temposu hiç durmadan devam etmektedir. Bu değişiklikler, bilgisayar bileşenlerini oluşturmak için kullanılan temel entegre devre teknolojisinden, bu bileşenlerin birleştirilmesinde paralel organizasyon özelliklerinin artan kullanımına kadar bilgisayar teknolojisini tüm yönlerini kapsamaktadır.

Bilgisayar alanındaki çeşitliliğe ve değişim hızına rağmen, bazı temel kavramlar sürekli olarak geçerlidir. Bu kavramların uygulanması teknolojinin mevcut durumuna ve tasarımcının fiyat/performans hedeflerine bağlıdır. Bu kitabın amacı, bilgisayar organizasyonu ve mimarisinin temelleri hakkında kapsamlı bir tartışma sağlamak ve bunları çağdaş tasarımlarıyla ilişkilendirmektir.

Alt başlık, bu kitabın konusunu ve yaklaşımını ortaya koymaktadır. Bilgisayar sistemlerini yüksek performans elde edecek şekilde tasarlamak her zaman önemli olmuştur, ancak bu gereklilik hiçbir zaman bugün olduğundan daha güçlü ya da karşılanması daha zor olmamıştır. İşlemci hızı, bellek hızı, bellek kapasitesi ve ara bağlantı veri hızları dahil olmak üzere bilgisayar sistemlerinin tüm temel performans özellikleri hızla artmaktadır. Üstelik bunlar farklı oranlarda artmaktadır. Bu durum, tüm unsurların performansını ve kullanımını en üst düzeye çıkarın dengeli bir sistem tasarlamayı zorlaştırmaktadır. Dolayısıyla, bilgisayar tasarıımı giderken bir alandaki yapıyı ya da işlevi değiştirmeden başka bir alandaki performans uyumsuzluğunu telsiz etme oygununa dönmektedir. Bu oyuncunun kitap boyunca sayısız tasarım kararında oynadığını göreceğiz.

Her sistem gibi bir bilgisayar sistemi de birbirile ilişkili bir dizi bileşenden oluşur. Sistem en iyi Gekilde yapı -bileGenlerin birbirine bağlanma Gekli- ve fonksiyon ayrı ayrı çalışması- açısından tanımlanabilir. Ayrıca, bir bilgisayarın organizasyonu hiyerarşiktir. Her bir ana bileGen, ana alt bileGenlerine ayrılarak ve bunların yapısı ve işlevi daha ayrıntılı olarak açıklanabilir. Açıklık ve kolay anlaşılabilirlik için, bu hiyerarşik organizasyon bu kitapta yukarıdan aşağıya doğru anlatılmaktadır:

- **Bilgisayar sistemi:** Başlıca bileşenler işlemci, bellek, I/O'dur.
- **İşlemci:** Başlıca bileşenler kontrol birimi, kayıtlar, ALU ve komut yürütme birimidir.
- **Kontrol ünitesi:** Tüm proses bileşenlerinin çalışması ve koordinasyonu için kontrol sinyalleri sağlar. Geleneksel , ana bileşenleri kontrol belgesi, mikro komut sıralama mantığı ve yazmaçlar olan bir mikro programlama uygulaması kullanılmıştır. Daha yakın zamanlarda, mikro programlama daha az öne çıkmıştır ancak önemli bir uygulama teknigi olmaya devam etmektedir.

Amaç, malzemeyi yeni malzemeyi açık bir bağlamda tutacak şekilde sunmaktadır. Bu, okuyucunun kaybolma olasılığını en aza indirmeli ve aşağıdan yukarıya bir yaklaşımından daha iyi motivasyon sağlamalıdır.

Tartışma boyunca, sistemin yönleri hem mimari (bir makine dili programcısı tarafından görülebilen sistemin nitelikleri) hem de organizasyon (gerçekleştirilen operasyonel birimler ve bunların ara bağlantıları) açısından ele alınmaktadır.

## ÖRNEK SİSTEMLER

Bu metin, okuyucuyu çağdaş işletim sistemlerinin tasarım ilkeleri ve uygulama sorunları hakkında bilgilendirmeyi amaçlamaktadır. Buna göre, salt kavramsal ya da teorik bir yaklaşım yetersiz kalacaktır. Kavramları açıklamak ve bunları yapılması gereken gerçek dünya tasarım seçimleriyle ilişkilendirmek için iki işlemci ailesi örnek olarak seçilmiştir:

- **Intel x86 mimarisi:** x86 mimarisi, gömülü olmayan bilgisayar sistemleri için en yaygın kullanılanıdır. X86 esasen bazı RISC özelliklerine sahip karmaşık bir komut seti bilgisayarıdır (CISC). X86 ailesinin son üyeleri süperskalar ve çok çekirdekli tasarım ilkelerini kullanmaktadır. X86 mimarisindeki özelliklerin evrimi, bilgisayar mimarisindeki tasarım ilkelerinin çögünün evrimine ilişkin benzersiz bir vaka çalışması sunmaktadır.
- **ARM:** ARM mimarisi, cep telefonlarında, iPod'larda, uzaktan sensör ekipmanlarında ve diğer birçok cihazda kullanılan tartışmasız en yaygın kullanılan gömülü işlemcidir. ARM esasen bir indirgenmiş komut seti bilgisayarıdır (RISC). ARM ailesinin son üyeleri süperskalar ve çok çekirdekli tasarım ilkelerini kullanmaktadır.

Bu kitaptaki örneklerin hepsi olmasa da birçoğu bu iki bilgisayar ailesinden alınmıştır. Hem çağdaş hem de tarihi çok sayıda başka sistem de önemli bilgisayar mimarisi tasarım özelliklerine örnekler sunmaktadır.

## METNİN PLANI

Kitap altı bölüm halinde düzenlenmiştir:

- Genel Bakış
- Bilgisayar sistemi
- Aritmetik ve mantık
- Merkezi işlem birimi
- Çok çekirdekli dahil olmak üzere paralel organizasyon
- Kontrol ünitesi

Kitap, interaktif simülasyonların kullanımı ve tartışmayı netleştirmek için çok sayıda şekil ve tablo da dahil olmak üzere bir dizi pedagojik özellik içermektedir. Her bölüm anahtar kelimelerin bir listesini, gözden geçirme sorularını, ev ödevi problemlerini ve ileri okuma önerilerini içermektedir. Kitapta ayrıca kapsamlı bir sözlük, sık kullanılan kısaltmaların bir listesi ve bir kaynakça yer almaktadır.

## EĞİTMEN DESTEK MALZEMELERİ

Eğitmenler için destek materyalleri, yayının [www.pearsonhighered](http://www.pearsonhighered.com) web sitesi üzerinden ulaşılabilen bu ders kitabının **Eğitmen Kaynak Merkezi'nde (IRC)** mevcuttur.  
[.com/stallings](http://www.pearsonhighered.com/stallings) adresinden veya bu adresdeki "Eğitmenler için Pearson Kaynakları" bağlantısına tıklayarak

kitabın WilliamStallings.com/ComputerOrganization adresindeki Companion Web sitesi. IRC'ye erişim sağlamak için lütfen pearsonhighered.com/educator/relocator/requestSalesRep.page adresinden yerel Pearson satış temsilcinizle iletişime geçin veya 1-800-526-0485 numaralı telefondan Pearson Faculty Services'i arayın. IRC aşağıdaki materyalleri sağlar:

- **Projeler el kitabı:** Belgeler ve taşınabilir yazılım dahil olmak üzere proje kaynakları, ayrıca bu Önsöz'de daha sonra listelenen tüm proje kategorileri için önerilen proje ödevleri.
- **Çözüm kılavuzu:** Bölüm sonu Gözden Geçirme Soruları ve Problemleri için Çözümler.
- **PowerPoint slaytları:** Tüm bölümleri kapsayan, ders anlatımında kullanılmaya uygun bir dizi slayt.
- **PDF dosyaları:** Kitaptaki tüm şekil ve tabloların kopyaları.
- **Test bankası:** Bölüm bölüm sorulardan oluşan bir set.
- **Örnek ders programları:** Metin, bir sömestrde rahatlıkla işlenebilecek olandan daha fazla materyal içermektedir. Bu nedenle, eğitmenlere metnin sınırlı bir süre içinde kullanılmasına rehberlik edecek birkaç örnek ders programı sunulmaktadır. Bu örnekler, profesörlerin ilk baskısı ile elde ettikleri gerçek dünya deneyimlerine dayanmaktadır.

WilliamStallings.com/ComputerOrganization adresindeki **Companion Web sitesi** (Eğitmen Kaynakları bağlantısına tıklayın) aşağıdakileri içerir:

- Bu kitap kullanılarak öğretilen diğer dersler için Web sitelerine bağlantılar.
- Bu kitabı kullanan eğitmenlerin birbirleriyle ve yazarla bilgi, öneri ve soru alışverişinde bulunabilecekleri bir İnternet posta listesi için kayıt bilgileri.

## ÖĞRENCİ KAYNAKLARI



Bu yeni baskı için, öğrenciler için muazzam miktarda orijinal destekleyici materyal iki Web konumunda çevrimiçi olarak kullanıma sunulmuştur. WilliamStallings.com/ComputerOrganization adresindeki **Companion Web Sitesi** (Öğrenci Kaynakları bağlantısına tıklayın), bölümlere göre düzenlenmiş ilgili bağlantıların bir listesini ve kitap için bir hata sayfası içerir.

Bu ders kitabı yeni satın almak, okuyucuya aşağıdaki materyalleri içeren **Premium İçerik Sitesine** altı aylık erişim hakkı verir:

- **Çevrimiçi bölümler:** Kitabın boyutunu ve maliyetini sınırlamak için, kitabın iki bölümü PDF formatında sağlanmıştır. Bölümler bu kitabındaki bölümlerde listelenmiştir.
- **Çevrimiçi ekler:** Metinde bulunan materyali destekleyen ancak basılı metinde yer almazı gerekmeyen çok sayıda ilginç konu bulunmaktadır. Toplam 13 ek, ilgilenen öğrenciler için bu konuları kapsamaktadır. Ekler bu kitabındaki bölümlerde listelenmiştir.
- **Ev ödevi problemleri ve çözümleri:** Öğrencinin materyali anlamasına yardımcı olmak için, çözümleriyle birlikte ayrı bir ev ödevi problemleri seti mevcuttur. Öğrenciler bu problemlerin çözümleri üzerinde çalışarak ve daha sonra cevaplarını kontrol ederek materyali daha iyi anlayabilirler.



Premium İçerik sitesine erişmek için, Companion Web sitesindeki veya [pearsonhighered.com/stallings](http://pearsonhighered.com/stallings) adresindeki Premium İçerik bağlantısına tıklayın ve kitabıñ önündeki kartta bulunan öğrenci erişim kodunu girin.

Son olarak, Bilgisayar Bilimleri Öğrenci Kaynak Sitesini şu adreste tutuyorum  
[WilliamStallings.com/StudentSupport.html](http://WilliamStallings.com/StudentSupport.html).

## PROJELER VE DIĞER ÖĞRENCİ ALIŞTIRMALARI

Birçok eğitmen için, bilgisayar organizasyonu ve mimarisi dersinin önemli bir bileşeni, öğrencinin metindeki kavramları pekiştirmek için uygulamalı deneyim kazandığı bir proje veya projeler dizisidir. Bu kitap, derse bir proje bileşeni dahil etmek için benzersiz derecede destek sağlar. Prentice Hall aracılığıyla temin edilebilen eğitim destek materyalleri sadece projelerin nasıl atanacağı ve yapılandırılacağı konusunda rehberlik etmekte kalmaz, aynı zamanda çeşitli proje türleri için bir dizi kullanım kılavuzu ve hepsi bu kitap için özel olarak yazılmış özel ödevler içerir. Eğitimmenler aşağıdaki alanlarda çalışma atayabilirler:

- **Etkileşimli simülasyon ödevleri:** Daha sonra açıklanacaktır.
- **Araştırma projeleri:** Öğrenciye internette belirli bir konuya araştırmasını ve bir rapor yazmasını söyleyen bir dizi araştırma ödevi.
- **Simülasyon projeleri:** IRC, iki simülasyon paketinin kullanımı için destek sağlamaktadır: SimpleScalár bilgisayar organizasyonu ve mimari tasarım konularını keşfetmek için kullanılabilir. SMPCache, simetrik çoklu işlemeciler için önbellek tasarım sorunlarını incelemek için güçlü bir eğitim aracı sağlar.
- **Assembly dili projeleri:** Basitleştirilmiş bir assembly dili olan CodeBlue kullanılmakta ve popüler Core Wars konseptine dayalı ödevler verilmektedir.
- **Okuma/rapor ödevleri:** Her bölüm için bir veya daha fazla olmak, öğrencinin okuması ve ardından kısa bir rapor yazması için verilebilecek literatürdeki makalelerin bir listesi.
- **Yazma ödevleri:** Materyalin öğrenilmesini kolaylaştmak için yazma ödevlerinin bir listesi.
- **Test bankası:** T/F, çoktan seçmeli ve boşluk doldurmaları sorular ve cevapları içerir.

Bu çeşitli projeler ve diğer öğrenci alıştırmaları, eğitmenin kitabı zengin ve çeşitli bir öğrenme deneyiminin bir bileşeni olarak kullanmasına ve eğitmen ve öğrencilerin özel ihtiyaçlarını karşılamak için bir kurs planı hazırlamasına olanak tanır. Ayrıntılar için bu kitaptaki Ek A'ya bakınız.

## INTERAKTİF SIMÜLASYONLAR

Bu baskındaki önemli bir özellik, etkileşimli simülasyonların dahil edilmesidir. Bu simülasyonlar, modern bir bilgisayar sisteminin karmaşık tasarım özelliklerini anlamak için güçlü bir araç sağlamaktadır. Bilgisayar organizasyonu ve mimari tasarımındaki temel fonksiyonları ve algoritmaları göstermek için toplam 20 interaktif simülasyon kullanılmıştır. Kitabın ilgili noktasında yer alan bir simge, ilgili interaktif simülasyonun öğrencilerin kullanımı için çevrimiçi olarak mevcut olduğunu göstermektedir. Animasyonlar kullanıcının başlangıç koşullarını belirlemesine olanak tanıdığı için

öğrenci ödevleri için temel oluşturmaktadır. Eğitmen eki, her animasyon için bir tane olmak üzere bir dizi ödev içerir. Her ödev, öğrencilere verilebilecek birkaç özel içermektedir.

Animasyonlara erişmek için, bu kitabın <http://williamstallings.com/ComputerOrganization> adresindeki Web sitesinde dönen küreye tıklayın.

## **TEŞEKKÜRLER**

Bu yeni baskı, zamanlarından ve uzmanlıklarından cömertçe yararlanan çok sayıda kişinin incelemesinden yararlanmıştır. Aşağıdaki profesörler ve eğitmenler makalenin tamamını ya da büyük bir bölümünü gözden geçirmiştir: Molisa Derk (Dickinson State University), Yaohang Li Old Dominion University), Dwayne Ockel (Regis University), Nelson Luiz Passos (Midwestern State University), Mohammad Abdus Salam (Southern University) ve Vladimir Zwass (Fair-leigh Dickinson University).

Ayrıca bir veya daha fazla bölüm için detaylı teknik incelemelerde bulunan birçok kişiye de teşekkür ederiz: Rekai Gonzalez Alberquilla, Allen Baum, Jalil Boukhobza, Dmitry Bufistov, Humberto Calderón, Jesus Carretero, Ashkan Eghbal, Peter Glaskowsky, Ram Huggahalli, Chris Jeshope, Athanasios Kakarountas, Isil Oz, Mitchell Poplingher, Roger Shepherd, Jigar Savla, Karl Stevens, Siri Uppalapati, Dr. Sriram Vajapeyam, Kugan Vivekanandarajah, Pooria M. Yaghini ve Peter Zeno,

Peter Zeno da GPGPU'larla ilgili 19. Bölümü katkıda bulunmuştur.

Appalachian Eyalet Üniversitesi'nden Profesör Cindy Norris, New Brunswick Üniversitesi'nden Profesör Bin Mu ve Alaska Üniversitesi'nden Profesör Kenrick Mock ev ödevi problemlerini nazikçe temin etmişlerdir.

Massachusetts Üniversitesi'nden Aswin Sreedhar interaktif simülasyon ödevlerini geliştirdi ve aynı zamanda test bankmasını yazdı.

İspanya'daki Extremadura Üniversitesi'nden Profesör Miguel Angel Vega Rodriguez, Profesör Dr. Juan Manuel Sánchez Pérez ve Profesör Dr. Juan Antonio Gómez Pulido, eğitmen kılavuzundaki SMPCache problemlerini hazırlamış ve SMPCache Kullanıcı Kılavuzu'nı kaleme almıştır.

Wisconsin Üniversitesi'nden Todd Bezenek ve Lehigh Üniversitesi'nden James Stine eğitmen SimpleScalar problemlerini hazırlamış ve Todd ayrıca SimpleScalar Kullanıcı Kılavuzu'nı yazmıştır.

Son olarak, kitabın yayınlanmasından sorumlu olan ve her zamanki gibi mükemmel bir iş çeken pek çok kişiye teşekkür etmek istiyorum. Bunlar arasında Pearson çalışanları, özellikle de editörüm Tracy Johnson, asistanı Kelsey Loanes, program yöneticisi Carole Snyder ve üretim müdürü Bob Engelhardt yer alıyor. Ayrıca Mahalatchoumy Saravanan'a ve Jouve India'daki produksiyon ekibine bir başka mükemmel ve hızlı iş için teşekkür ediyorum. Pearson'daki pazarlama ve satış personeline de teşekkürler, onların çabaları olmasaydı bu kitap önünüze .

# YAZAR HAKKINDA



**Dr. William Stallings** bilgisayar güvenliği, bilgisayar ağları ve bilgisayar mimarisi konularında 17 ders kitabı ve gözden geçirilmiş baskılarıyla birlikte 40'in üzerinde kitap yazmıştır. Bu alanda geçirdiği 30 yılı aşkın süre içinde çeşitli yüksek teknoloji firmalarında teknik katkı, teknik müdürlük ve yöneticilik görevlerinde bulunmuştur. Şu anda, kendisi

Müşterileri arasında bilgisayar ve ağ üreticileri ve müşterileri, yazılım geliştirme firmaları ve önde gelen devlet araştırma kurumları bulunan bağımsız bir danışmandır. Metin ve Akademik Yazarlar Derneği tarafından 13 kez yılın iyi bilgisayar bilimleri ders kitabı ödülüne layık görülmüştür.

ComputerScienceStudent.com adresinde Bilgisayar Bilimleri Öğrenci Kaynak Sitesi'ni oluşturmuş ve sürdürmektedir. Bu site, bilgisayar bilimleri öğrencilerinin (ve profesyonellerinin) genel ilgi alanına giren çeşitli alt konularda belgeler ve bağlantılar sunmaktadır. Makaleleri, Ağ Kategorisi Uzman Yazarı olduğu networking.answers.com'da düzenli olarak yayınlanmaktadır. Criptolojinin tüm yönlerine adanmış bilimsel bir dergi olan *Cryptologia*'nın yayin kurulu üyesidir.

Dr. Stallings, bilgisayar bilimleri alanında MIT'den doktora ve elektrik mühendisliği alanında Notre Dame'dan lisans derecesine sahiptir.

*Bu sayfa kasıtlı olarak boş bırakılmıştır*



# TEMEL KAVRAMLAR VE BİLGİSAYAR EVRİMİ

## 1.1 Organizasyon ve Mimari

## 1.2 Yapı ve İşlev

İşlev Yapısı

## 1.3 Bilgisayarların Kısa Tarihi

Birinci Nesil: Vakum Tüpleri İkinci Nesil:

Transistörler

Üçüncü Nesil: Entegre Devreler Sonraki Nesiller

## 1.4 Intel x86 Mimarisinin Evrimi

## 1.5 Gömülü Sistemler

Nesnelerin İnterneti Gömülü İşletim

Sistemleri

Uygulama İşlemcilerine Karşı Özel İşlemciler Mikroişlemcilere

Karşı Mikrodenetleyiciler Gömülü Sistemlere Karşı Derin

Gömülü Sistemler

## 1.6 ARM Mimarisi

ARM Evrimi

Komut Seti Mimarisi ARM

Ürünleri

## 1.7 Bulut Bilişim

Temel Kavramlar Bulut

Hizmetleri

## 1.8 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler

### ÖĞRENİM HEDEFLERİ

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- |r Dijital bir bilgisayarın genel işlevlerini ve yapısını açıklayabilecektir.
- |r İlk dijital bilgisayarlardan en yeni mikroişlemcilere kadar bilgisayar teknolojisinin evrimine genel bir bakış sunmak.
- |r x86 mimarisinin evrimine genel bir bakış sunun.
- |r Gömülü sistemleri tanımlayın ve çeşitli gömülü sistemlerin karşılaşması gereken bazı gereksinimleri ve kısıtlamaları listeleyin.

### 1.1 ORGANİZASYON VE ARŞİTEKTÜEL

Bilgisayarı tanımlarken, genellikle *bilgisayar mimarisi* ve *bilgisayar organizasyonu* arasında bir ayırım yapılır. Bu terimler için kesin tanımlar vermek zor olsa da, her birinin kapsadığı genel alanlar hakkında bir fikir birliği vardır. Örneğin, bakımınız [VRAN80], [SIEW82] ve [BELL78a]; ilginç bir alternatif görüş [REDD76]'da sunulmuştur.

**Bilgisayar mimarisi**, bir sistemin bir programcı tarafından görülebilen niteliklerini ya da başka bir deyişle, bir programın mantıksal yürütülmesi üzerinde doğrudan etkisi olan nitelikleri ifade eder. Genellikle bilgisayar mimarisi ile birbirinin yerine kullanılan bir terim **de komut seti mimarisidir (ISA)**. ISA, komut formatlarını, komut kodlarını, yazmaçları, komut ve veri belleğini; yürütülen komutların yazımcılar ve bellek üzerindeki etkisini ve komutların yürütülmesini kontrol etmek için bir algoritmayi tanımlar. **Bilgisayar organizasyonu**, mimari özellikleri gerçekleştiren operasyonel birimleri ve bunların ara bağlantılarını ifade eder. Mimari niteliklere örnek olarak komut seti, çeşitli veri türlerini (örneğin sayılar, karakterler) temsil etmek için kullanılan bit sayısı, G/C mekanizmaları ve belleği adresleme teknikleri verilebilir. Organizasyonel özellikler ise kontrol sinyalleri, bilgisayar ve çevre birimleri arasındaki arayüzler ve kullanılan bellek teknolojisi gibi programcı için şeffaf olan donanım detaylarını içerir.

Örneğin, bir bilgisayarın çarpmacı komutuna sahip olup olmayacağı mimari bir tasarım konusudur. Bu komutun özel bir çarpmacı birimi tarafından mı yoksa sistemin toplama birimini tekrar tekrar kullanan bir mekanizma tarafından mı uygulanacağı organizasyonel bir konudur. Organizasyonel karar, çarpmacı komutunun beklenen kullanım sıklığına, iki yaklaşımın göreceli hızına ve özel bir çarpmacı biriminin maliyetine ve fiziksel boyutuna dayanabilir.

Tarihsel olarak ve bugün de mimari ve organizasyon arasındaki ayırım önemli ayırım olmuştur. Birçok bilgisayar üreticisi, hepsi aynı mimariye sahip ancak organizasyonda farklılıklar olan bir bilgisayar modelleri ailesi sunmaktadır. Sonuç olarak, ailedeki farklı modeller farklı fiyat ve performans özelliklerine sahiptir. Ayrıca, belirli bir mimari uzun yıllara yayılabilir ve değişen teknolojiyle birlikte organizasyonu değişim bir dizi farklı bilgisayar modelini kapsayabilir. Bu iki olgunun da önemli bir örneği IBM System/370 mimarisidir. Bu mimari ilk olarak 1970 yılında tanıtılmış ve

bir dizi model içeriyordu. Mütevazı gereksinimleri olan müşteri daha ucuz, daha yavaş bir model satın alabilir ve talep artarsa, daha önce geliştirilmiş olan yazılımı terk etmek zorunda kalmadan daha pahali, daha hızlı bir modele yükseltebilir. Yıllar içinde IBM, müşteriye daha yüksek hız, daha düşük maliyet veya her ikisini birden sunarak eski modellerin yerini almak üzere gelişmiş teknolojiye sahip birçok yeni model tanıttı. Bu yeni modeller aynı mimariyi korudu, böylece müşterinin yazılım yatırımı korunmuş oldu. Dikkat çekici bir şekilde, System/370 mimarisi, birkaç geliştirme ile birlikte, IBM'in ana bilgisayar ürün serisinin mimarisi olarak günümüze kadar gelmiştir.

Mikrobilgisayarlar olarak adlandırılan bir bilgisayar sınıfında, mimari ve organizasyon arasındaki ilişki çok yakındır. Teknolojideki değişiklikler sadece organizasyonu etkilemeye kalmaz, aynı zamanda daha güçlü ve daha karmaşık mimarilerin ortayamasına neden olur. Genel olarak, bu küçük makineler için nesilden nesile uyumlu gereksinimi daha azdır. Dolayısıyla, organizasyonel ve mimari tasarım kararları arasında daha fazla etkileşim vardır. Bunun ilgi çekici bir örneği, Bölüm 15'te incelediğimiz azaltılmış komut seti bilgisayarıdır (RISC).

Bu kitap hem bilgisayar organizasyonunu hem de bilgisayar mimarisini incelemektedir. Belki de vurgu daha çok organizasyon tarafındadır. Ancak, bir bilgisayar organizasyonu belirli bir mimari spesifikasyonu uygulamak için tasarlanması gerektiğinden, organizasyonun kapsamlı bir şekilde ele alınması, mimarinin de ayrıntılı bir şekilde incelenmesini gerektirir.

## 1.2 YAPI VE İŞLEV

Bilgisayar karmaşık bir sistemdir; çağdaş bilgisayarlar milyonlarca temel elektronik bileşen içerir. O halde bu sistemler nasıl açık bir şekilde tanımlanabilir? Anahtar, bilgisayar dahil olmak üzere çoğu karmaşık sistemin hiyerarşik yapısını tanımaktır [SIMO96]. Hiyerarşik bir sistem, birbirile ilişkili alt sistemlerden oluşan bir kümedir ve bu alt sistemlerin her biri, en alt düzeydeki temel alt sisteme ulaşana kadar hiyerarşik bir yapıya sahiptir.

Karmaşık sistemlerin hiyerarşik yapısı, hem tasarımları hem de açıklamaları için esastır. Tasarımcının bir seferde sistemin yalnızca belirli bir seviyesiyle ilgilenmesi gereklidir. Her seviyede, sistem bir dizi bileşenden ve bunların karşılıklı ilişkilerinden oluşur. Her seviyedeki davranış, yalnızca bir sonraki alt seviyedeki sistemin basitleştirilmiş, soyutlanmış bir karakterizasyonuna bağlıdır. Tasarımcı her seviyede yapı ve işlevle ilgilinen:

- **Yapı:** Bileşenlerin birbirile ilişkili olma şekli.
- **İşlev:** Yapının bir parçası olarak her bir bileşenin çalışması.

Tanımlama açısından iki seçenekimiz vardır: en alttan başlayıp tam bir tanımlamaya kadar ilerlemek ya da en üstten başlayıp sistemi alt parçalarına ayırmak. Bir dizi alandan elde edilen kantlar, yukarıdan aşağıya yaklaşımın en net ve en etkili yaklaşım olduğunu göstermektedir [WEIN75].

Bu kitapta benimsenen yaklaşım bu bakış açısını takip etmektedir. Bilgisayar sistemi yukarıdan aşağıya doğru anlatılacaktır. Bir bilgisayarın ana bileşenleri ile başlayıp, bunların yapı ve işlevlerini tanımlayarak, sırayla

## 4 BÖLÜM 1 / TEMEL KAVRAMLAR VE BİLGİSAYARIN EVRİMİ

hiyerarşinin alt katmanları. Bu bölümün geri kalanında bu saldırının planına çok kısa bir genel bakış sunulmaktadır.

### Fonksiyon

Bir bilgisayarın hem yapısı hem de işleyışı özünde basittir. Genel anlamda, bir bilgisayarın gerçekleştirebileceği yalnızca dört temel işlev vardır:

- **Veri işleme:** Veriler çok çeşitli şekillerde olabilir ve işleme gereksinimleri de çok genişir. Bununla birlikte, veri işlemenin yalnızca birkaç temel yöntemi veya türü olduğunu göreceğiz.
- **Veri depolama:** Bilgisayar verileri yanında işliyor olsa bile (yani veriler gelir, işlenir ve sonuçlar hemen çıkar), en azından herhangi bir anda üzerinde çalışılan veri parçalarını geçici olarak saklaması gereklidir. Dolayısıyla, en azından kısa vadeli bir veri depolama işlevi vardır. Aynı derecede önemli olarak, bilgisayar uzun vadeli bir veri depolama işlevi de yerine getirir. Veri dosyaları daha sonra geri alınmak ve güncellenmek üzere bilgisayarda saklanır.
- **Veri hareketi:** Bilgisayarın çalışma ortamı, veri kaynağı ya da hedefi olarak hizmet veren aygıtlardan oluşur. Veriler bilgisayara doğrudan bağlı bir aygıtta alındığında veya bu aygıta gönderildiğinde, işlem *giriş-çıkış (I/O)* olarak bilinir ve aygit *çevre birimi* olarak adlandırılır. Veriler uzak bir cihaza veya uzak bir cihazdan daha uzun mesafelere taşındığında, bu işlem *veri iletişimini* olarak bilinir.
- **Kontrol:** Bilgisayar içinde, bir kontrol birimi bilgisayarın kaynaklarını yönetir ve talimatlara yanıt olarak işlevsel parçalarının performansını düzenler.

Yukarıdaki tartışma saçma bir şekilde genelleştirilmiş görünebilir. Bilgisayar yapısının en üst düzeyinde bile çeşitli işlevleri farklılaştmak kesinlikle mümkün değildir, ancak [SIEW82]'den alıntı yapmak gereklidir:

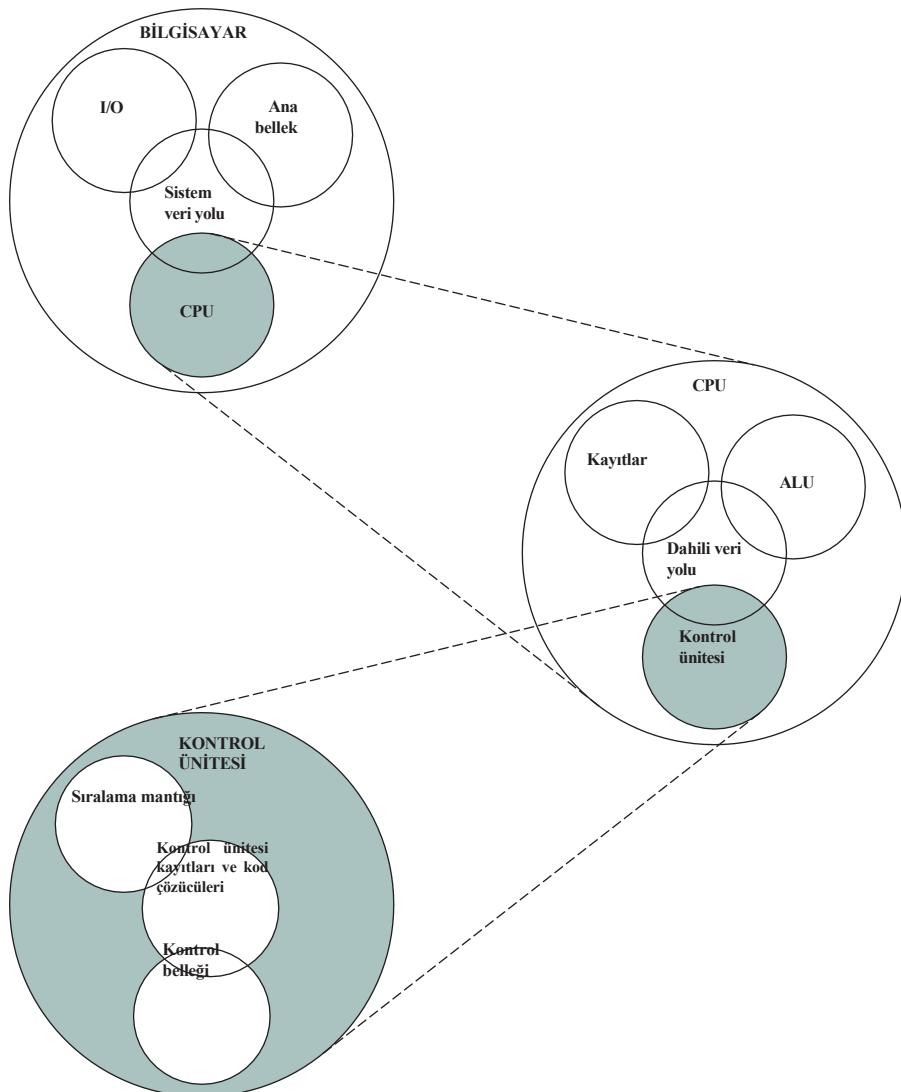
Gerçekleştirilecek iğleve uyacak. Geçikilde bilgisayar yapısının Gerçekleştirilmesi oldukça azdır. Bunun temelinde, tüm işlevsel uzmanlaşmanın tasarım sırasında değil programlama sırasında gerçekleştiği bilgisayarların genel amaçlı doğası yatmaktadır.

### Yapı

Şimdi genel olarak bir bilgisayarın iç yapısına bakacağız. Mikro programlanmış bir kontrol birimi kullanan tek bir işlemciye sahip geleneksel bir bilgisayarla başlıyoruz, ardından tipik bir çok çekirdekli yapıyı inceliyoruz.

**BASIT TEK İŞLEMCİLİ BİLGİSAYAR** Şekil 1.1, geleneksel tek işlemcili bir bilgisayarın iç yapısının hiyerarşik bir görünümünü summactadır. Dört ana yapısal bileşen vardır:

- **Merkezi işlem birimi (CPU):** Bilgisayarın çalışmasını kontrol eder ve veri işleme işlevlerini yerine getirir; genellikle basitçe *işlemci* olarak adlandırılır.
- **Ana bellek:** Verileri depolar.



**Şekil 1.1** Bilgisayar: Üst Düzey Yapı

- **I/O:** Bilgisayar ve dış ortamı arasında veri taşır.
- **Sistem ara bağlantıları:** CPU, ana bellek ve G/C arasında iletişim sağlayan bazı mekanizmalar. Sistem ara bağlantısının yaygın bir örneği, diğer tüm bileşenlerin bağlandığı bir dizi iletken kablodan oluşan bir **sistem veriyoludur**.

Yukarıda bahsedilen bileşenlerin her birinden bir veya daha fazla olabilir. Geleneksel olarak, sadece tek bir işlemci vardı. Son yıllarda, tek bir bilgisayarda birden fazla işlemci kullanımı artmaktadır. Çoklu işlemcilerle ilgili bazı tasarım sorunları ortaya çıkmakta ve metin ilerledikçe tartışılmaktadır; Beşinci Bölüm bu tür bilgisayarlara odaklanmaktadır.

## 6 BÖLÜM 1 / TEMEL KAVRAMLAR VE BİLGİSAYARIN EVRİMİ

Bu bileşenlerin her biri İkinci Bölümde ayrıntılı olarak incelenecaktır. Ancak, bizim amaçlarımız açısından en ilginç ve bazı açılarından en karmaşık bileşen CPU'dur. Başlıca yapısal bileşenleri aşağıdaki gibidir:

- **Kontrol ünitesi:** CPU'nun ve dolayısıyla bilgisayarın çalışmasını kontrol eder.
- **Aritmetik ve mantık birimi (ALU):** Bilgisayarın veri işleme işlevlerini yerine getirir.
- **Kayıtlar:** CPU'ya dahili depolama sağlar.
- **CPU ara bağlantıları:** Kontrol birimi, ALU ve kayıtlar arasında iletişim sağlayan bazı mekanizmalar.

Üçüncü Bölüm, paralel ve boru hattı organizasyon tekniklerinin kullanılmasıyla karmaşıklığın arttığını göreceğimiz bu bileşenleri . Son olarak, kontrol biriminin uygulanmasına yönelik çeşitli yaklaşımlar vardır; yaygın yaklaşımlardan biri *mikro programlı* uygulamadır. Temelde, mikro programlanmış bir kontrol ünitesi, kontrol ünitesinin işlevsellliğini tanımlayan mikro talimatları yürüterek çalışır. Bu yaklaşımla, kontrol ünitesinin yapısı Şekil 1.1'de olduğu gibi gösterilebilir. Bu yapı Dördüncü Bölümde incelenmiştir.

**ÇOK ÇEKİRDEKLİ İŞLEMCİ YAPISI** Daha önce de belirtildiği gibi, çağdaş bilgisayarlar genellikle birden fazla işlemciye sahiptir. Bu işlemcilerin hepsi tek bir yonga üzerinde yer alındığında, *çok çekirdekli bilgisayar* terimi kullanılır ve her bir işlem birimi (bir kontrol birimi, ALU, kayıtlar ve belki de önbellekten oluşan) bir *çekirdek* olarak adlandırılır. Terminolojiyi açılkıla kavuşturmak için bu metinde aşağıdaki tanımlar kullanılacaktır.

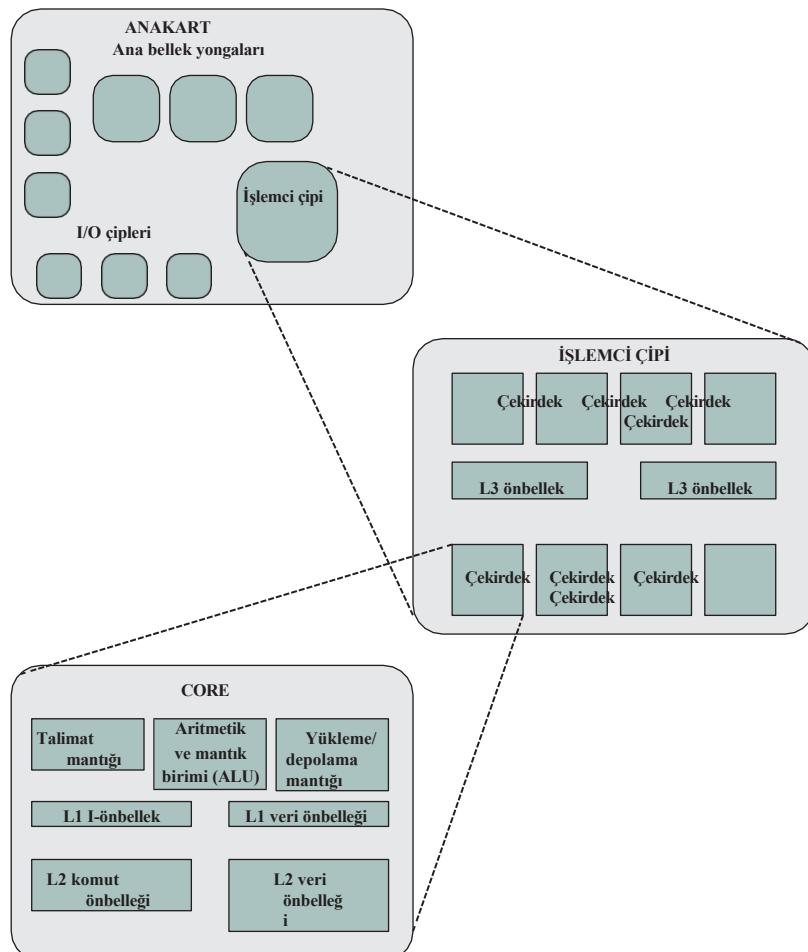
- **Merkezi işlem birimi (CPU):** Bir bilgisayarın talimatları alan ve yürütten kısmı. Bir ALU, bir kontrol birimi ve kayıtlardan oluşur. Tek bir işlem birimine sahip bir sistemde, genellikle basitçe *işlemci* olarak adlandırılır.
- **Çekirdek:** Bir işlemci çipi üzerindeki bireysel bir işlem birimi. Bir çekirdek, işlevsellik açısından tek CPU'lu bir sistemdeki CPU'ya eşdeğer olabilir. Vektör ve matris işlemleri için optimize edilmiş bir birim gibi diğer özel işlem birimleri de çekirdek olarak adlandırılır.
- **İşlemci:** Bir veya daha fazla çekirdek içeren fiziksel bir silikon parçası. İşlemci, talimatları yorumlayan ve yürütten bilgisayar bileşenidir. Bir işlemci birden fazla çekirdek içeriyorsa, *çok çekirdekli işlemci* olarak adlandırılır.

Yaklaşık on yıllık bir tartışmanın ardından, bu kullanım konusunda sektörde geniş bir fikir birliği oluşmuştur. Çağdaş bilgisayarların öne çıkan bir diğer özelliği de işlemci ile ana arasında *ön bellek* adı verilen çoklu bellek katmanlarının kullanılmasıdır. Bölüm 4 ön bellek konusuna ayrılmıştır. Bu bölümdeki amaçlarımız doğrultusunda, ön belleğin ana bellekten daha küçük ve daha hızlı olduğunu ve ana bellekten yakın gelecekte kullanılması muhtemel verileri ön belleğe yerleştirerek bellek erişimini hızlandırmak için kullanıldığını belirtmekle yetineceğiz. Çekirdeğe en yakın seviye 1 (L1) ve giderek uzaklaşan ek seviyeler (L2, L3.) olmak üzere birden fazla önbellek seviyesi kullanılarak daha yüksek bir performans artışı elde edilebilir. Bu durumda şemasında,  $n$  seviyesi  $n+1$  seviyesinden daha küçük ve daha hızlıdır.

## 1.2 / YAPI VE 7

Şekil 1.2 tipik bir çok çekirdekli bilgisayarın temel bileşenlerinin basitleştirilmiş bir görünümüdür. Akıllı telefonlar ve tabletlerdeki gömülü bilgisayarların yanı sıra kişisel bilgisayarlar, dizüstü bilgisayarlar ve iş istasyonları da dahil olmak üzere çoğu bilgisayar bir anakart üzerine yerleştirilmiştir. Bu düzenlemeyi açıklamadan önce bazı terimleri tanımlamamız gereklidir. **Baskılı devre kartı (PCB)**, çipleri ve diğer elektronik bileşenleri tutan ve birbirine bağlayan sert, düz bir karttır. Kart, tipik iki ila on katmandan oluşur ve bileşenleri karta kazınmış bakır yollar aracılığıyla birbirine bağlar. Bir bilgisayardaki ana baskılı devre kartına sistem kartı veya **anakart** adı, ana karttaki Yuvalara takılan daha küçük olanlara genişletme kartları denir.

Anakart üzerindeki en belirgin unsurlar çiplerdir. Bir **çip**, üzerinde elektronik devrelerin ve mantık kapılarının üretildiği tek bir yarı iletken malzeme parçasıdır, tipik olarak silikonudur. Ortaya çıkan ürün **entegre devre** olarak adlandırılır.



Şekil 1.2 Çok Çekirdekli Bir Bilgisayarın Başlıca Unsurlarının Basitleştirilmiş Görünümü

## 8 BÖLÜM 1 / TEMEL KAVRAMLAR VE BİLGİSAYARIN EVRİMİ

Anakart, çok çekirdekli işlemci olarak bilinen ve tipik olarak birden fazla ayrı çekirdek içeren işlemci yongası için bir yuva veya soket içerir. Ayrıca bellek yongaları, I/O denetleyici yongaları ve diğer önemli bilgisayar bileşenleri için de yuvalar vardır. Masaüstü bilgisayarlar için genişleme yuvaları, genişleme kartlarına daha fazla bileşenin eklenmesini sağlar. Bu nedenle, modern bir anakart yalnızca birkaç ayrı yonga bileşenini birbirine bağlar ve her yonga birkaç bin ila yüz milyonlarca transistör içerir.

Şekil 1.2 sekiz çekirdek ve bir L3 önbellek içeren bir işlemci yongasını göstermektedir. Çekirdekler ile önbellek arasındaki ve çekirdekler ile anakart üzerindeki harici devre arasındaki işlemleri kontrol etmek için gereken mantık gösterilmemiştir. Şekilde L3 önbelleğin çip yüzeyinin iki farklı bölümünü kapladığı görülmektedir. Ancak, tipik olarak, tüm çekirdekler yukarıda bahsedilen kontrol devreleri aracılığıyla L3 önbelleğin tamamına erişebilir. Şekil 1.2'de gösterilen işlemci yongası belirli bir ürünü temsil etmemekte, ancak bu tür yongaların nasıl dair genel bir fikir vermektedir.

Daha sonra, işlemci yongasının bir bölümünü kaplayan tek bir çekirdeğin yapısını yakındanıriz. Genel anlamda, bir çekirdeğin işlevsel unsurları şunlardır:

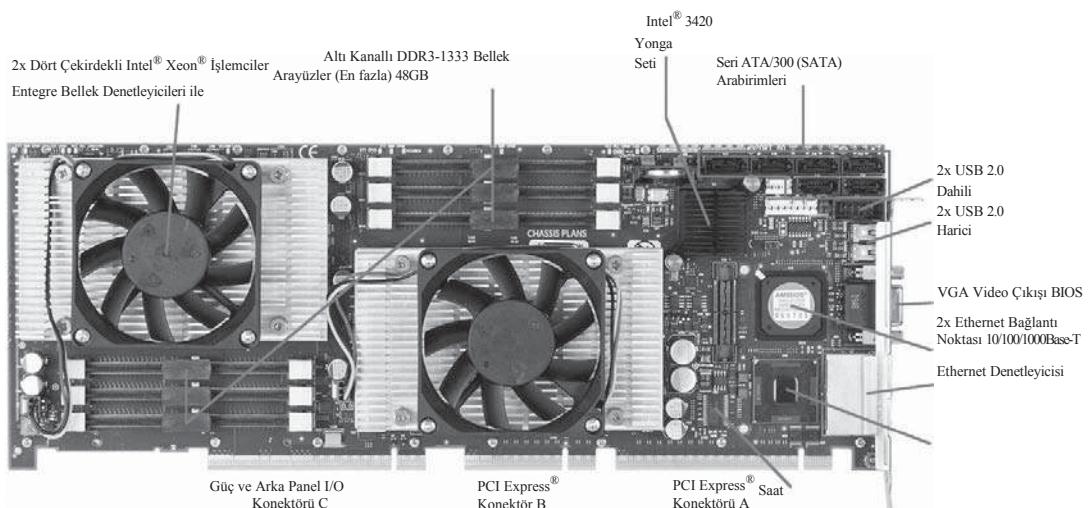
- **Talimat mantığı:** Bu, talimatların getirilmesiyle ilgili görevleri ve talimat işlemini ve herhangi bir işlenenin bellek konumlarını belirlemek için her bir talimin kodunu çözmemi içerir.
- **Aritmetik ve mantık birimi (ALU):** Bir komut tarafından belirtilen işlemi gerçekleştirir.
- **Yükleme/depolama mantığı:** Önbellek aracılığıyla ana belleğe ve ana bellekten veri aktarımını yönetir.

Cekirdek ayrıca, talimatların ana belleğe ve ana bellekten aktarılması için kullanılan bir talimat önbelleği (I-cache) ile operandların ve sonuçların aktarılması için bir L1 veri önbelleği arasında bölünmüş bir L1 önbelleği içerir. Tipik olarak, günümüzün işlemci yongaları çekirdeğin bir parçası olarak bir L2 önbelleği de içerir. Coğu durumda, bu önbellek de komut ve veri önbellekleri arasında bölünmüştür, ancak birleşik, tek bir L2 önbellek de kullanılmaktadır.

Cekirdeğin düzenebine ilişkin bu gösterimin yalnızca iç çekirdek yapısı hakkında genel bir fikir vermemi amaçladığını unutmayın. Belirli bir üründe, işlevsel unsurlar Şekil 1.2'de gösterilen üç ayrı unsur olarak düzenlenmeyebilir, özellikle de bu işlevlerin bir kısmı veya tamamı mikro gramlı bir kontrol ünitesinin parçası olarak uygulanıyor.

**ÖRNEKLER** Bilgisayarların hiyerarşik yapısını gösteren bazı gerçek dünya örneklerine bakmak öğretici olacaktır. Şekil 1.3, iki Intel Quad-Core Xeon işlemci yongası üzerine inşa edilmiş bir bilgisayarın anakartının fotoğrafıdır. Fotoğrafta etiketlenen öğelerin çoğu bu kitapta daha sonra ele alınacaktır. Burada, işlemci soketlerine ek olarak en önemlilerinden bahsedeceğiz:

- Üst düzey bir ekran bağırstırıcısı ve ek çevre birimleri için PCI-Express yuvaları (Bölüm 3.6'da PCIe açıklanmaktadır).
- Ağ bağlantıları için Ethernet denetleyicisi ve Ethernet bağlantı noktaları.
- Çevresel cihazlar için USB soketleri.

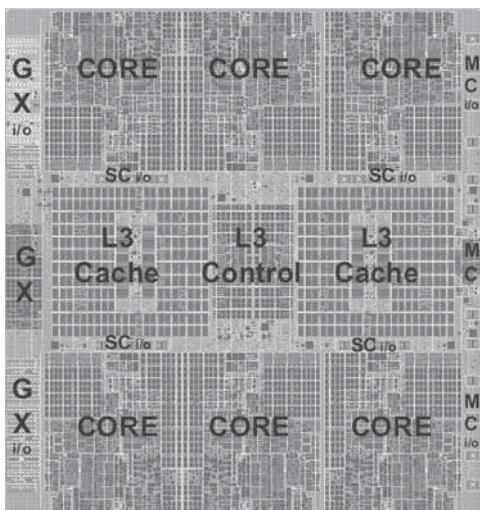


- Disk belleğine bağlantı için Seri ATA (SATA) soketleri (Bölüm 7.7'de Ethernet, USB ve SATA anlatılmaktadır).
- DDR (çift veri hızı) ana bellek yongaları için arayüzler (Bölüm 5.3'te DDR anlatılmaktadır).
- Intel 3420 yonga seti, çevresel aygıtlar ve ana bellek arasında doğrudan bellek erişim işlemleri için bir G/C denetleyicisidir (Bölüm 7.5'te DDR anlatılmaktadır).

Yukarıdan aşağıya stratejimizi izleyerek, Şekil 1.1 ve 1.2'de gösterildiği gibi, şimdilik yakından inceleyelim. Şekil 1.1'ye göre, Intel işlemci çipi yerine bir IBM işlemcisi kullanılmıştır. Bu çip, 2,75 milyar transistör bulunan bir silikon alandır. Üst üste bindirilmiş etiketler çipin silikon alanının nasıl tesis edildiğini göstermektedir. Bu çipin altı çekirdeğe ya da işlemciye sahip olduğunu görüyoruz. Buna ek olarak, altı işlemcinin tümü tarafından paylaşılan L3 önbellek etiketli iki büyük alan vardır. L3 kontrol mantığı L3 önbellek ile çekirdekler arasındaki ve L3 önbellek ile dış ortam arasındaki trafiği kontrol eder. Ayrıca, çekirdekler ve L3 önbellek arasında depolama yaşı kontrol (SC) mantığı vardır. Bellek denetleyicisi (MC) işlevi, yonganın haricindeki belleğe erişimi kontrol eder. GX I/O veri yolu, I/O'ya erişen kanal adaptörlerine giden arayüzü kontrol eder.

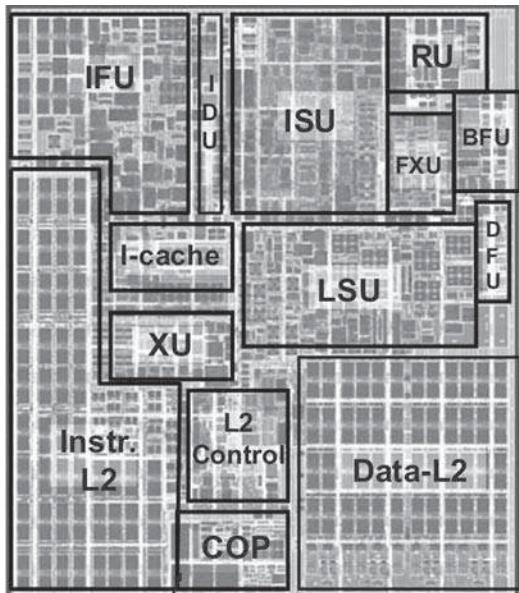
Bir seviye daha derine inerek, Şekil 1.5'teki fotoğrafta gösterildiği gibi tek bir çekirdeğin iç yapısını inceleyebiliriz. Bunun tek işlemcili bir çipi oluşturan silikon yüzey alanının bir kısmı olduğunu unutmuyın. Bu çekirdek alanı içindeki ana alt alanlar şunlardır:

- **ISU (komut sırası birimi):** Süperskalar mimari olarak adlandırılan mimaride talimatların yürütülme sırasını belirler (Bölüm 16).
- **IFU (talimat getirme birimi):** Talimat getirme mantığı.



**Şekil 1.4** zEnterprise EC12 İşlemci Birimi (PU) yonga divigramı

*Kaynak: IBM zEnterprise EC12 Teknik Kılavuzu, Aralık 2013, SG24-8049-01. IBM. İzleme Yeniden Basılmıştır*



**Şekil 1.5** zEnterprise EC12 Core düzeni *Kaynak:* IBM zEnterprise EC12 Teknik Kılavuzu, Aralık 2013, SG24-8049-01. IBM. İznine Yeniden Basılmıştır.

- **IDU (komut kod çözme birimi):** IDU, IFU tamponlarından beslenir ve tüm z/Mimari işlem kodlarının ayrıştırılması ve kodunun çözülmesinden sorumludur.
  - **LSU (yük depolama birimi):** LSU, 96-kB L1 veri önbelleğini<sup>1</sup> içerir ve L2 veri önbelleği ile işlevsel yürütme birimleri arasındaki veri trafigini yönetir. z/Mimarisinde tanımlandığı gibi tüm uzunluk, mod ve biçimlerdeki her tür işlenen erişimini işlemekten sorumludur.
  - **XU (çeviri birimi):** Bu birim, komutlardaki mantıksal adresleri ana bellekteki fiziksel adreslere çevirir. XU ayrıca bellek erişimini hızlandırmak için kullanılan bir çeviri ara belleği (TLB) içerir. TLB'ler Bölüm 8'de ele alınmaktadır.
  - **FXU (sabit nokta birimi):** FXU sabit noktalı aritmetik işlemleri yürütür.
  - **BFU (ikili kayan nokta birimi):** BFU, tüm ikili ve onaltılık kayan nokta işlemlerinin yanı sıra sabit nokta çarpma işlemlerini de gerçekleştirir.
  - **DFU (ondalık kayan nokta birimi):** DFU, ondalık basamak olarak saklanan sayılar üzerinde hem sabit nokta de kayan nokta işlemlerini gerçekleştirir.
  - **RU (kurtarma birimi):** RU, tüm kayıtları içeren sistemin tüm durumunun bir kopyasını tutar, donanım hatası sinyallerini toplar ve donanım kurtarma eylemlerini yönetir.

<sup>1</sup>kB= kilobyte= 2048 byte. Sayısal ön ekler ComputerScienceStudent.com adresinde "Diğer Faydalı" sekmesi altındaki bir belgede açıklanmıştır.

- **COP (özel yardımcı işlemci):** COP, her bir çekirdek için veri sıkıştırma ve şifreleme işlevlerinden sorumludur.
- **I-önbellek:** Bu 64 kB'lık bir L1 komut önbellegidir ve IFU'nun talimatları ihtiyaç duyulmadan önce önceden almasını sağlar.
- **L2 kontrolü:** Bu, iki L2 önbellek üzerinden trafiği yöneten kontrol mantığıdır.
- **Data-L2:** Komutlar dışındaki tüm bellek trafiği için 1 MB L2 veri önbellegi.
- **Instr-L2:** 1 MB L2 komut önbellegi.

Kitapta ilerledikçe, bu bölümde tanıtılan kavramlar daha açık hale gelecektir.

## 1.3 bilgisayarların kısa bir tarihi<sup>2</sup>

Bu bölümde, bilgisayarların gelişim tarihine kısa bir genel bakış sunuyoruz. Bu tarihçe kendi içinde ilginçtir, ancak daha da önemlisi, kitap boyunca ele aldığımız birçok önemli kavrama temel bir giriş sağlar.

### İlk Nesil: Vakum Tüpleri

İlk nesil bilgisayarlar dijital mantık elemanları ve bellek için vakum tüpleri kullanmıştır. Vakum tüpleri kullanılarak bir dizi araştırma ve ardından ticari bilgisayar üretilmiştir. Bizim amaçlarımız doğrultusunda, IAS bilgisayarı olarak bilinen belki de en ünlü birinci nesil bilgisayarı incelemek öğretici olacaktır.

İlk olarak IAS bilgisayarında uygulanan temel bir tasarım yaklaşımı *saklı program konsepti* olarak bilinir. Bu fikir genellikle matematikçi John von Neumann'a atfedilir. Alan Turing de bu fikri yaklaşık aynı zamanlarda geliştirmiştir. Fikrin ilk yayımı von Neumann'ın 1945 yılında yeni bir bilgisayar olan EDVAC (Elektronik Ayrık Değişkenli Bilgisayar) için yaptığı bir öneride yer almıştır.<sup>3</sup>

1946 yılında von Neumann ve meslektaşları Princeton İleri Araştırmalar Enstitüsü'nde IAS bilgisayarı olarak adlandırılan yeni bir depolanmış program bilgisayارının tasarımına başladılar. IAS bilgisayarı, 1952 yılına kadar tamamlanmamış olmasına rağmen, sonraki tüm genel amaçlı bilgisayarların prototipidir.<sup>4</sup>

Şekil 1.6'da IAS bilgisayarlarının yapısı gösterilmektedir (Şekil 1.1 ile karşılaştırınız). Sunlardan oluşur

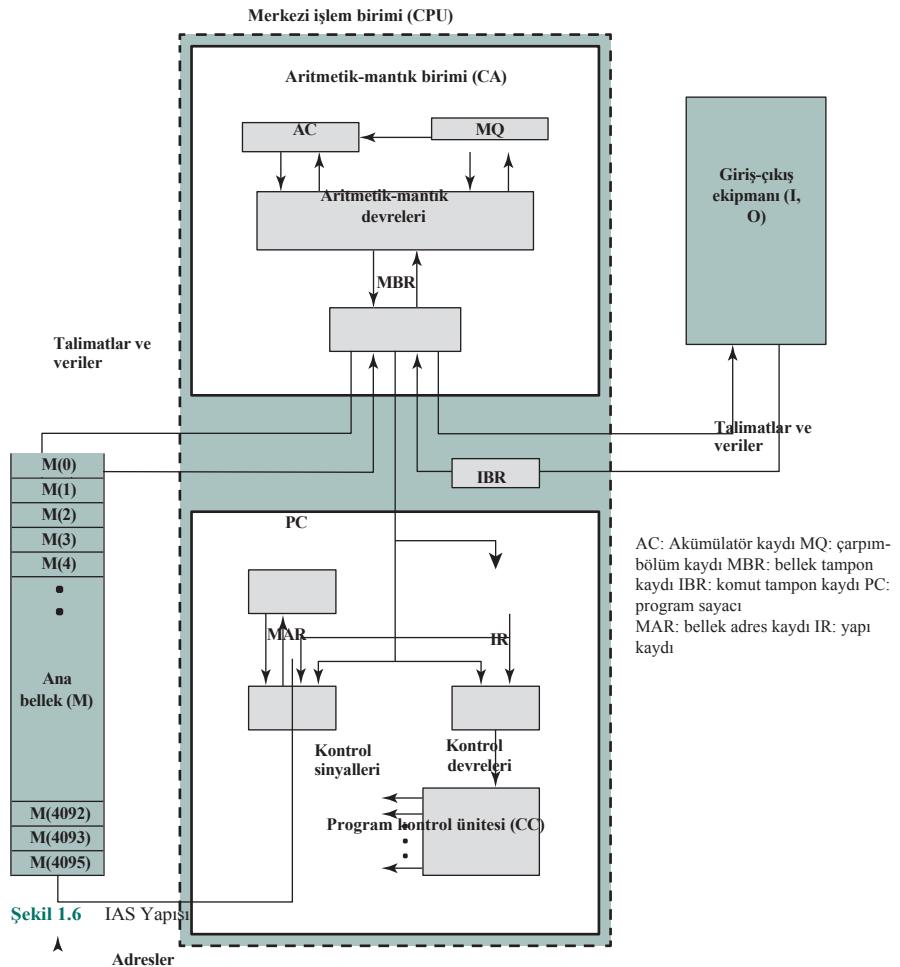
- Hem verileri hem de talimatları depolayan bir **ana bellek**<sup>5</sup>
- İki veriler üzerinde çalışabilen bir **aritmetik ve mantık birimi (ALU)**

<sup>2</sup>Bu kitabın Yardımcı Web sitesi ([WilliamStallings.com/ComputerOrganization](http://WilliamStallings.com/ComputerOrganization)), bu bölümde tartışılan cihazların ve bileşenlerin çoğunu fotoğraflarını sağlayan sitelere çeşitli bağlantılar içerir.

<sup>3</sup>EDVAC hakkındaki 1945 raporu [box.com/COA10e](http://box.com/COA10e) adresinden ulaşılabilir.

<sup>4</sup> 1954 tarihli bir rapor [GOLD54] uygulanan IAS makinesini tanımlamakta ve nihai komut setini listelemektedir. [Box.com/COA10e](http://Box.com/COA10e) adresinde mevcuttur.

<sup>5</sup>Bu kitapta, aksi belirtilmemiş olmak üzere, *talimat* terimi, Ada veya C++ yüksek seviyeli bir dilde, çalıştırılmadan önce bir dizi makine talimatına derlenmesi gereken bir ifadenin aksine, işlemci tarafından doğrudan yorumlanan ve çalıştırılan bir makine talimatını ifade eder.



- Bellekteki talimatları yorumlayan ve bunların yürütülmesini sağlayan bir **kontrol birimi**
- Kontrol ünitesi tarafından çalıştırılan **giriş-çıkış (I/O) ekipmanı**

Bu yapı, von Neumann'in bu noktada kısmen alıntılmaya değer olan önceki önerisinde ana hatlarıyla belirtilmiştir [VONN45]:

**2.2 Birincisi:** Cihaz öncelikle bir bilgisayar olduğundan, aritmetiğin temel işlemlerini en sık gerçeklestirmesi gerekecektir. Bunlar toplama, çıkarma, çarpmaya ve bölme işlemleridir. Bu nedenle sadece bu işlemler için özelleşmiş organlar içermesi mantıklıdır.

Bununla birlikte, bu prensip muhtemelen sağlam olsa da, gerçekleştirmelisinin yakından incelenmesi gerektiği göz önünde bulundurulmalıdır. Her halükarda cihazın merkezi bir *aritmetik* bölümünün bulunması gerekecektir ve bu da ilk özel oluşturmaktadır: *CA*.

**2.3 İlkincisi:** Cihazın mantıksal kontrolü, yani iğlemelerinin uygun şekilde sıralanması, en verimli şekilde merkezi bir kontrol organı tarafından gerçekleştirilebilir. Eğer cihaz *elastik* olacaksa, yani mümkün olduğunda *her amaca* uygun olacaksa, o zaman belirli bir sorun için verilen ve bu sorunu tanımlayan özel talimatlar ile bu talimatların -ne olursa olsun- yerine getirilmesini genel kontrol organları arasında bir ayırım yapılmalıdır. Birincisi bir şekilde saklanmalıdır; ikincisi ise cihazın belirli çalışma parçaları tarafından temsil edilir. *Merkezi kontrol* ile yalnızca bu ikinci işlevi kastediyoruz ve bunu yerine getiren organlar *ikinci özel kısmı* oluşturur: *CC*.

**2.4 Üçüncüsü:** Uzun ve karmaşık işlem dizileri (özellikle hesaplamalar) gerçekleştirecek herhangi bir cihazın hatırı sayılır bir belleğe sahip olması gereklidir...

Karmaşık bir problemi yöneten talimatlar, özellikle de kod önemli ise (ki çoğu düzenlemeye öyledir) önemli bir materyal oluşturabilir. Bu malzeme hatırlanmalıdır.

Her halükarda, toplam bellek cihazın üçüncü özel bölümünü oluşturmaktadır: *M*.

**2.6 Üç özel bölüm CA, CC (birlikte C) ve M** kor- insan sinir sistemindeki ilişkisel nöronlara yanıt verir. Geriye *duyusal* ya da *afferent* ve *motor* ya da *efferent* nöronların eşdeğerlerini tartışmak kalmaktadır. Bunlar cihazın *giriş* ve *çıkış* organlarıdır.

Cihaz, bu türden belirli bir ortamla girdi ve çıktı (duyusal ve motor) temasını sürdürme yeteneğine sahip olmalıdır. Bu ortam, cihazın *dış kayıt* ortamı adlandırılacaktır: *R*.

**2.7 Dördüncüsü:** Aygit, R'den C ve M'ye bilgi aktaracak organlara sahip olmalıdır. Bu organlar aygitin *girişini*, yani *dördüncü özel parçasını* oluşturur: R'den (I tarafından) M'ye tüm transferleri yapmanın ve asla doğrudan C'den yapmamanın en iyisi olduğu görülecektir.

**2.8 Beşincisi:** Aygit, C ve M'den R'ye aktarım yapacak organlara sahip olmalıdır. Bu organlar onun *çiktisini*, yani *beinci özel parçasını* oluşturur: M'den (O tarafından) R'ye tüm transferleri yapmanın ve asla doğrudan C'den yapmamanın yine en iyisi olduğu görülecektir.

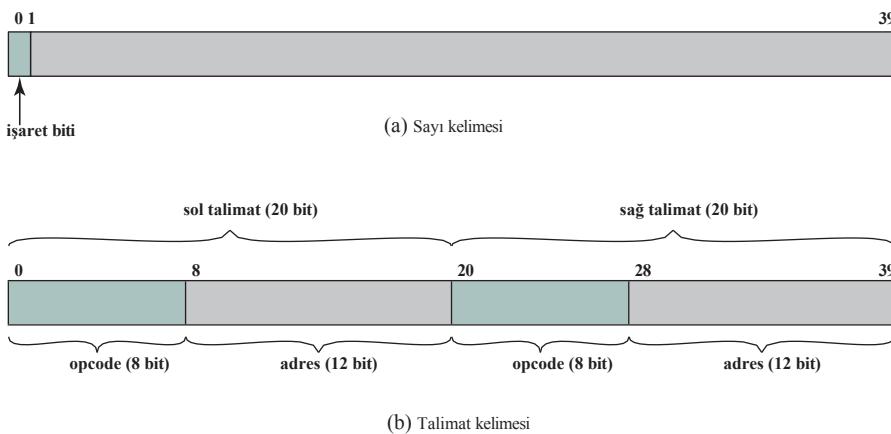
Nadir istisnalar dışında, günümüzün tüm bilgisayarıları aynı genel yapıya ve işlev sahiptir ve bu nedenle von Neumann makineleri olarak anılırlar. Bu nedenle, bu noktada IAS bilgisayarının işleyişini kısaca açıklamak faydalı olacaktır [BURK46, GOLD54]. HAYE98'i takiben, von Neumann terminolojisi ve notasyonu

modern kullanıma daha yakın olması için aşağıda değiştirilmiştir; bu tartışmaya eşlik eden örnekler bu son metne dayanmaktadır.

IAS'nin belleği, her biri 40 ikili basamaktan (bit) oluşan ve *sözcük* olarak adlandırılan 4.096 depolama konumundan oluşur.<sup>6</sup> Hem veriler hem de talimatlar burada saklanır. Sayılar ikili biçimde temsil edilir ve her talimat bir ikili koddur. Şekil 1.7 bu formatları göstermektedir. Her sayı bir işaret biti ve 39 bitlik bir değer ile temsil edilir. Bir kelime alternatif olarak iki adet 20 bitlik talimat içerebilir, her bir talimat gerçekleştirilecek işlemi belirten 8 bitlik bir işlem kodu (opcode) ve bellekteki kelimelerden birini (0'dan 999'a kadar numaralandırılmış) belirten 12 bitlik bir adresen oluşur.

Kontrol ünitesi IAS'yi bellekten talimatlar alarak ve bunları teker teker çalıştırarak çalıştırır. Bu işlemleri Şekil 1.6'yı referans olarak açıklayacağız. Bu şekil, hem kontrol biriminin hem de ALU'nun aşağıdaki gibi tanımlanan *yazmaca* adı verilen depolama konumları içerdigini ortaya koymaktadır:

- **Bellek tampon kaydı (MBR):** Bellekte saklanacak veya G/C birimine gönderilecek bir sözcük içerir ya da bellekten veya G/C biriminden bir sözcük almak için kullanılır.
- **Bellek adres kaydı (MAR):** MBR'den yazılacak veya MBR'ye okunacak kelimenin bellekteki adresini belirtir.
- **Talimat kaydı (IR):** Yürüttülmekte olan 8 bitlik işlem kodu komutunu içerir.
- **Talimat tampon kaydı (IBR):** Bellekteki bir kelimenin sağ komutu geçici olarak tutmak için kullanılır.
- **Program sayacı (PC):** Bellekten getirilecek bir sonraki komut çiftinin adresini içerir.
- **Akümülatör (AC) ve çarpan bölümü (MQ):** ALU işlemlerinin işlenenlerini ve sonuçlarını geçici olarak tutmak için kullanılır. Örneğin, sonuç



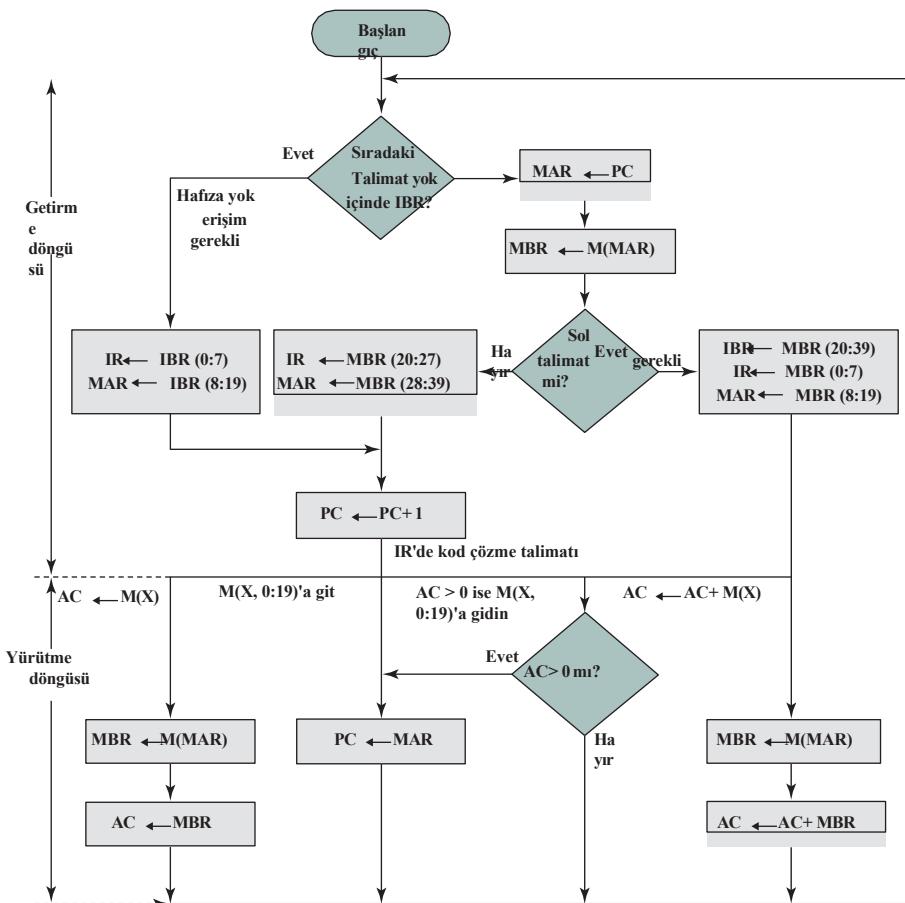
**Şekil 1.7** IAS Bellek Formatları

<sup>6</sup> *Kelime* teriminin evrensel bir tanımı yoktur. Genel olarak kelime, belirli bir bilgisayarda bilginin saklanabileceği, iletilebileceği üzerinde işlem yapılabileceği normal birim olan bayt veya bitlerden oluşan sıralı bir kümedir. Tipik olarak, bir işlemci sabit uzunlukta bir komut setine sahipse, komut uzunluğu kelime uzunluğuna eşittir.

İki 40 bitlik çarpımı 80 bitlik bir sayıdır; en anlamlı 40 bit AC'de ve en az anlamlı olan MQ'da saklanır.

IAS, Şekil 1.8'de gösterildiği gibi bir komut döngüsünü tekrar tekrar gerçekleştirecek çalışır. Her komut döngüsü iki alt döngüden oluşur. *Getirme döngüsü* sırasında, bir sonraki komutun işlem kodu IR'ye yüklenir ve adres kısmı MAR'a yüklenir. Bu komut IBR'den alınabilir veya MBR'ye ve ardından IBR, IR ve MAR'a bir kelime yüklenerek bellekten elde edilebilir.

Neden dolaylama? Bu işlemler elektronik devre tarafından kontrol edilir ve veri yollarının kullanılmasıyla sonuçlanır. Elektroniki basitleştirmek için, bir okuma veya yazma için bellekteki adresi belirtmek için kullanılan tek bir yazmaç ve kaynak veya hedef için kullanılan tek bir yazmaç vardır.



$M(X)=$  adresi X olan bellek konumunun içeriği ( $i:j$ ) =  $i$ 'den  $j$ 'ye kadar olan bitler

Şekil 1.8 IAS Operasyonunun Kısmi Akış Şeması

## 16 BÖLÜM 1 / TEMEL KAVRAMLAR VE BİLGİSAYARIN EVRİMİ

İşlem kodu IR'ye girdiğinde, *yürüitme döngüsü* gerçekleştirilir. Kontrol devresi işlem kodunu yorumlar ve verilerin taşınmasını veya ALU tarafından bir işlem yapılmasını sağlamak için uygun kontrol sinyallerini göndererek komutu yürütür.

IAS bilgisayarında Tablo 1.1'de listelenen toplam 21 talimat bulunmaktadır.

Bunlar aşağıdaki gibi grupperlendirilebilir:

- **Veri aktarımı:** Verileri bellek ve ALU kayıtları arasında veya iki ALU kaydı arasında taşıyın.
- **Koşulsuz dallanma:** Normalde kontrol ünitesi komutları bellekten sırayla yürütür. Bu sırada, tekrarlayan işlemleri kolaylaştırın bir dallanma talimatı ile değiştirebilir.

**Tablo 1.1** IAS Komut Seti

Talimat Türü	Sembolik Temsil		Açıklama
	Opcode		
Veri aktarımı	00001010	MQ YÜKLE	MQ kaydının içeriğini AC akümülatörüne aktarın
	00001001	LOAD MQ,M(X)	X bellek konumunun içeriğini MQ'ya aktarın
	00100001	STOR M(X)	Akümülatörün içeriğini X bellek konumuna aktarın
	00000001	YÜK M(X)	M(X)'i akümülatöre aktarın
	00000010	YÜK -M(X)	Akümülatöre -M(X) aktarın
	00000011	YÜK  M(X)	M(X)'in mutlak değerini akümülatöre aktarın
Koşulsuz şube	00001101	JUMP M(X,0:19)	M(X)'in sol yarısından bir sonraki talimatı alın
	00001110	ATLAMA M(X,20:39)	M(X)'in sağ yarısından bir sonraki talimatı alın
Koşullu şube	00001111	ATLAMA+ M(X,0:19)	Akümülatördeki sayı negatif değilse, bir sonraki M(X)'in sol yarısından gelen talimat
	00010000	ATLAMA+ M(X,20:39)	Akümülatördeki sayı negatif değilse, bir sonraki M(X)'in sağ yarısından gelen talimat
Aritmetik	00000101	EKLE M(X)	AC'ye M(X) ekleyin; sonucu AC'ye koyun
	00000111	ADD  M(X)	AC'ye  M(X)  ekleyin; sonucu AC'ye koyun
	00000110	SUB M(X)	AC'den M(X)'i çıkarın; sonucu AC'ye koyun
	00001000	SUB  M(X)	AC'den  M(X) 'i çıkarın; kalanı AC'ye koyun
	00001011	MUL M(X)	M(X) ile MQ'yı çarpın; sonucun en anlamlı bitlerini AC'ye, en az anlamlı bitlerini MQ'ya koyun
	00001100	DIV M(X)	AC'yi M(X)'e bölün; bölümü MQ'ya koyun ve AC'de kalan
	00010100	LSH	Akümülatörü 2 ile çarpin; yani, bir bit konumu sola kaydırın
	00010101	RSH	Akümülatörü 2'ye böl; yani bir konum sağa kaydır
Adres değiştir	00010010	STOR M(X,8:19)	M(X)'teki sol adres alanını en sağdaki 12 bit ile değiştirin AC'nin
	00010011	STOR M(X,28:39)	M(X)'teki sağ adres alanını en sağdaki 12 bit ile değiştirin AC'nin

- **Koşullu dallanma:** Dal bir koşula bağlı hale getirilebilir, böylece karar noktalarına izin verilir.
- **Aritmetik:** ALU tarafından gerçekleştirilen işlemler.
- **Adres değiştirme:** Adreslerin ALU'da hesaplanması ve daha sonra bellekte saklanan talimatlara eklenmesine izin verir. Bu, programa dikkate değer bir adresleme esnekliği sağlar.

Tablo 1.1 komutları (G/C komutları hariç) sembolik, okunması kolay bir biçimde sunmaktadır. İkili formda, her komut Şekil 1.7b'deki formata uygun olmalıdır. İşlem kodu kısmı (ilk 8 bit) 21 talimattan hangisinin yürütüleceğini belirtir. Adres kısmı (kalan 12 bit) 4.096 bellek konumundan hangisinin komutun yürütülmesinde yer alacağını belirtir.

Şekil 1.8'de kontrol ünitesi tarafından yürütülen birkaç komut örneği gösterilmektedir. Her işlemin, bazıları oldukça ayrıntılı olan birkaç adım gerektirdiğine dikkat edin. Çarpma işlemi, işaret biti hariç her bit konumu için bir tane olmak üzere 39 alt işlem gerektirir.

### İkinci Nesil: Transistorler

Elektronik bilgisayardaki ilk büyük değişiklik vakum tüpünün yerini transistörün almasıyla gerçekleşti. Vakum tüpünden daha küçük, daha ucuz ve daha az ısı üreten transistör, bilgisayar yapımında vakum tüpü ile aynı şekilde kullanılabilmektedir. Teller, metal plakalar, cam bir kapsül ve vakum gerektiren vakum tüpünün aksine, transistör silikondan yapılmış bir *katı hal cihazıdır*.

Transistör 1947'de Bell Laboratuvarlarında icat edildi ve 1950'lerde elektronik bir devrim başlattı. Ancak 1950'lerin sonlarına kadar tamamen transistörlü bilgisayarlar ticari olarak mevcut değildi. Transistörün kullanımı *ikinci nesil* bilgisayarları tanımlamaktadır. Bilgisayarları kullanılan temel donanım teknolojisine göre nesiller halinde sınıflandırmak yaygın olarak kabul görmüştür (Tablo 1.2). Her yeni nesil bir öncekine göre daha yüksek işlem performansı, daha büyük bellek kapasitesi ve daha küçük boyut ile karakterize edilmektedir.

Ancak başka değişiklikler de olmuştur. İkinci nesil, daha karmaşık aritmetik ve mantık birimlerinin ve kontrol birimlerinin kullanılmasına, yüksek seviyeli programlama dillerinin kullanılmasına ve *sistem yazılımının*

**Tablo 1.2** Bilgisayar Nesilleri

Nesil	Yaklaşık Tarihler	Teknoloji	Tipik Hız (saniye başına işlem)
1	1946-1957	Vakum tüpü	40,000
2	1957-1964	Transistör	200,000
3	1965-1971	Küçük ve orta ölçekli bütünlleşme	1,000,000
4	1972-1977	Büyük ölçekli entegrasyon	10,000,000
5	1978-1991	Çok büyük ölçekli entegrasyon	100,000,000
6	1991-	Ultra büyük ölçekli entegrasyon	>1,000,000,000

bilgisayar. Geniş anlamda, sistem yazılımı, Windows ve Linux gibi modern işletim sistemlerinin yaptığına benzer şekilde, programları yükleme, verileri çevre birimlerine taşıma ve ortak hesaplamaları gerçekleştirmek için kütüphaneler sağlama yeteneği sağladı.

İkinci neslin önemli bir üyesini incelemek faydalı olacaktır: IBM 7094 [BELL71]. 1952'de 700 serisinin tanıtılmasından 1964'te 7000 serisinin son üyesinin tanıtılmasına kadar, bu IBM ürün serisi bilgisayar ürünlerinde tipik olan bir evrim geçirmiştir. Ürün serisinin birbirini izleyen üyeleri artan performans, artan kapasite ve/veya daha düşük maliyet göstermiştir.

Ana belleğin boyutu,  $2^{10}$  36-bit kelimenin katları olarak,  $2^k$ 'dan ( $1k = 2^{10}$ ) 32k kelimeye çıkarken<sup>7</sup>, belleğin bir kelimesine erişim süresi, bellek çevrim süresi, 30 ms'den 1.4 ms'ye düştü. İşlem kodu sayısı mütevazı bir şekilde 24'ten 185'e çıkmıştır.

Ayrıca, bu bilgisayar serisinin ömrü boyunca, CPU'nun göreceli hızı 50 kat artmıştır. Hız iyileştirmeleri, geliştirilmiş elektronik (örneğin, transistör uygulaması vakum tüpü uygulamasından daha hızlıdır) ve daha karmaşık devrelerle elde edilir. Örneğin, IBM 7094 bir sonraki komutu tamponlamak için kullanılan bir Komut Yedekleme Kaydı içerir. Kontrol birimi, bir komut getirme işlemi için bellekten iki bitistik sözcük getirir. Nispeten seyrek olan (belki %10 ila 15) dallanma komutu oluşumu dışında, bu, kontrol biriminin komut döngülerinin yalnızca yarısında bir komut için belleğe erişmesi gerektiği anlamına gelir. Bu ön-getirme ortalaması komut çevrim süresini önemli ölçüde azaltır.

Sekil 1.9, ikinci nesil bilgisayarları temsil eden IBM 7094 için büyük (birçok çevre birimi) bir yapılandırmayı göstermektedir. IAS bilgisayardan birkaç farklılık dikkat çekmektedir. Bunlardan en önemli *veri kanallarının* kullanılmasıdır. Bir veri kanalı, kendi işlemcisi ve komut seti olan bağımsız bir I/O modülüdür. Bu tür cihazlara sahip bir bilgisayar sisteminde, CPU ayrıntılı I/O talimatlarını yürütmez. Bu tür talimatlar, veri kanalının kendisinde bulunan özel amaçlı bir işlemci tarafından yürütülmek üzere bir ana bellekte saklanır. CPU, veri kanalına bir kontrol sinyali göndererek bellekteki bir dizi talimatı yürütmesi vererek bir G/C aktarımını başlatır. Veri kanalı görevini CPU'dan bağımsız olarak yerine getirir ve işlem tamamlandığında CPU'ya sinyal gönderir. Bu düzenleme CPU'yu önemli bir işlem yükünden kurtarır.

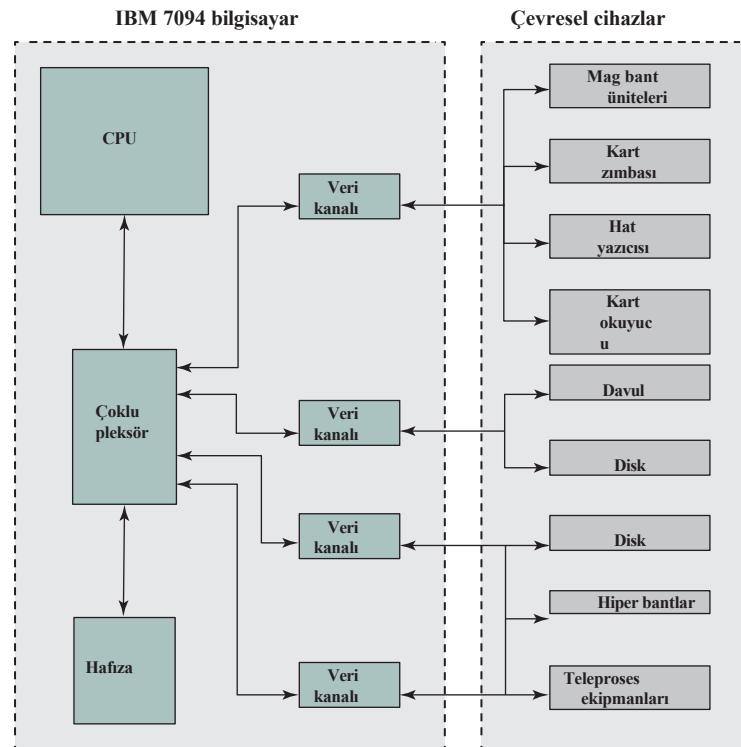
Bir diğer yeni özellik ise veri kanalları, CPU ve bellek için merkezi sonlandırma noktası olan çoklayıcıdır. Çoklayıcı, CPU ve veri kanallarından belleğe erişimi programlayarak bu cihazların bağımsız hareket etmesini sağlar.

### Üçüncü Nesil: Entegre Devreler

Tek, bağımsız bir transistör *ayrık bileşen* olarak adlandırılır. 1950'ler boyunca ve 1960'lارın başında, elektronik ekipmanlar büyük ölçüde ayrık bileşenlerden (transistörler, dirençler, kondensatörler vb.) oluşuyordu. Ayrık bileşenler ayrı ayrı üretilir, kendi kaplarında paketlenir ve lehimlenir veya kablolanır.

---

<sup>7</sup>Kilo ve giga gibi sayısal örneklerin kullanımına ilişkin bir tartışma ComputerScienceStudent.com adresindeki Bilgisayar Bilimleri Öğrenci Kaynak Sitesinde yer alan destekleyici bir dokümanta yer almaktadır.



Şekil 1.9 Bir IBM 7094 Yapılandırması

Masonit benzeri devre kartları üzerinde bir araya getirilerek bilgisayarlara, osiloskoplara ve diğer elektronik ekipmanlara yerleştiriliyordu. Elektronik bir cihaz transistör gerektirdiğinde, toplu iğne başı büyüğündünde bir silikon parçası içeren küçük bir metal tüpün devre kartına lehimlenmesi gerekiyordu. Transistörden devre kartına kadar tüm üretim süreci pahalı ve hantaldı.

Hayatın bu gerçekleri bilgisayar endüstrisinde sorunlar yaratmaya başlamıştı. İlk ikinci nesil bilgisayarlar yaklaşık 10.000 transistör içeriyordu. Bu rakam yüz binlere ulaştı ve daha yeni, daha güçlü makinelerin üretimini giderek zorlaştırdı.

1958 yılında elektronikte devrim yaratan ve mikroelektronik çağını başlatan bir başarıya imza atıldı: entegre devrenin icadı. Üçüncü nesil bilgisayarları tanımlayan entegre devredir. Bu bölümde entegre devre teknolojisine kısa bir giriş yapacağız. Daha sonra üçüncü neslin belki de en önemli iki üyesine bakacağımız, her ikisi de bu dönemin başında tanıtıldı: IBM System/360 ve DEC PDP-8.

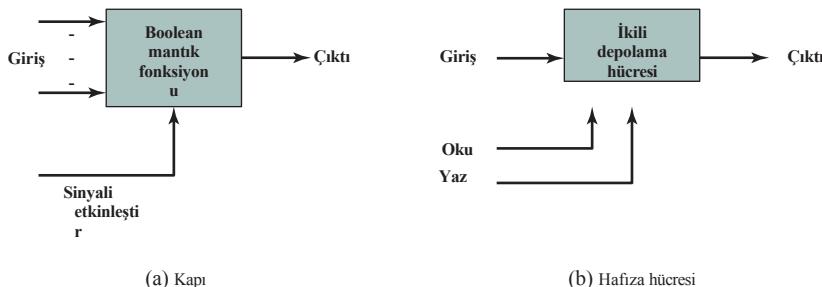
**MİKROELEKTRONİK** Mikroelektronik, kelimenin tam anlamıyla "küçük elektronik" anlamına gelmektedir. Dijital elektronliğin ve bilgisayar endüstrisinin başlangıcından bu yana, dijital elektronik devrelerin boyutlarının küçültülmesi yönünde sürekli ve tutarlı bir eğilim olmuştur. Bu eğilimin sonuçlarını ve faydalarnı incelemeden önce, dijital elektronliğin doğası hakkında bir şeyler söylememiz gereklidir. Daha ayrıntılı bir tartışma Bölüm 11'de yer almaktadır.

Bildiğimiz gibi bir dijital bilgisayarın temel elemanları veri depolama, hareket, işleme ve kontrol işlevlerini yerine getirmelidir. Yalnızca iki temel bileşen türü gereklidir (Şekil 1.10): kapılar ve bellek hücreleri. **Kapı**, basit bir Boolean ya da mantıksal işlevi yerine getiren bir aygittır. Örneğin,  $A$  ve  $B$  girişleri ve  $C$  çıkışı olan bir VE kapısı, EĞER  $A$  VE  $B$  DOĞRU ISE O ZAMAN  $C$  DOĞRUDUR ifadesini uygular. Bu tür cihazlara geçit adı verilir çünkü kanal geçitlerinin su akışını kontrol etmesine benzer şekilde veri akışını kontrol ederler. **Bellek hücresi**, 1 bit veri depolayabilen bir cihazdır; yani cihaz herhangi bir zamanda iki kararlı durumdan birinde olabilir. Bu temel aygitların çok sayıda olanlarını birbirine bağlayarak bir bilgisayar oluşturabiliriz. Bunu dört temel işlevimizle aşağıdaki gibi ilişkilendirebiliriz:

- **Veri depolama:** Bellek hücreleri tarafından sağlanır.
- **Veri işleme:** Gates tarafından sağlanmıştır.
- **Veri hareketi:** Bileşenler arasındaki yollar, verileri bellekten belleğe ve bellekten kapılar aracılığıyla belleğe taşımak için kullanılır.
- **Kontrol:** Bileşenler arasındaki yollar kontrol sinyalleri taşıyabilir. Örneğin, bir geçit bir veya iki veri girişine ve geçidi etkinleştirilen bir kontrol sinyali girişine sahip olacaktır. Kontrol sinyali AÇIK olduğunda, geçit veri girişleri üzerindeki işlevini yerine ve bir veri çıkışı üretir. Tersine, kontrol sinyali KAPALI olduğunda, çıkış hattı, yüksek empedans durumu tarafından üretilen gibi boştur. Benzer şekilde, bellek hücresi YAZMA kontrol sinyali AÇIK olduğunda giriş biti depolar ve OKU kontrol sinyali AÇIK olduğunda hücredeki biti çıkış ucuna yerleştirir.

Dolayısıyla, bir bilgisayar kapılardan, bellek hücrelerinden ve bu unsurlar arasındaki ara bağlantılarından oluşur. Kapılar ve hafıza hücreleri de transistörler ve kapasitörler gibi basit elektronik bileşenlerden oluşur.

Entegre devre, transistörler, dirençler ve iletkenler gibi bileşenlerin silikon gibi bir yarı iletkenden imal edilebileceği gerçekinden yararlanır. Ayrı silikon parçalarından yapılmış ayrı bileşenleri aynı devrede bir araya getirmek yerine tüm devreyi küçük bir silikon parçasında imal etmek katı hal sanatının bir uzantısıdır. Tek bir silikon plaka üzerinde aynı anda çok sayıda transistör üretilebilir. Aynı derecede önemli olarak, bu transistörler devreleri oluşturmak için bir metalizasyon işlemi ile birleştirilebilir.



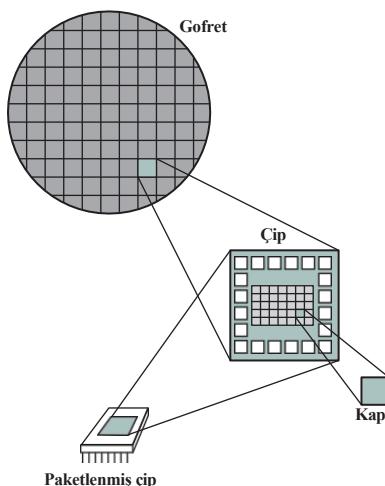
**Şekil 1.10** Temel Bilgisayar Elemanları

Şekil 1.11 bir entegre devredeki temel kavramları göstermektedir. İnce bir silikon plaka, her biri birkaç milimetre kare olan küçük alanlardan oluşan bir matrise bölünmüştür. Her bir alanda aynı devre modeli üretilir ve plaka *ciplere* ayrılır. Her bir çip çok sayıda kapı ve/veya bellek hücresi ile bir dizi giriş ve çıkış bağlantı noktasından oluşur. Bu çip daha sonra kendisini koruyan ve çipin ötesindeki cihazlara takılması için pimler sağlayan bir muhafaza içinde paketlenir. Bu paketlerin bir kısmı daha sonra daha büyük ve daha karmaşık devreler üretmek için bir baskılı devre kartı üzerinde birbirine bağlanabilir.

Başlangıçta, yalnızca birkaç kapı veya bellek hücresi güvenilir bir şekilde üretilip bir araya getirilebiliyordu. Bu ilk entegre devreler **küçük ölçekli entegrasyon (SSI)** olarak adlandırılır. Zaman geçtikçe, aynı çip üzerinde giderek daha fazla bileşeni bir araya getirmek mümkün hale geldi. Yoğunluktaki bu artış Şekil 1.12'de ; şimdiye kadar kaydedilmiş en dikkat çekici teknolojik eğilimlerden biridir.<sup>8</sup> Bu şekil, Intel'in kurucusu Gordon Moore tarafından 1965 yılında ortaya atılan ünlü Moore yassasını yansımaktadır [MOOR65]. Moore, tek bir çip üzerine yerleştirilebilen transistör sayısının her yıl iki katına çıktığını gözlemlemiş ve bu hızın yakın gelecekte de devam edeceğini doğru bir şekilde öngörmüştür. Moore da dahil olmak üzere pek çok kişiyi şaşırtan bir şekilde, bu hız her yıl ve her on yılda bir devam etti. Bu hız 1970'lerde her 18 ayda bir ikiye katlanacak şekilde yavaşladı ancak o zamandan beri bu oranı korudu.

Moore yasasının sonuçları çok derindir:

1. Yoğunluğun hızla arttığı bu dönemde bir çipin maliyeti neredeyse hiç değişmemiştir. Bu da bilgisayar mantığı ve bellek devrelerinin maliyetinin dramatik bir oranda düşüğü anlamına gelmektedir.



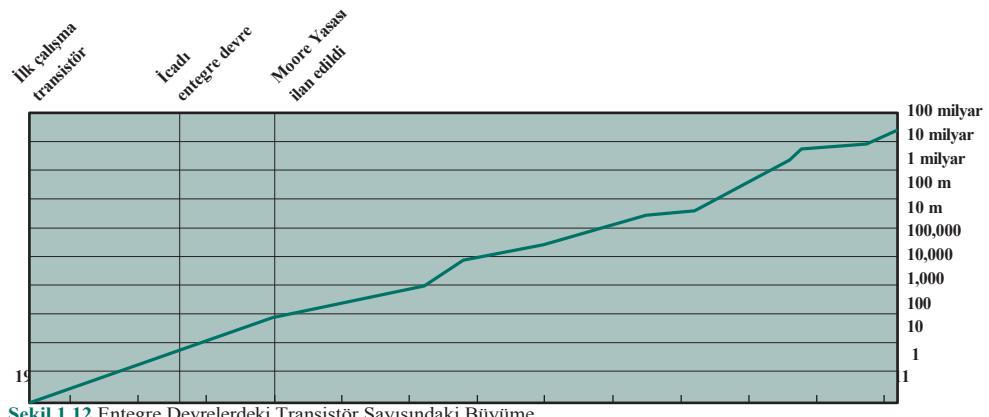
**Şekil 1.11** Wafer, Chip ve Gate arasındaki ilişki

<sup>8</sup>Dickey eksenin log ölçüği kullandığını dikkat edin. Log ölçeklerinin temel bir incelemesi ComputerScienceStudent.com adresindeki Bilgisayar Bilimi Öğrenci Kaynak Sitesinde yer alan matematik tazeleme belgesinde yer almaktadır.

2. Mantık ve bellek elemanları daha yoğun paketlenmiş çipler üzerinde birbirine daha yakın yerleştirildiğinden, elektrik yolu uzunluğu kısılır ve çalışma hızı artar.
3. Bilgisayar küçülür ve çeşitli ortamlara yerleştirmek için daha uygun hale gelir.
4. Güç gereksinimlerinde bir azalma vardır.
5. Entegre devre üzerindeki ara bağlantılar lehim bağlantılarından çok daha güvenilirdir. Her bir çipte daha fazla devre olduğundan, çipler arası daha az bağlantı vardır.

**IBM SYSTEM/360** 1964 yılına gelindiğinde IBM, 7000 serisi makineleriyle bilgisayar pazarında sağlam yer edimiştir. O yıl IBM yeni bir bilgisayar ürünleri ailesi olan System/360'ı duyurdu. Duyurunun kendisi sürpriz olmasa da, mevcut IBM müşterileri için bazı tatsız haberler içeriyordu: 360 ürün grubu eski IBM makineleriyle uyumsuzdu. Bu nedenle 360'a geçiş mevcut müşteri tabanı için zor olacaktı, ancak IBM bunun 7000 mimarisinin bazı kısıtlamalarından kurtulmak ve yeni entegre devre teknolojisiyle gelişebilecek bir sistem üretmek için gerekli olduğunu düşünüyordu [PADE81, GIFF87]. Bu strateji hem finansal hem de teknik olarak karşılığını verdi. 360 on yılın başarısıydı ve IBM'i %70'in üzerindeki pazar payıyla ezici bir üstünlüğe sahip bilgisayar satıcısı haline getirdi. Bazı değişiklikler ve genişletmelerle 360'in mimarisi bugün de IBM'in anabilgisayar<sup>9</sup> bilgisayarlarının mimarisi olmaya devam etmektedir. Bu mimariyi kullanan örnekler bu metin boyunca bulunabilir.

System/360 endüstrinin ilk planlı bilgisayar ailesiydi. Aile geniş bir performans ve maliyet yelpazesini kapsıyordu. Modeller aşağıdakilerle uyumluydu



<sup>9</sup>Mainframe terimi, süper bilgisayarlar dışındaki daha büyük, en güçlü bilgisayarlar için kullanılır. Bir anabilgisayarın tipik özellikleri, büyük bir veritabanını desteklemesi, ayrıntılı I/O donanımına sahip olması ve merkezi bir veri işleme tesisiinde kullanılmasıdır.

Bir model için yazılan bir programın, yalnızca yürütme süresinde bir farkla, serideki başka bir model tarafından yürütülebilmesi gerektiği anlamına gelir.

Birbirıyla uyumlu bilgisayarlardan oluşan bir aile konsepti hem yeni hem de son derece başarılıydı. Mütevazi gereksinimleri ve buna uygun bir bütçesi olan bir müşteri, nispeten ucuz Model 30 ile başlayabilirdi. Daha sonra, müşterinin ihtiyaçları artarsa, önceden geliştirilmiş yazılıma yapılan yatırımdan ödün vermeden daha fazla belleğe sahip daha hızlı bir makineye yükseltmek mümkündü. Bir ailenin özellikleri aşağıdaki gibidir:

- **Benzer veya aynı talimat seti:** Çoğu durumda, ailenin tüm üyelerinde tam olarak aynı makine talimatları seti desteklenir. Dolayısıyla, bir makinede çalışan bir program diğerlerinde de çalışacaktır. Bazı durumlarda, ailenin alt ucu, ailenin üst ucunun bir alt kümlesi bir komut setine sahiptir. Bu, programların yukarı hareket edebileceği ancak aşağı hareket edemeyeceği anlamına gelir.
- **Benzer veya aynı işletim sistemi:** Tüm aile üyeleri için aynı temel işletim sistemi mevcuttur. Bazı durumlarda, üst düzey üyelerde ek özellikler eklenir.
- **Artan hız:** Düşük aile üyelerinden yüksek aile üyelerine doğru gidildikçe komut yürütme hızı artar.
- **Artan I/O port sayısı:** Düşük aile üyelerinden yüksek aile üyelerine doğru gidildikçe I/O portlarının sayısı artar.
- **Artan bellek boyutu:** Düşük aile üyelerinden yüksek aile üyelerine doğru gidildikçe ana belleğin boyutu artar.
- **Artan maliyet:** Belirli bir zamanda, düşük aile üyelerinden yüksek aile üyelerine doğru gidildikçe bir sistemin maliyeti artar.

Böyle bir aile kavramı nasıl uygulanabilir? Farklılıklar üç faktöre bağlı olarak elde edilmiştir: temel hız, boyut ve eşzamanlılık derecesi [STEV64]. Örneğin, belirli bir komutun yürütülmesinde daha yüksek hız, ALU'da daha karmaşık devrelerin kullanılmasıyla elde edilebilir ve alt işlemlerin paralel olarak gerçekleştirilemesine olanak tanır. Hızı artırmanın bir başka yolu da ana bellek ile CPU arasındaki veri yolumun genişliğini artırmaktır. Model 30'da ana bellekten bir seferde sadece 1 bayt (8 bit) alınabilirden, Model 75'te bir seferde 8 bayt alınabiliyordu.

System/360 sadece IBM'in gelecekteki rotasını belirlemekle kalmadı, aynı zamanda tüm endüstri üzerinde de önemli bir etki yarattı. Birçok özelliği diğer büyük bilgisayarlarda standart hale geldi.

**DEC PDP-8** IBM'in ilk System/360'ı sevk ettiği yıl, bir başka önemli ilk sevkiyat gerçekleşti: Digital Equipment Corporation'dan (DEC) PDP-8. Ortalama bir bilgisayarın klimalı bir odaya ihtiyaç duyduğu bir dönemde, PDP-8 (endüstri tarafından günün mini eteklerinden sonra mini bilgisayar olarak adlandırıldı) bir laboratuvar tezgahının üstüne yerleştirilecek veya diğer ekipmanların içine yerleştirilebilecek kadar küçüktü. Ana bilgisayarın yapabildiği her şeyi yapamıyordu, ancak 16.000 dolarla her laboratuvar teknisyeninin bir tane alabileceği kadar ucuzdu. Buna karşılık, sadece birkaç ay önce tanıtılan System/360 serisi ana bilgisayarlar yüz binlerce dolara mal oluyordu.

PDP-8'in düşük maliyeti ve küçük boyutu, başka bir üreticinin bir PDP-8 satın almasını ve yeniden satış için toplam bir sisteme entegre etmesini sağladı. Bu diğer **üreticiler orijinal ekipman üreticileri (OEM)** olarak bilinmeye başlandı ve OEM pazarı bilgisayar pazarının önemli bir bölümünü haline getirdi ve hala da öyle.

IBM'in 700/7000 ve 360 sistemlerinde kullandığı merkezi anahtarlarlamalı mimarının (Şekil 1.9) aksine, PDP-8'in sonraki modelleri mikrobilgisayarlar için evrensel hale gelen bir yapı kullandı: veri yolu yapısı. Bu yapı Şekil 1.13'te gösterilmiştir. Omnibus olarak adlandırılan PDP-8 veriyolu, kontrol, adres ve veri sinyallerini taşımak için kullanılan 96 ayrı sinyal yolundan oluşur. Tüm sistem bileşenleri ortak bir sinyal yolu setini paylaştığından, bunların kullanımı CPU tarafından kontrol edilebilir. Bu mimari son derece esnekdir ve modüllerin çeşitli konfigürasyonlar oluşturmak için veri yoluna takılmasına izin verir. Veri yolu yapısı ancak son yıllarda yerini Bölüm 3'te açıklanan noktadan noktaya ara bağlantı olarak bilinen bir yapıya bırakmıştır.

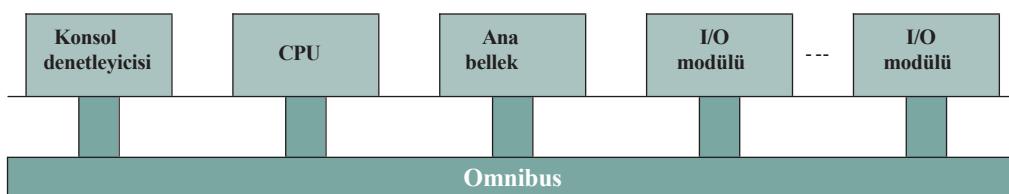
### Sonraki Nesiller

Üçüncü neslin ötesinde, bilgisayar nesillerinin tanımlanması konusunda daha az genel mutabakat vardır. Tablo 1.2, entegre devre teknolojisindeki gelişmeleri göstermektedir. **Büyük ölçekli entegrasyon (LSI)** ortaya çıkışıyla birlikte, tek bir entegre devre çipi üzerine 1.000'den fazla bileşen yerleştirilebilmiştir. Çok büyük ölçekli entegrasyon (VLSI) çip başına 10.000'den fazla bileşene ulaşırken, mevcut ultra büyük ölçekli entegrasyon (ULSI) çipleri bir milyardan fazla bileşen içerebilmektedir.

Teknolojinin hızla ilerlemesi, yeni ürünlerin yüksek oranda piyasaya sürülmESİ ve donanımın yanı sıra yazılım ve iletişimde önem kazanmasıyla, nesillere göre sınırlandırma daha az net ve daha az anlamlı hale gelmektedir. Bu bölümde, sonraki nesillerdeki gelişmelerin en önemlilerinden ikisine değineceğiz.

**YARI İLETKEN BELLEK** Entegre devre teknolojisinin bilgisayarlara ilk uygulaması, işlemcinin (kontrol birimi ve aritmetik ve mantık birimi) entegre devre yongalarından yapılmasıydı. Ancak aynı teknolojinin bellek yapımında da kullanılabilmesi.

1950'lerde ve 1960'larda bilgisayar belleğinin çoğu, her biri yaklaşık bir incin on altında biri içindeki küçük ferromanyetik malzeme halkalarından yapılyordu. Bu halkalar bilgisayarın ekranlarında asılı duran ince tellerden oluşan izgaralara dizilmişti. Bir yöne doğru miknatıslanan bir halka (çekirdek olarak adlandırılır) bir sayısını, diğer yöne doğru miknatıslanan ise sıfır sayısını temsil ederdi. Manyetik çekirdekli bellek oldukça hızlıdı; bellekte depolanan bir biti okumak saniyenin milyonda biri kadar kısa sürüyordu. Ancak



Şekil 1.13 PDP-8 Veri Yolu Yapısı

pahali ve hantaldi ve tahrif edici okuma kullanıyordu: Bir çekirdeğin basit bir okuma eylemi, içinde saklanan verileri siliyordu. Bu nedenle, verileri çıkarılır çıkarılmaz geri yüklemek için devreler kurmak gerekiyordu.

Ardından, 1970 yılında Fairchild ilk nispeten geniş yarı iletken belleği üretti. Yaklaşık tek bir çekirdek büyülüüğündeki bu çip 256 bit bellek tutabiliyordu. Tahribatsızdı ve çekirdektenden çok daha hızlıydı. Bir biti okumak saniyenin sadece 70 milyarda birini alıyordu. Ancak, bit başına maliyet çekirdeğe göre daha yükseltti.

1974 yılında ufuk açıcı bir olay meydana geldi: Yarı iletken belleğin bit başına fiyatı, çekirdek belleğin bit başına fiyatının altına düştü. Bunu takiben, bellek maliyetinde sürekli ve hızlı bir düşüş yaşanmış ve buna fiziksel bellek yoğunlığında karşılık gelen bir artış eşlik etmiştir. Bu durum, sadece birkaç yıl önceki daha büyük ve daha pahalı makinelerin bellek boyutlarına sahip daha küçük, daha hızlı makinelerin yolunu açmıştır. Bellek teknolojisindeki gelişmeler, daha sonra ele alınacak olan işlemci teknolojisindeki gelişmelerle birlikte, on yıldan kısa bir süre içinde bilgisayarların doğasını değiştirmiştir. Hantal, pahalı bilgisayarlar manzaranın bir parçası olmaya devam etse de bilgisayar, ofis makineleri ve kişisel bilgisayarlarla "son kullanıcıya" da sunulmuştur.

1970 yılından bu yana yarı iletken bellekler 13 nesilden geçmiştir: 1k, 4k, 16k, 64k, 256k, 1M, 4M, 16M, 64M, 256M, 1G, 4G ve bu yazı itibarıyle tek bir çip üzerinde 8 Gb ( $1\text{k} = 2^{10}$ ,  $1\text{M} = 2^{20}$ ,  $1\text{G} = 2^{30}$ ). Her nesil, azalan bit başına maliyet ve azalan erişim süresi ile birlikte artan depolama yoğunluğu sağlamıştır. Yoğunlıkların 2018 yılına kadar 16 Gb'a ve 2023 yılına kadar 32 Gb'a ulaşacağı öngörmektedir [ITRS14].

**İŞLEMCİLER** Bellek yongalarındaki elemanların yoğunluğu artmaya devam ettiği gibi, işlemci yongalarındaki elemanların yoğunluğu da artmaya devam etmiştir. Zaman geçtikçe, her bir çip üzerine daha fazla eleman yerleştirildi, böylece tek bir bilgisayar işlemcisi oluşturmak için giderek daha az çipe ihtiyaç duyuldu.

Intel'in 4004'ü geliştirdiği 1971 yılında bir atılım gerçekleştirildi. 4004, bir CPU'nun tüm bileşenlerini tek bir çip üzerinde barındıran ilk çipti: Mikroişlemci doğmuştur.

4004 iki adet 4 bitlik sayı toplayabilir ve sadece tekrarları toplama ile çarpma yapabilir. Bugünün standartlarına göre, 4004 umutsuzca ilkeldir, ancak mikroişlemci kapasitesi ve gücünde devam eden bir evrimin işaret etmiştir.

Bu evrim en kolay şekilde işlemcinin bir seferde uğraştığı bit sayısında görülebilir. Bunun kesin bir ölçüsü yoktur, ancak belki de en iyi ölçü veri yolu genişliğidir: bir işlemciye getirilebilen veya işlemciden gönderilebilen veri bitlerinin sayısı. Bir başka ölçü de toplayıcındaki ya da genel amaçlı kayıt kümelerindeki bit sayısıdır. Genellikle bu ölçütler çakışır, ancak her zaman çakışmaz. Örneğin, kaydedicilerdeki 16 bitlik sayılarla çalışan ancak bir seferde yalnızca 8 bit okuyup yazabilen bir dizi mikroişlemci geliştirilmiştir.

Mikroişlemcinin evrimindeki bir sonraki büyük adım 1972 yılında Intel 8008'in piyasaya sürülmüşiydi. Bu ilk 8-bit mikroişlemciyi ve 4004'ten neredeyse iki kat daha karmaşıktı.

Bu adımların hiçbirini bir sonraki büyük olay kadar etkili olmayacağından: 1974'te Intel 8080'in piyasaya sürülmemesi. Bu ilk genel amaçlı mikro işlemciydi. 4004 ve 8008 belirli uygulamalar için tasarlanmışken, 8080 genel amaçlı bir mikrobilgisayarın CPU'su olarak tasarlanmıştır. Tipki

## 26 BÖLÜM 1 / TEMEL KAVRAMLAR VE BİLGİSAYARIN EVRİMİ

8008, 8080 8 bitlik bir mikroişlemcidir. Ancak 8080 daha hızlıdır, daha zengin bir komut setine sahiptir ve geniş bir adresleme kapasitesine sahiptir.

Yaklaşık aynı zamanlarda 16-bit mikroişlemciler geliştirilmeye başlandı. Ancak 1970'lerin sonuna kadar güçlü, genel amaçlı 16-bit mikroişlemciler ortaya çıkmadı. Bunlardan biri 8086 idi. Bu trenddeki bir sonraki adım 1981 yılında Bell Labs ve Hewlett-Packard'ın 32-bit, tek çipli mikroişlemciler geliştirmesiyile gerçekleşti. Intel 1985 yılında kendi 32-bit mikroişlemcisi olan 80386'yi tanıttı (Tablo 1.3).

**Tablo 1.3** Intel Mikroişlemcilerin Evrimi (sayfa 1 / 2)

(a) 1970'lerin İşlemcileri

	4004	8008	8080	8086	8088
Tanıtıldı	1971	1972	1974	1978	1979
Saat hızları	108 kHz	108 kHz	2 MHz	5 MHz, 8 MHz, 10 MHz	5 MHz, 8 MHz
Otobüs genişliği	4 bit	8 bit	8 bit	16 bit	8 bit
Transistör sayısı	2,300	3,500	6,000	29,000	29,000
Özellik boyutu (mm)	10	8	6	3	6
Adreslenebilir bellek	640 bayt	16 KB	64 KB	1 MB	1 MB

(b) 1980'lerin İşlemcileri

	80286	386TM DX	386TM SX	486TM DX CPU
Tanıtıldı	1982	1985	1988	1989
Saat hızları	6-12,5 MHz	16-33 MHz	16-33 MHz	25-50 MHz
Otobüs genişliği	16 bit	32 bit	16 bit	32 bit
Transistör sayısı	134,000	275,000	275,000	1,2 milyon
Özellik boyutu ( $\mu m$ )	1.5	1	1	0.8-1
Adreslenebilir bellek	16 MB	4 GB	16 MB	4 GB
Sanal bellek	1 GB	64 TB	64 TB	64 TB
Önbellek	-	-	-	8 kB

(c) 1990'lارın İşlemcileri

	486TM SX	Pentium	Pentium Pro	Pentium II
Tanıtıldı	1991	1993	1995	1997
Saat hızları	16-33 MHz	60-166 MHz,	150-200 MHz	200-300 MHz
Otobüs genişliği	32 bit	32 bit	64 bit	64 bit
Transistör sayısı	1.185 milyon	3.1 milyon	5.5 milyon	7.5 milyon
Özellik boyutu ( $\mu m$ )	1	0.8	0.6	0.35
Adreslenebilir bellek	4 GB	4 GB	64 GB	64 GB
Sanal bellek	64 TB	64 TB	64 TB	64 TB
Önbellek	8 kB	8 kB	512 kB L1 ve 1 MB L2	512 kB L2

## (d) Son İşlemciler

	Pentium III	Pentium 4	Core 2 Duo	Core i7 EE 4960X
Tanıtıldı	1999	2000	2006	2013
Saat hızları	450-660 MHz	1.3-1.8 GHz	1,06-1,2 GHz	4 GHz
Otobüs genişliği	64 bit	64 bit	64 bit	64 bit
Transistör sayısı	9.5 milyon	42 milyon	167 milyon	1,86 milyar
Özellik boyutu (nm)	250	180	65	22
Adreslenebilir bellek	64 GB	64 GB	64 GB	64 GB
Sanal bellek	64 TB	64 TB	64 TB	64 TB
Önbellek	512 kB L2	256 kB L2	2 MB L2	1,5 MB L2/15 MB L3
Çekirdek sayısı	1	1	2	6

## 1.4 INTEL x86 ARChITECTUrE'ÜN EVRİMİ

Bu kitap boyunca, kavramları açıklamak ve ödünləşimleri aydınlatmak için bilgisayar tasarımları ve uygulamasına ilişkin birçok somut örneğe dayanıyoruz. Hem çağdaş hem de tarihi çok sayıda sistem, önemli bilgisayar mimarisini tasarım özelliklerine örnekler sunmaktadır. Ancak kitap esas olarak iki işlemci ailesinden örneklerle dayanmaktadır: Intel x86 ve ARM mimarileri. Mevcut x86 teknikleri, karmaşık komut kümeli bilgisayarlar (**CISC'ler**) üzerinde onlarca yıldır süren tasarım çabalarının sonuçlarını temsil etmektedir. Bir zamanlar yalnızca ana bilgisayarlarda veスーパー bilgisayarlarda bulunan sofistik bir tasarım ilkelerini bünyesinde barındıran x86, CISC tasarımlına mükemmel bir örnek teşkil etmektedir. İşlemci tasarımasına alternatif bir yaklaşım da **indirgenmiş komut seti bilgisayarıdır (RISC)**. ARM mimarisi çok çeşitli gömülü sistemlerde kullanılmaktadır ve piyasadaki en güçlü ve en iyi tasarlanmış RISC tabanlı sistemlerden biridir. Bu bölümde ve bir sonraki bölümde, bu iki sistem hakkında kısa bir genel bakış sunacağız. Pazar payı açısından Intel, on yıldır gömülü olmayan sistemler için mikro işlemcilerin bir numaralı üreticisi konumundadır ve bu konumunu kaybetmesi pek olası görünmemektedir. Amiral gemisi mikroişlemci ürününün evrimi iyi bir göstergə olarak hizmet vermektedir. Genel olarak bilgisayar teknolojisinin evrimi.

Tablo 1.3 bu evrimi göstermektedir. İlginç bir şekilde, mikroişlemciler daha hızlı ve çok daha karmaşık hale geldikçe, Intel aslında hızını artırmıştır. Intel eskiden her dört yılda bir birbiri ardına mikroişlemciler geliştirirdi. Ancak Intel bu geliştirme süresini bir ya da iki yıl kısaltarak rakiplerini uzak tutmayı ummamaktadır ve bunu en son x86 nesilleri ile yapmıştır.<sup>10</sup>

<sup>10</sup> Intel bunu *tik-tak modeli* olarak adlandırmaktadır. Bu modeli kullanan Intel, son birkaç yıldır yeni nesil silikon teknolojisini yanı sıra yeni işlemci mikro mimarisini dönüşümlü olarak başarıyla sunmuştur. Bkz. <http://www.intel.com/content/www/us/en/silicon-innovations/intel-tick-tock-model-general.html>.

Intel ürün serisinin gelişiminde öne çıkan bazı noktaları listelemek faydalı olacaktır:

- **8080:** Dünyanın ilk genel amaçlı mikroişlemcisi. Bu, belleğe 8 bitlik veri yolu olan 8 bitlik bir makineydi. 8080 ilk kişisel bilgisayar olan Altair'de kullanıldı.
- **8086:** Çok daha güçlü, 16 bitlik bir makine. Daha geniş bir veri yolu ve daha büyük yazmaçlara ek olarak 8086, çalıştırılmadan önce birkaç talimatı önceden alan bir talimat önbelleği ya da kuyruğuna sahipti. Bu öncünün bir çeşidi olan 8088, IBM'in ilk kişisel bilgisayarında kullanıldı ve Intel'in başarısını güvence altına aldı. 8086, x86 mimarisinin ilk ortaya çıkışıdır.
- **80286:** 8086'nın bu uzantısı, sadece 1 MB yerine 16 MB'lık bir belleğin adreslenmesini sağladı.
- **80386:** Intel'in ilk 32-bit makinesi ve不由得 büyük bir revizyon. 32-bit mimarisiyle 80386, sadece birkaç yıl önce piyasaya sürülen mini bilgisayarların ve ana bilgisayarların karmaşıklığına ve gecenin rakip oldu. Bu, çoklu görevi destekleyen ilk Intel işlemcisiydi, yani aynı anda birden fazla program çalıştırabilirdi.
- **80486:** 80486, çok daha sofistikte ve güçlü önbellek teknolojisinin ve sofistikte komut ardişik dözeninin kullanılmasını sağlamıştır. 80486 ayrıca, karmaşık matematik işlemlerini ana CPU'dan yükleyen yerleşik bir matematik yardımcı işlemcisi de sunuyordu.
- **Pentium:** Pentium ile Intel, birden fazla taliminin paralel olarak yürütülmesine olanak tanıyan süperskalar teknikleri kullanmaya başladı.
- **Pentium Pro:** Pentium Pro, kayıt yeniden adlandırma, dallanma tahmini, veri akışı analizi ve spekulatif yürütmenin agresif kullanımı ile Pentium ile başlayan süper skaler organizasyona geçişti sürdürdü.
- **Pentium II:** Pentium II, özellikle video, ses ve grafik verilerini verimli bir şekilde işlemek için tasarlanmış olan Intel MMX teknolojisini içeriyordu.
- **Pentium III:** Pentium III ek kayan nokta talimatları içermektedir: Streaming SIMD Extensions (SSE) komut seti uzantısı, birden fazla veri nesnesi üzerinde tam olarak aynı işlemlerin gerçekleştirileceği durumlarda performansı artırmak için tasarlanmış 70 yeni talimat ekledi. Tipik uygulamalar dijital sinyal işleme ve grafik işlemedir.
- **Pentium 4:** Pentium 4, multimedya için ek kayan nokta ve diğer geliştirmeleri içerir.<sup>11</sup>
- **Çekirdek:** Bu, tek bir çip üzerinde iki çekirdeğin uygulanmasına atıfta bulunan çift çekirdekli ilk Intel x86 mikroişlemcisidir.
- **Core 2:** Core 2, Core mimarisini 64 bite genişletir. Core 2 Quad, tek bir çip üzerinde dört çekirdek sağlar. Daha yeni Core teklifleri çip başına 10 çekirdeğe kadar sahiptir. Mimariye yapılan önemli bir ekleme, vektör verilerinin verimli bir şekilde işlenmesi için 256 bitlik ve daha sonra 512 bitlik bir dizi talimat sağlayan Gelişmiş Vektör Uzantıları talimat setiydi.

---

<sup>(11)</sup> Pentium 4 ile Intel, model numaraları için Roma rakamlarından Arap rakamlarına geçmiştir.

1978 yılında piyasaya sürülmüşinden neredeyse 40 yıl sonra, x86 mimarisi gömülü sistemler dışındaki işlemci pazarına hakim olmaya devam etmektedir. X86 makinelerinin ve teknolojisi on yıllar içinde önemli ölçüde değişmiş olsa da, komut seti mimarisi daha önceki sürümlerle geriye dönük olarak uyumlu kalacak şekilde gelişmiştir. Böylece, x86 mimarisinin eski bir sürümden yazılıan herhangi bir program daha yeni sürümlerde çalıştırılabilir. Komut kümesi mimarisinde yapılan tüm değişiklikler, komut kümeseine yapılan eklemeleri içermekte olup, herhangi bir çıkarma yapılmamıştır. Değişim oranı, mimariye her ay kabaca bir komut eklenmesi şeklinde olmuştur [ANTH08], böylece komut setinde şu anda binlerce komut bulunmaktadır.

x86, son 35 yılda bilgisayar donanımında kaydedilen ilerlemelerin mükemmel bir örneğini sunmaktadır. 1978 yılında piyasaya sürülen 8086, 5 MHz saat hızına ve 29.000 transistöre sahipti. 2013'te tanıtılan altı çekirdekli Core i7 EE 4960X ise 4 GHz hızında çalışmaktadır, 800 kat hızlanmakta ve 8086'nın yaklaşık 64.000 katı olan 1,86 milyar transistöre sahiptir. Yine de Core i7 EE 4960X, 8086'dan yalnızca biraz daha büyük bir pakette ve benzer bir maliyete sahip.

## 1.5 GÖMÜLÜ SİSTEMLER

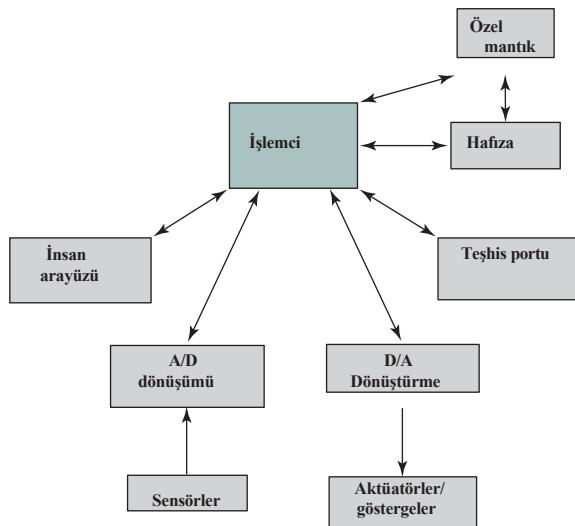
*Gömülü sistem* terimi, dizüstü veya masaüstü sistem gibi genel amaçlı bir bilgisayarın aksine, bir ürün içinde elektronik ve yazılım kullanımını ifade eder. Dizüstü bilgisayarlar, kişisel bilgisayarlar, iş istasyonları, sunucular, ana bilgisayarlar ve süper bilgisayarlar dahil olmak üzere her yıl milyonlarca bilgisayar satılmaktadır. Buna karşılık, her yıl daha büyük cihazların içine yerleştirilmiş milyarlarca bilgisayar sistemi üretilmektedir. Günümüzde elektrik enerjisi kullanan birçok cihazda, belki de çoğunda, gömülü bir iletişim sistemi bulunmaktadır. Yakın gelecekte bu tür cihazların neredeyse tamamının gömülü bilgi işlem sistemlerine sahip olması muhtemeldir.

Gömülü sistemlere sahip cihaz türleri neredeyse listelenemeyecek kadar çoktur. Örnekler arasında cep telefonları, dijital kameralar, video kameralar, hesap makineleri, mikro dalga fırınlar, ev güvenlik sistemleri, çamaşır makineleri, aydınlatma sistemleri, termostatlar, yazıcılar, çeşitli otomotiv sistemleri (örneğin şanzıman kontrolü, hız sabitleyici, yakıt enjeksiyonu, kilitlenmeyi önleyici frenler ve süspansiyon sistemleri), tenis raketleri, diş fırçaları ve otomatik sistemlerdeki çok sayıda sensör ve aktuatör türü yer almaktadır.

Gömülü sistemler genellikle çevreleriyle sıkı bir şekilde bağlıdır. Bu durum, çevreyle etkileşim ihtiyacının dayatığı gerçek zamanlı kısıtlamalara yol açabilir. Gerekli hareket hızları, gerekli ölçüm hassasiyeti ve gerekli zaman süreleri gibi kısıtlamalar yazılım işlemlerinin zamanlamasını belirler. Birden fazla faaliyetin aynı anda yönetilmesi gerekiyorsa, bu daha karmaşık gerçek zamanlı kısıtlamalar getirir.

Şekil 1.14 genel hatlarıyla bir gömülü sistem organizasyonunu göstermektedir. İşlemci ve belleğe ek olarak, tipik masaüstü veya dizüstü bilgisayardan olan bir dizi unsur vardır:

- Sistemin dış çevreyi ölçmesini, manipüle etmesini ve başka şekillerde etkileşime girmesini sağlayan çeşitli arayüzler olabilir. Gömülü sistemler genellikle sensörler ve aktuatörler aracılığıyla dış dünya ile etkileşime girer (algılar, manipüle eder ve iletişim kurar) ve bu nedenle tipik olarak reaktif sistemlerdir; a



**Şekil 1.14** Bir Gömülü Sistemin Olası Organizasyonu

Reaktif sistem çevre ile sürekli etkileşim halindedir ve çevre tarafından belirlenen bir hızda çalışır.

- İnsan arayüzü yanıp sönen bir ışık kadar basit veya gerçek zamanlı robotik görüş kadar karmaşık olabilir. Birçok durumda insan arayüzü yoktur.
- Diyagnoz portu sadece bilgisayarın diyagnozu için değil, kontrol edilen sistemin diyagnozu için de kullanılabilir.
- Performansı veya güvenilirliği artırmak için özel amaçlı sahada programlanabilir (FPGA), uygulamaya özel (ASIC) ve hatta dijital olmayan donanımlar kullanılabilir.
- Yazılım genellikle sabit bir işlev sahiptir ve uygulamaya özeldir.
- Verimlilik gömülü sistemler için büyük önem taşımaktadır. Enerji, kod boyutu, yürütme süresi, ağırlık ve boyutlar ve maliyet açısından edilirler.

Genel amaçlı bilgisayar sistemleriyle de bazı kayda değer benzerlik alanları vardır:

- Nominal olarak sabit işlevli yazılımlarda bile, hataları düzeltmek, güvenliği artırmak işlevsellik eklemek sahada yükseltme yeteneği, yalnızca tüketici cihazlarında değil, gömülü sistemler için de çok önemli hale gelmiştir.
- Nispeten yeni bir gelişme, çok çeşitli uygulamaları destekleyen gömülü sistem platformları olmuştur. Bunun iyi örnekleri akıllı telefonlar ve akıllı TV'ler gibi görsel/işitsel cihazlardır.

## Nesnelerin İnterneti

Gömülü sistemlerin yaygınlaşmasındaki en önemli etkenlerden birine ayrıca degeinmekte fayda var.

**Nesnelerin interneti (IoT)**, nesnelerin internetinin genişlemesini ifade eden bir terimdir.

cihazlardan küçük sensörlerle kadar uzanan akıllı cihazların birbirine bağlanması. Kısa menzilli mobil alıcı-vericilerin çok çeşitli aletlere ve gündelik eşyalara yerleştirilmesi, insanlar ve nesneler arasında nesnelerin kendi aralarında yeni iletişim biçimlerini mümkün kılması önemli bir temadır. İnternet artık genellikle bulut sistemleri aracılığıyla milyarlarca endüstriyel ve kişisel nesnenin birbirine bağlanmasını desteklemektedir. Nesneler sensör bilgilerini iletmekte, çevrelerine göre hareket etmekte ve bazı durumlarda kendilerini değiştirerek bir fabrika ya da şehir gibi daha büyük bir sistemin genel yönetimini oluşturmaktadır.

IoT temel olarak derin gömülü cihazlar (aşağıda tanımlanmıştır) tarafından yönlendirilmektedir. Bu cihazlar, birbirleriyle iletişim kuran ve kullanıcı arayüzleri aracılığıyla veri sağlayan düşük bant genişlikli, düşük tekrarlı veri yakalama ve düşük bant genişlikli veri kullanım cihazlarıdır. Yüksek çözünürlülüklü video güvenlik kameraları, görüntülü VoIP telefonlar ve diğer birkaç gibi gömülü cihazlar, yüksek bant genişliğinde akış yetenekleri gerektirir. Yine de sayısız ürün sadece veri paketlerinin aralıklı olarak iletilmesini gerektirir.

Desteklenen son sistemlere atıfta bulunarak, İnternet, IoT ile sonuçlanan kabaca dört nesil dağıtımdan geçmiştir:

- 1. Bilgi teknolojisi (BT):** Kurumsal BT çalışanları tarafından BT cihazları olarak satın alınan ve öncelikle kablolu bağlantı kullanan PC'ler, sunucular, yönlendiriciler, güvenlik duvarları vb.
- 2. Operasyonel teknoloji (OT):** Kurumsal OT çalışanları tarafından cihaz olarak satın alınan ve öncelikle kablolu bağlantı kullanan tıbbi makineler, SCADA (denetleyici kontrol ve veri toplama), süreç kontrolü ve kiosklar gibi BT dışı şirketler tarafından inşa edilen gömülü BT'ye sahip makineler/ cihazlar.
- 3. Kişisel teknoloji:** Tüketiciler (çalışanlar) tarafından BT cihazı olarak satın alınan ve yalnızca kablosuz bağlantı ve genellikle birden fazla kablosuz bağlantı biçimini kullanan akıllı telefonlar, tabletler ve e-Kitap okuyucular.
- 4. Sensör/aktüatör teknolojisi:** Tüketiciler, BT ve OT çalışanları tarafından satın alınan, daha büyük sistemlerin bir parçası olarak genellikle tek bir formda kablosuz bağlantı kullanan tek amaçlı cihazlar.

Bu, genellikle IoT olarak düşünülen dördüncü nesildir ve milyarlarca gömülü cihazın kullanımıyla dikkat çekmektedir.

## Gömülü İletim Sistemleri

Gömülü bir işletim sistemi (OS) geliştirmek için iki genel yaklaşım vardır. İlk yaklaşım mevcut bir işletim sistemini alıp gömülü uygulama için uyarlamaktır. Örneğin, Linux, Windows ve Mac'in gömülü sürümlerinin yanı sıra gömülü sistemler için özelleşmiş diğer ticari ve tescilli işletim sistemleri de bulunmaktadır. Diğer yaklaşım ise yalnızca gömülü kullanıma yönelik bir işletim sistemi tasarlamak ve uygulamaktır. İkincisine örnek olarak kablosuz sensör ağlarında yaygın olarak kullanılan TinyOS verilebilir. Bu konu [STAL15]'de derinlemesine incelemiştir.

## Uygulama İşlemcilerine Karşı Özel İşlemciler

Bu alt bölümde ve sonraki iki bölümde, gömülü sistemler literatüründe yaygın olarak bulunan bazı terimleri kısaca tanıtabağız. **Uygulama işlemcileri** tanımlanır

işlemcinin Linux, Android ve Chrome gibi karmaşık işletim sistemlerini çalışma kabiliyetine bağlıdır. Bu nedenle, uygulama işlemcisi doğası gereği genel amaçlıdır. Gömülü bir uygulama işlemcisinin kullanımına iyi bir örnek akıllı telefonlardır. Gömülü sistem çok sayıda uygulamayı destekleyecek ve çok çeşitli işlevleri yerine getirecek şekilde tasarlanmıştır.

Çoğu gömülü sistem, adından da anlaşılacağı üzere, ana cihazın gerektirdiği bir veya az sayıda özel görevde adanmış **özel** bir **İşlemci** kullanır. Böyle bir gömülü sistem belirli bir görevde veya görevlere adanmış olduğundan, işlemci ve ilgili bileşenler boyut ve maliyeti azaltacak şekilde tasarlanabilir.

### Mikroişlemcilere karşı Mikrodenetleyiciler

Gördüğümüz gibi, ilk **mikroişlemci** cipleri, bir ALU ve bir çeşit kontrol birimi veya komut işleme mantığı içeriyordu. Transistor yoğunluğu arttıkça, komut seti mimarisinin karmaşıklığını artırmak ve nihayetinde bellek ve birden fazla işlemci eklemek mümkün hale geldi. Şekil 1.2'de gösterildiği gibi çağdaş mikroişlemci cipleri birden fazla çekirdek ve önemli miktarda ön bellek içermektedir.

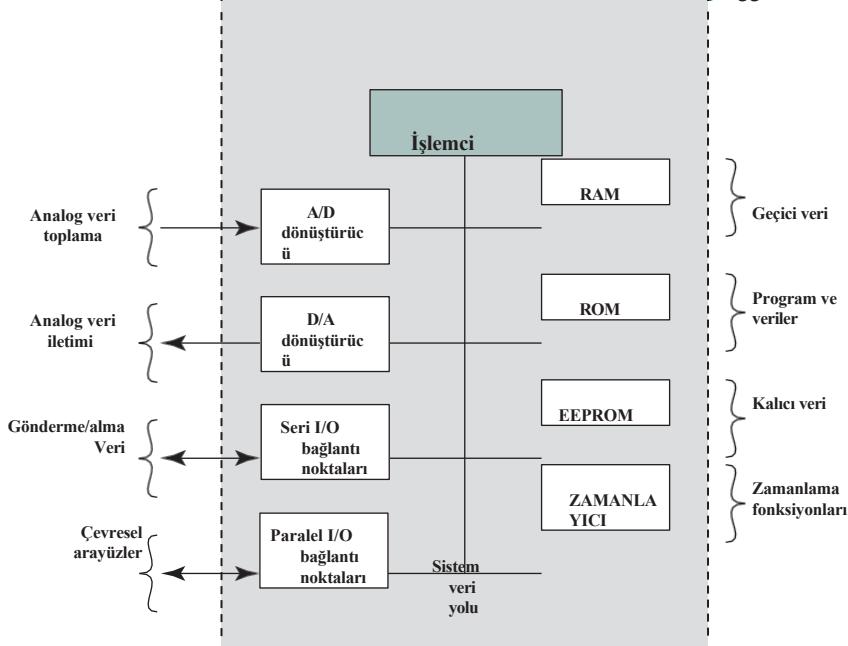
Bir **mikrodenetleyici** çipi, mevcut mantık alanını oldukça farklı bir şekilde kullanır. Şekil 1.15 genel hatlarıyla bir mikrodenetleyici yongasında tipik olarak bulunan unsurları göstermektedir. Göründüğü gibi, bir işlemci, program için ucucu olmayan bellek (ROM), giriş ve çıkış için ucucu bellek (RAM), bir saat ve bir I/O kontrol birimi içeren tek bir çiptir. Mikrodenetleyicinin işlemci kısmı diğer mikroişlemcilere göre çok daha düşük silikon alanına ve çok daha yüksek enerji verimliliğine sahiptir. Mikrodenetleyici organizasyonunu Bölüm 1.6'da daha ayrıntılı olarak inceleyeceğiz.

"Cip üzerinde bilgisayar" olarak da adlandırılan milyarlarca mikrodenetleyici birimi her yıl oyuncaklıardan beyaz eşyalara ve otomobilere kadar sayısız ürüne yerleştirilmektedir. Örneğin, tek bir araçta 70 veya daha fazla mikrodenetleyici kullanılmaktadır. Tipik olarak, özellikle daha küçük, daha ucuz mikrodenetleyiciler için, belirli görevler için özel işlemciler olarak kullanılır. Örneğin, mikrodenetleyiciler otomasyon süreçlerinde yoğun bir şekilde kullanılmaktadır. Girdilere basit tepkiler vererek makineleri kontrol edebilir, fanları açıp, vanaları açıp kapatabilir. Bunlar modern endüstriyel teknolojinin ayrılmaz parçalarıdır ve son derece karmaşık işlevleri yerine getirebilen makineler üretmenin en ucuz yolları arasındadır.

Mikrodenetleyiciler çeşitli fiziksel boyutlara ve işlem gücüne sahiptir. İşlemciler 4-bit ile 32-bit mimarileri arasında değişir. Mikrodenetleyiciler mikroişlemcilerden çok daha yavaş olma eğilimindedir, tipik GHz hızları yerine MHz aralığında çalışırlar. Mikrodenetleyicinin bir diğer tipik özelliği de insan etkileşimi sağlamamasıdır. Mikrodenetleyici belirli bir görev için programlanır, cihazına gömülü ve gerektiği gibi ve gerektiği zaman çalışır.

### Gömülü ve Derin Gömülü Sistemler

Bu bölümde gömülü sistem kavramını tanımladık. Gömülü sistemlerin bir alt kümesi ve oldukça fazla sayıda olan bir alt kümesi, **derin gömülü sistemler** olarak adlandırılır. Bu terim teknik ve ticari alanda yaygın olarak kullanılmasına rağmen



**Şekil 1.15** Tipik Mikrodenetleyici Çip Elemanları

literatürüne bakacak olursanız, doğrudan bir tanım için internette boşuna arama yapacaksınız (ya da en azından ben yaptım). Genel olarak, derin gömülü bir sistemin davranışının hem programcı hem de kullanıcı tarafından gözlemlenmesinin zor olduğu bir sürece sahip olduğunu söyleyebiliriz. Derin gömülü bir sistem mikroişlemci yerine bir mikrodenetleyici kullanır, cihazın program mantığı ROM'a (salt okunur bellek) yazıldıktan sonra programlanamaz ve bir kullanıcıyla etkileşimi yoktur.

Derin gömülü sistemler, ortamda bir şeyi algılayan, temel düzeyde bir işlem gerçekleştiren ve ardından sonuçlarla bir şeyler yapan özel, tek amaçlı cihazlardır. Derin gömülü sistemler genellikle kablosuz özelliğe sahiptir ve geniş bir alana yayılmış sensör ağları (örneğin fabrika, tarım alanı) gibi ağa bağlı konfigürasyonlarda görünür. Nesnelerin interneti büyük ölçüde derin gömülü sistemlere bağlıdır. Tipik olarak, derin gömülü sistemler bellek, işlemci boyutu, zaman ve güç tüketimi açısından aşırı kaynak kısıtlamalarına sahiptir.

## 1.6 ARM ARŞİTEKTÖRÜ

ARM mimarisi, RISC tasarım ilkelerinden gelişen ve gömülü sistemlerde kullanılan işlemci mimarisini ifade eder. Bölüm 15, RISC tasarım ilkelerini ayrıntılı olarak incelemektedir. Bu bölümde, ARM mimarisine kısa bir genel bakış sunulmaktadır.

## ARM Evrimi

ARM, Cambridge, İngiltere'de bulunan ARM Holdings tarafından tasarlanan RISC tabanlı mikroişlemci ve mikrodenetleyici ailesidir. Şirket işlemci üretmez, bunun yerine mikroişlemci ve çok çekirdekli mimariler tasarlarsa ve bunları üreticilere lisanslar. ARM Holdings'in özellikle iki tür lisanslanabilir ürünü vardır: işlemciler ve işlemci mimarileri. İşlemciler için, müşteri ARM tarafından sağlanan tasarımları kendi çiplerinde kullanma haklarını satın alır. Bir işlemci mimarisini için, müşteri ARM'ın mimarisile uyumlu kendi işlemcisini tasarlama haklarını satın alır.

ARM çipleri, küçük kalıp boyutları ve düşük güç gereksinimleri ile bilinen yüksek hızlı işlemcilerdir. Akıllı telefonlarda ve oyun sistemleri de dahil olmak üzere diğer el cihazlarının yanı sıra çok çeşitli tüketici ürünlerinde yaygın olarak kullanılmaktadır. ARM çipleri Apple'in popüler iPod ve iPhone cihazlarındaki işlemcilerdir ve neredeyse tüm Android akıllı telefonlarda da kullanılmaktadır. ARM muhtemelen en yaygın kullanılan gömülü işlemci mimarisidir ve aslında dünyada her türden en yaygın kullanılan işlemci mimarisidir [VANC14].

ARM teknolojisinin kökenleri İngiliz merkezli Acorn Computers şirketine kadar uzanmaktadır. 1980'lerin başında Acorn, BBC Bilgisayar Okuryazarlığı Projesi için yeni bir mikrobilgisayar mimarisi geliştirmek üzere Britanya Yayın Kurumu (BBC) tarafından bir sözleşme ile ödüllendirildi. Bu sözleşmenin başarısı Acorn'un ilk ticari RISC işlemcisi olan Acorn RISC Machine'i (ARM) geliştirmesini sağladı. İlk versiyon olan ARM1 1985 yılında faaliyete geçti ve BBC makinesinde yardımcı işlemci olarak kullanılmasının yanı sıra dahili araştırma ve geliştirme içinde de kullanıldı.

Bu ilk aşamada Acorn, işlemci çiplerinin gerçek üretimini yapmak için VLSI Technology şirketini kullandı. VLSI, çipi kendi başına pazarlamak için lisans aldı ve diğer şirketlerin ARM'yi ürünlerinde, özellikle de gömülü bir işlemci olarak kullanmalarını sağlamada bazı başarılar elde etti.

ARM tasarımları, gömülü uygulamalar için yüksek performanslı, düşük güç tüketimli, küçük boyutlu ve düşük maliyetli bir işlemciye yönelik artan ticari ihtiyacı karşılıyordu. Ancak daha fazla geliştirme Acorn'un yeteneklerinin kapsamı dışındaydı. Buna göre, Acorn, VLSI ve Apple Computer'in kurucusu ortak olduğu ve ARM Ltd. olarak bilinen yeni bir şirket . Acorn RISC Makinesi, Advanced RISC Makineleri haline geldi.<sup>12</sup>

## Komut Seti Mimarisi

ARM komut seti son derece düzenlidir, işlemcinin verimli bir şekilde uygulanması ve verimli yürütme için tasarlanmıştır. Tüm talimatlar 32 bit uzunluğundadır ve düzenli bir formatı takip eder. Bu, ARM ISA'yı geniş bir ürün yelpazesinde uygulama için uygun hale getirir.

Temel ARM ISA'yı tamamlayan Thumb komut seti, ARM komut setinin yeniden kodlanmış bir alt kümesidir. Thumb, 16 bit veya daha dar bellek veri yolu kullanan ARM uygulamalarının performansını artırmak için tasarlanmıştır,

---

<sup>12</sup> Şirket 1990'ların sonunda *Advanced RISC Machines* adını bırakmıştır. Artık sadece ARM mimarisi olarak bilinmektedir.

ve ARM komut setinin sağladığından daha iyi kod yoğunluğuna izin vermek. Thumb komut seti, ARM 32-bit komut setinin 16-bit komutlara yeniden kodlanmış bir alt kümесini içerir. Mevcut tanımlı sürüm Thumb-2'dir.

ARM ve Thumb-2 ISA'ları Bölüm 12 ve 13'te ele alınmıştır.

## ARM Ürünleri

ARM Holdings bir dizi özel mikroişlemcinin ve ilgili teknolojinin lisansını almaktadır, ancak ürün yelpazesinin büyük bir kısmını Cortex mikroişlemci mimarileri ailesi oluşturmaktadır. Baş harfleri A, R ve M ile uygun şekilde etiketlenmiş üç Cortex mimarisi vardır.

**cORTEX-A/cORTEX-A50** Cortex-A ve Cortex-A50, akıllı telefonlar ve e-Kitap okuyucular gibi mobil cihazların yanı sıra dijital TV ve ev ağ geçitleri (örneğin DSL ve kablolu İnternet modemleri) gibi tüketici cihazları için tasarlanmış uygulama işlemcileridir. Bu işlemciler daha yüksek saat frekansında (1 GHz'in üzerinde) çalışır ve Linux, Android, MS Windows ve mobil işletim sistemleri gibi tam özellikli işletim sistemleri için gerekli olan bir bellek yönetim birimini (MMU) destekler. MMU, sanal adresleri fiziksel adreslere çevirerek sanal belleği ve sayfalamayı destekleyen bir donanım modülüdür; bu konu Bölüm 8'de incelenmiştir.

İki mimari hem ARM hem de Thumb-2 komut setlerini kullanır; temel fark Cortex-A'nın 32 bitlik bir makine ve Cortex-A50'nin 64 bitlik bir makine olmasıdır.

**cORTEX-R** Cortex-R, olaylara hızlı yanıt vererek olayların zamanlamasının kontrol edilmesi gereken gerçek zamanlı uygulamaları desteklemek için tasarlanmıştır. Oldukça yüksek bir saat frekansında (örneğin, 200MHz ila 800MHz) çalışabilir ve çok düşük tepki gecikmesine sahiptir. Cortex-R, derin gömülü gerçek zamanlı cihazları desteklemek için hem komut setinde hem de işlemci organizasyonunda geliştirmeler içerir. Bu işlemcilerin çoğunda MMU yoktur; sınırlı veri gereksinimleri ve sınırlı sayıda eşzamanlı işlem, sanal bellek için ayrıntılı donanım ve yazılım desteği ihtiyacını ortadan kaldırır. Cortex-R, endüstriyel uygulamalar için tasarlanmış bir Bellek Koruma Birimi (MPU), önbellek ve diğer bellek özelliklerine sahiptir. MPU, bellekteki bir programın yanlışlıkla başka bir etkin programa atamış belleğe erişmesini yasaklayan bir donanım modülüdür. Çeşitli yöntemler kullanılarak programın etrafında koruyucu bir sınır oluşturulur ve program içindeki talimatların bu sınırın dışındaki verilere referans vermesi yasaklanır.

Cortex-R'yi kullanacak gömülü sistemlere örnek olarak otomotiv fren sistemleri, yığın depolama denetleyicileri, ağı ve baskı cihazları verebilir.

**cORTEX-M** Cortex-M serisi işlemciler, öncelikle hızlı, son derece deterministik kesme yönetimi ihtiyacının son derece düşük kapı sayısı ve mümkün olan en düşük güç tüketimi arzusuya birleştiği mikrodenetleyici alanı için geliştirilmiştir. Cortex-R serisinde olduğu gibi Cortex-M mimarisinde de bir MPU vardır ancak MMU yoktur. Cortex-M yalnızca Thumb-2 komut setini kullanır. Cortex-M için pazar IoT cihazları, fabrikalarda ve diğer işletmelerde kullanılan kablosuz sensör/aktüatör ağları, otomotiv gövde elektronigi vb. içerir.

Şu anda Cortex-M serisinin dört versiyonu bulunmaktadır:

- **Cortex-M0:** 8 ve 16 bit uygulamalar için tasarlanan bu model düşük maliyet, ultra düşük güç ve basitliği vurgular. Küçük silikon kalıp boyutu (12k kapıdan başlayarak) ve en düşük maliyetli çiplerde kullanım için optimize edilmiştir.
- **Cortex-M0+ :** M0'in enerji verimliliği daha yüksek olan geliştirilmiş bir versiyonu.
- **Cortex-M3:** 16 ve 32 bit uygulamalar için tasarlanan bu model, performans ve enerji verimliliğini vurgular. Ayrıca, yazılım geliştiricilerin uygulamalarını hızlı bir şekilde geliştirmelerini sağlamak için kapsamlı hata ayıklama ve izleme özelliklerine sahiptir.
- **Cortex-M4:** Bu model, dijital sinyal işleme görevlerini desteklemek için ek talimatlarla birlikte Cortex-M3'ün tüm özelliklerini sağlar.

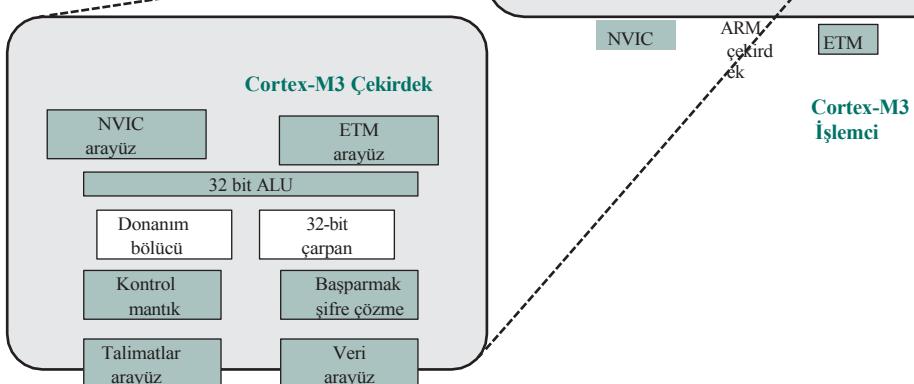
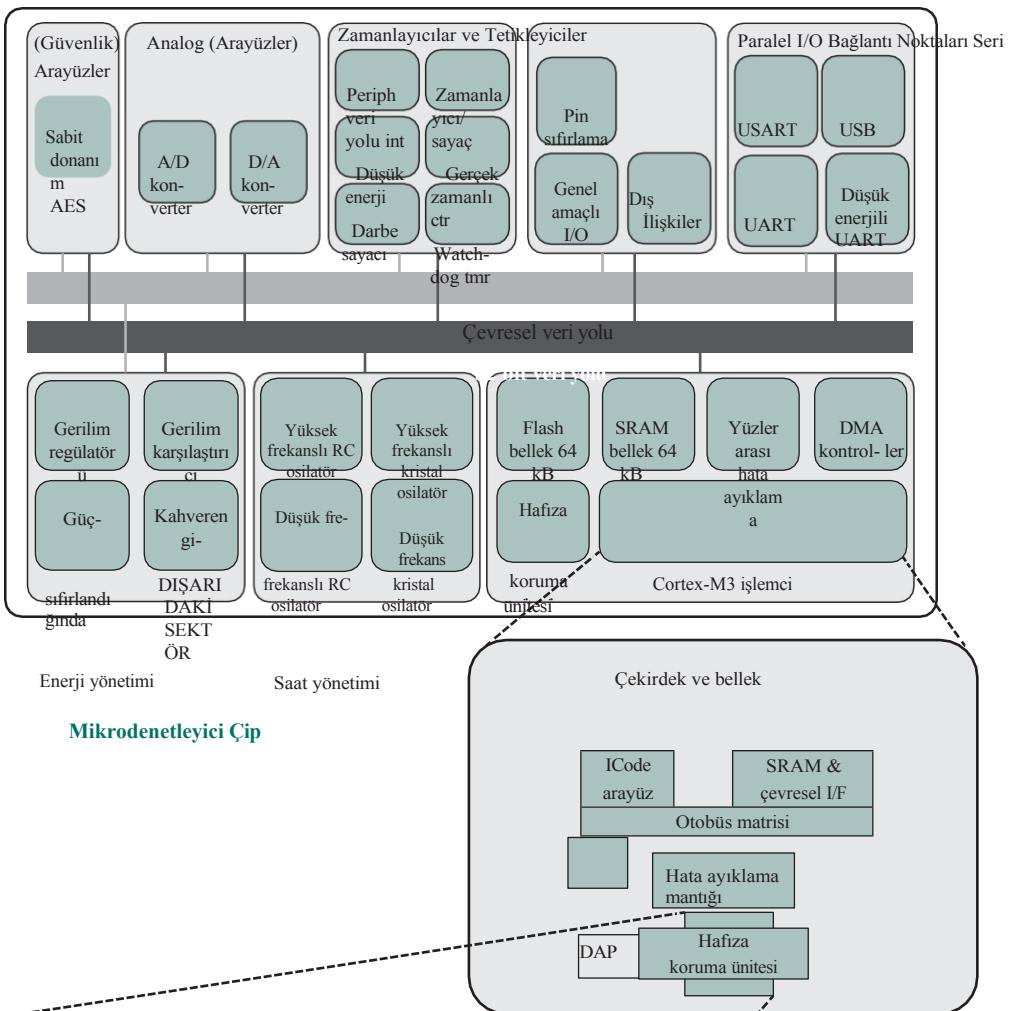
Bu metinde, örnek gömülü sistem işlemcimiz olarak öncelikle ARM Cortex-M3'ü kullanacağız. Genel amaçlı mikro denetleyici kullanımı için tüm ARM modelleri arasında en uygun olmalıdır. Cortex-M3, çeşitli mikro denetleyici ürünleri üreticileri tarafından kullanılmaktadır. Lider ortakların ilk mikro denetleyici cihazları, Cortex-M3 işlemciyi flaş, SRAM ve çoklu çevre birimleriyle birleştirerek sadece 1 \$ fiyatla rekabetçi bir teklif sunmaktadır.

Şekil 1.16, Sil- icon Labs'ın EFM32 mikrodenetleyicisinin blok diyagramını sunmaktadır. Şekilde ayrıca Cortex-M3 işlemci ve çekirdek bileşenlerinin ayrıntıları da gösterilmektedir. Her bir seviyeyi sırayla inceleyeceğiz.

**Cortex-M3 çekirdeği** komutlar ve veriler için ayrı veri yolları kullanır. Bu düzenleme bazen Harvard mimarisi olarak adlandırılır, von Neumann mimarisinin aksine hem talimatlar hem de veriler için aynı sinyal yollarını ve belleği kullanır. Cortex-M3 işlemci hem komutları hem de verileri bellekten aynı anda okuyabildiği için birçok işlemi paralel olarak gerçekleştirebilir ve uygulama yürütmemi hızlandırır. Çekirdek, Thumb talimatları için bir kod çözücü, donanım çarpma ve bölme desteği ile gelişmiş bir ALU, kontrol mantığı ve işlemcinin diğer bileşenlerine arayüzler içerir. Özellikle, iç içe geçmiş vektör kesme denetleyicisine (NVIC) ve gömülü izleme makro hücresi (ETM) modülüne bir arayüz vardır.

Cekirdek, **Cortex-M3 işlemci** adı verilen bir modülün parçasıdır. Bu terim biraz yanlıltıcıdır, çünkü literatürde genellikle çekirdek ve prosesör terimleri eşdeğer olarak görülmektedir. Çekirdeğe ek olarak, işlemci aşağıdaki unsurları içerir:

- **NVIC:** İşlemciye yapılandırılabilir kesme işleme yetenekleri sağlar. Düşük gecikmeli istisna ve kesme işlemlerini kolaylaştırır ve güç yönetimini kontrol eder.
- **ETM:** Program yürütmesinin yeniden yapılandırılmasını sağlayan isteğe bağlı bir hata ayıklama bileşeni. ETM, yalnızca komut izlemeyi destekleyen yüksek hızlı, düşük güçlü bir hata ayıklama aracı olarak tasarlanmıştır.
- **Hata ayıklama erişim portu (DAP):** Bu, işlemciye harici hata ayıklama erişimi için bir arayüz sağlar.
- **Hata ayıklama mantığı:** Temel hata ayıklama işlevi, işlemci durdurma, tek adımlı, işlemci çekirdeği kayıt erişimi, sınırsız yazılım kesme noktası ve tam sistem belleği erişimini içerir.



**Şekil 1.16** Cortex-M3 Tabanlı Tipik Mikrodenetleyici Çipi

- **ICode arayüzü:** Kod bellek alanından talimatları alır.
- **SRAM ve çevresel arayüz:** Veri belleğine ve çevresel aygıtlara okuma/yazma arabirimi.
- **Veri yolu matrisi:** Çekirdek ve hata ayıklama arayızlarını mikrodenetleyici üzerindeki harici veri yollarına bağlar.
- **Bellek koruma birimi:** İşletim sistemi tarafından kullanılan kritik verileri kullanıcı uygulamalarından korur, birbirlerinin verilerine erişime izin vermeyerek işlem görevlerini ayırrı, bellek bölgelerine erişimi devre dışı bırakır, bellek bölgelerinin salt okunur olarak tanımlanmasına izin verir ve sistemi bozabilecek beklenmedik bellek erişimlerini tespit eder.

Şekil 1.16'nın üst kısmı Cortex-M3 ile üretilmiş tipik bir mikro denetleyicinin, bu durumda EFM32 mikro denetleyicisinin blok diyagramını göstermektedir. Bu mikrodenetleyici, enerji, gaz ve su ölçümü; alarm ve güvenlik sistemleri; endüstriyel otomasyon cihazları; ev otomasyon cihazları; akıllı aksesuarlar ve sağlık ve fitness cihazları dahil olmak üzere çok çeşitli cihazlarda kullanılmak üzere pazarlanmaktadır. Sil- icon çipi 10 ana alandan oluşmaktadır:<sup>13</sup>

- **Çekirdek ve bellek:** Bu bölge Cortex-M3 işlemecisini, statik RAM (SRAM) veri belleğini<sup>14</sup> ve program talimatlarını ve değişmeyen uygulama verilerini saklamak için flash belleği<sup>15</sup> içerir. Flash bellek uçucu değildir (güç kapatıldığında veri kaybolmaz) ve bu nedenle bu amaç için idealdir. SRAM değişken verileri depolar. Bu alan aynı zamanda sistemin sahada yeniden programlanması ve güncellenmesini kolaylaştıran bir hata ayıklama arayüzü içerir.
- **Paralel I/O portları:** Çeşitli paralel I/O şemaları için yapılandırılabilir.
- **Seri arayızlar:** Çeşitli seri I/O şemalarını destekler.
- **Analog arayızlar:** Sensörleri ve aktuatörleri desteklemek için analog-dijital ve dijital-analog mantık.
- **Zamanlayıcılar ve tetikleyiciler:** Zamanlamayı takip eder ve olayları sayar, çıkış dalga formları oluşturur ve diğer çevre birimlerinde zamanlanmış eylemleri tetikler.
- **Saat yönetimi:** Çip üzerindeki saatleri ve osilatörleri kontrol eder. Güç tüketimini en aza indirmek ve kısa başlatma süreleri sağlamak için birden fazla saat ve osilatör kullanılır.
- **Enerji yönetimi:** Enerji tüketimini en aza indirmek amacıyla enerji ihtiyaçlarının gerçek zamanlı yönetimini sağlamak için işlemci ve çevre birimlerinin çeşitli düşük enerjili çalışma modlarını yönetir.
- **Güvenlik:** Çip, Gelişmiş Şifreleme Standardının (AES) donanım uygulamasını içerir.

<sup>13</sup>Bu tartışma tüm modüller hakkında ayrıntılara girmemektedir; ilgilenen okuyucu için, [box.com/COA10e](http://box.com/COA10e) adresinde bulunan EFM32G200.pdf belgesinde derinlemesine bir tartışma sunulmaktadır.

<sup>14</sup>Statik RAM (SRAM), ön için kullanılan bir rastgele erişimli bellek biçimidir; bkz.

<sup>15</sup>Flash bellek hem mikro denetleyicilerde hem de harici bellek olarak kullanılan çok yönlü bir biçimidir; Bölüm 6'da ele alınmıştır.

- **32 bit veri yolu:** Çip üzerindeki tüm bileşenleri birbirine bağlar.
- **Çevresel veri yolu:** Farklı çevresel modüllerin işlemciyi dahil etmeden doğrudan birbirleriyle iletişim kurmasını sağlayan bir ağ. Bu, zamanlama açısından kritik işlemleri destekler ve yazılım yükünü azaltır.

Şekil 1.16'yı Şekil 1.2 ile karşılaştığımızda birçok benzerlik ve aynı genel hiyerarşik yapıyı göreceksiniz. Bununla birlikte, bir mikrodenetleyici bilgisayar sisteminin en üst seviyesinin tek bir yonga olduğunu, oysa çok çekirdekli bir bilgisayar için en üst seviyenin bir dizi yonga içeren bir anakart olduğunu unutmayın. Dikkat çeken bir diğer fark ise ne Cortex-M3 işlemcisinde ne de bir bütün olarak mikrodenetleyicide, kod ya da verilerin harici bellekte bulunması durumunda önemli bir rol oynayan önbelleğin bulunmamasıdır. Talimat ya da verinin okunması için gereken döngü sayısı önbelleğin isabet ya da iskalamasına bağlı olarak değişse de, harici bellek kullanıldığında önbellek performansı büyük ölçüde artırır. Bir mikrodenetleyici için böyle bir ek üye gerek yoktur.

## 1.7 BULUT BİLİŞİM

Bulut bilişimin genel kavramları 1950'lere kadar uzansa da, bulut bilişim hizmetleri ilk olarak 2000'li yılların başında, özellikle büyük işletmeleri hedef alarak kullanıma sunulmuştur. O zamandan beri bulut bilişim küçük ve orta ölçekli işletmelere ve en son olarak da tüketicilere yayılmıştır. Apple'in iCloud'u 2012 yılında piyasaya sürüldü ve piyasaya sürüldükten sonraki bir hafta içinde 20 milyon kullanıcıya ulaştı. Bulut tabanlı not alma ve arşivleme hizmeti Evernote, 2008 yılında piyasaya sürüldü ve 6 yıldan kısa bir süre içinde 100 milyon kullanıcıya yaklaştı. Bu bölümde, kısa bir genel bakış sunuyoruz. Bulut bilişim 17. Bölümde daha ayrıntılı olarak incelenmektedir.

### Temel Kavramlar

Birçok kuruluşta bilgi teknolojisi (BT) operasyonlarının önemli bir bölümünü ve hatta tamamını kurumsal bulut bilişim olarak bilinen İnternet bağlantılı bir altyapıya taşıma eğilimi giderek daha fazla öne çıkmaktadır. Aynı zamanda, bireysel PC ve mobil cihaz kullanıcıları da kişisel bulut bilişimi kullanarak veri yedeklemek, cihazları senkronize etmek ve paylaşmak için bulut bilişim hizmetlerine giderek daha fazla güvenmektedir. NIST, bulut bilişimi NIST SP-800-145'te (*Bulut Bilişimin NIST Tanımı*) aşağıdaki şekilde tanımlamaktadır:

**Bulut bilişim:** Minimum yönetim çabası veya hizmet sağlayıcı etkileşimi ile hızlı bir şekilde sağlanabilen ve serbest bırakılabilen yapılandırılabilir bilgi işlem kaynaklarının (ör. ağlar, sunucular, depolama, uygulamalar ve hizmetler) paylaşılan bir havuzuna her yerde, uygun, isteğe bağlı ağ erişimi sağlamaya yönelik bir model.

Temel olarak, bulut bilişim ile ölçek ekonomisi, profesyonel ağ yönetimi ve profesyonel güvenlik yönetimi elde edersiniz. Bu özellikler büyük ve küçük şirketler, devlet kurumları ve bireysel PC ve mobil kullanıcılar için cazip olabilir. Birey veya şirketin yalnızca depolama için ödeme yapması gereklidir.

İhtiyaç duydukları kapasite ve hizmetler. Kullanıcı, ister şirket ister birey olsun, bir veritabanı sistemi kurmak, ihtiyaç duydukları donanımı edinmek, bakım yapmak ve verileri yedeklemek gibi zahmetlere katlanmak zorunda değildir - tüm bunlar bulut hizmetinin bir parçasıdır.

Teoride, verilerinizi depolamak ve başkalarıyla paylaşmak için bulut bilişim kullanmanın bir diğer büyük avantajı da bulut sağlayıcısının güvenlikle ilgilenmesidir. Ne yazık ki, müşteri her zaman korunmuyor. Bulut sağlayıcıları arasında bir dizi güvenlik hatası yaşandı. Evernote, 2013 yılının başlarında bir izinsiz girişin keşfedilmesinin ardından tüm kullanıcılarına şifrelerini sıfırlamalarını söylediğinde manşetlere çıktı.

Bulut **ağı**, bulut bilişimi etkinleştirmek için yerinde olması gereken ağları ve ağ yönetimi işlevsellliğini ifade eder. Çoğu bulut bilişim çözümü internete dayanır, ancak bu ağ altyapısının yalnızca bir parçasıdır. Bulut ağının bir örneği, sağlayıcı ve abone arasında yüksek performanslı ve/veya yüksek güvenilirlikli ağa sağlanmasıdır. Bu durumda, bir işletme ile bulut arasındaki trafiğin bir kısmı ya da tamamı İnternet'i atlar ve bulut hizmeti sağlayıcısının sahip olduğu ya da kiraladığı özel ağ tesislerini kullanır. Daha genel olarak bulut ağı, İnternet üzerinden özel hizmetlerden faydalananmak, kurumsal veri merkezlerini bir buluta bağlamak ve erişim güvenliği politikalarını uygulamak için kritik noktalarda güvenlik duvarları ve diğer ağ güvenliği cihazlarını kullanmak da dahil olmak üzere bir buluta erişmek için gereken ağ kapasitelerinin toplamını ifade eder.

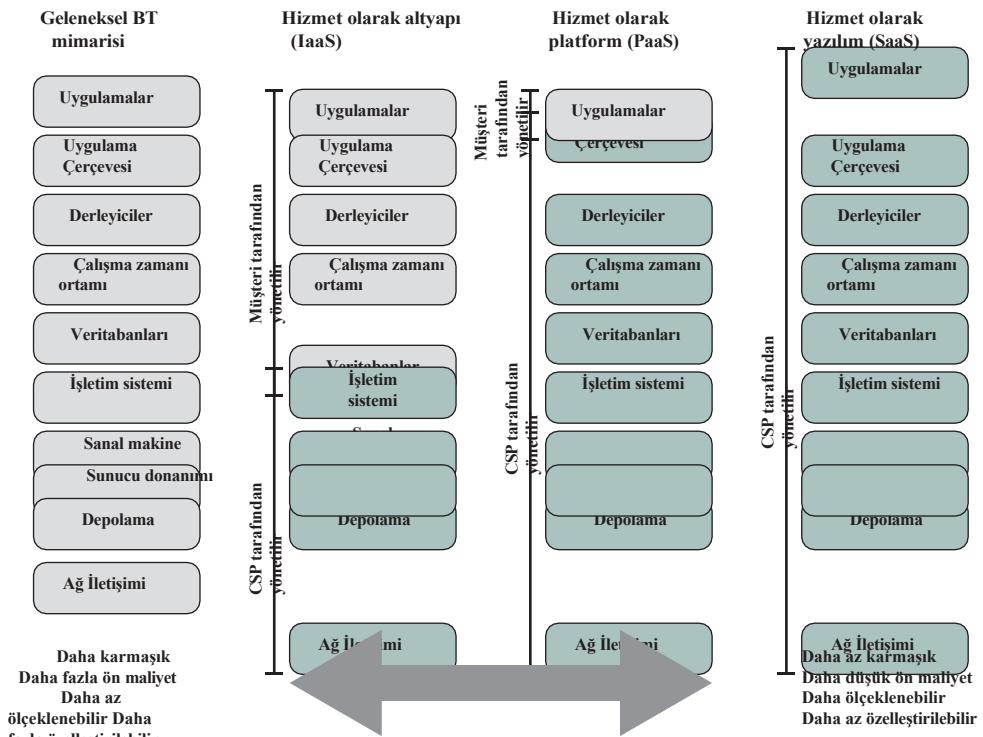
**Bulut depolamayı** bulut bilişimin bir alt kümesi olarak düşünebiliriz. Özünde bulut depolama, bulut sunucularında uzaktan barındırılan veritabanı depolama ve veritabanı uygulamalarından oluşur. Bulut depolama, küçük işletmelerin ve bireysel kullanıcıların, depolama varlıklarını satın almak, bakımı yapmak ve yönetmek zorunda kalmadan ihtiyaçlarına göre ölçeklenen veri depolamadan ve çeşitli veritabanı uygulamalarından yararlanmalarını sağlar.

## Bulut Hizmetleri

Bulut bilişimin temel amacı, bilgi işlem kaynaklarının uygun bir şekilde kiralamasını sağlamaktır. Bir bulut hizmet sağlayıcısı (CSP), İnternet veya özel ağlar üzerinden kullanılabilen bilgi işlem ve veri depolama kaynaklarını muhafaza eder. Müşteriler bu kaynakların bir kısmını gerektiği gibi kiralayabilir. Neredeyse tüm bulut hizmetleri üç modelden biri kullanılarak sağlanır (Şekil 1.17): Bu bölümde incelediğimiz SaaS, PaaS ve IaaS.

**HİZMET OLARAK YAZILIM (SAAS)** Adından da anlaşılacağı üzere, bir SaaS bulutu müşterilere bulut üzerinde çalışan ve buluttan erişilebilen yazılım, özellikle de uygulama yazılımı şeklinde hizmet sağlar. SaaS, Web hizmetlerinin bilinen modelini takip eder, bu durumda bulut kaynaklarına uygulanır. SaaS, müşterinin bulut sağlayıcısının bulut altyapısı üzerinde çalışan uygulamalarını kullanmasını sağlar. Uygulamalara, Web tarayıcı gibi basit bir arayüz aracılığıyla çeşitli istemci cihazlardan erişilebilir. Bir işletme, kullandığı yazılım ürünleri için masaüstü ve sunucu lisansları almak yerine, aynı işlevleri bulut hizmetinden elde eder. SaaS, yazılım kurulumu, bakımı, yükseltmeleri ve yamalarının karmaşıklığından kurtarır. Bu seviyedeki hizmetlere örnek olarak Google'in e-posta hizmeti olan Gmail ve firmaların müşterilerini takip etmelerine yardımcı olan Salesforce.com verilebilir.

SaaS'a yaygın olarak abone olanlar, çalışanlarına belge ve doküman yönetimi gibi tipik ofis üretkenlik yazılımlarına erişim sağlamak isteyen kuruluşlardır.



BT = bilgi teknolojisi CSP= bulut hizmet sağlayıcısı

**Sekil 1.17** Alternatif Bilgi Teknolojisi Mimarileri

yönetimi ve e-posta. Bireyler de bulut kaynaklarını edinmek için yaygın olarak SaaS modelini kullanmaktadır. Tipik olarak, aboneler talep üzerine belirli uygulamaları kullanırlar. Bulut sağlayıcısı genellikle otomatik yedekleme ve aboneler arasında veri paylaşımı gibi veriyle ilgili özellikler de sunar.

**HİZMET OLARAK PLATFORM (PaaS)** PaaS bulutu müşterilere, müşterinin uygulamalarının üzerinde çalışabileceği bir platform şeklinde hizmet sunar. PaaS, müşterinin müşteri tarafından oluşturulan veya satın alınan uygulamaları içeren bulut altyapısına konuşturmasını sağlar. Bir PaaS bulutu, kullanıcıları yazılım yapı taşlarının yanı sıra programlama dilleri, çalışma zamanı ortamları ve yeni uygulamaların dağıtılmasına yardımcı olan diğer araçlar gibi bir dizi geliştirme aracı sağlar. Aslında PaaS, buluttaki bir işletim sistemidir. PaaS, ihtiyaç duyulan bilgi işlem kaynakları için yalnızca ihtiyaç duyulduğunda ve yalnızca ihtiyaç duyulduğu sürece ödeme yaparken yeni veya özel uygulamalar geliştirmek isteyen bir kuruluş için kullanışlıdır. Google App Engine ve Salesforce.com'un Salesforce Platformu PaaS örnekleridir.

**BİR HİZMET OLARAK ALTYAPI (IaaS)** IaaS ile müşterilerin altta yatan bulut altyapısına erişebilir. IaaS, bir hizmet uygulama programlama arayüzü (API) aracılığıyla kontrol edilebilen sanal makineler ve diğer soylananmış donanım ve işletim sistemleri sağlar. IaaS müşteriyeye işleme, depolama, ağlar ve diğer temel bilgi işlem kaynaklarını sunar, böylece müşterileri işletim sistemleri ve uygulamaları içerebilen rastgele yazılımları dağıtabilir ve çalıştırılabilir. IaaS, müşterilerinin sayı hesaplama ve veri depolama gibi temel bilgi işlem hizmetlerini birebirleştirecek son derece uyaranabilir bilgisayar sistemleri oluşturmalarını sağlar. IaaS örnekleri Amazon Elastic Compute Cloud (Amazon EC2) ve Windows Azure'dur.

## 1.8 ANAHTAR TERİMLER, SORULAR VE ÖNERİLER

### Anahtar Terimler

uygulama işlemci arıtmetik ve mantik ünitesi (ALU) ARM merkezi işlem birimi (CPU) Cip bulut bilişim bulut ağı bulut depolama bilgisayar mimarisi bilgisayar organizasyonu kontrol birimi çekirdek özel işlemci derin gömülü sistem gömülü sistem	kapi hizmet olarak altyapı (IaaS) giriş-çıkış (I/O) komut seti mimarisi (ISA) entegre devre Intel x86 Nesnelerin interneti (IoT) ana belleği bellek hücresi bellek yönetimi birimi (MMU) bellek koruma birimi (MPU) mikrodenetleyici mikroelektronik	mikroişlemci anakart çok çekirdekli çok çekirdekli işlemci orijinal ekipman hizmet olarak üretici (OEM) platformu (PaaS) baskılı devre kartı işlemci kayıtlar yarı iletken yarı iletken bellek hizmet olarak yazılım (SaaS) sistem veri yolu sistem ara bağlantı vakum tüpleri
---	--	--

### Inceleme Soruları

- 1.1 Genel anlamda, bilgisayar organizasyonu ile bilgisayar mimarisi arasındaki fark ?
- 1.2 Genel anlamda, bilgisayar yapısı ile bilgisayar işlevi arasındaki ayırım nedir?
- 1.3 Bir bilgisayarın dört ana işlevi nedir?
- 1.4 Bir bilgisayarın ana yapısal bileşenlerini listeleyiniz ve kısaca tanımlayınız.
- 1.5 Bir işlemcinin ana yapısal bileşenlerini listeleyiniz ve kısaca tanımlayınız.
- 1.6 Saklı program bilgisayarı nedir?
- 1.7 Moore yasasını açıklayın.
- 1.8 Bir bilgisayar ailesinin temel özelliklerini listeleyin ve açıklayın.
- 1.9 Bir mikroişlemcinin temel ayırt edici özelliği nedir?

## Problemler

- 1.1** Aşağıdaki denklemin sonuçlarını hesaplamak için bir IAS programı yazmanız gerekmektedir.

$$Y = \sum_{X=1}^N X$$

Hesaplamanın bir aritmetik taşıma ile sonuçlanmadığını ve  $X$ ,  $Y$  ve  $N$ 'nin  $N \geq 1$  olan pozitif tamsayılar olduğunu varsayıñ. Not: IAS'de assembly dili yoktu, sadece makine dili vardı.

- a.** Sum( $Y$ ) denklemini kullanın  $= \frac{N(N+1)}{2}$  IAS programını yazarken. 2  
**b.** Bunu (a) bölümündeki denklemi kullanmadan "zor yoldan" yapın.

- 1.2** a. IAS'de, bellek adresi 2'nin bileşenlerini akümülatöre yüklemek için makine kodu nasıl görünür?  
b. CPU'nun komut döngüsü sırasında bu talimatı tamamlamak için belleğe kaç sefer yapması gereklidir?
- 1.3** IAS üzerinde, CPU'nun bellekten bir değer okumak ve belleğe bir değer yazmak için gerçekleştirmesi gereken süreci MAR, MBR, adres veriyolu, veri veriyolu ve kontrol veriyoluna ne konulduğu açısından İngilizce açıklayın.
- 1.4** Aşağıda gösterilen IAS bilgisayarının bellek içerikleri verilmiştir,

Adres	İçindekiler
08A	010FA210FB
08B	010FA0F08D
08C	020FA210FB

08A adresinden başlayan programın assembly dili kodunu gösterin. Bu programın ne yaptığıni açıklayın.

- 1.5** Şekil 1.6'da, her bir veri yoluñ (örneğin, AC ile ALU arasındaki) genişliğini bit cinsinden belirtin.
- 1.6** IBM 360 Model 65 ve 75'te, adresler iki ayrı ana bellek biriminde kademeleştirilmiştir (örneğin, tüm çift numaralı sözcükler bir birimde ve tüm tek numaralı sözcükler diğerinde). Bu tekninin amacı ne olabilir?
- 1.7** IBM 360 Model 75'in göreceli performansı 360 Model 30'un 50 katıdır, ancak komut döngü süresi yalnızca 5 kat daha hızlıdır. Bu tutarsızlığı nasıl açıklıyorsunuz?
- 1.8** Billy Bob'un bilgisayar mağazasında gezinirken, bir müşterisinin Billy Bob'a mağazada satın alabileceği en hızlı bilgisayarın hangisi olduğunu sorduğuna kulak misafiri oluyorsunuz. Billy Bob söyle cevap veriyor: "Macintosh'larañımıza bakıyorsunuz. Elimizdeki en hızlı Mac 1.2 GHz saat hızında çalışıyor. Eğer gerçekten en hızlı makineyi istiyorsanız, bunun yerine 2.4 GHz Intel Pentium IV'ümüzü satın almalısınız." Billy Bob haklı mı? Bu müşteriye yardımcı olmak için ne söyleyiniz?
- 1.9** ISA makinesinin öncüsü olan ENIAC, her bir kaydın 10 vakum tübünden oluşan bir halka ile temsil edildiği ondalık bir makineydi. Herhangi bir zamanda, 10 ondalık basamaktan birini temsil eden yalnızca bir vakum tübü AÇIK durumdaydı. ENIAC'in aynı anda birden fazla vakum tübü AÇIK ve KAPALI duruma getirme kapasitesine sahip olduğunu varsayırsak, bu temsil neden "savurgan" ve 10 vakum tübüñü kullanarak hangi tamsayı değer aralığını temsil edebiliriz?
- 1.10** Aşağıdaki örneklerin her biri için, bunun gömülü bir sistem olup olmadığını belirleyin ve nedenini veya neden olmadığını açıklayın.
- a.** Fizik ve/veya donanımdan anlayan programlar gömülü mü? Örneğin, uçak kanatları üzerindeki sıvı akışını tahmin etmek için sonlu eleman yöntemlerini kullanan bir program?
- b.** Bir disk sürücüsünü kontrol eden dahili mikroişlemci bir gömülü sistem örneği midir?

- c. sürücülerin donanımı kontrol eder, dolayısıyla bir G/Ç sürücüsünün varlığı sürücüyü çalıştırın bilgisayarın gömülü olduğu anlamına mı gelir?
- d. PDA (Kişisel Dijital Asistan) gömülü bir sistem midir?
- e. Bir cep telefonunu kontrol eden mikroişlemci gömülü bir sistem midir?
- f. Büyük bir faz dizi radarındaki bilgisayarlar gömülü olarak mı kabul edilir? Bu radarlar 10 katlı binalardır ve binanın eğimli kenarlarında bir ya da üç adet 100 feet çapında işiyici yamalar bulunur.
- g. Bir uçak kokpitine yerleştirilmiş geleneksel bir uçuş yönetim sistemi (FMS) gömülü olarak kabul edilir mi?
- h. Döngü içinde donanım (HIL) simülatöründeki bilgisayarlar gömülü mü?
- i. Bir kişinin göğüsündeki kalp pilini kontrol eden bilgisayar gömülü bir bilgisayar mıdır?
- j. Bir otomobil motorunda yakıt enjeksiyonunu kontrol eden bilgisayar gömülü mü?



# BÖLÜM

# 2

## PERFORMANS SORUNLARI

- 2.1 Performans için Tasarım**
  - Mikroişlemci Hız Performans
  - Dengesi
  - Çip Organizasyonu ve Mimarısındaki Gelişmeler
- 2.2 Çok Çekirdekliler, MIC'ler ve GPGPU'lar**
- 2.3 İçgörü Sağlayan İki Yasa: Amdahl Yasası ve Little Yasası**
  - Amdahl Yasası Little Yasası
- 2.4 Bilgisayar Performansının Temel Ölçütleri**
  - Saat Hızı
  - Komut Yürütme Hızı
- 2.5 Ortalamanın Hesaplanması**
  - Aritmetik Ortalama
  - Harmonik Ortalama
  - Geometrik Ortalama
- 2.6 Benchmarklar ve SPEC**
  - Benchmark İlkeleri SPEC
  - Benchmarkları
- 2.7 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler**

**ÖĞRENİM HEDEFLERİ**

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- Bilgisayar tasarımlıyla ilgili temel performans konularını anlamak.
- Çok çekirdekli organizasyona geçişin nedenlerini açıklayın ve tek bir çip üzerinde önbellek ve işlemci kaynakları arasındaki dengeyi anlayın.
- Çok çekirdekli, MIC ve GPGPU organizasyonları arasında ayrım yapabilme.
- Bilgisayar performans değerlendirmesindeki bazı konuları özetleyiniz.
- SPEC ölçütlerini tartışın.
- Aritmetik, harmonik ve geometrik ortalamalar arasındaki farkları açıklayın.

Bu bölüm bilgisayar sistemi performansı konusunu ele almaktadır. Kitap boyunca faydalı olacak bir bakış açısı sağlayan bilgisayar kaynaklarının dengeli kullanımı ihtiyacının değerlendirilmesiyle başlıyoruz. Daha sonra, mevcut ve öngörülen talebi karşılamak için performans sağlamayı amaçlayan çağdaş bilgisayar organizasyonu tasarımlarına bakıyoruz. Son olarak, karşılaşışlı bilgisayar sistemi performansını değerlendirmek için geliştirilmiş araç ve modellere bakıyoruz.

## 2.1 Performans için tasarlama

Her geçen yıl, bilgisayar sistemlerinin maliyeti önemli ölçüde düşmeye devam ederken, bu sistemlerin performansı ve kapasitesi de aynı şekilde önemli artmaya devam ediyor. Günümüzün dizüstü bilgisayarları, 10 ya da 15 yıl önceki bir IBM ana bilgisayarının işlem gücüne sahip. Böylece neredeyse "bedava" bilgisayar gücüne sahip oluyoruz. İşlemciler o kadar ucuzladı ki artık çöpe attığımız mikroişlemcilerimiz var. Dijital hamilelik testi buna bir örnektir (bir kez kullanılır ve sonra atılır). Ve bu süregelen teknolojik devrim, şartlı karmaşıklıkta ve güçte uygulamaların geliştirilmesine olanak sağlamıştır. Örneğin, günümüzün mikroişlemci tabanlı sistemlerinin büyük gücünü gerektiren masaüstü uygulamaları şunlardır

- Görüntü işleme
- Üç boyutlu render
- Konuşma tanıma
- Videokonferans
- Multimedya yazarlığı
- Dosyalara sesli ve görüntülü açıklama ekleme
- Simülasyon modelleme

İş istasyonu sistemleri artık son derece sofistike mühendislik ve bilimsel uygulamaları desteklemekte ve görüntü ve video uygulamalarını destekleme kapasitesine sahiptir. ek olarak, işletmeler işlem ve veritabanı işlemlerini yürütmek ve geçmiş yılların devasa ana bilgisayar merkezlerinin yerini alan devasa istemci/sunucu ağlarını desteklemek için giderek daha güçlü sunuculara güveniyor. Ayrıca, bulut hizmeti

sağlayıcılar, geniş bir müşteri yelpazesine yönelik yüksek hacimli, yüksek işlem oranlı uygulamaları karşılamak için devasa yüksek performanslı sunucu bankaları kullanmaktadır.

Bilgisayar organizasyonu ve mimarisi açısından tüm bunların büyüleyici , bir yandan bilgisayar mucizelerinin temel yapı taşlarının 50 yıl önceki IAS bilgisayarmatkilerle neredeyse aynı olması, diğer yandan da eldeki malzemelerden maksimum performans elde etme tekniklerinin giderek daha sofistike hale gelmesidir.

Bu gözlem, bu kitaptaki sunum için yol gösterici bir ilke olarak hizmet etmektedir. Bir bilgisayarın çeşitli unsurları ve bileşenleri arasında ilerlerken iki hedef gözetilmektedir. Birincisi, kitapta ele alınan her bir alandaki temel işlevsellik açıklanmakta, ikincisi ise maksimum performans elde etmek için gerekli teknikler incelenmektedir. Bu bölümün geri kalanında, performans için tasarım ihtiyacının arkasındaki bazı itici faktörleri vurguluyoruz.

### Mikroişlemci Hızı

Intel x86 işlemcilerine veya IBM ana bilgisayarlarına böylesine akıl almadır bir güç veren şey, işlemci yongası üreticilerinin durmak bilmeyen hız arayışıdır. Bu makinelerin evrimi, Bölüm 1'de açıklanan Moore yasasını doğrulamaya devam etmektedir. Bu yasa geçerli olduğu sürece, çip üreticileri her üç yılda bir dört kat daha fazla transistör içeren yeni bir çip neslini piyasaya sürebilirler. Belkem yongalarında bu durum, halen bilgisayar ana belleği için temel teknoloji olan **dinamik rastgele erişimli belleğin (DRAM)** kapasitesini her üç yılda bir dört katına **çıkarmıştır**. Mikroişlemcilerde, yeni devrelerin eklenmesi ve aralarındaki mesafelerin azaltılmasıyla gelen hız artışı, Intel'in 1978'de x86 ailesini piyasaya sürmesinden bu yana her üç yılda bir performansı dört ya da beş kat artırdı.

Ancak mikroişlemcinin ham hızı, bilgisayar talimatları şeklinde sürekli bir iş akışı ile beslenmediği sürece potansiyeline ulaşamayacaktır. Bu düzgün akışı engelleyen herhangi bir şey gücünü zayıflatır. Buna göre, yonga üreticileri giderek daha büyük yoğunlukta yongaların nasıl üreteceğini öğrenmekle meşgulken, işlemci tasarımcıları canavarı beslemek için her zamankinden daha ayrıntılı teknikler bulmak zorundadır. Çağdaş işlemcilerde yerlesik olarak bulunan teknikler arasında aşağıdakiler yer almaktadır:

- **Pipelining:** Bir komutun yürütülmesi, komutun alınması, işlem kodunun çözülmesi, işlemlerin alınması, hesaplama yapılması gibi birden fazla işlem aşamasını içerir. Pipelining, bir işlemcinin aynı anda birden fazla talimatın her biri için farklı bir aşama gerçekleştirerek birden fazla talimat üzerinde aynı anda çalışmasını sağlar. İşlemci, veri ya da talimatları kavramsal bir boruya taşıyarak ve borunun tüm aşamalarını aynı anda işleyerek işlemleri üst üste bindirir. Örneğin, bir talimat yürütürken bilgisayar bir sonraki kodunu çözmektedir. Bu, bir montaj hattında görülen prensiple aynıdır.
- **Dallanma tahmini:** İşlemci bellekten alınan talimat kodunda ileriye bakar ve hangi dalların veya talimat gruplarının daha sonra işleneceğini tahmin eder. İşlemci çoğu zaman doğru tahminde bulunursa, doğru talimatları önceden alabilir ve işlemcinin meskul kalmasını sağlamak için bunları arabelleğe alabilir. Bu stratejinin daha sofistike örnekleri sadece

bir sonraki dalı değil, birden fazla dalın ilerisini tahmin eder. Böylece dallanma tahmini, işlemcinin yürtütebileceği iş miktarını potansiyel olarak artırır.

- **Süperekler yürütme:** Bu, her işlemci saat döngüsünde birden fazla komut verme yeteneğidir. Gerçekte, birden fazla paralel boru hattı kullanılır.
- **Veri akışı analizi:** İşlemci hangi talimatların birbirlerinin sonuçlarına ya da verilerine bağımlı olduğunu analiz ederek optimize edilmiş bir talimat çizelgesi oluşturur. Aslında, talimatlar orijinal program sırasından bağımsız olarak hazır olduklarında yürütülmek üzere programlanır. Bu da gereksiz gecikmeleri önler.
- **Spekulatif yürütme:** Dallanma tahmini ve veri akışı analizini kullanan bazı işlemciler, komutları program yürütmesinde gerçek görüntülerinden önce spekulatif yürütür ve sonuçları geçici konumlarda tutar. Bu, işlemcinin ihtiyaç duyulması muhtemel talimatları yürütme motorlarını mümkün olduğunda meşgul tutmasını sağlar.

Bu ve diğer sofistike teknikler işlemcinin gücü sayesinde gerekli hale gelmiştir. Toplu olarak, komut başına birçok döngü almak yerine, işlemci döngüsü başına birçok talimatın yürütülmesini mümkün kılarlar.

### Performans Dengesi

İşlemci gücü baş döndürücü bir hızla ilerlerken, bilgisayarın diğer kritik bileşenleri buna ayak uyduramamıştır. Sonuç olarak performans dengesi aramaya ihtiyaç duyulmuştur: çeşitli bileşenlerin yetenekleri arasındaki uyumsuzluğu telsifi etmek için organizasyon ve mimarinin ayarlanması/ayarlanması

Bu tür uyumsuzlıkların yarattığı sorun özellikle işlemci ve ana bellek arasındaki ara yüzde kritik bir hal almaktadır. İşlemci hızı hızla artarken, ana bellek ile işlemci arasında veri aktarım hızı çok geride kalmıştır. İşlemci ve ana bellek arasındaki arayüz tüm bilgisayardaki en önemli yoldur çünkü bellek çipleri ve işlemci arasında sürekli bir program talimatları ve veri akışı sağlamak sorumludur. Bellek ya da yol, işlemcinin ısrarlı taleplerine ayak uyduramazsa, işlemci beklemeye durumunda kalır ve değerli işlem süresi kaybedilir.

Bir sistem mimarı soruna, hepsi de çağdaş bilgisayar tasarımlarına yansyan çeşitli yollarla saldırılabilir. Aşağıdaki örnekleri göz önünde bulundurun:

- DRAM'leri "daha derin" yerine "daha geniş" hale getirerek ve geniş veri yolu veri yolları kullanarak tek seferde alınan bit sayısını artırın.
- DRAM çipine bir önbellek<sup>1</sup> veya başka bir tamponlama şeması ekleyerek daha verimli hale getirmek için DRAM arayüzünü değiştirin.
- İşlemci ve ana arasında giderek daha karmaşık ve verimli önbellek yapıları kullanarak bellek erişim sıklığını azaltın. Bu, işlemci çipi üzerinde bir veya daha fazla önbelleğin yanı sıra işlemci çipine yakın bir çip dışı önbelleğin dahil edilmesini içerir.

---

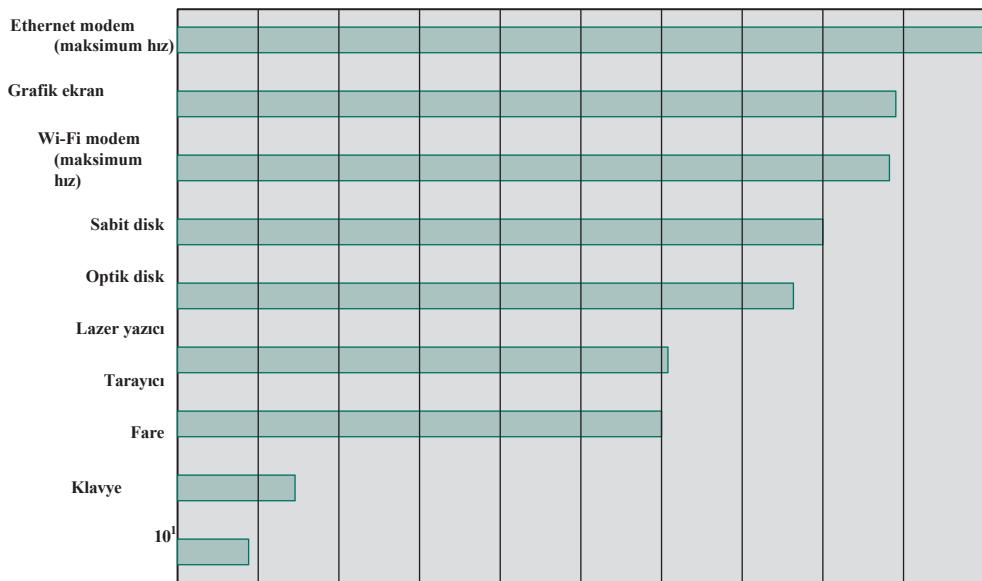
<sup>1</sup> Önbellek, daha büyük, daha yavaş bir bellek ile daha büyük erişen mantık arasına yerleştirilmiş nispeten küçük, hızlı bir bellektir. Önbellek yakın zamanda erişilen verileri tutar ve aynı verilere sonraki erişimleri hızlandırmak için tasarlanmıştır. Önbellekler Bölüm 4'te ele alınmıştır.

- Veri akışını tamponlamak ve yapılandırmak için daha yüksek hızlı veri yolları ve veri yolları hiyerarşisi kullanarak işlemciler ve bellek arasındaki ara bağlantı bant genişliğini artırır.

Tasarımının odaklandığı bir diğer alan da I/O cihazlarının kullanımıdır. Bilgisayarlar daha hızlı ve daha yetenekli hale geldikçe, yoğun I/O talepleri olan çevre birimlerinin kullanımını destekleyen daha sofistike uygulamalar geliştirilmektedir. Şekil 2.1, kişisel bilgisayarlarda ve iş istasyonlarında kullanılan tipik çevresel aygıtlara bazı örnekler vermektedir. Bu aygıtlar muazzam veri çıkışları talepleri yaratmaktadır. Mevcut nesil işlemciler bu aygıtlar tarafından pompalanın verinin üstesinden gelebilse de, bu verinin işlemci ve çevre birimi arasında taşınması sorunu devam etmektedir. Buradaki stratejiler arasında önbellekleme ve tamponlama şemalarının yanı sıra daha yüksek hızlı ara bağlantı veri yollarının ve daha ayrıntılı ara bağlantı yapılarının kullanılması yer almaktadır. Buna ek olarak, çoklu işlemci konfigürasyonlarının kullanılması I/O taleplerinin karşılanmasına yardımcı olabilir.

Tüm bunların anahtarı dengedir. Tasarımcılar sürekli olarak işlemci bileşenlerinin, ana belleğin, I/O cihazlarının ve ara bağlantı yapılarının verim ve işlem taleplerini dengelemeye çalışırlar. Bu tasarım, sürekli gelişen iki faktörle başa çıkmak için sürekli olarak yeniden düşünülmeliidir:

- Çeşitli teknoloji alanlarında (işlemci, veri yolları, bellek, çevre birimleri) performansın değişme hızı, bir öğe türünden diğerine büyük farklılıklar göstermektedir.
- Yeni uygulamalar ve yeni çevresel cihazlar, tipik komut profili ve veri erişim modelleri açısından sistem üzerindeki talebin doğasını sürekli olarak değiştirmektedir.



Şekil 2.1 Tipik G/C Cihazı Veri Hızları

Dolayısıyla, bilgisayar tasarımları sürekli gelişen bir sanat dalıdır. Bu kitap, bu sanatın dayandığı temelleri sunmaya ve bu sanatın mevcut durumuna ilişkin bir anket sunmaya çalışmaktadır.

## Çip Organizasyonu ve Mimarisindeki Gelişmeler

Tasarımcılar işlemci performansını ana bellek ve diğer bilgisayar bileşenleri ile dengeleme zorluğu ile boğuşurken, işlemci hızını artırma ihtiyacı devam etmektedir. İşlemci hızını artırmak için üç yaklaşım vardır:

- İşlemcinin donanım hızının artırılması. Bu artış temel olarak işlemci çipi üzerindeki mantık kapılarının boyutunun küçültülmesi, böylece daha fazla kapının daha sıkı bir şekilde bir araya getirilebilmesi ve saat hızının artırılmasından kaynaklanmaktadır. Birbirine daha yakın kapılar sayesinde sinyallerin yayılma süresi önemli ölçüde azalır ve işlemecinin hızlanması sağlanır. Saat hızındaki bir artış, bireysel işlemlerin daha hızlı yürütülmESİ anlamına gelir.
- İşlemci ile ana bellek arasına yerleştirilen önbelleklerin boyutunu ve hızını artırın. Özellikle, işlemci yongasının bir bölümünün önbelleğe ayrılmışıyla, önbellek erişim süreleri önemli ölçüde düşer.
- İşlemci organizasyonu ve mimarisinde komut yürütmenin etkin hızını artıran değişiklikler yapmak. Tipik olarak bu, bir şekilde paralel izmin kullanılmasını içerir.

Geleneksel olarak, performans kazanımlarındaki baskın faktör, saat hızı ve mantık yoğunluğunundaki artışlar olmuştur. Ancak, saat hızı ve mantık yoğunluğu arttıkça, bir dizi engel daha önemli hale gelmektedir [INTE04]:

- **Güç:** Bir çip üzerindeki mantık yoğunluğu ve saat hızı arttıkça güç yoğunluğu da artar ( $\text{Watt}/\text{cm}^2$ ). Yüksek yoğunluklu, yüksek hızlı yongalarda üretilen ısıyı dağıtmadan zorluğu ciddi bir tasarım sorunu haline gelmektedir [GIBB04, BORK03].
- **RC gecikmesi:** Elektronların bir çip üzerinde transizörler arasında akabilme hızı, onları birbirine bağlayan metal tellerin direnci ve kapasitansi ile sınırlıdır; özellikle RC ürünü arttıkça gecikme de artar. Çip üzerindeki bileşenlerin boyutu küçüldükçe, tel bağlantıları incelir ve direnç artar. Ayrıca, teller birbirine daha yakındır ve kapasitansi artırır.
- **Bellek gecikmesi ve aktarım hızı:** Bellek erişim hızı (gecikme) ve aktarım hızı (verim), daha önce tartışıldığı gibi işlemci hızlarının gerisinde kalır.

Bu nedenle, performansı artırmaya yönelik organizasyon ve mimari yaklaşımlara daha fazla vurgu yapılacaktır. Bu teknikler metnin ilerleyen bölümlerinde ele alınacaktır.

1980'lerin sonlarından başlayarak ve yaklaşık 15 yıl boyunca, sadece saat hızını artırarak elde edilemeyecek olanın ötesinde performansı artırmak için iki ana strateji kullanılmıştır. İlk olarak, önbellek kapasitesinde bir artış olmuştu. Artık işlemci ile ana arasında tipik olarak iki ya da üç önbellek seviyesi bulunmaktadır. Yonga yoğunluğu arttıkça, daha fazla ön bellek yongaya dahil edilerek daha hızlı ön bellek erişimi sağlanmıştır. Örneğin, orijinal Pentium

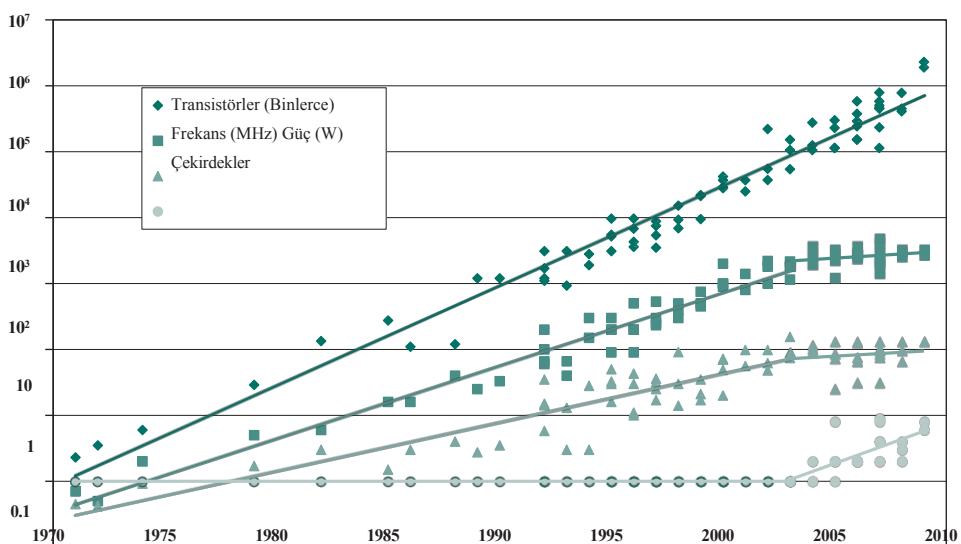
çip, çip üzerindeki alanın yaklaşık %10'unu önbelleğe ayıryordu. Çağdaş çipler çip alanının yarısından fazlasını önbelleklere ayırmaktadır. Ve tipik olarak, diğer yarısının yaklaşık dörtte üçü boru hattı ile ilgili kontrol ve tamponlama içindir.

İkinci olarak, bir işlemci içindeki komut yürütme mantığı, işlemci içindeki komutların paralel yürütülmesini sağlamak için giderek karmaşık hale gelmiştir. İki kayda değer tasarım yaklaşımı boru hattı ve süperskalar olmuştur. Bir boru hattı, bir üretim tesisiindeki montaj hattı gibi çalışarak farklı talimatların yürütülmesinin farklı aşamalarının boru hattı boyunca aynı anda gerçekleşmesini sağlar. Süperskalar yaklaşım özünde tek bir işlemci içinde birden fazla boru hattına izin verir, böylece birbirine bağlı olmayan talimatlar paralel olarak yürütülebilir.

90'ların ortalarından sonlarına doğru, bu yaklaşımın her ikisi de azalan getiri noktasına ulaşıyordu. Çağdaş işlemcilerin iç organizasyonu son derece karmaşıktır ve komut akışından büyük ölçüde paralellik çıkarabilemektedir. Bu yönde daha fazla önemli artışın nispeten mütevazı olacağı muhtemel görünmektedir [GIBB04]. İşlemci yongasında her biri önemli kapasite sağlayan üç önbellek seviyesi ile önbellekten elde edilen faydalardan bir sınıra ulaşıyor gibi görülmektedir.

Ancak, artan performans için sadece artan saat hızına güvenmek, daha önce bahsedilen güç dağılımı sorunuyla karşılaşır. Saat hızı arttıkça harcanan güç miktarı da artmakte ve bazı temel fiziksel sınırlara ulaşılmaktadır.

**Şekil 2.2** tartışmada olduğumuz kavramları göstermektedir.<sup>2</sup> En üstteki satır, Moore Yasası uyarınca, tek bir çip üzerindeki transistör sayısının artmaya devam ettiğini göstermektedir.



Şekil 2.2 İşlemci Eğilimleri

<sup>2</sup>Bu grafiği sağlayan UC Berkeley'den Profesör Kathy Yelick'e minnettarım.

<sup>3</sup> Bu arada, gücün daha da artmasını önlemek için saat hızı dengelenmiştir. Performansı artirmaya devam etmek için tasarımcılar, daha karmaşık bir işlemci inşa etmek dışında artan transistör sayısından faydalananın yollarını bulmak zorunda kalmıştır. Son yillardaki yanıt, çok çekirdekli bilgisayar çipinin geliştirilmesi olmuştur.

## 2.2 MULTICOrE, MICS VE GPGPUS

Önceki bölümde bahsedilen tüm zorluklar göz önünde bulundurulduğunda, tasarımcılar performansı artırmak için temelde yeni bir yaklaşımı yönelmişlerdir: büyük bir ortak önbellek ile aynı çip üzerine birden fazla işlemci yerleştirmek. **Coklu** çekirdek ya da çoklu çekirdek olarak da adlandırılan aynı çip üzerinde **coklu** işlemci kullanımı, saat hızını artırmadan performansı artırma potansiyeli sağlamaktadır. Araştırmalar, bir işlemci içinde performans artışının kabaca karmaşıklıktaki artışın karekökü ile orantılı olduğunu göstermektedir [BORK03]. Ancak yazılım birden fazla işlemcinin etkin kullanımını destekleyebiliyorsa, işlemci sayısını iki katına çıkarmak performansı neredeyse iki katına çıkarır. Bu nedenle strateji, daha karmaşık bir işlemci yerine çip üzerinde daha basit iki işlemci kullanmaktadır.

Buna ek olarak, iki işlemci ile daha büyük önbellekler haklı çıkar. Bu önemlidir çünkü bir çip üzerindeki bellek mantiğının güç tüketimi, işlem mantiğından çok daha azdır.

Çiplerdeki mantık yoğunluğu artmaya devam ettikçe, tek bir çip üzerinde hem daha fazla çekirdek hem de daha fazla önbellek eğilimi de devam ediyor. İki çekirdekli yongaları hızla dört çekirdekli yongalar, ardından 8, sonra 16 ve daha izledi. Önbellekler büyündükçe, bir çip üzerinde iki ve daha sonra üç seviyeli önbellek oluşturmak performans açısından mantıklı hale ; başlangıçta birinci seviye önbellek tek bir işlemciye tahsis edilirken, ikinci ve üçüncü seviyeler tüm işlemciler tarafından paylaştırılıyordu. Artık ikinci seviye önbelleğin de her bir çekirdeğe özel olması yaygındır.

Çip üreticileri şu anda çip başına 50'den fazla çekirdekle, çip başına çekirdek sayısında büyük bir sıçrama yapma sürecindedir. Performanstaki sıçrama ve bu kadar çok sayıda çekirdektenden faydalananmak için yazılım geliştirmedeki zorluklar yeni bir terimin ortaya çıkmasına neden oldu: **çok sayıda entegre çekirdek (MIC)**.

Çok çekirdekli ve MIC stratejisi, tek bir çip üzerinde genel amaçlı işlemcilerin homojen bir şekilde toplanmasını içerir. Aynı zamanda, çip üreticileri başka bir tasarım seçeneğinin peşindedir: birden fazla genel amaçlı işlemcinin yanı sıra **grafik işleme birimleri (GPU'lar)** ve video işleme ve diğer görevler için özel çekirdekler içeren bir çip. Geniş anlamda bir GPU, grafik verileri üzerinde paralel işlemler gerçekleştirmek üzere tasarlanmış bir çekirdektir. Geleneksel olarak bir eklenti grafik kartında (ekran adaptörü) bulunur ve 2D ve 3D grafikleri kodlamak, işlemek ve video işlemek için kullanılır.

GPU'lar birden fazla veri kümesi üzerinde paralel işlemler gerçekleştirdiğinden, tekrarlayan hesaplamalar gerektiren çeşitli uygulamalar için vektör işlemciler olarak giderek daha fazla kullanılmaktadır. Bu durum GPU ve CPU arasındaki çizgiyi bulanıklaştırmaktadır

---

<sup>3</sup> Dikkatli bir okuyucu bu şekildeki transistör sayısı değerlerinin Şekil 1.12'dekilerden önemli ölçüde daha az olduğunu fark edecektir. Bu son şekil, DRAM olarak bilinen ve işlemci yongalarından daha yüksek transistör yoğunluğunu destekleyen bir ana bellek türü için transistör sayısını göstermektedir (Bölüm 5'te tartışılmıştır).

[AROR12, FATA08, PROP11]. Böyle bir işlemci tarafından geniş bir uygulama yelpazesi desteklendiğinde, **GPU'larda genel amaçlı hesaplama (GPGPU)** terimi kullanılır.

Bölüm 18'de çok çekirdekli bilgisayarların ve Bölüm 19'da GPGPU'ların tasarım özelliklerini inceliyoruz.

## 2.3 İÇGÖRÜYE YÖN VEREN İKİ KANUN: AHMDAHL KANUNU VE LITTLE KANUNU

Bu bölümde "yasalar" olarak adlandırılan iki denklemi inceleyeceğiz. Bu iki yasa birbirile ilişkili değildir ancak her ikisi de paralel sistemlerin ve çok çekirdekli sistemlerin performansı hakkında fikir vermektedir.

### Amdahl Yasası

Bilgisayar sistemi tasarımcıları, teknolojideki ilerlemeler veya tasarımdaki değişikliklerle sistem performansını artırmayı ararlar. Örnekler arasında paralel işlemcilerin kullanımı, bellek önbellek hiyerarşisinin kullanımı ve teknolojik gelişmeler nedeniyle bellek erişim süresinde ve G/Ç aktarım hızında hızlanma sağlanabilir. Tüm bu durumlarda, teknolojinin veya tasarımin bir yönündeki hızlanmanın performansta karşılık gelen bir iyileşmeyle sonuçlanmadığına dikkat etmek önemlidir. Bu sınırlama Amdahl yasası ile kısaca ifade edilmektedir.

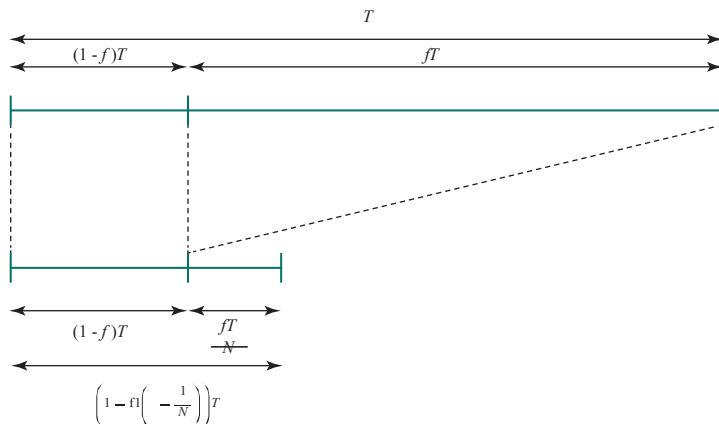
Amdahl yasası ilk olarak 1967 yılında Gene Amdahl tarafından önerilmiştir ([AMDA67], [AMDA13]) ve tek bir işlemci kiyasla birden fazla işlemci kullanan bir programın potansiyel hızlanmasıyla ilgilidir. Tek bir işlemci üzerinde çalışan bir program düşünün, öyle ki yürütme süresinin bir kısmı  $(1 - f)$  doğal olarak sıralı olan kodu ve bir kısmı  $f$  ise zamanlama ek yükü olmadan sonsuz paralelleştirilebilen kodu içeriyor olsun.  $T$ , tek bir işlemci kullanan programın toplam yürütme süresi olsun. Daha sonra, programın paralel kısmını tamamen kullanan  $N$  işlemcili paralel bir işlemci kullanılarak elde edilen hız artışı aşağıdaki gibidir:

$$\begin{aligned} \text{Hızlandırma} &= \frac{\text{Programı tek bir işlemci üzerinde çalıştırma süresi}}{\text{Programı } N \text{ paralel işlemci üzerinde çalıştırma süresi}} \\ &= \frac{T(1 - f) + Tf}{Tf} = \frac{1}{(1 - f) + \frac{f}{N}} \end{aligned}$$

Bu denklem Şekil 2.3 ve 2.4'te gösterilmektedir. İki önemli sonuç çıkarılabilir:

- $f$ 'küçük olduğunda, paralel işlemcilerin kullanımının çok az etkisi vardır.
- $N$  sonsuza yaklaştıkça, hızlanma  $1/(1 - f)$  ile sınırlanır, böylece daha fazla işlemci kullanmanın getirişi azalır.

Bu sonuçlar, ilk olarak [GUST88]'de ortaya atılan bir iddia olan çok kötümseldir. Örneğin, bir sunucu birden fazla istemciyi idare etmek için birden fazla iş parçacığı veya birden fazla görev tutabilir ve iş parçacıklarını veya görevleri işlemci sayısı sınırına kadar paralel olarak yürütebilir. Birçok veritabanı uygulaması, birden fazla paralel görevi bölünebilen büyük miktarda veri üzerinde hesaplama yapılmasını içerir.

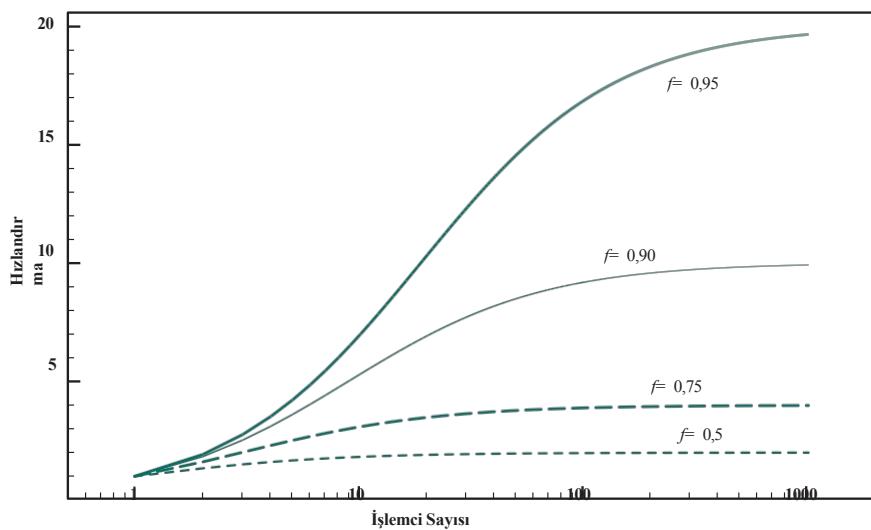


Şekil 2.3 Amdahl Yasasının Gösterimi

Bununla birlikte, Amdahl yasası, sürekli artan sayıda çekirdeğe sahip çok çekirdekli makinelerin geliştirilmesinde endüstrinin karşılaştığı sorunları göstermektedir. Bu tür makinelerde çalışan yazılımlar, paralel işlemenin gücünden faydalananmak için son derece paralel bir yürütme ortamına uyaranmalıdır.

Amdahl yasası, bir bilgisayar sistemindeki herhangi bir tasarım veya teknik iyileştirmeyi değerlendirmek için genelleştirilebilir. Bir sistemin bir özelliğinde hızlanma ile sonuçlanan herhangi bir iyileştirmeyi düşünün. Hızlanma şu şekilde ifade edilebilir

$$\text{Hızlandırma} = \frac{\text{İyileştirme sonrası performans}}{\text{İyileştirme öncesi performans}} = \frac{\text{İyileştirme öncesi yürütme süresi}}{\text{Geliştirmeden sonra yürütme süresi}}$$
(2.1)



Şekil 2.4 Çoklu İşlemciler için Amdahl Yasası

Sistemin bir özelliğinin iyileştirmeden önce yürütme sırasında  $f$  süresinin bir kısmı kadar kullanıldığını ve bu özelliğin iyileştirmeden sonraki hızlanmasıının  $SU_f$  olduğunu varsayılm. O zaman sistemin genel hızlanması

$$\text{Hızlandırma} = \frac{1}{(1-f) + \frac{f}{SU_f}}$$

**ÖRNEK 2.1** Bir görevin kayan nokta işlemlerini yoğun olarak kullandığını ve zamanın %40'ının kayan nokta işlemleri tarafından tüketildiğini varsayılm. Yeni bir donanım de-işareti ile kayan nokta modülü bir  $K$  faktörü kadar hızlandırılır:

$$\text{Hızlandırma} = \frac{1}{0,6 + \frac{0,4}{K}}$$

Böylece,  $K'$  bağımsız olarak, maksimum hızlanma 1,67'dir.

### Little Yasası

Geniş uygulamaları olan temel ve basit bir bağlantı Little Yasasıdır [LITT61, LITT11].<sup>4</sup> Bu yasayı istatistiksel olarak kararlı durumda olan ve sizintisinin olmadığı hemen hemen her sisteme uygulayabiliriz. Spesifik olarak, birim zamanda ortalama  $I$  öğe hızında öğelerin geldiği bir kararlı durum sistemimiz var. Ürünler sisteme ortalama  $W$  birim zaman kalmaktadır. Son olarak, sisteme herhangi bir zamanda ortalama  $L$  birim bulunmaktadır. Little Yasası bu üç değişkeni  $L = IW$  şeklinde ilişkilendirir.

Kuyruk teorisi terminolojisini kullanarak, Little Yasası bir kuyruk sisteme uygulanır. Sistemin merkezi unsuru, öğelere bazı hizmetler sağlayan bir sunucudur. Bazı öğe popülasyonlarından öğeler hizmet verilmek üzere sisteme gelir. Eğer sunucu boştaysa, bir öğeye hemen hizmet verilir. Aksi takdirde, gelen öğe bir bekleme hattına veya kuyruğa katılır. Tek bir sunucu için tek bir kuyruk, birden fazla sunucu için tek bir kuyruk veya birden fazla sunucunun her biri için bir tane olmak üzere birden fazla kuyruk olabilir. Bir sunucu bir öğeye hizmet vermeyi tamamladığında, öğe kuyruktan ayrılır. Eğer bekleyen öğeler varsa, bir tanesi derhal sunucuya gönderilir. Bu modeldeki sunucu, bir öğe koleksiyonu için bazı işlevleri veya hizmetleri yerine getiren herhangi bir şeyi temsil edebilir. Örnekler: Bir işlemci süreçlere hizmet sağlar; bir iletişim hattı paketlere veya veri çerçevelerine iletişim hizmeti sağlar; ve bir G/C cihazı G/C istekleri için okuma veya yazma hizmeti sağlar.

Little'in formülünü anlamak için, tek bir öğenin deneyimine odaklanan aşağıdaki argümanı göz önünde bulundurun. Ürün geldiğinde, üzerinde

<sup>4</sup> İlkinci referans, Little'in orijinal makalesinden 50 yıl sonra yasası üzerine yazdığı geriye dönük bir makaledir. Amdahl [AMDA67] ve [AMDA13] arasındaki 46 yıllık boşlukla yaklaşsa, bu teknik literatür tarihinde benzersiz olmalıdır.

Önünde ortalama  $L$  öğe vardır, biri servis edilirken diğerleri kuyrukta beklemektedir. Öğeye hizmet verildikten sonra sistemden ayrıldığında, geride sistemde ortalama olarak aynı sayıda, yani  $L$  sayıda öğe bırakacaktır, çünkü  $L$  bekleyen öğelerin ortalama sayısı olarak tanımlanır. Ayrıca, ögenin sistemde bulunduğu ortalama süre  $W$ 'dir. Öğeler I oranında geldiğine göre,  $W$  süresi içinde toplam  $IW$  ögenin gelmiş olmasının gerektiğini düşünebiliriz. Dolayısıyla  $w = IW$ .

Özetlemek gerekirse, kararlı durum koşulları altında, bir kuyruk sistemindeki ortalama öğe sayısı, öğelerin geldiği ortalama hız ile bir ögenin sistemde geçirdiği ortalama sürenin çarpımına eşittir. Bu ilişki çok az varsayılmıştır. Servis süresi dağılımının ne olduğunu, varış sürelerinin dağılımının ne olduğunu veya öğelerin edilme sırasını veya öncelliğini bilmemiz gerekmek. Basitliği ve genelliliği nedeniyle Little Yasası son derece kullanışlıdır ve çok çekirdekli bilgisayarlarla ilgili performans sorunlarına olan ilgi nedeniyle bir miktar canlanma yaşamıştır.

LITT11'de yer alan çok basit bir örnek Little Yasası'nın nasıl uygulanabileceğini göstermektedir. Her bir çekirdeğin birden fazla yürütme iş parçacığını desteklediği çok çekirdekli bir sistem düşünün. Bir seviyede, çekirdekler ortak bir belleği paylaşmaktadır. Çekirdekler ortak bir ana belleği paylaşır ve tipik olarak ortak bir ön belleği de paylaşır. Her durumda, bir iş parçacığı yürütülürken, ortak bellekten bir veri parçası alması gereken bir noktaya gelebilir. İş parçacığı durur ve bu veri için bir istek gönderilir. Bu şekilde durdurulan tüm iş parçacıkları bir kuyrukta yer alır. Sistem bir sunucu olarak kullanılıyorsa, bir analist sistem üzerindeki talebi kullanıcı isteklerinin oranı açısından belirleyebilir ve ardından bunu tek bir kullanıcı istegine yanıt vermek için oluşturulan iş parçacıklarından gelen veri isteklerinin oranına dönüştürebilir. Bu amaçla, her bir kullanıcı isteği iş parçacığı olarak uygulanan alt görevlere ayrılır. Daha sonra  $I =$  tüm üyelerin talepleri olan ayrıntılı alt görevlere ayrıldıktan sonra gereken ortalama toplam iş parçacığı işleme oranına sahip oluruz.  $L$ 'yi ilgili bir süre boyunca bekleyen durdurulmuş iş parçacıklarının ortalama sayısı olarak tanımlayın. Ardından  $W =$  ortalama yanıt süresi. Bu basit model, kullanıcı gereksinimlerinin karşılanıp konusunda tasarımcılara bir kılavuz görebilir ve karşılanmıyorsa, ihtiyaç duyulan iyileştirme miktarının niceliksel bir ölçüsünü sağlayabilir.

## 2.4 BİLGİSAYAR PERFORMANSININ TEMEL ÖLÇÜTLERİ

İşlemci donanımını değerlendirdirken ve yeni sistemler için gereksinimleri belirlerken, performans, maliyet, boyut, güvenlik, güvenilirlik ve bazı durumlarda güç tüketimi ile birlikte dikkate alınması gereken temel parametrelerden biridir.

Farklı işlemciler arasında, hatta aynı ailedeki işlemciler arasında bile anlamlı performans karşılaştırması yapmak zordur. Ham hız, bir işlemcinin belirli bir uygulamayı yürütürken nasıl performans gösterdiğiinden çok daha az önemlidir. Ne yazık ki, uygulama performansı sadece işlemcinin ham hızına değil, aynı zamanda komut setine, uygulama dilinin seçimine, derleyicinin verimliliğine ve uygulamayı gerçekleştirmek için yapılan programlama becerisine bağlıdır.

Bu bölümde, bazı geleneksel işlemci hızı ölçümüline bakacağız. Bir sonraki bölümde, işlemci ve bilgisayar sistemi performansını değerlendirmek için en yaygın yaklaşım olan kıyaslamayı inceleyeceğiz. Bir sonraki bölümde birden fazla testten elde edilen sonuçların ortalamasının nasıl alınacağı tartışılmaktadır.

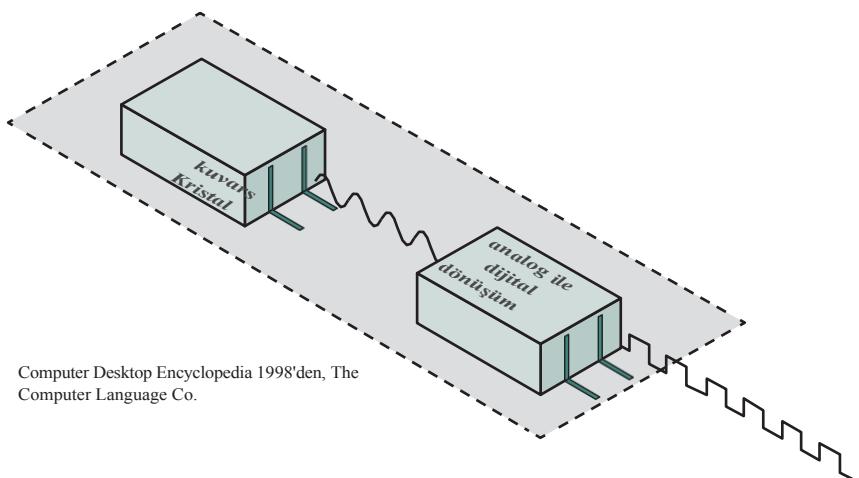
### Saat Hızı

Bir işlemci tarafından gerçekleştirilen bir talimatın alınması, talimatın kodunun çözülmesi, bir aritmetik işlemin gerçekleştirilmesi gibi işlemler bir sistem saatı tarafından yönetilir. Tipik olarak, tüm işlemler saatin darbesiyle başlar. Dolayısıyla, en temel düzeyde, bir işlemcinin hızı, saat tarafından üretilen ve saniye başına döngü veya Hertz (Hz) olarak ölçülen darbe frekansı tarafından belirlenir.

Tipik olarak saat sinyalleri, güç uygulandığında sabit bir sinüs dalgası üreten bir kuvars kristalı tarafından üretilir. Bu dalga, işlemci devresine sabit bir akışla sağlanan dijital bir voltaj darbe akışına dönüştürülür (Şekil 2.5). Örneğin, 1 GHz'lik bir işlemci saniyede 1 milyar darbe alır. Darbe oranı **saat oranı** veya **saat hızı** olarak bilinir. Saatin bir artışı veya darbesi bir saat **döngüsü** veya bir **saat tik'i** olarak adlandırılır. Darbeler arasındaki süre **döngü süresidir**.

Saat hızı keyfi olmayıp işlemcinin fiziksel düzenebine uygun olmalıdır. İşlemcideki eylemler, sinyallerin bir işlemci elemanından diğerine gönderilmesini gerektirir. İşlemci içindeki bir hatta bir sinyal yerleştirildiğinde, doğru bir değerin (mantıksal 1 veya 0) elde edilebilmesi için voltaj seviyelerinin oturması sınırlı bir süre alır. Ayrıca, işlemci devrelerinin fiziksel düzenevine bağlı olarak, bazı sinyaller diğerlerinden daha hızlı değişimlidir. Bu nedenle, her işlem için uygun elektriksel signaal (voltaj) değerlerinin mevcut olması için işlemler senkronize edilmeli ve hızlandırılmalıdır.

Bir talimatın yürütülmesi, talimatın bellekten alınması, talimatın çeşitli bölümlerinin kodunun çözülmesi, verilerin yüklenmesi ve depolanması ve aritmetik ve mantıksal işlemlerin gerçekleştirilmesi gibi bir dizi ayrı adımı içerir. Bu nedenle, çoğu işlemcideki çoğu talimatın tamamlanması için birden fazla saat döngüsü gereklidir. Bazı talimatlar yalnızca birkaç döngü alırken, diğerleri düzinelere döngü gerektirebilir. Buna ek olarak, pipelining kullanıldığında, birden fazla talimat aynı anda yürütülür. Bu nedenle, farklı işlemcilerdeki saat hızlarının doğrudan karşılaştırılması performans hakkında tüm hikayeyi anlatmaz.



**Şekil 2.5** Sistem Saati

### Komut Yürütme Hızı

Bir işlemci sabit bir  $f$  frekansına sahip bir saat ya da eşdeğer olarak sabit bir t çevrim süresi tarafından çalıştırılır, burada  $t = 1/f$ . Bir program için komut sayısını,  $I_c$ , program tamamlanana kadar ya da belirli bir zaman aralığında çalıştırılan makine komutlarının sayısı olarak tanımlayın. Bunun, programın nesne kodundaki komut sayısı değil, komut yürütme sayısı olduğuna dikkat edin. Önemli bir parametre de bir program için komut başına ortalama çevrim sayısıdır ( $CPI$ ). Eğer tüm komutlar aynı sayıda saat döngüsü gerektirseydi,  $CPI$  bir işlemci için sabit bir değer olurdu. Ancak, herhangi bir işlemcide, gerekli saat döngüsü sayısı yükleme, saklama, dalanma farklı komut türleri için değişir.  $CPI_i$ ,  $i$  komut türü için gereken döngü sayısı ve  $I_i$ , belirli bir program için  $i$  türünde yürütülen komutların sayısı olsun. Daha sonra genel bir  $CPI$ 'yi aşağıdaki gibi hesaplayabiliriz:

$$= \frac{\sum_{i=1}^n (T\text{ÜFE}_i * I_i)}{I_c} \quad (2.2)$$

Belirli bir programı çalıştmak için gereken işlemci süresi  $T$  şu şekilde ifade edilebilir

$$T = I_c * T\text{ÜFE} * t$$

Bu formülasyonu, bir komutun yürütülmesi sırasında işin bir kısmının işlemci tarafından yapıldığını ve zamanın bir kısmında bir kelimenin belleğe veya bellekten aktarıldığını kabul ederek geliştirebiliriz. Bu ikinci durumda, aktarım süresi bellek döngü süresine bağlıdır ve bu süre işlemci döngü süresinden daha fazla olabilir. Yukarıdaki denklemi şu şekilde yeniden yazabiliriz

$$T = I_c * [p + (m * k)] * t$$

Burada  $p$  kodunu çözmek ve yürütmek için gereken işlemci döngüsü sayısı,  $m$  gereken bellek referansı sayısı ve  $k$  bellek döngü süresi ile işlemci döngü süresi arasındaki orandır. Yukarıdaki denklemdeki beş performans faktörü ( $I_c, p, m, k, t$ ) dört sistem özelliğinden etkilenir: komut setinin tasarımı (*komut seti mimarisi* olarak bilinir); derleyici teknolojisi (derleyicinin yüksek seviyeli bir dil programından verimli bir makine dili programı üretmede ne kadar etkili olduğu); işlemci uygulaması; ve önbellek ve bellek hiyerarşisi. Tablo 2.1, bir boyutun beş performans faktörünü, diğer boyutun ise dört sistem özelliğini gösterdiği bir matristir. Bir hücredeki X işaretti, bir performans faktörünü etkileyen bir sistem özelliğini gösterir.

**Tablo 2.1** Performans Faktörleri ve Sistem Özellikleri

	$I_c$	$p$	$m$	$k$	
Komut seti mimarisi	X	X			
Derleyici teknolojisi	X	X	X		
İşlemci uygulaması		X			X
Önbellek ve bellek hiyerarşisi				X	X

Bir işlemci için yaygın bir performans ölçütü, saniyede milyonlarca talimat (MIPS) olarak ifade edilen ve **MIPS oranı** olarak adlandırılan talimatların yürütülme **hızıdır**. MIPS oranını saat hızı ve *CPI* cinsinden aşağıdaki gibi ifade edebiliriz:

$$\text{MIPS oranı} = \frac{\frac{I_c}{f}}{T * 10^6} = \frac{TÜFE * 10^6}{TÜFE * 10^6}$$
 (2.3)

**ÖRNEK 2.2** 400-MHz'lik bir işlemcide 2 milyon talimatın yürütülmesiyle sonuçlanan bir programın yürütülmesini düşünün. Program dört ana komut türünden oluşmaktadır. Her bir komut türü için komut karışımı ve *CPI*, bir program izleme deneyinin sonucuna dayalı olarak aşağıda verilmiştir:

Talimat Türü	TÜFE	Öğretim Karması (%)
Aritmetik ve mantık	1	60
Önbellek vuruşlu yükleme/depolama	2	18
Şube	4	12
Önbellek kaçırmalı bellek referansı	8	10

Program yukarıdaki izleme sonuçlarıyla tek işlemcili bir bilgisayarda çalıştırıldığında ortalama  $CPI = 0.6 + (2 * 0.18) + (4 * 0.12) + (8 * 0.1) = 2.24$ 'tür. Korelasyon-havuzlama MIPS oranı  $(400 * 10^6) / (2.24 * 10^6) \approx 178$  dir.

Diğer bir yaygın performans ölçütü sadece kayan noktalı talimatlarla ilgilidir. Bunlar birçok bilimsel ve oyun uygulamasında yaygındır. Kayan nokta performansı saniyede milyonlarca kayan nokta işlemi (MFLOPS) olarak ifade edilir ve aşağıdaki gibi tanımlanır:

$$\text{MFLOPS oranı} = \frac{\text{Bir programda yürütülen kayan nokta işlemlerinin sayısı}}{\text{Yürütme süresi} * 10^6}$$

## 2.5 ORTALAMANIN HESAPLANMASI

Bilgisayar sistemi performansının bazı yönlerini değerlendirdirken, performansı karakterize etmek ve sistemleri karşılaştırmak için genellikle yürütme süresi veya tüketilen bellek gibi tek bir sayı kullanılır. Açıkçası, tek bir sayı sistemin kapasitesinin yalnızca çok basitleştirilmiş bir görünümünü sağlayabilir. Yine de ve özellikle kriyaslama alanında, performans karşılaştırması için genellikle tek sayılar kullanılır [SMIT88].

Bölüm 2.6'da tartışıldığı gibi, sistemleri karşılaştırmak için kriyaslama ölçütlerinin kullanılması, yürütme süresiyle ilgili bir dizi veri noktasının ortalama değerinin hesaplanması içeriir. Ortalama değeri hesaplamak için kullanılabilecek birden fazla alternatif algoritma olduğu ortaya çıkmıştır ve bu durum

## 60 BÖLÜM 2 / PERFORMANS SORUNLARI

kiyaslama alanı. Bu bölümde, bu alternatif algoritmaları tanımlıyor ve bazı özellikleri hakkında yorum yapıyoruz. Bu bizi bir sonraki bölümde kiyaslamada ortalama hesaplama ile ilgili tartışmaya hazırlamaktadır.

Bir ortalamayı hesaplamak için kullanılan üç yaygın formül aritmetik, jeo-metrik ve harmoniktir. Bir dizi  $n$  gerçek sayı  $(x_1, x_2, \dots, x_n)$  verildiğinde, üç ortalama aşağıdaki gibi tanımlanır:

### Aritmetik ortalama

$$= \frac{x_1 + \cancel{x_2} + x_n}{n} = \frac{1}{n} \sum_{i=1}^n x_i \quad (2.4)$$

### Geometrik ortalama

$$GM = \sqrt[n]{x_1 \cdot \cancel{x_2} \cdot x_n} = \left( \prod_{i=1}^n x_i \right)^{1/n} = e^{\frac{1}{n} \sum_{i=1}^n \ln(x_i)} \quad (2.5)$$

### Harmonik ortalama

$$HM = \frac{n}{\frac{1}{x_1} + \cancel{\frac{1}{x_2}} + \frac{1}{x_n}} = \frac{n}{\sum_{i=1}^n \frac{1}{x_i}} \quad x \neq 0 \quad (2.6)$$

Aşağıdaki eşitsizliğin olduğu gösterilebilir:

$$AM \dots GM \dots HM$$

Değerler yalnızca  $x_1 = x_2 = \dots = x_n$  ise eşittir.

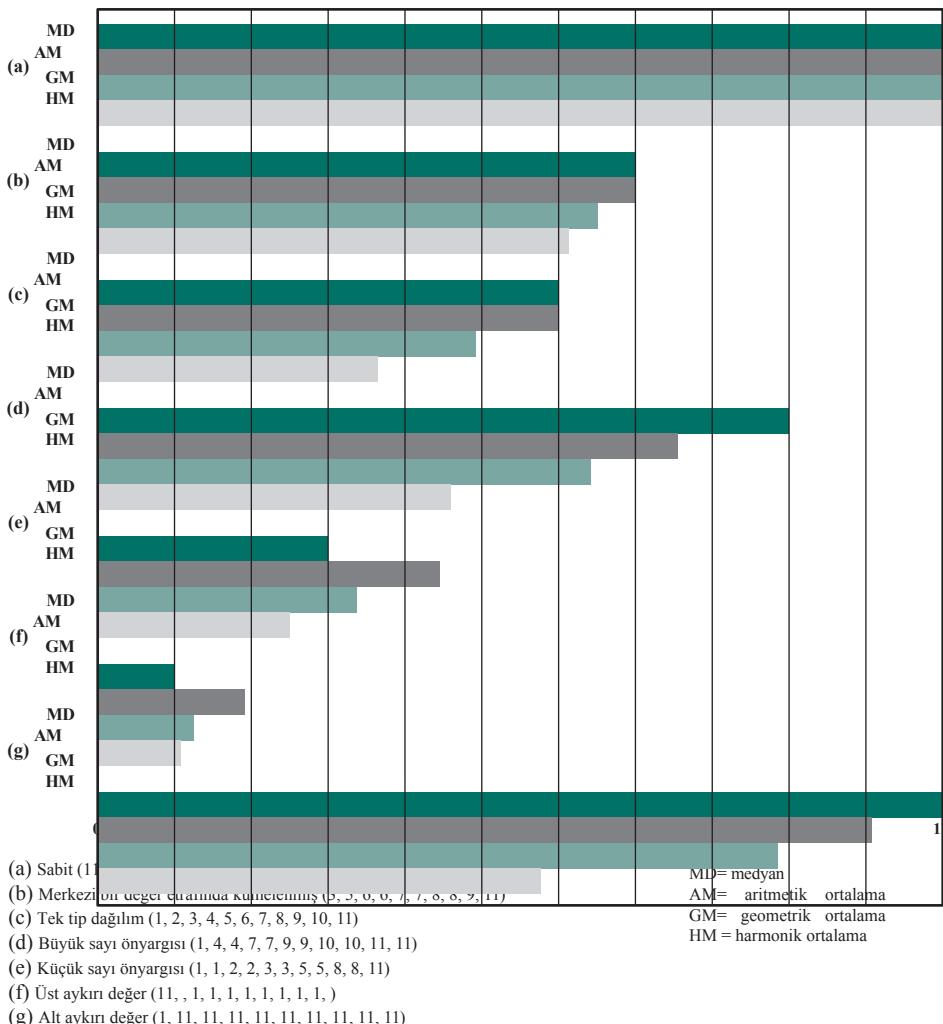
Fonksiyonel ortalamayı tanımlayarak bu alternatif hesaplamalar hakkında faydalı bir fikir edinebiliriz.  $f(x) \neq 0$  ...  $y \in \mathbb{R}$  aralığında tanımlanan sürekli monoton bir fonksiyon olsun.  $n$  pozitif reel sayı  $(x_1, x_2, \dots, x_n)$  için  $f(x)$  fonksiyonuna göre fonksiyonel ortalama şu şekilde tanımlanır

Fonksiyonel ortalama	$FM = f \left( \frac{-1(f(x_1) + \cancel{f(x_2)} + f(x_n))}{n} \right) = f \left( \frac{1}{n} \sum_{i=1}^n f(x_i) \right)$
-------------------------	--

Burada  $f^{-1}(x), f(x)$ 'in tersidir. Denklem (2.1) ile (2.3) arasında tanımlanan ortalama değerler, aşağıdaki gibi fonksiyonel ortalamanın özel durumlarıdır:

- $AM, f(x)$ 'e göre  $FM'$  dir  $= x$
- $GM, f(x)$ 'e göre  $FM'$  dir  $= \ln x$
- $HM, f(x)$ 'e göre  $FM'$  dir  $= 1/x$

**ÖRNEK 2.3** Şekil 2.6, her biri on bir veri noktasına ve 11 maksimum veri noktasına sahip çeşitli veri setlerine uygulanan üç ortalamayı göstermektedir. Medyan değeri de grafiğe dahil edilmiştir. Belki de bu şekilde en çok gözü çarpan şey, veriler daha büyük değerlere çarpık olduğunda veya küçük değerli bir aykırı değer olduğunda  $HM'$ 'nin yaniltıcı bir sonuç üretme eğiliminde olduğunu.



**Sekil 2.6** Çeşitli Veri Setleri Üzerindeki Ortalamaların Karşılaştırılması (her set 11 maksimum veri noktası değerine sahiptir)

Şimdi bu araçlardan hangilerinin belirli bir performans ölçütü için uygun olduğunu ele alalım. Bu açıklamalara bir önsöz olarak, bir dizi makalenin ([CITR06], FLEM86), [GILA95], [JACO95], JOHN04], MASH04], [SMIT88]) ve kitaplar ([HENN12], [HWAN93], [JAIN91], [LILJ00]) yıllar boyunca performans analizi için üç aracın artılarını ve eksilerini tartışmış ve çelişkili sonuçlara varmıştır. Karmaşık bir tartışmayı basitleştirmek için, varılan sonuçların büyük ölçüde seçilen örnekler ve hedeflerin belirtilme şékline bağlı olduğunu belirtmekle yetiniyoruz.

### Aritmetik Ortalama

Tüm ölçümlerin toplamı anlamlı ve ilginç bir değer ise AM uygun bir ölçütür. AM, birkaç sistemin yürütme süresi performansını karşılaştırmak için iyi bir adaydır. Örneğin, büyük ölçekli simülasyon çalışmaları için bir sistem kullanmak istediğimizi ve birkaç alternatif ürünü değerlendirmek istediğimizi varsayılm. Her sistemde simülasyonu her çalışma için farklı girdi değerleriyle birden çok kez çalıştırılabilir ve ardından tüm çalışırmalarda ortalama yürütme süresini alabiliriz. Farklı girdilerle birden fazla çalışırmayı kullanılması, sonuçların belirli bir girdi setinin olağandışı bir özelliği tarafından büyük ölçüde önyargılı olmamasını sağlamalıdır. Tüm çalışırmaların AM değeri, sistemin simülasyonlar üzerindeki performansının iyi bir ölçüsüdür ve sistem karşılaştırması için kullanılabilecek iyi bir sayıdır. Program yürütme süresi gibi zamana dayalı bir değişken (örn. saniye) için kullanılan AM, toplam süre ile doğru orantılı olması gibi önemli bir özelliğe sahiptir. Zaman. Yani, toplam süre iki katına çıkarsa, ortalama değer de iki katına çıkar.

### Harmonik Ortalama

Bazı durumlar için, bir sistemin yürütme hızı, sistemin değerinin daha kullanışlı bir ölçütü olarak görülebilir. Bu, MIPS veya MFLOPS olarak ölçülen komut yürütme hızı veya belirli bir program türünün yürütülebilme hızını ölçen bir program yürütme hızı olabilir. Hesaplanan ortalamanın nasıl davranışmasını istediğimizi düşünün. Ortalama oranın toplam oranın orantılı olmasını istediğimizi söylemenin bir anlamı yoktur, burada toplam oran bireysel oranların toplamı olarak tanımlanır. Oranların toplamı anlamsız bir istatistik olacaktır. Bunun yerine, ortalamanın toplam yürütme süresiyle ters orantılı olmasını isteriz. Örneğin, bir program paketindeki tüm kıyaslama programlarını yürütmek için gereken toplam süre C sistemi için D sisteminin iki katı ise, yürütme oranının ortalama değerinin C sistemi için D sisteminin yarısı kadar olmasını isteriz.

Temel bir örneğe bakalım ve ilk olarak AM'nin nasıl performans gösterdiğini inceleyelim. Elimizde  $n$  adet kıyaslama programı olduğunu varsayılm ve her bir programın belirli bir sistem üzerindeki yürütme sürelerini  $t_1, t_2, \dots, t_n$  olarak kaydedelim. Basitlik açısından, her programın aynı sayıda  $Z$  işlemi yürüttüğünü varsayılm; tek tek programları ağırlıklandırabilir ve buna göre hesaplayabiliz, ancak bu argümanımızın sonucunu değiştirmez. Her bir program için yürütme oranı  $R_i = Z/t_{(i)}$ dir. Ortalama yürütme oranını hesaplamak için AM'yi kullanırız.

$$AM = \frac{1}{n} \sum_{i=1}^n R_i = \frac{1}{n} \sum_{i=1}^n \frac{Z}{t_i} = \frac{Z}{n} \sum_{i=1}^n \frac{1}{t_i}$$

AM yürütme oranının ters yürütme sürelerinin toplamıyla orantılı olduğunu görüyoruz ki bu, yürütme sürelerinin toplamıyla ters orantılı olmakla aynı şey değildir. Dolayısıyla, AM istenen özelliğe sahip değildir.

HM aşağıdaki sonucu verir.

$$HM = \frac{n}{\sum_{i=1}^n \left(\frac{1}{R_i}\right)} = \frac{n}{\sum_{i=1}^n \left(\frac{1}{Z/t_i}\right)} = \frac{n}{\sum_{i=1}^n t_i}$$

HM, istenen özellik olan toplam yürütme süresiyle ters orantılıdır.

**ÖRNEK 2.4** Basit bir sayısal örnek, Tablo 2.2'de gösterilen oranların ortalama değerinin hesaplanmasıında iki ortalama arasındaki farkı gösterecektir. Tablo, iki programın yürütülmesinde üç bilgisayarın performansını karşılaştırmaktadır. Basitlik açısından, her bir programın yürütülmesinin  $10^8$  kayan noktalı işlemin yürütülmüşle sonuçlandığını varsayıyoruz. Tablonun sol yarısında, her bir programı çalıştırın her bir bilgisayar için yürütme süreleri, toplam yürütme süresi ve yürütme sürelerinin AM'si gösterilmektedir. A bilgisayar, C'den daha az toplam sürede çalışan B'den daha az toplam sürede ve bu AM'ye doğru bir şekilde yansımaktadır.

Tablonun sağ yarısı, MFLOPS cinsinden ifade edilen oranlar açısından bir karşılaşturma sağlar. Oran hesaplaması basittir. Örneğin, program 1 100 milyon kayan nokta işlemi gerçekleştirir. A bilgisayarının programı yürütmesi  $100/2 = 50$  MFLOPS oranı için 2 saniye sürer. Daha sonra, oranların AM'sini düşünün. En büyük değer A bilgisayarı içindir, bu da A'nın en hızlı bilgisayar olduğunu gösterir. Toplam çalışma süresi açısından A en az süreye sahiptir, bu nedenle üç bilgisayar arasında en hızlıdır. Ancak AM oranları B'yi C'den daha yavaş gösterir, oysa aslında B'C'den daha hızlıdır. HM değerlerine baktığımızda, bilgisayarların hız sıralamasını doğru bir şekilde yansıtıklarını görüyoruz. Bu da hızlar hesaplanırken HM'nin tercih edildiğini doğrulamaktadır.

Okuyucu neden bu kadar uğraştığımızı merak edebilir. Eğer yürütme sürelerini karşılaştırmak istiyorsak, basitçe üç sistemin toplam yürütme sürelerini karşılaştırabiliriz. Oranları karşılaştırmak istiyorsak, tabloda gösterildiği gibi toplam yürütme süresinin tersini alabiliriz. Sadece toplam sayılar bakmak yerine tek tek hesaplamalar yapmanın iki nedeni vardır:

**Tablo 2.2** Oranlar için Aritmetik ve Harmonik Ortalamaların Karşılaştırılması

	Bilgisayar A zamanı (saniye)	Bilgisayar B zamanı (sn)	Bilgisayar C süresi (sn)	Bilgisayar A oranı (MFLOPS)	Bilgisayar B oranı (MFLOPS)	Bilgisayar C oranı (MFLOPS)
Program 1 ( $10^8$ FP operasyonları)	2.0	1.0	0.75	50	100	133.33
Program 2 ( $10^8$ FP operasyonları)	0.75	2.0	4.0	133.33	50	25
Toplam yürütme süresi	2.75	3.0	4.75	-	-	-
Zamanların aritmetik ortalaması	1.38	1.5	2.38	-	-	-
Toplam uygulamamın tersi zaman (1/sn)	0.36	0.33	0.21	-	-	-
Oranların aritmetik ortalaması	-	-	-	91.67	75.00	79.17
Oranların harmonik ortalaması	-	-	-	72.72	66.67	42.11

1. Bir müşteri veya araştırmacı sadece genel ortalama performansla değil, aynı zamanda iş uygulamaları, bilimsel modelleme, multimedya uygulamaları ve sistem programları gibi farklı kıyaslama türlerine karşı performansla da ilgilenebilir. Bu nedenle, toplamın yanı sıra kıyaslama türüne göre bir döküm de gereklidir.
2. Genellikle, değerlendirme için kullanılan farklı programlar farklı şekilde ağırlıklandırılır. Tablo 2.2'de, iki test programının aynı sayıda işlem gerçekleştirdiği varsayılmaktadır. Eğer durum böyle değilse, buna göre ağırlıklandırma yapmak isteyebiliriz. Ya da farklı programlar, önem veya önceliği yansıtacak şekilde farklı ağırlıklarıdırabilir.

Test programları işlem sayısıyla orantılı olarak ağırlıklandırılsa sonucun ne olacağını görelim. Önceki gösterimi takiben, her  $i$  programı bir  $t_i$  zamanında  $Z_i$  talimatlarını yürütür. Her oran talimat sayısına göre ağırlıklarıdırılır. Bu nedenle ağırlıklı HM şöyledir:

$$WHM = \frac{1}{\sum_{i=1}^n \frac{Z_i}{(\sqrt{\sum_j^n Z_j})^R}} = \frac{n}{\sum_{i=1}^n \frac{Z_i}{t_i}} = \frac{\sum_{j=1}^n Z_j}{\sum_{i=1}^n t_i} \quad (2.7)$$

Ağırlıklı HM'nin işlem sayısı toplamının yürütme süreleri toplamına bölümü olduğunu görüyoruz.

### Geometrik Ortalama

Üç tür araç için denklemlere bakıldığında, AM ve HM'nin davranışı hakkında sezgisel bir fikir edinmek GM'ninkinden daha kolaydır. FEIT15]'ten birkaç gözlem bu konuda yardımcı olabilir. İlk olarak, değerlerdeki değişikliklerle ilgili olarak, GM'nin veri setindeki tüm değerler eşit ağırlık verdienenini not ediyoruz. Örneğin, ortalaması alınacak veri değerleri kümesinin birkaç büyük ve daha fazla küçük değer içerdigini varsayıyalım. Burada, AM büyük değerler tarafından domine edilir. En büyük değerde %10'luk bir değişiklik fark edilebilir bir etkiye sahip olurken, en küçük değerde aynı faktörde bir değişiklik ihmali edilebilir bir etkiye sahip olacaktır. Buna karşılık, veri değerlerinden herhangi birinde %10'luk bir değer değişikliği GM'de aynı değişikliği neden olur: 2.1.

ÖRNEK		Geometrik Ortalama	Aritmetik Ortalama
veya minimum ve maksimum değerlerini değiştirmek	Orijinal değer	3,37	4,45
	Maksimum değeri 11'den 12,1'e yükseltin (+ %10)	3,40 (+ %0,87)	4,55 (+ %2,24)
	Min değerini 1'den 1,1'e yükseltin (+ %10)	3,40 (+ %0,87)	4,46 (+ %0,20)

İkinci bir gözlem ise, bir oranın GM'si için, oranların GM'sinin GM'lerin oranına eşit olduğunu:

$$GM = \left( M \frac{\sum_{i=1}^n Z_i}{\sum_{i=1}^n t_i} \right)^{1/n} = \frac{\left( \prod_{i=1}^n Z_i \right)^{1/n}}{\left( \prod_{i=1}^n t_i \right)^{1/n}} \quad (2.8)$$

Bunu Denklem 2.4 ile karşılaştırın.

Oranların aksine yürütme süreleriyle kullanım için GM'nin bir dezavantajı, daha sezgisel olan AM'ye göre monotonik olmayabilmesidir. Başka bir deyişle, bir veri setinin AM'sinin başka bir setinkinden daha büyük olduğu, ancak GM'nin daha küçük olduğu durumlar olabilir.

#### ÖRNEK 2.6

bunun tersi ge-

Şekil 2.6'da, d veri seti için AM, c veri seti için AM'den daha büyüktür, ancak GM için

	Veri seti c	Veri seti d
Aritmetik ortalama	7.00	7.55
Geometrik ortalama	6.68	6.42

GM'nin kıyaslama analizi için cazip olmasını sağlayan bir özelliği, makinelerin göreceli performansını ölçerken tutarlı sonuçlar vermesidir. Aslında kıyaslamalar esas olarak bunun için kullanılır: bir makineyi diğeryle performans ölçütleri açısından karşılaştırmak. Gördüğümüz gibi sonuçlar, bir referans makineye göre normalize edilmiş değerler cinsinden ifade edilir.

#### ÖRNEK 2.7

Basit bir örnek, GM'nin normalleştirilmiş sonuçlar için nasıl tutarlılık sergilediğini göstererektir. Tablo 2.3'te, Tablo 2.2'de kullanılan performans sonuçlarının aynısını kullanıyoruz. Tablo 2.3'a da tüm sonuçlar A bilgisayarına göre normalize edilmiş ve ortalamalar normalize edilmiş değerler üzerinden hesaplanmıştır. Toplam yürütme süresine göre A, C'den daha hızlı olan B'den daha hızlıdır. Normalleştirilmiş sürelerin hem AM'leri hem de GM'leri bunu yansımaktadır. Tablo 2.3b'de, sistemler şimdi B'ye normalize edilmiştir. GM'ler yine üç bilgisayarın birbiriley ilişkili hızlarını doğru bir şekilde yansımaktadır, ancak şimdi AM farklı bir sıralama üretmektedir.

Ne yazık ki, tutarlılık her zaman doğru sonuçlar üretmez. Tablo 2.4'te bazı yürütme süreleri değiştirilmiştir. Bir kez daha, AM iki normalleştirme için çelişkili sonuçlar bildirmektedir. GM tutarlı sonuçlar bildirir, ancak sonuç B'nin eşit olan A ve C'den daha hızlı olduğunu.

Daha önce listelenen alıntınlarda "kıyaslama ölçüyü savaşlarını" körükleyen bu gibi örneklerdir. Tek bir sayının, sistemler arasında performans karşılaştırması yapmak için ihtiyaç duyulan tüm bilgileri sağlayamayacağını söyleyebiliriz. Ancak,

## 66 BÖLÜM 2 / PERFORMANS SORUNLARI

**Tablo 2.3** Normalleştirilmiş Sonuçlar için Aritmetik ve Geometrik Ortalamaların Karşılaştırılması

(a) Bilgisayar A'ya göre normalleştirilmiş sonuçlar

	Bilgisayar A zamanı	Bilgisayar B zamanı	Bilgisayar C zamanı
Program 1	2.0 (1.0)	1.0 (0.5)	0.75 (0.38)
Program 2	0.75 (1.0)	2.0 (2.67)	4.0 (5.33)
Toplam yürütme süresi	2.75	3.0	4.75
Normalize edilmiş sürelerin aritmetik ortalaması	1.00	1.58	2.85
Normalize edilmiş sürelerin geometrik ortalaması	1.00	1.15	1.41

(b) Bilgisayar B'ye normalize edilmiş sonuçlar

	Bilgisayar A zamanı	Bilgisayar B zamanı	Bilgisayar C zamanı
Program 1	2.0 (2.0)	1.0 (1.0)	0.75 (0.75)
Program 2	0.75 (0.38)	2.0 (1.0)	4.0 (2.0)
Toplam yürütme süresi	2.75	3.0	4.75
Normalize edilmiş sürelerin aritmetik ortalaması	1.19	1.00	1.38
Normalize edilmiş sürelerin geometrik ortalaması	0.87	1.00	1.22

**Tablo 2.4** Normalleştirilmiş Sonuçlar için Aritmetik ve Geometrik Ortalamaların Bir Başka Karşılaştırması

(a) Bilgisayar A'ya göre normalleştirilmiş sonuçlar

	Bilgisayar A zamanı	Bilgisayar B zamanı	Bilgisayar C zamanı
Program 1	2.0 (1.0)	1.0 (0.5)	0.20 (0.1)
Program 2	0.4 (1.0)	2.0 (5.0)	4.0 (10.0)
Toplam yürütme süresi	2.4	3.00	4.2
Normalize edilmiş sürelerin aritmetik ortalaması	1.00	2.75	5.05
Normalize edilmiş sürelerin geometrik ortalaması	1.00	1.58	1.00

(b) Bilgisayar B'ye normalize edilmiş sonuçlar

	Bilgisayar A zamanı	Bilgisayar B zamanı	Bilgisayar C zamanı
Program 1	2.0 (2.0)	1.0 (1.0)	0.20 (0.2)
Program 2	0.4 (0.2)	2.0 (1.0)	4.0 (2.0)
Toplam yürütme süresi	2.4	3.00	4.2
Normalize edilmiş sürelerin aritmetik ortalaması	1.10	1.00	1.10
Normalize edilmiş sürelerin geometrik ortalaması	0.63	1.00	0.63

Literatürdeki çelişkili görüşlere rağmen, SPEC çeşitli nedenlerden dolayı GM'yi kullanmayı tercih etmiştir:

1. Belirtildiği gibi, GM hangi sistemin referans olarak kullanıldığından bağımsız olarak tutarlı sonuçlar verir. Kiyaslama öncelikle bir karşılaştırma analizi olduğundan, bu önemli bir özellikleştir.
2. MCMA93]'te belgelendiği ve SPEC analistleri MASH04] tarafından daha sonra yapılan analizlerde doğrulandığı üzere GM, HM veya AM'ye kıyasla aykırı değerlerden daha az etkilenmektedir.
3. [MASH04], normalleştirilmiş sayıların genel olarak çarpık dağılımı nedeniyle performans oranlarının dağılımlarının normal dağılımlardan ziyade lognormal dağılımlarla daha iyi modellendiğini göstermektedir. Bu durum [CITR06]'da da doğrulanmıştır. Ve Denklem (2.5)'te gösterildiği gibi GM, lognormal dağılımin geri dönüştürülmüş ortalaması olarak tanımlanabilir.

## 2.6 tezgahlar ve özellikler

### Benchmark İlkeleri

MIPS ve MFLOPS gibi ölçütlerin işlemcilerin performansını değerlendirmede yetersiz kaldığı kanıtlanmıştır. Komut kümelerindeki farklılıklar nedeniyle, komut yürütme oranı farklı mimarilerin performansını karşılaştırmak için geçerli bir araç değildir.

**ÖRNEK 2.8** Bu üst düzey dil ifadesini göz önünde bulundurun:

```
A= B+ C      /* tüm miktarların ana bellekte olduğunu varsayı */
```

Karmaşık komut seti bilgisayarı (CISC) olarak adlandırılan geleneksel bir komut seti mimarisi ile bu komut tek bir işlemci komutuna derlenebilir:

```
ekle    mem(B(), (C(), mem(A()
```

Tipik bir RISC makinesinde, derleme gibi görünecektir:

```
mem(B(), reg(1) yükleyin;
mem(C(), reg(2) yükleyin;
ekle    reg(1(), (2(), reg(3());
reg(3()'ü sakla, mem(A()
```

RISC mimarisinin doğası gereği (Bölüm 15'te tartışılmıştır), her iki makine de orijinal üst düzey dil komutunu yaklaşık aynı sürede yürütübilir. Eğer bu örnek iki makineyi temsil ediyorsa, CISC makine 1 MIPS, RISC makine 4 MIPS değerinde olacaktır. Ancak her ikisi de aynı miktarda yüksek seviyeli dil işini aynı sürede yapar.

Bir başka husus da, belirli bir işlemcinin belirli bir program üzerindeki performansının, o işlemcinin çok farklı bir uygulama türünde nasıl performans göstereceğini belirlemeye yararlı olmayacağıdır. Buna bağlı olarak, 1980'lerin sonu ve 1990'lارın başından itibaren, endüstri ve akademik ilgi işlemcilerin performansını ölçmeye yönelmiştir.

bir dizi kıyaslama programı kullanan sistemler. Aynı program seti farklı makinelerde çalıştırılabilir ve yürütme süreleri karşılaştırılabilir. Kıyaslama programları, hangi sistemi satın alacaklarına karar vermeye çalışan müşterilere rehberlik eder ve kıyaslama hedeflerini karşılamak için sistemlerin nasıl tasarlanacağını belirlemeye yardım eder ve tasarımcılar için yararlı olabilir. [WEIC90] bir kıyaslama programının arzu edilen özellikleri olarak aşağıdakileri listelemektedir:

- 1.** Yüksek seviyeli bir dilde yazıldığı için farklı makineler arasında taşınabilir.
- 2.** Sistem, sayısal programlama veya ticari programlama gibi belirli bir programlama alanını veya paradigmاسını temsil eder.
- 3.** Kolayca ölçülebilir.
- 4.** Geniş bir dağıtıma sahiptir.

### SPEC Benchmarkları

Endüstride ve akademik ve araştırma topluluklarında genel kabul görmüş bilgisayar performans ölçümüne duyan ortak ihtiyaç, standartlaştırılmış kıyaslama geliştirilmesine yol açmıştır. Bir kıyaslama paketi, belirli bir uygulama veya sistem programlama alanında bir bilgisayarın temsili bir testini sağlamaya çalışır, yüksek seviyeli bir dilde tanımlanmış programlar topluluğudur. Bu tür kıyaslama paketlerinin en iyi bilineni, bir endüstri konsorsiyumu olan Standart Performans Değerlendirme Kuruluşu (SPEC) tarafından tanımlanmaktadır ve sürdürülmektedir. Bu kuruluş, bilgisayar sistemlerini değerlendirmeyi amaçlayan çeşitli kıyaslama paketleri tanımlamaktadır. SPEC performans ölçümleri karşılaştırma ve araştırma amacıyla yaygın olarak kullanılmaktadır.

SPEC kıyaslama paketleri arasında en iyi bilineni SPEC CPU2006'dır. Bu, işlemci yoğun uygulamalar için endüstri standartı paketidir. SPEC CPU2006, zamanının çoğunu I/O yerine hesaplama yaparak geçiren uygulamaların performansını ölçmek için uygundur.

Diğer SPEC süttleri arasında aşağıdakiler yer almaktadır:

- **SPECviewperf:** Profesyonel uygulamalara dayalı 3D grafik performansını ölçmek için standart.
- **SPECwpc:** medya ve eğlence, ürün geliştirme, yaşam bilimleri, finansal hizmetler ve enerji dahil olmak üzere çeşitli profesyonel uygulamalara dayalı olarak iş istasyonu performansının tüm önemli yönlerini ölçmek için karşılaştırma ölçütü.
- **SPECjvm2008:** Java Sanal Makinesi (JVM) istemci platformunun birleşik donanım ve yazılım yönlerinin performansını değerlendirmek için tasarlanmıştır.
- **SPECjbb2013 (Java Business Benchmark):** Hizmet tarafı Java tabanlı elektronik ticaret uygulamalarını değerlendirmek için bir ölçüt.
- **SPECcsfs2008:** Dosya sunucularının hızını ve istek işleme kapasitesini değerlendirmek için tasarlanmıştır.
- **SPECvirt\_sc2013:** Virtüelleştirilmiş sunucu konsolidasyonunda kullanılan veri merkezi sunucularının performans değerlendirmesi. Donanım, sanallaştırma platformu ve sanallaştırılmış konuk işletim sistemi ve uygulama yazılımı dahil olmak üzere tüm sistem bileşenlerinin uçtan uca performansını ölçer. Kıyaslama, donanım sanallaştırma, işletim sistemi sanallaştırma ve sabit donanım bölümeşemalarını destekler.

CPU2006 paketi, SPEC endüstri üyeleri tarafından çok çeşitli platformlara taşınmış olan mevcut uygulamalara dayanmaktadır. Benchmark sonuçlarını güvenilir ve gerçekçi kılmak için, CPU2006 benchmarkları yapay döngü programları veya sentetik benchmarklar kullanmak yerine gerçek hayatı uygulamalarından almıştır. Paket, C ve C++ dillerinde yazılmış 12 tamsayı karşılaştırma ölübü ve C, C++ ve Fortran dillerinde yazılmış 17 kayan nokta karşılaştırma ölübünden oluşmaktadır (Tablo 2.5 ve 2.6). Paket 3 milyon satırдан fazla kod içermektedir. Bu beşinci nesil

**Tablo 2.5** SPEC CPU2006 Tamsayı Benchmarkları

Benchmark	Referans süresi (saat)	Enstrüman sayısı (milyar)	Dil	Uygulama Alanı	Kısa Açıklama
400.perlbench	2.71	2378	C	Programlama Dili	PERL programlama dili yorumlayıcı, üç programdan oluşan bir kümeye uygulanmıştır.
401.bzip2	2.68	2472	C	Sıkıştırma	I/O yapmak yerine çoğu işin bellekte yapıldığı genel amaçlı veri sıkıştırma.
403.gcc	2.24	1064	C	C Derleyici	Gcc Sürüm 3.2'yi temel alır, Opteron için kod üretir.
429.mcf	2.53	327	C	Kombinatoryal Optimizasyon	Araç çizelgeleme algoritması.
445.gobmk	2.91	1603	C	Yapay Zeka	Basitçe tanımlanmış ancak son derece karmaşık bir oyun olan Go oyununu oynar.
456.hmmmer	2.59	3363	C	Gen Dizisi Arama	Profil-gizli Markov modelleri kullanarak protein dizisi analizi.
458.sjeng	3.36	2383	C	Yapay Zeka	Çeşitli satranç varyantlarını da oynatan yüksek dereceli bir satranç programı.
462.libquantum	5.76	3555	C	Fizik / Kuantum Hesaplama	Shor'un polinom zamanlı faktörleştirme algoritmasını çalıştan bir kuantum bilgisayarı simüle eder.
464.h264ref	6.15	3731	C	Video Sıkıştırma	H.264/AVC (Gelişmiş Video Kodlama) video sıkıştırma.
471.omnetpp	1.74	687	C++	Ayrık Olay Simülasyonu	Büyük bir Ethernet kampüs ağını modellemek için OMNet++ ayrık olay simülörünü kullanır.
473.astar	1.95	1200	C++	Yol Bulma Algoritmaları	2B haritalar için yol bulma kütüphanesi.
483.xalancbmk	1.92	1184	C++	XML İşleme	XML belgelerini diğer belge türlerine dönüştüren Xalan-C++'ın geliştirilmiş bir sürümü.

## 70 BÖLÜM 2 / PERFORMANS SORUNLARI

**Tablo 2.6** SPEC CPU2006 Kayan Nokta Benchmarkları

Benchmark	Referans süresi (saat)	Enstrüman sayısı (milyar)	Dil	Uygulama Alanı	Kısa Açıklama
410.bwaves	3.78	1176	Fortran	Akışkanlar Dinamigi	3D transonik geçici laminer viskoz akışı hesaplar.
416.gamess	5.44	5189	Fortran	Kuantum Kimyası	Kuantum kimyasal hesaplama
433.milc	2.55	937	C	Fizik / Kuantum Kromodinamigi	Kuarkların ve gluonların davranışını simüle eder.
434.zeusmp	2.53	1566	Fortran	Fizik / CFD	Astrofiziksel olayların hesaplamalı akışkanlar dinamigi simülasyonu.
435.gromacs	1.98	1958	C, Fortran	Biyokimya / Moleküler Dinamik	Yüzlerce ila milyonlarca parçacık için Newton hareket denklemlerini simüle eder.
436.cactusADM	3.32	1376	C, Fortran	Fizik / Genel Görelilik	Einstein evrim denklemlerini çözer.
437.leslie3d	2.61	1273	Fortran	Akışkanlar Dinamigi	Yakıt enjeksiyon akışlarını modeller.
444.namd	2.23	2483	C++	Biyoloji / Moleküler Dinamik	Büyük biyomoleküler sistemleri simüle eder.
447.dealII	3.18	2323	C++	Sonlu Elemanlar Analizi	Uyarlanabilir sonlu elemanlar ve hata təmininə yönelik program kütüphanesi.
450.soplex	2.32	703	C++	Doğrusal Programlama, Optimizasyon	Test vakaları arasında demiryolu planlaması ve askeri hava ikmal modelleri yer almaktadır.
453.povray	1.48	940	C++	Resim Işın İzleme	3D görüntü oluşturma.
454.calculix	2.29	3,04	C, Fortran	Yapısal Mekanik	Doğrusal ve doğrusal olmayan 3D yapısal uygulamalar için sonlu elemanlar kodu.
459.GemsFDTD	2.95	1320	Fortran	Hesaplamalı Elektromanyetik	Maxwell denklemlerini 3 boyutlu olarak çözer.
465.tonto	2.73	2392	Fortran	Kuantum Kimyası	Kristal grafik görevleri için uyarlanmış kuantum kimyası paketi.
470.lbm	3.82	1500	C	Akışkanlar Dinamigi	Sıkıştırılamaz akışkanları 3D olarak simüle eder.
481.wrf	3.10	1684	C, Fortran	Hava Durumu	Hava tahmin modeli.
482.sphinx3	5.41	2472	C	Konuşma Tanıma	Konuşma tanıma yazılımı.

SPEC CPU2000, SPEC CPU95, SPEC CPU92 ve SPEC CPU89'un yerini alan SPEC'in işlemci yoğun paketleri [HENN07].

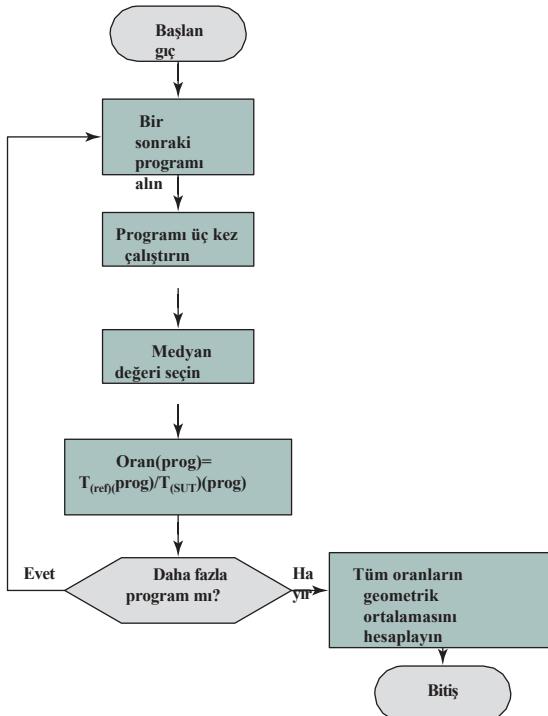
CPU2006 kullanan bir sistemin yayınlanan sonuçlarını daha iyi anlamak için SPEC belgelerinde kullanılan aşağıdaki terimleri tanımlıyoruz:

- **Benchmark:** Derleyiciyi uygulayan herhangi bir bilgisayarda derlenebilen ve çalıştırılabilen yüksek seviyeli bir dilde yazılmış bir program.
- **Test altındaki sistem:** Bu, değerlendirilecek olan sistemdir.
- **Referans makine:** Bu, SPEC tarafından tüm kıyaslamalar için bir temel performans oluşturmak için kullanılan bir sistemdir. Her bir kıyaslama bu makinede çalıştırılır ve ölçülecek o kıyaslama için bir referans zaman belirlenir. Test edilen bir sistem, CPU2006 kıyaslamalarını çalıştırarak ve aynı programları referans makinede çalışma sonuçlarını karşılaştırarak değerlendirilir.
- **Temel metrik:** Bunlar raporlanan tüm sonuçlar için gereklidir ve derleme için katı kılavuz çizgileri vardır. Temelde, karşılaştırılabilir sonuçlar elde etmek için test edilen her sistemde az çok varsayılan ayarlara sahip standart derleyici kullanılmalıdır.
- **Peak metric:** Bu, kullanıcıların derleyici çıktısını optimize ederek sistem performansını optimize etmeye çalışmasını sağlar. Örneğin, her kıyaslamada farklı derleyici seçenekleri kullanılabilir ve geri bildirime yönelik optimizasyona izin verilir.
- **Hız ölçütü:** Bu basitçe derlenmiş bir kıyaslamayı yürütmemek için gereken sürenin bir ölçümüdür. Hız metriği, bir bilgisayarın tek bir görevi tamamlama yeteneğini karşılaştırmak için kullanılır.
- **Hız ölçütü:** Bu, bir bilgisayarın belirli bir içinde kaç görevi yerine getirebileceğinin bir ölçümüdür; buna **verim**, kapasite veya hız ölçütü denir. Hız ölçütü, test edilen sistemin birden fazla işlemciden yararlanmak için eşzamanlı görevleri yürütmesine olanak tanır.

SPEC, referans makine olarak 1997'de tanıtılan tarihi bir Sun sistemi olan "Ultra Enterprise 2"yi kullanmaktadır. Referans makine 296-MHz UltraSPARC II işlemci kullanmaktadır. CPU2006 referans makinesinde CINT2006 ve CFP2006 için temel metriklerin kurallara uygun bir şekilde çalıştırılması yaklaşık 12 gün sürmektedir. Tablo 2.5 ve 2.6 referans makineyi kullanarak her bir kıyaslamayı çalıştmak için gereken süreyi göstermektedir. Tablolar ayrıca [PHAN07]'de bildirildiği gibi referans makinedeki dinamik komut sayılarını da göstermektedir. Bu değerler, her bir programın çalıştırılması sırasında yürütülen gerçek talimat sayısıdır.

Şimdi bir sistemi değerlendirmek için yapılan özel hesaplamaları ele alıyoruz. Tamsayı ölçütlerini ele alıyoruz; kayan nokta ölçüt değeri oluşturmak için de aynı prosedürler kullanılmaktadır. Tamsayı ölçütleri için test paketinde 12 program bulunmaktadır. Hesaplama üç aşamalı bir süreçtir (Şekil 2.7):

1. Test edilen bir sistemi değerlendirmenin ilk adımı, her bir programı derlemek ve sistem üzerinde üç kez çalıştmaktır. Her program için çalışma süresi ölçülür ve medyan değer seçilir. Üç çalışma kullanmanın ve medyan değeri alınan nedeni, disk erişim süresi değişiklikleri ve bir çalıştırmadan diğerine işletim sistemi çekirdeği çalışma değişiklikleri gibi programın içinde olmayan süresindeki değişiklikleri hesaba katmaktadır.



Şekil 2.7 SPEC Değerlendirme Açış Şeması

2. Ardından, 12 sonucun her biri, referans çalışma süresinin sistem çalışma süresine oran hesaplanarak normalleştirilir. Oran aşağıdaki şekilde hesaplanır:

$$\frac{T_{ref_i}}{T_{sut_i}} = r_i \quad (2.9)$$

Burada  $T_{ref_i}$  referans sistemindeki kıyaslama programı  $i$ 'nin yürütme süresi ve  $T_{sut_i}$  test edilen sistemindeki kıyaslama programı  $i$ 'nin yürütme süresidir. Dolayısıyla, oranlar daha hızlı makineler için daha yüksektir.

3. Son , genel metriği elde etmek için 12 çalışma süresi oranının geometrik ortalaması hesaplanır:

$$r_G = \left( \prod_{i=1}^{12} r_i \right)^{1/12}$$

Tamsayı kıyaslamaları için dört ayrı metrik hesaplanabilir:

- **SPECint2006:** Karşılaştırma işaretleri tepe ayarıyla derlendiğinde 12 normalleştirilmiş oranın geometrik ortalaması.
- **SPECint\_base2006:** Kıyaslamalar temel ayarlama ile derlendiğinde 12 normalleştirilmiş oranın geometrik ortalaması.
- **SPECint\_rate2006:** Kıyaslama ölçütleri en yüksek ayarlama ile derlendiğinde 12 normalleştirilmiş verim oranının geometrik ortalaması.
- **SPECint\_rate\_base2006:** Kıyaslamalar temel ayarlama ile derlendiğinde 12 normalleştirilmiş verim oranının geometrik ortalaması.

**ÖRNEK 2.9** Sun Blade 1000 için sonuçlar Tablo 2.7a'da gösterilmektedir. SPEC CPU2006 tamsayı karşılaştırma ölçütlerinden biri 464.h264ref'tir. Bu, en son teknoloji video sıkıştırma standarı olan H.264/ AVC'nin (Gelişmiş Video Kodlama) bir referans uygulamasıdır. Sun Blade 1000 bu programı ortalama 5,259 saniyede çalıştırmaktadır. Referans uygulama 22.130 saniye gerektirmektedir. Oran şu şekilde hesaplanır:  $22.130/5.259 = 4.21$ . Hız metriği, oranların çarpımının on ikinci kökü alınarak hesaplanır:

$$(3.18 * 2.96 * 2.98 * 3.91 * 3.17 * 3.61 * .51 * .01 *$$

$$4.21 * 2.43 * 2.75 * 3.42)^{1/12} = 3.12$$

Hız ölçümleri birden fazla işlemciye sahip bir sistemi dikkate alır. Bir makineyi test etmek için bir  $N$  kopya sayısı seçilir; bu genellikle işlemci sayısına veya test sistemindeki eşzamanlı yürütme iş parçacığı sayısına eşittir. Her bir test programının oranı, üç çalıştırmanın ortalaması alınarak belirlenir. Her çalışma, test sisteminde aynı anda çalışan programın  $N$  kopyasından oluşur. Yürütme süresi, tüm kopyaların bitmesi için geçen süredir (yani, ilk kopyanın başladığı andan son kopyanın bittiği ana kadar geçen süre). Bu program için hız metriği aşağıdaki formülle hesaplanır:

$$oran_i = N * \frac{Tref_i}{Tsut_i}$$

Test edilen sistem için oran puanı, test paketindeki her program için oranların geometrik ortalamasından belirlenir.

**ÖRNEK 2.10** Sun Blade X6250 için sonuçlar Tablo 2.7b'de gösterilmektedir. Bu sisteme iki işlemci yongası ve her yongada iki çekirdek olmak üzere toplam dört bulunmaktadır. Hız ölçütünü elde etmek için, her kıyaslama programı dört çekirdeğin tümünde aynı anda yürütülür ve yürütme süresi dört kopyanın başlangıcından en yavaş çalışmanın sonuna kadar geçen süredir. Hız oranı daha önce olduğu gibi hesaplanır ve hız değeri basitçe hız oranının dört katıdır. Nihai hız metriği, hız değerlerinin geometrik ortalaması alınarak bulunur:

$$(78.63 * 62.97 * 60.87 * 77.29 * 65.87 * 83.68 * 76.70 * 134.98 * \\ 106,65 * 40,39 * 48,41 * 65,40)^{1/(1)(2)} = 71,59$$

**Tablo 2.7** Bazı SPEC CINT2006 Sonuçları

(a) Sun Blade 1000

Benchmark	Yürütme süresi (sn)	Yürütme süresi (sn)	Yürütme süresi (sn)	Referans süresi (sn)	Oran
400.perlbench	3077	3076	3080	9770	3.18
401.bzip2	3260	3263	3260	9650	2.96
403.gcc	2711	2701	2702	8050	2.98
429.mcf	2356	2331	2301	9120	3.91
445.gobmk	3319	3310	3308	10,490	3.17
456.hmmer	2586	2587	2601	9330	3.61

(Devamı)

**Tablo 2.7** (*Devamı*)**(a) Sun Blade 1000**

Benchmark	Yürütme süresi (sn)	Yürütme süresi (sn)	Yürütme süresi (sn)	Referans süresi (sn)	Oran
458.sjeng	3452	3449	3449	12,100	3.51
462.libquantum	10,318	10,319	10,273	20,720	2.01
464.h264ref	5246	5290	5259	22,130	4.21
471.omnetpp	2565	2572	2582	6250	2.43
473.astar	2522	2554	2565	7020	2.75
483.xalancbmk	2014	2018	2018	6900	3.42

**(b) Sun Blade x6250**

Benchmark	Yürütme süresi (sn)	Yürütme süresi (sn)	Yürütme süresi (sn)	Referans süresi (sn)	Oran	Oran
400.perlbench	497	497	497	9770	19.66	78.63
401.bzip2	613	614	613	9650	15.74	62.97
403.gcc	529	529	529	8050	15.22	60.87
429.mcf	472	472	473	9120	19.32	77.29
445.gobmk	637	637	637	10,490	16.47	65.87
456.hmmer	446	446	446	9330	20.92	83.68
458.sjeng	631	632	630	12,100	19.18	76.70
462.libquantum	614	614	614	20,720	33.75	134.98
464.h264ref	830	830	830	22,130	26.66	106.65
471.omnetpp	619	620	619	6250	10.10	40.39
473.astar	580	580	580	7020	12.10	48.41
483.xalancbmk	422	422	422	6900	16.35	65.40

## 2.7 kEy TERMS, İNCELEME SORULARI VE PRBLEMLER

### Anahtar Terimler

Amdahl yasası aritmetik ortalama (AM) temel metrik ölçüdü saat döngüsü saat döngüsü zaman saat hızı saat hızı saat tık komut başına döngü (CPI)	fonsiyonel ortalama (FM) genel amaçlı hesaplama GPU üzerinde (GPGPU) geometrik ortalama (GM) grafik işlem birimi (GPU) harmonik ortalama (HM) komut yürütme hızı Little yasası birçok entegre çekirdek (MIC)	mikroişlemci MIPS hızı çok çekirdeklı tepe metrik oran metriği referans makine hızı metrik SPEC test edilen sistem verimi
---	---	--

## İnceleme Soruları

- 2.1** Çağdaş işlemcilerde hızı artırmak için kullanılan tekniklerden bazılarını listeleyiniz ve kısaca tanımlayınız.
- 2.2** Performans dengesi kavramını açıklar.
- 2.3** Çok çekirdekli sistemler, MIC'ler ve GPGPU'lar arasındaki farkları açıklayabilecektir.
- 2.4** Amdahl yasasını kısaca tanımlayınız.
- 2.5** Little yasasını kısaca tanımlayınız.
- 2.6** MIPS ve FLOPS'u tanımlayınız.
- 2.7** Bir dizi veri değerinin ortalama değerini hesaplamak için üç yöntemi listeleyiniz ve tanımlayınız.
- 2.8** Bir kıyaslama programının arzu edilen özelliklerini listeleyiniz.
- 2.9** SPEC ölçütleri nelerdir?
- 2.10** Baz metrik, pik metrik, hız metrik ve oran metrik arasındaki farklar nelerdir?

## Problemler

- 2.1** Bir kıyaslama programı 40 MHz'lik bir işlemci üzerinde çalıştırılır. Yürütülen program, aşağıdaki komut karışımı ve saat döngüsü sayısı ile 100.000 komut yürütmesinden oluşur:

Talimat Türü	Talimat Sayısı	Talimat Başına Döngüler
Tamsayı aritmetiği	45,000	1
Veri aktarımı	32,000	2
Kayan nokta	15,000	2
Kontrol aktarımı	8000	2

Bu program için etkin *CPI*, MIPS oranı ve yürütme süresini belirleyin.

- 2.2** Her ikisi de 200 MHz saat hızına sahip iki farklı komut setine sahip iki farklı makine düşünün. Aşağıdaki ölçümler, verilen bir dizi kıyaslama programını çalıştırın iki makinede kaydedilmiştir:

Talimat Türü	Talimat Sayısı (milyon)	Talimat Başına Döngüler
Makine A		
Aritmetik ve mantık	8	1
Yükleyin ve saklayın	4	3
Şube	2	4
Diğerleri	4	3
Makine B		
Aritmetik ve mantık	10	1
Yükleyin ve saklayın	8	2
Şube	2	4
Diğerleri	4	3

- a.** Her makine için etkin *CPI*, MIPS oranı ve yürütme süresini belirleyin.
- b.** Sonuçlar hakkında yorum yapın.

- 2.3** CISC ve RISC tasarımlının ilk örnekleri sırasıyla VAX 11/780 ve IBM RS/6000'dir. Tipik bir kıyaslama programı kullanıldığında, aşağıdaki makine karakteristikleri ortaya çıkar:

İşlemci	Saat Frekansı (MHz)	Performans (MIPS)	CPU Süresi (sn)
VAX 11/780	5	1	$12x$
IBM RS/6000	25	18	$x$

Son sütun, VAX'in CPU süresinde IBM'den 12 kat daha uzun süre gerektirdiğini göstermektedir.

- a.** İki makinede çalışan bu kıyaslama programı için makine kodunun komut sayısının göreceli boyutu nedir?
- b.** İki makine için *CPI* değerleri nedir?

- 2.4** Dört kıyaslama programı üç bilgisayarda çalıştırılmış ve aşağıdaki sonuçlar elde edilmiştir:

	Bilgisayar A	Bilgisayar B	Bilgisayar C
Program 1	1	10	20
Program 2	1000	100	20
Program 3	500	1000	50
Program 4	100	800	100

Tablo, dört programın her birinde 100.000.000 komut çalıştırıldığında saniye cinsinden yürütme süresini göstermektedir. Her program için her bilgisayarın MIPS değerlerini hesaplayın. Ardından dört program için eşit ağırlıklar varsayılarak aritmetik ve harmonik ortalamaları hesaplayın ve bilgisayarları aritmetik ortalama ve harmonik ortalamaya göre sıralayın.

- 2.5** Literatürde [HEAT84] bildirilen verilere dayanan aşağıdaki tablo, üç makinede beş farklı kıyaslama programı için saniye cinsinden yürütme sürelerini göstermektedir.

Benchmark	İşlemci		
	R	M	Z
E	417	244	134
F	83	70	70
H	66	153	135
I	39,449	35,527	66,000
K	772	368	369

- a.** Her kıyaslama için her bir işlemcinin hız metriğini R makinesine normalize hesaplayın. Yani, R için oran değerlerinin tümü 1,0'dır. Diğer oranlar, R referans sistem olarak ele alınarak Denklem (2.5) kullanılarak hesaplanır. Ardından Denklem (2.3) kullanılarak her sistem için aritmetik ortalama değer hesaplanır. Bu, [HEAT84]'te benimsenen yaklaşımdır.
- b.** M'yi referans makine olarak kullanarak (a) bölümünü tekrarlayın. Bu hesaplama [HEAT84]'te denenmemiştir.
- c.** Önceki iki hesaplamanın her birine göre en yavaş makine hangisidir?
- d.** Denklem (2.6)'da tanımlanan geometrik ortalamayı kullanarak (a) ve (b) bölümlerindeki hesaplamları tekrarlayın. İki hesaplama göre en yavaş makine hangisidir?

**2.6** Bir önceki sorunun sonuçlarını açıklığa kavuşturmak için daha basit bir örneğe bakalım.

Benchmark	İşlemci		
	X	Y	Z
1	20	10	40
2	40	80	20

- a. Referans makine olarak X'i ve ardından referans makine olarak Y'yi kullanarak her bir sistem için aritmetik ortalama değeri hesaplayın. Sezgisel olarak üç makinenin kabaca eşdeğer performansa sahip olduğunu ve aritmetik ortalamanın yanıltıcı sonuçlar verdiği iddia edin.
- b. Referans makine olarak X'i ve ardından referans makine olarak Y'yi kullanarak her bir sistem için geometrik ortalama değeri hesaplayın. Sonuçların aritmetik ortalamaya göre daha gerçekçi olduğunu iddia edin.

**2.7** Ortalama CPI ve MIPS oranının hesaplanması için Bölüm 2.5'teki örneği düşünün;  $CPI = 2.24$  ve MIPS oranı = 178 sonucunu vermiştir. Şimdi programın sekiz paralel görevde veya iş parçacığında yürütülebileceğini ve her görevde kabaca eşit sayıda talimatın yürütüleceğini varsayıyalım. Yürütme, her bir çekirdeğin (işlemci) başlangıçta kullanılan tek işlemciyle aynı performansa sahip olduğu 8 çekirdekli bir sistemde gerçekleştiriliyor. Parçalar arasındaki koordinasyon ve senkronizasyon, her bir görevde fazladan 25.000 komut yürütme ekler. Her görev için önektekiyle aynı komut karışımı varsayıyalım, ancak bellek için çakışma nedeniyle önbellek kaçırmalı bellek referansı için  $CPI$ 'yi 12 döngüye çıkaralım.

- a. Ortalama TÜFEY'yi belirleyin.
- b. Karşılık gelen MIPS oranını belirleyin.
- c. Hızlandırma faktörünü hesaplayın.
- d. Gerçek hızlandırma faktörünü Amdhal yasası tarafından belirlenen teorik hızlandırma faktörü ile karşılaştırın.

**2.8** Bir işlemci ana belleğe ortalama  $T_2$  erişim süresiyle erişir. İşlemci ile ana bellek arasına daha küçük bir ön yerleştirilmiştir. Ön bellek,  $T_1$  &  $T_2$ 'lik önemli ölçüde daha hızlı bir erişim süresine sahiptir. Ön bellek, herhangi bir zamanda bazı ana bellek sözcüklerinin kopyalarını tutar ve yakın gelecekte erişilme olasılığı daha yüksek olan sözcükler ön bellekte olacak şekilde tasarılanmıştır. İşlemci tarafından erişilen bir sonraki kelimenin önbellekte olma olasılığının isabet oranı olarak bilinen  $H$  olduğunu varsayıyalım.

- a. Herhangi bir tek bellek erişimi için, kelimeye ana bellek yerine önbellekten erişmenin teorik hız artışı nedir?
- b.  $T$  ortalama erişim süresi olsun.  $T$ 'yi  $T_1$ ,  $T_2$  ve  $H$ 'nin bir fonksiyonu olarak ifade edin.  $H$ 'nin bir fonksiyonu olarak genel hızlanma?
- c. Pratikte, bir sistem, işlemcinin kelimenin önbellekte olup olmadığını belirlemek için önbelleğe erişmesi ve değilse ana belleğe erişmesi gerekecek şekilde tasarlanabilir, böylece bir iskalama durumunda (isabetin tersi), bellek erişim süresi  $T_1 + T_2$  olur.  $T$ yi  $T_1$ ,  $T_2$  ve  $H$ 'nin bir fonksiyonu olarak ifade edin. Şimdi hızlanmayı hesaplayın ve bölüm (b)'de üretilen sonuçla karşılaştırın.

**2.9** Bir dükkanın sahibi, saatte ortalama 18 müşterinin geldiğini ve dükkan'da genellikle 8 müşteri olduğunu gözlemliyor. Her müşterinin dükkan'da geçirdiği ortalama süre ne kadardır?

**2.10** Şekil 2.8a'yı göz önünde bulundurarak Little yasası hakkında daha fazla bilgi edinebiliriz. Bir  $T$  süresi boyunca, bir sisteme toplam  $C$  öğe gelir, hizmet için bekler ve hizmeti tamamlar. Üstteki düz çizgi varişlerin zaman sırasını, alttaki düz çizgi ise ayrırlıkların zaman sırasını göstermektedir. İki çizgi tarafından sınırlanan gölgeli alan, iş-saniye birimi cinsinden sistem tarafından yapılan toplam "iş" temsil eder;  $A$  toplam iş olsun.  $L$ ,  $W$  ve  $\lambda$  arasındaki ilişkiye türetmek istiyoruz.

a. Şekil 2.8b toplam alanı her biri bir iş yüksekliğinde yatay dikdörtgenlere bölmektedir. Tüm bu dikdörtgenleri, sol kenarları  $t=0$  noktasında hizalanacak şekilde sola kaydırığınızı hayal edin.  $A$ ,  $C$  ve  $W$ 'yi ilişkilendiren bir denklem geliştirin.

b. Şekil 2.8c toplam alanı kesikli çizgilerle gösterilen dikey geçiş sınırlarıyla tanımlanan dikey dikdörtgenlere bölmektedir. Tüm bu alt kenarları  $N(t)=0$  noktasında hizalanacak şekilde sağa kaydırığınızı hayal edin.  $A$ ,  $T$  ve  $L$ 'yi ilişkilendiren bir denklem geliştirin.

c. Son olarak, (a) ve (b) sonuçlarından  $L=IW$ 'yi türetin.

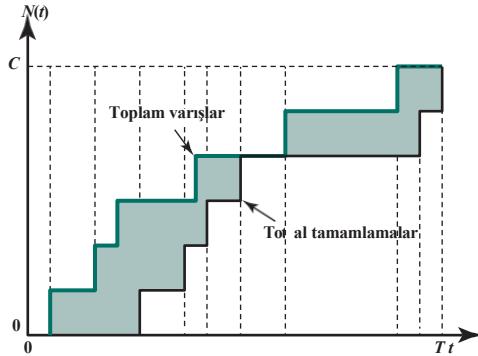
**2.11** Şekil 2.8'a'da işler  $t=0, 1, 1,5, 3,25, 5,25$  ve  $7,75$  zamanlarında gelmektedir. Karşılık gelen tamamlanma süreleri  $t=2, 3, 3,5, 4,25, 8,25$  ve  $8,75$ 'tir.

a. Şekil 2.8'b'deki altı dikdörtgenin her birinin alanını belirleyin ve toplam  $A$  alanını elde etmek için toplayın.

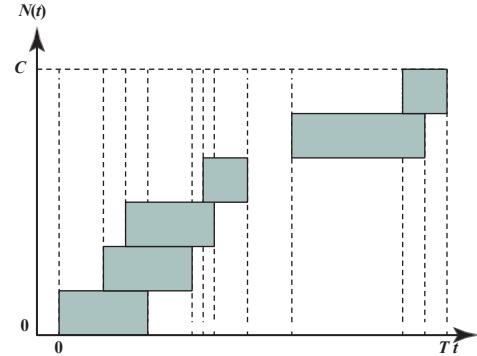
b. Şekil 2.8c'deki 10 dikdörtgenin her birinin alanını belirleyin ve toplam  $A$  alanını elde etmek için toplayın.

**2.12** Bölüm 2.6'da, test edilen bir sistemi referans bir sistemle karşılaştırmak için kullanılan temel oranın şu olduğunu belirtmişтик:

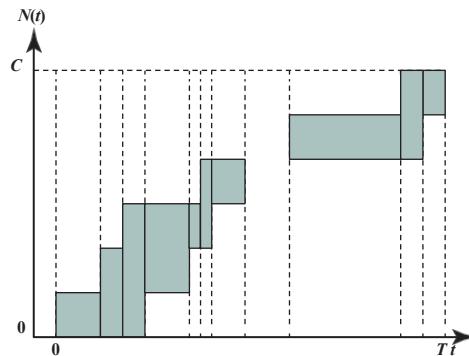
$$= \frac{Tref_i}{Tsut_i}$$



(a) İşlerin varışı ve tamamlanması



(b) Yatay dikdörtgenler olarak görüntülenir



(c) Dikey dikdörtgenler olarak görüntülenir

**Şekil 2.8** Little Yasasının Gösterimi

- a.** Yukarıdaki denklem, referans sisteme kıyasla test edilen sistemin hızlanması bir ölçüsünü sağlar. Test programında yürütülen kayan nokta işlemlerinin sayısının  $I_i$  olduğunu varsayılm. Şimdi hızlanması komut yürütme oranı  $FLOPS_{ijoin}$  bir fonksiyonu olarak gösterin.
- b.** Performansı normalleştirmek için bir başka teknik de bir sistemin performansını başka bir sistemin performansına göre yüzde değişim olarak ifade etmektir. Bu göreceli değişim önce komut yürütme hızının bir fonksiyonu olarak, sonra da yürütme sürelerinin bir fonksiyonu olarak ifade edin.
- 2.13** Bir kıyaslama programının A referans makinesinde 480 saniyede çalıştığını varsayılm. Aynı program B, C ve D sistemlerinde sırasıyla 360, 540 210 saniyede çalışır.
- a.** Test edilen üç sistemin her birinin A'ya göre hızmasını gösterin.
- b.** Şimdi üç sistemin göreceli hızmasını gösterin. Makineleri karşılaştırmanın üç yolu hakkında yorum yapın (yürütme süresi, hızlanma, göreli hızlanma).
- 2.14** Referans olarak D makinesini kullanarak önceki problemi tekrarlayın. Bu durum dört sistemin göreceli sıralamasını nasıl etkiler?
- 2.15** Tablo 2.4'teki bilgisayar zamanı verilerini kullanarak 2.2'deki sonuçları yeniden hesaplayın ve sonuçları yorumlayın.
- 2.16** Denklem 2.5, geometrik ortalamanın biri çarpım operatörü diğeri de toplama operatörü kullanan iki farklı formülasyonunu göstermektedir.
- a.** İki formülün eşdeğer olduğunu gösterin.
- b.** Geometrik ortalamayı hesaplamak için neden toplama formülasyonu tercih edilir?
- 2.17** **Proje.** Bölüm 2.5'te "kıyaslama araçları savaşlarını" belgeleyen bir dizi referans listelenmektedir. Referans verilen tüm makaleler [box.com/COA10e](http://box.com/COA10e) adresinde mevcuttur. Bu makaleleri okuyun ve SPEC hesaplamaları için geometrik ortalama kullanımının lehine ve aleyhine olan durumu özetleyin.

# **BÖLÜM** **3**

## **BİLGİSAYAR İŞLEVİ VE ARA BAĞLANTILARINA ÜST BİR BAKIŞ**

- 3.1 Bilgisayar Bileşenleri**
- 3.2 Bilgisayar Fonksiyonu**
  - Komut Alma ve Yürütme Kesmeleri
  - G/Ç İşlevi
- 3.3 Ara Bağlantı Yapıları**
- 3.4 Otobüs Ara Bağlantısı**
- 3.5 Noktadan Noktaya Ara Bağlantı**
  - QPI Fiziksel Katman
  - QPI Bağlantı Katmanı
  - QPI Yönlendirme
  - Katmanı QPI Protokol
  - Katmanı
- 3.6 PCI Express**
  - PCI Fiziksel ve Mantıksal Mimarisi PCIe Fiziksel
  - Katmanı
  - PCIe İşlem Katmanı PCIe Veri
  - Bağlantısı Katmanı
- 3.7 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler**

### ÖĞRENİM HEDEFLERİ

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- Bir komut döngüsünün temel unsurlarını ve kesmelerin rolünü anlamak.
- Bir bilgisayar sistemi içindeki ara bağlantı kavramını tanımlamak.
- Noktadan noktaya ara bağlantıının veri yolu ara bağlantısına kıyasla göreceli avantajlarını değerlendirin.
- QPI hakkında genel bir bakış sunun.
- PCIe'ye genel bir bakış sunun.

En üst düzeyde, bir bilgisayar CPU (merkezi işlem birimi), bellek ve I/O bileşenlerinden ve her türden bir veya daha fazla modülden oluşur. Bu bileşenler, bilgisayarin temel işlevi olan programların yürütülmesini sağlamak için bir şekilde birbirine bağlanır. Dolayısıyla, en üst düzeyde, bir bilgisayar sistemini (1) her bir bileşenin dış davranışını, yani diğer bileşenlerle değişim tokuş ettiği veri ve kontrol sinyallerini ve (2) ara bağlantı yapısını ve ara bağlantı yapısının kullanımını yönetmek için gereken kontrolleri tanımlayarak karakterize edebiliriz.

Bu üst düzey yapı ve iGlev görüpü, bilgisayarın doğasını anladamak açıklayıcı gücü nedeniyle önemlidir. Aynı derecede önemli olan bir başka husus da performans değerlendirmesinin giderek karmaşıklaşan konularını anlamak için kullanılmasıdır. Üst düzey yapı ve işlevin kavranması, sistem darboğazları, alternatif yollar, bir bileşenin arızalanması durumunda sistem arızalarının büyülüklüğü ve performans geliştirmelerinin eklenmesinin kolaylığı hakkında fikir verir. Birçok durumda, daha fazla sistem gücü ve arıza emniyetli yetenekler için gereksinimler, yalnızca tek tek bileşenlerin hızını ve güvenilirliğini artırmak yerine tasarımları değiştirmektedir.

Bu bölüm, bilgisayar bileşenlerinin birbirine bağlanması için kullanılan temel yapılarla odaklanmaktadır. Arka plan olarak, bölüm temel bileşenlerin ve arayüz gereksinimlerinin kısa incelemesi ile başlar. Daha sonra işlevsel bir genel bakış sağlanmaktadır. Daha sonra sistem bileşenlerini birbirine bağlamak için veri yollarının kullanımını incelemeye hazırız.

## 3.1 BİLGİSAYAR BİLEŞENLERİ

Bölüm 1'de tartışıldığı gibi, neredeyse tüm çağdaş bilgisayar tasarımları John von Neumann tarafından Princeton İleri Araştırmalar Enstitüsü'nde geliştirilen kavamlara dayanmaktadır. Böyle bir tasarım von Neumann mimarisı olarak adlandırılır ve üç temel kavrama dayanır:

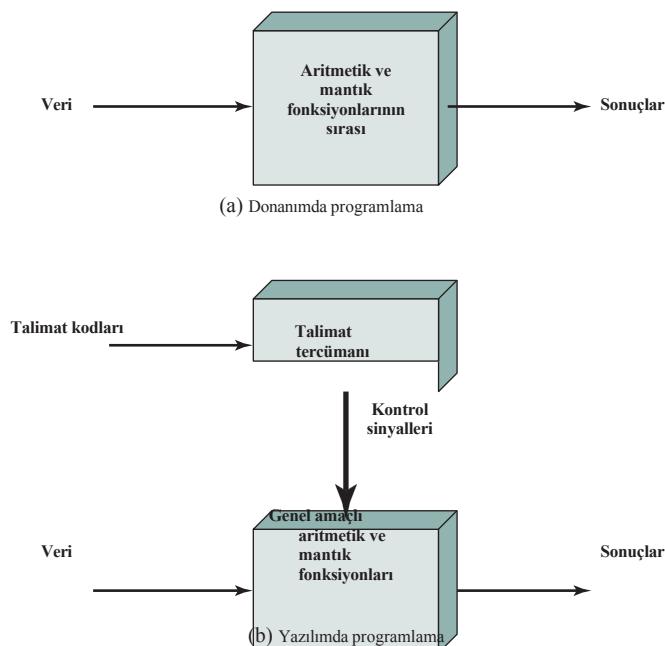
- Veri ve talimatlar tek bir okuma-yazma belleğinde saklanır.
- Bu belleğin içeriği, içerdiği verilerin türüne bakılmaksızın konuma göre adreslenebilir.

- Yürütme, bir komuttan diğerine sıralı bir şekilde (açıkça değiştirilmediği sürece) gerçekleşir.

Bu kavramların arkasındaki mantık Bölüm 2'de tartışılmıştır ancak burada özetlenmeye değerdir. İkili verileri depolamak ve bu üzerinde aritmetik ve mantıksal işlemler gerçekleştirmek için çeşitli şekillerde bir araya getirilebilen küçük bir temel mantık bileşenleri kümesi vardır. Gerçekleştirilecek belirli bir hesaplama varsa, bu hesaplama için özel olarak tasarılanmış bir mantık bileşenleri konfigürasyonu oluşturulabilir. Çeşitli bileşenleri istenen konfigürasyona bağlama işlemini bir tür programlama olarak düşünebiliriz. Ortaya çıkan "program" donanım şeklindedir ve *kablolu program* olarak adlandırılır.

Şimdi bu alternatifin düşünün. Aritmetik ve mantık fonksiyonlarından oluşan genel amaçlı bir konfigürasyon oluşturduğumuzu varsayıyalım. Bu donanım seti, donanıma uygulanan kontrol sinyallerine bağlı olarak veriler üzerinde çeşitli işlevler gerçekleştirecektir. Özelleştirilmiş donanımın orijinal durumunda, sistem verileri kabul eder ve sonuçlar üretir (Şekil 3.1a). Genel amaçlı donanımda ise sistem veri ve kontrol sinyallerini kabul eder ve sonuç üretir. Böylece, her yeni program için donanımı yeniden kablolamak yerine, programının yalnızca yeni bir kontrol sinyalleri seti sağlamaası gereklidir.

Kontrol sinyalleri nasıl sağlanmalıdır? Cevap basit ama incelikli. Tüm program aslında bir adımlar dizisidir. Her adımda, bazı veriler üzerinde bazı aritmetik veya mantıksal işlemler gerçekleştirilir. Her adım için yeni bir kontrol sinyali kümesine ihtiyaç vardır. Her olası kontrol sinyali kümesi için benzersiz bir kod sağlayalım,



**Şekil 3.1** Donanım ve Yazılım Yaklaşımları

ve genel amaçlı donanıma bir kod kabul edebilen ve kontrol sinyalleri üretebilen bir segment ekleyelim (Şekil 3.1b).

Programlama artık çok daha kolay. Her yeni program için donanımı yeniden kablolamak yerine, tek yapmamız gereken yeni bir kod dizisi sağlamaktır. Her kod aslında bir talimattır ve donanımın bir kısmı her talimat yorumları ve kontrol sinyalleri üretir. Bu yeni programlama yöntemini ayırt etmek için, bir dizi kod veya talimat *yazılım* olarak adlandırılır.

Şekil 3.1b sistemin iki ana bileşenini göstermektedir: bir komut yorumlayıcısı ve bir genel amaçlı aritmetik ve mantık fonksiyonları modülü. Bu ikisi CPU'yu oluşturur. İşleyen bir bilgisayar elde etmek için birkaç başka bileşene daha ihtiyaç vardır. Veri ve talimatlar sisteme girilmelidir. Bunun için bir çeşit giriş modülüne ihtiyacımız vardır. Bu modül, veri ve talimatları bir biçimde kabul etmek ve bunları sistem tarafından kullanılabilecek dahili bir sinyal biçimine dönüştürmek için temel bileşenleri içerir. Sonuçları raporlamak için bir aracı ihtiyaç ve bu da bir çıkış modülü şeklindedir. Bunlar birlikte ele alındığında *I/O bileşenleri* olarak adlandırılır.

Bir bileşene daha ihtiyaç vardır. Bir giriş aygıtı talimatları ve verileri sırayla getirecektir. Ancak bir program her zaman sıralı olarak yürütülmez; (örneğin, IAS atlama talimi). Benzer şekilde, veriler üzerindeki işlemler önceden belirlenmiş bir sırayla bir seferde birden fazla öğeye erişim gerektirebilir. Bu nedenle, hem talimatları hem de verileri geçici olarak saklayacak bir yer olmalıdır. Bu modül, harici depolama veya çevresel aygıtlardan ayırt etmek için *bellek* veya *ana bellek* olarak adlandırılır. Von Neumann, aynı belleğin hem talimatları hem de verileri depolamak için kullanılabilcecine dikkat çekmiştir.

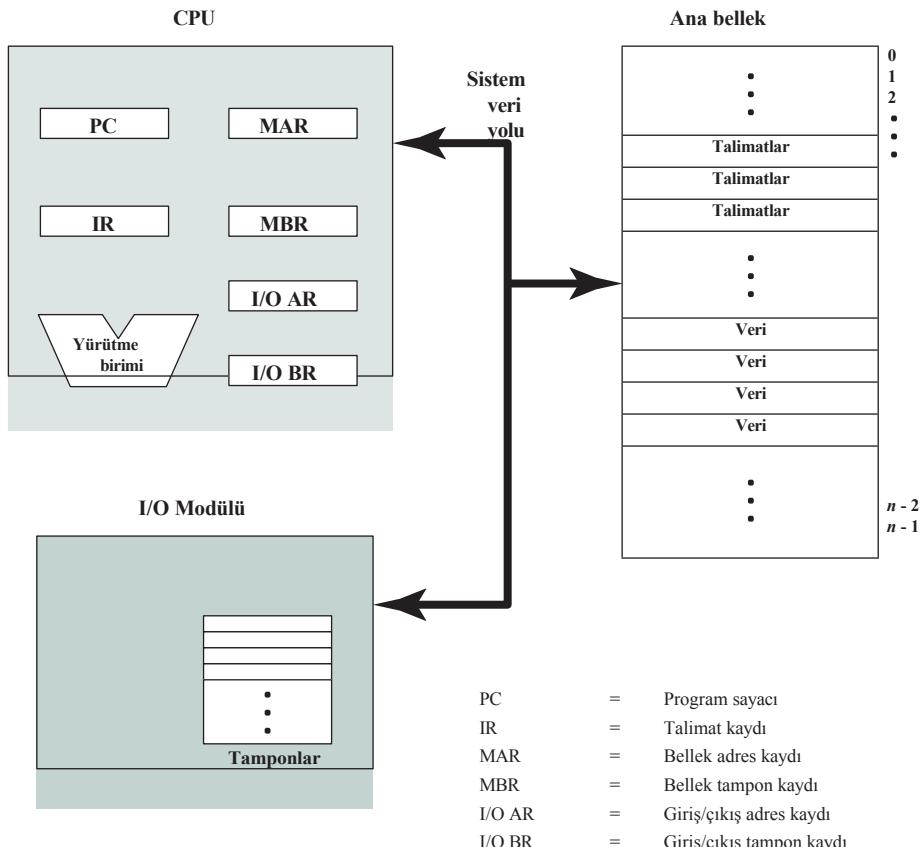
Şekil 3.2 bu üst düzey bileşenleri ve ilişkileri göstermektedir. CPU bellek ile veri alışverışı yapar. Bu amaçla, tipik olarak iki dahili (CPU'ya ait) kayıttan yararlanır: bir sonraki okuma veya yazma için bellekteki adresi belirten bir bellek **adres kaydı (MAR)** ve belleğe yazılacak verileri içeren veya okunan verileri alan bir bellek **tampon kaydı (MBR)**. Benzer şekilde, bir I/O adres kaydı (I/OAR) belirli bir I/O cihazını belirtir. Bir I/O tampon kaydı (I/OBR), bir I/O modülü ile CPU arasında veri alışverışı için kullanılır.

Bir bellek modülü, sıralı olarak numaralandırılmış adreslerle tanımlanan bir dizi konumdan oluşur. Her konum, bir komut veya veri olarak yorumlanabilen ikili bir sayı içerir. Bir G/C modülü verileri harici aygıtlardan CPU'ya ve belleğe aktarır ve bunun tersi de geçerlidir. Bu verileri gönderilinceye kadar geçici olarak tutmak için dahili tamponlar içerir.

Bu ana bileşenlere kısaca baktıktan sonra, şimdi bu bileşenlerin programları yürütmek için birlikte nasıl çalıştığını genel bir bakışa dönüyoruz.

## 3.2 BİLGİSAYAR İŞLEVİ

Bir bilgisayar tarafından gerçekleştirilen temel işlev, bellekte saklanan bir dizi talimattan oluşan bir programın yürütülmesidir. İşlemci, programda belirtilen talimatları uygulayarak asıl işi yapar. Bu bölüm, aşağıdaki konulara genel bir bakış sunmaktadır



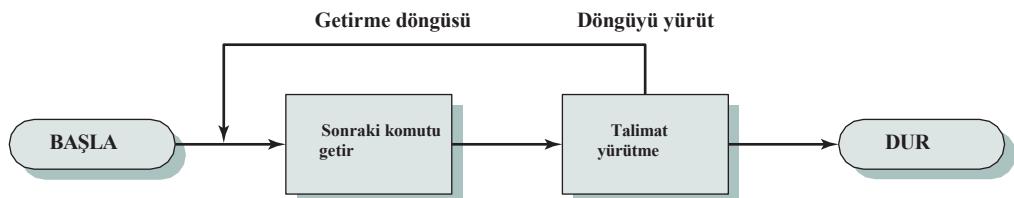
**Şekil 3.2** Bilgisayar Bileşenleri: Üst Düzey Görünüm

program yürütmenin temel unsurlarıdır. En basit haliyle, komut işleme iki adımdan oluşur: İşlemci komutları bellekten teker okur (*getirir*) ve her bir komutu yürütür. Programın yürütülmesi, komut getirme ve komut yürütme tekrarlanmasından oluşur. Komutun yürütülmesi birkaç işlem içerebilir ve komutun niteliğine bağlıdır (örneğin, Şekil 2.4'ün alt kısmına bakınız).

Tek bir komut için gereken işleme **komut döngüsü** denir. Daha önce verilen basitleştirilmiş iki adımlı açıklama kullanılarak, komut döngüsü Şekil 3.3'te gösterilmiştir. Bu iki adım **getirme döngüsü** ve **yürütme döngüsü** olarak adlandırılır. Programın yürütülmesi ancak makine kapatılırsa, bir tür düzeltilemez hata oluşursa veya bilgisayarı durduran bir program komutuyla karşılaşılırsa durur.

### Komut Alma ve Yürütme

Her komut döngüsünün başında, işlemci bellekten bir komut alır. Tipik bir işlemcide, program sayacı (PC) adı verilen bir yazmaç, bir sonraki getirilecek komutun adresini tutar. Aksi söylenenmedikçe, işlemci



**Şekil 3.3** Temel Komut Döngüsü

her komut getirme işleminden sonra PC'yi artırır, böylece sıradaki komutu (yani, bir sonraki yüksek bellek adresinde bulunan komutu) getirir. Örneğin, her bir komutun belleğin 16 bitlik bir kelimesini kapladığı bir bilgisayar düşünün. Program sayacının 300 numaralı bellek konumuna ayarlandığını varsayıyalım; burada konum adresi 16 bitlik bir sözcüğü ifade etmektedir. İşlemci daha sonra 300 numaralı konumdaki komutu getirecektir. Takip eden komut çevrimlerinde, 301, 302, 303 ve benzeri konumlardan komutları getirecektir. Bu sırada, ileride açıklanacağı gibi değiştirilebilir.

Getirilen talimat, işlemcide talimat kaydı (IR) olarak bilinen bir kayda yüklenir. Komut, işlemcinin gerçekleştireceğini eylemi belirten bitler içerir. İşlemci talimatı yorumlar ve gerekli eylemi gerçekleştirir. Genel olarak, bu eylemler dört kategoriye ayrılır:

- **İşlemci-bellek:** Veriler işlemciden belleğe veya bellekten işlemciye aktarılabilir.
- **İşlemci-I/O:** Veri, işlemci ve bir I/O modülü arasında aktarım yapılarak bir çevresel cihaza veya cihazdan aktarılabilir.
- **Veri işleme:** İşlemci veriler üzerinde bazı aritmetik veya mantık gerçekleştirilebilir.
- **Kontrol:** Bir komut, yürütme sırasının değiştirilmesini belirtebilir. Örneğin, işlemci 149 konumundan bir komut getirebilir ve bu komut bir sonraki komutun 182 konumundan geleceğini belirtir. İşlemci, program sayacını 182'ye ayarlayarak bu gerçeği hatırlayacaktır. Böylece, bir sonraki getirme çevriminde, komut 150 yerine 182 konumundan getirilecektir.

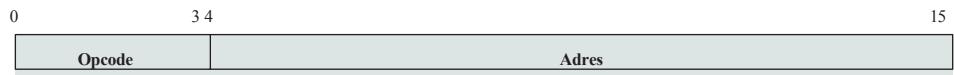
Bir taliminin yürütülmesi bu eylemlerin bir kombinasyonunu içerebilir.

Şekil 3.4'te listelenen özelliklerini içeren varsayımsal bir makine kullanarak basit bir örnek düşünün. İşlemci, akümülatör (AC) adı verilen tek bir veri kaydı içerir. Hem komutlar hem de veriler 16 bit uzunluğundadır. Bu nedenle, belleği 16 bitlik sözcükler kullanarak düzenlemek uygunudur. Komut formatı işlem kodu için 4 bit sağlar, böylece  $2^4 = 16$  farklı işlem kodu olabilir ve  $2^{12} = 4096$  (4K) kelimeye kadar bellek doğrudan adreslenebilir.

Şekil 3.5, bellek ve işlemci yazmalarının ilgili bölümlerini gösteren kısmi bir program yürütmesini göstermektedir.<sup>1</sup> Gösterilen program parçası, 940 adresindeki bellek sözcüğünün içeriğini

<sup>1</sup> Her bir basamağın 4 biti temsil ettiği onaltılık gösterim kullanılır. Bu, kelime uzunluğu 4'ten katı olduğunda bellek ve kayıtların içeriğini temsil etmek için en uygun gösterimdir. Sayı sistemleri (ondalık, ikili, onaltılık) hakkında temel bilgiler için Bölüm 9'a bakın.

## 86 BÖLÜM 3 / BİLGİSAYAR İŞLEVLERİİNİN VE BAĞLANTILARIİNİN ÜST DÜZEY GÖRÜNÜMÜ



(a) Talimat formatı



(b) Tamsayı biçimi

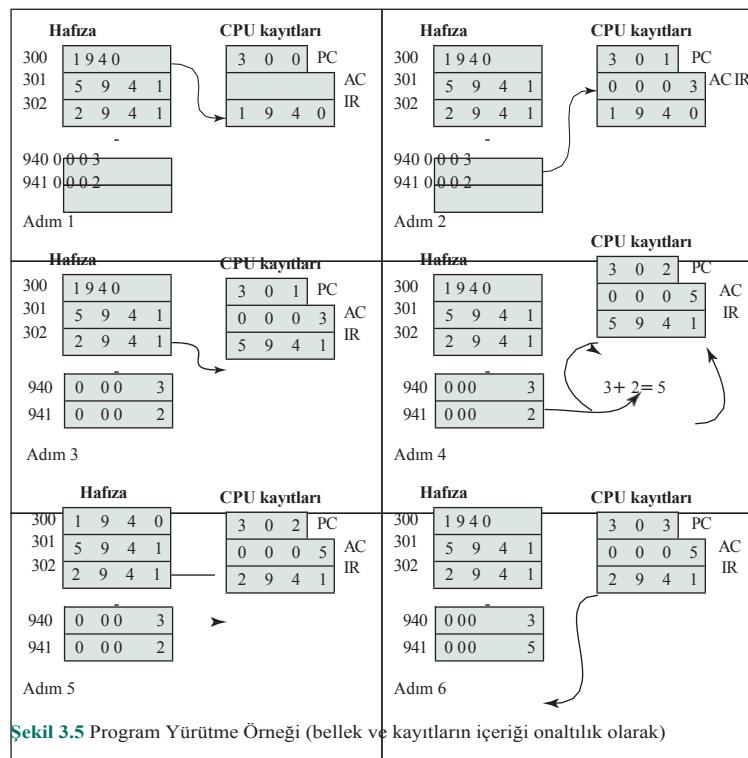
Program sayacı (PC)= Komutun adresi Komut kaydı (IR)= Yürütülen komut Akümülatör (AC)= Geçici depolama

(c) Dahili CPU kayıtları

0001= AC'yi bellekten yükle 0010= AC'yi belleğe kaydet 0101= Bellekten AC'ye ekle

(d) İşlem kodlarının kısmi listesi

**Şekil 3.4** Varsayımsal Bir Makinenin Özellikleri



941 adresini alır ve sonucu ikinci konumda saklar. Üç getirme ve üç yürütme döngüsü olarak tanımlanabilecek üç talimat gereklidir:

1. PC, ilk komutun adresi olan 300'ü içerir. Bu komut (onaltılık olarak 1940 değeri) IR komut kaydına yüklenir ve PC artırılır. Bu işlemin bir bellek adres kaydi ve bir bellek tampon kaydı kullanımını içerdigine dikkat edin. Basitlik için, bu ara yazmaçlar göz ardı edilir.
2. IR'deki ilk 4 bit (ilk onaltılık hane) AC'nin yükleneceğini gösterir. Kalan 12 bit (üç onaltılık basamak) verilerin yükleneceği adresi (940) belirtir.
3. Bir sonraki komut (5941) 301 konumundan getirilir ve PC artırılır.
4. AC'nin eski içeriği ile 941 konumunun içeriği toplanır ve sonuç AC'de saklanır.
5. Bir sonraki komut (2941) 302 konumundan getirilir ve PC artırılır.
6. AC'nin içeriği 941 numaralı konumda saklanır.

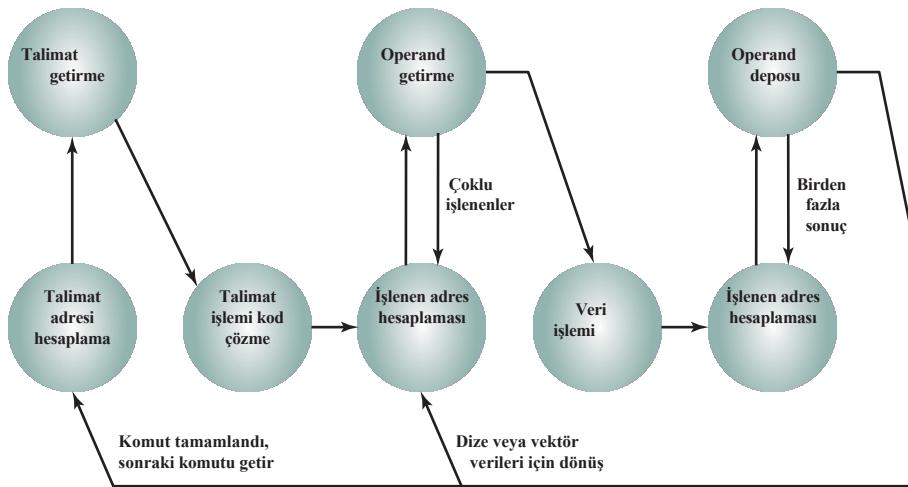
Bu örnekte, 940 konumunun içeriğini 941'in içeriğine eklemek için her biri bir getirme döngüsü ve bir yürütme döngüsünden oluşan üç komut döngüsü gereklidir. Daha karmaşık bir talimat setinde daha az döngü gerekebilir. Örneğin bazı eski işlemciler birden fazla bellek adresi içeren talimatlar içerir. Dolayısıyla, bu tür işlemcilerde belirli bir talimat için yürütme döngüsü belleğe birden fazla referans içerebilir. Ayrıca, bellek referansları yerine, bir komut bir I/O işlemi belirtebilir.

Örneğin, PDP-11 işlemcisi, simbolik olarak ADD B,A şeklinde ifade edilen ve B ve A bellek konumlarının içeriklerinin toplamını A bellek konumuna depolayan bir komut içerir:

- ADD komutunu getirin.
- A bellek konumunun içeriğini işlemciye okuyun.
- B bellek konumunun içeriğini işlemciye okuyun. A'nın içeriğinin kaybolmaması için, işlemcinin bellek değerlerini saklamak için tek bir akümülatör yerine en az iki kaydediciye sahip olması gereklidir.
- İki değeri toplayın.
- İşlemciden gelen sonucu A bellek konumuna yazın.

Bu nedenle, belirli bir komut için yürütme döngüsü belleğe birden fazla referans içerebilir. Ayrıca, bellek referansları yerine, bir komut bir I/O işlemi belirtebilir. Bu ek hususlar akılda tutularak, Şekil 3.6, Şekil 3.3'teki temel komut döngüsüne daha ayrıntılı bir bakış sağlar. Şekil bir durum diyagramı . Herhangi bir komut döngüsü için, bazı durumlar boş olabilir ve diğerleri birden fazla kez ziyaret edilebilir. Durumlar aşağıdaki gibi tanımlanabilir:

- **Komut adresi hesaplama (iac):** Yürüttülecek bir sonraki komutun adresini belirler. Genellikle bu işlem, sabit bir sayının



Şekil 3.6 Komut Döngüsü Durum Diyagramı

bir önceki adresi. Örneğin, her komut 16 bit uzunluğundaysa ve bellek 16 bitlik sözcükler halinde düzenlenmişse, önceki adrese 1 ekleyin. Bunun yerine, bellek ayrı ayrı adreslenebilir 8 bitlik baytlar olarak düzenlenmişse, önceki adrese 2 ekleyin.

- **Komut getirme (eğer):** Komutu bellek konumundan işlemciye okuyun.
- **Talimat işlemi kod çözme (iod):** Gerçekleştirilecek işlem türünü ve kullanılacak operand(lar)ı belirlemek için talimatı analiz edin.
- **İşlenen adres hesaplaması (oac):** İşlem bellekteki veya G/C yoluyla erişilebilen bir işlenene referans içeriyorsa, işlenenin adresini belirleyin.
- **İşlenen getirme (of):** İşleneni bellekten getirin veya G/C'den okuyun.
- **Veri işlemi (do):** Talimatta belirtilen işlemi gerçekleştirin.
- **İşlenen deposu (os):** Sonucu belleğe veya G/C'ye yazın.

Şekil 3.6'nın üst kısmındaki durumlar, işlemci ile bellek ya da G/C modülü arasında bir alışverişe işaret eder. Diyagramın alt kısmındaki durumlar yalnızca dahili işlemci işlemlerini içerir. Oac durumu iki kez görünür, çünkü bir komut bir okuma, bir yazma veya her ikisini de içerebilir. Ancak, bu durum sırasında gerçekleştirilen eylem her iki durumda da temelde aynıdır ve bu nedenle yalnızca tek bir tanımlayıcısına ihtiyaç vardır.

Ayrıca, bazı makinelerdeki bazı talimatlar bunu gerektirdiğinden, diyagramın birden fazla işlenen ve birden fazla sonuca izin verdiğine dikkat edin. Örneğin, PDP-11 ADD A,B komutu aşağıdaki durum dizisiyle sonuçlanır: iac, if, iod, oac, of, oac, of, do, oac, os.

Son olarak, bazı makinelerde, tek bir komut, sayılardan oluşan bir vektör (tek boyutlu dizi) veya bir dize (tek boyutlu dizi) üzerinde gerçekleştirilecek bir işlemi belirtebilir.

dizisi) karakterlerinden oluşur. Şekil 3.6'da gösterildiği gibi, bu tekrarlayan işlenen getirme ve/veya saklama işlemlerini içerecektir.

### Kesintiler

Neredeyse tüm bilgisayarlar, diğer modüllerin (G/C, bellek) işlemcinin normal işlemlerini **kesintiye uğratabileceği** bir mekanizma sağlar. Tablo 3.1 en yaygın kesme sınıflarını listeler. Bu kesmelerin özel doğası bu kitabın ilerleyen bölümlerinde, özellikle 7. ve 14. Bölümlerde incelenecaktır. Ancak, komut döngüsünün doğasını ve ara bağlantı yapısı üzerindeki etkilerini daha net anlamak için bu kavramı şimdi tanıtımız gerekmektedir. Okuyucunun bu aşamada kesmelerin oluşturulması ve işlenmesinin ayrıntıları ile ilgilenmesine gerek yoktur, sadece kesmelerden kaynaklanan modüller arasındaki iletişime odaklanmalıdır.

Kesmeler öncelikle işlem verimliliğini artırmak bir yolu olarak sağlanır. Örneğin, çoğu harici aygit işlemciden çok daha yavaştır. İşlemcinin Şekil 3.3'teki komut döngüsü şemasını kullanarak bir yazıcıya veri aktardığını varsayıyalım. Her yazma işleminden sonra, yazıcı yetişene kadar işlemcinin duraklaması ve boşta kalması gereklidir. Bu duraklamanın uzunluğu, bellek içermeyen yüzlerce hatta binlerce komut döngüsü mertebesinde olabilir. Bunun işlemci için çok savurgan bir kullanım olduğu açıklıktır.

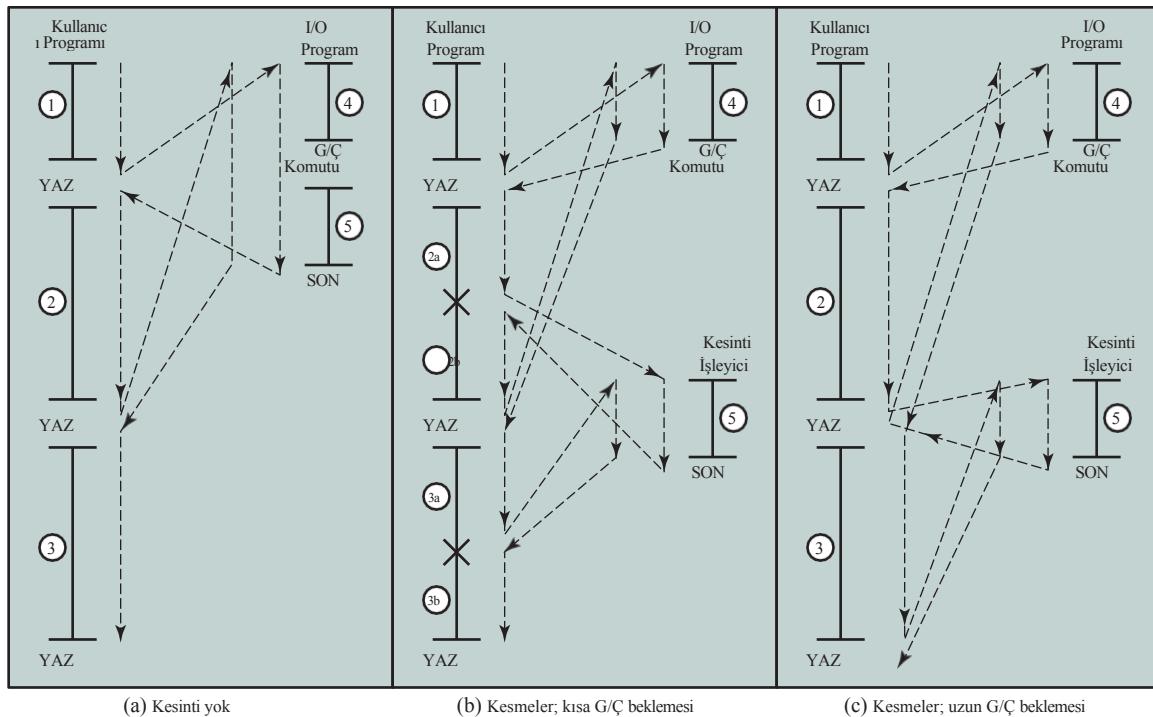
Şekil 3.7a bu durumu göstermektedir. Kullanıcı programı, işleme ile serpiştirilmiş bir dizi WRITE çağrıları gerçekleştirmektedir. Kod segmentleri 1, 2 ve 3, G/C içermeyen talimat dizilerini ifade eder. WRITE çağrıları, bir sistem yardımcı programı olan ve gerçek G/C işlemini gerçekleştirecek olan bir G/C programına yönelikler. G/C programı üç bölümünden oluşur:

- Gerçek G/C işlemine hazırlanmak için şekilde 4 olarak etiketlenmiş bir dizi talimat. Bu, çaptısı alınacak verilerin özel bir tampona kopyalanmasını ve bir cihaz komutu için parametrelerin hazırlanmasını içerebilir.
- Gerçek G/C komutu. Kesmeler kullanılmadığında, bu komut verildikten sonra program G/C aygıtının istenen işlevi yerine getirmesini beklemelidir (veya aygıtı periyodik olarak yoklamalıdır). Program, G/C işleminin yapılmış yapılmadığını belirlemek için bir test işlemini tekrar tekrar gerçekleştirerek bekleyebilir.
- İşlemi tamamlamak için şekilde 5 olarak etiketlenmiş bir dizi talimat. Bu, işlemin başarılı ya da başarısız olduğunu gösteren bir bayrağın ayarlanması içerebilir.

**Tablo 3.1** Kesinti Sınıfları

<b>Program</b>	Aritmetik taşıma, sıfır bölme, yasa dışı bir makine komutunu çalışma girişimi veya kullanıcının izin verdiği bellek alanının dışına referans verme gibi bir komutun yürütülmesi sonucunda ortaya çıkan bazı koşullar tarafından oluşturulur.
<b>Zamanlı</b>	İşlemci içindeki bir zamanlayıcı tarafından oluşturulur. Bu, işletim sisteminin belirli işlevleri düzenli olarak gerçekleştirmesini sağlar.
<b>ayıcı</b>	Bir işlemin normal şekilde tamamlandığını bildirmek, işlemciden servis talep etmek veya hata durumlarını bildirmek için bir G/C denetleyicisi tarafından oluşturulur.
<b>I/O</b>	Güç kesintisi veya bellek parite hatası gibi bir arıza nedeniyle oluşur.

Donanım Arızası



= kullanıcı programının yürütülmesi sırasında kesme olur

**Şekil 3.7** Kesmesiz ve Kesmeli Program Kontrol Akışı

G/C işleminin tamamlanması nispeten uzun sürebileceğinden, G/C programı işlemin bekler; bu nedenle, kullanıcı programı WRITE çağrı�ı noktasında önemli bir süre için durdurulur.

**KESMELER VE İŞLEM AKIŞI** Kesmeler sayesinde, bir G/C işlemi devam ederken işlemci diğer komutları yürütmeye meşgul olabilir. Şekil 3.7b'deki kontrol akışını düşünün. Daha önce olduğu gibi, kullanıcı programı bir WRITE çağrı�ı şeklinde bir sistem çağrı�ı yaptığı bir noktaya ulaşır. Bu durumda çağrılan G/C programı yalnızca hazırlık kodu ve gerçek G/C komutundan oluşur. Bu birkaç komut yerine getirildikten sonra kontrol kullanıcı programına geri döner. Bu sırada harici cihaz bilgisayar belleğinden veri almak ve yazdırmağa meşguldür. Bu G/C işlemi, kullanıcı programındaki talimatların yürütülmesiyle eşzamanlı olarak gerçekleştirilir.

Harici cihaz hizmete hazır hale geldiğinde, yani işlemciden daha fazla veri kabul etmeye hazır olduğunda, bu harici cihazın I/O modülü işlemciye bir *kesme isteği* sinyali gönderir. İşlemci, mevcut programın çalışmasını askıya alarak, **kesme işleyicisi** olarak bilinen söz konusu G/C aygitina hizmet verecek bir programa dalarak ve aygıtta hizmet verildikten sonra orijinal yürütümeye devam ederek yanıt verir. Bu tür kesintilerin meydana geldiği noktalar Şekil 3.7b'de yıldız işaretiley gösterilmiştir.

Şekil 3.7'de neler olduğunu açıklığa kavuşturtmaya çalışalım. İki WRITE komutu içeren bir kullanıcı programımız var. Başlangıçta bir kod parçası, sonra bir WRITE komutu, sonra ikinci bir kod parçası, sonra ikinci bir komutu, sonra üçüncü ve son bir kod parçası vardır. WRITE komutu işletim sistemi tarafından sağlanan I/O programını çağrıır. Benzer şekilde, G/C programı bir kod parçasından, ardından bir G/C komutundan ve ardından başka bir kod parçasından oluşur. G/C komutu bir donanım G/C işlemini çağrıır.

#### USER PROGRAM

```
{statement}  
{statement}  
:  
{statement}
```

Code segment 1

WRITE

```
{statement}  
{statement}  
:  
{statement}
```

Code segment 2

WRITE

```
{statement}  
{statement}  
:  
{statement}
```

Code segment 3

#### I/O PROGRAM

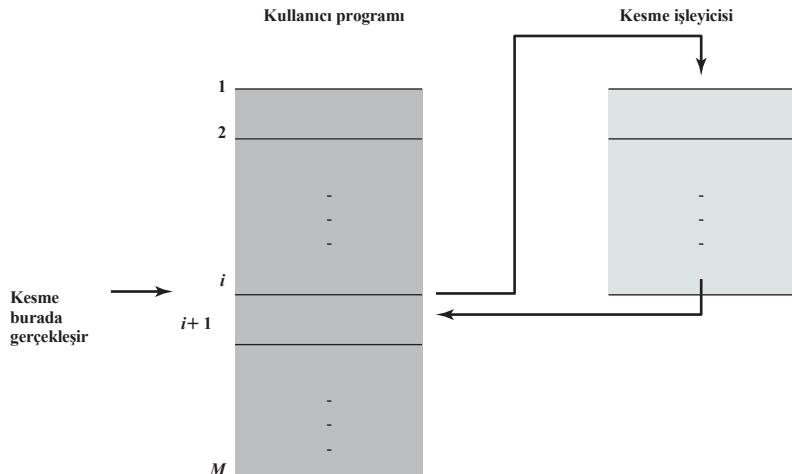
```
{statement}  
{statement}  
:  
{statement}
```

Code segment 4

#### I/O command

```
{statement}  
{statement}  
:  
{statement}
```

Code segment 5

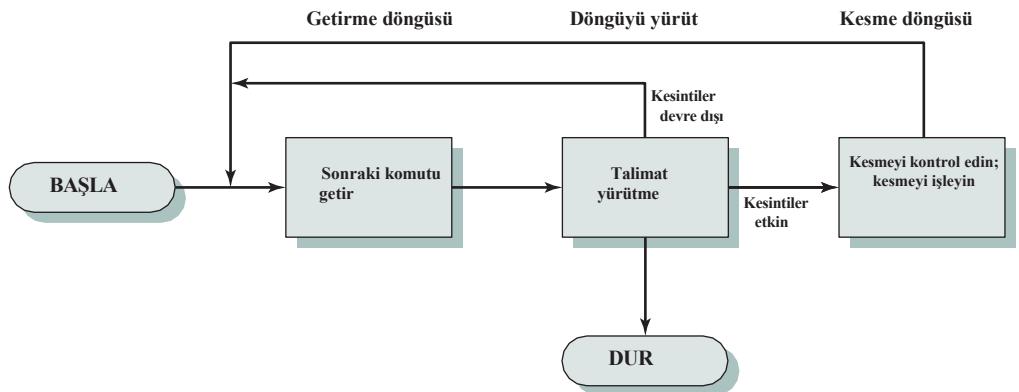


Şekil 3.8 Kesmeler Aracılığıyla Kontrol Aktarımı

Kullanıcı programının bakış açısından, bir kesme sadece şudur: normal yürütme sırasının kesilmesi. Kesme işlemi tamamlandığında, yürütme devam eder (Şekil 3.8). Bu nedenle, kullanıcı programının kesmeleri karşılamak için herhangi bir özel kod içermesi gerekmeyez; işlemci ve işletim sistemi kullanıcı programını askıya almaktan ve ardından aynı noktada yeniden başlatmaktan sorumludur.

Kesmeleri karşılamak için, Şekil 3.9'da gösterildiği gibi komut döngüsüne bir *kesme* döngüsü eklenir. Kesme işlemci, bir kesme sinyalinin varlığıyla gösterilen herhangi bir kesmelenin meydana gelip gelmediğini kontrol eder. Bekleyen bir kesme yoksa, işlemci getirme döngüsüne geçer ve mevcut programın bir sonraki komutunu getirir. Bir kesme beklemedeyse, işlemci aşağıdakileri yapar:

- Yürütmekte olan mevcut programın yürütülmesini askıya alır ve bağlamını kaydeder. Bu, yürütülecek bir sonraki komutun adresinin kaydedilmesi anlamına gelir



Şekil 3.9 Kesmeli Komut Döngüsü

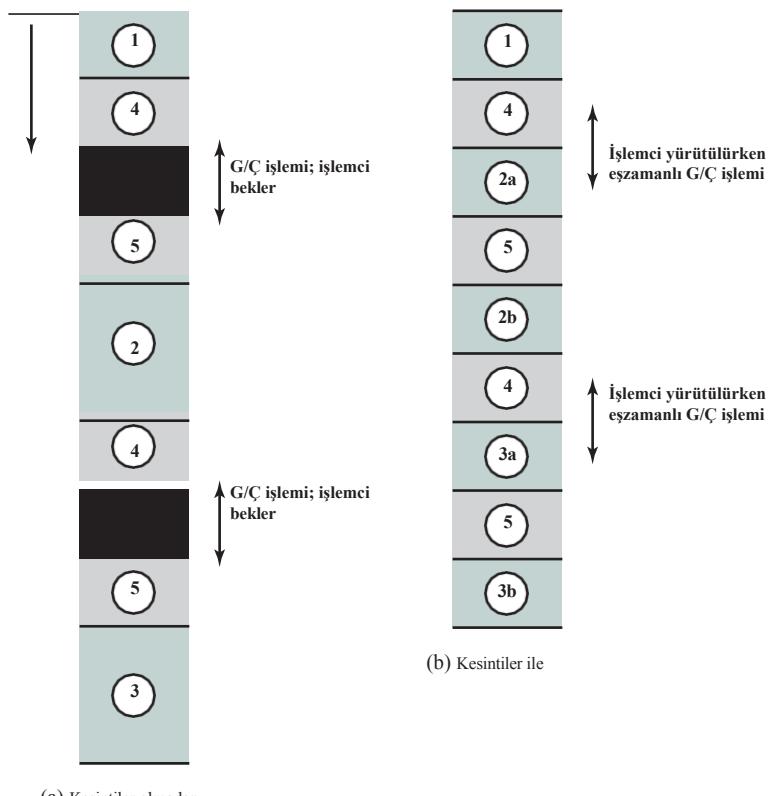
(program sayacının mevcut içeriği) ve işlemcinin mevcut faaliyetiyle ilgili diğer veriler.

- Program sayacını bir *kesme işleyici* rutininin başlangıç adresine ayarlar.

İşlemci şimdi getirme döngüsüne geçer ve kesmeye hizmet edecek olan kesme işleyici programındaki ilk komutu getirir. Kesme işleyici programı genellikle işletim sisteminin bir parçasıdır. Tipik olarak, bu program kesmenin nitelğini belirler ve gereken eylemleri gerçekleştirir. Kullandığımızörnekte, işleyici hangi I/O modülünün kesmeyi oluşturduğunu belirler ve bu I/O modülüne daha fazla veri yazacak bir programa dallanabilir. Kesinti işleyici rutini tamamlandığında, işlemci kesinti noktasında kullanıcı programının yürütülmesine devam edebilir.

Bu süreçte bir miktar ek yük olduğu açıklar. Kesintinin nitelini belirlemek ve uygun eyleme karar vermek için ekstra talimatların (kesme işleyicisinde) yürütülmesi gereklidir. Bununla birlikte, sadece bir G/C işlemini bekleyerek boş harcanacak nispeten büyük zaman miktarı nedeniyle, işlemci kesmelerin kullanımıyla çok daha verimli bir şekilde kullanılabilir.

Verimlilikteki kazancı anlamak için, Şekil 3.7a ve 3.7b'deki kontrol akışına dayalı bir zamanlama diyagramı olan Şekil 3.10'u düşünün. Bu şekilde, kullanıcı program kodu bölümleri yeşil gölgeli ve G/C program kodu bölümleri

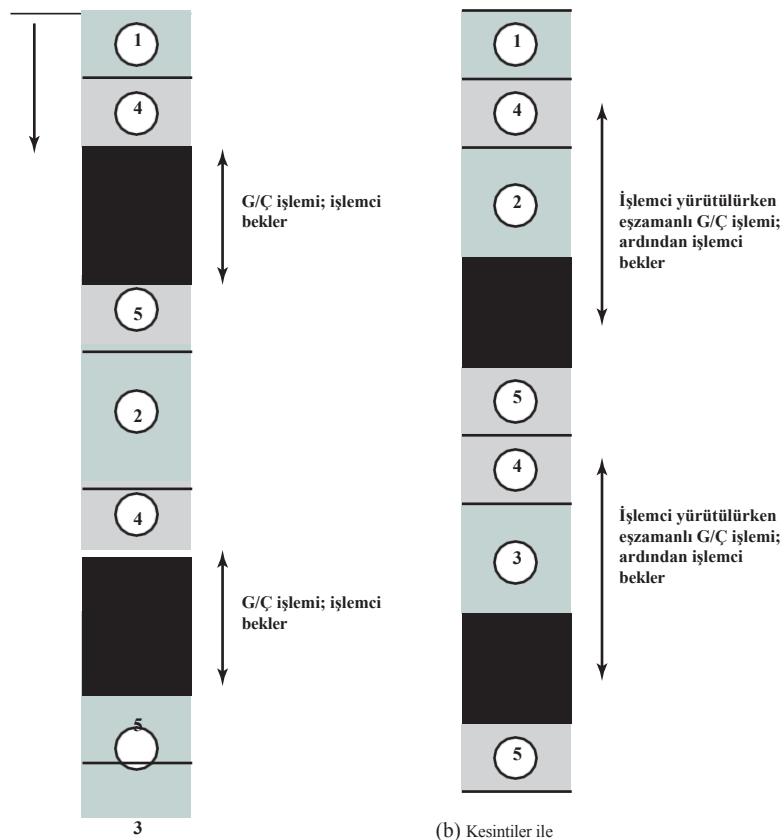


Şekil 3.10 Program Zamanlaması: Kısa G/C Beklemesi

gri gölgeli. Şekil 3.10a, kesmelerin kullanılmadığı durumu göstermektedir. Bir G/C işlemi gerçekleştirileşirken işlemci beklemelidir.

Şekil 3.7b ve 3.10b, G/C işlemi için gereken sürenin oldukça kısa olduğunu varsaymaktadır: kullanıcı programındaki yazma işlemleri arasındaki taliimatların yürütülmesini tamamlama süresinden daha az. Bu durumda, kod segmenti 2 olarak etiketlenen kod segmenti kesintiye uğrar. Kodun bir kısmı (2a) yürütülür (G/C işlemi gerçekleştirileşirken) ve ardından kesinti meydana gelir (G/C işleminin tamamlanmasının ardından). Kesinti servis edildikten sonra, yürütme kod segmenti 2'nin geri kalımıyla 2b) devam eder.

Özellikle yazıcı gibi yavaş bir cihaz için daha tipik durum, G/C işleminin bir dizi kullanıcı taliyatını yürütmekten çok daha fazla zaman almasıdır. Şekil 3.7c bu göstermektedir. Bu durumda, kullanıcı programı, ilk çağrı tarafından başlatılan G/C işlemi tamamlanmadan önce ikinci WRITE çağrısına ulaşır. Sonuç olarak kullanıcı programı bu noktada kapatılır. Önceki G/C işlemi tamamlandığında, bu yeni WRITE çağrı işlenebilir ve yeni bir G/C işlemi başlatılabilir. Şekil 3.11'de bu durum için zamanlama gösterilmektedir.



**Şekil 3.11** Program Zamanlaması: Uzun G/C Beklemesi

ve kesmeler kullanılmadan. G/C işleminin devam ettiği sürenin bir kısmı kullanıcı talimatlarının yürütülmesiyle çakıştığı için verimlilikte hala bir kazanç olduğunu görebiliriz.

Şekil 3.12, kesmeler arası döngü işlemeyi içeren revize edilmiş bir komut döngüsü durum diyagramını göstermektedir.

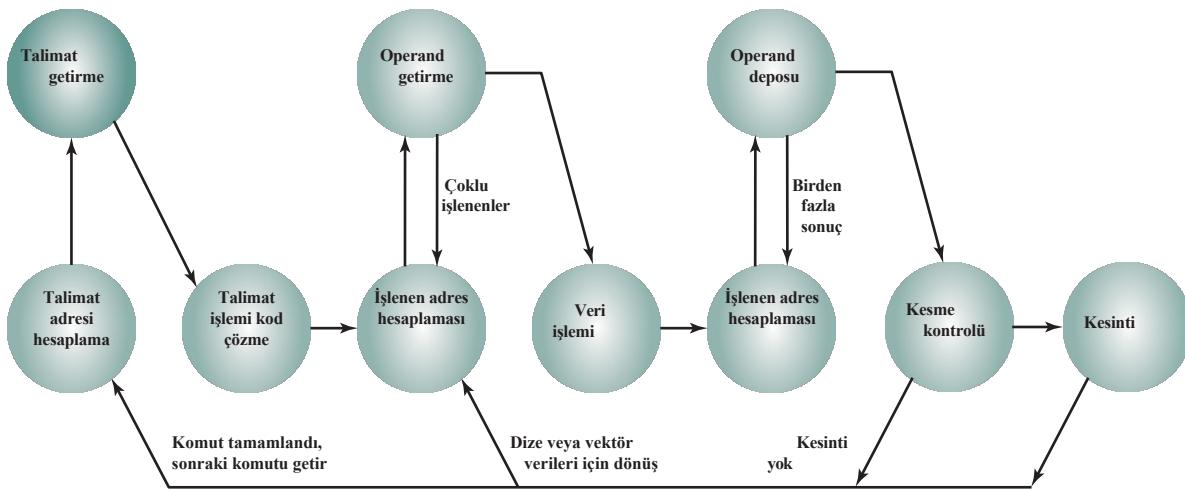
**ÇOKLU KESMELER** Şimdiye kadarki tartışma sadece tek bir kesmenin oluşmasına odaklanmıştır. Ancak birden fazla kesmenin meydana gelebileceğini varsayılmı. Örneğin, bir program bir iletişim hattından veri alıyor ve sonuçları yazdırıyor olabilir. Yazıcı her yazdırma işlemini tamamladığında bir kesme oluşturacaktır. İletişim hattı denetleyicisi, bir veri birimi her geldiğinde bir kesme oluşturacaktır. Birim, iletişim disiplininin yapısına bağlı olarak tek bir karakter ya da bir blok olabilir. Her durumda, bir yazıcı kesmesi işlenirken bir iletişim kesmesinin oluşması mümkündür.

Çoklu kesmelerle başa çıkmak için iki yaklaşım benimsenebilir. Birincisi, bir kesme işlenirken kesmeleri devre dışı bırakmaktadır. **Devre dışı bırakılmış** bir kesme basitçe işlemcinin bu kesme isteği sinyalini yok söylemeyeceği ve yok olacağı anlamına gelir. Bu süre zarfında bir kesme oluşursa, genellikle beklemede kalır ve işlemci kesmeleri etkinleştirirdikten sonra işlemci tarafından kontrol edilir. Böylece, bir kullanıcı programı yürütülürken bir kesme meydana gelirse, kesmeler hemen devre dışı bırakılır. Kesme işleyici rutini tamamlandıktan sonra, kullanıcı programına devam etmeden önce kesmeler etkinleştirilir ve işlemci başka kesmelerin meydana gelip gelmediğini kontrol eder. Bu yaklaşım güzel ve basittir, çünkü kesmeler katı sıralı düzende ele alınır (Şekil 3.13a).

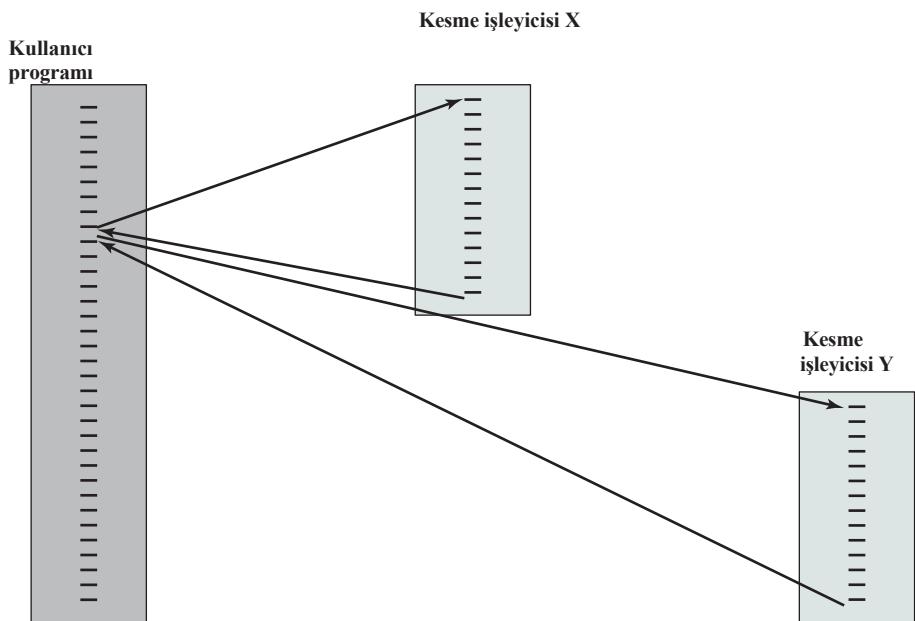
Önceki yaklaşımın dezavantajı, göreceli önceliği veya zaman açısından kritik ihtiyaçları dikkate almamasıdır. Örneğin, iletişim hattından girdi geldiğinde, daha fazla girdiye yer açmak için hızla emilmesi gerekebilir. İlk parti gelmeden önce ilk parti girdi işlenmemişse, veriler kaybolabilir.

İkinci bir yaklaşım, kesmeler için öncelikler tanımlamak ve daha yüksek öncelikli bir kesmenin daha düşük öncelikli bir kesme işleyicisinin kendisinin kesilmesine neden olmasına izin vermektedir (Şekil 3.13b). Bu ikinci yaklaşımı bir örnek olarak, üç G/C aygıtı olan bir sistem düşünün: sırasıyla 2, 4 ve 5 artan önceliklere sahip bir yazıcı, bir disk ve bir iletişim hattı. Şekil 3.14 olası bir sıralamayı göstermektedir. Bir kullanıcı programı  $t=0$  adresinde başlar.  $t=10$  adresinde bir yazıcı kesmesi meydana gelir; kullanıcı bilgileri sisteme yüklenir ve yürütme yazıcı kesme **hizmet rutininde (ISR)** devam eder. Bu rutin hala,  $t=15$  adresinde bir iletişim kesintisi meydana gelir. İletişim hattı yazıcıdan daha yüksek önceliğe sahip olduğundan, kesme onurlandırılır. Yazıcı ISR'si kesilir, durumu yükne itilir ve yürütme iletişim ISR'sinde devam eder. Bu rutin yürütülürken, bir disk kesmesi meydana gelir ( $t=20$ ). Bu kesme daha düşük önceliğe sahip olduğundan, basitçe bekletilir ve iletişim ISR'si tamamlanana kadar çalışır.

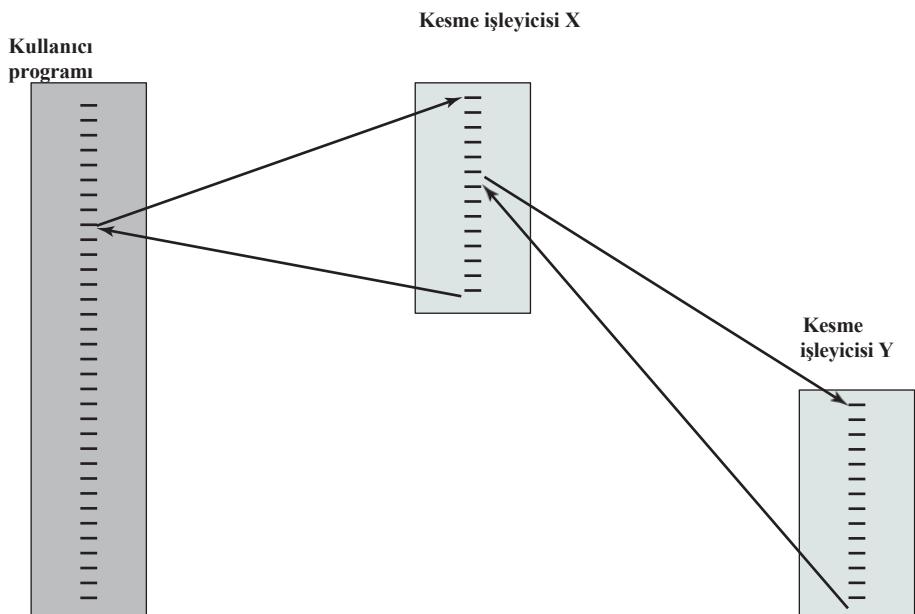
İletişim ISR'si tamamlandığında ( $t=25$ ), yazıcı ISR'sının yürütülmesi olan önceki işlemci durumuna geri dönülür. Ancak, bu rutindeki tek bir komut bile yürütülemeden önce, işlemci daha yüksek öncelikli disk kesmesini onurlandırır ve kontrol disk ISR'sine aktarılır. Sadece o zaman



Şekil 3.12 Kesmelerle Birlikte Komut Döngüsü Durum Diyagramı

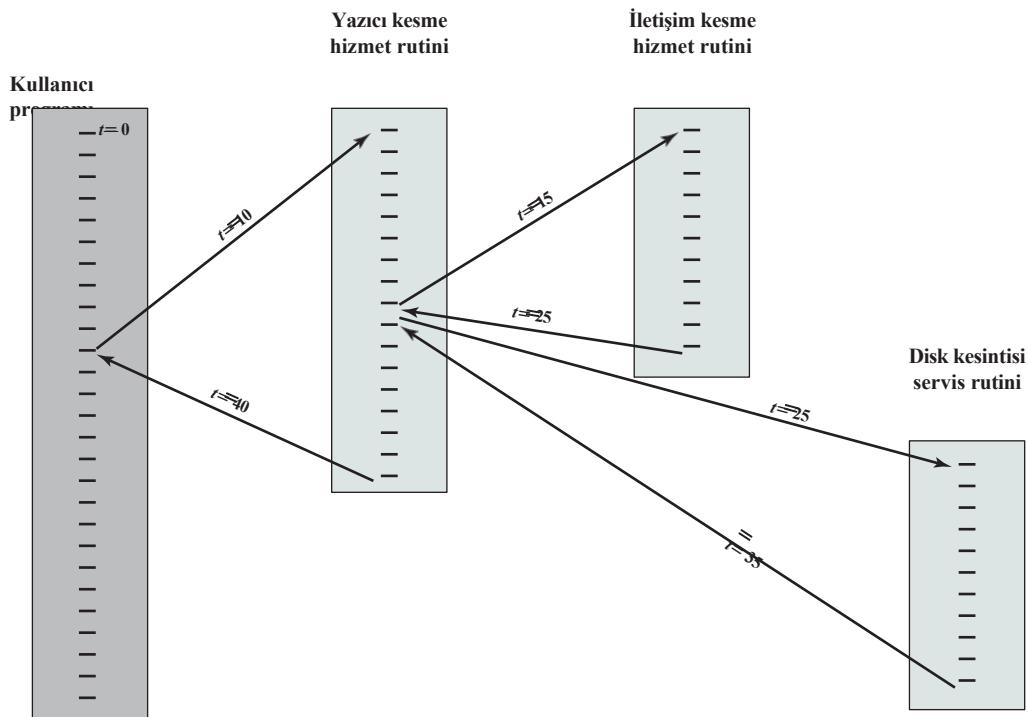


(a) Sıralı kesme işleme



(b) İç içe kesme işleme

**Şekil 3.13** Çoklu Kesintilerle Kontrol Aktarımı



Şekil 3.14 Çoklu Kesmelerin Örnek Zaman Sırası

rutini tamamlandığında ( $t= 35$ ) yazıcı ISR'si yeniden başlatılır. Bu rutin tamamlandığında ( $t= 40$ ), kontrol nihayet kullanıcı programına döner.

### I/O Fonksiyonu

Şimdije kadar, bilgisayarın işlemci tarafından kontrol edilen işleyişini tartışık ve öncelikle işlemci ile belleğin etkileşimine baktık. Tartışmada sadece I/O bileşeninin rolüne değinildi. Bu rol Bölüm 7'de ayrıntılı olarak ele alınmaktadır, ancak burada kısa bir özeti yapmak yerinde olacaktır.

Bir G/C modülü (örneğin bir disk denetleyicisi) doğrudan işlemciyle veri alışverişi yapabilir. İşlemcinin belirli bir konumun adresini belirleyerek bellekle bir okuma veya yazma işlemi başlatılabilmesi gibi, işlemci de bir G/C modülünden veri okuyabilir veya bir G/C modülüne veri yazabilir. Bu ikinci durumda, işlemci belirli bir G/C modülü tarafından kontrol edilen belirli bir cihazı tanımlar. Böylece, bellek referanslı talimatlar yerine G/C talimatları ile Şekil 3.5'tekine benzer bir talimat dizisi oluşturulabilir.

Bazı durumlarda, G/C değişimlerinin doğrudan bellekle gerçekleşmesine izin verilmesi istenebilir. Böyle bir durumda, işlemci bir G/C modülüne bellekten okuma veya belleğe yazma yetkisi verir, böylece G/C-bellek aktarımı işlemciyi bağlamadan gerçekleştirilebilir. Böyle bir aktarım sırasında, G/C modülü belleğe okuma ya da yazma komutları vererek işlemciyi değişim için sorumluluktan kurtarır. Bu işlem doğrudan bellek erişimi (DMA) olarak bilinir ve Bölüm 7'de incelenmiştir.

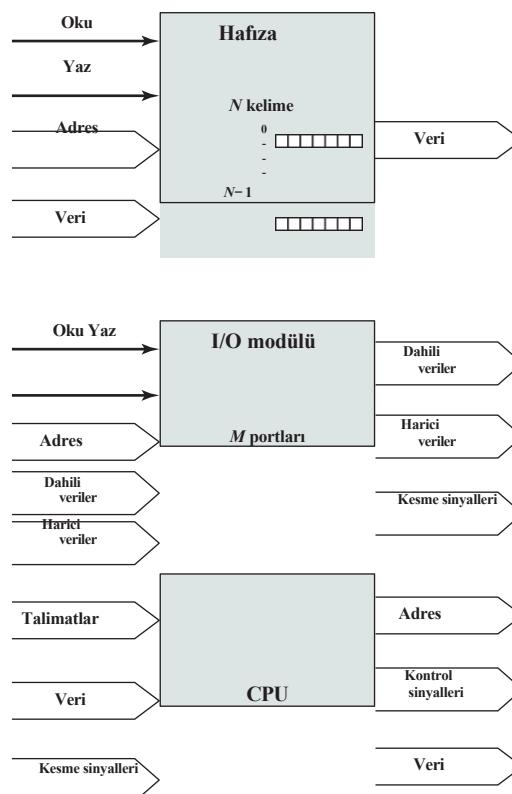
### 3.3 BAĞLANTI YAPILARI

Bir bilgisayar, birbirleriyle iletişim kuran üç temel tipte (işlemci, bellek, I/O) bir dizi bileşen veya modülinden oluşur. Aslında bir bilgisayar temel modüllerden oluşan bir ağdır. Bu nedenle, modülleri birbirine bağlamak için yollar olmalıdır.

Çeşitli modülleri birbirine bağlayan yollar topluluğuna *ara bağlantı yapısı* denir. Bu yapının tasarıımı, modüller arasında yapılması gereken alışverişlere bağlı olacaktır.

Şekil 3.15, her bir modül tipi için başlıca girdi ve çıktı biçimlerini göstererek ihtiyaç duyulan değişim türlerini ortaya koymaktadır<sup>2</sup>:

- **Bellek:** Tipik olarak, bir bellek modülü eşit uzunlukta  $N$  kelimedenden oluşur. Her kelimeye benzersiz bir sayısal adres atanır ( $0, 1, \dots, N - 1$ ). Bir kelime veri bellekten okunabilir veya belleğe yazılabilir. İşlemin doğası



Şekil 3.15 Bilgisayar Modülleri

<sup>2</sup> Geniş oklar, paralel olarak birden fazla bilgi biti taşıyan çoklu sinyal hatlarını temsil eder. Her dar ok tek bir sinyal hattını temsil eder.

okuma ve yazma kontrol sinyalleri ile gösterilir. İşlem için konum bir adres ile belirtilir.

- **G/Ç modülü:** Dahili (bilgisayar sistemine) bir bakış açısından, I/O işlevsel olarak belleğe benzer. İki işlem vardır; okuma ve yazma. Ayrıca, bir I/O modülü birden fazla harici cihazı kontrol edebilir. Harici bir cihaza giden arayüzlerin her birini bir *port* olarak adlandırabilir ve her birine benzersiz bir adres verebiliriz (örneğin, 0, 1,  $\leftarrow$ ,  $M - 1$ ). Buna ek olarak, harici bir cihazla veri girişi ve çıkışı için harici veri yolları vardır. Son olarak, bir I/O modülü işlemciye kesme sinyalleri gönderebilir.
- **İşlemci:** İşlemci talimatları ve verileri okur, işledikten sonra verileri yazar ve sistemin genel çalışmasını kontrol etmek için kontrol sinyallerini kullanır. Ayrıca kesme sinyallerini de alır.

Yukarıdaki liste değişim tokus edilecek verileri tanımlamaktadır. Ara bağlantı yapısı aşağıdaki aktarım türlerini desteklemelidir:

- **Bellekten işlemciye:** İşlemci bellekten bir komut veya veri birimi okur.
- **İşlemciden belleğe:** İşlemci belleğe bir veri birimi yazar.
- **G/Ç'den işlemciye:** İşlemci, bir I/O modülü aracılığıyla bir I/O cihazından veri okur.
- **İşlemciden G/Ç'ye:** İşlemci I/O cihazına veri gönderir.
- **Belleğe veya bellekten G/Ç:** Bu iki durum için, bir G/Ç modülünün doğrudan bellek erişimini kullanarak işlemciden geçmeden doğrudan bellekle veri alışverişi yapmasına izin verilir.

Yıllar boyunca bir dizi ara bağlantı yapısı denenmiştir. En yaygın olanları (1) veri **yolu** ve çeşitli çoklu veri **yolu** yapıları ve (2) paketlenmiş veri aktarımı ile noktadan noktaya ara bağlantı yapılarıdır. Bu bölümün geri kalanını bu yapıların tartışılmamasına ayıryoruz.

### 3.4 OTOBÜS BAĞLANTISI

Veri yolu, onlarca yıl boyunca bilgisayar sistemi bileşenlerinin birbirine bağlanmasında baskın bir araç olmuştur. Genel amaçlı bilgisayarlar için, yavaş yavaş yerini şu anda bilgisayar sistemi tasarıma hakim olan çeşitli noktadan noktaya ara bağlantı yapılarına bırakmıştır. Bununla birlikte, veri yolu yapıları gömülü sistemler, özellikle de mikro denetleyiciler için hala yaygın olarak kullanılmaktadır. Bu bölümde, veri yolu yapısına kısa bir genel bakış sunulmaktadır. Ek C daha fazla ayrıntı sağlanmaktadır.

Veri yolu, iki veya daha fazla cihazı birbirine bağlayan bir iletişim yoludur. Bir veri yolu temel özelliği, paylaşılan bir iletim ortamı olmasıdır. Veriyoluna birden fazla cihaz bağlanır ve herhangi bir cihaz tarafından iletlenen bir sinyal veriyoluna bağlı diğer tüm cihazlar tarafından alınabilir. Aynı zaman diliminde iki cihaz iletim yaparsa, sinyalleri üst üste biner ve bozulur. Bu nedenle, bir seferde yalnızca bir cihaz başarılı bir şekilde iletim yapabilir.

Tipik olarak, bir veri yolu birden fazla iletişim yolundan veya hattan oluşur. Her hat ikili 1 ve ikili 0'ı temsil eden sinyalleri iletibilir. Zaman içinde, bir dizi ikili rakam tek bir hat üzerinden iletilebilir. Birlikte ele alındığında, bir veri yolunun birkaç hattı ikili rakamları aynı anda (paralel olarak) iletmek için kullanılabilir. Örneğin, 8 bitlik bir veri birimi sekiz veri yolu hattı üzerinden iletilebilir.

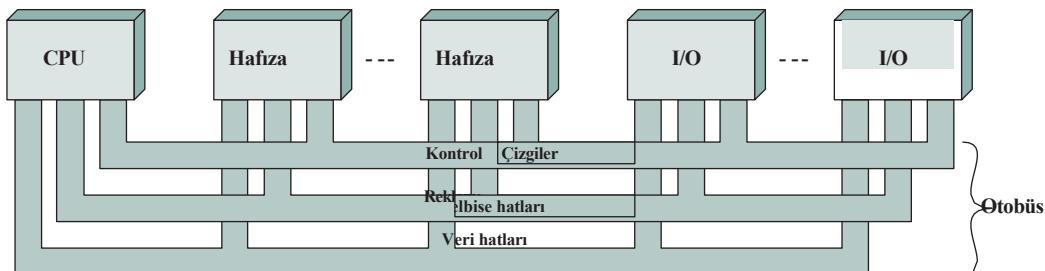
Bilgisayar sistemleri, bilgisayar sistemi hiyerarşisinin çeşitli seviyelerindeki bileşenler arasında yollar sağlayan bir dizi farklı veri yolu içerir. Ana bilgisayar bileşenlerini (işlemci, bellek, G/C) birbirine bağlayan veriyoluna **sistem veriyolu** denir. En yaygın bilgisayar ara bağlantı yapıları bir veya daha fazla sistem veriyolunun kullanımına dayanır.

Bir sistem veriyolu tipik olarak yaklaşık eli ila yüzlerce ayrı hattan oluşur. Her bir hatta belirli bir anlam ya da işlev atamıştır. Birçok farklı veri yolu tasarımını olmasına rağmen, herhangi bir veri yolunda hatlar üç işlevsel grupta sınıflandırılabilir (Şekil 3.16): veri, adres ve kontrol hatları. Buna ek olarak, bağlı modüllere güç sağlayarak güç dağıtım hatları da olabilir.

**Veri hatları**, sistem modülleri arasında veri taşımak için bir yol sağlar. Bu hatlar toplu olarak **veri yolu** olarak adlandırılır. Veri yolu 32, 64, 128 veya daha fazla ayrı hattan oluşabilir; hat sayısı veri yolunun *genişliği* olarak adlandırılır. Her hat bir seferde yalnızca bir bit taşıyabildiğinden, hat sayısı bir kaç bit aktarılabilceğini belirler. Veri yolunun genişliği, genel sistem performansının belirlenmesinde önemli bir faktördür. Örneğin veri yolu 32 bit genişliğindedeyse ve her komut 64 bit uzunluğundaysa, işlemcinin her komut döngüsü sırasında bellek modülüne iki kez erişmesi gereklidir.

**Adres hatları**, veri yolundaki verilerin kaynağını veya hedefini belirlemek için kullanılır. Örneğin, işlemci bellekten bir kelime (8, 16 veya 32 bit) veri okumak isterse, istenen kelimenin adresini adres hatlarına koyar. Açıksa, **adres yolunun genişliği** sistemin mümkün olan maksimum bellek kapasitesini belirler. Ayrıca, adres hatları genellikle I/O portlarını adreslemek için de kullanılır. Tipik olarak, yüksek sıralı bitler veri yolu üzerindeki belirli bir modülü seçmek için kullanılır ve düşük sıralı bitler modül içindeki bir bellek konumunu veya G/C portunu seçer. Örneğin, 8 bitlik bir adres veriyolunda, 01111111 ve altındaki adresler 128 kelimelik belleğe sahip bir bellek modülündeki (modül 0) konumlara referans verebilir ve 10000000 ve üstü adresler bir G/C modülüne (modül 1) bağlı aygıtları ifade eder.

**Kontrol hatları**, veri ve adres **hatlarına** erişimi ve bu hatların kullanımını kontrol etmek için kullanılır. Çünkü veri ve adres hatları tüm bileşenler tarafından paylaşılır,



Şekil 3.16 Bus Ara Bağlantı Şeması

kullanımlarını kontrol etmek için bir araç olmalıdır. Kontrol sinyalleri sistem modülleri arasında hem iletişim hem de zamanlama bilgilerini iletir. Zamanlama sinyalleri veri ve adres bilgilerinin geçerliliğini gösterir. Komut sinyalleri gerçekleştirilecek işlemleri belirtir. Tipik kontrol hatları şunları içerir:

- **Bellek yazma:** veri yolundaki verilerin adreslenen konuma yazılmasına neden olur.
- **Bellek okuma:** adreslenen konumdaki verilerin veri yoluna yerleştirilmesine neden olur.
- **G/C yazma:** veri yolundaki verilerin adreslenen G/C portuna çıkışmasına neden olur.
- **G/C okuma:** adreslenen G/C portundan gelen verilerin veri yoluna yerleştirilmesine neden olur.
- **Transfer ACK:** Verilerin veri yolundan kabul edildiğini veya veri yoluna yerleştirildiğini gösterir.
- **Veri yolu talebi:** bir modülün veri yolunun kontrolünü ele geçirmesi gerektiğini belirtir.
- **Veri yolu izni:** talep eden bir modüle veri yolunun kontrolünün verildiğini gösterir.
- **Kesme isteği:** bir kesmenin beklemeye olduğunu gösterir.
- **Interrupt ACK:** Bekleyen kesmenin tanındığını onaylar.
- **Saat:** işlemleri senkronize etmek için kullanılır.
- **Sıfırla:** tüm modüllerini başlatır.

Veri yolunun işleyişi aşağıdaki gibidir. Bir modül diğerine veri göndermek isterse, iki şey yapmalıdır: (1) veri kullanımını elde etmek ve (2) veri yolu üzerinden veri aktarmak. Eğer bir modül başka bir modülden veri talep etmek isterse, şunları yapmalıdır

(1) veri yolunun kullanımını elde etmek ve (2) uygun kontrol ve adres hatları üzerinden diğer modüle bir istek aktarmak. Daha sonra ikinci modülün veriyi göndemesini beklemelidir.

### 3.5 NOKTADAN NOKTAYA BAĞLANTI

Paylaşımı veri yolu mimarisini, on yıllar boyunca işlemci ve diğer bileşenler (bellek, G/C vb.) arasındaki ara bağlantıya yönelik standart yaklaşım olmuştur. Ancak geçici sistemler, paylaşımı veri yolları yerine giderek daha fazla noktadan noktaya ara bağlantıya güvenmektedir.

Veri yolundan noktadan noktaya ara bağlantıya geçişini tetikleyen temel neden, geniş senkron veri yollarının frekansının artmasıyla karşılaşılan elektriksel kısıtlamalardır. Giderek daha yüksek veri hızlarında, senkronizasyon ve tahkim işlevlerini zamanında yerine getirmek giderek daha zor hale gelmektedir. Ayrıca, tek bir çip üzerinde birden fazla işlemci ve önemli bellek içeren çok çekirdekli ciplerin ortaya çıkmasıyla, aynı çip üzerinde geleneksel paylaşımı veri yolu kullanımının, veri yolu veri hızını artırma ve işlemcilere ayak uydurmak için veri yolu gecikmesini azaltma zorluklarını büyütüğü görülmüştür. Paylaşımı veri yolu ile karşılaşıldığında, noktadan noktaya ara bağlantı daha düşük gecikme süresine, daha yüksek veri hızına ve daha iyi ölçeklenebilirliğe sahiptir.

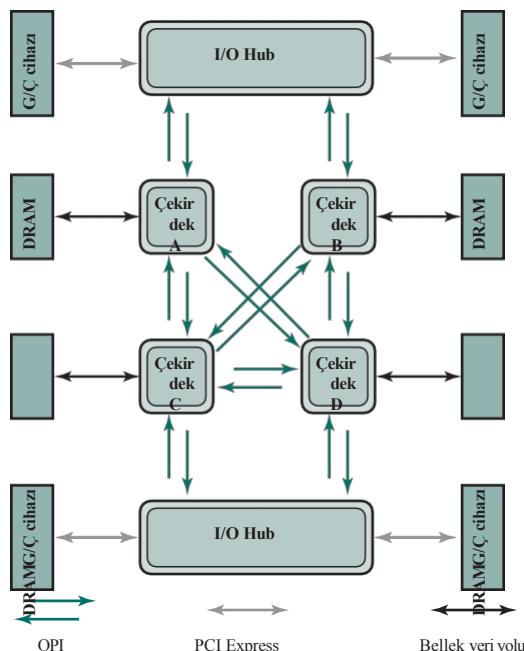
Bu bölümde, noktadan noktaya ara bağlantı yaklaşımının önemli ve temsili bir örneğine bakacağımız Intel'in 2008 yılında tanıttığı **QuickPath Interconnect (QPI)**.

Aşağıda QPI ve diğer noktadan noktaya ara bağlantı şemalarının önemli özellikleri yer almaktadır:

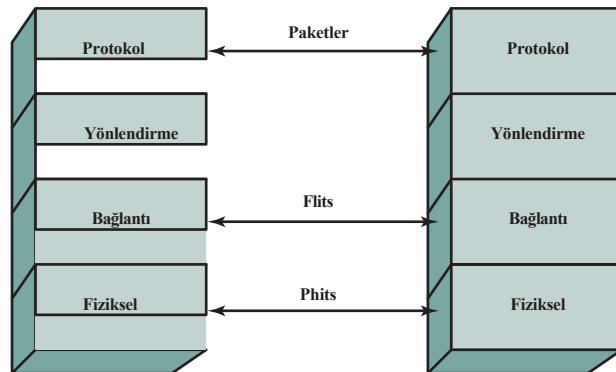
- **Çoklu doğrudan bağlantılar:** Sistem içindeki birden fazla bileşen, diğer doğrudan çift yönlü bağlantılarına sahiptir. Bu, paylaşımı iletim sistemlerinde bulunan tahkim ihtiyacını ortadan kaldırır.
- **Katmanlı protokol mimarisi:** TCP/IP tabanlı veri ağları ağ ortamlarında bulunduğu gibi, bu işlemci seviyesindeki ara bağlantılar, paylaşılan veri yolu düzenlemelerinde bulunan kontrol sinyallerinin basit kullanımı yerine katmanlı bir protokol mimarisi kullanır.
- **Paketlenmiş veri aktarımı:** Veriler ham bit akışı olarak gönderilmez. Bunun yerine, veriler her biri kontrol başlıklarları ve hata kontrol kodları içeren bir dizi paket olarak gönderilir.

Şekil 3.17'de çok çekirdekli bir bilgisayarda QPI'nin tipik bir kullanımı gösterilmektedir. QPI bağlantıları (şekilde yeşil ok çiftleriyle gösterilmiştir) verilerin ağ boyunca hareket etmesini sağlayan bir anahtarlama yapısı oluşturur. Her bir çekirdek işlemci çifti arasında doğrudan QPI bağlantıları kurulabilir. Şekil 3.17'deki A D çekirdeğindeki bellek denetleyicisine erişmesi gerekiyorsa, isteğiini B ya da C çekirdekleri üzerinden gönderir, bu çekirdekler de isteği D çekirdeğindeki bellek denetleyicisine iletir. Benzer şekilde, sekiz ya da daha fazla işlemciye daha büyük sistemler, üç bağlantılı işlemciler kullanılarak ve trafik ara işlemciler üzerinden yönlendirilerek oluşturulabilir.

Ayrıca QPI, I/O hub (IOH) adı verilen bir I/O modülüne bağlanmak için kullanılır. IOH, I/O cihazlarına giden ve gelen trafiği yönlendiren bir anahtar görevi görür. Tipik olarak daha yeni



**Şekil 3.17** QPI Kullanarak Çok Çekirdekli Yapılandırma



Şekil 3.18 QPI Katmanları

sistemlerinde, IOH'den I/O aygit denetleyicisine olan bağlantı, bu bölümün ilerleyen kısımlarında açıklanan PCI Express (PCIe) adlı bir ara bağlantı teknolojisini kullanır. IOH, QPI protokoller ve formatları ile PCIe protokoller ve formatları arasında çeviri yapar. Bir çekirdek ayrıca özel bir bellek veri yolu kullanarak bir ana bellek modülüne (tipik olarak bellek dinamik erişimli rastgele bellek (DRAM) teknolojisini kullanır) bağlanır.

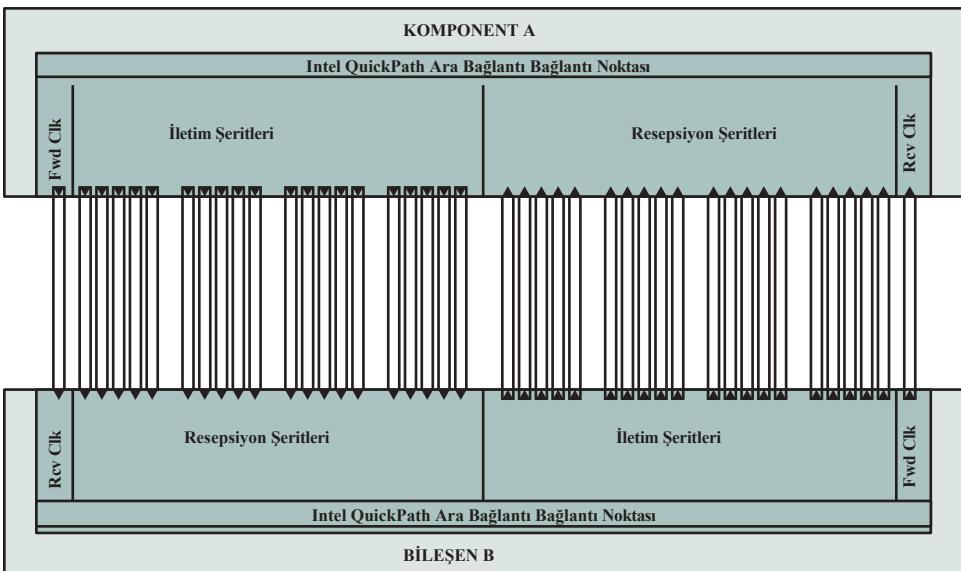
QPI, aşağıdaki katmanları kapsayan dört katmanlı bir protokol mimarisini<sup>3</sup> olarak tanımlanmaktadır (Şekil 3.18):

- **Fiziksel:** Sinyalleri taşıyan gerçek kabloların yanı sıra 1'lerin ve 0'ların iletilmesi ve alınması için gereken yardımcı özellikleri destekleyen devre ve mantıkta oluşur. Fiziksel katmandaki aktarım birimi 20 bittir ve buna **Phit** (fiziksel birim) denir.
- **Bağlantı:** Güvenilir iletim ve akış kontrolünden sorumludur. Link katmanının aktarım birimi 80 bitlik bir **Flit**'tir (akış kontrol birimi).
- **Yönlendirme:** Paketleri kumaş boyunca yönlendirmek için çerçeve sağlar.
- **Protokol:** Cihazlar arasında veri **paketlerinin** alışverişesi için üst düzey kurallar kümesi. Bir paket integral sayıda Flit'ten oluşur.

### QPI Fiziksel Katman

Şekil 3.19 bir QPI portunun fiziksel mimarisini göstermektedir. QPI portu aşağıdaki gibi gruplandırılmış 84 ayrı bağlantıdan oluşur. Her veri yolu, her seferinde bir bit veri iletken bir çift kablodan oluşur; çift, **şerit** olarak adlandırılır. Her yönde (gönderme ve alma) 20 veri şeridi ve ayrıca her yönde bir saat şeridi vardır. Böylece, QPI her yönde paralel olarak 20 bit iletебilir. 20 bitlik birim bir **phit** olarak adlandırılır. Mevcut ürünlerdeki bağlantıların tipik sinyalleşme hızları 6,4 GT/s'de (saniye başına transfer) çalışmayı gerektirir. Aktarım başına 20 bit ile bu 16 GB/s'ye ve QPI bağlantıları özel çift yönlü çiftler içerdiginden toplam kapasite 32 GB/s'dır.

<sup>3</sup> Protokol mimarisine aşıma olmayan okuyucu Ek D'de kısa bir genel bakış bulacaktır.



**Sekil 3.19** Intel OPI Ara Bağlantısının Fiziksel Arayüzü

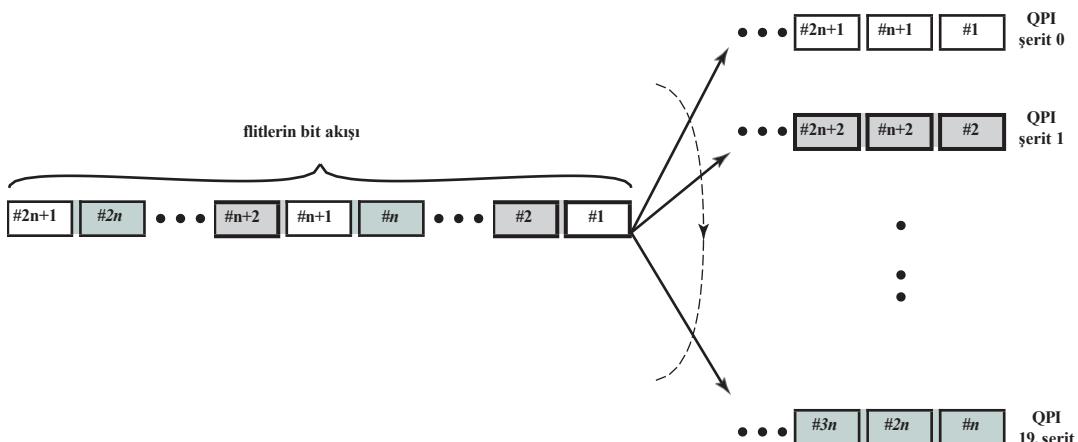
Her yöndeki şeritler, her biri 5 şeritten oluşan dört kadran halinde gruplandırılmıştır. Bazı uygulamalarda bağlantı, güç tüketimini azaltmak veya arızaları gidermek yarım veya çeyrek genişliklerde de çalışabilir.

Her bir hattaki iletişim şekli **diferansiyel sinyalizasyon** veya **dengeli iletişim** olarak bilinir. Dengeli iletişimde sinyaller bir iletkeninden aşağı inen ve diğerinden geri dönen bir akım olarak iletilir. İki tane değer voltaj farkına bağlıdır. Tipik olarak, bir hat pozitif voltaj değerine ve diğer hat sıfır voltaj değerine sahiptir ve bir hat ikili 1 ve bir hat ikili 0 ile ilişkilendirilir. Özellikle, QPI tarafından kullanılan teknik *disjunkt voltajlı diferansiyel sinyalleşme* (LVDS) olarak bilinir. Tipik bir uygulamada, aktarıcı, gönderilecek mantık seviyesine bağlı olarak bir kabloya veya diğeri küçük bir akım enjekte eder. Akım alıcı uçağı bir direnenin geçer ve daha sonra diğer tel boyunca ters yönde geri döner. Alıcı, mantık seviyesini belirlemek için direne üzerindeki voltajın kutuplarını algılar.

Fiziksel katman tarafından gerçekleştirilen bir diğer işlev **de çok şeritli dağıtım** olarak bilinen bir teknik kullanarak 80 bitlik bitler ve 20 bitlik bitler arasında geçiş yönetmesidir. Flitler, Şekil 3.20'de gösterildiği gibi, veri şeritleri boyunca yuvarlak robin şeklinde (ilk bit ilk şeride, ikinci bit ikinci , vb.) dağıtılan bir bit akışı olarak düşünülebilir. Bu yaklaşım, iki bağlantı noktası arasındaki fiziksel bağlantıyı çoklu paralel kanallar olarak uygulayarak QPI'nin çok yüksek veri hızlarına ulaşmasını sağlar.

QPI Bağlantı Katmanı

QPI bağlantı katmanı iki temel işlevi yerine getirir: akış kontrolü ve hata kontrolü. Bu işlevler QPI bağlantı katmanı protokolünün bir parçası olarak gerçekleştirilir ve flit (akış kontrol birimi) düzeyinde çalışır. Her bir flit 72 bitlik bir mesaj yükü ve



Şekil 3.20 QPI Çok Şeritli Dağılım

döngüsel artıklık kontrolü (CRC) adı verilen 8 bitlik bir hata kontrol kodu. Hata kontrol kodlarını Bölüm 5'te ele alacağız.

Bir flit yükü veri veya mesaj bilgisinden oluşabilir. Veri flitleri, gerçek veri bitlerini çekirdekler arasında veya bir çekirdek ile bir IOH arasında aktarır. Mesaj flitleri akış kontrolü, hata kontrolü ve önbellek tutarlılığı gibi işlevler için kullanılır. Önbellek tutarlılığını Bölüm 5 ve 17'de tartışıyoruz.

**Akış kontrol işlevi**, gönderen bir QPI varlığının, alıcının verileri işleyebileceğinden ve daha fazla gelen veri için arabellekleri temizleyebileceğinden daha hızlı veri göndererek alıcı bir QPI varlığını bunaltmamasını sağlamak için gereklidir. Veri akışını kontrol etmek için QPI bir kredi şeması kullanılır. Başlatma sırasında, bir göndericiye bir alıcıya flit göndermesi için belirli sayıda kredi verilir. Alıcıya her flit gönderildiğinde, gönderici kredi sayaçlarını bir kredi azaltır. Alıcıda bir tampon serbest bırakıldığında, o tampon için göndericiye bir kredi iade edilir. Böylece alıcı, verilerin QPI bağlantısı üzerinden iletilme hızını kontrol eder.

Bazen, fiziksel katmanda iletilen bir bit, gürültü veya başka bir olay nedeniyle aktarım sırasında değişir. Bağlantı katmanındaki **hata kontrol fonksiyonu** bu tür bit hatalarını tespit eder ve kurtarır ve böylece daha yüksek katmanların bit hatalarıyla karşılaşmasını engeller. Prosedür, A sisteminden B sistemine veri akışı için aşağıdaki gibi çalışır:

1. Belirtildiği gibi, her 80 bitlik flit 8 bitlik bir CRC alanı içerir. CRC, kalan 72 bitin değerinin bir fonksiyonudur. İletim sırasında, A her bir flit için bir CRC değeri hesaplar ve bu değeri flitin içine ekler.
2. Bir flit alındığında, B 72 bitlik yük için bir CRC değeri hesaplar ve bu değeri flitteki gelen CRC değerinin değerile karşılaştırır. İki CRC değeri eşleşmezse, bir hata tespit edilmiştir.
3. B bir hata tespit ettiğinde, flitin yeniden iletilmesi için A'ya bir istek gönderir. Ancak, A bir flit akışı göndermek için yeterli kredisi sahip olabileceğinden, hatalı flitten sonra ek flitterler iletilmiş ve

A yeniden iletim talebini almadan önce. Bu nedenle, talep A'nın hasarlı fliti ve sonraki tüm flitleri yedeklemesi ve yeniden iletmesi içindir.

### **QPI Yönlendirme Katmanı**

Yönlendirme katmanı, bir paketin mevcut sistem ara bağlantıları üzerinden geçeceği rotayı belirlemek için kullanılır. Yönlendirme tabloları ürün yazılımı tarafından tanımlanır ve bir paketin izleyebileceği olası yolları tanımlar. İki soketli bir platform gibi küçük yapılandırmalarda yönlendirme seçenekleri sınırlıdır ve yönlendirme tabloları oldukça basittir. Daha büyük sistemler için yönlendirme tablosu seçenekleri daha karmaşıktr ve (1) cihazların platforma nasıl, (2) sistem kaynaklarının nasıl böülümlendirildiğine ve (3) güvenilirlik olaylarının arızalı bir kaynak etrafında eşleme ile sonuçlanmasına bağlı olarak trafiği yönlendirme ve yeniden yönlendirme esnekliği .

### **QPI Protokol Katmanı**

Bu katmanda paket, aktarım birimi olarak tanımlanır. Paket içeriği tanımı, farklı pazar segmenti gereksinimlerini karşılamak içinizin verilen bir miktar esneklikle standartlaştırılmıştır. Bu seviyede gerçekleştirilen önemli bir işlev, birden fazla önbellekte tutulan ana bellek değerlerinin tutarlı olmasını sağlamakla ilgilenen bir önbellek tutarlılık protokolüdür. Tipik bir veri paketi yükü, bir önbelleğe veya önbellekten gönderilen bir veri bloğudur.

## **3.6 PCI ExPrESS**

**Çevresel bileşen ara bağlantısı (PCI)**, ara kat veya çevresel veri yolu olarak işlev görebilen popüler bir yüksek bant genişliğine sahip, işlemciinden bağımsız veri yoludur. Diğer yaygın veri yolu özellikleriyle karşılaşıldığında PCI, yüksek hızlı G/C alt sistemleri (örn. grafik ekran bağırstırıcıları, ağ ara yüz denetleyicileri ve disk denetleyicileri) için daha iyi sistem performansı sunar.

Intel, Pentium tabanlı sistemleri için 1990 yılında PCI üzerinde çalışmaya başladı. Intel kısa süre içinde tüm patentleri kamu mali olarak yayınladı ve PCI spesifikasyonlarının daha da geliştirilmesi ve uyumluluğunun sürdürülmesi için bir endüstri birligi olan PCI Özel İlgî Grubu'nun (SIG) kurulmasını teşvik etti. Sonuç olarak PCI yaygın olarak benimsenmiş ve kişisel bilgisayar, iş istasyonu ve sunucu sistemlerinde giderek artan bir kullanım alanı bulmuştur. Spesifikasyon kamu mali olduğundan ve mikroişlemci ve çevre birimi endüstrisinin geniş bir kesimi tarafından desteklendiğinden, farklı satıcılar tarafından üretilen PCI türleri uyumludur.

Önceki bölümlerde tartışılan sistem veriyolunda olduğu gibi, veriyolu tabanlı PCI düzeni, bağlı cihazların veri hızı taleplerine ayak uyduramamıştır. Buna bağlı olarak, **PCI Express (PCIe)** olarak bilinen yeni bir versiyon geliştirilmiştir. PCIe, QPI'da olduğu gibi, PCI gibi veri yolu tabanlı şemaların yerini alması amaçlanan noktaya bir ara bağlantı şemasıdır.

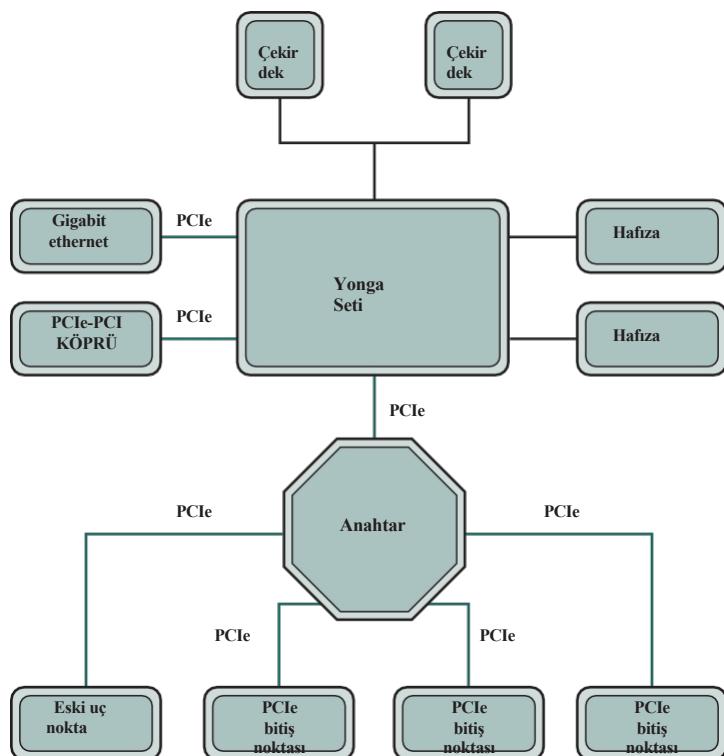
PCIe için temel bir gereksinim, Gigabit Ethernet gibi daha yüksek veri hızına sahip I/O cihazlarının ihtiyaçlarını desteklemek için yüksek kapasitedir. Bir başka gereksinim de zamana bağlı veri akışlarını destekleme ihtiyacıyla ilişkilidir. Talep üzerine video ve ses dağıtım gibi uygulamalar da sunuculara gerçek zamanlı kısıtlamalar getirmektedir. Birçok iletişim uygulaması ve gömülü PC kontrol sistemi de gerçek zamanlı olarak veri işlemektedir. Günümüz platformları aynı zamanda birden fazla eşzamanlı

sürekli artan veri hızlarında transferler. Artık tüm verilere eşit davranış kabul ; örneğin, geç gerçek zamanlı veriler hiç veri olmaması kadar yararsız olduğundan, önce akış verilerini işlemek daha önemlidir. Bir I/O sisteminin platform boyunca akışına öncelik verebilmesi için verilerin etiketlenmesi gereklidir.

### PCI Fiziksel ve Mantıksal Mimarisi

Şekil 3.21'de PCIe kullanımını destekleyen tipik bir yapılandırma gösterilmektedir. *Yonga seti* veya *ana bilgisayar köprüsü* olarak da adlandırılan bir **kök kompleks** cihazı, işlemci ve bellek alt sistemini bir veya daha fazla PCIe ve PCIe anahtar oluşturan PCI Express anahtar yapısına bağlar. Kök kompleksi, G/C denetleyicileri ile bellek ve işlemci bileşenleri arasındaki veri hızlarındaki farkla başa çıkmak için bir tamponlama cihazı görevi görür. Kök kompleks ayrıca PCIe işlem formatları ile işlemci ve bellek sinyal ve kontrol gereklilikleri arasında çeviri yapar. *Yonga seti* tipik olarak, bazıları doğrudan bir PCIe cihazına bağlanan birden fazla PCIe bağlantı noktasını ve birden fazla PCIe akışını yöneten bir anahtara bağlanan bir veya daha fazlasını destekleyecektir. *Yonga seti*nden gelen PCIe bağlantıları, PCIe uygulayan aşağıdaki cihaz türlerine bağlanabilir:

- **Anahtar:** Anahtar birden fazla PCIe akışını yönetir.
- **PCIe uç noktası:** Gigabit ethernet anahtarı, grafik veya video denetleyicisi, disk arabirimleri veya iletişim denetleyicisi gibi PCIe uyulayan G/C aygıtı veya denetleyicisi.



**Şekil 3.21** PCIe Kullanan Tipik Yapılandırma

■ **Eski uç nokta:** Eski uç nokta kategorisi, PCI Express'e geçirilmiş mevcut tasarımlar için tasarlanmıştır ve G/C alanı kullanımı ve kilitli işlemler gibi eski davranışlara izin verir. PCI Express uç noktalarının çalışma zamanında G/C alanı kullanımını gerektirmesine izin verilmey ve kilitli işlemler kullanmamalıdır. Bu kategorileri ayırt ederek, bir sistem tasarımcısının sistem performansı ve sağlamlığı üzerinde olumsuz etkileri olan eski davranışları kısıtlaması veya ortadan kaldırması mümkündür.

■ **PCIe/PCI köprüsü:** Eski PCI cihazlarının PCIe tabanlı sistemlere bağlanmasıyı sağlar.

QPI'da olduğu gibi, PCIe etkileşimleri de bir protokol mimarisi kullanılarak tanımlanır. PCIe protokol mimarisi aşağıdaki katmanları kapsar (Şekil 3.22):

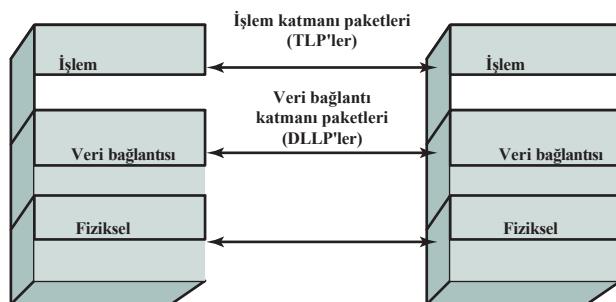
- **Fiziksel:** Sinyalleri taşıyan gerçek kabloların yanı sıra 1'lerin ve 0'ların iletilmesi ve alınması için gereken yardımcı özellikleri destekleyen devre ve mantıktan oluşur.
- **Veri bağlantısı:** Güvenilir iletim ve akış kontrolünden sorumludur. DLL tarafından üretilen ve tüketilen veri paketlerine Veri Bağlantısı Katmanı Paketleri (DLLPs) denir.
- **İşlem:** Yükle/depola veri aktarım mekanizmalarını uygulamak için kullanılan veri paketlerini üretir ve tüketir ve ayrıca bir bağlantı üzerindeki iki bileşen arasında bu paketlerin akış kontrolünü yönetir. TL tarafından üretilen ve tüketilen veri paketlerine İşlem Katmanı Paketleri (TLP'ler) denir.

TL'nin üzerinde, işlem katmanı tarafından paket tabanlı bir işlem protokolü kullanılarak G/C cihazlarına taşınan okuma ve yazma istekleri üreten yazılım katmanları bulunur.

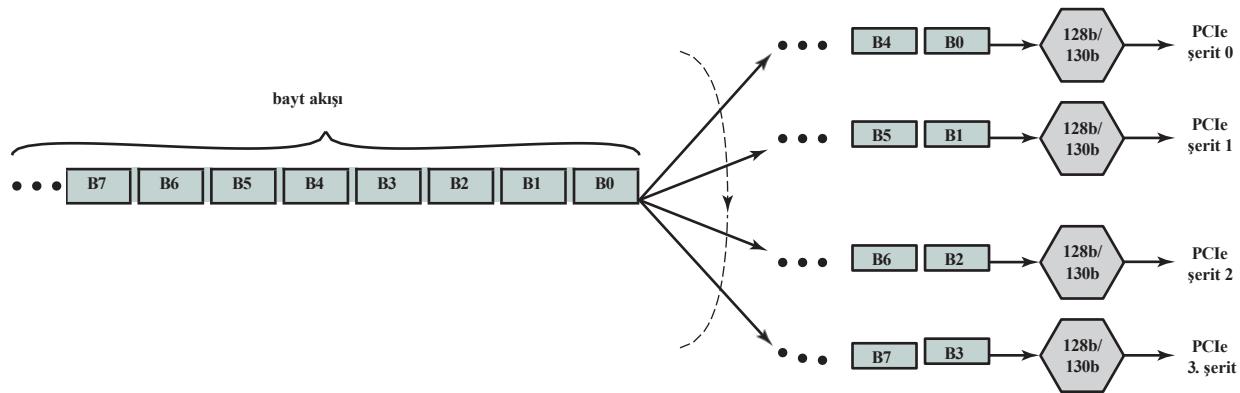
### PCIe Fiziksel Katmanı

QPI'a benzer şekilde, PCIe noktadan noktaya bir mimaridir. Her PCIe portu bir dizi çift yönlü şeritten oluşur (QPI'da şeridin yalnızca bir yönde aktarımı ifade ettiğini unutmayın). Bir şeritteki her yönde aktarım, bir çift kablo üzerinden diferansiyel sinyal yoluyla yapılır. Bir PCI portu 1, 4, 6, 16 ya da 32 şerit sağlayabilir. Aşağıda, 2010'un sonlarında tanıtılan PCIe 3.0 spesifikasyonuna atıfta bulunuyoruz.

QPI'da olduğu gibi, PCIe çok şeritli bir dağıtım tekniği kullanır. Şekil 3.23'te dört şeritten oluşan bir PCIe bağlantı noktası için bir örnek gösterilmektedir. Veriler dört şeride dağıtilır



Şekil 3.22 PCIe Protokol Katmanları



Şekil 3.23 PCIe Çok Kanallı Dağıtım

şeritleri basit bir round-robin şeması kullanarak her seferinde 1 bayt. Her bir fiziksel şeritte veriler arabelleğe alınır ve bir seferde 16 bayt (128 bit) işlenir. Her 128 bitlik blok, iletim için 130 bitlik benzersiz bir kod sözcüğünne kodlanır; buna 128b/130b kodlama denir. Böylece, tek bir şeridin etkin veri hızı 128/130 kat azalır.

128b/130b kodlamanın mantığını anlamak için, QPI'dan farklı olarak PCIe'nin bit akışını senkronize etmek için saat hattını kullanmadığını unutmayın. Yani, saat hattı gelen her bitin başlangıç ve bitiş noktasını belirlemek için kullanılmaz; sadece diğer sinyalizasyon amaçları için kullanılır. Ancak, alıcının verici ile senkronize olması gereklidir, böylece alıcı her bitin ne zaman başladığını ve bittigini bilir. Verici ve alıcının bit iletimi ve alımı için kullanılan saatleri arasında herhangi bir sapma varsa, hatalar meydana gelebilir. Sapma olasılığını telafi etmek için PCIe, alıcının iletilen sinyale dayalı olarak senkronize olmasına güvenir. QPI'da olduğu gibi, PCIe bir çift kablo üzerinden diferansiyel sinyalleşme kullanır. Senkronizasyon, alıcının verilerdeki geçişleri araması ve saatini geçişle senkronize etmesiyle sağlanabilir. Bununla birlikte, diferansiyel sinyalizasyon kullanan uzun bir 1'ler veya 0'lar dizisi ile çıkışın uzun bir süre boyunca sabit bir voltaj olduğunu düşünün. Bu koşullar altında, verici ve alıcının saatleri arasındaki herhangi bir kayma, ikisi arasındaki senkronizasyonu kaybolmasına neden olacaktır.

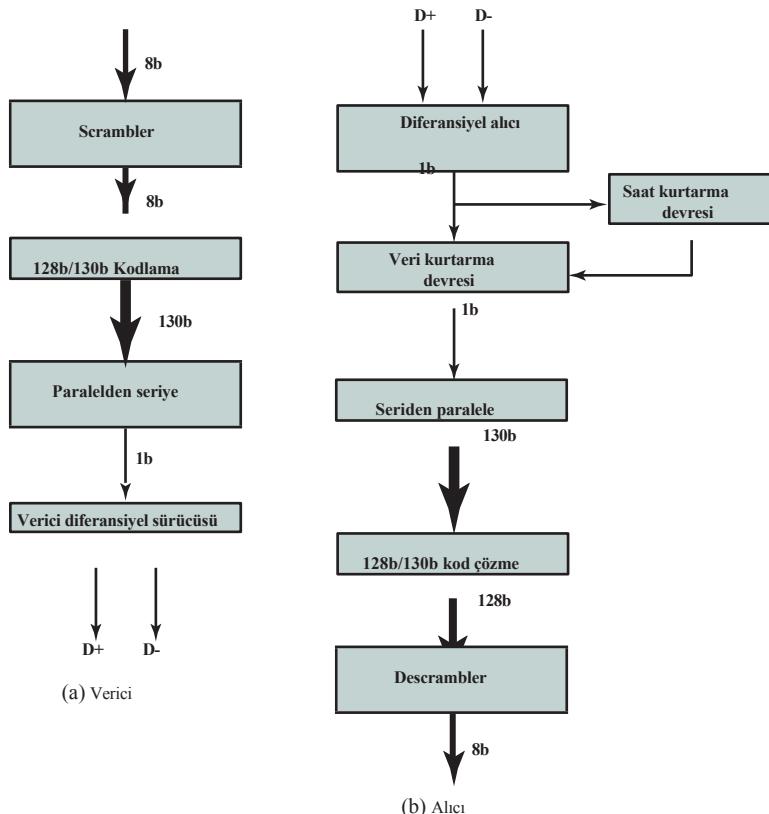
PCIe 3.0'da kullanılan ve tek bir değere sahip uzun bir bit dizisi sorununun üstesinden gelmek için yaygın bir yaklaşım karıştırmadır. İletilecek bit sayısını artırmayan karıştırma, verilerin daha rastgele görünümesini eğiliminde olan bir eşleme tekniğidir. Karıştırma, geçişlerin sayısını yama eğilimindedir, böylece alıcıda daha düzgün aralıklarla görünürler, bu da senkronizasyon için iyidir. Ayrıca, spektral özellikler gibi diğer iletim özellikleri, veriler sabit ya da tekrarlı ziyade neredeyse rastgele bir yapıya sahipse geliştirilir. Karıştırma hakkında daha fazla tartışma için Ek E'ye bakınız.

Senkronizasyona yardımcı olabilecek bir diğer teknik de geçişleri zorlamak için bit akışına ilave bitlerin eklendiği kodlamadır. PCIe 3.0 için 128 bitlik her giriş grubu, 2 bitlik bir blok senkronizasyon başlığı eklenecek 130 bitlik bir bloğa eşlenir. Başlığın değeri bir veri bloğu için 10 ve bağlantı düzeyinde bir bilgi bloğunu ifade eden *sıralı set bloğu* için 01'dir.

**Sekil 3.24** karıştırma ve kodlamanın kullanımını göstermektedir. Aktarılacak veriler bir beslenir. Karıştırılmış çıkış daha sonra 128 biti tamponlayan ve ardından 128 bitlik bloğu 130 bitlik bir bloğa eşleyen 128b/130b kodlayıcıya beslenir. Bu blok daha sonra bir paralel-seri dönüştürücüden geçer ve diferansiyel sinyalizasyon kullanılarak her seferinde bir bit ilettilir.

Alicıda, bit akışını kurtarmak için bir saat gelen veriyle senkronize edilir. Bu daha sonra 130 bitlik bloklardan oluşan bir akış üretmek için bir seri-paralel dönüştürücüden geçer. Her blok, orijinal karıştırılmış bit modelini kurtarmak için bir 128b/130b kod geçirilir ve daha sonra orijinal bit akışını üretmek için çözülür.

Bu teknikler kullanılarak 16 GB/s'lik bir veri hızına ulaşılabilir. Bahsedilmesi gereken son bir ayrıntı; bir PCI bağlantısı üzerinden bir veri bloğunun her iletimi, alıcıya gelen fiziksel katman bit akışıyla senkronize olması zaman vermeyi amaçlayan 8 bitlik bir çerçeveleme dizisiyle başlar ve biter.



Şekil 3.24 PCIe İletim ve Alım Blok Diyagramları

### PCIe İşlem Katmanı

İşlem katmanı (TL), TL'nin üzerindeki yazılımdan okuma ve yazma istekleri alır ve bağlantı katmanı üzerinden bir hedefe iletilmek üzere istek paketleri oluşturur. Çoğu işlem, aşağıdaki şekilde çalışan *bölünmüş işlem* teknğini kullanır. Kaynak PCIe cihazı tarafından bir istek paketi gönderilir ve ardından *tamamlama* paketi adı verilen bir yanıt beklenir. Bir isteği takip eden tamamlama, tamamlayıcı tarafından yalnızca teslimata hazır veri veya duruma sahip olduğunda başlatılır. Her paketin, tamamlama paketlerinin doğru kaynak kişiye yönlendirilmesini sağlayan benzersiz bir tanımlayıcısı vardır. Bölünmüş işlem teknigi ile tamamlama, bir işlemin her iki tarafının da veriyolunu ele geçirmek ve kullanmak için hazır olması gereken tipik bir veriyolu işleminin aksine, istekten zaman olarak ayırdır. İstek ve tamamlama arasında diğer PCIe trafiği bağlantılı kullanabilir.

TL mesajları ve bazı yazma *işlemleri gönderilmiş işlemlerdir*, yani yanıt beklenmez.

TL paket biçimi 32 bit bellek adreslemesini ve genişletilmiş 64 bit bellek adreslemesini destekler. Paketler ayrıca "no-snoop" gibi niteliklere de sahiptir.

"relaxed ordering," ve "priority," bu paketleri G/Ç alt sistemi üzerinden en iyi şekilde yönlendirmek için kullanılabilir.

#### **ADRES ALANLARI VE AKTARIM TİPLERİ** TL dört adres alanını destekler:

- **Bellek:** Bellek alanı sistem ana belleğini içerir. Ayrıca PCIe I/O aygıtlarını da içerir. Bellek adreslerinin belirli aralıkları G/Ç aygıtlarıyla eşleşir.
- **G/Ç:** Bu adres alanı, eski I/O cihazlarını adreslemek için kullanılan ayrılmış bellek adres aralıkları ile eski PCI cihazları için kullanılır.
- **Konfigürasyon:** Bu adres alanı, TL'nin G/Ç aygıtlarıyla ilişkili yapılandırma kayıtlarını okumasını/yazmasını sağlar.
- **Mesaj:** Bu adres alanı kesmeler, hata işleme ve güç yönetimi ile ilgili kontrol sinyalleri içindir.

Tablo 3.2 TL tarafından sağlanan işlem tiplerini göstermektedir. Bellek, G/Ç ve yapılandırma adres alanları için okuma ve yazma işlemleri vardır. Bellek işlemleri söz konusu olduğunda, bir okuma kildi isteği işlevi de vardır. Kilitli işlemleri, bir PCIe aygıtındaki kayıtlara atomik erişim talep eden aygit sürücülerinin bir sonucu olarak ortaya çıkar. Örneğin bir aygit sürücüsü, bir aygit kaydını atomik olarak okuyabilir, değiştirebilir ve ardından yazabilir. Bunu gerçekleştirmek için aygit sürücüsü işlemcinin bir talimat ya da talimat kümesi yürütmesine neden olur. Kök kompleks, bu işlemci talimatlarını, aygit sürücüsü için ayrı ayrı okuma ve yazma istekleri gerçekleştiren bir dizi PCIe işlemine dönüştürür. Bu işlemlerin atomik olarak yürütülmesi gerekiyorsa, kök kompleksi işlemleri yürütürken PCIe bağlantısını kilitler. Bu kilitleme, dizinin parçası olmayan işlemlerin gerçekleşmesini engeller. Bu işlem dizisine kilitli işlem adı verilir. Belirli bir set

**Tablo 3.2** PCIe TLP İşlem Türleri

Adres Alanı	TLP Tipi	Amaç
Hafıza	Bellek Okuma İsteği	Sistem bellek haritasındaki bir konuma veya konumdan veri aktarın.
	Bellek Okuma Kılıdi İsteği	
	Bellek Yazma İsteği	
I/O	G/Ç Okuma İsteği	Eski cihazlar için sistem bellek haritasındaki bir konuma veya konumdan veri aktarın.
	G/Ç Yazma İsteği	
Konfigürasyon	Konfigürasyon Tipi 0 Okuma Talebi	Bir PCIe cihazının yapılandırma alanındaki bir konuma veya konumdan veri aktarımı.
	Konfigürasyon Tipi 0 Yazma İsteği	
	Konfigürasyon Tipi 1 Okuma Talebi	
	Konfigürasyon Tipi 1 Yazma İsteği	
Mesaj	Mesaj Talebi	Bant içi mesajlaşma ve olay raporlaması sağlar.
	Veri İceren Mesaj Talebi	
Bellek, I/O, Yapılandırma	Tamamlama	Belirli talepler için iade edildi.
	Verilerle Tamamlama	
	Tamamlama Kilitli	
	Tamamlama Verilerle Kilitlendi	

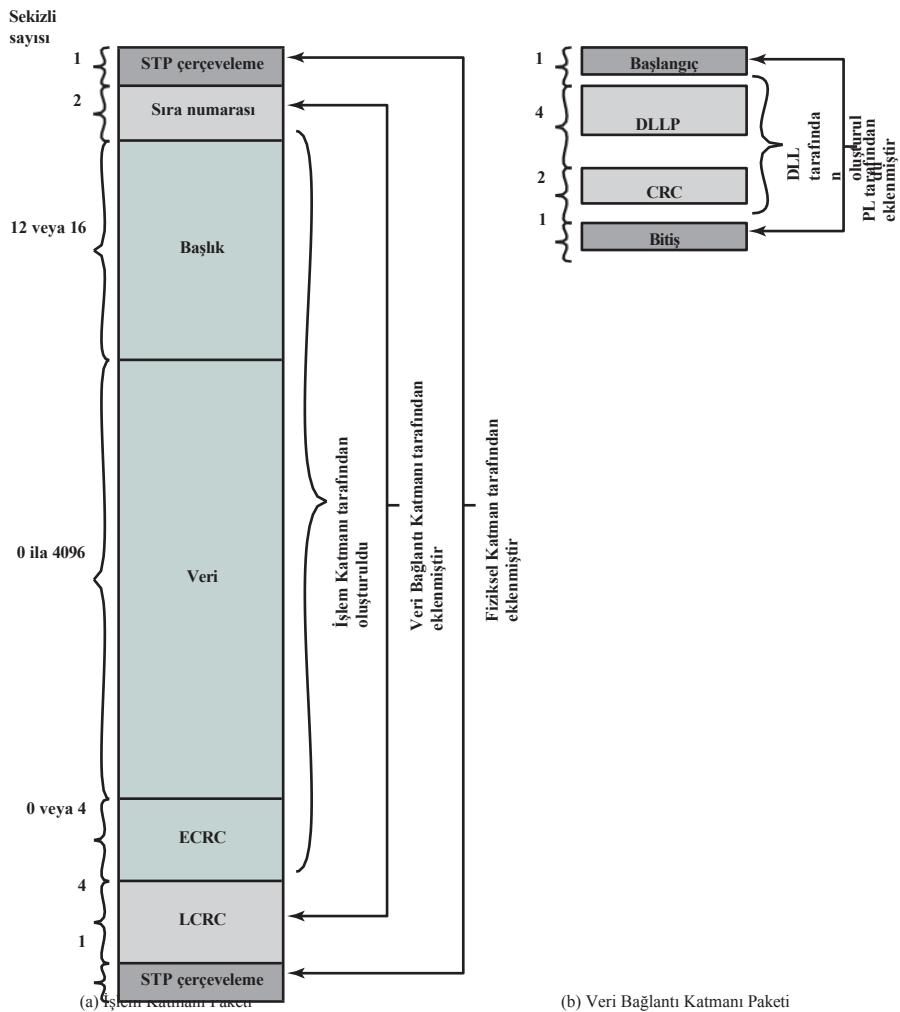
## 114 BÖLÜM3/BİLGİSAYAR İŞLEVİNİN VE BAĞLANTISININ ÜST DÜZEY GÖRÜNÜMÜ

Kilitli bir işlemin gerçekleşmesine neden olabilecek işlemci talimatlarının sayısı sistem çip setine ve işlemci mimarisine bağlıdır.

PCI ile uyumluluğu korumak için PCIe hem Tip 0 hem de Tip 1 yapılandırma döngülerini destekler. Bir Tip 1 döngüsü, hedef cihazın veri yolunu (bağlantı) barındıran köprü arayüzüne ulaşana kadar aşağı doğru yayılır. Yapılandırma işlemi hedef bağlantida köprü tarafından Tip 1'den Tip 0'a dönüştürülür.

Son olarak, tamamlama mesajları bellek, G/C ve yapılandırma işlemleri için bölünmüş işlemlerle birlikte kullanılır.

**TLP PaCKET** MONTAJI PCIe işlemleri, Şekil 3.25a'da gösterilen işlem katmanı paketleri kullanılarak iletilir. Bir TLP, gönderen cihazın işlem katmanında başlar ve alıcı cihazın işlem katmanında sonlanır.



Şekil 3.25 PCIe Protokolü Veri Birimi Formatı

Üst katman yazılımı, TL'nin TLP'nin çekirdeğini oluşturması için gerekli olan ve aşağıdaki alanlardan oluşan bilgileri TL'ye gönderir:

- **Başlık:** Başlık, paketin türünü açıklar ve gerekli yönlendirme bilgileri de dahil olmak üzere alıcıının paketi işlemek için ihtiyaç duyduğu bilgileri içerir. Dahili başlık formатı daha sonra ele alınacaktır.
- **Veri:** TLP'ye 4096 bayta kadar bir veri alanı dahil edilebilir. Bazı TLP'ler bir veri alanı içermez.
- **ECRC:** İsteğe bağlı bir uçtan uca CRC alanı, hedef TL katmanının TLP'nin başlık ve veri bölümlerindeki hataları kontrol etmesini sağlar.

## PCIe Veri Bağlantı Katmanı

PCIe veri bağlantı katmanının amacı, PCIe boyunca paketlerin güvenilir bir şekilde teslim edilmesini sağlamaktır. DLL, TLP'lerin oluşumuna katılır ve ayrıca DLLP'leri aktarır.

**VERİ BAĞLANTISI KATMANI PAKETLERİ** Veri bağlantısı katmanı paketleri, iletken bir aygıtın veri bağlantısı katmanında başlar ve bağlantının diğer ucundaki aygıtın DLL'sinde sonlanır. Şekil 3.25b bir DLLP'nin formatını göstermektedir. Bir bağlantının yönetiminde kullanılan üç önemli DLLP grubu vardır: akış kontrol paketleri, güç yönetim paketleri ve TLP ACK ve NAK paketleri. Güç yönetimi paketleri güç platformu bütçelememesini yönetmek için kullanılır. Akış kontrol paketleri, TLP'lerin ve DLLP'lerin bir bağlantı üzerinden iletilme hızını düzenler. ACK ve NAK paketleri, aşağıdaki paragraflarda ele alınan TLP işlemeye kullanılır.

**TRANSACTION LAYER PACKET PROCESSING** DLL, TL tarafından oluşturulan TLP'nin çekirdeğine iki alan ekler (Şekil 3.25a): 16 bit sıra numarası ve 32 bit bağlantı katmanı CRC'si (LCRC). TL'de oluşturulan çekirdek alanları yalnızca hedef TL' kullanılırken, DLL tarafından eklenen iki alan kaynaktan hedefe giden yoldaki her ara düğümde işlenir

Bir TLP bir cihaza ulaştığında, DLL sıra numarası ve LCRC alanlarını çıkarır ve LCRC'yi kontrol eder. İki olasılık vardır:

1. Herhangi bir hata tespit edilmezse, TLP'nin çekirdek kısmı yerel işlem katmanına ilettilir. Eğer bu alıcı cihaz amaçlanan hedef ise, TLP TLP'yi işler. Aksi takdirde TLP için bir rota belirler ve hedefe giden yolda bir sonraki bağlantı üzerinden iletilmek üzere DLL'ye geri gönderilir.
2. Bir hata tespit edilirse, DLL uzak vericiye geri dönmek için bir NAK DLL paketi planlar. TLP ortadan kaldırılır.

DLL bir TLP ilettiğiinde, TLP'nin bir kopyasını saklar. Bu sıra numarasına sahip TLP için bir NAK alırsa, TLP'yi yeniden iletir. Bir ACK aldığımda, arabelleğe alınmış TLP'yi atar.

## 3.7 ÖNEMLİ TEORİLER, SORULAR ve ÖNERİLER

### Anahtar Terimler

adres veri yolu adres hatları tahkim dengeli iletim veriyolu kontrol hatları veri yolu veri hatları diferansiyel sinyalleşme devre dişi kesme dağıtılmış tahkim hata kontrol fonksiyonu	yürütme döngüsü getirme döngüsü flit akış kontrol fonksiyonu talimat döngüsü kesmesi kesme işleyicisi kesme hızmeti rutini (ISR) şeridi bellek adres kaydı (MAR) bellek tampon kaydı (MBR)	çok şartlı dağıtım paketleri PCI Express (PCIe) çevresel bileşeni ara bağlantı (PCI) phit QuickPath Interconnect (QPI) kök kompleks sistem veri yolu
--	--	--

### İnceleme Soruları

- 3.1 Bilgisayar talimatları hangi genel işlev kategorilerini belirtir?
- 3.2 Bir komutun yürütülmesini tanımlayan olası durumları listeleyiniz ve kısaca tanımlayınız.
- 3.3 Çoklu kesintilerle başa çıkmak için iki yaklaşımlı listeleyin ve kısaca tanımlayın.
- 3.4 Bir bilgisayarın ara bağlantı yapısı (örn. veri yolu) ne tür transferleri desteklemelidir?
- 3.5 QPI protokol katmanlarını listeleyin ve kısaca tanımlayın.
- 3.6 PCIe protokol katmanlarını listeleyin ve kısaca tanımlayın.

### Problemler

- 3.1 Şekil 3.4'teki varsayımsal makinede ayrıca iki G/C talimiği vardır:

0011= G/C'den AC yükleyin 0111=  
AC'yi G/C'ye depolayın

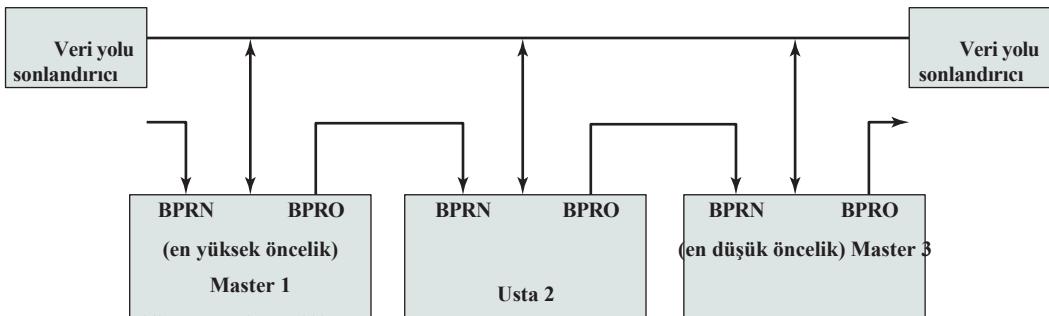
Bu durumlarda, 12 bitlik adres belirli bir G/C cihazını tanımlar. Aşağıdaki program için program yürütmesini (Şekil 3.5 formatını kullanarak) gösterin:

1. Cihazdan AC yükleyin 5.
2. Bellek konumu 940'in içeriğini ekleyin.
3. AC'yi cihaz 6'ya depolayın.

Cihaz 5'ten alınan bir sonraki değerini 3 olduğunu ve 940 konumunun 2 değerini içerdığını varsayıalım.

- 3.2 Şekil 3.5'teki programın yürütülmesi metinde altı adım kullanılarak açıklanmıştır. MAR ve MBR kullanımını göstermek için bu açıklamayı genişletin.

- 3.3** İki alandan oluşan 32 bitlik talimatlara sahip 32 bitlik varsayımsal bir mikroişlemci düşünün: ilk bayt işlem kodunu, geri kalani ise anlık veya işlenen adresini içerir.
- Doğrudan adreslenebilir maksimum bellek kapasitesi (bayt cinsinden) nedir?
  - Mikroişlemci veriyolunun sistem hızı üzerindeki etkisini :
    - 32 bit yerel adres veri yolu ve 16 bit yerel veri yolu veya
    - 16 bit yerel adres veri yolu ve 16 bit yerel veri yolu.
  - Program sayacı ve komut kaydı için kaç bit gereklidir?
- 3.4** Varsayımsal bir mikroişlemcinin 16 bitlik bir adres ürettiğini (örneğin, program sayacının ve adres kayıtlarının 16 bit genişliğinde olduğunu varsayıyalım) ve 16 bitlik bir veri yolu sahip olduğunu düşünün.
- İşlemcinin "16 bitlik bir belleğe" bağlı olması durumunda doğrudan erişebilecegi maksimum bellek adres alanı nedir?
  - İşlemcinin "8 bitlik bir belleğe" bağlı olması durumunda doğrudan erişebilecegi maksimum bellek adres alanı nedir?
  - Hangi mimari özellikler bu mikroişlemcinin ayrı bir "G/C alanına" erişmesini sağlayacaktır?
  - Eğer bir giriş ve çıkış komutu 8 bitlik bir I/O port numarası belirtebiliyorsa, mikroişlemci kaç tane 8 bitlik I/O portunu destekleyebilir? Kaç tane 16-bit G/C portu? Açıklayınız.
- 3.5** 8 MHz giriş saatı tarafından sürülen 16 bit harici veri yolu sahip 32 bit bir mikroişlemci düşünün. Bu mikroişlemcinin minimum süresi dört giriş saatı döngüsüne eşit olan bir veri yolu sahip olduğunu varsayıyın. Bu mikroişlemcinin veri yolu boyunca sürtürebileceği maksimum veri aktarım hızı bayt/sn cinsinden nedir? Performansını artırmak için harici veri yoluunu 32 bit yapmak mı yoksa mikroişlemciye sağlanan harici saat frekansını iki katına çıkarmak mı daha iyi olur? Yaptığınız diğer varsayımları belirtin ve açıklayın. *İpuç: Veri yolu döngüsü başına aktarılabilenek bayt sayısını belirleyin.*
- 3.6** Basit bir klavye/yazıcı teletipini kontrol eden bir I/O modülü içeren bir bilgisayar sistemi düşünün. Aşağıdaki kayıtlar işlemcide bulunur ve doğrudan sistemi veri yoluna bağlanır:
- INPR: GIRIŞ KAYDI: Giriş Kaydı, 8 bit  
 OUTR: Çıkış , 8 bit FGI: Giriş Bayrağı, 1 bit  
 FGO: Çıkış Bayrağı, 1 bit IEN:  
 Kesme Etkinleştirme, 1 bit
- Teletipte tuş girişi ve teletipe yazıcı çıkışı I/O modülü tarafından kontrol edilir. Teletype, alfantorümerik bir sembolü 8 bitlik bir kelimeye kodlayabilir ve 8 bitlik bir kelimeyi alfantorümerik bir sembole çözebilir.
- İşlemcinin bu probleme listelenen ilk dört kaydı kullanarak teletype ile nasıl G/C yapabileceğini açıklayın.
  - IEN'i de kullanarak işlevin nasıl daha verimli bir şekilde getirilebileceğini açıklayınız.
- 3.7** Sırasıyla 8 ve 16 bit genişliğinde harici veri yollarına sahip iki mikroişlemci düşünün. Bu iki işlemci birbirinin aynısıdır ve veri yolu döngülerini aynı uzunluktadır.
- Tüm komutların ve işlenenlerin iki bayt uzunlığında olduğunu varsayıyalım. Maksimum veri aktarım hızları hangi faktörle farklılık gösterir?
  - İşlenenlerin ve talimatların yarısının bir bayt uzunlığında olduğunu varsayıarak tekrarlayın.
- 3.8** Şekil 3.26, Multibus I olarak bilinen bir veri yolu şeması ile kullanılabilenek dağıtılmış bir tahkim şemasını göstermektedir. *Diyagramdaki en soldaki temsilci, daha yüksek öncelikli bir temsilcinin veri yolu istemediğini gösteren sabit bir veri yolu önceliği (BPRN) sinyali alır. Temsilci veri yoluna ihtiyaç duymuyorsa, veri yolu önceliği çıkış (BPRO) hattını ileri sürer. Bir saatin başlangıcında*



Şekil 3.26 Multibus I Dağıtılmış Tahkim

döngüsünde, herhangi bir temsilci BPRO hattını düşürerek veri yolunun kontrolünü talep edebilir. Bu, zincirdeki bir sonraki ajanın BPRN hattını düşürür ve ajanın da BPRO hattını düşürmesi gereklidir. Böylece sinyal zincir boyunca yayılır. Bu zincirleme reaksiyonun sonunda, BPRN'si onaylanmış ve BPRO'su onaylanmamış tek bir temsilci olmalıdır. Bu temsilcinin önceliği vardır. Eğer bir veri yolu döngüsünün başında veri yolu mesgul değilse (BUSY inaktif), önceliği olan aracı BUSY hattını ileri sürererek veri yolunun kontrolünü ele geçirilebilir.

BPR sinyalinin en yüksek öncelikli temsilciden en düşük öncelikli temsilciye yayılması belirli bir zaman alır. Bu süre saat döngüsünden daha mı az olmalıdır? Açıklayın.

- 3.9** VAX SBI veri yolu dağıtılmış, senkronize bir tahkim şeması kullanır. Her SBI aygıtı (yani, işlemci, bellek, G/C modülü) benzersiz bir önceliğe sahiptir ve benzersiz bir aktarım isteği (TR) hattı atar. SBI'da TR0 en yüksek önceliğe sahip olmak üzere bu tür 16 hat (TR0, TR1, ..., TR15) bulunur. Bir cihaz veriyolunu kullanmak istediğiinde, mevcut zaman dilimi sırasında TR hattını ileri sürererek gelecekteki bir zaman dilimi için rezervasyon yapar. Geçerli zaman aralığının sonunda, bekleyen bir rezervasyonu olan her cihaz TR hatlarını inceler; rezervasyonu olan en yüksek öncelikli cihaz bir sonraki zaman aralığını kullanır. Veriyoluna en fazla 17 cihaz bağlanabilir. Önceliği 16 olan cihazın TR hattı yoktur. Neden yok?
- 3.10** VAX SBI üzerinde, en düşük öncelikli aygit genellikle en düşük ortalama bekleme süresine sahiptir. Bu nedenle, işlemciye genellikle SBI üzerinde en düşük öncelik verilir. Öncelik 16 aygitı neden genellikle en düşük ortalama bekleme süresine sahiptir? Hangi koşullar altında bu doğru olmaz?
- 3.11** Senkronize bir okuma işlemi (Şekil 3.18), bellek modülü, sinyal yerleşmesine izin vermek için Okuma sinyalinin düşen kenarından yeterince önce veriyi veri yoluna yerleştirmelidir. Bir mikroişlemci veriyolunun 10 MHz'de saatlendirdiğini ve Okuma sinyalinin  $T_{(3)}$ 'ün ikinci yarısının ortasında düşmeye başladığını varsayıyın.
- Bellek okuma komut döngüsünün uzunluğunu belirleyin.
  - Bellek verileri en geç ne zaman veri yoluına yerleştirilmelidir? Veri hatlarının yerleşmesi için 20 ns bekleyin.
- 3.12** Şekil 3.18'de gösterildiği gibi bir bellek okuma zamanlamasına sahip bir mikroişlemci düşünün. Bazı analizlerden sonra tasarımcı, belleğin okuma verisini zamanında sağlamak yaklaşık 180 ns kadar yetersiz kalındığını bilirler.
- Veri yolu saat hızı 8 MHz ise, sistemin düzgün çalışması için kaç bekleme durumu (saat döngüsü) eklenmesi gereklidir?
  - Bekleme durumlarını uygulamak için bir Hazır durum hattı kullanılır. İşlemci bir Okuma komutu verdikten sonra, verileri okumaya başlamadan önce Hazır hattı çekilene kadar beklemelidir. İşlemciyi gerekli sayıda bekleme durumu eklemeye zorlamak için Hazır hattını hangi zaman aralığında düşük tutmaliyiz?

- 3.13** Bir mikroişlemci Şekil 3.18'de gösterildiği gibi bir bellek yazma zamanlamasına sahiptir. Üreticisi, Yazma sinyalinin genişliğinin  $T-50$  ile belirlenebileceğini belirtir; burada  $T$ , ns cinsinden saat periyodusudur.
- Veri yolu saat hızı 5 MHz ise Yazma sinyali için hangi genişliği beklemeliyiz?
  - Mikroişlemcinin veri sayfasında, Yazma sinyalinin düşen kenarından sonra verilerin 20 ns boyunca geçerli kalacağı belirtilmektedir. Belleğe geçerli veri sunumunun toplam süresi nedir?
  - Bellek en az 190 ns boyunca geçerli veri sunumu gerektiriyorsa kaçbekleme durumu eklemeliyiz?
- 3.14** Bir mikroişlemci, bir bellek konumundaki değere 1 ekleyen doğrudan bellek artırma komutuna sahiptir. Komutun beş aşaması vardır: işlem kodunu getir (dört veri yolu saat çevrimi), işlenen adresini getir (üç), işleneni getir (üç çevrim), işlenenе 1 ekle (üç çevrim) ve işlenenek sakla (üç çevrim).
- Her bellek okuma ve bellek yazma işlemine iki veri yolu bekleme durumu eklemek zorunda kalırsak komutun süresi ne kadar (yüzde olarak) artacaktır?
  - Arttırma işleminin 3 döngü yerine 13 döngü sürdüğünü varsayıarak tekrarlayın.
- 3.15** Intel 8088 mikroişlemcisi Şekil 3.18'dekine benzer bir okuma veriyolu zamanlamasına sahiptir, ancak dört işlemci saat döngüsü gerektirir. Geçerli veri, dördüncü işlemci saat çevrimine kadar uzanan bir süre boyunca veri yolu üzerindedir. İşlemci saat hızının 8 MHz olduğunu varsayıyın.
- Maksimum veri aktarımı hızı nedir?
  - Tekrarlayın, ancak aktarılan her bayt için bir bekleme durumu eklemesi gerektiğini varsayıyın.
- 3.16** Intel 8086, birçok yönden 8-bit 8088'e benzeyen 16-bit bir işlemcidir. 8086, alt sıradaki baytin çift adresе sahip olması koşuluyla, bir seferde 2 bayt aktarabilen 16 bitlik bir veri yolu kullanır. Bununla birlikte, 8086 hem çift hem de tek hizalı kelime operandlarına izin verir. Tek hizalı bir sözcüğe başvurulursa, sözcüğü aktarmak için her biri dört veri yolu döngüsünden oluşan iki bellek döngüsü gereklidir. 8086'da 16 bitlik iki içeren bir komut düşünün. İşlenenleri getirmek ne kadar zaman alır? Olası cevapların aralığını verin. Saat hızının 4 MHz olduğunu ve bekleme durumlarının olmadığını varsayıyın.
- 3.17** Veri yolu döngüsü 16 bitlik bir mikroişlemcininki ile aynı olan 32 bitlik bir mikroişlemci düşünün. Ortalama olarak, işlenenlerin ve %20'sinin 32 bit uzunluğunda, %40'ının 16 bit uzunluğunda ve %40'ının sadece 8 bit uzunluğunda olduğunu varsayıyın. 32-bit mikroişlemci ile komutları ve işlenenleri getirirken elde edilen iyileştirmeyi hesaplayın.
- 3.18** Problem 3.14'ün mikroişlemcisi, bir klavyenin bir kesme istek hattını harekete geçirmesiyle aynı anda, doğrudan bellek artırma komutunun işlenen getirme aşamasını başlatır. İşlemci ne kadar süre sonra kesme işleme döngüsüne girer? Veri yolu saat hızının 10 MHz olduğunu varsayıyın.

# 4

## BÖLÜM

### ÖN BELLEK

**4.1 Bilgisayar Bellek Sistemine Genel Bakış Bellek**  
Sistemlerinin Özellikleri Bellek Hiyerarşisi

**4.2 Önbellek Bellek İlkeleri**

**4.3 Önbellek Tasarımının Unsurları**

Önbellek Adresleri  
Önbellek Boyutu  
Eşleme İşlevi  
Değiştirme Algoritmaları  
Yazma Politikası  
Hat Boyutu  
Önbellek Sayısı

**4.4 Pentium 4 Önbellek Organizasyonu**

**4.5 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler**

**Ek 4A İki Seviyeli Belleklerin Performans Özellikleri**

Yerellik  
İki Seviyeli Bellek Performansının Çalışması

### ÖĞRENİM HEDEFLERİ

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- r Bilgisayar bellek sistemlerinin temel özelliklerine ve bellek hiyerarşisinin kullanımına genel bir bakış sunmak.
- r Ön belleğin temel kavramlarını ve amacını tanımlamak.
- r Önbellek tasarıminın temel unsurlarını tartışın.
- r Doğrudan eşleme, ilişkisel eşleme ve küme-ilişkisel eşleme arasında ayrıml yapabilecektir.
- r Birden fazla önbellek seviyesi kullanmanın nedenlerini açıklayın.
- r Çoklu bellek seviyelerinin performans üzerindeki etkilerini anlamak.

Kavram olarak basit görünmesine rağmen, bilgisayar belleği belki de bir bilgisayar sisteminin herhangi bir özelliği içinde en geniş tür, teknoloji, organizasyon, performans ve maliyet çeşitliliğini sergiler. Hiçbir teknoloji bir bilgisayar sisteminin bellek gereksinimlerini karşılamada optimal değildir. Sonuç olarak, tipik bir bilgisayar sistemi, bazıları sisteme dahili (işlemci tarafından doğrudan erişilebilir) ve bazıları harici (işlemci tarafından bir I/O modülü aracılığıyla erişilebilir) olmak üzere bir bellek alt sistemleri hiyerarşisi ile donatılmıştır.

Bu bölüm ve bir sonraki bölüm dahili bellek unsurlarına odaklanırken, 6. Bölüm harici belleğe ayrılmıştır. Başlangıç olarak, ilk bölüm bilgisayar belleklerinin temel karakteristiklerini incelemektedir. Bölümün geri kalanında tüm modern bilgisayar sistemlerinin önemli bir unsuru olan ön bellek incelenmektedir.

## 4.1 BİLGİSAYAR BELLEK SİSTEMİNE GENEL BAKIŞ

### Bellek Sistemlerinin Özellikleri

Bellek sistemlerini temel özelliklerine göre sınıflandırırsak, bilgisayar belleğinin karmaşık konusu daha yönetilebilir hale gelir. Bunların en önemlileri Tablo 4.1'de listelenmiştir.

Tablo 4.1'deki **konum** terimi belleğin bilgisayardan dahili ya da harici belleği olduğunu ifade eder. Dahili bellek genellikle ana bellek ile eş tutulur, ancak dahili belleğin başka biçimleri de vardır. İşlemci, kayıtlar şeklinde kendi yerel belleğine ihtiyaç duyar (örneğin, bkz. Şekil 2.3). Ayrıca, göreceğimiz gibi, işlemcinin kontrol birimi kısmı da kendi dahili belleğine ihtiyaç duyabilir. Bu son iki tür dahili belleğin tartışmasını daha sonraki bölümlere erteleyeceğiz. Önbellek dahili belleğin bir başka biçimidir. Harici bellek, I/O denetleyicileri aracılığıyla işlemcinin erişebildiği disk ve teyp gibi çevresel depolama aygıtlarından oluşur.

Belleğin belirgin bir özelliği **kapasitesidir**. Dahili bellek için bu tipik olarak bayt ( $1 = 8$  bit) veya kelime cinsinden ifade edilir. Yaygın kelime uzunlukları 8, 16 ve 32 bittir. Harici bellek kapasitesi tipik olarak bayt cinsinden ifade edilir.

**Tablo 4.1** Bilgisayar Bellek Sistemlerinin Temel Özellikleri

Konum	Performans
Dahili (örneğin, işlemci kayıtları, önbellek, ana belleği)	Erişim süresi Çevrim süresi Aktarım hızı
Harici (örn. optik diskler, manyetik diskler diskleri, bantlar)	
Kapasite	Fiziksel Tip
Kelime sayısı	Yarı İletken
Bayt sayısı	Manyetik
Transfer Birimi	Optik
Kelime	Manyeto-optik
Blok	
Erişim Yöntemi	Fiziksel Özellikler
Sıralı	Uçuçu/uçuçu olmayan Silinebilir/silinemeyez
Doğrudan	
Rastgele İlişkisel	
Organizasyon	Bellek modülleri

**Ilgili bir kavram da aktarım birimidir.** Dahili bellek için aktarım birimi, bellek modülüne giren ve çıkan elektrik hatlarının sayısına eşittir. Bu, kelime uzunluğuna eşit olabilir, ancak genellikle 64, 128 veya 256 bit gibi daha büyütür. Bu noktayı açılığa kavuşturmak için dahili bellekle ilgili üç kavramı ele alalım:

- **Kelime:** Bellek organizasyonunun "doğal" birimi. Bir kelimenin boyutu tipik olarak bir tamsayıyı temsil etmek için kullanılan bit sayısına ve komut uzunluğuna eşittir. Ne yazık ki, birçok istisna vardır. Örneğin CRAY C90 (eski model bir CRAY süper bilgisayarı) 64 bit kelime uzunluğuna sahiptir ancak 46 bitlik bir tamsayı gösterimi kullanır. Intel x86 mimarisi, baytların katları olarak ifade edilen çok çeşitli komut uzunluklarına ve 32 bitlik bir kelime boyutuna sahiptir.
  - **Adreslenebilir birimler:** Bazı sistemlerde adreslenebilir birim kelimedir. Bununla birlikte, birçok sistem bayt düzeyinde adreslemeye izin verir. Her durumda, bir adresin  $A$  cinsinden uzunluğu ile adreslenebilir birimlerin  $N$  sayısı arasındaki ilişki  $2^A = N$  şeklindedir.
  - **Aktarım birimi:** Ana bellek için bu, bir seferde bellekten okunan veya belleğe yazılan bit sayısıdır. Aktarım biriminin bir kelimeye ya da adreslenebilir bir eşit olması gerekmekz. Harici bellek için, veriler genellikle bir kelimededen çok daha büyük birimler halinde aktarılır ve bunlar blok olarak adlandırılır.
- Bellek türleri arasındaki bir diğer ayrımda veri birimlerine **erişim yöntemidir**. Bunlar aşağıdakileri içerir:
- **Sıralı erişim:** Bellek, kayıt adı verilen veri birimleri halinde düzenlenmiştir. Erişim belirli bir doğrusal sırayla yapılmalıdır. Kayıtları ayırmak ve geri alma işlemine yardımcı olmak için saklanan adresleme bilgileri kullanılır. Paylaşılan bir okuma-yazma mekanizması kullanılır ve bu, her ara kaydı geçerek ve reddederek mevcut konumundan istenen konuma taşınmalıdır. Bu nedenle, rastgele bir kayda erişim süresi oldukça değişkendir. Bölüm 6'da tartışılan teyp birimleri sıralı erişimdir.
  - **Doğrudan erişim:** Sıralı erişimde olduğu gibi, doğrudan erişim de paylaşılan bir okuma-yazma mekanizması içerir. Ancak, bireysel bloklar veya kayıtlar benzersiz bir

fiziksel konuma dayalı adres. Erişim, genel bir çevreye ulaşmak için doğrudan erişim ve nihai konuma ulaşmak için sıralı arama, sayma veya bekleme yoluyla gerçekleştirilir. Yine, erişim süresi değişkendir. Bölüm 6'da ele alınan disk birimleri doğrudan erişimlidir.

- **Rastgele erişim:** Bellekteki her adreslenebilir konumun benzersiz, fiziksel olarak kablolanmış bir adresleme mekanızması vardır. Belirli bir konuma erişme süresi önceki erişimlerin sırasından bağımsızdır ve sabittir. Böylece, herhangi bir konum rastgele seçilebilir ve doğrudan adreslenip erişilebilir. Ana bellek ve bazı önbellek sistemleri rastgele erişimlidir.
- **İlişkisel:** Bu, belirli bir eşleşme için bir kelime içindeki istenen bit konumlarının karşılaştırılmasını ve bunu tüm kelimeler için aynı anda yapmayı sağlayan rastgele erişimli bir bellek türüdür. Böylece, bir kelime adresinden ziyade içeriğinin bir kısmına dayalı olarak alınır. Sıradan rastgele erişimli belleklerde olduğu gibi, her konumun kendi adresleme mekanızması vardır ve geri getirme süresi konumdan veya önceki erişim modellerinden bağımsız olarak sabittir. Ön bellekler ilişkisel erişim kullanabilir.

Bir kullanıcının bakiş açısından, belleğin en önemli iki özelliği kapasite ve **performansı**. Üç performans parametresi kullanılır:

- **Erişim süresi (geçikme):** Rastgele erişimli bellek için bu, bir okuma veya yazma işlemini gerçekleştirmek için geçen süredir, yani bir adresin belleğe sunulduğu andan verilerin depolandığı veya kullanıma hazır hale getirildiği ana kadar geçen süredir. Rastgele erişimli olmayan bellekler için erişim süresi, okuma-yazma mekanızmasının istenen konuma yerleştirilmesi için geçen süredir.
- **Bellek döngü süresi:** Bu kavram öncelikle rastgele erişimli belleklere uygulanır ve erişim süresi artı ikinci bir erişimin başlayabilmesi için gereken ek süreden oluşur. Bu ek süre, sinyal hatlarındaki geçişlerin sonması veya tahrip edici bir şekilde okunmaları halinde verilerin yeniden oluşturulması için gerekli olabilir. Bellek döngü süresinin işlemciyle değil sistem veriyoluyla ilgili olduğunu unutmayın.
- **Aktarım hızı:** Bu, verilerin bir bellek biriminin içine veya aktarılabilme hızıdır. Rastgele erişimli bellek için  $1/(döngü süresi)$ 'ne eşittir. Rastgele erişimli olmayan bellek için aşağıdaki ilişki geçerlidir:

$$T_n = T_A + \frac{n}{R} \quad (4.1)$$

nerede

$T_n$ = n biti okumak veya yazmak için ortalama süre

$T_A$ = Ortalama erişim süresi

$n$ = Bit sayısı

$R$ = Aktarım hızı, saniye başına bit (bps) cinsinden

Çeşitli **fiziksel** bellek **türleri** kullanılmıştır. Günümüzde en yaygın olanları yarı iletken bellek, disk ve bant için kullanılan manyetik yüzey belleği ve optik ve manyeto-optiktir.

Veri depolamanın bazı **fiziksel özellikleri** önemlidir. Ucucu bir bellekte bilgi doğal olarak bozulur ya da elektrik gücü kapatıldığında kaybolur. Ucucu olmayan bir bellekte, bir kez kaydedilen bilgi kasıtlı olarak değiştirilene kadar bozulmadan kalır; bilgiyi korumak için elektrik gücüne gerek yoktur. Manyetik yüzeyli bellekler ucucu değildir. Yarı iletken bellekler (enteğre devreler üzerindeki bellekler) ucucu ya da ucucu olmayan bellekler olabilir. Ucucu olmayan bellek, depolama biriminin yok edilmesi dışında değiştirilemez. Bu tip yarı iletken bellekler *salt okunur bellek* (ROM) olarak bilinir. Zorunlu olarak, pratik bir ucucu olmayan bellek aynı zamanda ucucu olmamalıdır.

Rastgele erişimli bellekler için **organizasyon** önemli bir tasarım konusudur. Bu metinde *organizasyon*, kelimeleri oluşturmak için bitlerin fiziksel düzenini ifade eder. Bölüm 5'te açıkladığı gibi, bariz düzenleme her zaman kullanılmaz.

### Bellek Hiyerarşisi

Bir bilgisayarın belleği üzerindeki tasarım kısıtlamaları üç soruya özetlenebilir: Ne kadar? Ne kadar hızlı? Ne kadar pahalı?

Ne kadar sorusu biraz açık ucludur. Kapasite varsa, muhtemelen bunu kullanmak için uygulamalar geliştirilecektir. Ne kadar hızlı sorusuna cevap vermek bir anlamda daha kolaydır. En yüksek performansı elde etmek için belleğin işlemciye ayak uydurabilmesi gereklidir. Yani, işlemci talimatları yerine getirirken, talimatları ya da operandları beklerken duraklamasını istemeyiz. Son soru da dikkate alınmalıdır. Pratik bir sistem için, belleğin maliyeti diğer bileşenlere göre makul olmalıdır.

Bekleneceği gibi, belleğin üç temel özelliği arasında bir değişim tokuş söz konusudur: kapasite, erişim süresi ve maliyet. Bellek sistemlerini uygulamak için çeşitli teknolojiler kullanılır ve bu teknoloji yelpazesinde aşağıdaki ilişkiler geçerlidir:

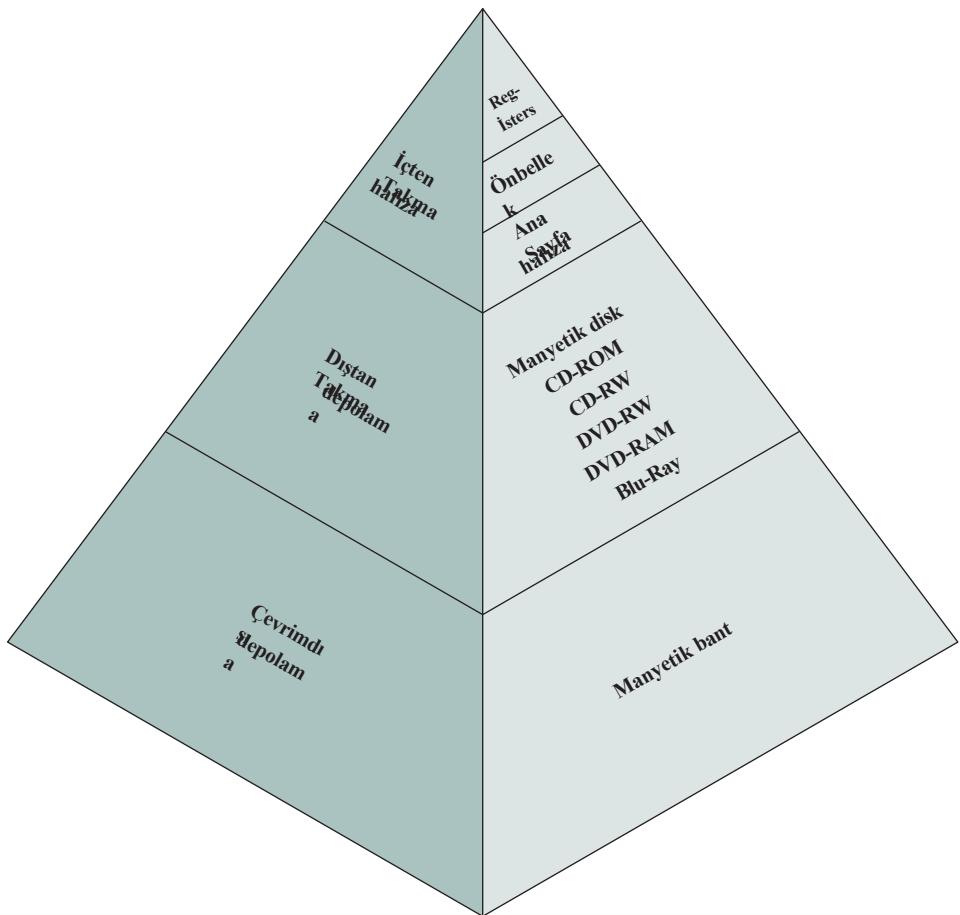
- Daha hızlı erişim süresi, bit başına daha yüksek maliyet;
- Daha yüksek kapasite, uç başına daha düşük maliyet;
- Daha yüksek kapasite, daha yavaş erişim süresi.

Tasarımının karşı karşıya olduğu ikilem açıkta. Tasarımcı, hem kapasite gerekliliği için hem de bit başına maliyet düşüğü için büyük kapasiteli bellek sağlayan bellek teknolojilerini kullanmak ister. Ancak, performans gereksinimlerini karşılamak için tasarımcının kısa erişim sürelerine sahip pahalı, nispeten daha düşük kapasiteli bellekler kullanması gereklidir.

Bu ikilemden kurtulmanın yolu tek bir bellek bileşenine veya teknolojisine güvenmek değil, bir bellek **hiyerarşisi** kullanmaktadır. Tipik bir hiyerarşi Şekil 4.1'de gösterilmiştir. Hiyerarşide aşağıya doğru inildikçe aşağıdakiler gerçekleşir:

- a. Bit başına azalan maliyet;
- b. Kapasite artırımı;
- c. Erişim süresinin artırılması;
- d. İşlemci tarafından belleğe erişim sıklığının azaltılması.

Böylece daha küçük, daha pahalı, daha hızlı bellekler daha büyük, daha ucuz, daha yavaş belleklerle desteklenir. Bu organizasyonun başarısının anahtarı



**Şekil 4.1** Bellek Hiyerarşisi

(d) maddesidir: azalan erişim sıklığı. Bu kavramı bu bölümün ilerleyen kısımlarında önbellek ve Bölüm 8'de sanal bellekten bahsederken daha ayrıntılı olarak inceleyeceğiz. Bu noktada kısa bir açıklama yapılacaktır.

Ortalama erişim süresini azaltmak için iki bellek seviyesinin kullanılması prensipte işe yarar, ancak yalnızca (a) ile (d) arasındaki koşullar . Çeşitli teknolojiler kullanılarak, (a)'dan (d)'ye kadar olan koşulları karşılayan bir bellek sistemleri yelpazesi mevcuttur.

(c). Neyse ki, (d) koşulu da genel olarak geçerlidir.

(d) koşulunun temeli **referansın yerelliği** olarak bilinen bir ilkedir [DENN68]. Bir programın yürütülmesi sırasında, hem talimatlar hem de veriler için işlemci tarafından yapılan bellek referansları kümelenme eğilimindedir. Programlar tipik olarak bir dizi yinelemeli döngü ve alt rutin içerir. Bir döngüye ya da alt rutine girildiğinde, küçük bir talimat kümelerine tekrar tekrar referanslar yapılır. Benzer şekilde, tablolar ve diziler üzerindeki işlemler de kümelenmiş veri sözcükleri kümelerine erişimi içerir. Uzun bir süre boyunca, kullanılan kümeler değişir, ancak kısa bir süre boyunca, işlemci öncelikle sabit bellek referans kümeleriyle çalışır.

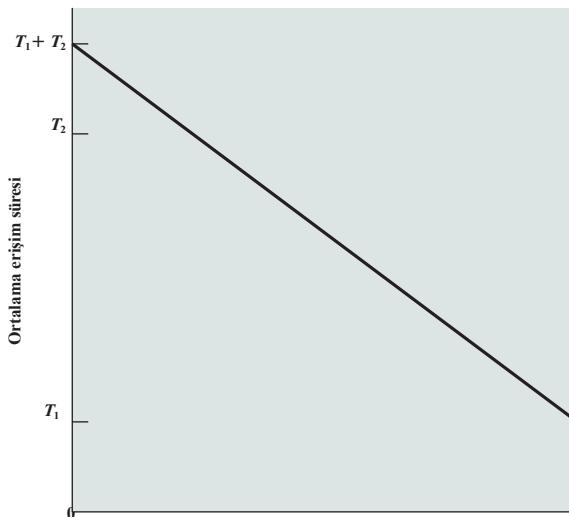
**ÖRNEK 4.1** İşlemcinin iki bellek seviyesine erişimi olduğunu varsayalım. Seviye 1 1000 kelime içerir ve erişim süresi  $0,01\text{ ms}'dır; seviye 2 100.000 kelime içerir ve erişim süresi  $0,1\text{ ms}'dır. Erişilecek bir kelime 1. seviyede ise, işlemcinin buna doğrudan eriştiğini varsayılmış. Eğer kelime 2. seviyede ise, kelime önce 1. seviyeye aktarılır ve daha sonra işlemci tarafından erişilir. Basitlik açısından, işlemcinin kelimenin 1. seviyede mi yoksa 2. seviyede mi olduğunu belirlemesi için gereken süreyi göz ardı ediyoruz. Şekil 4.2 bu durumu kapsayan eğrinin genel şeklini göstermektedir. Şekilde iki seviyeli bir belleğe ortalama erişim süresi isabet oranı  $H$ 'nin bir fonksiyonu olarak gösterilmektedir; burada  $H$  tüm bellek erişimlerinin daha hızlı bellekte (örneğin önbellekte) bulunan kısım,  $T_1$  1 seviyeye erişim süresi ve  $T_2$  2 seviyeye erişim süresidir.<sup>1</sup> Görülebileceği gibi, yüksek 1. seviye erişim için ortalama toplam erişim süresi 1. seviyeye 2. seviyeden çok daha yakındır.$$

Örneğimizde, bellek erişimlerinin  $\%95$ 'inin 1. seviyede bulunduğu varsayılmış. O zaman bir kelimeye erişmek için ortalama süre şu şekilde ifade edilebilir

$$(0.95)(0.01\text{ ms}) + (0.05)(0.01\text{ ms} + 0.1\text{ ms}) = 0.0095 + 0.0055 = 0.015\text{ ms}$$

Ortalama erişim süresi, istenildiği gibi  $0,1\text{ ms}'den çok  $0,01\text{ ms}'ye yakındır.$$

Buna göre, hiyerarşî boyunca verileri, birbirini izleyen her bir alt seviyeye erişim yüzdesi bir üst seviyeye göre önemli ölçüde daha az olacak şekilde düzenlemek mümkündür. Daha önce sunulan iki seviyeli örneği ele alalım. Seviye 2 olsun



Sadece 1. seviyeyi içeren erişimlerin oranı (isabet oranı)

**Şekil 4.2** Sadece 1. Seviyeyi İçeren Erişimlerin Performansı (isabet oranı)

<sup>1</sup> Erişilen sözcük daha hızlı bellekte bulunursa, bu bir **isabet** olarak tanımlanır. Erişilen sözcük daha hızlı bellekte bulunamazsa **iskalama** meydana gelir.

bellek tüm program talimatlarını ve verileri içerir. Mevcut kümeler geçici olarak 1. seviyeye yerleştirilebilir. Zaman, seviye 1'e gelen yeni bir kümeye yer açmak için seviye 1'deki kümelerin birinin seviye 2'ye geri takas edilmesi gerekecektir. Ancak ortalama olarak, referansların çoğu seviye 1'de bulunan talimatlara ve verilere olacaktır.

Bu prensip, Şekil 4.1'de gösterilen hiyerarşinin de gösterdiği gibi, ikiden fazla bellek seviyesine uygulanabilir. En hızlı, en küçük ve en masraflı bellek türü işlemcinin içindeki kayıtlardan oluşur. Tipik olarak, bir işlemci birkaç düzine yazmaç içerir, ancak bazı makineler yüzlerce içerir. Ana bellek bilgisayarın temel dahili bellek sistemidir. Ana bellekteki her konumun benzersiz bir adresi vardır. Ana bellek genellikle daha yüksek hızlı, daha küçük bir önbellek ile genişletilir. Önbellek genellikle programcı ya da işlemci tarafından görülmez. Önbellek, performansı artırmak için verilerin ana bellek ve işlemci kayıtları arasında hareketini düzenleyen bir cihazdır.

Az önce açıklanan üç bellek biçimini tipik olarak uçucudur ve yarı iletken teknolojisini kullanır. Üç seviyenin kullanılması, yarı iletken belleğin hız ve maliyet açısından farklılık gösteren çeşitli türleri olduğu gereğinden yararlanmaktadır. Veriler, en yaygın olanları sabit disk ve çıkarılabilir manyetik disk, bant ve optik depolama gibi çıkarılabilir ortamlar olan harici yoğun depolama cihazlarında daha kalıcı olarak saklanır. Harici, uçucu olmayan bellek, **ikincil** bellek veya **yardımcı bellek** olarak da adlandırılır. Bunlar program ve veri dosyalarını saklamak için kullanılır ve genellikle programcı tarafından tek tek baytar ya da sözcükler yerine yalnızca dosyalar ve kayıtlar olarak görülebilir. Disk ayrıca Bölüm 8'de ele alınan sanal bellek olarak bilinen ana belleğe bir uzanti sağlamak için de kullanılır.

Hiyerarşide başka bellek biçimleri de dahil edilebilir. Örneğin, büyük IBM ana bilgisayarları genişletilmiş depolama olarak bilinen bir tür dahili bellek içerir. Bu, ana bellekten daha yavaş ve daha ucuz olan bir yarı iletken teknolojisi kullanır., bu bellek hiyerarşide uymaz ancak bir yan daldır: Veriler ana bellek ile genişletilmiş depolama arasında taşınabilir ancak genişletilmiş depolama ile harici bellek arasında taşınamaz. Diğer ikincil bellek biçimleri arasında optik ve manyeto-optik diskler yer alır. Son olarak, yazılımda hiyerarşide etkin bir şekilde ek seviyeler eklenebilir. Ana belleğin bir kısmı, diske okunacak verileri geçici olarak tutmak için bir tampon olarak kullanılabilir. Bazen disk önbelleği olarak da adlandırılan böyle bir teknik<sup>2</sup> performansı iki şekilde artırır:

- Disk yazma işlemleri kümelenmiştir. Birçok küçük veri aktarımı yerine, birkaç büyük veri aktarımımız vardır. Bu, disk performansını artırır ve işlemci müdahalesini en aza indirir.
- Yazılması gereken bazı veriler, diske sonraki dökümden önce bir program tarafından referans alınabilir. Bu durumda veriler diskten yavaşça alınmak yerine yazılım önbelleğinden hızla alınır.

Ek 4A, çok seviyeli bellek yapılarının performans etkilerini incelemektedir.

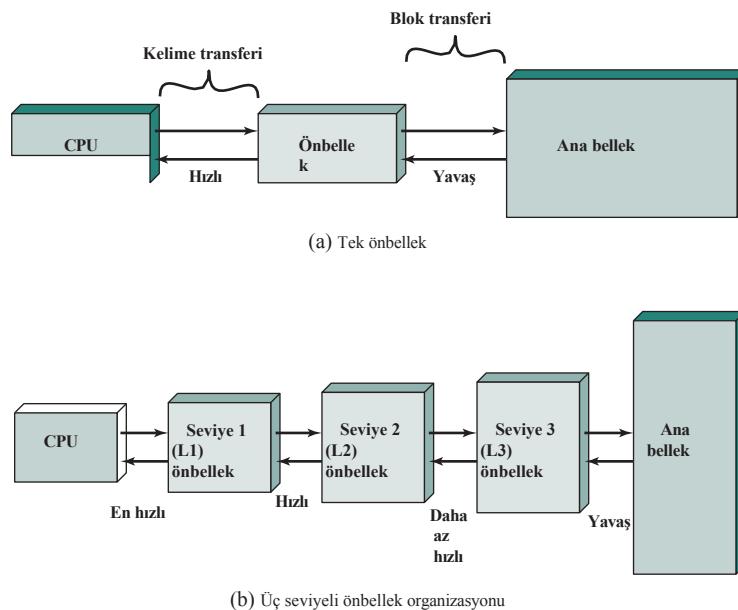
---

<sup>2</sup>Disk önbelleği genellikle tamamen yazılımsal bir tekniktir ve bu kitapta incelenmemiştir. Bir tartışma için [STAL15]'e bakınız.

## 4.2 ÖN BELLEK PRENSİPLERİ

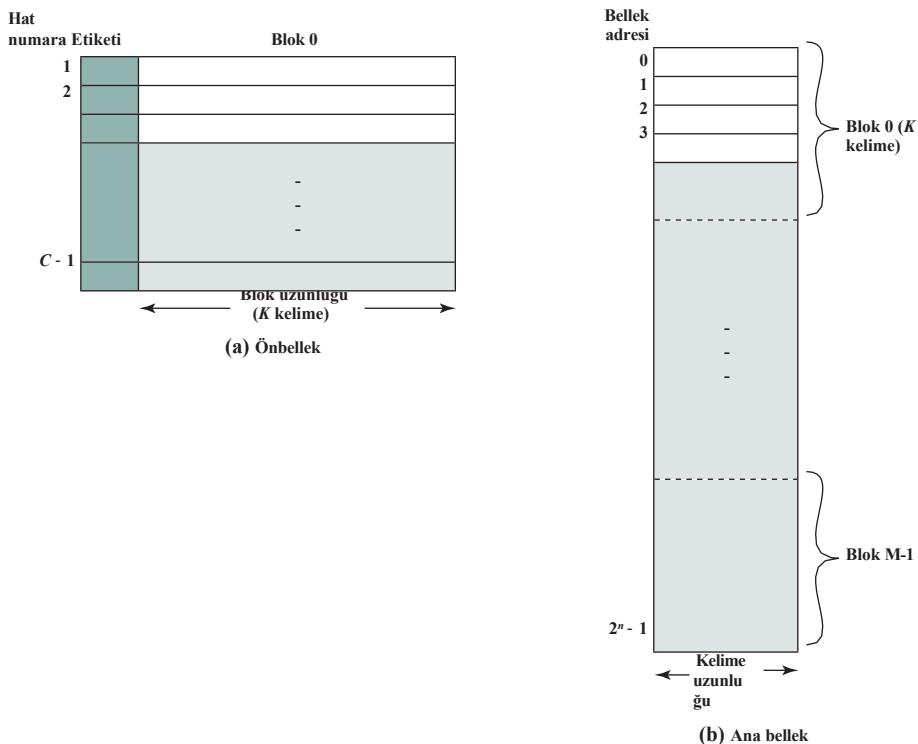
Ön bellek, pahali, yüksek hızlı belleğin bellek erişim süresi ile daha ucuz, düşük hızlı belleğin büyük bellek boyutunu birleştirmek için tasarlanmıştır. Bu kavram Şekil 4.3'a'da gösterilmiştir. Nispeten büyük ve yavaş bir ana bellek ile birlikte daha küçük, daha hızlı bir ön bellek bulunmaktadır. Ön bellek, ana belleğin bazı bölmelerinin bir kopyasını içerir. İşlemci bellekteki bir kelimeyi okumaya çalıştığında, kelimenin önbelleye olup olmadığı kontrol edilir. Eğer öyleyse, sözcük işlemciye ilettilir. Değilse, sabit sayıda sözcükten oluşan bir ana bellek bloğu önbelleye okunur ve ardından sözcük işlemciye teslim edilir. Referansın yerelliği olsusunu nedeniyle, bir veri bloğu tek bir bellek referansını karşılamak için önbelleye getirildiğinde, aynı bellek konumuna veya bloktaki diğer kelimelelere gelecekte referanslar olması muhtemeldir.

Şekil 4.3b birden fazla önbelleye ana bellek sisteminin yapısını göstermektedir. Ana bellek  $2^n$  adede kadar adreslenebilir sözcükten oluşur ve her sözcüğün benzersiz bir  $n$ -bit adresi vardır. Eşleme amacıyla, bu belleğin her biri  $K$  kelimedenden oluşan bir dizi sabit uzunlukta bloktan olduğu kabul edilir. , ana bellekte  $M = 2^n/K$  blok vardır. Önbelleye, **satır** adı verilen  $m$  bloktan oluşur.<sup>3</sup> Her satır  $K$  kelime içerir,



**Şekil 4.3** Önbelleye ve Ana Bellek

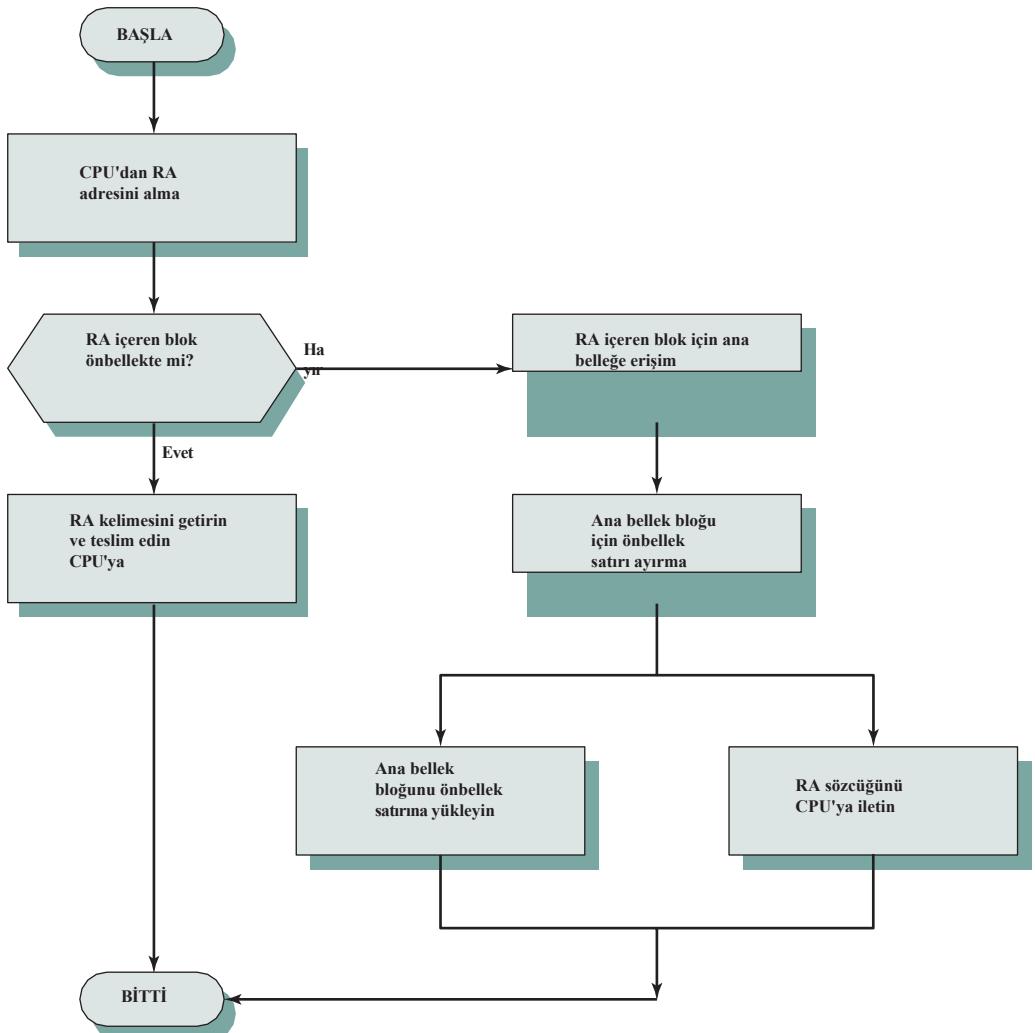
<sup>3</sup> Önbelleye temel birimine atıfta bulunurken, iki nedenden dolayı *blok* terimi yerine *satır* terimi kullanılır: (1) bir önbelleye satırıyla aynı sayıda veri sözcüğü içeren bir ana bellek bloğuya karşılmaması için; ve (2) bir önbelleye satırı, tipki bir ana bellek bloğu gibi yalnızca K veri sözcüğünü değil, aynı zamanda etiket ve kontrol bitlerini de içerdigi için.



**Şekil 4.4** Önbellek/Ana Bellek Yapısı

arti birkaç bitlik bir etiket. Her satır ayrıca, satırın önbelleğe yüklenildiğinden beri değiştirilip değiştirilmediğini gösteren bir bit gibi kontrol bitleri (gösterilmemiştir) içerir. Etiket ve kontrol bitlerini içermeyen bir satırın uzunluğu, **satır boyutudur**. Satır boyutu 32 bit kadar küçük olabilir, her "kelime" tek bir bayttır; bu durumda satır boyutu 4 bayttır. Satır sayısı ana bellek bloklarının sayısından ( $m \vee M$ ) önemli ölçüde daha azdır. Herhangi bir zamanda, bellek bloklarının bazı alt kümeleri önbellekteki satırlarda bulunur. Bir bellek bloğundaki bir kelime okunduğunda, bu önbellekteki satırlardan birine aktarılır. Satırlardan daha fazla blok olduğu için, tek bir satır belirli bir bloğa benzersiz ve kalıcı olarak atanamaz. Bu nedenle, her satır o anda hangi bloğun depolandığını tanımlayan bir **etiket** içerir. Etiket genellikle bu bölümün ilerleyen kısımlarında açıkladığı gibi ana bellek adresinin bir kısmıdır.

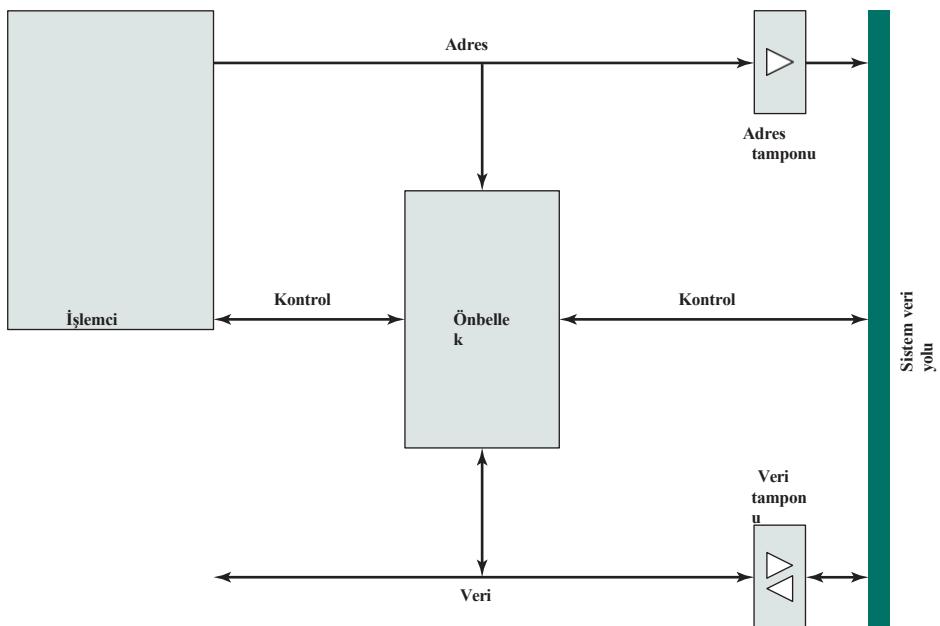
Şekil 4.5 okuma işlemini göstermektedir. İşlemci kelimenin okuma adresini (RA) üretir. Eğer kelime önbellekte bulunuyorsa, işlemciye teslim edilir. Aksi takdirde, bu kelimeyi içeren blok önbelleğe yüklenir ve kelime işlemciye teslim edilir. Şekil 4.5 bu son iki işlemin paralel olarak gerçekleştiğini göstermektedir ve Şekil 4.6'da gösterilen ve çağdaş önbellek organizasyonlarının tipik bir örneği olan organizasyonu yansımaktadır. Bu organizasyonda, önbellek işlemciye veri, kontrol ve adres hatları aracılığıyla bağlanır. Veri ve adres hatları aynı zamanda veri ve adres tamponlarına da bağlanır ve bunlar sistem veri yoluna



Şekil 4.5 Önbellek Okuma İşlemi

hangi ana belleğe ulaşıldığı. Bir önbellek vuruşu gerçekleştiğinde, veri ve adres tamponları devre dışı bırakılır ve iletişim yalnızca işlemci ve önbellek arasında gerçekleşir, sistem veri yolu trafiği olmaz. Bir önbellek meydana geldiğinde, istenen adres sistem veriyoluna yüklenir ve veriler veri tamponu aracılığıyla hem önbelleğe hem de işlemciye geri gönderilir. Diğer kuruluşlarda, önbellek tüm veri, adres ve kontrol hatları için işlemci ile ana bellek arasında fiziksel olarak yerleştirilir. Bu ikinci durumda, bir önbellek özlemi için, istenen kelime önce önbelleğe okunur ve daha sonra önbellekten işlemciye aktarılır.

Önbellek kullanımıyla ilgili performans parametrelerine ilişkin bir tartışma Ek 4A'da yer almaktadır.



Şekil 4.6 Tipik Önbellek Organizasyonu

### 4.3 ÖNBELLEK TASARIMININ UNSURLARI

Bu bölümde önbellek tasarım parametrelerine genel bir bakış sunulmakta ve bazı tipik sonuçlar bildirilmektedir. Zaman zaman yüksek **performanslı bilgi işlemde (HPC)** önbellek kullanımına atıfta bulunuyoruz. HPC, özellikle büyük miktarda veri, vektör ve matris hesaplama ve paralel algoritmaların kullanımını içeren bilimsel uygulamalar için süper bilgisayarlar ve yazılımları ile ilgilenebilir. HPC için önbellek tasarımcıları, diğer donanım platformları ve uygulamalarından oldukça farklıdır. Gerçekten de birçok araştırmacı, HPC uygulamalarının önbellek bilgisayar mimarilerinde düşük performans gösterdiğini tespit etmiştir [BAIL93]. Diğer araştırmacılar, uygulama yazılımının önbellekten yararlanacak şekilde ayarlanması halinde önbellek hiyerarşisinin performansı artırmada yararlı olabileceğini göstermiştir [WANG99, PRES01].<sup>4</sup>

Çok sayıda önbellek uygulaması olmasına rağmen, önbellek mimarilerini sınıflandırmaya ve farklılaştırılmaya hizmet eden birkaç temel tasarım ögesi vardır. Tablo 4.2 temel unsurları listelemektedir.

#### Önbellek Adresleri

Neredeyse tüm gömülü olmayan işlemciler ve birçok gömülü, Bölüm 8'de tartışılan bir kavram olan sanal belleği destekler. Özünde sanal bellek, programların fiziksel olarak mevcut ana bellek miktarına bakılmaksızın belleği mantıksal bir bakış açısıyla adreslemesine olanak tanıyan bir kolaylığıktır. Sanal bellek kullanıldığında, makine komutlarının adres alanları sanal adresler içerir. Okumalar için

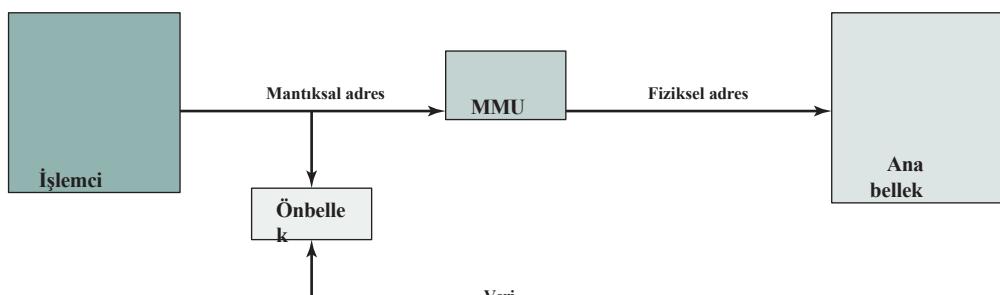
<sup>4</sup>HPC ile ilgili genel bir tartışma için bakınız [DOWD98].

Tablo 4.2 Önbellek Tasarımının Unsurları

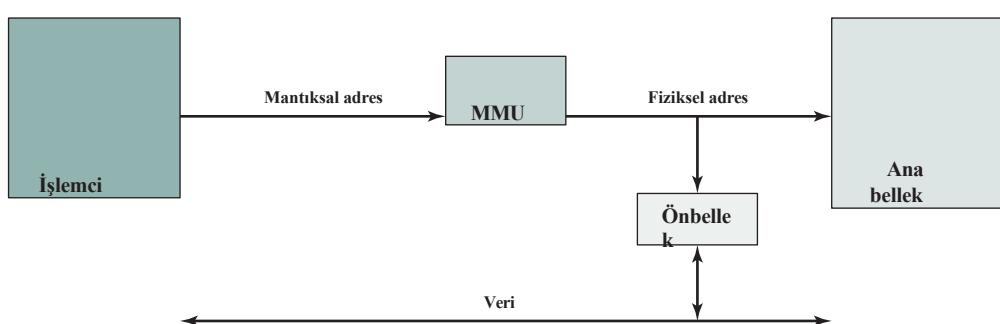
<b>Önbellek Adresleri</b>	<b>Politika Yazın</b>
Mantıksal	Aracılığıyla yaz Geri yaz
Fiziksel	
<b>Önbellek Boyutu</b>	<b>Hat Boyutu</b>
<b>Eşleme İşlevi</b>	<b>Önbellek Sayısı</b>
Doğrudan	Tek veya iki seviyeli Birleşik veya bölünmüş
İlişkisel Küme	
ilişkisel	
<b>Değiştirme Algoritması</b>	
En son kullanılan (LRU) İlk giren ilk çıkar (FIFO) En az sıkılıkla kullanılan (LFU) Rastgele	

donanım bellek yönetim birimi (MMU) her bir sanal adresi ana bellekteki fizikselle bir adrese çevirir.

Sanal adresler kullanıldığında, sistem tasarımcısı önbellegi işlemci ile MMU arasına veya MMU ile ana bellek arasında yerleştirmeyi seçebilir (Şekil 4.7). **Sanal önbellek** olarak da bilinen **mantıksal** önbellek, verileri



(a) Mantıksal önbellek



(b) Fizikselle önbellek

Şekil 4.7 Mantıksal ve Fizikselle Önbellekler

**sanal adresler.** İşlemci önbelleğe MMU'dan geçmeden doğrudan erişir. **Fiziksel önbellek**, ana bellek **fiziksel adreslerini** kullanarak verileri depolar.

Mantıksal önbelleğin belirgin bir avantajı, önbellek erişim hızının fiziksel daha hızlı olmasıdır, çünkü önbellek MMU bir adres çevirisi gerçekleştirmeden önce yanıt verebilir. Dezavantajı ise çoğu sanal bellek sisteminin her uygulamaya aynı sanal bellek adres alanını sağlamasıdır., her uygulama 0 adresinden başlayan bir sanal bellek görür. Bu nedenle, iki farklı uygulamada aynı sanal adres iki farklı fiziksel adrese karşılık gelir. Bu nedenle ön bellek her uygulama bağlam değişiminde tamamen temizlenmeli ya da bu adresin hangi sanal adres alanını ifade ettiğini belirlemek için ön belleğin her satırına ekstra bitler eklenmelidir.

Mantıksal ve fiziksel önbellek konusu karmaşık bir konudur ve bu kitabın kapsamı dışındadır. Daha derinlemesine bir tartışma için [CEKL97] ve [JACO08]'e bakınız.

## Önbellek Boyutu

Tablo 4.2'deki ikinci madde olan önbellek boyutu daha önce tartışılmıştı. Önbellek boyutunun, bit başına genel ortalama maliyetin yalnızca ana belleğe yakın olması için yeterince küçük ve genel ortalama erişim süresinin yalnızca önbelleğe yakın olması için yeterince büyük olmasını isteriz. Önbellek boyutunu en azı indirmek için başka motivasyonlar da vardır. Önbellek ne kadar büyük olursa, önbelleğin adreslenmesinde kullanılan kapı sayısı da o kadar büyük olur. Sonuç olarak, büyük önbellekler küçük olanlardan biraz daha yavaş olma eğilimindedir - aynı entegre devre teknolojisi ile üretilmiş ve çip ve devre kartı üzerinde aynı yere yerleştirilmiş olsalar bile. Mevcut çip ve kart alanı da önbellek boyutunu sınırlar. Önbelleğin performansı iş yükünün doğasına çok duyarlı olduğundan, tek bir "optimum" boyutuna imkansızdır. Tablo 4.3 bazı güncel ve geçmiş işlemcilerin önbellek boyutlarını listelemektedir.

## Haritalama Fonksiyonu

Ana bellek bloklarından daha az önbellek satırı olduğundan, ana bellek bloklarını önbellek satırlarına eşlemek için bir algoritma ihtiyaç vardır. Ayrıca, hangi ana bellek bloğunun o anda bir önbellek satırını işgal ettiğini belirlemek için bir araç gereklidir. Eşleme fonksiyonunun seçimi önbelleğin nasıl organize edileceğini belirler. Üç teknik kullanılabilir: doğrudan, ilişkisel ve set-ilişkisel. Bunların her birini sırayla inceleyeceğiz. Her durumda, genel yapıya ve ardından belirli bir örneğe bakacağız.

**ÖRNEK 4.2** Her üç durum için de örnek aşağıdaki unsurları içermektedir:

- Önbellek 64 kB tutabilir.
- Veriler ana bellek ve önbellek arasında her biri 4 baylıklı bloklar halinde aktarılır. Bu, önbelleğin her biri 4 baylıklı  $16\text{K} = 2^{14}$  satır olarak düzenlendiği anlamına gelir.
- Ana bellek 16 MB'den oluşur ve her bayt 24 bitlik bir adresle doğrudan adreslenebilir ( $2^{24} = 16\text{M}$ ). Dolayısıyla, eşleme amacıyla, ana belleğin her biri 4 baylıklı 4M bloktan olduğunu düşünebiliriz.

Tablo 4.3 Bazı İşlemcilerin Önbellek Boyutları

İşlemci	Tip	Giriş Yılı	L1 Önbellek <sup>a</sup>	L2 Önbellek	L3 Önbellek
IBM 360/85	Ana Bilgisayar	1968	16-32 kB	-	-
PDP-11/70	Minibilgisayar	1975	1 kB	-	-
VAX 11/780	Minibilgisayar	1978	16 kB	-	-
IBM 3033	Ana Bilgisayar	1978	64 kB	-	-
IBM 3090	Ana Bilgisayar	1985	128-256 kB	-	-
Intel 80486	PC	1989	8 kB	-	-
Pentium	PC	1993	8 kB/8 kB	256-512 kB	-
PowerPC 601	PC	1993	32 kB	-	-
PowerPC 620	PC	1996	32 kB/32 kB	-	-
PowerPC G4	PC/sunucu	1999	32 kB/32 kB	256 kB ila 1 MB	2 MB
IBM S/390 G6	Ana Bilgisayar	1999	256 kB	8 MB	-
Pentium 4	PC/sunucu	2000	8 kB/8 kB	256 kB	-
IBM SP	Üst düzey sunucu/süper bilgisayar	2000	64 kB/32 kB	8 MB	-
CRAY MTA <sup>b</sup>	Süper Bilgisayar	2000	8 kB	2 MB	-
İtanyum	PC/sunucu	2001	16 kB/16 kB	96 kB	4 MB
Itanium 2	PC/sunucu	2002	32 kB	256 kB	6 MB
IBM POWER5	Üst düzey sunucu	2003	64 kB	1.9 MB	36 MB
CRAY XD-1	Süper Bilgisayar	2004	64 kB/64 kB	1 MB	-
IBM POWER6	PC/sunucu	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Ana Bilgisayar	2008	64 kB/128 kB	3 MB	24-48 MB
Intel Core i7 EE 990	İş istasyonu/ sunucu	2011	6 * 32 kB/ 32 kB	1,5 MB	12 MB
IBM zEnterprise 196	Mainframe/ sunucu	2011	24 * 64 kB/ 128 kB	24 * 1,5 MB	24 MB L3 192 MB L4

Notlar: <sup>a</sup>Eğik çizgi ile ayrılmış iki değer komut ve veri ifade eder. <sup>b</sup>Her iki önbellek de yalnızca komut önbellegidir; veri önbellegi yoktur.

DOĞRUDAN **EŞLEME** Doğrudan eşleme olarak bilinen en basit teknik, ana belleğin her bloğunu yalnızca bir olası önbellek satırına eşler. Eşleme şu şekilde ifade edilir

$$i \equiv j \text{ modulo } m$$

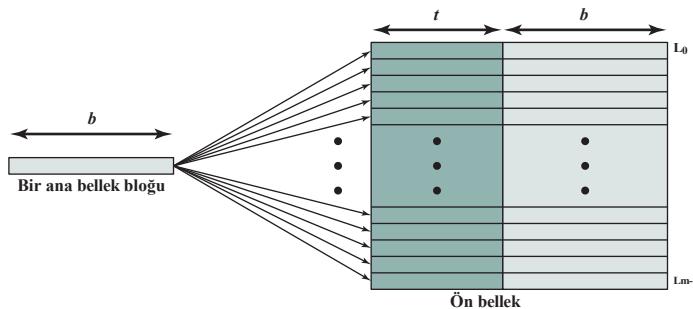
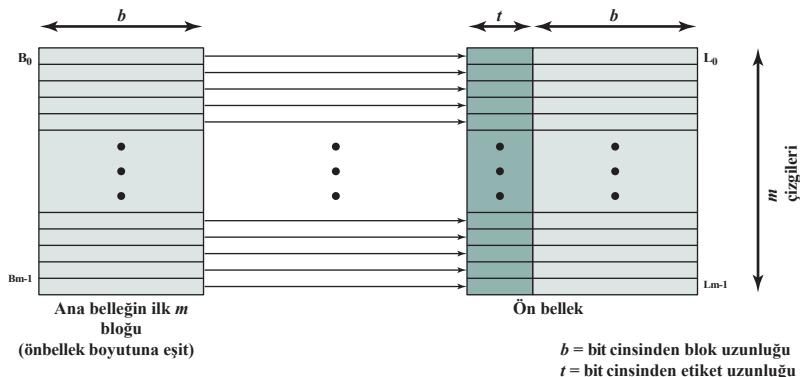
nerede

$i$ = önbellek satır numarası

$j$ = ana bellek blok numarası

$m$ = önbellekteki satır sayısı

Şekil 4.8a ana belleğin ilk  $m$  bloğu için eşlemeyi göstermektedir. Ana belleğin her bloğu önbellegin benzersiz bir satırına eşlenir. Sonraki  $m$  blok



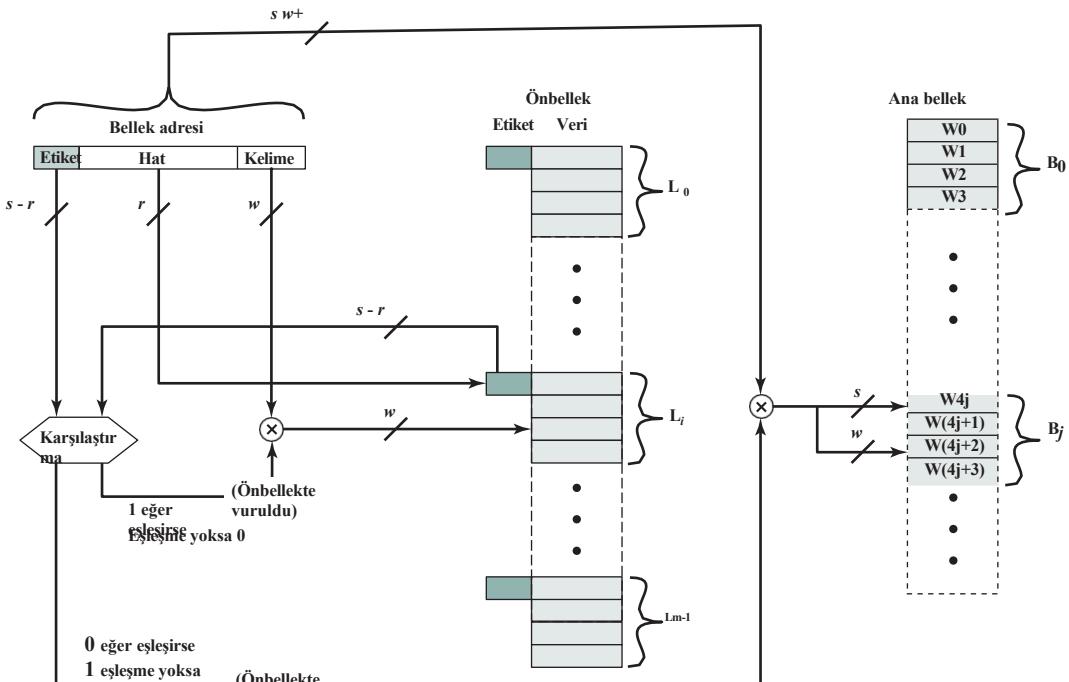
**Şekil 4.8** Ana Bellekten Önbellege Eşleme: Doğrudan ve İlişkisel

Ana belleğin  $B_m$  bloğu önbellegin  $L_0$  satırına,  $B_{m+1}$  bloğu  $L_{(1)}$  satırına eşlenir ve bu böyle devam eder.

Eşleme işlevi ana bellek adresi kullanılarak kolayca uygulanabilir. Şekil 4.9 genel mekanizmayı göstermektedir. Önbellek erişimi amacıyla, her ana bellek adresi üç alandan oluşuyormuş gibi görülebilir. En az anlamlı  $w$  bitleri ana bellek bloğu içindeki benzersiz bir kelime veya bayti tanımlar; çoğu çağdaş makinede adres bayt düzeyindedir. Kalan  $s$  bitleri ana belleğin  $2^s$  bloklarından birini belirtir. Önbellek mantığı bu  $s$  bitlerini bir etiket olarak yorumlar  $r$  bit (en anlamlı kısım) ve  $r$  bitlik bir satır alanı. Bu son alan, önbellegin  $m = 2^r$  satırlarından birini tanımlar. Özetlemek gerekirse,

- Adres uzunluğu=  $(s+w)$  bit
- Adreslenebilir birim sayısı=  $2^{(s)+w}$  kelime veya bayt
- Blok boyutu= satır boyutu=  $2^w$  sözcük veya bayt

- Ana blok sayısı =  $\frac{2^{s+w}}{2^w} = 2^{-s}$
- Önbellekteki satır sayısı=  $m = 2^r$
- Önbellek boyutu =  $2^{(r)+w}$  sözcük veya bayt
- Etiket boyutu=  $(s - r)$  bit



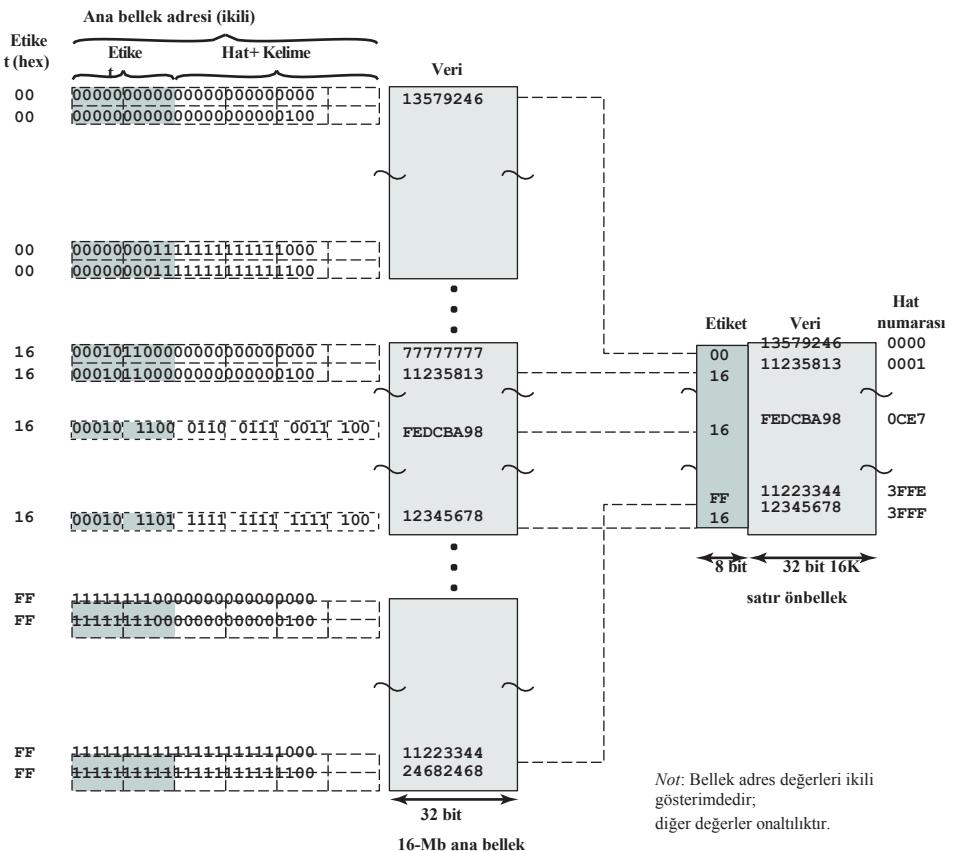
**Sekil 4.9** Doğrudan Eşleme (Yineleklek Organizasyonu)

**ÖRNEK 4.2a** Şekil 4.10 doğrudan eşleme kullanan örnek sistemimizi göstermektedir.<sup>5</sup> Örnekte,  $m = 16K = 2^{14}$  ve  $i = j$  modulo  $2^{14}$ . Eşleme su hale gelir.

<b>Önbellek Hattı</b>	<b>Bloğun Başlangıç Bellek Adresi</b>
0	000000, 010000, ..., FF0000
1	000004, 010004, ..., FF0004
<b>f</b>	<b>f</b>
$2^{14} - 1$	00FFFC, 01FFFC, ..., FFFFFC

Aynı satır numarasına eşlenen iki bloğun aynı etiket numarasına sahip olmadığını unutmayın. Dolayısıyla, başlangıç adresleri 000000, 010000, ..., FF0000 olan bloklar sırasıyla 00, 01, ..., FF etiket numaralarına sahiptir. Şekil 4.5'e geri dönersek, bir okuma işlemi aşağıdaki gibi çalışır. Önbellek sistemine 24 bitlik bir adres sunulur. 14 bitlik satır numarası, belirli bir satır erişmek için önbellekte bir dizin olarak kullanılır. Eğer 8 bitlik etiket numarası o anda o satırda saklanan etiket numarasıyla eşleşirse, o zaman 2 bitlik kelime numarası o satırda 4 bayttan birini seçmek için kullanılır. Aksi takdirde, 22 bitlik etiket artı satır alanı ana bellekten bir blok almak için kullanılır. Getirme işlemi için kullanılan gerçek adres, iki 0 bit ile birleştirilen 22 bitlik etiket-artı-satırdu, böylece bir blok sınırlarında basitleşen 4 bayt getirilir.

<sup>5</sup>Bu ve sonraki şekillerde, bellek değerleri onaltılık gösterimle gösterilmiştir. Sayı sistemleri (ondalık, ikili, ) hakkında temel bilgiler için Bölüm 9'a bakın.



**Şekil 4.10** Doğrudan Eşleme Örneği

Bu eşlemenin etkisi, ana bellek bloklarının önbellek satırlarına aşağıdaki gibi atanmasıdır:

Önbellek hattı	Atanan ana bellek blokları
0	$0, m, 2m, \lhd, 2^s - m$
1	$1, m+1, 2m+1, \lhd, 2^s - m+1$
f	f
$m - 1$	$m - 1, 2m - 1, 3m - , \lhd, 2^s - 1$

Böylece, adresin bir kısmının satır numarası olarak kullanılması, ana belleğin her bloğunun önbelleğe benzersiz bir şekilde eşlenmesini sağlar. Bir blok gerçekten

kendisine atanmış satırı okunduğunda, veriyi o satırı sigabilecek diğer bloklardan ayırt etmek için etiketlemek gereklidir. En önemli bitler bu amaca hizmet eder.

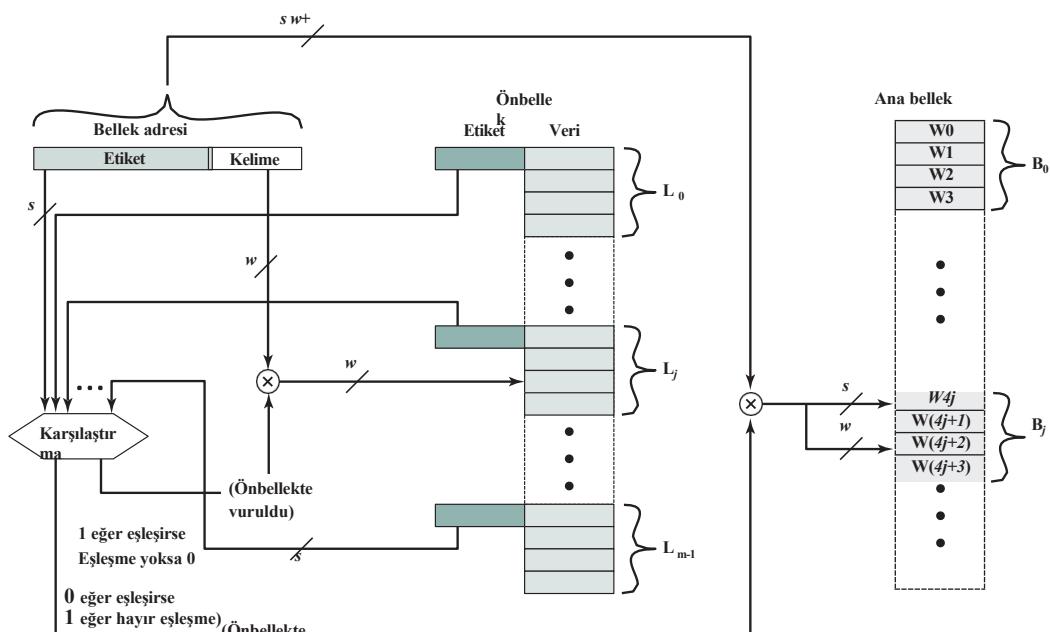
Doğrudan eşleme teknığının uygulanması basit ve ucuzdur. Ana dezavantajı, herhangi bir blok için sabit bir önbellek konumu olmasıdır. nedenle, bir program aynı satırda eşlenen iki farklı bloktan kelimele tekrar tekrar başvurursa, bloklar önbellekte sürekli olarak yer değiştirecek ve işaret oranı düşük olacaktır (*thrashing* olarak bilinen bir fenomen).



### Seçici Kurban Önbellek Simülatörü

**İskalama** cezasını azaltmaya yönelik bir yaklaşım, tekrar ihtiyaç duyulması halinde atılan veriyi hatırlamaktır. Atılan veri zaten getirilmiş olduğundan, küçük bir maliyetle tekrar kullanılabilir. Bu tür bir geri dönüşüm kurban kullanılarak mümkündür. Kurban önbelleği ilk olarak, hızlı erişim süresini etkilemeden doğrudan eşlenen önbelleklerin çıkışma kaçırmasını azaltmak için bir yaklaşım olarak önerilmiştir. Kurban önbelleği, boyutu tipik olarak 4 ile 16 önbellek satırı olan ve doğrudan eşlenen bir L1 ile bir sonraki bellek seviyesi arasında yer alan tamamen ilişkisel bir önbellektir. Bu kavram Ek F'de incelenmiştir.

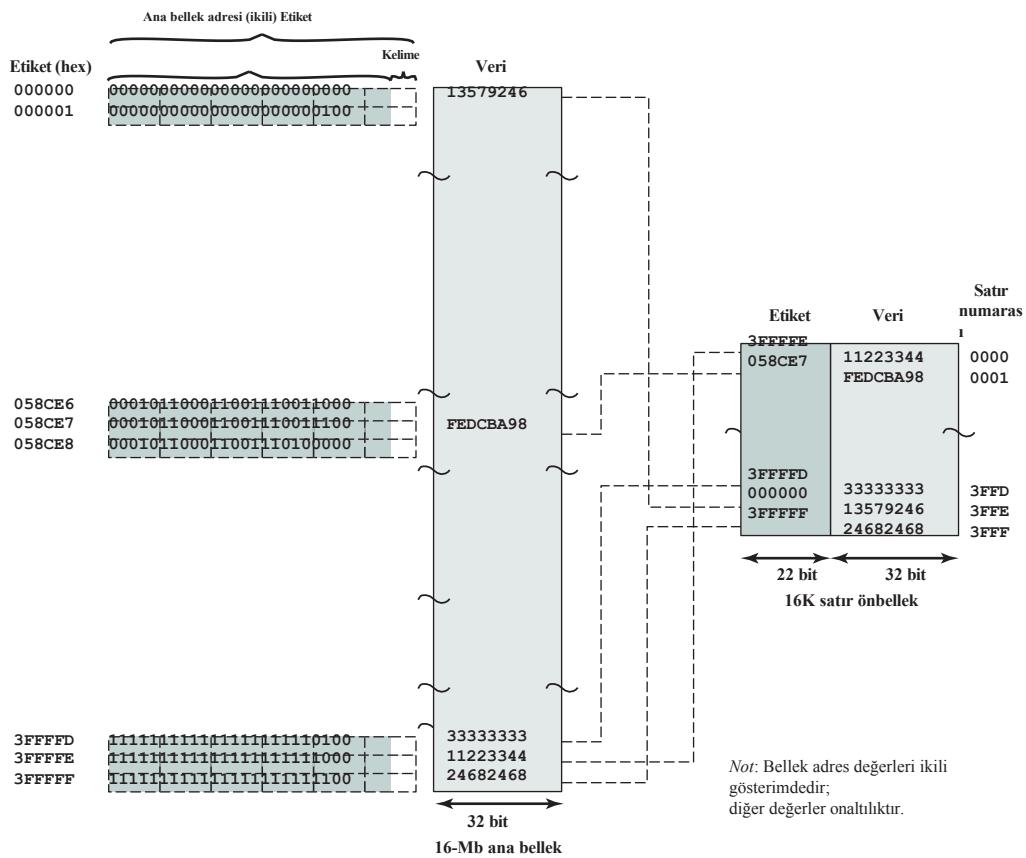
**İLİŞKİSEL EŞLEME** İlişkisel eşleme, her ana bellek bloğunun önbelleğin herhangi bir satırına yüklenmesine izin vererek doğrudan eşlemenin dezavantajının üstesinden gelir (Şekil 4.8b). Bu durumda, önbellek kontrol mantığı bir bellek adresini basitçe bir Etiket ve bir Kelime alanı olarak yorumlar. Etiket alanı bir ana bellek bloğunu bzersiz bir şekilde tanımlar. Bir bloğun önbellekte olup olmadığını belirlemek için, önbellek kontrol mantığı aynı anda her satırın etiketini bir eşleşme için incelemelidir. Şekil 4.11 mantığı göstermektedir.



Şekil 4.11 | Tam İlişkisel Önbellek Organizasyonu

**ÖRNEK 4.2b** Sekil 4.12 ilişkisel eşleme kullanan örneğimizi göstermektedir. Bir ana bellek adresi 22 bitlik bir etiket ve 2 bitlik bir bayt numarasından oluşur. 22 bitlik etiket, önbellekteki her satır için 32 bitlik veri bloğuyla birlikte saklanmalıdır. Etiketi oluşturanın adresin en soldaki (en anlamlı) 22 biti olduğuna dikkat edin. Böylece, 24 bitlik onaltılık 16339C adresi 22 bitlik 058CE7 etiketine sahiptir. Bu, ikili gösterimde kolayca görülebilir:

Bellek adresi	0001	0110	0011	0011	1001	1100	(ikili)
	1	6	3	3	9	C	(hex)
Etiket (en soldaki 22 bit)	00	0101	1000	1100	1110	0111	(ikili)
	0	5	8	C	E	7	(hex)



**Şekil 4.12** İlişkisel Eşleme Örneği

Adresteği hiçbir alanın satır numarasına karşılık gelmediğine dikkat edin, böylece önbellekteki satır sayısı adres biçimini tarafından belirlenmez. Özetlemek gerekirse,

- Adres uzunluğu =  $(s+w)$  bit
- Adreslenebilir birim sayısı =  $2^{(s+w)}$  kelime veya bayt
- Blok boyutu = satır boyutu =  $2^w$  sözcük veya bayt
- Ana bellekteki blok sayısı =  $\frac{2^{s+w}}{2^w} = 2^s$
- Önbellekteki satır sayısı = belirlenmemiş
- Etiket boyutu      bitler

İlişkisel eşleme ile, önbelleğe yeni bir blok okunduğunda hangi bloğun değiştirileceği konusunda esneklik vardır. Bu bölümde daha sonra ele alınan değiştirme algoritmaları isabet oranını en üst düzeye çıkarmak için tasarlanmıştır. İlişkisel eşlemenin temel dezavantajı, tüm önbellek satırlarının etiketlerini paralel olarak incelemek için gereken karmaşık devredir.

#### Önbellek Zaman Analizi S



**SET-ASSOCIATIVE MAPPING** Set-associative mapping, hem doğrudan hem de ilişkisel güçlü yönlerini sergilerken dezavantajlarını azaltan bir uzlaşmadır.

Bu durumda, önbellek, her biri dizi satırından oluşan sayı kümelerinden oluşur. İlişkiler şunlardır

$$\begin{aligned} m &= v * k \\ i &\equiv j \text{ modulo } v \end{aligned}$$

nerede

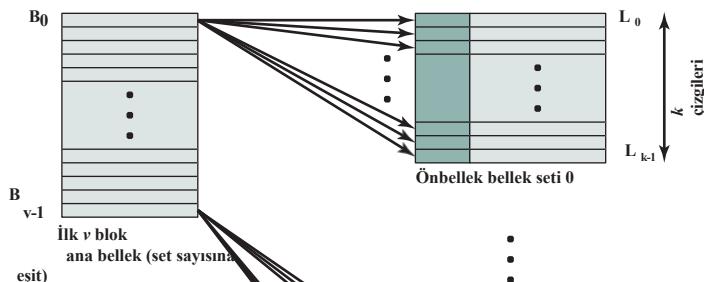
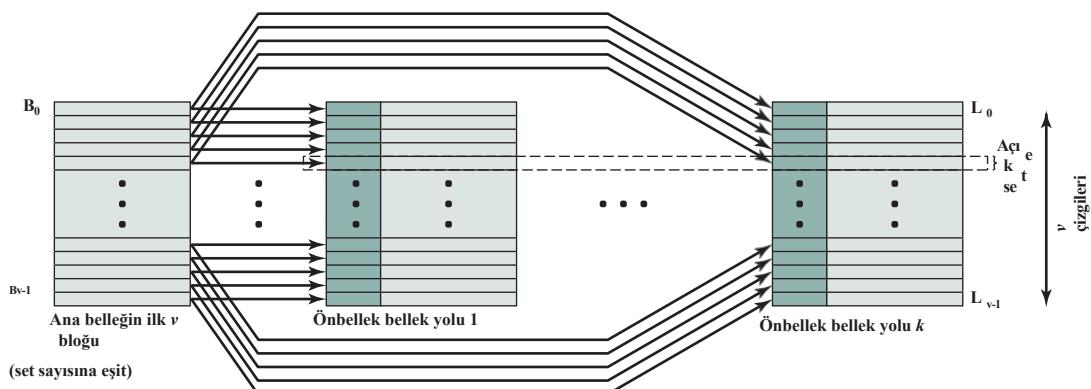
$i$  = önbellek seti numarası

$j$  = ana bellek blok numarası  $m$  =

önbellekteki satır sayısı  $v$  = set sayısı

$k$  = her setteki satır sayısı

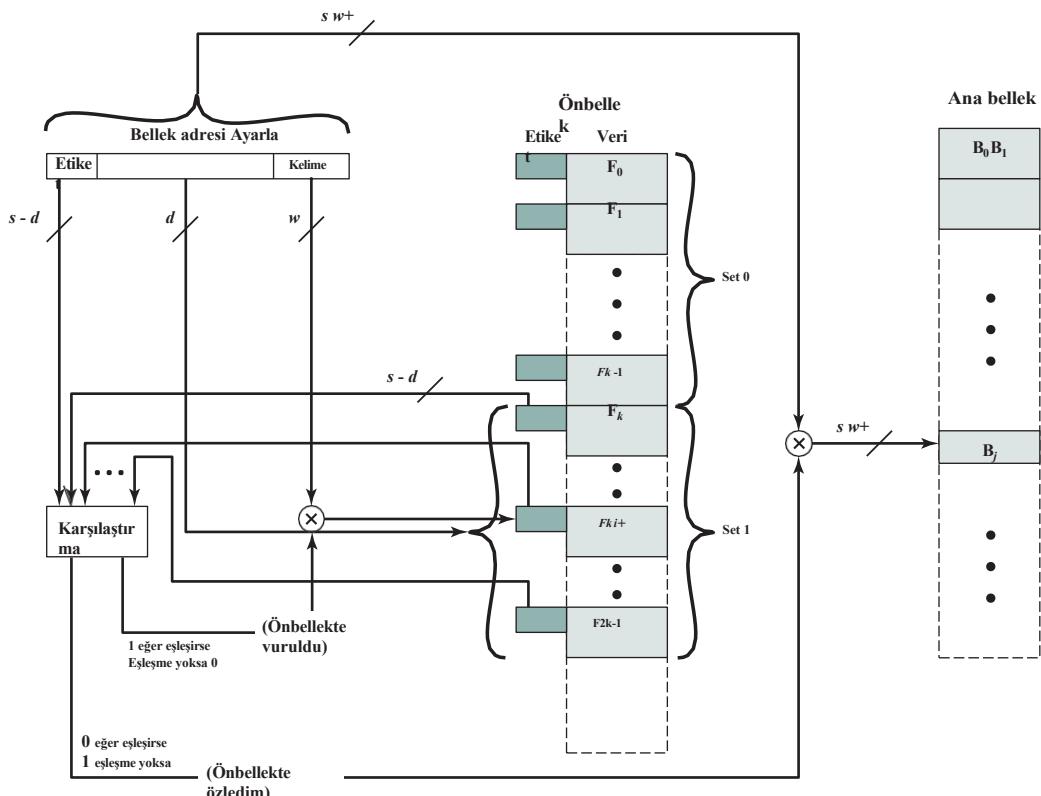
Buna  $k$ -yönlü set-ilişkili eşleme denir. Küme-ilişkili eşleme ile  $B_j$  bloğu  $j$  kümelerinin herhangi bir satırına eşlenebilir. Şekil 4.13a ana belleğin ilk  $v$  bloğu için bu eşlemeyi göstermektedir. İlişkisel eşlemede olduğu gibi, her kelime birden fazla önbellek satırına eşlenir. Küme-ilişkili eşleme için, her kelime belirli bir kümedeki tüm önbellek satırlarına eşlenir, böylece ana bellek bloğu  $B_0$ , 0 kümelerine eşlenir ve bu böyle devam eder. Böylece, set-ilişkilendirmeli önbellek fiziksel olarak  $v$  ilişkilendirmeli önbellek olarak uygulanabilir. *Küme-ilişkili* önbelleği Şekil 4.13b'de gösterildiği gibi  $k$  doğrudan eşlemeli önbellek olarak uygulamak da mümkündür. Her bir doğrudan eşlemeli önbellek  $v$  satırından oluşan bir *yol* olarak adlandırılır. Ana belleğin ilk  $v$  satırı her yolun  $v$  satırına doğrudan eşlenir; ana belleğin bir sonraki  $v$  satır grubu da benzer şekilde eşlenir ve bu böyle devam eder. Doğrudan eşlemeli uygulama tipik olarak kullanılır

(a)  $v$  ilişkisel eşlemeli önbellekler(b)  $k$  doğrudan eşlenmiş önbellekler**Şekil 4.13** Ana Bellekten Önbelleğe Eşleme:  $k$ -Yolu Küme İlişkisel

küçük birliktelik dereceleri ( $k$ 'nın küçük değerleri) için kullanılırken, birliktelik eşlemeli uygulama tipik olarak daha yüksek birliktelik dereceleri için kullanılır [JACO08].

Set-associative eşleme için, önbellek kontrol mantığı bir bellek adresini üç alan olarak yorumlar: Etiket, Set ve Kelime.  $d$  set bitleri  $v = 2^d$  birini belirtir. Tag ve Set alanlarının  $s$  bitleri ana belleğin  $2^s$  bloğundan birini belirtir. Şekil 4.14 önbellek kontrol mantığını göstermektedir. Tam ilişkilendirmeli eşleme ile, bir bellek adresindeki etiket oldukça büyütür ve önbellekteki her satırın etiketi ile karşılaştırılmalıdır.  $K$ -yönlü küme-ilişkili eşleme ile, bir bellek adresindeki etiket çok daha küçüktür ve yalnızca tek bir *küme* içindeki  $k$  etiketle karşılaşır. Özettemek gerekirse,

- Adres uzunluğu =  $(s + w)$  bit
- Adreslenebilir birim sayısı =  $2^{(s+w)}$  kelime veya bayt



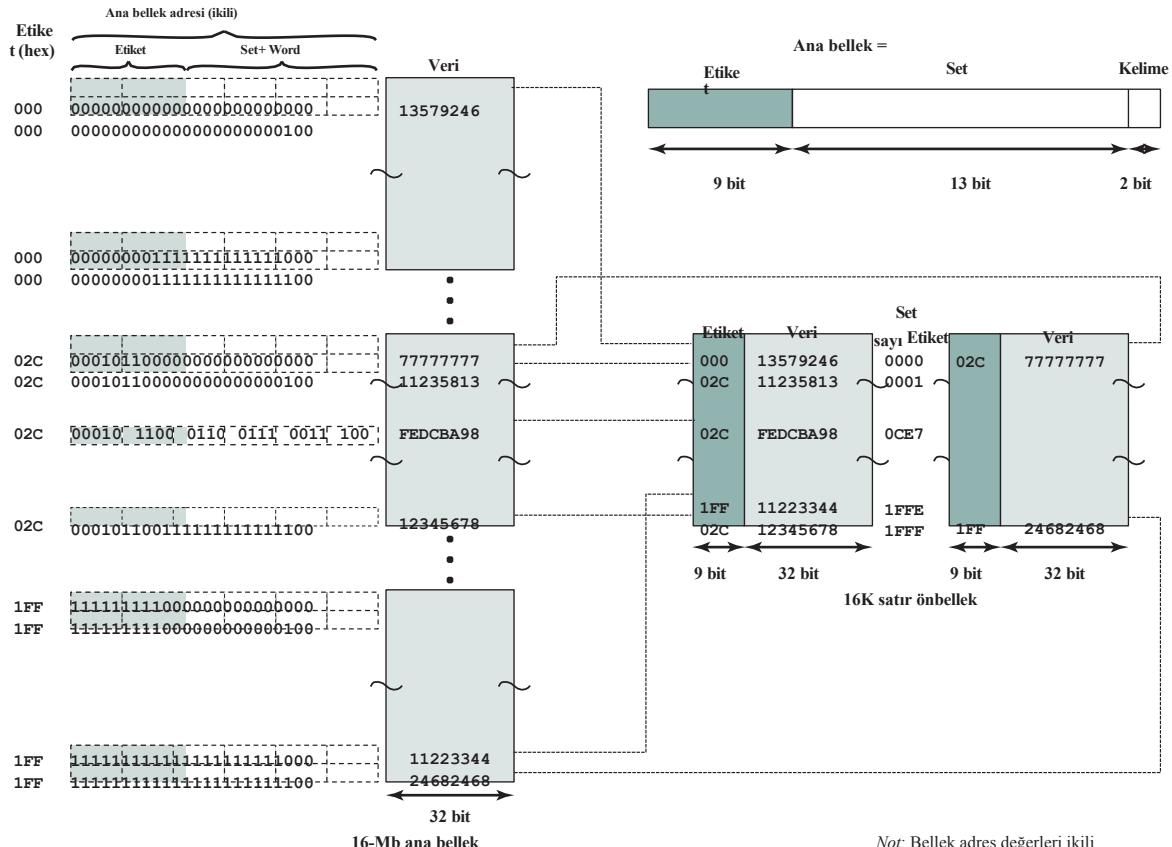
Şekil 4.14 k-Way Set-Associative Cache Organizasyonu

- Blok boyutu= satır boyutu=  $2^w$  sözcük veya bayt

$$\text{■ Ana blok sayısı } = \frac{2^{s+w}}{2^w} = 2^{-s}$$

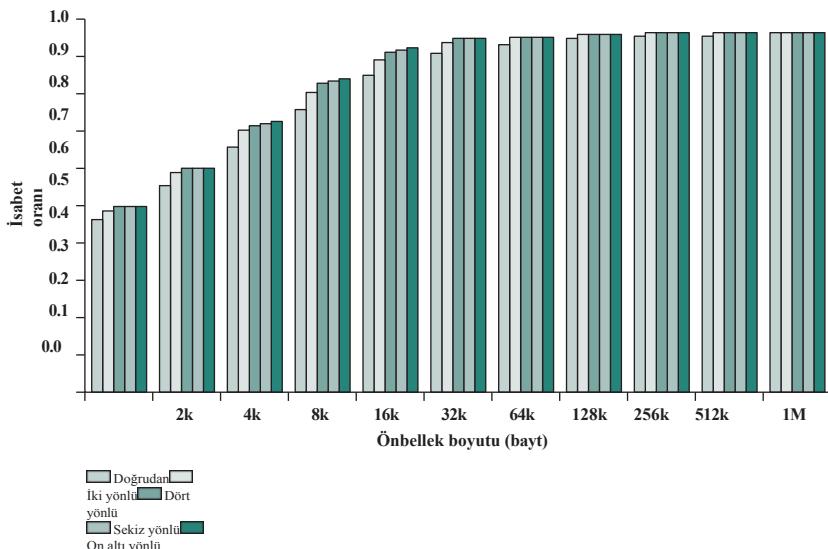
- Setteki satır sayısı=  $k$
- Set sayısı=  $v= 2^d$
- Önbellekteki satır sayısı=  $m= kv= k * 2^d$
- Önbellek boyutu=  $k * 2^{d+w}$  sözcük veya bayt
- Etiket boyutu=  $(s - d)$  bit

**ÖRNEK 4.2c** Şekil 4.15, iki yönlü set-associative olarak adlandırılan, her sette iki satır bulunan set-associative eşleme kullanılarak örneğimizi göstermektedir. 13 bitlik set numarası, önbellekteki iki satırdan oluşan benzersiz bir seti tanımlar. Ayrıca ana bellekteki bloğun modulo  $2^{13}$  sayısını da verir. Bu, blokların satırlara eşlenmesini belirler. Böylece, ana bellekteki 000000, 008000, ..., FF8000 blokları önbellek seti 0'a eşlenir. Bu bloklardan herhangi biri setteki iki satırdan birine yüklenebilir. Aynı önbellek setine eşlenen iki bloğun aynı etiket numarasına sahip olmadığını unutmayın. Bir okuma işlemi için, 13 bitlik set numarası hangi iki satır setinin inceleneceğini belirlemek için kullanılır. Kümdeki her iki satır da erişilecek adresin etiket numarasıyla eşleşip eşleşmediği açısından incelenir.



*Not:* Bellek adres değerleri ikili gösterimdedir;  
diğer değerler onaltılıktır.

Şekil 4.15 İki Yönlü Set-Associative Eşleme Örneği



**Şekil 4.16** Önbellek Boyutuna Göre Değişen Birliktelik

$V = m$ ,  $k = 1$  üç durumunda, küme-ilişkilendirme tekniği doğrudan eşlemeye ve  $v = 1$ ,  $k = m$  için ilişkisel eşlemeye indirgenir. Küme başına iki satır kullanımı ( $v = m/2$ ,  $k = 2$ ) en yaygın küme-ilişkilendirme organizasyonudur. Doğrudan eşlemeye göre isabet oranını önemli ölçüde artırır. Dört yönlü *küme* ilişkilendirme ( $v = m/4$ ,  $k = 4$ ) nispeten küçük bir ek maliyet karşılığında mütevazi bir ek iyileştirme sağlar [MAYB84, HILL89]. Küme başına satır sayısındaki daha fazla artışın çok az etkisi vardır.

Şekil 4.16, önbellek boyutunun bir fonksiyonu olarak set-associative önbellek performansının bir simülasyon çalışmasının sonuçlarını göstermektedir [GENU04]. Doğrudan ve iki yönlü set ilişkilendirmeli arasındaki performans farkı en az 64 kB önbellek boyutuna kadar önemlidir. Ayrıca, 4 kB'de iki yönlü ve dört yönlü arasındaki farkın, önbellek boyutunda 4 kB'den 8 kB'ye geçişteki farktan çok daha az olduğuna dikkat edin. Önbelleğin karmaşıklığı ilişkilendirilebilirlikle orantılı olarak artar ve bu durumda önbellek boyutunun 8 hatta 16 kB'a çıkarılmasına karşı haklı bir gerekçe olmayacağıdır. Dikkat edilmesi gereken son bir nokta, yaklaşık 32 kB'ın ötesinde, önbellek boyutundaki artışın performansta önemli bir artış getirmediğidir.

Şekil 4.16'daki sonuçlar bir GCC derleyicisinin yürütülmesinin simülasyonuna dayanmaktadır. Farklı uygulamalar farklı sonuçlar verebilir. Örneğin, [CANT01] CPU2000 SPEC kıyaslamalarının birçoğunu kullanarak önbellek performansı sonuçlarını rapor etmektedir. CANT01'in isabet oranı ile önbellek boyutunu karşılaştırılan sonuçları Şekil 4.16 ile aynı modeli takip eder, ancak belirli değerler biraz farklıdır.



Önbellek Simülatörü  
Çok Görevli Önbellek Simülatörü

## Değiştirme Algoritmaları

Önbellek doldurulduktan sonra, önbelleğe yeni bir blok getirildiğinde, mevcut bloklardan birinin değiştirilmesi gereklidir. Doğrudan eşleme için, herhangi bir blok için yalnızca bir olası satır vardır ve seçim yapmak mümkün değildir. Birleştirmeli ve set-birleştirmeli teknikler için bir değiştirme algoritması gereklidir. Yüksek hız elde etmek için, böyle bir algoritma donanımda uygulanmalıdır. Bir dizi algoritma denemistiştir. En yaygın dört tanesinden bahsedeceğiz. Muhtemelen en etkili olanı en **son kullanımlı (LRU)**: Önbellekte en uzun süredir bulunan ve kendisine hiç atıfta bulunulmayan kümedeki bloğu değiştirir. İki yönlü küme ilişkilendirmesi için bu kolayca uygulanabilir. Her satır bir USE biti içerir. Bir satır referans verildiğinde, USE biti 1'e ayarlanır ve bu kümedeki diğer satırın USE biti 0'a ayarlanır. Kümeye bir blok okunacağı zaman, USE biti 0 olan satır kullanılır. Daha yakın zamanda kullanılan bellek konumlarına başvurulma olasılığının daha yüksek olduğunu varsayıdığımız için, LRU en iyi isabet oranını vermelidir. LRU'nun tam ilişkisel bir önbellek için uygulanması da nispeten kolaydır. Önbellek mekanizması, önbellekteki tüm satırlar için ayrı bir dizin listesi tutar. Bir satırı başvurulduğunda, listenin önüne geçer. Değiştirme için listenin en arkasındaki satır kullanılır. Uygulama kolaylığı nedeniyle LRU en popüler değiştirme algoritmasıdır.

Başka bir olasılık da ilk giren ilk çıkar (FIFO): Önbellekte en uzun süre kalan kümedeki bloğu değiştirir. FIFO, round-robin veya devreli tampon teknigi olarak kolayca uygulanabilir. Bir başka olasılık da en az sıklıkla kullanılan (LFU): Kümede en az referansla karşılaşan bloğu değiştirir. LFU, her satırı bir sayı ile ilişkilendirerek uygulanabilir. Kullanıma dayalı olmayan bir teknik (yani LRU, LFU, FIFO veya başka bir varyant değil) aday satırlar arasından rastgele bir satır seçmektir. Simülasyon çalışmaları, rastgele değiştirmenin kullanımına dayalı bir algoritmanın yalnızca biraz daha düşük performans sağladığını göstermiştir [SMIT82].

## Politika Yazın

Önbellekte yer alan bir blok değiştirileceği zaman, dikkate alınması gereken iki durum vardır. Önbellekteki eski blok değiştirilmemişse, önce eski yazılmadan yeni bir blokla üzerine yazılabilir. Önbelleğin o satırındaki bir kelime üzerinde en az bir yazma işlemi gerçekleştirilmişse, yeni blok getirilmeden önce önbellek satırı bellek bloğuna yazılarak ana bellek güncellenmelidir. Performans ve eko-nomik ödüntüleşmelerle çeşitli yazma politikaları mümkündür. Karşılaşılması gereken iki sorun vardır. Birincisi, birden fazla cihazın ana belleğe erişimi olabilir. Örneğin, bir I/O modülü doğrudan belleğe okuma-yazma yapabilir. Bir sözcük yalnızca önbellekte değiştirilmişse, karşılık gelen bellek sözcüğü geçersizdir. Ayrıca, G/C cihazı ana belleği değiştirmişse, önbellek sözcüğü geçersizdir. Aynı veri yoluna birden fazla işlemci bağlılığında ve her işlemcinin kendi yerel önbelleği olduğunda daha karmaşık bir sorun ortaya çıkar. Bu durumda, bir bir kelime değiştirilirse, diğer önbelleklerdeki bir kelime geçersiz hale gelebilir.

En basit teknik **write through olarak** adlandırılır. Bu teknigi kullanarak, tüm yazma işlemleri önbelleğin yanı sıra ana belleğe de yapılır ve ana belleğin her zaman geçerli olması sağlanır. Diğer herhangi bir işlemci önbellek modülü, kendi önbelleği içinde tutarlılığı korumak için ana belleğe giden trafiği izleyebilir. Ana dezavantajı

Bu teknigin en büyük dezavantajı, önemli miktarda bellek trafigi oluşturması ve bir darboğaz yaratabilmesidir. **Geri yazma** olarak bilinen alternatif bir teknik, bellek yazmalarını en aza indirir. Geri yazma ile güncellemeler yalnızca önbellekte yapılır. Bir güncelleme gerçekleştiğinde, satırla ilişkili bir **kırıcı bit** ya da **kullanım biti** ayarlanır. Ardından, bir blok değiştirildiğinde, yalnızca kırıcı bit ayarlanmışa ve ayarlanmışa ana belleğe geri yazılır. Geri yazma ile ilgili sorun, ana belleğin bazı bölmelerinin geçersiz olması ve dolayısıyla G/C modüllerinin erişimlerine yalnızca önbellek üzerinden izin verilebilmesidir. Bu da karmaşık bir devre ve potansiyel bir darboğaz yaratır. Deneyimler, yazma olan bellek referanslarının yüzdesinin %15 civarında olduğunu göstermiştir [SMIT82]. Ancak, HPC uygulamaları için bu sayı %33'e yaklaşabilir (vektör-vektör çarpımı) ve %50'ye kadar çıkabilir (matris transpozisyonu).

**ÖRNEK 4.3** Satır boyutu 32 bayt bir önbellek ve 4 baylıklı bir sözcüğü aktarmak için 30 ns gerektiren bir ana bellek düşünün. Önbellekten çıkarılmadan önce en az bir kez yazılan herhangi bir satır için, bir geri yazma önbelleğinin yazma yoluya önbellekten daha verimli olması için satırın çıkarılmadan önce ortalama kaç kez yazılması gereklidir?

Geri yazma durumu için, her kırıcı satır takas zamanında bir kez geri yazılır, bu da  $8 * 30 = 240$  ns sürer. Yazma durumu için, satırın her güncellenmesi bir kelimenin ana belleğe yazılmasını gerektirir ve bu da 30 ns sürer. Bu nedenle, en az bir kez yazılan ortalama satır takas öncesinde 8'den fazla kez, geri yazma daha verimli olur.

Birden fazla cihazın (tipik olarak bir işlemci) bir önbelleğe sahip olduğu ve ana belleğin paylaşıldığı bir veri yolu organizasyonunda, yeni bir sorun ortaya çıkar. Bir önbellekteki veriler değiştirilirse, bu sadece ana bellekteki ilgili kelimeyi değil, aynı zamanda diğer önbelleklerdeki aynı kelimeyi de geçersiz kılar (eğer başka bir önbellekte aynı kelime varsa). Bir write-through politikası kullanılsa bile, diğer önbellekler geçersiz veri içerebilir. Bu sorunu önleyen bir sistemin önbellek tutarlılığını koruduğu söylenir. Önbellek tutarlılığı için olası yaklaşımlar aşağıdakileri içerir:

- **Yazma ile veri yolu izleme:** Her önbellek denetleyicisi, diğer veri yolu yöneticilerinin belleğe yazma işlemlerini tespit etmek için adres hatlarını izler. Başka bir master, önbellekte de bulunan paylaşılan bellekteki bir konuma yazarsa, önbellek denetleyicisi bu önbellek girişini geçersiz kılar. Bu strateji, tüm önbellek denetleyicileri tarafından bir write-through politikasının kullanılmasına bağlıdır.
- **Donanım şeffaflığı:** Önbellek aracılığıyla ana belleğe yapılan tüm güncellemelerin tüm yansıtılması sağlamak için ek donanım kullanılır. Böylece, bir önbellekçi kendi bir kelimeyi değiştirirse, bu güncelleme ana belleğe yazılır. Buna ek olarak, diğer önbelleklerdeki eşleşen kelimeler de benzer şekilde güncellenir.
- **Önbelleğe alınamayan bellek:** Ana belleğin yalnızca bir kısmı birden fazla işlemci tarafından paylaşıılır ve bu önbelleğe alınamaz olarak tanımlanır. Böyle bir sistemde, paylaşılan belleğe yapılan tüm erişimler önbellek iskalamalarıdır, çünkü paylaşılan bellek hiçbir zaman önbelleğe kopyalanmaz. Önbelleğe alınamayan bellek, yonga seçme mantığı veya yüksek adres bitleri kullanılarak tanımlanabilir.

Önbellek tutarlılığı aktif bir araştırma alanıdır. Bu konu Beşinci Bölümde daha ayrıntılı olarak incelenmektedir.

## Hat Boyutu

Bir diğer tasarım ögesi de satır boyutudur. Bir veri bloğu alınıp önbellege yerleştirildiğinde, sadece istenen kelime değil, aynı zamanda belirli sayıda komşu kelime de . Blok boyutu çok küçük boyutlardan daha büyük boyutlara arttıkça, referans verilen bir kelimenin verilerin yakın gelecekte referans vereceğini belirten yerellik ilkesi nedeniyle isabet oranı ilk başta artacaktır. Blok boyutu arttıkça, önbellege daha faydalı veriler getirilir. Bununla birlikte, blok daha da büyütükçe ve yeni getirilen bilgilerin kullanılma olasılığı, değiştirilmesi gereken bilgilerin yeniden kullanılma olasılığından daha az hale geldikçe isabet oranı düşmeye başlayacaktır. İki özel etki girer:

- Daha büyük bloklar önbellege siyan blok sayısını azaltır. Her blok getirme işlemi eski önbellek içeriklerinin üzerine yazıldıgından, az sayıda blok getirildikten kısa bir süre sonra verilerin üzerine yazılmasına neden olur.
- Bir blok büyütükçe, her ek kelime talep edilen kelimedenden daha uzaktır ve bu nedenle yakın gelecekte ihtiyaç duyulması daha az olasıdır.

Blok boyutu ve isabet oranı arasındaki ilişki, belirli bir programın yerellik özelliklerine bağlı karmaşıktır ve kesin bir optimum değer bulunamamıştır. 8 ila 64 bayt arasında bir boyut optimuma oldukça yakın görülmektedir [SMIT87, PRZY88, PRZY90, HAND98]. HPC sistemleri için 64 ve 128 baytlık önbellek satır boyutları en sık kullanılanlardır.

## Önbellek Sayısı

Önbellekler ilk ortaya çıktıığında, tipik bir sistemde tek bir önbellek vardı. Daha yakın zamanlarda, birden fazla önbellek kullanımı norm haline gelmiştir. Bu tasarım sorununun iki yönü, önbellek seviyelerinin sayısı ve birleşik ve bölünmüş önbelleklerin kullanımı ile ilgilidir.

**MULTILEVEL *cAcHES*** Mantık yoğunluğu arttıkça, işlemciyle aynı çip üzerinde bir sahip olmak mümkün hale gelmiştir: çip üstü önbellek. Harici bir veri yolu üzerinden erişilebilen bir önbellek ile karşılaşıldığında, yonga üstü önbellek işlemecinin harici veri yolu etkinliğini azaltır ve bu nedenle yürütme sürelerini hızlandırır ve genel sistem performansını artırır. İstenen komut veya veri yonga üstü önbellekte bulunduğuunda, veri yolu erişimi ortadan kalkar. Veri yolu uzunluklarıyla karşılaşıldığında işlemecinin içindeki kısa veri yolları nedeniyle, yonga üstü önbellek erişimleri sıfır beklemeye durumundaki veri yolu döngülerinden bile önemli ölçüde daha hızlı tamamlanacaktır. Ayrıca, bu süre boyunca veri yolu diğer aktarımı desteklemek için serbesttir.

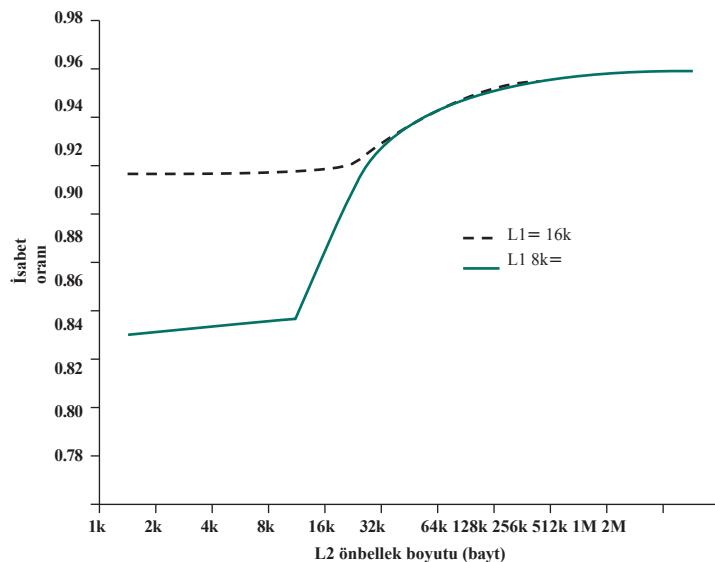
Cip üstü önbellegin dahil edilmesi, cip dışı veya harici önbellegin hala arzu edilip edilmediği sorusunu açık bırakır. Tipik olarak cevap evettir ve çoğu çağdaş tasarım hem yonga üstü hem de harici önbellekler içerir. Bu tür en basit organizasyon iki seviyeli önbellek olarak bilinir; dahili seviye 1 (L1) ve harici önbellek seviye 2 (L2) olarak adlandırılır. L2 önbellegin dahil edilmesinin nedeni aşağıdaki gibidir: L2 önbeliği yoksa ve işlemci L1 önbelleginde olmayan bir bellek konumu için erişim talebinde bulunursa, işlemci DRAM'e veya

ROM belleği veri yolu boyunca. Tipik olarak yavaş veri yolu hızı ve yavaş bellek erişim süresi nedeniyle, bu düşük performansla sonuçlanır. Öte yandan, bir L2 SRAM (statik RAM) önbelleyi kullanılırsa, sıklıkla eksik bilgiler hızlı bir şekilde geri alınabilir. SRAM veri yolu hızına uyacak kadar hızlısa, verilere en hızlı veri yolu aktarımı türü olan sıfır bekleme durumu işlemi kullanılarak erişilebilir.

Çok seviyeli önbellekler için çağdaş önbellek tasarımlının iki özelliği dikkate değerdir. Birincisi, çip dışı bir L2 önbellek için, birçok tasarım L2 önbellek ve işlemci arasındaki transfer yolu olarak sistem veriyolunu kullanmaz, ancak sistem veriyolundaki yükü azaltmak için ayrı bir veri yolu kullanır. İkinci olarak, işlemci bileşenlerinin küçülmeye devam etmesiyle birlikte, bazı işlemciler artık L2 önbellegini işlemci yongasına dahil ederek performansı artırmaktadır.

L2 önbellek kullanımından kaynaklanan potansiyel tasarruf, hem L1 hem de L2 önbelleklerdeki isabet oranlarına bağlıdır. Birçok çalışma, genel olarak ikinci seviye önbellek kullanımının performansı artırdığını göstermiştir (örneğin, bkz. [AZIM92], [NOVI93], [HAND98]). Bununla birlikte, çok seviyeli önbelleklerin kullanımı, boyut, değiştirme algoritması ve yazma politikası dahil olmak üzere önbelleklerle ilgili tüm tasarım konularını karmaşıklığından, tartışmalar için [HAND98] ve [PEIR99]'a bakın.

Şekil 4.17, önbellek boyutunun bir fonksiyonu olarak iki seviyeli önbellek performansına ilişkin bir simülasyon çalışmasının sonuçlarını göstermektedir [GENU04]. Şekilde her iki önbellegin de aynı satır boyutuna sahip olduğu varsayılmakta ve toplam isabet oranı gösterilmektedir. Yani, istenen veri hem L1 hem de L2 önbellekte görüntüyorsa bir isabet sayılır. Şekil, L1 boyutuna göre L2'nin toplam isabet üzerindeki etkisini göstermektedir. L2, L1 önbellek boyutunun en az iki katına çıkan kadar toplam önbellek isabet sayısını üzerinde çok az etkiye sahiptir. Eğimin 8 kB'lık bir L1 önbellesi için en dik kısmının 16 kB'lık bir L2 önbellesi için olduğuna dikkat edin. Yine 16 kB'lık bir L1 önbellesi için eğrinin en dik kısmı 32 kB'lık bir L2 önbellek boyutu içindir. Bu noktadan önce, L2 önbellegin toplam önbellek performansı üzerinde çok az etkisi vardır. L2 önbellegin aşağıdakilerden daha büyük olması gereklidir.



Şekil 4.17 8-kB ve 16-kB L1 için Toplam Isabet Oranı (L1 ve L2)

L1 önbelleğinin performansı etkilemesi mantıklıdır. L2 önbelleği L1 önbelleği ile aynı satır boyutuna ve kapasitesine sahipse, içeriği L1 önbelleğinin içeriğini aşağı yukarı yansıtacaktır. Önbellek için çip üzerinde kullanılabılır alanın artmasıyla birlikte, çoğu geçici mikroişlemci L2 önbelleği işlemci çipine taşmış ve bir L3 eklemiştir. Başlangıçta, L3 önbelleğe harici veri yolu üzerinden erişilebiliyordu. Daha yakın zamanlarda, çoğu mikroişlemci çip üzerinde bir L3 önbellek eklemiştir. Her iki durumda da üçüncü seviyenin eklenmesinin bir performans avantajı olduğu görülmektedir (örneğin, bkz. [GHA198]). Ayrıca, IBM ana bilgisayar zEnter-prise sistemleri gibi büyük sistemler artık 3 çip üstü önbellek seviyesi ve dördüncü bir önbellek seviyesi içermektedir birden fazla çip arasında paylaşılır [CURR11].

**BİRLEŞİK VE BÖLÜNMÜŞ ÖNBELLEKLER** Çip üstü önbellek ilk ortaya çıktığında, tasarımların çoğu hem veri hem de talimatlara referansları saklamak için kullanılan tek bir önbellekten oluşuyordu. Daha yakın zamanlarda, önbelleği ikiye ayırmak yaygın hale gelmiştir: biri talimatlara, diğeri verilere ayrılmıştır. Bu iki önbellek de aynı seviyede, tipik olarak iki L1 önbellek olarak bulunur. İşlemci ana bellekten bir komut almaya çalıştığında, ilk olarak komut L1 önbelleğine başvurur ve işlemci ana bellekten veri almaya çalıştığında ilk olarak veri L1 önbelleğine başvurur.

Birleştirilmiş bir önbelleğin iki potansiyel avantajı vardır:

- Belirli bir önbellek boyutu için, birleşik bir önbellek bölünmüş önbelleklerden daha yüksek bir isabet oranına sahiptir çünkü komut ve veri getirmeleri arasındaki yükü otomatik olarak dengeler. Yani, bir yürütme modeli veri getirme işlemlerinden çok daha fazla komut getirme işlemi içeriyorsa, önbellek talimatlarla dolma eğiliminde olacaktır ve bir yürütme modeli nispeten daha fazla veri getirme işlemi içeriyorsa, bunun tersi gerçekleşecektir.
- Yalnızca bir önbellek tasarlanmalı ve uygulanmalıdır.

Eğerim, L1'de bölünmüş önbelleklere ve daha yüksek seviyeler için birleşik önbelleklere, özellikle de paralel komut yürütmemeyi ve gelecekteki tahmini önceden alınmasını vurgulayan süperskalar makinelere yönelikdir. Bölünmüş önbellek tasarımının en önemli avantajı, komut getirme/kod çözme birimi ile yürütme birimi arasındaki önbellek çakışmasını ortadan kaldırmasıdır. Bu, komutların ardışık sıralanmasına dayanan herhangi bir tasarımda önemlidir. Tipik olarak, işlemci talimatları önceden alır ve bir tamponu ya da ardışık düzeni yürütülecek talimatlarla doldurur. Şimdi birleşik bir komut/veri önbelleğimiz olduğunu varsayıyalım. Yürütme birimi veri yüklemek ve saklamak için bir bellek erişimi gerçekleştirdiğinde, istek birleşik önbelleğe gönderilir. Aynı anda, komut ön hazırlayıcısı bir talimat için önbelleğe bir okuma isteği gönderirse, bu istek geçici olarak engellenir, böylece önbellek önce yürütme birimine hizmet vererek o anda yürütülmekte olan talimatı tamamlamasını sağlar. Bu önbellek çakışması, komut hattının verimli kullanımını engelleyerek performansı düşürebilir. Bölünmüş önbellek yapısı bu zorluğun üstesinden gelir.

## 4.4 PENTIUM 4 ÖNBELLEK ORGANİZASYONU

Önbellek organizasyonunun evrimi Intel mikro işlemcilerinin evriminde açıkça görülmektedir (Tablo 4.4). 80386 çip üzerinde bir önbellek içermez. 80486, 16 baylıklı bir satır boyutu ve dört yönlü bir önbellek kullanarak 8 kB'lık tek bir çip üstü önbellek içerir.

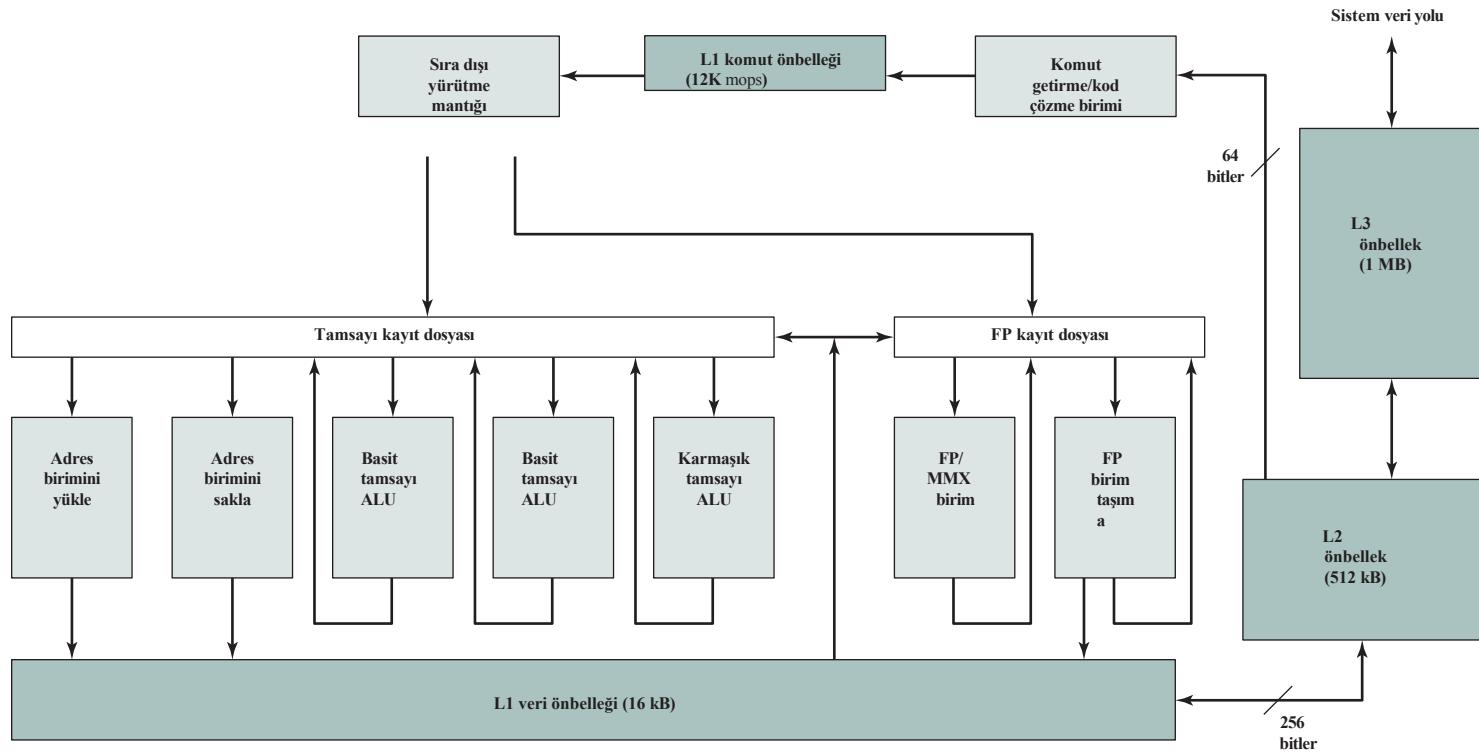
**Tablo 4.4** Intel Önbellek Evrimi

Problem	Çözüm	Özelliğin İlk Göründüğü İşlemci
Sistem veriyolundan daha yavaş harici bellek.	Daha hızlı bellek teknolojisi kullanarak harici önbellek ekleyin.	386
Artan işlemci hızı, harici veri yolunun önbellek erişimi için bir darboğaz haline gelmesine neden olur.	İşlemciyle aynı hızda çalışan harici önbellegi çip üzerine taşıyın.	486
Dahili önbellek, çip üzerindeki sınırlı alan nedeniyle oldukça küçüktür.	Ana bellekten daha hızlı teknoloji kullanan harici L2 önbellek ekleyin.	486
Çekişme, hem Komut Hazırlayıcı hem de Birimi aynı anda önbelleğe erişmek istediğiinde ortaya çıkar. Bu durumda, Yürütmeye Biriminin veri erişimi gerçekleşirken Ön Hazırlayıcı durur.	Ayrı veri ve talimat önbellekleri oluşturun.	Pentium
Artan işlemci hızı, harici veri yolunun L2 önbellek erişimi için bir darboğaz haline gelmesine neden olur.	Ana (ön taraf) harici veri daha yüksek hızda çalışan ayrı bir arka taraf veri yolu oluşturun. BSB, L2 önbellegine ayrılmıştır.	Pentium Pro
	L2 önbelleğini işlemci çipine taşıyın.	Pentium II
Bazı uygulamalar devasa veritabanlarıyla çalışır ve büyük miktarda veriye hızlı bir şekilde erişmeleri gereklidir. Çip üzerindeki önbellekler çok küçüktür.	Harici L3 önbellek ekleyin.	Pentium III
	L3 önbelleğini çip üzerine taşıyın.	Pentium 4

set-associative organizasyonu. Tüm Pentium işlemciler, biri veri diğeri talimatlar için olmak üzere iki adet yonga üstü L1 önbellek içerir. Pentium 4 için L1 veri önbelleği 16 kB olup 64 baytlik bir satır boyutu ve dört yönlü bir set-associative organizasyonu kullanır. Pentium 4 komut önbelleği daha sonra sonra açıklanmaktadır. Pentium II ayrıca her iki L1 önbelleğini de besleyen bir L2 önbelleği içerir. L2 önbellek 512 kB boyutunda ve 128 bayt satır boyutunda sekiz yolu set ilişkiseldir. Pentium III için bir L3 önbellek eklenmiş ve Pentium 4'ün üst düzey sürümleriley birlikte çip üzerinde yer almıştır.

Şekil 4.18, Pentium 4 organizasyonunun basitleştirilmiş bir görünümünü sunmakta ve üç önbelleğin yerleşimini aydınlatmaktadır. İşlemci çekirdeği dört ana bileşenden oluşur:

- **Getirme/çözme birimi:** Program talimatlarını L2 önbelleğinden sırayla alır, bunları bir dizi mikro işleme dönüştürür ve sonuçları L1 talimat önbelleğinde depolar.
- **Sıra dışı yürütme mantığı:** Veri bağımlılıklarına ve kaynak kullanılabilirliğine bağlı olarak mikro işlemlerin yürütülmesini planlar; bu nedenle, mikro işlemler komut akışından alındıklarından farklı bir sırada yürütülmek üzere planlanabilir. Zaman elverdiğinde, bu birim gelecekte gerekli olabilecek mikro işlemlerin spekülatif yürütülmesini programlar.



Şekil 4.18 Pentium 4 Blok Diyagramı

**Tablo 4.5** Pentium 4 Önbellek Çalışma Modları

Kontrol Bitleri		Çalışma Modu		
CD	NW	Önbellek Dolguları	İçinden Yazın	Geçersiz kılар
0	0	Etkin	Etkin	Etkin
1	0	Engelli	Etkin	Etkin
1	1	Engelli	Engelli	Engelli

Not: CD= 0; NW= 1 geçersiz bir kombinasyondur.

- Yürütme birimleri:** Bu birimler mikro işlemleri yürütür, gerekli verileri L1 veri önbelleğinden alır ve sonuçları geçici olarak kayıtlarda saklar.
- Bellek alt sistemi:** Bu birim, L2 ve L3 önbellekleri ile L1 ve L2 önbellekleri önbellek kaçırıldığında ana belleğe erişmek ve sistem G/C kaynaklarına erişmek için kullanılan sistem içerir.

Onceki tüm Pentium modellerinde ve diğer işlemcilerin çoğunda kullanılan organizasyondan farklı olarak Pentium 4 komut önbelleği, komut kod çözme mantığı ile yürütme çekirdeği arasında yer alır. Bu tasarım kararının arkasındaki mantık aşağıdaki gibidir: Bölüm 16'da daha ayrıntılı olarak tartışıldığı gibi, Pentium işlemi Pentium makine talimatlarını mikro-islemler adı verilen basit RISC benzeri talimatlara çözer veya çevirir. Basit, sabit uzunluklu mikro işlemlerin kullanılması, performansı artıran süper skaler ardışık sıralama ve çizelgeleme tekniklerinin kullanılmasını sağlar. Ancak, Pentium makine talimatlarının kodunu çözmem zahmetlidir; değişken sayıda bayt ve birçok farklı seçeneğe sahiptirler. Bu kod çözme işlemi zamanlama ve boru hattı mantığından bağımsız olarak yapılrsa performansın arttığı ortaya çıkmıştır. Bu konuya Bölüm 16'da geri döneceğiz.

Veri önbelleği bir geri yazma politikası kullanır: Veriler ana belleğe yalnızca önbellekten çıkarıldıklarında ve bir güncelleme yapıldığında yazılır. Pentium 4 işlemci, yazma yoluyla önbelleğe almayı desteklemek için dinamik olarak yapılandırılabilir.

L1 veri önbelleği, CD (önbellek devre dışı) ve NW (yazılamaz) bitleri olarak adlandırılan kontrol kayıtlarından birindeki iki bit tarafından kontrol edilir (Tablo 4.5). Veri önbelleğini kontrol etmek için kullanılabilen iki Pentium 4 komutu da vardır: INVD dahili önbelleği geçersiz kılar (temizler) ve harici önbelleğe (varsayımsa) geçersiz kılmaya sinyal verir. WBINVD dahili önbelleği geri yazar ve geçersiz kılar ve ardından harici önbelleği geri yazar ve geçersiz kılar.

Hem L2 hem de L3 önbellekleri 128 baytlık bir satır boyutuna sahip sekiz yolu set-associative'dır.

## 4.5 ANAHTAR TERİMLER, TEKRAR SORULARI VE PROBLEMLER

### Anahtar Terimler

erişim süresi ilişkisel eşleme önbellek vuruşu	önbellek satırı önbellek bellek önbellek kaçırma	önbellek seti veri önbelleği doğrudan erişim
--	--	--

doğrudan eşleme yüksek performanslı hesaplama (HPC) vurmak isabet oranı talimat önbelleği L1 önbellek L2 önbellek L3 önbellek hat yerellik	mantıksal önbellek bellek hiyerarşisi Bayan çok düzeyli önbellek FİZİKSEL ADRES fiziksel önbellek rastgele erişim değiştirme algoritması ikincil bellek sıralı erişim set-associative eşleme	mekansal yerellik bölmüş önbellek etiket zamansal yerellik birleşik önbellek sanal adres sanal önbellek geri yaz aracılığıyla yaz
--	--	---

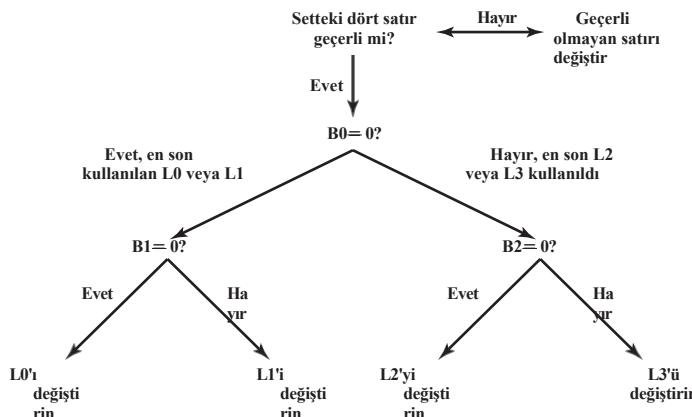
## İnceleme Soruları

- 4.1** Sıralı erişim, doğrudan erişim ve rastgele erişim arasındaki farklar nelerdir?
- 4.2** Erişim süresi, bellek maliyeti ve kapasite arasındaki genel ilişki nedir?
- 4.3** Yerellik ilkesi çoklu bellek seviyelerinin kullanımıyla nasıl ilişkilidir?
- 4.4** Doğrudan eşleme, ilişkisel eşleme ve kümeli-ilişkisel eşleme arasındaki farklar nelerdir?
- 4.5** Doğrudan eşlenen bir önbellek için, bir ana bellek adresi üç oluşuyormuş gibi görülür. Üç alanı listeleyin ve tanımlayın.
- 4.6** Bir ilişkisel önbellek için, bir ana bellek adresi iki alandan oluşuyormuş gibi görülür. İki alanı listeleyin ve tanımlayın.
- 4.7** Bir set-associative önbellek için, bir ana bellek adresi üç alandan oluşuyor olarak görülür. Üç alanı listeleyin ve tanımlayın.
- 4.8** Mekansal yerellik ile zamansal yerellik arasındaki fark nedir?
- 4.9** Genel olarak, mekansal yerellik ve zamansal yerellikten yararlanma stratejileri nelerdir?

## Problemler

- 4.1** Set-associative önbellek, dört satırlıksettlerde bölmüş 64 satır veya yuvalardan oluşur. Ana bellek, her biri 128 kelimealtı 4K blok içerir. Ana bellek adreslerinin biçimini gösterin.
- 4.2** İki yönlü set-associative önbellek 16 baytlık satırlara ve toplam 8 kB boyuta sahiptir. 64 MB ana bellek bayt adreslenebilir. Ana bellek adreslerinin biçimini gösterin.
- 4.3** Onaltılık ana bellek adresleri 111111, 666666, BBBB BBBB için aşağıdaki bilgileri onaltılık biçimde gösterin:
  - a.** Şekil 4.10'daki formатı kullanarak doğrudan eşlenen bir önbellek için Etiket, Satır ve Sözcük değerleri
  - b.** Şekil 4.12'deki formатı kullanarak bir ilişkisel önbellek için Etiket ve Kelime değerleri
  - c.** Şekil 4.15'teki formатı kullanarak iki yönlü bir set-ilişkili önbellek için Etiket, Set ve Kelime değerleri
- 4.4** Aşağıdakileri listeleyin:
  - a.** Şekil 4.10'daki doğrudan önbellek örneği için: adres uzunluğu, adreslenebilir birim sayısı, blok boyutu, ana bellekteki blok sayısı, önbellekteki satır sayısı, etiket boyutu
  - b.** Şekil 4.12'deki ilişkisel önbellek örneği için: adres uzunluğu, adreslenebilir birim sayısı, blok boyutu, ana bellekteki blok sayısı, önbellekteki satır sayısı, etiket boyutu

- c.** Şekil 4.15'teki iki yönlü set-iliskili önbellek örneği için: adres uzunluğu, adreslenebilir birim , blok boyutu, ana bellekteki blok sayısı, setteki satır sayısı, set sayısı, önbellekteki satır sayısı, etiket boyutu
- 4.5** Yonga üzerinde 16-kB dört-yönlü set-associative önbelleğe sahip 32-bit bir mikroişlemci düşünün. Önbelleğin dört adet 32 bitlik sözcükten oluşan bir satır boyutuna sahip olduğunu varsayın. Bu önbelleğin organizasyonunu ve farklı adres alanlarının bir önbellek isabetini/hatasını belirlemek için nasıl kullanıldığını gösteren bir blok diyagram çizin. ABCDE8F8 bellek konumundan gelen sözcük önbellekte nereye eslenir?
- 4.6** Harici bir önbellek için aşağıdaki özellikler verilmiştir: dört yolu set associative; iki 16-bit kelime satır boyutu; ana bellekten toplam 4K 32-bit kelime barındırılabilir; 24-bit adresler veren 16-bit bir işlemci ile kullanılır. Önbellek yapısını tüm ilgili bilgilerle birlikte tasarlın ve işlemcinin adreslerini nasıl yorumladığını gösterin.
- 4.7** Intel 80486 çip üzerinde birleşik bir önbelleğe sahiptir. Bu önbellek 8 kB içerir ve dört yönlü set-associative organizasyona ve dört adet 32-bit kelimeblok uzunluğuna sahiptir. Önbellek 128 set halinde düzenlenmiştir. Satır başına tek bir "satır geçerli biti" ve üç bit, B0, B1 ve B2 ("LRU" bitleri) vardır. Bir önbellek özleminde, 80486 ana bellekten 16 baytlık bir satırı bir veri yolu belleğe okuma patlamasında okur. Önbelleğin basitleştirilmiş bir diyagramını çizin ve adresin farklı alanlarının nasıl yorumlandığını gösterin.
- 4.8** Bayt adreslenebilir ana belleği  $2^{16}$  bayt ve blok boyutu 8 bayt olan bir makine düşünün. Bu makinede 32 satırдан oluşan doğrudan eşlenmiş bir önbellek kullanıldığını varsayıyalım.
- 16 bitlik bir bellek adresi etiket, satır numarası ve bayt nasıl bölünür?
  - Aşağıdaki adreslerin her birine sahip baytlar hangi satırda saklanır?
- |      |      |      |      |
|------|------|------|------|
| 0001 | 0001 | 0001 | 1011 |
| 1100 | 0011 | 0011 | 0100 |
| 1101 | 0000 | 0001 | 1101 |
| 1010 | 1010 | 1010 | 1010 |
- c.** Adresi 0001 1010 0001 1010 olan baytin önbellekte saklandığını varsayıyalım. Onunla birlikte saklanan diğer baytların adresleri nedir?
- d.** Önbellekte toplam kaç bayt bellek depolanabilir?
- e.** Etiket neden önbellekte de saklanıyor?
- 4.9** Intel 80486, çip üstü önbelleği için **sözde en son kullanılan** olarak adlandırılan bir değiştirme algoritması kullanır. Dört satırda oluşan 128 kümenin (L0, L1, L2, L3 olarak etiketlenmiş) her biri ile ilişkili üç B0, B1 ve B2 biti vardır. Değiştirme algoritması aşağıdaki gibi çalışır: Bir satırın değiştirilmesi gerektiğinde, önbellek önce en son kullanımın L0 ve L1'den mi yoksa L2 ve L3'ten mi olduğunu belirleyecektir. Ardından önbellek, blok çiftlerinden hangisinin en son kullanıldığıni belirleyecek ve değiştirilmek üzere işaretleyecektir. Şekil 4.19 mantığı göstermektedir.
- B0, B1 ve B2 bitlerinin nasıl ayarlandığını belirtin ve ardından Şekil 4.19'da gösterilen değiştirme algoritmasında nasıl kullanıldıklarını kelimelerle açıklayın.
  - 80486 algoritmasının gerçek bir LRU algoritmasına yaklaştığını gösterin. *İpucu:* En son kullanım sırasının L0, L2, L3, L1 olduğu durumu göz önünde bulundurun.
  - Gerçek bir LRU algoritmasının set başına 6 bit gerektireceğini gösterin.
- 4.10** Küme ilişkilendirmeli bir önbellek, dört adet 16 bitlik sözcükten oluşan bir blok boyutuna ve 2'lik bir küme boyutuna sahiptir. Önbellek toplam 4096 sözcük barındırılabilir. Önbelleğe alınabilen ana bellek boyutu  $64K * 32$  bittir. Önbellek yapısını tasarlın ve işlemci adreslerinin nasıl yorumlandığını gösterin.
- 4.11** Bayt düzeyinde adresleme yapmak için 32 bit adres kullanan bir bellek sistemi ve 64 bayt satır boyutu kullanan bir önbellek düşünün.
- Adresinde 20 bitlik bir etiket alanı olan doğrudan eşlenmiş bir önbellek varsayıyalım. Adres formatını ve aşağıdaki parametreleri belirleyin: adreslenebilir birim sayısı, ana bellekteki blok sayısı, önbellekteki satır sayısı, etiket boyutu.



Şekil 4.19 Intel 80486 Yonga Üstü Önbellek Değiştirme Stratejisi

- b.** İlişkisel bir önbellek varsayıınız. Adres formatını gösterin ve aşağıdaki parametreleri belirleyin: adreslenebilir birim sayısı, ana bellekteki blok sayısı, önbellekteki satır sayısı, etiket boyutu.
  - c.** Adresinde 9 bitlik bir etiket alanı olan dört-yolu set-iliskili bir önbellek varsayıınız. Adres formatını gösterin ve aşağıdaki parametreleri belirleyin: adreslenebilir birim sayısı, ana bellekteki blok sayısı, setteki satır sayısı, önbellekteki set sayısı, önbellekteki satır sayısı, etiket boyutu.
- 4.12** Aşağıdaki özelliklere sahip bir bilgisayar düşünün: toplam 1 MB ana bellek; 1 bayt kelime boyutu; 16 bayt blok boyutu; ve 64 kB önbellek boyutu.
- a.** F0010, 01234 ve CABBE ana bellek adresleri için, doğrudan eşlenen bir önbellek için karşılık gelen etiketi, önbellek satır adresini ve sözcük uzaklıklarını verin.
  - b.** Doğrudan eşlenen bir önbellek için aynı önbellek yuvasına eşlenen farklı etiketlere sahip herhangi iki ana bellek adresi verin.
  - c.** F0010 ve CABBE ana bellek adresleri için, tam ilişkisel bir önbellek için karşılık gelen etiket ve ofset değerlerini verin.
  - d.** F0010 ve CABBE ana bellek adresleri için, iki yönlü set-associative önbellek için karşılık gelen etiket, önbellek seti ve ofset değerlerini verin.
- 4.13** Dört yolu küme ilişkilendirmeli bir önbelekte LRU değiştirme algoritması uygulamak için basit bir teknik tanımlayın.
- 4.14** Örnek 4.3'ü tekrar düşünün. Ana bellek, ilk kelimeye erişim süresi 30 ns ve sonraki her kelime için erişim süresi 5 ns olan bir blok aktarım özelliği kullanırsa cevap nasıl değişir?
- 4.15** Aşağıdaki kodu göz önünde bulundurun:
- ```

İçin (i= 0; i <= 20; i++)
  for (j= 0; j <= 10; j++)
    a[i]= a[i]*j
  
```
- a.** Koddaki uzamsal yerelliğe bir örnek veriniz.
  - b.** Koddaki zamansal yerelliğe bir örnek veriniz.
- 4.16** Ek 4A'daki Denklem (4.2) ve (4.3)'ü N-seviyeli bellek hiyerarşilerine genelleştirin.
- 4.17** Bir bilgisayar sistemi 32K 16-bit sözcükten oluşan bir ana bellek içerir. Ayrıca, her satırda 64 sözcük bulunan dört satırlık kümelere bölünmüş 4K sözcüklük bir önbellege sahiptir. Önbellegen başlangıçta boş olduğunu varsayılmı. İşlemci kelimeleri 0, 1, 2, ..., konumlarından alır., 4351 konumlarından

siparişi verir. Daha sonra bu getirme sırasını dokuz kez daha tekrarlar. Önbellek ana bellekten 10 kat daha hızlıdır. Önbellek kullanımından kaynaklanan iyileşmeyi tahmin edin. Blok değiştirme için bir LRU politikası varsayılmı.

- 4.18** Her biri 16 baytlık 4 satırda oluşan bir önbellek düşünün. Ana bellek her biri 16 baytlık bloklara bölünmüştür. Yani, 0. blokta 0'dan 15'e kadar adresleri olan baytlar bulunur ve bu böyle devam eder. Şimdi belleğe aşağıdaki adres sırasına göre erişim bir program düşünün: Bir kez: 63'ten 70'e kadar. On kez döngü: 15'ten 32'ye; 80'den 95'e.
- Önbellegin doğrudan eşlenmiş olarak düzenlendığını varsayılm. Bellek blokları 0, 4, vb. satır 1'e; bloklar 1, 5, vb. satır 2'ye atanır; ve bu böyle devam eder. Isabet oranını hesaplayın.
  - Önbellegin, her biri iki satırda oluşan iki set ile iki yönlü set ilişkisel olarak düzenlendığını varsayılm. Çift numaralı bloklar küme 0'a ve tek numaralı bloklar küme 1'e atanır. En son kullanılan değiştirme şemasını kullanarak iki yönlü set ilişkilendirmeli önbellek için isabet oranını hesaplayın.
- 4.19** Aşağıdaki parametrelerle sahip bir bellek sistemi düşünün:
- |                         |                                |
|-------------------------|--------------------------------|
| $T_c = 100 \text{ ns}$  | $C_c = 10^{-4} \text{ \$/bit}$ |
| $T_m = 1200 \text{ ns}$ | $C_m = 10^{-5} \text{ \$/bit}$ |
- 1 MB ana belleğin maliyeti nedir?
  - Ön bellek teknolojisini kullanan 1 MB ana belleğin maliyeti nedir?
  - Etkin erişim süresi önbellek erişim süresinden %10 daha fazlaysa, isabet oranı  $H$  nedir?
- 4.20** Erişim süresi 1 ns ve isabet oranı  $H = 0.95$  olan bir L1 önbellegi düşünün. Önbellek tasarnımını (önbellek boyutu, önbellek organizasyonu)  $H$ 'yi 0,97'ye çıkaracak, ancak erişim süresini 1,5 ns'ye çıkaracak şekilde değiştirebileceğimizi varsayılm. Bu değişikliğin performans artışı ile sonuçlanması için hangi koşullar yerine getirilmelidir?
- Bu sonucun neden sezgisel olarak mantıklı olduğunu açıklayın.
- 4.21** Erişim süresi 2,5 ns, satır boyutu 64 bayt ve isabet oranı  $H = 0.95$  olan tek seviyeli önbellek düşünün. Ana bellek, 50 ns'lik ilk kelime (4 bayt) erişim süresine ve sonraki her kelime için 5 ns'lik erişim süresine sahip bir blok aktarım özelliği kullanır.
- Bir önbellek iskasi olduğunda erişim süresi nedir? Önbellegin, satır ana bellekten alınana kadar beklediğini ve ardından bir isabet için yeniden yürütüldüğünü varsayıın.
  - Satır boyutunu 128 bayta çarpanın  $H$ 'yi 0,97'ye yükselttiğini varsayılm. Bu ortalama bellek erişim süresini azaltır mı?
- 4.22** Bir bilgisayarın önbellegi, ana belleği ve sanal bellek için kullanılan bir diski vardır. Eğer başvurulan kelime önbellette ise, ona erişmek için 20 ns gereklidir. Eğer kelime ana bellekte ama önbellette değilse, önbellege yüklemek için 60 ns gereklidir ve sonra referans tekrar başlatılır. Kelime ana bellekte değilse, kelimeyi diskten almak için 12 ms, ardından önbellege kopyalamak için 60 ns gereklidir ve ardından referans yeniden başlatılır. Önbellek isabet oranı 0,9 ve ana bellek isabet oranı 0,6'dır. Bu sisteme referans verilen bir kelimeye erişmek için gereken ortalama süre nanosaniye cinsinden nedir?
- 4.23** Satır boyutu 64 bayt olan bir önbellek düşünün. Önbellekteki satırların ortalama %30'unun kirli olduğunu varsayılm. Bir kelime 8 bayttan oluşmaktadır.
- 3 iskalama oranı (0,97 isabet oranı) olduğunu varsayıın. Ana bellek trafiğinin miktarını, hem yazma hem de geri yazma politikaları için komut başına bayt cinsinden hesaplayın. Bellek önbellege her seferinde bir satır okunur. Ancak, geri yazma için önbellekten ana belleğe tek bir kelime yazılabilir.
  - 5'lük bir oran için a bölümünü tekrarlayın.
  - 7'lük oran için a bölümünü tekrarlayın.
  - Bu nasıl bir sonuç çıkarıbilirsiniz?
- 4.24** Motorola 68020 mikroişlemcisinde bir önbellek erişimi iki saat döngüsü sürer. Ana bellekten veri yolu üzerinden işlemciye veri erişimi, ana bellekte üç saat döngüsü sürer.

Bekleme durumu ekleme olmaması durumunda; veriler önbelleğe teslim ile paralel olarak işlemciye teslim edilir.

- Vuruş oranı 0,9 ve saat hızı 16,67 MHz olan bir bellek döngüsünün etkin uzunluğunu hesaplayın.
- Bellek çevrimi başına birer çevrimlik iki bekleme durumu eklendiğini varsayararak hesaplamaları tekrarlayın. Sonuçlardan nasıl bir sonuç çıkarılabilirsiniz?

**4.25** Bellek çevrim süresi 300 ns ve komut işleme hızı 1 MIPS olan bir işlemci varsayılmış. Ortalama olarak, her komut, komutun alınması için bir ve içeriği işlenen için bir veri yolu bellek döngüsü gerektirir.

- İşlemci tarafından veri yolunun kullanımını hesaplayın.
- İşlemcinin bir komut önbelleği ile donatıldığını ve ilgili isabet oranının 0,5 olduğunu varsayılmış. Veri yolu kullanımını üzerindeki etkiye belirleyin.

**4.26** Bir okuma işlemi için tek seviyeli bir önbellek sisteminin performansı aşağıdaki denklemle karakterize edilebilir:

$$T_a = T_c + (1 - H)T_m$$

Burada  $T_a$  ortalama erişim süresi,  $T_c$  önbellek erişim süresi,  $T_m$  bellek erişim süresi (bellekten işlemci kaydına) ve  $H$  isabet oranıdır. Basitlik için, söz konusu kelimenin önbelleğe işlemci yazmacına paralel olarak yüklediğini varsayıyoruz. Bu, Denklem (4.2) ile aynı formdadır.

- $T_b$ = önbellek ve ana bellek arasında bir satır aktarma süresi ve  $W$ = yazma referanslarının kesri olarak tanımlayın. Bir yazma politikası kullanarak okumaların yanı sıra yazmaları da hesaba katmak için önceki denklemi revize edin.
- $W_b$ 'yi önbellekteki bir satırın değiştirilme olasılığı olarak tanımlayın. Geri yazma politikası için  $T_a$  denklemini sağlayınız.

**4.27** İki seviyeli **önbelleğe** sahip bir sistem için,  $T_{(c)_1}$ = birinci önbellek erişim süresi;  $T_{(c)_2}$ = ikinci seviye önbellek erişim süresi;  $T_{(m)}$ = bellek erişim süresi;  $H_{(1)}$ = birinci seviye önbellek isabet oranı;  $H_{(2)}$ = birleşik birinci/ikinci seviye önbellek isabet tanımlayınız. Bir okuma işlemi  $T_a$  denklemini sağlayınız.

**4.28** Bir önbellek okuma özleminde aşağıdaki performans özelliklerini varsayılmış: ana belleğe bir adres göndermek için bir saat döngüsü ve ana bellekten 32 bitlik bir kelimeye erişmek ve bunu işlemciye ve önbelleğe aktarmak için dört saat döngüsü.

- Önbellek satırı boyutu bir kelime ise, ıskalama cezası (yani, bir okuma ıskalama durumunda okuma için gereken ek süre) nedir?
- Önbellek satırı boyutu dört kelimeyse ve çoklu, olmayan bir aktarım gerçekleştirilirse ıskalama cezası nedir?
- Önbellek satırı boyutu dört sözcükse ve sözcük aktarımı başına bir saat döngüsü ile bir aktarım gerçekleştirilirse ıskalama cezası nedir?

**4.29** Önceki problemde önbellek tasarımlı için, satır boyutunun bir kelimededen dört kelimeye çıkarılmasının okuma kaçırma oranının %3,2'den %1,1'e düşmesine neden olduğunu varsayılmış. Hem patlamsız aktarım hem de patlamlı aktarım durum için, iki farklı satır boyutu için tüm okumaların ortalaması alındığında ortalama ıskalama cezası nedir?

## EK 4A

### PERFORMANS ÖZELLİKLERİ IKI SEVİYELİ BELLEKLERİN

Bu bölümde, ana bellek ile işlemci arasında tampon görevi gören ve iki seviyeli bir dahili bellek oluşturan bir atıfta bulunulmaktadır. Bu iki seviyeli mimari yerellik olarak bilinen bir özellikten yararlanarak tek seviyeli belleklere göre daha iyi performans sağlamaktadır.

Ana bellek önbellek mekanizması bilgisayar mimarisinin bir parçasıdır, donanımda uygulanır ve genellikle işletim sistemi tarafından görülmez. Yerelliği de kullanan ve en azından kısmen işletim sisteminde uygulanan iki seviyeli bellek yaklaşımının iki örneği daha vardır: sanal bellek ve disk önbelleği (Tablo 4.6). Sanal bellek Bölüm 8'de incelenmiştir; disk önbelleği bu kitabın kapsamı dışındadır ancak [STAL15]'de incelenmiştir. Bu ekte, iki seviyeli belleklerin her üç yaklaşım için de ortak olan bazı performans özelliklerine bakacağız.

### Yerellik

İki seviyeli belleğin performans avantajının temeli, **referansın yerelliği olarak** bilinen bir ilkedir [DENN68]. Bu ilke, bellek referanslarının kümelenme eğiliminde olduğunu belirtir. Uzun süre boyunca, kullanıldığı kümeler değişir, ancak kısa bir süre boyunca, işlemci öncelikle sabit bellek referans kümeleriyle çalışır.

Sezgisel olarak, yerellik ilkesi mantıklıdır. Aşağıdaki mantık silsilesini göz önünde bulundurun:

1. Tüm program talimatlarının yalnızca küçük bir oluşturan dallanma ve çağrı talimatları dışında, program yürütme sıralıdır. Bu nedenle, çoğu durumda, getirilecek bir sonraki komut, getirilen son komutu hemen takip eder.
2. Uzun ve kesintisiz bir yordam çağrıları dizisi ve ardından gelen geri dönüşler dizisi nadiren görülür. Bunun yerine, bir program oldukça dar bir yordam çağrıma derinliği penceresiyle sınırlı kalır. Bu nedenle, kısa bir süre içinde talimatlara yapılan referanslar birkaç prosedürle sınırlı kalma eğilimindedir.
3. Çoklu yinelemeli yapı, birçok kez tekrarlanan nispeten az sayıda talimattan oluşur. Yineleme süresi boyunca, hesaplama bu nedenle programın küçük bir bitişik bölümüyle sınırlıdır.
4. Birçok programda, hesaplamanın büyük bir kısmı diziler veya kayıt dizileri gibi veri yapılarının işlenmesini içerir. Çoklu durumda, bu veri yapılarına yapılan arduşık atıflar yakın konumdaki veri öğelerine olacaktır.

**Tablo 4.6** İki Seviyeli Belleklerin Özellikleri

|                                | Ana Bellek<br>Önbelleği            | Sanal Bellek (disk belleği)             | Disk Önbelleği                             |
|--------------------------------|------------------------------------|-----------------------------------------|--------------------------------------------|
| Tipik erişim süresi            | 5:1 (ana bellek vs.                | 10 <sup>6</sup> :1 (ana bellek vs.      | 10 <sup>6</sup> :1 (ana bellek vs.         |
| Oranlar                        | önbellek)                          | disk)                                   | disk)                                      |
| Bellek yönetimi                | Tarafından uygulandı               | Donanım kombinasyonu                    | Sistem yazılımı                            |
| Sistem                         | özel donanım                       | ve sistem yazılımı                      |                                            |
| Tipik blok veya sayfa boyut    | 4 ila 128 bayt<br>(önbellek bloğu) | 64 ila 4096 bayt (sanal bellek sayfası) | 64 ila 4096 bayt (disk blok veya sayfalar) |
| İşlemciye erişim ikinci seviye | Doğrudan erişim                    | Dolaylı erişim                          | Dolaylı erişim                             |

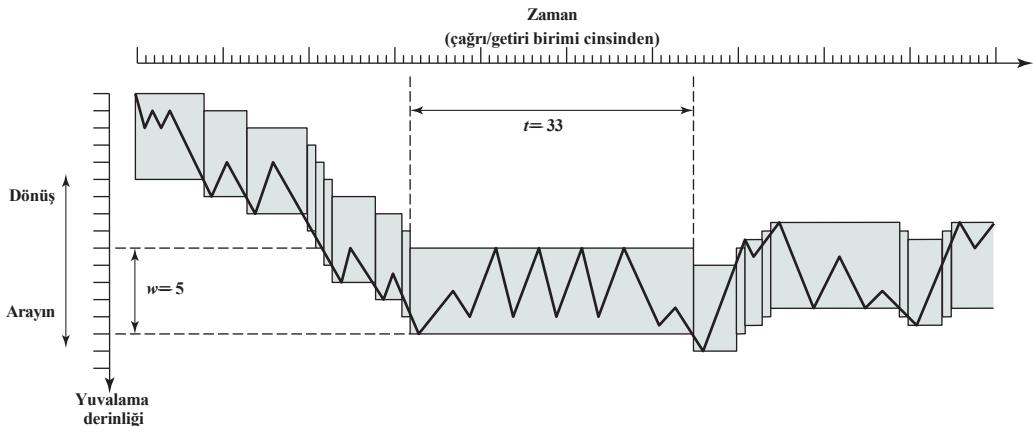
**Tablo 4.7** Üst Düzey Dil İşlemlerinin Görelî Dinamik Sıklığı

| Çalışma Dili<br>İş Yükü | [HUCK83]<br>Pascal<br>Scientific | [KNUT71]<br>FORTRAN<br>Öğrenci | [PATT82a] |    | [TANE78] SAL<br>Sistem |
|-------------------------|----------------------------------|--------------------------------|-----------|----|------------------------|
|                         | Pascal<br>Sistem                 | C<br>Sistem                    |           |    |                        |
| Atama                   | 74                               | 67                             | 45        | 38 | 42                     |
| Döngü                   | 4                                | 3                              | 5         | 3  | 4                      |
| Arayın                  | 1                                | 3                              | 15        | 12 | 12                     |
| EĞER                    | 20                               | 11                             | 29        | 43 | 36                     |
| GOTO                    | 2                                | 9                              | -         | 3  | -                      |
| Diğer                   | -                                | 7                              | 6         | 1  | 6                      |

Bu mantık silsilesi birçok çalışmada doğrulanmıştır. Birinci maddeye istinaden, çeşitli çalışmalar üst düzey dil programlarının davranışını analiz etmiştir. Tablo 4.7, aşağıdaki çalışmalarlardan elde edilen, yürütme sırasında çeşitli deyim türlerinin görünümünü ölçen temel sonuçları içermektedir. Knuth [KNUT71] tarafından gerçekleştirilen en eski programlama dili davranışlı çalışması, öğrenci alıştırmaları olarak kullanılan bir FORTRAN programları koleksiyonunu incelemiştir. Tanenbaum [TANE78] işletim sistemi programlarında kullanılan ve yapılandırılmış programlamayı destekleyen bir dilde (SAL) yazılmış 300'den fazla prosedürden toplanan ölçümleri yayinallyamıştır. Paterson ve Sequinein [PATT82a] derleyicilerden ve dizgi, bilgisayar destekli tasarım (CAD), sıralama ve dosya karşılaştırma programlarından alınan bir dizi ölçümü analiz etmiştir. C ve Pascal programlama dilleri üzerinde çalışılmıştır. Huck [HUCK83], hızlı Fourier dönüşümü ve farklı denklem sistemlerinin entegrasyonu da dahil olmak üzere genel amaçlı bilimsel hesaplamanın bir karışımını temsil etmesi amaçlanan dört programı analiz etmiştir. Bu dil ve uygulama karışımının sonuçlarında, dallanma ve çağrı talimatlarının bir programın ömrü boyunca yürütülen ifadelerin yalnızca bir kısmını temsil ettiği konusunda iyi bir uyum vardır. Dolayısıyla, bu çalışmalar 1. iddiayı doğrulamaktadır.

Iddia 2 ile ilgili olarak, [PATT85a]'da bildirilen çalışmaları doğrulama sağlamaktadır. Bu, çağrı-dönüş davranışını gösteren Şekil 4.20'de gösterilmektedir. Her çağrı aşağı ve sağa doğru hareket eden çizgi ile her dönüş yukarı ve sağa doğru hareket eden çizgi ile temsil edilmektedir. Şekilde, derinliği 5'e eşit olan bir pencere tanımlanmıştır. Yalnızca her iki yönde net hareketi 6 olan bir çağrı ve geri dönüş dizisi pencerenin hareket etmesine neden olur. Görülebileceği gibi, çalışan program uzun süreler boyunca sabit bir pencere içinde kalabilir. Aynı analistler tarafından C ve Pascal programları üzerinde yapılan bir çalışma, 8 derinlikli bir pencerenin çağrıların veya geri dönüşlerin yalnızca %1'inden daha azında kayması gerektiğini göstermiştir [TAMI83].

Literatürde uzamsal yerellik ve zamansal yerellik arasında bir ayırım yapılmaktadır. **Uzamsal yerellik**, kümelenmiş bir dizi bellek konumunu işleme eğilimini ifade eder. Bu, bir işlemcinin talimatlara sırayla erişme eğilimini yansıtır. Uzamsal konum aynı zamanda bir veri tablosunu işlerken olduğu gibi, bir programın veri konumlarına sırayla erişme eğilimini de yansıtır. **Zamansal yerellik**, bir işlemcinin yakın zamanda kullanılmış olan bellek konumlarına erişme eğilimini ifade eder. Örneğin, bir yineleme döngüsü yürütüldüğünde, işlemci aynı talimat setini tekrar yürütür.



Şekil 4.20 Bir Programın Örnek Çağrı-Dönüş Davranışı

Geleneksel olarak zamansal yerellik, son kullanılan komut ve veri değerlerinin ön bellekte tutulması ve bir ön bellek hiyerarşisinden yararlanması yoluyla kullanılır. Uzamsal yerellik genellikle daha büyük önbellek blokları kullanılarak ve önbellek kontrol mantığına ön-getirme mekanizmaları (beklenen kullanım öğelerini getirme) dahil edilerek kullanılır. Son zamanlarda, daha yüksek performans elde etmek için bu tekniklerin iyileştirilmesi konusunda önemli araştırmalar yapılmıştır, ancak temel stratejiler aynı kalmıştır.

### **İki Seviyeli Belleğin Çalışması**

Yerellik özelliğinden iki seviyeli bir bellek oluşturulurken faydalansılabilir. Üst seviye bellek (M1) alt seviye bellekler M2) daha küçük, daha hızlı ve daha pahalıdır (bit başına). M1, daha büyük olan M2'nin içeriğinin bir kısmı için geçici bir depo olarak kullanılır. Bir bellek referansı yapıldığında, M1'deki öğeye erişilmeye çalışılır. Bu başarılı olursa, hızlı bir erişim yapılır. Başarılı olmazsa, bir bellek konumu bloğu M2'den M1'e kopyalanır ve erişim M1 üzerinden gerçekleşir. Yerellik nedeniyle, bir blok M1'e getirildiğinde, bu konumlara bir dizi erişim olmalı ve bu da hızlı genel hizmetle sonuçlanmalıdır.

Bir öğeye erişmek için ortalama süreyi ifade etmek için, sadece iki bellek seviyesinin hızlarını değil, aynı zamanda belirli bir referansın M1'de bulunma olasılığını da dikkate almalıyız. Elimizde

$$\begin{aligned} T_s &= H * T_{(1)} + (1 - H) * (T_1 + T_2) \\ &= T_1 + (1 - H) * T_2 \end{aligned} \tag{4.2}$$

nerede

$T_s$ = ortalama (sistem) erişim süresi

$T_1$ = M1 erişim süresi (örn. önbellek, disk önbelleği)  $T_2$ = M2 erişim süresi (örn. ana bellek, disk)

$H$ = isabet oranı (referansın M1'de bulunduğu zamanın oranı)

Şekil 4.2 isabet oranının bir fonksiyonu olarak ortalama erişim süresini göstermektedir. Görülebileceği gibi, yüksek isabet yüzdesi için ortalama toplam erişim süresi M1'e M2'den çok daha yakındır.

## Performans

İki seviyeli bir mem- ory mekanizmasının değerlendirilmesiyle ilgili bazı parametrelere bakalım. İlk olarak maliyeti düşünün. Sahip olduğumuz

$$= \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2} \quad (4.3)$$

nerede

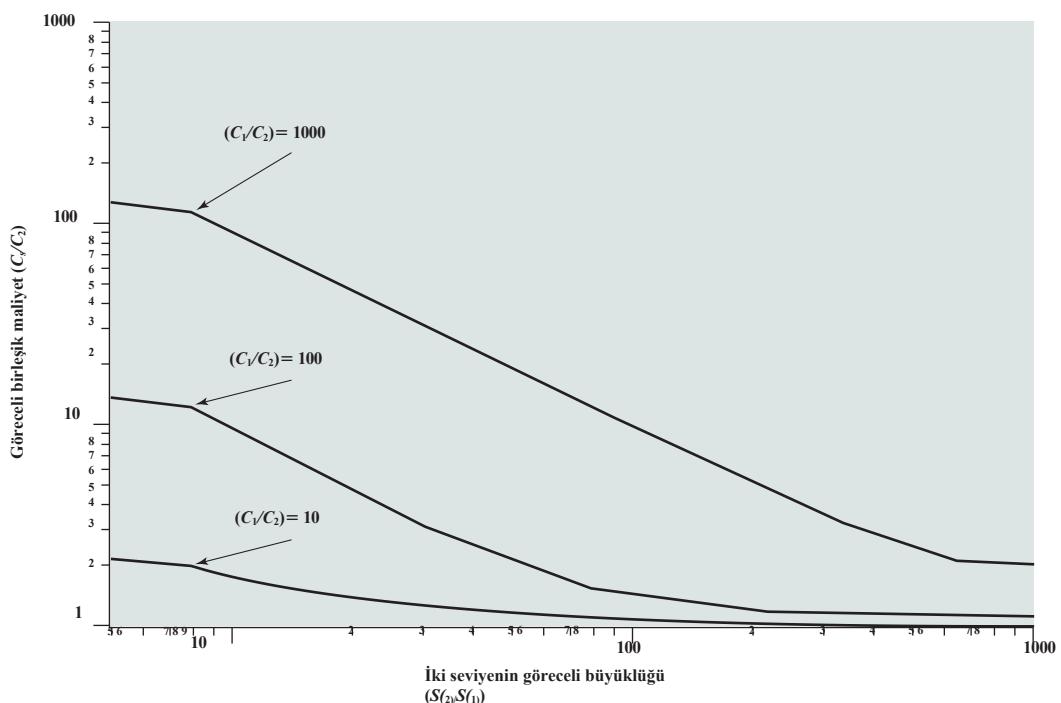
$C_s$ = birleştirilmiş iki@seviyeli bellek için bit başına ortalama maliyet

$C_1$ = üst@seviye bellek M1'in bit başına ortalama maliyeti  $C_2$ =

alt@seviye bellek M2'nin bit başına ortalama maliyeti  $S_1$ = M1'in boyutu

$S_2$ = M2'nin boyutu

$C_s \approx C_{(2)}$  olmasını istiyoruz.  $C_1 \gg C_2$  olduğuna göre, bu  $S_1 \gg S_2$  gerektirir. Şekil 4.21 ilişkisi göstermektedir.



Şekil 4.21 İki Seviyeli Bellek için Ortalama Bellek Maliyetinin Göreli Bellek Boyutu ile İlişkisi

Daha sonra, erişim süresini ele alalım. İki seviyeli bir belleğin önemli bir performans artışı sağlamaası için  $T_{(s)}$  değerinin yaklaşık olarak  $T_{(1)}$  değerine eşit olması gereklidir ( $T_{(s)} \approx T_{(1)}$ ).  $T_1$ 'in  $T_{(2)}/T_{(1)}$  66  $T_2$ 'den çok daha az olduğu göz önüne alındığında, 1'e yakın bir isabet oranına ihtiyaç vardır.

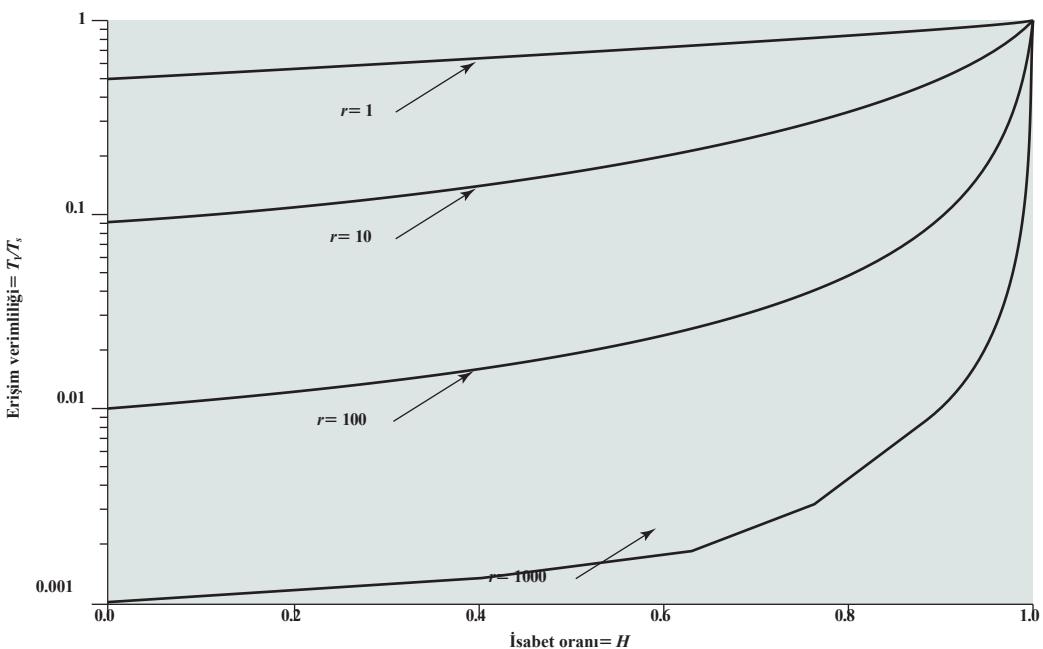
Dolayısıyla M1'in maliyeti düşürmek için küçük, isabet oranını ve dolayısıyla performansı artırmak için büyük olmasını istiyoruz. Her iki gereksinimi de makul ölçüde karşılayan bir M1 boyutu var mıdır? Bu soruyu bir dizi alt soruya yanıtlayabiliriz:

- $T_s \approx T_1$  olması için hangi isabet oranı değeri gereklidir?
- Hangi büyülükteki M1 gerekli isabet oranını sağlayacaktır?
- Bu boyut maliyet gerekliliğini karşılıyor mu?

Buna ulaşmak için, *erişim verimliliği* olarak adlandırılan  $T_1/T_s$  miktarını göz önünde bulundurun. Ortalama erişim süresinin ( $T_s$ ) M1 erişim süresine ( $T_1$ ) ne kadar yakın olduğunun bir ölçüsüdür. Denklem (4.2)'den,

$$\frac{T_1}{T_s} = \frac{1}{1 + (1 - H) \frac{T_2}{T_1}} \quad (4.4)$$

Şekil 4.22, isabet oranı  $H$ 'nin fonksiyonu olarak  $T_1/T_s$ 'yi, parametre olarak  $T_2/T_1$  miktarıyla birlikte çizmektedir. Tipik olarak, çip içi önbellek erişim süresi ana bellek erişim yaklaşık 25 ila 50 kat daha hızlıdır (yani,  $T_2/T_1$  25 ila 50'dir), çip dışı önbellek erişim süresi

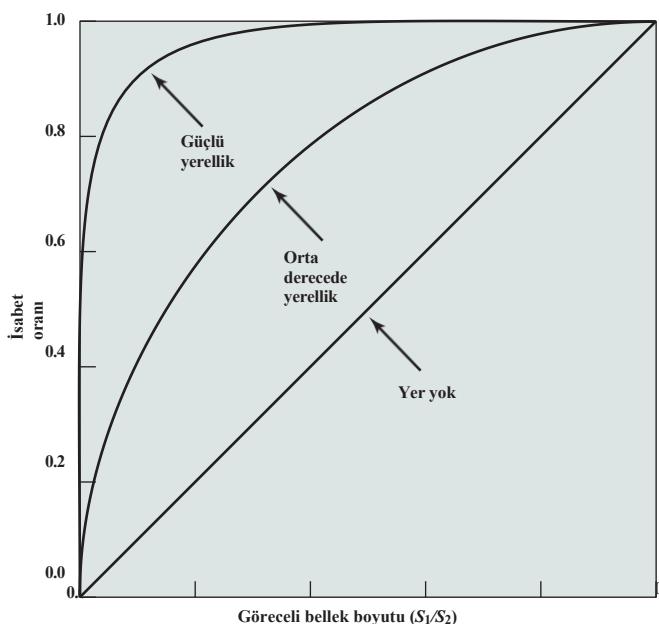


**Şekil 4.22** Isabet Oranının Bir Fonksiyonu Olarak Erişim Verimliliği ( $r = T_2/T_1$ )

ana bellek erişim süresinden yaklaşık 5 ila 15 kat daha hızlıdır (yani,  $T_2/T_1$  5 ila 15'tir) ve ana bellek erişim süresi disk erişim süresinden yaklaşık 1000 kat daha hızlıdır ( $T_2/T_1 = 1000$ ). Dolayısıyla, performans gereksinimini karşılamak için 0,9'a yakın bir isabet oranı gereklili görülmektedir.

Artık göreceli bellek boyutularındaki soruyu daha kesin bir şekilde ifade edebiliriz.  $S_1 66 S_2$  için 0,8 ya da daha iyi bir isabet oranı makul müdür? Bu, çalıştırılan yazılımları niteliği ve iki seviyeli belleğin tasarımının ayrıntıları dahil olmak üzere bir dizi faktöre bağlı olacaktır. Ana belirleyici elbette yerellik derecesidir. Şekil 4.23 yerelligin isabet oranı üzerindeki etkisini göstermektedir. Açıkça, eğer  $M_1 M_2$  ile aynı boyutta ise, isabet oranı 1.0 olacaktır:  $M_2$ 'deki tüm öğeler her zaman  $M_1$ 'de de depolanır. Şimdi yerelligin olmadığını, yani referansların tamamen rastgele olduğunu varsayıyalım. Bu durumda isabet oranı göreceli bellek boyutunun tam olarak doğrusal bir fonksiyonu olmalıdır. Örneğin,  $M_1, M_2$ 'nin yarısı büyütülgündeyse, herhangi bir zamanda  $M_2$ 'deki öğelerin yarısı  $M_1$ 'de de bulunur ve isabet oranı 0,5 olur. Ancak pratikte, referanslarda bir dereceye kadar yerellik vardır. Orta ve güçlü yerelligin etkileri şekilde gösterilmiştir. Şekil 4.23'ün belirli bir veri ya da modelden türetilmediğini unutmayın; şekil, çeşitli yerellik derecelerinde görülen performans türünü göstermektedir.

Dolayısıyla, güçlü bir yerellik varsa, nispeten küçük üst düzey bellek boyutuyla bile yüksek isabet oranı değerlerine ulaşmak mümkündür. Örneğin, çok sayıda çalışma oldukça küçük *önbellek* boyutlarının *ana belleğin* boyutundan olarak 0,75'in üzerinde bir isabet oranı sağlayacağını göstermiştir (örneğin, [AGAR89], [PRZY88], STRE83] ve [SMIT82]). 1K ila 128K kelime aralığındaki bir önbellek genellikle yeterli olurken, ana



**Şekil 4.23** Bağıl Bellek Boyutunun Bir Fonksiyonu Olarak İabet Oranı

bellek artık tipik olarak gigabayt aralığındadır. Sanal bellek ve disk önbelleğini ele aldığımızda, aynı olguyu, yani nispeten küçük bir M1'in yerellik nedeniyle yüksek bir isabet oranı değerini verdigini doğrulayan diğer çalışmalardan bahsedeceğiz.

Bu da bizi daha önce sıralanan son soruya getiriyor: İki belleğin göreceli boyutu maliyet gereksinimini karşılıyor mu? Cevap açıkça evettir. İyi bir performans elde etmek için yalnızca küçük bir üst seviye belleğe ihtiyacımız varsa, iki bellek seviyesinin bit başına ortalama maliyeti, daha ucuz olan alt seviye belleğin maliyetine yaklaşacaktır.

L2 önbellek, hatta L2 ve L3 önbellekler söz konusu olduğunda analizin çok daha karmaşık olduğunu lütfen unutmayın. Tartışmalar için [PEIR99] ve [HAND98]'e bakınız.

# 5

## BÖLÜM

### DAHİLİ BELLEK

#### 5.1 Yarı İletken Ana Bellek

- Organizasyon DRAM ve
- SRAM
- ROM Çipi Mantık
- Çipi Paketleme
- Türleri
- Modül Organizasyonu Aralıklı Bellek

#### 5.2 Hata Düzeltme

#### 5.3 DDR DRAM

- Senkron DRAM DDR
- SDRAM

#### 5.4 Flash Bellek

- Operasyon
- NOR ve NAND Flash Bellek

#### 5.5 Yeni Uçucu Olmayan Katı Hal Bellek Teknolojileri

- STT-RAM
- PCRAM
- ReRAM

#### 5.6 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler

**ÖĞRENİM HEDEFLERİ**

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- Yarı iletken ana belleğin temel türlerine genel bir bakış sunmak.
- 8 bitlik sözcüklerdeki tek bitlik hataları tespit edip düzeltibilem temel bir kodun işleyişini anlamak.
- Çağdaş DDR DRAM organizasyonlarının özelliklerini özetleyiniz.
- NOR ve NAND flaş bellek arasındaki farkı anlamak.
- Yeni uçucu olmayan katı hal bellek teknolojilerine genel bir bakış sunmak.

Bu bölümde ROM, DRAM ve SRAM de dahil olmak üzere yarı iletken ana bellek alt sistemlerinin bir incelemesiyle başlıyoruz. Daha sonra bellek güvenilirliğini artırmak için kullanılan hata kontrol tekniklerine bakacağız. Bunu takiben, daha gelişmiş DRAM mimarilerine bakacağız.

## 5.1 YARI KONDÜKSİYONEL ANA MEMOrY

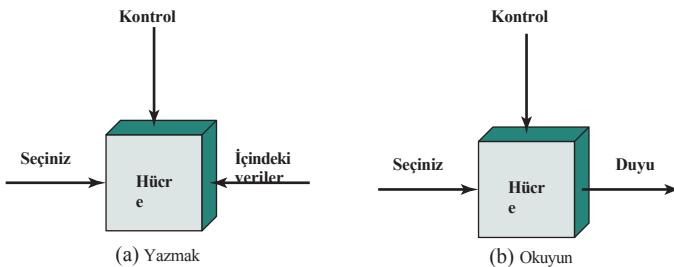
Daha önceki bilgisayarlarda, bilgisayar ana belleği için rastgele erişimli depolamanın en yaygın biçim, **çekirdek** olarak adlandırılan halka şeklindeki ferromanyetik döngülerden oluşan bir dizi kullanıyordu. Bu nedenle, ana bellek genellikle **çekirdek** olarak adlandırılır ve bu terim günümüzde kadar varlığını sürdürmüştür. Mikroelektronikin ortaya çıkış ve sağladığı avantajlar, manyetik çekirdekli belleği uzun zaman ortadan kaldırılmıştır. Bugün, ana bellek için yarı iletken çiplerin kullanımı neredeyse evrenseldir. Bu teknolojinin temel yönleri bu bölümde incelenmektedir.

### Organizasyon

**Yarı iletken** bir **belleğin** temel unsuru bellek hücresidir. Çok çeşitli elektronik teknolojiler kullanılmasına rağmen, tüm yarı iletken bellek hücreleri belirli özellikleri paylaşırlar:

- İkili 1 ve 0'ı temsil etmek için kullanılabilen iki kararlı (veya yarı kararlı) durum sergilerler.
- Durumu ayarlamak için (en az bir kez) içine yazılabilirler.
- Durumu algılamak için okunabilirler.

Şekil 5.1'de bir bellek hücresinin çalışması gösterilmektedir. En yaygın olarak, hücrenin elektrik sinyali taşıyabilen üç işlevsel terminali vardır. Seçme terminali, adından da anlaşılacağı gibi, okuma veya yazma işlemi için bir bellek hüresini seçer. Kontrol terminali okuma ya da yazmayı gösterir. Yazma işlemi için, diğer terminal hücrenin durumunu 1 ya da 0 olarak ayarlayan bir elektrik sinyali sağlar. Okuma işlemi için, bu terminal hücrenin durumunun çıkışı için kullanılır. Bellek hüresinin iç organizasyonu, işleyışı ve zamanlamasının ayrıntıları kullanılan özel entegre devre teknolojisine bağlıdır ve kısa bir özet dışında bu kitabın kapsamı dışındadır. Bizim amaçlarımız doğrultusunda, okuma ve yazma işlemleri için tek tek hücrelerin seçilebileceğini kabul edeceğiz.



### **Şekil 5.1** Bellek Hücresinin Çalışması

## DRAM ve SRAM

Bu bölümde inceleyeceğimiz bellek türleri rastgele erişimlidir. Yani, tek tek bellek sözcüklerine kablolu adresleme mantığı aracılığıyla doğrudan erisilir.

Tablo 5.1'de baslıca varyiletken bellek türleri listelenmektedir. En yaygın olanı rastgele erişimli bellek

(RAM) olarak adlandırılır. Bu aslında terimin yanlış kullanılmıştır, çünkü tabloda listelenen tüm türler rastgele erişimlidir. RAM olarak adlandırılan belleğin ayırt edici özelliklerinden biri, hem bellekten veri okumanın hem de belleğe kolayca ve hızla yeni veri yazmanın mümkün olmasıdır. Hem okuma hem de yazma işlemi elektrik sinyalleri kullanılarak gerçekleştirilir. Geleneksel RAM'in diğer ayırt edici

Bir RAM'e sabit bir güç kaynağı sağlanmalıdır. Güç kesilirse, veriler kaybolur. Bu nedenle, RAM yalnızca geçici depolama olarak kullanılabilir. Bilgisayarlarda kullanılan iki geleneksel RAM çeşidi DRAM ve SRAM'dir. Bölüm 5.5'te tartışılan daha veni RAM bicimleri ucucu değildir.

**DİNAMİK RAM** RAM teknolojisi iki teknolojiye ayrılr: dinamik ve statik. **Dinamik RAM (DRAM)**, verileri kapasitörler üzerinde yük olarak depolayan hücrelerle yapılır. Bir kondansatördeki yükün varlığı veya yokluğu ikili 1 veya 0 olarak yorumlanır. Kondansatörlerin doğal bir deşarj eğilimi olduğundan, dinamik RAM'ler veri depolamayı sürdürmek için periyodik sari venileme gerektirir. Terim

**Tablo 5.1** Yarı İletken Bellek Türleri

| Bellek Türü                          | Kategori                 | Silinme                              | Yazma Mekanizması  | Volatilite    |
|--------------------------------------|--------------------------|--------------------------------------|--------------------|---------------|
| Rastgele erişimli bellek (RAM)       | Okuma-yazma belleği      | Elektriksel olarak, bayt düzeyinde   | Elektriksel olarak | Uçucu         |
| Salt okunur bellek (ROM)             | Salt okunur bellek       | Mümkin değil                         | Maskeler           | Uçucu olmayan |
| Programlanabilir ROM (PROM)          |                          |                                      |                    |               |
| Silinebilir PROM (EPROM)             | Çoğunlukla okunur bellek | UV ışığı, çip seviyesi               | Elektriksel olarak |               |
| Elektrikle Silinebilir PROM (EEPROM) |                          | Elektriksel olarak, bayt düzeyinde   |                    |               |
| Flash bellek                         |                          | Elektriksel olarak, blok seviyesinde |                    |               |

*dinamik*, depolanan yükün sürekli güç uygulansa bile sızma eğilimini ifade eder.

Sekil 5.2a, bir bit saklayan tek bir hücre için tipik bir DRAM yapısıdır. Adres hattı, bu hücreden bit değeri okunacağı veya yazılıcağı zaman etkinleştirilir. Transistör, adres hattına bir voltaj uygulandığında kapalı olan (akımın akmasına izin veren) ve adres hattında voltaj yoksa açık olan (akım akmayan) bir anahtar görevi görür.

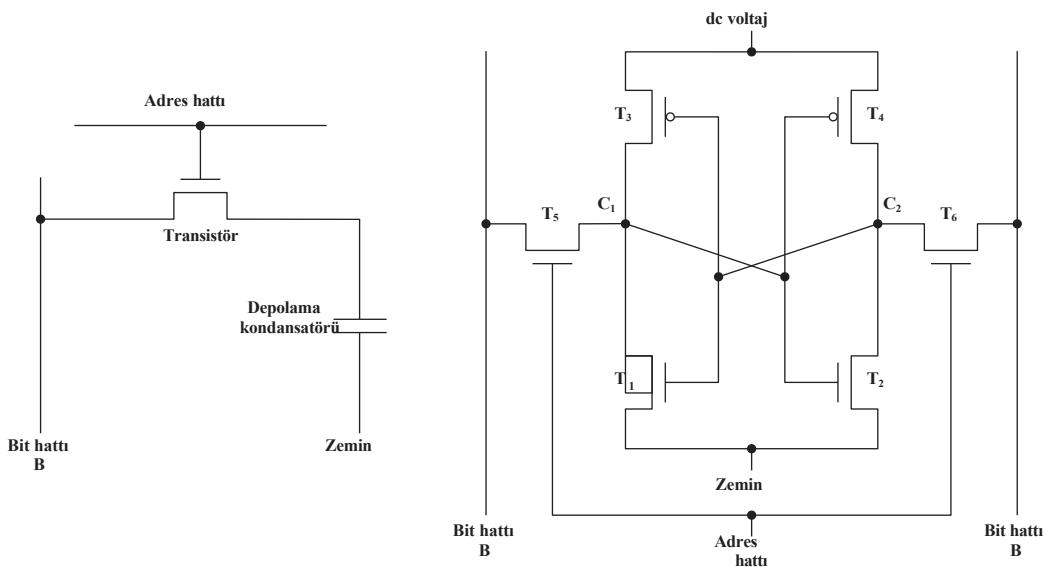
Yazma işlemi için bit hattına bir voltaj sinyali uygulanır; yüksek voltaj 1'i, düşük voltaj ise 0'ı temsil eder. Daha sonra adres hattına bir sinyal uygulanarak kondansatöre bir yük aktarılması sağlanır.

Okuma işlemi için, adres hattı seçildiğinde, transistör açılır ve kapasitörde depolanan yük bir bit hattına ve bir algılama amplifikatörüne beslenir. Algılama amplifikatörü kapasitör voltajını bir referans değeriyle karşılaştırır ve hücrenin bir mantık 1 veya bir mantık 0 içerip içermediğini belirler. Hücreden okunan değer kondansatörü boşaltır ve işlemin tamamlanması için kondansatörün tekrar doldurulması gereklidir.

DRAM hücresi tek bir bit (0 veya 1) depolamak için kullanılsa da, esasen analog bir cihazdır. Kondansatör bir aralıktaki herhangi bir yük değerini depolayabilir; bir eşik değeri yükün 1 ya da 0 olarak yorumlanıp yorumlanmayacağı belirler.

**STATİK RAM** Buna karşın, **statik RAM (SRAM)** işlemcide kullanılan aynı mantık elemanlarını kullanan dijital bir cihazdır. Bir SRAM'de ikili değerler geleneksel flip-flop mantık-kapı konfigürasyonları kullanılarak saklanır (flip-flopların açıklaması için Bölüm 11'e bakın). Statik bir RAM, kendisine güç sağlandığı sürece verilerini tutacaktır.

Sekil 5.2b tek bir hücre için tipik bir SRAM yapısıdır. Dört transistör ( $T_1$ ,  $T_{(2)}$ ,  $T_3$ ,  $T_4$ ) kararlı bir mantık üreten bir düzenlemeye çapraz bağlanmıştır



(a) Dinamik RAM (DRAM) hüresi

(b) Statik RAM (SRAM) hüresi

**Sekil 5.2** Tipik Bellek Hüresi Yapıları

durum. Mantık durumu 1'de  $C_{(1)}$ noktası yüksek ve  $C_2$ noktası düşüktür; bu durumda  $T_1$ 've  $T_4$  kapalı ve  $T_2$ 've  $T_3$  açıktır.<sup>1</sup> Mantık durumu 0'da  $C_{(1)}$ noktası düşük ve  $C_2$ noktası yüksektir; bu durumda  $T_1$ 've  $T_4$  açık ve  $T_2$ 've  $T_3$  kapalıdır. Doğru akım (dc) voltagı uygulandığı sürece her iki durum da kararlıdır. DRAM'in aksine, verileri korumak için yenileme gerekmekz. DRAM'de olduğu gibi, SRAM adres hattı bir anahtarı açmak veya kapatmak için kullanılır.

Adres hattı iki transistörü ( $T_5$  ve  $T_6$ ) kontrol eder. Bu hatta bir sinyal uygulandığında, iki transistör açılarak bir okuma veya yazma işlemine izin verir. Bir yazma işlemi için, istenen bit değeri B hattına uygulanırken, tamamlayıcısı B hattına uygulanır. Bu, dört transistörü ( $T_1$ ,  $T_2$ ,  $T_3$ ,  $T_4$ ) uygun duruma zorlar. Bir okuma işlemi için, bit değeri B hattından okunur.

**SRAM VERSUS DRAM** Hem statik hem de dinamik RAM'ler uçucudur; bit değerlerini korumak için belleğe sürekli olarak güç sağlanmalıdır. Dinamik bir bellek hücresi statik bir bellek hücreinden daha basit ve daha küçüktür. Bu nedenle, bir DRAM daha yoğundur (daha küçük hücreler= birim alan başına daha fazla hücre) ve karşılık gelen bir SRAM'den ucuzdur. Öte yandan, bir DRAM destekleyici yenileme devresi gerektir. Daha büyük bellekler için, yenileme devresinin sabit maliyeti, DRAM hücrelerinin daha küçük değişken maliyeti ile telafi edilenden daha fazladır. Bu nedenle, DRAM'ler büyük bellek gereksinimleri için tercih edilme eğilimindedir. Son bir nokta da SRAM'lerin DRAM'lerden biraz daha hızlı olmasıdır. Bu göreceli özellikler nedeniyle, SRAM ön bellek için (hem yonga üzerinde hem de yonga dışında) ve DRAM ana bellek için kullanılır.

## ROM Türleri

Adından da anlaşılabileceği gibi, **salt okunur bellek (ROM)** değiştirilemeyen kalıcı bir veri modeli içerir. Bir ROM uçucu değildir; yani bellekteki bit değerlerini korumak için herhangi bir güç kaynağı gerekmekz. Bir ROM'u okumak mümkün olsa da, içine yeni veri yazmak mümkün değildir. ROM'ların önemli bir uygulaması, Dördüncü Bölüm'de ele alınan mikro programlamadır. Diğer potansiyel uygulamalar şunlardır

- Sık aranan fonksiyonlar için kütüphane alt rutinleri
- Sistem programları
- Fonksiyon tabloları

Mütevazi büyülüklükteki bir gereksinim için ROM'un avantajı, veri veya programın kalıcı olarak ana bellekte bulunuması ve asla ikincil bir depolama cihazından yüklenmesine gerek olmamasıdır.

Bir ROM diğer entegre devre çipleri gibi oluşturulur ve veriler üretim sürecinin bir parçası olarak çipe bağlanır. Bu durum iki sorun ortaya çıkarmaktadır:

- Veri ekleme adımı, belirli bir ROM'un bir veya binlerce kopyasının üretilip üretilmediğine bakılmaksızın nispeten büyük bir sabit maliyet içerir.
- Hataya yer yoktur. Eğer bir bit yanlışsa, tüm ROM grubu atılmalıdır.

Belirli bir bellek içeriğine sahip az sayıda ROM'a ihtiyaç duyulduğunda, daha ucuz bir alternatif **programlanabilir ROM**'dur (PROM). Tipki

---

<sup>1</sup> Şekil 5.2b'de  $T_3$  ve  $T_4$  ile ilişkili daireler sinyal olumsuzlaşmasını gösterir.

ROM, PROM **uçucu** degildir ve yalnızca bir kez yazılabilir. PROM için yazma işlemi elektriksel olarak gerçekleştirilir ve orijinal çip üretiminden daha sonraki bir zamanda bir tedarikçi veya müsteri tarafından gerçekleştirilebilir. Yazma veya "programlama" işlemi için özel ekipman gereklidir. PROM'lar esneklik ve kolaylık sağlar. ROM, yüksek hacimli üretim çalışmaları için cazibesini korumaktadır.

Salt okunur belleğin bir diğer çeşidi, okuma işlemlerinin yazma işlemlerinden çok daha sık olduğu ancak uçucu olmayan depolamanın gerekli olduğu uygulamalar için yararlı olan salt **okunur** bellektir. Çoğunlukla okunur belleğin üç yaygın biçimi vardır: EPROM, EEPROM ve flash bellek.

Optik olarak **silinebilir programlanabilir salt okunur bellek (EPROM)**, PROM'da olduğu gibi elektriksel olarak okunur ve yazılır. Bununla birlikte, bir yazma işleminden önce, paketlenmiş çipin ultraviyole radyasyona maruz bırakılmasıyla tüm depolama hücrelerinin aynı başlangıç durumuna silinmesi gereklidir. Silme işlemi, bellek yongasının içine tasarlannmış bir pencereden yoğun bir ultraviyole ışığın geçirilmesiyle gerçekleştirilir. Bu silme işlemi tekrar tekrar gerçekleştirilebilir; her bir silme işleminin gerçekleştirilmesi 20 dakika kadar sürebilir. Böylece EPROM birçok kez değiştirilebilir ve ROM ve PROM gibi verilerini neredeyse süresiz olarak tutar. Karşılaştırılabilir miktarda depolama için, EPROM PROM'dan daha pahalıdır, ancak çoklu güncelleme özelliğinin avantajına sahiptir.

Çoğunlukla **okunabilir** belleğin daha çekici bir biçimi **elektrikle silinebilir programlanabilir salt okunur bellektir (EEPROM)**. Bu, önceki içerikleri silmeden herhangi bir zamanda içine yazılabilen çoğunlukla okunabili bir bellektir; yalnızca adreslenen bayt veya baytlar güncellenir. Yazma işlemi, bayt başına birkaç yüz mikrosaniye mertebesinde, okuma işleminden çok daha uzun sürer. EEPROM, sıradan veri yolu kontrol, adres ve veri hatlarını kullanarak yerinde güncellenebilir olma esnekliği ile uçuculuk avantajını birleştirir. EEPROM, EPROM'dan daha pahalıdır ve ayrıca çip başına daha az biti destekleyerek daha az yoğundur.

Yarı iletken belleğin bir başka biçimi de **flash bellektir** (yeniden programlanabilme hızı nedeniyle şekilde adlandırılmıştır). İlk olarak 1980'lerin ortalarında tanıtılan flash bellek, hem maliyet hem de işlevsellik açısından EPROM ve EEPROM arasında yer alır. EEPROM gibi flash bellek de elektriksel silme teknolojisi kullanır. Bir flash belleğin tamamı bir ya da birkaç saniye içinde silinebilir ki bu EPROM'dan çok daha hızlıdır. Buna ek olarak, tüm bir çip yerine sadece bellek bloklarını silmek mümkündür. Flash bellek adını, mikroçipin bellek hücrelerinin bir bölümünün tek bir işleme veya "flash" olarak silineceği şekilde organize edilmesinden alır. Ancak, flash bellek bayt düzeyinde silme sağlamaz. EPROM gibi, flash bellek de bit başına yalnızca bir transistör kullanır ve böylece 'un yüksek yoğunluğuna (EEPROM ile karşılaştırıldığında) ulaşır.

### **Çip Mantığı**

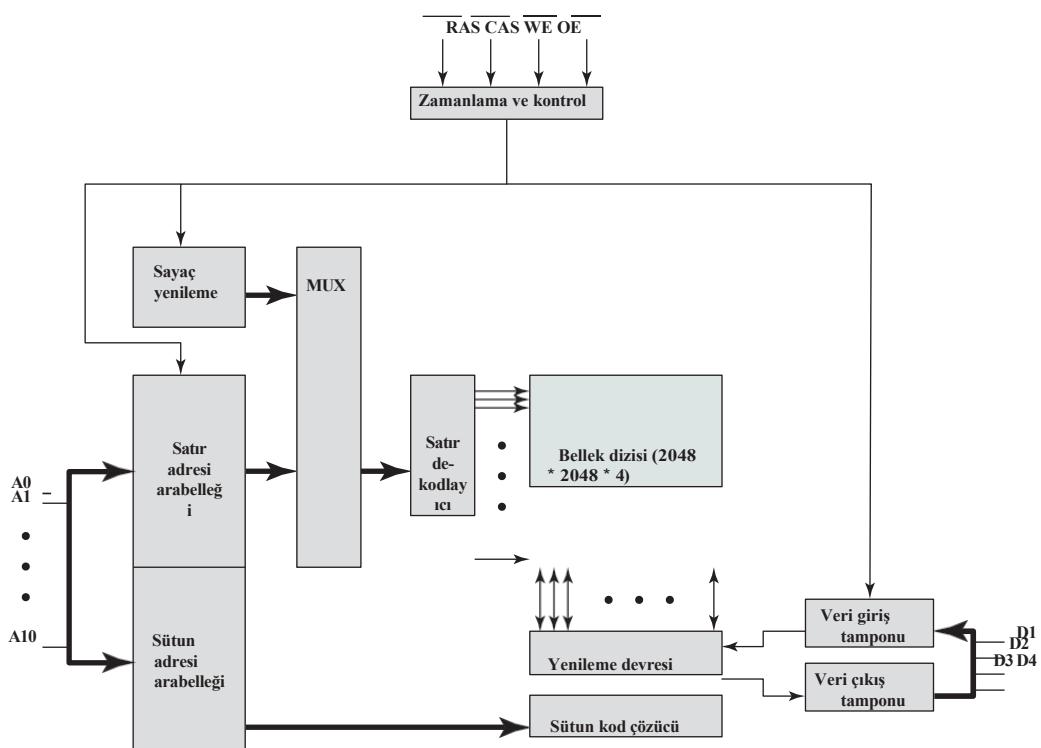
Diğer entegre devre ürünlerinde olduğu gibi, yarı iletken bellekler de paketlenmiş yongalar halinde gelir (Şekil 1.11). Her çip bir dizi bellek hücresi içerir.

Bir bütün olarak bellek hiyerarşisinde hız, yoğunluk ve maliyet arasında ödünləşmeler olduğunu gördük. Bu değişim tokuslar, bellek hücrelerinin ve işlevsel mantığın bir çip üzerindeki organizasyonunu göz önünde bulundurduğumuzda da mevcuttur. Yarı iletken bellekler için temel tasarım konularından biri, bir seferde okunabilecek/yazılabilecek veri bitlerinin sayısıdır. Bir ucta, dizideki hücrelerin fiziksel düzeninin bellekteki kelimelerin mantıksal düzeniyle (algılayıcı tarafından algılandığı gibi) aynı olduğu bir organizasyon vardır. Dizi, her biri  $B$  bitlik  $W$  kelime şeklinde düzenlenmiştir.

Örneğin, 16 Mbit'lik bir çip 1 milyon 16 bitlik kelime olarak düzenlenebilir. Diğer uçta ise verilerin her seferinde bir bit okunduğu/yazıldığı çip başına 1 bit organizasyonu yer alır. Bellek yongası organizasyonunu bir DRAM ile göstereceğiz; ROM organizasyonu daha basit olsa da benzerdir.

**Şekil 5.3** 16-Mbit DRAM'in tipik bir organizasyonunu göstermektedir. Bu durumda, bir 4 bit okunur veya yazılır. Mantıksal olarak, bellek dizisi 2048'e 2048 elemandan oluşan dört kare dizi olarak düzenlenmiştir. Çeşitli fiziksel düzenlemeler mümkündür. Her durumda, dizinin elemanları hem yatay (satır) hem de dikey (sütun) çizgilerle bağlanır. Her yatay çizgi kendi satırındaki her hücrenin Select terminaline bağlanır; her dikey çizgi kendi sütunundaki her hücrenin Data-In/Sense terminaline bağlanır. Adres satırları seçilecek kelimenin adresini verir. Toplam  $\log_2 W$  satırına ihtiyaç vardır. Örneğimizde, 2048 satırdan birini seçmek için 11 adres satırına ihtiyaç vardır. Bu 11 satır, 11 satır girişe ve 2048 satır çıkışa sahip bir satır kod çözücüye beslenir. Kod çözücüünün mantığı, 11 giriş hattındaki bit desenine bağlı olarak 2048 çıkıştan tek birini etkinleştirir ( $2^{11} = 2048$ ).

İlave 11 adres hattı, sütün başına 4 bitlik 2048 sütünden birini seçer. Dört veri hattı, bir veri tamponuna 4 bitlik giriş ve çıkış için kullanılır. Girişte (yazma), her bir bit hattının bit sürücüsü, ilgili veri değerine göre 1 veya 0 için etkinleştirilir. Çıkışta (okuma), her bit hattının değeri bir algılama amplifikatöründen geçirilir ve veri hattlarına sunulur. Satır satırı, okuma veya yazma için hangi hücre satırının kullanılacağını seçer.



**Şekil 5.3** Tipik 16-Mbit DRAM (4M \* 4)

Bu DRAM'e sadece 4 bit okunduğundan/yazıldığından, veri yoluna bir kelime veri okumak/yazmak için bellek denetleyicisine bağlı birden fazla DRAM olmalıdır. Sadece 11 adres hattı (A0-A10) olduğuna dikkat edin, 2048 \* 2048 dizisi için beklediğiniz sayının yarısı. Bu, pin sayısından tasarruf etmek için yapılmıştır. Gerekli 22 adres hattı, çipin dışındaki seçme mantığından geçirilir ve 11 adres çoğullarıdır. İlk olarak, dizinin satır adresini tanımlamak için 11 adres sinyali çipe ve ardından diğer 11 adres sinyali sütun adresi için sunulur. Bu sinyallere, çipe zamanlama sağlamak için satır adresi seçme (RAS) ve sütun adresi seçme (CAS) sinyalleri eşlik eder. Yazma etkinleştirme (WE) ve çıkış etkinleştirme (OE) pinleri bir yazma veya okuma işleminin gerçekleştirilip gerçekleştirilmeyeceğini belirler. Şekil 5.3'te gösterilmeyen diğer iki pin şunlardır toprak ( $V_{ss}$ ) ve bir gerilim kaynağı ( $V_{cc}$ ).

Bir kenara, çoklu adresleme ve kare dizilerin kullanımı, her yeni nesil bellek yongasında bellek boyutunun dört katına çıkmasına neden olmaktadır. Adreslemeye ayrılmış bir pin daha satır ve sütun sayısını ikiye katlar ve böylece yonga belleğin boyutu 4 kat büyür.

Şekil 5.3 ayrıca yenileme devresinin dahil edildiğini de göstermektedir. Tüm DRAM'ler bir yenileme işlemi gerektirir. Yenileme için basit bir teknik, aslında, tüm veri hücreleri DRAM çipini devre dışı bırakmaktadır. Yenileme sayacı tüm satır değerleri boyunca adım atar. Her satır için, yenileme sayacından gelen çıkış hatları satır kod çözücüye beslenir ve RAS hattı etkinleştirilir. Veriler okunur ve aynı konuma geri yazılır. Bu, satırda her hücrenin yenilenmesine neden olur.

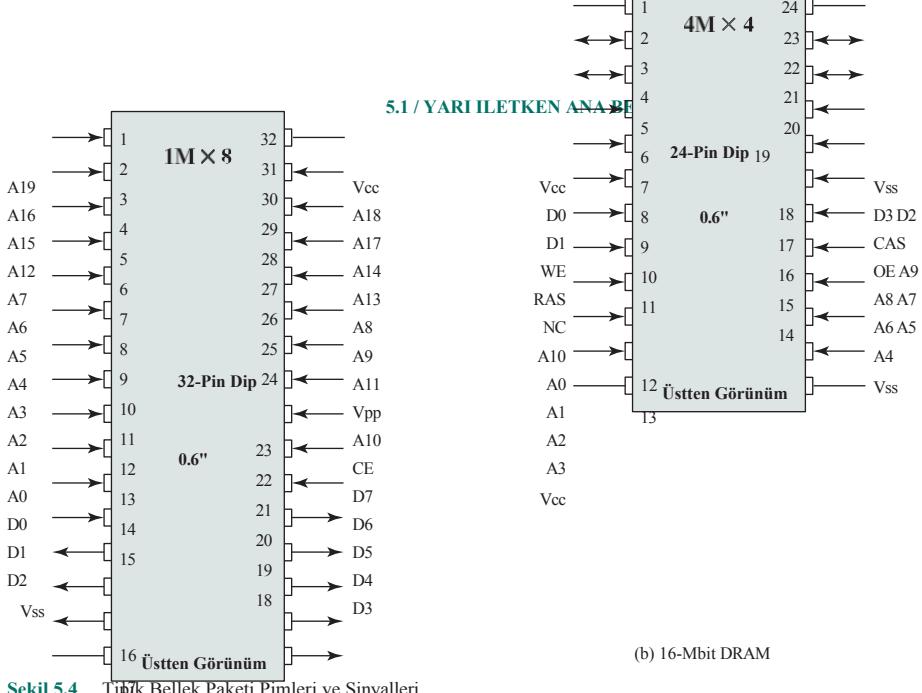
### Çip Paketleme

Bölüm 2'de belirtildiği gibi, bir entegre devre dış dünyaya bağlantı için pinler içeren bir paket üzerine monte .

Şekil 5.4a, 1M \* 8 olarak düzenlenmiş Mbit'lik bir çip olan örnek bir EPROM paketini göstermektedir. Bu durumda, organizasyon çip başına bir kelime paketi olarak ele alınır. Paket, standart çip paketi boyutlarından biri olan 32 pin içerir. Pinler aşağıdaki sinyal hatlarını destekler:

- Erişilen kelimenin adresi. 1M kelime için toplam 20 ( $2^{20} = 1M$ ) pine ihtiyaç vardır (A0-A19).
- Okunacak veri, 8 satırdan oluşur (D0-D7).
- Çipe giden güç kaynağı ( $V_{cc}$ ).
- Bir toprak pimi ( $V_{ss}$ ).
- Bir çip etkinleştirme (CE) pimi. Her biri aynı adres veriyoluyla bağlı birden fazla bellek yongası olabileceğinden, CE pimi adresin bu yonga için geçerli olup olmadığını belirtmek için kullanılır. CE pimi, adres veriyolunun daha yüksek sıralı bitlerine (yani A19'un üzerindeki adres bitlerine) bağlı mantık tarafından etkinleştirilir. Bu sinyalin kullanımı aşağıda gösterilmiştir.
- Programlama (yazma işlemleri) sırasında sağlanan bir program voltağı ( $V_{pp}$ ).

Tipik bir DRAM pin konfigürasyonu, 4M \* 4 olarak düzenlenen 16-Mbit'lik bir çip için Şekil 5.4b'de gösterilmiştir. Bir ROM çipinden birkaç farkı vardır. Bir RAM güncellenebildiğinden, veri pinleri giriş/çıkıştır. Yazma etkinleştirme (WE) ve çıkış etkinleştirme (OE) pinleri bunun bir yazma veya okuma işlemi olduğunu gösterir.



**Şekil 5.4** Tipik Bellek Paketi Pimleri ve Sinyaller

DRAM'e satır ve sütun bazında erişildiğinden ve adres çok katlı olduğundan, 4M satır/sütün kombinasyonunu belirtmek için sadece  $^{11}$  adres pini gereklidir ( $2^{11} * 2^{11} = 2^{22} = 4M$ ). Satır adresi seçme (RAS) ve sütun adresi seçme (CAS) pinlerinin işlevleri daha önce tartışılmıştı. Son olarak, çift sayıda pin olması için bağlantı yok (NC) pini sağlanmıştır.

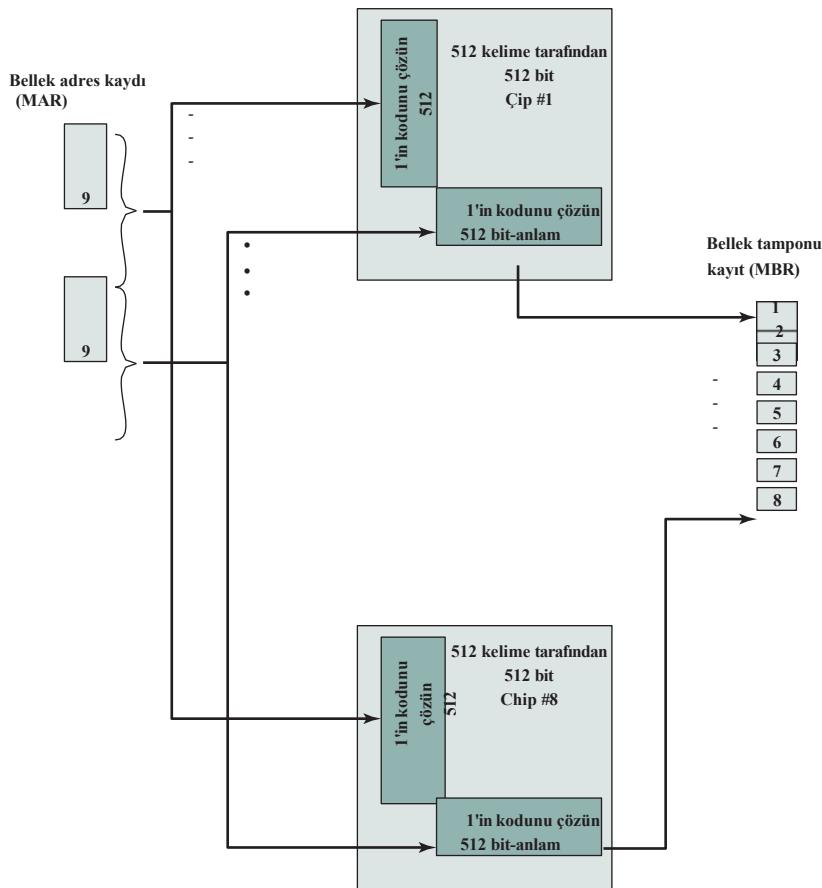
### Modül Organizasyonu

Bir RAM yongası kelime başına yalnızca bir bit, en azından kelime başına bit sayısına eşit sayıda yongaya ihtiyacımız olacağı açıklıktır. Örnek olarak, Şekil 5.5 256K 8-bit sözcükten oluşan bir bellek modülünün nasıl organize edilebileceğini göstermektedir. 256K kelime için 18 bitlik bir adres gereklidir ve modüle harici bir kaynaktan (örneğin modülün bağlı olduğu bir veri yoluun adres hatları) sağlanır. Adres, her biri bir bitlik giriş/çıkış sağlayan 8 adet 256K \* 1@bit yongaya sunulur.

Bu organizasyon, bellek boyutu çip başına düşen bit sayısına eşit sürece çalışır. Daha büyük belleğin gerekliliği olduğu durumda, bir yonga dizisine ihtiyaç vardır. Şekil 5.6, kelime başına 8 bit ile 1 milyon kelimededen oluşan bir belleğin olası organizasyonunu göstermektedir. Bu durumda, Şekil 5.5'teki gibi düzenlenmiş 256K kelime içeren her bir sütun olmak üzere dört sütun vardır. 1M kelime için 20 adres hattına ihtiyaç vardır. En az anlamlı 18 bit 32 modülün tamamına yönlendirilir. Yüksek sıralı 2 bit, dört modül sütunundan birine bir çip etkinleştirme sinyali gönderen bir grup seçme mantık modülüne girilir.

### Aralıklı Bellek

Ana bellek, DRAM bellek yongalarının bir araya gelmesinden oluşur. Bir dizi çip bir *bellek bankası* oluşturmak üzere bir araya getirilebilir. Belleği organize etmek mümkündür



Şekil 5.5 256-KByte Bellek Organizasyonu

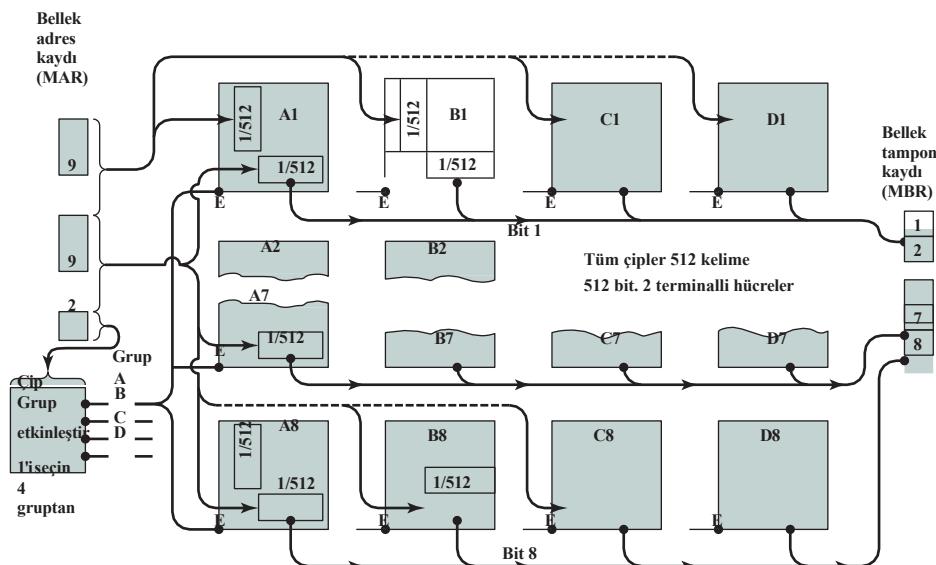
bankalar, serpiştirilmiş bellek olarak bilinen bir şekilde. Her banka bağımsız olarak bir bellek okuma ya da yazma isteğini karşılayabilir, böylece  $K$  bankalı bir sistem  $K$  isteği aynı anda karşılayabilir, bu da bellek okuma ya da yazma hızlarını  $K$  kat artırır. Ek G, serpiştirilmiş bellek konusunu incelemektedir.



Aralıklı Bellek Simülatörü

## 5.2 ErrOr cOrrEcTION

Yarı iletken bir bellek sistemi hatalara maruz kalır. Bunlar sert arızalar ve yumuşak hatalar olarak kategorize edilebilir. **Sert** hata kalıcı bir fiziksel kusurdur, böylece etkilenen bellek hücresi veya hücreleri verileri güvenilir bir şekilde depolayamaz, ancak 0 veya 1 veya



Şekil 5.6 1-MB Bellek Organizasyonu

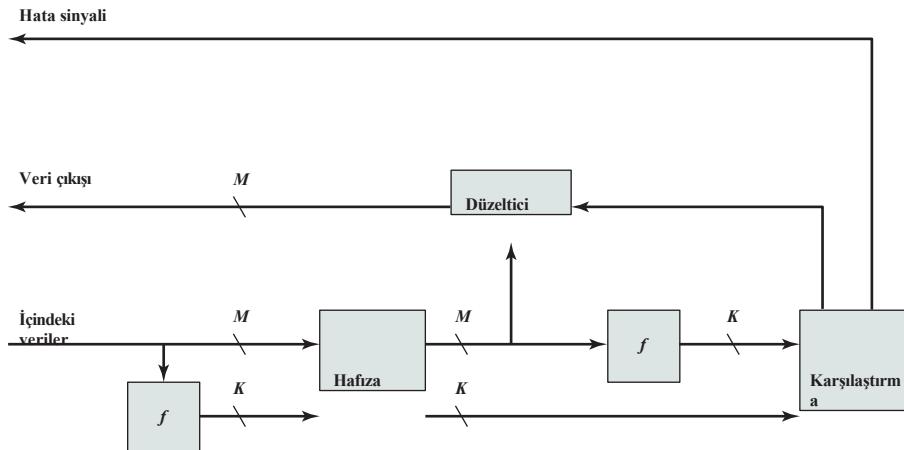
Sert hatalar zorlu çevre koşullarından, üretim hatalarından ve aşınmadan kaynaklanabilir. **Yumuşak hata**, belleğe zarar vermeden bir veya daha fazla bellek hücresinin içeriğini değiştiren rastgele, tahrif edici olmayan bir olaydır. Yumuşak hatalara güç kaynağı sorunları ya da alfa parçacıkları neden olabilir. Bu parçacıklar radyoaktif bozunmadan kaynaklanır ve radyoaktif çekirdekler neredeyse tüm malzemelerde küçük miktarlarda bulunduğu için endişe verici derecede yaygındır. Hem sert hem de yumuşak hatalar açıkça istemeyen durumlardır ve modern ana bellek sistemlerinin çoğu hataları tespit etmek ve düzeltmek için mantık içerir.

Şekil 5.7 sürecin nasıl yürütüldüğünü genel hatalarıyla göstermektedir. Veriler belleğe yazılacağı zaman, bir kod üretmek için veriler üzerinde  $f$  fonksiyonu olarak gösterilen bir hesaplama yapılır. Hem kod hem de veri saklanır. Dolayısıyla,  $M$  bitlik bir veri kelimesi saklanacaksa ve kod  $K$  bit uzunluğundaysa, saklanan kelimenin gerçek boyutu  $M+K$  bittir.

Önceden saklanan kelime okunduğunda, kod hataları tespit etmek ve muhtemelen düzeltmek için kullanılır.  $M$  veri bitinden yeni bir  $K$  kod biti kümesi oluşturulur ve getirilen kod karşılaştırılır. Karşılaştırma üç sonucdan birini verir:

- Hiçbir hata algılanmaz. Getirilen veri bitleri gönderilir.
- Bir hata tespit edilir ve hatayı düzeltmek mümkündür. Veri bitleri artı **hata düzeltme** bitleri, gönderilerek üzere düzeltilmiş bir  $M$  bitleri kümesi üreten bir düzelticiye beslenir.
- Bir hata tespit edildi, ancak düzeltilemesi mümkün değil. Bu durum rapor edilir.

Bu şekilde çalışan kodlar **hata düzeltici kodlar** olarak adlandırılır. Bir kod, bir kelimedeki düzeltilebileceği ve tespit edebileceği bit hatalarının sayısını ile karakterize edilir.



Şekil 5.7 Hata Düzeltme Kodu İşlevi

Hata düzeltme kodlarının en basitini Bell Laboratuvarlarında Richard **Hamming** tarafından tasarlanan Hamming **kodudur**. Şekil 5.8, bu kodun 4 bitlik sözcüklerde ( $M= 4$ ) kullanımını göstermek için Venn diyagramlarını kullanır. Kesişen üç daire ile yedi bölge vardır. İç bölmelere 4 veri biti atarız (Şekil 5.8a). Kalan bölmeler *eşlik bitleri* olarak adlandırılan bitlerle doldurulur. Her eşlik biti, kendi çemberindeki toplam 1 sayısı çift olacak şekilde seçilir (Şekil 5.8b). Böylece, A çemberi üç veri 1'i içerdikinden, bu çemberdeki eşlik biti olarak ayarlanır. Şimdi, bir hata veri bitlerinden birini değiştirirse (Şekil 5.8c), kolayca bulunur. Eşlik bitleri kontrol edildiğinde, A ve C dairelerinde tartsızlıklar bulunur ancak B dairesinde bulunmaz. Yedi bölmenden yalnızca biri A ve C'de bulunur ancak B'de bulunmaz (Şekil 5.8d). Bu nedenle hata bir bit değiştirilerek düzeltilebilir.

İlgili kavramları açıklığa kavuşturmak için, 8 bitlik sözcüklerdeki tek bitlik hataları tespit edip düzeltibilecek bir kod geliştireceğiz.

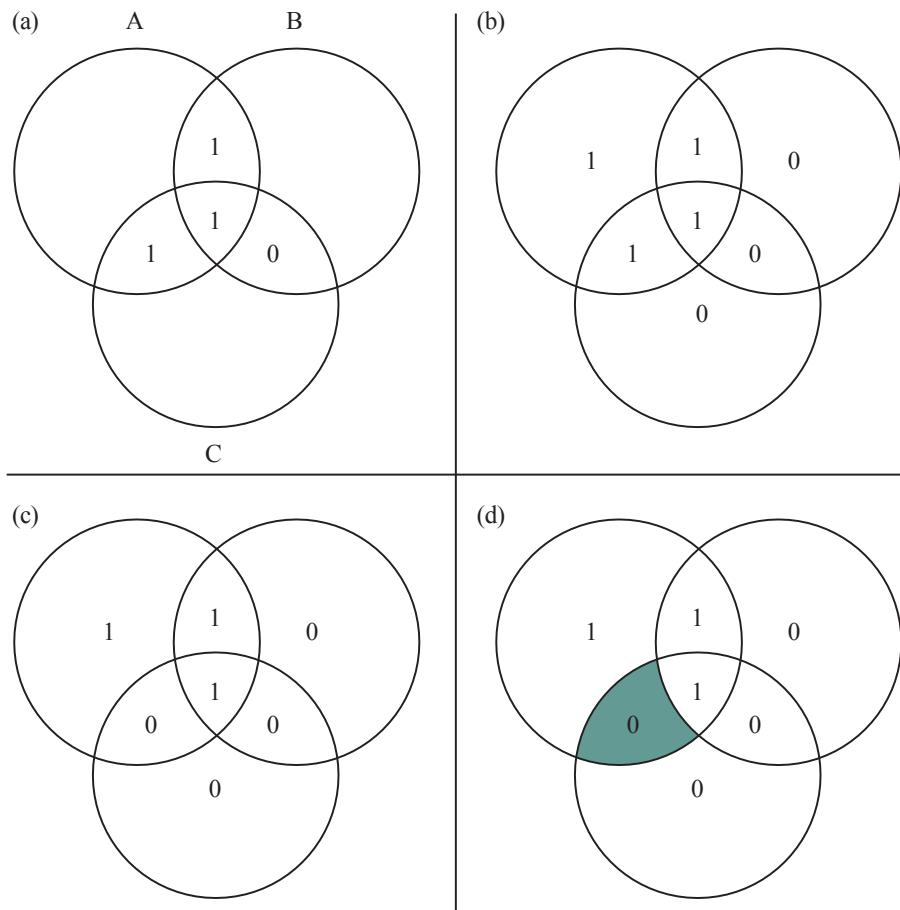
Başlangıç olarak, kodun ne kadar uzun olması gerektiğini belirleyelim. Şekil 5.7'ye bakacak olursak, karşılaştırma mantığı iki  $K$ -bit değeri giriş olarak alır. İki girişin özel-OR'si alınarak bit-bit karşılaştırma yapılır. Sonuç, *sendrom kelimesi* olarak adlandırılır. Böylece, *sendromun* her biti, iki giriş için o bit konumunda bir eşleşme olup olmamasına göre 0 veya 1 olur.

Bu nedenle sendrom kelimesi  $K$  bit genişliğindedir ve 0 ile  $2^K-1$  arasında bir aralığa sahiptir. 0 değeri herhangi bir hata tespit edilmediğini gösterir, geriye hata varsa hangi bitin olduğunu belirtmek için  $2^K-1$  değerleri kalır. Şimdi,  $M$  veri biti veya  $K$  kontrol bitinden herhangi birinde hata oluşabileceğinden, şu değerlere sahip olmamalıyız

$$2^K-1 \leq M+K$$

Bu eşitsizlik,  $M$  veri içeren bir kelimedeki tek bir bit hatasını düzeltmek için gereken bit sayısını verir. Örneğin, 8 veri bitinden oluşan bir kelime için ( $M= 8$ )

- $K= 3: 2^3-1 \geq 8+3$
- $K= 4: 2^4-1 \geq 8+4$



**Şekil 5.8** Hamming Hata Düzeltme Kodu

Böylece, sekiz veri biti dört kontrol biti gerektirir. Tablo 5.2'nin ilk üç sütunu çeşitli veri kelimesi uzunlukları için gereken kontrol bitlerinin sayısını listeler.

Kolaylık olması açısından, aşağıdaki özelliklere sahip 8 bitlik bir veri sözcüğü için 4 bitlik bir sendrom oluşturmak istiyoruz:

- Sendrom tüm 0'ları içeriyorsa, herhangi bir hata tespit edilmemiştir.
- Sendrom 1'e ayarlanmış bir ve sadece bir bit içeriyorsa, 4 kontrol bitinden birinde bir hata meydana gelmiştir. Düzeltmeye gerek yoktur.
- Sendrom 1'e ayarlanmış birden fazla bit içeriyorsa, sendromun sayısal değeri hatalı veri bitinin konumunu gösterir. Bu veri biti düzeltme için ters çevrilir.

Bu özellikleri elde etmek için, veri ve kontrol bitleri Şekil 5.9'da gösterildiği gibi 12 bitlik bir kelime halinde düzenlenmiştir. Bit konumları 1'den 12'ye kadar numaralandırılır. Pozisyon numaraları 2'nin kuvvetleri olan bit pozisyonları kontrol biti olarak belirlenir.

**Tablo 5.2** Hata Düzeltme ile Sözcük Uzunluğundaki Artış

|              |   | Tek Hatalı Düzeltme |         |                 | Tek Hatalı Düzeltme/ Çift Hatalı Algılama |         |
|--------------|---|---------------------|---------|-----------------|-------------------------------------------|---------|
| Veri Bitleri |   | Kontrol Bitleri     | % Artış | Kontrol Bitleri |                                           | % Artış |
| 8            | 4 |                     | 50.0    | 5               |                                           | 62.5    |
| 16           | 5 |                     | 31.25   | 6               |                                           | 37.5    |
| 32           | 6 |                     | 18.75   | 7               |                                           | 21.875  |
| 64           | 7 |                     | 10.94   | 8               |                                           | 12.5    |
| 128          | 8 |                     | 6.25    | 9               |                                           | 7.03    |
| 256          | 9 |                     | 3.52    | 10              |                                           | 3.91    |

bitleri. Kontrol aşağıdaki gibi hesaplanır, burada  $\oplus$  simbolü exclusive-OR işlemini belirtir:

$$\begin{aligned}
 C1 &= D1 \ D2 \ D4 \oplus \oplus \quad \oplus \oplus \quad D7 \\
 D5C2 &= D1 \ D3 \oplus \oplus \ D4 \ D6 \oplus \oplus \ D7 \\
 C4 \ D2 \ D3 &= \oplus \oplus \oplus \quad D8 \ D4C8 = \oplus \\
 D5 \oplus D6 \oplus D7 \oplus D8
 \end{aligned}$$

Her kontrol biti, konum numarası kontrol bitinin konum numarasıyla aynı bit konumunda 1 içeren her veri biti üzerinde çalışır. Böylece, 3, 5, 7, 9 ve 11 numaralı veri biti konumlarının (D1, D2, D4, D5, D7) tümü C1 gibi konum numaralarının en az anlamlı bitinde bir 1 içerir; 3, 6, 7, 10 ve 11 numaralı bit konumlarının tümü C2 gibi ikinci bit konumunda bir 1 içerir; ve bu böyle devam eder. Başka bir şekilde bakıldığında, bit pozisyonu n

$C_i$  bitleri tarafından kontrol edilir, öyle  $\text{iki } \backslash \quad i = n$ . Örneğin, 7 konumu 4, 2 ve 1 konumlarındaki bitler kontrol edilir; ve  $7 = 4 + 2 + 1$ .

Bu şemanın çalıştığını bir örnekle doğrulayalım. 8 bitlik giriş sözcüğünün 00111001 olduğunu ve veri biti D1'in en sağ konumda bulduğunu varsayıyalım. Hesaplamlar aşağıdaki gibidir:

$$\begin{aligned}
 C1 &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\
 C2 &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\
 C4 &= 0 \oplus 0 \oplus 1 \oplus 0 = 1 \\
 C8 &= 1 \oplus 1 \oplus 0 \oplus 0 = 0
 \end{aligned}$$

|                   |      |      |      |      |      |      |      |      |      |      |      |      |
|-------------------|------|------|------|------|------|------|------|------|------|------|------|------|
| Bit pozisyon      | 12   | 11   | 10   | 9    | 8    | 7    | 6    | 5    | 4    | 3    | 2    | 1    |
| Pozisyon numarası | 1100 | 1011 | 1010 | 1001 | 1000 | 0111 | 0110 | 0101 | 0100 | 0011 | 0010 | 0001 |
| Veri biti         | D8   | D7   | D6   | D5   |      | D4   | D3   | D2   |      | D1   |      |      |
| Kontrol biti      |      |      |      |      | C8   |      |      |      | C4   |      | C2   | C1   |

**Şekil 5.9** Veri Bitleri ve Kontrol Bitlerinin Düzeni

Şimdi veri biti 3'ün bir hata aldığı ve 0'dan 1'e değiştigini varsayılmı. Kontrol bitleri yeniden hesaplandığında

$$C1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$C2 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$

$$C4 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$$

$$C8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

Yeni kontrol bitleri eski kontrol bitleriyle karşılaştırıldığında sendrom kelimesi oluşur:

$$\begin{array}{cccc}
 C8 & C4 & C2 & C1 \\
 0 & 1 & 1 & 1 \\
 \oplus 0 & 0 & 0 & 1 \\
 \hline
 0 & 1 & 1 & 0
 \end{array}$$

Sonuç 0110'dur ve veri biti 3'ü içeren bit konumu 6'nın hatalı olduğunu gösterir. Şekil 5.10 önceki hesaplamayı göstermektedir. Veri ve kontrol bitleri 12 bitlik kelime içinde düzgün bir şekilde konumlandırılmıştır. Veri bitlerinden dördünün değeri 1'dir (tabloda gölgeli) ve bit konum değerleri XORlanarak dört kontrol basamağını oluşturan 0111 Hamming kodu elde edilir. Depolanan tüm blok 001101001111 şeklindedir. Şimdi 6. bit konumundaki veri biti 3'ün bir hata aldığı ve 0'dan 1'e değiştirildiğini varsayılmı. Ortaya çıkan blok, Hamming kodu 0001 olan 00110110111'dur. Hamming kodunun ve sıfır olmayan veri bitleri için tüm bit konumu değerlerinin XOR'u 0110 ile sonuçlanır. Sıfır olmayan sonuç bir hata tespit eder ve

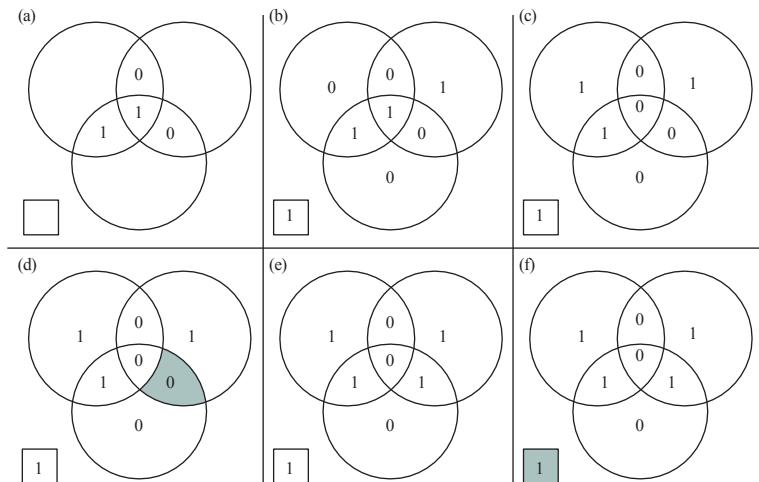
hata bit konumu 6'dadır.

Az önce açıklanan **kod tek hata düzeltmeli (SEC)** kod olarak bilinir. Daha yaygın olarak, yarı iletken bellek **tek hata düzeltmeli, çift hata algılamalı (SEC-DED)** bir kodla donatılmıştır. Tablo 5.2'de gösterildiği gibi, bu tür kodlar SEC kodlarına kıyasla bir ek bit gerektirir.

Şekil 5.11, yine 4 bitlik bir veri sözcüğü ile böyle bir kodun nasıl çalıştığını göstermektedir. Sıralama, iki hata meydana gelirse (Şekil 5.11c), kontrol prosedürünün yoldan çıktığını (d) ve üçüncü bir hata yaratarak e) sorunu daha da kötüleştirdiğini göstermektedir. Üstesinden gelmek için

| Bit pozisyon                | 12   | 11   | 10   | 9    | 8    | 7    | 6    | 5    | 4    | 3    | 2    | 1    |
|-----------------------------|------|------|------|------|------|------|------|------|------|------|------|------|
| Pozisyon numarası           | 1100 | 1011 | 1010 | 1001 | 1000 | 0111 | 0110 | 0101 | 0100 | 0011 | 0010 | 0001 |
| Veri biti                   | D8   | D7   | D6   | D5   |      | D4   | D3   | D2   |      | D1   |      |      |
| Kontrol biti                |      |      |      |      | C8   |      |      |      | C4   |      | C2   | C1   |
| Olarak saklanan kelime      | 0    | 0    | 1    | 1    | 0    | 1    | 0    | 0    | 1    | 1    | 1    | 1    |
| Kelime şu şekilde getirildi | 0    | 0    | 1    | 1    | 0    | 1    | 1    | 0    | 1    | 1    | 1    | 1    |
| Pozisyon sayı               | 1100 | 1011 | 1010 | 1001 | 1000 | 0111 | 0110 | 0101 | 0100 | 0011 | 0010 | 0001 |
| Kontrol biti                |      |      |      |      | 0    |      |      |      | 0    |      | 0    | 1    |

Şekil 5.10 Kontrol Biti Hesaplaması



Şekil 5.11 Hamming SEC-DEC Kodu

Sorun çözüldüğünde, diyagramdaki toplam 1 sayısı çift olacak şekilde ayarlanan sekizinci bir bit eklenir. Ekstra eşlik biti hatayı yakalar (f).

Hata düzeltici bir kod, ilave karmaşıklık pahasına belleğin güvenilirliğini artırır. Çip başına 1 bitlik bir organizasyonla, bir SEC-DED kodu genel olarak yeterli kabul edilir. Örneğin, IBM 30xx uygulamalarında ana bellekteki her 64 bitlik veri için 8 bitlik bir SEC-DED kodu kullanılmıştır. Böylece, ana belleğin boyutu aslında kullanıcıya görünenden yaklaşık %12 daha büyütür. VAX bilgisayarları her 32 bitlik bellek için 7 bitlik SEC-DED kodu kullanarak %22'lük bir ek yük oluşturmuştur. Çağdaş DRAM sistemleri %7 ile %20 arasında bir ek yükle sahip olabilir [SHAR03].

### 5.3 ddr drAM

Bölüm 1'de tartışıldığı gibi, yüksek performanslı işlemciler kullanılırken en kritik sistem darboğazlarından biri dahili ana belleğe giden arayözdür. Bu ara yüz tüm bilgisayar sistemindeki en önemlidir. Ana belleğin temel yapı taşı onlarca yıldır olduğu gibi DRAM çipi olmaya devam etmektedir; yakın zamana kadar DRAM mimarisinde 1970'lerin başından bu yana önemli bir değişiklik yapılmamıştı. Geleneksel DRAM çipi hem iç mimarisini hem de işlemcinin bellek veri yoluna olan arayüzü tarafından kısıtlamıştır.

DRAM ana belleğin performans sorununa yönelik bir saldırının DRAM ana bellek ile işlemci arasına bir ya da daha fazla seviyede yüksek hızlı SRAM önbelleyek eklemek olduğunu görmüştük.

Ancak SRAM, DRAM'den çok daha maliyetlidir ve önbelleyek boyutunu belirli bir noktanın ötesine genişletmek azalan getiri sağlar.

Son yıllarda, temel DRAM mimarisinde bir dizi iyileştirme araştırılmıştır. Şu anda piyasaya hakim olan şemalar SDRAM ve DDR-DRAM'dir. Bunların her birini sırayla inceleyeceğiz.

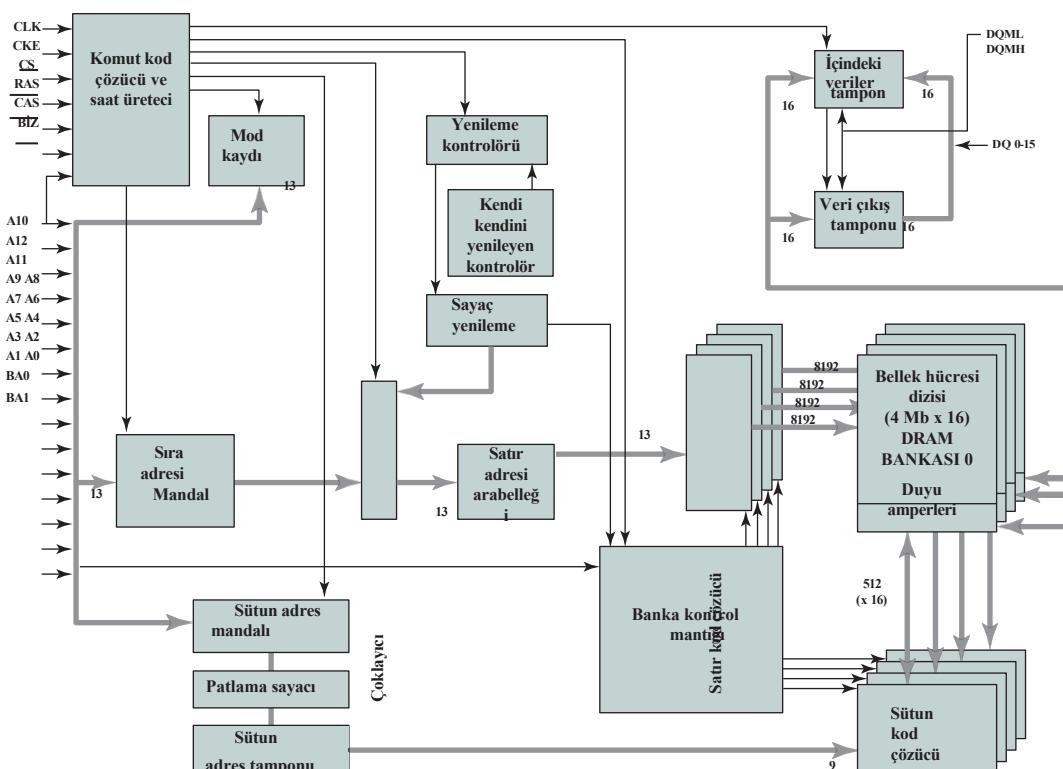
## Senkron DRAM

DRAM'in en yaygın kullanılan formlarından biri **senkron DRAM**'dir (**SDRAM**). Asenkron olan geleneksel DRAM'in aksine SDRAM, harici bir saat sinyaline senkronize olan ve bekleme durumları uygulamadan işlemci/bellek veriyolunun tam hızında çalışan işlemciyle veri alışverişini yapar.

Tipik bir DRAM'de, işlemci adresler ve kontrol seviyeleri sunarak, bellekteki belirli bir konumdaki bir dizi verinin DRAM'den okunması ya da DRAM'e yazılması gerektiğini belirtir. Erişim süresi olarak adlandırılan bir gecikmeden sonra DRAM verileri yazar ya da okur. Erişim zamanı gecikmesi sırasında DRAM, satır ve sütun hatlarının yüksek kapasitansını etkinleştirmek, verileri algılamak ve verileri çıkış tamponlarından dışarı yönlendirmek gibi çeşitli dahili işlevleri yerine getirir.

İşlemcinin bu gecikme boyunca beklemesi gereklidir, bu da sistem performansını yavaşlatır. Senkron erişim ile DRAM, sistem saatinin kontrolü altında verileri içeri ve dışarı taşırlar. İşlemci veya diğer ana birim, DRAM tarafından kilitlenen komut ve adres bilgilerini verir. DRAM daha sonra belirli sayıda saat döngüsünden sonra yanıt verir. arada, SDRAM isteği işlerken master güvenli bir şekilde diğer görevleri yerine getirebilir.

**Sekil 5.12** tipik bir 256-Mb SDRAM' dahili mantığını göstermektedir ve Tablo 5.3 çeşitli pin atamalarını tanımlamaktadır. Tablo 5.3



**Sekil 5.12** 256-Mb Senkron Dinamik RAM (SDRAM)

Tablo 5.3 SDRAM Pin Atamaları

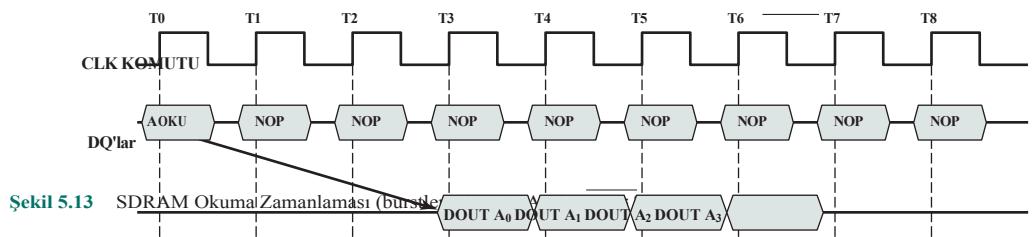
|            |                       |
|------------|-----------------------|
| A0 - A13   | Adres girişleri       |
| BA0, BA1   | Banka adres satırları |
| CLK        | Saat girişi           |
| CKE        | Saat etkinleştirme    |
| <u>CS</u>  | Çip seçimi            |
| <u>RAS</u> | Satır adresi flaşı    |
| <u>CAS</u> | Sütun adresi flaşı    |
| <u>BLZ</u> | Yazma etkinleştirme   |
| DQ0 - DQ7  | Veri girişi/çıkışı    |
| DQM        | Veri maskesi          |

SDRAM, ilk erişimden sonra adres kurulum süresini ve satır ve sütun hattı ön şarj süresini ortadan kaldırarak için bir seri mod kullanır. Seri modda, ilk bit erişildikten sonra bir dizi veri biti hızla saatte çıkarılabilir. Bu mod, erişilecek tüm bitler sıralı olduğunda ve ilk erişimle dizinin aynı satırında olduğunda . Ayrıca SDRAM, çip içi paralellik fırsatlarını geliştiren çok bankalı bir iç mimariye sahiptir.

Mod kaydı ve ilgili kontrol mantığı, SDRAM'leri geleneksel DRAM'lerden ayıran bir diğer önemli özelliklektir. SDRAM'i belirli sistem ihtiyaçlarına uyacak şekilde için bir mekanizma sağlar. Mod kaydı, veri yoluna olaraq beslenen ayrı veri birimlerinin sayısı olan patlama uzunluğunu belirler. Kayıt ayrıca programcının bir okuma talebinin alınması ile veri aktarımının başlaması arasındaki gecikmeyi ayarlamasına olanak tanır.

SDRAM, kelime işlem, elektronik tablolar ve multimedya gibi uygulamalarda olduğu gibi büyük veri bloklarını sıralı olarak aktarırken en iyi performansı gösterir.

Şekil 5.13 SDRAM işleminin bir örneğini gösterecektir. Bu durumda, burst uzunluğu 4 ve gecikme süresi 2'dir. Burst okuma komutu, saatin yükselen kenarında RAS ve WE'yi yüksek tutarken CS ve CAS'ı düşük tutarak başlatılır. Adres girişleri seri okuma için başlangıç sütun adresini belirler ve mod kaydı seri okuma türünü (sıralı veya interleave) ve seri okuma uzunluğunu (1, 2, 4, 8, tam sayfa) ayarlar. Komutun ilk hücreden gelen verinin çıkışlarında görünmesine kadar olan gecikme, mod kaydında ayarlanan CAS gecikmesinin değerine eşittir.



## DDR SDRAM

SDRAM asenkron RAM üzerinde önemli bir gelişme olmasına rağmen, hala elde edilebilecek I/O veri hızını gereksiz yere sınırlayan eksiklikleri vardır. Bu eksiklikleri gidermek için SDRAM'in çift veri oranlı DRAM (DDR DRAM) olarak adlandırılan daha yeni bir versiyonu veri oranını önemli ölçüde artıran çeşitli özellikler sunmaktadır. DDR DRAM, Electronic Industries Alliance'ın yarı iletken mühendisliği standardizasyon organı olan JEDEC Solid State Technology Association tarafından geliştirilmiştir. Çok sayıda şirket, masaüstü bilgisayarlarda ve sunucularda yaygın olarak kullanılan DDR yongaları üretmektedir.

DDR üç şekilde daha yüksek veri hızlarına ulaşır. Birincisi, veri aktarımı saatin sadece yükselen kenarı yerine hem yükselen hem de düşen kenarı ile senkronize edilir. Bu veri hızını iki katına çıkarır; dolayısıyla *çift veri hızı* terimi kullanılır. İkinci olarak, DDR aktarım hızını artırmak için veri yolunda daha yüksek saat hızı kullanır. Üçüncü olarak, daha sonra açıklanacağı gibi bir tamponlama şeması kullanılır.

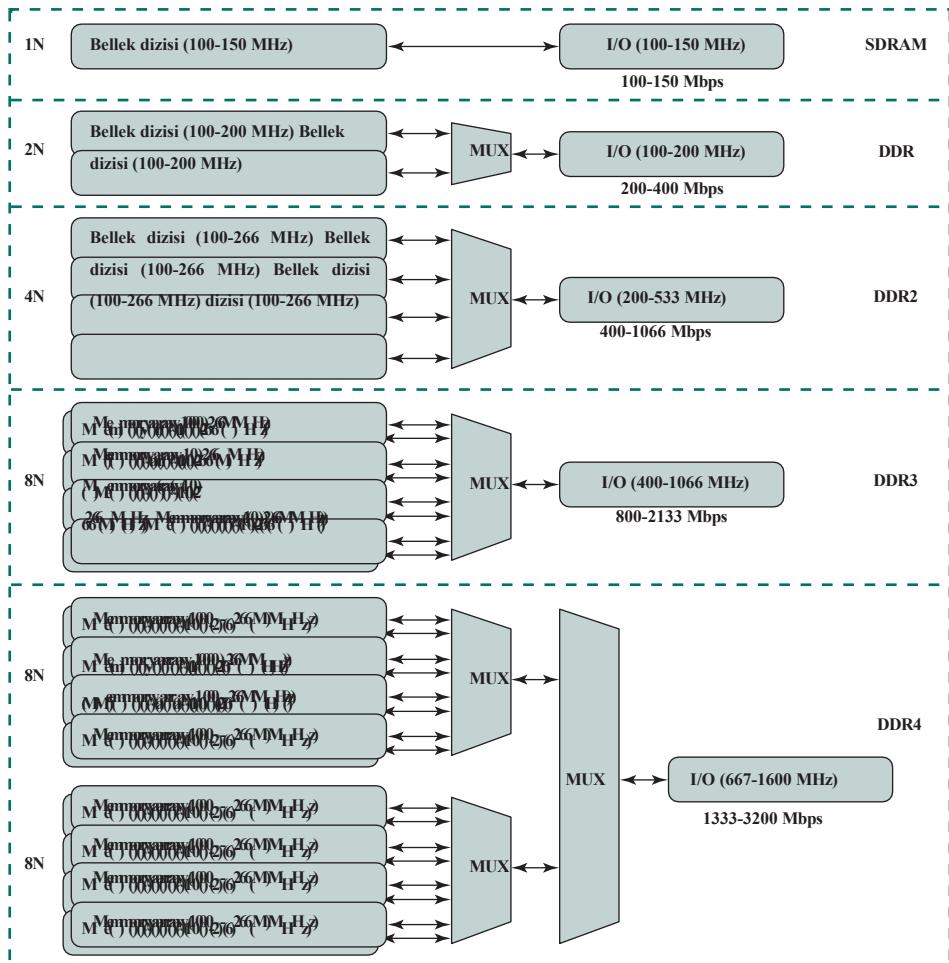
JEDEC şimdije kadar DDR teknolojisinin dört neslini tanımlamıştır (Tablo 5.4). İlk DDR versiyonu 2 bitlik bir prefetch buffer kullanır. Prefetch tamponu SDRAM yongası üzerinde bulunan bir bellek önbelleğidir. SDRAM yongasının veri yoluna yerleştirilecek bitleri mümkün olduğunda hızlı bir şekilde önceden konumlandırmamasını sağlar. DDR I/O veri yolu, bellek yongasıyla aynı saat hızını kullanır, ancak döngü başına iki bit işleyebildiğinden, saat hızının iki katı bir veri hızına ulaşır. 2 bitlik prefetch tamponu SDRAM yongasının I/O veri yoluna ayak uydurmasını sağlar.

Prefetch tamponunun çalışmasını anlamak için ona bir kelime transferi açısından bakmamız gereklidir. Prefetch tampon boyutu, DDR belleklerle her sütn komutu gerçekleştirildiğinde kaç kelime verinin getirileceğini (birden fazla SDRAM yongası üzerinden) belirler. DRAM'in çekirdeği arayüzden çok daha yavaş olduğu için aradaki fark, bilgiye paralel olarak erişip daha sonra bir çoklayıcı (MUX) aracılığıyla arayüzden seri hale getirilerek kapatılır. Böylece, DDR iki kelimeyi önceden alır, yanı her okuma ya da yazma işlemi yapıldığında, iki kelime veri üzerinde gerçekleştirilir ve toplam iki ardışık işlem için her iki saat kenarında bir saat döngüsü boyunca SDRAM'den dışarı ya da SDRAM'e patlar. Sonuç olarak, DDR I/O arayüzü SDRAM çekirdeğinden iki kat daha hızlıdır.

Her yeni SDRAM nesli çok daha yüksek kapasite ile sonuçlanır da SDRAM'in çekirdek hızı nesilden nesile önemli ölçüde değişmemiştir. SDRAM saat hızındaki oldukça mütevazi artışların sağladığından daha yüksek veri hızları elde etmek için JEDEC tampon boyutunu artırmıştır. DDR2 için, kelimelerin paralel olarak aktarılmasına olanak tanıyan ve etkin veri hızını 4 kat artıran 4 bitlik bir tampon kullanılmıştır. DDR3 için 8 bir tampon kullanılmış ve 8 kat hızlandırma elde edilmiştir (Şekil 5.14).

**Tablo 5.4** DDR Özellikleri

|                                        | DDR1    | DDR2     | DDR3     | DDR4      |
|----------------------------------------|---------|----------|----------|-----------|
| Prefetch arabelleği (bit)              | 2       | 4        | 8        | 8         |
| Gerilim seviyesi (V)                   | 2.5     | 1.8      | 1.5      | 1.2       |
| Ön taraf veri yolu veri hızları (Mbps) | 200-400 | 400-1066 | 800-2133 | 2133-4266 |



Şekil 5.14 DDR Nesilleri

Prefetch'in dezavantajı, SDRAM'ler için minimum burst uzunluğunu etkin bir şekilde belirlemesidir. Örneğin, DDR3'ün sekiz kelimelek prefetch'i ile dört kelimelek verimli bir burst uzunluğuna sahip olmak çok zordur. Buna göre, JEDEC tasarımcıları DDR4 için tampon boyutunu 16 bite çıkarmayı değil, bunun yerine bir **banka grubu** kavramını tanıtmayı seçmiştir [ALLA13]. Bank grupları, bir sütun döngüsünün bir bank grubu içinde tamamlanmasına izin verecek şekilde ayrı entibağlardır, ancak bu sütun döngüsü başka bir bank grubunda olanları etkilemez. Böylece, iki banka grubunda paralel olarak sekiz adet prefetch çalışabilir. Bu düzenleme, prefetch tampon boyutunu DDR3 ile aynı tutarken, prefetch daha büyümüş gibi performansı artırır.

Şekil 5.14'te iki grupta bir yapılandırma gösterilmektedir. DDR4 ile 4 adede kadar banka grubu kullanılabilir.

## 5.4 FLAŞ BELLEK

Yarı iletken belleğin bir başka biçimde flash bellektir. Flash bellek hem dahili bellek hem de harici bellek uygulamaları için kullanılır. Burada, teknik bir genel bakış sunacağız ve dahili bellek için kullanımına bakacağız.

İlk olarak 1980'lerin ortalarında tanıtılan flash bellek, hem maliyet hem de işlevsellik açısından EPROM ve EEPROM arasında yer alır. EEPROM gibi flash bellek de elektriksel silme teknolojisi kullanır. Tüm bir flash bellek bir ya da birkaç saniye içinde silinebilir ki bu EPROM'dan çok daha hızlıdır. Buna ek olarak, tüm bir çip yerine sadece bellek bloklarını silmek mümkündür. Flash bellek adını, mikroçipin bellek hücrelerinin bir bölümünün tek bir işlemle veya "flash" olarak silineceği şekilde organize edilmesinden alır. Ancak, flash bellek bayt düzeyinde silme sağlamaz. EPROM gibi, flash bellek de bit başına yalnızca bir transistör kullanır ve böylece EPROM'un yüksek yoğunluğuna (EEPROM ile karşılaştırıldığında) ulaşır.

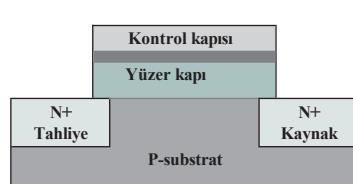
### Operasyon

Şekil 5.15 bir flaş belleğin temel çalışmasını göstermektedir. Karşılaştırma için, Şekil 5.15a bir transistörün çalışmasını göstermektedir. Transistörler yarı iletkenlerin faydalanan, böylece kapiya uygulanan küçük bir voltaj kaynak ve drenaj arasındaki büyük bir akımın akışını kontrol için kullanılabilir.

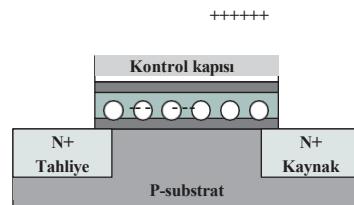
Bir flash bellek hücresinde, transistöre ikinci bir geçit ince bir oksit tabakası ile kaplandığı için yüzen geçit olarak adlandırılır - eklenir. Başlangıçta, yüzen geçit transistörün çalışmasına müdahale etmez (Şekil 5.15b). Bu durumda, hücrenin ikili 1'i temsil ettiği kabul edilir. Oksit tabakası boyunca büyük bir voltaj uygulamak elektronların tünelden geçmesine ve güç kesilse bile kalacakları yüzen geçitte sıkışmasına neden olur (Şekil 5.15c). Bu durumda hücrenin ikili 0'i temsil ettiği kabul edilir. Transistörün çalışıp çalışmadığını test etmek için harici devre kullanılarak hücrenin durumu okunabilir. Ters yönde büyük bir voltaj uygulamak elektronları yüzen geçitten uzaklaştırır ve ikili 1 durumuna geri döner.



(a) Transistör yapısı



(b) Flash bellek bir durumda



(c) Sıfır durumundaki Flash bellek hücresi

**Şekil 5.15** Flash Bellek Çalışması

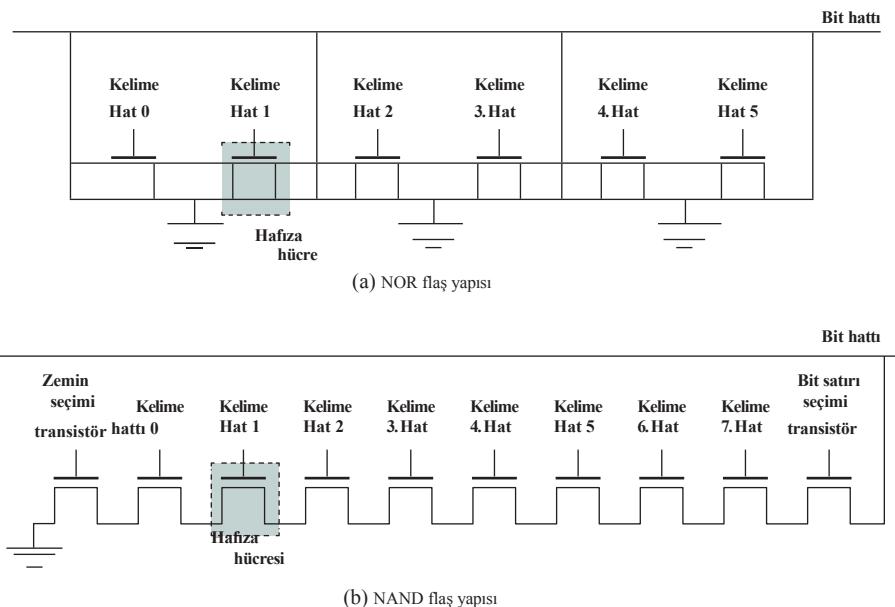
Flaş belleğin önemli bir özelliği kalıcı bellek olmasıdır, yani belleğe güç uygulanmadığında verileri korur. Bu nedenle, ikincil (harici) depolama için ve bilgisayarlardaki rastgele erişimli belleğe alternatif olarak kullanılmıştır.

### NOR ve NAND Flash Bellek

NOR ve NAND olarak adlandırılan iki farklı flash bellek türü vardır (Şekil 5.16). **NOR flash bellekte**, temel erişim birimi *bellek hücresi* olarak adlandırılan bir bittir. NOR flaş bellekteki hücreler bit hatalarına paralel olarak bağlanır, böylece her hücre ayrı ayrı okunabilir/yazılabilir/silinebilir. Cihazın herhangi bir bellek hücresi ilgili kelime hattı tarafından açılırsa, bit hattı düşük olur. Bu, işlev olarak bir NOR mantık kapısına benzer.<sup>2</sup>

**NAND flash bellek**, seri olarak 16 veya 32 transistörlü transistör dizileri halinde düzenlenmiştir. Bit hattı, yalnızca ilgili kelime hatlarındaki tüm transistörler açıksa düşük gider. Bu, işlev olarak bir NAND mantık kapısına benzer.

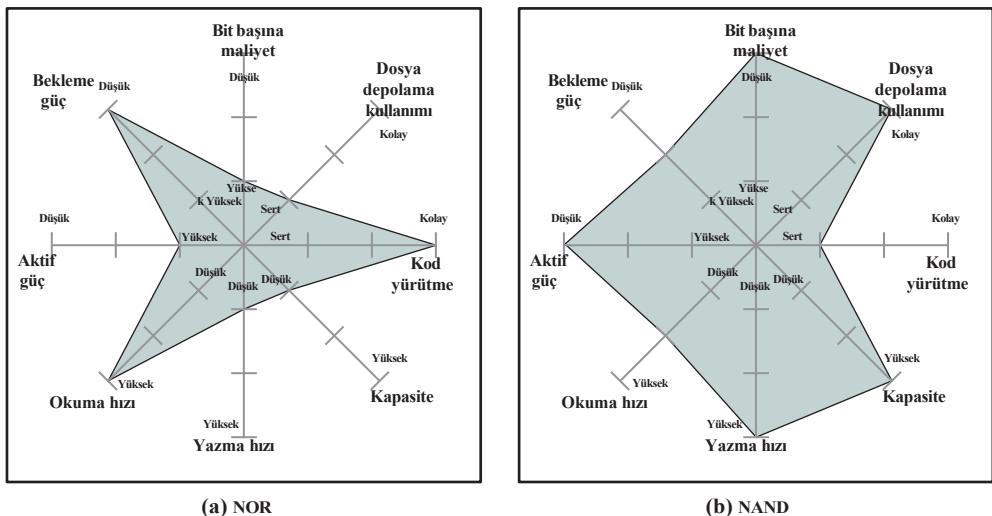
NOR ve NAND'in çeşitli özelliklerinin belirli niceliksel değerleri yıldan yıla değişse de, iki tür arasındaki göreceli farklar sabit kalmıştır. Bu farklılıklar Şekil 5.17'de gösterilen Kiviat grafikleri<sup>3</sup> ile faydalı bir şekilde gösterilmiştir.



**Şekil 5.16** Flash Bellek Yapıları

<sup>2</sup>Şekil 5.2b'deki ve ile ilişkili daireler sinyal olumsuzlaşmasını gösterir.

<sup>3</sup> Kiviat grafiği, sistemleri çoklu değişkenler boyunca karşılaştırmak için resimsel bir araç sağlar [MORR74]. Değişkenler daire içinde eşit aralıklarla çizgiler halinde düzenlenir ve her çizgi dairenin merkezinden çevresine doğru gider. Belirli bir sistem her çizgi üzerinde bir nokta ile tanımlanır; çevreye ne kadar yakınsa değer o kadar iyidir. Noktalar, o sistemin karakteristiği olan bir şekil elde etmek için birleştirilir. Şeklin kapsadığı alan ne kadar fazla sistem o kadar "iyi" demektir.



**Şekil 5.17** Flash Bellek için Kiviat Grafikleri

NOR flash bellek yüksek hızlı rastgele erişim sağlar. Belirli konumlara veri okuyabilir ve yazabilir ve tek bir bayta referans verebilir ve geri alabilir. NAND küçük bloklar halinde okur ve yazar. NAND, NOR'dan daha yüksek bit yoğunluğu ve daha yüksek yazma hızı sağlar. NAND flaş rastgele erişimli bir harici adres veriyolu sağlamaz, bu nedenle veriler blok bazında okunmalıdır (sayfa erişimi olarak da bilinir), burada her blok yüzlerce ila binlerce bit içerir.

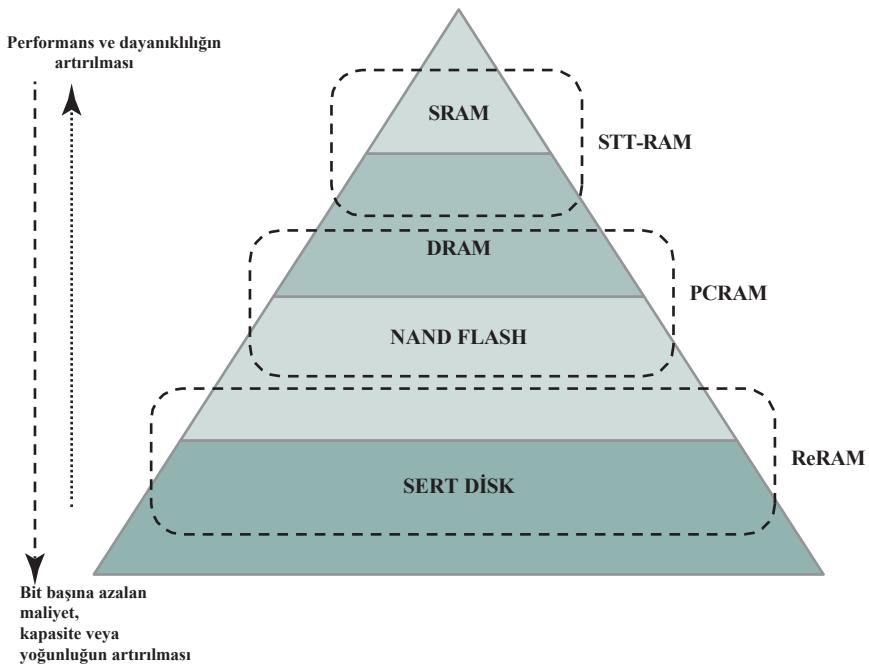
Gömülü sistemlerde dahili bellek için geleneksel olarak NOR flash bellek tercih edilmiştir. NAND bellek bazı ilerlemeler kaydetmiştir, ancak NOR dahili için baskın teknoloji devam etmektedir. Program kodu miktarının nispeten az olduğu ve belirli miktarda uygulama verisinin değişmediği mikro denetleyiciler için idealdir. Örneğin, Şekil 1.16'daki flash bellek NOR bellektir.

NAND bellek, USB flash sürücüler, bellek kartları (dijital kameralarda, MP3 çalarlarda vb.) ve katı hal diskleri (SSD'ler) olarak bilinen harici bellekler için daha uygundur. SSD'leri Bölüm 6'da ele alacağız.

## **5.5 YENİ DEVİRİMSEL OLMAYAN KATIL DURUMLU MEMOrY TEKNOLOJİLERİ**

Geleneksel bellek hiyerarşisi üç seviyeden oluşmaktadır (Şekil 5.18):

- **Statik RAM (SRAM):** SRAM hızlı erişim süresi sağlar, ancak en pahalı ve en az yoğun (bit yoğunluğu) olanıdır. SRAM ön bellek için uygundur.
  - **Dinamik RAM (DRAM):** SRAM'den daha ucuz, daha yoğun ve daha yavaş olan DRAM, geleneksel olarak çip dışı ana bellek tercihi olmuştur.
  - **Sabit disk:** Manyetik disk, nispeten yavaş erişim süreleri ile çok yüksek bit yoğunluğu ve bit başına çok düşük maliyet sağlar. Bellek hiyerarşisinin bir parçası olarak harici depolama için geleneksel seçimidir.



Şekil 5.18 Bellek Hiyerarşisi içinde Uçuçu Olmayan RAM

Bu karışımı, gördüğümüz gibi, flash bellek de eklenmiştir. Flash bellek, geleneksel belleklere göre uçucu olmama avantajına sahiptir. NOR flaş, gömülü sistemlerde programları ve statik uygulama verilerini depolamak için en uygun olanıdır, NAND flaş ise DRAM ve sabit diskler arasında ara özelliklere sahiptir.

Zaman içinde, bu teknolojilerin her biri ölçeklendirmede gelişmeler kaydetti: daha yüksek bit yoğunluğu, daha yüksek hız, daha düşük güç tüketimi ve daha düşük maliyet. Ancak, yarı iletken bellekler için bu gelişme hızını sürdürmek giderek zorlaşmaktadır [ITRS14].

Son zamanlarda, flaş belleğin ötesinde ölçeklendirmeye devam eden yeni uçucu olmayan yarı iletken bellek formlarının geliştirilmesinde atımlar olmuştur. En umut verici teknolojiler spin-transfer tork RAM (STT-RAM), faz değişimli RAM (PCRAM) ve dirençli RAM'dir (ReRAM) ([ITRS14], [GOER12]). Bunların hepsi hacimli üretimdedir. Ancak, NAND Flash ve dereceye kadar NOR Flash hala uygulamalara hakim olduğundan, bu yeni bellekler özel uygulamalarda kullanılmış ve ana akım yüksek yoğunluklu uçucu olmayan bellek olma vaadini henüz yerine getirememiştir. Bu durumun önmüzdeki birkaç yıl içinde değişmesi muhtemeldir.

Şekil 5.18, bu üç teknolojinin bellek hiyerarşisine nasıl uyacağı göstermektedir.

## STT-RAM

STT-RAM, ucuçuluk içermeyen, hızlı yazma/okuma hızına ( $6\text{--}10\text{ ns}$ ) ve yüksek programlama dayanıklılığına ( $7\text{--}10^{15}$  döngü) ve sıfır bekleme gücüne sahip yeni bir **manyetik RAM (MRAM)** türüdür [KULT13]. MRAM'in depolama kapasitesi veya programlanabilirliği, ince bir tünelleme dielektriginin iki ferromanyetik katman arasına sıkıştırıldığı manyetik tünelleme bağlantısından (MTJ) kaynaklanmaktadır. Bir ferromanyetik katman (sabitlenmiş veya referans katman) mıknatıslanması sabitlenmiş olacak şekilde tasarlanırken, diğer katmanın (serbest) mıknatıslanması bir yazma olayı ile çevrilebilir. Serbest katman ve sabitlenmiş mıknatıslamaları paralel (anti-paralel) ise bir MTJ düşük (yüksek) dirence sahiptir. Birinci nesil MRAM tasarımda, serbest katmanın mıknatıslanması akım kaynaklı manyetik alan tarafından değiştirilir. STT-RAM'de, polarizasyon-akım-indüklenmiş *mıknatıslanma anahtarlaması* adı verilen yeni bir yazma mekanizması geliştirilmiştir. STT-RAM için, serbest katmanın mıknatıslanması doğrudan elektrik akımı tarafından çevrilir. Bir MTJ direnç durumunu değiştirmek için gereken akım MTJ hücre alıyla orantılı olduğundan, STT-RAM'in birinci nesil MRAM'den daha iyi bir ölçeklendirme özelliğine sahip olduğuna inanılmaktadır. Şekil 5.19a genel konfigürasyonu göstermektedir.

STT-RAM, önbellek ya da ana bellek için iyi bir adaydır.

## PCRAM

**Faz değişimi RAM (PCRAM)**, geniş bir teknik literatüre ([RAOU09], [ZHOU09], [LEE10]) sahip en olgun veya yeni teknolojilerdir.

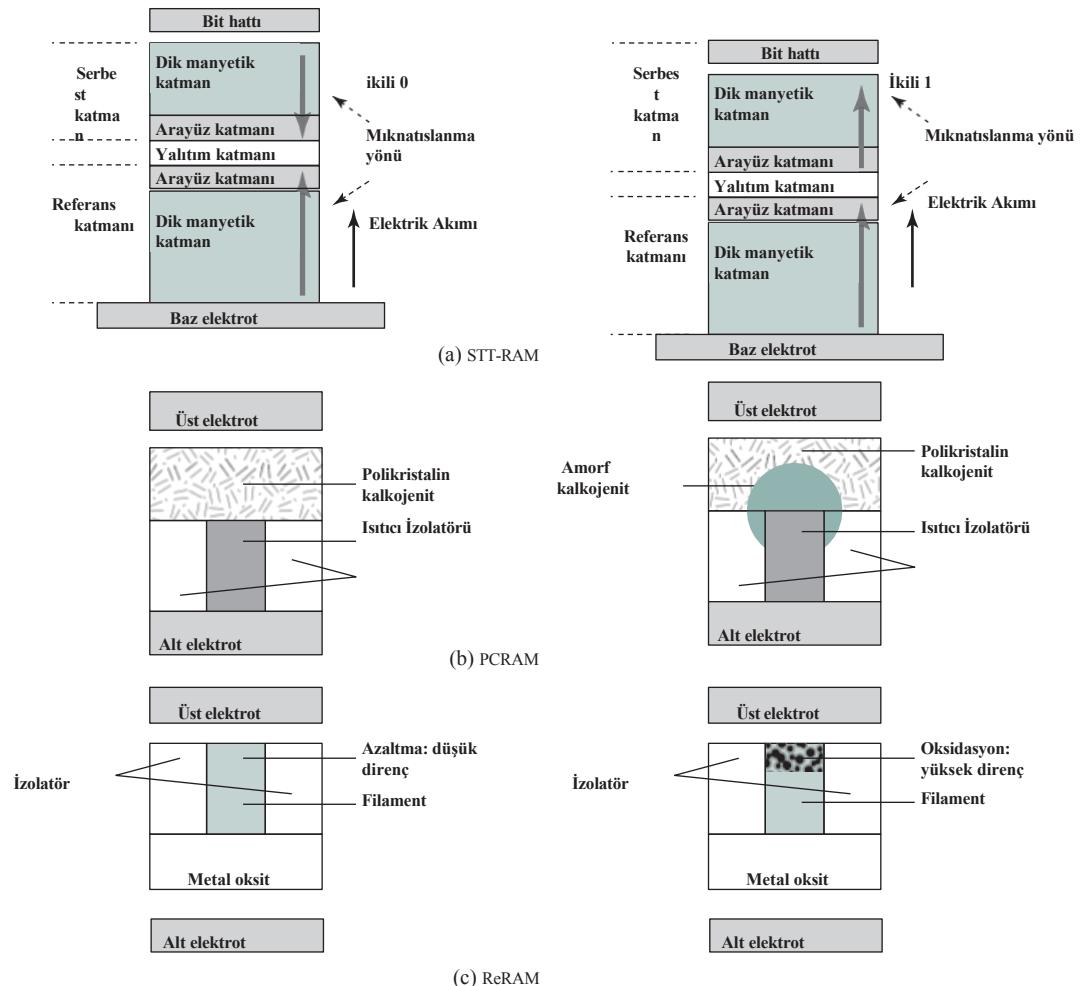
PCRAM teknolojisi, optik depolama ortamlarında (kompakt diskler ve dijital versa- tile diskler) yaygın olarak kullanınlara benzer bir kalkojenit alışım malzemesine dayanmaktadır. Veri depolama kapasitesi, kalkojenit bazlı malzemenin amorf (yüksek dirençli) ve kristalin (düşük dirençli) fazları arasındaki direnç farklılıklarından elde edilir. SET işleminde faz değişim malzemesi, hücrenin önemli bir kısmını kristalleşme sıcaklığının üzerinde isıtın bir elektrik darbesi uygulanarak kristalleştirilir. RESET işleminde, daha büyük bir elektrik akımı uygulanır ve ardından malzemeyi eritmek ve ardından söndürmek için aniden kesilir ve amorf durumda bırakılır. Şekil 5.19b genel konfigürasyonu göstermektedir.

PCRAM, ana bellek için DRAM'in yerini almak veya onu tamamlamak için iyi bir adaydır.

## ReRAM

ReRAM (RRAM olarak da bilinir) doğrudan yük depolamak yerine direnç yaratarak çalışır. Bir malzemeye elektrik akımı uygulanarak o malzemenin direnci değiştirilir. Direnç durumu daha sonra ölçülebilir ve sonuç olarak bir 1 veya 0 okunur. Bugüne kadar ReRAM üzerinde yapılan çalışmaların çoğu, uygun malzemelerin bulunmasına ve hücrelerin direnç durumunun ölçülmesine odaklanmıştır. ReRAM tasarımları düşük voltajlıdır, dayanıklılık flaş bellekten çok daha üstündür ve hücreler çok daha küçüktür - en azından teoride. Şekil 5.19c bir ReRAM konfigürasyonunu göstermektedir.

ReRAM, hem ikincil depolamayı hem de ana belleği değiştirmek veya tamamlamak için iyi bir adaydır.



Şekil 5.19 Uçucu Olmayan RAM Teknolojileri

## 5.6 KEy TERMIER, kEVEP SORuLARI vE PROBLEMLER

### Anahtar Terimler

|                                                                      |                                                                                                     |                                                                                         |
|----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| banka grubu<br>çift veri hızlı DRAM (DDR DRAM)<br>dinamik RAM (DRAM) | elektrikle silinebilir<br>programlanabilir ROM (EEPROM)<br>silinebilir programlanabilir ROM (EPROM) | hata düzeltme kodu (ECC) hata<br>düzelte<br>flash bellek<br>Hamming kodu<br>sabit arıza |
|----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|

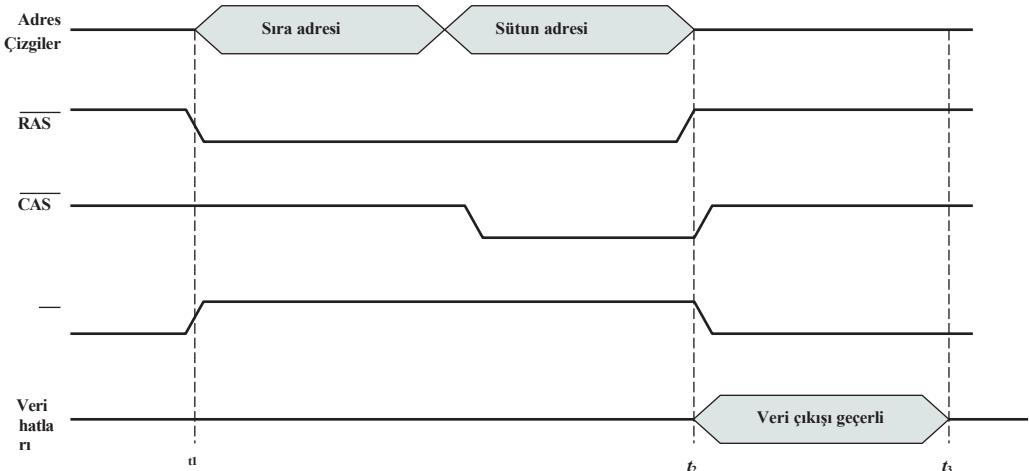
|                                                                                                                                                                                       |                                                                                                                                                                                                      |                                                                                                                              |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| manyetik RAM (MRAM)<br>NAND flaş bellek uçucu<br>olmayan bellek NOR flaş bellek<br>faz değişimi RAM<br>(PCRAM)<br>programlanabilir ROM<br>(PROM)<br>rastgele erişimli bellek<br>(RAM) | çoğulukla okunur bellek<br>salt okunur bellek<br>(ROM)<br>dirençli RAM (ReRAM) yarı<br>iletken bellek tek hata düzeltici<br>(SEC) kodu<br>tek hata düzeltmeli, çift hata<br>algılamalı (SEC-DED) kod | yumuşak hata<br>spin-transfer tork RAM (STT-<br>RAM)<br>statik RAM (SRAM) senkron DRAM<br>(SDRAM)<br>sendrom uçucu<br>bellek |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|

## İnceleme Soruları

- 5.1** Yarı iletken belleğin temel özelliklerini nelerdir?
- 5.2** *Rastgele erişimli bellek* teriminin iki yorumu nedir?
- 5.3** Uygulama açısından DRAM ve SRAM arasındaki fark nedir?
- 5.4** Hız, boyut ve maliyet gibi özellikler açısından DRAM ve SRAM arasındaki fark nedir?
- 5.5** Neden bir RAM türünün analog, diğerinin dijital olarak kabul edildiğini açıklayınız.
- 5.6** ROM için bazı uygulamalar nelerdir?
- 5.7** EPROM, EEPROM ve flash bellek arasındaki farklar nelerdir?
- 5.8** Şekil 5.4b'deki her bir pinin işlevini açıklayınız.
- 5.9** Eşlik biti nedir?
- 5.10** Hamming kodu için sendrom nasıl yorumlanır?
- 5.11** SDRAM'in sıradan DRAM'den farklı nedir?
- 5.12** DDR RAM nedir?
- 5.13** NAND ve NOR flaş bellek arasındaki fark nedir?
- 5.14** Üç yeni uçucu olmayan katı hal bellek teknolojisini listeleyiniz ve kısaca tanımlayınız.

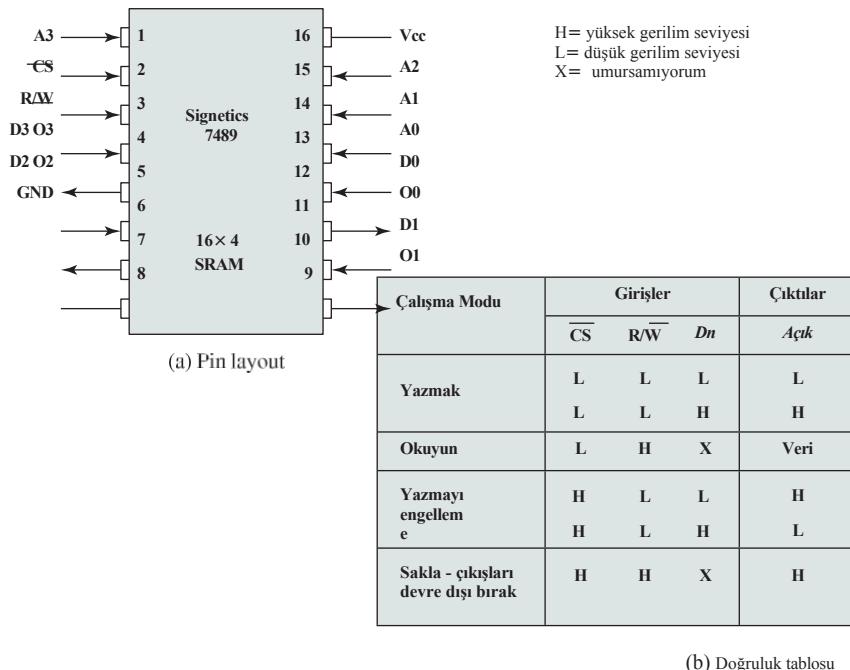
## Problemler

- 5.1** RAM'lerin geleneksel olarak çip başına sadece bir bit olarak organize edilmesine karşın ROM'ların genellikle çip başına birden fazla bit ile organize edilmesinin nedenlerini öneriniz.
- 5.2** Ms başına 64 kez yenileme döngüsü verilmesi gereken bir dinamik RAM düşünün. Her bir yenileme işlemi 150 ns; bir bellek döngüsü 250 gerektirir. Belleğin toplam çalışma süresinin yüzde kaçını yenilemelere ayrılmalıdır?
- 5.3** Şekil 5.20'de bir veri yolu üzerinden DRAM okuma işlemi için basitleştirilmiş bir zamanlama diyagramı gösterilmektedir. Erişim süresinin  $t_{(1)}$ den  $t_{(2)}$ ye kadar sürdüğü kabul edilir. Daha sonra  $t_2$ 'den  $t_3$ 'e kadar süren bir yeniden şarj süresi vardır ve bu süre boyunca DRAM yongalarının işlemecinin tekrar erişebilmesi için yeniden şarj edilmesi gereklidir.
  - a.** Erişim süresinin 60 ns ve yeniden şarj süresinin 40 ns olduğunu varsayıyın. Bellek döngü süresi nedir? Bu DRAM'in 1 bitlik bir çıkışı varsayarak sürdürilebileceği maksimum veri hızı nedir?
  - b.** Bu çipleri kullanarak 32 bit genişliğinde bir bellek sistemi oluşturmak hangi veri aktarım hızını verir?
- 5.4** Şekil 5.6, dört adet 256-Kbyte çiplen oluşan bir grup temelinde 1 MB depolayabilen bir çip modülünün nasıl oluşturulacağını göstermektedir. Diyelim ki bu yonga modülü, kelime boyutu 1 bayt olan tek bir 1 MB yonga olarak paketlendi. Sekiz adet 1-MB yonga kullanarak 8-MB'lık bir bilgisayar belleğinin nasıl oluşturulacağına dair üst düzey bir yonga diyagramı verin. Diyagramınızda adres hatlarını ve adres hatlarının ne için kullanıldığını gösterdiğinizden emin olun.

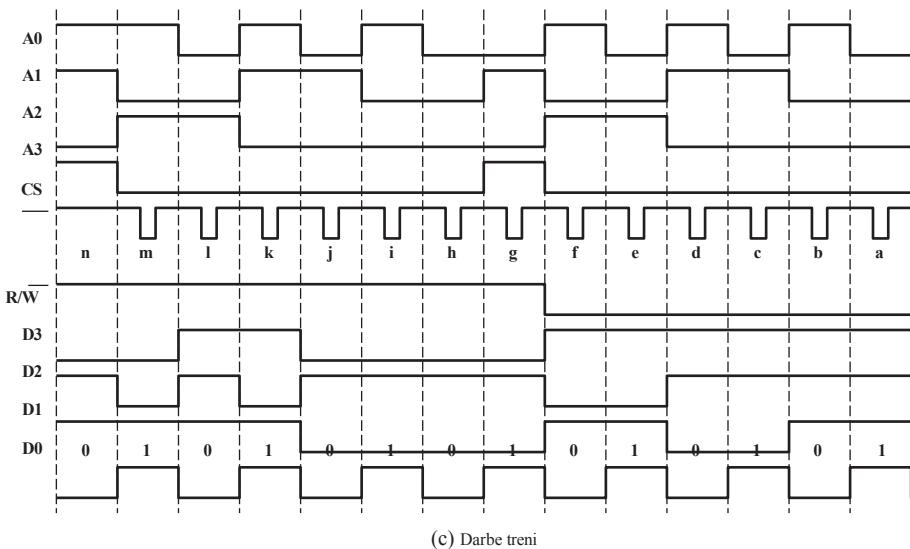


Sekil 5.20 Basitleştirilmiş DRAM Okuma Zamanlaması

- 5.5** Sistem veriyolu üzerinden DRAM belleğe bağlı tipik bir Intel 8086 tabanlı sistemde, bir okuma işlemi için RAS, Adres Etkinleştirme sinyalinin son kenarı tarafından etkinleştirilir (Ek C'deki Şekil C.1). Ancak, yayılma ve diğer gecikmeler nedeniyle RAS, Address Enable düşük seviyeye döndükten 50 ns sonrasına kadar etkinleşmez. İkincisinin  $T_{(1)}$  durumunun ikinci yarısının ortasında (Şekil C.1'dekinden biraz daha erken) gerçekleştiğini varsayıyın. Veriler işlemci tarafından  $T_{(3)}$ 'ün sonunda okunur. Ancak işlemciye zamanında sunulması için verilerin bellek tarafından 60 ns önce sağlanması gereklidir. Bu aralık, veri yolları (bellekler işlemciye) boyunca yayılma gecikmelerini ve işlemci veri tutma süresi gereksinimlerini hesaba katar. Saat hızının 10 MHz olduğunu varsayıyalım.
- Bekleme durumları eklenmeyecekse DRAM'ler ne kadar hızlı (erişim süresi) olmalıdır?
  - DRAM'lerin erişim süresi 150 ns ise, bellek okuma işlemi başına kaç bekleme durumu eklememiz gereklidir?
- 5.6** Belirli bir mikrobilgisayarın belleği  $64K * 1$  DRAM'lerden oluşturulmuştur. Veri sayfasına göre, DRAM'in hücre dizisi 256 satır halinde düzenlenmiştir. Her satır en az 4 ms'de bir yenilenmelidir. Belleği kesinlikle periyodik olarak yenilememizi varsayıyalım.
- Birbirini izleyen yenileme istekleri arasındaki süre nedir?
  - Ne kadar uzun bir yenileme adres sayacına ihtiyacımız var?
- 5.7** Şekil 5.21'de illüSTRAM'lardan biri olan  $16 * 4$  Signetics 7489 yongası gösterilmektedir, yonga 16 adet 4 bitlik sözcük depolamaktadır.
- Şekil 5.21'de gösterilen her CS giriş darbesi için çipin çalışma modunu listeleyin.
  - N darbesinden sonra 0 ile 6 arasındaki sözcük konumlarının bellek içeriklerini listeleyin.
  - h'den m'ye kadar olan giriş darbeleri için çıkış veri uçlarının durumu nedir?
- 5.8**  $64 * 1$  bit boyutunda SRAM çipleri kullanarak toplam kapasitesi 8192 bit olan 16 bitlik bir bellek tasarlayınız. Bu belleği en düşük adres alanına atmak için gerekli tüm giriş ve çıkış sinyallerini gösteren bellek kartındaki çiplerin dizi yapılandırmasını verin. Tasarım hem bayt hem de 16 bitlik kelime erişimlerine izin vermelidir.
- 5.9** Elektronik bileşenlerin ariza oranları için yaygın bir ölçü birimi, milyar cihaz saatı başına ariza oranı olarak ifade edilen **Arızasızlık** birimidir (FIT). İyi bilinen ancak daha az kullanılan bir başka ölçü de **arızalar arasındaki ortalama süredir (MTBF)** ve bu da belirli bir bileşenin arızalanana kadar ortalama çalışma süresidir.  $256K * 1$  DRAM'lere sahip 16 bitlik bir mikroişlemcinin 1 MB belleğini düşünün. Her DRAM için 2000 FITS varsayıarak MTBF'sini hesaplayın.



(b) Doğruluk tablosu

**Şekil 5.21** Signetics 7489 SRAM

- 5.10** Şekil 5.10'da gösterilen Hamming kodu için, bir veri biti yerine bir kontrol biti hatalı olduğunda ne olacağını gösteriniz?
- 5.11** Bellekte saklanan 8 bitlik bir veri sözcüğünün 11000010 olduğunu varsayılmı. Hamming algoritmasını kullanarak, veri sözcüğü ile birlikte bellekteki kontrol bitlerinin saklanacağını belirleyin. Cevabınızı nasıl aldıgınızı gösteriniz.
- 5.12** 8 bitlik 00111001 sözcüğü için, onunla birlikte saklanan kontrol bitleri 0111 olacaktır. Sözcük bellekten okunduğunda, kontrol bitlerinin 1101 olarak hesaplandığını varsayılmı. Bellekten okunan veri sözcüğü nedir?
- 5.13** Hamming hata düzeltme kodu 1024 bitlik bir veri sözcüğündeki tek bitlik hataları tespit etmek için kullanılırsa kaç kontrol bitine ihtiyaç duyulur?

- 194 BÖLÜM 15/ DAKİ İKİ BELİREK** sözcüğü için bir SEC kodu geliştirin. 010100000111001 veri sözcüğü için kodu oluşturun. Kodun veri biti 5'teki bir hatayı doğru şekilde tanımlayacağını gösterin.

# 6

## HARICI BELLEK

### 6.1 Manyetik Disk

Manyetik Okuma ve Yazma Mekanizmaları  
Veri Organizasyonu ve Biçimlendirme Fiziksel Özellikler  
Disk Performans Parametreleri

### 6.2 RAID

RAID Seviye 0  
RAID Seviye 1  
RAID Seviye 2  
RAID Seviye 3  
RAID Seviye 4  
RAID Seviye 5  
RAID Seviye 6

### 6.3 Katı Hal Sürücüler

HDD ile Karşılaştırıldığında  
SSD SSD Organizasyonu  
Pratik Konular

### 6.4 Optik Bellek

Kompakt Disk  
Dijital Çok Yönlü Disk  
Yüksek Çözünürlüklü Optik Diskler

### 6.5 Manyetik Bant

### 6.6 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler



### ÖĞRENİM HEDEFLERİ

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- Manyetik disklerin temel özelliklerini anlamak.
- **Manyetik disk** erişimi ile ilgili performans sorunlarını anlamak.
- **RAID** kavramını açıklayın ve çeşitli seviyeleri tanımlayın. ■ Sabit disk sürücülerile katı disk sürücülerini karşılaştırın. ■ **Flaş belleğin** çalışmasını genel hatlarıyla açıklayın.
- Farklı optik disk depolama ortamları arasındaki farkları anlamak.
- **Manyetik bant** depolama teknolojisine genel bir bakış sunun.

Bu bölümde bir dizi harici bellek aygıtı ve sistemi incelenmektedir. En önemli aygit olan manyetik disk ile başlıyoruz. Manyetik diskler neredeyse tüm bilgisayar sistemlerinde harici belleğin temelini oluşturmaktadır. Bir sonraki bölümde daha yüksek performans elde etmek için disk dizilerinin kullanımı incelenmekte ve özellikle RAID (Redundant Array of Independent Disks) olarak bilinen sistem ailesi ele alınmaktadır. Birçok bilgisayar sisteminin giderek daha önemli bir bileşeni haline gelen katı hal diski daha sonra ele alınmaktadır. Daha sonra harici **optik bellek** incelenmiştir. Son olarak, manyetik bant açıklanmaktadır.

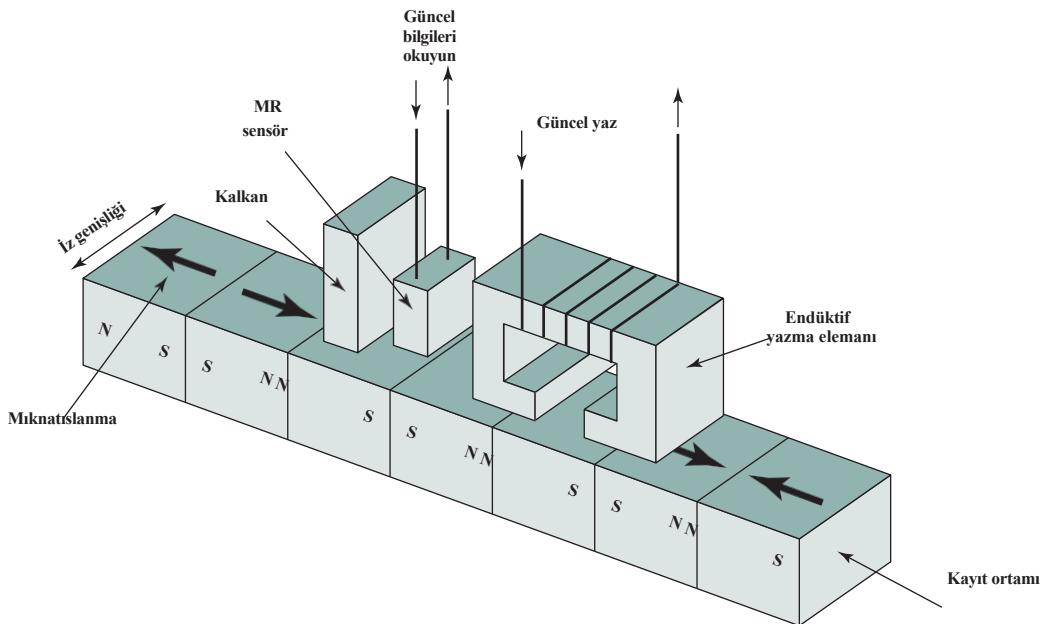
## 6.1 MANYETİK DİSK

Bir disk, mıknatıslanabilir bir ile kaplanmış, **alt tabaka** olarak adlandırılan manyetik olmayan malzemeden yapılmış dairesel bir **plakadır**. Geleneksel olarak, alt tabaka alüminyum veya alüminyum alaşımı bir malzemedir. Daha yakın zamanlarda cam alt tabakalar da kullanılmaya başlanmıştır. Cam alt tabakanın aşağıdakiler de dahil olmak üzere bir dizi faydası vardır:

- Disk güvenilirliğini artırmak için manyetik film yüzeyinin homojenliğinde iyileştirme.
- Okuma-yazma hatalarını azaltmaya yardımcı olmak için genel yüzey kusurlarında önemli bir azalma.
- Daha düşük sinek yüksekliklerini destekleme yeteneği (daha sonra açıklanacaktır).
- Disk dinamığını azaltmak için daha iyi sertlik.
- Şok ve hasara karşı daha fazla dayanma kabiliyeti.

### Manyetik Okuma ve Yazma Mekanizmaları

Veriler, **kafa** adı verilen iletken bir bobin aracılığıyla diske kaydedilir ve daha sonra diskten alınır; birçok sistemde, bir okuma **kafası** ve bir yazma kafası olmak üzere iki vardır. Bir okuma ya da yazma işlemi sırasında, disk plakası altında dönerken kafa sabit durur. Yazma mekanizması, bir bobinden akan elektriğin bir manyetik alan oluşturduğu gerçeğinden yararlanır. Elektrik darbeleri yazma kafasına gönderilir ve ortaya çıkan manyetik desenler, pozitif ve negatif akımlar için farklı desenlerle aşağıdaki yüzeye kaydedilir. Yazma kafasının kendisi kolayca mıknatıslanabilir malzemeden yapılmıştır.



**Şekil 6.1** Endüktif Yazma/Manyetorezistif Okuma Başlığı

malzeme ve bir kenarı boyunca bir boşluk ve diğer kenarı boyunca birkaç tur iletken tel bulunan dikdörtgen bir çörek şeklindedir (Şekil 6.1). Teldeki bir elektrik akımı boşluk boyunca bir manyetik alan indükler ve bu da kayıt ortamının küçük alanını manyetize eder. Akımın yönünü tersine çevirmek kayıt ortamındaki mıknatışlanmanın yönünü tersine çevirir.

Geleneksel okuma mekanizması, bir bobine göre hareket eden bir manyetik alanın bobinde bir elektrik akımı ürettiği gerçeginden yararlanır. Diskin yüzeyi kafanın altında döndüğünde, önceden kaydedilmiş olanla aynı kutupta bir akım üretir. Bu durumda okuma için kullanılan kafanın yapısı yazma için kullanılan aynıdır ve bu nedenle her ikisi için de aynı kafa kullanılabilir. Bu tür tek kafalar disket sistemlerinde eski sert disk sistemlerinde kullanılır.

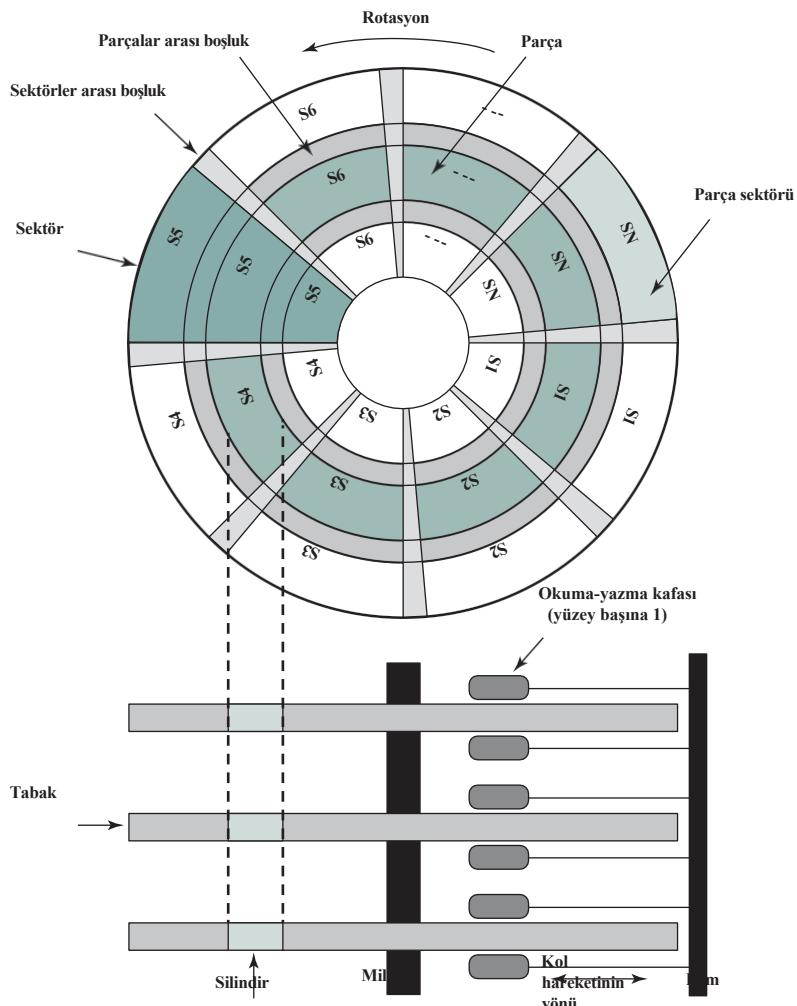
Çağdaş riyit disk sistemleri farklı bir okuma mekanizması kullanır ve kolaylık sağlamak için yazma kafasına yakın konumlandırılmış ayrı bir okuma kafası gerektirir. Okuma kafası kısmen korumalı bir **manyetorezistif (MR)** sensörden oluşur. MR matrisi, altında hareket eden ortamın mıknatışlanma yönüne bağlı olan bir elektrik direncine sahiptir. MR sensöründen bir akım geçirilerek direnç değişiklikleri voltaj sinyalleri olarak algılanır. MR tasarımları daha yüksek frekanslı çalışmaya izin verir, bu da daha yüksek depolama yoğunlukları ve çalışma hızları anlamına gelir.

### Veri Organizasyonu ve Biçimlendirme

Kafa, altında dönen plakanın bir kısmından okuma veya bir kısmına yazma kapasitesine sahip nispeten küçük bir cihazdır. Bu, verilerin plaka üzerinde *iz* adı verilen eşmerkezli bir dizi halka halinde düzenlenmesine yol açar. Her iz kafa ile aynı genişliğindedir. Yüzey başına binlerce iz vardır.

Şekil 6.2'de bu veri düzeni gösterilmektedir. Bitişik izler, izler **arası boşluklarla** ayrılmıştır. Bu, kafanın yanlış hizalanmasından ya da manyetik alanların karışmasından kaynaklanan hataları önler ya da en azından en aza indirir. Veriler diske ve disktken **sektörler** halinde aktarılır. Tipik olarak parça başına yüzlerce sektör vardır ve bunlar sabit ya da değişken uzunlukta olabilir. Çoğu çağdaş sistemde, 512 bayt neredeyse evrensel sektör boyutu olmak üzere sabit uzunlukta sektörler kullanılır. Sisteme mantıksız hassasiyet gereksinimleri yüklemekten kaçınmak için, bitişik sektörler sektörler arası boşluklarla ayrılr.

Dönen bir diskin merkezine yakın bir bit, sabit bir noktadan (okuma-yazma kafası gibi) dışarıdaki bir bitten daha yavaş geçer. Bu nedenle, kafanın tüm bitleri aynı hızda okuyabilmesi için hızdaki değişikliği telafi etmenin bir yolu bulunmalıdır. Bu, kaydedilen bilgi bitleri arasında değişken bir aralık tanımlayarak yapılabilir.

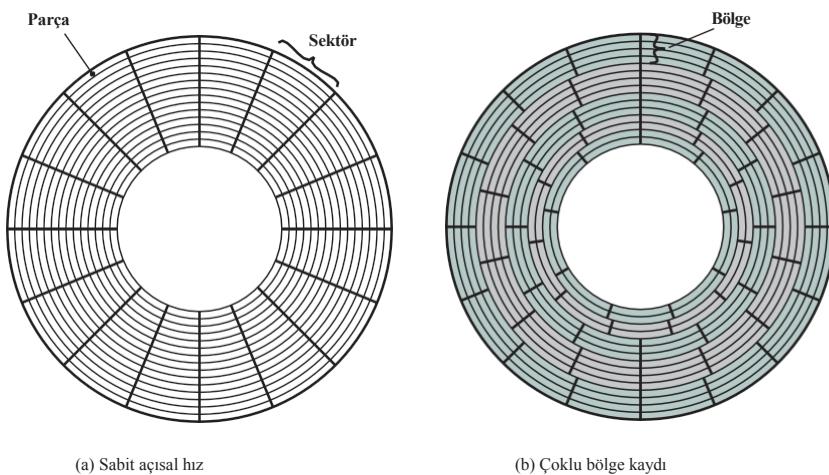


Şekil 6.2 Disk Veri Düzeni

Disk üzerindeki konumlar, en dıştaki izler daha büyük aralıklı sektör'lere sahip olacak şekilde. Bilgi daha sonra disk'i **sabit açısal hız (CAV)** olarak bilinen **sabit bir hızda** döndürerek aynı **hızda** taranabilir. Şekil 6.3a CAV kullanan bir disk'in düzenini göstermektedir. Disk bir dizi pasta biçimli sektör'e ve bir dizi eşmerkezli yola bölünmüştür. CAV kullanmanın avantajı, tek tek veri bloklarının doğrudan iz ve sektör'e göre adreslenebilmesidir. Kafayı mevcut konumundan belirli bir adrese taşımak için, kafanın belirli bir yola kısa bir hareketi ve uygun sektörün kafanın altında dönmesi için kısa birbekleme yeterlidir. CAV'ın dezavantajı, uzun dış izlerde saklanabilecek veri miktarının kısa iç izlerde saklanabilecek veri miktarıyla aynı olmasıdır.

Doğrusal inç başına bit cinsinden **yoğunluk** en dıştaki izden en içteki ize doğru arttığinden, basit bir CAV sistemindeki disk depolama kapasitesi en içteki izde elde edilebilecek maksimum kayıt yoğunluğu ile sınırlıdır. Depolama kapasitesini en üst düzeye çıkarmak için her iz üzerinde aynı doğrusal bit yoğunluğuna sahip olmak tercih edilir. Bu da kabul edilemeyecek kadar karmaşık bir sistem gerektirecektir. Modern sabit disk sistemleri, iz başına eşit bit yoğunluğuna yaklaşan, çoklu bölge kaydı (MZR) olarak bilinen ve yüzeyin bir dizi eş merkezli bölgeye (tipik olarak 16) bölündüğü daha basit bir teknik kullanır. Her bölge, tipik olarak binlerle ifade edilen sayıda bitistik iz içerir. Bir bölge içinde iz başına düşen bit sayısı sabittir. Merkezden uzak bölgeler yakın bölgelere göre daha fazla bit (daha fazla sektör) içerir. Bölgeler, diskin tüm izlerinde satır başı bit yoğunluğu yaklaşık olarak aynı olacak şekilde tanımlanır. MZR, biraz daha karmaşık devre pahasına daha fazla genel depolama kapasitesi sağlar. Disk kafası bir bölgeden diğerine geçerken, tek tek bitlerin uzunluğu (iz boyunca) değişir ve bu da okuma ve yazma zamanlamasında değişikliği neden olur.

Şekil 6.3b basitleştirilmiş bir MZR düzenidir ve 15 hat 5 bölge halinde düzenlenmiştir. En içteki iki bölgenin her birinde ikişer hat vardır ve her hat dokuz sektör'e sahiptir; bir sonraki bölgede her biri 12 sektör'e sahip 3 hat vardır; ve en dıştaki 2 bölgenin her birinde 4 hat vardır ve her hat 16 sektör'e sahiptir.



**Şekil 6.3** Disk Yerleşim Yöntemlerinin Karşılaştırılması

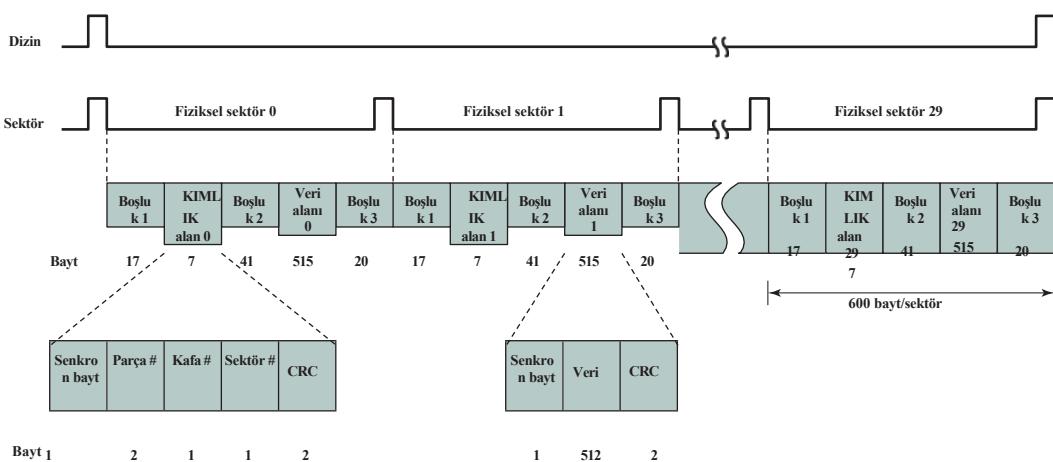
Bir pist içindeki sektör konumlarını belirlemek için bazı araçlara ihtiyaç vardır. Açıkçası, iz üzerinde bir baĢlangıç noktası ve her sektörün baĢlangıç ve bitiĢini tanımlamanın bir yolu olmalıdır. Bu gereklıklar diske kaydedilen kontrol verileri aracılığıyla yerine getirilir. Böylece disk, yalnızca disk sürücüsü tarafından kullanılan ve kullanıcı tarafından erişilememeyen bazı ekstra verilerle biçimlendirilir.

Şekil 6.4'te bir disk biçimlendirme örneği gösterilmektedir. Bu durumda, her iz her biri 600 baylık 30 sabit uzunlukta sektör içerir. Her sektör 512 bayt veri ve disk denetleyicisi için yararlı kontrol bilgileri içerir. ID alanı, belirli bir sektörün yerini belirlemek için kullanılan benzersiz bir tanımlayıcı ya da adrestir. SYNCH baytı, alanın başlangıcını sınırlayan özel bir bit modelidir. Parça numarası bir yüzey üzerindeki bir parçayı tanımlar. Kafa numarası bir kafayı tanımlar, çünkü bu diskin birden fazla yüzeyi vardır (ileride açıklanacaktır). ID ve veri alanlarının her biri bir hata tespit kodu içerir.

### Fiziksel Özellikler

Tablo 6.1 çeşitli manyetik disk türlerini birbirinden ayıran başlıca özellikleri listelemektedir. İlk olarak kafa plakalarının radyal yönüne göre sabit ya da hareketli olabilir. **Sabit kafalı** bir **diskte**, parça başına bir okuma-yazma **kafası** vardır. Tüm kafalar tüm izler boyunca uzanan sert bir kol üzerine monte edilmiştir; bu tür sistemler günümüzde nadirdir. **Hareketli kafalı** bir **diskte** yalnızca bir okuma-yazma **kafası** vardır. Yine, kafa bir kol üzerine monte edilmiştir. Kafanın herhangi bir iz üzerinde konumlandırılabilmesi gerektiğinden, kol bu amaçla uzatılabilir veya geri çekilebilir.

Diskin kendisi, kol, diski döndüren bir mil ve ikili verilerin giriĢ ve çıkışı için gerekli elektronik aksamdan oluĢan bir disk sürücüsüne monte edilir. **Çıkarılamayan** bir disk disk sürücüsüne kalıcı olarak monte edilir; kişisel bir bilgisayardaydı sabit disk çıkarılamayan bir diskdir. **Çıkarılabilir** bir **disk çıkarılabilir** ve başka bir diskle değiştirilebilir. İkinci türün avantajı, sınırlı sayıda disk sistemiyle sınırsız miktarda verinin kullanılabilir olmasıdır. Ayrıca, böyle bir disk bir bilgisayar sisteminden diğerine taşınabilir. Disketler ve ZIP kartuş diskleri çıkarılabilir disklere önektiler.



**Şekil 6.4** Winchester Disk Formatı (Seagate ST506)

**Tablo 6.1** Disk Sistemlerinin Fiziksel Özellikleri

|                                        |                                 |
|----------------------------------------|---------------------------------|
| <b>Kafa Hareketi</b>                   | <b>Tabaklar</b>                 |
| Sabit kafa (ray başına bir adet)       | Tekli tabak Çoklu               |
| Hareketli kafa (yüzey başına bir adet) | tabak                           |
| <b>Disk Taşınabilirliği</b>            | <b>Kafa Mekanizması</b>         |
| Çıkarılmaz disk Çıkarılabilir disk     | Kontak (disket)                 |
| <b>Taraflar</b>                        | Sabit aralık                    |
| Tek taraflı Çift taraflı               | Aerodinamik boşluk (Winchester) |

Çoğu disk için, mıknatıslanabilir kaplama plakanın her iki tarafına da uygulanır ve bu da **çift taraflı** olarak adlandırılır. Bazı daha ucuz disk sistemleri **tek taraflı** diskler kullanır.

Bazı disk sürücülerinde bir inçten daha az aralıklarla dikey olarak istiflenmiş **birden fazla plaka** bulunur. Birden fazla kol bulunur (Şekil 6.2). Çoklu plakalı disklerde, her plaka yüzeyi için bir okuma-yazma kafası olmak üzere hareketli bir kafa kullanılır. Tüm kafalar diskin merkezinden aynı uzaklıktta olacak ve birlikte hareket edecek şekilde mekanik olarak sabitlenmiştir. Böylece, herhangi bir zamanda, tüm kafalar diskin merkezinden eşit uzaklıkta olan izler üzerinde konumlandırılır. Plaka üzerinde aynı göreceli konumda bulunan tüm izlerin oluşturduğu kümeye **silindir** olarak adlandırılır. Bu durum Şekil 6.2'de gösterilmiştir.

Son olarak, kafa mekanizması disklerin üç tipte sınıflandırılmasını sağlar. Geleneksel olarak, okuma-yazma kafası bir hava boşluğununa izin verecek şekilde plakanın üzerinde sabit bir mesafeye yerleştirilmiştir. Diğer uça ise okuma ya da yazma işlemi sırasında ortamla fiziksel temas'a geçen bir kafa mekanizması yer alır. Bu mekanizma, küçük, esnek bir plaka olan ve en ucuz disk türü olan **diskette** kullanılır.

Üçüncü disk türünü anlamak için veri yoğunluğu ile hava boşluğunun boyutu arasındaki ilişki hakkında yorum yapmamız gereklidir. Kafa, düzinin yazmak ve okumak için yeterli büyülüklükte bir elektromanyetik alan üretmeli ya da algılamalıdır. Kafa ne kadar dar olursa, işlevini yerine getirebilmesi için plaka yüzeyine o kadar yakın olması gereklidir. Daha dar bir kafa, daha dar izler ve dolayısıyla daha yüksek veri yoğunluğu anlamına gelir ki bu da arzu edilen bir durumdur. Ancak, kafa diske ne kadar yakın olursa, kirlilik veya kusurlardan kaynaklanan hata riski de o kadar artar. Teknolojiyi daha da ileri götürmek için Winchester diskleri geliştirilmiştir. Winchester kafaları, neredeyse hiç kirletici içermeyen kapalı sürücü tertibatlarında kullanılır. Geleneksel sert disk kafalarına kıyasla diskin yüzeyine daha yakın çalışacak şekilde tasarlanılmışlardır, böylece daha fazla veri yoğunluğu sağlarlar. Kafa aslında disk hareketsizken plakanın yüzeyinde hafifçe duran aerodinamik bir folyodur. Dönen bir disk tarafından üretilen hava basıncı, folyonun yüzeyin üzerine çıkışmasını sağlamak için yeterlidir. Ortaya çıkan temassız sistem, geleneksel sert disk kafalarına göre plakanın yüzeyine daha yakın çalışan daha dar kafalar kullanmak üzere tasarlanabilir.

Tablo 6.2 tipik çağdaş yüksek performanslı diskler için disk parametrelerini vermektedir.

**Tablo 6.2** Tipik Sabit Disk Sürücü Parametreleri

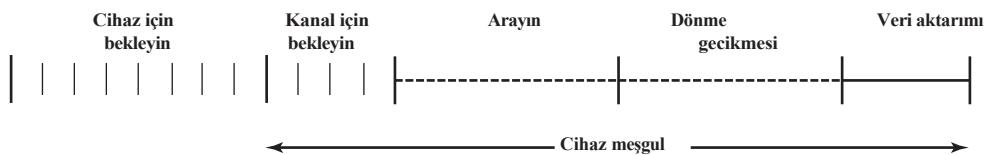
| Özellikler                                     | Seagate Enterprise | Seagate Barracuda XT | Seagate Cheetah NS                      | Seagate Dizüstü Bilgisayar HDD'si |
|------------------------------------------------|--------------------|----------------------|-----------------------------------------|-----------------------------------|
| Uygulama                                       | Kurumsal           | Masaüstü             | Ağa bağlı depolama, uygulama sunucuları | Dizüstü Bilgisayar                |
| Kapasite                                       | 6 TB               | 3 TB                 | 600 GB                                  | 2 TB                              |
| Ortalama arama süresi                          | 4,16 ms            | N/A                  | 3,9 ms okuma<br>4,2 ms yazma            | 13 ms                             |
| İş mili hızı                                   | 7200 rpm           | 7200 rpm             | 10,075 rpm                              | 5400 rpm                          |
| Ortalama gecikme süresi                        | 4,16 ms            | 4,16 ms              | 2.98                                    | 5,6 ms                            |
| Maksimum sürekli aktarım hızı                  | 216 MB/sn          | 149 MB/sn            | 97 MB/sn                                | 300 MB/sn                         |
| Sektör başına bayt                             | 512/4096           | 512                  | 512                                     | 4096                              |
| Silindir başına iz (plaka yüzeylerinin sayısı) | 8                  | 10                   | 8                                       | 4                                 |
| Önbellek                                       | 128 MB             | 64 MB                | 16 MB                                   | 8 MB                              |

### Disk Performans Parametreleri

Disk G/Ç işleminin gerçek ayrıntıları bilgisayar sistemine, işletim sistemine ve G/Ç kanalının ve disk denetleyici donanımının yapısına bağlıdır. Disk G/Ç aktarımının genel bir zamanlama diyagramı Şekil 6.5'te gösterilmektedir.

Disk sürücüsü çalışırken, disk sabit hızda dönmektedir. Okumak veya yazmak için kafanın istenen izde ve bu izde istenen sektörün başında konumlandırılması gereklidir. İz seçimi, hareketli kafalı bir sistemde kafanın hareket ettirilmesini veya sabit kafalı bir sistemde bir kafanın elektronik olarak seçilmesini içerir. Hareketli kafalı bir sistemde, kafanın iz üzerinde konumlandırılması için geçen süre **arama süresi** olarak bilinir. Her iki durumda da, parça seçildikten sonra, disk denetleyicisi uygun sektör dönerken kafaya aynı hizaya gelene kadar bekler. Sektörün başlangıcının kafaya ulaşması için geçen süre **dönme gecikmesi** veya **dönme gecikmesi** olarak bilinir. Varsa arama süresi ile dönme gecikmesinin toplamı **erişim süresine** eşittir; bu da okuma veya yazma için pozisyon alma süresidir. Kafa pozisyonunu aldıktan sonra, okuma veya yazma işlemi sektör kafanın altında hareket ederken gerçekleştirilir; bu işlemin veri aktarımı kısımidır; aktarım için gereken süre aktarım **süresidir**.

Erişim süresi ve aktarım süresine ek olarak, normalde bir disk G/Ç işlemiyle ilişkili birkaç kuyruk gecikmesi vardır. Bir işlem bir G/Ç işlemi gerçekleştirdiğinde

**Şekil 6.5** Bir Disk G/Ç Aktarımının Zamanlaması

isteğinde bulunursa, önce cihazın kullanılabilir olması için bir kuyrukta beklemesi gerekir. zaman, aygit işleme atanır. Aygit tek bir G/C kanalını veya dizi G/C kanalını diğer disk sürücülerile paylaşıyorsa, kanalın kullanılabilir olması için ek bir bekleme olabilir. Bu noktada, disk erişimini başlatmak için arama gerçekleştirilir.

Sunuculara yönelik bazı üst düzey sistemlerde rotasyonel konum algılama (RPS) olarak bilinen bir teknik kullanılmaktadır. Bu şu şekilde çalışır: Arama komutu, kanal diğer G/C işlemlerini gerçekleştirmek için serbest bırakılır. Arama tamamlandığında, cihaz verilerin kafanın altında ne zaman-doneceğini belirler. Bu sektör kafaya yaklaşıırken, cihaz ana bilgisayara giden iletişim yolunu yeniden kurmaya çalışır. Kontrol ünitesi veya kanal başka bir G/C ile meşgulse, yeniden bağlanma girişimi başarısız olur ve cihazın yeniden bağlanması deneyebilmesi için bir tam tur dönmesi gerekir, buna RPS ıskası denir. Bu, Şekil 6.5'teki zaman çizelgesine eklenmesi gereken ekstra bir gecikme unsurudur.

**ARAMA SÜRESİ** Arama süresi, disk kolunu gerekli yola hareket ettirmek için gereken süredir. Bunun tespit edilmesi zor bir miktar olduğu ortaya çıkmıştır. Arama süresi iki temel bileşenden oluşur: ilk başlatma süresi ve erişim kolu hızlandıktan sonra geçilmesi gereken izleri geçmek için geçen süre. Ne yazık ki, geçiş süresi iz sayısının doğrusal bir fonksiyonu olmayıp, bir yerleşme süresi (kafanın hedef iz üzerinde konumlandırılmışından iz tanımlamasının onaylanması kadar geçen süre) içerir. Daha küçük ve daha hafif disk bileşenleri sayesinde büyük bir gelişme sağlanmıştır. Birkaç yıl önce tipik bir diskin çapı 14 inç (36 cm) iken günümüzde en yaygın boyut 3,5 inç (8,9 cm) olup kolun kat etmesi gereken mesafeyi azaltmaktadır.

seyahat. Çağdaş sabit disklerde tipik bir ortalama arama süresi 10 ms'nin altındadır.

**DÖNÜŞ GECİKMESİ** Disketler, disketler dışında, 3600 rpm'den (dijital kameralar gibi taşınabilir aygıtlar için) bu yazı itibarıyle 20.000 rpm'ye kadar değişen hızlarda dönerler; bu son hızda, 3 ms'de bir devir vardır. Dolayısıyla, ortalama olarak, dönme gecikmesi 1,5 ms olacaktır.

**AKTARIM SÜRESİ** Diske veya diskten aktarım süresi, aGağıdaki Gekilde diskin dönüG hızına bağlıdır:

$$T = \frac{b}{rN}$$

nerede

$T$ = aktarım süresi

$b$ = aktarılacak bayt sayısı

$N$ = bir iz üzerindeki bayt sayısı

$r$ = dönüş hızı, saniye başına devir cinsinden

Böylece toplam ortalama okuma veya yazma süresi  $T_{toplam}$  olarak ifade edilebilir

$$T_{toplam} = T_s + \frac{1}{2r} + \frac{b}{rN} \quad (6.1)$$

Burada  $T_s$  ortalama arama süresidir. Bölgelere ayrılmış bir sürücüde parça başına bayt sayısının değişken olduğunu ve bu durumun hesaplamayı zorlaştırdığını unutmayın.<sup>1</sup>

<sup>1</sup> Önceki iki denklemi Denklem (4.1) ile karşılaştırın.

**ZAMANLAMA KARŞILAŞTIRMASI** Yukarıdaki parametreler tanımlandıktan sonra, ortalama değerlere güvenmenin tehlikesini gösteren iki farklı I/O işlemine bakalım. Reklamı yapılan ortalama arama süresi 4 ms, dönüş hızı 15.000 rpm ve 512 baytlık sektörleri ve parça başına 500 sektörü olan bir disk düşünün. Toplam 1,28 Mbyte'luk 2500 sektörden oluşan bir dosyayı okumak istediğimizi varsayalım. Aktarım için toplam süreyi tahmin etmek istiyoruz.

Öncelikle, dosyanın disk üzerinde mümkün olduğunda kompakt bir şekilde saklandığını varsayalım. , dosya 5 bitişik iz üzerindeki tüm sektörleri kaplar ( $5 \text{ iz} * 500 \text{ sektör/iz} = 2500 \text{ sektör}$ ). Bu sıralı düzenlemeye olarak bilinir. Şimdi, ilk parçayı okuma zamanı aşağıdaki gibidir:

|                          |              |
|--------------------------|--------------|
| Ortalama arama           | 4 ms         |
| Ortalama dönme gecikmesi | 2 ms         |
| 500 sektör okuma         | 4 ms         |
|                          | <u>10 ms</u> |

Kalan parçaların artık arama süresi olmadan okunabileğini varsayalım. Yani, G/C işlemi diskiten gelen akışa ayak uydurabilir. O zaman, en fazla, kalan dört parça için dönme gecikmesiyle uğraşmamız gereklidir. Böylece birbirini izleyen her parça  $2 + 4 = 6 \text{ ms}'de$  okunur. Tüm dosyayı okumak için,

$$\text{Toplam süre} = 10 + (4 * 6) = 34 \text{ ms} = 0,034 \text{ saniye}$$

Şimdi aynı veriyi sıralı erişim yerine rastgele erişim kullanarak okumak gereken süreyi hesaplayalım; yani sektörlerde erişimler disk üzerinde rastgele dağıtıılır. Her bir sektör için

|                 |                 |         |
|-----------------|-----------------|---------|
| Ortalama arama  | 4               | ms      |
| Dönme gecikmesi | 2 ms            | Okuma 1 |
| sektör          | 0.008 ms        |         |
|                 | <u>6.008 ms</u> |         |

$$\text{Toplam süre} = 2500 * 6.008 = 15,020 \text{ ms} = 15.02 \text{ saniye}$$

Sektörlerin diskiten okunma sırasının I/O performansı üzerinde çok büyük bir etkisi olduğu açıktır. Birden fazla sektörün veya yazıldığı dosya erişimi durumunda, veri sektörlerinin dağıtım şekli üzerinde bir miktar kontrolümüz vardır. Ancak, bir dosya erişimi durumunda bile, çoklu programlama ortamında, aynı disk için yarışan G/C istekleri olacaklardır. Bu nedenle, disk G/C performansının diske tamamen rastgele erişimle elde edilen performansa kıyasla nasıl geliştirilebileceğini incelemek faydalı olacaktır. Bu, işletim sisteminin alanı olan ve bu kitabın kapsamı dışında kalan disk zamanlama algoritmalarının değerlendirilmesine yol açar (bir tartışma için [STAL15]'e bakın).



RAID Simülörü

## 6.2 RAID

Daha önce de belirtildiği gibi, ikincil depolama performansındaki gelişme oranı, iGlemci ve ana bellek performansındaki gelişme oranından çok daha düşük olmuştur. Bu uyumsuzluk, disk depolama sistemimi genel bilgisayar sistemi performansının iyileştirilmesinde belki de ana endişe odağı haline getirmiştir.

Bilgisayar performansının diğer alanlarında olduğu gibi, disk depolama tasarımcıları da tek bir bileğen bu kadar ileri gidilebiliyorsa, birden fazla paralel bileğen kullanılarak performansta ek kazanımlar elde edilebileceğini kabul etmektedir. Disk depolama söz konusu olduğunda, bu durum bağımsız ve paralel olarak çalışan disk dizilerinin geliştirilmesine yol açar. Birden fazla diskle, gerekli veriler ayrı disklerde bulunduğu sürece ayrı G/C istekleri paralel olarak işlenebilir. Ayrıca, erişilecek veri bloğu birden fazla diske dağıtılmışsa, tek bir G/C isteği paralel olarak yürütülebilir.

Birden fazla diskin kullanılmasıyla, verilerin düzenlenebileceği ve güvenilirliği artırmak için yedeklilikin eklenebileceği çok çeşitli yollar vardır. Bu durum, çok sayıda platform ve işletim sisteminde kullanılabilen veritabanı şemalarının geliştirilmesini zorlaştıracaktır. Neyse ki endüstri, RAID (Redundant Array of Independent Disks) olarak bilinen çoklu disk veritabanı tasarımı için standartlaştırılmış bir şema üzerinde anlaşmaya varmıştır. RAID şeması sıfırdan altıya kadar yedi seviyeden<sup>2</sup> oluşur. Bu seviyeler hiyerarşik bir ilişki anlamına gelmez, ancak üç ortak özelliği paylaşan farklı tasarım mimarilerini belirtir:

- 1.** RAID, işletim sistemi tarafından tek bir mantıksal sürücü olarak görülen bir dizi fizikal disk sürücüsüdür.
- 2.** Veriler, daha sonra açıklanacak olan şeritleme olarak bilinen bir şemada bir dizinin fizikal sürücülerleri arasında dağıtılır.
- 3.** Yedek disk kapasitesi, bir disk arızası durumunda verilerin kurtarılabilirliğini garanti eden eşlik bilgilerini depolamak için kullanılır.

İkinci ve üçüncü özelliklerin ayrıntıları farklı RAID seviyeleri için farklılık gösterir. RAID 0 ve RAID 1 üçüncü özelliğini desteklemez.

*RAID* terimi ilk olarak Berkeley'deki Kaliforniya Üniversitesi'nde bir grup araştırmacı tarafından bir makalede ortaya atılmıştır [PATT88].<sup>3</sup> Makalede çeşitli RAID yapılandırmaları ve uygulamaları özetlenmiş ve halen kullanılan RAID seviyelerinin tanımları yapılmıştır. RAID stratejisi birden fazla disk sürücüsü kullanır ve verileri birden fazla verilere eşzamanlı erişim sağlayacak şekilde dağıtır, böylece I/O performansını iyileştirir ve kapasitede daha kolay artışlara izin verir.

---

<sup>2</sup> Bazı araştırmacılar ve bazı şirketler tarafından ilave seviyeler tanımlanmıştır, ancak bu bölümde açıklanan yedi seviye evrensel olarak kabul edilen seviyelerdir.

<sup>3</sup> Bu makalede RAID kısaltması Redundant Array of Inexpensive Disks ('Ucuz Disklerin Yedekli Dizisi) anlamına geliyor. *Ucuz* terimi, RAID dizisindeki küçük nispeten ucuz diskleri alternatif olan tek bir büyük pahali diskle (SLED) karşılaştırmak için kullanılmıştır. Hem RAID hem de RAID olmayan konfigürasyonlar için benzer disk teknolojisi kullanıldığından SLED asında geçmişi kalmıştır. Buna göre endüstri, RAID dizisinin önemli performans ve güvenilirlik kazanımları yarattığını vurgulamak için *bağımsız* terimini benimsemiştir.

RAID önerisinin benzersiz katkısı, yedeklilik ihtiyacını etkin bir şekilde ele almasıdır. Birden fazla kafanın ve aktüatörün aynı anda çalışmasına izin vermek daha yüksek G/C ve aktarım hızları elde edilmesini sağlasa da, birden fazla cihazın kullanılması arıza olasılığını artırır. Bu azalan güvenilirliği telafi etmek için RAID, bir disk arızası nedeniyle kaybolan verilerin kurtarılmasını sağlayan depolanmış eşlik bilgisini kullanır.

Şimdi RAID seviyelerinin her birini inceleyeceğiz. Tablo 6.3 yedi seviye için kabaca bir kılavuz sunmaktadır. Tabloda, G/C performansı hem veri aktarım kapasitesi ya da veri taşıma yeteneği hem de G/C istek hızı ya da G/C isteklerini karşılama yeteneği açısından gösterilmektedir, çünkü bu RAID seviyeleri bu iki ölçüme göre doğal olarak farklı performans göstermektedir. Her RAID seviyesinin güçlü noktası daha koyu gölgelerle vurgulanmıştır. Şekil 6.6, artıkklık olmadan dört disk gerektiren bir veri kapasitesini desteklemek için yedi RAID şemasının kullanımını göstermektedir. Şekiller kullanıcı verilerinin ve yedek verilerin düzenini vurgulamakta ve çeşitli seviyelerin göreceli depolama gereksinimlerini göstermektedir. Aşağıdaki tartışma boyunca bu şekillere atıfta bulunacağız. Açıklanan yedi RAID seviyesinden yalnızca dördü yaygın olarak kullanılmaktadır: RAID seviyeleri 0, 1, 5 ve 6.

## RAID Seviye 0

RAID seviye 0, RAID ailesinin gerçek bir üyesi değildir çünkü performansı artırmak için yedeklilik içermez. Bununla birlikte, performans ve kapasitenin birincil kaygıları olduğu ve düşük maliyetin geliştirilmiş güvenilirlikten daha önemli olduğu süper bilgisayarlardaki bazı uygulamalar gibi birkaç uygulama vardır.

RAID 0 için kullanıcı ve sistem verileri dizideki tüm disklere dağıtılr. Bunun tek bir büyük disk kullanımına göre önemli bir avantajı vardır: İki farklı veri bloğu için iki farklı G/C isteği bekliyorsa, istenen blokların farklı disklerde olma ihtimali yüksektir. Böylece, iki istek paralel olarak yayınlanabilir ve G/C kuyruk süresi azaltılabilir.

Ancak RAID 0, tüm RAID seviyelerinde olduğu gibi, verileri bir disk dizisine dağıtmadan ötesine gecer: Veriler mevcut diskler arasında *seritlenir*. Bu en iyi Şekil 6.7'ye bakılarak anlaşılabilir. Tüm kullanıcı ve sistem verileri mantıksal bir diskte depolanmış olarak görülür. Mantıksal disk şeritlere bölünmüştür; bu şeritler fiziksel bloklar, sektörler ya da başka bir birim olabilir. Şeritler, RAID dizisindeki ardışık fiziksel disklerle yuvarlak robin olarak eşleştirilir. Her bir dizi üyesine tam olarak bir şerit eşleyen, mantıksal olarak birbirine bağlı şeritler kümesi *şerit* olarak adlandırılır. Bir  $n$ -disk dizisinde, ilk  $n$  mantıksal şerit fiziksel olarak  $n$  diskin her birinde ilk şerit olarak saklanır ve ilk şeridi oluşturur; ikinci  $n$  şerit her diskte ikinci şerit olarak dağıtılır; ve bu böyle devam eder. Bu düzenin avantajı, tek bir G/C isteği mantıksal olarak bitişik birden fazla şeritten oluşuyorsa, bu istek için  $n$  adede kadar şeridin paralel olarak işlenebilmesi ve G/C aktarım süresinin büyük ölçüde azaltılmasıdır.

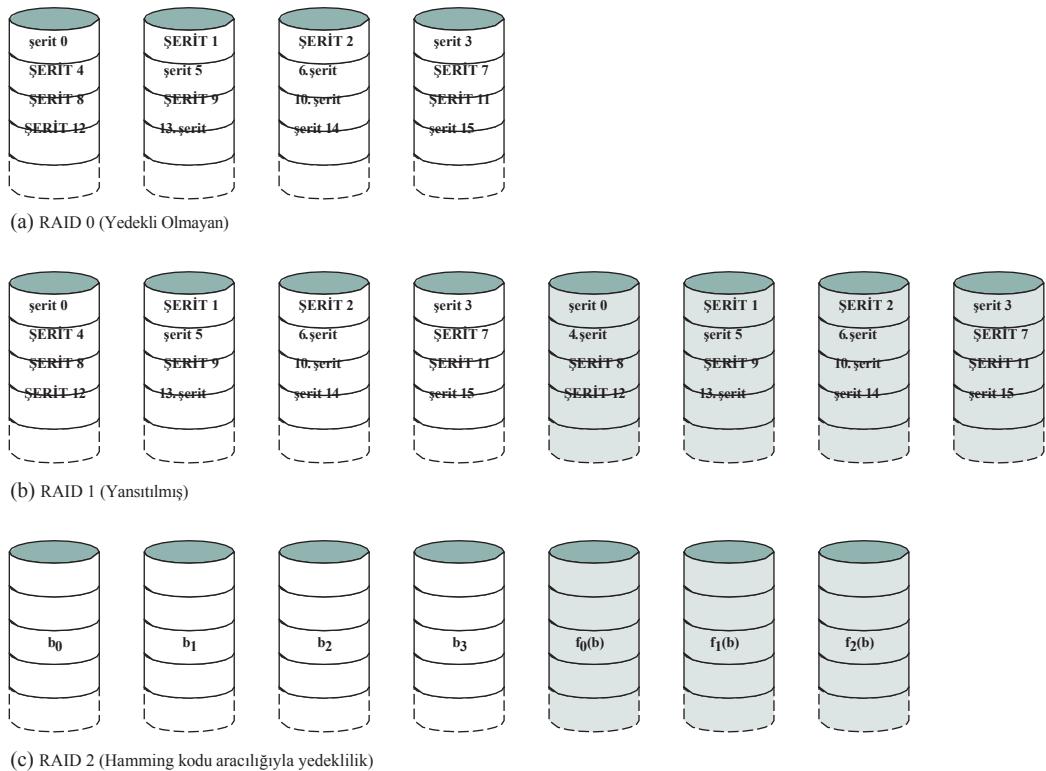
Şekil 6.7, mantıksal ve fiziksel disk alanı arasında eşleme yapmak için dizi yönetim yazılımının kullanımını göstermektedir. Bu yazılım disk alt sisteminde ya da bir ana bilgisayarda çalışabilir.

**YÜKSEK VERİ AKTARIM KAPASİTESİ İÇİN RAID 0** RAID seviyelerinden herhangi birinin performansı kritik ölçüde ana bilgisayar sisteminin istek modellerine ve verilerin düzeneğine bağlıdır. Bu sorunlar en açık şekilde RAID 0'da ele alınabilir, burada

**Tablo 6.3** RAID Seviyeleri

| Kategori        | Seviye | Açıklama                            | Gerekli Diskler | Veri Kullanılabilirliği                                               | Büyük I/O Veri Aktarım Kapasitesi                                           | Küçük I/O İstek Oranı                                                       |
|-----------------|--------|-------------------------------------|-----------------|-----------------------------------------------------------------------|-----------------------------------------------------------------------------|-----------------------------------------------------------------------------|
| Şeritleme       | 0      | Nonredundant                        | $N$             | Tek diskten daha düşük                                                | Çok yüksek                                                                  | Hem okuma hem de yazma için çok yüksekk                                     |
| Yansıtma        | 1      | Aynalı                              | $2N$            | RAID 2, 3, 4 veya 5'ten daha yüksek; daha düşük RAID 6                | Okuma için tek diskten daha yüksek; yazma için tek diske benzer             | Okuma için tek bir diskin iki katına kadar; yazma için tek bir diske benzer |
| Paralel erişim  | 2      | Hamming kodu aracılığıyla yedekli   | $N+m$           | Tek diskten çok daha yüksek; RAID 3, 4 veya 5 ile karşılaştırılabilir | Listelenen tüm alternatifler arasında en yükseği                            | Tek bir diskin yaklaşık iki katı                                            |
|                 | 3      | Bit aralıklı eşlik                  | $N+1$           | Tek diskten çok daha yüksek; RAID 2, 4 veya 5 ile karşılaştırılabilir | Listelenen tüm alternatifler arasında en yükseği                            | Tek bir diskin yaklaşık iki katı                                            |
| Bağımsız erişim | 4      | Blok aralıklı parite                | $N+1$           | Tek diskten çok daha yüksek; RAID 2, 3 veya 5 ile karşılaştırılabilir | Okuma için RAID 0'a benzer; yazma için tek diskten önemli ölçüde daha düşük | Okuma için RAID 0'a benzer; yazma için tek diskten önemli ölçüde daha düşük |
|                 | 5      | Blok aralıklı dağıtılmış eşlik      | $N+1$           | Tek diskten çok daha yüksek; RAID 2, 3 veya 4 ile karşılaştırılabilir | Okuma için RAID 0'a benzer; yazma için tek diskten daha düşük               | Okuma için RAID 0'a benzer; yazma için genellikle tek diskten daha düşük    |
|                 | 6      | Blok aralıklı çift dağıtılmış eşlik | $N+2$           | Listelenen tüm alternatifler arasında en yükseği                      | Okuma için RAID 0'a benzer; yazma için RAID 5'ten daha düşük                | Okuma için RAID 0'a benzer; yazma için RAID 5'ten önemli ölçüde daha düşük  |

Not:  $N$ = veri diskleri sayısı;  $m$  log  $N$  ile orantılı

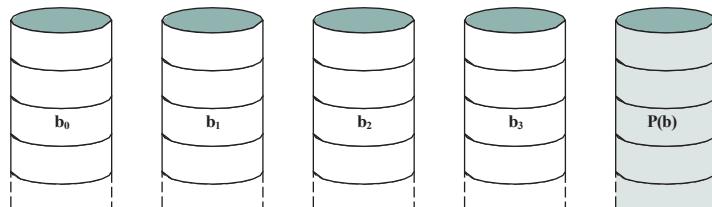
**Şekil 6.6** RAID Seviyeleri (*Devam*)

yedekliliğin etkisi analize müdahale etmez. İlk olarak, yüksek bir veri aktarım hızı elde etmek için RAID 0 kullanımını ele Uygulamaların yüksek aktarım hızı elde edebilmesi için iki gereksininin karşılanması gereklidir. İlk olarak, ana bellek ile disk sürücülerini arasındaki tüm yol boyunca yüksek bir aktarım kapasitesi mevcut olmalıdır. Buna dahili denetleyici veri yolları, ana sistem G/C veri yolları, G/C bağıdaştırıcıları ve ana bellek veri yolları dahildir.

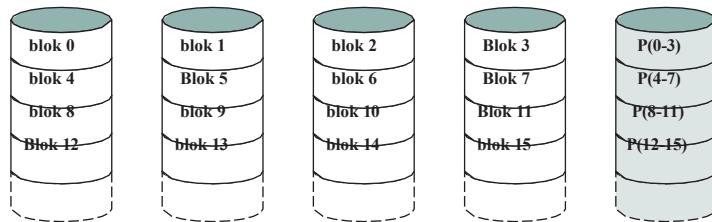
İkinci gereksinim, uygulamanın disk dizisini verimli bir şekilde çalıstırın G/C istekleri yapmasıdır. Tipik istek, bir şeridin boyutuna kıyasla büyük miktarlarda mantıksal olarak bitişik veri içinse bu gereksinin karşılanması gereklidir. Bu durumda, tek bir G/C isteği birden fazla disktten paralel veri aktarımını içerir ve tek diskli aktarımı kıyasla etkin aktarım hızını artırır.

**YÜKSEK G/C İSTEK HIZI İÇİN RAID 0** İşlem odaklı bir ortamda, kullanıcı genellikle aktarım hızından çok yanıt süresiyle ilgilenir. Az miktarda veri için tek bir G/C talebi için, G/C süresi disk kafalarının hareketi (arama süresi) ve diskin hareketi (dönme gecikmesi) tarafından domine edilir.

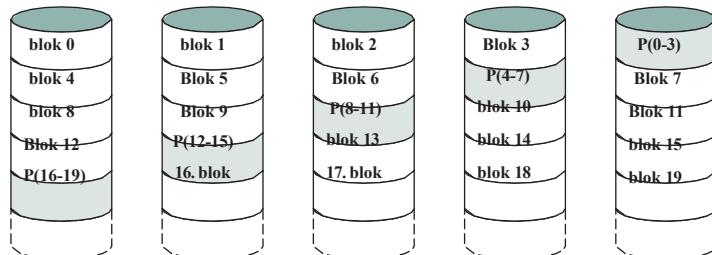
Bir işlem ortamında, saniye başına yüzlerce G/C isteği olabilir. Bir disk dizisi, G/C yükünü birden fazla disk arasında dengeleyerek yüksek G/C yürütme hızları sağlayabilir. Etkili yük dengeleme yalnızca tipik olarak



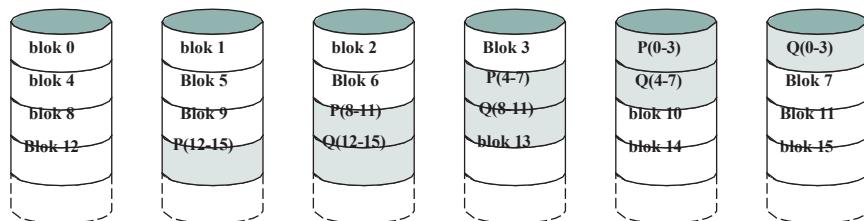
(d) RAID 3 (Bit interleaved parity)



(e) RAID 4 (Blok düzeyinde eşlik)



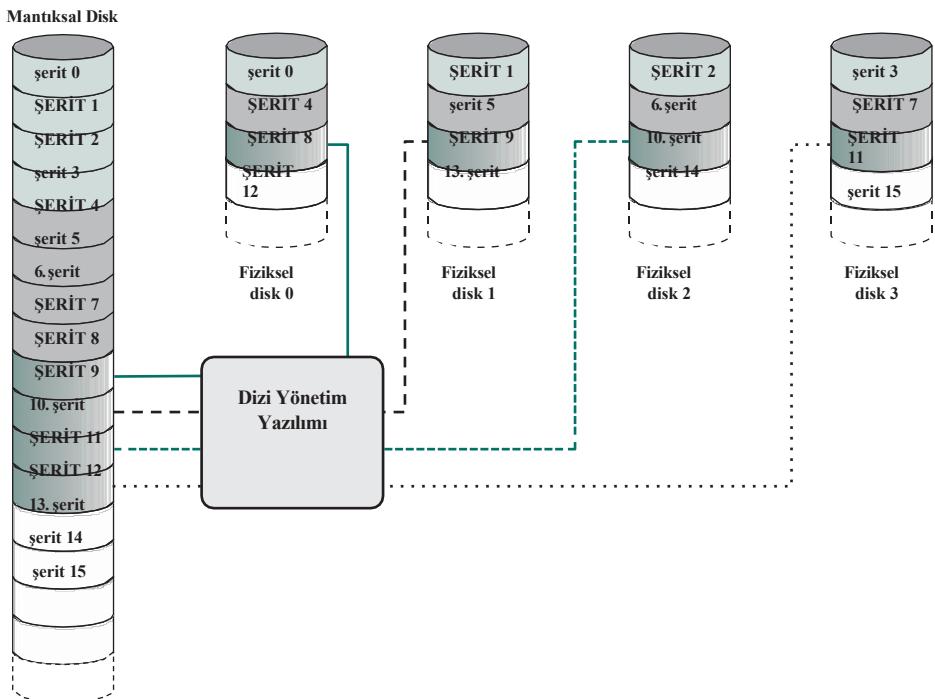
(f) RAID 5 (Blok düzeyinde dağıtılmış eşlik)



(g) RAID 6 (Çift yedeklilik)

**Şekil 6.6** RAID Seviyeleri (*Devamı*)

birden fazla G/C isteği beklemektedir. Bu da birden fazla bağımsız uygulama ya da birden fazla eşzamansız G/C isteği yapabilen tek bir işlem odaklı uygulama olduğu anlamına gelir. Performans ayrıca şerit boyutundan da etkilenecektir. Şerit boyutu nispeten büyükse, yani tek bir G/C isteği yalnızca tek bir disk erişimini içeriyorsa, bekleyen birden fazla G/C isteği paralel olarak ele alınabilir ve her bir istek için kuyruk süresi azaltılabilir.



**Sekil 6.7** RAID Seviye 0 Dizisi için Veri Eşleme

## RAID Seviye 1

RAID 1, yedekliliğin elde edilme şekli bakımından RAID düzey 2 ile 6'dan farklıdır. Bu diğer RAID şemalarında, yedekliliği sağlamak için bir çeşit eşlik hesaplaması, RAID 1'de tüm verilerin çoğaltıması gibi basit bir yöntemle elde edilir. Şekil 6.6b'de gösterildiği gibi, RAID 0'da olduğu gibi veri şeritleme kullanılır. Ancak bu durumda, her mantıksal şerit iki ayrı fiziksel diske eşlenir, böylece dizideki her disk aynı verileri içeren bir ayna diske sahip olur. RAID 1 veri şeritleme olmadan da uygulanabilir, ancak bu daha az yaygındır.

RAID 1 organizasyonunun bir dizi olumlu yönü vardır:

1. Bir okuma isteği, istenen verileri içeren iki diskten hangisi minimum arama süresi artı rotasyonel gecikme içeriyorsa o disk tarafından karşılanabilir.
2. Bir yazma isteği, karşılık gelen her iki şeridin de güncellenmesini gerektirir, ancak bu paralel olarak yapılabilir. Bu nedenle, yazma performansı iki yazma işleminden daha yavaş olanı tarafından belirlenir (yani, daha büyük arama süresi artı dönme gecikmesi içeren). Ancak RAID 1'de "yazma cezası" yoktur. RAID 2 ile 6 düzeyleri eşlik bitlerinin kullanımını içerir. Bu nedenle, tek bir şerit güncellendiğinde, dizi yönetim yazılımı önce eşlik bitlerini hesaplamalı ve güncellemeli, ayrıca söz konusu gerçek şeridi güncellemelidir.
3. Bir arızadan kurtarma basittir. Bir sürücü arızalandığında, verilere ikinci sürücüden erişilmeye devam edilebilir.

RAID 1'in başlıca dezavantajı maliyetidir; desteklediği mantıksal diskin iki katı disk alanı gerektirir. Bu nedenle, bir RAID 1 yapılandırması muhtemelen sistem yazılımı ve verileri ile diğer son derece kritik dosyaları depolayan sürücülerle sınırlı olacaktır. Bu durumlarda RAID 1 tüm verilerin gerçek zamanlı kopyasını sağlar, böylece bir disk arızası durumunda tüm kritik veriler anında kullanılabilir durumda kalır.

İşlem odaklı bir ortamda, isteklerin büyük kısmı okuma ise RAID 1 yüksek I/O istek oranlarına ulaşabilir. Bu durumda RAID 1'in performansı RAID 0'ın iki katına yaklaşabilir. Ancak, G/C isteklerinin önemli bir kısmı yazma istekleri ise, RAID 0'a göre önemli bir performans kazancı olmayabilir. RAID 1, yüksek okuma yüzdesine sahip veri aktarımı yoğun uygulamalar için RAID 0'a göre daha iyi performans da sağlayabilir. Uygulama her okuma isteğini her iki disk üyesinin de katılacağı şekilde bölebilirse iyileşme gerçekleşir.

## RAID Seviye 2

RAID düzey 2 ve 3 paralel erişim tekniğini kullanır. Bir paralel erişim dizisinde, tüm üye diskler her G/C isteğin yürüttülmesine katılır. Tipik olarak, her bir disk kafasının herhangi bir zamanda her bir diskte aynı konumda olması için ayrı sürücülerin milleri senkronize edilir.

Diğer RAID şemalarında olduğu gibi, veri şeritleme kullanılır. RAID 2 ve 3 durumunda, şeritler çok küçüktür, genellikle tek bir bayt veya kelime kadar küçüktür. RAID 2 ile, her bir veri diskindeki karşılık gelen bitler arasında bir hata düzeltme kodu hesaplanır ve kodun bitleri birden fazla par-iti diskindeki karşılık gelen bit konumlarında saklanır. Tipik olarak, tek bitlik hataları düzeltibilecek ve çift bitlik hataları tespit edebilen bir Hamming kodu kullanılır.

RAID 2, RAID 1'den daha az disk gerektirse de yine de oldukça maliyetlidir. Yedek disklerin sayısı veri disklerinin sayısının logu ile orantılıdır. Tek bir okumada, tüm disklere aynı anda erişilir. İstenen veriler ve ilgili hata düzeltme kodu dizi denetleyicisine ilettilir. Tek bitlik bir hata varsa, denetleyici hatayı anında tanıyor ve düzeltilebilir, böylece okuma erişim süresi yavaşlamaz. Tek bir yazma işleminden, yazma işlemi için tüm veri disklerine ve eşlik disklerine erişilmelidir.

RAID 2 yalnızca çok sayıda disk hatasının meydana geldiği bir ortamda etkili bir seçim olacaktır. Tek tek disklerin ve disk sürücülerinin yüksek güvenilirliği göz önüne alındığında, RAID 2 aşırıdır ve uygulanmaz.

## RAID Seviye 3

RAID 3, RAID 2'ye benzer bir şekilde düzenlenmiştir. Aradaki fark, RAID 3'ün disk dizisi ne kadar büyük olursa olsun yalnızca tek bir yedek disk gereklidir. RAID 3, verilerin küçük şeritler halinde dağıtıldığı paralel erişim kullanır. Bir hata düzeltme kodu yerine, tüm veri disklerinde aynı konumda bulunan bireysel bitler kümesi için basit bir eşlik biti hesaplanır.

**YEDEKLEME** Bir sürücü arızası durumunda, eşlik sürücüsüne erişilir ve veriler kalan cihazlardan yeniden yapılandırılır. Arızalı sürücü değiştirildikten sonra, eksik veriler yeni sürücüye geri yüklenebilir ve çalışma devam ettirilebilir.

Veri yeniden yapılandırması basittir. X0'dan X3'e kadar veri içeren ve X4'ün eşlik diski olduğu beş sürücüden oluşan bir dizi düşünün. Birinci bit için eşlik aşağıdaki gibi hesaplanır:

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

burada  $\oplus$  özel-OR fonksiyonudur.

X1 sürücüsünün arızalandığını varsayıyalım. Yukarıdaki denklemin her iki tarafına  $X4(i) \oplus X1(i)$  eklersek, şunları elde ederiz

$$X1(i) = X4(i) \oplus X3(i) \oplus X2(i) \oplus X0(i)$$

Böylece, X1 üzerindeki her bir veri şeridinin içeriği, dizideki diğer diskler üzerindeki ilgili şeritlerin içeriğinden yeniden oluşturulabilir. Bu ilke RAID düzeyleri 3 ile 6 için de geçerlidir.

Bir disk arızası durumunda, tüm veriler indirgenmiş mod olarak adlandırılacak modda hala kullanılabilir durumdadır. Bu modda, okumalar için, eksik veriler exclusive-OR hesaplaması kullanılarak anında yeniden oluşturulur. Veriler azaltılmış RAID 3 dizisine yazıldığında, daha sonra yeniden oluşturma için eşlik tutarlığı korunmalıdır. Tam çalışmaya dönmek için arızalı diskin değiştirilmesi ve arızalı diskin tüm yeni diskte yeniden oluşturulması gereklidir.

**PERFORMANS** Veriler çok küçük şeritler halinde şeritlendiğinden, RAID 3 çok yüksek veri aktarım hızlarına ulaşabilir. Herhangi bir I/O isteği, tüm veri disklerinden paralel veri aktarımını içerecektir. Büyük transferler için performans artışı özellikle . Öte yandan, bir seferde yalnızca bir G/C isteği gerçekleştirilebilir. Bu nedenle, işlem odaklı bir ortamda performans düşer.

## RAID Seviye 4

RAID 4 ile 6 seviyeleri bağımsız erişim teknğini kullanır. Bağımsız erişimli bir dizide, her üye disk bağımsız olarak çalışır, böylece ayrı G/C istekleri paralel olarak karşılanabilir. Bu nedenle, bağımsız erişim dizileri yüksek G/C istek hızları gerektiren uygulamalar için daha uygundur ve yüksek veri aktarım hızları gerektiren uygulamalar için nispeten daha az uygundur.

Diğer RAID şemalarında olduğu gibi, veri şeritleme kullanılır. RAID 4 ile 6 durumunda, şeritler nispeten büyütür. RAID 4 ile, her bir veri diskindeki karşılık gelen şeritler boyunca bit bit eşlik şeridi hesaplanır ve eşlik bitleri eşlik diskindeki karşılık gelen şeritte saklanır.

RAID 4, küçük boyutlu bir G/C yazma isteği olduğunda bir yazma cezası içerir. Bir yazma işlemi her gerçekleştiginde, dizi yönetim yazılımı yalnızca kullanıcı verilerini değil aynı zamanda ilgili eşlik bitlerini de güncellemelidir. X0'dan X3'e kadar veri içeren ve X4'ün eşlik diski olduğu beş sürücüden oluşan bir dizi düşünün. X1 diskinde yalnızca bir şerit içeren bir yazma işlemi gerçekleştirildiğini varsayıyalım. Başlangıçta, her  $i$  biti için aşağıdaki ilişkiye sahibiz:

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i) \quad (6.2)$$

Güncellemeden sonra, potansiyel olarak değiştirilmiş bitler asal bir sembolle gösterilir:  $X4'(i) =$

$$\begin{aligned} X3(i) \oplus X2(i) \oplus X1'(i) X0(i) \\ = X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \oplus X1(i) \oplus X1(i) \\ = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i) \oplus X1(i) \oplus X1'(i) \\ = X4(i) \oplus X1(i) \oplus X1'(i) \end{aligned}$$

Onceki denklem seti aşağıdaki şekilde türetilmiştir. İlk satır, X1'deki bir değişikliğin X4 parite diskini de etkileyeceğini göstermektedir. İkinci satırda, şu terimleri ekliyoruz

$\oplus X1(i) \oplus X1(i)$ ]. Herhangi bir niceliğin kendisiyle özel-OR'su 0 olduğundan, bu denklemi etkilemez. Ancak, yeniden sıralama yoluyla üçüncü satırı oluşturmak için kullanılan bir kolaylıktır. Son olarak, ilk dört terimi X4(i) ile değiştirmek için Denklem (6.2) kullanılır.

Yeni pariteyi hesaplamak için, dizi yönetim yazılımı eski kullanıcı şeridini ve eski parite şeridini okumalıdır. Daha sonra bu iki şeridi yeni veriler ve yeni hesaplanan eşlik ile güncelleyebilir. Bu nedenle, her şerit yazma işlemi iki okuma ve iki yazma içerir.

Tüm disk sürücülerindeki şeritleri içeren daha büyük boyutlu bir I/O yazma durumunda, eşlik yalnızca yeni veri bitleri kullanılarak hesaplama yoluyla kolayca hesaplanır. Böylece eşlik sürücüsü veri sürücüleriyle paralel olarak güncellenebilir ve fazladan okuma ya da yazma yapılmaz. Her durumda, her yazma işlemi eşlik diskini içermelidir, ki orada bir darboğaz haline gelebilir.

### RAID Seviye 5

RAID 5, RAID 4'e benzer bir şekilde düzenlenmiştir. Aradaki fark RAID 5'in eşlik şeritlerini tüm disklere dağıtmıştır. Tipik bir dağıtım, Şekil 6.6'de gösterildiği gibi bir round-robin şemasıdır. Bir  $n$ -disk dizisi için, eşlik şeridi ilk  $n$  şerit için farklı bir diskte bulunur ve model daha sonra tekrar eder.

Eşlik şeritlerinin tüm sürücülere dağıtılması, RAID 4'te bulunan potansiyel I/O darboğazını önler.

### RAID Seviye 6

RAID 6, Berkeley araştırmacıları tarafından daha sonraki bir makalede tanıtılmıştır [KATZ89]. RAID 6 şemasında, iki farklı eşlik hesaplaması ve farklı disklerde ayrı bloklarda saklanır. Böylece, kullanıcı verileri  $N$  disk gerektiren bir RAID 6 dizisi  $N+2$  diskten oluşur.

Şekil 6.6g şemayı göstermektedir. P ve Q iki farklı veri kontrol algo- ritmasıdır. İkisinden biri RAID 4 ve 5'te kullanılan exclusive-OR hesaplamasıdır. Ancak diğer bağımsız bir veri kontrol algoritmasıdır. Bu, kullanıcı verilerini içeren iki disk arızalanasa bile verilerin yeniden oluşturulmasını mümkün kılar.

RAID 6'nın avantajı, son derece yüksek veri kullanılabilirliği sağlasıdır. Verilerin kaybolması için MTTR (ortalama onarım süresi) aralığında üç diskin arızalanması gereklidir. Öte yandan, RAID 6 önemli bir yazma cezasına maruz kalır, çünkü her yazma iki eşlik bloğunu etkiler. Performans kıyaslamaları [EISC07] bir RAID 6 denetleyicisinin RAID 5 uygulamasına kıyasla genel yazma performansında %30'dan fazla düşüş yaşayabileceğini göstermektedir. RAID 5 ve RAID 6 okuma performansı karşılaştırılabilir düzeydedir.

Tablo 6.4 yedi seviyenin karşılaştırmalı bir özeti

## 6.3 KATI HAL SÜRÜCÜLERİ

Son yıllarda bilgisayar mimarisindeki en önemli gelişmelerden biri, hem dahili hem de harici ikincil bellek olarak **sabit disk sürücülerini (HDD'ler)** tamamlamak ve hatta değiştirmek için kati hal **sürücülerinin (SSD'ler)** artan kullanılmıştır. *Kati hal* terimi

**Tablo 6.4** RAID Karşılaştırması

| Seviye | Avantajlar                                                                                                                                                                                                                                                                                         | Dezavantajlar                                                                                                                                                                                                          | Uygulamalar                                                                                                                                          |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0      | G/C yükü birçok kanala ve sürücüye yayılarak G/C performansı büyük ölçüde iyileştirilir<br>Parite hesaplama ek yükü yoktur Çok basit tasarım<br>Uygulaması kolay                                                                                                                                   | Tek bir sürücünün arızalanması, dizideki tüm verilerin kaybolmasına neden olur                                                                                                                                         | Video produksiyonu ve kurgusu<br>Görüntü Düzenleme<br>Baskı öncesi uygulamalar<br>Yüksek bant genişliği gerektiren her türlü uygulama                |
| 1      | 100 veri yedekliliği, bir disk arızası durumunda yeniden oluşturulmaya gerek olmadığı, sadece yedek diske bir kopyalama yapıldığı anlamına gelir<br><br>Belirli koşullar altında, RAID 1 aynı anda birden fazla sürücü arızasına dayanabilir<br><br>En basit RAID depolama alt sistemi tasarımları | Tüm RAID türleri arasında en yüksek disk ek yükü (%100)-verimsiz                                                                                                                                                       | Muhasebe Bordro<br>Finansal<br>Çok yüksek kullanılabılırlik gerektiren tüm uygulamalar                                                               |
| 2      | Son derece yüksek veri aktarım hızları mümkün<br>Gereken veri aktarım hızı ne kadar yüksekse, veri disklerinin ECC disklerine oranı o kadar iyii olur<br><br>RAID seviyeleri 3, 4 ve 5 ile karşılaştırıldığında nispeten basit kontrolör tasarımları                                               | ECC disklerinin veri disklerine oranının çok yüksek olması daha küçük kelime boyutları ile - verimsiz<br><br>Giriş seviyesi maliyeti çok yüksektir, haklı çikarmak için çok yüksek aktarım hızı gereksinimi gerektirir | Ticari uygulama mevcut değil/ticari olarak uygulanabilir değil                                                                                       |
| 3      | Cök yüksek okuma veri aktarım hızı Çok yüksek yazma veri aktarım hızı<br>Disk arızasının verim üzerinde önemiz bir etkisi vardır<br>ECC (eşlik) disklerinin veri disklerine oranının düşük olması yüksek verimlilik anlamına gelir                                                                 | İşlem hızı en iyi ihtimalle tek bir disk sürücüsünün kine eşittir (iğler senkronize edilmişse)<br>Kontrolör tasarımları oldukça karmaşıktr                                                                             | Video produksiyonu ve canlı yayın<br>Görüntü düzenleme<br>Video düzenleme<br>Baskı öncesi uygulamaları<br>Yüksek verim gerektiren her türlü uygulama |
| 4      | Cök yüksek Okuma verisi işlem hızı<br>ECC (eşlik) disklerinin veri disklerine oranının düşük olması yüksek verimlilik anlamına gelir                                                                                                                                                               | Ölçük karmaşık kontrolör tasarımları<br>En kötü yazma işlem hızı ve Yazma toplam aktarım hızı<br>Disk arızası durumunda zor ve verimsiz veri yeniden oluşturma                                                         | Ticari uygulama mevcut değil/ticari olarak uygulanabilir değil                                                                                       |
| 5      | En Yüksek Okuma verisi işlem hızı<br>ECC (eşlik) disklerinin veri disklerine oranının düşük olması yüksek verimlilik anlamına gelir<br>İyi toplam aktarım hızı                                                                                                                                     | En karmaşık kontrolör tasarımları<br>Bir disk arızası durumunda yeniden zor olmasa (RAID seviye 1 ile karşılaştırıldığında)                                                                                            | Dosya ve uygulama sunucuları<br>Veritabanı sunucuları<br>Web, e-posta ve haber sunucuları<br>Intranet sunucuları<br>En çok yönlü RAID seviyesi       |
| 6      | Son derece yüksek bir veri hata toleransı sağlar ve aynı anda birden fazla sürücü arızasını sürdürbilir                                                                                                                                                                                            | Daha karmaşık kontrolör tasarımları<br>Eşlik adreslerini hesaplamak için kontrolör ek yükü son derece yüksektir                                                                                                        | Kritik uygulamalar için mükemmel çözüm                                                                                                               |

yarı iletkenler ile oluşturulmuş elektronik devre anlamına gelmektedir. Bir SSD, sabit disk sürücüsünün yerine kullanılabilen katı hal bileşenlerinden yapılmış bir bellek cihazıdır. Şu anda piyasada bulunan ve piyasaya çıkacak SSD'ler, Bölüm 5'te açıklanan NAND flash bellek kullanmaktadır.

### HDD ile Karşılaştırıldığında SSD

Flash tabanlı SSD'lerin maliyeti düştükçe ve performans ve bit yoğunluğu arttıkça SSD'ler HDD'lerle giderek daha rekabetçi hale gelmiştir. Tablo 6.5, bu yazının yazıldığı sırada tipik karşılaştırma ölçülerini göstermektedir.

SSD'ler HDD'lere göre aşağıdaki avantajlara sahiptir:

- **Saniye başına yüksek performanslı giriş/çıkış işlemleri (IOPS):** G/C alt sistemlerinin performansını önemli ölçüde artırır.
- **Dayanıklılık:** Fiziksel şok ve titreşime daha az duyarlıdır.
- **Daha uzun kullanım ömrü:** SSD'ler mekanik aşınmaya karşı hassas değildir.
- **Daha düşük güç tüketimi:** SSD'ler, benzer boyuttaki HDD'lere göre önemli ölçüde daha az güç kullanır.
- **Daha sessiz ve daha serin çalışma özellikleri:** Daha az yer ihtiyacı, daha düşük enerji maliyetleri ve daha çevreci bir işletme.
- **Daha düşük erişim süreleri ve gecikme oranları:** Bir HDD'deki dönen disklerden 10 kat daha hızlıdır.

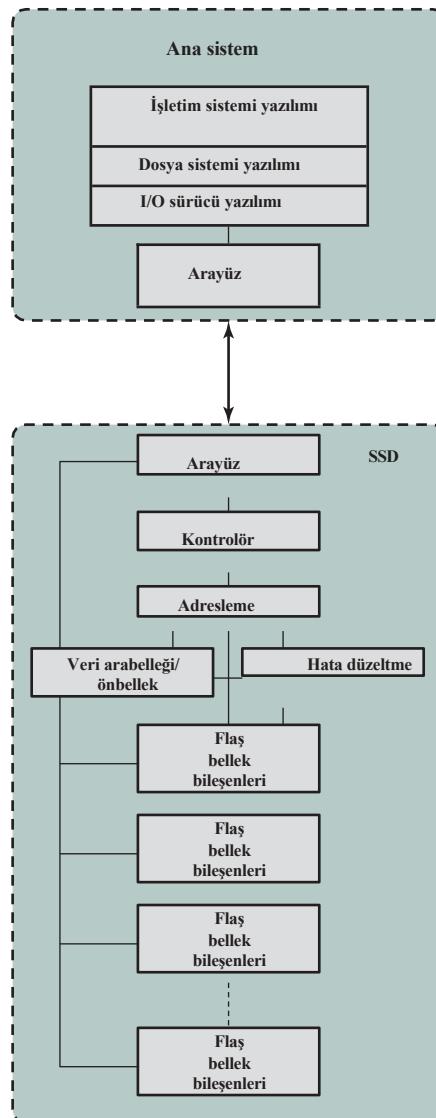
Şu anda HDD'ler bit başına maliyet avantajına ve kapasite avantajına sahiptir, ancak bu farklar azalmaktadır.

### SSD Organizasyonu

Şekil 6.8, herhangi bir SSD sistemi ile ilişkili ortak mimari sistem bileşeninin genel bir görünümünü göstermektedir. Ana sistemde işletim sistemi, diskteki verilere erişmek için dosya sistemi yazılımını çağırır. Dosya sistemi de I/O sürücü çağrıır. I/O sürücü yazılımı, belirli SSD ürününe ana bilgisayar erişimi sağlar. Şekil 6.8'deki arayüz bileşeni, ana bilgisayar işlemcisi ile SSD çevresel aygıtı arasındaki fiziksel ve elektriksel arayüzü ifade etmektedir. Eğer cihaz dahili bir sabit sürücü ise, ortak arayüz PCIe'dir. Harici cihazlar için yaygın bir arayüz USB'dir.

**Tablo 6.5** Katı Hal Sürücüler ve Disk Sürücülerinin Karşılaştırılması

|                            | NAND Flash Sürücüler                                                                                                          | Seagate Dizüstü Bilgisayar Dahili HDD                                                                                                  |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| Dosya kopyalama/yazma hızı | 200-550 Mbps                                                                                                                  | 50-120 Mbps                                                                                                                            |
| Güç çekişi/pil ömrü        | Daha az güç çeker, ortalama 2-3 watt, 30+ dakika pil artışı sağlar                                                            | Daha fazla güç çeker, ortalama 6-7 watt ve bunedenle daha fazla pil kullanır                                                           |
| Depolama kapasitesi        | Dizüstü bilgisayar boyutundaki sürücüler için genellikle 512 GB'tan büyük değildir; masaüstü bilgisayarlar için maksimum 1 TB | Dizüstü bilgisayar boyutundaki sürücüler için tipik olarak yaklaşık 500 GB ve maksimum 2 TB; masaüstü bilgisayarlar için maksimum 4 TB |
| Maliyet                    | 1 TB sürücü için GB başına yaklaşık 0,50 ABD doları                                                                           | 4 TB sürücü için GB başına yaklaşık 0,15 ABD doları                                                                                    |



Şekil 6.8 Katı Hal Sürücü Mimarisi

Ana sisteme giden arayüze ek olarak SSD, aşağıdaki bileşenleri içerir:

- **Denetleyici:** SSD cihaz düzeyinde arayüz ve ürün yazılımı yürütme sağlar.
- **Adresleme:** Flaş bellek bileşenleri arasında seçim işlemini gerçekleştiren mantık.
- **Veri tamponu/önbellegi:** Hız eşleştirme ve veri verimini artırmak için kullanılan yüksek hızlı RAM bellek bileşenleri.

- **Hata düzeltme:** Hata tespiti ve düzeltmesi için mantık.
- **Flash bellek bileşenleri:** Bireysel NAND flaş yongaları.

### Pratik Sorunlar

SSD'lere özgü olan ve HDD'lerde karşılaşılmayan iki pratik sorun vardır. Birincisi, SSD performansı cihaz kullanıldıkça yavaşlama eğilimindedir. Bunun nedenini anlamak için, dosyaların diskte tipik olarak 4 KB uzunluğunda bir dizi sayıa olarak saklandığını bilmeniz gereklidir. Bu sayfaların disk üzerinde bitişik sayfalar depolanması gerekmek ve aslında tipik olarak da böyle değildir. Bu düzenlemenin nedeni Bölüm 8'deki sanal bellek tartışmamızda açıklanmıştır. Bununla birlikte, flash belleğe bloklar halinde erişilir, tipik blok boyutu 512 KB'dır, böylece blok başına tipik olarak 128 sayfa bulunur. Şimdi flash belleğe bir sayfa yazmak için ne yapılması gerektiğini düşünün.

1. Tüm blok flash bellekten okunmalı ve bir RAM tamponuna yerleştirilmelidir. Ardından RAM tamponundaki uygun sayfa güncellenir.
2. Bloğun flash belleğe geri yazılabilmesi için flash bellek bloğunun tamamının silinmesi gereklidir; flash belleğin sadece bir sayfasının silinmesi mümkün değildir.
3. Tampondaki tüm blok şimdi flash belleğe geri yazılır.

Şimdi, bir flash sürücü nispeten boş olduğunda ve yeni bir dosya oluşturulduğunda, bu dosyanın sayfaları sürücüye bitişik olarak yazılır, böylece bir veya sadece birkaç blok etkilenir. Ancak zamanla, sanal belleğin çalışma şekli nedeniyle dosyalar parçalanır ve sayfalar birden fazla bloğa dağılır. Sürücü daha fazla dolduğunda, daha fazla parçalanma olur, bu nedenle yeni bir dosyanın yazılması birden fazla bloğu etkileyebilir. Böylece, bir bloktan birden fazla sayfanın yazılması, disk ne kadar dolu olursa o kadar yavaşlar. Üreticiler, flash belleğin bu özelliğini telafi etmek için SSD'nin önemli bir bölümünü yazma işlemleri için ekstra alan olarak ayırmak (asıri provizyon olarak adlandırılır) ve ardından diskı bireştirmek için kullanılan boşta kalma süresi boyunca etkin olmayan sayfaları silmek gibi teknikler geliştirmiştir. Bir diğer teknik ise işletim sisteminin SSD'ye hangi veri bloklarının artık kullanılmış olduğunu ve dahili olarak silinebileceğini sağlayan TRIM komutudur<sup>4</sup>.

Flash bellek sürücüleriley ilgili ikinci bir pratik sorun, bir flaş belleğin belirli sayıda yazmadan sonra kullanılamaz hale gelmesidir. Flaş hücreler strese girdikçe, değerleri kaydetme ve saklama yeteneklerini kaybederler. Tipik bir sınır 100.000 yazmadır [GSOE08]. Bir SSD sürücünün ömrünü uzatmak için kullanılan teknikler arasında yazma işlemlerini geciktirmek ve gruplandırmak için flash'ı bir önbellekle önden bağlamak, yazma işlemlerini hücre bloklarına eşit olarak dağıtan aşınma seviyelendirme algoritmaları kullanmak ve gelişmiş bozuk blok yönetimi teknikleri yer almaktadır. Ayrıca satıcılar, veri kaybı olasılığını daha da azaltmak için SSD'leri RAID konfigürasyonlarında kullanmaktadır. Çoğu flash cihazı ayrıca kendi kalan ömrülerini tahmin edebilmektedir, böylece sistemler arızayı öngörebilmekte ve önleyici tedbirler alabilmektedir.

---

<sup>4</sup>TRIM sıkılıkla büyük harflerle yazılsa da, bir kısaltma değildir; yalnızca bir komut adıdır.

## 6.4 OPTİK BELLEK

1983 yılında, tüm zamanların en başarılı tüketici ürünlerinden biri tanıtıldı: kompakt disk (CD) dijital ses sistemi. CD, bir yüzünde 60 dakikadan fazla ses bilgisi saklayabilen, silinemeyen bir disktir. CD'nin büyük ticari başarısı, bilgisayar veri depolamasında devrim yaratınan düşük maliyetli optik disk depolama teknolojisinin geliştirilmesini sağladı. Çeşitli optik disk sistemleri piyasaya sürülmüştür (Tablo 6.6). Bunların her birini kısaca gözden geçireceğiz.

### Kompakt Disk

**CD-ROM** Hem ses CD'si hem de **CD-ROM** (kompakt disk salt okunur bellek) benzer bir teknolojiyi paylaşır. Temel fark, CD-ROM oynatıcıların daha sağlam olması ve verilerin diskten bilgisayara düzgün bir şekilde aktarılmasını sağlamak için hata düzeltme cihazlarına sahip olmasıdır. Her iki disk türü de aynı şekilde yapılır. Disk, polikarbonat gibi bir reçineden oluşturmaktadır. Dijital olarak kaydedilen bilgiler (müzük ya da bilgisayar verileri) polikarbonat yüzeyine bir dizi mikroskopik çukur olarak basılır. Bu işlem öncelikle, bir ana disk oluşturmak için ince odaklanmış, yüksek yoğunluklu bir lazerle yapılır. Master disk, polikarbonat üzerine kopya basmak üzere bir kalıp yapmak için kullanılan Çukurlu yüzey daha sonra genellikle alüminyum veya altın gibi yüksek yansıtıcı bir yüzeyle kaplanır. Bu parlak yüzey toz ve çizilmelere karşı şeffaf akrilik bir üst kat ile korunur. Son olarak, akrilik üzerine bir etiket serigrafi ile basılabilir.

**Tablo 6.6** Optik Disk Ürünleri

|                    |                                                                                                                                                                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CD</b>          | Kompakt Disk. Sayısalştırılmış ses bilgilerini depolayan, silinemeyen bir disk. Standart sistem 12 cm'lik diskler kullanır ve 60 dakikadan fazla kesintisiz çalışma süresi kaydedebilir.                                                                                                                 |
| <b>CD-ROM</b>      | Kompakt Disk Salt Okunur Bellek. Bilgisayar verilerini depolamak için kullanılan silinemeyen bir disk. Standart sistemde 12 cm'lik diskler kullanılır ve 650 Mbyte'tan daha fazlasını tutabılır.                                                                                                         |
| <b>CD-R</b>        | CD Kaydedilebilir. CD-ROM'a benzer. Kullanıcı diske yalnızca bir kez yazabilir.                                                                                                                                                                                                                          |
| <b>CD-RW</b>       | CD Yeniden Yazılabilir. CD-ROM'a benzer. Kullanıcı diski birden çok kez silebilir ve yeniden yazabilir.                                                                                                                                                                                                  |
| <b>DVD</b>         | Dijital Çok Yönlü Disk. Video bilgilerinin yanı sıra büyük hacimli diğer dijital verilerin sayısallaştırılmış, sıkıştırılmış gösterimini üretmek için bir teknoloji. Hem 8 hem de 12 cm çaplarında kullanılır ve çift taraflı kapasitesi 17 Gbyte'a kadar çıkabilir. Temel DVD salt okunurdur (DVD-ROM). |
| <b>DVD-R</b>       | DVD Kaydedilebilir. DVD-ROM'a benzer. Kullanıcı diske yalnızca bir kez yazabilir. Yalnızca tek taraflı diskler kullanılabilir.                                                                                                                                                                           |
| <b>DVD-RW</b>      | DVD Yeniden Yazılabilir. DVD-ROM'a benzer. Kullanıcı diski birden çok kez silebilir ve yeniden yazabilir. Yalnızca tek taraflı diskler kullanılabilir.                                                                                                                                                   |
| <b>Blu-ray DVD</b> | Yüksek çözünürlüklü video diski. 405-nm (mavi-mor) lazer kullanarak DVD'den çok daha fazla veri depolama yoğunluğu sağlar. Tek bir taraftaki tek bir katman 25 Gbyte depolayabilir.                                                                                                                      |

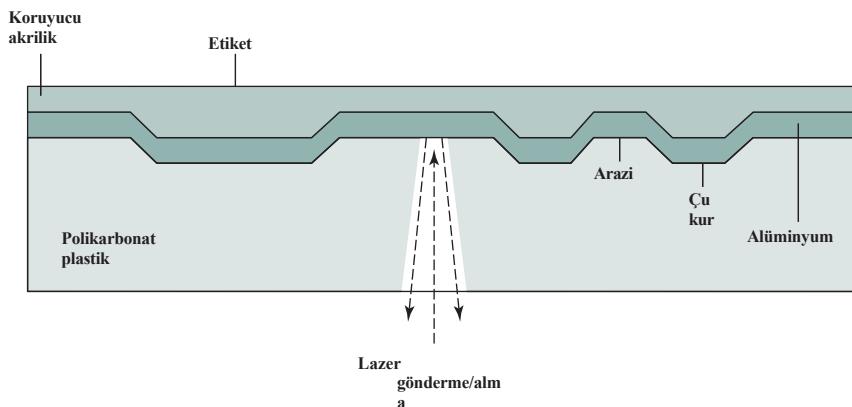
Bilgiler bir CD veya CD-ROM'dan optik disk oynatıcı veya sürücü ünitesinde bulunan düşük güçlü bir lazer tarafından alınır. Lazer, bir motor diski döndürürken şeffaf polikarbonatın içinden parlar (Şekil 6.9). Lazerin yansiyan ışığının yoğunluğu bir **çukurla** karşılaşlığında değişir. Özellikle, lazerini biraz pürüzlü bir yüzeye sahip bir çukura düşerse, ışık dağılır ve düşük bir yoğunluk kaynağına geri yansitılır. Çukurlar arasındaki alanlara **arazi** denir. Arazi, daha yüksek yoğunlukta geri yansiyen pürüzsüz bir yüzeydir. Çukurlar ve topraklar arasındaki değişim bir fotosensör tarafından algılanır ve dijital bir sinyale dönüştürülür. Sensör yüzeyi düzenli aralıklarla test eder. Bir çukurun başlangıcı veya sonu 1'i temsil eder; aralıklar arasında yükseklikte herhangi bir değişiklik olmadığında 0 kaydedilir.

Manyetik bir diskte bilgilerin eşmerkezli izler halinde kaydedildiğini hatırlayın. En basit sabit açısal hız (CAV) sisteminde iz başına düşen bit sayısı . Yoğunluktaki artışı, yüzeyin bir dizi bölgeye ayrıldığı ve merkezden uzak bölgelerin yakın bölgelerden daha fazla bit içeriği çoklu **bölge kaydı** ile elde edilir. Bu teknik kapasiteyi artırır da yine de optimum değildir.

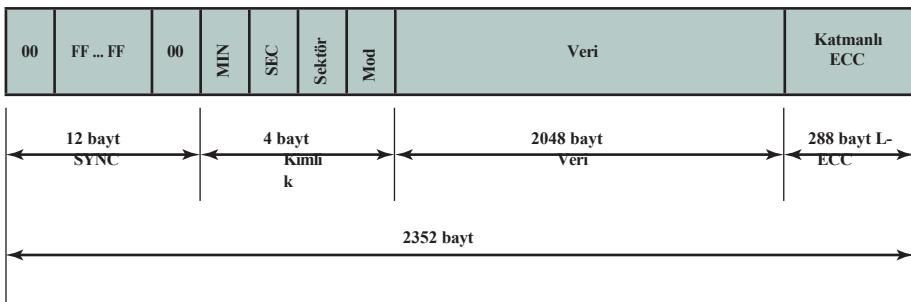
Daha fazla kapasite elde etmek için CD'ler ve CD-ROM'lar bilgileri eşmerkezli izler üzerinde organize etmez. Bunun yerine disk, merkeze yakın bir yerden başlayan ve diskin dış kenarına doğru spiral şeklinde uzanan tek bir螺旋 iz içerir. Diskin dışına yakın sektörler, içine yakın olanlara aynı uzunluktur. Böylece bilgiler disk boyunca aynı boyuttaki bölmeler halinde eşit olarak paketlenir ve bunlar disk değişken bir hızda döndürülerek aynı oranda taranır. Çukurlar daha sonra lazer tarafından **sabit bir doğrusal hızda (CLV)** okunur. Disk, dış kenara yakın erişimler için merkeze yakın olanlara göre daha yavaş döner. Böylece, bir parçanın kapasitesi ve dönme gecikmesi diskin dış kenarına yakın konumlar için artar. Bir CD-ROM için veri kapasitesi yaklaşık 680 MB'dır.

CD-ROM'daki veriler bir dizi blok olarak düzenlenmiştir. Tipik bir blok biçimi Şekil 6.10'da gösterilmiştir. Aşağıdaki alanlardan oluşur:

- **Senkronizasyon:** Senkronizasyon alanı bir bloğun başlangıcını tanımlar. Tüm 0'lardan oluşan bir bayt, tüm 1'lardan oluşan 10 bayt ve tüm 0'lardan oluşan bir bayttan oluşur.
- **Başlık:** Başlık blok adresini ve mod baytını içerir. Mod 0 boş bir veri alanını belirtir; mod 1 hata düzeltici bir veri alanının kullanımını belirtir.



Şekil 6.9 CD Çalışması



**Şekil 6.10** CD-ROM Blok Formatı

kodu ve 2048 bayt veri; mod 2, hata düzeltme kodu olmadan 2336 bayt kullanıcı verisi belirtir.

- **Veri:** Kullanıcı verileri.
- **Yardımcı:** Mod 2'de ek kullanıcı verileri. Mod 1'de bu 288 baytlık bir hata düzeltme kodudur.

CLV kullanımı ile rastgele erişim daha zor hale gelir. Belirli bir adresin bulunması, kafanın genel alana taşınmasını, dönüş hızının ayarlanması ve adresin okunmasını ve ardından belirli sektörü bulmak ve erişmek için küçük ayarlamalar yapılmasını içerir.

CD-ROM büyük miktarda verinin çok sayıda kullanıcıya dağıtılması için uygundur. İlk yazma işleminin masraflı olması nedeniyle, bireyselleştirilmiş uygulamalar için uygun değildir. Geleneksel manyetik disklerle karşılaştırıldığında CD-ROM'un iki avantajı vardır:

- Optik disk, üzerinde depolanan bilgilerle birlikte, manyetik diskten farklı olarak ucuz bir şekilde toplu olarak çoğaltılabılır. Manyetik disk üzerindeki veritabanının iki disk sürücüsü kullanılarak her seferinde bir disk kopyalanarak çoğaltılmazı gereklidir.
- Optik disk çıkarılabilir ve diskin kendisinin için kullanılmasına olanak tanır. Manyetik disklerin çoğu çıkarılamaz. Çıkarılamayan manyetik disklerdeki bilgiler, disk sürücüsünün/diskin yeni bilgileri depolamak için kullanılabilmesi için önce başka bir depolama ortamına kopyalanmalıdır.

CD-ROM'un dezavantajları aşağıdaki gibidir:

- Salt okunurdur ve güncellenemez.
- Manyetik disk sürücüsünden çok daha uzun, yarımi saniye kadar bir erişim süresine sahiptir.

**CD KAYDEDİLEBİLİR** Bir veri kümnesinin yalnızca bir ya da az sayıda kopyasına ihtiyaç duyulan uygulamalara uyum sağlamak için **CD kaydedilebilir (CD-R)** olarak bilinen bir kez yaz bir kez oku **CD'si** geliştirilmiştir. CD-R için bir disk, daha sonra orta yoğunlukta bir lazer ışını ile bir kez yazılabilecek şekilde hazırlanır. Böylece, CD-ROM'a göre biraz daha pahalı bir disk denetleyici ile müsteri diski okumanın yanı sıra bir kez yazılabilir.

CD-R ortamı CD ya da CD- ROM ortamına benzer ancak aynı değildir. CD'ler ve CD-ROM'lar için bilgi, yüzeyin oyulmasıyla kaydedilir

ortamın yansıtıcılığını değiştirir. Bir CD-R için, ortam bir boyalı katmanı içerir. Boya yansıtıcılığı değiştirmek için kullanılır ve yüksek yoğunluklu bir lazer tarafından etkinleştirilir. Ortaya çıkan disk bir CD-R sürücüsünde ya da bir CD-ROM sürücüsünde okunabilir.

CD-R optik disk, belgelerin ve dosyaların arşivsel depolanması için cazipdir. Büyük hacimli kullanıcı verilerinin kalıcı kaydını sağlar.

**cD YENİDEN YAZILABİLİR CD-RW** optik disk, manyetik diskte olduğu gibi tekrar yazılabilir ve üzerine yazılabilir. Bir dizi yaklaşım denememiş olmasına rağmen, cazip olduğu kanıtlanan tek saf optik yaklaşım **faz değişimi** olarak adlandırılmaktadır. Faz değişimi diski, iki farklı faz durumunda iki önemli ölçüde farklı yansıtma özelliğine sahip bir malzeme kullanır. Moleküllerin ışığı zayıf yansitan rastgele bir yönelik sergilediği amorf bir durum ve ışığı iyi yansitan pürüzsüz bir yüzeye sahip olan kristal bir durum vardır. Bir lazer ışığı demeti malzemeyi bir fazdan diğerine değiştirebilir. Faz değişimi optik disklerin birincil dezavantajı, malzemenin eninde sonunda ve kalıcı olarak istenen özelliklerini kaybetmesidir. Mevcut malzemeler 500.000 ila 1.000.000 silme döngüsü için kullanılabilirmektedir.

CD-RW, CD-ROM ve CD-R'a göre yeniden yazılabilmesi ve böylece gerçek bir ikincil depolama olarak kullanılabilmesi gibi bariz bir avantaja sahiptir. Bu nedenle manyetik disk ile rekabet etmektedir. Optik diskin önemli bir avantajı, optik diskler için mühendislik toleranslarının yüksek kapasiteli manyetik disklere göre çok daha az olmasıdır. Böylece daha yüksek güvenilirlik ve daha uzun عمر sergilerler.

### Dijital Çok Yönlü Disk

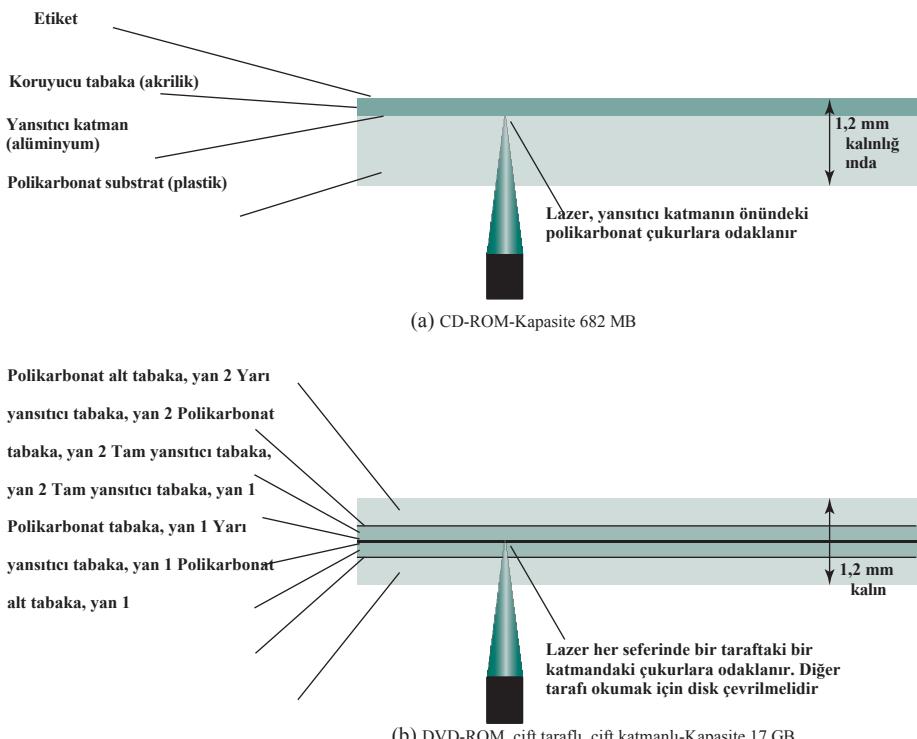
Geniş **dijital çok yönlü disk (DVD)** ile elektronik endüstrisi sonunda analog VHS video kasetinin yerine kabul edilebilir bir alternatif bulmuştur. DVD, video kaset kaydedicilerde (VCR) kullanılan video kasetin yerini almış ve bu tartışma için daha da , kişisel bilgisayarlar ve sunuculardaki CD-ROM'un yerini almıştır. DVD videoyu dijital çağ'a taşıyor. Etkileyici görüntülü kalitesine sahip filmler sunar ve DVD makinelerinin de çalabildiği ses CD'leri gibi rastgele erişilebilir. Şu anda bir CD-ROM'dan yedi kat daha fazla olan büyük hacimli veriler diske süzürlenebilir. DVD'nin büyük depolama kapasitesi ve canlı kalitesi sayesinde PC oyuncuları daha gerçekçi hale gelmiş ve eğitim yazılımları daha fazla video içermeye başlamıştır. Bu gelişmelerin ardından, bu materyalin Web sitelerine dahil edilmesiyle Internet ve kurumsal intranetler üzerinde yeni bir trafik patlaması yaşanmıştır.

DVD'nin daha yüksek kapasitesi CD'lardan üç farklılığından kaynaklanmaktadır (Şekil 6.11):

1. Bitler bir DVD'de daha sıkı paketlenmiştir. Bir CD üzerindeki spiralin halkaları arasındaki boşluk 1,6 mm'dir ve spiral boyunca çukurlar arasındaki minimum mesafe 0,834 mm'dır.

DVD daha kısa dalga boyuna sahip bir lazer kullanmakta ve 0,74 mm'lik bir döngü aralığı ve 0,4 mm'lik çukurlar arasında minimum mesafe elde etmektedir. Bu iki iyileştirmenin sonucu olarak kapasite yaklaşık yedi kat artarak 4.7 GB'a ulaşmıştır.

2. DVD, birinci katmanın üzerinde ikinci bir çukur ve toprak katmanı kullanır. Çift katmanlı bir DVD'de yansıtıcı katmanın üstünde yarı yansıtıcı bir katman bulunur ve odağı ayarlayarak DVD sürücülerindeki lazerler her katmanı ayrı ayrı okuyabilir. Bu teknik diskin kapasitesini neredeyse iki katına, yaklaşık 8,5 GB'a çıkarır. İkinci katmanın daha düşük yansıtıcılığı depolama kapasitesini sınırlar, böylece tam bir ikiye katlama sağlanamaz.



**Şekil 6.11** CD-ROM ve DVD-ROM

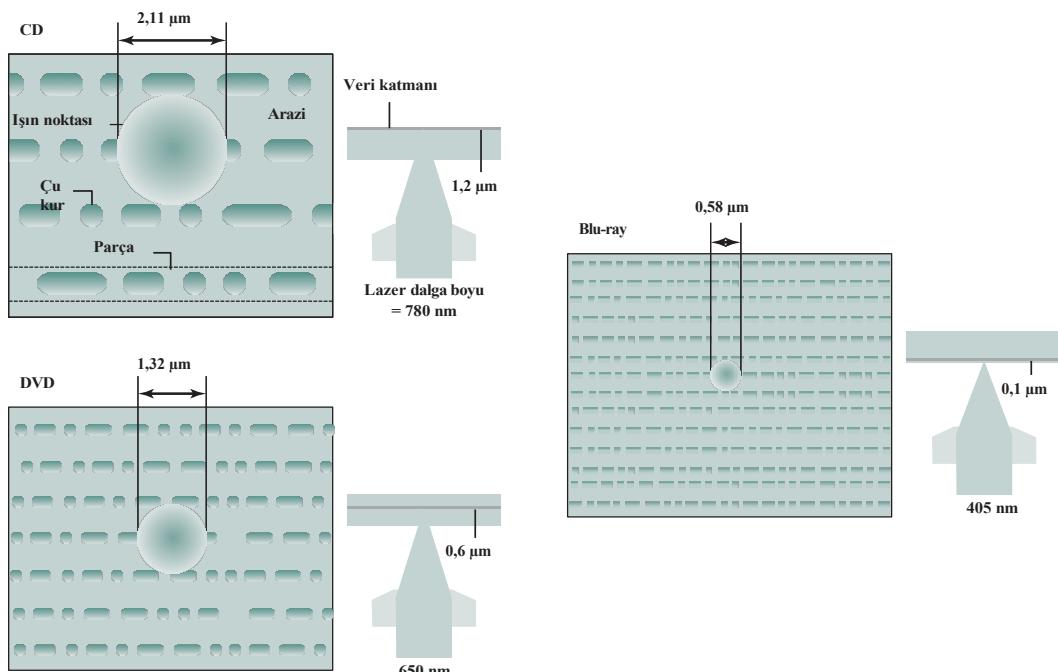
**3. DVD-ROM** iki taraflı olabilirken, veriler bir CD'nin yalnızca bir tarafına kaydedilir. Bu da toplam kapasiteyi 17 GB'a çıkarır.

CD'de olduğu gibi, DVD'lerin de hem yazılabilir hem de salt okunur sürümleri vardır (Tablo 6.6).

### **Yüksek Çözünürlüklü Optik Diskler**

Yüksek çözünürlüklü optik diskler, yüksek çözünürlüklü videoları depolamak ve DVD'lere kıyasla önemli ölçüde daha yüksek depolama kapasitesi sağlamak üzere tasarlanmıştır. Daha yüksek bit yoğunluğu, mavi-mor aralığında daha kısa dalga boyuna sahip bir lazer kullanılarak elde edilir. Dijital 1'leri ve 0'ları oluşturan veri çukurları, daha kısa lazer dalga boyu nedeniyle yüksek çözünürlüklü optik disklerde DVD'ye kıyasla daha küçüktür.

İki rakip disk formatı ve teknolojisi başlangıçta pazarda kabul görmek için yarıştı: HD DVD ve **Blu-ray** DVD. Blu-ray şemasi nihayetinde pazar hakimiyetini elde etti. HD DVD şemasi tek bir tarafta tek bir katman üzerinde 15 GB depolayabilir. Blu-ray, disk üzerindeki veri katmanını lazere daha yakın konumlandırmaktadır (Şekil 6.12'deki her bir diyagramın sağ tarafında gösterilmektedir). Bu, daha sıkı bir odaklanma ve daha az bozulma ve dolayısıyla daha küçük çukurlar ve izler sağlar. Blu-ray tek bir katmanda 25 GB depolayabilir. Üç versiyonu mevcuttur: sadece okunabilir (BD-ROM), bir kez kaydedilebilir (BD-R) ve yeniden kaydedilebilir (BD-RE).



Şekil 6.12 Optik Bellek Özellikleri

## 6.5 MANYETİK BANT

Teyp sistemleri disk sistemleri ile aynı okuma ve kayıt tekniklerini kullanır. Ortam, mıknatışlanabilir malzeme ile kaplanmış esnek polyester (bazı giysilerde kullanılan benzer) banttır. Kaplama, özel bağlayıcılar içindeki saf metal parçacıklarından veya buharla kaplanmış metal filmlerden oluşabilir. Teyp ve teyp sürücüsü, bir ev teyp kayıt sistemine benzer. Bant genişlikleri 0,38 cm (0,15 inç) ile 1,27 cm (0,5 inç) arasında değişir. Bantlar eskiden kullanım için ikinci bir milden geçirilmesi gereken açık makaralar olarak paketlenirdi. Günümüzde neredeyse tüm bantlar kartuşlar içinde bulunmaktadır.

Teyp üzerindeki veriler, uzunlamasına uzanan bir dizi paralel iz olarak yapılandırılmıştır. Daha önceki teyp sistemlerinde genellikle dokuz iz kullanılmıştır. Bu, dokuzuncu iz olarak ek bir eşlik biti ile her seferinde bir bayt veri depolamayı mümkün kılmıştır. Bunu, dijital bir kelime veya çift kelimeye karşılık gelen 18 veya 36 iz kullanan teyp sistemleri takip etti. Verilerin bu biçimde kaydedilmesi **paralel kayıt olarak** adlandırılır. Çoğu modern sistemi bunun yerine, manyetik disklerde olduğu gibi, verilerin her bir iz boyunca bir dizi bit olarak yerleştirildiği **seri kayıt** kullanır. Diskte olduğu gibi, veriler bir kaset üzerinde **fiziksel kayıtlar** olarak adlandırılan bitişik bloklar halinde okunur ve yazılır. Teyp üzerindeki bloklar, **kayıtlar arası** olarak adlandırılan boşluklar ayrılr. Diskte olduğu gibi, teyp de fiziksel kayıtların bulunmasına yardımcı olmak için biçimlendirilir.

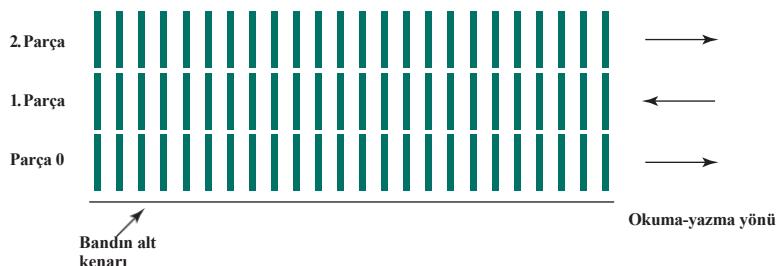
Seri bantlarda kullanılan tipik kayıt tekniği **serpen-tine kayıt** olarak adlandırılır. Bu teknikte, veriler kaydedilirken, ilk bit seti bandın tüm uzunluğu boyunca kaydedilir. Bandın sonuna ulaşıldığında,

kafalar yeni bir parça kaydetmek için yeniden konumlandırılır ve bant tekrar tüm uzunluğu boyunca, bu kez ters yönde kaydedilir. Bu işlem bant dolana kadar ileri geri devam eder (Şekil 6.13a). Hızı artırmak için, okuma-yazma kafası aynı anda bir dizi bitişik izi okuyabilir ve yazabilir (tipik olarak iki ila sekiz iz). Veriler hala tek tek izler boyunca seri olarak kaydedilir, ancak Şekil 6.13b'de gösterildiği gibi sıralı bloklar bitişik izlerde saklanır.

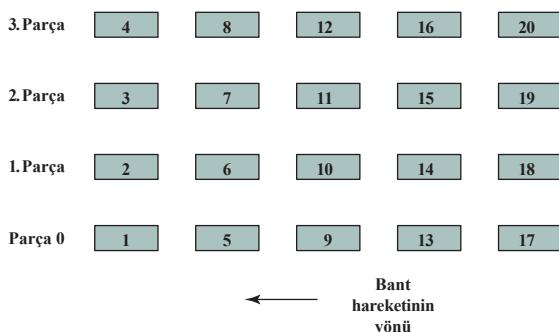
Bir teyp sürücüsü *sıralı erişimli* bir cihazdır. Teyp kafası kayıt 1'de konumlandırılmışsa,  $N$  kaydını okumak için her bir tane olmak üzere 1'den  $N - 1$ 'e kadar fiziksel kayıtları okumak gereklidir. Eğer kafa o anda istenen kaydın ötesinde konumlanmışsa, bandı belirli bir mesafe geri sarmak ve ileriye doğru okumaya başlamak gereklidir. Diskten farklı olarak, kaset yalnızca okuma veya yazma işlemi sırasında hareket halindedir.

Teybin aksine, disk sürücüsü *doğrudan erişimli* bir cihaz olarak adlandırılır. Bir disk sürücüsünün istenilen sektörre ulaşmak için bir diskteki tüm sektörleri sırayla okuması gerekmektedir. Yalnızca bir parça içindeki araya giren sektörleri beklemek zorundadır ve herhangi bir parçaya art arda erişim sağlanabilir.

Manyetik bant ilk ikincil bellek türüydü. Bellek hiyerarşisinin en düşük maliyetli, en yavaş hızlı üyesi olarak hala yaygın şekilde kullanılmaktadır.



(a) Serpentine okuma ve yazma



(b) Dört parçayı aynı anda okuyan-yazan sistem için blok düzeni

**Şekil 6.13** Tipik Manyetik Bant Özellikleri

**Tablo 6.7** LTO Teyp Sürücülerü

|                            | LTO-1   | LTO-2   | LTO-3    | LTO-4    | LTO-5    | LTO-6    | LTO-7    | LTO-8     |
|----------------------------|---------|---------|----------|----------|----------|----------|----------|-----------|
| Çıkış tarihi               | 2000    | 2003    | 2005     | 2007     | 2010     | 2012     | TBA      | TBA       |
| Sıkıştırılmış kapasite     | 200 GB  | 400 GB  | 800 GB   | 1600 GB  | 3,2 TB   | 8 TB     | 16 TB    | 32 TB     |
| Sıkıştırılmış aktarım hızı | 40 MB/s | 80 MB/s | 160 MB/s | 240 MB/s | 280 MB/s | 400 MB/s | 788 MB/s | 1.18 GB/s |
| Doğrusal yoğunluk (bit/mm) | 4880    | 7398    | 9638     | 13,250   | 15,142   | 15,143   |          |           |
| Bant izleri                | 384     | 512     | 704      | 896      | 1280     | 2176     |          |           |
| Bant uzunluğu (m)          | 609     | 609     | 680      | 820      | 846      | 846      |          |           |
| Bant genişliği (cm)        | 1.27    | 1.27    | 1.27     | 1.27     | 1.27     | 1.27     |          |           |
| Öğeleri yazın              | 8       | 8       | 16       | 16       | 16       | 16       |          |           |
| SOLUCAN MI?                | Hayır   | Hayır   | Evet     | Evet     | Evet     | Evet     | Evet     | Evet      |
| Şifreleme Özelliği Var mı? | Hayır   | Hayır   | Hayır    | Evet     | Evet     | Evet     | Evet     | Evet      |
| Bölme mi?                  | Hayır   | Hayır   | Hayır    | Hayır    | Evet     | Evet     | Evet     | Evet      |

Günümüzün baskın teyp teknolojisi lineer teyp-açık (LTO) olarak bilinen bir kartuş sistemidir. LTO 1990'larda piyasadaki çeşitli tescilli sistemlere açık kaynaklı bir alternatif olarak geliştirilmiştir. Tablo 6.7 çeşitli LTO nesilleri için parametrelerini göstermektedir. Ayrintılar için Ek J'ye bakınız.

## 6.6 ANAHTAR TERİMLER, TEKRAR SORULARI VE PROBLEMLER

### Anahtar Terimler

|                             |                           |                         |
|-----------------------------|---------------------------|-------------------------|
| erişim süresi               | DVD-RW                    | optik bellek            |
| Blu-ray                     | sabit kafalı disk         | Çukur                   |
| CD                          | flaş bellek               | Tabak                   |
| CD-R                        | disket                    | RAID                    |
| CD-ROM                      | boşluk                    | çıkarılabilir disk      |
| CD-RW                       | sabit disk sürücüsü (HDD) | rotasyonel gecikme      |
| sabit açısal hız<br>(CAV)   | Kafa                      | sektör                  |
| sabit doğrusal hız<br>(CLV) | kara                      | arama süresi            |
| Silindir                    | manyetik disk             | serpentine kayıt        |
| DVD                         | manyetik bant             | katı hal sürücüsü (SSD) |
| DVD-R                       | manyetorezistif           | çizgili veri            |
| DVD-ROM                     | hareketli kafalı disk     | substrat                |
|                             | çoklu bölge kaydı         | parça                   |
|                             | çıkarılamayan disk        | transfer süresi         |

## İnceleme Soruları

- 6.1** Manyetik disk için cam alt tabaka kullanmanın avantajları nelerdir?
- 6.2** Veriler bir manyetik diske nasıl yazılır?
- 6.3** Manyetik bir diskten veriler nasıl okunur?
- 6.4** Basit bir CAV sistemi ile çok bölgeli bir kayıt sistemi arasındaki farkı açıklayınız.
- 6.5** Ray, silindir ve sektör terimlerini tanımlayın.
- 6.6** Tipik disk sektörü boyutu nedir?
- 6.7** Arama süresi, dönme gecikmesi, erişim süresi ve aktarım süresi terimlerini tanımlayınız.
- 6.8** Tüm RAID seviyeleri tarafından paylaşılan ortak özellikler nelerdir?
- 6.9** Yedi RAID seviyesini kısaca tanımlayınız.
- 6.10** Şeritli veri terimini açıklayınız.
- 6.11** Bir RAID sisteminde yedeklilik nasıl sağlanır?
- 6.12** RAID bağlamında, paralel erişim ile bağımsız erişim arasındaki fark nedir?
- 6.13** CAV ve CLV arasındaki fark nedir?
- 6.14** CD ve DVD arasındaki hangi farklar ikincisinin daha büyük kapasiteye sahip olmasını açıklar?
- 6.15** Serpentin kaydını açıklayın.

## Problemler

- 6.1** Denklem 6.1'i gerekçelendirin. Yani, denklemin sağ tarafındaki üç terimin her birinin sol taraftaki değere nasıl katkıda bulunduğu açıklayınız.
- 6.2** O'dan ( $N - 1$ )'e kadar numaralandırılmış  $N$  izli bir disk düşünün ve istenen sektörlerin disk üzerinde rastgele ve eşit olarak dağıtıldığını varsayıñ. Bir aramanın geçtiği ortalama iz sayısını hesaplamak istiyoruz.
  - a.** İlk olarak, kafa şu anda  $t$  izi üzerindeyken  $j$  uzunluğunda bir aramanın olasılığını hesaplayın. *İpucu:* Bu, aramanın hedefi için tüm iz konumlarının eşit olasılıkta olduğunu kabul ederek toplam kombinasyon sayısını belirleme meselesiştir.
  - b.** Ardından,  $K$  uzunluğundaki bir aramanın olasılığını hesaplayın. *İpucu:* Bu,  $K$  parçasının hareketlerinin tüm olası kombinasyonlarının toplanmasını içerir.
  - c.** Beklenen değer formülünü kullanarak bir aramanın kat ortalama yol sayısını hesaplayın

$$E[x] = \sum_{i=0}^{N-1} i * \Pr[x=i]$$

$$\text{İpucu: Eşitlikleri kullanın: } \sum_{i=1}^n = \frac{n(n+1)}{2}; \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

- d.**  $N$ 'nin büyük değerleri için, bir aramanın geçtiği ortalama iz sayısının  $N/3$ 'e yaklaştığını gösterin.
- 6.3** Bir disk sistemi için aşağıdakileri tanımlayın:
 

$t_s$ = görme süresi; kafayı iz üzerinde konumlandırmak için ortalama süre  $r$ = diskin dönüş hızı, saniyedeki devir cinsinden  $n$ = sektör başına bit sayısı

$N$ = bir hattın bit cinsinden kapasitesi

$t_{sektor}$ = bir sektörde erişim süresi

Diger parametrelerin bir fonksiyonu olarak  $t_{sektor}$  için bir formül geliştirin.

- 6.4** Bir manyetik disk sürücüsünün 8 yüzeyi, yüzey başına 512 izi ve iz başına 64 sektörü olduğunu düşünün. Sektör boyutu 1 kB'dır. Ortalama arama süresi 8 ms, izden iz erişim süresi 1,5 ms'dir ve sürücü 3600 rpm'de dönmektedir. Bir silindirdeki ardışık izler kafa hareketi olmadan okunabilir.
- Disk kapasitesi nedir?
  - Ortalama erişim süresi nedir? Bu dosyanın *i* silindirinin 0. sektörü, 0. izinden başlayarak birbirini izleyen silindirlerin birbirini izleyen sektörlerinde ve izlerinde depolandığını varsayıñ.
  - 5-MB'lık bir dosyayı aktarmak için gereken süreyi tahmin edin.
  - Seri aktarım hızı nedir?
- 6.5** Aşağıdaki parametrelerle sahip tek plakalı bir disk düşünün: dönüş hızı: 7200 rpm; plakanın bir tarafındaki iz sayısı: 30.000; parça başına sektör sayısı: 600; arama süresi: geçen her yüz iz için bir ms. Disk, rastgele bir iz üzerinde rastgele bir sektörde erişim isteği alıñ ve disk kafasının 0. izden başladığını varsayıñ.
- Ortalama arama süresi nedir?
  - Ortalama dönme gecikmesi nedir?
  - Bir sektör için transfer süresi nedir?
  - Bir talebi karşılamak için toplam ortalama süre nedir?
- 6.6** Fiziksel kayıtlar ile mantıksal kayıtlar arasında bir ayırım yapılır. **Mantıksal kayıt**, bilginin nasıl veya nerede depolandığından bağımsız olarak kavramsal bir birim olarak ele alınan ilgili veri öğelerinin bir koleksiyonudur. **Fiziksel kayıt**, depolama aygıtinin ve işletim sisteminin özellikleri tarafından tanımlanan bitişik bir depolama alanıdır. Her bir fiziksel kaydın otuz adet 120 baytlik mantıksal içerdiği bir disk sistemi varsayılm. Disk 512 bayt/saniye ile sabit sektörlü, 96 sektör/iz, yüzey başına 110 iz ve 8 kullanılabılır yüzeye 300.000 mantıksal kaydı depolamak için ne kadar disk alanı (sektör, iz ve yüzey olarak) gerekeceğini hesaplayın. Dosya başlık kayıtlarını ve iz indekslerini göz ardı edin ve kayıtların iki sektörde yayılmasını varsayıñ.
- 6.7** Dakikada 3600 devirle dönen bir disk düşünün. Kafayı bitişik izler arasında hareket ettirmek için arama süresi 2 ms'dir. Iz başına 32 sektör vardır ve bunlar sektör 0'dan sektör 31'e kadar doğrusal sırada depolanır. Kafa, sektörleri artan sırada görür. Okuma/yazma kafasının 8. izdeki 1. sektörün başlangıcında konumlandığını varsayıñ. Büttün bir izi tutabilecek bütyüklükte bir ana bellek tamponu vardır. Veriler disk konumları arasında kaynak izden ana bellek tamponuna okunarak ve ardından veriler tampondan hedef izे yazılarak aktarılır.
- Hat 8 üzerindeki sektör 1'i hat 9 üzerindeki sektör 1'e aktarmak ne kadar sürer?
  - İz 8'in tüm sektörlerini iz 9'un ilgili sektörlerine aktarmak ne kadar sürer?
- 6.8** Disk şeritlemenin, şerit boyutu G/C isteği boyutuna kıyasla küçük olduğunda veri aktarım hızını artırabilecegi açık olmalıdır. RAID 0'in tek bir büyük diske göre daha iyi performans sağladığı da açık olmalıdır, çünkü birden fazla G/C isteği paralel olarak işlenebilir. Ancak, bu ikinci durumda, disk şeritleme gerekli midir? Yani, disk şeritleme, şeritleme olmadan karşılaşılabilir bir disk dizisine kıyasla G/C istek hızı performansını artırır mı?
- 6.9** 4 sürücülü, sürücü başına 200 GB'lık bir RAID dizisi düşünün. RAID düzeyleri 0, 1, 3, 4, 5 ve 6'nın her biri için kullanılabilir veri depolama kapasitesi nedir?
- 6.10** Bir kompakt disk için ses 16 bitlik örneklerle dijital dönüştürülür ve depolama için 8 bitlik baytlardan oluşan bir akış olarak elde edilir. Doğrudan kayıt olarak adlandırılan bu verileri depolamak için basit bir şema, 1'i bir kara ve 0'ı bir çukur ile temsil etmek olacaktır. Bunun yerine, her bayt 14 bitlik bir ikili sayıya genişletilir. Toplam  $16 \cdot 134 \cdot (2^{14})$  14 bitlik sayının tam  $256^2$ 'sında ( $2^8$ ) her 1 çifti arasında en az iki 0 olduğu ortaya çıkmıştır ve bunlar 8'den 14 bite genişletme için seçilen sayılardır. Optik sistem, çukurdan karaya veya karadan çukura geçisi tespit ederek 1'lerin varlığını algılar. 0'ları ise yoğunluk değişimleri arasındaki mesafeleri ölçerek tespit eder. Bu şema arka arkaya 1'lerin olmamasını gerektirir, dolayısıyla 8'den 14'e kodunun kullanılması gerekip.

Bu şemanın avantajı aşağıdaki gibidir. Belirli bir lazer işini çapı için, bitlerin nasıl temsil edildiğine bakılmaksızın minimum bir çukur boyutu vardır. Bu şema ile, bu minimum çukur boyutu 3 bit saklar, çünkü her 1'i en az iki 0 takip eder. Doğrudan kayıt ile aynı çukur sadece bir bit depolayabilir. Hem çukur başına depolanan bit sayısı hem de 8-14 bit genişlemesi göz önüne alındığında, hangi şema en fazla biti ve hangi faktörle depolar?

- 6.11** Bir bilgisayar sistemi için yedekleme stratejisi tasarlayın. Bir seçenek, her biri 500 GB sürücü için 150 ABD doları olan takılabilir harici diskler kullanmaktadır. Başka bir seçenek de şu fiyatta bir teyp sürücüsü satın almaktır 2500 dolar ve 400 GB'lık kasetlerin tanesi 50 dolar. (Bunlar 2008'deki gerçekçi fiyatlardır.) Tipik bir yedekleme stratejisi, sahada iki set yedekleme medyası bulundurmaktır; yedekler dönüşümlü olarak bunlara yazılır, böylece sistem bir yedekleme yaparken arızalanırsa, önceki sürüm hala sağlam kalır. Ayrıca tesis dışında tutulan üçüncü bir set vardır tesisındaki set periyodik olarak tesis içindeki setle değiştirilir.
- a.** Yedeklenecek 1 TB (1000 GB) veriniz olduğunu varsayıñ. Bir disk yedekleme sistemi ne kadara mal olur?
  - b.** Bir teyp yedekleme sistemi 1 TB için ne kadara mal olur?
  - c.** Bir teyp stratejisinin daha ucuz olması için her bir yedeklemenin ne kadar büyük olması gereklidir?
  - d.** Ne tür bir yedekleme stratejisi kasetleri tercih eder?

# 7

## BÖLÜM

### GİRİŞ/ÇIKIŞ

- 7.1** Harici Cihazlar
- 7.2** I/O Modülleri
- 7.3** Programlanmış G/Ç
- 7.4** Kesme Tahrikli I/O
- 7.5** Doğrudan Bellek Erişimi
- 7.6** Doğrudan Önbellek Erişimi
- 7.7** G/Ç Kanalları ve İşlemciler
- 7.8** Harici Arabağlantı Standartları
- 7.9** IBM zEnterprise EC12 G/Ç Yapısı
- 7.10** Anahtar Terimler, Gözden Geçirme Soruları ve Problemler

## ÖĞRENİM HEDEFLERİ

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- Bilgisayar organizasyonunun bir parçası olarak I/O modüllerinin kullanımını açıklayınız.
- Programlanmış G/C ve kesme güdümlü G/C arasındaki farkı anlayın ve bunların göreceli değerlerini tartışın.
- Doğrudan bellek erişiminin işleyişine genel bir bakış .
- Doğrudan önbellek erişimine genel bir bakış sunar.
- I/O kanallarının işlevini ve kullanımını açıklayın.

## I/O Sistem Tasarım Aracı

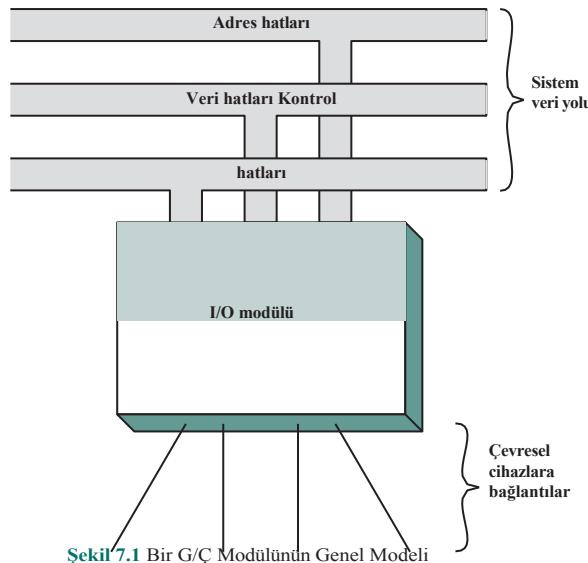


İşlemci ve bir dizi bellek modülüne ek olarak, bir bilgisayar sisteminin üçüncü temel unsuru bir dizi I/O modülüdür. Her modül sistem veri yoluna ya da merkezi anahtara arayüz oluşturur ve bir ya da daha fazla çevresel aygıtı kontrol eder. Bir G/C modülü sadece bir cihazı sistem veri yoluna bağlayan bir dizi mekanik konektörden ibaret değildir. Bunun yerine, G/C modülü çevre birimi ile veri yolu arasında bir iletişim işlevi gerçekleştirmek için mantık içerir.

Okuyucu, çevre birimlerinin neden doğrudan sistem veri yoluna bağlanmadığını merak edebilir. Bunun nedenleri aşağıdaki gibidir:

- Çeşitli çalışma yöntemlerine sahip çok çeşitli çevre birimleri vardır. Bir dizi cihazı kontrol etmek için gerekli mantığı işlemciye dahil etmek pratik olmayacaktır.
- Çevre birimlerinin veri aktarım hızı genellikle bellek veya işlemcininkinden çok daha yavaştır. Bu nedenle, bir çevre birimiyle doğrudan iletişim kurmak için yüksek hızlı sistem kullanmak pratik değildir.
- Öte yandan, bazı çevre birimlerinin veri aktarım hızı bellek ya da işlemcininkinden daha hızlıdır. Yine bu uyumsuzluk, doğru yönetilmemiği takdirde verimsizliğe yol açabilir.
- Çevre birimleri genellikle bağlı oldukları bilgisayardan farklı veri formatları ve kelime uzunlukları kullanır.
  - Bu nedenle, bir I/O modülü gereklidir. Bu modülün iki ana işlevi vardır (Şekil 7.1):
    - Sistem veriyolu veya merkezi anahtar üzerinden işlemci ve belleğe arayüz.
    - Özel veri bağlantıları ile bir veya daha fazla çevresel cihaza arayüz.

Bu bölüme harici aygıtlar hakkında kısa bir tartışma ile başlıyoruz, ardından bir I/O modülünün yapısı ve işlevine genel bir bakış sunuyoruz. Daha sonra G/C işlevinin işlemci ve bellek ile işbirliği içinde gerçekleştirilebileceği çeşitli yollara bakacağız: dahili G/C arayüzü. Daha sonra, bazı



doğrudan bellek erişimi ve daha yeni bir yenilik olan doğrudan önbellek erişimi. Son olarak, I/O modülü ile dış dünya arasındaki harici I/O arayüzünü inceleyeceğiz.

## 7.1 harici cihazlar

İşlemleri, dış ortam ile bilgisayar arasında veri alışverişi sağlayan çok çeşitli harici aygıtlar aracılığıyla gerçekleştiriliyor. Harici bir aygit, bir G/C modülüne bağlanarak bilgisayara bağlanır (Şekil 7.1). Bağlantı, G/C modülü ile harici aygit arasında kontrol, durum ve veri alışverişi için kullanılır. Bir G/C modülüne bağlı harici bir aygit genellikle *çevresel aygit* ya da basitçe *aygit* olarak adlandırılır.

Harici cihazları genel olarak üç kategoride sınıflandırabiliriz:

- **İnsan tarafından okunabilir:** Bilgisayar kullanıcısı ile iletişim kurmaya uygun;
- **Makine tarafından okunabilir:** Ekipmanla iletişim kurmak için uygundur;
- **Haberleşme:** Uzak cihazlarla iletişim kurmak için uygundur.

İnsan tarafından okunabilen cihazlara örnek olarak video görüntü terminalleri (VDT'ler) ve yazıcılar verilebilir. Makine tarafından okunabilen aygıtlara örnek olarak manyetik disk ve teyp sistemleri ile robotik uygulamlarda kullanılan sensörler ve aktüatörler verilebilir. Bu bölümde disk ve teyp sistemlerini I/O aygıtları olarak gördüğümüzü, oysa Bölüm 6'da bunları bellek aygıtları olarak incelediğimizi unutmayın. İşlevsel açıdan bakıldığında, bu cihazlar bellek hiyerarşisinin bir parçasıdır ve kullanımları Bölüm 6'da uygun bir şekilde ele alınmıştır. Yapısal bir bakış açısıyla, bu cihazlar G/C modülleri tarafından kontrol edilir ve bu nedenle bu bölümde ele alınacaktır.

İletişim cihazları bir bilgisayarın uzaktaki bir cihazla veri alışverişi yapmasını sağlar; bu cihaz terminal, makine tarafından okunabilir bir cihaz veya başka bir bilgisayar gibi insan tarafından okunabilir bir cihaz olabilir.

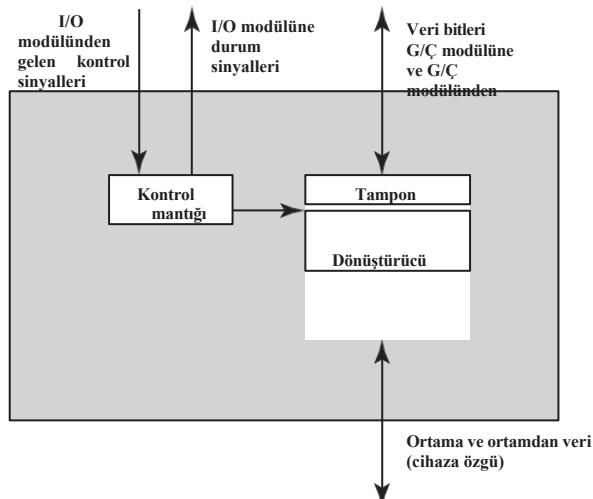
Çok genel anlamda, harici bir cihazın yapısı Şekil 7.2'de gösterilmiştir. I/O modülünün arayüzü kontrol, veri ve durum sinyalleri şeklindedir. *Kontrol sinyalleri*, I/O modülüne veri göndermek (INPUT veya READ), I/O modülünden veri kabul etmek (OUTPUT veya WRITE), durumu bildirmek veya cihaza özel bir kontrol işlevi gerçekleştirmek (örneğin, bir disk kafasını konumlandırmak) gibi cihazın gerçekleştireceği işlevi belirler. *Veriler*, G/C modülüne gönderilecek veya G/C modülünden alınacak bir dizi bit şeklindedir. *Durum sinyalleri* cihazın durumunu gösterir. Örnekler, cihazın veri aktarımı için hazır olup olmadığını göstermek için HAZIR/HAZIR DEĞİL şeklinde dir.

Cihazla ilişkili *kontrol mantığı*, I/O modülünden gelen yönlendirmeye yanıt olarak cihazın çalışmasını *dönüştürücü* eder. Verileri çıkış sırasında elektrikten diğer enerji biçimlerine ve giriş sırasında diğer biçimlerden elektriğe dönüştürür. Tipik olarak, I/O modülü ile dış ortam arasında aktarılan verileri geçici olarak tutmak için dönüştürücü ile bir tampon ilişkilendirilir. Seri için 8 ila 16 bitlik bir tampon boyutu yaygınken, disk sürücüsü denetleyicileri gibi blok odaklı aygıtlar çok daha büyük tamponlara sahip olabilir.

I/O modülü ile harici cihaz arasındaki arayüz Bölüm 7.7'de inceleneciktir. Harici cihaz ve ortam arasındaki arayüz bu kitabın kapsamı dışındadır, ancak burada birkaç kısa örnek verilmiştir.

### Klavye/Monitör

Bilgisayar/kullanıcı etkileşiminin en yaygın aracı klavye/monitör düzenlemesidir. Kullanıcı klavye aracılığıyla girdi sağlar, girdi daha sonra bilgisayara aktarılır ve monitörde görüntülenebilir. Buna ek olarak bilgisayar tarafından sağlanan verileri görüntüler.



Şekil 7.2 Harici Bir Cihazın Blok Diyagramı

Temel değişim birimi karakterdir. Her karakterle ilişkili, tipik olarak 7 veya 8 bit uzunlığında bir kod vardır. En yaygın kullanılan metin kodu Uluslararası Referans Alfabetesidir (IRA).<sup>1</sup> Bu koddaki her karakter benzersiz bir 7 bitlik ikili kodla temsil edilir; böylece 128 farklı karakter temsil edilebilir. Karakterler iki tiptir: yazdırılabilir ve kontrol. Yazdırılabilir karakterler kağıda basılabilen ya da ekranada görüntülenebilen alfabetik, sayısal ve özel karakterlerdir. Kontrol karakterlerinin bazıları karakterlerin yazdırılmasını ya da görüntülenmesini kontrol etmekle ilgilidir; satır başı buna bir örnektir. Diğer kontrol karakterleri iletişim prosedürleriyle ilgilidir. Ayrıntılar için Ek H'ye bakın.

Klavye girişi için, kullanıcı bir tuşa bastığında, bu klavyeden dönüştürücü tarafından yorumlanan ve ilgili IRA kodunun bit modeline çevrilen bir elektronik sinyal üretir. Bu bit deseni daha sonra bilgisayardaki I/O modülüne aktarılır. Bilgisayarda metin aynı IRA kodunda saklanabilir. Çıkışta, IRA kodu karakterleri I/O modülünden harici bir cihaza iletilir. Cihazdaki dönüştürücü bu kodu yorumlar ve belirtilen karakteri görüntülemek ya da istenen kontrol işlemini gerçekleştirmek için çıkış cihazına gerekli elektronik sinyalleri gönderir.

### Disk Sürücüsü

Bir disk sürücüsü, bir G/Ç modülü ile veri, kontrol ve durum sinyalleri alışverişi için elektronik aksamın yanı sıra disk okuma/yazma mekanizmasını kontrol etmek için elektronik aksam içerir. Sabit kafalı bir diskte dönüştürücü, hareketli disk yüzeyindeki manyetik desenler ile cihazın tamponundaki bitler arasında dönüşüm yapabilmektedir (Şekil 7.2). Hareketli kafalı bir disk ayrıca disk kolumnun disk yüzeyi boyunca radyal olarak içeri ve dışarı hareket etmesine neden olabilmelidir.

## 7.2 G/Ç MODÜLLERİ

### Modül İşlevi

Bir I/O modülünün başlıca işlevleri veya gereksinimleri aşağıdaki kategorilere ayrılır:

- Kontrol ve zamanlama
- İşlemci iletişimi
- Cihaz iletişimi
- Veri tamponlama
- Hata algılama

Herhangi bir boyunca, işlemci, programın aşağıdakilere olan ihtiyacına bağlı olarak, öngörülemeyen şekillerde bir veya daha fazla harici cihazla iletişim kurabilir

---

<sup>1</sup>IRA ITU-T Recommendation T.50'de tanımlanmıştır ve daha önce International Alphabet Number 5 (IA5) olarak bilinmektedir. IRA'nın ABD ulusal versiyonu Bilgi Değişimi için Amerikan Standart Kodu (ASCII) olarak adlandırılır.

I/O. Ana bellek ve sistem veri yolu gibi dahili kaynaklar, veri G/C dahil olmak üzere bir dizi faaliyet arasında paylaşılmalıdır. Bu nedenle, G/C işlevi, dahili kaynaklar ve harici cihazlar arasındaki trafik akışını koordine etmek için bir **kontrol ve zamanlama** gereksinimi içerir. Örneğin, harici bir cihazdan işlemciye veri aktarımının kontrolü aşağıdaki adımlar dizisini içerebilir:

- 1.** İşlemci, bağlı cihazın durumunu kontrol etmek için G/C modülünü sorgular.
- 2.** G/C modülü cihaz durumunu döndürür.
- 3.** Cihaz çalışır durumdaysa ve iletme hazırlrsa, işlemci I/O modülüne bir komut göndererek veri aktarımını talep eder.
- 4.** G/C modülü harici cihazdan bir veri birimi (örneğin 8 veya 16 bit) alır.
- 5.** Veriler I/O modülünden işlemciye aktarılır.

Sistem bir veri yolu kullanıyorsa, işlem ve I/O modülü arasındaki etkileşimlerin her biri bir veya daha fazla veri yolu tahkimini içerir.

Yukarıdaki basitleştirilmiş senaryo, G/C modülünün işlemci ve harici cihazla iletişim kurması gerektiğini göstermektedir. **İşlemci iletişimi** aşağıdakileri içerir:

- **Komut kod çözme:** G/C modülü işlemciden gelen ve genellikle kontrol veriyolunda sinyal olarak gönderilen komutları kabul eder. Örneğin, bir disk sürücüsü için bir G/C modülü aşağıdaki komutları kabul edebilir: READ SECTOR, WRITE SECTOR, SEEK track number ve SCAN record ID. Son iki komutun her biri veri yolu üzerinde gönderilen bir parametre içerir.
- **Veri:** Veri veri yolu üzerinden işlemci ve I/O modülü arasında veri alışverişi .
- **Durum raporlama:** Çevre birimleri çok yavaş olduğundan, G/C modülünün durumunu bilmek önemlidir. Örneğin, bir G/C modülünden işlemciye veri göndermesi (okuma) istendiğinde, hala önceki G/C komutu üzerinde çalıştığı için bunu yapmaya hazır olmayabilir. Bu durum bir durum sinyali ile bildirilebilir. Yaygın durum sinyalleri BUSY ve READY'dır. Ayrıca çeşitli hata durumlarını bildiren sinyaller de olabilir
- **Adres tanıma:** Her bellek sözcüğünün bir adresi olduğu gibi, her G/C cihazının da bir adresi vardır. Bu nedenle, bir G/C modülü kontrol ettiği her çevre birimi için benzersiz bir adres tanımlıdır.

Diğer taraftan, I/O modülü **cihaz haberleşmesini** gerçekleştirebilmelidir. Bu iletişim komutları, durum bilgilerini ve verileri içerir (Şekil 7.2).

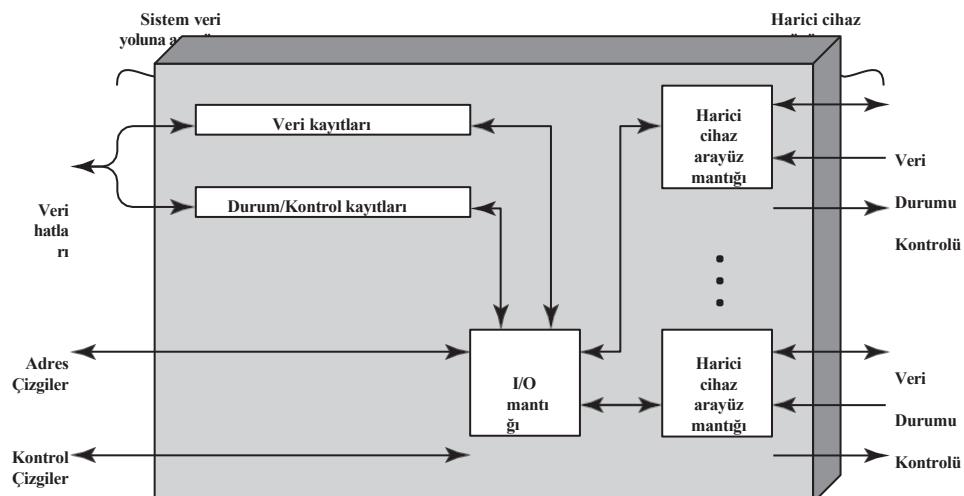
Bir I/O modülünün temel görevlerinden **biri veri tamponlamasıdır**. Bu fonksiyona duyulan ihtiyaç Şekil 2.1'den açıkça görülmektedir. Ana belleğe ya da işlemciye aktarım hızı oldukça yüksekken, birçok çevresel aygit için bu hız çok daha düşüktür ve geniş bir aralığı kapsar. Ana bellekten gelen veriler bir I/O modülüne hızlı bir şekilde . Veriler G/C modülünde arabelleğe alınır ve daha sonra veri hızında çevresel aygit'a gönderilir. Ters yönde, yavaş bir aktarım işleminde belleği bağlamamak için veriler tamponlanır. Böylece

G/C modülü hem cihaz hem de bellek hızlarında çalışabilmelidir. Benzer şekilde, G/C cihazı bellek erişim hızından daha yüksek bir hızda çalışıyorsa, G/C modülü gerekli tamponlama işlemini gerçekleştirir.

Son , bir I/O modülü genellikle **hata tespitinden** ve hataları işlemciye bildirmekten sorumludur. Hataların bir sınıfı, cihaz tarafından bildirilen mekanik ve elektriksel arızaları içerir (örneğin, kağıt sıkışması, bozuk disk izi). Diğer bir sınıf ise cihazdan G/C modülüne iletilirken bit deseninde yapılan kasıtsız değişikliklerden oluşur. İletim hatalarını tespit etmek için genellikle bir hata tespit kodu kullanılır. Basit bir örnek, her veri karakterinde bir eşlik bitinin kullanılmasıdır. Örneğin, IRA karakter kodu bir baytin 7 bitini kaplar. Sekizinci bit, bayttaki toplam 1 sayısı çift (çift eşlik) veya tek (tek eşlik) olacak şekilde ayarlanır. Bir bayt alındığında, G/C modülü bir hata oluşup olmadığını belirlemek için pariteyi kontrol eder.

### I/O Modül Yapısı

G/C modülleri karmaşıklık ve kontrol etikleri harici cihaz sayısı bakımından önemli ölçüde farklılık gösterir. Burada sadece çok genel bir açıklama yapmaya çalışacağız. (Belirli bir aygit olan Intel 8255A, Bölüm 7.4'te açıklanmıştır.) Şekil 7.3'te G/C modülünün genel blok şeması verilmiştir. Modül, bilgisayarın geri kalanına bir dizi sinyal hattı (örneğin, sistem veri yolu hatları) aracılığıyla bağlanır. Modüle ve modülden aktarılan veriler bir veya daha fazla veri kaydında tamponlanır. Ayrıca mevcut durum bilgisi sağlayan bir ya da daha fazla durum kaydı da . Bir durum kaydı, işlemciden ayrıntılı kontrol bilgilerini kabul etmek için bir kontrol kaydı olarak da işlev görebilir. Modül içindeki mantık, bir dizi kontrol hattı aracılığıyla işlemciyle etkileşime girer. İşlemci komutları vermek için kontrol hatlarını kullanır



**Şekil 7.3** Bir G/C Modülünün Blok Diyagramı

I/O modülüne. Kontrol hatlarından bazıları G/C modülü tarafından kullanılabilir (örneğin, tahlim ve durum sinyalleri için). Modül ayrıca kontrol ettiği cihazlarla ilişkili adresleri tanıyalırmalı ve oluşturulabilmelidir. Her G/C modülünün benzersiz bir adresi veya birden fazla harici cihazı kontrol ediyorsa benzersiz bir adres kümesi vardır. Son olarak, G/C modülü kontrol ettiği her bir cihazla arayüze özel mantık içerir.

Bir G/C modülü, işlemcinin çok çeşitli cihazları basit bir şekilde görüntülemesini sağlamak için işlev görür. Sağlanabilecek bir yetenekler yelpazesesi vardır. G/C modülü zamanlama, formatlar ve harici bir cihazın elektromekanığının ayrıntılarını gizleyebilir, böylece işlemci basit okuma yazma komutları ve muhtemelen dosya açma ve kapatma komutları açısından işlev görebilir. En basit haliyle, G/C modülü bir cihazı kontrol etme işinin çoğunu (örneğin, bir kaseti geri sarma) işlemciye bırakabilir.

Ayrıntılı işlem yükünün çoğunu üstlenen ve işlemciye üst düzey bir arayüz sunan bir G/C modülü genellikle *G/C kanalı* veya *G/C işlemcisi* olarak adlandırılır. Oldukça ilkel olan ve ayrıntılı

kontrol gerektiren bir G/C modülü genellikle *G/C denetleyicisi* veya *aygit denetleyicisi* olarak adlandırılır. G/C denetleyicileri genellikle mikrobilgisayarlarda görülürken, G/C kanalları ana bilgisayarlarında kullanılır.

Aşağıda, karışıklığa yol açmadığı durumlarda genel *I/O modülü* terimini kullanacağız ve gerektiğinde daha spesifik terimler kullanacağız.

### 7.3 PrOgRAMlı G/C

G/C işlemleri için üç teknik mümkündür. *Programlanmış I/O* ile veriler işlemci ve I/O modülü arasında değişim gerçekleşir. İşlemci, cihaz durumunu algılama, okuma veya yazma komutu gönderme ve verileri aktarma dahil olmak üzere G/C işleminin doğrudan kontrolünü sağlayan bir program yürütür. İşlemci modülüne bir komut verdiğiinde, G/C işlemi tamamlanana kadar beklemelidir. İşlemci G/C modülünden daha hızlıysa, bu işlemci zamanının boş harcanması anlamına gelir. Kesme güdümlü I/O ile işlemci bir *I/O komutu* verir, diğer komutları yürütmeye devam eder ve I/O modülü işini tamamladığında I/O modülü tarafından kesilir. Hem programlanmış hem de *kesmeli G/C* ile işlemci, çıkış için ana bellekten veri çekmekten ve giriş için ana bellekte veri depolamaktan sorumludur. Bunun alternatifisi **doğrudan bellek erişimi (DMA)** olarak bilinir. Bu modda, G/C modülü ve ana bellek, işlemci müdahalesi olmadan doğrudan veri alışverişinde bulunur.

Tablo 7.1 bu üç teknik arasındaki ilişkiyi göstermektedir. Bu bölümde, programlanmış G/C'yi inceleyeceğiz. Kesme G/C ve DMA aşağıdaki bölümlerde incelenmiştir sırasıyla iki bölümden oluşmaktadır.

**Tablo 7.1** I/O Teknikleri

|                                           | Kesinti Yok       | Kesmelerin Kullanımı          |
|-------------------------------------------|-------------------|-------------------------------|
| İşlemci üzerinden I/O'dan belleğe aktarım | Programlanmış G/C | Kesme güdümlü I/O             |
| Doğrudan G/C'den belleğe aktarım          |                   | Doğrudan bellek erişimi (DMA) |

## Programlanmış G/Ç'ye Genel Bakış

İşlemci bir programı yürütürken G/Ç ilgili bir komutla karşılaşlığında, uygun G/Ç modülüne bir komut vererek bu komutu yürütür. Programlanmış ile, G/Ç modülü istenen eylemi gerçekleştirir ve ardından G/Ç durum kaydında uygun bitleri ayarlar (Şekil 7.3). G/Ç modülü işlemciyi uyarmak için başka bir işlem yapmaz. Özellikle, işlemciyi kesmez. Bu nedenle, işlemin tamamlandığını tespit edene kadar I/O modülünün durumunu periyodik olarak kontrol etmek işlemcinin sorumluluğundadır.

Programlanmış G/Ç tekniniğini açıklamak için, önce işlemci tarafından G/Ç modülüne verilen G/Ç komutları açısından, sonra da işlemci tarafından yürütülen G/Ç talimatları açısından bakacağız.

### G/Ç Komutları

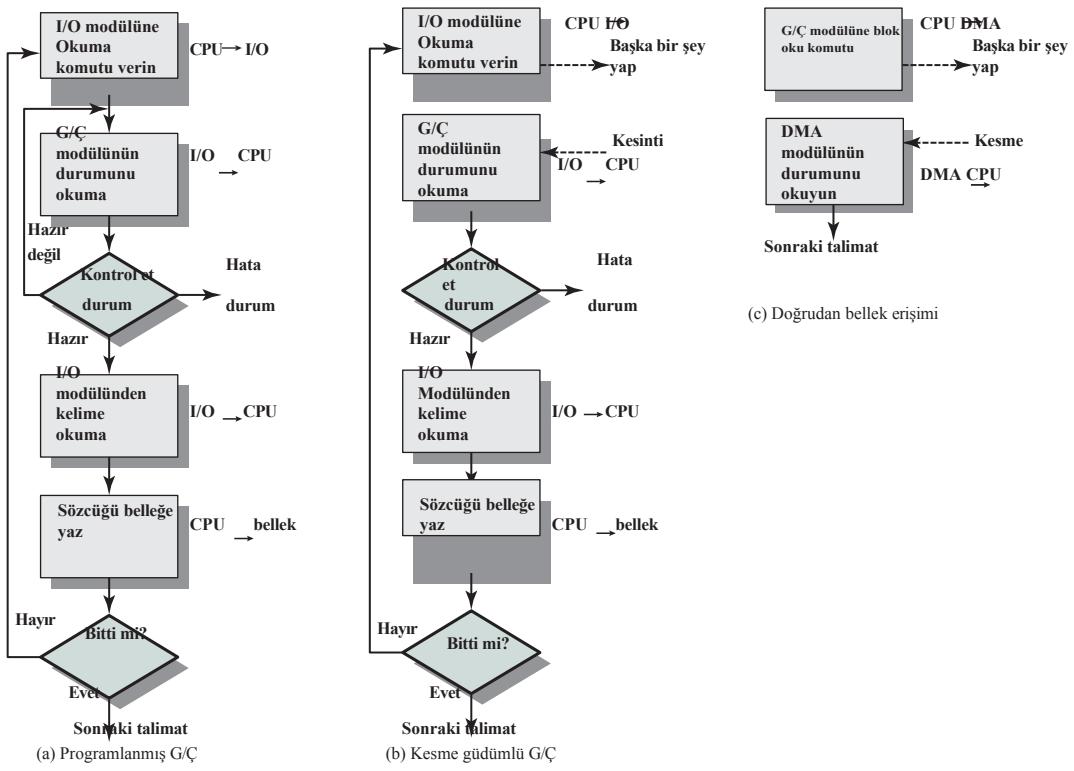
G/Ç ile ilgili bir komutu yürütmek için işlemci, belirli G/Ç modülüne ve harici cihazı belirten bir adres ve bir G/Ç komutu verir. Bir I/O modülünün işlemci tarafından adreslendiğinde alabileceği dört tür I/O komutu vardır:

- **Kontrol:** Bir çevre birimini etkinleştirmek ve ona ne yapacağını söylemek için kullanılır. Örneğin, bir manyetik bant ünitesine geri sarma veya bir kayıt ileri alma talimatı verilebilir. Bu komutlar, belirli bir çevresel aygit türüne göre uyarlanır.
- **Test:** Bir I/O modülü ve çevre birimleriyle ilişkili çeşitli durum koşullarını test etmek için kullanılır. İşlemci, ilgili çevre biriminin açık ve kullanıma hazır olduğunu bilmek isteyecektir. Ayrıca en son G/Ç işleminin tamamlanıp tamamlanmadığını ve herhangi bir hata oluşup olmadığını bilmek isteyecektir.
- **Oku:** G/Ç modülünün çevre biriminden bir veri ögesi alınmasına ve bunu dahili bir tampona yerleştirmesine neden olur (Şekil 7.3'te bir veri kaydı olarak gösterilmiştir). İşlemci daha sonra G/Ç modülünün veri yoluna yerleştirmesini talep ederek veri ögesini elde edebilir.
- **Yazma:** G/Ç modülünün veri yolundan bir veri ögesi (bayt veya kelime) alınmasına ve daha sonra bu veri ögesini çevre birimine iletmesine neden olur.

Şekil 7.4a, bir çevresel aygıtın (örn. teypten bir kayıt) belleğe bir veri bloğu okumak için programlanmış G/Ç kullanımına bir örnek vermektedir. Veriler her seferinde bir kelime (örneğin 16 bit) olarak okunur. Okunan her kelime için işlemci, kelimenin G/Ç modülünün veri kaydında mevcut olduğunu belirleyene kadar bir durum kontrol döngüsünde kalmalıdır. Bu akış şeması, bu tekniğin ana dezavantajını vurgular: işlemciyi gereksiz yere mesgul eden zaman alıcı bir süreçtir.

### G/Ç Talimatları

Programlanmış G/Ç ile, işlemcinin bellekten aldığı G/Ç ile ilgili talimatlar ile işlemcinin talimatları yürütmek için bir G/Ç modülüne verdiği G/Ç komutları arasında yakın bir ilişki vardır., talimatlar kolayca G/Ç komutlarıyla eşleştirilir ve genellikle basit bire bir ilişki vardır. Talimatın şekli, harici cihazların adreslenme şekline bağlıdır.



**Şekil 7.4** Bir Veri Bloğunun Girilmesi için Üç Teknik

Tipik olarak, I/O modülleri aracılığıyla sisteme bağlı birçok I/O cihazı olacaktır. Her cihaza benzersiz bir tanımlayıcı veya adres verilir. İşlemci bir G/C komutu verdiğiinde, komut istenen cihazın adresini içerir. Bu nedenle, her G/C modülü komutun kendisi için olup olmadığını belirlemek için adres satırlarını yorumlamalıdır.

İşlemci, ana bellek ve G/C ortak bir veri yolunu paylaştığında, iki adresleme modu mümkündür: bellek eşlemeli ve yaşılmış. **Bellek eşlemeli G/C** ile bellek konumları ve G/C aygıtları için tek bir adres alanı vardır. Programcı, G/C modüllerinin durum ve veri kayıtlarını bellek konumları olarak ele alır ve hem belleğe hem de G/C aygıtlarına erişmek için aynı makine talimatlarını kullanır. Örneğin, 10 adres hattı ile toplam  $2^{10} = 1024$  bellek konumu ve G/C adresi herhangi bir kombinasyonda desteklenebilir.

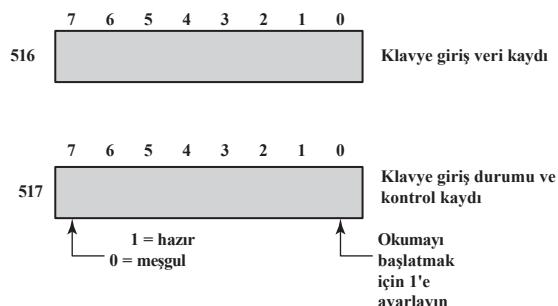
Bellek eşlemeli G/C ile veri yolu üzerinde tek bir okuma hattı ve tek bir yazma hattı gereklidir. Alternatif olarak, veri yolu bellek okuma ve yazma arı̄ giriş ve çıkış komut hatları ile donatılabilir. Komut satırı, adresin bir bellek konumuna mı yoksa bir G/C aygitına mı işaret ettiğini belirtir. Her ikisi için de tam adres aralığı mevcut olabilir. Yine, 10 adres hattı ile sistem artık hem 1024 bellek konumunu hem de 1024 G/C adresini destekleyebilir. G/C için adres alanı bellek için izole edildiğinden, bu **izole G/C** olarak adlandırılır.

## 238 BÖLÜM 7 / GİRİŞ/ÇIKIŞ

Sekil 7.5 bu iki programlanmış G/C teknigini karşılaştırmaktadır. Sekil 7.5a, terminal klavyesı gibi basit bir giriş cihazının arayüzünün bellek eşlemeli G/C kullanan bir programciye nasıl görünebileceğini göstermektedir. 10 bitlik bir adres, 512 bitlik bir bellek (0-511 konumları) ve 512 adede kadar G/C adresi (512-1023 konumları) olduğunu varsayılmıştır. İki adres belirli bir terminalden klavye girişi için ayrılmıştır. Adres 516 veri kaydını, adres 517 ise işlemci komutlarını almak için bir kontrol kaydı olarak da işlev gören durum kaydını ifade eder. Gösterilen program klavyeden 1 baytlık veriyi işlemcideki bir akümülatör kaydına okuyacaktır. İşlemcinin veri baytı mevcut olana kadar döngü yaptığıunu unutmayın.

Yalıtılmış G/C ile (Sekil 7.5b), G/C portlarına yalnızca veri yolundaki G/C komut hatlarını etkinleştiren özel G/C komutları ile erişilebilir.

Çoğu işlemci türü için, belleğe başvurmak için nispeten büyük bir dizi farklı talimat vardır. Eğer yalıtılmış G/C kullanılıyorsa, sadece birkaç G/C talimatı vardır. Bu nedenle, bellek eşlemeli G/C'nin bir avantajı, bu geniş talimat repertuarının daha verimli programlamaya izin vererek kullanılabilirliğidir. Dezavantaj ise değerli bellek adres alanının kullanılmasıdır. Hem bellek eşlemeli hem de yalıtılmış G/C yaygın kullanımdadır.



| ADRES | TALİMAT                 | OPERAND | YORUM                   |
|-------|-------------------------|---------|-------------------------|
| 200   | Yük AC                  | "1"     | Yük akümülatörü         |
|       | Mağaza AC               | 517     | Klavye okumayı başlat   |
| 202   | Yük AC                  | 517     | Durum baytını al        |
|       | İşaret = 0 ise dallanma | 202     | Hazır olana kadar döngü |
|       | Yük AC                  | 516     | Veri baytını yükle      |

(a) Bellek eşlemeli G/C

| ADRES | TALİMAT                 | OPERAND | YORUM                                    |
|-------|-------------------------|---------|------------------------------------------|
| 200   | Yük G/C                 | 5       | Klavye okumayı başlat                    |
| 201   | Test G/C                | 5       | Tamamlanıp tamamlanmadığını kontrol edin |
|       | Şube Hazır Değil İçinde | 201     | Tamamlanana kadar döngü                  |
|       |                         | 5       | Veri baytını yükle                       |

(b) İzole I/O

**Şekil 7.5** Bellek Eşlemeli ve İzole G/C

## 7.4 INTErrUPT-DrIVEN I/O

Programlanmış G/C ile ilgili sorun, işlemcinin ilgili G/C modülünün veri alımı ya da iletimi için hazır olması için uzun süre beklemek zorunda olmasına rağmen. İşlemci beklerken I/O modülünün durumunu tekrar tekrar sorgulamak zorundadır. Sonuç olarak, tüm sistemin performans seviyesi ciddi şekilde düşer.

Bir alternatif de işlemcinin bir module G/C komutu vermesi ve ardından başka yararlı işler yapmaya devam etmesidir. G/C modülü daha sonra işlemciyle veri alışverişi yapmaya hazır olduğunda servis talebinde bulunmak için işlemciyi kesecektir. İşlemci daha sonra, daha önce olduğu gibi veri aktarımını gerçekleştirir ve ardından ön-islemelerine devam eder.

Bunun nasıl çalıştığını önce I/O modülü açısından ele alalım. Giriş için, G/C modülü işlemciden bir OKU komutu alır. G/C modülü daha sonra ilişkili bir çevre biriminden veri okumaya devam eder. Veriler modülün veri kaydına girdiğinde, modül bir kontrol hattı üzerinden işlemciye bir kesme sinyali gönderir. Modül daha sonra verileri işlemci tarafından talep edilene kadar bekler. İstek yapıldığında, modül verilerini veri yoluyla yerleştirir ve ardından başka bir G/C işlemi için hazır olur.

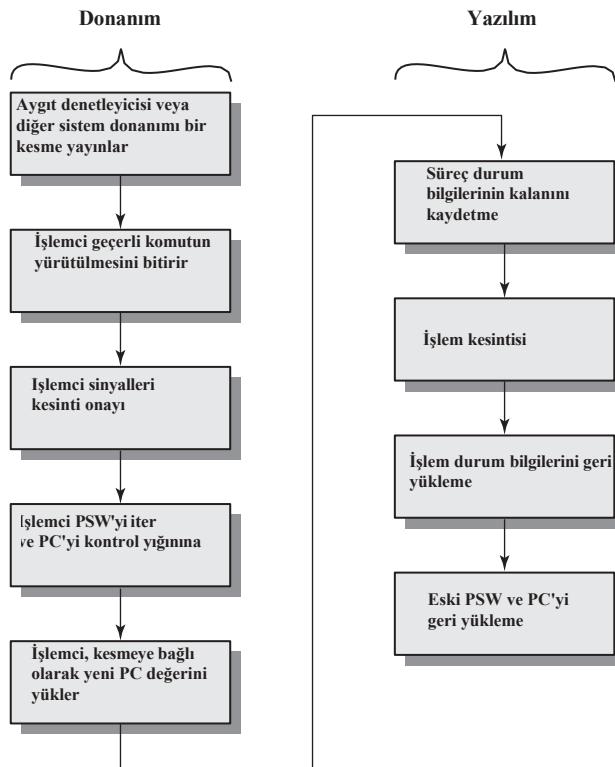
İşlemcinin bakiş açısından, giriş için eylem aşağıdaki gibidir. İşlemci bir OKU komutu verir. Daha sonra gider ve başka bir şey yapar (örneğin işlemci aynı birkaç farklı program üzerinde çalışıyor olabilir). Her komut döngüsünün sonunda, işlemci kesmeleri kontrol eder (Şekil 3.9). I/O modülünden gelen kesme oluştuğunda, işlemci mevcut programın bağlamını (örneğin, program sayıçı ve işlemci kayıtları) kaydeder ve kesmeyi işler. Bu durumda, işlemci I/O modülünden veri sözcüğünü okur ve bellekte saklar. Daha sonra üzerinde çalıştığı programın (ya da başka bir programın) bağlamını geri yükler ve yürütmeye devam eder.

Şekil 7.4b, bir veri bloğunu okumak için kesme G/C kullanımını göstermektedir. Bunu Şekil 7.4a ile karşılaştırır. Kesme G/C, programlanmış G/C'den daha verimlidir çünkü gereksiz beklemeyi ortadan kaldırır. Ancak, bellekten G/C modülüne veya G/C modülünden belleğe giden her veri sözcüğünün işlemciden geçmesi gereğinden, kesme G/C'si hala çok fazla işlemci zamanı tüketir.

### Kesme İşleme

İşlemcinin kesme güdümlü I/O'daki rolünü daha ayrıntılı olarak ele alalım. Bir kesmenin meydana gelmesi hem işlemci donanımında hem de yazılımda bir dizi olayı tetikler. Şekil 7.6 tipik bir sıralamayı göstermektedir. Bir G/C cihazı bir G/C işlemini tamamladığında, aşağıdaki donanım olayları dizisi meydana gelir:

1. Cihaz işlemciye bir kesme sinyali gönderir.
2. İşlemci, Şekil 3.9'da gösterildiği gibi, kesmeye yanıt vermeden önce geçerli komutun yürütülmesini tamamlar.
3. İşlemci bir kesme olup olmadığını test eder, bir olduğunu belirler ve kesmeyi veren cihaza bir onay sinyali gönderir. Onay sinyali cihazın kesme sinyalini kaldırmasını sağlar.



Şekil 7.6 Basit Kesme İşlemleri

4. İşlemcinin şimdi kontrolü kesme hattına aktarmak için hazırlanması gereklidir. Başlamak için, kesme noktasında mevcut programı devam ettirmek için gereken bilgileri kaydetmesi gereklidir. Gerekli minimum bilgi (a) **program durum kelimesi (PSW)** adı verilen bir kayıtta bulunan işlemcinin durumu; ve (b) program sayacında bulunan yürütülecek bir sonraki komutun konumudur. Bunlar sistem kontrol yiğinına itilebilir.<sup>2</sup>
5. İşlemci şimdi program sayacına bu kesmeye yanıt verecek olan kesme işleme programının giriş konumunu yükler. Bilgisayar mimarisine ve işletim sistemi tasarımasına bağlı olarak, tek bir program; her kesme türü için bir program; veya her cihaz ve her kesme türü için bir program olabilir. Birden fazla kesme işleme rutini varsa, işlemci hangisini çağıracağını belirlemelidir. Bu bilgi orijinal kesme sinyaline dahil edilmiş olabilir veya işlemcinin gerekli bilgileri içeren bir yanıt almak için kesmeyi veren aygıtın istek göndermesi gerekebilir.

<sup>2</sup>Baca işletimi ile ilgili bir tartışma için Ek I'e bakınız.

Program sayacı yüklenikten sonra, işlemci bir sonraki komut döngüsüne geçer ve bu döngü komut getirme ile başlar. Komut getirme program sayacının içeriği tarafından belirlendiğinden, sonuç olarak kontrol kesme-işleyici programına aktarılır. Bu programın yürütülmesi aşağıdaki işlemlerle sonuçlanır:

6. Bu noktada, kesilen programla ilgili program sayacı ve PSW sistem yiğinına kaydedilmiştir. Ancak, yürütülmekte olan programın "durumunun" bir parçası olarak kabul edilen başka bilgiler de vardır. Özellikle, işlemci kayıtlarının içeriğinin kaydedilmesi gereklidir, çünkü bu kayıtlar kesme işleyicisi tarafından kullanılabilir. Dolayısıyla, tüm bu değerlerin ve diğer durum bilgilerinin kaydedilmesi gereklidir. Tipik olarak, kesme işleyicisi yiğindaki tüm kayıtların içeriğini kaydederek başlayacaktır. Şekil 7.7a'da basit bir örnek gösterilmektedir. Bu durumda, bir kullanıcı programı  $N$  konumundaki komuttan sonra kesilir. Tüm kayıtların içeriği artı bir sonraki komutun adresi ( $N+1$ ) yiğine itilir. Yiğin işaretçisi yiğinin yeni tepesini gösterecek şekilde güncellenir ve program sayacı kesme hizmet rutininin başlangıcını gösterecek şekilde .
7. Kesme işleyicisi daha sonra kesmeyi işler. Bu, G/C işlemi veya kesmeye neden olan diğer olayla ilgili durum bilgilerinin incelenmesini içerir. Ayrıca G/C cihazına ek komutlar veya onaylar göndermeyi de içerebilir.
8. Kesme işlemi tamamlandıında, kaydedilen kayıt değerleri yiğinden alınır ve kayıtlara geri yüklenir (örneğin, bkz. Şekil 7.7b).
9. Son işlem PSW ve program sayacı değerlerini yiğinden geri yüklemektir. Sonuç olarak, yürütülecek bir sonraki komut daha önce kesilen programdan olacaktır.

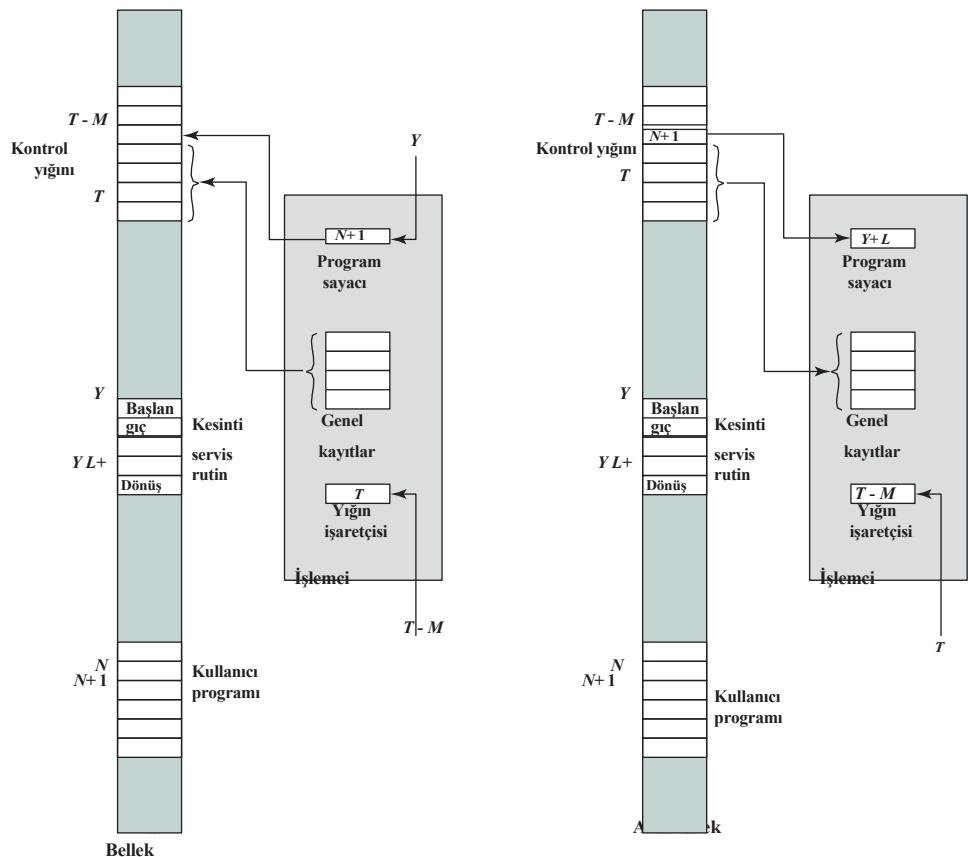
Kesilen programla ilgili tüm durum bilgilerini daha sonra yeniden başlatmak için kaydetmenin önemli olduğunu unutmayın. Bunun nedeni, kesmenin programdan çağrılan bir rutin olmamasıdır. Aksine, kesme herhangi bir zamanda ve dolayısıyla bir kullanıcı programının yürütülmesinin herhangi noktasında meydana gelebilir. Oluşumu tahmin edilemez. Gerçekten de, bir sonraki bölümde görüşeceğimiz gibi, iki programın hiçbir ortak noktası olmayabilir ve iki farklı kullanıcıya ait olabilir.

### Tasarım Sorunları

Kesme I/O uygulamasında iki tasarım sorunu ortaya çıkmaktadır. Birinci, neredeyse değişmez bir şekilde birden fazla I/O modülü olabileceği gibi, işlemci hangi cihazın kesme verdienenğini nasıl belirler? İkinci, birden fazla kesme oluşmuşsa, işlemci hangisini işleyeceğini nasıl karar verir?

Önce cihaz tanımlamayı ele alalım. Dört genel teknik kategorisi yaygın olarak kullanılmaktadır:

- Çoklu kesme hatları
- Yazılım anketi
- Papatya zinciri (donanım yoklaması, vektörlü)
- Veri yolu tahlimi (vektörlü)



Şekil 7.7 Bir Kesme için Bellek ve Kayıtlardaki Değişiklikler

Bu soruna en basit yaklaşım, işlemci ile G/C modülleri arasında **birden fazla kesme hattı** sağlamaktır. Ancak, birkaç veri yolu hattından veya işlemci pininden fazlasını kesme hatlarına ayırmak pratik değildir. Sonuç olarak, birden fazla hat kullanılsa bile, her hattın kendisine bağlı birden fazla I/O modülüne sahip olması muhtemeldir. Bu nedenle, her bir hat üzerinde diğer üç teknikten biri kullanılmalıdır. Bir alternatif de **yazılım yoklamasıdır**. İşlemci bir kesme algıladığından, hangi modülün kesmeye neden olduğunu belirlemek için her bir G/C modülünü sorgulayan bir kesme hizmeti rutinine dallanır. Yoklama ayrı bir komut hattı şeklinde olabilir (örneğin, TESTI/O). Bu durumda, işlemci TESTI/O'yu yükseltir ve belirli bir G/C modülünün adresini adres hattlarına yerlestirir. G/C modülü kesmeyi ayarlarla olumlu yanıt verir. Alternatif olarak, her I/O modülü adreslenebilir bir durum kaydı içerebilir. İşlemci daha sonra kesen modülü tanımlamak için her bir G/C modülünün durum kaydını okur. Doğru modül belirlendikten sonra, işlemci o cihaza özel bir cihaz servis rutinine dallanır.

Yazılım yoklamasının dezavantajı zaman alıcı olmasıdır. Daha verimli bir teknik, aslında bir donanım yoklaması sağlayan bir **papatya zinciri** kullanmaktadır. Papatya zinciri yapılandırmamasının bir örneği Şekil 3.26'da gösterilmektedir. Kesmeler için, tüm I/O modülleri ortak bir kesme istek hattını paylaşır. Kesme onay hattı modüller arasında papatya zinciri şeklinde bağlanır. İşlemci bir kesme algıladığında, bir kesme onayı gönderir. Bu sinyal, talep eden bir module ulaşana kadar bir dizi I/O modülü boyunca yayılır. Talep eden modül tipik olarak veri hatlarına bir kelime yerleştirerek yanıt verir. Bu kelime *vektör* olarak adlandırılır ve I/O modülinin adresi ya da başka bir benzersiz tanımlayıcıdır. Her iki durumda da, işlemci vektörü uygun cihaz servis rutinine bir işaretçi olarak kullanır. Bu, önce genel bir kesme hizmeti rutini çalışma ihtiyacını ortadan kaldırır. Bu tekniğe *vektörlü kesme* adı verilir.

Vektörlü kesmelerden yararlanan başka bir teknik daha vardır ve bu da veri **yolu arbitrasyonudur**. Veri yolu arbitrasyonu ile, bir G/C modülü kesme istek hattını yükseltilmeden önce veri yolunun kontrolünü ele geçirmektedir. Böylece, aynı anda yalnızca bir modül hattı yükseltebilir. İşlemci kesmeyi algıladığında, kesme onay hattından yanıt verir. Talep eden modül daha sonra vektörünü veri hatlarına yerleştirir.

Yukarıda bahsedilen teknikler, talep eden G/C modülini tanımlamaya yarar. Ayrıca birden fazla cihaz kesme hizmeti talep ettiğinde öncelikleri atamak için bir yol sağlarlar. Birden fazla hat olduğunda, işlemci sadece en yüksek önceliğe sahip kesme hattını seçer. Yazılım yoklaması ile, modüllerin yoklanma sırasında önceliklerini belirler. Benzer şekilde, bir papatya zincirindeki modüllerin sırası da belirler. Son olarak, veri yolu tâkîmî Bölüm 3.4'te tartışıldığı gibi bir öncelik şeması kullanılabilir.

Şimdi iki kesme yapısı örneğine dönüyoruz.

### Intel 82C59A Kesme Denetleyicisi

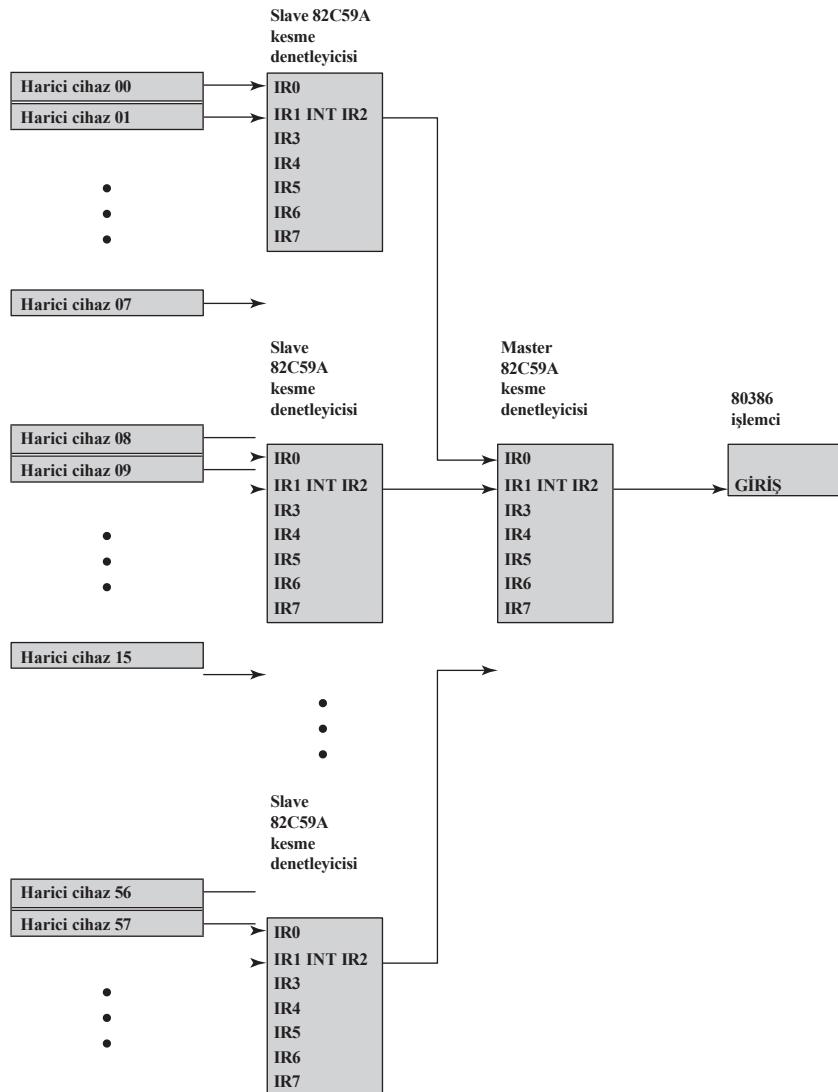
Intel 80386 tek bir Kesme İsteği (INTR) ve tek bir Onay (INTA) hattı sağlar. 80386'nın çeşitli aygıtları ve öncelik yapılarını idare etmesine izin vermek için, genellikle harici bir kesme hakemi olan 82C59A ile yapılandırılır. Harici cihazlar 82C59A'ya bağlanır, o da 80386'ya bağlanır.

Şekil 7.8, 80386 için birden fazla I/O modülini bağlamak üzere 82C59A'nın kullanımını göstermektedir. Tek bir 82C59A sekiz adede kadar modülü idare edebilir. Sekizden fazla modül için kontrol gerekiyorsa, 64 modüle işlemek için kademeli bir düzenleme kullanılabilir.

82C59A'nın tek sorumluluğu kesmelerin yönetimidir. Bağlı modüllerden gelen kesme isteklerini kabul eder, hangi kesmenin en yüksek önceliğe sahip olduğunu belirler ve ardından INTR hattını yükselterek işlemciye sinyal gönderir. İşlemci INTA hattı aracılığıyla onaylar. Bu, 82C59A'nın uygun vektör bilgisini veri yoluna yerleştirmesini ister. İşlemci daha sonra kesmeyi işlemeye ve veri okumak veya yazmak için I/O modülüyle doğrudan iletişim kurmaya devam edebilir.

82C59A programlanabilir. 80386, 82C59A'da bir kontrol sözcüğü ayırararak kullanılacak öncelik şemasını belirler. Aşağıdaki kesme modları mûmkündür:

- **Tamamen iç içe:** Kesme istekleri öncelik sırasına göre 0'dan (IR0) 7'ye (IR7) kadar sıralanır.



#### **Şekil 7.8 Harici 82C59A Kesme Denetleyicisinin Kullanımı**

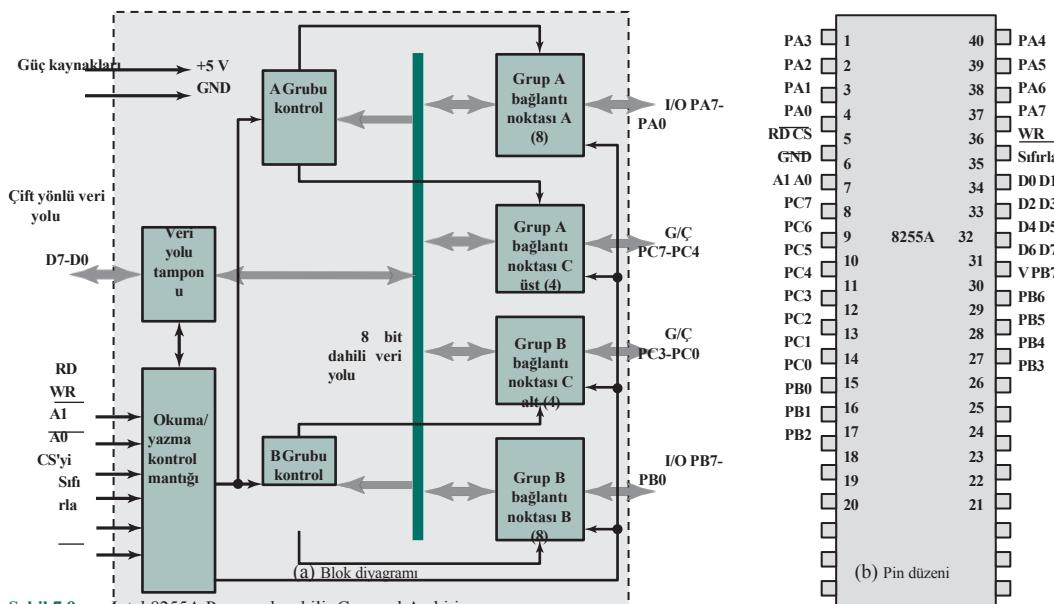
- **Dönen:** Bazı uygulamalarda bir dizi kesinti yapan cihaz eGit önceliğe sahiptir. Bu modda bir cihaz, servis verildikten sonra gruptaki en düşük önceliği alır.
- **Özel maske:** Bu, işlemcinin belirli aygitlardan gelen kesmeleri engellemesini sağlar.

## Intel 8255A Programlanabilir Çevresel Arabirim

Programlanmış G/C ve kesme güdümlü G/C için kullanılan bir G/C modülü örneği olarak Intel 8255A Programlanabilir Çevresel Arabirimi ele alıyoruz. 8255A, orijinal olarak Intel 80386 işlemci ile kullanılmak üzere tasarlanmış tek çipli, genel amaçlı bir I/O modülür. O zamandan beri diğer üreticiler tarafından klonlanmıştır ve yaygın olarak kullanılan bir çevresel denetleyici yongasıdır. Kullanımları arasında mikroişlemciler için basit G / C aygıtları için bir denetleyici olarak ve mikrodenetleyici sistemleri de dahil olmak üzere gömülü sistemlerde bulunur.

**YAPI VE ÇALIŞMA** Şekil 7.9'da genel bir blok şeması ve içinde bulunduğu 40 pinli paket için pin ataması gösterilmektedir. Pin düzende gösterildiği gibi, 8255A aşağıdaki hatları içerir:

- **D0-D7:** Bunlar cihaz için veri I/O hatlarıdır. 8255A'dan okunan ve 8255A'ya yazılan tüm bilgiler bu sekiz veri hattı üzerinden gerçekleşir.
- **CS (Çip Seçme Girişisi):** Bu hat mantıksal 0 ise, mikroişlemci 8255A'yı okuyabilir ve yazabilir.
- **RD (Okuma Girişisi):** Bu hat mantıksal 0 ise ve CS girişi mantıksal 0 ise, 8255A veri çıkışları sistem veri yoluna etkinleştirilir.
- **WR (Yazma Girişisi):** Bu giriş hattı mantıksal 0 ise ve CS girişi de mantıksal 0 ise, sistem veri yolundan 8255A'ya veri yazılır.
- **RESET:** Bu giriş hattı mantıksal 1 ise 8255A sıfırlama durumuna getirilir. Tüm çevresel portlar giriş moduna ayarlanır.



Şekil 7.9 Intel 8255A Programlanabilir Çevresel Arabirim

- **PA0-PA7, PB0-PB7, PC0-PC7:** Bu sinyal hatları 8 bit I/O portları olarak kullanılır. Çevresel cihazlara bağlanabilirler.
- **A0, A1:** Bu iki giriş hattının mantıksal kombinasyonu, 8255A verilerinin hangi dahili kayda yazılacağını veya okunacağını belirler.

Şekil 7.9'a daki blok diyagramın sağ tarafı 8255A'nın harici arayüzüdür. 24 G/C hattı üç adet 8 bitlik gruba (A, B, C) ayrılmıştır. Her grup 8-bit G/C portu olarak işlev görebilir, böylece üç çevresel cihaz için bağlantı sağlar. Ayrıca C grubu, A ve B G/C portları ile birlikte kullanılabilen 4 bitlik gruplara ( $C_A$  ve  $C_B$ ) bölünmüştür. Bu şekilde yapılandırılan C grubu hatları kontrol ve durum sinyallerini taşıır.

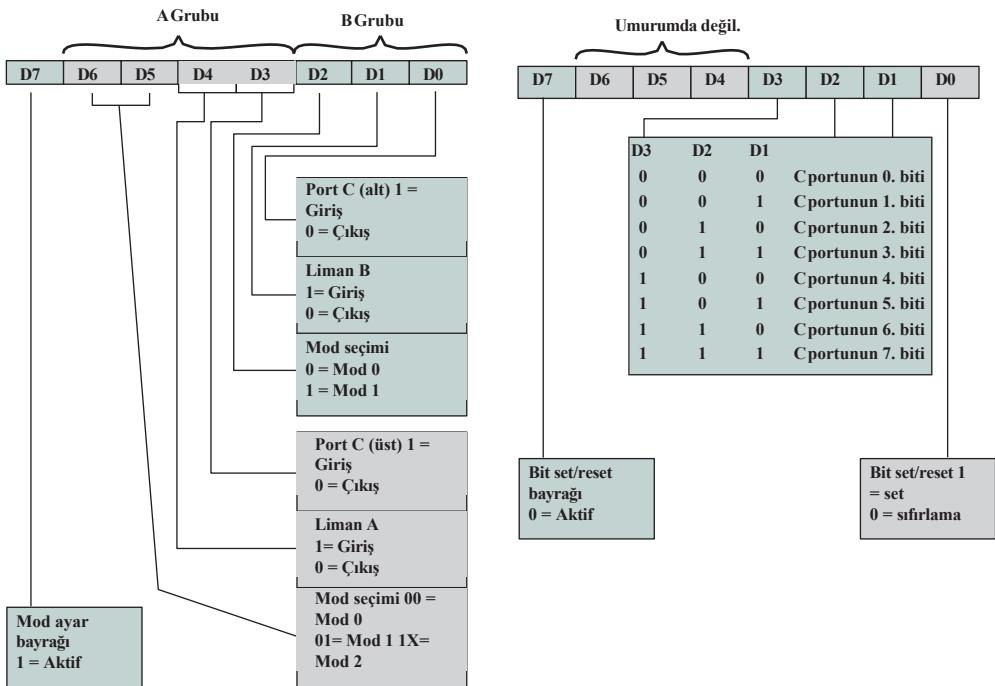
Blok diyagramın sol tarafı mikroişlemci sistem dahili arayüzüdür. Mikroişlemci ile G/C portları arasında veri aktarımı ve kontrol bilgilerini aktarmak için kullanılan 8 bitlik çift yönlü bir veri yolu (D0 ila D7) içerir.

İşlemci, 8255A'yı işlemcideki 8 bitlik bir kontrol kaydı aracılığıyla kontrol eder. İşlemci, çeşitli çalışma modlarını ve yapılandırmaları belirtmek için kontrol kaydının değerini ayarlayabilir. İşlemci açısından, bir kontrol portu vardır ve kontrol kayıt bitleri işlemcide ayarlanır ve ardından D0-D7 hatları üzerinden kontrol portuna gönderilir. İki adres hattı, aşağıdaki gibi üç G/C portundan birini veya kontrol kaydını belirtir:

| A1 | A2 | Seçer         |
|----|----|---------------|
| 0  | 0  | Liman A       |
| 0  | 1  | Liman B       |
| 1  | 0  | C Limanı      |
| 1  | 1  | Kontrol kaydı |

Böylece, işlemci hem A1 hem de A2'1'e ayarladığında, 8255A veri yolundaki 8 bitlik değeri bir kontrol kelimesi olarak yorumlar. İşlemci D7 hattı 1'e ayarlıken 8 bitlik bir kontrol kelimesi aktardığında (Şekil 7.10a), kontrol kelimesi 24 I/O hattının çalışma modunu yapılandırmak için kullanılır. Üç mod sunlardır:

- **Mod 0:** Bu temel G/C modudur. Sekiz harici hattan oluşan üç grup, üç adet 8 bit G/C portu olarak işlev görür. Her port giriş veya çıkış olarak belirlenebilir. Veri yalnızca port çıkış olarak tanımlanmışsa bir porta gönderilebilir ve veri yalnızca port giriş olarak ayarlanmışsa bir porttan okunabilir.
- **Mod 1:** Bu modda, A ve B portları giriş veya çıkış olarak yapılandırılabilir ve C portundan gelen hatlar A ve B için kontrol hatları olarak hizmet eder: "el sıkışma" ve kesme isteği. El sıkışma basit bir zamanlama mekanizmasıdır. Bir kontrol hattı, verilerin G/C veri hatlarında mevcut olduğunu belirtmek için gönderici tarafından VERİ HAZIR hattı olarak kullanılır. Diğer bir hat alıcı tarafından, verilerin okunduğunu ve veri hatlarının temizlenebileceğini belirten bir ONAY hattı olarak kullanılır. Başka bir hat KESME İSTEK hattı olarak belirlenebilir ve sistem veriyoluna geri bağlanabilir.



(a) 8255'i yapılandırmak için 8255 kontrol kaydının mod tanımı

(b) Port C'nin tek bitlerini değiştirmek için 8255 kontrol kaydının bit tanımları

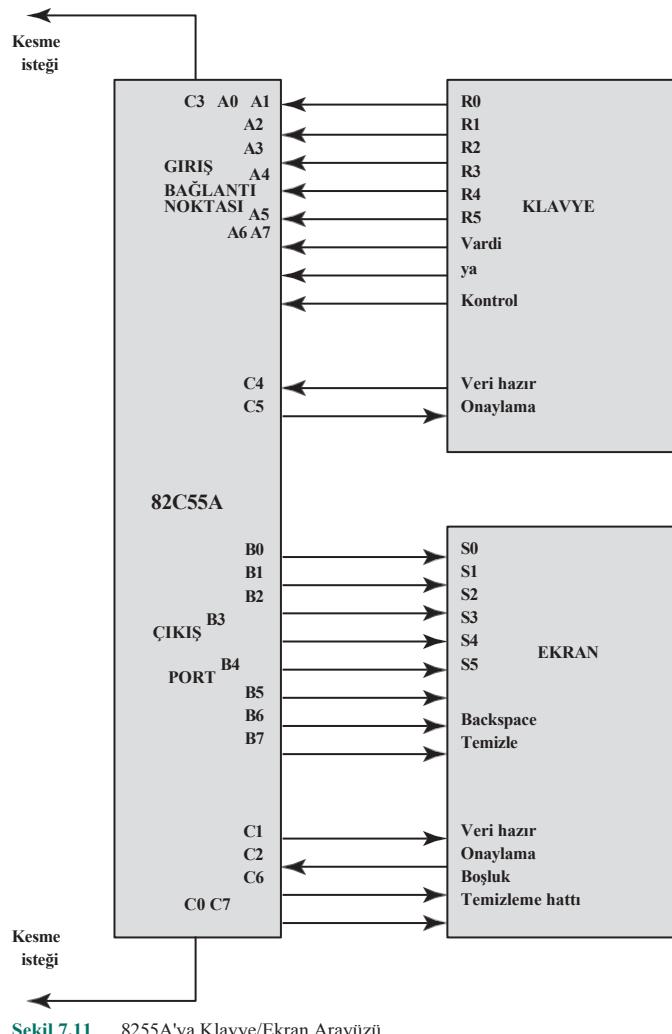
Şekil 7.10 Intel 8255A Kontrol Sözcüğü

**■ Mod 2:** Bu çift yönlü bir moddur. Bu modda, A portu B portundaki çift yönlü trafik için giriş veya çıkış hatları olarak yapılandırılabilir ve B portu hatları ters yönü sağlar. Yine, port C hatları kontrol sinyali için kullanılır.

İşlemci D7'yi 0'a ayarladığında (Şekil 7.10b), kontrol kelimesi C portunun bit değerlerini ayrı ayrı programlamak için kullanılır. Bu özellik nadiren kullanılır.

**KLAVYE/EKRAN** ÖRNEĞİ 8255A kontrol yazmacı aracılığıyla programlanabilir olduğundan, çeşitli basit çevreSEL aygıtları kontrol etmek için kullanılabilir. Şekil 7.11'de bir klavye/ekran terminalini kontrol etmek için kullanımı gösterilmektedir. Klavye 8 bit giriş sağlar. Bu bitlerden ikisi, SHIFT ve CONTROL, işlemcide çalışan klavye işleme programı için özel bir anlama sahiptir. Ancak, bu yorumlama 8255A için şeffaftır ve 8 bit veriyi kabul eder ve sistem veri yolunda sunar. Klavye ile kullanım için iki el sıkışma kontrol hattı sağlanmıştır.

Ekran ayrıca 8 bitlik bir veri portu ile bağlantılıdır. Yine, bitlerden ikisinin 8255A için şeffaf olan özel anlamları vardır. İki el sıkışma hattına ek olarak, iki hat ek kontrol işlevleri sağlar.



## 7.5 DİREKT MEMOrY ERİŞİMİ

### Programlanmış ve Kesme Tahraklı G/C'nin Dezavantajları

Kesme güdümlü G/C, basit programlanmış G/C'den daha verimli olsa da, bellek ve G/C modülü arasında veri aktarımı için hala işlemcinin aktif müdahalesini gerektirir ve herhangi bir veri aktarımı işlemci boyunca bir yoldan geçmelidir. Dolayısıyla, bu G/C biçimlerinin her ikisi de iki doğal dezavantajdan muzdariptir:

1. G/C aktarım hızı, işlemcinin bir cihazı test edebileceği ve servis verebileceği hız ile sınırlıdır.

- 2.** İşlemci bir G/C aktarımını yönetmekte yükümlüdür; her G/C aktarımı için bir dizi talimatın yürütülmesi gereklidir (örneğin, Şekil 7.5).

Bu iki dezavantaj arasında bir miktar denge vardır. Bir veri bloğunun transferini düşünün. Basit programlanmış G/C kullanıldığında, işlemci G/C görevine adanır ve başka hiçbir şey yapmama pahasına verileri oldukça yüksek bir hızda taşıyabilir. Kesintili G/C, G/C aktarım hızı pahasına işlemciyi bir dereceye kadar serbest bırakır. Bununla birlikte, her iki yöntem de hem işlemci etkinliği hem de G/C aktarım hızı üzerinde olumsuz bir etkiye sahiptir.

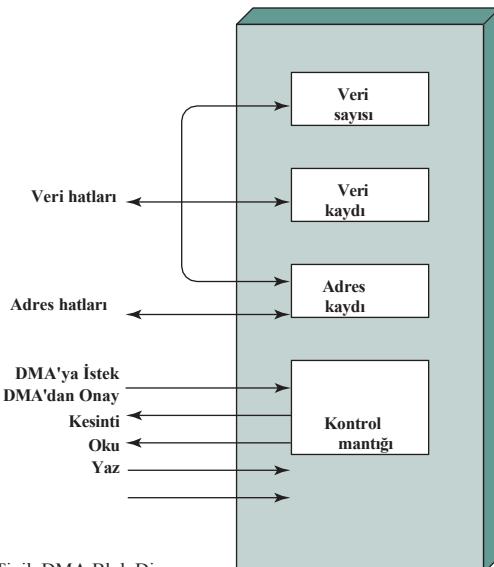
Büyük hacimli verilerin taşınması gerektiğinde, daha verimli bir teknik gereklidir: doğrudan bellek erişimi (DMA).

### DMA İşlevi

DMA sistem veriyolu üzerinde ek bir modül içerir. DMA modülü (Şekil 7.12) işlemciyi taklit edebilir ve aslında sistemin kontrolünü işlemciden devralabilir. Sistem veriyolu üzerinden belleğe ve bellekten veri aktarmak için bunu yapması gereklidir. Bu amaçla, DMA modülü yalnızca işlemcinin ihtiyaç duymadığı zamanlarda kullanmalı ya da işlemciyi geçici olarak çalışmayı aşağıya zorlamalıdır. İlkinci teknik daha yaygındır ve *döngü çalışma* olarak adlandırılır, çünkü DMA modülü aslında bir veri yolu döngüsü çalar.

İşlemci bir veri bloğunu okumak ya da yazmak istediğiinde, DMA modülüne aşağıdaki bilgileri göndererek bir komut verir:

- İşlemci ve DMA modülü arasındaki okuma veya yazma kontrol hattı kullanılarak bir okuma veya yazma talep edilip edilmediği.
- İlgili G/C cihazının veri hatlarında iletilen adresi.



Şekil 7.12 Tipik DMA Blok Diyagramı

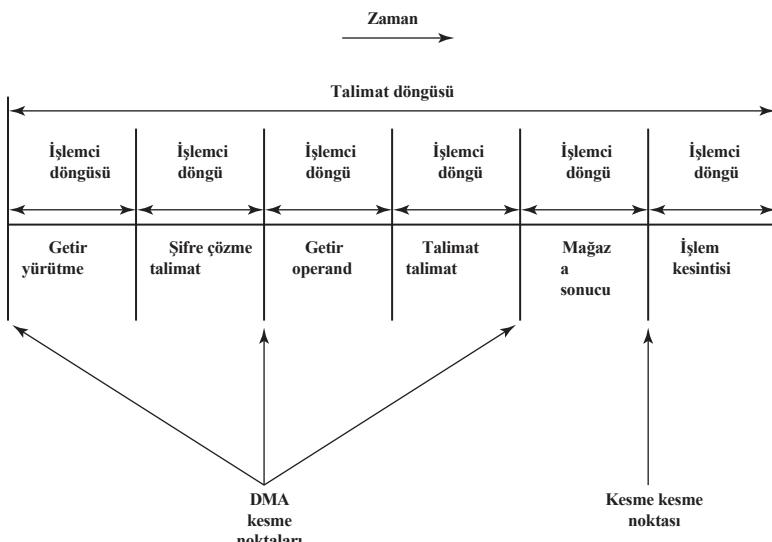
- Veri hatlarında iletilen ve DMA modülü tarafından adres kaydında saklanan, okunacak veya yazılacak bellekteki başlangıç konumu.
- Okunacak veya yazılacak kelime sayısı, yine veri hatları üzerinden iletilir ve veri sayım kaydında saklanır.

İşlemci daha sonra diğer işlere devam eder. Bu I/O işlemini DMA modülüne devretmiştir. DMA modülü tüm veri bloğunu, her bir kelime olmak üzere, geçmeden doğrudan belleğe veya bellekten aktarır. Aktarım tamamlandığında, DMA modülü işlemciye bir kesme sinyali gönderir. Böylece işlemci yalnızca aktarımın başında ve sonunda devreye girer (Şekil 7.4c).

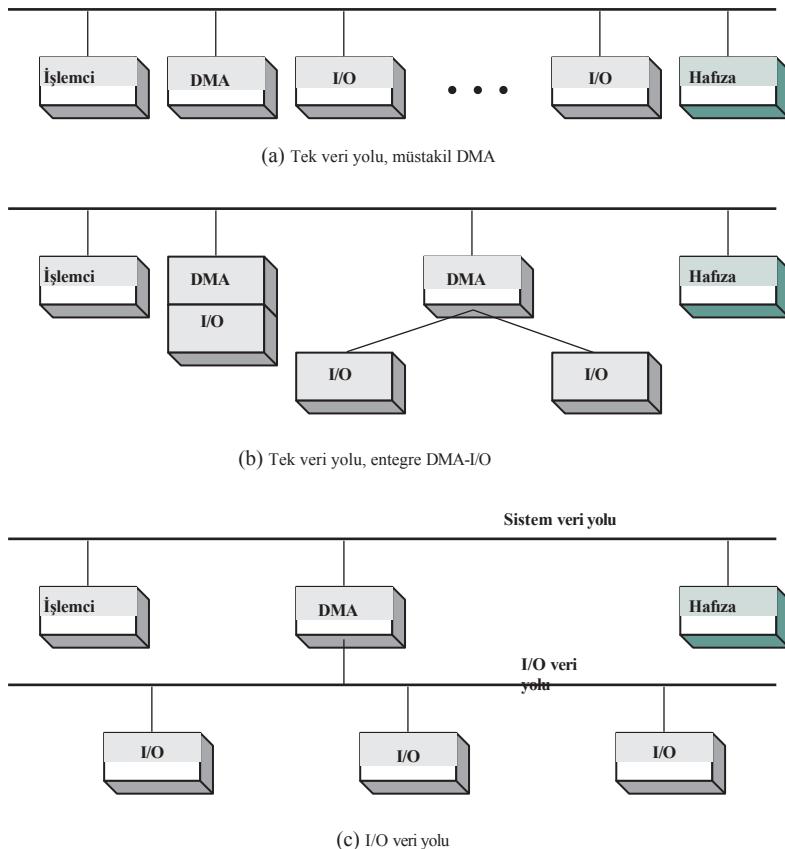
Şekil 7.13, komut çevriminde işlemcinin nerede askiya alınabileceğini göstermektedir. Her durumda, işlemci veri yolunu kullanması gerekmenden hemen önce askiya alınır. DMA modülü daha sonra bir kelime aktarır ve kontrolü işlemciye geri verir. Bunun bir kesme olmadığına dikkat edin; işlemci bir bağlamı kaydetmez ve başka bir şey yapmaz. Aksine, işlemci bir veri yolu döngüsü için duraklar. Bunun genel etkisi işlemcinin daha yavaş çalışmasına neden olmaktadır. Bununla birlikte, çok kelimeli bir G/C aktarımı için DMA, kesme güdümlü veya programlanmış G/C'den çok daha verimlidir.

DMA mekanizması çeşitli şekillerde yapılandırılabilir. Bazı olasılıklar Şekil 7.14'te gösterilmektedir. İlkörnekte, tüm modüller aynı sistem veriyolunu paylaşmaktadır. Vekil işlemci olarak hareket eden DMA modülü, DMA modülü aracılığıyla bellek ve bir I/O modülü arasında veri alışverişini yapmak için programlanmış I/O kullanır. Bu yapılandırma ucuz olsa da açıkça verimsizdir. İşlemci kontrollü programlanmış G/C'de olduğu gibi, bir kelimenin her aktarımı iki veri yolu döngüsü tüketir.

DMA ve I/O işlevleri entegre edilerek gerekli yolu döngülerinin sayısı önemli ölçüde azaltılabilir. Şekil 7.14b'de gösterildiği gibi, bu, DMA modülü ile bir veya daha fazla G/C modülü arasında aşağıdakileri içermeyen bir yol olduğu anlamına gelir



**Şekil 7.13** Bir Komut Döngüsü Sırasında DMA ve Kesme Noktaları

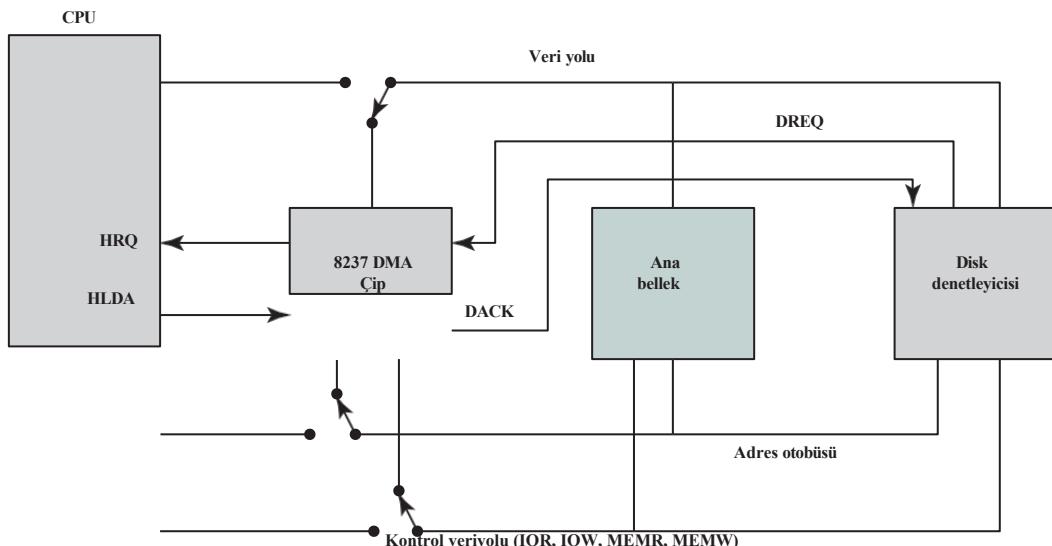


Şekil 7.14 Alternatif DMA Yapılandırmaları

sistem veriyolu. DMA mantığı aslında bir I/O modülünün parçası olabilir ya da bir veya daha fazla I/O kontrol eden ayrı bir modül olabilir. Bu konsept, I/O modüllerini bir I/O veri yolu kullanarak DMA modülüne bağlayarak bir adım daha ileri götürülebilir (Şekil 7.14c). Bu, DMA modültündeki G/C arayüzlerinin sayısını bire indirir ve kolayca genişletilebilir bir yapılandırma sağlar. Her iki durumda da (Şekil 7.14b ve c), DMA modülünün işlemci ve bellekle paylaştığı sistem veri yolu, DMA modülü tarafından yalnızca bellekle veri alışverişi yapmak için kullanılır. DMA ve I/O modülleri arasındaki veri alışverişi sistem veriyolunun dışında gerçekleşir.

### Intel 8237A DMA Denetleyicisi

Intel 8237A DMA denetleyicisi, bir DMA özelliği sağlamak için 80 \* 86 işlemci ailesine ve DRAM belleğe arabirim oluşturur. Şekil 7.15 DMA modülünün göstermektedir. DMA modülünün veri aktarmak için sistem veri yollarını (veri, adres ve kontrol) kullanması gerektiğinde, işlemciye HOLD adı verilen bir sinyal gönderir. İşlemci HLDA (tutma onayı) sinyali ile yanıt vererek şunu belirtir



**Sekil 7.15** 8237 Sistem Veriyolunun DMA Kullanımı

DMA modülü veri yollarını kullanabilir. Örneğin, DMA modülü bir veri bloğunu bellekten diske aktaracaksa, aşağıdakileri yapacaktır:

1. Çevresel aygit (disk denetleyicisi gibi) DREQ (DMA isteği) değerini yükseğe çekerek DMA hizmeti talep edecektir.
2. DMA, HRQ (tutma isteği) değerine yüksek bir değer vererek HOLD pini aracılığıyla CPU'ya veri yollarını kullanması gerektiğini bildirir.
3. CPU mevcut veri yolu döngüsünü (mevcut talimat olmasa gerekmez) bitirecek ve DMA isteğine HDLA'sına (tutma onayı) yüksek değer vererek yanıt verecek, böylece 8237 DMA'ya devam edebileceğini ve görevini yerine getirmek için veri yollarını kullanabileceğini söyleyecektir. DMA görevini yerine getirdiği sürece HOLD aktif yüksek olarak kalmalıdır.
4. DMA, çevresel aygıta veri aktarmaya başlayacağını söyleyen DACK'i (DMA onayı) etkinleştiricektir.
5. DMA, bloğun ilk baytinın adresini adres yoluna koyarak ve MEMR'yi etkinleştirerek verileri bellekten çevre birimine aktarmaya başlar, böylece baytı bellekten veri yoluna okur; daha sonra çevre birimine yazmak için IOW'yi etkinleştirir. Daha sonra DMA sayacı azaltır ve adres işaretçisini artırır ve sayı sıfıra ulaşana ve görev bitene kadar bu işlemi tekrarlar.
6. DMA işini bitirdikten sonra HRQ'yu devre dışı bırakarak CPU'ya veri yolları üzerindeki kontrolünü yeniden kazanabileceği sinyalini verir.

DMA veri aktarmak için veri yollarını kullanırken, işlemci boşadır. Aynı şekilde, işlemci veri yolunu kullanırken DMA boşadır. 8237 DMA bir *fly-by* DMA denetleyicisi olarak bilinir. Bu, bir konumdan diğerine taşınan verilerin DMA yongasından geçmediği ve DMA yongasında saklanmadığı anlamına gelir. Bu nedenle, DMA yalnızca bir G/C portu ile bir bellek adresi arasında veri aktarabilir, iki G/C portu veya iki bellek konumu arasında veri aktaramaz. Bununla birlikte, daha sonra açıklanacağı üzere, DMA yongası bir yazmaç aracılığıyla bellekten belleğe aktarım gerçekleştirebilir.

8237 bağımsız olarak programlanabilen dört DMA kanalı içerir ve kanallardan herhangi biri herhangi bir anda aktif olabilir. Bu kanallar 0, 1, 2 ve 3 olarak numaralandırılmıştır.

8237, kanallarından biri üzerinden DMA işlemini programlamak ve kontrol etmek için beş kontrol/komut kaydedicisinden oluşan bir sete sahiptir (Tablo 7.2):

- **Komut:** İşlemci, DMA'nın çalışmasını kontrol etmek için bu kaydı yükler. D0, kanal 0'in bir baytı 8237 geçici kaydına aktarmak için kullanıldığı ve kanal 1'in baytı kayıttan belleğe aktarmak için kullanıldığı bir bellekten belleğe aktarımı etkinleştirir. Bellekten belleğe etkin olduğunda, D1 kanal 0'daki artırma/azaltmayı devre dışı bırakmak için kullanılabilir, böylece bir bellek bloğuna sabit bir değer yazılabilir. D2, DMA'yı etkinleştirir veya devre dışı bırakır.
- **Durum:** İşlemci DMA durumunu belirlemek için bu kaydı okur. D0-D3 bitleri, 0-3 kanallarının TC'lerine (terminal sayısı) ulaşmışlığını belirtmek için kullanılır. D4-D7 bitleri işlemci tarafından herhangi bir kanalda bekleyen bir DMA isteği olmadığını belirlemek için kullanılır.
- **Mod:** İşlemci, DMA'nın çalışma modunu belirlemek için bu kaydı ayarlar. D0 ve D1 bitleri bir kanal seçmek için kullanılır. Diğer bitler seçilen kanal için çeşitli çalışma modlarını seçer. D2 ve D3 bitleri, aktarımın bir G/C aygitinden belleğe mi (yazma) yoksa bellekten G/C'ye mi (okuma) ya da bir doğrulama işlemi mi olduğunu belirler. D4 ayarlanması, bellek adres kaydı ve sayımları bir DMA veri aktarımının sonunda orijinal değerleriyle yeniden yüklenir. D6 ve D7 bitleri 8237'nin kullanım şeklini belirler. Tek modda, tek bir bayt veri aktarılır. Blok ve talep modları bir blok aktarımı için kullanılır, talep modu aktarımın erken sonlandırmasına izin verir. Kademeli mod, kanal sayısını 4'ten fazlasına genişletmek için birden fazla 8237'nin kademelendirilmesine olanak tanır.
- **Tek Maske:** İşlemci bu kaydı ayarlar. D0 ve D1 bitleri kanalı seçer. D2 biti o kanal için maske bitini temizler veya ayarlar. Bu kayıt aracılığıyla belirli bir kanalın DREQ girişi maskelenebilir (devre dışı bırakılabilir) veya maskesi kaldırılabilir (etkinleştirilebilir). Komut kaydı tüm DMA çipini devre dışı bırakmak için kullanılırken, tek maske kaydı programcının belirli bir kanalı devre dışı bırakmasına veya etkinleştirmesine olanak tanır.
- **Tümü Maskesi:** Bu kayıt, tek bir yazma işlemiyle dört kanalın tümünün maskelenebilmesi veya maskesinin kaldırılabilmesi dışında tekli maske kaydına benzer.

Buna ek olarak, 8237A'nın sekiz veri kaydı vardır: her kanal için bir bellek adresi ve bir sayımları kaydi. İşlemci bu kayıtları, transferlerden etkilenecek ana bellek boyutunun yerini belirtmek için ayarlar.

## 254 BÖLÜM 7 / GİRİŞ/ÇIKIŞ

**Tablo 7.2** Intel 8237A Kayıtları

| Bit | Komuta                           | Durum                | Mod                              | Tekli Maske                 | Tüm Maskeler                        |
|-----|----------------------------------|----------------------|----------------------------------|-----------------------------|-------------------------------------|
| D0  | Bellekten belleğe E/D            | Kanal 0 TC'ye ulaştı | Kanal seçimi                     | Kanal maskesi bitini seçin  | Kanal 0 maske bitini temizle/ayarla |
| D1  | Kanal 0 adres tutma E/D          | Kanal 1 TC'ye ulaştı |                                  |                             | Kanal 1 maske bitini temizle/ayarla |
| D2  | Kontrolör E/D                    | Kanal 2 TC'ye ulaştı | Aktarımı doğrulama/yazma(okuma)  | Maske bitini temizle/ayarla | Kanal 2 maske bitini temizle/ayarla |
| D3  | Normal/basılı zamanlama          | Kanal 3 TC'ye ulaştı |                                  | Kullanılmadı                | Kanal 3 maske bitini temizle/ayarla |
| D4  | Sabit/dönen öncelik              | Kanal 0 isteği       | Otomatik başlatma E/D            |                             | Kullanılmadı                        |
| D5  | Geç/uzatılmış yazma seçimi       | Kanal 0 isteği       | Adres artırma/azaltma seçimi     |                             |                                     |
| D6  | DREQ algılama aktif yüksek/düşük | Kanal 0 isteği       |                                  |                             |                                     |
| D7  | DACK algılama aktif yüksek/düşük | Kanal 0 isteği       | Talep/tek/blok/kaskad mod seçimi |                             |                                     |

E/D= etkinleştirme/devre dışı  
birakma TC= terminal sayısı

## 7.6 DİREKT CACHÉ ERİŞİMİ

DMA, çevresel cihazlar ve ağ I/O trafiği ile I/O performansını artırmanın etkili bir yolu olduğunu kanıtlamıştır. Ancak, ağ I/O veri hızlarındaki dramatik artışlar için DMA artan talebi karşılayacak şekilde ölçeklenmemektedir. Bu talep, öncelikle veritabanı sunucularına ve diğer yüksek performanslı sistemlere büyük miktarda veri aktarımını işlemek için 10 Gbps ve 100 Gbps Ethernet anahtarlarının yaygın olarak kullanılmamasından kaynaklanmaktadır [STAL14a]. İkincil ancak giderek daha önemli hale gelen bir trafik kaynağı da gigabit aralığındaki Wi-Fi'den gelmektedir. Ağ Wi-Fi cihazları 3.2 Gbps ve 6.76 Gbps hızlarında çalışabilmekte ve kurumsal sistemlerde talep yaratmaktadır [STAL14b].

Bu bölümde, **doğrudan önbellek erişimi (DCA)** olarak bilinen bir teknikle G/C işlevinin önbelleğe **doğrudan erişiminin** sağlanmasıın performansı nasıl artırabileceğini göstereceğiz. Bu bölüm boyunca, yalnızca ana belleğe en yakın olan ve **son seviye** önbellek olarak adlandırılan **önbellekle** ilgileneceğiz. Bazı sistemlerde bu bir L2 önbellek, bazlarında ise bir L3 önbellek olacaktır.

Başlangıç olarak, çağdaş çok çekirdekli sistemlerin DMA performansını artırmak için çip üzerinde paylaşılan önbelleği nasıl kullandığını açıklayacağız. Bu yaklaşım, DMA işlevinin son seviye önbelleğe doğrudan erişmesini sağlamayı içerir. Daha sonra, yüksek hızlı ağ trafiği işlendiğinde ortaya önbellekle ilgili performans sorunlarını inceliyoruz. Buradan, ağ protokoltü işleme performansını artırmak için tasarlanmış birkaç farklı DCA stratejisine bakıyoruz. Son olarak, bu bölümde Intel tarafından uygulanan ve Doğrudan Veri I/O olarak adlandırılan bir DCA yaklaşımı açıklanmaktadır.

## Paylaşılan Son Seviye Önbellek Kullanan DMA

Bölüm 1'de tartışıldığı gibi (bkz. Şekil 1.2), çağdaş çok çekirdekli sistemler hem her çekirdeğe ayrılmış önbellek hem de L2 veya L3 olmak üzere ek bir paylaşılan seviyesi içerir. Mevcut son seviye önbelleğin artan boyutıyla birlikte, sistem tasarımcıları DMA işlevini geliştirerek DMA denetleyicisinin çekirdeklere benzer bir şekilde paylaşılan önbelleğe erişmesini sağlamıştır. DMA ve önbellek etkileşimi açıkça kavuşturmak için öncelikle belirli bir sistem mimarisini tanımlamak faydalı olacaktır. Bu amaçla, aşağıda Intel Xeon sistemine genel bir bakış sunulmaktadır.

**XEON ÇOK ÇEKİRDEKLİ İŞLEMÇİ** Intel Xeon, Inin sunucularda, yüksek performanslı iş istasyonlarında ve süper bilgisayarlarla kullanılan üst düzey, yüksek performanslı işlemci ailesidir. Xeon ailesinin birçok üyesi, Şekil 7.16'da Xeon E5-2600/4600 için gösterildiği gibi bir halka ara bağlantı sistemi kullanır.

E5-2600/4600 tek bir yonga üzerinde sekiz adede kadar çekirdekle yapılandırılabilir. Her çekirdeğin özel L1 ve L2 önbellekleri vardır. 20 MB'a kadar paylaşılan bir L3 önbelleği vardır. L3 önbellek, her çekirdek tüm önbelleği adresleyebilmesine rağmen, her çekirdekle ilişkili bir tane olmak üzere dilimlere ayrılmıştır. Ayrıca, her dilimin kendi önbellek boru hattı vardır, böylece istekler dilimlere paralel olarak gönderilebilir.

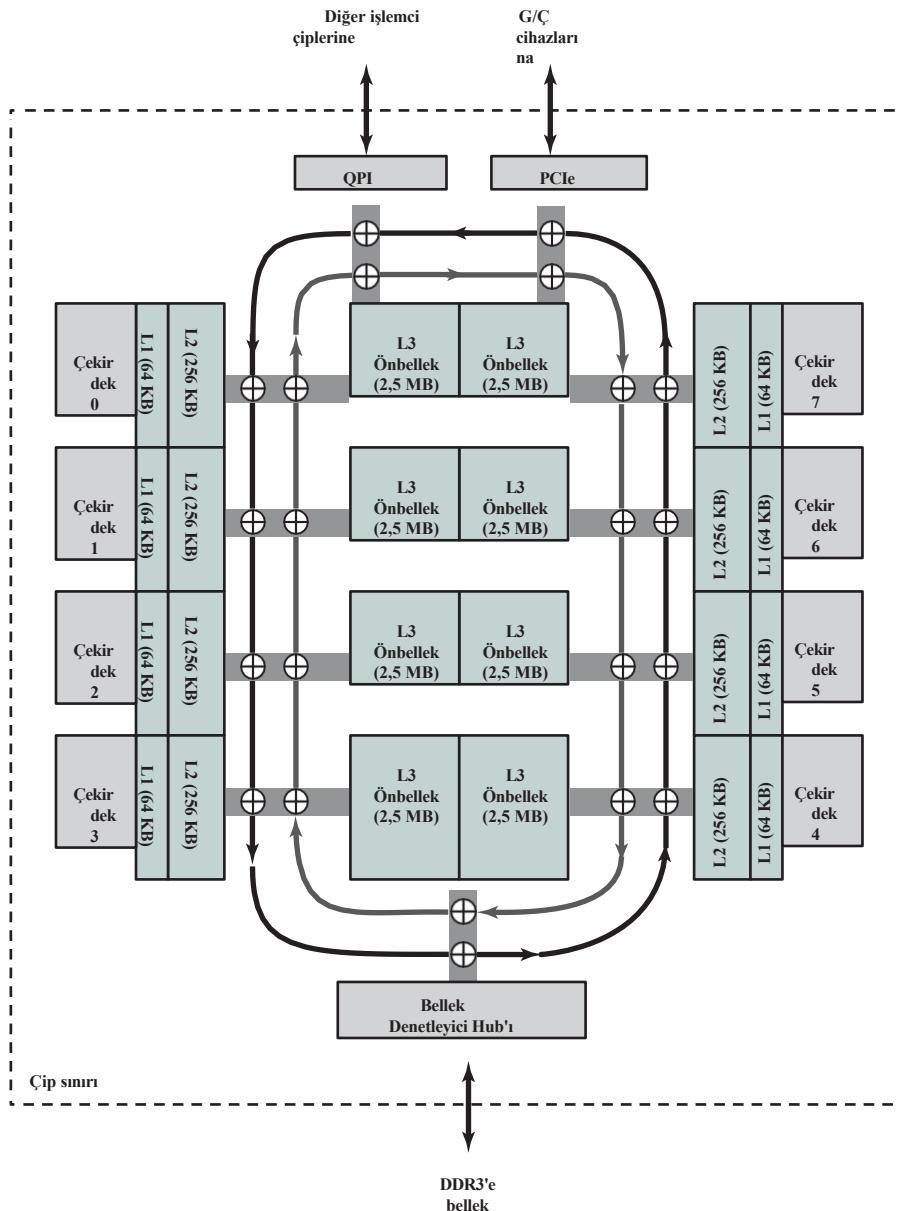
Çift yönlü yüksek hızlı halka ara bağlantısı çekirdekleri, son seviye önbelleği, PCIe'yi ve entegre bellek denetleyicisini (IMC) birbirine bağlar.

Özünde, halka aşağıdaki şekilde çalışır:

1. Çift yönlü halkaya bağlanan her bileşen (QPI, PCIe, L3 önbellek, L2 önbellek) bir halka aracı olarak kabul edilir ve halka aracı mantığını uygular.
2. Halka araçları, zaman dilimleri şeklinde halkaya erişim talep etmek ve tahsis etmek için dağıtılmış bir protokol aracılığıyla işbirliği yapar.
3. Bir temsilcinin göndereceği veri olduğunda, hedefe giden en kısa yolla sonuçlanan halka yönünü seçer ve bir zamanlama yuvası mevcut olduğunda ileter.

Halka mimarisi iyi performans sağlar ve bir noktaya kadar birden fazla çekirdek için iyi ölçeklenir. Daha fazla sayıda çekirdege sahip sistemler için, her bir halkanın bazı çekirdekleri desteklediği birden fazla halka kullanılır.

**CACHE'İN DMA KULLANIMI** Geleneksel DMA işleminde, veri yolu, halka veya QPI noktadan noktaya matris gibi sistem ara bağlantı yapısı aracılığıyla ana bellek ve bir G/C cihazı arasında veri alışverişi yapılır. Örneğin Xeon E5-2600/4600 geleneksel bir DMA teknigi kullanıyordu, çıktı aşağıdaki gibi ilerlerdi. Çekirdek üzerinde çalışan bir G/C sürücüsü, G/C denetleyicisine (Şekil 7.16'da PCIe olarak etiketlenmiştir) aktarılacak verileri içeren ana bellekteki arabelleğin konumunu ve boyutıyla birlikte bir G/C komutu gönderir. G/C denetleyicisi, DDR3 bellekteki verilere erişen ve G/C denetleyicisine teslim edilmek üzere sistem halkasına yerleştirilen bellek hub'ına (MCH) yönlendirilen bir okuma isteği yayınlar. L3 önbellek bu işleme dahil değildir ve bir veya daha fazla çip dışı bellek okuması gereklidir. Benzer şekilde, giriş için veriler G/C denetleyicisinden gelir ve sistem halkası üzerinden MCH'ye iletilir ve ana belleğe yazılır. MCH ayrıca güncellenen bellek konumlarına karşılık gelen L3 önbellek satırlarını da geçersiz kılmaktadır. Bu durumda, bir veya daha fazla çip dışı bellek yazımı gereklidir. Ayrıca, bir uygulama yeni verilere erişmek isterse, bir ana bellek okuması gereklidir.



Şekil 7.16 Xeon E5-2600/4600 Yonga Mimarisi

Büyük miktarlarda son seviye önbelleğin mevcut olmasıyla, daha verimli bir teknik mümkündür ve Xeon E5-2600/4600 tarafından kullanılır. Çıkış için, G/C denetleyicisi bir okuma isteği gönderdiğinde, MCH ilk olarak verilerin L3 önbellekte olup olmadığını kontrol eder. Bir uygulama yakın zamanda çıkış yapılacak bellek bloğuna veri yazmışsa bu durum söz konusu olabilir. Bu durumda, MCH verileri L3 önbellekten G/C denetleyicisine yönlendirir; ana bellek erişimine gerek yoktur. Bununla birlikte, verilerin önbellekten çıkarılmasına da neden olur, yani bir I/O cihazı tarafından okuma eylemi

verilerin tahliye edilmesine neden olur. Böylece, G/C işlemi ana bellek erişimi gerektirmediği için verimli bir şekilde ilerler. Ancak, bir uygulama gelecekte bu verilere ihtiyaç duyarsa, ana bellekten L3 önbelleğine geri okunması gereklidir. Xeon E5-2600/4600 üzerindeki giriş işlemi bir önceki paragrafta açıkladığı gibi çalışır; L3 önbelleği dahil değildir. Bu nedenle, performans iyileştirmesi yalnızca çıkış işlemlerini içerir.

Son bir nokta. Çıkış aktarımı *doğrudan önbellekten* G/C denetleyicisinemasına rağmen, *doğrudan önbellek erişimi* terimi bu özellik için kullanılmaz. Bunun yerine, bu terim bu bölümün geri kalanında açıkladığı gibi G/C protokolü uygulaması için ayrılmıştır.

## Önbellekle İlgili Performans Sorunları

Ağ trafiği, paketler veya protokol veri birimleri olarak adlandırılan bir dizi protokol bloğu şeklinde iletilir. En düşük veya bağlantı seviyesi protokolü tipik olarak Ethernet'tir, böylece her gelen ve giden veri bloğu, yük olarak daha üst seviye protokol paketini içeren bir Ethernet paketinden oluşur. Üst düzey protokoller genellikle Ethernet üzerinde çalışan İnternet Protokolü (IP) ve IP üzerinde çalışan İletim Kontrol Protokolüdür (TCP). Buna göre Ethernet yükü bir TCP başlığı ve bir IP başlığı içeren bir veri bloğundan oluşur. Giden veriler için Ethernet paketleri G/C kontrolörü veya ağ arayüz kontrolörü (NIC) gibi bir çevresel bileşende oluşturulur. Benzer şekilde, gelen trafik için, G/C denetleyicisi Ethernet bilgilerini çıkarır ve TCP/IP paketini ana CPU'ya ileter.

Hem giden hem de gelen trafik için çekirdek, ana bellek ve önbellek devreye girer. Bir DMA şemasında, bir uygulama veri istediğiinde, bu veriyi ana belleke uygulama tarafından atanmış bir tampona yerleştirir. Çekirdek bunu ana bellekteki bir sistem tamponuna aktarır ve gerekli TCP ve IP başlıklarını oluşturur, bunlar da sistem belleğinde tamponlanır. Paket daha sonra NIC üzerinden aktarılacak üzere DMA aracılığıyla alınır. Bu faaliyet yalnızca ana belleği değil aynı önbelleği de meşgul eder. Gelen trafik için, sistem ve uygulama tamponları arasında benzer aktarımalar gereklidir.

Büyük hacimli protokol trafiği işlendiğinde, bu senaryodaki iki faktör performansı düşürür. Birincisi, çekirdek sistem ve uygulama tamponları arasında veri kopyalarken değerli saat döngülerini tüketir. Ikincisi, bellek hızları CPU hızlarına ayak uyduramadığından, çekirdek bellek okuma ve yazmalarını beklerken zaman kaybeder. Protokol trafiğini işlemeenin bu geleneksel yolunda, önbellek yardımcı olmaz çünkü veriler ve protokol başlıklarını sürekli değiştir ve bu nedenle önbelleğin sürekli güncellenmesi gereklidir.

Performans konusunu açıklığa kavuşturmak ve performansı artırmanın bir yolu olarak DCA'nın faydasını açıklamak için, gelen trafik için protokol trafiğinin işlenmesine daha ayrıntılı olarak bakalım. Genel anlamda, aşağıdaki adımlar gerçekleşir:

- 1. Paket geldi:** NIC gelen bir Ethernet paketini alır. NIC, Ethernet kontrol bilgilerini işler ve çıkarır. Bu, bir hata algılama hesaplaması yapmayı da içerir. Kalan TCP/IP paketi daha sonra genellikle NIC'nin bir parçası olan sistemin DMA modülüne aktarılır. NIC ayrıca paket hakkında bellekteki tampon konumu gibi bilgiler içeren bir paket tanımlayıcısı oluşturur.

2. **DMA:** DMA modülü, paket tanımlayıcısı da dahil olmak üzere verileri ana belleğe aktarır. Ayrıca varsa ilgili önbellek satırlarını da geçersiz kılmalıdır.
3. **NIC ana bilgisayarı keser:** Birkaç paket aktarıldıktan sonra, NIC ana bilgisayar işlemcisine bir kesme gönderir.
4. **Tanımlayıcıları ve başlıklarını alın:** Çekirdek, alınan paketlerin tanımlayıcısını ve başlığını okuyan bir kesme işleme prosedürünü çağırarak kesmeyi işler.
5. **Önbellek ıskası meydana gelir:** Bu yeni bir veri olduğu için, yeni veriyi içeren sistem tamponuna karşılık gelen önbellek satırları geçersiz kılır. Bu nedenle çekirdek, verileri ana bellekten önbelleğe ve ardından çekirdek kayıtlarına okumak için durmalıdır.
6. **Başlık işlenir:** Protokol yazılımı, TCP ve IP başlıklarının içeriğini analiz etmek için çekirdek üzerinde çalışır. Bu, muhtemelen TCP ile ilgili bağlam bilgilerini içeren bir taşıma kontrol bloğuna (TCB) erişimi içerecektir. TCB erişimi, bir ana bellek erişimini gerektiren önbellek özlemesini tetikleyebilir veya tetiklemeyebilir.

**7. Yük aktarılır:** Paketin veri kısmı sistem tamponundan uygun uygulama tamponuna aktarılır.

Giden paket trafiği için de benzer bir adım dizisi gerçekleşir, ancak önbelleğin nasıl yönetildiğini etkileyen bazı farklılıklar vardır. Giden trafik için aşağıdaki adımlar gerçekleşir:

1. **Paket aktarımı talep edildi:** Bir uygulama uzak bir sisteme aktarılacak bir veri bloğuna sahip olduğunda, verileri bir uygulama arabelleğine yerleştirir ve işletim sistemini bir tür sistem çağrıları ile uyarır.
2. **Paket oluşturuldu:** İşletim sistemi, iletişim için TCP/IP paketini oluşturmak üzere bir TCP/IP işlemi çağırır. TCP/IP işlemi TCB'ye (önbellek ıskalamasını içerebilir) ve uygun başlıklar oluşturur. Ayrıca uygulama arabelleğindeki verileri okur ve ardından tamamlanan paketi (başlıklar artı veriler) bir sistem arabelleğine yerleştirir. Sistem buf- ferine yazılan verilerin önbellekte de bulunduğuunu unutmayın. TCP/IP işlemi ayrıca DMA modülü ile paylaşılan belleğe yerleştirilen bir paket tanımlayıcısı oluşturur.
3. **Çıkış işlemi çağrıldı:** Bu, DMA modülüne çıkışın NIC için hazır olduğunu bildirmek için bir aygit sürücüsü programı kullanır.
4. **DMA aktarımı:** DMA modülü paket tanımlayıcısını okur, ardından ana bellekten veya son seviye önbellekten NIC'ye bir DMA aktarımı gerçekleştirir. DMA aktarımlarının, okuma durumunda bile (DMA modülü tarafından) önbellekteki önbellek satırını geçersiz kıldığını unutmayın. Satır değiştirilirse, bu bir geri yazmaya neden olur. Geçersiz kılma işlemini çekirdek yapmaz. Geçersiz kılma işlemi DMA modülü verileri okudüğünde gerçekleşir.
5. **NIC tamamlanma sinyali verir:** Aktarım tamamlandıktan sonra NIC, gönderme sinyalini oluşturan çekirdekteki sürücüye sinyal gönderir.
6. **Sürücü arabelleği serbest bırakır:** Sürücü tamamlanma bildirimini aldıktan sonra tampon alanını yeniden kullanım için boşaltır. Çekirdek ayrıca tampon verisini içeren önbellek satırlarını da geçersiz kılmalıdır.

Gördüğü gibi, ağ G/C'si önbelleğe ve ana belleğe bir dizi erişimi ve verilerin bir uygulama tamponu ile bir sistem tamponu arasında hareket etmesini içerir. Hem çekirdek hem de ağ performansı bellek erişim sürelerindeki kazanımları geride bıraktığından, ana belleğin yoğun katılımı bir darboğaz haline gelir.

## Doğrudan Önbellek Erişimi Stratejileri

Ağ I/O için önbelleklerin daha verimli kullanılması için çeşitli stratejiler önerilmiştir ve bu stratejilerin tümüne *doğrudan önbellek erişimi* genel terimi uygulanmıştır.

En basit strateji, 2006 ve 2010 yılları arasında bir dizi Intel Xeon işlemcisinde prototip olarak uygulanan stratejidir [KUMA07, INTE08]. Bu DCA biçimini yalnızca gelen ağ trafiği için geçerlidir. Bellek denetleyicisindeki DCA işlevi, veriler sistem belleğinde mevcut olur olmaz çekirdeğe bir öngörme ipucu gönderir. Bu, çekirdeğin veri paketini sistem tamponundan önceden almasını sağlar, böylece önbellek kaçırılmaları ve ilgili çekirdek döngülerinin israfı önlenir.

DCA'nın bu basit şekli bir miktar iyileşme sağlasa da, ana bellekteki sistem tamponundan tamamen kaçınarak çok daha önemli kazanımlar elde edilebilir. Protokol işlemenin özel işlevi için, paket ve paket tanımlayıcı bilgilerine çekirdek tarafından sistem tamponunda yalnızca bir kez erişildiğine dikkat edin. Gelen paketler için çekirdek verileri tampondan okur ve paket yükünü bir uygulama tamponuna aktarır. Sistem tamponundaki bu verilere tekrar erişmesine gerek yoktur. Benzer şekilde, giden paketler için, çekirdek verileri sistem tamponuna yerleştirildikten sonra, bu verilere tekrar erişmesine gerek yoktur. Bu nedenle, G/C sisteminin yalnızca ana belleğe doğrudan erişmeyeceğini kalmayıp, aynı zamanda hem giriş hem de çıkış işlemleri için önbelleğe erişime yeteneğiyle donatıldığını varsayıyalım. O zaman gelen ve giden paketlerin paketlerini ve tanımlayıcılarını tamponlamak için ana bellek yerine son seviye önbelleği kullanmak mümkün olacaktır.

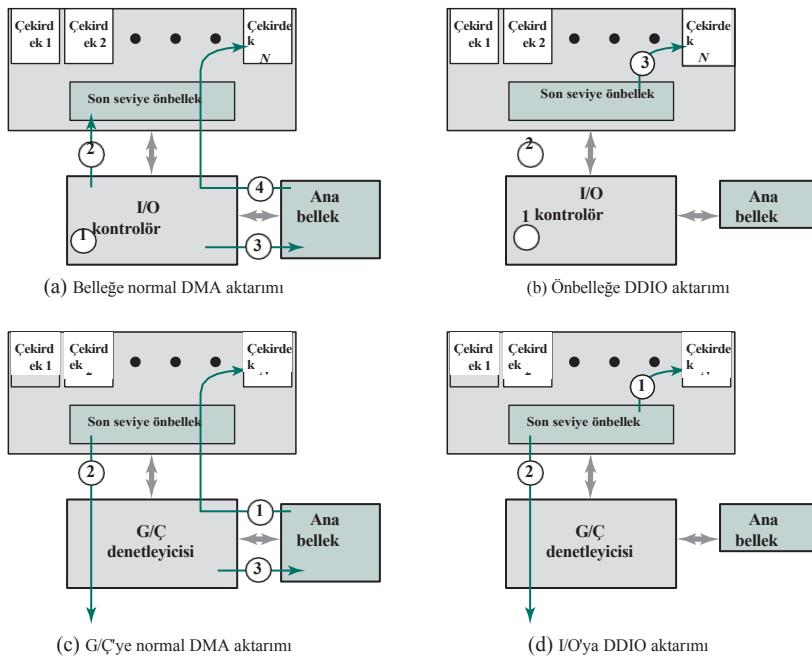
Gerçek bir DCA olan bu son yaklaşım [HUGG05]'de önerilmiştir. Ayrıca **önbellek enjeksiyonu** olarak da tanımlanmıştır [LEON06]. DCA'nın bu daha eksiksiz biçiminin bir versiyonu Intel'in Xeon işlemci serisinde **Direct Data I/O** [INTE12] olarak uygulanmaktadır.

## Doğrudan Veri I/O

Intel Direct Data I/O (DDIO), tüm Xeon E5 işlemci ailesinde uygulanmaktadır. Çalışması en iyi DDIO'lu ve DDIO'suz aktarımların yan yana karşılaştırılmasıyla açıklanabilir.

**Packet Input** İlk olarak, NIC'ye ağdan gelen bir paketin durumuna bakacağız. Şekil 7.17a bir DMA işlemi için gerekli adımları göstermektedir. NIC bir bellek yazma işlemi başlatır (1). Ardından NIC, sistem tamponuna karşılık gelen önbellek satırlarını geçersiz kılar (2). Ardından, DMA işlemi gerçekleştirilir ve paket doğrudan ana belleğe yerleştirilir (3). Son olarak, ilgili çekirdek bir DMA kesme sinyali aldıktan sonra, çekirdek paket verilerini önbellek aracılığıyla okuyabilir (4).

DDIO kullanarak gelen bir paketin işlenmesini tartışmadan önce, Bölüm 4'teki önbellek yazma politikası tartışmasını özetlememiz ve yeni bir teknik tanıtımız gereklidir. Aşağıdaki tartışma için, çok işlemcili veya çok çekirdekli bir ortamda ortaya çıkan önbellek tutarlılığı ile ilgili sorunlar vardır. Bunlar tartışılmıştır



Şekil 7.17 DMA ve DDIO karşılaştırması

Bölüm 17'de açıklanmıştır, ancak ayrıntıların burada bizi ilgilendirmesine gerek yoktur. Bir önbellek satırındaki güncellemeyle başa çıkmak için iki teknik olduğunu hatırlayın:

- **Yazma işlemi:** Tüm yazma işlemleri önbelleğin yanı sıra ana belleğe de yapılır ana belleğin her zaman geçerli olması sağlanır. Diğer herhangi bir çekirdek önbellek modülü, kendi yerel önbelleği içinde tutarlılığı korumak için ana belleğe giden trafiği izleyebilir.
- **Geri yaz:** Güncellemeler yalnızca önbellekte yapılır. Bir güncelleme gerçekleştiğinde, satırla ilişkili bir kirli bit ayarlanır. Ardından, bir blok değiştirildiğinde, yalnızca kirli bit ayarlanmışsa ve ayarlanmışsa ana belleğe geri yazılır.

DDIO, L3 önbelleğinde geri yazma stratejisini kullanır.

Bir önbellek yazma işlemi, iki stratejiden ele alınan bir önbellek kaçırma durumuyla karşılaşabilir:

- **Write allocate:** Gerekli satır ana bellekten önbelleğe yüklenir. Ardından, önbellekteki satır yazma işlemi ile güncellenir. Bu şema tipik olarak geri yazma yöntemiyle kullanılır.
- **Yazılı olmayan tahsis:** Blok doğrudan ana bellekte değiştirilir. Önbellekte herhangi bir değişiklik yapılmaz. Bu şema tipik olarak write-through yöntemiyle kullanılır.

Yukarıdakileri akılda tutarak, NIC tarafından başlatılan gelen transferler için DDIO stratejisini tanımlayabiliriz.

1. Bir önbellek isabeti varsa, önbellek satırı güncellenir, ancak ana bellek güncellenmez; bu sadece bir önbellek isabeti için geri yazma stratejisidir. Intel literatürü bunu **yazma güncellemesi** olarak adlandırır.

- Bir önbellek iskasi varsa, yazma işlemi önbellekte ana bellege geri yazılmayacak bir satırı gerçekleştirecektir. Sonraki yazmalar, yine ana bellege referans olmadan veya bu veriyi ana bellege yazan gelecekteki bir eylem olmadan önbellek satırını günceller. Intel dokümantasyonu [INTE12] bunu *write allocate* olarak adlandırır, ancak ne yazık ki bu terim genel önbellek literatüründe aynı anlama gelmemektedir.

DDIO stratejisi bir ağ protokolü uygulaması için etkilidir çünkü gelen verilerin ileride kullanılmak üzere saklanması gerekmektedir. Protokol uygulaması verileri bir uygulama tamponuna yazacaktır ve geçici olarak bir sistem tamponunda saklamaya gerek yoktur.

Şekil 7.17b DDIO girişi için işlemi göstermektedir. NIC bir bellek yazma işlemi başlatır (1). Ardından NIC, sistem tamponuna karşılık gelen önbellek satırlarını geçersiz kılar ve gelen verileri önbelleğe yerleştirir (2). Son olarak, uygun çekirdek bir DCA kesme sinyali aldıktan sonra, çekirdek paket verilerini önbellekten okuyabilir (3).

**PacKET ÇIKIŞI** Şekil 7.17c, giden paket iletimi için bir DMA işleminin adımlarını göstermektedir. Çekirdek üzerinde çalışan TCP/IP protokol işleyicisi verileri bir uygulama tamponundan okur ve bir sistem tamponuna yazar. Bu veri erişim işlemleri önbellek iskalamalarına neden olur ve verilerin bellekten ve L3 önbelleğinden okunmasına neden olur (1). NIC bir iletim işlemini başlatmak için bildirim aldıktan sonra, verileri L3 önbelleğinden okur ve iletir (2). NIC tarafından yapılan önbellek erişimi, verilerin önbellekten çıkarılmasına ve ana bellege geri yazılmasına neden olur (3).

Şekil 7.17d, paket aktarımı için bir DDIO işleminin adımlarını göstermektedir. TCP/IP protokol işleyicisi iletilecek paketi oluşturur ve ana bellekte (2) değil, L3 önbelleğinde (1) ayrılmış alanda saklar. NIC tarafından başlatılan okuma işlemi, ana bellege tahliyeeye neden olmadan önbellekten gelen verilerle karşılaşır.

Bu yan yana karşılaştırmalardan DDIO'nun hem gelen hem de giden paketler için DMA'dan daha verimli olduğu ve bu nedenle yüksek paket trafiği hızına daha iyi ayak uydurabildiği açıkça anlaşılmalıdır.

## 7.7 G/Ç KANALLARI VE PROSESLER

### G/Ç İşlevinin Evrimi

Bilgisayar sistemleri gelişikçe, her bir bileşenin karmaşaklılığı ve gelişmişliği de artmıştır. Bu hiçbir yerde I/O fonksiyonunda olduğu kadar belirgin değildir. Bu evrimin bir kısmını zaten gördük. Evrimsel adımlar aşağıdaki gibi özetlenebilir:

- CPU doğrudan bir çevresel cihazı kontrol eder. Bu, basit mikroişlemci kontrollü cihazlarda görülür.
- Bir kontrolör veya I/O modülü eklenir. CPU programlanmış I/O'yu kesmeler olmadan kullanır. Bu adımla CPU, harici cihaz arayüzlerinin özel ayrıntılarından bir şekilde ayrılmış olur.
- Adım 2'deki ile aynı yapılandırma kullanılır, ancak şimdi kesmeler . CPU'nun bir I/O işleminin gerçekleştirilmesini beklemek için zaman harcaması gerekmek, böylece verimlilik artar.

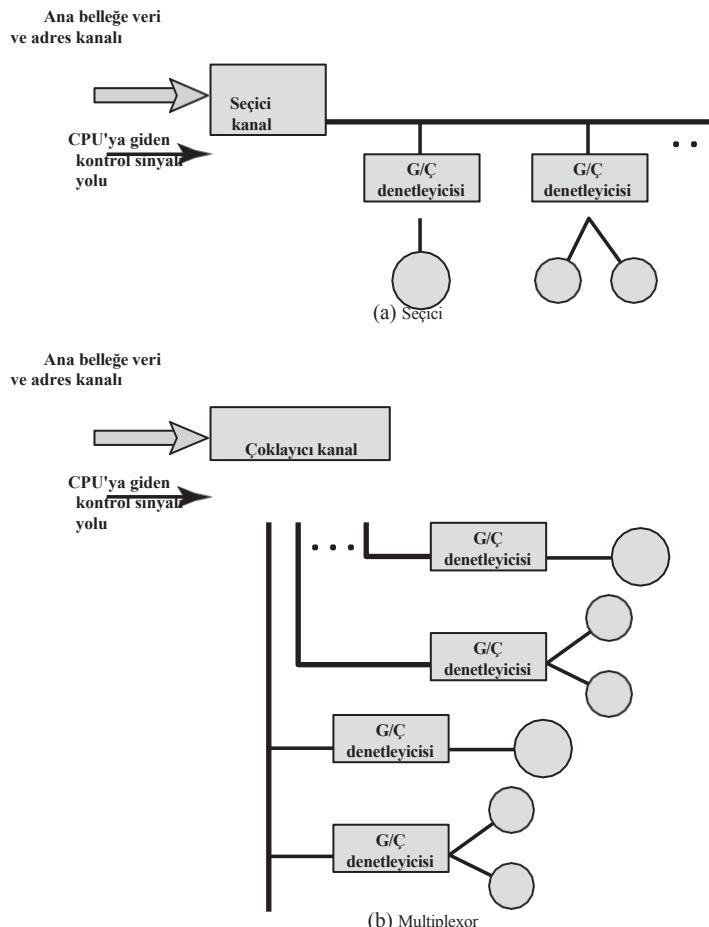
4. I/O modülüne DMA aracılığıyla belleğe doğrudan erişim verilir. Artık aktarımın başlangıcı ve sonu dışında CPU'yu dahil etmeden bir veri bloğunu belleğe veya bellekten taşıyabilir.
5. G/C modülü, için uyarlanmış özel bir komut seti ile kendi başına bir işlemci olacak şekilde geliştirilmiştir. CPU, I/O işlemcisini bellekteki bir I/O programını yürütmesi için yönlendirir. G/C işlemcisi bu talimatları CPU müdahalesi olmadan alır ve yürütür. Bu, CPU'nun bir dizi G/C faaliyeti belirlemesine ve yalnızca tüm dizi gerçekleştirildiğinde kesintiye uğramasına olanak tanır.
6. G/C modültünün kendine ait bir yerel belleği ve aslında kendi başına bir bilgisayardır. Bu mimari ile çok sayıda I/O cihazı minimum CPU müdahalesi ile kontrol edilebilir. Böyle bir mimari için yaygın bir kullanım, etkileşimli terminallerle iletişimini kontrol etmek olmuştur. G/C işlemcisi terminalleri kontrol etmekle ilgili görevlerin çoğunu üstlenir.

Bu evrimsel yolda ilerledikçe, G/C işlevinin giderek daha fazla CPU müdahalesi olmadan gerçekleştirilebilir. CPU, I/O ile ilgili görevlerden giderek daha fazla kurtulur ve performansı artırır. Son iki adımda (5-6), bir programı yürütürebilen bir G/C modülü kavramının ortaya çıkmasıyla büyük bir değişiklik meydana gelir. Adım 5 için G/C modülü genellikle *G/C kanalı* olarak adlandırılır. Adım 6 için genellikle *G/C işlemcisi* terimi kullanılır. Bununla birlikte, her iki terim de zaman zaman her iki duruma da uygulanır. Aşağıda, *G/C kanalı* terimini kullanacağız.

### **G/C Kanallarının Özellikleri**

G/C kanalı, DMA konseptinin bir uzantısını temsil eder. Bir G/C kanalı, G/C işlemleri üzerinde tam kontrol sağlayan G/C talimatlarını yürütme yeteneğine sahiptir. Bu tür cihazlara sahip bir bilgisayar sisteminde, CPU G/C talimatlarını yürütmez. Bu talimatlar, G/C kanalının kendisinde bulunan özel amaçlı bir işlemci tarafından yürütülmek üzere ana bellekte saklanır. Böylece CPU, G/C kanalına bellekteki bir programı yürütme talimatı vererek bir G/C aktarımı başlatır. Program aygit ya da aygitları, depolama için bellek alanını ya da alanlarını, önceliği ve belirli hata durumları için yapılacak işlemleri belirtir. G/C kanalı bu talimatları izler ve veri aktarımını kontrol eder.

Şekil 7.18'de gösterildiği gibi iki tür I/O kanalı yaygındır. Bir *seçici kanal* birden fazla yüksek hızlı cihazı kontrol eder ve herhangi bir zamanda bu biriyle veri aktarımına adanmıştır. Böylece, G/C kanalı bir cihazı seçer ve veri aktarımını gerçekleştirir. Her bir cihaz ya da küçük bir cihaz grubu, daha önce tartıştığımız I/O modüllerine çok benzeyen *bir kontrolör* ya da I/O modülü tarafından idare edilir. Böylece, G/C kanalı bu G/C denetleyicilerini kontrol etmede CPU'nun yerine hizmet eder. Bir *çoklayıcı kanal* aynı anda birden fazla aygitla G/C işlemini gerçekleştirebilir. Düşük hızlı cihazlar için, bir *bütçet çoklayıcı* karakterleri mümkün olduğunda hızlı bir şekilde birden fazla cihaza kabul eder veya iletir. Örneğin, farklı hızlara ve bağımsız akışlara sahip üç cihazdan elde edilen karakter aksişi  $A_1A_2A_3A_4 \hookleftarrow$ ,  $B_{(1)}B_{(2)}B_{(3)}B_{(4)}\hookleftarrow$  ve  $C_{(1)}C_{(2)}C_{(3)}C_{(4)}\hookleftarrow$   $A_{(1)}B_{(1)}C_{(1)}A_{(2)}C_{(2)}A_{(3)}B_{(2)}C_{(3)}A_{(4)}$ , vb. olabilir. Yüksek hızlı aygitlar için, bir *blok çoklayıcı* birkaç aygıtta gelen veri bloklarını birbirine ekler.



**Sekil 7.18** G/C Kanal Mimarisi

## 7.8 DIŞ BAĞLANTI STANDARTLARI

Bu bölümde, G/C'yi desteklemek için en yaygın kullanılan harici ara yüz standartlarına kısa bir genel bakış sunuyoruz. Bunlardan ikisi, Thunderbolt ve InfiniBand, Ek J'de ayrıntılı olarak incelenmiştir.

### Evransel Seri Veri Yolu (USB)

USB, çevresel bağlantılar için yaygın olarak kullanılır. Klavye ve işaretleme aygıtları gibi daha yavaş hızlı aygıtlar için varsayılan arabirimdir, ancak yazıcılar, disk sürücülerleri ve ağ bağıdıştırıcıları dahil olmak üzere yüksek hızlı G/C için de yaygın olarak kullanılır.

USB birden fazla nesilden geçmiştir. İlk versiyon olan USB 1.0, 1,5 Mbps'lik bir *Düşük Hız* veri hızı ve 12 Mbps'lik bir *Tam Hız* veri hızı tanımlamıştır. USB 2.0 480 Mbps veri hızı sağlar. USB 3.0 yeni, daha yüksek hızlı bir veri yolu içerir.

USB 2.0 veri yoluna paralel olarak *SuperSpeed* olarak adlandırılır. Super Speed'in sinyalleşme hızı 5 Gbps'dır, ancak sinyalleşme ek yükü nedeniyle kullanılabılır veri hızı 4 Gbps'ye kadardır. En yeni spesifikasiyon, *SuperSpeed+* adı verilen daha hızlı bir aktarım modu içeren USB 3.1'dir. Bu aktarım modu 10 Gbps sinyalleşme hızına ve 9,7 Gbps teorik kullanılabılır veri hızına ulaşır.

Bir USB sistemi, hiyerarşik bir ağaç topolojisi ile yerel bir ağ oluşturmak için cihazlara bağlanan bir kök ana bilgisayar denetleyicisi tarafından kontrol edilir.

### **FireWire Seri Veri Yolu**

FireWire, kişisel bilgisayarlar, iş istasyonları ve sunucular gibi daha küçük sistemlerde kullanılmak üzere küçük bilgisayar sistemi arayüzüne (SCSI) alternatif olarak geliştirilmiştir. Amaç, ana bilgisayar ve süper bilgisayar sistemleri için geliştirilen hantal ve pahalı I/O kanal teknolojilerinden kaçınırken, bu sistemlerde yüksek I/O hızlarına yönelik artan talepleri karşılamaktır. Sonuç, genellikle FireWire olarak bilinen Yüksek Performanslı Seri Veri Yolu için IEEE standarı 1394'tür.

FireWire, tek bir bağlantı noktasından 63 adede kadar cihazın bağlanıldığı bir papatya zinciri yapılandırması kullanır. Ayrıca, köprüler kullanılarak 1022 adede kadar FireWire veriyolu birbirine bağlanabilir ve böylece bir sistemin gerektiği kadar çevre birimini desteklemesi sağlanabilir.

FireWire, hot plugging olarak bilinen, bilgisayar sistemini kapatmaya veya sistemi yeniden yapılandırmaya gerek olmadan çevre birimlerini bağlamayı ve çıkarmayı mümkün kılan özelliğidir. Ayrıca FireWire otomatik yapılandırma sağlar; aygit kimliklerini manuel olarak ayarlamak veya aygitların ilgili konumlarıyla ilgilenmek gerekmek. FireWire ile sonlandırma yoktur ve sistem adresleri atamak için otomatik olarak bir yapılandırma işlevi gerçekleştirir. Bir FireWire veri yolunun katı bir papatya zinciri olması gerekmek. Bunun yerine, ağaç yapılı bir yapılandırma mümkündür.

FireWire standardının önemli bir özelliği, ana bilgisayar sisteminin seri veri yolu üzerinden çevre cihazlarıyla etkileşime girme şeklini standartlaştmak için bir dizi üç protokol katmanı belirlemesidir. Fiziksel katman, FireWire kapsamında izin verilen iletim ortamını ve her birinin elektrik ve sinyal özelliklerini tanımlar. Veri hızları 25 Mbps'den 3.2 Gbps'ye kadar tanımlanmıştır. Bağlantı katmanı, paketlerdeki veri iletimini tanımlar. İşlem katmanı, FireWire'in alt katman ayrıntılarını uygulamalardan gizleyen bir istek-yanıt protokolü tanımlar.

### **Küçük Bilgisayar Sistemi Arayüzü (SCSI)**

SCSI, çevre aygıtlarını (diskler, modemler, yazıcılar, vb.) küçük ve orta ölçekli bilgisayarlara bağlamak için bir zamanlar yaygın olan bir standarttır. SCSI daha yüksek veri hızlarına evrilmiş olsa da, daha küçük sistemlerde USB ve FireWire gibi rakiplerine karşı popülerliğini kaybetmiştir. Bununla birlikte, SCSI'nin hızlı sürümleri kurumsal sistemlerde yoğun bellek desteği için popülerliğini korumaktadır. Örneğin, IBM zEnterprise EC12 ve diğer IBM ana bilgisayaları SCSI desteği sunar ve bir dizi Seagate sabit disk sistemi SCSI kullanır.

SCSI'nin fiziksel organizasyonu, standardın nesline bağlı olarak 16 veya 32 cihazı destekleyebilen paylaşılan bir veriyoludur. Veri yolu, önceki nesillerde 16 bit ve sonraki nesillerde 32 bit veri yolu genişliği ile seri yerine paralel iletim sağlar. Hızlar orijinal SCSI-1 spesifikasiyonunda 5 Mbps ile SCSI-3 U3'te 160 Mbps arasında değişir.

## Thunderbolt

Genel amaçlı kullanıma sunulan en yeni ve en hızlı çevre birimi bağlantı teknolojisi, Intel tarafından Apple işbirliğiyle geliştirilen Thunderbolt'tur. Bir Thunderbolt kablosu daha önce birden fazla kablounun yapması gereken işi yapabilmektedir. Bu teknoloji sabit diskler, RAID (Redundant Array of Independent Disks) dizileri, video yakalama kutuları ve ağ arayüzleri gibi çevre birimleri için veri, video, ses ve gücü tek bir yüksek hızlı bağlantıda birleştiriyor. Her yönde 10 Gbps'ye kadar çıkış ve bağlı çevre birimlerine 10 watt'a kadar güç sağlar.

Thunderbolt Ek J'de ayrıntılı olarak açıklanmıştır.

## InfiniBand

InfiniBand, üst düzey sunucu pazarını hedefleyen bir I/O spesifikasyonudur. Spesifikasyonun ilk versiyonu 2001 yılının başlarında piyasaya sürülmüş ve çok sayıda satıcının ilgisini çekmiştir. Örneğin, IBM zEnterprise serisi ana bilgisayarlar birkaç yıldır büyük ölçüde InfiniBand'e . Standart, işlemciler ve akıllı I/O cihazları arasındaki veri akışı için bir mimariyi ve özellikleri tanımlamaktadır. InfiniBand, depolama alanı ağı ve diğer büyük depolama konfigürasyonları için popüler bir arayüz haline gelmiştir. Temelde InfiniBand, sunucuların, uzak depolama alanlarının ve diğer ağ cihazlarının anahtarlar ve bağlantılardan oluşan merkezi bir yapıya bağlanmasını sağlar. Anahtar tabanlı mimari 64.000'e kadar sunucuya, depolama sistemini ve ağ cihazını birbirine bağlayabilir.

Infiniband Ek J'de ayrıntılı olarak açıklanmaktadır.

## PCI Express

PCI Express, çok çeşitli tip ve hızlardaki çevre birimlerini bağlamak için kullanılan yüksek hızlı bir veri yolu sistemidir. Bölüm 3'te PCI Express ayrıntılı olarak ele alınmaktadır.

## SATA

Serial ATA (Serial Advanced Technology Attachment) disk depolama sistemleri için bir arabirimdir. Cihaz başına maksimum 300 Mbps olmak üzere 6 Gbps'ye kadar veri hızları sağlar. SATA masaüstü bilgisayarlarda, endüstriyel ve gömülü uygulamalarda yaygın olarak kullanılmaktadır.

## Ethernet

Ethernet, evlerde, ofislerde, veri merkezlerinde, işletmelerde ve geniş alan ağlarında kullanılan baskın kablolu ağ teknolojisidir. Ethernet, 100 Gbps'ye kadar veri hızlarını ve birkaç metreden onlarca km'ye kadar mesafeleri destekleyecek şekilde gelişikçe, büyük ve küçük kuruluşlardaki kişisel bilgisayarıları, iş istasyonlarını, sunucuları ve büyük veri depolama cihazlarını desteklemek için gerekli hale gelmiştir.

Ethernet deneysel bir veri yolu tabanlı 3 Mbps sistemi olarak başladı. Bir veri yolu sistemi ile, PC'ler gibi bağlı tüm cihazlar, ev tipi kablo TV sistemlerine benzer şekilde ortak bir koaksiyel kabloya bağlanır. Ticari olarak kullanılabilen ilk Ether- net ve IEEE 802.3'un ilk versiyonu 10 Mbps hızında çalışan bus tabanlı sistemlerdi. Teknoloji ilerledikçe, Ethernet veri yolu tabanlıdan anahtar tabanlıya geçti ve veri hızı periyodik olarak büyülük sırasına göre arttı. İle

## 266 BÖLÜM 7 / GİRİŞ/ÇIKIŞ

anahtar tabanlı sistemlerde, tüm cihazların doğrudan anahtara bağlı olduğu merkezi bir anahtar vardır. Şu anda Ethernet sistemleri 100 Gbps'ye varan hızlarda mevcuttur. İşte kısa bir kronoloji.

- 1983: 10 Mbps (megabit per second, saniyede milyon bit)
- 1995: 100 Mbps
- 1998: 1 Gbps (gigabit per second, saniyede milyar bit)
- 2003: 10 Gbps
- 2010: 40 Gbps ve 100 Gbps

### Wi-Fi

Wi-Fi, evlerde, ofislerde ve kamusal alanlarda kullanılan baskın kablosuz Internet erişim teknolojisidir. Evlerdeki Wi-Fi artık bilgisayarları, tabletleri, akıllı telefonları ve video kameralar, TV'ler ve termostatlar gibi bir dizi elektronik cihazı birbirine bağlamaktadır. İşletmelerde Wi-Fi, çalışan verimliliğini ve ağ etkinliğini artırmanın temel bir aracı haline gelmiştir. Halka açık Wi-Fi erişim noktaları ise çoğu halka açık yerde ücretsiz Internet erişimi sağlamak üzere önemli ölçüde genişlemiştir.

Anten teknolojisi, kablosuz iletişim teknikleri ve kablosuz protokol tasarımları gelişikçe, IEEE 802.11 komitesi daha yüksek hızlarda Wi-Fi'nin yeni sürümleri için standartlar getirebilmiştir. Standart yayılmışından sonra endüstri hızlı bir şekilde ürünlerini geliştirmektedir. Burada, basitçe IEEE 802.11 olarak adlandırılan orijinal standarttan başlayarak ve her sürüm için maksimum veri hızını gösteren kısa kronoloji bulunmaktadır:

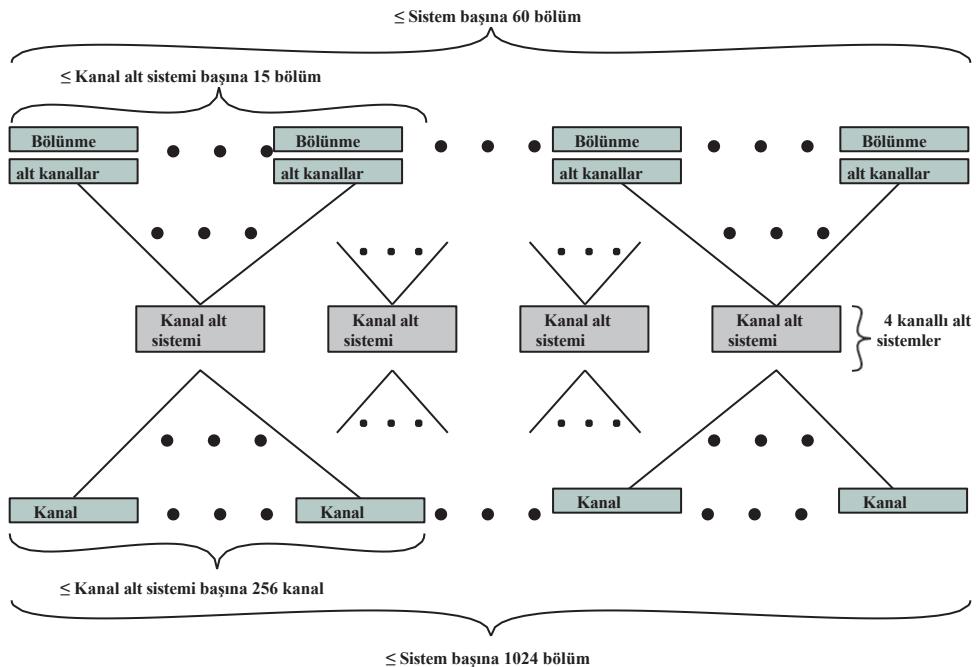
- 802.11 (1997): 2 Mbps (megabit per second, saniyede milyon bit)
- 802.11a (1999): 54 Mbps
- 802.11b (1999): 11 Mbps
- 802.11n (1999): 600 Mbps
- 802.11g (2003): 54 Mbps
- 802.11ad (2012): 6.76 Gbps (saniye başına milyar bit)
- 802.11ac (2014): 3,2 Gbps

## 7.9 IBM zENTERPrISE EC12 I/O YAPISI

zEnterprise EC12 IBM'in en yeni ana bilgisayar ürünüdür (bu yazının yazıldığı sırada). Sistem, altı çekirdekli 5,5 GHz çok çekirdekli bir çip olan zEC12 işlemci çipinin kullanımına dayanmaktadır. zEC12 mimarisi, toplam 606 çekirdek için maksimum 101 işlemci yongasına sahip olabilir. Bu bölümde, zEnterprise EC12'nin I/O yapısına bakacağız.

### Kanal Yapısı

zEnterprise EC12, tüm G/Ç işlemlerini yöneten ve bu işlem ve bellek yükünü ana sistemden tamamen alan özel bir G/Ç alt sistemine sahiptir.



Şekil 7.19 IBM zEC12 I/O Kanal Alt Sistem Yapısı

İşlemciler. Şekil 7. 21 G/Ç alt sisteminin mantıksal yapısını göstermektedir. 96 çekirdek işlemciden en fazla 4 tanesi G/Ç kullanımı için ayrılabilir ve böylece 4 **kanal alt sistemi (CSS)** oluşturulur. Her CSS aşağıdaki unsurlardan oluşur:

- **Sistem yardımcı işlemcisi (SAP):** SAP, G/Ç işlemi için yapılandırılmış bir çekirdek işlemcidir. Görevi G/Ç işlemlerini boşaltmak ve kanalları ve G/Ç işlem kuyruklarını yönetmektir. Diğer işlemcileri tüm G/Ç görevlerinden kurtararak uygulama mantığına adanmalarını sağlar.
- **Donanım sistem alanı (HSA):** HSA, I/O yapılandırmasını içeren sistem belleğinin ayrılmış bir parçasıdır. SAP'ler tarafından kullanılır. Müşteri tarafından satın alınan belleğin bir parçası olmayan 32 GB'lık sabit bir miktar ayrılmıştır. Bu, planlı ve önceden planlanmış kesintileri ortadan kaldırarak daha fazla yapılandırma esnekliği ve daha yüksek kullanılabilirlik sağlar.
- **Mantıksal bölümler:** Mantıksal bölüm, özünde işletim sistemi düzeyinde tanımlanan mantıksal bir işlemci olan bir sanal makine biçimidir.<sup>3</sup> Her CSS en fazla 16 mantıksal bölümü destekler.

<sup>3</sup> Sanal makine, bilgisayar içinde izole bir bellek bölümünde çalışan bir veya daha fazla uygulama ile birlikte bir işletim sisteminin bir örneğidir. Farklı işletim sistemlerinin aynı bilgisayarda aynı anda çalışmasını sağlar ve uygulamaların birbirine karışmasını öner. Sanal makinelerle ilgili bir tartışma için [STAL12]ye bakınız.

- **Alt kanallar:** Bir alt kanal bir programa mantıksal bir aygit olarak görünür ve bir G/C işlemi gerçekleştirmek için gereken bilgileri içerir. CSS tarafından adreslenebilen her G/C cihazı için bir alt kanal mevcuttur. Bir alt kanal, bir bölüm üzerinde çalışan kanal alt sistemi kodu tarafından bir G/C isteğini kanal alt sistemine iletmek için kullanılır. Mantıksal bölüme tanımlanan her aygit için bir alt kanal atanır. CSS başına 196 bine kadar alt kanal desteklenir.
- **Kanal yolu:** Kanal yolu, bir kanal alt sistemi ile bir veya daha fazla kontrol birimi arasında bir kanal aracılığıyla tek bir arabirimdir. Komutlar ve veriler, G/C isteklerini gerçekleştirmek için bir kanal yolu üzerinden gönderilir. Her CSS 256 adede kadar kanal yoluna sahip olabilir.
- **Kanal:** Kanallar, G/C kontrol birimleri (CU'lar) ile iletişim kuran küçük işlemcilerdir. Bellek ve harici aygitlar arasındaki veri aktarımını yönetirler.

Bu ayrıntılı yapı, ana bilgisayarın çok sayıda I/O cihazını ve iletişim bağlantısını yönetmesini sağlar. Tüm G/C işlemleri uygulama ve sunucu işlemcilerinden alınarak performans artırılır. Kanal alt sistem işlemcileri, çok çeşitli G/C görevlerini yönetebilmelerini ve gelişen gereksinimlere ayak uydurabilmelerini sağlayan genel bir yapılandırmaya sahiptir. Kanal işlemcileri, arayüz oluşturdukları G/C kontrol birimleri için özel olarak programlanmıştır.

### I/O Sistem Organizasyonu

I/O sistem organizasyonunu açıklamak için öncelikle zEnterprise EC12'nin fiziksel düzenini kısaca açıklamamız gerekmektedir. Şekil 7. 20 makinenin su soğutmalı versiyonunun önden görünüşüdür (hava soğutmalı versiyonu da vardır). Sistem aşağıdaki özelliklere sahiptir:

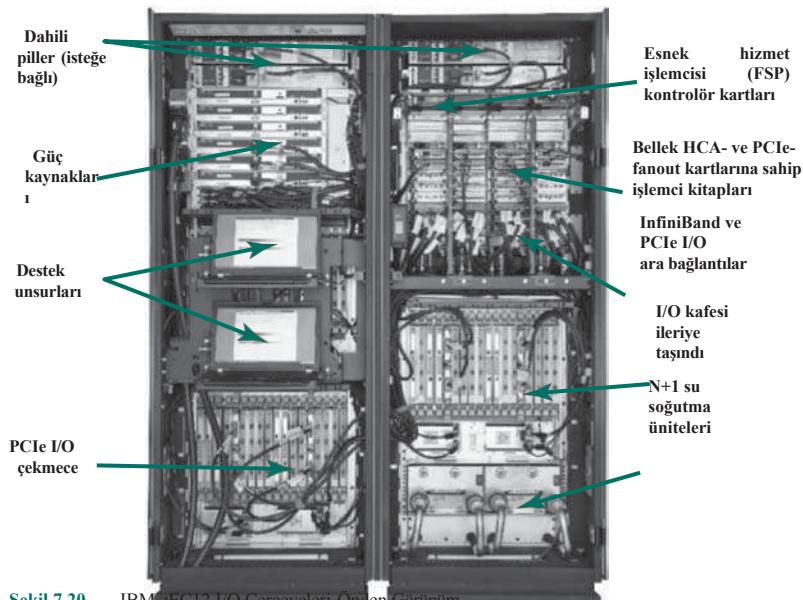
- Ağırlık: 2430 kg (5358 lbs)
- Genişlik: 1,568 m (5,14 ft)
- Derinlik: 1,69 m (6,13 ft)
- Yükseklik: 2,015 m (6,6 ft)

Tam olarak bir dizüstü bilgisayar değildir.

Sistem, zEnterprise EC12'nin çeşitli bileşenlerini barındıran çerçeveye adı verilen iki büyük bölmeden oluşur. Sağ taraftaki A çerçevesi iki büyük kafesin yanı sıra kablolama ve diğer bileşenler için yer içerir. Üst kafes, tamamen birbirine bağlı dört adede kadar işlemci kitabını barındırmak için dört yuvaya sahip bir işlemci kafesiidir. Her kitap bir çoklu çip modülü (MCM), bellek kartları ve G/C kafesi bağlantıları içerir. Her MCM, altı çok çekirdekli yonga ve iki depolama kontrol yongası barındıran bir karttır.

A çerçevesindeki alt kafes, çoklayıcılar ve kanallar da dahil olmak üzere G/C donanımını içeren bir G/C kafesiidir. G/C kafesi, IBM tarafından fabrikada müşteri özelliklerine göre kurulan sabit bir birimdir.

Sol taraftaki Z çerçevesi dahili piller ve güç kaynakları ile platform yönetimi için bir sistem yönetici tarafından kullanılan bir veya daha fazla destek elemanı için yer içerir. Z çerçevesi ayrıca iki veya daha fazla I/O çekmecesi için yuvalar içerir.



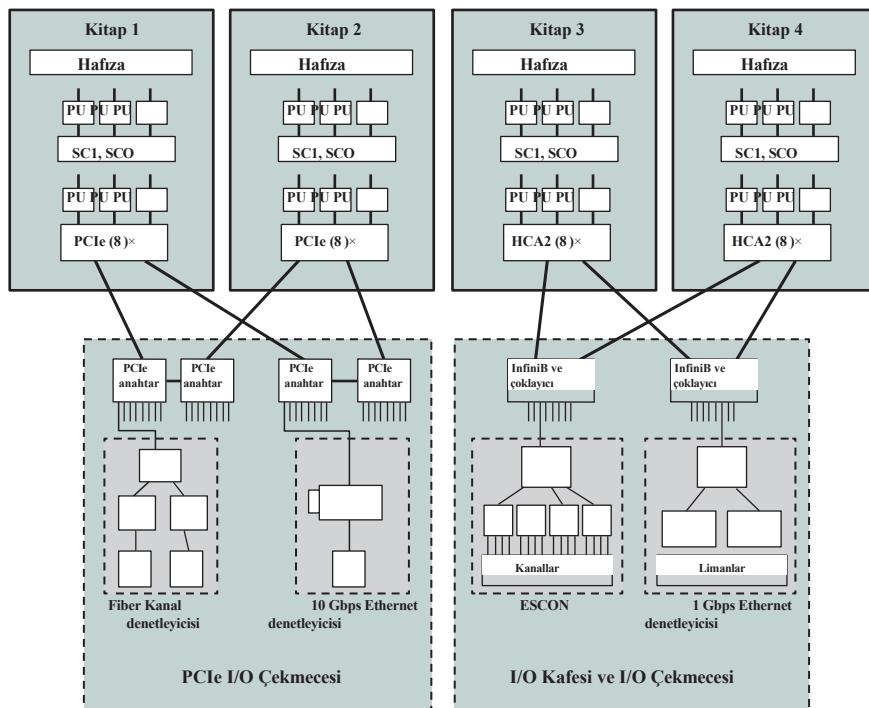
**Şekil 7.20** IBM zEC12 I/O Çerçeveleri-Önden Görünüm

G/C çekmecesi, G/C kafesine benzer bileşenler içerir. Aralarındaki fark, çekmecenin daha küçük olması ve değişen gereksinimleri karşılamak için müşteri sahasında kolayca takılıp çıkarılabilmesidir.

Bu arka planla birlikte, şimdi zEnterprise EC12 I/O sistem yapısının tipik bir yapılandırmasını gösteriyoruz (Şekil 7.21). Her zEC12 işlemci kitabı iki dahili (yani A ve Z çerçevelerine dahili) G/C alt yapısını destekler: G/C kafesleri ve G/C çekmeceleri için InfiniBand ve G/C çekmeceleri için PCI Express (PCIe). Bu kanal denetleyicileri **fanout** olarak adlandırılır.

İşlemci kitabından G/C kafeslerine ve G/C çekmecelerine olan InfiniBand bağlantıları, G/C kafesi veya çekmecesindeki InfiniBand çoklayıcılarına InfiniBand bağlantıları olan bir Ana Bilgisayar Kanal Adaptörü (HCA) çıkışı üzerinden yapılır. InfiniBand çoklayıcılar sunucuları, iletişim altyapısı ekipmanlarını, depolamayı ve gömülü sistemleri birbirine bağlamak için kullanılır. InfiniBand çoklayıcı, tümü InfiniBand kullanan sistemleri birbirine bağlamak için InfiniBand kullanmanın yanı sıra diğer I/O teknolojilerini de destekler. ESCON (Kurumsal Sistem Bağlantısı), tescilli fiber tabanlı bir teknoloji kullanarak disklere, bantlara ve yazıcı cihazlarına bağlantıyi destekler. Ethernet bağlantıları, bu popüler alan ağı teknolojisini destekleyen çeşitli cihazlara 1 Gbps ve 10 Gbps bağlantılar sağlar. Ethernet'in kayda değer bir kullanımı, özellikle blade birbirleriyle ve diğer anabilgisayarlarla birbirine bağlamak için büyük sunucu çiftlikleri oluşturmaktır.<sup>4</sup>

<sup>4</sup> Blade sunucu, tek bir kasada birden fazla sunucu modülünü (blade) barındıran bir sunucu mimarisidir. Yerden tasarruf etmek ve sistem yönetimini iyileştirmek için veri merkezlerinde yaygın olarak kullanılır. Kendi başına duran veya rafa monte edilen kasa, güç kaynağını sağlar ve her blade'in kendi CPU'su, belleği ve sabit diski vardır.



Şekil 7.21 IBM zEC12 I/O Sistem Yapısı

İşlemci kitabından G/C çekmecelerine PCIe bağlantıları, PCIe anahtarlarına bir PCIe çıkışı üzerinden yapılır. PCIe anahtarları bir dizi I/O aygit denetleyicisine bağlanabilir. zEnterprise EC12 için tipik örnekler 1-Gbps ve 10-Gbps Ethernet ve Fiber Kanalıdır.

Her kitap 8 adede kadar InfiniBand HCA ve PCIe çıkışının bir kombinasyonunu içerir. Her bir fanout, her bir bağlantı bir kanal işlemcisi tarafından kontrol edilen işlemci kitabı başına toplam maksimum 256 bağlantı için 32 adede kadar bağlantı destekler.

## 7.10 ÖNERİLER, DEĞERLENDİRME SORULARI VE ÖRNEKLER

### Anahtar Terimler

|                                                                                                                                                                                         |                                                                                                                                                             |                                                                                                                                                  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| önbellek enjeksiyonu<br>döngü çalma<br>doğrudan önbellek erişimi (DCA)<br>Doğrudan Veri I/O<br>doğrudan bellek erişimi (DMA)<br>InfiniBand<br>kesmek<br>kesme güdümlü I/O<br>I/O kanalı | G/C komutu<br>I/O modülü<br>G/C işlemcisi<br>izole I/O<br>son seviye önbellek<br>bellek eşlemeli G/C<br>çoklayıcı kanal<br>yazılmayan tahsis<br>paralel I/O | çevresel aygit<br>programlanmış I/O<br>seçici kanal<br>Seri I/O<br>Thunderbolt<br>yazma ayırma<br>geri yaz<br>aracılığıyla yaz<br>güncelleme yaz |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|

## İnceleme Soruları

- 7.1** Harici veya çevresel cihazların üç geniş sınıflandırmasını listeleyin.
- 7.2** Uluslararası Referans Alfabesi nedir?
- 7.3** Bir I/O modülünün başlıca işlevleri nelerdir?
- 7.4** G/C gerçekleştirmek için üç teknigi listeleyin ve kısaca tanımlayın.
- 7.5** Bellek eşlemeli G/C ile yalıtılmış G/C arasındaki fark nedir?
- 7.6** Bir aygit kesmesi oluştuğunda, işlemci kesmeyi hangi aygitin verdiğini nasıl belirler?
- 7.7** Bir DMA modülü bir veri yolunun kontrolünü ele geçirdiğinde ve veri yolunun kontrolünü elinde tutarken işlemci ne yapar?

## Problemler

- 7.1** Tipik bir mikroişlemcide, G/C veri kayıtlarına atıfta bulunmak için ayrı bir G/C adresi ve belirli bir aygit için G/C denetleyicisindeki kontrol ve durum kayıtları için ayrı bir adres kullanılır. Bu tür kayıtlar **port** olarak adlandırılır. Intel 8088'de iki I/O komut biçimi kullanılır. Bir formatta, 8 bitlik işlem kodu bir G/C işlemini belirtir; bunu 8 bitlik bir port adresi izler. Diğer G/C işlem kodları, bağlantı noktası adresinin 16 bitlik DX kaydında olduğunu gösterir. 8088 her bir G/C adresleme modunda kaç bağlantı noktasını adresleyebilir?
- 7.2** Zilog Z8000 mikroişlemci ailesinde de benzer bir komut formatı kullanılmaktadır. Bu durumda, 16 bitlik port adresinin komutun bir parçası olduğu doğrudan port adresleme özelliği ve komutun port adresini içeren 16 bitlik genel amaçlı kayıtlardan birini referans aldığı dolaylı port adresleme özelliği vardır. Z8000 her bir G/C adresleme modunda kaç port adresleyebilir?
- 7.3** Z8000 ayrıca DMA'dan farklı olarak işlemcinin doğrudan kontrolü altında olan bir blok I/O aktarım özelliğine sahiptir. Blok transfer talimatları bir port adres kaydı (Rp), bir sayımlık kaydı (Rc) ve bir hedef kaydı (Rd) belirter. Rd, giriş portundan okunan ilk baytin saklanacağı ana bellek adresini içerir. Rc, 16 bitlik genel amaçlı yazmaçlarından herhangi biridir. Ne kadar büyütülükte bir veri bloğu aktarılabilir?
- 7.4** Z8000'de bulunan gibi bir blok G/C aktarım komutuna sahip bir mikroişlemci düşünün. İlk yürütülmesinin ardından, böyle bir komutun yeniden yürütülmesi beş saat döngüsü sürer. Ancak, bloklamayan bir G/C komutu kullanırsak, bu komutun alınması ve yürütülmesi toplam 20 saat döngüsü sürer. Blok G/C komutu ile 128 baytlık bloklar aktarılırken hızdaki artışı hesaplayın.
- 7.5** Bir sistem 8 bitlik bir mikroişlemciye dayanmaktadır ve iki G/C aygitine sahiptir. Bu sistem için G/C kontrolörleri ayrı kontrol ve durum kayıtları kullanılır. Her iki aygit da verileri bir seferde 1 bayt olarak işler. İlk cihazın iki durum hattı ve üç kontrol hattı vardır. İkinci aygit üç durum hattına ve dört kontrol hattına sahiptir.
- Her bir cihazın durumunu okumak ve kontrol etmek için kaç tane 8 bit I/O kontrol modülü kaydına ihtiyacımız var?
  - İlk aygit yalnızca çıkış aygıtı olduğu düşünüldüğünde, gerekli kontrol modülü kayıtlarının toplam sayısı nedir?
  - İki cihazı kontrol etmek için kaç farklı adrese ihtiyaç vardır?
- 7.6** Programlanmış G/C için Şekil 7.5, işlemcinin bir G/C cihazının durum kontrolünü yaparken bir bekleme döngüsüne takıldığını göstermektedir. Verimliliği artırmak için G/C yazılımı, işlemcinin cihazın durumunu periyodik olarak kontrol edeceğini şekilde yazılabılır. Eğer cihaz hazır değilse, işlemci diğer görevlere atlayabilir. Belirli bir zaman aralığından sonra, işlemci durumu tekrar kontrol etmek için geri gelir.
- Saniyede 10 karakter (cps) hızında çalışan bir yazıcıya her seferinde bir karakter veri çıkışı için yukarıdaki şemayı düşünün. Durumu her 200 ms'de bir taranırsa ne olur?

- b.** Daha sonra tek bir karakter tamponuna sahip bir klavye düşünün. Ortalama olarak, karakterler 10 cps hızında girilir. Bununla birlikte, iki ardışık tuşa basma arasındaki zaman aralığı 60 ms kadar kısa olabilir. Tuş panosu I/O programı tarafından hangi sıklıkta taramalıdır?
- 7.7** Bir mikroiçlemci her 20 ms'de bir çıkış I/O cihazının durumunu tarar. Bu, işlemciyi her 20 ms'de bir uyaran bir zamanlayıcı aracılığıyla gerçekleştirir. Cihazın arayüzü iki port içermektedir: biri durum için diğer veri çıkışı için. Saat hızının 8 MHz olduğu düşünülürse, cihazı taramak ve servis vermek ne kadar sürer? Basit olması için ilgili tüm komut 12 saat döngüsü süredüğünü varsayıñ.
- 7.8** Bölüm 7.3'te, izole G/Ç ile karşılaşıldığında bellek eşlemeli G/Ç'nin bir avantajı ve bir dezavantajı listelenmiştir. İki avantaj ve iki dezavantaj daha listeleyiniz.
- 7.9** Belirli bir sistem, bir operatör tarafından girdiyen komutlar aracılığıyla kontrol edilir. Sekiz saatlik bir aralıktaki girdiler ortalama komut sayısı 60'tır.
- İşlemcinin klavyeyi her 100 ms'de bir taradığını varsayılmıñ. Klavye 8 saatlik bir süre içinde kaç kez kontrol edilecektir?
  - Kesme güdümlü G/Ç kullanılsaydı işlemcinin klavyeye yaptığı ziyaret sayısı ne kadar azalırıñ?
- 7.10** Şekil 7.9'da gösterilen 8255A'nın şu şekilde yapılandırıldıñını varsayılmıñ: A portu giriş, B portu çıkış ve C portunun tüm bitleri çıkış olarak. Bu yapılandırmayı tanımlamak için kontrol regülatörünün bitlerini gösterin.
- 7.11** Sürekli olarak ortalama 8 KB/s hızında veri aktaran belirli bir cihaz için kesme güdümlü G/Ç kullanan bir sistem düşünün.
- Kesme işleminin yaklaşık  $100 \mu s$  süredüğünü varsayılmıñ (yani, kesme hizmet rutinine (ISR) atlama, onu yürütme ve ana programa dönme süresi). Her bayt için kesme yapıyorsa, bu G/Ç aygitının işlemci zamanının ne kadarını tükettiñini belirleyin.
  - Şimdi aygitin iki adet 16 baytlık tamponu olduğunu ve tamponlardan biri doldugunda işlemin kesildiğini varsayılmıñ. Doğal olarak, kesme işlemi daha uzun süreler, çünkü ISR'nin 16 bayt aktarması gereklidir. ISR yürütülürken işlemci her baytin aktarımı için yaklaşık  $8 \mu s$  süreler. Bu durumda bu G/Ç aygit tarafından işlem süresinin ne kadarının tükettiñini belirleyin.
  - Şimdi işlemcinin Z8000'de olduğu gibi bir blok transfer G/Ç komutu ile donatıldığını varsayılmıñ. Bu, ilgili ISR'nin bir bloğun her baytını yalnızca  $2 \mu s$  içinde aktarmasına izin verir. Bu durumda işlemci zamanının ne kadarlık bir kısmının bu G/Ç cihazı tarafından kullanıldığını belirleyin.
- 7.12** DMA modülleri içeren hemen hemen tüm sistemlerde, ana belleğe DMA'ya CPU'nun ana belleğe erişiminden daha yüksek öncelik verilir. Neden?
- 7.13** Bir DMA modülü, 9600 bps hızında iletişim yapan bir cihazdan **döngü çalma** yöntemini kullanarak karakterleri belleğe aktarmaktadır. İşlemci saniyede 1 milyon komut (1 MIPS) hızında komut getirmektedir. DMA faaliyeti nedeniyle işlemci ne kadar yavaşlayacaktır?
- 7.14** Veri yolu döngülerinin 500 ns süredüğü bir sistem düşünün. Veri yolu kontrolünün işlemciden G/Ç cihazına veya tersi yönde aktarılması 250 ns süreler. G/Ç aygitlarından biri 50 KB/s veri aktarım hızına sahip ve DMA kullanıyor. Veriler her seferinde 1 bayt aktarılır.
- DMA'yı burst modunda kullandığımızı varsayılmıñ. , DMA arayüzü bir blok aktarımı başladığında önce veri yolu hakimiyetini kazanır ve tüm blok aktarılana kadar veri yolu kontrolünü elinde tutar. Cihaz 128 baytlık bir bloğu aktarırken veriyolunu ne kadar süreyle bağlar?
  - Çevrim çalışma modu için hesaplamayı tekrarlayın.
- 7.15** 8237A'nın zamanlama diyagramının incelenmesi, bir blok aktarımı başladığında, DMA döngüsü başına üç veri yolu saat döngüsü süredüğünü gösterir. DMA döngüsü sırasında, 8237A bellek ve G/Ç cihazı arasında bir bayt bilgi aktarır.
- 8237A'yı 5 MHz hızında saatlediğimizi varsayılmıñ. Bir baytin aktarılması ne kadar süreler?

- b.** Ulaşabilecek maksimum veri aktarım hızı ne olabilir?
- c.** Belleğin yeterince hızlı olmadığını ve DMA döngüsü başına iki bekleme durumu eklememiz gerektiğini varsayılm. Gerçek veri aktarım hızı ne olacaktır?
- 7.16** Önceki problemdeki sistemde bir bellek döngüsünün 750 ns süregünü varsayılm. Elde edilen veri aktarım hızını etkilemeden veri yolunu saat hızını hangi değere düşürebiliriz?
- 7.17** Bir DMA denetleyicisi, her biri 64 Kbps hızda sahip dört adet sadece alıcı telekomünikasyon bağlantısına (DMA kanalı başına bir adet) hizmet vermektedir.
- a.** Kontrol ünitesini seri modda mı yoksa döngü çalma modunda mı çalıştırırsınız?
- b.** DMA kanallarının hizmeti için hangi öncelik planınızı kullanırdınız?
- 7.18** 32 bitlik bir bilgisayarın iki seçici kanalı ve bir çoklayıcı kanalı vardır. Her seçici kanal iki manyetik disk ve iki manyetik bant ünitesini destekler. Çoklayıcı kanala bağlı iki satır yazıcı, iki kart okuyucu ve 10 VDT terminali vardır. Aşağıdaki aktarım hızlarını varsayınez:
- |                        |             |
|------------------------|-------------|
| Disk sürücüsü          | 800 Kbyte/s |
| Manyetik bant sürücüsü | 200 Kbyte/s |
| Hat yazıcısı           | 6,6 Kbyte/s |
| Kart okuyucu           | 1,2 Kbyte/s |
| VDT                    | 1 Kbyte/s   |
- Bu sistemdeki maksimum toplam G/C aktarım hızını tahmin edin.
- 7.19** Bir bilgisayar, bir işlemci ve ana bellek M'ye bir kelime veri yolu genişliğine sahip ortak bir veri yolu üzerinden bağlı bir G/C aygıtından (D) oluşur. İşlemci saniyede en fazla  $10^{10}$  komut çalıştırılabilir. Ortalama bir komut, üçü bellek veriyolunu kullanan beş makine çevrimi gerektirir. Bir bellek okuma veya yazma işlemi bir makine çevrimi kullanılır. İşlemcinin sürekli olarak komut yürütme hızının %95'ini gerektiren ancak herhangi bir G/C komutu gerektirmeyen "arka plan" programları yürütüğünü varsayılm. Bir işlemci çevriminin bir veri yolu çevrimine eşit olduğunu varsayıyın. Şimdi G/C cihazının M ve D arasında çok büyük veri bloklarını aktarmak için kullanılacağını varsayılm.
- a.** Programlanmış G/C kullanılıyorsa ve her bir kelimeyi G/C aktarımı işlemecinin iki komut yürütmesini gerektiriyorsa, D aracılığıyla mümkün olan maksimum G/C veri aktarım hızını saniye başına kelime cinsinden tahmin edin.
- b.** DMA kullanılıyorsa aynı oranı tahmin edin.
- 7.20** Bir veri kaynağı, her birine bir eşlik biti eklenmiş 7 bitlik IRA karakterleri üretir. Aşağıdakiler için bir  $R_{bps}$  hattı üzerinden maksimum etkin veri hızı (IRA veri bitlerinin hızı) için bir ifade türetilin:
- a.** Asenkron iletim, 1,5 birim durdurma biti ile;
- b.** Bit-senkron iletim, 48 kontrol biti ve 128 bilgi bitinden oluşan bir çerçeve ile;
- c.** 1024-bit bilgi alanı ile (b) ile aynıdır;
- d.** Karakter eşzamanlıçerçeve başına dokuz kontrol karakteri ve 16 bilgi karakteri;
- e.** (d) ile aynı, 128 bilgi karakteri ile.
- 7.21** İki kadın yüksek bir çitin iki tarafındadır. Kadınlardan birinin adı , çitin kendi tarafında yetişen lezzetli elmlarla dolu güzel bir elma ağacı var; ihtiyaç duyduğunda diğer kadına elma tedarik etmekte mutluluk duyuyor. Elma yiyan adındaki diğer kadın ise elma yemeyi çok seviyor ama hiç elması yok. Aslında, elmlarını sabit bir oranda yemesi gerekiyor (günde bir elma doktoru uzak tutar). Eğer bu orandan daha hızlı yerse hastalanacaktır. Daha yavaş yerse yetersiz maruz kalacaktır. Her iki kadın da konuşamaz ve bu yüzden sorun, elmları elma sunucusundan elma yiyeceği doğru oranda ulaşmaktadır.
- a.** Çitin üzerinde bir çalar saat olduğunu ve saatin birden fazla alarm ayarına sahip olabileceğini varsayılm. Problemi çözmek için saat nasıl kullanılabilir? Çözümü göstermek için bir zamanlama diyagramı çizin.

- b.** Şimdi çalar saat olmadığını varsayıyalım. Bunun yerine Elma Yiyen'in bir elmaya ihtiyacı olduğunda sallayabileceği bir bayrağı var. Yeni bir çözüm önerin. Elma-sunucunun da bir bayrağı olması yardımcı olur mu? Eğer öyleyse, bunu da çözüme dahil edin. Bu yaklaşımın sakincalarını tartışın.
- c.** Şimdi bayrağı kaldırın ve uzun bir ip parçasının var olduğunu varsayıyın. İpi kullanarak (b)'dekinden daha üstün bir çözüm öneriniz.

**722** Bir 16-bit ve iki 8-bit mikroişlemcisinin bir sistem veriyoluna arayüzleneceğini varsayıyın. Aşağıdaki detaylar verilmiştir:

- 1.** Tüm mikroişlemciler her tür veri aktarımı için gerekli donanım özelliklerine sahiptir: programlanmış G/C, kesme güdümlü G/C ve DMA.
- 2.** Tüm mikroişlemcilerin 16 bitlik bir adres veriyolu vardır.
- 3.** Her biri 64 Kbyte kapasiteli iki bellek kartı veriyolu ile arayüzlenmiştir. Tasarımcı mümkün olduğunda büyük bir paylaşılan bellek kullanmak istemektedir.
- 4.** Sistem veriyolu en fazla dört kesme hattını ve bir DMA hattını destekler. Gerekli diğer varsayımları yapın ve:
  - a.** Sistem veri yolu özelliklerini hat sayısı ve türleri açısından veriniz.
  - b.** Veri yolu üzerinde iletişim kurmak için olası bir protokolü tanımlayın (örn. okuma-yazma, kesme ve DMA dizileri).
  - c.** Yukarıda bahsedilen cihazların sistem nasıl bağlandığını açıklayınız.



## BÖLÜM

# 8

## OPERATING SİSTEM DESTEĞİ

### 8.1 İşletim Sistemine Genel Bakış

İşletim Sistemi Amaçları ve İşlevleri İşletim Sistemi Türleri

### 8.2 Programlama

Uzun Vadeli Çizelgeleme Orta  
Vadeli Çizelgeleme Kısa Vadeli  
Çizelgeleme

### 8.3 Bellek Yönetimi

Bölümlemeyi  
Değiştirme Disk  
belleği  
Sanal Bellek  
Çeviri Lookaside Arabellek  
Segmentasyonu

### 8.4 Intel x86 Bellek Yönetimi Adres

Alanları Segmentasyonu  
Çağrı

### 8.5 ARM Bellek Yönetimi

Bellek Sistemi Organizasyonu Sanal Bellek  
Adres Çevirisi Bellek Yönetimi Formatları  
Erişim Kontrolü

### 8.6 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler

**ÖĞRENİM HEDEFLERİ**

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- Bir **işletim sisteminin (OS)** temel işlevlerini en üst düzeyde özetleyiniz.
- İlk basit toplu sistemlerden modern karmaşık sistemlere kadar işletim sistemlerinin evrimini tartışmak.
- Uzun, orta ve kısa vadeli çizelgeleme arasındaki farkları açıklayın.
- Bellek **bölümlemenin** nedenini anlamak ve kullanılan çeşitli teknikleri açıklamak.
- Sayfalama ve segmentasyonun göreceli avantajlarını değerlendirin.
- Sanal belleği tanımlayın.

Bu metnin odak noktası bilgisayar donanımı olmasına rağmen, ele alınması gereken bir yazılım alanı vardır: bilgisayarın işletim sistemi. İşletim sistemi, bilgisayarın kaynaklarını yöneten, programcılar için hizmetler sağlayan ve diğer programların çalıştırılmasını planlayan bir programdır. CPU'nun bilgisayar sistemini kontrol ettiği mekanizmaları anlamak için işletim sistemlerini biraz anlamak gereklidir. Özellikle, kesintilerin etkisi ve bellek hiyerarşisinin yönetimine ilişkin açıklamalar en iyi bu bağlamda açıklanabilir.

Bu bölüm işletim sistemlerine genel bir bakış ve kısa bir tarihçe ile başlamaktadır. Bölümün büyük bir kısmı, bilgisayar organizasyonu ve mimarisi çalışmalarıyla en çok ilgili olan iki işletim sistemi işlevini incelemektedir: zamanlama ve bellek yönetimi.

## 8.1 SİSTEMİN İŞLETİMİ

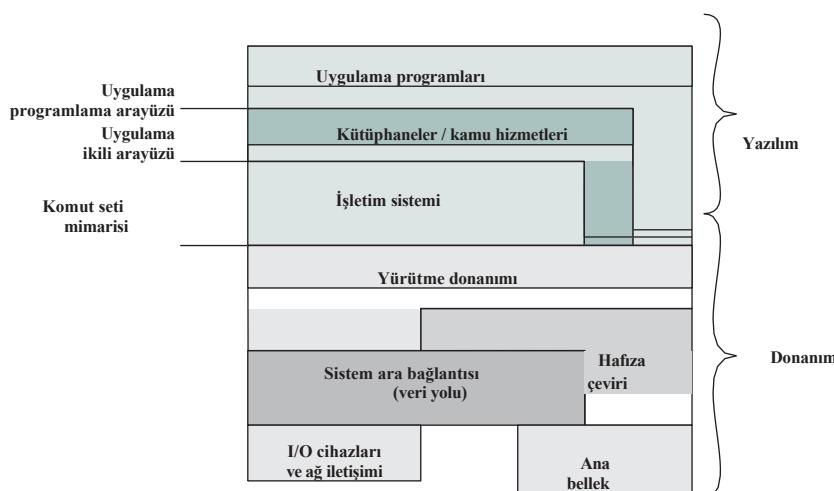
### İşletim Sistemi Amaçları ve İşlevleri

İşletim sistemi, uygulama programlarının yürütülmesini kontrol eden ve uygulamalar ile bilgisayar donanımı arasında bir arayüz gören bir programdır. İki amacı olduğu

- **Kolaylık:** Bir işletim sistemi bilgisayarı daha hale getirir.
- **Verimlilik:** Bir işletim sistemi, bilgisayar sistem kaynaklarının verimli bir şekilde kullanılmasını sağlar.

Bir işletim sisteminin bu iki yönünü sırayla inceleyelim.

**KULLANICI/ÇEVİRİCİ ARAYÜZÜ OLARAK İŞLETİM SİSTEMİ** Donanım ve bir kullanıcıya uygulama sağladıkça kullanılan yazılım, Şekil 8.1'de gösterildiği gibi katmanlı veya hiyerarşik bir şekilde görülebilir. Bu uygulamaların kullanıcısı olan son kullanıcı genellikle bilgisayarın mimarisi ile ilgilenmez. Bu nedenle son kullanıcı bir bilgisayar sistemini bir uygulama açısından görür. Bu uygulama bir programlama dilinde ifade edilebilir ve bir uygulama programcısı tarafından geliştirilir. Bir uygulama programını bir dizi işlemci talimatı olarak geliştirmek için



Şekil 8.1 Bilgisayar Donanım ve Yazılım Yapısı

Bilgisayar donanımını kontrol etmekten tamamen sorumlu olan bir sistem kurmak son derece karmaşık bir görev olacaktır. Bu görevi kolaylaştırmak için bir dizi sistem programı sağlanmıştır. Bu programlardır bazıları **yardımcı** programlar olarak adlandırılır. Bunlar, program oluşturma, dosya yönetimi ve G/C aygıtlarının kontrolüne yardımcı olan sık kullanılan işlevleri yerine getirir. Bir programcı bir uygulama geliştirdiğinde bu olsanlardan yararlanır ve uygulama çalışırken belirli işlevleri yerine getirmek için yardımcı programları çağırır. En önemli sistem programı işletim sistemidir. İşletim sistemi, donanımın ayrıntılarını programcının gizler ve programcuya sistemi kullanması için uygun bir arayüz sağlar. Programcının ve uygulama programlarının bu olsanlara ve hizmetlere erişmesini ve bunları kullanmasını kolaylaştırarak arabulucu görür.

Kısaca, işletim sistemi tipik olarak aşağıdaki alanlarda hizmet vermektedir:

- **Program oluşturma:** İşletim sistemi, programcuya program oluşturmada yardımcı olmak için editörler ve hata ayıklayıcılar gibi çeşitli olanaklar ve hizmetler sağlar. Tipik olarak, bu hizmetler aslında işletim sisteminin bir parçası olmayan ancak işletim sistemi aracılığıyla erişilebilen yardımcı programlar şeklindedir.
- **Program yürütme:** Bir programı yürütmek için bir dizi adımın gerçekleştirilmesi gereklidir. Talimatlar ve veriler ana belleğe yüklemeli, G/C aygıtları ve dosyalar başlatılmalı ve diğer kaynaklar hazırlanmalıdır. İşletim sistemi tüm bunları kullanıcı için halleder.
- **I/O cihazlarına erişim:** Her G/C cihazı, çalışması için kendine özgü bir dizi talimat veya kontrol sinyali gerektirir. İşletim sistemi ayrıntılarla ilgilenir, programcı basit okuma ve yazmalar açısından düşünebilir.
- **Dosyalara kontrollü erişim:** Dosyalar söz konusu olduğunda, kontrol yalnızca G/C aygıtlının (disk sürücüsü, teyp sürücüsü) doğasının değil, aynı zamanda depolama ortamındaki dosya formatının da anlaşılmasını içermelidir. Yine, işletim sistemi ayrıntılar hakkında endişelenir. Ayrıca, birden fazla eşzamanlı kullanıcısı olan bir sistem konusu olduğunda, işletim sistemi dosyalara erişimi kontrol etmek için koruma mekanizmaları sağlayabilir.

- **Sistem erişimi:** Paylaşılan veya genel bir sistem söz konusu olduğunda, işletim sistemi bir bütün olarak sisteme ve belirli sistem kaynaklarına erişimi kontrol eder. Erişim işlevi, kaynakların ve verilerin yetkisiz kullanıcılarından korunmasını sağlamalı ve kaynak çekişmesi için çalışmaları çözmeliidir.
- **Hata tespiti ve yanıt:** Bir bilgisayar sistemi çalışırken çeşitli hatalar meydana gelebilir. Bunlar arasında bellek hatası ya da aygit arızası gibi dahili ve harici donanım hataları ve aritmetik taşıma, yasak bellek konumuna erişim girişimi ve işletim sisteminin bir uygulamanın isteğini yerine getirememesi gibi çeşitli yazılım hataları yer alır. Her durumda, işletim sistemi çalışan uygulamalar üzerinde en az etkiyle hata durumunu ortadan kaldıracak yanıt vermelidir. Yanıt, hataya neden olan programı sonlandırmaktan, işlemi yeniden denemeye ve hatayı uygulamaya bildirmeye kadar değişebilir.
- **Muhasebe:** İyi bir işletim sistemi çeşitli kaynaklar için kullanım istatistikleri toplar ve yanıt süresi gibi performans parametrelerini izler. Herhangi bir sistemde, bu bilgiler gelecekteki geliştirmelere olan ihtiyacı tahmin ve performansı artırmak için sistemi ayarlamada faydalıdır. Çok kullanıcılı bir sistemde bu bilgiler faturalandırma amacıyla kullanılabilir. Şekil 8.1 ayrıca tipik bir bilgisayar sistemindeki üç temel arayüzü de göstermektedir:
  - **Komut kümesi mimarisı (ISA):** ISA, bir bilgisayarın takip edebileceği makine dili talimatları repertuarını tanımlar. Bu arayüz, donanım ve yazılım arasındaki sınırları. Hem uygulama programlarının hem de yardımcı programların ISA'ya doğrudan erişebileceğini unutmayın. Bu programlar için komut repertuarının bir altkümesi kullanılabilir (kullanıcı ISA'sı). İşletim sistemi, sistem kaynaklarının yönetimiyle ilgilenen ek makine dili talimatlarına erişebilir (sistem ISA'sı).
  - **Uygulama ikili arayüzü (ABI):** ABI, programlar arasında taşınabilirliği için bir standart tanımlar. ABI, işletim sisteme sistem çağrıları ara yüzünü ve kullanıcı ISA'sı aracılığıyla bir sistemde mevcut olan donanım kaynaklarını ve hizmetlerini tanımlar.
  - **Uygulama programlama arayüzü (API):** API, bir programa **yüksek seviyeli dil (HLL)** kütüphane çağrıları ile desteklenen kullanıcı ISA'sı aracılığıyla bir sistemde bulunan donanım kaynaklarına ve hizmetlerine erişim sağlar. Tüm sistem çağrıları genellikle kütüphaneler aracılığıyla gerçekleştirilir. Bir API kullanmak, uygulama yazılımının yeniden derleme yoluyla aynı API'yi destekleyen diğer sistemlere kolayca taşınmasını sağlar.

**KAYNAK YÖNETİCİSİ OLARAK İŞLETİM SİSTEMİ** Bilgisayar, verilerin taşınması, depve işlenmesi ve bu işlevlerin kontrolü için bir dizi kaynaktır. İşletim sistemi bu kaynakların yönetiminden sorumludur.

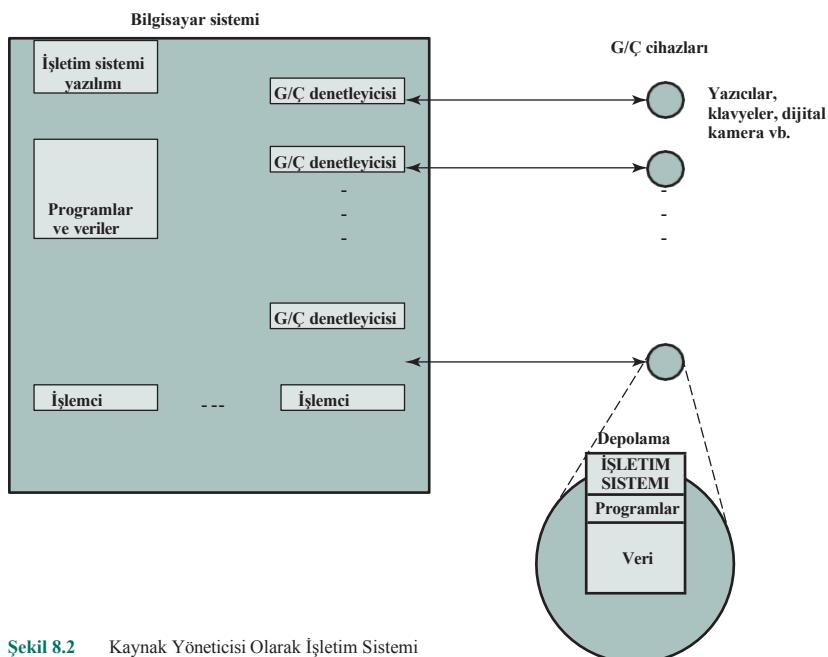
İşletim sisteminin verilerin hareketini, depolanmasını ve işlenmesini kontrol ettiğini söyleyebilir miyiz? Bir bakış açısına göre cevap evet: İşletim sistemi bilgisayarın kaynaklarını yöneterek bilgisayarın temel işlevlerini kontrol eder. Ancak bu kontrol ilginç bir şekilde uygulanmaktadır. Normalde, bir kontrol mekanizmasını kontrol edilen şeyin dışında bir şey olarak ya da en azından edilen şeyin farklı ve ayrı bir parçası olarak düşünürüz. (Örneğin, bir konut ısıtma sistemi

isi üretimi ve isi dağıtımları apartalarından tamamen farklı olan bir termostat tarafından kontrol edilir). Bu durum, kontrol mekanizması olarak iki açıdan sıra dışı olan işletim sistemi için geçerli değildir:

- İşletim sistemi sırada bilgisayar yazılımlarıyla aynı şekilde çalışır; yani işlemci tarafından yürütülen bir programdır.
- İşletim sistemi sık sık kontrolü bırakır ve kontrolü yeniden kazanması için işlemciye bağlı olması gereklidir.

Diğer bilgisayar programları gibi, işletim sistemi de işlemler için talimatlar sağlar. Temel fark programın amacındadır. İşletim sistemi, işlemciyi diğer sistem kaynaklarının kullanımında ve diğer programların yürütülme zamanlamasında yönlendirir. Ancak işlemcinin bunlardan herhangi birini yapabilmesi için işletim sistemi programını yürütmemeyi bırakması ve diğer programları yürütmesi gereklidir. Böylece, işletim sistemi işlemciyi bazı "faydalı" işler yapması için kontrolü bırakır ve daha sonra işlemciyi bir sonraki iş yapmaya hazırlayacak kadar uzun süre kontrolü sürdürür. Tüm bunlarla ilgili mekanizmalar bölüm ilerledikçe netleşecektir.

Şekil 8.2 işletim sistemi tarafından yönetilen ana kaynakları göstermektedir. İşletim sisteminin bir kısmı ana bellektedir. Bu, işletim sisteminde en sık kullanılan işlevleri ve belirli bir zamanda işletim sisteminin o anda kullanıldığımda olan diğer bölgelerini içeren **çekirdeği** veya **nükleusu** içerir. Ana belleğin geri kalımı kullanıcı programları ve verilerini içerir. Bu kaynağın (ana bellek) tahsis, göreceğimiz gibi, işletim sistemi ve işlemcideki bellek yönetim donanımı tarafından ortaklaşa kontrol edilir. İşletim sistemi, bir G/C aygıtlarının yürütülmekte olan bir program tarafından ne zaman kullanılabilirliğine karar verir ve bu aygıta erişimi kontrol eder.



Şekil 8.2 Kaynak Yöneticisi Olarak İşletim Sistemi

dosyaların kullanımı. İşlemcinin kendisi bir kaynaktır ve işletim sistemi belirli bir kullanıcı programının yürütülmesine ne kadar işlemci zamanı ayrılmışlığını belirlemelidir. Çok işlemeli bir sistem söz konusu olduğunda, bu karar tüm işlemcileri kapsamalıdır.

## İşletim Sistemi Türleri

Bazı temel özellikler çeşitli işletim sistemlerini birbirinden ayırmaya yarar. Bu özellikler iki bağımsız boyutta ele alınmaktadır. Birinci boyut, sistemin toplu mu yoksa etkileşimli mi olduğunu belirtir. **Etkileşimli** bir sistemde, kullanıcı/programcı bir iGin yürütülmüşünü talep etmek veya bir iGlem gerçekleĢtirmek için genellikle bir klavye/ekran terminali aracılığıyla bilgisayarla doğrudan etkileĢime girer. Ayrıca kullanıcı, uygulamanın niteliğine bağlı olarak, işin yürütülmesi sırasında bilgisayarla iletişim kurabilir. **Toplu iş sistemi**, etkileşimli sistemin tam tersidir. Kullanıcının programı diğer kullanıcıların programlarıyla birlikte grupperlendirilir ve bir bilgisayar operatörü tarafından gönderilir. Program tamamlandıktan sonra sonuçlar kullanıcı için yazdırılır. Saf toplu iş sistemleri günümüzde nadirdir, ancak toplu iş sistemlerini kısaca incelemek çağdaş işletim sistemlerinin tanımı için faydalı olacaktır.

Bağımsız bir boyut, sistemin çoklu **programlama** kullanıp kullanmadığını belirtir. Çoklu programlama ile, işlemcinin aynı anda birden fazla program üzerinde çalışmasını sağlayarak mümkün olduğunda meşgul tutmaya çalışılır. Belleğe birkaç program yüklenir ve işlemci bunlar arasında hızla geçiş yapar. Bunun alternatifisi, **aynı** anda yalnızca bir program üzerinde çalışan tek **programlama** sistemidir.

**ERKEN SİSTEMLER** 1940'ların sonlarından 1950'lerin ortalarına kadar ilk bilgisayarlarda programcı doğrudan bilgisayar donanımıyla etkileşime giriyyordu; işletim sistemi yoktu. Bu işlemciler, ekran ışıkları, geçiş anahtarları, bir çeşit giriş cihazı ve bir yazıcıdan oluşan bir konsoldan çalıştırılıyordu. İşlemci kodundaki programlar giriş aygıtı (örneğin bir kart okuyucu) aracılığıyla yüklenirdi. Eğer bir hata programı durdurursa, hata durumu ışıklar tarafından gösterilirdi. Programcı hatanın nedenini belirlemek için kayıtları ve ana belleği incelemeye devam edebiliyordu. Program normal bir şekilde tamamlanırsa, çıktı yazıcıda göründürdü.

Bu ilk sistemler iki ana sorun ortaya koymuştur:

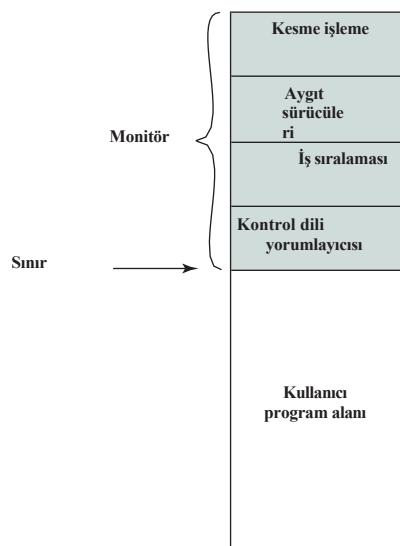
- **Planlama:** Çoğu kurulumda işlemci zamanını rezerve etmek için bir kayıt sayfası kullanıldı. Tipik olarak, bir kullanıcı yarınlık saatin katları şeklinde bir zaman bloğu için kaydolabilirdi. Bir kullanıcı bir saat için kaydolabilir ve 45 dakika içinde bitirebilir; bu bilgisayarın boşta kalmasına neden olur. Öte yandan, kullanıcı sorunlarla karşılaşabilir, ayrılan sürede bitiremeyecek ve sorunu çözmeden önce durmak zorunda kalabilir.
- **Kurulum süresi:** İş olarak adlandırılan tek bir program, derleyiciyi ve üst düzey dil programını (kaynak program) belleğe yüklemeyi, derlenmiş programı (nesne programı) kaydetmeyi ve ardından nesne programını ve ortak işlevleri yüklemeyi ve birbirine bağlamayı içerebilir. Bu adımların her biri kasetlerin takılmasını veya sökülmesini ya da kart destelerinin ayarlanması içerebilir. Eğer bir hata meydana gelirse, talihsiz kullanıcı genellikle kurulum sırasının başına dönmek zorunda kalıyordu. Bu nedenle, sadece programı çalıştırılmak üzere ayarlamak için önemli miktarda zaman harcanyordu.

Bu çalışma şekli, kullanıcıların bilgisayara olarak eriştiği gerçeğini yansıtacak şekilde seri işlemeye olarak adlandırılabilir. Zaman içinde, seri işlemeyi daha verimli hale getirmek için çeşitli sistem yazılım araçları geliştirilmiştir. Bunlar arasında ortak fonksiyon kütüphaneleri, bağlayıcılar, yükleyiciler, hata ayıklayıcılar ve tüm kullanıcılar için ortak yazılım olarak mevcut olan I/O sürücü rutinleri yer almaktadır.

**BASIT BATCH SİSTEMLERİ** İlk işlemciler çok pahalıydı ve bu nedenle işlemci kullanımını en üst düzeye çıkarmak önemlidiydi. Zamanlama ve kurulum süresi nedeniyle bosa harcanan zaman kabul edilemezdi.

Kullanımı iyileştirmek için basit toplu işletim sistemleri geliştirilmiştir. **Monitör** olarak da adlandırılan böyle bir sistemde, kullanıcının artık prosesöre doğrudan erişimi yoktur. Bunun yerine, kullanıcı işi kartlar ya da kaset üzerinde bir bilgisayar operatörüne gönderir, operatör de işleri sırayla bir araya getirir ve tüm partiyi monitör tarafından kullanılmak üzere bir giriş cihazına yerleştirir.

Bu şemanın nasıl çalıştığını anlamak için iki açıdan bakalım: monitörün ve işlemcinin bakış açısından. Monitörün bakış açısından, monitör olayların sırasını kontrol eder. Bunun böyle olabilmesi için, monitörün bir kısmının her zaman ana bellekte ve yürütülmeye hazır olması gereklidir (Şekil 8.3). Bu kısım **yerleşik monitör** olarak adlandırılır. Monitörün geri kalanı, bunları gerektiren herhangi bir işin başında kullanıcı programına alt yordamlar olarak yüklenen yardımcı programlardan ve ortak işlevlerden oluşur. Monitör işleri giriş cihazından (tipik olarak bir kart okuyucu veya manyetik bant sürücüsü) teker teker okur. Okundukça, geçerli iş kullanıcı programı alanına yerleştirilir ve kontrol bu işe aktarılır. İş tamamlandığında, kontrol monitörde geri döner ve monitör hemen bir sonraki işi okur. Her işin sonuçları kullanıcıya teslim üzere yazdırılır.



**Şekil 8.3** Yerleşik Monitör için Bellek Düzeni

Şimdi bu diziyi işlemcinin bakış açısından ele alın. Zamanın belirli bir noktasında, işlemci ana belleğin monitörü içeren bölümündeki talimatları yürütürmektedir. Bu talimatlar bir sonraki işin ana belleğin başka bir bölümüne okunmasına neden olur. Bir iş okunduğunda, işlemci monitörde işlemciye kullanıcı programının başlangıcında yürütümeye devam etme talimatı veren bir dallanma talimatı ile karşılaşacaktır. İşlemci daha sonra bir bitiş veya hata durumuyla karşılaşana kadar kullanıcının programındaki komutu yürütücektir. Her iki olay da işlemcinin bir sonraki komutu monitör programından almasına neden olur. Dolayısıyla "kontrol bir işe geçti" ifadesi basitçe işlemcinin kullanıcı programındaki talimatları alıp yürütüdüğü anlamına gelir ve "kontrol monitöre geri döndü" ifadesi de işlemcinin monitör programından talimatları alıp yürütüdüğü anlamına gelir.

Monitörün çizelgeleme sorununu ele aldığı açık olmalıdır. Bir yoğun iş konur ve işler, araya boş zaman girmeden mümkün olduğunda hızlı bir şekilde yürütülür.

Peki ya iş kurulum süresi? Monitör bunu da halleder. Her işe birlikte, talimatlar bir **iş kontrol diline (JCL)** dahil edilir. Bu, monitöre talimatlar sağlamak için kullanılan özel bir programlama dili türüdür. Basit bir örnek olarak, FORTRAN diliinde yazılmış bir programı ve program tarafından kullanılacak bazı verileri gönderen bir kullanıcı verilebilir. Her FORTRAN talimiği ve her veri ögesi ayrı bir delikli kartta veya teypde ayrı bir kayıttta bulunur. FORTRAN ve veri satırlarına ek olarak, iş başında "\$" ile gösterilen iş kontrol talimatlarını da içerir. İşin genel formatı şu şekildedir:

```
$İŞ
$FTN
f          6 FORTRAN talimatları
$YÜKLE
$RUN
f          6 Veri
$END
```

Bu işi yürütmek için monitör \$FTN satırını okur ve yoğun depolama alanından (genellikle teyp) uygun derleyiciyi yükler. Derleyici kullanıcının programını nesne koduna çevirir ve bu kod bellekte ya da yoğun depolama alanında saklanır. Bellekte saklanıyorsa, işlem "derle, yükle ve git" olarak adlandırılır. Kasette saklanıyorsa, \$LOAD komutu gereklidir. Bu komut, derleme işleminden sonra kontrolü yeniden ele alan monitör tarafından okunur. Monitör, nesne programını derleyicinin yerine belleğe yükleyen ve kontrolü ona devreden yükleyiciyi çağrıır. Bu şekilde, ana belleğin büyük bir bölümü farklı alt sistemler arasında paylaşılabilir, ancak aynı anda yalnızca bir alt sistem yerleşik olabilir ve yürütülebilir.

Monitörün ya da toplu işletim sisteminin basitçe bir bilgisayar programı olduğunu görüyoruz. İşlemcinin ana bilgisayarın çeşitli bölümlerinden talimatları alma yeteneğine dayanır.

kontrolü olarak ele geçirmek ve bırakmak için bellek. Diğer bazı donanım özellikleri de arzu edilir:

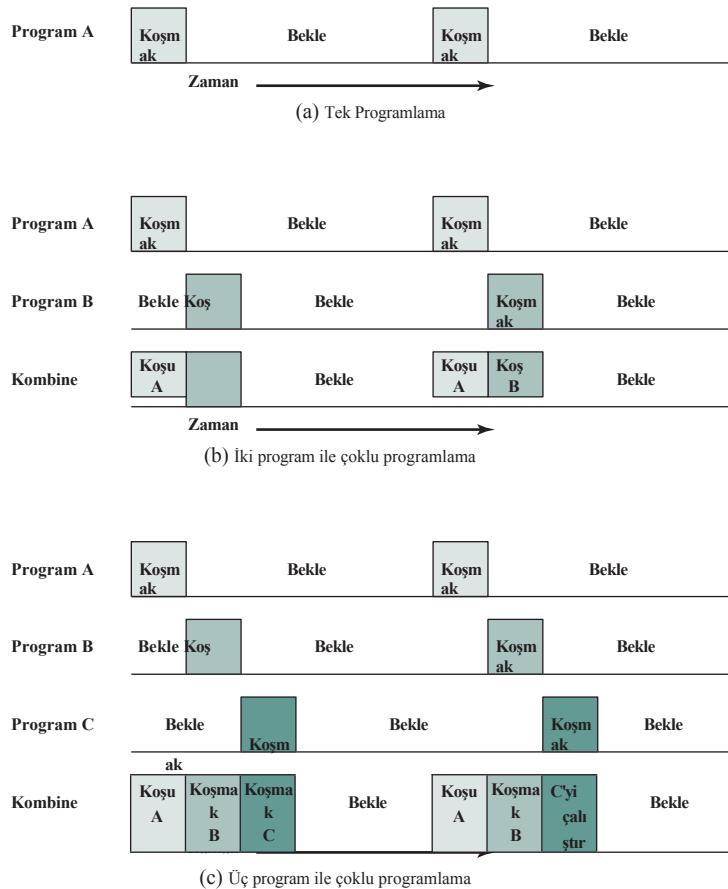
- **Bellek koruması:** Kullanıcı programı yürütülürken, monitörü içeren bellek alanını değiştirmemelidir. Böyle bir girişimde, işlem donanımı bir hata tespit etmeli ve kontrolü monitöre aktarmalıdır. Monitör daha sonra işi iptal eder, bir hata mesajı yazdırır ve bir sonraki işi yükler.
- **Zamanlayıcı:** Tek bir işin sistemi tekeline almasını önlemek için bir zamanlayıcı kullanılır. Zamanlayıcı her işin başında ayarlanır. Zamanlayıcının süresi dolarsa, bir kesme oluşur ve kontrol monitöre döner.
- **Ayrıcalıklı talimatlar:** Bazı talimatlar ayrıcalıklı olarak belirlenmiştir ve yalnızca monitör tarafından yürütülebilir. İşlemci bir kullanıcı programını yürütürken böyle bir talimatla karşılaşırsa, bir hata kesmesi meydana gelir. Ayrıcalıklı talimatlar arasında I/O talimatları da vardır, böylece monitör tüm I/O cihazlarının kontrolünü elinde tutar. Bu, örneğin bir kullanıcı programının bir sonraki işin iş kontrol talimatlarını yanlışlıkla okumasını öner. Bir kullanıcı programı G/C gerçekleştirmek isterse, monitörün kendisi için bu işlemi gerçekleştirmesini talep etmelidir. Bir kullanıcı programı yürütürken işlemci tarafından ayrıcalıklı bir komutla karşılaşılırsa, işlemci donanımı bunu bir hata olarak değerlendirir ve kontrolü monitöre aktarır.
- **Kesmeler:** İlk bilgisayar modellerinde bu özellik yoktu. Bu özellik işletim sistemine kullanıcı programlarına kontrolü bırakma ve onlardan kontrolü geri alma konusunda daha fazla esneklik sağlar.

İşlemci zamanı, kullanıcı programlarının yürütülmesi ile monitörün yürütülmesi arasında değişmektedir. İki fedakarlık söz konusudur: Bir miktar ana bellek artık monitöre verilmiştir ve bir miktar işlemci zamanı monitör tarafından tüketilmektedir. Bunların her ikisi de ek yük oluşturur. Bu ek yük rağmen, basit toplu iş sistemi bilgisayar kullanımını artırır.

**ÇOK PROGRAMLI TOPLU İŞ SİSTEMLERİ** Basit bir toplu iş işletim sistemi tarafından sağlanan otomatik iş sıralamasında bile işlemci genellikle boşta kalır. Sorun, I/O cihazlarının işlemciye kiyasla yavaş olmasıdır. Şekil 8.4 temsili bir hesaplamayı detaylandırmaktadır. Hesaplama, bir kayıt dosyasını işleyen ve kayıt başına ortalama 100 işlemci talimatı gerçekleştiren bir programla ilgilidir. Bu örnekte bilgisayar zamanının %96'sından fazlasını G/C aygıtlarının veri aktarımını bitirmesini bekleyerek geçirmektedir! Şekil 8.5a bu durumu göstermektedir. İşlemci zamanının belli bir kısmını

|                                                       |            |
|-------------------------------------------------------|------------|
| Dosyadan bir kayıt okuma                              | 15 $\mu$ s |
| 100 talimat yürütme                                   | 1 $\mu$ s  |
| Dosyaya bir kayıt yaz                                 | 15 $\mu$ s |
| TOPLAM                                                | 31 $\mu$ s |
| CPU kullanım yüzdesi = $\frac{1}{31} = 0,032 = \%3,2$ |            |

Şekil 8.4 Sistem Kullanım Örneği



Şekil 8.5 Çoklu Programlama Örneği

Bir G/C komutuna ulaşana kadar yürütme süresi. Daha sonra devam etmeden önce bu G/C komutunun tamamlanmasını beklemelidir.

Bu verimsizlik gereklidir. İşletim sistemi (yerleşik monitör) ve bir kullanıcı programını tutmak için yeterli bellek olması gerektiğini biliyoruz. İşletim sistemi ve iki kullanıcı programı için yer olduğunu varsayıyalım. Şimdi, bir işin G/C için beklemesi gerekiyinde, işlemci muhtemelen G/C'yi beklemeyen diğer işe geçebilir (Şekil 8.5b). Ayrıca, belleği üç, dört veya daha fazla programı tutacak şekilde genişletebilir ve hepsi arasında geçiş yapabiliriz (Şekil 8.5c). Bu teknik **çoklu programlama** ya da **çoklu görev** olarak bilinir.<sup>1</sup> Modern işletim sistemlerinin ana temasıdır.

<sup>1</sup> Çoklu görev terimi bazen, birden fazla programdan birden fazla süreci ifade eden çoklu programmanın aksine, işletim sistemi tarafından eş zamanlı olarak ele alınabilen aynı program içindeki birden fazla görevi ifade etmek için ayrılmıştır. Ancak, çoğu standart sözlükte (örneğin, IEEE Std 100-1992, *The New IEEE Standard Dictionary of Electrical and Electronics Terms*) yapıldığı gibi, çoklu görev ve çoklu programlama terimlerini eşitlemek daha yaygındır.

**ÖRNEK 8.1** Bu örnek çoklu programlamanın faydalarını göstermektedir. 250 Mbyte kullanılabilir belleği (işletim sistemi tarafından kullanılmayan), bir diski, bir terminali ve bir yazıcısi olan bir bilgisayar düşünün. Üç program, JOB1, JOB2 ve JOB3, Tablo 8.1'de listelenen özelliklerle aynı anda yürütülmek üzere gönderilir. JOB1 ve JOB2 için minimum işlemci gereksinimi ve JOB3 için sürekli disk ve yazıcı kullanımı varsayılmaktadır. Basit bir toplu iş ortamı için bu işler sırayla yürütülecektir. Böylece, JOB1 5 dakika içinde tamamlanır. JOB2 5 dakika dolana kadar beklemek zorundadır ve bundan 15 dakika sonra tamamlanır. JOB3 20 dakika sonra başlar ve ilk gönderildiği zamandan itibaren 30 dakika sonra tamamlanır. Ortalama kaynak kullanımı, verim ve yanıt süreleri Tablo 8.2'nin tek programlama sütununda gösterilmektedir. Cihaz bazında kullanım Şekil 8.6'a gösterilmiştir. Gerekli 30 dakikalık zaman diliminin ortalaması alındığında tüm kaynaklar için büyük bir eksik kullanım olduğu akıtsır.

Şimdi işlerin çok programlı bir işletim sistemi altında eşzamanlı olarak çalıştırıldığını varsayıyalım. İşler arasında çok az kaynak çekmesi olduğundan, üçde bilsayardaki diğer işlerle bir arada nerdedeysse minimum sürede çalışılabilir (JOB2 ve JOB3'e giriş ve çıkış işlemlerini etkin tutmak için yeterli işlemci süresi ayrıldığı varsayılırsa). JOB1'in tamamlaması için hala 5 dakika gerekecektir ancak bu sürenin sonunda JOB2 üçte birini, JOB3 ise yanısını olacaktır. Her üç iş de 15 dakika içinde bitmiş olacaktır. İyileşme, Şekil 8.6b'de gösterilen histogramdan elde edilen Tablo 8.2'nin çoklu programlama sütunu incelendiğinde açıkça görülmektedir.

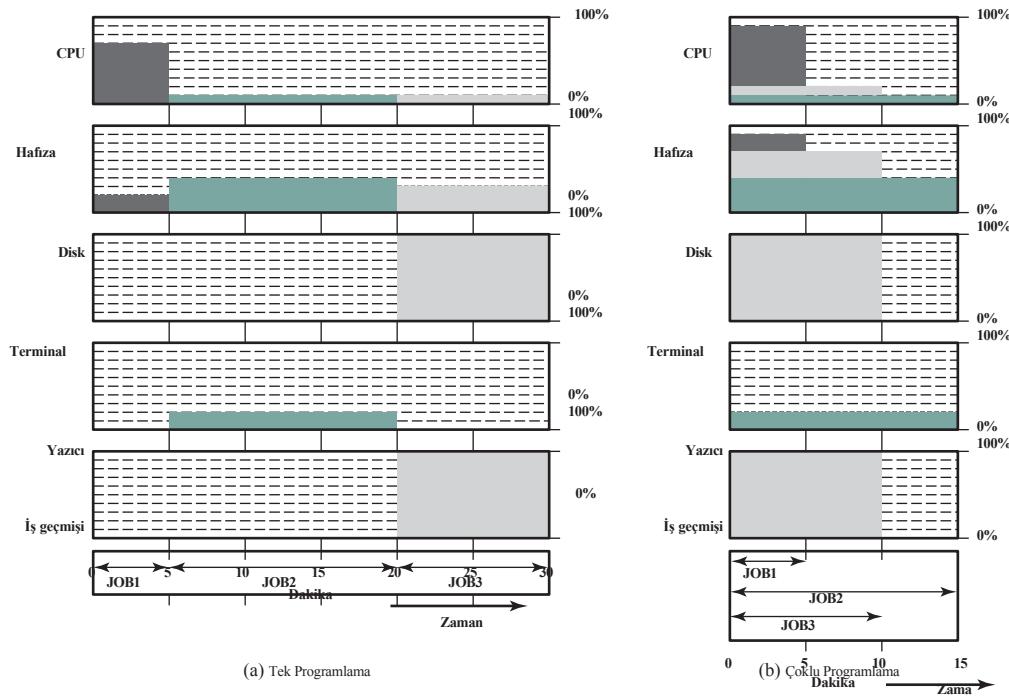
Basit bir toplu iG sisteminde olduğu gibi, çok programlı bir toplu iG sistemi de belirli bilgisayar donanım özelliklerine dayanmalıdır. Çoklu programlama için yararlı olan en önemli ek özellik, G/C kesintilerini destekleyen donanımdır.

**Tablo 8.1** Örnek Program Yürütme Öznitelikleri

|                             | JOB1           | JOB2     | JOB3     |
|-----------------------------|----------------|----------|----------|
| <b>İş türü</b>              | Ağır hesaplama | Ağır I/O | Ağır I/O |
| <b>Süre (dakika)</b>        | 5              | 15       | 10       |
| <b>Gerekli bellek (M)</b>   | 50             | 100      | 80       |
| <b>Disk mi lazımlı?</b>     | Hayır          | Hayır    | Evet     |
| <b>Terminal mi lazımlı?</b> | Hayır          | Evet     | Hayır    |
| <b>Yazıcı mı lazımlı?</b>   | Hayır          | Hayır    | Evet     |

**Tablo 8.2** Çoklu Programlamanın Kaynak Kullanımı Üzerindeki Etkileri

|                                       | Tek Programlama | Çoklu Programlama |
|---------------------------------------|-----------------|-------------------|
| <b>İşlemci kullanımı (%)</b>          | 20              | 40                |
| <b>Bellek kullanımı (%)</b>           | 33              | 67                |
| <b>Disk kullanımı (%)</b>             | 33              | 67                |
| <b>Yazıcı kullanımı (%)</b>           | 33              | 67                |
| <b>Geçen süre (dakika)</b>            | 30              | 15                |
| <b>Verim oranı (iş/saat)</b>          | 6               | 12                |
| <b>Ortalama yanıt süresi (dakika)</b> | 18              | 10                |



Şekil 8.6 Kullanım Histogramları

ve DMA. Kesme güdümlü I/O veya DMA ile işlemci bir iş için I/O komutu verebilir ve I/O aygit denetleyicisi tarafından gerçekleştirilirken başka bir işin yürütülmesine edebilir. G/C işlem tamamlandığında, işlemci kesilir ve kontrol işletim sistemindeki bir kesme işleme programına aktarılır. İşletim sistemi daha sonra kontrolü başka bir işe aktarır.

Çok programlı işletim sistemleri, tek programlı ya da tek **programlı** sistemlere kıyasla oldukça karmaşıktır. Birden fazla işin çalışmaya hazır olması için, işlerin ana bellekte tutulması gereklidir, bu da bir çeşit bellek **yönetimi** gerektirir. Buna ek olarak, birden fazla iş çalışmaya hazırlırsa, işlemci çalıştırılacağına karar vermelidir, bu da zamanlama için bazı algoritmalar gerektirir. Bu kavramlar bu bölümün ilerleyen kısımlarında ele alınacaktır.

**ZAMAN PAYLAŞIMLI SİSTEMLER** Çoklu programlama kullanıldığında, toplu işlem oldukça verimli olabilir. Bununla birlikte, birçok iş için, kullanıcının bilgisayarla doğrudan etkileşime girdiği bir modun sağlanması arzu edilir. Gerçekten de, işlem işleme gibi bazı işler için etkileşimli bir mod şarttır.

Günümüzde, interaktif bir bilgi işlem tesisi gereksinimi, özel bir mikrobilgisayar kullanılarak karşılanabilir ve çoğu zaman karşılanmaktadır. Çoklu bilgisayarın büyük ve maliyetli olduğu 1960'larda bu seçenek mevcut değildi. Bunun yerine zaman paylaşımı geliştirildi.

Çoklu programmanın işlemcisinin bir seferde birden fazla toplu iş yapmasına izin vermesi gibi, çoklu programlama da birden fazla etkileşimli iş yapmak için kullanılabilir. Bu ikinci durumda, teknik zaman paylaşımı olarak adlandırılır, çünkü prosesin zamanı birden fazla kullanıcı arasında paylaşılır. **Zaman paylaşımı bir sistemde**, birden fazla kullanıcı

**Tablo 8.3** Zaman Paylaşımına Karşı Toplu Çoklu Programlama

| Toplu Çoklu Programlama             | Zaman Paylaşımı                                  |
|-------------------------------------|--------------------------------------------------|
| Temel hedef                         | İşlemci kullanımını en üst düzeye çıkarın        |
| İşletim sistemine yönelerin kaynağı | İşle birlikte sağlanan iş kontrol dili komutları |
|                                     | Terminalde girilen komutlar                      |

İşletim sistemi her bir kullanıcı programının yürütülmesini kısa bir hesaplama patlamasına veya kuantumuna bölgerek terminaller aracılığıyla sisteme eş zamanlı olarak erişir. Dolayısıyla, aynı anda aktif olarak hizmet talep eden  $n$  kullanıcı varsa, her kullanıcı işletim sistemi ek yükünü saymazsa, etkin bilgisayar hızının yalnızca ortalama  $I/n$ 'sının görecektir. Bununla birlikte, insanların tepki süresinin nispeten yavaş olduğu göz önüne alındığında, uygun şekilde tasarlanmış bir sistemdeki tepki süresinin özel bir bilgisayardakiyle karşılaşılabilir olması gereklidir.

Hem toplu çoklu programlama hem de zaman paylaşımı çoklu programlamayı kullanır. Temel farklılıklar Tablo 8.3'te listelenmiştir.

## 8.2 PROGRAMLAMA

Çoklu programmanın anahtarı çizelgelemedir. Aslında, tipik olarak dört tür programlama söz konusudur (Tablo 8.4). Bunları ileride inceleyeceğiz. Ama önce **süreç** kavramını tanıtabiliriz. Bu terim ilk olarak 1960'larda Multics OS tasarımcıları tarafından kullanılmıştır. *İşten* biraz daha genel bir terimdir. *Süreç* terimi için aşağıdakiler de dahil olmak üzere birçok tanım

- Yürütülmekte olan bir program
- Bir programın "canlandırılmış ruhu"
- Bir işlemcinin atandığı varlık Bu kavram ilerledikçe daha

özellik hale gelecektir.

### Uzun Vadeli Programlama

Uzun vadeli programlayıcı, hangi programların işlenmek üzere sisteme kabul edileceğini belirler. Böylece çoklu programmanın derecesini (bellekteki süreç sayısı) kontrol eder. Bir iş ya da kullanıcı programı kabul edildikten sonra bir süreç haline gelir ve kısa vadeli zamanlayıcı için kuyruğa eklenir. Bazı sistemlerde, yeni oluşturulan bir süreç takas edilmiş bir durumda başlar, bu durumda orta vadeli zamanlayıcı için bir kuyruğa eklenir.

**Tablo 8.4** Çizelgeleme Türleri

|                                                        |                                                                                                                                                                              |
|--------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Uzun vadeli çizelgeleme Orta vadeli çizelgeleme</b> | Yürüttülecek süreçler havuzuna ekleme kararı.<br>Kısmen veya tamamen ana bellekte bulunan süreçlerin sayısına ekleme kararı.                                                 |
| <b>Kısa vadeli programlama</b>                         | İşlemci tarafından hangi mevcut işlemin yürütüleceğine ilişkin karar.<br>Hangi sürecin bekleyen G/C talebinin mevcut bir G/C cihazı tarafından yönetileceğine karar verilir. |
| <b>G/C zamanlaması</b>                                 |                                                                                                                                                                              |

Bir toplu iş sisteminde ya da genel amaçlı bir işletim sisteminin toplu iş bölümünde, yeni gönderilen işler diske yönlendirilir ve bir toplu iş kuyruğunda tutulur. Uzun vadeli zamanlayıcı, yapabildiğinde kuyrukta süreçler oluşturur. Burada iki karar söz konusudur. İlk olarak, zamanlayıcı işletim sisteminin bir ya da daha fazla ek süreç alabileceğine karar vermelidir. İkinci olarak, zamanlayıcı hangi iş ya da işlerin kabul edileceğine ve süreçlere dönüştürüleceğine karar vermelidir. Kullanılan kriterler öncelik, beklenen yürütme süresi ve G/C gereksinimlerini içerebilir. Zaman paylaşımı bir sistemdeki etkileşimli programlar için, bir kullanıcı sisteme bağlanmaya çalıştığında bir süreç talebi . Zaman paylaşımı kullanıcılar basitçe sıraya konmaz ve sistem onları kabul edene kadar bekletmez. Bunun yerine, işletim sistemi önceden tanımlanmış bir doygunluk ölçüsü kullanarak sistem doygunluğa ulaşana kadar tüm yetkili gelenleri kabul edecektir. Bu noktada, bir bağlantı isteği bir Sistemin dolu olduğunu ve kullanıcının daha sonra tekrar denemesi gerektiğini belirten mesaj.

### Orta Vadeli Programlama

Orta vadeli çizelgeleme, Bölüm 8.3'te açıklanan takas işlevinin bir parçasıdır. Tipik olarak, takas kararı çoklu programlama derecesini yönetme ihtiyacına dayanır. Sanal bellek kullanmayan bir sistemde, bellek yönetimi de bir sorundur. Bu nedenle, takas kararı takas edilen süreçlerin bellek gereksinimlerini dikkate alacaktır.

### Kısa Vadeli Programlama

Uzun vadeli zamanlayıcı nispeten seyrek olarak çalışır ve yeni bir süreç alıp almayacağına ve hangisini alacağına ilişkin kaba taneli kararları verir. **Dağıtıcı** olarak da bilinen kısa vadeli zamanlayıcı sık sık çalışır ve hangi işin bir sonraki aşamada yürütüleceğine dair ince taneli kararları verir.

**İŞLEM DURUMLARI** Kısa vadeli zamanlayıcının işleyişini anlamak için, **işlem durumu** kavramını göz önünde bulundurmamız gereklidir. Bir sürecin ömrü boyunca, durumu birkaç kez değişecektir. Zamanın herhangi bir noktasındaki durumu *durum* olarak adlandırılır. *Durum* terimi, o noktadaki durumu tanımlayan belirli bilgilerin mevcut olduğunu çağrıstdırdığı için kullanılır. Bir süreç için en az beş tanımlı durum vardır (Şekil 8.7):

- **Yeni:** Bir program üst düzey zamanlayıcı tarafından kabul edilir ancak henüz yürütülmeye hazır değildir. İşletim sistemi süreci başlatabak hazır duruma getirecektir.



Şekil 8.7

Beş Durumlu Süreç Modeli

- **Hazır:** İşlem yürütülmeye hazır ve işlemciye erişim beklemektedir.
  - **Çalışıyor:** İşlem, işlemci tarafından yürütülmektedir.
  - **Bekliyor:** Süreç, G/C gibi bazı sistem kaynaklarını beklerken yürütme askıya alınır.
  - **Durduruldu:** Süreç sonlandırıldı ve işletim sistemi tarafından yok edilecek.

Sistemdeki her bir süreç için, işletim sistemi sürecin durumunu gösteren bilgileri ve sürecin yürütülmesi için gerekli diğer bilgileri muhafaza etmelidir. Bu amaçla, her süreç işletim sisteminde tipik olarak aşağıdakileri içeren bir **süreç kontrol bloğu** (Şekil 8.8) ile temsil edilir:

- **Tanımlayıcı:** Her mevcut sürecin benzersiz bir tanımlayıcısı vardır.
  - **Durum:** Sürecin mevcut durumu (yeni, hazır vb.).
  - **Öncelik:** Görecli öncelik seviyesi.
  - **Program sayacı:** Yürüttülecek programdaki bir sonraki komutun adresi.
  - **Bellek işaretçileri:** Sürecin bellekteki başlangıç ve bitiş konumları.
  - **Bağlam verileri:** Bunlar, işlem yürütülürken işlemcideki kayıtlarda bulunan verilerdir ve Üçüncü Bölüm'de ele alınacaktır. Şimdilik, bu verilerin sürecin "bağlamını" temsil ettiğini söylemek yeterlidir. Bağlam verileri artı program sayacı, süreç çalışma durumundan ayrıldığında kaydedilir. İşlemci sürecin yürütülmesine devam ettiğinde bunlar işlemci tarafından geri alınır.

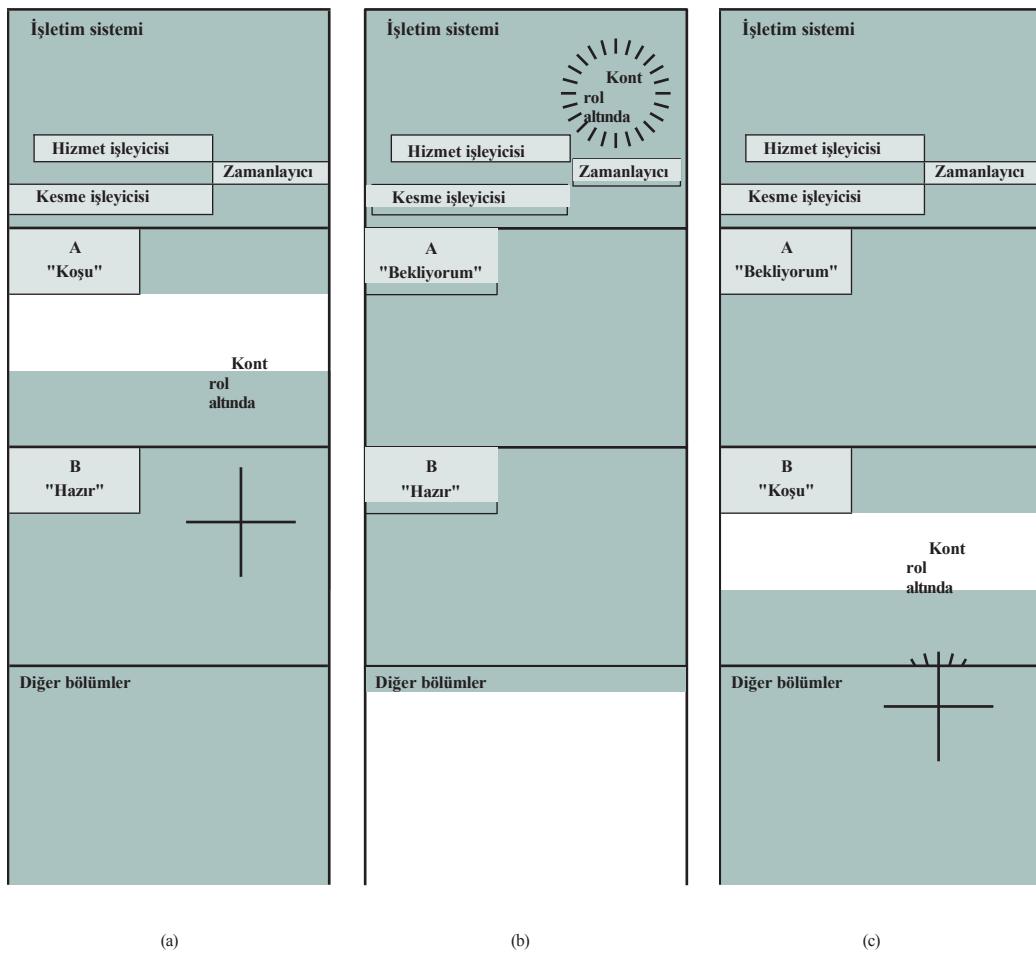
|                     |
|---------------------|
| Tanımlayıcı         |
| Eyalet              |
| Öncelik             |
| Program sayacı      |
| Bellek işaretçileri |
| Bağlam verileri     |
| G/Ç durum bilgisi   |
| Muhasebe bilgileri  |
| -                   |
| -                   |
| -                   |

**Şekil 8.8** Proses Kontrol Bloğu

- G/C durum bilgileri:** Bekleyen G/C isteklerini, bu işleme atanan G/C aygıtlarını (örn. teyp sürücülerü), işleme atanan dosyaların bir listesini vb. içerir.
- Muhasebe bilgileri:** Kullanılan işlemci zamanı ve saat zamanı miktarmı, zaman sınırlarını, hesap numaralarını vb. içerebilir.

Zamanlayıcı yürütme için yeni bir iş veya kullanıcı talebi kabul ettiğinde, boş bir süreç kontrol bloğu oluşturur ve ilgili süreci yeni duruma yerleştirir. Sistem süreç kontrol沼unu düzgün bir şekilde doldurduktan sonra, süreç hazır duruma aktarılır.

**ÇİZELGELEME TEKNİKLERİ** İşletim sisteminin bellekteki çeşitli işlerin çizelgelenmesini nasıl yönettiğini anlamak için, Şekil 8.9'daki basit örneği ele alarak başlayalım. Şekil, ana belleğin belirli bir zamanda nasıl bölümlendiğini göstermektedir. İşletim sisteminin çekirdeği elbette her zaman yerlesktir. Buna ek olarak, her birine belleğin bir kısmı tahsis edilmiş olan **A** ve **B** dahil olmak üzere bir dizi aktif süreç vardır.

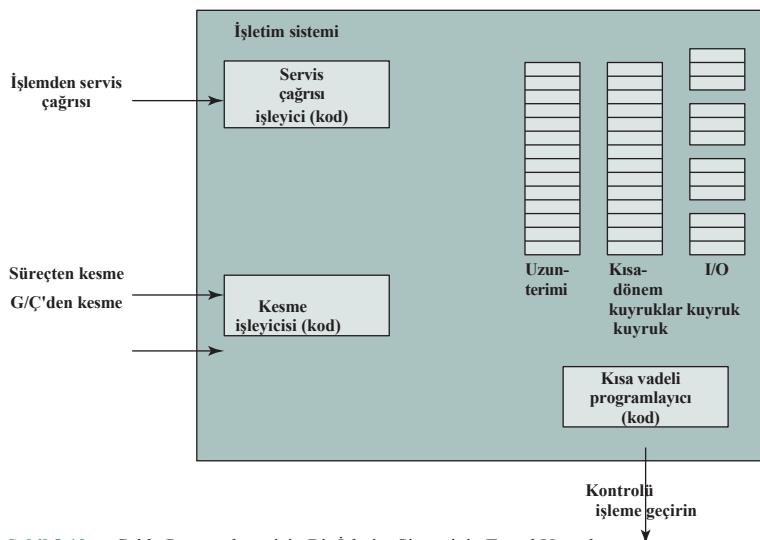


**Şekil 8.9** Çizelgeleme Örneği

A işleminin çalıştığı bir zaman noktasında başlıyoruz. İşlemci, A'nın bellek bölümünden bulunan programdaki talimatları yürütmektedir. Zamanın ilerleyen bir noktasında, işlemci A'daki talimatları yürütmeyi bırakır ve OS alanındaki talimatları başlar. Bu üç nedenden biri nedeniyle gerçekleşecektir:

1. A süreci işletim sistemine bir hizmet çağrı (örneğin, bir G/C isteği) gönderir. A'nın yürütülmesi bu çağrı işletim sistemi tarafından karşılanana kadar askıya alınır.
2. A işlemi bir kesmeye neden olur. Kesme, işlemciye donanım üretilen bir sinyaldir. Bu sinyal algılandığında, işlemci A'yı yürütmeyi durdurur ve işletim sistemindeki kesme işleyicisine aktarır. A ile ilgili çeşitli olaylar bir kesmeye neden olur. Bir örnek, ayrıcalıklı bir komutun yürütülmeye çalışılması gibi bir hatadır. Başka bir örnek de zaman aşımıdır; herhangi bir sürecin işlemciyi tekeline almasını önlemek için her sürece işlemci bir seferde yalnızca kısa bir süre için verilir.
3. A işlemiyle ilgisi olmayan ve dikkat gerektiren bir olay kesmeye neden olur. Bir G/C işleminin tamamlanması buna örnektir.

Her durumda, sonuç aşağıdaki gibidir. İşlemci geçerli bağlam verilerini ve A için program sayacını A'nın işlem kontrol bloğuna kaydeder ve ardından işletim sisteminde yürütmeye başlar. İşletim sistemi bir G/C işlemi başlatmak gibi bazı işler gerçekleştirebilir. Daha sonra işletim sisteminin kısa dönemli zamanlayıcı kısmı bir sonraki işlemin hangisi olacağına karar verir. Bu örnekte B seçilmiştir. İşletim sistemi işleme B'nin bağlam verilerini geri yüklemesini ve B'nin yürütülmesine kaldığı yerden devam etmesini söyler. Bu basit örnek, kısa vadeli programın temel işleyişini vurgular. Şekil 8.10, süreçlerin çoklu programlanması ve çizelgelelenmesinde yer alan işletim sisteminin ana unsurlarını göstermektedir. İşletim sistemi işlemcinin kontrolünü



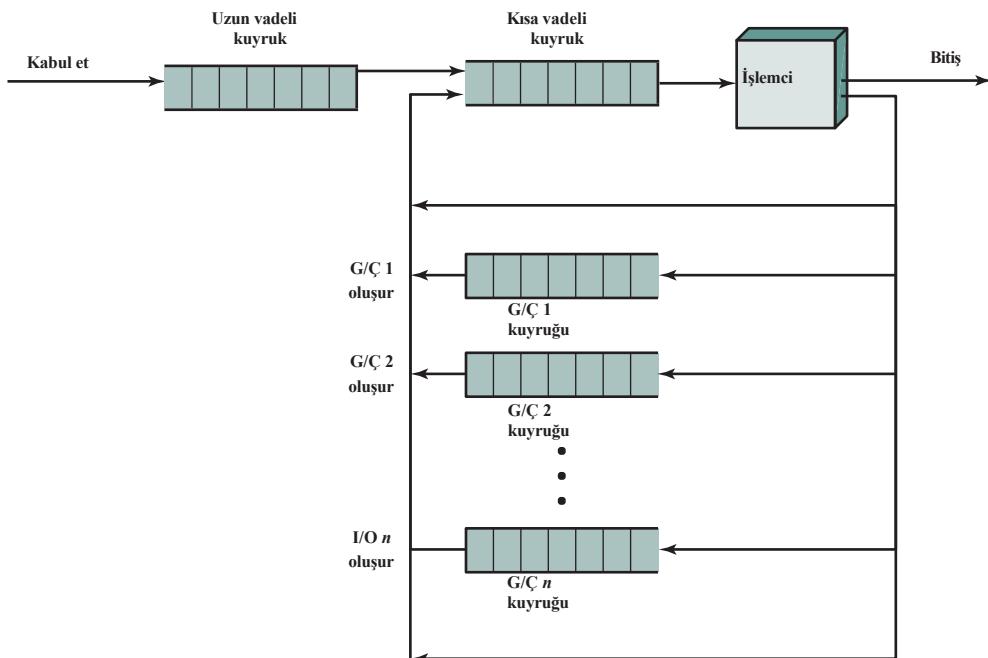
Şekil 8.10 Çoklu Programlama için Bir İşletim Sisteminin Temel Unsurları

bir kesme meydana gelirse kesme işleyicisinde ve bir hizmet çağrısi gelirse hizmet çağrısi işleyicisinde. Kesme veya hizmet çağrısi işlendikten sonra, yürütme için bir işlem seçmek üzere kısa vadeli zamanlayıcı .

İşini yapmak için işletim sistemi bir dizi kuyruk tutar. Her kuyruk basitçe bazı kaynakları bekleyen işlemlerin bekleme listesidir. **Uzun vadeli kuyruk**, sistemi kullanmak için bekleyen işlemlerin bir listesidir. Koşullar izin , üst düzey zamanlayıcı bellek ayırrır ve bekleyen öğelerden biri için bir süreç oluşturur. **Kısa vadeli kuyruk**, hazır durumındaki tüm süreçlerden oluşur. Bu işlemlerden herhangi biri bir sonraki işlemciyi kullanabilir. Birini seçmek kısa vadeli zamanlayıcıya bağlıdır. Genellikle bu, her süreçse sırayla biraz zaman taniyan bir round-robin algoritması ile yapılır. Öncelik seviyeleri de kullanılabilir. Son olarak, her G/C aygıti için bir G/C **kuyruğu** vardır. Birden fazla süreç aynı G/C aygitini kullanmayı talep edebilir. Her aygıti kullanmak için bekleyen tüm süreçler o aygitin kuyruğunda sıralanır.

Şekil 8.11, işlemlerin işletim sisteminin kontrolü altında bilgisayarda nasıl ilerlediğini göstermektedir. Her işlem isteği (toplu iş, kullanıcı tanımlı etkileşimli iş) uzun vadeli kuyruğa yerleştirilir. Kaynaklar kullanılabilir hale geldikçe, bir süreç isteği bir süreç haline gelir ve ardından hazır duruma getirilerek kısa vadeli kuyruğa yerleştirilir. İşlemci, işletim sistemi talimatlarını yürütmem ve kullanıcı işlemlerini yüklemek arasında gidip gelir. İşletim sistemi kontrolü elinde tutarken, kısa vadeli kuyruktaki hangi sürecin daha sonra yürütüleceğine karar verir. İşletim sistemi acil görevlerini tamamladığında, işlemciyi seçilen işleme devreder.

Daha önce de belirtildiği gibi, yürütülmekte olan bir süreç çeşitli nedenlerle askıya alınabilir. Süreç G/C talep ettiği için askıya alınmışsa, o zaman



**Şekil 8.11** İşlemci Çizelgelemesinin Kuyruk Diyagramı Gösterimi

uygun G/C kuyruğuna yerleştirilir. Zaman aşımı nedeniyle veya işletim sisteminin acil bir işle ilgilenesmesi gerektiği için askıya alınırsa, hazır duruma getirilir ve kısa vadeli kuyruğa yerleştirilir.

Son olarak, işletim sisteminin G/C kuyruklarını da yönettiğini belirtelim. Bir G/C işlemi tamamlandığında, işletim sistemi memnun olan süreci G/C kuyruğundan kaldırır ve kısa vadeli yerleştirir. Daha sonra bekleyen başka bir süreci (varsayıf) sefer ve G/C cihazına bu sürecin talebini karşılaması için sinyal gönderir.

### 8.3 MEMOrY YÖNETİMİ

Tek programlı bir sistemde, ana bellek iki bölüme ayrılır: bir bölüm işletim sistemi (yerleşik monitör) ve bir bölüm o anda yürütülmekte olan program için. Çok programlı bir sistemde, belleğin "kullanıcı" kısmı birden fazla işlemi barındırmak için alt böülümlere ayrılır. Alt böülümlere ayırma görevi işletim sistemi tarafından dinamik olarak gerçekleştirilebilir ve **bellek yönetimi** olarak bilinir.

Etkili bellek yönetimi çok programlı bir sistemde hayatı önem taşır. Bellekte yalnızca birkaç işlem varsa, zamanın çoğunda tüm işlemler G/C için bekliyor olacak ve işlemci boşta kalacaktır. Bu nedenle, mümkün olduğunca çok sayıda işlemi belleğe sıkıştırmak için belleğin verimli bir şekilde tahsis edilmesi gereklidir.

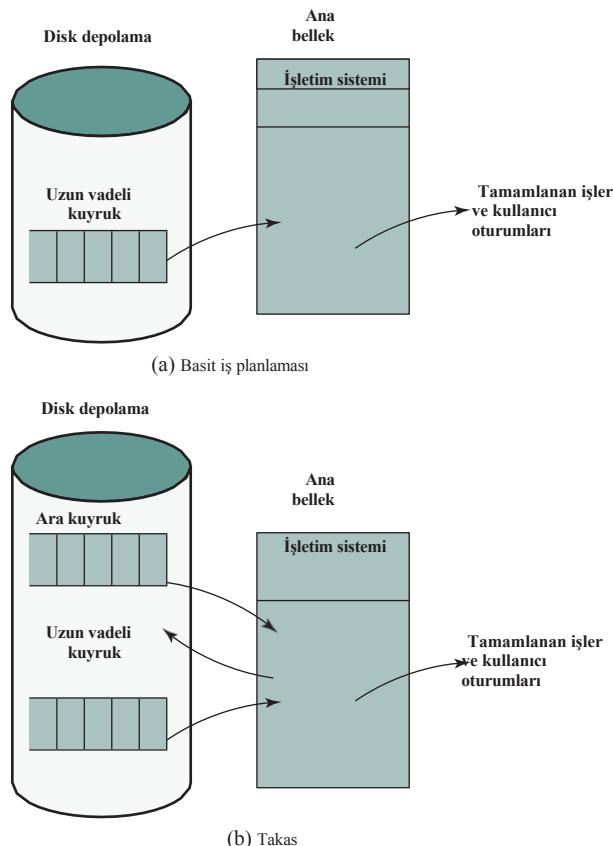
#### Takas

Şekil 8.11'e geri dönersek, üç tip kuyruktan bahsetmiştim: yeni süreçler için uzun vadeli talep kuyruğu, işlemciyi kullanmaya hazır süreçlerin kısa vadeli kuyruğu ve işlemciyi kullanmaya hazır olmayan süreçlerin çeşitli I/O kuyrukları. Bu ayrıntılı mekanizmanın nedeninin G/C faaliyetlerinin hesaplamadan çok daha yavaş olması ve bu nedenle tek programlı bir sistemde işlemcinin çoğu zaman boşta kalması olduğunu hatırlayın.

Ancak Şekil 8.11'deki düzenleme sorunu tamamen çözmez. Bu durumda, belleğin birden fazla işlemi tuttuğu ve bir işlem beklerken işlemcinin başka bir işleme geçebileceği doğrudur. Ancak işlemci G/C'den o hızlıdır ki, bellekteki *tüm* işlemlerin G/C'yi beklemesi yaygın bir durum olacaktır. Bu nedenle, çoklu programlama olsa bile, bir işlemci çoğu zaman boşta kalabilir.

Ne yapmalı? Ana bellek genişletilebilir ve böylece daha fazla işlem gerçekleştirilebilir. Ancak bu yaklaşımın iki kusuru vardır. Birincisi, ana bellek bugün bile pahalıdır. İkincisi, programların belleğe olan istahı, bellek maliyetinin düşmesi kadar hızlı arımıştır. Dolayısıyla daha büyük bellek daha fazla işlemle değil daha büyük süreçlerle sonuçlanır.

Bir başka çözüm de Şekil 8.12'de gösterilen **takas** yöntemidir. Tipik olarak diskte depolanan uzun vadeli bir işlem istekleri kuyruğumuz vardır. Bunlar her seferinde bir tane olmak yer açıldıktan sonra getirilir. İşlemler tamamlandıktan sonra bellekten çıkarılırlar. Şimdi, bellekteki işlemlerin hiçbirinin hazır durumda olmadığı bir durum ortaya çıkacaktır (örneğin, hepsi bir G/C işlemini beklemektedir). İşlemci boşta kalmak yerine bu işlemlerden birini diske geri göndererek *bir ara kuyruğa* yerleştirir. Bu, geçici olarak hazır durumda olan mevcut işlemlerden oluşan bir kuyruktur.



Şekil 8.12 Takas Kullanımı

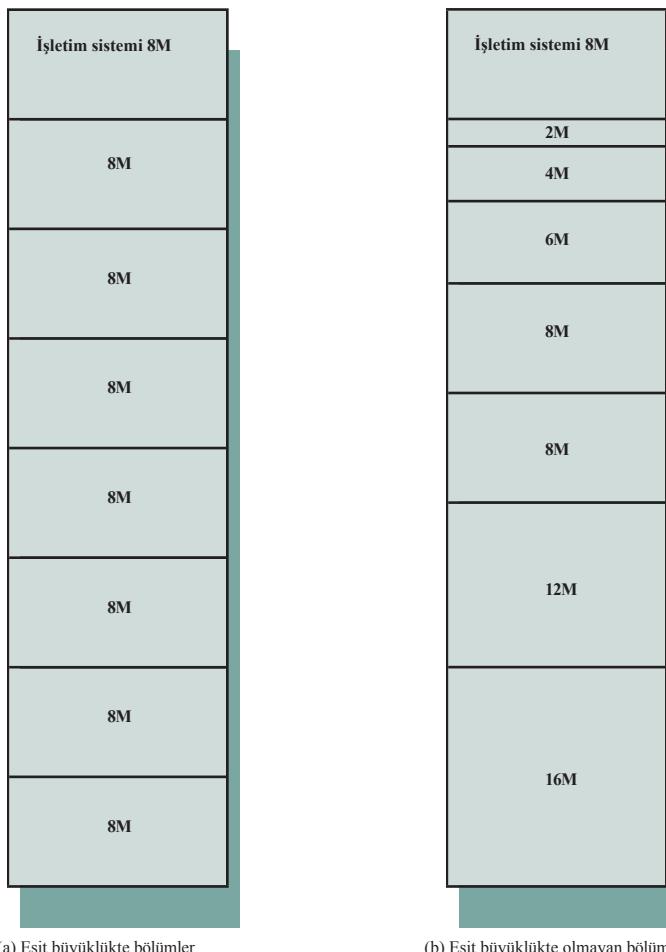
bellekten atılır. İşletim sistemi daha sonra ara kuyruktan başka bir süreç getirir veya uzun vadeli kuyruktan yeni bir süreç talebini kabul eder. Yürütmeye daha sonra yeni gelen süreçle devam eder.

Ancak takas bir G/Ç işlemidir ve bu nedenle sorunu iyileştirmek yerine daha da kötüleştirme potansiyeli vardır. Ancak disk G/Ç'si genellikle bir sistemdeki en hızlı G/Ç olduğu için (örneğin, teyp ya da yazıcı G/Ç'si ile karşılaştırıldığında), takas işlemi genellikle performansı artıracaktır. Sanal bellek içeren daha sofistik bir şema, basit takaslamaya göre performansı artırır. Bu kısa bir süre sonra tartışılacaktır. Ama önce, bölümleme ve sayfalamayı açıklayarak zemini hazırlamalıyız.

### Bölümleme

Kullanılabilir belleği bölümlemek için en basit şema, Şekil 8.13'te gösterildiği gibi *sabit boyutlu bölümler* kullanmaktadır. Bölümler sabit boyutta olsa da, eşit boyutta olmaları gerekmeye dikkat edin. Bir işlem belleğe getirildiğinde, onu tutacak en küçük kullanılabilir bölüme yerleştirilir.

Eşit olmayan sabit boyutlu bölümler kullanılsa bile, boş harcanan bellek olacaktır. Çoğu durumda, bir işlem tam olarak sağlanan kadar belleğe ihtiyaç duymayacaktır.



Şekil 8.13 64 Mbyte Belleğin Sabit Bölümlendirilmesi Örneği

bölüm tarafından. Örneğin, 3M bayt belleğe ihtiyaç duyan bir işlem Şekil 8.13b'deki 4M'lik bölüme yerleştirilir ve başka bir işlem tarafından kullanılabilecek 1M boş harcanır.

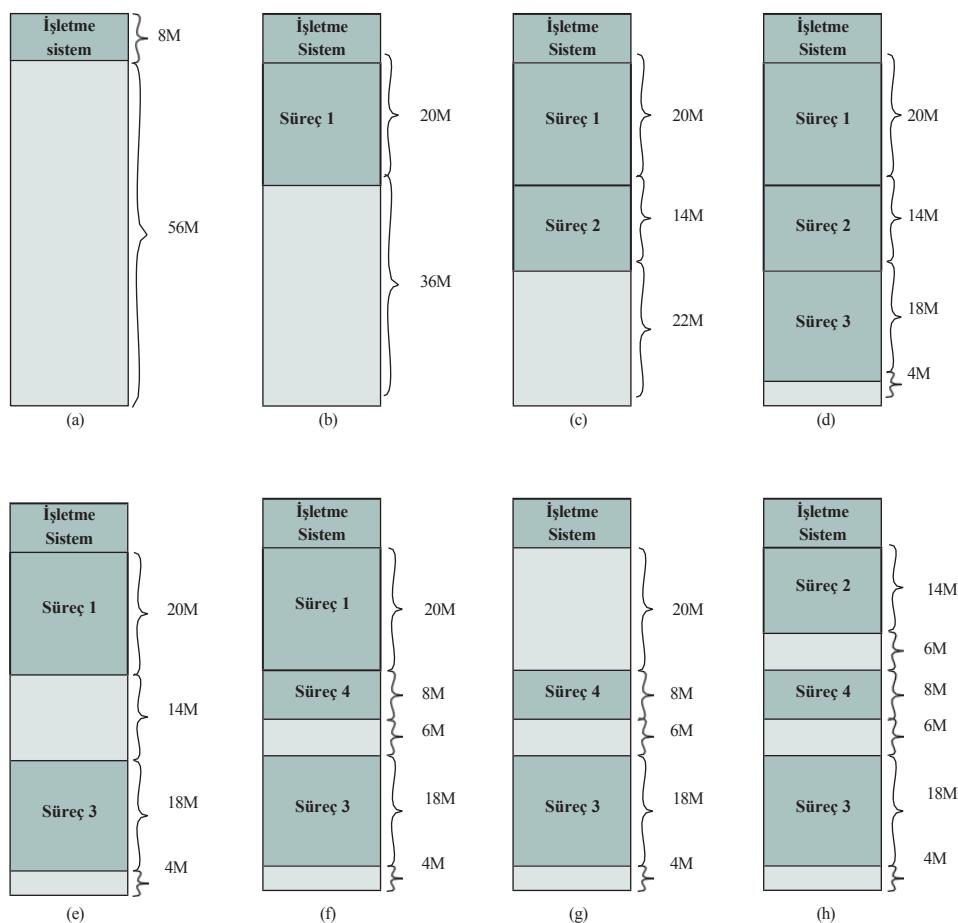
Daha verimli bir yaklaşım *değişken boyutlu bölmeler* kullanmaktadır. Bir süreç belleğe getirildiğinde, tam olarak ihtiyaç duyduğu kadar bellek tahsis edilir ve daha fazlası tahsis edilmez.

**ÖRNEK 8.2** 64 Mbyte ana bellek kullanan bir örnek Şekil 8.14'te gösterilmektedir. Başlangıçta, ana bellek işletim sistemi dışında boştur (a). İlk üç süreç, işletim sisteminin bittiği yerden başlayarak ve her bir süreç için yeterli yer kaplayarak yüklenir (b, c, d). Bu, belleğin sonunda dördüncü bir işlem için çok küçük bir "delik" bırakır. Bir noktada, bellekteki süreçlerden hiçbirini hazır değildir. İşletim sistemi süreç 2'yi (e) değiştirerek yeni bir süreç olan süreç 4'ü (f) yüklemek için yeterli alan bırakır. Süreç 4, süreç 2'den daha küçük olduğu için küçük bir delik daha yaratılır. Daha sonra, ana bellekteki hiçbir sürecin hazır olmadığı, ancak hazır-askıda süreç 2'nin kullanılabilir olduğu bir noktaya ulaşılır. Bellekte süreç 2 için yeterli yer olmadıktan sonra, işletim sistemi süreç 1'i dışarı (g) ve süreç 2'yi içeri (h) değiştirir.

Bu örnekte görüldüğü gibi, bu yöntem iyi başlar, ancak sonunda bellekte çok sayıda küçük deliklerin olduğu bir duruma yol açar. Zaman geçtikçe bellek giderek daha fazla parçalanır ve bellek kullanımı azalır. Bu sorunun üstesinden gelmek için kullanılan tekniklerden biri **sıkıştırmadır**: İşletim sistemi zaman zaman bellekteki işlemleri kaydırarak tüm boş belleği tek bir blokta toplar. Bu zaman alıcı bir işlemidir ve işlemci zamanını boş harcar.

Bölümlemin eksiklikleriyle başa çıkmanın yollarını düşünmeden önce, yarı kalmış bir iş tamamlamalıyız. Şekil 8.14'ü düşünün; bir sürecin her takas edildiğinde ana bellekte aynı yere yüklenmeyeceği açık olmalıdır. Ayrıca, sıkıştırma kullanılıyorsa, bir süreç ana bellekteyken kaydırılabilir. Bellekteki bir işlem talimatlar artı verilerden oluşur. Talimatlar iki tür bellek konumu için adresler içerecektir:

- Veri öğelerinin adresleri
- Talimatların adresleri, dallanma talimatları için kullanılır



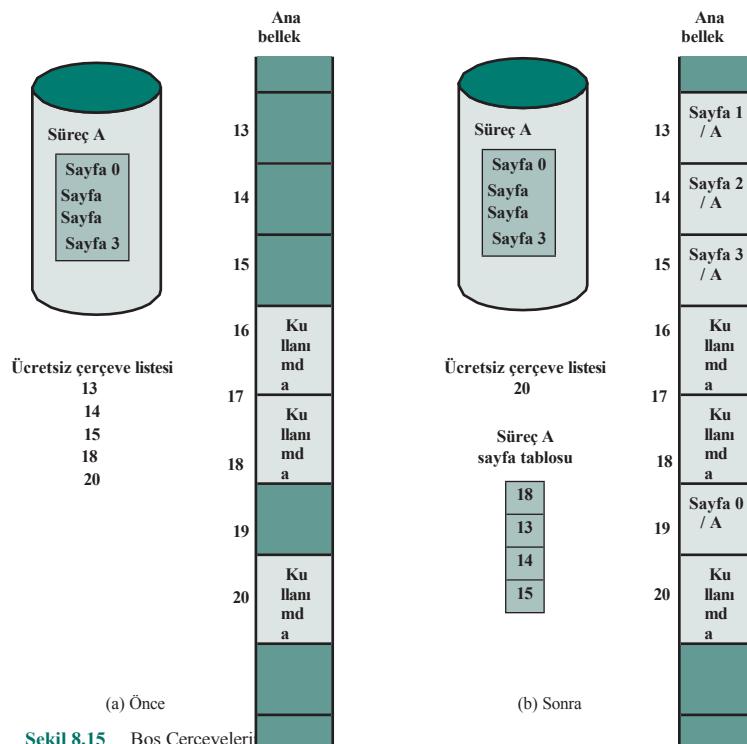
**Şekil 8.14** Dinamik Bölümlendirmenin Etkisi

Ancak bu adresler sabit değildir. Bir işlem her değişeceklereidir. Bu sorunu çözmek için mantıksal adresler ile fiziksel adresler arasında bir ayrımlı yapılmıştır. **Mantıksal adres**, programın başlangıcına göre bir konum olarak ifade edilir. Programdaki talimatlar yalnızca mantıksal adresleri içerir. **Fiziksel adres** ana bellekteki gerçek bir konumdur. İşlemci bir işlemi yürütürüğünde, işlemin **temel adresi** olarak adlandırılan mevcut başlangıç konumunu her bir mantıksal adrese ekleerek otomatik olarak mantıksal adresen fiziksel adrese dönüştürür. Bu, bir işletim sistemi gereksinimini karşılamak için tasarlanmış bir başka işlemci donanım özelliği örneğidir. Bu donanım özelliğinin tam doğası, kullanılan bellek yönetim stratejisine bağlıdır. Bu bölümün ilerleyen kısımlarında birkaç örnek göreceğiz.

### Çağrı

Hem eşit olmayan sabit boyutlu hem de değişken boyutlu bölümler bellek kullanımında verimsizdir. Bununla birlikte, belleğin nispeten küçük olan eşit sabit boyutlu parçalara bölündüğünü ve her işlemin de belirli boyutta küçük sabit boyutlu parçalara bölündüğünü varsayılmı. O zaman bir programın **sayfalar** olarak bilinen parçaları, **çerçeveeler** ya da sayfa çerçeveleri olarak bilinen mevcut bellek parçalarına atanabilir. Bu durumda, o süreç için bellekte boş harcanan alan en fazla son sayfanın bir kısmıdır.

Şekil 8.15 sayfaların ve çerçeveelerin kullanımına bir örnek göstermektedir. Belirli bir zamanda, bellekteki çerçeveelerin bazıları kullanılmadır ve bazıları boştur. Boş çerçevelerin listesi işletim sistemi tarafından tutulur. Diskte saklanan A işlemi dört sayfadır.



Bu işlemi yükleme zamanı geldiğinde, işletim sistemi dört boş çerçeve bulur ve A işleminin dört sayfasını dört çerçeveye yükler.

Şimdi, bu örnekte olduğu gibi, süreci tutmak için yeterli sayıda kullanılmayan bitişik çerçeve olmadığını varsayıyalım. Bu durum işletim sisteminin A'yi yüklemesini engeller mi? Cevap hayır, çünkü bir kez daha mantıksal adres kavramını kullanabiliriz. Basit bir taban adresi artık yeterli olmayacağından. Bunun yerine, işletim sistemi her işlem için bir **sayfa tablosu** tutar. Sayfa tablosu sürecin her sayfası için çerçeve konumunu gösterir. Program içinde, her mantıksal adres bir sayfa numarası ve sayfa içindeki görelİ adresten oluşur. Basit bölümleme durumunda, mantıksal adresin programın başlangıcına göre bir sözcüğün konumu olduğunu hatırlayın; işlemci bunu fiziksel bir adrese çevirir. Sayfalama ile, mantıksal adres-fiziksel adres çevirisi hala işlemci donanımı tarafından yapıılır. İşlemci, mevcut sürecin sayfa tablosuna nasıl erişeceğini bilmelidir. Mantıksal bir adres (sayfa numarası, görelİ adres) verildiğinde, işlemci fiziksel bir adres (çerçeve numarası, görelİ adres) üretmek için sayfa tablosunu kullanır. Şekil 8.16'da bir örnek gösterilmektedir.

Bu yaklaşım daha önce ortaya konan sorunları çözmektedir. Ana bellek birçok küçük eşit boyutlu çerçeveye bölünmüştür. Her işlem çerçeve boyutunda sayflara bölünür: daha küçük işlemler daha az sayfa gerektirir, daha büyük işlemler daha fazla sayfa gerektirir. Bir süreç, sayfaları mevcut çerçevelere yüklenir ve bir sayfa tablosu oluşturulur.

Ana bellek

The diagram illustrates the memory addressing process across three levels:

- Sayfa numarası (Page Number):** Represented by the number 1 in the first column of the page table.
- Göreceli adres sayfa içinde (Logical Address within page):** Represented by the number 30 in the first column of the page table.
- Fiziksel adres (Physical Address):** Composed of the page number (1) and the frame number (13).
- Cerçeve Bağlı adres numarası çerçeveye içinde (Frame Number within page frame):** Represented by the number 30 in the second column of the page table.
- İşlem A sayfa tablosu (Page Table A):** A table showing page mappings for different logical addresses. It has four rows corresponding to logical addresses 15, 14, 13, and 18. Row 13 maps to physical address 13, which is further broken down into page number 1 and frame number 13.
- Main Memory:** A vertical stack of pages labeled Sayfa 0/A, Sayfa 1/A, Sayfa 2/A, Sayfa 3/A, and Sayfa 13/A. The physical address 13 points to the start of the first page in memory.

**Sekil 8.16** Mantıksal ve Fiziksel Adresler

## Sanal Bellek

**DEMAND PAGING Sayfalamamın** kullanılmasıyla, gerçekten etkili çoklu programlama sistemleri ortaya çıktı. Dahası, bir süreci sayfalara bölmek bir taktiği bir başka önemli kavramın gelişmesine yol açtı: sanal bellek.

Sanal belleği anlamak için, az önce tartışılan disk belleği şemasına bir iyileştirme eklemeliyiz. Bu iyileştirme, basitçe bir işlemin her sayfasının yalnızca ihtiyaç duyulduğunda, yani **talep** üzerine getirildiği anlamına gelen talep sayfalamasıdır.

Uzun bir program ve çok sayıda veri dizisinden oluşan büyük bir süreç düşünün. Herhangi bir kısa zaman diliminde, yürütme programın küçük bir bölümüyle (örneğin bir alt rutin) sınırlı olabilir ve belki de yalnızca bir veya iki veri dizisi kullanılıyor olabilir. Bu, Ek 4A'da tanıttığımız yerelik ilkesidir. Program askıya alınmadan önce yalnızca birkaç sayfa kullanılabileceksa, bu işlem için düzinelere sayfa yüklemek açıkça savurganalık olacaktır. Sadece birkaç sayfa yükleyerek belleği daha iyi. Ardından, program ana bellekte olmayan bir sayfadaki bir komuta dallanırsa veya program bellekte olmayan bir sayfadaki verilere başvurursa, bir **sayfa hatası** tetiklenir. Bu, işletim sisteme istenen sayfayı getirmesini söyler.

Böylece, herhangi bir zamanda, herhangi bir işlemin yalnızca birkaç sayfası bellekte bulunur ve bu nedenle bellekte daha fazla işlem tutulabilir. Ayrıca, kullanılmayan sayfalar belleğe girip çıkmadığı için zamandan tasarruf edilir. Ancak, işletim sistemi bu şemayı nasıl yöneteceğini konusunda akıllı olmalıdır. Bir sayfayı içeri aldıgında, başka bir sayfayı dışarı atmalıdır; bu **sayfa değiştirme** olarak bilinir. Eğer bir sayfayı tam kullanılmak üzereken dışarı atarsa, o zaman hemen o sayfayı tekrar almak zorunda kalacaktır. Bunun çok fazla yapılması, **thrashing** olarak bilinen bir duruma yol açar: işlemci zamanının çoğunu talimatları yerine getirmek yerine sayfaları değiştirerek geçirir. Thrashing'in önlenmesi 1970'lerde önemli bir araştırma alanıydı ve çeşitli karmaşık ama etkili algoritmalarla yol açtı. Temelde, işletim sistemi yakın geçmişte hangi sayfaların yakın gelecekte kullanılma olasılığının en düşük olduğunu tahmin etmeye çalışır.



### Sayfa Değiştirme Algoritması Simülatörleri

Sayfa değiştirme algoritmalarının tartışıması bu bölümün kapsamı dışındadır. Potansiyel olarak etkili bir teknik, Bölüm 4'te önbellek değiştirme için tartışılan aynı algoritma olan en son kullanılan (LRU) tekniktir. Pratikte LRU'nun bir sanal bellek sayfalaması şemasi için uygulanması zordur. LRU'nun performansına yaklaşmaya çalışan birkaç alternatif yaklaşım kullanılmıştır; ayrıntılar için Ek K'ya bakın.

İsteğe bağlı sayfalaması ile bir sürecin tamamının ana belleğe yüklenmesi gerekmeyez. Bu gerçeğin dikkate değer bir sonucu vardır: *Bir sürecin ana belleğin tamamından daha büyük olması mümkün değildir.* Programlamadaki en temel kısıtlamalardan biri ortadan kalkmıştır. İsteğe bağlı sayfalaması olmadan, bir programcı ne kadar bellek bulunduğu konuda olmalıdır. Yazılan program çok büyükse, programcı programı daha sonra kullanabilecek parçalar halinde yapılandırmanın yollarını bulmalıdır.

her seferinde bir tane yüklenir. İsteğe bağlı sayfalama ile bu iş işletim sisteme ve sabit donanıma bırakılır. Programcı söz konusu olduğunda, disk depolama ile ilişkili boyutta büyük bir uğraşmaktadır.

Bir süreç yalnızca ana bellekte çalıştığından, bu bellek **gerçek bellek** olarak adlandırılır. Ancak bir programcı ya da kullanıcı çok daha büyük bir bellek alımları - diskte tahsis edilen bellek. Bu nedenle bu sonucusu **sanal bellek** olarak adlandırılır. Sanal bellek çok etkili çoklu programlamaya izin verir ve kullanıcıyı ana belleğin gereksiz sıkı kısıtlamalarından kurtarır.

**SAYFA TABLOSU YAPISI** Bellekten kelime okumanın temel mekanizması, sayfa numarası ve ofsetten oluşan sanal ya da mantıksal adresin, sayfa tablosu kullanılarak çerçeve numarası ve ofsetten oluşan fiziksel adrese çevrilmesini içerir. Sayfa tablosu, işlemin boyutuna bağlı olarak değişken uzunlukta olduğundan, onu kayıtlarda tutmayı bekleyemeyiz. Bunun yerine, erişilebilmesi için ana bellekte olması gereklidir. Şekil 8.16 bu şemanın bir donanım uygulamasını göstermektedir. Belirli bir işlem çalışırken, bir yazmaç o işlem için sayfa tablosunun başlangıç adresini tutar. Bir sanal adresin sayfa numarası bu tabloyu indekslemek ve ilgili çerçeve numarasını aramak için kullanılır. Bu, istenen gerçek adresi üretmek için sanal adresin ofset kısmını ile birleştirilir.

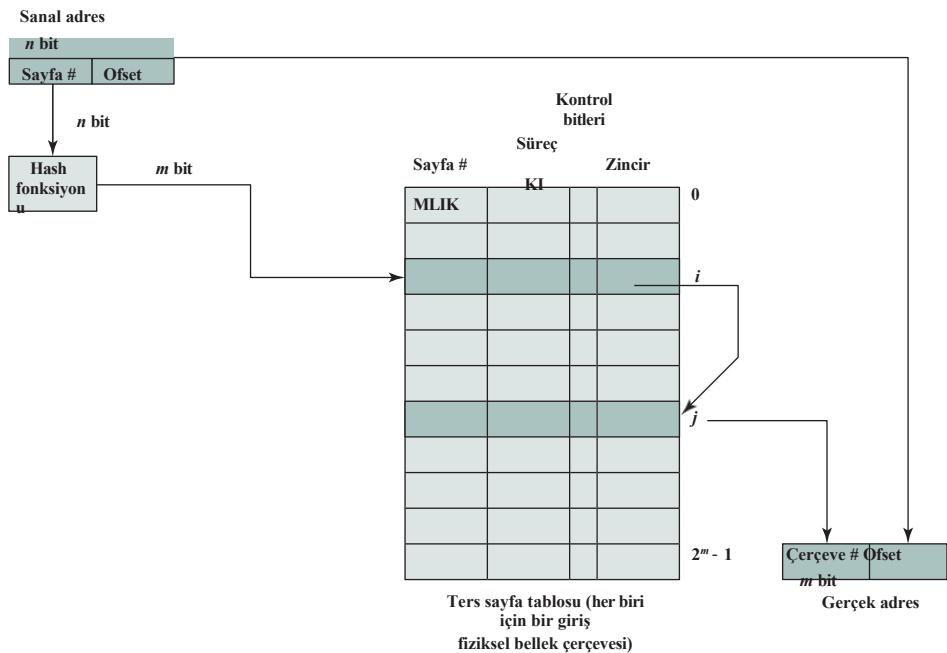
Çoğu sistemde, işlem başına bir sayfa tablosu vardır. Ancak her işlem çok büyük miktarlarda sanal bellek kullanabilir. Örneğin, VAX mimarisinde, her işlem  $2^{31} = 2$  Gbyte'a kadar sanal belleğe sahip olabilir.  $2^9 = 512$  baytlık sayfalar kullanıldığında, bu *işlem başına*  $2^{22}$  kadar sayfa tablosu girişü gerektiği anlamına gelir. Açıkçası, yalnızca sayfa tablolarına ayrılan bellek miktarı kabul edilemeyecek kadar yüksek olabilir. Bu sorunun üstesinden gelmek için, çoğu sanal bellek şeması sayfa tablolarını gerçek bellek yerine sanal bellekte saklar. Bu, sayfa tablolarının da diğer sayfalar gibi sayfalama tabi olduğu anlamına gelir. Bir süreç çalışırken, sayfa tablosunun en azından bir kısmı, o anda yürütülen sayfanın sayfa tablosu girişini de dahil olmak üzere ana bellekte olmalıdır. Bazı işlemciler büyük sayfa tablolarını düzenlemek için iki seviyeli bir şema kullanır. Bu şemada, her girişin bir sayfa tablosuna işaret ettiği bir sayfa dizini vardır. Böylece, sayfa dizininin uzunluğu  $X$  ise ve bir sayfa tablosunun maksimum uzunluğu  $Y$  ise, bir işlem en fazla  $X * Y$  sayfadan oluşabilir. Tipik olarak, bir sayfa tablosunun maksimum uzunluğu bir sayfaya eşit olacak şekilde sınırlanır. Bu bölümün ilerleyen kısımlarında Intel x86'yi ele aldığımızda bu iki seviyeli yaklaşımın bir örneğini göreceğiz.

Bir veya iki seviyeli sayfa tablolarının kullanımına alternatif bir yaklaşım da ters çevrilmiş sayfa tablosu yapısının kullanılmasıdır (Şekil 8.17). Bu yaklaşımın varyasyonları PowerPC, UltraSPARC ve IA-64 mimarisinde kullanılmaktadır. Mach OS'nin RT-PC üzerindeki bir uygulaması da bu teknigi kullanmaktadır.

Bu yaklaşımında, bir sanal adresin sayfa numarası kısmı basit bir hashing fonksiyonu kullanılarak bir hash değerine eşlenir.<sup>2</sup> Hash değerisayfa tablosu girişlerini içeren ters çevrilmiş sayfa tablosuna bir işaretçidir. Sayfa tablosunda bir giriş vardır

---

<sup>2</sup>Bir hash fonksiyonu 0 ile  $M$  aralığındaki sayıları 0 ile  $N$  aralığındaki sayılarla eşler, burada  $M \geq N$ . Hash fonksiyonunun çıktıtı hash tablosunda bir indeks olarak kullanılır. Birden fazla girdi aynı çıktıya eşlenildiğinden, bir girdi öğesinin zaten dolu olan bir hash tablosu girdisine eşlenmesi mümkündür. Bu durumda, yeni öğe başka bir hash tablosu konumuna taşmalıdır. Tipik olarak, yeni öğe sonraki ilk boş alana yerleştirilir ve girişleri birbirine zincirlemek için orijinal konumdan bir işaretçi sağlanır. Hash fonksiyonları hakkında daha fazla bilgi için Ek L'ye bakın.

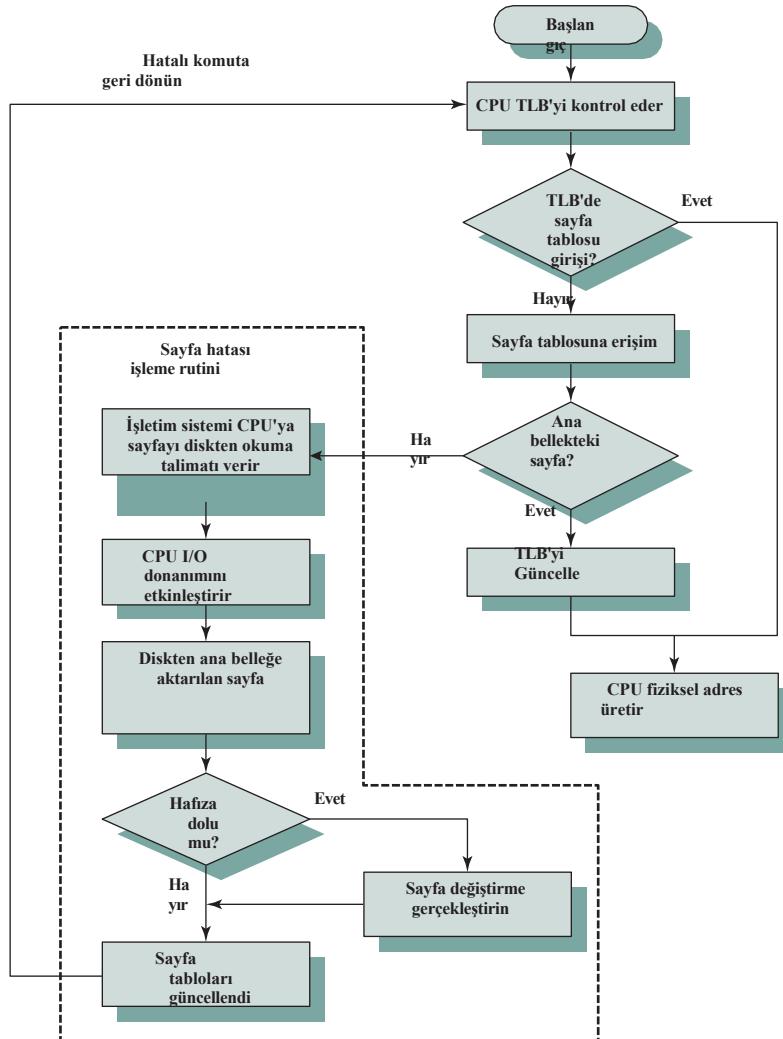


**Sekil 8.17** Ters Sayfa Tablo Yapısı

sanal sayfa başına bir tane yerine her gerçek bellek sayfası çerçevesi için ters çevrilmiş sayfa tablosu. Böylece desteklenen işlem veya sanal sayfa sayısından bağımsız olarak tablolar için sabit oranda gerçek bellek gereklidir. Birden fazla sanal adres aynı hash tablosu girişine eşlenebileceğinden, taşımayı yönetmek için bir zincirleme tekniği kullanılır. Karma tekniği, genellikle bir İLA iki giriş arasında kısa olan zincirlerle sonuçlanır. Sayfa tablosunun yapısı *ters çevrilmiş* olarak adlandırılır çünkü sayfa tablosu girişlerini sanal sayfa numarası yerine çerçeve numarasına göre indeksler.

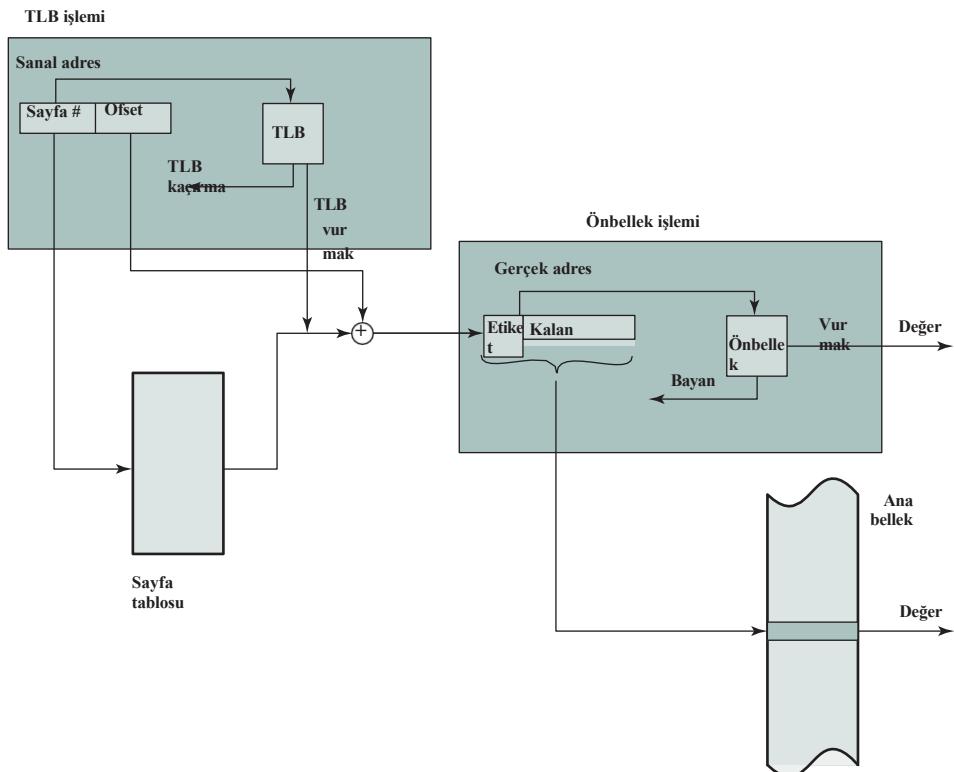
Çeviri Lookaside Tamponu

Prensip olarak, her sanal bellek referansı iki fiziksel bellek erişimine neden olabilir: biri uygun sayfa tablosu girişini almak için, diğeri de istenen veriyi almak için. Bu nedenle, basit bir sanal bellek şeması bellek erişim süresini iki katına çıkarma etkisine sahip olacaktır. Bu sorunun **üstesinden** gelmek için çoğu sanal bellek şeması, sayfa tablosu girişleri için genellikle **çeviri ara belleği (TLB)** adı verilen özel bir önbellek kullanır. Bu önbellek, bellek önbelleği ile aynı şekilde çalışır ve en son kullanılan sayfa tablosu girdilerini içerir. Şekil 8.18 TLB'nin kullanımını gösteren birakis şemasıdır. Yerelilik ilkesi gereği, sanal bellek referanslarının çoğu son kullanılan sayfalardaki konumlara olacaktır. Bu nedenle, referansların çoğu önbellekteki sayfa tablosu girişlerini içerecektir. VAX TLB üzerinde yapılan çalışmalar, bu şemanın performansı ölçüde artırabileceğini göstermiştir [CLAR85, SATY81].



Sanal bellek mekanizmasının önbellek sistemiyle (TLB değil, ana bellek ) etkileşime girmesi gerektiğini unutmayın. Bu Şekil 8.19'da gösterilmiştir. Bir sanal adres genellikle bir sayfa numarası, ofset şeklinde olacaktır. İlk olarak, bellek sistemi eşleşen sayfa tablosu girişinin mevcut olup olmadığını görmek için TLB'ye danışır. Eğer , gerçek (fiziksel) adres, çerçeve numarası ile ofset birləşdirilerek oluşturulur. Değilse, girişe sayfa tablosundan erişilir. Bir etiket ve bir kalan şeklinde olan gerçek adres oluşturulduktan sonra, bu kelimeyi içeren bloğun mevcut olup olmadığını görmek için önbelleğe başvurulur (bkz. Şekil 4.5). Eğer , işlemciye geri gönderilir. Değilse, kelime ana bellekten alınır.

Okuyucu, tek bir bellek referansında yer alan işlemci donanımının karmaşıklığını takdir edebilmelidir. Sanal adres gerçek adrese çevrilir. Bu işlem, TLB'de bulunabilecek bir sayfa tablosuna başvurmayı içerir.



Şekil 8.19 Translation Lookaside Buffer ve Cache İşlemi

ana bellekte veya diskte olabilir. Başvurulan sözcük önbellekte, ana bellekte veya diskte olabilir. İkinci durumda, sözcüğü içeren sayfa ana belleğe yüklenmeli ve bloğu önbelleğe yüklenmelidir. Ayrıca, bu sayfa için sayfa tablosu girişü güncellenmelidir.

### Segmentasyon

Adreslenebilir belleğin alt böülümlere ayrılması *segmentasyon* olarak bilinen bir başka yolu daha vardır. Sayfalama programcı tarafından görülmez ve programcıya daha geniş bir adres alanı sağlama amacıyla hizmet ederken, bölümleme genellikle programcı tarafından görülebilir ve programları ve verileri düzenlemek için bir kolaylık olarak ve ayıralık ve koruma niteliklerini talimatlar ve verilerle ilişkilendirmek için bir araç olarak sağlanır.

Segmentasyon, programının belleği birden fazla adres alanı veya segmentten oluşuyormuş gibi görmesini sağlar. Segmentler değişken, hatta dinamik boyuttadır. Tipik olarak, programcı ya da işletim sistemi programları ve verileri farklı segmentlere atayacaktır. Çeşitli program türleri için bir dizi program segmentinin yanı sıra bir dizi veri segmenti de olabilir. Her segmente erişim ve kullanım hakları atanabilir. Bellek referansları bir (segment numarası, offset) adres biçiminden oluşur.

Bu organizasyon programcı için bölümlenmemiş adres alanına göre bir dizi avantajı vardır:

1. Büyüyen veri yapılarının işlenmesini kolaylaştırır. Programcı belirli bir veri yapısının ne kadar büyüyeceğini önceden bilmeyip, tahmin etmesi gerekmeyez. Veri yapısına kendi segmenti atanabilir ve işletim sistemi segmenti gerektiği gibi genişletir veya daraltır.
2. Bütün bir program setinin yeniden bağlanması ve yeniden yüklenmesine gerek kalmadan programların bağımsız olarak değiştirilmesine ve yeniden derlenmesine olanak tanır. Bu da yine çoklu segmentler kullanılarak gerçekleştirilir.
3. Süreçler arasında paylaşılabilir. Bir programcı, diğer süreçler tarafından ele alınabilecek bir segmente bir yardımcı program veya yararlı bir veri tablosu yerleştirebilir.
4. Kendini korumaya borçludur. Bir segment iyi tanımlanmış dizi program veya veri içerecek şekilde oluşturulduğundan, programcı veya sistem yöneticisi erişim ayrıcalıklarını uygun bir şekilde atayabilir.

Bu avantajlar, programcı tarafından görülemeyen sayfalama mevcut değildir. Öte yandan, sayfalamanın verimli bir bellek yönetimi biçimini sağladığını gördük. Her ikisinin avantajlarını birebirleştirmek için, bazı sistemler her ikisini de sağlayacak donanım ve işletim sistemi yazılımı ile donatılmıştır.

## 8.4 INTEL x86 BELLEK YÖNETİMİ

32-bit mimarisinin başlanmasıından bu yana, mikroişlemciler orta ve büyük ölçekli sistemlerde öğrenilen dersler üzerine inşa edilen sofistike bellek yönetim şemaları geliştirmiştir. Birçok durumda, mikroişlemci sürümleri daha büyük sistem öncülerinden daha üstündür. Şemalar mikroişlemci donanım satıcıları tarafından geliştirildiğinden ve çeşitli işletim sistemlerinde kullanılıldığından, oldukça genel amaçlı olma eğilimindedirler. Temsili bir örnek Intel x86 mimarisinde kullanılan şemadır.

### Adres Alanları

X86 hem segmentasyon hem de sayfalama için donanım içerir. Her iki mekanizma da devre dışı bırakılabilir ve kullanıcının dört farklı bellek görünümünden birini seçmesine olanak tanır:

- **Bölümlememmiş sayfalanmamış bellek:** Bu durumda, sanal adres fiziksel adresle aynıdır. Bu, örneğin düşük karmaşıklıkta, yüksek performanslı denetleyici uygulamalarında kullanılmıştır.
- **Bölümlememmiş sayfalı bellek:** Burada bellek, sayfalanan doğrusal bir adres alanı olarak görülür. Belleğin korunması ve yönetimi sayfalama yoluyla yapılır. Bu, bazı işletim sistemleri tarafından tercih edilir (örneğin, Berkeley UNIX).
- **Bölümlere ayrılmış sayfalanmamış bellek:** Burada bellek, mantıksal adres alanlarının bir koleksiyonu olarak görülür. Bu görüşün sayfalı yaklaşma göre avantajı, gerektiğinde tek bir bayt seviyesine kadar koruma sağlamasıdır. Ayrıca, sayfalamaadan farklı olarak, segment bellekteyken ihtiyaç duyulan çeviri tablosunun (segment tablosu) çip üzerinde olmasını garanti eder. Bu nedenle, bölümlere ayrılmış sayfalanmamış bellek öngörülebilir erişim süreleri sağlar.

- **Bölümlemiş sayfallanmış bellek:** Segmentasyon, erişim kontrolüne tabi mantıksal bellek bölgelerini tanımlamak için kullanılır ve sayfalama, bölgeler içindeki bellek tahsisini yönetmek için kullanılır. UNIX System V gibi işletim sistemleri bu görüşü desteklemektedir.

## Segmentasyon

Segmentasyon kullanıldığında, her sanal adres (x86 belgelerinde mantıksal adres olarak adlandırılır) 16 bitlik bir segment referansı ve 32 bitlik bir ofsetten oluşur. Segment referansının iki biti koruma mekanizması ile ilgilidir ve belirli bir segmenti belirtmek için 14 bit bırakır. Böylece, bölümlememiş bellekte, kullanıcının sanal belleği  $2^{32} = 4$  Gbyte'tır. Bölümlere ayrılmış bellek ile, bir kullanıcı tarafından görülen toplam sanal bellek alanı  $2^{46} = 64$  terabayttır (Tbytes). Fiziksel adres alanı maksimum 4 Gbyte için 32 bitlik bir adres kullanır.

Sanal bellek miktarıaslın da 64 Tbyte'tan daha büyük olabilir. Bunun nedeni, işlemcinin bir sanal adresi yorumlamasının o anda hangi işlemin etkin olduğunu bağlı olmasıdır. Sanal adres alanı iki ayrılmıştır. Sanal adres alanının yarısı (8K segment \* 4 Gbyte) globaldir, tüm işlemler tarafından paylaştırılır; geri kalanı yereldir ve her işlem için ayrıdır.

Her segmentle ilişkili iki koruma biçimi vardır: ayrıcalık düzeyi ve erişim niteliği. En çok korunandan (seviye 0) en az korunana (seviye 3) kadar dört ayrıcalık seviyesi vardır. Bir veri segmenti ile ilişkilendirilen ayrıcalık seviyesi onun "sınıflandırması"; bir program segmenti ile ilişkilendirilen ayrıcalık seviyesi ise onun "açıklığı"dır. Exe-kesen bir program, yalnızca açıklık seviyesi veri segmentinin ayrıcalık seviyesinden daha düşük (daha ayrıcalıklı) veya eşit (aynı ayrıcalık) olan veri erişebilir.

Donanım bu ayrıcalık seviyelerinin nasıl kullanılacağını belirlemek; bu işletim sistemi tasarımasına ve uygulamasına bağlıdır. Ayrıcalık seviyesi 1'in işletim sisteminin çoğu için kullanılması ve seviye 0'in işletim sisteminin bellek yönetimi, koruma ve erişim kontrolüne ayrılmış küçük bir kısmı için kullanılması amaçlanmıştır. Bu da uygulamalar için iki seviye bırakmaktadır. Birçok sistemde, uygulamalar 3. seviyede yer alacak ve 2. seviye kullanılmayacaktır. Kendi güvenlik mekanizmalarını uyguladıkları için korunmalari gereken özel uygulama alt sistemleri 2. seviye için iyi adaylardır. Bazı örnekler veritabanı yönetim sistemleri, ofis otomasyon sistemleri ve yazılım mühendisliği ortamlarıdır.

Veri segmentlerine erişimi düzenlemenin yanı sıra, ayrıcalık mekanizması belirli talimatların kullanımını da sınırlar. Bellek yönetimi kayıtları ile ilgili olanlar gibi bazı talimatlar yalnızca seviye 0'da yürütülebilir. G/C talimatları yalnızca işletim sistemi tarafından belirlenen belirli bir seviyeye kadar çalıştırılabilir; tipik olarak bu seviye 1 olacaktır. Bir veri segmentinin erişim niteliği, okuma/yazma ya da okuma-yazma olup olmadığını belirtir. yalnızca erişimlere izin verilir. Program segmentleri için erişim özniteliği okuma/çalıştırma veya salt okuma erişimini belirtir.

Segmentasyon için adres çeviri mekanizması, sanal bir adresin doğrusal adres olarak adlandırılan bir adresle eşleştirilmesini içerir (Şekil 8.20b). Bir sanal adres 32 bitlik ofset ve 16 bitlik segment seçiciden oluşur (Şekil 8.20a). Bir işleneni getiren veya depolayan bir komut, ofseti ve segment içeren bir kaydı belirtir. Segment seçici aşağıdaki alanlardan oluşur:

- **Tablo Göstergesi (TI):** Çeviri için global segment tablosunun mu yoksa yerel segment tablosunun mu kullanılması gerektiğini belirtir.

## 306 BÖLÜM 8 / İŞLETİM SİSTEMİ DESTEĞİ

|       |         |
|-------|---------|
| 15    | 3 2 1 0 |
| Dizin | T<br>I  |

TI = Tablo göstergesi

RPL = Talep sahibi ayrıcalık düzeyi

(a) Segment seçici

|        |       |       |   |
|--------|-------|-------|---|
| 31     | 22 21 | 12 11 | 0 |
| Rehber | Tablo | Ofset |   |

(b) Doğrusal adres

|             |             |             |                          |                                 |                                     |
|-------------|-------------|-------------|--------------------------|---------------------------------|-------------------------------------|
| 31          | 24 23 22    | 20 19       | 16 15 14 13 12 11        | 8 7                             | 0                                   |
| Baz 31...24 | G<br>/<br>B | D<br>/<br>L | A<br>V<br>L              | Segment<br>sınırları<br>19...16 | P<br>DPL<br>S<br>Tip<br>Baz 23...16 |
| Baz 15...0  |             |             | Segment sınırları 15...0 |                                 |                                     |

AVL= Sistem yazılımı tarafından kullanılabilir Base

= Segment temel adresi

D/B = Varsayılan işlem boyutu DPL =

Tanımlayıcı ayrıcalık boyutu G

= Granülerlik

L = 64 bit kod segmenti  
(yalnızca 64 bit modu)

P = Segment mevcut

Türü = Segment türü

S = Tanımlayıcı tipi

(c) Segment tanımlayıcısı (segment tablosu girişi)

|                                |         |       |                  |                                           |
|--------------------------------|---------|-------|------------------|-------------------------------------------|
| 31                             | 12 11 9 | 7 6 5 | 4 3 2 1 0        |                                           |
| Sayfa çerçevesi adresi 31...12 |         | AVL   | P<br>S<br>0<br>A | P<br>C<br>W<br>T<br>U<br>S<br>R<br>W<br>P |

AVL= Sistem programcısı kullanımı için mevcuttur P

= Sayfa boyutu

A = Erişilen PCID=

Önbellek devre dışı

PWT= US üzerinden yazma= Kullanıcı/denetmen

■ = Ayrılmış

RW = Okuma-yazma

P = Mevcut

(d) Sayfa dizini girişi

|                                |         |       |           |                                                |
|--------------------------------|---------|-------|-----------|------------------------------------------------|
| 31                             | 12 11 9 | 7 6 5 | 4 3 2 1 0 |                                                |
| Sayfa çerçevesi adresi 31...12 |         | AVL   | D<br>A    | P<br>C<br>D<br>W<br>T<br>U<br>S<br>R<br>W<br>P |

D = Kirli

(e) Sayfa tablosu girişi

**Şekil 8.20** Intel x86 Bellek Yönetimi Formatları

■ **Segment Numarası:** Segmentin numarası. Bu, segment tablosunda bir indeks görevi görür.

■ **Talep Edilen Ayrıcalık Düzeyi (RPL):** Bu erişim için talep edilen ayrıcalık düzeyi.

Bir segment tablosundaki her giriş Şekil 8.20c'de gösterildiği gibi 64 bitten oluşur. Alanlar Tablo 8.5'te tanımlanmıştır.

**Tablo 8.5** x86 Bellek Yönetimi Parametreleri

| Segment Tanımlayıcısı (Segment Tablosu Giriş) |                                                                                                                                                                                                                                                    |
|-----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Üs</b>                                     | 4-Gbyte doğrusal adres alanı içindeki segmentin başlangıç adresini tanımlar.                                                                                                                                                                       |
| <b>D/B biti</b>                               | Bir kod segmentinde, bu D bitidir ve işlenenlerin ve adresleme modlarının 16 veya 32 olup olmadığını gösterir.                                                                                                                                     |
| <b>Tanımlayıcı Ayrıcalık Düzeyi (DPL)</b>     | Bu segment tanımlayıcısı tarafından başvurulan segmentin ayrıcalık düzeyini belirtir.                                                                                                                                                              |
| <b>Granüllerlik biti (G)</b>                  | Limit alanının bir baytılık veya 4 Kbaytılık birimler halinde yorumlanacağını belirtir.                                                                                                                                                            |
| <b>Limit</b>                                  | Segmentin boyutunu tanımlar. İşlemci limit alanını granularity bitine bağlı olarak iki şekilde yorumlar: 1 Mbyte segment boyutu limitine kadar byte'lık birimler halinde veya 4 Gbyte segment boyutu limitine kadar 4 Kbyte'lık birimler halinde.  |
| <b>S bit</b>                                  | Belirli bir segmentin bir sistem segmenti mi yoksa bir kod veya veri mi olduğunu belirler.                                                                                                                                                         |
| <b>Segment Mevcut biti (P)</b>                | Sayfalanmamış sistemler için kullanılır. Segmentin ana bellekte bulunup bulunmadığını gösterir. Sayfalı sistemler için bu bit her zaman 1 olarak ayarlanır.                                                                                        |
| <b>Tip</b>                                    | Çeşitli segment türleri arasında ayırm yapar ve erişim niteliklerini belirtir.                                                                                                                                                                     |
| Sayfa Dizini Giriş ve Sayfa Tablosu Giriş     |                                                                                                                                                                                                                                                    |
| <b>Erişilen bit (A)</b>                       | Bu bit, ilgili sayfaya bir okuma veya yazma işlemi gerçekleştiğinde işlemci tarafından sayfa tablolarının her iki seviyesinde de 1'e ayarlanır.                                                                                                    |
| <b>Kırkı bit (D)</b>                          | Bu bit, ilgili sayfaya bir yazma işlemi gerçekleştiğinde işlemci tarafından 1'e ayarlanır.                                                                                                                                                         |
| <b>Sayfa Çerçevesi Adresi</b>                 | Mevcut biti ayarlanmışsa, sayfanın bellekte fiziksel adresini sağlar. Sayfa çerçeveleri 4K sınırlarında hizalandığından, alt 12 bit 0'dır ve yalnızca üst 20 bit giriş dahil edilir. Bir sayfa yönergesinde, adres bir sayfa tablosunun adresidir. |
| <b>Sayfa Önbellege Devre Dışı Biti (PCD)</b>  | Sayfadaki verilerin önbellege alınıp alınamayacağını belirtir.                                                                                                                                                                                     |
| <b>Sayfa Boyutu biti (PS)</b>                 | Sayfa boyutunun 4 Kbyte mi yoksa 4 Mbyte mi olduğunu belirtir.                                                                                                                                                                                     |
| <b>Sayfa Yazma Biti (PWT)</b>                 | İlgili sayfadaki veriler için write-through veya write-back önbellege alma ilkesinin kullanılacağını belirtir.                                                                                                                                     |
| <b>Mevcut bit (P)</b>                         | Sayfa tablosunun veya sayfanın ana bellekte olup olmadığını gösterir.                                                                                                                                                                              |
| <b>Okuma/Yazma biti (RW)</b>                  | Kullanıcı düzeyindeki sayfalar için, sayfanın kullanıcı düzeyindeki programlar için salt okunur erişimli mi yoksa okuma/yazma erişimli mi olduğunu belirtir.                                                                                       |
| <b>Kullanıcı/Süpervizör biti (ABD)</b>        | Sayfanın yalnızca işletim sistemi (gözetmen düzeyi) tarafından mı yoksa hem işletim sistemi hem de uygulamalar (kullanıcı düzeyi) tarafından mı kullanılabileceğini belirtir.                                                                      |

### Çağrı

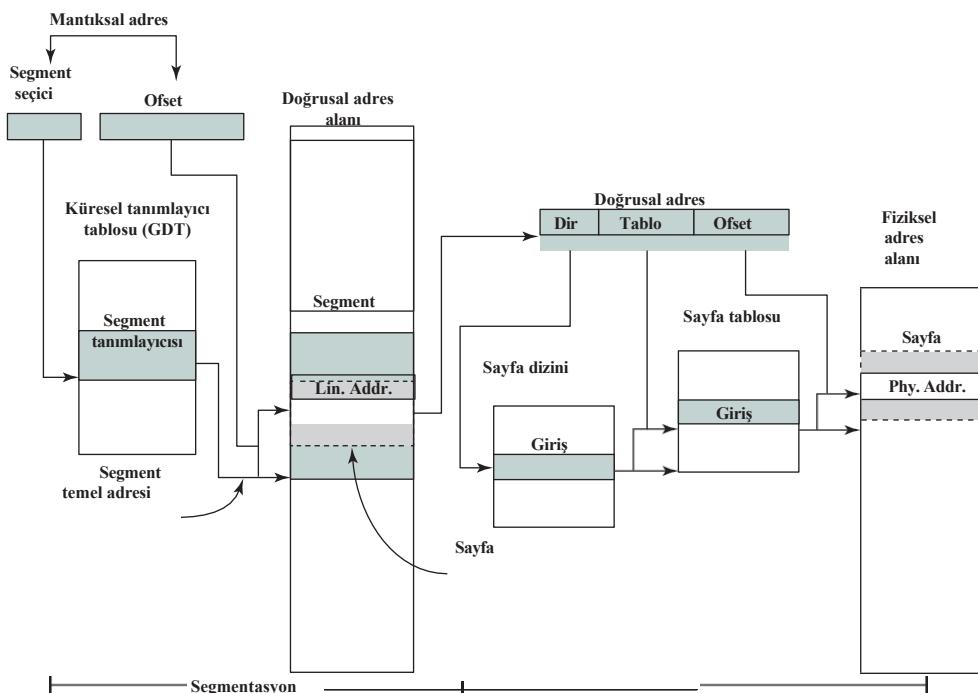
Segmentasyon isteğe bağlı bir özelliktir ve devre dışı bırakılabilir. Segmentasyon kullanıldığında, programlarda kullanılan adresler sanal adreslerdir ve az önce açıklandığı gibi doğrusal adreslere dönüştürülür. Segmentasyon kullanılmadığında, programlarda doğrusal adresler kullanılır. Her iki durumda da, bir sonraki adım bu doğrusal adresi gerçek bir 32 bit adrese dönüştürmektedir.

Doğrusal adresin yapısını anlamak için, x86 sayfalama mekanizmasının aslında iki seviyeli bir tablo arama işlemi olduğunu bilmeniz gerekmektedir. İlk seviye, en fazla 1024 giriş içeren bir sayfa dizinidir. Bu, 4 Gbyte'lık doğrusal bellek alanını, her biri kendi sayfa tablosuna sahip ve her biri 4 Mbyte uzunluğunda olan 1024 sayfa grubuna böler. Her sayfa tablosu en fazla 1024 giriş içerir; her giriş tek bir 4 Kbyte sayfaya karşılık gelir. Bellek yönetimi, tüm işlemler için tek bir sayfa dizini, her işlem için bir sayfa dizini ya da ikisinin bir kombinasyonunu kullanma seçeneğine sahiptir. Geçerli görev için sayfa dizini her zaman ana bellekteki sayfa tablolari sanal bellekte olabilir.

Şekil 8.20'de sayfa dizinleri ve sayfa tablolardaki girişlerin biçimleri gösterilmektedir ve alanlar Tablo 8.5'te tanımlanmıştır. Erişim kontrol mekanizmalarının sayfa veya sayfa grubu bazında sağlanabileceğini unutmayın.

x86 ayrıca bir çeviri ara belleği de kullanır. Tampon 32 sayfa tablosu girdisi tutabilir. Sayfa dizini her değiştirildiğinde tampon temizlenir.

Şekil 8.21 segmentasyon ve sayfalama mekanizmalarının kombinasyonunu göstermektedir. Anlaşılmamış olması için, çeviri ara belleği ve bellek önbelleği mekanizmaları gösterilmemiştir.



**Şekil 8.21** Intel x86 Bellek Adres Çeviri Mekanizmaları

Son olarak x86, daha önceki 80386 veya 80486'da bulunmayan yeni bir uzanti, iki sayı boyutu hükmü içerir. Kontrol kaydı 4'teki PSE (sayfa boyutu uzantısı) biti 1 olarak ayarlanırsa, disk belleği birimi işletim sistemi programcısının bir sayfayı 4 Kbyte veya 4 Mbyte boyutunda tanımlamasına izin verir.

4 Mbyte'lık sayfalar kullanıldığında, sayfalar için yalnızca tek bir tablo arama düzeyi vardır. Donanım sayfa dizinine eriştiğinde, sayfa dizini girişinde (Şekil 8.20d) PS biti 1 olarak ayarlanır. Bu durumda, 9 ile 21 arasındaki bitler yok sayılır ve 22 ile 31 arasındaki bitler bellekteki 4-Mbyte'lık bir sayfanın temel adresini tanımlar. Dolayısıyla, tek bir sayfa tablosu vardır.

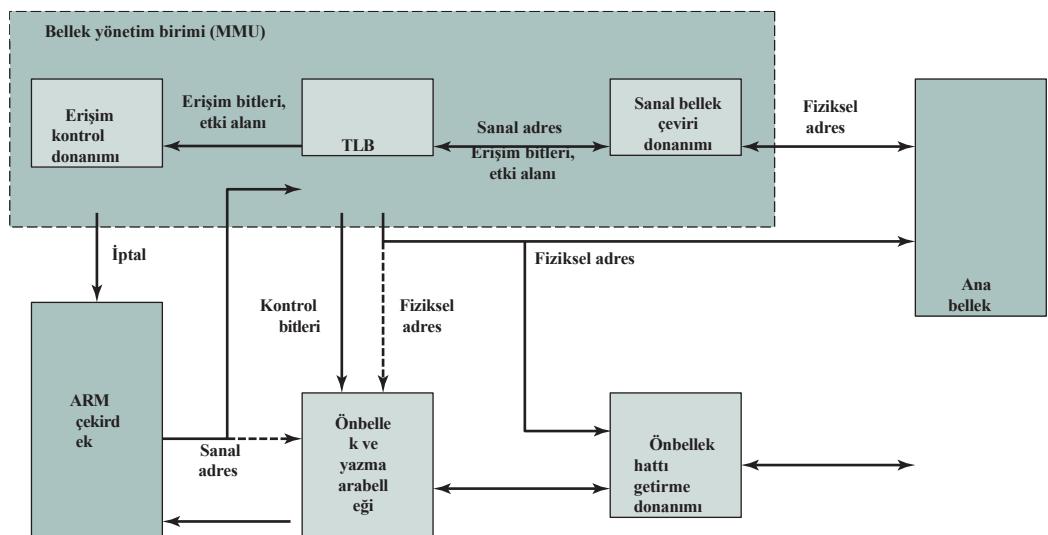
4 Mbyte'lık sayfaların kullanımı, büyük ana için bellek yönetimi depolama gereksinimlerini azaltır. 4-Kbyte sayfalarla, 4-Gbyte'lık tam bir ana bellek sadece sayfa tabloları için yaklaşık 4 Mbyte bellek gerektirir. 4-Mbyte sayfalarla, sayfa bellek yönetimi için 4 Kbyte uzunluğunda tek bir tablo yeterlidir.

## 8.5 ARM MEMOrY YÖNETİMİ

ARM, gömülü sistem tasarımcısının ihtiyaçlarına göre uyarlanabilen çok yönlü bir sanal bellek sistemi mimarisi sağlar.

### Bellek Sistemi Organizasyonu

Şekil 8.22, sanal bellek için ARMdeki bellek yönetim donanımına genel bir bakış sağlar. Sanal bellek çeviri donanımı, daha sonra açıklanacağı gibi, sanal adreslerden fiziksel adreslere çeviri için bir veya iki seviye tablo kullanır. Çeviri ara belleği (TLB), son sayfa tablosu girişlerinin bir önbellegidir. TLB'de bir giriş mevcutsa, TLB bir okuma veya yazma işlemi için doğrudan ana belleğe fiziksel bir adres gönderir. Bölüm 4'te açıkladığı gibi, veri alışverişleri



**Şekil 8.22** ARM Bellek Sistemine Genel Bakış

aracılığıyla işlemci ve ana bellek arasında. Mantıksal bir önbellek organizasyonu kullanılıyorsa (Şekil 4.7a), ARM bu adresi doğrudan önbelleğe sağlar ve bir önbellek özlemi oluştuğunda TLB'ye sağlar. Fiziksnel bir önbellek organizasyonu kullanılırsa (Şekil 4.7b), TLB fiziksnel adresi sağlamalıdır.

Çeviri tablolarındaki girişler, belirli bir işlemin belleğin belirli bir bölümüne erişip erişemeyeceğini belirleyen erişim kontrol bitlerini de içerir. Erişim reddedilirse, erişim kontrol donanımı ARM işlemcisine bir iptal sinyali gönderir.

### **Sanal Bellek Adres Çevirisı**

ARM, bölümlere ya da sayfalara dayalı bellek erişimini destekler:

- **Üst bölmeler (isteğe bağlı):** 16 MB'lık ana bellek bloklarından oluşur.
- **Bölmeler:** 1 MB'lık ana bellek bloklarından oluşur.
- **Büyük sayfalar:** 64 kB'lık ana bellek bloklarından oluşur.
- **Küçük sayfalar:** Ana belleğin 4 kB'lık bloklarından oluşur.

TLB'de yalnızca tek bir giriş kullanırken geniş bir bellek bölgesinin eşlenmesine izin vermek için bölmeler ve üst bölmeler desteklenir. Ek erişim kontrol mekanizmaları küçük sayfalar içinde 1kB alt sayfalara ve büyük sayfalar içinde 16kB alt sayfalara kadar genişletilmiştir. Ana bellekte tutulan çeviri tablosunun iki seviyesi vardır:

- **Seviye 1 tablosu:** Bir Bölüm ve Üst Bölüm için temel adres ve çeviri özelliklerini içeren düzey 1 ve büyük bir sayfa veya küçük bir sayfa için düzey 2 tablosuna çeviri özelliklerini ve işaretçileri tutar.
- **Seviye 2 tablosu:** Bir Küçük sayfa veya bir Büyük sayfa için temel adres ve aktarım özelliklerini içeren düzey 2 tanımlayıcılarını tutar. Seviye 2 tablosu 1 kB bellek gerektirir.

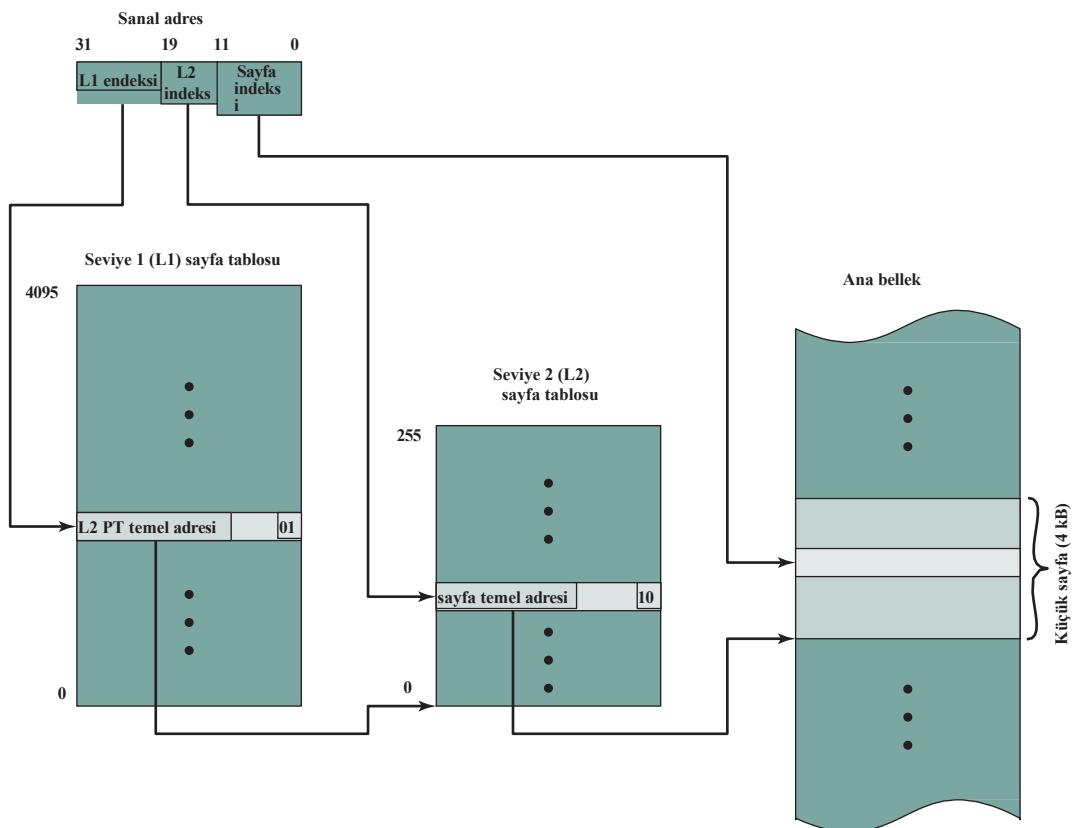
Bellek yönetim birimi (MMU), ana belleğe erişmek için işlemci tarafından oluşturulan sanal adresleri fiziksnel adreslere çevirir ve ayrıca erişim iznini türetir ve kontrol eder. Çeviriler bir TLB iskalamasının sonucu olarak gerçekleşir ve birinci seviye bir getirme ile başlar. Bölüm eşlemeli bir erişim yalnızca birinci seviye bir getirme gerektirirken, sayfa eşlemeli bir erişim ayrıca ikinci seviye bir getirme gerektirir.

Şekil 8.23 küçük sayfalar için iki seviyeli adres çeviri işlemini göstermektedir. 4K 32 bit girişi tek bir seviye 1 (L1) sayfa tablosu vardır. Her bir L1 girişi 256 adet 32 bitlik sahip bir seviye 2 (L2) sayfa tablosuna işaret eder. L2 girişlerinin her biri ana bellekteki 4 kB'lık bir sayfaya işaret eder. 32 bitlik sanal adres aşağıdaki gibi yorumlanır: En anlamlı 12 bit L1 sayfa tablosuna bir indekstir. Sonraki 8 bit ilgili L2 sayfa tablosuna bir indekstir. En az anlamlı 12 bit, ana bellekteki ilgili sayfadaki bir baytı indeksler.

Büyük sayfalar için benzer bir iki sayfalı arama prosedürü kullanılır. Bölmeler ve üst bölmeler için yalnızca L1 sayfa tablosu araması gereklidir.

### **Bellek Yönetimi Formatları**

ARM bellek yönetim şemasını daha iyi anlamak için, Şekil 8.24'te gösterildiği gibi anahtar formatları ele alıyoruz. Bu şekilde gösterilen kontrol bitleri Tablo 8.6'da tanımlanmıştır.



**Şekil 8.23** Küçük Sayfalar için ARM Sanal Bellek Adres Çevirisi

L1 tablosu için her giriş, ilgili 1-MB sanal adres aralığının nasıl eşlendiğinin bir tanımlayıcısıdır. Her giriş dört alternatif biçimden birine sahiptir:

- Bitler [1:0]= 00: İlişkili sanal adresler eşlenmemiştir ve bunlara erişim denemeleri bir çeviri hatası oluşturur.
- [1:0]= 01: Giriş, ilişkili sanal adres aralığının nasıl eşlendiğini belirten bir L2 sayfa tablosunun fiziksel adresini verir.
- [1:0]= 01: ve bit 19= 0: Giriş, ilgili sanal adresler için bir bölüm tanımlayıcısıdır.
- Bit [1:0]= 01: ve bit 19= 1: Giriş, ilişkili sanal adresleri için bir üst bölüm tanımlayıcısıdır.

Bitleri [1:0]= 11 olan girişler ayrılmıştır.

Sayfalar halinde yapılandırılmış bellek için iki seviyeli bir sayfa tablosu erişimi gereklidir. L1 sayfa girişiinin [31:10] bitleri L2 sayfa tablosuna bir işaretçi içerir. Küçük sayfalar için, L2 girişi ana bellekteki 4 kB'lık bir sayfanın taban adresine 20 bitlik bir işaretçi içerir.

Büyük sayfalar için yapı daha karmaşıktır. Küçük sayfalar için sanal adreslerde olduğu gibi, büyük bir sayfa yapısı için sanal adres 12 bitlik bir dizin içerir

## 312 BÖLÜM 8 / İŞLETİM SİSTEMİ DESTEĞİ

|                                                      |                        |                          |                                                                |                              |                         |            |       |     |
|------------------------------------------------------|------------------------|--------------------------|----------------------------------------------------------------|------------------------------|-------------------------|------------|-------|-----|
|                                                      | 31                     | 24 23                    | 20 19                                                          | 14                           | 12 11 10 9 8            | 5 4 3      | 2 1 0 |     |
| Ariza                                                |                        |                          |                                                                | IGN                          |                         |            | 0 0   |     |
| Sayfa tablo                                          |                        |                          | Kaba sayfa tablosu taban adresi                                |                              | P                       | Etki Alanı | SBZ   | 0 1 |
| Bölüm                                                |                        | Bölüm temel adresi       | S<br>B<br>Z<br>0<br>n<br>G<br>S<br>AP<br>X<br>TEX<br>AP<br>P   | Etki Alanı                   | X<br>N<br>C<br>B<br>1 0 |            |       |     |
| Süper Bölüm                                          | Üst bölüm temel adresi | Temel adres [35:32]      | S<br>B<br>Z<br>1<br>n<br>G<br>S<br>AP<br>X<br>TEX<br>AP<br>P   | Temel adres [39:36]          | X<br>N<br>C<br>B<br>1 0 |            |       |     |
| (a) Alternatif birinci seviye tanımlayıcı formatları |                        |                          |                                                                |                              |                         |            |       |     |
|                                                      | 31                     |                          | 16 15 14                                                       | 12 11 10 9 8 7 6 5 4 3 2 1 0 |                         |            |       |     |
| Ariza                                                |                        |                          | IGN                                                            |                              |                         |            | 0 0   |     |
| Küçük sayfa                                          |                        | Küçük sayfa temel adresi | n<br>G<br>S<br>AP<br>X<br>TEX<br>AP<br>C<br>B<br>1 X N         |                              |                         |            |       |     |
| Büyük sayfa                                          |                        | Büyük sayfa temel adresi | X<br>N<br>n<br>G<br>S<br>AP<br>X<br>SBZ<br>AP<br>C<br>B<br>0 1 |                              |                         |            |       |     |
| (b) Alternatif ikinci seviye tanımlayıcı formatları  |                        |                          |                                                                |                              |                         |            |       |     |
|                                                      | 31                     | 24 23                    | 20 19                                                          |                              |                         |            | 0     |     |
| Süper Bölüm                                          | Seviye 1 tablo dizini  |                          |                                                                | Üst bölüm indeksi            |                         |            |       |     |
|                                                      | 31                     |                          | 20 19                                                          |                              |                         |            | 0     |     |
| Bölüm                                                | Seviye 1 tablo dizini  |                          |                                                                | Bölüm indeksi                |                         |            |       |     |
|                                                      | 31                     |                          | 20 19                                                          | 12 11                        |                         |            | 0     |     |
| Küçük sayfa                                          | Seviye 1 tablo dizini  |                          | Seviye 2 tablo dizini                                          |                              | Sayfa indeksi           |            |       |     |
|                                                      | 31                     | 20 19                    | 16 15                                                          | 12 11                        |                         |            | 0     |     |
| Büyük sayfa                                          | Seviye 1 tablo dizini  | Seviye 2 tablo indeksi   |                                                                |                              | Sayfa indeksi           |            |       |     |
| (c) Sanal bellek adres biçimleri                     |                        |                          |                                                                |                              |                         |            |       |     |

Şekil 8.24 ARM Bellek Yönetimi Formatları

birinci seviye tablo ve L2 8 bitlik bir indeks. 64 kB'lık büyük sayfalar için sanal adresin sayfa indeksi kısmı 16 bit olmalıdır. Tüm bu bitleri 32 bitlik bir formatta barındırmak için, sayfa indeksi alanı ile L2 tablosu indeksi alanı arasında 4 bitlik bir örtüşme vardır. ARM bu çakışmayı, büyük sayfaları destekleyen bir L2 sayfa tablosundaki her sayfa tablosu girişinin 16 kez çoğaltılmasını gerektirerek sağlar. Gerçekte, tüm girişler büyük sayflara atıfta bulunuyorsa, L2 sayfa tablosunun boyutu 256 girişten 16 girişe düşürülür. Ancak, belirli bir L2 sayfası büyük ve küçük sayfaların bir karışımına hizmet edebilir, bu nedenle büyük sayfa girişleri için çoğaltma ihtiyacı vardır.

**Tablo 8.6** ARM Bellek Yönetimi Parametreleri

|                                                                                                                                                                                                                              |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Erişim İzni (AP), Erişim İzni Uzantısı (APx)</b>                                                                                                                                                                          |
| Bu bitler ilgili bellek bölgесine erişimi kontrol eder. Gerekli izinler olmadan bir bellek alanına erişim yapılrsa, bir İzin Hatası oluşur.                                                                                  |
| <b>Arabelleğe alınabilir (B) biti</b>                                                                                                                                                                                        |
| TEX bitleri ile yazma tamponunun önbelleklenebilir bellek için nasıl kullanılacağını belirler.                                                                                                                               |
| <b>Önbelleklenebilir (C) biti</b>                                                                                                                                                                                            |
| Bu bellek bölgesinin önbellek aracılığıyla eşlenip eşlenemeyeceğini belirler.                                                                                                                                                |
| <b>Etki Alanı</b>                                                                                                                                                                                                            |
| Bellek bölgelerinin toplanması. Erişim kontrolü etki alanı temelinde uygulanabilir.                                                                                                                                          |
| <b>Küresel Değil (nG)</b>                                                                                                                                                                                                    |
| Cevirinin genel (0) veya sürece özel (1) olarak işaretlenip işaretlenmeyeceğini belirler.                                                                                                                                    |
| <b>Paylaşilan (S)</b>                                                                                                                                                                                                        |
| Cevirinin paylaşılmayan (0) ya da paylaşılan (1) bellek için olup olmadığını belirler.                                                                                                                                       |
| <b>SBZ</b>                                                                                                                                                                                                                   |
| Sıfır olmalı.                                                                                                                                                                                                                |
| <b>Tip Uzatma (TEx)</b>                                                                                                                                                                                                      |
| Bu bitler, B ve C bitleriyle birlikte, önbelleklere erişimi, yazma arabelleğinin nasıl kullanıldığını ve bellek bölgesinin paylaşılabilir olup olmadığını ve bu nedenle tutarlı tutulması gerekip gerekmedinin kontrol eder. |
| <b>Asla Yürütme (xN)</b>                                                                                                                                                                                                     |
| Bölgemin çalıştırılabilir (0) veya çalıştırılamaz (1) olup olmadığını belirler.                                                                                                                                              |

Bölümler veya üst bölümler halinde yapılandırılmış bellek için tek seviyeli bir sayfa tablosu erişimi gereklidir. Bolumler için, L1 girişinin [31:20] ana bellekteki 1 MB'lık bölümün tabanına 12 bitlik bir işaretçi içerir.

Üst bölümler için, L1 girişinin [31:24] bitleri ana bellekteki 16 MB'lık bölümün tabanına 8 bitlik bir işaretçi içerir. Büyük sayfalarda olduğu gibi, sayfa tablosu girişinin çoğaltıması gereklidir. Üst bölümler söz konusu olduğunda, sanal adresin L1 tablosu dizin kısmı, sanal adresin üst bölüm dizin kısmı ile 4 bit örtüsü Bu nedenle, 16 özdeş L1 sayfa tablosu girişü gereklidir.

Fiziksel adres alanı aralığı sekiz adede kadar ek adres biti ile genişletilebilir (bitler [23:20] ve [8:5]). Ek bitlerin sayısı uygulamaya bağlıdır. Bu ek bitler fiziksel belleğin boyutunu  $2^8 = 256$  katına kadar genişlettiği şekilde yorumlanabilir. Dolayısıyla, fiziksel bellek aslında her bir işlem için mevcut bellek alanının 256 katı kadar büyük olabilir.

## Erişim Kontrolü

Her tablo girişindeki AP erişim kontrol bitleri, belirli bir işlem tarafından bir bellek bölgесine erişimi kontrol eder. Bir bellek bölgesi erişim yok, yalnızca okunur veya okunur-yazılır olarak belirlenebilir. Ayrıca, bölge yalnızca ayrıcalıklı erişim olarak belirlenebilir, uygulamalar tarafından değil işletim sistemi tarafından kullanılmak üzere ayrılabilir.

ARM ayrıca, belirli erişim izinlerine sahip bölümler ve/veya sayfalar topluluğu olan bir etki alanı kavramını da kullanır. ARM mimarisi

16 etki alanını destekler. Etki alanı özelliği, birden fazla işlemin aynı çeviri tablolarını kullanmasına izin verirken, birbirlerinden bir miktar koruma sağlar.

Her sayfa tablosu girişi ve TLB girişi, hangi etki alanında olduğunu belirten bir alan içerir. Etki Alanı Erişim Kontrol Kaydındaki 2 bitlik bir alan her bir etki alanına erişimi kontrol eder. Her alan, tüm bir alana erişimin çok hızlı bir şekilde etkinleştirilmesine devre dışı bırakılmasına olanak tanır, böylece tüm bellek alanları sanal belleğe çok verimli bir şekilde takas edilebilir. İki tür etki alanı erişimi desteklenir:

- **İstemciler:** Etki alanlarını oluşturan bölümerni ve/veya sayfaların erişim izinlerine uyması gereken etki alanı kullanıcıları (programları çalıştırıp ve verilere erişen).
- **Yöneticiler:** Etki alanının davranışını (etki alanındaki geçerli bölüm ve sayfalar ile etki alanı erişimi) kontrol eder ve bu etki alanındaki tablo girişleri için erişim izinlerini atlar.

Bir program bazı etki alanlarının istemcisini ve diğer bazı etki alanlarının yönetici olabilir ve geri kalan etki alanlarına erişimi olmamayı da. Bu, farklı bellek kaynaklarına erişen programlar için çok esnek bellek koruması sağlar.

## 8.6 ANAHTAR TERİMLER, SORULAR VE ÖNERİLER

### Anahtar Terimler

|                                                                                                                                                                                                                                  |                                                                                                                                                                          |                                                                                                                                                               |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| toplu sistem talep çağrıları<br>etkileşimli işletim sistemi kesintisi<br>iş kontrol dili (JCL) çekirdeği<br>mantıksal adres<br>uzun vadeli programlama orta vadeli programlama bellek yönetimi bellek koruması çoklu programlama | çok görevli çekirdek işletim sistemi (OS) sayfa tablosu disk belleği bölümleme fiziksel adresi ayrıcalıklı talimat süreci süreç kontrol bloğu süreç durumu gerçek hafıza | yerleşik monitör segmentasyonu kısa vadeli program değiştirme thrashing zaman paylaşımı sistem çevirisü lookaside tampon (TLB) yardımcı programı sanal bellek |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|

### İnceleme Soruları

- 8.1 İşletim sistemi nedir?
- 8.2 Bir işletim sistemi tarafından sağlanan temel hizmetleri listeleyiniz ve kısaca tanımlayınız.
- 8.3 Başlıca işletim sistemi zamanlama türlerini listeleyin ve kısaca tanımlayın.
- 8.4 Süreç ve program arasındaki fark nedir?
- 8.5 Takasın amacı nedir?
- 8.6 Eğer bir süreç dinamik olarak ana bellekte farklı konumlara atanabiliyorsa, adresleme mekanizması için bunun anlamı nedir?
- 8.7 Süreç yürütülürken bir sürecin tüm sayfalarının ana bellekte olması gereklidir?

- 8.8** Ana bellekteki bir işlemin sayfaları bitişik mi olmalıdır?
- 8.9** Ana bellekteki bir işlemin sayfalarının olması gerekli midir?
- 8.10** Çeviri ara tamponunun amacı nedir?

## Problemler

- 8.1** Her bir işin aynı özelliklere sahip olduğu çok programlı bir bilgisayarımız olduğunu varsayıyalım. Bir iş için bir hesaplama periyodunda,  $T$ , zamanın yarısı I/O'da ve diğer yarısı işlemci faaliyetinde harcanır. Her iş toplam  $N$  dönem boyunca çalışır. Basit bir round-robin önceligi kullanıldığını ve G/C işlemlerinin işlemci işlemlerle çakışabileceğini varsayıyorum. Aşağıdaki büyüklükleri tanımlayın:
- Geri dönüş süresi= bir iş tamamlamak için gerçek.
  - Verim=  $T$  zaman periyodu başına tamamlanan ortalama iş sayısı.
  - İşlemci kullanımı= işlemcinin aktif olduğu (beklemediği) zamanın yüzdesi.  $T$  süresinin aşağıdaki yollardan her birine dağıldığını varsayıyarak bu miktarları bir, iki ve dört eşzamanlı iş için hesaplayın:
    - I/O ilk yarı, işlemci ikinci yarı;
    - I/O birinci ve dördüncü çeyrekler, işlemci ikinci ve üçüncü çeyrekler.
- 8.2** G/C'ye bağlı bir program, tek başına çalıştırıldığında G/C'yi beklemek için işlemciyi kullanmaktan daha fazla zaman harcayan bir programdır. İşlemciye bağlı bir program ise bunun tam tersidir. Kısa vadeli bir çizelgeleme algoritmasını yakın geçmişte az işlemci zamanı kullanmış olan programları tercih ettiğini varsayıyalım. Bu algoritmanın neden G/C'ye bağlı programları tercih ettiğini ve buna rağmen işlemciye bağlı programların işlemci zamanını kalıcı olarak reddetmediğini açıklayın.
- 8.3** Bir program satır toplamlarını hesaplar

$$C_i = \sum_{j=1}^n a_{ij}$$

100'e 100 olan bir  $A$  dizisi. Bilgisayarın 1000 kelimelik bir sayfa boyutuya talep sayfalaması kullandığını ve veriler için ayrılan ana bellek miktarının beş sayfa çerçevesi olduğunu varsayıyorum.  $A$ 'nın sanal bellekte satır ya da sütün olarak saklanması durumunda sayfa hata oranında herhangi bir fark var mıdır? Açıklayınız.

- 8.4**  $2^{16}$  baytlık eşit boyutlu bölmelere ve  $2^{24}$  baytlık toplam ana bellek boyutuna sahip sabit bir bölümleme şeması düşünün. Her yerleşik süreç için bir bölüme işaretçi içeren bir süreç tablosu tutulmaktadır. İşaretçi için kaç bit gereklidir?
- 8.5** Dinamik bir bölümleme şeması düşünün. Ortalama olarak, belleğin segmentlerin yarısı kadar delik içerdiğini gösterin.
- 8.6** Şu anda işlemci üzerinde çalışmaktadır sürecin sayfa tablosunun aşağıdaki gibi olduğunu varsayıyalım. Tüm sayılar ondalıktır, her şey sıfırdan başlayarak numaralandırılmıştır ve tüm adresler bellek bayt adresleridir. Sayfa boyutu 1024 bayttır.

| Sanal sayfa numarası | Geçerli bit | Referans biti | Bit değiştir | Sayfa çerçeve numarası |
|----------------------|-------------|---------------|--------------|------------------------|
| 0                    | 1           | 1             | 0            | 4                      |
| 1                    | 1           | 1             | 1            | 7                      |
| 2                    | 0           | 0             | 0            | -                      |
| 3                    | 1           | 0             | 0            | 2                      |
| 4                    | 0           | 0             | 0            | -                      |
| 5                    | 1           | 0             | 1            | 0                      |

## 316 BÖLÜM 8 / İŞLETİM SİSTEMİ DESTEĞİ

- a.** Genel olarak, CPU tarafından oluşturulan bir sanal adresin fiziksel bir ana bellek adresine nasıl dönüştürüldüğünü tam olarak açıklayınız.
- b.** Aşağıdaki sanal adreslerin her biri, eğer varsa, hangi fiziksel adrese karşılık gelir? (, herhangi bir sayfa hatasını işlemeye çalışmayın).
- 1052
  - 2221
  - 5499
- 8.7** Bir sanal bellek sistemindeki sayfa boyutunun ne çok küçük ne de çok büyük olması gerektiğine dair nedenler belirtiniz.
- 8.8** Bir süreç, aşağıdaki sırayla A, B, C, D ve E olmak üzere beş sayfaya atıfta bulunur: A; B; C; D; A; B; E; A; B; C; D; E  
Değiştirme algoritmasının ilk giren ilk çıkar olduğunu varsayıñ ve üç sayfa çerçeveli boş bir ana bellekle başlayan bu referans dizisi sırasında sayfa aktarımlarının sayısını bulun. Dört sayfa çerçevesi için tekrarlayın.
- 8.9** Sanal belleğe sahip bir bilgisayarda yürütme sırasında aşağıdaki sanal sayfa numaraları dizisiyle karşılaşılır:
- 3 4 2 6 4 7 1 3 2 6 3 5 1 2 3
- En son kullanılan sayfa değiştirme politikasının benimsendiğini varsayıñ.  $1 \dots n \dots 8$  için  $n$  ana bellek sayfa kapasitesinin bir fonksiyonu olarak sayfa isabet oranının (sayfanın ana bellekte olduğu sayfa referanslarının oranı) bir grafiğini çizin. Ana belleğin başlangıçta boş olduğunu varsayıñ.
- 8.10** VAX bilgisayarında, kullanıcı sayfa tabloları sistem alanındaki sanal adreslerde bulunur. Kullanıcı sayfa tablolarının ana bellek yerine sanal bellekte olmasına avantajı nedir? Dezavantajı nedir?
- 8.11** Program ifadesini varsayılmı
- ```
for (i= 1; i <= n; i+ ) a[i]=  
    b[i]+ c[i];
```
- sayfa boyutu 1000 kelime olan bir bellekte çalıştırılır.  $n = 1000$  olsun. Tüm kayıttan kayda talimatları içeren ve indeks kullanan bir makine kullanarak, yukarıdaki ifadeyi uygulamak için varsayımsal bir program yazın. Daha sonra yürütme sırasında sayfa referanslarının sırasını gösteriniz.
- 8.12** IBM System/370 mimarisı iki düzeyli bir bellek yapısı kullanır ve bu iki düzeyi kesimler ve sayfalar olarak adlandırır, ancak kesim yaklaşımı bu bölümde daha önce açıklanan özelliklerin doğrudan yoktur. Temel 370 mimaris için sayfa boyutu 2 Kbyte ya da 4 Kbyte olabilir ve kesim boyutu 64 Kbyte ya da 1 Mbyte olarak sabittir. 370/XA ve 370/ESA mimarileri için sayfa boyutu 4 Kbyte ve segment boyutu 1 Mbyte'tır. Bu şema segmentasyonun hangi avantajlarından yoksundur? Segmentasyonun 370 için faydası nedir?
- 8.13** Hem segmentasyon hem de sayfalama içeren bir bilgisayar sistemi düşünün. Bir segment bellekteyken, bazı kelimeler son sayfada boşça harçanır. Buna ek olarak, bir segment boyutu  $s$  ve sayfa boyutu  $p$  için,  $s/p$  sayfa tablosu girişleri vardır. Sayfa boyutu ne kadar küçükse, segmentin son sayfasındaki israf o kadar az olur, ancak sayfa tablosu o kadar büyük olur. Hangi sayfa boyutu toplam ek yükü en azı indirir?
- 8.14** Bir bilgisayarın önbelleği, ana belleği ve sanal bellek için kullanılan bir diski vardır. Eğer referans alınan kelime ön bellekte ise, ona erişmek için 20 ns gereklidir. Eğer kelime ana bellekteyse ama önbellekte değilse, önbelleğe yüklemek için 60 ns gereklidir ve sonra referans tekrar başlatılır. Sözcük ana bellekte değilse, sözcüğü diskiten almak için 12 ms, ardından önbelleğe kopyalamak için 60 ns gereklidir ve ardından referans yeniden başlatılır. Önbellek isabet oranı 0,9 ve ana bellek isabet oranı 0,6'dır. Bu sisteme başvurulan bir sözcüğe erişmek için gereken ortalama süre ns cinsinden nedir?
- 8.15** Bir görevin dört eşit boyutlu bölüme ayrıldığını ve sistemin her için sekiz girişli bir sayfa tanımlayııcı tablosu oluşturduğunu varsayılmı. Böylece sistem segmentasyon ve sayfalama kombinasyonuna sahip olur. Ayrıca sayfa boyutunun 2 Kbyte olduğunu varsayılmı.

- a. Her segmentin maksimum boyutu nedir?
  - b. Görev için maksimum mantıksal adres alanı nedir?
  - c. Bu görev tarafından 00021ABC fiziksel konumundaki bir öğeye erişildiğini varsayıñ. Görevin bunun için ürettiği mantıksal adresin biçimini nedir? Sistem için maksimum fiziksel adres alanı nedir?
- 8.16**  $2^{32}$  bayta kadar fiziksel ana belleğe erişebilen bir mikroişlemci varsayılm. Maksimum boyutu  $2^{31}$  bayt olan bir böülümlenmiş mantıksal adres alanı uygular. Her komut iki parçalı adresin tamamını içerir. Harici bellek yönetim birimleri (MMU) kullanılır ve bunların yönetim şeması segmentlere sabit  $2^{22}$  bayt boyutunda bitişik fiziksel bellek blokları atar. Bir segmentin başlangıç fiziksel adresi her zaman  $1024^4$  bölünür. Uygun sayıda MMU kullanarak mantıksal adresleri fiziksel adreslere dönüştüren harici işlemme mekanizmasının ayrıntılı ara bağlantısını gösterin ve bir MMU'nun ayrıntılı iç yapısını (her MMU'nun 128 girişli doğrudan eşlenmiş segment tanımlayıcı önbelleği içerdiğini varsayıarak) ve her MMU'nun nasıl seçildiğini gösterin.
- 8.17** 1-Mbyte'lik bir fiziksel bellek alanına eşlenmiş (her biri 2 Kbyte'lik 32 sayfadan oluşan) sayfalı bir mantıksal adres alanı düşünün.
- a. İşlemcinin mantıksal adresinin biçimini nedir?
  - b. Sayfa tablosunun uzunluğu ve genişliği nedir ("erişim hakları" bitlerini göz ardi ederek)?
  - c. Fiziksel bellek alanı yarı yarıya azaltılırsa sayfa tablosu üzerindeki etkisi ne olur?
- 8.18** IBM'in ana bilgisayar işletimi sistemi OS/390'da, ker-neldeki ana modüllerden biri Sistem Kaynak Yöneticisidir (SRM). Bu modül, kaynakların adres alanları (sureçler) arasında tahlis edilmesinden sorumludur. SRM, OS/390'a işletim sistemleri arasında benzersiz bir karmaşıklık derecesi kazandırır. Başka hiçbir anabilgisayar işletim sistemi ve kesinlikle başka hiçbir işletim sistemi türü SRM tarafından gerçekleştirilen işlevlerle eşleşmez. Kaynak kavramı işlemci, gerçek bellek ve I/O kanallarını içerir. SRM, işlemci, kanal ve çeşitli temel veri yapılarının kullanımına ilişkin istatistikleri toplar. Amacı, performans izlemeye ve analizine dayalı olarak optimum performans sağlamaktır. Kurulum çeşitli performans hedefleri belirler ve bunlar sistem kullanımına bağlı olarak kurulum ve iş performansı özelliklerini dinamik olarak değiştiren SRM'ye rehberlik eder. Buna karşılık SRM, eğitimli operatörün kullanıcı hizmetini iyileştirmek için yapılandırır ve parametre ayarlarını düzeltmesini sağlayan raporlar sunar.
- Bu sorun SRM faaliyetinin bir örneği ile ilgilidir. Gerçek bellek, çerçeve adı verilen ve binlerce olabilen eşit boyutlu bloklara bölünmüştür. Her çerçeve, sayfa olarak adlandırılan bir sanal bellek bloğunu tutabilir. SRM saniyede yaklaşık 20 kez kontrol alır ve her bir sayfa çerçevesini inceler. Sayfaya başvurulmamış ya da sayfa değiştirilmemişse bir sayاق 1 artırılır. Zaman içinde SRM bu sayıların ortalamasını alarak sistemdeki bir sayfa çerçevesine dokunmadan geçen ortalama saniye sayısını belirler. Bunun amacı ne olabilir ve SRM hangi eylemi gerçekleştirebilir?
- 8.19** Şekil 8.24'te gösterilen ARM sanal adres biçimlerinin her biri için fiziksel adres biçimini gösterin.
- 8.20** Ana bellek böülümlere ayrıldığında ARM sanal bellek çevirisini için Şekil 8.23'e benzer bir şekil çizin.

## SAYI SİSTEMLERİ

- 9.1** Ondalık Sistem
- 9.2** Konumsal Sayı Sistemleri
- 9.3** İkili Sistem
- 9.4** İkili ve Onlu Arasında Dönüşümme
  - Tamsayılar
  - Kesirler
- 9.5** Onaltılık Gösterim
- 9.6** Anahtar Terimler ve Sorunlar

**ÖĞRENİM HEDEFLERİ**

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- **Konumsal sayı sistemlerinin** temel kavramlarını ve terminolojisini anlamak.
- Hem tamsayılar hem de kesirler için **ondalık** ve **ikili** sayılar arasında dönüştürme tekniklerini açıklayın.
- **Onaltılık gösterimin** kullanılma gerekçesini açıklayınız.

## 9.1 ONDALIK SİSTEM

Günlük hayatı temsil etmek için ondalık basamaklara (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) dayalı bir sistem kullanırız ve bu sistemi ondalık sistem olarak adlandırırız. 83 sayısının ne anlama geldiğini düşünün. Sekiz onluk artı üç anlamına gelir:

$$83 = (8 * 10) + 3$$

4728 sayısı dört bin, yedi yüz, iki on, artı sekiz anlamına gelir:  $4728 = (4 * 1000) + (7 * 100) + (2 * 10) + 8$

Ondalık sistemin 10'luk bir **tabana** veya radikse sahip olduğu söylenir. Bu, sayıdaki her bir basamağın, o basamağın konumuna karşılık gelen bir güce yükseltilmiş 10 ile çarpıldığı anlamına gelir:

$$\begin{aligned} 83 &= (8 * 10^1) + (3 * 10^0) \\ 4728 &= (4 * 10^3) + (7 * 10^2) + (2 * 10^1) + (8 * 10^0) \end{aligned}$$

Aynı ilke ondalık kesirler için de geçerlidir, ancak 10'un negatif kuvvetleri kullanılır. Böylece, 0.256 ondalık kesri 2 onda bir artı 5 yüzde bir artı 6 binde bir anlamına gelir:

$$0.256 = (2 * 10^{-1}) + (5 * 10^{-2}) + (6 * 10^{-3})$$

Hem tamsayı hem de kesirli kısmı olan bir sayının rakamları 10'un hem pozitif hem de negatif kuvvetlerine yükseltilir:

$$\begin{aligned} 442.256 &= (4 * 10^2) + (4 * 10^1) + (2 * 10^0) + (2 * 10^{-1}) + (5 * 10^{-2}) \\ &\quad + (6 * 10^{-3}) \end{aligned}$$

Herhangi bir sayıda, en soldaki rakam **en yüksek** değeri taşıdığı için **en anlamlı rakam** olarak adlandırılır. En sağdaki basamak ise **en az anlamlı basamak** olarak adlandırılır. Bir önceki ondalık sayıdaki soldaki 4 en anlamlı rakamdır ve sağdaki 6 en az anlamlı rakamdır.

Tablo 9.1 her bir rakam pozisyonu ile o pozisyonu atanan değer arasındaki ilişkiyi göstermektedir. Her bir pozisyon, sağdaki pozisyonun değerinin 10 katı ve soldaki pozisyonun değerinin onda biri ile ağırlıklandırılmıştır. Böylece, pozisyonlar 10'un ardışık kuvvetlerini temsil eder. Pozisyonları Tablo 9.1'de gösterildiği gibi numaralandırırsak,  $i$  pozisyonu  $10^i$  değeri ile ağırlıklandırılır.

## 320 BÖLÜM 9 / SAYI SİSTEMLERİ

**Tablo 9.1** Ondalık Sayının Konumsal Yorumu

4	7	2	2	5	6
100s	10s	1s	onda bir	yüzdeler	binde
$10^2$	$10^1$	$10^0$	$10^{-1}$	$10^{-2}$	$10^{-3}$
Pozisyon 2	Pozisyon 1	Pozisyon 0	pozisyon -1	Pozisyon -2	Pozisyon -3

Genel olarak,  $X$ in ondalık gösterimi için  $= 5cd_2d_1d_0.d_{-1}d_{(-2)}d_{(-3)}c6$ ,  $X$ in değeri

$$X = \sum_i (d_i * 10^i) \quad (9.1)$$

Bir başka gözlem daha yapmaya değer. 509 sayısını düşünün ve bu sayı içinde kaç tane onluk olduğunu sorun. Onlu konumda bir 0 olduğu için, hiç olmadığını söylemek isteyebilirsiniz. Ama aslında 50 tane onluk vardır. Onluk konumundaki 0'ın anlamı, yüzüklere, binliklere benzerlerine eklenemeyen onlukların kalmadığıdır. Bu nedenle, her konum yalnızca daha yüksek konumlara toplanamayan artık sayıları içerdiğinden, her basamak konumunun dokuzdan büyük olmayan bir değere sahip olması gereklidir. Dokuz, bir pozisyonun bir sonraki yüksek pozisyon'a geçmeden önce tutabileceğii maksimum değerdir.

## 9.2 KONUMSAL SAYI SİSTEMLERİ

Konumsal bir sayı sisteminde, her sayı, her  $i$  rakam konumunun ilişkili bir  $r^i$  ağırlığına sahip olduğu bir rakam dizisi ile temsil edilir; burada  $r$ , sayı sisteminin radiksi veya tabanıdır. Böyle bir sistemde  $r$  radiksli bir sayının genel biçimini söyleyelim

$$(ca_3a_2a_1a_0.a_{-1}a_{(-2)}a_{(-3)}c)_r$$

Burada herhangi bir  $a_i$  rakamının değeri 0 ...  $a_i 6$  aralığında bir tamsayıdır.

$a_0$  ve  $a_{-1}$  radiks noktası olarak adlandırılır. Sayı şu değere sahip olacak şekilde tanımlanır

$$c + a_3r^3 + a_2r^2 + a_1r^{(1)} + a_0r^{(0)} + a_{-1}^{(1)}r^{(-1)} + a_{-2}^{(2)}r^{(-2)} + a_{-3}^{(3)}r^{(-3)} + \dots$$

$$= \sum_i (a_i * b^i) \quad (9.2)$$

O halde ondalık sistem, 10 radiksli ve 0 ile 9 aralığında rakamları olan konumsal sayı sisteminin özel bir durumudur.

Başka bir pozisyonel sisteme örnek olarak, 7 tabanlı sistemi ele alalım. Tablo 9.2 -1 ile 4 arasındaki pozisyonlar için ağırlık değerlerini göstermektedir. Her bir pozisyonda, rakam değeri 0 ile 6 arasında değişmektedir.

**Tablo 9.2** 7 Tabanındaki Bir Sayının Konumsal Yorumu

Pozisyon	4	3	2	1	0	-1
Üstel Formda Değer	$7^4$	$7^3$	$7^2$	$7^1$	$7^0$	$7^{-1}$
Ondalık Değer	2401	343	49	7	1	1/7

### 9.3 İKİLİ SİSTEM

Ondalık sistemde, 10 tabanlı sayıları temsil etmek için 10 farklı basamak kullanılır. İkili ise sadece iki basamak vardır: 1 ve 0. Dolayısıyla, ikili sistemdeki sayılar 2 tabanına göre temsil edilir.

Karışıklığı önlemek için, bazen bir sayının tabanını belirtmek için üzerine bir alt simge koyarız. Örneğin,  $83_{10}$  ve  $4728_{10}$  ondalık gösterimde temsil edilen sayılardır veya daha kısaca ondalık sayılardır. İkili gösterimdeki 1 ve 0 rakamları ondalık gösterimdeki ile aynı anlamda sahiptir:

$$0_2 = 0_{10}$$

$$1_2 = 1_{10}$$

Daha büyük sayıları temsil etmek için, ondalık gösterimde olduğu gibi, ikili her rakamın konumuna bağlı olarak bir değeri vardır:

$$10_2 = (1 * 2^1) + (0 * 2^0) = 2_{10}$$

$$11_2 = (1 * 2^1) + (1 * 2^0) = 3_{10}$$

$$100_2 = (1 * 2^2) + (0 * 2^1) + (0 * 2^0) = 4_{10}$$

ve benzeri. Yine, kesirli değerler radiksin negatif kuvvetleri ile temsil edilir:

$$1001.101 = 2^3 + 2^0 + 2^{-1} + 2^{(-3)} = 9.625_{10}$$

Genel olarak,  $Y$ 'nin ikili gösterimi için  $= 5cb_2b_1b_0.b_{-1}b_{(-2)}b_{(-3)}$ ,  $Y$ 'nin değeri

$$Y = \sum_i (b_i * 2^i) \quad (9.3)$$

### 9.4 İKİLİ VE ONDALIK ARASINDA DÖNÜŞTÜRME

Bir sayıyı ikili gösterimden ondalık dönüştürmek basit bir konudur. Aslında, önceki alt bölümde birkaç örnek göstermiştık. Gerekli olan tek şey, her bir ikili basamağı 2'nin uygun kuvvetiyle çarpmak ve sonuçları toplamaktır.

Ondalıktan ikiliye dönüştürmek için tamsayı ve kesirli kısımlar ayrı ayrı ele alınır.

#### Tamsayılar

Tamsayı kısmı için, ikili gösterimde bir tamsayının şu şekilde temsil edildiğini hatırlayın

$$b_{m-1}b_{(m)(-2)}cb_2b_1b_0 \quad b_i = 0 \text{ veya } 1$$

değerine sahiptir

$$(b_{m-1} * 2^{(m)(-1)}) + (b_{(m)(-2)} * 2^{m(-2)}) + c + (b_{(-1)} * 2^{(1)}) + b_0$$

## 322 BÖLÜM 9 / SAYI SİSTEMLERİ

Ondalık bir  $N$  tamsayısının ikili forma dönüştürülmesi gerektiğini varsayıyalım. Eğer  $N$ 'yi onluk sistemde 2'ye bölersek ve bir  $N_1$  bölümü ve bir  $R_0$  kalanı elde edersek, şunları yazabiliriz

$$N = 2 * N_1 + R_0 \quad R_0 = 0 \text{ veya } 1$$

Daha sonra,  $N_1$  bölümünü 2'ye böleriz. Yeni bölümün  $N_2$  ve yeni kalanın  $R_{(1)}$  olduğunu varsayıyalım. O zaman

$$N_1 = 2 * N_2 + R_1 \quad R_1 = 0 \text{ veya } 1$$

böylece

$$N = 2(2N_2 + R_1) + R_0 = (N_2^2 * 2^2) + (R_{(1)} * 2^1) + R_0$$

Eğer bir sonraki

$$N_2 = 2N_3 + R_2$$

biz varız

$$N = (N_3 * 2^3) + (R_{(2)} * 2^2) + (R_1 * 2^1) + R_0$$

$N \neq N_1 \neq N_2 \neq \dots$  olduğundan, bu diziye devam etmek sonunda bir  $N_{(m)(-)(1)} = 1$  bölümü (ikili eşdeğerleri sırasıyla 0 ve 1 olan ondalık tamsayılar 0 ve 1 hariç) ve 0 veya 1 olan bir  $R_{(m)(-)(2)}$  kalanı üretecektir. O halde

$$N = (1 * 2^{(m)(-)(1)}) + (R_{(m)(-)(2)} * 2^{(m)(-)(2)}) + \dots + (R_{(2)} * 2^{(2)}) + (R_1 * 2^{(1)}) + R_0$$

Bu nedenle, 10 tabanından 2 tabanına 2 ile tekrarlanan bölmelerle dönüştürüruz. Kalanlar ve son bölüm olan 1, bize artan önem sırasına göre  $N$ 'nin ikili basamaklarını verir. Şekil 9.1 iki örnek göstermektedir.

### Kesirler

Kesirli kısım için, ikili gösterimde 0 ile 1 arasında bir değere sahip bir sayının şu şekilde temsil edildiğini hatırlayın

$$0.b_{(-1)}b_{(-2)}b_{(-3)}\dots$$

ve değerine sahiptir

$$(b_{(-1)} * 2^{(-1)}) + (b_{(-2)} * 2^{(-2)}) + (b_{(-3)} * 2^{(-3)}) \dots$$

Bu şu şekilde yeniden yazılabilir

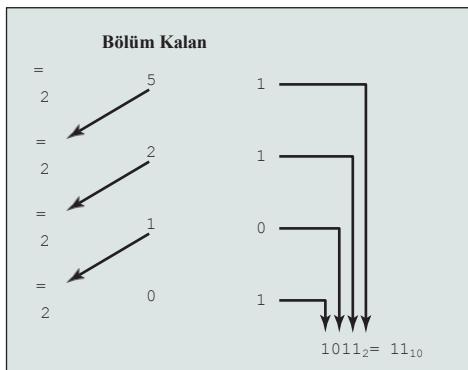
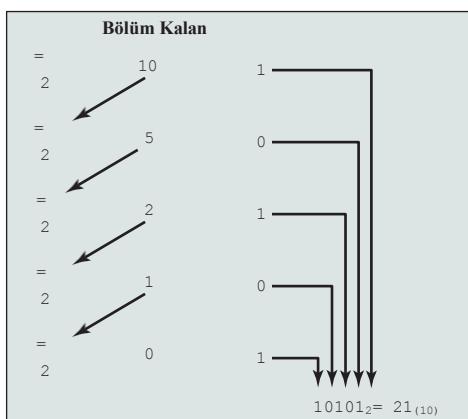
$$2^{-1} * (b_{(-1)} + 2^{-2} * (b_{(-2)} + 2^{-3} * (b_{(-3)} + \dots)))$$

Bu ifade dönüşüm için bir teknik önermektedir.  $F(0 \ 6 \ F \ 6 \ 1)$  sayısını ondalık gösterimden ikili gösterime dönüştürmek istediğimizi varsayıyalım.  $F$ 'nin şu şekilde ifade edilebileceğini biliyoruz

$$F = 2^{-1} * (b_{(-1)} + 2^{-2} * (b_{(-2)} + 2^{-3} * (b_{(-3)} + \dots)))$$

$F$ 'yi 2 ile çarparsa, elde ederiz,

$$2 * F = b_{(-1)} + 2^{-2} * (b_{(-2)} + 2^{-3} * (b_{(-3)} + \dots))$$

(a)  $11_{10}$ (b)  $21_{10}$ 

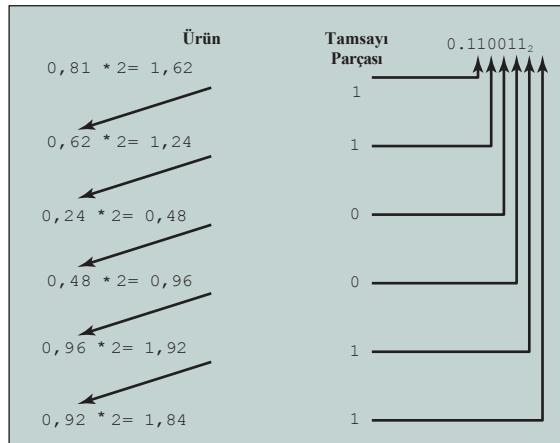
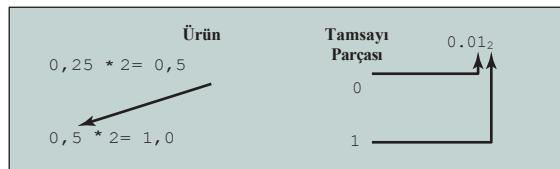
**Şekil 9.1** Tamsayılar için Ondalık Gösterimden İkili Gösterime Dönüşüm Örnekleri

Bu denklemden,  $0 \leq F \leq 1$  olduğu için ya  $0$  ya da  $1$  olması gereken  $(2^*F)$ 'nin tamsayı kısmının basitçe  $b_{-1}$  olduğunu görürüz. Dolayısıyla  $(2^*F) = b_{(-1)} + F_1$  diyebiliriz, burada  $0 \leq F_1 < 1$  ve burada

$$F_1 = 2^{(-1)*} (b_{(-1)} 2 + 2^{(0)*} (b_{(-3)} + 2^{(-1)*} (b_{(-4)} + c_1 c_2)))$$

$b_{(-2)}$ yi bulmak için işlemi tekrarlarız. Her adımda, bir önceki adımdaki sayının kesirli kısmı ile çarpılır. Çarpımındaki ondalık noktanın solundaki rakam  $0$  veya  $1$  olur ve en anlamlı rakamdan başlayarak ikili gösterime katkıda bulunur. Çarpımın kesirli kısmı bir sonraki adımda çarpım olarak kullanılır. Şekil 9.2'de iki örnek gösterilmektedir.

Bu işlemin kesin olması gerekmektedir; yani, sonlu sayıda basamağa sahip ondalık bir kesir, sonsuz sayıda basamağa sahip ikili bir kesir gerektirebilir. Bu gibi durumlarda, dönüşüm algoritması genellikle istenen doğruluğa bağlı olarak önceden belirlenmiş sayıda adımdan sonra durdurulur.

(a)  $0,81_{10} = 0,110011_2$  (yaklaşık)(b)  $0,25_{10} = 0,01_2$  (tam olarak)**Şekil 9.2** Kesirler için Ondalık Gösterimden İkili Gösterime Dönüşürme Önekleri

## 9.5 ONALTILIK GÖSTERİM

Dijital bilgisayar bileşenlerinin doğasında var olan ikili yapı nedeniyle, bilgisayarlardaki her türlü veri çeşitli ikili kodlarla temsil edilir. Ancak, ikili sistem bilgisayarlar için ne kadar kullanışlı olursa olsun, insanlar için son derece zahmetlidir. Sonuç olarak, bilgisayardaki gerçek ham verilerle çalışmak için zaman harcamak zorunda olan çoğu bilgisayar uzmanı daha kompakt bir gösterimi tercih eder.

Hangi notasyon kullanılmalı? Bir olasılık ondalık göstergemdir. Bu kesinlikle ikili göstergemden daha kompakttır, ancak 2 tabanı ile 10 tabanı arasında dönüştürme yapmanın sıkılığı nedeniyle gariptir.

Bunun yerine heksadesimal olarak bilinen bir göstergem benimsenmiştir. İkili rakamlar, **nibble** adı verilen dört bitlik kümeler halinde grupperdir. Dört ikili rakamın her olası kombinasyonuna aşağıdaki gibi bir simbol verilir:

0000=0	0100=4	1000=8	1100=C
0001=1	0101=5	1001=9	1101=D
0010=2	0110=6	1010=A	1110=E
0011=3	0111=7	1011=B	1111=F

16 symbol kullanıldığından, gösterim **onaltılık** olarak adlandırılır ve 16 symbol **onaltılık basamaklardır**.

Onaltılık basamaklar dizisi 16 tabanında bir tamsayıyı temsil ediyor gibi düşünülebilir (Tablo 9.3). Böylece,

$$\begin{aligned}2C_{16} &= (2^{16} * 16^1) + (C(16, * 16^0) \\&= (2_{10} * 16^1) + (12_{10} * 16^0) = 44\end{aligned}$$

Böylece, onaltılık sayıları 16 tabanında konumsal sayı sistemindeki sayılar olarak görürsek

$$Z = \sum_i (h_i * 16^i) \quad (9.4)$$

Burada 16 tabandır ve her onaltılık  $h_i$  basamağı  $0 \dots h_0$  6 15 ondalık aralığındadır ve  $0 \dots h_i \dots F$  onaltılık aralığına eşdeğerdir.

**Tablo 9.3** Ondalık, İkili ve Onaltılık

Ondalık (10 tabanı)	İkili (taban 2)	Onaltılık (16 tabanı)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	0001 0000	10
17	0001 0001	11
18	0001 0010	12
31	0001 1111	1F
100	0110 0100	64
255	1111 1111	FF
256	0001 0000 0000	100

## 326 BÖLÜM 9 / SAYI SİSTEMLERİ

Onaltılık gösterim sadece tam sayıları temsil etmek için değil, aynı zamanda metin, sayı veya başka bir veri türünü temsil etsin etmesin, herhangi bir ikili rakam dizisini temsil etmek için kısa bir gösterim olarak . Onaltılık gösterimin kullanılmasının nedenleri aşağıdaki gibidir:

1. İkili gösterimden daha kompaktır.
2. Çoğu bilgisayarda, ikili veriler 4 bitin bazı katlarını ve dolayısıyla tek bir onaltılık basamağın bazı katlarını kaplar.
3. İkili ve onaltılık gösterim arasında dönüşüm yapmak son derece kolaydır.

Son noktaya bir örnek olarak,  $110111100001$  ikili dizesini düşünün. Bu şu anlama gelir

$$\begin{array}{ccc} 1101 & 1110 & 0001 = DE1_{16} \\ D & E & 1 \end{array}$$

Bu işlem o kadar doğal bir şekilde gerçekleştirilir ki, deneyimli bir programcı ikili verilerin görsel temsillerini yazılı bir çaba sarf etmeden zihinsel olarak onaltılık eşdeğerlerine dönüştürebilir.

### 9.6 ANAHTAR TERİMLER VE SORUNLAR

#### Anahtar Terimler

taban	onaltılık	nibble
İkili	tamsayı	konumsal sayı sistemi
ondalık	en az anlamlı basamak	radix
kesir	en anlamlı basamak	radix noktası

#### Problemler

- 9.1** Aşağıdaki bazlarda 1'den 20'ye(<sub>10</sub>) kadar sayın:  
**a.** 8      **b.** 6      **c. 5**      **d. 3**
- 9.2**  $(1.1)_2$ ,  $(1.4)_{10}$  ve  $(1.5)_{16}$  sayılarını küçükten büyüğe sıralayınız.
- 9.3** Belirtilen baz dönüşümlerini gerçekleştirin:  
**a.**  $54_{(8)}$ den 5 tabanına      **b.**  $312_{(4)}$ ten 7 tabanına      **c.**  $520_6$ dan 7 tabanına      **d.**  $12212_{(3)}$ ten 9 tabanına
- 9.4** Bir sayıyı bir tabandan o bir kuvvette dönüştürme konusunda ne gibi genellemeler yapılabılırınız; örneğin, 3 tabanından 9 tabanına  $(3^2)$  veya 2 tabanından 4 tabanına  $(2^2)$  veya 8 tabanına  $(2^3)$ ?
- 9.5** Aşağıdaki ikili sayıları ondalık eşdeğerlerine dönüştürün:  
**a.** 001100      **b.** 000011      **c.** 011100      **d.** 111100      **e.** 101010
- 9.6** Aşağıdaki ikili sayıları ondalık eşdeğerlerine dönüştürün:  
**a.** 11100.011      **b.** 110011.10011      **c.** 1010101010.1
- 9.7** Aşağıdaki ondalık sayıları ikili eşdeğerlerine dönüştürün:  
**a.** 64      **b.** 100      **c.** 111      **d.** 145      **e.** 255
- 9.8** Aşağıdaki ondalık sayıları ikili eşdeğerlerine dönüştürün:  
**a.** 34.75      **b.** 25.25      **c.** 27.1875

- 9.9** Sonlu bir ikili gösterime (ikili noktanın sağında sonlu sayıda basamak) sahip her gerçek sayının aynı zamanda sonlu bir ondalık gösterime (ondalık noktanın sağında sonlu sayıda basamak) sahip olduğunu kanıtlayın.
- 9.10** Aşağıdaki sekizli sayıları (radix 8'li sayı) onaltılık gösterimde ifade edin:
- a. 12      b. 5655      c. 2550276      d. **76545336**      e. 3726755
- 9.11** Aşağıdaki onaltılık sayıları ondalık eşdeğerlerine dönüştürün:
- a. C      b. 9F      c. D52      d. 67E      e. ABCD
- 9.12** Aşağıdaki onaltılık sayıları ondalık eşdeğerlerine dönüştürün:
- a. F.4      b. D3.E      c. 1111.1      d. 888.8      e. EBA.C
- 9.13** Aşağıdaki ondalık sayıları onaltılık eşdeğerlerine dönüştürün:
- a. 16      b. 80      c. 2560      d. 3000      e. 62,500
- 9.14** Aşağıdaki ondalık sayıları onaltılık eşdeğerlerine dönüştürün:
- a. 204.125      b. 255.875      c. 631.25      d. 10000.00390625
- 9.15** Aşağıdaki onaltılık sayıları ikili eşdeğerlerine dönüştürün:
- a. E      b. 1C      c. **A64**      d. 1F.C      e. 239.4
- 9.16** Aşağıdaki ikili sayıları onaltılık eşdeğerlerine dönüştürün:
- a. 1001.1111      b. 110101.011001      c. 10100111.111011

# 10

## BÖLÜM

### BİLGİSAYAR ARITMETİĞİ

#### 10.1 Aritmetik ve Mantık Birimi

#### 10.2 Tamsayı Gösterimi

- İşaret-Büyüklük Gösterimi İki Tümleyen
- Aralık Genişletme
- Sabit Nokta Gösterimi

#### 10.3 Tamsayı Aritmetiği

- Olumsuzlama
- Toplama ve Çıkarma Çarpma
- Bölüm

#### 10.4 Kayan Nokta Gösterimi

- Prensipler
- İkili Kayan Nokta Gösterimi için IEEE Standardı

#### 10.5 Kayan Nokta Aritmetiği

- Toplama ve Çıkarma Çarpma ve
- Bölme Hassasiyetle İlgili Hususlar
- İkili Kayan Nokta Aritmetiği için IEEE Standardı

#### 10.6 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler

### ÖĞRENİM HEDEFLERİ

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- Sayıların temsil edilme şekli (ikili format) ile temel aritmetik işlemler için kullanılan algoritmalar arasındaki ayrimi anlamak.
- **İkiye tümleyen gösterimini** açıklayın.
- İki tümleyen gösteriminde temel aritmetik işlem yapma tekniklerine genel bir bakış sunmak.
- **Kayan noktalı sayıların** gösteriminde anlamlı, taban ve üs kullanımını anlamak.
- Kayan nokta gösterimi için IEEE 754 standartına genel bir bakış sunun.
- Koruma bitleri, yuvarlama, normal altı sayılar, alttan taşıma ve üstten taşıma dahil olmak üzere kayan nokta aritmetiği ile ilgili bazı temel kavramları anlamak.

İşlemci incelememize aritmetik ve mantık birimine (ALU) genel bir bakış ile başlıyoruz. Bu bölüm daha sonra ALU'nun en karmaşık yönü olan bilgisayar aritmetiğine odaklanmaktadır. Sayısal mantıktaki basit mantık ve aritmetik fonksiyonlarının uygulamaları Bölüm 11'de, ALU'bir parçası olan mantık fonksiyonları ise Bölüm 12'de açıklanmaktadır.

Bilgisayar aritmetiği genellikle iki farklı sayı türü üzerinde gerçekleştirilir: tamsayı ve kayan nokta. Her iki durumda da, seçilen gösterim önemli bir tasarım konusudur ve önce ele alınır, ardından aritmetik işlemler tartışılr.

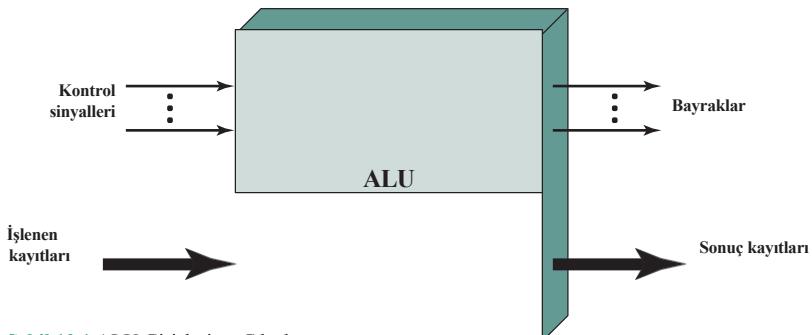
Bu bölüm, her biri gölgeli bir kutuda vurgulanan bir dizi örnek içermektedir.

## 10.1 ARİTMETİK VE MANTIK BİRİMİ

ALU, bilgisayarın veriler üzerinde aritmetik ve mantıksal işlemler gerçekleştiren kismıdır. Bilgisayar sisteminin diğer tüm unsurları - kontrol birimi, kayıtlar, bellek, I/O - esas olarak ALU'ya işlemesi için veri getirmek ve ardından sonuçları geri almak için vardır. ALU'yu ele aldığımızda bir anlamda bilgisayarın çekirdeğine ya da özüne ulaşmış oluruz.

Bir ALU ve aslında bilgisayardaki tüm elektronik bileşenler, ikili rakamları saklayabilen ve basit Boolean mantık işlemlerini gerçekleştirebilen basit dijital mantık cihazlarının kullanımına dayanır.

Şekil 10.1 genel hatlarıyla ALU'nun işlemcinin geri kalanıyla nasıl bağlılı olduğunu göstermektedir. Aritmetik ve mantık işlemleri için operandlar ALU'ya yazmaçlarda sunulur ve bir işlemin sonuçları yazmaçlarda saklanır. Bu yazmaçlar, ALU'ya sinyal yollarıyla bağlanan işlemci içindeki geçici depolama konumlarıdır (örneğin, bkz. Şekil 2.3). ALU, bir işlemin sonucu olarak bayrakları da ayarlayabilir. Örneğin, bir işlemin sonucu saklanacağı yazmacın uzunluğunu aşarsa taşıma bayrağı 1'e ayarlanır.



Şekil 10.1 ALU Girişleri ve Çıkışları

Bayrak değerleri de işlemci içindeki kayıtlarda saklanır. İşlemci, ALU'nun çalışmasını ve verilerin ALU'nun içine ve dışına hareketini kontrol eden sinyaller sağlar.

## 10.2 ENTEGRE SUNUM

İkili sayı sisteminde,<sup>(1)</sup> rastgele sayılar sadece sıfır ve bir rakamları, eksi işaretti (negatif sayılar için) ve **nokta** veya **radix noktası** (kesirli bileşene sahip sayılar için) ile gösterilebilir.

$$-1101.0101_2 = -13.3125_{10}$$

Ancak bilgisayar depolama ve işleme amaçları doğrultusunda, eksi işaretti ve radix noktası için özel sembollerden faydalananamayız. Sayıları temsил etmek için yalnızca ikili rakamlar (0 ve 1) kullanılabilir. Negatif olmayan tamsayılarla sınırlı kalırsak, gösterim basittir.

8 bitlik bir sözcük 0'dan 255'e kadar olan sayıları temsil edebilir, örneğin

$$\begin{aligned} &= \\ &= \\ &= 001010011 \\ &= 10000000128 \\ &= 1111111255 \end{aligned}$$

Genel olarak,  $a_{(n)-1}a_{(n)-2} \dots a_1a_0$  ikili rakamlarından oluşan  $n$  bitlik bir dizi, işaretsiz bir tamsayı olan  $A$  ile gösterilirse, değeri

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

<sup>1</sup>Sayı sistemleri ondalık, ikili, ) hakkında temel bilgiler için Bölüm 9'a bakınız.

## İşaret-Büyüklük Gösterimi

Negatif ve pozitif tamsayıları temsil etmek için kullanılan birkaç alternatif kural vardır ve bunların tümü kelimedeki en anlamlı (en soldaki) bitin işaret biti olarak ele alınmasını içerir. İşaret biti 0 ise sayı pozitiftir; işaret biti 1 ise sayı negatiftir.

Bir işaret biti kullanan en basit gösterim şekli işaret-büyüklük gösterimidir. N bitlik bir sözcükte, en sağdaki  $n - 1$  bit tamsayının büyüklüğünü tutar.

+ 18	$= 00010010$
-18	$= 10010010$ (İşaret Büyüklüğü)

Genel durum aşağıdaki gibi ifade edilebilir:

$$\begin{array}{ll} \text{İşaret Büyüklüğü} & A = \mu \\ & \begin{array}{l} \sum_{i=0}^{n-2} 2^i a_i \quad a_{n-1} = 0 \text{ ise} \\ \overline{a_{n-1}} = \sum_{i=0}^{n-2} 2^i a_i \quad \text{eğer} \end{array} \end{array} \quad (10.1)$$

İşaret-büyüklük gösteriminin çeşitli dezavantajları vardır. Bunlardan biri, toplama ve çıkarma işlemlerinin, gerekli işlemi gerçekleştirmek için hem sayıların işaretlerinin hem de göreli büyülüklerinin dikkate alınmasını gerektirmesidir. Bu durum Bölüm 10.3'teki tartışmada açıklığa kavuşacaktır. Diğer bir dezavantaj ise 0'ın iki gösterimi olmasıdır:

+ 0 <sub>10</sub>	$= 00000000$
- 0 <sub>10</sub>	$= 10000000$ (İşaret Büyüklüğü)

Bu elverişsiz bir durumdur çünkü 0'ı test etmek (bilgisayarlarda sıkça yapılan bir işlem) tek bir gösterime göre biraz daha zordur.

Bu dezavantajlar nedeniyle, ALU'nun tamsayı kısmının uygulanmasında işaret-büyüklük gösterimi nadiren kullanılır. Bunun yerine, en yaygın şema ikiye tümleyen gösterimidir.<sup>2</sup>

## İkili Tümleyen Temsili

İşaret büyülüğü gibi, ikiye tümleyen gösterimi de en anlamlı biti işaret biti olarak kullanır ve bir tamsayının pozitif mi negatif mi olduğunu test etmeyi kolaylaştırır. İşaret büyülüğü gösteriminin kullanımından diğer bitlerin yorumlanma şekliyle ayrılır. Tablo 10.1, bu bölümde ve bir sonraki bölümde detaylandırılan ikiye tümleyen gösteriminin ve aritmetiğinin temel özelliklerini vurgulamaktadır.

İkiye tümleyen gösterimine ilişkin çoğu uygulama, şemanın geçerli olduğuna dair resmi bir kanıt olmaksızın negatif sayı üretme kurallarına odaklanır. Bunun yerine,

<sup>2</sup>Literatürde, *ikinin tümleyeni* veya *2'nin tümleyeni* terimleri sıkılıkla kullanılmaktadır. Burada standart belgelerinde kullanılan uygulamayı takip ediyor ve kesme işaretini atlıyoruz (örneğin, IEEE Std 100-1992, *The New IEEE Standard Dictionary of Electrical and Electronics Terms*).

**Tablo 10.1** İki Tümleyen Gösterimi ve Arithmetığının Özellikleri

<b>Menzil</b>	$-2^{n-1}$ ila $2^{(n)(-1)} - 1$
<b>Sıfırın Temsil Sayısı</b>	Bir
<b>Olumsuzlama</b>	Karşılık gelen pozitif sayının her bitinin Boole tümleyenini alın, ardından ortaya çıkan bit desenine işaretsiz bir tamsayı olarak bakıldığından 1 ekleyin.
<b>Bit Uzunluğunun Genişletilmesi</b>	Sola ek bit konumları ekleyin ve orijinal işaret bitinin değeriyle doldurun.
<b>Taşma Kurallı</b>	Aynı işaretre sahip iki sayı (her ikisi de pozitif veya her ikisi de negatif) toplanırsa, ancak ve ancak sonuç ters işaretre sahipse taşıma meydana gelir.
<b>Çıkarma Kurallı</b>	$B'$ yi $A$ 'dan çıkarmak için $B'$ nin ikili tümleyenini alın ve $A'$ ya ekleyin.

Bu bölümdeki ve Bölüm 10.3'teki ikiye tümleyen tamsayılar sunumumuz, daha önce işaretsiz ve işaret büyülüğu için yaptığımız gibi, ikiye tümleyen gösteriminin en iyi bitlerin ağırlıklı toplamı cinsinden tanımlanarak anlaşılabileceğini öneren [DATT93]'e dayanmaktadır. Bu yaklaşımın avantajı, ikiye tümleyen gösteriminde arithmetik işlemler için kuralların bazı özel durumlar için çalışmayıabileceğine dair herhangi bir şüphe bırakmamasıdır.

İkiye tümleyen gösteriminde  $n$  bitlik bir tamsayı,  $A$ , düşünün. Eğer  $A$  pozitif ise, işaret biti,  $a_{n-1}$ , sıfırdır. Kalan bitler, işaret büyülüğu ile aynı şekilde sayının temsil eder:

$$A = \sum_{i=0}^{n-2} 2^i a_i \quad A \geq 0 \text{ için}$$

Sıfır sayısı pozitif olarak tanımlanır ve bu nedenle bir 0 işaret bitine ve tüm 0'lardan oluşan bir büyülüğe sahiptir. Temsil edilebilecek pozitif tamsayı aralığının 0 (tüm büyülüklük bitleri 0'dır) ile  $2^{n-1} - 1$  (tüm büyülüklük bitleri 1'dir) arasında olduğunu görebiliriz. Daha büyük bir sayı daha fazla bit gerektirecektir.

Şimdi, negatif bir  $A$  sayısı için ( $A \neq 0$ ), işaret biti,  $a_{n-1}$ , birdir. Geriye kalan  $n - 1$  bit,  $2^{(n)(-1)}$  değerlerinden herhangi birini alabilir. Bu nedenle, temsil edilebilecek negatif tamsayı aralığı  $-1$  ila  $-2^{(n)(-1)}$  arasındadır. Negatif tamsayılar bit değerlerini öyle bir şekilde atamak isteriz ki arithmetik, işaretsiz tamsayı arithmetigine benzer şekilde basit bir şekilde ele alınabilisin. İşaretsiz tamsayı gösteriminde, bit gösteriminden bir tamsayıının değerini hesaplamak için, en anlamlı bitin ağırlığı  $+ 2^{(n)(-1)}$ dir. İşaret biti olan bir gösterim için, Bölüm 10.3'te göreceğimiz gibi, en anlamlı bitin ağırlığı  $-2^{(n)(-1)}$  ise istenen arithmetik özelliklerin elde edildiği ortaya çıkmaktadır. Bu, negatif sayılar için aşağıdaki ifadeyi veren ikiye tümleyen gösteriminde kullanılan konvansiyondur:

$$\text{İkili Tamamlayıcı} \quad A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i \quad (10.2)$$

Denklem (10.2) hem pozitif hem de negatif sayılar için ikiye tümleyen gösterimini tanımlar.  $a_{n-1} = 0$  için,  $-2^{(n)(-1)} a_{n-1} = 0$  terimi ve denklem şunları tanımlar

**Tablo 10.2** 4-Bit Tamsayılar için Alternatif Gösterimler

Ondalık Gösterimi	İşaret-Büyüklük Gösterimi	İkiye Tümleyen Temsili	Önyargılı Temsil
+ 8	-	-	1111
+ 7	0111	0111	1110
+ 6	0110	0110	1101
+ 5	0101	0101	1100
+ 4	0100	0100	1011
+ 3	0011	0011	1010
+ 2	0010	0010	1001
+ 1	0001	0001	1000
+ 0	0000	0000	0111
-0	1000	-	-
-1	1001	1111	0110
-2	1010	1110	0101
-3	1011	1101	0100
-4	1100	1100	0011
-5	1101	1011	0010
-6	1110	1010	0001
-7	1111	1001	0000
-8	-	1000	-

negatif olmayan bir  $a_{n(-1)} = 1$  olduğunda,  $2^{n-1}$  terimi toplama teriminden çıkarılır ve negatif bir tamsayı elde edilir.

Tablo 10.2, 4-bit tamsayılar için işaret-büyüklük ve ikiye tümleyen gösterimlerini karşılaştırmaktadır. İkiye tümleyen insan bakış açısından garip bir gösterim olmasına rağmen, en önemli aritmetik işlemleri olan toplama ve çıkarma işlemlerini kolaylaştırdığını göreceğiz. Bu nedenle, neredeyse evrensel olarak tamsayılar için işlemci gösterimini olarak kullanılır.

İkiye tümleyen temsilinin doğasına ilişkin faydalı örnek, kutunun en sağındaki değerin 1 ( $2^{(0)}$ ) olduğu ve sola doğru her bir sonraki konumun, negatif olan en soldaki kadar değer olarak iki katına çıktıığı bir değer kutusudur. Şekil 10.2'a'da görebileceğiniz gibi, temsil edilebilecek en negatif ikiye tümleyen sayı  $-2^{n-1}$ dir; işaret biti dışındaki bitlerden herhangi biri ise, sayıya pozitif bir miktar ekler. Ayrıca, negatif sayının en sol konumunda 1 olması gerektiği ve pozitif bir sayının bu konumda 0 olması gerektiği açıklır. Dolayısıyla, en büyük pozitif sayı bir 0 ve ardından gelen tüm 1'lerdir, bu da  $2^{n-1} - 1$ 'e eşittir.

Şekil 10.2'nin geri kalanında, ikiye tamamlayıcıdan ondalık sayıya ve ondalık sayıdan ikiye tamamlayıcıya dönüştürmek için değer kutusunun kullanımı gösterilmektedir.

### Menzil Uzatma

Bazen  $n$  bitlik bir tamsayı almak ve bunu  $m$  bitte saklamak istenebilir, burada  $m > n$ . Bit uzunluğunun bu şekilde genişletilmesi **aralık genişletme** olarak adlandırılır, çünkü ifade edilebilecek sayı aralığı bit uzunluğunun artırılmasıyla genişletilir.

### 334 BÖLÜM 10 / BİLGİSAYAR ARİTMETİĞİ

-128	64	32	16	8	4	2	1

(a) Sekiz konumlu ikiye tümleyen değer kutusu

-128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

$-128 - +2 + 1 = -125$

(b) İkili 10000011 adresini ondalık sayıya dönüştürme

-128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

$-120 = -128 + 8$

(c) Ondalık - 120 değerini ikiliye dönüştür

**Şekil 10.2** İkiye Tümleyen İkili ve Onlu Arasında Dönüşüm için Değer Kutusu Kullanımı

İşaret-büyüklük gösteriminde bu kolayca gerçekleştirilebilir: işaret bitini en soldaki yeni konuma taşıyın ve sıfırlarla doldurun.

+18	=	00010010	(işaret büyülüğu, 8 bit)
+18	=	0000000000010010	(işaret büyülüğu, 16 bit)
-18	=	10010010	(işaret büyülüğu, 8 bit)
-18	=	100000000010010	(işaret büyülüğu, 16 bit)

Bu prosedür ikiye tümleyen negatif tamsayılar için çalışmayaçaktır. Aynı örneği kullanarak,

+18	=	00010010	(ikiye tümleyen, 8 bit)
+18	=	0000000000010010	(ikiye tümleyen, 16 bit)
-18	=	11101110	(ikiye tümleyen, 8 bit)
-32,658	=	1000000001101110	(ikiye tümleyen, 16 bit)

Sondan bir sonraki satır Şekil 10.2'deki değer kutusu kullanılarak kolayca görülebilir. Son satır Denklem (10.2) veya 16 bitlik bir değer kutusu kullanılarak doğrulanabilir.

Bunun yerine, ikiye tümleyen tamsayılar için kural, işaret bitini en soldaki yeni konuma taşımak ve işaret bitinin kopyalarıyla doldurmaktır. Pozitif sayılar için sıfırlarla ve negatif sayılar için birlerle doldurun. Buna işaret uzantısı denir.

-18	=	11101110	(ikiye tümleyen, 8 bit)
-18	=	11111111101110	(ikiye tümleyen, 16 bit)

Bu kuralın neden işe yaradığını görmek için, yine  $n$ -bitlik bir ikili rakam dizisi  $a_{(n)(-1)} a_{(n)(-2)} \dots a_1 a_0$  düşünelim ve diziyi ikili tümleyen  $A$  tamsayısı olarak yorumlayalım, böylece değeri

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Eğer  $A$  pozitif bir sayı ise, kural açıkça işler. Şimdi, eğer  $A$  negatifse ve  $m$  bitlik bir gösterim oluşturmak istiyorsak,  $m$

$$A = -2^{m-1}a_{m-1} + \sum_{i=0}^{m-2} 2^i a_i$$

İki değer eşit olmalıdır:

$$\begin{aligned} -2^{m-1} + \sum_{i=0}^{m-2} 2^i a_i &= -2^{n-1} + \sum_{i=0}^{n-2} 2^i a_i \\ -2^{m-1} + \sum_{i=n-1}^{m-2} 2^i a_i &= -2^{n-1} \\ -2^{n-1} + \sum_{i=n-1}^{m-2} 2^i a_i &= 2^{m-1} \\ 1 + \sum_{i=0}^{n-2} 2^i a_i + \sum_{i=n-1}^{m-2} 2^i a_i &= 1 + \sum_{i=0}^{m-2} 2^i \\ \sum_{i=n-1}^{m-2} 2^i a_i &= \sum_{i=n-1}^{m-2} 2^i \end{aligned}$$

$$1 \cdot a_{m-2} = \dots = a_{(n)(-2)} = a_{(n)(-1)} = 1$$

İlk denklemden ikinci denkleme geçerken, en az anlamlı  $n - 1$  bitin iki gösterim arasında değişmemesini şart koşuyoruz. Daha sonra sondan bir önceki denkleme geliriz ki bu da ancak  $n - 1$  ile  $m - 2$  arasındaki tüm bitler 1 ise doğrudur. Bu nedenle, işaret genişletme kuralı çalışır. Okuyucu, Bölüm 10.3'ün başındaki ikiye tümleyen olumsuzlama tartışmasını inceledikten sonra kuralı daha kolay kavrayabilir.

### Sabit Nokta Gösterimi

Son olarak, bu bölümde tartışılan gösterimlerin bazen sabit nokta olarak adlandırıldığını belirtmek isteriz. Bunun nedeni, radix noktasının (ikili) sabit olması ve en sağdaki basamağın sağında olduğunun varsayılmıştır. Programcı, sayıları ölçeklendirerek ikili kesirler için aynı gösterimi kullanabilir, böylece ikili nokta dolaylı olarak başka bir konuma yerleştirilir.

## 10.3 ENTEGRE ARİTMETİK

Bu bölümde, ikili tümleşik gösterimdeki sayılar üzerindeki yaygın aritmetik fonksiyonlar incelenmektedir.

### Olumsuzlama

İşaret-büyüklük gösteriminde, bir tamsayının olumsuzunu oluşturmak için kural basittir: işaret bitini ters çevirin. İkiye tümleyen gösteriminde, bir tamsayının olumsuzu aşağıdaki kurallarla oluşturulabilir:

1. Tamsayının her bitinin Boole tümleyenini alın (işaret biti dahil). Yani, her 1'i 0'a ve her 0'ı 1'e ayarlayın.
2. Sonucu işaretsiz ikili sayı olarak değerlendirek 1 ekleyin.

Bu iki adımlı işlem, **ikiye tümleyen işlemi** veya bir tamsayının ikiye tümleyeninin alınması olarak adlandırılır.

$$\begin{array}{r}
 +18 = 00010010 \text{ (ikiye tümleyen) bitsel tümleyen=} \\
 11101101 \\
 + \quad \quad \quad 1 \\
 \hline
 = 11101110-18
 \end{array}$$

Beklendiği gibi, bu sayının negatifinin negatifi kendisidir:

$$\begin{array}{r}
 -18 = 11101110 \text{ (ikiye tümleyen) bitsel tümleyen=} \\
 00010001 \\
 + \quad \quad \quad 1 \\
 \hline
 = +00010010 18
 \end{array}$$

Denklem (10.2)'deki ikiye tümleyen gösteriminin tanımını kullanarak az önce açıklanan işlemin geçerliliğini gösterebiliriz. Yine,  $n$  bitlik bir ikili rakam dizisini  $a_{(n)(-1)}a_{(n)(-2)} \subset a_1a_0$  ikiye tümleyen bir  $A$  tamsayısı olarak yorumlayın, böylece değeri

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Şimdi bitsel tümleyenini oluşturun,  $a_{n-1}a_{n(-2)} \subset a_0$ , ve bunu işaretsiz bir tamsayı olarak ele alarak 1 ekleyin. Son olarak, elde edilen n-bit ikili sayı dizisini ikili tümleyen bir  $B$  tamsayısı olarak yorumlayın, böylece değeri

$$B = -2^{n-1}\overline{a_{n-1}} + 1 + \sum_{i=0}^{n-2} 2^i \overline{a_i}$$

Şimdi,  $A = -B$  istiyoruz, yani  $A + B = 0$ . Bunun doğru olduğu kolayca gösterilebilir:

$$\begin{aligned}
 A + B &= -(a_{n(-1)} + a^{(n)(-1)})\overline{2^{(n)(-1)}} + 1 + \left( \sum_{i=0}^{n-2} 2^i (a_i + \overline{a_i}) \right) \\
 &= -2^{n-1} + 1 + \left( \sum_{i=0}^{n-2} 2^i \right) \\
 &= -2^{n-1} + 1 + (2^{(n)(-1)} - 1) \\
 &= -2^{n-1} + 2^{n(-1)} = 0
 \end{aligned}$$

Önceki türetme, 1 eklemek amacıyla önce  $A$ 'nın bitsel tümleyenini işaretetsiz bir tamsayı olarak ele alabileceğimizi ve ardından sonucu ikiye tümleyen bir tamsayı olarak ele alabileceğimizi varsayar. Dikkate alınması gereken iki özel durum vardır. İlk olarak,  $A=0$ . Bu durumda, 8 bitlik bir gösterim için:

$$\begin{array}{r}
 & 0 = 00000000 \text{ (ikiye tümleyen) bitsel tümleyen=} \\
 11111111 & \\
 + & \underline{\quad} \quad 1 \\
 = & 100000000
 \end{array}$$

En anlamlı bit konumunda bir *carry out* vardır ve bu göz arı edilir. Sonuç, 0'in olumsuzlaşmasının olması gerektiği gibi 0 olmalıdır.

İkinci özel durum daha büyük bir sorundur. Eğer 1 ve ardından  $n - 1$  sıfırdan oluşan bit deseninin tersini alırsak, aynı sayıyı geri elde ederiz. Örneğin, 8 bitlik kelimeler için,

$$\begin{array}{r}
 & +128 = 10000000 \text{ (ikiye tümleyen) bitsel tümleyen=} \\
 01111111 & \\
 + & \underline{\quad} \\
 = & 10000000-128
 \end{array}$$

Böyle bir anomalilik kaçınılmazdır.  $N$  bitlik bir sözcükteki farklı bit desenlerinin sayısı  $2^n$  dir ve bu çift sayıdır. Pozitif ve negatif tamsayıları ve 0'ı temsil etmek istiyoruz. Eğer eşit sayıda pozitif ve negatif tamsayı temsil ediliyorsa ( işaret büyülüğu ), 0 için iki temsil vardır. 0'in yalnızca bir temsili varsa (ikiye tümleyen), o zaman eşit olmayan sayıda negatif ve pozitif sayı temsil edilmelidir. İkiye tümleyen durumunda, n-bit uzunluk için,  $-2^{(n)(-1)}$  için bir temsil vardır ancak  $+2^{(n)(-1)}$  için yoktur.

### Toplama ve Çıkarma

İkili tümleyenlerde toplama işlemi Şekil 10.3'te gösterilmiştir. Toplama işlemi, iki sayı işaretetsiz tamsayılarla gibi ilerler. İlk dört örnek başarılı işlemleri göstermektedir. İşlemenin sonucu pozitifse, ikiye tümleyen biçiminde pozitif bir sayı elde ederiz, bu da işaretetsiz tamsayı biçimindekiyle aynıdır. İşlemenin sonucu negatifse, ikiye tümleyen biçiminde negatif bir sayı elde ederiz. Bazı durumlarda, sözcüğün sonunun ötesinde (gölgelendirme ile gösterilen) bir taşıma biti olduğunu ve bunun göz arı edildiğini unutmayın.

Herhangi bir toplama işleminde sonuç, kullanılan kelime boyutunda tutulabileceğinden daha büyük olabilir. Bu duruma **taşma** denir. Taşma meydana geldiğinde, ALU bu durumu bildirmelidir, böylece sonuç kullanılmaya çalışılmaz. Taşmayı tespit etmek için aşağıdaki kurala uyulur:

**TAŞMA KURALI:** İki sayı toplanırsa ve her ikisi de pozitif veya her ikisi de negatifse, ancak ve ancak sonuç ters işaretre sahipse taşma meydana gelir.

$  \begin{array}{r}  1001 = -7 \\  +\underline{0101} = 5 \\  1110 = -2 \\  \hline  \end{array}  $ <p>(a) <math>(-7) + (+5)</math></p>	$  \begin{array}{r}  1100 = -4 \\  +\underline{0100} = 4 \\  \hline  10000 = 0  \end{array}  $ <p>(b) <math>(-4) + (+4)</math></p>
$  \begin{array}{r}  0011 = 3 \\  +\underline{0100} = 4 \\  0111 = 7 \\  \hline  \end{array}  $ <p>(c) <math>(+3) + (+4)</math></p>	$  \begin{array}{r}  1100 = -4 \\  +\underline{1111} = -1 \\  \hline  11011 = -5  \end{array}  $ <p>(d) <math>(-4) + (-1)</math></p>
$  \begin{array}{r}  0101 = 5 \\  +\underline{0100} = 4 \\  1001 = \text{Overflow} \\  \hline  \end{array}  $ <p>(e) <math>(+5) + (+4)</math></p>	$  \begin{array}{r}  1001 = -7 \\  +\underline{1010} = -6 \\  \hline  10011 = \text{Overflow}  \end{array}  $ <p>(f) <math>(-7) + (-6)</math></p>

Şekil 10.3 İkiye Tümleyen Gösteriminde Sayıların Toplanması

Şekil 10.3e ve f taşıma örneklerini göstermektedir. Taşma olsa da olmasa da taşmanın meydana gelebileceğini unutmayın.

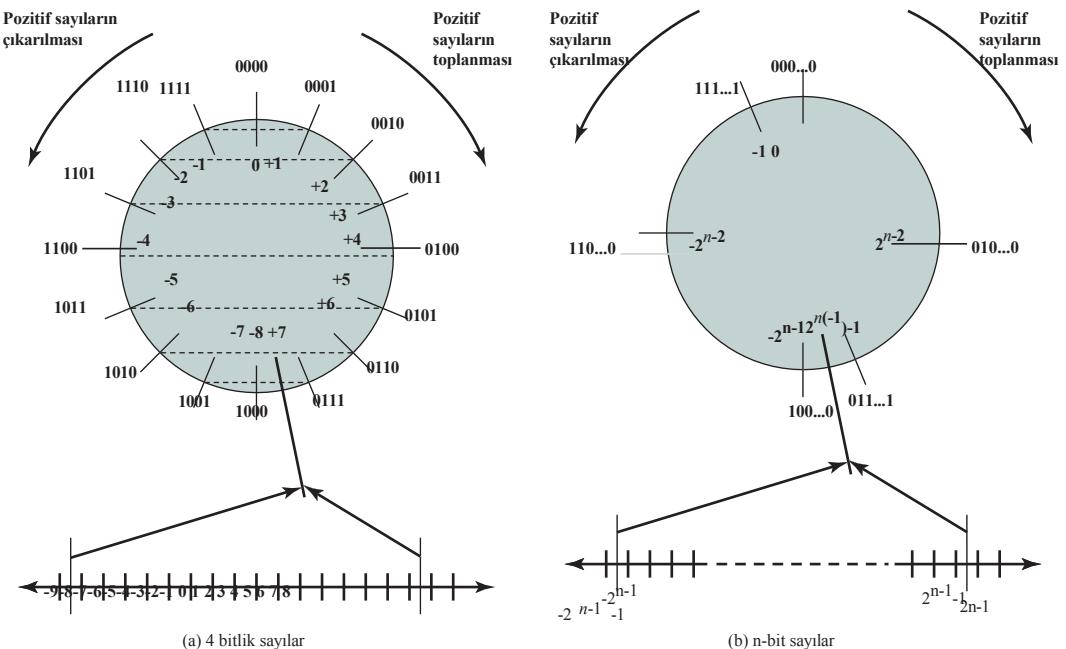
Çıkarma işlemi aşağıdaki kuralla kolayca halledilir:

**ÇIKARMA KURALI:** Bir sayıyı (alt uç) diğerinden (alt uç) çıkarmak için, alt ucun ikili tümleyenini (negatifini) alın ve alt uca ekleyin.

Böylece, Şekil 10.4'te gösterildiği gibi toplama işlemi kullanılarak çıkarma işlemi gerçekleştirilebilir. Son iki örnek taşıma kuralının hala geçerli olduğunu göstermektedir.

$  \begin{array}{r}  0010 = 2 \\  +\underline{1001} = -7 \\  1011 = -5  \end{array}  $ <p>(a) <math>M = 2 = 0010</math>  <math>S = 7 = 0111</math>  <math>-S = 1001</math></p>	$  \begin{array}{r}  0101 = 5 \\  +\underline{1110} = -2 \\  \hline  10011 = 3  \end{array}  $ <p>(b) <math>M = 5 = 0101</math>  <math>S = 2 = 0010</math>  <math>-S = 1110</math></p>
$  \begin{array}{r}  1011 = -5 \\  +\underline{1110} = -2 \\  \hline  11001 = -7  \end{array}  $ <p>(c) <math>M = -5 = 1011</math>  <math>S = 2 = 0010</math>  <math>-S = 1110</math></p>	$  \begin{array}{r}  0101 = 5 \\  +\underline{0010} = 2 \\  0111 = 7  \end{array}  $ <p>(d) <math>M = 5 = 0101</math>  <math>S = -2 = 1110</math>  <math>-S = 0010</math></p>
$  \begin{array}{r}  0111 = 7 \\  +\underline{0111} = 7 \\  1110 = \text{Overflow}  \end{array}  $ <p>(e) <math>M = 7 = 0111</math>  <math>S = -7 = 1001</math>  <math>-S = 0111</math></p>	$  \begin{array}{r}  1010 = -6 \\  +\underline{1100} = -4 \\  \hline  10110 = \text{Overflow}  \end{array}  $ <p>(f) <math>M = -6 = 1010</math>  <math>S = 4 = 0100</math>  <math>-S = 1100</math></p>

Şekil 10.4 İkiye Tümleyen Gösteriminde Sayıların Çıkarılması ( $M - S$ )

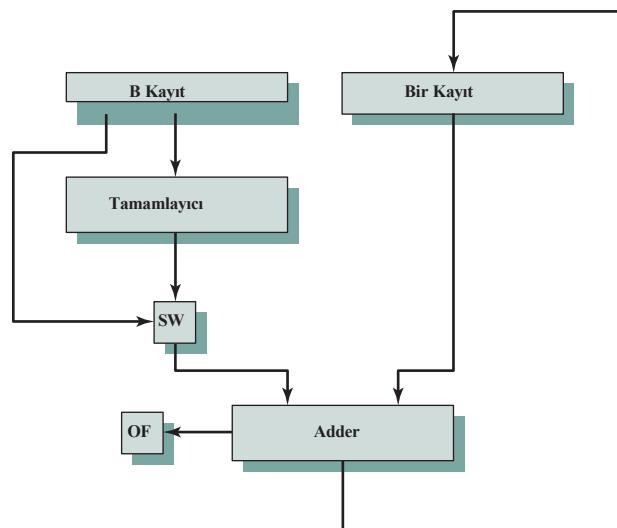


**Şekil 10.5** İkiye Tümleyen Tam Sayıların Geometrik Gösterimi

Şekil 10.5'te gösterildiği gibi geometrik bir tasvire [BENH92] bakarak ikili tümleyen toplama ve çıkarma işlemlerine ilişkin bazı bilgiler edinilebilir. Şeklin her bir bölümünün üst yarısındaki daire, sayı doğrusunun uygun bölümünün seçilmesi ve üç noktalarının birlleştirilmesiyle oluşturulmuştur. Sayılar bir daire üzerinde yerleştirildiğinde, herhangi bir sayının ikili tümleyeninin o sayının yatay olarak karşısında olduğuna dikkat ediniz (kesikli yatay çizgilerle gösterilmiştir). Daire üzerindeki herhangi bir sayıdan başlayarak,  $k$  konumunu saat yönünde hareket ettirerek bu sayıya pozitif  $k$  ekleyebilir (veya negatif  $k$  çıkarabiliriz) ve  $k$  konumunu saat yönünün tersine hareket ettirerek bu sayıdan pozitif  $k$  çıkarabiliriz (veya negatif  $k$  ekleyebiliriz). Eğer bir aritmetik işlem üç noktaların birleştiği noktadan geçişle sonuçlanırsa, yanlış bir yanıt verilir (taşma).

Şekil 10.3 ve 10.4'teki örneklerin **TÜMÜ** Şekil 10.5'teki daire içinde kolayca izlenebilir.

Şekil 10.6 toplama ve çıkarma işlemlerini gerçekleştirmek için gereken veri yollarını ve donanım elemanlarını göstermektedir. Merkezi eleman, toplama için önceki iki sayı gönderilen ve bir toplam ve bir taşıma göstergesi üreten bir ikili toplayıcıdır. İkili toplayıcı iki sayıyı işaretsız tam sayılar olarak ele alır. (Bir toplayıcının mantıksal uygulaması Bölüm 11'de verilmiştir.) Toplama işlemi için iki sayı, bu durumda **A** ve **B** kaydedicileri olarak adlandırılan iki kaydediciden toplayıcıya önceki gönderilir. Sonuç bu yazmaçlardan birinde ya da üçüncü bir yazmaça saklanabilir. Taşıma göstergesi 1 bitlik bir taşıma bayrağından saklanır ( $0 = \text{taşma yok}; 1 = \text{taşma}$ ). Alt çekiş için, alt üç (**B** kaydedicisi) bir ikilik tamamlayıcıdan geçirilir, böylece ikilik tamamlayıcı toplayıcıya sunulur. Şekil 10.6'nın yalnızca



OF = Taşma biti

SW = Anahtar (toplama veya çıkarmayı seçin)

Şekil 10.6 Toplama ve Çıkarma için Donanım Blok Diyagramı

veri yolları. İşlemin toplama veya çıkarma olmasına bağlı olarak tamamlayıcının kullanılıp kullanılmayacağını kontrol etmek için kontrol sinyallerine ihtiyaç vardır.

### Carpma İşlemi

Toplama ve çıkarma ile karşılaştırıldığında, çarpma işlemi ister donanım ister yazılımda gerçekleştirilsin karmaşık bir işlemidir. Çeşitli bilgisayarlarla çok çeşitli algoritmalar kullanılmıştır. Bu alt bölümün amacı, okuyucuya tipik olarak benimsenen yaklaşım türü hakkında bir fikir vermektedir. İki işaretsiz (negatif olmayan) tam sayının çarpılması gibi daha basit bir probleme başlayacağız ve daha sonra ikiye tümleyen gösterimindeki sayıların çarpılması için en yaygın tekniklerden birine bakacağız.

**İŞARETSİZ TAMSAYILAR** Şekil 10.7, kağıt ve kalem kullanılarak gerçekleştirilebileceği gibi, işaretsiz ikili tamsayıların çarpımını göstermektedir. Birkaç önemli gözlem yapılabilir:

1. Çarpma işlemi, çarpanındaki her basamak için bir tane olmak üzere kısmi çarpımların oluşturulmasını içerir. Bu kısmi ürünler daha sonra nihai ürünü üretmek için toplanır.

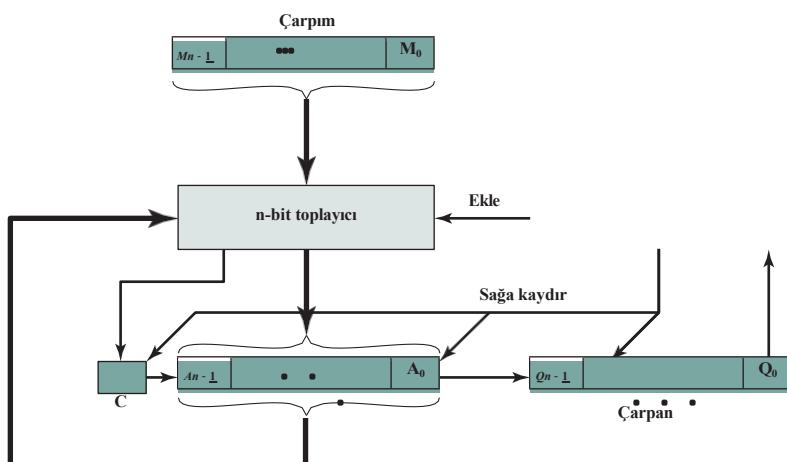
$  \begin{array}{r}  1011 \\  \times 1101 \\  \hline  1011 \\  0000 \\  1011 \\  1011 \\  \hline  10001111  \end{array}  $	<b>Carpım (11)</b> <b>Çarpan (13)</b>  <b>Kısmi ürünler</b>  <b>Ürün (143)</b>
--	---

Şekil 10.7 İşaretsiz İkili Tamsayıların Çarpımı

2. Kısımlı çarpımlar kolayca tanımlanır. Çarpan biti 0 olduğunda, kısımlı çarpım 0'dır. Çarpan 1 olduğunda, kısımlı çarpım çarpılan değerdir.
3. Toplam çarpım, kısımlı çarpımların toplanmasıyla elde edilir. Bu işlem için, birbirini izleyen her kısımlı çarpım, bir önceki kısımlı çarpıma göre bir konum sola kaydırılır.
4. İki  $n$  bitlik ikili tamsayıının çarpımı,  $2n$  bit uzunluğuna kadar bir çarpımla sonuçlanır (örneğin,  $11 * 11 = 1001$ ).

Kalem ve kağıt yaklaşımıyla karşılaşıldığında, bilgisayarlı çarpmaya işlemini daha verimli hale getirmek için yapabileceğimiz birkaç şey vardır. İlk olarak, sonuna kadar beklemek yerine kısımlı çarpımlar üzerinde çalışan bir toplama işlemi gerçekleştirebiliriz. Bu, tüm kısımlı çarpımların saklanması ihtiyacını ortadan kaldırır; daha az kaydediciye ihtiyaç duyulur. İkinci olarak, kısımlı ürünlerin oluşturulmasında biraz zaman kazanabiliyoruz. Çarpandaki her 1 için bir toplama ve bir kaydırma işlemi gereklidir; ancak her 0 için yalnızca bir kaydırma .

Şekil 10.8a bu önlemleri kullanan olası bir uygulamayı göstermektedir. Çarpan ve çarpılan iki yazmaca (Q ve M) yüklenir. Üçüncü bir



(a) Blok diyagramı

C	A	Q	M		İlk değerler
0	0000	1101	1011		
0	1011	1101	1011	Ekle	Birinci döngü
	0101	1110	1011	Vardiyा	
0	0010	1111	1011	Vardiyा	İkinci döngü
	1101	1111	1011	Ekle	
0	0110	1111	1011	Vardiyा	Üçüncü döngü
	0001	1111	1011	Ekle	
1	1000	1111	1011	Vardiyा	Dördüncü döngü
	0				

(b) Şekil 10.7'den örnek (A, Q'daki ürün)

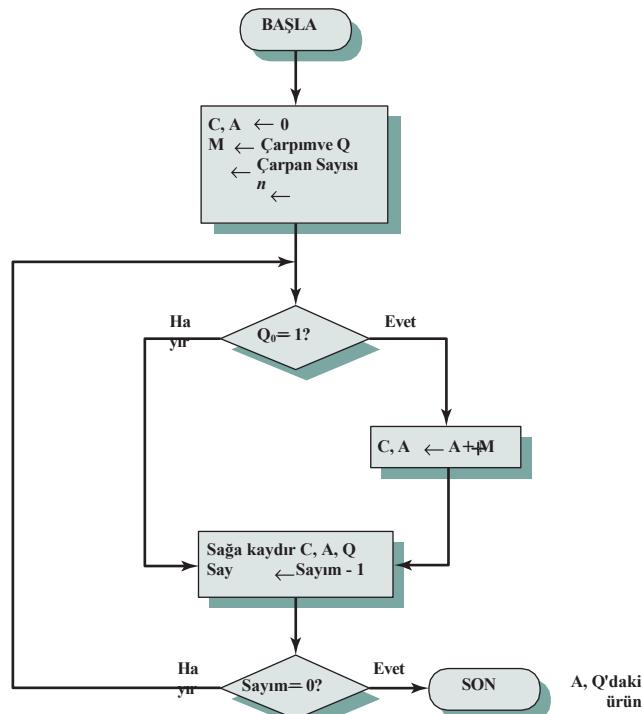
**Şekil 10.8** İşaretsiz İkili Çarpmaya Uygun Donanım

A yazmacına da ihtiyaç vardır ve başlangıçta 0 olarak ayarlanır. 1 bitlik bir C yazmacı da vardır, başlangıçta 0 olarak ayarlanır ve toplama işleminden kaynaklanan potansiyel bir taşıma bitini tutar. Çarpanın çalışması aşağıdaki gibidir. Kontrol mantığı çarpanın bitlerini teker teker okur.  $Q_0$  1 ise, çarpanın A yazmacına eklenir ve sonuç A yazmacında saklanır, C biti taşıma için kullanılır. Daha sonra C, A ve Q yazmaçlarının tüm bitleri bir bit sağa kaydırılır, böylece C biti  $A_{n-1}e$ ,  $A_0 Q_{(n)-1}e$ 'ye gider ve  $Q_{(0)}$  kaybolur.  $Q_0$  0 ise toplama işlemi yapılmaz, sadece kaydırma yapılır. Bu işlem orijinal çoklu çarpanın her biti için tekrarlanır. Elde edilen  $2n-bit$  çarpım A ve Q yazmaçlarında bulunur. İşlemenin akış şeması Şekil 10.9'da gösterilmiştir ve bir örnek Şekil 10.8b'de verilmiştir. İkinci döngüde, çarpan biti 0 olduğunda, toplama işlemi olmadığına dikkat edin.

**İKİ TAMLAYANLI ÇOĞALTMA** Toplama ve çıkarma işlemlerinin, sayıları işaretetsiz tamsayılar olarak ele alarak iki tümleyenli gösterimde gerçekleştirilebileceğini gördük. Şöyleden düşünün

$$\begin{array}{r} 1001 \\ + 0011 \\ \hline 1100 \end{array}$$

Bu sayılar işaretetsiz tamsayılar olarak kabul edilirse, 12 (1100) elde etmek için 9 (1001) artı 3 0011 ekliyoruz. İkiye tümleyen tamsayılar olarak, sunuları ekliyoruz  
- 7 (1001) ile 3 (0011) arasında - 4 (1100) elde edilir.



Şekil 10.9 İşaretetsiz İkili Çarpma için Akış Şeması

$  \begin{array}{r}  1011 \\  \times 1101 \\  \hline  00001011 \\  00000000 \\  00101100 \\  01011000 \\  \hline  10001111  \end{array}  $	$  \begin{array}{l}  1011 \times 1 \times 2^0 \\  1011 \times 0 \times 2^1 \\  1011 \times 1 \times 2^2 \\  1011 \times 1 \times 2^3 \\  \hline  10001111  \end{array}  $
--	---

**Şekil 10.10** 8-Bit Sonuç Veren İki İşaretsiz 4-Bit Tamsayının Çarpması

Ne yazık ki, bu basit şema çarpma işlemi için çalışmayaçaktır. Bunu görmek için Şekil 10.7'yi tekrar düşünün. 143 (10001111) elde etmek için 11 (1011) ile 13'ü (1101) çarpmıştık. Bunları ikiye tümleyen sayılar olarak yorumlarsak,  $-5(1011)$  çarpı  $-3(1101)$  eşittir  $-113(10001111)$  elde ederiz. Bu örnek, hem çarpan hem de çarpılan negatifse doğrudan çarmanın işe yaramayacağını göstermektedir. Aslında, çarpılan ya da çarpan negatif ise çalışmayaçaktır. Bu ifadeyi gerekçelendirmek için Şekil 10.7'ye geri dönmemiz ve  $2^n$ in kuvvetleriyle yapılan işlemler açısından ne yapıldığını açıklamamız gereklidir. Herhangi bir işaretsiz ikili sayının  $2^n$ in kuvvetlerinin toplamı olarak ifade edilebileceğini hatırlayın,

$$1101 = 1 * 2^3 + 1 * 2^2 + 0 * 2^{(1)} + 1 * 2^{(0)} = 2^3 + 2^2 + 2^{(0)}$$

Ayrıca, ikili bir sayının  $2^n$  ile çarpılması, bu sayının  $n$  bit sola kaydırılmasıyla gerçekleştirilebilir. Bunu akılda tutarak, Şekil 10.10, Şekil 10.7'de olduğu gibi çarpma işlemiyle kısmi çarpımların oluşturulmasını açık hale getirir. Şekil 10.10'daki tek fark, kısmi çarpımların  $n$ -bitlik çarpımdan üretilen  $2n$ -bitlik sayılar olarak görülmesi gerektiğini kabul etmemesidir.

Böylece, işaretsiz bir tamsayı olarak, 4 bitlik 1011 çarpımı 8 bitlik bir sözcükte 00001011 olarak saklanır. Her kısmı çarpım ( $2^0$  hariç) bu sayının sola kaydırılmış halinden oluşur ve sağıdaki boş yerler sıfırlarla doldurulur (örneğin, iki basamak sola kaydırma 00101100 sonucunu verir).

Şimdi basit çarmanın çarpılanın negatif olması durumunda çalışmayıcağını gösterebiliriz. Sorun, negatif çarpımın kısmı çarpım olarak her bir katısının  $2n$  bitlik bir alanda negatif bir sayı olması gereklidir; kısmi çarpımların işaret bitleri aynı olmalıdır. Bu, 1001 ile 0011'in çarpımını gösteren Şekil 10.11'de gösterilmiştir. Bunlar tamsayılar olarak ele alınırsa,  $9 * 3 = 27$  çarpımı basitçe ilerler. Ancak, 1001 yorumlanırsa

$  \begin{array}{r}  1001 \quad (9 \\  \times 0011 \quad (3 \\  \hline  00001001 \quad 1001 \times 2^0 \\  00010010 \quad 1001 \times 2^1 \\  00011011 \quad (27 \\  \hline  \end{array}  $	$  \begin{array}{r}  1001 \quad (-7 \\  \times 0011 \quad (3 \\  \hline  11111001 \quad (-7 \times 2^0 = (-7 \\  11110010 \quad (-7 \times 2^1 = (-14 \\  11101011 \quad (-21 \\  \hline  \end{array}  $
---	--

(a) İşaretsiz tamsayılar

(b) İkiye tümleyen tamsayılar

**Şekil 10.11** İşaretsiz ve İkiye Tümleyen Tamsayıların Çarpımının Karşılaştırılması

ikiye tümleyen değeri - 7 ise, her bir kısmi çarpım Şekil 10.11b'de gösterildiği gibi  $2n$  (8) bitlik negatif ikiye tümleyen sayısını olmalıdır. Bunun, her bir kısmi çarpımın sola doğru ikili 1'lerle doldurulmasıyla gerçekleştirildiğini unutmayın.

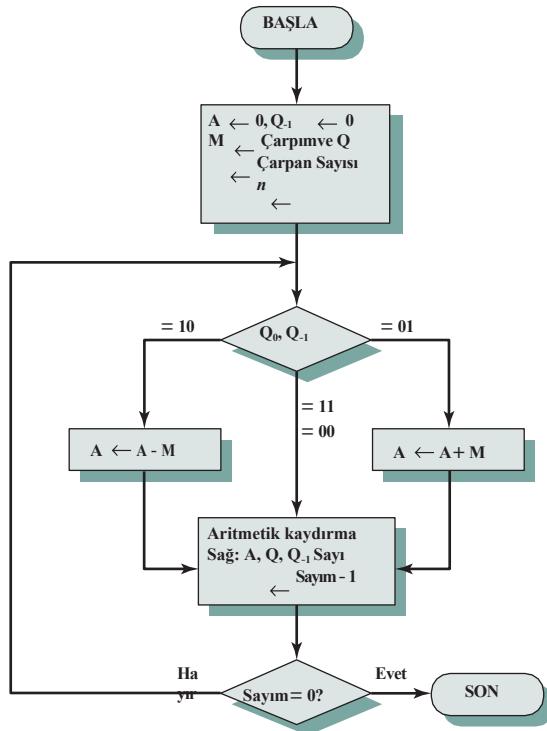
Çarpan negatifse, doğrudan çarpmaya işlemi de çalışmamış olacaktır. Bunun nedeni, çarpanın bitlerinin artık gerçekleştirmesi gereken kaydırma veya çarpmaya işlemlerine karşılık gelmemesidir. Örneğin, 4 bitlik ondalık sayı - 3, ikiye tümleyen olarak 1101 şeklinde yazılır. Eğer basitçe her bit pozisyonuna göre kısmi çarpımlar alsaydık, aşağıdaki gibi yazışmaya sahip olurduk:

$$1101 \cdot 4 = -(1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0) = -(2^3 + 2^2 + 2^0)$$

Aslında, istenen şey  $-(2^1 + 2^0)$ dir. Dolayısıyla bu çarpan, açıkladığımız şekilde doğrudan kullanılamaz.

Bu ikilemden kurtulmanın birkaç yolu vardır. Bunlardan biri, hem çarpanı hem de çarpılanı pozitif sayılarla dönüştürmek, çarpmayı gerçekleştirmek ve ardından yalnızca iki orijinal sayının işaretini farklıysa sonucun ikiye tümleyenini almak olabilir. Uygulayıcılar bu son dönüştürme adımı gerçekleştirmeyen teknikleri kullanmayı tercih etmişlerdir. Bunların en yaygın olanlarından biri Booth'un algoritmasıdır [BOOTH51]. Bu algoritma, daha basit bir yaklaşımına göre çarpmayı hızlandırmaya avantajına da sahiptir.

Booth'un algoritması Şekil 10.12'de gösterilmiştir ve aşağıdaki gibi tanımlanabilir. Daha önce olduğu gibi, çarpan ve çarpılan Q ve M yazımcalarına yerleştirilir,



**Şekil 10.12** İki Tümleyenli Çarpmaya Uygun Booth Algoritması

A 0000	Q 0011	$Q_{-1}$ 0	M 0111	İlk değerler	
1001	0011	0	0111	$A \leftarrow A - M$	} İlk döngü
1100	1001	1	0111	Vardiya	
1110	0100	1	0111	Vardiya	} İkinci döngü
0101	0100	1	0111	$A \leftarrow +M$	
0010	1010	0	0111	Vardiya	} Üçüncü döngü
0001	0101	0	0111	Vardiya	

Şekil 10.13 Booth Algoritması Örneği (7\*3)

sırasıyla. Ayrıca Q yazmacının en az anlamlı bitinin ( $Q_0$ ) mantıksal olarak sağına yerleştirilen ve  $Q_{-1}$  olarak adlandırılan 1 bitlik bir yazmaç vardır; kullanımı kısaca açıklanmaktadır. Çarpma işleminin sonuçları A ve Q yazmaclarında görünecektir. A ve  $Q_{-1}$  0 olarak başlatılır. Daha önce olduğu gibi, kontrol mantığı çarpanın bitlerini teker teker tarar. Şimdi, her bit incelendiğinde, sağındaki bit de. İki bit aynıysa (1-1 veya 0-0), A, Q ve  $Q_{(-1)}$  kayıtlarının tüm bitleri sağdaki 1 bite kaydırılır. İki bit farklıysa, iki bitin 0-1 veya 1-0 olmasına bağlı olarak çarpım A kaydedicisine eklenir veya A kaydedicisinden çıkarılır. Toplama veya çıkarma işleminin ardından sağa kaydırma gerçekleşir. Her iki durumda da sağa kaydırma, A'nın en soldaki biti, yani  $A_{n-1}$ , sadece  $A_{(n)(-2)}$ ye kaydırılmakla kalmaz, aynı zamanda  $A_{(n)(-1)}$ de kalır. Bu, A ve Q'daki sayının işaretini korumak için gereklidir. İşaret bitini koruduğu için **aritmetik kaydırma** olarak bilinir.

Şekil 10.13, Booth'un 7'nin 3 ile çoku çarpımına ilişkin algoritmasındaki olayları dizisini göstermektedir. Daha derli toplu olarak, aynı işlem Şekil 10.14'a'da gösterilmiştir. Şekil 10.14'ün geri kalanı algoritmanın diğer örneklerini vermektedir. Görülebileceği gibi, pozitif ve negatif sayıların herhangi bir kombinasyonu ile çalışır. Ayrıca algoritmanın verimliliğine de dikkat edin. Blok başına ortalama sadece bir toplama veya çıkarma işlemi ile 1'lerin veya 0'ların blokları .

$  \begin{array}{r}  0111 \\  \times 0011 \\  \hline  11111001 \\  0000000 \\  \hline  000111 \\  00010101  \end{array}  $	$  \begin{array}{r}  0111 \\  \times 1101 \\  \hline  11111001 \\  0000111 \\  \hline  111001 \\  11101011  \end{array}  $
--	--

(a)  $(7) \times (3) = (21)$ (b)  $(7) \times (-3) = (-21)$ 

$  \begin{array}{r}  1001 \\  \times 0011 \\  \hline  00000111 \\  0000000 \\  \hline  111001 \\  11101011  \end{array}  $	$  \begin{array}{r}  1001 \\  \times 1101 \\  \hline  00000111 \\  1111001 \\  \hline  000111 \\  00010101  \end{array}  $
--	--

(c)  $(-7) \times (3) = (-21)$ (d)  $(-7) \times (-3) = (21)$ 

Şekil 10.14 Booth Algoritmasını Kullanan Örnekler

Booth'un algoritması neden çalışır? İlk olarak pozitif bir çoklu çarpan durumunu ele alalım. Özellikle, 0'larla yuvarlanmış 1'lerden oluşan bir bloktan oluşan pozitif bir çarpan düşünün (örneğin, 00011110). Bildiğimiz gibi, çarpma işlemi çarpılanın uygun şekilde kaydırılmış kopyalarının toplanmasıyla gerçekleştirilebilir:

$$\begin{aligned} M * (00011110) &= M * (2^4 + 2^3 + 2^2 + 2^1) \\ &= M * (16 + 8 + 4 + 2) \\ &= M * 30 \end{aligned}$$

Şunu gözlemlersek bu tür işlemlerin sayısı ikiye indirilebilir

$$2^n + 2^{(n)-1} + \dots + 2^{(n)-(K)} = 2^{(n)+1} - 2^{(n)-(K)} \quad (10.3)$$

$$\begin{aligned} M * (00011110) &= M * 2^5 - 2^1) \\ &= M * (32 - 2) \\ &= M * 30 \end{aligned}$$

Böylece çarpım, çoklu bileşenin bir toplama ve bir çıkarma işlemiyle oluşturulabilir. Bu şema, tek bir 1'in bir blok olarak ele alındığı durum da dahil olmak üzere, bir çarpandaki herhangi bir sayıda 1 bloğuna kadar uzanır.

$$\begin{aligned} M * (01111010) &= M * (2^6 + 2^5 + 2^4 + 2^3 + 2^1) \\ &= M * (2^7 - 2^3 + 2^2 - 2^1) \end{aligned}$$

Booth'un algoritması, bloğun ilk 1'i ile karşılaşıldığında (1-0) bir çıkarma ve bloğun sonu ile karşılaşıldığında (0-1) bir toplama işlemi gerçekleştirerek bu şemaya uymaktadır.

Aynı şemanın negatif bir çarpan için de çalıştığını göstermek için aşağıdakileri gözlemlememiz gereklidir.  $X$ , ikiye tümleyen gösteriminde negatif bir sayı olsun:

$$X \text{in gösterimi} = 51x_{n-2}x_{(n)-(3)} \dots x_1x_0 6$$

O halde  $X$ 'in değeri aşağıdaki gibi ifade edilebilir:

$$X = -2^{n-1} + (x^{(n)}_{(n)2} * 2^{(n)-(2)}) + (x^{(n)}_{(n)3} * 2^{(n)-(3)}) + \dots + (x^1 * 2^1) + (x_0 * 2^0) \quad (10.4)$$

Okuyucu, algoritmayı Tablo 10.2'deki sayılarla uygulayarak bunu doğrulayabilir.

$X$ 'in en soldaki biti 1'dir, çünkü  $X$  negatiftir. En soldaki 0'ın  $k$ . konumda olduğunu varsayılmı. Böylece,  $X$  şu biçimdedir

$$X \text{in gösterimi} = 5111 \dots 10x_{k-1}x_{(k)-(2)} \dots x_1x_0 6 \quad (10.5)$$

O zaman  $X$ 'in değeri

$$X = -2^{n-1} + 2^{n-(2)} + \dots + 2^{(k+1)} + (x^k_{(k)1} * 2^{(k)-(1)}) + \dots + (x_0 * 2^0) \quad (10.6)$$

Denklem (10.3)'ten şunu söyleyebiliriz

$$2^{n-2} + 2^{n-3} + \dots + 2^{k-1} = 2^{n-1} - (2)^{k-1}$$

### Yeniden Düzenleme

$$- 2^{n-1} + 2^{(n)(-)(2)} + 2^{(n)(-)(3)} + \text{C} + 2^{k+1} = -2^{(k)+(1)} \quad (10.7)$$

Denklem (10.7)'yi Denklem (10.6)'da yerine koyduğumuzda

$$X = -2^{(k)+1} + (x^{(k)}_{(-)(1)}) * 2^{(k)(-)(1)} + \text{C} + (x_0 * 2^0) \quad (10.8)$$

Sonunda Booth'un algoritmasına dönebiliriz.  $X$ 'in gösterimini [Denklem (10.5)] hatırlarsak,  $x_0$ 'dan en soldaki 0'a kadar olan tüm bitlerin doğru şekilde ele alındığı açıktır çünkü bunlar Denklem (10.8)'deki tüm terimleri üretirler ancak  $(-2^{k+1})$  ve dolayısıyla uygun formdadırlar. Algoritma en soldaki 0'ı tarayıp bir sonraki 1 ile karşılaşlığında  $(2^{(k)+1})$ , 1-0 geçisi gerçekleşir ve bir çıkarma işlemi gerçekleşir  $(-2^{(k)+(1)})$ . Bu, Denklem (10.8)'de kalan terimdir.

Örnek olarak, bir çarpanın  $(-6)$  ile çarpımını düşünün. İkiye tümleyen gösteriminde, 8 bitlik bir sözcük kullanılarak,  $(-6)$  11111010 olarak gösterilir. Denklem (10.4) ile şunu biliyoruz

$$-6 = -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1$$

Okuyucu bunu kolayca doğrulayabilir. Böylece,

$$M * (11111010) = M * (-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1)$$

Denklem (10.7)'yi kullanarak,

$$M * (11111010) = M * (-2^3 + 2^1)$$

Okuyucu bunun hala  $M * (-6)$  olduğunu doğrulayabilir. Son olarak, daha önceki akıl yürütmemizi takip ederek,

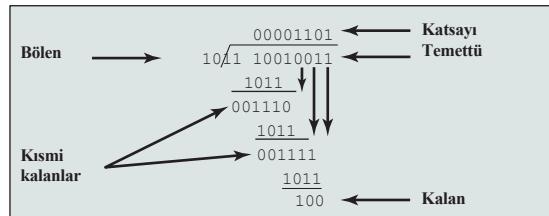
$$M * (11111010) = M * (-2^3 + 2(2) - 2^1)$$

Booth'un algoritmasının bu şemaya uygun olduğunu görebiliriz. İlk 1 ile karşılaşlığında (10) bir alt çekme, (01) ile karşılaşlığında bir toplama ve son olarak bir sonraki 1'ler bloğunun ilk 1'i ile karşılaşlığında başka bir çıkışma işlemi gerçekleştirir. Böylece, Booth'un algoritması daha basit bir algoritmaya göre daha az toplama ve çıkışma işlemi gerçekleştirir.

### Bölüm

Bölme işlemi çarpma işleminden biraz daha karmaşıktır ancak aynı genel ilkelere dayanır. Daha önce olduğu gibi, algoritmanın temeli kağıt-kalem yaklaşımıdır ve işlem tekrarlayan kaydırma ve toplama ya da çıkarmayı içerir. Şekil 10.15, işaretsiz ikili tamsayıların uzun bölme işleminin bir örneğini göstermektedir.

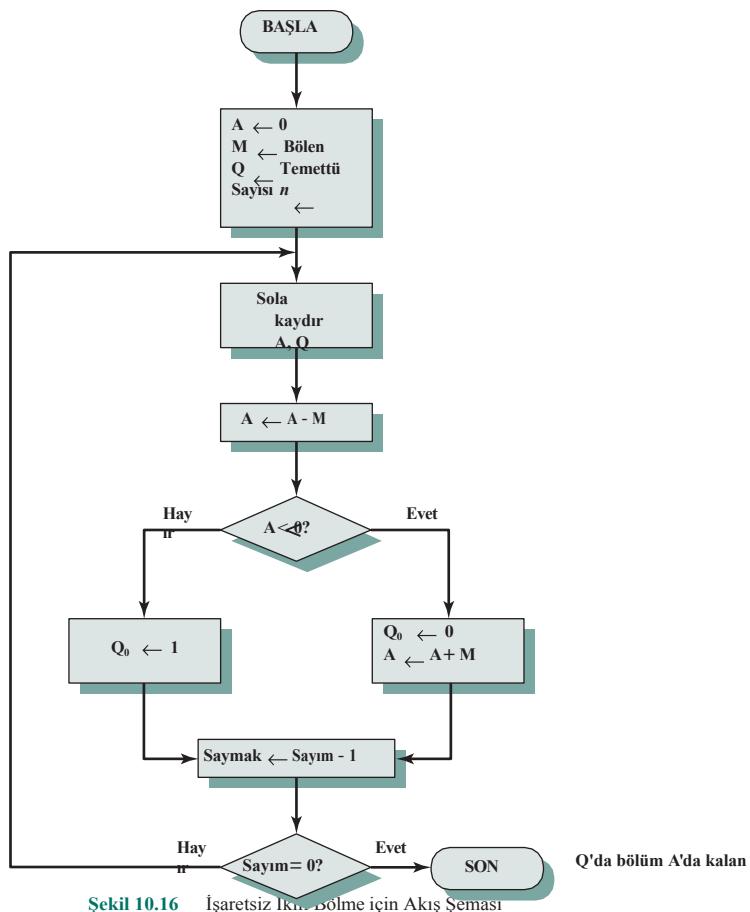
Süreci ayrıntılı olarak açıklamak öğreticidir. İlk olarak, bölenin bitleri soldan sağa doğru incelenir, ta ki incelenen bitler kümesi bölene eşit veya daha büyük bir sayıyı temsil edene kadar; bu bölenin sayımı bölebilmesi olarak adlandırılır. Bu olay gerçekleşene kadar, bölüme soldan sağa doğru 0'lar yerleştirilir. Bu olay gerçekleştiğinde, bölüme bir 1 yerleştirilir ve bölen kısmi temettüden çıkarılır. Sonuç *kısmi kalan* olarak *adlandırılır*.



**Şekil 10.15** İşaretsiz İkili Tamsayıların Bölünmesine Örnek

Bu noktadan itibaren bölme işlemi döngüsel bir model izler. Her döngüde, sonuç bölen sayıdan büyük veya ona eşit olana kadar kısmi kalanı bölen sayıdan ek bitler eklenir. Daha önce olduğu gibi, yeni bir kısmi kalan üretmek için bölen bu sayıdan çıkarılır. İşlem, bölenin tüm bitleri tükene kadar devam eder.

Şekil 10.16 uzun bölme işlemine karşılık gelen bir makine algoritmasını göstermektedir. Bölün M kaydına, bölünen Q kaydına yerleştirilir. Şu anda



A	Q	
0000	0111	İlk değer
0000	1110	Vardiya
<u>1101</u>		Çıkarma işlemi için 0011'in ikiye tümleyenini kullanın
1101	1110	Çıkarma
0000		Geri yükleyin, $Q_0 = 0$ olarak ayarlayın
0001	1100	Vardiya
<u>1101</u>		Çıkarma
1110	1100	Geri yükleyin, $Q_0 = 0$ olarak ayarlayın
0001		
0011	1000	Vardiya
<u>1101</u>	1001	Çıkarm, $Q_0 = 1$ olarak ayarlayın
0000		
0001	0010	Vardiya
<u>1101</u>		Çıkarma
1110	0010	Geri yükleyin, $Q_0 = 0$ olarak ayarlayın
0001		

Şekil 10.17 İkili Tümleyen Bölmesinin Geri Yüklenmesi Örneği (7/3)

Her adımda, A ve Q yazmacıları birlikte 1 bit sola kaydırılır. A'nın kısmi kalanı bölüp bölmemiğini belirlemek için M, A'dan çıkarılır.<sup>3</sup> Eğer bölerse,  $Q_0$  1 biti alır. Aksi takdirde,  $Q_0$  0 biti alır ve önceki değeri geri yüklemek için M'nin A'ya geri eklenmesi gereklidir. Daha sonra sayı azaltılır ve işlem n adım boyunca devam eder. Sonunda, bölüm Q yazmacında ve kalan A yazmacındadır.

Bu işlem biraz zorlukla negatif sayılar da genişletilebilir. Burada ikiye tümleyen sayılar için bir yaklaşım vereceğiz. Bu yaklaşımın bir örneği Şekil 10.17'de gösterilmektedir.

Algoritma, bölen  $V$  ve bölünen  $D$ 'nin pozitif olduğunu ve  $V \cdot 6 \cdot D$  olduğunu varsayar. Eğer  $V \cdot 6 \cdot D$  ise, bölüm  $Q = 1$  ve kalan  $R = 0$  olur. Eğer  $V \cdot 7 \cdot D$  ise,  $Q = 0$  ve  $R = D$  olur:

1. Bölenin ikiye tümleyenini M yazmacına yükleyin; yani M yazmacı bölenin negatifini içerir. Bölüneni A, Q yazmaclarına yükleyin. Bölten 2n bitlik pozitif bir sayı olarak ifade edilmelidir. Böylece, örneğin, 4 bit 0111 00000111 olur.
2. A, Q'yı 1 bit sola kaydırın.
3. A d A - M işlemini gerçekleştirin. Bu işlem A'nın içeriğinden böleni çıkarır.
4. a. Sonuç negatif değilse (A'nın en anlamlı biti= 0),  $Q_0 \text{ d} 1$  olarak ayarlayın.  
b. Sonuç negatifse (A'nın en anlamlı biti= 1),  $Q_0 \text{ d} 0$  olarak ayarlanır ve A'nın önceki değeri geri yüklenir.
5. Q'da bit pozisyonu sayısı kadar 2'den 4'e kadar olan adımları tekrarlayın.
6. Kalan A'da ve bölüm Q'da yer alır.

<sup>3</sup>Bu işaretetsiz tamsayıların çıkartılmasıdır. En anlamlı bitin ödünc alınmasını gerektiren bir sonuç negatif bir sonuçtır.

Negatif sayılarla başa çıkmak için, kalanın şu şekilde tanımlandığını kabul ederiz

$$D = Q * V + R$$

, kalan, önceki denkemin geçerli olması için gereken  $R$  değeridir. Aşağıdaki tamsayı bölme örneklerini,  $D$  ve  $V$  işaretlerinin tüm olası kombinasyonlarıyla birlikte ele alın:

$D = 7$	$V = 3$	<b>1</b>	$Q = 2$	$R = 1$
$D = 7$	$V = -3$	<b>1</b>	$Q = -2$	$R = 1$
$D = -7$	$V = 3$	<b>1</b>	$Q = -2$	$R = -1$
$D = -7$	$V = -3$	<b>1</b>	$Q = 2$	$R = -1$

Okuyucu Şekil 10.17'den  $(-7)/(3)$  ve  $(7)/(-3)$ 'ün farklı kalanlar ürettiğine dikkat edecektr.  $Q$  ve  $R$ 'nin büyülüklerinin giriş işaretlerinden etkilenmediğini ve  $Q$  ve  $R$ 'nin işaretlerinin  $D$  ve  $R$ 'nin işaretlerinden kolayca türetilmesini görüyoruz.

$V$ . Özellikle,  $\text{sign}(R) = \text{sign}(D)$  ve  $\text{sign}(Q) = \text{sign}(D) * (V)$ . Bu nedenle, ikiye tümleyen bölme işlemini yapmanın bir yolu, işlenenleri işaretsız değerlere dönüştürmek ve sonunda, gerektiğinde tümleme yoluyla işaretleri hesaba katmaktadır. Bu, geri yükleme bölme algoritması için tercih edilen yöntemdir [PARH10].

## 10.4 FLOATING-POINT SUNUMU

### Prensipler

Sabit noktalı bir gösterimle (örneğin, ikiye tümleyen), 0 veya 0'a yakın merkezli bir dizi pozitif ve negatif tamsayıyı temsil etmek mümkündür. Sabit bir ikili veya radiks noktası varsayıarak, bu format kesirli bir bileşene sahip sayıların da temsil edilmesine izin verir.

Bu yaklaşımın sınırlamaları vardır. Çok büyük sayılar temsil edilemediği gibi çok küçük kesirler temsil edilemez. Ayrıca, iki büyük sayının bölümnesinde bölümün kesirli kısmı kaybolabilir.

Ondalık sayılar için bu sınırlamayı bilimsel gösterim kullanarak aşabılırız. Böylece, 976,000,000,000,000 9.76 \*  $10^{14}$  olarak gösterilebilir ve 0.0000000000000976 9.76 \*  $10^{-14}$  olarak gösterilebilir. Gerçekte yaptığımız şey, ondalık noktayı dinamik olarak uygun konuma kaydırarak ve bu ondalık noktayı takip etmek için 10'un üssünü kullanmaktır. Bu, çok büyük ve çok küçük sayıların yalnızca birkaç basamakla temsil edilmesini sağlar.

Aynı yaklaşım ikili sayılar için de kullanılabilir. Bir sayıyu şu şekilde temsil edebiliriz

$$\{S * B^E\}$$

Bu sayı üç alanlı bir ikili kelimedede saklanabilir:

- İşaret: artı veya eksı
- Anlam ve S
- Üs E



$1.1010001 \times 2^{10100} = 0.10010011 \cdot 101000100000000000000000 = 1.6328125 \times 2^{20}$   
 $-1.1010001 \times 2^{10100} = 1.10010011 \cdot 101000100000000000000000 = -1.6328125 \times 2^{20}$   
 $1.1010001 \times 2^{-10100} = 0.01101011 \cdot 101000100000000000000000 = 1.6328125 \times 2^{-20}$   
 $-1.1010001 \times 2^{-10100} = 1.01101011 \cdot 101000100000000000000000 = -1.6328125 \times 2^{-20}$

(b) Örnekler

**Şekil 10.18** Tipik 32-Bit Kayan Nokta Formatı

B tabanı örtüktür ve tüm sayılar için aynı olduğuundan saklanması gerekmektedir. Tipik olarak, radix noktasının en sol veya en anlamlı bitinin sağında olduğu varsayılmaktadır.

İkili kayan noktalı sayıların gösteriminde kullanılan ilkeler en iyi bir örnekle açıklanabilir. Şekil 10.18a tipik bir 32-bit kayan nokta formatını göstermektedir. En soldaki bit sayının işaretini saklar ( $=$ pozitif,  $=$ negatif). Üs değeri sonraki 8 bitte saklanır. Kullanılan gösterim, **önyargılı gösterim** olarak bilinir. Bias adı verilen sabit bir değer, gerçek üs değerini elde etmek için alandan çıkarılır. Tipik olarak, önyargı ( $2^{k-1} - 1$ ) değerine eşittir, burada  $k$  ikili üsdeki bit sayısıdır. Bu durumda, 8 bitlik alan 0 ile 255 arasındaki sayıları verir. 127 ( $2^7 - 1$ )'lık bir sapma ile, gerçek üs değerleri şu aralıktadır 127 ile + 128. Bu örnekte, tabanın 2 olduğu varsayılmıştır.

Tablo 10.2, 4 bitlik tamsayılar için yanlış gösterimi göstermektedir. Taraflı gösterimin bitleri işaretetsiz tamsayılar olarak ele alındığında, sayıların göreli büyüklüklerinin değişmediğine dikkat edin. Örneğin, hem yanlış hem de işaretetsiz gösterimlerde, en büyük sayı 1111 ve en küçük sayı 0000'dır. Bu durum işaret büyülüklüğü ya da ikiye tümleyen gösterimi için geçerli değildir. İki tabanlı gösterimin bir avantajı, negatif olmayan kayan noktalı sayıların karşılaştırma amacıyla tamsayı olarak ele alınabilmesidir.

Kelimenin son kısmı (bu durumda 23 bit) **anlamlıdır**.<sup>4</sup>

Herhangi bir kayan noktalı sayı birçok şekilde ifade edilebilir.

Aşağıdakiler eşdeğerdir, burada anlamlı ikili biçimde ifade edilir:

$$\begin{aligned} & 0.110 * 2^5 \\ & 110 * 2^2 \\ & 0.0110 * 2^6 \end{aligned}$$

Kayan noktalı sayılar üzerindeki işlemleri basitleştirmek için genellikle normalize edilmeleri gerekmektedir. **Normal** bir sayı, kayan noktalı **sayının** en anlamlı basamağının

<sup>4</sup>Bazen *significant* yerine kullanılan **mantissa** teriminin modası geçmiş olduğu düşünülmektedir. *Mantissa* aynı zamanda "bir logaritmanın kesirli kısmı" anlamına gelir, bu nedenle bu bağlamda kullanılmasından kaçınılmazı en iyisidir.

anlamlı bit sıfır değildir. Bu nedenle, 2 tabanı gösterimi için normal bir sayı, anlamlının en önemlidinin bir olduğu bir sayıdır. Daha önce de belirtildiği gibi, tipik konvansiyon radix noktasının solunda bir bit olmalıdır. Bu nedenle, sıfır olmayan bir normal sayı şu şekilde bir sayıdır

$$\{ 1.bbb \leftarrow b * 2^{e}$$

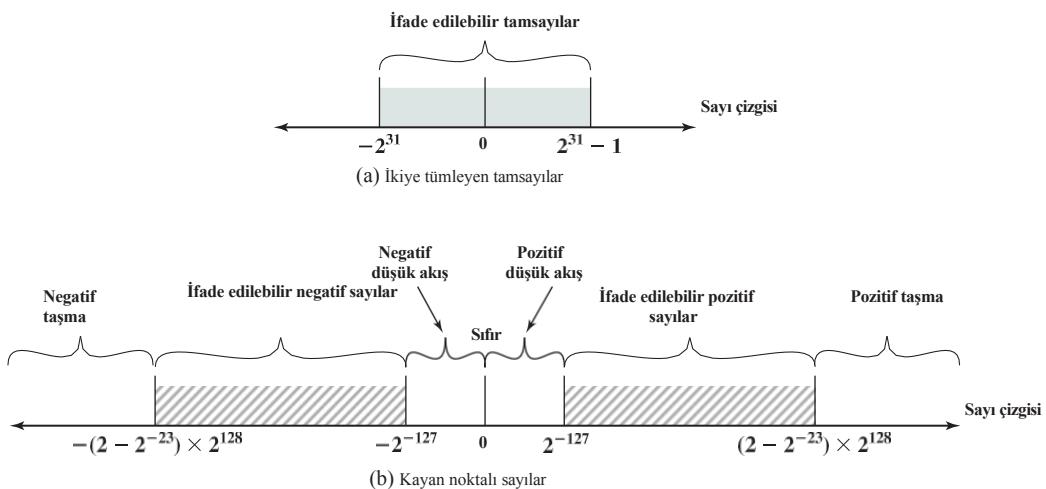
burada  $b$  ikili basamaklardan biridir (0 veya 1). En anlamlı bit her zaman bir olduğundan, bu biti saklamak gereksizdir; bunun yerine, örtüktür. Böylece, 23 bitlik alan, yarı açık aralıkta [1, 2) bir değere sahip 24 bitlik bir anlamlılığı saklamak için kullanılır. Normal olmayan bir sayı verildiğinde, radix noktası en soldaki 1 bitin sağına kaydırılarak ve üs buna göre ayarlanarak sayı normalleştirilebilir.

Şekil 10.18b bu formatta saklanan bazı sayı örneklerini vermektedir. Her örnek için, solda ikili sayı; ortada karşılık gelen bit paterni; sağda ise ondalık değer yer almaktadır. Aşağıdaki özelliklere dikkat edin:

- İşaret kelimenin ilk bitinde saklanır.
- Gerçek anlamlı ilk biti her zaman 1'dir ve anlamlı değer alanında saklanması gereklidir.
- 127 değeri, üs alanında saklanacak gerçek üssü eklenir.
- Taban 2'dir.

Karşılaştırma için, Şekil 10.19 32 bitlik bir sözcükte temsil edilebilecek sayı aralığını göstermektedir. İkiye tümleyen gösterimi kullanılarak,  $-2^{(31)}$  ile  $2^{(31)} - 1$  arasındaki tüm tamsayılar, toplam  $2^{(32)}$  farklı sayı için temsil edilebilir. Şekil 10.18'deki örnek kayan nokta formатıyla, aşağıdaki sayı aralıkları mümkündür:

- $(2 - 2^{-23})$  arasındaki negatif sayılar  $* 2^{128}$  ve  $-2^{-127}$
- $2^{-127}$  ile  $(2 - 2^{-23})$  arasındaki pozitif sayılar  $* 2^{128}$



Şekil 10.19 Tipik 32-Bit Formatlarında İfade Edilebilir Sayılar

Sayı doğrusu üzerindeki beş bölge bu aralıklara dahil değildir:

- $'$ -den küçük negatif sayılar ( $2 - 2^{-23}$ ) \*  $2^{128}$ , **negatif taşıma** olarak adlandırılır
- $2^{-(127)}$ -den büyük **negatif** sayılar, **negatif alt akış** olarak adlandırılır
- Sıfır
- $2^{-(127)}$ -den küçük **pozitif** sayılar, **pozitif düşük akış** olarak adlandırılır
- ( $2 - 2^{-23}$ )'ten büyük pozitif sayılar \*  $2^{128}$ , **pozitif taşıma** olarak adlandırılır

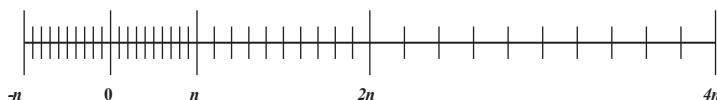
Ancak, göreceğimiz gibi, gerçek kayan nokta gösterimleri sıfırı belirtmek için özel bir bit modeli içerir. Taşıma, bir aritmetik işlem 128'lik bir üs ile ifade edilebileceğinden daha büyük bir mutlak değerle sonuçlandığında meydana gelir (örneğin,  $2^{120} * 2^{100} = 2^{220}$ ). Düşük akış, kesirli büyüklük çok küçük olduğunda meydana gelir (örneğin,  $2^{-120} * 2^{(-100)} = 2^{(-220)}$ ). Altın taşıma daha az ciddi bir sorundur çünkü sonuç genellikle tatmin edici bir şekilde 0'a yaklaştırılabilir.

Kayan nokta gösterimiyle daha fazla bireysel değeri temsil etmediğimize dikkat etmek önemlidir. Maksimum 32 bit ile temsil edilebilecek farklı değer sayısı hala  $2^{(32)}$ 'dir. Yaptığımız şey, bu sayıları biri pozitif diğeri negatif olmak üzere iki aralığa yaymaktadır. Pratikte, temsil edilmek istenen çoğu kayan noktalı sayı sadece yaklaşık olarak temsil edilir. Ancak, orta büyülükteki tamsayılar için gösterim tamdır.

Ayrıca, kayan noktalı gösterimde temsil edilen sayıların, sabit noktalı sayıarda olduğu gibi sayı doğrusu boyunca eşit aralıklarla yerleştirilmediğine dikkat edin. Şekil 10.20'de gösterildiği gibi, olası değerler orijin yakınında birbirine yaklaşır ve uzaklaşıkça birbirinden uzaklaşır. Bu, kayan noktalı matematiğin değiş tokuşlarından biridir. Birçok hesaplama tam olmayan sonuçlar üretir ve gösterimin temsil edebileceği en yakın değere yuvarlanması gereklidir.

Şekil 10.18'de gösterilen format türünde, aralık ve hassasiyet arasında bir denge vardır. Örnekte üsse 8 bit ve anlamlıya 23 bit ayrılmıştır. Üssün bit sayısını artırırsak, ifade edilebilir sayı aralığını genişletmiş oluruz. Ancak yalnızca sabit sayıda farklı değer ifade edilebildiğinden, bu sayıların yoğunluğunu ve dolayısıyla hassasiyeti azaltmış oluruz. Hem aralığı hem de hassasiyeti artırmanın tek yolu daha fazla bit kullanmaktır. Bu nedenle, çoğu bilgisayar en azından tek hassasiyetli sayılar ve çift hassasiyetli sayılar sunar. Örneğin, bir işlemci 64 bitlik bir tek hassasiyetli formатı ve 128 bitlik bir çift hassasiyetli formатı destekleyebilir.

Dolayısıyla, üs birimdeki bit sayısı ile anlamlı birimdeki bit sayısı arasında bir değişim tokus söz konusudur. Ancak durum bundan daha karmaşıktır. Üssün zımi tabanının 2 olması gerekmekz. Örneğin IBM S/390 mimarisi 16'lık bir taban kullanır [ANDE67b]. Biçim 7 bitlik bir üs ve 24 bitlik bir anlamlıdan oluşur.



**Şekil 10.20** Kayan Noktalı Sayıların Yoğunluğu

IBM baz-16 formatında,

$$0.11010001 * 2^{10100} = 0.11010001 * 16^{101}$$

ve üs 20 yerine 5'i temsil edecek şekilde saklanır.

Daha büyük bir üs kullanmanın avantajı, aynı sayıda üs biti için daha geniş bir aralık elde edilebilmesidir. Ancak, temsil edilebilecek farklı değerlerin sayısını artırmadığımızı unutmayın. Bu nedenle, sabit bir format için, daha büyük bir üs tabanı, daha az hassasiyet pahasına daha geniş bir aralık sağlar.

### **İkili Kayan Nokta Gösterimi için IEEE Standardı**

En önemli kayan nokta gösterimi 1985 yılında kabul edilen ve 2008 yılında revize edilen IEEE Standardı 754'te tanımlanmıştır. Bu standart, programların bir işlemcisinden diğerine taşınabilirliğini ve sofistike, sayısal yönelik programların geliştirilmesini teşvik etmek için geliştirilmiştir. Standart yaygın olarak benimsenmiştir ve neredeyse tüm çağdaş işlemcilerde ve aritmetik yardımcı işlemcilerde kullanılmaktadır. IEEE 754-2008 hem ikili hem de ondalık kayan nokta gösterimlerini kapsar. Bu bölümde sadece ikili gösterimler ele alınacaktır.

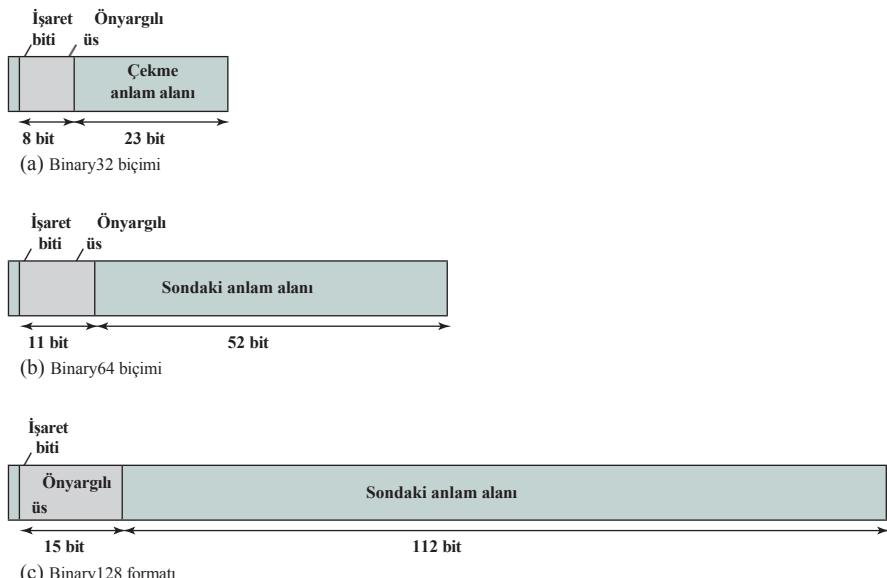
IEEE 754-2008 aşağıdaki farklı kayan nokta formatlarını tanımlar:

- **Aritmetik format:** Standart tarafından tanımlanan tüm zorunlu işlemler format tarafından desteklenir. Biçim, standartta tanımlanan işlemler için kayan noktalı işlenenleri veya sonuçları temsil etmek için kullanılabilir.
- **Temel format:** Bu format, kodlamaları standart tarafından belirtilen ve aritmetik için kullanılabilen üç ikili ve ikisi ondalık olmak üzere beş kayan nokta gösterimini kapsar. Temel biçimlerden en az biri herhangi bir uygun uygulamada gerçekleştirilir.
- **Değişim formatı:** Farklı platformlar arasında veri alışverişine izin veren ve depolama için kullanılabilen, tam olarak belirlenmiş, sabit uzunlukta ikili kodlama.

Üç temel ikili formatın bit uzunlukları 32, 64 ve 128 bit, üsleri ise sırasıyla 8, 11 ve 15 bittir (Şekil 10.21). Tablo 10.3 üç formatın özelliklerini özetlemektedir. İki temel ondalık biçiminin bit uzunlukları 64 ve 128 bittir. Tüm temel formatlar aynı zamanda aritmetik format tipleri (aritmetik işlemler için kullanılabilir) ve değişim format tipleridir (platformdan bağımsız).

Standartta başka formatlar da belirtilmiştir. *Binary16* formatı yalnızca bir değişim formatıdır ve daha yüksek ön kesinlik gereklilikinde değerlerin depolanması için tasarlanmıştır. *Binary{k}* formatı ve *decimal{k}* formatı, toplam uzunluğu  $k$  bit olan ve anlamlı ve üs için tanımlanmış uzunluklara sahip değişim formatlarıdır. Format 32 bitin katı olmalıdır; bu nedenle  $k$  için formatlar tanımlanmıştır: 160, 192, vb. Bu iki format ailesi aynı zamanda aritmetik formatlardır. Buna ek olarak, standart, üs (**genişletilmiş aralık**) ve anlamlılıkta (**genişletilmiş hassasiyet**) ek bitler sağlayarak desteklenen bir temel formatı genişleten **genişletilmiş hassasiyet** formatlarını tanımlar.

Tam format



**Şekil 10.21** IEEE 754 Formatları

uygulamaya bağlıdır, ancak standart üs ve anamlarının uzunluğuna belirli kısıtlamalar getirir. Bu biçimler aritmetik biçim türleridir ancak değişim biçimleri değildir. Genişletilmiş formatlar ara hesaplamalar için kullanılmalıdır. Genişletilmiş formatlar, daha yüksek hassasiyetleri ile

**Tablo 10.3** IEEE 754 Format Parametreleri

Parametre	Birim		
	Binary32	Binary64	Binary128
Depolama genişliği (bit)	32	64	128
Üs genişliği (bit)	8	11	15
Üs sapması	127	1023	16383
Maksimum üs	127	1023	16383
Minimum üs	-126	-1022	-16382
Yaklaşık normal sayı aralığı (10 tabanı)	$10^{-38} \text{--} 10^{+38}$	$10^{-308} \text{--} 10^{+308}$	$10^{-4932} \text{--} 10^{+4932}$
Sondaki anlamlı genişliği (bit)*	23	52	112
Üslerin sayısı	254	2046	32766
Kesir sayısı	$2^{23}$	$2^{52}$	$2^{112}$
Değer sayısı	$1.98 \times 2^{31}$	$1.99 \times 2^{63}$	$1.99 \times 2^{128}$
En küçük pozitif normal sayı	$2^{-126}$	$2^{-1022}$	$2^{-16362}$
En büyük pozitif normal sayı	$2^{128} - 2^{104}$	$2^{1024} - 2^{971}$	$2^{16384} - 2^{16271}$
En küçük normal altı büyüklük	$2^{-149}$	$2^{-1074}$	$2^{-16494}$

*Not:* \* Zumni bit ve işaret biti dahil değildir.

## 356 BÖLÜM 10 / BİLGİSAYAR ARİTMETİĞİ

Aşırı yuvarlama hatası nedeniyle kırılmış bir nihai sonuç olasılığı; daha geniş aralıklarıyla, nihai sonucu temel bir formatta temsil edilemeyecek bir hesaplamayı iptal eden bir ara taşıma olasılığını da azaltırlar. Genişletilmiş format için ek bir motivasyon, genellikle daha yüksek hassasiyetle ilişkili zaman cezasına maruz kalmadan daha büyük bir temel formatın bazı avantajlarını sağlamasıdır.

Son olarak, IEEE 754-2008 **genişletilebilir hassasiyet formatını** kullanıcı kontrolü altında tanımlanan hassasiyet ve aralığa sahip bir olarak tanımlar. Yine, bu formatlar ara hesaplamalar için kullanılabilir, ancak standart herhangi bir kısıtlama veya format veya uzunluk koymaz.

Tablo 10.4, tanımlanmış biçimler ve biçim türleri arasındaki ilişkiyi göstermektedir. IEEE biçimlerindeki tüm bit desenleri olağan şekilde yorumlanmaz; Bunun yerine, bazı bit kalıpları özel değerleri temsil etmek için kullanılır. Tablo 10.5 çeşitli bit kalıplarına atanmış değerleri gösterir. Tüm sıfırların (0 bit) ve tüm birlerin (1 bit) üs değerleri özel değerleri tanımlar. Aşağıdaki sayı sınıfları temsil edilir:

- 32-bit format için 1 ila 254, 64-bit format için 1 ila 2046 ve 1 ila 16382 aralığındaki üs değerleri için sıfır olmayan normal kayan noktalı sayılar temsil edilir. Üs değeri önyargılıdır, böylece üs değerleri aralığı 32 bit format için - 126 ile + 127 arasındadır ve bu şekilde devam eder. Normal bir sayı, ikili noktanın solunda bir 1 biti gerektirir; bu bit ima edilir ve bir 24 bit, 53 bit veya 113 bit anlamlılık verir. Bitlerden biri ima edildiğinden, ikili formattaki ilgili alan **sondaki işaret** alanı olarak adlandırılır.
- Sıfırın bir kesri ile birlikte sıfırın bir üssü, işaret bitine bağlı olarak pozitif veya negatif sıfırı temsil eder. Daha önce de belirtildiği gibi, 0 değerinin tam olarak temsil edilmesi yararlıdır.

**Tablo 10.4** IEEE Formatları

Biçim	Format Türü		
	Aritmetik Biçim	Temel Format	Değişim Formatı
binary16			X
binary32	X	X	X
binary64	X	X	X
binary128	X	X	X
binary{k} (k= n * 32 for n > 4)	X		X
decimal64	X	X	X
decimal128	X	X	X
ondalık{k} (k= n * 32 for n > 4)	X		X
genişletilmiş hassasiyet	X		
uzatılabilir hassasiyet	X		

**Tablo 10.5** IEEE 754 Kayan Noktalı Sayıların Yorumlanması**(a) binary32 biçimini**

	İşaret	Önyargılı Üs	Kesir	Değer
pozitif sıfır	0	0	0	0
negatif sıfır	1	0	0	-0
arti sonsuzluk	0	tüm 1'ler	0	
eksi sonsuzluk	1	tüm 1'ler	0	
sessiz NaN	0 veya 1	tüm 1'ler	$\neq 0$ ; ilk bit=1	qNaN
NaN sinyali	0 veya 1	tüm 1'ler	$\neq 0$ ; ilk bit=0	sNaN
pozitif normal sıfır olmayan	0	0 6 e 6 225	f	$2^{e-(127)}(1.f)$
negatif normal sıfır olmayan	1	0 6 e 6 225	f	$-2^{e-(127)}(1.f)$
pozitif subnormal	0	0	$f \neq 0$	$2^{e-(126)}(0.f)$
negatif subnormal	1	0	$f \neq 0$	$-2^{e-(126)}(0.f)$

**(b) binary64 biçimini**

	İşaret	Önyargılı Üs	Kesir	Değer
pozitif sıfır	0	0	0	0
negatif sıfır	1	0	0	-0
arti sonsuzluk	0	tüm 1'ler	0	
eksi sonsuzluk	1	tüm 1'ler	0	
sessiz NaN	0 veya 1	tüm 1'ler	$\neq 0$ ; ilk bit=1	qNaN
NaN sinyali	0 veya 1	tüm 1'ler	$\neq 0$ ; ilk bit=0	sNaN
pozitif normal sıfır olmayan	0	0 6 e 6 2047	f	$2^{e-1023}(1.f)$
negatif normal sıfır olmayan	1	0 6 e 6 2047	f	$-2^{e-1023}(1.f)$
pozitif subnormal	0	0	$f \neq 0$	$2^{e-1022}(0.f)$
negatif subnormal	1	0	$f \neq 0$	$-2^{e-1022}(0.f)$

**(c) binary128 formatını**

	İşaret	Önyargılı Üs	Kesir	Değer
pozitif sıfır	0	0	0	0
negatif sıfır	1	0	0	-0
arti sonsuzluk	0	tüm 1'ler	0	
eksi sonsuzluk	1	tüm 1'ler	0	
sessiz NaN	0 veya 1	tüm 1'ler	$\neq 0$ ; ilk bit=1	qNaN
NaN sinyali	0 veya 1	tüm 1'ler	$\neq 0$ ; ilk bit=0	sNaN
pozitif normal sıfır olmayan	0	tüm 1'ler	f	$2^{e-16383}(1.f)$
negatif normal sıfır olmayan	1	tüm 1'ler	f	$-2^{e-16383}(1.f)$
pozitif subnormal	0	0	$f \neq 0$	$2^{e-16383}(0.f)$
negatif subnormal	1	0	$f \neq 0$	$-2^{e-16383}(0.f)$

- Sıfırın bir kesri ile birlikte tüm birlerden oluşan bir üs, işaret bitine bağlı olarak pozitif veya negatif sonsuzluğu temsil eder. Sonsuzluğun bir temsilinin olması da yararlıdır. Bu, aşırı akışın bir hata durumu olarak ele alınıp alınmayacağına veya değerini taşıyıp taşımayacağına ve yürütülmekte olan programa devam edip etmeyeceğine karar vermeyi kullanıcıya bırakır.
- Sıfır olmayan bir kesirle birlikte sıfırın üssü normal altı bir sayıyı temsil eder. Bu durumda, ikili noktanın solundaki bit sıfırdır ve gerçek üs  $-126$  veya  $-1022$ 'dir. Sayı, işaret bitine bağlı pozitif veya negatiftir.
- Sıfır olmayan bir kesirle birlikte tüm birlerden oluşan bir üsse *Sayı Değil* anlamına gelen NaN değeri verilir ve çeşitli istisna durumlarını işaret etmek için kullanılır.

Normalin altındaki sayıların ve NaN'ların önemi Bölüm 10.5'te ele alınmaktadır.

## 10.5 FLOATING-POINT ARİTMETİĞİ

Tablo 10.6 kayan noktalı aritmetik için temel işlemleri özetlemektedir. Toplama ve çıkarma işlemleri için, her iki işlenenin de aynı üs değerine sahip olmasını sağlamak gereklidir. Bu, hizalama elde etmek için işlenenlerden birinde radix noktasının kaydırılmasını gerektirebilir. Çarpma ve bölme işlemleri daha basittir.

Bir kayan nokta işlemi bu koşullardan birini üretebilir:

- **Üs değeri taşması:** Pozitif bir üs, mümkün olan maksimum üs değerini aşar. Bazı sistemlerde bu durum  $\infty$  veya  $\text{NaN}$  olarak belirtilebilir.
- **Üs değerinin altında taşıma:** Negatif bir üs, mümkün olan minimum üs değerinden daha azdır (örneğin,  $-200$ ,  $-127$ 'den daha azdır). Bu, sayının temsil edilemeyecek kadar küçük olduğu ve 0 olarak raporlanabileceği anlamına gelir.

**Tablo 10.6** Kayan Noktalı Sayılar ve Aritmetik İşlemler

Kayan Noktalı Sayılar	Aritmetik İşlemler
$X = X_S \cdot B^{(X_{(E)})}$ $Y = Y_S \cdot B^{(Y_{(E)})}$	$X + Y = (X_S \cdot B^{(X_{(E)}+(-Y_{(E)})}) \cdot B^{Y_E}) + Y_S \cdot B^{Y_E}$ $X - Y = X \cdot B^{(X_{(E)}-(-Y_{(E)}))} \cdot B^{Y_E} - Y \cdot B^{(Y_{(E)})}$ $X * Y = (X_S \cdot Y_S) \cdot B^{X_E + Y_E}$ $\frac{X}{Y} = (\frac{X_S}{Y_S}) \cdot B^{X_E - Y_E}$

Örnekler:

$$X = 0.3 \cdot 10^2 = 30$$

$$Y = 0.2 \cdot 10^3 = 200$$

$$X + Y = (0.3 \cdot 10^{2-(3)}) + 0.2 \cdot 10^3 = 0.23 \cdot 10^3 = 230$$

$$X - Y = (0.3 \cdot 10^{2-(3)}) - 0.2 \cdot 10^3 = (-0.17) \cdot 10^3 = -170$$

$$X * Y = (0.3 \cdot 0.2) \cdot 10^{2+3} = 0.06 \cdot 10^5 = 6000$$

$$X / Y = (0.3 / 0.2) \cdot 10^{2-(3)} = 1.5 \cdot 10^{-1} = 0.15$$

- **Anlamlı alt akışı:** İşaretleri hizalama sürecinde, rakamlar işaretin sağ ucundan akabilir. Tartışacağımız gibi, bir çeşit yuvarlama gereklidir.
- **Anlamlı bit taşması:** Aynı işaretre sahip iki anlamlının toplanması, en anlamlı bitin dışı taşınamasına neden olabilir. Bu, açıklayacağımız gibi gerçekleme ile düzeltilebilir.

## Toplama ve Çıkarma

Kayan noktalı aritmetikte, toplama ve çıkarma işlemleri çoklu çarpma ve bölme işlemlerinden daha karmaşıktır. Bunun nedeni hizalama ihtiyacıdır. Toplama ve çıkarma için algoritmanın dört temel aşaması vardır:

1. Sıfırları kontrol edin.
2. Anlamlılıarı hizalayın.
3. Anlamları toplayın veya çıkarın.
4. Sonucu normalleştirin.

Tipik bir akış şeması Şekil 10.22'de gösterilmektedir. Adım adım anlatım, kayan noktalı toplama ve çıkarma için gereken ana işlevlere ışık tutar. Şekil 10.21'dekine benzer bir format varsayıyoruz. Toplama veya çıkarma işlemi için, iki işlenen ALU tarafından kullanılacak aktarılmalıdır. Kayan nokta biçimi örtük bir anlamlı bit içeriyorsa, bu bit işlem için açık hale getirilmelidir.

**Aşama 1. Sıfır kontrolü:** Toplama ve çıkarma işlemleri işaret değişikliği dışında aynı olduğundan, işlem bir çıkarma işlemiyse alt ucun işaretini değiştirerek başlar. Ardından, biri 0 ise, diğeri sonuç olarak bildirilir.

**2. Aşama. Anlamdaş hizalama:** Bir sonraki aşama, iki üssün eşit olması için sayıları manipüle etmektir.

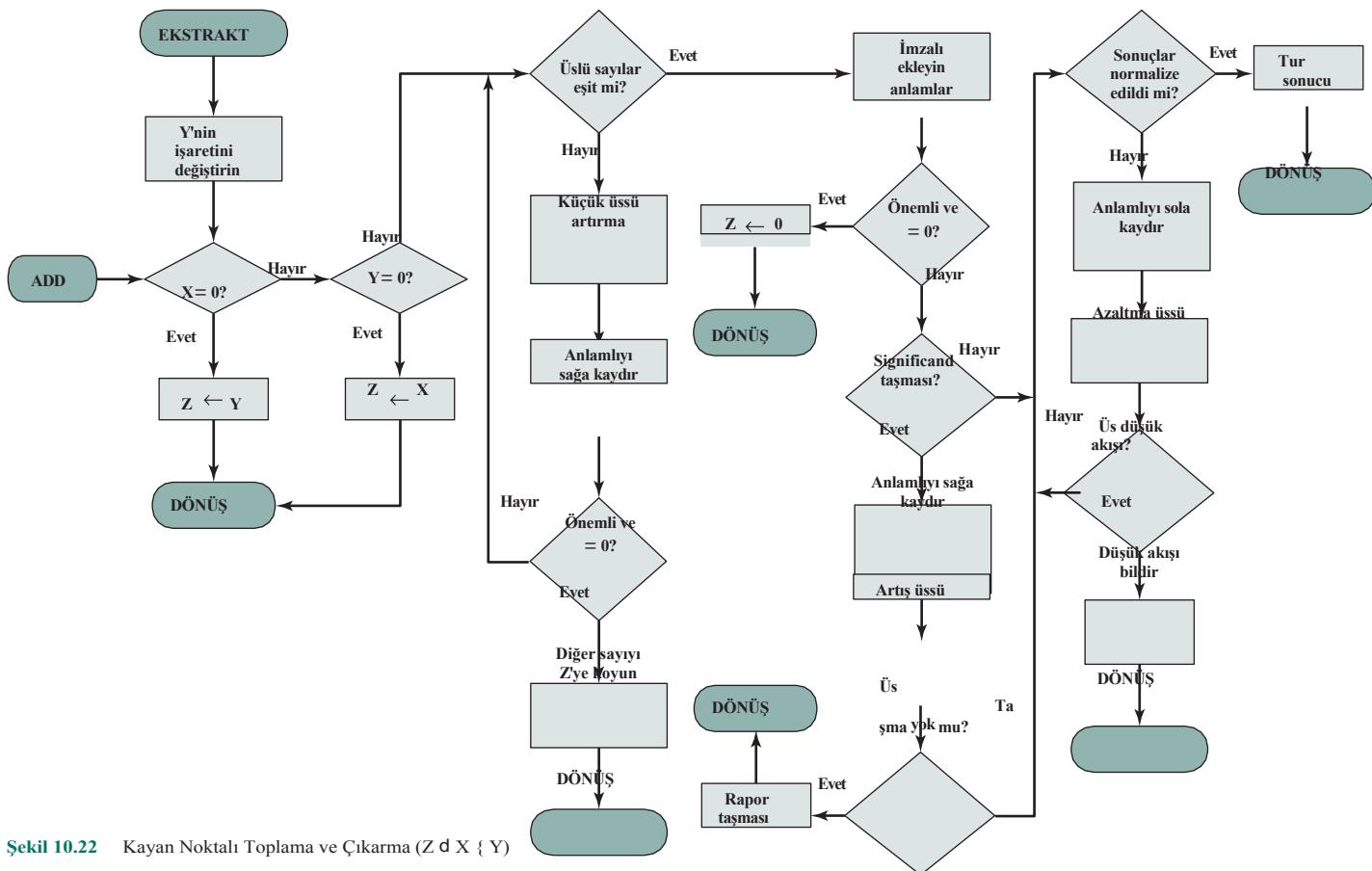
Üsleri hizalama ihtiyacını görmek için aşağıdaki ondalık toplama işlemini düşünün:

$$(123 * 10^0) + (456 * 10^{-2})$$

Açıkçası, sadece anlamlılıarı toplayamayız. Rakamlar önce eşdeğer konumlara getirilmelidir, yani ikinci sayının 4'ü birincinin 3'ü ile hizalanmalıdır. Bu koşullar altında, iki üs eşit olacaktır, bu da bu formdaki iki sayının toplanabileceği matematiksel koşuldur. Böylece,

$$(123 * 10^0) + (456 * 10^{-2}) = (123 * 10^{(0)}) + (4.56 * 10^{(0)}) = 127.56 * 10^{(0)}$$

Hizalama, ya küçük sayıyı sağa kaydırarak (üssünü artırarak) ya da büyük sayıyı sola kaydırarak gerçekleştirilebilir. Her iki işlem de basamak kaybına neden olabileceğinden, kaydırılan daha küçük sayıdır; bu nedenle kaybedilen basamakların önemi nispeten daha azdır. Hizalama



Şekil 10.22 Kayan Noktalı Toplama ve Çıkarma ( $Z = X - Y$ )

iki üs eşit olana kadar anımlının büyüklik kısmının tekrar tekrar 1 basamak sağa kaydırılması ve üssün artırılmasıyla elde edilir. (İma edilen taban 16 ise, 1 basamak kaydırmanın 4 bitlik bir kaydırma olduğunu unutmayın). Bu işlem anımlı sayı için 0 değeriyle sonuçlanırsa, diğer sayı olarak bildirilir. Böylece, iki sayının üsleri önemli ölçüde farklıysa, daha küçük olan sayı kaybedilir.

**3. Aşama. Toplama:** Daha sonra, iki anımlı işaretleri dikkate alınarak toplanır. İşaretler farklı olabileceğinden, sonuç 0 olabilir. 1 basamak taşıma olasılığı da vardır. Böyle bir durumda, sonucun anımlı değeri sağa kaydırılır ve üs artırılır. Sonuç olarak bir üs taşıması meydana gelebilir; bu durum rapor edilir ve işlem durdurulur.

**Aşama 4. Normalleştirme:** Son aşama sonucu normalleştirir. Normalleştirme, en anımlı basamak (bit veya 16 tabanlı üs için 4 bit) sıfır olmayana kadar anımlı basamakların sola kaydırılmasından oluşur. Her kaydırma üssün azalmasına neden olur ve bu nedenle üs düşük taşımasına neden olabilir. Son olarak, sonuç yuvarlanmalı ve ardından raporlanmalıdır. Yuvarlama ile ilgili tartışmayı çarpma ve bölme ile ilgili tartışmadan sonraya erteliyoruz.

## Çarpma ve Bölme İşlemleri

Aşağıdaki tartışmanın da gösterdiği gibi, kayan noktalı çarpma ve bölme işlemleri toplama ve çıkarmaya göre çok daha basit işlemlerdir.

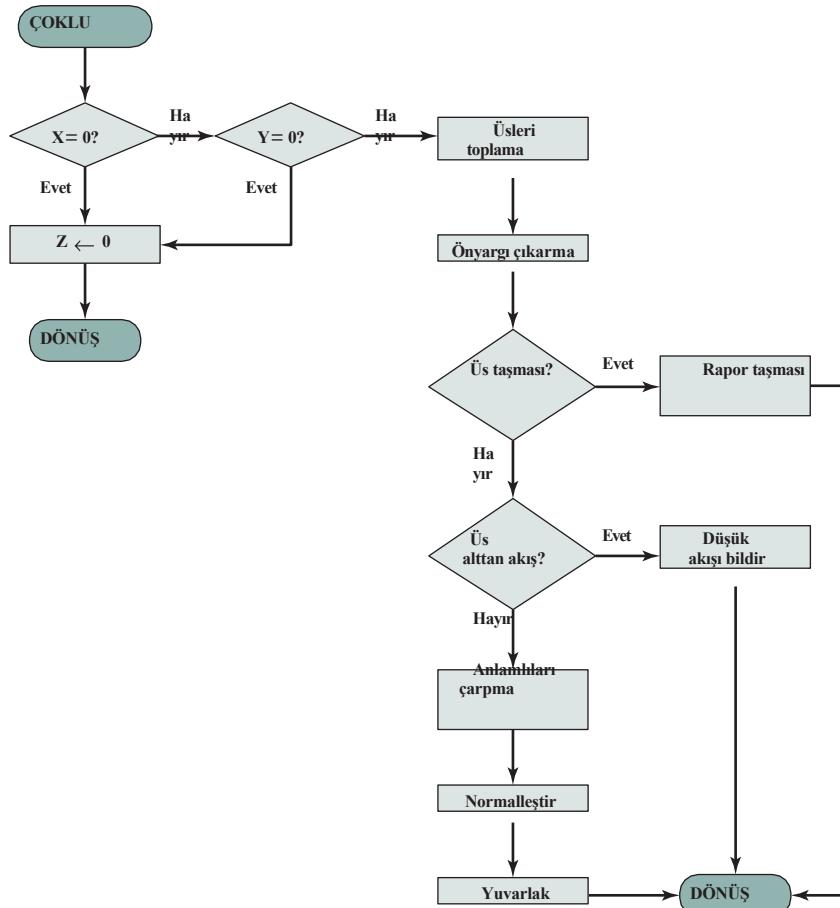
İlk olarak Şekil 10.23'te gösterilen çarpma işlemini ele alalım. İlk olarak, işlenenlerden biri 0 ise, sonuç olarak 0 bildirilir. Bir sonraki adım üsleri toplamaktır. Üsler önyargılı biçimde saklanıyorsa, üs toplamı önyargıyı iki katına çıkaracaktır. Bu nedenle, önyargı değeri toplamdan çıkarılmalıdır. Sonuç, üs taşıması ya da alt taşıması olabilir ve bu durum rapor edilerek algoritma sonlandırılır.

Çarpımın üssü uygun aralıktı ise, bir sonraki adım işaretlerini dikkate alarak anımlıları çarpmaktır. Çarpma işlemi tamsayılar için olduğu gibi gerçekleştirilir. Bu durumda, bir işaret-büyüklük gösterimi ile uğraşıyoruz, ancak ayrıntılar ikiye tümleyen gösterimi için olanlara benzer. Çarpım, çarpan ve çarpılanın iki katı uzunluğunda olacaktır. Ekstra bitler yuvarlama sırasında kaybolacaktır.

Çarpım hesaplandıktan sonra, toplama ve çıkarma işlemlerinde olduğu gibi sonuç normalleştirilir ve yuvarlanır. Normalleştirmenin üs düşük akışına neden olabileceği unutmayın.

Son olarak, Şekil 10.24'te gösterilen bölme işlemi için akış şemasını ele alalım. Yine, ilk adım 0'ı test etmektir. Bölgen 0 ise, uygulamaya bağlı olarak bir hata raporu verilir veya sonuç sonsuza ayarlanır. 0'lık bir bölgen 0 ile sonuçlanır. Daha sonra, bölgen üssü bölgen üssünden çıkarılır. Bu, geri eklenmesi gereken sapmayı ortadan kaldırır. Daha sonra üs düşüklüğü veya taşıması için testler yapılır.

Bir sonraki adım, anımlıları bölmektir. Bunu olağan nor-malizasyon ve yuvarlama takip eder.

Şekil 10.23 Kayan Noktalı Çarpma ( $Z = X \times Y$ )

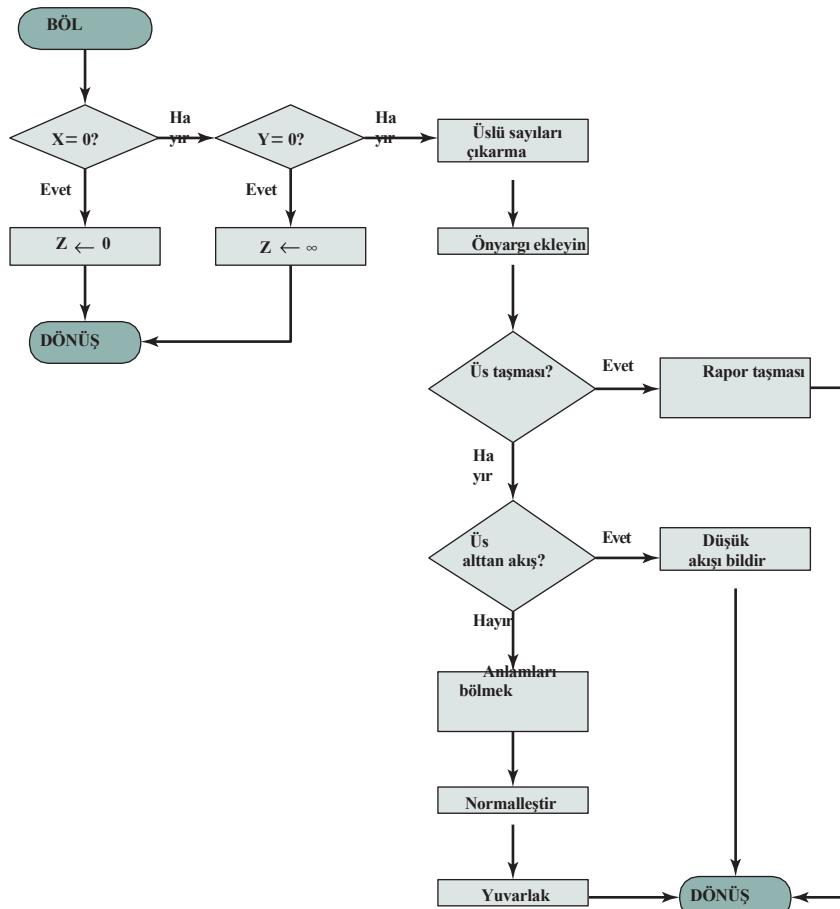
### Hassasiyetle İlgili Hususlar

**KORUYUCU BİTLER** Bir kayan nokta işleminden önce, her bir işlenenin üssünün ve anlamlılığının ALU yazmaçlarına yüklediğinden bahsetmiştim. Anlamlı bit durumunda, uzunluğu neredeyse her zaman anlamlı bitin uzunluğundan artı bir zımnı bitten daha büyktür. Yazmaç, koruma bitleri olarak adlandırılan ve anlamlının sağ ucunu 0'lara doldurmak için kullanılan ek bitler içerir.

Koruma bitlerinin kullanılmasının nedeni Şekil 10.25'te gösterilmiştir. IEEE formatındaki sayıları ele alalım; bu sayıların 24 bitlik bir anlamlı değeri vardır ve bu değerin solundaki 1 biti

ikili nokta. Değerleri birbirine çok yakın olan iki sayı  $x = 1.00\text{ g } 00 * 2^1$

ve  $y = 1.11\text{ g } 11 * 2^0$ . Küçük sayı büyük sayıdan çıkarılacaksa, üsleri hizalamak için 1 bit sağa kaydırılmalıdır. Bu Şekil 10.25a'da gösterilmiştir. Bu işlem sırasında y 1 bit önem kaybeder; sonuç  $2^{-22}$  olur. Aynı işlem şu şekilde tekrarlanır

Şekil 10.24 Kayan Noktalı Bölme ( $Z \leftarrow X/Y$ )

$x = 1.000\ldots00 \times 2^1$	$x = .100000 \times 16^1$
$-y = 0.111\ldots11 \times 2^1$	$-y = .0FFFFF \times 16^1$
$z = 0.000\ldots01 \times 2^1$ $= 1.000\ldots00 \times 2^{-22}$	$z = .000001 \times 16^1$ $= .100000 \times 16^{-4}$

(a) İkili örnek, koruma bitleri olmadan

(c) Koruma bitleri olmadan onaltılık örnek

$x = 1.000\ldots00 0000 \times 2^1$	$x = .100000 00 \times 16^1$
$-y = 0.111\ldots11 1000 \times 2^1$	$-y = .0FFFFF F0 \times 16^1$
$z = 0.000\ldots00 1000 \times 2^1$ $= 1.000\ldots00 0000 \times 2^{-23}$	$z = .000000 10 \times 16^1$ $= .100000 00 \times 16^{-5}$

(b) İkili örnek, koruma bitleri ile

(d) Koruma bitleriyle birlikte onaltılık örnek

Şekil 10.25 Koruma Bitlerinin Kullanımı

(b) bölümüne koruma bitlerinin eklenmesiyle elde edilir. Artık en az anlamlı bit hizalama nedeniyle kaybolmaz ve sonuç  $2^{-23}$  olur, bu da önceki yanıtta 2 katlık bir farktır. Radix 16 olduğunda, hassasiyet kaybı daha büyük olabilir. Şekil 10.25c ve (d)'nin gösterdiği gibi, fark 16 kat olabilir.

**YUVARLAMA** Sonucun hassasiyetini etkileyen bir diğer ayrıntı da yuvarlama politikasıdır. Anlamlılar üzerinde yapılan herhangi bir işlemin sonucu genellikle daha uzun bir kayıtta saklanır. Sonuç tekrar kayan nokta formatına sokulduğunda, tam sonuca yakın bir sonuç üretceek şekilde fazladan bitlerin eleinmesi gereklidir. Bu işleme **yuvarlama** denir.

Yuvarlama yapmak için bir dizi teknik araştırılmıştır. Aslında, IEEE standarı dört alternatif yaklaşım listelemektedir:

- **En yakına yuvarla:** Sonuç en yakın temsil edilebilir sayıya yuvarlanır.
- **Yuvarlak doğru**      H : Sonuç artı sonsuza doğru yuvarlanır.
- **Yuvarlak yanlış**      H : Sonuç negatif sonsuza doğru aşağı yuvarlanır.
- **0'a doğru yuvarla:** Sonuç sıfıra doğru yuvarlanır.

Bu politikaların her birini sırayla ele alalım. **En yakına** yuvarla, standartta listelenen varsayılan yuvarlama modudur ve aşağıdaki şekilde tanımlanır: Sonsuz kesin sonuca en yakın temsil edilebilir değer teslim edilecektir.

Saklanabilen 23 bitin ötesindeki ekstra bitler 10010 ise, ekstra bitler son temsil edilebilir bit konumunun yarısından fazlasına denk gelir. Bu durumda doğru cevap, son temsil edilebilir bite ikili 1 eklemek ve bir sonraki temsil edilebilir sayıya yuvarlamaktır. Şimdi fazladan bitlerin 01111 olduğunu düşünün. Bu durumda, fazladan bitler son temsil edilebilir bit konumunun yarısından daha azına denk gelir. Doğru cevap basitçe fazladan bitleri atmaktadır (truncate), bu da bir sonraki temsil edilebilir sayıya yuvarlama etkisine sahiptir.

Standart ayrıca 10000.... formundaki ekstra bitlerin özel durumunu da ele almaktadır. Burada sonuç iki olası temsil edilebilir değerin tam ortasındadır. Burada olası bir teknik, en basit işlem olacağının her zaman kesmek olacaktır. Ancak bu basit yaklaşımın zorluğu, bir dizi hesaplama küçük ama kümülatif bir yanılık katmasıdır. Gerekli olan tarafsız bir yuvarlama yöntemidir. Olası bir yaklaşım, rastgele bir sayı temelinde yukarı veya aşağı yuvarlama yapmaktadır, böylece ortalama olarak sonuç tarafsız olacaktır. Bu yaklaşımı karşı olan argüman, tahmin edilebilir, deterministik sonuçlar üretmemesidir. IEEE standarı tarafından benimsenen yaklaşım, sonucu çift olmaya zorlamaktır: Bir hesaplamanın sonucu iki temsil edilebilir sayının tam ortasındaysa, temsil edilebilir son bit o anda 1 ise değer yukarı yuvarlanır ve o anda 0 ise yukarı yuvarlanmaz.

Sonraki iki seçenek, **arti ve eksı sonsuza yuvarlama**, aralık aritmetiği olarak bilinen bir teknigin uygulanmasında kullanışlıdır. Aralık aritmetiği, her sonuç için iki değer tıreterek kayan noktalı hesaplamalardaki hataları izlemek ve kontrol etmek için etkili bir yöntem sağlar. Bu iki değer, gerçek sonucu içeren bir aralığın alt ve üst uç noktalarına karşılık gelir. Aralığın genişliği, yani üst ve alt uç noktalar arasındaki fark, sonucun doğruluğunu gösterir. Bir aralığın uç noktaları temsili değilse, aralık uç noktaları sırasıyla aşağı ve yukarı yuvarlanır. Aralığın genişliği uygulamaya göre değişebilse de, birçok algoritma dar aralıklar üretmek üzere tasarlanmıştır. Üst ve alt sınırlar arasındaki aralık yeterince darsa, yeterince doğru bir sonuç elde edilmiş demektir. Değilse, en azından bunu biliyoruz ve ek analiz yapabiliriz.

Standartta belirtilen son teknik **sıfıra doğru yuvarlamadır**. Bu aslında basit bir kesme işlemidir: Fazladan bitler göz ardı edilir. Bu kesinlikle en basit tekniktir. Ancak sonuç, kesilen değerin büyüklüğünün her zaman daha kesin olan orijinal değerden küçük veya ona eşit olması ve işlemede sıfıra doğru tutarlı bir sapmanın ortaya çıkmasıdır. Bu ciddi bir sapmadır çünkü sıfır olmayan ekstra bitlerin bulunduğu her işlemi etkiler.

### **İkili Kayan Nokta Aritmetiği için IEEE Standardı**

IEEE 754, kayan noktalı aritmetığın donanım platformundan bağımsız olarak tek tip, öngörülebilir sonuçlar üretmesi için özel uygulamalar ve prosedürler belirlemek üzere bir formatın basit tanımının ötesine geçer. Bunun bir yönü, yani yuvarlama konusu daha önce tartışılmıştı. Bu alt bölümde diğer üç konu ele alınacaktır: sonsuzluk, NaN'lar ve normal altı sayılar.

Sonsuzluk aritmetiği, gerçek aritmetığın sınırlayıcı durumu olarak ele alınır ve sonsuzluk değerlerine aşağıdaki yorum verilir:

$$\infty - 6 \text{ (her sonlu sayı)} 6 +\infty$$

Daha sonra ele alınacak özel durumlar haricinde, sonsuzu içeren herhangi bir aritmetik işlem bariz bir sonuç verir.

Örneğin:

$5 + (+\infty) = +\infty$	$5 , ( +\infty ) = + 0$
$5 - (+\infty) = \infty$	$( +\infty ) + ( +\infty ) = +\infty$
$5 + (-\infty) = \infty$	$( - ) \infty + ( - ) \infty = \infty$
$5 - ( - ) \infty = +\infty$	$( -\infty ) + \infty = \infty$
$5 * ( +\infty ) = +\infty$	$( +\infty ) - ( - ) \infty = +\infty$

**SESSİZ VE SINYALLİ NAN'LAR** NaN, kayan nokta biçiminde kodlanmış sembolik bir varlıktır ve iki türü vardır: sinyalli ve sessiz. Sinyal veren bir NaN, bir işlenen olarak göründüğünde geçersiz bir işlem istisnasına işaret eder. Sinyalleme

**Tablo 10.7** Sessiz NaN Üreten İşlemler

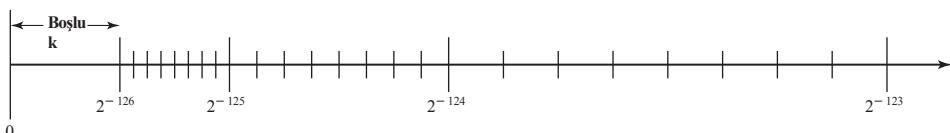
Operasyon	Quiet NaN Yapımı
Herhangi bir	Sinyal veren NaN üzerinde herhangi bir işlem
Ekleme veya çıkarma	Sonsuzlukların büyüklük çıkarımı: $(+)\infty + (-)\infty = (-)\infty$ $(+)\infty + (+\infty) = (+\infty)$ $(+\infty) - (+\infty) = (-\infty)$ $(-\infty) - (-\infty) = (-\infty)$
Çarpma	$0 * \infty$
Bölüm	$\frac{0}{0} \text{ veya } \frac{\infty}{\infty}$
Kalan	$x \bmod 0$ veya $\bmod y$
Karekök	$\sqrt{-x}$ , burada $x \neq 0$

NaN'ler, standardın konusu olmayan başlatılmamış değişkenler ve aritmetik benzeri geliştirmeler için değerler sağlar. Sessiz bir NaN, neredeyse tüm aritmetik işlemlerde bir istisna sinyali vermeden yayılır. Tablo 10.7 sessiz bir NaN üretenecek işlemleri gösterir.

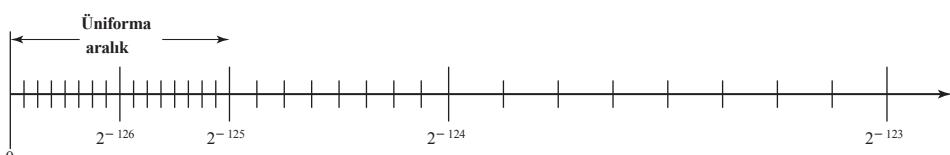
Her iki NaN türünün de aynı genel biçimde sahip olduğuna dikkat edin (Tablo 10.4): tüm birlерden oluşan bir üs ve sıfır olmayan bir kesir. Sıfır olmayan kesrin gerçek bit düzene uygulamaya bağlıdır; kesir değerleri sessiz NaN'ları sinyal veren NaN'lardan ayırt etmek ve belirli istisna koşullarını belirtmek için kullanılabilir.

**ALTNORMAL SAYILAR** Altnormal sayılar, üs düşüklüğü durumlarını ele almak için IEEE 754'e dahil edilmiştir. Sonucun üssü çok küçük olduğunda (çok büyük bir büyülüğe sahip negatif bir üs), kesir sağa kaydırılırak ve üs temsil edilebilir bir aralıkta olana kadar her kaydırma için üs artırılarak sonuç normal altı hale getirilir.

Şekil 10.26 normal altı sayıların dahil edilmesinin etkisini göstermektedir. Temsil edilebilir sayılar  $[2^n, 2^{n+1}]$  biçiminde aralıklar halinde gruplandırılabilir. İçinde



(a) Alt normal sayılar olmadan 32 bit format



(b) Normal altı sayılarla 32 bit format

**Şekil 10.26** IEEE 754 Normal Altı Sayıların Etkisi

tür aralıkların her birinde, kesir değişirken sayının üs kısmı sabit kalır ve aralık içinde temsil edilebilir sayıların tekduze bir aralığını oluşturur. Sıfıra yaklaşıkça, birbirini izleyen her aralık bir önceki aralığın yarısı genişliğinde olur, ancak aynı sayıda temsil edilebilir sayı içerir. Dolayısıyla, sıfıra yaklaşıkça temsil edilebilir sayıların yoğunluğu artar. Ancak, yalnızca normal sayılar kullanılıyorsa, en küçük normal sayı ile 0 arasında bir boşluk vardır. 32 bit IEEE 754 formatı durumunda, her aralikta  $2^{23}$  temsil edilebilir sayı vardır ve temsil edilebilir en küçük pozitif sayı  $2^{-126}$ dir. Alt normal sayıların eklenmesiyle, 0 ile  $2^{(-126)}$ arasına eşit olarak  $2^{23} - 1$  sayı daha eklenir.

Normal altı sayıların kullanımı *kademeli düşük akış* olarak adlandırılır [COON81]. Normal altı sayılar olmadan, temsil edilebilen en küçük sıfır olmayan sayı ile sıfır arasındaki boşluk, temsil edilebilen en küçük sıfır olmayan sayı ile bir sonraki daha büyük sayı arasındaki boşluktan çok daha genişir. Kademeli düşük akış bu boşluğu doldurur ve üs düşük akışının etkisini normal sayılar arasındaki yuvarlama ile karşılaştırılabilir bir seviyeye indirir.

## 10.6 ÖNERİLER, SORULAR VE SORUNLAR

### Anahtar Terimler

aritmetik ve mantık birimi (ALU)	minuend multiplicand çarpan negatif taşıma negatif düşük taşıma normal sayı birler tamamlayıcı temsili taşma kısmı çarpımı pozitif taşıma pozitif alt taşıma ürün bölüm	radix nokta aralığı genişletme kalan yuvarlama i işaret biti i işaret-magnitud temsili significand significand overflow significand underflow subnormal number subtrahend ikiye tümleyen gösterimi
----------------------------------	---	--

### İnceleme Soruları

- 10.1** Aşağıdaki gösterimleri kısaca açıklayınız: işaret büyüklüğü, ikiye tümleyen, önyargılı.
- 10.2** Aşağıdaki gösterimlerde bir sayının negatif olup olmadığını nasıl belirleneceğini açıklayın: işaret büyüklüğü, ikiye tümleyen, önyargılı.
- 10.3** İkiye tümleyen sayılar için işaret uzatma kuralı nedir?
- 10.4** Bir tamsayıının ikiye tümleyen gösteriminde olumsuzunu nasıl oluşturabilirsiniz?
- 10.5** Genel anlamda,  $n$  bitlik bir tamsayı üzerinde ikiye tümleyen işlemi ne zaman aynı tamsayıyı üretir?

- 10.6** Bir sayının ikiye tümleyen gösterimi ile bir sayının ikiye tümleyeni arasındaki fark nedir?
- 10.7** Toplama işlemi için iki ikiye tümleyen sayıı işaretsiz tamsayılar olarak ele alırsak, sonuç iki ikiye tümleyen sayı olarak yorumlanırsa doğru olur. Bu çarpma işlemi için doğru değildir. Neden?
- 10.8** Kayan nokta gösteriminde bir sayının dört temel ögesi nedir?
- 10.9** Kayan noktalı bir sayının üs kısmı için yanlı gösterim kullanmanın faydası nedir?
- 10.10** Pozitif taşıma, üs taşıması ve anlamlı arasındaki farklar nelerdir?
- 10.11** Kayan noktalı toplama ve çıkarma işlemlerinin temel unsurları nelerdir?
- 10.12** Koruma bitlerinin kullanımı için bir neden belirtin.
- 10.13** Bir kayan nokta işleminin sonucunu yuvarlamak için dört alternatif yöntem listeleyin.

## Problemler

- 10.1** Aşağıdaki ondalık sayıları 16 bit kullanarak hem ikili işaret/büyüklük hem de ikiye tümleyen olarak gösteriniz: + 512; - 29.
- 10.2** Aşağıdaki ikiye tümleyen değerleri ondalık olarak temsil edin: 1101011; 0101101.
- 10.3** İkili tamsayıların bazeen karşılaşılan **bir** başka gösterimi de **birler tümleyenidir**. Pozitif tamsayılar işaret büyülüğu ile aynı şekilde temsil edilir. Negatif bir tamsayı, karşılık gelen pozitif sayının her bir bitinin Boole alınarak temsil edilir.
- Denklem (10.1) ve (10.2)'ye benzer şekilde, bitlerin ağırlıklı toplamını kullanarak birler tümleyen sayılarının bir tanımını verin.
  - Birler tümleyeninde temsil edilebilen sayı aralığı nedir?
  - Birler tümleyen aritmetığında toplama yapmak için bir algoritma tanımlayın. *Not:* Birler tümleyen aritmetiği 1960'larda donanımdan kayboldum, ancak İnternet Protokolü (IP) ve İletim Kontrol Protokolü (TCP) içinde toplama hesaplamaları hala varlığını sürdürmektedir.
- 10.4** Tablo 10.1'e işaret büyülüğu ve birler tümleyen için sütunlar ekleyin.
- 10.5** İkili bir sözcük üzerinde aşağıdaki işlemi düşünün. En az anlamlı bit ile başlayın. İlk bite ulaşana kadar 0 olan tüm bitleri kopyalayın ve bu biti de kopyalayın. Sonra bundan sonrası her bitin tümleyenini alın. Sonuç ne olur?
- 10.6** Bölüm 10.3'te, ikiye tümleyen işlemi aşağıdaki gibi tanımlanmıştır.  $X$ 'in ikiye tümleyenini bulmak için,  $X$ 'in her bitinin Boole tümleyenini alın ardından 1 ekleyin.
- Aşağıdakının eşdeğer bir tanım olduğunu gösteriniz.  $n$ -bitlik bir  $X$  tamsayı için,  $X$ 'in ikili tümleyeni,  $X$ 'i işaretsiz bir tamsayı olarak ele alıp  $(2^n - X)$ 'i hesaplayarak oluşturulur.
  - Çıkarma işlemini gerçekleştirmek için saat yönünde hareketin nasıl kullanıldığını göstererek, Şekil 10.5'in (a) bölümündeki iddiayı grafiksel olarak desteklemek için kullanılabilceğini gösteriniz.
- 10.7**  $r$  tabanındaki  $n$  basamaklı bir  $N$  sayısının  $r'$ ye tümleyeni,  $N \neq 0$  için  $r' = N$  ve  $N = 0$  için 0 olarak tanımlanır. 13.250 ondalık sayısının onlar tümleyenini bulunuz.
- 10.8** Onluk tümleyen aritmetiğini kullanarak (72, 530 - 13, 250) değerini hesaplayın. İki tümleyen aritmetiği için olanlara benzer kurallar varsayız.
- 10.9** İki n-bit sayının ikiye tümleyen toplamını düşünün:

$$zn - lzn - 2 \leftarrow z0 = xn - lxn - 2 \leftarrow x0^+ yn - lyn - 2 \leftarrow y0$$

Bitsel toplamanın  $x_{(i)}, y_{(i)}$  ve  $c_{(i)(-)(1)}$ in oluşturulan bir  $c(i)$  taşıma biti ile gerçekleştirildiğini varsayıñ.  $n, n=1$  olduğunda taşmayı gösteren ikili bir değişken olsun. Tablodaki değerleri doldurun.

	$xn - 1$	0	0	0	0	1	1	1	1
Giriş	$yn - 1$	0	0	1	1	0	0	1	1
	$cn - 2$	0	1	0	1	0	1	0	1
	$zn - 1$								
Çıktı	n								

- 10.10** Sayıların 8 bitlik ikiye tümleyen gösteriminde temsil edildiğini varsayıñ. Aşağıdakilerin hesaplanması :  
**a.**  $6+13$       **b.**  $-6+13$       **c.**  $6-13$       **d.**  $-6-13$
- 10.11** İkiye tümleyen aritmetiğini kullanarak aşağıdaki farkları bulun:  
**a.**  $\begin{array}{r} 111000 \\ - 110011 \end{array}$       **b.**  $\begin{array}{r} 11001100 \\ - 101110 \end{array}$       **c.**  $\begin{array}{r} 111100001111 \\ - 110011110011 \end{array}$       **d.**  $\begin{array}{r} 11000011 \\ - 11101000 \end{array}$
- 10.12** Aşağıdakı, ikiye tümleyen aritmetiğinde taşmanın geçerli bir alternatif tanımı mıdır?  
En soldaki sütuna giren ve çıkan carry bitlerinin exclusive-OR değeri 1 ise, bir taşıma durumu vardır. Aksi takdirde, .
- 10.13** Şekil 10.9 ve 10.12'yi karşılaştırın. İkincisinde C biti neden kullanılmamıştır?
- 10.14** İkili tümleyen gösteriminde  $x=0101$  ve  $y=1010$  verildiğinde (yani,  $x=5$ ,  $y=-6$ ), Booth algoritması ile  $p=x^*y$  çarpımını hesaplayın.
- 10.15** Her sayının 6 bit kullanılarak temsil edildiği 23 (çarpan) ile 29'u (çarpan) çarpmak için Booth algoritmasını kullanın.
- 10.16**  $B$  tabanında  $n$  basamaklı iki sayının çarpımının  $2n$  basamaktan daha fazla olmayan bir çarpım verdiğiini kanıtlayın.
- 10.17** Şekil 10.5'te gösterilen bölme işleminin hesaplanmasında yer alan adımları göstererek Şekil 10.16'daki işaretsiz ikili bölme algoritmasının geçerliliğini doğrulayın. Şekil 10.17'dekine benzer bir ön gösterim kullanın.
- 10.18** Bölüm 10.3'te açıklanan ikiye tümleyen tamsayı bölme algoritması geri yüklem yöntemi olarak bilinir çünkü başarısız çıkarma işleminden sonra A yazmacındaki değerin geri yüklenmesi gereklidir. Geri yüklemesiz olarak bilinen biraz daha karmaşık bir yaklaşım, gereksiz çıkarma ve toplama işlemlerini önerir. Bu ikinci yaklaşım için bir algoritma öneriniz.
- 10.19** Bilgisayar tamsayı aritmetiği altında, iki J ve K tamsayısının J/K bölümü normal bölümden küçük veya ona eşittir. Doğru mu yanlış mı?
- 10.20** 12 bitlik sözcükler kullanarak ikili tümleyen gösteriminde  $-145^{\prime}i$  13'e bölün. Bölüm 10.3'te açıklanan algoritmayı kullanın.
- 10.21** a. Ondalık basamaklar kullanan sabit noktalı bir gösterim düşünün, burada ima edilen radix noktası herhangi bir konumda olabilir (en az anlamlı basamağın sağında, en anlamlı basamağın sağında, vb.) Hem Planck sabitinin  $(6,63 * 10^{-27})$  hem de Avogadro sayısının  $(6,02 * 10^{23})$  yaklaşımlarını temsil etmek için kaç ondalık basamak gereklidir? İma edilen radix noktası her iki sayı için de aynı konumda olmalıdır.  
b. Şimdi, üssü 50 önyargılı bir temsilde saklanan ondalık kayan nokta biçimini düşünün. Normalleştirilmiş bir gösterim varsayılmaktadır. Bu sabitleri bu kayan nokta biçiminde temsil etmek için kaç ondalık basamak gereklidir?
- 10.22** Üs  $e$ 'nin  $0 \dots e \dots X$  aralığında,  $q^{\prime}luk$  bir sapma ile kısıtladığını, tabanın  $b$  olduğunu ve anlamlının  $p$  basamak uzunluğunda olduğunu varsayıñ.
- a. Yazılabilecek en büyük ve en küçük pozitif değerler nelerdir?  
b. Normalleştirilmiş kayan noktalı sayılar olarak yazılabilecek en büyük ve en küçük pozitif değerler nelerdir?

- 10.23** Aşağıdaki sayıları IEEE 32-bit karan nokta biçiminde ifade edin:
- a. -5      b. -6      c. -1,5      d. 384      e. 1/16      f. -1/32
- 10.24** Aşağıdaki sayılar IEEE 32-bit karan nokta formatını kullanmaktadır. Eşdeğer ondalık değer nedir?
- a. 1 1000011 11000000000000000000000000  
 b. 0 0111110 10100000000000000000000000  
 c. 0 1000000 00000000000000000000000000
- 10.25** Üs için 3 bit ve anlamlı için 3 bit indirgenmiş 7 bit IEEE karan nokta formatını göz önünde bulundurun. Tüm 127 değeri listeleyin.
- 10.26** Aşağıdaki sayıları IBM'in 32 bit karan nokta formatında ifade edin; bu formatta 7 bitlik üs, 16'lık zimni taban ve 64'lük (40 onaltılık) bir üs sapması kullanılır. Normalleştirilmiş bir karan noktalı sayı, en soldaki onaltılık basamağının sıfır olmadığını gerektirir; ima edilen radix noktası bu basamağın solundadır.
- |               |                |                            |                           |
|---------------|----------------|----------------------------|---------------------------|
| <b>a.</b> 1.0 | <b>c.</b> 1/64 | <b>e.</b> - 15.0           | <b>g.</b> $7.2 * 10^{75}$ |
| <b>b.</b> 0.5 | <b>d.</b> 0.0  | <b>f.</b> $5.4 * 10^{-79}$ | <b>h.</b> 65,535          |
- 10.27** 5BCA0000 IBM formatında onaltılık olarak ifade edilen bir karan noktalı sayı olsun. Sayının ondalık değeri nedir?
- 10.28** için önyargı değeri ne olurdu?
- a. 6 bitlik bir alanda 2 taban üssü ( $B=2$ )?  
 b. 7 bitlik bir alanda 8 taban üssü ( $B=8$ )?
- 10.29** Şekil 10.21b'karan nokta formatı için Şekil 10.19b'dekine benzer bir sayı doğrusu çizin.
- 10.30** İki taraflı üs için 8 bit ve anlamlı için 23 bit içeren bir karan nokta biçimi düşünün. Bu formatta aşağıdaki sayılar için bit modelini gösterin:
- a. -720      b. 0.645
- 10.31** Metinde 32 bitlik bir formatın en fazla  $2^{32}$  farklı sayıyı temsil edebileceğiinden bahsedilmektedir. IEEE 32-bit formatında kaç farklı sayı temsil edilebilir? Açıklayınız.
- 10.32** Bilgisayarda kullanılan herhangi bir karan noktalı gösterim yalnızca belirli gerçek sayıları tam olarak temsil edebilir; diğerlerinin hepsine yaklaşılması gereklidir. Eğer  $A'$ ,  $A$  gerçek değerine yaklaşan saklanan değer ise, o zaman bağıl hata,  $r$ , şu şekilde ifade edilir
- $$r = \frac{A - A'}{A}$$
- Ondalık büyülüklüğü + 0.4 aşağıdaki karan nokta biçiminde gösterin: taban= 2; üs: iki değerli, 4 bit; anlamlı, 7 bit. Bağıl hata nedir?
- 10.33**  $A=1.427$  ise,  $A$  1.42'ye kesilirse ve 1.43'e yuvarlanırsa bağıl hatayı bulun.
- 10.34** İnsanlar karan nokta aritmetiğindeki yanlışlıklarından bahsederken, genellikle hataları neredeyse eşit büyülüklüklerin çıkarılması sırasında meydana gelen iptallere bağlarlar. Ancak  $X$  ve  $Y$  yaklaşık olarak eşit olduğunda,  $X - Y$  farklı hiçbir hata olmadan tam olarak elde edilir. Bu insanlar gerçekten ne demek istiyor?
- 10.35**  $A$  ve  $B$  sayısal değerleri bilgisayarda  $A'$  ve  $B'$  yaklaşımları olarak saklanır. Diğer kesme veya yuvarlama hatalarını ihmal ederek, çarpımın görelî hatasının yaklaşık olarak faktörlerdeki görelî hataların toplamını olduğunu gösterin.
- 10.36** Bilgisayar hesaplamlarındaki en ciddi hatalardan biri, neredeyse eşit iki sayı çıkarıldığında ortaya çıkar.  $A=0.22288$  ve  $B=0.22211$  olsun. Bilgisayar tüm değerleri dört ondalık basamağa böler. Böylece  $A'=0.2228$  ve  $B'=0.2221$  olur.
- a.  $A'$  ve  $B'$  için görelî hatalar nelerdir?  
 b.  $C'=A'-B'$  için bağıl hata nedir?

- 10.37** Normalleştirmenin ve kademeli düşük akışın etkilerini anlamak için, anlamlı sayı için 6 ondalık basamak sağlayan ve normalleştirilmiş en küçük sayının  $10^{-99}$  olduğu bir ondalık sistem düşünün. Normalleştirilmiş bir sayı, ondalık noktanın solunda sıfır olmayan bir ondalık basamağa sahiptir. Aşağıdaki hesaplamaları yapın ve sonuçları normalleştirin. Sonuçlar hakkında yorum yapın.
- a.  $(2.50000 * 10^{-60}) * (3.50000 * 10^{(-43)})$   
b.  $(2.50000 * 10^{-60}) * (3.50000 * 10^{(-60)})$   
c.  $(5.67834 * 10^{-97}) - (5.67812 * 10^{(-97)})$
- 10.38** Aşağıdaki kayan noktalı toplama işlemlerinin nasıl yapıldığını gösterin (burada anlamlılar 4 ondalık basamağa kesilir). Sonuçları normalleştirilmiş biçimde gösterin.
- a.  $5.566 * 10^{2+} 7.777 * 10^2$   
b.  $3.344 * 10^{1+} 8.877 * 10^{-2}$
- 10.39** Aşağıdaki kayan noktalı çıkarma işlemlerinin nasıl yapıldığını gösterin (burada işaretler 4 ondalık basamağa kesilmiştir). Sonuçları normalleştirilmiş biçimde gösterin.
- a.  $7.744 * 10^{-3} - 6.666 * 10^{(-3)}$   
b.  $8.844 * 10^{-3} - 2.233 * 10^{(-1)}$
- 10.40** Aşağıdaki kayan nokta hesaplamalarının nasıl yapıldığını gösterin (burada anlamlılar 4 ondalık basamağa kesilir). Sonuçları normalleştirilmiş biçimde gösterin.
- a.  $(2.255 * 10^1) * (1.234 * 10^0)$   
b.  $(8.833 * 10^2), (5.555 * 10^4)$

# 11

## BÖLÜM

### DIjITAL L0gIC

#### 11.1 Boole Cebiri

#### 11.2 Gates

#### 11.3 Kombinasyonel Devreler

Boole Fonksiyonlarının Uygulanması Çoklayıcılar  
Kod Çözüçüler  
Salt Okunur Bellek  
Toplayıcıları

#### 11.4 Sırah Devreler

Flip-Flopolar  
Kaydediciler  
Sayaçlar

#### 11.5 Programlanabilir Mantık Aygıtları

Programlanabilir Mantık Dizisi Alan  
Programlanabilir Kapı Dizisi

#### 11.6 Anahtar Terimler ve Sorunlar

**ÖĞRENİM HEDEFLERİ**

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- **Boole cebirinin** temel işlemlerini anlamak.
- Farklı **flip-flop** türleri arasında ayırmayı yapabilme.
- Bir Boole ifadesini basitleştirmek için bir Karnaugh haritası kullanın.
- **Programlanabilir mantık cihazlarına** genel bir bakış sunmak.

Dijital bilgisayarın çalışması ikili verilerin depolanması ve işlenmesine dayanır. Bu kitap boyunca, iki kararlı durumdan birinde bulunabilecek depolama elemanlarının ve çeşitli bilgisayar işlevlerini uygulamak için kontrol sinyallerinin kontrolü altında ikili veriler üzerinde çalışabilecek devrelerin varlığını varsayıdık. Bu bölümde, bu depolama elemanlarının ve devrelerin sayısal mantıkta, özellikle de birleşik ve sıralı devrelerle nasıl uygulanabileceğini öneriyoruz. Bölüm, dijital mantığın matematiksel temeli olan Boole cebirinin kısa bir incelemesi ile başlamaktadır. Daha sonra, kapı kavramı tanıtmaktadır. Son olarak, **kapılardan** inşa edilen birleşik ve sıralı devreler açıklanmaktadır.

## 11.1 BOOLE CEBRI

Dijital bilgisayarlardaki ve diğer dijital sistemlerdeki dijital devreler, **Boole cebiri** olarak bilinen matematiksel bir disiplin kullanılarak tasarlanır ve davranışları analiz edilir. Bu isim, bu cebirin temel ilkelerini 1854 yılında *Mantık ve Olasılıkların Matematiksel Teorilerini Kurmak İçin Düşünce Yasalarının Araştırılması* adlı çalışmasında ortaya koyan İngiliz matematikçi George Boole'un onuruna verilmiştir. 1938 yılında M.I.T. Elektrik Mühendisliği Bölümü'nde araştırma görevlisi olan Claude Shannon, Boole cebirinin röle-anahtarlama devresi tasarımindaki sorunları çözmek için kullanılabileceğini öne sürdü [SHAN38].<sup>1</sup> Shannon'ın teknikleri daha sonra elektronik dijital devrelerin analizi ve tasarımda kullanıldı. Boole cebirinin iki alanda kullanılmış bir araç olduğu ortaya çıkmıştır:

- **Analiz:** Dijital devrenin işlevini tanımlamanın ekonomik bir .
- **Tasarım:** İstenen bir fonksiyon verildiğinde, Boole cebiri bu fonksiyonun basitleştirilmiş bir uygulamasını geliştirmek için uygulanabilir.

Her cebirde olduğu gibi Boole cebirinde de değişkenler ve işlemler kullanılır. Bu durumda, değişkenler ve işlemler mantıksal değişkenler ve işlemlerdir. Böylece, bir değişken 1 (DOĞRU) veya 0 (YANLIŞ) değerini alabilir. Temel mantıksal

---

<sup>1</sup> Bu makaleye [box.com/COA10e](http://box.com/COA10e) adresinden ulaşılabilir.

işlemler VE, VEYA ve DEĞİL'dir ve sembolik olarak nokta, artı işaretti ve üst çubuk ile temsil edilir:<sup>(2)</sup>

$$\begin{array}{l} A \text{ VE } B = A \cdot B \\ \text{VEYA } B = A + B \\ \text{DEĞİL} = \overline{A} \end{array}$$

AND işlemi, yalnızca ve yalnızca işlenenlerinin her ikisi de doğru (ikili değer 1) verir. OR işlemi, işlenenlerinden biri veya her ikisi de doğruya doğru sonucunu verir. Tekli işlem olan NOT, işleneninin değerini ters çevirir. Örneğin, şu denklemi düşünün

$$D = A + (B \cdot \overline{C})$$

A 1 ise veya hem B=0 hem de C=1 ise D 1'e eşittir. Aksi takdirde D 0'a eşittir.

Gösterimle ilgili birkaç noktaya ihtiyaç vardır. Parantezlerin olmadığı durumlarda AND işlemi OR işlemine göre önceliklidir. Ayrıca, herhangi bir belirsizlik oluşmadığında, AND işlemi nokta operatörü yerine basit birleştirme ile temsil edilir. Böylece,

$$A + B \cdot C = A + (B \cdot C) = A + BC$$

hepsi anlamına gelir: B ve C'nin VE'sini alın; ardından sonuç ve A'nın VEYA'sını alın.

Tablo 11.1a, temel mantıksal işlemleri *doğruluk tablosu* olarak bilinen bir biçimde tanımlar; bu tablo, işlenenlerin değerlerinin her olası kombinasyonu için bir işlemin değerini listeler. Tabloda ayrıca diğer üç kullanışlı operatör de listelenmektedir: XOR, NAND ve NOR. İki mantıksal operandın exclusive-or (XOR) işlemi, ancak ve ancak operandlardan tam olarak biri 1 değerine sahipse 1'dir. NAND işlemi AND işleminin tümleyenidir (NOT) ve NOR OR işleminin tümleyenidir:

$$\begin{aligned} A \text{ NAND } B &= \text{NOT}(A \text{ AND } B) = AB \cdot \overline{A} \overline{B} \\ \text{NOT}(A \text{ OR } B) &= A + B \end{aligned}$$

Göreceğimiz gibi, bu üç yeni işlem bazı dijital devrelerin gerçekleştirilemesinde faydalı olabilir.

Mantıksal işlemler, NOT hariç, Tablo 11.1b'de gösterildiği gibi ikiden değişikene genelleştirilebilir.

Tablo 11.2 Boole cebirinin temel özdeşliklerini özetlemektedir. Denklemler, VE ve VEYA işlemlerinin tamamlayıcı ya da ikili doğasını göstermek için iki sütun halinde düzenlenmiştir. İki sınıf özdeşlik vardır: kanıt olmadan ifade edilen temel kurallar (ya da postülatlar) ve temel postülatlardan türetilebilen diğer özdeşlikler. Önermeler Boolean ifadelerinin yorumlanması şeklini tanımlar. İki dağıtım yasasından biri dikkat çekicidir çünkü sıradan cebirde bulacağımızdan farklıdır:

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

---

<sup>2</sup> Mantıksal NOT genellikle kesme işaretti ile gösterilir: NOT A = A'.

**Tablo 11.1** Boolean Operatörleri

(a) İki Giriş Değişkeninin Boolean Operatörleri

P	Q	P DEĞİL ( $\bar{P}$ )	P AND Q ( $P \cdot Q$ )	P VEYA Q ( $P + Q$ )	P NA ND Q ( $P \cdot \bar{Q}$ )	P NOR Q ( $\bar{P} + Q$ )	P XOR Q ( $P \oplus Q$ )
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0

(b) İkiden Fazla Girişe Genişletilmiş Boole Operatörleri ( $A, B, \dots$ )

Operasyon	İfade	Çıktı= 1 olduğunda
VE	$A \cdot B \cdot C$	$A, B, \dots$ kümesinin tümü 1'dir.
VEYA	$A + B + C$	$A, B, \dots$ kümelerinden herhangi biri 1'dir.
NAND	$\overline{A \cdot B \cdot C}$	$A, B, \dots$ kümelerinden herhangi biri 0'dır.
NOR	$\overline{A + B + C}$	$A, B, \dots$ kümesinin tamamı 0'dır.
XOR	$A \oplus B \oplus C$	$A, B, \dots$ kümesi tek sayıda bir içerir.

En alttaki iki ifade DeMorgan teoremi olarak adlandırılır. Bunları aşağıdaki gibi yeniden ifade edebiliriz:

$$\begin{aligned} A \text{ NOR } B &= A \text{ AND } \overline{B} \\ \text{NAND } B &= A \text{ OR } B \end{aligned}$$

Okuyucu, A, B ve C değişkenleri için gerçek değerleri (1'ler ve 0'lar) yerine koyarak Tablo 11.2'deki ifadeleri doğrulamaya davet edilmektedir.

**Tablo 11.2** Boole Cebirinin Temel Özdeşlikleri

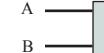
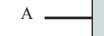
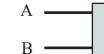
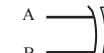
Temel Önermeler		
$A \cdot B = B \cdot A$	$A + B = B + A$	Değişmeli Yasalar
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B + C) = (A + B) + (A + C)$	Dağıtım Kanunları
$1 \cdot A = A$	$0 + A = A$	Kimlik Unsurları
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$	Ters Elemanlar
Diğer Kimlikler		
$0 \cdot A = 0$	$1 + A = 1$	
$A \cdot A = A$	$A + A = A$	
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$	Dernek Kanunları
$\overline{\overline{A}} = A$	$\overline{A + B} = \overline{A} \cdot \overline{B}$	DeMorgan Teoremi

## 11.2 KAPILAR

Tüm dijital mantık devrelerinin temel yapı taşı kapılardır. Mantıksal işlevler kapıların birbirine bağlanmasıyla gerçekleştirilir.

Bir geçit, giriş sinyalleri üzerinde basit bir Boole işlemi olan bir çıkış sinyali üreten elektronik bir devredir. Dijital mantıkta kullanılan temel kapılar AND, OR, NOT, NAND, NOR ve XOR'dur. Şekil 11.1'de bu altı kapı gösterilmektedir. Her kapı üç şekilde tanımlanır: grafik sembolü, cebirsel gösterim ve doğruluk tablosu. Bu bölümde kullanılan sembololoji IEEE standarı olan IEEE Std 91'den alınmıştır. Ters çevirme (NOT) işleminin bir daire ile gösterildiğine dikkat edin.

Şekil 11.1'de gösterilen her geçidin bir veya iki girişi ve bir çıkışı vardır. Bununla birlikte, Tablo 11.1b'de gösterildiği gibi, NOT hariç tüm kapıların ikiden girişi olabilir. Böylece,  $(X+Y+Z)$  üç girişli tek bir **VEYA kapısı** ile gerçekleştirilebilir. Girişteki değerlerden biri veya daha fazlası değiştirildiğinde, doğru çıkış sinyali neredeyse anında ortaya çıkar, sadece sinyallerin geçit boyunca yayılma süresi kadar gecikir (*geçit gecikmesi* olarak bilinir). Bu gecikmenin önemi Bölüm 11.3'te ele alınmaktadır. Bazı durumlarda, bir geçit iki çıkışla uygulanır, bir çıkış diğer çıkışın olumsuzlamasıdır.

İsim	Grafik Sembol	Cebirsel Fonksiyon	Doğruluk Tablosu															
VE		$F = A \cdot B$ veya $F = AB$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
VEYA		$F = A + B$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
DEĞİL		$F = \overline{A}$ veya $F = A'$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <th>A</th> <th>F</th> </tr> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND		$F = \overline{AB}$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{A+B}$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$F = A \oplus B$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

Şekil 11.1 Temel Mantık Kapıları

Burada yaygın bir terimi tanııyoruz: bir sinyali **ileri** sürmenin, bir sinyal hattının mantıksal olarak yanlış (0) durumundan mantıksal olarak doğru durumuna geçiş yapmasına neden olmak olduğunu söylüyoruz

(1) durumu. Gerçek (1) durumu, elektronik devrenin türüne bağlı olarak ya yüksek ya da düşük voltaj durumudur.

Tipik olarak, uygulamada tüm kapı türleri kullanılmaz. Sadece bir veya iki tip kapı kullanıldığında tasarım ve üretim daha basittir. Bu nedenle, *işlevsel olarak eksiksiz* kapı setlerini tanımlamak önemlidir. Bu, herhangi bir Boolean fonksiyonunun yalnızca kümelerdeki kapılar kullanılarak uygulanabileceği anlamına gelir. Aşağıdakeri işlevsel olarak eksiksiz kümelerdir:

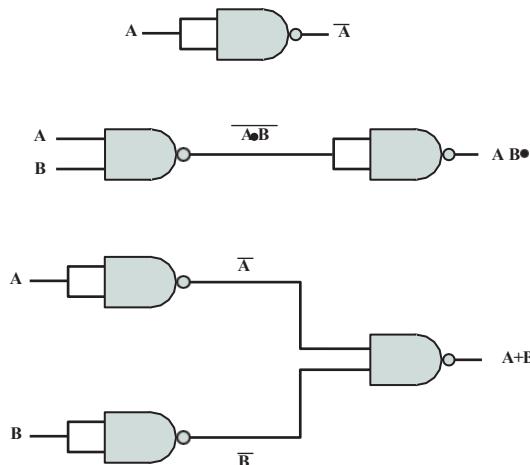
- VE, VEYA, DEĞİL
- VE, DEĞİL
- VEYA, DEĞİL
- NAND
- NOR

AND, OR ve NOT kapılarının Boole cebirinin üç işlemini temsil ettikleri için işlevsel olarak tam bir küme oluşturdukları açık olmalıdır. AND ve NOT kapılarının işlevsel olarak tam bir küme oluşturması için AND ve NOT işlemlerinden OR işlemini sentezlemenin bir yolu olmalıdır. Bu, DeMorgan teoremi uygulanarak yapılabilir:

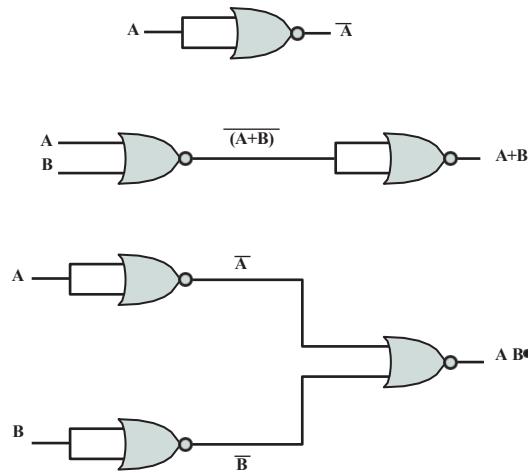
$$\begin{aligned} A + B &= A \overline{\overline{B}} \\ A \text{ VEYA } B &= \text{NOT}((\text{NOT } A) \text{AND } (\text{NOT } B)) \end{aligned}$$

Benzer şekilde, VEYA ve DEĞİL işlemleri AND işlemini sentezlemek için kullanılabildiğinden işlevsel olarak tamamlanmıştır.

Şekil 11.2 AND, OR ve NOT fonksiyonlarının sadece NAND kapıları ile nasıl gerçekleştirilebileceğini gösterirken, Şekil 11.3 aynı şeyi NOR kapıları için göstermektedir. Bu nedenle, dijital devreler yalnızca NAND kapıları veya yalnızca NOR kapıları ile gerçekleştirilebilir ve sıkılıkla da gerçekleştiriliyor.



**Şekil 11.2** NAND Kapılarının Bazı Kullanım Alanları



Şekil 11.3 NOR Kapılarının Bazı Kullanım Alanları

Kapılar ile bilgisayar donanımının en ilkel devre seviyesine ulaşmış bulunuyoruz. Kapıları oluşturmak için kullanılan transistör kombinasyonlarının incelenmesi bu alandan ayrılr ve elektrik mühendisliği alanına girer. Ancak bizim amacımız, kapıların dijital bir bilgisayarın temel mantıksal devrelerini uygulamak için yapı taşları olarak nasıl kullanabileceğini açıklamaktır.

### 11.3 KOMBİNASYONEL DEVRELER

**Kombinasyonel** bir **devre**, herhangi bir zamandaki çıktısı yalnızca o girdinin bir fonksiyonu olan birbirine bağlı bir kapılar kümesidir. Tek bir geçitte olduğu gibi, girişin ortaya çıkışını hemen ardından çıkışın ortaya çıkması izler, sadece geçit gecikmeleri olur.

Genel anlamda, bir kombinasyonel devre  $n$  adet ikili giriş ve  $m$  adet ikili girişten oluşur. ikili çıkışlar. Bir geçitte olduğu gibi, bir kombinasyonel devre üç şekilde tanımlanabilir:

- **Doğruluk tablosu:** Giriş sinyallerinin  $2^n$  olası kombinasyonunun her biri için,  $m$  çıkış sinyalinin her birinin ikili değeri listelenir.
- **Grafik semboller:** Kapıların birbirine bağlı düzene tasvir edilmiştir.
- **Boolean denklemleri:** Her bir çıkış sinyali, giriş sinyallerinin Boolean fonksiyonu olarak ifade edilir.

#### Boole Fonksiyonlarının Uygulanması

Herhangi bir Boolean fonksiyon elektronik ortamda bir kapılar ağı olarak uygulanabilir. Verilen herhangi bir fonksiyon için, bir dizi alternatif gerçekleme vardır. Tablo 11.3'teki doğruluk tablosu ile temsil edilen Boole fonksiyonunu düşünün. Bu fonksiyonu,  $F$ 'nin 1 olmasına neden olan  $A$ ,  $B$  ve  $C$  değerlerinin kombinasyonlarını basitçe maddeleyerek ifade edebiliriz:

$$F = \overline{ABC} + \overline{ABC} + \overline{ABC} \quad (11.1)$$

Tablo 11.3 Üç Değişkenli Bir Boole Fonksiyonu

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

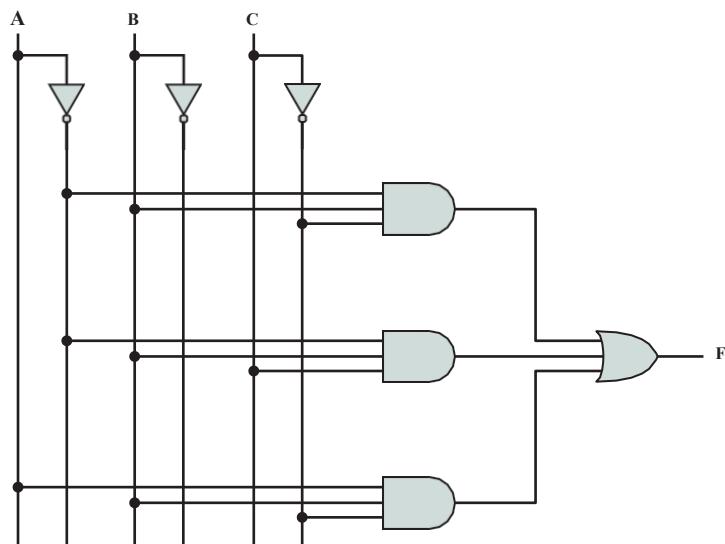
F'nin 1 olmasına neden olan üç giriş değeri kombinasyonu vardır ve bu kombinasyonlardan herhangi biri gerçekleşse sonuç 1 olur. Bu ifade biçimini, açık nedenlerden dolayı, **çarpımların toplamı (SOP)** biçimini olarak bilinir. Şekil 11.4 AND, OR ve NOT kapıları ile düz bir uygulamayı göstermektedir.

Doğruluk tablosundan başka bir form da türetilabilir. SOP formu, 1 üreten girdi kombinasyonlarından herhangi birinin doğru olması durumunda çıktıının 1 olduğunu ifade eder. Ayrıca 0 üreten girdi kombinasyonlarından hiçbirini doğru değilse çıktıının 1 olduğunu söyleyebiliriz. Böylece,

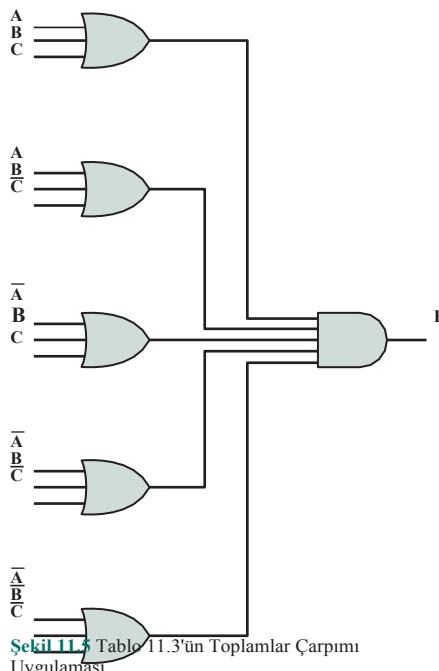
$$\overline{F} = \overline{(A B C)} \cdot \overline{(A B \bar{C})} \cdot \overline{(A \bar{B} C)} \cdot \overline{(A \bar{B} \bar{C})} \cdot \overline{(A B C)}$$

Bu, DeMorgan teoreminin bir genellemesi kullanılarak yeniden yazılabılır:

$$\overline{(X \cdot Y \cdot Z)} = \overline{X} + \overline{Y} + \overline{Z}$$



Şekil 11.4 Tablo 11.3'ten Ürün Toplamı Uygulaması



Böylece,

$$\begin{aligned} F &= (A + \overline{B+C}) \cdot (\overline{A} + \overline{B+C}) \cdot (\overline{A} + B + \overline{C}) \cdot (A + \overline{B+C}) \quad (11.2) \\ &= (A + B + C) \cdot (A + B + C) \cdot (A + B + C) \cdot (A + \overline{B+C}) \end{aligned}$$

Bu, Şekil 11.5'te gösterilen **toplamların çarpımı (POS)** formundadır. Açıklık için, DEĞİL kapıları gösterilmemiştir. Bunun yerine, her bir giriş sinyalinin ve tamamlayıcısının mevcut olduğu varsayılmaktadır. Bu, mantık diyagramını basitleştirir ve kapıların girişlerini daha kolay anlaşırlı hale getirir.

Böylece, bir Boolean fonksiyonu SOP ya da POS formunda gerçekleştirilebilir. Bu noktada, seçim doğruluk tablosunun çıkış fonksiyonu için daha fazla 1 mi yoksa 0 mi içerdigine bağlı gibi görülmektedir: SOP her 1 için bir terime sahiptir ve POS her 0 için bir terime sahiptir:

- Doğruluk tablosundan SOP veya POS'tan daha basit bir Boolean ifadesi türetmek genellikle mümkünündür.
- Fonksiyonun tek bir kapı tipi (NAND veya NOR) ile uygulanması tercih edilebilir.

İlk noktanın önemi, daha basit bir Boolean ifadesiyle, işlevi uygulamak için daha az kapıya ihtiyaç duyulacak olmasıdır. Basitleştirmeye ulaşmak için kullanılabilen üç yöntem şunlardır

- Cebirsel sadeleştirme
- Karnaugh haritaları
- Quine-McCluskey tabloları

**CEBİRSEL BASITLEŞTİRME** Cebirsel basitleştirme, Boolean ifadesini daha az elemanlı bir ifadeye indirmek için Tablo 11.2'deki özdeşliklerin uygulanmasını içerir. Örneğin, Denklem (11.1)'i tekrar ele alalım. Biraz düşünmek okuyucuya eşdeğer bir ifadenin şu olduğuna ikna etmeli

$$F = AB + BC \quad (11.3)$$

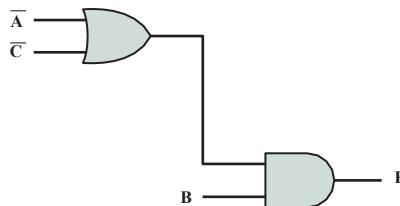
Ya da daha da basit,

$$F = B(\overline{A} + C) \quad \text{—}$$

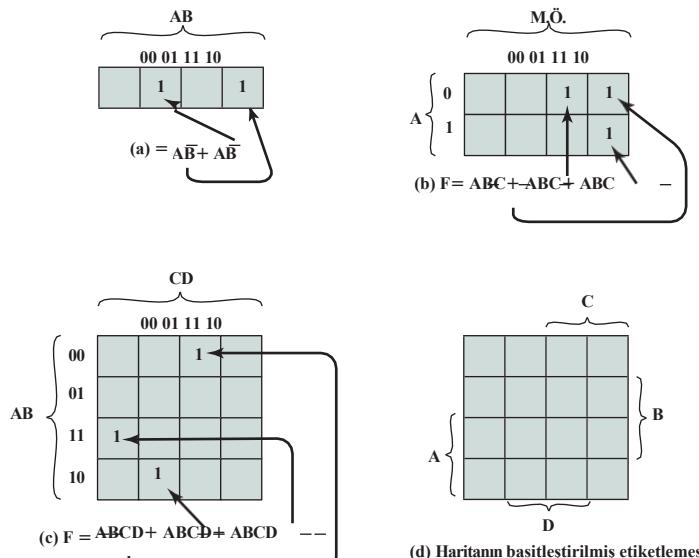
Bu ifade Şekil 11.6'da gösterildiği gibi uygulanabilir. Denklem (11.1)'in basitleştirilmesi esasen gözlem yoluyla yapılmıştır. Daha karmaşık ifadeler için daha sistematik bir yaklaşımaya ihtiyaç vardır.

**KARNAUGH HARİTALARI** Basitleştirme amacıyla, **Karnaugh haritası** az sayıda (en fazla dört) değişkeneden oluşan bir Boole fonksiyonunu temsil etmenin uygun bir yoludur. Harita,  $n$  ikili değişkenin değerlerinin tüm olası kombinasyonlarını temsil eden  $2^n$  kareden oluşan bir dizidir. Şekil 11.7a iki değişkenli bir fonksiyon için dört kareden oluşan haritayı göstermektedir. Daha sonraki amaçlar için kombinasyonların 00, 01, 11, 10 sırasına göre listelenmesi önemlidir. Kombinasyonlara karşılık gelen kareler bilgi kaydetmek için kullanılacağından, kombinasyonlar geleneksel olarak karelerin üzerine yazılır. Üç değişken söz konusu olduğunda, gösterim sekiz kareden oluşan bir düzlemedir (Şekil 11.7b), değişkenlerden birinin değerleri solda ve diğer iki değişkenin değerleri karelerin üzerinde yer alır. Dört değişken için, Şekil 11.7c'de gösterilen düzenleme ile 16 kareye ihtiyaç vardır.

Harita, herhangi bir Boole fonksiyonunu aşağıdaki şekilde temsil etmek için kullanılabilir. Her bir kare, değişkene karşılık gelen 1 değeri ve bu DEĞİL'ine karşılık gelen 0 değeri ile çarpımların toplamı formunda benzersiz bir çarpıma karşılık gelir. Böylece,  $AB$  çarpımı Şekil 11.7a'daki dördüncü kareye karşılık gelir. Fonksiyondaki bu tür her çarpım için 1 değeri ilgili yerleştirilir. Böylece, iki değişkenli örnek için harita  $AB + AB'$ ye karşılık . Bir Boole fonksiyonunun doğruluk tablosu verildiğinde, haritayı oluşturmak kolaydır: doğruluk tablosunda 1 sonucunu üreten değişkenlerin her bir değer kombinasyonu için, haritanın karşılık gelen karesini 1 ile doldurun. Şekil 11.7b, Tablo 11.3'ün doğruluk tablosu için sonucu göstermektedir. Bir Boole ifadesinden haritaya dönüştürmek için, öncelikle ifadeyi *kanonik* form olarak adlandırılan forma sokmak gereklidir: ifadedeki her terim her değişkeni içermelidir. Örneğin, Denklem (11.3)'e sahipsek, önce bunu Denklem (11.1)'in tam formuna genişletmeli ve ardından bunu bir haritaya dönüştürmeliyiz.



**Şekil 11.6** Tablo A.3'ün Basitleştirilmiş Uygulaması



Şekil 11.7 Boole Fonksiyonlarını Temsil Etmek için Karnaugh Haritalarının Kullanımı

Şekil 11.7'de kullanılan etiketleme, değişkenler ile haritanın satır ve sütunları arasındaki ilişkiyi vurgulamaktadır. Burada A simbolü tarafından kucaklanan iki satır A değişkeninin 1 değerine sahip olduğu satırlardır; A simbolü tarafından kucaklanmayan satırlar A'nın 0 olduğu satırlardır; benzer şekilde B, C ve D için de geçerlidir.

Bir fonksiyonun haritası oluşturduğuktan sonra, 1'lerin dizilişine dikkat ederek genellikle basit bir cebirsel ifade yazabiliriz. Prensip aşağıdaki gibidir. Bitişik olan herhangi iki kare değişkenlerden yalnızca birinde farklılık gösterir. İki bitişik karenin her ikisinde de bir girişi varsa, karşılık gelen çarpım terimleri yalnızca bir değişikende farklılık gösterir. Böyle bir durumda, iki terim o değişkeni ortadan kaldırarak birleştirilebilir. Örneğin, Şekil 11.8'a'da, bitişik iki kare ABCD ve ABCD terimlerine karşılık gelmektedir. Böylece, ifade edilen fonksiyon

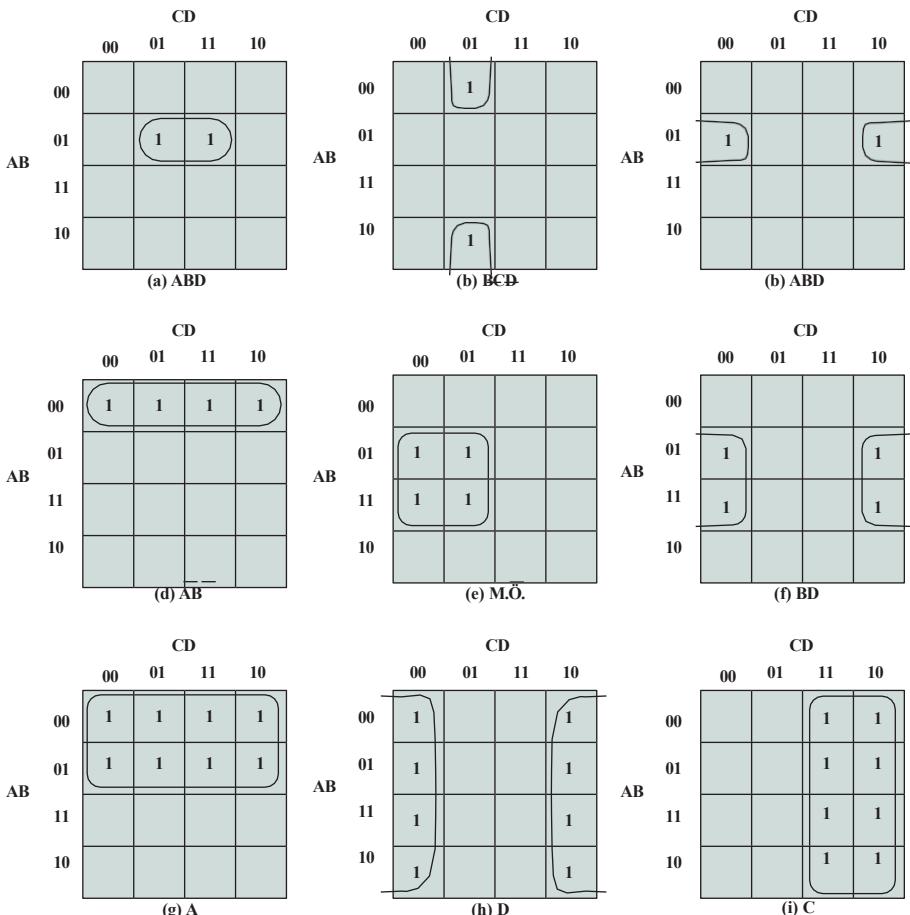
$$\overline{ABC\bar{D}} + \overline{AB\bar{C}D} = ABD$$

Bu süreç birkaç şekilde genişletilebilir. İlk olarak, bitişiklik kavramı haritanın kenarının etrafını sarmayı içerecek şekilde genişletilebilir. Böylece, bir sütunun en üst karesi en alt kareye ve bir satırın en sol karesi en sağ kareye bitişiktir. Bu koşullar Şekillerde gösterilmiştir

11.8b ve c. İkinci olarak, sadece 2 kareyi değil,  $2^{(n)}$  komşu kareyi (yani , 4, 8, vb.) gruplayabiliz. Şekil 11.8'deki sonraki üç örnek 4 karelük gruplamaları göstermektedir. Bu durumda değişkenlerden ikisinin elenebileceğini . Son üç örnek, üç değişkenin elenmesine olanak tanıyan 8 karelük gruplamaları göstermektedir.

Sadeleştirme kurallarını aşağıdaki gibi özetleyebiliriz:

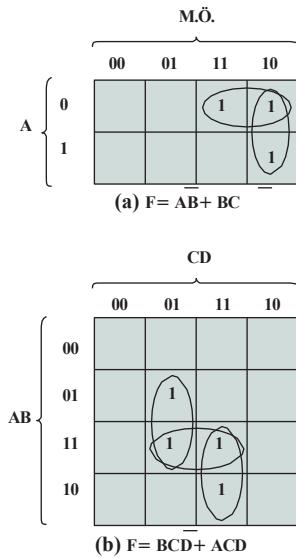
1. İşaretli kareler (1'li kareler) arasından 1, 2, 4 veya 8'li benzersiz en büyük bloğa ait olanları bulun ve bu blokları daire içine alın.



Şekil 11.8 Karnaugh Haritalarının Kullanımı

- Mümkün olduğunda büyük ve mümkün olduğunda az sayıda, ancak her işaretli kareyi en az bir kez içeren ek işaretli kare blokları seçin. Sonuçlar bazı durumlarda benzersiz olmayabilir. Örneğin, işaretli bir kare tam olarak diğer iki kareyle birleşiyorsa ve daha büyük bir grubu tamamlayacak dördüncü bir işaretli kare yoksa, iki gruptan hangisinin seçileceği konusunda bir seçim yapılması gereklidir. Grupları daire içine alırken, aynı 1 değerini birden fazla kez kullanmanızı izin verilir.
- Her işaretli kare en az bir ilmeğe ait olacak şekilde tek işaretli karelerin ya da bitişik işaretli kare çiftlerinin ya da dörtlü, sekizli vb. grupların etrafına ilmekler çizmeye devam edin; ardından tüm işaretli kareleri içerecek şekilde bu bloklardan mümkün olduğunda azını kullanın.

Tablo 11.3'e dayanan Şekil 11.9a, basitleştirme sürecini göstermektedir. Gruplamalardan sonra izole 1'ler kalırsa, bunların her biri bir 1'ler grubu olarak daire içine alınır.



Şekil 11.9 Örtüsen Gruplar

Son olarak, haritadan basitleştirilmiş bir Boole ifadesine geçmeden önce, diğer gruplarla tamamen örtüsen herhangi bir 1 grubu elenebilir. Bu durum Şekil 11.9b'de gösterilmektedir. Bu durumda, yatay grup gereksizdir ve Boolean ifadesinin oluşturulmasında göz ardı edilebilir.

Karnaugh haritalarının bir başka özelliğinden daha bahsetmek gereklidir. Bazı durumlarda, değişkenlerin değerlerinin belirli kombinasyonları asla oluşmaz ve bu nedenle karşılık gelen çıktı asla oluşmaz. Bunlar "umursama" koşulları olarak adlandırılır. Bu tür her koşul için haritanın ilgili karesine "d" harfi girilir. Gruplama ve sadeleştirme yapılırken, her bir "d" 1 veya 0 olarak ele alınabilir, hangisi en basit ifadeye yol açarsa.

HAYE98]'de sunulan bir örnek, tartıştığımız noktaları açıklamaktadır. Paketlenmiş bir ondalık basamağa 1 ekleyen bir devre için Boolean ifadeleri geliştirmek istiyoruz. Paketlenmiş ondalık için, her ondalık basamak bariz bir şekilde 4 bitlik bir kod ile temsil edilir. Böylece,  $0 = 0000$ ,  $1 = 0001$ ,  $C = 1000$  ve  $9 = 1001$ . Kalan 4 bitlik değerler, 1010'dan 1111'e kadar, kullanılmaz. Bu kod aynı zamanda **İkili Kodlu Ondalık (BCD)** olarak da adlandırılır.

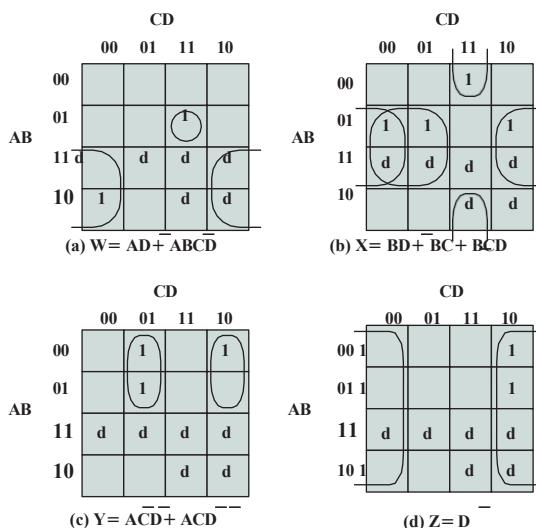
Tablo 11.4, 4 bitlik bir BCD girişinden bir fazla olan 4 bitlik bir sonuç üretmek için doğruluk tablosunu göstermektedir. Toplama işlemi modulo 10'dur. Böylece,  $9 + 1 = 0$ . Ayrıca, giriş kodlarından altısının "umurumda değil" sonuçları ürettiğine dikkat edin, çünkü bunlar geçerli BCD girişleri değildir. Şekil 11.10, çıktı değişkenlerinin her biri için ortaya çıkan Karnaugh haritalarını göstermektedir. Mümkün olan en iyi gruplamaları elde etmek için d kareleri kullanılmıştır.

**QUINE-McCLUSKEY YÖNTEMİ** Dörtten fazla değişken için Karnaugh harita yöntemi giderek daha hantal hale gelir. Beş değişkenle, iki  $16 \times 16$  haritaya ihtiyaç duyulur ve bitişikliği sağlamak için bir haritanın üç boyutta diğerinin üzerinde olduğu düşünülür. Altı değişken dört adet  $16 \times 16$  harita kullanılmasını gerektirir.

**Tablo 11.4** Bir Basamaklı Paketlenmiş Ondalık Arttırıcı için Doğruluk Tablosu

Sayı	Giriş				Sayı	Çıktı			
	A	B	C	D		W	X	Y	Z
0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	2	0	0	1	0
2	0	0	1	0	3	0	0	1	1
3	0	0	1	1	4	0	1	0	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	6	0	1	1	0
6	0	1	1	0	7	0	1	1	1
7	0	1	1	1	8	1	0	0	0
8	1	0	0	0	9	1	0	0	1
9	1	0	0	1	0	0	0	0	0
	1	0	1	0		d	d	d	d
	1	0	1	1		d	d	d	d
Umurumda değil durumu	1	1	0	0		d	d	d	d
	1	1	0	1		d	d	d	d
	1	1	1	0		d	d	d	d
	1	1	1	1		d	d	d	D

dört boyutlu tablolar! Alternatif bir yaklaşım, Quine-McCluskey yöntemi olarak adlandırılan bir tablo teknigidir. Bu yöntem, minimize edilmiş Boolean ifadeleri üretmek için otomatik bir araç sağlamak üzere bilgisayarda programlanmaya uygundur.

**Şekil 11.10** Arttırıcı için Karnaugh Haritaları

Yöntem en iyi şekilde bir örnekle açıklanabilir. Aşağıdaki ifadeyi ele alalım:

$$ABCD+ ABCD+ \overline{ABC} D+ \overline{ABC}\overline{D}+ ABCD+ \overline{ABCD}+ ABCD+ \overline{ABCD}+ A \overline{B} CD = \dots$$

Bu ifadenin bir doğruluk tablosundan türetildiğini varsayılm. Geçitlerle uygulamaya uygun minimal bir ifade üretmek istiyoruz.

İlk adım, her bir satırın ifadenin çarpım terimlerinden karşılık geldiği bir tablo oluşturmaktır. Terimler, tamamlanan değişkenlerin sayısına göre grupperdirilir. , hiç tümleyeni olmayan terimle başlarız, eğer varsa, sonra bir tümleyeni olan tüm terimler ve bu böyle devam eder. Tablo 11.5 örnek ifademiz için listeyi göstermektedir ve yatay çizgiler grupperdirilmeyi belirtmek için kullanılmıştır. Anlaşıllır olması için, her terim, tamamlanmamış her değişken için bir 1 ve tamamlanmış her değişken için bir 0 ile temsil edilir. Böylece, terimleri içerdikleri 1'lerin sayısına göre grupperdiririz. İndeks sütunu basitçe ondalık eşdeğerdir ve aşağıda anlatılanlar için kullanılmıştır.

Bir sonraki adım, yalnızca bir değişkende farklılık gösteren tüm terim çiftlerini, bir değişkenin terimlerinden birinde 0, diğerinde 1 olması dışında aynı olan tüm terim çiftlerini bulmaktadır. Terimleri grupperdirme şeklimiz nedeniyle, bunu ilk grubtan başlayarak ve ilk grubun her terimini ikinci grubun her terimiyle karşılaştırarak . Sonra ikinci grubun her terimini üçüncü grubun tüm terimleriyle karşılaşır ve bu şekilde devam edin. Bir eşleşme bulunduğuanda, her terimin yanına bir onay işaretini koyun, iki terimde farklı olan değişkeni ortadan kaldırarak çifti birleştirin ve bunu yeni bir listeye ekleyin. Böylece, örneğin, ABCD ve ABCD terimleri birleştirilerek ABC elde edilir. Bu işlem orijinal tablonun tamamı incelenene kadar devam eder. Sonuç, aşağıdaki girdileri içeren yeni bir tablodur:

$\overline{B} \overline{I} \overline{R} \overline{C} \overline{D}$	$\overline{A} \overline{B} \overline{C} \overline{D}$	$\overline{A} \overline{B} \overline{D} \overline{U}$	$\overline{A} \overline{B} \overline{C} \overline{D} \overline{U}$	$\overline{A} \overline{B} \overline{C} \overline{D} \overline{U}$	$\overline{A} \overline{B} \overline{C} \overline{D} \overline{U}$	$\overline{A} \overline{B} \overline{C} \overline{D} \overline{U}$
		$\overline{B} \overline{C} \overline{D} \overline{U}$	$\overline{A} \overline{C} \overline{D}$	$\overline{A} \overline{B} \overline{D} \overline{U}$		
		$\overline{A} \overline{B} \overline{C}$	$\overline{B} \overline{C} \overline{D}$			
		$\overline{A} \overline{B} \overline{D}$				
				$\overline{A} \overline{B} \overline{C} \overline{D}$		
					$\overline{A} \overline{B} \overline{C} \overline{D}$	
						$\overline{A} \overline{B} \overline{C} \overline{D}$

**Tablo 11.5** Quine-McCluskey Yönteminin İlk Aşaması

(F için=  $ABCD+ ABCD+ \overline{ABC} D+ \overline{ABC}\overline{D}+ ABCD+ \overline{ABCD}+ ABCD+ \overline{ABCD}+ ABCD$ )

Ürün Terimi	Dizin	A	B	C	D	
$\overline{B} \overline{I} \overline{R} \overline{C} \overline{D}$	1	0	0	0	1	✓
$\overline{A} \overline{B} \overline{C} \overline{D}$	5	0	1	0	1	✓
$\overline{A} \overline{B} \overline{C} \overline{D}$	6	0	1	1	0	✓
$\overline{A} \overline{B} \overline{C} \overline{D}$	12	1	1	0	0	✓
$\overline{A} \overline{B} \overline{C} \overline{D}$	7	0	1	1	1	✓
$\overline{A} \overline{B} \overline{C} \overline{D}$	11	1	0	1	1	✓
$\overline{A} \overline{B} \overline{C} \overline{D}$	13	1	1	0	1	✓
$A \overline{B} \overline{C} \overline{D}$	15	1	1	1	1	✓

Yeni tablo, ilk aynı şekilde, belirtildiği gibi gruplar halinde düzenlenir. İkinci tablo daha sonra ilkiyle aynı şekilde işlenir. , sadece bir değişkende farklılık gösteren terimler kontrol edilir ve üçüncü bir tablo için yeni bir terim üretilir. Bu örnekte, üretilen üçüncü tablo sadece bir terim içermektedir: BD.

Genel olarak, eşleşme olmayan bir tablo üretilinceye kadar süreç birbirini izleyen tablolar üzerinden ilerleyecektir. Bu durumda, bu üç tabloyu içermiştir.

Az önce açıklanan süreç tamamlandığında, ifadenin olası terimlerinin çoğunu elemiş oluruz. Elenmemiş olan bu terimler Tablo 11.6'da gösterildiği gibi bir matris oluşturmak için kullanılır. Matrisin her satırı, şimdije kadar kullanılan tabloların hiçbirinde elenmemiş (kontrolü olmayan) terimlerden birine karşılık gelir. Her sütun orijinal ifadedeki terimlerden birine karşılık gelir. Bir satır ve bir sütunun her kesişimine, satır elemanı sütun elemanı ile "uyumlu" olacak şekilde bir X yerleştirilir. Yani, satır elemanında bulunan değişkenler sütun elemanında bulunan değişkenlerle aynı değere sahiptir. Daha sonra, bir sütündeki tek başına bulunan her bir  $X'$ 'i daire içine alın. Ardından, daire içine alınmış bir  $X'$ 'in bulunduğu herhangi bir satırda daire içine alınmış bir  $X'$ 'in etrafına bir kare yerleştirin. Her sütunda artık ya kare içine alınmış ya da daire içine alınmış bir  $X$  varsa, o zaman işimiz bitmiştir ve  $X'$ 'leri işaretlenmiş olan satır öğeleri minimal ifadeyi oluşturur. Böylece, örneğimizde, nihai ifade

$$\overline{A}BC + ACD + \overline{A}\overline{B}C + ACD$$

Bazı sütunlarda ne bir daire ne de bir kare olduğu durumlarda, ek işlem yapılması gereklidir. Esasen, tüm sütunlar kapsanana kadar satır öğeleri eklemeye devam ediyoruz.

Quine-McCluskey yöntemini özetleyerek neden işe yaradığını sezgisel olarak açıklamaya çalışalım. İşlemenin ilk aşaması oldukça basittir. İşlem, çarpım terimlerindeki gereksiz değişkenleri ortadan kaldırır. Böylece,  $ABC + ABC$  ifadesi  $AB$ 'ye eşdeğerdir, çünkü

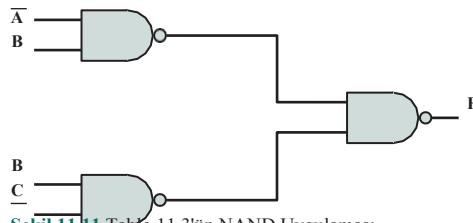
$$\overline{A}BC + \overline{A}\overline{B}C = AB(C + C) = \overline{A}B$$

Değişkenlerin elenmesinden sonra, elimizde orijinal ifadeye açıkça eşdeğer olan bir ifade kalır. Ancak, tipki Karnaugh haritalarında gereksiz grüplamalar bulduğumuz gibi, bu ifadede de gereksiz terimler olabilir. Matris düzeli, orijinal ifadedeki her terimin kapsammasını sağlar ve bunu nihai ifadedeki terim sayısını en azı indirecek şekilde yapar.

**Tablo 11.6** Quine-McCluskey Yönteminin Son Aşaması

(F için=  $ABCD + ABCD + AB\overline{C}D + ABCD + \overline{A}BCD + \overline{A}\overline{B}CD + ABC\overline{D} + ABCD$ )

	$ABCD$	$\overline{ABCD}$	$ABC\overline{D}$	$\overline{ABC}D$	$\overline{AB}CD$	$\overline{A}BCD$	$\overline{ABC}D$	$\overline{AB}C\overline{D}$	$\overline{ABC}\overline{D}$
BD	X	X				X		X	
$\overline{ACD}$								[X]	$\otimes$
$\overline{ABC}$					[X]		$\otimes$		
$\overline{ABC}$		[X]	$\otimes$						
ACD	[X]			$\otimes$					



Şekil 11.11 Tablo 11.3'ün NAND Uygulaması

**NAND VE NOR UYGULAMALARI** Boole fonksiyonlarının uygulanmasında dikkat edilmesi gereken bir diğer husus da kullanılan kapı türleriyle ilgilidir. Bazen bir Boole fonksiyonunun yalnızca NAND kapıları veya yalnızca NOR kapıları ile uygulanması istenebilir. Bu minimum kapı uygulaması olmasa da, üretim sürecini basitleştirebilecek düzenlilik avantajına sahiptir. Denklem (11.3)'ü tekrar düşünün:

$$F = B(\bar{A} + \bar{C})$$

Çünkü bir değerin tümleyeninin tümleyeni sadece orijinal değerdir,

$$F = \overline{B}(\overline{A} + \overline{C}) = (\overline{\overline{AB}} + \overline{\overline{BC}})$$

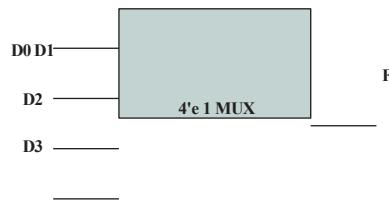
DeMorgan teoremini uygulamak,

$$F = \overline{\overline{(AB)}} - \overline{\overline{(BC)}}$$

Şekil 11.11'de gösterildiği gibi üç NAND formuna sahiptir.

### Çoklayıcılar

**Çoklayıcı** birden fazla girişi tek bir çıkışa bağlar. Herhangi bir zamanda, girişlerden biri çıkışa aktarılmak üzere seçilir. Genel bir blok diyagramı gösterimi Şekil 11.12'de gösterilmiştir. Bu, 4'e 1 çoklayıcıyı temsil eder. D0, D1, D2 ve D3 olarak etiketlenmiş dört giriş hattı vardır. Bu hatlardan biri çıkışa sağlamak için seçilir



Şekil 11.12 4'ten 1'e Çoklayıcı Gösterimi

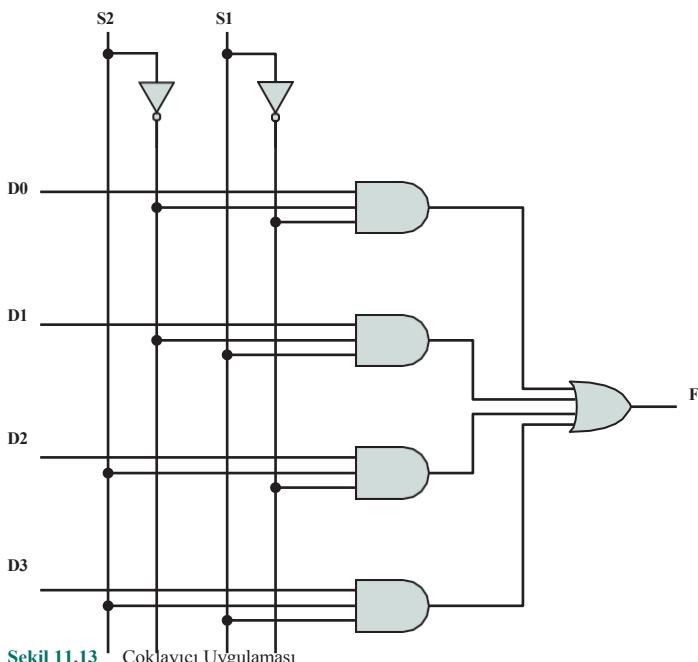
Tablo 11.7 4'ten 1'e Çoklayıcı Doğruluk Tablosu

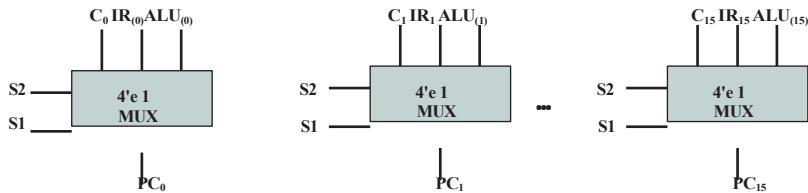
S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

F sinyali. Dört olası girişten birini seçmek için 2 bitlik bir seçim kodu gereklidir ve bu S1 ve S2 etiketli iki seçim hattı olarak uygulanır.

Örnek bir 4'ten 1'e çoklayıcı Tablo 11.7'deki doğruluk tablosu ile tanımlanır. Bu doğruluk tablosunun basitleştirilmiş bir şeklidir. Giriş değişkenlerinin tüm olası kombinasyonlarını göstermek yerine, çıkışlı D0, D1, D2 veya D3 hattından gelen veri olarak gösterir. Şekil 11.13 AND, OR ve NOT kapılarını kullanan bir uygulamayı göstermektedir. S1 ve S2, AND kapılarına öyle bir şekilde bağlanmıştır ki, S1 ve S2'nin herhangi bir kombinasyonu için, AND kapılarından üçü 0 çıkışını verecektir. Dördüncü AND kapısı, 0 veya 1 olan seçilen satırın değerini verecektir. Böylece VEYA kapısının girişlerinden üçü her zaman 0 olur ve VEYA kapısının çıkışı seçilen giriş kapısının değerine eşit olur. Bu düzenli organizasyonu kullanarak, 8'e 1, 16'ya 1 ve benzeri boyutlarda çoklayıcılar oluşturmak kolaydır.

Çoklayıcılar dijital devrelerde sinyal ve veri yönlendirmesini kontrol etmek için kullanılır. Program **sayacının** (PC) yüklenmesi buna bir örnektir. Program sayacına yüklenecek değer birkaç farklı kaynaktan birinden gelebilir:





**Şekil 11.14** Program Sayacına Çoklayıcı Girişи

- PC bir sonraki komut için artırılacaksa ikili bir sayaç.
- Doğrudan adres kullanan bir dallanma komutu henüz yürütülmüşse, komut **kayıdı**.
- Dallanma komutu bir yer değiştirme modu kullanarak adresi belirtiyorsa, ALU'nun çıkışını.

Bu çeşitli girişler, PC çıkış hattına bağlı olacak şekilde bir giriş hatlarına bağlanabilir. Seçme hatları PC'ye hangi değerin yükleneceğini belirler. PC birden fazla bit içerdiginden, her bit için bir tane olmak üzere birden fazla çoklayıcı kullanılır. Şekil 11.14 bunu 16 bitlik adresler için göstermektedir.

### Kod Çözücüler

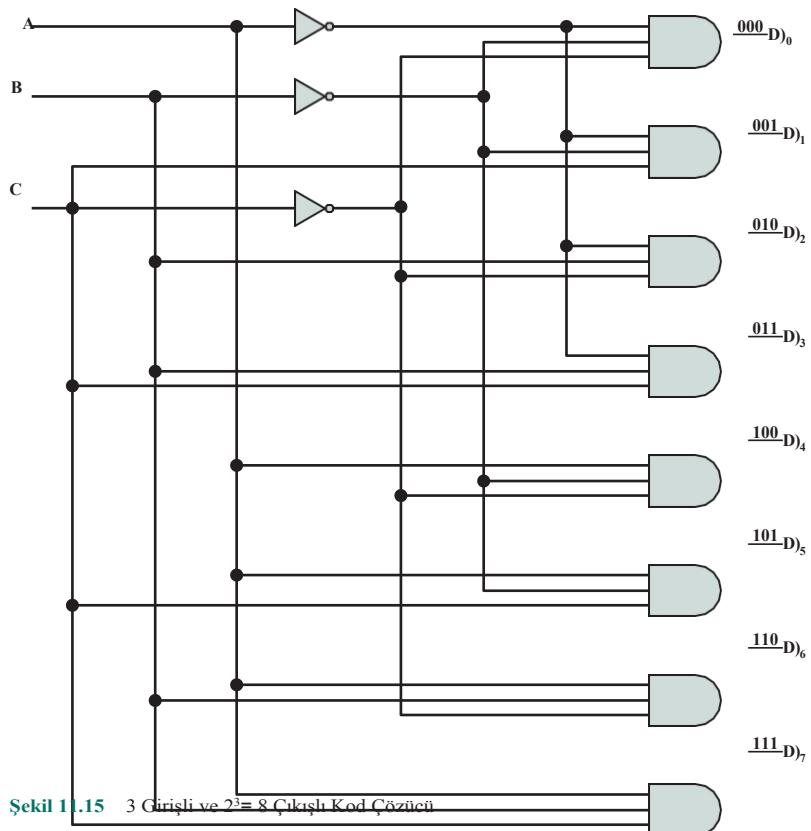
Bir **kod çözücü**, herhangi bir zamanda yalnızca bir tanesi onaylanan bir dizi çıkış hattına sahip bir kombinasyonel devredir. Hangi çıkış hattının onaylanacağı giriş hatlarının modeline bağlıdır. Genel olarak, bir kod çözücü  $n$  girişe ve  $2^n$  çıkışa sahiptir. Şekil 11.15 üç girişli ve sekiz çıkışlı bir kod çözücüyü göstermektedir.

Kod çözüçüler dijital bilgisayarlarda birçok kullanım alanı bulur. Buna bir örnek adres kod çözme işlemidir. Dört adet  $2 \times 6 \times 8$  @bit RAM yongası kullanarak 1K baytlık bir bellek oluşturmak istediğimizi varsayıyalım. Aşağıdaki gibi parçalanabilen tek bir birleşik adres alanı istiyoruz:

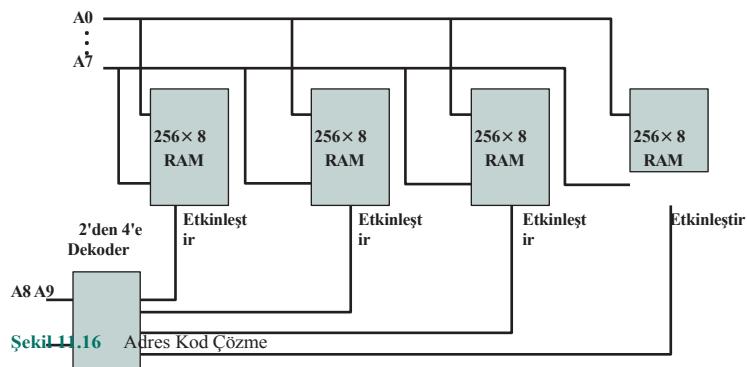
Adres	Çip
0000-00FF	0
0100-01FF	1
0200-02FF	2
0300-03FF	3

Her çip 8 adres hattı gerektirir ve bunlar adresin alt sıradaki 8 biti tarafından sağlanır. 10 bitlik adresin üst sıradaki 2 biti dört RAM yongasından birini seçmek için kullanılır. Bu amaçla, Şekil 11.16'da gösterildiği gibi, çıkış dört yongadan birini etkinleştiren bir 2'den 4'e kod çözücü kullanılır.

Ek bir giriş hattıyla, bir kod çözücü bir demultiplexer olarak kullanılabilir. Demultiplexer bir multiplexer'in ters işlevini yerine getirir; tek bir girişi birkaç çıkıştan birine bağlar. Bu Şekil 11.17'de gösterilmiştir. Daha önce olduğu gibi,  $2^n$  çıkıştan birini üretmek için  $n$  girişin kodu çözülür.  $2^n$  çıkış hattının tümü VE'lenir



**Şekil 11.15** 3 Girişli ve  $2^3=8$  Çıktılı Kod Çözücü



**Şekil 11.16** Adres Kod Çözme



Şekil 11.17 Kod Çözücü Kullanan Bir Demultiplexer Uygulaması

bir veri giriş hattı ile. Böylece,  $n$  girişleri belirli bir çıkış hattını seçmek için bir adres görevi görür ve veri giriş hattındaki değer (0 veya 1) bu çıkış hattına yönlendirilir.

Şekil 11.17'deki yapılandırma başka bir şekilde de görüntülenebilir. Yeni satırda etiketi *Veri Girişi'nden Etkinleştir* olarak değiştirin. Bu, kod çözücüünün zamanlamasının kontrol edilmesini sağlar. Kod çözülmüş çıkış yalnızca kodlanmış giriş mevcut ve etkinleştirme satırı 1 değerine sahip olduğunda görünür.

### Salt Okunur Bellek

Kombinasyonel devreler genellikle "hafızasız" devreler olarak adlandırılır, çünkü çıktıları yalnızca mevcut girişlerine bağlıdır ve önceki girişlerin geçmişi tutulmaz. Bununla birlikte, kombinasyonel devrelerde uygulanan bir tür bellek vardır, yani **salt okunur bellek (ROM)**.

ROM'un yalnızca okuma işlemini gerçekleştiren bir bellek birimi olduğunu hatırlayın. Bu, ROM'da depolanan ikili bilginin kalıcı olduğu ve üretim süreci sırasında oluşturulduğu anlamına gelir. Dolayısıyla, ROM'a verilen bir girdi (adres satırları) her zaman aynı çıktıyı (veri satırları) üretir. Çıkışlar sadece mevcut girişlerin bir fonksiyonu olduğundan, ROM aslında bir kombinasyonel devredir.

Bir ROM, bir kod çözücü ve bir dizi VEYA kapısı ile uygulanabilir. Örnek olarak Tablo 11.8'i ele alalım. Bu, dört giriş ve dört çıkış sahip bir doğruluk tablosu olarak görülebilir. Olası 16 giriş değerinin her biri için, çıkışların gelen değer kümesi gösterilmektedir. Her biri 4 bitlik 16 kelimeden oluşan 64 bitlik bir ROM'un içeriğini tanımlıyor olarak da görülebilir. Dört giriş bir adres belirtir ve dört çıkış adres tarafından belirlenen konumun içeriğini belirtir. Şekil 11.18 bu belleğin 4-16 kod çözücü ve dört VEYA kapısı kullanılarak nasıl uygulanabileceğini göstermektedir. PLA'da olduğu gibi, düzenli bir organizasyon kullanılır ve ara bağlantılar istenen sonucu yansıtacak şekilde yapılır.

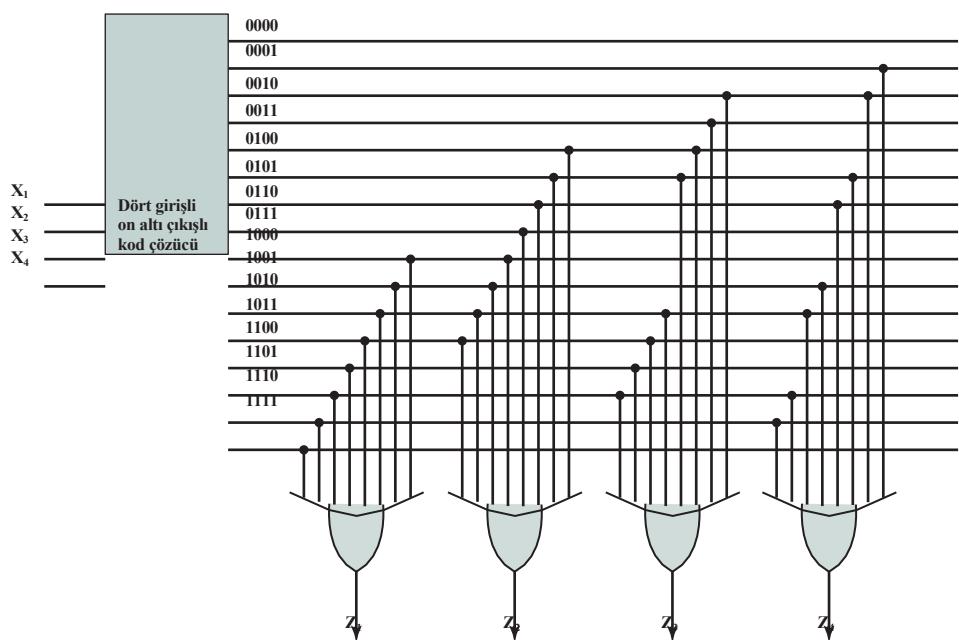
### Adders

Şimdiye kadar, sinyallerin yönlendirilmesi, kod çözme ve ROM gibi işlevleri yerine getirmek için birbirine bağlı kapıların nasıl kullanılabilceğini gördük. Henüz ele alınmamış önemli bir alan ise aritmetiktir. Bu kısa genel bakışta, toplama işlevine bakmakla yetineceğiz.

İkili toplama işlemi, sonucun bir taşıma terimi içermesi bakımından Boole cebirinden farklıdır. Böylece,

**Tablo 11.8** Bir ROM için Doğruluk Tablosu

Giriş				Çıktı			
$X_1$	$X_2$	$X_3$	$X_4$	$Z_1$	$Z_2$	$Z_3$	$Z_4$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0



**Şekil 11.18** 64-Bit ROM

Tablo 11.9 İkili Toplama Doğruluk Tabloları

(a) Tek-Bit Toplama				(b) Carry Giriş ile Toplama				
A	B	Toplam	Taşıma	Cin	A	B	Toplam	Cout
0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	1	1	0
1	0	1	0	0	1	0	1	0
1	1	0	1	1	1	0	0	1

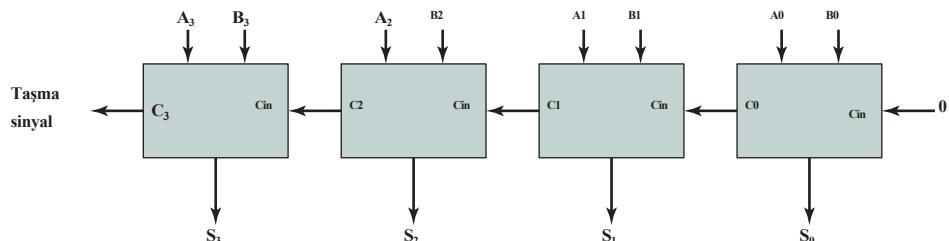
$$\begin{array}{r}
 \begin{array}{|c|c|c|c|} \hline
 0 & 0 & 1 & 1 \\ \hline
 + 0 & + 1 & + 0 & + 1 \\ \hline
 0 & 1 & 1 & 10 \\ \hline
 \end{array}
 \end{array}$$

Ancak toplama işlemi yine de Boolean terimleriyle ele alınabilir. Tablo 11.9a'da, 1 bitlik bir toplam ve bir taşıma biti üretmek için iki giriş bitini toplama mantığını gösteriyoruz. Bu doğruluk tablosu dijital mantıkta kolayca uygulanabilir. Ancak, biz sadece tek bir bit çifti üzerinde toplama işlemi yapmakla ilgilenmemyorum. Bunun yerine, iki *n-bitlik* sayıyı toplamak istiyoruz. Bu, bir **toplayıcıdan** gelen carry biti bir sonrakine giriş olarak sağlanacak şekilde bir dizi toplayıcıyı bir araya getirerek yapılabilir. Şekil

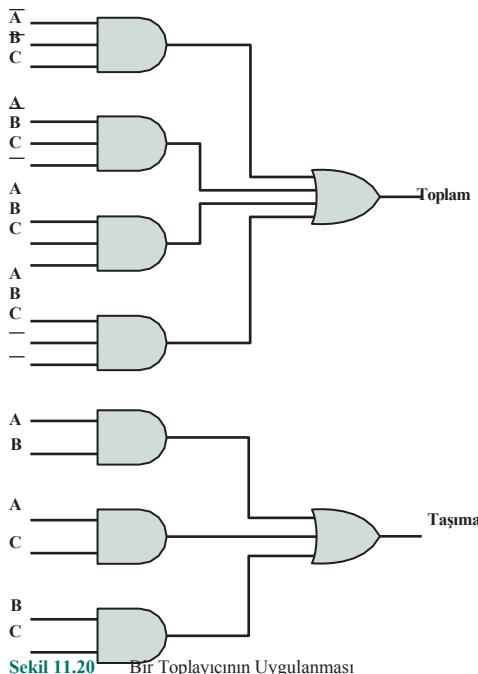
11.19'da 4 bitlik bir toplayıcı gösterilmektedir. Çok bitli toplayıcının çalışması için, tek bitli toplayıcıların her birinin, bir sonraki düşük dereceli toplayıcıdan gelen taşıma dahil olmak üzere üç giriş olmalıdır. Revize edilmiş doğruluk tablosu Tablo 11.9b'de görülmektedir. İki çıkış ifade edilebilir:

$$\begin{aligned}
 \text{Toplam} = & \overline{ABC} + ABC + \overline{ABC} + ABC \\
 \text{Taşıma} = & \overline{AB} + \\
 & AC + BC
 \end{aligned}$$

Şekil 11.20 AND, OR ve NOT kapılarını kullanan bir uygulamadır.



Şekil 11.19 4-Bitlik Toplayıcı

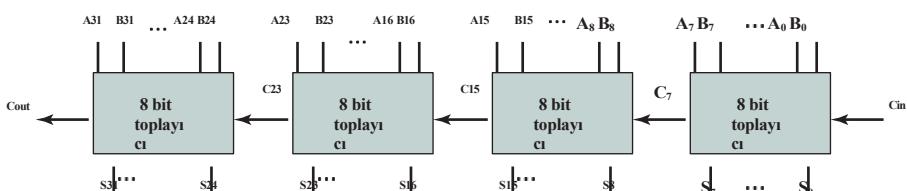


Şekil 11.20 Bir Toplayıcının Uygulanması

Böylece Şekil 11.21'de gösterildiği gibi çok bitli bir toplayıcıyı uygulamak için gerekli mantıka sahip oluruz. Her toplayıcının çıkışı bir önceki toplayıcıdan gelen carry'ye bağlı olduğundan, en az anlamlı bitten en anlamlı bite doğru artan bir gecikme olduğunu unutmayın. Her bir tek bitlik toplayıcı belirli miktarda kapı gecikmesi yaşar ve bu kapı gecikmesi birikir. Daha büyük toplayıcılar için, biriken gecikme kabul edilemeyecek kadar yüksek olabilir.

Taşıma değerleri önceki tüm aşamalarda dalgalandırmak zorunda kalmadan belirlenebilseydi, her bir tek bitlik toplayıcı bağımsız olarak çalışabilir ve gecikme birikmezdi. Bu, *carry lookahead* olarak bilinen bir yaklaşımla başarılabilir. Bu yaklaşımı açıklamak için 4 bitlik toplayıcıya tekrar bakalım.

Onceki carry değerlerine atıfta bulunmadan toplayıcının herhangi bir aşamasına carry girişini belirten bir ifade bulmak istiyoruz. Elimizde



Şekil 11.21 8-Bitlik Toplayıcılar Kullanarak 32-Bitlik Toplayıcı Oluşturma

$$C_0 = A_0 B_0 \quad (11.4)$$

$$C_1 = A_{(1)} B_1 + A_{(1)} A_0 B_{(0)} + B_{(1)} A_{(0)} B_{(0)} \quad (11.5)$$

Aynı prosedürü izleyerek şunları elde ederiz

$$\begin{aligned} C_2 = & A_{(2)} B_2 + A_{(2)} A_1 B_{(1)} + A_{(2)} A_{(1)} A_0 B_0 + A_{(2)} B_{(1)} A_{(0)} B_{(0)} + B_{(2)} A_{(1)} B_{(1)} \\ & + B_2 A_1 A_0 B_{(0)} + B_2 B_1 A_{(0)} B_{(0)} \end{aligned}$$

Bu işlem keyfi uzunluktaki toplayıcılar için tekrarlanabilir. Her bir taşıma terimi, SOP formunda yalnızca orijinal girişlerin bir fonksiyonu olarak ifade edilebilir ve terimlerine bağlı değildir. Böylece, toplayıcının bağımsız olarak sadece iki kapı gecikmesi seviyesi oluşur.

Uzun sayılar için bu yaklaşım aşırı derecede karmaşık hale gelir. N bitlik bir toplayıcının en anlamlı *bitti* için ifadenin değerlendirilmesi,  $2^n - 1$  girişli bir VEYA kapısı ve 2 ile  $n+1$  girişli  $2(n) - 1$  VE kapıları gerektirir. Buna göre, tam carry lookahead tipik olarak bir seferde sadece 4 ile 8 bit yapılır. Şekil 11.21, 32 bitlik bir toplayıcının dört adet 8 bitlik toplayıcıdan nasıl oluşturulabileceğini göstermektedir. Bu durumda, taşıma dört 8 bitlik toplayıcı boyunca dalgalanmalıdır, ancak bu otuz iki 1 bitlik toplayıcı boyunca dalgalanmadan önemli ölçüde daha hızlı olacaktır.

## 11.4 SIRALI DEVRELER

Kombinasyonel devreler dijital bir bilgisayarın temel işlevlerini yerine getirir. Ancak, ROM'un özel durumu dışında, dijital bir bilgisayarın çalışması için de gerekli olan bellek veya durum bilgisi sağlanamazlar. İkinci amaçlar için, daha karmaşık bir dijital mantık devresi biçimini kullanılır: **sıralı devre**. Sıralı bir devrenin mevcut çıktısı sadece mevcut girdiye değil, aynı zamanda girdilerin geçişine de bağlıdır. Bunu görmemnin bir başka ve genellikle daha kullanışlı yolu, sıralı bir devrenin mevcut çıkışının mevcut girişe ve bu devrenin mevcut durumuna bağlı olduğunu.

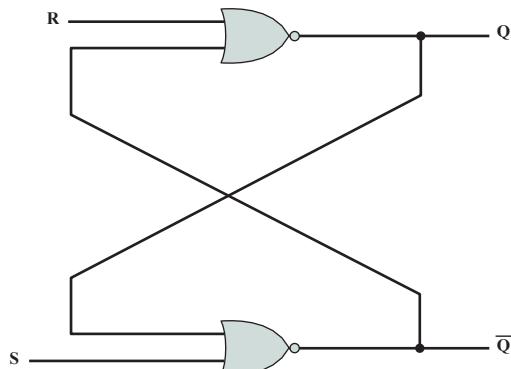
Bu bölümde, ardışıl devrelerin bazı basit ama kullanışlı örneklerini inceleyeceğiz. Görüleceği üzere, ardışıl devre kombinasyonel devrelerden yararlanmaktadır.

### Parmak Arası Terlikler

Sıralı devrenin en basit şekli **flip-flop'tur**. Hepsi iki özelliği paylaşan çeşitli flip-flops vardır:

- Flip-flop iki durumlu bir cihazdır. İki durumdan birinde bulunur ve giriş olmadığından o durumda kalır. Böylece, flip-flop 1 bitlik bir bellek olarak işlev görebilir.
- Flip-flop'un her zaman birbirinin tamamlayıcısı olan iki çıkışı vardır. Bunlar genellikle Q ve  $\bar{Q}$  olarak etiketlenir.

**S-R MANDALI** Şekil 11.22, S-R flip flop veya **S-R mandali** olarak bilinen yaygın bir konfigürasyonu göstermektedir. Devrenin S (Set) ve R (Reset) olmak üzere iki girişi ve Q ve  $\bar{Q}$  olmak üzere iki çıkışı vardır ve geri besleme düzeneinde bağlı iki NOR kapısından oluşur.



Şekil 11.22 NOR Kapıları ile Gerçekleştirilen S-R Mandalı

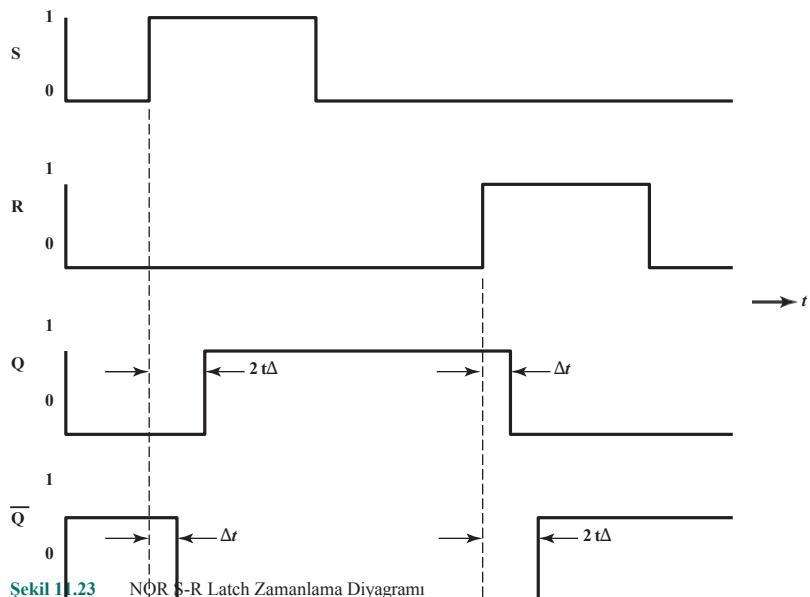
İlk olarak, devrenin çift kararlı olduğunu gösterelim. Alt NOR geçidinin girişleri  $Q=0$  ve  $S=0$ 'dır. Dolayısıyla,  $Q=1$  çıkışı, üst NOR geçidinin girişlerinin  $Q=1$  ve  $R=0$  veya  $Q=0$  çıkışına sahip olduğu anlamına gelir. Dolayısıyla, devrenin durumu içsel olarak tutarlıdır ve  $S=R=0$  olduğu sürece kararlı kalır. Benzer bir mantık yürütme,  $Q=1$ ,  $Q=0$   $R=S=0$  için de kararlı olduğunu gösterir.

Böylece, bu devre 1 bitlik bir bellek olarak işlev görebilir.  $Q$  çıkışını bitin "değeri" olarak görebiliriz.  $S$  ve  $R$  girişleri, sırasıyla 1 ve 0 değerlerini belleğe yazmaya yarar. Bunu görmek için  $Q=0$ ,  $Q=1$ ,  $S=0$ ,  $R=0$  durumunu göz önünde bulundurun.  $S$ 'nin 1 değerine değiştigini varsayıyalım. Şimdi alt NOR geçidinin girişleri  $S=1$ ,  $Q=0$ 'dır.  $\Delta t$  zaman gecikmesinden sonra, üst NOR geçidinin çıkışı  $Q=0$  (bkz. Şekil 11.23). Dolayısıyla, zamanın bu noktasında, üst NOR kapısının girişleri  $R=0$ ,  $Q=0$  olur.  $\Delta t$ 'lik bir başka kapı gecikmesinden sonra  $Q$  çıkışı 1 olur. Bu yine kararlı bir durumdur. Alt geçidin girişleri artık  $S=1$ ,  $Q=1$ 'dir ve  $Q=0$  çıkışını korur.  $S=1$  ve  $R=0$  olduğu sürece, çıkışlar  $Q=1$ ,  $Q=0$  olarak kalacaktır. Ayrıca,  $S$  0'a dönerse, çıkışlar değişmeden kalacaktır.

$R$  çıkışı tam tersi bir işlevi yerine getirir.  $R$  1'e gittiğinde,  $Q$  ve  $Q$ 'nun önceki durumundan bağımsız olarak  $Q=0$ ,  $Q=1$ 'e zorlar. Yine, son durum oluşturulmadan önce  $2\Delta t$ 'lik bir zaman gecikmesi meydana gelir (Şekil 11.23).

S-R mandali, *karakteristik tablo* adı verilen ve mevcut durumların ve girişlerin bir fonksiyonu olarak sıralı bir devrenin bir sonraki durumunu veya durumlarını gösteren doğruluk tablosuna benzer bir tablo ile tanımlanabilir. S-R mandali durumunda, durum  $Q$  değeri ile tanımlanabilir. Tablo 11.10a ortaya çıkan karakteristik tabloyu göstermektedir.  $S=1$ ,  $R=1$  girişlerine izin verilmeyğini gözlemleyin, çünkü bunlar tutarsız bir çıkışa neden olur (hem  $Q$  hem de  $Q$  0'a eşittir). Tablo, Tablo 11.10b'de olduğu gibi daha kompakt bir şekilde ifade edilebilir. S-R mandalının davranışının bir örneği Tablo 11.10c'de gösterilmiştir.

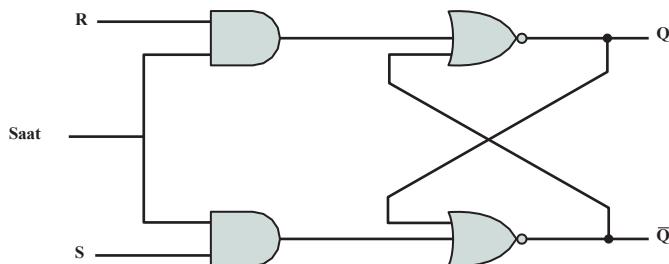
**CLOCKED S-R FLIP-FLOP** S-R mandalının çıkışı, girişteki bir değişikliğe yanıt olarak kısa bir zaman gecikmesinden sonra değişir. Buna asenkron çalışma denir. Daha tipik olarak, dijital bilgisayardaki olaylar bir saat darbesine senkronize edilir, böylece değişiklikler yalnızca bir saat darbesi meydana gerçekleşir. Şekil 11.24 bunu göstermektedir



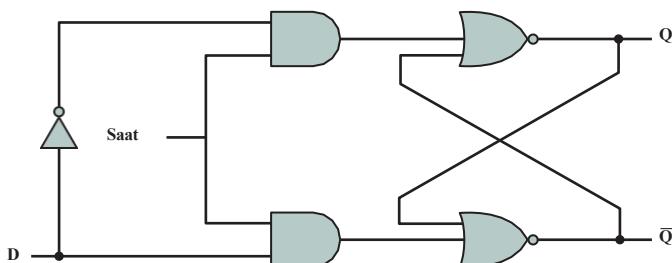
**Tablo 11.10** S-R Mandali

(a) Karakteristik Tablo			(b) Basitleştirilmiş Karakteristik Tablo		
Akum Girişleri	Mevcut Durum	Sonraki Durum	S	R	$Q_{n+1}$
SR	$Q_n$	$Q_{n+1}$	0	0	$Q_n$
00	0	0	0	1	0
00	1	1	1	0	1
01	0	0	1	1	-
01	1	0	0	0	
10	0	1	0	1	
10	1	1	1	0	
11	0	-	-	-	
11	1	-	-	-	

(c) Bir Dizi Girdiye Yanıt										
$t$	0	1	2	3	4	5	6	7	8	9
S	1	0	0	0	0	0	0	0	1	0
R	0	0	0	1	0	0	1	0	0	0
$Q_{n+1}$	1	1	1	0	0	0	0	0	1	1



Şekil 11.24 Saatli S-R Flip-Flop



Şekil 11.25 D Flip-Flop

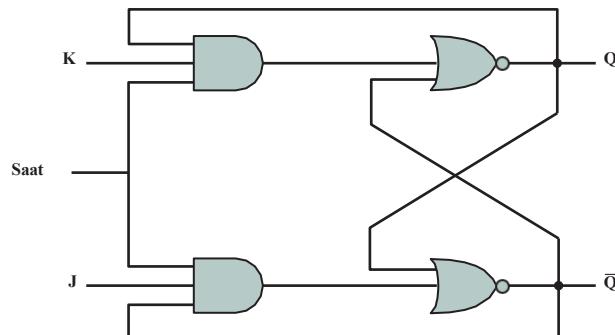
düzenleme. Bu cihaz **saatli S-R flip-flop** olarak adlandırılır. R ve S girişlerinin NOR kapılarına yalnızca saat darbesi sırasında iletildiğine dikkat edin.

**D FLIP-FLOP** S-R flip-flop ile ilgili bir sorun,  $R = 1$ ,  $S = 1$  koşulundan kaçınılması gerektidir. Bunu yapmanın bir yolu sadece tek bir girişe izin vermektedir. **D flip-flop** bunu başarır. Şekil 11.25'te D flip-flop'un bir kapı uygulaması gösterilmektedir. Bir invertör kullanılarak, iki AND kapısının saat dışı girişlerinin birbirinin tersi garanti edilir.

D flip-flop bazen veri flip-flop olarak da adlandırılır çünkü aslında bir bit veri için depolama alanıdır. D flip-flop'un çıkışı her zaman girişe uygulanan en son değere eşittir. Dolayısıyla, son giriş hatları ve üretir. Gecikmeli flip-flop olarak da adlandırılır, çünkü girişine uygulanan bir 0 veya 1'i tek bir saat darbesi için geciktirir. D flip-flop'un mantığını aşağıdaki doğruluk tablosunda yakalayabiliriz:

D	$Q_{n+1}$
0	0
1	1

**J-K FLIP-FLOP** Bir başka kullanışlı flip-flop da **J-K flip-flop**'tur. S-R flip-flop gibi bunun da iki giriş vardır. Ancak, bu durumda giriş değerlerinin tüm olası kombinasyonları geçerlidir. Şekil 11.26'da J-K flip-flop'un bir kapı uygulaması ve Şekil 11.27'de karakteristik tablosu (S-R ve D flip-flop'ları için olanlarla birlikte) gösterilmektedir. İlk üç kombinasyonun S-R flip-flop ile aynı olduğuna dikkat edin. Hiçbir giriş uygulanmadığında çıkış kararlıdır. Yalnızca J giriş'i onaylanırsa, sonuç bir settir

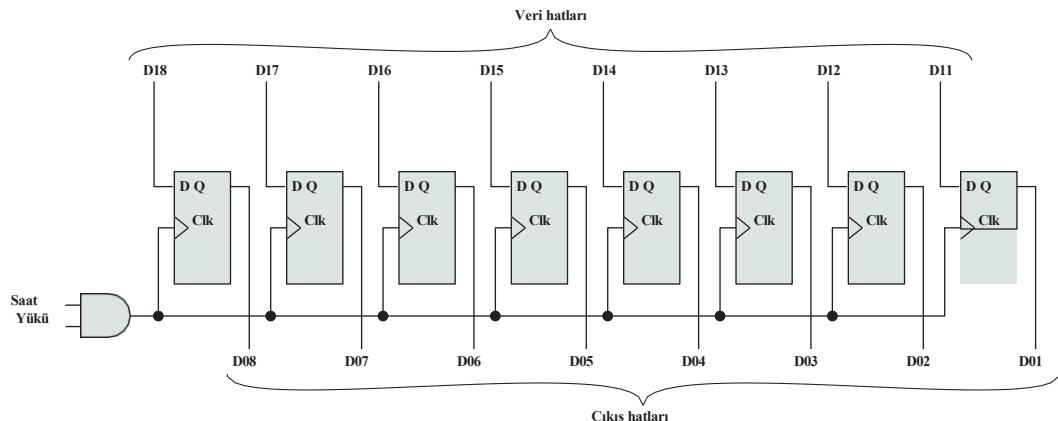


Şekil 11.26 J-K Flip-Flop

fonksiyonudur ve çıkışın 1 olmasına neden olur; yalnızca K girişi uygulanırsa, sonuç bir sıfırlama fonksiyonudur ve çıkışın 0 olmasına neden olur. Hem J hem de K 1 olduğunda, gerçekleştirilen fonksiyon geçiş olarak adlandırılır: çıkış tersine çevrilir. Böylece, Q 1 ise ve J ve K'ya 1 uygulanırsa, Q 0 olur. Okuyucu Şekil 11.26'daki uygulamanın bu karakteristik fonksiyonu ürettiğini doğrulamalıdır.

İsim	Grafik Sembol	Doğruluk Tablosu															
S-R		<table border="1"> <thead> <tr> <th>S</th><th>R</th><th><math>Q_{n+1}</math></th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td><math>Q_n</math></td></tr> <tr> <td>0</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>-</td></tr> </tbody> </table>	S	R	$Q_{n+1}$	0	0	$Q_n$	0	1	0	1	0	1	1	1	-
S	R	$Q_{n+1}$															
0	0	$Q_n$															
0	1	0															
1	0	1															
1	1	-															
J-K		<table border="1"> <thead> <tr> <th>J</th><th>K</th><th><math>Q_{n+1}</math></th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td><math>Q_n</math></td></tr> <tr> <td>0</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td><math>\overline{Q_n}</math></td></tr> </tbody> </table>	J	K	$Q_{n+1}$	0	0	$Q_n$	0	1	0	1	0	1	1	1	$\overline{Q_n}$
J	K	$Q_{n+1}$															
0	0	$Q_n$															
0	1	0															
1	0	1															
1	1	$\overline{Q_n}$															
D		<table border="1"> <thead> <tr> <th>D</th><th><math>Q_{n+1}</math></th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td></tr> <tr> <td>1</td><td>1</td></tr> </tbody> </table>	D	$Q_{n+1}$	0	0	1	1									
D	$Q_{n+1}$																
0	0																
1	1																

Şekil 11.27 Temel Flip-Floplar



**Şekil 11.28** 8-Bit Paralel Yazmaç

### Kayıtlar

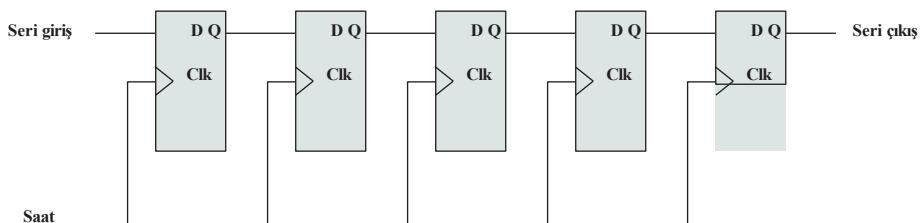
Flip-flopların kullanımına bir örnek olarak, ilk önce CPU temel unsurlarından birini inceleyelim: register. Bildiğimiz gibi yazmaç, bir veya daha fazla veri bitini saklamak için CPU içinde kullanılan dijital bir devredir. Yaygın olarak iki temel yazmaç türü kullanılır: paralel yazmaçlar ve kaydirmalı yazmaçlar.

**PARALEL KAYITLAR** Bir **paralel kayıt**, aynı anda okunabilen veya yazılabilen bir dizi 1 bitlik bellekten oluşur. Veri depolamak için kullanılır. Bu kitap boyunca ele aldığımız kaydediciler paralel kaydedicilerdir.

Şekil 11.28'deki 8 bitlik yazmaç, D flip-flopları kullanan bir paralel yazmaçın çalışmasını göstermektedir. *Yük* etiketli bir kontrol sinyali, D11'den D18'e kadar olan sinyal hatlarından yazmaçın içine yazmayı kontrol eder. Bu hatlar çoklayıcıların çıkışını olabilir, böylece çeşitli kaynaklardan gelen veriler yazmaca yüklenebilir.

**SHIFT REGISTER** Bir **shift register** bilgiyi seri olarak kabul eder ve/veya aktarır. Örneğin, saatlı D flip-foplardan oluşturulmuş 5 bitlik bir kaydırma kaydedicisini gösteren Şekil 11.29'u düşünün. Veriler yalnızca en soldaki flip-flop'a girilir. Her saat darbesinde, veriler bir konum sağa kaydırılır ve en sağdaki bit dışarı aktarılır.

Kaydırma kayıtları seri I/O cihazlarına arayüz oluşturmak için kullanılabilir. Ayrıca, mantıksal kaydırma ve döndürme işlevlerini gerçekleştirmek için ALU içinde kullanılabilirler. Bunun içinde



**Şekil 11.29** 5-Bit Kaydirmalı Yazmaç

ikinci kapasite, seri olduğu kadar paralel okuma/yazma devresiyle de donatılmaları gereklidir.

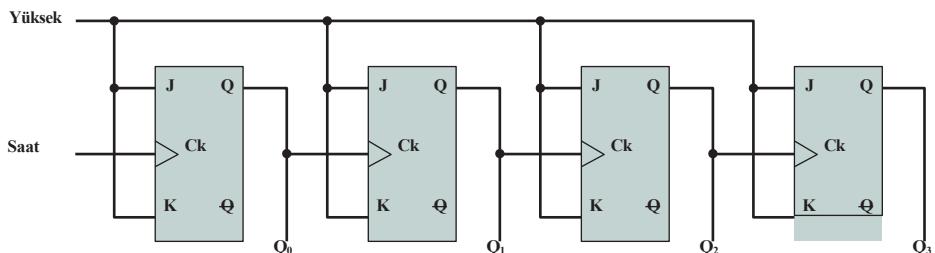
### Sayaçlar

Bir diğer kullanışlı sıralı devre kategorisi de **sayaçtır**. Bir sayaç, değeri yazmacın kapasitesinin modülüyle kolayca 1 artırılan bir yazmacıdır; , maksimum değere ulaşıldıktan sonra bir sonraki artış sayaç değerini

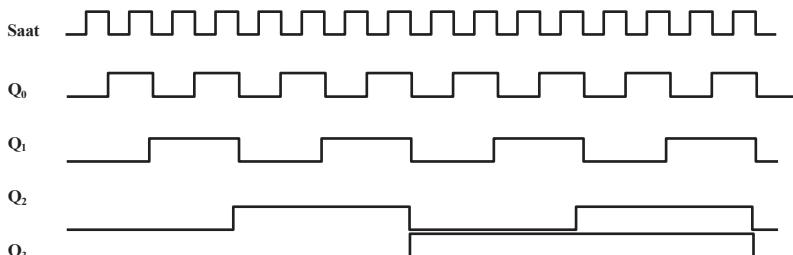
0. Böylece,  $n$  flip-floptan oluşan bir register  $2^n - 1$ 'e kadar sayabilir. CPU'daki bir sayaç örneği program sayacıdır.

Sayaçlar, çalışma şekillerine bağlı olarak asenkron veya senkron olarak tanımlanabilir. Asenkron sayaçlar nispeten yavaştır çünkü bir flip-flop'un çıkışı bir sonraki flip-flop'un durumundaki değişikliği tetikler. **Senkron bir sayıda**, tüm flip-floplar aynı anda durum değiştirir. İkinci tür çok daha hızlı olduğu için CPU'larda kullanılanlardır. Ancak, tartışmaya asenkron sayıçının bir tanımıyla başlamak yararlı olacaktır.

**DALGALI SAYICI** Asenkron bir sayıçuya **dalgalı sayıçı** da denir, çünkü sayıçayı artırmak için meydana gelen değişiklik bir uçtan başlar ve diğer uca doğru "dalgalanır". Şekil 11.30'da J-K flip-flopları kullanan 4 bitlik sayıci uygulaması ve davranışını gösteren bir zamanlama diyagramı gösterilmektedir. Zamanlama diyagramı, sinyaller flip-flop serisinde aşağı doğru hareket ederken meydana gelen yayılma gecikmesini gösterdiği için idealleştirilmiştir. En soldaki flip-flopun çıkışı ( $Q_0$ ) en az anlamlı比特dir. Tasarım, daha fazla flip-flop basamaklandırılarak açıkça rastgele sayıda bite genişletilebilir.



(a) Sırah devre



(b) Zamanlama şeması

Şekil 11.30 Dalgalanma Sayacı

Gösterilen uygulamada, sayaç her saat darbesinde artırılır. Her bir flip-flop'un J ve K girişleri sabit 1'de tutulur. Bu, bir saat darbesi olduğunda Q çıkışının ters çevrileceği anlamına (1'den 0'a; 0'dan 1'e). Durum değişikliğinin saat darbesinin düşen kenarı ile gerçekleştiğinin gösterildiğine dikkat edin; bu kenar tetiklemeli flip-flop olarak bilinir. Darbenin kendisi yerine bir saat darbesindeki geçişe yanıt veren flip-flopların kullanılması karmaşık devrelerde daha iyi zamanlama kontrolü sağlar. Bu sayıçı için çıkış modellerine bakılırsa, 0000, 0001, ..., 1110, 1111, 0000 ve şekilde döndüğü görülebilir.

**SENKRON SAYICILAR** Dalgalı sayıçı, sayıcının uzunluğu ile orantılı olan değer değişimindeki gecikme dezavantajına sahiptir. Bu dezavantajın üstesinden gelmek için CPU'lar, sayıcının tüm flip-floplarının aynı anda değiştiği senkron sayıcılarından yararlanır. Bu alt bölümde, 3 bitlik bir senkron sayıçı için bir tasarım sunuyoruz. Bunu yaparken, senkron bir devrenin tasarımindaki bazı temel kavramları göstereceğiz.

3 bitlik bir sayıçı için üç flip-flop gerekecektir. J-K flip-floplarını kullanalım. Üç flip-flopun tamamlanmamış çıkışlarını sırasıyla C, B ve A olarak etiketleyin, C en anlamlı biti temsil eder. İlk adım, genel devreyi tasarlayabilmemiz için J-K giriş ve çıkışlarını ilişkilendiren bir doğruluk tablosu oluşturmaktır. Böyle bir doğruluk tablosu Şekil 11.31a'da gösterilmektedir. İlk üç sütun C, B ve A çıkışlarının olası kombinasyonlarını göstermektedir. Bunlar sayaç artırıldıkça görünecekleri sırayla listelenmiştir. Her satır C, B ve A'nın mevcut değerini ve C, B ve A'nın bir sonraki değerine ulaşmak için gerekli olan üç flip-flopun girişlerini listeler.

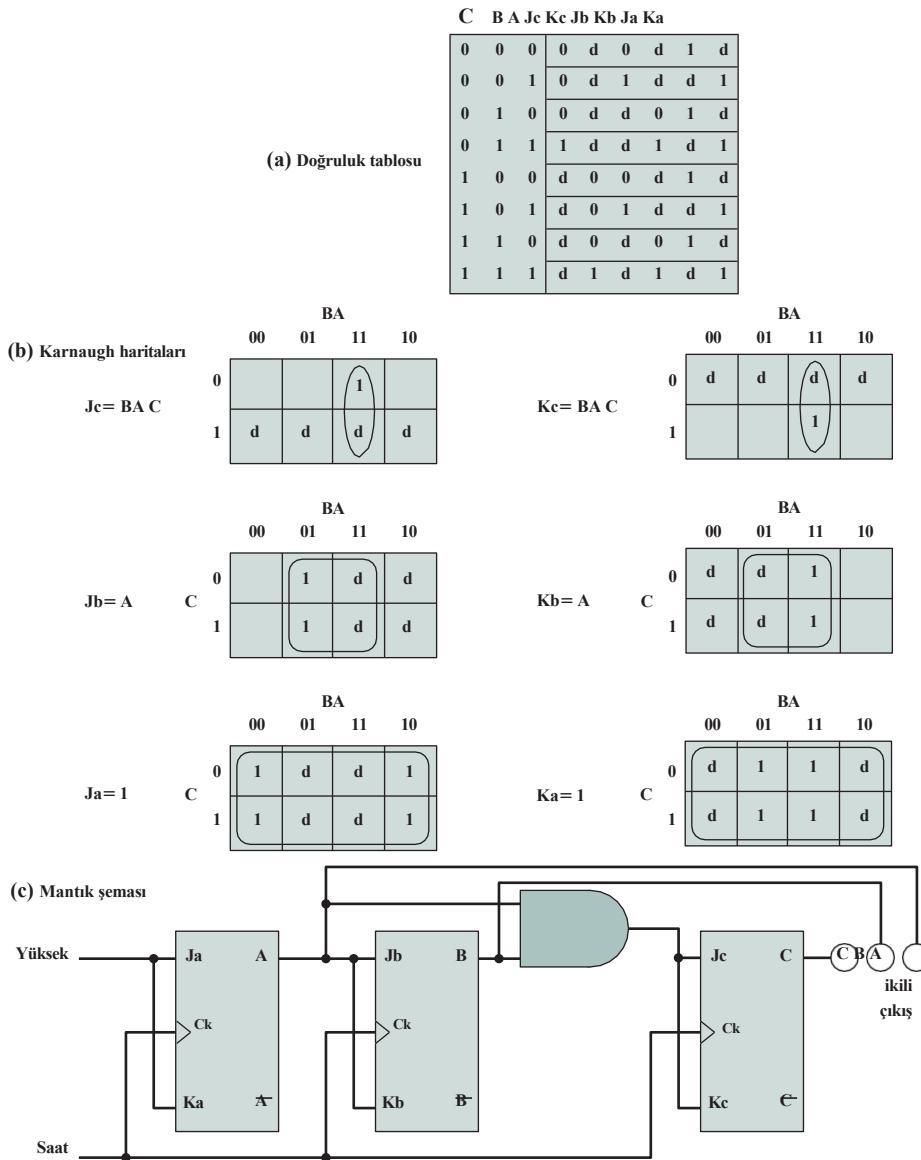
Şekil 11.31a'daki doğruluk tablosunun nasıl oluşturulduğunu anlamak için, J-K flip-flop için karakteristik tablosunu yeniden düzenlemek yararlı olabilir. Bu tablonun aşağıdaki gibi sunulduğunu hatırlayın:

J	K	$Q_{n+1}$
0	0	$Q_n$
0	1	0
1	0	1
1	1	$\overline{Q_{n+1}}$

Bu haliyle tablo, J ve K girdilerinin çıktı üzerindeki etkisini göstermektedir. Şimdi aynı bilginin aşağıdaki organizasyonunu düşünün:

$Q_n$	J	K	$Q_{n+1}$
0	0	d	0
0	1	d	1
1	d	1	0
1	d	0	1

Bu haliyle tablo, girişler ve mevcut çıkış bilindiğinde bir sonraki çıkışın değerini verir. Bu tam sayıçı veya herhangi bir sıralı devreyi tasarlamak için gereken bilgidir. Bu formda, tablo bir **uyarma tablosu** olarak adlandırılır.



Şekil 11.31 Bir Senkron Sayıcının Tasarımı

Şekil 11.31'a'ya geri dönelim. İlk satırı düşün. C değerinin 0 olarak kalmasını, B değerinin 0 olarak kalmasını ve A değerinin bir sonraki saat darbesi uygulamasında 0'dan 1'e geçmesini istiyoruz. Uyarma tablosu, 0 çıkışını korumak için  $J=0$  girişlerine sahip olmamız gerektiğini ve  $K$ 'yı önemsememiizi göstermektedir. 0'dan 1'e geçişini gerçekleştirmek için girişler  $J=1$  ve  $K=d$  olmalıdır. Bu değerler tablonun ilk satırında gösterilmektedir. Benzer bir mantıkla, tablonun geri kalanı doldurulabilir.

Şekil 11.31a'daki doğruluk tablosunu oluşturduktan sonra, tablonun C, B ve A'nın mevcut değerlerinin fonksiyonları olarak tüm J ve K girişlerinin gerekli değerlerini gösterdiğini görüyoruz. Karnaugh haritalarının yardımıyla, bu altı fonksiyon için Boolean ifadeleri geliştirebiliriz. Bu, şeklärin b bölümünde gösterilmektedir. Örneğin, Ja değişkeni için Karnaugh haritası (A çıkışını üreten flip-flop'un J girişi)  $Ja = BC$  ifadesini verir. Altı ifadenin tümü türetildiğinde, şeklärin c bölümünde gösterildiği gibi gerçek devreyi tasarlamak basit bir meseledir.

## 11.5 PROGRAMLANABİLİR MANTIK CİHAZLARI

Şimdiye kadar, tek tek kapıları keyfi işlevlerin gerçekleştirilebileceği yapı taşıları olarak ele aldık. Tasarımcı, ilgili Boolean ifadelerini manipüle ederek kullanılacak kapı sayısını en azı indirme stratejisini izleyebilir. Entegre devreler tarafından sağlanan entegrasyon seviyesi arttıkça, başka hususlar da geçerli olmaktadır. Küçük ölçekli entegrasyon (SSI) kullanan ilk entegre devreler, bir çip üzerinde bir ila on kapı sağlıyordu. Şimdiye kadar açıklanan yapı taşı yaklaşımında her kapı bağımsız olarak ele alınmaktadır. Bir mantık fonksiyonu oluşturmak için, bu çiplerin bir kısmı bir baskılı devre kartı üzerine yerleştirilir ve uygun pin ara bağlantıları yapılır.

Artan entegrasyon seviyeleri, bir çip üzerine daha fazla kapı koymayı ve çip üzerinde de kapı ara bağlantıları yapmayı mümkün kılmıştır. Bu da daha düşük maliyet, daha düşük boyut ve daha yüksek hız (çip üzerindeki gecikmeler çip dışı gecikmelerden daha kısa süreli için) gibi avantajlar sağlamlamaktadır. Ancak bir tasarım sorunu ortaya çıkmaktadır. Her bir mantık işlevi ya da işlev grubu için çip üzerindeki kapıların ve bağlantılarının yerlesimi tasarlanmalıdır. Bu tür bir özel çip tasarımının maliyeti ve süresi yüksektir. Bu nedenle, belirli amaçlara kolayca uygulanabilen genel amaçlı bir çip geliştirmek cazip hale gelmektedir. *Programlanabilir mantık cihazının* (PLD) amacı budur.

Ticari kullanımında bir dizi farklı PLD türü vardır. Tablo 11.11 bazı temel terimleri listelemekte ve en önemli türlerden bazılarını tanımlamaktadır. Bu bölümde, ilk olarak bu tür cihazların en basitlerinden biri olan *programlanabilir mantık dizisine* (PLA) bakacağız ve ardından belki de en önemli ve yaygın olarak kullanılan PLD türü olan sahada programlanabilir kapı dizisini (FPGA) tanıtabağımız.

### Programlanabilir Mantık Dizisi

PLA, gördüğümüz gibi herhangi bir Boole fonksiyonunun (doğruluk tablosu) çarpımların toplamı (SOP) şeklinde ifade edilebileceği gerçekine dayanmaktadır. PLA, bir çip üzerindeki NOT, AND ve OR kapılarının düzenli bir şekilde düzenlenmesinden oluşur. Her çip girişi bir DEĞİL kapısından geçirilir, böylece her giriş ve tamamlayıcısı her VE için kullanılabilir. Her VE kapısının çıkışı her VEYA kapısı için kullanılabilir ve her VEYA kapısının çıkışı bir çip çıkışıdır. Uygun bağlantılar yapılarak keyfi SOP ifadeleri uygulanabilir.

Şekil 11.32a'da üç girişi, sekiz kapısı ve iki çıkışı olan bir PLA gösterilmektedir. Sol tarafta programlanabilir bir AND dizisi bulunmaktadır. AND dizisi, herhangi bir PLA girişi veya olumsuzlaması ile herhangi bir AND kapısı girişi arasında, kesişme noktalarında karşılık gelen çizgileri birleştirerek bir bağlantı kurarak programlanır. Üzerinde

**Tablo 11.11** PLD Terminolojisi

<b>Programlanabilir Mantık Aygıtı (PLD)</b>
Çipin son kullanıcı tarafından farklı tasarımları gerçekleştirmek üzere yapılandırılabildiği, dijital donanımları uygulamak için kullanılan her tür entegre devreyi ifade eden genel bir terimdir. Böyle bir cihazın programlanması genellikle çipin özel bir programlama ünitesine yerleştirilmesini içerir, ancak bazı çipler "sistem içinde" de yapılandırılabılır. Sahada programlanabilir cihaz (FPGD) olarak da adlandırılır.
<b>Programlanabilir Mantık Dizisi (PLA)</b>
Her iki seviyeden de programlanabilir olduğu iki mantık seviyesi, bir VE düzlemi ve bir VEYA düzlemi içeren nispeten küçük bir PLD.
<b>Programlanabilir Dizi Mantığı (PAL)</b>
Programlanabilir bir VE düzlemine ve ardından sabit bir VEYA düzlemine sahip nispeten küçük bir PLD.
<b>Basit PLD (SPLD)</b>
Bir PLA veya PAL.
<b>Karmaşık PLD (CPLD)</b>
Birden fazla SPLD benzeri bloğun tek bir blok üzerinde düzenlenmesinden oluşan daha karmaşık bir PLD Çip.
<b>Alan Programlanabilir Kapı Dizisi (FPGA)</b>
Cok yüksek mantık kapasitesine izin veren genel bir yapıya sahip bir PLD. CPLD'ler çok sayıda girişe (AND düzlemleri) sahip mantık kaynaklarına sahipken, FPGA'lar daha dar mantık kaynakları sunar. FPGA'lar ayrıca CPLD'lere kıyasla daha yüksek oranda flip-flop/mantık kaynağı sunar.
<b>Mantık Bloğu</b>
Bir FPD'deki bir dizede çoğaltılan nispeten küçük bir devre bloğu. Bir devre bir FPD'de uygulandığında, ilk olarak her biri bir mantık bloğuna eşlenebilen daha küçük alt devrelerle ayrıstırılır. <i>Mantık bloğu</i> terimi çoğunlukla FPGA'lar bağlamında kullanılır, ancak bir CPLD'deki bir devre bloğunu da edebilir.

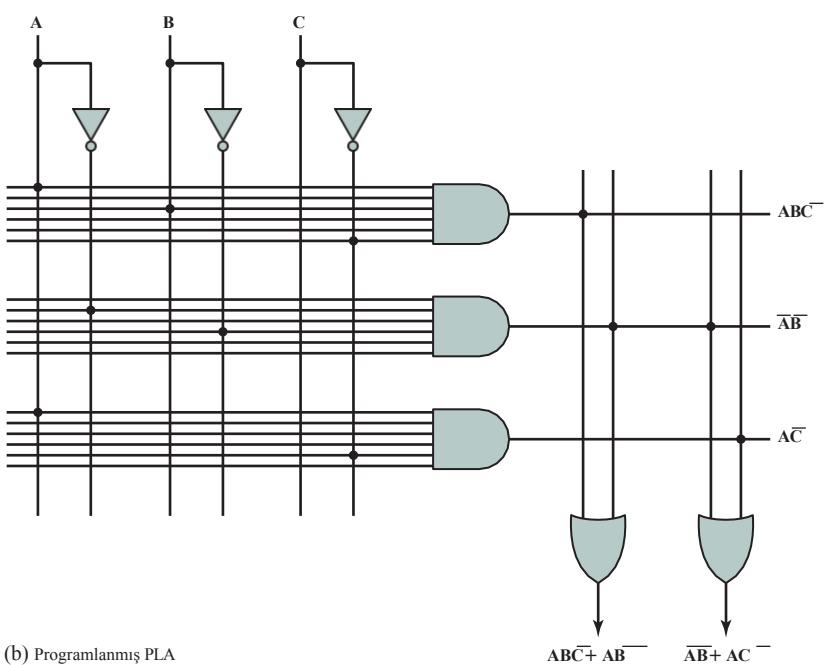
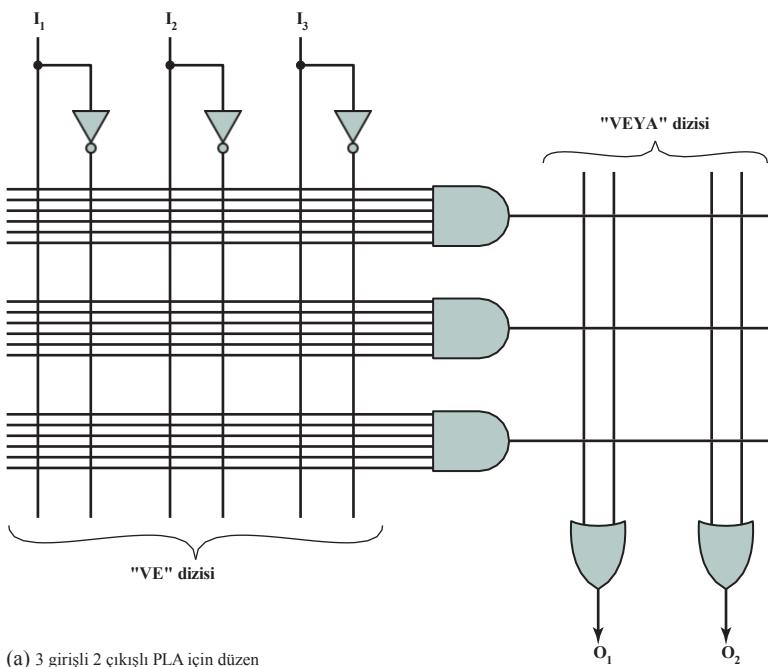
doğru programlanabilir bir VEYA dizisidir ve VE kapısı çıkışlarının VEYA kapısı girişlerine bağlanması içerir. Coğu büyük PLA birkaç yüz kapı, 15 ila 25 giriş ve 5 ila 15 çıkış içerir. Girişlerden VE kapılarına ve VE kapılarından VEYA kapılarına bağlantılar programlama zamanına kadar belirtilmemez.

PLA'lar kolay programlama (bağlantı kurma) için iki farklı şekilde üretilmektedir. İlkinde, her olası bağlantı her kesişme noktasında bir sigorta aracılığıyla yapılır. İstenmeyen bağlantılar daha sonra sigortaların attırılmasıyla ortadan kaldırılabilir. Bu tür PLA, *sahada programlanabilir mantık dizisi (FPLA)* olarak adlandırılır. Alternatif olarak, uygun bağlantılar çip üretimi sırasında belirli bir ara bağlantı için sağlanan uygun bir maske kullanılarak yapılabilir. Her iki durumda da PLA, dijital mantık fonksiyonlarını uygulamak için esnek ve ucuz bir yol sağlar.

Şekil 11.32b, iki Boolean ifadesini gerçekleştiren programlanmış bir PLA'yı göstermektedir.

### Alan Programlanabilir Kapı Dizisi

PLA basit bir PLD (SPLD) örneğidir. Katı bir SPLD mimarisinin kapasitesini arturanın zorluğu, programlanabilir mantık düzlemlerinin yapısının, giriş sayısı arttıkça boyut olarak çok hızlı büyümeleridir. SPLD mimarilerine dayalı büyük kapasiteli cihazlar sağlamamanın tek uygulanabilir yolu, birden fazla SPLD'yi tek bir çip üzerine entegre etmek ve SPLD bloklarını programatik olarak bağlamak için ara bağlantı sağlamaktır. Birçok ticari PLD ürünü



**Şekil 11.32** Programlanabilir Mantık Dizisi (PLA) Örneği

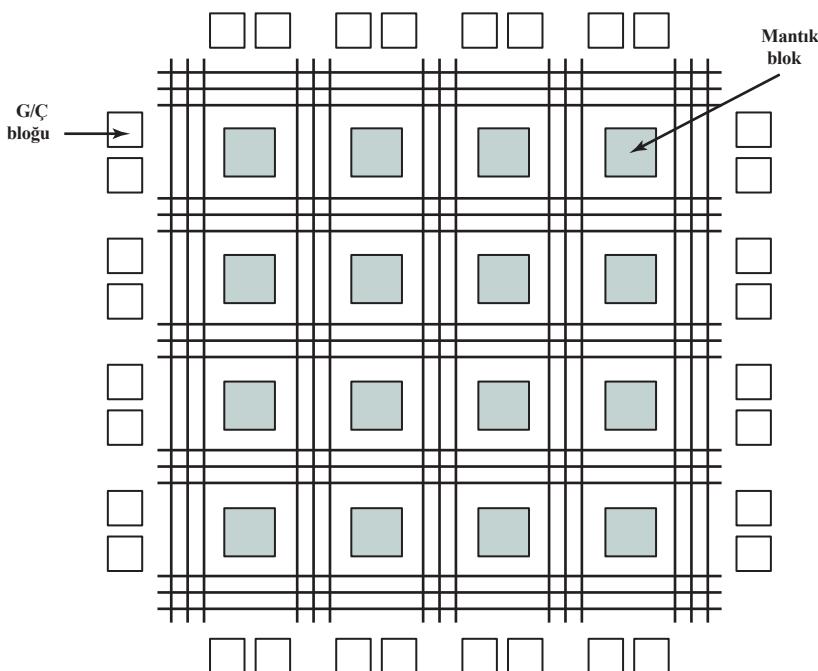
Bugün piyasada bu temel yapıya sahip ve toplu olarak Karmaşık PLD'ler (CPLD'ler) olarak adlandırılan cihazlar mevcuttur. En önemli CPLD türü FPGA'dır.

Bir FPGA, **mantık blokları** olarak adlandırılan bir dizi işlenmemiş devre elemanından ve ara bağlantı kaynaklarından oluşur. Tipik bir FPGA mimarisinin bir örneği Şekil 11.33'te gösterilmektedir. Bir FPGA'nın temel bileşenleri şunlardır;

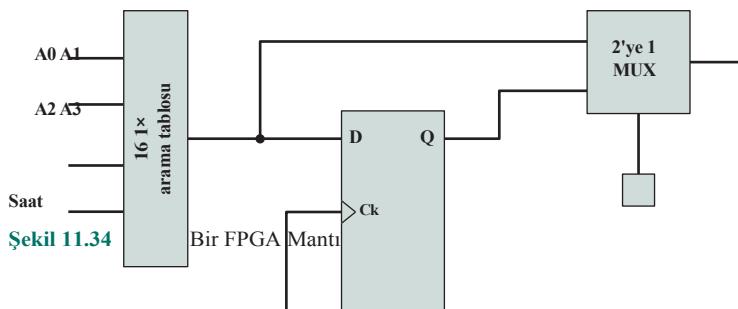
- **Mantık bloğu:** Yapılandırılabilir mantık blokları, kullanıcının devresinin hesaplamasının gerçekleştiği yerdir.
- **G/C bloğu:** G/C blokları G/C pinlerini çip üzerindeki devrelere bağlar.
- **Ara bağlantı:** Bunlar, G/C blokları ve mantık blokları arasında bağlantı kurmak için kullanılabilen sinyal yollarıdır.

Mantık bloğu kombinasyonel bir devre ya da sıralı bir devre olabilir. Temelde, bir mantık bloğunun programlanması, bir mantık işlevi için bir doğruluk tablosunun içeriğinin indirilmesiyle yapılır. Şekil 11.34'te bir D flip-flop, 2'den 1'e çoklayıcı ve 16 bitlik bir **arama tablosundan** oluşan basit bir mantık bloğu örneği gösterilmektedir. Arama tablosu 16 adet 1 bitlik elemandan oluşan bir bellektir, böylece 16 bitten birini seçmek için 4 giriş hattı gereklidir. Daha büyük mantık blokları daha büyük arama ve birbirine bağlı birden fazla arama tablosuna sahiptir. Arama tablosu tarafından gerçekleştirilen kombinasyonel mantık doğrudan çakılabilir veya D flip-flop'ta saklanabilir ve eşzamanlı olarak çakılabilir. Ayrı bir tek bitlik bellek, çıkışın doğrudan arama tablosundan mı yoksa flip-floptan mı geldiğini belirlemek için çoklayıcıyı kontrol eder.

Çok sayıda mantık bloğunun birbirine bağlanmasıyla, çok karmaşık mantık işlevleri kolayca uygulanabilir.



**Şekil 11.33** Bir FPGA'nın Yapısı



## 11.6 ANAHTAR TERİMLER VE SORUNLAR

### Anahtar Terimler

topluyıcı VE kapısı assert Boole cebiri saatli S-R flip-flop D flip-flop kapılar grafik simbol J-K flip-flop Karnaugh haritası mantık bloğu arama tablosu çoklayıcı NAND kapısı NOR	VEYA kapısı paralel kayıt kombinasyonel devre kompleks PLD (CPLD) sayacı kod çözücü toplamların çarpımı (POS) programlanabilir dizi mantığı (PAL) programlanabilir mantık dizisi (PLA) programlanabilir mantık cihazı (PLD) Quine-McCluskey yöntemi salt okunur bellek (ROM)	uyarma tablosunu kaydet sahada programlanabilir kapı dizisi (FPGA) flip-flop dalgalanma sayacı sıralı devre kaydırmalı kaydedici basit PLD (SPLD) çarpımların toplamı (SOP) senkron sayıcı S-R Latch doğruluk tablosu XOR geçidi
---	--	---

### Problemler

- 11.1** Aşağıdaki Boole ifadeleri için bir doğruluk tablosu oluşturun:
- $ABC + A \underline{B} \underline{C}$
  - $ABC + \underline{A} \underline{B} C + A \underline{B} C$
  - $A(M\bar{O} + M\bar{O})$
  - $(A + B)(A + C)(A + B)$
- 11.2** Aşağıdaki ifadeleri değişmeli yasaya göre sadeleştiriniz:
- $A \quad B + B \quad A + C \quad D \quad E + C \quad D \quad E + E \quad C \quad D$   
 $A \quad B + \quad A \quad C + B \quad A$
  - $(L \cdot M \cdot N)(A \cdot B)(C \cdot D) \cdot E \cdot (M \cdot N \cdot L)$
  - $F \cdot (K + R) + S \cdot V + W \cdot X + V \cdot S + X \cdot W + (R + K) \cdot F$

**11.3** DeMorgan teoremini aşağıdaki denklemlere uygulayın:

- a.  $F = \overline{V + A + L}$
- b.  $F = \overline{A + B + C + D}$

**11.4** Aşağıdaki ifadeleri sadeleştirin:

- a.  $A = S \cdot T + V \cdot W + R \cdot S \cdot T$   
 $A = T \cdot U + V \cdot X + Y \cdot Y$
- b.  $A = F \cdot (E + F + G) Q +$
- c.  $A = (P \cdot R + S \cdot D + E \cdot T) T \cdot S$   
 $A = D$
- d.  $A = Y \cdot (W + X + Y + Z) \cdot Z$
- e.  $A = (B \cdot E + C + F) \cdot C$

**11.5** Temel Boolean işlemleri AND, OR ve NOT'tan XOR işlemini oluşturun.

**11.6** Bir NOR kapısı ve NOT kapıları verildiğinde, üç girişli AND işlevini gerçekleştirecek bir mantık diyagramı çizin.

**11.7** Dört girişli bir **NAND kapısı** için Boolean ifadesini yazınız.

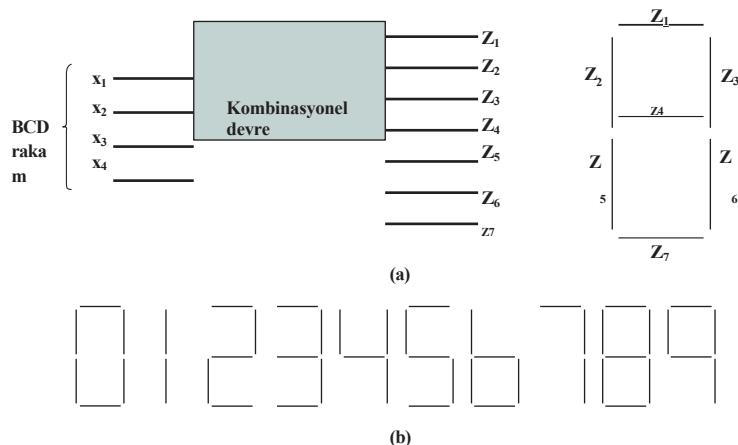
**11.8** Şekil 11.35'te gösterildiği gibi, ondalık basamaklardan oluşan yedi segmentli bir ekranı kontrol etmek için bir kombinasyonel devre kullanılır. Devrenin, paketlenmiş ondalık gösterimde kullanılan dört bitlik kodu sağlayan dört girişi vardır ( $0_{10} = 0000$ ,  $C$ ,  $9_{10} = 1001$ ). Yedi çıkış, belirli bir ondalık basamağı görüntülemek için hangi segmentlerin etkinleştirileceğini tanımlar. Bazı giriş ve çıkış kombinasyonlarına gerek olmadığını unutmayın.

- a. Bu devre için bir doğruluk tablosu geliştirin.
- b. Doğruluk tablosunu SOP formunda ifade edin.
- c. Doğruluk tablosunu POS formunda ifade edin.
- d. Basitleştirilmiş bir ifade sağlayınız.

**11.9** 8'e 1 çoklayıcı tasarlayın.

**11.10** Şekil 11.15'e ek bir hat ekleyin, böylece bir demultiplexer olarak işlev görür.

**11.11** Gray kodu tam sayılar için ikili bir koddur. Herhangi iki sayının gösterimi arasında sadece tek bir bit değişikliği olması nedeniyle sıradan ikili gösterimden farklıdır. Bu, bir dizi sayının üretildiği sayıclar veya analog-dijital dönüştürücüler gibi uygulamalar için kullanılmıştır. Her seferinde yalnızca bir bit değiştiği için, küçük zamanlama farklılıklarından kaynaklanan herhangi bir belirsizlik asla söz konusu değildir. Kodun ilk sekiz ögesi şunlardır



**Şekil 11.35** Yedi Segmentli LED Ekran Örneği

İkili Kod	Gri Kod
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

İkili koddan Gri koda dönüştüren bir devre tasarlın.

- 11.12** Dört adet  $3 * 8$  kod çözücü (etkinleştirme girişleri ile) ve bir adet  $2 * 4$  kod çözücü kullanarak  $5 * 32$  kod çözücü tasarlın.
- 11.13** Şekil 11.20'deki tam toplayıcıyı sadece beş kapı ile gerçekleştirin. (*İpucu:* Kapılardan bazıları XOR kapılarıdır.)
- 11.14** Şekil 11.20'yi göz önünde bulundurun. Her bir kapının 10 ns'lik bir gecikme ürettiğini varsayıñ. Böylece, toplam çıkış 20 ns sonra ve taşıma çıkış 20 ns sonra geçerli olur. 32-bit toplayıcı için toplam toplama süresi nedir?
- Şekil 11.19'daki gibi carry lookahead olmadan uygulandı mı?
  - Şekil 11.21'deki gibi carry lookahead ile ve 8 bitlik toplayıcılar kullanılarak mı uygulanır?
- 11.15** S-R mandalının alternatif bir biçimini Şekil 11.22 ile aynı yapıya sahiptir ancak NOR kapıları yerine NAND kapıları kullanır.
- NAND kapıları ile uygulanan S-R mandalı için Tablo 11.10a ve 11.10b'yi tekrarlayın.
  - Tablo 11.10c'ye benzer şekilde aşağıdaki tabloyu doldurun.

<i>t</i>	0	1	2	3	4	5	6	7	8	9
S	0	1	1	1	1	1	0	1	0	1
R	1	1	0	1	0	1	1	1	0	0

- 11.16** Şekil 11.27'deki S-R flip-flop için grafik simbolünü düşünün. S-R flip-flop'tan bağlanan bir D flip-flop'u göstermek için ek çizgiler ekleyin.

- 11.17** Üç girişi (C, B, A) ve dört çıkıştı ( $O_0$ ,  $O_1$ ,  $O_2$ ,  $O_3$ ) olan ve çıkışları aşağıdaki gibi tanımlanan bir PLA'nın yapısını gösteriniz:

$$\begin{aligned} O_0 &= A \overline{B} \overline{C} + \overline{A} \overline{B} + \overline{A} \overline{B} C \quad O_1 = A \overline{C} \\ &\quad \overline{B} C + A B C \\ O_2 &= C \\ O_3 &= \overline{A} \overline{B} + A B C \end{aligned}$$

- 11.18** PLA'nın ilginç bir uygulaması da eski, modası geçmiş delikli kart karakter kodlarının ASCII kodlarına dönüştürülmesidir. Geçmişte bilgisayarlarda çok popüler olan standart delikli kartlarda deliklerin açılabildiği 12 satır ve 80 sütun vardı. Her sütun bir karaktere karşılık geliyordu, dolayısıyla her karakterin 12 bitlik bir kodu vardı. Ancak, gerçekte yalnızca 96 karakter kullanılıyordu. Delikli kartları okuyan ve karakter kodlarını ASCII'ye dönüştüren bir uygulama düşünün.

- Bu uygulamanın bir PLA uygulamasını açıklayın.
- Bu sorun bir ROM ile çözülebilir mi? Açıkla.

# BÖLÜM 12

## KOMUT KÜMELERİ: ÖZELLİKLER VE İŞLEVLER

- 12.1 Makine Özellikleri** Makine Talimatının  
Unsurları Talimat Gösterimi Talimat  
Türleri  
Adres Sayısı Komut  
Kümesi Tasarımı

- 12.2 Operand Türleri**  
Sayılar Karakterler  
Mantıksal Veriler

- 12.3 Intel x86 ve ARM Veri Türleri**  
x86 Veri Türleri ARM  
Veri Türleri

- 12.4 Operasyon Türleri**  
Veri Aktarımı  
Aritmetik  
Mantıksal  
Dönüşüm  
Giriş/Çıkış Sistem  
Kontrolü  
Kontrolün Devri

- 12.5 Intel x86 ve ARM İşlem Türleri**  
x86 İşlem Türleri ARM  
İşlem Türleri

- 12.6 Anahtar Terimler, Gözden Geçirme Soruları ve  
Problemler Ek 12A Küçük, Büyük ve İki Kızılderili**

### ÖĞRENİM HEDEFLERİ

Bu bölümü çalıştıktan sonra şunları yapabilmelisiniz:

- **Makine komutlarının** temel özelliklerine genel bir bakış sunmak. ■ Tipik makine komut setlerinde kullanılan operand türlerini tanımlamak. ■ x86 ve ARM veri türlerine genel bir bakış sunmak.
- Tipik makine komut setleri tarafından desteklenen operand türlerini tanımlamak.
- x86 ve ARM işlem türlerine genel bir bakış sunar.
- **Büyük endian, küçük endian ve bi-endian** arasındaki farkları anlamak.

Bu kitapta tartışılanların çoğu, bir bilgisayar kullanıcısı ya da programcısı tarafından kolayca görülemez. Eğer bir programcı Pascal ya da Ada gibi yüksek seviyeli bir dil kullanıyorsa, alatta yatan makinenin mimarisinin çok azı görülebilir.

Bilgisayar tasarımcısı ve bilgisayar programcısının aynı makineyi görebildiği sınırlardan biri makine komut setidir. Tasarımcının bakış açısından, makine komut seti işlemci için işlevsel gereksinimleri sağlar: işlemcinin uygulanması büyük ölçüde makine komut setinin uygulanmasını içeren bir görevdir. Makine dilinde (aslında assembly ; bkz. Ek B) programlamayı seçen kullanıcı, yazmaç ve bellek yapısı, makine tarafından doğrudan desteklenen veri türleri ve ALU'nun işleyişi bilgi sahibi olur.

Bir bilgisayarin makine komut setinin tanımı, bilgisayarin işlemcisini açıklamak için uzun bir yol kat eder. Buna göre, bu bölümde ve bir sonraki bölümde makine komutlarına odaklanacağız.

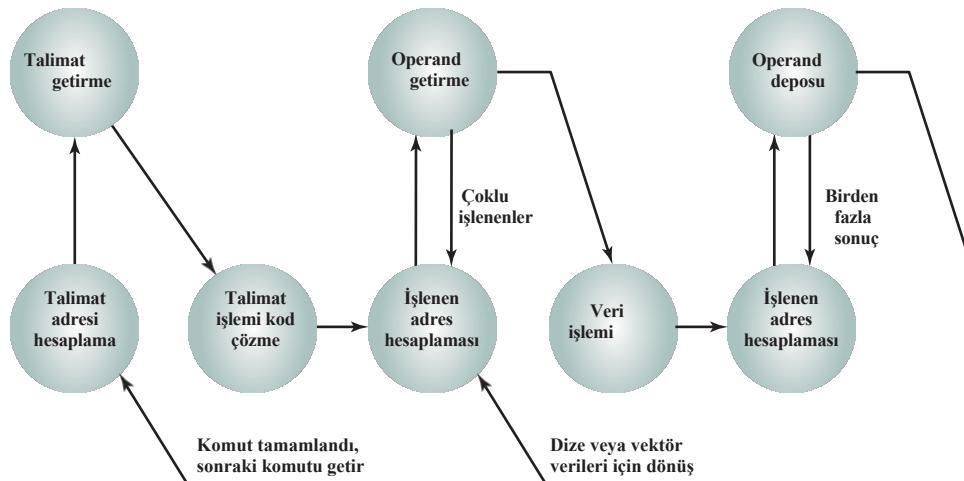
## 12.1 MAKİNE TALİMATI ÖZELLİKLERİ

İşlemcinin çalışması, *makine talimatları* veya *bilgisayar talimatları* olarak adlandırılan, yürütüldüğü talimatlar tarafından belirlenir. İşlemcinin yürütebileceği farklı talimatlar topluluğu işlemcinin *talimat seti* olarak adlandırılır.

### Bir Makine Talimatının Unsurları

Her komut, işlemci tarafından yürütülmesi için gereken bilgileri içermelidir. Şekil 3.6'nın tekrarı olan Şekil 12.1, komutun yürütülmesinde yer alan adımları göstermekte ve dolaylı olarak bir makine komutunun öğelerini tanımlamaktadır. Bu unsurlar aşağıdaki gibidir:

- **İşlem kodu:** Gerçekleştirilecek işlemi belirtir (örneğin, ADD, I/O). İşlem, işlem kodu veya **opcode** olarak bilinen ikili bir kodla belirtilir.
- **Kaynak işlenen referansı:** İşlem bir veya daha fazla kaynak işleneni, yani işlem için girdi olan işlenenleri içerebilir.



Şekil 12.1 Komut Döngüsü Durum Diyagramı

- **Sonuç işlenen referansı:** İşlem bir sonuç üretебilir.
- **Sonraki komut referansı:** Bu, işlemciye bu komutun yürütülmesi tamamlandıktan sonra bir sonraki komutun nereden alınacağını söyler.

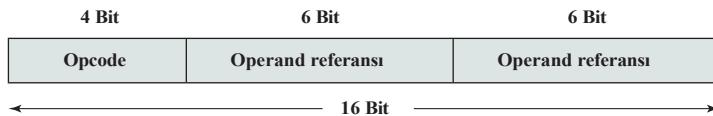
Getirilecek bir sonraki komutun adresi, mimariye bağlı olarak gerçek bir adres veya sanal bir adres olabilir. Genel olarak, bu ayrı komut kümesi mimarisi için şeffaftır. Çoğu durumda, getirilecek bir sonraki komut mevcut komutu hemen takip eder. Bu durumlarda, bir sonraki komuta açık bir referans yoktur. Açık bir referans gerekiğinde ana bellek veya sanal bellek adresi sağlanmalıdır. Bu adresin ne şekilde verileceği Bölüm 13'te ele alınacaktır.

Kaynak ve sonuç operandları dört alandan birinde olabilir:

- **Ana ya da sanal bellek:** Sonraki komut referanslarında olduğu gibi, ana veya sanal bellek adresi sağlanmalıdır.
- **İşlemci kaydı:** Nadir istisnalar dışında, bir işlemci makine komutları tarafından başvurulabilen bir veya daha fazla yazmaç içerir. Yalnızca bir yazmaç varsa, bu yazmacın referansı örtük olabilir. Birden fazla yazmaç varsa, her yazmaca bensersiz bir ad veya numara atanır ve komut istenen yazmacın numarasını içermelidir.
- **Immediate:** İşlenenin değeri, yürütülmekte olan komuttaki bir alanda bulunur.
- **G/C cihazı:** Komut, işlem için G/C modülünü ve cihazını belirtmelidir. Bellek eşlemeli G/C kullanılıyorsa, bu sadece başka bir ana veya sanal bellek adresidir.

### Talimat Temsili

Bilgisayar içinde her bir talimat bir dizi bit ile temsil edilir. Talimat, bilgisayarn kurucu unsurlarına karşılık gelen alanlara bölünmüştür.



**Şekil 12.2** Basit Bir Talimat Formatı

talimat. Basit bir komut formatı örneği Şekil 12.2'de gösterilmektedir. Başka bir örnek olarak, IAS komut formatı Şekil 2.2'de gösterilmiştir. Çoğu komut setinde birden fazla format kullanılır. Komutun yürütülmesi sırasında, bir komut işlemcideki bir komut kaydına (IR) okunur. İşlemci, gerekli işlemi gerçekleştirmek için çeşitli komut alanlarından verileri çıkarabilmelidir.

Hem programcı hem de ders kitaplarını okuyanlar için makine komutlarının ikili gösterimleriyle zordur. Bu nedenle, makine komutlarının *sembolik gösterimini* kullanmak yaygın bir uygulama haline gelmiştir. Bunun bir örneği Tablo 1.1'de IAS komut seti için kullanılmıştır.

Opcode'lar, işlemi belirten *mnemonics* adı verilen kısaltmalarla temsil edilir. Yaygın örnekler şunlardır

ADD	Ekle
SUB	Çıkarma
MUL	Çarpma
DIV	Bölmek
YÜKLE	Bellekten veri yükle STOR Verileri belleğe kaydetme

Operandlar ayrıca sembolik olarak da temsil edilir. Örneğin, komut

ADD R, Y

Y veri konumunda bulunan değeri R kayıtlısının içeriğine ekle anlamına gelebilir. Bu örnekte, Y bellekteki bir konumun adresini ve R belirli bir kayıtçayı eder. İşlemin bir konumun adresi üzerinde değil, içeriği üzerinde gerçekleştirildiğine dikkat edin.

Böylece, bir makine dili programını sembolik biçimde yazmak mümkündür. Her sembolik işlem kodunun sabit bir ikili gösterimi vardır ve programcı her sembolik işlemin yerini belirler. Örneğin, programcı bir tanımlar listesi ile başlayabilir:

X= 513

Y= 514

ve bunun gibi. Basit bir program bu sembolik girdiyi kabul eder, işlem kodlarını ve işlenen referanslarını ikili forma dönüştürür ve ikili makine komutları oluşturur.

Makine dili programcılar yok deneyecek kadar azdır. Günümüzde çoğu program yüksek seviyeli bir dilde ya da bu mümkün değilse Ek B'de ele alınan assembly dilinde yazılmaktadır. Bununla birlikte, sembolik makine dili makine talimatlarını tanımlamak için yararlı bir araç olmaya devam etmektedir ve biz de bu amaçla kullanacağız.

## Talimat Türleri

BASIC veya FORTRAN gibi bir dilde ifade edilebilecek yüksek seviyeli bir dil talimatı düşünün. Örneğin,

$$X = X + Y$$

Bu ifade bilgisayara Y'de saklanan değeri X'te saklanan değere eklemesi ve sonucu 'koyması' talimatını verir. X ve Y değişkenlerinin 513 ve 514 konumlarına karşılık geldiğini varsayıyalım. Basit bir makine talimatları seti varsayıarsak, bu işlem üç talimatla gerçekleştirilebilir:

- 1.** Bellek konumu 513'ün içeriğiyle bir kayıt yükleyin.
- 2.** Bellek konumu 514'ün içeriğini yazmaca ekleyin.
- 3.** Kaydın içeriğini bellek konumu 513'te saklayın.

Göründüğü gibi, tek bir BASIC komutu üç makine komutu gerektirebilir. Bu, yüksek seviyeli bir dil ile makine dili arasındaki ilişkinin tipik bir örneğidir. Yüksek seviyeli bir dil, değişkenleri kullanarak işlemleri kısa ve öz şekilde ifade eder. Bir makine dili ise işlemleri, verilerin yazımcılara veya yazımcılardan hareketini içeren temel bir biçimde ifade eder.

Bize yol gösterecek bu basit örnekle, pratik bir bilgisayarda bulunması gereken talimat türlerini ele alalım. Bir bilgisayar, kullanıcının herhangi bir veri işleme görevini formüle etmesini sağlayan bir dizi talimata sahip olmalıdır. Bunu görmenin bir başka yolu da yüksek seviyeli bir programlama dilinin yeteneklerini düşünmektir. Yüksek seviyeli bir dilde yazılan herhangi bir programın çalıştırılabilmesi için makine diline çevrilmesi gereklidir. Bu nedenle, makine talimatları kümlesi, yüksek seviyeli bir dildeki herhangi bir talimi ifade etmek için yeterli olmalıdır. Bunu akılda tutarak, komut türlerini aşağıdaki gibi kategorize edebiliriz:

- **Veri işleme:** Aritmetik ve mantık talimatları.
- **Veri depolama:** Verilerin kayıt veya bellek konumlarına girmesi veya çıkması.
- **Veri hareketi:** I/O talimatları.
- **Kontrol:** Test ve branş talimatları.

*Aritmetik* komutlar sayısal verilerin işlenmesi için hesaplama yetenekleri sağlar. *Mantık* (Boolean) komutları, bir kelimenin bitleri üzerinde sayılar yerine bitler olarak çalışır; böylece, kullanıcının kullanmak isteyebileceği diğer veri türlerini işlemek için yetenekler sağlarlar. Bu işlemler öncelikle işlemci kayıtlarındaki veriler üzerinde gerçekleştirilir. Bu nedenle, verileri bellek ve yazımcılar arasında taşımak için bellek komutları olmalıdır. Programları ve verileri belleğe aktarmak ve hesaplama sonuçlarını kullanıcıya geri göndermek için *I/O* talimatlarına ihtiyaç vardır. *Test* talimatları bir veri sözcüğünün değerini veya bir hesaplamadan durumunu test için kullanılır. *Dallanma* talimatları daha sonra verilen karara bağlı olarak farklı bir talimat setine dallanmak için kullanılır.

Çeşitli talimat türlerini bu bölümün ilerleyen kısımlarında daha ayrıntılı olarak inceleyeceğiz.

## Adres Sayısı

İşlemci mimarisini tanımlamanın geleneksel yollarından biri, her bir komutun içерdiği adres sayısıdır. Bu boyut, işlemci tasarıminın artan karmaşıklığı ile daha az önemli hale gelmiştir. Yine de, bu noktada bu ayrimı çizmek ve analiz etmek faydalı olacaktır.

Bir talimatta ihtiyaç duyulabilecek maksimum adres sayısı nedir? Aritmetik ve mantık komutlarının en fazla işlem gerektireceği açıklır. Neredeyse tüm aritmetik ve mantık işlemleri ya tek (bir kaynak işlenen) ya da ikilidir (iki kaynak işlenen). Dolayısıyla, kaynak işlenenleri referans almak için en fazla iki adrese ihtiyacımız olacaktır. Bir işlemin sonucu saklanmalıdır, bu da bir hedef işlenemi tanımlayan üçüncü bir adrese işaret eder. Son olarak, bir komut tamamlandıktan sonra, bir sonraki komut getirilmelidir ve adresi gereklidir.

Bu mantık silsilesi, bir komutun makul olarak dört adres referansı içermesi gerekgöstermektedir: iki kaynak operand, bir hedef operand ve bir sonraki komutun adresi. Çoğu mimaride, birçok komutun bir, iki veya üç işlenen adresi vardır ve bir sonraki komutun adresi örtütür (program sayacıdan elde edilir). Çoğu mimaride ayrıca daha fazla işlenene sahip birkaç özel amaçlı komut bulunur. Örneğin, Bölüm 13'te açıklanan ARM mimarisinin çoklu yükleme ve saklama komutları, tek bir komutta 17'ye kadar yazmaç işlenenini tanımlar.

**Şekil 12.3.**  $Y = (A - B)/[C + (D * E)]$  hesaplamak için kullanılabilecek tipik bir, iki ve üç adresli komutları karşılaştırmaktadır. Üç adresle, her komut iki kaynak işlenen konumu ve bir hedef işlenen konumu belirtir. Çünkü işlenen konumlarından herhangi birinin değerini değiştirmemeyi seçiyoruz,

Talimatlar	Yorum
Y, A, B	$Y \leftarrow A - B$
MPY T, D, E	$T \leftarrow D * E$
ADD T, T, C	$T \leftarrow T + C$
DIV Y, Y, T	$Y \leftarrow Y \div T$

(a) Üç adresli talimatlar

Talimatlar	Yorum
MOVE Y, A	$Y \leftarrow A$
SUB Y, B	$Y \leftarrow Y - B$
MOVE T, D	$T \leftarrow D$
MPY T, E	$T \leftarrow T * E$
EKLE T, C	$T \leftarrow T + C$
DIV Y, T	$Y \leftarrow Y \div T$

(b) İki adresli talimatlar

Talimatlar
YÜK D
MPY E
ADD C
STOR Y
LOAD A
SUB B
DIV Y
STOR Y

AC $\leftarrow$ D
AC $\leftarrow$ AC $\times$ E
AC $\leftarrow$ AC + C
Y $\leftarrow$ AC
AC $\leftarrow$ A
AC $\leftarrow$ AC - B
AC $\leftarrow$ AC $\div$ Y
Y $\leftarrow$ AC
$\leftarrow$

(c) Tek adresli talimatlar

$$\text{Şekil 12.3} \quad Y \text{yi Yürütecek Programlar} = \frac{A - B}{C + (D * E)}$$

## 418 BÖLÜM 12 / YÖNERGE SETLERİ: ÖZELLİKLER VE İŞLEVLER

Bazı ara sonuçları saklamak için geçici bir konum olan T kullanılır. Dört komut olduğuna ve orijinal ifadenin beş işlenene sahip olduğuna dikkat edin.

Üç adresli komut formatları yaygın değildir çünkü üç adres referansını tutmak için nispeten uzun bir komut formatı gerektirirler. İki adresli komutlarda ve ikili işlemlerde, bir adres hem işlenen hem de sonuç olarak çift görev yapmalıdır. Böylece, SUB Y, B komutu Y - B hesaplamasını gerçekleştirir ve sonucu Y'de saklar. İki adresli format alan gereksinimini azaltır ancak aynı zamanda bazı gariplikler de getirir. Bir operandın değerini değiştirmekten kaçınmak için, işlemi gerçekleştirirmeden önce değerlerden birini bir sonuca veya geçici bir konuma taşımak için bir MOVE komutu kullanılabilir. Örnek programımız altı komuta genişlemektedir.

Daha basit olanı ise tek adresli komuttur. Bunun çalışması için ikinci bir adresin örtük olması gereklidir. Bu, daha önceki makinelerde yaygındı ve zimni adres **akümülatör** (AC) olarak bilinen bir işlemci kaydı idi. Akümülatör işlenenlerden birini tutar ve sonucu saklamak için kullanılır. Örneğimizde, görevi yerine getirmek için sekiz talimat gereklidir.

Aslında, bazı talimatlar için sıfır adresle yetinmek mümkündür. Sıfır adresli komutlar yoğun adı verilen özel bir bellek organizasyonuna uygulanabilir. Yiğin, son giren ilk çıkar konum kümeleridir. Yiğin bilinen bir konumdadır ve genellikle en azından en üstteki iki öğe işlemci yazımcılarındadır. Böylece, sıfır adresli talimatlar en üstteki iki yiğin elemanına referans verir. Yiğinlar Ek I'de açıklanmıştır. Kullanımları bu bölümün ilerleyen kısımlarında ve Bölüm 13'te daha ayrıntılı olarak incelenmiştir.

Tablo 12.1, sıfır, bir, iki veya üç adresli komutlara getirilecek yorumları özetlemektedir. Tablodaki her durumda, bir sonraki komutun adresinin örtük olduğu ve iki kaynak işlenenli ve bir sonuç işlenenli bir işlemin gerçekleştirileceği varsayılmaktadır.

Komut başına adres sayısı temel bir tasarım kararıdır. Komut başına daha az adres, daha ilkel olan ve daha az karmaşık bir işlemci gerektiren komutlarla sonuçlanır. Aynı zamanda daha kısa uzunlukta komutlarla sonuçlanır. Öte yandan, programlar daha fazla toplam komut içerir, bu da genel olarak daha uzun yürütme süreleri ve daha uzun, daha karmaşık programlarla sonuçlanır. Ayrıca, tek adresli ve çok adresli komutlar arasında önemli bir eşik vardır. Tek adresli komutlarda, programcı genellikle yalnızca bir genel amaçlı komuta sahiptir.

**Tablo 12.1** Komut Adreslerinin Kullanımı (Dallanmayan Komutlar)

Adres Sayısı	Sembolik Temsil	Yorumlama
3	OP A, B, C	A <b>d</b> B OP C
2	OP A, B	A <b>d</b> A OP B
1	OP A	AC <b>d</b> AC OP A
0	OP	T <b>d</b> (T - 1) OP T

AC= akümülatör T=

yığının üstü

(T - 1)= yiğinin ikinci elemanı A, B, C=

bellek veya kayıt konumları

kaydedici, akümülatör. Çoklu adres talimatlarında, birden fazla genel amaçlı yazmaç olması yaygındır. Bu, bazı işlemlerin yalnızca üzerinde gerçekleştirilmesine olanak tanır. Yazmaç referansları bellek referanslarından daha hızlı olduğundan, bu durum yürütmeyi hızlandırır. Esneklik ve birden fazla kullanma yeteneği nedeniyle, çoğu çağdaş makine iki ve üç adresli talimatların bir karışımını kullanır.

Talimat başına adres sayısının seçilemesiyle ilgili tasarım ödünləşimleri başka faktörler tarafından karmaşık hale getirilir. Bir adresin bir bellek konumunu mu yoksa bir yazmacı mı ifade ettiği konusu vardır. Daha az sayıda yazmaç olduğundan, bir yazmaç referansı için daha az bit gerekir. Ayrıca, Bölüm 13'te göreceğimiz gibi, bir makine çeşitli adresleme modları sunabilir ve modun belirtilmesi bir veya daha fazla bit gerektirir. Sonuç olarak çoğu işlemci tasarımları çeşitli komut formatları içerir.

## Komut Seti Tasarımı

Bilgisayar tasarımlının en ilginç ve en çok analiz edilen yönlerinden biri komut seti tasarımıdır. Bir komut setinin tasarımları karmaşıktır çünkü bilgisayar sisteminin pek çok yönünü etkiler. Komut kümesi, işlemci tarafından gerçekleştirilen işlevlerin çoğunu tanımlar ve bu nedenle işlemcinin uygulanması üzerinde önemli bir etkiye sahiptir. Komut seti, programcının işlemciyi kontrol etme aracıdır. Bu nedenle, komut kümesi tasarlanırken programcının gereksinimleri göz önünde bulundurulmalıdır.

Komut setlerinin tasarımlıyla ilgili en temel konulardan bazlarının hala tartışılmış olduğunu bilmek sizi şaşırtabilir. Aslında, son yıllarda bu temel konulara ilişkin anlaşmazlıkların seviyesi artmıştır. Bu temel tasarım konularının en önemlileri aşağıdakileri içermektedir:

- **İşlem repertuarı:** Kaç tane ve hangi işlemlerin sağlanacağı ve işlemlerin ne kadar karmaşık olması gerektiği.
- **Veri türleri:** Üzerinde işlem yapılan çeşitli veri türleri.
- **Talimat formatı:** Komut uzunluğu (bit cinsinden), adres sayısı, çeşitli alanların boyutu vb.
- **Kayıtlar:** Talimatlar tarafından başvurulabilen işlemci kayıtlarının sayısı ve bunların kullanımı.
- **Adresleme:** Bir işlenenin adresinin belirtildiği mod veya modlar.

Bu konular birbiriley oldukça ilişkilidir ve bir komut seti tasarlanırken birlikte ele alınmalıdır. Bu kitap, elbette bunları bir sırayla ele almak zorundadır, ancak birbirleriyle olan ilişkilerini göstermeye çalışılmıştır.

Bu konunun önemi nedeniyle, Üçüncü Bölümün büyük bir kısmı komut seti tasarımlına ayrılmıştır. Bu genel bakış bölümünün ardından, bu bölümde veri türleri ve işlem repertuarı incelenmektedir. Bölüm 13 adresleme modlarını (değerlendirilmesini de içerir) ve komut formatlarını inceler. Bölüm 15 indirgenmiş komut seti bilgisayarını (RISC) incelemektedir. RISC mimarisini, ticari bilgisayarlarla geleneksel olarak verilen komut kümesi tasarım kararlarının birçoğunu sorgulamaktadır.

## 12.2 İŞLENEN TÜRLERİ

Makine talimatları veriler üzerinde çalışır. En önemli genel veri kategorileri şunlardır

- Adresler
- Sayılar
- Karakterler
- Mantıksal veri

Bölüm 13'te adresleme modlarını tartışıırken, adreslerin aslında bir veri biçimi olduğunu göreceğiz. Birçok durumda, ana veya sanal bellek adresini belirlemek için bir komuttaki işlenen referansı üzerinde bazı hesaplamalar yapılmalıdır. Bu bağlamda, adresler işaretsiz tamsayılar olarak düşünülebilir.

Diğer yaygın veri türleri sayılar, karakterler ve mantıksal verilerdir ve bunların her biri bu bölümde kısaca incelenmiştir. Bunun ötesinde, bazı makineler özel veri türleri veya veri yapıları tanımlar. Örneğin, doğrudan bir liste veya karakter dizisi üzerinde çalışan makine işlemleri olabilir.

### Sayılar

Tüm makine dilleri sayısal veri türleri içerir. Sayısal olmayan veri işlemlerinde bile, sayıçalar, alan genişlikleri ve benzeri işlevler için sayılara ihtiyaç vardır. Sıradan matematikte kullanılan sayılar ile bilgisayarda saklanan sayılar arasındaki önemli bir ayrim, ikincisinin sınırlı olmasıdır. Bu iki anlamda doğrudur. Birincisi, bir makinede temsil edilebilen sayıların büyüğüğünde bir sınır vardır ve ikincisi, kayan noktalı sayılar söz konusu olduğunda, hassasiyetlerinde bir sınır vardır. Bu nedenle, programcı yuvarlama, taşıma ve düşük taşmanın sonuçlarını anlamakla karşı karşıya kalır.

Bilgisayarlarda üç tür sayısal veri yaygındır:

- İkili tamsayı veya ikili sabit nokta
- İkili kayan nokta
- Ondalık

İlk ikisini Bölüm 10'da biraz ayrıntılı olarak inceledik. Geriye ondalık sayılar hakkında birkaç söz söylemek kalıyor.

Tüm dahili bilgisayar işlemleri doğası gereği ikili olmasına rağmen, sistemin insan kullanıcıları ondalık sayılarla uğraşır. Bu nedenle, girişte ondalıktan ikiliye ve çıkışta ikiliden ondalık sayıya dönüştürme zorunluluğu vardır. Çok fazla G/C ve nispeten az, nispeten basit hesaplamanın olduğu uygulamalar için sayıların ondalık biçimde saklanması ve üzerinde işlem yapılması tercih edilir. Bu amaç için en yaygın gösterim **paketlenmiş ondalıktır**.<sup>1</sup>

---

<sup>1</sup> Ders kitaplarında bu genellikle ikili kodlanmış ondalık (BCD) olarak adlandırılır. , BCD her bir ondalık basamağının bzersiz bir 4-bit dizisi ile kodlanması ifade eder. Paketlenmiş ondalık, her iki basamak için bir bayt kullanarak BCD kodlu basamakların depolanmasını ifade eder.

Paketlenmiş ondalıkta, her ondalık basamak 4 bitlik bir kodla temsil edilir ve bayt başına iki basamak saklanır. Böylece, 0= 000, 1= 0001, 2= 1000 ve 9= 1001. Bunun oldukça verimsiz bir kod olduğunu unutmuyan çünkü 16 olası 4 bitlik değerden sadece 10 tanesi kullanılır. Sayıları oluşturmak için, 4 bitlik kodlar genellikle 8 bitin katları şeklinde bir araya getirilir. Böylece, 246 için kod 0000 0010 0100 0110'dur. Bu kod, düz bir ikili gösterimden açıkça daha az, ancak dönüştürme ek yükünü örter. Negatif sayılar, paketlenmiş ondalık basamak dizisinin sol ya da sağ ucuna 4 bitlik bir işaret basamağı eklenerek gösterilebilir. Standart işaret değerleri pozitif (+) için 1100 ve negatif (-) için 1101'dir.

Birçok makine, doğrudan paketlenmiş ondalık sayılar üzerinde işlem yapmak için aritmetik talimatlar sağlar. Algoritmalar Bölüm 9.3'te açıklanılanlara oldukça benzerdir, ancak ondalık taşıma işlemini dikkate almmalıdır.

## Karakterler

Yaygın bir veri biçimini metin ya da karakter dizileridir. Metinsel veriler için en kullanışlı veriler olsa da, karakter formunda veri işleme ve iletişim sistemleri tarafından kolayca saklanamaz veya aktarılamazlar. Bu tür sistemler ikili veriler için tasarlanmıştır. Bu nedenle, karakterlerin bir dizi bit ile temsil edildiği bir dizi kod geliştirilmiştir. Bunun belki de en eski yaygın örneği Mors kodudur. Günümüzde en yaygın kullanılan karakter kodu, Amerika Birleşik Devletleri'nde Bilgi Değisimi için Amerikan Standart Kodu (ASCII; bkz. Ek H) olarak adlandırılan Uluslararası Referans Alfabesi'dir (IRA). Bu koddaki her bir karakter 7 bitlik benzersiz bir desenle temsil edilir; böylece 128 farklı karakter temsil edilebilir. Bu, yazdırılabilir karakterleri temsil etmek için gerekenden daha büyük bir sayıdır ve bazı kalıplar *kontrol* karakterlerini temsil eder. Bu kontrol karakterlerinden bazıları sayfadaki karakterlerin yazdırılmasını kontrol ilgilidir. Diğerleri ise iletişim prosedürleriyle ilgilidir. IRA kodlu karakterler neredeyse her zaman karakter başına 8 bit kullanılarak saklanır ve iletilir. Sekizinci bit 0'a ayarlanabilir ya da hata tespiti için eşlik biti olarak kullanılabilir. İlkinci durumda, bit, her sekizlideki toplam ikili 1 sayısı her zaman tek (tek eşlik) veya her zaman çift (çift eşlik) olacak şekilde ayarlanır.

Tablo H.1'de (Ek H) IRA bit modeli 011XXXX için 0'dan 9'a kadar olan rakamların en sağdaki 4 bitte 0000'dan 1001'e kadar olan ikili eşdeğerleriyle temsil edildiğine dikkat edin. Bu, paketlenmiş ondalık ile aynı koddur. Bu, 7 bitlik IRA ve 4 bitlik paketlenmiş ondalık gösterim arasındaki uyumu kolaylaştırır.

Karakterleri kodlamak için kullanılan bir başka kod da Genişletilmiş İkili Kodlu Ondalık Değişim Kodu'dur (EBCDIC). EBCDIC IBM ana bilgisayarlarında kullanılır. Bu 8 bitlik bir koddur. IRA'da olduğu gibi, EBCDIC de paketlenmiş ondalık ile uyumludur. EBCDIC durumunda, 1111000 ile 11111001 arasındaki kodlar 0 ile 9 arasındaki rakamları temsil eder.

## Mantıksal Veri

Normalde, her kelime veya diğer adreslenebilir birim (bayt, yarımkelime...) tek bir veri birimi olarak ele alınır. Ancak bazen  $n$  bitlik bir birimi, her biri 0 veya 1 değerine sahip  $n$  adet bitlik veri öğesinden oluşmuş gibi düşünmek yararlı olabilir. Veriler bu şekilde, *mantıksal* veri olarak kabul edilirler.

Bit yönelimli görünümün iki avantajı vardır. Birincisi, bazen her bir ögenin yalnızca 1 (doğru) ve 0 (yanlış) değerlerini alabildiği Boolean veya ikili veri öğelerinden oluşan bir dizi saklamak isteyebiliriz. Mantıksal verilerle, bellek bu depolama için en verimli şekilde kullanılabilir. İkinci olarak, bir veri ögesinin bitlerini değiştirmek istediğimiz durumlar vardır. Örneğin, kayan noktalı işlemler yazılımda uygulanırsa, bazı işlemlerde önemli bitleri kaydırıbmamız gereklidir. Başka bir örnek: IRA'dan paketlenmiş ondalık sayıya dönüştürmek için her baytin en sağdaki 4 bitini çıkarmamız gereklidir. Yukarıdaki örneklerde, aynı verilerin bazen mantıksal, bazen de sayısal ya da metin olarak ele alındığını dikkat edin. Bir veri biriminin "türü", gerçekleştirilen işleme göre belirlenir. Normalde durum böyle olmasa da yüksek seviyeli dillerde, neredeyse her zaman makine dilinde durum böyledir.

## 12.3 INTEL x86 VE ARM VERİ TÜRLERİ

### x86 Veri Türleri

x86 8 (byte), 16 (word), 32 (doubleword), 64 (quad-word) ve 128 (double quadword) bit uzunluğundaki veri tipleriyle çalışabilir. Veri yapılarında maksimum esneklik ve verimli bellek kullanımı sağlamak için, kelimelerin çift sayılı adreslerde hizalanması gerekmek; çift kelimelerin 4'e eşit olarak bölünebilen adreslerde hizalanması gerekmek; dört kelimelerin 8'e eşit olarak bölünebilen adreslerde hizalanması gerekmek; vb. Ancak, verilere 32 bitlik bir veri yolu üzerinden erişildiğinde, veri aktarımı 4'e bölünebilen adreslerden başlayarak çift kelimelek birimler halinde gerçekleşir. İşlemciyanlı hizalanmış değerler için talebi veri yolu aktarımı için bir dizi talebe dönüştürür. Tüm Intel 80x86 makinelerinde olduğu gibi, x86 little-endian stilini kullanır; , en az anlamlı bayt en düşük adreste saklanır (endianlık tartışması için Ek 12A'ya bakın).

Byte, word, doubleword, quadword ve double quadword genel veri tipleri olarak adlandırılır. Buna ek olarak, x86 belirli talimatlar tarafından tanımlanır ve üzerinde işlem yapılan etkileyici bir dizi özel veri türünü destekler. Tablo 12.2 bu türleri özetlemektedir.

Şekil 12.4'te x86 sayısal veri türleri gösterilmektedir. İşaretli tamsayılar ikiye tümleyen gösterimindedir ve 16, 32 veya 64 bit uzunluğunda olabilir. Kayan nokta türü aslında kayan nokta birimi tarafından kullanılan ve kayan nokta komutları tarafından işletilen bir dizi türü ifade eder. Kayan nokta gösterimleri IEEE 754 standardına uygundur.

Paketlenmiş SIMD (tek komut-çoklu veri) veri türleri, multimedya uygulamalarının performansını optimize etmek için komut setinin uzantılarının bir parçası olarak x86 mimarisine dahil edilmiştir. Bu uzantılar MMX (multimedya uzantıları) ve SSE'yi (akış SIMD uzantıları) içerir. Temel kavram, birden fazla işlenenin tek bir referanslı bellek öğesine paketlenmesi ve bu birden fazla işlenenin paralel olarak çalıştırılmasıdır. Veri türleri aşağıdaki gibidir:

- **Paketlenmiş bayt ve paketlenmiş bayt tamsayısı:** Bit alanı veya tamsayı olarak yorumlanan 64 bitlik bir dört kelimeye veya 128 bitlik çift dört kelimeye paketlenmiş baytlar.
- **Paketlenmiş kelime ve paketlenmiş kelime tamsayısı:** 64 bitlik dörtlü kelime veya 128 bitlik çift dörtlü kelime içine paketlenmiş 16 bitlik kelimeler, bit alanı veya tamsayı olarak yorumlanır.

**Tablo 12.2** x86 Veri Türleri

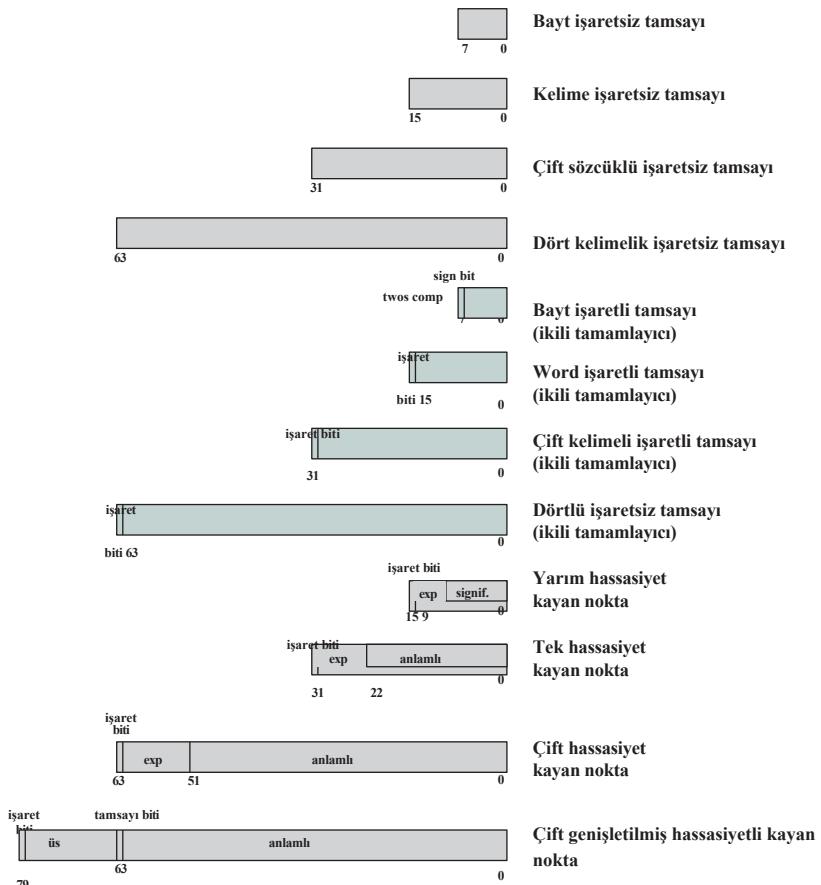
Veri Tipi	Açıklama
Genel	Bayt, word (16 bit), doubleword (32 bit), quadword (64 ) ve double rastgele ikili içeriğe sahip dört kelimeli (128 bit) konumlar.
Tamsayı	Bir bayt, kelime veya çift kelimedede bulunan, ikişer ikişer kullanılan işaretli bir ikili değer Tamamlayıcı temsil.
Sıralı	Bir bayt, sözcük veya çift sözcük içinde bulunan işaretsiz bir tamsayı.
Paketlenmemiş ikili kodlanmış ondalık (BCD)	0'dan 9'a kadar olan aralıktaki bir BCD rakamının gösterimi, bir rakam her bir bayt.
Paketlenmiş BCD	İki BCD basamağının paketlenmiş bayt gösterimi; 0 ila 99 aralığında değer.
Yakın işaretçi	16 bitlik, 32 bitlik veya 64 bitlik etkin bir adres olup, bir segmente edilmemiş bellekteki tüm işaretçiler ve referanslar için kullanılabilir bölümlenmiş bir bellekte bir bölüm içinde.
Uzak işaretçi	16-bit segment seçici ve 16'lık bir offsetten oluşan mantıksal bir adres, 32 veya 64 bit. Uzak işaretçiler, böülümlere ayrılmış bellek referansları için kullanılır erişilen segmentin kimliğinin bilinmesi gereken bellek modeli açıkça belirtilmiştir.
Bit alanı	Her bir bitin konumunun dikkate alındığı bitişik bir bit dizisi bağımsız bir birim olarak. Bir bit dizesi herhangi bir baytın herhangi bir bit konumundan başlayabilir ve en fazla 32 bit içerebilir.
Bit dizesi	Sıfır ila $2^{23}$ - 1 bit içeren bitişik bir bit dizisi.
Bayt dizesi	Aşağıdakileri içeren bitişik bir bayt, sözcük veya çift sözcük dizisi sıfır ila $2^{23}$ - 1 bayt.
Kayan nokta	Şekil 12.4'e bakınız.
Paketlenmiş SIMD (tek talimat, çoklu veri)	Paketlenmiş 64 bit ve 128 bit veri türleri.

- **Paketlenmiş çift sözcük ve paketlenmiş çift sözcük tamsayısı:** 64 bitlik bir dört kelimeye veya 128 bitlik bir çift dört kelimeye paketlenmiş 32 bitlik çift kelimeler, bir bit alanı veya bir tamsayı olarak yorumlanır.
- **Paketlenmiş dört sözcük ve paketlenmiş dört sözcük tamsayısı:** Bit alanı veya tamsayı olarak yorumlanan 128 bitlik bir çift dört kelimeye paketlenmiş iki 64 bitlik dört kelime.
- **Paketlenmiş tek hassasiyetli kayan nokta ve paketlenmiş çift hassasiyetli kayan nokta:** Dört adet 32 bit kayan nokta veya iki 64 bit kayan nokta değeri 128 bitlik bir çift dört kelimeye paketlenir.

## ARM Veri Türleri

ARM işlemcileri 8 (bayt), 16 (yarım kelime) ve 32 (kelime) bit uzunluğundaki veri türlerini destekler. Normalde, yarınl kelime erişimi yarınl kelime hizalamalı ve kelime erişimleri kelime hizalamalı olmalıdır. Hizalanmamış erişim girişimleri için mimari üç alternatif destekler.

- Varsayılan durum:
  - Adres kesilmiş olarak kabul edilir, adres bitleri[1:0] word erişimleri için sıfır olarak kabul edilir ve adres biti[0] yarınwörde erişimleri için sıfır olarak kabul edilir.

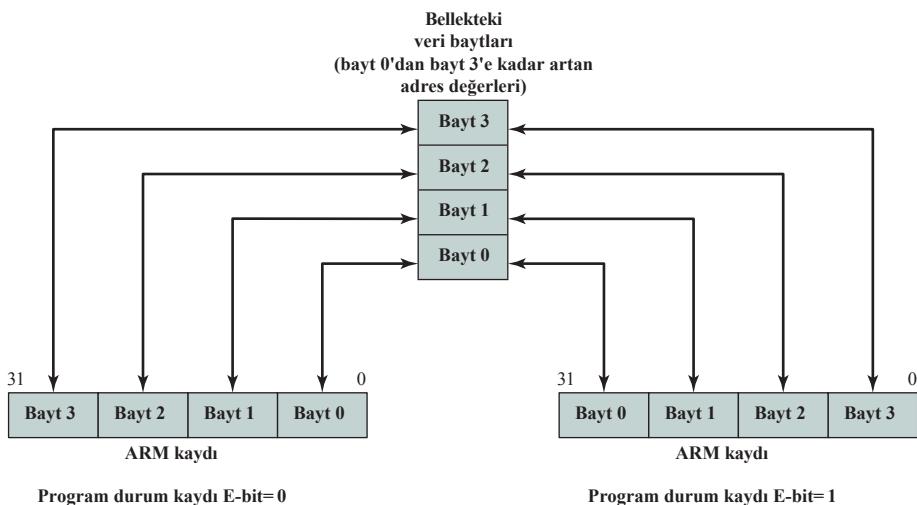


Şekil 12.4 x86 Sayısal Veri Formatları

- Tek kelime yükle ARM talimatları, mimari olarak kelime hizalamalı olmayan bir adres tarafından aktarılan kelime hizalamalı verileri, en az anlamlı iki adres bitinin değerine bağlı olarak bir, iki veya üç bayt sağa döndürmek için tanımlanmıştır.
- **Hızalama kontrolü:** Uygun kontrol biti ayarlandığında, bir veri iptal sağlanır. Hizalanmamış erişim girişimi için bir hızalama hatası olduğunu gösterir.
- **Hizalanmamış erişim:** Bu seçenek etkinleştirildiğinde işlemci, bitişik baytların programlayıcıya şeffaf bir şekilde aktarılmasını sağlamak için bir veya daha fazla bellek erişimi kullanır.

Her üç veri tipi (bayt, yarımkelime ve kelime) için, değerin işaretetsiz, negatif olmayan bir tamsayıyı temsil ettiği işaretetsiz bir yorumlama desteklenir. Her üç veri tipi de ikiye tümleyen işaretli tamsayılar için kullanılabilir.

ARM işlemci uygulamalarının çoğu kayan nokta donanımı sağlamaz, bu da güç ve alan tasarrufu sağlar. Bu tür işlemcilerde kayan nokta aritmetiği gerekiyorsa, yazılımda uygulanmalıdır. ARM, IEEE 754'te tanımlanan tek ve çift hassasiyetli kayan nokta veri türlerini destekleyen isteğe bağlı bir kayan nokta yardımcı işlemcisini destekler.



**Şekil 12.5** E-Bit ile ARM Endian Desteği-Word Yükleme/Depolama

**ENDIAN DESTEĞİ** Sistem kontrol kaydındaki bir durum biti (E-bit) SETEND komutu kullanılarak program kontrolü altında ayarlanır ve temizlenir. E biti, verilerin hangi endian değerinde yükleneceğini ve saklanacağını tanımlar. Şekil 12.5, bir word yükleme veya saklama işlemi için E-bit ile ilişkili işlevselliği göstermektedir. Bu mekanizma, veri yapılarına işletim sistemlerinin/ortamlarının tersi endianlıkta erişmeleri gerektiğini bilen sistem tasarımcıları için verimli dinamik veri yükleme/depolama sağlar. Her bir veri baytinin adresinin bellekte sabit olduğunu unutmayın. Ancak, bir yazmaştaki bayt şeridi farklıdır.

## 12.4 OPERASYON TÜRLERİ

Farklı işlem kodlarının sayısı makineden makineye büyük ölçüde değişir. Ancak, tüm makinelerde aynı genel işlem türleri bulunur. Yararlı ve tipik bir sınıflandırma aşağıdaki gibidir:

- Veri aktarımı
- Aritmetik
- Mantıksal
- Dönüşüm
- I/O
- Sistem kontrolü
- Kontrolün devri

Tablo 12.3 (HAYE98)'e dayanarak her bir kategorideki yaygın komut türlerini listeler. Bu bölüm, bu çeşitli işlem türlerinin kısa bir incelemesini ve işlemcinin belirli bir işlem türünü yürütmek için gerçekleştirdiği eylemlerin kısa bir tartışmasını (Tablo 12.4'te özetlenmiştir) sunmaktadır. İkinci konu Bölüm 14'te daha ayrıntılı olarak incelenmektedir.

**Tablo 12.3** Ortak Komut Kümesi İşlemleri

Tip	Operasyon Adı	Açıklama
Veri aktarımı	Taşı (transfer)	Kaynaktan hedefe kelime veya blok aktarımı
	Mağaza	İşlemciden belleğe kelime aktarımı
	Yükle (getir)	Kelimeyi bellekten işlemciye aktarma
	Değişim	Kaynak ve hedefin içeriğini değiştir
	Temizle (sıfırla)	0'lardan oluşan kelimeyi hedefe aktarın
	Set	1'lerden oluşan kelimeyi hedefe aktarın
	İtme	Sözcüğü kaynaktan yiğinin en üstüne aktarın
	Pop	Sözcüğü yiğinin üstünden hedefe aktarma
Aritmetik	Ekle	İki işlenenin toplamını hesaplama
	Çıkarma	İki işlenenin farkını hesaplama
	Çarpma	İki işlenenin çarpımını hesaplama
	Bölmek	İki işlenenin bölümünü hesaplama
	Mutlak	İşlenenin mutlak değeriyle değiştirir
	Negate	İşlenenin işaretini değiştir
	Artış	İşlenene 1 ekler
	Azalma	İşlenenden 1 çıkarma
Mantıksal	VE	Mantıksal AND gerçekleştirin
	VEYA	Mantıksal VEYA gerçekleştirin
	DEĞİL	(tamamlayıcı) Mantıksal DEĞİL gerçekleştirin
	Exclusive-OR	Mantıksal XOR gerçekleştirin
	Test	Belirtilen koşulu test edin; sonuca göre bayrak(lar) ayarlayın
	Karşılaştırma	İki veya daha fazla işlenenin mantıksal veya aritmetik karşılaştırmasını yapar; sonuca göre bayrak(lar) ayarlar
	Kontrol Değişkenleri ni Ayarlama	Koruma amaçlı kontrolleri, kesme işlemlerini, zamanlayıcı kontrolünü vb. ayarlamak için talimatlar sıfırı.
	Vardiya	İşleneni sola (sağa) kaydırma, sonunda sabitleri tanıtmaya
	Döndür	Sola (sağa) kaydırma işleneni, sarma sonu ile
	Atlama (dal)	Koşulsuz aktarım; PC'yi belirtilen adresle yükleyin
Kontrolün devri	Koşullu Atlama	Belirtilen durumu test edin; duruma bağlı olarak bilgisayarı belirtilen adresle yükleyin veya hiçbir şey yapmayın
	Alt Programa Atlama	Geçerli program kontrol bilgilerini bilinen konuma yerleştirin; belirtilen adrese atlayın
	Dönüş	PC ve diğer kayıtların içeriğini bilinen konumdan değiştirin
	Yürütmek	İşleneni belirtilen konumdan getirin ve talimat olarak yürütün; PC'yi değiştirmeyin
	Atla	Sonraki komutu atlama için PC'yi artırın
	Koşullu Atla	Belirtilen koşulu test edin; koşula bağlı olarak atlayın veya hiçbir şey yapmayın
	Dur!	Program yürütmemeyi durdur
	Bekle (hold)	Program yürütmesini durdurun; belirtilen koşulu tekrar test edin; koşul sağlanlığında yürütmeye devam edin
	Operasyon yok	Hiçbir işlem yapılmaz, ancak program yürütülmeye devam edilir

Tip	Operasyon Adı	Açıklama
Giriş/çıkış	Giriş (okuma)	Verileri belirtilen G/C portundan veya cihazdan hedefe aktarma (örn. ana bellek veya işlemci kaydı)
	Çıkış (yazma)	Verileri belirtilen kaynaktan G/C portuna veya cihaza aktarma
	G/C'yi başlat	G/C işlemini başlatmak için talimatları G/C işlemcisine aktarın
	Test G/C	Durum bilgilerini G/C sisteminden belirtilen hedefe aktarma
Dönüşüm	Tercüme et	Bir yazışma tablosuna dayalı olarak belleğin bir bölümündeki değerleri çevirme
	Dönüştür	Bir sözcüğün içeriğini bir biçimden diğerine dönüştürme (örneğin, paketlenmiş ondalıktan ikiliye)

**Tablo 12.4** Çeşitli İşlem Türleri için İşlemci Eylemleri

Veri aktarımı	Verileri bir konumdan diğerine aktarma
	Eğer bellek söz konusu ise: Bellek adresini belirleyin Sanaldan gerçek belleğe adres dönüşümü gerçekleştirmeye Önbellegi kontrol etme Bellek okuma/yazmayı başlatma
Aritmetik	Öncesinde ve/veya sonrasında veri aktarımı içerebilir
	ALU'da işlev gerçekleştirme
	Durum kodlarını ve bayrakları ayarlama
Mantıksal	Aritmetik ile aynı
Dönüşüm	Aritmetik ve mantıksal işlemlere benzer. Dönüşüm gerçekleştirmek için özel mantık içerebilir
Kontrolün devri	Program sayacını güncelleyin. Alt yordam çağrı/geri dönüşü için parametre geçişini ve bağlantıyı yönetin
I/O	I/O modülüne komut verin
	Bellek eşlemeli G/C ise, bellek eşlemeli adresi belirleyin

## Veri Aktarımı

En temel makine komutu türü veri aktarım komutudur. Veri aktarım talimatı birkaç şeyi belirtmelidir. İlk olarak, kaynak ve hedef işlenenlerin konumu. Her konum bellek, bir yazmaç ya da yiğinin üstü olabilir. İkinci olarak, aktarılacak verinin uzunluğu belirtilmelidir. Üçüncü olarak, işlenenleri olan tüm komutlarda olduğu gibi, her işlenen için adresleme modu. Bu son nokta Bölüm 13'te tartışılmıştır.

Bir komut setine dahil edilecek veri aktarım talimatlarının seçimi, tasarımcının yapması gereken ödüntleşim türlerini örneklerdir. Örneğin, bir işlenenin genel konumu (bellek veya yazmaç) ya işlem kodunun ya da işlenenin spesifikasyonunda belirtilebilir. Tablo 12.5, en yaygın IBM EAS/390 veri aktarım yönergelerinin örneklerini gösterir. Belirtmek için varyantlar olduğunu unutmuyın.

**Tablo 12.5** IBM EAS/390 Veri Aktarım İşlemlerine Örnekler

İşlem Anımsatıcısı	İsim	Aktarılan Bit Sayısı	Açıklama
L	Yük	32	Bellekten kayda aktarım
LH	Yarım Kelime Yükle	16	Bellekten kayda aktarım
LR	Yük	32	Sicilden sicile transfer
LER	Yük (kısa)	32	Kayan nokta yazmacından kayan nokta yazmacına aktarım
LE	Yük (kısa)	32	Bellekten kayan nokta yazmacına aktarım
LDR	Yük (uzun)	64	Kayan nokta yazmacından kayan nokta yazmacına aktarım
LD	Yük (uzun)	64	Bellekten kayan nokta yazmacına aktarım
ST	Mağaza	32	Kayıttan belleğe aktarım
STH	Yarım Kelime Sakla	16	Kayıttan belleğe aktarım
STC	Mağaza Karakteri	8	Kayıttan belleğe aktarım
STE	Mağaza (kısa)	32	Kayan nokta yazmacından belleğe aktarım
STD	Mağaza (uzun)	64	Kayan nokta yazmacından belleğe aktarım

aktarılacak veri miktarı (8, 16, 32 veya 64 bit). Ayrıca, register'dan register'a, register'dan belleğe, bellekten register'a ve bellekten belleğe için farklı talimatlar vardır. Buna karşılık, VAX'in taşınacak farklı veri miktarları için varyantları olan bir taşıma (MOV) talımı vardır, ancak bir işlenenin işlenenin bir parçası olarak kayıt mı yoksa bellek mi olduğunu belirtir. VAX yaklaşımı, uğraşacak daha az anımsatıcıya sahip olan programcı için biraz daha kolaydır. Bununla birlikte, IBM EAS/390 yaklaşımına göre biraz daha az derli topludur, çünkü her işlenenin konumu (yazmaç veya bellek) komutta ayrı olarak belirtilmelidir. Bölüm 13'te komut formatlarını tartışıırken bu ayırma geri doneceğiz.

İşlemci eylemi açısından, veri aktarım işlemleri belki de en basit türdür. Eğer hem kaynak hem de hedef ise, işlemci basitçe verinin bir yazmacından diğerine aktarılmasına neden olur; bu işlemci için dahili bir işlemidir. İşlenenlerden biri ya da her ikisi de bellekteyse, işlemci aşağıdaki eylemlerden bazılarını ya da tümünü gerçekleştirir

1. Adres moduna bağlı bellek adresini hesaplayın (Bölüm 13'te ele alınmıştır).
2. Adres sanal belleği ifade ediyorsa, sanal bellek adresinden gerçek bellek adresine çevirin.
3. Adreslenen ögenin önbellekte olup olmadığını belirleyin.
4. Değilse, bellek modülüne bir komut verin.

## Aritmetik

Çoğu makine toplama, çıkarma, çoklu katlama ve bölme gibi temel aritmetik işlemleri sağlar. Bunlar her zaman işaretli tamsayı (sabit nokta) sayıları için sağlanır. Genellikle kayan noktalı ve paketlenmiş ondalık sayılar için de sağlanırlar.

Diğer olası işlemler çeşitli tek-işlemli talimatları içerir; örneğin,

- **Mutlak:** İşlenenin mutlak değerini alır.
- **Negate:** İşleneni negatifleştirir.
- **Arttırma:** İşlenene 1 ekleyin.
- **Azaltma:** İşlenenden 1 çıkarır.

Bir aritmetik komutun yürütülmesi, ALU'ya giriş için işlenenleri konumlandırmak ve ALU'nun çıkışını sağlamak için veri aktarım işlemlerini içerebilir. Şekil 3.5 hem veri transferi hem de aritmetik işlemlerde yer alan hareketleri göstermektedir. Buna ek olarak, elbette, işlemcinin ALU kısmı istenen işlemi gerçekleştirir.

## Mantıksal

Çoğu makine ayrıca bir sözcüğün veya diğer adreslenebilir birimlerin tek tek bitlerini değiştirmek için genellikle "bit çevirme" olarak adlandırılan çeşitli işlemler sağlar. Bunlar Boolean işlemlerine dayanır (bkz. Bölüm 11).

Boolean veya ikili veriler üzerinde gerçekleştirilebilecek temel mantıksal işlemlerden bazıları Tablo 12.6'da gösterilmektedir. DEĞİL işlemi bir biti ters çevirir. AND, OR ve Exclusive-OR (XOR) iki operatörlü en yaygın mantıksal fonksiyonlardır. EQUAL kullanışlı bir ikili testtir.

Bu mantıksal işlemler n *bitlik* mantıksal veri birimlerine bitsel olarak uygulanabilir. Böylece, eğer iki kayıtçı veri içeriyorsa

$$(R1)=10100101$$

$$(R2)=00001111$$

sonra

$$(R1) \text{ VE } (R2)=00000101$$

**Tablo 12.6** Temel Mantıksal İşlemler

P	Q	P DEĞİL	P VE Q	P VEYA Q	P XOR Q	P=Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

Burada (X) gösterimi X konumunun içeriğini ifade eder. Böylece, AND işlemi bir kelimedeki belirli bitleri seçer ve kalan bitleri sıfırlayan bir *maske* olarak kullanılabilir. Başka bir örnek olarak, iki yazmaç şunları içeriyorsa

$$(R1)= 10100101$$

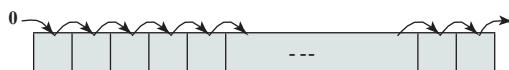
$$(R2)= 11111111$$

sonra

$$(R1) \text{ XOR } (R2)= 01011010$$

Bir kelime tüm 1'lere ayarlıken, XOR işlemi diğer kelimedeki tüm bitleri ters çevirir (birler tümleyen).

Bitsel mantıksal işlemlere ek olarak, çoğu makine kaydırma ve döndürme işlevleri sağlar. En temel işlemler Şekil 12.6'da gösterilmiştir. **Mantıksal kaydırma** ile bir sözcüğün bitleri sola veya sağa kaydırılır. Bir ucta, dışarı kaydırılan bit kaybolur. Diğer ucta, bir 0 içeri kaydırılır. Mantıksal kaydirmalar öncelikle bir kelime içindeki alanları izole etmek için kullanışlıdır. Bir sözcüğün içine kaydırılan 0'lar, diğer uçtan kaydırılan istemeyen bilgileri devre dışı bırakır.



(a) Mantıksal sağa kaydırma



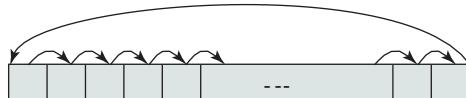
(b) Mantıksal sola kaydırma



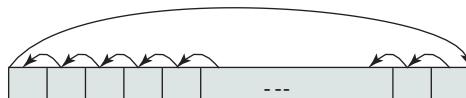
(c) Aritmetik sağa kaydırma



(d) Aritmetik sola kaydırma



(e) Sağda döndür



(f) Solda döndürme

**Şekil 12.6** Kaydırma ve Döndürme İşlemleri

Örnek olarak, bir I/O cihazına her seferinde 1 karakter olmak üzere veri karakterleri iletmek istediğimizi varsayalım. Her bellek sözcüğü 16 bit uzunluğundaysa ve iki karakter içeriyorsa, gönderilmeden önce karakterleri *paketinden* çıkarmamız gereklidir. İki karakteri bir kelime içinde göndermek için;

1. Sözcüğü bir yazmaca yükleyin.
2. Sekiz kez sağa kaydırın. Bu, kalan karakteri kaydın sağ yarısına kaydırır.
3. G/C gerçekleştirsin. G/C modülü veri yolundan alt sıradaki 8 biti okur.

Onceki adımlar sol taraftaki karakterin gönderilmesiyle sonuçlanır. Sağ taraftaki karakteri göndermek için;

1. Kelimeyi tekrar yazmacın içine yükleyin.
2. VE 00000001111111 ile. Bu, soldaki karakteri maskeler.
3. I/O gerçekleştirsin.

**Aritmetik kaydırma** işlemi veriyi işaretli bir tamsayı olarak ele alır ve işaret bitini kaydurmaz. Sağa aritmetik kaydırma, işaret biti bit konumuna çoğaltırlar. Sola aritmetik kaydırma, işaret biti hariç tüm bitlerde mantıksal bir sola kaydırma gerçekleştirilir ve bu bit korunur. Bu işlemler belirli aritmetik hızlandırılabilir. İkiye tümleyen gösterimindeki sayılarla, sağa aritmetik kaydırma, tek sayılar için kesmeyle birlikte 2'ye bölmeye karşılık gelir. Hem aritmetik sola kaydırma hem de mantıksal sola kaydırma, taşıma olmadığından 2 ile çarpma karşılık gelir. Taşma meydana gelirse, aritmetik ve mantıksal sola kaydırma işlemleri farklı sonuçlar üretir, ancak aritmetik sola kaydırma sayının işaretini korur. Taşma potansiyeli nedeniyle, PowerPC ve Itanium dahil olmak üzere birçok işlemci bu talimatı içermez. IBM EAS/390 gibi diğerleri ise bu talimatı sunmaktadır. İlginç bir şekilde, x86 mimarisinde aritmetik sola kaydırma içerir ancak bunu mantıksal sola kaydırma ile aynı olarak tanımlar.

**Döndürme** veya döngüsel kaydırma işlemleri, üzerinde işlem yapılan tüm bitleri korur. Döndürme işleminin bir kullanımı, her biti art arda en soldaki bitin içine getirmektir; burada verinin işaretini test edilerek tanımlanabilir (bir sayı olarak ele alınır).

Aritmetik işlemlerde olduğu gibi, mantıksal işlemler de ALU etkinliği içerir ve veri aktarım işlemlerini içerebilir. Tablo 12.7, bu alt bölümde tartışılan tüm kaydırma ve döndürme işlemlerinin örneklerini vermektedir.

**Tablo 12.7** Kaydırma ve Döndürme İşlemlerine Örnekler

Giriş	Operasyon	Sonuç
10100110	Mantıksal sağa kaydırma (3 bit)	00010100
10100110	Mantıksal sola kaydırma (3 bit)	00110000
10100110	Aritmetik sağa kaydırma (3 bit)	11110100
10100110	Aritmetik sola kaydırma (3 bit)	10110000
10100110	Sağ döndürme (3 bit)	11010100
10100110	Sola döndürme (3 bit)	00110101

## Dönüşüm

Dönüştürme talimatları, veri biçimini değiştiren veya veri biçimini üzerinde işlem yapan talimatlardır. Ondalıktan ikiliye dönüştürme buna bir örnektir. Daha karmaşık bir düzenleme talimatına örnek olarak EAS/390 Çeviri (TR) talimatı verilebilir. Bu komut 8 bitlik bir koddan diğerine dönüştürmek için kullanılabilir ve üç işlenen alır:

TR R1 (L), R2

R2 operandı, 8 bitlik kodlardan oluşan bir tablonun başlangıç adresini içerir. R1'de belirtilen adresten başlayan L baytları çevrilir, her bayt o bayt tarafından indekslenen bir tablo girişinin içeriği ile değiştirilir. Örneğin, EBCDIC'ten IRA'ya çevirmek için, önce 1000-10FF onaltılık gibi depolama konumlarında 256 baylıklı bir tablo oluşturuyoruz. Tablo, EBCDIC ikili gösteriminin sırasına göre IRA kodunun karakterlerini içerir; , IRA kodu tabloda aynı karakterin EBCDIC kodunun ikili değerine eşit görelî konuma yerleştirilir. Böylece, 10F0'dan 10F9'a kadar olan konumlar 30'dan 39'a kadar olan değerleri içerecektir, çünkü F0 0 rakamının EBCDIC kodudur ve 30 0 rakamının IRA kodudur ve bu şekilde 9 rakamına kadar devam eder. Şimdi 2100 konumundan başlayarak 1984 rakamları için EBCDIC koduna sahip olduğumuzu ve IRA'ya çevirmek istediğimizi varsayıyalım. Aşağıdakileri varsayıyalım:

- 2100-2103 konumları F1 F9 F8 F4 içerir.
- R1 2100 içerir.
- R2 1000 içerir.

Sonra, eğer çalıştırırsak

TR R1 (4), R2

2100-2103 konumları 31 39 38 34 içerecektir.

## Giriş/Çıkış

Giriş/çıkış talimatları Bölüm 7'de ayrıntılı olarak ele alınmıştır. Gördüğümüz gibi, izole programlanmış G/C, bellek eslemeli programlanmış G/C, DMA ve bir G/C işlemcisinin kullanımı dahil olmak üzere çeşitli yaklaşımlar benimsenmiştir. Birçok uygulama, parametreler, kodlar veya komut sözcükleriyle belirtilen belirli eylemlerle birlikte yalnızca birkaç G/C talimatı sağlar.

## Sistem Kontrolü

Sistem kontrol talimatları, yalnızca işlemci belirli bir ayrıcalıklı durumdayken veya belleğin özel bir ayrıcalıklı alanında bir program yürütülürken çalıştırılabilen talimatlardır. Tipik olarak, bu talimatlar işletim sisteminin kullanımı için ayrılmıştır.

Sistem kontrol işlemlerinin bazı örnekleri aşağıdaki gibidir. Bir sistem kontrol talımı bir kontrol kaydını okuyabilir ya da değiştirebilir; kontrol kayıtlarını Bölüm 14'te ele alacağız. Başka bir örnek, EAS/390 bellek sisteminde kullanıldığı gibi bir depolama koruma anahtarını okuma ya da değiştirme komutudur. Bir başka örnek de çoklu programlama sistemindeki işlem kontrol bloklarına erişimdir.

## Kontrolün Devri

Şimdiye kadar tartışılan tüm işlem türleri için, gerçekleştirilecek bir sonraki komut, bellekte mevcut komutu hemen takip eden komuttur. Ancak, herhangi bir programdaki komutların önemli bir kısmının işlevi komut yürütme sırasını değiştirmektir. Bu için işlemci tarafından gerçekleştirilen işlem, program sayacını bellekteki bir komutun adresini içerecek şekilde güncellemektedir.

Kontrol devri işlemlerinin gerekli olmasının birçok nedeni vardır. En önemlileri arasında aşağıdakiler yer almaktadır:

- 1.** Bilgisayarların pratik kullanımında, her bir komutu birden fazla kez ve belki de binlerce kez yürütüebilmek çok önemlidir. Bir uygulamayı hayatı geçirmek için binlerce ya da belki milyonlarca komut gerekebilir. Her bir komutun ayrı ayrı yazılması gerekseydi bu düşünülemezdi. Eğer bir tablo ya da öğeler listesi işlenecekse, bir program döngüsüne ihtiyaç vardır. Tüm verileri işlemek için bir dizi talimat tekrar yürütültür.
- 2.** Neredeyse tüm programlar bazı karar verme süreçlerini içerir. Bilgisayarın bir koşul gerçekleşirse bir şey, başka bir koşul gerçekleşirse başka bir şey yapmasını isteriz. Örneğin, bir dizi talimat bir sayının karekökünü hesaplar. Dizinin başlangıcında, sayının işaretini test edilir. Sayı negatifse, hesaplama gerçekleştirilmmez, ancak bir hata durumu bildirilir.
- 3.** Büyüük ve hatta orta ölçekli bir bilgisayar programını doğru bir şekilde oluşturmak son derece zor bir iştir. Görevi her seferinde üzerinde çalışılabilen daha küçük parçalara ayırmak için mekanizmalar varsa yardımcı olur.

Şimdi komut setlerinde bulunan en yaygın kontrol aktarım işlemlerinin tartışmasına geçiyoruz: **dallanma, atlama ve prosedür çağrıları**.

**BRANS TALİMATLARI** Atlama talimiği olarak da adlandırılan bir dallanma talimiği, işlenenlerinden biri olarak yürütülecek bir sonraki taliminin adresini içerir. Çoğu zaman, bu komut **koşullu bir dallanma** komutudur. Yani, yalnızca belirli bir koşul karşılandığında dallanma yapılır (program sayacı işlenende belirtilen adrese eşit olacak şekilde güncellenir). Aksi takdirde, sıradaki bir sonraki komut yürütülür (program sayacı her gibi artırılır). Dallanmanın her zaman yapıldığı bir dallanma komutu **koşulsuz bir dallanmadır**.

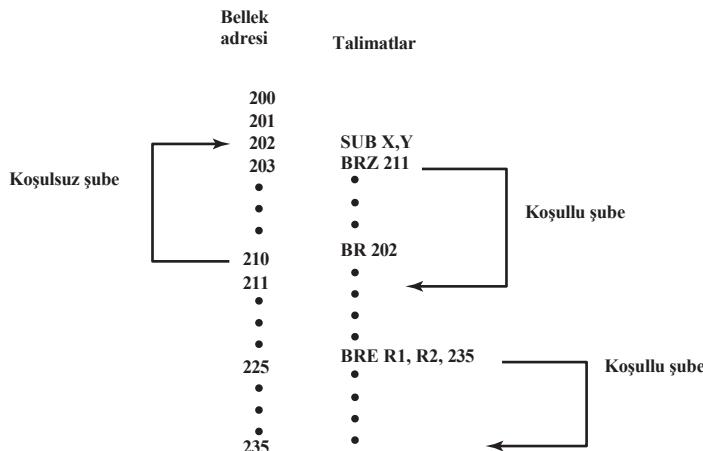
Bir koşullu dallanma komutunda test edilecek koşulu oluşturmanın iki yaygın yolu vardır. Birincisi, çoğu makine bazı işlemlerin sonucu olarak ayarlanan 1 bitlik veya çok bitlik bir koşul kodu sağlar. Bu kod, kullanıcı tarafından görülebilen kısa bir kayıt olarak düşünülebilir. Örnek olarak, bir aritmetik işlem (ADD, SUBTRACT, vb.) aşağıdaki dört değerden birine sahip 2 bitlik bir koşul kodu ayarlayabilir: 0, pozitif, negatif, taşıma. Böyle bir makinede, dört farklı koşullu dallanma talimiği olabilir:

BRP X Sonuç pozitifse X konumuna dallanma. BRN X

Sonuç negatifse X konumuna dallanma. BRZ X Sonuç

sıfırsa X konumuna dallanma.

BRO X Taşma meydana gelirse X konumuna dallanma.



Şekil 12.7 Dallanma Komutları

Tüm bu durumlarda, atıfta bulunulan sonuç, koşul kodunu belirleyen en son sonucudur. Üç adresli komut formatıyla kullanılabilen bir başka yaklaşım da aynı bir karşılaştırma yapmak ve bir dallanma belirtmektedir. Örneğin,

**BRE R1, R2, X** R1'in içeriği= R2'nin içeriği ise X'e dallanır.

Şekil 12.7 bu işlemlerin örneklerini göstermektedir. Bir dallanmanın *ileriye* (daha yüksek adresli bir komut) ya da *geriye* (daha düşük adresli) doğru olabileceğine dikkat edin. Örnek, tekrar eden bir komut döngüsü oluşturmak için koşulsuz ve koşullu dallanmanın nasıl kullanılabileceği göstermektedir. 202'den 210'a kadar olan konumlardaki talimatlar, Y'nin X'ten çıkarılmasının sonucu 0 olana kadar tekrar yürütülecektir.

**ATLAMA TALİMATLARI** Kontrol aktarım talimatının bir başka şekli de atlama talimatıdır. Atlama talimi bir zımnı adres içerir. Tipik olarak, atlama bir komutun atlanacağını ima eder; bu nedenle, ima edilen adres bir sonraki komutun adresi artı bir komut uzunluğuna eşittir. Atlama komutu bir hedef adres alanı gerektirmeden, başka şeyler yapmakta serbesttir. Tipik bir örnek, sıfırda artı ve atla (ISZ) komutudur. Aşağıdaki program parçasını düşünün:

```

301
f
309 ISZ R1
310 BR 301
311

```

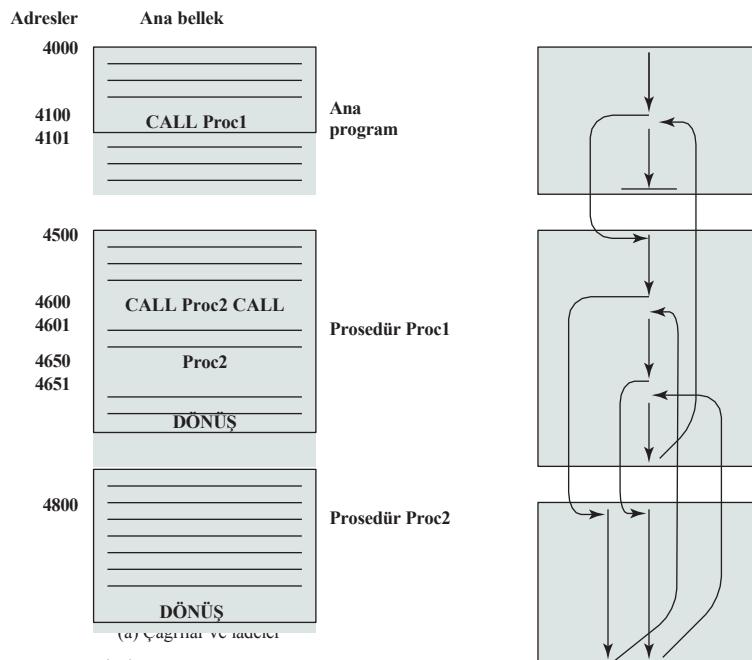
Bu parçada, iki kontrol aktarımı talimi yinelemeli bir döngü uygulamak için kullanılır. R1, gerçekleştirilecek iterasyon sayısının negatif ile ayarlanır. Döngünün sonunda R1 artırılır. Eğer 0 değilse, program döngünün başına geri dallanır. Aksi takdirde, dallanma atlanır ve program döngünün sonundan sonraki komutla devam eder.

**PROCEDURE cALL INSTRUCTIONS** Programlama dillerinin gelişimindeki belki de en önemli yenilik *prosedürdür*. Yordam, daha büyük bir programa dahil edilen, kendi kendine yeten bir bilgisayar programıdır. Programın herhangi bir noktasında prosedür çağrılabılır veya çağrırlabilir. İşlemciye gidip tüm prosedürü çalıştırması ve ardından çağrının gerçekleştiği noktaya geri dönmesi talimatı verilir.

Yordamların kullanılmasının iki temel nedeni ekonomi ve modülerliktir. Bir prosedür, aynı kod parçasının birçok kez kullanılmasına olanak tanır. Bu, programlama çabasında ekonomi ve sistemdeki depolama alanının en verimli şekilde kullanılması için önemlidir (program saklanmalıdır). Prosedürler ayrıca büyük programlama görevlerinin daha küçük birimlere bölünmesini sağlar. Bu *modülerlik* kullanımı programlama görevini büyük ölçüde kolaylaştırır.

Yordam mekanizması iki temel yönerge içerir: mevcut konumdan yordama dallanan bir çağrı ve yordamdan çağrıldığı yere geri dönen bir geri dönüş yönergesi. Bunların her ikisi de dallanma talimatlarıdır.

Şekil 12.8a, bir program oluşturmak için prosedürlerin kullanımını göstermektedir. Bu örnekte, 4000 konumunda başlayan bir ana program vardır. Bu program, 4500 konumunda başlayan PROC1 prosedürüne bir çağrı içerir. Bu çağrı komutuyla karşılaşıldığında, işlemci ana programın yürütülmesini askıya alır ve 4500 konumundan bir sonraki komutu getirerek PROC1'in yürütülmesine başlar. PROC1 içinde, 4800 konumunda PROC2'ye iki çağrı vardır. Her durumda, PROC1'in yürütülmesi



Şekil 12.8 İç İçe Prosedürler

askıya alınır ve PROC2 yürütülür. RETURN deyimi, işlemcinin çağrıran programa geri dönmesine ve ilgili CALL komutundan sonraki komutta yürütmeye devam etmesine neden olur. Bu davramış Şekil 12.8b'de gösterilmiştir.

Üç nokta kayda değerdir:

1. Bir prosedür birden fazla konumdan çağrılabılır.
2. Bir prosedür çağrısı bir prosedürün içinde görünebilir. Bu, prosedürlerin keyfi bir derinlige kadar iç içe geçmesine izin verir.
3. Her prosedür çağrısı, çağrılan programda bir geri dönüş ile eşleştirilir.

Bir yordamı çeşitli noktalardan çağrılmak istedigimizden, işlemcinin dönüş adresini bir şekilde kaydetmesi gerekir, böylece dönüş uygun şekilde gerçekleştirilebilir. Dönüş adresini saklamak için üç yaygın yer vardır:

- Kayıt Olun
- Çağrılan prosedürün başlangıcı
- Yiğinin üstü

*X konumundaki çağrı* yordamı anlamına gelen CALL X makine dili yönergesini düşünün. Kayıt yaklaşımı kullanılırsa, CALL X aşağıdaki eylemlere neden olur:

RN **d** PC +Δ  
PC **d** X

Burada RN her zaman bu amaç için kullanılan bir yazmaç, PC program sayacı ve Δ komut uzunluğudur. Çağrılan prosedür artık daha sonra geri dönüş için kullanılmak üzere RN'nin içeriğini kaydedebilir.

İkinci bir olasılık, dönüş adresini prosedürün başlangıcında saklamaktır. Bu durumda, CALL X aşağıdakilere neden olur

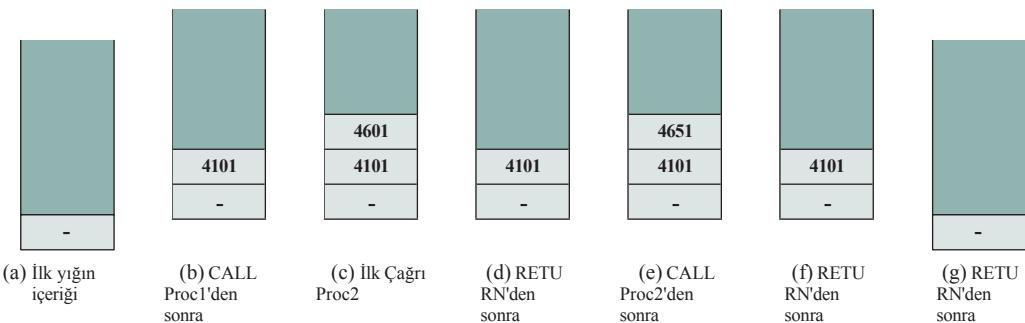
X **d** PC +Δ  
PC **d** X+1

Bu oldukça kullanışlı. İade adresi güvenli şekilde saklandı.

Onceki yaklaşımların her ikisi de işe yarar ve kullanılmıştır. Bu yaklaşımların tek sınırlaması, *tekrarlı* kullanımını zorlaştırmalarıdır. Yinelemeli bir yordam, aynı anda birden fazla çağrıının açık olmasının mümkün olduğu bir **yordamdır**. Özyinelemeli bir yordam (kendi kendini çağrıran bir yordam) bu özelliğin kullanımına bir örnektir (bkz. Ek M). Yinelemeli bir için parametreler yazmaçlar ya da bellek aracılığıyla aktarılıyorsa, yazmaçların ya da bellek alanının diğer yordam çağrıları için kullanılabilir olması için bazı kodların parametreleri kaydetmekten sorumlu olması gereklidir.

Daha genel ve güçlü bir yaklaşım ise yiğin kullanmaktadır (ilgili tartışma için Ek I'e bakınız). İşlemci bir çağrıyı yürütüğünde, dönüş adresini yiğine yerleştirir. Bir geri dönüş gerçekleştirdiğinde, yiğindaki adresi kullanır. Şekil 12.9 yiğinin kullanımını göstermektedir.

Bir dönüş adresi sağlanmanın yanı sıra, genellikle bir yordam çağrısı ile parametrelerin aktarılması da gereklidir. Bunlar içinde geçirilebilir. Bir başka olasılık da parametreleri CALL komutundan hemen sonra bellekte saklamaktır. Bu durumda, dönüş parametreleri takip eden konuma olmalıdır. Yine, her iki

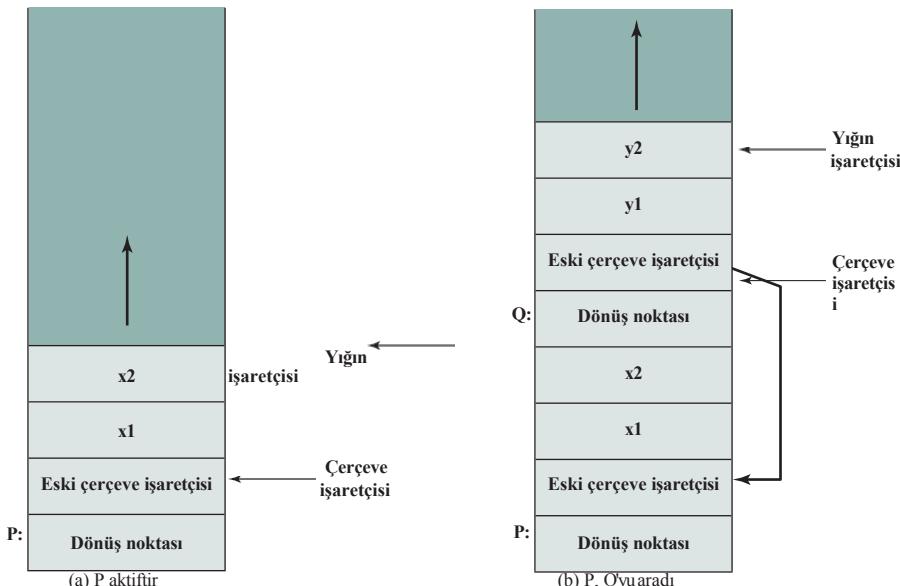


**Şekil 12.9** Şekil 12.8'deki İç İçe Alt Programları Uygulamak için Yığın Kullanımı

bu yaklaşımın dezavantajları vardır. Eğer yazmaçlar kullanılıyorsa, çağrılan program ve çağrıran program yazmaçların doğru kullanıldığından emin olmak için yazılmalıdır. Parametrelerin bellekte saklanması, değişken sayıda parametre değişimini zorlaştırır. Her iki yaklaşım da yeniden girişli prosedürlerin kullanımını engeller.

Parametre geçisi için daha esnek bir yaklaşım yiğittir. Programlayıcı bir çağrıyı yürütüğünde, yalnızca dönüş adresini değil, çağrılan yordama aktarılacak parametreleri de yiğinler. Çağrılan prosedür parametrelere yiğinden erişebilir. Geri dönüste, geri dönüş parametreleri de yiğine yerleştirilebilir. Bir yordam çağrısi için saklanan, dönüş adresi de dahil olmak üzere tüm parametre kümesi *yığın çerçevesi* olarak adlandırılır.

Şekil 12.10'da bir örnek verilmiştir. Örnek,  $x_1$  ve  $x_2$  yerel değişkenlerinin bildirildiği P yordamına ve P'nin çağrılabileceği ve  $y_1$  ve  $y_2$  yerel değişkenlerinin bildirildiği Q yordamına atıfta bulunmaktadır. Bu şekilde, geri dönüş



**Şekil 12.10** Örnek Prosedürler P ve Q Kullanılarak Yığın Çerçevesinin Büyütülmesi

Her prosedür için nokta, ilgili yiğin çerçevesinde saklanan ilk ögedir. Daha sonra, önceki çerçevenin başlangıcına bir işaretçi saklanır. Bu, istiflenecek parametrelerin sayısı veya uzunluğu değişkense gereklidir.

## 12.5 INTEL x86 VE ARM İŞLEM TÜRLERİ

### x86 İşlem Türleri

x86, bir dizi özel talimat da dahil olmak üzere karmaşık bir dizi işlem sağlar. Amaç, yüksek seviyeli dil programlarının optimize edilmiş makine dili çevirisini üretmek için derleyici yazarına araçlar sağlamak. Bunların çoğu makine komut setinde bulunan geleneksel talimatlardır, ancak birkaç talimat türü x86 mimarisine uyarlanmıştır ve özellikle ilgi çekicidir. CART06] Ek A, x86 komutlarını, her biri için işlem ve komutun durum kodları üzerindeki etkisi ile birlikte listeler. NASM montaj dili kılavuzunun [NASM12] Ek B'si her x86 komutunun daha ayrıntılı bir tanımını sağlar. Her iki belge de box.com/COA10e adresinde mevcuttur.

**CALL/RETURN TALİMATLARI** x86, prosedür çağrısını/geri dönüşünü desteklemek için dört talimat sağlar: CALL, ENTER, LEAVE, RETURN. Bu komutlar tarafından sağlanan desteği bakmak öğretici olacaktır. Şekil 12.10'dan, yordam çağrıma/döndürme mekanizmasını uygulamanın yaygın bir yolunun yiğin çerçeveleri kullanmak olduğunu hatırlayın. Yeni bir prosedür çağrılığında, yeni prosedüre girişte aşağıdakiler gerçekleştirilmelidir:

- Dönüş noktasını yiğina itin.
- Geçerli çerçeve işaretçisini yiğina iter.
- Yiğin işaretçisini çerçeve işaretçisinin yeni değeri olarak kopyalayın.
- Bir çerçeve ayırmak için yiğin işaretçisini ayarlayın.

CALL komutu, geçerli komut işaretçisi değerini yiğina iter ve giriş noktasının adresini komut işaretçisine yerleştirerek prosedürün giriş noktasına atlamaya neden olur. 8088 ve 8086 makinelerinde, tipik prosedür şu diziyle başlar

İTME	EBP
MOV	EBP, ESP
SUB	ESP, space_for_locals

Burada EBP çerçeve işaretçisi ve ESP yiğin işaretçisidir. 80286 ve sonraki makinelerde, ENTER komutu yukarıda bahsedilen tüm işlemleri tek bir komutta gerçekleştirir.

ENTER komutu, derleyiciye doğrudan destek sağlamak için komut setine eklenmiştir. Bu komut ayrıca Pascal, COBOL ve Ada gibi dillerde (C veya FORTAN'da bulunmayan) iç içe prosedürler olarak adlandırılan işlemlerin desteklenmesi için bir özellik içermektedir. Bu diller için iç içe geçmiş prosedür çağrılarını ele almanın daha iyi yolları olduğu ortaya çıkmıştır. Ayrıca, ENTER komutuna rağmen

PUSH, MOV, SUB dizisine kıyasla birkaç bayt bellek tasarrufu sağlar (4 bayta karşı 6 bayt), aslında yürütülmesi daha uzun sürer (10 saat döngüsüne karşı 6 saat döngüsü). Bu nedenle, komut seti tasarımcılarına bu özgürlüğü eklemek iyi bir fikir gibi görünmüştür olsa da, çok az fayda sağlarken ya da hiç fayda sağlamazken işlemcinin uygulanmasını karmaşıklaştırır. Bunun aksine, işlemci tasarımasına yönelik bir RISC yaklaşımının ENTER gibi karmaşık komutlardan kaçınacağını ve daha basit bir dizi komutla daha verimli bir uygulama üretebileceğini göreceğiz.

**BELLEK YÖNETİMİ** Bir başka özel talimatlar kümesi de bellek böülümlendirmesi ile ilgilidir. Bunlar yalnızca işletim sisteminden çalıştırılabilen ayrıcalıklı talimatlardır. Yerel ve global segment tablolarının (tanımlayıcı tablolar olarak adlandırılır) yüklenmesine ve okunmasına ve bir segmentin ayrıcalık seviyesinin kontrol edilmesine ve değiştirilmesine izin verirler.

Üstü önbellek ile ilgili özel talimatlar Bölüm 4'te ele alınmıştır.

**DURUM BAYRAKLARI VE KOŞUL KODLARI** Durum bayrakları, belirli işlemler tarafından ayarlanabilen ve koşullu dallanma talimatlarında kullanılan özel kayıtlardaki bitlerdir. *Koşul kodu* terimi, bir veya daha fazla durum bayrağının ayarlarını ifade eder. X86 ve diğer birçok mimaride, durum bayrakları aritmetik ve karşılaştırma işlemleri tarafından ayarlanır. Çoğu dilde karşılaştırma işlemi, çıkarma işleminde olduğu gibi iki işleneni çıkarır. Aradaki fark, karşılaştırma işleminin yalnızca durum bayraklarını ayarlaması, çıkarma işleminin ise çıkarma işleminin sonucunu hedef işlenende saklamasıdır. Bazı mimariler veri aktarım talimatları için de durum bayrakları ayarlar.

Tablo 12.8'de x86'da kullanılan durum bayrakları listelenmiştir. Her bayrak veya bu bayrakların kombinasyonları, koşullu bir atlama için test edilebilir. Tablo 12.9, koşullu atlama işlem kodlarının tanımlandığı koşul kodlarını (durum bayrağı değerlerinin kombinasyonları) göstermektedir.

Bu liste hakkında birkaç ilginç gözlem yapılabilir. İlk olarak, bir sayının diğerinden daha büyük olup olmadığını belirlemek için iki işleneni test etmek isteyebiliriz. Ancak bu, sayıların işaretli veya işaretsiz olmasına bağlı olacaktır. Örneğin, 8 bitlik 11111111 sayısı, iki sayı yorumlanması 00000000 sayısından daha büyuktur

**Tablo 12.8** x86 Durum Bayrakları

Durum Biti	İsim	Açıklama
C	Taşıma	Bir aritmetik işlemin ardından en soldaki bit konumunun taşınmasını veya ödünc alınmasını gösterir. Bazı kaydırma ve döndürme işlemleri tarafından da değiştirilir.
P	Parite	Bir aritmetik veya mantık işleminin sonucunun en az anlamlı baytinın eşlik derecesi. 1 çift pariteyi gösterir; 0 tek pariteyi gösterir.
A	Yardımcı Taşıma	8-bitlik bir aritmetik veya mantık işleminin yarımbaytları arasında taşıma veya ödünc almayı temsil eder. İkili kodlu ondalık aritmetikte kullanılır.
Z	Sıfır	Bir aritmetik veya mantık işleminin sonucunun 0 olduğunu gösterir.
S	İşaret	Bir aritmetik veya mantık işleminin sonucunun işaretini belirtir.
O	Taşma	İkili tümleyen aritmetiği için bir toplama veya çıkarma işleminden sonra bir aritmetik taşıma olduğunu gösterir.

Tablo 12.9 x86 Koşullu Jump ve SETcc Komutları için Durum Kodları

Sembol	Test Edilen Durum	Yorum
A, NBE	C= 0 VE Z= 0	Üstünde; Altında veya eşit değil (daha büyük, işaretsiz)
AE, NB, NC	C= 0	Üstünde veya eşit; Altında değil (büyük veya eşit, işaretsiz); Taşımaz
B, NAE, C	C= 1	Altında; Üstünde veya eşit değil (az, işaretsiz); Taşıma kümesi
BE, NA	C= 1 VEYA Z= 1	Altında veya eşit; Üstünde değil (daha az veya eşit, işaretsiz)
E, Z	Z= 1	Eşit; Sıfır ( işaretli veya işaretsiz)
G, NLE	[(S= 1 VE O= 1) VEYA (S= 0 VE O= 0)]VE[Z= 0]	büyük; Daha az veya eşit değil ( işaretli)
GE, NL	(S= 1 VE O= 1) VEYA (S= 0 VE O= 0)	Daha büyük veya eşit; Daha az değil ( işaretli)
L, NGE	(S= 1 VE O= 0) VEYA (S= 0 VE O= 0)	az; Daha büyük veya eşit değil ( işaretli)
LE, NG	(S= 1 VE O= 0) VEYA (S= 0 VE O= 1) VEYA (Z= 1)	Daha az veya eşit; Daha büyük değil ( işaretli)
NE, NZ	Z= 0	Eşit değil; Sıfır değil ( işaretli veya işaretsiz)
HAYIR	O= 0	Taşma yok
NS	S= 0	İşaret değil (negatif değil)
NP, PO	P= 0	Parite değil; Parite tek
O	O= 1	Taşma
P	P= 1	Parite; Parite çift
S	S= 1	İşaret (negatif)

İşaretsiz tamsayılar olarak (255 7 0), ancak 8 bitlik ikişerli birleşik sayılar (- 1 6 0) olarak kabul edilirlerse daha azdır. Bu nedenle birçok assembly dili iki durumu birbirinden ayırmak için iki terim seti sunar: Eğer iki sayıyı işaretli tamsayılar olarak karşılaştırıyorsak, *ktiçüktür* ve terimlerini kullanırız; eğer onları işaretsiz tamsayılar olarak karşılaştırıyorsak, *altındadır* ve *üstündedir* terimlerini kullanırız.

İkinci bir gözlem, işaretli tamsayıları karşılaştırmanın karmaşaklılığıyla ilgilidir. İşaretli bir sonuç, (1) işaret biti sıfırsa ve taşıma yoksa (S= 0 AND O= 0) veya (2) işaret biti birse ve taşıma varsa sıfırdan büyük veya sıfıra eşittir. Şekil 10.4'ün incelenmesi, çeşitli işaretli işlemler için test edilen koşulların uygun olduğu konusunda sizin ikna etmelidir.

**X86 SIMD INSTRUCTIONS** 1996 yılında Intel, Pentium ürün serisine MMX teknolojisini tanıttı. MMX, multimedya görevleri için son derece optimize edilmiş komutlar kümesidir. Verileri SIMD (tek komut, çoklu veri) biçiminde işleyen 57 yeni komut vardır, bu da toplama veya çarpma gibi aynı işlemi aynı anda birden fazla veri öğesi üzerinde gerçekleştirmeyi mümkün kılar. Her komutun yürütülmesi genellikle tek bir saat döngüsü alır. Uygun uygulama için, bu hızlı paralel işlemler MMX talimatlarını kullanmayan karşılaştırılabilir algoritmalarla göre iki ila sekiz kat hızlanma sağlayabilir [ATKI96]. Intel, 64-bit x86 mimarisinin piyasaya sürülmemesiyle birlikte bu uzantısı çift

dört kelimelik (128 bit) işlenenler ve kayan nokta işlemleri. Bu alt bölümde MMX özelliklerini açıklayacağız.

MMX'in odak noktası multimedya programlamadır. Video ve ses verileri tipik olarak 8 veya 16 bit gibi küçük veri türlerinin büyük dizilerinden, geleneksel talimatlar 32 veya 64 bit veriler üzerinde çalışacak şekilde uyarlanmıştır. İşte bazı örnekler: Grafik ve videoda, tek bir sahne bir dizi pikselden oluşur<sup>2</sup> ve her piksel için 8 bit veya her piksel renk bileşeni (kırmızı, yeşil, mavi) için 8 bit vardır. Tipik ses örnekleri 16 bit kullanılarak kuantize edilir. Bazı 3D grafik algoritmalarında, temel veri türleri için 32 bit yaygındır. Bu veri uzunlukları üzerinde paralel çalışma sağlamak için MMX'te üç yeni veri tipi tanımlanmıştır. Her veri tipi 64 bit uzunluğundadır ve biri sabit noktalı bir tamsayı tutan birden fazla küçük veri alanından oluşur. Tipler aşağıdaki gibidir:

- **Paketlenmiş bayt:** Sekiz bayt bir 64 bitlik miktar içine paketlenmiştir.
- **Paketlenmiş kelime:** 64 bit içine paketlenmiş dört adet 16 bitlik sözcük.
- **Paketlenmiş çift sözcük:** 64 bit içine paketlenmiş iki 32 çift sözcük.

Tablo 12.10 MMX komut kümelerini listeler. Komutların çoğu baytlar, sözcükler veya çift sözcükler üzerinde paralel işlem içerir. Örneğin, PSLLW komutu, paketlenmiş kelime işlenenindeki dört kelimenin her birinde ayrı ayrı bir sol mantıksal kaydırma gerçekleştirir; PADDB komutu, paketlenmiş bayt işlenenlerini girdi olarak alır ve paketlenmiş bayt çıktılarını üretmek için her bayt konumunda bağımsız olarak paralel toplama işlemi gerçekleştirir.

Yeni komut setinin alışılmadık bir özelliği, bayt ve 16 bitlik kelime işlenenleri için **aritmetiğinin** getirilmesidir. Sıradan işaretsiz aritmetikte, bir işlem taşlığında (yani, en anlamlı bitin dışına taşıldığında), fazladan bit kesilir. Buna sarma adı verilir, çünkü kesme işleminin etkisi, örneğin iki giriş işleneninden daha küçük bir toplama sonucu üretmek olabilir. Onaltılık olarak F000h ve 3000h şeklinde iki kelimenin toplandığını düşünün. Toplam şu şekilde ifade edilir

$$\begin{aligned} \text{F000h} &= 1111\ 0000\ 0000\ 0000 \\ +\ 3000h &= \underline{0011\ 0000\ 0000\ 0000} \\ &\quad 10010\ 0000\ 0000\ 0000 = 2000h \end{aligned}$$

İki sayı görüntü yoğunluğunu temsil ediyorsa, toplama işleminin sonucu iki koyu tonun kombinasyonunun daha açık hale gelmesidir. Bu genellikle amaçlanan şey değildir. Doygunluk aritmetiğinde, toplama işlemi taşıma ile sonuçlanırsa veya çıkarma işlemi taşıma ile sonuçlanırsa, sonuç temsil edilebilen en büyük veya en küçük değere ayarlanır. Yukarıdaki örnek için, doygunluk aritmetiği ile

$$\begin{aligned} \text{F000h} &= 1111\ 0000\ 0000\ 0000 \\ +\ 3000h &= \underline{0011\ 0000\ 0000\ 0000} \\ &\quad 10010\ 0000\ 0000\ 0000 \\ &\quad 1111\ 1111\ 1111\ 1111 = \text{FFFFh} \end{aligned}$$

---

<sup>2</sup>Piksel veya resim ögesi, dijital bir görüntünün gri bir seviye atanabilen en küçük ögesidir. Eşdeğer olarak, bir piksel, bir resmin nokta matrisindeki tek bir noktadır.

## 442 BÖLÜM 12 / YÖNERGE SETLERİ: ÖZELLİKLER VE İŞLEVLER

**Tablo 12.10** MMX Komut Kümesi

Kategori	Talimatlar	Açıklama
Aritmetik	PADD [B, W, D]	Paketlenmiş sekiz bayt, dört 16 bit kelime veya iki 32 bit çift kelimenin sarma ile paralel eklenmesi.
	PADDS [B, W]	Doygunluk ile ekleyin.
	PADDUS [B, W]	Doygunluk ile işaretetsiz ekleyin.
	PSUB [B, W, D]	Sarma ile çıkarma.
	PSUBS [B, W]	Doygunluk ile çıkarın.
	PSUBW [B, W]	Doygunluk ile işaretetsiz çıkarma.
	PMULHW	Dört işaretelli 16 bitlik kelimenin paralel çarpımı, 32 bitlik sonucun yüksek dereceli 16 biti seçilerek.
	PMULLW	Dört işaretelli 16 bitlik kelimenin paralel çarpımı, 32 bitlik sonucun düşük sıralı 16 biti seçilerek.
	PMADDWD	Dört işaretelli 16 bitlik kelimenin paralel çarpımı; 32 bitlik sonuçların bitişik çiftlerini bir araya getirin.
Karşılaştırma	PCMPEQ [B, W, D]	Eşitlik için paralel karşılaştırma; sonuç doğrusa 1'lerin, yanlışsa 0'ların maskesidir.
	PCMPGT [B, W, D]	Büyükür için paralel karşılaştırma; sonuç doğrusa 1'lerden, yanlışsa 0'lardan oluşan maskedir.
Dönüşüm	PACKUSWB	İşaretsiz doygunluk ile kelimeleri baytlara paketleyin.
	PACKSS [WB, DW]	İşaretli doygunluk ile sözcükleri baytlara veya çift sözcükleri sözcüklerle paketleyin.
	PUNPCKH [BW, WD, DQ]	MMX yazmacından yüksek sıralı baytları, sözcükleri veya çift sözcükleri paralel açma (serpiştirilmiş birleştirme).
	PUNPCKL [BW, WD, DQ]	MMX yazmacından düşük sıralı baytları, sözcükleri veya çift sözcükleri paralel açma (serpiştirilmiş birleştirme).
Mantıksal	PAND	64 bit bitsel mantıksal AND
	PNDN	64 bit bitsel mantıksal VE DEĞİL
	POR	64 bit bitsel mantıksal VEYA
	PXOR	64 bitlik bitsel mantıksal XOR
Vardiya	PSLL [W, D, Q]	Paketlenmiş sözcüklerin, çift sözcüklerin veya dört sözcüğün MMX kaydında veya anlık değerde belirtilen miktar kadar paralel mantıksal sola kaydırılması.
	PSRL [W, D, Q]	Paketlenmiş sözcüklerin, çift sözcüklerin veya dört sözcüğün paralel mantıksal sağa kaydırılması.
	PSRA [W, D]	Paketlenmiş sözcüklerin, çift sözcüklerin veya dört sözcüğün paralel aritmetik sağa kaydırılması.
Veri aktarımı	MOV [D, Q]	Çift sözcüğü veya dört sözcüğü MMX yazmacından taşır.
Açıklama	EMMS	Boş MMX durumu (boş FP kayıtları etiket bitleri).

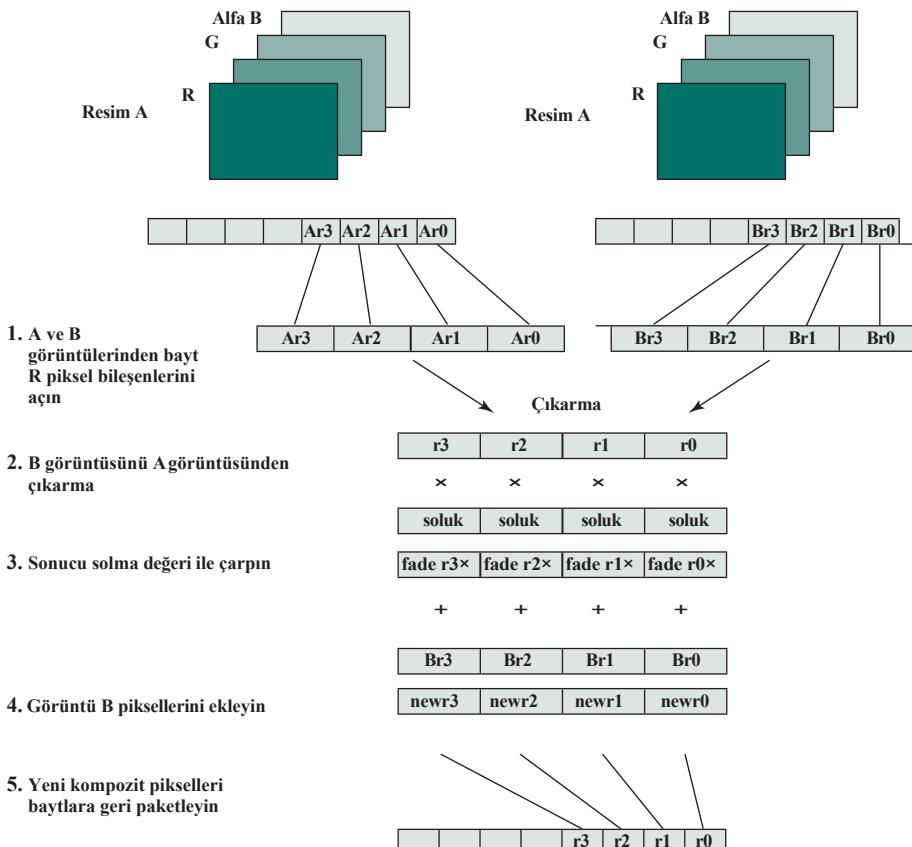
*Not:* Bir komut birden fazla veri türünü [byte (B), word (W), doubleword (D), quadword (Q)] destekliyorsa, veri türleri parantez içinde belirtilir.

MMX komutlarının kullanımını hakkında bir fikir vermek için [PELE97]'den alınan bir örneğe bakalım. Yaygın bir video uygulaması, bir sahnenin yavaş yavaş diğerine dönüştüğü fade-out, fade-in efektidir. İki görüntü ağırlıklı bir ortalama ile birlleştirilir:

$$\text{Result\_pixel} = \text{A\_pixel} * \text{fade} + \text{B\_pixel} * (1 - \text{fade})$$

Bu hesaplama A ve B'deki her piksel konumunda gerçekleştirilir. Solma değeri kademeli olarak 1'den 0'a değiştirilirken (8 bitlik bir tamsayı için uygun şekilde ölçeklendirilir) bir dizi video karesi üretilirse, sonuç A görüntüsünden B görüntüsüne solmaktadır.

Şekil 12.11'de bir piksel kümlesi için gereken adımların sırası gösterilmektedir. 8 bit piksel bileşenleri, MMX 16 bit çarpma özelliğine uyum sağlamak için 16 bit öğelere dönüştürülür. Bu görüntüler 640 \* 480 çözünürlük kullanıyorsa ve çözme tekniği solma değerinin tüm 255 olası değerini kullanıyorsa, toplam



Bu işlemi gerçekleştiren MMX kod dizisi:

```

pxor      mm7, mm7    ;mm7'yi sıfırlayın
movq      mm3, fad_val ;4 kez çoğaltılan solma değerini yükle
movd      mm0, imageA ;A görüntüsünden 4 kırmızı piksel bileşeni yükleyin
movd      mm1, imageB ;B görüntüsünden 4 kırmızı piksel bileşeni yükleyin
punckblw mm0, mm7    ;4 pikseli 16 bite aç
punckblw mm1, mm7    ;4 pikseli 16 bite aç psubw mm0,
mm1      ;A görüntüsünden B görüntüsünü çıkarma
pmulhw   mm0, mm3    ;çıkarma sonucunu soluk değerlerle çarpın
paddw    mm0, mm1    ;sonucu B görüntüsüne ekleyin
packuswb mm0, mm7    ;16 bitlik sonuçları baytlara geri paketleyin

```

**Şekil 12.11** Renk Düzlemi Temsilinde Görüntü Birleştirme

MMX kullanılarak yürütülen komut sayısı 535 milyondur. Aynı hesaplama MMX komutları olmadan yapıldığında 1,4 milyar komut yürütülmesi gereklidir [INTE98].

## ARM İşlem Türleri

ARM mimarisinin geniş bir işlem türü koleksiyonu sağlar. Aşağıda başlıca kategoriler yer almaktadır:

- **Yükle ve sakla talimatları:** ARM mimarisinde, yalnızca yükleme ve saklama komutları bellek konumlarına erişir; aritmetik ve mantıksal komutlar yalnızca yazmaçlar ve komutta kodlanan anlık değerler üzerinde gerçekleştirilir. Bu sınırlama RISC tasarıının karakteristik özelliğidir ve Bölüm 15'te daha ayrıntılı olarak incelenmiştir. ARM mimarisinin tek bir ya da bir çift değerini bellekten ya da belleğe yükleyen ya da saklayan iki geniş komut türünü destekler: (1) 32 bitlik bir sözcüğü ya da 8 bitlik işaretetsiz bir baytı yüklemek ya da saklamak ve (2) 16 bitlik işaretetsiz bir yarımsözcüğü yüklemek ya da saklamak ve 16 bitlik bir yarımsözcüğü ya da 8 bitlik bir baytı yüklemek ve işaretlemek.
- **Dallanma talimatları:** ARM, 32 MB'ye kadar ileriye veya geriye doğru koşullu dallanmaya izin veren bir dallanma talimatını destekler. Bir alt rutin çağrısı, standart dallanma komutunun bir varyantı ile gerçekleştirilebilir. Bağlantılı Dallanma (BL) komutu, 32 MB'a kadar ileri veya geri dallanmaya izin vermesinin yanı sıra, dallanmadan sonraki komutun adresini (dönüş adresi) LR'de (R14) korur. Dallanmalar, komuttaki 4 bitlik bir koşul alanı tarafından belirlenir.
- **Veri işleme talimatları:** Bu kategori mantıksal talimatları (AND, OR, XOR), toplama ve çıkarma talimatlarını ve test ve karşılaştırma talimatlarını içerir.
- **Çarpma talimatları:** Tamsayı çarpma komutları sözcük veya yarımsözcük işlenenler üzerinde çalışır ve normal veya uzun sonuçlar üretilebilir. Örneğin, 32 bitlik iki işlenen alan ve 64 bitlik bir sonuç üreten bir çarpma talımı vardır.
- **Paralel toplama ve çıkarma talimatları:** Normal veri işleme ve çarpma talimatlarına ek olarak, iki işlenenin bölümlerinin paralel olarak çalıştırıldığı bir dizi paralel toplama ve çıkarma talımı vardır. Örneğin, ADD16 sonucun üst yarımsözcüğünü oluşturmamak için iki kaydırın üst yarımsözcüklerini toplar ve sonucun alt yarımsözcüğünü oluşturmamak için aynı iki kaydırın alt yarımsözcüklerini toplar. Bu talimatlar, x86 MMX talimatlarına benzer şekilde görüntü işleme uygulamalarında kullanılmışlardır.
- **Genişletme talimatları:** Verileri işaret veya sıfır ile açmak için baytları yarımsözcüklerde veya kelimeye ve yarımkelimelerde genişleten birkaç talimat vardır.
- **Durum kaydı erişim talimatları:** ARM, durum kaydırının bazı bölümlerini okuma ve yazma olanağı sağlar.

**KOŞUL KODLARI** ARM mimarisinin program durum kaydırında saklanan dört koşul bayrağı tanımları: N, Z, C ve V (Negative, Zero, Carry ve oOverflow), anlamları temelde program durum yazmacındaki S, Z, C ve V bayrakları ile aynıdır.

**Tablo 12.11** Koşullu Komut Yürütme için ARM Koşulları

Kod	Sembol	Test Edilen Durum	Yorum
0000	EQ	Z= 1	Eşit
0001	NE	Z= 0	Eşit değil
0010	CS/HS	C= 1	Taşıma seti/ işaretetsiz daha yüksek veya aynı
0011	CC/LO	C= 0	Açık/ işaretetsiz alt taşı
0100	MI	N= 1	Eksi/negatif
0101	PL	N= 0	Artı/pozitif veya sıfır
0110	VS	V= 1	Taşma
0111	VC	V= 0	Taşma yok
1000	HI	C= 1 VE Z= 0	İşretsiz daha yüksek
1001	LS	C= 0 VEYA Z= 1	İşretsiz düşük veya aynı
1010	GE	N= V [(N= 1 VE V= 1) VEYA (N= 0 VE V= 0)]	İşaretli büyük veya eşit
1011	LT	N≠ V [(N= 1 VE V= 0) VEYA (N= 0 VE V= 1)]	Daha az imzalandı
1100	GT	(Z= 0) VE (N= V)	Daha büyük işaretli
1101	LE	(Z= 1) VEYA (N≠ V)	Daha az veya eşit işaretli
1110	AL	-	Her zaman (koşulsuz)
1111	-	-	Bu komut yalnızca koşulsuz olarak çalıştırılabilir

x86 mimarisi. Bu dört bayrak ARM'de bir koşul kodu oluşturur. Tablo 12.11, koşullu yürütmenin tanımlanıldığı koşulların kombinasyonunu göstermektedir.

ARM'de koşul kodlarının kullanımının iki alışılmadık yönü vardır:

1. Sadece dallanma talimatları değil, tüm talimatlar bir koşul kodu alanı içerir, bu da neredeyse tüm talimatların koşullu olarak yürütülebileceği anlamına gelir. Bir komutun koşul kodu alanındaki 1110 veya 1111 dışındaki bayrak ayarlarının herhangi bir kombinasyonu, komutun yalnızca koşul karşılandığında yürütüleceğini belirtir.
2. Tüm veri işleme talimatları (aritmetik, mantıksal), taliminin koşul bayraklarını güncelleştirip güncellemeyi bir S biti içerir.

Koşullu yürütmenin kullanılması ve koşul bayraklarının koşullu olarak ayarlanması, daha az bellek kullanan daha kısa programların tasarlanmasına yardımcı olur. Öte yandan, tüm komutlar koşul kodu için 4 bit içerir, bu nedenle 32 bitlik komutta daha az bitin işlem kodu ve işlenenler için kullanılabilir olması nedeniyle bir değişim tokus söz konusudur. ARM büyük ölçüde kayıt adreslemeye dayanan bir RISC tasarımlı olduğundan, bu makul bir değişim tokus gibi

## 12.6 ANAHTAR TERİMLER, GÖZDEN GEÇİRME SORULARI VE PROBLEMLER

### Anahtar Terimler

akümülatör	Zıpla	prosedür çağrısı
adres	küçük endian	prosedür dönüşü
aritmetik kaydırma	mantıksal kaydırma	itmek
bi-endian	makine talimatı	reentrant prosedür
büyük endian	operand	döndürmek
şube	operasyon	atlamak
koşullu dal	paketlenmiş ondalık	yığın
talimat seti	pop	

### İnceleme Soruları

- 12.1 Bir makine talimatının tipik unsurları nelerdir?
- 12.2 Ne tür konumlar kaynak ve hedef işlenenleri tutabilir?
- 12.3 Bir komut dört adres içeriyorsa, her adresin amacı ne olabilir?
- 12.4 Beş önemli komut seti tasarımlı konusunu listeleyip kısaca açıklayınız.
- 12.5 Makine komut setlerinde tipik olarak ne tür işlenenler vardır?
- 12.6 IRA karakter kodu ile paketlenmiş ondalık gösterim arasındaki ilişki nedir?
- 12.7 Aritmetik kaydırma ile mantıksal arasındaki fark nedir?
- 12.8 Kontrol aktarımı talimatlarına neden ihtiyaç duyulur?
- 12.9 Bir koşullu dallanma komutunda test edilecek koşulu oluşturmanın iki yaygın yolunu listeleyiniz ve kısaca açıklayınız.
- 12.10 Prosedürlerin iç içe geçmesi terimi ile ne *kastedilmektedir*?
- 12.11 Bir prosedür *iadesi* için iade adresinin saklanabileceği üç olası yeri listeleyiniz.
- 12.12 Reentrant prosedür nedir?
- 12.13 Ters Lehçe notasyonu nedir?
- 12.14 Büyük endian ve küçük endian arasındaki fark nedir?

### Problemler

- 12.1 Onaltılık gösterimde gösterin:
  - a. 23 için paketlenmiş ondalık biçimini.
  - b. ASCII karakterleri 23.
- 12.2 Aşağıdaki paketlenmiş ondalık sayıların her biri için ondalık değeri gösteriniz:
  - a. 0111 0011 0000 1001
  - b. 0101 1000 0010
  - c. 0100 1010 0110
- 12.3 Verilen bir mikroişlemcinin 1 baylıklı sözcükleri vardır. Aşağıdaki gösterimlerde temsil edilebilecek en küçük ve en büyük tamsayı nedir?
  - a. İmzasız.
  - b. İşaret büyüklüğü.
  - c. Tamamlayıcılar.
  - d. İlkili tamamlayıcı.

- e. İşaretsiz paketlenmiş ondalık.  
 f. İşaretli paketlenmiş ondalık.
- 12.4** Birçok işlemci, paketlenmiş ondalık sayılar üzerinde aritmetik gerçekleştirmek için mantık sağlar. Ondalık aritmetik için kurallar ikili işlemler için olanlara benzer olsa da, ikili mantık kullanılırsa ondalık sonuçlar tek tek basamaklarda bazı düzeltmeler gerektirebilir.
- İki işaretsiz sayının ondalık toplamını düşünün. Her sayı  $N$  basamaktan, her sayıda 4N bit vardır. İki sayı bir ikili toplayıcı kullanılarak toplanacaktır. Sonucu düzeltmek için basit bir kural öneriniz. Bu şekilde toplama işlemini 1698 ve 1786 sayıları üzerinde gerçekleştirin.
- 12.5**  $X$  ondalık sayısının onlar tümleyeni  $10^N - X$  olarak tanımlanır; burada  $N$  sayıdaki ondalık basamak sayısıdır. Onluk çıkarma işlemini gerçekleştirmek için onluk tümleyen gösteriminin kullanımını açıklayınız.  $(0326)_{(10)}$  ve  $(0736)_{(10)}$ 'dan çıkararak prosedürü örneklendirin.
- 12.6**  $X$ 'i hesaplamak için programlar yazarak sıfır, bir, iki ve üç adresli makineleri karşılaştırın=  $(A + B * C)/(D - E * F)$

dört makinenin her biri için. Kullanım için mevcut talimatlar aşağıdaki gibidir:

0 Adres	1 Adres	2 Adres	3 Adres
PUSH M	YÜK M	HAREKET (X d Y)	HAREKET (X d Y)
POP M	M MAĞAZA	ADD (X d X+ Y)	ADD (X d Y+ Z)
ADD	ADD M	SUB (X d X - Y)	SUB (X d Y - Z)
SUB	SUB M	MUL (X d X * Y)	MUL (X d Y * Z)
MUL	MUL M	DIV (X d X/Y)	DIV (X d Y/Z)
DIV	DIV M		

- 12.7** Yalnızca iki  $n$ -bit komuttan oluşan bir komut setine sahip varsayımsal bir bilgisayar düşünün. İlk bit işlem kodunu belirtir ve kalan bitler ana belleğin  $2^{(n)(-)(1)}$  n-bitlik sözcüklerinden birini belirtir. İki talimat aşağıdaki gibidir:

ŞUBELEX XX konumunun içeriğini akümülatörden ve sonucu X konumunda ve akümülatörde saklayın.

JUMP X X adresini program sayacına yerleştirin.

Ana bellekteki bir sözcük bir komut ya da ikiye tümleyen gösteriminde ikili bir sayı içerebilir. Aşağıdaki işlemlerin nasıl programlanabileceğini belirterek bu komut repertuarının mantıklı bir şekilde eksiksiz olduğunu gösterin:

- a. Veri aktarımı: X konumundan akümülatöre, akümülatörden X konumuna.
- b. Toplama: X konumunun içeriğini akümülatöre ekler.
- c. Şartlı şube.
- d. Mantıksal VEYA.
- e. I/O İşlemleri.

- 12.8** Birçok komut seti, program sayacını artırmak dışında işlemci durumu üzerinde hiçbir etkisi olmayan NOOP, yani işlem yok komutunu içerir. Bu komutun bazı kullanımlarını öneriniz.

- 12.9** Bölüm 12.4'te, hem aritmetik sola kaydırmanın hem de mantıksal sola kaydırmanın taşıma olmadığından 2 ile çarpmaya karşılık geldiği ve taşıma olursa, aritmetik ve mantıksal sola kaydırma işlemlerinin farklı sonuçlar ürettiği, ancak aritmetik sola kaydırmanın sayının işaretini koruduğu belirtildi. Bu ifadelerin 5 bitlik ikiye tümleyen tamsayılar için doğru olduğunu gösterin.

- 12.10** Aritmetik sağa kaydırma kullanılarak sayılar ne şekilde yuvarlanır (örn.  $\infty$ , 'a doğru yuvarlayın, sıfırı doğru, 0'dan uzağa)?
- 12.11** Yordam çağrılarını ve geri dönüşleri yönetmek için işlemci tarafından bir yiğin kullanılacağını varsayıyalım. Yiğinin üst kısmı program sayacı olarak kullanılarak program sayacı ortadan kaldırılabilir mi?
- 12.12** x86 mimarisinde, Toplama Sonrası Ondalık Ayarlama (DAA) adı verilen bir komut içerir. DAA aşağıdaki talimat dizisini gerçekleştirir:

```

eğer ((AL VE 0FH >9( VEYA (AF= 1( o zaman
    AL d AL+ 6; AF
    d 1;

başka
    AF d 0;
endif;
eğer (AL> 9FH( VEYA (CF= 1( o zaman
    AL d AL+ 60H;
    CF d 1;

başka
    CF d 0;
endif.

```

"H" hexadecimal'i gösterir. AL, iki 8 bitlik tamsayının toplama işleminin sonucunu tutan 8 bitlik bir kayttır. AF, bir toplama işleminin sonucunda bit 3'ten bit 4'e bir taşıma varsa ayarlanan bir bayraktır. CF, bit 7'den bit 8'e bir taşıma varsa ayarlanan bir bayraktır. DAA komutu tarafından gerçekleştirilen işlevi açıklayın.

- 12.13** x86 Karşılaştırma komutu (CMP) kaynak işleneni hedef işlenenden çıkarır; durum bayraklarını (C, P, A, Z, S, O) günceller ancak işlenenlerden herhangi birini değiştirmez. CMP komutu, hedef kaynak operanddan büyük, eşit veya küçük olup olmadığını belirlemek için kullanılabilir.
- İki işlenenin işaretsiz tamsayılar olarak ele alındığını varsayıyalım. İki tamsayının görelî boyutunu belirlemek için hangi durum bayraklarının ilgili olduğunu ve bayrakların hangi değerlerinin büyük, eşit veya küçük değerlerine karşılık geldiğini gösterin.
  - İki işlenenin ikiye tümleyen işaretli tamsayılar olarak ele alındığını varsayıyalım. İki tamsayının görelî boyutunu belirlemek için hangi durum bayraklarının ilgili olduğunu ve bayrakların hangi değerlerinin büyük, eşit veya küçük değerlerine karşılık geldiğini gösterin.
  - CMP komutunu koşullu bir Jump (Jcc) veya Set Condition (SETcc) komutu izleyebilir; burada cc, Tablo 12.11'de listelenen 16 koşuldan birini ifade eder. İşaretli sayı karşılaştırması için test edilen koşulların doğru olduğunu gösterin.
- 12.14** X86 CMP komutunu, kayan nokta biçiminde sayılar içeren 32 bitlik işlenenlere uygulamak istediğimizi varsayıyalım. Doğru sonuçlar için, aşağıdaki alanlarda hangi gereksinimlerin karşılanması gereklidir?
- Anlam, işaret ve üs alanlarının görelî konumu.
  - Sıfır değerinin gösterimi.
  - Üs değerinin gösterimi.
  - IEEE formatı bu gereksinimleri karşılıyor mu? Açıklayın.
- 12.15** Birçok mikroişlemci komut seti, bir koşulu test eden ve koşul doğrusa bir hedef işleneni ayarlayan bir komut içerir. Örnekler arasında x86'daki SETcc, Motorola MC68000'deki Scc ve National NS32000'deki Scond sayılabilir.
- Bu talimatlar arasında birkaç farklılık vardır:
    - SETcc ve Scc yalnızca bir bayt üzerinde çalışırken, Scond bayt, word ve doubleword işlenenler üzerinde çalışır.
    - SETcc ve Scond, işleneni doğrusa tamsayı bire, yanlışsa sıfırı ayarlar. Scc, baytı doğrusa tüm ikili birlere ve yanlışsa tüm sıfırlara ayarlar. Bu farklılıkların göreceli avantajları ve dezavantajları nelerdir?

- b.** Bu talimatların hiçbirini koşul kodu bayraklarından herhangi birini ayarlamaz ve bu nedenle değerini belirlemek için talimatın sonucunun açık bir şekilde test edilmesi gereklidir. Bu talimin sonucu olarak koşul kodlarının ayarlanıp ayarlanmayacağına tartışınız.
- c.** IF a 7 b THEN gibi basit bir IF ifadesi, Boolean ifadesinin değerini programda ulaşılan bir nokta ile temsil eden *kontrol akışı* yönteminin aksine, Boolean değerini açık hale getiren sayısal bir temsil yöntemi kullanılarak uygulanabilir. Bir derleyici IF a 7 ssb THEN ifadesini aşağıdaki x86 kodu ile uygulayabilir:

SUB	CX, CX	;CX kaydını 0'a ayarlayın
MOV	AX, B	;B konumunun içeriğini AX kaydına taşı
CMP	AX, A	;AX yazmacının içeriğini ve A konumunu karşılaştırın
JLE	TEST	;A ... B ise atlayın
INC	CX	;CX kaydının içeriğine 1 ekleyin
TEST	JCXZ	;CX'in içeriği 0'a eşitse atla
SONRA		DIŞARI

(A 7 B)'nin sonucu, bir yazımcıkta tutulan ve daha , az önce gösterilen kod akışının bağlamı dışında kullanılabilen bir Boole değeridir. Bunun için CX yazmacını kullanmak uygundur, çünkü dallanma ve döngü işlem kodlarının çoğu CX için yerleşik bir test vardır.

SETcc komutunu kullanarak bellekten ve yürütme süresinden tasarruf sağlayan alternatif bir uygulama gösterin. (İpucu: SETcc dışında yeni bir x86 komutuna gerek yoktur).

- d.** Şimdi üst düzey dil ifadesini düşünün:

$$A := (B \neq C) \text{ VEYA } (D = F)$$

Bir derleyici aşağıdaki kodu üretebilir:

MOV	EAX, B	;B konumunun içeriğini EAX kaydına taşı
CMP	EAX, C	;EAX yazmacının içeriğini ve C konumunu karşılaştırın
MOV	BL, 0	;0 yanlışlı temsil eder
JLE	N1	;jump if (B ... C)
MOV	BL, 1	;1 yanlışlı temsil eder
N1	MOV	EAX, D
	CMP	EAX, F
	MOV	BH, 0
	JNE	N2
	MOV	BH, 1
N2	VEYA	BL, BH

SETcc komutunu kullanarak bellekten ve yürütme süresinden tasarruf sağlayan alternatif bir uygulama gösterin.

- 12.16** İki kaydın aşağıdaki onaltılık değerleri içerdiğini varsayılmı: AB0890C2, 4598EE50. MMX talimatlarını kullanarak bunları toplamanın sonucu nedir?

- a.** paketlenmiş bayt.  
**b.** Paketlenmiş kelime.

Doygunluk aritmetığının kullanılmadığını varsayıyın.

- 12.17** Ek I, eğer yoğun işlemci tarafından sadece prosedür işleme gibi amaçlarla kullanılacaksa, bir komut setinde yiğina yönelik komutların bulunmaması gerektiğine işaret etmektedir. Yiğin yönelikli talimatlar olmadan işlemci yiğini herhangi bir amaç için nasıl kullanabilir?
- 12.18** Matematiksel formüller genellikle infix notasyonu olarak bilinen ve işlenenler arasında ikil bir operatörün göründüğü ifade edilir. Alternatif bir teknik, operatörün iki operandını takip ettiği **ters Lehçe** veya **postfix** notasyon olarak bilinir. Daha fazla ayrıntı için Ek I'e bakınız. Aşağıdaki formülleri ters Lehçe'den infix'e dönüştürün:
- AB+ C+ D \*
  - +
  - ABCDE+ \* \* /
  - ABCDE+ F/+ G - H/ +
- 12.19** Aşağıdaki formülleri infix'ten ters Polish'e dönüştürün:
- A+ B+ C+ D+ E
  - (A+ B) \* (C+ D)+ E
  - (A \* B)+ (C \* D)+ E
  - (A - B) \* (((C - D \* E)/F)/G) \* H
- 12.20** A+ B - C ifadesini Dijkstra algoritmasını kullanarak postfix gösterimine dönüştürün. Ilgili adımları gösteriniz. Sonuç (A+ B) - C'ye mi yoksa A+ (B - C)'ye mi eşittir? Fark eder mi?
- 12.21** Ek I'de tanımlanan infix'i postfix'e dönüştürme algoritmasını kullanarak, Şekil I.3'teki ifadenin postfix'e dönüştürülmesinde yer alan adımları gösteriniz. Şekil I.5'e benzer bir sunum kullanınız.
- 12.22** Şekil I.4'e benzer bir sunum kullanarak Şekil I.5'ifadenin hesaplanması gösteriniz.
- 12.23** Şekil 12.13'teki little-endian düzenini, baytlar big-endian düzende numaralandırılmış olarak görünecek şekilde yeniden çizin. , belleği 64 bitlik satırlar halinde, baytlar soldan sağa, yukarıdan aşağıya listelenmiş olarak gösterin.
- 12.24** Aşağıdaki veri yapıları için, Şekil 12.13'teki formatı kullanarak big-endian ve little-endian düzenlerini çizin sonuçlar hakkında yorum yapın.
- struct {
 double i; //0x1112131415161718
 } s1;
  - struct {
 int i; //0x11121314
 int j; //0x15161718
 } s2;
  - struct {
 Kısa i; //0x1112
 Kısa j; //0x1314
 kısa k; //0x1516
 Kısa l; //0x1718
 } s3;
- 12.25** IBM Power mimarisi belirtimi, bir işlemcinin little-endian kipini nasıl uygulaması gerektiğini belirlemektedir. Yalnızca bir işlemcinin little-endian modunda çalışırken sahip olması gereken bellek görüntümünü belirtir. Bir veri yapısını big endian'dan little endian'a dönüştürmen, işlemciler gerçek bir bayt değiştirme mekanizması uygulamakta ya da bir çeşit adres değiştirme mekanizması kullanmaktadır serbesttir. Mevcut Power işlemcilerin tümü varsayılan big-endian makinelerdir ve veriyi little-endian olarak ele almak için adres modifikasyonu kullanırlar.
- Şekil 12.13'te tanımlanan s yapısını düşünün. Şeklin sağ alt kısmındaki düzen, işlemci tarafından görülen s yapısını göstermektedir. Aslında, yapılar little-endian modunda derlenirse, bellekteki yerlesimi Şekil 12.12'de gösterilmiştir.

Bayt adresi	Little-endian adres esleme							
	00	01	02	03	04	05	06	07
00	21	22	23	24	25	26	27	28
08	08	09	0A	0B	0C	0D	0E	0F
10	'D'	'C'	'B'	'A'	31	32	33	34
18	10	11	12	13	14	15	16	17
		51	52		'G'	'F'	'E'	
18	18	19	1A 1B	1C	1D	1E	1F	
20	20	21	22	23	24	25	26	27

**Şekil 12.12** Power Architecture Bellekteki Little-Endian Yapılar

İlgili eşlemeyi açıklayın, eşlemeyi uygulamanın kolay bir yolunu tarif edin ve bu yaklaşımın etkinliğini tartışın.

- 12.26** Makinenin endianlığını belirlemek ve sonuçları raporlamak için küçük bir program yazın. Programı size uygun bir bilgisayarda çalıştırın ve çıktısını teslim edin.
- 12.27** MIPS işlemcisi big-endian ya da little-endian modunda çalışacak şekilde ayarlanabilir. Bellekten bir baytı bir yazmacın düşük sıralı 8 bitine yükleyen ve yazmacın yüksek sıralı 24 bitini sıfırlarla dolduran Load Byte Unsigned (LBU) komutunu düşünün. LBU'nun açıklaması MIPS referans kılavuzunda bir kayıt aktarım dili kullanılarak şu şekilde verilmiştir

```

mem d LoadMemory(...(
byte d VirtualAddress1..0 if
CONDITION then
    GPR[rt] d 024|| mem31 - 8 * bayt ... 24 - 8 * bayt
başka
    GPR[rt] d 024|| mem7+ 8 * bayt ... 8 * bayt
endif

```

burada *byte* etkin adresin düşük sıralı iki bitini ve *mem* bellekten yüklenen değeri ifade eder. Kılavuzda, CONDITION kelimesi yerine aşağıdaki iki kelimedenden biri kullanılmıştır: BigEndian, LittleEndian. Hangi sözcük kullanılmıştır?

- 12.28** Hepsi olmasa çoğu işlemci, çok baytlı bir skaler içindeki baytların big- veya little-endian sıralamasıyla tutarlı olan bir bayt içinde big- veya little-endian bit sıralaması kullanır. Big-endian byte sıralaması kullanan Motorola 68030'u ele alalım. 68030'un formatlarla ilgili dokümantasyonu kafa karıştırıcıdır. Kullanım kılavuzu, bit alanlarının bit sıralamasının tamsayıların bit sıralamasının ters olduğunu açıklar. Çoğu bit alanı işlemi bir endian sıralama ile çalışır, ancak birkaç bit alanı ters sıralama gerektirir. Kullanım kılavuzundan alınan aşağıdaki açıklama bit alanı işlemlerinin çoğunu açıklamaktadır:

Bir bit operandi, bellekteki bir baytı (temel bayt) seçenek bir temel adres ve bu bayttaki bir biti seçenek bir bit numarası ile belirtilir. En anlamlı bit yedinci bittir. Bir bit alanı operandi şu şekilde belirtilir: (1) bellekte bir bayt seçenek bir temel adres; (2) temel baytin en anlamlı bitine göre bit alanının en soldaki (temel) bitini gösteren bir bit alanı ofseti; ve (3) bit alanında temel baytin sağında kaç bit olduğunu belirleyen bir bit alanı genişliği. Temel baytin en anlamlı biti bit alanı ofseti 0'dır, temel baytin en az anlamlı biti bit alanı ofseti 7'dir.

Bu talimatlar big-endian veya little-endian bit sıralaması kullanıyor mu?

## EK 12A KÜÇÜK-, BÜYÜK- VE Bİ-ENDIAN

Can sıkıcı ve ilginç bir olgu, bir kelime içindeki baytların ve bir bayt içindeki bitlerin hem nasıl referans alındığı hem de nasıl temsil edildiği ile ilgilidir. İlk olarak bayt sıralaması sorununa bakacağız ve daha sonra bitlerinkini ele alacağız.

### Bayt Sıralaması

Endianness kavramı literatürde ilk olarak Cohen [COHE81] tarafından tartışılmıştır. Baytlarla ilgili olarak, endianness çok baytlı sca- lar değerlerinin bayt sıralaması ile ilgilidir. Bu konu en iyi bir örnekle açıklanabilir. Elimizde 32-bit onaltılık 12345678 değeri olduğunu ve bu değerin bayt adreslenebilir bellekte 184 bayt konumunda 32-bitlik bir kelime saklandığını varsayıyalım. Değer 4 bayttan oluşan az anlamlı bayt 78 değerini ve en anlamlı bayt 12 değerini içerir. Bu değeri saklamadan iki belirgin yolu vardır:

Adres	Değer	Adres	Değer
184	12	184	78
185	34	185	56
186	56	186	34
187	78	187	12

Soldaki eşleme en anlamlı baytı en düşük sayısal bayt adresinde saklar; bu **big endian** olarak bilinir ve Batı kültür dillerindeki soldan sağa yazma sırasına eşdeğerdir. Sağdaki eşleme en az anlamlı baytı en düşük sayısal bayt adresinde saklar; bu **little endian** olarak bilinir ve aritmetik birimlerdeki aritmetik işlemlerin sağdan sola sırasını anımsatır.<sup>3</sup> Belirli bir çok baytlı skaler değer için, big endian ve little endian birbirlerinin bayt tersine çevrilmiş eşlemeleridir.

Endianlık kavramı, çok baytlı bir varlığın daha küçük adreslenebilir birimlerden oluşmasına rağmen tek bir adres'e sahip tek bir veri öğesi olarak ele alınması gereğinden ortaya çıkar. Intel 80x86, x86, VAX ve Alpha gibi bazı makineler little-endian makineler iken IBM System 370/390, Motorola 680x0, Sun SPARC ve çoğu RISC makinesi gibi diğerleri big endian'dır. Bu durum, veriler bir endian tipindeki makineden diğerine aktarıldığında ve programcı çok baytlı bir skaler içinde tek tek baytları veya bitleri değiştirmeye çalıştığında sorunlara açar.

Endianness özelliği tek bir veri biriminin ötesine geçmez. Herhangi bir makinede, dosyalar, veri yapıları ve diziler gibi toplamlar, her biri endianness özelliğine sahip birden fazla veri biriminden oluşur. Bu nedenle, bir bellek bloğunun bir endianness stilinden diğerine dönüştürülmesi, veri yapısı hakkında bilgi sahibi olmayı gerektirir.

Şekil 12.13, endianlığın adresleme ve bayt sırasını nasıl belirlediğini göstermektedir. Üstteki C yapısı bir dizi veri türü içerir. İçindeki bellek düzeni

<sup>3</sup> *Büyük endian* ve *küçük endian* terimleri Jonathan Swift'in *Gulliver'in Seyahatleri* adlı eserinin I. Bölüm, 4. Kısımından gelmektedir. Bu terimler, biri büyük üçta yumurta kıran, diğeri ise küçük üçta yumurta kıran iki grup arasındaki dini bir savaşa atıfta bulunmaktadır.

```

struct{
    int      a;           //0x1112_1314          kelime
    int      ped;         //
    çift     b;           //0x2122_2324_2526_2728   çift kelime
    char*   c;           //0x3132_3334          kelime
    char    d[7];        //'A','B','C','D','E','F','G' bayt dizisi
    kısa   e;           //0x5152            yarıml kelime
    int      f;           //0x6162_6364          kelime
} s;

```

Big-endian adres esleme										
Bayt adresi	11	12	13	14						
00	00	01	02	03	04	05	06	07		
	21	22	23	24	25	26	27	28		
08	08	09	0A	0B	0C	0D	0E	0F		
	31	32	33	34	'A'	'B'	'C'	'D'		
10	10	11	12	13	14	15	16	17		
	'E'	'F'	'G'		51	52				
18	18	19	1A	1B	1C	1D	1E	1F		
	61	62	63	64						
20	20	21	22	23						

Little-endian adres esleme										
Bayt adresi	11	12	13	14						
00	07	06	05	04	03	02	01	00		
	21	22	23	24	25	26	27	28		
08	0F	0E	0D	0C	0B	0A	09	08		
	'D'	'C'	'B'	'A'	31	32	33	34		
10	17	16	15	14	13	12	11	10		
		51	52			'G'	'F'	'E'		
18	1F	1E	1D	1C	1B	1A	19	18		
					61	62	63	64		
20					23	22	21	20		

Şekil 12.13 Örnek C Veri Yapısı ve Endian Haritaları

Sol alttaki, bu yapının big-endian bir makine için ve sağ alttaki de little-endian bir makine için derlenmesinden kaynaklanmaktadır. Her iki durumda da bellek 64 bitlik bir dizi satır olarak gösterilmektedir. Big-endian durumunda bellek tipik olarak soldan sağa, yukarıdan aşağıya doğru görüntülenirken, little-endian durumunda bellek tipik olarak sağdan sola, yukarıdan aşağıya doğru görüntülenir. Bu düzenlerin keyfi olduğunu unutmayın. Her iki düzen de bir satır içinde soldan sağa ya da sağdan sola kullanılabılır; bu bir tasvir meselesidir, bellek ataması değil. Aslında, çeşitli makineler için programcı kılavuzlarına bakıldığında, aynı kılavuz içinde bile şartlı bir tasvir koleksiyonu bulunabilir.

```

struct{
    int a; //0x1112_1314          word
    int pad; //
    double b; //0x2122_2324_2526_2728   çift kelime
    char* c; //0x3132_3334          kelime
    char d[7]; //'A','B','C','D','E','F','G' bayt dizisi
    short e; //0x5152            yarıml kelime
    int f; //0x6162_6364          kelime
} s;

```

Bu veri yapısı hakkında birkaç gözlem yapabiliriz:

- Her veri ögesi her iki şemada da aynı adrese sahiptir. Örneğin, onaltılk değeri 2122232425262728 olan çift sözcüğün adresi 08'dir.
- Verilen herhangi bir çok bayılı skaler değer içinde, little-endian yapısındaki baytların sıralaması big-endian yapısındaki tersidir.

- Endianness, bir yapı içindeki veri öğelerinin sıralamasını etkilemez. Dolayısıyla, dört karakterli c sözcüğü bayt tersine çevrilebilir, ancak yedi karakterli d bayt dizisi böyle değildir. Dolayısıyla, d'nin her bir elemanının adresi her iki yapıda da aynıdır.

Endianlığın etkisi, belleği Şekil 12.14'te gösterildiği gibi dikey bir bayt dizisi olarak belki de daha açık bir şekilde ortaya çıkmaktadır.

Hangi endian stilinin daha üstün olduğu konusunda genel bir fikir birliği yoktur.<sup>4</sup> Aşağıdaki noktalar big-endian stilini desteklemektedir:

- **Karakter dizisi sıralama:** Bir big-endian işlemci, tamsayı hizalı karakter dizilerini karşılaştırmada daha hızlıdır; tamsayı ALU, birden çok baytı paralel olarak karşılaştırabilir.

	00	00
04	11 12 13 14	14 13 12 11
08		
0C	21 22 23 24 25 26 27 28 31 32 33 34 'A' 'B' 'C' 'D' 'E' 'F' 'G'	28 27 26 25 24 23 22 21 34 33 32 31 'A' 'B' 'C' 'D' 'E' 'F' 'G'
10		
14	51 52	52 51
18	61 62 63 64	64 63 62 61
1C	dian	an
20		

**Şekil 12.14** Başka Bir Bakış Açısı  
Şekil 12.13

<sup>4</sup> Gulliver'in Seyahatleri'ndeki Endian Savaşları'nda her iki grup tarafından da saygı duyulan peygamber şunları söylemiştir. "Tüm gerçek İnananlar uygun Son'da Yumurtalarını kıracıklardır." Pek yardımcı olmadı!

- **Ondalık/IRA dökümleri:** Tüm değerler karışıklığa neden olmadan soldan sağa yazdırılabilir.
- **Tutarlı sıra:** Big-endian işlemciler tamsayılarını ve karakter dizilerini aynı sırada saklar (en anlamlı bayt önce gelir).

Aşağıdaki noktalar little-endian stilini desteklemektedir:

- Bir big-endian işlemci, 32 bitlik bir tamsayı adresini 16 bitlik bir tamsayı adresine dönüştürken, en az anlamlı baytları kullanmak için toplama işlemi yapmak zorundadır.
- Little-endian stili ile daha yüksek hassasiyetli aritmetik yapmak daha kolaydır; en az anlamlı bayti bulup geriye doğru hareket etmek zorunda kalmazsınız.

Farklılıklar küçüktür ve endian stilinin seçimi genellikle her şeyden çok önceki makinelere uyum sağlama meselesidir.

PowerPC, hem big-endian hem de little-endian modlarını destekleyen bi-endian bir işlemcidir. Bi-endian mimarisi, yazılım geliştiricilerin işletim sistemlerini ve uygulamaları diğer makinelerden taşırken her iki mod da seçebilmelerini sağlar. İşletim sistemi, işlemlerin yürütüleceği endian modunu belirler. Bir mod seçildikten sonra, sonraki tüm bellek yüklemeleri ve depolamaları bu modun bellek adresleme modeli tarafından belirlenir. Bu donanım özelliğini desteklemek için, işlem durumunun bir parçası olarak işletim sistemi tarafından tutulan makine durum kaydında (MSR) 2 bit tutulur. Bitlerden biri çekirdeğin çalıştığı endian modunu belirtir; diğeri ise işlemcinin mevcut çalışma modunu belirtir. Böylece, mod işlem bazında değiştirilebilir.

### Bit Sıralaması

Bir bayt içindeki bitleri sıralarken, hemen iki soruya karşı karşıya kalırız:

1. İlk biti sıfır biti olarak mı yoksa bir biti olarak mı sayıyorsunuz?
2. En düşük bit numarasını baytin en az anlamlı bitine mi (little endian) yoksa baytin en anlamlı bitine mi (big endian) atarsınız?

Bu sorular tüm makinelerde aynı şekilde yanıtlanmaz. Aslında, bazı makinelerde cevaplar farklı durumlarda farklıdır. Ayrıca, bir bayt içindeki big- veya little-endian bit sıralaması seçimi, çok bayaklı bir skaler içindeki baytların big- veya little-endian sıralamasıyla her zaman tutarlı değildir. Programcının tek tek bitleri işlerken bu konularla ilgilenesmesi gereklidir.

Bir başka endişe alanı da verilerin bit-seri hat üzerinden iletilmesidir. Tek bir bayt iletildiğinde, sistem en anlamlı biti mi yoksa en az anlamlı biti mi ileter? Tasarımcı, gelen bitlerin düzgün bir şekilde işlendiğinden emin olmalıdır. Bu konuya ilgili bir tartışma için bakınız [JAME90].

# 13

## KOMUT SETLERİ: ADRESLEME MODLARI VE FORMATLARI

### 13.1 Adresleme Modları

- Anlık Adresleme
- Doğrudan Adresleme
- Dolaylı Adresleme Kayıt
- Adresleme
- Kayıt Dolaylı Adresleme Yer
- Değiştirme Adresleme Yiğin
- Adresleme

### 13.2 x86 ve ARM Adresleme Modları x86

- Adresleme Modları ARM
- Adresleme Modları

### 13.3 Talimat Formatları

- Talimat Uzunluğu
- Bitlerin Tahsisı
- Değişken Uzunluklu Talimatlar

### 13.4 x86 ve ARM Komut Formatları x86

- Komut Formatları ARM Komut
- Formatları

### 13.5 Assembly Dili

### 13.6 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler

### ÖĞRENİM HEDEFLERİ

Bu bölümü çalıştıktan sonra şunları yapabilmelisiniz:

- Komut setlerinde yaygın olan çeşitli adresleme modlarını tanımlayabilecektir.
- x86 ve ARM adresleme modlarına genel bir bakış sunar.
- Bir **talimat formatının** tasarlanmasımda yer alan konuları ve ödünləşimleri özetleyiniz.
- x86 ve ARM komut formatlarına genel bir bakış sunar.
- Makine dili ve assembly dili arasındaki ayrimi anlamak.

Bölüm 12'de, bir komut küməsinin *ne* yaptığına odaklandık. Özellikle, makine komutları tarafından belirtilebilecek işlenen ve işlem türlerini inceledik. Bu bölüm, komutların ve işlemlerinin *nasıl* belirtileceği sorusuna dönmektedir. İki sorun ortaya çıkmaktadır. Birincisi, bir işlenenin adresi nasıl belirlenir ve ikincisi, bir komutun bitleri işlenen adreslerini ve o komutun işlemini tanımlamak nasıl düzenlenir?

## 13.1 ADRESLEME MODLARI

Tipik bir komut formatındaki adres alanı ya da alanları nispeten küçüktür. Ana bellekte ya da bazı sistemler için sanal bellekte geniş bir konum aralığına referans verebilmek isteriz. Bu amaca ulaşmak için çeşitli adresleme teknikleri kullanılmıştır. Bunların hepsi, bir yandan adres aralığı ve/veya adresleme esnekliği ile diğer yandan komuttaki bellek referanslarının sayısı ve/veya adres hesaplamasının karmaşıklığı arasında bazı ödünləşimler içerir. Bu bölümde, en yaygın adresleme tekniklerini ya da modlarını inceleyeceğiz:

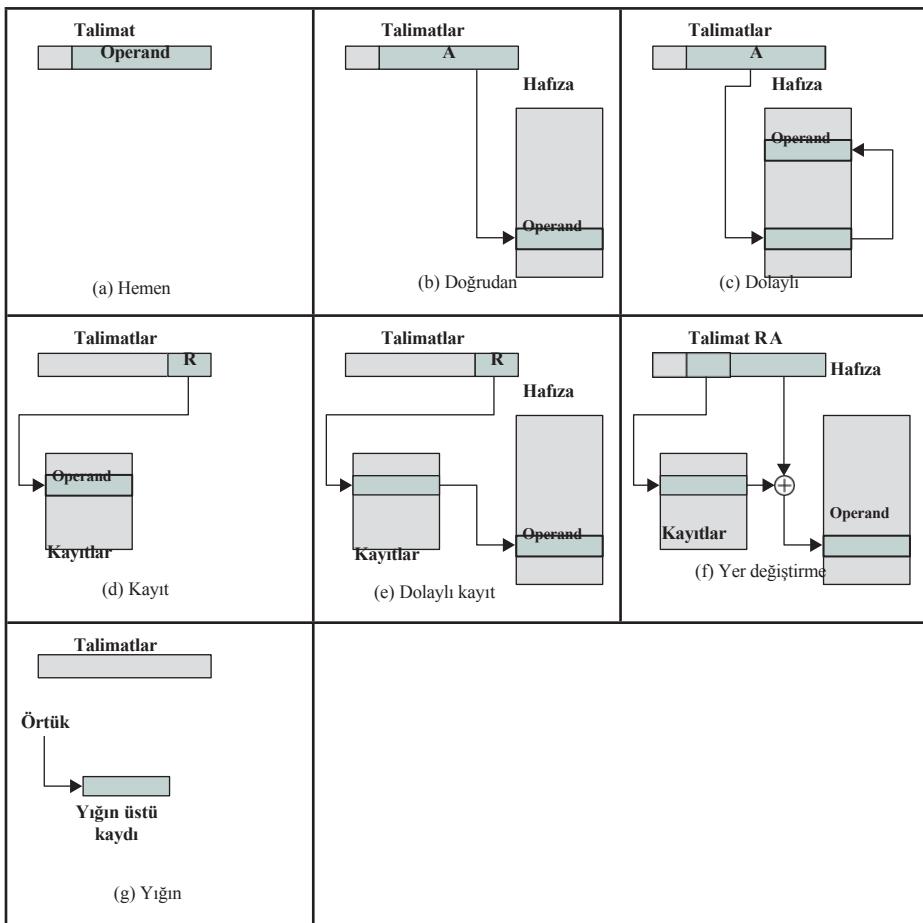
- Hemen
- Doğrudan
- Dolaylı
- Kayıt Olun
- Dolaylı kayıt
- Yer değiştirme
- Yiğin

Bu modlar Şekil 13.1'de gösterilmiştir. Bu bölümde aşağıdaki notasyonu kullanacağız:

**A**= talimatındaki bir adres alanının içeriği

**R**= bir yazmaca atıfta bulunan komuttaki adres alanının içeriği **EA**= atıfta bulunan işleneni içeren konumun gerçek (etkin) adresi

**(X)**= X bellek konumunun veya X kaydının içeriği



Şekil 13.1 Adresleme Modları

Tablo 13.1 her adresleme modu için gerçekleştirilen adres hesaplamasını gösterir.

Bu tartışmaya başlamadan önce iki yorum yapılması gerekmektedir. Birincisi, neredeyse tüm bilgisayar mimarileri bu adresleme modlarından birden fazlasını sağlamaktadır. İşlemcinin belirli bir komutta hangi adres modunun kullanıldığına nasıl belirleyebileceğini sorusu ortaya çıkar. Çeşitli yaklaşımlar benimsenmiştir. Genellikle, farklı işlem kodları farklı adresleme modları kullanacaktır. Ayrıca, komut formatındaki bir veya daha fazla bit *mod alanı* olarak kullanılabilir. Mod alanının değeri hangi adresleme modunun kullanılacağını belirler.

İkinci yorum etkin adresin (EA) yorumlanmasıyla ilgilidir. Sanal belleği olmayan bir sistemde, **etkin adres** ya bir ana bellek adresi ya da bir yazmaç olacaktır. Bir sanal bellek sisteminde, etkin adres bir sanal adres ya da bir yazmaçtır. Fiziksel adrese gerçek eşleme bellek yönetim biriminin (MMU) bir işlevidir ve programcı tarafından görülmez.

**Tablo 13.1** Temel Adresleme Modları

Mod	Algoritma	Ana Avantaj	Ana Dezavantaj
Hemen	Operand= A	Bellek referansı yok	Sınırlı operand büyüklüğü
Doğrudan	EA= A	Basit	Sınırlı adres alanı
Dolaylı	EA= (A)	Geniş adres alanı	Coklu bellek referansları
Kayıt Olun	EA= R	Bellek referansı yok	Sınırlı adres alanı
Dolaylı kayıt	EA= (R)	Geniş adres alanı	Ekstra bellek referansı
Yer Değiştirme	EA= A+ (R)	Esneklik	Karmaşıklık
Yığın	EA= yiğinin üstü	Bellek referansı yok	Sınırlı uygulanabilirlik

### Anında Adresleme

Adreslemenin en basit şekli, işlenen değerin komutta mevcut olduğu **anlık adreslemedir**

$$\text{Operand} = \text{A}$$

Bu mod, sabitleri tanımlamak veya kullanmak veya değişkenlerin başlangıç değerlerini ayarlamak için kullanılabilir. Tipik olarak, sayı ikiye tümleyen biçiminde saklanır; işlenen alanının en soldaki biti işaret biti olarak kullanılır. İşlenen bir veri kaydediciye yüklenliğinde, işaret biti tam veri **sözcüğü** boyutuna kadar sola doğru genişletilir. Bazı durumlarda ikili değer işaretsiz negatif olmayan bir tamsayı olarak yorumlanır.

Anında adreslemenin avantajı, işleneni elde etmek için komut getirme dışında bir bellek referansı gerekmemesi ve böylece komut bir bellek veya önbellek çevriminden tasarruf edilmesidir. Dezavantajı ise sayının boyutunun adres alanının boyutuya sınırlı olmasıdır ki bu da çoğu komut setinde kelime uzunluğuna kıyasla küçüktür.

### Doğrudan Adresleme

Adreslemenin çok basit bir şekli, adres alanının işlenenin etkin adresini içerdiği doğrudan :

$$\text{EA} = \text{A}$$

Bu teknik daha önceki bilgisayar nesillerinde yaygındı ancak çağdaş mimarilerde yaygın değildir. Sadece bir bellek referansı gerektirir ve özel bir hesaplama . Bariz sınırlaması ise sadece sınırlı bir adres alanı sağlamasıdır.

### Dolaylı Adresleme

Doğrudan adreslemede, adres alanının uzunluğu genellikle kelime uzunluğundan daha azdır, dolayısıyla adres aralığını sınırlar. Bir çözüm, adres alanının bellekteki bir sözcüğün adresini ifade etmesini sağlamaktır, bu da işlenenin tam uzunluktaki adresini içerir. Bu **dolaylı adresleme** olarak bilinir:

$$\text{EA} = (\text{A})$$

Daha önce tanımlandığı gibi, parantezler *içeriği* anlamına gelecek şekilde yorumlanmalıdır. Bu yaklaşımın bariz avantajı,  $N$  kelime uzunluğu için artık  $2^N$ 'lik bir adres alanının mevcut olmasıdır. Dezavantaj ise komutun yürütülmesinde işleneni almak için iki bellek referansı gerekmektedir: biri adresini almak için, ikincisi ise değerini almak için.

Adreslenebilecek kelime sayısı artık  $2^N$ 'ye eşit olsa da, herhangi bir zamanda başvurulabilecek farklı etkin adreslerin sayısı  $2^K$  ile sınırlıdır, burada  $K$  adres alanının uzunluğudur. Tipik olarak, bu çok yoğun bir kısıtlama değildir ve bir da olabilir. Sanal bellek ortamında, tüm etkin adres konumları herhangi bir işlemin 0. sayfasıyla sınırlanır. Bir komutun adres alanı küçük olduğundan, doğal olarak sayfa 0'da görünecek olan düşük numaralı direct adresleri üretecektir. (Tek kısıtlama, sayfa boyutunun  $2^K$ 'dan büyük veya eşit olmasıdır). Bir süreç aktif olduğunda, sayfa 0'a tekrarlanan referanslar olacaktır, bu da gerçek bellekte kalmasına neden olur. Böylece, dolaylı bir bellek referansı iki yerine en fazla bir sayfa hatası içerecektir.

Dolaylı adreslemenin nadiren kullanılan bir çeşidi de çok seviyeli veya kademeli dolaylı adreslemedir:

$$EA = (C (A) C)$$

Bu durumda, tam kelime adresinin bir biti dolaylı bayraktır (I). I biti 0 ise, kelime EA'yı içerir. I biti 1 ise, başka bir dolaylama seviyesi çağrılır. Bu yaklaşımın özel bir avantajı yok gibi görülmektedir ve dezavantajı, bir işleneni almak için üç veya daha fazla bellek referansının gerekebilmesidir.

### Kayıt Adresleme

**Kayıt adresleme** doğrudan adreslemeye benzer. Tek fark, adres alanının bir ana bellek adresi yerine bir yazmacı ifade etmesidir:

$$EA = R$$

Açıklamak, bir komuttaki yazmacı adres alanının içeriği 5 ise, R5 yazmacı amaçlanan adresdir ve işlenen değer R5'te bulunur. Tipik olarak, yazmacılara referans veren adres alanı 3 ila 5 bit arasında olacaktır, böylece toplam 8 ila 32 genel amaçlı yazmacı referans alınabilir.

Yazmacı adreslemenin avantajları (1) komutta sadece küçük bir adres alanına ihtiyaç duyulması ve (2) zaman alıcı bellek referanslarına gerek olmamasıdır. Bölüm 4'te tartışıldığı gibi, işlemciye dahili bir yazmacı için bellek erişim süresi, bir ana bellek adresi için olandan çok daha azdır. Yazmacı adreslemenin dezavantajı adres alanının çok sınırlı olmasıdır.

Eğer bir komut setinde yazmacı adresleme yoğun olarak kullanılıyorsa, bu işlemci yazmacılarının da yoğun olarak kullanılacağı anlamına gelir. (Ana bellek konumları ile karşılaşıldığında) son derece sınırlı yazmacı olduğundan, bu şekilde kullanılmaları ancak verimli bir şekilde kullanılması durumunda anlamlıdır. Eğer her işlenen ana bellekten bir kaydediciye getirilir, üzerinde bir kez işlem yapılır ve sonra ana belleğe geri gönderilirse, o zaman boş harcanan bir ara adım eklenmiş olur. Bunun yerine, bir yazmacı işlenen birden fazla işlem için kullanımda kalırsa, o zaman gerçek bir tasarruf elde edilir. Bir hesaplamadaki ara sonuç buna bir örnektir. Özellikle, algoritmanın aşağıdaki gibi olduğunu varsayıyalım

ikiye tamamlama çarpımı için yazılımda uygulanacaktır. Akış şemasında (Şekil 10.12) A olarak etiketlenen konuma birçok kez başvurulmaktadır ve bir ana bellek konumu yerine bir yazmaca uygulanmalıdır.

Hangi değerlerin yazmaçlarda kalacağına ve hangilerinin ana bellekte saklanacağına karar vermek programcıya veya derleyiciye bağlıdır. Modern işlemcilerin çoğu birden fazla genel amaçlı yazmaç kullanır ve bu da assembly dili programcısına (örneğin derleyici yazarına) verimli bir yürütme yükü getirir.

### Kayıt Dolaylı Adresleme

Yazmaç doğrudan **adreslemeye** benzediği gibi, **yazmaç dolaylı** adresleme de **dolaylı adreslemeye** benzer. Her iki durumda da tek fark, adres alanının bir bellek konumuna mı yoksa bir yazmaca mı atıfta bulunduğuudur. Böylece, kayıt dolaylı adresleme için,

$$EA = (R)$$

Kayıt dolaylı adreslemenin avantajları ve sınırlamaları temelde dolaylı adresleme ile aynıdır. Her iki durumda da, adres alanının adres alanı sınırlaması (sınırlı adres aralığı), bu alanın bir içeren kelime uzunluğundaki bir konuma atıfta bulunmasıyla aşılır. Ayrıca, kayıt dolaylı adresleme, dolaylı adreslemeye göre bir daha az bellek referansı kullanır.

### Yer Değiştirme Adresleme

Doğrudan adresleme ve kayıt dolaylı özelliklerini birleştiren çok güçlü bir adresleme modudur. Kullanım bağlamına bağlı çeşitli isimlerle bilinir, ancak temel mekanizma aynıdır. Biz bunu **yer değiştirme adreslemesi** olarak adlandıracagız:

$$EA = A + (R)$$

Yer değiştirme adreslemesi, komutun en az biri açık olmak üzere iki adres alanına sahip olmasını gerektirir. Bir adres alanında bulunan değer (değer= A) doğrudan kullanılır. Diğer adres alanı veya işlem koduna dayalı örtük bir referans, etkin adresi üretmek içeriği A'ya eklenen bir yazmacı ifade eder.

Yer değiştirme adreslemesinin en yaygın üç kullanımını açıklayacağız:

- Göreceli adresleme
- Temel kayıt adresleme
- İndeksleme

**GÖRECELİ ADRESLEME** PC-göreceli adresleme olarak da adlandırılan göreceli adresleme için, dolaylı olarak başvurulan kayıt program sayacıdır (PC). Yani, bir sonraki komut adresi EA'yı üretmek için adres alanına eklenir. Tipik olarak, adres alanı bu işlem için ikiye tümleyen bir sayı olarak ele alınır. Böylece etkin adres, komutun adresine göre bir yer değiştirmedir.

Göreceli adresleme, Bölüm 4 ve 8'de tartışılan yerellik kavramından faydalıdır. Eğer bellek referanslarının çoğu çalıştırılan komuta nispeten yakınsa, göreceli adresleme kullanımı komuttaki adres bitlerinden tasarruf sağlar.

**TABAN KAYIT ADRESLEME** Taban kayıt adresleme için yorumlama aşağıdaki gibidir: Referans edilen kayıt bir ana bellek adresi içerir ve adres alanı bu adresten bir yer değiştirme (genellikle işaretsiz bir tamsayı gösterimi) içerir. Kayıt referansı açık ya da örtük olabilir.

Taban-kayıt adresleme bellek referanslarının yerelligidenden de faydalanan. Bölüm 8'de tartışılan segmentasyonu uygulamak için uygun bir araçtır. Bazı uygulamalarda, tek bir segment-base register kullanılır ve örtük olarak kullanılır. Diğerlerinde, programcı bir segmentin taban adresini tutmak için bir register seçebilir ve komut buna açıkça referans vermelidir. Bu ikinci durumda, adres alanının uzunluğu  $K$  ve olası yazmaçların sayısı  $N$  ise, bir komut  $2^K$  kelimelek  $N$  alandan herhangi birine başvurabilir.

**İNDEKSLEME** İndeksleme için yorumlama tipik olarak aşağıdaki gibidir: Adres alanı bir ana bellek adresine referans verir ve referans verilen yazmaç bu adresten pozitif bir yer değiştirme içerir. Bu kullanımın, temel kayıt adresleme yorumunun tam tersi olduğuna dikkat edin. Tabii ki, bu sadece bir kullanıcı yorumu meselesinden daha fazlasıdır. Adres alanı indekslemede bir bellek adresi olarak kabul edildiğinden, genellikle karşılaştırılabilir bir taban-kayıt komutundaki adres alanından daha fazla bit içerir. Ayrıca, indekslemede taban-kayıt bağlamında o kadar kullanışlı olmayacağı bazı iyileştirmeler olduğunu göreceğiz. Bununla birlikte, EA hesaplama yöntemi hem taban-kayıt adresleme hem de indeksleme için aynıdır ve her iki durumda da kayıt referansı bazen açık bazen de örtüktür (farklı işlemci türleri için).

İndekslemenin önemli bir kullanımı da yinelemeli işlemlerin gerçekleştirilmesi için etkin bir mekanizma sağlamaktır. Örneğin, A konumundan saklanan bir sayı listesi düşünün. Listedeki her bir öğeye 1 eklemek istediğimizi varsayıyalım. Her bir değeri almamız, ona 1 eklememiz ve geri depolamamız gerekdir. İhtiyacımız olan etkin adresler dizisi A, A+ 1, A+ 2, ..., listedeki son konuma kadar. İndeksleme ile bu kolayca yapılır. A değeri komutun adres alanında saklanır ve indeks kaydı olarak adlandırılan seçilen kayıt 0'a başlatılır. Her işlemden sonra indeks kaydı 1 artırılır.

Dizin yazmaçları bu tür yinelemeli görevler için yaygın olarak kullanıldığından, dizin yazmaçlarına yapılan her başvurudan sonra bu yazmaçların artırılması ya da azaltılması tipik bir ihtiyaçtır. Bu çok yaygın bir işlem olduğundan, bazı sistemler bunu aynı komut döngüsünün bir parçası olarak otomatik olarak yapar. Bu **otomatik indeksleme** olarak bilinir. Belirli yazmaçlar yalnızca dizinlemeye ayrılmışsa, otomatik dizinleme dolaylı ve otomatik olarak çağrılabılır. Genel amaçlı yazmaçlar kullanılıyorsa, otomatik indeksleme işleminin komuttaki bir bit tarafından bildirilmesi gerekebilir. Artış kullanarak otomatik indeksleme aşağıdaki gibi gösterilebilir.

$$EA = A + (R)$$

$$(R) \leftarrow (R) + 1$$

Bazı makinelerde hem dolaylı adresleme hem de indeksleme sağlanır ve her ikisini de aynı komutta kullanmak mümkündür. İki olasılık vardır: indeksleme, dolaylamadan önce ya da sonra gerçekleştirilir.

**İndeksleme** dolaylamadan sonra gerçekleştirilirse, **postindeksleme** olarak adlandırılır:

$$EA = (A) + (R)$$

İlk olarak, adres alanının içeriği doğrudan adres içeren bir bellek konumuna erişmek için kullanılır. Bu adres daha sonra kayıt değeri tarafından indekslenir. Bu teknik, sabit bir formattaki bir dizi veri bloğundan birine erişmek için kullanılmıştır. Örneğin, Bölüm 8'de işletim sisteminin her bir süreç için bir süreç kontrol bloğu kullanması gerektiği açıklanmıştır. Gerçekleştirilen işlemler, hangi bloğun manipüle edildiğine bakılmaksızın ayındır. Bu nedenle, bloğa referans veren talimatlardaki adresler, bir süreç kontrol bloğunun başlangıcına değişken bir işaretçi içeren bir konuma (değer= A) işaret edebilir. İndeks kaydi blok içindeki yer değiştirmeyi içerir.

**Ön** indeksleme ile indeksleme, dolaylamadan önce gerçekleştirilir:

$$EA = (A + (R))$$

Basit indekslemede olduğu gibi bir adres hesaplanır. Ancak bu durumda, hesaplanan adres işleneni değil, işlenenin adresini içerir. Bu tekniğin kullanımına bir örnek, çok yönlü bir dallanma tablosu oluşturmaktır. Bir programın belirli bir noktasında, koşullara bağlı olarak bir dizi konumdan birine dallanma olabilir. A konumundan başlayan bir adres tablosu oluşturulabilir. Bu tabloya indeksleme yapılarak gerekli konum bulunabilir.

Tipik olarak, bir komut seti hem ön indeksleme hem de son indeksleme içermez.

## Yığın Adresleme

Ele aldığımız son adresleme modu yığın adreslemedi. Ek I'de tanımlandığı gibi, yığın doğrusal bir konum dizisidir. Bazen *pushdown listesi* ya da *son giren ilk çıkar kuyruğu* olarak da adlandırılır. Yığın, ayrılmış bir konum bloğudur. Öğeler yığının en üstüne eklenir, herhangi bir zamanda blok kısmen doldurulmuş olur. Yığınla ilişkili olarak, değeri yığının en üstünün adresi olan bir işaretçi bulunur. Alternatif olarak, yığının ilk iki elemanı işlemci kayıtlarında olabilir, bu durumda yığın işaretçisi yığının üçüncü elemanını referans alır. Yığın işaretçisi bir kayıtta tutulur. Bu nedenle, bellekteki yığın konumlarına yapılan referanslar aslında yazmaç dolaylı adresleridir.

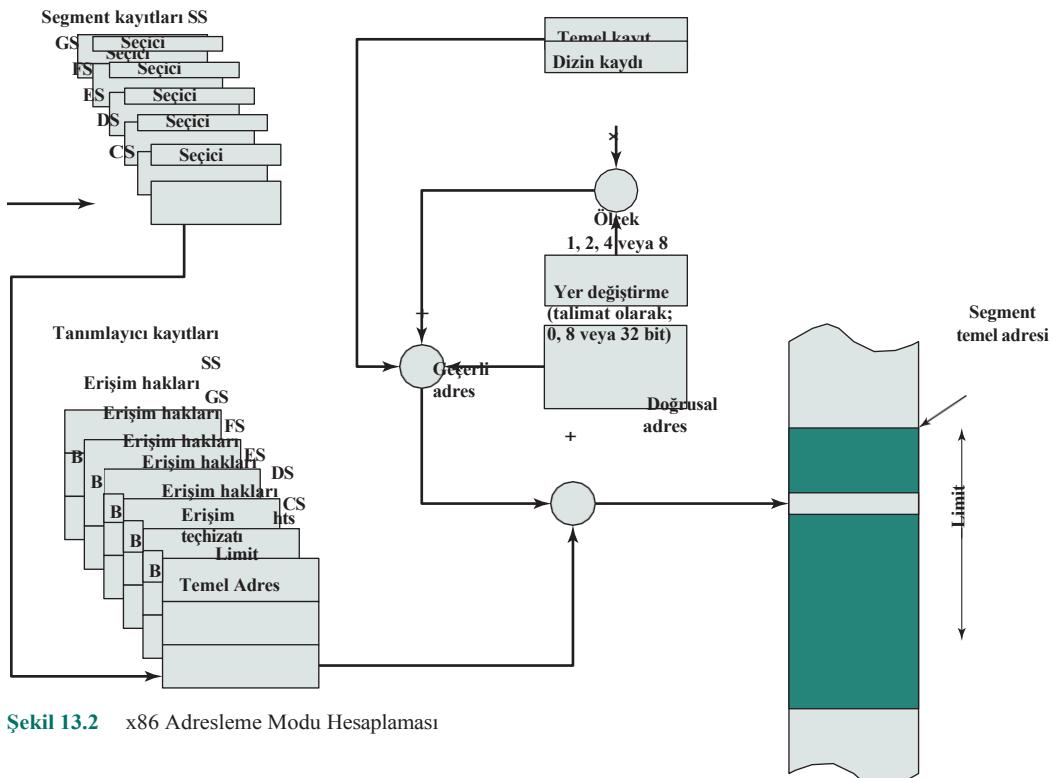
Yığın adresleme modu bir zimni adresleme biçimidir. Makine talimatlarının bir bellek referansı içermesi gerekmek, ancak dolaylı olarak yığının en üstünde çalışır.

## 13.2 x86 VE ARM ADRESLEME MODLARI

### x86 Adresleme Modları

Şekil 8.21'den x86 adres çeviri mekanizmasının sanal ya da etkin adres olarak adlandırılan ve bir segmentin ofseti olan bir adres ürettiğini hatırlayın. Segmentin başlangıç adresi ile etkin adresin toplamı doğrusal bir üretir. Eğer sayfalama kullanılıyorsa, bu doğrusal adres fiziksel bir adres üretmek için bir sayfa çevirme mekanizmasından geçmelidir. Aşağıda, bu son adımı görmezden geliyoruz çünkü komut seti ve programcı için şeffaftır.

x86, yüksek seviyeli dillerin verimli bir şekilde yürütülmesini sağlamak amacıyla çeşitli adresleme modlarıyla donatılmıştır. Şekil 13.2 mantığı gösterir



Şekil 13.2 x86 Adresleme Modu Hesaplaması

dahil. Segment kaydı, referans konu olan segmenti belirler. Altı segment kaydedicisi vardır; belirli bir referans için kullanılacak olan, yürütme bağlamına ve komuta bağlıdır. Her segment kaydı, ilgili segmentlerin başlangıç adresini tutan segment tanımlayıcı tablosuna (Şekil 8.20) bir indeks tutar. Kullanıcı tarafından görülebilen her segment kaydı ile ilişkili bir segment tanımlayıcı kaydı vardır (programcının tarafından görülemez) ve segmentin erişim haklarının yanı sıra segmentin başlangıç adresini ve limitini (uzunluk) kaydeder. Buna ek olarak, bir adres oluşturmada kullanılabilecek iki yazmaç vardır: temel yazmaç ve dizin yazmaç.

Tablo 13.2 x86 adresleme modlarını listeler. Bunların her birini sırayla ele alalım. **Anlık mod** için, işlenen komuta dahil edilir. Operatör ve bir bayt, kelime veya çift kelime veri olabilir.

**Kayıt operand modu** için, operand bir bulunur. Veri aktarımı, aritmetik ve mantıksal komutlar gibi genel komutlar için, işlenen 32 bit genel yazmaçlardan (EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP) biri, 16 bit genel yazmaçlardan (AX, BX, CX, DX, SI, DI, SP, BP) biri veya 8 bit genel yazmaçlardan (AH, BH, CH, DH, AL, BL, CL, DL) biri olabilir. Ayrıca segment seçici kayıtlarına (CS, DS, ES, SS, FS, GS) referans veren bazı talimatlar da vardır.

**Tablo 13.2** x86 Adresleme Modları

Mod	Algoritma
Hemen	$\text{Operand} = A$
Kayıt Operandı	$LA = R$
Yer değiştirme	$LA = (SR) + A$
Üs	$LA = (SR) + (B)$
Deplasmanlı Taban	$LA = (SR) + (B) + A$
Yer Değiştirme ile Ölçeklendirilmiş Endeks	$LA = (SR) + (I) * S + A$
İndeks ve Deplasmanlı Taban	$LA = (SR) + (B) + (I) + A$
Ölçekli Endeks ve Deplasmanlı Taban	$LA = (SR) + (I) * S + (B) + A$
Göreceli	$LA = (PC) + A$

LA= doğrusal adres

R= kayıt

(X)= X'in içeriği SR=

B= temel kayıt I=

segment kaydı PC=

dizin kaydı S=

program sayacı

ölçekleme faktörü

A= talimatındaki bir adres alanının içeriği

Geri kalan adresleme modları bellekteki konumları referans alır. Bellek konumu, konumu içeren segment ve başlangıcından itibaren ofset cinsinden belirtilmelidir. Bazı durumlarda, bir segment açıkça belirtilir; diğerlerinde, segment varsayılan olarak bir segment atayan basit kurallarla belirtilir.

**Yer değiştirme modunda**, işlenenin ofseti (Şekil 13.2'deki etkin adres) komutun bir parçası olarak 8-, 16- veya 32-bit yer değiştirme olarak bulunur. Segmentasyon ile, talimatlardaki tüm adresler sadece segmentteki bir ofsete atıfta bulunur. Yer değiştirme adresleme modu çok az makinede bulunur çünkü daha önce de belirtildiği gibi uzun talimatlara yol açar. X86 durumunda, yer değiştirme değeri 32 bit kadar uzun olabilir, bu da 6 baytlık bir komut oluşturur. Yer değiştirme adreslemesi global değişkenlere referans vermek için kullanışlı olabilir.

Geri kalan adresleme modları, komutun adres kısmının işlemciye adresi bulmak için nereye bakması gerektiğini söylemesi anlamında dolaylıdır. **Temel mod**, 8, 16 ya da 32 bitlik yazmacılardan birinin etkin adresi içerdigini belirtir. Bu, kayıt dolaylı adresleme olarak adlandırdığımız seye eşdeğerdir.

**Yer değiştirmeli taban modunda** komut, genel amaçlı kayıtlardan herhangi biri olabilecek bir taban kaydına eklenecek bir yer içerir. Bu modun kullanım örnekleri aşağıdaki gibidir:

- Derleyici tarafından bir yerel değişken alanının başlangıcını işaret etmek için kullanılır. Örneğin, taban kaydı, ilgili prosedür için yerel değişkenleri içeren bir yığın çerçevesinin başlangıcına işaret edebilir.
- Eleman boyutu 1, 2, 4 veya 8 bayt olmadığından ve nedenle bir dizin yazmacı kullanılarak dizinlenemeyen bir diziye dizin oluşturmak için kullanılır. Bu durumda, yer değiştirme dizinin başlangıcına işaret eder ve taban yazmacı, dizi içindeki belirli bir öğeye olan ofseti belirlemek için hesaplamaların sonuçlarını tutar.

- Bir kaydın bir alanına erişmek için kullanılır. Temel kayıt kaydın gösterirken, yer değiştirme alana bir ofsettir.

**Yer değiştirmeli** ölçeklendirilmiş **indeks modunda** komut, bu durumda indeks kaydı olarak adlandırılan bir kayda eklenecek bir yer içerir. İndeks yazmacı, genellikle yoğun işleme için kullanılan ESP adlı yazmaç dışında genel amaçlı yazmaçlardan herhangi biri olabilir. Etkin adres hesaplanırken, indeks kaydının içeriği 1, 2, 4 veya 8 ölçekleme faktörü ile çarpılır ve ardından bir yer değiştirmeye eklenir. Bu mod dizileri indekslemek için çok uygundur. 16-bit tamsayılarından oluşan bir dizi için 2 ölçekleme faktörü kullanılabilir. 32 bitlik tamsayılar veya kayan noktalı sayılar için 4 ölçekleme faktörü kullanılabilir. Son olarak, çift hassasiyetli kayan noktalı sayılarından oluşan bir dizi için 8 ölçekleme faktörü kullanılabilir.

**Dizin ve yer değiştirme modu ile** taban, taban yazmacının, dizin yazmacının ve bir yer değiştirmenin içeriğini etkin adresi oluşturmak için toplar. Yine, taban yazmacı herhangi bir genel amaçlı yazmaç olabilir ve dizin yazmacı ESP hariç herhangi bir genel amaçlı yazmaç olabilir. Örnek olarak, bu adresleme modu bir yoğun çerçevesi üzerindeki yerel bir diziye erişmek için kullanılabilir. Bu mod ayrıca iki boyutlu bir diziyi desteklemek için de kullanılabilir; bu durumda, yer değiştirme dizinin başlangıcına işaret eder ve her yazmaç dizinin bir boyutunu işler.

**Yer değiştirme modlu** temel **ölçekli** dizin, dizin yazmacının içeriğini bir ölçekleme faktörü, temel yazmacın içeriği ve yer değiştirme ile çarparak toplar. Bu, bir dizi bir yoğun çerçevesi içinde saklanırsa kullanılabilir; bu durumda, dizi elemanlarının her biri 2, 4 veya 8 bayt uzunlığında olacaktır. Bu mod ayrıca, dizi elemanları 2, 4 veya 8 bayt uzunlığında olduğunda iki boyutlu bir dizinin verimli bir şekilde indekslenmesini sağlar.

Son olarak, **göreli adresleme** kontrol aktarımı talimatlarında kullanılabilir. Bir sonraki komutu işaret eden program sayacının değerine bir yer değiştirme eklenir. Bu durumda, yer değiştirme işaretli bir bayt, sözcük ya da iki sözcük değeri olarak ele alınır ve bu değer program sayacındaki adresi artırır ya da azaltır.

## ARM Adresleme Modları

Tipik olarak, bir RISC makinesi, bir CISC makinesinin aksine, basit ve nispeten anlaşılır bir dizi adresleme modu kullanır. ARM mimarisi, nispeten zengin bir adresleme modu seti sağlayarak bu gelenekten biraz ayrılır. Bu modlar en uygun şekilde komut türüne göre sınıflandırılır.<sup>1</sup>

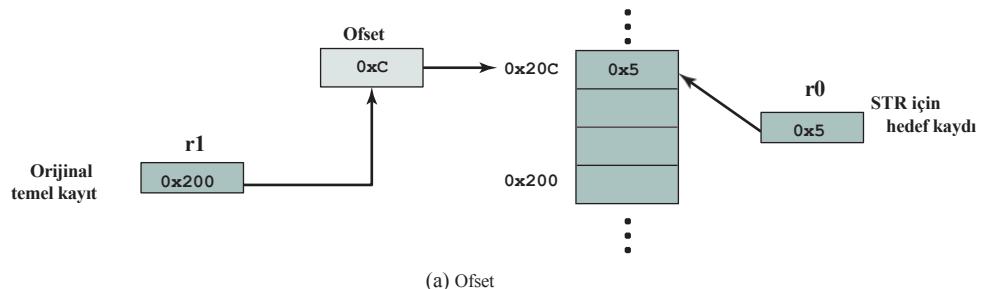
**LOAD/STORE ADRESLEME** Load ve store komutları belleğe referans veren tek komutlardır. Bu her zaman bir temel kayıt artı ofset aracılığıyla dolaylı olarak yapılır. İndeksleme ile ilgili üç alternatif vardır (Şekil 13.3):

- **Offset:** Bu adresleme yöntemi için **indeksleme** kullanılmaz. Bellek adresini oluşturmak için taban yazmacındaki değere bir ofset değeri eklenir veya çıkarılır. Örnek olarak Şekil 13.3a bu yöntemi STRB r0, [r1, #12] assembly dili komutu ile göstermektedir. Bu, bayt saklama komutudur.

---

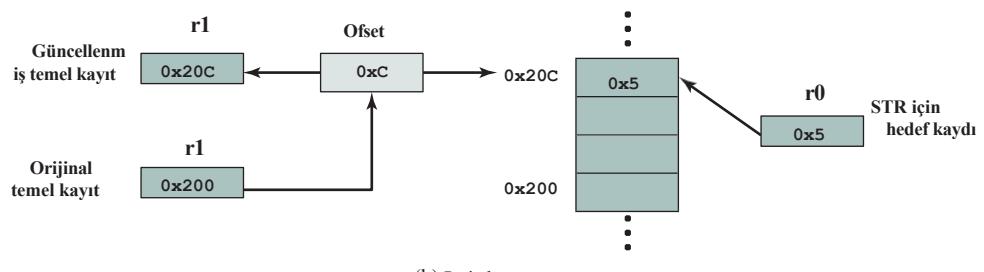
<sup>1</sup>x86 adresleme olduğu gibi, aşağıdaki tartışmada sanal adresten fiziksel adrese çeviriyi göz ardi ediyoruz.

`STRB r0, [r1, #12]`



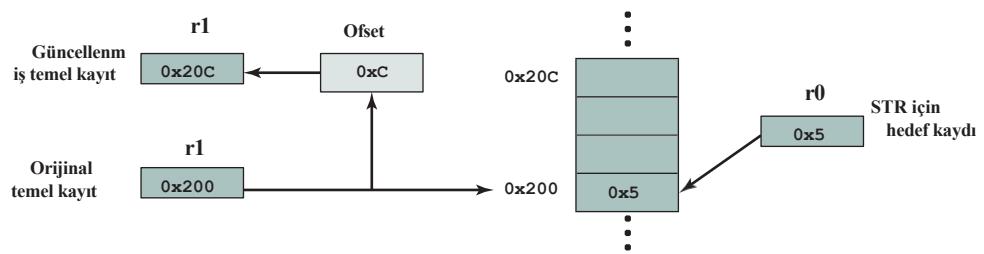
(a) Offset

`STRB r0, [r1, #12]!`



(b) Preindex

`STRB r0, [r1], #12`



(c) Postindex

**Şekil 13.3** ARM İndeksleme Yöntemleri

Bu durumda temel adres `r1` kaydedicisindedir ve yer değiştirme ondalık 12'nin yakın bir değeridir. Ortaya çıkan adres (taban artı kaydırma), `r0`'dan en az anlamlı baytin saklanacağı konumundur.

- **Ön indeks:** Bellek adresi ofset adresleme ile aynı şekilde oluşturulur. Bellek adresi de temel yazmaca geri yazılır. Başka bir deyişle, temel yazmaca değeri ofset değeri kadar artırılır veya azaltılır. Şekil 13.3b bu yöntemi assembly dilindeki `STRB r0, [r1, #12]!` talimatıyla göstermektedir. Ünlem işaretçi ön indeksleme anlamına gelir.

- Postindex:** Bellek adresi temel kayıt değeridir. Bir ofset, temel kayıt değerine eklenir veya ondan çıkarılır ve sonuç temel kayda geri yazılır. Şekil 13.3c bu yöntemi assembly dilinde STRB r0, [r1], #12 komutu ile göstermektedir.

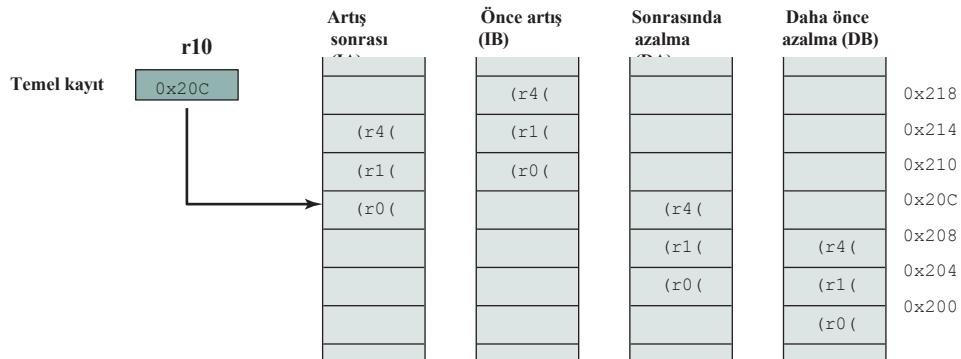
ARM'nin temel kayıt olarak adlandırdığı şeyin, ön indeks ve son indeks adresleme için bir indeks kaydı görevi gördüğünü unutmayın. Ofset değeri, komutta saklanan anlık bir değer olabileceği gibi başka bir yazmaçta da olabilir. Ofset değeri bir yazmaçta ise, başka bir kullanışlı özellik mevcuttur: ölçeklendirilmiş yazmaç adresleme. Ofset yazmacındaki değer kaydırma operatörlerinden biri tarafından ölçeklendirilir: Mantıksal Sola Kaydırma, Sağa Kaydırma, Aritmetik Sağa Kaydırma, Sağa Döndürme veya Genişletilmiş Sağa Döndürme (döndürmeye taşıma bitini de dahil eder). Kaydırma miktarı komutta bir anlık değer olarak belirtilir.

**VERİ İŞLEME TALİMATI ADRESLEME** Veri işleme talimatları ya register adresleme ya da register ve immediate adreslemenin bir karışımını kullanır. Yazmaç adresleme için, yazmaç işlenenlerinden birindeki değer, önceki paragrafta tanımlanan beş kaydırma işlecinden biri kullanılarak ölçeklendirilebilir.

DALLANMA Dallanma talimatları için tek adresleme şekli anlık . Dallanma komutu 24 bitlik bir değer içerir. Adres hesaplaması için bu değer 2 bit sola kaydırılır, böylece adres bir word sınırında olur. Böylece etkin aralığı program sayacından itibaren {32 MB'dır.

**LOAD/STORE ÇOKLU ADRESLEME** Load Çoklu komutları, genel amaçlı kayıtların bir alt kümesini (muhtemelen tümünü) bellekten yükler. Store Çoklu komutlar genel amaçlı yazmaçların bir alt kümesini (muhtemelen tümünü) belleğe depolar. Yüklenen ya da saklanacak yazmaçların listesi, her biti 16 yazmaçtan birine karşılık gelecek şekilde komutta 16 bitlik bir alanda belirtilir. Yükle ve Sakla Çoklu adresleme modları sıralı bir bellek adresleri aralığı üretir. En düşük numaralı yazmaç en düşük bellek adresinde ve en yüksek numaralı yazmaç en yüksek bellek adresinde saklanır. Dört adresleme modu kullanılır (Şekil 13.4): sonra artırma, önce artırma, sonra azaltma ve önce azaltma. Bir taban

```
LDMxx r10, {r0, r1, r4}
STMxx r10, {r0, r1, r4}
```



Şekil 13.4 ARM Yükle/Depola Çoklu Adresleme

register, register değerlerinin artan (artan) veya azalan (azalan) word konumlarında saklandığı veya yükleniği bir ana bellek adresini belirtir. Artırma ya da azaltma işlemi ilk bellek erişiminden önce ya da sonra başlar.

Bu talimatlar blok yüklemeler veya depolamalar, yoğun işlemleri ve prosedür çıkış dizileri için kullanışlıdır.

## 13.3 TALIMAT FORMATLARI

Bir komut biçimi, kendisini oluşturan alanlar açısından bir komutun bitlerinin düzenini tanımlar. Bir komut biçimini bir işlem kodu ve örtük ya da açık olarak sıfır ya da daha fazla işlenen içermelidir. Her açık işlenen, Bölüm 13.1'de açıklanan adresleme modlarından biri kullanılarak referanslanır. Biçim, her bir işlenen için adresleme modunu örtük ya da açık olarak belirtmelidir. Çoğu komut kümesi için birden fazla komut biçimi kullanılır.

Bir komut formatının tasarımını karmaşık bir sanattır ve inanılmaz çeşitlilikte tasarımlar uygulanmıştır. Temel tasarım konularını inceleyeceğiz, noktaları göstermek için bazı tasarımlara kısaca bakacağız ve ardından x86 ve ARM çözümlerini ayrıntılı olarak inceleyeceğiz.

### Talimat Uzunluğu

Karşılaşılması gereken en temel tasarım sorunu komut formatı uzunluğudur. Bu karar bellek boyutunu, bellek organizasyonunu, veri yolu yapısını, işlemci karmaşıklığını ve işlemci hızını etkiler ve bunlardan etkilenir. Bu karar, assembly dili programcısı tarafından görüldüğü gibi makinenin zenginliğini ve esnekliğini belirler.

Buradaki en belirgin değiş tokusu, güçlü bir talimat repertuarı arzusu ile yerden tasarruf etme ihtiyacı arasındadır. Programcılar daha fazla işlem kodu, daha fazla işlenen, daha fazla adresleme modu ve daha geniş adres aralığı isterler. Daha fazla işlem kodu ve daha fazla işlenen, programcının hayatını kolaylaştırır, çünkü verilen görevleri yerine getirmek için daha kısa programlar yazılabilir. Benzer şekilde, daha fazla adresleme modu, programcıya tablo manipülasyonları ve çok yönlü dallanma gibi belirli işlevlerin uygulanmasında daha fazla esneklik sağlar. Ve elbette, ana bellek boyutundaki artış ve sanal bellek kullanımının artmasıyla, programcılar daha geniş bellek aralıklarını adresleyebilmek isterler. Tüm bunlar (işlem kodları, işlenenler, adresleme modları, adres aralığı) bitler gerektirir ve daha uzun komut uzunlukları yönünde iter. Ancak daha uzun komut uzunluğu savurganlık olabilir. 64 bitlik bir komut 32 bitlik bir komutun iki katı yer kaplar ancak muhtemelen iki katından daha az kullanışlıdır.

Bu temel değişim tokusun ötesinde, başka hususlar da vardır. Ya talimat uzunluğu bellek aktarım uzunluğuna (veri yolu sisteminde veri yolu uzunluğu) eşit olmalı ya da biri diğerinin katı olmalıdır. Aksi takdirde, bir getirme döngüsü sırasında integral sayıda talimat alamayız. İlgili bir husus da bellek aktarım hızıdır. Bu hız işlemci hızındaki artışa ayak uyduramamıştır. Buna göre, işlemci komutları getirebildiğinden daha hızlı çalıştırılabilirse bellek bir darboğaz haline gelebilir. Bu sorunun çözümlerinden biri ön bellek kullanmaktadır (bkz. Bölüm 4.3); bir diğer ise daha kısa komutlar kullanmaktadır. Böylece, 16 bitlik talimatlar 32 bitlik talimatların iki katı hızda, ancak muhtemelen iki katından daha az hızla yürütülebilir.

Görünüşte sıradan ama yine de önemli bir özellik, talimat uzunluğunun genellikle 8 bit olan karakter uzunluğunun ve sabit noktalı sayıların uzunluğunun bir katı olması gerektidir. Buunu görebilmek için o pek de iyi tanımlanmamış kelime olan *word*'ü kullanmamız gereklidir [FRAI83]. Belleğin kelime uzunluğu bir anlamda organizasyonun "doğal" birimidir. Bir sözcüğün boyutu genellikle sabit noktalı sayıların boyutunu belirler (genellikle ikisi eşittir). Kelime boyutu aynı zamanda tipik olarak bellek aktarım boyutuna eşittir ya da en azından onunla bütünsel olarak ilişkilidir. Yaygın bir veri biçimini karakter verisi olduğundan, bir sözcüğün tam sayıda karakter saklamasını isteriz. Aksi takdirde, birden fazla karakter saklanırken her kelimedeki boş harcanan bitler olur ya da bir karakter bir kelime sınırını aşmak sorunda kalır. Bu noktanın önemi o kadar büyükür ki IBM, System/360'ı tanıttığında ve 8 bitlik karakterler kullanmak istediğiinde, 700/7000 serisinin bilimsel üyelerinin 36 bitlik mimarisinden 32 bitlik bir mimariye geçmek gibi sarsıcı bir karar almıştır.

### **Bitlerin Tahsisı**

Talimat formatının uzunluğuna karar vermede etkili olan bazı faktörleri inceledik. Aynı derecede zor bir konu da bu formattaki bitlerin nasıl tahsis edileceğidir. Buradaki ö dünlüşimler karmaşıktır.

Belirli bir komut uzunluğu için, işlem kodu sayısı ile adresleme kapasitesinin gücü arasında açıkça bir değişim tokus vardır. Daha fazla işlem kodu, işlem kodu alanında daha fazla bit anlamına gelir. Belirli uzunluktaki bir komut formatı için bu, adresleme için mevcut bit sayısını azaltır. Bu değişim tokusla ilginç bir iyileştirme vardır ve bu da değişken uzunluklu işlem kodlarının kullanılmasıdır. Bu yaklaşımın, minimum bir işlem kodu uzunluğu vardır, ancak bazı işlem kodları için, komutta ek bitler kullanılarak ek işlemler belirtilebilir. Sabit uzunlukta bir komut için bu, adresleme için daha az bit bırakır. Dolayısıyla, bu özellik daha az işlenen ve/veya daha az güçlü adresleme gerektiren komutlar için kullanılır.

Aşağıdaki birbiriley ilişkili faktörler adresleme bitlerinin kullanımını belirler.

- **Adresleme modlarının sayısı:** Bazen bir adresleme modu dolaylı olarak belirtilebilir. Örneğin, belirli işlem kodları her zaman indeksleme gerektirebilir. Diğer durumlarda, adresleme modları açık olmalıdır ve bir veya daha fazla mod bitine ihtiyaç duyulacaktır.
- **İşlenen sayısı:** Daha az adresin daha uzun, daha garip programlar oluşturabileceğini gördük (örneğin, Şekil 12.3). Günümüz makinelerindeki tipik komut formatları iki operand içerir. her bir operand adresi kendi mod göstergesini gerektirebilir veya mod göstergesinin kullanımı adres alanlarından sadece biriyle sınırlanır.
- **Kaydedicilere karşı bellek:** Bir makine, verilerin işlenmek üzere işlemciye getirilebilmesi için yazmaçlara sahip olmalıdır. Kullanıcı tarafından görülebilen tek bir yazmaçla (genellikle akümülatör olarak adlandırılır), bir işlenen adresi örtüktür ve hiçbir komut biti içermez. Bununla birlikte, tek kaydedicili programlama gariptir ve çok sayıda komut gerektirir. Çoklu yazmaçlarda bile, yazmacı belirtmek için sadece birkaç bit gereklidir. Yazmaçların kullanılabileceği daha fazla

operand referansları, daha az bit gereklidir. Bir dizi çalışma, kullanıcı tarafından görülebilen toplam 8 ila 32 yazmacın arzu edildiğini göstermektedir [LUND77, HUCK83]. Coğu çağdaş mimaride en az 32 yazmaç bulunmaktadır.

- **Kayıt setlerinin sayısı:** Coğu çağdaş makinede bir genel amaçlı yazmaç seti bulunur ve bu sette tipik olarak 32 veya daha fazla yazmaç bulunur. Bu yazmaçlar veri depolamak için kullanılabileceği gibi yer değiştirme adreslemesi için adres depolamak amacıyla da kullanılabilir. Bazı mimariler, x86 da dahil olmak üzere, iki ya da daha fazla özel setten (veri ve yer değiştirme gibi) oluşan bir koleksiyona sahiptir. Bu ikinci yaklaşımın bir avantajı, sabit sayıda yazmaç için, işlevsel bir bölünmenin komutta kullanılmak üzere daha az bit gerektirmesidir. Örneğin, sekiz yazmaçtan oluşan iki sette, bir yazmacı tanımlamak için yalnızca 3 bit gereklidir; işlem kodu veya mod yazmacı hangi yazmaç setine başvurulduğunu belirleyecektir.
- **Adres aralığı:** Belleğe başvuran adresler için, başvurulabilecek adres aralığı adres bitlerinin sayısıyla ilişkilidir. Bu ciddi bir sınırlama getirdiğinden, doğrudan adresleme nadiren kullanılır. Yer değiştirmeli adresleme ile, aralık adres kaydının uzunluğuna kadar açılır. Buna rağmen, komutta nispeten çok sayıda adres biti gerektiren kayıt adresinden oldukça büyük yer değiştirmelere izin vermek hala uygundur.
- **Adres tanecikliliği:** Kayıtlar yerine belleği referans alan adresler için bir başka faktör de adreslemenin ayrıntı düzeyiyidir. 16 ya da 32 bitlik kelimeye sahip bir sistemde, bir adres tasarımcının seçimine göre bir kelimeye ya da bir bayta referans verebilir. Bayt adresleme karakter manipülasyonu için uygundur, ancak sabit boyutlu bir bellek için daha fazla adres biti gerektirir.

Dolayısıyla tasarımcı, dikkate alınması ve dengelemesi gereken bir dizi faktörle karşı karşıyadır. Çeşitli seçeneklerin ne kadar kritik olduğu açık değildir. Örnek olarak, yoğun kullanımı, genel amaçlı kayıtlar, akümülatör ve yalnızca bellekten kayda yaklaşımları dahil olmak üzere çeşitli komut biçimleri karşılaştırın bir çalışmadan [CRAG79] bahsediyoruz. Tutarlı bir dizi varsayımda kullanılarak, kod alanı veya yürütme süresinde önemli bir fark gözlenmemiştir.

Şimdi iki tarihi makine tasarıminın bu çeşitli faktörleri nasıl dengelediğine kısaca bakalım.

**PDP-8** Genel amaçlı bir bilgisayar için en basit komut tasarımlarından biri PDP-8 içindedir [BELL78b]. PDP-8 12 bitlik komutlar kullanır ve 12 bitlik sözcükler üzerinde çalışır. Tek bir genel amaçlı yazmaç, yani akümülatör vardır.

Bu tasarımin sınırlamalarına rağmen adresleme oldukça esnektir. Her bellek referansı 7 bit artı iki adet 1 bitlik değiştiriciden oluşur. Bellek, her biri  $2^7 = 128$  kelimeye sabit uzunlukta sayfalara . Adres hesaplaması, sayfa biti tarafından belirlendiği gibi sayfa 0'a veya geçerli sayfaya (bu komutu içeren sayfa) yapılan referanslara dayanır. İkinci değiştirici bit doğrudan ya da dolaylı adreslemenin kullanılacağını belirtir. Bu iki mod birlikte kullanılabilir, böylece dolaylı bir adres sayfa 0 veya geçerli sayfanın bir kelimesinde bulunan 12 bitlik bir adresdir. Buna ek olarak, sayfa 0'daki 8 özel sözcük otomatik indeks "kayıtları "dır. Bu konumlardan birine dolaylı bir referans yapıldığında, önde indeksleme gerçekleşir.

Şekil 13.5 PDP-8 komut formatını göstermektedir. 3 bitlik bir işlem kodu ve üç tür talimat vardır. 0'dan 5'e kadar olan işlem kodları için format tek adreslidir

Bellek referans talimatları											
Opcode	D/I	Z/C	Yer değiştirme								
0	2	3	4	5						11	
Giriş/çıkış talimatları											
1	1	0	Cihaz								
0	2	3						8	9	11	
Kayıt referans talimatları											
Grup 1 mikro talimatlar											
1	1	1	0	CLA	KLL	CMA	KML	RAR	RAL	BSW	IAC
0	1	2	3	4	5	6	7	8	9	10	11
Grup 2 mikro talimatlar											
1	1	1	0	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11
Grup 3 mikro talimatlar											
1	1	1	0	CLA	MQA	0	MQL	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11

D/I= Doğrudan/Dolaylı adres Z/C= Sayfa 0 veya Geçerli CLA= Akümülatörü Temizle CLL= Bağlantıyı Temizle CMA= CoMplement Akümülatör CML= CoMplement Bağlantı RAR= Akümülatörü Sağa Döndür RAL= Akümülatörü Sola Döndür BSW= Byte SWap

IAC= Artış AC Akümülatörü SMA= Eksi Akümülatörde Atla SZA= Sıfır Akümülatörde Atla SNL= Sıfır Olmayan Bağlantıda Atla RSS= Reverse Skip Sense OSR= Or with Switch Register HLT= HaLT MQA= Akümülatöre Çarpan Katsayı MQL= Çarpan Katsayı Yüksü

**Şekil 13.5** PDP-8 Komut Formatları

Bir sayfa biti ve bir dolaylı bit içeren bellek referans talimatı. Böylece, sadece altı temel işlem vardır. İşlem grubunu genişletmek için, opcode 7 bir register referansı veya *mikro talimat* tanımlar. Bu formatta, kalan bitler ek işlemleri kodlamak için kullanılır. Genel olarak, her bit belirli bir işlemi tanımlar (örneğin, akümülatörü temizle) ve bu bitler tek bir komutta birleştirilebilir. Mikro komut stratejisi DEC tarafından PDP-1'e kadar kullanılmıştır ve bir anlamda, Dördüncü Bölümde tartışılabacak olan günümüzün mikro programlı makinelerinin öncüsüdür. Opcode 6 I/O işlemidir; 6 bit 64 aygıtta birini seçmek için kullanılır ve 3 bit belirli bir I/O komutunu belirtir.

PDP-8 komut formatı oldukça verimlidir. Dolaylı adresleme, yer değiştirme adresleme ve indekslemeyi destekler. İşlem kodu uzantısının kullanımıyla birlikte toplamda yaklaşık 35 komutu destekler. Tasarımcılar, 12 bitlik komut uzunluğu kısıtlamaları göz önüne alındığında, daha iyisini yapamazlardı.

**PDP-10** PDP-8'in komut seti ile PDP-10'unki arasında keskin bir karşılıklık vardır. PDP-10, büyük ölçekli zaman paylaşımı bir sistem olarak tasarlanmış ve ek donanım masrafları söz konusu olabile sistemin programlanması kolaylaştırılmaya önem verilmiştir.

Komut setinin tasarılanmasında kullanılan tasarım ilkeleri arasında şunlar da vardı [BELL78c]:

- **Ortogonalite:** Ortogonalite, iki değişkenin birbirinden bağımsız olduğu bir ilkedir. Bir komut seti bağlamında, bu terim şunları belirtir

Bir komutun diğer öğelerinin işlem kodundan bağımsız olması (işlem kodu tarafından belirlenmemesi). PDP-10 tasarımcıları bu terimi, bir adresin işlem kodundan bağımsız olarak her zaman aynı şekilde hesaplandığı gerçekini tanımlamak için kullanmaktadır. Bu, adres modunun bazen kullanılan operatöre dolaylı olarak bağlı olduğu birçok makinenin aksine bir durumdur.

- **Bütünlük:** Her aritmetik veri tipi (tamsayı, sabit nokta, kayan nokta) eksiksiz ve aynı işlem setine sahip olmalıdır.
- **Doğrudan adresleme:** Programcıya hafıza organizasyonu yükü getiren taban artı yer değiştirme adreslemesinden doğrudan adresleme lehine kaçınılmıştır.

Bu ilkelerin her biri programlama kolaylığı ana hedefini ilerletir.

PDP-10 36 bit kelime uzunluğuna ve 36 bit komut uzunluğuna sahiptir. Sabit komut formatı Şekil 13.6'da gösterilmiştir. İşlem kodu 9 bit kaplar ve 512 işleme kadar izin verir. Aslında, toplam 365 farklı komut tanımlanmıştır. Çoğu komutun iki adresi vardır, bunlardan biri 16 genel amaçlı kayttan biridir. Böylece, bu işlenen referansı 4 bit kaplar. Diğer işlenen referansı 18 bitlik bir bellek adresi alanı ile başlar. Bu bir anlık operand ya da bellek adresi olarak kullanılabilir. İkinci kullanımında, hem indeksleme hem de dolaylı adreslemeye izin verilir. Aynı genel amaçlı yazmaçlar dizin yazmaçları olarak da kullanılır.

36 bitlik komut uzunluğu gerçek bir lüktür. Daha fazla işlem kodu elde etmek için akillîca şeyler yapmaya gerek yoktur; 9 bitlik bir işlem kodu alanı fazlasıyla yeterlidir. Adresleme de basittir. 18 bitlik adres alanı doğrudan adreslemeyi cazip hale getirir.  $2^{18}$ 'den daha büyük bellek boyutları için dolaylı adresleme sağlanır. Programının kolaylığı için, tablo manipülasyonu ve yinelemeli programlar için indeksleme sağlanmıştır. Ayrıca, 18 bitlik bir operand alanı ile, anında adresleme cazip hale gelir.

PDP-10 komut seti tasarımı daha önce listelenen hedefleri gerçekleştirmektedir [LUND77]. Programının ya da derleyicinin görevini, alanın verimsiz kullanımı pahasına kolaylaştırır. Bu, tasarımcılar tarafından yapılan bilinçli bir seçimdi ve bu nedenle kötü tasarım olarak suçlanamaz.

### Değişken Uzunluklu Talimatlar

Şimdije kadar incelediğimiz örneklerde tek bir sabit komut uzunluğu kullandık ve bu bağlamda ödünlendirmeleri dolaylı olarak tartıştık. Ancak tasarımcı bunun yerine farklı uzunluklarda çeşitli komut formatları sağlamayı seçebilir. Bu taktik, farklı işlem kodu uzunluklarına sahip geniş bir işlem kodu repertuarı sağlamayı kolaylaştırır. Adresleme, kayıt ve bellek referanslarının çeşitli kombinasyonları ve adresleme modları ile daha esnek. Değişken uzunluklu talimatlarla, bu birçok varyasyon verimli ve kompakt bir şekilde sağlanabilir.

Opcode	Kayıt Olun	I	Dizin kayıt olun	Bellek adresi	
0	8 9	12	14	17 18	35

I= indirect bit

**Şekil 13.6** PDP-10 Komut Formatı

Değişken uzunluklu komutlar için ödenmesi gereken başlıca bedel işlemcinin karmaşıklığındaki artıtır. Düşen donanım fiyatları, mikro programlama kullanımı (Dördüncü Bölümde ele alınmıştır) ve işlemci tasarım ilkelerinin anlaşılmasındaki genel artış, bunun ödenmesi gereken küçük bir bedel olmasına katkıda bulunmuştur. Ancak, RISC ve süperskalar makinelerin daha iyi performans sağlamak için sabit uzunluklu komutların kullanımından yararlanabileceğini göreceğiz.

Değişken uzunluklu komutların kullanılması, tüm komut kelime uzunluğuyla bütünsel olarak ilişkili olması isteyen ortadan kaldırır. İşlemci getirilecek bir sonraki komutun uzunluğunu bilmediğinden, tipik bir strateji en azından mümkün olan en uzun komuta eşit sayıda bayt veya kelime getirmektir. Bu, bazen birden fazla komutun getirildiği anlamına gelir. Ancak, Bölüm 14'te göreceğimiz gibi, bu her durumda izlenecek iyi bir stratejidir.

**PDP-11** PDP-11, 16 bitlik bir mini bilgisayarın kısıtlamaları dahilinde güçlü ve esnek bir komut seti sağlamak için tasarlanmıştır [BELL70].

PDP-11 sekiz adet 16 bitlik genel amaçlı kullanır. Bu yazmaçlardan ikisi ek öneme sahiptir: biri özel amaçlı yoğun işlemleri için yoğun işaretçisi olarak kullanılır ve biri de bir sonraki komutun adresini içeren program sayacı olarak kullanılır.

Şekil 13.7 PDP-11 komut formatlarını göstermektedir. Sıfır, bir ve iki adresli komut tiplerini kapsayan on üç farklı format kullanılmaktadır. İşlem kodunun uzunluğu 4 ila 16 bit arasında değişebilir. Kayıt referansları 6 bit uzunluğundadır. Üç bit yazmacı tanımlar ve kalan 3 bit adresleme modunu tanımlar. PDP-11 zengin bir adresleme modu setine sahiptir. Bazen yapıldığı gibi, adresleme modunu işlem kodu yerine işlenenle ilişkilendirmenin bir avantajı, herhangi bir adresleme modunun herhangi bir işlem koduya kullanılabilmesidir. Daha önce de belirtildiği gibi, bu bağımsızlık *orthogonalilik* olarak adlandırılır.

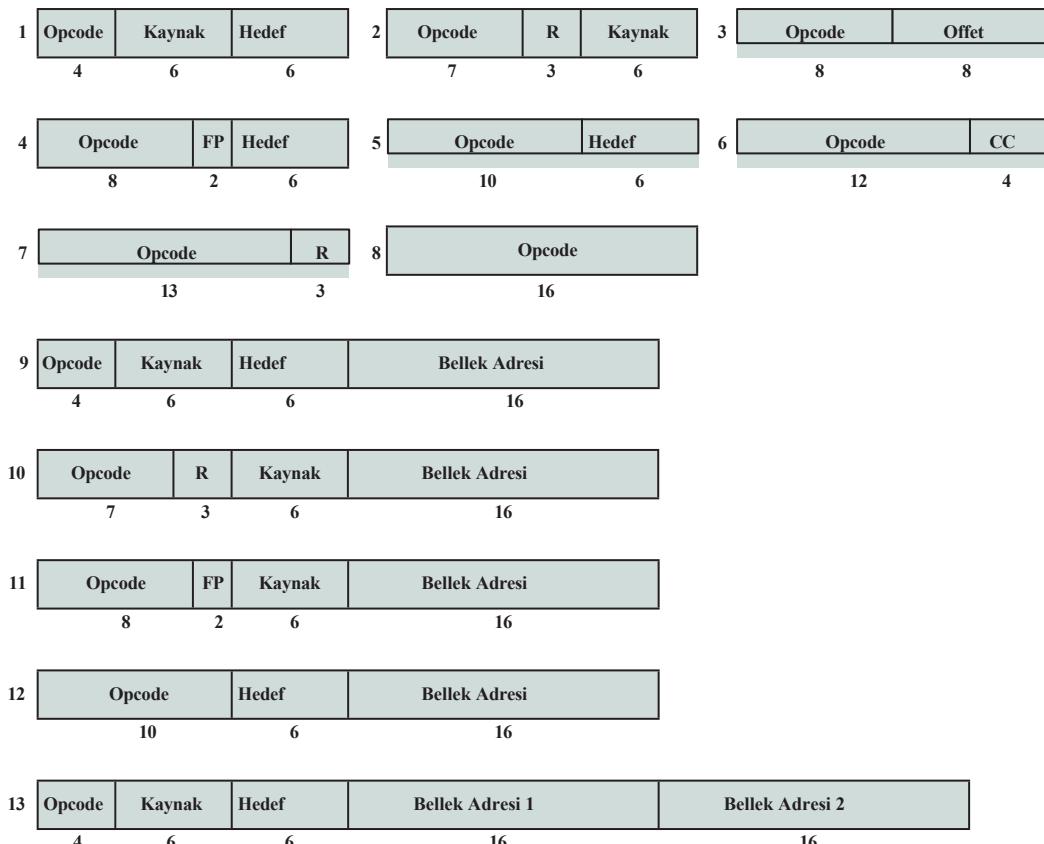
PDP-11 talimatları genellikle bir kelime (16 bit) uzunluğundadır. Bazı talimatlar için bir veya iki bellek adresi eklenir, böylece 32 bit ve 48 bit talimatlar repertuarın bir parçası olur. Bu, adreslemede daha fazla esneklik sağlar.

PDP-11 komut seti ve adresleme yeteneği karmaşıktır. Bu hem donanım malyetini hem de programlama karmaşıklığını artırır. Avantajı ise daha verimli veya kompakt programların geliştirilebilmesidir.

**VAX** Çoğu mimari nispeten az sayıda sabit komut biçimini sağlar. Bu, programcı için iki soruna neden olabilir. Birincisi, adresleme modu ve işlem kodu ortogonal değildir. Örneğin, belirli bir işlem için, bir işlenen bir yazmacıtan ve diğer bellekten veya her ikisi de yazmaçlardan gelmelidir ve bu böyle devam eder. İkincisi, yalnızca sınırlı sayıda işlenen barındırılabilir: tipik olarak en fazla iki veya üç. Bazı işlemler doğal olarak daha fazla işlenen gerektirdiğinden, iki veya daha fazla komut kullanarak istenen sonuca ulaşmak için çeşitli stratejiler kullanılmalıdır.

Bu sorumlardan kaçınmak için VAX formatı tasarılanırken iki kriter kullanılmıştır [STRE78]:

1. Tüm talimatlar "doğal" sayıda işleve sahip olmalıdır.
2. Tüm operandlar spesifikasyonda aynı genelliğe sahip olmalıdır.



Alanların altındaki sayılar bit uzunluğunu gösterir.

Kaynak ve hedefin her biri 3 bitlik bir adresleme modu alanı ve 3 bitlik bir kayıt numarası içerir. FP, dört kayan noktalı kayıtından birini gösterir.

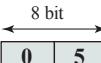
R genel amaçlı kaytlardan birini gösterir. CC durum kodu alanıdır.

#### Şekil 13.7 PDP-11 için Komut Formatları

Sonuç, oldukça değişken bir komut formatıdır. Bir komut, 1 veya 2 baytlık bir işlem kodundan ve ardından işlem koduna bağlı olarak sıfırdan altıya kadar işlenen belirticisinden oluşur. Minimum komut uzunluğu 1 bayttır ve 37 bayta kadar komutlar oluşturulabilir. Şekil 13.8'de birkaç örnek verilmiştir.

VAX komutu 1 baytlık bir işlem kodu ile başlar. Bu, çoğu VAX talimatını işlemek için yeterlidir. Ancak, 300'den fazla farklı komut olduğundan, 8 bit yeterli değildir. FD ve FF onaltılık kodları genişletilmiş bir işlem kodunu belirtir ve gerçek işlem kodu ikinci baytta belirtilir.

Komutun geri kalanı altı adede kadar işlenen belirticisinden oluşur. Bir operand belirteci, en azından, en soldaki 4 bitin adres modu belirteci olduğu 1 baytlık bir formattır. Bu kuralın tek istisnası literal moddur,

Onaltılık Biçim	Açıklama	Assembler Notasyonu ve Açıklaması												
 8 bit	RSB için işlem kodu	RSB Alt rutinden dönüş												
<table border="1" data-bbox="332 383 439 437"> <tr> <td>D</td><td>4</td></tr> <tr> <td>5</td><td>9</td></tr> </table>	D	4	5	9	CLRL için Opcode Kayıt R9	CLRL R9 R9 kaydını temizle								
D	4													
5	9													
<table border="1" data-bbox="332 517 439 704"> <tr> <td>B</td><td>0</td></tr> <tr> <td>C</td><td>4</td></tr> <tr> <td>6</td><td>4</td></tr> <tr> <td>0</td><td>1</td></tr> <tr> <td>A</td><td>B</td></tr> <tr> <td>1</td><td>9</td></tr> </table>	B	0	C	4	6	4	0	1	A	B	1	9	MOVW için işlem kodu Kelime yer değiştirme modu, Taşı a) kelime) (gelen) adres Kaydi 356 içeriği Bayt yer değiştirme modu, R11 Kaydi Onaltılık olarak 25	MOVW 356(R4), 25(R11) R4 bu 356 artı onaltılık olarak R4'ün adresinin 25) (arti) içerik of R11)
B	0													
C	4													
6	4													
0	1													
A	B													
1	9													
<table border="1" data-bbox="332 776 439 999"> <tr> <td>C</td><td>1</td></tr> <tr> <td>0</td><td>5</td></tr> <tr> <td>5</td><td>0</td></tr> <tr> <td>4</td><td>2</td></tr> <tr> <td>D</td><td>F</td></tr> <tr> <td colspan="2"></td></tr> </table>	C	1	0	5	5	0	4	2	D	F			ADDL3 için işlem kodu Kısa gerçek 5 Kayıt modu R0 Dizin öneki R2 Dolaylı kelime bağlı (PC'den yer değiştirme) A konumuna göre PC'den yer değiştirme miktarı	ADDL3 #5, R0, @A[R2] R0'daki 32 bitlik bir tamsayıya 5 ekleyin ve sonucu adresi A ve R2'nin içeriğinin 4 katının toplamı olan konumda saklayın
C	1													
0	5													
5	0													
4	2													
D	F													

Şekil 13.8 VAX Komutları Örneği

Bu da en soldaki 2 bitte 00 kalıbıyla belirtilir ve 6 bitlik bir değişmez için yer bırakır. Bu istisna nedeniyle, toplam 12 farklı adresleme modu belirtilebilir.

Bir operand belirteci genellikle sadece bir bayttan oluşur ve en sağdaki 4 bit 16 genel amaçlı kayıttan birini belirtir. İşlenen belirleyicisinin uzunluğu iki yoldan biriyle uzatılabilir. İlk olarak, bir veya daha fazla baytlık bir sabit değer, işlenen belirtecinin ilk baytını hemen takip edebilir. Bunun bir örneği, 8-, 16- veya 32-bit yer değiştirmenin kullanıldığı yer değiştirme modudur. İkinci olarak, bir indeks adresleme modu kullanılabilir. Bu durumda, işlenen belirtecinin ilk baytı 4 bitlik 0100 adresleme modu kodu ve 4 bitlik bir dizin kayıt tanımlayıcısından oluşur. İşlenen belirtecinin geri kalanı, kendisi bir veya daha fazla bayt uzunlığında olabilen temel adres belirtecisinden oluşur.

Okuyucu, yazarın yaptığı gibi, ne tür bir komutun altı işlenen gerektirdiğini merak ediyor olabilir. Şaşırıcı bir şekilde, VAX'te bu tür bir dizi talimat bulunmaktadır. Düşünün

ADDP6 OP1, OP2, OP3, OP4, OP5, OP6

Bu komut iki paketlenmiş ondalık sayı ekler. OP1 ve OP2 bir ondalık dizinin uzunluğunu ve başlangıç adresini belirtir; OP3 ve OP4 ikinci bir diziyi belirtir. Bu iki dizi toplanır ve sonuç, uzunluğu ve başlangıç konumu OP5 ve OP6 tarafından belirtilen ondalık dizide saklanır.

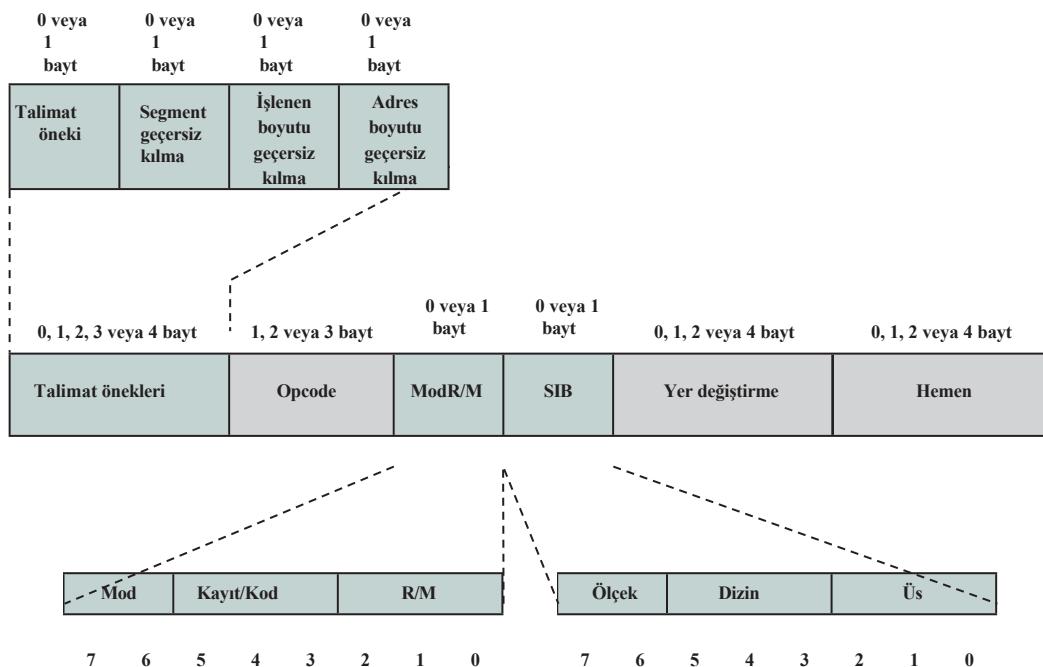
VAX komut seti çok çeşitli işlemler ve adresleme modları sağlar. Bu, derleyici yazılı gibi bir programcıya program geliştirmek için çok güçlü ve esnek bir araç verir. Teorik olarak, bu durum yüksek seviyeli dil programlarının verimli makine dili derlemelerine ve genel olarak işlemci kaynaklarının etkili ve verimli bir şekilde kullanılmasına yol açmalıdır. Bu faydalar için ödenmesi gereken ceza, daha basit komut seti ve formatına sahip bir işlemciye kıyasla işlemcinin artan karmaşıklığıdır.

Bu konulara, çok basit komut setleri için durumu incelediğimiz 15. Bölümde geri dönüyoruz.

## 13.4 x86 VE ARM YAPILANDIRMA FORMATLARI

### x86 Komut Formatları

x86 çeşitli komut formatları ile donatılmıştır. Bu alt bölümde açıklanan ögelerden yalnızca işlem kodu alanı her zaman mevcuttur. Şekil 13.9 genel komut formatını göstermektedir. Komutlar sıfırdan dörde kadar isteğe bağlı komut öneği, 1 veya 2 baytlık bir işlem kodu, isteğe bağlı bir adres belirteci (ModR/M bayti ve Ölçek İndeksi Tabanı baytından oluşur), isteğe bağlı bir yer değiştirme ve isteğe bağlı bir anlık alandan oluşur.



Şekil 13.9 x86 Komut Biçimi

İlk olarak önek baytlarını ele alalım:

- **Talimat önekleri:** Komut öneki, varsa, LOCK önekinden veya tekrar öneklerinden birinden oluşur. LOCK öneki, çok işlemcili ortamlarda paylaşılan belleğin dışlayıcı kullanımını sağlamak için kullanılır. Tekrar önekleri bir dizginin tekrarlanan işlemini belirtir, bu da x86'nın dizgileri normal bir yazılım döngüsünden çok daha hızlı işlemesini sağlar. Beş farklı tekrar öneki vardır: REP, REPE, REPZ, REPNE ve REPNZ. Mutlak REP öneki mevcut olduğunda, komutta belirtilen işlem dizenin ardışık öğeleri üzerinde tekrar tekrar yürütülür; tekrar sayısı CX kaydında belirtilir. Koşullu REP öneki, CX'teki sayı sıfır inene veya koşul karşılanması kadar komutun tekrarlanması neden olur.
- **Segment geçersiz kılma:** Bir hangi segment kaydını kullanması gerektiğini açıkça belirtir ve bu komut için x86 tarafından oluşturulan varsayılan segment kaydı seçimini geçersiz kılar.
- **İşlenen boyutu:** Bir komutun varsayılan işlenen boyutu 16 veya 32 bittir ve işlenen öneki 32 bit ve 16 bit işlenenler arasında geçiş yapar.
- **Adres boyutu:** İşlemci belleği 16 ya da 32 bitlik adresler kullanarak adresleyebilir. Adres boyutu, komutlardaki yer değiştirme boyutunu ve etkin adres hesaplaması sırasında oluşturulan adres offsetlerinin boyutunu belirler. Bu boyutlardan biri varsayılan olarak belirlenir ve adres boyutu öneki 32 bit ve 16 bit adres üretimi arasında geçiş yapar.

Talimatın kendisi aşağıdaki alanları içerir:

- **Opcode:** İşlem kodu alanı 1, 2 veya 3 bayt uzunluğundadır. İşlem kodu ayrıca verinin bayt mı yoksa tam boyutlu mu olduğunu (bağlama bağlı 16 veya 32 bit), veri işleminin yönünü (belleğe veya bellekten) ve anlık veri alanının işaret genişletmesi gerekip gerekmediğini belirten bitler içerebilir.
- **ModR/M:** Bu bayt ve sonraki bayt adresleme bilgisi sağlar. ModR/M baytı, bir işlenenin bir yazmaça mı yoksa bellekte mi olduğunu belirtir; bellekte ise, bayt içindeki alanlar kullanılacak adresleme modunu belirtir. ModR/M baytı üç alandan oluşur: Mod alanı (2 bit) R/M alanı ile birleşerek 32 olası değer oluşturur: 8 yazmaç ve 24 dizinleme modu; Reg/Opcode alanı (3 bit) ya bir yazmaç numarasını ya da üç bit daha opcode bilgisini belirtir; R/M alanı (3 ) bir işlenenin konumunu olarak bir yazmaç belirtebilir ya da Mod alaniyla birlikte adresleme modu kodlamasının bir parçasını oluşturabilir.
- **SIB:** ModR/M baytinin belirli kodlaması, adresleme modunu tam olarak belirtmek için SIB baytinin dahil edilmesini belirtir. SIB baytı üç alandan oluşur: Scale alanı (2 bit) ölçekli indeksleme için ölçek faktörünü belirtir; Index alanı (3 bit) indeks kaydını belirtir; Base alanı (3 bit) baz belirtir.
- **Yer değiştirme:** Adresleme modu belirticisi bir yer değiştirme kullanıldığını belirttiğinde, 8, 16 veya 32 bitlik bir işaretli tamsayı yer değiştirme alanı eklenir.
- **Immediate:** 8-, 16- veya 32 bitlik bir işlenenin değerini sağlar.

Burada birkaç karşılaştırma faydalı olabilir. x86 biçiminde, adresleme modu her bir işlenenle birlikte değil, işlem kodu dizisinin bir parçası olarak sağlanır.

Yalnızca bir işlenen adres modu bilgisine sahip olabileceğiinden, bir komutta yalnızca bir bellek işleneni referans alınabilir. Buna karşın VAX, adres modu bilgisini her bir işlenenle birlikte taşıyarak bellekten belleğe işlemlere olanak tanır. Bu nedenle x86 komutları daha kompakttır. Ancak, bellekten belleğe bir işlem gerekiyorsa, VAX bunu tek bir komutla gerçekleştirebilir.

x86 formatı, indeksleme için yalnızca 1 baytlık değil, aynı zamanda 2 baytlık ve 4 baytlık ofsetlerin de kullanılmasına izin verir. Daha büyük indeks uzaklıklarının kullanılması daha uzun komutlara neden olsa da, bu özellik gerekli esnekliği sağlar. Örneğin, büyük dizilerin veya büyük yığın çerçevelerinin adreslenmesinde yararlıdır. Buna karşılık, IBM S/370 talimat formatı 4 Kbyte'tan (12 bit ofset bilgisi) daha büyük olmayan ofsetlere izin verir ve ofset pozitif olmalıdır. Bir konum bu ofsete erişemediğinde, derleyicinin gerekli adresi oluşturmak için ekstra kod üretmesi gereklidir. Bu sorun özellikle 4 Kbyte'tan fazla yer kaplayan yerel değişkenlere sahip yığın çerçeveleri ile uğraşırken ortaya çıkmaktadır. DEWA90]'ın belirttiği gibi, "370 için kod üretmek bu kısıtlamanın bir sonucu olarak o kadar zahmetlidir ki, 370 için yığın çerçevesinin boyutunu 4 Kbyte ile sınırlamayı seçen derleyiciler bile olmuştur."

Gördüğü gibi, x86 komut setinin kodlanması çok karmaşıkta. Bu kısmen 8086 makinesi ile geriye dönük uyumlu olma ihtiyacı ve kısmen de tasarımcıların derleyici yazarına verimli kod üretmede mümkün olan her türlü yardımı sağlama arzusu ile ilgilidir. Bu kadar karmaşık bir komut setinin RISC komut setlerinin tam tersi bir uç noktaya tercih edilip edilmeyeceği tartışmalıdır.

## ARM Komut Formatları

ARM mimarisindeki tüm komutlar 32 bit uzunluğundadır ve düzenli bir format izler (Şekil 13.10). Bir komutun ilk dört biti koşul kodudur. Bölüm 12'de anlatıldığı gibi, neredeyse tüm ARM komutları koşullu olarak çalıştırılabilir. Sonraki üç bit komutun genel tipini belirtir. Dallanma komutları dışındaki çoğu komut için, sonraki beş bit bir işlem kodu ve/veya işlem için değiştirici bitleri oluşturur. Kalan 20 bit işlenen adresleme içindir. Komut formatlarının düzenli yapısı, komut kod çözme birimlerinin işini kolaylaştırır.

**IMMEDIATE CONSTANTS** Daha geniş bir immediate değer aralığı elde etmek için, veri işleme immediate formatı hem bir immediate değer hem de bir döndürme değeri belirtir. 8 bitlik anlık değer 32 bite genişletilir ve ardından 4 bitlik döndürme değerinin iki katına eşit sayıda bit sağa döndürülür. Şekil 13.11'de birkaç örnek gösterilmektedir.

**THUMB INSTRUCTION SET** Thumb komut seti, ARM komut setinin yeniden kodlanmış bir alt kümeleridir. Thumb, 16 bit veya daha dar bellek veri yolu kullanan ARM uygulamalarının performansını artırmak ve hem 16 bit hem de 32 bit işlemciler için ARM komutunun sağladığından daha iyi kod yoğunluğuna izin vermek için tasarlanmıştır. Thumb komut seti, 32 bit ARM komut setinin analiz edilmesi ve en uygun 16 bit komut setinin türetilmesiyle oluşturulmuş ve böylece kod boyutu azaltılmıştır. Tasarruf aşağıdaki şekilde sağlanmıştır:

1. Thumb komutları koşulsuzdur, bu nedenle koşul kodu alanı kullanılmaz. Ayrıca, tüm Thumb aritmetik ve mantık komutları koşul bayraklarını günceller, böylece update-flag bitine gerek kalmaz. Tasarruflar: 5 bit.

S= Veri işleme talimatları için, talimatın koşul kodlarını güncellediğini belirtir

S= Çoklu yükleme/depolama talimatları için, talimat yürütmenin gözetmen moduyla sınırlanıp sınırlanmadığını belirtir.

P, U, W = farklı adresleme modu türleri  
arasında ayrim yapan bitler

B = İşaretsiz bayt ( $B==1$ ) ve sözcük ( $B==0$ ) erişimi arasında ayırım yapar

$L = \text{Yükleme/depolama talimatları için, bir Yükleme}$   
 $(L==1) \text{ ve bir Depolama (L==0) arasında ayrımlı vurur.}$

L= Dallanma talimatları için, bir dönüş adresinin bağlantı kaydında saklanıp saklanmayacağı belirler.

### Sekil 13.10 ARM Komut Formatları

2. Thumb, tam komut setindeki işlemlerin yalnızca bir alt kümesine sahiptir ve yalnızca 2 bitlik bir işlem kodu alanı ve 3 bitlik bir tür kullanır. Tasarruf: 2 bit.
  3. Kalan 9 bitlik tasarruf, işlenen belirtimlerindeki azaltmalardan kaynaklanmaktadır. Örneğin, Thumb komutları yalnızca  $r0$ 'dan  $r7$ 'ye kadar olan yazmaçları referans alır, bu nedenle yazmaç referansları için 4 bit yerine yalnızca 3 bit gereklidir. Immediate değerler 4 bitlik bir dönürtme alanı içermez.

**ror #0-arabık 0 ila 0x000000FF-adım 0x00000001**

**Sekil 13.11** ARM Immediate Sabitlerinin Kullanım Örnekleri

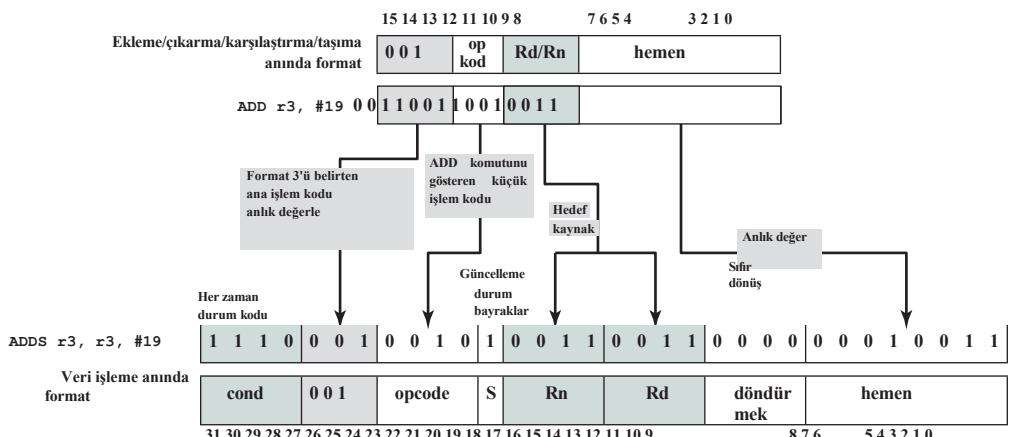
ARM işlemci, Thumb talimatları ve 32 bit ARM talimatlarının karışımından oluşan bir programı yürütebilir. İşlemci kontrol yazmacındaki bir bit, o anda hangi komut türünün yürütülmekte olduğunu belirler. Şekil 13.12'de bir örnek gösterilmektedir. Şekilde hem genel biçim hem de bir komutun hem 16 bit hem de 32 bit biçimlerindeki belirli bir örneği gösterilmektedir.

**THUMB-2 INSTRUCTION SET** Thumb komut setinin kullanılmaya başlanmasıyla birlikte, kullanıcının performans açısından kritik kodları ARM'a, geri kalanını ise Thumb'a derleyerek komut setlerini harmanlaması gerekmıştır. Bu manuel kod harmanlama işlemi ek çaba gerektirir ve en iyi sonuçları elde etmek zordur. Bu sorunların üstesinden gelmek için ARM, Cortex-M mikrodenetleyici ürünlerinde mevcut olan tek komut seti olan Thumb-2 komut setini geliştirmiştir.

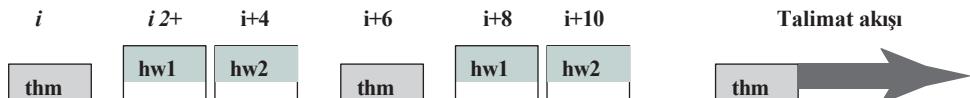
Thumb-2, Thumb komut seti mimarisinde (ISA) yapılan büyük bir geliştirmedir. Eski 16-bit Thumb ile serbestçe karıştırılabilen 32-bit talimatlar sunar. Bu yeni 32-bit komutlar ARM komut setinin neredeyse tüm işlevlerini kapsar. Thumb ISA ve ARM ISA arasındaki en önemli fark, 32-bit Thumb talimatlarının çoğunun koşulsuz olması, buna karşın neredeyse tüm ARM talimatlarının koşullu olabilmesidir. Bununla birlikte, Thumb-2, ARM talimatlarındaki koşul alanının işlevselliliğinin çoğunu sağlayan yeni bir If-Then (IT) talımı sunar. Thumb-2, ARM ISA ile ilişkili performans seviyeleriyle birlikte Thumb ile karşılaştırılabilir genel kod yoğunluğu sunar. Thumb-2'den önce, geliştiriciler boyut için Thumb ve performans için ARM arasında seçim yapmak zorundaydı.

[ROBI07], Thumb-2 komut setinin ARM ve orijinal Thumb komut setleriyle karşılaştırıldığı bir analizi rapor etmiştir. Analiz, üç komut setini kullanarak Gömülü Mikroişlemci Kiyaslama Konsorsiyumu (EEMBC) kıyaslama paketinin derlenmesini ve çalıştırılmasını içeriyordu ve aşağıdaki sonuçlar elde edildi:

- Performans için optimize edilmiş derleyicilerle, Thumb-2 boyutu ARM'den %26 daha küçük ve orijinal Thumb'dan biraz daha büyüktü.
  - Alan için optimize edilmiş derleyicilerle, Thumb-2 boyutu ARM'den %32 daha küçük ve orijinal Thumb'dan biraz daha küçüktü.



**Sekil 13.12** Bir Thumb ADD Komutunun ARM Esdeğeri Genişletilmesi



Yarım Kelime1 [15:13]	Yarım Kelime1 [12:11]	Uzunluk	İşlevsellik
111 değil	xx	16 bit (1 yarı kelime)	16 bit Thumb talimi
111	00	16 bit (1 yarı kelime)	16 bit Thumb koşulsuz dallanma talimi
111	Degisil 00	32 bit (2 yarı kelime)	32 bit Thumb-2 talimi

**Şekil 13.13** Thumb-2 Kodlaması

- Performans için optimize edilmiş derleyicilerle, Thumb-2'nin kıyaslama takımındaki performansı ARM performansının %98'i ve orijinal Thumb performansının %125'i olmuştur.

Bu sonuçlar Thumb-2'nin tasarım hedeflerini karşıladığıını doğrulamaktadır.

Şekil 13.13 yeni 32-bit Thumb komutlarının nasıl kodlandığını göstermektedir. Kodlama, komutun en sol beş bitinde 11100 bit desenine sahip olan mevcut Thumb koşulsuz dallanma komutlarıyla uyumludur. Başka hiçbir 16 bitlik komut en soldaki üç bitte 111 kalibriyle başlamaz, bu nedenle 11101, 11110 ve 11111 bit kalıpları bunun 32 bitlik bir Thumb komutu olduğunu gösterir.

## 13.5 MONTAJ DİLİ

Bir işlemci makine talimatlarını anlayabilir ve yürütübilir. Bu talimatlar basitçe bilgisayarda saklanan ikili sayılardır. Eğer bir programcı doğrudan makine dilinde yapmak isterse, o zaman programı ikili veri olarak girmek gereklidir.

Basit BASIC ifadesini düşünün

$$N = I + J + K$$

Bu ifadeyi makine dilinde programlamak ve I, J ve K'yı sırasıyla 2, 3 ve 4 olarak başlatmak istediğimizi varsayıyalım. Bu Şekil 13.14a'da gösterilmiştir. Program 101 (onaltılık) konumunda başlar. Beltek 201 konumundan başlayarak dört değişken için ayrılmıştır. Program dört talimattan oluşmaktadır:

- Konum 201'in içeriğini AC'ye yükleyin.
- Konum 202'nin içeriğini AC'ye ekleyin.
- Konum 203'un içeriğini AC'ye ekleyin.
- AC'nin içeriğini 204 numaralı konumda saklayın.

Bunun sıkıcı ve hataya çok açık bir süreç olduğu açıktır.

Programı ikili gösterim yerine onaltılık olarak yazmak küçük bir iyileştirmeyidir (Şekil 13.14b). Programı bir dizi satır olarak yazabiliz. Her bir

Contents					İçindekiler	
Adres					Adres	
101	0010	0010	101	2201	101	2201
102	0001	0010	102	1202	102	1202
103	0001	0010	103	1203	103	1203
104	0011	0010	104	3204	104	3204
201	0000	0000	201	0002	201	0002
202	0000	0000	202	0003	202	0003
203	0000	0000	203	0004	203	0004
204	0000	0000	204	0000	204	0000

(a) İkili program

(b) Onaltılık program

Adres	Talimatlar	Etiket	Operasyon	Operand
101	LDA 201	FORMÜL	LDA	I
102	ADD 202		ADD	J
103	ADD 203		ADD	K
104	STA 204		STA	N
201	DAT 2	I	VERİ	2
202	DAT 3	J	VERİ	3
203	DAT 4	K	VERİ	4
204	DAT 0	N	VERİ	0

(c) Sembolik program

(d) Montaj programı

**Şekil 13.14** N Formülünün Hesaplanması = I + J + K

satırı bir bellek konumunun adresini ve bu konumda saklanacak ikili değerin onaltılık kodunu içerir. O zaman bu girdiyi kabul edecek, her satırı ikili sayıya çevirecek ve belirtilen konumda saklayacak bir programa ihtiyacımız var.

Daha fazla iyileştirme için, her komutun sembolik adını veya anımsatıcısını kullanabiliriz. Bu, Şekil 13.14c'de gösterilen *sembolik programla* sonuçlarıdır. Her girdi satırı hala bir bellek konumunu temsil etmektedir. Her satır boşluklarla ayrılmış üç alandan oluşur. İlk alan bir konumun adresini içerir. Bir komut için, ikinci alan işlem kodu için üç harfli sembolü içerir. Eğer bu bir bellek referanslı komut ise, üçüncü alan adresi içerir. Bir konumda rastgele veri depolamak için, DAT simgesi ile bir sözde komut icat ederiz. Bu sadece satırındaki üçüncü alanın ilk alanda belirtilen konumda saklanacak onaltılık bir sayı içerdigini gösterir.

Bu tür bir girdi için biraz daha karmaşık bir programa ihtiyacımız var. Program her girdi satırını kabul eder, ikinci ve üçüncü (varsayı) alanlara dayalı olarak ikili bir sayı üretir ve bunu ilk alan tarafından belirtilen konumda saklar.

Sembolik bir programın kullanılması hayatı çok daha kolaylaştırır ancak yine de gariptir.

Özellikle, her kelime için mutlak bir adres vermemeliyiz. Bu, programın ve verilerin bellekte yalnızca tek bir yere yüklenmesi gereklidir ve bu yeri bilmemiz gerektiği anlamına gelir. Daha da kötüsü, bir gün bir satır ekleyerek ya da silerek programı değiştirmek istediğimizi varsayıyalım. Bu, sonraki tüm kelimelerin adreslerini değiştirecektir. Çok daha iyi ve yaygın olarak kullanılan bir sistem sembolik adresler kullanmaktadır.

Bu durum Şekil 13.14d'de gösterilmektedir. Her satır hala üç alandan oluşmaktadır. İlk alan hala adres içindir, ancak mutlak bir sayısal adres yerine bir sembol kullanılır. Bazı satırların adresi yoktur, bu da o satırın adresinin bir

bir önceki satırın adresinden daha fazla. Bellek referanslı talimatlar için üçüncü alan da sembolik bir adres içerir.

Bu son iyileştirme ile bir *assembly diline* sahip oluruz. Assembly dilinde yazılmış programlar (assembly programları) bir *assembler* tarafından makine diline çevrilir. Bu program sadece daha önce tartışılan sembolik çeviriyi yapmakla kalmamalı, aynı zamanda sembolik adreslere bir çeşit bellek adresi atamalıdır.

Assembly dilinin geliştirilmesi bilgisayar teknolojisinin evriminde önemli bir dönüm noktası olmuştur. Bugün kullanılan yüksek seviyeli dillere atılan ilk adımdı. Çok az programcı assembly dilini kullanır da, neredeyse tüm makineler bir assembly dili sağlamaktadır. Derleyiciler ve I/O rutinleri gibi sistem programları için kullanılır.

Ek B, assembly dilinin daha ayrıntılı bir incelemesini sunmaktadır.

## 13.6 ANAHTAR TERİMLER, GÖZDEN GEÇİRME SORULARI VE PROBLEMLER

### Anahtar Terimler

otomatik indeksleme temel kayıt adresleme doğrudan adresleme yer değiştirme adresleme etkili adres	acil adresleme indeksleme dolaylı adresleme talimat formати postindexing	ön endeksleme kayıt adresleme kayıt dolaylı adresleme göreli adresleme kelime
--	--	---

### İnceleme Soruları

- 13.1 Doğrudan adreslemeyi kısaca tanımlayınız.
- 13.2 **Doğrudan adreslemeyi** kısaca tanımlayınız.
- 13.3 Dolaylı adreslemeyi kısaca tanımlayınız.
- 13.4 Kayıt adreslemeyi kısaca tanımlayınız.
- 13.5 Kayıt dolaylı adreslemeyi kısaca tanımlayınız.
- 13.6 Yer değiştirme adreslemesini kısaca tanımlayınız.
- 13.7 Göreli adreslemeyi kısaca tanımlayınız.
- 13.8 Otomatik indekslemenin avantajı nedir?
- 13.9 Postindexing ve preindexing arasındaki fark nedir?
- 13.10 Bir taliminin adresleme bitlerinin kullanımını belirleyen gerçekler nelerdir?
- 13.11 Değişken uzunluklu komut formatı kullanmanın avantajları ve dezavantajları nelerdir?

### Problemler

- 13.1 Aşağıdaki bellek değerleri ve akümülatörlü tek adresli bir makine göz önüne alındığında, aşağıdaki talimatlar akümülatöre hangi değerleri yükler?
  - Kelime 20, 40 içerir.
  - Kelime 30, 50 içerir.
  - Kelime 40, 60 içerir.
  - Kelime 50, 70 içerir.

- a. HEMEN YÜKLE 20
  - b. YÜK DOĞRUDAN 20
  - c. YÜK DOLAYLI 20
  - d. HEMEN YÜKLE 30
  - e. YÜK DOĞRUDAN 30
  - f. YÜK DOLAYLI 30
- 13.2** Program sayacında saklanan adres X1 simbolü ile gösterilsin. X1'de saklanan komutun bir adres parçası (operand referansı) X2'dir. Komutu çalıştırmak için gereken işlenen, X3 adresli bellek sözcüğünde saklanır. Bir indeks yazmacı X4 değerini içerir. Komutun adresleme modu (a) doğrudan; b) dolaylı; (c) PC göreli; (d) indeksli ise bu çeşitli büyütükler arasındaki ilişki nedir?
- 13.3** Bir komuttaki adres alanı ondalık 14 değerini içermektedir. Buna karşılık gelen operand nerede bulunur?
- a. acil adresleme?
  - b. doğrudan adresleme?
  - c. dolaylı adresleme?
  - d. kayıt adresleme?
  - e. kayıt dolaylı adresleme?
- 13.4** Ana bellekte 200 konumundan başlayarak aşağıdakilerin göründüğü 16 bitlik bir işlemci düşünün:

200	AC'ye Yük	Mod
201	500	
202	Sonraki talimat	

İlk kelimenin ilk kısmı, bu komutun bir akümülatöre değer yüklediğini gösterir. Mode alanı bir adresleme modunu belirtir ve uygunsa bir kaynak yazmacını gösterir; kullanıldığındaysa kaynak yazmacının 400 değerine sahip R1 varsayılmı. Ayrıca 100 değerini içeren bir temel yazma da vardır. 201 konumundaki 500 değeri adres hesaplamasının bir parçası olabilir. 399'un 999 değerini, konum 400'ün 1000 değerini içerdigini ve böyle devam ettiğini varsayıñ. Aşağıdaki adres modları için etkin adresi ve yüklenen işleneni belirleyin:

- a. Doğrudan
  - b. Hemen
  - c. Dolaylı
  - d. PC akrabası
  - e. Yer değiştirme
  - f. Kayıt Olun
  - g. Dolaylı kayıt
  - h. R1 kullanarak artış ile otomatik indeksleme
- 13.5** PC-göreceli mod dallanma talimi 3 bayt uzunluğundadır. Komutun ondalık olarak adresi 256028'dir. Komuttaki işaretli yer değiştirme **-3 1** ise dallanma hedef adresini belirleyin.
- 13.6** Bir PC-göreceli mod dallanma talimi bellekte  $620_{10}$  adresinde saklanır. Dallanma  $530_{10}$  konumuna yapılır. Komuttaki adres alanı 10 bit uzunluğundadır. Komuttaki ikili değer nedir?
- 13.7** Eğer komut (a) tek bir işlenen gerektiren bir hesaplama; (b) bir dallanma ise, işlemcinin dolaylı adres modlu bir komutu getirip çalıştırırken kaç kez belleğe başvurması gereklidir?
- 13.8** IBM 370 dolaylı adresleme sağlamaz. Bir işlenenin adresinin ana bellekte olduğunu varsayıñ. İşlenene nasıl erişirsiniz?
- 13.9** COOK82]de yazar, PC-göreceli adresleme modlarının, yoğun kullanımı gibi diğer modlar lehine elimine edilmesini önermektedir. Bu önerinin dezavantajı ?

**13.10** x86 aşağıdaki talimatı içerir:

IMUL op1, op2, acil

Bu komut, yazmaç ya da bellek olabilen op2'yi yakın operand değeri ile çarpar ve sonucu yazmaç olması gereken op1'e yerleştirir. Komut kümelerinde bu türden başka bir üç işlenenli komut yoktur. Böyle bir komutun olası kullanımını nedir? (*İpucu: İndekslemeyi düşünün.*)

**13.11** Indeksleme adresleme moduna sahip bir taban içeren bir işlemci düşünün. Bu adresleme modunu kullanan ve ondalık olarak 1970'lik bir yer belirleyen bir komutla karşılaşıldığını varsayalım. Şu anda taban ve indeks kayıtları sırasıyla 48,022 ve 8 ondalık sayılarını içeriyor. İşlenenin adresi nedir?

**13.12** Tanımlayın: EA=  $(X) +$ , etkin adres hesaplandıktan sonra bir sözcük uzunluğu kadar X konumunun içeriğine eşit etkin adresdir; EA=  $-(X)$ , etkin adres önce X bir sözcük uzunluğu kadar azaltılarak X konumunun içeriğine eşit etkin adresdir; EA=  $(X) -$ , etkin adres hesaplandıktan sonra X bir sözcük uzunluğu kadar azaltılarak X konumunun içeriğine eşit etkin adresdir. Her biri (İşlem Kaynak İşlenen, Hedef İşlenen) biçiminde olan ve işlemin sonucu hedef işlenene yerleştirilen aşağıdaki talimatları göz önünde bulundurun.

- OP X,  $(X)$
- OP  $(X)$ ,  $(X) +$
- OP  $(X) +$ ,  $(X)$
- OP  $-(X)$ ,  $(X)$
- OP  $-(X)$ ,  $(X) +$
- OP  $(X) +$ ,  $(X) +$
- OP  $(X) -, (X)$

Yığın işaretçisi olarak X'i kullanarak, bu hangisi yığından en üstteki iki öğeyi alabilir, belirlenen işlemi gerçekleştirebilir (örneğin, kaynağı hedefe EKLE ve hedefte sakla) ve sonucu yığına geri itebilir? Bu tür her bir yapı için, yığın bellek konumu 0'a doğru mu yoksa ters yönde mi büyür?

**13.13** PUSH ve POP yığın işlemlerini içeren yığın yönelikli bir işlemci varsayıyalım. Aritmetik işlemler otomatik olarak en üstteki bir ya da iki yığın elemanını içerir. Boş bir yığınla başlayın. Aşağıdaki talimatlar yürütüldükten sonra hangi yığın elemanları kalır?

İTME	4
İTME	7
İTME	8
ADD	
İTME	10
SUB	
MUL	

**13.14** 32-bitlik bir komutun 16-bitlik bir komuttan muhtemelen iki kat daha az kullanışlı olduğu iddiasını gerekçelendirin.

**13.15** IBM'in kelime başına 36 bitten 32 bite geçme kararı neden kimin için üzücüydü?

**13.16** Sabit 16 bit komut uzunluğu kullanan bir komut seti varsayıyalım. İşlenen özellikleri 6 bit uzunluğundadır.  $K$  adet iki operandlı ve  $L$  adet sıfır operandlı komut vardır. Desteklenebilecek maksimum tek-işlemeli komut sayısı nedir?

**13.17** Aşağıdakilerin tümünün 36 bitlik bir komutta kodlanması sağlamak için değişken uzunluklu bir işlem kodu tasarlayın:

- iki adet 15 bitlik adres ve bir adet 3 bitlik kayıt numarası içeren talimatlar;
- 15 bitlik bir adres ve 3 bitlik bir kayıt numarası içeren talimatlar;
- adresleri veya kayıtları olmayan talimatlar.

- 13.18** Problem 10.6'nın sonuçlarını göz önünde bulundurun. M'nin 16 bitlik bir bellek adresi olduğunu ve X, Y ve Z'nin 16 bitlik adresler ya da 4 bitlik yazmaç numaraları olduğunu varsayıyın. Tek adresli makine bir akümülatör kullanır ve iki ve üç adresli makinelerde 16 yazmaç ve bellek konumları ve yazmaçların tüm kombinasyonları üzerinde çalışan komutlar vardır. 8 işlem kodları ve 4 bitin katları olan komut uzunlukları varsayıldığında, her makinenin X'i hesaplamak için kaç bite ihtiyacı vardır?
- 13.19** İki işlem kodlu bir talimat için herhangi bir gerekçe var mı?
- 13.20** 16-bit Zilog Z8001 aşağıdaki genel komut formatına sahiptir:

15	14	13	12	11	10 9 8 7 6 5 4 3 2 1 0	
Mod	Opcode			w/b	Operand 2	Operand 1

*Mod* alanı, *operand* alanlarından *operandların* nasıl bulunacağını belirtir. *w/b* alanı belirli komutlarda işlenenlerin bayt mı yoksa 16 bitlik sözcükler mi olduğunu belirtmek için kullanılır. İşlenen 1 alanı (*mod* alanı içeriğine bağlı olarak) 16 genel amaçlı kayıttan birini belirtebilir. İşlenen 2 alanı, yazmaç 0 hariç herhangi bir genel amaçlı yazmaç belirtebilir. İşlenen 2 alanının tümü sıfır olduğunda, orijinal işlem kodlarının her biri yeni bir anlam kazanır.

- a. Z8001'de kaç tane işlem kodu bulunmaktadır?
- b. Daha fazla işlem kodu sağlamak için etkili bir yol önerin ve ilgili değişim tokusu belirtin.

# 14

## BÖLÜM

### İŞLEMCI YAPISI VE İŞLEVI

#### 14.1 İşlemci Organizasyonu

#### 14.2 Kayıt Kuruluşu

- Kullanıcı Tarafından Görülebilen Kayıtlar Kontrol ve Durum Kayıtları Örnek Mikroişlemci Kayıt Organizasyonları

#### 14.3 Talimat Döngüsü

- Dolaylı Döngü Veri Akışı

#### 14.4 Talimat Pipelining

- Pipelining Stratejisi Pipeline Performansı Pipeline Tehlikeleri Dallarla Başa Çıkma Intel 80486 Pipelining

#### 14.5 x86 İşlemci Ailesi

- Kayıt Organizasyonu Kesme İşlemi

#### 14.6 Kol İşlemcisi

- İşlemci Organizasyonu İşlemci Modları Kayıt Organizasyonu Kesme İşleme

#### 14.7 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler

### ÖĞRENİM HEDEFLERİ

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- Kullanıcı tarafından **görülebilen** ve **kontrol/durum kayıtları** arasında ayrı yapabilecek ve her bir kategorideki kayıtların amaçlarını tartışabilecektir.
- **Komut döngüsünü** özetleyin.
- **Komut ardışık diziliminin** arkasındaki prensibi ve pratikte nasıl çalıştığını tartışınız.
- Boru hattı tehlikelerinin çeşitli biçimlerini karşılaştırın ve kıyaslayın.
- x86 işlemci yapısına genel bir bakış sunar.
- ARM işlemci yapısına genel bir bakış sunar.

Bu bölüm, işlemcinin henüz Üçüncü Bölüm'de ele alınmayan yönlerini tartışmakta ve Bölüm 15 ve 16'daki RISC ve superscalar mimari tartışmasına zemin hazırlamaktadır.

İşlemci organizasyonunun bir özeti ile başlıyoruz. Daha sonra işlemcinin dahili belleğini oluşturan yazımcılar analiz edilmektedir. Daha sonra komut döngüsü tartışmasına (Bölüm 3.2'de başlamıştık) geri donecek konumdayız. Komut döngüsünün bir tanımı ve komut arası ilişkili olarak bilinen yaygın bir teknik tamamlar. Bölüm, x86 ve ARM organizasyonlarının bazı yönlerinin incelenmesi ile sona ermektedir.

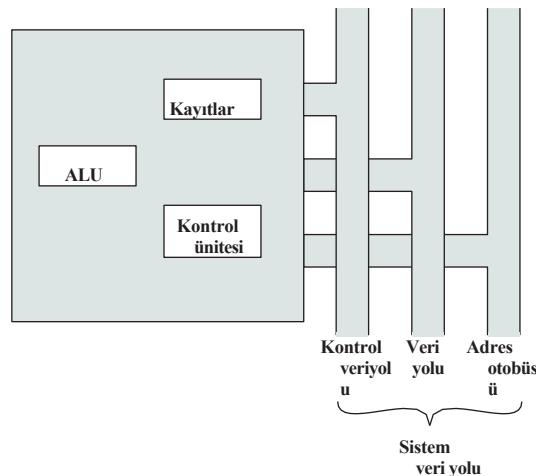
## 14.1 İŞLEMCİ ORGANİZASYONU

İşlemcinin organizasyonunu anlamak için, işlemciye yüklenen gereksinimleri, yapması gereken şeyleri göz önünde

- **Komut getirme:** İşlemci bellekten (kayıt, önbellek, ana bellek) bir komut okur.
- **Talimatı yorumlayın:** Hangi eylemin gerekli olduğunu belirlemek için talimatın kodu çözülür.
- **Veri getirme:** Bir komutun yürütülmesi için bellekten veya bir G/Ç modülünden veri okunması gerekebilir.
- **Veri işleme:** Bir komutun yürütülmesi, veriler üzerinde bazı aritmetik veya mantıksal işlemlerin yapılmasını gerektirebilir.
- **Veri yazma:** Bir yürütmenin sonuçları, verilerin belleğe veya bir G/Ç modülüne yazılmasını gerektirebilir.

Bunları yapmak için, işlemcinin bazı verileri geçici olarak saklaması gereği açık olmalıdır. Bir sonraki komutu nereden alacağını bilebilmesi için son yerini hatırlaması gereklidir. Bir komut yürütülürken komutları ve verileri geçici olarak saklaması gereklidir. Başka bir deyişle, işlemcinin küçük bir dahili belleğe ihtiyacı vardır.

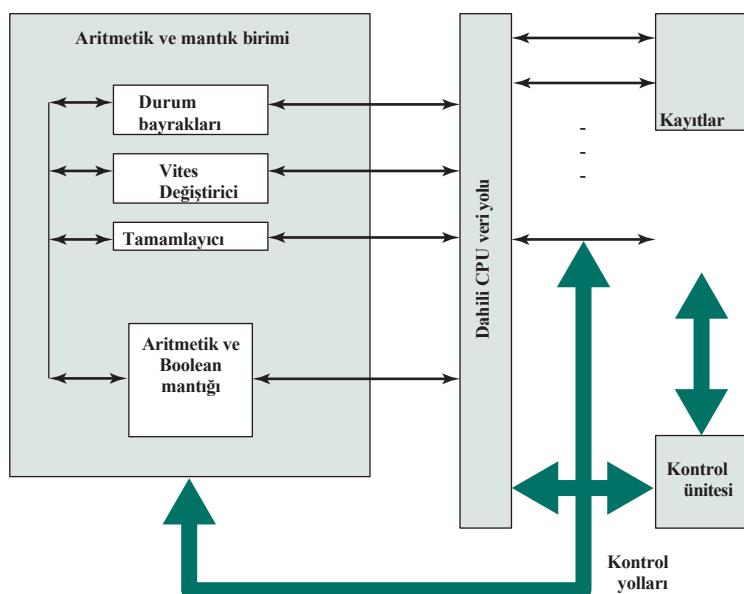
Şekil 14.1, bir işlemcinin basitleştirilmiş bir görünümüdür ve sistem veriyolu aracılığıyla sistemin geri kalıyla bağlantısını gösterir. Benzer bir arayüz herhangi bir işlemci için de gereklidir.



Şekil 14.1 Sistem Veriyolu ile CPU

Bölüm 3'te açıklanan ara bağlantı yapıları. Okuyucu, işlemcinin ana bileşenlerinin bir *aritmetik ve mantık birimi* (ALU) ve bir *kontrol birimi* (CU) olduğunu hatırlayacaktır. ALU gerçek hesaplamayı ya da veri işlemeyi yapar. Kontrol ünitesi veri ve talimatların işlemcinin içine ve dışına hareketini ve ALU'nun çalışmasını kontrol eder. Buna ek olarak, şekilde *register* adı verilen bir dizi depolama konumundan oluşan minimal bir dahili bellek gösterilmektedir.

Şekil 14.2 işlemcinin biraz daha ayrıntılı bir görünümüdür. Veri aktarım ve mantık kontrol yolları, *dahili* olarak etiketlenmiş bir eleman da dahil olmak üzere gösterilmiştir



Şekil 14.2 CPU'nun İç Yapısı

*işlemci veri yolu.* Bu eleman, çeşitli kaydediciler ve ALU arasında veri aktarımı için gereklidir çünkü ALU aslında sadece dahili işlemci belleğindeki veriler üzerinde çalışır. Şekilde ayrıca ALU'nun tipik temel elemanları da gösterilmektedir. Bir bütün olarak bilgisayarın iç yapısı ile işlemcinin iç yapısı arasındaki benzerlige dikkat edin. Her iki durumda da, veri yolları ile birbirine bağlanan küçük bir ana elemanlar topluluğu (bilgisayar: işlemci, G/Ç, bellek; işlemci: kontrol birimi, ALU, kaydediciler) vardır.

## 14.2 KAYIT ORGANİZASYONU

Bölüm 4'te tartıştığımız gibi, bir bilgisayar sistemi bir bellek hiyerarşisi kullanır. Hiyerarşinin daha yüksek seviyelerinde bellek daha hızlı, daha küçük ve daha pahalıdır (bit başına). İşlemci içinde, hiyerarşide ana bellek ve önbelleğin üzerinde bir bellek seviyesi olarak işlev gören bir dizi yazmaç vardır. İşlemeideki kayıtlar iki rol oynar:

- **Kullanıcı tarafından görülebilir kayıtlar:** Makine veya montaj dili programcısının kullanımını optimize ederek ana bellek referanslarını en aza indirmesini sağlar.
- **Kontrol ve durum kayıtları:** İşlemeinin çalışmasını kontrol etmek için kontrol birimi tarafından ve programların yürütülmesini kontrol etmek için ayrıcalıklı, işletim sistemi programları tarafından kullanılır.

Kayıtların bu iki kategoriye ayrılması kesin değildir. Örneğin, bazı makinelerde program sayacı kullanıcı tarafından görülebilir (örneğin, x86), ancak birçogunda görülemez. Bununla birlikte, aşağıdaki tartışmanın amaçları doğrultusunda, bu kategorileri kullanacağız.

### Kullanıcı Tarafından Görünür Kayıtlar

Kullanıcı tarafından görülebilen bir yazmaç, işlemeinin yürüttüğü makine dili aracılığıyla başvurulabilen bir yazmaçtır. Bunları aşağıdaki kategorilerde karakterize edebiliriz:

- Genel amaçlı
- Veri
- Adres
- Durum kodları

**Genel amaçlı yazmaçlar** programcı tarafından çeşitli işlevlere atanabilir. Bazen komut kümesi içindeki kullanımları işlemle ortogonaldır., herhangi bir genel amaçlı yazmaç herhangi bir işlem kodu için işlenen içerebilir. Bu gerçek genel amaçlı yazmaç kullanımı sağlar. Bununla birlikte, genellikle kısıtlamalar vardır. Örneğin, kayan nokta ve yığın işlemleri için özel yazmaçlar olabilir.

Bazı durumlarda, genel amaçlı yazmaçlar adresleme işlevleri için kullanılabilir (örneğin, dolaylı yazmaç, yer değiştirme). Diğer durumlarda, veri yazmaçları ile adres yazmaçları arasında kısmi ya da tam bir ayrım vardır. **Veri yazmaçları** yalnızca veri tutmak için kullanılabilir ve bir işlenen adresinin hesaplanmasıında kullanılamaz.

**Adres kayıtlarının** kendileri genel amaçlı olabilir ya da belirli bir adresleme moduna ayrılmış . Örnekler aşağıdakileri içerir:

- **Segment işaretçileri:** Segmentli adreslemeye sahip bir makinede (bkz. Bölüm 8.3), bir segment yazmacı segment tabanının adresini tutar. Birden fazla yazmaç : örneğin, biri işletim sistemi için diğerleri süreç için.
- **Dizin kayıtları:** Bunlar indeksli adresleme için kullanılır ve otomatik indekslenebilir.
- **Yığın işaretçisi:** Kullanıcı tarafından görülebilen yığın adreslemesi varsa, tipik olarak yığının en üstünü gösteren özel bir kayıt . Bu, örtük adreslemeye izin verir; , push, pop ve diğer yığın komutlarının açık bir yığın operandı içermesi gerekmekz.

Burada ele alınması gereken birkaç tasarım sorunu vardır. Önemli bir konu tamamen genel amaçlı yazmaçların mı kullanılacağı yoksa kullanımlarının özelleştirileceğidir. Komut seti tasarımını etkilediği için bu konuya bir önceki bölümde değinmiştık. Özelleşmiş yazmaçların kullanımıyla, genellikle işlem kodunda belirli bir işlenen belirticisinin hangi yazmaç türünü ifade ettiği belirtilebilir. İşlenen belirteci, tüm yazmaçlardan biri yalnızca bir dizi özel yazmaçtan birini tanımlamalıdır, böylece bit tasarrufu sağlanır. Öte yandan, bu özelleştirme programcının esnekliğini sınırlar.

Bir diğer tasarım konusu da, genel amaçlı ya da veri artı adres olmak üzere sağlanacak yazmaç sayısıdır. Bu da komut seti tasarımını etkiler çünkü daha fazla yazmaç daha fazla işlenen belirleyici bit gerektirir. Daha önce tartıştığımız gibi, 8 ila 32 yazmaç arasında bir yer optimum görünümektedir [LUND77]. Daha az yazmaç daha fazla bellek referansına neden olur; daha fazla yazmaç bellek referanslarını belirgin bir şekilde azaltmaz (örneğin, bkz. [WILL90]). Bununla birlikte, çok sayıda yazmaç kullanımında avantaj sağlayan yeni bir yaklaşım bazı RISC sistemlerinde sergilenmektedir ve Bölüm 15'te tartışılmaktadır.

Son olarak, kayıt uzunluğu sorunu vardır. Adresleri tutması gereken yazmaçların en azından en büyük adresi tutacak kadar uzun olması gereklidir. Veri yazmaçları çoğu veri türünün değerlerini tutabilmelidir. Bazı makineler, çift uzunluktaki değerleri tutmak için iki bitistik yazmacın tek bir yazmaç olarak kullanılmasına izin verir.

Kullanıcı tarafından en azından kısmen görülebilen son bir kayıt kategorisi, **koşul kodlarını** (*bayraklar* olarak da adlandırılır) tutar. Durum kodları, işlemlerin sonucu olarak kaydedici donanımı tarafından ayarlanan bitlerdir. Örneğin, bir aritmetik işlem pozitif, negatif, sıfır veya taşıma sonucu üretебilir. Sonucun kendisinin bir kayıtta veya bellekte saklanması ek olarak, bir koşul kodu da ayarlanır. Kod daha sonra koşullu dallanma işleminin bir parçası olarak test edilebilir.

Durum kodu bitleri bir veya daha fazla kaydedicide toplanır. Genellikle bir kontrol kaydının parçasını oluştururlar. Genel olarak, makine talimatları bu bitlerin örtük referansla okunmasına izin verir, ancak programcı bunları değiştiremez.

IA-64 mimarisine ve MIPS işlemcilerine dayalı olanlar da dahil olmak üzere birçok işlemci koşul kodlarını hiç kullanmaz. Bunun yerine, koşullu dallanma talimatları yapılacak bir karşılaştırmayı belirtir ve bir koşul kodu saklamadan karşılaşırsonucuna göre hareket eder. DERO87]ye dayanan Tablo 14.1, koşul kodlarının temel avantaj ve dezavantajlarını listeler.

**Tablo 14.1** Durum Kodları

Avantajlar	Dezavantajlar
<ol style="list-style-type: none"> <li>1. Koşul kodları normal aritmetik ve veri hareketi tarafından ayarlandığından, gerekli KARŞILAŞTIRMA ve TEST komutlarının sayısını azaltmalarıdır.</li> <li>2. BRANCH gibi koşullu talimatlar, TEST ve BRANCH gibi bileşik talimatlara göre basitleştirilmiştir.</li> <li>3. Koşul kodları çok yönlü dallanmaları kolaylaştırır. Örneğin, bir TEST komutunu, biri sıfırdan küçük veya eşit, diğerisi sıfırdan büyük olmak üzere iki dallanma izleyebilir.</li> <li>4. Durum kodları, diğer kayıt bilgileriyle birlikte alt rutin çağrıları sırasında yiğine kaydedilebilir.</li> </ol>	<ol style="list-style-type: none"> <li>1. Durum kodları hem donanıma hem de yazılıma karmaşık katar. Durum kodu bitleri genellikle farklı talimatlar tarafından farklı şekillerde değiştirilir, bu da hem mikro programcı hem de derleyici yazılımı daha zor hale getirir.</li> <li>2. Durum kodları düzensizdir; tipik olarak ana veri yolunun bir parçası değildirler, bu nedenle ekstra donanım bağlantıları gerektirirler.</li> <li>3. Genellikle koşul kodu makineleri, bit kontrolü, döngü ve atomik semafor işlemleri gibi özel durumlar için koşul kodu olmayan özel talimatlar eklemek zorundadır</li> <li>4. Boru hattı uygulamasında, koşul kodları çakışmaları önlemek için özel senkronizasyon gerektirir.</li> </ol>

Bazı makinelerde, bir alt rutin çağrısı, kullanıcı tarafından görülebilen tüm kayıtların, dönüste geri yüklenmek üzere otomatik olarak kaydedilmesiyle sonuçlanır. İşlemci, kaydetme ve geri yükleme işlemlerini çağrı ve dönüş komutlarının yürütülmesinin bir parçası olarak gerçekleştirir. Bu, her alt rutinin kullanıcı tarafından görülebilen kayıtları bağımsız olarak kullanmasına olanak tanır. Diğer makinelerde, bir alt rutin çağrılarından önce ilgili kullanıcı tarafından görülebilen kayıtların içeriğini kaydetmek, programa bu amaçla talimatlar ekleyerek programcının sorumluluğundadır.

### Kontrol ve Durum Kayıtları

İşlemcinin çalışmasını kontrol etmek için kullanılan çeşitli işlemci kayıtları vardır. Bunların çoğu, çoğu makinede, kullanıcı tarafından görülemez. Bazıları kontrol veya işletim sistemi modunda çalıştırılan makine talimatları tarafından görülebilir.

Elbette, farklı makineler farklı kayıt organizasyonlarına sahip olacak ve farklı terminoloji kullanacaktır. Burada, kısa bir açıklama ile birlikte kayıt türlerinin makul ölçüde eksiksiz bir listesini listeliyoruz.

Komutların yürütülmesi için dört yazmacı gereklidir:

- **Program sayacı (PC):** Getirilecek bir komutun adresini içerir.
- **Talimat kaydı (IR):** En son getirilen komutu içerir.
- **Bellek adres kaydı (MAR):** Bellekteki bir konumun adresini içerir.
- **Bellek tampon kaydı (MBR):** Belleğe yazılacak bir veri sözcüğünü veya en son okunan sözcüğü içerir.

Tüm işlemcilerde MAR ve MBR olarak adlandırılan dahili kayıtlar bulunmaz, ancak aktarılacak bitler için eşdeğer bir tamponlama mekanizmasına ihtiyaç vardır

sistem veriyoluna giden bitler kademelendirilir ve veri yolundan okunacak bitler geçici olarak saklanır.

Tipik olarak, işlemci her komut getirme işleminden sonra PC'yi günceller, böylece PC her zaman yürütülecek bir sonraki komutu gösterir. Bir dallanma veya atlama talimatı da PC'nin içeriğini değiştirecektir. Getirilen komut, işlem kodu ve işlenen belirticilerinin analiz edildiği bir IR'ye yüklenir. Veri, MAR ve MBR kullanılarak bellek ile değiştirilir. Veri yolu ile organize edilmiş bir sistemde, MAR doğrudan adres veri yoluna bağlanır ve MBR doğrudan veri yoluna bağlanır. Kullanıcı tarafından görülebilen yazmaçlar da MBR ile veri alışverişinde bulunur.

Az önce bahsedilen dört yazmaç, verilerin işlemci ve bellek arasında taşınması için kullanılır. İşlemci içinde, veriler işlenmek üzere ALU'ya sunulmalıdır. ALU'nun MBR'ye ve kullanıcı tarafından görülebilen yazmaçlara doğrudan erişimi olabilir. Alternatif olarak, ALU'nun sınırında ek tamponlama kayıtları olabilir; bu kayıtlar ALU için giriş ve çıkış kayıtları olarak hizmet eder ve MBR ve kullanıcı tarafından görülebilir kayıtlarla veri alışverişi yapar.

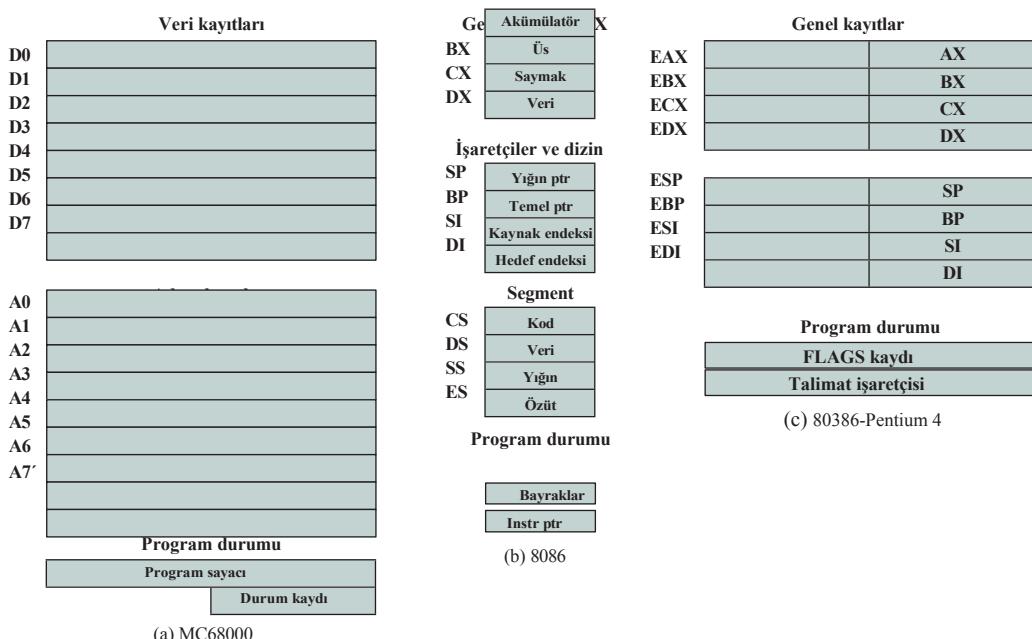
Birçok işlemci tasarıımı, genellikle *program durum kelimesi* (PSW) olarak bilinen ve durum bilgisi içeren bir kayıt ya da kayıt kümesi içerir. PSW tipik olarak durum kodlarının yanı sıra diğer durum bilgilerini de içerir. Yaygın alanlar veya bayraklar aşağıdakileri içerir:

- **İşaret:** Son aritmetik işlemin sonucunun işaret bitini içerir.
- **Sıfır:** Sonuç 0 olduğunda ayarlanır.
- **Carry:** Bir işlem yüksek mertebeden bir bitin içine taşıma (toplama) veya dışına ödünc alma (alt çekme) ile sonuçlandıysa ayarlanır. Çok kelimeli aritmetik işlemler için kullanılır.
- **Eşitlik:** Mantıksal karşılaşılma sonucu eşitlik ise ayarlanır.
- **Taşma:** Aritmetik taşımayı belirtmek için kullanılır.
- **Kesme Etkinleştirme/Devre Dışı Bırakma:** Kesmeleri etkinleştirmek veya devre dışı bırakmak için kullanılır.
- **Gözetmen:** İşlemcinin gözetmen modunda mı yoksa kullanıcı modunda mı çalıştığını gösterir. Belirli ayrıcalıklı talimatlar yalnızca süpervizör modunda yürütülebilir ve belirli bellek alanlarına yalnızca süpervizör modunda erişilebilir.

Belirli bir işlemci tasarıımında durum ve kontrolle ilgili bir dizi başka kayıt bulunabilir. Ek durum bilgisi içeren bir bellek bloğuna bir işaretçi olabilir (örneğin, işlem kontrol blokları). Vektörlü kesmeler kullanan makinelerde, bir kesme vektörü kaydı sağlanabilir. Belirli işlevleri uygulamak için bir yoğun kullanılıyorsa (örneğin, alt rutin çağrısı), bir sistem yiğimi işaretçisi gereklidir. Sanal bellek sistemi ile bir sayfa tablosu işaretçisi kullanılır. Son olarak, G/C işlemlerinin kontrolünde regis- ters kullanılabilir.

Kontrol ve durum kayıt organizasyonunun tasarıımında bir dizi faktör rol oynar. Önemli konulardan biri işletim sistemi desteğidir. Belirli kontrol bilgi türleri işletim sistemi için özel bir faydaya sahiptir. İşlemci tasarımcısı kullanılacak işletim sistemi hakkında işlevsel bir anlayışa sahipse, yazmaç organizasyonu bir dereceye kadar işletim sistemine göre uyarlanabilir.

Bir diğer önemli tasarım kararı da kontrol bilgilerinin yazmaçlar ve bellek arasında paylaşılmamasıdır. İlk (en düşük) birkaç yüz ya da birkaç yüz



**Şekil 14.3** Örnek Mikroişlemci Kayıt Organizasyonları

kontrol amaçlı bin kelimelik bellek. Tasarımcı ne kadar kontrol bilgisinin yazmaçlarda ve ne kadarının bellekte olması gerektigine karar vermelidir. Maliyet ve hız arasındaki olağan değişim tokus ortaya çıkar.

### Örnek Mikroişlemci Kayıt Organizasyonları

Karşılaştırılabilir sistemlerin yazmaç organizasyonunu incelemek ve karşılaştırmak öğreticidir. Bu bölümde, yaklaşık aynı zamanda tasarlanmış iki 16-bit mikroişlemciye bakacağız: Motorola MC68000 [STRI79] ve Intel 8086 [MORS78]. Şekil 14.3a ve b her birinin kayıt organizasyonunu göstermektedir, bellek adres kaydı gibi tamamen dahili kayıtlar gösterilmemiştir.

MC68000, 32 bit kayıtlarını sekiz veri kaydı ve dokuz adres kaydı olarak böülümlere ayırır. Sekiz veri kaydı öncelikle veri manipülasyonu için kullanılır ve ayrıca adreslemede indeks kayıtları olarak kullanılır. Kayıtların genişliği, işlem koduna göre belirlenen 8, 16 ve 32 bit veri işlemlerine izin verir. Adres yazmaçları 32 bitlik (böülüme yok) adresler içerir; bu yazmaçlardan ikisi, geçerli yürütme moduna bağlı olarak biri kullanıcılar ve diğer işletim sistemi için olmak üzere yığın işaretçisi olarak da kullanılır. Her iki kayıt da 7 olarak numaralandırılmıştır, çünkü aynı anda yalnızca bir tanesi kullanılabilir. MC68000 ayrıca 32 bitlik bir program sayacı ve 16 bitlik bir durum kaydı içerir. Motorola ekibi, özel amaçlı yazmaçlar içermeyen çok düzenli bir komut seti istiyordu. Kod verimliliği kaygısı, onları yazmaçları aşağıdakilere bölmeye yöneltti

iki işlevsel bileşen, her kayıt belirtecinde bir bit tasarruf sağlar. Bu, tam genellik ve kod sıkıştırma arasında makul bir uzlaşma gibi görülmektedir.

Intel 8086 register organizasyonuna farklı bir yaklaşım getirmektedir. Bazı yazmaçlar genel amaçlı olarak da kullanılabilmesine rağmen, her yazmaç özel . 8086, bayt ya da 16 bit bazında adreslenebilen dört adet 16 bit veri kaydedicisi ve dört adet 16 bit işaretçi ve dizin kaydedicisi içerir. Veri kayıtları bazı komutlarda genel amaçlı olarak kullanılabilir. Diğerlerinde, kayıtlar dolaylı olarak kullanılır. Örneğin, bir çarpma komutu her zaman akümülatörü kullanır. Dört işaretçi kaydedicisi de bir dizi işlemde dolaylı olarak kullanılır; her biri bir segment ofseti içerir. Ayrıca dört adet 16 bit segment kaydedicisi vardır. Dört segment register'ından üçü, sırasıyla geçerli komutun segmentine (dallanma komutları için kullanılmıştır), veri içeren bir segmente ve yığın içeren bir segmente işaret için özel ve örtük bir şekilde kullanılır. Bu özel ve örtük kullanımlar, esnekliğin azalması pahasına kompakt kodlama sağlar. 8086 ayrıca bir komut işaretcisini ve bir dizi 1 bitlik durum ve kontrol bayrağı içerir.

Bu karşılaşmanın amacı açık olmalıdır. İşlemci kayıtlarını düzenlemenin en iyi yolu konusunda evrensel olarak kabul edilmiş bir felsefe yoktur [TOON81]. Genel komut seti tasarımda ve diğer pek çok işlemci tasarım sorununda gibi, bu hala bir yargı ve zevk meselesiştir.

Kayıt organizasyonu tasarımları ile ilgili ikinci bir öğretici nokta Şekil 14.3c'de gösterilmiştir. Bu şekil, 8086'nın bir uzantısı olarak tasarlanmış 32 bitlik bir mikroişlemci olan Intel 80386 [ELAY85] için kullanıcı tarafından görülebilen yazmaç organizasyonunu göstermektedir.<sup>1</sup> 80386 32 bitlik yazmaçlar kullanır. Bununla birlikte, daha önceki makinelerde yazılan programlar için yukarı doğru uyumluluk sağlamak için 80386, yeni organizasyona gömülü orijinal kayıt organizasyonunu korur. Bu tasarım kısıtlaması göz önüne alındığında, 32 bit işlemcilerin mimarları kayıt organizasyonunu tasarlama konusunda sınırlı esnekliğe sahipti.

### 14.3 TALIMAT DÖNGÜSÜ

Bölüm 3.2'de işlemcinin komut döngüsünü tanımlamıştık (Şekil 3.9). Hatırlamak gerekirse, bir komut döngüsü aşağıdaki aşamaları içerir:

- **Getir:** Bir sonraki komutu bellekten işlemciye okur.
- **Yürütme:** İşlem kodunu yorumlayın ve belirtilen işlemi gerçekleştirin.
- **Kesme:** Kesmeler etkinleştirilmişse ve bir kesme meydana gelmişse, geçerli işlem durumunu kaydedin ve kesmeye hizmet verin.

Şimdi komut döngüsünü biraz daha detaylandıracak durumdayız. Öncelikle, dolaylı döngü olarak bilinen ek bir aşamayı tanıtmalıyız.

---

<sup>(1)</sup> MC68000 zaten 32 bit yazmaçlar kullandığından, tam bir 32 bit mimari olan MC68020 [MACD84] de aynı yazmaç organizasyonunu kullanır.

## Dolaylı Döngü

Bölüm 13'te, bir komutun yürütülmesinin bellekte her biri bir bellek erişimi gerektiren bir veya daha fazla işlenen içerebileceğini gördük. Ayrıca, dolaylı adresleme kullanılıyorsa, ek bellek erişimleri gereklidir.

Dolaylı adreslerin getirilmesini bir komut aşaması daha olarak düşünebiliriz. Sonuç Şekil 14.4'te gösterilmiştir. Ana faaliyet hattı, değişen komut getirme ve komut yürütme faaliyetlerinden oluşur. Bir komut getirildikten sonra, herhangi bir dolaylı adreslemenin söz konusu olup olmadığını belirlemek için incelenir. Eğer öyleyse, gerekli operandlar dolaylı adresleme kullanılarak getirilir. Yürütmenin ardından, bir sonraki komut getirme işleminden önce bir kesme işlenebilir.

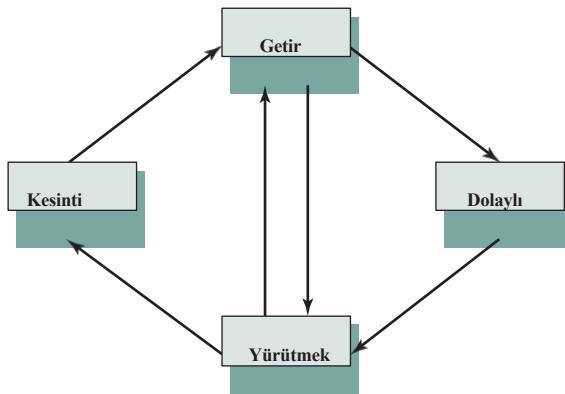
Bu süreci görmeyen bir başka yolu da Şekil 3.12'nin gözden geçirilmiş bir versiyonu olan Şekil 14.5'te gösterilmektedir. Bu, komut döngüsünün doğasını daha doğru bir şekilde göstermektedir. Bir komut getirildikten sonra, işlenen belirticileri tanımlanmalıdır. Bellekteki her bir giriş operandı daha sonra getirilir ve bu işlem dolaylı adresleme gerektirebilir. Register tabanlı operandların getirilmesine gerek yoktur. İşlem kodu sonra, sonucu ana bellekte saklamak için benzer bir işlem gerekebilir.

## Veri Akışı

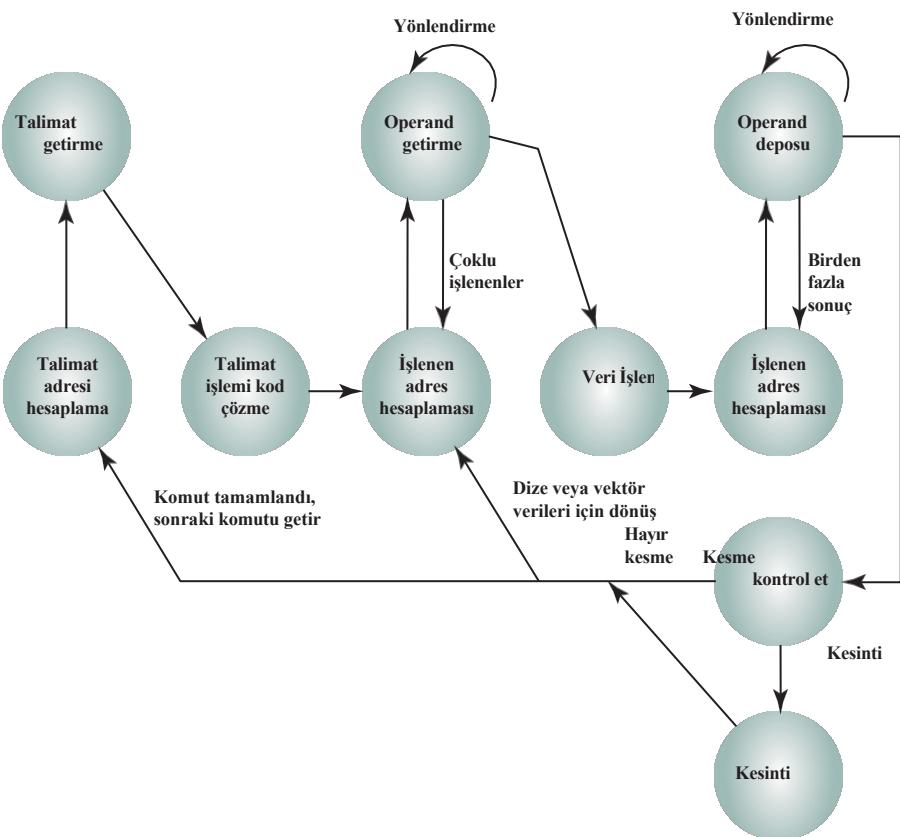
Bir komut döngüsü sırasında olayların tam sırası işlemcinin tasarımasına bağlıdır. Bununla birlikte, ne gerektiğini genel hatlarıyla belirtebiliriz. Bir işlemcinin bir bellek adres kaydı (MAR), bir bellek tampon kaydı (MBR), bir program sayacı (PC) ve bir komut kaydı (IR) kullandığını varsayıyalım.

*Getirme döngüsü* sırasında bellekten bir komut okunur. Şekil 14.6 bu döngü sırasında veri akışını göstermektedir. PC, getirilecek bir sonraki talimatın adresini içerir. Bu adres MAR'a taşınır ve adres veriyoluna yerleştirilir. Kontrol birimi bir bellek okuması talep eder ve sonuç veri yolu yerleştirilir ve MBR'ye kopyalanır ve ardından IR'ye taşınır. Bu sırada PC 1 artırılarak bir sonraki getirme işlemi için hazırlanır.

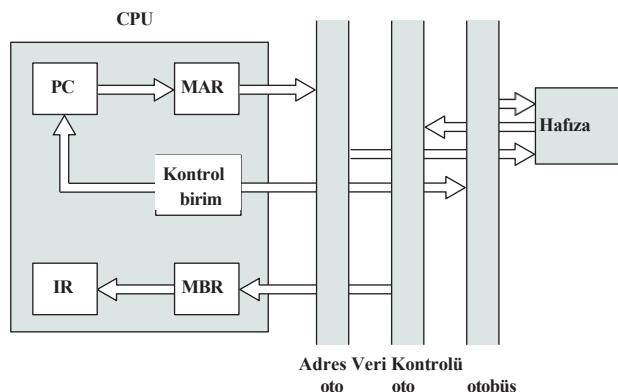
Getirme döngüsü sona erdiğinde, kontrol birimi IR'nin içeriğini inceleyerek dolaylı adresleme kullanan bir işlenen belirtici içerip içermediğini belirler. Eğer öyleyse, bir



Şekil 14.4 Komut Döngüsü



Şekil 14.5 Komut Döngüsü Durum Diyagramı



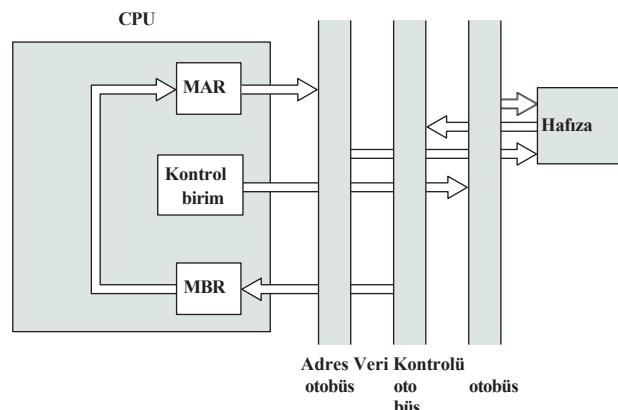
**MBR**= Bellek tampon kaydı  
**MAR**= Bellek adres kaydı **IR**= Komut kaydı  
**PC**= Program sayacı

Şekil 14.6 Veri Akışı, Getirme Döngüsü

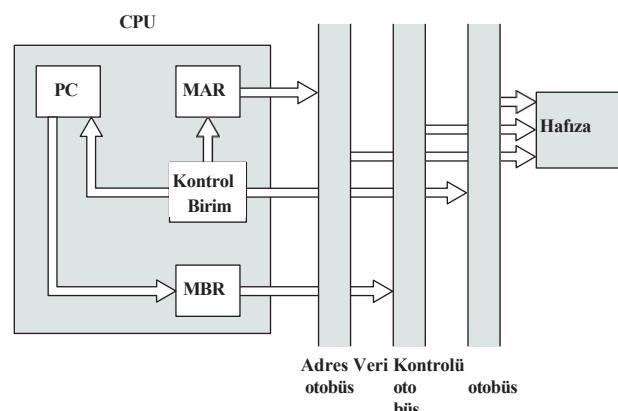
*dolaylı döngü* gerçekleştirilir. Şekil 14.7'de gösterildiği gibi, bu basit bir döngüdür. MBR'nin adres referansını içeren en sağ  $N$  biti MAR'a aktarılır. Daha sonra kontrol birimi, MBR'ye işlenenin istenen adresini almak için bir bellek okuması talep eder.

Getirme ve dolaylı döngüler basit ve tahmin edilebilirdir. *Yürütmeye döngüsü* çeşitli şekillerde gerçekleştir; bu şekil çeşitli makine talimatlarından hangisinin IR'de olduğuna bağlıdır. Bu döngü, kayıtlar arasında veri aktarımını, bellekten veya G/C'den okuma veya yazmayı ve veya ALU'nun çağrılmasını içerebilir.

Getirme ve dolaylı döngüler gibi, *kesme döngüsü* de basit ve tahmin edilebilirdir (Şekil 14.8). İşlemcinin kesmeden sonra normal faaliyetine devam edebilmesi için PC'nin mevcut içeriğinin kaydedilmesi gerekir. Böylece, PC'nin içeriği belleğe yazılmak üzere MBR'ye aktarılır. Bu amaç için ayrılmış özel bellek konumu kontrol ünitesinden MBR'a yüklenir. Örneğin, bir yığın işaretçisi olabilir. PC, kesme rutininin adresi ile yüklenir. Sonuç olarak, bir sonraki komut döngüsü uygun komutun getirilmesiyle başlayacaktır.



Şekil 14.7 Veri Akışı, Dolaylı Döngü



Şekil 14.8 Veri Akışı, Kesme Döngüsü

## 14.4 KOMUT ARDIŞIK DÜZENLEMESİ

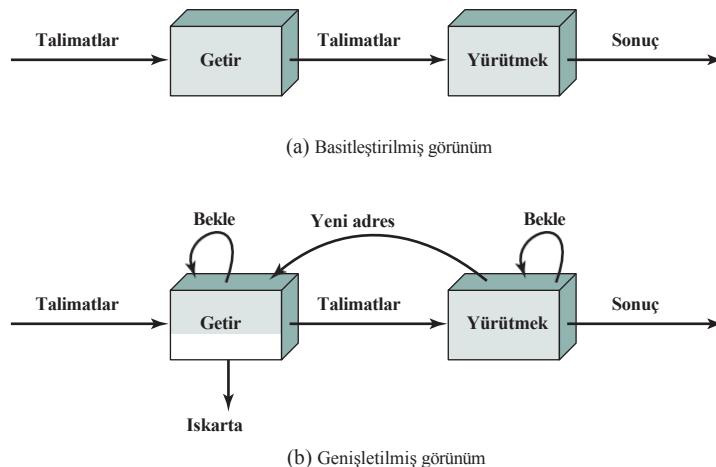
Bilgisayar sistemleri gelişikçe, daha hızlı devre gibi teknolojideki gelişmelerden yararlanarak daha yüksek performans elde edilebilir. Buna ek olarak, işlemcide yapılan organizasyonel geliştirmeler de performansı artırabilir. Tek bir akümülatör yerine birden fazla yazmaç kullanılması ve ön bellek kullanımı gibi bazı örnekleri daha önce görmüştük. Oldukça yaygın olan bir diğer organizasyonel yaklaşım ise komut ardışık dizilimidir.

### Pipelining Stratejisi

Komut ardışık dizilimi, bir üretim tesisinde montaj hattı kullanımına benzer. Bir montaj hattı, bir ürünün üretimin çeşitli aşamalarından geçtiği gerçekinden yararlanır. Üretim sürecinin bir montaj hattına yerleştirilmesiyle, çeşitli aşamalardaki ürünler üzerinde aynı anda çalışılabilir. Bu süreç aynı zamanda *boru* hattı olarak da adlandırılır, çünkü bir boru hattında olduğu gibi, daha önce kabul edilen girdiler diğer ucta çıktı olarak görünmeden önce bir yeni girdiler kabul edilir.

Bu kavramı talimatın yürütülmesine uygulamak için, aslında bir talimatın bir dizi aşaması olduğunu kabul etmeliyiz. Örneğin Şekil 14.5, talimat döngüsünü sırayla gerçekleşen 10 görevle ayırmaktadır. Açıka görüldüğü üzere, ardışık sıralama için bazı fırsatlar olmalıdır.

Basit bir yaklaşım olarak, komut işlemeyi iki aşamaya ayırmayı düşünün: komut alma ve yürütme. Bir komutun yürütülmesi sırasında ana belleğe erişilmediği zamanlar vardır. Bu zaman, mevcut komutun yürütülmesine paralel olarak bir sonraki komutu getirmek için kullanılabilir. Şekil 14.9a bu yaklaşımı göstermektedir. Boru hattının iki bağımsız aşaması vardır. İlk aşama bir komutu alır ve tamponlar. İkinci aşama boş olduğunda, birinci aşama tamponlanmış talimatı ona iletir. İkinci aşama komutu yürütürken, birinci aşama kullanılmayan bellek döngülerinden yararlanarak



**Şekil 14.9** İki Aşamalı Komut İşlem Hattı

ve bir sonraki komutu arabelleğe alır. Buna komut ön getirme ya da *getirme örtüşmesi* denir. Komut tamponlama içeren bu yaklaşımın daha fazla gerektirdiğini unutmayın. Genel olarak, pipelining aşamalar arasında veri depolamak için yazmaçlar gerektir.

Bu işlemin komut yürütmemi hızlandıracığı açık olmalıdır. Eğer getirme ve yürütme aşamaları eşit sürelerde olsaydı, komut çevrim süresi yarıya inerdi. Ancak, bu boru hattına daha yakından bakarsak (Şekil 14.9b), yürütme hızının iki katına çıkmasının iki nedenden dolayı olmadığını görürüz:

- 1. Yürütme süresi genellikle getirme daha uzun olacaktır.** Yürütme, işlenenlerin okunmasını ve saklanması ve bazı işlemlerin gerçekleştirilemesini içerecektir. Bu nedenle, getirme aşamasının arabelleğeni boşaltabilmesi için bir süre beklemesi gerekebilir.
- 2. Koşullu dallanma komutu, getirilecek bir sonraki komutun adresini bilinmez hale getirir.** Bu nedenle, getirme aşaması yürütme bir sonraki komutun adresini alana kadar beklemelidir. Yürütme aşaması daha sonra bir sonraki komut getirilirken beklemek zorunda kalabilir.

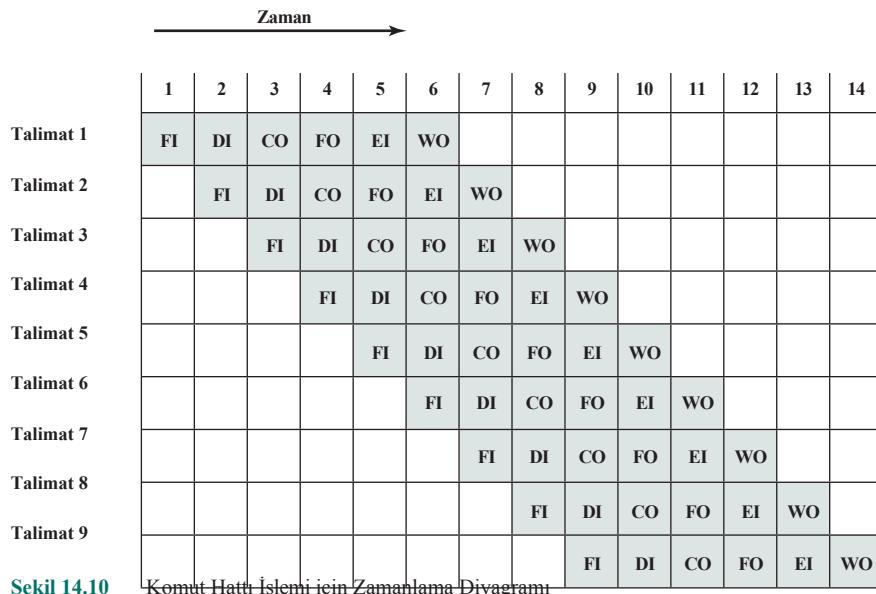
Tahmin etmek ikinci nedenden kaynaklanan zaman kaybını azaltabilir. Basit bir kural aşağıdaki gibidir: Bir koşullu dallanma komutu getirme aşamasından yürütme aşamasına aktarıldığında, getirme aşaması dallanma komutundan sonra bellekteki bir sonraki getirir. Eğer dallanma yapılmazsa zaman kaybedilmez. Eğer dallanma, getirilen komut atılmalı ve yeni bir komut getirilmelidir.

Bu faktörler iki aşamalı boru hattının potansiyel etkinliğini azaltırken, bir miktar hızlanma meydana gelir. Daha fazla hızlanma elde etmek için boru hattının daha fazla aşamaya sahip olması gereklidir. Komut işlemenin aşağıdaki ayırtmasını düşünelim.

- **Getirme talimi (FI):** Bir sonraki beklenen komutu bir arabelleğe okuyun.
- **Kod çözme talimi (DI):** İşlem kodunu ve işlenen belirticilerini belirleyin.
- **İşlenenleri hesaplayın (CO):** Her kaynak operandın etkin adresini hesaplayın. Bu, yer değiştirme, kayıt dolaylı, dolaylı veya diğer adres hesaplama biçimlerini içerebilir.
- **İşlenenleri getir (FO):** Her bir işleneni bellekten getirin. Kaytlardaki işlenenlerin getirilmesine gerek yoktur.
- **Yürütme talimi (EI):** Belirtilen işlemi gerçekleştirir ve varsa sonucu belirtilen hedef işlenen konumuna depolar.
- **İşleneni yaz (WO):** Sonucu bellekte saklar.

Bu ayırtma ile çeşitli aşamalar neredeyse eşit süreye sahip olacaktır. Örnek olması açısından, sürelerin eşit olduğunu varsayıyalım. Bu varsayıımı kullanarak, Şekil 14.10 altı aşamalı bir boru hattının 9 komut için yürütme süresini 54 zaman biriminden 14 zaman birimine düşürebileceğini göstermektedir.

Birkaç yorum yapmak yerinde olacaktır: Diyagramda her komutun boru hattının altı aşamasından da geçtiği varsayılmaktadır. Bu her zaman böyle olmayacağındır. Örneğin, bir yükleme komutu WO aşamasına ihtiyaç duymaz. Ancak, boru hattı donanımını basitleştirmek için, zamanlama her komutun altı da gerektirdiği varsayırlarak ayarlanmıştır. Ayrıca, diyagramda tüm aşamaların aynı anda gerçekleştirilebileceği varsayılmıştır. Özellikle, bellek çakışması olmadığı varsayılmıştır. Örneğin, FI, FO ve WO aşamaları bir bellek erişimi içerir. Diyagram, tüm bu erişimlerin aynı anda gerçekleşebileceğini ima eder. Çoğu bellek sistemi buna izin vermeyecektir.



Ancaq, istenen değer önbellekte olabilir veya FO ya da WO aşaması boş olabilir. Bu nedenle, çoğu zaman bellek çalışmaları boru hattını yavaşlatmaz.

Diğer bazı faktörler de performans artışı sınırlamaktadır. Eğer altı aşama eşit sürelerde değilse, daha önce iki aşamalı boru hattı için tartışıldığı gibi, çeşitli boru hattı aşamalarında bir miktar bekleme olacaktır. Bir başka zorluk, birkaç komut getirme işlemini geçersiz kılabilen koşullu dallanma komutudur. Benzer bir öngörülemeyen olay da bir kesmedir. Şekil 14.11, Şekil 14.10 ile aynı programı kullanarak koşullu dallanmanın etkilerini göstermektedir. Talimat 3'ün talimat 15'e koşullu dallanma olduğunu varsayıyalım. Komut yürütülene kadar, hangi komutun sonra geleceğini bilmenin hiçbir yolu yoktur. Boru hattı, bu örnekte, sadece sıradaki komutu (komut 4) yükler ve ilerler. Şekil 14.10'da dallanma yapılmaz ve geliştirmenin tüm performans avantajını elde ederiz. Şekil 14.11'de dallanma. Bu, zaman birimi 7'nin sonuna kadar belirlenmez. Bu noktada, boru hattı işe yaramayan talimatlardan temizlenmelidir. Zaman birimi 8 sırasında, komut 15 boru hattına girer. 9'dan 12'ye kadar olan zaman birimlerinde hiçbir komut tamamlanmamıştır; bu, dallanmayı öngöremedigimiz için ortaya çıkan performans cezasıdır. Şekil 14.12, dallanma ve kesmeleri hesaba katmak için boru hattı için gereken mantıq göstermektedir.

Basit iki aşamalı organizasyonumuzda ortaya çıkmayan başka sorunlar da ortaya çıkmaktadır. CO aşaması, hala boru hattında olan önceki bir komut tarafından değiştirilebilecek bir yazmacın içeriğine bağlı olabilir. Bu türden başka yazmaç ve bellek çalışmaları da meydana gelebilir. Sistem bu tür çalışmaları hesaba katacak bir mantık içermelidir.

Boru hattı işleyişini açılığa kavuşturmak için alternatif bir tasvire bakmak faydalı olabilir. Şekil 14.10 ve 14.11'de zamanın yatay olarak şekil boyunca ilerleyisi gösterilmektedir ve her bir satır tek bir talimin ilerleyişini göstermektedir. Şekil 14.13 aynı olay dizisini, zaman dikey olarak aşağıya doğru ilerlerken göstermektedir

	Zaman → ← Şube cezası													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Talimat 1	FI	DI	CO	FO	EI	WO								
Talimat 2		FI	DI	CO	FO	EI	WO							
Talimat 3			FI	DI	CO	FO	EI	WO						
Talimat 4				FI	DI	CO	FO							
Talimat 5					FI	DI	CO							
Talimat 6						FI	DI							
Talimat 7							FI							
Talimat 15								FI	DI	CO	FO	EI	WO	
Talimat 16									FI	DI	CO	FO	EI	WO

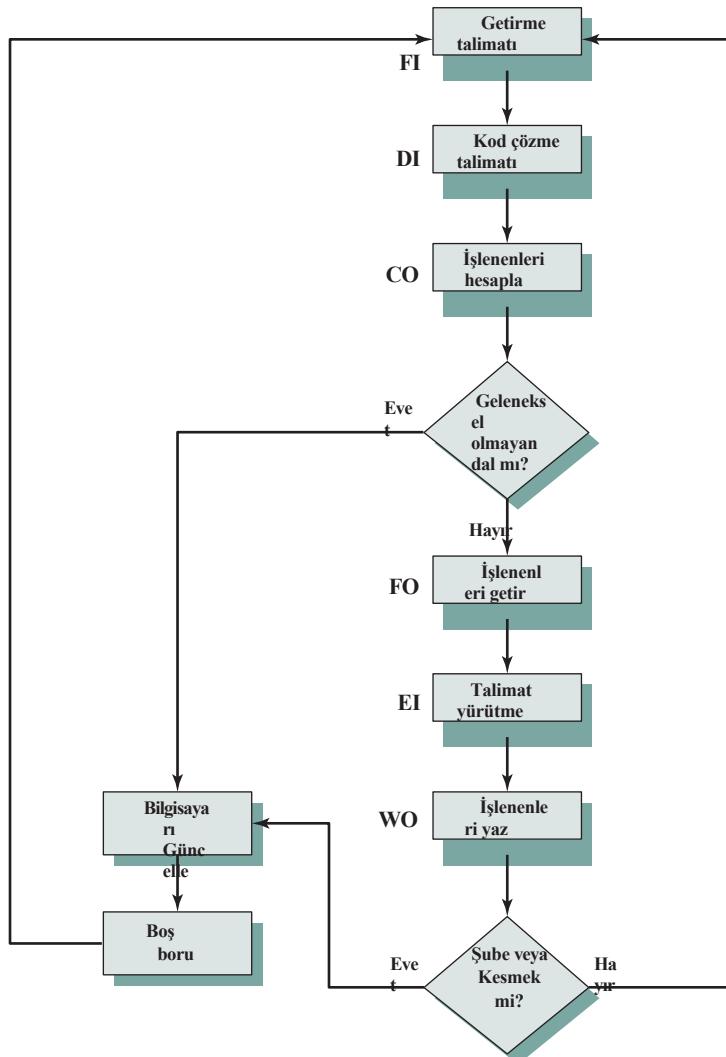
**Şekil 14.11** Koşullu Dallanmanın Komut İş Hattı İşlemi Üzerindeki Etkisi

Şekilde, her satır boru hattının belirli bir zamandaki durumunu göstermektedir. Şekil 14.13a'da (Şekil 14.10'a karşılık gelir), boru hattı 6. zamanda doludur, 6 farklı komut yürütmenin çeşitli aşamalarındadır ve 9. zamana kadar dolu kalır; 19 komutunun yürütülecek son komut olduğunu varsayıyoruz. Şekil 14.13b'de (Şekil 14.11'e karşılık gelmektedir), boru hattı 6. ve 7. zamanlarda doludur. Zaman 7'de, komut 3 yürütme aşamasındadır ve komuta bir dallanma gerçekleşir 15. Bu noktada, I4'ten I7'ye kadar olan komutlar boru hattından, böylece 8. zamanda boru hattında yalnızca iki komut kalır: I3 ve I15.

Önceki tartışmadan, boru hattındaki aşama sayısı arttıkça yürütme hızının da artacağı anlaşılabılır. IBM S/360 tasarımcılarından bazıları, yüksek performanslı tasarım için bu basit görünen modeli engelleyen iki faktöre dikkat çekmiştir [ANDE67a] ve bu faktörler tasarımcının hala dikkate alması gereken unsurlar olmaya devam etmektedir:

1. Boru hattının her aşamasında, verilerin tampondan tampona taşınması ve çeşitli hazırlık ve dağıtım işlevlerinin yerine getirilmesiyle ilgili bir miktar ek yük vardır. Bu ek yük, tek bir komutun toplam yürütme süresini kayda değer ölçüde uzatabilir. Bu durum, sıralı komutların yoğun dallanma kullanımını ya da bellek erişim bağımlılıkları yoluyla log-tik olarak bağımlı olduğu durumlarda önemlidir.
2. Bellek ve kayıt bağımlılıklarını ele almak ve boru hattının kullanımını optimize etmek için gereken kontrol mantığı miktarı, aşama sayısıyla birlikte muazzam ölçüde. Bu durum, aşamalar arasındaki geçiş kontrollen mantığın kontrol edilen aşamalardan daha karmaşık olduğu bir duruma yol açabilir.

Bir diğer husus da mandallama gecikmesidir: Boru hattı tamponlarının çalışması zaman alır ve bu da komut çevrim süresine eklenir.



Şekil 14.12 Altı Aşamalı CPU Komut İşlem Hattı

Komut ardışık sıralama, performansı artırmak için güçlü bir tekniktir ancak makul karmaşıklıkta optimum sonuçlar elde etmek için dikkatli tasarım gerektir.

### Boru Hattı Performansı

Bu alt bölümde, işlem hattı performansı ve göreceli hızlanma için bazı basit ölçütler geliştiriyoruz ([HWAN93]'teki bir tartışmaya dayanarak). Bir **komut ardışık düzeninin** döngü süresi, bir dizi komutu ardışık boyunca bir aşama ilerletmek için gereken süredir; Şekil 14.10 ve 14.11'deki her sütun bir döngü süresini temsil eder. Çevrim süresi şu şekilde belirlenebilir

$$t = \max_i[t_i] + d = t_m + d_1 \dots i \dots k$$

	FI	DI	CO	FO	EI	WO
1	II					
2	I2	I1				
3	I3	I2	II			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I8	I7	I6	I5	I4	I3
9	I9	I8	I7	I6	I5	I4
10		I9	I8	I7	I6	I5
11			I9	I8	I7	I6
12				I9	I8	I7
13					I9	I8
14						(a) Sube yok

	FI	DI	CO	FO	EI	WO
1	II					
2	I2	I1				
3	I3	I2	II			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I15					I3
9	I16	I15				
10		I16	I15			
11			I16	I15		
12				I16	I15	
13					I16	I15
14						I16
	(b) Koşullu dallanma ile					I16

Şekil 14.13 Alternatif Bir Boru Hattı Tasviri

nerede

$t_i$ = boru hattının birinci aşamasındaki devrenin zaman gecikmesi

$t_m$ = maksimum kademe gecikmesi (en büyük gecikmeyi yaşayan kademeden geçen gecikme)

$k$ = komut ardisık düzeneindeki aşama sayısı

$d$ = sinyalleri ve verileri bir aşamadan diğerine ilerletmek için gereken bir mandalın zaman gecikmesi

Genel olarak,  $d$  zaman gecikmesi bir saat darbesine eşdeğerdir ve  $t_m \leq d$ . Şimdi dallanma olmaksızın  $n$  komutun işlendiğini varsayalım.  $T_{k,n}$ ,  $k$  aşamalı bir boru hattının  $n$  komutu yürütmesi için gereken toplam süre olsun. O zaman

$$T_{k,n} = [k + (n - 1)]t \quad (14.1)$$

İlk talimatın yürütülmesini tamamlamak için toplam  $k$  döngü gereklidir ve kalan  $n - 1$  talimat  $n - 1$  döngü gerektirir.<sup>(2)</sup> Bu denklem Şekil 14.10'dan kolayca doğrulanabilir. Dokuzuncu komut 14. zaman döngüsünde tamamlanır:

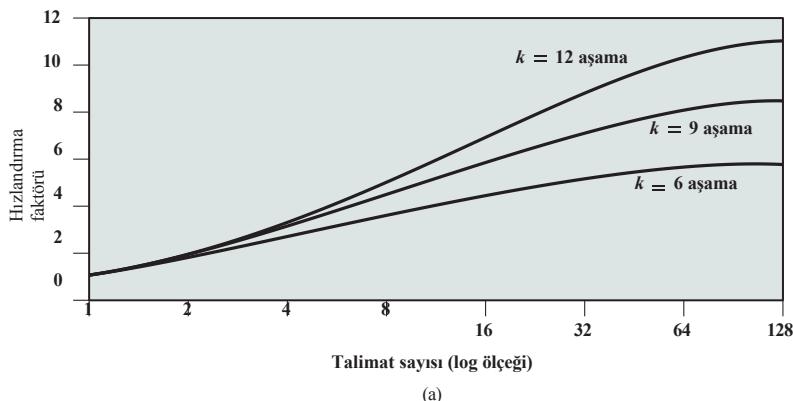
$$14 = [6 + (9 - 1)]$$

<sup>2</sup> Burada biraz özensiz davranışlıyız. Çevrim süresi sadece tüm aşamalar dolu olduğunda 'un maksimum değerine eşit olacaktır. Başlangıçta, döngü süresi ilk bir veya birkaç döngü için daha az olabilir.

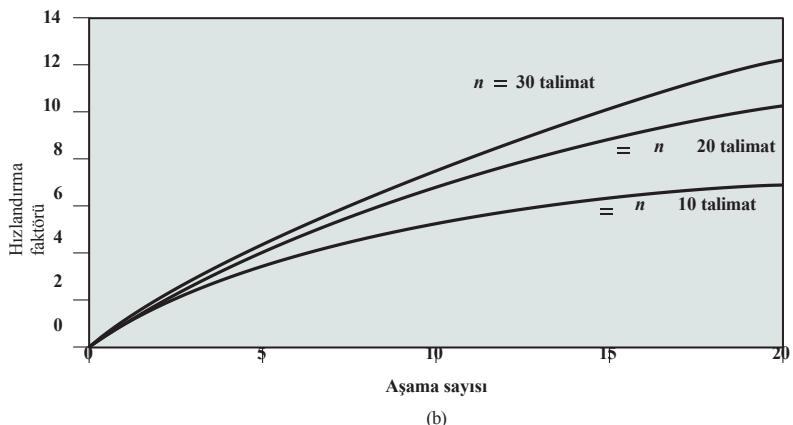
Şimdi eşdeğer işlevlere sahip ancak işlem hattı olmayan bir işlemci düşünün ve komut çevrim süresinin  $kt$  olduğunu varsayıñ. İş hattı olmadan yürütmeye kıyasla komut iş hattı için hızlandırma faktörü şu şekilde tanımlanır

$$\frac{S}{k} = \frac{\frac{Tl}{n}}{\frac{Tk}{n} + (n-1)t} = \frac{nkt}{[k+(n-1)]t} = \frac{nk}{k+(n-1)} \quad (14.2)$$

Şekil 14.14a, dallanma olmadan yürütülen talimat sayısının bir fonksiyonu olarak hızlandırma faktörünü göstermektedir. Beklenebileceği gibi, sınırlı ( $n \leq \infty$ ),  $k$  kat hızlandırma elde ederiz. Şekil 14.14b, komut boru hattındaki aşama sayısının bir fonksiyonu olarak hızlandırma faktörünü göstermektedir.<sup>3</sup> Bu durumda, hızlandırma faktörü dallanma olmadan boru hattına beslenebilen komut sayısına yaklaşmaktadır. Dolayısıyla, boru hattı aşamalarının sayısı arttıkça hızlanma de artar. Bununla birlikte, pratik bir konu olarak, ek komutların potansiyel kazançları



(a)



(b)

**Şekil 14.14** Komut Ardışık Sıralama ile Hızlandırma Faktörleri

<sup>(3)</sup> Şekil 14.14a'da x ekseninin logaritmik, Şekil .14b'de ise doğrusal olduğuna dikkat ediniz.

boru hattı aşamaları, maliyet artıları, aşamalar arasındaki gecikmeler ve boru hattının yıkamasını gerektiren dallarla karşılaşacağı gerçeği ile karşılaşmaktadır.

## Boru Hattı Tehlikeleri

Bir önceki alt bölümde, boru hattı performansının optimumdan daha düşük olmasına neden olabilecek bazı durumlardan bahsetti. Bu alt bölümde, bu konuya daha sistematik bir şekilde inceleyeceğiz. Bölüm 16, süperskalar boru hattı organizasyonlarında bulunan karmaşıklıkları tanıttıktan sonra bu daha ayrıntılı olarak tekrar ele.

**Boru hattı tehlikesi**, boru hattının veya boru hattının bir kısmının, koşullar devam etmesine izin vermediği için durması gerekiğinde meydana gelir. Böyle bir boru hattı durması *boru hattı balonu* olarak da adlandırılır. Üç tür tehlike vardır: kaynak, veri ve kontrol.

**KAYNAK TEHLİKELERİ** Kaynak tehlikesi, halihazırda işlem hattında bulunan iki (veya daha fazla) talimatın aynı kaynağa ihtiyaç duyması durumunda ortaya çıkar. Sonuç olarak, talimatların boru hattının bir kısmı için paralel yerine seri olarak yürütülmesi gereklidir. Kaynak tehlikesi bazen *yapısal* tehlike olarak da adlandırılır.

Kaynak tehlikesinin basit bir örneğini ele alalım. Her aşamanın bir saat döngüsü sürdüğü basitleştirilmiş beş aşamalı bir boru hattı varsayılmış. Şekil 14.15a, her saat döngüsünde yeni bir komutun boru hattına girdiği ideal durumu göstermektedir. Şimdi ana belleğin tek bir bağlantı noktası olduğunu ve tüm komut getirme ve veri okuma ve yazma işlemlerinin tek seferde yapılması gerektiğini varsayıy়. Ayrıca, önbelleği göz ardı edin. Bu durumda, bellekten okunan veya bellekten yazılıan bir işlenen paralel olarak gerçekleştirilemez

Saat döngüsü									
	1	2	3	4	5	6	7	8	9
Talimatlar	I1	FI	DI	FO	EI	WO			
	I2		FI	DI	FO	EI	WO		
	I3			FI	DI	FO	EI	WO	
	I4				FI	DI	FO	EI	WO

(a) Beş aşamalı boru hattı, iucar durumı

Saat döngüsü									
	1	2	3	4	5	6	7	8	9
Talimatlar	I1	FI	DI	FO	EI	WO			
	I2		FI	DI	FO	EI	WO		
	I3			Boşta	FI	DI	FO	EI	WO
	I4				FI	DI	FO	EI	WO

(b) Bellekteki I1 kaynak işleneni

**Şekil 14.15** Kaynak Tehlikesi Örneği

bir komut getirme ile. Bu durum, I1 komutu için kaynak işlenenin bir yazmaç yerine bellekte olduğunu varsayan Şekil 14.15b'de gösterilmiştir. Bu nedenle, boru hattının komut getirme aşaması, I3 komutu için komut getirmeye başlamadan önce bir döngü boyunca boşta kalmalıdır. Şekilde diğer tüm işlenenlerin yazmaçlarda olduğu varsayılmaktadır.

Kaynak çatışmasının bir başka örneği de birden fazla komutun yürütme komutu aşamasına girmeye hazır olduğunu ve tek bir ALU'nun bulunduğu durumdur. Bu tür kaynak tehlikelerine bir çözüm, ana belleğe birden fazla bağlantı noktası ve birden fazla ALU birimine sahip olmak gibi mevcut kaynakları artırmaktır.



#### Rezervasyon Tablosu Analizörü

Kaynak çatışmalarını analiz etmeye ve boru hatlarının tasarımasına yardımcı olmaya yönelik yaklaşımlardan biri rezervasyon tablosudur. Rezervasyon tablolarını Ek N'de inceleyeceğiz.

**VERİ TEHLİKELERİ** Bir veri tehlikesi, bir operand konumuna erişimde bir çakışma olduğunda ortaya çıkar. Genel anlamda tehlikeyi şu şekilde ifade edebiliriz: Bir programdaki iki komut sırayla yürütülecektir ve her ikisi de belirli bir bellek veya yazmaç operandına erişecektir. Eğer bu iki komut tam bir sırayla yürütülsürse, herhangi bir sorun oluşmaz. Ancak, talimatlar bir ardışık düzende yürütülsürse, işlenen değerinin katı sıralı yürütme ile ortaya çıkandan farklı bir sonucu üretecek şekilde güncellenmesi mümkündür. Başka bir deyişle, program ardışık sıralama kullanımı nedeniyle yanlış bir sonuç üretir.

Örnek olarak, aşağıdaki x86 makine komut dizisini düşünün:

```
ADD EAX, EBX /* EAX= EAX+ EBX SUB
ECX, EAX /* ECX= ECX - EAX
```

İlk komut 32 bitlik EAX ve EBX kayıtlarının içeriğini toplar ve sonucu EAX içinde saklar. İkinci komut EAX'in içeriğini ECX'ten çıkarır ve sonucu te saklar. Şekil 14.16 boru hattı davranışını göstermektedir.

Saat döngüsü										
	1	2	3	4	5	6	7	8	9	10
ADD EAX, EBX SUB ECX, EAX	FI	DI	FO	EI	WO					
		FI	DI	Boşta		FO	EI	WO		
			FI			DI	FO	EI	WO	
						FI	DI	FO	EI	WO

**Şekil 14.16** Veri Tehlikesi Örneği

ADD komutu, saat döngüsü 5'te gerçekleşen aşama 5'in sonuna kadar EAX kaydını güncellemez. Ancak SUB komutu, saat döngüsü 4'te gerçekleşen aşama 2'nin başında bu değere ihtiyaç duyar. Doğru çalışmayı sürdürmek için boru hattı iki saat çevrimi boyunca durmalıdır. Dolayısıyla, özel donanım ve özel önleme algoritmalarının yokluğunda, bu tür bir veri tehlikesi boru hattının verimsiz kullanılmasına neden olur.

Üç tür veri tehlikesi vardır:

- **Yazdıktan sonra okuma (RAW) veya gerçek bağımlılık:** Bir komut bir kayıt veya bellek konumunu değiştirir ve ardından gelen bir komut bu bellek veya kayıt konumundaki verileri okur. Okuma işlemi yazma işlemi tamamlanmadan önce gerçekleşirse bir tehlike oluşur.
- **Okuma sonrası yazma (WAR) veya bağımlılık karşılığı:** Bir komut bir kayıt veya bellek konumunu okur ve ardından gelen bir komut bu konuma yazar. Okuma işlemi gerçekleşmeden önce yazma işlemi tamamlandırsa bir tehlike oluşur.
- **Yazma sonrası yazma (WAW) veya çıktı bağımlılığı:** İki talimatın her ikisi de aynı konuma yazar. Yazma işlemleri amaçlanan sıranın tersine gerçekleşirse bir tehlike oluşur.

Şekil 14.16'daki örnek bir RAW tehlikesidir. Diğer iki tehlike en iyi Bölüm 16'da ele alınan süper skaler organizasyon bağlamında tartışılabilir.

**KONTROL TEHLİKELERİ** Dallanma tehlikesi olarak da bilinen kontrol tehlikesi, boru hattı bir dallanma tahmininde yanlış karar verdiğide ve bu nedenle daha sonra atılması gereken komutları boru hattına getirdiğinde ortaya çıkar. Kontrol tehlikeleri ile başa çıkma yaklaşımlarını daha sonra tartışacağız.

## Şubeler ile Çalışmak

Bir komut ardışık düzeninin tasarlanmasındaki en önemli sorunlardan biri, ardışık düzenin ilk aşamalarına düzeltilmesi ve bu nedenle daha sonra atılması gereken komutları boru hattına getirdiğinde ortaya çıkar. Kontrol tehlikeleri ile başa çıkma yaklaşımlarını daha sonra tartışacağımız.

Koşullu dallarla başa çıkmak için çeşitli yaklaşımlar benimsenmiştir:

- Çoklu akışlar
- Şube hedefini önceden getir
- Döngü tamponu
- Şube tahmini
- Gecikmeli şube

**COKLU AKIŞLAR** Basit bir ardışık düzen, bir dallanma komutu için ceza çeker çünkü bir sonraki getirilecek iki komuttan birini seçmesi gereklidir ve yanlış seçim yapabilir. Kaba kuvvet yaklaşımı, boru hattının ilk kısımlarını çoğaltmak ve boru hattının iki akıştan yararlanarak her iki talimatı da getirmesine izin vermektedir. Bu yaklaşımla ilgili iki sorun vardır:

- Çoklu boru hatlarında, kayıtlara ve belleğe erişim için çekisme gecikmeleri vardır.

- Orijinal dallanma kararını çözülmeden önce ek dallanma talimatları boru hattına (her iki akış) girebilir. Bu tür her talimatın bir ek akışa ihtiyacı vardır.

Bu dezavantajlara rağmen, bu strateji performansı artırabilir. İki veya daha fazla boru hattı akışına sahip makinelere örnek olarak IBM 370/168 ve IBM 3033 verilebilir.

**BRANS HEDEFİNİ** ÖNCEDEN BELİRLE Koşullu bir dallanma tanındığında, dallanmayı izleyen komuta ek olarak hedefi önceden belirlenir. Bu hedef daha sonra dallanma komutu yürütülene kadar saklanır. Dallanma, hedef zaten öncelenmiştir.

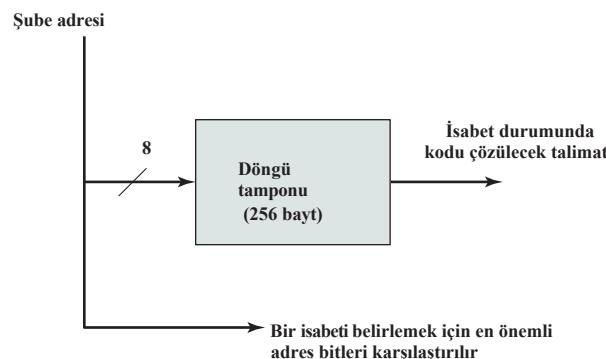
IBM 360/91 bu yaklaşımı kullanır.

DÖNGÜ TAMPONU Döngü *tamponu*, boru hattının komut getirme aşaması tarafından tutulan ve en son getirilen  $n$  komutu sırayla içeren küçük, çok yüksek hızlı bir bellektir. Bir dallanma yapılacaksa, donanım önce dallanma hedefinin tampon içinde olup olmadığını kontrol eder. Eğer öyleyse, bir sonraki komut tampondan getirilir. Döngü tamponunun üç faydası vardır:

1. Prefetching kullanımı ile döngü tamponu, mevcut komut getirme adresinden önce sırayla bazı komutlar içerecektir. Böylece, sırayla getirilen talimatlar normal bellek erişim süresi olmadan kullanılabilir olacaktır.
2. Eğer bir dallanma, dallanma komutunun adresinden sadece birkaç konum ilerideki bir hedefe gerçekleşirse, hedef zaten tamponda olacaktır. Bu, oldukça yaygın olan IF-THEN ve IF-THEN-ELSE dizileri için kullanılышılır.
3. Bu strateji özellikle döngülerle veya yinelemelerle başa çıkmak için çok uygundur; bu nedenle döngü *tamponu* olarak adlandırılır. Döngü tamponu bir döngüdeki tüm talimatları içerecek kadar büyükse, bu talimatların ilk yineleme için bellekten yalnızca bir kez alınması gereklidir. Sonraki yinelemeler için gerekli tüm talimatlar zaten tamponda bulunur.

Döngü tamponu prensip olarak talimatlara ayrılmış bir önbelleğe benzer. Aradaki farklar, döngü tamponunun yalnızca talimatları sırayla tutması ve boyutunun çok daha küçük ve dolayısıyla maliyetinin daha düşük olmasıdır.

Şekil 14.17'de bir döngü tamponu örneği verilmiştir. Tampon 256 bayt içeriyorsa ve bayt adresleme kullanılıyorsa, en az anlamlı 8 bit



Şekil 14.17 Döngü Tamponu

tampon. Kalan en önemli bitler, dallanma hedefinin tampon tarafından yakalanan ortam içinde olup olmadığını belirlemek için kontrol edilir.

Döngü tamponu kullanan makineler arasında bazı CDC makineleri (Star-100, 6600, 7600) ve CRAY-1 bulunmaktadır. Motorola 68010'da DBcc (koşulda azaltma ve dallanma) komutunu içeren üç komutlu bir döngüyü yürütmek için özel bir döngü tamponu biçimini mevcuttur (bkz. Problem 14.14). Üç kelimelek bir tampon tutulur ve işlemci döngü koşulu sağlanana kadar bu komutları tekrar yürütür.



#### Dal Tahmin Simülatörü Dal Hedef Tamponu

**DAL TAHMİNİ** Bir dalın alınıp alınmayacağı tahmin etmek için çeşitli teknikler kullanılabilir. En yaygın olanları arasında aşağıdakiler yer almaktadır:

- Tahmin hiç alınmadı
- Her zaman alınan tahmin
- İşlem koduna göre tahmin
- Alınan/alınmayan anahtar
- Şube geçmişi tablosu

İlk üç yaklaşım statiktir: koşullu dallanma komutu zamanına kadar olan yürütme geçmişine bağlı değildirler. Son iki yaklaşım dinamiktir: Yürütme geçmişine bağlıdır.

İlk iki yaklaşım en basit olanlardır. Bunlar ya her zaman dallanmanın alınmayacağıını varsayar ve talimatları sırayla getirmeye devam eder ya da her zaman dallanmanın alınacağını varsayar ve her zaman dallanma tar- get'inden getirir. Hiç alınmayacak tahmini, tüm dallanma yöntemleri arasında en popüler olanıdır.

Program davranışını analiz eden çalışmalar, koşullu dallanmaların zamanın %50'sinden fazlasında yapıldığını göstermiştir [LILJ88] ve bu nedenle her iki yoldan da ön-getirmenin maliyeti aynıysa, dallanma hedef adresinden her zaman ön-getirme yapmak, sıralı yoldan her zaman ön-getirme yapmaktan daha iyi performans vermelidir. Ancak, sayfali bir makinede, dallanma hedefini önceden getirmenin bir sayfa hatasına neden olma olasılığı, sıradaki bir sonraki komutu önceden getirmeye göre daha yüksektir ve bu nedenle bu performans cezası dikkate alınmalıdır. Bu cezayı azaltmak için bir kaçınma mekanizması kullanılabilir.

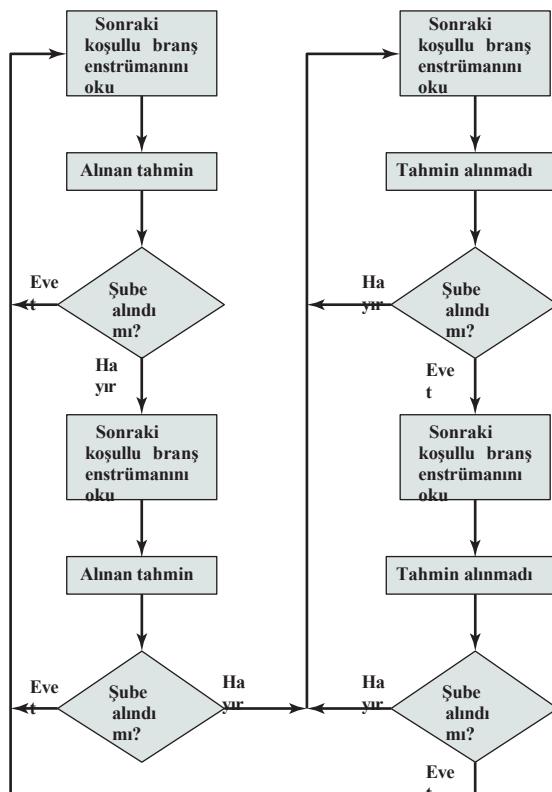
Son statik yaklaşım, dallanma komutunun işlem koduna göre karar verir. İşlemci, dallanmanın belirli dallanma işlem kodları için alınacağını ve diğerleri için alınmayacağıını varsayar. [LILJ88] bu strateji ile %75'in üzerinde başarı oranları bildirmektedir.

Dinamik dallanma stratejileri, bir programdaki koşullu dallanma talimatlarının geçmişini kaydederek tahminin doğruluğunu artırmaya çalışır. Örneğin, her bir koşullu dallanma talimiği ile bir veya daha fazla bit ilişkilendirilebilir.

komutun yakın geçmişini yansıtır. Bu bitler, komutla bir sonraki karşılaşmada işlemciyi belirli bir karar vermeye yönlendiren alındı/alınmadı anahtarı olarak adlandırılır. Tipik olarak, bu geçmiş bitleri ana bellekteki komutla ilişkilendirilmez. Bunun yerine, geçici yüksek hızlı depolama alanında tutulurlar. Bir olasılık, bu bitleri bir önbellekte bulunan herhangi bir koşullu dallanma komutuya ilişkilendirmektir. Komut önbellekte değiştirildiğinde, geçmiş biti kaybolur. Diğer bir olasılık ise her bir girdisinde bir ya da daha fazla geçmiş biti bulunan son çalıştırılan dallanma talimatları için küçük bir tablo tutmaktadır. İşlemci bu tabloya bir önbellek gibi ilişkisel olarak ya da dallanma komutunun adresinin düşük sıralı bitlerini kullanarak erişebilir.

Tek bir bit ile kaydedilebilecek tek şey, bu komutun son çalıştırılmasının bir dallanma ile sonuçlanıp sonuçlanmadığıdır. Tek bir bit kullanmanın bir eksikliği, döngü komutu gibi neredeyse her zaman alınan koşullu bir dallanma komutu durumunda ortaya çıkar. Sadece bir bitlik geçmişle, tahmin hatası döngünün her kullanımında iki kez meydana gelecektir: bir kez döngüye girerken, bir kez de çıkışken.

İki bit kullanılırsa, bunlar ilgili komutun yürütülmesinin son iki örneğinin sonucunu kaydetmek ya da başka bir şekilde bir durumu kaydetmek için kullanılabilir. Şekil 14.18 tipik bir yaklaşımı göstermektedir (diğer yaklaşımlar için Problem 14.13'e bakınız).



Şekil 14.18 Dal Tahmini Akış Şeması

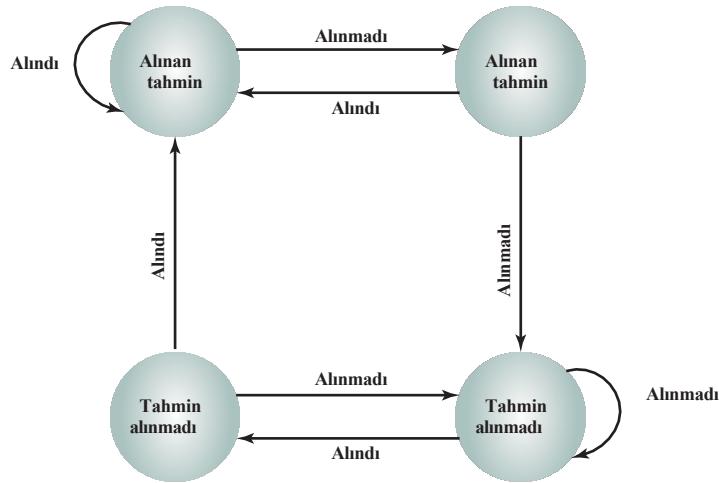
olasılıklar). Algoritmanın akış şemasının sol üst köşesinden başladığını varsayıyalım. Karşılaşılan her bir koşullu dallanma talimatı alındığı sürece, karar süreci bir sonraki dallanmanın tahmin eder. Tek bir tahmin yanlışsa, algoritma sonraki dalın alımılığını tahmin etmeye devam eder. Yalnızca iki ardışık dal almazsa algoritma akış şemasının sağ tarafına kayar. Daha sonra algoritma, arka arkaya iki dal alınana kadar dalların alınmadığını tahmin edecektir. Böylece, algoritma tahmin kararını değiştirmek için iki ardışık yanlış tahmin gerektirir.

Karar süreci, Şekil 14.19'da gösterilen sonlu durum makinesi ile daha kompakt bir şekilde temsil edilebilir. Sonlu durum makinesi gösterimi literatürde yaygın olarak kullanılmaktadır.

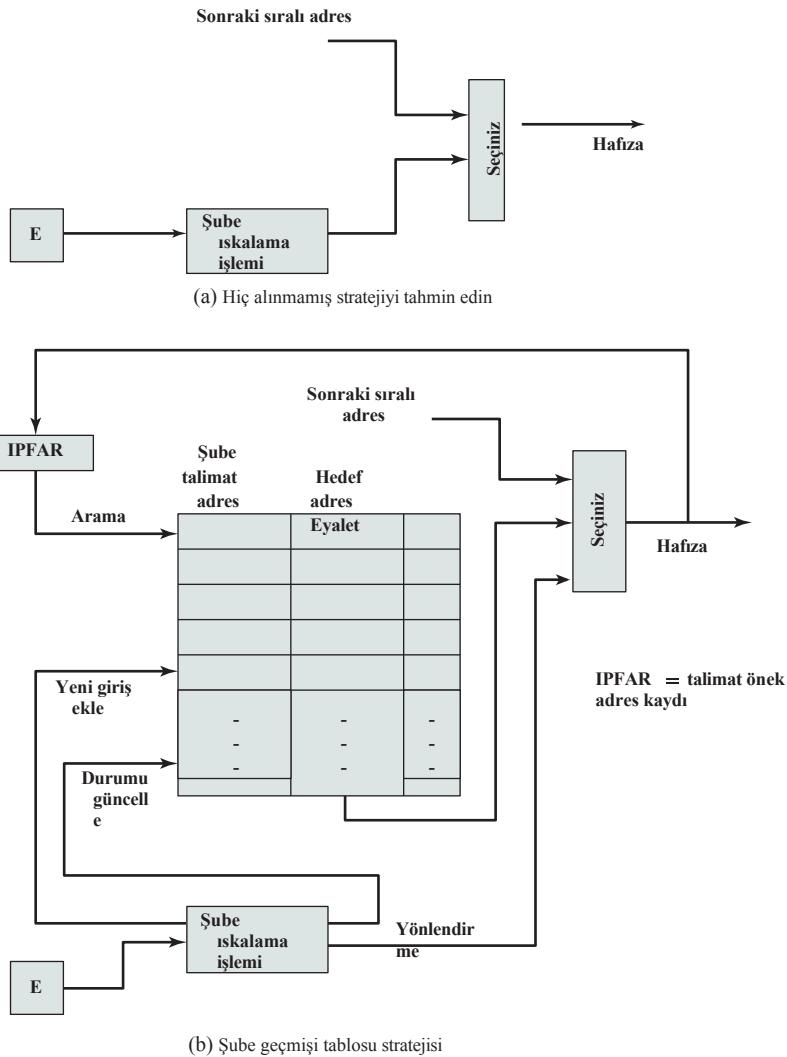
Az önce açıklandığı gibi geçmiş bitlerinin kullanımının bir dezavantajı vardır: Dallanma kararı verilirse, koşullu dallanma komutunda bir operand olan tar- get adresinin kodu çözüleme kadar hedef komut getirilemez. Dallanma kararı verilmez komut getirme işlemi başlatılabilirse daha yüksek verimlilik elde edilebilir. Bu amaçla, dallanma hedefi tamponu ya da dallanma geçmişi tablosu olarak bilinen daha fazla bilgi kaydedilmelidir.

Dallanma geçmişi tablosu, boro hattının talimat getirme aşamasıyla ilişkili küçük bir ön bellektir. Tablodaki her girdi üç unsurdan oluşur: dallanma komutunun adresi, bu komutun kullanım durumunu kaydeden bir miktar geçmiş biti ve hedef komut hakkında bilgi. Çoğu öneri ve uygulamada, bu üçüncü alan hedef komutun adresini içerir. Bir başka olasılık da üçüncü alanın gerçekten hedef komutu içermesidir. Aradaki fark açıkta: Hedef adresin saklanması, hedef komutun saklanması kiyasla daha küçük bir tablo ancak daha büyük bir komut getirme süresi sağlar [RECH98].

Şekil 14.20, bu şemayı tahmin-etme stratejisi ile karşılaştırmaktadır. İlk stratejide, komut getirme aşaması her zaman bir sonraki sıralı komutu getirir.



**Şekil 14.19** Dal Tahmini Durum Diyagramı



Şekil 14.20 Şubelerle İlgilenme

adresi. Bir dallanma yapılrsa, işlemcideki bazı mantık bunu algılar ve bir sonraki komutun hedef adresinden alınmasını ister (boru hattının temizlenmesine ek olarak). Dallanma geçmişi tablosu bir önbellek olarak değerlendirilir. Her ön-getirme dallanma geçmişi tablosunda bir aramayı tetikler. Eşleşme bulunamazsa, getirme işlemi için bir sonraki sıralı adres kullanılır. Bir eşleşme bulunursa, komutun durumuna göre bir tahmin yapılır: Ya bir sonraki sıralı adres ya da dallanma hedef adresi seçme mantığına beslenir.

Dallanma komutu yürütüldüğünde, yürütme aşaması geçmişi tablosu mantığına sonucu bildirir. Komutun durumu doğru veya yanlış tahmini yansıtacak şekilde güncellenir. Tahmin yanlışsa, seçme mantığı

bir sonraki getirme işlemi için doğru adrese yönlendirilir. Tabloda olmayan bir koşullu dallanma komutuyla karşılaşıldığında, bu komut tabloya eklenir ve Bölüm 4'te tartışılan önbellek değiştirme algoritmalarından biri kullanılarak mevcut girdilerden biri atılır.

Dallanma geçmişini yaklaşımlının bir iyileştirmesi iki seviyeli veya korelasyon tabanlı dallanma geçmişini olarak adlandırılır [YEH91]. Bu yaklaşım, döngü kapatan dallarda, belirli bir dal komutunun geçmişinin gelecekteki davranışının iyi bir tahminci olduğu varsayıma dayanırken, daha karmaşık kontrol akışı yapılarında, bir dalın yönünün genellikle ilgili yön ile ilişkili olduğu varsayıma dayanır. Buna örnek olarak if-then-else veya case yapısı verilebilir. Mümkün olan çok sayıda strateji vardır. Tipik olarak, mevcut dallanma talimatının geçmişine ek olarak son global dallanma geçmişini (yani, sadece bu dallanma talimatının değil, en son dallanmaların geçmişini) kullanılır. Genel yapı, mevcut dallanma talimi için  $2^m n\text{-bit}$  dallanma tahmincileri arasından seçim yapmak üzere son  $m$  dallanmanın davranışını kullanan bir ( $m, n$ ) korelatör olarak tanımlanır. Başka bir deyişle, en son  $m$  dal tarafından alınan dalların her olası kombinasyonu için verilen bir dal için  $n$  bitlik bir geçmiş tutulur.

**GECİKMELİ BRANCH** Bir program içindeki talimatları otomatik olarak yeniden düzenleyerek boru hattı performansını artırmak mümkündür, böylece dallanma talimatları gerçekle istenenden daha geç gerçekleşir. Bu ilgi çekici yaklaşım Bölüm 15'te incelenmiştir.

### Intel 80486 Pipelining

Talimat ardışık düzeninin öğretici bir örneği Intel 80486'dır. 80486 beş aşamalı bir ardışık düzen uygular:

- **Getirme:** Komutlar önbellekten veya harici bellekten getirilir ve iki adet 16 baytlık önbellek tamponundan birine yerleştirilir. Getirme aşamasının amacı, eski veriler komut kod çözücü tarafından tüketilir tüketilmez, prefetch tamponlarını yeni verilerle doldurmaktır. Komutlar değişken uzunlukta olduğundan (ön ekler sayılmasa 1 ila 11 bayt arasında), ön getiricinin diğer boru hattı aşamalarına göre durumu komuttan komuta değişir. Ortalama olarak, her 16 baytlık yüklemeye yaklaşık beş komut getirilir [CRAW90]. Getirme aşaması, ön getirme tamponlarını dolu tutmak için diğer aşamalardan bağımsız olarak çalışır.
- **Kod çözme aşaması 1:** Tüm işlem kodu ve adresleme modu bilgileri D1 aşamasında çözülür. Gerekli bilgiler ve komut uzunluğu bilgisi, komutun en fazla ilk 3 baytında yer alır. Bu nedenle, 3 bayt ön-getirme tamponlarından D1 aşamasına aktarılır. D1 kod çözücü daha sonra D2 aşamasını, D1 kod çözme işlemine dahil olmayan komutun geri kalanını (yer değiştirme ve anlık veri) yakalaması için yönlendirebilir.
- **Kod çözme aşaması 2:** D2 aşaması her bir işlem kodunu ALU için kontrol sinyallerine genişletir. Ayrıca daha karmaşık adresleme modlarının hesaplamasını da kontrol eder.
- **Yürütme:** Bu aşama ALU işlemlerini, önbellek erişimini ve kayıt güncellemesini içerir.

## 516 BÖLÜM 14 / İŞLEMCİ YAPISI VE İŞLEVİ

- Geri yaz:** Bu aşama, gerekirse, önceki yürütme aşaması sırasında değiştirilen kayıtları ve durum bayraklarını günceller. Geçerli komut belleği güncellerse, hesaplanan değer aynı anda önbelleğe ve veri yolu arayüzü yazma tamponlarına gönderilir.

İki kod çözme aşamasının kullanılmasıyla, boru hattı saat döngüsü başına bir komuta yakın bir verimi südürebilir. Karmaşık komutlar ve koşullu dallanmalar bu hızı yavaşlatabilir.

Şekil 14.21'de boru hattının çalışmasına ilişkin örnekler gösterilmektedir. Şekil 14.21a, bir bellek erişimi gerektiğinde boru hattına herhangi bir gecikme eklenmediğini göstermektedir. Ancak, Şekil 14.21b'de gösterildiği gibi, bellek adreslerini hesaplamak için kullanılan değerler için bir gecikme olabilir. Yani, bir değer bellekten bir yazmaca yüklenirse ve bu yazmaca daha sonra bir sonraki komutta temel yazmaca olarak kullanılırsa, işlemci bir döngü boyunca duracaktır. Buörnekte, işlemci ilk komutun EX aşamasında önbelleğe erişir ve WB aşamasında alınan değeri yazmaca saklar. Ancak, bir sonraki komut D2 aşamasında bu yazmaca ihtiyaç duyar. D2 aşaması bir önceki komutun WB aşamasıyla aynı hizaya geldiğinde, bypass sinyal yolları D2 aşamasının WB aşaması tarafından yazma için kullanılan aynı veriye erişmesini sağlayarak bir boru hattı aşamasından tasarruf sağlar.

Şekil 14.21c, dallanmanın varsayıarak bir dallanma komutunun zamanlamasını göstermektedir. Karşılaştırma talimatı WB aşamasındaki koşul kodlarını günceller ve bypass yolları bunu aynı zamanda atlama talimatının EX aşaması için kullanılabilir hale getirir. Paralel olarak, işlemci atlama komutunun EX aşaması sırasında atlama hedefine spekulatif bir getirme döngüsü çalıştırır. İşlemci yanlış bir dallanma koşulu belirlerse, bu ön getirmeyi atar ve yürütmeye bir sonraki sıralı komutla (zaten getirilmiş ve kodu çözülmüş) devam eder

Getir	D1	D2	EX	WB		MOV Reg1, Mem1	
	Getir	D1	D2	EX	WB	MOV Reg1, Reg2	
		Getir	D1	D2	EX	WB	MOV Mem2, Reg1

(a) Boru hattında veri yükleme gecikmesi yok

Getir	D1	D2	EX	WB		MOV Reg1, Mem1
	Getir	D1		D2	EX	MOV Reg2, (Reg1)

(b) İşaretçi yükleme gecikmesi

Getir	D1	D2	EX	WB		CMP Reg1, Imm
	Getir	D1	D2	EX		Jcc Hedef Hedef
			Getir	D1	D2	EX

(c) Dallanma talimatı zamanlaması

**Şekil 14.21** 80486 Komut İşlem Hattı Örnekleri

## 14.5 x86 İŞLEMCİ AİLESİ

x86 organizasyonu yıllar içinde önemli ölçüde gelişmiştir. Bu bölümde, tek işlemcilerdeki ortak unsurlara odaklanarak en yeni işlemci organizasyonlarının bazı ayrıntılarını inceleyeceğiz. Bölüm 16 x86'nın superskalar yönlerini, Bölüm 18 ise çok çekirdekli organizasyonu incelemektedir. Pentium 4 işlemci organizasyonunun bir üst görünümü Şekil 4.18'de gösterilmektedir.

### Kayıt Kuruluşu

Kayıt organizasyonu aşağıdaki kayıt türlerini içerir (Tablo 14.2):

- Genel:** Sekiz adet 32-bit genel amaçlı yazmaç vardır (bkz. Şekil 14.3c). Bunlar her tür x86 komutu için kullanılabilir; ayrıca adres hesaplamaları için operatörleri de tutabilirler. Ayrıca, bu yazmaçlardan bazıları özel amaçlara da hizmet eder. Örneğin, string komutları ECX, ESI ve EDI kayıtlarının içeriğini, komutta bu kayıtlara açıkça atıfta bulunmak zorunda kalmadan işlenen olarak kullanır. Sonuç olarak, dizi talimat

**Tablo 14.2** x86 İşlemci Kayıtları

(a) 32-bit Modunda Tamsayı Birimi

Tip	Sayı	Uzunluk (bit)	Amaç
Genel	8	32	Genel amaçlı kullanıcı kayıtları
Segment	6	16	Segment seçicileri içerir
EFLAGS	1	32	Durum ve kontrol bitleri
Talimat İşaretçisi	1	32	Talimat işaretçisi

(b) 64-bit Modunda Tamsayı Birimi

Tip	Sayı	Uzunluk (bit)	Amaç
Genel	16	32	Genel amaçlı kullanıcı kayıtları
Segment	6	16	Segment seçicileri içerir
RFLAGS	1	64	Durum ve kontrol bitleri
Talimat İşaretçisi	1	64	Talimat işaretçisi

(c) Kayan Nokta Birimi

Tip	Sayı	Uzunluk (bit)	Amaç
Sayısal	8	80	Kayan noktalı sayıları tutma
Kontrol	1	16	Kontrol bitleri
Durum	1	16	Durum bitleri
Etiket Kelime	1	16	Sayısal kayıtların içeriğini belirtir
Talimat İşaretçisi	1	48	İstisna tarafından kesintiye uğratılan talimata işaret eder
Veri İşaretçisi	1	48	İstisna tarafından kesintiye uğratılan işlenene işaret eder

daha kompakt bir şekilde kodlanmıştır. 64 bit modunda, on altı adet 64 bit genel amaçlı yazmaç vardır.

- **Segment:** Altı adet 16-bit segment kaydedicisi, Bölüm 8'de tartışıldığı gibi segment tablolara indekslenen segment seçicileri içerir. Kod segmenti (CS) kaydı, yürütülmekte olan komutu içeren segmente referans verir. Yiğin segmenti (SS) kaydı, kullanıcı tarafından görülebilen bir yiğin içeren segmenti referans alır. Kalan segment register'ları (DS, ES, FS, GS) kullanıcının bir seferde en fazla dört ayrı veri segmentine referans vermesini sağlar.
- **Bayraklar:** 32 bitlik EFLAGS kaydı koşul kodlarını ve çeşitli mod bitlerini içerir. 64 bit modunda, bu kayıt 64 bite genişletilir ve RFLAGS olarak. Mevcut mimari tanımında, RFLAGS'nın üst 32 biti kullanılmamaktadır.
- **Komut işaretçisi:** Geçerli komutun adresini içerir. Özel olarak kayan nokta birimine ayrılmış yazmaçlar da vardır:
- **Sayısal:** Her yazmaç, genişletilmiş hassasiyetli 80-bit kayan noktalı sayı tutar. Yiğin olarak işlev gören sekiz yazmaç vardır ve komut setinde push ve pop işlemleri mevcuttur.
- **Kontrol:** 16 bitlik kontrol kaydı, yuvarlama kontrolü türü; tek, çift veya genişletilmiş hassasiyet; ve çeşitli istisna koşullarını etkinleştirmek veya devre dışı bırakmak için bitler dahil olmak üzere kayan nokta biriminin çalışmasını kontrol eden bitler içerir.
- **Durum:** 16 bitlik durum kayıt, yiğinin tepesine 3 bitlik bir işaretçi, son işlemin sonucunu bildiren durum kodları ve istisna bayrakları dahil olmak üzere kayan nokta biriminin mevcut durumunu yansitan bitler içerir.
- **Etiket kelimesi:** Bu 16 bitlik kayıt, her bir kayan noktalı sayısal kayıt için, karşılık gelen kaydın içeriğinin niteliğini gösteren 2 bitlik bir etiket içerir. Dört olası değer geçerli, sıfır, özel (NaN, sonsuz, denormalize) ve boştur. Bu etiketler, programların kayittaki gerçek verilerin karmaşık kod çözme işlemini gerçekleştirmeden sayısal bir kaydın içeriğini kontrol etmesini sağlar. Örneğin, bir bağlam geçişi yapıldığında, işlemcinin boş olan kayan noktalı kayıtları kaydetmesi gerekmekz.

Yukarıda bahsedilen kayıtların çoğunun kullanımı kolayca anlaşılabılır. Bazı kayıtları kısaca açıklayalım.

**EFLAGS REGISTER** EFLAGS register (Şekil 14.22) işlemcinin durumunu gösterir ve çalışmasını kontrol etmeye yardımcı olur. Bir tamsayı işleminin sonuçlarını bildiren Tablo 12.9'da tanımlanan altı durum kodunu (carry, parity, auxiliary, zero, sign, overflow) içerir. Ayrıca, yazmaçta kontrol bitleri olarak bitler de vardır:

- **Tuzak bayrağı (TF):** Ayarlandığında, her komutun yürütülmesinden sonra bir kesmeye neden olur. Bu hata ayıklama için kullanılır.
- **Kesme etkinleştirme bayrağı (IF):** Ayarlandığında, işlemci harici kesmeleri tanıယacaktır.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	I D	V I P	V I F	A C	V M	R F	0	N T	I O P L	O F	D F	I F	T F	S F	Z F	0	A F	0	P F	1	C F	

X ID = Tanımlama bayrağı  
X VIP= Sanal kesme beklemede X VIF= Sanal kesme bayrağı  
X AC = Hızalama kontrolü X  
VM= Sanal 8086 modu X RF = Özgeçmiş bayrağı  
X NT = İç içe görev bayrağı  
X IOPL = G/C ayrıcalık düzeyi S OF = Taşma bayrağı

C DF= Yön bayrağı  
X IF= Kesme etkinleştirme bayrağı X TF= Tuzak bayrağı  
S SF= İşaret bayrağı S ZF= Sıfır bayrağı  
S AF= Yardımcı taşıma bayrağı S PF= Eşlik bayrağı  
S CF= Taşıma bayrağı

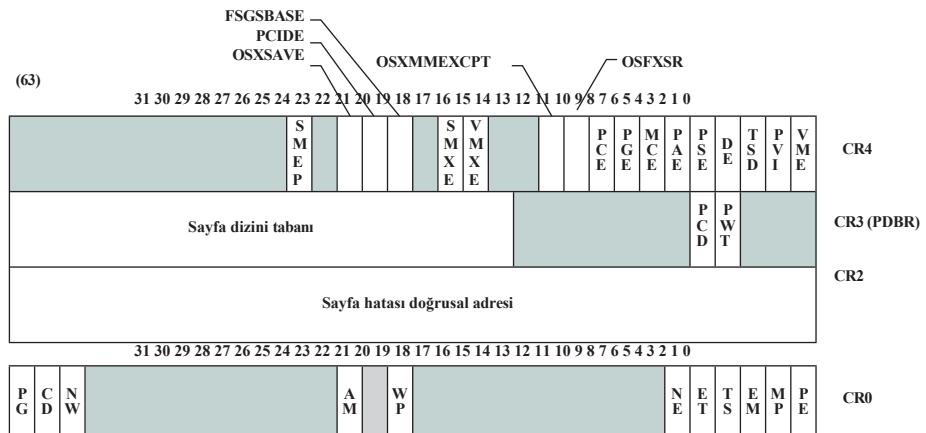
S bir durum bayrağını gösterir. C bir kontrol bayrağını gösterir. X bir sistem bayrağını gösterir.  
Gölgeli bitler ayrılmıştır.

**Sekil 14.22** x86 EFLAGS Kaydı

- **Yön bayrağı (DF):** Dize işleme talimatlarının 16 bitlik yarı kayıtlar SI ve DI'yi (16 bitlik işlemler için) veya 32 bitlik kayıtlar ESI ve EDI'yi (32 bitlik işlemler için) artırmayacağını veya azaltacağını belirler.
- **G/C ayrıcalık bayrağı (IOPL):** Ayarlandığında, işlemcinin korumalı mod çalışması sırasında G/C aygıtlarına tüm erişimlerde bir istisna oluşturmasına neden olur.
- **Devam bayrağı (RF):** Programcının hata ayıklama istisnalarını devre dışı bırakmasına izin verir, böylece komut bir hata ayıklama istisnasından sonra hemen başka bir hata ayıklama istisnasına neden olmadan yeniden başlatılabilir.
- **Hızalama kontrolü (AC):** Bir sözcük veya çift sözcük, sözcük olmayan veya çift sözcük olmayan bir sınırdı adreslenirse etkinleştir.
- **Tanımlama bayrağı (ID):** Bu bit ayarlanabiliyor ve temizlenebiliyorsa, bu işlemci processorID komutunu destekliyor demektir. Bu komut, satıcı, aile ve model hakkında bilgi sağlar.

Buna ek olarak, çalışma modıyla ilgili 4 bit vardır. İç İçe Görev (NT) bayrağı, mevcut görevin korumalı mod çalışmasında başka bir görevin içinde iç içe olduğunu gösterir. Sanal Mod (VM) biti, programcının 8086 makinesi olarak çalışıp çalışmayaacağını belirleyen sanal 8086 modunu etkinleştirmesini veya devre dışı bırakmasını sağlar. Virtual Interrupt Flag (VIF) ve Virtual Interrupt Pending (VIP) bayrağı çoklu görev ortamında kullanılır.

**KONTROL KAYITLARI** x86, işlemci çalışmasının çeşitli yönlerini kontrol etmek için dört kontrol kaydı kullanır (CR1 kaydı kullanılmaz) (Şekil 14.23). CR0 dışındaki tüm kayıtlar, uygulamanın x86 64 bit mimarisini destekleyip desteklemediğine bağlı olarak 32 bit veya 64 bit uzunluğundadır. CR0 kaydedicisi, modları kontrol eden veya genel olarak geçerli olan durumları gösteren sistem kontrol bayraklarını içerir



Gölgeli alan ayrılmış bitleri gösterir.

<b>OSXSAVE</b>	= XSAVE etkinleştirme biti	<b>VME</b>	= Sanal 8086 modu uzantıları
<b>PCIDE</b>	= Süreç bağlamı tanımlayıcılarını etkinleştirir	<b>PCD</b>	= Sayfa düzeyinde önbellek devre dışı
<b>FSGSBASE</b>	= Segment tabanı taliimatlarını etkinleştirir	<b>PWT</b>	= Sayfa düzeyinde şeffaf yazma
<b>SMXE</b>	= Daha güvenli mod uzantılarını etkinleştirir	<b>PG</b>	= Çağrı
<b>VMXE</b>	= Sanal makine uzantılarını etkinleştirme	<b>CD</b>	= Önbellek devre dışı
<b>OSXXMEXCPT</b>	= Masklenmemiş SIMD FP istisnalarını destekleme	<b>NW</b>	= Yazarak değil
<b>OSFXSR</b>	= FXSAVE, FXSTOR desteği	<b>AM</b>	= Hızalama maskesi
<b>PCE</b>	= Performans sayacı etkinleştirme	<b>WP</b>	= Koruma yazın
<b>PGE</b>	= Sayfa genel etkinleştirme	<b>NE</b>	= Sayısal hata
<b>MCE</b>	= Makine kontrolü etkinleştirme	<b>ET</b>	= Uzatma tipi
<b>PAE</b>	= Fiziksel adres uzantısı	<b>TS</b>	= Görev değiştirildi
<b>PSE</b>	= Sayfa boyutu uzantıları	<b>EM</b>	= Emülatyon
<b>DE</b>	= Hata ayıklama uzantıları	<b>MP</b>	= Yardımcı işlemciyi izleyin
<b>TSD</b>	= Zaman damgası devre dışı	<b>PE</b>	= Koruma etkinleştirme
<b>PVI</b>	= Korumalı mod sanal kesme		

**Şekil 14.23** x86 Kontrol Kayıtları

tek bir görevin yürütülmesinden ziyade işlemciye yönelikdir. Bayraklar aşağıdaki gibidir:

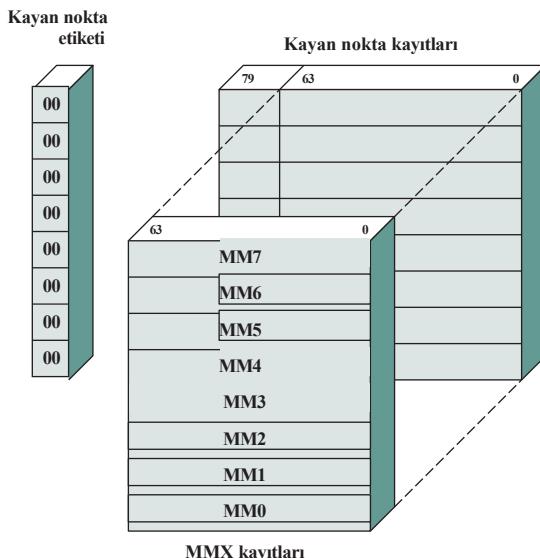
- **Koruma Etkinleştirme (PE):** Korumalı çalışma modunu etkinleştirir/devre dışı bırakır.
- **Monitor Coprocessor (MP):** Yalnızca x86 üzerinde daha eski makinelerdeki programları çalıştırırken ilgi çeker; aritmetik yardımcı işlemcinin varlığıyla ilgilidir.
- **Emülatyon (EM):** İşlemcinin kayan nokta birimi olmadığındada ayarlanır ve kayan nokta komutları yürütülmeye çalışıldığında bir kesmeye neden olur.
- **Görev Değiştirildi (TS):** İşlemcinin görevleri değiştirdiğini gösterir.
- **Uzantı Türü (ET):** Pentium ve sonraki makinelerde kullanılmaz; önceki makinelerde matematik yardımcı işlemci taliimatlarının desteğini belirtmek için kullanılır.

- **Sayısal Hata (NE):** Harici veri yolu hatlarındaki kayan nokta hatalarını bildirmek için standart mekanizmayı etkinleştirir.
- **Yazma Koruması (WP):** Bu bit olduğunda, salt okunur kullanıcı düzeyi sayfalar bir gözetmen işlemi tarafından yazılabılır. Bu özellik bazı işletim sistemlerinde süreç oluşturmayı desteklemek için kullanılmıştır.
- **Hızalama Maskesi (AM):** Hızalama denetimini etkinleştirir/devre dışı bırakır.
- **Üzerinden Yazılmıyor (NW):** Veri çalışma modunu seçer. Bu bit ayarlandığında, veri önbelleğinin önbelleğe yazma işlemleri engellenir.
- **Önbellek Devre Dışı (CD):** Dahili önbellek doldurma mekanizmasını etkinleştirir/devre dışı bırakır.
- **Çağrı (PG):** Çağrı özelliğini etkinleştirir/devre dışı bırakır.

Sayfalama etkinleştirildiğinde, CR2 ve CR3 kayıtları geçerlidir. CR2 kaydı, bir sayfa hatası kesintisinden önce erişilen son sayfanın 32 bit doğrusal adresini tutar. CR3'ün en soldakı 20 biti sayfa dizininin temel adresinin en anlamlı 20 bitini tutar; adresin geri kalani sıfırlar içerir. CR3'ün iki biti harici bir önbelleğin çalışmasını kontrol eden pinleri sormak için kullanılır. Sayfa seviyesi önbellek devre dışı bırakma (PCD) harici önbelleği etkinleştirir veya devre dışı bırakır ve sayfa seviyesi şeffaf yazma (PWT) biti harici yazmayı kontrol eder. CR4 ek kontrol bitleri içerir.

**MMX KAYITLARI** Bölüm 10.3'ten x86 MMX özelliğinin çeşitli 64-bit veri tiplerini kullandığını hatırlayın. MMX komutları 3 bitlik kayıt adres alanlarını kullanır, böylece sekiz MMX kaydı desteklenir. Aslında, işlemci belirli MMX kayıtları içermez. Bunun yerine, işlemci bir takma ad teknigi kullanır (Şekil 14.24). MMX değerlerini saklamak için mevcut kayan noktalı yazmaçlar kullanılır. Özellikle, her bir kayan nokta kaydının düşük mertebeli 64 biti (mantissa) sekiz MMX kaydını oluşturmak için kullanılır. Böylece, eski 32-bit x86 mimarisi MMX özelliğini desteklemek için kolayca genişletilebilir. Bu kayıtların MMX kullanımının bazı temel özellikleri aşağıdaki gibidir:

- Kayan nokta kayıtlarının kayan nokta işlemleri için bir yığın olarak ele alındığını hatırlayın. MMX işlemleri için aynı kayıtlara doğrudan erişilir.
- Herhangi bir kayan nokta işleminden sonra bir MMX komutu ilk kez yürütüldüğünde, FP etiket sözcüğü geçerli olarak işaretlenir. Bu, yığın işleminden doğrudan kayıt adreslemeye geçişini yansıtır.
- EMMS (Boş MMX Durumu) komutu, tüm kayıtların boş olduğunu belirtmek için FP etiket sözcüğünün bitlerini ayarlar. Sonraki kayan nokta işlemlerinin düzgün çalışması için programcının bu komutu MMX kod bloğunun sonuna eklemesi önemlidir.
- Bir MMX kaydına bir değer yazıldığında, ilgili FP kaydının [79:64] bitleri ( işaret ve üs bitleri ) tüm birlere ayarlanır. Bu, kayan nokta değeri olarak bakıldığıda FP yazmacındaki değeri NaN (sayı değil) veya sonsuz olarak ayarlar. Bu, bir MMX veri değerinin geçerli bir kayan nokta değeri gibi görünmemesini sağlar.



**Şekil 14.24** MMX Kayıtlarının Kayan Nokta Kayıtlarına Eşlenmesi

## Kesme İşleme

Bir işlemci içindeki kesme işlemi, işletim sistemini desteklemek için sağlanan bir olanaktır. Bir uygulama programının, çeşitli kesme koşullarına hizmet edilebilmesi ve daha sonra devam ettirilebilmesi için aşağıya alınmasına izin verir.

**KESMELER VE İSTİSNALAR** İki olay sınıfı x86'nın mevcut komut akışının yürütülmesini aşağımasına ve olaya yanıt vermesine neden olur: kesmeler ve istisnalar. Her iki durumda da, işlemci mevcut işlemin bağlamını kaydeder ve duruma hizmet etmek için önceden tanımlanmış bir rutine aktarır. *Kesme*, donanımdan gelen bir sinyal tarafından oluşturulur ve bir programın yürütülmESİ sırasında rastgele zamanlarda meydana gelebilir. Bir *istisna* yazılımdan üretilir ve bir talimatın yürütülmESİyle tetiklenir. İki kesme kaynağı ve iki istisna kaynağı vardır:

### 1. Kesintiler

- **Maskelenebilir kesmeler:** İşlemcinin INTR pininden alınır. Kesme etkinleştirme bayrağı (IF) ayarlanmadığı sürece işlemci maskelenebilir bir kesmeyi tanımaz.
- **Maskelenemeyen kesmeler:** İşlemcinin NMI pininden alınır. Bu tür kesmelerin algılanması engellenemez.

### 2. İstisnalar

- **İşlemci tarafından algılanan istisnalar:** İşlemci bir komutu执行meye çalışırken bir hataya karşılaştığında ortaya çıkar.

- **Programlanmış istisnalar:** Bunlar bir istisna oluşturan talimatlardır (örneğin, INTO, INT3, INT ve BOUND).

KESME **VEKTÖR TABLOSU** x86'daki kesme işlemleri kesme vektör tablosunu kullanır. Her kesme türüne bir numara atanır ve bu numara kesme vektör tablosunu indekslemek için kullanılır. Bu tablo, söz konusu kesme numarası için kesme hizmet rutininin adresi (segment ve offset) olan 256 adet 32 bit kesme vektörü içerir.

Tablo 14.3, kesme vektör tablosundaki numaraların atamasını göstermektedir; gölgeli girişler kesmeleri temsil ederken, gögesiz girişler istisnalardır. NMI donanım kesmesi tip 2'dir. INTR donanım kesmelerine 32 ile 255 aralığında numaralar atanır; bir INTR kesmesi oluşturulduğunda, bu kesme için kesme vektör numarası ile veri yoluna eşlik edilmelidir. Kalan vektör numaraları istisnalar için kullanılır.

Birden fazla istisna veya kesme beklemeyeseye, işlemci bunları öngörelebilir bir sırayla servis eder. Vektör numaralarının tablo içindeki konumu önceliği yansıtmez. Bunun yerine, istisnalar ve kesmeler arasındaki öncelik beş sınıf halinde düzenlenmiştir. Azalan öncelik sırasına göre bunlar

- **Sınıf 1:** Önceki komutta tuzaklar (vektör numarası 1)
- **Sınıf 2:** Harici kesmeler (2, 32-255)
- **Sınıf 3:** Sonraki komutun getirilmesinden kaynaklanan hatalar (3, 14)
- **Sınıf 4:** Bir sonraki talimatın kodunun çözülmesinden kaynaklanan hatalar (6, 7)
- **Sınıf 5:** Bir komutun yürütülmesi sırasında oluşan hatalar (0, 4, 5, 8, 10-14, 16, 17)

KESME **İŞLEMI** Tıpkı bir CALL komutu kullanılarak yapılan yürütme aktarımında olduğu gibi, bir kesme işleme rutinine yapılan aktarım da işlemci durumunu saklamak için sistem yiğinini kullanır. Bir kesme oluştuğunda ve işlemci tarafından tanındığında, dizi olay gerçekleşir:

1. Aktarım bir ayrıcalık seviyesi değişikliği içeriyorsa, geçerli yiğin segmenti kaydı ve geçerli genişletilmiş yiğin işaretçisi (ESP) kaydı yiğina itilir.
2. EFLAGS kaydının geçerli değeri yiğine itilir.
3. Hem kesme (IF) hem de tuzak (TF) bayrakları temizlenir. Bu, INTR kesmelerini ve tuzak veya tek adım özelliğini devre dışı bırakır.
4. Geçerli kod segmenti (CS) işaretçisi ve geçerli komut işaretçisi (IP veya EIP) yiğine itilir.
5. Kesmeye bir hata kodu eşlik ediyorsa, hata kodu yiğine itilir.
6. Kesme vektörü içeriği alınır ve CS ve IP veya EIP kayıtlarına yüklenir. Yürütme, kesme hizmet rutininden devam eder.

Bir kesmeden dönmek için, kesme hizmet rutini bir IRET komutu yürütür. Bu, yiğinda kayıtlı tüm değerlerin geri yüklenmesine neden olur; yürütme, kesme noktasından devam eder.

Tablo 14.3 x86 İstisna ve Kesme Vektör Tablosu

Vektör Numarası	Açıklama
0	Bölme hatası; bölüm taşması veya sıfır bölüm
1	Hata ayıklama istisnası; hata ayıklama ile ilgili çeşitli hataları ve tuzakları içerir
2	NMI pini kesmesi; NMI pini üzerindeki sinyal
3	Kesme noktası; 1 baylıklı bir komut olan INT 3 komutunun neden olduğu hata ayıklama
4	INTO tarafından algılanan taşıma; işlemci INTO'yu OF ile yürütüğünde oluşur bayrak seti
5	BOUND aralığı aşındı; BOUND talimatı bir kaydı bound- ile karşılaşır. bellekte saklanır ve kaydın içeriği bellekte saklananın dışındaysa bir kesme oluşturur. sınırların
6	Tanımsız işlem kodu
7	Aygıt mevcut değil; ESC veya WAIT komutunu kullanma girişimi, aygıt olmaması nedeniyle başarısız harici cihaz
8	Çift hata; aynı komut sırasında iki kesme meydana gelir ve işlenemez seri olarak
9	Ayrılmış
10	Geçersiz görev durumu segmenti; istenen bir görevi tanımlayan segment başlatılmamış veya geçerli değil
11	Segment mevcut değil; gerekli segment mevcut değil
12	Yığın hatası; yığın segmenti sınırı aşındı veya yığın segmenti mevcut değil
13	Genel koruma; başka bir istisnaya neden olmayan koruma ihlali (örn, salt okunur bir segmente yazma)
14	Sayfa hatası
15	Ayrılmış
16	Kayan nokta hatası; bir kayan nokta aritmetik talimatı tarafından oluşturulur
17	Hızalama kontrolü; tek bayt adresinde saklanan bir sözcüğe veya bir çift sözcüğe erişim 4'ün katı olmayan bir adreste saklanır
18	Makine kontrolü; modele özel
19-31	Ayrılmış
32-255	Kullanıcı kesme vektörleri; INTR sinyali etkinleştirildiğinde sağlanır

Gölgeliler: istisnalar Gölgeli:  
kesintiler

## 14.6 KOL İŞLEMÇİSİ

Bu bölümde, ARM mimarisinin ve organizasyonunun bazı temel unsurlarına bakacağız. Organizasyon ve boru hattının daha karmaşık yönlerinin tartışımasını Bölüm 16'ya erteliyoruz. Bu bölümdeki ve Bölüm 16'daki tartışmalar için ARM mimarisinin temel özelliklerini akılda tutmak faydalı olacaktır. ARM öncelikli olarak aşağıdaki önemli özelliklere sahip bir RISC sistemidir:

- Bazı CISC sistemlerinde bulunandan daha fazla, ancak birçok RISC sisteminde bulunandan daha az olan orta düzeyde bir dizi tek tip yazmaç.
- İşlemlerin doğrudan bellekte değil, yalnızca yazımcılardaki işlenenler üzerinde gerçekleştirildiği bir veri işleme yükleme/depolama modeli. Bir işlem gerçekleştirilmeden önce tüm veriler kayıtlara yüklenmelidir; sonuç daha sonra başka işlemler için kullanılabilir veya belleğe depolanabilir.
- Standart set için 32 bitlik ve Thumb komut seti için 16 tek tip sabit uzunlukta bir komut.
- Her bir veri işleme komutunu daha esnek hale getirmek için, bir kaydırma veya döndürme kaynak kayıtlarından birini önceden işleyebilir. Bu özelliği verimli bir şekilde desteklemek için ayrı aritmetik mantık birimi (ALU) ve kaydırıcı birimleri vardır.
- Tüm yükleme/depolama adreslerinin yazımcılardan ve komut alanlarından belirlendiği az sayıda adresleme modu. Bellekteki değerleri içeren dolaylı veya indeksli adresleme kullanılmaz.
- Otomatik artırma ve otomatik azaltma adresleme modları, program döngülerinin çalışmasını iyileştirmek için kullanılır.
- Talimatların koşullu olarak yürütülmesi, koşullu dallanma talimatlarına olan ihtiyacı en aza indirir ve böylece boru hattı verimliliğini artırır, çünkü boru hattının yılanması azalır.

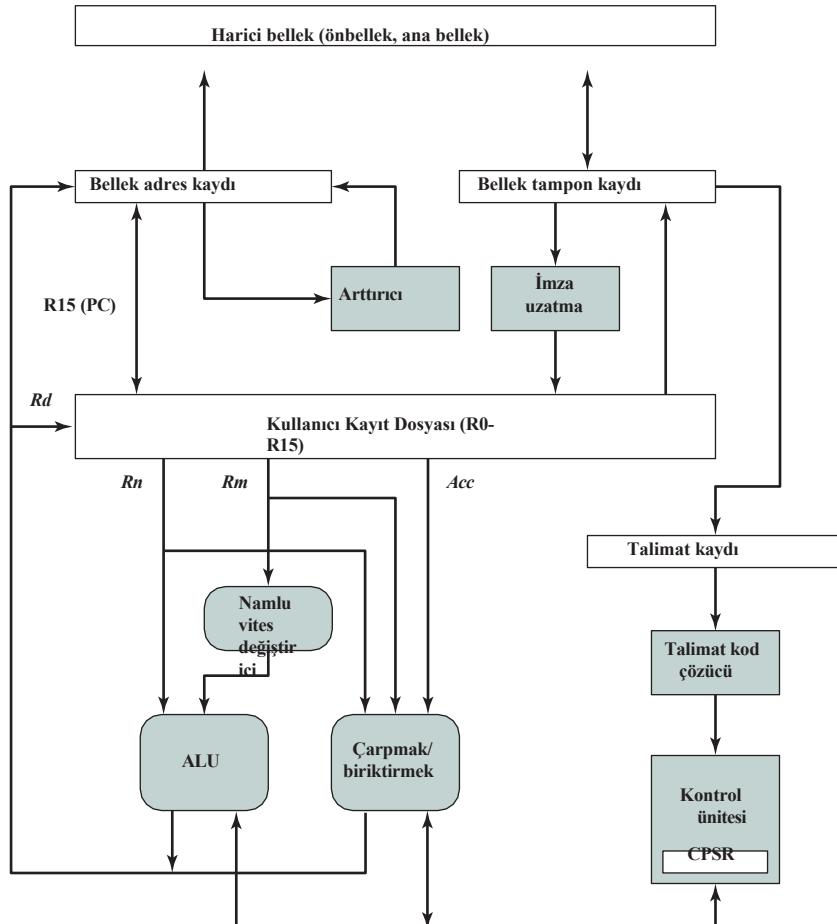
## İşlemci Organizasyonu

ARM işlemci organizasyonu, özellikle ARM mimarisinin farklı sürümlerine dayandığında, bir uygulamadan diğerine önemli ölçüde değişir. Ancak, bölümdeki tartışma için Şekil 14.25'te gösterilen basitleştirilmiş, genel bir ARM organizasyonu summak yararlı olacaktır. Bu şekilde oklar veri akışını göstermektedir. Her kutu işlevsel bir donanım birimini veya bir depolama birimini temsil etmektedir.

Veriler, bir veri yolu aracılığıyla harici bellekten işlemciyle değişim tokusu edilir. Aktarılan değer, bir yükleme veya saklama komutunun sonucu olarak bir veri öğesi ya da bir komut getirme işlemidir. Getirilen talimatlar yürütülmeden önce bir kontrol ünitesinin kontrolü altında bir talimat kod çözücüinden geçer. Bu birim boru hattı mantığı içerir ve işlemcinin tüm donanım elemanlarına kontrol sinyalleri (gösterilmemiştir) sağlar. Veri öğeleri, 32 bitlik bir dizi kayıtten oluşan kayıt dosyasına yerleştirilir. İkili tamamlayıcı sayılar olarak ele alınan bayt veya yarımkilime öğeleri 32 bite genişletilir.

ARM veri işleme talimatları tipik olarak *Rn* ve *Rm* olmak üzere iki kaynak ve *Rd* olmak üzere tek bir sonuç veya hedef sahiptir. Kaynak kayıt değerleri ALU'ya ya da kısmi sonuçları biriktirmek için ek bir kayıtten yararlanan ayrı bir çarpma birimine beslenir. ARM işlemcisi ayrıca *Rm* değerini ALU'ya girmeden önce kaydırabilen veya döndüribilecek bir donanım birimi içerir. Bu kaydırma veya döndürme işlemi komutun döngü süresi içinde gerçekleşir ve birçok veri işleme işleminin gücünü ve esnekliğini artırır.

Bir işlemin sonuçları hedef geri beslenir. Yükleme/depolama talimatları, bir yükleme veya depolama için bellek adresi oluşturmak üzere aritmetik birimlerin çıkışını da kullanabilir.



Şekil 14.25 Basitleştirilmiş ARM Organizasyonu

## İşlemci Modları

Bir işlemcinin yalnızca az sayıda işlemci modunu desteklemesi oldukça yaygındır. Örneğin, birçok işletim sistemi sadece iki moddan yararlanır: bir kullanıcı modu ve bir çekirdek modu, ikinci mod ayrıcalıklı sistem yazılımını çalıştırmak için kullanılır. Buna karşın ARM mimarisi, işletim sistemlerinin çeşitli koruma politikaları uygulaması için esnek bir temel sağlar.

ARM mimarisi yedi yürütme modunu destekler.Çoğu uygulama programı **kullanıcı modunda** yürütülür. İşlemci kullanıcı , çalıştırılan program korumalı sistem kaynaklarına erişemez veya bir istisna oluşmasına neden olmak dışında modu değiştiremez.

Kalan altı yürütme modu ayrıcalıklı modlar olarak adlandırılır. Bu modlar sistem yazılımını çalıştırmak için kullanılır. Bu kadar çok farklı ayrıcalıklı mod tanımlamanın iki temel avantajı vardır: (1) İşletim sistemi, sistem yazılımının kullanımını çeşitli koşullara göre uyarlayabilir ve (2) belirli kayıtlar ayrıcalıklı modların her biri için kullanılmak üzere ayrılmıştır, bu da bağlamdaki daha hızlı değişikliklere izin verir.

İstisna modları sistem kaynaklarına tam erişime sahiptir ve modları serbestçe değiştirebilir. Bu modlardan beşi istisna modları olarak bilinir. Bunlar belirli istisnalar oluşturgunda girilir. Bu modların her biri, bazı kullanıcı modu kayıtlarının yerine geçen ve istisna oluşturgunda Kullanıcı modu durum bilgilerinin bozulmasını önlemek için kullanılan bazı özel kayıtlara sahiptir. İstisna modları aşağıdaki gibidir:

- **Gözetmen modu:** Genellikle işletim sisteminin çalıştığı moddur. İşlemci bir yazılım kesme komutıyla karşılaşlığında girilir. Yazılım kesmeleri ARM üzerinde işletim sistemi hizmetlerini çağırmak için standart bir yoldur.
- **İptal modu:** Bellek hatalarına yanıt olarak girilir.
- **Tanımsız mod:** İşlemci, ne ana tamsayı çekirdeği ne de yardımcı işlemcilerden biri tarafından desteklenmeyen bir talimatı yürütmeye çalışlığında girilir.
- **Hızlı kesme modu:** İşlemci belirlenen hızlı kesme kaynağından bir kesme sinyali alduğında girilir. Bir hızlı kesme kesintiye uğratılamaz, ancak bir hızlı kesme normal bir kesmeyi kesintiye uğratabilir.
- **Kesme modu:** İşlemci herhangi bir kesme kaynağından (hızlı kesme dışında) bir kesme sinyali alduğında girilir. Bir kesme yalnızca bir hızlı kesme tarafından kesilebilir.

Kalan ayrıcalıklı **mod Sistem modudur**. Bu moda herhangi bir istisna ile girilmez ve Kullanıcı modunda bulunan aynı kayıtları kullanır. Sistem modu belirli ayrıcalıklı işletim sistemi görevlerini çalıştmak için kullanılır. Sistem modu görevleri beş istisna kategorisinden herhangi biri tarafından kesintiye uğratılabilir.

## Kayıt Kuruluşu

Şekil 14.26'da ARM için kullanıcı tarafından görülebilen yazımcılar gösterilmektedir. ARM işlemcisinde aşağıdaki gibi sınıflandırılmış toplam 37 adet 32-bit yazmaç bulunmaktadır:

- ARM kılavuzunda genel amaçlı yazımcılar olarak adlandırılan otuz bir yazmaç. Aslında, program sayaçları gibi bunlardan bazılarının özel amaçları vardır.
- Altı program durum kaydı.

Kayıtlar kısmen üst üste binen bankalar halinde düzenlenmiştir ve hangi bankanın kullanılabilir olduğunu mevcut pro-külatör modu belirler. Herhangi bir zamanda, yazılım tarafından görülebilen toplam 17 veya 18 yazmaç için on altı numaralı ve bir veya iki program durum yazmacı görülebilir. Şekil 14.26 aşağıdaki gibi yorumlanır:

- R0'dan R7'ye kadar olan kayıtlar, R15 kaydı (program sayacı) ve mevcut program durum kaydı (CPSR) tüm modlarda görülebilir ve tüm modlar tarafından paylaşıılır.
- R8 ile R12 arasındaki kayıtlar, R8\_fiq ile R12\_fiq arasındaki kendi özel kayıtlarına sahip olan hızlı kesme hariç tüm modlar tarafından paylaşırlı.
- Tüm istisna modları R13 ve R14 kayıtlarının kendi versiyonlarına sahiptir.
- Tüm istisna modlarının özel bir kayıtlı program durum kaydı (SPSR) vardır.

Modlar						
Kullanıcı	Sistem	Ayrıcalıklı modlar				
		Süpervizör	İptal	Tanımsız	Kesinti	Hızlı kesme
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13(SP)	R13(SP)	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14(LR)	R14(LR)	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

Gölgelendirme, Kullanıcı veya Sistem modu tarafından kullanılan normal kaydın istisna moduna özgü alternatif bir kayıtla değiştirildiğini gösterir.

SP = yoğun işaretçi

CPSR= geçerli program durum kaydı LR=

bağlantı kaydı

SPSR= kayıtlı program durum kaydı PC=

program sayacı

**Şekil 14.26** ARM Kayıt Organizasyonu

**GENEL AMAÇLI KAYITLAR** R13 yazmacı normalde bir yoğun işaretçi olarak kullanılır ve SP olarak da bilinir. Her istisna modu ayrı bir R13'e sahip olduğundan, her istisna modu kendi özel program yığınına sahip olabilir. R14 bağlantı kaydedicisi (LR) olarak bilinir ve alt program dönüş adreslerini ve istisna modu dönüşlerini tutmak için kullanılır. R15 kaydedicisi program sayacıdır (PC).

**PROGRAM DURUM KAYITLARI** CPSR'ye tüm işlemci modlarında erişilebilir. Her istisna modunda ayrıca, ilgili istisna gerçekleştiğinde CPSR değerini korumak için kullanılan özel bir SPSR vardır.

CPSR'nin en anlamlı 16 biti Kullanıcı modunda görülebilen ve bir programın çalışmasını etkilemek için kullanılabilen kullanıcı bayraklarını içerir (Şekil 14.27). Bunlar aşağıdaki gibidir:

- **Durum kodu bayrakları:** Bölüm 12'de ele alınan N, Z, C ve V bayrakları.
  - **Q bayrağı:** bazı SIMD yönelimli talimatlarda taşıma ve/veya doygunluğun meydana gelip gelmediğini belirtmek için kullanılır.
  - **J bit:** Jazelle olarak bilinen ve tartıştığımız kapsamı dışında kalan özel 8 bit talimatların kullanımını gösterir.
  - **GE[3:0] bitleri:** SIMD komutları [19:16] bitlerini sonucun tek tek baytları veya yarımsözcükleri için Büyüktür veya Eşittir (GE) bayrakları olarak kullanır.

CPSR'nin en az anlamlı 16 biti, yalnızca işlemci ayrıcalıklı bir moddayken değiştirilebilen sistem kontrol bayraklarını içerir. Alanlar aşağıdaki gibidir:

- **E biti:** Veriler için yükleme ve saklama endianlığını kontrol eder; komut getirmeleri için göz ardı edilir.
  - **Kesme devre dışı bırakma bitleri:** A biti kesin olmayan veri iptallerini devre dışı bırakır; I biti ayarlandığında IRQ kesmelerini devre dışı bırakır; ve F biti ayarlandığında FIQ kesmelerini devre dışı bırakır.
  - **T biti:** Talimatların normal ARM talimatları mı yoksa Thumb talimatları olarak mı yorumlanması gerektiğini belirtir.
  - **Mod bitleri:** İşlemci modunu gösterir.

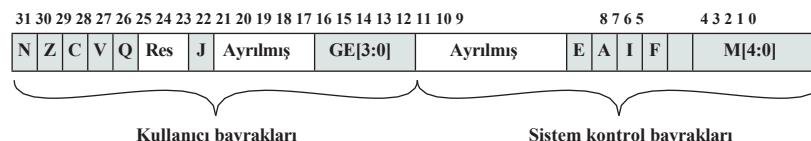
## Kesme İşleme

Her işlemcide olduğu gibi ARM'de de işlemcinin istisnai durumlarla başa çıkmak için o anda yürütülmekte olan programı kesmesini sağlayan bir olanak bulunmaktadır. İstisnalar, işlemcinin bir olayı ele almasına neden olmak dahili ve harici kaynaklar tarafından üretilir. İstisnanın işlenmesinden hemen önceki işlemci durumu normalde korunur, böylece istisna rutini tamamlandığında orijinal programa devam edilebilir. Aynı anda birden fazla istisna ortaya çıkabilir. ARM mimarisi yedi tür istisnayı destekler. Tablo 14.4 istisna türlerini ve her bir türü işlemek için kullanılan işlemci modunu listeler. Bir istisna oluştuğunda, yürütme istisna türüne karşılık gelen sabit bir bellek adresinden zorlanır. Bu sabit adresler istisna vektörleri olarak adlandırılır.

Birden fazla kesinti varsa, bunlar öncelik sırasına göre ele alınır.

Tablo 14.4 istisnaları en yüksekten en düşüğe doğru öncelik sırasına göre listelemektedir.

Bir istisna oluşduğunda, işlemci geçerli komuttan sonra yürütmemeyi durdurur. İşlemcinin durumu, aşağıdakilere karşılık gelen SPSR'de korunur.



**Sekil 14.27** ARM CPSR ve SPSR Formatı

**Tablo 14.4** ARM Kesme Vektörü

İstisna türü	Mod	Normal giriş adresi	Açıklama
Sıfırla	Süervizör	0x00000000	Sistem başlatıldığında meydana gelir.
Veri iptal	İptal	0x00000010	Geçersiz bir bellek adresine erişildiğinde meydana gelir, örneğin bir adres için fiziksel bellek yoksa veya doğru erişim izni yoksa.
FIQ (hızlı kesme)	FIQ	0x0000001C	Harici bir aygit işlemci üzerindeki FIQ pinine uyarı verdiğinde oluşur. Bir kesme, bir FIQ tarafından kesilmediğe kesilemez. FIQ bir veri aktarımının veya kanal sürecini desteklemek üzere tasarlanmıştır ve bu tür uygulamalarla kayıt tasarrufu ihtiyacını ortadan kaldırma için yeterli özel kayıtlara sahiptir, bu nedenle bağlam değiştirme ek yükünü en aza indirir. Hızlı bir kesme kesintiye uğratılamaz.
IRQ (kesme)	IRQ	0x00000018	Harici bir aygit işlemci üzerindeki IRQ pinine uyarı oluşur. Bir FIQ dışında bir kesme kesintiye uğratılamaz.
Prefetch iptal	İptal	0x0000000C	Bir komut alma girişimi bir bellek hatasıyla sonuçlandığında oluşur. Komut, boru hattının yürütme aşamasına girdiğinde istisna ortaya çıkar.
Tanımlanmamış talimatlar	Tanımsız	0x00000004	Talimat setinde olmayan bir talimat boru hattının yürütme aşamasına ulaştığında meydana gelir.
Yazılım kesintisi	Süervizör	0x00000008	Genellikle kullanıcı modu programlarının işletim sistemini çağırmasına izin vermek için kullanılır. Kullanıcı programı, kullanıcının gerçekleştirmek istediği işlevi tanımlayan bir argümanla birlikte bir SWI komutu yürütür.

Böylece rutini tamamlandığında orijinal programa devam edilebilir. İşlemcinin yürütmek üzere olduğu komutun adresi uygun işlemci modunun bağlantı kaydına yerleştirilir. İstisnayı işledikten sonra geri dönmek için, SPSR CPSR'ye ve R14 PC'ye taşınır.

## 14.7 ANAHTAR TERİMLER, GÖZDEN GEÇİRME SORULARI VE PROBLEMLER

### Anahtar Terimler

dal tahmini koşul kodu gecikmeli dal	bayrak talimat döngüsü talimat boru hattı	komut ön getirme program durum sözcüğü (PSW)
---	---	---

## İnceleme Soruları

- 14.1** İşlemci kayıtları hangi genel rolleri yerine getirir?
- 14.2** Kullanıcı tarafından görülebilen kayıtlar genellikle hangi veri kategorilerini destekler?
- 14.3** Koşul kodlarının işlevi nedir?
- 14.4** Program durum sözcüğü nedir?
- 14.5** İki aşamalı bir komut ardışık düzeninin, ardışık düzen kullanılmamasına kıyasla komut çevrim süresini yarıya indirmesi neden olası değildir?
- 14.6** Bir talimat hattının koşullu dallanma talimatlarıyla başa çıkabilmesinin çeşitli yollarını listeleyiniz ve kısaca açıklayınız.
- 14.7** Dal tahmini için geçmiş bitleri nasıl kullanılır?

## Problemler

- 14.1**
  - a. Eğer 8 bitlik bir sözcüğe sahip bir bilgisayarda gerçekleştirilen son işlem, iki işlenenin 00000010 ve 00000011 olduğu bir toplama işlemi olsayı, aşağıdaki bayrakların değeri ne olurdu?
    - Taşıma
    - Sıfır
    - Taşma
    - İşaret
    - Çift Parite
    - Yarım Taşıma
  - b. -1'in (ikiye tümleyen) ve +1'in toplanması için tekrarlayın.
- 14.2** A'nın 11110000 ve B'nin 0010100 A-B işlemi için Problem 14.1'i tekrarlayın.
- 14.3** Bir mikroişlemci 5 GHz hızında saatlenir.
  - a. Bir saat döngüsü ne kadar sürer?
  - b. Üç saat döngüsünden oluşan belirli bir makine komutu türünün süresi nedir?
- 14.4** Bir mikroişlemci, bir bayt dizisini belleğin bir alanından diğerine taşıyabilen bir komut sağlar. Komutun alınması ve ilk kod çözme işlemi 10 saat çevrimi sürer. Bundan sonra, her baytin aktarılması 15 saat çevrimi sürer. Mikroişlemci 10 GHz hızında çalıştırılır.
  - a. Bir dizinin 64 bayt olması durumunda komut döngüsünün uzunluğunu belirleyin.
  - b. Talimat kesintiye uğramıyorsa, bir kesintinin onaylanması için en kötü durum gecikmesi nedir?
  - c. Komutun her bayt aktarımının başında kesilebileceğini varsayıarak (b) bölümünü tekrarlayın.
- 14.5** Intel 8088, 2 aşamalı bir boru hattı oluşturan bir veri yolu arayüz birimi (BIU) ve bir yürütütme biriminden (EU) oluşur. BIU, talimatları 4 baylıklı bir talimat kuyruğuna getirir. BIU ayrıca adres hesaplamalarına katılır, işlenenleri alır ve EU tarafından talep edildiği şekilde sonuçları belleğe yazar. Böyle bir istek yoksa ve veri yolu boşsa, BIU komut kuyruğundaki boş yerleri doldurur. EU bir komutun yürütülmesini tamamladığında, tüm sonuçları BIU'ya aktarır (bellek veya G/C'ye yönelik) ve bir sonraki komuta geçer.
  - a. BIU ve EU tarafından gerçekleştirilen görevlerin yaklaşık eşit zaman aldığına varsayılm. Pipelining 8088'in performansını hangi faktörle artırır? Dallanma talimatlarının etkisini göz arı edin.
  - b. AB'nin BIU'dan iki kat daha uzun sürdüğünü varsayıarak hesaplamayı tekrarlayın.
- 14.6** Bir 8088'in, program atlama olasılığının aşağıdaki gibi olduğu bir programı yürütüğünü varsayı 0.1. Basitlik için, tüm talimatların 2 bayt uzunluğunda olduğunu varsayılm.

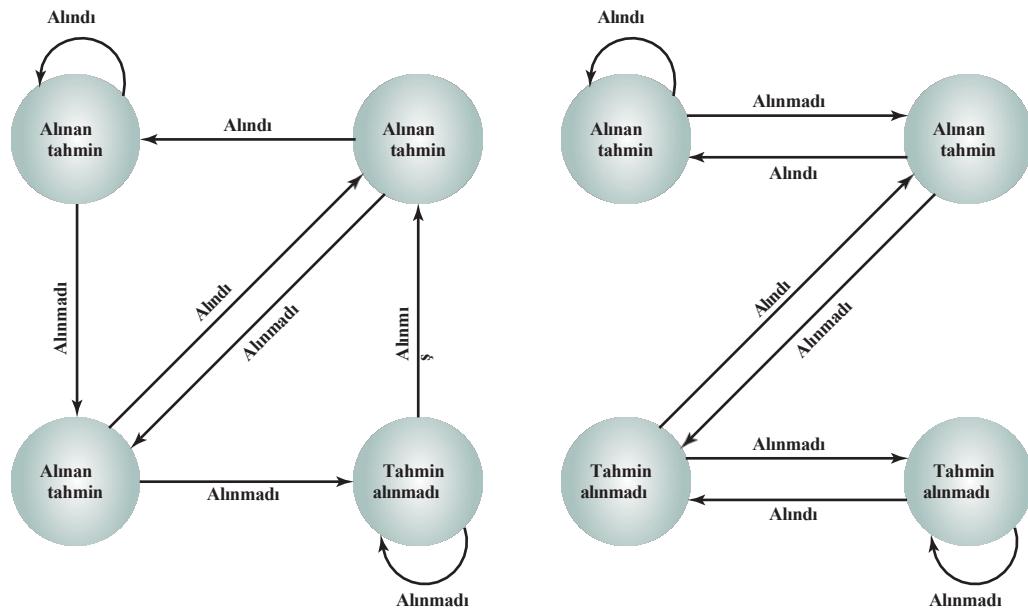
- a. Komut getirme veri yolu döngülerinin ne kadarı boş harcanır?
  - b. Komut kuyruğu 8 bayt uzunluğundaysa tekrarlayın.
- 14.7** Şekil 14.10'daki zamanlama diyagramını göz önünde bulundurun. Sadece iki aşamalı bir boru hattı olduğunu varsayıñ (getirme, yürütme). Dört komut için kaç zaman birimine ihtiyaç duyulduğunu göstermek için diyagramı yeniden .
- 14.8** Dört aşamalı bir boru hattı varsayıñ: komut getirme (FI), komut çözme ve adres hesaplama (DA), işlenen getirme (FO) ve yürütme EX). Üçüncü komutun bir dallanma olduğu ve veri bağımlılığının olmadığı 7 komutluk bir dizi için Şekil 14.10'a benzer bir diyagram çizin.
- 14.9** Bir boru hattı işlemcisi 2,5 GHz saat hızına sahiptir ve 1,5 milyon içeren bir programı yürütür. Boru hattının beş aşaması vardır ve komutlar saat döngüsü başına bir hızda verilir. Dallanma talimatları ve sıra dışı yürütümlerden kaynaklanan cezaları göz ardı edin.
- a. Bölüm 14.4'te kullanılan varsayımların aynısını yaparak, bu işlemcinin bu program için boru hattı olmayan bir işlemciye kıyasla hız artışı nedir?
  - b. Boru hattı işlemcisinin verimi (MIPS cinsinden) nedir?
- 14.10** Boru hattı olmayan bir işlemcinin saat hızı 2,5 GHz ve ortalama CPI (komut başına döngü) 4'tür. İşlemciye yapılan bir yükseltme beş aşamalı bir boru hattı getirmektedir. Ancak, mandal gecikmesi gibi dahili boru hattı gecikmeleri nedeniyle, yeni işlemcinin saat hızının 2 GHz'e düşürülmesi gereklidir.
- a. Tipik bir program için elde edilen hızlanma nedir?
  - b. Her bir işlemci için MIPS oranı nedir?
- 14.11** Talimat boru hattından geçen  $n$  uzunluğunda bir talimat dizisi düşünün.  $P$ , koşullu veya koşulsuz bir dallanma komutuyla karşılaşma olasılığı olsun ve  $q$ , bir dallanma komutu I'nin yürütülmesinin ardışık olmayan bir adres'e atlamaya neden olma olasılığı olsun. Bu tür her atlamanın, I son aşamadan çıktıığında devam eden tüm komut işlemlerini yok ederek boru hattının temizlenmesini gerektirdiğini varsayıñ. Bu olasılıkları dikkate almak için Denklem (14.1) ve (14.2)'yi gözden .
- 14.12** Bir boru hattındaki dallanmalarla başa çıkmak için çoklu akış yaklaşımının bir sınırlaması, ilk dallanma çözülmeden önce ek dallanmalarla karşılaşılacak olmasıdır. İki ek sınırlama veya dezavantaj öneriniz.
- 14.13** Şekil 14.28'deki durum diyagramlarını göz önünde bulundurun.
- a. Her birinin davranışını tanımlayın.
  - b. Bunları Bölüm 14.4'teki dal tahmini durum diyagramı ile karşılaştırın. Dal tahminine yönelik üç yaklaşımın her birinin göreceli değerlerini tartışın.
- 14.14** Motorola 680x0 makineleri, aşağıdaki biçimde sahip olan Azaltma ve Koşula Uygun Dallanma komutunu içerir:

```
DBcc Dn, yer değiştirme
```

Burada cc test edilebilir koşullardan biridir, Dn genel amaçlı bir yazmacıtır ve dis- placement geçerli adrese göre hedef adresi belirtir. Komut aşağıdaki gibi tanımlanabilir:

```
if (cc= False)
    sonra başla
        Dn:= (Dn) -1;
        Dn Z -1 ise PC:= (PC)+ yer değiştirme end else PC:=
            (PC)+ 2;
```

Komut yürütüldüğünde, döngü için sonlandırma koşulunun sağlanıp sağlanmadığını belirlemek için önce koşul test edilir. Eğer öyleyse, hiçbir işlem yapılmaz ve yürütme sıradaki bir sonraki komutla devam eder. Koşul yanlışşabelirtilen veri kaydı azaltılır ve sıfırdan küçük olup olmadığı kontrol edilir. Eğer öyleyse



**Şekil 14.28** İki Dalli Tahmin Durum Diyagramları

sıfırdan küçükse, döngü sonlandırılır ve yürütme sıradaki bir sonraki komutla devam eder. Aksi takdirde, program belirtilen konuma dallanır. Şimdi aşağıdaki assembly dili program parçasını düşünün:

TEKRAR	CMPM.L	(A0)+ , (A1+
	DBNE	D1,
TEKRAR NOP		

A0 ve A1 tarafından adreslenen iki dizgi eşitlik açısından karşılaştırılır; dizgi işaretçileri her başvuruda artırılır. D1 başlangıçta karşılaşılacak uzun kelimelerin (4 bayt) sayısını içerir.

- Kayıtların ilk içerikleri A0= \$00004000, A1= \$00005000 ve D1= \$000000FF'dir (\$ onaltılı gösterimi belirtir). 4000\$ ile 6000\$ arasındaki bellek \$AAAA sözcükleriyle yüklenir. Yukarıdaki program çalıştırılırsa, DBNE döngüsünün kaç kez çalışıldığını ve NOP komutuna ulaşıldığında üç kaydedicinin içeriğini belirtin.
- (a)'yı tekrarlayın, ancak şimdi \$4000 ile \$4FEE arasındaki belleğin \$0000 ile ve \$5000 ile \$6000 arasındaki belleğin \$AAA ile yüklendiğini varsayıın.

**14.15** Koşullu dalın alınmadığını varsayıarak Şekil 14.19c'yi yeniden çizin.

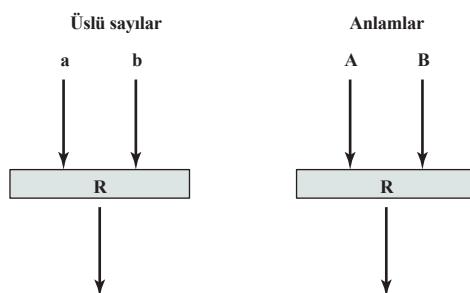
**14.16** Tablo 14.5, [MACD84]'ten çeşitli uygulama sınıfları için dallanma davranışına ilişkin istatistikleri özetlemektedir. Tip 1 dallanma davranışı haricinde, uygulama sınıfları arasında belirgin bir fark yoktur. Bilimsel ortam için şube hedef adresine giden tüm şubelerin oranını belirleyin. Ticari ve sistem ortamları için tekrarlayın.

**14.17** Kayan nokta işlemlerini hızlandırmak için ALU içinde boru hattı uygulanabilir. Kayan noktalı toplama ve çıkarma işlemlerini göz önünde bulundurun. Basitleştirilmiş bir ifadeyle, boru hattı dört aşamadan oluşabilir: (1) Üsleri karşılaştır; (2) Üs seç ve anımları hizala; (3) Anımları topla veya çıkar; (4) Sonuçları normalleştir. Sonuçlar

**Tablo 14.5** Örnek Uygulamalarda Dallanma Davranışı

Branş sınıflarının oluşumu:			
Tip 1: Şube	72.5%	Ticari	Sistemler
Tip 2: Döngü kontrolü	9.8%		
Tip 3: Prosedür çağrıları, dönüş	17.7%		
Tip 1 branş: nereye gider	Bilimsel	Ticari	Sistemler
Koşulsuz-%100 hedefe git	20%	40%	35%
Şartlı-hedefe gitti	43.2%	24.3%	32.5%
Koşullu-hedefe gitmedi (satır içi)	36.8%	35.7%	32.5%
Tip 2 branş (tüm ortamlar)			
Hedefe giden	91%		
Bu satır içi	9%		
Tip 3 şube			
100 hedefe gidin			

borusu, biri üsleri işleyen ve diğerini anımları işleyen iki paralel iş parçacığına sahip olarak düşünülebilir ve bu şekilde başlayabilir:



Bu şekilde, R etiketli kutular geçici sonuçları tutmak için kullanılan bir dizi kaydediciyi ifade eder. Boru hattının yapısını en üst düzeyde gösteren blok diyagramını tamamlayın.

# 15

## BÖLÜM

# AZALTILMIŞ KOMUT KÜMELİ BİLGİSAYARLAR

### 15.1 Komut Yürütme Özellikleri

- İşlemler
- Operandlar
- Prosedür Çağrıları
- Çıkarımlar

### 15.2 Büyük Bir Kayıt Dosyasının Kullanımı

- Windows Küresel
- Değişkenlerini
- Kaydetme
- Önbelleğe Karşı Büyük Kayıt Dosyası

### 15.3 Derleyici Tabanlı Kayıt Optimizasyonu

### 15.4 Azaltılmış Komut Kümesi Mimarisi

- Neden CISC
- Azaltılmış Komut Kümesi Mimarilerinin Özellikleri CISC ve RISC Karakteristikleri

### 15.5 RISC Pipelining

- Düzenli Komutlarla Ardişik Astarlama Ardişik Astarlamayı Optimizasyonu

### 15.6 MIPS R4000

- Komut Kümesi Komut İşlem Hattı

### 15.7 SPARC

- SPARC Kayıt Komut Kümesi Komut Formatı

### 15.8 RISC ve CISC Tartışması

### 15.9 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler

### ÖĞRENİM HEDEFLERİ

Bu bölümü çalıştıktan sonra şunları yapabilmelisiniz:

- RISC yaklaşımının geliştirilmesini motive eden komut yürütme özelliklerini hakkında genel bir araştırma sonuçları sunmak.
- RISC makinelerinin temel özelliklerini özetleyiniz.
- Büyük bir kayıt dosyası kullanmanın tasarım ve performans üzerindeki etkilerini anlamak.
- Performansı artırmak için derleyici tabanlı kayıt optimizasyonunun kullanımını anlamak.
- RISC mimarisinin boru hattı tasarımı ve performansındaki etkilerini tartışınız.
- Bir RISC makinesinde boru hattı optimizasyonuna yönelik temel yaklaşımları listeleyebilir ve açıklayabilir.

Saklı program bilgisayarının 1950 civarında geliştirilmesinden bu yana, bilgisayar organizasyonu ve mimarisi alanlarında oldukça az sayıda gerçek yenilik olmuştur. Aşağıda bilgisayarın doğuştan bu yana gerçekleştiren bazı önemli gelişmeler yer almaktadır:

- **Aile konsepti:** IBM tarafından 1964 yılında System/360 ile tanıtılmış ve kısa bir süre sonra DEC tarafından PDP-8 ile takip edilmiştir. Aile, bir makinenin mimarisini uygulamasından ayırr. Kullanıcıya aynı mimariyi sunan farklı fiyat/performans özelliklerine sahip dizi bilgisayar sunulmaktadır. Fiyat ve performansta farklıklar aynı mimarinin farklı uygulamalarından kaynaklanmaktadır.
- **Mikro programlı kontrol ünitesi:** Wilkes tarafından 1951 yılında önerilmiş ve IBM tarafından 1964 yılında S/360 serisinde tanıtılmıştır. Mikro programlama, kontrol birimini tasarlama ve uygulama görevini kolaylaştırır ve aile konsepti için destek sağlar.
- **Ön bellek:** İlk olarak 1968 yılında IBM S/360 Model 85'te ticari olarak tanıtılmıştır. Bu öğenin bellek hiyerarşisine eklenmesi performansı ölçüde artırır.
- **Pipelining:** Bir makine komutu programının esasen sıralı doğasına paralellik katmanın bir yolu. Örnekler komut ardisıklığı ve vektör işlemedir.
- **Çoklu işlemciler:** Bu kategori bir dizi farklı kuruluşa ve hedefi kapsar.
- **Azaltılmış komut seti bilgisayarı (RISC) mimarisi:** Bu bölümün odak noktası budur.

Ortaya çıktığında, RISC mimarisi işlemci mimarisindeki geleneksel eğilimden dramatik bir sapmaydı. RISC mimarisinin analizi, bilgisayar organizasyonu ve mimarisindeki birçok önemli konuya odaklanılmasını sağlamaktadır.

RISC mimarileri farklı gruplar tarafından şekillerde tanımlanmış ve tasarılanmış olsa da, çoğu tasarım tarafından paylaşılan temel unsurlar şunlardır:

- Çok sayıda genel amaçlı kayıt ve/veya kayıt kullanımını optimize etmek için derleyici teknolojisinin kullanılması.
- Sınırlı ve basit bir komut seti.
- Talimat hattının optimize edilmesine vurgu.

Tablo 15.1 çeşitli RISC ve RISC olmayan sistemleri karşılaştırmaktadır.

Bu bölümde komut setleri ile ilgili bazı sonuçların kısa bir incelemesi ile başlıyoruz ve daha sonra az önce listelenen üç konunun her birini inceliyoruz. Bunu, en iyi belgelenmiş iki RISC tasarımlının tanımlanması takip etmektedir.

## 15.1 KOMUT YÜRÜTME ÖZELLİKLERİ

Bilgisayarlarla ilgili evrimin en görünür biçimlerinden biri programlama dilleridir. Donanımın maliyeti düştükçe, yazılımın göreceli maliyeti artmıştır. Bununla birlikte, kronik programcı sıkıntısı yazılım maliyetlerini mutlak anlamda artırılmıştır. Dolayısıyla, bir sistemin yaşam döngüsündeki en büyük maliyet donanım değil yazılımdır. Maliyete ve rahatsızlığa bir de güvenilmezlik unsuru eklenir: hem sistem hem de uygulama programlarının yıllarca çalışıktan sonra yeni hatalar göstermeye devam etmesi yaygın bir durumdur.

Araştırmacıların ve endüstrinin buna yanıt, giderek daha güçlü ve karmaşık yüksek seviyeli programlama dilleri geliştirmek olmuştur. Bu **üst düzey diller (HLL'ler)**: (1) programcının algoritmaları daha kısa ve öz bir şekilde ifade etmesini sağlar; (2) derleyicinin programın algoritmaları ifade etmesinde önemli olmayan ayrıntılarla ilgilenmesine izin verir; ve (3) genellikle yapılandırılmış programlama ve/veya nesne yönelimli tasarımın kullanımını doğal olarak destekler.

Ne yazık ki, bu çözüm *seman-tik boşluk* olarak bilinen, HLL'erde sağlanan işlemler ile bilgisayar mimarisinde sağlanan işlemler arasındaki fark olarak algılanan bir soruna yol açmıştır. Bu boşluğun belirtilerinin yürütme verimsizliği, aşırı makine programı boyutu ve derleyici karmaşıklığı olduğu iddia edilmektedir. Tasarımcılar bu açığı kapatmayı amaçlayan mimarilerle karşılık vermişlerdir. Temel özellikler arasında büyük komut setleri, düzinelere adresleme modu ve donanımda uygulanan çeşitli HLL ifadeleri yer almaktadır. VAX üzerindeki CASE makine talimatı buna bir . Bu tür karmaşık komut setleri aşağıdakileri amaçlamaktadır:

- Derleyici yazarının görevini kolaylaştırır.
- Karmaşık işlem dizileri mikro kodda uygulanabildiği için yürütme verimliliğini artırır.
- Daha da karmaşık ve sofistike HLL'ler için destek sağlayın.

Bu arada, HLL programlarından üretilen makine talimatlarının özelliklerini ve yürütülme şekillerini belirlemek için yıllar boyunca bir dizi çalışma yapılmıştır. Bu çalışmaların sonuçları bazı araştırmacılara şu konulara bakmaları için ilham vermiştir

## 538 BÖLÜM 15 / AZALTILMIŞ KOMUT SETLİ BİLGİSAYARLAR

**Tablo 15.1** Bazı CISC, RISC ve Superscalar İşlemcilerin Özellikleri

	Karmaşık Komut Seti (CISC) Bilgisayar			Azaltılmış Komut Seti (RISC) Bilgisayar	
Karakteristik	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000
Geliştirildiği yıl	1973	1978	1989	1987	1991
Talimat sayısı	208	303	235	69	94
Komut boyutu (bayt)	2-6	2-57	1-11	4	4
Adresleme modları	4	22	11	1	1
Genel amaçlı kayıtların sayısı	16	16	8	40-520	32
Kontrol belleği boyutu (kbits)	420	480	246	-	-
Önbellek boyutu (kB)	64	64	8	32	128

	Superskalar		
Karakteristik	PowerPC	Ultra SPARC	MIPS R10000
Geliştirildiği yıl	1993	1996	1996
Talimat sayısı	225		
Komut boyutu (bayt)	4	4	4
Adresleme modları	2	1	1
Genel amaçlı kayıtların sayısı	32	40-520	32
Kontrol belleği boyutu (kbits)	-	-	-
Önbellek boyutu (kB)	16-32	32	64

Farklı bir yaklaşım için: yani, HLL'yi destekleyen mimariyi daha karmaşık hale getirmek yerine daha basit hale getirmek.

RISC savunucularının mantık çizgisini anlamak için, komut yürütme özelliklerinin kısa bir incelemesi ile başlayacağız. İlgilinen hesaplama yönleri aşağıdaki gibidir:

- **Gerçekleştirilen işlemler:** Bunlar, işlemci tarafından gerçekleştirilecek işlevleri ve işlemcinin bellekle etkileşimini belirler.
- **Kullanılan operandalar:** İşlenenlerin türleri ve kullanım sıklıkları, bunları saklamak için bellek organizasyonunu ve bunlara erişmek için adresleme modlarını belirler.
- **Yürütme sıralaması:** Bu, kontrol ve boru hattı organizasyonunu belirler.

Bu bölümün geri kalanında, yüksek seviyeli dil programları üzerine yapılan bir dizi çalışmanın sonuçlarını özetliyoruz. Tüm sonuçlar dinamik ölçümlere dayanmaktadır. , ölçümler program çalıştırılarak ve bazı özelliklerin kaç kez ortaya çıktıığı ya da belirli bir özelliğin kaç kez doğru olduğu sayilarak toplanır. Buna karşılık, statik ölçümler bu sayımları yalnızca bir programın kaynak metni üzerinde gerçekleştirir. Performans hakkında faydalı bilgi vermezler, çünkü her bir ifadenin yürütülme sayısına göre ağırlıklandırılmazlar.

## Operasyonlar

HBÖ programlarının davranışını analiz etmek için çeşitli çalışmalar yapılmıştır. Bölüm 4'te tartışılan Tablo 4.7, bir dizi çalışmadan elde edilen temel sonuçları içermektedir. Bu dil ve uygulama karışımının sonuçlarında oldukça iyi bir uyum vardır. Atama ifadelerinin baskın olması, verilerin basitçe taşınmasının yüksek öneme sahip olduğunu göstermektedir. Ayrıca koşullu ifadeler (IF, LOOP) de ağırlıktadır. Bu ifadeler makine dilinde bir çeşit karşılaştırma ve dallanma talimatı ile uygulanmaktadır. Bu da komut setinin sıra kontrol mekanizmasının önemini olduğunu göstermektedir.

Bu sonuçlar makine komut seti tasarımcısı için öğreticidir, hangi tür ifadelerin en sık ortaya çıktığını ve bu nedenle "en uygun" şekilde desteklenmesi gerektiğini gösterir. Ancak bu sonuçlar tipik bir programın yürütülmesinde en çok hangi deyimlerin kullanıldığını ortaya koymaz. , şu soruya cevap vermek istiyoruz: Derlenmiş bir makine dili programı göz önüne alındığında, kaynak dildeki hangi ifadeler en çok makine dili talimatının yürütülmesine neden olur ve bu talimatların yürütülme süresi nedir?

Bu temel olguya ulaşmak için, Ek 4A'da açıklanan Patterson programları [PATT82a] VAX, PDP-11 ve Motorola 68000 üzerinde derlenerek her bir deyim türü için ortalama makine komutu ve bellek referansı sayısı belirlenmiştir. Tablo 15.2'deki ikinci ve üçüncü sütunlar çeşitli HLL deyimlerinin çeşitli programlarda ortaya çıkma sıklığını göstermektedir; veriler deyimlerin kaynak kodda ortaya çıkma sayısından ziyade çalışan programlarda ortaya çıkma sayıları gözlemlenerek elde edilmiştir. Dolayısıyla bu metrikler dinamik davranışını yakalamaktadır. Dördüncü ve beşinci sütunlardaki (makine komutu ağırlıklı) verileri elde etmek için, ikinci ve üçüncü sütunlardaki her bir değer derleyici tarafından üretilen makine komutlarının sayısı ile çarpılır. Bu sonuçlar daha sonra normalleştirilir, böylece dördüncü ve beşinci sütunlar HLL deyimi başına makine komutu sayısıyla ağırlıklandırılmış göreceli oluşma sıklığını gösterir. Benzer şekilde, altıncı ve yedinci sütunlar her bir deyim türünün ortaya çıkma sıklığı ile her bir deyimin neden olduğu bellek referanslarının göreceli sayısının çarpılmasıyla elde edilir. Dördüncü sütundan yedinci sütuna kadar olan sütunlardaki veriler, çeşitli deyim türlerini yürütmek için harcanan gerçek sürenin vekil ölçümlerini göstermektedir. Sonuçlar, tipik HLL programlarında en çok zaman alan işlemin yordam çağrısı/geri dönüş olduğunu göstermektedir.

Okuyucu Tablo 15.2'nin önemi konusunda açık olmalıdır. Bu tablo, aşağıdaki durumlarda bir HLL'deki çeşitli ifade türlerinin göreceli performans etkisini göstermektedir

**Tablo 15.2** HLL İşlemlerinin Ağırlıklı Göreli Dinamik Frekansı [PATT82a]

	Dinamik Oluşum		Makine-Talimat Ağırlıkları		Bellek-Referans Ağırlıkları	
	Pascal	C	Pascal	C	Pascal	C
GÖREVLENDİR	45%	38%	13%	13%	14%	15%
DÖNGÜ	5%	3%	42%	32%	33%	26%
ÇAĞRI	15%	12%	31%	33%	44%	45%
EĞER	29%	43%	11%	21%	7%	13%
GOTO	-	3%	-	-	-	-
DİĞER	6%	1%	3%	1%	2%	1%

HLL tipik bir çağdaş komut seti mimarisi için derlenmiştir. Başka bir mimari muhtemelen farklı sonuçlar üretebilir. Ancak bu çalışma, çağdaş **karmaşık komut seti bilgisayar (CISC)** temsil eden sonuçlar üretmektedir. Böylece, HLL'leri desteklemek için daha verimli yollar arayanlara rehberlik edebilirler.

## Operandalar

Bu konunun önemine rağmen, işlenen türlerinin oluşumu üzerine daha az çalışma yapılmıştır. Önemli olan birkaç husus vardır.

Daha önce atıfta bulunan Patterson çalışması [PATT82a] değişken sınıflarının dinamik oluşum sıklığını da incelemiştir (Tablo 15.3). Pascal ve C programları arasında tutarlı olan sonuçlar, referansların çoğunun basit skaler değişkenlere olduğunu göstermektedir. Ayrıca, skalerlerin %80'inden fazlası yerel (prosedür için). Buna ek olarak, bir diziyeye ya da yapıya yapılan her referans, yine genellikle yerel bir skaler olan bir indekse ya da işaretçiye referans gerektirir. Dolayısıyla, skalerlere yapılan referansların bir üstünlüğü vardır ve bunlar yüksek oranda yerelleştirilmiştir.

Patterson çalışması, temel mimariden bağımsız olarak HLL programlarının dinamik davranışını incelemiştir. Daha önce de tartışıldığı gibi, program davranışını daha derinlemesine incelemek için gerçek mimarilerle uğraşmak gereklidir. Bir çalışma, [LUND77], DEC-10 komutlarını dinamik olarak incelemiş ve her komutun ortalamaya olarak bellekte 0,5 işlenene ve 1,4 reg-lers'e referans verdigini bulmuştur. Benzer sonuçlar [HUCK83]'de S/370, PDP-11 ve VAX üzerindeki C, Pascal ve FORTRAN programları için rapor edilmiştir. Elbette, bu rakamlar büyük ölçüde mimariye ve derleyiciye, ancak işlenen erişiminin sıklığını göstermektedir.

**Tablo 15.3** İşlenenlerin Dinamik Yüzdesi

	Pascal	C	Ortalama
Tamsayı sabiti	16%	23%	20%
Skaler değişken	58%	53%	55%
Dizi/Yapı	26%	24%	25%

Bu son çalışmalar, hızlı operand erişimine uygun bir mimarının önemini ortaya koymaktadır, çünkü bu işlem çok sık yapılmaktadır. Patterson çalışması, optimizasyon için başlıca adayın yerel skaler değişkenleri saklama ve bunlara erişme mekanizması olduğunu öne sürmektedir.

### Prosedür Çağrıları

Yordam çağrılarının ve geri dönüşlerin HLL önemli bir yönü olduğunu gördük. Kanıtlar (Tablo 15.2) bunların derlenmiş HLL programlarında en çok zaman alan işlemler olduğunu göstermektedir. Bu nedenle, bu işlemleri verimli bir şekilde uygulamanın yollarını düşünmek karlı olacaktır. İki husus önemlidir: bir prosedürün ilgilendiği parametre ve değişkenlerin sayısı ve iç içe geçme derinliği.

Tanenbaum'un çalışması [TANE78] dinamik olarak çağrılan prosedürlerin %98'inin altıdan az argüman geçirdiğini ve bunların %92'sinin altıdan az yerel skaler değişken kullandığını bulmuştur. Benzer sonuçlar Tablo 15.4'te gösterildiği gibi Berkeley RISC ekibi [KATE83] tarafından da rapor edilmiştir. Bu sonuçlar, prosedür aktivasyonu başına gereken kelime sayısının büyük olmadığını göstermektedir. Daha önce rapor edilen çalışmalar, operand referanslarının yüksek bir oranının yerel skaler değişkenlere olduğunu göstermiştir. Bu çalışmalar, bu referansların aslında nispeten az sayıda değişkenle sınırlı olduğunu göstermektedir.

Aynı Berkeley grubu, HLL programlarındaki yordam çağrıları ve geri dönüşleri modelini de incelemiştir. Uzun ve kesintisiz bir yordam çağrısı dizisinin ardından buna karşılık gelen bir geri dönüş dizisinin nadir olduğunu bulmuşlardır. Bunun yerine, bir programın oldukça dar bir yordam çağrıma derinliği penceresiyle sınırlı kaldığını bulmuşlardır. Bu durum, Bölüm 4'te tartışılan Şekil 4.21'de gösterilmektedir. Bu sonuçlar, operand referanslarının oldukça yerel olduğu sonucunu güçlendirmektedir.

### Çkarımlar

Bir dizi grup, az önce bildirilenler gibi sonuçları incelemiş ve komut seti mimarisini HLL'lere yakın hale getirme girişiminin en etkili tasarım stratejisi olmadığı sonucuna varmıştır. Bunun yerine, HLL'ler en iyi şekilde tipik HLL programlarının en çok zaman alan özelliklerinin performansını optimize ederek desteklenebilir.

**Tablo 15.4** Yordam Bağımsız Değişkenleri ve Yerel Skaler Değişkenler

İle Yürüttülen Prosedür Çağrılarının Yüzdesi	Derleyici, Yorumlayıcı ve Dizgi	Sayısal Olmayan Küçük Programlar
> 3 argüman	0-7%	0-5%
> 5 argüman	0-3%	0%
> 8 kelime argüman ve yerel skaler	1-20%	0-6%
> 12 kelimelik argümanlar ve yerel skaler	1-6%	0-3%

Bir dizi araştırmacının çalışmalardan yola çıkarak, RISC mimarilerini genel olarak karakterize üç unsur ortaya çıkmaktadır. Birincisi, çok sayıda yazmaç kullanmak ya da yazmaç kullanımını optimize etmek için bir derleyici kullanmak. Bu, operand referansını optimize etmeyi amaçlamaktadır. Az önce tartışılan çalışmalar, HLL deyimi başına birkaç referans olduğunu ve yüksek oranda taşıma (atama) deyimi olduğunu göstermektedir. Bu, skaler referansların yerelliği ve baskınlığı ile birleştiğinde, daha fazla yazmaç referansı pahasına bellek referanslarını performansın artırılabilceğini göstermektedir. Bu referansların yerelliği nedeniyle, genisletilmiş bir kayıt seti pratik görülmektedir.

İkinci olarak, komut boru hatlarının tasarımasına dikkat edilmesi gerekmektedir. Koşullu dallanma ve yordam çağrıları talimatlarının yüksek oranı nedeniyle, basit bir talimat boru hattı verimsiz olacaktır. Bu durum, öncelenen ancak asla yürütülmeyen talimatların oranının yüksek olması şeklinde kendini gösterir.

Son olarak, yüksek performanslı ilkellerden oluşan bir komut seti belirlenmiştir. Komutlar öngörülebilir maliyetlere sahip olmalı (yürütme süresi, kod boyutu ve giderek artan enerji kaybı ile ölçülür) ve yüksek performanslı bir uygulama ile tutarlı olmalıdır (öngörülebilir yürütme süresi maliyeti ile uyumludur).

## 15.2 BÜYÜK BİR KAYIT DOSYASININ KULLANIMI

Bölüm 15.1'de özetlenen sonuçlar, operandlara hızlı erişimin arzu edilebilirliğine işaret etmektedir. HLL programlarında büyük oranda atama ifadeleri olduğunu ve bunların çoğunun basit A  $\leftarrow$  B şeklinde gördük. Bu sonuçları, erişimlerin çoğunun yerel skalerlere olduğu gerçeğiyle birleştirirsek, yazmaç depolamaya büyük ölçüde güvenildiği ortaya çıkar.

Kayıt depolama alanının belirtilmesinin nedeni, hem ana bellekten hem de önbellekten daha hızlı olan mevcut en hızlı depolama aygıtı olmasıdır. Yazmaç dosyası fiziksel olarak küçüktür, ALU ve kontrol birimi ile aynı yonga üzerindedir ve önbellek ve bellek adreslerinden çok daha kısa adresler kullanır. Bu nedenle, en sık erişilen işlenenlerin yazmaçlarda tutulmasını sağlayacak ve yazmaç-bellek işlemlerini enaza bir stratejiye ihtiyaç vardır.

Biri yazılıma diğer de donanıma dayalı iki temel yaklaşım mümkündür. Yazılım yaklaşımı, register kullanımını maksimize etmek için derleyiciye güvenmektir. Derleyici, belirli bir zaman diliminde en çok kullanılacak değişkenlere yazmaç atamaya çalışacaktır. Bu yaklaşım sofistik program analiz algoritmalarının kullanımını gerektirir. Donanım yaklaşımı basitçe daha fazla yazmaç kullanmaktadır, böylece daha fazla değişken daha uzun süreler boyunca yazmaçlarda tutulabilir.

Bu bölümde donanım yaklaşımını tartışacağız. Bu yaklaşım Berkeley RISC grubu tarafından öncülük edilmiştir [PATT82a]; ilk ticari RISC ürünü olan Pyramid'de kullanılmıştır [RAGA83]; ve şu anda popüler **SPARC** mimarisinde kullanılmaktadır.

## Pencereleri Kaydedin

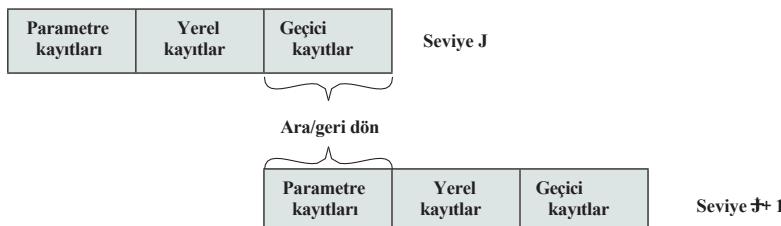
Görünüşe göre, geniş bir yazmaç kümesinin kullanılması belleğe erişim ihtiyacını azaltmalıdır. Tasarım görevi, yazmaçları bu hedefi gerçekleştirecek şekilde düzenlemektedir.

Çoğu operand referansı yerel skalerlere yapıldığından, bariz yaklaşım bunları global değişkenler için ayrılmış belki birkaç register ile registerlarda saklamaktır. Sorun, *yerel* tanımının her yordam çağrısında ve geri dönüste, yani sık sık gerçekleşen işlemlerde değişmesidir. Her çağrıda, yerel değişkenler belleğe kaydedilmelidir, böylece yazmaçlar çağrılan yordam tarafından yeniden kullanılabilir. Ayrıca, parametreler de aktarılmalıdır. Geri dönüste, çağrıran prosedürün değişkenleri geri yüklenmeli (kayıtlara geri yüklenmeli) ve sonuçlar çağrıran prosedüre geri aktarılmalıdır.

**Cözüm**, Bölüm 15.1'de bildirilen diğer iki sonuca dayanmaktadır. Birincisi, tipik bir prosedür sadece birkaç parametre ve yerel değişken kullanır (Tablo 15.4). İkincisi, prosedür aktivasyonunun derinliği nispeten dar bir aralıkta dalgalanmaktadır (Şekil 4.21). Bu özelliklerden yararlanmak için, her biri farklı bir yordama atanmış birden fazla küçük kayıt kümesi kullanılır. Bir yordam çağrısı, işlemciyi, yazmaçları belleğe kaydetmek yerine, farklı bir sabit boyutlu yazmaç penceresini kullanması için otomatik olarak değiştirir. Bitişik prosedürler için pencereler, parametre geçişine izin vermek için üst üste bindirilir.

Bu kavram Şekil 15.1'de gösterilmiştir. Herhangi bir zamanda, yalnızca bir kayıt penceresi görünür ve tek kayıt kümesiyim gibi adreslenebilir (örneğin, 0'dan  $N - 1$ 'e kadar adresler). Pencere üç sabit boyutlu alana bölünmüştür. Parametre yazmaçları, geçerli yordamı çağrıran yordamdan aşağı aktarılan parametreleri tutar ve yukarı aktarılacak sonuçları tutar. Yerel kayıtlar, derleyici tarafından atanın yerel değişkenler için kullanılır. Geçici kayıtlar, bir sonraki alt seviye (mevcut tarafından çağrılan prosedür) ile parametre ve sonuç alışverişi yapmak için kullanılır. Bir seviyedeki geçici kayıtlar fiziksel olarak bir sonraki alt seviyedeki parametre kayıtlarıyla aynıdır. Bu örtüşme, verilerin gerçek hareketi olmadan parametrelerin aktarılmasına izin verir. Örtüşme dışında, iki farklı seviyedeki yazmaçların fiziksel olarak olduğunu  $J$  seviyesindeki parametre ve yerel kayıtlar,  $J$  seviyesindeki yerel ve geçici kayıtlardan ayırdır+ 1.

Her türlü olası çağrı ve geri dönüş modelini ele almak için, **kayıt pencerelerinin** sayısının sınırsız olması gereklidir. Bunun yerine, kayıt pencereleri en son birkaç prosedür aktivasyonunu tutmak için kullanılabilir. Daha eski aktivasyonlar kaydedilmelidir.

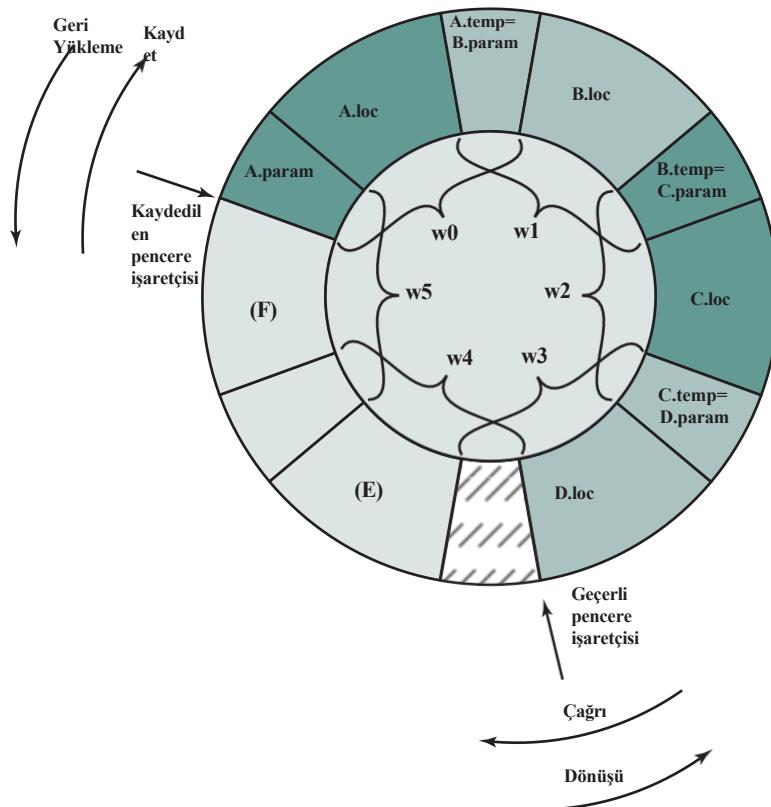


**Şekil 15.1** Örtünen Kayıt Pencereleri

ve daha sonra yuvalama derinliği azaldığında geri yüklenir. Böylece, kayıt dosyasının gerçek organizasyonu üst üste binen pencerelerden oluşan dairesel bir tampon şeklindedir. Bu yaklaşımın iki önemli örneği Bölüm 15.7'de açıklanan Sun'un SPARC mimarisini ve Intel'in Itanium işlemcisinde kullanılan IA-64 mimarisidir.

Dairesel organizasyon Şekil 15.2'de altı pencereden oluşan dairesel bir tamponu göstermektedir. Tampon, D prosedürü aktifken 4 derinliğe kadar doldurulur (A, B'yi; B, C'yi; C, D'yi çağrıır). Geçerli pencere işaretçisi (CWP) o anda etkin olan yordamın penceresine işaret eder. Bir makine komutu tarafından yapılan kayıt referansları, gerçek fiziksel kaydı belirlemek için bu işaretçi tarafından kaydırılır. Kayıtlı pencere işaretçisi (SWP) bellekte en son kaydedilen pencereyi tanımlar. Eğer D prosedürü şimdî E prosedürüne çağrırsa, E için argümanlar D'nin geçici kayıtlarına yerleştirilir (w3 ve w4 arasındaki örtüşme) ve CWP bir pencere ilerletilir.

E yordamı daha sonra F yordamına bir çağrı yaparsa, çağrı mevcut durumıyla yapılamaz. Bunun nedeni F'nin penceresinin A'nın çakışmasıdır. Eğer F bir çağrıya hazırlık olarak kendi geçici kayıtlarını yüklemeye başlarsa, A'nın parametre kayıtlarının üzerine yazacaktır (A.in). Böylece, CWP SWP'ye eşit olacak şekilde artırıldığında (modulo 6), bir kesme oluşur ve A'nın penceresi kaydedilir. Sadece



**Şekil 15.2** Örtüsen Pencerelerin Dairesel Tampon Organizasyonu

ilk iki bölümün (A.in ve A.loc) kaydedilmesi gereklidir. Daha sonra SWP artırılır ve F çağrıları devam eder. Dönüşlerde de benzer bir kesme meydana gelebilir. Örneğin, F'nin etkinleştirilmesinin ardından, B A'ya döndüğünde, CWP azalır ve SWP'ye eşit olur. Bu, A'nın penceresinin geri yüklenmesiyle sonuçlanan bir kesmeye neden olur.

Yukarıda anlatılanlardan,  $N$  pencereyi bir kayıt dosyasının sadece  $N - 1$  prosedür aktivasyonunu tutabileceğini söyleyebilir.  $N$  değerinin büyük olması gerekmektedir. Ek 4A'da belirtildiği gibi, bir çalışma [TAMI83] 8 pencere ile çağrıların veya geri dönüşlerin yalnızca %1'inde bir kaydetme veya geri yükleme gerektiğini bulmuştur. Berkeley RISC bilgisayarları her biri 16 kaydediciden oluşan 8 pencere kullanmaktadır. Pyramid bilgisayarı her biri 32 kayıtta oluşan 16 pencere kullanmaktadır.

### Global Değişkenler

Az önce açıklanan pencere şeması, yerel skaler değişkenleri kayıtlarda saklamak için verimli bir organizasyon sağlar. Ancak bu şema, birden fazla prosedür tarafından erişilen global değişkenlerin saklanması ihtiyacını karşılamaz. İki seçenek kendini göstermektedir. İlk olarak, bir HLL'de global olarak bildirilen değişkenlere derleyici tarafından bellek konumlarını atanabilir ve bu değişkenlere referans veren tüm makine komutları bellek referanslı işlenenleri kullanacaktır. Bu, hem donanım hem de yazılım (derleyici) basittir. Ancak, sık erişilen global değişkenler için bu şema verimsizdir.

Bir alternatif de iGlemciye bir dizi global yazmaç eklemektir. Bu kayıtların sayısı sabit olacak ve tüm prosedürler tarafından kullanılabilir. Birleştirilmiş bir numaralandırma şeması komut formatını basitleştirmek için kullanılabilir. Örneğin, 0'dan 7'ye kadar olan kayıtlara yapılan atıflar bensiz全球 kayıtlara atıfta bulunabilir ve 8'den 31'e kadar olan kayıtlara yapılan atıflar mevcut penceredeki fiziksel kayıtlara atıfta bulunmak için kaydırılabilir. Kayıt terimi adreslemesindeki bölümmayı karşılamak için artan bir donanım yükü vardır. Buna ek olarak, bağlayıcı hangi global değişkenlerin kayıtlara atanması gereğine karar vermelidir.

### Önbelleğe Karşı Büyük Kayıt Dosyası

Penceler halinde düzenlenen kayıt dosyası, en yoğun şekilde kullanılması muhtemel tüm değişkenlerin bir alt kümesini tutmak için küçük, hızlı bir tampon görevi görür. Bu açıdan, kayıt dosyası çok daha hızlı bir bellek olmasına rağmen bir ön bellek gibi davranıştır. Bu nedenle, bir önbellek ve küçük bir geleneksel kayıt dosyası kullanmanın daha basit ve daha iyi olup olmayacağı sorusu ortaya çıkmaktadır.

Tablo 15.5 iki yaklaşımın özelliklerini karşılaştırmaktadır. Pencere tabanlı kayıt dosyası, en son  $N - 1$  prosedür aktivasyonunun tüm yerel skaler değişkenlerini (nadir pencere taşması durumu hariç) tutar. Önbellek, son kullanılan skaler değişkenlerin bir seçkisini tutar. Kayıt dosyası zaman kazandırmalıdır, çünkü tüm yerel skaler değişkenler korunur. Öte yandan önbellek, duruma dinamik olarak tepki verdiği için alanın daha verimli kullanılmasını sağlayabilir. Dahası, önbellekler genellikle talimatlar ve diğer veri türleri de dahil olmak üzere tüm bellek referanslarını aynı şekilde ele alır. Dolayısıyla, bu diğer alanlardaki tasarıflar bir kayıt dosyası yerine bir önbellek ile mümkündür.

**Tablo 15.5** Büyük Kayıt-Dosya ve Önbellek Kuruluşlarının Özellikleri

Büyük Kayıt Dosyası	Önbellek
Tüm yerel skalerler	Son zamanlarda kullanılan yerel skalerler
Bireysel değişkenler	Bellek blokları
Derleyici tarafından atanın global değişkenler	Son kullanılan global değişkenler
Prosedür yuvalama derinliğine göre Kaydet/Geri Yükle	Önbellek değiştirme algoritmasına dayalı Kaydet/Geri Yükle
Kayıt adresleme	Bellek adresleme
Tek döngüde adreslenen ve erişilen birden fazla işlenen	Döngü başına adreslenen ve erişilen bir işlenen

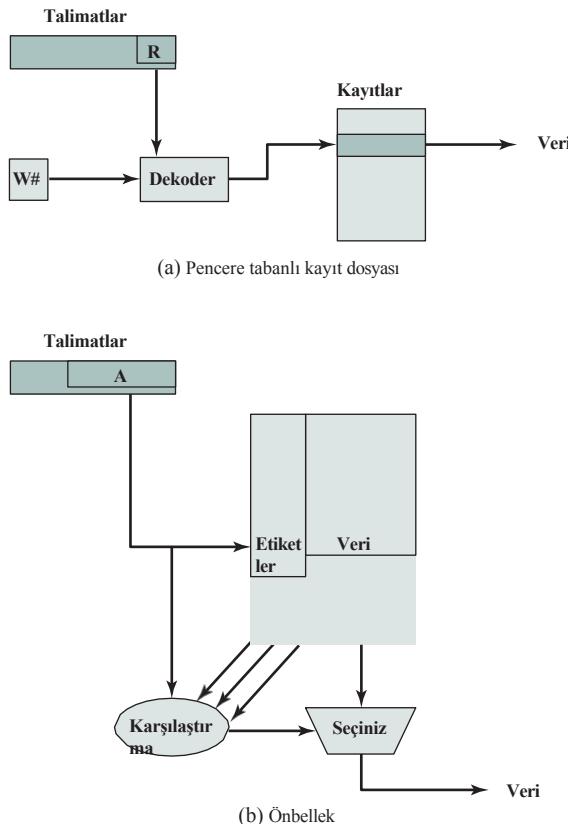
Bir kayıt dosyası alanı verimsiz kullanabilir, çünkü tüm prosedürler kendilerine ayrılan pencere alanının tamamına ihtiyaç duymayacaktır. Öte yandan, önbellek başka bir tür verimsizlikten muzdariptir: Veriler önbellege bloklar halinde okunur. Kayıt dosyası yalnızca kullanımında olan değişkenleri içerirken, önbellek bir kısmı ya da çoğu kullanılmayacak olan bir veri bloğunu okur.

Önbellek, yerel değişkenlerin yanı sıra global değişkenleri de işleyebilir. Genellikle çok sayıda global skaler vardır, ancak bunlardan yalnızca birkaç yoğun olarak kullanılır [KATE83]. Bir önbellek bu değişkenleri dinamik olarak keşfedecek ve tutacaktır. Pencere tabanlı kayıt dosyası global kayıtlarla desteklenirse, o da bazı global tutabilir. Ancak, program modülleri ayrı ayrı, derleyicinin yazmaçlara global değerler ataması değildir; bu görevi bağlayıcı gerçekleştirmelidir.

Kayıt dosyası ile, kayıtlar ve bellek arasındaki veri hareketi prosedür yuvalama derinliği tarafından belirlenir. Bu derinlik genellikle bir aralıkta dalgalandığından, bellek kullanımı nispeten seyrektr. Önbellek çoğu küçük bir küme boyutu ile ilişkilendirilebilir. Bu nedenle, diğer veri veya talimatların önbellette yer almak için rekabet etme tehlikesi vardır.

Simdiye kadarki tartışmalara dayanarak, büyük pencere tabanlı kayıt dosyası ile önbellek arasındaki seçim net değildir. Bununla birlikte, kayıt yaklaşımının açıkça üstün olduğu ve önbellek tabanlı bir sistemin fark edilir derecede daha yavaş olacağını gösteren bir özellik vardır. Bu fark, iki yaklaşım tarafından deneyimlenen adresleme ek yükü miktarında ortaya çıkmaktadır.

**Şekil 15.3** farklı göstermektedir. Pencere tabanlı bir kayıt dosyasında yerel bir skale referans vermek için bir "sanal" kayıt numarası ve bir pencere numarası kullanılır. Bunlar fiziksel kayıtlardan birini seçmek için nispeten basit bir kod çözümünden geçebilir. Önbelletteki bir bellek konumuna başvurmak için tam genişlikte bir bellek adresi oluşturulmalıdır. Bu işlemin karmaşıklığı adresleme moduna bağlıdır. Küme ilişkilendirmeli önbellette, adresin bir kısmı küme boyutuna eşit sayıda kelime ve etiketi okumak için kullanılır. Adresin diğer bir kısmı etiketlerle karşılaşılır ve okunan kelimelerden biri seçilir. Önbellek kayıt dosyası kadar hızlı olsa bile erişim süresinin oldukça uzun olacağı açıkları. Dolayısıyla, performans açısından bakıldığında, pencere tabanlı kayıt dosyası yerel skalerler için daha üstündür. Yalnızca komutlar için bir önbellek eklenecek daha fazla performans artışı sağlanabilir.



Şekil 15.3 Bir Skaler Referanslama

### 15.3 DERLEYİCİ TABANLI KAYIT OPTİMİZASYONU

Şimdi hedef RISC makinesinde yalnızca az sayıda (örneğin 16-32) yazmaç bulunuşunu varsayıyalım. Bu durumda, optimize edilmiş yazmaç kullanımının sorumluluğundadır. Yüksek seviyeli bir dilde yazılan bir programda elbette yazmaçlara açık referanslar yoktur (C dilindeki anahtar kelime register'a rağmen). Bunun yerine, program büyükliklerine sembolik olarak atıfta bulunulur. Derleyicinin amacı, mümkün olduğunda çok sayıda hesaplama için işlenenleri ana bellek yerine yazmaçlarda tutmak ve yükleme-depolama işlemlerini en aza indirmektir.

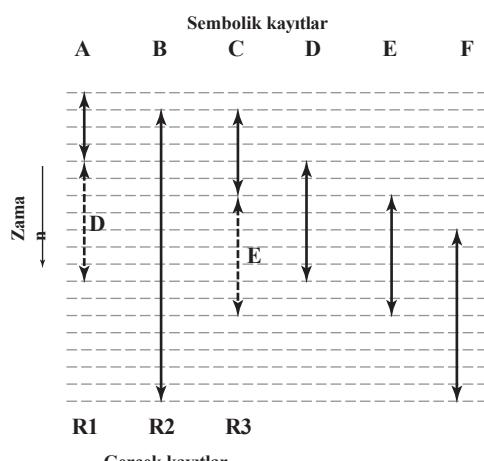
Genel olarak benimsenen yaklaşım aşağıdaki gibidir. Bir yazmaça bulunmaya aday olan her program büyükliği sembolik ya da sanal bir yazmaca atanır. Derleyici daha sonra sınırsız sayıda sembolik yazmacı sabit sayıda gerçek içine eşler. Kullanımları çakışmayan sembolik yazmaçlar aynı gerçek yazmacı paylaşabilir. Programın belirli bir bölümünde, gerçek yazmaçlardan daha fazla sayıda büyülükle uğraşılması gerekiyorsa, bazı büyülükler bellek konumlarına atanır. Load-and-store komutları, hesaplama işlemleri için nicelikleri geçici olarak yazmaçlara yerleştirmek için kullanılır.

Optimizasyon görevinin özü, programın herhangi bir noktasında yazmaçlara hangi büyülüklüklerin atanacağına karar vermektedir. RISC derleyicilerde en yaygın olarak kullanılan teknik, topoloji disiplininden ödünç alınmış bir teknik olan grafik renklendirme olarak bilinir [CHAI82, CHOW86, COUT86, CHOW90].

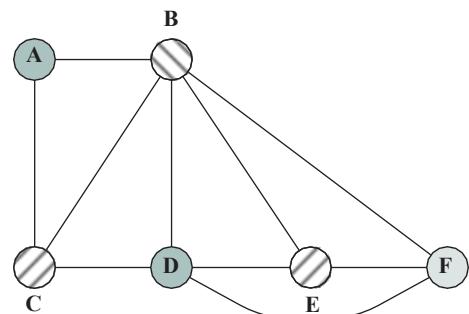
Çizge renklendirme problemi şudur. Düğümler ve kenarlardan oluşan bir grafik verildiğinde, bitişik düğümler farklı renklere sahip olacak şekilde düğümlere renkler atayın ve bunu farklı renklerin sayısını en azı indirecek şekilde yapın. Bu problem derleyici problemine aşağıdaki şekilde uyarlanmıştır. İlk olarak, program bir kayıt girişim grafiği oluşturmak için analize edilir. Grafiğin düğümleri sembolik yazmaçlardır. Eğer iki sembolik yazmaç aynı program parçası sırasında "canlı" ise, o zaman paraziti göstermek için bir kenar ile birləşterilirler. Daha sonra grafiği  $n$  renklendirmeye çalışılır, burada  $n$  sayısıdır. Aynı rengi paylaşan düğümler aynı yazmaca atanabilir. Bu işlem tam olarak başarılı olmazsa, renklendirilemeyen düğümler belleğe yerleştirilmeli ve ihtiyaç duyulduğunda etkilenen miktarlar için yer açmak için yüklemeler ve kullanılmalıdır.

Şekil 15.4 sürecin basit bir örneğidir. Üç gerçek yazmaca derlenecek altı sembolik yazmaca içeren bir program varsayılmıştır. Şekil 15.4a, her bir sembolik aktif kullanımının zaman sırasını göstermektedir. Kesikli yatay çizgiler ardışık komut yürütütmelerini göstermektedir. Şekil 15.4b yazmaç etkileşim grafiğini göstermektedir (renkler yerine gölgelendirme ve çizgiler kullanılmıştır). Üç renk ile olası bir renklendirme gösterilmiştir. Sembolik yazmaçlar A ve D birbirine karışmadığından, derleme bunların her ikisini de fiziksel yazmaç R1'e atayabilir. Benzer şekilde, sembolik yazmaçlar C ve E, yazmaç R3'e atanabilir. Bir sembolik yazmaç, F, renksiz bırakılmıştır ve depolamalar kullanılarak ele alınmalıdır.

Genel olarak, geniş bir yazmaç kümesinin kullanımı ile derleyici tabanlı yazmaç optimizasyonu arasında bir denge vardır. Örneğin, [BRAD91a] bir çalışma hakkında rapor vermektedir



(a) Kayıtların aktif kullanımının zaman sırası



(b) Kayıt girişim grafiği

**Şekil 15.4** Grafik Renklendirme Yaklaşımı

Motorola 88000 ve MIPS R2000'e benzer özelliklere sahip bir RISC mimarisini modellemiştir. Araştırmacılar yazmaç sayısını 16 ila 128 arasında değiştirmiş ve hem tüm genel amaçlı yazmaçların hem de tamsayı ve kayan nokta kullanımını arasında bölünmüş yazmaçların kullanımını dikkate almışlardır. Çalışmaları, basit yazmaç optimizasyonunda bile 64'ten fazla yazmaç kullanımının çok az faydası olduğunu göstermiştir. Oldukça sofistike yazmaç optimizasyon teknikleriyle, 32'den fazla yazmaçla yalnızca marjinal performans artışı sağlanır. Son olarak, az sayıda yazmaçla (örneğin 16), paylaşılan yazmaç organizasyonuna sahip bir makinenin bölünmüş organizasyona sahip bir makineden daha hızlı çalıştığını belirtmişlerdir. Benzer sonuçlar, optimizasyon çabaları ile büyük yazmaç kümelerinin kullanımını karşılaştırmak yerine öncelikle az sayıda yazmaç kullanımının optimize ilgilenen bir çalışmaya rapor eden [HUGU91]'den de çıkarılabilir.

## 15.4 AZALTILMIŞ KOMUT KÜMESİ MİMARİSİ

Bu bölümde, azaltılmış komut seti mimarisinin bazı genel özelliklerine ve motivasyonuna bakacağız. Spesifik örnekler bu bölümün ilerleyen kısımlarında görülecektir. Çağdaş karmaşık komut kümesi mimarileri için motivasyonların tartışımasıyla başlıyoruz.

### Neden CISC

Daha fazla sayıda komut ve daha karmaşık komutlar içeren daha zengin komut setlerine doğru bir eğilim belirtmiştik. Bu eğilimi motive eden iki temel neden vardır: derleyicileri basitleştirme arzusu ve performansı artırma arzusu. Bu nedenlerin her ikisinin de altında programcıların HLL'lere geçisi yatkınlığıdır; mimarlar HLL'ler için daha iyi destek sağlayan makineler tasarlamaya çalışmışlardır.

Bu bölümün amacı CISC tasarımcılarının yanlış yönde ilerlediğini söylemek değildir. Aslında, teknoloji gelişmeye devam ettiğinden ve mimariler iki düzgün kategoriden ziyade bir spektrum boyunca var olduğundan, siyah-beyaz bir değerlendirmenin ortaya çıkması pek olası değildir. Bu nedenle, aşağıdaki yorumlar sadece CISC yaklaşımındaki bazı potansiyel tuzaklara işaret etmek ve RISC taraftarlarının motivasyonunun biraz anlaşılmamasını sağlamak içindir.

Belirtilen nedenlerden ilki olan derleyicinin basitleştirilmesi açık gibi görünse de öyle değildir. Derleyici yazarının görevi, HLL programları için iyi (hızlı, küçük, hızlı ve küçük) makine talimatları dizileri üreten bir derleyici oluşturmaktır (yani, derleyici bireysel HLL ifadelerini çevreleyen HLL durumları bağlamında görür). HLL ifadelerine benzeyen makine talimatları varsa, bu görev basitleştirilir. Bu mantığa RISC araştırmacıları tarafından itiraz edilmiştir ([HENN82], [RADI83], [PATT82b]). Karmaşık makine talimatlarından yararlanmanın genellikle zor olduğunu, çünkü derleyicinin yapıya tam olarak uyan durumları bulması gerektiğini bulmuşlardır. Kod boyutunu en aza indirmek, komut yürütme sayısını azaltmak ve pipelining'i geliştirmek için üretilen kodu optimize etme görevi, karmaşık bir komut kümesinde çok daha zordur. Bunun kanıtı olarak, bu bölümde daha önce bahsedilen çalışmalar, derlenmiş bir programdaki talimatların çoğunun nispeten basit olanlar olduğunu göstermektedir.

Belirtilen diğer önemli neden ise CISC'nin daha küçük ve daha hızlı programlar üreteceği beklenstisidir. Bu iddianın her iki yönünü de inceleyelim: programların daha küçük olacağı ve daha hızlı çalışacakları.

Daha küçük programların iki avantajı vardır. Program daha az bellek kapladığından, bu kaynakta bir tasarruf söz konusudur. Günümüzde bellek çok yoğun olduğundan, bu potansiyel avantaj artık zorlayıcı değildir. Daha da önemlisi, daha küçük programlar performansı artırmalıdır ve bu üç şekilde gerçekleşecektir. Birincisi, daha az komut, daha az komut bayti getirileceği anlamına gelir. İkincisi, sayfalama ortamında, daha küçük programlar daha az sayfa işgal ederek sayfa hatalarını azaltır. Üçüncüsü, önbellek(ler)de daha fazla komut siğar.

Bu mantık silsilesindeki sorun, bir CISC programının karşılık gelen bir RISC programından daha küçük olacağının kesin olmamak uzak olmasıdır. Birçok durumda, sembolik makine diliinde ifade edilen CISC programı *daha kısa* olabilir (yani, daha az komut), ancak kullanılan bellek bitlerinin sayısı fark edilir derecede daha az olmayabilir. Tablo 15.6, azaltılmış komut seti mimarisine sahip RISC I de dahil olmak üzere çeşitli makinelerde birleştirilmiş C programlarının boyutunu üç çalışmadan elde edilen sonuçları göstermektedir. Bir RISC yerine bir CISC kullanıldığında çok az tasarruf olduğunu ya da hiç tasarruf olmadığını unutmayın. PDP-11'den çok daha karmaşık bir komut setine sahip olan VAX'in PDP-11'e göre çok az tasarruf sağladığını belirtmek de ilginçtir. Bu sonuçlar IBM 801'in (bir RISC) IBM S/370 üzerindeki kodun 0,9 katı büyüklüğünde kod ürettiğini tespit eden IBM araştırmacıları [RADI83] tarafından da doğrulanmıştır. Çalışmada bir dizi PL/I programı kullanılmıştır.

Bu oldukça şaşırtıcı sonuçların birkaç nedeni vardır. CISC'lerdeki derleyicilerin daha basit komutları tercih etme eğiliminde olduğunu, böylece karmaşık komutların karmaşaklığının nadiren devreye girdiğini daha önce belirtmiştık. Ayrıca, bir CISC'de daha fazla komut olduğundan, daha uzun işlem kodları gereklidir ve bu da daha uzun komutlar üretir. Son olarak, RISC'ler bellek referansları yerine kayıtları vurgulama eğilimindedir ve birincisi daha az bit gerektirir. Bu son etkinin bir örneği ileride tartışılacaktır.

Dolayısıyla, CISC'nin dikkat çekici avantajları birlikte daha küçük programlar üretceği beklenisi gerçekleştirmeyebilir. Giderek karmaşıklaşan komut setleri için ikinci motive edici faktör, komut yürütmenin daha hızlı olacağıdır. Karmaşık bir HLL işleminin bir dizi daha ilkel talimat yerine tek bir makine talimi olarak daha hızlı yürütüleceği mantıklı . Ancak, daha basit talimatların kullanımına yönelik önyargı nedeniyle, bu böyle olmayıabilir.

**Tablo 15.6** RISC I'e Göre Kod Boyutu

	[PATT82a] 11 C Programlar	[KATE83] 12 C Programlar	[HEAT84] 5 C Programlar
RISC I	1.0	1.0	1.0
VAX-11/780	0.8	0.67	
M68000	0.9		0.9
Z8002	1.2		1.12
PDP-11/70	0.9	0.71	

Daha zengin bir komut setini barındırmak için tüm kontrol birimi daha karmaşık hale getirilmeli ve/veya mikro program kontrol deposu daha büyük hale getirilmelidir. Her iki faktör de basit komutların yürütme süresini artırır.

Aslında, bazı araştırmacılar karmaşık işlevlerin yürütülmesindeki hızlanmanın karmaşık makine talimatlarının gücünden çok yüksek hızlı kontrol deposunda bulunmalarından kaynaklandığını bulmuşlardır [RAD183]. Gerçekte, kontrol deposu bir komut önbelleği görevi görür. Dolayısıyla, donanım mimarı hangi alt programların ya da işlevlerin en sık kullanılacağını belirlemeye çalışmak ve bunları mikro kodda uygulayarak kontrol deposuna atamak durumundadır. Sonuçlar pek de iç açıcı olmamıştır. S/390 sistemlerinde, Translate ve Extended-Precision-Floating-Point-Divide gibi talimatlar yüksek depolamada bulunurken, prosedür çağrılarının ayarlanması veya bir kesme işleyicisinin başlatılmasıyla ilgili sıralama daha yavaş ana bellektedir.

Dolayısıyla, giderek karmaşıklaşan komut setlerine doğru bir eğilimin uygun olduğu açık olmaktan uzaktır. Bu durum, bazı grupların tam tersi bir yol izlemesine yol açmıştır.

### Azaltılmış Komut Kümesi Mimarilerinin Özellikleri

Azaltılmış komut seti mimarisine yönelik çeşitli farklı yaklaşımalar benimsenmiş olsa da, hepsinde ortak olan bazı özellikler vardır:

- Döngü başına bir komut
- Kayıtta kayda işlemler
- Basit adresleme modları
- Basit talimat formatları

Burada, bu özelliklerin kısa bir tartışmasını sunuyoruz. Spesifik örnekler bu bölümün ilerleyen kısımlarında inceleneceler.

Listelenen ilk özellik, **makine döngüsü başına bir makine komutu** olmasıdır. Bir *makine çevrimi*, yazmaçlardan iki işlemin alınması, bir ALU işleminin gerçekleştirilmesi ve sonucun bir yazmaçta saklanması için geçen süre olarak tanımlanır. Dolayısıyla, RISC makine komutları CISC makinelerindeki mikro komutlardan (Dördüncü Bölümde ele alınmıştır) daha karmaşık olmamalı ve onlar kadar hızlı çalışmalıdır. Basit, tek çevrimli talimatlarda mikro koda çok az ihtiyaç vardır ya da hiç yoktur; makine talimatları donanımla bağlanabilir. Bu tür talimatlar diğer makinelerdeki benzer makine talimatlarından daha hızlı çalışmalıdır, çünkü talimin yürütülmesi sırasında bir mikro program kontrol deposuna erişmek gerekli değildir.

İkinci bir özellik ise çoğu işlemin **register'dan register'a** olması, sadece basit LOAD ve STORE işlemlerinin belleğe erişmesidir. Bu tasarım özelliği komut setini ve dolayısıyla kontrol birimini basitleştirir. Örneğin, bir RISC komut seti yalnızca bir veya iki ADD komutu içerebilir (örneğin, tamsayı toplama, taşıma ile toplama); VAX 25 farklı ADD komutuna sahiptir. Diğer bir fayda ise, böyle bir mimarinin yazmaç kullanımının optimizasyonunu teşvik etmesidir, böylece sık erişilen işlenenler yüksek hızlı depolamada kalır.

Kaydediciden kaydediciye işlemlere yapılan bu vurgu RISC tasarımları için dikkate değerdir. Çağdaş CISC makineleri bu tür talimatlar sağlar, ancak aynı zamanda bellekten belleğe ve karışık yazmaç/bellek işlemlerini de içerir. Bunları karşılaştırma girişimleri

## 552 BÖLÜM 15 / AZALTILMIŞ KOMUT SETLİ BİLGİSAYARLAR

8	16	16	16
Ekle	B	C	A
<b>Hafızadan hafızaya</b> <b>I= 56, D= 96, M= 152</b>			
<b>(a) A<math>\leftarrow</math> B+ C</b>			
<b>8      16      16      16</b>			
Ekle	B	C	A
Ekle	A	C	B
Alt	B	D	D
<b>Hafızadan hafızaya</b> <b>I= 168, D= 288, M= 456</b>			
<b>(b) A<math>\leftarrow</math> B+ C; B<math>\leftarrow</math> A+ C; D<math>\leftarrow</math> D- B</b>			
<b>8      4      16</b>			
Yük	RB	B	
Yük	RC	B	
Ekle	R A	RB	RC
Mağaza	R A	A	
<b>Belleğe kaydetme</b> <b>I= 104, D= 96, M= 200</b>			
<b>8      4      4      4</b>			
Ekle	RA	RB	RC
Ekle	RB	RA	RC
Alt	RD	RD	RB
<b>Belleğe kayıt I= 60, D= 0, M= 60</b>			

**Şekil 15.5** Register-to-Register ve Memory-to-Memory Yaklaşımlarının İki Karşılaştırması

yaklaşımalar 1970'lerde, RISC'lerin ortaya çıkışmasından önce yapılmıştır. Şekil 15.5a, benimsenen yaklaşımı göstermektedir. Varsayımsal mimariler gram boyutu ve bellek trafiğinin bit sayısı üzerinden değerlendirilmiştir. Bunun gibi sonuçlar bir araştırmacının gelecekteki mimarilerin hiç yazmaç içermemesi gerektiğini önermesine yol açmıştır [MYER78]. Bir zamanlar Pyramid tarafından üretilen ve en az 528 içeren RISC makinesi hakkında o zaman ne düşündüğü merak ediliyor!

Bu çalışmalarında eksik olan şey, az sayıda yerel skalerlere sık erişimin ve büyük bir yazmaç bankası veya optimizeli bir derleyici ile çoğu operandın uzun boyunca yazmaçlarda tutulabileceği kabul edilmesiydi. Bu nedenle, Şekil 15.5b daha adil bir karşılaştırma olabilir.

Üçüncü bir özellik ise **basit adresleme modlarının** kullanılmasıdır. Neredeyse tüm RISC komutları basit kayıt adresleme kullanır. Yerleştirme dışı ve PC-göreceli gibi birkaç ek mod dahil edilebilir. Diğer, daha karmaşık modlar yazılımda basit olanlardan sentezlenebilir. Yine bu tasarım özelliği komut setini ve kontrol birimini basitleştirir.

Son bir ortak özellik de **basit komut formatlarının** kullanılmasıdır. Genelde sadece bir ya da birkaç format kullanılır. Komut uzunluğu sabittir ve kelime sınırlarına hizalanır. Alan konumları, özellikle de işlem kodu. Bu tasarım özelliğinin bir dizi faydası vardır. Sabit alanlarla, işlem kodu kod çözme ve yazmaç işlem ve erişimi aynı anda gerçekleştirilebilir. Basitleştirilmiş formatlar kontrol birimini basitleştirir. Kelime uzunlığundaki birimler getirildiği için komut optimize edilmiştir. Bir kelime sınırında hizalama aynı zamanda tek bir komutun sayfa sınırlarını geçmemesi anlamına gelir. Birlikte ele alındığında, bu özellikler potansiyelin belirlenmesi için değerlendirilebilir.

RISC yaklaşımının özel performans avantajları. Belirli bir miktar "dolaylı

kanıt" sunulabilir. İlk olarak, daha etkili optimizasyon derleyicileri geliştirilebilir. Daha ilkel talimatlarla, fonksiyonları döngülerden çıkarmak, verimlilik için kodu yeniden düzenlemek, kayıt kullanımını en üst düzeye çıkarmak ve benzeri için daha fazla fırsat vardır. Karmaşık talimatların bazı kısımlarını derleme zamanında hesaplamak bile mümkünktür. Örneğin, S/390 Move Characters (MVC) komutu bir karakter dizisini bir konumdan diğerine taşıır. Her çalıştırıldığında, hareket dizenin uzunluğuna, konumların çıkışıp çıkışmadığına ve hangi yönde çıkışlığını ve hizalamaya özelliklerinin ne olduğuna bağlı olacaktır. Çoğu durumda, bunların hepsi derleme zamanında bilinecektir. Böylece, derleyici bu işlev için optimize edilmiş bir ilkel talimat dizisi üretebilir.

Daha önce de belirtildiği gibi ikinci bir nokta, bir derleyici tarafından üretilen talimatların çoğunun zaten nispeten basit olmasıdır. Özellikle bu talimatlar için üretilmiş ve çok az ya da hiç mikro kod kullanmayan bir kontrol ünitesinin bunları benzer bir CISC'den daha hızlı yürütmesi makul görülmektedir.

Üçüncü bir nokta da komut ardisık diziliminin kullanımıyla ilgilidir. RISC araştırmacıları, komut sıralama tekniğinin azaltılmış komut seti ile çok daha etkili bir şekilde uygulanabileceğini düşünmektedir. Bu noktayı ilerde biraz ayrıntılı olarak inceleyeceğiz.

Son ve biraz daha az önemli bir nokta ise RISC işlemcilerin kesmeleri karşı daha duyarlı olmasıdır çünkü kesmeler daha ziyade temel işlemler arasında kontrol edilir. Karmaşık talimatlara sahip mimariler ya kesintileri talimat sınırlarıyla kısıtlar ya da belirli kesilebilir noktalar tanımlamalı ve bir talimatı yeniden başlatmak için mekanizmalar uygulamalıdır.

Azaltılmış komut seti mimarisi için gelişmiş performans durumu güçlündür, ancak yine de CISC için bir argüman ileri sürülebilir. Bir dizi çalışma yapılmıştır, ancak bunlar karşılaştırılabilir teknoloji ve gücü sahip makineler üzerinde yapılmamıştır. Ayrıca, çoğu çalışma azaltılmış komut setinin etkileri ile büyük kayıt dosyasının etkilerini ayırmaya çalışmamıştır. Bununla birlikte, "ikinci derece kanıtlar" düşündürücüdür.

## CISC ve RISC Karakteristikleri

RISC makinelerine yönelik ilk coşkunun ardından, (1) RISC tasarımlarının bazı CISC özelliklerinin dahil edilmesinden fayda sağlayabileceği ve

(2) CISC tasarımları bazı RISC özelliklerinin dahil edilmesinden fayda sağlayabilir. Sonuç olarak, PowerPC başta olmak üzere daha yeni RISC tasarımları artık "safer" RISC değildir ve Pentium II ve sonraki Pentium modelleri başta olmak üzere daha yeni CISC tasarımları bazı RISC özelliklerini içermektedir.

MASH95]'te yer alan ilginç bir karşılaştırma bu konuda bazı fikirler vermektedir. Tablo 15.7 bir dizi işlemciyi listelemekte ve bunları bir dizi özellik açısından karşılaştırmaktadır. Bu karşılaştırmannın amaçları doğrultusunda, aşağıdakiler klasik bir RISC için tipik kabul edilir:

- 1.** Tek bir talimat boyutu.
- 2.** Bu boyut genellikle 4 bayttır.
- 3.** Az sayıda veri adresleme modu, tipik olarak beşten az. Bu parametreyi saptamak zordur. Tabloda register ve literal modlar sayılmamış ve farklı offset boyutlarına sahip farklı formatlar ayrı ayrı sayılmıştır.

**Tablo 15.7** Bazı İşlemcilerin Özellikleri

İşlemci	Talimat boyutu sayısı	Maksimum talimat boyutu bayt cinsinden	Adresleme modlarının sayısı	Dolaylı adresleme	Aritmetik ile birlleştirilmiş yükleme/dep olama	Maksimum bellek operand sayısı	Hizalanmamış adreslemeye izin verilir	Maksimum MMU kullanım sayısı	Tamsayı kayıt belirteci için bit sayısı	FP kayıt belirteci için bit sayısı
AMD29000	1	4	1	Hayır	Hayır	1	Hayır	1	8	3 <sup>a</sup>
MIPS R2000	1	4	1	Hayır	Hayır	1	Hayır	1	5	4
SPARC	1	4	2	Hayır	Hayır	1	Hayır	1	5	4
MC88000	1	4	3	Hayır	Hayır	1	Hayır	1	5	4
HP PA	1	4	10 <sup>a</sup>	Hayır	Hayır	1	Hayır	1	5	4
IBM RT/PC	2 <sup>a</sup>	4	1	Hayır	Hayır	1	Hayır	1	4 <sup>a</sup>	3 <sup>a</sup>
IBM RS/6000	1	4	4	Hayır	Hayır	1	Evet.	1	5	5
Intel i860	1	4	4	Hayır	Hayır	1	Hayır	1	5	4
IBM 3090	4	8	2 <sup>b</sup>	hayır <sup>b</sup>	Evet.	2	Evet.	4	4	2
Intel 80486	12	12	15	hayır <sup>b</sup>	Evet.	2	Evet.	4	3	3
NSC 32016	21	21	23	Evet.	Evet.	2	Evet.	4	3	3
MC68040	11	22	44	Evet.	Evet.	2	Evet.	8	4	3
VAX	56	56	22	Evet.	Evet.	6	Evet.	24	4	0
Clipper	4 <sup>a</sup>	8 <sup>a</sup>	9 <sup>a</sup>	Hayır	Hayır	1	0	2	4 <sup>a</sup>	3 <sup>a</sup>
Intel 80960	2 <sup>a</sup>	8 <sup>a</sup>	9 <sup>a</sup>	Hayır	Hayır	1	evet <sup>a</sup>	-	5	3 <sup>a</sup>

Notlar: <sup>a</sup> Bu özelliğe uymayan RISC.

<sup>b</sup> Bu özelliğe uymayan CISC.

4. Bellekteki başka bir operandın adresini almak için bir bellek erişimi yapmanızı gerektiren dolaylı adresleme yok.
5. Yükleme/depolamayı aritmetikle birleştiren işlemler yok (örneğin, bellekten toplama, belleğe ekleme).
6. Komut başına en fazla bir bellek adresli işlenen.
7. Yükleme/depolama işlemleri için verilerin rastgele hizalanmasını desteklemez.
8. Bir komuttaki bir veri adresi için bellek yönetim biriminin (MMU) maksimum kullanım sayısı.
9. Tamsayı kayıt belirteci için bit sayısı beş veya eşittir. Bu, bir seferde en az 32 tamsayı kaydına açıkça başvurulabileceği anlamına gelir.
10. Kayan nokta kayıt belirteci için bit sayısı dört veya eşittir. Bu, bir seferde en az 16 kayan nokta kaydına açıkça başvurulabileceği anlamına gelir.

1'den 3'e kadar olan maddeler komut kod çözme karmaşıklığının bir göstergesidir. 4'ten 8'e kadar olan maddeler, özellikle sanal bellek gereksinimlerinin varlığında, boru hattı oluşturmanın kolaylığını veya zorluğunu göstermektedir. 9. ve 10. maddeler derleyicilerden iyi bir şekilde faydalananma becerisi ile ilgilidir.

Tabloda, ilk sekiz işlemci açıkça RISC mimarileridir, sonraki beşi açıkça CISC'dir ve son ikisi genellikle RISC olarak düşünülen ancak aslında birçok CISC özelliğine sahip işlemcilerdir.

## 15.5 RISC PIPELINING

### Normal Talimatlarla Pipelining

Bölüm 12.4'te tartıştığımız gibi, komut dizilimi genellikle performansı artırmak için kullanılır. Bunu bir RISC mimarisi bağlamında yeniden ele alalım. Komutların çoğu yazmacıtan yazmaçadır ve bir komut döngüsü aşağıdaki iki aşamaya sahiptir:

- I: Talimat getirme.
- E: Yürütme. Kayıt girişi ve çıkışları ile bir ALU işlemi gerçekleştirir. Yükleme ve saklama işlemleri için üç aşama gereklidir:

- I: Talimat getirme.
- E: Yürüt. Bellek adresini hesaplar.
- D: Bellek. Kayıtta belleğe veya bellekten kayda işlem.

Şekil 15.6a'da boru hattı kullanılmadan bir dizi talimatın zamanlaması gösterilmektedir. Açıkça görülmüyor ki bu savurgan bir işlemdir. Çok basit bir boru hattı bile performansı ölçüde artırabilir. Şekil 15.6b, iki farklı talimatın I ve E aşamalarının aynı anda gerçekleştirildiği iki aşamalı bir boru hattı şemasını . Boru hattının iki aşaması, bir komut getirme ve yazmacıtan belleğe ve bellekten yazmaçlara işlemler de dahil olmak üzere komutu yürütünen bir yürütme/bellek aşamasıdır. Böylece boru hattının komut getirme aşamasının

Yük rA  $\leftarrow M$   
 Yük rB  $\leftarrow M$   
 rC  $\leftarrow rA + rB$   
 ekleyin  
 Mağaza M  $\leftarrow rC$   
 Şube X

I	E	D										
	I	E	D									
		I	E									
			I	E	D							
				I	E							
					I	E						
						I	E					

(a) Sıralı yürütme

Yük rA  $\leftarrow M$   
 Yük rB  $\leftarrow M$   
 Ekle rC  $\leftarrow rA + rB$   
 Mağaza M  $\leftarrow rC$   
 a  
 Şube X  
 HAYIR

I	E	D										
	I		E	D								
		I		E								
			I		E							
				I		E						
					I		E					
						I		E				

(b) İki aşamalı pipelined zamanlama

Yük rA  $\leftarrow M$   
 Yük rB  $\leftarrow M$   
 HAYIR  
 Ekle rC  $\leftarrow rA + rB$   
 Mağaza M  $\leftarrow rC$   
 Şube X  
 HAYIR

I	E	D										
	I	E	D									
		I	E									
			I	E								
				I	E	D						
					I	E						
						I	E					

(c) Üç aşamalı pipelined zamanlama

Yük rA  $\leftarrow M$   
 Yük rB  $\leftarrow M$   
 HAYIR  
 HAYIR  
 Ekle rC  $\leftarrow rA + rB$   
 Mağaza M  $\leftarrow rC$   
 Şube X  
 HAYIR  
 HAYIR

I	E <sub>1</sub>	E <sub>2</sub>	D									
	I	E <sub>1</sub>	E <sub>2</sub>	D								
		I	E <sub>1</sub>	E <sub>2</sub>								
			I	E <sub>1</sub>	E <sub>2</sub>							
				I	E <sub>1</sub>	E <sub>2</sub>						
					I	E <sub>1</sub>	E <sub>2</sub>	D				
						I	E <sub>1</sub>	E <sub>2</sub>				
							I	E <sub>1</sub>	E <sub>2</sub>			
								I	E <sub>1</sub>	E <sub>2</sub>		

(d) Dört aşamalı pipelined zamanlama

**Sekil 15.6** Pipelining'in Etkileri

ikinci komut yürütme/bellek aşamasının ilk kısmına paralel olarak gerçekleştirilebilir. Ancak, ikinci komutun yürütme/bellek aşaması, ilk komut boru hattının ikinci aşamasını temizleyene kadar ertelemmelidir. Bu şema, seri bir şemanın iki katına kadar yürütme hızı sağlayabilir. İki sorun maksimum hızı ulaşılmasını engellemektedir. İlk olarak, tek portlu bir bellek kullanıldığını ve her aşamada yalnızca bir bellek erişiminin mümkün olduğunu varsayıyoruz. Bu da bazı komutlara bekleme durumunun eklenmesini gerektirmektedir. İkinci olarak, bir dallanma talimatı sıralı yürütme akışını kesintiye uğratır. Bunu mini devre ile karşılamak için, derleyici veya birleştirici tarafından komut akışına bir NOOP komutu eklenebilir.

Aşama başına iki bellek erişimine izin verilerek ardışık sıralama daha da geliştirilebilir. Bu, Sekil 15.6c'de gösterilen diziyi verir. Şimdi, üç adede kadar komut üst üste bindirilebilir ve iyileşme 3 kata kadar çıkabilir. Yine, dallanma komutları hızlanması mümkün olan maksimum değerin altında kalmasına neden olur. Ayrıca, veri bağımlılıklarının da bir etkisi olduğunu unutmayın. Eğer bir komut önceki komut tarafından değiştirilmiş bir işlenene ihtiyaç duyuyorsa, bir gecikme gereklidir. Yine, bu bir NOOP ile .

Şimdide kadar tartışılan ardışık sıralama, üç aşama yaklaşık olarak eşit süreliyse en iyi şekilde çalışır. E aşaması genellikle bir ALU işlemi içерdiğinde, daha uzun olabilir. Bu durumda, iki alt aşamaya bölebiliriz:

- E<sub>1</sub>: Kayıt dosyası okuma
- E<sub>2</sub>: ALU işlemi ve kayıt yazma

RISC komut setinin basitliği ve düzenliliği nedeniyle, aşamalandırmanın üç veya dört aşamalı olarak tasarlanması kolayca gerçekleştirilebilir. Sekil 15.6d dört aşamalı bir boru hattı ile elde edilen sonucu göstermektedir. Bir seferde en fazla dört komut işlenebilir ve maksimum potansiyel hızlanma 4 kattır. Veri ve dallanma gecikmelerini hesaba katmak için NOOP'ların kullanımına tekrar dikkat edin.

## Pipelining Optimizasyonu

RISC komutlarının basit ve düzenli yapısı nedeniyle, bir donanum tasarımcısının basit, hızlı bir boru hattı uygulaması daha kolaydır. Komut yürütme süresinde çok az değişiklik vardır ve boru hattı bunu yansıtacak şekilde uyarlanabilir. Ancak, veri ve dallanma bağımlılıklarının genel yürütme hızını düşürdüğüne gördük.

**GECİKMELİ DALLANMA** Bu bağımlılıkları telafi etmek için kod yeniden düzenleme teknikleri geliştirilmiştir. İlk olarak, dallanma talimatlarını ele alalım. Boru hattının verimliliğini artırmanın bir yolu olan **gecikmeli** dallanma, bir sonraki komutun yürütülmesinden sonra kadar etkili olmayan bir dallanma kullanır (*gecikmeli* terimi buradan gelir). Dallanmanın hemen ardından gelen komut konumu *gecikme yuvası* olarak adlandırılır. Bu garip prosedür Tablo 15.8'de gösterilmiştir. "Normal dallanma" etiketli sütunda, normal bir sembolik komut makine dili programı görüyoruz. 102 yürütüldükten sonra, yürütülecek bir sonraki komut 105'tir. Boru hattını düzenli hale getirmek için, bu dallanmadan sonra bir NOOP eklenir. Bununla birlikte, 101 ve 102'deki komutlar değiştirilirse daha yüksek performans elde edilir.

Şekil 15.7 sonucu göstermektedir. Şekil 15.7a, Bölüm 14'te tartışılan türden geleneksel ardışık sıralama yaklaşımını göstermektedir (örneğin, bkz. Şekil 14.11 ve 14.12). JUMP komutu 4. zamanda getirilir. 5. zamanda, JUMP komutu 103 numaralı komutun (ADD) getirilmesiyle aynı anda yürütülür. Program sayacını güncelleyen bir JUMP gerçekleştigiden, boru hattı 103 komutundan temizlenmelidir; 6. zamanda, JUMP'in hedefi olan 105 komutu yüklenir. Şekil 15.7b tipik bir RISC organizasyonu tarafından işlenen aynı boru hattını göstermektedir. Zamanlama aynıdır. Bununla birlikte, NOOP komutunun eklenmesi nedeniyle, boru hattını temizlemek için özel bir devreye ihtiyacımız yoktur; NOOP hiçbir etkisi olmadan basitçe yürütülür. Şekil 15.7c gecikmeli dallanmanın kullanımını göstermektedir. JUMP komutu 2. zamanda, 3. zamanda getirilen ADD komutundan önce getirilir. Ancak ADD komutunun, JUMP komutunun yürütülmesinin program sayacını değiştireme şansı önce alındığına dikkat edin. Bu nedenle, 4. zamanda ADD komutu 105. komutun getirilmesiyle aynı anda yürütülür. Böylece, programın orijinal semantiği korunur ancak yürütme için iki saat döngüsü daha az gereklidir.

Bu talimat değişimi koşulsuz dallanmalar, çağrılar ve geri dönüşler için başarılı bir şekilde çalışacaktır. Koşullu dallanmalar için bu prosedür körü körüne uygulanamaz. Eğer dallanma için test edilen koşul

**Tablo 15.8** Normal ve Gecikmeli Branşman

Adres	Normal Şube		Gecikmeli Şube		Optimize Edilmiş Gecikmeli Şube	
100	YÜKLE	X, rA	YÜKLE	X, rA	YÜKLE	X, rA
101	ADD	1, rA	ADD	1, rA	ZIPLA	105
102	ZIPLA	105	ZIPLA	106	ADD	1, rA
103	ADD	rA, rB	HAYIR		ADD	rA, rB
104	SUB	rC, rB	ADD	rA, rB	SUB	rC, rB
105	MAĞAZ	rA, Z A	SUB	rC, rB	MAĞAZA	rA, Z
106			MAĞAZA	rA, Z		

## 558 BÖLÜM 15 / AZALTILMIŞ KOMUT SETLİ BİLGİSAYARLAR

Zaman →								
1	2	3	4	5	6	7	8	
100 YÜK X, rA	I	E	D					
101 EKLE 1, rA		I		E				
102 JUMP 105				I	E			
103 ADD rA, rB					I	E		
105 MAĞAZA rA, Z						I	E	D

(a) Geleneksel boru hattı

1	2	3	4	5	6	7	8	
100 YÜK X, rA	I	E	D					
101 ADD 1, rA		I		E				
102 JUMP 106				I	E			
103 HAYIR					I	E		
106 MAĞAZA rA, Z						I	E	D

(b) Eklenmiş NOOP ile RISC boru hattı

1	2	3	4	5	6	
100 YÜK X, Ar	I	E	D			
101 JUMP 105		I	E			
102 EKLE 1, rA			I	E		
105 MAĞAZA rA, Z				I	E	D

(c) Tersine çevrilmiş talimatlar

**Şekil 15.7** Gecikmeli Dalın Kullanımı

hemen önceki talimat, o zaman derleyici değişimi yapmaktan kaçınmalı ve bunun yerine bir NOOP eklemelidir. Aksi takdirde, derleyici dallanmadan sonra faydalı bir komut eklemeye çalışabilir. Hem Berkeley RISC hem de IBM 801 sistemlerindeki deneyimler, koşullu dallanma komutlarının çoğunu bu şekilde optimize göstermektedir ([PATT82a], [RAD83]).

**GECİKMELİ YÜKLEME** Gecikmeli yükleme adı verilen benzer bir taktik, LOAD talimatlarında kullanılabilir. LOAD komutlarında, yüklemenin hedefi olacak yazmacı işlemci tarafından kilitlenir. İşlemci daha sonra bu yazmacı gerektiren bir komuta ulaşana kadar komut akışını yürütmeye devam eder ve bu noktada yükleme tamamlanana kadar boşta kalır. Derleyici komutları, yükleme boru hattındayken faydalı işler yapılabilecek şekilde yeniden düzenleyebilirse verimlilik artar.



Döngü Açıma Simülatörü

```

do i=2, n-1
    a[i] = a[i] + a[i-1] * a[i+1]
end do

```

(a) Orijinal döngü

```

do i=2, n-2, 2
    a[i] = a[i] + a[i-1] * a[i+1]
    a[i+1] = a[i+1] + a[i] * a[i+2]
end do

eğer (mod(n-2, 2) == i o zaman
    a[n-1] = a[n-1] + a[n-2] * a[n]
eğer son

```

(b) Döngü iki kez açıldı

**Şekil 15.8** Döngü Açma

DÖNGÜ AÇMA Komut paralellliğini geliştirmek için kullanılan bir başka derleyici teknigi de döngü açmadır [BACO94]. Döngü açma, döngünün gövdesini döngü açma faktörü ( $u$ ) olarak adlandırılan sayıda çoğaltır ve 1. adım yerine  $u$  adımı ile yineleme yapar.

Açma işlemi performansı şu şekilde artırabilir

- döngü yükünü azaltma
- boru hattı performansını iyileştirmek komut paralellini artırmak
- kayıt, veri önbelleği veya TLB yerellliğini iyileştirme

Şekil 15.8 bu iyileştirmelerin üçünü de bir örnekte göstermektedir. Döngü ek yükü yarı yarıya azalır çünkü testten önce iki yineleme gerçekleştiriliyor ve döngünün sonunda dallanma yapılır. Komut paralelligi artar çünkü ikinci atama, ilkinin sonuçları saklanırken ve döngü değişkenleri güncellenirken gerçekleştirilebilir. Dizi elemanları yazmacılara atanırsa, yazmacı yerelligi artar çünkü  $a[i]$  ve  $a[i+1]$  döngü gövdesinde iki kez kullanılır ve yineleme başına yüklemeye sayısı üçten ikiye düşer.

Son bir not olarak, komut ardışık düzeninin tasarımının sisteme uygulanan diğer optimizasyon tekniklerinden ayrı olarak gerçekleştirilmemesi gerektiğini belirtmeliyiz. Örneğin, [BRAD91b] en yüksek verimi elde etmek için komutların ardışık düzen için zamanlanmasıın ve yazmacıların dinamik tâhsisinin birlikte ele alınması gerektiğini göstermektedir.

## 15.6 MIPS R4000

Piyasada bulunan ilk RISC yonga setlerinden biri MIPS Technology Inc. tarafından geliştirilmiştir. Sistem, Stanford'da geliştirilen MIPS adını da kullanan deneyel bir sistemden esinlenmiştir [HENN84]. Bu bölümde MIPS'e bakacağız

R4000. Daha önceki MIPS tasarımları olan R2000 ve R3000 ile büyük ölçüde aynı mimariye ve komut setine sahiptir. En önemli fark, R4000'in tüm dahili ve harici veri yolları ve adresler, kayıtlar ve ALU için 32 yerine 64 bit kullanmasıdır.

64 bit kullanımının 32 bit mimariye göre bir dizi avantajı vardır. Bir işletim sisteminin bir terabayttan fazla dosyayı kolay erişim için doğrudan sanal belleğe eşlemesine yetecek kadar büyük bir adres alanı sağlar. Artık yaygın olan 1 tera bayt ve daha büyük disk sürücülerile 32 bit makinenin 4 gigabaytlık adres alanı sınırlayıcı hale gelmektedir. Ayrıca 64 bit kapasite, R4000'in IEEE çift hassasiyetli kayan noktalı sayılar ve karakter dizileri gibi verileri tek bir işlemde sekiz karaktere kadar işleyebilmesini sağlar.

R4000 işlemci, biri CPU ve diğeri bellek yönetimi için bir yardımcı işlemci içeren iki bölüme ayrılmıştır. İşlemci çok basit bir mimariye sahiptir. Amaç, komut yürütme mantığının mümkün olduğunda basit olduğu bir sistem tasarlamak ve performansı artıracak mantık için (örneğin, tüm bellek yönetim birimi) alan bırakmaktır.

İşlemci otuz iki adet 64 bitlik kaydi destekler. Ayrıca, her biri talimatlar ve veriler için yarımsız adet olmak üzere 128 Kbyte'a kadar yüksek hızlı önbellek sağlar. Nispeten büyük önbellek (IBM 3090 128 ila 256 Kbyte önbellek sağlar), sistemin büyük program kodu ve veri kümelerini işlemcinin yerelinde tutmasını, ana bellek veri yolunun yükünü ve eşlik eden pencereleme mantığı ile büyük bir kayıt dosyası ihtiyacını ortadan kaldırmasını sağlar.

### Talimat Seti

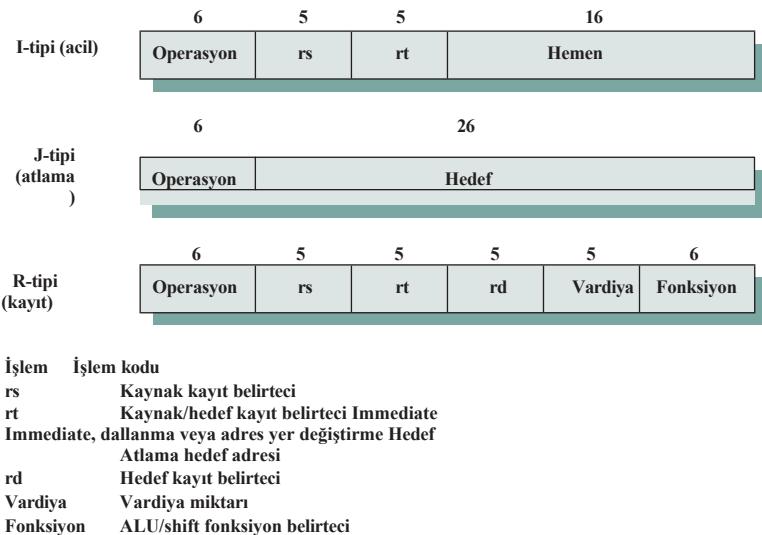
Tüm MIPS R serisi talimatları 32 bitlik tek bir kelime formatında kodlanmıştır. Tüm veri işlemleri kayıttan kayda yapılır; tek bellek referansları saf yükleme/depolama işlemleridir.

R4000 koşul kodlarını kullanmaz. Bir komut bir koşul üretirse, ilgili bayraklar genel amaçlı bir kayıtta saklanır. Bu, boru hattı mekanizmasını ve derleyici tarafından komutların yeniden sıralanmasını etkiledikleri için koşul kodlarıyla başa çıkmak için özel mantık ihtiyacını ortadan kaldırır. Bunun yerine, yazmaç-değer bağımlılıkları ile başa çıkmak için halihazırda uygulanan mekanizmalar kullanılır. Ayrıca, yazmaç dosyalarına eşlenen koşullar, yazmaçlarda saklanan diğer değerler gibi tahsis ve yeniden kullanımında aynı derleme zamanı optimizasyonlarına tabidir.

Çoğu RISC tabanlı makinede olduğu gibi MIPS de 32 bitlik tek bir komut uzunluğu kullanır. Bu tek komut uzunluğu, komut getirme ve kod çözmemi basitleştirir ve ayrıca komut getirmenin sanal bellek yönetim birimi ile etkileşimi de basitleştirir (yani, komutlar kelime veya sayfa sınırlarını geçmez). Üç komut biçimi (Şekil 15.9), işlem kodlarının ve kayıt referanslarının ortak biçimlendirmesini paylaşarak komut kod çözme işlemini basitleştirir. Daha karmaşık komutların etkisi derleme zamanında sentezlenebilir.

Donanımda yalnızca en basit ve en sık kullanılan bellek adresleme modu uygulanır. Tüm bellek referansları 32 bitlik bir kayıtten 16 bitlik bir offsetten oluşur. Örneğin, "load word" komutu şu şekildedir

```
lw r2, 128(r3) /* 128 ofset adresindeki sözcüğü şuradan yükle  
                    kayıt 3'ten kayıt 2'ye
```



**Şekil 15.9** MIPS Komut Formatları

32 genel amaçlı kaydedicinin her biri temel kaydedici olarak kullanılabilir. Bir yazmaç, r0, her zaman 0 içerir.

Derleyici, geleneksel makinelerdeki tipik adresleme modlarını sentezlemek için çoklu makine talimatlarını kullanır. Burada, lui (load upper immediate) komutunu kullanan [CHOW87]'den bir örnek verilmiştir. Bu komut, bir yazmacın üst yarısını 16 bitlik bir anlık değerle yükler ve alt yarısını sıfır ayarlar.

32-bit anlık bağımsız değişken kullanan bir assembly dili yönergesi düşünün

```
lw r2, #imm(r4) /* 32 bit kullanarak adrese sözcük yükle
                    anlık ofset #imm
                    /* register 4'ten register 2'ye ofset Bu komut
```

aşağıdaki MIPS komutlarına derlenebilir

```
lui r1, #imm-hi          /* burada #imm-hi yüksek dereceli
                            16 bit #imm
addu r1, r1, r4          /* r4'e işaretsiz #imm-hi ekleyin ve r1'e
                            koyun
lw r2, #imm-lo(r1) /* burada #imm-lo düşük dereceli
                    16 bit #imm
```

## Talimat Boru Hattı

Basitleştirilmiş komut mimaris ile MIPS çok verimli bir boru hattı oluşturabilmektedir. Genel olarak RISC pipelining'in evrimini gösterdiği için MIPS pipeline'in evrimine bakmak öğreticidir.

İlk deneysel RISC sistemleri ve ilk nesil ticari RISC işlemcileri, sistem saat döngüsü başına bir komuta yaklaşan yürütme hızlarına ulaşmaktadır. Bu performansı geliştirmek için iki işlemci sınıfı gelmiştir

Saat döngüsü başına birden fazla talimatın yürütülmesini sağlamak için: süper skaler ve süper pipelined mimariler. Özünde, bir süper skalar mimarı, boru hattının aynı aşamasındaki iki veya daha fazla talimatın aynı anda işlenebilmesi için boru hattı aşamalarının her birini çoğaltır. Süperpipelined mimarı, daha fazla ve daha ince taneli boru hattı aşamalarından yararlanan bir mimaridir. Daha fazla aşama ile, daha fazla talimat aynı anda boru hattında olabilir ve paralellik artar.

Her iki yaklaşımın da sınırlamaları vardır. Superscalar pipelining ile, farklı pipelinedeki komutlar arasındaki bağımlılıklar sistemi yavaşlatılabilir. Ayrıca, bu bağımlılıkları koordine etmek için üst kafa mantığı gereklidir. Süper pipelining ile, talimatların bir aşamadan diğerine aktarılmasıyla ilgili ek yük vardır.

Bölüm 16 süperskalar mimari üzerine bir çalışmaya ayrılmıştır. MIPS R4000, RISC tabanlı süperpipeline mimarisine iyi bir örnektir.



#### MIPS R3000 Beş Aşamalı Boru Hattı Simülörü

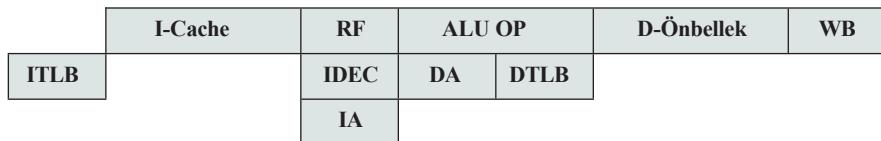
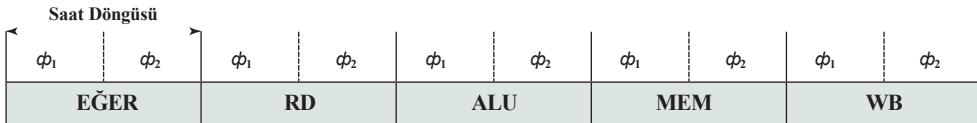
**Şekil 15.10a**, R3000'in komut işlem hattını göstermektedir. R3000'de işlem hattı her saat döngüsünde bir kez ilerler. MIPS derleyicisi, zamanın %70 ila 90'ında gecikme boşluklarını kodla doldurmak için talimatları yeniden sıralayabilir. Tüm talimatlar beş boru hattı aşamasından oluşan aynı sırayı takip eder:

- Talimat getirme;
- Kayıt dosyasından kaynak işlenen getirme;
- ALU işlemi veya veri operand adresi üretimi;
- Veri belleği referansı;
- Kayıt dosyasına geri yazın.

**Şekil 15.10a**'da gösterildiği gibi, sadece boru hattı nedeniyle paralellik değil, aynı zamanda tek bir komutun yürütülmesi içinde de paralellik vardır. 60 ns'lik saat döngüsü iki 30 ns'lık aşamaya bölünmüştür. Önbellege harici komut ve veri erişim işlemlerinin her biri, ana dahili işlemler (OP, DA, IA) gibi 60 ns gerektirir. Komut kod çözme daha basit bir işlemdir ve aynı komuttaki yazmaç getirme ile çıkışan yalnızca tek bir 30 ns aşaması gerektirir. Bir dallanma talimiği için bir adresin hesaplanması da talimat kod çözme ve yazmaç getirme ile örtüşür,  $i$  talimatındaki bir dallanma  $i$  talimatının ICACHE erişimini adresleyebilir+ 2. Benzer şekilde,  $i$  komutundaki bir yükleme,  $i+1$  komutunun OP'si tarafından hemen kullanılan verileri getirilen, bir ALUshift sonucu hiçbir gecikme olmaksızın doğrudan  $i+1$  komutuna aktarılır. Komutlar arasındaki bu sıkı bağlantı, son derece verimli bir boru hattı oluşturur.

Ayrıntılı olarak, her saat döngüsü f1 ve f2 olarak gösterilen ayrı aşamalara bölünmüştür. Her aşamada gerçekleştirilen işlevler Tablo 15.9'da özetlenmiştir.

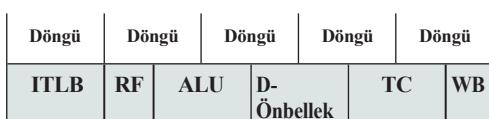
R4000, R3000'e göre bir dizi teknik ilerleme içermektedir. Daha gelişmiş teknolojinin kullanılması, saat döngü süresinin yarıya indirilmesini ve



(a) Detaylı R3000 boru hattı



(b) Gecikme süreleri azaltılmış modifiye R3000 boru hattı



(c) Paralel TLB ve önbellek erişimleri ile optimize edilmiş R3000 işlem hattı

**Sekil 15.10** R3000 Boru Hattının Geliştirilmesi

EĞER	= Talimat getirme
RD	= Okuyun
MEM	= Bellek erişimi
WB	= Kayıt dosyasına geri yazma
I-Cache	Cache= Komut önbelleği erişimi
RF	İşleneni yazmacaçtan getir D-
ALU	Cache= Veri önbelleği erişimi
DTLB	ITLB= Komut adresi çevirisi IDEC = Talimat kod çözme
D-Önbellek	IA = Talimat adresini hesapla DA = Veri sanal adresini hesapla
WB	DTLB= Veri adresi çevirisi
	TC = Veri önbellek etiketi kontrolü

**Tablo 15.9** R3000 Boru Hattı Aşamaları

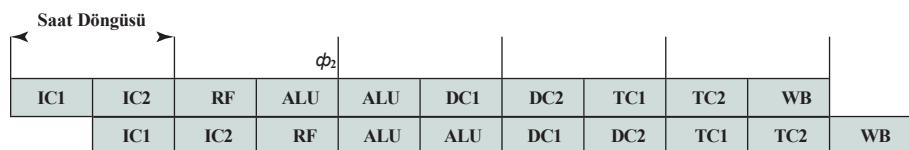
Boru Hattı Aşaması	Aşama	Fonksiyon
EĞER	f1	TLB'yi kullanarak bir komutun sanal adresini fiziksel adrese çevirme (bir dallanma kararından sonra).
EĞER	f2	Fiziksel adresi talimat adresine gönderin.
RD	f1	Komut önbelleğinden komut döndürür.
		Getirilen talimatın etiketlerini ve geçerliğini karşılaştırır.
RD	f2	Kod çözme talimatı.
		Kayıt dosyasını okuyun.
		Dallanma varsa, dallanma hedef adresini hesaplayın.
ALU	f1 + f2	Kayıttan kayda işlem yapılyorsa, aritmetik veya mantıksal işlem gerçekleştirilir.
ALU	f1	Eğer bir dal varsa, dalın alınıp alınmayacağına karar verin.
		Bir bellek referansı (yükleme veya depolama) ise, veri sanal adresini hesaplayın.
ALU	f2	Bir bellek referansı ise, TLB kullanarak veri sanal adresini fiziksel adrese çevirin.
MEM	f1	Bir bellek referansı ise, fiziksel adresi veri önbelleğine gönderir.
MEM	f2	Bir bellek referansı ise, veri önbelleğinden veri döndürür ve etiketleri kontrol eder.
WB	f1	Kayıt dosyasına yazın.

30 ns ve kayıt dosyasına erişim süresinin yarıya indirilmesi. Buna ek olarak, çip üzerinde daha fazla yoğunluk bulunmakta ve bu da komut ve veri önbelleklerinin çip üzerinde yer almasını sağlamaktadır. Nihai R4000 işlem hattına bakmadan önce, R4000 teknolojisini kullanarak performansı artırmak için R3000 işlem hattının nasıl değiştirebileceğini ele alalım.

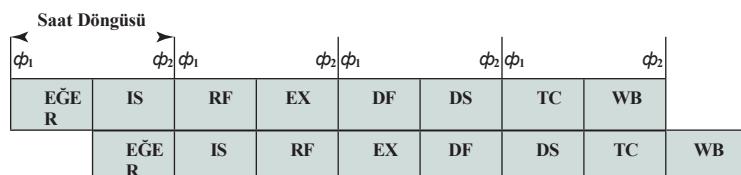
Şekil 15.10b ilk adımı göstermektedir. Bu şekildeki döngülerin Şekil 15.10a'dakilerin yarısı kadar uzun olduğunu unutmayın. Aynı üzerinde oldukları için, talimat ve veri önbellek aşamaları sadece yarısı kadar sürer; bu nedenle hala sadece bir saat döngüsü kaplarlar. Yine, kayıt dosyası erişiminin hızlandırılması nedeniyle, kayıt okuma ve yazma hala bir saat döngüsünün sadece yarısını kaplar.

R4000 önbellekleri çip üzerinde olduğundan, sanaldan fiziksel adrese geçiş önbellek erişimi geciktirebilir. Bu gecikme, sanal olarak indekslenmiş önbelleklerin uygulanması ve paralel önbellek erişimi ve adres çevirisine geçirilmesiyle azaltılır. Şekil 15.10c'de bu iyileştirme ile optimize edilmiş R3000 boru hattı gösterilmektedir. Olayların sıkıştırılması nedeniyle, veri önbellek etiketi kontrolü önbellek erişiminden sonraki döngüde ayrı olarak gerçekleştirilir. Bu kontrol, veri öğesinin olup olmadığını belirler. Süper boru hatlı bir sisteme, mevcut donanım, her boru aşamasını bölmek için boru hattı kayıtları eklenerken döngü başına birkaç kez kullanılır. Esasen, her bir süper boru hattı aşaması, süper boru hattı derecesine bağlı olarak temel saat frekansının bir katında çalışır. R4000 teknolojisi, 2. derece süper boru hattına izin verecek hız ve yoğunluğa sahiptir. Şekil 15.11a'da bu süper boru hattı kullanılarak optimize edilmiş R3000 boru hattı gösterilmektedir. Bunun esasen aynı dinamik olduğunu unutmamız gerekmektedir. Şekil 15.10c'deki gibi bir yapı.

Daha fazla iyileştirme yapılmaktadır. R4000 için çok daha büyük ve özelleşmiş bir toplayıcı tasarlanmıştır. Bu, ALU işlemlerinin aşağıdaki hızlarda yürütülmesini mümkün kılar



(a) Optimize edilmiş R3000 boru hattının süperpipelined uygulaması



(b) R4000 boru hattı

IF= Talimat getirme ilk yarısı IS =  
 Talimat getirme ikinci yarısı  
 RF= İşlenenleri yazmaçtan getirme EX=  
 Komut yürütme  
 IC = Öğretim önbelleği

DC= Veri önbelleği  
 DF= Veri önbelleği ilk yarısı DS=  
 Veri önbelleği ikinci yarısı TC=  
 Etiket kontrolü  
 WB= Kayıt dosyasına geri yaz

**Sekil 15.11** Teorik R3000 ve Gerçek R4000 Süper Boru Hatları

iki kat daha hızlı. Diğer iyileştirmeler, yüklemelerin ve depolamaların iki kat daha hızlı yürütülmesini sağlar. Ortaya çıkan boru hattı Şekil 15.11b'de gösterilmektedir.

R4000 sekiz ardışık düzen aşamasına sahiptir, yani ardışık düzende aynı anda sekiz talimat bulunabilir. İşlem hattı, saat döngüsü başına iki aşama hızında ilerler. Sekiz ardışık düzen aşaması aşağıdaki gibidir:

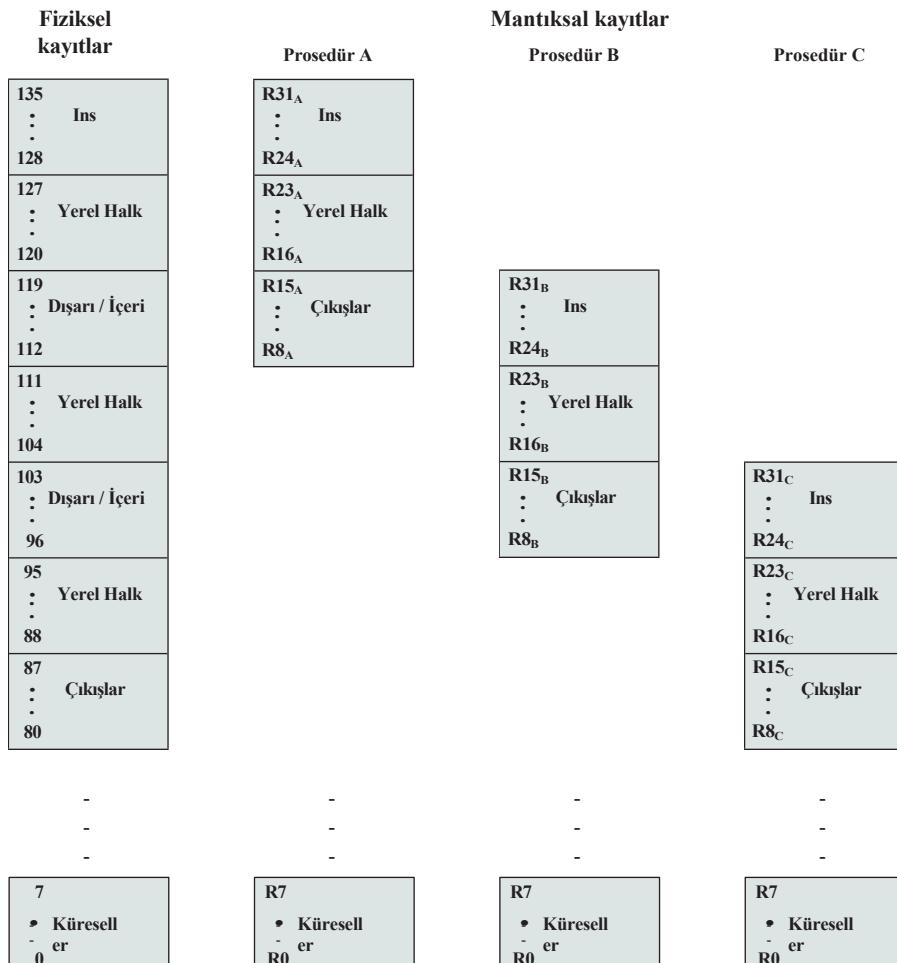
- **Komut getirme ilk yarısı:** Sanal adres, komut önbelleğine ve çeviri ara belleğine sunulur.
- **Talimat getirme ikinci yarı:** Komut önbelleği komutu çıkarır ve TLB fiziksel adresi oluşturur.
- **Kayıt dosyası:** Üç faaliyet paralel olarak gerçekleşir:
  - Talimatın kodu çözülür ve kilitlenme koşulları kontrol edilir (yani, bu talimat önceki bir talimatın sonucuna bağlıdır).
  - Talimat önbellek etiketi kontrolü yapıılır.
  - Operandlar kayıt dosyasından alınır.
- **Talimat yürütme:** Üç faaliyetten biri gerçekleştirilebilir:
  - Komut bir kayıttan kayda işlemse, ALU aritmetik veya mantıksal işlemi gerçekleştirir.
  - Komut bir yükleme veya depolama ise, veri sanal adresi hesaplanır.
  - Komut bir dallanma ise, dallanma hedefi sanal adresi hesaplanır ve dallanma koşulları kontrol edilir.
- **Önce veri önbelleği:** Sanal adres veri önbelleğine ve TLB'ye sunulur.
- **Veri önbelleği ikinci:** TLB fiziksel adresi üretir ve veri önbelleği verileri çıkarır.
- **Etiket kontrolü:** Yüklemeler ve depolamalar için önbellek etiketi kontrolleri gerçekleştirilir.
- **Geri yaz:** Komut sonucu kayıt dosyasına geri yazılır.

## 15.7 SPARC

SPARC (Ölçeklenebilir İşlemci Mimarisi) Sun Microsystems tarafından tanımlanan bir mimariyi ifade eder. Sun kendi SPARC uygulamasını geliştirmiştir ancak SPARC uyumlu makineler üretmeleri için mimariyi diğer satıcılar da lisanslamaktadır. SPARC mimarisi Berkeley RISC I makinesinden esinlenmiştir ve komut seti ve kayıt organizasyonu Berkeley RISC modeline yakından dayanmaktadır.

### SPARC Kayıt Kümesi

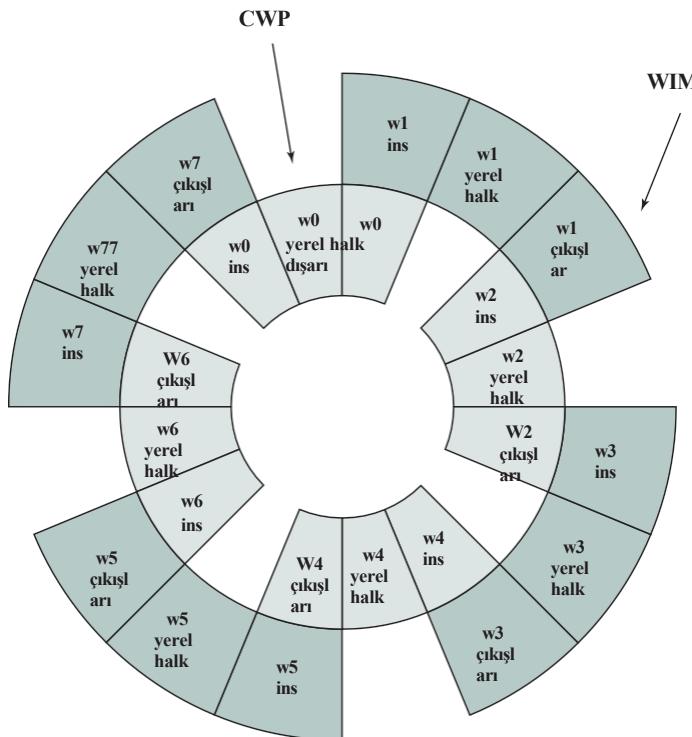
Berkeley RISC'de olduğu gibi SPARC da kayıt pencerelerini kullanır. Her bir pencere 24 yazmaç için adreslenebilirlik sağlar ve toplam pencere sayısı uygulamaya bağlıdır ve 2 ila 32 pencere arasında değişir. Şekil 15.12, toplam 136 fiziksel kayıt kullanarak 8 pencereyi destekleyen bir uygulamayı göstermektedir; Bölüm 15.2'deki tartışmanın gösterdiği gibi, bu makul bir pencere sayısı gibi görülmektedir. 0'dan 7'ye kadar olan fiziksel kayıtlar tüm prosedürler tarafından paylaşılan global kayıtlardır.



Şekil 15.12 Üç Yordamlı SPARC Kayıt Penceresi Düzeni

Her prosedür 0 ile 31 arasındaki mantıksal kayıtları görür. *Giriş* olarak adlandırılan 24 ile 31 arasındaki mantıksal , çağrıran (üst) yordamla paylaşırlar; ve çıkış olarak adlandırılan 8 ile 15 arasındaki mantıksal , çağrılan (alt) herhangi bir yordamla paylaşırlar. Bu iki kısım diğer pencerelelerle örtüşür. *Locals* olarak adlandırılan 16 ile 23 arasındaki mantıksal paylaşılmaz ve diğer pencerelelerle çakışmaz. Yine, Bölüm 12.1'deki tartışmanın gösterdiği gibi, parametre geçiş için 8 yazmacın kullanılabilirliği çoğu durumda yeterli olmalıdır (örneğin, bkz. Tablo 15.4).

Şekil 15.13 register çakışmasının başka bir görünümüdür. Çağırılan yordam aktarılacak parametreleri kendi *dış* yerleştirir; çağrılan yordam da bu fiziksel kayıtları kendi  *iç* kayıtları olarak ele alır. İşlemci, işlemci durum kaydında (PSR) bulunan ve o anda yürütülen prosedürün penceresini gösteren bir geçerli pencere işaretçisi WP) tutar. Yine PSR'de bulunan pencere geçersiz maskesi (WIM) hangi pencerelerin geçersiz olduğunu gösterir.



**Şekil 15.13** SPARC'ta Dairesel Yığın Oluşturan Sekiz Kayıt Penceresi

SPARC kayıt mimarisi ile, bir prosedür çağrısı için kayıtları kaydetmek ve geri yüklemek genellikle gerekli değildir. Derleyicinin yalnızca bir yordam için yerel yazmaçları verimli bir şekilde tahsis etmemek ilgilenmesi ve arasında yazmaç tahsisini ile ilgilenmemesi gereğinden derleyici basitleştirilmiştir.

### Talimat Seti

SPARC komutlarının çoğu yalnızca yazmaç işlenenlerine başvurur. Register'dan register'a talimatların üç işleneni vardır ve şu şekilde ifade edilebilir

$$R_d \mathbf{S} R_{S1} op S2$$

Burada  $R_d$  ve  $R_{S1}$  yazmaç referanslarıdır;  $S2$  bir yazmaç ya da 13 bitlik bir anlık işlev işaret edebilir. Sıfırıncı yazmaç ( $R_0$ ) 0 değeri ile sabitlenmiştir. Bu form, yüksek oranda yerel skaler ve sabit içeren tipik programlar için çok uygundur.

Mevcut ALU işlemleri aşağıdaki gibi grupperlabilir:

- Tamsayı toplama (taşıma ile veya taşımasız).
- Tamsayı çıkarma (taşıma ile veya taşıma olmadan).
- Bitsel Boole AND, OR, XOR ve bunların olumsuzlamaları.
- Sol mantıksal, sağ mantıksal veya sağ aritmetik kaydırma.

## 568 BÖLÜM 15 / AZALTILMIŞ KOMUT SETLİ BİLGİSAYARLAR

Kaydirmalar hariç tüm bu komutlar isteğe bağlı olarak dört durum kodunu (SIFIR, NEGATİF, AŞIRI AKIŞ, TAŞIMA) ayarlayabilir. İşaretli tamsayılar 32-bit ikiye tümleyen biçiminde gösterilir.

Yalnızca basit yükleme ve saklama talimatları belleğe başvurur. Word (32 bit), doubleword, halfword ve byte için ayrı yükleme ve saklama komutları vardır. Son iki durum için, bu büyülükleri işaretli veya işaretsiz sayılar olarak yüklemek için talimatlar vardır. İşaretli sayılar 32 bitlik hedef yazmacını doldurmak için işaretle genişletilir. İşaretsiz sayılar sıfırlarla doldurulur.

Register dışında kullanılabilen tek adresleme modu yer değiştirme modudur., bir işlenenin etkin adresi (EA), bir yazmaca bulunan bir adresten bir yer değiştirmeden oluşur:

$$\begin{aligned} EA &= (R_{(S1)}) + S2 \\ \text{veya } EA &= (R_{(S1)}) + (R_{S2}) \end{aligned}$$

İkinci işlenenin hemen veya bir yazmaca referansı olmasına bağlı olarak. Bir yükleme ya da saklama işlemi gerçekleştirmek için komut döngüsüne ekstra bir aşama eklenir. İkinci aşama sırasında, bellek adresi ALU kullanılarak hesaplanır; yükleme veya saklama üçüncü bir aşamada gerçekleşir. Bu tek adresleme modu oldukça çok yönlüdür ve Tablo 15.10'da belirtildiği gibi diğer adresleme modlarını sentezlemek için kullanılabilir.

SPARC adresleme kabiliyetini MIPS'inkiyle karşılaştırmak öğreticidir. MIPS, SPARC'taki 13 bitlik ofsete kıyasla 16 bitlik bir ofset kullanır. Öte yandan, MIPS bir adresin iki kayıtçının içeriğinden oluşturulmasına izin vermez.

### Talimat Formatı

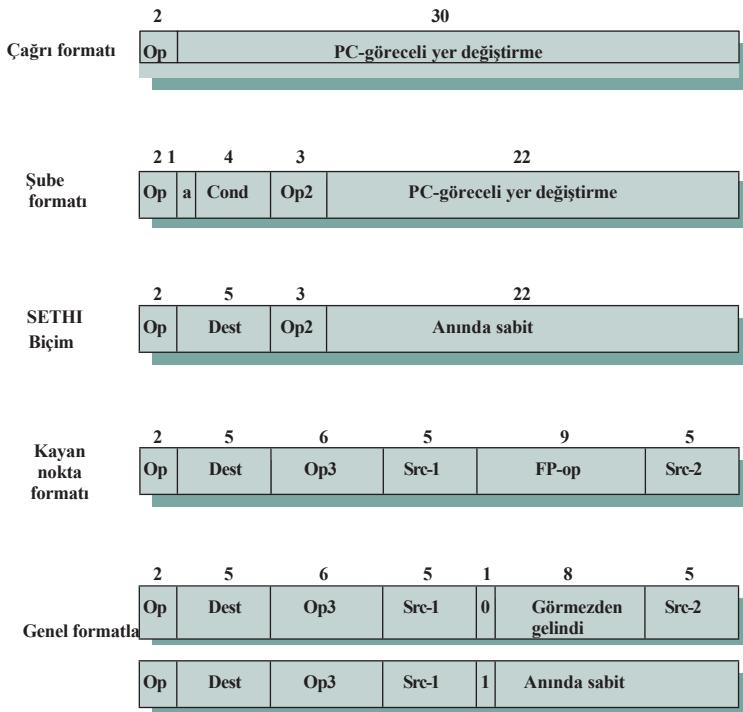
MIPS R4000'de olduğu gibi, SPARC da basit bir 32 bit komut biçimini seti kullanır (Şekil 15.14). Tüm komutlar 2 bitlik bir işlem kodu ile başlar. Çoğu komut için bu, formattın başka bir yerindeki ek işlem kodu bitleriyle genişletilir. Call için, 30 bitlik bir immediate operand, iki sıfır biti ile sağa doğru genişletilerek ikiye tümleyen formunda 32 bitlik bir PC-relative adres oluşturur. Talimatlar 32 bitlik bir sınır üzerinde hizalanır, böylece bu adresleme biçimini yeterli olur.

Branch komutu, dört standart koşul kodu bitine karşılık gelen 4 bitlik koşul alanı içerir, böylece herhangi bir koşul kombinasyonu test edilebilir. 22 bitlik PC-göreli adres, sağıdaki iki sıfır biti ile genişletilerek

**Tablo 15.10** SPARC Adresleme Modları ile Diğer Adresleme Modlarının Sentezlenmesi

Talimat Türü	Adresleme Modu	Algoritma	SPARC Eşdeğeri
Kayıttan kayda	Hemen	operand = A	S2
Yükle, sakla	Doğrudan	EA = A	R <sub>0</sub> + S <sub>2</sub>
Kayıttan kayda	Kayıt Olun	EA = R	R <sub>S1</sub> , SS2
Yükle, sakla	Dolaylı Kayıt	EA = (R)	R <sub>S1</sub> + 0
Yükle, sakla	Yer değiştirme	EA = (R) + A	R <sub>S1</sub> + S2

*Not: S2 = ya bir kayıt operandı ya da 13 bitlik bir anlık .*



**Şekil 15.14** SPARC Komut Formatları

24 bitlik ikiye tümleyen göreceli adres oluşturur. Dallanma komutunun alışılmadık bir özelliği de iptal bitidir. Annul biti ayarlanmadığında, dallanmanın yapılip yapılmadığına bakılmaksızın, dallanmadan sonraki komut her zaman yürütülür. Bu, birçok RISC makinesinde bulunan ve Bölüm 15.5'te açıklanan tipik gecikmeli işlemidir (bkz. Şekil 15.7). Ancak, iptal biti ayarlandığında, dallanmayı izleyen komut yalnızca dallanma gerçekleşse yürüttür. İşlemci, zaten boru hattında olsa bile bu komutun etkisini bastırır. Bu iptal biti kullanışlıdır çünkü derleyicinin koşullu bir dallanmayı takip eden gecikme boşluğunu doldurmasını kolaylaştırır. Dallanmanın hedefi olan komut her zaman gecikme yuvasına yerleştirilebilir, çünkü dallanma gerçekleşmezse komut iptal edilebilir. Bu tekniğin tercih edilmesinin nedeni, koşullu dallanmaların genellikle zamanın yarısından fazlasında gerçekleşmesidir.

SETHI komutu 32 bitlik bir sabit oluşturmak için kullanılan özel bir komuttur. Bu özellik büyük veri sabitleri oluşturmak için gereklidir; örneğin, bir load veya store komutu için büyük bir offset oluşturmak için kullanılabilir. SETHI komutu, 22 bitlik acil ile bir kaydin 22 yüksek sıralı bitini ayarlar ve düşük sıralı 10 biti sıfırlar. Genel formatlardan birinde 13 bite kadar bir anlık sabit belirtilebilir ve böyle bir komut yazmacın kalan 10 bitini doldurmak için kullanılabilir. Bir yükleme veya saklama komutu da doğrudan bir

## 570 BÖLÜM 15 / AZALTILMIŞ KOMUT SETLİ BİLGİSAYARLAR

adresleme modu. Bellekteki K konumundan bir değer yüklemek için aşağıdaki SPARC talimatlarını kullanabiliriz:

sethi	%hi(K), %r8	:konum adresinin yüksek sıralı 22 bitini yükle :K, r8 kaydına
Ld	[%r8+ %lo(K)], %r8	:K konumunun içeriğini r8'e yükle

hi ve %lo makroları, bir konumun uygun adres bitlerinden oluşan immediate operandları tanımlamak için kullanılır. SETHInin bu kullanımı MIPS üzerindeki lui komutunun kullanımına benzer.

Kayan nokta formatı, kayan nokta işlemleri için kullanılır. İki kaynak ve bir hedef kaydedici belirlenmiştir.

Son olarak, yüklemeler, depolamalar, aritmetik ve mantıksal işlemler dahil olmak üzere diğer tüm işlemler Şekil 15.14'te gösterilen son iki formattan birini kullanır. Biçimlerden biri iki kaynak kaydedici ve bir hedef kaydedici kullanırken, diğeri bir kaynak kaydedici, bir 13 bitlik anlık işlenen ve bir hedef kaydedici kullanır.

### 15.8 RISC VE CISC TARTIŞMASI

Uzun yıllar boyunca bilgisayar mimarisinin ve organizasyonundaki genel eğilim, işlemci karmaşıklığının artırılması yönünde olmuştur: daha fazla komut, daha fazla adresleme modu, daha fazla özel kayıt ve . RISC hareketi bu eğilimin arkasındaki felsefeden eğlenceli bir kopusu temsil etmektedir. Doğal olarak, RISC sistemlerinin ortaya çıkması ve savunucuları tarafından RISC erdemlerini öven makalelerin yayınlanması, CISC mimarilerinin tasarımda yer alanların tepkisine yol açtı.

RISC yaklaşımının yararlarını değerlendirmek için yapılan çalışmalar iki kategoride :

- **Niceliksel:** Karşılaştırılabilir teknoloji kullanan RISC ve CISC makinelerindeki program boyutunu ve yürütme hızını karşılaştırma girişimleri.
- **Niteliksel:** Üst düzey dil desteği ve VLSI gayrimenkulünün optimum kullanımı gibi konuları inceler.

Nicel değerlendirme konusundaki çalışmaların çoğu RISC sistemleri üzerinde çalışanlar tarafından yapılmıştır [PATT82b, HEAT84, PATT84] ve genel olarak RISC yaklaşımı lehine olmuştur. Diğerleri konuyu incelemiş ve ikna olmamışlardır [COLW85a, FLYN87, DAVI87]. Bu tür karşılaştırmalara kalkışmanın çeşitli sorunları vardır [SERL86]:

- Yaşam döngüsü maliyeti, teknoloji düzeyi, kapı karmaşıklığı, derleyicinin gelişmişliği, işletim sistemi desteği vb. açılardan karşılaştırılabilir bir RISC ve CISC makine çifti yoktur.
- Kesin bir program test seti mevcut değildir. Performans programa göre değişir.
- Derleyici yazımındaki beceri nedeniyle donanım etkilerini etkilerden ayırmak zordur.
- RISC üzerine yapılan karşılaştırmalı analizlerin çoğu ticari ürünler yerine "oyuncak" makineler üzerinde yapılmıştır. Ayrıca, piyasada bulunan çoğu

RISC olarak tanıtılan makineler RISC ve CISC karakteristiklerinin bir karışımıma sahiptir. Bu nedenle, ticari, "saf" bir CISC makinesi (örneğin VAX, Pentium) ile adil bir karşılaşılma yapmak zordur.

Niteliksel değerlendirme neredeyse tanımı gereği özneldir. Bazı dikkatlerini bu tür bir değerlendirmeye yöneltmiştir [COLW85a, WALL85], ancak sonuçlar en iyi ihtimalle belirsizdir ve kesinlikle çürütmeye [PATT85b] ve elbette karşı çürütmeye [COLW85b] tabidir.

Son yıllarda, RISC ve CISC tartışması büyük ölçüde sona ermiştir. Bunun nedeni, teknolojilerin kademeli olarak birbirine yaklaşmasıdır. Çip yoğunlukları ve ham donanım hızları arttıkça, RISC sistemleri daha karmaşık hale gelmiştir. Aynı zamanda, maksimum performans elde etme çabasıyla, CISC tasarımları geleneksel olarak RISC ile ilişkilendirilen, genel amaçlı kayıtların sayısının artırılması ve komut hattı tasarımasına daha fazla vurgu yapılması gibi konulara odaklanmıştır.

## 15.9 ANAHTAR TERİMLER, GÖZDEN GEÇİRME SORULARI VE PROBLEMLER

### Anahtar Terimler

karmaşık komut seti komputer (CISC) gecikmeli branşman gecikmeli yük	yüksek seviyeli dil (HLL) azaltılmış komut seti bilgisayar (RISC) kayıt dosyası	kayıt penceresi SPARC
---	---	-----------------------

### İnceleme Soruları

- 15.1** RISC organizasyonunun bazı tipik ayırt edici özellikleri nelerdir?
- 15.2** RISC makinelerinde kayıt-bellek işlemlerini enaza indirmek için kullanılan iki temel yaklaşımı kısaca açıklayınız.
- 15.3** İç içe prosedürlerin yerel değişkenlerini işlemek için dairesel bir kayıt tamponu kullanılıyorsa, global değişkenleri işlemek için iki yaklaşım açıklayın.
- 15.4** RISC komut seti mimarisinin bazı tipik özelliklerini nelerdir?
- 15.5** Gecikmeli şube nedir?

### Problemler

- 15.1** Şekil 4.21'deki çağrı-geri dönüşü göz önüne alındığında, kaç tane taşma ve düşük akış (her biri bir yazmacık kaydetme/geri yüklemeye neden olur) pencere boyutu
  - a. 5?
  - b. 8?
  - c. 16?
- 15.2** Şekil 15.2'nin tartışmasında, bir kazancın yalnızca ilk iki bölümünün kaydedildiği veya geri yükleniği belirtildi. Geçici kayıtları kaydetmek neden gerekli değildir?
- 15.3** Bölüm 15.5'te tartışılan çeşitli boru hattı şemalarını kullanarak belirli bir program için yürütme süresini belirlemek istiyoruz. İzin verin

$N$ = yürütülen talimat sayısı  
 $D$ = bellek erişim sayısı  
 $J$ = atlama talimatlarının sayısı

Basit sıralı şema (Şekil 15.6a) için yürütme süresi  $2N + D$  aşamadır. İki aşamalı, üç aşamalı ve dört aşamalı pipelining için formüller türetiniz.

- 15.4** NOOP sayısını azaltmak için Şekil 15.6d'deki kod dizisini yeniden düzenleyin.

- 15.5** Yüksek seviyeli bir dilde aşağıdaki kod parçasını düşünün:

```
for I in 1...100 döngü
    S d S+ Q(I). VAL
Döngüyü sonlandır;
```

Q'nun 32 baytlık kayıtlardan oluşan bir dizi olduğunu ve VAL alanının her kaydın ilk 4 baytında yer aldığı varsayılmı. X86 kodunu kullanarak bu program parçasını aşağıdaki gibi derleyebiliriz:

	MOV	ECX,1	İyi tutmak için ECX kaydını kullanın
LP:	IMUL	EAX, ECX, 32	;EAX içinde ofset al
	MOV	EBX, Q[EAX]	;VAL alanını yükle
	EKLE	S, EBX	;S'ye ekleyin
	INC	ECX	;artış I
	CMP	ECX, 101	:101 ile karşılaşır
	JNE	LP	;I= 100 olana kadar döngü

Bu program, ikinci işleneni üçüncü işlenenekti anlık değerle çarpan ve sonucu birinci işlenene yerleşiren IMUL komutunu kullanır (bkz. Problem 10.13). Bir RISC savunucusu, akıllı bir derleyicinin IMUL gibi gereksiz karmaşık komutları ortadan kaldırabileceğini göstermek istemektedir. Yukarıdaki x86 programını IMUL komutunu kullanmadan yeniden yazarak gösterimi sağlayın.

- 15.6** Aşağıdaki döngüyü düşünün:

```
S := 0;
for K:= 1 ila 100 do
    S:= S - K;
```

Bunun genel bir assembly diline basit bir çevirisi aşağıdaki gibi görünecektir:

	LD	R1, 0	;S değerini R1'de tutun
	LD	R2,1	K değerini R2'de tutun
LP	SUB	R1, R1, R2	;S:= S - K
	BEQ	R2, 100, ÇIKIŞ	;done if K= 100
	ADD	R2, R2, 1	;else increment K
	JMP	LP	;döngü başlangıcına geri dön

Bir RISC makinesi için derleyici, işlemcinin gecikmeli dallanma mekanizmasını kullanabilmesi için bu koda gecikme yuvaları ekleyecektir. JMP komutu ile başa çıkmak kolaydır, çünkü bu komutu her zaman SUB komutu takip eder; bu nedenle,

SUB komutunun bir kopyasını JMP'den sonraki gecikme yuvasına yerlestirebiliriz. BEQ bir zorluk teşkil eder. Kodu olduğu gibi bırakamayız, çünkü ADD komutu çok fazla kez çalıştırılacaktır. Bu nedenle, bir NOP komutuna ihtiyaç vardır. Ortaya çıkan kodu gösterin.

- 15.7** Bir RISC makinesinin derleyicisi hem sembolik yazmaçların gerçek yazmaçlara eşlenmesini hem de boru hattı verimliliği için komutların yeniden düzenlenmesini yapabilir. Bu iki işlemin hangi sırayla yapılması gerektiği ilginç bir soru olarak ortaya çıkmaktadır. Aşağıdaki program parçasını düşünün:

LD	SR1, A	;A'yi sembolik kayıt 1'e yükle
LD	SR2, B	;B'yi sembolik kayıt 2'ye yükle
ADD	SR3, SR1, SR2	;SR1 ve SR2'nin içeriğini toplar ve SR3'te saklar
LD	SR4, C	
LD	SR5, D	
ADD	SR6, SR4, SR5	

- a. Önce kayıt eşlemesini yapın ve ardından olası komut sıralamasını yapın. Kaç makine kaydı kullanılıyor? Herhangi bir boru hattı iyileştirmesi yapıldı mı?
- b. Orijinal programdan başlayarak, şimdi komutları yeniden sıralayın ve ardından olası eşleştirmeleri yapın. Kaç makine kaydı kullanıyor? Herhangi bir boru hattı iyileştirmesi oldu mu?

- 15.8** Tablo 15.7'ye aşağıdaki işlemciler için girişler ekleyin:

- a. Pentium II
- b. ARM

- 15.9** Birçok durumda, MIPS komut setinin bir parçası olarak listelenmeyen yaygın makine komutları tek bir MIPS komutu ile sentezlenebilir. Bunu aşağıdakiler için gösterin:

- a. Kayıttan kayda geçiş
- b. Artırma, azaltma
- c. Tamamlayıcı
- d. Negate
- e. Temiz

- 15.10** Bir SPARC uygulamasında  $K$  kayıt penceresi vardır. Fiziksnel yazmaçların sayısı  $N$  kaçtır?

- 15.11** SPARC, CISC makinelerinde yaygın olarak bulunan bir dizi talimattan yoksundur. Bunlardan bazıları, her zaman 0 olarak ayarlanan R0 yazmacı veya sabit bir işlenen kullanılarak kolayca simülle edilebilir. Bu simüle talimatlara sözde talimatlar denir ve SPARC assembler tarafından tanınır. Her biri tek bir SPARC komutu içeren aşağıdaki sözde yapıların nasıl simüle edileceğini gösterin. Bunların hepsinde src ve dst yazmaçları ifade eder. (İpucu: R0'a depolamanın bir etkisi yoktur.)

- |                                |            |            |
|--------------------------------|------------|------------|
| a. MOV src, dst                | d. dst     | g. DEC dst |
| b. src1, src2'yi karşılaştırın | DEĞİL      | h. CLR dst |
| c. TEST src1                   | e. NEG dst | i. NOP     |
|                                | f. INC dst |            |

- 15.12** Aşağıdaki kod parçasını göz önünde bulundurun:

```

eğer K> 10
    L := K+ 1
başka
    L := K - 1
  
```

Bu ifadenin SPARC assembler'a basit bir aşağıdaki şekilde olabilir:

sethi	%hi(K), %r8	;konum adresinin yüksek sıralı 22 bitini yükle ;K, r8 kaydına
ld	[%r8+ %lo(K)], %r8	;K konumunun içeriğini r8'e yükle
cmp	%r8, 10	;r8'in içeriğini 10 ile karşılaştır
ble	L1	;branch if (r8) ... 10
nop		
sethi	%hi(K), %r9	
ld	[%r9+ %lo(K)], %r9	;K konumunun içeriğini r9'a yükle
inc	%r9	;1'i (r9)'a ekle
sethi	%hi(L), %r10	
st	r9, [%r10+ %lo(L)] L2	;L konumuna (r9) depolayın
b		
nop		
L1:	sethi %hi(K), %r11	
	ld [%r11+ %lo(K)], %r12	;K konumunun içeriğini r12'ye yükle
	dec %r12	;(%r12)'den 1 çıkar
	sethi %hi(L), %r13	
	st r12, [%r13+ %lo(L)]	;L konumuna (r12) depolayın

L2:

Kod, gecikmeli dallanma işlemine izin vermek için her dallanma komutundan sonra bir nop içerir.

- RISC makineleriyle hiçbir ilgisi olmayan standart derleyici optimizasyonları genellikle yukarıdaki kod üzerinde iki dönüşümün gerçekleştirilebilmesinde etkilidir. Yüklerden ikisinin gereksiz olduğuna ve deponun kodda farklı bir yere taşınması halinde iki deponun birleştirileceğine dikkat edin. Bu iki değişikliği yaptıktan sonra programı gösterin.
- Artık SPARC'a özgü bazı optimizasyonlar gerçekleştirmek mümkündür. Ble komutundan sonra gelen nop komutu, bu gecikme yuvasına başka bir komut taşıyarak ve ble komutundaki annul bitini ayarlayarak değiştirilebilir (ble,a L1 olarak ifade edilir). Bu değişiklikten sonra programı .
- Şimdi iki gereksiz talimat var. Bunları kaldırın ve ortaya çıkan programı gösterin.

# 16

## BÖLÜM

### KOMUT DÜZEYİNDE PARALELLİK VE SÜPERSKALAR İŞLEMCİLER

#### 16.1 Genel Bakış

Superscalar ve Superpipelined  
Kısıtlamaları

#### 16.2 Tasarım Sorunları

Komut Düzeyinde Paralellik ve Makine Paralelliği Talimat Yayın  
Politikası  
Register Yeniden  
Adlandırma Makine  
Paralelliği Branch  
Tahmini Superscalar  
Yürütme  
Superscalar Uygulama

#### 16.3 Intel Çekirdek Mikro Mimarisi

Ön Uç  
Sıra Dışı Yürütme Mantığı  
Tamsayı ve Kayan Nokta Yürütme Birimleri

#### 16.4 Arm Cortex-A8

Komut Getirme Birimi  
Komut Kod Çözme Birimi  
Tamsayı Yürütme Birimi  
SIMD ve Kayan Noktalı İşlem Hattı

#### 16.5 ARM Cortex-M3

Branşlarla İlgilenen Boru  
Hattı Yapısı

#### 16.6 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler

### ÖĞRENİM HEDEFLERİ

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- |r Süperskalar ve süperpipelined yaklaşımalar arasındaki farkı açıklayınız.
- |r Komut düzeyinde paralelliği tanımlayın.
- |r Bağımlılıkları ve kaynak tartışmalarını komut düzeyinde paralelliğin sınırlamaları olarak tartışmak
- |r Komut düzeyinde paralellik ile ilgili tasarım konularına genel bir bakış sunmak.
- |r RISC makinelerinde ve süperskalar makinelerde boru hattı performansını iyileştirme tekniklerini karşılaştırmak ve karşılaştırmak.

Bir işlemci mimarisinin süperskalar uygulaması, ortak komutların -tamsayı ve kayan nokta aritmetiği, yüklemeler, depolamalar ve koşullu dallanmalar- aynı anda başlatılabilen ve bağımsız olarak yürütülebildiği bir uygulamadır. Bu tür uygulamalar, komut boru hattıyla ilgili bir dizi karmaşık tasarım sorununu gündeme getirir.

Süperskalar tasarım, RISC mimarisinin hemen ardından sahneye çıkmıştır. Bir RISC makinesinin basitleştirilmiş komut seti mimarisini süperskalar tekniklere kolayca uyum sağlasa da, süperskalar yaklaşım bir RISC ya da CISC mimarisinde kullanılabilir.

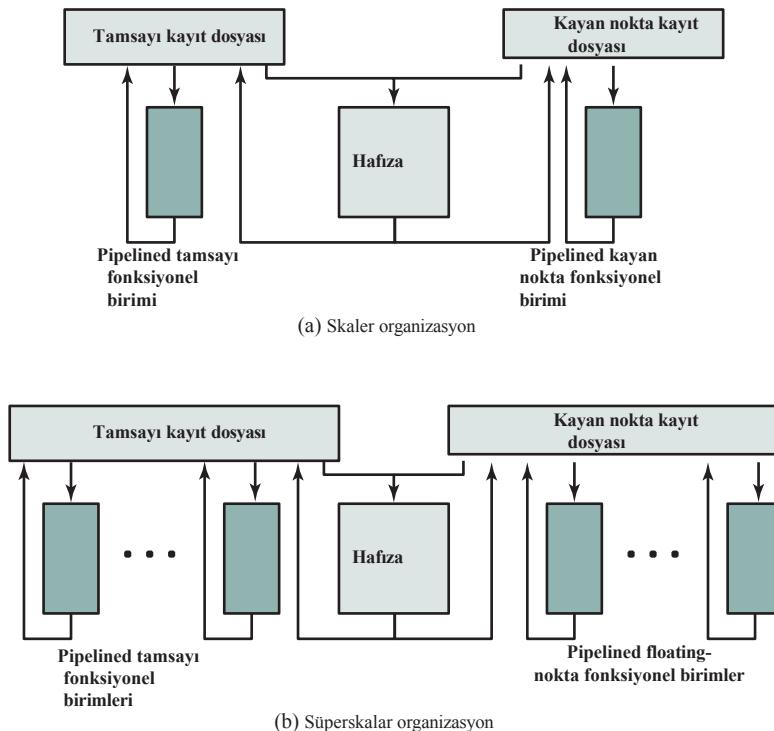
IBM 801 ve Berkeley RISC I ile gerçek RISC araştırmalarının başlangıcından itibaren ticari RISC makinelerinin gelişisi için gebelik süresi yedi ya da sekiz yıl iken, ilk süperskalar makineler **süperskalar** teriminin ortaya atılmasından sadece bir ya da iki yıl sonra ticari olarak kullanılabilir hale geldi. Süperskalar yaklaşımı artık yüksek performanslı mikroişlemcilerin uygulanmasında standart yöntem haline gelmiştir.

Bu bölümde, süperskalar yaklaşımı genel bir bakışla başlıyoruz ve bunu süperpipelining ile karşılaştırıyoruz. Daha sonra, süperskalar uygulama ile ilgili temel tasarım sorunlarını sunuyoruz. Daha sonra süperskalar mimarının birkaç önemli örneğine bakacağız.

### 16.1 GENEL BAKIŞ

İlk olarak 1987 yılında ortaya atılan *süper skalar* terimi [AGER87], skaler talimatların yürütülme performansını artırmak için tasarlanmış bir makineyi ifade eder. Çoğu uygulamada, işlemlerin büyük kısmı skaler büyüklükler üzerindedir. Buna göre, süperskalar yaklaşımı, yüksek performanslı genel amaçlı işlemcilerin evriminde bir sonraki adımı temsil etmektedir.

Superscalar yaklaşımının özü, talimatların farklı boru hatlarında bağımsız ve eşzamanlı olarak yürütülebilmesidir. Bu kavram, talimatların program sırasından farklı bir sirada yürütülmemesine izin verilerek daha da kullanılabilir. Şekil 16.1'de skaler ve süperskalar yaklaşımalar genel hatlarıyla karşılaştırılmaktadır. Geleneksel bir skaler organizasyonda, tamsayı işlemleri için tek bir boru hattı işlevsel birimi ve kayan nokta işlemleri için bir tane vardır. Paralellik, birden fazla talimatın boru hattının farklı aşamalarında yer olmasını sağlayarak elde edilir.



**Sekil 16.1** Sıradan Skaler Organizasyona Kiyasla Süper Skaler Organizasyon

tek seferde. Süperskalar organizasyonda, her biri bir boru hattı olarak uygulanan birden fazla işlevsel birim vardır. Her bir işlevsel birim, boru hattı yapısı sayesinde bir dereceye kadar paralellik sağlar. Birden fazla işlevsel birimin kullanılması, işlemcinin her bir boru hattı için bir akış olmak üzere talimat akışlarını paralel olarak yürütmesini sağlar. Paralel yürütmenin programın amacını ihlal etmediğinden emin olmak, derleyici ile birlikte donanımın sorumluluğundadır.

Birçok araştırmacı süperskalar benzeri işlemcileri araştırılmıştır ve araştırmaları bir dereceye kadar performans iyileştirmesinin mümkün olduğunu göstermektedir. Tablo 16.1'de bildirilen performans avantajları sunulmaktadır. Tablo 16.1'deki farklılıklar

**Tablo 16.1** Superscalar Benzeri Makinelerin Bildirilen Hızları

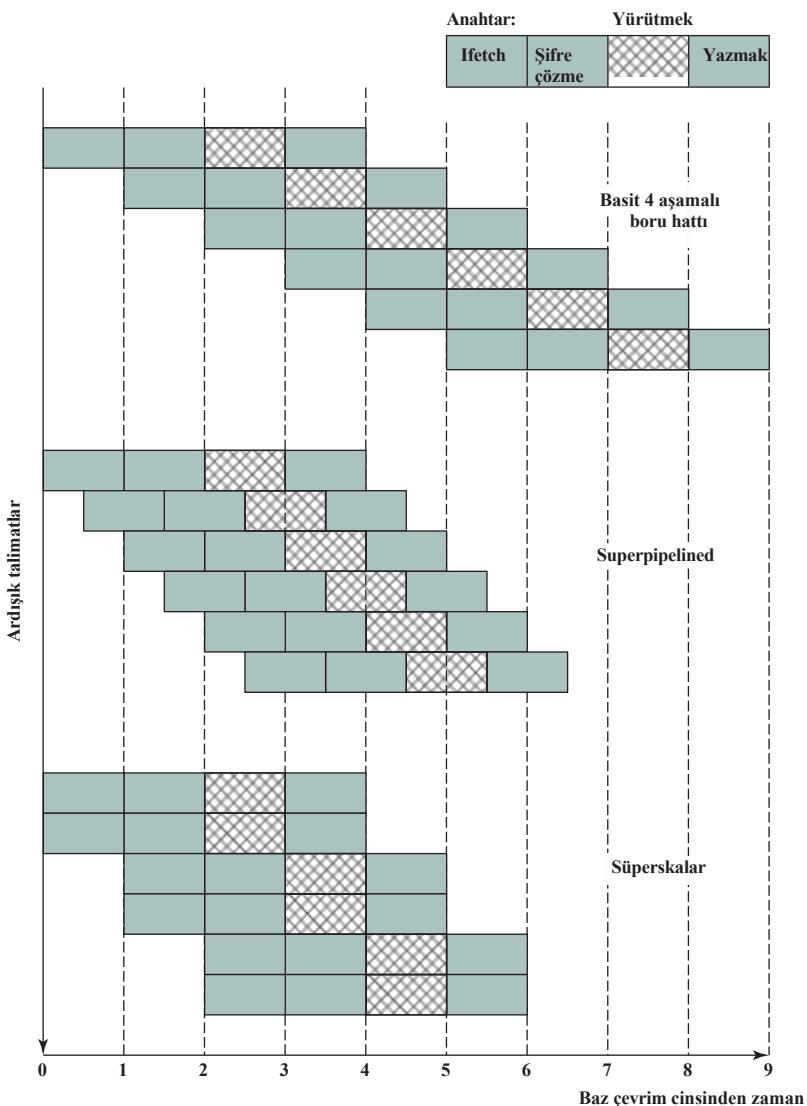
Referans	Hızlandırma
[TJAD70]	1.8
[KUCK77]	8
[WEIS84]	1.58
[ACOS86]	2.7
[SOHI90]	1.8
[SMIT89]	2.3
[JOUP89b]	2.2
[LEE91]	7

sonuçlar hem simüle edilen makinelerin donanımındaki hem de simüle edilen uygulamalardaki farklılıklardan kaynaklanmaktadır.

### Superscalar'a karşı Superpipelined

Daha yüksek performans elde etmek için alternatif bir yaklaşım, ilk olarak 1988 yılında ortaya atılan bir terim olan süper pipelining olarak adlandırılır [JOUN88]. Süper pipelining, birçok pipeline aşamasının yarım saat döngüsünden daha az gerektiren görevleri yerine getirdiği gereğinden yararlanır. Böylece, iki katına çıkarılmış bir dahili saat hızı, bir harici saat çevriminde iki görevin yerine getirilmesini sağlar. Bu yaklaşımın bir örneğini MIPS R4000'de görmüştük.

Şekil 16.2 iki yaklaşımı karşılaştırmaktadır. Diyagramın üst kısmı, karşılaştırma için temel olarak kullanılan sıradan bir boru hattını göstermektedir. Temel boru hattı sorunları



**Şekil 16.2** Superscalar ve Superpipeline Yaklaşımlarının Karşılaştırılması

saat döngüsü başına bir komut ve saat döngüsü başına bir boru hattı aşaması gerçekleştirilebilir. Boru hattının dört aşaması vardır: komut alma; işlem kod çözme; işlem yürütme; ve sonuç geri yazma. Yürütme aşaması netlik için çapraz çizilmiştir. Birkaç komut aynı anda yürütülmesine rağmen, herhangi bir zamanda yalnızca bir komutun yürütme aşamasında olduğuna dikkat edin.

Diyagramın bir sonraki kısmı, saat döngüsü başına iki boru hattı aşaması gerçekleştirilebilen bir **süper** boru hattı uygulamasını göstermektedir. Buna bakmanın alternatif bir yolu, her aşamada gerçekleştirilen işlevlerin birbirine örtüşmeyen iki parçaya bölünebileceği ve her birinin yarı saat döngüsünde yürütülebileceğidir. Bu şekilde davranış bir superpipeline uygulamasının 2. dereceden olduğu söylenir. Son olarak, diyagramın en alt kısmı, her bir aşamanın iki örneğini paralel olarak yürütüebilen bir süper skaler uygulamayı göstermektedir. Daha yüksek dereceli süper boru hattı ve süper skaler uygulamalar elbette mümkündür.

Şekil 16.2'de gösterilen süper boru hattı ve süper skalar uygulamalarının her ikisi de sabit durumda aynı anda çalışan aynı sayıda komuta sahiptir. Superpipelined işlemci, programın başlangıcında ve her dallanma hedefinde superscalar işlemcinin gerisinde kalmaktadır.

## Kısıtlamalar

Süperskalar yaklaşım, birden fazla talimatın paralel olarak yürütülebilmesine bağlıdır. **Komut düzeyinde paralellik** terimi, ortalama olarak bir programın komutlarının paralel olarak yürütülebilme derecesini ifade eder. Komut düzeyinde paralelligi en üst düzeye çıkarmak için derleyici tabanlı optimizasyon ve donanım tekniklerinin bir kombinasyonu kullanılabilir. Süper skaler makinelerde komut düzeyinde paralelligi artırmak için kullanılan tasarım tekniklerini incelemeden önce, sistemin başa çıkması gereken paralelliğe yönelik işlevsel sınırlamalara bakmamız gereklidir. [JOHN91] beş sınırlama listelemektedir:

- Gerçek veri bağımlılığı;
- Prosedürel bağımlılık;
- Kaynak çatışmaları;
- Çıktı bağımlılığı;
- Bağımlılık karşılılığı.

Bu bölümün geri kalanında bu sınırlamalardan ilk üçünü inceleyeceğiz. Son ikisinin tartışılması için bir sonraki bölümdeki bazı gelişmelerin beklenmesi gerekmektedir.

### **DOĞRU VERİ BAĞIMLILIĞI**

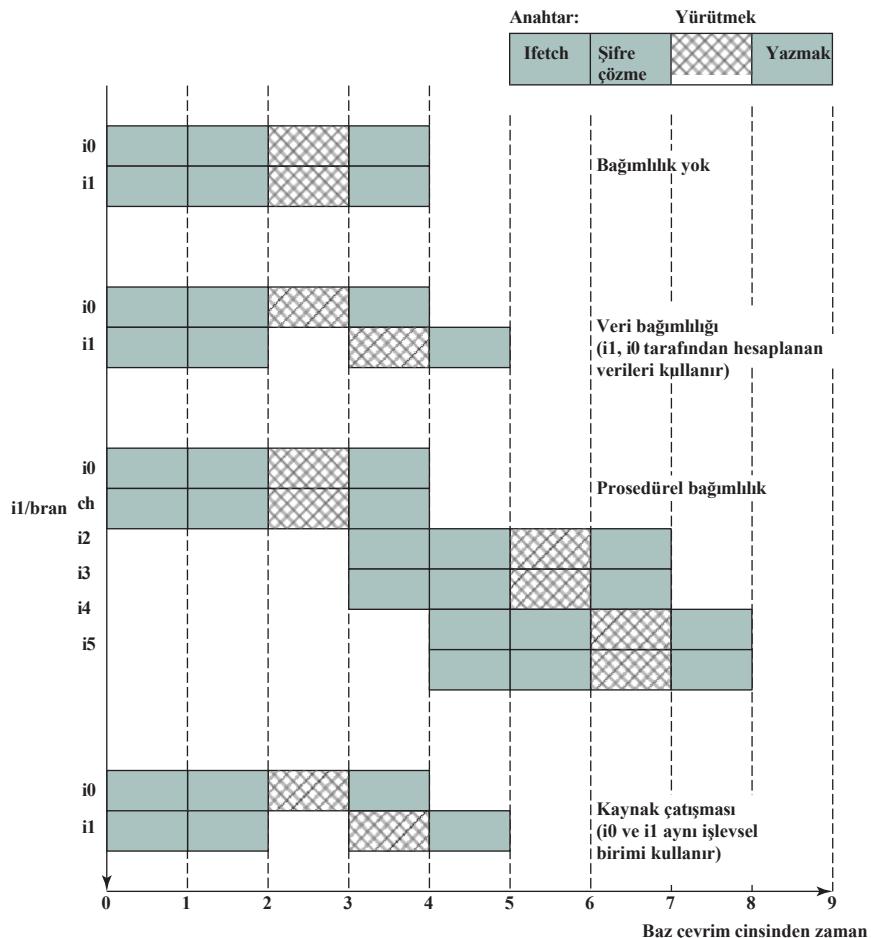
Aşağıdaki diziyi göz önünde bulundurun:<sup>1</sup>

```
ADD EAX, ECX ;EAX yazmacını konfigürasyon ile yükleyin
ECX'in çadırları artı içindekiler
EAX'ın
MOV EBX, EAX ;EBX'i EAX'ın içeriği ile yükleyin
```

İkinci komut alınabilir ve kodu çözülebilir ancak ilk komut kadar çalıştırılamaz. Bunun nedeni, ikinci komutun veriye ihtiyaç duymasıdır

---

<sup>1</sup>Intel x86 assembly dili için, noktalı virgül bir yorum alanını başlatır.



**Şekil 16.3** Bağımlılıkların Etkisi

ilk komut tarafından üretilir. Bu durum **gerçek veri bağımlılığı** olarak adlandırılır (**akış bağımlılığı** veya **yazmadan sonra okuma [RAW] bağımlılığı** olarak da adlandırılır).

- Şekil 16.3, bu **bağımlılığı** süper skaler dereceli bir makinede göstermektedir  
 2. Bağımlılık olmadığından, iki komut paralel alınabilir ve yürütülebilir. Birinci ve ikinci komutlar arasında bir veri bağımlılığı varsa, ikinci komut bağımlılığını ortadan kaldırırmak için gereken saat döngüsü kadar geciktirilir. Genel olarak, herhangi bir komut tüm giriş değerlerini üretilemeye kadar geciktirmeliidir.

Şekil 16.2'nin üst kısmında gösterildiği gibi basit bir boru hattında, yukarıda bahsedilen komut dizisi hiçbir gecikmeye neden olmaz. Ancak, yüklemelerden birinin bir yazmaç yerine bellekten yapıldığı aşağıdaki durumu düşünün:

```

MOV EAX, eff ;EAX yazmacını
eff etkin bellek adresinin içeriği
MOV EBX, EAX ;EBX'i EAX'ın içeriği ile yükleyin
  
```

Tipik bir RISC işlemcinin bellekten bir yüklemeyi gerçekleştirmesi, yük bir önbellek vuruşu olduğunda iki veya daha fazla döngü alır. Çip dışı bellek erişiminin gecikmesi nedeniyle tüm önbellek seviyelerinde bir önbellek kaçırma işlemi onlarca hatta yüzlerce döngü alabilir. Bu gecikmeyi telafi etmenin bir yolu, derleyicinin talimatları yeniden sıralamasıdır, böylece bellek yüküne bağlı olmayan bir veya daha fazla sonraki talimat boru hattından akmağa başlayabilir. Bu şema süperskalar boru hattı durumunda daha az etkilidir: Yükleme sırasında çalıştırılan bağımsız talimatlar muhtemelen ilk döngüsünde çalıştırılacak ve işlemciye yükleme tamamlanana kadar yapacak bir şey bırakmayacaktır.

**PROSEDÜREL BAĞIMLILIKLAR** Bölüm 14'te tartışıldığı gibi, bir komut dizisinde dallanmaların varlığı boru hattı işlemini karmaşıklığındır. Bir dallanmayı takip eden komutlar (dallanma ya da olmasın) dallanmaya **prosedürel** olarak **bağımlıdır** ve dallanma gerçekleşene kadar yürütülemez. Şekil 16.3, bir dallanmanın 2. dereceden bir süper skaler boru hattı üzerindeki etkisini göstermektedir.

Gördüğümüz gibi, bu tür prosedürel bağımlılık skaler bir boru hattını da etkilemektedir. Süperskalar bir boru hattı için sonuç daha ağırdır, çünkü her gecikme ile daha büyük bir fırsat kaybedilir.

Eğer değişken uzunlukta komutlar kullanılıyorsa, o zaman başka bir tür prosedürel bağımlılık ortaya çıkar. Herhangi bir komutun uzunluğu bilinmediğinden, bir sonraki komutun getirilebilmesi için en azından kısmen kodunun çözülmesi gereklidir. Bu da superscalar boru hattında gerekli olan eşzamanlı getirme işlemini engeller. Bu, süperskalar tekniklerin sabit komut uzunluğuna sahip bir RISC veya RISC benzeri mimariye daha kolay uygulanabilir olmasının nedenlerinden biridir.

**KAYNAK ÇATIŞMASI** Kaynak çatışması, iki veya daha fazla komutun aynı kaynak için aynı anda rekabet etmesidir. Kaynaklara örnek bellekler, önbellekler, veri yolları, kayıt dosyası bağlantı noktaları ve işlevsel birimler (örneğin, ALU toplayıcı) verilebilir. Boru hattı açısından, bir kaynak çatışması veri bağımlılığına benzer bir davranış sergiler (Şekil 16.3). Ancak bazı farklılıklar vardır. Birincisi, kaynak çatışmaları kaynakların çoğaltılmasıyla aşılabilirken, gerçek bir veri bağımlılığı ortadan kaldırılamaz. Ayrıca, bir işlemin tamamlanması uzun sürtüğünde, kaynak çatışmaları, uygun işlevsel birimin boru hattıyla bağlanmasıyla en aza indirilebilir.

## 16.2 TASARIM KONULARI

### Komut Seviyesi Paralelliği ve Makine Paralelliği

[JOP89a], birbirile ilişkili iki kavram olan komut seviyesi paralelliği ve makine paralelliği arasında önemli bir ayrim yapmaktadır. Komut **düzeyinde paralellik**, bir dizideki komutlar bağımsız olduğunda ve böylece üst üste bindirilerek paralel olarak yürütülebildiğiinde mevcuttur.

Komut düzeyinde paralellik kavramına bir örnek olarak aşağıdaki iki kod parçasını ele alalım [JOP89b]:

```

Yük R1 d R2           R3 d R3, "1" ekle
R3 d R3, "1" ekle R4 d R3, R2 ekle
R4 ekle d R4, R2 Sakla [R4] d R0

```

## 582 BÖLÜM 16 / KOMUT DÜZEYİ PARALELLİK VE SÜPERSKALAR İŞLEMÇİLER

Soldaki üç komut birbirinden bağımsızdır ve teorik olarak üçü de paralel olarak çalıştırılabilir. Buna karşın, sağdaki üç komut paralel olarak çalıştırılamaz çünkü ikinci komut sonucunu, üçüncü komut ise ikincinin sonucunu kullanır.

Komut düzeyinde paralelliğin derecesi, koddaki gerçek veri bağımlılıklarının ve prosedürel bağımlılıkların sıklığına göre belirlenir. Bu faktörler de komut kümesi mimarisine ve uygulamaya bağlıdır. Komut düzeyinde paralellik ayrıca [JOUP89a]'nın işlem gecikmesi olarak adlandırdığı şey tarafından da belirlenir: bir komutun sonucunun sonraki bir işlenen olarak kullanılabilmesi için geçen süre. Gecikme, bir veri veya prosedürel bağımlılığın ne kadar gecikmeye neden olacağını belirler.

**Makine paralelliği**, işlemcinin komut düzeyinde paralellikten yararlanma yeteneğinin bir ölçüsüdür. Makine paralelliği, aynı anda getirilebilen ve yürütülebilen talimatların sayısı (paralel boru hatlarının sayısı) ve işlemcinin bağımsız talimatları bulmak için kullandığı mekanizmaların hızı ve karmaşıklığı ile belirlenir.

Hem komut düzeyinde hem de makine paralelliği performansı artırmada önemli faktörlerdir. Bir program makine tam olarak yararlanmak için yeterli komut düzeyinde paralelliğe sahip olmayabilir. Bir RISC'de olduğu gibi sabit uzunlukta talimat seti mimarisinin kullanılması, komut düzeyinde paralelliği artırır. Öte yandan, sınırlı makine paralelliği, programın doğası olursa olsun performansı sınırlayacaktır.

### Talimat Düzenleme Politikası

Daha önce de belirtildiği gibi, makine paralelliği sadece her bir boru hattı aşamasının birden fazla örneğine sahip olma meselesi değildir. İşlemci aynı zamanda talimat düzeyinde paralelliği tanımlayabilmeli ve talimatların paralel olarak getirilmesini, kodlarının çözülmesini ve yürütülmesini düzenleyebilmelidir. [JOHN91], işlemcinin işlevsel birimlerinde komut yürütmemi başlatma sürecine atıfta bulunmak için komut verme terimini ve komutları vermek için kullanılan protokole atıfta bulunmak için **komut verme politikası** terimini kullanır. Genel olarak, talimatın boru hattının kod çözme aşamasından boru hattının ilk yürütme aşamasına geçtiğinde talimatın gerçekleştiğini söyleyebiliriz.

Özünde işlemci, boru hattına getirilebilecek ve yürütülebilecek talimatları bulmak için mevcut yürütme noktasının ilerisine bkmaya çalışmaktadır. Bu bağlamda üç tür sıralama önemlidir:

- Talimatların alınma sırası;
- Talimatların yürütülme sırası;
- Talimatların kayıt ve bellek konumlarının içeriğini güncelleme sırası.

İşlemci ne kadar sofistike olursa, bu sıralamalar arasındaki katı ilişkiye o kadar az bağlı olur. Çeşitli boru hattı unsurlarının kullanımını optimize etmek için, işlemcinin bu sıralamalarдан birini veya daha fazlasını katı bir sıralı yürütmede bulunacak sıralamaya göre değiştirmesi gerekecektir. İşlemci üzerindeki tek kısıtlama, sonucun doğru olması gereğidir. Bu nedenle, işlemci daha önce tartışılan çeşitli bağımlılıklar ve çatışmalara uyum sağlamalıdır.

Genel anlamda, superscalar komut politikalarını aşağıdaki kategorilerde grupperləbiliriz:

- Sipariş içi tamamlama ile ilgili sipariş içi sorun.
- Sipariş **dışı** tamamlama ile sipariş içi sorun.
- Sıra dışı tamamlama ile sıra dışı sorun.

**SIRA İÇİ TAMAMLAMA İLE SIRA İÇİ VERME** En basit komut verme politikası, komutları sıralı yürütme ile elde edilecek tam sırayla vermek (sıra **İç** verme) ve sonuçları aynı sırayla yazmaktr (sıra **İç** tamamlama). Skaler boru hatları bile bu kadar basit düşünen bir politika izlemez. Bununla birlikte, bu politikayı daha karmaşık yaklaşımları karşılaştırmak için bir temel olarak düşünmek faydalıdır.

Şekil 16.4a bu politikanın bir örneğini vermektedir. Bir iki komut getirip çözebilen, üç ayrı işlevsel birime (örneğin, iki tamsayı aritmetiği ve bir kayan nokta aritmetiği) sahip olan ve geri yazma boru hattı aşamasının iki örneğine sahip olan bir süperskalar boru hattı varsayıyoruz. Örnek, altı komutlu bir kod parçası üzerinde aşağıdaki kısıtlamaları varsayar:

- I1'in yürütülmesi için iki döngü gereklidir.
- I3 ve I4 aynı işlevsel birim için çakışır.
- G5, G4 tarafından üretilen değere bağlıdır.
- I5 ve I6 işlevsel bir birim için çakışır.

Talimatlar her seferinde iki tane getirilir ve kod çözme birimine aktarılır. Talimatlar çiftler halinde alındığından, sonraki iki talimatın kod çözme boru hattı aşamalarının temizlenene kadar beklemesi gereklidir. Sıralı **tamamlamayı** garanti etmek için, bir işlevsel birim için çıkışma olduğunda veya bir işlevsel birimin sonuç üretmesi için birden fazla döngü gerektiğiinde, talimatların verilmesi geçici olarak durur.

Bu örnekte, ilk komutun kodunun çözülmesinden son sonuçların yazılmasına kadar geçen süre sekiz döngüdür.

**SIRA DİŞİ TAMAMLAMA İLE SIRA İÇİ SORUNU** **Sıra dışı tamamlama**, skaler RISC işlemcilerde birden fazla döngü gerektiren komutların performansını artırmak için kullanılır. Şekil 16.4b'de bunun süperskalar bir kullanımı gösterilmektedir. I2 komutunun I1'den önce tamamlanmasına izin verilir. Bu, I3'ün daha erken tamamlanmasını sağlar ve net sonuç olarak bir döngü tasarruf edilir.

Sıra dışı tamamlama ile, tüm işlevsel birimlerdeki maksimum makine paralelliği derecesine kadar herhangi bir sayıda komut herhangi bir zamanda yürütme aşamasında olabilir. Talimatların verilmesi bir kaynak çakışması, veri bağımlılığı ya da prosedürel bağımlılık nedeniyle durur.

Yukarıda bahsedilen sınırlamalara ek olarak, daha önce **çıktı bağımlılığı (write after write [WAW] bağımlılığı** olarak da adlandırılır) olarak bahsettiğimiz yeni bir bağımlılık ortaya çıkar. Aşağıdaki kod parçası bu bağımlılığı göstermektedir (*op* herhangi bir işlemi temsil etmektedir):

I1: R3 <b>d</b>	R3 op R5
I2: R4 <b>d</b>	R3+ 1
I3: R3 <b>d</b>	R5+ 1
I4: R7 <b>d</b>	R3 op R4

## 584 BÖLÜM 16 / KOMUT DÜZEYİ PARALELLİK VE SÜPERSKALAR İŞLEMCİLER

Şifre çözme		Yürütmek		Yazmak		Döngü
I1	I2					1
I3	I4	I1	I2			2
I3	I4					3
	I4			I3		4
I5	I6			I4		5
	I6		I5			6
			I6			7
					I5	8
				I6		

(a) Sipariş içi düzenleme ve sipariş içi tamamlama

Şifre çözme		Yürütmek		Yazmak		Döngü
I1	I2					1
I3	I4	I1	I2			2
	I4	I1		I3		3
I5	I6			I4		4
	I6		I5			5
			I6			6
					I5	7
				I6		

(b) Sipariş içi düzenleme ve sipariş dışı tamamlama

Şifre çözme		Pencere	Yürütmek		Yazmak	Döngü
I1	I2		I1	I2		1
I3	I4	I1, I2				2
I5	I6	I3, I4	I1	I2		3
		I4, I5, I6		I3		4
		I5		I6	I4	5
				I5		6
					I5	7
					I6	

(c) Sipariş dışı düzenleme ve sipariş dışı tamamlama

**Şekil 16.4** Superscalar Komut Verme ve Tamamlama Politikaları

I2 komutu I1 komutundan önce yürütülemez, çünkü I1'de üretilen R3 yazmacındaki sonuca ihtiyaç duyar; bu, Bölüm 16.1'de açıklandığı gibi gerçek bir veri bağımlılığı örneğidir. Benzer şekilde I4, I3 tarafından üretilen bir sonucu kullandığı için I3'ü beklemelidir. Peki ya I1 ve I3 arasındaki ilişki? Burada tanımladığımız gibi bir veri bağımlılığı yoktur. Ancak, I3 I1'den önce tamamlanırsa, R3'ün içeriğinin yanlış değeri I4'ün yürütülmesi için getirilecektir. Sonuç olarak, doğru çıktı değerlerini üretmek için I3'ün I1'den sonra tamamlanması gereklidir. Bunu sağlamak için, üçüncü komutun verilmesi, sonucunun daha sonra tamamlanması daha uzun süren daha eski bir komut tarafından üzerine yazılabileceği durumlarda durdurulmalıdır.

Sıra dışı tamamlama, sıra içi tamamlamaya göre daha karmaşık komut verme mantığı gerektirir. Buna ek olarak, komut kesintileri ve istisnalarla başa çıkmak daha zordur. Bir kesme oluştuğunda, o anki komut yürütme

noktası daha sonra devam ettirmek üzere askıya alınır. İşlemci, yeniden başlatma işleminin, kesinti anında, kesintiye neden olan komuttan önceki komutların zaten tamamlanmış olabileceğini dikkate alması gerektiğini sağlamalıdır.

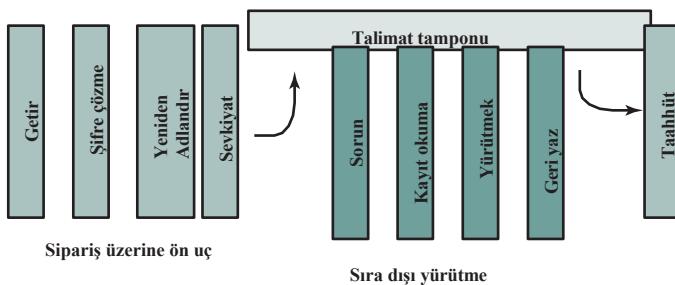
### SİPARİŞ **DİŞI** İLE SİPARİŞ **DİSİ SORUNU**

Sipariş içi ile sorununda, işlemci yalnızca bir bağımlılık veya çakışma noktasına kadar olan talimatların kodunu çözcektir. Çakışma çözülene kadar başka bir talimatın kodu çözülmmez. Sonuç olarak, işlemci çakışma noktasından sonra, hali hazırda boru hattında bulunanlardan bağımsız olabilecek ve boru hattına faydalı bir şekilde dahil edilebilecek sonraki talimatlara bakamaz.

**Sıra dışı soruna** izin vermek için, boru hattının kod çözme ve yürütme aşamalarını ayırmak gereklidir. Bu, **komut penceresi** olarak adlandırılan bir tampon ile yapılır. Bu organizasyonla, bir işlemci bir komutun kodunu çözmemeyi bitirdikten sonra, komut penceresine yerleştirilir. Bu tampon dolmadığı sürece, işlemci yeni talimatları almaya ve çözmeye devam edebilir. Bir işlevsel birim yürütme aşamasında kullanılabilir hale geldiğinde, komut penceresinden bir komut yürütme verilebilir. Herhangi bir komut, (1) mevcut olan belirli bir işlevsel birime ihtiyaç duyması ve (2) herhangi bir çakışma ya da bağımlılığın bu engellememesi koşuluyla verilebilir. Şekil 16.5 bu organizasyonu göstermektedir.

Bu organizasyonun sonucu, işlemcinin yürütme aşamasına getirebilecek bağımsız talimatları tanımlamasına izin veren bir ileriye bakma yeteneğine sahip olmasıdır. Komutlar, orijinal program sıraları çok az dikkate alınarak komut penceresinden verilir. Daha önce olduğu gibi, tek kısıtlama programının doğru şekilde yürütülmesidir.

Şekil 16.4c bu politikayı göstermektedir. İlk üç döngünün her birinde, kod çözme aşamasına iki talimat getirilir. Her döngü sırasında, tampon boyutu kısıtlamasına tabi olarak, iki komut kod çözme aşamasından komut penceresine geçer. Buörnekte, I6 komutunu I5'ten önce vermek mümkündür (I5'in I4'e bağlı olduğunu, ancak I6'nın bağlı olmadığını hatırlayın). Böylece, hem yürütme hem de geri yazma aşamalarında bir çevrim tasarruf edilir ve Şekil 16.4b ile karşılaştırıldığında uçtan uca tasarruf bir çevrimidir.



**Şekil 16.5** Sipariş Dışı Tamamlama ile Sipariş Dışı Sorun için Organizasyon

## 586 BÖLÜM 16 / KOMUT DÜZEYİ PARALELLİK VE SÜPERSKALAR İŞLEMCİLER

Komut penceresi, rolünü göstermek için Şekil 16.4c'de gösterilmiştir. Bununla birlikte, bu pencere ek bir boru hattı aşaması değildir. Bir komutun pencerede olması, işlemcinin o komut hakkında ne zaman verilebileceğine karar vermek için yeterli bilgiye sahip olduğu anlamına gelir.

Sıra dışı verme, sıra dışı tamamlama ilkesi daha önce açıklanan aynı kısıtlamalara tabidir. Bir komut, bir bağımlılığı veya çıkışmayı ihlal ederse verilemez. Aradaki fark, bir boru hattı aşamasının durma olasılığını azaltacak şekilde daha fazla komutun verilebilmesidir. Buna ek olarak, daha önce **antidependency** (**okuma sonrası yazma [WAR] bağımlılığı** olarak da adlandırılır) olarak adlandırdığımız yeni bir bağımlılık ortaya çıkar. Daha önce ele alınan kod parçası bu bağımlılığı göstermektedir:

```
I1: R3 d R3 op R5  
I2: R4 d R3+ 1 I3:  
    R3 d R5+ 1 I4: R7  
    d R3 op R4
```

I3 komutu, I2 komutu yürütülmeye başlamadan ve işlenenlerini almadan önce yürütmeye tamamlayamaz. Bunun nedeni I3'ün I2 için bir kaynak işlenen olan R3 yazmacını güncellememesidir. *Antidependency* terimi kullanılır çünkü kısıtlama gerçek bir veri bağımlılığına benzer, ancak tersine çevrilmiştir: İlk komutun ikinci komutun kullandığı bir değeri üretmesi yerine, ikinci komut ilk komutun kullandığı bir değeri yok eder.

**Yeniden Sıralama Tampon  
Simülatörü Tomasulo'nun Algoritma**



**Simülörü  
Tomasulo Algoritmasının Alternatif Simülasyonu**

Sıra dışı tamamlamayı desteklemek için kullanılan yaygın biri yeniden sıralama tamponudur. Yeniden sıralama tamponu, sıra tamamlanan ve daha sonra program sırasına göre kayıt dosyasına işlenen sonuçlar için geçici depolama alamıdır. İlgili bir kavram Tomasulo'nun algoritmasıdır. Ek N bu kavramları incelemektedir.

### Kayıt Yeniden Adlandırma

Sıra dışı komut verilmesine ve/veya sıra dışı komut tamamlanmasına izin verildiğinde, bunun WAW bağımlılıkları ve WAR bağımlılıkları olasılığına yol açtığını gördük. Bu bağımlılıklar, bir programdaki veri akışını ve yürütme sırasını yansitan RAW veri bağımlılıklarından ve kaynak çatışmalarından farklıdır. Öte yandan, WAW bağımlılıkları ve WAR bağımlılıkları, yazmaçlardaki değerlerin artık program akışı tarafından dikte edilen değer sırasını yansımaması nedeniyle ortaya çıkar.

Talimatlar sırayla verildiğinde ve sırayla tamamlandığında, yürütmenin her noktasında her bir yazmacın içeriğini belirtmek mümkündür. Sıra dışı teknikler kullanıldığında, yazmaçlardaki değerler zamanın her noktasında sadece dikte edilen talimatların sırası göz önünde bulundurularak tam olarak bilinemez

program tarafından. Gerçekte, değerler yazmaçların kullanımı için çalışma halindedir ve programcı bu çalışmaları zaman zaman bir boru hattı aşamasını durdurarak çözmelidir.

Karşıt bağımlılıklar ve çıktı bağımlılıklarının her ikisi de depolama kısıtlamalarına örnektir. Birden fazla komut aynı yazmaç konumlarını kullanmak için rekabet etmekte ve performansı geciktiren boru hattı kısıtlamaları oluşturmaktadır. Kayıt optimizasyon teknikleri kullanıldığında (Bölüm 15'te tartışıldığı gibi) sorun daha da ciddi hale gelir, çünkü bu derleyici teknikleri kayıtların kullanımını en üst düzeye çıkarmaya çalışır, dolayısıyla depolama çakışmalarının sayısını en üst düzeye çıkarır.

Bu tür depolama çatışmalarıyla başa çıkanın bir yöntemi, geleneksel bir kaynak çatışması çözümüne dayanır: kaynakların çoğaltıması. Bu bağlamda, bu teknik yazmaç **yeniden adlandırma** olarak adlandırılır. Temelde, yazmaçlar işlemci donanımı tarafından dinamik olarak tahsis edilir ve zaman içinde çeşitli noktalarda talimatlar tarafından ihtiyaç duyulan değerlerle ilişkilendirilir. Yeni bir yazmaç değeri oluşturulduğunda (yani, hedef operatör olarak bir yazmaç içeren bir komut yürütüldüğünde), bu değer için yeni bir yazmaç tahsis edilir. Bu yazmaçtaki kaynak işlenen olarak bu değere erişen sonraki talimatlar bir yeniden adlandırma sürecinden geçmelidir: bu talimatlardaki yazmaç referansları, gerekli değeri içeren yazmaca atıfta bulunacak şekilde revize edilmelidir. Bu nedenle, birkaç farklı talimatta aynı orijinal kayıt referansı, farklı değerler amaçlanmışsa, farklı gerçek kayıtlara atıfta bulunabilir.

İncelemekte olduğumuz kod parçası üzerinde register yeniden adlandırma'nın nasıl kullanılabileceğini ele alalım:

```
I1: R3b d R3a op R5a
I2: R4b d R3b+ 1 I3:
    R3c d R5a+ 1
I4: R7b d R3c op R4b
```

Alt simge içermeyen kayıt referansı, komutta bulunan mantıksal kayıt referansını ifade eder. Alt simge içeren kayıt referansı, yeni bir değeri tutmak için tahsis edilen bir donanım kaydına atıfta bulunur. Belirli bir mantıksal kayıt için yeni bir tahsis yapıldığında, kaynak operand olarak bu mantıksal kayda yapılan sonraki komut referansları, en son tahsis edilen donanım kaydına (program talimat dizisi açısından en son) atıfta bulunacak şekilde yapılır.

Bu örnekte, I3 komuttunda R3<sub>c</sub> kaydının oluşturulması, ikinci komuttaki WAR bağımlılığını ve ilk komuttaki WAW bağımlılığını örler ve I4 tarafından erişilen doğru değere müdahale etmez. Sonuç olarak I3 hemen verilebilir; yeniden adlandırma olmadan, ilk komut tamamlanana ve ikinci komut verilene kadar I3 verilemez.



#### Scoreboarding Simülörü

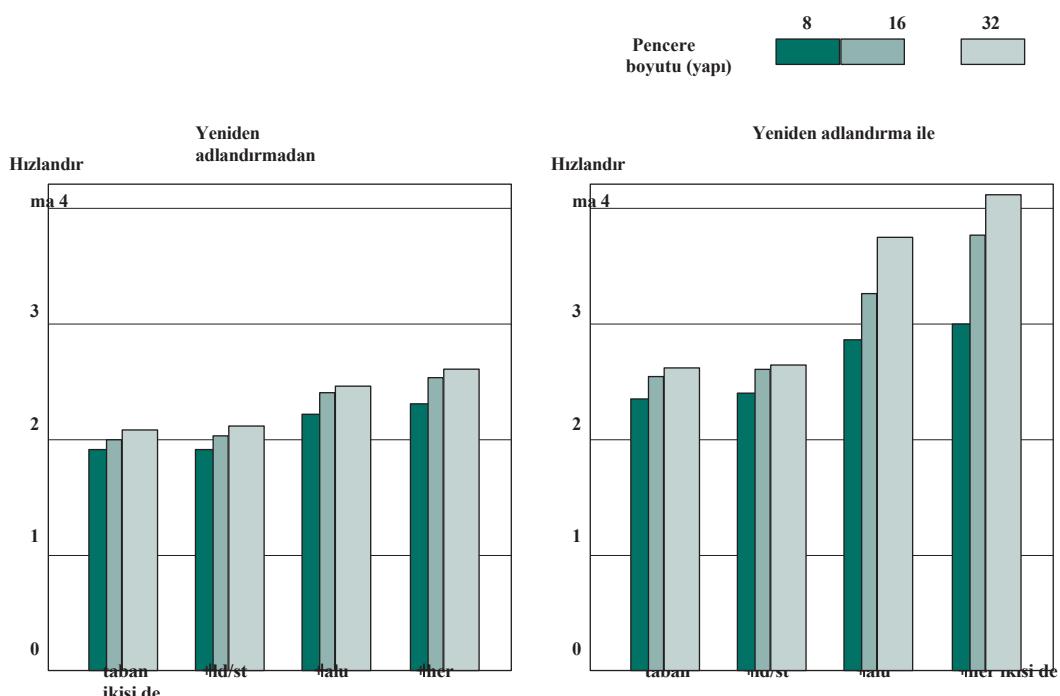
Kayıtların yeniden adlandırılmasına bir alternatif de skor tahtasıdır. Özünde, skor tahtası, önceki talimatlara bağımlı olmadıklarında ve yapısal tehlikeler mevcut olmadığındada talimatların yürütülmesine izin veren bir defter tutma tekniğidir. Bir tartışma için Ek N'ye bakınız.

## Makine Paralelliği

Önceki tartışmadı, performansı artırmak için bir süperskalar işlemcide kullanılabilecek üç donanım tekniğini inceledik: kaynakların çoğaltılması, sıra dışı sorun ve yeniden adlandırma. Bu teknikler arasındaki ilişkiyi aydınlatan bir çalışma [SMIT89]'da rapor edilmiştir. Çalışmada, çeşitli süperskalar özelliklerle güçlendirilmiş MIPS R2000'in sahip bir makineyi modifiye eden bir simülasyon kullanılmıştır. Bir dizi farklı program dizisi simüle edilmiştir.

**Şekil 16.6** sonuçları göstermektedir. Grafiklerin her birinde dikey eksen, skaler makineye kıyasla süper skaler makinenin ortalama hızlanmasıne karşılık gelmektedir. Yatay eksen dört alternatif işlemci organizasyonu için sonuçları göstermektedir. Temel makine işlevsel birimlerin hiçbirini çoğaltmaz, ancak talimatları sıra dışı verebilir. İkinci yapılandırma, bir veri önbelleğine erişen yükleme/depolama işlevsel birimini çoğaltır. Üçüncü yapılandırma ALU'yu çoğaltır ve dördüncü yapılandırma hem yükleme/depolama hem de ALU'yu çoğaltır. Her bir grafikte sonuçlar 8, 16 ve 32 komutluk komut penceresi boyutları için gösterilmiştir, bu da işlemcinin yapabileceği lookahead miktarını belirler. İki grafik arasındaki fark, ikincisinde yazmaçların yeniden adlandırılmasına izin verilmesidir. Bu, ilk grafiğin tüm bağımlılıklarla sınırlı bir makineyi yansıttığını, ikinci grafiğin ise yalnızca gerçek bağımlılıklarla sınırlı bir makineye karşılık geldiğini söylemekle eşdeğerdir.

İki grafik birleştirildiğinde bazı önemli sonuçlar ortaya çıkmaktadır. Birincisi, kayıtları yeniden adlandırmadan işlevsel birimler eklemenin muhtemelen faydalı olmayacağıdır. Orada



**Şekil 16.6** Prosedürel Bağımlılıklar Olmadan Çeşitli Makine Organizasyonlarının Hızlandırmaları

performansta hafif bir iyileşme sağlanır, ancak bunun bedeli donanım karmaşıklığının artmasıdır. Karşıt bağımlılıkları ve dışa bağımlılıkları ortadan kaldırın kayıtların yeniden adlandırılmasıyla, daha fazla işlevsel birim eklenerken gözle görülür kazançlar elde edilir. Bunun birlikte, 8'lik bir komut penceresi ile daha büyük bir komut penceresi kullanmak arasında elde edilebilecek kazanç miktarında önemli bir fark olduğunu unutmayın. Bu, komut penceresinin çok küçük olması durumunda veri bağımlılıklarının ekstra işlevsel birimlerin etkin kullanımını engelleyeceğini göstermektedir; işlemcinin donanımı daha kullanabilmesi için bağımsız talimatlar bulmak üzere oldukça ileriye bakılmasına gerekmektedir.



### Statik ve Dinamik Çizelgeleme ile Boru Hattı-Simülatör

## Şube Tahmini

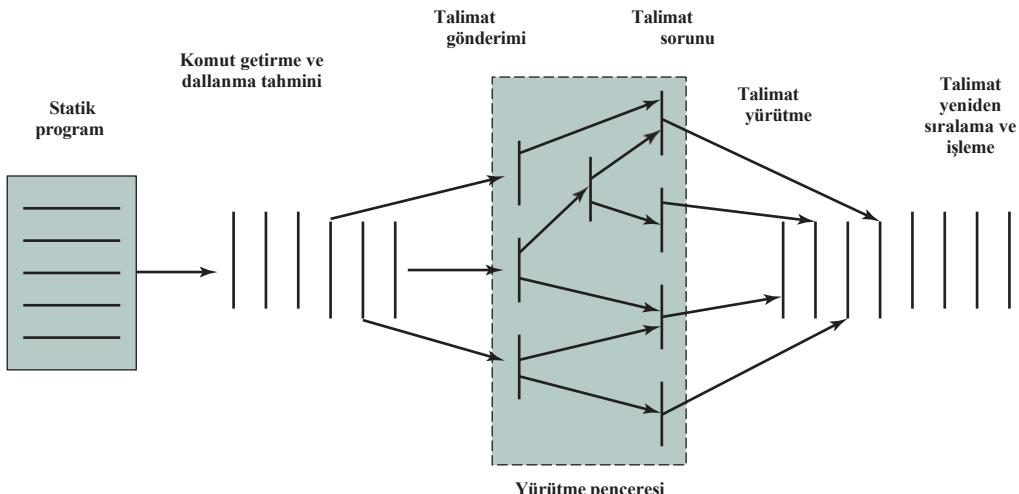
Herhangi bir yüksek performanslı pipelined makine dallanmalarla başa çıkma sorununu ele almıştır. Örneğin, Intel 80486 bu sorunu hem dallanmadan sonraki sıralı komutu getirerek hem de spekulatif olarak dallanma tar- get komutunu getirerek ele almıştır. Ancak, ön-getirme ve yürütme arasında iki boru hattı aşaması olduğundan, bu strateji dallanma iki çevrimlik bir gecikmeye neden olur.

RISC makinelerinin ortaya çıkmasıyla birlikte gecikmeli dallanma stratejisi keşfedilmiştir. Bu, işlemcinin koşullu dallanma komutlarının sonucunu, kullanılamayan komutlar öncelenmeden önce hesaplamasına olanak tanır. Bu yöntemle, işlemci her zaman hemen ardından gelen tek komutu çalıştırır. Bu, işlemci yeni bir komut akışı getirirken boru hattını dolu tutar.

Superscalar makinelerin gelişmesiyle birlikte, gecikmeli dallanma stratejisi daha az cazip hale gelmiştir. Bunun nedeni, gecikme aralığında birden fazla komutun çalıştırılması gereklmesi ve komut bağımlılıklarıyla ilgili çeşitli sorunların ortaya çıkmasıdır. Bu nedenle, süperskalar makineler **dallanma tahmininde** RISC öncesi tekniklere geri dönmüştür. PowerPC 601 gibi bazlıları basit bir statik dallanma tahmin teknigi kullanmaktadır. PowerPC 620 ve Pentium 4 gibi daha sofistikte işlemciler ise dallanma geçmişi analizine dayalı dinamik dallanma tahmini kullanmaktadır.

## Superscalar Yürütme

Artık programların superscalar yürütülmeye genel bir bakış sunabilecek durumdayız; bu Şekil 16.7'de gösterilmiştir. Yürüttülecek program doğrusal bir talimat dizisinden oluşur. Bu, programcı tarafından yazılan ya da derleyici tarafından oluşturulan statik programdır. Dallanma tahminini de içeren komut getirme aşaması, dinamik bir komut akışı oluşturmak için kullanılır. Bu akış bağımlılıklar açısından incelenir ve işlemci yapay bağımlılıkları kaldırabilir. İşlemci daha sonra talimatları bir yürütme penceresine gönderir. Bu pencerede, talimatlar artık sıralı bir akış oluşturmaz ancak gerçek veri bağımlılıklarına göre yapılandırılır. İşlemci her bir talimatı gerçek veri bağımlılıkları ve donanım kaynaklarının kullanılabilirliği tarafından belirlenen bir sırada yürütür. Son olarak, talimatlar kavramsal olarak tekrar sıralı hale getirilir ve sonuçları kaydedilir.



**Şekil 16.7** Superscalar İşlemenin Kavramsal Tasviri

Bir önceki paragrafta bahsedilen son adım, talimatın işlenmesi veya **emekliye ayrılması** olarak adlandırılır. Bu adım aşağıdaki nedenden dolayı gereklidir. Paralel, çoklu boru hatlarının kullanımı nedeniyle, komutlar statik programda gösterilenden farklı bir sırada tamamlanabilir. Ayrıca, dallanma tahmini ve spekülatif yürütme kullanımı, bazı talimatların yürütmemeyi tamamlayabileceği ve daha sonra temsil ettikleri dallanma alınmadığı için terk edilmesi gerektiği anlamına gelir. Bu nedenle, kalıcı depolama ve program tarafından görülebilen kayıtlar, komutlar yürütmemeyi tamamladığında hemen güncellenmez. Sonuçlar, bağımlı komutlar tarafından kullanılabilen bir tür geçici depoda tutulmalı ve ardından sıralı modelin komutu yürüteceği belirlendiğinde kalıcı hale getirilmelidir.

### Superscalar Uygulama

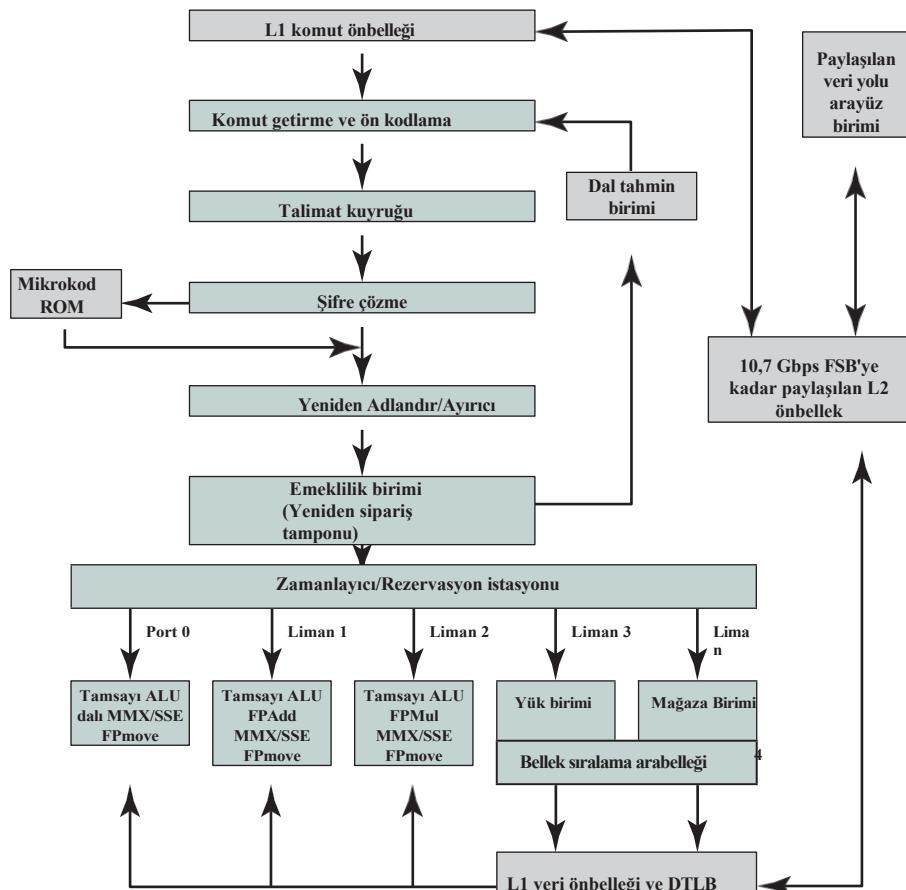
Şimdiki kadarki tartışmalarımıza dayanarak, süper skaler yaklaşım için gerekli olan prodüktör donanımı hakkında bazı genel yorumlar yapabiliriz. [SMIT95] aşağıdaki temel unsurları listelemektedir:

- Genellikle koşullu dallanma talimatlarının sonuçlarını tahmin ederek ve ötesinde getirecek birden fazla talimatı aynı anda getiren talimat getirme stratejileri. Bu işlevler, çoklu boru hattı getirme ve kod çözme aşamalarının ve dallanma tahmin mantığının kullanılmasını gerektirir.
- Kayıt değerlerini içeren gerçek bağımlılıkları belirlemek için mantık ve bu değerleri yürütme sırasında ihtiyaç duyulan yerlere iletmek için mekanizmalar.
- Paralel olarak birden fazla talimatın başlatılması veya verilmesi için mekanizmalar.
- Birden fazla pipelined fonksiyonel birim ve birden fazla bellek referansına aynı anda hizmet verebilen bellek hiyerarşileri dahil olmak üzere birden fazla talimatın paralel yürütülmesi için kaynaklar.
- Süreç durumunun doğru sırada işlenmesi için mekanizmalar.

## 16.3 INTEL ÇEKİRDEK MİKRO MİMARİSİ

Süperskalar tasarım kavramı genellikle RISC mimarisi ile ilişkilendirilse de, aynı süperskalar ilkeler bir CISC makinesine de uygulanabilir. Belki de bunun en önemli örneği Intel x86 mimarisidir. Intel serisinde süperskalar kavramlarının not etmek ilginçtir. 386 geleneksel bir CISC pipelined olmayan .486, tamsayı işlemlerinin ortalaması gecikme süresini iki ila dört döngüden bir döngüye düşüren ilk pipelined x86 işlemciyi tanıttı, ancak yine de her döngüde tek bir komut yürütütmekle sınırlı kaldı ve süperskalar öğeler içermedi. Orijinal Pentium, iki ayrı tamsayı yürütütme biriminin kullanımından oluşan mütevazı bir süperskalar bileşene sahipti. Pentium Pro, sıra dışı yürütütme ile tam gelişmiş bir süperskalar tasarımmını tamtti. Sonraki x86 modelleri süperskalar tasarımmını rafine etmiş ve geliştirmiştir.

Şekil 16.8 x86 boru hattı mimarisinin mevcut versiyonunu göstermektedir. Intel bir boru hattı mimarisini *mikro mimari* olarak adlandırır. Mikro mimari



**Sekil 16.8** Intel Çekirdek Mikro Mimarisi

## 592 BÖLÜM 16 / KOMUT DÜZEYİ PARALELLİK VE SÜPERSKALAR İŞLEMCİLER

makinenin komut seti mimarisinin temelini oluşturur ve uygular. Bu mikro mimari Intel Core Mikro Mimari olarak adlandırılır. Intel Core 2 ve Intel Xeon işlemci ailelerindeki her işlemci çekirdeğinde uygulanmaktadır. Ayrıca bir de Geliştirilmiş Intel Çekirdek Mikro Mimarisi vardır. İki mikro mimari arasındaki önemli bir fark, Geliştirilmiş Intel Çekirdek Mikro Mimarisinin üçüncü bir önbellek düzeyi sağlamasıdır.

Tablo 16.2'de önbellek mimarisinin bazı parametreleri ve performans özellikleri gösterilmektedir. Tüm önbellekler bir geri yazma güncelleme politikası kullanır. Bir komut bir bellek konumundan veri okuduğunda, işlemci bu veriyi içeren önbellek satırını önbelleklerde ve ana bellekte aşağıdaki sırayla arar:

- 1.** Başlatan çekirdeğin L1 veri önbelleği
- 2.** Diğer çekirdeklerin L1 veri önbelleği ve L2 önbelleği
- 3.** Sistem belleği

Önbellek satırı, L2 önbellekteki önbellek satırı kullanılabilirliği veya durumu göz ardı edilerek, yalnızca değiştirilirse başka bir çekirdeğin L1 veri önbelleğinden alınır. Tablo 16.2b

**Tablo 16.2** Intel Core Mikro Mimarisi Dayalı İşlemcilerin Önbellek/Bellek Parametreleri ve Performansı

(a) Önbellek Parametreleri				
Önbellek Seviyesi	Kapasite	İlişkisellik (yollar)	Satır Boyutu (bayt)	Geri Yazma Güncelleme Politikası
L1 verileri	32 kB	8	64	Geri Yazma
L1 talimatı	32 kB	8	N/A	N/A
L2 (paylaşılan) <sup>1</sup>	2, 4 MB	8 veya 16	64	Geri Yazma
L2 (paylaşılan) <sup>2</sup>	3, 6 MB	12 veya 24	64	Geri Yazma
L3 (paylaşılan) <sup>2</sup>	8, 12, 16 MB	15	64	Geri Yazma

*Notlar:*

1. Intel Çekirdek Mikro Mimarisi
2. Geliştirilmiş Intel Core Mikro Mimarisi

(b) Yük/Depo Performansı				
Veri Yerelligi	Yük		Mağaza	
	Gecikme	Verim	Gecikme	Verim
L1 veri önbelleği	3 saat döngüsü	1 saat döngüsü	2 saat döngüsü	3 saat döngüsü
Diger çekirdeğin L1 veri önbelleği değiştirilmiş durumda	14 saat + 5,5 veri yolu döngüsü	14 saat + 5,5 veri yolu döngüsü	14 saat + 5,5 veri yolu döngüsü	N/A
L2 önbellek	14	3	14	3
Hafiza	14 saat döngüleri+ 5,5 veri yolu döngüleri+ bellek gecikmesi	Veri yolu okuma protokolüne bağlıdır	14 saat döngüleri+ 5,5 veri yolu döngüleri+ bellek gecikmesi	Veri yolu okuma protokolüne bağlıdır

bellek kümesinden farklı yerelliklerin ilk dört baytını getirme özelliklerini göstermektedir. Gecikme sütunu erişim gecikmesinin bir tahminini sağlar. Ancak, gerçek gecikme süresi önbellek yüküne, bellek bileşenlerine ve bunların parametrelerine bağlı olarak değişebilir.

Intel Core mikro mimarisinin boru hattı şunları içerir:

- Talimat akışlarını bellekten alan ve dört talimat kod çözücü ile kod çözülmüş talimatları sıra dışı yürütme çekirdeğine besleyen bir sıra içi sorun ön ucu. Her komut, **mikro-işlemler** veya **mikro-ops** olarak bilinen bir veya daha fazla sabit uzunluklu RISC komutuna çevrilir.
- Döngü başına altı adede kadar mikro işlem yapabilen ve kaynaklar hazır olduğunda ve yürütme kaynakları mevcut olduğunda yürütmek için mikro işlemleri yeniden sıralayabilecek sıra dışı bir süper skaler yürütme çekirdeği.
- Mikro işlemlerin yürütülmesinin sonuçlarının işlenmesini ve mimari durumların ve işlemcinin kayıt setinin orijinal programmasına göre güncellenmesini sağlayan sıralı bir emeklilik birimi.

Aslında, Intel Core Mikro mimarisi bir RISC mikro mimarisi üzerinde bir CISC komut seti mimarisi uygular. İç RISC mikro-işlemleri en az 14 aşamalı bir boru hattından geçer; bazı durumlarda mikro-işlem birden fazla yürütme aşaması gerektirir ve bu da daha uzun bir boru hattına neden olur. Bu, daha önceki Intel x86 işlemcilerde ve Pentium'da kullanılan beş aşamalı işlem hattı (Şekil 14.21) ile tezat oluşturmaktadır.

## Ön Uç

Ön uç, kodu çözülmüş talimatları (mikro-ops) sağlamalı ve akışı altı sayı genişliğinde sıra dışı bir motora sürdürmelidir. Üç ana bileşenden oluşur: dal tahmin birimi (BPU), talimat getirme ve ön kodlama birimi ve talimat kuyruğu ve kod çözme birimi.

**BRANCH PREDiction UNIT** Bu birim, komut getirme biriminin çeşitli dallanma türlerini tahmin ederek yürütülecek en olası komutu getirmesine yardımcı olur: koşullu, dolaylı, doğrudan, çağrı ve dönüş. BPU her dallanma türü için özel donanım kullanır. Dallanma tahmini, işlemcinin dallanma sonucuna karar verilmeden çok önce talimatları yürütmeye başlamasını sağlar.

Mikro mimarı, son dallanma talimatlarının yürütülme geçmişine dayanan dinamik dallanma tahmin stratejisi kullanır. Son zamanlarda karşılaşılan dallanma talimatları hakkındaki bilgileri önbelleğe alan bir dallanma hedef tamponu (BTB) tutulur. Komut akışında bir dallanma komutuyla karşılaşıldığında BTB kontrol edilir. BTB'de halihazırda bir girdi mevcutsa, komut birimi dallanmanın yapılmış yapılmayacağını tahmin ederken bu geçmiş bilgileri tarafından yönlendirilir. Bir dallanma öngörülürse, bu girdiyle ilişkilendirilen dallanma hedef adresi, dallanma hedef komutunu ön-getirmek için kullanılır.

Komut yürütüldüğünde, ilgili girdinin geçmiş kısmı dallanma komutunun sonucunu yansıtacak şekilde güncellenir. Bu komut BTB'de temsil edilmiyorsa, bu komutun adresi BTB'deki bir girdiye yüklenir; gereklirse eski bir girdi silinir.

Önceki iki paragrafta yapılan açıklama, genel anlamda, orijinal Pentium modelinde kullanılan dal tahmin stratejisine ve daha sonraki

## 594 BÖLÜM 16 / KOMUT DÜZEYİ PARALELLİK VE SÜPERSKALAR İŞLEMÇİLER

Mevcut Intel modelleri de dahil olmak üzere Pentium modelleri. Bununla birlikte, Pentium söz konusu olduğunda, nispeten basit bir 2-bit geçmiş şeması kullanılır. Daha sonraki modeller çok daha uzun boru hatlarına sahiptir (Pentium için 5 aşamaya Intel Core Mikro mimarisi için 14 aşama) ve bu nedenle yanlış tahmin için ceza daha büyütür. Buna göre, sonraki modeller yanlış tahmin oranını azaltmak için daha fazla geçmiş biti içeren daha ayrıntılı bir dal tahmin şeması kullanır.

BTB'de geçmiş olmayan koşullu dallar, aşağıdaki kurallara göre statik bir tahmin algoritması kullanılarak tahmin edilir:

- Komut işaretçisi (IP) göreli olmayan dallanma adresleri için, dallanma bir dönüş ise tahmin alınır, aksi takdirde alınmaz.
- IP'ye bağlı geriye dönük koşullu dallar için tahmin alınır. Bu kural, döngülerin tipik davranışını yansıtır.
- IP'ye bağlı ileri koşullu dallar için tahmin alınmaz.

### **INSTRUCTION FETcH AND PREDECODE UNIT**

Komut getirme birimi, komut çeviri ara belleği (ITLB), bir komut ön bellekleyicisi, komut ön belleği ve ön kod mantığından oluşur.

Talimat getirme işlemi L1 talimat önbelleğinden gerçekleştirilir. Bir L1 önbelleği kaçırlığında, sıralı ön üç L2 önbelleğinden L1 önbelleğine her seferinde 64 bayt yeni yönerge besler. Varsayılan olarak, talimatlar sırayla getirilir, böylece her L2 önbellek satırı getirme işlemi bir sonraki getirilecek talimatı içerir. Dal tahmin birimi aracılığıyla dal tahmini, bu sıralı getirme işlemini değiştirebilir. ITLB, kendisine verilen doğrusal IP adresini L2 önbelleğe erişmek için gereken fiziksel adreslere çevirir. Ön uçtaki statik dallanma tahmini, hangi komutların daha sonra getirileceğini belirlemek için kullanılır.

Ön kodlama birimi, komut önbelleğinden veya ön besleme tamponlarından on altı baytı kabul eder ve aşağıdaki görevleri yerine getirir:

- Talimatların uzunluğunu belirleyin.
- Talimatlarla ilişkili tüm öneklerin kodunu çözün.
- Kod çözümcüler için talimatların çeşitli özelliklerini işaretleyin (örneğin, "is branch").

Ön kodlama birimi, komut kuyruğuna döngü başına en fazla altı komut yazabilir. Bir getirme işlemi altıdan fazla komut içeriyorsa, ön kodlayıcı, getirme işlemindeki tüm komutlar komut kuyruğuna yazılına kadar döngü başına altı komuta kadar kod çözmeye devam eder. Sonraki getirmeler ancak mevcut getirme tamamlandıktan sonra ön kodlamaya girebilir.

**INSTRUCTION QUEUE AND DECODE UNIT** Alınan talimatlar bir talimat kuyruğuna yerleştirilir. Buradan, kod çözme birimi komut sınırlarını belirlemek için baytları tarar; x86 komutlarının değişken uzunluğu nedeniyle bu gerekli bir işlemidir. Kod çözücü her bir makine komutunu, her biri 118 bitlik bir RISC komutu olan bir ila dört mikro-op'a çevirir. Karşılaştırma için çoğu saf RISC makinesinin sadece 32 bitlik bir komut uzunluğuna sahip olduğunu unutmam. Daha uzun mikro-op uzunluğu, daha karmaşık x86 komutlarını barındırmak için gereklidir. Bununla birlikte, mikro-op'ların yönetimi, türetildikleri orijinal talimatlardan daha kolaydır.

Birkaç talimat dörtten fazla mikro-ops gerektirir. Bu talimatlar, karmaşık bir makine talimatıyla ilişkili mikro işlem serilerini (beş veya daha fazla) içeren mikro kod ROM'una aktarılır. Örneğin, bir dize komutu çok büyük (hatta yüzlerce), tekrarlayan bir mikro-ops dizisine dönüştürilebilir. Dolayısıyla, mikro kod ROM'u Altıncı Bölümde tartışılan anlamda mikro programlanmış bir kontrol ünitesidir. Ortaya çıkan mikro-op dizisi yeniden adlandırma/ayırma modülüne gönderilir.

## Sıra Dışı Yürütmeye Mantığı

İşlemcinin bu kısmı, mikro-ops'ları, giriş operandları hazır olduğunda en kısa sürede yürütülmelerini sağlamak için yeniden sıralar.

**ALLOCATE** Allocate aşaması yürütme için gerekli kaynakları tahsis eder. Aşağıdaki işlevleri yerine getirir:

- Bir saat döngüsü sırasında tahsisatçıya gelen üç mikro-ops'tan biri için yazmaç gibi gerekli bir kaynak mevcut değilse, tahsisatçı boru hattını durdurur.
- Tahsis edici, herhangi bir zamanda işlemde olabilecek 126 mikro-islemlen bininin tamamlanma durumunu izleyen bir yeniden sıralama tamponu (ROB) giriş'i tahsis eder.<sup>2</sup>
- Tahsis edici, mikro işlemin sonuç veri değeri için 128 tamsayı veya kayan noktalı kayıt girişinden birini ve muhtemelen makine ardışık düzeneındaki 48 yükleme veya 24 depolamadan birini izlemek için kullanılan bir yükleme veya depolama tamponunu tahsis eder.
- Tahsis edici, komut zamanlayıcılarının iki mikro-op kuyruğundan birine bir giriş tahsis eder.

ROB, 126 mikro-ops'a kadar tutabilen dairesel bir tampondur ve 128 donanım kaydını da içerir. Her tampon giriş'i aşağıdaki alanlardan oluşur:

- **Durum:** Bu mikro-op'un yürütme için planlanıp planlanmadığını, yürütme için devre dışı bırakılıp bırakılmadığını veya yürütmemeyi tamamlayıp tamamlamadığını ve emekliye ayrılmaya hazır olup olmadığını gösterir.
- **Bellek Adresi:** Mikro-op'u oluşturan Pentium komutunun adresi.
- **Mikro-op:** Gerçek operasyon.
- **Takma Ad Kaydı:** Eğer mikro-op 16 mimari kayıtten birine referans veriyorsa, bu giriş bu referansı 128 donanım kayıtlarından birine yönlendirir.

Mikro-oplar ROB'a sırayla girer. Mikro operasyonlar daha sonra ROB'dan Sevk/Çalıştırma birimine sırasız olarak gönderilir. Gönderme kriteri, uygun yürütme biriminin ve bu mikro işlem için gereken tüm gerekli veri öğelerinin mevcut olmasıdır. Son olarak, mikro-islemler ROB'dan sırayla emekli edilir. Sırayla emekliye ayırma işlemini gerçekleştirmek için, her bir mikro işlem emekliye ayrılmaya hazır olarak belirlendikten sonra en eski mikro işlemler emekliye ayrılır.

**KAYITLARIN YENİDEN ADLANDIRILMASI** Yeniden adlandırma aşaması, 16 mimari kayıtlara (8 kayan noktalı kayıt, artı EAX, EBX, ECX, EDX, ESI, EDI, EBP ve ESP) referansları 128 fiziksel kayıt kümeseine yeniden eşler. Aşama yanlış bağımlılıkları ortadan kaldırır

---

<sup>2</sup>Yeniden sıralama tamponlarına ilişkin bir tartışma için Ek N'ye bakın.

## 596 BÖLÜM 16 / KOMUT DÜZEYİ PARALELLİK VE SÜPERSKALAR İŞLEMCİLER

gerçek veri bağımlılıklarını (yazmadan sonra okuma) korurken sınırlı sayıda mimari kayıttan kaynaklanır.

**MİKRO-İŞLEM KUYRUĞU** Kaynak ve kayıtların yeniden adlandırılmasından sonra mikro-işlemler iki mikro-işlem kuyruğundan birine yerleştirilir ve zamanlayıcılarda yer açılanda kadar burada tutulurlar. İki kuyruktan biri bellek işlemleri (yüklemeler ve depolamalar) için, diğer ise bellek referansları içermeyen mikro işlemler içindir. Her kuyruk bir FIFO (ilk giren ilk çıkar) disiplinine uyar, ancak kuyruklar arasında herhangi bir sıra gözetilmez. Yani, bir mikro-işlem bir kuyruktan diğer kuyruktaki mikro-işlemlere göre sıra dışı olarak okunabilir. Bu, programlayıcılara daha fazla esneklik sağlar.

**MICRO-OP SCHEDULING AND DISPATCHING** Zamanlayıcılar mikro-op kuyruklarından mikro-op'ları almak ve bunları yürütme için göndermeye sorumludur. Her zamanlayıcı, durumu mikro-op'un tüm işlenenlerine sahip olduğunu gösteren mikro-op'ları arar. Bu mikro-op için gereken yürütme birimi mevcutsa, zamanlayıcı mikro-op'u alır ve uygun yürütme birimine gönderir. Bir döngüde en fazla altı mikro işlem gönderilebilir. Belirli bir yürütme birimi için birden fazla mikro işlem mevcutsa, zamanlayıcı bunları kuyruktan sırayla gönderir. Bu, sıralı yürütmemeyi tercih eden bir tür FIFO disiplinidir, ancak bu zamana kadar komut akışı bağımlılıklar ve dallanmalar tarafından o kadar yeniden düzenlenmiştir ki büyük ölçüde sıra dışıdır.

Dört bağlantı noktası zamanlayıcıları yürütme birimlerine bağlar. Port 0, Port 1'e tahsis edilen basit tamsayı işlemleri ve dallanma yanlış tahminlerinin işlenmesi haricinde hem tamsayı hem de kayan noktalı komutlar için kullanılır. Ek olarak, MMX yürütme birimleri bu iki port arasında tahsis edilir. Geriye kalan portlar bellek yüklemeleri ve depolamaları içindir.

### Tamsayı ve Kayan Nokta Yürütme Birimleri

Tamsayı ve kayan nokta kayıt dosyaları, yürütme birimlerinin bekleyen işlemleri için kaynaktır. Yürütme birimleri değerleri kayıt dosyalarının yanı sıra L1 veri önbelleğinden de alır. Bayrakları (örneğin, sıfır, negatif) hesaplamak için ayrı bir boru hattı aşaması kullanılır; bunlar tipik olarak bir dallanma komutunun girdisidir.

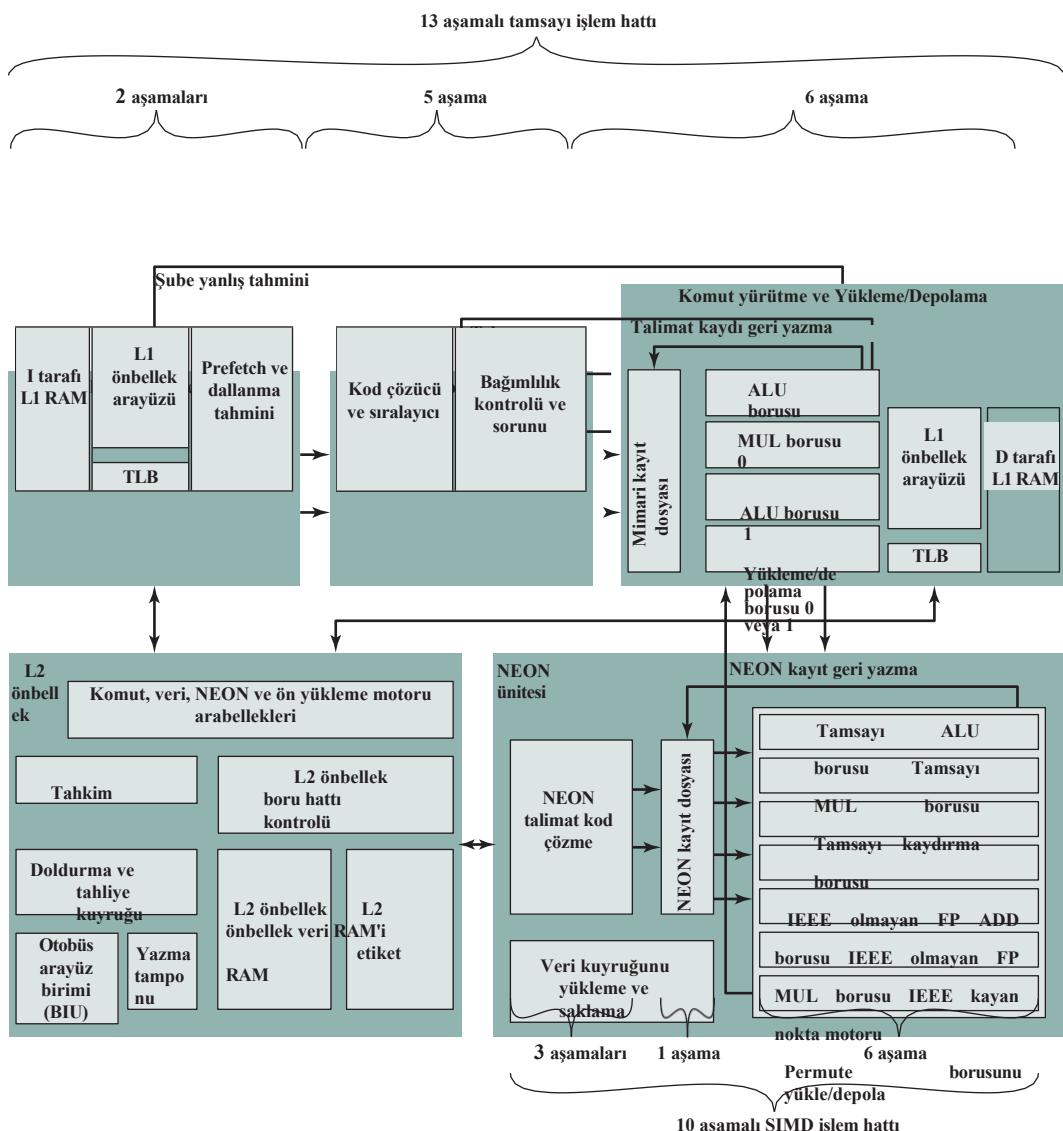
Sonraki bir boru hattı aşaması dallanma kontrolü gerçekleştirir. Bu işlev gerçek dallanma sonucunu tahmin ile karşılaştırır. Bir dallanma tahmininin yanlış olduğu ortaya çıkarsa, işlemin çeşitli aşamalarında boru hattından çıkarılması gereken mikro işlemler vardır. Doğru dallanma hedefi daha sonra bir sürücü aşaması sırasında Dallanma Tahmincisine sağlanır ve bu da tüm boru hattını yeni hedef adresten yeniden başlatır.

## 16.4 ARM CORTEX-A8

ARM mimarisinin son uygulamaları, komut işlem hattında süperskalar tekniklerin tanık olmuştur. Bu bölümde, RISC tabanlı süperskalar tasarıma iyi bir örnek teşkil eden ARM Cortex-A8'e odaklanacağız.

Cortex-A8, ARM'nin uygulama işlemcileri olarak adlandırdığı ARM işlemci ailesinde yer almaktadır. Bir ARM uygulama işlemcisi, kablosuz, tüketici ve görüntüleme uygulamaları için karmaşık işletim sistemleri çalıştırın gömülü bir işlemcidir. Cortex-A8, cep telefonları, set üstü kutular, oyun konsolları ve otomotiv navigasyon/eğlence sistemleri dahil olmak üzere çok çeşitli mobil ve tüketici uygulamalarını hedeflemektedir.

Şekil 16.9, Cortex-A8 mimarisinin mantıksal bir görünümünü göstermektedir. Ana komut akışı, çift, sıralı sorunlu, 13 aşamalı bir boru hattı uygulayan üç işlevsel birimden geçer. Cortex tasarımcıları ek komut akışını korumak için sıralı komut akışında kalmaya karar vermişlerdir.



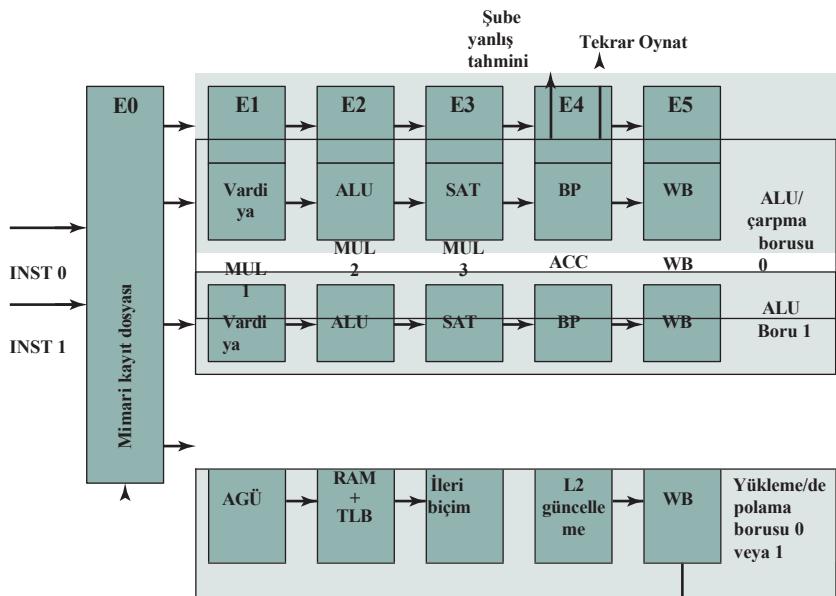
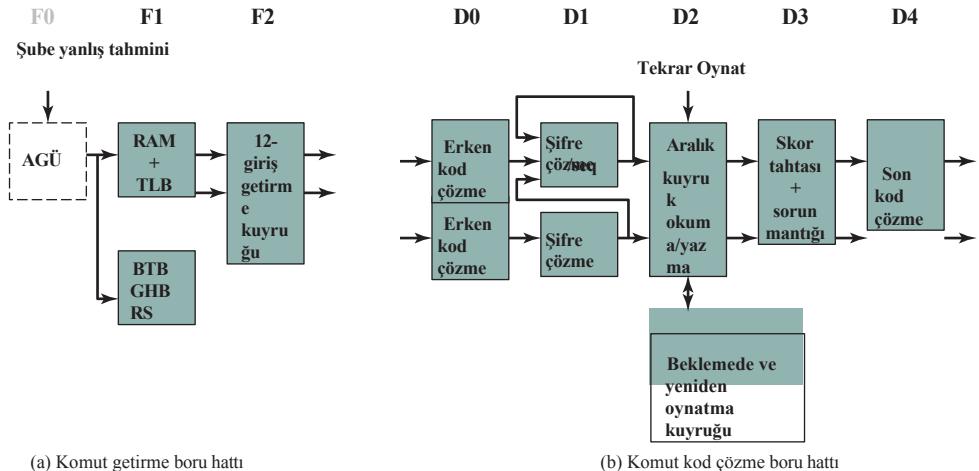
Şekil 16.9 ARM Cortex-A8'in Mimari Blok Diyagramı

minimum güç gerektirir. Sıra dışı yayılama ve **geri çekme**, ekstra güç tüketen büyük miktarda mantık gerektirebilir.

Sekil 16.10'da ana Cortex-A8 işlem hattının ayrıntıları gösterilmektedir. SIMD (tek komut-çoklu veri) birimi için 10 aşamalı bir boru hattı uygulayan ayrı bir birim vardır.

### Komut Getirme Birimi

Talimat getirme birimi talimat akışını tahmin eder, talimatları L1 talimat önbelleğinden alır ve alınan talimatları kod çözme işlem hattı tarafından kullanılmak üzere bir tampona yerleştirir. Komut getirme birimi aynı zamanda L1



(c) Komut yürütme ve yükleme/depolama işlem hattı

**Şekil 16.10** ARM Cortex-A8 Tamsayı İşlem Hattı

komut önbelleği. Boru hattında birden fazla çözülmemiş dallanma olabileceğinden, komut getirme işlemleri spekülatiftir, yani bu işlemlerin gerçekleştirileceğinin garantisini yoktur. Kod akışındaki bir dallanma ya da istisnai bir komut, boru hattının temizlenmesine neden olarak o anda getirilmiş komutları atabilir. Komut getirme birimi döngü başına dört adede kadar komut getirebilir ve aşağıdaki aşamalardan geçer:

**F0:** Adres üretim birimi (AGU) yeni bir sanal üretir. Genelde bu adres, önceki getirme adresinden sırayla bir sonraki adresdir. Adres, önceki bir komut için dallanma tahmini tarafından sağlanan bir dallanma hedef adresi de olabilir. ARM işlemcileri geleneksel olarak komut önbelleği erişimini ilk aşama olarak tanımladığından, F0 13 aşamalı işlem hattının bir parçası olarak sayılmaz.

**F1:** Hesaplanan adres, L1 taliyat önbelleğinden talimatları getirmek için kullanılır. Paralel olarak getirme adresi, bir sonraki getirme adresinin bir dallanma tahminine dayanıp dayanmayacağı belirlemek için dallanma tahmini dizilerine erişmek için kullanılır.

**F3:** Komut verileri komut kuyruğuna yerleştirilir. Bir komut dallanma tahminiyle sonuçlanırsa, yeni hedef adres oluşturma birimine gönderilir.

Tipik olarak daha derin bir boru hattıyla ilişkili dallanma cezalarını en aza indirmek için Cortex-A8 işlemci, dallanma hedef tamponu (BTB) ve küresel geçmiş tamponundan (GHB) oluşan iki seviyeli bir küresel geçmiş dallanma tahlincisi uygular. Bu veri yapılarına komut getirme işlemleriyle paralel olarak erişilir. BTB, geçerli getirme adresinin bir dallanma komutu döndürüp döndürmeyeceğini ve dallanma hedef adresini gösterir. 512 girdi içerir. BTB'deki bir işaret üzerine bir dallanma önceden tahmin edilir ve GHB'ye erişilir. GHB, dallanmaların güç ve yön bilgilerini kodlayan 4096 adet 2 bitlik sayaçtan oluşur. GHB, karşılaşılan son on dalın yönünün 10 bitlik bilgisi ve PC'nin 4 biti ile indekslenir. Dinamik dallanma tahlincisine ek olarak, alt rutin dönüş adreslerini tahmin etmek için bir dönüş yığını kullanılır. Dönüş yığını, r14'teki bağlantı kayıt değerini ve çağırılan işlevin ARM veya Thumb durumunu saklayan sekiz adet 32 bit girişe sahiptir. Geri dönüş tipi bir talimat tahmin edildiğinde, geri dönüş yığını son itilen adresi ve durumu sağlar.

Komut getirme birimi 12 adede kadar getirebilir ve sıraya koyabilir. Kod çözme birimine her seferinde iki talimat verir. Kuyruk, komut getirme biriminin tamsayı ardışık düzeninin kalanından önce komut getirmesini ve kod çözme için hazır bir komut birikimi oluşturmasını sağlar.

## Talimat Kod Çözme Birimi

Komut çözümleme birimi tüm ARM ve Thumb komutlarının kodunu çözer ve sıralar. *Boru0* ve *boru1* olarak adlandırılan ikili bir boru hattı yapısına sahiptir, böylece iki komut aynı anda üniteden geçebilir. Komut kod çözme boru hattından iki komut verildiğinde, pipe0 her zaman program sırasına göre daha eski olan komutu içerecektir. Bu, *boru0*'daki komutun verilememesi durumunda *boru1*'deki komutun da verilemeyeceği anlamına gelir. Verilen tüm talimatlar yürütme hattında sırayla ilerler ve sonuçlar yürütme hattının sonunda kayıt dosyasına geri yazılır. Bu sıralı komut verme ve geri alma WAR tehlikelerini önler ve WAW'ın izlenmesini sağlar

## 600 BÖLÜM 16 / KOMUT DÜZEYİ PARALELLİK VE SÜPERSKALAR İŞLEMÇİLER

tehlikeleri ve floş koşullarından kurtarmayı kolaylaştırır. Bu nedenle, komut kod çözme işlem hattının ana kaygısı RAW tehlikelerinin önlenmesidir.

Her talimat beş işlem aşamasından geçer.

**D0:** Thumb komutları 32 bit ARM komutlarına ayrıstırılır. Bir ön kod çözme işlevi gerçekleştirilir.

**D1:** Komut kod çözme işlevi tamamlanmıştır.

**D2:** Bu aşama talimatları bekleyen/tekrar kuyruk yapısına yazar ve bu yapıdan talimatları okur.

**D3:** Bu aşama komut çizelgeleme mantığını içerir. Bir çetele, statik çizelgeleme tekniklerini kullanarak kayıt kullanılabilirliğini tahmin eder.<sup>3</sup> Tehlike kontrolü de bu aşamada yapılır.

**D4:** Bütünleşik yürütme ve yükleme/depolama birimlerinin ihtiyaç duyduğu tüm kontrol sinyalleri için son kod çözme işlemini gerçekleştirir.

İlk iki aşamada, komut türü, kaynak ve hedef operatörler ve komut için kaynak gereksinimleri belirlenir. Daha az yaygın olarak kullanılan birkaç talimat çok döngülü talimatlar olarak adlandırılır. D1 aşaması bu talimatları, yürütme boru hattı boyunca ayrı ayrı sıralanan çoklu talimat kodlarına ayırrı.

Bekleme kuyruğu iki amaca hizmet eder. Birincisi, D3'ten gelen bir durma sinyalinin ardışık düzende daha fazla dalgalanmasını önler. İkincisi, talimatları tamponlayarak, çift boru hattı için her zaman iki taliminin mevcut olması gereklidir. Yalnızca bir taliminin verildiği durumda, bekleyen kuyruk, başlangıçta getirme biriminden farklı döngülerde gönderilmiş olsalar bile, iki talimin boru hattında birlikte ilerlemesini sağlar.

Yeniden oynatma işlemi, bellek sisteminin komut zamanlaması üzerindeki etkileriyle başa çıkmak için tasarlanmıştır. Komutlar, kaynak operandın ne zaman kullanılabilir olacağına dair bir tahmine dayalı D3 aşamasında statik olarak programlanır. Bellek sisteminden kaynaklanan herhangi bir duraklama en az 8 çevrimlik bir gecikmeye neden olabilir. Bu minimum 8 çevrimlik gecikme, L1 yükünün kaçırılması durumunda L2 önbelleğinden veri almak için mümkün olan minimum çevrim sayısı ile dengelenir. Tablo 16.3, bellek sisteminin durması nedeniyle komut tekrarına neden olabilecek en yaygın durumları vermektedir.

Bu duraklamalarla başa çıkmak için, yürütme hattındaki sonraki tüm talimatları temizlemek ve bunları yeniden yayılmak (yeniden oynatmak) için bir kurtarma mekanizması kullanılır. Tekrar oynatmayı desteklemek için, talimatlar yayınlanmadan önce tekrar oynatma kuyruğuna kopyalanır ve sonuçlarını geri yazıp emekli olurken kaldırılır. Bir yeniden oynatma sinyali, talimatlar yeniden oynatma kuyruğundan alınır ve boru hattına yeniden girer.

Kod çözme birimi, bir sorun kısıtlamasıyla karşılaşmadığı sürece yürütme birimine paralel olarak iki talimat yayırlar. Tablo 16.4 en yaygın kısıtlama durumlarını göstermektedir.

### Tamsayı Yürütme Birimi

Komut yürütme birimi iki simetrik aritmetik mantık birimi (ALU) boru hattı, yükleme ve saklama komutları için bir adres üreteci ve çarpma boru hattından oluşur. Yürütme boru hatları ayrıca yazma geri yazma işlemini de gerçekleştirir. Komut yürütme birimi:

<sup>3</sup>Puanlama ile ilgili bir tartışma için Ek N'ye bakınız.

**Tablo 16.3** Cortex-A8 Bellek Sisteminin Komut Zamanlamaları Üzerindeki Etkileri

Tekrar Oynatma Etkinliği	Gecikme	Açıklama
Veri yükleme hatası	8 döngü	<p><b>1.</b> Bir yükleme talimi L1 veri önbelleğinde kaçırılır.</p> <p><b>2.</b> Daha sonra L2 veri önbelleğine bir istek yapılır.</p> <p><b>3.</b> L2 veri önbelleğinde de bir ıskalama meydana gelirse, ikinci bir yeniden oynatma gerçekleşir. Durma döngülerinin sayısı harici sistem belleği zamanlamasına bağlıdır. Bir L2 önbellek kaçırması için kritik kelimeyi almak için gereken minimum süre yaklaşık 25 döngür, ancak L3 bellek gecikmeleri nedeniyle çok daha uzun olabilir.</p>
Veri TLB kaçırma	24 döngü	<p><b>1.</b> L1 TLB'deki bir ıskalama nedeniyle bir tablo yürütüsü, çeviri tablosu girdilerinin L2 önbelleğinde bulunduğu varsayılarak 24 çevrimlik bir gecikmeye neden olur.</p> <p><b>2.</b> Çeviri tablosu girdileri L2 önbelleğinde mevcut değilse, durma döngülerinin sayısı harici sistem belleği zamanlamasına bağlıdır.</p>
Depo arabelleği dolu	8 döngü artı dolum tamponunu boşaltma gecikmesi	<p><b>1.</b> Saklama tamponu dolu olmadığı sürece bir saklama talimi kaçırma herhangi bir duraklamaya neden olmaz.</p> <p><b>2.</b> Depo tamponunun dolu olması durumunda, gecikme en az sekiz döngür. Depo tamponundan bazı girdilerin boşaltılması daha uzun sürerse gecikme daha fazla olabilir.</p>
Hizalanmamış yükleme veya depolama isteği	8 döngü	<p><b>1.</b> Bir yükleme talimi adresi hizalanmamışsa ve tam erişim 128 bitlik bir sınır içinde yer alımırsa, 8 çevrimlik bir ceza söz konusudur.</p> <p><b>2.</b> Bir depolama komutu adresi hizalanmamışsa ve tam erişim 64 bitlik bir sınır içinde yer alımırsa, 8 çevrimlik bir ceza söz konusudur.</p>

- Bayrak üretimi de dahil olmak üzere tüm sayı ALU ve çarpma işlemlerini yürütür.
- Gerektiğinde yüklemeler ve depolamalar için sanal adresleri ve temel geri yazma değerini oluşturur.
- Veri ve bayrakları depolamak ve iletmek için biçimlendirilmiş veri sağlar.
- Dallanmaları ve komut akışındaki diğer değişiklikleri işler ve durum kodlarını değerlendirir.

ALU talimatları için, aşağıdaki aşamalardan oluşan her iki boru hattı da kullanılabilir:

**E0:** Kayıt dosyasına erişim. İki komut için kayıt dosyasından altı adede kadar kayıt okunabilir.

**E1:** Fıçı değiştirici (bkz. Şekil 14.25) gerektiğinde işlevini yerine getirir.

**E2:** ALU birimi (bkz. Şekil 14.25) işlevini yerine getirir.

**E3:** Gerekirse, bu aşama bazı ARM veri işleme talimatları tarafından kullanılan doygunluk arıtmayı tamamlar.

**E4:** Dal yanlış tahminleri, istisnalar ve bellek sistemi tekrarları dahil olmak üzere kontrol akışındaki her türlü değişiklik önceliklendirilir ve işlenir.

**E5:** ARM talimatlarının sonuçları kayıt dosyasına geri yazılır.

## 602 BÖLÜM 16 / KOMUT DÜZEYİ PARALELLİK VE SÜPERSKALAR İŞLEMCİLER

**Tablo 16.4** Cortex-A8 Çift Konu Kısıtlamaları

Kısıtlama Türü	Açıklama	Örnek	Döngü	Kısıtlama
Yükle/depola Kaynak Tehlike	Sadece tek bir LS boru hattı vardır. Yalnızca bir LS talimatı döngü başına verilebilir. İçinde olabilir boru hattı 0 veya boru hattı 1.	LDR r5, [r6] STR r7, [r8] MOV r9, r10	1 2 2	LS birimini bekleyin İkili sorun mümkün
Kaynak tehlikesinin i çoğaltın	Yalnızca bir çarpma boru hattı vardır ve yalnızca boru hattı 0'da kullanılabilir.	ADD r1, r2, r3 MUL r4, r5, r6 MUL r7, r8, r9	1 2 3	Boru hattını bekleyin 0 Çarpma birimini bekleyin
Şube kaynak tehlikesi	Sadece bir şube olabilir döngü başına. Boru hattı 0 veya boru hattı 1'de olabilir. Bir dallanma değiştiren herhangi bir talimat PC.	BX r1 BEQ 0x1000 ADD r1, r2, r3	1 2 2	Şubeyi bekleyin ikili sorun mümkün
Veri çıkıştı tehlikesi	Aynı sahip komutlar aynı döngü içinde verilemez. Bu durum koşullu kod ile gerçekleştirilebilir.	MOVEQ r1, r2 MOVNE r1, r3 LDR r5, [r6]	1 2 2	Cıktı bağımlılığı nedeniyle bekleyin İkili sorun mümkün
Veri kaynak tehlikesi	Talimatlar yayınlanamaz eğer verileri mevcut değilse. Aşağıdakiler için zamanlama tablolarına bakın kaynak gereksinimleri ve aşamaları Sonuçlar.	ADD r1, r2, r3 ADD r4, r1, r6 LDR r7, [r4]	1 2 4	r1 için bekleyin r4 için iki döngü bekleyin
Çoklu döngü talimatları	Çok döngülü talimatlar boru hattı 0'da sorun ve yalnızca son yinelemelerinde ikili sorun.	MOV r1, r2 LDM r3, {r4-r7} LDM (döngü 2) LDM (döngü 3)  ADD r8, r9, r10	1 2 3 4  4	0 boru hattını bekle, r4'ü aktar r5, r6'yi aktar Transfer r7 Son transferde çifte ihraç mümkün

Çarpma birimini çağırılan komutlar (bkz. Şekil 14.25) boru0'a yönlendirilir; çarpma işlemi E1'den E3'e kadar olan aşamalarda ve çok katlı biriktirme işlemi E4 aşamasında gerçekleştirilir.

Yükleme/depolama işlem hattı, tamsayı işlem hattına paralel olarak çalışır. Aşamalar aşağıdaki gibidir:

**E1:** Bellek adresi taban ve indeks kayıtlarından oluşturulur.

**E2:** Adres önbellek dizilerine uygulanır.

**E3:** Yükleme durumunda, veriler döndürülür ve ALU veya MUL birimine iletilmek üzere biçimlendirilir. Depolama durumunda, veriler biçimlendirilir ve önbelleğe yazılmasına hazır hale gelir.

**E4:** Gerekirse L2 önbelleğinde güncellemeler gerçekleştirir.

**E5:** ARM komutlarının sonuçları kayıt dosyasına geri yazılır.

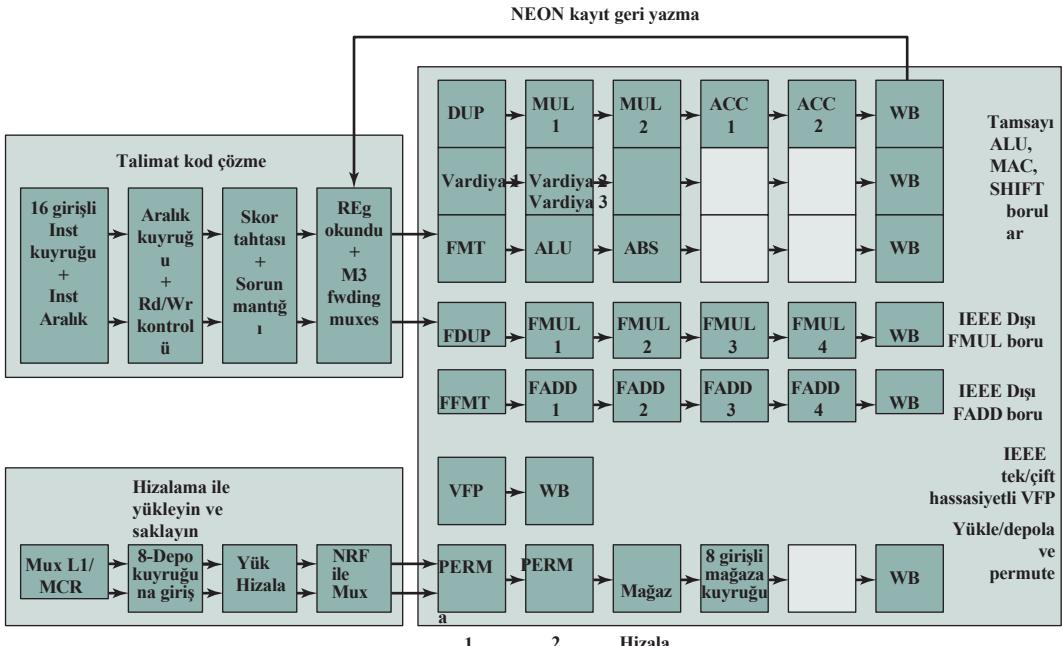
Tablo 16.5'te örnek bir kod segmenti gösterilmekte ve işlemcinin bunu nasıl programlayabileceğini belirtilmektedir.

**Tablo 16.5** Cortex-A8 Tamsayı İş Hattı için Örnek Çift Sayı Komut Sırası

Döngü	Program Sayacı	Talimatlar	Zamanlama Açıklaması
1	0x00000ed0	BX r14	İkili ihraç boru hattı 0
1	0x00000ee4	CMP r0,#0	Boru hattında ikili sorun 1
2	0x00000ee8	MOV r3,#3	İkili ihraç boru hattı 0
2	0x00000eec	MOV r0,#0	Boru hattında ikili sorun 1
3	0x00000ef0	STREQ r3,[r1,#0]	Boru hattı 0'da ikili sorun, r3 E3'e kadar gerekli değil
3	0x00000ef4	CMP r2,#4	Boru hattında ikili sorun 1
4	0x00000ef8	LDRLS pc,[pc,r2,LSL #2]	Tek sorunlu boru hattı 0,+ bilgisayara yükleme için 1 döngü, LSL #2'den beri kaydırma için ekstra döngü yok
5	0x00000f2c	MOV r0,#1	Boru hattı 1'de yükün 2. yinelemesiyle ilgili ikili sorun
6	0x00000f30	B 5 adet 6+ 8	#0xf38 çift sayı boru hattı 0
6	0x00000f38	STR r0,[r1,#0]	Çift sayı boru hattı 1
7	0x00000f3c:	LDR pc,[r13],#4	Tek sayı boru hattı 0,+ bilgisayara yükleme için 1 döngü
8	0x0000017c	ADD r2,r4,#0xc	Boru hattı 1'de yükün 2. yinelemesiyle ilgili ikili sorun
9	0x00000180	LDR r0,[r6,#4]	İkili ihraç boru hattı 0
9	0x00000184	MOV r1,#0xa	Çift sayı boru hattı 1
12	0x00000188	LDR r0,[r0,#0]	Tek sorunlu boru hattı 0: r0 E3'te üretildi, E1'de gerekli, bu nedenle+ 2 döngü durma
13	0x0000018c	STR r0,[r4,#0]	LS kaynak tehlikesi nedeniyle tek sorunlu boru hattı 0, E3'te üretildiği ve E3'te tüketildiği için r0 için ekstra gecikme yok
14	0x00000190	LDR r0,[r4,#0xc]	LS kaynak tehlikesi nedeniyle tek sorunlu boru hattı 0
15	0x00000194	LDMFD r13!,{r4-r6,r14}	Çoklu yükleme: r4'ü 1. döngüde, r5 ve r6'yı 2. döngüde, r14'ü 3. döngüde, toplam 3 döngüde yükler
17	0x00000198	B 5 pc 6+ 0xda8	LDM'nin 3. döngüsü ile boru hattı 1'de #0xf40 ikili sorun
18	0x00000f40	ADD r0,r0,#2 ARM	Boru hattında tek sayı 0
19	0x00000f44	ADD r0,r1,r0 ARM	Boru hattı 0'da tek sorun, E2'de üretilen ve E2'de gerekli olan r0 üzerindeki tehlike nedeniyle ikili sorun yok

## SIMD ve Kayan Noktalı İşlem Hattı

Tüm SIMD ve kayan nokta komutları tamsayı ardışık düzendinden geçer ve 10 aşamalı ayrı bir ardışık düzende işlenir (Şekil 16.11). NEON birimi olarak adlandırılan bu birim, paketlenmiş SIMD talimatlarını işler ve iki tür kayan nokta desteği sağlar. Uygulanırsa, bir vektör kayan nokta (VFP) yardımcı işlemci IEEE 754 ile uyumlu olarak kayan nokta işlemlerini gerçekleştirir. Yardımcı işlemci mevcut değilse, ayrı çarpma ve ekleme boru hatları kayan nokta işlemlerini gerçekleştirir.



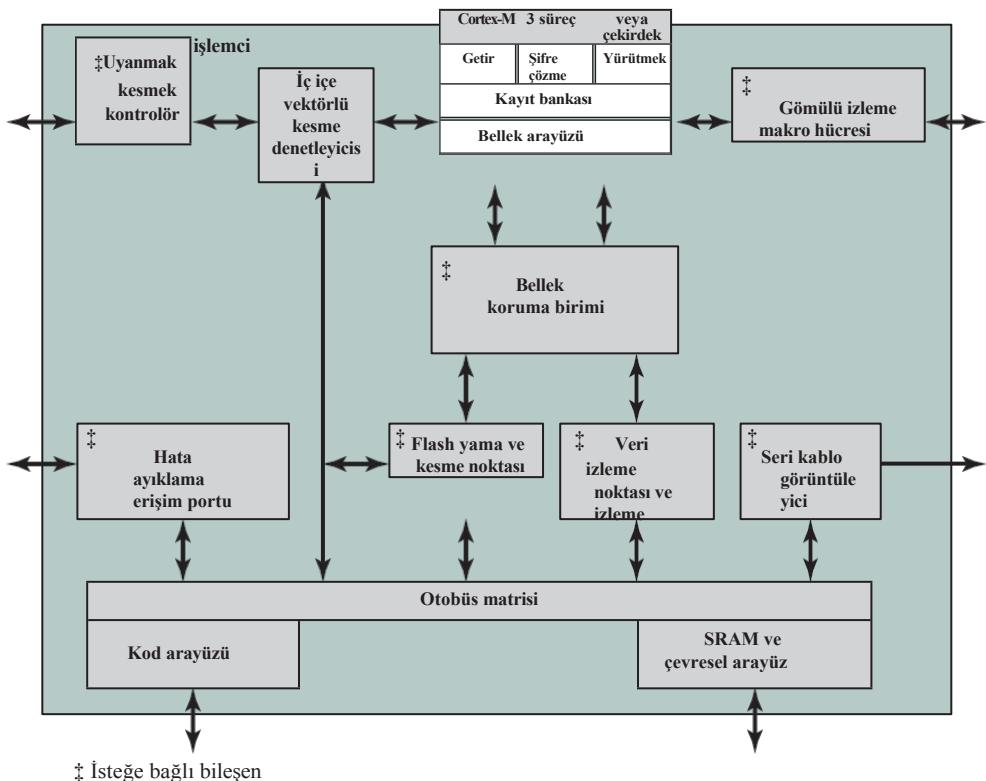
Şekil 16.11 ARM Cortex-A8 NEON ve Kayan Nokta İşlem Hattı

## 16,5 KOL KORTEKS-M3

Bir önceki bölümde bir uygulama işlemcisi olan Cortex-A8'in oldukça karmaşık işlem hattı organizasyonu incelenmişti. Faydalı bir kontrast olarak, bu bölüm Cortex-M3'ün oldukça basit boru hattı organizasyonunu incelemektedir. Cortex-M serisi mikrodenetleyici alanı için tasarlanmıştır. Bu nedenle, Cortex-M işlemcilerinin mümkün olduğunda basit ve verimli olması.

Şekil 16.12, Cortex-M3 işlemcisinin blok diyagramına genel bir bakış sağlar. Bu şekil, Şekil 1.16'da gösterilen daha fazla ayrıntı sağlar. Anahtar öğeler şunları içerir:

- İşlemci çekirdeği:** Üç aşamalı bir boru hattı, bir kayıt bankası ve bir bellek arayüzü içerir.
- Bellek koruma birimi:** İşletim sistemi tarafından kullanılan kritik verileri kullanıcı uygulamalarından korur, birbirlerinin verilerine erişime izin vermeyerek işlem görevlerini ayırrı, bellek bölgelerine erişimi devre dışı bırakır, bellek bölgelerinin salt okunur olarak tanımlanmasına izin verir ve sistemi bozabilecek bellek erişimlerini tespit eder.
- İç içe vektörlü kesme denetleyicisi (NVIC):** İşlemciye yapılandırılabilir kesinti işleme yetenekleri sağlar. Düşük gecikmeli istisna ve kesme işlemlerini kolaylaştırır ve güç yönetimini kontrol eder.
- Uyandırma kesme denetleyicisi (NVIC):** İşlemciye yapılandırılabilir kesme yönetimi yetenekleri sağlar. Düşük gecikmeli istisna ve kesmeler arası işlemeyi kolaylaştırır ve güç yönetimini kontrol eder.
- Flash yama ve kesme noktası birimi:** Kesme noktaları ve kod yamaları uygular.



**Şekil 16.12** ARM Cortex-M3 Blok Diyagramı

- **Veri izleme noktası ve izleme (DWT):** İzleme noktaları, veri izleme ve sistem profili oluşturma uygular.
- **Seri kablo görüntüleyici:** Yazılım tarafından oluşturulan mesaj akışını, veri izini ve profil oluşturma bilgilerini tek bir pin aracılığıyla dışa aktarabilir.
- **Hata ayıklama erişim portu:** İşlemciye harici hata ayıklama erişimi için bir arayüz sağlar.
- **Gömülü izleme makro hücresi:** İşletim sistemi ve uygulama olaylarını izlemek için printf() tarzı hata ayıklamayı destekleyen ve tanılayıcı sistem bilgileri üreten uygulama odaklı bir izleme kaynağıdır.
- **Veri yolu matrisi:** Çekirdek ve hata ayıklama arayüzlerini mikrodenetleyici üzerindeki harici veri yollarına bağlar.

### Boru Hattı Yapısı

Cortex-M3 işlem hattının üç aşaması vardır (Şekil 16.12). Bunları sırayla inceleyeceğiz. Getirme aşaması sırasında, bir seferde 32 bitlik bir sözcük getirilir ve bir 3 kelimelik tampon, 32 bitlik kelime şunlardan olusabilir:

- iki Thumb talimatı,
- bir sözcük hizalı Thumb-2 talımı veya

## 606 BÖLÜM 16 / KOMUT DÜZEYİ PARALELLİK VE SÜPERSKALAR İŞLEMCİLER

- ile yarım sözcük hizalı bir Thumb-2 komutunun üst/alt yarım sözcüğü
  - bir Thumb talimatı veya
  - başka bir yarım sözcük hizalı Thumb-2 komutunun alt/üst yarım sözcüğü.

Çekirdektenden gelen tüm getirme adresleri kelime hizalıdır. Bir Thumb-2 komutu yarım kelime , Thumb-2 komutunu getirmek için iki getirme gereklidir. Bununla birlikte, üç girişli ön getirme tamponu, bir durma döngüsünün yalnızca getirilen ilk yarım sözcük Thumb-2 komutu için gerekli olmasını sağlar.

Bu kod çözme aşaması üç temel işlevi yerine getirir:

- **Komut kod çözme ve kayıt okuma:** Thumb ve Thumb-2 komutlarının kodunu çözer.
- **Adres üretimi:** Adres üretim birimi (AGU), yükleme/depolama birimi için ana bellek adreslerini üretir.
- **Dallanma:** Dallanma komutundaki anlık ofsete veya bağlantı kaydının (kayıt R14) içeriğine dayalı bir dönüşe dayalı dallanma gerçekleştirir.

Son olarak, ALU, load/store ve dallanma talimatlarını içeren komut yürütme için tek bir yürütme aşaması vardır.

### Şubeler ile Çalışmak

İşlemciyi mümkün olduğunda basit tutmak için Cortex-M3 işlemci dal tahminini kullanmaz, bunun yerine aşağıdaki gibi tanımlanan basit dal yönlendirme ve dal speküasyon tekniklerini kullanır:

- **Dal yönlendirme:** *Yönlendirme* terimi, bellekten getirilecek bir komut adresinin sunulması anlamına gelir. İşlemci, dallanmanın bellek işleminin, işlem kodunun yürütmeye ulaştığı zamandan en az bir döngü önce sunulduğu belirli dallanma türlerini iletir. Dallanma iletimi çekirdeğin performansını artırır, çünkü dallanmalar gömülü denetleyici uygulamalarının önemli bir parçasıdır. Etkilenen dallar, hemen ofset ile PC görelidir veya hedef kayıt olarak bağlantı kaydını (LR) kullanır.
- **Dallanma speküasyonu:** Koşullu dallanmalar için, komut adresi spekülatif olarak önceden gönderilir, böylece komutun yürütülüp yürütülmeyeceği bilinmeden önce komut bellekten getirilir.

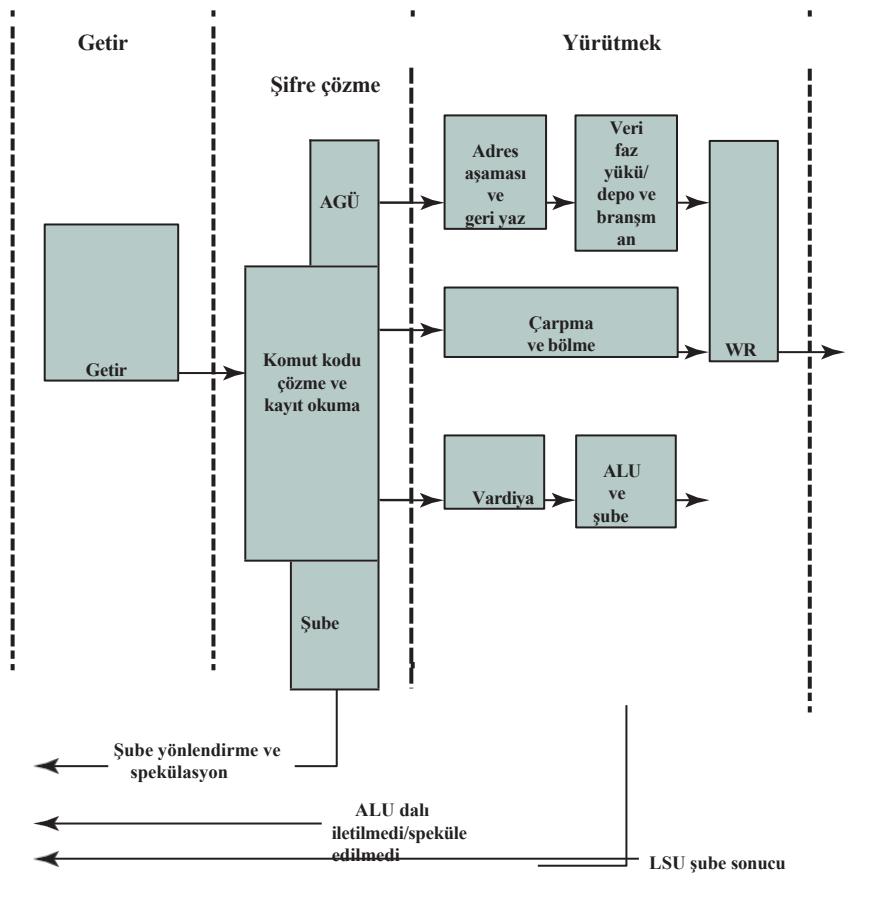
Cortex-M3 işlemci, getirme tamponunu kullanarak yürütme öncesinde komutları önceden alır. Ayrıca dallanma hedef adreslerinden spekülatif olarak önceden alır. Özellikle, koşullu bir dallanma komutıyla karşılaşıldığında, kod çözme aşaması daha hızlı yürütmeye yol açabilecek spekülatif bir komut getirmeyi de içerir. İşlemci, kod çözme aşaması sırasında dallanma hedefi komutunu getirir. Daha sonra, yürütme aşaması sırasında dallanma çözülür ve sonraki komutun yürütüleceği bilinir.

Dallanma , bir sonraki sıralı komut zaten kullanılabilir durumdadır. Eğer dallanma yapılacaksa, dallanma komutu karar verildiği anda kullanıma sunulur ve boşta kalma süresi sadece bir döngü ile sınırlandırılır.

Şekil 16.13, aşağıdaki gibi tanımlanabilecek dalların ele alınma şéklini açıklamaktadır:

1. Kod çözme aşaması koşulsuz gelen adresleri iletir ve adresi hesaplamak mümkün olduğunda koşullu dallanmalardan gelen adresleri spesifik olarak iletir.
2. ALU bir dallanmanın belirlerse, bu bilgi komut önbelleğini boşaltmak için geri beslenir.
3. Program sayacına bir yükleme talimatı, getirme için iletilecek bir dallanma adresi ile sonuçlanır.

Görülebileceği gibi, dalların işlenme şéklı Cortex-M için Cortex-A'dan çok daha basittir ve daha az işlemci mantığı ve işleme gerektirir.



AGU= adres üretim birimi

**Şekil 16.13** ARM Cortex-M3 İş Hattı

## 16.6 ANAHTAR TERİMLER, GÖZDEN GEÇİRME SORULARI VE PROBLEMLER

### Anahtar Terimler

bağımlılık karşıtı dal tahmin taahhüdü akış bağımlılığı sıra-içi tamamlama sıra-içi talimat sorunu komut düzeyinde paralellik komut penceresi	makine paralelliği mikro-işlemler mikro-ops sıra dışı tamamlama sıra dışı sorun çıktı bağımlılığı prosedürel bağımlılık okuma-yazma bağımlılığı	kayıt yeniden adlandırma kaynak çakışması süperpipelined superscalar'ı emekli et doğru veri bağımlılığı yazma-okuma bağımlılığı yazma-yazma <b>BAĞIMLİLİK</b>
---	---	---

### İnceleme Soruları

- 16.1** İşlemci tasarımindan süperskalar yaklaşımının temel özelliği nedir?
- 16.2** Süperskalar ve süperpipelined yaklaşımalar arasındaki fark nedir?
- 16.3** Komut düzeyinde paralellik nedir?
- 16.4** Aşağıdaki terimleri kısaca tanımlayınız:
  - Gerçek veri bağımlılığı
  - Prosedürel bağımlılık
  - Kaynak çatışmaları
  - Çıktı bağımlılığı
  - Bağımlılık Karşıtı
- 16.5** Komut düzeyinde paralellik ile makine arasındaki fark nedir?
- 16.6** Üç tip superscalar komut verme politikasını listeleyiniz ve kısaca tanımlayınız.
- 16.7** Talimat penceresinin amacı nedir?
- 16.8** Kayıt yeniden adlandırma nedir ve amacı nedir?
- 16.9** Süperskalar işlemci organizasyonunun temel unsurları nelerdir?

### Problemler

- 16.1** Superscalar bir işlemcide sıra dışı tamamlama kullanıldığında, kesme işleminden sonra yürütmenin yeniden başlatılması karmaşıkta, çünkü istisnai durum, sonucunu sıra dışı üreten bir komut olarak algılanmış olabilir. Program, istisnai komutu takip eden komutta yeniden başlatılamaz, çünkü sonraki komutlar zaten tamamlanmıştır ve bunu yapmak bu komutların iki kez yürütülmesine neden olur. Bu durumla başa çıkmak için bir mekanizma ya da mekanizmalar öneriniz.
- 16.2** Sözdiziminin bir işlem kodu, ardından hedef kayıt ve ardından bir veya iki kaynak kayıttan oluşanluğu aşağıdaki talimat dizisini göz önünde bulundurun:

0	ADD	R3, R1, R2
1	YÜKLE	R6, [R3]
2	VE	R7, R5, 3
3	ADD	R1, R6, R7
4	SRL	R7, R0, 8

5	VEYA	R2, R4, R7
6	SUB	R5, R3, R4
7	ADD	R0, R1, 10
8	YÜKLE	R6, [R5]
9	SUB	R2, R1, R6
10	VE	R3, R7, 15

Dört aşamalı bir boru hattı kullanıldığını varsayı: getirme, kod çözme/çıkarma, yürütme, geri yazma. Yürütme aşaması hariç tüm boru hattı aşamalarının bir saat döngüsü sürdüğünü varsayı. Basit tamsayı aritmetik ve mantıksal talimatlar için yürütme aşaması bir döngü alır, ancak bellekten bir LOAD için yürütme aşamasında beş döngü tüketilir.

Basit bir skaler ardışık düzenniz varsa ancak sıra dışı yürütmeye izin veriyorsak, ilk yedi talimin yürütülmesi için aşağıdaki tabloyu oluşturabiliriz:

Talimatlar	Getir	Şifre çözme	Yürütmek	Geri Yaz
0	0	1	2	3
1	1	2	4	9
2	2	3	5	6
3	3	4	10	11
4	4	5	6	7
5	5	6	8	10
6	6	7	9	12

Dört boru hattı aşamasının altındaki girişler, her bir komutun her bir aşamaya başladığı saat döngüsünü gösterir. Bu programda, ikinci ADD talimatı (talimat 3), işlenenlerinden biri olan r6 için LOAD talimatına (talimat 1) bağlıdır. LOAD komutu beş saat çevrimi süredünden ve yayılama mantığı iki saat sonra bağımlı ADD komutıyla karşılaşduğundan, yayılama mantığı ADD komutunu üç saat çevrimi geciktirmelidir. Sıra dışı bir yetenekle, işlemci 3 numaralı talimi 4 numaralı saat döngüsünde durdurabilir ve ardından 6, 8 ve 9 numaralı saatlerde yürütmeye giren sonraki üç bağımsız talimi vermeye devam edebilir. LOAD saat 9'da yürütmemeyi bitirir ve böylece bağımlı ADD saat 10'da yürütmeye başlatılabilir.

- a. Önceki tabloyu tamamlayın.
- b. Sipariş dışı kabiliyeti olmadığını varsayıarak tabloyu yeniden oluşturun. Yeteneği kullanarak elde edilen tasarruf nedir?
- c. Her aşamada bir seferde iki komutu işleyebilen bir süper skaler uygulama varsayıarak tabloyu yeniden yapın.

#### 16.3 Aşağıdaki assembly dili programını düşünün:

```

G1: R3, R7'yi hareket ettirin      /R3 d (R7(/)
I2: R8'i yükle, (R3(      /R8 d Bellek (R3(/ 
I3: R3, R3, 4 ekleyin      /R3 d (R3(+ 4/
I4: R9'u yükle, (R3(      /R9 d Bellek (R3(/ 
I5: BLE R8, R9, L3      /Şube eğer (R9(> (R8(/)

```

Bu program WAW, RAW ve WAR bağımlılıklarını içerir. Bunları göster.

#### 16.4 a. Aşağıdaki talimat dizisinde RAW, WAR ve WAW bağımlılıklarını tanımlayın:

```

I1: R1= 100 I2:
R1= R2+ R4 I3:
R2= r4 - 25

```

**610 BÖLÜM 16 / KOMUT DÜZEYİ PARALELLİK VE SÜPERSKALAR İŞLEMCİLER**

I4: R4= R1+ R3 I5:  
R1= R1+ 30

- b.** Bağımlılık sorunlarını önlemek için (a) bölümündeki kayıtları yeniden adlandırın. Kayıt referansına "a" alt simgesi kullanarak ilk kayıt değerlerine referansları tanımlayın.

**16.5** Şekil 16.14'te gösterilen "sıra-içi-sorun/sıra-içi-tamamlama" yürütme sırasını düşünün.

  - I2'nin dördüncü döngüye kadar yürütme aşamasına girememesinin en olası nedenini belirleyin. "Sıra içi sorun/sıra dışı tamamlama" veya "sıra dışı sorun/sıra dışı tamamlama" bunu çözecek mi? Eğer öyleyse, hangisi?
  - I6'nın dokuzuncu döngüye kadar yazma aşamasına girememesinin nedenini belirleyin. "Sıra içi sorun/sıra dışı tamamlama" veya "sıra dışı sorun/sıra dışı tamamlama" bunu çözecek mi? Eğer öyleyse, hangisi?

**16.6** Şekil 16.15'te süperskalar işlemci organizasyonuna bir örnek gösterilmektedir. Kaynak çakışması ve veri bağımlılığı sorunu yoksa işlemci döngü başına iki komut verebilir. Esasen dört işlem aşamalı (getirme, kod çözme, yürütme ve saklama) iki boru hattı vardır. Her boru hattının kendi getirme kod çözme ve depolama birimi vardır. Dört işlevsel birim (çarpan, toplayıcı, mantık birimi ve yükleme birimi) yürütme aşamasında kullanılabilir ve iki boru hattı tarafından dinamik olarak paylaşılır. İki depolama birimi, belirli bir döngüdeki kullanılabilirliğine bağlı olarak iki boru hattı tarafından dinamik olarak kullanılabilir. Kendi getirme ve kod çözme mantığına sahip bir lookahead penceresi vardır. Bu pencere, sıra dışı komutlar için komut önceliği için kullanılır.

Bu işlemci üzerinde çalıştırılacak aşağıdaki programı düşünün:

```
I1: Yük R1, A          /R1 d Bellek (A ()  
I2: R2, R1 ekleyn /R2 d (R2 (+ R(1 ()  
I3: R3, R4 ekleyn /R3 d (R3 (+ R(4 ()  
I4: Mul , R5          /R4 d (R4 (+ R(5 ()  
G5: Comp R6           /R6 d (R6 ()  
I6: Mul R6, R7         /R6 d (R6 (* R(7 ())
```

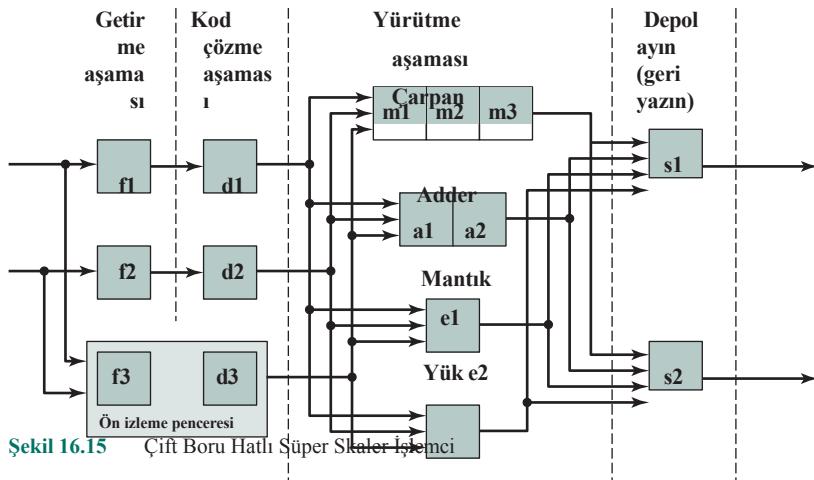
  - Programda hangi bağımlılıklar var?
  - Bu program için boru hattı faaliyetini Şekil 16.15'teki işlemcide sıra-içi tamamlama ilkeleriyle sıra-içi sorunu kullanarak ve Şekil 16.2'ye benzer bir sunumla gösterin.
  - Sıra dışı tamamlama ile sıra içi sorun için tekrarlayın.
  - Sıra dışı tamamlama ile sıra dışı sorun için tekrarlayın.

**16.7** Şekil 16.16 süperskalar tasarım üzerine bir makaleden almıştır. Şeklin üç bölümünü açıklayın ve w, x, y ve z'yi tanımlayın.

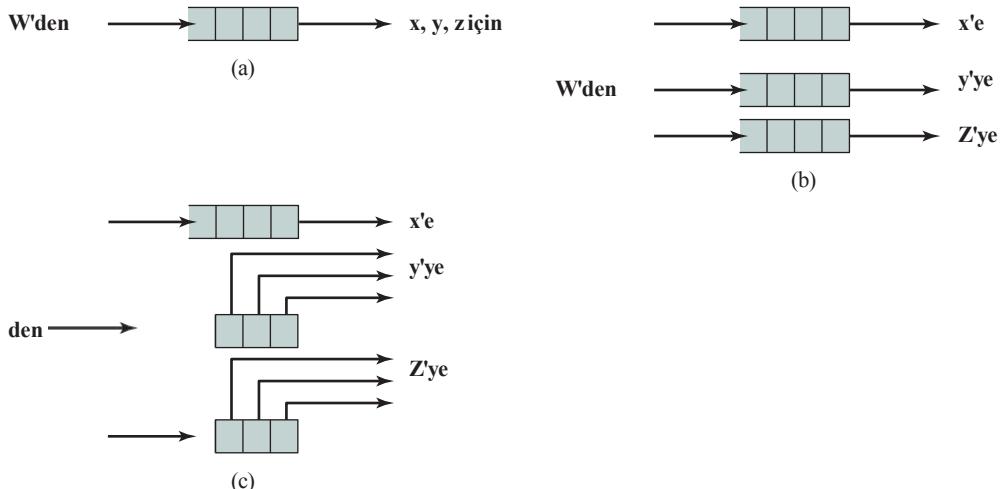
**16.8** Pentium 4'te kullanılan Yeh'in dinamik dal tahmin algoritması iki seviyeli bir dal tahmin algoritmasıdır. İlk seviye son  $n$  şubenin geçmişiştir. İkinci seviye

Şifre çözme		Yürütmek			Yazmak		Döngü
I1	I2						1
	I2			I1			2
	I2			I1			3
I3	I4			I2			4
I5	I6			I4	I3		5
I5	I6	I5		I3		I1 I2	6
		I5	I6				7
						I3 I4	8
							9

**Sekil 16.14** Bir Sırah Sorun Sırah Tamamları ve Sırası

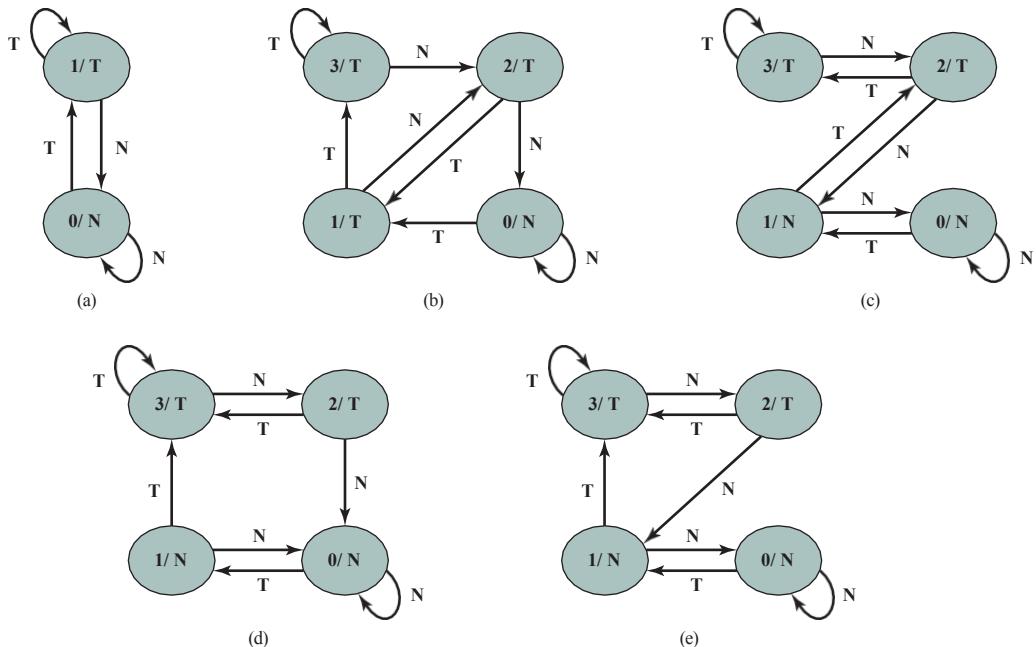


İkinci seviye, son  $n$  dallanmanın bu benzersiz modelinin son  $s$  oluşumunun dallanma davranışıdır. Bir programdaki her koşullu dallanma komutu için Dallanma Geçmiş Tablosunda (BHT) bir giriş vardır. Her giriş, dallanma komutunun son  $n$  yürütmesine karşılık gelen  $n$  bitten oluşur ve dallanma gerçekleştirmişse 1, gerçekleştirmemişse 0 alır. Her BHT girişi,  $n$  bitlik her olası desen için bir tane olmak üzere  $2^n$  girişin bir Desen Tablosuna (PT) indekslenir. Her PT girişi, Bölüm 14'te açıkladığı gibi dallanma tahmininde kullanılan  $s$  bitten oluşur (örneğin, Şekil 14.19). Komut getirme ve kod çözme sırasında koşullu bir dallanma ile karşılaşıldığında, komutun adresi, komutun yakın geçmişini gösteren uygun BHT girişini almak için kullanılır. Ardından, BHT girişi dallanma tahmini için uygun PT giriğini almak için kullanılır. Dallanma gerçekleştirildikten sonra BHT girişi güncellenir ve ardından uygun PT girişi güncellenir.



**Şekil 16.16** Problem 16.7 için Şekil

## 612 BÖLÜM 16 / KOMUT DÜZEYİ PARALELLİK VE SÜPERSKALAR İŞLEMÇİLER



**Şekil 16.17** Problem 16.8 için Şekil

- Bu şemanın performansını test ederken Yeh, Şekil 16.17'de gösterilen beş farklı tahmin şeması denemiştir. Bu şemalardan hangi üçünün Şekil 14.19 ve 14.28'de gösterilenlere karşılık geldiğini belirleyiniz. Kalan iki şemayı açıklayınız.
- Bu algoritma ile tahmin sadece bu özel dallanma talimatının yakın geçmişine dayanmaz. Aksine, bu talimat için BHT girişindeki  $n$ -bit deseniyle eşleşen tüm dallanma paternlerinin yakın geçmişine dayanır. Böyle bir strateji için bir gerekçe önerin.

## **PARALEL İŞLEM**

### **17.1 Çoklu İşlemci Kuruluşları**

Paralel İşlemci Sistemleri Türleri Paralel Organizasyonlar

### **17.2 Simetrik Çoklu İşlemciler**

Organizasyon  
Çok İşlemcili İşletim Sistemi Tasarımında Dikkat Edilmesi Gerekenler

### **17.3 Önbellek Tutarlılığı ve MESI Protokolü**

Yazılım Çözümleri  
Donanım Çözümleri MESI  
Protokolü

### **17.4 Çoklu İş Parçacığı ve Yonga Çoklu İşlemcileri**

Örtük ve Açık Çoklu İş Parçacığı Açık Çoklu İş Parçacığı Yaklaşımları

### **17.5 Kümeler**

Küme Yapılandırmaları İşletim Sistemi Tasarım Sorunları Küme Bilgisayar Mimarisi Blade Sunucular SMP ile Karşılaştırıldığında Kümeler

### **17.6 Tekdüze Olmayan Bellek Erişimi**

Motivasyon Organizasyonu  
NUMA Artıları ve Eksileri

### **17.7 Bulut Bilişim**

Bulut Bilişim Unsurları  
Bulut Bilişim Referans Mimarisi

### **17.8 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler**

**ÖĞRENİM HEDEFLERİ**

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- Paralel **İşlemci organizasyonlarının** türlerini özetleyiniz.
- **Simetrik çoklu işlemcilerin** tasarım özelliklerine genel bir bakış sunmak.
- Çoklu işlemcili bir sistemde **önbellek tutarlılığı** konusunu anlamak.
- **MESI protokolünün** temel özelliklerini açıklayın.
- **Örtük ve açık çoklu iş parçacığı** arasındaki farkı **açıklayın**.
- **Kümeler** için temel tasarım konularını özetleyin.
- **Tekdüze olmayan bellek** erişimi kavramını açıklayın.
- **Bulut bilişim** kavramlarına genel bir bakış sunmak

Geleneksel olarak, bilgisayar sıralı bir makine olarak görülmüştür. Çoğu bilgisayar programlama dili, programının algoritmaları talimat dizileri olarak belirtmesini gerektirir. İşlemciler, makine talimatlarını bir sira halinde ve teker teker yürüterek programları . Her komut bir dizi işleme yürütülür (komutu getir, işlenenleri getir, işlemi gerçekleştir, sonuçları sakla).

Bilgisayarlarındaki bu görüş hiçbir zaman tamamen doğru olmamıştır. Mikro işlem düzeyinde, aynı anda birden fazla kontrol sinyali üretilir. En azından getirme ve yürütme işlemlerinin çakışması ölçüünde komut ardışık dizilimi uzun kullanılmaktadır. Bunların her ikisi de bağımsız işlemlerin paralel olarak gerçekleştirilemesine örnektir. Bu yaklaşım, komut düzeyinde paralelligi kullanan süperskalar organizasyon ile daha da ileri götürür. Bir süperskalar makinede, tek bir işlemci içinde birden fazla yürütme birimi ve bunlar aynı programdan birden fazla talimatı paralel olarak yürütübilir.

Bilgisayar teknolojisi gelişikçe ve bilgisayar donanımlarının maliyeti düştükçe, bilgisayar tasarımcıları genellikle performansı artırmak ve bazı durumlarda kullanılabilirliği artırmak için paralellik için giderek daha fazla fırsat aramışlardır. Genel bir bakışın ardından bu bölümde paralel organizasyona yönelik en önemli yaklaşımlardan bazıları ele alınmaktadır. İlk olarak, paralel organizasyonun en eski ve hala en yaygın örneklerinden biri olan simetrik çoklu işlemcileri (SMP'ler) inceleyeceğiz. Bir SMP organizasyonunda, birden fazla işlemci ortak bir belleği paylaşır. Bu organizasyon, ayrı bir bölümün ayrıldığı önbellek tutarlılığı sorununu ortaya çıkarır. Daha sonra, bölüm çok iş parçacıklı işlemcileri ve yonga çoklu işlemcileri incelemektedir. Daha sonra, işbirlikçi bir şekilde organize edilmiş birden fazla bağımsız bilgisayardan oluşan kümeleri tanımlıyoruz. Kümeler, tek bir SMP'nin kapasitesini aşan iş yüklerini desteklemek için giderek daha yaygın hale gelmiştir. İncelediğimiz çoklu işlemci kullanımına yönelik bir diğer yaklaşım ise tek tip olmayan bellek erişimli (NUMA) makineleremdir. NUMA yaklaşımı nispeten yenidir ve pazarda henüz kanıtlanmamıştır, ancak genellikle SMP veya kümeye yaklaşımına bir alternatif olarak düşünülmektedir. Son olarak, bu bölümde bulut bilişim mimarisi ele alınmaktadır.

## 17.1 ÇOKLU İŞLEMCİ KURULUŞLARI

### Paralel İşlemci Sistemleri Türleri

İlk olarak Flynn [FLYN72] tarafından ortaya atılan bir taksonomi, paralel işleme kapasitesine sahip sistemleri sınıflandırmanın hala en yaygın yoludur. Flynn aşağıdaki bilgisayar sistemleri kategorilerini önermiştir:

- **Tek komut, tek veri (SISD) akışı:** Tek bir işlemci, tek bir bellekte saklanan veriler üzerinde çalışmak için tek bir komut akışı yürütür. Tek işlemciler bu kategoriye girer.
- **Tek komut, çoklu veri (SIMD) akışı:** Tek bir makine talimatı, bir dizi işlem elemanının eşzamanlı olarak kilit adımda yürütülmesini kontrol eder. Her işlem elemanın ilişkili bir veri belleği vardır, böylece talimatlar farklı işlemciler tarafından farklı veri kümeleri üzerinde yürütülür. Vec-tor ve dizi işlemcileri bu kategoriye girer ve Bölüm 18.7'de tartışılmıştır.
- **Çoklu komut, tek veri (MISD) akışı:** Bir dizi veri, her biri farklı bir komut dizisini yürüten bir dizi işlemciye aktarılır. Bu yapı ticari olarak uygulanmamaktadır.
- **Çoklu komut, çoklu veri (MIMD) akışı:** Bir dizi işlemci aynı anda farklı veri setleri üzerinde farklı komut dizileri yürütür. SMP'ler, kümeler ve NUMA sistemleri bu kategoriye girer.

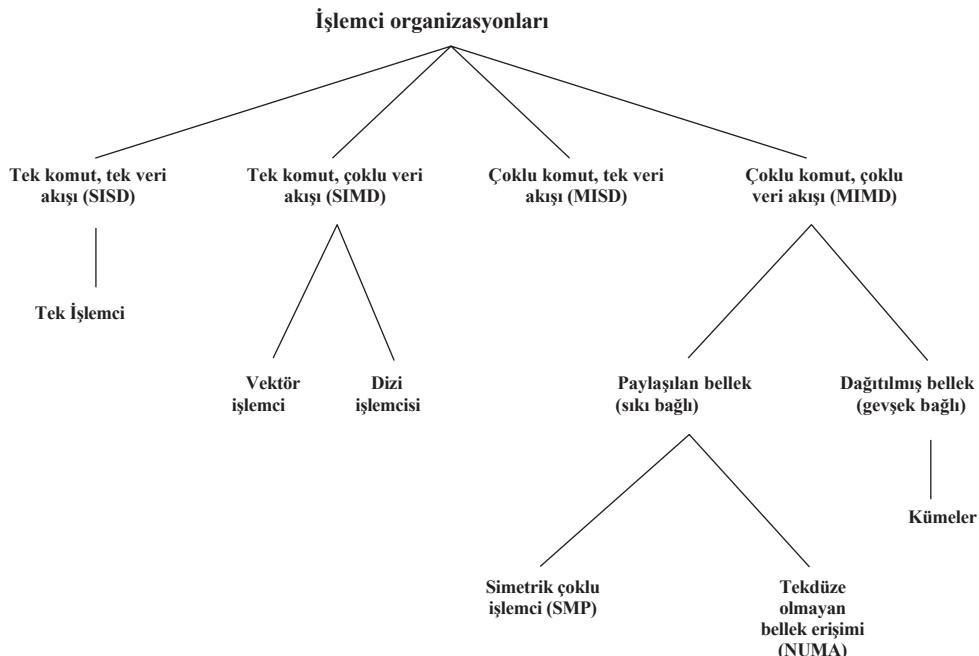
MIMD organizasyonu ile işlemciler genel amaçlıdır; her biri uygun veri aktarımını gerçekleştirmek için gerekli tüm talimatları işleyebilir. MIMD'ler işlemcilerin iletişim kurma yöntemlerine göre daha da alt böümlere ayrılabilir (Şekil 17.1). İşlemciler ortak bir belleği paylaşıyorsa, her işlemci paylaşılan bellekte saklanan programlara ve verilere erişir ve işlemciler birbirleriyle bu bellek üzerinden iletişim kurar. Bu tür bir sistemin en yaygın şekli **simetrik çoklu işlemci (SMP)** olarak bilinir ve Bölüm 17.2'de incelenecektir. Bir SMP'de birden fazla işlemci, paylaşılan bir veri yolu ya da başka bir ara bağlantı mekanizması aracılığıyla tek bir belleği ya da bellek havuzunu paylaşır; ayırt edici bir özellik, belleğin herhangi bir bölgesine erişim süresinin her işlemci için yaklaşık olarak aynı olmasıdır. Daha yeni bir gelişme ise Bölüm 17.5'te açıklanan **tek tip olmayan bellek erişimi (NUMA)** organizasyonudur. Adından da anlaşılacağı gibi, belleğin farklı bölgelerine bellek erişim süresi bir NUMA işlemcisi için farklı olabilir.

Bağımsız tek işlemcilerden veya SMP'lerden oluşan bir koleksiyon, bir **küme** oluşturmak üzere birbirine bağlanabilir. Bilgisayarlar arasındaki iletişim ya sabit yollar üzerinden ya da bazı ağ olanakları üzerinden gerçekleşir.

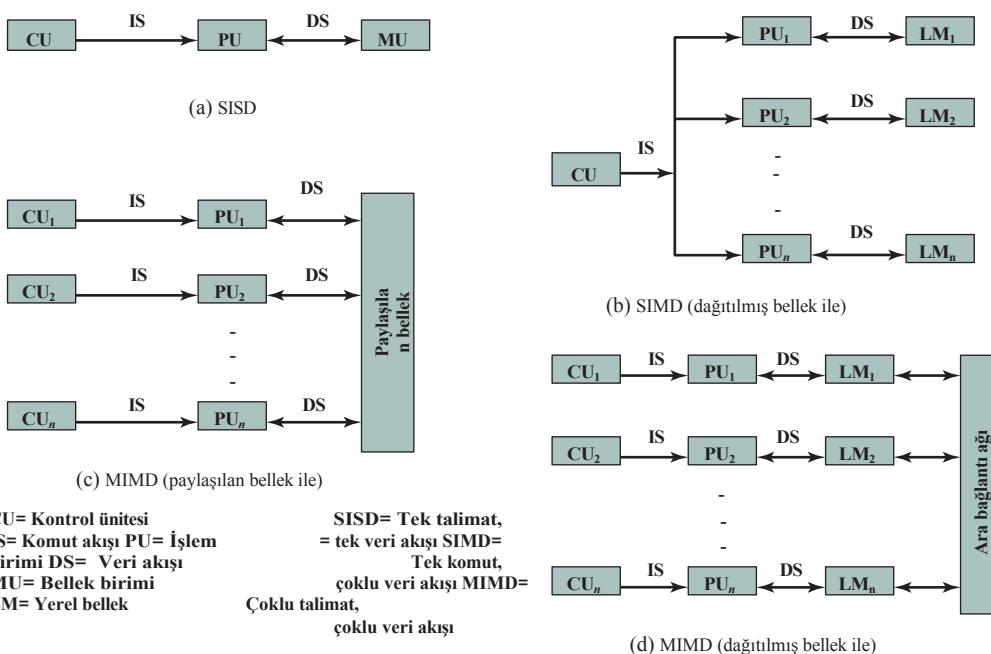
### Paralel Kuruluşlar

Şekil 17.2, Şekil 17.1'deki taksonominin genel organizasyonunu göstermektedir. Şekil 17.2a bir SISD'nin yapısını göstermektedir. Bir işlem birimine (PU) talimat akışı (IS) sağlayan bir çeşit kontrol birimi CU vardır. İşlem birimi

## 616 BÖLÜM 17 / PARALEL İŞLEM



**Şekil 17.1** Paralel İşlemci Mimarilerinin Taksonomisi



CU= Kontrol birimi  
IS= Komut akışı PU= İşlem birimi DS= Veri akışı  
MU= Bellek birimi LM= Yerel bellek

SISD= Tek talimat,  
= tek veri akışı SIMD= Tek komut,  
çoklu veri akışı MIMD= Çoklu talimat,  
çoklu veri akışı

**Şekil 17.2** Alternatif Bilgisayar Organizasyonları

birim, bir bellek biriminden (MU) gelen tek bir veri akışı (DS) üzerinde çalışır. SIMD'de hala tek bir kontrol birimi vardır, şimdi birden fazla PU'ya tek bir komut akışı beslemektedir. Her PU kendi özel belleğine sahip olabilir (Şekil 17.2b'de gösterilmiştir) veya paylaşılan bir olabilir. Son olarak, MIMD ile, her biri kendi PU'suna ayrı bir komut akışı besleyen birden fazla kontrol birimi vardır. MIMD paylaşılan bellekli çoklu işlemci (Şekil 17.2c) ya da dağıtılmış bellekli çoklu bilgisayar (Şekil 17.2d) olabilir.

SMP'ler, kümeler ve NUMA'larda ilgili tasarım sorunları karmaşıktır; fiziksel organizasyon, ara bağlantı yapıları, işlemciler arası iletişim, işletim sistemi tasarımları ve uygulama yazılımı teknikleriyle ilgili konuları içerir. GGiletim sistemi tasarımları konularına kısaca değinmemizde rağmen buradaki kaygımız öncelikle organizasyonla ilgilidir.

## 17.2 SIMETRİK ÇOKLU İŞLEMCİLER

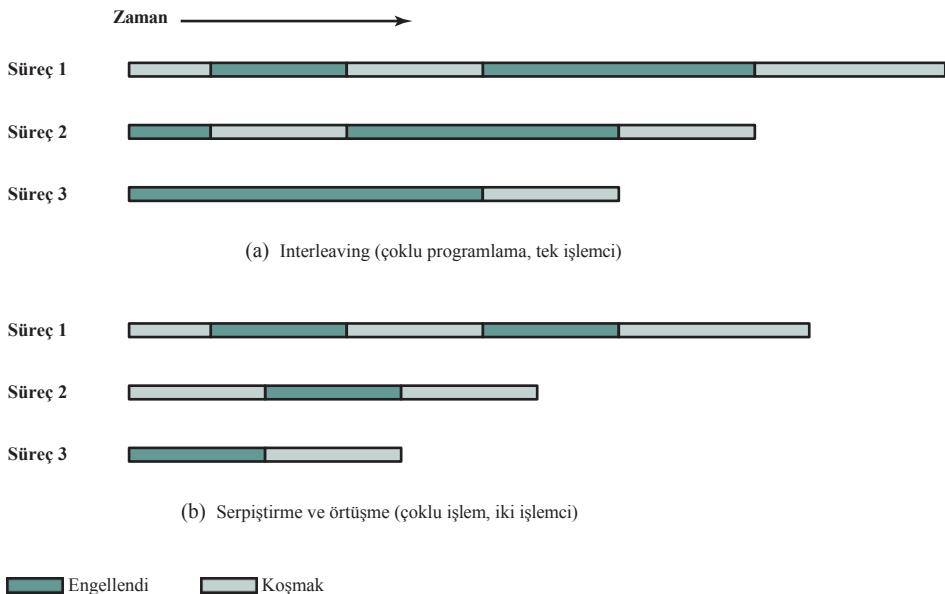
Yakın zamana kadar, neredeyse tüm tek kullanıcılı kişisel bilgisayarlar ve çoğu iş istasyonu tek bir genel amaçlı mikroişlemci içeriyordu. Performans talepleri arttıkça ve mikroişlemcilerin maliyeti düşmeye devam, satıcılar SMP organizasyonuna sahip sistemleri piyasaya sürmüştür. SMP terimi bir bilgisayar donanım mimarisini ve aynı zamanda bu mimariyi yansıtan işletim sistemi davranışını ifade eder. Bir SMP aşağıdaki özelliklere sahip bağımsız bir bilgisayar sistemi olarak tanımlanabilir:

1. Karşılaştırılabilir kapasitede iki veya daha fazla benzer işlemci vardır.
2. Bu işlemciler aynı ana belleği ve G/C olanaklarını paylaşır ve bellek erişim süresi her işlemci için yaklaşık olarak aynı olacak şekilde bir veri yolu veya başka bir dahili bağlantı şeması ile birbirine bağlanır.
3. Tüm işlemciler ya aynı kanallar üzerinden ya da aynı cihaza giden yolları sağlayan farklı kanallar üzerinden G/C cihazlarına erişimi paylaşır.
4. Tüm işlemciler aynı işlevleri yerine getirebilir (dolayısıyla *simetrik* terimi kullanılır).
5. Sistem, işlemciler ve programları arasında iş, görev, dosya ve veri öğesi seviyelerinde etkileşim sağlayan entegre bir işletim sistemi tarafından kontrol edilir.

1'den 4'e kadar olan noktalar kendi kendini açıklayıcı olmalıdır. Madde 5, bir kümeye gibi gevşek bir şekilde bağlanmış çoklu işlem sistemiyle olan zıtlıklardan birini göstermektedir. İkincisinde, etkileşimin fiziksel birimi genellikle bir mesaj ya da tam bir dosyadır. Bir SMP'de, etkileşim seviyesini bireysel veri öğeleri oluşturabilir ve süreçler arasında yüksek derecede işbirliği olabilir.

Bir SMP'nin işletim sistemi süreçleri veya iş parçacıklarını tüm işlemciler arasında programlar. Bir SMP organizasyonu, tek işlemcili bir organizasyona göre aşağıdakiler de dahil olmak üzere bir dizi potansiyel avantaja sahiptir:

- **Performans:** Bir bilgisayar tarafından yapılacak iş, işin bazı bölümlerinin paralel olarak yapılabileceği düzenlenebilirse, birden fazla işlemciye sahip bir sistem, aynı türden tek bir işlemciye sahip bir daha yüksek performans sağlayacaktır (Şekil 17.3).



### **Sekil 17.3 Coklu Programlama ve Coklu Islem**

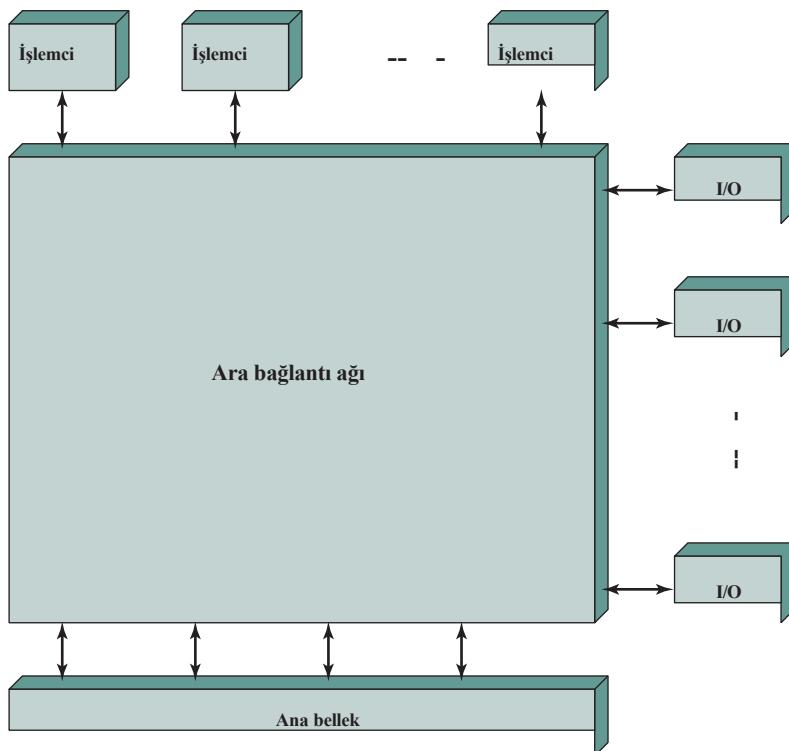
- Kullanılabilirlik:** Simetrik çoklu işlemcilerde, tüm işlemciler aynı işlevleri yerine getirebildiğinden, tek bir işlemcinin arızalanması makineyi durdurmaz. Bunun yerine, sistem düşük performansla çalışmaya devam edebilir.
  - Artımlı büyümeye:** Bir kullanıcı ek bir işlemci ekbir sistemin performansını artırabilir.
  - Ölçeklendirme:** Satıcılar, sistemde yapılandırılan işlemci sayısına bağlı olarak farklı fiyat ve performans özelliklerine sahip bir dizi ürün sunabilir.

Bunların garanti edilen faydalardan ziyade potansiyel faydalar olduğuna dikkat etmek önemlidir. İşletim sistemi, bir SMP sistemindeki paralellikten faydalananmak için araçlar ve islevler sağlanmalıdır.

SMP'nin çekici bir özelliği, birden fazla işlemcinin varlığının kullanıcı için şeffaf olmasıdır. Gölgen sistemi, iş parçacıklarının ya da süreçlerin ayrı ayrı iGlemciler üzerinde zamanlanmasını ve iGlemciler arasında senkronizasyonu sağlar.

## Organizasyon

Şekil 17.4 genel hatlarıyla çok işlemcili bir sistemin organizasyonunu göstermektedir. İki ya da daha fazla işlemci vardır. Her işlemci, bir kontrol birimi, ALU, yazmaçlar ve tipik olarak bir veya daha fazla önbellek seviyesi içeren bağımsız bir yapıya sahiptir. Her işlemci, bir çeşit ara bağlantı mekanizması aracılığıyla paylaşılan bir ana belleğe ve G/C aygıtlarına erişebilir. İşlemciler birbirleriyle bellek aracılığıyla iletişim kurabilirler (ortak veri alanlarında bırakılan mesajlar ve durum bilgileri). İşlemcilerin doğrudan sinyal alışverisi yapması da mümkün olabilir. Bellek genellikle şu şekilde organize edilir



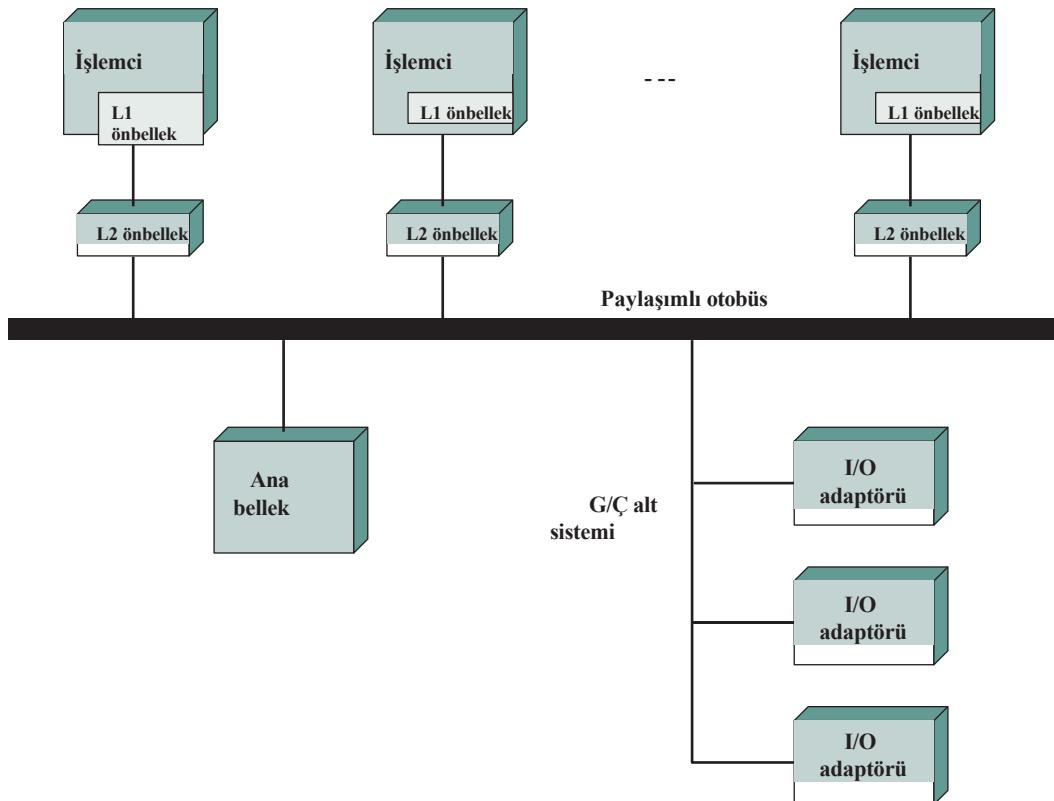
**Şekil 17.4** Sıkı Bağlantılı Çoklu İşlemcinin Genel Blok Diyagramı

Böylece ayrı bellek bloklarına birden fazla eşzamanlı erişim mümkün olur. Bazı konfigürasyonlarda, her işlemci paylaşılan kaynaklara ek olarak kendi özel ana belleğine ve G/C kanallarına da sahip olabilir.

Kişiisel bilgisayarlar, iş istasyonları ve sunucular için en yaygın organizasyon zaman paylaşımı veriyoldur. Zaman paylaşımı veri yolu, çok işlemcili bir sistem oluşturmak için en basit mekanizmadır (Şekil 17.5). Yapı ve arayüzler temelde bir veri yolu ara bağlantısı kullanan tek işlemcili bir sistemle aynıdır. Veri yolu kontrol, adres ve veri hatlarından oluşur. G/C alt sistemlerinden işlemcilere DMA aktarımlarını kolaylaşdırmak için aşağıdaki özellikler sağlanmıştır:

- **Adresleme:** Verilerin kaynağını ve hedefini belirlemek için veri yolu üzerindeki modüller ayırt etmek mümkün olmalıdır.
- **Arbitrasyon:** Herhangi bir I/O modülü geçici olarak "master" olarak işlev görebilir. Bir tür öncelik şeması kullanarak veri yolu kontrolü için rekabet eden talepleri tahkim etmek için bir mekanizma sağlanmıştır.
- **Zaman paylaşımı:** Bir modül veri yolunu kontrol ederken, diğer modüler kilitlenir ve gerekirse veri yolu erişimi sağlanana kadar çalışmaya askıya almalıdır.

Bu tek işlemcili özellikler bir SMP organizasyonunda doğrudan kullanılabilir. Bu ikinci durumda, artık birden fazla işlemci ve birden fazla G/C işlemcisi vardır ve bunların hepsi veri yolu üzerinden bir veya daha fazla bellek modülüne erişmeye çalışır.



**Şekil 17.5** Simetrik Çoklu İşlemci Organizasyonu

Otogüs organizasyonu birçok cazip özelliğe sahiptir:

- **Basitlik:** Bu, çoklu işlemci organizasyonuna en basit yaklaşımındır. Her işlemcinin fiziksel arayüzü ve adresleme, tahlim ve zaman paylaşımı mantığı tek işlemcili bir sistemdekiyle aynı kalır.
- **Esneklik:** Veri yoluna daha fazla ekleyerek sistemi genişletmek genellikle kolaydır.
- **Güvenilirlik:** Veri yolu esasen pasif bir ortamdır ve bağlı herhangi bir cihazın arızalanması tüm sistemin arızalanmasına neden olmamalıdır.

Veri yolu organizasyonunun ana dezavantajı performansıdır. Tüm bellek referansları ortak veri yolundan geçer. Bu nedenle, veri yolu çevrim süresi sistemin hızını sınırlar. Performansı artırmak için her bir işlemcinin bir ön ile donatılması arzu edilir. Bu, veri yolu erişimlerinin sayısını önemli ölçüde azaltacaktır. Tipik olarak, iş istasyonu ve PC SMP'lerinde L1 önbelleği dahili (işlemciye aynı yonga) ve L2 önbelleği dahili ya da harici olmak üzere iki önbellek seviyesi bulunur. Bazı işlemciler artık bir L3 önbellek de kullanmaktadır.

Önbellek kullanımı bazı yeni tasarım hususlarını beraberinde getirir. Her yerel önbellek, belleğin bir bölümünün görüntüsünü içerdiginden, bir sözcük diğerinde değiştirilirse

bir sözcüğü geçersiz kılabılır. Bunu önlemek için diğer işlemcilerin bir güncellemenin gerçekleştiği konusunda uyarılması gerekdir. Bu sorun *önbellek tutarlılığı* sorunu olarak bilinir ve genellikle işletim sisteminde ziyade donanımda ele alınır. Bu konuyu Bölüm 17.4'te ele alıyoruz.

#### Çok İşlemcili İşletim Sistemi Tasarımında Dikkat Edilmesi Gerekenler

Bir SMP işletim sistemi işlemci ve diğer bilgisayar kaynaklarını yönetir, kullanıcı sistem kaynaklarını kontrol eden tek bir işletim sistemi algılar. Aslında, böyle bir yapılandırma tek işlemcili çok programlı bir sistem olarak görünmelidir. Hem SMP hem de tek işlemcili durumlarda, aynı anda birden fazla iş veya süreç aktif olabilir ve bunların yürütülmüşünü planlamak ve kaynakları tahsis etmek işletim sisteminin sorumluluğundadır. Bir kullanıcı, tek bir işlemcinin mi yoksa birden fazla işlemcinin mi mevcut olacağına baksızın birden fazla işlem veya işlemler içinde birden fazla iş parçacığı kullanan uygulamalar oluşturabilir. Bu nedenle, çok işlemcili bir işletim sistemi, çok programlı bir sistemin tüm işlevsellüğünün yanı sıra birden fazla işlemciyi barındıracak ek özellikler de sağlamalıdır. Temel tasarım konuları arasında:

- **Eşzamanlı eşzamanlı süreçler:** Birden fazla işlemcinin aynı IS kodunu eş zamanlı olarak yürütmesine izin vermek için işletim sistemi rutinlerinin yeniden girilebilir olması gerekdir. Birden fazla işlemci işletim sisteminin aynı ya da farklı bölümlerini çalıştırırken, kilitlenme ya da geçersiz işlemleri önlemek için işletim sistemi tabloları ve yönetim yapıları düzgün bir şekilde yönetilmelidir.
- **Çizelgeleme:** Herhangi bir işlemci çizelgeleme yapabilir, bu nedenle çakışmalardan kaçınılmalıdır. Zamanlayıcı, hazır süreçleri mevcut işlemcilere atamalıdır.
- **Senkronizasyon:** Paylaşılan adres alanlarına veya paylaşılan G/C kaynaklarına potansiyel erişimi olan birden fazla aktif, etkili senkronizasyon sağlamak için dikkatli olunmalıdır. Senkronizasyon, karşılıklı dışlama ve olay sıralamasını zorlayan bir olanaktır.
- **Bellek yönetimi:** Çok işlemcili bir makinede bellek yönetimi, Bölüm 8'de tartışıldığı gibi tek işlemcili makinelerde bulunan tüm sorunlarla ilgilenmelidir. Buna ek olarak, işletim sisteminin en iyi performansı elde etmek için çoklu bellekler gibi mevcut donanım paralellliğinden yararlanması gerekdir. Farklı işlemcilerdeki sayfalama mekanizmaları, birden fazla işlemci bir sayfayı ya da segmenti paylaştığında tutarlılığı sağlamak ve sayfa değişimine karar vermek için koordine edilmelidir.
- **Güvenilirlik ve hata toleransı:** İşletim sistemi, işlemci arızası karşısında zarif bir bozulma sağlamalıdır. Zamanlayıcı ve işletim sisteminin diğer bölümleri bir işlemcinin kaybını tanımlı ve yönetim tablolarını buna göre yeniden yapılandırmalıdır.

#### 17.3 ÖNBELLEK TUTARLILIĞI VE MESİ PROTOKOLÜ

Çağdaş çok işlemcili sistemlerde, her işlemciyle ilişkili bir veya iki önbellek seviyesine sahip olmak gelenekseldir. Bu organizasyon, makul bir performans elde etmek için gereklidir. Bununla birlikte, *önbellek tutarlılığı* olarak bilinen bir sorun yaratır

problem. Sorunun özü şudur: Aynı verinin birden fazla kopyası aynı anda farklı önbelleklerde bulunabilir ve işlemcilerin kendi kopyalarını serbestçe güncellemelerine izin verilirse, tutarsız bir bellek görünümü ortaya çıkabilir. Bölüm 4'te iki yaygın yazma politikası tanımladık:

- **Geri yazma:** Yazma işlemleri genellikle sadece önbelleğe yapılır. Ana bellek yalnızca ilgili önbellek satırı önbellekten çıkarıldığında güncellenir.
- **Yazma işlemi:** Tüm yazma işlemleri ana belleğin yanı sıra önbelleğe de yapılır ve ana belleğin her zaman geçerli olması sağlanır.

Geri yazma politikasının tutarsızlığa neden olabileceği açıklıktır. İki önbellek aynı satırı içeriye veya satır bir önbellekte güncellenirse, diğer önbellek bilmeden geçersiz bir değere sahip olacaktır. Bu geçersiz satırda yapılan sonraki okumalar geçersiz sonuçlar üretir. Doğrudan yazma ilkesiyle bile, diğer önbellekler bellek trafigini izlemediği veya güncellemeyle ilgili doğrudan bir bildirim almadiği sürece tutarsızlık meydana gelebilir.

Bu bölümde, önbellek eşgüdüm sorununa yönelik çeşitli yaklaşımları kısaca inceleyeceğiz ve ardından en yaygın kullanılan yaklaşıma odaklanacağız: MESI (değiştirilmiş/özel/paylaşılan/geçersiz) protokolü. Bu protokolün bir versiyonu hem x86 mimarisinde kullanılmaktadır.

Herhangi bir önbellek tutarlılığı protokolü için amaç, son kullanılan yerel değişkenlerin uygun önbelleğe girmesine ve çok sayıda okuma ve yazma işlemi boyunca orada kalmasına izin verirken, aynı anda birden fazla önbellekte olabilecek paylaşılan değişkenlerin tutarlığını korumak için protokolü kullanmaktadır. Önbellek tutarlılığı yaklaşımları genellikle yazılım ve donanım yaklaşımları olarak ikiye ayrılır. Bazı uygulamalar hem yazılım hem de donanım unsurlarını içeren bir strateji benimsedektedir. Bununla birlikte, yazılım ve donanım yaklaşımları olarak yapılan sınıflandırma hala öğreticidir ve önbellek tutarlılığı stratejilerinin incelenmesinde yaygın olarak kullanılmaktadır.

## **Yazılım Çözümleri**

Yazılım önbellek tutarlılığı şemaları, sorunla başa çıkmak için derleyiciye ve işletim sisteme güvenerek ek donanım devresi ve mantık ihtiyacından kaçınmaya çalışır. Yazılım yaklaşımları cayıptır çünkü olası sorunları tespit etme yükü çalışma zamanından derleme zamanına aktarılır ve tasarım karmaşaklığını donanımdan yazılıma aktarılır. Öte yandan, derleme zamanı yazılım yaklaşımları genellikle muhafazakar kararlar vermek zorundadır ve bu da verimsiz önbellek kullanımına yol açar.

Derleyici tabanlı tutarlılık mekanizmaları, hangi veri öğelerinin önbellekleme için güvenli olmayacağı belirlemek için kod üzerinde bir analiz gerçekleştirir ve bu öğeleri buna göre işaretler. İşletim sistemi veya donanım daha sonra önbelleğe alınamayan öğelerin önbelleğe alınmasını engeller.

En basit yaklaşım, paylaşılan veri değişkenlerinin önbelleğe alınmasını önlemektir. Bu çok ihtiyatlı bir yaklaşımdır çünkü paylaşılan bir veri yapısı bazı dönemlerde yalnızca kullanılırken diğer dönemlerde etkin bir şekilde salt okunur olabilir. Yalnızca en az bir sürecin değişkeni güncelleyebileceği ve en az bir diğer sürecin değişkene erişebileceği dönemlerde önbellek tutarlılığı bir sorun teşkil eder.

Daha verimli yaklaşımlar, paylaşılan değişkenler için güvenli dönemleri belirlemek üzere kodu analiz eder. Derleyici daha sonra kritik dönemlerde önbellek tutarlığını zorlamak için oluşturulmuş koda talimatlar ekler. Analizi gerçekleştirmek ve sonuçları zorlamak için bir dizi teknik geliştirilmiştir; incelemeler için [LILJ93] ve [STEN90]'a bakın.

## Donanım Çözümleri

Donanım tabanlı çözümler genellikle önbellek tutarlığını protokoller olarak adlandırılır. Bu çözümler, potansiyel tutarsızlık durumlarının çalışma zamanında dinamik olarak tanımmasını sağlar. Sorun yalnızca gerçekten ortaya çıktığında ele alındığından, önbelleklerin daha etkin kullanımı söz konusudur ve bu da yazılım yaklaşımına kıyasla performansın artmasını sağlar. Ayrıca bu yaklaşım programcı ve derleyici için şeffaftır ve yazılım geliştirme yükünü azaltır.

Donanım şemaları, veri hatlarıyla ilgili durum bilgilerinin nerede tutulduğu, bu bilgilerin nasıl organize edildiği, uyumun nerede sağlandığı ve uygulama mekanizmaları dahil olmak üzere bir dizi ayrıntıda farklılık gösterir. Genel olarak, donanım şemaları iki kategoriye ayrılabilir: **dizin protokollerı ve snoopy protokollerı**.

**DİZİN PROTOKOLLERİ** Dizin protokollerı, satırların kopyalarının nerede bulunduğu hakkında bilgi toplar ve bu bilgileri korur. Tipik olarak, ana bellek denetleyicisinin bir parçası olan merkezi bir denetleyici ve ana bellekte depolanan bir dizin vardır. Dizin, çeşitli yerel önbelleklerin içerikleri hakkında genel durum bilgileri içerir. Bir önbellek denetleyicisi bir talepte bulunduğu anda, merkezi denetleyici bellek ve önbellekler arasında veya önbellekler arasında veri aktarımı için gerekli komutları kontrol eder ve yayınlar. Ayrıca durum bilgisini güncel tutmaktadır ve sorguludur; bu nedenle, bir satırın global durumunu etkileyebilecek her yerel eylem merkezi denetleyiciye bildirilmelidir.

Tipik olarak, denetleyici hangi işlemcilerin hangi satırların bir kopyasına sahip olduğu bilgisini tutar. Bir işlemci bir satırın yerel kopyasına yazmadan önce, denetleyiciden satra özel erişim talep etmelidir. Bu özel erişimi vermeden önce denetleyici, bu satırın önbelleğe alınmış bir kopyasına sahip tüm işlemcilere bir mesaj göndererek her işlemciyi kendi kopyasını geçersiz kılmaya zorlar. Bu tür işlemcilerin her birinden geri bildirim aldıktan sonra, denetleyici talep eden işlemciye özel erişim izni verir. Başka bir işlemci, başka bir işlemciye özel olarak verilen bir satırı okumaya çalıştığında, denetleyiciye bir kaçırma bildirimi gönderir. Kontrolör daha sonra o satırı tutan işlemciye, işlemcinin ana belleğe geri yazmasını gerektiren bir komut verir. Hat artık orijinal işlemci ve talep eden işlemci tarafından okunmak üzere paylaşılabilir.

Dizin şemaları merkezi bir darboğazın ve çeşitli önbellek denetleyicileri ile merkezi denetleyici arasındaki iletişim yükünün dezavantajlarından muzdariptir. Bununla birlikte, birden fazla veri yolu veya başka bir karmaşık ara bağlantı şeması içeren büyük ölçekli sistemlerde etkilidirler.

**SNOOPY PROTOKOLLERİ** Snoopy protokollerı, önbellek tutarlığını sürdürme sorumluluğunu bir çoklu işlemcideki tüm önbellek denetleyicileri arasında dağıtır. Bir önbellek, tuttuğu bir satırın diğer önbellek denetleyicileriyle paylaşıldığını fark etmelidir.

önbellekler. Paylaşılan bir önbellek satırında bir güncelleme işlemi gerçekleştirildiğinde, bunun bir yayım mekanizması ile diğer tüm önbelleklere duyurulması gereklidir. Her önbellek denetleyicisi bu yayım bildirimlerini gözlemlerek için ağ üzerinde "snoop" yapabilir ve buna göre tepki verebilir.

Snoopy protokolleri veri yolu tabanlı çoklu işlemciler için idealdir, çünkü paylaşılan veri yolu yayılma ve gözetleme için basit bir araç sağlar. Bununla birlikte, yerel önbellek kullanımının amaçlarından biri veri yolu erişimlerinden kaçınmak olduğundan, yayın ve gözetleme için gerekli artan veri yolu trafığının yerel önbellek kullanımından elde edilen kazanımları ortadan kaldırılmamasına dikkat edilmelidir.

Snoopy protokolüne yönelik iki temel yaklaşım araştırılmıştır: yazma geçersiz kılma ve yazma güncelleme (veya yazma yayımı). Yazma geçersiz kılma protokolünde, birden fazla okuyucu olabilir ancak aynı anda yalnızca bir yazar olabilir. Başlangıçta, bir satır okuma amacıyla birkaç önbellek arasında paylaştırılabilir. Önbelleklerden biri satırı yazmak istediğiinde, önce diğer önbelleklerde o satırı geçersiz kılan bir bildirim yayınlar ve satırı yazma önbelleğine özel hale getirir. Satır ayrıcalıklı hale geldiğinde, sahibi olan işlemci, başka bir işlemci aynı satırı ihtiyaç duyana kadar ucuz yerel yazmalar yapabilir.

Yazma-güncelleme protokolü ile birden fazla okuyucunun yanı sıra birden fazla yazar da olabilir. Bir işlemci paylaşılan bir satırı güncellemek istediğiinde, güncellenecek kelime diğerlerine dağıtılr ve bu satırı içeren önbellekler onu güncelleyebilir.

Bu iki yaklaşımın hiçbirinin tüm koşullar altında diğerine üstün değildir. Performans, yerel önbelleklerin sayısına ve bellek okuma ve yazma düzeneğine bağlıdır. Bazı sistemler hem yazma-geçersiz kılma hem de yazma-güncelleme mekanizmalarını kullanan uygulanabilir protokoller uygular.

Geçersiz yazma yaklaşımı, x86 mimaris gibi ticari çoklu işlemci sistemlerinde en yaygın kullanılanıdır. Her önbellek satırının durumunu (önbellek etiketindeki iki ekstra biti kullanarak) değiştirilmiş, özel, paylaşılan veya geçersiz olarak işaretler. Bu nedenle, yazma-geçersiz kılma protokolü MESI olarak adlandırılır. Bu bölümün geri kalanında, çoklu işlemcideki yerel önbellekler arasında kullanımına bakacağız. Sunumda basitlik sağlamak için, hem seviye 1 hem de seviye 2 arasında yerel olarak ve aynı zamanda dağıtılmış çoklu işlemci boyunca koetme ile ilgili mekanizmaları incelemiyoruz. Bu yeni bir ilke eklemeyecek ancak tartışmayı büyük ölçüde karmaşıklaştıracaktır.

### MESI Protokolü

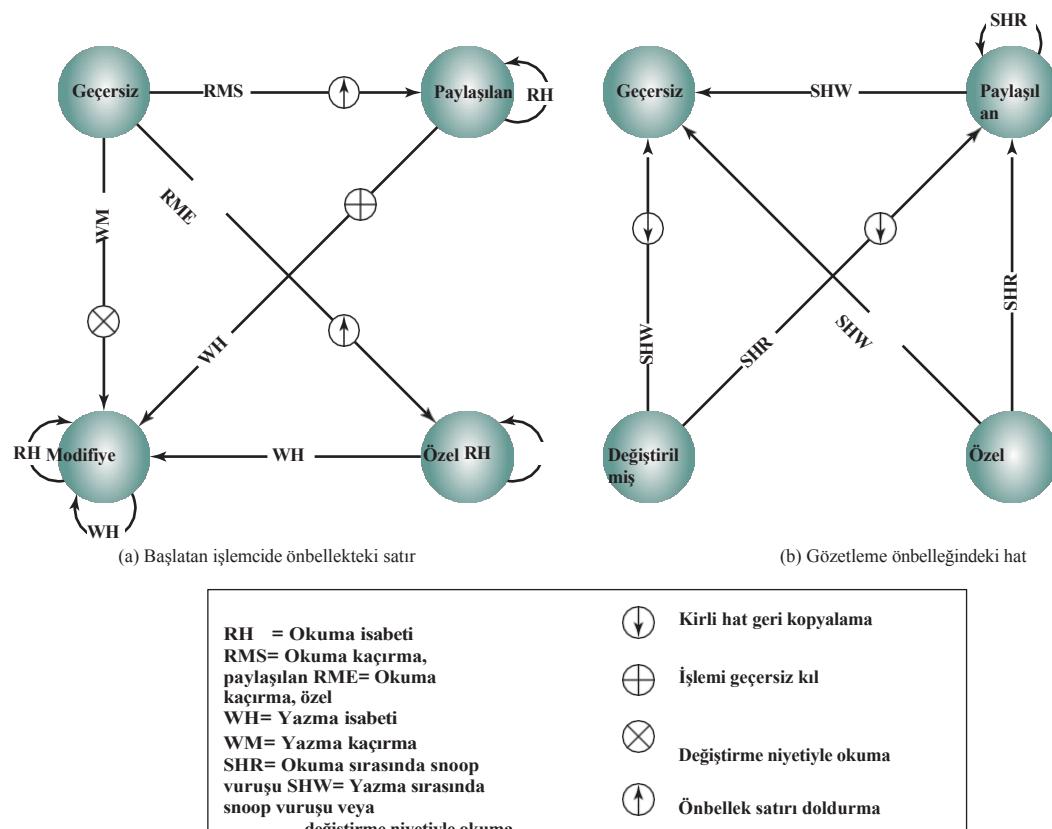
Bir SMP'de önbellek tutarlılığı sağlamak için veri önbelleği genellikle MESI olarak bilinen bir protokolü destekler. MESI için veri önbelleği etiket başına iki durum biti içerir, her satır dört durumdan birinde olabilir:

- **Değiştirilmiş:** Önbellekteki satır değiştirilmiştir (ana bellekten farklı olarak) ve yalnızca bu önbellekte kullanılabilir.
- **Özel:** Önbellekteki satır ana bellekte ile aynıdır ve başka hiçbir önbellekte bulunmaz.
- **Paylaşılan:** Önbellekteki satır ana bellekte ile aynıdır ve başka bir önbellekte de bulunabilir.
- **Geçersiz:** Önbellekteki satır geçerli veri içermiyor.

Tablo 17.1 MESI Önbellek Hattı Durumları

	M Değiştirilmiş	E Özel	S Paylaşılan	I Geçersiz
Bu önbellek satırı geçerli mi?	Evet	Evet	Evet	Hayır
Bellek kopyası ...	güncel değil	geçerli	geçerli	-
Diğer önbelleklerde kopyaları var mı?	Hayır	Hayır	Belki de	Belki de
Bu hatta bir yazı ...	otobüse gitmiyor	otobüse gitmiyor	veri yoluna gider ve önbelleğin günceller	doğrudan otobüse gider

Tablo 17.1 dört durumun anlamını özetlemektedir. Şekil 17.6'da MESI protokolü için bir durum diyagramı gösterilmektedir. Önbelleğin her satırının kendi durum bitlerine ve dolayısıyla durum diyagramının kendi gerçeklemesine sahip olduğunu unutmayın. Şekil 17.6a, bu önbelleğe bağlı işlemci tarafından başlatılan eylemler nedeniyle meydana gelen geçişleri göstermektedir. Şekil 17.6b, ortak veri yolu üzerinde gözetlenen olaylar nedeniyle meydana gelen geçişleri göstermektedir. İşlemci tarafından başlatılan ve veri yolu tarafından başlatılan eylemler için ayrı durum diyagramlarının bu şekilde sunulması, MESI'nin mantığını netleştirmeye yardımcı olur.



Şekil 17.6 MESI Durum Geçiş Diyagramı

## 626 BÖLÜM 17 / PARALEL İŞLEM

protokolü. Herhangi bir zamanda bir önbellek satırı tek bir durumdadır. Eğer bir sonraki olay bağlı işlemciden geliyorsa, geçiş Şekil 17.6a ile belirlenir ve eğer bir sonraki olay veri yolundan geliyorsa, geçiş Şekil 17.6b ile belirlenir. Bu geçişleri daha ayrıntılı olarak inceleyelim.

**READ MISS** Yerel önbellekte bir read miss olduğunda, işlemci eksik adresi içeren ana bellek satırını okumak için bir bellek okuması başlatır. İşlemci veri yoluna bir sinyal ekleyerek diğer tüm işlemci/önbellek birimlerini işlemi gözetlemeleri için uyarır. Bir dizi olası sonuç vardır:

- Başka bir önbellekte satırın temiz (okunduğundan beri değiştirilmemiş) bir kopyası özel durumda ise, bu satırı paylaştığını belirten bir sinyal döndürür. Yanıt veren işlemci daha sonra kopyasının durumunu dışlayıcıdan paylaşımı geçirir ve başlattan işlemci satırı ana bellekten okur ve önbelleğindeki satırı geçersizden paylaşımı geçirir.
- Bir veya daha fazla önbellekte paylaşılan durumdaki satırın temiz bir kopyası varsa, her biri paylaştığını bildirir. Başlattan işlemci satırı okur ve önbelleğindeki satırı geçersiz durumdan paylaşılan duruma geçirir.
- Başka bir önbellekte satırın değiştirilmiş bir kopyası varsa, bu önbellek bellek okumasını engeller ve satırı paylaşılan veri yolu üzerinden talep eden önbelleğe sağlar. Yanıt veren önbellek daha sonra satırını değiştirilmişten paylaşılana değiştirir.<sup>1</sup> Talep eden önbelleğe gönderilen satır, bloğu bellekte saklayan bellek denetleyicisi tarafından da alınır ve işlenir.
- Başka hiçbir önbellekte satırın bir kopyası yoksa (temiz veya değiştirilmiş), hiçbir sinyal döndürülmez. Başlattan işlemci satırı okur ve kendi önbelleğindeki satırı geçersizden özele geçirir.

**READ HIT** Yerel önbellekte bulunan bir satırda bir read hit olduğunda, işlemci sadece gerekli öğeyi okur. Durum değişikliği olmaz: Durum değiştirilmiş, paylaşılmış veya özel olarak kalır.

**YAZMA HATASI** Yerel önbellekte bir yazma hatası olduğunda, işlemci eksik adresi içeren ana bellek satırını okumak için bir bellek okuması başlatır. Bu amaçla, işlemci veri yolu üzerinde *okuma niyetiyle değiştirme* (RWITM) anlamına gelen bir sinyal yayınları. Satır yüklenliğinde, hemen değiştirildi olarak işaretlenir. Diğer ilgili olarak, veri satırının yüklenmesinden önce iki olası senaryo vardır.

İlk olarak, başka bir önbellek bu satırın değiştirilmiş bir kopyasına sahip olabilir (durum= modify). Bu durumda, uyarılan işlemci başlattan işlemciye başka bir işlemcinin satırın değiştirilmiş bir kopyasına sahip olduğunu bildirir. Başlattan işlemci veriyolunu teslim eder ve bekler. Diğer işlemci veri yoluna erişim kazanır, değiştirilmiş önbelleği yazar

---

<sup>1</sup> Bazi uygulamalarda, değiştirilmiş satıra sahip önbellek, başlattan işlemciye yeniden deneme sinyali verir. Bu sırada, değiştirilmiş kopyaya sahip işlemci veri yolu ele geçirir, değiştirilmiş satırı ana belleğe geri yazar ve önbelleğindeki satırı değiştirilmişten paylaşılana geçirir. Daha sonra, talepte bulunan işlemci tekrar dener ve bir önceki noktada açıkladığı gibi, bir veya daha fazla işlemcinin paylaşılan durumdaki satırın temiz bir kopyasına sahip olduğunu görür.

satırını ana belleğe geri gönderir ve önbellek satırının durumunu geçersiz hale getirir (çünkü başlatan işlemci bu satırı değiştirecektir). Daha sonra, başlatan işlemci tekrar RWITM veriyoluna bir sinyal gönderecek ve ardından satırı ana bellekten okuyacak, önbellekteki satırı değiştirecek ve satırı değiştirilmiş durumda işaretleyecektir.

İkinci senaryo, başka hiçbir önbellekte istenen satırın değiştirilmiş bir kopyasının bulunmamasıdır. Bu durumda, hiçbir sinyal döndürülmez ve başlatan işlemci satırı okumaya ve değiştirmeye devam eder. Bu arada, bir veya daha fazla önbellekte satırın paylaşılan durumda temiz bir kopyası varsa, her önbellek satırın kopyasını geçersiz kılar ve bir önbellekte satırın özel durumda temiz bir kopyası varsa, satırın kopyasını geçersiz kılar.

**YAZMA İSABETİ** O anda yerel önbellekte bulunan bir satırda bir yazma isabeti meydana geldiğinde, etki yerel önbellekteki o satırın mevcut durumuna bağlıdır:

- **Paylaşıldı:** Güncellemeyi gerçekleştirmeden önce, işlemci hattın özel sahipliğini kazanmalıdır. İşlemci veriyolunda niyetini bildirir. Önbelleğinde hattın paylaşılan bir kopyasına sahip olan her işlemci sektörü paylaşılandan geçersiz olana geçirir. Başlatan işlemci daha sonra güncellemeyi gerçekleştirir ve satırın kendi kopyasını paylaşılandan değiştirilmiş olana geçirir.
- **Özel:** İşlemci bu satırın özel kontrolüne zaten sahiptir ve bu nedenle güncellemeyi gerçekleştirir ve satırın kopyasını değiştirilmiş olana geçirir.
- **Değiştirildi:** İşlemci zaten bu satırın özel kontrolüne sahiptir ve satır değiştirilmiş olarak işaretlenmiştir ve bu nedenle güncellemeyi gerçekleştirir.

**L1-L2** ÖNBELEK TUTARLILIĞI Şimdiye kadar önbellek tutarlılık protokollerini aynı veri yoluna veya diğer SMP ara bağlantı tesisine bağlı önbellekler arasındaki işbirliği faaliyeti açısından tanımladık. Tipik olarak, bu önbellekler L2 önbelleklərdir ve her işlemci ayrıca doğrudan veri yoluna bağlanmayan ve bu nedenle bir snoopy protokolüne katılmayan bir L1 önbelleğe sahiptir. Bu nedenle, her iki önbellek seviyesinde ve SMP yapılandırmasındaki tüm önbelleklerde veri bütünlüğünü korumak için bazı şemalara ihtiyaç vardır.

Strateji, MESİ protokolünü (veya herhangi bir önbellek tutarlılık protokolünü) L1 önbelleklerine genişletmektedir. Böylece, L1 önbelleğindeki her satır durumu gösteren bitler içerir. Temelde amaç : hem L2 önbelleğinde hem de ona karşılık gelen L1 bulunan herhangi bir satır için L1 satırının durumu L2 satırının durumunu takip etmelidir. Bunu yapmanın basit bir yolu, L1 önbelleğinde yazma politikasını benimsemektir; bu durumda yazma işlemi belleğe değil L2 önbelleğine yapılır. L1 write-through politikası, bir L1 satırındaki herhangi bir değişikliği L2 önbelleğine zorlar ve bu nedenle diğer L2 önbellekleri tarafından görülebilir hale getirir. L1 write-through politikasının kullanılması, L1 içeriğinin L2 içeriğinin bir alt kümesi olmasını gerektirir. Bu da L2 önbelleğinin ilişkiselliğinin L1 ilişkiselliğine eşit ya da daha büyük olması gerektiğini gösterir. L1 write-through ilkesi IBM S/390 SMP'de kullanılır.

L1 önbelleğinin bir geri yazma politikası varsa, iki önbellek arasındaki ilişki daha karmaşıktır. Kapsamımız dışında kalan bir konu olan bakım için çeşitli yaklaşımalar vardır.

## 17.4 ÇOKLU İŞLEMCİLİK VE CİP ÇOKLU İŞLEMCİLER

Bir işlemci için en önemli performans ölçütü, komutları yürütme hızıdır. Bu şu şekilde ifade edilebilir

$$\text{MIPS oranı} = f^* \text{ IPC}$$

Burada  $f$  MHz cinsinden işlemci saat frekansıdır ve *IPC* (döngü başına talimat) döngü başına yürütülen ortalama talimat sayısıdır. Buna göre, tasarımcılar iki cephede performans artışı hedeflemiştirler: saat frekansını artırmak ve yürütülen komut sayısını ya da daha doğru bir ifadeyle bir işlemci döngüsü sırasında tamamlanan komut sayısını artırmak. Daha önceki bölgümlerde gördüğümüz gibi, tasarımcılar *IPC*'yi bir komut boru hattı kullanarak ve daha sonra bir süperskalar mimaride çoklu paralel komut boru hatları kullanarak artırmışlardır. Boru hatlı ve çoklu boru hatlı tasarımlarda temel sorun, her bir boru hattı aşamasının kullanımını en üst düzeye çıkarmaktır. Verimi artırmak için tasarımcılar, bazı talimatları talimat akışında meydana gelme şekillerinden farklı bir sırada yürütmemek ve hiçbir zaman ihtiyaç duyulmayacak talimatların yürütülmüşe başlamak gibi daha karmaşık mekanizmalar oluşturmuşlardır. Ancak Bölüm 2.2'de tartışıldığı gibi, bu yaklaşım karmaşıklık ve güç tüketimi endişeleri nedeniyle bir sınıra ulaşıyor olabilir.

Devre karmaşıklığını veya güç tüketimini artırmadan yüksek derecede komut düzeyinde paralellige izin veren alternatif bir yaklaşımı çoklu iş parçacığı denir. Temelde, komut akışı, iş parçacıkları olarak bilinen birkaç küçük akışa bölünür, böylece iş parçacıkları paralel olarak yürütülebilir.

Hem ticari sistemlerde hem de deneyel sistemlerde gerçekleştirilen özel çoklu iş parçacığı tasarımlarının çeşitliliği çok. Bu bölümde, başlıca kavramların kısa bir incelemesine vereceğiz.

### Örtük ve Açık Çoklu İş Parçacığı

Çok iş parçacıklı işlemcileri tartışırken kullanılan iş parçacığı kavramı, çok programlı bir işletim sistemindeki yazılım iş parçacığı kavramı ile aynı olabilir veya olmayıabilir. Terimleri kısaca tanımlamak faydalı olacaktır:

- **Süreç:** Bilgisayarda çalışan bir program örneği. Bir süreç iki temel özelliği bünyesinde barındırır:
  - **Kaynak sahipliği:** Bir süreç, süreç görüntüsünü tutmak için sanal bir adres alanı içerir; süreç görüntüsü, süreci tanımlayan program, veri, yığın ve öznitelikler koleksiyonudur. Zaman zaman bir sürece ana bellek, G/C kanalları, G/C aygıtları ve dosyalar gibi kaynakların kontrolü veya sahipliği tahsis edilebilir.
  - **Programlama/yürütme:** Bir sürecin yürütülmesi, bir veya daha fazla program aracılığıyla bir yürütme yolunu (iz) takip eder. Bu yürütme diğer süreçlerinkile iç içe geçmiş olabilir. Dolayısıyla, bir sürecin bir yürütme durumu (Çalışıyor, Hazır, vb.) ve bir gönderme önceliği vardır ve işletim sistemi tarafından programlanan ve gönderilen varlıklı.

- **İşlem değiştirme:** İşlemciyi bir işleminden diğerine geçiren, ilk işlem için tüm işlem kontrol verilerini, kayıtları ve diğer bilgileri kaydeden ve bunları ikincisi için işlem bilgileriyle değiştiren bir işlem<sup>2</sup>
- **İplik:** Bir süreç içinde gönderilebilir bir iş birimi. Bir işlemci bağlamı (program sayacı ve yoğun işaretçisini içerir) ve bir yoğun için kendi veri alanını (alt program dallanmasını etkinleştirmek için) içerir. Bir iş parçacığı sıralı olarak yürütülür ve işlemcinin başka bir iş parçacığına dönebilmesi için kesilebilir.
- **İş parçacığı değiştirme:** İşlemci kontrolünü aynı süreç içinde bir iş parçacığından diğerine geçirme eylemi. Tipik olarak, bu tür bir anahtarlama işlem anahtarlamasından çok daha az maliyetlidir.

Dolayısıyla, bir iş parçacığı zamanlama ve yürütme ile ilgiliyken, bir süreç hem zamanlama/yürütme hem de kaynak sahipliği ile ilgilidir. Bir süreç içindeki birden fazla iş parçacığı aynı kaynakları paylaşır. Bu nedenle bir iş parçacığı değişimi bir süreç değişimidenden çok daha az zaman alır. Unix'in önceki sürümleri gibi geleneksel işletim sistemleri iş parçacıklarını desteklemiyordu. Linux, unix'in diğer sürümleri ve Windows gibi modern işletim sistemlerinin çoğu iş parçacıklarını destekler. Uygulama programı tarafından görülebilen kullanıcı seviyesi iş parçacıkları ile yalnızca işletim sistemi tarafından görülebilen çekirdek seviyesi iş parçacıkları arasında bir ayırım yapılır. Bunların her ikisi de yazılımda tanımlanan açık iş parçacıkları olarak adlandırılabilir.

Şimdide kadar tüm ticari işlemciler ve deneysel işlemcilerin çoğu açık çoklu iş parçacığı kullanmıştır. Bu sistemler, farklı iş parçacıklarından gelen talimatları paylaşılan boru hatlarına serpiştirerek ya da paralel boru hatlarında paralel yürütme yoluyla farklı açık iş parçacıklarından gelen talimatları eşzamanlı olarak yürütür. Örtük çoklu iş parçacığı, tek bir sıralı programdan çıkarılan birden fazla iş parçacığının eş zamanlı yürütülmüşünü ifade eder. Bu örtük iş parçacıkları derleyici tarafından statik olarak ya da donanım tarafından dinamik olarak tanımlanabilir. Bu bölümün geri kalanında açık çoklu iş parçacığını ele alacağız.

### Açık Çoklu İş Parçacığı Yaklaşımları

En azından, çok iş parçacıklı bir işlemci, eş zamanlı olarak yürütülecek her bir yürütme iş parçacığı için ayrı bir program sayacı sağlamalıdır. Tasarımlar, eşzamanlı iş parçacığı yürütmemi desteklemek için kullanılan ek donanımın miktarı ve türü farklılık gösterir. Genel olarak, komut getirme işlemi iş parçacığıbazında gerçekleşir. İşlemci her bir iş parçacığını ayrı ayrı ele alır ve tek iş parçacıklı yürütmemi optimize etmek için dal tahmini, kayıt yeniden adlandırma ve süper skaler teknikler dahil olmak üzere bir dizi kullanabilir. Elde edilen şey, iş parçacığı düzeyinde paralelliktir ve komut düzeyinde paralellikle birleştirildiğinde büyük ölçüde gelişmiş performans sağlayabilir.

Genel olarak, çoklu iş parçacığına yönelik dört temel yaklaşım vardır:

- **Arahlı çoklu iş parçacığı:** Bu aynı zamanda **ince taneli çoklu iş parçacığı** olarak da bilinir. İşlemci aynı anda iki veya daha fazla iş parçacığı bağlamıyla ilgilenir ve her saat döngüsünde bir iş parçacığından diğerine geçiş yapar. Bir iş parçacığı şu nedenle engellenirse

---

<sup>2</sup> Bağlam anahtarı terimi genellikle işletim sistemi literatüründe ve ders kitaplarında bulunur. Ne yazık ki, literatürün çoğu bu terimi burada süreç anahtarı olarak adlandırılan şey anlamında kullanısa, diğer kaynaklar bunu iş parçacığı anahtarı anlamında kullanmaktadır. Belirsizliği önlemek için bu terim bu kitapta kullanılmamıştır.

## 630 BÖLÜM 17 / PARALEL İŞLEM

veri bağımlılıkları veya bellek gecikmeleri varsa, bu iş parçacığı atlanır ve hazır bir iş parçacığı yürütülür.

- **Bloklanmış çoklu iş parçacığı:** Bu aynı zamanda **kaba taneli çoklu iş parçacığı** olarak da bilinir. Bir iş parçacığının talimatları, önbellek ıskalama gibi gecikmeye neden olabilecek bir olay gerçekleşene kadar art arda yürütülür. Bu olay başka bir parçacığına geçişe neden olur. Bu yaklaşım, önbellek kaçırma gibi bir gecikme olayı için boru hattını durduracak sıralı bir işlemcide etkilidir.
- **Eşzamanlı çoklu iş parçacığı (SMT):** Talimatlar aynı anda birden fazla iş parçacığından superscalar işlemcinin yürütme birimlerine verilir. Bu, geniş süperskalar komut verme kapasitesini çoklu iş parçacığı bağlamlarının kullanımını ile birleştirir.
- **Cip çoklu işlem:** Bu durumda, birden fazla çekirdek tek bir cip üzerinde uygulanır ve her çekirdek ayrı iş parçacıklarını yönetir. Bu yaklaşımın avantajı, bir cip üzerindeki mevcut mantık alanının boru hattı tasarımda sürekli artan karmaşıklığa bağlı olmadan etkin bir şekilde kullanılmasıdır. Bu çoklu çekirdek olarak adlandırılır; bu konuya Bölüm 18'de ayrıca inceleyeceğiz.

İlk iki yaklaşım için, farklı iş parçacıklarından gelen talimatlar aynı anda uygulanmaz. Bunun yerine, işlemci farklı bir kayıt seti ve diğer bağlam bilgilerini kullanarak bir iş parçacığından diğerine hızla geçiş yapabilir. Bu, işlemcinin yürütme kaynaklarının daha iyi kullanılmasını sağlar ve önbellek ıskalamaları ve diğer gecikme olayları nedeniyle büyük bir cezayı önler. SMT yaklaşımı, çoğaltılmış yürütme kaynaklarını kullanarak farklı iş parçacıklarından gelen talimatların gerçek eşzamanlı yürütülmesini içerir. Yonga çoklu işlem ayrıca farklı iş parçacıklarından gelen talimatların eşzamanlı olarak yürütülmemesini sağlar.

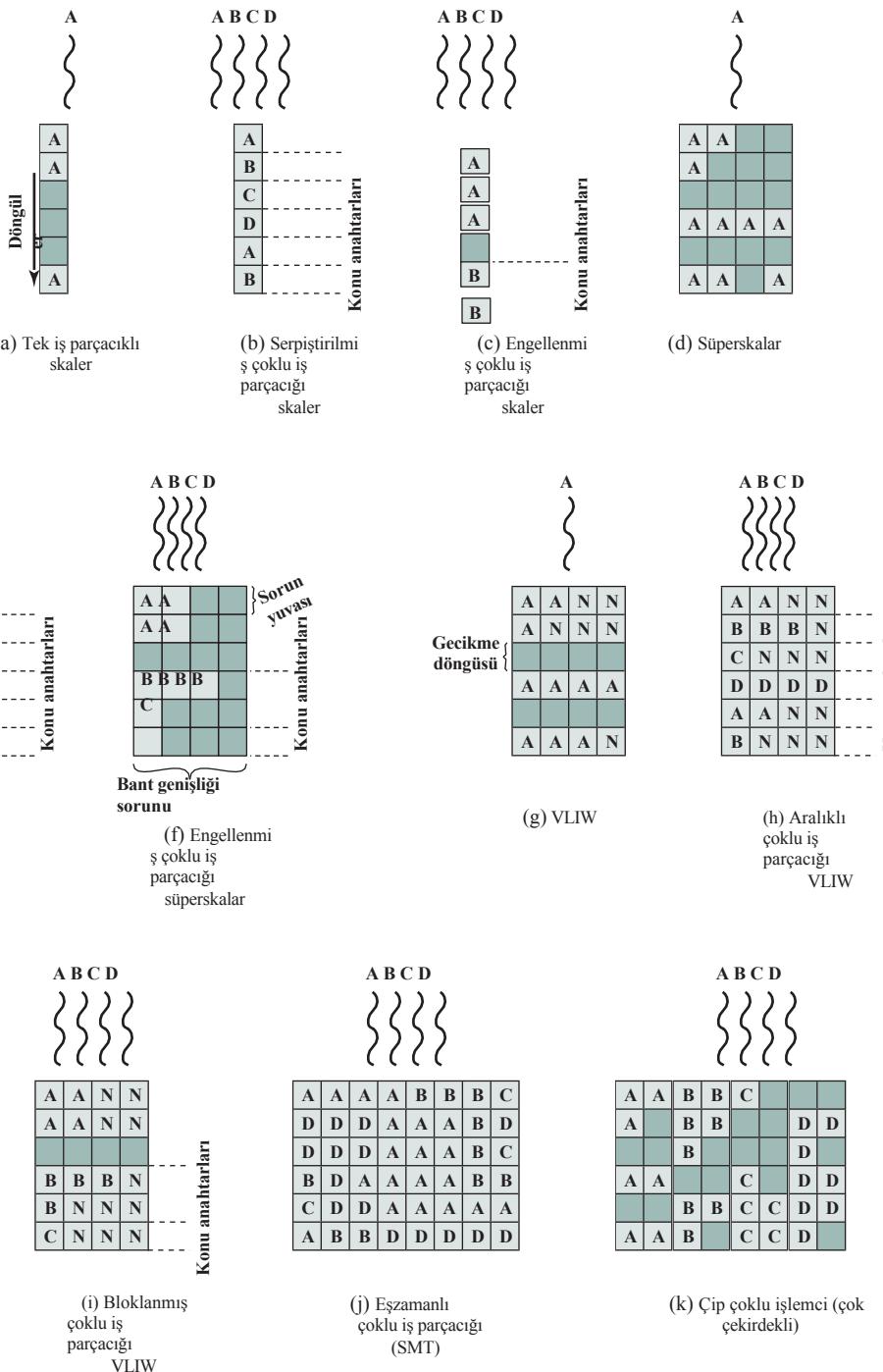
Şekil 17.7, [UNGE02]'deki bir örnek temel alınarak, çoklu iş parçacığı içeren bazı olası boru hattı mimarilerini göstermektedir. Bu, işlemcinin yürütme döngüsünü potansiyel sorun yuvasını veya yuvalarını temsil eder; her satırın genişliği, tek bir saat döngüsünde verilebilecek maksimum talimat sayısına karşılık gelir.<sup>3</sup> Dikey boyut, saat döngülerinin zaman sırasını temsil eder. Boş (gölgeli) bir yuva, bir boru hattında kullanılmayan bir yürütme yuvasını temsil eder. Bir no-op N ile gösterilir.

Şekil 17.7'deki ilk üç resim skaler (yani tek sorunlu) bir işlemci ile farklı yaklaşımları göstermektedir:

- **Tek iş parçacıklı skaler:** Bu, geleneksel RISC ve CISC makinelerinde bulunan ve çoklu iş parçacığı içermeyen basit boru hattıdır.
- **Araya serpiştirilmiş çok iş parçacıklı skaler:** Bu, uygulanması en kolay çoklu iş parçacığı yaklaşımıdır. Her saat döngüsünde bir iş parçacığından diğerine geçerek, boru hattı aşamaları tamamen dolu veya tamamen olmaya yakın tutulabilir. Donanım, döngüler arasında bir iş parçacığı bağlamından diğerine geçiş yapabilmelidir.

---

<sup>3</sup> Verme yuvaları, belirli bir saat döngüsünde talimatların verilebileceği konumdur. Bölüm 16'dan hatırlanacağı üzere, komut verme, işlemcinin işlevsel birimlerinde komut yürütmemeyi başlatma sürecidir. Bu, bir komut boru hattının kod çözme aşamasından boru hattının ilk yürütme aşamasına geçtiğinde gerçekleşir.



### **Şekil 17.7** Çoklu İş Parçacığı Yürütmeye Yaklaşımları

- **Bloklanmış çok iş parçacıklı skaler:** Bu durumda, boru hattını durduracak bir gecikme olayı meydana gelene kadar tek bir iş parçacığı yürütülür ve bu sırada işlemci başka bir iş parçacığına geçer.

Şekil 17.7c, iş parçacığı değiştirme süresinin bir döngü olduğu bir durumu gösterirken,

## **632 BÖLÜM 17 / PARALEL İŞLEM**

Şekil 17.7b iş parçacığı değiştirmenin sıfır döngüde gerçekleştiğini göstermektedir.

Aralıklı çoklu iş parçacığı durumunda, iş parçacıkları arasında kontrol veya veri bağımlılığı olmadığı varsayılar, bu da boru hattı tasarımını basitleştirir ve bu nedenle gecikme olmaksızın bir iş parçacığı geçişine izin vermelidir. Bununla birlikte, özel tasarım ve uygulamaya bağlı olarak, blok çoklu iş parçacığı, Şekil 17.7'de gösterildiği gibi, bir iş parçacığı geçişini gerçekleştirmek için bir saat döngüsü gerektirebilir. Bu durum, getirilen bir komutun iş parçacığı geçişini tetiklemesi ve boru hattından atılması gerektiğiinde geçerlidir [UNGE03].

Araya serpiştirilmiş çoklu iş parçacığı, bloklanmış çoklu iş parçacığına göre daha iyi işlemci kullanımını sunuyor gibi görünse de, bunu tek iş parçacığı performansından ödün vererek yapar. Çoklu iş parçacıkları önbellek kaynakları için rekabet eder, bu da belirli bir iş parçacığı için önbellek iskalama olasılığını artırır.

İşlemci döngü başına birden fazla komut verebiliyorsa, paralel yürütme için daha fazla fırsat mevcuttur. Şekil 17.7'den 17.7'iye kadar olan kısımlar, döngü başına dört yönerge yayımlamak için donanıma sahip işlemciler arasındaki bir dizi varyasyonu göstermektedir. Tüm bu durumlarda, tek bir döngüde yalnızca tek bir iş parçacığından talimatlar verilir. Aşağıdaki alternatifler gösterilmiştir:

- **Superscalar:** Bu, çoklu iş parçacığı içermeyen temel süperskalar yaklaşımıdır. Yakın zamana kadar bu, bir işlemci içinde paralellik sağlamaya yönelik en güçlü yaklaşımıdır. Bazı döngüler sırasında mevcut sorun yuvalarının tamamının kullanılmadığını unutmayın. Bu döngüler sırasında, maksimum talimat sayısından daha az talimat ; bu *yatay kayıp* olarak adlandırılır. Diğer talimat döngülerinde, hiçbir yayın yuvası kullanılmaz; bunlar hiçbir talimatın yayınlanmadığı döngülerdir; *dikey kayıp* olarak adlandırılır.
- **Serpiştirilmiş çoklu iş parçacığı superscalar:** Her döngü sırasında, tek bir iş parçacığından mümkün olduğunca çok sayıda talimat verilir. Bu teknikle, daha önce tartışıldığı gibi iş parçacığı geçişlerinden kaynaklanan potansiyel gecikmeler ortadan kaldırılır. Bununla birlikte, herhangi bir döngüde verilen talimatların sayısı hala herhangi bir iş parçacığı içinde var olan bağımlılıklarla sınırlıdır.
- **Bloklanmış çok iş parçacıklı süper skalar:** Yine, herhangi bir döngü sırasında yalnızca bir iş parçacığından talimatlar verilebilir ve bloklanmış çoklu iş parçacığı kullanılır.
- **Çok uzun komut sözcüğü (VLIW):** IA-64 gibi bir VLIW mimarisi, birden fazla talimiği tek bir kelimeye yerleştirir. Tipik bir VLIW, paralel olarak yürütülebilecek işlemleri aynı sözcüğe yerlestiren derleyici tarafından oluşturulur. Basit bir VLIW makinesinde (Şekil 17.7g), kelimeyi paralel olarak verilecek talimatlarla tamamen doldurmak mümkün değilse, no-ops kullanılır.
- **Interleaved multithreading VLIW:** Bu yaklaşım, superscalar mimaride interleaved multithreading tarafından sağlananlara benzer verimlilik sağlamalıdır.
- **Bloklanmış çoklu iş parçacıklı VLIW:** Bu yaklaşım, süper skalar mimaride bloklanmış çoklu iş parçacığı tarafından sağlananlara benzer verimlilik sağlamalıdır.

Şekil 17.7'de gösterilen son iki yaklaşım, birden fazla iş parçacığının paralel ve eşzamanlı olarak yürütülmesini sağlar:

- **Eşzamanlı çoklu iş parçacığı:** Şekil 17.7j bir seferde 8 komut verebilen bir sistemi göstermektedir. Bir iş parçacığı yüksek derecede komut düzeyinde paralelligę sahipse, bazı çevrimlerde yatay yuvaların tümünü doldurabilir. Diğer döngülerde, iki veya daha fazla iş parçacığından talimatlar verilebilir. Eğer yeterli



is parçacıkları aktif olduğunda, genellikle her döngüde maksimum sayıda talimat vermek mümkün olmalı ve yüksek düzeyde verimlilik sağlanmalıdır.

- **Çip çoklu işlemci (çok çekirdekli):** Şekil 17.7k, her biri iki sorulu süper skaler işlemciye sahip dört çekirdek içeren bir çipi göstermektedir. Her bir çekirdeğe, döngü başına en fazla iki talimat verebileceği bir iş parçası atanmıştır. Çok çekirdekli bilgisayarları Bölüm 18'de tartışıyoruz.

Şekil 17.7j ve 17.7k karşılaştırıldığında, SMT ile aynı komut verme kapasitesine sahip bir çip çoklu işlemcinin aynı derecede komut düzeyinde paralelliğe ulaşamadığını görüyoruz. Bunun nedeni, yonga çoklu işlemcinin diğer iş parçacıklarından talimatlar yayınlayarak gecikmeleri gizleyememesidir. Öte yandan, çip çoklu işlemci aynı komut kapasitesine sahip bir süper skaler işlemciden daha iyi performans göstermelidir, çünkü yatay kayıplar süper skaler işlemci için daha büyük olacaktır. Buna ek olarak, bir çip çoklu işlemcideki çekirdeklerin her birinde çoklu iş parçası kullanmak mümkündür ve bu bazı çağdaş makinelerde yapılmaktadır.

## 17.5 KÜMELER

Önemli ve nispeten yeni bir gelişme olan bilgisayar sistemi tasarımları kümelenmedir. Kümeleme, yüksek performans ve yüksek kullanılabilirlik sağlamaya yönelik bir yaklaşım olarak simetrik çoklu işleme bir alternatifdir ve özellikle sunucu uygulamaları için caziptir. Bir kümeyi, tek bir makine olduğu yanılsamasını yaratabilecek birlesik bir bilgi işlem kaynağı olarak birlikte çalışan, birbirine bağlı, bütün bilgisayarlar grubu olarak tanımlayabiliriz. *Bütün bilgisayar* terimi, kümelenen ayrı olarak kendi başına çalışabilen bir sistem anlamına gelir; literatürde, bir kümelenen her bir bilgisayar genellikle *düğüm* olarak adlandırılır. [BREW97] kümelenme ile elde edilebilecek dört faydayı listelemektedir.

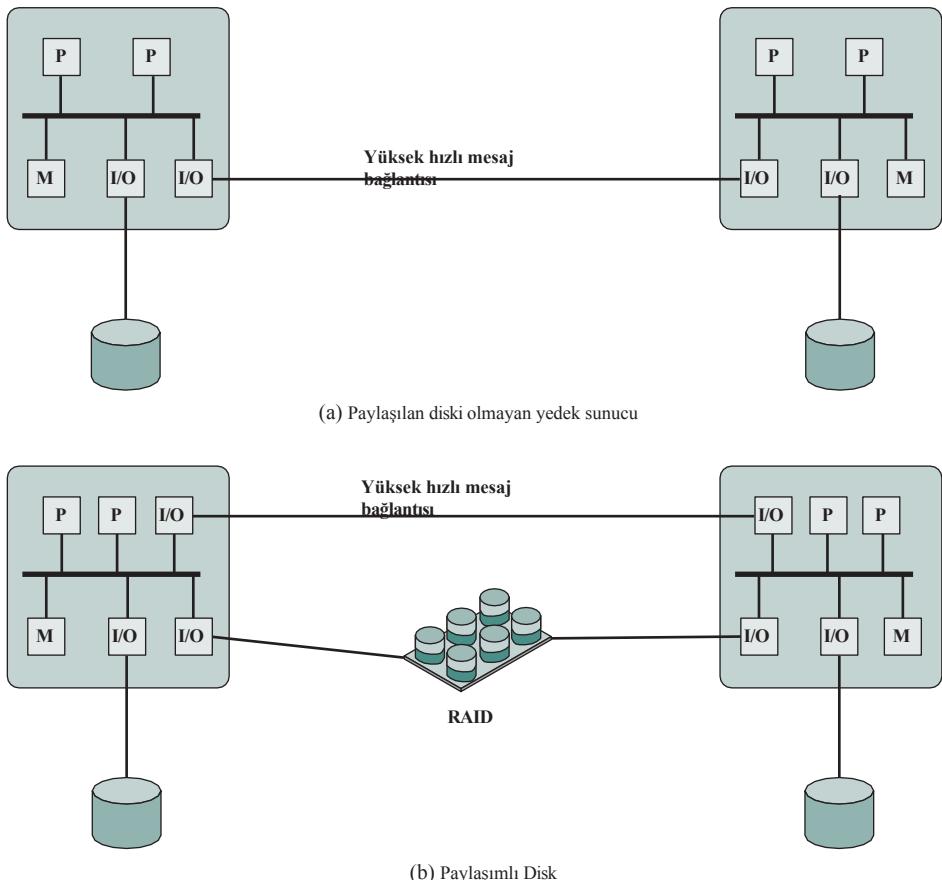
Bunlar

hedefler veya tasarım gereksinimleri olarak da düşünülebilir:

- **Mutlak ölçeklenebilirlik:** En büyük bağımsız makinelerin bile gücünü büyük kümeler oluşturmak mümkündür. Bir kümelenen her biri çoklu işlemci olan onlarca, yüzlerce, hatta binlerce makine olabilir.
- **Artımlı ölçeklenebilirlik:** Bir kümelenen her biri küçük artışlarla yeni sistemler eklenebilecek şekilde yapılandırılır. Böylece, bir kullanıcı mütevazı bir sistemle başlayabilir ve mevcut küçük bir sistemin daha büyük bir sistemle değiştirildiği büyük bir yükseltme yapmak zorunda kalmadan ihtiyaçlar arttıkça sistemi genişletebilir.
- **Yüksek kullanılabilirlik:** Bir kümelenen her bir düğüm bağımsız bir bilgisayar olduğundan, bir düğümün arızalanması hizmet kaybı anlamına gelmez. Birçok türünde, hata toleransı yazılımda otomatik olarak ele alınır.
- **Üstün fiyat/performans:** Emtia yapı taşlarını kullanarak, tek bir büyük makineye eşit veya daha fazla bilgi işlem gücüne sahip bir kümeyi çok daha düşük maliyetle bir araya getirmek mümkündür.

### Küme Yapılandırmaları

Literatürde kümeler bir dizi farklı şekilde sınıflandırılmaktadır. Belki de en basit sınıflandırma, bir kümelenen bilgisayarların aynı disklere erişimi paylaşıp paylaşmadığına dayanmaktadır. Şekil 17.8a'da tek ara bağlantının bilgisayarlar arasında olduğu iki düğümlü bir kümelenen gösterilmektedir.

**Şekil 17.8** Küme Yapılandırmaları

küme faaliyetlerini koordine etmek amacıyla mesaj alışverişi için kullanılabilen yüksek hızlı bir bağlantı aracıyla gerçekleşir. Bağlantı, kümenin parçası olmayan diğer bilgisayarlarla paylaşılabilir bir LAN olabilir ya da bağlantı özel bir ara bağlantı tesisi olabilir. İkinci durumda, kümeye bir veya daha fazla bilgisayarın bir LAN veya WAN bağlantısı olacaktır, böylece sunucu kümesi ile uzak istemci sistemleri arasında bir bağlantı olacaktır. Şekilde her bilgisayarın çok işlemcili olarak gösterildiğine dikkat edin. Bu gerekli değildir ancak hem performansı hem de kullanılabilirliği artırır.

Şekil 17.8'de gösterilen basit sınıflandırmada, diğer alternatif bir paylaşımı disk kümeleridir. Bu durumda, genellikle düğümler arasında hala bir mesaj bağlantısı vardır. Buna ek olarak, küme içindeki birden fazla bilgisayara doğrudan bağlı olan bir disk alt sistemi vardır. Bu şekilde, ortak disk alt sistemi bir RAID . RAID ya da benzer bir yedek disk teknolojisinin kullanımı kümelerde yaygındır, böylece birden fazla bilgisayarın varlığıyla elde edilen yüksek kullanılabilirlik, tek bir arıza noktası olan paylaşım bir disk tarafından tehlikeye atılmaz.

İşlevsel alternatiflere bakarak küme seçenekleri yelpazesine ilişkin daha net bir resim elde edilebilir. Tablo 17.2, şimdi tartışacağımız işlevsel hatlar boyunca faydalı bir sınıflandırma sunmaktadır.

**Tablo 17.2** Kümeleme Yöntemleri: Faydalar ve Sınırlamalar

Kümeleme Yöntemi	Açıklama	Avantajlar	Sınırlamalar
<b>Pasif Bekleme</b>	Birincil sunucunun arızalanması durumunda ikincil sunucu devreye girer.	Uygulaması kolay.	İkincil sunucu diğer işlem görevleri için kullanılmadığından yüksek maliyet.
<b>Aktif ikincil:</b>	İkincil sunucu da görevleri işlemek için kullanılır.	İşlem için ikincil sunucular kullanılabilirliğinden daha düşük maliyet.	Artan karmaşıklık.
Ayrı Sunucular	Ayrı sunucuların kendi diskleri vardır. Veriler sürekli olarak kopyalanır birincil sunucudan ikinci sunucuya.	Yüksek kullanılabilirlik.	Kopyalama işlemleri nedeniyle yüksek ağ ve sunucu ek yükü.
Disklere Bağlı Sunucular	Sunucular aynı disklere bağlanır, ancak her sunucu kendi disklerine sahiptir. Bir sunucu arızalanırsa, diskleri diğer tarafından devralınır.	Kopyalama işlemlerinin ortadan kaldırılması ağ ve sunucu ek yükünde azalma.	Disk arızası riskini telafi etmek için genellikle disk yansıtma veya RAID teknolojisi gereklidir.
Sunucular Diskleri Paylaşır	Birden fazla sunucu aynı anda disklere erişimi paylaşır.	Düşük ağ ve sunucu ek yükü. Disk arızasından kaynaklanan kesinti riskinde azalma.	Kilit yönetici yazılımı gerektirir. Genellikle disk yansıtma veya RAID teknolojisi ile kullanılır.

**Pasif bekleme** olarak bilinen yaygın ve eski bir yöntem, bir bilgisayarın tüm işlem yükünü üstlenirken diğer bilgisayarın etkin olmaması ve birincil bilgisayarın arızalanması durumunda görevi devralmak üzere hazır beklemesi şeklindeki bir yöntemdir. Makineleri koordine etmek için, aktif ya da birincil sistem yedek makineye periyodik olarak bir "kalp atışı" mesajı gönderir. Bu mesajların gelmesi durursa, yedek makine birincil sunucunun arızalandığını varsayar ve kendisini devreye sokar. Bu yaklaşım kullanılabilirliği artırır ancak performansı iyileştirmez. Ayrıca, iki sistem arasında değişim tokus edilen tek bilgi kalp atışı mesajıyla iki sistem ortak diskleri paylaşmıyorsa, yedek makine işlevsel bir yedekleme sağlar ancak birincil tarafından yönetilen veritabanlarına erişimi yoktur.

Pasif yedek genellikle küme olarak adlandırılmaz. *Küme* terimi, dış dünyaya karşı tek bir sistem görüntüsünü korurken aktif olarak işlem yapan birbirine bağlı birden fazla bilgisayar için ayrılmıştır. **Aktif ikincil** terimi genellikle bu yapılandırmaya atıfta bulunmak için kullanılır. Kümelemenin üç sınıflandırması tanımlanabilir: ayrı sunucular, paylaşılan hiçbir şey ve paylaşılan bellek.

Kümelemeye yönelik bir yaklaşımın, her bilgisayar kendi disklerine sahip **ayrı bir sunucudur** ve sistemler arasında paylaşılan disk yoktur (Şekil 17.8a). Bu düzenleme yüksek performansın yanı sıra yüksek kullanılabilirlik de sağlar. Bu durumda, gelen istemci isteklerini sunuculara atamak için bir tür yönetim veya zamanlama yazılımına ihtiyaç duyulur, böylece yük dengelenir ve yüksek kullanım elde edilir. Yük devretme özelliğine sahip olması arzu edilir, yani bir bilgisayar bir uygulamayı çalıştırırken arızalanırsa, kümedeki başka bir bilgisayar devreye girip uygulamayı tamamlayabilir.

uygulama. Bunun gerçekleşmesi için verilerin arasında sürekli olarak kopyalanması gereklidir, böylece her sistem diğer sistemlerin güncel verilerine erişebilir. Bu veri alışverişi, performans kaybı pahasına yüksek kullanılabilirlik sağlar.

İletişim yükünü azaltmak için, çoğu küme artık ortak disklere bağlı sunuculardan olusmaktadır (Şekil 17.8b). **Paylaşılan hiçbir şey** olarak adlandırılan bu yaklaşımın bir varyasyonunda, ortak diskler birimlere ayrılr ve her bir birim tek bir bilgisayara aittir. Bu bilgisayar arızalanırsa, küme, arızalanan bilgisayarın birimlerine başka bir bilgisayarın sahip olacağı şekilde yeniden yapılandırılmalıdır.

Birden fazla bilgisayarın aynı anda aynı diskleri paylaşması da mümkündür (**paylaşılan** disk yaklaşımı olarak adlandırılır), böylece her bilgisayarn tüm disklerdeki tüm birimlere erişimi olur. Bu yaklaşım, verilere aynı anda yalnızca bir bilgisayar tarafından erişilebilmesini sağlamak için bir tür kilitleme olanağı kullanılmasını gerektirir.

## **İşletim Sistemi Tasarım Sorunları**

Bir küme donanım yapılandırmasından tam olarak faydalananm için tek sistemli bir işletim sisteminde bazı geliştirmeler yapılması gereklidir.

**ARIZA YÖNETİMİ** Arızaların bir küme tarafından nasıl yönetileceği kullanılan kümelerle yöntemine bağlıdır (Tablo 17.2). Genel olarak, arızalarla başa çıkmak için iki yaklaşım benimsenebilir: yüksek oranda kullanılabilir kümeler ve hataya dayanıklı kümeler. Yüksek oranda kullanılabilir bir küme, tüm kaynakların hizmette olma olasılığının yüksek olmasını sağlar. Bir sistemin çökmesi veya bir disk biriminin kaybolması gibi bir arıza meydana gelirse, devam eden sorgular kaybolur. Kaybedilen herhangi bir sorğu, yeniden denenirse, kümelerde farklı bir bilgisayar tarafından hizmet verilecektir. Ancak, küme işletim sistemi kısmen yürütülen işlemlerin durumu hakkında hiçbir garanti vermez. Bunun uygulama düzeyinde ele alınması gereklidir.

Hata toleranslı bir küme, tüm kaynakların her zaman kullanılabilir olmasını sağlar. Bu, yedekli paylaşılan disklerin ve işlenmemiş işlemlerin yedeklenmesi ve tamamlanan işlemlerin işlenmesi için mekanizmaların kullanılmasıyla elde edilir.

Uygulamaları ve veri kaynaklarını arızalı bir sistemden kümelerdeki alternatif bir sisteme geçirme islevi **yük devretme** olarak adlandırılır. İlgili bir işlev de, sorun giderildikten sonra uygulamaların ve veri kaynaklarının orijinal sisteme geri yüklenmesidir; bu da **fallback** olarak adlandırılır. Yük devretme otomatik hale getirilebilir, ancak bu yalnızca sorun gerçekten giderilmişse ve tekrarlanma olasılığı düşükse arzu edilir. Aksi takdirde, otomatik geri dönüş daha sonra arızalanan kaynakların bilgisayarlar arasında gidip gelmesine neden olarak performans ve kurtarma sorunlarına yol açabilir.

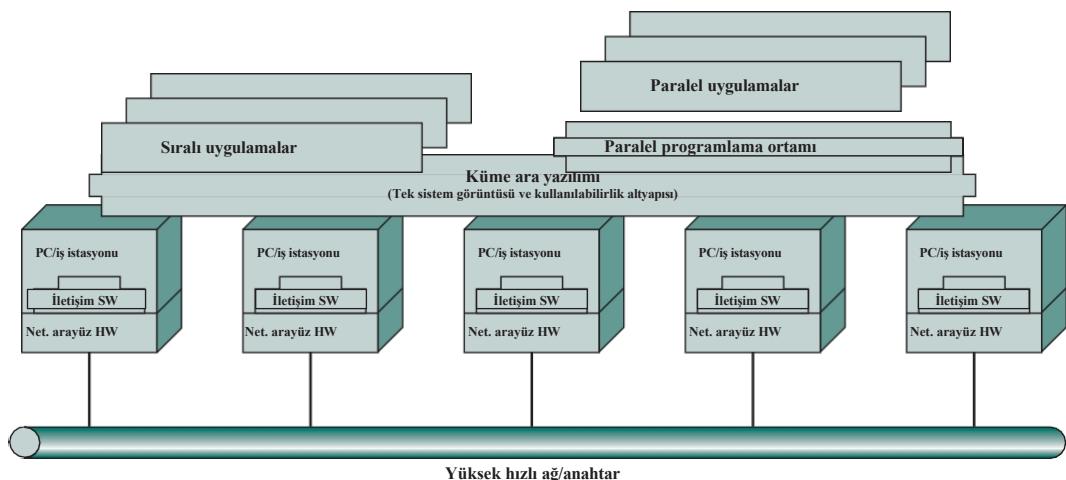
**YÜK DENGELİME** Bir küme, mevcut bilgisayarlar arasında yükü dengellemek için etkili bir kabiliyet gerektirir. Bu, kümeyi kademeli olarak ölçeklenebilir olması gerekliliğini de içerir. Kümeye yeni bir bilgisayar eklendiğinde, yük dengeleme olanağı bu bilgisayarı otomatik olarak zamanlama uygulamalarına dahil etmelidir. Ara katman mekanizmalarının, hizmetlerin kümeyi farklı üyelerinde görünebileceğini ve bir üyeden diğerine geçebileceğini kabul etmesi gereklidir.

**PARALELLEŞTİRME** Bazı durumlarda, bir kümeyi etkin kullanımı tek bir uygulamadaki yazılımın paralel olarak yürütülmesini gerektirir. [KAPP00] bu soruna yönelik üç genel yaklaşım listelemektedir:

- **Paralelleştirici derleyici:** Paralelleştirici bir derleyici, derleme zamanında bir uygulamanın hangi bölümlerinin paralel olarak yürütülebileceğini belirler. Bunlar daha sonra kümeye farklı bilgisayarlarla atanmak üzere ayrılır. Performans, problemin doğasına ve derleyicinin ne kadar iyi tasarılandığına bağlıdır. Genel olarak, bu tür derleyicilerin geliştirilmesi zordur.
- **Paralelleştirilmiş uygulama:** Bu yaklaşımda, programcı uygulamayı en baştan bir küme üzerinde çalışacak şekilde yazar ve gerektiğinde verileri küme düğümleri arasında taşımak için mesaj geçisi kullanır. Bu, programcıya yüksek bir yük getirir ancak bazı uygulamalarda kümelerden yararlanmak için en iyi yaklaşım olabilir.
- **Parametrik hesaplama:** Bu yaklaşım, uygulamanın özü, her seferinde farklı bir başlangıç koşulları veya parametreleri kümeye çok sayıda kez yürütülmesi gereken bir algoritma veya program ise kullanılabilir. Çok sayıda farklı senaryoyu çalıştıracak ve ardından sonuçların istatistiksel özetlerini geliştirecek bir simülasyon modeli buna iyi bir örnektir. Bu yaklaşımın etkili olabilmesi için, işleri etkili bir şekilde organize etmek, çalıştmak ve yönetmek için parametrik işleme araçlarına ihtiyaç vardır.

## Küme Bilgisayar Mimarisi

Sekil 17.9 tipik bir küme mimarisini göstermektedir. Tek tek bilgisayarlar bazı yüksek hızlı LAN ya da anahtar donanımlarıyla birbirine bağlanır. Her bilgisayar bağımsız olarak çalışabilir. Buna ek olarak, kümeyi çalışmasını sağlamak için her bilgisayara bir ara katman yazılımı yüklenir. Küme ara yazılımı, kullanıcıya tek sistem görüntüsü olarak bilinen birleşik bir sistem görüntüsü sağlar. Ara yazılım aynı zamanda yük dengeleme ve yük dengeleme yoluyla yüksek kullanılabilirlik sağlamaktan da sorumludur.



**Şekil 17.9** Küme Bilgisayar Mimarisi [BUYY99]

## 638 BÖLÜM 17 / PARALEL İŞLEM

Bireysel bileşenlerdeki arızalara yanıt verme. [HWAN99] aşağıdakileri arzu edilen küme arakatman hizmetleri ve işlevleri olarak listelemektedir:

- **Tek giriş noktası:** Bir kullanıcı tek bir bilgisayar yerine kümede oturum açar.
- **Tek dosya hiyerarşisi:** Kullanıcı aynı kök dizin altında tek bir dosya dizini hiyerarşisi görür.
- **Tek kontrol noktası:** Küme yönetimi ve kontrolü için kullanılan varsayılan bir iş istasyonu vardır.
- **Tek sanal ağ:** Gerçek küme yapılandırması birden fazla birbirine bağlı ağdan oluşsa bile, herhangi bir düğüm kümedeki diğer noktalara erişebilir. Tek bir sanal ağ işlemi vardır.
- **Tek bellek alanı:** Dağıtılmış paylaşılan bellek, programların değişkenleri paylaşmasını sağlar.
- **Tek iş yönetim sistemi:** Bir küme iş zamanlayıcısı altında, bir kullanıcı işi yürütecek ana bilgisayarı belirtmeden bir işi .
- **Tek kullanıcı arayüzü:** Ortak bir grafik arayüzü, kümeye hangi iş istasyonundan girdiklerine bakılmaksızın tüm kullanıcıları destekler.
- **Tek I/O alanı:** Herhangi bir düğüm, fiziksel konumu hakkında bilgi sahibi olmadan herhangi bir G/Ç çevre birimine veya disk aygitine uzaktan erişebilir.
- **Tek işlem alanı:** Tek tip bir süreç tanımlama şeması kullanılır. Herhangi bir düğümdeki bir süreç, uzak bir düğümde başka bir süreç oluşturabilir veya bu süreçle iletişim kurabilir.
- **Kontrol noktası oluşturma:** Bu işlev, bir hatadan sonra geri dönüş kurtarmaya izin vermek için işlem durumunu ve aracı hesaplama sonuçlarını periyodik olarak kaydeder.
- **Süreç geçisi:** Bu işlev yük dengelemeyi mümkün kılar.

Yukarıdaki listede yer alan son dört madde kümenin kullanılabilirliğini artırmaktadır.

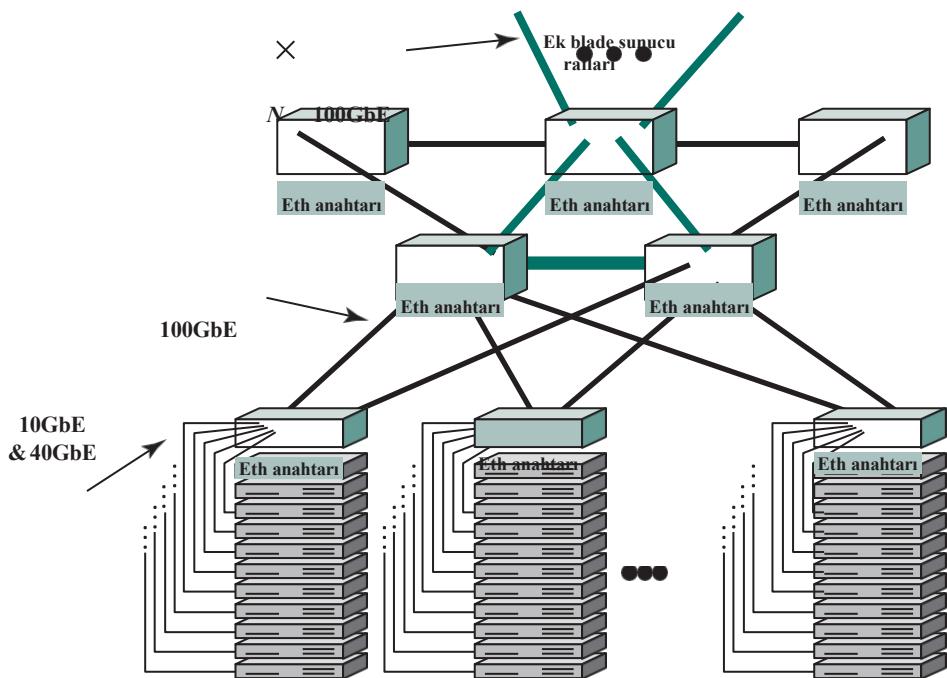
Kalan öğeler tek bir sistem görüntüsü sağlamakla ilgilidir.

Şekil 17.9'a dönecek olursak, bir küme paralel yürütme yeteneğine sahip programların verimli bir şekilde yürütülmesini sağlayan yazılım araçlarını da içerecektir.

### Blade Sunucular

Küme yaklaşımının yaygın bir uygulaması blade sunucudur. Blade sunucu, tek bir kasada birden fazla sunucu modülü ("blade") barındıran bir sunucu mimarisidir. Yerden tasarruf etmek ve sistem yönetimini iyileştirmek için veri merkezlerinde yaygın olarak kullanılır. Kendi başına duran ya da rafa monte edilen kasa güç kaynağını sağlar ve her blade'in kendi işlemcisi, belleği ve sabit diski vardır.

Uygulamanın bir örneği Şekil 17.10'da gösterilmektedir. Büyük veri merkezlerindeki eğilim, önemli sayıda blade sunucusunun bulunduğu yerlerde, bu sunucular tarafından sağlanan devasa multimedya trafiğini işlemek için tek tek sunuculara 10 Gbps bağlantı noktalarının yerleştirilmesidir. Bu tür düzenlemeler, çok sayıda sunucuya birbirine bağlamak için gereken tesis içi Ethernet anahtarlarını zorlamaktadır. 100 Gbps hız, artan trafik yükünü idare etmek için gereken bant genişliğini sağlar. 100 Gbps



**Şekil 17.10** Büyük Blade Sunucu Sitesi için Örnek 100-Gbps Ethernet Yapılandırması

Ethernet anahtarları, kurumsal ağlar için binalar arası, kampüsler arası, geniş alan bağlantıları sağlamanın yanı sıra veri merkezi içindeki anahtar yukarı bağlantılarında da kullanılır.

### SMP ile Karşılaştırıldığında Kümeler

Hem kümeler hem de simetrik çoklu işlemciler, yüksek talep gören uygulamaları desteklemek için birden fazla işlemciye sahip bir yapılandırma sağlar. SMP şemaları çok daha uzun süredir kullanılıyor olsa da her iki çözüm de ticari olarak mevcuttur.

SMP yaklaşımının temel gücü, bir SMP'nin yönetilmesinin ve yapılandırılmasının bir kümeden daha kolay olmasıdır. SMP, neredeyse tüm uygulamaların yazıldığı orijinal tek işlemci modelde çok daha yakındır. Tek işlemciden SMP'ye geçişte gerekli olan temel değişiklik zamanlayıcı işlevidir. SMP'nin bir diğer faydası da genellikle daha az fiziksel alan kaplaması ve benzer bir kümeye göre daha az güç çekmesidir. Son bir önemli fayda da SMP ürünlerinin iyi kurulmuş ve istikrarlı olmasıdır.

Ancak uzun vadede, kümelerin yaklaşımının avantajları, kümelerin yüksek performanslı sunucu pazarına hakim olmasına sonuclanacaktır. Kümeler, artımlı ve mutlak ölçeklenebilirlik açısından SMP'lerden çok daha üstündür. Kümeler kullanılabilirlik açısından da üstündür, çünkü sistemin tüm bileşenleri kolaylıkla yüksek oranda yedekli hale getirebilir.

## 17.6 TEKDÜZE OLMAYAN BELLEK ERIŞİMİ

Ticari ürünler açısından, uygulamaları desteklemek için çok işlemcili bir sistem sağlamaya yönelik iki yaygın yaklaşım SMP'ler ve kümelerdir. Birkaç yıldır, tek tip olmayan bellek erişimi (NUMA) olarak bilinen başka bir yaklaşım araştırma konusu olmuştur ve ticari NUMA ürünleri artık mevcuttur.

Devam etmeden önce, NUMA literatüründe sıkça rastlanan bazı terimleri tanımlamamız gereklidir.

- **Tek tip bellek erişimi (UMA):** Tüm işlemciler yükleme ve işlemlerini kullanarak ana belleğin tüm bölmelerine erişebilir. Bir işlemcinin belleğin tüm bölgelerine erişim süresi aynıdır. Farklı işlemciler tarafından deneyimlenen erişim süreleri aynıdır. Bölüm 17.2'de tartışılan SMP organizasyonu ve 17.3 UMA'dır.
- **Tekdüze olmayan bellek erişimi (NUMA):** Tüm işlemciler yükleme ve saklama işlemlerini kullanarak ana belleğin tüm bölmelerine erişebilir. Bir işlemcinin bellek erişim süresi, ana belleğin hangi bölgesine erişildiğine bağlı olarak farklılık gösterir. Son ifade tüm işlemciler için doğrudur; ancak farklı işlemciler için hangi bellek bölgelerinin daha yavaş ve hangilerinin daha hızlı olduğu farklılık gösterir.
- **Önbellek tutarlı NUMA (CC-NUMA):** Çeşitli işlemcilerin önbellekleri arasında önbellek tutarlılığının korunduğu bir NUMA sistemi.

Önbellek tutarlılığı olmayan bir NUMA sistemi aşağı yukarı bir cluster'a eşdeğerdir. Son zamanlarda en çok ilgi gören ticari ürünler, hem SMP'lerden hem de kümelerden oldukça farklı olan CC-NUMA sistemleridir. Genellikle, ancak ne yazık ki her zaman değil, bu tür sistemler aslında ticari literatürde CC-NUMA sistemleri olarak adlandırılır. Bu bölüm sadece CC-NUMA sistemleri ile ilgilidir.

### Motivasyon

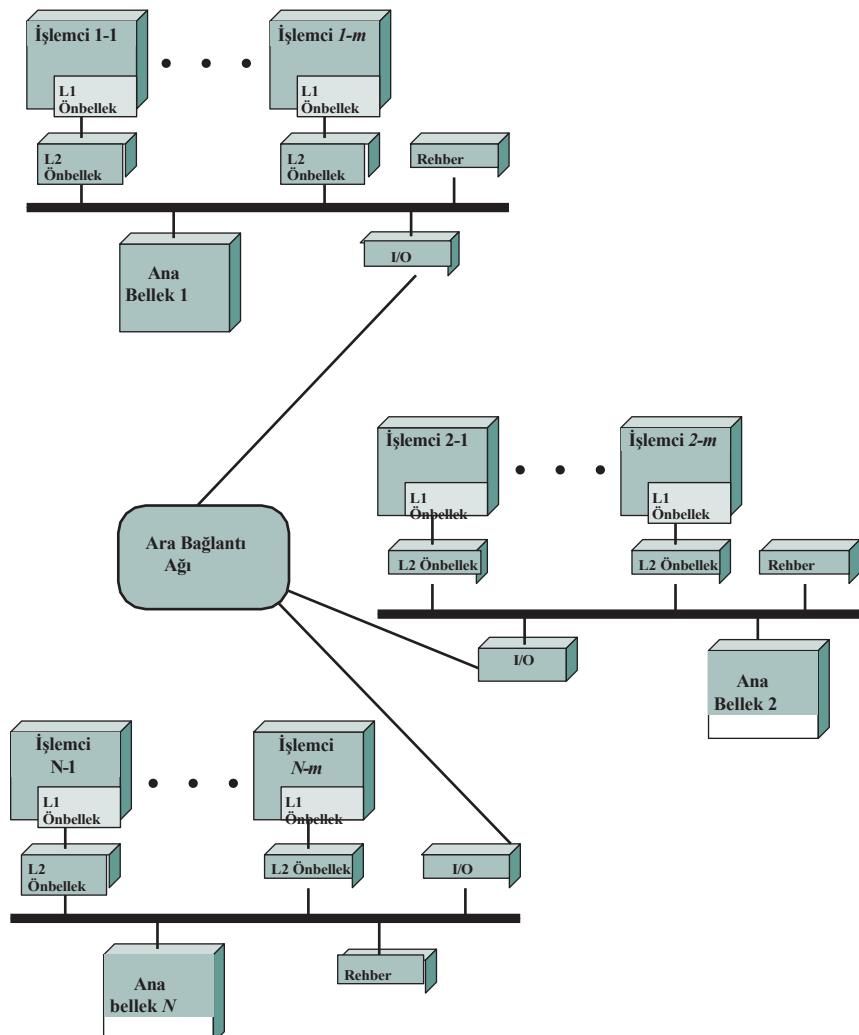
Bir SMP sisteminde, kullanılabilecek işlemci sayısının pratik bir sınırı vardır. Etkili bir önbellek şeması, herhangi bir işlemci ile ana bellek arasındaki veri yolu trafigini azaltır. İşlemci sayısı arttıkça, bu veri yolu trafigi de artar. Ayrıca, veri yolu önbellek uyum sinyallerini değişim tokus etmek için kullanılır ve bu da yükü daha da artırır. Bir noktada veri yolu bir performans darboğazı haline gelir. Performans düşüşü, SMP konfigürasyonundaki işlemci sayısını 16 ila 64 işlemci arasında bir yerde sınırlı gibi görülmektedir. Örneğin, Silicon Graphics'in Power Challenge SMP'si tek bir sisteme 64 R10000 işlemciyle sınırlıdır; bu sayının ötesinde performans önemli ölçüde düşer.

Bir SMP'deki işlemci sınırı, küme sistemlerinin geliştirilmesinin arkasındaki itici motivasyonlardan biridir. Ancak bir küme ile her düğümün kendi özel ana belleği vardır; uygulamalar büyük bir global bellek görmez. Aslında, tutarlılık donanımından ziyade yazılımda korunur. Bu bellek granülerliği performansı etkiler ve maksimum performans elde etmek için yazılımın bu ortama uygun hale getirilmesi gereklidir. SMP'nin lezzetini korurken büyük ölçekli çoklu işleme ulaşmaya yönelik bir yaklaşım NUMA'dır.

NUMA ile amaç, her biri kendi veri yoluna veya diğer dahili ara bağlantı sistemine sahip çoklu işlemci düğümlerine izin verirken şeffaf bir sistem geniş belleğini korumaktır.

## Organizasyon

Şekil 17.11 tipik bir CC-NUMA organizasyonunu göstermektedir. Her biri gerçekte bir SMP organizasyonu olan birden fazla bağımsız düğüm vardır. Böylece her düğüm, her biri kendi L1 ve L2 önbelleklere ve ana belleğe sahip birden fazla işlemci içerir. Düğüm, genel CC-NUMA organizasyonunun temel yapı taşıdır. Örneğin, her Silicon Graphics Origin düğümü iki MIPS R10000 işlemci içerir;



Şekil 17.11 CC-NUMA Organizasyonu

## 642 BÖLÜM 17 / PARALEL İŞLEM

Her Sequent NUMA-Q düğümü dört Pentium II işlemci içerir. Düğümler, bir anahtarlarla mekanizması, halka veya başka bir ağ tesisini olabilen bazı iletişim tesisleri aracılığıyla birbirine bağlanır.

CC-NUMA sistemindeki her düğüm bir miktar ana bellek içerir. Ancak işlemciler açısından bakıldığında, her bir konumun sistem genelinde benzersiz bir adrese sahip olduğu tek bir adreslenebilir bellek vardır. Bir işlemci bir bellek erişimi başlattığında, istenen bellek konumu o işlemcinin önbellegeinde değilse, L2 önbellegi bir getirme işlemi başlatır. İstenen satır ana belleğin yerel bölümündeyse, satır yerel veri yolu üzerinden getirilir. İstenen satır ana belleğin uzak bir bölümündeyse, bu satırı ara bağlantı ağı üzerinden almak, yerel veri yoluna teslim etmek ve ardından bu veri yolu üzerindeki istekte bulunan önbellege teslim etmek için otomatik bir istek gönderilir. Tüm bu faaliyetler otomatiktir ve işlemci ile önbellegi için şeffaftır.

Bu yapılandırmada, önbellek tutarlılığı temel bir husustur. Uygulamalar ayrıntılara göre farklılık gösterse de, genel anlamda her düğümün belleğin çeşitli bölümlerinin konumunu ve ayrıca önbellek durum bilgisini gösteren bir tür dizin tutması gerektiğini söyleyebiliriz. Bu şemanın nasıl çalıştığını görmek için [PFIS98]’den alınan bir örnek vereceğiz. Düğüm 2 (P2-3) üzerindeki işlemci 3’ün düğüm 1’in belleğinde bulunan 798 numaralı bellek konumunu talep ettiğini varsayalım. Aşağıdaki sıra gerçekleşir:

1. P2-3, düğüm 2’nin snoopy veriyolunda 798 konumu için bir okuma isteği yayınlar.
2. Düğüm 2’deki dizin isteği görür ve konumun düğüm 1’de olduğunu anlar.
3. Düğüm 2’nin dizini düğüm 1’e bir istek gönderir ve bu istek düğüm 1’in dizini tarafından alınır.
4. P2-3’ün vekili olarak hareket eden Node 1’in dizini, sanki bir işlemciymiş gibi 798’in içeriğini talep eder.
5. Düğüm 1’in ana belleği istenen verileri veri yoluna koyarak yanıt verir.
6. Düğüm 1’in dizini verileri alır.
7. Değer düğüm 2’nin dizinine geri aktarılır.
8. Düğüm 2’nin dizini, verileri düğüm 2’nin veri yoluna geri yerleştirir ve başlangıçta onu tutan bellek için bir vekil görevi görür.
9. Değer almır ve P2-3’ün önbellegine yerleştirilir ve P2-3’e teslim edilir.

Önceki sekans, verilerin uzak bir bellekten, işlemi işlemciye şeffaf hale getiren donanım mekanizmaları kullanılarak nasıl okunduğunu açıklamaktadır. Bu mekanizmaların yanı sıra, bir çeşit önbellek tutarlılık protokolüne ihtiyaç vardır. Çeşitli sistemler bunun tam olarak nasıl yapıldığı konusunda farklılık gösterir. Biz burada sadece birkaç genel açıklama yapacağız. İlk olarak, önceki dizinin bir parçası olarak, düğüm 1’in dizini, uzak bir önbellegin 798 numaralı konumu içeren satırın bir kopyasına sahip olduğuna dair bir kayıt tutar. Daha sonra, değişikliklerle ilgilenmek için işbirlikçi bir protokol olması gereklidir. Örneğin, bir önbellette bir değişiklik yapılarsa, bu gerçek diğer düğümlere yayınlanabilir. Böyle bir yayımı alan her düğümün dizini daha sonra herhangi bir yerel önbellegin bu satırı sahip olup olmadığını belirleyebilir ve eğer öyleyse, temizlenmesine neden olabilir. Eğer gerçek bellek konumu yayın bildirimini alan düğümdeyse, o zaman bu düğümün

dizininin o bellek satırının geçersiz olduğunu ve bir geri yazma işlemi gerçekleşene kadar öyle kalacağını belirten bir girdi tutması gereklidir. Başka bir işlemci (yerel veya uzak) geçersiz satırı talep ederse, yerel dizin veriyi sağlamadan önce belleği güncellemek için bir geri yazmaya zorlamalıdır.

## NUMA Artıları ve Eksileri

Bir CC-NUMA sisteminin ana avantajı, büyük yazılım değişiklikleri gerektirmeden SMP'den daha yüksek paralellik seviyelerinde etkili performans sunabilmesidir. Birde fazla NUMA düğümüyle, herhangi bir düğümdeki veri yolu trafiği veri yolumun kaldırabilecegi bir taleple sınırlanır. Ancak, bellek erişimlerinin çoğu uzak düğümlere yapılmıysa, performans başlar. Bu performans düşününün önlenebileceğine inanmak için nedenler vardır. İlk olarak, L1 ve L2 önbelleklerinin kullanımı, uzak olanlar da dahil olmak üzere tüm bellek erişimlerini en aza indirmek için tasarlanmıştır. Eğer yazılımın büyük bir kısmı iyi bir zamansal yerellige sahipse, o zaman uzak bellek erişimleri aşırı olmamalıdır. İkinci olarak, yazılım iyi bir uzamsal yerellige sahipse ve sanal bellek kullanılıyorsa, bir uygulama için gereklen veriler, başlangıçta çalışan uygulamanın yerel belleğine yüklenenin sınırlı sayıda sık kullanılan sayfada bulunacaktır. Sequent tasarımcıları bu tür uzamsal yerelligin temsili uygulamalarda görüldüğünü bildirmektedir [LOVE96]. Son olarak, sanal bellek şeması, işletim sistemine bir sanal bellek sayfasını sık kullanılan bir düğüme taşıyacak bir sayfa taşıma mekanizması dahil edilerek geliştirilebilir; Silicon Graphics tasarımcıları bu yaklaşımın başarılı olduğunu bildirmiştir [WHIT97].

Uzaktan erişimden kaynaklanan performans düşüşü ele alınsa bile, CC-NUMA yaklaşımı için iki dezavantaj daha vardır [PFIS98]. Birincisi, bir CC-NUMA şeffaf bir şekilde SMP gibi görünmez; bir işletim sistemini ve uygulamaları SMP'den CC-NUMA sistemine taşımak için yazılım değişiklikleri gerekecektir. Bunlar daha önce bahsedilen sayfa tahsisi, süreç tahsisi ve işletim sistemi tarafından yük dengelemeyi içerir. İkinci bir endişe de kullanılabilirliktir. Bu oldukça karmaşık bir konudur ve CC-NUMA sisteminin tam olarak uygulanmasına bağlıdır; ilgilenen okuyucu [PFIS98]'e yönlendirilir.



Vektör İşlemci Simülatörü

## 17.7 BULUT BİLİŞİM

Bulut bilişim, üç hizmet modelinin ele alındığı Bölüm 1'de tanıtılmıştır. Burada daha fazla ayrıntıya giriyoruz.

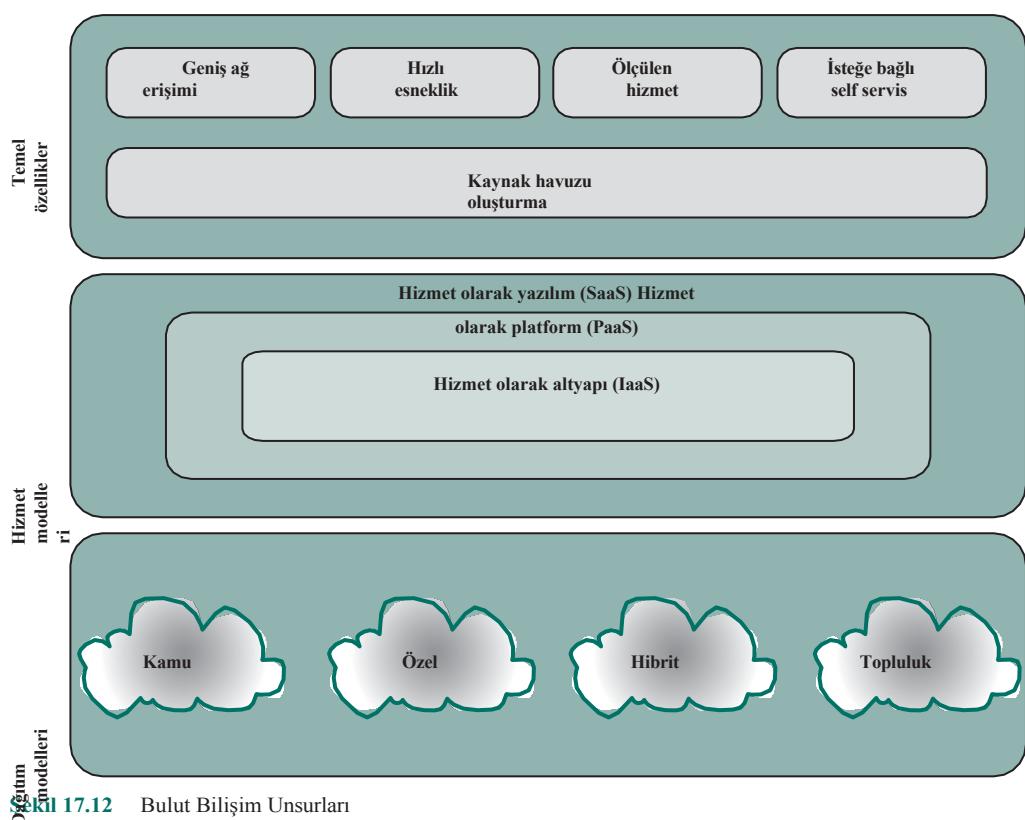
### Bulut Bilişim Unsurları

NIST SP-800-145 (*Bulut Bilişimin NIST Tanımı*), **bulut bilişimin** beş temel özellik, üç hizmet modeli ve aşağıdakilerden oluştuğunu belirtir

## 644 BÖLÜM 17 / PARALEL İŞLEM

dört dağıtım modeli. Şekil 17.12'de bu özellikler arasındaki ilişki gösterilmektedir. Bulut bilişim'in temel özellikleri aşağıdakileri içerir:

- **Geniş ağ erişimi:** Yetenekler ağ üzerinden kullanılabilir ve heterojen ince veya kalın istemci platformlarının (örn. cep telefonları, dizüstü bilgisayarlar ve tabletler) yanı sıra diğer geleneksel veya bulut tabanlı yazılım hizmetleri tarafından kullanımı teşvik eden standart mekanizmalar aracılığıyla erişilebilir.
- **Hızlı esneklik:** Bulut bilişim, size özel hizmet gereksiniminize göre kaynakları genişletme ve azaltma olağanı sağlar. Örneğin, belirli bir görev süresince çok sayıda sunucu kaynağına ihtiyacınız olabilir. Daha sonra görev tamamlandığında bu kaynakları serbest bırakabilirsiniz.
- **Ölçülen hizmet:** Bulut sistemleri, hizmet türüne (örn. depolama, işleme, bant genişliği ve aktif kullanıcı hesapları) uygun bir soyutlama düzeyinde bir ölçüm özelliğinden yararlanarak kaynak kullanımını otomatik olarak kontrol eder ve optimize eder. Kaynak kullanımı izlenebilir, kontrol edilebilir ve raporlanabilir, böylece kullanılan hizmetin hem sağlayıcısı hem de tüketici için şeffaflık sağlanır.
- **Talep üzerine self servis:** Bir tüketici, sunucu süresi ve ağ depolama alanı gibi bilgi işlem yeteneklerini gerektirdiğinde otomatik olarak tek taraflı olarak sağlayabilir



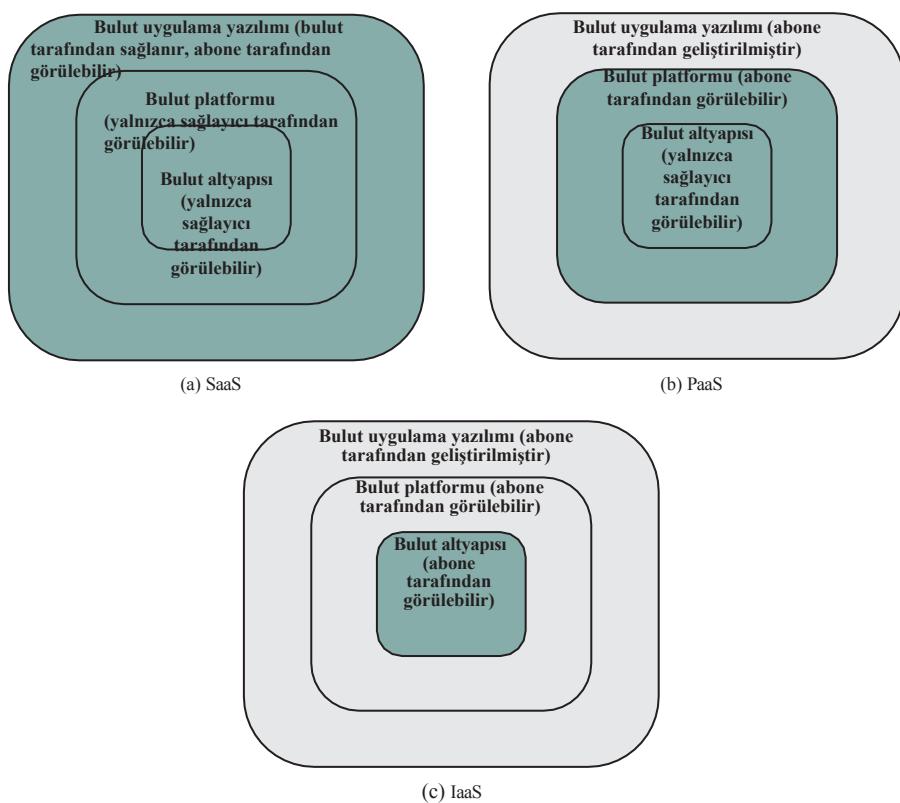
Şekil 17.12 Bulut Bilişim Unsurları

Her bir hizmet sağlayıcı ile insan etkileşimi gerektirmeden. Hizmet isteği bağlı olduğundan, kaynaklar BT altyapınızın kalıcı parçaları değildir.

- **Kaynak havuzu:** Sağlayıcının bilgi işlem kaynakları çok kiracılı bir model kullanılarak birden fazla tüketiciye hizmet vermek üzere bir havuzda toplanır ve farklı fiziksel ve sanal kaynaklar tüketici talebine göre dinamik olarak atanır ve yeniden tahsis edilir. Müşterinin genel olarak sağlanan kaynakların tam konumu üzerinde hiçbir kontrolü veya bilgisi olmadığı, ancak daha yüksek bir soyutlama düzeyinde (örneğin, ülke, eyalet veya veri merkezi) konumu belirleyebildiği için bir dereceye kadar konum bağımsızlığı vardır. Kaynaklara örnek olarak depolama, işleme, bellek, ağ bant genişliği ve sanal makineler verilebilir. Özel bulutlar bile kaynakları aynı kuruluşun farklı bölümleri arasında havuzlama eğilimindedir.

NIST, iç içe geçmiş hizmet alternatifleri olarak görülebilecek üç **hizmet modeli** tanımlamaktadır (Şekil 17.13). Bunlar Bölüm 1'de tanımlanmıştır ve kısaca aşağıdaki şekilde özetlenebilir:

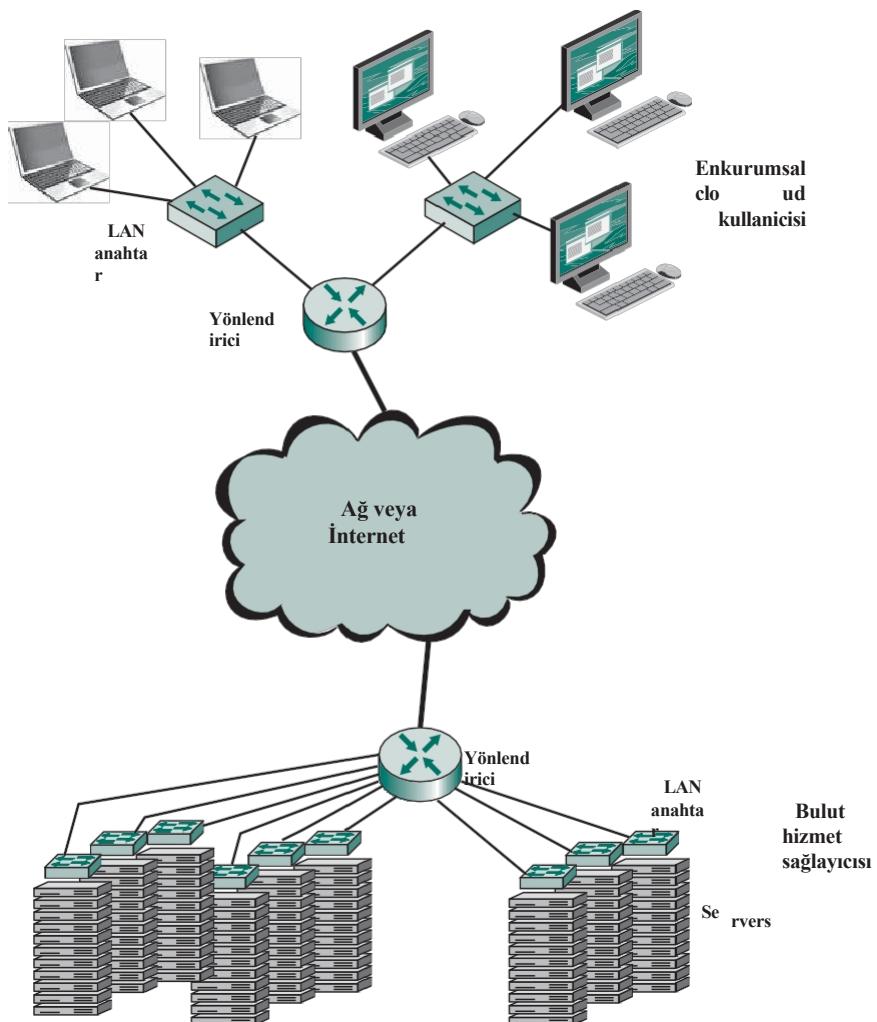
- **Hizmet olarak yazılım (SaaS):** Müşterilere bulut üzerinde çalışan ve buluttan erişilebilen yazılım, özellikle de uygulama yazılımı şeklinde hizmet sağlar.



**Şekil 17.13** Bulut Hizmet Modelleri

- **Hizmet olarak platform (PaaS):** Müşterilere, müşterinin uygulamalarının üzerinde çalışabileceği bir platform şeklinde hizmet sağlar.
  - **Hizmet olarak altyapı (IaaS):** Müşteriye alta yatan bulut altyapısına erişim sağlar.
- NIST dört **dağıtım modeli** tanımlar:
- **Genel bulut:** Bulut altyapısı genel kamunun veya büyük bir endüstri grubunun kullanımına sunulur ve bulut hizmetleri satan bir kuruluşu aittir. Bulut sağlayıcısı hem bulut altyapısından hem de bulut içindeki veri ve işlemlerin kontrolünden sorumludur. Genel bulutun en büyük avantajı maliyettir. Abone olan bir kuruluş yalnızca ihtiyaç duyduğu hizmetler ve kaynaklar için ödeme yapar ve bunları gerektiği gibi ayarlayabilir. Ayrıca, abonenin yönetim yükü büyük ölçüde azalır. Temel endişe güvenlidir. Bununla birlikte, güçlü güvenlik kontrolleri sergileyen bir dizi genel bulut sağlayıcısı vardır ve aslında bu tür sağlayıcılar, özel bir bulutta bulunabilecek güvenlik için daha fazla kaynak ve uzmanlığa sahip olabilir.
  - **Özel bulut:** Özel bulut, kuruluşun dahili BT ortamında uygulanan bir bulut altyapısıdır. Kuruluş bulutu kendi bünyesinde yönetmeyi seçebilir veya yönetim işlevini üçüncü bir tarafa ihale edebilir. Ayrıca, bulut sunucuları ve depolama cihazları kurum içinde veya kurum dışında bulunabilir. Özel bulutu tercih etmenin temel motivasyonlarından biri güvenlidir. Özel bir bulut altyapısı, veri depolamanın coğrafi konumu ve güvenliğin diğer yönleri üzerinde daha sıkı kontroller sunar.
  - **Topluluk bulutu:** Topluluk bulutu, özel ve genel bulutların özelliklerini paylaşır. Özel bir bulut gibi, bir topluluk bulutu da herhangi bir alt yazmana açık değildir. Genel bulutta olduğu gibi, bulut kaynakları bir dizi bağımsız kuruluş arasında paylaşılır. Topluluk bulutunu paylaşan kuruluşlar benzer gereksinimlere ve tipik olarak birbirleriyle veri alışverişi yapma ihtiyacına sahiptir. Topluluk bulutu konseptini kullanan sektörlerde örnek olarak sağlık hizmetleri verilebilir. Devletin gizlilik ve diğer düzenlemelerine uymak için bir topluluk bulutu uygulanabilir. Topluluk katılımcıları kontrollü bir şekilde veri alışverişi yapabilirler. Bulut altyapısı katılımcı kuruluşlar veya üçüncü bir tarafça yönetilebilir ve içinde veya dışında bulunabilir. Bu dağıtım modelinde, maliyetler genel buluttan daha az (ancak özel buluttan daha fazla) kullanıcıya yayılır, bu nedenle bulut bilişimin maliyet tasarrufu potansiyelinin yalnızca bir kısmı gerçekleştirilir.
  - **Hibrit bulut:** Bulut altyapısı, benzersiz varlıklar olarak kalan ancak veri ve uygulama taşınabilirliği sağlayan standartlaştırılmış veya tescilli teknoloji ile birbirine bağlanan iki veya daha fazla bulutun (özel, topluluk veya genel) bir bileşimidir (örneğin, bulutlar arasında yük dengeleme için bulut patlaması). Bir hibrit bulut çözümü ile hassas bilgiler bulutun özel bir alanına yerleştirilebilir ve daha az hassas veriler genel bulutun maliyet avantajlarından yararlanabilir.

Sekil 17.14 tipik bulut hizmeti bağlamını göstermektedir. Bir kuruluş, bir ağ veya İnternet üzerinden bir yönlendirici ile bulut hizmet sağlayıcısına bağlanan bir kurumsal LAN veya LAN kümesi içinde iş istasyonları bulundurur. Bulut hizmet sağlayıcısı, çeşitli sunucularla yönettiği büyük bir sunucu koleksiyonuna sahiptir.



**Şekil 17.14** Bulut Bilişim BağlAMI

ağ yönetimi, yedeklilik ve güvenlik araçları. Şekilde bulut altyapısı, yaygın bir mimari olan blade sunucuların bir koleksiyonu olarak gösterilmektedir.

### Bulut Bilişim Referans Mimarisi

NIST SP 500-292 (*NIST Bulut Bilişim Referans Mimarisi*), aşağıdaki gibi tanımlanan bir referans mimarisi oluşturur:

NIST bulut bilişim referans mimarisi, bir "nasıl yapılır" tasarım çözümü ve uygulamasına değil, bulut hizmetlerinin "ne" sağladığını ilişkin gereksinimlere odaklanmaktadır. Referans mimarı, bulut bilişimdeki operasyonel karmaşıklıkların anlaşılması kolaylaştırmayı amaçlamaktadır. Belirli bir bulut bilişim sisteminin sistem mimarisini temsil etmez; bunun yerine ortak bir referans çerçevesi kullanarak sisteme özgü bir mimariyi tanımlamak, tartışmak ve geliştirmek için bir araçtır.

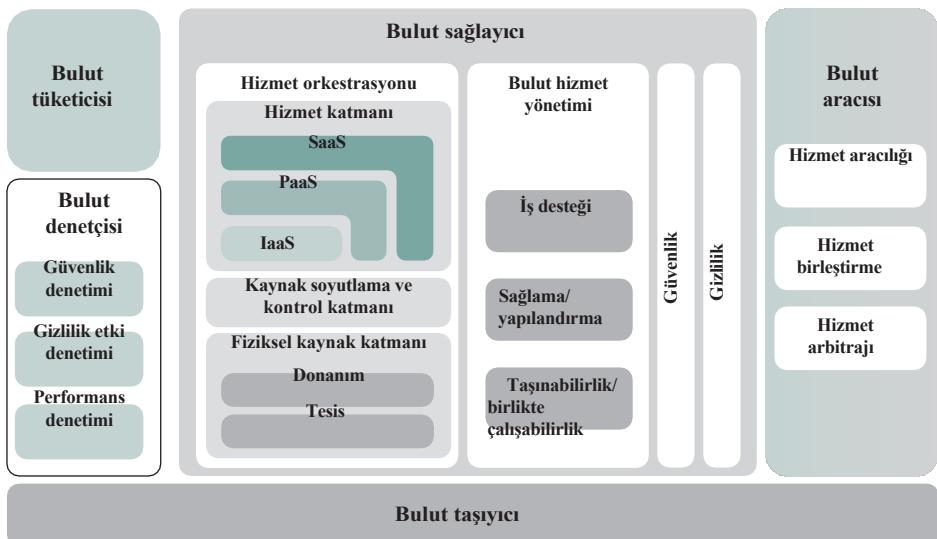
## 648 BÖLÜM 17 / PARALEL İŞLEM

NIST referans mimariyi aşağıdaki hedefleri göz önünde bulundurarak geliştirmiştir:

- Genel bir bulut bilişim kavramsal modeli bağlamında çeşitli bulut hizmetlerini göstermek ve anlamak.
- Tüketicilerin bulut hizmetlerini anlaması, tartışması, sınıflandırması ve karşılaştırması için teknik bir referans sağlamak.
- Güvenlik, birlikte çalışabilirlik ve taşınabilirlik için aday standartların ve referans uygulamaların analizini kolaylaştırmak.

Şekil 17.15'te gösterilen referans mimari, roller ve sorumluluklar açısından beş ana aktörü :

- **Bulut tüketici:** Bulut sağlayıcıları ile iş ilişkisi sürdürün ve bu sağlayıcıların hizmetlerini kullanan kişi veya kuruluş.
- **Bulut sağlayıcı (CP):** Bir hizmeti ilgili tarafların kullanımına sunmaktan sorumlu kişi, kuruluş veya varlık.
- **Bulut denetçisi:** Bulut hizmetlerinin, bilgi sistemi operasyonlarının, performansın ve bulut uygulamasının güvenliğinin bağımsız değerlendirmesini yapabilecek bir taraf.
- **Bulut aracı:** Bulut hizmetlerinin kullanımını, performansını ve teslimatını yöneten ve CP'ler ile bulut tüketicileri arasındaki ilişkileri müzakere eden bir varlık.
- **Bulut taşıyıcı:** Bulut hizmetlerinin CP'lerden bulut tüketicilerine bağlanması ve taşınmasını sağlayan bir aracı.



Şekil 17.15 NIST Bulut Bilişim Referans Mimarisi

Bulut tüketici ve sağlayıcısının rolleri daha önce tartışılmıştır. Öztemek gerekirse, bir **bulut sağlayıcısı, bulut tüketicilerinin** BT ve iş gereksinimlerini karşılamak için bir veya daha fazla bulut hizmeti sağlayabilir. Üç hizmet modelinin (SaaS, PaaS, IaaS) her biri için CP, bulut hizmeti tüketicileri için bir bulut arayüzü ile birlikte söz konusu hizmet modelini desteklemek için gereken depolama ve işleme olanaklarını sağlar. SaaS için CP, yazılım uygulamalarının çalışmasını bir bulut altyapısı üzerinde dağıtır, yapılandırır, sürdürür ve günceller, böylece hizmetler bulut müşterilerine beklenen hizmet seviyelerinde sağlanır. SaaS tüketicileri, üyelerine yazılım erişim sağlayan kuruluşlar, yazılım uygulamalarını doğrudan kullanan son kullanıcılar veya son kullanıcılar için uygulamaları yapılandıran yazılım uygulama yöneticileri olabilir.

PaaS için CP, platform için bilgi işlem altyapısını yönetir ve çalışma zamanı yazılım yürütme yiğini, veritabanları ve diğer ara katman bileşenleri platform bileşenlerini sağlayan bulut yazılımını çalıştırır. PaaS'ın bulut tüketicileri, bir bulut ortamında barındırılan uygulamaları geliştirmek, test etmek, dağıtmak ve yönetmek için CP'ler tarafından sağlanan araçları ve yürütme kaynaklarını kullanabilir.

IaaS için CP, sunucular, ağlar, depolama ve barındırma altyapısı dahil olmak üzere hizmetin temelini oluşturan fiziksel bilgi işlem kaynaklarını satın alır. IaaS bulut tüketici de temel bilgi işlem ihtiyaçları için sanal bilgisayar gibi bu bilgi işlem kaynaklarını kullanır.

**Bulut taşıyıcıı**, bulut tüketicileri ve CP'ler arasında bulut hizmetlerinin bağlantısını ve taşınmasını sağlayan bir ağ tesisidir. Tipik olarak bir CP, bulut tüketicilerine sunulan SLA'ların seviyesiyle tutarlı hizmetler sağlamak için bir bulut taşıyıcısıyla hizmet seviyesi anlaşmaları (SLA'lar) yapar ve bulut taşıyıcısından bulut ile CP'ler arasında özel ve güvenli bağlantılar sağlamasını isteyebilir.

Bir **bulut aracı**, bulut hizmetleri bir bulut sağlayıcısının kolayca yönetemeyeceği kadar karmaşık olduğunda faydalıdır. Bir bulut aracı üç alanda destek sunabilir:

- **Hizmet aracılığı:** Bunlar kimlik yönetimi, performans raporlaması ve gelişmiş güvenlik katma değerli hizmetlerdir.
- **Hizmet birleştirme:** Aracı, tek bir CP tarafından özel olarak ele alınmayan tüketici ihtiyaçlarını karşılamak veya performansı optimize etmek veya maliyeti en aza indirmek için birden fazla bulut hizmetini birleştirir.
- **Hizmet arbitrajı:** Bu, bir araya getirilen hizmetlerin sabit olmaması dışında hizmet bir araya getirmeye benzer. Hizmet arbitrajı, bir brokerin birden fazla acenteden hizmet seçme esnekliğine sahip olduğu anlamına gelir. Örneğin bulut aracı, en iyi puana sahip bir acenteyi ölçmek ve seçmek için bir kredi puanlama hizmeti kullanabilir.

Bir **bulut denetçisi**, bir CP tarafından sağlanan hizmetleri güvenlik kontrolleri, gizlilik etkisi, performans . açısından değerlendirebilir. Denetçi, CP'nin bir dizi standarda uygunluğunu temin edebilecek bağımsız bir kurumdur.

## 17.8 ANAHTAR TERIMLER, GÖZDEN GEÇİRME SORULARI VE PROBLEMLER

### Anahtar Terimler

aktif yedek önbellek tutarlılığı kümesi bulut denetçi bulut aracısı bulut taşıyıcısı bulut bilişim bulut tüketicisı bulut sağlayıcısı topluluğu bulut dizini protokoltü	failback yük devretme hibrıt bulut hizmet olarak altyapı (IaaS) MESI protokolü çoklu işlemci tekdüze olmayan bellek erişimi (NUMA) pasif bekleme hizmet olarak platform (PaaS)	özel bulut genel bulut hizmet toplama hizmet arbitrajı hizmet aracılığı snoopy protokolü hizmet olarak yazılım (SaaS) simetrik çoklu işlemci (SMP) tek tip bellek erişimi (UMA) tek işlemci
--	--	--

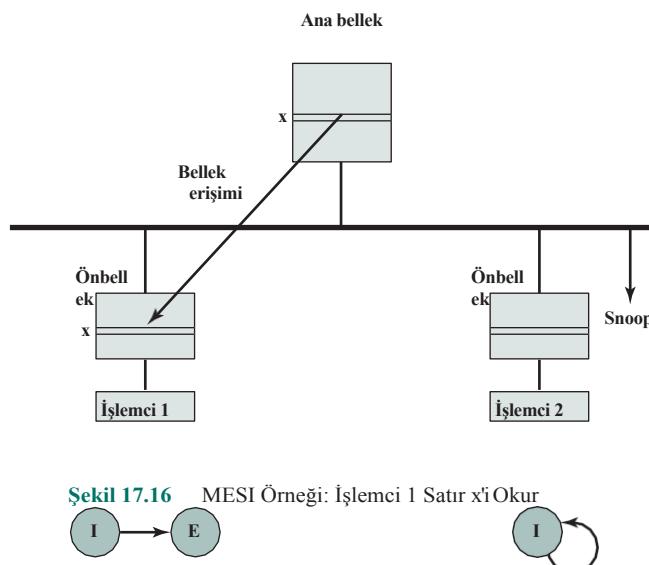
### Inceleme Soruları

- 17.1** Üç tür bilgisayar sistemi organizasyonunu listeleyiniz ve kısaca tanımlayınız.
- 17.2** Bir SMP'nin başlıca özellikleri nelerdir?
- 17.3** Tek işlemciye kıyasla bir SMP'nin potansiyel avantajlarından bazıları nelerdir?
- 17.4** Bir SMP için temel işletim sistemi tasarım sorunlarından bazıları nelerdir?
- 17.5** Yazılım ve donanım önbellek tutarlı şemaları arasındaki fark nedir?
- 17.6** MESI protokolündeki dört durumun her birinin anlamı nedir?
- 17.7** Kümelemenin bazı temel faydaları nelerdir?
- 17.8** Yük devretme ve geri yükleme arasındaki fark nedir?
- 17.9** UMA, NUMA ve CC-NUMA arasındaki farklar nelerdir?
- 17.10** Bulut bilişim referans mimarisi nedir?

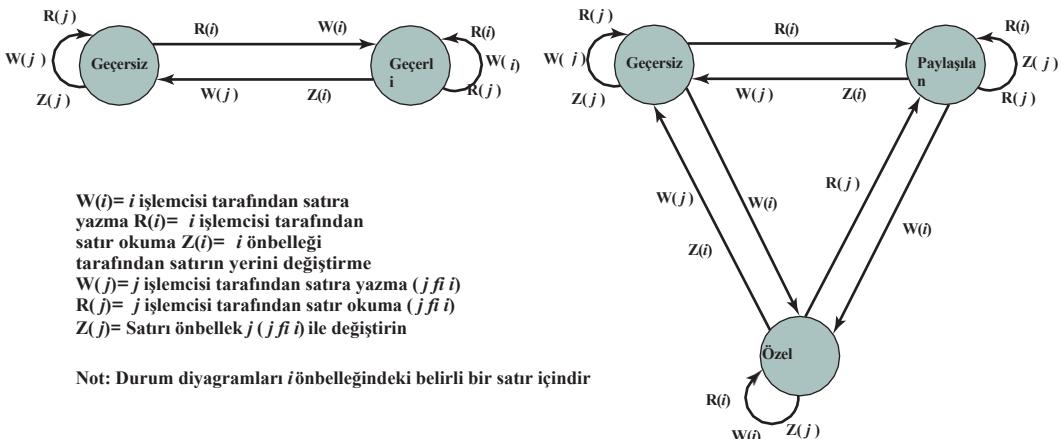
### Problemler

- 17.1** Bir bilgisayar sisteminde  $n$  işlemci tarafından aynı anda yürütülebilecek program kodu yüzdesi olsun. Kalan kodun tek bir işlemci tarafından sırayla yürütülmeli gerektiğini varsayıyalım. Her işlemci  $x$  MIPS'lik bir yürütme hızına sahiptir.
  - a. Bu programın hariç tutularak yürütülmesi için sistem kullanıldığından etkin MIPS oranı için  $n$ ,  $x$  ve  $y$  cinsinden bir ifade türetin.
  - b. Eğer  $n= 16$  ve  $x= 4$  MIPS ise, 40 MIPS'lik bir sistem performansı sağlayacak değeri belirleyin.
- 17.2** Sekiz işlemcili bir çoklu işlemcide 20 adet bağlı teyp sürücüsü bulunmaktadır. Sisteme gönderilen ve yürütmemi tamamlamak için her biri en fazla dört teyp sürücüsü gerektiren çok sayıda iş vardır. Her bir işin uzun bir süre boyunca sadece üç teyp sürücüsü ile çalışmaya başladığını, ardından çalışmasının sonuna doğru kısa bir süre için dördüncü teyp sürücüsüne ihtiyaç duyduğunu varsayıyın. Ayrıca bu tür işlerin sonsuz sayıda olduğunu varsayıyın.
  - a. İşletim sistemindeki zamanlayıcının dört teyp sürücüsü mevcut olmadığı sürece bir işi başlatmayacağı varsayıyın. Bir iş, dört sürücü hemen atanır ve iş bitene kadar serbest bırakılmaz. Aynı anda devam edebilecek maksimum iş sayısı nedir? Bu politikanın bir sonucu olarak boşta bırakılabilenek maksimum ve minimum teyp sürücüsü sayısı nedir?

- b.** Teyp sürücüsü kullanımını iyileştirmek ve aynı zamanda sistemin kilitlenmesini önlemek için alternatif bir politika önerin. Aynı anda devam edebilecek maksimum iş sayısı nedir? Boşa duran teyp sürücülerinin sayısına ilişkin sınırlar nelerdir?
- 17.3** Veri yolu tabanlı çoklu işlemcilerde bir kez yazma önbelleği yaklaşımıyla ilgili herhangi bir sorun öngörebilir misiniz? Eğer öyleyse, bir çözüm öneriniz.
- 17.4** Bir SMP yapılandırmasındaki iki işlemcinin zaman içinde ana bellekten aynı veri satırına erişmek istediği bir durumu düşünün. Her iki işlemcinin de bir önbelleği vardır ve MESI protokolünü kullanmaktadır. Başlangıçta, her iki önbellekte de satırın geçersiz bir kopyası vardır. Şekil 17.16, x satırının İşlemci P1 tarafından okunmasının sonucunu göstermektedir. Eğer bu bir erişim dizisinin başlangıcısısa, takip eden dizi için sonraki şekilleri çizin:
1. P2 x'i okur.
  2. P1 x'e yazar (anlaşılr olmasi için P1'in önbelleğindeki satırı x' olarak etiketleyin).
  3. P1 x'e yazar (P1'in önbelleğindeki satırı x" olarak etiketleyin).
  4. P2 x'i okur.
- 17.5** Şekil 17.17 iki olası önbellek tutarlılığı protokolünün durum diyagramlarını göstermektedir. Her bir protokolü çıkarıp açıklayın ve her birini MESI ile karşılaştırın.
- 17.6** MESI protokolünü kullanan hem L1 hem de L2 önbellekli bir SMP düşünün. Bölüm 17.3'te açıklandığı gibi, dört durumdan biri L2 önbelleğindeki her satırla ilişkilendirilir. L1 her satır için de dört durum gereklidir. Eğer öyleyse, neden? Değilse, hangi durum ya da durumların ortadan kaldırılabilceğini açıklayın.
- 17.7** IBM ana bilgisayarının önceki bir sürümü olan S/390 G4, üç seviyeli önbellek kullanıyordu. Z990'da olduğu gibi, yalnızca ilk seviye işlemci yongası üzerindeydi [İşlemci birimi (PU) olarak adlandırılır]. L2 önbelleği de z990'a benziyordu. L3 önbellek, bellek denetleyicisi olarak görev yapan ayrı bir çip üzerindeydi ve L2 önbellekler ile bellek kartları arasına yerleştirilmiştir. Tablo 17.3, IBM S/390 için üç seviyeli önbellek düzenlemesinin performansını göstermektedir. Bu sorunun amacı, üçüncü seviye önbelleğin dahil edilmesinin olup olmadığını belirlemektir. Yalnızca L1 önbelleği olan bir sistem için erişim cezasını (ortalama PU döngü sayısı) belirleyin ve bu değeri 1,0'a normalleştirin. Ardından hem L1 hem de L2 önbelleği olduğunda normalleştirilmiş erişim cezasını belirleyin.



## 652 BÖLÜM 17 / PARALEL İŞLEM



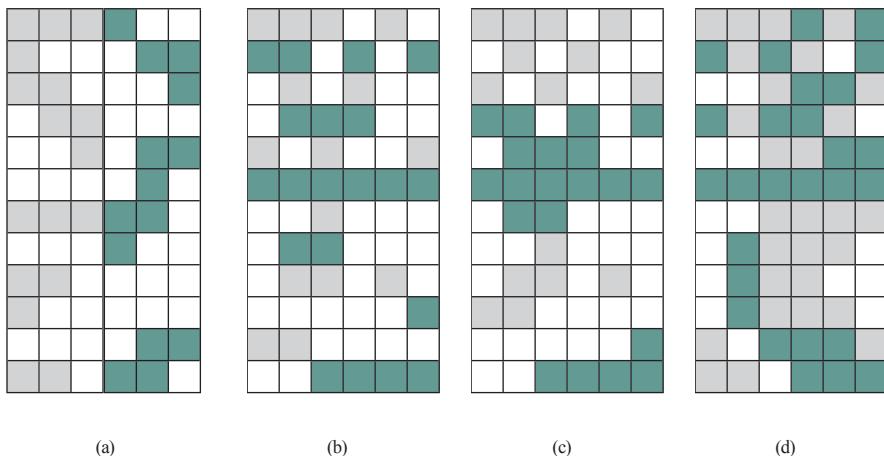
**Şekil 17.17** İki Önbellek Tutarlılık Protokolü

kullanıldığından ve üç önbellek de kullanıldığından erişim cezası. Her bir durumda iyileştirme miktarını not edin ve L3 önbelleginin değerilarındaki görüşünüzü belirtin.

- 17.8** a. Sırasıyla  $H_d$  ve  $H_i$  isabet oranlarına sahip ayrı veri ve komut önbellekleri olan bir tek işlemci düşünün. İşlemciden önbellege erişim süresi  $c$  saat çevrimidir ve bellek ile önbellek arasındaki bir blok için aktarım süresi  $b$  saat çevrimidir.  $f_i$  bellek erişimlerinin talimatlar için olan kısmı ve  $f_{(d)}$  veri önbellegindeki kirli satırların değiştirilen satırlar arasındaki kısmı olsun. Bir geri yazma politikası varsayıyın ve etkin bellek erişim süresini az önce tanımlanan parametreler açısından belirleyin.  
 b. Şimdi her işlemcinin (a) bölümündeki özelliklere sahip olduğu veri yolu tabanlı bir SMP varsayıyalım. Her işlemci bellek okuma ve yazma işlemlerine ek olarak önbellek geçersiz kılma işlemlerini de gerçekleştirmelidir. Bu da etkin bellek erişim süresini etkiler. Geçersiz kılma sinyallerinin diğer veri önbelleklerine gönderilmesine neden olan veri referanslarının oranı  $f_{inv}$  olsun. Sinyali gönderen işlemcinin geçersiz kılma işlemini tamamlaması için  $t$  saat döngüsü gereklidir. Diğer işlemciler geçersiz kılma işlemine dahil değildir. Etkin bellek erişim süresini belirleyin.
- 17.9** Şekil 17.18'deki resimlerin her biri hangi organizasyonel alternatifin önermektedir?
- 17.10** Şekil 17.7'de, bazı diyagramlar kısmen doldurulmuş yatay satırlar göstermektedir. Diğer durumlarda, tamamen boş olan satırlar vardır. Bunlar iki farklı verimlilik kaybı türünü temsil etmektedir. Açıklayınız.
- 17.11** Şekil 14.13b'deki boru hattı tasvirini düşünün; bu tasvir, A iş parçacığının yürütülmesini temsil etmek için getirme ve kod çözme aşamaları yok sayılıarak Şekil 17.19'a'da yeniden çizilmiştir.

**Tablo 17.3** S/390 SMP Yapılandırmasında Tipik Önbellek Isabet Oranı [MAK97]

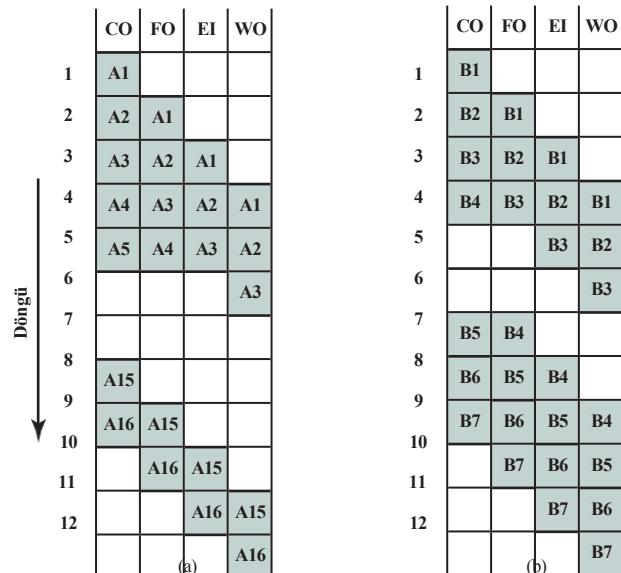
Bellek Alt Sistemi	Erişim Cezası (PU döngüler)	Önbellek Boyutu	Isabet Oranı (%)
L1 önbellek	1	32 KB	89
L2 önbellek	5	256 KB	5
L3 önbellek	14	2 MB	3
Hafıza	32	8 GB	3



Şekil 17.18 Problem 17.9 için Diyagram

Şekil 17.19b ayrı bir B iş parçacığının yürütülmesini göstermektedir. Her iki durumda da basit bir pipelined işlemci kullanılmıştır.

- Her iki iş parçacığı için Şekil 17.7a'ya benzer bir komut sorun diyagramı gösterin.
- İki iş parçacığının bir çip çoklu işlemcide paralel yürütüleceğini ve cipteki iki çekirdeğin her birinin basit bir ardışık düzen kullandığını varsayıñ. Şekil 17.7k'ya benzer bir komut sorunu diyagramı gösterin. Ayrıca Şekil 17.19 tarzında bir boru hattı yürütme diyagramı gösteriniz.
- İki sorulun bir süperskalar mimari varsayıñ. Veri bağımlılığı olmadığını varsayıarak (b) bölümünü serpiştiirilmiş çok iş parçacıklı bir süperskalar uygulama için tekrarlayın.



Şekil 17.19 İki Yürütme İş Parçacığı

## 654 BÖLÜM 17 / PARALEL İŞLEM

*Not:* Tek bir cevap yoktur; gecikme süresi ve öncelik hakkında varsayımlarda bulunmanız gereklidir.

- d. Bloklanmış çok iş parçacıklı süper skaler uygulama için (c) bölümünü tekrarlayın.  
e. Dört sorunlu SMT mimarisini için tekrarlayın.
- 17.12** Bir uygulama programı dokuz bilgisayarlı bir kümede çalıştırılır. Bir kıyaslama programı bu küme üzerinde  $T$  kadar zaman almıştır. Ayrıca,  $T$  süresinin %25'inin uygulamanın dokuz bilgisayarda aynı anda çalıştığı süre olduğu tespit edilmiştir. Kalan zamanda uygulama tek bir bilgisayarda çalışmak zorunda kalmıştır.
- a. Programın tek bir bilgisayarda yürütülmesine kıyasla yukarıda belirtilen koşul altında etkin hızlanmayı hesaplayın. Ayrıca, önceki programda paralelleştirilmiş (küme modunu kullanacak şekilde programlanmış veya derlenmiş) kod yüzdesini de hesaplayın.  
b. Kodun paralelleştirilmiş kısmında 9 bilgisayar yerine 17 bilgisayarı etkin bir şekilde kullanabildiğimizi varsayıyalım. Elde edilen etkin hız artısını hesaplayın.
- 17.13** Aşağıdaki FORTRAN programı bir bilgisayarda çalıştırılacak ve paralel bir versiyonu 32 bilgisayarlı bir kümede çalıştırılacaktır.

```
L1:      DO 10 I= 1, 1024
L2:          SUM(I (= 0
L3:      DO 20J= 1,I
L4: 20           SUM(I (= SUM(I (+ I
L5:      10 CONTINUE
```

2. ve 4. satırların her birinin tüm işlemci ve bellek erişim faaliyetleri dahil olmak üzere iki makine çevrim süresi aldığına varsayıyalım. Yazılım döngüsü kontrol deyimlerinin (satır 1, 3, 5) neden olduğu ek yükü ve diğer tüm sistem ek yükünü ve kaynak çakışmalarını göz ardı edin.

- a. Programın tek bir bilgisayarda toplam yürütme süresi (makine çevrim süreleri cinsinden) nedir?  
b. I-döngüsünün yinelemelerini 32 bilgisayar arasında aşağıdaki gibi bölgüstürün: Bilgisayar 1 ilk 32 iterasyonu yürütür ( $I= 1$  ila 32), işlemci 2 sonraki 32 iterasyonu yürütür ve bu böyle devam eder. Bölüm (a) ile karşılaşıldığında yürütme süresi ve hızlandırma faktörü nedir? ( $J$  döngüsü tarafından belirlenen hesaplama iş yükünün bilgisayalar arasında dengeleştirildiğini unutmayın).  
c. Tüm hesaplama iş yükünün 32 bilgisayar üzerinde dengeli bir şekilde paralel yürütülmesini kolaylaştırmak için paralelleştirmenin nasıl değiştirileceğini açıklayın. Dengeli bir yük ile her iki döngüye göre her bilgisayara eşit sayıda ekleme atanması kastedilmektedir.  
d. Paralel yürütme sonucunda 32 bilgisayarda elde edilen minimum yürütme süresi nedir? Tek bir bilgisayara göre elde edilen hız artışı nedir?

- 17.14** İki vektörü toplayan bir programın aşağıdaki iki versiyonunu göz önünde bulundurun:

<pre>L1:      DO 10 I= 1, N L2:          A(I (= B(I(+ C(I( L3: 10    DEVAM L4:          SUM= 0 L5:      DO 20J= 1, N L6:          SUM= SUM+ A(J( L7: 20    DEVAM</pre>	<pre>DOALL K= 1, M DO 10 I= L(K-1(+1, KL A(I (= B(I(+C(I( 10 DEVAM SUM(K (= 0 DO 20 J= 1, L SUM(K (= SUM(K (+ A(L(K-1 (+J( 20 DEVAM ENDALL</pre>
--	--

- a. Soldaki program tek bir işlemci üzerinde yürütülmektedir. L2, L4 ve L6 kod satırlarının her birinin yürütülmesinin bir işlemci saat döngüsü aldığına varsayıyalım. Basitlik açısından, diğer kod satırları için gereken süreyi göz ardı edin. Başlangıçta tüm diziler ana belleğe yüklenmiştir ve kısa program parçası komut önbelleğindedir. Bu programı çalıştırmak için kaç saat çevrimi gereklidir?

- b.** Sağdaki program  $M$  işlemcili bir çoklu işlemci üzerinde çalıştırılmak üzere yazılmıştır. Döngü işlemlerini bölüm başına  $L = N/M$  elemanlı  $M$  bölüme ayırıyoruz. DOALL, tüm  $M$  bölümlerinin paralel olarak yürütüldüğünü bildirir. Bu programın sonucu  $M$  kısmi toplam üretmektir. Paylaşılan bellek üzerinden her bir işlemciler arası iletişim işlemi için  $k$  saat döngüsü gerektiğini ve bu nedenle her bir kısmi toplamın toplanmasının  $k$  döngü gerektirdiğini varsayıyalım. Bir  $l$ -seviyeli ikili toplayıcı ağacı tüm kısmi toplamları bireleştirilebilir, burada  $l = \log_2 M$ . Nihai toplamı üretmek için kaç döngü gereklidir?
- c.** Dizide  $=2^{20}$  eleman ve  $=256$  eleman olduğunu varsayıyalım. Çoklu işlemci kullanarak elde edilen hız artışı nedir?  $k = 200$  olduğunu varsayıyalım. Bu, 256 katlık teorik hızlanmanın yüzde kaçıdır?

# BÖLÜM 18

## ÇOK ÇEKİRDEKLI BİLGİSAYARLAR

### 18.1 Donanım Performans Sorunları

Paralellik ve Karmaşıklıkta Artış Güç Tüketimi

### 18.2 Yazılım Performans Sorunları

Çok Çekirdekli Yazılım  
Uygulama Örneği: Valve Oyun Yazılımı

### 18.3 Çok Çekirdekli Organizasyon

Önbellek Düzeyleri Eşzamanlı  
Çoklu İş Parçacığı

### 18.4 Heterojen Çok Çekirdekli Organizasyon Farklı Komut

Seti Mimarileri Eşdeğer Komut Seti Mimarileri  
Önbellek Tutarlılığı ve MOESI Modeli

### 18.5 Intel Core i7-990X

### 18.6 ARM Cortex-A15 MPCore

Kesme İşleme Önbellek  
Tutarlılığı  
L2 Önbellek Tutarlılığı

### 18.7 IBM zEnterprise EC12 Mainframe

Organizasyon  
Önbellek Yapısı

### 18.8 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler

## ÖĞRENİM HEDEFLERİ

Bu bölümü çalıştıktan sonra şunları yapabilmelisiniz:

- Çok çekirdekli bilgisayarlara geçişe neden olan **donanım performansı sorunlarını** anlamak.
- Çok iş parçacıklı çok çekirdekli bilgisayarların kullanımının ortaya çıkardığı **yazılım performansı sorunlarını** anlamak.
- **Heterojen çok çekirdekli organizasyona** yönelik iki temel yaklaşımı genel bir bakış sunmak.
- Gömülü sistemlerde, PC'lerde ve sunucularda ve ana bilgisayarlarda **çok çekirdekli organizasyonun** kullanımını takdir etmek.

**Yonga çoklu işlemci** olarak da bilinen **çok çekirdekli** bir işlemci, iki veya daha fazla işlemci birimini (çekirdek olarak adlandırılır) tek bir silikon parçası (kalıp olarak adlandırılır) üzerinde birleştirir. Tipik olarak her bir çekirdek, bağımsız bir işlemcinin yazımcılar, ALU, boru hattı donanımı ve kontrol birimi gibi tüm bileşenlerinin yanı sıra L1 komut ve veri önbelleklerinden oluşur. Çoklu çekirdeklerle ek olarak, çağdaş çok çekirdekli yongalar L2 önbelleği ve giderek artan bir şekilde L3 önbelleği de içerir. Çip üzerinde sistemler (SoC'ler) olarak bilinen en yüksek düzeyde entegre çok çekirdekli işlemciler, bellek ve çevresel denetleyicileri de içerir.

Bu bölüm çok çekirdekli sistemlere genel bir bakış sunmaktadır. Çok çekirdekli bilgisayarların geliştirilmesine yol açan donanım performansı faktörlerine ve çok çekirdekli bir sistemin gücünden yararlanmanın yazılım zorluklarına bir bakışla başlıyoruz. Daha sonra, çok çekirdekli organizasyona bakıyoruz. Son olarak, kişisel bilgisayar ve iş istasyonu sistemleri (Intel), gömülü sistemler (ARM) ve ana bilgisayarıları (IBM) kapsayan üç çok çekirdekli ürün örneğini inceliyoruz.

## 18.1 DONANIM PERFORMANS SORUNLARI

Bölüm 2'de tartıştığımız gibi, mikroişlemci sistemleri on yillardır yürütme performansında istikrarlı bir artış yaşamıştır. Bu artış, saat frekansındaki artış, transistör yoğunluğunundaki artış ve çip üzerindeki işlemci organizasyonundaki iyileştirmeler gibi bir dizi faktörden kaynaklanmaktadır.

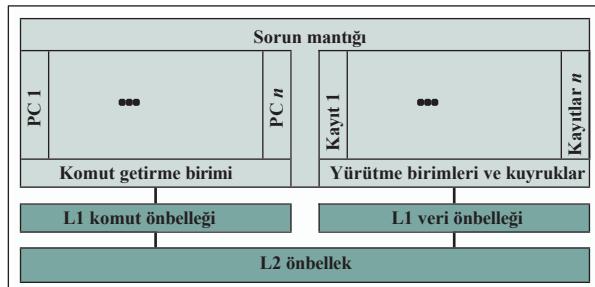
### Paralellik ve Karmaşıklıkta Artış

İşlemci tasarımindaki organizasyonel değişiklikler öncelikle ILP'den yararlanmaya odaklanmıştır, böylece her saat döngüsünde daha fazla iş yapılır. Bu değişiklikler kronolojik sırayla şunlardır (Şekil 18.1):

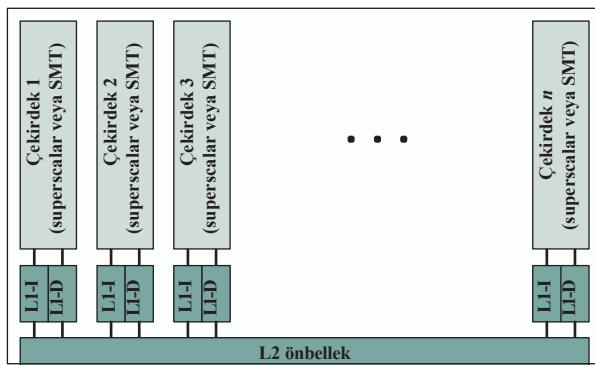
- **Boru hattı:** Bireysel talimatlar aşamalardan oluşan bir boru hattı üzerinden yürütülür, böylece bir talimat boru bir aşamasında yürütülürken, başka bir talimat boru hattının başka bir aşamasında yürütülür.
- **Superscalar:** Yürütme kaynaklarının çoğaltılmasıyla çoklu boru hatları oluşturulur. Bu, tehlikelerden kaçınıldığı sürece talimatların paralel boru hatlarında paralel yürütülmesini sağlar.



(a) Süperskalar



(b) Eşzamanlı çoklu iş parçacığı



(c) Çok Çekirdekli

**Şekil 18.1** Alternatif Çip Organizasyonları

- Eşzamanlı çoklu iş parçacığı (SMT):** Birden fazla iş parçacığının boru hattı kaynaklarının kullanımını paylaşabilmesi için kayıt bankaları genişletilir.

Bu yeniliklerin her birinde, tasarımcılar yıllar boyunca karmaşıklık ekleyerek sistemin performansını artırmaya çalışmışlardır. Boru hattı örneğinde, basit üç aşamalı boru hatlarının yerini beş aşamalı boru hatları almıştır. Intel'in Pentium 4 "Prescott" çekirdeği bazı komutlar için 31 aşamaya sahipti.

Bu eğilimin ne kadar ileri götürülebileceğinin pratik bir sınırı vardır, çünkü daha fazla aşama ile daha fazla mantık, daha fazla ara bağlantı ve daha fazla kontrol sinyaline ihtiyaç vardır. Süperskalar organizasyonla, paralel boru hattı sayısı artırılarak daha yüksek performans elde edilebilir. Yine, azalan getiriler vardır

boru hattı sayısı arttıkça. Tehlikeleri yönetmek ve komut kaynaklarını düzenlemek için daha fazla mantık gereklidir. Sonunda, tek bir yürütme iş parçacığı, tehlikelerin ve kaynak bağımlılıklarının mevcut çoklu boru hatlarının tam olarak kullanılmasını engellediği noktaya ulaşır. Ayrıca, derlenmiş ikili kod nadiren altıdan fazla paralel boru hattından yararlanmak için yeterli ILP'yi ortaya çıkarır.

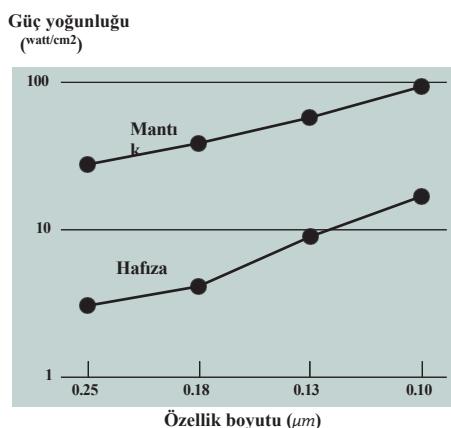
Aynı azalan getiri noktasına SMT ile ulaşılır, çünkü bir dizi boru hattı üzerinde birden fazla iş parçacığını yönetmenin karmaşıklığı, etkili bir şekilde kullanılabilecek iş parçacığı sayısını ve boru hattı sayısını sınırlar. SMT'nin avantajı, iki (veya daha fazla) program akışının mevcut ILP için aranabilmesi çerçevede yatkınlıkta.

Bilgisayar çipinin tasarımını ve imalatı ile ilgili bir dizi sorun bulunmaktadır. Çok uzun boru hatları, çoklu süperskalar boru hatları ve çoklu SMT kayıt bankaları ile ilgili tüm mantıksal sorunlarla başa çıkmak için karmaşıklığın artması, çip alanının artan miktarlarının koordinasyon ve sinyal aktarım mantığı ile işgal edildiği anlamına gelir. Bu da tasarım, üretim ve hata ayıklama zorluklarını artırmaktadır. İşlemci mantığıyla ilgili mühendislik giderek artması, işlemci çipinin giderek artan bir kısmının daha basit bellek mantığına ayrılmasının nedenlerinden biridir. Bir sonraki bölümde ele alınacak olan güç sorunları da bir başka nedendir.

## Güç Tüketimi

Çip başına transistör sayısı arttıkça daha yüksek performans eğilimini sürdürmek için tasarımcılar daha ayrıntılı işlemci tasarımlarına (pipelining, süper skaler, SMT) ve yüksek saat frekanslarına başvurmuşlardır. Ne yazık ki, çip yoğunluğu ve saat frekansı arttıkça güç gereksinimleri de katlanarak artmıştır. Bu durum Şekil 2.2'de gösterilmiştir.

Güç yoğunlığını kontrol etmenin bir yolu, çip alanının daha büyük bir kısmını ön bellek için kullanmaktır. Bellek transistörleri daha küçüktür ve güç yoğunluğu mantık transistörlerinden bir kat daha düşüktür (bkz. Şekil 18.2). Yonga transistör yoğunluğu arttıkça, belleğe ayrılan yonga alanı yüzdesi de artmış ve artık genellikle yonga alanının yarısına ulaşmıştır. Buna rağmen, hala önemli miktarda çip alanı işlem mantığına ayrılmıştır.



**Şekil 18.2** Güç ve Bellekle İlgili Hususlar

Tüm bu mantık transistörlerinin nasıl kullanılacağı önemli bir tasarım sorunudur. Bu bölümde daha önce de tartışıldığı gibi, superscalar ve SMT gibi tekniklerin etkin kullanımının sınırları vardır. Genel anlamda, son on yılın deneyimi **Pollack kuralı** [POLL99] olarak bilinen ve performans artışının kabaca karmaşılıktaki artışın karekökü ile orantılı olduğunu belirten bir kuralda özetlenmiştir. Başka bir deyişle, bir işlemci çekirdeğindeki mantığı iki katına çıkarırsanız, yalnızca %40 daha fazla performans sağlar. Prensiple, çoklu çekirdek kullanımı, çekirdek sayısındaki artısla birlikte doğrusala yakın performans artışı sağlama potansiyeline sahiptir - ancak yalnızca bundan yararlanabilecek yazılımlar için.

Güç hususları, çok çekirdekli bir organizasyona doğru ilerlemek için başka bir neden sağlar. Yonga çok büyük miktarda ön belleğe sahip olduğundan, herhangi bir yürütme iş parçacığının tüm bu belleği etkin bir şekilde kullanması olası değildir. SMT ile bile, çoklu iş parçacığı nispeten sınırlı bir şekilde yapılır ve nedenle devasa bir önbellekten tam olarak yararlanamazken, nispeten bağımsız bir dizi iş parçacığı veya işlem önbellek belleğinden tam olarak yararlanmak için daha büyük bir fırsatı sahiptir.

## 18.2 YAZILIM PERFORMANS SORUNLARI

Çok çekirdekli organizasyonla ilgili yazılım performansı sorunlarının ayrıntılı bir incelemesi kapsamımız dışındadır. Bu bölümde, önce bu konulara genel bir bakış sunacağız ve ardından çok çekirdekli yeteneklerden yararlanmak üzere tasarlanmış bir uygulama örneğine bakacağız.

### Çok Çekirdekli Yazılım

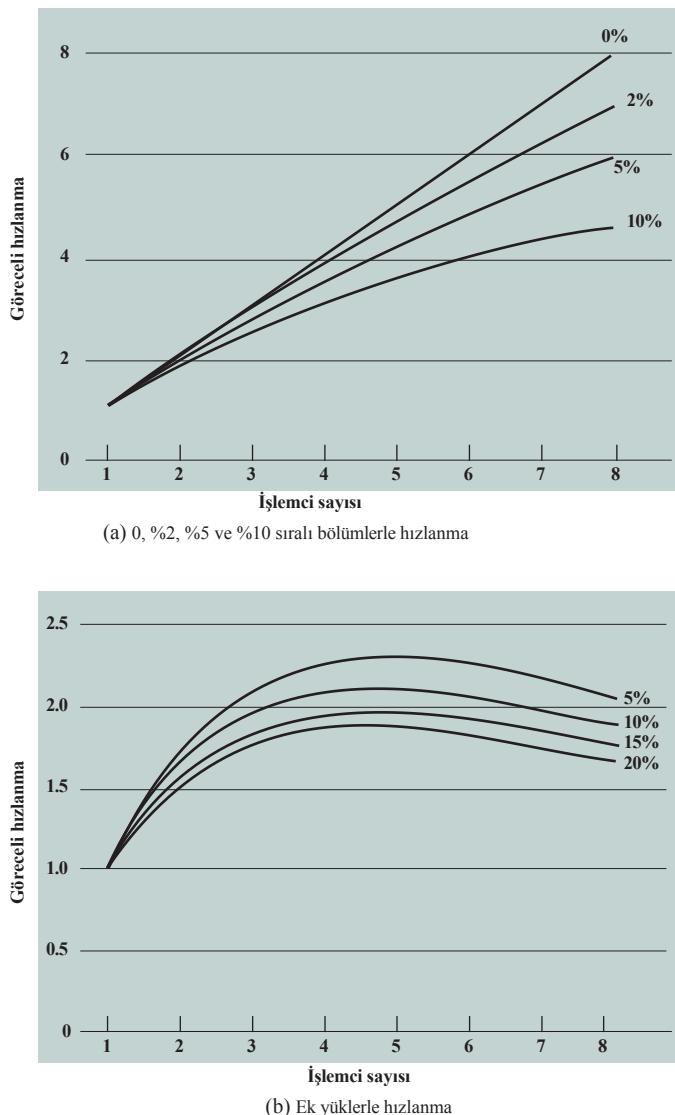
Çok çekirdekli bir organizasyonun potansiyel performans faydaları, uygulama için mevcut olan paralel kaynaklardan etkin bir şekilde faydalama becerisine bağlıdır. İlk olarak çok çekirdekli bir sistem üzerinde çalışan tek bir uygulamaya odaklıyalım. Bölüm 2'den Amdahl yasasının şunu ifade ettiğini hatırlayın:

$$\text{Hızlandırma} = \frac{\text{programı tek bir işlemci üzerinde çalıştırma süresi}}{\text{programı } N \text{ paralel işlemci üzerinde çalıştırma süresi}}$$

$$= \frac{1}{(1 - \frac{1}{f}) +} \quad (18.1)$$

Yasa, yürütme süresinin bir kısmının  $(\frac{1}{f})$  doğası gereği sıralı kodu ve bir kısmının da  $(f)$  zamanlama ek yükü olmadan sonsuza kadar paralelleştirilebilen kodu içerdiği bir programı varsayar.

Bu yasa, çok çekirdekli bir organizasyon olasılığını cazip hale getiriyor gibi görünmektedir. Ancak Şekil 18.3'a'nın gösterdiği gibi, az miktarda seri kodun bile gözle görülebilir bir etkisi vardır. Kodun yalnızca %10'u doğal olarak seri ise ( $f = 0,9$ ), programı sekiz işlemcili çok çekirdekli bir sisteme çalıştırmak yalnızca 4,7 kat performans artışı sağlar. Buna ek olarak, yazılım tipik olarak iletişimini ve işin birden fazla işlemci arasında dağıtılmışının ve önbelleğin bir sonucu olarak ek yüze maruz kalır.



**Sekil 18.3** Çoklu Çekirdeklerin Performans Etkisi

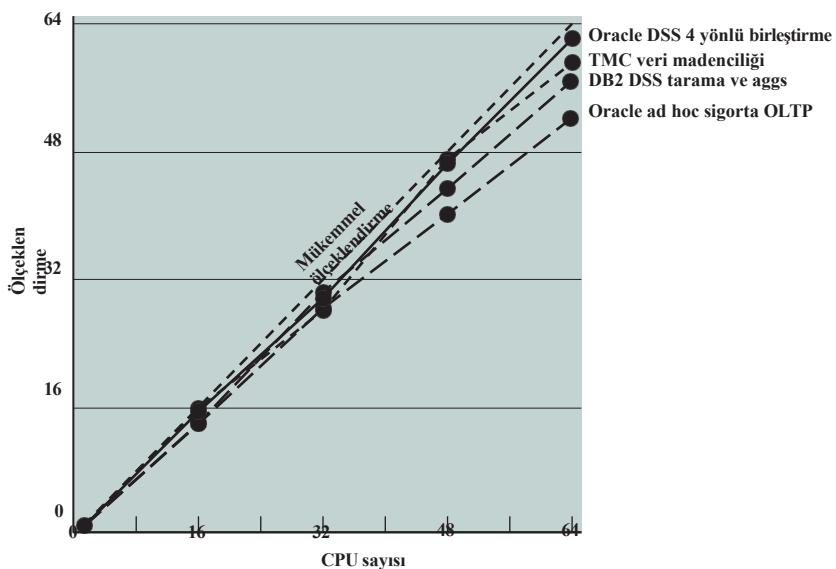
tutarlılık ek yükü. Bu ek yük, birden fazla işlemci kullanmanın getirdiği ek yükün (örn. koordinasyon ve işletim sistemi yönetimi) artması nedeniyle performansın zirve yaptığı ve ardından düşmeye başladığı bir eğriyle sonuçlanır. Şekil 18.3b, [MCDO05]'ten temsili bir örnektir.

Bununla birlikte, yazılım mühendisleri bu sorunu ele almaktadır ve çok çekirdekli bir sistemden etkili bir şekilde yararlanmanın mümkün olduğu çok sayıda uygulama vardır. [MCDO05], çok çekirdekli sistemlerin bir dizi veri tabanı uygulaması üzerindeki etkinliğini analiz etmekte ve bu analizde donanım mimarileri, işletim sistemleri, ara katman yazılımları ve veritabanı içindeki seri kesri azaltmaya büyük önem verilmektedir.

uygulama yazılımı. Şekil 18.4 sonucu göstermektedir. Bu örnekte de görüldüğü gibi, veritabanı yönetim sistemleri ve veritabanı uygulamaları çok çekirdekli sistemlerin etkin bir şekilde kullanılabileceği alanlardan biridir. Birçok sunucu türü de paralel çok çekirdekli organizasyonu etkin bir şekilde kullanabilir, çünkü sunucular genellikle çok sayıda birbirinden bağımsız işlemi paralel olarak gerçekleştirir.

Genel amaçlı sunucu yazılımına ek olarak, bir dizi uygulama sınıfı, verimi çekirdek sayısıyla ölçeklendirme becerisinden doğrudan yararlanır. [MCDO06] aşağıdaki örnekleri listelemektedir:

- **Çok iş parçacıklı yerel uygulamalar (iş parçacığı düzeyinde paralellik):** Çok iş parçacıklı uygulamalar, az sayıda yüksek iş parçacıklı işleme sahip olmaları ile karakterize edilir.
- **Çok işlemli uygulamalar (işlem düzeyinde paralellik):** Çok işlemli uygulamalar, çok sayıda tek iş parçacıklı işlemin varlığı ile karakterize edilir.
- **Java uygulamaları:** Java uygulamaları iş parçacığını temel bir şekilde benimser. Java dili çok iş parçacıklı uygulamaları büyük ölçüde kolaylaştırmakla kalmaz, aynı zamanda Java Sanal Makinesi de Java uygulamaları için zamanlama ve bellek yönetimi sağlayan çok iş parçacıklı bir süreçtir.
- **Çok örnekli uygulamalar (uygulama düzeyinde paralellik):** Bireysel bir uygulama çok sayıda iş parçacığından faydalananak şekilde ölçeklenmese bile, uygulamanın birden fazla örneğini paralel olarak çalıştırarak çok çekirdekli mimariden faydalananmak mümkündür. Birden fazla uygulama örneği bir dereceye kadar izolasyon gerektiriyorsa, her birine kendi ayrı ve güvenli alanını sağlamak için sanallaştırma teknolojisi (işletim sisteminin donanımı için) kullanılabilir.



**Şekil 18.4** Çoklu İşlemci Donanımında Veritabanı İş Yüklerinin Ölçeklendirilmesi

Bir örneğe geçmeden önce, faydalı bir şekilde paralelleştirilecek minimum iş birimi olarak tanımlanabilecek iş parçacığı **tanecikliliği** kavramını tanitarak iş parçacığı düzeyinde paralellik konusunu detaylandıracagız. Genel olarak, sistem ne kadar ince tanecikliliğe olanak tanırsa, programcı bir programı paralelleştirme konusunda o kadar az kısıtlıdır. Sonuç olarak, daha ince taneli iş parçacığı sistemleri, kaba taneli olamlara göre daha fazla durumda paralelleştirmeye izin verir. Bir mimarinin hedef tanecikliliğinin seçimi doğal bir değişim tokusu içerir. Bir yandan, daha ince taneli sistemler programcıya sağladıkları esneklik nedeniyle tercih edilir. Öte yandan, iş parçacığı taneciği ne kadar ince olursa, yürütmenin daha önemli bir kısmı iş parçacığı sistemi ek yükü tarafından alınr.

### **Uygulama Örneği: Valve Oyun Yazılımı**

Valve, bir dizi popüler oyunun yanı sıra mevcut en yaygın oyun motorlarından biri olan Source motorunu geliştiren bir eğlence ve teknoloji şirketidir. Source, Valve tarafından oyunları için kullanılan ve diğer oyun geliştiricilerine lisanslanan bir animasyon motorudur.

Valve, Intel ve AMD'nin çok çekirdekli işlemci yongalarının ölçeklenebilirliğinden yararlanmak için Source motor yazılımını çoklu iş parçacığı kullanacak şekilde yeniden programlamıştır [REIM06]. Revize edilmiş Source motor kodu Half Life 2 gibi Valve oyunları için daha güçlü bir destek sağlamaktadır.

Valve'in bakış açısından, iş parçacığı ayrıntı seçenekleri aşağıdaki gibi tanımlanır [HARR06]:

- **Kaba taneli iş parçacığı:** Sistem adı verilen ayrı modüller ayrı işlemcilere atanır. Kaynak motoru örneğinde bu, render işlemini bir işlemciye, AI'yi (yapay zeka) başka bir işlemciye, fiziki başka bir işlemciye vb. koymak anlamına gelir. Bu basit bir işlemmdir. Özünde, her ana modül tek iş parçacıklıdır ve temel koordinasyon, tüm parçacıklarının bir zaman çizelgesi iş parçacığı ile senkronize edilmesini içerir.
- **İnce taneli iş parçacığı:** Birçok benzer veya aynı görev birden fazla işlemciye . Örneğin, bir veri dizisi üzerinde yinelenen bir döngü, paralel olarak zamanlanabilen aynı iş parçacıklarında bir dizi daha küçük paralel döngüye bölünebilir.
- **Hibrit dış açma:** Bu, bazı sistemler için ince taneli iş parçacığının ve diğer sistemler için tek iş parçacığının seçici olarak kullanılmasını içerir.

Valve, kaba iş parçacıkları sayesinde tek bir işlemcide yürütmeye kıyasla iki işlemcide iki kata kadar performans elde edebileceğini bulmuştur. Ancak bu performans kazancı sadece uydurma durumlarda elde edilebildi. Gerçek dünya oyunlarında bu iyileşme 1,2 kat düzeyindeydi. Valve ayrıca ince taneli iş parçacıklarının etkin bir şekilde kullanılmasının zor olduğunu tespit etmiştir.

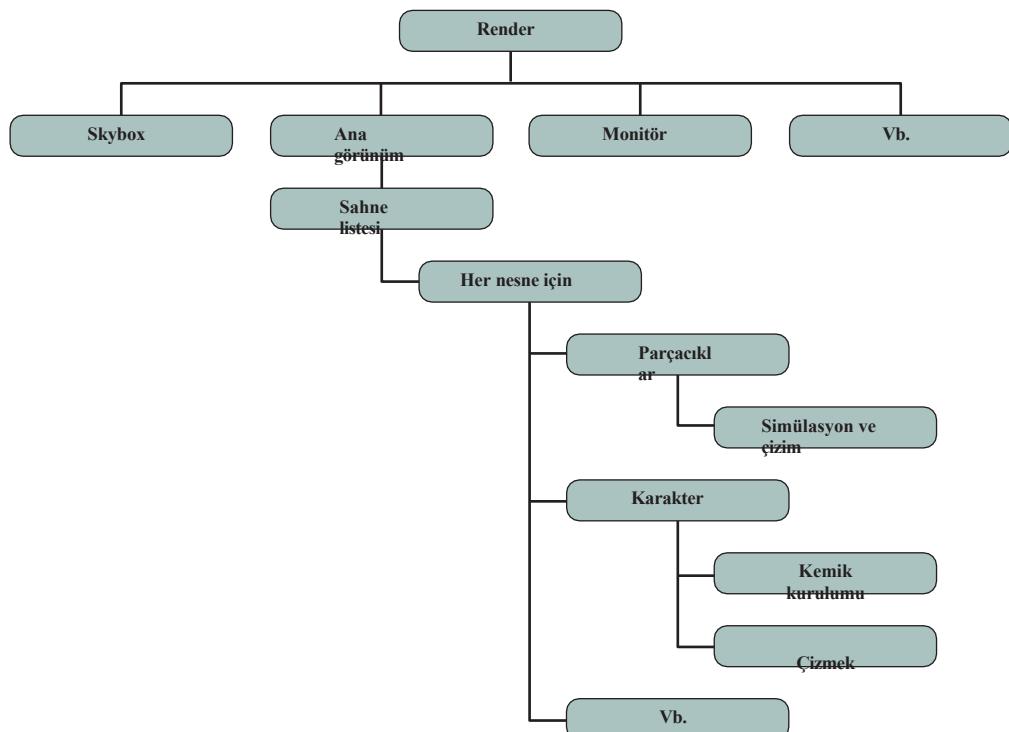
İş birimi başına düşen süre değişken olabilir ve sonuçların ve sonuçların zaman çizelgesini yönetmek karmaşık programlama gerektirir. Valve, hibrit iş parçacığı yaklaşımının en umut verici yaklaşım olduğunu ve sekiz ya da on altı işlemcili çok çekirdekli sistemler kullanılabilir hale geldikçe en iyi şekilde ölçekleneceğini tespit etmiştir. Valve, sürekli olarak tek bir işlemciye atandığında çok etkili bir şekilde çalışan sistemleri tanımlamıştır. Örnek olarak, kullanıcı etkileşiminin az olduğu, pencerelerin çerçeveye yapılandırmasıyla kısıtlanmaya

kendi veri kümesi. Sahne oluşturma gibi diğer modüller, modülün tek bir işlemci üzerinde çalışabilmesi ancak daha fazla işlemciye yayıldıkça daha yüksek performans elde edebilmesi için bir dizi iş parçacığı halinde düzenlenebilir.

Şekil 18.5, işleme modülü için iş parçacığı yapısını göstermektedir. Bu hiyerarşik yapıda, üst düzey iş parçacıkları gereği içinde alt düzey iş parçacıklarını doğurur. Render modülü, Source motorunun kritik bir parçası olan ve oyun dünyasındaki görsel öğelerin bir veritabanı temsili olan dünya listesine dayanır. İlk görev, dünyyanın işlenmesi gereken alanlarının neler olduğunu belirlemektir. Bir sonraki görev, birden fazla açıdan bakıldığından sahne de hangi nesnelerin olduğunu belirlemektir. Daha sonra işlemci yoğun iş gelir. Render modülü, oyuncunun görüşü, TV monitörlerinin görüşü ve sudaki yansımaların bakış açısı gibi birden fazla bakış açısından her bir nesnenin render edilmesi üzerinde çalışmak zorundadır.

İşleme modülü için iş parçacığı stratejisinin bazı temel unsurları [LEON07]'de listelenmiştir ve aşağıdakileri içerir:

- Paralel olarak birden fazla sahne için sahne oluşturma listeleri oluşturun (örneğin, dünya ve sudaki yansımıası).
- Örtüsen grafik simülasyonu.
- Tüm sahnelerdeki tüm karakterler için karakter kemik dönüşümlerini paralel olarak hesaplayın.
- Birden fazla iş parçacığının paralel olarak çizim yapmasına izin verin.



**Şekil 18.5** Rendering Modülü için Hibrit Threading

Tasarımcılar, dünya listesi gibi önemli veritabanlarını bir iş parçacığı için kilitlemenin çok verimsiz olduğunu keşfettiler. Bir iş parçacığı zamanın %95'inden fazlasını bir veri kümesinden okumaya çalışırken, zamanın en fazla %5'i bir veri kümesine yazmak için harcanır. Bu nedenle, tek-yazıcı-çoklu-okuyucu modeli olarak bilinen bir eşzamanlılık mekanizması etkili bir şekilde çalışır.

## 18.3 ÇOK ÇEKİRDEKLİ ORGANİZASYON

En üst düzeyde bir tanımla, çok çekirdekli bir organizasyondaki ana değişkenler aşağıdaki gibidir:

- Çip üzerindeki çekirdek işlemci sayısı
- Ön bellek seviyelerinin sayısı
- Ön belleğin çekirdekler arasında nasıl paylaşıldığı
- Eşzamanlı çoklu iş parçacığının (SMT) kullanılıp kullanılmadığı
- Çekirdek türleri

Bu bölümde bu hususların sonucusu hariç hepsini inceleyeceğiz ve çekirdek türlerine ilişkin tartışmayı bir sonraki bölüme erteleyeceğiz.

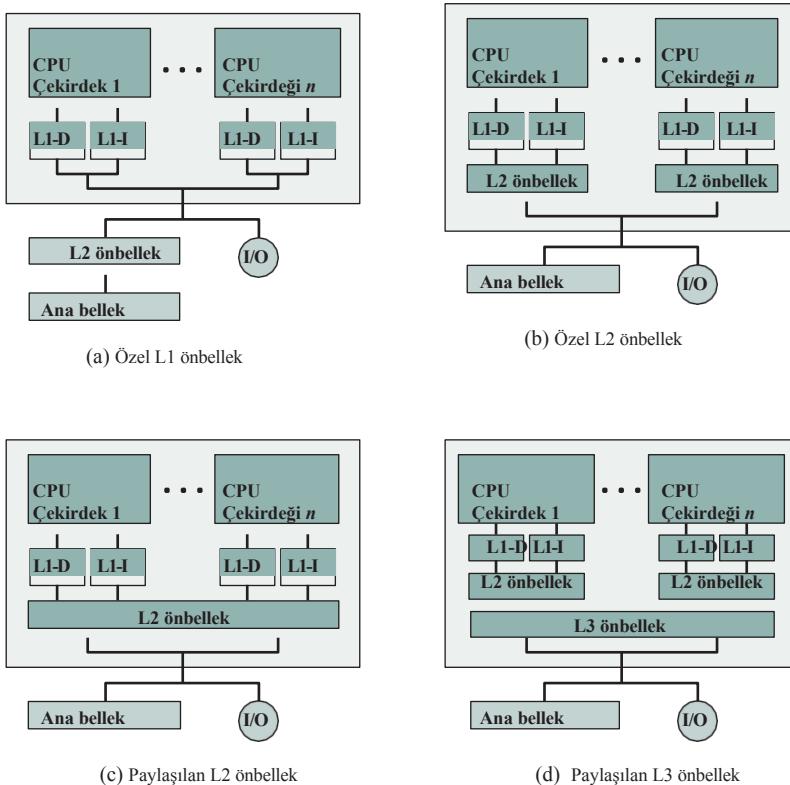
### Ön bellek Seviyeleri

Şekil 18.6'da çok çekirdekli sistemler için dört genel organizasyon gösterilmektedir. Şekil 18.6a, daha önceki çok çekirdekli bilgisayar yongalarının bazlarında bulunan ve hala bazı gömülü yongalarda görülen bir organizasyondur. Bu organizasyonda, çip üzerindeki tek ön bellek L1 önbellegidir ve her çekirdek kendi özel L1 önbellegine sahiptir. Neredeyse her zaman, L1 önbeliği performans nedenleriyle komut ve veri ön belleklere bölünürken, L2 ve daha yüksek seviyeli ön belleklere bireştirilir. Bu organizasyona bir örnek ARM11 MPCore'dur.

Şekil 18.6b'deki organizasyon da çip üzerinde ön bellek paylaşımının olmadığı bir organizasyondur. Bu durumda, L2 önbellege izin vermek için çip üzerinde yeterli alan mevcuttur. Bu organizasyonun bir örneği AMD Opteron'dur. Şekil 18.6c'de benzer bir çip alanının belleğe tahsis, ancak paylaşılan bir L2 kullanımı gösterilmektedir. Intel Core Duo bu organizasyona sahiptir. Son olarak, çip üzerinde mevcut olan ön bellek miktarı artmaya devam ettikçe, performans hususları her bir çekirdek işlemci için özel L1 ve L2 ön bellekleri ile ayrı, paylaşılan bir L3 önbeleginin (Şekil 18.6d) ayrılmasını gerektirmektedir. Intel Core i7 bu organizasyona bir örnektir.

Çip üzerinde paylaşılan bir üst düzey ön bellek kullanımı, özel ön belleklere özel olarak güvenmeye göre çeşitli avantajlara sahiptir:

1. Yapıcı girişim genel ıskalama oranlarını azaltabilir. , bir bir iş parçacığı bir ana bellek konumuna erişirse, bu, başvurulan konumu içeren satırı paylaşılan önbellege getirir. Kısa süre sonra başka bir çekirdekteki bir iş parçacığı aynı bellek bloğuna erişirse, bellek konumları zaten paylaşılan yonga üstü ön bellekte mevcut olacaktır.
2. Bununla ilgili bir avantaj, birden fazla çekirdek tarafından paylaşılan verilerin paylaşılan ön bellek düzeyinde çoğaltılmamasıdır.

**Şekil 18.6** Çok Çekirdekli Organizasyon Alternatifleri

3. Uygun satır değiştirme algoritmalarıyla, her bir çekirdeğe tahsis edilen paylaşılan önbellek miktarı dinamiktir, böylece daha az yerelliğe sahip iş parçacıkları (daha büyük çalışma kümeleri) daha fazla önbellek kullanabilir.
4. Çekirdekler arası iletişimini, paylaşılan bellek konumları aracılığıyla uygulamak kolaydır.
5. Paylaşılan bir üst düzey önbelleğin kullanılması, önbellek tutarlılığı sorununu daha düşük önbellek düzeyleriyle sınırlandırır ve bu da bazı ek performans avantajları sağlayabilir.

Çip üzerinde yalnızca özel L2 önbelleklere sahip olmanın potansiyel bir avantajı, her çekirdeğin kendi özel L2 önbelleğine daha hızlı erişebilmesidir. Bu, güçlü yerellik sergileyen iş parçacıkları için avantajlıdır.

Hem mevcut bellek miktarı hem de çekirdek sayısı arttıkça, paylaşılan bir L3 önbelleğinin özel çekirdek başına L2 önbellekleriyle birlikte kullanılması, büyük bir paylaşılan L2 önbelleğinden veya çip üzerinde L3 olmadan çok büyük özel L2 önbelleklerinden daha iyi performans sağlayacak görünmektedir. Bu ikinci düzenlemeye örnek Xeon E5-2600/4600 çip işlemcisi verilebilir (Şekil 7.1)

L1'lerin her çekirdek için yerel olduğu, L2'lerin 2 ila 4 çekirdek arasında paylaşıldığı ve L3'ün tüm global olduğu düzenleme gösterilmemiştir. Bu düzenlemenin zaman içinde daha yaygın hale gelmesi muhtemeldir.

## Eşzamanlı Multithreading

Çok çekirdekli bir sistemdeki bir diğer organizasyonel tasarım kararı, bağımsız çekirdeklerin **eşzamanlı çoklu iş parçacığı (SMT)** uygulayıp uygulamayacağıdır. Örneğin, Intel Core Duo saf superskalar çekirdekler kullanırken, Intel Core i7 SMT çekirdekleri kullanır. SMT, çok çekirdekli sistemin desteklediği donanım düzeyindeki iş parçacığı sayısını ölçeklendirme etkisine sahiptir. Dolayısıyla, dört çekirdekli çok çekirdekli bir sistem ve her çekirdekte dört eşzamanlı iş parçacığını destekleyen SMT, uygulama düzeyinde 16 çok çekirdekli bir sistemle aynı görünür. Paralel kaynaklardan daha fazla yararlanmak için yazılım geliştirildikçe, SMT yaklaşımı tamamen süper skaler bir yaklaşımından daha çekici görünümlere sahip olmuştur.

## 18.4 HETEROJEN ÇOK ÇEKİRDEKLİ ORGANİZASYON

Bir işlemci yongası üzerindeki silikon alanını en iyi şekilde kullanma arayışı hiç bitmiyor. Saat hızları ve mantık arttıkça, tasarımcılar performansı en üst düzeye çıkarmak ve güç tüketimini en aza indirmek için birçok tasarım unsuruunu dengelemek zorundadır. Şimdiye kadar aşağıdakiler de dahil olmak üzere bu türden bir dizi yaklaşımı inceledik:

1. Ön belleğe ayrılan çip yüzdesini artırın.
2. Ön bellek seviyelerinin sayısını artırın.
3. Talimat ardisık düzeninin uzunluğunu (artırma veya azaltma) ve işlevsel bileşenlerini değiştirin.
4. Eşzamanlı çoklu iş parçacığı kullanın.
5. Birden fazla çekirdek kullanın.

Çoklu çekirdek kullanımı için tipik bir durum, **homojen çoklu çekirdek organizasyonu** olarak bilinen çoklu özdeş çekirdekler sahip bir çaptır. Performans ve/veya güç tüketimi açısından daha iyi sonuçlar elde etmek için, giderek daha popüler hale gelen bir tasarım seçeneği, birden fazla türde çekirdek içeren bir işlemci yongasını ifade eden **heterojen çok çekirdekli organizasyondur**. Bu bölümde, heterojen çok çekirdekli organizasyona yönelik iki yaklaşımı inceleyeceğiz.

### Farklı Komut Seti Mimarileri

Sektörde en çok ilgi gören yaklaşım, farklı ISA'lara sahip çekirdeklerin kullanılmasıdır. Tipik olarak bu, bu bağlamda CPU olarak adlandırılan geleneksel çekirdeklerin belirli veri veya uygulama türleri için optimize edilmiş özel çekirdeklerle karıştırılmasını içerir. Coğu zaman, ek çekirdekler vektör ve matris veri işleme ile başa çıkmak için optimize edilir.

**cPU/GPU ÇOK ÇEKİRDEKLİ** Heterojen çok çekirdekli tasarım açısından en belirgin eğilim, aynı yonga üzerinde hem CPU'ların hem de grafik işlem birimlerinin (GPU'lar) kullanılmasıdır. GPU'lar bir sonraki bölümde ayrıntılı olarak ele alınmaktadır. Kisaca, GPU'lar binlerce paralel yürütme iş parçacığını destekleyebilme özelliği ile karakterize edilir. Bu nedenle, GPU'lar büyük miktarlarda işlem yapan uygulamalar için çok uygunlardır.

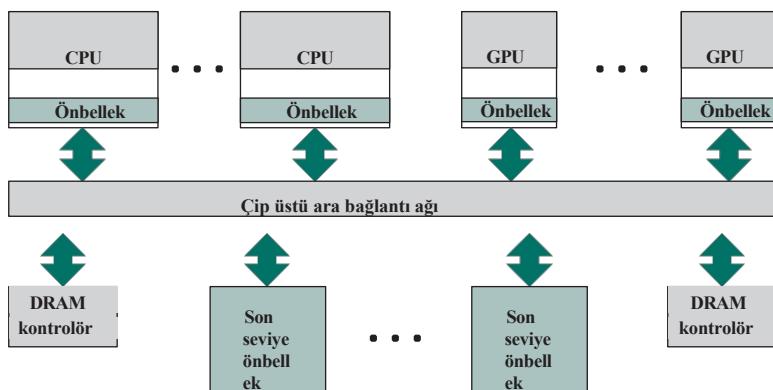
vektör ve matris verileri. Başlangıçta grafik uygulamalarının performansını artırmayı amaçlayan bu yeni işlemciler, CUDA (Compute Unified Device Architecture) gibi benimsenmesi kolay programlama modelleri sayesinde, yapılandırılmış veriler üzerinde çok sayıda tekrarlayan işlem içeren genel amaçlı ve bilimsel uygulamaların performansını artırmak için giderek daha fazla uygulanmaktadır. Günümüzün bilgi işlem ortamındaki hedef uygulamaların çeşitliliğiyle başa çıkmak için, hem GPU'lari hem de CPU'lari içeren çoklu çekirdek, performansı artırma potansiyeline sahiptir. Ancak bu heterojen karışım, koordinasyon ve iş birliği sorunlarını da beraberinde getirmektedir. doğruluk.

Şekil 18.7 tipik bir çok çekirdekli işlemci organizasyonudur. Birden fazla CPU ve GPU, son seviye önbellek (LLC), ara bağlantı ağı ve bellek denetleyicileri gibi çip üzerindeki kaynakları paylaşır. En kritik olanı, önbellek yönetim politikalarının LLC'nin etkili bir şekilde paylaşılmasını sağlama şeklidir. CPU'lar ve GPU'lar arasındaki önbellek duyarlılığı ve bellek erişim hızı farklılıklarını, LLC'nin verimli paylaşımı için önemli zorluklar yaratmaktadır.

Tablo 18.1, bilimsel uygulamalar için CPU ve GPU'lari birleştirmenin potansiyel performans avantajını göstermektedir. Bu tablo bir AMD çipi olan A10 5800K'nin temel çalışma parametrelerini göstermektedir [ALTS12]. Kayan nokta hesaplamaları için CPU'nun 121,6 GFLOPS'luk performansı, kaynağı etkin bir şekilde kullanabilen uygulamalara 614 GFLOPS sunan GPU tarafından gölgede bırakılmaktadır.

İster bilimsel uygulamalar ister geleneksel grafik işleme olsun, eklenen GPU işlemcilerinden yararlanmanın anahtarı, bir veri bloğunu ya aktarmak, işlemek ve ardından sonuçları ana uygulama iş parçacığına döndürmek için gereken süreyi dikkate almaktır. GPU'lari içeren çiplerin önceki uygulamalarında, fiziksel bellek CPU ve GPU arasında bölünmüştür. CPU üzerinde GPU işlemesi gerektiren bir uygulama iş parçacığı çalışıyorsa, CPU verileri açıkça belleğe kopyalar. GPU hesaplamayı tamamlar ve ardından sonucu CPU geri kopyalar. Bellek böülümlere ayrıldığı için CPU ve GPU bellek önbellekleri arasında önbellek tutarlılığı sorunları ortaya çıkmaz. Öte yandan, verilerin fiziksel olarak ileri geri taşınması bir performans kaybına neden olur.

Bir önceki paragrafta açıklanan performansın üzerine çıkmak için bir dizi araştırma ve geliştirme çalışması yürütülmektedir ve bunlardan en önemlileri şunlardır



**Şekil 18.7** Heterojen Çok Çekirdekli Çip Elemanları

**Tablo 18.1** AMD 5100K Heterojen Çok Çekirdekli İşlemcinin İşletim Parametreleri

	CPU	GPU
Saat frekansı (GHz)	3.8	0.8
Cekirdeklər	4	384
FLOPS/çekirdek	8	2
GFLOPS	121.6	614.4

FLOPS= saniye başına kayan nokta işlemleri.

FLOPS/core= gerçekleştirilebilen paralel kayan nokta işlemlerinin sayısı.

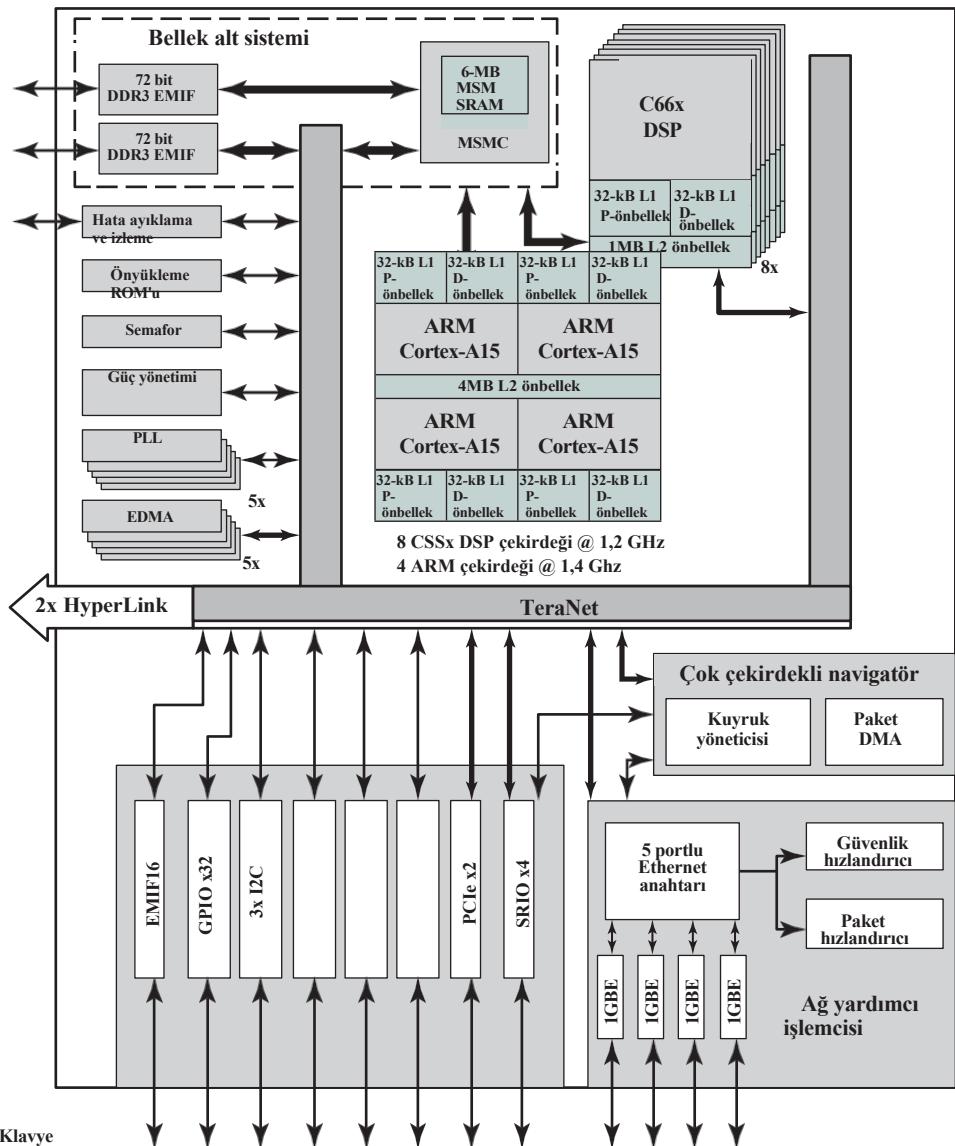
Heterojen Sistem Mimarisi (HSA) Vakfı'nın girişimidir. HSA yaklaşımının temel özellikleri arasında aşağıdakiler yer almaktadır:

1. Tüm sanal bellek alanı hem CPU hem de GPU tarafından görülebilir. Hem CPU hem de GPU, sistemin sanal bellek alanındaki herhangi bir konuma erişebilir ve bu konumu tahsis edebilir.
2. Sanal bellek sistemi gerektiğinde sayfaları fiziksel ana belleğe getirir.
3. Tutarlı bir bellek politikası, CPU ve GPU önbelleklerinin her ikisinin de verilerin güncel bir görünümünü görmesini sağlar.
4. Kullanıcıların CPU yürütmesine de dayanan programlarda GPU'ların paralel yeteneklerinden yararlanmasını sağlayan birleşik bir programlama arayüzü.

Genel amaç, programcıların CPU'ların seri gücünden ve GPU'ların paralel işleme gücünden, işletim sistemi ve donanım düzeyinde verimli bir koordinasyonla daha az yararlanarak uygulamalar yazmasına olanak sağlamaktır. Belirtildiği gibi, bu devam eden bir araştırma ve geliştirme alanıdır.

**cPU/DSP MULTicORE** Heterojen çok çekirdekli ciplerin bir diğer yaygın örneği CPU'lar ve dijital sinyal işlemcilerinin (DSP'ler) bir karışımıdır. Bir DSP, matematik yoğun dijital sinyal işleme uygulamalarında yaygın olarak kullanılan ultra hızlı komut dizileri (kayırdırma ve toplama; çarpmaya ve toplama) sağlar. DSP'ler ses, hava durumu uyduları ve deprem monitörleri gibi kaynaklardan gelen analog verileri işlemek için kullanılır. Sinyaller dijital verilere dönüştürülür ve Hızlı Fourier Dönüşümü gibi çeşitli algoritmalar kullanılarak analiz edilir. DSP çekirdekləri cep telefonları, ses kartları, faks makineleri, modemler, sabit diskler ve dijital TV'ler dahil olmak üzere sayısız cihazda yaygın olarak kullanılmaktadır.

İyi bir temsili örnek olarak Şekil 18.8'de Texas Instruments (TI) K2H SoC platformunun [TI12] yeni bir sürümü gösterilmektedir. Bu heterojen çok çekirdekli işlemci, üst düzey görüntüleme uygulamaları için güç tasarruflu işleme çözümleri sunmaktadır. TI, performansi 352 GMACS, 198 GFLOPS ve 19.600 MIPS'e kadar olarak listelemektedir. GMACS, DSP performansının yaygın bir ölçüsü olan saniyede giga (milyarlarca) çarpmaya-biriktirme işlemi anlamına geliyor. Bu sistemler için hedef uygulamalar arasında endüstriyel otomasyon, video gözetimi, üst düzey denetim sistemleri, endüstriyel yazılımlar/tarayıcılar ve para birimi/sahte tespiti yer almaktadır.



**Şekil 18.8** Texas Instruments 66AK2H12 Heterojen Çok Çekirdekli Çip

TI çipi dört ARM Cortex-A15 çekirdeği ve sekiz TI C66x DSP çekirdeği içerir.

Her DSP çekirdeği 32 kB L1 veri önbelleği ve 32 kB L1 program (komut) önbelleği içerir. Buna ek olarak, her DSP'de tüm L2, tüm ana bellek veya ikisinin karışımı olarak yapılandırılabilen 1 MB özel SRAM bellek bulunur. Ana bellek olarak yapılandırılan kısım, basitçe SRAM olarak adlandırılan "yerel" bir ana bellek olarak işlev görür. Bu yerel ana bellek geçici veriler için kullanılabilir ve önbellek ile çip dışı bellek arasındaki trafik ihtiyacını ortadan kaldırır. Her birinin L2 önbelleği

sekiz DSP çekirdeği diğer DSP ile paylaşılmak yerine ayrılmıştır. Bu, çok çekirdekli bir DSP organizasyonu için tipiktir: Her DSP paralel olarak ayrı bir veri bloğu üzerinde çalışır, bu nedenle veri paylaşımı çok az ihtiyaç vardır.

Her ARM Cortex-A15 CPU çekirdeği 32 kB L1 veri ve program önbelleklere sahiptir ve dört çekirdekte 4 MB L2 önbelleği paylaşır.

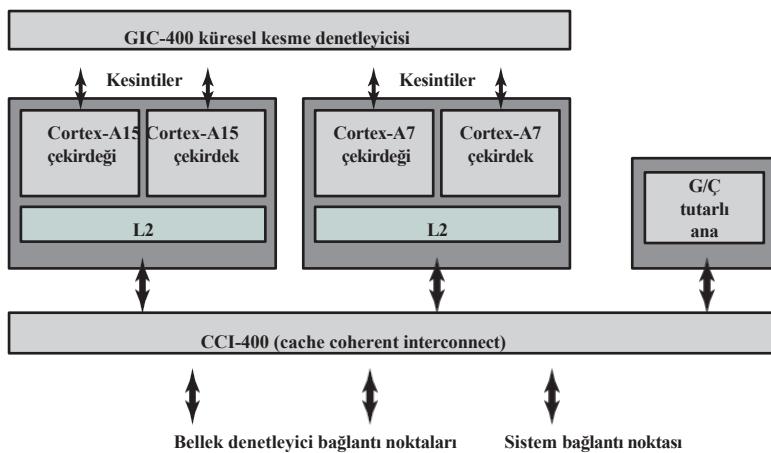
6 MB çok çekirdekli paylaşılan bellek (MSM) her zaman tüm SRAM olarak yapılandırılır, önbellek yerine ana bellek gibi davranışır. Doğrudan L1 DSP ve CPU önbelleklere besleyecek şekilde ya da L2 DSP ve CPU önbelleklere besleyecek şekilde yapılandırılabilir. Bu yapılandırma kararı beklenen uygulama profiline bağlıdır. Çok çekirdekli paylaşılan bellek denetleyicisi (MSMC), ARM çekirdekleri, DSP, DMA, diğer ana çevre birimleri ve harici bellek arabirimini (EMIF) arasındaki trafiği yönetir. MSMC, cihazdaki tüm çekirdekler ve ana çevre birimleri tarafından erişilebilen MSM'ye erişimi kontrol eder.

### Eşdeğer Komut Seti Mimarileri

Heterojen çok çekirdekli organizasyona yönelik bir diğer yeni yaklaşım, eşdeğer ISA'lara sahip ancak performans veya güç verimliliği açısından farklılık gösteren birden fazla çekirdeğin kullanılmasıdır. Bunun önde gelen örneği, bu bölümde ARM'in big.Little mimarisidir.

Şekil 18.9 bu mimariyi göstermektedir. Şekilde iki adet yüksek performanslı Cortex-A15 çekirdeği ve iki adet daha düşük performanslı, daha düşük güç tüketen Cortex-A7 çekirdeği içeren çok çekirdekli bir prosesör yongası gösterilmektedir. A7 çekirdekleri arka plan işleme, müzik çalma, mesaj gönderme ve telefon görüşmesi yapma gibi daha az hesaplama gerektiren görevleri yerine getirir. A15 çekirdekleri ise video, oyun ve navigasyon gibi yüksek yoğunluklu görevler için kullanılıyor.

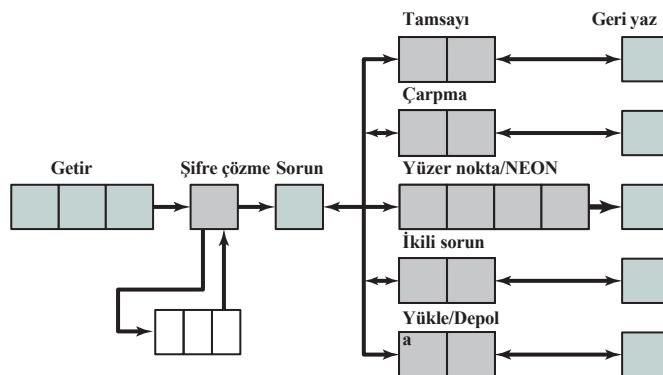
Big.Little mimarisi akıllı telefon ve tablet pazarına yöneliktedir. Bu cihazlar, kullanıcılarından gelen performans talepleri baryaların kapasitesinden ya da yarı iletken süreçlerindeki ilerlemelerin sağladığı güç tasarrufundan çok daha hızlı artan cihazlardır. Akıllı telefon ve tabletlerin kullanım şekli oldukça dinamiktir. Oyun oynama ve web'de gezinme gibi yoğun işlem gerektiren görevler, dönüşümlü olarak



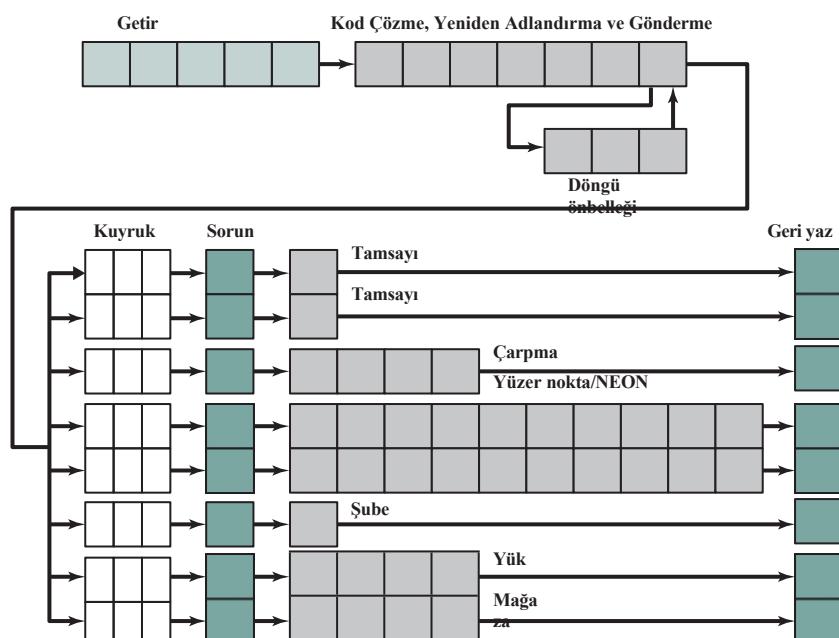
**Sekil 18.9** big.Little Çip Bileşenleri

mesajlaşma, e-posta ve ses gibi düşük işlem yoğunluğuna sahip görevlerde tipik olarak daha uzun süreler. Big.Little mimarisi, gerekli performanstanın bu çeşitlilikten yararlanır. A15, mobil güç bütçesi dahilinde maksimum performans için tasarlanmıştır. A7 işlemci, maksimum verimlilik ve en yoğun çalışma süreleri hariç tüm işlerin üstesinden gelebilecek kadar yüksek performans için tasarlanmıştır.

**A7 VE A15 KARAKTERİSTİKLERİ** A7, A15'ten çok daha basit ve daha az güçlidür. Ancak basitliği, A15'in karmaşıklığından çok daha az transistör gerektirir ve daha az transistörün çalışması için daha az enerji gereklidir. A7 ve A15 çekirdekleri arasındaki farklar en açık şekilde Şekil 18.10'da gösterildiği gibi komut işlem hatları incelenerek görülebilir.



(a) Cortex A-7 Boru Hattı



(b) Cortex A-15 Boru Hattı

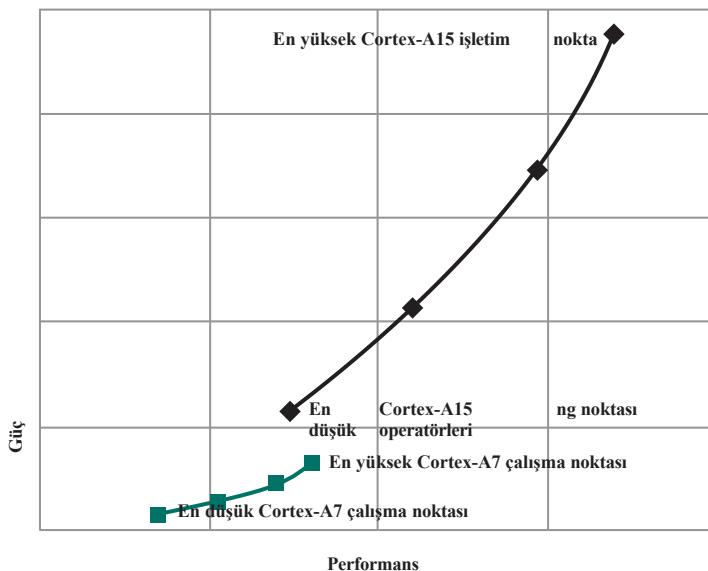
**Şekil 18.10** Cortex A-7 ve A-15 Boru Hatları

A7, 8 ile 10 aşamalı bir boru hattı uzunluğuna sahip sıralı bir CPU'dur. Tüm yürütme birimleri için tek bir kuyruğu vardır ve saat döngüsü başına beş yürütme birimine iki talimat gönderebilir. Öte yandan A15, 15 ile 24 aşamalı boru hattı uzunluğuna sahip sıra dışı bir işlemcidir. Sekiz yürütme biriminin her biri kendi çok aşamalı kuyruğuna sahiptir ve saat döngüsü başına üç talimat işleyebilir.

Bir komutun yürütülmesiyle tüketilen enerji, kısmen geçmesi gereken boru hattı aşamalarının sayısıyla ilgilidir. Bu nedenle, Cortex-A15 ve Cortex-A7 arasındaki enerji tüketimindeki önemli bir fark, farklı boru hattı karmaşıklığından kaynaklanmaktadır. Bir dizi kıyaslamada Cortex-A15, birim MHz başına Cortex-A7'nin kabaca iki katı performans sunar ve Cortex-A7 aynı iş yüklerini tamamlamada Cortex-A15'ten kabaca üç kat daha fazla enerji verimlidir [JEFF12]. Performans ödünləşimi Şekil 18.11'de gösterilmiştir [STEV13].

**YAZILIM İŞLEME MODELLERİ** big.Little mimarisi iki yazılım işleme modelinden birini kullanacak şekilde yapılandırılabilir: geçiş ve çoklu işlem (MP). Yazılım modelleri, temel olarak bir iş yükünün çalışma zamanı yürütülmesi sırasında işi büyük veya Küçük çekirdeklerde tahsis etme şekillerine göre farklılık gösterir.

Geçiş modelinde, büyük ve küçük çekirdekler eşleştirilir. İşletim sistemi çekirdeği zamanlayıcısı için her bir big/Little çifti tek bir çekirdek olarak görünür. Güç yönetimi yazılımı, yazılım bağamlarının iki çekirdek arasında taşınmasından sorumludur. Bu model, işletim sisteminin platformun performansını uygulamanın gerektirdiği performansla eşleştirmesine izin vermek için mevcut mobil platformlar tarafından sağlanan dinamik voltaj ve frekans ölçeklendirme (DVFS) çalışma noktalarının doğal bir uzantısıdır. Günümüzün akıllı telefon SoC'lerinde, `cpu_freq` gibi DVFS sürücülerinin işletim sistemi performansını düzenli ve sık aralıklarla örneklemekte ve DVFS yöneticiyi daha yüksek veya daha düşük bir çalışma noktasına geçip geçmeyeceğine veya mevcut çalışma noktasında kalıp kalmayacağına karar vermektedir. Şekil 18.11'de gösterildiği gibi, hem A7 hem de A15 dört farklı işletim noktasında çalışabilir



Şekil 18.11 Cortex-A7 ve A15 Performans Karşılaştırması

noktaları. DVFS yazılımı, belirli bir CPU saat frekansı ve voltaj seviyesi ayarlayarak eğri üzerindeki çalışma noktalarından birini etkin bir şekilde arayabilir.

Bu çalışma noktaları tek bir CPU kümесinin voltaj ve frekansını etkiler; ancak bir big.Little sisteminde bağımsız voltaj ve frekans alanlarına sahip iki CPU kümesi vardır. Bu, büyük kümenin Little işlemci kümesi tarafından sağlanan DVFS çalışma noktalarının mantıksal bir uzantısı olarak hareket etmesini sağlar. Geçiş kontrol modu altındaki bir big.Little sisteminde, Cortex-A7 çalışırken, DVFS sürücüsü CPU kümесinin performansını daha yüksek seviyelere ayarlayabilir. Cortex-A7 en yüksek çalışma noktasına ulaştığında, daha fazla performans, işletim sistemini ve uygulamaları alıp Cortex-A15'e taşıyan bir görev geçisi çağrılabilir. Günümüzün akıllı telefon SoC'lerinde, cpu\_freq gibi DVFS sürücülerini işletim sistemi performansını düzenli ve sık aralıklarla örnekleme ve DVFS yöneticisi daha yüksek veya daha düşük bir çalışma noktasına geçmeye veya mevcut çalışma noktasında karar vermektedir.

Geçiş modeli basittir ancak her çiftteki CPU'lardan birinin her zaman boşta olmasını gerektirir. MP modeli, A15 ve A7 çekirdeklerinin herhangi bir karışımının aynı anda çalıştırılmasına ve yürütülmeye izin verir. Büyük bir işlemcinin çalıştırılması gerekip gerekmemişti, o anda yürütülen görevlerin performans gereksinimlerine göre belirlenir. Zorlu görevler varsa, bunları严格执行 için büyük bir işlemci açılabilir. Düşük talepli görevler küçük bir işlemcide yürütülebilir. Son olarak, kullanılmayan tüm işlemciler kapatılabilir. Bu, büyük ya da küçük çekirdeklerin yalnızca ihtiyaç duyulduğunda aktif olmasını ve herhangi bir iş yükünü执行mek için uygun çekirdeğin kullanılmasını sağlar.

MP modelinin uygulanması biraz daha karmaşıktır ancak aynı zamanda kaynaklar açısından daha verimlidir. Görevleri uygun şekilde atar ve talep gerektirdiğinde daha fazla çekirdeğin aynı anda çalışmasını sağlar.

## Önbellek Tutarlılığı ve MOESI Modeli

Tipik olarak, heterojen çok çekirdekli bir işlemci, farklı işlemci türlerine atanmış özel L2 önbelleğe olacaktır. Bunu Şekil 18.7'deki CPU/GPU şemasının genel tasvirinde görmekteyiz. CPU ve GPU oldukça farklı görevlerle meşgul olduğundan, her birinin benzer CPU'lar arasında paylaşılan kendi L2 önbelleğine sahip olması mantıklıdır. Bunu, A7 çekirdeklerinin bir L2 önbelleği paylaştığı ve A15 çekirdeklerinin ayrı bir L2 önbelleği paylaştığı big.Little mimarisinde de (Şekil 18.9) görüyoruz.

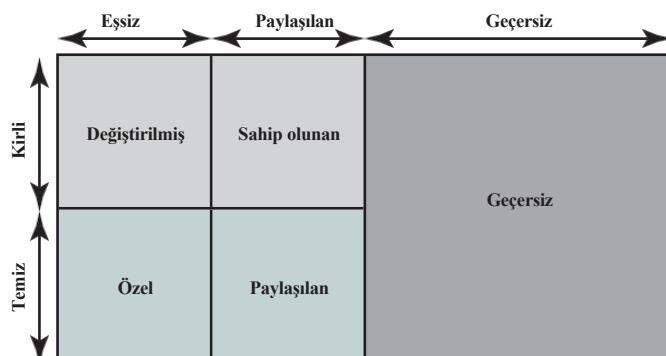
Birden fazla önbellek mevcut olduğunda, geçersiz verilere erişimi önlemek için bir önbellek tutarlılık şemasına ihtiyaç vardır. Önbellek tutarlılığı yazılım tabanlı tekniklerle ele alınabilir. Önbelleğin eski veri içermesi durumunda, önbelleğe alınan kopya geçersiz ve tekrar ihtiyaç duyulduğunda bellekten yeniden okunabilir. Kirli veri içeren bir geri yazma önbelleği nedeniyle bellek eski veri, önbellek belleğe geri yazmaya zorlanarak temizlenebilir. Diğer önbelleklerde bulunabilecek diğer önbellek kopyaları geçersiz kılınmalıdır. Bu yazılım yükü, bir SoC yongasında çok fazla kaynak tüketir ve özellikle heterojen çok çekirdekli işlemcilerde donanım önbelleği uyumlu uygulamaların kullanılmasına yol açar.

Bölüm 17'de açıklandığı gibi, donanım tarafından uygulanan önbellek tutarlılığı için iki ana yaklaşım vardır: dizin protokolleri ve snoopy protokolleri. ARM, ACE (Advanced Extensible Cache) adı verilen bir donanım tutarlılığı özelliği geliştirmiştir.

Arayüz Tutarlılık Uzantıları), directory veya snoopy yaklaşımını veya hatta bir kombinasyonu uygulamak için yapılandırılabilir. ACE, farklı yeteneklere sahip çok çeşitli tutarlı ana bilgisayarları desteklemek üzere tasarlanmıştır. ACE, Cortex-A15 ve Cortex-A7 işlemciler gibi farklı işlemciler arasında tutarlılığı destekleyerek ARM big.Little teknolojisini mümkün kılar. Önbellege alınmamış ana bilgisayarlar için I/O tutarlığını destekler, farklı önbellek satırı boyutlarına sahip ana bilgisayarları, farklı iç önbellek durumu modellerini ve geri yazma veya yazma önbellekli ana bilgisayarları destekler. Başka bir örnek olarak ACE, Şekil 18.8'deki TI SoC yongasında bellek alt sistemi bellek kontrolünde (MSMC) uygulanmaktadır. MSMC, ARM CorePac L1/L2 önbellekleri ve paylaşılan SRAM ve DDR alanları için EDMA/IO çevre birimleri arasında donanım önbellek uyumunu destekler. Bu özellik, açık yazılım önbellek bakım tekniklerini kullanmak zorunda kalmadan MSMC SRAM ve DDR veri alanlarının çip üzerindeki bu ana birimler tarafından paylaşılmasını sağlar.

ACE beş durumlu bir önbellek modeli kullanır. Her önbellekte, her satır ya Geçerli ya da Geçersizdir. Bir satır Geçerli ise, iki boyutla tanımlanan dört durumdan birinde olabilir. Bir satır Paylaşılan veya Benzersiz veriler içerebilir. Paylaşılan bir satır, harici (ana) belleğin potansiyel olarak paylaşılabilen bir bölgesinden veri içerir. Benzersiz bir satır, bu önbelleğin sahibi olan çekirdeğe adanmış bir bellek bölgesinden veri içerir. Ve satır ya Temiz ya da Kirli'dir, yani genellikle ya bellek en son, en güncel veriyi içerir ve önbellek satırı yalnızca belleğin bir kopyasıdır ya da Kirli ise önbellek satırı en son, en güncel veridir bir aşamada belleğe geri yazılması gereklidir. Yukarıdaki açıklamanın tek istisnası, birden fazla önbelleğin bir satırı paylaşması ve bu satırın kirli olmasıdır. Bu durumda, tüm önbellekler her zaman en son veri değerini içermelidir, ancak yalnızca biri Paylaşılan/Kirli durumda olabilir, diğerleri Paylaşılan/Temiz tutulur. Bu nedenle Paylaşımı/Kirli durumu, hangi önbelleğin veriyi belleğe geri yazma sahip olduğunu belirtmek için kullanılır ve Paylaşımı/Temiz daha doğru bir şekilde verinin paylaşıldığı ancak belleğe geri yazılmasına gerek olmadığı anlamına gelir.

ACE durumları, MOESI olarak bilinen beş durumlu bir önbellek tutarlılık modeline karşılık gelir (Şekil 18.12). Tablo 18.2, MOESI modelini Bölüm 17'de açıklanan MESI modeliyle karşılaştırmaktadır.



Şekil 18.12 ARM ACE Önbellek Hattı Durumları

**Tablo 18.2** Snoop Protokollerindeki Durumların Karşılaştırılması

(a) MESIM				
	Değiştirilmiş	Özel	Paylaşılan	Geçersiz
Temiz/Kirli	Kirli	Temiz	Temiz	N/A
Eşsiz mi?	Evet	Evet	Hayır	N/A
Yazabiliyor musun?	Evet	Evet	Hayır	N/A
İleri gidebilir misin?	Evet	Evet	Evet	N/A
Yorumlar	Paylaşmak veya değiştirmek için geri yazmalısınız	Yazma sırasında M'ye geçişler	Paylaşılan temiz anlamına gelir, iletebilir	Okunamıyor

(b) MOESI					
	Değiştirilmiş	Sahip olunan	Özel	Paylaşılan	Geçersiz
Temiz/Kirli	Kirli	Kirli	Temiz	Ya da	N/A
Eşsiz mi?	Evet	Evet	Evet	Hayır	N/A
Yazabiliyor musun?	Evet	Evet	Evet	Hayır	N/A
İleri gidebilir misin?	Evet	Evet	Evet	Hayır	N/A
Yorumlar	Geri yazmadan paylaşabilir	Geçiş için geri yazmalı	Yazma sırasında M'ye geçişler	Paylaşılır, kirli veya temiz olabilir	Okunamıyor

## 18.5 INTEL CORE i7-990X

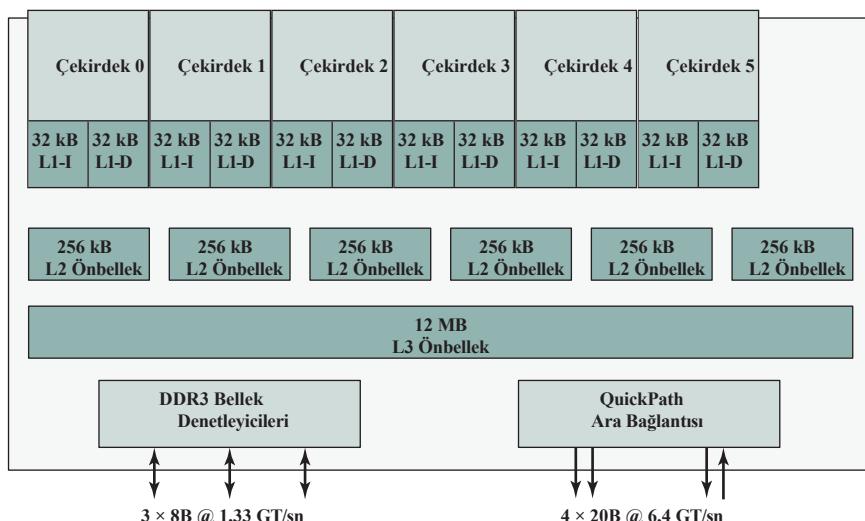
Intel son yıllarda bir dizi çok çekirdekli ürün tanıttı. Bu bölümde Intel Core i7-990X'i inceleyeceğiz.

Intel Core i7-990X'in genel yapısı Şekil 18.13'te gösterilmektedir. Her çekirdeğin kendi **özel L2** önbelleği vardır ve altı çekirdek 12 MB'lık bir **L3** önbelleği paylaşır. Intel'ün belleklerini daha etkili hale getirmek için kullandığı bir mekanizma, donanımın bellek erişim modellerini incelediği ve önbellekleri yakında talep edilmesi muhtemel verilerle spesifik olarak doldurmaya çalıştığı prefetching'dir.

Core i7-990X yongası diğer yongalarla iki tür harici destekler. **DDR3 bellek** denetleyicisi, DDR ana bellek<sup>1</sup> için bellek denetleyicisini çip üzerine getirir. Arayüz, 32 GB/s'ye toplam veri hızı için 192 bitlik toplam veri yolu genişliği için 8 bayt genişliğinde üç kanalı destekler. Çip üzerindeki bellek denetleyicisi ile Ön Yan Veri Yolu ortadan kaldırılmıştır. **QuickPath Interconnect (QPI)**,

Intel işlemciler ve yonga setleri için önbellek uyumlu, noktadan noktaya bağlantı tabanlı bir elektriksel ara bağlantı spesifikasyonudur. Bağlı işlemci yongaları arasında yüksek hızlı iletişim sağlar. QPI bağlantısı 6,4 GT/s (saniye başına aktarım) hızında çalışır. Aktarım başına 16 bit ile bu 12,8 GB/s ve QPI bağlantıları özel çift yönlü çiftler içerdiginden, toplam bant genişliği 25,6 GB/s'dir. Bölüm 3.5 QPI'yi biraz ayrıntılı olarak ele almaktadır.

<sup>1</sup> DDR senkronize RAM bellek Bölüm 5'te ele alınmıştır.



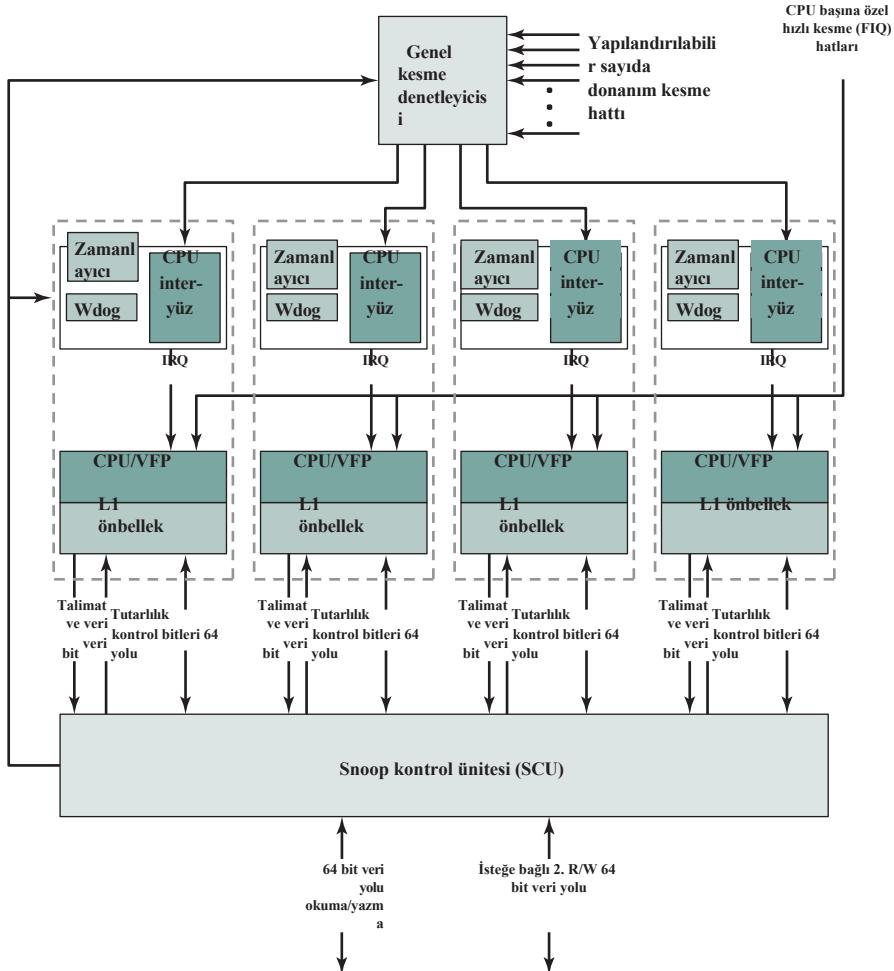
**Şekil 18.13** Intel Core i7-990X Blok Diyagramı

## 18.6 ARM CORTEX-A15 MPCORE

ARM çekirdeklərini kullanan heterojen çox çekirdekli işlemcilerin ikiörneğini Bölüm 18.4'te görmüştük: ARM Cortex-A7 ve Cortex-A15 çekirdeklərini bir arada kullanan big.Little mimarisi ve Cortex-A15 çekirdeklərini TI DSP çekirdekləriyle birləştiren Texas Instruments DSP SoC mimarisi. Bu bölümde, birden fazla A15 çekirdeği kullanan homojen bir çox çekirdekli işlemci olan Cortex-A15 MPCore çox çekirdekli yongayı tanıtıyoruz. A15 MPCore, mobil bilgi işlem, üst düzey dijital ev sunucuları ve kablosuz altyapı gibi uygulamaları hedefleyen yüksek performanslı bir çiptir.

Şekil 18.14' te Cortex-A15 MPCore'un blok diyagramı gösterilmektedir. Sistemin temel unsurları aşağıdaki gibidir:

- **Genel kesme denetleyicisi (GIC):** Kesme algılama ve kesme önceliklendirme işlemlerini gerçekleştirir. GIC, kesintileri ayrı çekirdeklere dağıtır.
- **Hata ayıklama ünitesi ve arayüzü:** Hata ayıklama birimi harici bir hata ayıklama ana bilgisayarının şunları yapmasını sağlar: program yürütmesini durdurma; işlem ve yardımcı işlemci durumunu inceleme ve değiştirme; bellek ve giriş/çıkış çevre birimi durumunu inceleme ve değiştirme; ve işlemciyi yeniden başlatma.
- **Genel zamanlayıcı:** Her çekirdeğin kesme üretebilen kendi özel zamanlayıcısı vardır.
- **İzleme:** Performans izleme ve program izleme araçlarını destekler.
- **Çekirdek:** Tek bir ARM Cortex-15 çekirdeği.
- **L1 önbellek:** Her çekirdek kendi özel L1 veri önbelleğine ve L1 komut önbelleğine sahiptir.
- **L2 önbellek:** Paylaşılan L2 bellek sistemi, her çekirdektenden gelen L1 komut ve veri önbelleği özlemlerine hizmet eder.
- **Snoop kontrol birimi (SCU):** L1/L2 önbellek tutarlığını korumaktan sorumludur.



Şekil 18.14 ARM Cortex-A15 MPCore Çip Blok Diyagramı

## Kesme İşleme

GIC çok sayıda kaynaktan gelen kesintileri harmanlar. Sağlar

- Kesmelerin maskelenmesi
- Kesmelerin önceliklendirilmesi
- Kesmelerin hedef A15 çekirdeklerine dağılımı
- Kesmelerin durumunu izleme
- Yazılım tarafından kesmelerin oluşturulması

GIC, A15 çekirdekerinin yanında sisteme yerleştirilen tek bir işlevsel birimdir. Bu, sistemde desteklenen kesme sayısının A15 çekirdek tasarımlarından bağımsız olmasını sağlar. GIC bellek eşlemelidir; yani GIC için kontrol kayıtları bir ana bellek temel adresine göre tanımlanır. GIC'ye A15 çekirdekleri tarafından SCU üzerinden özel bir arayüz kullanılarak erişilir.

GIC iki işlevsel gereksinimi karşılamak üzere tasarlanmıştır:

- Bir kesme isteğini tek bir CPU'ya veya gerektiği gibi birden fazla CPU'ya yönlendirmek için bir araç sağlayın.
- Bir CPU'daki bir iş parçacığının başka bir CPU'daki bir iş parçacığı tarafından etkinlige neden olabilmesi için işlemciler arası iletişim aracı sağlayın.

Her iki gereksinimi de kullanan bir örnek olarak, birden fazla işlemci üzerinde çalışan iş parçacıklarına sahip çok iş parçacıklı bir uygulama düşünün. Uygulamanın bir miktar sanal bellek tassis ettiğini varsayıyalım. Tutarlılığı korumak için işletim sisteminin tüm işlemcilerdeki bellek çeviri tablolarını güncellemesi gerekir. İşletim sistemi, sanal bellek tassisinin gerçekleştiği işlemcideki tabloları güncelleyebilir ve ardından bu uygulamayı çalıştan diğer tüm işlemcilere bir kesme gönderebilir. Diğer işlemciler daha sonra bellek çeviri tablolarını güncellemeleri gerektiğini belirlemek için bu kesmenin kimliğini kullanabilir.

GIC bir kesmeyi bir veya daha fazla CPU'ya aşağıdaki üç şekilde yönlendirebilir yollar:

- Bir kesme yalnızca belirli bir işlemciye yönlendirilebilir.
- Bir kesme, tanımlanmış bir işlemci grubuna yönlendirilebilir. MPCore, kesmeyi kabul eden ilk işlemciyi, tipik olarak en az yüklü olamı, kesmeyi işlemek için en iyi konumda olarak görür.
- Bir kesme tüm işlemcilere yönlendirilebilir.

Belirli bir CPU üzerinde çalışan yazılım açısından bakıldığında, işletim sistemi kendi dışında herkese, kendine veya belirli diğer CPU'lara kesme oluşturabilir. Farklı CPU'larda çalışan iş parçacıkları arasındaki iletişim için, kesme mekanizması tipik olarak mesaj geçişi için paylaşılan bellek ile birleştirilir. Böylece, bir iş parçacığı bir işlemciler arası iletişim tarafından kesintiye uğradığında, kesintiyi tetikleyen iş parçacığından bir mesaj almak için paylaşılan belleğin uygun bloğundan okur. İşlemciler arası iletişim için CPU başına toplam 16 kesme kimliği mevcuttur.

Bir A15 çekirdeğinin bakış açısından, bir kesme :

- **Etkin Değil:** Etkin Olmayan bir kesme, onaylanmamış veya çoklu işlem ortamında söz konusu CPU tarafından tamamen işlenmiş ancak hedeflendiği bazı CPU'larda hala Beklemede veya olabilen ve bu nedenle kesme kaynağında temizlenmemiş olabilen bir kesmedir.
- **Beklemede:** Beklemede olan bir kesme, ileri sürülen ve söz konusu CPU'da işlemin başlamadığı bir kesmedir.
- **Aktif:** Aktif bir kesme, söz konusu CPU'da başlatılmış ancak işleme tamamlanmamış bir kesmedir. Aktif bir kesme, daha yüksek önceliğe sahip yeni bir kesme A15 çekirdek kesme işlemini kestiğinde öncelenebilir.

Kesintiler aşağıdaki kaynaklardan gelir:

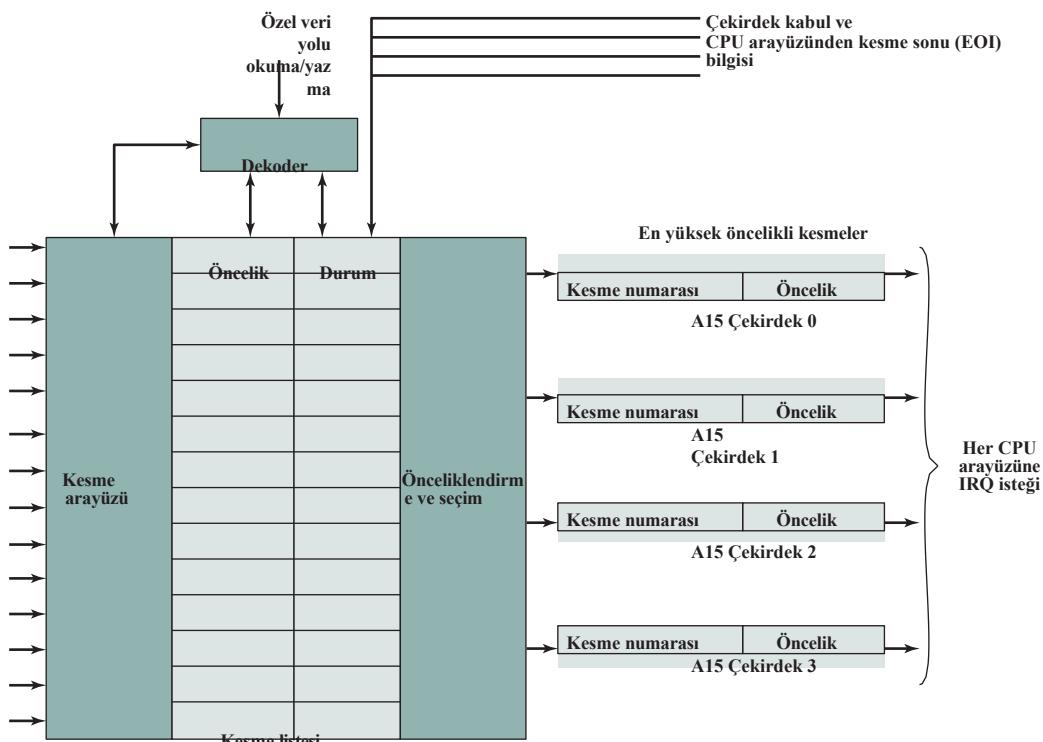
- **İşlemciler arası kesmeler (IPI'lar):** Her CPU'nun sadece yazılım tarafından tetiklenebilen ID0-ID15 özel kesmeleri vardır. Bir IPI'nin önceliği, gönderen CPU'ya değil alıcı CPU'ya bağlıdır.

- **Özel zamanlayıcı ve/veya watchdog kesmeleri:** Bunlar 29 ve 30 numaralı kesme kimliklerini kullanır.
  - **Eski FIQ hattı:** Eski IRQ modunda, eski FIQ pimi, CPU bazında, Kesme Dağıtıcı mantığını atlar ve kesme isteklerini doğrudan CPU'ya yönlendirir.
  - **Donanım kesmeleri:** Donanım kesmeleri, ilişkili kesme giriş hatlarındaki programlanabilir olaylar tarafından tetiklenir. CPU'lar 224 adede kadar kesme giriş hattını destekleyebilir. Donanım kesmeleri ID32'de başlar.

Şekil 18.15 GIC'nin bir blok diyagramıdır. GIC, 0 ile 255 arasında donanım kesme girişini destekleyecek şekilde yapılandırılabilir. GIC, önceliklerini ve durumlarını gösteren bir kesme listesi tutar. Kesme Dağıtıcısı her CPU Arayüzüne o arayüz için Bekleyen en yüksek kesmeyi iletir. Kesmenin onaylandığı bilgisini geri alır ve daha sonra ilgili kesmenin durumunu değiştirebilir. CPU Arayüzü ayrıca Kesme Sonu (EOI) bilgisini de iletir, bu da Kesme Dağıtıcısının bu kesmenin durumunu Aktif'ten Aktif Değil'e güncellemesini sağlar.

## Önbellek Tutarlılığı

MPCore'un Snoop Kontrol Birimi (SCU), paylaşılan verilere erişim ve tutarlılık trafiğinin getirdiği ölçeklenebilirlik sınırlamasıyla ilgili geleneksel darboğazların çoğunu çözmek için tasarlanmıştır.



**Sekil 18.15** Genel Kesme Denetlevicisi Blok Divagramı

**L1** önbellek tutarlılığı L1 önbellek tutarlılığı şeması Bölüm 17'de açıklanan MESI protokolünü temel alır. SCU, MESI durum geçişini optimize etmek için paylaşılan verilerle yapılan işlemleri izler. SCU üç tür optimizasyon sunar: doğrudan veri müdahalesi, çoğaltılmış etiket RAM'leri ve geçiş hatları.

**Doğrudan veri müdahalesi (DDI)**, harici belleğe erişmeden temiz verilerin bir CPU L1 veri önbelleğinden başka bir CPU L1 veri önbelleğine kopyalanmasını sağlar. Bu, Seviye 1 önbellekten Seviye 2 önbelleğe okuma sonrası okuma etkinliğini azaltır. Böylece, yerel bir L1 önbellek özlemi, paylaşılan L2 önbelleğe erişim yerine uzak bir L1 önbellekte çözülür.

Bir önbellek içindeki her satırın ana bellek konumunun o satır için bir etiketle tanımlandığını . Etiketler, önbellekteki satır sayısı ile aynı uzunlukta ayrı bir RAM bloğu olarak uygulanabilir. SCU'da, çoğaltılmış etiket RAM'leri, SCU tarafından ilgili CPU'lara tutarlılık komutları göndermeden önce veri kullanılabilirliğini kontrol etmek için kullanılan L1 etiket RAM'lerinin çoğaltılmış versiyonlarıdır. Tutarlılık komutları yalnızca tutarlı veri önbelleğini güncellemesi gereken CPU'lara gönderilir. Bu, her bellek güncellemesinde her bir işlemciinin önbelleğine girmekten ve onu manipüle etmekten kaynaklanan güç tüketimini ve performans etkisini azaltır. Etiket verilerinin yerel olarak mevcut olması SCU'nun önbellek manipülasyonlarını ortak önbellek satırlarına sahip işlemcilerle sınırlamasını sağlar.

**Geçiş hatları** özelliği, L2'ye yazmadan ve verileri harici bellekten geri okumadan kirli verilerin bir CPU'dan diğerine taşınmasını sağlar. İşlem aşağıdaki gibi tanımlanabilir. Tipik bir MESI protokolünde, bir işlemci değiştirilmiş bir satırı sahiptir ve başka bir işlemci bu satırı okumaya çalışır, aşağıdaki eylemler gerçekleşir:

1. Hat içeriği değiştirilen hattan okumayı başlatan işlemciye aktarılır.
2. Satır içerikleri ana belleğe geri yazılır.
3. Satır her iki önbellekte de paylaşılan duruma getirilir.

## L2 Önbellek Tutarlılığı

SCU, ayrı L1 veri önbellekleri ve L2 önbellek arasındaki tutarlılığı korumak için hibrit MESI ve MOESI protokollerini kullanır. L2 bellek sistemi, L1 veri önbellek dizinlerinin her birinin yinelenen bir kopyası olan bir snoop etiket dizisi içerir. Snoop etiket dizisi, L2 bellek sistemi ile L1 bellek sistemi arasındaki snoop trafiği miktarını azaltır. Snoop etiket dizisinde Mod- ified/Exclusive durumunda bulunan herhangi bir satır L1 bellek sistemine aittir. Bu durumda bir satır isabet eden herhangi bir erişim L1 bellek sistemi tarafından servis edilmeli ve L2 bellek sistemine aktarılmalıdır. Satır geçersizse veya snoop etiket dizisinde paylaşılan , L2 önbelleği verileri sağlayabilir. SCU, ACE üzerinde herhangi bir veri okumadan veya yazmadan çekirdekler arasında doğrudan önbellekten önbelleğe aktarımları işleyebilen tamponlar içerir. Satırlar, L2 önbellekteki satırın MOESI durumunda herhangi bir değişiklik olmadan ileri geri taşınabilir. ACP üzerindeki paylaşılabilebilir işlemler de uyumludur, bu nedenle snoop etiket dizileri ACP işlemlerinin bir sonucu olarak sorgulanır. Paylaşılabilebilir satırın L1 veri önbelleklerinden birinde Modified/Exclusive durumunda bulunduğu okumalarda, satır L1 bellek sisteminden L2 bellek sistemine aktarılır ve ACP üzerinden geri iletilir.

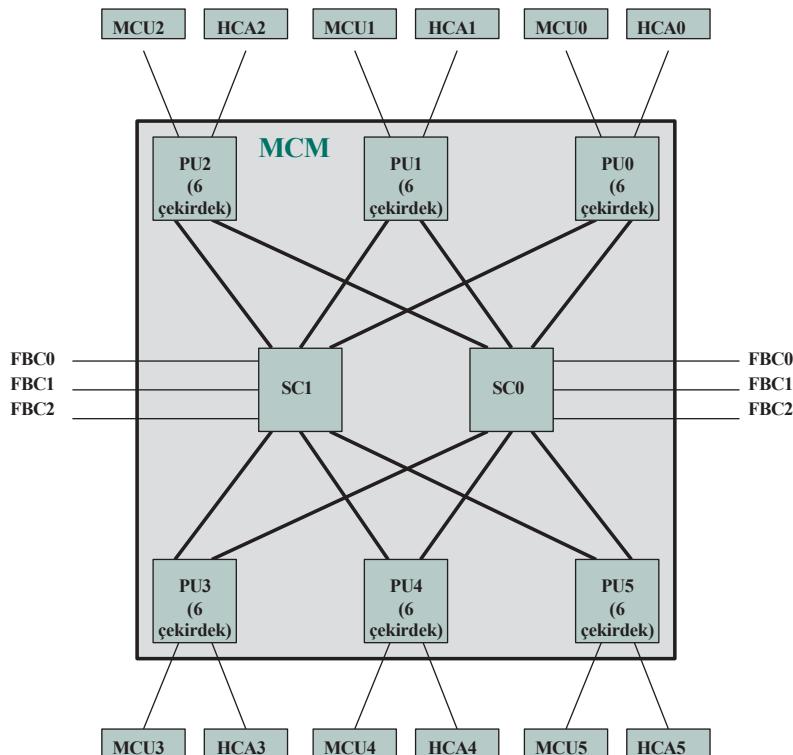
## 18.7 IBM zENTERPRISE EC12 ANA ÇERÇEVE

Bu bölümde, çok çekirdekli işlemci yongaları kullanan bir ana bilgisayar organizasyonuna bakacağınız. Kullandığımız örnek, 2010 yılının sonlarında piyasaya sürülmeye başlanan IBM zEnterprise EC12 anabilgisayar bilgisayarıdır [SHUM13, DOBO13]. Bölüm 7.8, G/C yapısının tartışılmıştır birlikte EC12'ye genel bir bakış sunmaktadır.

### Organizasyon

Ana bilgisayarın temel yapı taşı çok çipli modüldür (MCM). MCM, sekiz çip ve 7356 bağlantı içeren 103 katmanlı bir cam seramik alt tabakadır (96-96 mm boyutunda). Toplam transistör sayısı 23 milyarın üzerindedir. MCM, kitap ambalajının bir parçası olan bir karta takılır. Kitabın kendisi, kitaplar arasında ara bağlantı sağlamak için orta düzlem sistem kartına takılır.

Bir MCM'nin temel bileşenleri Şekil 18.16'da gösterilmektedir:



FBC= kumaş kitabı bağlantıları  
HCA= ana bilgisayar kanal  
bağdaştırıcı MCM = çok çipli  
modül

MCU= bellek kontrol birimi  
PU= işlemci birimi  
SC= depolama kontrolü

**Şekil 18.16** IBM EC12 İşlemci Düğüm Yapısı

- İşlemci birimi (PU):** Her biri dört işlemci çekirdeği artı üç seviye önbellek içeren altı adet 5,5 GHz işlemci PU yongası vardır. PU'ların bellek kontrol birimleri aracılığıyla ana belleğe ve ana bilgisayar kanal adaptörleri aracılığıyla I/O'ya harici bağlantıları vardır. Böylece her MCM 24 çekirdek içerir.
- Depolama kontrolü (SC):** İki SC yongası ek bir önbellek seviyesi ve diğer üç MCM'ye bağlanmak için ara bağlantı mantığı içerir.

Mikroişlemci çekirdeği, saat döngüsü başına üç z/Mimari CISC komutunun kodunu çözebilen (6 0,18 ns) ve döngü başına yedi işleme kadar yürütürebilen geniş bir süper skaler, sıra dışı boru hattına sahiptir. Komut yürütme yolu, dallanma yönü ve hedef tahmin mantığı ile tahmin edilir. Her çekirdek iki tamsayı birimi, iki yükleme/depolama birimi, bir ikili kayan nokta birimi ve bir kayan nokta birimi içerir.

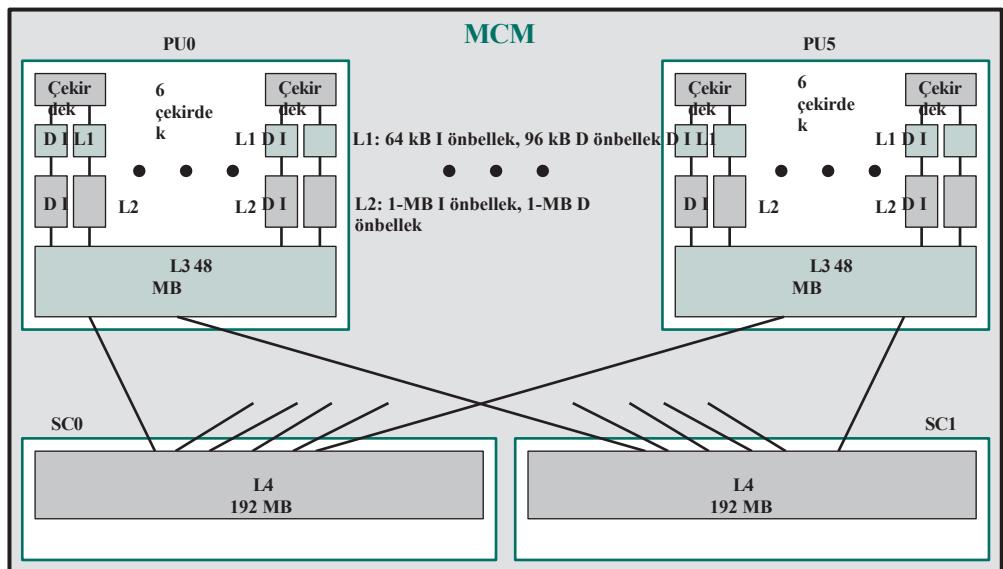
## Önbellek Yapısı

EC12 dört seviyeli bir önbellek yapısına sahiptir. Her seviyeyi sırayla inceleyeceğiz (Şekil 18.17).

Her çekirdek, 96-kB veri önbelleği ve 64-kB komut önbelleğine bölünmüş özel bir 160-kB **L1 önbelleğine** sahiptir. L1 önbelleği, L2'ye yazma olarak tasarlanmıştır, yani değiştirilen veriler bir sonraki bellek seviyesine de depolanır. Bu önbellekler 8 yolu set ilişkiseldir.

Her çekirdek ayrıca 1 MB veri önbelleği ve 1 MB komut önbelleğine eşit olarak bölünmüş özel bir 2 MB L2'ye sahiptir. L2 önbellekleri L3'e yazılabilir ve 8 yolu set ilişkiseldir.

Her 4 çekirdekli işlemci yongası, altı çekirdek tarafından paylaşılan 24 MB **L3 önbellek** içerir. L1 ve L2 önbellekleri write-through olduğundan, L3 önbelleği her



Şekil 18.17 IBM EC12 Önbellek Hiyerarşisi

çipindeki altı çekirdek tarafından üretilen depolar. Bu özellik, bir çekirdek arızası sırasında veri kullanılabilirliğini korur. L3 önbellek 12-yolu set ilişkiseldir. EC12, çip üzerinde L3 önbellek olarak gömülü DRAM (eDRAM) kullanır. Bu eDRAM bellek, normalde ön belleği uygulamak için kullanılan statik RAM'den (SRAM) daha yavaş olsa da, belirli bir alana çok daha fazlasını koyabilirsiniz. Birçok iş yükü için, çekirdeğe daha yakın daha fazla bellege sahip olmak, hızlı belleğe sahip olmaktan daha önemlidir.

Son olarak, bir MCM üzerindeki 6 PU'nun tamamı 160 MB **L4 önbelleği** paylaşır ve bu önbellek her SC yongasında 92 MB'lık bir ön belleğe bölünür. Seviye 4 önbelleğin dahil edilmesindeki temel motivasyon, çekirdek işlemcilerin çok yüksek saat hızının ana bellek hızıyla önemli bir uyumsuzluğa yol açmasıdır. Çekirdeklerin verimli bir şekilde çalışmasını sağlamak için dördüncü önbellek katmanına ihtiyaç duyulmaktadır. Büyük paylaşımlı L3 ve L4 önbellekleri, yüksek derecede veri paylaşımı ve görev değişimi sergileyen işlem işleme iş yükleri için uygundur. L4 önbellek 24 yolu set ilişkiseldir. L4 önbelleğini barındıran SC yongası, aynı zamanda üç çift yönlü veri yolu ile üç uzak kitaba<sup>2</sup> kadar L4'ten L4'e trafik için bir çapraz nokta anahtarı görevi görür. L4 önbelleği tutarlılık yöneticisidir, yani verilerin işlemci tarafından kullanılabilmesi için tüm bellek getirmelerinin L4 önbelleğinde olması gereklidir.

Dört önbellek de 256 baytlık bir satır boyutu kullanılır.

EC12, tasarım ödünləşimleri ve mevcut teknolojiyle giderek daha güçlü hale gelen işlemcilerden yararlanmanın zorluğu konusunda ilginç bir çalışmıştır. Büyüklük L4 önbelleği, ana belleğe erişim ihtiyacını en aza indirmeyi amaçlamaktadır. Ancak, çip dışı L4 önbelleğe olan mesafe bir dizi komut döngüsüne mal olmaktadır. Bu nedenle, çip üzerinde önbelleğe ayrılan alan mümkün olduğunda büyüktür, hatta çip üzerinde mümkün olandan daha az çekirdeğe sahip olma noktasına kadar. L1 önbellekleri, çekirdektenden uzaklığı en aza indirmek ve erişimin tek bir döngüde gerçekleşmesini sağlamak için küçüktür. Her bir L2 önbelleği, paylaşılan bir önbelleğe başvurmadan erişilebilecek önbelleğe alınmış veri miktarını en üst düzeye çıkarmak amacıyla tek bir çekirdeğe ayrılmıştır. L3 önbelleği bir çip üzerindeki dört çekirdek tarafından paylaşılır ve L4 önbelleğine gitme ihtiyacını en aza indirmek için mümkün olduğunda büyütür.

*zEnterprise 196'nın tüm defterleri iş yükünü paylaştığından, dört defterdeki dört L4 önbelleği tek bir L4 önbellek havuzu oluşturur. Bu nedenle, L4'e erişim yalnızca çip dışına değil, belki de kitap dışına çıkmak anlamına gelir ve erişim gecikmesini daha da artırır. Bu, prosesörlerdeki üst düzey önbellekler ile L4 önbellek içeriği arasında nispeten büyük mesafeler olduğu anlamına gelir. Yine de, başka bir kitaptaki L4 önbellek verilerine erişmek, diğer kitaptaki DRAM'e erişmekten daha hızlıdır, bu yüzden L4 önbellekler bu şekilde çalışır.*

Kitap tasarıminin doğasında olan gecikmelerin üstesinden gelmek ve kitap dışı L4 içeriğine erişmek için döngü tasarrufu sağlamak için tasarımcılar, belirli bir mantıksal bölüm iş yükünün mümkün olduğunda fazla işini L4 önbelleği ile aynı kitapta bulunan çekirdeklerle yönlendirerek talimatları ve verileri çekirdeklere mümkün olduğunda yakın tutmaya çalışırlar. Bu, sistem kaynak yöneticisi/zamanlayıcısı ve z/OS dağıticısının birlikte çalışarak mümkün olduğunda az sayıda çekirdek ve L4 önbellek alanı (en iyisi bir kitap sınırı içinde) sınırları içinde mümkün olduğunda fazla işi, verimi ve yanıt sürelerini etkilemeden tutmasıyla elde edilir. Kaynak yöneticisi/çizelgeleyici ve dağıticının iş yüklerini daha az verimli çalışabilecekleri işlemcilere atamasını engellemek, EC12 gibi yüksek frekanslı bir işlemci tasarımda gecikmenin üstesinden gelinmesine katkıda bulunur.

---

<sup>2</sup>Bölüm 7'den EC12 kitabının bir MCM, bellek kartları ve I/O kafes bağlantılarından oluştuğunu hatırlayın.

## 18.8 ANAHTAR TERIMLER, GÖZDEN GEÇİRME SORULARI VE PROBLEMLER

### Anahtar Terimler

Amdahl yasası çip çoklu işlemci kaba taneli iş parçacığı ince taneli iş parçacığı heterojen çoklu çekirdek organizasyon	homojen çok çekirdekli organizasyon hibrit iş parçacığı MOESI protokolü çok çekirdekli işlemci pipelining	Pollack'ın kuralı eszamanlı çoklu iş parçacığı (SMT) süperskalar iş parçacığı tanecikliği
--	---	--

### İnceleme Soruları

- 18.1** Basit komut ardışık dizilimi, süperskalar ve eşzamanlı çoklu iş parçacığı arasındaki farkları özetleyiniz.
- 18.2** Tasarımcıların tek bir işlemci içindeki paralelligi artırmak yerine çok çekirdekli bir organizasyona geçmeyi tercih etmelerinin birkaç nedenini belirtiniz.
- 18.3** Neden çip alanının giderek artan bir bölümünü ön belleğe ayırma yönünde bir eğilim var?
- 18.4** Çekirdek sayısı ile verimi ölçeklendirme becerisinden doğrudan yararlanan bazı uygulama örneklerini listeleyin.
- 18.5** En üst düzeyde, çok çekirdekli bir organizasyondaki ana tasarım değişkenleri nelerdir?
- 18.6** Her bir çekirdek için ayrı ayrı ayrılmış L2 kıyasla çekirdekler arasında paylaşılan bir L2 önbelleğin bazı avantajlarını listeleyin.

### Problemler

- 18.1** Aşağıdaki problemi düşünün. Bir tasarımcının elinde bir çip var ve çipin hangi kısmının ön belleğe (L1, L2, L3) ayrılacagina karar vermesi gerekiyor. Çipin geri kalani bir ya da daha fazla karmaşık süperskalar ve/veya SMT çekirdeğine ayrılacaktır. Aşağıdaki parametreleri tanımlayın:

- $n$ = çip üzerinde bulunabilecek maksimum çekirdek sayısı.
- $k$ = uygulanan gerçek çekirdek sayısı ( $1 \dots k$  ...  
 $n$ , burada  $n/k$  bir tam sayıdır).
- $perf(r)$ =  $r$ 'ye eşdeğer kaynakları kullanarak sıralı performans kazancı  
 $perf(1)=1$  olduğu tek bir işlemci oluşturmak için çekirdekler.
- $f$ = birden fazla çekirdekte paralelleştirilebilen yazılım oranı.

Dolayısıyla,  $n$  çekirdekli bir çip inşa edersek, her çekirdeğin 1'lük sıralı performans sağlamasını ve  $n$  çekirdeğin  $n$  paralel iş parçacığı derecesine kadar paralellikten yararlanabilmesini bekleriz. Benzer şekilde, çipte  $k$  çekirdek varsa, her çekirdek  $perf(r)$  performansı sergilemeli ve çip  $k$  paralel iş parçacığı derecesine kadar paralellikten yararlanabilmelidir. Bu durumu yansıtmak için Amdhal yasasını (Denklem 18.1) aşağıdaki gibi değiştirebiliriz:

$$\text{Hızlandırma} = \frac{1}{\frac{1 - f}{perf(r)} + \frac{f * r}{perf(r) * n}}$$

- a. Amdahl yasasının bu değişikliğini gerekçelendirin.
  - b. Pollack kuralını kullanarak  $\text{perf}(r) = 2r$  olarak ayarlarız.  $n= 16$  olsun. Hızlanmayı  $f= 0,5$ ;  $f= 0,9$ ;  $f= 0,975$ ;  $f= 0,99$ ;  $f= 0,999$  için  $r$ 'nin bir fonksiyon olarak çizmek istiyoruz. Sonuçlar bu kitabın Premium İçerik sitesindeki bir belgede mevcuttur (multicore-performance.pdf). Ne gibi sonuçlar çıkarabileceğiniz?
  - c. (b) bölümünü  $n= 256$  için tekrarlayın.
- 18.2** Cortex-A15'in teknik referans kılavuzunda GIC'nin bellek eşlemeli olduğu belirtilmektedir. , çekirdek işlemciler GIC ile iletişim kurmak için bellek eşlemeli I/O kullanır. Bellek eşlemeli G/C ile bellek konumları ve G/C aygıtları için tek bir adres alanı olduğunu Bölüm 7'den hatırlayın. İşlemci, G/C modüllerinin durum ve veri kayıtlarını bellek konumları olarak ele alır ve hem belleğe hem de G/C aygıtlarına erişmek için aynı makine komutlarını kullanır. Bu bilgilere dayanarak, çekirdek işlemcilerin GIC ile iletişim kurması için Şekil 18.15'teki blok diyagramında hangi yol kullanılır?
- 18.3** Bu soruda, aşağıdaki C programının çok iş parçacıklı bir mimarideki performansını analiz ediyoruz. A, B ve C dizilerinin bellekte üst üste binmediğini varsayıyoruz.

```
for (i=0; i<328; i++) {
    A[i] = A[i]*B[i];
    C[i] = C[i]+A[i];
}
```

- Makinemiz tek sorunlu, sıralı bir işlemcidir. Sabit yuvarlak robin çizelgeleme kullanarak her döngüde farklı bir iş parçacığına geçer. N iş parçacığının her biri her N döngüde bir komut yürütür. Kodu, her iş parçacığı orijinal C kodunun her N. yinelemesini yürütecek şekilde iş parçacıklarına tahsis ediyoruz.
- Tamsayı komutlarının yürütülmesi 1 döngü, kayan nokta komutlarının yürütülmesi 4 ve bellek komutlarının yürütülmesi 3 döngü sürer. Tüm yürütme birimleri tamamen boru hattına sahiptir. Bir komut, verileri henüz mevcut olmadığı için yayınlanamazsa, boru hattına bir baloncu ekler ve I döngünden sonra yeniden dener.
- Aşağıda, tüm döngüyü yürüten tek bir iş parçacığı için bu makine için assembly kodundaki programımız yer almaktadır.

```
döngü: ld f1, 0 (r1)          ;f1= A[i]
        ld f2, 0 (r2)          ;f2 = B[i]
        fmul f4, f2, f1         ;f4= f1*f2
        st f4 0(r1)            ;A[i] = f4
        ld f3, 0(r3)            ;f3 = C[i]
        fadd f5, f4, f3         ;f5= f4+ f3 st
        f5 0(r3)                ;C[i] = f5
        r1, r1, 4 ekle           ;i++
        add r2, r2, 4
        r3, r3, 4 ekle
        r4, r4, -1 ekle
        bnez r4, loop           ;loop
```

- a. Döngünün montaj kodunu  $N$  iş parçacığına tahsis ediyoruz, öyle ki her iş parçacığı orijinal her  $N$ 'inci yinelemesini yürütüyor.  $N$  iş parçacığından birinin bu çok iş parçacıklı makinede yürüteceği montaj kodunu yazın.
- b. Programımız için her döngüde bir komut yayınlayan bu makinenin tam olarak kullanılmaya devam etmesi için gereken minimum iş parçacığı sayısı nedir?
- c. Talimatları yeniden düzenleyerek bu programı daha az iş parçacığı kullanarak çalıştırarak en yüksek performansa ulaşabilir miyiz? Kısaca açıklayın.
- d. Bu program için flop/çevrim cinsinden en yüksek performans ne olacaktır?

- 18.4** MOESI protokolü için herhangi bir önbellek çifti düşünün. Belirli bir önbellek satırı için hangi durumlara izin verildiğini belirtmek için aşağıdaki matrisi kullanın; yasak için X ve izin verilen için onay işaretini kullanın.

	M	O	E	S	I
M					
O					
E					
S					
I					

- 18.5** MOESI protokolü için Şekil 17.6'ya benzer şekilde, geçişler üzerindeki etiketler de dahil olmak üzere bir durum geçiş diyagramı çizin.
- 18.6** MESI veya MOESI tabanlı olanlar gibi dizin önbellek uyum protokollerinde, sessiz geçiş, bir önbellek satırının bu değişikliği merkezi denetleyiciye bildirmeden bir durumdan diğerine geçtiği bir durumdur.
- MESI protokolündeki her bir durum için, eğer varsa, hangi hedef durumlara sessiz geçişin mümkün olduğunu belirtin.
  - MOESI için tekrarlayın.

# BÖLÜM 19

## GENEL AMAÇLI GRAFIK İŞLEME BİRİMLERİ

Katkıda bulunan  
**Peter Zeno**

Doktora Adayı, Bridgeport Üniversitesi

### 19.1 CUDA Temelleri

### 19.2 GPU'ya karşı CPU

CPU ve GPU Mimarileri Arasındaki Temel Farklar Performans ve Watt Başına Performans Karşılaştırması

### 19.3 GPU Mimarisine Genel Bakış

Temel GPU Mimarisi Tam Çip  
Düzeni  
Akiş Çoklu İşlemci Mimarisi Ayrıntıları  
Hafıza Türlerini Bilmenin ve Programlamanın Önemi

### 19.4 Intel'in Gen8 GPU'su

### 19.5 GPU Ne Zaman Yardımcı İşlemci Olarak Kullanılır?

### 19.6 Anahtar Terimler ve Gözden Geçirme Soruları

## ÖĞRENİM HEDEFLERİ

Bu bölümü çalışıktan sonra şunları yapabilmelisiniz:

- 『 CUDA'ya genel bir bakış sunun.
- 『 GPU ve CPU arasındaki farkı anlayın.
- 『 Tipik bir GPU mimarisinin temel unsurlarını tanımlayın.
- 『 Bir GPU'nun ne zaman yardımcı işlemci olarak kullanılacağını tartışın.

**Grafik işlemci birimi (GPU)**, hızlı üç boyutlu (3D) grafik oluşturma ve video işleme için optimize edilmek üzere özel olarak tasarlanmıştır. GPU'lar günümüzün neredeyse tüm iş istasyonlarında, dizüstü bilgisayarlarında, tabletlerinde ve akıllı telefonlarında bulunabilir [OWEN08]. GPU'nun birçok boyutu vardır. Daha büyük birimler, tek bir entegre devre (IC) üzerinde birkaç yüz ila binlerce paralel işlemci çekirdeğine sahiptir. Bunlar iş istasyonlarında, oyun sistemlerinde ve hatta süper bilgisayarlarda genellikle PCIe tabanlı ayrı yardımcı işlemci kartları olarak bulunabilir [SLAV12]. En küçük GPU'lar, GPU'nun yalnızca tek haneli sayıda çekirdekten oluşan tabletler ve akıllı telefonlar gibi gömülü sistemlerde bulunur ve tipik olarak aynı silikon IC üzerinde **merkezi işlem birimleri (CPU'lar)** olarak adlandırılan bir dizi geleneksel çekirdekle birleştirilir.

Geçtiğimiz birkaç yıl içinde GPU, biyoinformatic, moleküler dinamikler, petrol ve gaz arama, hesaplamalı finans, sinyal ve ses işleme, istatistiksel modelleme, bilgisayarla görme ve tıbbi görüntüleme gibi çok çeşitli uygulamalar için büyük ölçüde paralel programlama ortamlarında kendine yer buldu. **GPU (GPGPU) kullanarak genel amaçlı hesaplama** terimi buradan türetilmiştir. Yüksek oranda paralelleştirilebilir uygulamaların GPU'ya taşınmasının ana nedenleri, NVIDIA'nın CUDA'sı ve Khronos Group'un OpenCL'si programcı dostu GPGPU dillerinin ortaya çıkması, genel amaçlı hesaplamayı kolaylaştırmak için GPU mimarisinde yapılan bazı küçük değişiklikler [SAND10] (bundan sonra GPGPU mimarisi olarak anılacaktır) ve GPU'ların düşük maliyeti ve yüksek performansından kaynaklanmaktadır. Örneğin, yaklaşık 200 \$ karşılığında iş istasyonunuz için 960 paralel işlemci çekirdeğine sahip bir GPU satın alabilirsiniz (örneğin, NVIDIA'nın GeForce GTX 660'i).

Bu bölümde, GPU'ların tasarımını ve kullanımını anlamak için çok önemli olan CUDA modeline genel bir bakış ile başlıyoruz. Daha sonra, bölüm GPU'lar ve CPU'ları karşılaştırmaktadır. Bunu GPU mimarisine detaylı bir bakış takip etmektedir. Daha sonra Intel'in GPU'su incelenmektedir. Son olarak, bölüm bir GPU'nun ne zaman yardımcı işlemci olarak kullanılacağını tartışmaktadır.

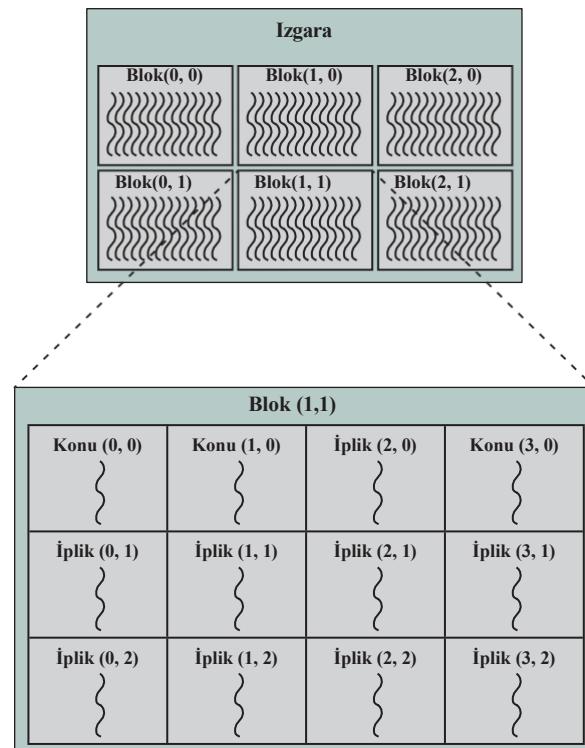
## 19.1 CUDA TEMELLERİ

**CUDA (Compute Unified Device Architecture)**, NVIDIA tarafından oluşturulan ve üretikleri grafik işlem birimleri (GPU'lar) tarafından uygulanan bir paralel hesaplama platformu ve programlama modelidir. GPGPU mimarısını yeterince açıklamak için öncelikle birkaç CUDA yazılım terimi ve kavramının ele alınması gereklidir. Bu bölüm ve kitabın odak noktası bilgisayar mimarisi olduğundan, bu hiçbir şekilde CUDA programlama diline kapsamlı bir giriş değildir. Bununla birlikte

CUDA yazılım terminolojisi ve programlama çerçevesi ile temel oluşturmadan GPGPU sisteminin donanım kısmını tanımlamak zordur. Bu kavramlar GPU/GPGPU mimarisi alanına taşınacaktır.

CUDA C, C/C++ tabanlı bir dildir. Bir CUDA programı üç genel bölüme ayrılabilir: (1) ana (CPU) çalıştırılacak kod; (2) cihazda (GPU) çalıştırılacak kod; ve (3) ana bilgisayar ile cihaz arasında veri aktarımıyla ilgili kod. Ana bilgisayarda çalıştırılacak kod elbette paralelleştirilememen ya da paralelleştirilmeye değmeyen seri koddur. GPU üzerinde çalıştırılacak veri paralel koduna **çekirdek** adı verilirken, bir **iş parçacığı** bu çekirdek işlevinin tek bir örneğidir. Çekirdek tipik olarak çok az dallanma ifadesine sahip olacak ya da hiç olmayacağıdır. Çekirdekteki dallanma ifadeleri GPU donanımındaki iş parçacıklarının seri olarak yürütülmesine neden olur. Bu konu hakkında daha fazla bilgi Bölüm 19.3'te ele alınacaktır.

Programcı, çekirdek işlevi çağrıldığında başlatılan iş parçacığı sayısını tanımlar. **GPU işlemci çekirdeklerinin (CUDA çekirdekleri)** olarak da bilinir) kullanımını en üst düzeye çıkarmak ve mevcut hızlanmayı en üst düzeye çıkarmak için tanımlanan toplam iş parçacığı sayısı genellikle binlerce olur. Ayrıca programcı bu iş parçacıklarının nasıl bir araya getirileceğini de belirler. Daha olmak gereklirse, iş parçacıkları tek biçimli olarak **bloklar** halinde paketlenir ve çekirdek başlatma başına blok sayısı (**iş parçacığı blokları** olarak da bilinir) **izzara** olarak adlandırılır (bkz. Şekil 19.1). Tablo 19.1, az önce tanımlanan CUDA terimlerinin bir özeti vermektedir.



**Şekil 19.1** İş Parçacıkları, Bloklar ve Izgara Arasındaki İlişki

**Tablo 19.1** CUDA Terimleri ile GPU'nun Donanım Bileşenleri Eşdeğerlik Eşlemesi

CUDA Terimi	Tanım	Eşdeğer GPU Donanım Bileşeni
Çekirdek	GPU üzerinde çalıştırılacak bir fonksiyon biçiminde paralel kod	Geçerli değil
Konu	GPU üzerinde bir çekirdek örneği	GPU/CUDA işlemci çekirdeği
Blok	Belirli bir SM'ye atanmış bir grup iş parçası	CUDA çoklu işlemci (SM)
Izgara	GPU	GPU

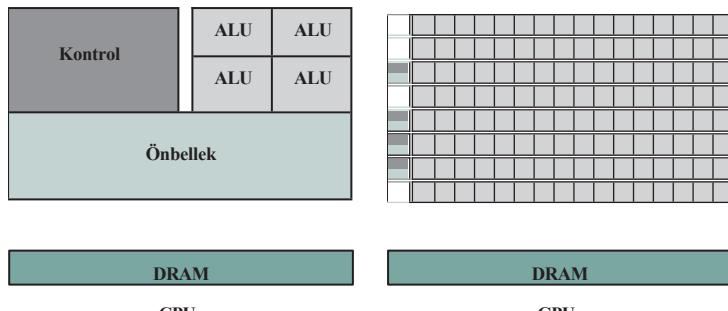
Şekil 19.1'de iki boyutlu iplik bloklarından oluşan iki boyutlu bir **izgara** gösterilmektedir. Hem izgara hem de blok boyutları bir, iki ya da üç boyutlu olabilir. Aynı boyutlara sahip olmaları gerekmektedir. Örneğin, izgara bir boyuta ve iplik bloğu üç ayarlanmış olabilir. Ancak, birazdan göreceğimiz gibi, bu yapılandırma GPU işlemcisini tam olarak kullanamaz, çünkü bir blok birkaç GPU **akış çoklu** işlemcisinden (**SM**) yalnızca birine atanır. Bir blok asla SM'ler arasında bölünmez. Bu nedenle, bir SM tüm işlem yükünü taşıırken GPU işlemci çekirdeklerinin biri hariç tümü boşta kalacaktır. Ayrıca, bir SM'nin kabul edebileceği maksimum iş parçası sayısı vardır. Bu sayı aşılırsa kod derlenmeyecektir. Bu nedenle, kullanılacak GPU'nun özellik verilerini kullanmak ve yükü mümkün olduğunda eşit bir şekilde dağıtmak programcıya bağlıdır. En azından, başlatılan iş parçası bloklarının sayısı GPU'daki SM sayısından az olmamalıdır. Ancak, optimum yapılandırmayı bulmak çok zaman alan ve göz korkutucu bir süreç olabilir.

## 19.2 GPU'YA KARŞI CPU

Bu bölümde GPU ve CPU'nun birbirini tamamlayan mimarileri karşılaştırılmaktadır. GPU ve CPU birbirlerine göre ortogonal olarak optimize edildiğinden, heterojen bir GPGPU sisteminde bir araya getirmeleri, saf CPU yaklaşımına kıyasla belirli uygulamalar için üstün maliyet ve performans kazanımları sağlar.

### CPU ve GPU Mimarileri Arasındaki Temel Farklar

GPU ve CPU önemli ölçüde farklı iki uygulama türü için tasarlanıp optimize edildiğinden, mimarileri de önemli ölçüde farklılık göstermektedir. Bu, iki tür işlemci teknolojisi için önbellek, kontrol mantığı ve işleme mantığına ayrılan kalıp alanının (transistör sayısı) göreceli miktarını karşılaştırarak görülebilir (bkz. Şekil 19.2). CPU'da, Bölüm 18'de tartışıldığı gibi, kontrol mantığı ve ön bellek CPU'nun gayrimenkulünün çoğunu oluşturmaktadır. Bu, sıralı kodu olabildiğince hızlı işlemek üzere ayarlanmış bir mimari için beklentiği gibidir. Öte yandan, bir GPU esas olarak matematiksel işlemleri gerçekleştirmek için büyük ölçüde paralel bir SIMD (tek komut çoklu veri) mimarisini kullanır. Bu nedenle, bir GPU CPU'nun kontrol mantığının aynı karmaşık yeteneklerini gerektirmez (yani, sıra dışı yürütme, dal tahmini, veri tehlikeleri, vb.) Büyük miktarlarda ön bellek de gerektirmez. GPU'lardan basitçe aynı kod dizisini büyük miktarlarda veri üzerinde çalıştırır.



**Şekil 19.2** CPU ve GPU Silikon Alanı/Transistör Ayrimı

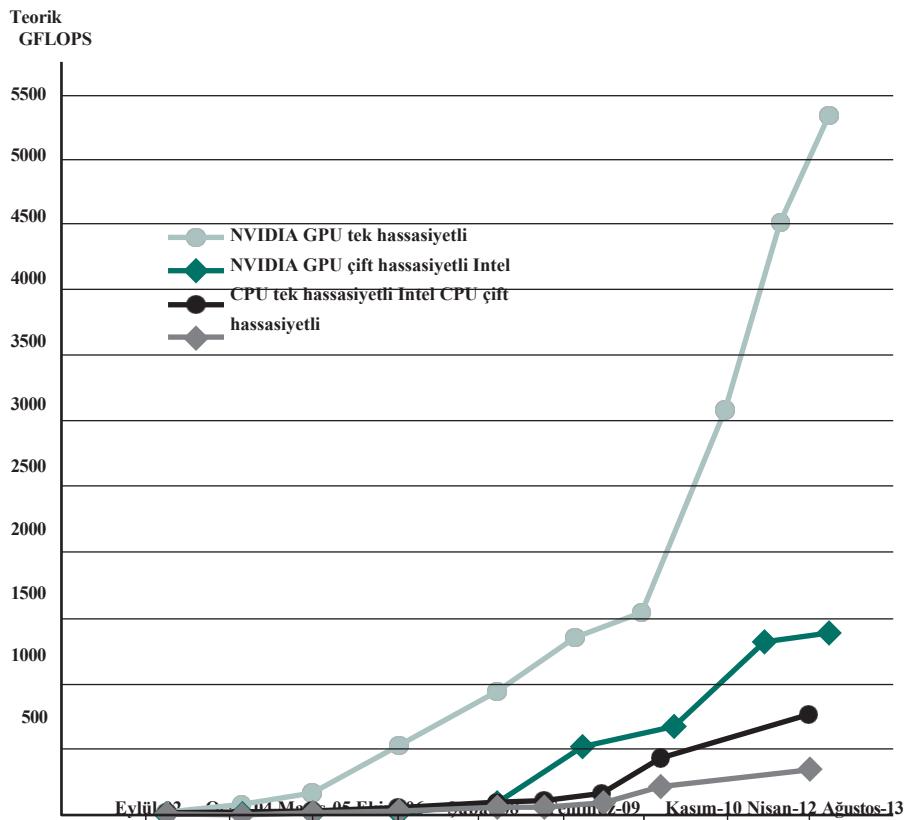
ve mevcut işlemci çekirdeklerinden daha fazla iş parçacığının yürütülmesini yöneterek bellek gecikmesini gizleyebilirler.

### Performans ve Watt Başına Performans Karşılaştırması

Video oyun pazarı, sürekli artan gerçek zamanlı grafik gerçekçiliğine ihtiyaç duyulmasına neden olmuştur. Bu da daha fazla kayan nokta sahip daha fazla paralel GPU işlemci çekirdeği anlamına gelmektedir. Sonuç olarak GPU, gerçekleştirebileceği saniye başına kayan nokta işlemlerinin (FLOPs) sayısını en üst düzeye çıkarmak için tasarlanmıştır. Ayrıca, Kepler ve Maxwell mimarileri gibi daha yeni NVIDIA mimarileri, her bir GPU işlemci çekirdeğinin ihtiyaç duyduğu gücün azaltarak önceki GPU mimarilerine göre watt başına performans oranını (FLOPs/watt) artırmaya odaklanmıştır. Bu, Kepler ile işlemci çekirdeklerinin saatini düşürerek gerçekleştiriliyorken, çip üzerindeki transistörlerin sayısını artırarak (Moore Yasası'nı takip ederek) Fermi mimarisine göre watt başına 3 kat daha fazla performans elde edilmesini sağladı. Buna ek olarak, Maxwell mimarisini yürütme verimliliğini de artırmıştır. Bir GPU'nun çok çekirdekli bir CPU'ya kıyasla gerçekleştirebileceği FLOP'bu artış eğilimi üstel bir oranda farklılaşmış (bkz. Şekil 19.3 [NVID14]) ve böylece büyük bir performans farkı yaratmıştır. Bu iki farklı işleme teknolojisi arasındaki watt başına performans farkı için de benzer şeyler söylenebilir.

## 19.3 GPU MİMARİSİNE GENEL BAKIŞ

GPU mimarisinin tarihsel evrimi üç ana aşamaya veya döneme ayrılabilir. İlk aşama GPU ilk günlerini (1980'lerin başından 1990'ların sonuna kadar) kapsar; burada GPU sabit, programlanamayan, özel işlem aşamalarından (ör. vertex, raster, shader, vb.) oluşmaktadır. Buna ek olarak, bu dönemde devam eden teknolojik gelişmeler, bir grafik sisteminin boyutunda ve maliyetinde dramatik bir düşüşe izin vererek, grafik işlemcilerini 1990'ların ortalarından sonlarına kadar PC'ye getirdi. İkinci aşama, ortaya çıkan Faz I GPU mimarisinin sabit, özelleşmiş bir donanım borusundan tamamen programlanabilir bir işlemciye (yaklaşık olarak 2000'lerin başından ortalarına kadar) yinelemeli olarak değiştirilmesini kapsayacaktır. NVIDIA tarafından 2006 yılında tanıtılan genel ve son değişiklik, yeni GPGPU dili olan CUDA'nın kullanımını kolaylaştırmıştır. Üçüncü aşama, ikinci aşamanın kaldığı yerden devam etmeye ve GPU/GPGPU mimarisinin aşağıdakiler için nasıl mükemmel ve uygun fiyatlı, yüksek oranda paralelleştirilmiş bir SIMD yardımcı işlemci oluşturduğunu ele almaktadır.



**Şekil 19.3** CPU ve GPU için Saniye Başına Kayan Nokta İşlemleri

Grafikle ilgili olmayan bazı programların çalışma sürelerini hızlandırmamanın yanı sıra bir GPGPU dilinin (bu durumda CUDA) bu mimariyle nasıl eşleştiği. Bu bölümün odak noktası GPU'nun bu üçüncü aşamasını veya çağlığını takip etmektedir.

GPGPU destek donanımı eklenen ilk NVIDIA GPU'su GeForce 8800 GTX idi. GPU'nun programcılar tarafından genel amaçlı paralel hesaplama uygulamaları için kullanılmasını sağlamak amacıyla gerçek bir önbellek hiyerarşisi ve kullanıcı tarafından adreslenebilir bir paylaşılan bellek eklenmiştir. Ayrıca, programlanabilir GPU işlemci çekirdeklerinin dizileri ölçeklenebilir SM'lere eşit olarak bölünmüştür. Böyle bir mimarinin avantajı, CUDA programlama dilinde değişiklik gerektirmeden GPU işlemci çekirdeklerinin yanı sıra yeni nesil veya farklı GPU modellerinde SM'lerin ölçeklenebilirliğidir.

### Temel GPU Mimarisi

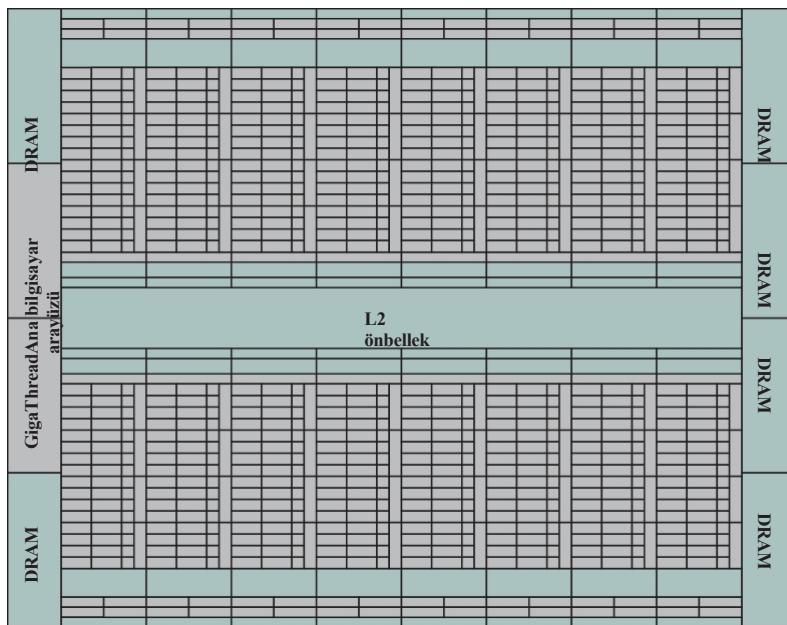
Daha önce de belirtildiği gibi, NVIDIA, her biri bir önceki modele göre mikro mimarisinde küçük ila orta düzeyde farklılıklar gösteren birkaç nesil GPU işleme teknolojisi (yani Tesla, Fermi, Kepler ve Maxwell) boyunca ilerlemiştir. SM için isimlendirme kuralı yeni nesiller için biraz değiştirilmiştir

Kepler için SMX ve Maxwell için SMM gibi GPU teknolojileri. Bu, SM mimarisinde selefine göre nispeten önemli bir değişikliğe işaret etmeye yardımcı oluyor (aynı zamanda yeni ürünün tanıtım pazarlamasına da yardımcı oluyor!). Bununla birlikte, CUDA programlama perspektifinden bakıldığından, tüm bu işlem teknolojileri hala aynı üst düzey mimarilere sahiptir.

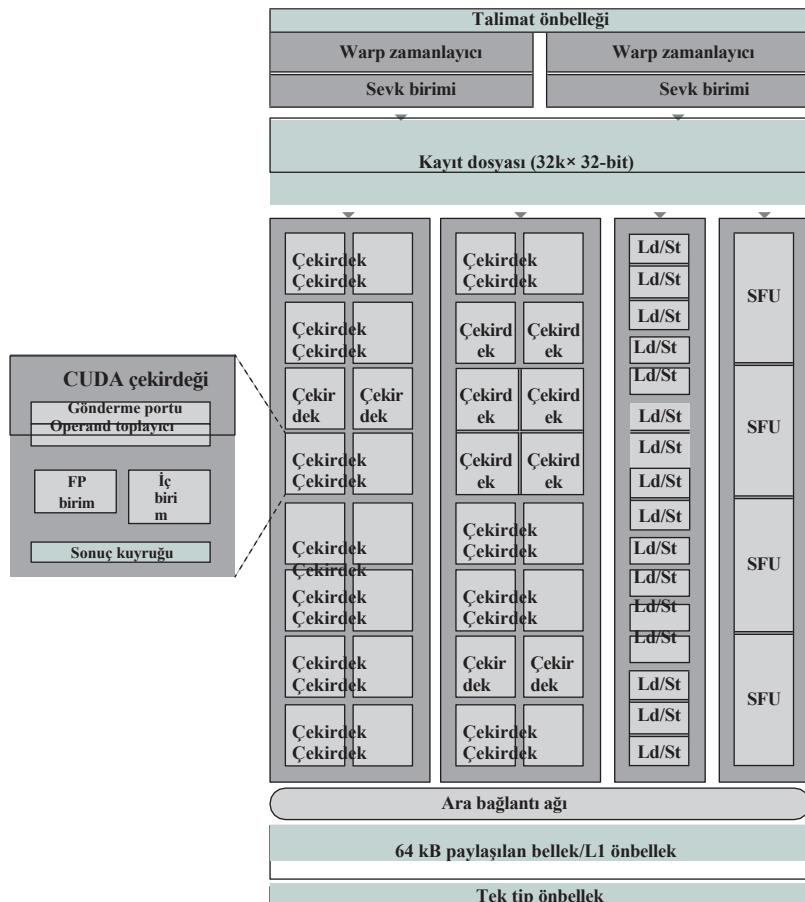
Bu bölümün geri kalanında örnek temel mimari olarak NVIDIA'nın Fermi mimarisini kullanacağız. Fermi mimarisi, oldukça temsili GPU mimarisi ve GPU donanımı ile CUDA yazılımları arasındaki eşleşmeyi basitleştiren düşük CUDA çekirdeği/SM sayısı nedeniyle seçilmiştir. Bu örnek mimari, her SM'nin 32 CUDA çekirdeğinden oluşan bir grup içerdeği 16 SM'den oluşmaktadır. Bu nedenle, Fermi GPU'da toplam 16 SM \* 32 CUDA çekirdeği / SM veya 512 CUDA çekirdeği vardır.

### Tam Çip Düzeni

Şekil 19.4, NVIDIA Fermi mimarili GPU'nun genel düzenini göstermektedir. Bu şekilde görülebileceği gibi, L2 önbelleği 16 SM'nin (yukarıda ve aşağıda 8 SM) merkezine yerleştirilmiştir. Her SM, 16 yükleme/depolama biriminden oluşan bir sütun ve 4 özel işlev biriminden (SFU) oluşan bir sütun ile birlikte 2 bitişik sütun ve 16 sıra dikdörtgen GPU işlemci çekirdeği ile temsil edilir. SM modülünün daha ayrıntılı bir gösterimi Şekil 19.5'te gösterilmektedir [NIVD09]. Şekil 19.4'te SM'lerin baş ve ayak kısımlarındaki dikdörtgenler yazmaçların ve L1/paylaşılan belleğin bulunduğu yerlerdir. Altı DRAM I/O arayüzünün her biri 64 bit bellek arayüzüne sahiptir (DRAM arayüz devresi en dişındaki sol ve sağ taraflarda koyu mavi dikdörtgenlerle gösterilmiştir). Böylece, toplu olarak, GPU'nun GDDR5'ine (grafik çift verisi) 384 bitlik bir arayüz vardır.



**Şekil 19.4** NVIDIA Fermi Mimarisi



**Şekil 19.5** Tek SM Mimarisi

oranı, özellikle grafik işleme için tasarlanmış bir DDR bellek) DRAM, toplam 6 GB'a kadar SM çip dışı bellek (yani global, sabit, doku ve yerel) destegine izin verir. Bu farklı bellek türleri hakkında daha fazla ayrıntı bir sonraki bölümde ele alınacaktır. Ayrıca, Şekil 19.4'te GPU yerleşim şemasının sol tarafında bulunan ana bilgisayar arayüzü gösterilmektedir. Ana bilgisayar arayüzü GPU ve CPU arasında PCIe bağlantısına izin verir. Son olarak, turuncu renkte ve ana bilgisayar arayüzünün yanında bulunan GigaThread global , iş parçacığı bloklarının SM'nin tüm warp zamanlayıcılarına dağıtılmamasından sorumludur (bkz. Şekil 19.5).

### Akış Çoklu İşlemci Mimarisi Ayrıntıları

Şekil 19.5'in sağ tarafı NVIDIA Fermi mimarisini tek bir SM için temel bileşenlerine ayırır. Bu bileşenler şunlardır

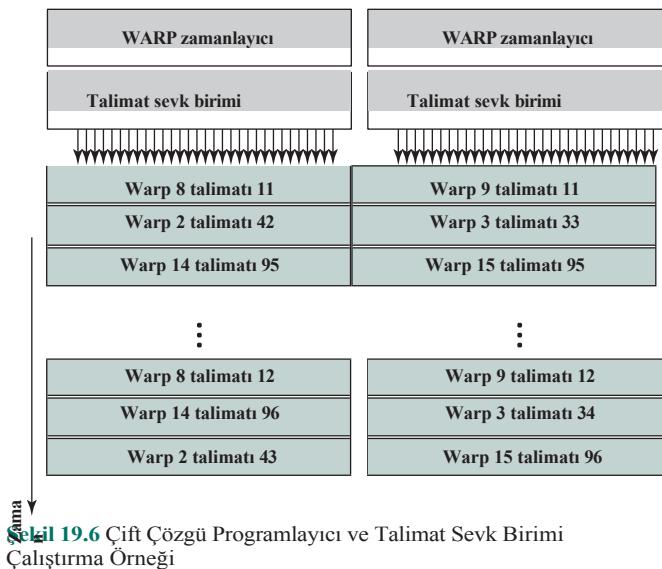
- GPU işlemci çekirdekleri (toplam 32 CUDA )
- Warp zamanlayıcı ve sevk portu

- On altı yükleme/depolama ünitesi
- Dört SFU
- $32k * 32$  bit kayıtlar
- Paylaşılan bellek ve L1 önbellek (toplam 64 kB)

**DUAL WARP SCHEDULER** Daha önce anlatıldığı gibi, GPU çipindeki GigaThread global scheduler birimi iş parçacığı bloklarını SM'lere dağıtır. İkili **çözgү** zamanlayıcı daha sonra işlediği her bir iş parçacığı bloğunu **çözgүlere** ayırır; burada çözgү, aynı başlangıç adresinden başlayan ve iş parçacığı kimlikleri ardışık olan 32 iş parçacığından oluşan bir demetidir. Bir çözgү yayınlandığında, her iş parçacığı kendi komut adres sayacına ve kayıt setine sahip olacaktır. Bu, SM'deki her bir iş parçacığının bağımsız olarak dallanmasına ve yürütülmesine olanak tanır.

GPU, CUDA çekirdeklerinin maksimum düzeyde kullanılmasını sağlamak için mümkün olduğunda çok çözgү işlediğinde en verimli hale gelir. Şekil 19.6'da gösterildiği gibi, maksimum SM donanım kullanımını, çift çözgү zamanlayıcıları ve komut gönderme birimleri her iki saat döngüsünde iki çözgү yayınlayabildiğinde gerçekleşecektir (Fermi mimaris). Daha sonra açıklanacağı üzere, yapısal tehlikeler bir SM'nin bu maksimum işlem hızına ulaşamamasının ana kaynağıdır, çip dışı bellek erişim gecikmesi ise daha kolay gizlenebilir.

Her bir 16 CUDA çekirdeği (\* 2), 16 yükleme/depolama birimi ve 4 SFU'dan (bkz. Şekil 19.5) bölünmüş sütun, bileşen sütunun yapısal bir tehlke yaşamaması koşuluyla, saat döngüsü başına iki çözgү zamanlayıcı/dağıtım biriminin her birinden işlenmek üzere yarım çözgü (16 iş parçacığı) atanmaya uygundur. Yapısal tehlikeler sınırlı SFU'lar, çift hassasiyetli çarpma ve dallanmadan kaynaklanır. Bununla birlikte, çözgү zamanlayıcıları, yapısal tehlikelerin yanı sıra yürütme için mevcut olan çözgülerini izlemek için yerleşik bir puan tablosuna sahiptir. Bu, SM'nin hem yapısal tehlikeler etrafında çalışmasına hem de çip dışı bellek erişim gecikmesini mümkün olduğunda en iyi şekilde gizlemeye yardımcı olmasına olanak tanır.



**Şekil 19.6** Çift Çözgү Programlayıcı ve Talimat Sevk Birimi Çalıştırma Örneği

Bu nedenle, programcının iş parçacığı blok boyutunu bir SM'deki toplam CUDA çekirdeği sayısından daha büyük, ancak blok başına izin verilen maksimum iş parçacığından daha küçük olarak ayarlaması ve SM'lerin optimuma yakın kullanımını sağlamak için iş parçacığı blok boyutunun ( $x$  ve/veya  $y$  boyutlarında) 32'nin çözgü ) bir katı olduğundan emin olması önemlidir.

**CUDA ÇEKİRDEKLERİ** CUDA Temelleri bölümünde belirtildiği gibi, NVIDIA GPU işlemci çekirdekleri CUDA çekirdekleri olarak da bilinir (bkz. Şekil 19.5). Daha önce de tanımlandığı ve Şekil 19.4'te görülebileceği gibi, Fermi mimarisinde her SM için ayrılmış toplam 32 CUDA çekirdeği vardır. Her CUDA çekirdeği iki ayrı işlem hattına veya veri yoluna sahiptir: bir tamsayı (INT) birimi işlem hattı ve bir kayan nokta (FP) birimi işlem hattı (bkz. Şekil 19.5). Tek bir saat periyodu sırasında bu veri yollarından yalnızca biri kullanılabilir. INT birimi, tamsayı ve mantıksal/bitsel işlemler için 32 bit, 64 bit ve genişletilmiş hassasiyet kapasitesine sahiptir. FP birimi tek hassasiyetli bir FP işlemi gerçekleştirebilirken, çift hassasiyetli bir FP işlemi iki CUDA çekirdeği gerektirir. Bu nedenle, yalnızca çift hassasiyetli FP işlemleri gerçekleştiren iş parçacıklarının çalışması, tek hassasiyetli bir FP iş parçacığına kıyasla iki kat daha uzun sürer. Çift hassasiyetli FP aritmetiğinin bu performans etkisi, Kepler mimarisinde her SM'de özel çift hassasiyetli birimlerin yanı sıra tek hassasiyetli çoğulğunun dahil edilmesiyle ele alınmaktadır. Neyse ki, iş parçacığı düzeyinde FP tek ve çift hassasiyetli işlemlerin yönetimi CUDA programcisinden gizlenmiştir. Ancak programcı, kullanılan GPU'ya bağlı olarak iki hassasiyet türünün kullanımı arasında ortaya çıkabilecek potansiyel performans etkisinin farkında olmalıdır.

Fermi mimarisi, CUDA çekirdeğinin FP birimine önceliklere göre bir iyileştirme ekledi. IEEE 754-1985 kayan noktalı aritmetik standardından IEEE 754-2008 standardına yükseltildi. Bu, çarpma-ekleme komutunun (MAD) doğruluğunu birleştirilmiş çarpma-ekleme (FMA) komutuya geliştirerek berhasilmıştır. FMA komutu hem tek hem de çift hassasiyetli aritmetik için geçerlidir. Fermi mimarisi, bir FMA komutunun sonunda yalnızca tek bir yuvarlama gerçekleştirir. Sadece sonucun doğruluğu artmakla kalmaz, aynı zamanda bir FMA komutunun gerçekleştirilmesi tek bir işlemci saat döngüsüne sıkıştırılır. Bu nedenle, SM başına tek bir işlemci saat döngüsünde 32 tek hassasiyetli veya 16 çift hassasiyetli FMA işlemi gerçekleştirilebilir.

**ÖZEL FONKSİYON BİRİMLERİ** Her SM dört SFU'ya sahiptir. SFU kosinüs, sinüs, resiprokal ve karekök gibi transandalitik işlemleri tek bir saat döngüsünde gerçekleştirir. Bir SM'de yalnızca 4 SFU ve bir warp'ta tek bir komutun 32 paralel iş parçacığı olduğundan, SFU'ları gerektiren bir warp'i tamamlamak 8 saat döngüsü alır. Bununla birlikte, CUDA işlemcileri, yükleme ve depolama birimleriyle birlikte aynı anda kullanılabilir.

**YÜKLEME VE DEPOLAMA BİRİMLERİ** SM'nin 16 yükleme ve depolama biriminin her biri saat döngüsü başına tek bir iş parçacığı için kaynak ve hedef adresleri hesaplar. Adresler, iş parçacıklarının veri yazmak veya veri okumak istediği önbellek veya DRAM .

**KAYITLAR, PAYLAŞILAN BELLEK VE L1** ÖNBELLEĞİ Şekil 19.5'te gösterildiği gibi, her SM'nin kendi (çip üzerinde) özel kayıtlar seti ve paylaşılan bellek/L1 önbellek bloğu vardır. Bu düşük gecikmeli, çip üstü belleklere ilişkin ayrıntılar ve faydalalar aşağıda açıklanmaktadır.

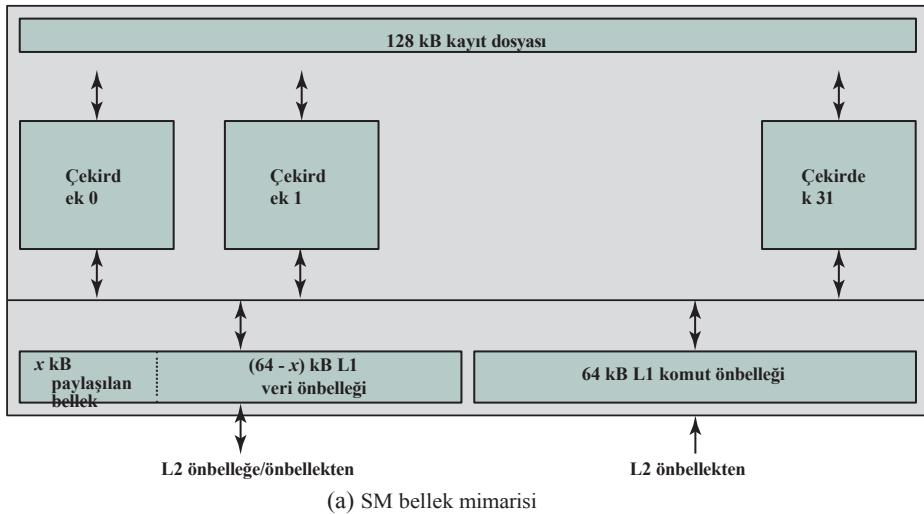
**Tablo 19.2** GPU'nun Bellek Hiyerarşisi Öz nitelikleri

Bellek Türü	Göreceli Erişim Süreleri	Erişim Türü	Kapsam	Veri Ömrü
Kayıtlar	En hızlı. Çip üzerinde	R/W	Tek iplik	Konu
Paylaşılan	Hızlı. Çip üzerinde	R/W	Bir bloktaki tüm iş parçacıkları	Blok
Yerel	100 * ila 150 * paylaşılan ve kayıttan daha yavaş. Çip dışı	R/W	Tek iplik	Konu
Küresel	100 * ila 150 * paylaşaklı ve kayıttan daha yavaş. Çip dışı.	R/W	Tüm iş parçacıkları ve ana bilgisayar	Uygulama
Sabit	100 * ila 150 * paylaşılan ve kayıttan daha yavaş. Çip dışı	R	Tüm iş parçacıkları ve ana bilgisayar	Uygulama
Doku	100 * ila 150 * paylaşılan ve kayıttan daha yavaş. Çip dışı	R	Tüm iş parçacıkları ve ana bilgisayar	Uygulama

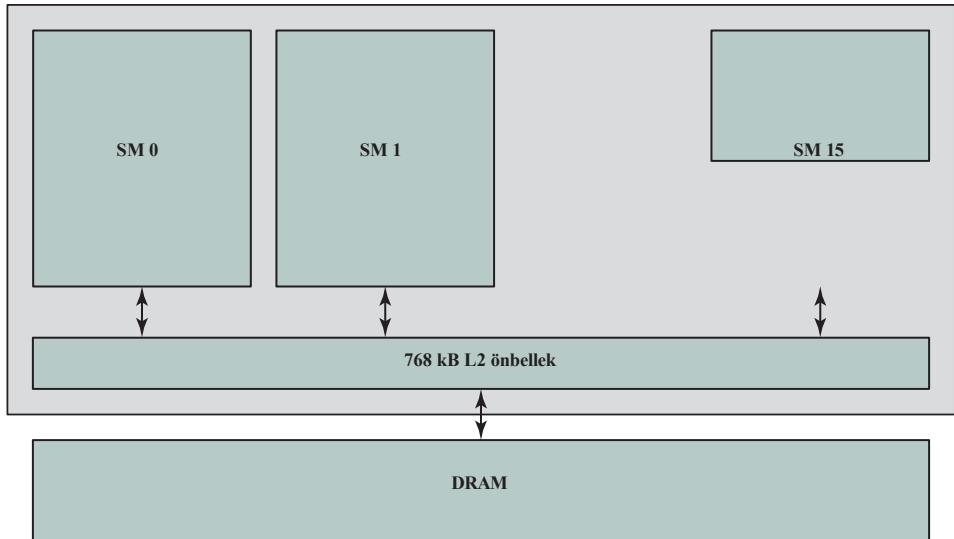
Fermi mimarisi SM başına 32k \* 32 bitlik etkileyici kayıtlara sahip olsa da, her iş parçacığı, SM başına izin verilen maksimum aktif çözgü sayısunun yanı sıra SM başına kayıt sayısının bir fonksiyonu olan CUDA hesaplama yeteneği sürüm 2.x tarafından tanımlandığı gibi kendisine tahsis edilen maksimum 64 \* 32 bitlik kayıtlara sahiptir. Tablo 19.2'de gösterildiği gibi, kayıtlar, paylaşılan bellekle birlikte, yalnızca birkaç nanosaniye (ns) ile en hızlı erişim sürelerine sahiptir. Herhangi bir geçici yazmaç dökülmesi olursa, veriler önce L1 önbelleğe taşınır, ardından L2 önbelleğe, daha sonra da uzun erişim gecikmeli yerel belleğe gönderilir (bkz. Şekil 19.7a). L1 önbelleğin kullanılması, veri okuma/yazma tehlikelerinin oluşmasını önlemeye yardımcı olur. Bu nedenle, bir iş parçacığına atanen kayıtlardaki verilerin ömrü yalnızca iş parçacığının ömrü kadardır.

Bir SM'nin GPU işlemci çekirdeklerine tahsis edilen adreslenebilir, çip üzerinde paylaşılan bellek, CPU gibi çağdaş çok çekirdeklilik mikroişlemcilerle karşılaşıldığında benzersiz bir yapılandırmadır. Bu çağdaş mimariler, Bölüm 18'de ele aldığı ve Şekil 18.6'da gösterildiği gibi, çip üzerinde özel bir L1 önbelleğe ve çekirdek başına bir dizi kayıt cihazı sahiptir. Ancak, tipik olarak çip üzerinde adreslenebilir belleğe sahip değildirler. Bunun yerine, özel bellek yönetim donanımı, programının kontrolü olmadan önbellek ve ana bellek arasındaki veri hareketini düzenler. Bu GPU mimarisinden önemli ölçüde farklıdır (bkz. Şekil 19.5).

Bu bölümün başında tartışıldığı gibi, paylaşılan bellek GPU mimarisine özellikle GPGPU uygulamalarına yardımcı olmak için . Paylaşılan bellek kullanımının optimize edilmesi, çip dışı belleğe gereksiz uzun gecikmeli erişimleri ortadan kaldırarak bir GPGPU uygulamasının hızını ve performansını önemli ölçüde artırabilir. PaylaGilan belleğin boyutu her SM için küçük olmasına rağmen (maksimum yapılandırmada 48 kB), global bellekten 100 \* ila 150 \* daha az olmak üzere çok güçlü bir erişim gecikmesine sahiptir (bkz. Tablo 19.2). Dolayısıyla, paylaşılan belleğin paralel işlem görevlerini hızlandırmasının üç ana yolu vardır: (1) bir bloğun tüm iş parçacıkları tarafından paylaşılan bellek verilerinin çoklu tekrarlı kullanımı (örneğin, matris matris çarpımı için kullanılan veri blokları); (2) bir bloğun belirli iş parçacıkları (belirli kimliklere dayalı olarak) verileri global bellekten paylaşılan aktarmak için kullanılır, böylece gereksiz



(a) SM bellek mimarisi



(b) Genel bellek mimarisi

**Sekil 19.7** Fermi Bellek Mimarisi

Aynı bellek konumlarına yapılan okumalar ve yazmalar kaldırılır; ve (3) kullanıcı, mümkün olduğunda erişimlerin birleştirildiğinden emin olarak global belleğe veri erişimlerini optimize edebilir. Tüm bu noktalar aynı zamanda çip dışı bellek bant genişliği kısıtlaması sorunlarının azaltılmasına da yardımcı olur. Bir SM'nin paylaşılan belleğindeki verilerin ömrü, işlenen iş parçacığı bloğunun ömrü kadardır. Dolayısıyla, bloktaki tüm iş parçacıkları tamamlandıında, SM'nin paylaşılan belleğindeki veriler artık geçerli değildir.

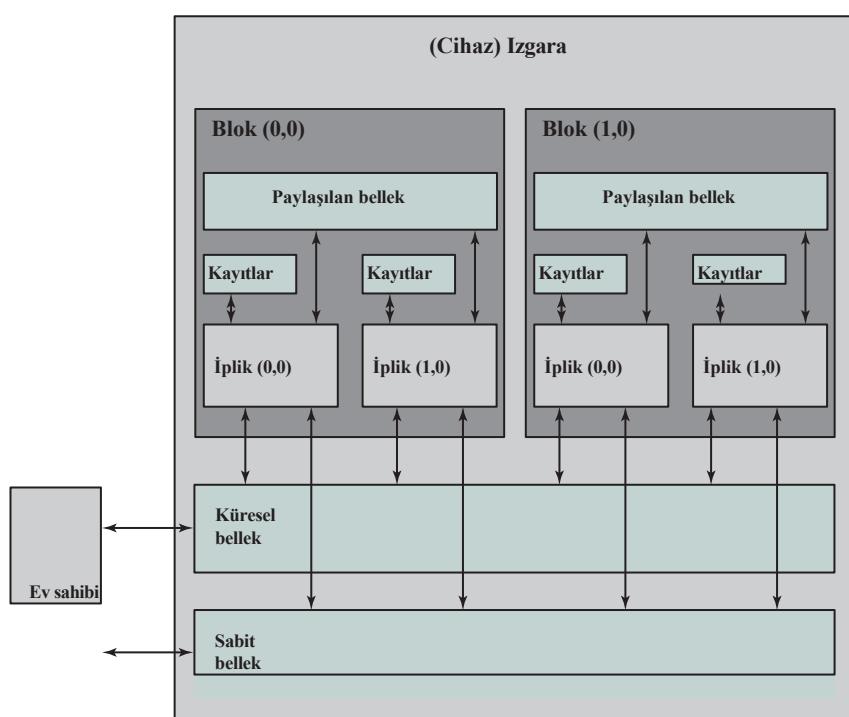
Paylaşılan bellek kullanımı optimum çalışma süreleri sağlayacak olsa da, bazı uygulamalarda bellek erişimleri programlama aşamasında bilinmemektedir. İşte bu noktada daha fazla L1 önbelleğe sahip olmak (maksimum 48 kB ayarı) en iyi sonuçları verecektir. Ek olarak, L1 önbelleği kayıt dökülmelerine yardımcı olur,

doğrudan yerel (çip dışı) DRAM belleğe gitmek yerine. İki seviyeli önbellek hiyerarşisi - SM başına tek L1 önbellek ve çip boyunca SM paylaşımı L2 önbellek - geleneksel çok çekirdeklilik mikroişlemcilerde bulunanlarla aynı avantajları sağlar.

### Hafıza Türlerini Bilmenin ve Programlamanın Önemi

CUDA kullanarak doğru ve verimli kod geliştirmeyi sağlamak için programcının çeşitli GPU belleklerinin nüanslarını, özellikle de her bellek türü için mevcut boyutları, göreceli erişim sürelerini ve erişilebilirlik sınırlamalarını anlaması önemlidir. Bölümün başında ele alınan CUDA Temelleri bölümünden, az önce ele alınan SM seviyesi belleklerden ve Tablo 19.2'de listelenen terminoloji ve parametrelerden de görülebileceği gibi, GPGPU programlama için, kullanılan belirli veri depolama donanımının (dosya I/O dışında) programcının gizlendiği bir CPU için hedeflenen program geliştirmeden çok daha farklı bir yaklaşım gereklidir.

Örneğin, GPU mimarisinde, bir CUDA çekirdeğine atanmış her iş parçacığı kendi yazmaç kümese sahiptir, öyle ki bir iş parçacığı aynı SM'de olsun ya da olmasın başka bir iş parçacığının yazmaçlarına erişemez. Belirli bir SM içindeki iş parçacıklarının birbirleriyle işbirliği yapabilmesinin (veri paylaşımı yoluyla) tek yolu paylaşılan bellektir (bkz. Şekil 19.8). Bu genellikle programcının bir SM'nin yalnızca belirli iş parçacıklarını paylaşılan belleğin belirli konumlarına yazmaya atamasıyla gerçekleştirilir, böylece yazma tehlikeleri veya boş harcanan döngüler (örneğin, birçok iş parçacığının aynı veriyi okuması) önlenir.



**Şekil 19.8** Bir GPU'nun Temel Mimarisinin CUDA Gösterimi

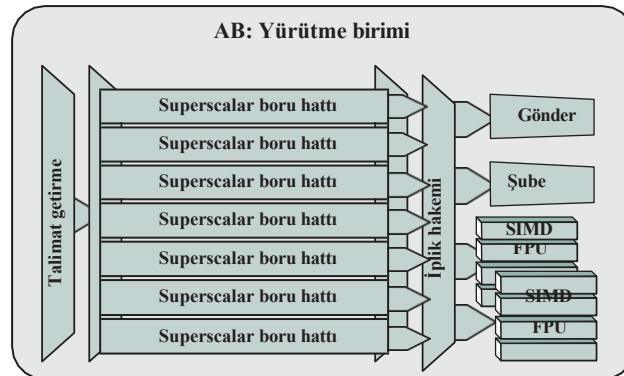
global bellekten alıp aynı paylaşılan bellek adresine yazmak). Belirli bir SM'nin tüm iş parçacıklarının henüz yazılmış olan paylaşılan bellekten okumasına izin verilmeden önce, yazma sonrası okuma (RAW) veri tehlikesini önlemek için o SM'nin tüm iş parçacıklarının senkronizasyonunun gerçekleşmesi gereklidir.<sup>1</sup>

## 19.4 INTEL'İN GEN8 GPU'SU

Bir başka GPGPU mimarisinin örneği olarak, bu bölümde Gen8 işlemci grafik mimarisine genel bir bakış sunulmaktadır [INTE14, PEDD14].

Gen8 mimarisinin temel yapı taşı Şekil 19.9'da gösterilen yürütme birimidir (EU). EU, yedi iş parçacıklı bir eşzamanlı çoklu iş parçacıklı (SMT) mimaridir. Bölüm 17 ve 18'den hatırlanacağı üzere, SMT mimarisinde kayıt bankaları genişletilir, böylece birden fazla iş parçacığı boru hattı kaynaklarının kullanımını paylaşabilir. EU yedi iş parçacığına sahiptir ve süper skaler bir boru hattı mimarisi olarak uygulanmaktadır. Her iş parçacığı 128 genel amaçlı yazmaç içerir. Her bir EU içinde, birincil hesaplama birimleri hem kayan nokta hem de tamsayı hesaplamasını destekleyen iki SIMD kayan nokta birimidir. Her SIMD FPU, her döngüde eşzamanlı olarak kayan nokta ekleme ve çarpma komutlarını tamamlayabilir. Ayrıca dallanma talimatlarına adanmış bir dallanma birimi ve bellek işlemleri için bir gönderme birimi vardır.

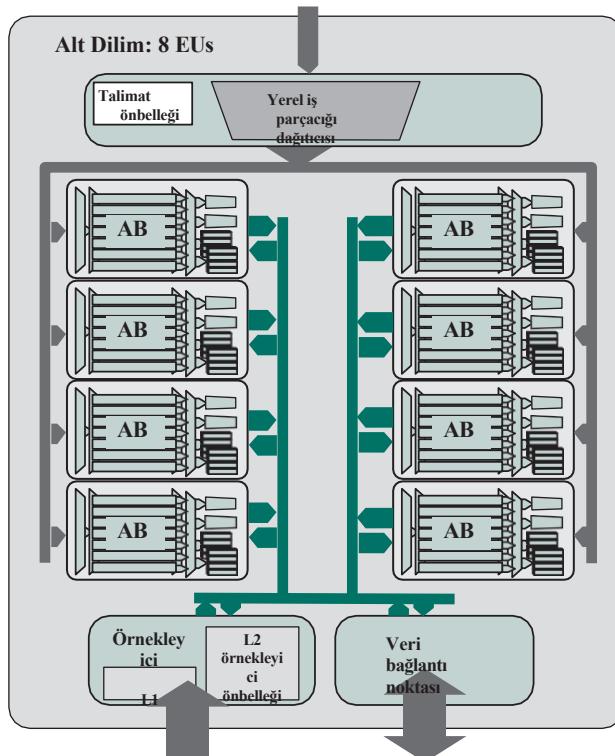
Her bir kayıt 32 bayt saklar ve 32 bit veri elemanlarından oluşan 8 elemanlı bir SIMD vektörü olarak erişilebilir. Böylece her Gen8 iş parçacığı, AB başına toplam 28 kB GRF için 4 kB genel amaçlı kayıt dosyasına (GRF) sahiptir. Esnek adresleme modları, etkin bir şekilde daha geniş kayıtlar oluşturmak veya hatta kademeli dikdörtgen blok veri yapılarını temsil etmek için kayıtların birlikte adreslenmesine izin verir.<sup>2</sup> İş parçacığı başına mimari durum, ayrı bir özel mimari kayıt dosyasında (ARF) tutulur.



**Şekil 19.9** Intel Gen8 Yürütme Birimi

<sup>1</sup>RAW tehlikelerinin tartışılmaması için Bölüm 16'ya bakın.

<sup>2</sup> Strided terimi, her biri bir öncekinden *stride uzunluğu* adı verilen sabit bir aralıkla ayrılan adreslere bir dizi bellek okuması ve yazması anlamına gelir. Dizili referanslar genellikle bir dizideki döngüler tarafından oluşturulur ve (veriler erişim süresinin önemli olduğu kadar büyükse) çift döngüler ters çevrerek veya bir döngü yuvasının dış döngüsünü kısmen açarak daha iyi yerellik için ayarlama yapmak faydalı olabilir.



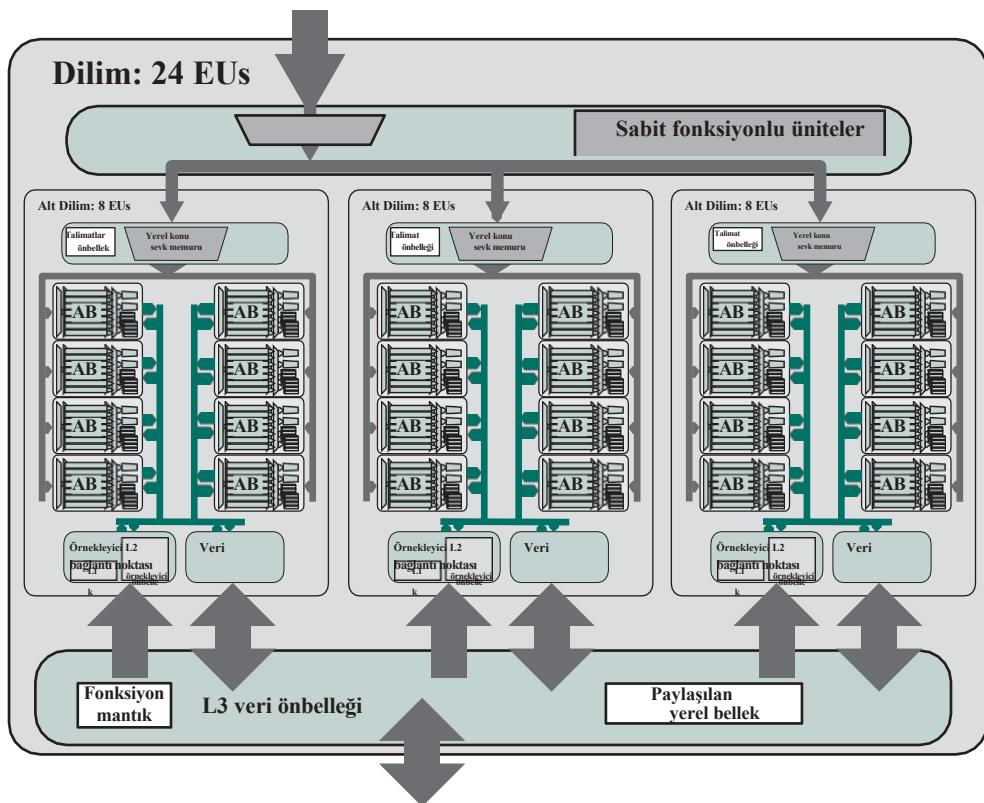
Şekil 19.10 Intel Gen8 Alt Dilimi

AB, farklı iş parçacıklarından aynı anda dört adede kadar farklı talimat verebilir. Thread arbiter her bir talimatı yürütülmESİ için dört fonksiyonel birimden birine gönderir.

EU'lar, sekiz adede kadar içerebilen bir alt dilimde (Şekil 19.10) düzenlenir. Her alt dilim kendi yerel iş parçacığı gönderme birimini ve kendi destekleyici komut önbelleklerini içerir. Böylece, tek bir alt dilimde toplam 56 eşzamanlı iş parçacığı için özel donanım kaynakları ve kayıt dosyaları bulunur.

Bir alt dilim ayrıca kendi yerel L1 ve L2 önbelleğine sahip örnekleyici adı verilen bir birim içerir. Örnekleyici, doku ve görüntü yüzeylerini örneklemek için kullanılır. Örnekleyici, blok sıkıştırma doku formatlarının dinamik dekompresyonunu desteklemek için mantık içerir. Örnekleyici ayrıca görüntü  $(u,v)$  koordinatlarının adres dönüşümünü ve ayna, sarma, kenarlık ve kelepçe gibi adres sıkıştırma modlarını sağlayan sabit işlevli mantık içerir. Örnekleyici, nokta, bilinear, trilinear ve anizotropik gibi çeşitli örneklemeye modlarını destekler. Veri portu, farklı iş parçacıklarından okuma işlemlerini birleştirmek için önbellek satır boyutundan yararlanmaya çalışan verimli okuma / yazma işlemleri sağlar.

Ürün varyantları oluşturmak için alt dilimler, dilim adı verilen gruplar halinde kümelenebilir (Şekil 19.11). Şu anda, toplam 24 EU için üç adede kadar alt dilim tek bir dilim halinde düzenlenlenebilir. Alt dilimlere ek olarak, dilim, iş parçacığı gönderme yönlendirmesi için mantık, grafik veri işlemeyi optimize etmek için diğer işlev mantığı, paylaşılan bir



**Şekil 19.11** Intel Gen8 Dilimi

L3 önbellek ve daha küçük bir paylaşilan yerel bellek yapısı. İkincisi EU'lar tarafından görülebilir (abellek) ve geçici değişkenleri paylaşmak için kullanılmıştır.

Performansı artırmak için, paylaşilan L3 veri önbelleği için **önbellek bankacılığı** olarak bilinen bir teknik kullanılır. Yüksek bant genişliği elde etmek için önbellek, aynı anda erişilebilen ve banka adı verilen eşit boyutlu bellek modüllerine ayrılr. Bu nedenle,  $n$  farklı bellek bankasına düşen  $n$  adreslenen yapılan herhangi bir bellek okuma veya yazma isteği, tek bir modülün bant genişliğinden  $n$  kat daha yüksek bir genel bant genişliği sağlayarak eşzamanlı olarak sunulabilir. Bununla birlikte, bir bellek isteğininki iki adresi aynı bellek bankasına düşerse, bir banka çakışması olur ve erişimin serileştirilmesi gereklidir. Donanım, banka çakışması olan bir bellek talebini gerektiği kadar çakışmasız ayrı taleplerin sayısına eşit bir faktörle azaltır. Aynı bellek isteklerinin sayısı  $n$  ise, ilk bellek isteğininki  $n$  yönlü banka çakışmalarına neden olduğu söylenir. Bu nedenle, maksimum performans elde etmek için, bellek adreslerinin bellek bankalarına nasıl eşlendiğini anlamak, bellek isteklerini banka çakışmalarını en aza indirecek şekilde planlamak için önemlidir.

Son olarak, bir SoC ürün mimarı, bir SoC yongasına tek bir dilim veya birden fazla dilim yerleştirerek ürün aileleri veya bir aile içinde belirli bir ürün oluşturabilir. Bu dilimler, komutu yönetmek için ek ön uç mantığı ile birleştirilir.

gonderiminin yanı sıra 3D render ve medya boru hatlarını desteklemek için sabit işlevli mantık. Ayrıca, Gen8 bilgi işlem mimarisinin tamamı, grafik teknolojisi arayüzü (GTI) adı verilen özel bir birim aracılığıyla SoC bileşenlerinin geri kalımıyla arayüz oluşturuyor.

Böyle bir SoC'ye örnek olarak Intel HD Graphics 5300 Gen8'li Intel Core M İşlemci verilebilir (Şekil 19.12). GPU kısmına ek olarak, çip birden fazla CPU çekirdeği, bir LLC önbelleği ve bir sistem aracı içerir. Sistem aracı DRAM bellek, ekran ve PCIe aygıtları için denetleyiciler içerir. Processor Graphics Gen8, CPU'lar, LLC önbellek ve sistem aracı, Xeon işlemci için gördüğümüz gibi bir halka yapısıyla birbirine bağlanır (Şekil 7.16).

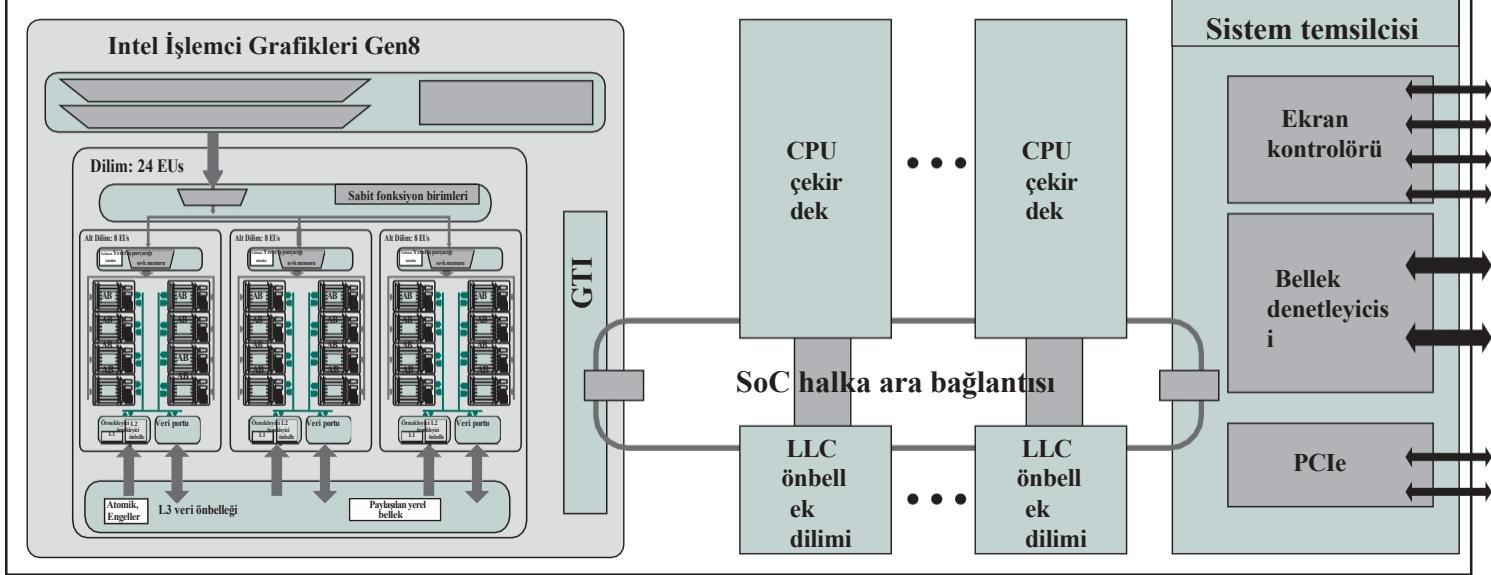
## 19.5 BİR GPU NE ZAMAN YARDIMCI İŞLEMCI OLARAK KULLANILIR

Bu bölüm, yazılım tasarımi perspektifinden aday GPGPU uygulamalarının belirlenmesi ve bu sürece yardımcı olacak bazı ilgili yazılım araçları hakkında kısa bir tartışma ile bitriyoruz.

Kodunun bir kısmını GPU'da (dolayısıyla heterojen bir bilgi işlem platformunda) çalıştırılmaktan fayda sağlayacak bir programla sağlamayacak bir programı birbirinden ayıran nedir? Bu bölümde gösterildiği ve tartışıldığı gibi, GPU yüzlerce ila binlerce işlemci çekirdeğinden oluşur ve bir SIMD mimarisine sahiptir. Bu nedenle, büyük veri kümeleri üzerinde eşzamanlı olarak çalışmak için binlerce hafif iş parçacığına çoğaltılabilecek yüksek oranda paralelleştirilebilir kod bölmüllerine sahip programlar, bir GPGPU sisteminde çalışma sürelerini hızlandırmak için en iyi adaylardır. Burada hafif iş parçacığı, dallanma içermeyen veya çok az dallanma içeren nispeten küçük, büyük ölçüde paralelleştirilebilir bir kod parçacığının örneği olarak tanımlanmaktadır. Tipik olarak, orijinal seri kod, iterasyonlar arasında veri bağımlılığı olmayan denklemler (örneğin matris aritmetiği) üzerinde hesaplamalar yapan büyük bir iterasyon for döngüsü veya birkaç gömülü for döngüsü şeklinde dir. Ayrıca, GNU komut satırı tabanlı gprof veya NVIDIA'nın nvprof görsel tabanlı profilleyicisine benzer araçlarla (her iki profilleyici de tercihen tipik temsili verilerle çalıştırılır) tüm programın başlangıçta profili çıkarılırken, paralelleştirilecek bölüm(ler) programın toplam çalışma süresinin makul bir yüzdesini oluşturmalıdır. Bu gereklilik hem elde edilebilecek hızlanmayı en üst düzeye çıkaracak (Amdahl yasası) hem de CPU ile GPU arasındaki veri aktarım süresinin genel hızlanma üzerindeki etkisini en aza indirecektir.

Büyük ölçüde paralelleştirilebilir bir aday kod segmenti belirlendikten sonra, seri koddan paralel koda veya bir CUDA çekirdeğine dönüştürülmesi gereklidir. Bu dönüştürme işlemini kullanıcından girdi almadan otomatik olarak yapabilen ve aynı zamanda optimuma yakın, doğru bir çözüm sunan bir paralelleştirme derleyicisi mevcut olsaydı, bu büyük ölçüde zaman, para ve emek tasarrufu sağlardı. Ne yazık ki böyle bir araç henüz mevcut değildir. Bu durumda geriye iki seçenek kalmıştır: (1) CUDA, OpenCL veya benzerlerinde karmaşık planlama ve programlama yoluyla kodu dönüştürmek; veya (2) OpenACC, hiCUDA veya benzeri bir derleyici yönerge dili kullanmak. Derleyici için koda paralelleştirici "ipuçları" yerleştirmek üzere bir derleyici yönerge dili kullanmak programlama süresinden büyük ölçüde tasarruf sağlayabilecektir, bu yine de yinelemeli bir süreçtir ve elde edilen optimum çalışma süresi garanti edilmez. Bununla birlikte, bu yöntem

## Intel Core M İşlemci



Şekil 19.12 Intel Core M İşlemci SoC

## **706 BÖLÜM 19 / EEEAAL-PURPpSE RAPHIC PRpCESSIe UeITS**

Son birkaç yıldır artan ilgi ve CUDA derleyicisinin yeni sürümleri OpenACC dilini desteklemektedir. Yine de iyi planlanmış/mühendislik yapılmış ve kodlanmış bir CUDA programı neredeyse her zaman bugüne kadarki en iyi çalışma sürelerini verecektir.

### **19.6 ANAHTAR TERİMLER VE İNCELEME SORULARI**

#### **Anahtar Terimler**

blok önbellek bankacılığı merkezi işlem birimi (CPU) Birleşik Hesaplama Cihazı Mimari (CUDA) CUDA çekirdeği	GPU (GPGPU) kullanarak genel amaçlı hesaplama GPU işlemci çekirdeği grafik işleme birimi (GPU) izgara	çekirdek akışlı çoklu işlemciler (SM'ler) iplik iplik blok çözümü
--	---	--

#### **İnceleme Soruları**

- 19.1** CUDA'yı tanımlayın.
- 19.2** CPU ve GPU mimarileri arasındaki temel farkları listeleyiniz.
- 19.3** Çekirdek, iş parçacığı ve blok arasındaki farklar nelerdir?
- 19.4** Warp'ı tanımla.
- 19.5** Özel fonksiyon birimleri nedir?

## **KONTROL ÜNITESİ ÇALIŞMASI**

### **20.1 Mikro Operasyonlar**

- Getirme Döngüsü
- Dolaylı Döngü Kesme
- Döngüsü Yürütme
- Döngüsü
- Talimat Döngüsü

### **20.2 İşleminin Kontrolü**

- Fonksiyonel Gereksinimler Kontrol
- Sinyalleri
- Bir Kontrol Sinyalleri Örneği Dahili  
İşlemci Organizasyonu Intel 8085

### **20.3 Kablolu Uygulama**

- Kontrol Ünitesi
- Girişleri Kontrol  
Ünitesi Mantığı

### **20.4 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler**

## **708 BÖLÜM 20 / KONTROL ÜNİTESİNİN ÇALIŞTIRILMASI**

### **ÖĞRENİM HEDEFLERİ**

Bu bölümü çalıştıktan sonra şunları yapabilmelisiniz:

- Mikro-işlemler kavramını açıklamak ve mikro-işlemler açısından temel komut döngüsü aşamalarını tanımlamak.
- Bir işlemciyi kontrol etmek için mikro işlemlerin nasıl organize edildiğini tartışınız.
- Kablolu kontrol ünitesi organizasyonunu anlamak.

Bölüm 12'de, makine komut setinin işlemciyi tanımlamak için uzun bir yol kat ettiğini belirtmiştık. Her bir işlem kodunun etkisinin anlaşılması ve adresleme modlarının anlaşılması da dahil olmak üzere makine komut setini biliyorsak ve kullanıcı tarafından görülebilen yazmaçlar setini biliyorsak, o zaman işlemcinin yerine getirmesi gereken işlevleri de biliyoruz demektir. Bu resmin tamamı değildir. Genellikle bir veri yolu aracılığıyla harici arayüzleri ve kesmelerin nasıl ele alındığını bilmemiz gerekir. Bu mantık ile, bir işlemcinin işlevini belirlemek için gerekenlerin aşağıdaki listesi ortaya çıkar:

- 1. İşlemler (opcodes)**
- 2. Adresleme modları**
- 3. Kayıtlar**
- 4. I/O modülü arayüzü**
- 5. Bellek modülü arayüzü**
- 6. Kesintiler**

Bu liste genel olmakla birlikte oldukça eksiksizdir. 1'den 3'e kadar olan maddeler komut seti tarafından tanımlanır. Madde 4 ve 5 tipik olarak sistem veri yolu belirtilerek tanımlanır. Madde 6 kısmen sistem ve kısmen de işlemcinin işletim sistemine sunduğu destek türü tarafından tanımlanır.

Altı maddeden oluşan bu liste, bir işlemci için işlevsel gereksinimler olarak adlandırılabilir. Bunlar bir işlemcinin ne yapması gerektiğini belirler. İkinci ve Dördüncü Bölümlerde bizi meşgul eden de buydu. Şimdi, bu işlevlerin nasıl yerine getirildiği ya da daha spesifik olarak, bu işlevleri sağlamak için işlemcinin çeşitli unsurlarının nasıl kontrol edildiği sorusuna dönüyoruz. Böylece, işlemcinin çalışmasını kontrol eden kontrol ünitesinin tartışılmasına geçiyoruz.

### **20.1 MIKRO-OPERASYONLAR**

Bir bilgisayarnın bir programı yürütürken, her döngüde bir makine komutu olmak üzere bir dizi komut döngüsünden oluştugunu gördük. Elbette, bu komut döngülerini dizisinin, dallanma talimatlarının varlığı nedeniyle, programı oluşturan yazılı talimat *dizisiyle* aynı olması gerekmediğini unutmamalıyız. Burada sözünü ettigimiz şey, talimatların yürütme zamanı sırasıdır.

Ayrıca her komut döngüsünün bir dizi küçük birimden oluştuğunu gördük. Uygun bulduğumuz bir alt bölüm, her zaman yalnızca getirme ve yürütme döngülerinin gerçekleştiği getirme, dolaylı, yürütme ve kesme şeklindedir.

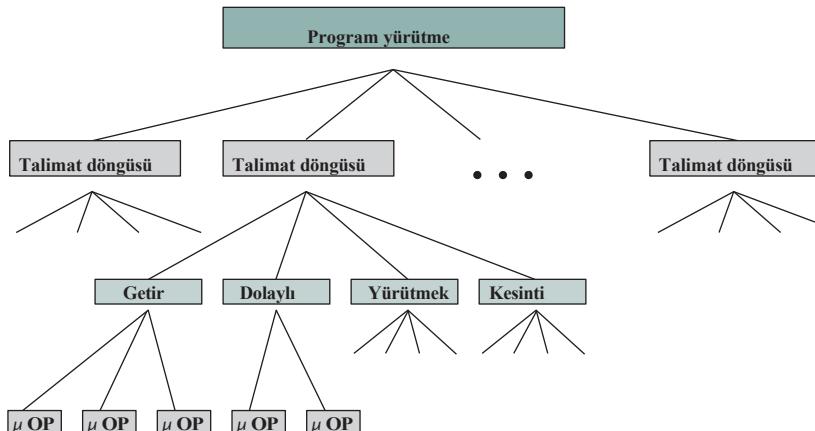
Ancak bir kontrol ünitesi tasarlamak için tanımlamayı daha da parçalara ayırmamız gereklidir. Bölüm 14'teki boru hattı tartışmamızda, daha ileri bir ayırtmanın mümkün olduğunu görmeye başladık. Aslında, daha küçük döngülerin her birinin, her biri işlemci kayıtlarını içeren bir dizi adım içerdiğini göreceğiz. Bu adımları **mikro-işlemler** olarak adlandıracagız. Mikro öneki, her adının çok basit olduğu ve çok az şey gerçeklestirdiği gerektiğini ifade eder. Şekil 20.1, tartışmaka olduğumuz çeşitli kavramlar arasındaki ilişkiyi göstermektedir. Özettelemek gerekirse, bir programın yürütülmesi talimatların sırayla yürütülmesinden oluşur. Her komut, daha kısa alt döngülerden (örneğin, getirme, dolaylı, yürütme, kesme) oluşan bir komut döngüsü sırasında yürütülür. Her bir alt döngünün yürütülmesi bir veya daha fazla kısa işlemi, yani mikro işlemleri içerir.

Mikro-işlemler bir işlemcinin işlevsel ya da atomik işlemleridir. Bu bölümde, herhangi bir komut döngüsündeki olayların bu tür mikro-işlemlerin bir dizisi olarak nasıl tanımlanabileceğini anlamak için mikro-işlemleri inceleyeceğiz. Basit bir örnek kullanılacaktır. Bu bölümün geri kalanında, mikro-işlemler kavramının kontrol ünitesinin tasarımında nasıl bir rehber görevi gördüğünü göstereceğiz.

## Getirme Döngüsü

Her talimat döngüsünün başında gerçekleşen ve bir komutun bellekten alınmasına neden olan getirme döngüsüne bakarak başlayacağız. Tartışma amacıyla, Şekil 14.6'da gösterilen organizasyonu varsayıyoruz (*Veri Akışı, Getirme Döngüsü*). Dört yazmaç söz konusudur:

- **Bellek adres kaydı (MAR):** Sistem veriyolunun adres hatlarına bağlıdır. Bir okuma veya yazma işlemi için bellekteki adresi belirtir.
- **Bellek tampon kaydı (MBR):** Sistem veriyolunun veri hatlarına bağlıdır. Bellekte depolanacak değeri veya bellekten okunan son değeri içerir.



**Şekil 20.1** Bir Program Yürütmesinin Kurucu Unsurları

## 710 BÖLÜM 20 / KONTROL ÜNİTESİNİN ÇALIŞTIRILMASI

- **Program sayacı (PC):** Getirilecek bir sonraki komutun adresini tutar.
- **Talimat kaydı (IR):** Getirilen son talimatı tutar.

Getirme döngüsü için olayların sırasına işlemci yazmaçları üzerindeki etkisi açısından bakalım. Şekil 20.2'de bir örnek görülmektedir. Getirme döngüsünün başında, yürütülecek bir sonraki komutun adresi program sayacındadır (PC); bu durumda adres 1100100'dür. İlk adım, bu adresi bellek adres kaydına (MAR) taşımaktır çünkü bu, sistem veriyolunun adres hatlarına bağlı tek kayittir. İkinci adım, komutu getirmektir. İstenen adres (MAR'da) adres veriyoluna yerleştirilir, kontrol birimi kontrol veriyolunda bir READ komutu verir ve sonuç veri veriyolunda görünür ve bellek tampon yazmacına (MBR) kopyalanır. Ayrıca bir sonraki komuta hazırlanmak için PC'yi komut uzunluğu kadar artırmanız gereklidir. Bu iki işlem (bellekten kelime okuma, PC'yi artırma) birbiriyle etkileşime girmemişinden, zaman kazanmak için bunları aynı anda yapabiliriz. Üçüncü adım MBR'nin içeriğini komut kaydına (IR) taşımaktır. Bu, MBR'yi olası bir dolaylı döngü sırasında kullanılmak üzere serbest bırakır.

Bu nedenle, basit getirme döngüsü aslında üç adım ve dört mikro işleminden oluşur. Her mikro-islem verinin bir yazmacın içine veya dışına hareketini içerir. Bu hareketler birbirini engellememiş sürece, birkaçı bir adımda gerçekleştirilebilir ve zamanandan tasarruf sağlar. Sembolik olarak, bu olay dizisini aşağıdaki gibi yazabiliriz:

$t_1$ : MAR **d** (PC (   
 $t_2$ : MBR **d** Bellek PC  
**d** (PC (+ *I*  
 $t_3$ : IR **d** (MBR (

Burada *I* komut uzunluğudur. Bu dizi hakkında birkaç yorum yapmamız gerekiyor. Zamanlama amaçları için bir saatin mevcut olduğunu ve düzenli aralıklarla saat darbeleri yaydığını varsayıyoruz. Her saat darbesi bir zaman birimi tanımlar. Böylece, tüm zaman birimleri

tMAR		MAR	0000000001100100
MBR		MBR	
PC IR	0000000001100100	PC IR	0000000001100100
AC		AC	

(a) Başlangıç ( $t_1$ den önce)
(b) İlk adımdan sonra

MAR	0000000001100100	MAR	0000000001100100
MBR		MBR	
PC IR	0000000001100100	PC IR	0000000001100100
AC	0001000000100000	AC	0001000000100000
	0000000001100101		0000000001100101
			0001000000100000

(c) İkinci adımdan sonra
(d) Üçüncü adımdan sonra

**Şekil 20.2** Olay Sırası, Getirme Döngüsü

şet sürelidir. Her bir mikro-islem tek bir zaman birimi içerisinde gerçekleştirilebilir. ( $t_1$ ,  $t_2$ ,  $t_3$ ) gösterimi ardışık zaman birimlerini temsil etmektedir. Başka bir deyişle

- **İlk sefer birimi:** PC içeriğini MAR'a taşıyın.
  - **İkinci zaman birimi:** MAR tarafından belirtilen bellek konumunun içeriğini MBR'ye taşı. PC'nin içeriğini *I* kadar artırın.
  - **Üçüncü zaman birimi:** MBR içeriğini IR'ye taşıyın.

İkinci ve üçüncü mikro işlemlerin her ikisinin de ikinci zaman birimi sırasında gerçekleş diligine dikkat edin. Üçüncü , getirme işlemini etkilemeden dördüncü ile gruplandırılabilir:

t<sub>1</sub>: MAR d (PC)  
t<sub>2</sub>: MBR d Bellek t<sub>3</sub>:  
PC d (PC(+ I  
TR d (MBR (

Mikro operasyonlarının gruplandırılması iki basit kurala uymalıdır:

- Uygun olay sırası takip edilmelidir. Bu nedenle (MAR d (PC)), (MBR d Bellek)'ten önce gelmelidir çünkü bellek okuma işlemi MAR'daki adresi kullanır.
  - Çakışmalardan kaçınılmalıdır. Bir zaman biriminde aynı yazmacın okunması ve yazılması denenmemelidir, çünkü sonuçlar tahmin edilemez olacaktır. Örneğin, (MBR d Memory) ve (IR d MBR) mikro-islemleri aynı zaman biriminde gerçekleşmemelidir.

Dikkat edilmesi gereken son bir nokta da mikro işlemlerden birinin toplama işlemi içermesidir. Devrelerin tekrarlanması önlemek için bu toplama işlemi tarafından gerçekleştirilebilir. ALU'nun kullanımı, ALU'nun işlevsellüğüne ve işlemcinin organizasyonuna bağlı olarak ek mikro işlemler içerebilir. Bu konunun tartışılmasını bu bölümün ilerleyen kısımlarına erteliyoruz.

Bu ve bundan sonraki alt bölümlerde açıklanan olayları *Şekil 3.5 (Program Yürütme Örneği)* ile karşılaştırımkarşılık faydalı olacaktır. Bu şekilde mikro işlemler göz ardi edilirken, bu tartışma komut döngüsünün alt döngülerini gerçekleştirmek için gereken mikro işlemleri göstermektedir.

### Dolaylı Döngü

Bir komut getirildikten sonra, bir sonraki adım kaynak işlenenlerin getirilmesidir. Basit örneğimizde devam ederek, doğrudan ve dolaylı adreslemeye izin verilen tek bir komut formatı varsayılmıştır. Komut dolaylı bir adres belirtiyorsa, yürütme döngüsünden önce bir dolaylı döngü gelmelidir. Veri akışı Şekil 14.7'de (*Veri Akışı, Dolaylı Cevrim*) gösterileninden biraz farklıdır ve aşağıdaki mikro-islemleri içerir:

$t_1: MAR \text{ } d \text{ (IR(Adres)) } t_2: MBR$   
**d** Bellek  
 $t_3: IR(Adres) \text{ } d \text{ (MBR(Adres)) }$

Komutun adres alanı MAR'a aktarılır. Bu daha sonra işlenenin adresini almak için kullanılır. Son olarak, IR'nin adres alanı MBR'den güncellenir, böylece artık dolaylı bir adres yerine doğrudan bir adres içerir.

## 712 BÖLÜM 20 / KONTROL ÜNİTESİNİN ÇALIŞTIRILMASI

IR artık dolaylı adresleme kullanılmamış gibi aynı durumdadır ve yürütme döngüsü için hazırda. Kesme döngüsünü değerlendirmek için bir an bu döngüyü atlıyoruz.

### Kesme Döngüsü

Yürütme döngüsü tamamlandığında, etkinleştirilmiş herhangi bir kesmenin oluşup oluşmadığını belirlemek için bir test yapılır. Eğer öyleyse, kesme döngüsü gerçekleşir. Bu döngünün doğası bir makineden diğerine büyük farklılıklar gösterir. Şekil 14.8'de (*Veri Akıtı, Dolaylı Döngü*) gösterildiği gibi çok basit bir olaylar dizisi sunuyoruz. Sahip olduğumuz

$t_1: MBR \rightarrow (PC)$   
 $t_2: MAR \rightarrow \begin{array}{l} \text{Kaydet_Adres PC} \\ \text{d Rutin_Adres} \end{array}$   
 $t_3: Bellek \rightarrow \begin{array}{l} \text{d (MBR (}} \end{array}$

İlk adımda, PC'nin içeriği MBR'ye aktarılır, böylece kesmeden dönüş için kaydedilebilirler. Daha sonra MAR, PC içeriğinin kaydedileceği adresle yüklenir ve PC, kesme işleme rutininin başlangıç adresiyle yüklenir. Bu iki işlemin her biri tek bir mikro işlem olabilir. Ancak, çoğu işlemci birden fazla kesme türü ve/veya seviyesi sağladığından, sırasıyla MAR ve PC'ye aktarılmadan önce Save\_Address ve Routine\_Address'i elde etmek için bir veya daha fazla ek mikro işlem gerekebilir. Her durumda, bu yapıldıktan sonra, son adım PC'nin eski değerini içeren MBR'yi belleğe kaydetmektir. İşlemci artık bir sonraki komut döngüsüne başlamaya hazırlıdır.

### Yürütme Döngüsü

Getirme, dolaylı ve kesme döngülerini basit ve tahmin edilebilirdir. Her biri küçük, sabit bir mikro işlem dizisi içerir ve her durumda aynı mikro işlemler her seferinde tekrarlanır.

Bu durum yürütme döngüsü için geçerli değildir. İşlem kodlarının çeşitliliği nedeniyle, gerçekleştirebilecek çok sayıda farklı mikro işlem dizisi vardır. Kontrol birimi işlem kodunu inceler ve işlem kodunun değerine bağlı olarak bir dizi mikro işlem oluşturur. Bu, komut kod çözme olarak adlandırılır.

Birkaç varsayımsal örneği ele alalım. İlk olarak, bir toplama komutunu ele alalım:

ADD R1, X

X konumunun içeriğini R1 kaydına ekler. Aşağıdaki mikro-işlemler dizisi gerçekleştirilebilir:

$t_1: MAR \rightarrow \begin{array}{l} \text{d (IR(adres (}} \\ \text{MBR \rightarrow \begin{array}{l} \text{d Bellek} \\ \text{t_2:} \end{array} \end{array}$   
 $t_3: R1 \rightarrow \begin{array}{l} \text{d (R1 (+ (MBR (}} \end{array}$

ADD komutunu içeren IR ile başlarız. İlk adımda, IR'nin adres kısmı MAR'a yüklenir. Ardından başvurulan bellek konumu okunur. Son olarak, R1 ve MBR'nin içeriği ALU tarafından toplanır. Yine, bu basitleştirilmiş bir öрnekdir. Ek mikro işlemler çıkarmak için gerekli olabilir

IR'den kayıt referansı ve belki de ALU girişlerini veya çıkışlarını bazı ara kayıtlara yerleştirmek için.

Şimdi daha karmaşık iki örneğe bakalım. Yaygın bir komut, sıfır ise artır ve atla şeklindedir:

ISZ X

X konumunun içeriği 1 ile artırılır. Sonuç 0 ise, bir sonraki komut atlanır. Olası bir mikro-islem dizisi şöyledir

```
t1: MAR d (IR(adres() t2:  
MBR d Bellek  
t3: MBR d (MBR(+ 1  
t4: Bellek d (MBR()  
Eğer ((MBR=0( o zaman (PC d (PC(+ I(
```

Burada tanıtılan yeni özellik koşullu eylemdir. Eğer (MBR)= 0 ise PC artırılır. Bu test ve eylem tek bir mikro işlem olarak uygulanabilir. Ayrıca, bu mikro işlemin MBR'deki güncellenmiş değerin belleğe geri depolandığı aynı zaman birimi sırasında gerçekleştirilebileceğini unutmayın.

Son olarak, bir alt rutin çağrı komutunu düşünün. Örnek olarak, bir dallanma ve adres kaydetme komutunu ele alalım:

BSA X

BSA komutunu takip eden komutun adresi X konumuna kaydedilir ve yürütme X konumunda devam eder+ I. Kaydedilen adres daha sonra geri dönüş için kullanılacaktır. Bu, alt rutin çağrılarını desteklemek için basit bir tekniktir. Aşağıdaki mikro-islemeler yeterlidir:

```
t1: MAR d (IR(adres() MBR d  
(PC()  
t2: PC d (IR(adres()  
Bellek d (MBR()  
t3: PC d (PC(+ I
```

Komutun başlangıcında PC'de bulunan adres, sıradaki bir sonraki komutun adresidir. Bu, IR'de belirtilen adrese kaydedilir. Son adres de bir sonraki komut döngüsü için komutun adresini sağlamak üzere artırılır.

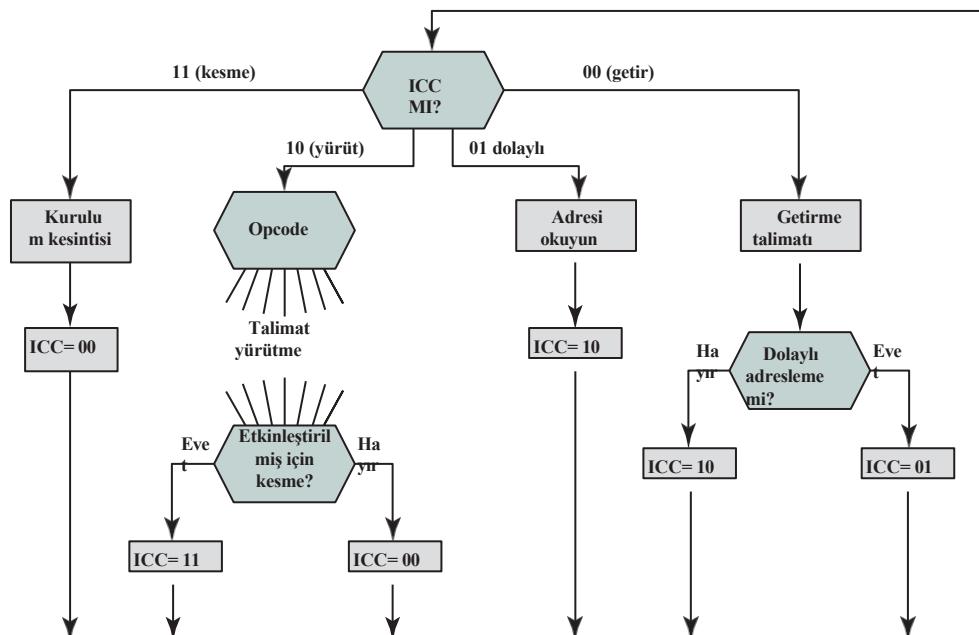
## Talimat Döngüsü

Komut döngüsünün her aşamasının bir dizi temel mikro-isleme ayırtılabilceğini gördük. Örneğimizde, getirme, dolaylı ve kesme döngüleri için birer dizi vardır ve yürütme döngüsü için her bir işlem kodu için bir mikro işlem dizisi .

Resmi tamamlamak için mikro-islem dizilerini birbirine bağlamamız gerekir ve bu Şekil 20.3'te yapılmaktadır. *Komut döngüsü kodu* (ICC) adı verilen yeni bir 2 bitlik kayıt varsayıyoruz. ICC, işlemcinin durumunu döngünün hangi kısmında olduğu açısından belirler:

00: Getir

01: Dolaylı



**Şekil 20.3** Komut Döngüsü için Akış Şeması

10: Yürüt

11: Kesme

Dört döngünün her birinin sonunda, ICC uygun şekilde ayarlanır. Dolaylı döngüyü her zaman yürütme döngüsü takip eder. Kesme *döngüsünü* her zaman getirme *döngüsü* takip eder (bkz. Şekil 14.4, *Komut Döngüsü*). Hem getirme hem de yürütme döngülerini için bir sonraki döngü sistemin durumuna bağlıdır.

Böylece, Şekil 20.3'teki akış şeması, yalnızca komut dizisine ve kesme modeline bağlı olarak mikro işlemlerin tam sırasını tanımlar. Elbette bu basitleştirilmiş bir örnektir. Gerçek bir işlemci için akış şeması daha karmaşık olacaktır. Her , tartışmamızda işlemcinin çalışmasının bir dizi mikro işlemin gerçekleştirilmesi olarak tanımlandığı noktaya ulaşmış bulunuyoruz. Şimdi kontrol ünitesinin bu dizinin gerçekleşmesine nasıl neden olduğunu ele alabiliriz.

## 20.2 İŞLEMCİNİN KONTROLÜ

### İşlevsel Gereksinimler

Bir önceki bölümde yaptığımız analiz sonucunda, işlemcinin davranışını veya işleyişini mikro-işlemler olarak adlandırılan temel işlemlere ayırttık. İşlemcinin çalışmasını en temel seviyesine indirgeyerek, kontrol biriminin gerçekleşmesine neden olması gereken şeyin tam olarak ne olduğunu tanımlayabiliriz. Böylece, kontrol ünitesi için *işlevsel gereksinimleri* tanımlayabiliriz: kontrol ünitesinin yerine getirmesi gereken işlevler. Bu işlevsel gereksinimlerin tanımı, kontrol biriminin tasarımını ve uygulanması için temel oluşturur.

Eldeki bilgilerle, aşağıdaki üç adımlı süreç kontrol ünitesinin karakterizasyonuna yol açar:

1. İşlemcinin temel unsurlarını tanımlayın.
2. İşlemcinin gerçekleştirdiği mikro işlemlerini tanımlayın.
3. Mikro işlemlerin gerçekleştirilemesi için kontrol yerine getirmesi gereken işlevleri belirleyin.

Adım 1 ve 2'yi zaten gerçekleştirdik. Sonuçları özetleyelim. İlk olarak, işlemcinin temel işlevsel unsurları şunlardır:

- ALU
- Kayıtlar
- Dahili veri yolları
- Harici veri yolları
- Kontrol ünitesi

Biraz düşününce bunun eksiksiz bir liste olduğuna ikna olacaksınız. ALU bilgisayarın işlevsel özüdür. Kayıtlar, işlemcinin içindeki verileri saklamak için kullanılır. Bazı kayıtlar komut sıralamasını yönetmek için gereken durum bilgilerini içerir (örneğin, bir program durum sözcüğü). Diğerleri ise ALU, bellek ve I/O modüllerine giden ya da bu modüllerden gelen verileri içerir. Dahili veri yolları, verileri yazmaçlar arasında ve yazmaç ile ALU arasında taşımak için kullanılır. Harici veri yolları, genellikle bir sistem veriyolu aracılığıyla yazmaçları belleğe ve G/C modüllerine bağlar. Kontrol birimi işlemci içinde işlemlerin gerçekleştirmesine neden olur.

Bir programın yürütülmesi, bu öncül öğeleri içeren işlemlerden oluşur. Gördüğümüz gibi, bu işlemler bir dizi mikro-islemden oluşmaktadır. Bölüm 20.1 incelediğinde, okuyucu tüm mikro işlemlerin aşağıdaki kategorilerden birine girdiğini görmeliydi:

- Verileri bir kayıttan diğerine aktarın.
- Verileri bir kayıttan harici bir arayüze (örn. sistem veriyolu) aktarın.
- Harici bir arayüzden bir yazmaç'a veri aktarın.
- Giriş ve çıkış için kayıtları kullanarak bir aritmetik veya mantık işlemi gerçekleştirin.

Bir komut döngüsünü gerçekleştirmek için gereken tüm mikro işlemler, komut setindeki her komutu yürütmek için gereken tüm mikro işlemler de dahil olmak üzere, bu kategorilerden birine .

Şimdi kontrol ünitesinin çalışma şekli hakkında biraz daha açık olabiliriz. Kontrol ünitesi iki temel görevi yerine getirir:

- **Sıralama:** Kontrol ünitesi, yürütülen programa bağlı olarak işlemcinin bir dizi mikro işlemi uygun sırayla gerçekleştirmesini sağlar.
- **Yürütme:** Kontrol ünitesi her bir mikro-operasyonun gerçekleştirilemesini sağlar.

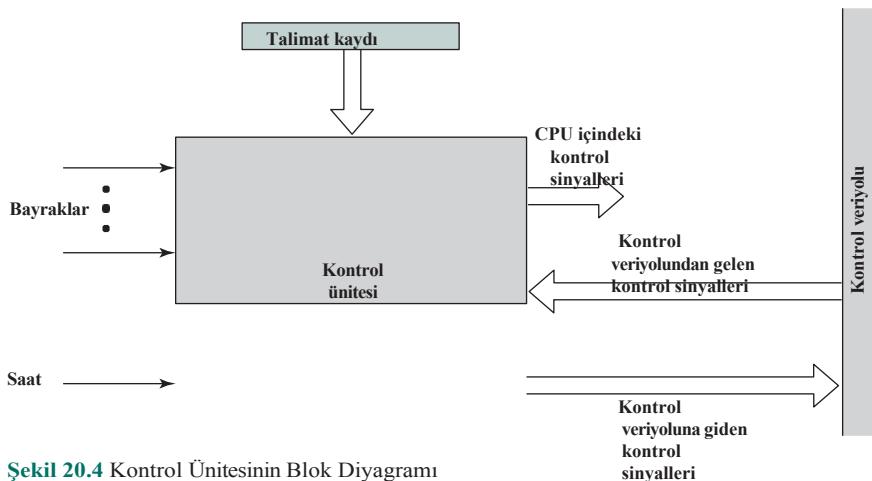
Yukarıda anlatılanlar kontrol ünitesinin ne yaptığıının işlevsel bir açıklamasıdır. Kontrol ünitesinin nasıl çalışığının anahtarı kontrol sinyallerinin kullanılmasıdır.

### Kontrol Sinyalleri

İşlemciyi oluşturan unsurları (ALU, kaydediciler, veri yolları) ve gerçekleştirilen mikro işlemleri tanımladık. Kontrol işlevini yerine getirebilmesi için, sistemin durumunu belirlemesini sağlayan girişlere ve sistemin davranışını kontrol etmesini sağlayan çıkışlara sahip olması gereklidir. Bunlar kontrol ünitesinin harici özellikleridir. Dahili olarak, kontrol ünitesi sıralama ve yürütme işlevlerini yerine getirmek için gereken mantığa sahip olmalıdır. Kontrol ünitesinin dahili işleyişine ilişkin bir tartışmayı Bölüm 20.3 ve Bölüm 21'e erteliyoruz. Bu bölümün geri kalanı kontrol birimi ile işlemcinin diğer elemanları arasındaki etkileşimle ilgilidir.

Şekil 20.4 kontrol ünitesinin tüm giriş ve çıkışlarını gösteren genel bir modelidir. Girdiler şunlardır:

- **Saat:** Kontrol ünitesi bu şekilde "zamanı tutar". Kontrol ünitesi her saat darbesi için bir mikro işlemin (veya bir dizi eşzamanlı mikro işlemin) gerçekleştirilmemesine neden olur. Bu bazen işlemci döngü süresi veya saat döngü süresi olarak adlandırılır.
- **Talimat kaydı:** Geçerli talimatın işlem kodu ve adresleme modu, çıkış döngüsü sırasında hangi mikro işlemlerin gerçekleştirileceğini belirlemek için kullanılır.
- **Bayraklar:** Bunlar kontrol birimi tarafından işlemcinin durumunu ve önceki ALU işlemlerinin sonucunu belirlemek için gereklidir. Örneğin, sıfır artı ve atla (ISZ) komutu için, sıfır bayrağı ayarlanmışsa kontrol birimi PC'yi artıracaktır.
- **Kontrol gelen kontrol sinyalleri:** Sistem veriyolunun kontrol kısmı kontrol ünitesine sinyaller sağlar.



**Şekil 20.4** Kontrol Ünitesinin Blok Diyagramı

Çıktılar aşağıdaki gibidir:

- **İşlemci içindeki kontrol sinyalleri:** Bunlar iki türdür: verilerin bir kayıttan diğerine taşınmasına neden olanlar ve belirli ALU işlevlerini etkinleştirenler.
- **Kontrol veriyoluna giden kontrol sinyalleri:** Bunlar da iki tiptir: belleğe giden kontrol sinyalleri ve I/O modüllerine giden kontrol sinyalleri.

Üç tür kontrol sinyali kullanılır: bir ALU işlevini etkinleştirenler; bir veri yolunu etkinleştirenler; ve harici sistem veriyolu veya diğer harici arayüzdeki sinyaller. Tüm bu sinyaller sonuçta doğrudan bireysel mantık kapılarına ikili girişler olarak uygulanır.

Kontrol biriminin kontrolü nasıl sağladığını görmek için getirme döngüsünü tekrar ele alalım. Kontrol birimi komut döngüsünün neresinde olduğunu takip eder. Belirli bir noktada, bir sonraki getirme döngüsünün gerçekleştirileceğini bilir. İlk adım PC'nin içeriğini MAR'a aktarmaktır. Kontrol birimi bunu, PC'nin bitleri ile MAR'in bitleri arasındaki kapıları açan kontrol sinyalini etkinleştirerek yapar. Bir sonraki adım bellekten MBR'ye bir kelime okumak ve PC'yi artırmaktır. Kontrol ünitesi bunu aşağıdaki kontrol sinyallerini aynı anda göndererek yapar:

- Kapıları açarak MAR'in içeriğinin adres veriyoluna sağlayan bir kontrol sinyali;
- Kontrol veriyolunda bir bellek okuma kontrol sinyali;
- Kapıları açarak veri yolu içeriğinin MBR'de saklanması sağlayan bir kontrol sinyali;
- PC'nin içeriğine 1 ekleyen ve sonucu PC'ye geri depolayan mantığa kontrol sinyalleri.

Bunu takiben, kontrol ünitesi MBR ve IR arasındaki kapıları açan bir kontrol sinyali gönderir.

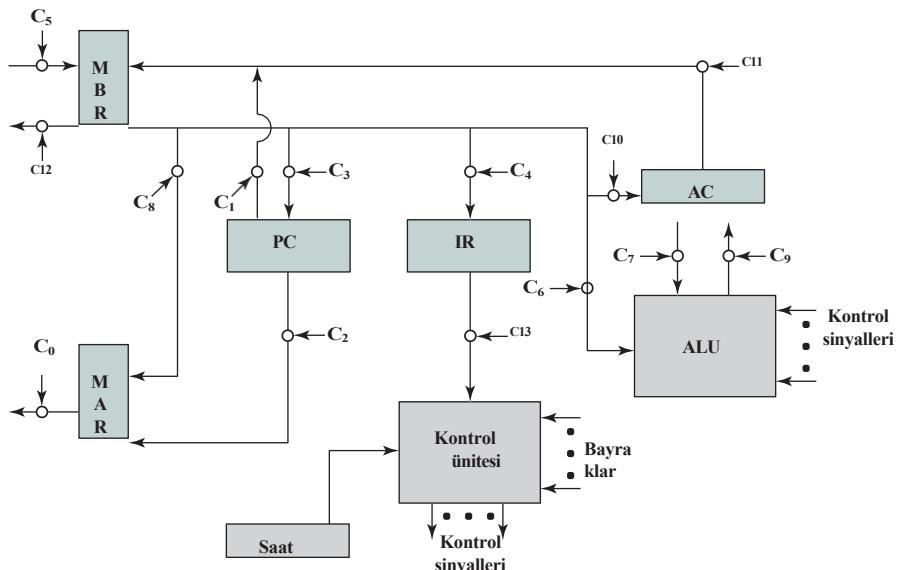
Bu, bir şey dışında getirme döngüsünü tamamlar: Kontrol birimi bir sonraki aşamada dolaylı döngü mü yoksa yürütme döngüsü mü gerçekleştireceğine karar vermelidir. Buna karar vermek için, dolaylı bir bellek başvurusu yapılmışlığını görmek için IR'yi inceler.

Dolaylı ve kesme döngülerini benzer şekilde çalışır. Yürütme döngüsü için, kontrol birimi işlem kodunu inceleyerek başlar ve buna dayanarak yürütme döngüsü için hangi mikro işlem dizisinin gerçekleştirileceğine karar verir.

### Bir Kontrol Sinyalleri Örneği

Kontrol ünitesinin işleyişini göstermek için basit bir örneği inceleyelim. Şekil 20.5 bu örneği göstermektedir. Bu, tek bir hesaplayıcıya (AC) sahip basit bir işlemcidir. Elemanlar arasındaki veri yolları gösterilmiştir. Kontrol biriminden çıkan sinyaller için kontrol yolları gösterilmemiştir, ancak kontrol sinyallerinin sonlandırmaları  $C_i$  olarak etiketlenmiş ve bir daire ile gösterilmiştir. Kontrol birimi saatten, IR'den ve bayraklardan girişler alır. Her saat döngüsünde, kontrol ünitesi

## 718 BÖLÜM 20 / KONTROL ÜNİTESİNİN ÇALIŞTIRILMASI



**Şekil 20.5** Veri Yolları ve Kontrol Sinyalleri

tüm girişlerini okur ve bir dizi kontrol sinyali yayar. Kontrol sinyalleri üç ayrı hedefe gider:

- **Veri yolları:** Kontrol birimi dahili veri akışını kontrol eder. Örneğin, komut getirildiğinde, bellek tampon yazmacının içeriği IR'ye aktarılır. Kontrol edilecek her yol için bir anahtar vardır (şekilde daire ile gösterilmiştir). Kontrol ünitesinden gelen bir kontrol sinyali, verilerin geçmesine izin vermek için kapıyı geçici olarak açar.
- **ALU:** Kontrol ünitesi ALU'nun çalışmasını bir dizi kontrol sinyali ile kontrol eder. Bu sinyaller ALU içindeki çeşitli mantık devrelerini ve kapılarını etkinleştirir.
- **Sistem veriyolu:** Kontrol ünitesi, sistem veriyolunun kontrol hatlarına kontrol sinyalleri gönderir (örneğin, bellek READ).

Kontrol ünitesi komut döngüsünün neresinde olduğu bilgisini muhafaza etmelidir. Bu bilgiyi kullanarak ve tüm girişlerini okuyarak, kontrol ünitesi mikro işlemlerin gerçekleşmesine neden olan bir dizi kontrol sinyali yayar. Olayların sırasını zamanlamak için saat darbelerini kullanır ve olaylar arasında sıgnal seviyelerinin dengelenmesi için zaman tanır. Tablo 20.1, daha önce açıklanan bazı mikro-islem dizileri için gerekli olan kontrol sinyallerini göstermektedir. Basitlik açısından, PC'nin artırılması ve sabit adreslerin PC ve MAR'a yüklenmesi için veri ve kontrol yolları gösterilmemiştir.

Kontrol ünitesinin minimal doğası üzerinde düşünmeye değer. Kontrol ünitesi tüm bilgisayarı çalıştıran motordur. Bunu yalnızca yürütülecek talimatları ve aritmetik ve mantıksal işlemlerinin sonuçlarının doğasını (örneğin, pozitif, taşıma, vb.) bilerek yapar. Hiçbir zaman işlenen verileri ya da üretilen gerçek sonuçları göremez. Ve her şeyi işlemci içindeki noktalara giden birkaç kontrol sinyali ve sistem veri yoluna giden birkaç kontrol sinyali ile kontrol eder.

**Tablo 20.1** Mikro-işlemler ve Kontrol Sinyalleri

	Mikro operasyonlar	Aktif Kontrol Sinyalleri
Getir:	$t_1: MAR \leftarrow PC$	$C_2$
	$t_2: MBR \leftarrow Bellek[PC] + 1$	$C_S, C_R$
	$t_3: IR \leftarrow MBR$	$C_4$
Dolaylı:	$t_1: MAR \leftarrow IR[Adres]$	$C_8$
	$t_2: MBR \leftarrow Bellek[IR]$	$C_S, C_R$
	$t_3: IR[Adres] \leftarrow MBR[Adres]$	$C_4$
Kesildi:	$t_1: MBR \leftarrow PC$	$C_1$
	$t_2: MAR \leftarrow Kaydet[adres], PC \leftarrow Rutin[adres], s \leftarrow 1$	
	$t_3: Bellek \leftarrow MBR$	$C_{12}, C_W$

$C_R$ = Sistem kontrol sinyali okuma.  $C_W$ = Kontrol

sinyalini sistem veriyoluna yazın.

## Dahili İşlemci Organizasyonu

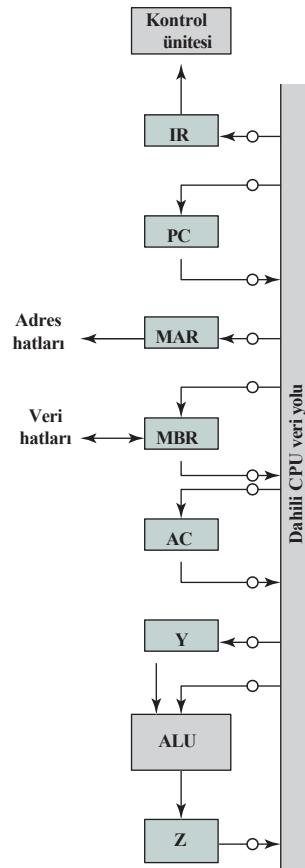
Şekil 20.5 çeşitli veri yollarının kullanımını göstermektedir. Bu tür bir organizasyonun karmaşıklığı açık olmalıdır. Daha tipik olarak, Şekil 14.2'de (*CPU'nun İç Yapısı*) önerildiği gibi bir tür dahili veri yolu düzenlemesi kullanılacaktır.

Dahili bir işlemci veri yolu kullanarak, Şekil 20.5, Şekil 20.6'da gösterildiği gibi yeniden düzenlenlenebilir. Tek bir dahili veri yolu ALU ve tüm işlemci yazmaçlarını birbirine bağlar. Kapılar ve kontrol sinyalleri, her bir kaydediciden veriyoluna ve veriyolundan verinin hareketi için sağlanır. Ek kontrol sinyalleri, sistem (harici) veri yoluna veri ve ALU'nun çalışmasını kontrol eder.

Organizasyona Y ve Z olarak etiketlenmiş iki yeni yazmaç eklenmiştir. Bunlar ALU'nun düzgün çalışması için gereklidir. İki işlenen içeren bir işlem yapıldığında, biri dahili veri yolundan elde edilebilir, ancak diğerı başka bir kaynaktan elde edilmelidir. AC bu amaç için kullanılabilir, ancak bu sistemin esnekliğini sınırlar ve birden fazla genel amaçlı kaydediciye sahip bir işlemci ile çalışmaz. Y kaydedicisi diğer giriş için geçici depolama sağlar. ALU, dahili depolama alanı olmayan bir kombinatoryal devredir (bkz. Bölüm 11). Bu nedenle, kontrol sinyalleri bir ALU işlevini etkinleştirdiğinde, ALU'ya giriş çıkış dönüştürülür. Bu nedenle, ALU'nun çıkışı doğrudan veri yoluna bağlanamaz, çünkü bu çıkış girişe geri beslenir. Register Z geçici çıkış depolama sağlar. Bu düzenlemeyle, bellekten AC'ye bir değer ekleme işlemi aşağıdaki adımlardan oluşur:

$t_1: MAR \leftarrow IR[adres]$      $t_2: MBR \leftarrow Bellek$   
 $t_3: Y \leftarrow MBR$      $t_4: Z \leftarrow AC + (Y)$   
 $t_5: AC \leftarrow (Z)$

Diğer organizasyonlar da mümkündür, ancak genel olarak bir tür dahili veri yolu veya dahili veri yolu kümesi kullanılır. Ortak veri yollarının kullanılması



Şekil 20.6 Dahili Veri Yolu ile CPU

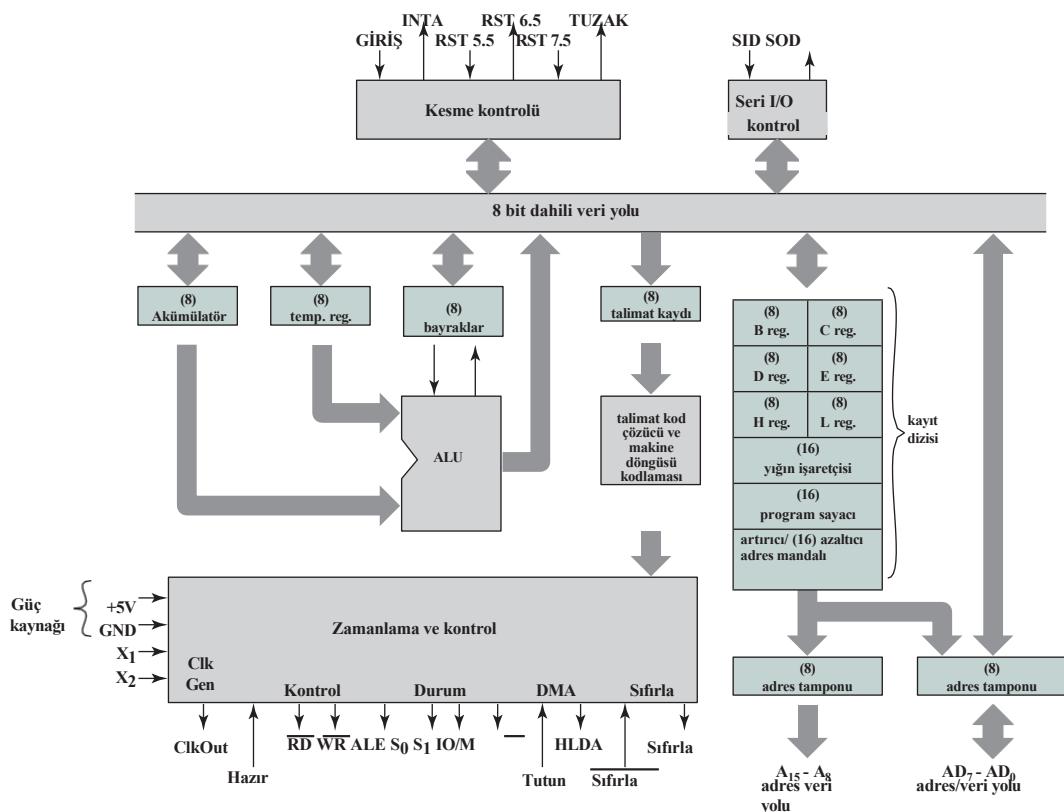
ara bağlantı düzeni ve işlemcinin kontrolü. Dahili veri yolu kullanımının bir başka pratik nedeni de yerden tasarruf etmektir.

### Intel 8085

Bu bölümde şimdije kadar tanıtılan bazı kavramları örneklemek için Intel 8085'i ele alalım. Organizasyonu Şekil 20.7'de gösterilmiştir. Kendini açıklayıcı olmayabilecek birkaç temel bileşen şunlardır:

- **Artırıcı/azaltıcı adres mandası:** Yiğin işaretçisinin veya program sayacının içeriğine 1 ekleyebilen veya içeriğinden 1 çıkarabilen mantık. Bu, ALU'nun bu amaç için kullanılmasını önleyerek zaman kazandırır.
- **Kesme kontrolü:** Bu modül birden fazla kesme sinyalini seviyesini yönetir.
- **Seri I/O kontrolü:** Bu modül, her seferinde 1 bit iletişim kurmak üzere cihazlara arayüz oluşturur.

Tablo 20.2'de 8085'e giren ve çıkan harici sinyaller açıklanmaktadır. Bunlar harici sistem veri yoluna bağlıdır. Bu sinyaller 8085 işlemci ile sistemin geri kalımı arasındaki arayıldır (Şekil 20.8).



**Şekil 20.7** Intel 8085 CPU Blok Diyagramı

Kontrol ünitesinin (1) talimat kod çözücü ve makine döngüsü kodlaması ve (2) zamanlama ve olarak etiketlenmiş iki bileşeni olduğu tanımlanmıştır. İlk bileşenin tartışılması bir sonraki bölüme ertelenmiştir. Kontrol ünitesinin özü zamanlama ve kontrol modülüdür. Bu modül bir saat içerir ve giriş olarak mevcut talimiği ve bazı harici kontrol sinyallerini kabul eder. Çıkışı, işlemcinin diğer bileşenlerine giden kontrol sinyalleri ve harici sistem veri yoluna giden kontrol sinyallerinden oluşur.

İşlemci işlemlerinin zamanlanması saat tarafından senkronize edilir ve kontrol birimi tarafından kontrol sinyalleri ile kontrol edilir. Her komut çevrimi bir ila beş makine çevrimine bölünür; her makine çevrimi de üç ila beş duruma bölünür. Her durum bir saat çevrimi sürer. Bir durum sırasında işlemci, kontrol sinyalleri tarafından belirlenen bir veya bir dizi eşzamanlı mikro işlemi gerçekleştirir. Makine çevrimlerinin sayısı belirli bir komut için sabittir ancak bir komuttan diğerine değişir. Makine döngüleri veri yolu erişimlerine eşdeğer olarak tanımlanır.

Dolayısıyla, bir komut için makine çevrimi sayısı aşağıdakilere bağlıdır. İşlemcinin harici cihazlarla iletişim kurma sayısı. Örneğin, bir komut iki adet 8 bitlik bölümden oluşuyorsa, komutu almak için iki makine çevrimi gereklidir. Bu komut 1 baytlık bir bellek veya G/C işlemi içeriyorrsa, yürütme için üçüncü bir makine çevrimi gereklidir.

## 722 BÖLÜM 20 / KONTROL ÜNİTESİNİN ÇALIŞTIRILMASI

**Tablo 20.2** Intel 8085 Harici Sinyaller

<i>Adres ve Veri Sinyalleri</i>	
<b>Yüksek Adres (A15-A8)</b>	16 bitlik bir adresin yüksek sıralı 8 biti.
<b>Adres/Veri (AD7-AD0)</b>	16 bitlik bir adresin veya 8 bitlik verinin alt sıradaki 8 biti. Bu çoklama pimlerden tasarruf sağlar.
<b>Seri Giriş Verileri (SID)</b>	Seri olarak (her seferinde bir bit) iletim yapan cihazları barındırmak için tek bitlik bir giriş.
<b>Seri Çıkış Verileri (SOD)</b>	Seri alım yapan cihazlara uyum sağlamak için tek bitlik bir çıkış.
<i>Zamanlama ve Kontrol Sinyalleri</i>	
<b>CLK (ÇIKIŞ)</b>	Sistem saat. CLK sinyali çevresel yongalara gider ve zamanlamalarını senkronize eder.
<b>X1, X2</b>	Bu sinyaller, dahili saat üreticini çalıştırılmak için harici bir kristalden veya başka bir cihazdan gelir.
<b>Adres Mandallı Etkin (ALE)</b>	Bir makine döngüsünün ilk saat durumu sırasında meydana gelir ve çevresel yongaların adres hatlarını saklamasına neden olur. Bu, adres modülünün (örneğin bellek, G/Ç) adreslendirdiğini fark etmesini sağlar.
<b>Durum (S0, SI)</b>	Bir okuma veya yazma işleminin gerçekleşip gerçekleşmediğini belirtmek için kullanılan kontrol sinyalleri.
<b>IO/M</b>	Okuma ve yazma işlemleri için G/Ç veya bellek modüllerini etkinleştirmek için kullanılır.
<b>Okuma Kontrolü (RD)</b>	Seçilen bellek veya G/Ç modülünün okunacağını ve veri yolunun veri aktarımı için kullanılabilir olduğunu gösterir.
<b>Yazma Kontrolü (WR)</b>	Veri yolundaki verilerin seçilen belleğe veya G/Ç konumuna yazılacağını belirtir.
<i>Bellek ve G/Ç Başlatmalı Semboller</i>	
<b>Tutun</b>	CPU'dan harici sistem veriyolunun kontrolünü ve kullanımını bırakmasını ister. CPU, anda IR'de bulunan komutun yürütülmesini tamamlayacak ve ardından CPU tarafından kontrol, adres veya veri yollarına hiçbir sinyalin eklenmediği bir bekleme durumuna girecektir. Bekleme durumu sırasında veri yolu DMA işlemleri için kullanılabilir.
<b>Tutma Onaylama (HOLDA)</b>	Bu kontrol ünitesi çıkış sinyali TUTMA sinyalini onaylar ve veri yolunun artık kullanılabilir olduğunu belirtir.
<b>HAZIR</b>	CPU'yu daha yavaş bellek veya I/O cihazlarıyla senkronize etmek için kullanılır. Adreslenen bir cihaz READY HAZIR uyarıtı verdiğiinde, CPU bir giriş (DBIN) veya çıkış (WR) işlemine devam edebilir. Aksi takdirde, CPU cihaz hazır olana kadar bekleme durumuna girer.
<i>Kesintiyle İlgili Sinyaller</i>	
<b>KAPAN</b>	Yeniden Başlatma Kesmeleri (RST 7.5, 6.5, 5.5)
<b>Kesme Talebi (INTR)</b>	Bu beş hat harici bir cihaz tarafından CPU'yu kesmek için kullanılır. CPU bekleme durumundaysa veya kesme devre dışı bırakılmışsa isteği yerine getirmeyecektir. Bir kesme sadece bir komut tamamlandığında yerine getirilir. Kesmeler azalan öncelik sırasına göreler.
<b>Kesme Onaylama</b>	Bir kesintiyi onaylar.

*CPU Başlatma***SİFIRLAMA GİRİŞİ**

PC içeriğinin sıfırına ayarlanması neden olur. CPU sıfır konumunda yürütmeye devam eder.

**SİFIRLAMA ÇIKIŞI**

CPU'nun sıfırlandığını onaylar. Sinyal, sistemin geri kalanını sıfırlamak için kullanılabilir.

*Gerilim ve Toprak***VCC**

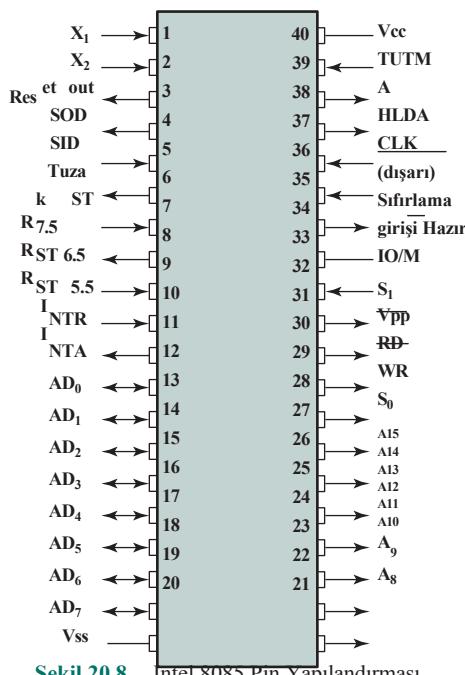
+5 voltlu güç kaynağı

**VSS**

Elektriksel topraklama

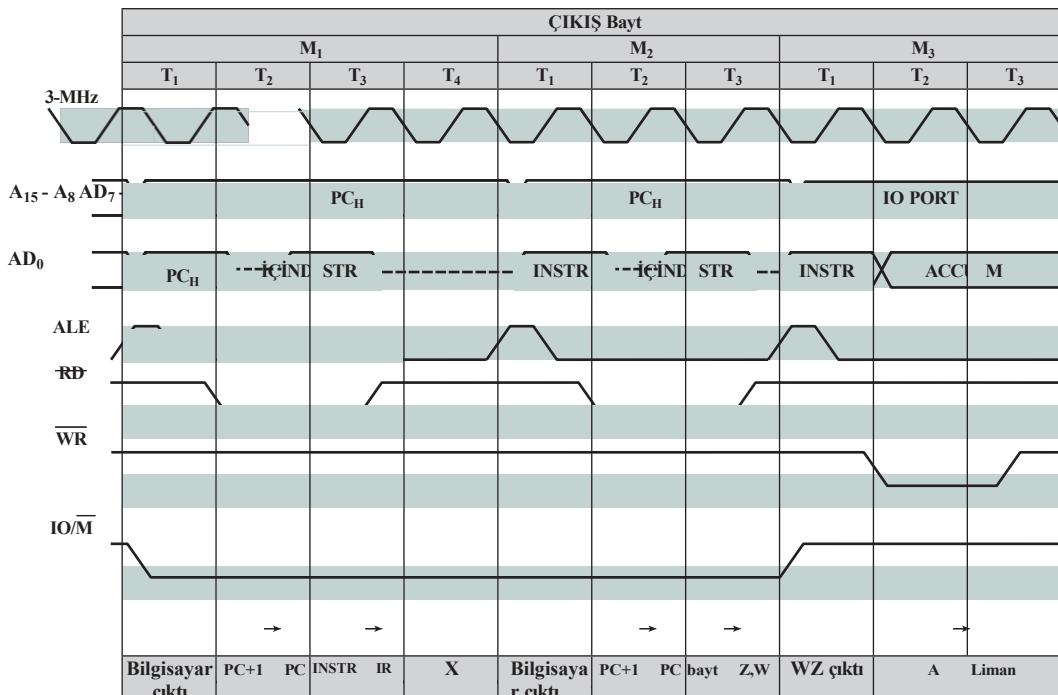
**Şekil 20.9**, harici kontrol sinyallerinin değerini gösteren bir 8085 zamanlama örneği vermektedir. Tabii ki, aynı zamanda, kontrol ünitesi dahili veri transferlerini kontrol eden dahili kontrol sinyalleri üretir. Diyagram bir OUT komutu için komut döngüsünü göstermektedir. Üç makine döngüsü ( $M_1$ ,  $M_2$ ,  $M_3$ ) gereklidir. İlk sırasında OUT komutu getirilir. İkinci makine döngüsü, çıkış için seçilen G/C cihazının numarasını içeren komutun ikinci yarısını alır. Üçüncü döngü sırasında, AC'nin içeriği veri yolu üzerinden seçilen cihaza yazılır.

Adres Mandali Etkin (ALE) darbesi kontrol ünitesinden her makine döngüsünün başlığını bildirir. ALE darbesi harici devreleri uyarır. Makine döngüsü  $M_1$ 'in  $T_1$  zamanlama durumu sırasında, kontrol ünitesi bunun bir bellek işlemi olduğunu belirtmek için IO/M sinyalini ayarlar. Ayrıca, kontrol ünitesi PC'nin içeriğinin



**Şekil 20.8** Intel 8085 Pin Yapılandırması

## 724 BÖLÜM 20 / KONTROL ÜNİTESİNİN ÇALIŞTIRILMASI



**Şekil 20.9** Intel 8085 OUT Komutu için Zamanlama Diyagramı

adres veriyoluna ( $A_{15}$  ile  $A_8$ ) ve adres/veri veriyoluna ( $AD_7$  ile  $AD_0$ ) yerleştirilir. ALE darbesinin düşen kenarı ile veri yolundaki diğer modüller adresi kaydeder.

T<sub>2</sub> zamanlama durumu sırasında, adreslenen bellek modülü adreslenen bellek konumunun içeriğini adres/veri yoluna yerleştirir. Kontrol ünitesi bir okumayı belirtmek için Okuma Kontrolü (RD) sinyalini ayarlar, ancak veriyi veri yolundan kopyalamak için T<sub>(3)</sub>e kadar bekler. Bu, bellek modülüne veriyi veri yoluna koyması ve sinyal seviyelerinin dengelenmesi için zaman verir. Son durum olan T<sub>4</sub>, işlemcinin komutun kodunu çözdüğü bir veri *yolu boşta* durumudur. Kalan makine döngüleri benzer şekilde ilerler.

### 20.3 KABLOLU UYGULAMA

Kontrol ünitesini girişleri, çıkışları ve işlevleri açısından ele alındı. Şimdi kontrol ünitesinin uygulanması konusuna dönüyoruz. Çok çeşitli teknikler kullanılmıştır. Bunların çoğu iki kategoriden birine girmektedir:

- Kablolu uygulama
- Mikro programlı uygulama

**Kablolu bir uygulamada**, kontrol ünitesi esasen bir durum makinesi devresidir. Giriş mantık sinyalleri bir dizi çıkış mantık sinyaline dönüştürülür ve bunlar

kontrol sinyalleridir. Bu yaklaşım bu bölümde incelenmiştir. Mikro programlı uygulama 21. Bölümün konusudur.

## Kontrol Ünitesi Girişleri

Şekil 20.4, şimdije kadar tartıştığımız şekliyle kontrol ünitesini göstermektedir. Temel girişler IR, saat, bayraklar ve kontrol veri yolu sinyalleridir. Bayraklar ve kontrol veriyolu sinyalleri söz konusu olduğunda, her bir bitin tipik olarak bir anlamı vardır (örneğin, taşıma). Ancak diğer iki giriş kontrol ünitesi için doğrudan faydalı değildir.

İlk olarak IR'yi düşünün. Kontrol ünitesi işlem kodunu kullanır ve farklı talimatlar için farklı eylemler gerçekleştirir (farklı kontrol sinyalleri kombinasyonu yayarlar). Kontrol birimi mantığını basitleştirmek için, her işlem kodu için benzersiz bir mantık girişi olmalıdır. Bu işlev, kodlanmış bir girişi alan ve tek bir çıkış üreten bir kod çözücü tarafından gerçekleştirilebilir. Genel olarak, bir kod çözücü  $n$  adet ikili girişe ve  $2^{(n)}$  adet ikili çıkış sahip olacaktır.  $2^n$  farklı giriş deseninin her biri tek bir benzersiz çıkış etkinleştirecektir. Tablo 20.3  $n = 4$  için bir örnektir. Bir kontrol ünitesinin kod çözucusu, değişken uzunlukta işlem kodlarını hesaba katmak için tipik olarak bundan daha karmaşık olmak zorundadır. Bir kod çözücüyü uygulamak için kullanılan dijital mantığın bir örneği Bölüm 11'de sunulmuştur.

Kontrol ünitesinin saat kısmı tekrarlayan bir darbe dizisi yayarlar. Bu, mikro işlemlerin süresini ölçmek için kullanışlıdır. Esasen, saat darbelerinin periyodu, sinyallerin boyunca yayılmasına izin verecek kadar uzun olmalıdır

**Tablo 20.3** 4 Girişli ve 16 Çıkışlı Bir Kod Çözücü

## 726 BÖLÜM 20 / KONTROL ÜNİTESİNİN ÇALIŞTIRILMASI

veri yolları ve işlemci devresi aracılığıyla. Ancak, gördüğümüz gibi, kontrol ünitesi tek bir komut döngüsü içinde farklı zaman birimlerinde farklı kontrol sinyalleri yayar. Bu nedenle, kontrol ünitesine giriş olarak bir sayıç ve  $T_1$ ,  $T_2$ , ... için farklı bir kontrol sinyali kullanılmasını istiyoruz. Bir komut döngüsünün sonunda, kontrol ünitesi sayıçı  $T_{(1)}$ de yeniden başlatmak için sayaca geri besleme yapmalıdır.

Bu iki iyileştirme ile kontrol ünitesi Şekil 20.10'daki gibi gösterilebilir.

### Kontrol Ünitesi Mantığı

Bir kontrol ünitesinin kablolu uygulamasını için geriye kalan tek şey, giriş sinyallerinin bir fonksiyonu olarak çıkış kontrol sinyalleri üreten kontrol ünitesinin iç mantığını tartısmaktır.

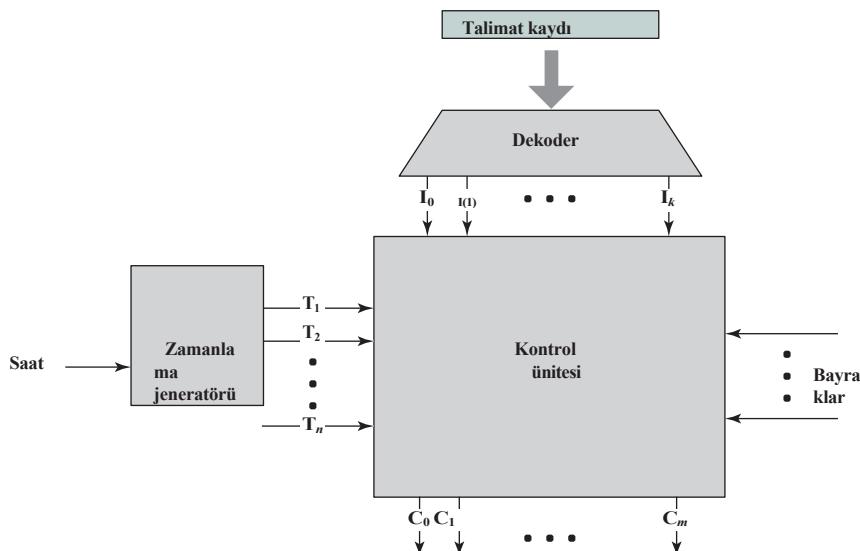
Esasen yapılması gereken, her bir kontrol sinyali için, girişlerin bir fonksiyonu olarak bu sinyalin Boolean ifadesini türetmektir. Bu en iyi örnekle açıklanabilir. Şekil 20.5'te gösterilen basit örneğimizi tekrar ele alalım. Tablo 20.1'de komut döngüsünün dört aşamasından üçünü kontrol etmek için gereken mikro-işlem dizilerini ve kontrol sinyallerini gördük.

Tek bir kontrol sinyalini ele alalım,  $C_5$ . Bu sinyal, verilerin harici veri yolundan MBR'ye okunmasına neden olur. Tablo 20.1'de iki kez kullanıldığını görebiliriz. Aşağıdaki yorumu sahip iki yeni kontrol sinyali, P ve Q, tanımlayalım:

$PQ = 00$	Getirme Döngüsü
$PQ = 01$	Dolaylı Döngü
$PQ = 10$	Döngüyü Yürüt
$PQ = 11$	Kesme Döngüsü

Ardından aşağıdaki Boolean ifadesi  $C_{(5)}$ 'i tanımlar:

$$C_5 = P \cdot \overline{Q} \cdot \overline{T_2} + P \cdot Q \cdot T_2$$



Şekil 20.10 Şifresi Çözülmüş Girişlere Sahip Kontrol Ünitesi

,  $C_5$  kontrol sinyali hem getirme hem de dolaylı döngülerin ikinci zaman birimi sırasında onaylanacaktır.

Bu ifade tam değildir.  $C_5$  ayrıca yürütme döngüsü sırasında da gereklidir. Basit örneğimiz için, bellekten okuma yapan sadece üç komut olduğunu varsayıyalım: LDA, ADD ve AND. Şimdi  $C_{(5)}$  şu şekilde tanımlayabiliriz

$$\overline{C_5} = \overline{P-Q-T_2} + \overline{P-Q-T_2} + P-Q-(\overline{\text{LDA}} + \overline{\text{ADD}} + \overline{\text{AND}})-T_{(2)}$$

Aynı süreç işlemci tarafından üretilen her kontrol sinyali için tekrarlanabilir. Sonuç, kontrol ünitesinin ve dolayısıyla işlemcinin davranışını tanımlayan bir dizi Boole denklemi olacaktır.

Her şeyi birbirine bağlamak için, kontrol birimi komut döngüsünün durumunu kontrol etmelidir. Daha önce de belirtildiği gibi, her alt döngünün (getirme, dolaylı, yürütme, kesme) sonunda, kontrol birimi zamanlama üreticinin yeniden başlatılmasına ve  $T_1$  vermesine neden olan bir sinyal yayırlar. Kontrol birimi ayrıca gerçekleştirilecek bir sonraki alt döngüyü tanımlamak için uygun P ve Q değerlerini ayarlamalıdır.

Okuyucu, modern bir karmaşık işlemcide, kontrol birimini tanımlamak için gereken Boole denklemelerinin sayısının çok olduğunu takdir edebilmelidir. Bu denklemelerin tümünü karşılayan bir kombinatoryal devreyi uygulama görevi son derece zor hale gelir. Sonuç olarak, genellikle *mikro programlama* olarak bilinen çok daha basit bir yaklaşım kullanılır. Bu bir sonraki bölümün konusudur.

## 20.4 ANAHTAR TERİMLER, GÖZDEN GEÇİRME SORULARI VE PROBLEMLER

### Anahtar Terimler

kontrol veriyolu kontrol yolu	kontrol sinyali kontrol ünitesi	kablolu uygulama mikro- işlemleri
-------------------------------------	------------------------------------	--------------------------------------

### İnceleme Soruları

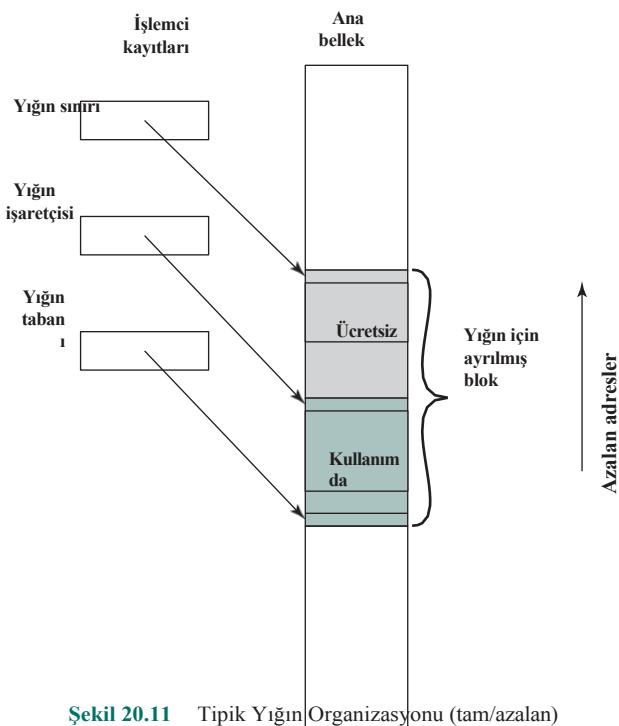
- 20.1** Bir talimatın yazılısı ile zaman sırası arasındaki farkı açıklayınız.
- 20.2** Talimatlar ve mikro-işlemler arasındaki ilişki nedir?
- 20.3** Bir işlemcinin kontrol biriminin genel işlevi nedir?
- 20.4** Kontrol ünitesinin karakterizasyonuna götüren üç adımlı bir süreci ana hatlarıyla belirtin.
- 20.5** Bir kontrol ünitesi hangi temel görevleri yerine getirir?
- 20.6** Bir kontrol ünitesinin giriş ve çıkışlarının tipik bir listesini veriniz.
- 20.7** Üç tip kontrol sinyalini listeleyiniz.
- 20.8** Bir kontrol ünitesinin kablolu uygulaması ile ne kastedildiğini kısaca açıklayınız.

### Problemler

- 20.1** ALU'nuz iki giriş kaydını toplayabilir ve herhangi bir giriş kaydının mantıksal olarak tamamlayabilir, ancak çıkışma yapamaz. Sayilar ikili komplement gösteriminde saklanacaktır. Kontrol ünitenizin bir çıkışma işlemine neden olmak için gerçekleştirmesi gereken mikro işlemleri listeleyin.

## 728 BÖLÜM 20 / KONTROL ÜNİTESİNİN ÇALIŞTIRILMASI

- 20.2** Aşağıdaki talimatlar için mikro işlemleri ve kontrol sinyallerini Şekil 20.5'teki işlemci için Tablo 20.1 ile aynı şekilde gösterin:
- Yük Akümülatörü
  - Mağaza Akümülatörü
  - Akümülatöre Ekle
  - VE Akümülatöre
  - Atlama
  - AC= 0 ise atlayın
  - Tamamlayıcı Akümülatör
- 20.3** Şekil 20.6'da veri yolu boyunca ve ALU boyunca yayılma gecikmesinin sırasıyla 20 ve 100 ns olduğunu varsayalım. Bir yazmacın veri yolundan veri kopyalaması için gereken süre 10 ns'dir. Aşağıdakiler için izin verilmesi gereken süre nedir?
- bir kayıttan diğerine veri aktarımı?
  - program sayacı?
- 20.4** Şekil 20.6'daki veri yolu yapısının AC'ye bir sayı eklemesi için gerekli mikro-işlem sırasını yazınız.
- acil işlenen;
  - doğrudan adres operandı;
  - dolaylı adres işleneni.
- 20.5** Şekil 20.11'de gösterildiği gibi bir yiğin uygulanmıştır (yiğinlar hakkında bir tartışma için Ek I'e bakınız). Aşağıdakiler için mikro işlemlerin sırasını gösterin
- haşhaş;
  - Yiğin.



Şekil 20.11 Tipik Yiğin Organizasyonu (tam/azalan)



## BÖLÜM

# 21

# MİKROPROGRAMLI KONTROL

### 21.1 Temel Kavramlar

- Mikroinstructions Mikro Programlı
- Kontrol Ünitesi Wilkes Kontrol
- Avantajlar ve Dezavantajlar

### 21.2 Mikro Talimat Sıralaması

- Tasarımla İlgili Hususlar
- Sıralama Teknikleri Adres
- Üretilimi
- LSI-11 Mikro Komut Sıralaması

### 21.3 Mikro Komut Yürütme

- Mikro Talimatların Taksonomisi Mikro Talimat Kodlaması
- LSI-11 Mikro Komut Yürütme IBM 3033
- Mikro Komut Yürütme

### 21.4 TI 8800

- Mikro Komut Formati Mikro
- Sıralayıcı Kayıtlı ALU

### 21.5 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler

### ÖĞRENİM HEDEFLERİ

Bu bölümü çalıştaktan sonra şunları yapabilmelisiniz:

- 『 Mikro programlı kontrolün temel kavramlarına genel bir bakış sunmak.
- 『 Kablolu kontrol ile mikro programlı kontrol arasındaki farkı anlamak.
- 『 Sıralama tekniklerinin temel kategorilerini tartışınız.
- 『 Mikro yapıların taksonomisine genel bir bakış sunmak.

*Mikro program* terimi ilk olarak 1950'lerin başında M. V. Wilkes tarafından ortaya atılmıştır [WILK51]. Wilkes, kontrol ünitesi tasarımları için organize ve sistematik bir yaklaşım önermiş ve kablolu bir uygulamanın karmaşıklıklarından kaçınmıştır. Bu fikir birçok araştırmacının ilgisini çekmiş ancak hızlı ve nispeten ucuz bir kontrol belleği gerektireceği için uygulanamaz görünmüştür.

Mikro programlama sanatının durumu *Datamation* tarafından Şubat 1964 sayısında gözden geçirilmiştir. O tarihte hiçbir mikro programlı sistem yaygın olarak kullanılmıyordu ve makalelerden biri [HILL64] mikro programlanmanın geleceğinin "biraz bulanık olduğu" şeklindeki o zamanki popüler görüşü özetliyordu. Büyük üreticilerin hiçbirini bu teknigue ilgi göstermemiştir, ancak muhtemelen hepsi bu teknigi incelemiştir."

Bu durum birkaç ay içinde dramatik bir şekilde değişti. IBM'in System/360'ı Nisan ayında duyuruldu ve en büyük modeller hariç hepsi mikro programlandı. Her ne kadar 360 serisi yarı iletken ROM'un bulunmasından önce ortaya çıkmış olsa da, mikro programlanmanın avantajları IBM'in bu hamleyi yapması için yeterince zorlayıcıydı. Mikro programlama, CISC işlemcilerin kontrol birimini uygulamak için popüler bir teknik haline geldi. Son yıllarda, mikro daha az kullanılır hale geldi ancak bilgisayar tasarımcıları için kullanılabilir bir araç olmaya devam ediyor. Örneğin, Pentium 4'te gördüğümüz gibi, makine talimatları RISC benzeri bir formata dönüştürüller ve bunların çoğu mikro programlama kullanılmadan yürütülür. Ancak, bazı talimatlar mikro programlama kullanılarak yürütülür.

## 21.1 TEMEL KAVRAMLAR

### Mikro Talimatlar

Kontrol ünitesi oldukça basit bir cihaz gibi görünmektedir. Bununla birlikte, bir kontrol ünitesini temel mantık elemanlarının birbirine bağlanması olarak uygulamak kolay bir iş değildir. Tasarım, mikro-işlemler arasında sıralama yapmak, mikro-işlemleri yürütmek, işlem kodlarını yorumlamak ve ALU bayraklarına dayalı kararlar almak için mantık içermelidir. Böyle bir donanım parçasını tasarlamak ve test etmek zordur. Dahası, tasarım nispeten esnek değildir. Örneğin, yeni bir makine komutu eklemek istediğiinde tasarım değiştirmek zordur.

Birçok CISC işlemcisinde kullanılan bir alternatif, **mikro programlanmış** bir **kontrol birimi** uygulamaktır.

Tablo 21.1'i göz önünde bulundurun. Kontrol sinyallerinin kullanımına ek olarak, her bir mikro işlem sembolik gösterimle tanımlanmıştır. Bu gösterim şüpheli bir şekilde bir programlama diline benzemektedir. Aslında **mikro programlama dili** olarak bilinen bir dildir. Her satır, bir seferde gerçekleşen bir dizi mikro işlemi tanımlar ve **mikro talimat olarak** bilinir. Bir talimatlar dizisi **mikro program** ya da *aygit yazılımı* olarak bilinir. Bu son terim, bir mikro programın donanım ve yazılım arasında bir yerde olduğu gerçekini yansıtmaktadır. Donanım yazılımında tasarım yapmak donanımdan daha kolaydır, ancak bir donanım yazılımı programı yazmak bir yazılım programı yazmaktan daha zordur.

Mikro programlama kavramını bir kontrol ünitesini uygulamak için nasıl kullanabiliriz? Her bir mikro-işlem için kontrol ünitesinin yapmasına izin verilen tek şeyin bir dizi kontrol sinyali üretmek olduğunu düşünün. Böylece, herhangi bir mikro-işlem için, kontrol ünitesinden çıkan her bir kontrol hattı ya açık ya da kapalıdır. Bu durum elbette her bir kontrol hattı için ikili bir rakamla temsil edilebilir. Böylece her bitin bir *kontrol* hattını temsil ettiği bir *kontrol kelimesi* oluşturabiliriz. Bu durumda her mikro-işlem kontrol kelimesinde farklı 1 ve 0'larla temsil edilecektir.

Kontrol birimi tarafından gerçekleştirilen mikro-işlemlerin sırasını temsil etmek için bir dizi kontrol kelimesini bir araya getirdiğimizi varsayıyalım. Daha sonra, mikro işlem sırasının sabit olmadığını kabul etmeliyiz. Bazen dolaylı bir döngüye sahibizdir; bazen de değilizdir. Bu yüzden kontrol sözcüklerimizi, her sözcüğün benzersiz bir adresi olacak şekilde bir belleğe koyalım. Şimdi her kontrol sözcüğüne, belirli bir koşulun doğru olması halinde (örneğin, bir bellek referans komutundaki dolaylı bitin 1 olması) yürütülecek bir sonraki kontrol sözcüğünün konumunu belirten bir adres alanı ekleyin. Ayrıca, koşulu belirtmek için birkaç bit ekleyin.

**Tablo 21.1** Wilkes Örneği için Makine Komut Seti

Sipariş	Siparişin Etkisi
<i>A n</i>	$C(Acc) + C(n) - Acc_1$
<i>S n</i>	$C(Acc) - C(n) - Acc_1$
<i>H n</i>	$C(n) - Acc_2$
<i>V n</i>	$C(Acc2) * C(n)$ to $Acc$ , burada $C(n) \geq 0$
<i>T n</i>	$C(Acc_{(1)})$ ila $n$ , 0 ila $Acc$
<i>U n</i>	$C(Acc_1)$ ila $n$
<i>R n</i>	$C(Acc) * 2^{(n+1)}$ ila $Acc$
<i>L n</i>	$C(Acc) * 2^{n+1}$ ila $Acc$
<i>G n</i>	EĞER $C(Acc) \geq 0$ ise, kontrolü $n$ 'ye aktar; eğer $C(Acc) < 0$ ise, göz ardı et (yani, seri olarak ilerleyin)
<i>I n</i>	Giriş mekanizmasındaki bir sonraki karakteri $n$ 'ye oku
<i>O n</i>	$C(n)$ 'yi çıkış mekanizmasına gönderin

Notasyon:  $Acc$ = akümülatör

$Acc_1$ = akümülatörün en anlamlı yarısı  $Acc_2$ =

akümülatörün en az anlamlı yarısı  $n$ = depolama konumu  $n$

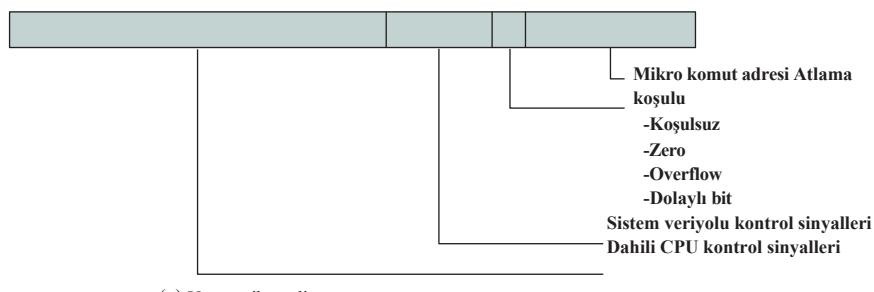
$C(X)$ =  $X$ 'in içeriği ( $X$ = kayıt veya depolama konumu)

Sonuç **yatay mikro komut** olarak bilinir bir örneği Şekil 21.1a'da gösterilmiştir. Mikro komut veya kontrol sözcüğünün biçimini aşağıdaki gibidir. Her dahili işlemci kontrol hattı için bir bit ve her sistem veri yolu kontrol hattı için bir bit vardır. Bir dallanma olması gereken koşulu belirten bir koşul alanı ve bir dallanma olduğunda bir sonraki çalıştırılacak mikro komutun adresini içeren bir vardır. Böyle bir mikro komut aşağıdaki gibi yorumlanır:

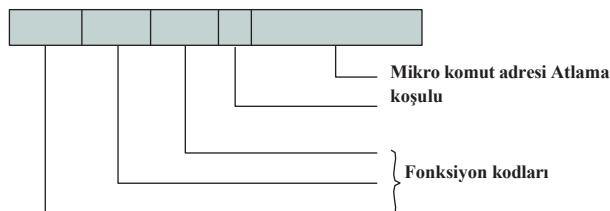
1. Bu mikro talimatı yürütmemek için, 1 biti ile gösterilen tüm kontrol hatlarını açın; 0 ile gösterilen tüm kontrol hatlarını kapalı bırakın. Elde edilen kontrol sinyalleri bir veya daha fazla mikro işlemin gerçekleştirilemesine neden olacaktır.
2. Koşul bitleri tarafından belirtilen koşul yanlışsa, sırayla bir sonraki mikro komutu çalıştırın.
3. Koşul bitleri tarafından belirtilen koşul doğrusa, yürütülecek bir sonraki mikro yapılandırma adres alanında belirtilir.

Şekil 21.2'de bu kontrol kelimelerinin veya mikro komutların bir **kontrol belleğinde** nasıl düzenlenebileceği gösterilmektedir. Her bir rutindeki mikro talimatlar sırayla çalıştırılacaktır. Her rutin, bir sonraki adının nereye gideceğini gösteren bir dallanma komutu ile sona erer. Tek amacı, mevcut işlem koduna bağlı olarak makine komutu rutinlerinden birinin (AND, ADD, vb.) daha sonra belirtmek olan özel bir yürütme döngüsü rutini vardır.

Şekil 21.2'deki kontrol belleği, kontrol ünitesinin tüm çalışmasının kısa bir açıklamasıdır. Yapılacak mikro işlemlerin sırasını tanımlar

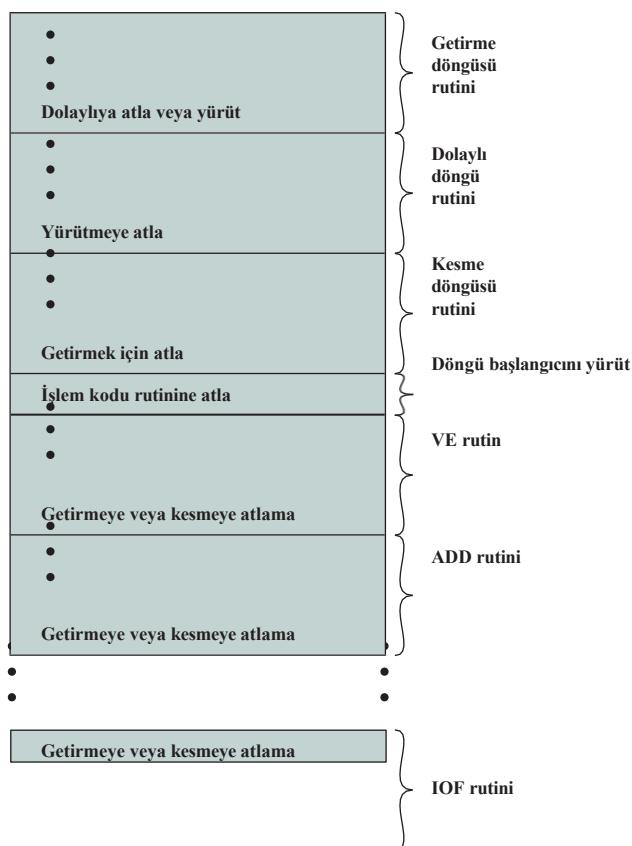


(a) Yatay mikro talimat



(b) Dikey mikro öğretim

**Şekil 21.1** Tipik Mikro Komut Formatları



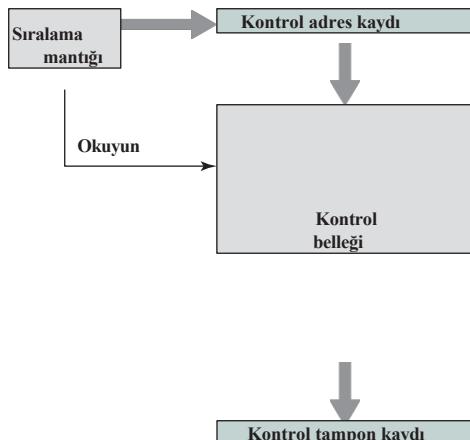
**Sekil 21.2** Kontrol Belleği Organizasyonu

Her döngü (getirme, dolaylı, yürütme, kesme) sırasında gerçekleştirilir ve bu döngülerin sıralamasını belirtir. Başka hiçbir şey olmasa bile, bu gösterim belirli bir bilgisayarın kontrol ünitesinin işleyişini belgelemek için kullanışlı bir araç olacaktır. Ancak bundan daha fazlasıdır. Aynı zamanda kontrol birimini uygulamanın bir yoludur.

Mikroprogramlı Kontrol Ünitesi

Şekil 21.2'deki kontrol belgesi, kontrol biriminin davranışını tanımlayan bir program içerir. Kontrol birimini sadece bu programı çalıştırarak uygulayabileceğimiz sonucu çıkar.

Şekil 21.3 böyle bir uygulamanın temel unsurlarını göstermektedir. Mikro komutlar kümlesi *kontrol belleğinde* saklanır. *Kontrol adres kaydi*, okunacak bir sonraki mikro komutun adresini tutar. Bir mikro komut kontrol belleğinden okunduğunda, bir *kontrol tampon yazmacına* aktarılır. Bu yazmacın sol kısmı (bkz. Şekil 21.1a) kontrol ünitesinden gelen kontrol hatlarına bağlanır. Böylece, kontrol belleğinden bir mikro komutun *okunması*, o mikro komutun *yürüttülmesiyle* aynı şeydir. Şekilde gösterilen üçüncü eleman, kontrol adres kaydını yükleyen ve bir okuma komutu yeren bir sıralama birimidir.

**Şekil 21.3** Kontrol Birimi Mikromimarisi

Şekil 21.4'te gösterildiği gibi bu yapıyı daha ayrıntılı olarak inceleyelim. Bunu Şekil 21.3 ile karşılaştırıldığımızda, kontrol biriminin hala aynı girişlere (IR, ALU bayrakları, saat) ve çıkışlarına (kontrol sinyalleri) sahip olduğunu görürüz. Kontrol birimi aşağıdaki gibi çalışır:

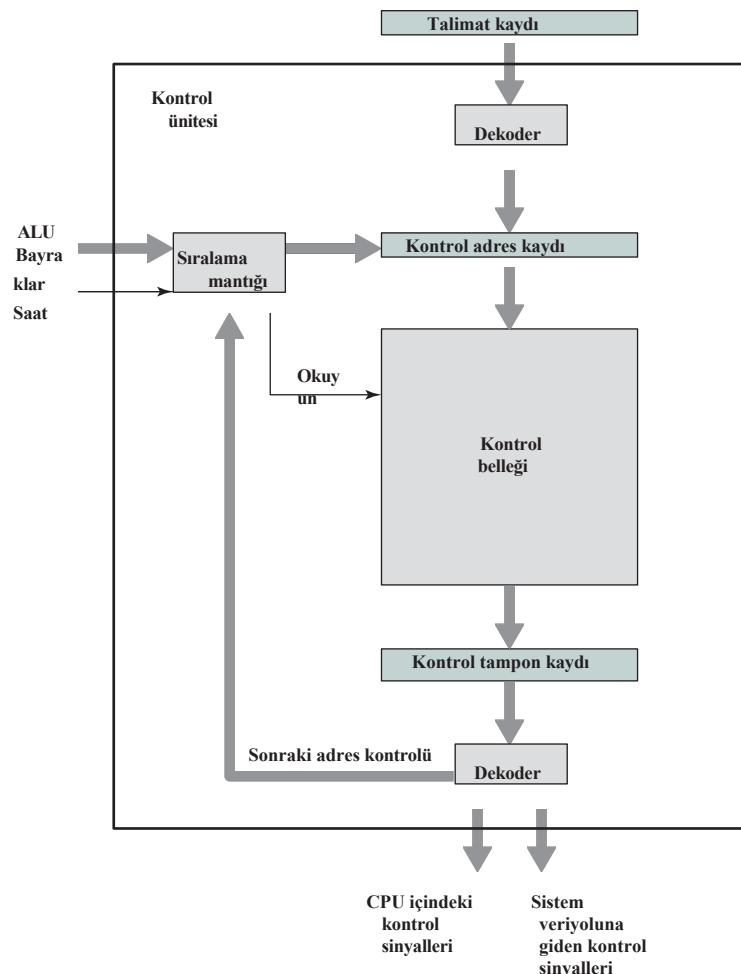
1. Bir talimatı yürütmem için sıralama mantık birimi kontrol belleğine bir OKU komutu verir.
2. Adresi kontrol adres kaydında belirtilen kelime kontrol tampon kaydına okunur.
3. Kontrol tampon yazmacının içeriği, sıralama mantık birimi için kontrol sinyalleri ve sonraki adres bilgileri üretir.
4. Sıralama mantık birimi, kontrol yazmacından gelen sonraki adres bilgisine ve ALU bayraklarına dayanarak kontrol adres yazmacına yeni bir adres yükler.

Tüm bunlar bir saat darbesi sırasında gerçekleşir.

Az önce listelenen son adının detaylandırılması gerekmektedir. Her mikro komutun sonunda, sıralama mantık birimi kontrol adres kaydına yeni bir adres yükler. ALU bayraklarının ve kontrol tampon yazmacının değerine bağlı olarak üç karardan biri verilir:

- **Sonraki komutu alın:** Kontrol adresi kaydına 1 ekleyin.
- **Bir atlama mikro talimatına dayalı olarak yeni bir rutine atlama:** Kontrol tampon yazmacının adres alanını kontrol adres yazmacına yükleyin.
- **Bir makine komutu rutinine atlayın:** IR'deki işlem koduna bağlı olarak kontrol adres kaydını yükleyin.

Şekil 21.4 kod çözücü olarak etiketlenmiş iki modülü göstermektedir. Üst kod çözücü IR'nin işlem kodunu bir kontrol bellek adresine dönüştürür. Alt kod çözücü yatay mikro talimatlar için kullanılmaz ancak **dikey mikro talimatlar** için kullanılır (Şekil 21.1b). Daha önce de belirtildiği gibi, yatay bir mikro komutta her bit



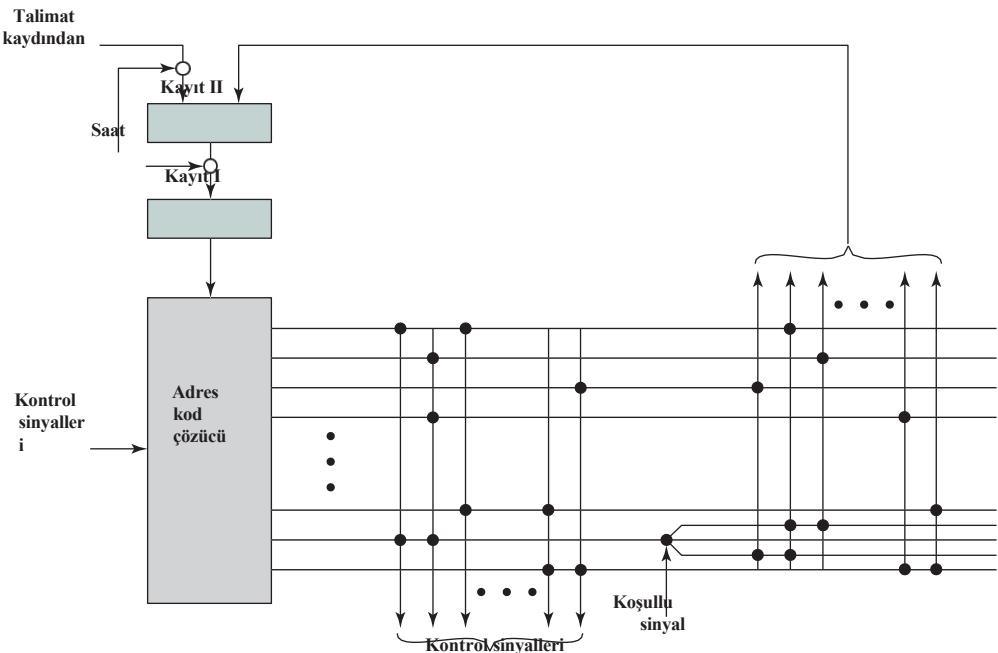
**Şekil 21.4** Mikro Programlı Kontrol Ünitesinin Çalışması

kontrol alanı bir kontrol hattına bağlanır. Dikey bir mikro talimatta, gerçekleştirilecek her eylem için bir kod kullanılır [örneğin, MAR **d** (PC)] ve kod çözücü bu kodu ayrı kontrol sinyallerine çevirir. Dikey mikro talimatların avantajı, az miktarda ek mantık ve zaman gecikmesi pahasına yatay mikro talimatlara göre daha kompakt (daha az bit) olmalarıdır.

### Wilkes Kontrol

Daha önce de belirtildiği gibi, Wilkes ilk olarak 1951 yılında mikro programlı bir kontrol ünitesinin kullanılmasını önermiştir [WILK51]. Bu öneri daha sonra daha ayrıntılı bir tasarıma dönüştürülmüştür [WILK53]. Bu ufuk açıcı öneriyi incelemek öğretici olacaktır.

Wilkes tarafından önerilen konfigürasyon Şekil 21.5'te gösterilmektedir. Sistemin kalbi kısmen dijitalerle dolu bir matristir. Bir makine döngüsü sırasında, matrisin bir satırı bir darbe ile etkinleştirilir. Bu, bir dijitalın bulunduğu noktalarda sinyaller üretir (diagramda bir nokta ile gösterilmiştir). Sıranın ilk kısmı işlemcinin çalışmasını kontrol eden kontrol sinyallerini üretir. İkinci kısmı, işlemcinin çalışmasını



Şekil 21.5 Wilkes'in Mikro Programlı Kontrol Ünitesi

Bir sonraki makine çevriminde darbelenecek satırın adresi. Böylece, matrisin her satırı bir mikro komuttur ve matrisin düzeni kontrol belleğidir.

Döngünün başında, darbelenecek satırın adresi Register I'de bulunur. Bu adres, bir saat darbesi ile etkinleştirildiğinde matrisin bir satırını etkinleştiren kod çözücünün girişidir. Kontrol sinyallerine bağlı olarak, komut yazmacındaki işlem kodu veya darbeli satırın ikinci kısmı döngü sırasında Yazmaç II'ye aktarılır. Kayıt II daha sonra bir saat darbesi ile Kayıt I'e geçilir. Dönüşümlü saat darbeleri matrisin bir satırını etkinleştirmek ve Register II'den Register I'e aktarmak için kullanılır. İki kayıt düzlenmesi gereklidir çünkü kod çözücü basitçe bir kombinasyon devresidir; yalnızca bir kayıtla, çıkış bir döngü sırasında giriş haline gelir ve kararsız bir duruma neden olur.

Bu şema daha önce açıklanan yatay mikro programlama yaklaşımına çok benzemektedir (Şekil 21.1a). Temel fark şudur: Önceki, kontrol adres kaydı bir sonraki adresi almak için bir artırılabiliriyordu. Wilkes şemasında, bir sonraki adres mikro yapıda bulunur. Dallanmaya izin vermek için, bir satır şekilde gösterildiği gibi koşullu bir sinyal (örneğin bayrak) tarafından kontrol edilen iki adres parçası içermelidir.

Wilkes bu şemayı önerdiğinden sonra, basit bir makinenin kontrol birimini uygulamak için kullanımının bir örneğini sunar. Mikro programlanmış bir işlemcinin bilinen ilk tasarımını olan bu örnek, mikro programlamadan çağdaş ilkelerinin çoğunu gösterdiği için burada tekrarlanmaya değerdir.

Varsayımsal makinenin işlemcisi (Wilkes'in örnek makinesi) aşağıdaki kayıtları içerir:

A	Çarpım
B	Akümülatör (en az anlamlı yarı)
C	Akümülatör (en anlamlı yarı)
D	Kaydirmalı kayıt

Buna ek olarak, sadece kontrol ünitesinin erişebildiği üç kayıt ve iki adet 1-bit bayrak bulunmaktadır. Kayıtlar aşağıdaki gibidir:

E	Hem bellek adres kaydı (MAR) hem de geçici depolama alanı olarak hizmet verir
F	Program sayacı
G	Başka bir geçici kayıt; sayımla kullanılır

Tablo 21.1 bu örnek için makine komut setini listelemektedir. Tablo 21.2, kontrol ünitesini uygulayan ve sembolik biçimde ifade edilen mikro talimatların tam setidir. Böylece, sistemi tamamen tanımlamak için toplam 38 mikro komut gereklidir.

İlk tam sütun her mikro komutun adresini (satır numarası) verir. İşlem kodlarına karşılık gelen adresler etiketlenmiştir. Böylece, toplama komutu (A) için işlem koduyla karşılaşıldığında, 5 konumundaki mikro komut çalıştırılır. Sütun 2 ve 3 sırasıyla ALU ve kontrol birimi tarafından gerçekleştirilecek eylemleri ifade eder. Her sembolik ifade bir dizi kontrol sinyaline (mikro komut bitleri) çevrilmiştir. Sütun 4 ve 5, iki bayrağın (flip-flop) ayarlanması ve kullanılmasıyla ilgilidir. Sütun 4 bayrağı ayarlayan sinyali belirtir. Örneğin, (1)C<sub>S</sub>, 1 numaralı bayrağın reg- ister C'deki sayının işaret biti tarafından ayarlandığı anlamına gelir. 5. sütun bir bayrak tanımlayıcısı içermiyorsa, 6. ve 7. sütunlar kullanılacak iki alternatif mikro komut adresini içerir. Aksi takdirde, 6. sütun getirilecek bir sonraki mikro komutun adresini belirtir.

0'dan 4'e kadar olan komutlar getirme döngüsünü oluşturur. Mikro komut 4, işlem kodunu bir kod çözücüye sunar, bu da getirilecek makine komutuna karşılık gelen bir mikro komutun adresini oluşturur. Okuyucu Tablo 21.2yi dikkatle inceleyerek kontrol ünitesinin tüm işleyişini anlayabilmelidir.

## Avantajlar ve Dezavantajlar

Bir kontrol ünitesini uygulamak için mikro programlamanın kullanılmasının temel avantajı, kontrol ünitesinin tasarımını basitleştirmesidir. Böylece hem daha ucuz hem de daha az hataya açıktır. *Kablolu* bir kontrol ünitesi, komut döngüsünün birçok mikro işlemi boyunca sıralama yapmak için karmaşık bir mantık içermelidir. Öte yandan, mikro programlı bir kontrol ünitesinin kod çözüçüleri ve sıralama mantık birimi çok basit mantık parçalarıdır.

Mikro programlanmış bir birimin temel dezavantajı, benzer teknolojiye sahip kablolu bir birimden biraz daha yavaş olmasıdır. Buna rağmen, mikro programlama, uygulama kolaylığı nedeniyle saf CISC mimarilerinde kontrol birimlerini uygulamak için baskın tekniktir. RISC işlemcileri, daha basit komut форматları ile tipik olarak kablolu kontrol birimleri kullanırlar. Şimdi mikro programlama yaklaşımını daha ayrıntılı olarak inceleyeceğiz.

## 738 BÖLÜM 21 / MİKROPROGRAMLI KONTROL

**Tablo 21.2** Wilkes Örneği için Mikro İstrüksyonlar

Notasyonlar:  $A, B, C, \dots$  aritmetik ve kontrol kayıt birimlerindeki çeşitli kayıtları temsil eder.  $C$  ile  $D$  şunları gösterir anahtarlama devrelerinin  $C$  yazmacının çıkışını  $D$  giriş yazmacına bağladığını;  $(D+ A) C$ 'ye,  $A$ 'nın çıkış yazmacının toplama ünitesinin bir girişine ( $D$ 'nin çıkışı kalıcı olarak diğer bağlıdır) ve toplayıcının çıkışının  $C$  yazmacına bağlandığını gösterir. Tırnak içindeki bir  $n$  sayısal simbolü (örneğin, "n"), çıkış en az anlamlı basamak birimi cinsinden  $n$  sayısı olan kaynağı ifade eder.

		Aritmetik Birim	Kontrol Kayıt Birimi	Koşullu Flip-Flop		Sonraki Mikro Öğretim	
				Set	Kullanım	0	1
	0		$F'$ den $G'$ ye ve $E'$ ye			1	
	1		$(G'$ den "1" e) $F'$ ye			2	
	2		$G$ ye depolayın			3	
	3		$G'$ den $E'$ ye			4	
	4		$E'$ den kod çözücüye			-	
$A$	5	$C$ 'den $D$ 'ye				16	
$S$	6	$C$ 'den $D$ 'ye				17	
$H$	7	$B$ 'ye depolayın				0	
$V$	8	$A$ 'ya depolayın				27	
$T$	9	$C$ 'den Depoya				25	
$U$	10	$C$ 'den Depoya				0	
$R$	11	$B$ 'den $D$ 'ye	$E$ 'den $G$ 'ye			19	
$L$	12	$C$ 'den $D$ 'ye	$E$ 'den $G$ 'ye			22	
$G$	13		$E$ 'den $G$ 'ye	(1) $E_5$		18	
$I$	14	Depoya Giriş				0	
$O$	15	Çıkış Sakla				0	
	16	$(D+ \text{Store})$ ile $C$				0	
	17	$(D - \text{Store})$ ile $C$				0	
	18				1	0	1
	19	$D$ 'den $B$ 'ye ( $R$ )*	$(G - 1)$ ile $E$			20	
	20	$C$ 'den $D$ 'ye		(1) $E_5$		21	
	21	$D$ 'den $C$ 'ye ( $R$ )			1	11	0
	22	$D$ 'den $C$ 'ye ( $L$ )†	$(G - 1)$ ile $E$			23	
	23	$B$ 'den $D$ 'ye		(1) $E_5$		24	
	24	$D$ 'den $B$ 'ye ( $L$ )			1	12	0
	25	$B$ 'ye "0"				26	
	26	$B$ 'den $C$ 'ye				0	
	27	$C$ 'ye "0"	"18" ila $E$			28	
	28	$B$ 'den $D$ 'ye	$E$ 'den $G$ 'ye	(1) $B_1$		29	
	29	$D$ 'den $B$ 'ye ( $R$ )	$(G - "1")$ ile $E$			30	
	30	$C$ 'den $D$ 'ye ( $R$ )		(2) $E_5$	1	31	32

		Aritmetik Birim	Kontrol Kayıt Birimi	Koşullu Flip-Flop		Sonraki Mikro Öğretim	
				Set	Kullanım	0	1
31	$D$ 'den $C$ 'ye				2	28	33
32	$(D + A)$ ila $C$				2	28	33
33	$B$ 'den $D$ 'ye			(1) $B_1$		34	
34	$D$ 'den $B$ 'ye ( $R$ )					35	
35	$C$ 'den $D$ 'ye ( $R$ )				1	36	37
36	$D$ 'den $C$ 'ye					0	
37	$(D - A)$ ila $C$					0	

\* Sağ kaydırma. Aritmetik birimdeki anahtarlama devreleri, sağ kaydırma mikro işlemleri sırasında  $C$  yazmacının en az anlamlı basamağı  $B$  yazmacının en anlamlı yerine yerleştirilecek ve  $C$  yazmacının en anlamlı basamağı ( işaret ) tekrarlanacak şekilde düzenlenmiştir (böylece negatif sayılar için düzeltme yapılır).

† Sola kaydırma. Anahtarlama devreleri benzer şekilde sola kaydırma mikro işlemleri sırasında  $B$  yazmacının en anlamlı basamağını  $C$  yazmacının en az anlamlı yerine geçirecek şekilde düzenlenmiştir.

## 21.2 MIKRO TALIMAT SIRALAMASI

Mikro programlı bir kontrol ünitesi tarafından gerçekleştirilen iki temel görev aşağıdaki gibidir:

- **Mikro komut sıralaması:** Kontrol belleğinden bir sonraki mikro komutu alın.
- **Mikro komut yürütme:** Mikro komutu yürütmek için gereken kontrol sinyallerini üretin.

Bir kontrol birimi tasarlarken, bu görevler birlikte düşünülmelidir, çünkü her ikisi de mikro komutun formatını ve kontrol biriminin zamanlamasını etkiler. Bu bölümde, sıralamaya odaklanacağız ve format ve zamanlama konuları hakkında mümkün olduğunda az şey söyleyeceğiz. Bu konular bir sonraki bölümde daha ayrıntılı olarak inceleneciktir.

### Tasarım Hususları

Bir mikro komut sıralama tekniğinin tasarımında iki kaygı söz konusudur: mikro komutun boyutu ve adres oluşturma süresi. İlk kaygı açıkta; kontrol belleğinin boyutunu en aza indirmek bu bileşenin maliyetini azaltır. İkinci kaygı ise mikro komutları mümkün olduğunda hızlı yürütme isteğidir.

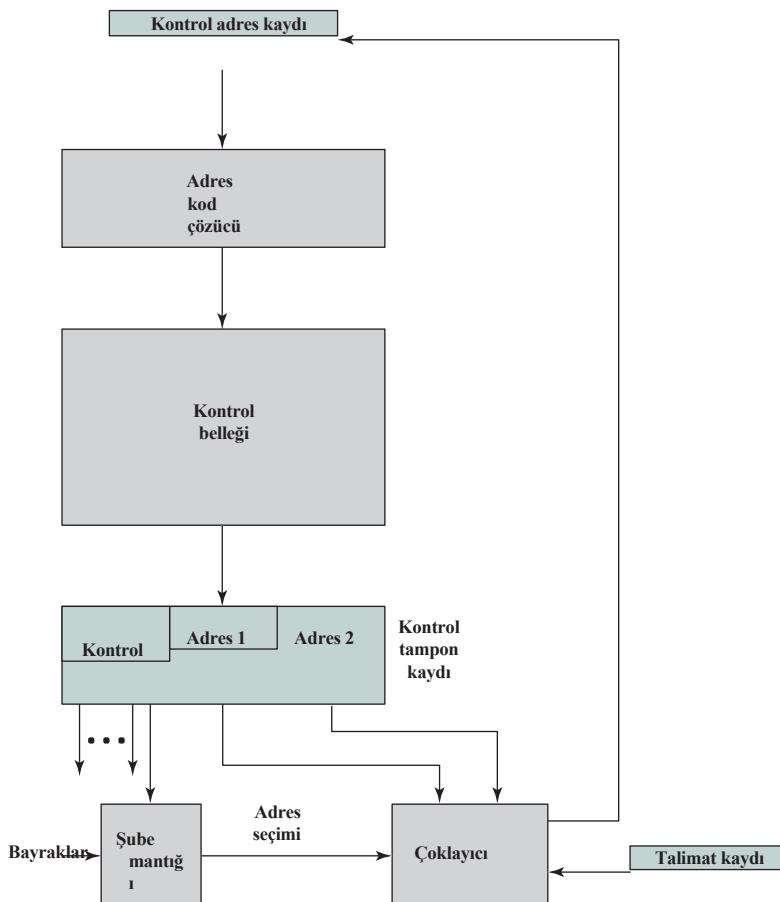
Bir mikro program yürütülürken, yürütülecek bir sonraki mikro komutun adresi bu kategorilerden birinde yer alır:

- Talimat kaydı tarafından belirlenir
- Sonraki sıralı adres
- Şube

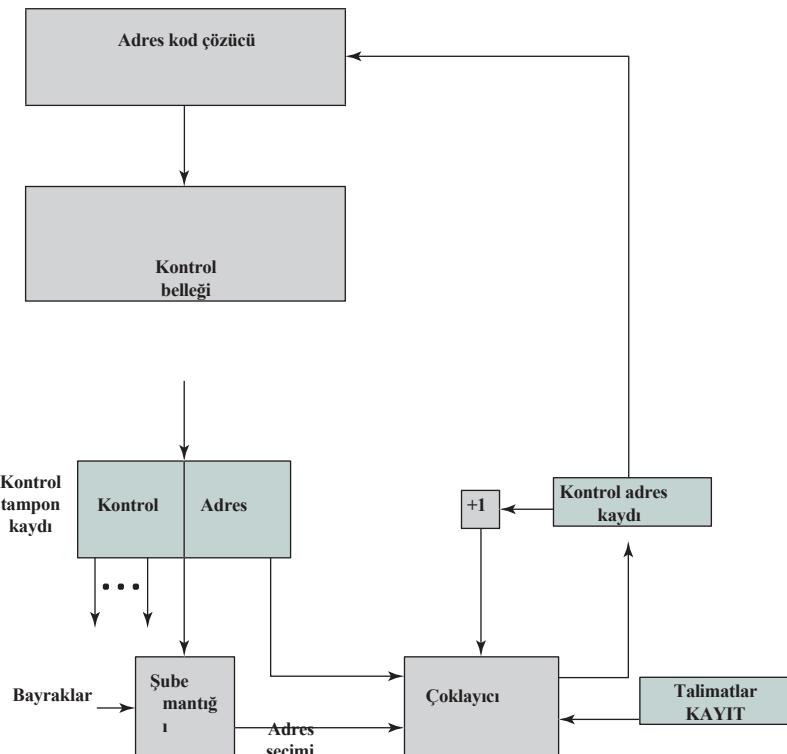
İlk kategori her komut döngüsünde yalnızca bir kez, bir komut alındıktan hemen sonra İkinci kategori çoğu tasarımda en yaygın olanıdır. Ancak, tasarım sadece sıralı erişim için optimize edilemez. Hem koşullu hem de koşulsuz dallanmalar mikro programın gerekli bir parçasıdır. Ayrıca, mikro yapı dizileri kısa olma eğilimindedir; her üç veya dört mikro yapıdan biri dallanma olabilir [SIEW82]. Bu nedenle, mikro komut dallanması için kompakt, zaman açısından verimli teknikler tasarlamak önemlidir.

### Sıralama Teknikleri

Mevcut mikro komuta, durum bayraklarına ve komut kaydının içeriğine bağlı olarak, bir sonraki mikro komut için bir kontrol belgesi adresi oluşturulmalıdır. Çok çeşitli teknikler kullanılmıştır. Bunları Şekil 21.6 ile 21.8'de gösterildiği gibi üç genel kategoride gruplayabiliz. Bu kategoriler mikro komuttaki adres bilgisinin formatına dayanmaktadır:



**Şekil 21.6** Dal Kontrol Mantığı: İki Adres Alanı



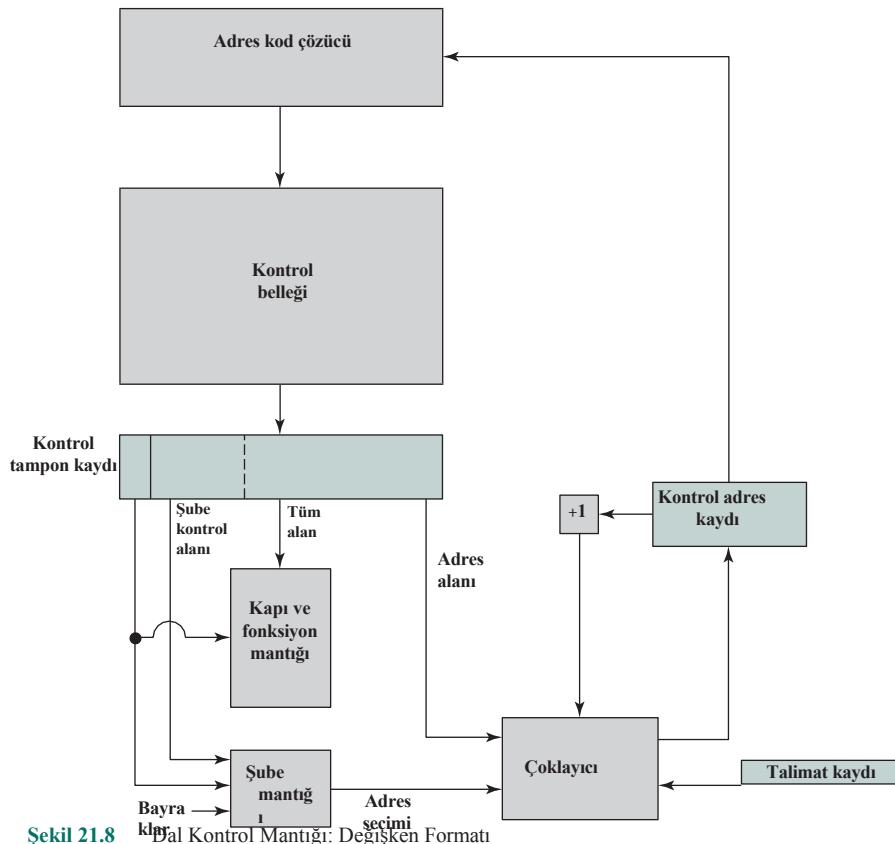
**Şekil 21.7** Dal Kontrol Mantığı: Tek Adres Alanı

- İki adres alanı
- Tek adres alanı
- Değişken format

En basit yaklaşım, her mikro yapıda iki adres alanı sağlamaktır. Şekil 21.6 bu bilginin nasıl kullanılacağını göstermektedir. Her iki adres alanı artı komut kaydı için bir hedef olarak hizmet veren bir çoklayıcı sağlanır. Bir adres seçimi girişine bağlı olarak, çoklayıcı ya işlem kodunu ya da iki adresten birini kontrol adres yazmacına (CAR) ileter. CAR daha sonra bir sonraki mikro komut adresini üretmek için deşifre edilir. Adres seçim sinyalleri, giriş kontrollü bayrakları artı mikro komutun kontrol kısmından bitlerden oluşan bir dallanma mantık modülü tarafından sağlanır.

İki adresli yaklaşım basit olmasına rağmen, mikro komutta diğer yaklaşılara göre daha fazla bit gerektirir. Bazı ek mantıklarla tasarruf. Yaygın bir yaklaşım tek bir adres alanına sahip olmaktadır (Şekil 21.7). Bu yaklaşımımla, bir sonraki adres için seçenekler aşağıdaki gibidir:

- Adres alanı
- Talimat kayıt kodu
- Sonraki sıralı adres



Adres seçim sinyalleri hangi seçenekin seçileceğini belirler. Bu yaklaşım adres alanlarının sayısını bire indirir. Bununla birlikte, adres alanının genellikle kullanılmayacağını unutmayın. Bu nedenle, mikro komut kodlama şemasında bir miktar verimsizlik vardır.

Diğer bir yaklaşım ise tamamen farklı iki mikro komut formatı sağlamaktır (Şekil 21.8). Bir bit hangi formatın kullanıldığını belirler. Bir formatta, kalan bitler kontrol sinyallerini etkinleştirerek için kullanılır. Diğer formatta, bazı bitler dallanma mantık modülünü çalıştırır ve kalan bitler adresi sağlar. İlk formatta, bir sonraki adres ya bir sonraki sıralı adresdir ya da komut yazmacından türetilen bir adresdir. İkinci formatta, koşullu ya da koşulsuz dallanma belirtilir. Bu yaklaşımın bir dezavantajı, her dallanma mikro komutu için bir çevrimin tamamının tüketilmesidir. Diğer yaklaşımarda, adres üretimi kontrol sinyali üretimi ile aynı döngünün bir parçası olarak gerçekleşir ve kontrol belleği erişimlerini en aza indirir.

Az önce açıklanan yaklaşımlar geneldir. Spesifik uygulamalar genellikle bu tekniklerin bir varyasyonunu veya kombinasyonunu içerecektir.

## Adres Üretimi

Sıralama problemine biçim ve genel mantık gereklilikleri açısından baktık. Bir başka bakış açısı da bir sonraki adresin türetilmesi ya da hesaplanabileceği çeşitli yolları göz bulundurmaktır.

Tablo 21.3 çeşitli adres üretme tekniklerini listeler. Bunlar, adresin mikro komutta açıkça mevcut olduğu açık teknikler ve adresi oluşturmak için ek mantık gerektiren örtük teknikler olarak ikiye ayırlabilir.

Biz esasen açık tekniklerle uğraştık. İki alanlı bir yaklaşımla, her mikro talimatta iki alternatif adres mevcuttur. Tek bir adres alanı ya da değişken bir format kullanılarak çeşitli dallanma talimatları uygulanabilir. Koşullu bir dallanma talimatı aşağıdaki bilgi türlerine bağlıdır:

- ALU bayrakları
- Makine komutunun işlem kodu veya adres modu alanlarının bir kısmı
- İşaret biti gibi seçilen bir kaydın parçaları
- Kontrol ünitesindeki durum bitleri

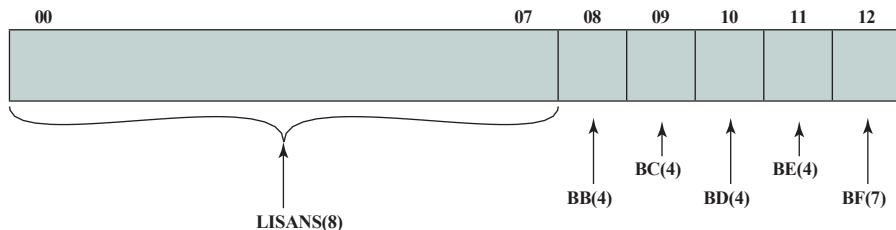
Birkaç örtük teknik de yaygın olarak kullanılmaktadır. Bunlardan biri olan eşleme, neredeyse tüm tasarımlarda gereklidir. Bir makine komutunun işlem kodu kısmı bir mikro komut adresine eşlenmelidir. Bu, talimat döngüsü başına yalnızca bir kez gerçekleşir.

Yaygın bir örtük teknik, tam adresi oluşturmak için bir adresin iki bölümünü birleştirmeyi veya eklemeyi içeren bir tekniktir. Bu yaklaşım IBM S/360 ailesi için benimsenmiş [TUCK67] ve birçok S/370 modelinde kullanılmıştır. Örnek olarak IBM 3033'ü kullanacağız.

IBM 3033 üzerindeki kontrol adres kaydı 13 bit uzunluğundadır ve Şekil 21.9'da gösterilmiştir. Adresin iki kısmı ayırt edilebilir. En üst sıradaki 8 bit (00-07) normalde bir mikro komut çevriminden diğerine değişmez. Bir mikro komutun yürütülmesi sırasında, bu 8 bit doğrudan mikro komutun 8 bitlik bir alanından (BA alanı) kontrol adres kaydının en yüksek sıralı 8 bitine kopyalanır. Bu, kontrol belleginde 32 mikro komuttan oluşan bir blok tanımlar. Kontrol adres kaydının kalan 5 biti, bir sonraki getirilecek mikro komutun özel adresini belirtmek için ayarlanır. Bu bitlerin her biri mevcut mikro yapılandırılmış 4 bitlik bir alan (biri hariç 7 bitlik bir ) tarafından belirlenir; alan, ilgili bitin ayarlanması için gerekli koşulu belirtir. Örneğin, kontrol adres yazmacındaki bir bit, son ALU işleminde bir carry oluşup oluşmadığına bağlı olarak 1 veya 0 olarak ayarlanabilir.

**Tablo 21.3** Mikro Komut Adres Oluşturma Teknikleri

Açık	Örtük
İki alanlı	Haritalama
Koşulsuz şube	İlave
Koşullu şube	Kalıntı kontrolü



Şekil 21.9 IBM 3033 Kontrol Adres Kaydı

Tablo 21.3'te listelenen son yaklaşım *artık kontrol* olarak adlandırılır. Bu yaklaşım, daha önce kontrol ünitesi içinde geçici depolamaya kaydedilmiş bir mikro komut adresinin kullanılmasını içerir. Örneğin, bazı mikroişlem setleri bir alt program olağlığı ile donatılmıştır. Dönüş adreslerini tutmak için dahili bir kayıt ya da kayıt yiğini kullanılır. Bu yaklaşımın bir örneği, şimdi incelemekte olduğumuz LSI-11'de yer almaktadır.

### LSI-11 Mikro Komut Sıralaması

LSI-11, PDP-11'in mikrobilgisayar versiyonudur ve sistemin ana bileşenleri tek bir kart üzerinde bulunur. LSI-11, mikro gramlı bir kontrol ünitesi kullanılarak uygulanmıştır [SEBE76].

LSI-11, 22 bitlik bir mikro komut ve 2K 22 bitlik sözcükten oluşan bir kontrol belleği kullanır. Bir sonraki mikro komut adresi beş yoldan biriyle belirlenir:

- **Sonraki sıralı adres:** Başka bir talimat olmadığından, kontrol ünitesinin kontrol adresi kaydı 1 artırılır.
- **Opcode eşlemesi:** Her komut döngüsünün başında, bir sonraki mikro yapı adresi opcode tarafından belirlenir.
- **Alt yordam olağlığı:** Şu anda açıklanmaktadır.
- **Kesme testi:** Bazı mikro talimatlar kesmeler için bir test belirtir. Bir kesme olmuşsa, bu bir sonraki mikro komut adresini belirler.
- **Dallanma:** Koşullu ve koşulsuz dallanma mikro talimatları kullanılır.

Tek seviyeli bir alt rutin olağlığı sağlanmıştır. Her mikro komutta bir bit bu görevde ayrılmıştır. Bit ayarlandığında, 11 bitlik bir dönüş kontrol adresi kaydının güncellenmiş içeriğile yüklenir. Bir dönüşü belirten sonraki bir mikro komut, kontrol adres kaydının dönüş kaydından yüklenmesine neden olur.

Dönüş, koşulsuz dallanma komutunun bir şeklidir. Başka bir koşulsuz dallanma şekli, kontrol adres kaydının bitlerinin mikro talimatın 11 bitinden yüklenmesine neden olur. Koşullu dallanma talimiği mikro talimat içinde 4 bitlik bir test kodu kullanır. Bu kod, dallanma kararını belirlemek için çeşitli ALU koşul kodlarının test edilmesini belirtir. Eğer koşul doğru değilse, bir sonraki sıralı adres seçilir. Doğru ise, kontrol adres kaydının en düşük sıralı 8 biti mikro komutun 8 bitinden yüklenir. Bu, 256 kelimelek bir bellek sayfası içinde dallanmaya izin verir.

Göründüğü gibi LSI-11, kontrol ünitesi içinde güçlü bir adres sıralama olağanlığı içermektedir. Bu, mikro programcıya önemli ölçüde esneklik sağlar ve mikro programlama görevini kolaylaştırabilir. Öte yandan, bu yaklaşım daha basit yeteneklere göre daha fazla kontrol ünitesi mantığı gerektirir.

## 21.3 MIKRO KOMUT YÜRÜTME

Mikro komut döngüsü, mikro programlanmış bir işlemcideki temel olaydır. Her döngü iki kısımdan oluşur: getirme ve yürütme. Getirme kısmı bir mikro komut adresinin oluşturulmasıyla belirlenir ve bu önceki bölümde ele alınmıştır. Bu bölüm bir mikro komutun yürütülmesi ile ilgilidir.

Bir mikro komutun yürütülmesinin etkisinin kontrol sinyalleri üretmek olduğunu hatırlayın. Bu sinyallerin bazıları işlemcinin içindeki noktaları kontrol eder. Geri kalan sinyaller harici kontrol veriyoluna ya da diğer harici arayzlere gider. Tesadüfi bir işlev olarak, bir sonraki mikro komutun adresi belirlenir.

Önceki açıklama Şekil 21.10'da gösterilen bir kontrol ünitesinin organizasyonunu önermektedir. Şekil 21.4'ün biraz revize edilmiş bu versiyonu bu bölümün odak noktasını vurgulamaktadır. Bu diyagramdaki ana modüller artık açık olmalıdır. Sıralama mantık modülü, önceki bölümde tartışılan işlevleri yerine getirecek mantığı içerir. Girdi olarak komut, ALU bayraklarını, kontrol adres yazmacını (artırma için) ve kontrol tampon yazmacını kullanarak bir sonraki mikro komutun adresini üretir. Sonucusu gerçek bir adres, kontrol bitleri veya her ikisini de sağlayabilir. Modül, mikroinşa döngüsünün zamanlamasını belirleyen bir saat tarafından çalıştırılır.

Kontrol mantık modülü, mikro komuttaki bazı bitlerin bir fonksiyonu olarak kontrol sinyalleri üretir. Mikro talimatın biçim ve kontrol mantık modülünün karmaşaklığını belirlediği açık olmalıdır.

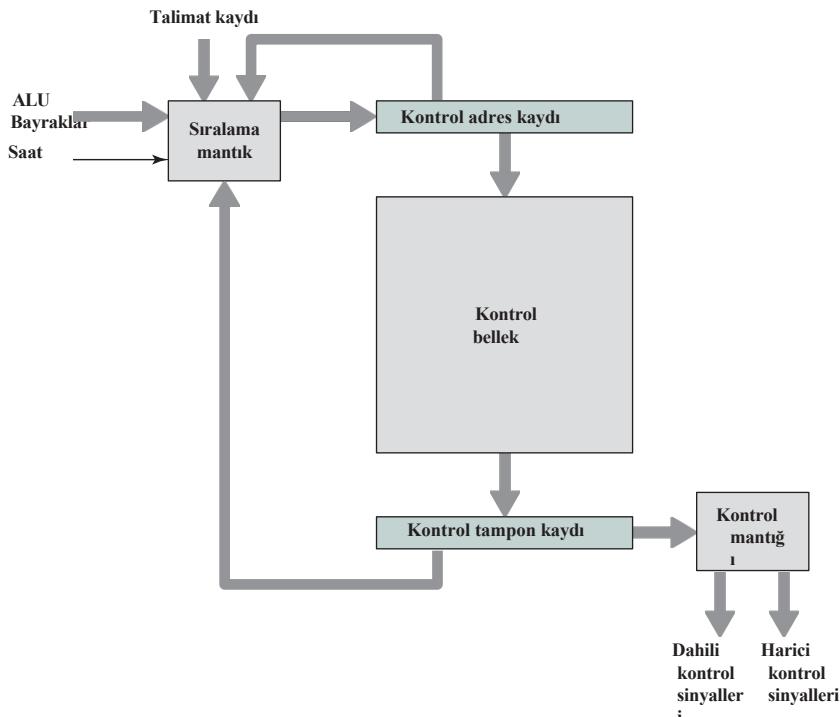
### Mikro İformasyonların Taksonomisi

Mikro talimatlar çeşitli şekillerde sınıflandırılabilir. Literatürde yaygın olarak yapılan ayırmalar aşağıdakileri içerir:

- Dikey/yatay
- Paketlenmiş/paketlenmemiş
- Sert/yumuşak mikro programlama
- Doğrudan/dolaylı kodlama

Bunların hepsi mikro öğretimin formatına bağlıdır. Bu terimlerin hiçbir literatürde tutarlı ve kesin bir şekilde kullanılmamıştır. Ancak, bu nitelik çiftlerinin incelenmesi, mikro öğretim tasarım alternatiflerini aydınlatmaya hizmet eder. Aşağıdaki paragraflarda, ilk olarak tüm bu özellik çiftlerinin altında yatan temel tasarım konusuna bakacağız ve ardından her bir çift tarafından önerilen kavramları inceleyeceğiz.

Wilkes'in [WILK51] orijinal önerisinde, bir mikro yapının her biti ya doğrudan bir kontrol sinyali üretiyor ya da doğrudan bir sonraki adresin bir bitini üretiyordu. Bir önceki bölümde, daha karmaşık adreslerin



Şekil 21.10 Kontrol Ünitesi Organizasyonu

Daha az mikro komut biti kullanan sıralama şemaları mümkündür. Bu şemalar daha karmaşık bir sıralama mantık modülü gerektirir. Mikro talimatın kontrol sinyalleriyle ilgili kısmı için de benzer bir değişim tokuş söz konusudur. Kontrol bilgilerinin kodlanması ve daha sonra kontrol sinyalleri üretmek için kodlarının çözülmesiyle, kontrol kelimesi bitlerinden tasarruf edilebilir.

Bu kodlama nasıl yapılabilir? Buna cevap vermek için, kontrol birimi tarafından sürülmeli gerekken toplam  $K$  farklı dahili ve harici kontrol sinyali olduğunu düşünün. Wilkes'in şemasında, mikro komutun  $K$  biti bu amaca . Bu, herhangi bir komut döngüsünde 2 $K$  olası kontrol sinyali kombinasyonunun tamamının üretilmesine olanak tanır. Ancak olası kombinasyonların hepsinin kullanılmayacağını gözlemlersek bundan daha iyisini yapabiliyoruz. Örnekler aşağıdakileri içerir:

- İki kaynak aynı hedefe yönlendirilemez (örneğin, Şekil 21.5'teki  $C_{(2)}$  ve  $C_{(8)}$ ).
- Bir kayıt hem kaynak hem de hedef olamaz (örneğin, Şekil 21.5'teki  $C_5$  ve  $C_{12}$ ).
- Bir seferde ALU'ya yalnızca bir kontrol sinyali modeli sunulabilir.
- Harici kontrol veriyoluna bir seferde yalnızca bir kontrol sinyali modeli sunulabilir.

Dolayısıyla, belirli bir işlemci için, izin verilen tüm olası kontrol sinyali kombinasyonları listelenebilir ve bu da belirli sayıda  $Q < 2^K$  olasılığı verir. Bunlar en az  $\log_2 Q$  bit ile kodlanabilir,  $(\log_2 Q) < K$ . Bu, tüm izin verilebilir kontrol sinyalleri kombinasyonlarını koruyan mümkün olan en sıkı kodlama şekli olacaktır. Pratikte, bu kodlama biçimini iki nedenden dolayı kullanılmamaktadır:

- Saf kod çözme (Wilkes) şeması kadar programlanması zordur. Bu nokta ilerde daha ayrıntılı olarak ele alınacaktır.
- Karmaşık ve dolayısıyla yavaş bir kontrol mantık modülü gerektirir.

Bunun yerine bazı ödünlər verilir. Bunlar iki çeşittir:

- Olası kombinasyonları kodlamak için kesinlikle gerekli olanın daha fazla bit kullanılır.
- Fiziksel olarak izin verilen bazı kombinasyonların kodlanması mümkün değildir.

İkinci tür uzlaşmanın bit sayısını azaltma etkisi vardır. Ancak net sonuç  $\log_2 Q$  bitten daha fazla bit kullanmaktadır.

Bir sonraki alt bölümde, belirli kodlama tekniklerini tartışacağız. Bu alt bölümün geri kalanında kodlamanın etkileri ve bunu tamamlamak için kullanılan çeşitli terimler ele alınmaktadır.

Yukarıdakilere dayanarak, mikro talimat formatının kontrol sinyali kısmının bir spektruma düşüğünü görebiliriz. Bir uça, her kontrol sinyali için bir bit vardır; diğer uça ise yüksek oranda kodlanmış bir format kullanılır. Tablo 21.4, mikro programlı bir kontrol ünitesinin diğer özelliklerinin de bir spektrum üzerinde yer aldığı ve bu spektrumların büyük ölçüde kodlama derecesi spektrumu tarafından belirlendiğini göstermektedir.

Tablodaki ikinci öğe çifti oldukça açıkta. Saf Wilkes şeması en fazla biti gerektirecektir. Bu üç noktanın donanımın en ayrıntılı görünümünü sunduğu da açıkça görülmeliidir. Her kontrol sinyali ayrı ayrı kontrol edilebilir.

**Tablo 21.4** Mikro Talimat Spektrumu

Özellikler	
Kodlanmamış	Yüksek oranda kodlanmış
Birçok bit	Birkaç parça
Donanımın ayrıntılı görünümü	Donanımın toplu görünümü
Programlaması zor	Programlaması kolay
Eşzamanlılıktan tamamen yararlanıldı	Eşzamanlılıktan tam olarak yararlanılamıyor
Kontrol mantığı çok az veya hiç yok	Karmaşık kontrol mantığı
Hızlı uygulama	Yavaş yürütme
Performansı optimize edin	Programlamayı optimize edin
Terminoloji	
Ambalajsız	Paketlenmiş
Yatay	Dikey
Sert	Yumuşak

mikro programcı tarafından. Kodlama, işlevleri veya kaynakları bir araya getirecek şekilde yapılır, böylece mikro programcı işlemciyi daha yüksek, daha az ayrıntılı bir seviyede görür. Ayrıca, kodlama mikro programlama yükünü hafifletmek için tasarlanmıştır. Yine, tüm kontrol sinyallerinin kullanımını anlama ve düzenlemeye görevinin zor bir görev olduğu açık olmalıdır. Daha önce de belirtildiği gibi, kodlamadan sonuçlarından biri, tipik olarak, aksi takdirde izin verilebilecek bazı kombinasyonların kullanımını engellemektir.

Bir önceki paragraf mikro komut tasarımlını mikro programcinin bakış açısından tartısmaktadır. Ancak kodlama derecesi donanım etkileri açısından da görülebilir. Saf kodlanmamış bir formatta, kod çözme mantığına çok az ihtiyaç duyulur ya da hiç ihtiyaç duyulmaz; her bit belirli bir kontrol sinyali üretir. Daha kompakt ve daha yoğun kodlama şemaları kullanıldıkça, daha karmaşık kod çözme mantığına ihtiyaç duyulur. Bu da performansı etkileyebilir. Sinyalleri daha karmaşık kontrol mantığı modülünün kapılardan geçirmek için daha fazla zamana ihtiyaç duyulur. Bu nedenle, kodlanmış mikro talimatların yürütülmESİ, kodlanmamış olanların yürütülmESİnden daha uzun sürer.

Bu nedenle, Tablo 21.4'te listelenen tüm özellikler bir tasarım stratejileri spektrumu boyunca yer almaktadır. Genel olarak, spektrumun sol ucuna düşen bir tasarım kontrol ünitesinin performansını optimize etmeyi amaçlar. Sağ uca doğru olan tasarımlar daha çok mikro programlama sürecini optimize etmekle ilgiliidir. Gerçekten de, spektrumun sağ ucuna yakın mikro komut setleri makine komut setlerine çok benzer. Bunun iyi bir örneği, bu bölümün ilerleyen kısımlarında açıklanan LSI-11 tasarımidır. Tipik olarak, amaç sadece bir kontrol birimi uygulamak olduğunda, tasarım spektrumun sol ucuna yakın olacaktır. Şu anda tartışılan IBM 3033 tasarımları bu kategoridedir. Daha sonra tartışacağımız gibi, bazı sistemler çeşitli kullanıcıların aynı mikro komut kolaylığını kullanarak farklı mikro programlar oluşturmasına izin verir. Bu ikinci durumda, tasarımın spektrumun sağ ucuna yakın olması muhtemeldir.

Şimdi daha önce tanıtılan bazı terminolojiyi ele alabiliriz. Tablo 21.4, bu terim çiftlerinden üçünün mikro talimat spektrumıyla nasıl ilişkili olduğunu göstermektedir. Özünde, bu çiftlerin hepsi aynı şeyi tanımlamakta ancak farklı tasarım özelliklerini vurgulamaktadır.

Paketleme derecesi, belirli bir kontrol görevi ile belirli mikro talimat arasındaki tanımlama derecesiyle ilgiliidir. Bitler daha fazla paketlendikçe, belirli sayıda bit daha fazla bilgi içerir. Dolayısıyla, paketleme kodlama anlamına gelir. *Yatay* ve *dikey* terimleri mikro talimatların göreceli genişliği ile ilgiliidir. [SIEW82], genel bir kural olarak dikey mikro yapıların 16 ila 40 bit aralığında, yatay mikro yapıların ise 40 ila 100 aralığında uzunluğa sahip olmasını önermektedir. *Sert* ve *yumuşak* mikro programlama terimleri, alta yatan kontrol sinyallerine ve donanım düzenebine yakınlık derecesini belirtmek için kullanılır. Sert mikro programlar genellikle sabittir ve salt okunur belleğe işlenir. Yumuşak mikro programlar daha değişkendir ve kullanıcı mikro programlamasını düşündür.

Bu alt bölümün başında bahsedilen diğer terim çifti, şimdi döneceğimiz bir konu olan doğrudan ve dolaylı kodlamaya atıfta bulunmaktadır.

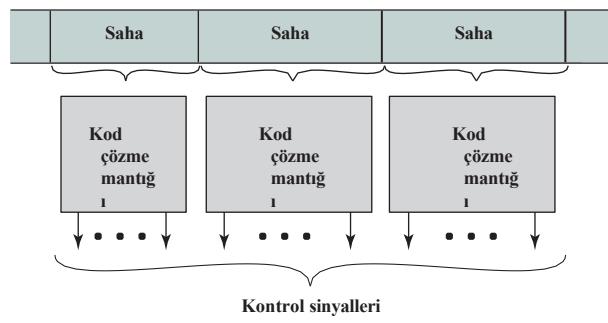
### Mikro Talimat Kodlaması

Uygulamada, mikro programlı kontrol üniteleri saf kodlanmamış veya yatay mikro komut formatı kullanılarak tasarlanmaktadır. Kontrol belleği genişliğini azaltmak ve mikro programlama görevini basitleştirmek için en azından bir dereceye kadar kodlama kullanılır.

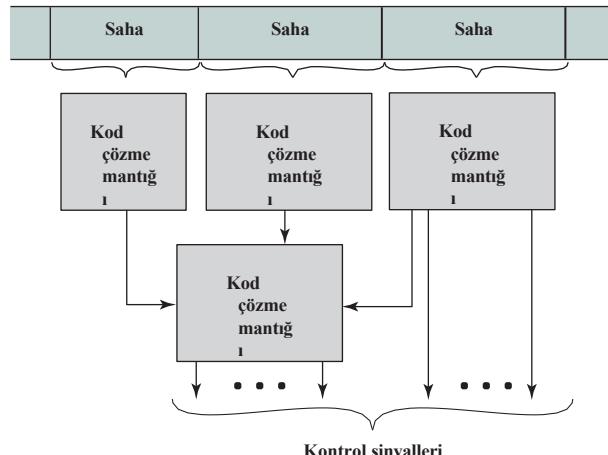
Kodlama için temel teknik Şekil 21.11a'da gösterilmiştir. Mikro yapı bir dizi alan olarak düzenlenmiştir. Her alan, kodu çözüldüğünde bir veya daha fazla kontrol sinyalini etkinleştirten bir kod içerir.

Bu düzenin sonuçlarını ele alalım. Mikro komut yürütüldüğünde, her alanın kodu çözülür ve kontrol sinyalleri üretir. Böylece,  $N$  alan ile  $N$  eşzamanlı eylem belirtilir. Her eylem bir veya daha fazla kontrol sinyalinin etkinleştirilmesiyle sonuçlanır. Genel olarak, ancak her zaman değil, formatı her kontrol sinyali birden fazla alan tarafından etkinleştirilmeyecek şekilde tasarlama isteyeceğiz. Bununla birlikte, her kontrol sinyalinin en az bir alan tarafından etkinleştirilmesinin mümkün olması gerektiği açıklar.

Şimdi bireysel alanı ele alalım.  $L$  bitten oluşan bir alan, her biri farklı bir kontrol sinyali modeline kodlanabilen  $2^L$  koddan birini içerebilir. Bir alanda aynı anda yalnızca bir kod bulunabileceğinden, kodlar birbirini dışlar ve dolayısıyla neden oldukları eylemler de birbirini dışlar.



(a) Doğrudan kodlama



(b) Dolaylı kodlama

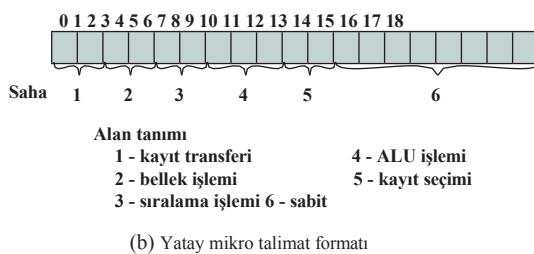
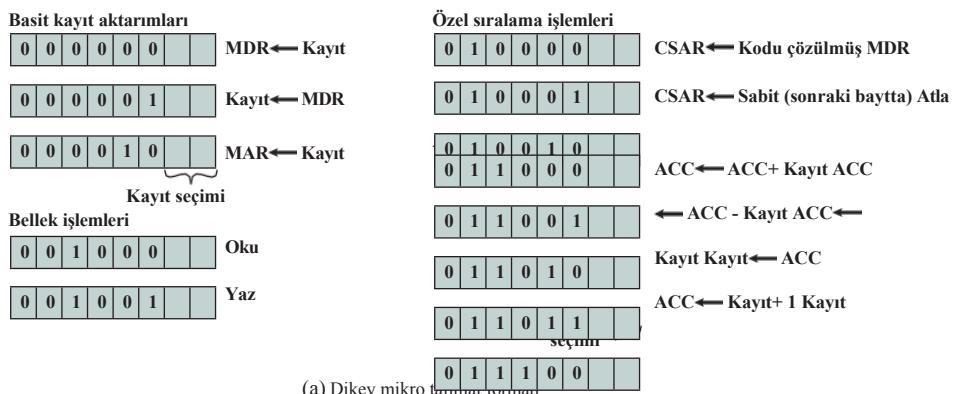
**Şekil 21.11** Mikro Komut Kodlaması

Kodlanmış bir mikro talimat formatının tasarımları artık basit terimlerle ifade edilebilir:

- Formatı bağımsız alanlar halinde düzenleyin. Yani, her alan, farklı alanlardaki eylemlerin aynı anda gerçekleşebileceğinin şekilde bir dizi eylemi (kontrol sinyalleri modeli) tasvir eder.
- Her alanı, alan tarafından belirtilebilecek alternatif eylemler birbirini dışlayacak şekilde tanımlayın. Yani, belirli bir alan için belirtilen eylemlerden yalnızca biri aynı anda gerçekleşebilir.

Kodlanmış mikro talimatları alanlar halinde düzenlemek için iki yaklaşım benimsenebilir: işlevsel ve kaynak. *İşlevsel kodlama* yöntemi makine içindeki işlevleri tanımlar ve alanları işlev türüne göre. Örneğin, akümülatöre veri aktarmak için çeşitli kaynaklar kullanılabilir, bu amaç için bir alan belirlenebilir ve her kod farklı bir kaynağı belirtir. *Kaynak kodlaması*, makineyi bir dizi bağımsız kaynaktan oluşuyormuş gibi görür ve her birine bir alan ayırrır (örneğin, G/C, bellek, ALU).

Kodlamamanın bir başka yönü de doğrudan mı yoksa dolaylı mı olduğunu (Şekil 21.11b). Dolaylı kodlamada, bir alan başka bir alanın yorumunu belirlemek için kullanılır.



**Şekil 21.12** Basit Bir Makine için Alternatif Mikro Talimat Formatları

alan. Örneğin, sekiz farklı aritmetik işlem ve sekiz farklı kaydırma işlemi gerçekleştirebilen bir ALU düşünün. Bir kaydırma veya aritmetik işlemin kullanılacağını belirtmek için 1 bitlik bir alan kullanılabılır; 3 bitlik bir alan işlemi belirtecektir. Bu teknik genellikle iki seviyeli kod çözme anlamına gelir ve yayılma gecikmelerini artırır.

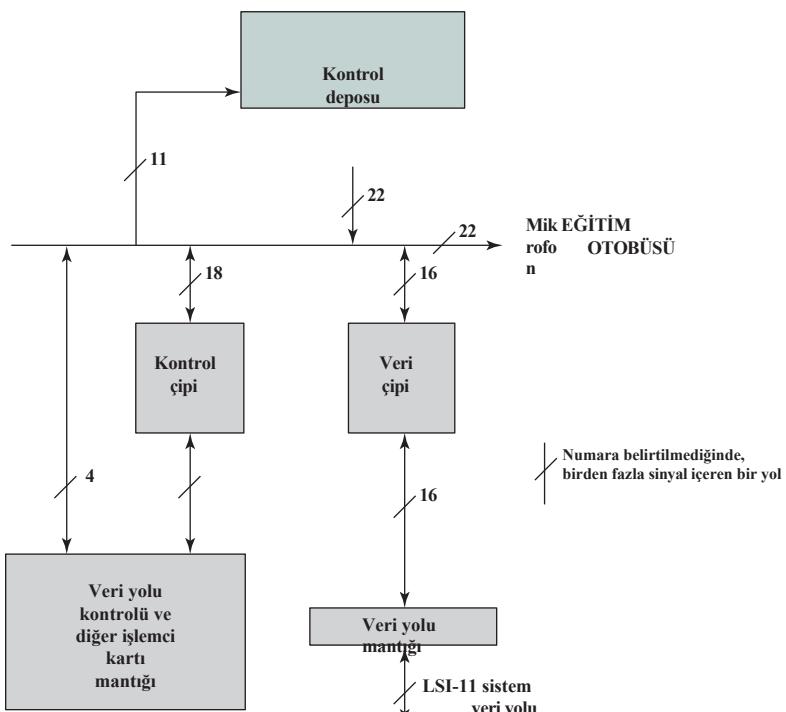
Şekil 21.12 bu kavramların basit bir örneğidir. Tek bir akümülatöre ve program sayacı ve ALU girişi için geçici bir yazmaç gibi birkaç dahili yazmaca sahip bir işlemci varsayılmış. Şekil 21.12a oldukça dikey bir format göstermektedir. İlk 3 bit işlem türünü gösterir, sonraki 3 bit işlemi kodlar ve son 2 bit dahili bir kaydediciyi seçer. Şekil 21.12b, kodlama hala kullanılmasına rağmen daha yataş bir yaklaşımındır. Bu durumda, farklı işlevler farklı alanlarda görünürlü.

### LSI-11 Mikro Komut Yürütme

LSI-11 [SEBE76] dikey mikro komut yaklaşımına iyi bir örnektir. Önce kontrol biriminin organizasyonuna, sonra da mikro komut formatına bakacağımız.

**LSI-11 KONTROL BİRİMİ ORGANİZASYONU** LSI-11, PDP-11 ailesinin tek kartlı işlemci olarak sunulan ilk üyesidir. Kart üç LSI yongası, *mikro komut veriyolu* (MIB) olarak bilinen bir dahili *veriyolu* ve bazı ek arayüz mantığı içerir.

Şekil 21.13 LSI-11 prosesörünün organizasyonunu basitleştirilmiş bir şekilde göstermektedir. Üç yonga veri, kontrol ve kontrol depolama yongalarıdır. Veri yongası 8 bitlik bir ALU, yirmi altı adet 8 bitlik yazmaç ve birkaç durum için depolama alanı içerir.



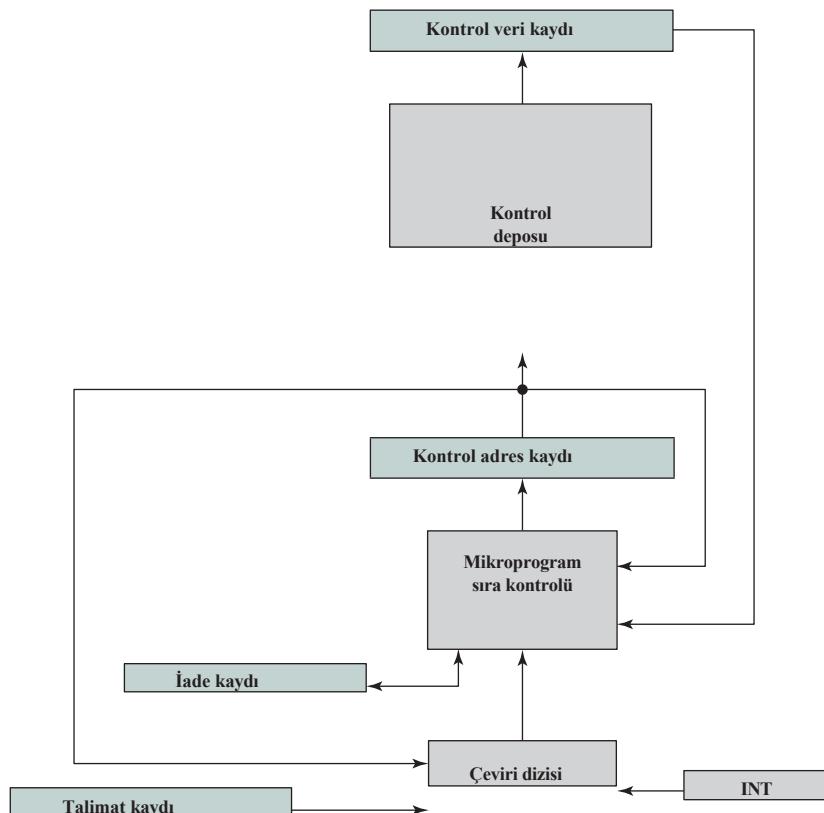
**Şekil 21.13** LSI-11 İşlemcinin Basitleştirilmiş Blok Diyagramı

kodlar. Kayıtların on altısı PDP-11'in sekiz adet 16 bitlik genel amaçlı kaydını uygulamak için kullanılır. Diğerleri bir program durum sözcüğü, bellek adres kaydı (MAR) ve bellek tampon kaydır içerir. ALU bir seferde sadece 8 bit ile ilgilendiğinden, 16 bitlik bir PDP-11 aritmetik işlemini uygulamak için ALU'dan iki geçiş gereklidir. Bu mikro program tarafından kontrol edilir.

Kontrol deposu yongası veya yongaları 22 bit genişliğinde kontrol belleğini içerir. Kontrol çipi mikro talimatların sıralanması ve yürütülmesi için mantık içerir. Kontrol adres kaydını, kontrol veri kaydını ve makine komutu kaydının bir kopyasını içerir.

MIB tüm bileşenleri birbirine bağlar. Mikro komut getirme sırasında kontrol çipi MIB üzerinde 11 bitlik bir adres üretir. Kontrol deposuna erişilir ve MIB üzerine yerleştirilen 22 bitlik bir mikro talimat üretir. Düşük sıralı 16 bit veri çipine giderken, düşük sıralı 18 bit kontrol çipine gider. Yüksek sıralı 4 bit özel işlemci kartı işlevlerini kontrol eder.

Şekil 21.14 LSI-11 kontrol unitesine hala basitleştirilmiş ancak daha ayrıntılı bir bakış sunmaktadır: şekil bireysel çip sınırlarını göz ardi etmektedir. Bölüm 21.2'de açıklanan adres sıralama şeması iki modülde uygulanmaktadır. Genel sıra kontrolü mikro program sıra kontrol modülü tarafından sağlanır ve bu modül



**Şekil 21.14** LSI-11 Kontrol Biriminin Organizasyonu

mikro komut adres kaydını artırma ve koşulsuz dallanma gerçekleştirmeye. Diğer adres hesaplama biçimleri ayrı bir trans-lasyon dizisi tarafından gerçekleştirilir. Bu, mikro komuta, makine komutuna, mikro komut program sayacına ve bir kesme kaydına dayalı olarak bir adres üreten kombinatoryal bir devredir.

Çeviri dizisi aşağıdaki durumlarda devreye girer:

- İşlem kodu, bir mikro rutinin başlangıcını belirlemek için kullanılır.
- Uygun zamanlarda, mikro komutun adres modu bitleri uygun adreslemeyi gerçekleştirmek için test edilir.
- Kesme koşulları periyodik olarak test edilir.
- Koşullu dal mikro talimatları değerlendirilir.

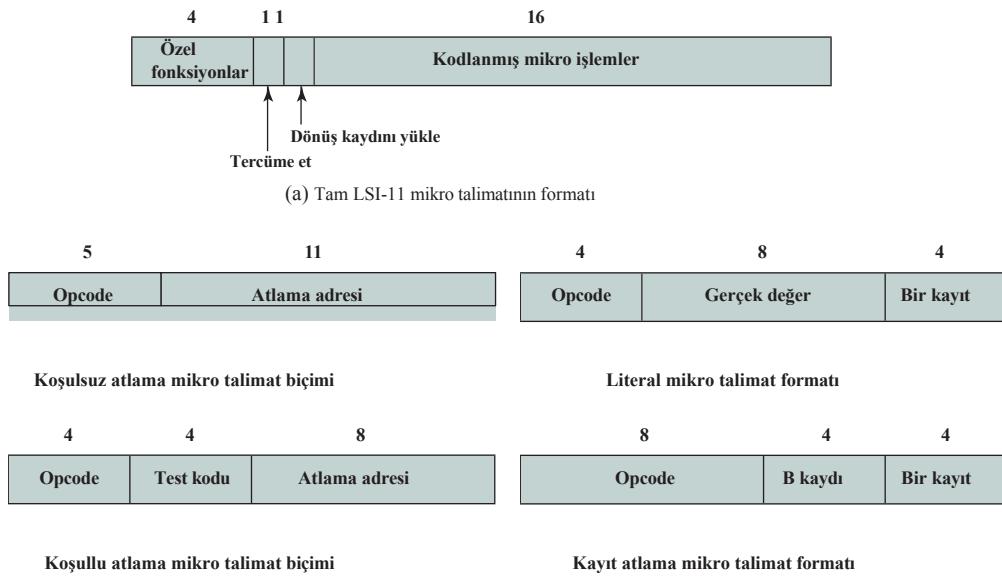
**LSI-11 MİKROKOMUT FORMATI** LSI-11 sadece 22 bit genişliğinde olan son derece dikey bir mikro komut formatı kullanır. Mikro komut seti, uyguladığı PDP-11 makine komut setine büyük ölçüde benzemektedir. Bu tasarım, dikey, kolay programlanabilir bir tasarım kısıtlaması içinde kontrol biriminin performansını optimize etmemi amaçlamıştır. Tablo 21.5 LSI-11 mikro komutlarından bazılarını listelemektedir. Şekil 21.15 22-bit LSI-11 mikro komut formatını göstermektedir. Yüksek mertebeli

4 bit işlemci kartındaki özel fonksiyonları kontrol eder. Translate çeviri dizisinin bekleyen kesmeleri kontrol etmesini sağlar. Load return register biti bir mikro rutinin sonunda bir sonraki mikro komut adresinin return register'dan yüklenmesini sağlamak için kullanılır.

Kalan 16 bit yüksek kodlanmış mikro-işlemler için kullanılır. Format, değişken uzunlukta bir işlem kodu ve bir veya daha fazla işleneni olan bir makine komutuna çok benzer.

**Tablo 21.5** Bazi LSI-11 Mikro Talimatları

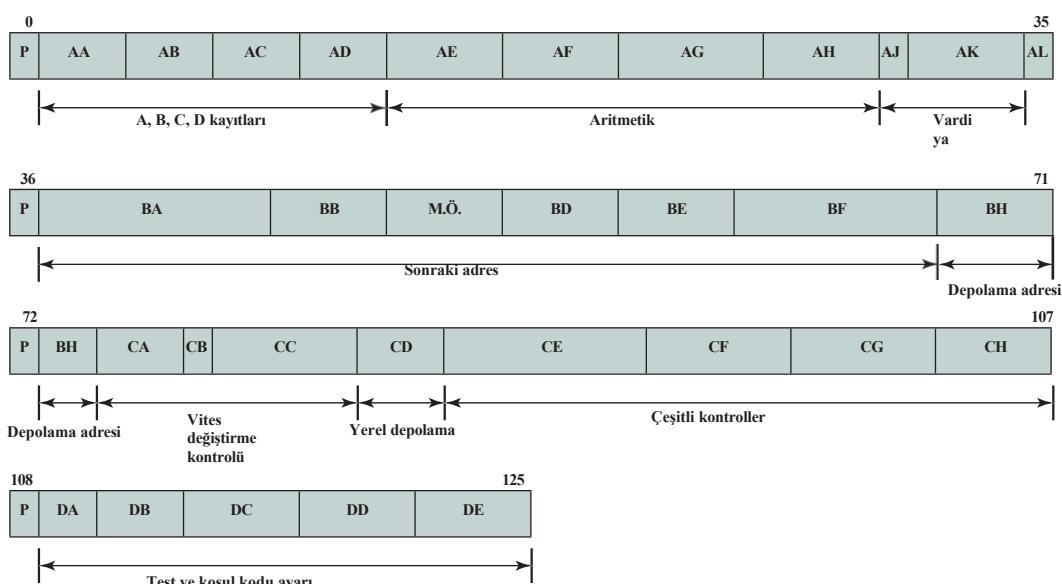
Aritmetik İşlemler	Genel Operasyonlar
Sözcük ekle (bayt, değişmez)	MOV word (bayt)
Test sözcüğü (bayt, değişmez)	Atlama
Sözcüğü (bayt) 1 artırma	Dönüş
Sözcüğü (bayt) 2 artırma	Koşullu atlama
Negate word (byte)	Bayrakları
ayarlama (sıfırlama) Bayt koşullu olarak artırma (azaltma)	G düşük
yükleyin	
Koşullu olarak sözcük (bayt) ekleme	Koşullu olarak MOV sözcüğü (bayt)
Taşımalı sözcük (bayt) ekleme	
Koşullu olarak (ekle) (rakamlar)	
Sözcük (bayt) çıkarma	Giriş/Çıkış İşlemleri
Sözcüğü karşılaştır (bayt, değişmez)	Giriş sözcüğü (bayt)
Kelimeyi (bayt) taşıma ile çıkarma	Giriş durumu kelimesi (bayt)
Sözcüğü (bayt) 1 azaltır	Okuyun
Mantıksal İşlemler	Yazmak
kelime (bayt, değişmez)	Okuma (yazma) ve kelimeyi (bayt) 1 artırma
Test sözcüğü (bayt)	Okuma (yazma) ve kelimeyi (bayt) 2 artırma VE
VEYA sözcüğü (bayt)	Okuma (yazma) onayı
Exclusive-OR word (bayt) Bit	Çıkış sözcüğü (bayt, durum)
clear word (bayt)	
Sözcüğü (bayt) sağa (sola) taşıma ile (taşımaz) kaydırma Sözcüğü (bayt) tamamlama	



Şekil 21.15 LSI-11 Mikro Komut Formatları

### IBM 3033 Mikro Komut Yürütme

Standart IBM 3033 kontrol belleği 4K sözcükten oluşur. Bunların ilk yarısı (0000-07FF) 108 bitlik mikro talimatlar içerirken, geri kalanı (0800-0FFF) 126 bitlik mikro talimatları saklamak için kullanılır. Format Şekil 21.16'da gösterilmiştir.



Şekil 21.16 IBM 3033 Mikro Komut Formatı

**Tablo 21.6** IBM 3033 Mikro Komut Kontrol Alanları

ALU Kontrol Alanları	
AA(3)	Veri kayıtlarından birinden A kaydını yükle
AB(3)	Veri kayıtlarından birinden B kaydını yükle
AC(3)	Veri kayıtlarından birinden C kaydını yükle
AD(3)	Veri kayıtlarından birinden D kaydını yükle
AE(4)	Belirtilen A bitlerini ALU'ya yönlendirin
AF(4)	Belirtilen B bitlerini ALU'ya yönlendirin
AG(5)	A girişi üzerinde ALU aritmetik işlemini belirtir
AH(4)	B girişi üzerinde ALU aritmetik işlemini belirtir
AJ(1)	B tarafındaki ALU'ya D veya B girişini belirtir
AK(4)	Aritmetik çıkışı kaydırıcıya yönlendirin
CA(3)	F kaydını yükle
CB(1)	Vites değiştiriciyi etkinleştir
CC(5)	Mantıksal ve taşıma fonksiyonlarını belirtir
CE(7)	Vardiya miktarını belirtir
Sıralama ve Dallanma Alanları	
AL(1)	İşlemi sonlandırın ve dallanma gerçekleştirein
LISANS(8)	Kontrol adresi kaydının yüksek sıralı bitlerini (00-07) ayarlayın
BB(4)	Kontrol adresi kaydının 8. bitini ayarlama koşulunu belirtir
BC(4)	Kontrol adresi kaydının 9. bitini ayarlama koşulunu belirtir
BD(4)	Kontrol adresi kaydının 10. bitini ayarlama koşulunu belirtir
BE(4)	Kontrol adresi kaydının 11. bitinin ayarlanması için koşulu belirtir
BF(7)	Kontrol adresi kaydının 12. bitini ayarlama koşulunu belirtir

Bu oldukça yatay bir format olmasına rağmen, kodlama hala yaygın olarak kullanılmaktadır. Bu formatın temel alanları Tablo 21.6'da özetlenmiştir.

ALU, A, B, C ve D olmak üzere dört özel, kullanıcı tarafından görülememeyen yazmacıtan gelen girişlerle çalışır. Mikro komut formatı, bu yazmacıları kullanıcı tarafından görülebilen yazmacılardan yüklemek, bir ALU işlevi gerçekleştirmek ve sonucu saklamak için kullanıcı tarafından görülebilen bir yazmaç belirtmek için alanlar içerir. Ayrıca yazmacılar ve bellek arasında veri yüklemek ve depolamak için alanlar da vardır.

IBM 3033 için sıralama mekanizması Bölüm 21.2'de tartışılmıştır.

## 21.4 TI 8800

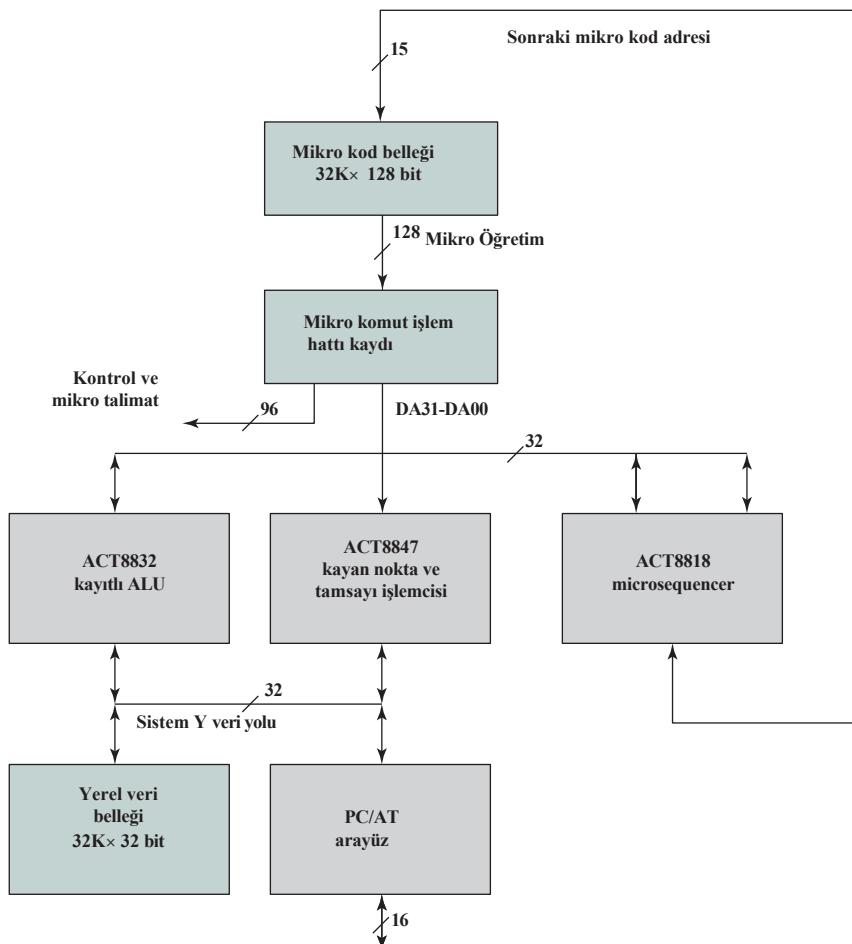
Texas Instruments 8800 Yazılım Geliştirme Kartı (SDB) mikroprogramlanabilir 32-bit bilgisayar kartıdır. Sistem, ROM yerine RAM'e yerleştirilmiş yazılabilir bir kontrol deposuna sahiptir. Böyle bir sistem hız ya da

ROM kontrol deposu ile mikro programlanmış bir sistemin yoğunluğu. Bununla birlikte, prototip geliştirmek ve eğitim amaçları için kullanılmıştır.

8800 SDB aşağıdaki bileşenlerden oluşur (Şekil 21.17):

- Mikro kod belleği
- Microsequencer
- 32 bit ALU
- Kayan nokta ve tamsayı işlemcisi
- Yerel veri belleği

İki veri yolu sistemin dahili bileşenlerini birbirine bağlar. DA veri yolu, mikro komut veri alanından ALU'ya, kayan nokta işlemcisine veya mikro sıralayıcıya veri sağlar. İkinci durumda, veri bir dallama komutu için kullanılacak bir adresten oluşur. Veri yolu ayrıca ALU veya mikro sıralayıcı için de kullanılabilir



Şekil 21.17 TI 8800 Blok Diyagramı

diğer bileşenlere veri sağlar. Sistem Y veriyolu, ALU ve kayan noktalı işlemciyi yerel belleğe ve PC arayüzü üzerinden harici modüllere bağlar.

Kart, IBM PC uyumlu bir ana bilgisayara takılır. Ana bilgisayar, mikro kod montajı ve hata ayıklama için uygun bir platform sağlar.

## Mikro Talimat Formатı

8800 için mikro komut formatı, Tablo 21.7'de gösterildiği gibi 30 işlevsel alana bölünmüş 128 bitten oluşur. Her alan bir veya daha fazla bitten oluşur ve alanlar beş ana kategoride gruplandırılmıştır:

- Yönetim kurulu kontrolü
- 8847 kayan nokta ve tamsayı işlemci çipi
- 8832 kayıtlı ALU
- 8818 mikrosequencer
- WCS veri alanı

Şekil 21.17'de gösterildiği gibi, WCS veri alanının 32 biti ALU, kayan noktalı işlemci veya mikro sıralayıcıya veri olarak verilmek üzere DA veri yoluna beslenir. Mikro komutun diğer 96 biti (1-27 alanları) doğrudan uygun module beslenen kontrol sinyalleridir. Basitlik açısından, bu diğer bağlantılar Şekil 21.17'de gösterilmemiştir.

İlk altı alan, tek bir bileşenin kontrolünden ziyade panonun kontrolüne ilişkin işlemlerle ilgilidir. Kontrol işlemleri aşağıdakileri içerir:

- Sıralayıcı kontrolü için koşul kodlarının seçilmesi. Alan 1'in ilk biti koşul bayrağının 1'e mi yoksa 0'a mı ayarlanacağını belirtir ve kalan 4 bit hangi bayrağın ayarlanacağını gösterir.
- PC/AT'ye bir I/O isteği gönderme.
- Yerel veri belleği okuma/yazma işlemlerinin etkinleştirilmesi.
- Sistem Y veriyolunu süren birimi belirleme. Veriyoluna bağlı dört cihazdan biri seçilir (Şekil 21.17).

Son 32 bit, belirli bir mikro komuta özgü bilgileri içeren veri alanıdır.

Mikro komutun geri kalan alanları en iyi şekilde kontrol ettikleri cihazın metninde tartışırlır. Bu bölümün geri kalanında, mikro sıralayıcı ve kayıtlı ALU'yu tartışacağız. Kayan nokta birimi yeni kavramlar getirmez ve atlanır.

## Microsequencer

8818 mikro sıralayıcının temel işlevi mikro program için bir sonraki mikro yapı adresini üretmektir. Bu 15 bitlik adres mikro kod belleğine verilir (Şekil 21.17).

Bir sonraki adres beş kaynaktan birinden seçilebilir:

1. Mikro program sayacı (MPC) kaydı, tekrarlama (aynı adresi yeniden kullanma) ve devam etme (adresi 1 artırma) talimatları için kullanılır.

**Tablo 21.7** TI 8800 Mikro Komut Formatı

Alan Numarası	Bit Sayısı	Açıklama
<b>Yönetim Kurulu Kontrolü</b>		
1	5	Durum kodu girişini seçin
2	1	Harici I/O istek sinyalini etkinleştirme/devre dışı bırakma
3	2	Yerel veri belleği okuma/yazma işlemlerini etkinleştirme/devre dışı bırakma
4	1	Yük durumu/yük yok durumu
5	2	Y otobüsünü süren birimi belirleyin
6	2	DA veri yolunu süren birimi belirleyin
<b>8847 Kayan Nokta ve Tamsayı İşleme Çipi</b>		
7	1	C kayıt kontrolü: saat, saat yapmayın
8	1	Y veri yolu için en anlamlı veya en az anlamlı bitleri seçin
9	1	C kayıt veri kaynağı: ALU, çoklayıcı
10	4	ALU ve MUL için IEEE veya FAST modunu seçin
11	8	Veri operandları için kaynakları seçin: RA kayıtları, RB kayıtları ters, P kaydı, 5 kaydı, C kaydı
12	1	RB kayıt kontrolü: saat, saat yapmayın
13	1	RA kayıt kontrolü: saat, saat yapmayın
14	2	Veri kaynağı onayı
15	2	Boru hattı kayıtlarını etkinleştirme/devre dışı bırakma
16	11	8847 ALU işlevi
<b>8832 Kayıtlı ALU</b>		
17	2	Seçilen veri çıkışını yazmayı etkinleştirir/devre dışı bırakır: en anlamlı yarı, en az anlamlı yarı
18	2	Kayıt dosyası veri kaynağını seçin: DA veri yolu, DB veri yolu, ALU Y MUX çıkışı, sistem Y veriyolu
19	3	Kaydırma talimatı değiştiricisi
20	1	İçeri taşıyın: zorlayın, zorlamayın
21	2	ALU yapılandırma modunu ayarlayın: 32, 16 veya 8 bit
22	2	5 çoklayıcıya giriş seçin: kayıt dosyası, DB veri yolu, MQ KAYIT
23	1	R çoklayıcıya giriş seçin: kayıt dosyası, DA veri yolu
24	6	Yazmak için C dosyasında kayıt seçin
25	6	Okuma için B dosyasındaki kaydı seçin
26	6	Yazmak için A dosyasındaki kaydı seçin
27	8	ALU işlevi
<b>8818 Mikrosequencer</b>		
28	12	8818'e giden kontrol giriş sinyalleri
<b>WCS Veri Alanı</b>		
29	16	Yazılabilir kontrol deposu veri alanının en anlamlı bitleri
30	16	Yazılabilir kontrol deposu veri alanının en küçük anlamlı bitleri

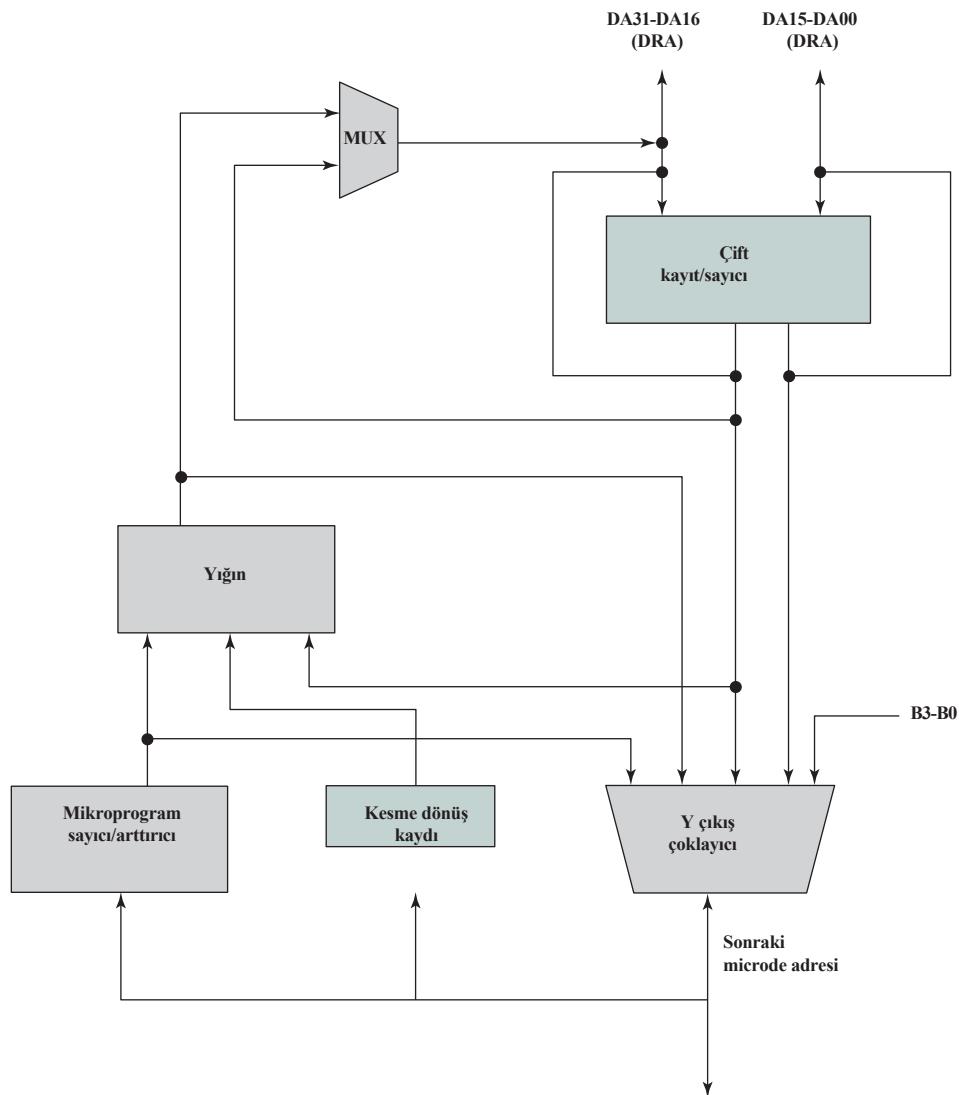
2. Mikro program alt program çağrılarının yanı sıra yinelemeli döngüler ve kesmelerden dönüşleri destekleyen yiğin.
  3. DRA ve DRB portları, harici donanımdan mikro program adreslerinin üretilebileceği iki ek yol sağlar. Bu iki port DA veriyolunun sırasıyla en anlamlı ve en az anlamlı 16 bağlanır. Bu, mikro sıralayıcının bir sonraki komut adresini mevcut mikro komutun WCS veri alanından veya ALU tarafından hesaplanan bir sonuçtan elde etmesini sağlar.
  4. Ek adres depolama için kullanılabilen RCA ve RCB kayıt sayaçları.
  5. Harici kesintileri desteklemek için çift yönlü Y portuna harici bir giriş.
- Şekil 21.18 8818'in mantıksal blok diyagramıdır. Cihaz aşağıdaki ana işlevsel gruptardan oluşur:
- Bir yazmaç ve bir artırıcıdan oluşan 16 bitlik bir mikro program sayacı (MPC).
  - Döngüler ve yinelemeleri saymak, dallanma adreslerini saklamak veya harici aygıtları sürmek için iki kayıt sayaç, RCA ve RCB.
  - Mikro program alt program çağrılarına ve kesmelerle izin veren 65 kelimelik 16 bitlik bir yiğin.
  - Mikro komut seviyesinde kesme işlemi için bir kesme dönüş kaydı ve Y çıkışını etkinleştirme.
  - Bir sonraki adresin MPC, RCA, RCB, harici veri yolları DRA ve DRB veya yiğinden seçilebildiği bir Y çıkış çöklayıcısı.

**KAYITLAR/SAYICILAR** RCA ve RCB kayıtları, mevcut mikro komuttan veya ALU çıkışından DA veriyolundan yüklenebilir. Değerler yürütme akışını kontrol etmek için sayaç olarak kullanılabilir ve erişildiğinde otomatik olarak azaltılabilir. Değerler ayrıca Y çıkış çöklayıcısına sağlanacak mikro komut adresleri olarak da kullanılabilir. Tek bir mikro komut döngüsü sırasında her iki kaydedicinin bağımsız kontrolü, her iki kaydedicinin aynı anda azaltılması haricinde desteklenir.

**YIĞIN** Yiğin, iç içe geçmiş çağrıların veya kesmelerin birden fazla seviyesine izin verir ve dallanma ve döngüyü desteklemek için kullanılabilir. Bu işlemlerin genel işlemciye değil kontrol birimine atıfta bulunduğu ve ilgili adreslerin kontrol belleğindeki mikro yapılarla ait olduğunu unutmayın.

Altı yiğin işlemi mümkündür:

1. Clear, yiğin işaretçisini sıfır ayarlayarak yiğini boşaltır;
2. Pop, yiğin işaretçisini azaltır;
3. MPC, kesme dönüş kaydı veya DRA veriyolunun içeriğini yiğine koyan ve yiğin işaretçisini artıran Push;
4. Okuma, okuma işaretçisi tarafından belirtilen adresi Y çıkış çöklayıcısında kullanılabilir hale getirir;



Şekil 21.18 TI 8818 Microsequencer

5. Hold, yığın işaretçisinin adresinin değişmeden kalmasına neden olur;
6. Yığın işaretçisini yükle, DRA'nın en az anlamlı yedi bitini yığın işaretçisine girer.

MİKROSEKANSERİN **KONTROLÜ** Mikrosekanser öncelikle mevcut mikro komutun 12 bitlik alanı olan alan 28 tarafından kontrol edilir (Tablo 21.7). Bu alan aşağıdaki alt alanlardan oluşur:

- **OSEL (1 bit):** Çıkış seçimi. DRA veriyolunu besleyen çöklayıcının çıkışına hangi değerin yerleştirileceğini belirler (DRA veriyolunun sol üst köşesi)

Şekil 21.18). Çıkış ya yiğinden ya da ister RCA'dan gelecek şekilde seçilir. DRA daha sonra Y çıkış çoklayıcısına ya da RCA kaydına giriş olarak hizmet .

- **SELDRL (1 bit):** DR veriyolunu seçin. 1'e ayarlanırsa, bu bit DRA/DRB veri yollarına giriş olarak harici DA veri yolunu seçer. 0'a ayarlanırsa, DRA çoklayıcısının çıkışını DRA veriyoluna (OSEL tarafından kontrol edilir) ve RCB içeriğini DRB veriyoluna seçer.
- **ZEROIN (1 bit):** Koşullu bir dallanmayı belirtmek için kullanılır. Mikro sıralayıcının davranışları daha sonra alan 1'de seçilen koşul koduna bağlı olacaktır (Tablo 21.7).
- **RC2-RC0 (3 bit):** Kayıt kontrolleri. Bu bitler RCA ve RCB yazmaçlarının içeriğindeki değişikliği belirler. Her yazmaç aynı kalabilir, azaltılabilir veya DRA/DRB veri yollarından yüklenebilir.
- **S2-S0 (3 bit):** Yiğin kontrolleri. Bu bitler hangi yiğin işleminin gerçekleştirileceğini belirler.
- **MUX2-MUX0:** Çıkış kontrolleri. Bu bitler, eğer kullanılıyorsa durum koduyla birlikte Y çıkış çoklayıcısını ve dolayısıyla bir sonraki mikroyapı adresini kontrol eder. Çoklayıcı çıkışını yiğinden, DRA'dan, DRB'den veya MPC'den seçebilir.

Bu bitler programcı tarafından ayrı ayrı ayarlanabilir. Ancak bu genellikle yapılmaz. Bunun yerine, programcı normalde gerekli olan bit modellerine eşit olan animsaticalar kullanır. Tablo 21.8, 28. alan için 15 animsaticayı listeler. Bir mikrokod birleştirici bunları uygun bit modellerine dönüştürür.

**Tablo 21.8** TI 8818 Microsequencer Mikro Komut Bitleri (Alan 28)

Animsatıcı	Değer	Açıklama
RST8818	000000000110	Sıfırlama Talimatı
BRA88181	011000111000	DRA Talimatına Şube
BRA88180	010000111110	DRA Talimatına Şube
INC88181	000000111110	Talimata Devam Et
INC88180	001000001000	Talimata Devam Et
CAL88181	010000110000	DRA Tarafından Belirtilen Adreste Alt Programa Atlama
CAL88180	010000101110	DRA Tarafından Belirtilen Adreste Alt Programa Atlama
RET8818	000000011010	Alt Yordamdan Dönüş
PUSH8818	0000000110111	Kesme Dönüş Adresini Yiğine İtin
POP8818	100000010000	Kesmeden Dönüş
LOADDRA	000010111110	DA Veriyolundan DRA Sayacı Yükle
LOADDRB	000110111110	DRB Sayacını DA Veriyolundan Yükle
YÜK BİNASI	000110111100	Yük DRA/DRB
DECRRDRA	010001111100	DRA Sayacını Azaltma ve Sıfır Değilse Dallanma
DECRRDRB	010101111100	DRB Sayacını Azalt ve Sıfır Değilse Dallan

Örnek olarak, INC88181 komutu, seçili durum kodu 1 ise, sıradaki bir sonraki mikro yapının seçilmesine neden olmak için kullanılır. Tablo 21.8'den

$$\text{INC88181} = 000000111110$$

doğrudan kod çözen

- **OSEL= 0:** RCA'yı DRA çıkış MUX'undan çıkış olarak seçer; bu durumda seçim önemsizdir.
- **SELDR= 0:** Daha önce tanımlandığı gibi; yine bu komut için bu konu önemsizdir.
- **ZEROIN= 0:** MUX değeri ile birlikte, dallanma yapılmaması gerektiğini belirtir.
- **R= 000:** RA ve RC'nin mevcut değerini koruyun.
- **S= 111:** Yiğinin mevcut durumunu korur.
- **MUX= 110:**= koşul kodu 1 olduğunda MPC'yi,= koşul kodu 0 olduğunda DRA'yı seçin.

### Kayıtlı ALU

8832, dört adet 8-bit ALU, iki adet 16-bit ALU veya tek bir 32-bit ALU olarak çalışacak şekilde yapılandırılabilen 64 kayitlı 32-bit bir ALU'dur.

8832, mikro komutun 17'den 27'ye kadar olan alanlarını oluşturan 39 bit tarafından kontrol edilir (Tablo 21.7); bunlar ALU'ya kontrol sinyalleri olarak verilir. Ayrıca, Şekil 21.17'de gösterildiği gibi, 8832'nin 32 bit DA veri yoluna ve 32 bit sistem Y veri yoluna harici bağlantıları vardır. DA'dan gelen girişler aynı anda 64 kelimelik kayıt dosyasına ve ALU mantık moduna giriş verisi olarak sağlanabilir. Sistem Y veriyolundan gelen girişler ALU mantık modülüne sağlanır. ALU ve kaydırma işlemlerinin sonuçları DA veri yoluna veya sistem Y veri yoluna gönderilir. Sonuçlar ayrıca dahili kayıt dosyasına da geri beslenebilir.

Üç adet 6-bit adres portu, iki operandlı bir getirme ve bir operand yazma işleminin aynı anda kayıt dosyası içinde gerçekleştirilemesine olanak tanır. Bir MQ kaydırıcı ve MQ register ayrıca çift hassasiyetli 8-bit, 16-bit ve 32-bit kaydırma işlemlerini uygulamak için bağımsız olarak çalışacak şekilde yapılandırılabilir.

Her mikro talimatın 17 ila 26 arasındaki alanları, 8832 içinde ve 8832 ile dış ortam arasında veri akışını kontrol eder. Alanlar aşağıdaki gibidir:

- 17. Yazma Etkinleştir.** Bu iki bit, kayıt dosyasına 32 bit, 16 en anlamlı bit, 16 en az anlamlı yazmayı veya yazmamayı belirtir. Hedef kayıt 24 numaralı alan tarafından tanımlanır.
- 18. Kayıt Dosyası Veri Kaynağını Seçin.** Kayıt dosyasına bir yazma işlemi yapılacaksas, bu iki bit kaynağı belirtir: DA veriyolu, DB veriyolu, ALU çıkışı veya sistem Y veriyolu.
- 19. Kaydırma Komutu Değiştiricisi.** Uç doldurma bitlerinin sağlanması ve kaydırma talimatları sırasında kaydirlan bitlerin okunmasıyla ilgili seçenekleri belirtir.
- 20. Taşıma Girişi.** Bu bit, bu işlem için ALU'ya bir bit taşınıp taşınmadığını gösterir.

- 21. ALU Yapılandırma Modu.** 8832, tek bir 32 bit ALU, iki 16 bit ALU veya dört 8 bit ALU olarak çalışacak şekilde yapılandırılabilir.
- 22. S Giriş.** ALU mantık modülü girişleri, S ve R olarak adlandırılan iki dahili çoklayıcı tarafından sağlanır. Bu alan S çoklayıcı tarafından sağlanacak girişin secer: kayıt dosyası, DB veri yolu veya MQ kaydı. Kaynak kayıt 25 numaralı alan tarafından tanımlanır.
- 23. R Giriş.** R çoklayıcı tarafından sağlanacak girişin secer: kayıt dosyası veya DA veri yolu.
- 24. Hedef Kaydı.** Hedef işlenen için kullanılacak kayıt dosyasındaki kaydın adresi.
- 25. Kaynak Kaydı.** S çoklayıcı tarafından sağlanan kaynak işlenen için kullanılacak kayıt dosyasındaki kaydın adresi.
- 26. Kaynak Kaydı.** R çoklayıcı tarafından sağlanan kaynak işlenen için kullanılacak kayıt dosyasındaki kaydın adresi.

Son olarak, alan 27, ALU tarafından gerçekleştirilecek aritmetik veya mantıksal işlevi belirten 8 bitlik bir işlem kodudur. Tablo 21.9 gerçekleştirilebilecek farklı işlemleri listeler.

17'den 27'ye kadar olan alanları belirtmek için kullanılan kodlamaya örnek olarak, yazmaç 1'in içeriğini yazmaç 2'ye ekleme ve sonucu yazmaç 3'e yerleştirme komutunu düşünün. Sembolik komut şöyledir

CONT11 [17], WELH, SELRFYMX, [24], R3, R2, R1, PASS+ ADD  
Birleştirici bunu uygun bit modeline çevirecektir. Komutun bireysel bileşenleri aşağıdaki gibi tanımlanabilir:

- CONT11 temel NOP komutudur.
- 17] alanı WELH (write enable, low and high) olarak değiştirilir, böylece 32 bitlik bir yazmaca yazılır.
- Alan [18], ALU Y MUX çıkışından geri beslemeyi seçmek için SELRFYMX olarak değiştirilir.
- Alan [24], hedef kayıt için R3 kaydını belirlemek üzere değiştirilir.
- Alan [25], kaynak kayıtlarından biri için R2 kaydını belirlemek üzere değiştirilir.
- Alan [26], kaynak kayıtlarından biri için R1 kaydını belirlemek üzere değiştirilir.
- Alan [27], ADD'nin ALU işlemini belirtecek şekilde değiştirilir. ALU kaydırıcı komutu PASS'tır; bu nedenle ALU çıkışı kaydırıcı tarafından kaydırılmaz.

Sembolik gösterim hakkında birkaç noktaya degeinilebilir. Ardışık alanlar için alan numarasını belirtmek gereklidir. Yani,

CONT11 [17],WELH, [18],SELRFYMX

olarak yazılabilir

CONT11 [17],WELH,SELRFYMX

çünkü SELRFYMX 18 numaralı alandadır.

Tablo 21.9'daki Grup 1'de yer alan ALU komutları her zaman Grup 2 ile birlikte kullanılmalıdır. Grup 3 ile 5'teki ALU komutları Grup 2 ile birlikte kullanılmamalıdır.

**Tablo 21.9** TI 8832 Kayıtlı ALU Komut Alanı (Alan 27)

<b>Grup 1</b>		<b>Fonksiyon</b>
ADD	H#01	R+ S+ Cn
SUBR	H#02	(R DEĞİL)+ S+ Cn
ŞUBELER	H#03	R= (S DEĞİL)+ Cn
INSC	H#04	S+ Cn
INCNS	H#05	(S DEĞİL)+ Cn
INCR	H#06	R+ Cn
INCNR	H#07	(R DEĞİL)+ Cn
XOR	H#09	R XOR S
VE	H#0A	R VE S
VEYA	H#0B	R VEYA S
NAND	H#0C	R NAND S
NOR	H#0D	R NOR S
ANDNR	H#0E	(R DEĞİL) VE S
<b>Grup 2</b>		<b>Fonksiyon</b>
SRA	H#00	Aritmetik sağ tek hassasiyetli kaydırma
SRAD	H#10	Aritmetik sağ çift duyarlıklı kaydırma
SRL	H#20	Mantıksal sağ tek hassasiyetli kaydırma
SRLD	H#30	Mantıksal sağ çift duyarlıklı kaydırma
SLA	H#40	Aritmetik sola tek hassasiyetli kaydırma
SLAD	H#50	Aritmetik sol çift hassasiyetli kaydırma
SLC	H#60	Dairesel sola tek hassasiyetli kaydırma
SLCD	H#70	Dairesel sol çift hassasiyetli kaydırma
SRC	H#80	Dairesel sağ tek hassasiyetli kaydırma
SRCD	H#90	Dairesel sağ çift hassasiyetli kaydırma
MQSRA	H#A0	Aritmetik sağa kaydırma MQ yazmacı
MQSRL	H#B0	Mantıksal sağa kaydırma MQ yazmacı
MQSLL	H#C0	Mantıksal sola kaydırma MQ yazmacı
MQSLC	H#D0	Dairesel sola kaydırma MQ yazmacı
LOADMQ	H#E0	MQ kaydını yükle
GEÇTİ	H#F0	ALU'yu Y'ye geçirin (kaydırma işlemi yok)
<b>Grup 3</b>		<b>Fonksiyon</b>
SET1	H#08	Bit 1'i ayarla
Set0	H#18	Bit 0'i ayarla
TB1	H#28	Test biti 1
TB0	H#38	Test biti 0
ABS	H#48	Mutlak değer
SMTC	H#58	İşaret büyüklüğü/twos-complement

<b>Grup 3</b>		<b>Fonksiyon</b>
ADDI	H#68	Hemen ekle
SUBI	H#78	Anında çıkarma
BADD	H#88	Bayt R'yi Sye ekler
BSUBS	H#98	Bayt S'yi R'den çıkarır
BSUBR	H#A8	Bayt S'den R'yi çıkarır
BINCS	H#B8	Bayt artışı S
BINCNS	H#C8	Bayt artışı negatif S
BXOR	H#D8	Bayt XOR R ve S
BAND	H#E8	Bayt VE R ve S
BOR	H#F8	Bayt VEYA R ve S
<b>Grup 4</b>		<b>Fonksiyon</b>
CRC	H#00	Döngüsel artıklık karakter birikimi.
SEL	H#10	S veya R'yı seçin
SNORM	H#20	Tek uzunluk normalleştirme
DNORM	H#30	Çift uzunluk normalleştirme
DIVRF	H#40	Kalan düzeltmeyi böl
SDIVQF	H#50	İşareti bölmeye bölümü düzeltmesi
SMULI	H#60	İşareti çarpma yinelemesi
SMULT	H#70	İşareti çarpma sonlandırma
SDIVIN	H#80	İşareti bölmeye başlatma
SDIVIS	H#90	İşareti bölmeye başlangıcı
SDIVI	H#A0	İşareti bölmeye yineleme
UDIVIS	H#B0	İşretsiz bölmeye başlangıcı
UDIVI	H#C0	İşretsiz bölmeye yineleme
UMULI	H#D0	İşretsiz çarpma yineleme
SDIVIT	H#E0	İşareti bölmeye sonlandırma
UDIVIT	H#F0	İşretsiz bölmeye sonlandırma
<b>Grup 5</b>		<b>Fonksiyon</b>
LOADFF	H#0F	Yük bölmeye/BCD flip-flopları
CLR	H#1F	Temiz
DUMPFF	H#5F	Çıkış bölmeye/BCD flip-flopları
BCDBIN	H#7F	BCD'den ikiliye
EX3BC	H#8F	Fazlalık (3 bayt düzeltme)
EX3C	H#9F	Excess (3 kelime düzeltme)
SDIVO	H#AF	İşareti bölmeye taşıma testi
BINEX3	H#DF	İkiliden fazlasına - 3
NOP32	H#FF	Operasyon yok

## 21.5 ANAHTAR TERIMLER, GÖZDEN GEÇİRME SORULARI VE PROBLEMLER

### Anahtar Terimler

kontrol belleği kontrol kelimesi firmware sert mikroprogramlama yatay mikro talimat	mikro talimat kodlaması mikro talimat yürütme mikro talimat sıralaması mikro talimatlar mikroprogram	mikroprogramlı kontrol ünitesi mikroprogramlama dili yumuşak mikro programlama paketlenmemiş mikro talimat dikey mikro talimat
---	--	--

### İnceleme Soruları

- 21.1** Bir kontrol ünitesinin kablolu uygulaması ile mikro programlı uygulaması arasındaki fark nedir?
- 21.2** Yatay bir mikro talimat nasıl yorumlanır?
- 21.3** Kontrol belleğinin amacı nedir?
- 21.4** Yatay bir mikro talimatın yürütülmesinde tipik bir sıra nedir?
- 21.5** Yatay ve dikey mikro talimatlar arasındaki fark nedir?
- 21.6** Mikro programlanmış bir kontrol ünitesi tarafından gerçekleştirilen temel görevler nelerdir?
- 21.7** Paketlenmiş ve paketlenmemiş mikro talimatlar arasındaki fark nedir?
- 21.8** Sert ve yumuşak mikro programlama arasındaki fark nedir?
- 21.9** İşlevsel ve kaynak kodlaması arasındaki fark nedir?
- 21.10** Mikro programlamanın bazı yaygın uygulamalarını listeleyiniz.

### Problemler

- 21.1** Wilkes tarafından tasarlanan varsayımsal makinede çarpma komutunun uygulanmasını açıklayınız. Anlatım ve bir akış şeması kullanınız.
  - 21.2** Aşağıdaki sembolik forma sahip bir mikro talimat içeren bir mikro talimat kümesi varsayıyalım:
- IF ( $AC_0 = 1$ ) THEN CAR  $\triangleleft$  ( $C_{(0)(-6)}$ ) ELSE CAR  $\triangleleft$  (CAR)+ 1
- Burada  $AC_0$  aktüümülatörün işaret bitidir ve  $C_{0-6}$  mikro komutun ilk yedi bitidir. Bu mikro komutu kullanarak, AC negatifse dallanan bir Dallanma Kaydı Eksi (BRM) makine komutunu uygulayan mikro program yazın. Mikro taliminin  $C_1$ 'den  $C_{(n)}$ e kadar olan bitlerinin paralel bir mikro işlem kümesi belirttiğini varsayıyın. Programı sembolik olarak ifade edin.
- 21.3** Basit bir işlemcinin komut döngüsünde dört ana aşama vardır: getirme, dolaylı, yürütme ve kesme. İki adet 1 bitlik bayrak, kablolu bir uygulamada mevcut aşamayı belirler.
    - a.** Bu bayraklara neden ihtiyaç duyuluyor?
    - b.** Mikro programlı bir kontrol ünitesinde neden bunlara ihtiyaç duyulmuyor?
  - 21.4** Şekil 21.7'deki kontrol ünitesini düşünün. Kontrol belleğinin 24 bit genişliğinde olduğunu varsayıyın. Mikro komut formatının kontrol kısmı iki alana bölünmüştür. 13 bitlik bir mikro işlem alanı gerçekleştirilecek mikro işlemleri belirtir. Bir adres seçim alanı, bayraklara dayalı olarak bir mikro komut dallanmasına neden olacak bir koşulu belirtir. Sekiz bayrak vardır.

- a.** Adres seçim alanında kaç bit vardır?
  - b.** Adres alanında kaç bit vardır?
  - c.** Kontrol belleğinin boyutu nedir?
- 21.5** Önceki problemin koşulları altında koşulsuz dallanma nasıl yapılabilir? Dallanmadan nasıl kaçınılabilir; yani, koşullu veya koşulsuz herhangi bir dallanma belirtmeyen bir mikro talimat tanımlayın.
- 21.6** Her makine komutu rutini için 8 kontrol kelimesi sağlamak istiyoruz. Makine komutu kodlarının 5 biti ve kontrol belleğinin 1024 kelimesi vardır. Komut yazmacından kontrol adres yazmacına bir eyleme önerin.
- 21.7** Kodlanmış bir mikro talimat formatı kullanılacaktır. 9 bitlik bir mikro-işlem alanının 46 farklı eylemi belirtmek için nasıl alt alanlara bölünebileceğini gösterin.
- 21.8** Bir 16 yazmaç, 16 mantık ve 16 aritmetik işlevli bir ALU ve 8 işlemli bir kaydırıcı bulunur ve bunların hepsi dahili bir işlemci veri yolu ile birbirine bağlıdır. İşlemci için çeşitli mikro işlemleri belirtmek üzere bir mikro komut formatı tasarlaymentınız.

## **BİLGİSAYAR ORGANİZASYONU VE MİMARISINI ÖĞRETMEK İÇİN PROJELER**

- A.1** Interaktif Simülasyonlar
- A.2** Araştırma Projeleri
- A.3** Simülasyon Projeleri
  - SimpleScalar SMPCache
- A.4** Assembly Dili Projeleri
- A.5** Okuma/Rapor Ödevleri
- A.6** Yazma Ödevleri
- A.7** Test Bankası

Birçok eğitmen, bilgisayar organizasyonu ve mimarisi kavramlarının net bir şekilde anlaşılması için araştırma veya uygulama projelerinin çok önemli olduğunu inanmaktadır. Projeler olmadan öğrencilerin bazı temel kavramları ve bileşenler arasındaki etkileşimleri kavraması zor olabilir. Projeler, kitapta tanıtılan kavramları pekiştirir, öğrencilere işlemcilerin ve bilgisayar sistemlerinin iç işleyişini daha iyi anlamalarını sağlar ve öğrencileri motive ederek onlara materyale hakim olduklarına dair güven verebilir.

Bu metinde, bilgisayar organizasyonu ve mimarisi kavramlarını olabildiğince açık bir şekilde sunmaya çalıştım ve bu kavramları pekiştirmek için çok sayıda ev ödevi problemi sundum. Birçok eğitmen bu materyali projelerle desteklemek isteyecektir. Bu ek, bu konuda bazı rehberlik sağlamakta ve eğitimmenler tarafından Pearson'dan çevrimiçi olarak erişilebilen bu kitap için **Eğitmen Kaynak Merkezi'nde (IRC)** bulunan destek materyalini açıklamaktadır. Destek materyali altı tip proje ve diğer öğrenci alıştırmalarını kapsamaktadır:

- Etkileşimli simülasyonlar
- Araştırma projeleri
- Simülasyon projeleri
- Assembly dili projeleri
- Okuma/rapor ödevleri
- Yazma ödevleri
- Test bankası

## A.1 ETKİN SİMÜLASYONLAR

Etkileşimli simülasyonlar, modern bir bilgisayar sisteminin karmaşık tasarım özelliklerini anlamak için güçlü bir araç sağlar. Günümüz öğrencileri çeşitli karmaşık bilgisayar sistemleri mekanizmalarını kendi bilgisayar ekranlarında görselleştirebilmek istemektedirler. Bilgisayar organizasyonu ve mimari tasarımındaki temel işlevleri ve algoritmaları göstermek için toplam 20 simülasyon kullanılmaktadır. Tablo A.1 simülasyonları bölümlere göre listelemektedir. Kitabın ilgili noktasında, bir simge, ilgili etkileşimli simülasyonun öğrencilerin kullanımı için çevrimiçi olarak mevcut olduğunu gösterir.

Simülasyonlar kullanıcının başlangıç koşullarını belirlemesine olanak tanıyıldından, öğrenci ödevleri için temel oluşturabilirler. Bu kitap için IRC, interaktif simülasyonların her biri için bir set olmak üzere bir dizi ödev içermektedir. Her ödev, öğrencilere verilebilecek birkaç özel problem içermektedir.

Etkileşimli simülasyonlar Massachusetts Üniversitesi Elektrik ve Bilgisayar Mühendisliği Bölümü'nden Profesör Israel Koren yönetiminde geliştirilmiştir. Massachusetts Üniversitesi'nden Aswin Sreedhar interaktif simülasyon ödevlerini geliştirmiştir. Animasyonlara erişmek için, bu kitabın <http://williamstallings.com/> ComputerOrganization adresindeki web sitesinde dönen küreye tıklayın.

## 770 EK A / BİLGİSAYAR ORGANİZASYONU ÖĞRETİMİ İÇİN PROJELER

**Tablo A.1** Bölümlere Göre Bilgisayar Organizasyonu ve Mimarisi-İnteraktif Simülasyonlar

<b>Bölüm 4-Önbellek</b>	
Önbellek Simülörü	Kullanıcı tarafından girilen bir önbellek modeline dayalı olarak küçük boyutlu önbellekleri taklit eder ve simülasyon döngüsünün sonunda kullanıcı tarafından girilen veya seçiliirse rastgele oluşturulan bir giriş dizisine dayalı olarak önbellek içeriğini görüntüler.
Önbellek Zaman Analizi	Belirtilmişiniz önbellek parametreleri için Ortalama Bellek Erişim Süresi analizini gösterir.
Çoklu Görev Önbellek Göstericisi	Çoklu görevi destekleyen bir sisteme önbellek modelleri.
Seçici Kurban Önbellek Simülörü	Üç farklı önbellek ilkesini karşılaştırır.
<b>Bölüm 5-Dahili Bellek</b>	
Aralıklı Bellek Simülörü	Araya serpiştirilmiş belleğin etkisini gösterir.
<b>Bölüm 6-Harici Bellek</b>	
RAID	Depolama verimliliğini ve güvenilirliğini belirleyen.
<b>Bölüm 7-Giriş/Çıkış</b>	
I/O Sistem Tasarım Aracı	Farklı I/O sistemlerinin karşılaştırmalı maliyet ve performansını değerlendirir.
<b>Bölüm 8-OS Desteği</b>	
Sayfa Değiştirme Algoritmaları	LRU, FIFO ve Optimal'i karşılaştırır.
Daha Fazla Sayfa Değiştirme Algoritması	Bir dizi poliçeyi karşılaştırır.
<b>Bölüm 14-CPU Yapısı ve İşlevi</b>	
Rezervasyon Tablosu Analizörü	Boru hatlı bir sistemin görev akış modelini temsil etmenin bir yolu olan rezervasyon tablolarını değerlendirir.
Şube Tahmini	Üç farklı dal tahmin şemasını gösterir.
Şube Hedef Tamponu	Kombine dal tahlincisi/dal hedef tampon simülörü.
<b>Bölüm 15-Azaltılmış Komut Kümeli Bilgisayarlar</b>	
MIPS 5 Aşamalı Boru Hattı	Boru hattını simüle eder.
Döngü Açıma	Komut düzeyinde paralellikten yararlanmak için döngü açma yazılım teknigini simüle eder.
<b>Bölüm 16 - Komut Düzeyinde Paralellik ve Süperskalar İşlemciler</b>	
Statik ve Dinamik Çizelgelemeli Boru Hattı	MIPS boru hattının daha karmaşık bir simülasyonu.
Yeniden Sıralama Tampon Simülatörü	Bir RISC boru hattında komut yeniden sıralamasını simüle eder.
Dinamik Çizelgeleme için Puanlama Tekniği	Bir dizi işlemcide kullanılan bir komut çizelgeleme tekniğinin simülasyonu.
Tomasulo'nun Algoritması	Başka bir komut çizelgeleme tekniğinin simülasyonu.
Tomasulo Algoritmasının Alternatif Simülasyonu	Tomasulo'nun algoritmasının bir başka simülasyonu.
<b>Bölüm 17-Paralel İşleme</b>	
Vektör İşlemci Simülasyonu	Vektör işleme talimatlarının yürütülmesini gösterir.

## A.2 RESE ARCH PROJELERİ

Dersteki temel kavramları pekiştirmenin ve öğrencilere araştırma becerilerini öğretmenin etkili bir yolu da bir araştırma projesi vermektir. Böyle proje, literatür taramasının yanı sıra satıcı ürünleri, araştırma laboratuvarı faaliyetleri ve standardizasyon çabalarının Web'de araştırılmasını da içerebilir. Projeler ekiplere ya da daha küçük projeler için bireylere verilebilir. Her durumda, dönemde bir tür proje önerisi istemek en iyisidir, böylece eğitmene öneriyi uygun konu ve uygun çaba düzeyi açısından değerlendirmesi için zaman tanınmış olur. Araştırma projeleri için öğrenci bildirileri şunları içermelidir

- Teklif için bir format
- Nihai rapor için bir format
- Ara ve son teslim tarihlerini içeren bir program
- Olası proje konularının bir listesi

Öğrenciler listelenen konulardan birini seçebilir veya kendi karşılaştırılabilir projelerini tasarlayabilirler. IRC, teklif ve nihai rapor için önerilen bir formatın yanı sıra olası araştırma konularının bir listesini de içermektedir.

## A.3 SİMÜLASYON PROJELERİ

Bir işlemcinin iç işleyişini kavramak ve bazı tasarım ödünlendirimlerini ve performans etkilerini incelemek ve değerlendirmek için mükemmel bir yol, işlemcinin temel unsurlarını simüle etmektir. Bu amaç için yararlı olan iki araç SimpleScalar ve SMPCache'dir.

Gerçek donanım uygulaması ile karşılaşıldığında, simülasyon hem araştırma hem de eğitim amaçlı kullanım için iki avantaj sağlar:

- Simülasyon ile bir organizasyonun çeşitli unsurlarını değiştirmek, çeşitli bileşenlerin performans özelliklerini değiştirmek ve daha sonra bu değişikliklerin etkilerini analiz etmek kolaydır.
- Simülasyon, performans ödünlendirimlerini anlamak için kullanılabilecek ayrıntılı performans istatistiklerinin toplanmasını sağlar.

### SimpleScalar

SimpleScalar [BURG97, MANJ01a, MANJ01b], bir dizi modern işlemci ve sistem üzerinde gerçek programları simüle etmek için kullanılabilecek bir araç setidir. Araç seti derleyici, birleştirici, bağlayıcı ve simülasyon ve görselleştirme araçlarını içerir. Simple- Scalar, son derece hızlı bir işlevsel simülatörden ayrıntılı bir sıra dışı soruna, engellemeyen önbellekleri ve spekulatif yürütmemi destekleyen süperskalar işlemci simülatörüne kadar değişen işlemci simülatörleri sağlar. Komut seti mimarisи ve organizasyonel parametreler çeşitli deneyler oluşturmak için değiştirilebilir.

Bu kitabın IRC'si, öğrenciler için SimpleScalar'a kısa bir giriş ve SimpleScalar'ın nasıl yükleneceği ve kullanılmaya başlanacağı ile ilgili talimatlar içermektedir. Kılavuzda ayrıca önerilen bazı proje ödevleri de yer almaktadır.

SimpleScalar, çoğu UNIX platformunda çalışan taşınabilir bir yazılım paketidir. SimpleScalar yazılımı SimpleScalar Web sitesinden indirilebilir. Ticari olmayan kullanım için ücretsiz olarak temin edilebilir.

### **SMP Cache Önbellek**

SMP Cache, simetrik çoklu işlemcilerdeki önbellek sistemlerinin analizi ve öğretimi için iz odaklı bir simülatördür [RODR01]. Simülasyon, bu sistemlerin mimari temel ilkelerine göre oluşturulmuş bir modeldir. Simülatör tam grafik ve kullanıcı dostu bir arayüze sahiptir. Simülatör ile çalışılabilen parametrelerden bazıları şunlardır: program yerelligi; işlemci sayısının etkisi, önbellek tutarlılık protokoller, veri yolu tahkimi için şemalar, eşleme, değiştirme politikaları, önbellek boyutu (önbellekteki bloklar), önbellek kümelerinin sayısı (küme ilişkisel önbellekler için) ve blok başına kelime sayısı (bellek bloğu boyutu).

Bu kitabın IRC'si öğrenciler için SMP Cache'e kısa bir giriş ve SMP Cache'in nasıl yükleneceği ve kullanılmaya başlanacağı ile ilgili talimatlar içermektedir. Kılavuzda ayrıca önerilen bazı proje ödevleri de yer almaktadır.

SMP Cache, Win- dows yüklü PC sistemlerinde çalışan taşınabilir bir yazılım paketidir. SMP Cache yazılımı SMP Cache Web sitesinden indirilebilir. Ticari olmayan kullanım için ücretsiz olarak temin edilebilir.

## **A.4 ASSEMBLY DİL PROJELERİ**

Assembly dili programlama genellikle öğrencilere düşük seviyeli donanım bileşenlerini ve bilgisayar mimarisinin temellerini öğretmek için kullanılır. CodeBlue, ABD Hava Kuvvetleri Akademisi'nde geliştirilen basitleştirilmiş bir assembly dili programıdır. Çalışmanın amacı, öğrencilerin tek bir sınıfta öğrenebilecekleri görsel bir simülatör kullanarak assembly dili kavramlarını geliştirmek ve öğretmektir. Geliştiriciler ayrıca öğrencilerin bu dili motive edici ve eğlenceli bulmalarını istiyordu. CodeBlue dili, SC123 gibi çoğu basitleştirilmiş mimari komut setinden çok daha basittir. Yine de öğrencilerin çok daha karmaşık SPIMbot simülatörüne benzer şekilde turnuvalarda yarışan ilginç assembly düzeyinde programlar geliştirmelerine olanak tanır. En önemlisi, CodeBlue programlama sayesinde öğrenciler, komutlar ve verilerin bellekte birlikte bulunması, kontrol yapısı uygulaması ve adresleme modları gibi temel bilgisayar mimarisi kavramlarını öğrenirler.

Projeler için bir temel sağlamak amacıyla geliştiriciler, öğrencilerin bir program oluşturmamasına, bellekteki temsiliğini görmesine, programın yürütülmesi boyunca adım atmasına ve görsel bir bellek ortamında rakip programların savaşını simüle etmesine olanak tanıyan görsel bir geliştirme ortamı oluşturmuşlardır.

Projeler Core War turnuvası konsepti etrafında oluşturulabilir. Core War, 1980'lerin başında halka tanıtılan ve 15 yıl kadar bir süre popüler bir programlama oyunudur. Core War'in dört ana bileşeni vardır: 8000 adresli bir bellek dizisi, basitleştirilmiş bir assembly dili Redcode, MARS adı verilen bir yürütme programı (Memory Array Redcode Simulator'un kısaltması) ve çekişen savaş programları kümlesi. İki savaş programı rastgele seçilen konumlarda bellek dizisine girilir; iki program da diğerinin nerede olduğunu bilmey.

diğer ise. MARS programları zaman paylaşımının basit bir versiyonunda yürütür. İki program sırayla : ilk programın tek bir komutu, ardından ikincinin tek bir komutu ve bu şekilde devam eder. Bir savaş programının kendisine ayrılan yürütme döngüleri sırasında ne yapacağı tamamen programcıya bağlıdır. Amaç, diğer programın talimatlarını bozarak onu yok etmektir. CodeBlue ortamı Redcode yerine CodeBlue'yı kullanır ve kendi etkileşimli yürütme arayüzünü sağlar.

IRC, CodeBlue ortamını, öğrenciler için bir kullanım kılavuzunu, diğer destekleyici materyalleri ve önerilen ödevleri içerir.

## A.5 rEADING/rEPORT GÖREVLENDİRMELERİ

Dersteki kavramları pekiştirmenin ve öğrencilere araştırma deneyimi kazandırmmanın bir başka mükemmel yolu da literatürden okunacak ve analiz edilecek makaleler atamaktır. IRC, bölümlere göre düzenlenmiş, atanacak makalelerin önerilen bir listesini içerir. Premium Content Web sitesi her bir makalenin bir kopyasını sunmaktadır. IRC ayrıca önerilen bir ödev ifadesi de içermektedir.

## A.6 YAZMA ÖDEVLERİ

Yazma ödevleri, bilgisayar organizasyonu ve mimarisi gibi teknik bir disiplinde öğrenme sürecinde güçlü bir çapran etkisine sahip olabilir. Writing Across the Curriculum (WAC) hareketinin taraftarları (<http://wac.colostate.edu/>) yazma ödevlerinin öğrenmeyi kolaylaştırmada önemli faydaları olduğunu bildirmektedir. Yazma ödevleri, belirli bir konu hakkında daha detaylı ve eksiksiz düşünmeyi sağlar. Buna ek olarak, yazma ödevleri, öğrencilerin bir en az kişisel katılımla takip etme eğiliminin üstesinden gelmeye yardımcı olur, konu hakkında derin bir anlayış elde etmeden sadece gerçekleri ve problem çözme tekniklerini öğrenir.

IRC, bölümlere göre düzenlenmiş bir dizi önerilen yazma ödevi içermektedir. Eğitmenler sonuça bunun materyali öğretme yaklaşımlarının en önemli parçası olduğunu düşünebilirler. Bu alandaki geri bildirimleriniz ve ilave yazma ödevlerine ilişkin önerileriniz için çok teşekkür ederim.

## A.7 TEST BANKASI

Kitap için bir test bankası bu kitabı için IRC sitesinde mevcuttur. Test bankası her bölüm için doğru/yanlış, çoktan seçmeli ve boşluk doldurmaları sorular içermektedir. Test bankası, öğrencilerin materyali anlamalarını değerlendirmek için etkili bir yoldur.

# **E**k B

---

## **ASSEMBLY DİLİ VE İLGİLİ KONULAR**

### **B.1 Assembly Dili**

Assembly Dili Öğeleri  
Assembly Dili İfade Türleri Örnek: En Büyük Ortak  
Bölen Programı

### **B.2 Montajcılar**

İki Geçişli Birleştirici  
Tek Geçişli Birleştirici  
Örnek: Asal Sayı Programı

### **B.3 Yükleme ve Bağlama**

Yer  
Değiştirme  
Yükleme  
Bağlama

### **B.4 Anahtar Terimler, Gözden Geçirme Soruları ve Problemler**

Assembly dili konusu Bölüm 13'te kısaca tanıtılmıştır. Bu ekte daha fazla ayrıntı verilmekte ve ayrıca ilgili bazı konular ele alınmaktadır. Assembly dilinde programlamayı (daha yüksek seviyeli bir dilde programlamaya kıyasla) öğrenmenin faydalı olması için aşağıdakiler de dahil olmak üzere birçok neden vardır:

- 1.** Talimatların yürütülmesini açılığa kavuşturur.
- 2.** Verilerin bellekte nasıl temsil edildiğini gösterir.
- 3.** Bir programın işletim sistemi, işlemci ve I/O sistemi ile nasıl etkileşime girdiğini gösterir.
- 4.** Bir programın harici cihazlara nasıl eriştiğini açıklar.
- 5.** Assembly dili programcılarını anlamak, öğrencilere HLL'nin çevrilmesi gereken hedef dil hakkında daha iyi bir fikir vererek onları daha yüksek seviyeli dil (HLL) programcılar yapar.

Bu bölüme, örneklerimiz için x86 mimarisini kullanarak bir assembly dilinin temel öğelerini inceleyerek başlıyoruz.<sup>1</sup> Daha sonra, assembler'in çalışmasına bakıyoruz. Bunu bağlayıcılar ve yükleyicilerin tartışılması takip etmektedir.

Tablo B.1 bu ekte kullanılan bazı anahtar terimleri tanımlamaktadır.

## B.1 ASSEMBLY DİLİ

Assembly dili, makine dilinden bir adım uzakta olan bir programlama dilidir. Tipik olarak, her bir assembly dili talimi assemblers tarafından bir makine talimatına çevrilir. Assembly dili donanıma bağlıdır ve her işlemci türü için farklı bir assembly dili vardır. Özellikle, assembly dili talimatları işlemecideki belirli kayıtlara atıfta bulunabilir, işlemcinin tüm opcode'larını içerebilir ve işlemecinin çeşitli kayıtlarının ve makine dilinin operandlarının bit uzunluğunu yansıtabilir. Bu nedenle bir assembly dili programcısı bilgisayarın mimarisini anlamak zorundadır.

Programcılar uygulamalar ve hatta sistem programları için assembly dilini nadiren kullanırlar. HLL'ler programcının görevlerini büyük ölçüde kolaylaştırın bir ifade gücü ve özlülük sağlar. Bir HLL assembly dili kullanmanın dezavantajları arasında aşağıdakiler yer alır [FOG08]:

- 1. Geliştirme süresi.** Assembly dilinde kod yazmak, yüksek seviyeli bir dilde yazmaktan çok daha uzun sürer.
- 2. Güvenilirlik ve güvenlik.** Assembly kodunda hata yapmak kolaydır. Assembler, çağrıma kurallarına ve kayıt kaydetme kurallarına uyulmadığını kontrol etmez. Kimse sizin için PUSH ve POP talimatlarının sayısının tüm olası dallarda ve yollarda aynı olup olmadığını kontrol etmiyor. Assembly kodunda o kadar çok gizli hata olasılığı vardır ki, test etme ve doğrulama konusunda çok sistematik bir yaklaşımınız yoksa bu durum projenin güvenilirliğini ve güvenliğini etkiler.

---

<sup>1</sup> x86 mimarisini için bir dizi assemblers vardır. Örneklerimizde açık kaynak kodlu bir assembler olan NASM (Netwide Assembler) kullanılmaktadır. NASM kılavuzunun bir kopyası bu kitabın Premium İçerik sitesinde bulunmaktadır.

## 776 EK B / TOPLANTI ALANI VE İLGİLİ KONULAR

**Tablo B.1** Bu Ek için Anahtar Terimler

<b>Montajci</b>
Assembly dilini makine koduna çeviren bir program.
<b>Assembly Dili</b>
Belirli bir işlemcinin makine dilinin, program yazımını kolaylaştıran ve assembler'a talimatlar sağlayan ek ifade türleriyle zenginleştirilmiş sembolik bir gösterimi.
<b>Derleyici</b>
Başka bir programı kaynak dilden (veya programlama dilinden) makine diline (nesne kodu) dönüştüren bir program. Bazı derleyiciler assembly dili çıktılarını verir ve bu dil daha sonra ayrı bir assembler tarafından makine diline dönüştürülür. Bir derleyici, her bir girdi ifadesinin genel olarak tek bir makine talimatına veya sabit talimat dizisine karşılık gelmemesiyle bir assembler'dan ayrılır. Bir derleyici değişkenlerin otomatik tahsisi, keyfi aritmetik ifadeler, FOR ve WHILE döngüler gibi kontrol yapıları, değişken kapsamı, giriş/çıkış işlemleri, üst düzey fonksiyonlar ve kaynak kodun taşınabilirliği gibi özellikleri destekleyebilir.
<b>Yürüttülebilir Kod</b>
Birleştirici veya derleyici gibi bir kaynak kod dili işlemcisini tarafından üretilen makine kodu. Bu, bilgisayarda çalıştırılabilen formda bir yazılımdır.
<b>Talimat Seti</b>
Belirli bir bilgisayar için mümkün tüm talimatların koleksiyonu; yani, belirli bir işlemcinin anladığı makine dili talimatlarının koleksiyonu.
<b>Bağlayıcı</b>
Ayrı ayrı derlenmiş program modüllerinden nesne kodu içeren bir veya daha fazla dosyayı yüklenebilir veya çalıştırılabilir kod içeren tek bir dosyada birlestiren bir yardımcı program.
<b>Yükleyici</b>
Yürüttülebilir bir programı yürütülmek üzere belleğe kopyalayan bir program rutini.
<b>Makine Dili veya Makine Kodu</b>
Bir bilgisayar programının bilgisayar tarafından okunan ve yorumlanan ikili gösterimi. Makine kodundaki bir program, bir dizi makine talimatından (muhtemelen verilerin arasına serpiştirilmiş) oluşur. Talimatlar, hepsi aynı boyutta (örneğin, birçok modern RISC mikroişlemci için bir 32 bit kelime) veya farklı boyutlarda olabilen ikili dizelerdir.
<b>Nesne Kodu</b>
Programlama kaynak kodunun makine dilindeki gösterimi. Nesne kodu bir derleyici veya birleştirici tarafından oluşturulur ve daha sonra bağlayıcı tarafından çalıştırılabilir koda dönüştürülür.

- 3. Hata ayıklama ve doğrulama.** Assembly kodunda hata ayıklama ve doğrulama daha zordur çünkü yüksek seviyeli koda göre daha fazla hata olasılığı vardır.
- 4. Sürdürülebilirlik.** Assembly kodunun değiştirilmesi ve bakımı daha zordur çünkü dil, yapılandırılmamış spaghetti koduna ve başkalarının anlaması zor olan her türlü hileye izin verir. Kapsamlı dokümantasyon ve tutarlı bir programlama stili gereklidir.
- 5. Taşınabilirlik.** Assembly kodu platforma özgüdür. Farklı bir platforma taşımak zordur.

- 6. Sistem kodu assembly yerine içsel fonksiyonları kullanabilir.** En iyi modern C++ derleyicileri, sistem kontrol kayıtlarına ve diğer sistem yönergelerine erişmek için içsel işlevlere sahiptir. İçsel işlevler mevcut olduğunda aygit sürücülerini ve diğer sistem kodları için assembly koduna artık gerek yoktur.
- 7. Uygulama kodu assembly yerine içsel fonksiyonları veya vektör sınıflarını kullanabilir.** En iyi modern C++ derleyicileri, daha önce assembly programlama gerektiren vektör işlemleri ve diğer özel talimatlar için içsel fonksiyonlara sahiptir.
- 8. Derleyiciler son yıllarda çok geliştirildi.** En iyi derleyiciler artık oldukça iyi. En iyi C++ derleyicisinden daha iyi optimize etmek için çok fazla uzmanlık ve deneyim gereklidir.

Yine de montaj dilinin ara sıra kullanılmasının bazı avantajları vardır, bunlar arasında aşağıdakiler de yer almaktadır [FOG08a]:

- 1. Hata ayıklama ve doğrulama.** Hata ayıklayıcıda derleyici tarafından oluşturulan montaj koduna veya demontaj penceresine bakmak, hataları bulmak ve derleyicinin belirli bir kod parçasını ne kadar iyi optimize ettiğini kontrol etmek için kullanışlıdır.
- 2. Derleyici yapmak.** Assembly kodlama tekniklerini anlamak, derleyiciler, hata ayıklayıcılar ve diğer geliştirme araçlarını yapmak için gereklidir.
- 3. Gömülü sistemler.** Küçük gömülü sistemler, PC'lere ve ana bilgisayarlara göre daha az kaynağa sahiptir. Assembly programlama, küçük gömülü sistemlerde kod hız veya boyut açısından optimize etmek için gerekli olabilir.
- 4. Donanım sürücüleri ve sistem kodu.** Donanıma, sistem kontrol kayıtlarına ve benzerlerine erişmek bazen yüksek seviye kodla zor veya imkansız olabilir.
- 5. Yüksek seviyeli dilden erişilemeyen talimatlara erişim.** Bazı assembly talimatlarının yüksek seviyeli dilde karşılığı yoktur.
- 6. Kendi kendini değiştiren kod.** Kendi kendini değiştiren kod genellikle karlı değildir çünkü verimli kod önbelleğe almayı engeller. Bununla birlikte, örneğin kullanıcı tanımlı bir fonksiyonun birçok kez hesaplanması gereken matematik programlarına küçük bir derleyici eklemek avantajlı olabilir.
- 7. Boyut için kod optimizasyonu.** Günümüzde depolama alanı ve bellek o kadar ucuzdur ki kod boyutunu küçültmek için assembly dilini kullanmaya değil. Bununla birlikte, önbellek boyutu hala o kadar kritik bir kaynaktır ki, bazı durumlarda kritik bir kod parçasını kod önbelleğine sığdırmak için boyuta göre optimize etmek yararlı olabilir.
- 8. Hız için kod optimizasyonu.** Modern C++ derleyicileri çoğu durumda kodu oldukça iyi optimize eder. Ancak derleyicilerin kötü performans gösterdiği ve dikkatli bir derleme programlamasıyla hızda önemli artışların elde edilebileceği durumlar hala vardır.
- 9. Fonksiyon kütüphaneleri.** Kod optimizasyonunun toplam faydası, birçok programcı tarafından kullanılan fonksiyon kütüphanelerinde daha yüksektir.
- 10. Fonksiyon kütüphanelerini birden fazla derleyici ve işletim sistemiyle uyumlu hale getirme.** Farklı derleyiciler ve farklı işletim sistemleri ile uyumlu birden fazla girişe sahip kütüphane fonksiyonları oluşturmak mümkündür. Bunun için assembly programlama gereklidir.

*Assembly dili ve makine dili* terimleri bazen yanlışlıkla eşanlamlı olarak kullanılmaktadır. Makine dili, işlemci tarafından doğrudan çalıştırılabilen talimatlardan oluşur. Her makine dili komutu, bir işlem kodu, işlenen referansları ve belki de bayraklar gibi yürütmeyle ilgili diğer bitleri içeren ikili bir dizedir. Kolaylık sağlamak için, bir komutu bit dizisi olarak yazmak yerine, işlem kodları ve kayıtlar için isimlerle sembolik olarak . Assembly dili, belirli ana bellek konumlarına ve belirli komut konumlarına isimler atamak da dahil olmak üzere sembolik isimlerden çok daha fazla yararlanır. Assembly dili ayrıca doğrudan çalıştırılamayan ancak bir assembly dili programından makine kodu üreten assembler için talimat görevi gören ifadeler içerir.

## Assembly Dili Öğeleri

Tipik bir assembly dilindeki bir deyim Şekil B.1'de gösterilen biçimde sahiptir. Dört öğeden oluşur: etiket, animsatıcı, işlenen ve yorum.

**ETİKET** Bir etiket mevcutsa, assembler etiketi o komut için oluşturulan nesne kodunun ilk baytinın yükleneceği adrese eşdeğer olarak tanımlar. Programcı daha sonra etiketi bir adres olarak veya başka bir komutun adres alanında veri olarak kullanabilir. Birleştirici, bir nesne programı oluştururken etiketi atanın değerle değiştirir. Etiketler en sık dallanma talimatlarında kullanılır.

Örnek olarak, işte bir program parçası:

```
L2: SUB EAX, EDX      ;EDX yazmacının içeriğini şuradan çıkar
                    ;EAX'ın içeriği ve sonucu EAX JG L2'de saklayın
                    ;çıkarma işleminin sonucu L2 ise L2'ye atla
                    ;pozitif
```

Program, sonuç sıfır veya negatif olana kadar L2 konumuna geri dönmeye devam edecektir. Böylece, `jg` komutu yürütüldüğünde, sonuç pozitifse, işlemci L2 etiketine eşdeğer adresi program sayacına yerleştirir.

Etiket kullanma nedenleri arasında aşağıdakiler yer almaktadır;

1. Etiket, bir program konumunun bulunmasını ve hatırlamasını kolaylaştırır.
2. Etiket, bir programı düzeltmek için kolayca taşınabilir. Birleştirici, program yeniden bir araya getirildiğinde etiketi kullanan tüm komutlardaki adresi otomatik olarak değiştirecektir.
3. Programcının görelî veya mutlak bellek adreslerini hesaplaması gerekmez, sadece gerektiğinde etiketleri kullanır.



**Şekil B.1** Assembly Dili Deyim Yapısı

**MNEMONIC** Mnemonic, assembly dili deyiminin işleminin veya işlevinin adıdır. Daha sonra tartışılacağı gibi, bir deyim bir makine komutuna, bir assembler direktifine veya bir makroya karşılık gelebilir. Bir makine komutu söz konusu olduğunda, animsatıcı belirli bir işlem kodıyla ilişkili sembolik ismidir.

Tablo 12.8, x86 talimatlarının çoğunun animsatıcısını veya talimat adını listeler. CART06] Ek A, x86 komutlarını, her biri için işlem ve komutun durum kodları üzerindeki etkisi ile birlikte listeler. NASM kılavuzunun Ek B'si her x86 komutu için daha ayrıntılı bir açıklama sağlar. Her iki belge de bu kitabın Premium İçerik sitesinde mevcuttur.

**OPERAND(LAR)** Bir assembly dili deyimi sıfır veya daha fazla operand içerir. Her operand bir anlık değeri, bir register değerini veya bir bellek konumunu tanımlar. Tipik olarak, assembly dili üç tür işlenen referansı arasında ayırmak için kurallar ve adresleme modunu belirtmek için kurallar sağlar.

x86 mimarisi için, bir assembly dili ifadesi bir yazmaç işlecine adıyla başvurabilir. Şekil B.2, sembolik adları ve bit kodlamalarıyla birlikte genel amaçlı x86 yazmaçlarını göstermektedir. Montajçı, sembolik ismi yazmaç için ikili tanımlayıcıya çevirecektir.

Genel amaçlı kayıtlar			0 16 bit	32-bit
	AH	AL	AX	EAX (000)
	BH	BL	BX	EBX (011)
	CH	CL	CX	ECX (001)
	DH	DL	DX	EDX (010)
				ESI (110)
				EDI (111)
				EBP (101)
				ESP (100)

Segment kayıtları	
15	0
	CS
	DS
	SS
	ES
	FS
	GS

Şekil B.2 Intel x86 Programlama

Bölüm 11.2'de tartışıldığı gibi, x86 mimarisi, her biri assembly dilinde sembolik olarak ifade edilmesi gereken zengin bir adresleme modları kümesine sahiptir. Burada yaygın örneklerden birkaçına değineceğiz. Yazmaç **adresleme** için, **yazmaç** adı komutta kullanılır. Örneğin, MOV ECX, EBX, EBX yazmacının içeriğini ECX yazmacına kopyalar. Anlık adresleme, değerin komutta kodlandığını gösterir. Örneğin, MOV EAX, 100H, 100 onaltılık değerini EAX yazmacına kopyalar. Anlık değer, B son ekine sahip ikili bir sayı veya son eki olmayan ondalık bir sayı olarak ifade edilebilir. Dolayısıyla, bir öncekine eşdeğer durumlar MOV EAX, 100000000 ve MOV EAX, 256'dır. **Doğrudan adresleme** bir bellek konumunu ifade eder ve DS segment register'ından bir yer değiştirmeye olarak ifade edilir. Bu en iyi örnekle açıklanabilir. 16 bitlik veri segmenti kaydı DS'nin 1000H değerini içerdigini varsayıyın. Ardından aşağıdaki sıra gerçekleşir:

```
MOV AX, 1234H MOV  
[3518H], AX
```

İlk olarak 16 bitlik AX kaydı 1234H olarak başlatılır. Ardından, ikinci satırda, AX'in içeriği DS:3518H mantıksal adresine taşınır. Bu adres, DS içeriğinin 4 bit sola kaydırılması ve 3518H eklenerek 32 bit mantıksal adres 13518H'nin oluşturulmasıyla oluşturulur.

**YORUM** Tüm assembly dilleri programa yorum yerleştirilmesine izin verir. Bir yorum bir assembly deyiminin sağ ucunda yer alabilir ya da bir metin satırının tamamını kaplayabilir. Her iki durumda da yorum, assembler'a satırın geri kalanının bir yorum olduğunu ve assembler tarafından göz ardi edileceğini bildiren özel bir karakterle başlar. Genellikle, x86 mimarisi için assembly dilleri özel karakter için noktalı virgül (;) kullanır.

### Assembly Dili İfadelerinin Türü

Assembly dili deyimleri dört türden biridir: komut, yönerge, makro tanımı ve yorum. Yorum deyimi, tamamen bir oluşan bir deyimdir. Kalan türler bu bölümde kısaca açıklanmıştır.

**INSTRUCTIONS** Bir assembly dili programındaki yorum içermeyen ifadelerin büyük bir kısmı makine dili talimatlarının sembolik gösterimleridir. Neredeyse değişmez bir şekilde, bir assembly dili komutu ile bir makine komutu arasında bire bir ilişki vardır. Montajçı sembolik referansları çözer ve assembly dili talimatını makine talimatını oluşturan ikili dizeye çevirir.

**DİREKTİFLER** **Sözde talimatlar** olarak da adlandırılan direktifler, doğrudan makine dili talimatlarına çevrilmeyen assembly dili ifadeleridir. Bunun yerine, yönergeler assembler'a assembly işlemini yaparken belirli eylemleri gerçekleştirmesi için verilen talimatlardır. Örnekler aşağıdakileri içerir:

- Sabitleri tanımlayın
- Veri depolama için bellek alanları belirleme
- Bellek alanlarını başlatma
- Tabloları veya diğer sabit verileri belleğe yerleştirme
- Diğer programlara referanslara izin verin

Tablo B.2 NASM direktiflerinden bazılarını listelemektedir. Örnek olarak, aşağıdaki ifade dizisini göz önünde bulundurun:

**Tablo B.2** Bazı NASM Assembly Dili Direktifleri

(a) *RESx* ve *Dx* Direktifleri için Mektuplar

Birim	Mektup
bayt	B
kelime (2 bayt)	W
çift kelime (4 bayt)	D
dörtlü kelime (8 bayt)	Q
on bayt	T

(b) Direktifler

İsim	Açıklama	Örnek
DB, DW, DD, DQ, DT	Konumları başlatma	L6 DD 1A92H ;L6'daki çift sözcük 1A92H olarak başlatıldı
RESB, RESW, RESD, RESQ, REST	Başlatılmamış konumları rezerve edin	BUFFER RESB 64 ;BUFFER'dan başlayarak 64 bayt ayırm
INCBIN	İkili çıktıya dahil et	INCBIN "file.dat" ; bu dosyayı dahil et
EQU	Bir simbolü belirli bir sabit değere tanımlama	MSGLEN EQU 25 ;MSGLEN sabiti ondalık 25'e eşittir
TIMES	Talimatı birden çok kez tekrarlayın	ZEROBUF ÇARPI 64 DB 0 ;64 baylıklı tamponu tümüyle sıfır başlatır

```
L2 DB      "A"           ;bayt A için ASCII koduna başlatıldı (65 ( 
    MOV AL, [L1]       ;L1'deki baytı AL'ye kopyala
    MOV EAX, L1         ;L1'deki baytin adresini EAX'de saklar
    MOV [L1], AH        ;AH'nin içeriğini L1'deki bayta kopyalayın
```

Düz bir etiket kullanılırsa, verinin adresi (veya ofseti) olarak yorumlanır. Etiket köşeli parantezler içine yerleştirilirse, adresteki veri olarak yorumlanır.

**MAKRO TANIMLARI** Bir makro tanımı çeşitli yönlerden bir alt rutine benzer. Bir alt rutin, bir kez yazılan ve programın herhangi bir noktasından alt rutin çağrılarak birden çok kez kullanılabilen bir bölümündür. Bir program derlendiğinde veya birleştirildiğinde, alt rutin yalnızca bir kez yüklenir. Alt rutine yapılan bir çağrı kontrolü alt rutine aktarır ve alt rutindeki bir geri dönüş talimatı kontrolü yapıldığı noktaya geri döndürür. Benzer şekilde, bir makro tanımı, programının bir kez yazıldığı ve daha sonra birçok kullanabileceğii bir kod bölümündür. Temel fark, assembler'in bir makro çağrısıyla karşılaşlığında, makro çağrısını makronun kendisiyle değiştirmesidir. Bu işleme **makro genişletme** adı verilir. Yani, eğer bir makro bir

## 782 EK B / TOPLANTI ALANI VE İLGİLİ KONULAR

ve 10 kez çağrırlırsa, makronun 10 örneği birleştirilmiş kodda görünecektir. Özünde, alt rutinler çalışma zamanında donanım tarafından ele alınırken, makrolar montaj montajcı tarafından ele alınır. Makrolar modüler programlama açısından alt rutinlerle aynı avantajı sağlar, ancak bir alt rutin çağrısının ve geri dönüşünün çalışma zamanı ek yükü yoktur. Bunun karşılığında makro yaklaşımı nesne kodunda daha fazla alan kullanır.

NASM ve diğer birçok assembler'da, tek satırlı makro ile çok satırlı makro arasında bir ayırım yapılır. NASM'de, tek satırlı makrolar %DEFINE yönergesi kullanılarak tanımlanır. Burada birden fazla tek satırlı makronun genişletildiği bir örnek verilmiştir. İlk olarak, iki makro tanımlıyoruz:

```
DEFINE B(X= 2*X  
TANIM A(X= 1+ B(X)
```

Assembly dili programının bir noktasında aşağıdaki ifade görünür:

```
MOV AX, A(8)
```

Birleştirici bu ifadeyi şu şekilde genişletir:

```
MOV AX, 1+2*8
```

Bu da 17 anlık değerini AX kaydına taşımak bir makine komutuna bağlanır.

Çok satırlı makrolar %MACRO kısaltması kullanılarak tanımlanır. Aşağıda çok satırlı makro tanımına bir örnek verilmiştir:

```
MAKRO PROLOG 1  
PUSH EBP ;EBP'nin içeriğini yığına iter  
;ESP tarafından işaret edilen ve  
;ESP'nin içeriğini 4 azalt MOV EBP, ESP  
;ESP'nin içeriğini EBP'ye kopyala  
SUB ESP, %1 ;ESP'den ilk parametre değerini çıkarır
```

MACRO satırında makro adından sonra gelen 1 sayısı, makronun almayı beklediği parametre sayısını tanımlar. Makro tanımı içinde %1 kullanımı, makro çağrısının ilk parametresini ifade eder.

Makro çağrısı

```
MYFUNC: PROLOG 12
```

aşağıdaki kod satırlarına genişler:

```
MYFUNC: PUSH EBP  
MOV EBP, ESP  
SUB ESP, 12
```

### Örnek: En Büyük Ortak Bölten Programı

Assembly dilinin kullanımına bir örnek olarak, iki tamsayının en büyük ortak bölenini hesaplayan bir programa bakalım.  $a$  ve  $b$  tamsayılarının en büyük ortak bölenini aşağıdaki gibi tanımlız:

$$\text{gcd}(a, b) = \max[k, \text{öyle ki } k, a'yi böler \text{ ve } k, b'yi böler]$$

Burada, eğer kalan yoksa  $k$ 'nın  $a$ 'yı böldüğünü söyleyiz. Öklid'in en büyük ortak bölen algoritması aşağıdaki teoreme dayanır. Negatif olmayan herhangi bir  $a$  ve  $b$  tamsayısı için,

$$\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$$

İşte Öklid'in algoritmasını uygulayan bir C dili programı:

```
unsigned int gcd (unsigned int a, unsigned int b)
{
    if (a== 0 && b== 0( b =
        1;
    else if (b== 0( b =
        a;
    else if (a != 0(
        while (a != b(
            if (a< b(
                b -= a;
            başka
            a -= b;
    b'ye dön;
}
```

Şekil B.3 önceki programın iki assembly dili versiyonunu göstermektedir. Soldaki program bir C derleyicisi tarafından yapılmıştır; sağdaki program ise elle programlanmıştır. İlkinci program daha sıkı, daha verimli bir uygulama üretmek için bir dizi programcı hilesi kullanmaktadır.

## B.2 ASSEMBLERs

**Montajçı**, bir montaj programını girdi olarak alan ve çıktı olarak nesne kodu üreten bir yazılım yardımcı programıdır. Nesne kodu ikili bir dosyadır. Montajçı bu dosyayı görelî konum 0'dan başlayan bir bellek bloğu olarak görür.

Birleştiriciler için iki genel yaklaşım vardır: iki geçişli birleştirici ve tek geçişli birleştirici.

### İki Geçişli Birleştirici

İlk olarak daha yaygın olan ve anlaşılması biraz daha kolay olan iki geçişli assembler'a bakacağız. Birleştirici kaynak kod üzerinden iki geçiş yapar (Şekil B.4):

**İLK** GEÇİŞ İlk geçişte, assembler yalnızca etiket tanımlarıyla ilgilenir. İlk geçiş, tüm etiketlerin ve ilişkili **konum sayacı** (LC) değerlerinin bir listesini içeren bir **sembol tablosu** oluşturmak için kullanılır. Nesne kodunun ilk bayti 0 LC değerine sahip olacaktır. İlk geçiş her bir assembly ifadesini inceler. Assembler henüz talimatları çevirmeye hazır olmasa da

## 784 EK B / TOPLANTI ALANI VE İLGİLİ KONULAR

<pre> gcd:          mov     ebx,eax               mov     eax,edx               test    ebx,ebx               jne    L1               test    edx,edx               jne    L1               mov     eax,1               ret L1:          test    eax,eax               jne    L2               mov     eax,ebx               ret L2:          test    ebx,ebx               je     L5 L3:          cmp     ebx,eax               je     L5               Jae    L4               alt    eax,ebx               jmp    L3 L4:          alt    ebx,eax               jmp    L3 L5:          ret </pre>	<pre> gcd:          neg    eax               je     L3 L1:          neg    eax               xchg  eax,edx L2:          alt    eax,edx               jg    L2               jne    L1               ekle  eax,edx L3:          jne    L4               inc    eax               ret L4:          ret </pre>
---	---

(a) Derlenmiş program

(b) Doğrudan assembly dilinde yazılmış

**Şekil B.3** En Büyük Ortak Bölüm için Assembly Programları

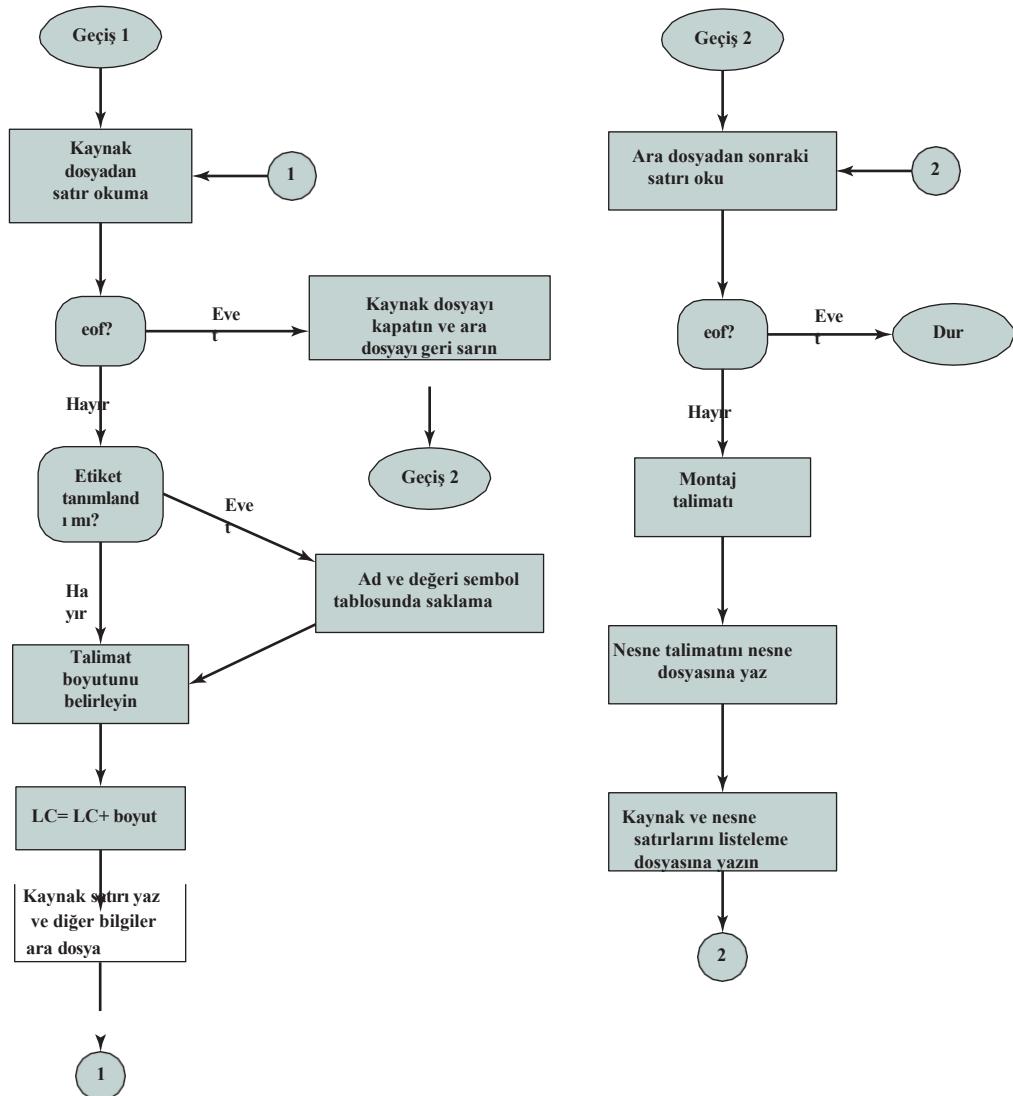
her bir komut, karşılık gelen makine komutunun uzunluğunu ve dolayısıyla LC'nin ne kadar artırılacağını belirlemek için yeterlidir. Bu, yalnızca işlem kodunun incelenmesini değil, aynı zamanda işlenenlere ve adresleme modlarına da bakılmasını gerektirebilir.

DQ ve REST gibi yönergeler (bkz. Tablo B.2) konum sayacının ne kadar depolama alanı belirtildiğine göre ayarlanması neden olur.

Assemblerler etiketli bir deyimle karşılaşlığında, etiketi geçerli LC değeriyle birlikte sembol tablosuna yerleştirir. Assemblerler tüm assembly dili deyimlerini okuyana kadar devam eder.

**İKİNCİ GEÇİŞ** İkinci geçiş programı baştan tekrar okur. Her komut uygun ikili makine koduna çevrilir. Çeviri aşağıdaki işlemleri içerir:

1. Anımsatıcıyı ikili bir işlem koduna çevirin.
2. Komutun biçimini ve komuttaki çeşitli alanların konumunu ve uzunluğunu belirlemek için işlem kodunu kullanın.
3. Her bir operand adını uygun register veya bellek koduna çevirin.
4. Her anlık değeri bir ikili dizeye çevirin.
5. Etiketlere yapılan referansları sembol tablosunu kullanarak uygun LC değerine çevirin.
6. Adresleme modu göstergeleri, durum kodu bitleri vb. dahil olmak üzere komutta gereklili olan diğer bitleri ayarlayın.



**Şekil B.4** İki Geçişli Assembler Akış Şeması

ARM assembly dilini kullanan basit bir örnek Şekil B.5'te gösterilmektedir. ARM assembly dili talimatı ADDS r3, r3, #19, ikili makine talimatı 1110 0010 0101 0011 0011 0000 0001 0011'e çevrilir.

**ZEROTH PASS** Çoğu assembly dili makro tanımlama yeteneğini içerir. Makrolar mevcut olduğunda, assembler'in ilk geçişten önce yapması gereken ek bir vardır. Tipik olarak, assembly dili tüm makro tanımlarının programın başında yer olmasını gerektirir.

## 786 EK B / TOPLANTI ALANI VE İLGİLİ KONULAR

	Her zaman koşul kodu	Koşul bayraklarını güncelleme	Sıfır dönüş
ADDS r3, r3, #19	1 1 1 0 0 0 1 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 1		
Veri işleme anında format	cond instrumaln Biçim 2 21 20 19 18 17 16	opcode S Rn Rd döndürmek 8 7 6 5	hemen 4 3 2 1 0

**Sekil B.5** Bir ARM Assembly Komutunun İkili Makine Komutuna Çevrilmesi

Birleştirici bu "sıfırıncı geçiş" tüm makro tanımlarını okuyarak başlar. Tüm makrolar tanındıktan sonra, asembler kaynak kodu gözden geçirir ve bir makro çağrı ile karşılaşıldığında makroları ilişkili parametreleri ile genişletir. Makro işleme geçisi, kaynak kodun tüm makro genişletmelerinin yerinde olduğu ve tüm makro tanımlarının kaldırıldığı yeni bir sürümünü oluşturur.

### Tek Geçişli Birleştirici

Kaynak kod üzerinden sadece tek bir geçiş yapan bir asembler uygulamak mümkündür (makro işleme saymazsak). Bir programı tek geçişte bir araya getirmeye çalışmanın ana zorluğu etiketlere yapılan ileri referanslarla ilgilidir. Komut operandları kaynak programda henüz tanımlanmamış semboller olabilir. Bu nedenle, asembler aktarılan komuta hangi görelî adresi ekleyeceğini bilemez.

Temelde, ileri referansları çözümleme süreci aşağıdaki gibi çalışır. Birleştirici, henüz tanımlanmamış bir sembol olan bir komut işleneniyle karşılaşlığında, birleştirici aşağıdakileri yapar:

1. Birleştirilmiş binary komutunda komut operand alanını boş (tümü sıfır) bırakır.
2. İşlenen olarak kullanılan sembol, sembol tablosuna girilir. Tablo girişi, sembolün tanımsız olduğunu belirtmek için işaretlenir.
3. Tanımlanmamış sembole atıfta bulunan komuttaki işlenen alanının adresi, sembol tablosu girişiyle ilişkili ileri referanslar listesine eklenir.

Bir LC değeri ile ilişkilendirilebilecek şekilde sembol tanımiyla karşılaşlığında, asembler LC değerini sembol tablosundaki uygun girişe ekler. Sembolle ilişkili bir ileri referans listesi varsa, asembler uygun adresi ileri referans listesinde bulunan daha önce üretilmiş herhangi bir komuta ekler.

### Örnek: Asal Sayı Programı

Şimdi yönergeler içeren bir örneğe bakacağız. Bu örnek asal sayıları bulan bir programa baktmaktadır. Asal sayıların sadece 1'e ve kendilerine eşit olarak bölünebildiğini hatırlayın. Bunu yapmak için bir formül yoktur. Bu programın kullandığı temel yöntem, belirli bir limitin altındaki tüm tek sayıların çarpanlarını bulmaktır. Eğer hiçbir çarpan bulunamazsa

```

işaretsiz tahmin;
işaretsiz faktör;
işaretsiz limit;

printf ("kadar olan asal sayıları
bulun: "; scanf("%u", &limit);
printf ("2\n";
al
printf ("3\n";
tahmin= 5;
while (tahmin < = limit( {
    faktör = 3;
    while (faktör * faktör< tahmin && tahmin% faktör != 0( faktör + =
    2;
    if (tahmin % faktör != 0( printf
        ("%d\n", tahmin);
    tahmin et+= 2;
                                /* sadece tek sayılara bak */
}

```

**Şekil B.6** Asallığı Test Etmek için C Programı

tek bir sayı için bulunursa, asaldır. Şekil B.6'da aşağıdaki dilde yazılmış temel algoritma gösterilmektedir C. Şekil B.7 NASM assembly dilinde yazılmış aynı algoritmayı göstermektedir.

## B.3 YÜKLEME VE BAĞLAMA

Aktif bir sürecin oluşturulmasındaki ilk adım, ana belleğe bir program yüklemek ve bir süreç görüntüsü oluşturmaktır (Şekil B.8). Şekil B.9 çoğu sistem için tipik bir senaryoyu göstermektedir. Uygulama, nesne kodu biçiminde bir dizi derlenmiş veya birleştirilmiş modülden oluşur. Bunlar, modüller arasındaki referansları çözmek için birbirine bağlanır. Aynı zamanda, kütüphane rutinlerine yapılan referanslar da çözümlenir. Kütüphane rutinlerinin kendileri programa dahil edilebilir veya çalışma zamanında işletim sistemi tarafından sağlanması gereken paylaşılan kod olarak referans verilebilir. Bu bölümde, bağlayıcıların ve yükleyicilerin temel özelliklerini özetleyeceğiz. İlk olarak, yer değiştirme kavramını tartışıyoruz. Ardından, sunumda netlik sağlamak için, tek bir program modülü söz konusu olduğunda yükleme görevini açıklıyoruz; bağlama gerekmez. Daha sonra bağlama ve yükleme işlevlerine bir bütün olarak bakabiliriz.

### Yer Değiştirme

Çok programlı bir sisteme, mevcut ana bellek genellikle bir dizi işlem arasında paylaşılır. Tipik olarak, programcının kendi programının yürütülmesi sırasında ana bellekte hangi diğer programların bulunacağını önceden bilmesi mümkün değildir. Buna ek olarak, çalıştırılmaya hazır geniş bir süreç havuzu sağlayarak işlemci kullanımını en üst düzeye çıkarmak için aktif süreçler ana belleğe takas edebilmek ve ana bellekten çıkarabilmek isteriz. Bir program diske takas edildikten sonra, bir sonraki takas işleminde daha önce olduğu gibi aynı ana bellek bölgесine yerleştirilmesi gerektiğini beyan etmek oldukça sınırlayıcı olacaktır. Bunun yerine, süreci belleğin farklı bir alanına **yerlestirmemiz** gerekebilir.

## 788 EK B / TOPLANTI ALANI VE İLGİLİ KONULAR

```
"asm_io.inc" segmentini
%include .data
Mesaj db "fazla asal sayıları bul: ", 0

segment .bss
Limit resd 1
Tahmin et resd 1

segment .text
    global _asm_main
_asm_main:
    0,0 girin
    pusha

    mov eax, Mesaj call
    print_string
    read_int'i çağırın
    mov [Limit], eax
    mov eax, 2
    print_int çağrısı
    print_nl'yi çağırın
    mov eax, 3
    print_int çağrısı
    print_nl'yi çağırın

    mov dword [Tahmin], 5
    mov eax, [Tahmin] cmp
    eax, [Sınır]
    jnbe end_while_limit

    mov ebx, 3
while_factor:
    mov eax, ebx
    mul eax
    jo end_while_factor
    eax, [Tahmin]
    jnb end_while_factor
    eax,[Tahmin]
    mov edx, 0
    div ebx
    cmp edx, 0
    je end_while_factor

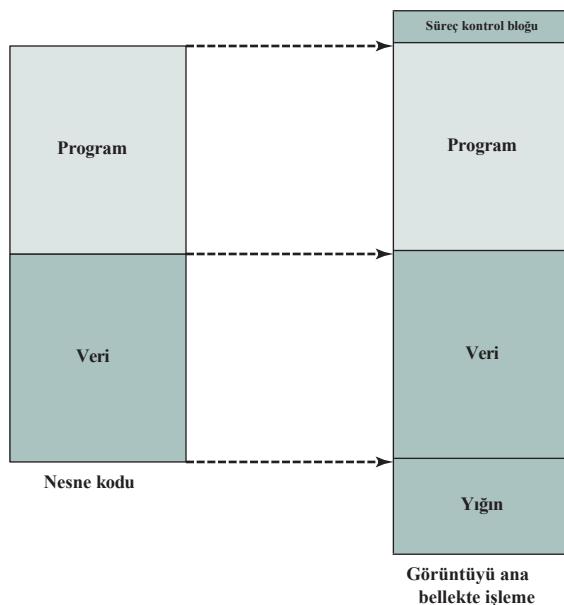
    ebx,2 ekle; faktör+= 2;
    jmp while_factor
end_while_factor:
    je end_if
    mov eax,[Guess]
    call print_int
    print_nl end_if'i
çağırın:
    add dword [Guess], 2
    jmp while_limit
end_while_limit:

    popa
    mov eax, 0
    ret

    Bu limite kadar olan asal sayıları bulmak
    ; asal için mevcut tahmin
    kurulum rutini

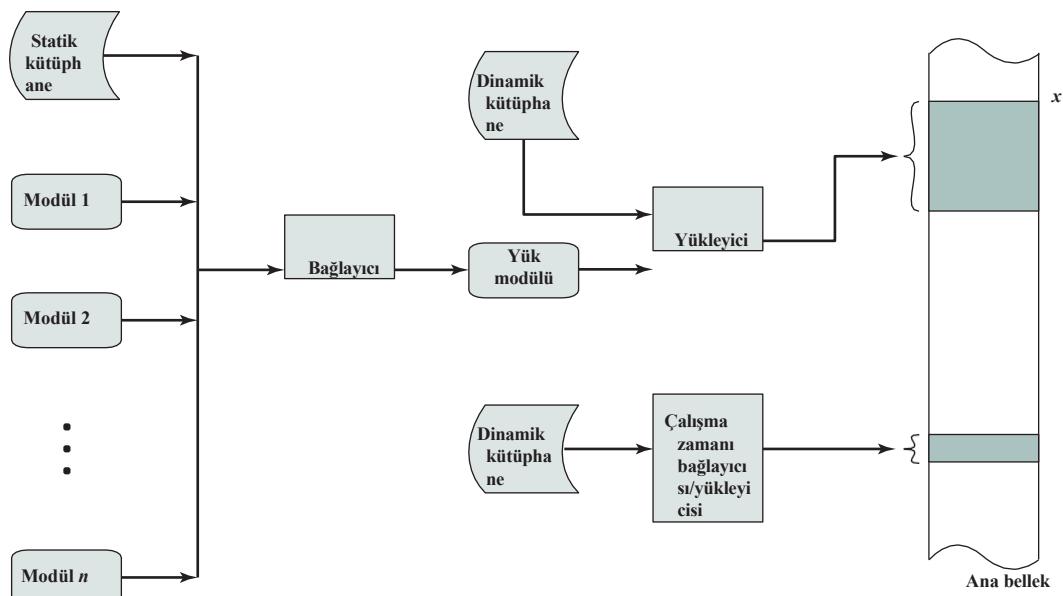
    ; scanf("%u", & limit());
    ; printf("2\n");
    ; printf("3\n");
    ; Tahmin = 5; while_limit:
    ; while (Tahmin<= Limit(
    sayılar işaretsiz olduğu için jnbe kullanın
    ; ebx faktördür= 3;
    ; edx:eax= eax*eax
    ; yanıt yalnızca eax'a sızmazsa cmp
    ; if !(faktör*faktör< tahmin( mov
    ; edx= edx:eax% ebx
    ; eğer !(tahmin% faktörü != 0(
    ; eğer !(tahmin% faktörü != 0(
    ; printf("%u\n"
    C iznine geri dönün
```

**Sekil B.7** Asallığı Test Etmek için Montaj Programı

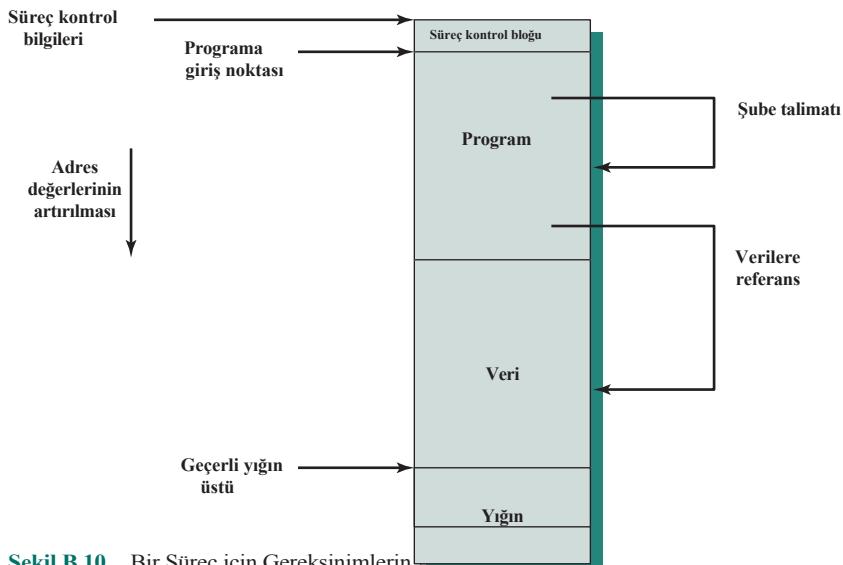


Şekil B.8 Yükleme Fonksiyonu

Bu nedenle, bir programın nereye yerleştirileceğini önceden bilemeyeiz ve takas nedeniyle programın ana bellekte yer değiştirmesine izin vermemiyiz. Bu gerçekler, Şekil B.10'da gösterildiği gibi adresleme ile ilgili bazı teknik endişeleri ortaya çıkarmaktadır. Şekilde bir süreç görüntüsü gösterilmektedir. Basitlik açısından, işlem görüntüsünün ana belleğin bitişik bir bölgesini kapladığını varsayalım. Açıkça görüldüğü üzere



Şekil B.9 Bir Bağlama ve Yükleme Senaryosu



Şekil B.10 Bir Süreç için Gereksinimler

İşletim sistemi, süreç kontrol bilgilerinin ve yürütme yığınının konumunun yanı sıra bu süreç için programın yürütülmeye başlanacağı giriş noktasını da bilmelidir. İşletim sistemi belleği yönettiğinden ve bu süreci ana belleğe getirmekten sorumlu olduğundan, bu adreslere ulaşmak kolaydır. Ancak buna ek olarak, işlemci program içindeki bellek referanslarıyla da ilgilenmek zorundadır. Dallanma talimatları, bir sonraki yürütülecek talimata referans veren bir adres içerir. Veri referans talimatları, referans verilen verinin bayt ya da sözcüğünün adresini içerir. Bir şekilde, işlemci donanımı ve işletim sistemi yazılımı, program kodunda bulunan bellek referanslarını, programın ana bellekteki mevcut konumunu yansıtan gerçek fiziksel bellek adreslerine çevirebilmelidir.

### Yükleniyor

Şekil B.9'da yükleyici, yük modülünü ana belleğe konumdan başlayarak yerleştirir. Program yüklenirken, Şekil B.10'da gösterilen adresleme gereksinimi karşılanmalıdır. Genel olarak, üç yaklaşım benimsenebilir:

- Mutlak yükleme
- Yer değiştirilebilir yükleme
- Dinamik çalışma zamanı yüklemesi

**MUTLAK YÜKLEME** Mutlak yükleyici, belirli bir yükleme modülünün ana bellekte her zaman aynı konuma yüklenmesini gerektirir. Bu nedenle, yükleyiciye sunulan yükleme modülünde, tüm adres referansları belirli veya mutlak ana

bellek adresleri. Örneğin, Şekil B.9'daki  $x$  konumu 1024 ise, belleğin o bölgesine yönelik bir yükleme modülündeki ilk sözcük 1024 adresine sahiptir.

Belirli adres değerlerinin program içindeki bellek referanslarına atanması programcı tarafından ya da derleme veya birleştirme zamanında yapılabilir (Tablo B.3a). İlk yaklaşımın birkaç dezavantajı vardır. Birincisi, her programının modülleri ana bellege yerlestirmek için amaçlanan atama stratejisini bilmesi gerekecektir. İkincisi, eğer programda modülün gövdesinde ekleme ya da çıkarma içeren herhangi bir değişiklik yapılrsa, tüm adreslerin değiştirilmesi gerekecektir. Buna göre, programlar içindeki bellek referanslarının sembolik olarak ifade edilmesine izin vermek ve daha sonra derleme veya montaj sırasında bu sembolik referansları çözmek tercih edilir. Bu Şekil B.11'de gösterilmiştir. Bir komuta veya veri öğesine yapılan her referans başlangıçta bir sembolle temsil edilir. Modülü mutlak yükleyiciye giriş için hazırlarken, birleştirici veya derleyici bu referansların tümünü Şekil B.11b'de gösterildiği gibi belirli adreslere (bu örnekte, 1024 konumundan başlayarak yüklenecek bir modül için) dönüştürecekтир.

**Tablo B.3** Adres Bağlama

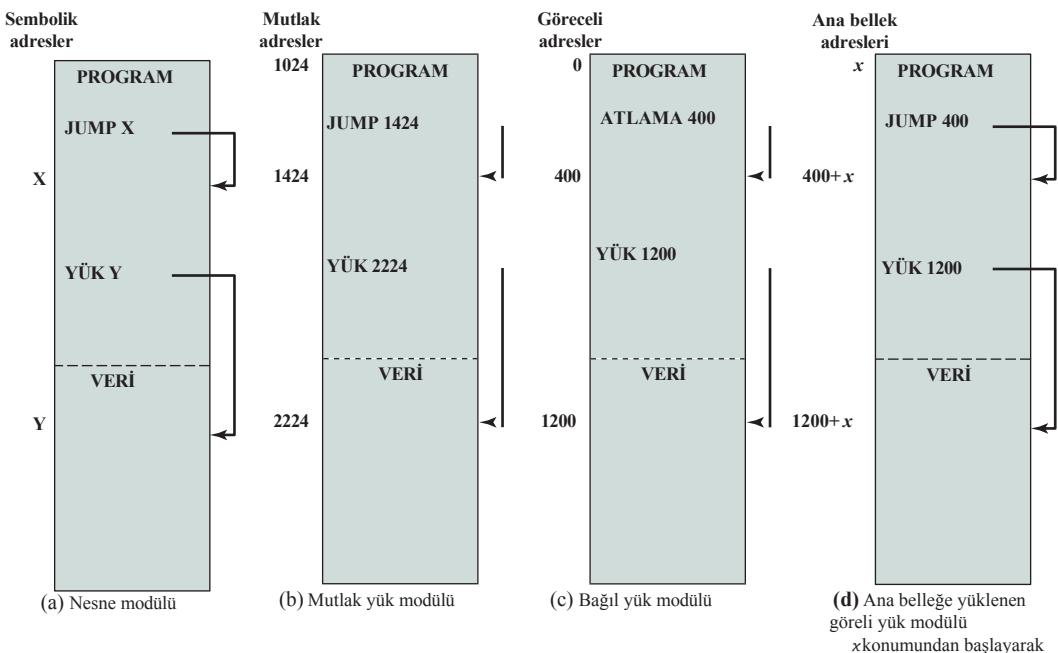
(a) Yükleyici

Bağlama Süresi	Fonksiyon
Programlama süresi	Tüm gerçek fiziksel adresler doğrudan programcı tarafından programın kendisinde belirtilir.
Derleme veya montaj zamanı	Program sembolik adres referansları içerir ve bunlar derleyici veya birleştirici tarafından gerçek fiziksel adreslere dönüştürülür.
Yükleme süresi	Derleyici ya da birleştirici görelİ adresler üretir. Yükleyici, program yükleme sırasında bunları mutlak adreslere çevirir.
Çalışma süresi	Yüklenen program görelİ adresleri korur. Bunlar işlemci donanımı tarafından dinamik olarak mutlak adreslere dönüştürülür.

(b) Bağlayıcı

Bağlantı Süresi	Fonksiyon
Programlama süresi	Harici program veya veri referanslarına izin verilmez. Programcı, referans verilen tüm alt programların kaynak kodunu programa yerleştirmelidir.
Derleme veya montaj zamanı	Birleştirici, başvurulan her alt rutinin kaynak kodunu almalı ve bunları bir birim olarak birleştirmelidir.
Yük modülü oluşturma	Tüm nesne modülleri görelİ adresler kullanılarak bir araya getirilmiştir. Bu modüller birbirine bağlanır ve tüm referanslar son yükleme modülünün göre yeniden düzenlenir.
Yükleme süresi	Dış referanslar, yükleme modülü ana belleğe kadar çözümlenmez. O zaman, referans verilen dinamik bağlantı modülleri yükleme modülüne eklenir ve tüm paket ana veya sanal belleğe yüklenir.
Çalışma süresi	Harici referanslar, harici çağrı işlemci tarafından yürütülene kadar çözümlenmez. O zaman, işlem ve istenen modül çağrıran programa bağlanır.

## 792 EK B / TOPLANTI ALANI VE İLGİLİ KONULAR



**Şekil B.11** Mutlak ve Yer Değiştirebilir Yük Modülleri

**İLİŞKİLENDİRİLEBİLİR YÜKLEME** Yükleme öncesinde bellek referanslarını belirli adreslere bağlamadan dezavantajı, ortaya çıkan yükleme modülünün ana belleğin yalnızca bir bölgesine yerleştirilebilmesidir. Bununla birlikte, birçok program ana belleği paylaştığında, belirli bir modülün belleğin hangi bölgesine yükleneceğine önceden karar vermek istenmeyebilir. Bu kararı yükleme zamanında vermek daha iyidir. Bu nedenle, ana bellekte herhangi bir yere yerleştirilebilecek bir yükleme modülünde ihtiyacımız vardır.

Bu yeni gereksinimi karşılamak için, birleştirici veya derleyici gerçek ana bellek adreslerini (mutlak adresler) değil, programın başlangıcı gibi bilinen bir noktaya göre olan adresleri üretir. Bu teknik Şekil B.11'de gösterilmiştir. Yükleme modülünün başlangıcına göreceli adres 0 atanır ve modül içindeki diğer tüm bellek referansları modülün başlangıcına görelilik olarak ifade edilir.

Tüm bellek referansları göreceli biçimde ifade edildiğinde, yükleyicinin modülü istenen konuma yerleştirmesi basit bir görev haline gelir. Modül x konumundan başlayarak yüklenenekse, yükleyici modülü belleğe yüklerken her bellek referansına x eklemelidir. Bu görevde yardımcı olmak için yükleme modülü, yükleyiciye adres referanslarının nerede olduğunu ve nasıl yorumlanacağını (genellikle programın başlangıcına göre, ancak muhtemelen programdaki mevcut konum gibi başka bir noktaya göre) söyleyen bilgileri içermelidir. Bu bilgi kümlesi derleyici veya birleştirici tarafından hazırlanır ve genellikle yer değiştirme sözlüğü olarak adlandırılır.

**DİNAMİK ÇALIŞMA ZAMANI YÜKLEME** Yer değiştirebilen yükleyiciler yaygındır ve mutlak göre bariz avantajlar sağlar. Ancak, çoklu programlamada

ortamında, hatta sanal bellege bağlı olmayan bir ortamda bile, yeniden konumlandırılabilir yükleme şeması yetersizdir. İşlemci kullanımını en üst düzeye çıkarmak için işlem görüntülerini ana bellege takas etme ve ana bellekten çıkışma ihtiyacından bahsetmiştik. Ana bellek kullanımını en üst düzeye çıkarmak için, işlem görüntüsünü farklı zamanlarda farklı konumlara geri takas edebilmek isteriz. Böylece, bir program yüklenildikten sonra diske takas edilebilir ve daha sonra farklı bir konuma geri takas edilebilir. Eğer bellek referansları ilk yükleme zamanında mutlak adreslere bağlı olsaydı bu imkansız olurdu.

Bunun alternatifisi, mutlak adresin hesaplanması çalışma zamanında gerçekten ihtiyaç duyulana kadar ertelemektir. Bu amaçla, yükleme modülü ana bellege tüm bellek referansları görelî biçimde yüklenir (Şekil B.11c). Bir komut gerçekten çalıştırılana kadar mutlak adres hesaplanmaz. Bu işlevin performansı düşürmemesini sağlamak için, yazılım yerine özel işlem donanımı tarafından yapılmalıdır. Bu donanım Bölüm 8'de açıklanmıştır.

Dinamik adres hesaplaması tam esneklik sağlar. Bir program ana belleğin herhangi bir bölgesine yüklenebilir. Daha sonra, programın yürütülmesi kesilebilir ve program daha sonra farklı bir konumda tekrar takas edilmek üzere ana bellekten çıkarılabilir.

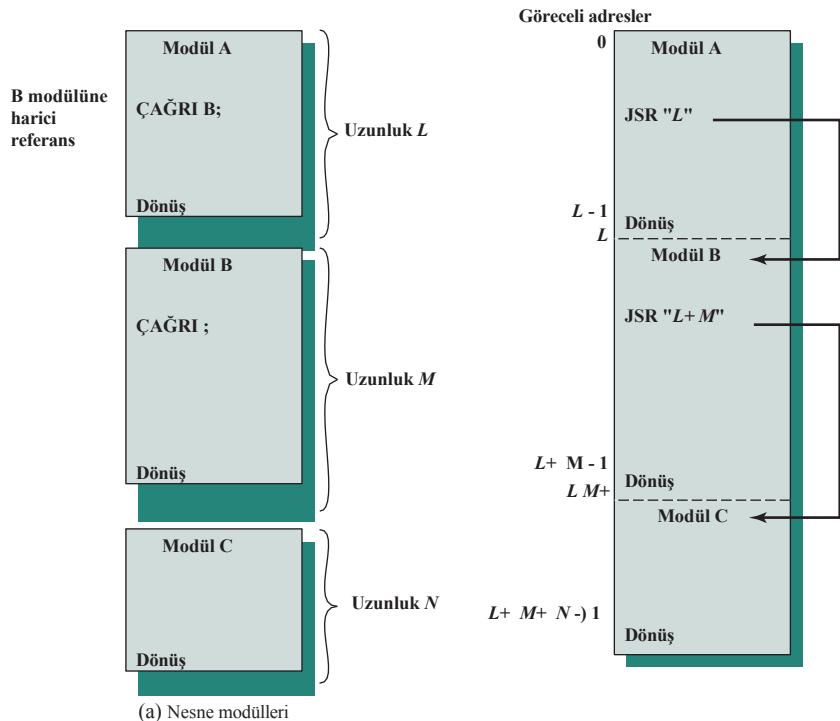
## Bağlantı

Bir bağlayıcıının işlevi, girdi olarak bir nesne modülleri koleksiyonu almak ve yükleyiciye aktarılmak üzere entegre bir program ve veri modülleri kümelerinden oluşan bir yükleme modülü oluşturmaktır. Her nesne modülünde, diğer modüllerdeki konumlara adres referansları olabilir. Bu tür referansların her biri yalnızca bağlantısız bir nesne modülünde sembolik olarak ifade edilebilir. Bağlayıcı, tüm nesne modüllerinin bitişik birleşimi olan tek bir yükleme modülü oluşturur. Her modül içi referans sembolik bir adresden genel yükleme modülü içindeki bir konuma referansa değiştirilmelidir. Örneğin, Şekil B.12a'daki A modülü, B modülünün bir prosedür çağrısını içerir. Bu modüller yükleme modülünde birleştirildiğinde, B modülüne yapılan bu sembolik referans, yükleme modülü içinde B'nin giriş noktasının konumuna özel bir referans olarak değiştirilir.

**BAĞLANTI DÜZENLEYİCİ** Bu adres bağlantısının niteliği, oluşturulacak yük modülünün türüne ve bağlantının ne zaman gerçekleşeceğini bağlı olacaktır (Tablo B.3b). Genellikle gibi, yeniden konumlandırılabilir bir yük modülü isteniyorsa, bağlantı genellikle aşağıdaki şekilde yapılır. Derlenen veya birleştirilen her nesne modülü, nesne modülünün başlangıcına göre referanslarla oluşturulur. Tüm bu modüller, tüm yükleme modülünün başlangıcına göre olan tek bir yeniden konumlandırılabilir yükleme modülünde bir araya getirilir. Bu modül, yeniden konumlandırılabilir yükleme veya dinamik çalışma zamanı yüklemesi için girdi olarak kullanılabilir.

Yeniden konumlandırılabilir bir yük modülü üreten bir bağlayıcı genellikle bir yaşı düzenleyicisi olarak adlandırılır. Şekil B.12 bağlantı düzenleyici işlevini göstermektedir.

**DİNAMİK BAĞLANTI** Yüklemeye olduğu gibi, bazı bağlantı işlevlerini ertelemek mümkündür. *Dinamik bağlantı* terimi, bazı harici modüllerin yükleme modülü oluşturulmasına kadar ertelenmesi uygulamasına atıfta bulunmak için kullanılır. Böylece, yükleme modülü diğer programlara çözülmemiş referanslar içerir. Bu referanslar yükleme zamanında ya da çalışma zamanında çözülebilir.



Şekil B.12 Bağlama İşlevi

**Yükleme zamanı dinamik bağlama** için (Şekil B.9'daki üst dinamik kütüphaneyi içeren), aşağıdaki adımlar gerçekleşir. Yüklenen olana yük modülü (uygulama modülü) belleğe okunur. Harici bir module (hedef modül) yapılan herhangi bir referans, yükleyicinin hedef modülü bulmasına, yüklemesine ve referansı uygulama modülünün başlangıcından itibaren bellekteki göreli bir adrese değiştirmesine neden olur. Bu yaklaşımın statik bağlama olarak adlandırılabilcecak yaklaşıma göre birkaç avantajı vardır:

- Bir işletim sistemi yardımcı programı veya başka bir genel amaçlı rutin olabilecek hedef modülün değiştirilmiş veya yükseltilmiş sürümlerini dahil etmek daha kolay hale gelir. Statik bağlama ile, böyle bir destekleyici modülde yapılacak bir değişiklik, tüm uygulama modülünün yeniden bağlanması gerektirecektir. Bu sadece verimsiz olmakla kalmaz, aynı zamanda bazı durumlarda imkansız da olabilir. Örneğin, kişisel bilgisayar alanında çoğu ticari yazılım yük modülü şeklinde yayınlanır, kaynak ve nesne sürümleri yayınlanmaz.
- Dinamik bağlantı dosyasında hedef kodun bulunması otomatik kod paylaşımının önünü açar. İşletim sistemi birden fazla uygulamanın aynı hedef kodu kullandığını anlayabilir çünkü bu kodu yüklemiş ve bağlamıştır. Bu bilgiyi, her uygulama için bir kopya yüklemek yerine, hedef kodun tek bir kopyasını yüklemek ve her iki uygulamaya da bağlamak için kullanabilir.

- Bağımsız yazılım geliştiricilerin Linux gibi yaygın olarak kullanılan bir işletim sisteminin işlevsellliğini genişletmesi daha kolay hale gelmektedir. Bir geliştirici, çeşitli uygulamalar için yararlı olabilecek yeni bir işlev bulabilir ve bunu bir dinamik bağlantı modülü olarak paketleyebilir.

**Çalışma zamanı dinamik bağlama** ile (Şekil B.9'daki alt dinamik kütüphaneyi içeren), bağlama işleminin bir kısmı yürütme zamanına kadar ertelenir. Hedef modüllere yapılan harici referanslar yüklenen programda kalır. Olmayan modüle bir çağrı yapıldığında, işletim sistemi bulur, yükler ve çağrıran modüle bağlar. Bu tür modüller tipik olarak paylaşılabilir. Windows ortamında bunlara dinamik bağlantı kütüphaneleri (DLL'ler) denir. Bu nedenle, bir süreç dinamik olarak bağlanmış paylaşılan bir modülü zaten kullanıyorsa, bu modül ana bellektedir ve yeni bir süreç zaten yüklenmiş olan modüle kolayca bağlanabilir.

DLL'lerin kullanımı genellikle **DLL cehennemi** olarak adlandırılan bir soruna yol açabilir. İki veya daha fazla işlem bir DLL modülünü paylaşıyor ancak modülün farklı sürümlerini bekliyorsa DLL cehennemi oluşur. Örneğin, bir uygulama veya sistem işlevi yeniden yüklenebilir ve beraberinde bir DLL dosyasının eski bir sürümünü getirebilir.

Dinamik yüklemenin tüm bir yükleme modülünün yer değiştirmesine izin verdiğiini gördük; ancak modülün yapısı statiktir, işlemin yürütülmesi boyunca ve bir yürütmeden diğerine değişmez. Ancak bazı durumlarda, hangi nesne modüllerinin gerekli olacağını yürütmeden önce belirlemek mümkün değildir. Bu durum, havayolu rezervasyon sistemi veya bankacılık uygulaması gibi işlem-işlem uygulamaları ile tipikleştirilir. İşlemin niteliği hangi program modüllerinin gerekli olduğunu belirler ve bunlar uygun şekilde yüklenir ve ana programa bağlanır. Böyle bir dinamik bağlayıcının kullanılmasının avantajı, bu birimlere başvurulmadığı sürece program birimleri için bellek ayırmayan gereklilik olmamasıdır. Bu özellik segmentasyon sistemlerini desteklemek için kullanılır.

Bir ek iyileştirme daha mümkündür: Bir uygulamanın çağrılabilecek tüm modüllerin veya giriş noktalarının adlarını bilmesi gerekmek. Örneğin, bir grafik programı, her biri farklı bir sürücü paketi tarafından çalıştırılan çeşitli çizicilerle çalışmak üzere yazılmış olabilir. Uygulama, o anda sisteme yükü olan çizicisinin adını başka bir işleminden veya bir yapılandırma dosyasından öğrenebilir. Bu, uygulamanın kullanımısının, uygulamanın yazıldığı sırada mevcut olmayan yeni bir çizici kurmasına olanak tanır.

## B.4 ANAHTAR TERİMLER, SORULAR VE SORUNLAR

### Anahtar Terimler

Montajçı montaj dili	etiket bağlantı düzenleyici	animsatıcı tek geçişli birleştirici
Yorum	bağlama	operand
yönerge	yükleme zamanı dinamik bağlama	yer değiştirme
dinamik bağlayıcı	yükleme	çalışma zamanı dinamik bağlama
talimat	makro	iki geçişli birleştirici

## **İnceleme Soruları**

- B.1** Assembly dili programlama eğitimi almanın neden değerli olduğuna dair bazı nedenleri sıralayınız.
- B.2** Assembly dili nedir?
- B.3** Assembly dilinin yüksek seviyeli dillere kıyasla bazı dezavantajlarını listeleyiniz.
- B.4** Assembly dilinin yüksek seviyeli dillere kıyasla bazı avantajlarını listeleyiniz.
- B.5** Bir assembly dili ifadesinin tipik öğeleri nelerdir?
- B.6** Dört farklı assembly dili deyimini listeleyiniz ve kısaca tanımlayınız.
- B.7** Tek geçişli bir birleştirici ile iki geçişli bir birleştirici arasındaki fark nedir?

## **Problemler**

- B.1** Core War, 1980'lerin başında halka tanıtılan [DEWD84] ve 15 yıl kadar bir süre popüler olan bir programlama oyunudur. Core War'in dört ana bileşeni vardır: 8000 adresli bir bellek dizisi; basitleştirilmiş bir assembly dili Redcode; MARS adı verilen bir yürütme programı (Memory Array Red- code Simulator'un kısaltması); ve çekişen savaş programları kümesi. İki savaş programı rastgele seçilen konumlarda bellek dizisine girilir; programlardan hiçbiri diğerinin nerede olduğunu bilmem. MARS programları zaman paylaşımlının basit bir versiyonunda yürütür. İki program sırayla çalışır; ilk programın tek bir komutu yürütülür, ardından ikincisinin tek bir komutu yürütülür ve bu böyle devam eder. Bir savaş programının kendisine ayrılan yürütme döngüleri sırasında ne yapacağı tamamen programcıya bağlıdır. Amaç, diğer programın talimatlarını bozarak onu yok etmektir. Bu problemde ve sonraki birkaç problemde, bazı Core War kavramlarını keşfetmek için CodeBlue adı verilen daha basit bir dil kullanıyoruz.

CodeBlue yalnızca beş assembly dili deyimi içerir ve üç ad- dressing modu kullanır (Tablo B.4). Adresler etrafına sarılır, böylece bellekteki son konum için+ 1'in görelî adresi bellekteki ilk konuma atıfta bulunur. Örneğin, ADD #4, 6 görevci konum 6'nın içeriğine 4 ekler ve sonuçları konum 6'da saklar; JUMP @5 yürütmemeyi geçerli JUMP komutunun konumundan beş yuva sonraki konumda bulunan bellek adresine aktarır.

- a.** Imp programı tek bir COPY 0, 1 komutudur. Bu komut ne yapar?
- b.** Dwarf programı aşağıdaki talimatlar dizisidir:

```
ADD #4, 3  
KOZYALA 2, @2  
JUMP -2  
VERİ 0
```

Ne işe yariyor?

- c.** Dwarf'ı semboller kullanarak yeniden yazın, böylece tipik bir assembly language programına daha çok benzeyecektir.
- B.2** Imp ile Dwarf'ı karşı karşıya getirirsek ne olur?
- B.3** CodeBlue'da tüm belleği sıfırlayan bir "hali bombalama" programı yazın (program konumları hariç).
- B.4** Önceki program Imp'e karşı nasıl performans sergileyecək?
- B.5**
  - a.** Aşağıdaki diziden sonra C durum bayrağının değeri nedir?

```
mov al, 3  
al ekle, 4
```

- b.** Aşağıdaki diziden sonra C durum bayrağının değeri nedir?

```
mov al, 3  
alt al, 4
```

**Tablo B.4** CodeBlue Assembly Dili

## (a) Talimat Seti

Biçim	Anlamı
VERİ <değer>	<değer> geçerli konuma ayarlanır
KOPYA A, B	A kaynagini B hedefine kopyalar
ADD A, B	A'yı B'ye ekler, sonucu B'ye koyar
ZIPLA A	yürütmemeyi A'ya aktarın
JUMPZ A, B	B= 0 ise, A'ya aktarın

## (b) Adresleme Modları

Mod	Biçim	Anlamı
Gerçek	# ardından değer gelir	Bu bir anlık moddur, işlenen değeri komutun içindedir.
Göreceli	Değer	Değer, işleneni içeren geçerli konumdan bir ofseti temsil eder.
Dolaylı	@ ve ardından değer	Değer, geçerli konumdan bir ofseti temsil eder; ofset konumu, işleneni içeren konumun göreli adresini içerir.

Döngü KOPYA #0, -1  
JUMP -1

*İpucu:* Komut yürütmenin iki karşıt program arasında dönüşümlü olduğunu unutmayın.

**B.6** Aşağıdaki NAMS talimatını göz önünde bulundurun:

cmp vleft, vright

İşareti tamsayılar için ilgili üç durum bayrağı vardır. Eğer vleft= vright ise ZF ayarlanır. vleft > vright ise, ZF ayarlanmamış (0'a ayarlanmış) ve SF= OF olur. Eğer vleft < vright ise, ZF ayarsızdır ve SF≠ OF. vleft < vright ise neden SF= OF yapar?

**B.7** Aşağıdaki NASM kod parçasını göz önünde bulundurun:

```
mov , 0 cmp
al, al je
next
```

Tek bir komuttan oluşan eşdeğer bir program yazınız.

**B.8** Aşağıdaki C programını düşünün:

```
/* 3 tamsayının ortalamasını almak için basit bir C
programı */ main ()
{ int avg;
  int i1= 20;
  int i2= 13;
  int i3= 82;
  avg= (i1+ i2+ i3)/3;
}
```

Bu programın NASM versiyonunu yazın.

## 798 EK B / TOPLANTI ALANI VE İLGİLİ KONULAR

- B.9** Aşağıdaki C kod parçasını göz önünde bulundurun:

```
if (EAX== 0 ( EBX= 1;  
else EBX= 2;
```

Eşdeğer bir NASM kod parçası yazın.

- B.10** Initialize data yönergeleri birden fazla konumu başlatmak için kullanılabilir. Örneğin,

```
db 0x55,0x56,0x57
```

üç bayt ayırır ve değerlerini başlatır.

NASM, hesaplamların geçerli assembly konumunu içermesine izin vermek için \$ özel belirteçini destekler. Yani \$, ifadeyi içeren satırın başındaki montaj konumuna göre değerlendirilir. Yukarıdaki iki gerçeği aklımızda tutarak, aşağıdaki yönerge dizisini göz önünde bulundurun:

```
message db 'hello, world' msglen  
equ $-message
```

msglen sembolüne hangi değer atanır?

- B.11** Üç sembolik değişken V1, V2, V3'ün tamsayı değerler içerdiğini varsayıñ. En küçük değeri ax tamsayısına taşıyan bir NASM kod parçası yazın. Yalnızca mov, cmp ve jbe komutlarını kullanın.

- B.12** Bu komutun etkisini açıklayın: cmp eax, 1 Hemen önceki komutun eax'in içeriğini güncellediğini varsayıñ.

- B.13** xchg komutu iki içeriğini değiştirmek için kullanılabilir. X86 komut setinin bu komutu desteklemediğini varsayıñ.

- a. Sadece push ve pop komutlarını kullanarak xchg ax, bx uygulayın.  
b. Sadece xor komutunu kullanarak xchg ax, bx komutunu uygulayın (diğer yazmaçları dahil etmeyin).

- B.14** Aşağıdaki programda, a, b, x, y'nin ana bellek konumları için semboller olduğunu varsayıñ. Bu program ne yapar? Eşdeğer mantığı C'de yazarak soruya cevap verebilirsınız.

```
mov    eax, a  
mov    ebx, b  
xor    eax, x  
xor    ebx, y  
veya  eax, ebx  
jnz   L2  
L1:          talimatlar dizisi...  
            jmp   L3  
L2:          başka bir talimat dizisi...  
L3:
```

- B.15** Bölüm B.1, iki tamsayıının en büyük ortak bölenini hesaplayan bir C programı içermektedir.

- a. Algoritmayı kelimelerle tanımlayın ve programın en büyük ortak bölenin hesaplanmasına yönelik Öklid algoritması yaklaşımını nasıl uyguladığını gösterin.  
b. Şekil B.3'a daki assembly programına, C programı ile aynı mantığı uyguladığını açıktığa kavuşturmak için yorumlar ekleyin.  
c. Şekil B.3'b'deki program için (b) bölümünü tekrarlayın.

- B.16** a. İki geçişli bir assembler gelecek sembollerini işleyebilir ve bu nedenle bir komut gelecek sembolünü bir operand olarak kullanabilir. Bu, yönergeler için her zaman doğru değildir. Örneğin EQU yönergesi bir gelecek sembolü kullanamaz. "A EQU B+ 1" yönergesi, B önceden tanımlanmışsa kolayca çalıştırılabilir, ancak B bir gelecek sembolü ise imkansızdır. Bunun nedeni nedir?

- b.** Birleştiricinin bu sınırlamayı ortadan kaldırması için bir yol önerin, böylece herhangi bir kaynak satırı gelecekteki sembollerini kullanabilir.
- B.17** Aşağıdaki formda bir MAX symbol yönüğü düşünün: symbol MAX ifadeler listesi  
Etiket zorunludur ve işlenen alanındaki en büyük ifadenin değerine atanır. Örnek:  
MSGLEN MAX A, B, C ;burada A, B, C tanımlanmış sembollerdir  
MAX, Assembler tarafından nasıl ve hangi geçişte yürütülür?

# REFERANSLAR

## KISALTMALAR

ACM Association for Computing Machinery  
IEEE Elektrik ve Elektronik Mühendisleri Enstitüsü NIST  
Ulusal Standartlar ve Teknoloji Enstitüsü

- AGAR89** Agarwal, A. *Analysis of Cache Performance for Operating Systems and Multiprogramming*. Boston: Kluwer Academic Publishers, 1989.
- AGER87** Agerwala, T., ve Cocke, J. *Yüksek Performanslı Azaltılmış Komut Seti İşlemcileri*. Teknik Rapor RC12434 (#55845). Yorktown, NY: IBM Thomas J. Watson Araştırma Merkezi, Ocak 1987.
- ALLA13** Allan, G. "Gömülü Uygulamalarda DDR4 Banka Grupları." *Chip Design*, 26 Ağustos 2013. chipdesignmag.com
- ALTS12** Alschuler, F., ve Gallmeier, J. "Heterojen Sistem Mimarisi: Çok Çekirdekli Görüntü İşleme CPU ve GPU Elemanlarının Bir Karışımını Kullanır." *Embedded Computing Design*, 6 Aralık 2012.
- AMDA67** Amdahl, G. "Büyük Ölçekli Hesaplama Yeteneği Elde Etmek için Tek İşlemci Yaklaşımının Geçerliliği." *AFIPS Konferansı Bildirileri*, 1967.
- AMDA13** Amdahl, G. "Bilgisayar Mimarisi ve Amdahl Yasası." *Bilgisayar*, Aralık 2013. **ANDE67a** Anderson, D., Sparacio, F. ve Tomasulo, F. "IBM System/360 Model 91: Makine Felsefesi ve Komut İşleme." *IBM Araştırma ve Geliştirme Dergisi opment*, Ocak 1967.
- ANDE67b** Anderson, S., vd. "IBM System/360 Model 91: Floating-Point Execution Unit." *IBM Araştırma ve Geliştirme Dergisi*, Ocak 1967. SWAR90, Cilt 1]'de yeniden basılmıştır.
- ANTH08** Anthes, G. "x86 için Sirada Ne Var?" *ComputerWorld*, 16 Haziran 2008.
- AROR12** Arora, M., ve diğerleri. "CPU-GPU Entegrasyonu Çağında CPU'nun Rolünün Yeniden Tanımlanması." *IEEE Micro*, Kasım/Aralık 2012.
- ATKI96** Atkins, M. "PC Yazılım Performansı Ayarlama." *IEEE Bilgisayar*, Ağustos 1996.
- AZIM92** Azimi, M., Prasad, B., ve Bhat, K. "Two Level Cache Architectures." *Bildiriler, COMPCON '92*, Şubat 1992.
- BACO94** Bacon, F., Graham, S., ve Sharp, O. "Compiler Transformations for High-Performance Computing." *ACM Computing Surveys*, Aralık 1994.
- BAIL93** Bailey, D. "RISC Mikroişlemciler ve Bilimsel Hesaplama." *Proceedings, Supercomputing'93*, 1993.
- BELL70** Bell, C., Cady, R., McFarland, H., Delagi, B., O'Loughlin, J. ve Noonan, R. "A New Architecture for Minicomputers-The DEC PDP-11." *Proceedings, Spring Joint Computer Conference*, 1970.
- BELL71** Bell, C., ve Newell, A. *Bilgisayar Yapıları: Okumalar ve Örnekler*. New York: McGraw-Hill, 1971.
- BELL78a** Bell, C., Mudge, J., ve McNamara, J. *Bilgisayar Mühendisliği: Donanım Sistemleri Tasarımına DEC Bakışı*. Bedford, MA: Digital Press, 1978.
- BELL78b** Bell, C., Newell, A. ve Siewiorek, D. "PDP-8'in Yapısal Düzeyleri." İçinde [BELL78a].
- BELL78c** Bell, C., Kotok, A., Hastings, T. ve Hill, R. "DEC System-10'un Evrimi." *Communications of the ACM*, Ocak 1978.
- BENH92** Benham, J. "Integers'ın Bilgisayar Temsillerini Sunmak için Geometrik Bir Yaklaşım." *SIGCSE Bulletin*, Aralık 1992.

- BOOT51** Booth, A. "İşareti İkili Çarpma Tekniği." *The Quarterly Journal of Mechanics and Applied Mathematics*, Cilt 4, No. 2, 1951.
- BORK03** Borkar, S. "Gigascale Cipler Edinmek: Moore Yasasının Devamında Karşılaşılan Zorluklar ve Fırsatlar." *ACM Queue*, Ekim 2003.
- BRAD91a** Bradlee, D., Eggers, S., ve Henry, R. "Kayıt Kümesi Büyüklüğü ve Yapısı ile Kod Üretme Stratejisinin RISC Performansı Üzerindeki Etkisi." *Bildiriler, 18. Yıllık Uluslararası Bilgisayar Mimarisi Sempozyumu*, Mayıs 1991.
- BRAD91b** Bradlee, D., Eggers, S., ve Henry, R. "Integrating Register Allocation and Instruction Scheduling for RISCs." *Proceedings, Programlama Dilleri ve İşletim Sistemleri için Mimari Destek Dördüncü Uluslararası Konferansı*, Nisan 1991.
- BREW97** Brewer, E. "Kümeleme: Çarp ve Fethet." *Veri İletişimi*, Temmuz 1997.
- BURG97** Burger, D., ve Austin, T. "SimpleScalar Araç Seti, Sürüm 2.0." *Computer Architecture News*, Haziran 1997.
- BURK46** Burks, A., Goldstine, H. ve von Neumann, J. *Bir Elektronik Bilgisayar Aletinin Mantıksal Tasarımının Ön Tartışması*. U.S. Army Ordinance Department için hazırlanan rapor, 1946, [BELL71]'de yeniden basılmıştır.
- BUYY99** Buyya, R. *Yüksek Başarımlı Küme Hesaplama: Mimariler ve Sistemler*. Upper Saddle River, NJ: Prentice Hall, 1999.
- CANT01** Cantin, J., ve Hill, H. "Seçilmiş SPEC CPU2000 Benchmarks için Önbellek Performansı." *Bilgisayar Mimarisi Haberleri*, Eylül 2001.
- CART06** Carter, P. *PC Assembly Dili*. 23 Temmuz 2006. <http://www.drpaulcarter.com/pcasm/>.
- CEKL97** Cekleov, M., ve Dubois, M. "Sanal-Adres Önbellekleri, Bölüm 1: Tek İşlemcilerde Sorunlar ve Çözümleri." *IEEE Micro*, Eylül/Ekim 1997.
- CHAI82** Chaitin, G. "Grafik Renklendirme Yoluyla Kayıt Tahsisi ve Dökümü." *Proceedings, SIGPLAN Symposium on Compiler Construction*, Haziran 1982.
- CHOW86** Chow, F., Himmelstein, M., Killian, E., ve Weber, L. "Engineering a RISC Compiler System." *Bildiriler, COMPCON Bahar '86*, Mart 1986.
- CHOW87** Chow, F., Correll, S., Himmelstein, M., Killian, E. ve Weber, L. "How Many Addressing Modes Are Enough?" *Proceedings, Second International Conference on Architectural Support for Programming Languages and Operating Systems*, Eylül 1987.
- CHOW90** Chow, F., ve Hennessy, J. "The Priority-Based Coloring Approach to Register Allocation." *ACM Transactions on Programming Languages*, Eylül 1990.
- CITR06** Citron, D., Hurani, A. ve Gnädrey, A. "Harmonik veya Geometrik Ortalama: Gerçekten Önemli mi?" *Computer Architecture News*, Eylül 2006.
- CLAR85** Clark, D., ve Emer, J. "VAX-11/780 Çeviri Performansı: Simülasyon ve Ölçüm." *ACM Transactions on Computer Systems*, Şubat 1985.
- COHE81** Cohen, D. "On Holy Wars and a Plea for Peace." *Computer*, Ekim 1981.
- COOK82** Cook, R., ve Dande, N. "An Experiment to Improve Operand Addressing." *Proceedings, Programlama Dilleri ve İşletim Sistemleri için Mimari Desteği Sempozyumu*, Mart 1982.
- COLW85a** Colwell, R., Hitchcock, C., Jensen, E., Brinkley-Sprunt, H. ve Kollar, C. "Computers, Complexity, and Controversy." *Computer*, Eylül 1985.
- COLW85b** Colwell, R., Hitchcock, C., Jensen, E., Brinkley-Sprunt, H., ve Kollar, C. "More Controversy About 'Computers, Complexity, and Controversy.'" *Computer*, Aralık 1985.
- COON81** Coonen, J. "Underflow and Denormalized Numbers." *IEEE Bilgisayar*, Mart 1981.
- COUT86** Coutant, D., Hammond, C. ve Kelley, J. "Yeni Nesil Hewlett-Packard Bilgisayarları için Derleyiciler." *Bildiriler, COMPCON Bahar '86*, Mart 1986.
- CRAG79** Cragon, H. "Kod Alanı Gereksinimleri ve Çeşitli Mimarilerin Performansı Üzerine Bir Değerlendirme." *Computer Architecture News*, Şubat 1979.
- CRAW90** Crawford, J. "i486 CPU: Talimatları Bir Saat Döngüsünde Yürütme." *IEEE Micro*, Şubat 1990.

- CURR11** Curran, B., ve diğerleri. "zEnterprise 196 Sistemi ve Mikroişlemcisi." *IEEE Micro*, Mart/Nisan 2011.
- DATT93** Dattatreya, G. "İlk Derste İkili Aritmetik Öğretimine Sistematisk Bir Yaklaşım." *IEEE Transactions on Education*, Şubat 1993.
- DAVI87** Davidson, J., ve Vaughan, R. "Komut Kümesi Karmaşıklığının Program Boyutu ve Bellek Performansı Üzerindeki Etkisi." *Proceedings, Second International Conference on Architectural Support for Programming Languages and Operating Systems*, Ekim 1987.
- DENN68** Denning, P. "Program Davranışı için Çalışma Kümesi Modeli." *Communications of the ACM*, Mayıs 1968.
- DERO87** DeRosa, J., ve Levy, H. "Dal Mimarilerinin Değerlendirilmesi." *Proceedings, Fourteenth Annual International Symposium on Computer Architecture*, 1987.
- DEWA90** Dewar, R., ve Smosna, M. *Mikroişlemciler: Bir Programın Görüşü*. New York: McGraw-Hill, 1990.
- DEWD84** Dewdney, A. "Çekirdek Savaşçı Denen Oyunda Düşman Programlar Bitlerin Savaşına Giriyor." *Scientific American*, Mayıs 1984.
- DOBO13** Dobos, I. ve . *IBM zEnterprise EC12 Teknik Kılavuzu*. IBM Redbook SG24-8049-01, Aralık 2013.
- DOWD98** Dowd, K., ve Severance, C. *Yüksek Performanslı Hesaplama*. Sebastopol, CA: O'Reilly, 1998.
- EISC07** Eischen, C. "RAID 6 Daha Fazla Alanı Kapsıyor." *Network World*, 9 Nisan 2007.
- ELAY85** El-Ayat, K. ve Agarwal, R. "Intel 80386-Mimari ve Uygulama." *IEEE Micro*, Aralık 1985.
- FATA08** Fatahalian, K., ve Houston, M. "A Closer Look at GPUs." *Communications of the ACM*, Ekim 2008.
- FEIT15** Feitelson, D. *Bilgisayar Sistemleri Performans Değerlendirmesi İçin İş Yükü Modellemesi*. Cambridge, Birleşik Krallık: Cambridge University Press, 2015.
- FLEM86** Fleming, P. ve Wallace, J. "How Not to Lie with Statistics: Kıyaslama Sonuçlarını Toplamanın Doğru Yolu." *Communications of the ACM*, Mart 1986.
- FLYN72** Flynn, M. "Bazı Bilgisayar Organizasyonları ve Etkinlikleri." *IEEE Transactions on Computers*, Eylül 1972.
- FLYN87** Flynn, M., Mitchell, C., ve Mulder, J. "And Now a Case for More Complex Instruction Sets." *Computer*, Eylül 1987.
- FOG08** Fog, A. *Assembly Dilinde Alt Programları Optimize Etme: x86 Platformları İçin Bir Optimizasyon Kılavuzu*. Kopenhag Üniversitesi Mühendislik Fakültesi, 2008. <http://www.agner.org/optimize/>
- FRAI83** Frailey, D. "Bir Bilgisayar Mimarisinin Kelime Uzunluğu: Tanımlar ve Uygulamalar." *Computer Architecture News*, Haziran 1983.
- GENU04** Genu, P. *A Cache Primer*. Uygulama Notu AN2663. Freescale Semiconductor, Inc., 2004. (Premium İçerik Belgesi bölümünde mevcuttur)
- GHAI98** Ghai, S., Joyner, J. ve John, L. *Üçüncü Seviye Önbellegin Etkinliğinin Araştırılması*. Teknik Rapor TR-980501-01, Bilgisayar Mimarisi Laboratuvarı, Austin'deki Texas Üniversitesi, 1998.
- GIBB04** Gibbs, W. "A Split at the Core." *Scientific American*, Kasım 2004.
- GIFF87** Gifford, D. ve Spector, A. "Örnek Olay İncelemesi: IBM'in System/360-370 Mimarisi." *Communications of the ACM*, Nisan 1987.
- GILA95** Giladi, R., ve Ahituv, N. "Bir Performans Değerlendirme Ölçütü Olarak SPEC." *Bilgisayar*, Ağustos 1995.
- GOER12** Goering, R. "Yeni Bellek Teknolojileri NAND Flash ve DRAM'e Meydan Okuyor." *Cadence Industry Insight Blogs*, 22 Ağustos 2012. [http://community.cadence.com/cadence\\_blogs\\_8/b/ii/archive/2012/08/22/keynote-new-memory-technologies-challenge-nand-flash-and-dram](http://community.cadence.com/cadence_blogs_8/b/ii/archive/2012/08/22/keynote-new-memory-technologies-challenge-nand-flash-and-dram)

- GOLD54** Goldstine, H., Pomerene, J. ve Smith, C. *Bir Elektronik Hesaplama Aracının Fiziksel Gerçekliği Üzerine Nihai İlerleme Raporu*. Princeton: İleri Araştırmalar Enstitüsü Elektronik Bilgisayar Projesi, 1954.
- GSOE08** Gsoedl, J. "Kati Hal: Depolamada Yeni Sınır." *Storage*, Temmuz 2008.
- GUST88** Gustafson, J. "Amdahl Yasasının Yeniden Değerlendirilmesi." *Communications of the ACM*, Mayıs 1988.
- EL98** Handy, J. *The Cache Memory Book*. San Diego: Academic Press, 1998.
- HARR06** Harris, W. "Multi-Core in the Source Engine." *bit-tech.net teknik raporu*, 2 Kasım 2006.
- HAYE98** Hayes, J. *Bilgisayar Mimarisi ve Organizasyonu*. New York: McGraw-Hill, 1998.
- HEAT84** Heath, J. "RISC l'in Yeniden Değerlendirilmesi." *Computer Architecture News*, Mart 1984.
- HENN07** Henning, J. "SPEC CPU Suite Büyümesi: Tarihsel Bir Perspektif." *Computer Architecture News*, Mart 2007.
- HENN12** Hennessy, J., ve Patterson, D. *Bilgisayar Mimarisi: Nicel Bir Yaklaşım*. Waltham, MA: Morgan Kaufman, 2012.
- HENN82** Hennessy, J., . "Performans Artışı için Donanım/Yazılım Ödünleşimleri." *Proceedings, Symposium on Architectural Support for Programming Languages and Operating Systems*, Mart 1982.
- HENN84** Hennessy, J. "VLSI İşlemci Mimarisi." *IEEE Transactions on Computers*, Aralık 1984.
- HILL64** Hill, R. "Stored Logic Programming and Applications." *Datamation*, Şubat 1964.
- HILL89** Hill, M. "CPU Önbelleklerinde İlişkiselligin Değerlendirilmesi." *IEEE Transactions on Computers*, Aralık 1989.
- HUCK83** Huck, T. *Bilgisayar Mimarilerinin Karşılaştırmalı Analizi*. Stanford Üniversitesi Teknik Rapor No. 83-243, Mayıs 1983.
- HUGG05** Huggahalli, R., Iyer, R. ve Tetrick, S. "Yüksek Bant Genişlikli Ağ I/O için Doğrudan Önbellek Erişimi." *Bildiriler, 32. Yıllık Uluslararası Bilgisayar Mimarisi Sempozyumu*, 2005.
- HUGU91** Huguet, M., ve Lang, T. "Tek Pencereli Kayıt Dosyalarında Azaltılmış Kayıt Kaydetme/Geri Yükleme için Mimar Destek." *ACM Transactions on Computer Systems*, Şubat 1991.
- HWAN93** Hwang, K. *Gelişmiş Bilgisayar Mimarisi*. New York: McGraw-Hill, 1993.
- HWAN99** Hwang, K., ve diğerleri. "Hiyerarşik Kontrol Noktası ve Tek G/C Alanı ile SSI Kümeleri Tasarlama." *IEEE Concurrency*, Ocak-Mart 1999.
- INTE98** Intel Corp. *Pentium Pro ve Pentium II İşlemciler ve İlgili Ürünler*. Aurora, CO, 1998.
- INTE04** Intel Araştırma ve Geliştirme. *Tera Çağının Mimarisi*. Intel Beyaz Kitap, Şubat 2004.
- INTE08** Intel Corp. *Intel I/O Acceleration Technology ve Microsoft Windows Server 2008'in Tümleşik Ağ Hızlandırma Özellikleri*. Intel Beyaz Kitap, Şubat 2004.
- INTE12** Intel Corp. *Intel Veri Doğrudan I/O Teknolojisi (Intel DDIO): A Primer*. Intel Beyaz Kitap, Şubat 2012.
- INTE14** Intel Corp. *Intel Processor Graphics Gen8'in Bilgisayar Mimarisi*. Intel Beyaz Kitap, Eylül 2014.
- ITRS14** *The International Technology Roadmap For Semiconductors, 2013 Edition*, 2014. <http://www.itrs.net>
- JACO95** Jacob, B., ve Mudge, T. "Bilgisayar Performansının Hesaplanması Üzerine Notlar." *Michigan Üniversitesi Teknik Raporu CSE-TR-231-95*, Mart 1995.
- JACO08** Jacob, B., Ng, S. ve Wang, D. *Bellek Sistemleri: Önbellek, DRAM, Disk*. Boston: Morgan Kaufmann, 2008.
- JAIN91** Jain, R. *Bilgisayar Sistemi Performans Analizi Sanatı*. New York: Wiley, 1991.

- JAME90** James, D. "Multiplexed Buses: Endian Savaşları Devam Ediyor." *IEEE Micro*, Eylül 1983.
- JEFF12** Jeff, B. *Güç ve Enerji Tasarrufu için big.LITTLE Teknolojisindeki Gelişmeler*. ARM Beyaz Kitap, Eylül 2012.
- JOHN91** Johnson, M. *Superscalar Microprocessor Design*. Englewood Cliffs, NJ: Prentice Hall, 1991.
- JOHN04** John, L. "Bir Kiyaslama Paketinin Genel Performansını Göstermek için Tek Bir Sayı Bulma Üzerine Daha Fazla Bilgi." *Computer Architecture News*, Mart 2004.
- JOUP88** Jouppi, N. "Superscalar versus Superpipelined Machines." *Computer Architecture News*, Haziran 1988.
- JOUP89a** Jouppi, N., ve Wall, D. "Superscalar ve Superpipelined Makineler için Kullanılabilir Komut Seviyesi Paralelliği." *Proceedings, Üçüncü Uluslararası Programlama Dilleri ve İşletim Sistemleri için Mimari Destek Konferansı*, Nisan 1989.
- JOUP89b** Jouppi, N. "Komut Seviyesi ve Makine Paralelliğinin Düzgün Olmayan Dağılımı ve Performans Üzerindeki Etkisi." *IEEE Transactions on Computers*, Aralık 1989.
- KAPP00** Kapp, C. "Küme Bilgisayarları Yönetmek." *Dr. Dobb's Journal*, Temmuz 2000.
- KATE83** Katevenis, M. *VLSI için Azaltılmış Komut Kümelii Bilgisayar Mimarileri*. Doktora Tezi, Bilgisayar Bilimleri Bölümü, Berkeley'deki California Üniversitesi, Ekim 1983. MIT Press, Cambridge, MA, 1985 tarafından yeniden basılmıştır.
- KATZ89** Katz, R., Gibson, G. ve Patterson, D. "Yüksek Performanslı Hesaplama için Disk Sistemi Mimarisi." *Proceedings of the IEEE*, Aralık 1989.
- KNUT71** Knuth, D. "FORTRAN Programları Üzerine Ampirik Bir Çalışma." *Software Practice and Experience*, Cilt 1, 1971.
- KUCK77** Kuck, D., Parker, D. ve Sameh, A. "Kayan Nokta Aritmetiğinde Yuvarlama Yöntemlerinin Analizi." *IEEE Transactions on Computers*, Temmuz 1977.
- KULT13** Kulrursay, E., vd. "STT-RAM'in Enerji Verimli Bir Ana Bellek Alternatifisi Olarak Değerlendirilmesi." *IEEE Uluslararası Sistem ve Yazılım Performans Analizi Sempozyumu (ISPASS)*, 2013.
- KUMA07** Kumar, A., ve Huggahalli, R. "Önbellek Tutarlılık Protokollerinin Ağ Trafiğinin Üretilmesi Üzerindeki Etkisi." *40th IEEE/ACM International Symposium on Microar- chitecture*, 2007.
- LEE91** Lee, R., Kwok, A. ve Briggs, F. "Bir Superscalar SPARC İşlemcisinin Kayan Nokta Performansı." *Proceedings, Programlama Dilleri ve İşletim Sistemleri için Mimari Destek Dördüncü Uluslararası Konferansı*, Nisan 1991.
- LEE10** Lee, B., ve diğerleri. "Faz Değişimi Teknolojisi ve Ana Belleğin Geleceği." *IEEE Micro*, Ocak/Şubat 2010.
- LEAN06** Lean, E., ve Maccabe, A. "Önbellek Enjeksiyonu Kullanarak Çip-Çoklu İşlemciler için Bellek Bant Genişliğini Azaltma." *15. IEEE Yüksek Performanslı Ara Bağlantılar Sempozyumu*, Ağustos 2007.
- LEON07** Leonard, T. "Dragged Kicking and Screaming: Source Multicore." *Proceedings, Game Developers Conference 2007*, Mart 2007.
- LILJ88** Lilja, D. "Boru Hatlı İşlemcilerde Dallanma Cezasının Azaltılması." *Bilgisayar*, Temmuz 1988.
- LILJ93** Lilja, D. "Büyük Ölçekli Paylaşımı Bellekli Çoklu İşlemcilerde Önbellek Tutarlılığı: Sorunlar ve Karşılaştırmalar." *ACM Computing Surveys*, Eylül 1993.
- LILJ00** Lilja, D. *Bilgisayar Performansının Ölçülmesi: Bir Uygulayıcının Rehberi*. Cambridge, Ingiltere: Cambridge Üniversitesi Yayınları, 2000.
- LITT61** Küçük, J. "Kuyruk Formülü İçin Bir Kanıt:  $L = \lambda W$ ." *Operations Research*, Mayıs-Haziran 1961.
- LITT11** Little, J. "50. Yıldönümünde Little Yasası." *Yöneylem Araştırması*, Mayıs-Haziran 2011.

- AŞK96** Lovett, T. ve Clapp, R. "Bir CC-NUMA Sisteminin Uygulanması ve Performansı." *Bildiriler, 23. Yıllık Uluslararası Bilgisayar Mimarisi Sempozyumu*, Mayıs 1996.
- LUND77** Lunde, A. "Komut Kümesi İşlemci Mimarilerinin Bazı Özelliklerinin Ampirik Değerlendirmesi." *Communications of the ACM*, Mart 1977.
- MACD84** MacDougall, M. "Komut Düzeyinde Program ve Süreç Modelleme." *IEEE Computer*, Temmuz 1984.
- MANJ01a** Manjikian, N. "SimpleScalar Araç Setinde Daha Fazla Geliştirme." *Computer Architecture News*, Eylül 2001.
- MANJ01b** Manjikian, N. "SimpleScalar Araç Setinin Çok İşlemcili Geliştirmeleri." *Computer Architecture News*, Mart 2001.
- MASH04** Mashey, J. "Benchmark Araçları Savaşçı: Ateşkes Zamanı." *Computer Architecture News*, Eylül 2004.
- MASH95** Mashey, J. "CISC vs. (ya da RISC gerçekten nedir)." *USENET comp.arch haber grubu, makale 46782*, Şubat 1995.
- MAK97** Mak, P., ve . "Tam Paylaşımı Belleğe Sahip Bir Sistemde Paylaşımı Önbellek Kümeleri." *IBM Araştırma ve Geliştirme Dergisi*, Temmuz/Eylül 1997.
- MAYB84** Mayberry, W., ve Efland, G. "Cache Boosts Multiprocessor Performance." *Bilgisayar Tasarımı*, Kasım 1984.
- MCDO05** McDougall, R. "Extreme Software Scaling." *ACM Queue*, Eylül 2005.
- MCDO06** McDougall, R., ve Laudon, J. "Multi-Core Microprocessors are Here." ; *login*, Ekim 2006.
- MCMA93** McMahon, F., "L.L.N.L Fortran Çekirdekleri Testi." *Kaynak*, Ekim 1993. [www.netlib.org/benchmark/livermore](http://www.netlib.org/benchmark/livermore)
- MOOR65** Moore, G. "Cramming More Components Onto Integrated Circuits." *Electronics Magazine*, 19 Nisan 1965. *Proceedings of the IEEE*, Ocak 1998'de yeniden basılmıştır.
- MORR74** Morris, M. "Kiviat Graphs-Conventions and Figures of Merit." *ACM SIGMETRICS Performans Değerlendirme İncelemesi*, Ekim 1974.
- MORS78** Morse, S., Pohlman, W. ve Ravenel, B. "Intel 8086 Mikroişlemcisi: 8080'in 16-bit Evrimi." *Computer*, Haziran 1978.
- MYER78** Myers, G. "Depodan Depoya Mimaride İfadelerin Değerlendirilmesi." *Computer Architecture News*, Haziran 1978.
- NASM12** NASM Geliştirme Ekibi. *NASM-The Netwide Assembler*. <http://nasm.us/>, 2012. **NOVI93** Novitsky, J., Azimi, M., ve Ghaznavi, R. "Pentium İşlemcilere Dayalı Sistem Performansının Optimize Edilmesi." *Bildiriler, COMPCON '92*, Şubat 1993.
- NVID09** NVIDIA, "NVIDIA'nın Yeni Nesil CUDA Hesaplama Mimarisi: Fermi." *NVIDIA Beyaz Kitap*, Ağustos 2009.
- NVID14** NVIDIA, "CUDA C Programlama Kılavuzu." *NVIDIA Belgeleri*, 2014.
- OWEN08** Owens, J., ve . "GPU Computing." *Proceedings of the IEEE*, Mayıs 2008.
- PADE81** Padegs, A. "System/360 ve Ötesi." *IBM Araştırma ve Geliştirme Dergisi*, Eylül 1981.
- PARH10** Parhami, B. *Bilgisayar Aritmetiği: Algoritmalar ve Donanım Tasarımı*. Oxford: Oxford Üniversitesi Yayıncıları, 2010.
- PATT82a** Patterson, D., ve Sequin, C. "Bir VLSI RISC." *Computer*, Eylül 1982.
- PATT82b** Patterson, D. ve Piepho, R. "Üst Düzey Dil Desteğinde RISC'lerin Değerlendirilmesi." *IEEE Micro*, Kasım 1982.
- PATT84** Patterson, D. "RISC Watch." *Computer Architecture News*, Mart 1984.
- PATT85a** Patterson, D. "Azaltılmış Komut Kümeli Bilgisayarlar." *ACM İletişim*, Ocak 1985.
- PATT85b** Patterson, D., ve Hennessy, J. "Response to 'Computers, Complexity, and Controversy.'" *Computer*, Kasım 1985.

- PATT88** Patterson, D., Gibson, G. ve Katz, R. "A Case for Redundant Arrays of Inexpensive Disks (RAID)." *Bildiriler, ACM SIGMOD Veri Yönetimi Konferansı*, Haziran 1988.
- PEDD14** Peddle, J. "Inside Intel's Gen 8 GPU." *EE Times*, 22 Eylül 2014.
- PEIR99** Peir, J., Hsu, W. ve Smith, A. "CPU Önbellek Bellekleri için İşlevsel Uygulama Teknikleri." *IEEE Transactions on Computers*, Şubat 1999.
- PELE97** Peleg, A., Wilkie, S. ve Weiser, U. "Intel MMX for Multimedia PCs." *Communications of the ACM*, Ocak 1997.
- PFIS98** Pfister, G. *In Search of Clusters*. Upper Saddle River, NJ: Prentice Hall, 1998.
- PHAN07** Phanskar, A., Joshi, A. ve John, L. "Analysis of Redundancy and Application Balance in the SPEC CPU2006 Benchmark Suite." *ACM Uluslararası Bilgisayar Mimarisi Sempozyumu, ISCA'07*, 2007.
- POLL99** Pollack, F. "CMOS Süreç Teknolojilerinin Gelecek Nesillerinde Yeni Mikro Mimari Zorlukları" (açılış konuşması). *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*, 1999.
- PRES01** Pressel, D. "Bilimsel Uygulamalar için Ön Getirme ve Akış Tamponlarının Kullanımına İlişkin Temel Sınırlamalar." *Proceedings, ACM Symposium on Applied Computing*, Mart 2001.
- PROP11** Prophet, G. "Hızlandırmayı Artırmak için GPU'ları Kullanın." *IDN*, 2 Aralık 2011.
- PRZY88** Przybylski, S., Horowitz, M., ve Hennessy, J. "Performance Trade-offs in Cache Design." *Proceedings, 15th Annual International Symposium on Computer Architecture*, Haziran 1988.
- PRZY90** Przybylski, S. "Blok Boyutu ve Getirme Stratejilerinin Performans Etkisi." *Proceedings, 17. Yıllık Uluslararası Bilgisayar Mimarisi Sempozyumu*, Mayıs 1990.
- RADI83** Radin, G. "The 801 Minicomputer." *IBM Araştırma ve Geliştirme Dergisi*, Mayıs 1983.
- RAGA83** Ragan-Kelley, R., ve Clark, R. "RISC Teorisinin Büyük Bir Bilgisayara Uygulanması." *Computer Design*, Kasım 1983.
- RAOU09** Raouk, S., . "Faz Değişimli Rastgele Erişimli Bellek: Ölçeklenebilir Bir Teknoloji." *IBM Araştırma ve Geliştirme Dergisi*, Temmuz/Eylül 2008.
- RECH98** Reches, S., ve Weiss, S. "Dinamik Dallanma Tahmin Şemalarında Yol Geçmişinin Uygulanması ve Analizi." *IEEE Transactions on Computers*, Ağustos 1998.
- REDD76** Reddi, S. ve Feustel, E. "Bilgisayar Mimarisi için Kavramsal Bir Çerçeve." *Computing Surveys*, Haziran 1976.
- REIM06** Reimer, J. "Valve Goes Multicore." *ars technica*, 5 Kasım 2006. [arstechnica.com/articles/paedya/cpu/valve-multicore.ars](http://arstechnica.com/articles/paedya/cpu/valve-multicore.ars)
- ROBI07** Robin, P. "Linux ve ARM Thumb-2 ISA ile Deney." *Embedded Linux Conference*, 2007.
- RODR01** Rodriguez, M., Perez, J., ve Pulido, J. "Simetrik Çoklu İşlemcilerde Önbellekleri Test Etmek İçin Bir Eğitim Aracı." *Microprocessors and Microsystems*, Haziran 2001.
- SAND10** Sanders, J., ve Kandrot, E. *Örneklerle CUDA: Genel Amaçlı GPU Programlamaya Giriş*. Reading, MA: Addison-Wesley Professional, 2010.
- SATY81** Satyanarayanan, M., ve Bhandarkar, D. "VAX-11 Çeviri Arabalığı Organizasyonunda Tasarım Ödünleşimleri." *Computer*, Aralık 1981.
- SEBE76** Sebern, M. "Minibilgisayar Uyumlu Bir Mikrobilgisayar Sistemi: DEC LSI-11." *Proceedings of the IEEE*, Haziran 1976.
- SERL86** Serlin, O. "MIPS, Dhrystones, and Other Tales." *Datamation*, 1 Haziran 1986.
- SHAN38** Shannon, C. "Rôle ve Anahtarlama Devrelerinin Sembolik Analizi." *AIEE Transactions*, Cilt 57, 1938.
- SHAR03** Sharma, A. *Gelişmiş Yarı İletken Bellekler: Architectures, Designs, and Applications*. New York: IEEE Press, 2003.

- SHUM13** Shum, C., Susaba, F. ve Jacobi, C. "IBM zEC12: Üçüncü Nesil Yüksek Frekanslı Ana Bilgisayar Mikroişlemcisi." *IEEE Micro*, Mart/Nisan 2013.
- SIEW82** Siewiorek, D., Bell, C., ve Newell, A. *Bilgisayar Yapıları: İlkeler ve Örnekler*. New York: McGraw-Hill, 1982.
- SIMO96** Simon, H. *Yapay Bilimler*. Cambridge, MA: MIT Press, 1996.
- SLAV12** Slavici, V., . "MADNESS Çerçeveşinde Düzensiz Hesaplama Büyüklüğü CPU-GPU Kümelerine Uyarlanması." *IEEE International Conference on Cluster Computing*, 2012.
- SMIT82** Smith, A. "Cache Memories." *ACM Computing Surveys*, Eylül 1982.
- SMIT87** Smith, A. "CPU Önbellek Bellekleri için Hat (Blok) Boyutu Seçimi." *IEEE Transactions on Communications*, Eylül 1987.
- SMIT88** Smith, J. "Bilgisayar Performansının Tek Bir Sayı ile Karakterize Edilmesi." *Communications of the ACM*, Ekim 1988.
- SMIT89** Smith, M., Johnson, M. ve Horowitz, M. "Limits on Multiple Instruction Issue." *Proceedings, Üçüncü Uluslararası Programlama Dilleri ve İşletim Sistemleri için Mimari Destek Konferansı*, Nisan 1989.
- SMIT95** Smith, J., ve Sohi, G. "Superscalar İşlemcilerin Mikro Mimarisi." *Proceedings of the IEEE*, Aralık 1995.
- SOHI90** Sohi, G. "Yüksek Performanslı Kesilebilir, Çok Fonksiyonlu Birim, Pipelined Bilgisayarlar için Komut Mantığı." *IEEE Transactions on Computers*, Mart 1990.
- STAL14a** Stallings, W. "Gigabit Wi-Fi." *Internet Protocol Journal*, Eylül 2014.
- STAL14b** Stallings, W. "Gigabit Ethernet." *Internet Protocol Journal*, Aralık 2014.
- STAL15** Stallings, W. *İşletim Sistemleri, İç Yapılar ve Tasarım İlkeleri*, Sekizinci Baskı. Upper Saddle River, NJ: Pearson, 2015.
- STEN90** Stenstrom, P. "A Survey of Cache Coherence Schemes of Multiprocessors." *Computer*, Haziran 1990.
- STEV64** Stevens, W. "The Structure of System/360, Part II: Implementation." *IBM Systems Journal*, Cilt 3, No. 2, 1964. SIEW82]de yeniden basılmıştır.
- STEV13** Stevens, A. *AMBS 4 ACE ve big.little İşleme Teknolojisine Giriş*. ARM Beyaz Kitap, 29 Temmuz 2013.
- STRE78** Strecker, W. "VAX-11/780: DEC PDP-11 Ailesine Sanal Adres Uzantısı." *Bildiriler, Ulusal Bilgisayar Konferansı*, 1978.
- STRE83** Strecker, W. "Önbellek Belleklerinin Geçici Davranışı." *ACM Transactions on Computer Systems*, Kasım 1983.
- STR179** Stritter, E., ve Gunter, T. "Değişen Dünya için Bir Mikroişlemci Mimarisi: Motorola 68000." *Computer*, Şubat 1979.
- TAMI83** Tamir, Y., ve Sequin, C. "RISC'de Kayıt Dosyasını Yönetmek için Stratejiler." *IEEE Transactions on Computers*, Kasım 1983.
- TANE78** Tanenbaum, A. "Makine Mimarisi için Yapılandırılmış Programlamanın Etkileri." *ACM İletişim*, Mart 1978.
- TI12** Texas Instruments. *66AK2H12/06 Çok Çekirdekli DSP+ARM KeyStone II Çip Üzerinde Sistem (SoC)*. Veri Kılavuzu SPRS866, Kasım 2012.
- TJAD70** Tjaden, G. ve Flynn, M. "Detection and Parallel Execution of Independent Instructions." *IEEE Transactions on Computers*, Ekim 1970.
- TOON81** Toong, H. ve Gupta, A. "An Architectural Comparison of Contemporary 16-Bit Microprocessors." *IEEE Micro*, Mayıs 1981.
- TUCK67** Tucker, S. "Sistem/360 için Mikroprogram Kontrolü." *IBM Systems Journal*, No. 4, 1967.
- UNGE02** Ungerer, T., Rubic, B., ve Silc, J. "Multithreaded Processors." *The Computer Journal*, No. 3, 2002.
- UNGE03** Ungerer, T., Rubic, B., ve Silc, J. "A Survey of Processors with Explicit Multithreading." *ACM Computing Surveys*, Mart 2003.

## 808 REFERANSLAR

- VANC14** Vance, A. "Dünyadaki Mobil Cihazların %99'u ARM Çipi İçeriyor" *Business Week*, 10 Şubat 2014.
- VONN45** Von Neumann, J. *EDVAC Üzerine Bir Raporun İlk Taslağı*. Moore Okulu, Pennsylvania Üniversitesi, 1945. *IEEE Annals on the History of Computing*, No. 4, 1993'te yeniden basılmıştır.
- VRAN80** Vranesic, Z. ve Thurber, K. "Teaching Computer Structures." *Computer*, Haziran 1980.
- WALL85** Wallich, P. "Toward Simpler, Faster Computers." *IEEE Spectrum*, Ağustos 1985.
- WANG99** Wang, G., ve Tafti, D. "Büyük Seyrek Doğrusal Sistemlerin Çözümü için Hiyerarşik Bellek Sistemi Mikroişlemcilerde Performans Artışı." *International Journal of Supercomputing Applications*, Vol. 13, 1999.
- WEIC90** Weicker, R. "Yayın Benchmark'lara Genel Bir Bakış." *Bilgisayar*, Aralık 1990. **WEIN75** Weinberg, G. *Genel Sistem Düşünçesine Giriş*. New York: Wiley, 1975. **WEIS84** Weiss, S., ve Smith, J. "Pipelined Supercomputers'da Komut Düzenleme Mantığı." *IEEE Transactions on Computers*, Kasım 1984.
- WHIT97** Whitney, S., vd. "SGI Origin Yazılım Ortamı ve Uygulama Performansı." *Bildiriler, COMPCON Bahar '97*, Şubat 1997.
- WILK51** Wilkes, M. "Otomatik Hesaplama Makinesi Tasarlamannın En İyi Yolu." *Bildiriler, Manchester Üniversitesi Bilgisayar Açıltı Konferansı*, Temmuz 1951.
- WILK53** Wilkes, M., ve Stringer, J. "Mikro Programlama ve Elektronik Dijital Bilgisayarda Kontrol Devrelerinin Tasarımı." *Proceedings of the Cambridge Philosophical Society*, Nisan 1953. SIEW82]'de yeniden basılmıştır.
- WILL90** Williams, F., ve Steven, G. "M68000 Ailesinde Adres ve Veri Kaydi Ayırımı." *Computer Architecture News*, Haziran 1990.
- YEH91** Yeh, T., ve Patt, N. "İki Seviyeli Uyarlamalı Eğitim Dalı Tahmini." *Proceedings, 24th Annual International Symposium on Microarchitecture*, 1991.
- ZHOU09** Zhou, P., vd. "Faz Değişimli Bellek Teknolojisi Kullanan Dayanıklı ve Enerji Verimli Bir Ana Bellek." *ACM International Symposium on Computer Architecture, ISCA'09*, 2009.

# DİZİN

## A

Mutlak adres, 483  
Mutlak ölçeklenebilirlik, 633  
Erişim kontrolü, 313-314 Erişim süresi (gecikme), 123 Akümülatör (AC), 14, 85, 418  
ACE (Advanced Extensible Interface Coherence Extensions), 674-675  
Acorn RISC Makinesi (ARM), 34. *Ayrıca bakınız*  
    ARM mimarisi  
Aktif ikincil kümeleme yöntemi, 635  
Toplayıcılar, 392-396  
    4 bit, 394  
    bir uygulama, 395 çoklu bit, 394-395  
        tek bit, 394  
    İlave, 337-340  
    ikili, 392  
    taşma kuralı, 337-338  
    ikili tamamlayıcı, 338-339  
Adreslenebilir birimler, 122  
Adres otobüsü, 101  
Adres oluşturma sıralaması, 743-744 Adresleme modları, 457-463  
    KOL, 466-469, 526-527  
    otomatik indeksleme, 462  
    temel kayıt, 462  
    temel, 459  
    doğrudan, 459  
    yer değiştirme, 461-463 etkin adres (EA), 458 acil, 459  
    indeksleme, 462-463  
    dolaylı, 459-460  
    Intel x86, 463-466  
    MIPS R4000, 560-561  
    mod alanı, 458  
    PC-relative, 461  
    postindexing, 462-463  
    ön endeksleme, 463  
    kayıt, 460-461  
    dolaylı kayıt, 461  
    göreceli, 461  
    SPARC, 568  
    yığın, 463  
Adres satırları, 101  
Adres değiştirme talimatları, 17 Adres kayıtları, 492  
Adres alanı, 304-305  
Gelişmiş RISC Makineleri. *Bkz:* ARM mimarisi

Hızalama kontrolü (AC), 519 Hızalama Maskesi (AM), 521 Tahsis, Pentium 4 işlemci, 517  
Amdahl, Gene, 53  
Amdahl yasası, 53-55, 660  
Bilgi Değişimi için Amerikan Standart Kodu (ASCII), 232, 421, 422  
AND kapı, 389  
AND işlemi, 430  
Bağımlılık Karşılığı, 509, 586  
Uygulama düzeyinde paralellik, 662  
Uygulama işlemcileri, 31-32  
Uygulama programlama arayüzü (API), 42 Aritmetik ve mantık birimi (ALU), 490, 494, 542  
    ekleme, 337-340  
    ARM Cortex-A8, 600-601  
    bölüm, 347-350  
    bayrak değerleri, 329-330  
    kayan nokta gösterimi, 350-358  
    IAS bilgisayar, 11, 13, 16  
    IBM sistem/360, 22  
    IBM 3033 mikro komut yürütme, 755 giriş ve çıkışlar, 330  
    tam sayılar, 330-350  
    çok çekirdekli bilgisayar, 8  
    çarpma, 340-347  
    için operandlar, 329  
    tek işlemcili bilgisayar, 6  
    SPARC mimarisi, 567  
    çıkarma, 337-340  
    Texas Instruments 8800 Yazılım Geliştirme Kartı (SDB), 762-765  
Aritmetik talimatlar, 416, 421, 429  
Aritmetik ortalaması, 60, 62  
Aritmetik işlemler, 429, 431  
Aritmetik kaydırma, 345, 431  
ARM adresleme modları, 466-469, 526-527  
    iptal modu, 527  
    şube talimatı, 468  
    veri işleme talimatları, 468 istisna modları, 526, 527  
    hızlı kesme modu, 527  
    dizinleme yöntemleri, 466-467  
    kesme modu, 527 yükleme  
    ve saklama, 466-468  
    load/store çoklu adresleme, 468-469 ofset değeri, 466-467  
    postindexing, 468  
    ön endeksleme, 467  
    ayrıcalıklı modlar, 526  
    gözetmen modu, 527

- ARM adresleme modları (*devam*) sistem modu, 527  
 tanımlanmamış mod, 527  
 kullanıcı modu, 526
- ARM mimarisı, 33-39 ACE  
 önbellek satırı durumları, 675  
 hizalama denetimi, 424  
 şube talimatları, 444  
 veri işleme talimatları, 444  
 veri türleri, 423-425  
 Endian desteği, 425  
 evrim, 34  
 talimatları genişletme, 444  
 talimat seti, 34-35  
 talimatları, 417  
 yükleme ve saklama talimatları, 444  
 çarpma talimatları, 444  
 hizalanmamış erişim, 423  
 paralel toplama ve çıkarma talimatları, 444  
 ürünler, 35-39  
 SETEND talımı, 425  
 durum kaydı erişim talimatları, 444 hizalanmamış erişim, 424  
 durum kodlarının kullanımı, 444-445  
 VLSI, 34
- ARM Cortex-A8, 596-604  
 adres oluşturma birimi (AGU), 599  
 dallanma hedef arabelleği (BTB), 599  
 çift sorun kısıtlamaları, 602  
 talimat akışı, blok diyagramı, 597 küresel geçmiş tamponu (GHB), 599  
 in-order sorunu, 597-598 komut kod çözme birimi, 599-600 komut getirme birimi, 598-599 tamsayı yürütme birimi, 600-603 tamsayı boru hattı, 598  
 yükleme/depolama boru hattı, 602  
 bellek sisteminin komut zamanlamaları üzerindeki etkileri, 601  
 SIMD ve kayan nokta talimatları, 603-604 ARM Cortex-A15 çekirdeği, 670-673  
 ACE önbellek satırı durumları, 675 enerji tüketimi, 673  
 boru hatları, 672
- ARM Cortex-A15 MPCore, 677-681  
 aktif kesme, 679 blok diyagramı, 677-678 önbellek tutarlılığı, 680-681  
 çekirdek, 677  
 CPU arayüzü, 680  
 hata ayıklama birimi ve arayüzü, 677  
 doğrudan veri müdahalesi (DDI), 681 çoğaltılmış etiket RAM'leri, 681  
 genel kesme denetleyicisi (GIC), 677  
 genel zamanlayıcı, 677  
 donanım kesmeleri, 680
- etkin olmayan kesme, 679 işlemcilerarası kesmeler (IPI'lар), 679 kesme işleme, 678-680  
 L1 önbellek tutarlılığı, 681  
 L2 önbellek tutarlılığı, 681  
 eski FIQ hattı, 680 geçiş hatları, 681  
 bekleyen kesme, 679  
 özel zamanlayıcı ve/veya watchdog kesmeleri, 680  
 program izleme, 677  
 snoop kontrol birimi (SCU), 677, 680-681
- ARM Cortex-A7 çekirdeği, 671-673  
 ACE önbellek satırı durumları, 675 enerji tüketimi, 673  
 boru hatları, 672  
 ARM Cortex-M3, 604-607  
 şube yönlendirme, 606  
 şube speküasyonu, 606  
 otobüs matrisi, 605  
 veri izleme noktası ve izleme (DWT), 605  
 dalarla başa çıkma, 606-607  
 hata ayıklama erişim portu, 605 kod çözme aşaması, 606  
 gömülü izleme makro hücresi, 605 flaş yama ve kesme noktası birimi, 604  
 bellek koruma birimi, 604  
 iç içe vektörlü kesme denetleyicisi (NVIC), 604
- boru hattı, 607  
 boru hattı yapısı, 605-606  
 işlemci çekirdeği, 604  
 seri kablo görüntüleyici, 605  
 Thumb-2 talımı, 605-606  
 uyandırma kesme denetleyicisi (NVIC), 604
- ARM komut biçimi, 479-482  
 anlık sabitler, 479 Thumb komut seti, 479-481  
 Thumb-2 komut seti, 481-482 ARM bellek yönetimi, 309-314  
 erişim kontrolü, 313-314  
 Etki Alanı Erişim Kontrol Kaydı, 314 formatlar, 310-313  
 bellek organizasyonu, 309-310 parametreler, 313 çeviri görünüm arabelleği (TLB), 309-310 sanal bellek adres çevirisi, 310-311 ARM işlemci, 524-530  
 nitelikler, 525  
 kesinti işleme, 529-530  
 işlemci organizasyonu, 525  
 kayıtlar, 527-529
- Dizi işlemcisi, 615  
 Montajcılar, 761, 763  
*Assembly dili, 413, 415, 482-484. Ayrıca bakınız Talimat formatları*  
 BASIC ifadesi, 482

- pseudoinstruction, 483  
 sembolik program, 483
- Onaylama, sinyal, 377  
 İlişkisel erişim, 123  
 İlişkisel haritalama, 138-140  
 İlişkisel bellek, 123  
 Otomatik indeksleme, 462  
 Yardımcı bellek, 127
- B**
- Geriye dönük uyumluluk, 29  
 Dengeli aktarım, 105  
 Banka grupları, 184  
 Taban, 307  
 Temel adres, 297  
 Temel rakam, 319  
 Temel kayıt adresleme, 462  
 Toplu iş sistemi, 280  
 Bell Labs, 17  
 Benchmark programları, 68  
 BFU (ikili kayan nokta birimi), 10 Önyargılı gösterim, 351  
 Büyük endian sıralaması, 452  
 Büyük.Küçük Çip, 671  
 İkili toplayıcı, 339  
 İkili toplama, 392  
 İkili Kodlanmış Ondalık (BCD), 384 İkili sistem, 321  
 Bit-interleaved parity disk performansı (RAID seviye 3), 210-211  
 Bit uzunluğu dönüştürme, 332  
 Bit sıralaması, endian, 455  
 Blade sunucular, 638-639  
 Bloklanmış çok iş parçacıklı skaler, 631  
 Bloklanmış çok iş parçacıklı superscalar, 632  
 Bloklanmış çok iş parçacıklı VLIW, 632  
 Bloklanmış çok iş parçacıklı, 630  
 Blok düzeyinde dağıtılmış eşlik diskı performansı (RAID seviye 5), 212  
 Blok düzeyinde eşlik diskı performansı (RAID seviye 4), 211-212  
 Blok çoğlayıcı, 262  
 Bloklar, 122, 690  
     Booth'un algoritması, 346-347  
     önbellek, 160  
     I/O, 408  
     mantık, 408  
      $m$ , 129, 134-135  
     bellek, 133, 137, 140-142, 619  
     paketler veya protokol, 257  
     süreç kontrolü, 494  
     SDRAM'ler, 182  
     SPLD, 406  
     bant, 222  
     iplik, 690-691, 696  
 Blu-ray DVD, 217, 221  
 Boole, George, 373
- Boole cebiri, 373-375, 392  
 VE işlemi, 374 temel kimlikleri, 375 Boole işleçleri, 375  
 exclusive-or (XOR) işlemi, 374 NAND işlevi, 374  
 NOT işlemi, 374  
 VEYA işlemi, 374  
 Boole fonksiyonları, cebirsel sadeleştirme uygulanması, 381  
 kanonik form, 381  
 Karnaugh haritaları, 381-386  
 NAND ve NOR kapıları, 388  
 Quine-McCluskey yöntemi, 384-387  
 basitleştirme kuralları, 382-383  
 ürünlerin toplamı (SOP) formu, 379, 380 üç kombinasyonun toplamı, 379  
 Boolean (mantık) talimatları, 416  
 Booth algoritması, 344-347 Dalar koşullu talimatlar, 509-515  
 kontrol tehlikesi (branş tehlikesi), 508-509  
 korelatör olarak, 515  
 Cortex-M3 işlemci, 606-607  
 gecikmeli, 515, 557-558  
 dinamik stratejiler, 51-512  
 tarih yaklaşımı, 513-515  
 tarih tablosu, 513  
 komut getirme aşaması, 513-514 döngü tamponu için, 510-511  
 döngü kapatma, 515  
 mikro talimatlar, 744 çoklu akışlar için, 509-510 pipelining ve, 509-515  
 tahmin, 511-515, 589, 593-594 önceden hazırlanmış şube hedefi, 510  
 Şube tahmini, 47-48  
 Şube hedef tamponu (BTB), 511, 513, 593  
 British Broadcasting Corporation (BBC), 34  
 Tamponlar, 83  
 Veri yolu tahkim tekniği, G/Ç, 243 Veri yolu ara bağlantısı, 100-102  
 Bus master, 146  
 Otobüs izleme yaklaşımı, 146  
 Otobüs genişliği, 25-27  
 Bayt, 111  
 Bayt çoğlayıcı, 262 Bayt sıralaması, endian, 452
- C**
- Önbellek, 6  
 bankacılık, 703  
 Cortex-R, 35  
 enjeksiyon, 259  
 Bayan, 130, 146, 152, 258-259, 261, 310, 581, 594, 630, 632, 677, 681

- Önbellek tutarlılığı, 621-624  
 dizin protokolleri, 623  
 donanım tabanlı çözümler, 623-624  
 çok çekirdekli bilgisayarlar, 674-675  
 snoo protokolleri, 623-624  
 yazılım, 622-623  
 write-validate yaklaşımı, 624  
 politika yazma, 622  
 yazma-güncelleme protokolü, 624
- Önbellek uyumlu tekdüze olmayan bellek erişimi (CC-NUMA), 640  
 avantajlar ve dezavantajlar, 643 organizasyon, 641-642
- Önbellek Devre Dışı (CD), 521
- Önbellek vuruşu, 130, 148
- Önbellek hattı, 133
- Ön bellek, 128-149, 536  
 adresler, 131-133  
 yüksek performanslı hesaplama (HPC), 131 hat, 129  
 hat boyutu, 129, 147  
 mantıksal önbellek, 132  
 haritalama işlevi, 133-144  
 çok düzeyli önbellek, 147-149  
 önbellek sayısı, 147-149  
 fiziksel önbellek, 132-133  
 okuma işlemi, 129-130  
 değiştirme algoritmaları, 145  
 boyutlar, 133, 134  
 bölünmüş önbellek, 149  
 yapısı, 128, 129  
 etiket, 129  
 birleşik önbellek, 149  
 sanal adres, 133  
 sanal önbellek, 132  
 yazma politikası, 145-147
- Önbellek kaçırma, 130
- Önbellek seti, 140
- Arama/iade talimatları, 438-439
- Kondansatörler, 20
- Carry lookahead, 395
- CD-ROM, 217
- CD-RW, 217
- Merkezi işlem birimi (CPU), 83, 689 genel amaçlı mikrobilgisayar, 25 Intel 8085, 721  
 ara bağlantı, 6  
 iç yapı, 490  
 G/C kanallarına katılım, 261-262 bellek ve, 83  
 çok çekirdekli bilgisayar, 6, 667-671 performans ve watt başına performans, 692 ikinci nesil bilgisayarlarında, 18  
 tek işlemcili bilgisayar, 4
- GPU'ya karşı, 691-692  
 sistem veriyolu ile, 490
- Zincirleme, 301  
 Karakter veri işlenenleri, 470  
 Karakteristik tablo, 397  
 Çip çoklu işlem, 630  
 Çip çoklu işlemci (çok çekirdekli), 628-633, 657  
 Cips, 7-8, 21  
 ARM, 34  
 kontrol deposu, 752  
 DDR, 183  
 DRAM bellek, 172-173 EPROM paketi, 172-173 dört çekirdekli, 52  
 yüksek hız, 50  
 entegre devre, 21, 24  
 Intel Dört Çekirdekli Xeon işlemci, 8-9 G/C denetleyicisi, 8  
 LSI, 751  
 bellek, 8, 9, 25, 47-48, 172-173  
 mikrodenetleyici, 32  
 mikroişlemci, 32  
 çok çekirdekli, 102, 268, 657, 663, 665, 668, 682  
 PU, 683  
 RAM, 390  
 yarı iletken bellek, 170-172  
 iki çekirdekli, 52  
 ultra büyük ölçekli entegrasyon (ULSI), 24 Yonga seti, PCI Express, 108  
 Saat (veri yolu) döngüsü, 57  
 Saatlı S-R flip-flop, 397-399 Saat hızı, 57  
 Saat hızı, 57  
 Saat tik tak, 57  
 Bulut denetçi, 648-649  
 Bulut komisyoncusu, 648-649  
 Bulut taşıyıcı, 648-649  
 Bulut bilişim, 39-42  
 aktörler, 648-649  
 geniş ağ erişimi, 644 topluluk bulutu, 646  
 bilgi işlem, 39  
 dağıtım modelleri, 646  
 unsurlar, 643-647  
 temel özellikleri, 644-645 hibrit bulut, 646  
 hizmet olarak altyapı (IaaS), 42 ölçülen hizmet, 644  
 ağ oluşturma, 40  
 isteğe bağlı self-servis, 644-645  
 hizmet olarak platform (PaaS), 41  
 özel bulut, 646  
 genel bulut, 646  
 hızlı esneklik, 644  
 referans mimarı, 647-649  
 kaynak havuzu, 645  
 hizmet modelleri (SaaS, PaaS, IaaS), 645-646, 649

- hizmet olarak yazılım (SaaS), 40-41  
depolama, 40  
Bulut tüketicileri, 649  
Bulut sağlayıcı, 648-649  
Kümeler, 615, 633-639  
aktif ikincil, 635  
faydalı ve sınırlamalar, 633, 635  
blade sunucular, SMP ile  
karşılaştırıldığında 638-639, 639  
bilgisayar mimarisi, 637-638  
konfigürasyonlar, 633-636  
yük dengeleme, 636  
ara yazılım hizmetleri ve işlevleri, 638 işletim sistemi tasarımları, 636-637 hesaplamanın paralelleştirilmesi, 636-637  
pasif bekleme, 635  
ayrı sunucu, 635  
sunucu, 634  
paylaşımı disk, 634  
paylaşılan disk yaklaşımı, 636  
paylaşılan hiçbir şey yaklaşımı, 636 tek sistem görüntüsü, 637  
Kaba taneli iş parçacığı, 663 Kombinasyonel devre  
Boole denklemleri, 378-388  
kod çözüçüler, 390-392  
tanımlanmış, 378  
grafik semboller, 378  
çoklayıcılar, 388-390  
salt okunur bellek (ROM), 392  
sırılı devreler, 396-405 doğruluk tablosu, 378  
Ticari bilgisayarlar, 11 Talimatların işlenmesi (emekliye ayrılması), 590 İletişim cihazları, 231  
Topluluk bulutu, 646 Kompakt disk (CD), 217-220  
CD kaydedilebilir (CD-R), 219-220  
CD yeniden yazılabilir (CD-RW), 220  
CD-ROM (kompakt disk salt okunur bellek), 217-219  
CD-R optik disk, 220 Sıkıştırma, I/O bellek, 296  
Derleyici tabanlı kayıt optimizasyonu, 547-549  
Karmaşık komut kümelii bilgisayar (CISC), 27, 540  
özellikleri, 538  
çağdaş motivasyonlar, 549-551 vs. RISC tasarımları, 553-555, 570-571  
Karmaşık PLD'ler (CPLD'ler), 406, 408  
Bilgisayar mimarisi, 2  
Bilgisayar talimatı, 413 Çip üzerinde bilgisayar, 32 Bilgisayar organizasyonu, 2 Bilgisayarlar  
mimarlık, 2  
bileşenler, 81-83  
aile kavramı, 536  
işlevi, 83-98 temel unsurları, 11  
nesiller, 17  
nesilden nesile uyumluluğu, 2  
tarihçe, 11-27  
talimat, getirme ve yürütme işlevi, 84-89  
komut seti mimarisi (ISA), 2 ara bağlantı yapıları, 99-100  
organizasyon, 2  
yapı ve işlev, 3-11 Bilgisayar sistemi performansı  
Amdahl yasası, 53-55  
kıyaslama ilkeleri, 67-68 ortalama değer hesaplama, 59-67 saat hızı, 57  
için tasarım, 46-52  
çip organizasyonu ve mimarisindeki gelişmelerin ardından, 50-52  
GPU'larda genel amaçlı hesaplama (GPGPU), 52-53  
grafik işlem birimleri (GPU'lar), 52-53 komut yürütme hızı, 58-59  
Little yasası, 55-56  
birçok entegre çekirdek (MIC), 52 mikroişlemci hızı, 47-48  
çoklu işlemciler, 52  
performans dengesi, 48-49  
SPEC karşılaştırmaları, 68-74 Koşullu dallanma talimatları, 433 Koşullu atlama, 439  
Durum kodları, 492, 744, 757 avantajları ve dezavantajları, 493 ARM mimarisi, 444-445  
EFLAGS kaydı, 518  
Intel x86, 438-440  
program durum sözcüğü (PSW), 495  
RISC tabanlı makineler, 560  
Sabit açısal hız (CAV), 198, 218  
Sabit doğrusal hız (CLV), 198, 218  
Kontrol, 85  
erişim, 313-314  
kesme, 720  
G/C modülleri, 231, 232-233  
çizgiler, 101-102  
mantıksal, 13  
makine talimatları, 415  
depolama kontrolü (SC), 683  
ve zamanlama, 232-233  
Kontrol tehlikesi (dallanma tehlikesi), pipelining, 508-509  
Kontrol tehlikeleri, pipelining, 509

Kontrolörler  
önbeliği, 146,  
623  
disk, 107  
disk sürücüsü, 231  
fanouts, 269  
I/O, 108, 121, 235, 236, 262  
yığın depolama, 35  
bellek ve çevre birimi, 657, 668  
mikrodenetleyiciler, 32, 187  
ağ arayüzü, 107  
Kontrol hatları, 101  
Kontrol kayıtları, 519-521  
Kontrol sinyalleri, 716-719  
Kontrol ünitesi (CU), 4, 6, 490  
karakterizasyonu, 715  
kontrol sinyalleri, 716-719  
yürütme döngüsü, 712-713  
getirme döngüsü, 709-711  
ilevsel gereksinimler, 714-715  
kablolu uygulama, 724-727  
IAS bilgisayar, 11, 13  
dolaylı döngü, 711-712 girişler  
ve çıkışlar, 716-717 komut  
döngüsü, 713-714  
dahili işlemci organizasyonu ve, 719-720 kesme  
döngüsü, 712  
mikro-operasyonlar, 708-714  
işlemci, 714-724  
COP (özel yardımcı işlemci), 11 Core i7  
EE 4960X mikroişlemci, 29 Cortex-A ve  
Cortex-A50, 35  
Cortex-M serisi işlemciler, 35-39 analog  
arayızlar, 38  
otobüs matrisi, 38  
saat yönetimi, 38  
çekirdek, 38  
hata ayıklama erişim portu (DAP), 36  
hata ayıklama mantığı, 36  
gömülü izleme makro hücresi (ETM)  
modülü, 36  
enerji yönetimi, 38  
ICODE arayüzü, 38  
bellek, 38  
bellek koruma ünitesi, 38  
yuvalanmış vektör kesme denetleyicisi (NVIC),  
36 paralel G/C bağlantı noktaları, 38  
çevresel veri yolu, 39  
güvenlik, 38  
seri arayızlar, 38  
SRAM ve çevresel arabirim, 38 32 bit  
veri yolu, 39  
zamanlayıcılar ve  
tetikleyiciler, 38 Cortex-R,  
35  
Sayaçlar, 402-405  
dalgalanma, 402-  
403  
eszamanlı, 403-404

Texas Instruments 8800 Yazılım Geliştirme Kartı  
(SDB), 759  
C programlama, 159  
CRAY C90, 122  
CUDA (Compute Unified Device Architecture), 689-  
691  
çekirdeklər, 690, 696, 697  
CUDA çekirdek/SM sayısı, 694  
programlama dili, 689, 690  
Geçerli program durum kayıtları (CPSR), ARM, 527  
Bir program için komut başına döngü (CPI), 58  
Döngü çalışma, 249  
Çevrim süresi, 57-58, 525, 562, 620  
eğitim, 18, 501, 503, 716  
bellek, 18, 58, 123  
boru hattı, 504-506  
işlemci, 58  
Döngüsel artıklık kontrolü (CRC), 106

**D**

Papatya zinciri teknigi, G/C, 243  
Veritabanı ölçeklendirme, 618  
Veri tamponlama, I/O modülleri, 233  
Veri yolu, 101  
Veri önbeliği, 152  
Veri kanalı, 18  
Veri iletişim, 4  
Veri alışverişleri, 636  
Veri akışı, komut döngüleri, 497-499 Veri  
akışı analizi, 48  
Veri biçimlendirme, manyetik diskler, 196-199  
Veri tehlikeleri, pipelining, 508-509  
Veri (veri yolu) hatları,  
101 Data-L2, 11  
Veri hareketi, 4  
Veri işleme, 4, 20, 85, 416, 421, 444, 601, 667  
ARM, 525  
talimat adresleme, 468  
yükleme/depolama modeli,  
525 makine talimatları, 415  
Veri işleme talimat adreslemesi, 468 Veri kayıtları,  
491  
Veri depolama, 4, 20, 40, 42, 124, 167, 265, 416  
makine talimatları, 415  
Veri aktarımı, 427-428  
IAS bilgisayar, 16  
talimatlar, 427-428  
I/O modülleri, 231  
paketlenmiş, 103  
Veri türleri  
ARM mimarisi, 423-425  
IEEE 754 standartı, 424  
Intel x86 mimarisi, 422-423  
paketlenmiş SIMD, 422  
Hata ayıklama erişim bağlantı noktası (DAP), 36

- Hata ayıklama mantığı, 36  
 Ondalık sistem, 319-320  
 Kod çözme komut birimi, Cortex-A8 işlemci, 599  
 Kod çözücüleri, 390-392, 595  
     demultiplexer olarak, 390, 392  
 DEC P DP- 8, 23-24  
 Özel işlemci, 32  
 Derin gömülü sistemler, 32-33 Gecikmeli dallanma, boru hattı, 557-558 Gecikmeli yük, boru hattı, 558-559 Gecikme yuvası, 557  
 Talep çağrıları, 299-300  
 DeMorgan teoremi, 375, 377, 388 Cihaz iletişim, I/O modülleri, 233 D flip-flop, 399, 401  
 DFU (ondalık kayan nokta birimi), 10 Diferansiyel sinyalizasyon, 105  
 Dijital bilgisayar, 20  
 Digital Equipment Corporation (DEC), PDP serisi bilgisayarlar, 23  
 Dijital mantık  
     Boole cebiri, 373-375  
     kombinasyonel devreler, 378-396  
     kapılar, 376-378  
     programlanabilir mantık cihazı (PLD), 405-409  
     sırılı devreler, 396-405  
 Dijital çok yönlü disk (DVD), 217, 220-221  
 Doğrudan erişim, 122-123  
 Doğrudan erişimli cihaz, 223  
 Doğrudan adres, 463  
 Doğrudan adresleme, 459, 473  
 Doğrudan önbellek erişimi (DCA), 254-261  
     performans sorunu ve faydalari, 257-259  
     stratejiler, 259  
 Doğrudan Veri G/Ç, 254  
     önbellek yazma işlemi, 260  
     DMA ile karşılaştırma, 260 paket girişi, 259-261  
     paket çiktısı, 259-261  
     strateji, 261  
     TCP/IP protokol işleyicisi, 261  
     geri yazma stratejisi, 260  
     write-through stratejisi, 260 Yön bayrağı (DF), 519 Yönergeler, 704  
 Doğrudan eşleme teknigi, 134-138 Doğrudan bellek erişimi (DMA), 98  
     DDIO ile karşılaşılması, Intel 8237'nin 260 kontrol/komut kayıtları,  
         253-254  
     8237 Sistem veriyolunun DMA kullanımı, 252 fly-by DMA denetleyicisi, 253  
     fonksiyon, 249-251  
     bir komut döngüsü sırasında kesme kesme noktaları, 250  
             programlanmış ve kesme güdümlü G/Ç, 248-249  
             SMP'ler, 619  
             paylaşılan son düzey önbelleği kullanma, 255-257 Dizin protokolleri, 623  
     Kirli (kullanım) biti, 146 Devre dışı  
     birakılmış kesme, 95  
     Ayrık bileşenler, 20-21 Disk önbelleği, 158 Disk sürücüsü, G/Ç, 232 Dağıtıcı, 288  
     Yer değiştirme adreslemesi, 461-463  
         modu, 465  
     Dağıtılmış tahlkim, 118  
     Temetti, 347  
     Bölüm, 347-350  
         işaretsiz ikili için akış şeması, 348  
         kısmi kalan, 347-349  
         ikili tamamlayıcı, 349  
     Bölen, 347  
     Çift veri hızı, 183  
     Çift veri oranlı DRAM (DDR RAM), 183-184  
     Çift taraflı diskler, 200  
     Sürücü, Pentium 4 işlemci, 517  
     Çift yedekli disk performansı (RAID seviye 6), 212  
     DVD, 217  
     DVD-R, 217  
     DVD-ROM, 217  
     DVD-RW, 217  
     Dinamik erişimli rastgele bellek (DRAM) teknolojisi, 104  
     Dinamik RAM (DRAM), 147, 167-168, 171-172, 187  
         DDR SDRAM, 183-184  
         pin yapılandırması, 172  
         senkron DRAM (SDRAM), 181-182
- E**  
 EAS/390 bellek sistemi, 432  
 Kenar tetiklemeli flip-flop, 403  
 EDVAC (Elektronik Ayrık Değişken Bilgisayar), 11  
 Etkin adres (EA), 458, 460  
 EFLAGS kaydı, Intel x86 işlemciler, 518-519  
 Elektriksel olarak silinebilir programlanabilir salt okunur bellek (EEPROM), 170 Gömülü Mikroişlemci Benchmark  
     Konsorsiyumu (EEMBC) kıyaslaması, 482  
 Gömülü sistemler, 29-33  
     derin, 32-33  
     işletim sistemi (OS), 31 organizasyon, 29-30  
     Gömülü izleme makro hücresi (ETM) modülü, 36  
     Emülasyon (EM), 520

Etkinleştirilmiş kesme, 95, 712  
 Kodlanmış mikro talimat formatı, 748-751  
 Silinebilir programlanabilir salt okunur bellek (EPROM), 170, 172  
 Hata kontrol işlevi, 106 Hata düzeltme kodları, 175  
 Hata düzeltme, 216-217  
     yarı iletken bellek, 174-180 Hata algılama, G/C modülleri, 234  
 ESCON (Kurumsal Sistem Bağlantısı), 269  
 Ethernet, 265-266  
 İstisnalar, kesintiler ve 522-523, 529  
 Uyarma tablosu, 403  
 Yürütme döngüsü, 84, 87, 92  
     mikro işlemler (mikro-ops), 712-713 Yürütme.  
*Ayrıca bakınız* Program yürütme  
     getirme ve talimat, 496-497 getirilmiş talimat, 85  
 IBM 3033 mikro talimat, 743, 754-755 talimat yürütme hızı, 58-59  
 I/O programı, 89, 91  
 MIPS R4000 mikroişlemcisinde yüklemeler ve depolamalar, 565  
 LSI-11 mikro talimat, 751-754  
 mikro programlama, 745-755  
 multithreading, 628  
 RISC talimi, 537-542  
 spekulatif, 48  
 superscalar, 48, 589-590  
 Genişletme kartları, 7  
 Üs taşması, 358  
 Üs değeri, 351  
 Genişletilmiş İkili Kodlu Ondalık Değişim Kodu (EBCDIC), 421, 432  
 Uzatma Tipi (ET), 520  
 Harici arayüz standartları, 263-266 Harici bellek, 39, 121-122, 127, 185, 187  
     manyetik disk, 195-203  
     manyetik bant, 222-224  
     optik disk sistemleri, 217-222  
     RAID, 204-213  
     katı hal sürücüler (SSD'ler), 212-216

**F**

Failback, 636  
 Yük Devretme, 636  
 Hata yönetimi, kümeler, 636 Aile kavramı, 536  
 Fanouts, 269  
 Getirme döngüsü, 15, 84, 85, 87, 92, 93, 469, 497-498, 709-711  
     mikro-İşlemler (mikro-ops), 709-711 Getirilen kod bitleri, 175  
     Getirme komut birimi, 489, 496  
         Cortex-A8 işlemci, 599  
         infaz, 85

Getirme çakışması, ardışık sıralama, 501  
 Sahada programlanabilir kapı dizisi (FPGA), 406-409  
     G/C blokları, 408  
     mantık bloğu, 409  
     yapı, 408  
 İnce taneli iş parçacığı, 663  
 FireWire Seri Veri Yolu, 264  
 Ürün Yazılımı, 107, 215, 731  
 İlk nesil bilgisayarlar. IAS bilgisayarına *bakınız*  
 İlk giren ilk çıkar (FIFO) algoritması, 145  
 Sabit kafalı disk, 199  
 Sabit nokta gösterimi, 335  
 Sabit boyutlu böltümler, 294-295  
 Bayrak, kayıt organizasyonu, 527-528  
 Bayraklar. *Bkz.* Durum kodları  
 Flash bellek, 170, 185-187 NOR  
     ve NAND, 186-187  
     operasyon, 185-186  
 Parmak arası terlikler, 396-400  
     temel, 400  
     saatli S-R, 397-399  
     D, 399, 401  
     kenar tetiklemeli, 403  
     J-K, 399-400, 402-403  
 Flit, 104  
 Kayan noktalı aritmetik, 358-367, 424, 576, 583, 697  
     ekleme, 359-361  
     bölüm, 361  
     üs taşması, 358  
     üs düşük akışı, 358  
     İkili için IEEE standartı, 365-367 eksiz sonsuzluk, 365  
     çarpma, 361-362  
     normalleştirme, 361  
     hassasiyetle ilgili hususlar, 362-365 artıya yuvarlama, 365  
     en yakına yuvarlama, 364  
     sıfır doğru yuvarlama, 365  
     anlamlı taşıma, 359  
     significand düşük akışı, 359  
     çıkarma, 359-361  
 Kayan nokta, 350-358  
     taban, 351  
     önyargılı gösterim, 351 ikili sayılarla, 350-352 üs değeri, 351  
     IBM S/390 mimarisi, 353-354 ikili için IEEE standartı, 354-358 negatif taşıma, 353  
     negatif düşük akış, 353  
     normal sayı, 351-352  
     pozitif taşıma, 353  
     pozitif düşük akış, 353

ilkeler, 350-354  
 anlam, 351-352  
 işaret, 351  
 Saniye başına kayan nokta işlemleri (FLOPs), 692, 200  
 Disket (temaslı) manyetik diskler, 196, 199  
 Akış kontrol işlevi, 104, 106, 109, 115  
 Akış bağımlılığı, 580  
 FORTRAN programları, 159, 282, 540  
 Kesirler, 322-324  
 Çerçeveeler, G/C belleği, 269  
 Ön uç, Pentium 4 işlemci, 593-594 Tam yuvalanmış kesme modu, 243 İşlevsel kodlama, 750  
 Fonksiyonel ortalama, 60  
 Fonksiyonlar, 276-280  
 I/O, 98, 232-234  
 FXU (sabit nokta birimi), 10

**G**

Bosluklar, manyetik diskler, 197 Kapılar, 20, 376-378  
 gecikme, 376  
 işlevsel tam kümeler, 377 NAND, 377  
 NOR, 377-378  
 GeForce 8800 GTX, 693  
 GPU kullanarak genel amaçlı hesaplama (GPGPU), 52-53, 689  
 Genel amaçlı kayıt, 460-462, 466, 491-492, 517-518, 528  
 Geometrik ortalama, 60, 64-67  
 Gigabit Ethernet, 107  
 Küresel geçmiş tamponu (GHB), 599  
 Kademeli alt akış, 367  
 Grafik simbol, 376, 378  
 Grafik işleme birimleri (GPU'lar), 52-53, 689  
 mimariye genel bakış, 692-701  
 yardımcı işlemci olarak, 704-706  
 CUDA çekirdekleri, 696-697  
 çift warp zamanlayıcı, 696-697  
 Fermi, 694  
 saniye başına kayan nokta işlemleri, 693 kayan nokta (FP) birimi işlem hattı, 697 GDDR5 (grafik çift veri hızı), 694-695 Gen8 mimarisi, 701-704 izgara ve blok boyutları, 691  
 donanım bileşenleri eşdeğerlik haritalaması, 691 tamsayı (INT) birimi işlem hattı, 697 yükleme ve depolama birimleri, 697 L1 önbellek, 697-700 bellek hiyerarşisi nitelikleri, 698 bellek türleri, 700-701 çok çekirdekli bilgisayarlar, 667-669

NVIDIA, 693-694  
 performans ve watt başına performans, 692 işlemci çekirdekleri, 690 yazıldıktan sonra okuma (RAW) veri tehlikesi, 701 kayıtlar, 697-700 paylaşılan bellek, 697-700 özel işlev birimleri (SFU), 694, 697 akış çoklu işlemci mimarisi, 695-700 akıslı çoklu işlemciler (SM'ler), 691 vs. CPU, 691-692 Grafik teknolojisi arayüzü (GTI), 704 Koruma bitleri, 362

**H**

Hamming, Richard, 176  
 Hamming kodu, 176  
 Sabit disk, 199, 201  
 Sabit disk sürücüler (HDD'ler), 212, 214 parametreler, 201  
 Sert ariza, 174  
 Donanım önbelgesi tutarlılığı, 623-624  
 Donanım şeffaflığı yaklaşımı, 146 Kablolu uygulama, 724-727  
 kontrol ünitesi girişleri, 725-726  
 kontrol ünitesi mantığı, 726-727  
 Kablolu programlar, 82  
 Harmonik ortalama, 60, 62-64  
 Hash fonksiyonları, 300  
 Hashing teknigi, 301  
 Heterojen Sistem Mimarisi (HSA) Vakfı, 669  
 Onaltilik, 324-326  
 Onaltilik basamaklar, 325  
 Onaltilik gösterim, 324-326  
 Yüksek çözünürlüklü optik diskler (HD DVD), 221-222

Yüksek performanslı bilgi işlem (HPC), 131  
 İabet oranı, 138, 140, 144  
 Yatay kayıp, 632  
 Ana bilgisayar kanal adaptörü (HCA), 269 İnsan tarafından okunabilir cihazlar, 230  
 Hibrit bulut, 646  
 Hibrit diş açma, 663

**I**

IAS bilgisayar, 11  
 aritmetik ve mantık birimi (ALU), 11, 14, 16  
 adreslerin hesaplanması, 17  
 koşullu dallanma talimatı, 17 kontrol ünitesi, 12, 14  
 veri aktarımı, 16  
 yürütme döngüsü, 16  
 getirme döngüsü, 15  
 akış şeması, 15  
 giriş-çıkış (I/O) ekipmanı, 12

- IAS bilgisayarı (*devam*) talimat  
döngüsü, 15  
eğitim grupları, 16-17  
mantıksal kontrol, 13  
, 11, 14-15  
işlem kodu (opcode) talimatı, 14, 16  
kayıtlar, 14-15  
depolama yerleri, 14  
yapısı, 12  
koşulsuz dallanma talimatı, 16 von  
Neumann'in önceki önerisi, 12-14  
IA-64 mimarisи, 492  
IBM 801 sistemi, 558  
IBM 7094, 18, 19  
yapılardırma, 18  
Talimat Yedekleme Kaydi, 18 IBM  
system/360, 22-23  
ALU, 23  
CPU, 23  
üçüncü nesil bilgisayarlar, 22-23 IBM  
370/168, boru hattı akışları, 510 IBM  
360/91, boru hattı akışları, 510 IBM 3033,  
boru hattı akışları, 510  
IBM 3033 mikro komut yürütme, 743,  
754-755  
IBM zEnterprise EC12 G/C kanal  
yolu, 268  
kanallar, 268  
kanal yapısı, 266-268 kanal alt  
sistemleri (CSS), 267 donanım sistem  
alanı (HSA), 267 G/C çerçeveleri-ön  
görünüm, 269  
G/C sistem organizasyonu, 268-270  
mantıksal bölüm, 267  
alt kanal, 268  
sistem yardımcı işlemcisi (SAP), 267 Z  
çerçevesi, 268  
IBM zEnterprise EC12 anabilgisayar bilgisayarı, 9  
önbellek yapısı, 683-684  
gömülü DRAM (eDRAM), 684 çok çipli  
modül (MCM), 682 organizasyon, 682-  
683  
islemci düğüm yapısı, 682 işlemci  
birimi (PU), 683 depolama  
kontrolü (SC), 683  
I-cache, 11  
ICode arayüzü, 38 Tanımlama  
bayrağı (ID), 519  
IDU (komut kod çözme birimi), 10  
If-Then (IT) komutu, 481  
IFU (komut getirme birimi), 9 Anlık  
adres, 459 Anlık adresleme modu, 459  
Anlık sabitler, ARM, 479-480 Artımlı  
ölçeklenebilirlik, 633  
İndeksli adres, 492  
İndeksleme, 462-463  
Dizin kayıtları, 462-463, 492  
Dolaylı adresleme, 459-460  
Dolaylı döngü, 711-712 Dolaylı  
komut döngüsü, 458 InfiniBand,

- JUMP talimatı, 557  
 uzunluk, 469-470  
 bellek aktarım uzunluğu, 469, 470  
 MIPS R4000 mikroişlemci, 560-561 döngü başına çoklu komutlar, 632 NOOP komutu, 557  
 operand adresi, 470  
 Patterson programları, 539  
 PDP-8, 471-472  
 PDP-11, 474  
 PDP-10, 472-473  
 adres aralığı, 471  
 azaltılmış komut kümesi mimarileri, 551-553 kayıt ve bellek adresi, 470-471  
 SETHI komutu, 569 kayıt kümesi, 471  
 SPARC (Ölçeklenebilir İşlemci Mimarisi), 568-569  
 S/390 Karakterleri Taşı (MVC) talimatı, 553  
 32 bit Thumb yönergeleri, 482 değişken uzunluklu yönergeler, 473-477  
 VAX, 474-477, 479, 537, 539, 540  
 Talimat sorunu, 582-583  
 Komut ardışık sıralama, 500-516, 536, 576, 596, 628, 672  
 şube tahmini, 511-515  
 kontrol tehlikesi (branş tehlikesi), 508-509 veri tehlikesi, 508-509  
 koşullu dallarla ilgilenmek, 509-515, 516  
 gecikmeli şube, 515, 557-558  
 gecikmeli yük, 558  
 Intel x86 mimarisi, 593 döngü tamponu, 510-511  
 döngü açma, 559  
 boru hattı performansı ölçümleri, 504-507 MIPS R4000 mikroişlemci, 561-565 çoklu akışlar, 509-510  
 optimizasyonu, 557-559  
 boru hattı kabarcığı, 507 önceden hedeflenmiş dallanma hedefi, 510  
 azaltılmış komut seti bilgisayarı (RISC), 555-559  
 düzenli talimatlarla, 555-556 kaynak tehlikesi, 507-508  
 strateji, 500-504  
 Talimat ön getirme (getirme örtüşmesi), 501 Talimat kaydı (IR), 14, 85, 493, 497, 710  
 Komut kümesi mimarisi (ISA), 2, 58, 278, 482, 667, 671  
 ARM, 35, 481  
 Başparmak-2, 35, 481  
 Komut kümeleri. *Bkz.* Adresleme modları  
 Komut penceresi, 585  
 Tamsayılar, 321-322  
 ekleme, 337-340  
 bölüm, 347-350  
 sabit nokta, 335  
 olumsuzlama, 336-337  
 negatif olmayan, 330, 424  
 taşıma kuralı, 337-338  
 paketlenmiş bayt ve paketlenmiş bayt, 422  
 paketlenmiş çift sözcük ve paketlenmiş çift sözcük, 423  
 paketlenmiş quadword ve paketlenmiş quaudword, 423 paketlenmiş tek hassasiyetli kayan nokta ve paketlenmiş çift hassasiyetli kayan nokta, 423  
 paketlenmiş kelime ve paketlenmiş kelime, 422  
 radix noktası, 330  
 menzil genişletme, 333-335  
 imzalı, 424, 568  
 işaret büyülüğu, 331  
 çıkışma, 337-340  
 ikili çalışması, 331-333, 336, 342-347  
 işaretsiz, 330  
 Entegre devre (IC), 7, 20-22  
 desen, 21  
 Tümleşik bellek denetleyicisi (IMC), 255 Intel önbellek evrimi, 150  
 Intel Core i7-990X, 676-677  
 Intel Core mikro mimarisi, 592  
 Intel 82C55A programlanabilir çevresel arabirim, 245-248  
 Intel 82C59A kesme denetleyicisi, 243-244 Kesme Onaylama (INTA), 243 Kesme İsteği (INTR), 243 sorumluluğu, 243  
 Intel 3420 yonga seti, 9  
 Intel 8085, 720-724  
 Adres Mandali Etkin (ALE) darbe sinyalleri, 723  
 kontrol ünitesi, 721  
 CPU blok diyagramı, 721  
 harici sinyaller, 722  
 artırıcı/azaltıcı adres mandali, 720 kesme kontrolü, 720  
 makine döngüleri, 721  
 OUT talımı, 723-724  
 pin yapılandırması, 723 seri I/O kontrolü, 720  
 Intel 80386  
 kesme modları, 243-244 çoklu I/O modülleri, 243  
 için kullanıcı tarafından görülebilir kayıt organizasyonu, 496 Intel 80486 pipelining koşul kodlarında, 516 kod çözme, 515  
 yürütme döngüsü, 515  
 getirme döngüsü, 515  
 geri yazma aşaması, 516

- Intel 8237A  
 DMA denetleyicisi, 251-253  
 DMA kayıtları, 254
- Intel 8255A programlanabilir çevresel arabirim mimarisi ve çalışması, 245-247 klavye/ekran terminali, 247-248 çalışma modları ve yapılandırımları, 246-247 pin düzeni, 245-246
- Intel Gen8 mimarisi, 701-704 yürütme birimi (EU), 701 bellek modülleri, 703  
 kayıtlar, 701  
 SIMD kayan nokta birimleri, 701 eşzamanlı çoklu iş parçacığı (SMT)  
 mimarlık, 701  
 alt dilimler, 702
- Intel HD Graphics 5300 Gen8, 704
- Intel mikroişlemci  
 Core M İşlemci, 704 mevcut zamanlar, 27  
 1970s, 26  
 1980s, 26  
 1990s, 26
- Intel Quad-Core Xeon işlemci yongaları, 8-9 Intel x86 adresleme modları, 463-466  
 yer değiştirme moduna dayalı ölçeklendirilmiş indeks, 466  
 temel mod, 465  
 yer değiştirme modu ile taban, 465-466  
 indeks ve deplasman modu ile taban, 466  
 deplasman modu, 465  
 acil mod, 464  
 modu hesaplaması, 464 kayıt  
 operandı modu, 464 görelî  
 adresleme, 466  
 yer değiştirme modu ile ölçeklendirilmiş indeks, 466 için segment kayıtları, 464
- Intel x86 mimarisi, 27-29 ayırma aşaması, 595  
 dal tahmin stratejisi, 593-594 dal hedef tamponu (BTB), 593-594 önbellek/bellek parametreleri, 592  
 kontrol kayıtları, 519-521 Core serisi mikroişlemci, 28 veri tehlikeleri, 508  
 veri türleri, 422-423  
 kod çözme birimi, 594-595 sevkiyat, 596  
 8080 mikroişlemci, 28  
 8086 mikroişlemci, 28  
 80286 mikroişlemci, 28  
 80386 mikroişlemci, 28  
 80486 mikroişlemci, 28  
 evrimi, 28  
 ön uçlar, 593-595  
 donanım kayıtları, 595
- talimat getirme birimi, 594 talimat kuyruğu birimi, 594-595 talimat seti, 28 komut çevirisi lookaside tamponu (ITLB), 594  
 tamsayı ve kayan noktalı kayıt dosyaları, 596 kesme işleme, 522-524  
 mikro mimari, 591-596  
 mikro-op kuyruklama, 596  
 mikro-op çizelgeleme, 596  
 sıra dışı yürütme mantığı, 595-596 Pentium serisi mikroişlemci, 28 pipelining, 593  
 ön kod birimi, 594  
 kayıt organizasyonu, 517-522  
 kayıt yeniden adlandırma, 595-596  
 yeniden sıralama arabelleği (ROB) girişî, 595 statik tahmin algoritması, 594  
 Intel x86 komut biçimi, 477-479 adres boyutu, 478  
 yer değiştirme alanı, 478  
 talimat örnekleri, 478  
 ModR/M bayti, 478  
 işlem kodu alanı, 478  
 operand boyutu, 478  
 segment geçersiz kılma, 478  
 SIB bayti, 478
- Intel x86 bellek yönetimi, 304-309 adres alanları, 304-305  
 4 Gbyte doğrusal bellek alanı, 308  
 mantıksal adres, 305  
 İşletim sistemi tasarımı ve uygulaması, 305  
 parametreler, 307  
 ayrıcalık düzeyi ve erişim özneliği, 305  
 istenen ayrıcalık düzeyi (RPL), 306  
 bölümlemiş sayfalanmış bellek, 305  
 bölümlemiş sayfalanmamış bellek, 304  
 bölüm numarası, 306  
 tablo göstergesi (TI), 305  
 bölümlememmiş sayfalanmış bellek, 304 bölümlememmiş sayfalanmamış bellek, 305  
 Intel x86 işlem türleri çağrı/dönüş mekanizması, 438-439  
 bellek yönetimi, 439  
 MMX talimatları, 440-442  
 SIMD talimatları, 440-444  
 durum bayrakları ve koşul kodları, 439-440 Intel x86 işlemci ailesi  
 istisna ve kesme vektör tablosu, 524 istisnalar, 522-523  
 kesme işleme rutini, 523-524  
 kesinti işleme, 522-524  
 kayıt organizasyonu, 517-522 Intel Xeon işlemciler  
 doğrudan önbellek erişim stratejileri, 259

- E5-2600/4600, 255-257  
 çok çekirdekli işlemciler, 255  
     Ara bağlantı yapıları, 99-100  
     otobüs ara bağlantısı, 100-102  
     noktadan noktaya, 100, 102-107
- Arabağlı transferleri**  
 Bellekten G/C'ye veya bellekten G/C'ye, 100 G/C'den işlemciye, 100 bellekten işlemciye, 100  
     işlemciden G/C'ye, 100  
     işlemciden belleğe, 100  
 Aralıklı bellek, 173-174  
 Aralıklı çok iş parçacıklı skaler, 630-631 Aralıklı çok iş parçacıklı, 629-630 Aralıklı çok iş parçacıklı superscalar, 632 Aralıklı çok iş parçacıklı VLIW, 632 Ara kuyruklar, 293  
 Dahili işlemci veri yolu, 490-491  
 Uluslararası Referans Alfabesi (IRA), 232, 421  
 Nesnelerin interneti (IoT), 30-31  
 Kayıtlar arası boşluklar, 222  
 Kesme güdümlü G/C, 239-248  
     otobüs tahkimi, 243  
     papatya zinciri yapılandırması, 243  
     tasarım sorunları, 241-243  
 Intel 82C59A kesme denetleyicisi, 243-244 kesme işleyici programı, 241  
 kesme işleme, 239-241 çoklu  
 kesme hatları, 242 yazılım  
 yoklaması, 242  
 TESTI/O, 242  
     vektörülü kesme, 243 Kesme etkinleştirme bayrağı (IF), 518  
 Kesmeler, 89-98  
     siniflar, 89  
     kontrol hatları, 102  
     döngü, 92, 498, 712  
     engelli, 95  
     işleyici, 91, 93  
     komut döngüsü ve, 91-96 çoklu, 95-98  
     iç içe kesme işleme, 97 kullanıcı programının bakış açısı, 92  
     olmadan ve birlikte program kontrol akışı, 90 sıralı kesme işleme, 97  
 Kesme hizmeti rutini (ISR), 95 İzler arası boşluklar, 197  
 Ters çevrilmiş sayfa tablosu, 301  
 G/C adres kaydı (IOAR), 83 G/C tampon (I/OBR) kaydı, 83 G/C kanalları, 261-263  
     mimari, 263  
     blok çoklayıcı, 262  
     bayt çoklayıcı, 262  
     özellikleri, 262  
     fonksiyon, 261-262  
     IBM zEnterprise EC12 G/C, 268-270  
     çoklayıcı kanalı, 262  
     seçici kanal, 262  
 G/C komutu, 235  
 I/O bileşenleri, 83  
 G/C kontrolörleri, 108, 121, 235, 236, 262  
 G/C cihazları  
     disk sürücüsü, 232  
     harici, 230-232  
     insan tarafından okunabilir cihazlar, 230  
     klavye/monitör düzenlemesi, 231-232  
     makine tarafından okunabilir aygıtlar, 230 G/C sürücü yazılımı, 214  
 G/C işlevleri, 98  
 G/C hub'ı (IOH), 103-104  
 I/O-bellek aktarımı, 98  
 I/O modülleri, 715  
     adres tanıma, 233  
     komut kod çözme, 233  
     kontrol işlevi, 231, 232-233  
     kontrol hatları, 101-102  
     veri arabelleğe alma, 233  
     veri aktarımı, 231  
     cihaz iletişim, 231, 233  
     hata tespiti, 234  
     sahada programlanabilir mantık dizisi, 408  
     işlevler veya gereksinimler, 232-234 ara bağlantı yapıları, 100  
 arabağlı transferleri, 100  
 arayüzüne, 231  
 RAID şemalarında G/C istekleri, 211 makine talimatları, 414  
 PCIe TLP işlem türleri, 113 işlemci iletişim, 233  
 QPI bağlantıları, 103-104  
 okuma-yazma işlemi, 145-146  
 kayıtlar ve, 494  
 yarı iletken bellek, 173  
 durum raporlaması, 233  
 yapı, 234-235  
 zamanlama, 203, 232-233  
 dönüştürücü, rolü, 231  
 G/C ayrıcalık bayrağı (IOPL), 519  
 G/C işlemcisi, 235  
 I/O programının yürütülmesi, 89, 91 için  
     gerekен süre, 94  
 IPC (döngü başına talimat), 628 İzole G/C, 237  
 ISU (talimat dizisi birimi), 9

**J**

- Java uygulamaları, 662 Java Sanal Makinesi, 662  
 JEDEC Katı Hal Teknoloji Birliği, 183

J-K flip-flop, 399-400, 402-403  
 İş kontrol dili (JCL), 282 İş programı, 280-282  
 Atlama talimatı, 433

**K**  
 Karnaugh haritaları, 381-386  
 Çekirdek (nucleus), 279  
 Khronos Group'un OpenCL'i, 689  
*K-yolu* küme ilişkisel önbellek organizasyonu, 140-142

**L**  
 Araziler, kompakt diskler, 218  
 Lane, 104  
 Büyük ölçekli entegrasyon (LSI), 24  
 Son giren ilk çıkar (LIFO) kuyruğu, 463  
 L1 önbellek, 128  
 L3 önbellek, 128  
 L2 önbellek, 128  
 L2 kontrolü, 11  
 En az sıklıkta kullanılan (LFU) algoritma, 132, 145  
 En son kullanılan (LRU) algoritması, 132, 145, 299  
 En küçük anlamlı rakam, 319  
 Doğrusal teyp-çıkar (LTO) sistemi, 224  
 Doğrusal teyp-çıkar (LTO) teyp sürücüler, 224  
 Bağlama, 40, 281, 474  
 Bağlı katmanı, 105-107, 115  
 Bağlantılar, InfiniBand, 265  
 Linux, 18  
 Little endian sıralaması, 455 Little yasası, 55-56  
 Yük dengeleme, kümeler, 636 Yük/depo adresleme, ARM, 466-468 Yük/depo çoklu adresleme, ARM, 468-469  
 Referans yeri, 125, 128, 158  
 Yerel değişken, 437  
 Kilitli çalışma, 113  
 Mantıksal adres, 297  
 Mantıksal önbellek, 132  
 Mantıksal veri işlenenleri, 421-422  
 Mantıksal işlemler (opcode), 429  
 Mantıksal kaydırma, 430  
 Mantık bloğu, 406, 408  
 Mantık (Boolean) talimatları, 417  
 Mantık-bellek performans dengesi, 48-50 Uzun vadeli veri depolama işlevi, 4  
 Uzun vadeli programlama, 287-288  
 Arama tablosu, 408  
 Döngü tamponu, ardışık sıralama, 510-511 Döngü açma, ardışık sıralama, 559  
 Düşük volajlı diferansiyel sinyalleşme (LVDS), 105  
 LSI-11 mikro komut yürütme, 751-754 LSU (yük depolama birimi), 10

**M**

Makine döngüler, 721  
 Makine talimatları. Ayrıca *bz.* Talimat döngüsü; Talimat formatları adresleri, 417-419  
 aritmetik talimatlar, 416  
 ARM mimarisi, 417  
 BASIC talimi, 416  
 dallanma talimatları, 433-434 koşullu dallanma talimi, 433 dönüştürme talimatları, 432  
 veri aktarım talimatları, 427-428 unsurları, 413-414  
 yüksek seviyeli dil, 416 sıfır ise artır ve atla (ISZ) talimi, 434 giriş/çıkış talimatları, 432 talimat kaydi (IR), 415 talimat seti tasarımları, 419 talimatları, 416 mantık (Boolean) talimatları, 416 bellek talimatları, 416 MMX talimatları, 440-442 çoklu adres komutları, 419 sonraki komut referansı, 414 işlenenler, 420-422 işlemler (opcode), 413 indirgenmiş komut kümel bilgisayar (RISC), 419 sonuç operand referansı, 414 SETEND komutu, 425 atlama talimatları, 434 kaynak ve sonuç, 414 kaynak operand referansı, 413 yiğinlar ve, 418 sembolik gösterim, 415 sistem kontrol talimatları, 432 test talimatları, 416 kontrol transferi talimatları, 433-438 koşulsuz dallanma talimi, 434 sıfır-adres talimatları, 418  
 Makine paralelligi, 581-582, 588-589  
 Makine tarafından okunabilir cihazlar, 230  
 Manyetik çekirdekli bellek, 24  
 Manyetik disk  
 erişim süresi, 201  
 çağdaş katı, 196  
 silindir, 200  
 veri düzenlemeye ve biçimlendirme, 196-199 çift taraflı diskler, 200  
 hatlar arası boşluklar, 197  
 çoklu tabak, 200  
 çoklu bölge kaydi (MZR), 198 performans parametreleri, 201-203  
 fiziksel özellikler, 199-201 okuma ve yazma mekanizmaları, 195-196

- rotasyonel gecikme, 201-202  
 rotasyonel konumsal algılama (RPS), 202  
 sektörler, 197  
 arama süresi, 201, 202  
 sıralı organizasyon, 203  
 tek taraflı diskler, 200  
 zamanlama, 203  
 transfer süresi, 202  
**Manyetik RAM (MRAM)**, 189 Manyetik bantlar, 222-224  
 Manyetik tünelleme bağlantısı (MTJ), 189  
 Manyetorezistif (MR) sensör, 196 Mainframe bilgisayarlar, 23, 47  
 Mantissa, 351  
 Birçok entegre çekirdek (MIC), 52 Ön belleğin eşleme işlevi, 133-144  
 çağrımsal, 138-140  
 doğrudan, 134-138  
*k-yollu* küme ilişkisel önbellek organizasyonu, 140-142  
 set-associative, 140-144  
**Maske**, 430  
 Orta vadeli çizelgeleme, 288 Bellek, 83.  
*Ayrıca bkz.* önbellek  
 erişim süresi (gecikme), 123 ilişkisel erişim, 123  
 yardımcı, 127  
 banka, 173  
 kapasite, 121  
 özellikler, 121-127 dahili kavramlar, 121-122 kontrol hatları, 102  
 döngü süresi, 123  
 doğrudan erişim, 122-123  
 hiyerarşi, 124-127  
 talimatlar, 416  
 ara bağlantı yapıları, 99-100  
 arabağlantı transferleri, 100  
 seviyeleri, 125  
 referans yeri, 125  
 birimlere erişim yöntemi, 122-123 "doğal" organizasyon birimi, 122 önbelleğe alınamaz, 146  
 PCIe TLP işlem türleri, 113 performans parametreleri, 123 fiziksel türleri, 123 rastgele erişim, 123  
 salt okunur bellek (ROM), 124 gerçek, 300  
 ikincil, 127  
 sıralı erişim, 122  
 aktarım hızı, 123  
 iki seviyeli, 157-164  
**Bellek adres kaydı (MAR)**, 14, 83, 493-494, 497, 709-710  
**Hafıza bankası**, 173  
**Bellek tampon kaydı (MBR)**, 14, 83, 493-494, 497, 499, 709-710, 712-713  
**Hafıza hücresi**, 20  
**Bellek denetleyici hub'ı (MCH)**, 255-256 Bellek döngü süresi, 18, 123  
 Bellek hiyerarşisi, 124-127  
 Bellek talimatları, 416 Bellek yönetimi  
 ARM, 309-314  
 temel adresler, 297  
 sıkıştırma, 296  
 Intel x86, 304-309  
 ara kuyruk, 293  
 mantıksal adresler, 297  
 sayfa çerçeveleri, 297  
 sayfa tablosu, 298  
 çağrı, 297, 308-309  
 bölümleme, 294-297  
 fiziksel adresler, 297  
 segmentasyon, 303-306  
 SMP, 621  
 takas, 293-294  
 zaman alıcı prosedür, 296 çeviri lookaside tamponu (TLB), 301-303  
 sanal bellek, 299-301  
**Bellek yönetim birimi (MMU)**, 35, 132, 310, 458  
 Cortex-A ve Cortex-A50, 35  
 Cortex-R, 35  
 Bellek eşlemeli G/C, 237-238  
 Bellek modülleri, 83, 84, 99 Bellek koruması, OS, 289 Bellek Koruma Birimi (MPU), 35  
 MESI (değiştirilmiş/özel/paylaşılan/geçersiz) protokolü, 621-627  
 hat devletleri, 625  
 L1-L2 önbellek tutarlılığı, 627  
 okuma isabeti, 626  
 bayan oku, 626  
 değiştirme niyetiyle okuma (RWITM), 626  
 durum geçiş diyagramı, 625  
 yazma vuruşu, 627  
 bayan yazmak, 626-627  
 Metalizasyon, 20  
 Mikrobilgisayarlar, 3, 24  
 Mikrodenetleyici çipi, 32-33  
 Mikroelektronik, 19-23  
 kontrol ünitesi, 20  
 veri hareketi, 20  
 veri işleme, 20  
 veri depolama, 20  
 gelişimi, 20-22 Mikro talimat veri yolu (MIB), 751 Mikro talimat spektrumu, 747

- Mikro-işlemler (mikro-ops), 152, 708-714  
 yürütme döngüsü, 712-713  
 getirme döngüsü, 709-711  
 dolaylı döngü, 711-712  
 talimat döngüsü, 713-714  
 talimat seti, 715  
 kesme döngüsü, 712  
 kurallar, 711  
 sıralama, 715  
 zaman birimleri, 711
- Mikroişlemci çipleri, 32
- Mikroişlemci kayıt organizasyonları, 495-496
- Mikroişlemciler, 25-26
- Mikro programlı kontrol üniteleri, 536, 733-735
- Mikro programlı uygulama, 6
- Mikroprogramlama, 727, 730  
 adres oluşturma teknikleri, 743-744 avantajları, 737  
 tasarım hususları, 739-740  
 dezavantajları, 737  
 kodlama, 748-751  
 yürütme, 745-755  
 sert, 748  
 yatay, 748  
 kesinti testi, 744  
 LSI-11 mikro komut yürütme, 751-754 LSI-11  
 mikro komut sıralama, 744-745 mikro komutlar,  
 730-733 mikro programlanmış kontrol birimi, 733-  
 735 sonraki sıralı adres, 744  
 işlem kodu eşleme, 744  
 dizileme teknikleri, 740-742  
 yumuşak, 748  
 alt rutin tesisi, 744  
 taksonomi, 745-748  
 dikey, 748  
 Wilkes kontrolü, 735-739
- Mikro programlama dili, 731
- Göç hatları, 681
- Sanyiede milyonlarca kayan nokta işlemi (MFLOPS) oranı, 59
- Saniye başına milyonlarca talimat (MIPS) oranı, 59
- Minuend, 338
- MIPS oranı, 59
- MIPS R4000 mikroişlemci, 559-565 boru hattının iyileştirilmesi, 563  
 yüklemelerin ve depolamaların yürütülmesi, 565 komut seti, 560-561  
 çipin bölümlenmesi, 560 pipelining  
 talimatları, 561-565
- Bayan, 126, 138
- MMX (çoklu ortam görevi) talimatları,  
 440-442, 444, 521  
 kayıtlar, 521-522
- Anımsatıcılar, 415, 761
- Monitör (basit toplu işletim sistemi), 281 Monitör düzenlemesi, G/C, 231 Monitör Yardımcı İşlemcisi (MP), 520 Moore, Gordon, 21  
 Moore yasası, 21, 47, 51, 692  
 sonuçları, 21-22 En önemli basamak, 319  
 Anakart, 7
- Motorola MC68000 mikroişlemci kayıtları, 495  
 Hareketli kafa diski, 199
- Çok çekirdekli bilgisayarlar, 6-8  
 aritmetik ve mantık birimi (ALU), 8  
 önbellek tutarlılığı, 674-675  
 ön bellek, 6  
 merkezi işlem birimi (CPU), 6, 667-671  
 çekirdek, 6  
 sayısal sinyal işlemcileri (DSP'ler), 669-671 eşdeğer komut seti mimarileri, 671-674  
 harici bellek arayüzü (EMIF), 671 grafik işleme birimleri (GPU'lar), 667-669 donanım performansı, 657-660 heterojen çok çekirdekli organizasyon, 667-675  
 homojen çok çekirdekli organizasyon, 667  
 talimat mantığı, 8  
 önbellek seviyeleri, 665-666  
 yükleme/depolama mantığı, 8  
 bellek alt sistemi bellek denetleyicisi (MSMC), 675  
 MOESI modeli, 675  
 anakart, 7-8  
 çok çekirdekli paylaşılan bellek (MSM), 671 çok çekirdekli paylaşılan bellek denetleyicisi (MSMC), 671  
 organizasyon, 665-667  
 güç tüketimi, 659-660 baskılı devre kartı (PCB), 7 işlemci, 6, 8  
 ana unsurlarının basitleştirilmiş görünümü, 7  
 yazılım performansı, 660-665  
 Çok çekirdekli işlemciler, 6, 8, 657  
 Çok bölmeli dağıtım, 105 Çok düzeyli önbellek, 147-149 Çok bitli toplayıcılar, 394-395  
 Çoklu komut, çoklu veri (MIMD) akışı, 615  
 Çoklu komut, tek veri (MISD) akışı, 615  
 Çoklu kesme hatları, G/C, 242 Çoklu paralel işleme, 628 Çoklu plakalar, manyetik diskler, 200 Çoklu akışlar, pipelining, 509-510 Çoklayıcılar, 388-390  
 sinyal ve veri yönlendirmesini kontrol etmek için dijital devrelerde, 389

- 4'e 1, 388-389  
 VE, VEYA ve DEĞİL kapılarını kullanma, 389  
**C**oklayıcı, 18  
 Multiplexor kanalı, 262  
 Çoklu bölge kaydı (MZR), 198, 218  
 Çarpım, 340-341 Çarpma  
     aritmetik kaydırma, 345  
     Booth algoritması, 344-347 işaretsiz ikili  
     için akış şeması, 342  
     i işaretsiz ikilinin donanım uygulaması, 341  
     ikili tamamlayıcı, 342-347  
     i işaretsiz tamsayılar, 330 Çarpan  
 bölümü (MQ), 14  
 Çok işlemcili işletim sistemi tasarımı, SMP ile ilgili  
     hususlar, 284  
 Multithreading, 628-633  
     engellendi, 630-632  
     iri taneli, 630  
     ince taneli, 629  
     örtük ve açık, 628-629  
     serpiştirilmiş, 629-632  
     temel yaklaşımlar, 629-630  
     süreç, 628-629  
     süreç anahtarları, 629  
     kaynak sahipliği, 628  
     programlama/yürütmeye, 628  
     eszamanlı (SMT), 630, 632-633, 667  
     iplik, 629  
     iplik anahtarları, 629
- N**  
 NAND flaş bellek, 186-187, 188, 214  
 NAND geçidi, 377, 388  
 NaN'ler, IEEE standartları, 365-366  
 N-disk dizisi, 212  
 Negasyon, tam sayılar, 336-337  
 Negatif taşıma, 353  
 Negatif alt akış, 353 İç İş Görev  
 (NT) bayrağı, 519  
 Yuvalanmış vektör kesme denetleyicisi (NVIC), 36  
 Neumann, John von, 11, 81  
 Nibble, 324  
 NIST SP-800-145, 39  
 NIST SP 500-292 (*NIST Bulut Bilişim Referans Mimarisi*), 647-648  
 Önbellege alınamayan bellek yaklaşımı, 146 Yedekli  
 olmayan disk performansı (RAID düzeyi  
     0), 205  
 Çıkarılamayan disk, 199  
 Tekdüze olmayan bellek erişimli (NUMA) makineler,  
     614, 615, 640-643  
     avantajlar ve dezavantajlar, 643  
     motivasyon, 640-641  
     organizasyon, 641-642  
     düğüm 2'deki işlemci 3 (P2-3) istekleri, 642  
     Geçici olmayan bellek, 124, 127  
     Uçucu olmayan RAM teknolojileri, 188, 190  
     NOR flaş bellek, 186-188  
     NOR kapısı, 377  
     Normalleştirilmiş sayılar, 67  
     NOR S-R mandali, 398  
     NOT işlemi, 429  
     Üzerinden Yazılmayan (NW), 521 Sayı  
     sistemi  
         temel rakam, 319  
         ikili sistem, 321  
         ikili ve ondalık arasında dönüştürme, 321-324  
         ondalık sistem, 319-320  
         kesirler, 322-324  
         onaltılık gösterim, 324-326  
         tamsayılar, 321-322  
         en az anlamlı basamak, 319  
         en anlamlı basamak, 319  
         nibble, 324  
         konumsal sayı sistemi, 320 radiks  
         noktası, 320  
 Sayısal Hata (NE), 521  
 NVIDIA'nın CUDA'sı, 689
- O**  
 Ofset adresleme, ARM, 467 Omnibus, 24  
 Operandlar, 420-422  
     bit odaklı görünüm, 422  
     karakterler, 421  
     mantıksal veri, 421-422  
     sayılar, 420-421  
     paketlenmiş ondalık, 420-421  
 İşletim sistemi (OS), 494  
     toplu iş sistemi, 280  
     fonksiyonlar, 276-280  
     interaktif, 280  
     kesme güdümlü G/C veya DMA işlemleri, 285-  
         286  
     kesintiler, 283  
     bellek yönetimi, 286  
     bellek koruması, 283  
     Multics İşletim Sistemi, 287  
     çoklu programlama toplu işi, 283-286  
     hedefler, 276-280  
     ayrılaklı talimatlar, 283  
     programlama, 280, 287-293  
     kurulum süresi, 280  
     basit parti, 281-283  
     simetrik çoklu işlemciler (SMP'ler), 617-621  
     zamanlayıcı, 283  
     zaman paylaşımı, 286-287  
     türleri, 280-287  
     tek programlama, 286

- Operasyonel teknoloji (OT), 31  
 Operasyonlar (opcode), 425-438  
     VE, 430  
     aritmetik kaydırma işlemi, 431 ARM mimarisini, 444-445  
     temel aritmetik, 429  
     ortak komut kümesi, 426-427  
     dönüştürme, 432  
     veri aktarımı, 427-428  
     IBM EAS/390 veri aktarım işlemleri, 428  
     giriş/çıkış, 432  
     Intel x86 işlem türleri, 438-444  
     mantıksal, 429-431  
     iç içe prosedürler, 435, 436  
     NOT, 429  
     prosedür çağrıları talimatları, 435-438 çeşitli süreç eylemleri, 427 yeniden girişimi prosedür, 436  
     döndürme veya döngüsel kaydırma, 430-431 yoğun çerçevesi, 437  
     sistem kontrolü, 432  
     kontrolün devri, 433-438  
     XOR, 430
  - Optik bellek, 195
  - özellikler, 222
  - kompakt disk (CD), 217-220 yüksek çözünürlüklü optik diskler, 221**VEYA** kapısı, 376  
 Orijinal ekipman üreticileri (OEM'ler), 24  
 Ortogonalite, 472-473  
 Sıra dışı yürütme, 595-596  
 Sıra dışı sorun, 585-586  
 Çıktı bağımlılığı, 509, 579, 583  
 Taşma, 337
- P**
- Paketlenmiş ondalık gösterim, 421  
 Paketler, veri, 109  
 Sayfa hatası, 299  
 Sayfa çerçevesi, 297  
 Sayfa düzeyi önbellek devre dışı bırakma (PCD), 521 Sayfa düzeyi şeffaf yazma (PWT) biti
  - kontroller, 521
  - Sayfa değiştirme, 300
  - Sayfa sayısı, 297
  - Sayfa tabloları, 298, 300-301
  - Çağrı, 297-298, 303-304, 521
    - talep, 299-300
    - sanal bellek, 158
    - x86, 308-309, 463
 Paralellik, 576
  - uygulama düzeyinde, 662
  - temel sınırlamalar, 579-581 talimat düzeyi, 579, 581-582  
 makine düzeyinde, 581-582, 588-589  
 çok çekirdekli bilgisayarlar, 657-659
 prosedürel bağımlılık ve, 581 süreç düzeyinde, 662  
 kaynak çatışması ve, 581 iş paracığı seviyesi, 662  
 gerçek veri bağımlılığı ve, 579-581  
 Paralelleştirilmiş uygulama, 637  
 Paralelleştirici derleyici, 637  
 Paralel organizasyonlar, 615-617 Paralel işleme
  - önbellek tutarlılığı, 621-624
  - çip çoklu işlem, 630
  - bulut bilişim, 643-649
  - kümeler, 633-639
  - MESI (değiştirilmiş/özel/paylaşilan/geçersiz) protokolü, 624-627
  - çoklu komut, çoklu veri (MIMD) akışı, 615, 617
  - çoklu komut, tek veri (MISD) akışı, 615
  - çoklu işlemci organizasyonları, 615-617
  - multithreading, 628-633
  - tekdüze olmayan bellek erişimi (NUMA), 640-643
  - tek komut, çoklu veri (SIMD) akışı, 615, 617
  - tek komut, tek veri (SISD) akışı, 615
  - simetrik çoklu işlemciler (SMP), 617-621 yazma ilkeleri, 622
 Paralel kayıt, 222  
 Paralel kayıt, 401  
 Parametreler, manyetik diskler, 201-203  
 Parametrik hesaplama, 637  
 Eşlik bitleri, 176  
 Kısmi ürün, 341  
 Kısmi kalan, 347-349  
 Bölümleme, G/C bellek yönetimi, 294-297 Pascal, 159  
 Pasif beklemeye kümeleme yöntemi, 635 Patterson programları, 539  
 PCI Express (PCIe), 104, 107-115, 214, 265, 704
  - adres alanları ve işlem türleri, 113-114 veri bağlantı katmanı paketleri, 115
  - uygulanan cihazlar, 108-109 G/C cihazı veya denetleyici, 108
  - I/O çekmeceleri, 270
  - eski uç nokta kategorisi, 109 çok şeritli dağıtım, 110 sıralı set bloğu, 111 fiziksel katman, 109-111
  - protokol mimarisini, 109
  - kök kompleksi, 108
  - TLP paket montajı, 114-115 işlem katmanı (TL), 112-115 işlem katmanı paket işleme, 115
  - Tip 0 ve Tip 1 yapılandırma döngüleri, 114

- PCI Özel İlgi Grubu (SIG), 107 PC-göreceli adresleme, 461  
 PDP-8 Veri Yolu Yapısı, ana bellek, 24  
 PDP-8 komut formatı, 471-472  
 PDP-8 komut biçim tasarımı, 471-472 PDP-11 komut biçim tasarımı, 474 PDP-11 işlemci, 87  
 PDP-10 komut biçim, 472-473 PDP-10 komut biçim tasarımı,  
     472-473  
 Pentium 4 önbelleği, 149-152  
     yürütme birimi, 152  
     getirme/kod çözme birimi, 150  
     talimat önbelleği, 152  
     bellek alt sistemi, 152  
     çalışma modları, 152  
     sira dışı yürütme mantığı, 150 geri yazma politikası, 152  
 Çevresel bileşen ara bağlantısı (PCI),  
     107  
 Çevresel (harici) aygıtlar, G/Ç, 233 Kişisel teknoloji, 31  
 Faz değişim diskı, 220  
 Faz değişimli RAM (PCRAM), 188, 189 SET ve RESET işlemi, 189  
 Phit (fiziksel birim), 104 Fiziksel adres, 297  
 Fiziksel önbellek, 133  
 Veri depolamanın fiziksel özellikleri, 124 Fiziksel katman, 104-105  
 Fiziksel kayıtlar, 222  
 Fiziksel bellek türleri, 123  
 Pipelining, 47. Ayrıca bkz. Komut ardışık sıralama  
 Çukur, 218  
 Hizmet olarak platform (PaaS), 41, 646  
 Plakalar, 195, 200  
 Noktadan noktaya ara bağlantı yapıları, 24, 100,  
     102-107  
     QPI bağlantı katmanı, 105-107  
     QPI fiziksel katmanı, 104-105  
     QPI protokol katmanı, 107 QPI  
         yönlendirme katmanı, 107  
 Noktadan noktaya arayızler, 255  
 Polarizasyon-akım kaynaklı mıknatışlanma  
     anahtarlaması, 189  
 Pollack'ın kuralı, 660  
 Polikarbonatlar, 217  
 POP yoğun işlemi, 469, 474  
 Konumsal sayı sistemi, 320 Pozitif taşıma, 353  
 Pozitif düşük akış, 353  
 Postindexing, 462-463  
 Güç tüketimi, 659-660  
 Güç yoğunluğu, 50  
 Ön endeksleme, 463  
 Baskılı devre kartı (PCB), 7  
 Yazıcılar, 230  
 Özel bulut, 646  
 Ayrıcalıklı talimatlar, 283  
 Prosedürel bağımlılık, paralellik, 581 Prosedür dönüşü, 438  
 Süreç  
     blok, 107, 489, 560  
     kontrol bloğu, 289  
     multithreading, 628-633  
     kaynak sahipliği, 628  
     programlama, 287-293  
     devletler, 288-290  
     anahtar, 629  
 Süreç düzeyinde paralellik, 662  
 İşlemci organizasyonu, 489-491 ortak alanlar veya bayraklar, 494 işlevsel unsurları, 715 gereksinimler, 489  
 İşlemciler  
     ara bağlantı yapıları, 100  
     araağlığı transferleri, 100  
     1970s, 26  
     1980s, 26  
     1990s, 26  
     şimdiki zamanlar, 27  
     Toplamların ürünü (POS), 380  
     Program sayacı (PC), 14, 84, 289, 389-390, 493,  
         497, 499, 710  
     Program yürütme örneği, 86  
         yürütme döngüsü, 84  
         getirme döngüsü, 84, 87  
         getirilen talimat, yürütülmesi, 85 talimat  
             döngüsü, 84, 85, 87  
         kesintiler, 89-98  
         I/O programı, 89, 91  
     Programlanabilir dizi mantığı (PAL), 406  
     Programlanabilir mantık dizisi (PLA), 405-406  
     Programlanabilir mantık cihazı (PLD),  
         405-409  
         karmaşık PLD'ler (CPLD'ler), 408  
         sahada programlanabilir mantık dizisi, 406-409  
         programlanabilir mantık dizisi (PLA), 405-406 basit  
         PLD (SPLD), 406  
         terminoloji, 406  
     Programlanabilir salt okunur bellek (PROM),  
         169, 170  
     Programlanmış G/Ç  
         komutları, 236, 238  
         talimatlar, 236-238  
         kesme gündemli G/Ç ve, 239-248 yalıtılmış,  
             237  
         bellek eşlemeli, 237  
         bellek eşlemeli G/Ç, 237-238  
         genel bakış, 236  
         teknikler, 235

Programlama, 83  
 Program durum sözcüğü (PSW), 494  
 Koruma Etkinleştirme (PE), 520  
 Sözde komut, 483  
 Genel bulut, 646  
 İtme listesi, 463

**Q**  
 Kuyruklar, 55  
 G/C işlemleri, 267  
 QuickPath Interconnect (QPI), 102-107  
 dengeli iletişim, 105  
 diferansiyel sinyalizasyon, 105  
 doğrudan bağlantılar, 103 hata  
 kontrol işlevi, 106 akış kontrol  
 işlevi, 106  
 katmanlı protokol mimarisi, 103 çoklu  
 doğrudan bağlantılar, 103 paketlenmiş  
 veri aktarımı, 103 fiziksel Arayüz, 105  
 QPI bağlantı katmanı, 105-107  
 QPI fiziksel katmanı, 104-105  
 QPI protokol katmanı, 107 QPI  
 yönlendirme katmanı, 107  
 çok çekirdekli bilgisayarda kullanım,  
 103 Sessiz NaN, 365-366  
 Quine-McCluskey yöntemi, 384-388

**R**  
 Radix noktası, 320, 330  
 RAID (Redundant Array of Independent Disks),  
 195, 204-213  
 karşılaştırma, 213  
 RAID seviye 5, 212  
 RAID seviye 4, 211-212  
 RAID seviye 1, 209-210  
 RAID seviye 6, 212  
 RAID seviye 3, 210-211  
 RAID seviye 2, 210  
 RAID seviye 0, 205-209  
 Rastgele erişim, 123  
 Rastgele erişimli bellek (RAM), 167 Oran  
 metrik ölçümleri, 71, 73  
 Okuma işaret/hatalı, 626  
 Okuma mekanizmaları, manyetik diskler,  
 196 Çokunlukla okunan bellek, 170  
 Salt okunur bellek (ROM), 124, 169-170, 392 doğruluk  
 tablosu, 393  
 Değiştirme niyetiyle okuma (RWITM), 626  
 Okuma-yazma bağımlılığı, 509  
 Gerçek hafıza, 300  
 Kaydedilebilir (CD-R), 219  
 Azaltılmış komut seti bilgisayarı (RISC), 3, 27, 536  
 mimari, 549-555  
 Berkeley çalışması, 541-542, 565

önbellek, 545-546  
 özellikleri, 538  
 klasik, 553-555  
 derleyici tabanlı kayıt optimizasyonu, 547-549 karmaşık  
 komut setleri, 537  
 koşullu ifadeler, 539 tasarım  
 unsurları, 537 global  
 değişkenler, 545  
 yüksek seviyeli dil (HLL) ve, 537, 539-542,  
 545  
 komut yürütme, 537-542 büyük  
 kayıt dosyası, 545-546  
 akıl yürütme çizgisi, 538  
 makine döngüsü başına bir makine talimatı, 551  
 işlenenler, 540-541  
 operasyonlar, 539-540  
 pipelining, 555-559  
 prosedür çağrıları, 541  
 nitel değerlendirme, 570-571  
 nice değerlendirme, 570-571 yerel bir skalere  
 referans verme, 546-547 register'dan register'a  
 işlemler, 551-552 register pencereleri, 543-  
 545  
 basit adresleme modları, 552 basit  
 komut formatları, 552  
 CISC tasarımına karşı, 553-555, 570-571  
 pencere tabanlı kayıt dosyası, 546-547 Hamming  
 aracılığıyla yedek disk performansı  
 kodu (RAID seviye 2), 210  
 Reentrant prosedürü, 436  
 Kayıt adresleme, 460-461, 551-552 Kayıt  
 dosyası, komut boru hattı, 542-547 Kayıt dolaylı  
 adresleme, 461  
 Kayıt organizasyonu, 491-496  
 Kayıt yeniden adlandırma, 586-587  
 Kayıtlar, 401-402, 490  
 adres, 492  
 ARM, 527-529  
 kontrol ve durum, 491, 493-495, 518, 519-521 G/C  
 işlemlerinin kontrolünde, 494  
 mevcut program durum kaydı (CPSR), 527-529  
 veri, 491  
 kayan nokta birimine ayrılmış, 518  
 EFLAGS ve RFLAGS, 518-519  
 genel amaçlı, 491-492, 517-518, 528 grafik  
 işlemci birimi (GPU), 697-700 dizin, 492  
 komut kaydı (IR), 493 komut  
 seti tasarımı, 419 Intel x86, 517-  
 524  
 bellek adres kaydı (MAR), 493-494, 497  
 bellek tampon kaydı (MBR), 493-494, 497,  
 499

- mikroişlemci kayıt organizasyonları, 495-496  
 MMX, 521-522  
 sayısal, 518  
 program durumu, 527, 528  
 azaltılmış komut seti bilgisayarı (RISC), 543-545, 551-552  
 kaydedilmiş program durum kaydı (SPSR), 527-529  
 segment, 518  
 16 bit veri, 496  
 yazılım tarafından görülebilir, 527  
 etiketler, 518  
 Texas Instruments 8800 Yazılım Geliştirme Kartı (SDB), 759  
 kullanıcı tarafından görülebilir, 491-493, 496  
 Kayıtta kayda organizasyon, 551  
 Kayıt penceresi, 543-545  
 Göreceli adres, 298  
 Göreceli adresleme, 461  
 Çıkarılabilir disk, 199  
 Değiştirme algoritmaları, ön bellek, 145 Yerleşik monitör, 281  
 Rezistif-kapasitif (RC) gecikme, 50 Rezistif RAM (ReRAM), 188, 189 Kaynak çatışması, paralellik, 581 Kaynak kodlaması, 750  
 Kaynak tehlikesi (yapısal tehlike), pipelining, 507-508  
 Kaynak sahipliği süreci, 628 Devam bayrağı (RF), 519  
 Emekli, 598-600  
 Dalgalanma sayıçları, 402-403  
 Kök kompleksi, 108  
 Döndürme (döngüsel kaydırma) işlemi, 431  
 Dönen kesme modu, 244  
 Dönme gecikmesi (latency), manyetik diskler, 201  
 Dönme pozisyonu algılama (RPS), 202 Yuvarlama, 364-365  
 Yuvarlama, IEEE standartları, 364 RU (kurtarma birimi), 10
- S**
- Doygunluk aritmetiği, 441  
 Skaler değerler, 452  
 Çizelgeleme, 287-293  
 G/Ç kuyruğu, 292  
 uzun vadeli, 287-288  
 uzun vadeli kuyruk, 292  
 orta vadeli, 288  
 süreç kontrol bloğu, 289-290 süreç durumları, 288-290  
 işlemcinin kuyruk diyagramı gösterimi, 292  
 kısa vadeli, 288-293  
 kısa vadeli kuyruk, 292
- teknikler, 290-293  
 zaman paylaşım sistemi, 288 İlkinci (yardımcı) bellek, 127 İlkinci nesil bilgisayarlar, 17-18  
 CPU, 18  
 veri kanalı, 18  
 çoklayıcı programları, 18  
 Sektörler, manyetik diskler, 197  
 Arama süresi, manyetik diskler, 202  
 Segmentasyon, Pentium II işlemci, 303-304 Segment işaretçileri, 492  
 Seçici kanal, 262  
 Anlamsal boşluk, 537  
 Yarı iletken bellek, 24-25, 167, 174  
 adres hatları, 171  
 dizideki hücrelerin düzenlenmesi, 170 çip mantığı, 170-172  
 çip paketleme, 172-173  
 dinamik RAM (DRAM), 167-168  
 elektriksel olarak silinebilir programlanabilir salt okunur bellek (EEPROM), 170  
 silinebilir programlanabilir salt okunur bellek (EPROM), 170  
 hata düzeltme, 174-180 flash bellek, 170  
 serpiştirilmiş bellek, 173-174  
 G/C modülü, 173  
 organizasyon, 166  
 programlanabilir ROM (PROM), 169, 170  
 rastgele erişimli bellek (RAM), 167 en çok okunan bellek, 170  
 salt okunur bellek (ROM), 169-170 SRAM ve DRAM, 169  
 statik RAM (SRAM), 168-169  
 hız, yoğunluk ve maliyet arasındaki dengeler, 170  
 türleri, 167  
 yazma etkinleştirme (WE) ve çıkış etkinleştirme (OE) pinleri, 172, 173  
 Yarı iletkenler, 127, 185, 214  
 Sensör/aktüatör teknolojisi, 31  
 Sıralama, 739-745  
 Siralı erişim, 122  
 Siralı erişim cihazı, 223  
 Siralı devreler, 396-405  
 sayıçalar, 402-405  
 parmak arası terlik, 396-400  
 kayıtlar, 401-402  
 Sıralı organizasyon, manyetik diskler, 203 Seri ATA (SATA) soketleri, 9  
 Serial ATA (Serial Advanced Technology Attachment), 265  
 Seri kayıt, 222  
 Serpentine kaydı, 222  
 Sunucu kümelleme yaklaşımları, 635  
 Set-associative mapping, 140-144

Kurulum süresi, işletim sistemi (OS) verimliliği, 280-281  
 Shannon, Claude, 373  
 Kaydirmalı kaydediciler, 401-402  
 Kısa vadeli veri depolama işlevi, 4 Kısa vadeli zamanlama, 288-290  
 NaN sinyali, 365-366  
 Sinyal hatları, PCI, 99  
 İşaret biti, 331  
 İşaret uzatma, 334  
 Significand, 359  
     taşma, 359  
     alt akış, 359  
 İşaret-magnitüd gösterimi, 331  
 Silikon, 20  
 Basit PLD (SPLD), 406  
 Eşzamanlı çoklu iş parçacığı (SMT), 630, 667 Tek hata düzeltme, çift hata algılama (SEC-DED) kodu, 179-180  
 Tek hata düzeltme (SEC) kodu, 179  
 Tek komut, çoklu veri (SIMD) akışı, 615  
 Tek komut, tek veri (SISD) akışı, 615  
 Tek büyük pahalı disk (SLEP), 204 Tek işlemcili bilgisayar, 4-6  
     aritmetik ve mantık birimi (ALU), 6  
     merkezi işlem birimi (CPU), 4 iç yapısı, 9  
 ana bellek, 4  
 işlemci, 4  
 kayıtlar, 6  
     sistem veri yolu, 5  
     sistem ara bağlantısı, 5  
 Tek taraflı diskler, 200  
 Tek sistemli görüntü, 637  
 Tek iş parçaklı skaler, 630-631  
 Atlama talimatları, 434  
 Küçük Bilgisayar Sistemi Arayüzü (SCSI), 264  
 Küçük ölçekli entegrasyon (SSI), 21, 405  
 Snoop kontrol ünitesi (SCU), 677  
 Snoopy protokolleri, önbellek tutarlılığı, 623-624  
 Yumuşak hatalar, 175  
 Yazılım, 18, 83  
     önbellek tutarlılığı çözümleri, 622-623 dinamik voltaj ve frekans ölçeklendirme  
     (DVFS), 673-674  
 G/C sürücüsü, 214  
 çok çekirdekli bilgisayar performansı, 660-665  
 anket, 242  
 işleme modelleri, 673-674  
 hizmet olarak yazılım (SaaS) modeli, 40-41 Valve oyun iş parçacığı, 663-665  
 Hizmet olarak yazılım (SaaS), 40-41, 645 Yazılım yoklama tekniği, G/C, 242  
 Katı hal bileşeni, 17, 20  
 Katı hal sürücülerı (SSD'ler), 17, 187, 212-216

HDD ile karşılaşıldığında, 214  
 organizasyon, 214-216  
 pratik sorunlar, 216  
 SPARC (Ölçüklenebilir İşlemci Mimarisi), 542  
 adresleme modları, 568  
 ALU işlemleri, 567  
 şube talimatı, 568-569  
 geçerli pencere işaretçisi (CWP), 566 bir işlenenin etkin adresi (EA), 568 komut biçimi, 568-570  
 talimat seti, 567-568  
 işlemci durum kaydı (PSR), 566 kayıt seti, 565-567  
 kayıt penceresi düzeni, 566 Sun SPARC, 452  
 UltraSPARC, 71, 300  
 pencere geçersiz maskesi (WIM), 566  
 Uzamsal yerellik, 159, 160  
 SPEC dokümantasyon  
     temel metriği, 71  
     kiyaslama, 71  
     tepe metriği, 71  
     oran metriği, 71  
     referans makinesi, 71  
     hız ölçütü, 71 test edilen sistem, 71  
 Özel ilgi grubu (SIG), PCI, 107 Özel maske kesme modu, 244 Spekulatif yürütme, 48  
 Hız metrik ölçümleri, 71  
 Hızlanma faktörü, 506-507  
 Sistemin hızlandırılması, 53-55, 660-661  
 Spin-transfer tork RAM (STT-RAM), 188, 189  
     Bölünmüş  
     önbellek, 149  
     bellek, 147  
 S-R Mandali, 396-398  
 Yiğin adresleme, 463  
 Yiğin işaretçisi, 492  
 Standart Performans Değerlendirme Şirketi (SPEC)  
     kiyaslamaları  
     kayan nokta kıyaslamaları, 70  
     tamsayı ölçütleri, 69, 72  
 SPEC CPU2006, 68-71  
 SPECint\_base2006, 72  
 SPECint\_rate\_base2006, 72  
 SPECint\_rate2006, 72  
 SPECint2006, 72  
 SPECjbb2013 (Java Business Benchmark), 68 SPECjvm2008, 68  
 SPECfsfs2008, 68  
 SPECviewperf, 68  
 SPECvirt\_sc2013, 68  
 SPECwpc, 68  
 Durum diyagramları, talimat döngüleri, 414  
 Bir sürecin durumu, 288-290  
 Statik RAM (SRAM), 36, 38, 148, 168-169, 187

- Durum bayrakları, 439  
 Durum kayıtları, 493-495  
 Durum sinyalleri, G/C, 231-232  
 Saklanan program kavramı, 11  
 Akış çoklu işlemcileri (SM'ler), 691 Şerit, 205, 211, 212  
 Çizgili veriler, 211  
 Şeritli disk performansı (RAID seviye 0), 205-209  
 Yapılandırılmış programlama (SAL), 159  
 Normal altı sayılar, 366-367  
 Substrat, 195  
 Çıkarma, 337-340  
 kural, 338  
 ikili tamamlayıcı, 338-339  
 Subrahend, 338  
 Ürünlerin toplamı (SOP), 379 Superpipelined yaklaşım, 578-579  
 Superpipelined işlemci, 578-579  
 Superscalar, 9, 28, 51, 149, 474, 632  
 şube tahmini, 589  
 öğretimin taahhüt edilmesi veya geri çekilmesi, 590  
 bağımlılık, 579-581  
 infaz, 48  
 programların yürütülmesi, 589-590 uygulama, 590  
 in-order tamamlama, 583 komut  
 getirme aşaması, 589 komut verme  
 politikası, 582-586  
 komut düzeyinde paralellik, 581-582 makine  
 paralelliği, 581-582, 588-589  
 organizasyon, 577  
 sırada tamamlama, 583-586  
 genel bakış, 576-581  
 pipelining ve çizelgeleme teknikleri, 152, 507  
 işlemciler, özellikler, 538  
 kayıt yeniden adlandırma, 586-587  
 bildirilen hızlanmalar, 577  
 sıralama türleri, 582  
 superpipelining'e karşı, 578-579  
 SuperSpeed, 264  
 Değiştirme, G/C bellek yönetimi, 293-294 Anahtar, 108  
 Simetrik çoklu işlemciler (SMP'ler), 614, 615, 617-621  
 kullanılabilirlik, 618  
 otobüs organizasyonu, 620  
 özellikleri, 617  
 DMA aktarımları, 619  
 çoklu işlemcilerin varlığı, 618 artan büyümeye, 618  
 bellek ve G/C kanalları, 619 bellek  
 yönetimi, 621 işletim sistemi, 617-621  
 performans, 617  
 güvenilirlik ve hata toleransı, 621  
 ölçeklendirme, 618  
 programlama, 621  
 eşzamanlı eşzamanlı süreçler, 621 senkronizasyon, 621  
 SYNCH baytı, 199  
 Senkron sayaç, 403-405 Senkron DRAM (SDRAM), 181-182  
 DDR SDRAM, 183-184  
 Sendrom kelimeleri, 176  
 Sistem otobüsleri, 5, 101  
 Sistem kontrol işlemleri, 432 Sistem ara bağlantısı (veri yolu), 5  
 Sistem Performansı Değerlendirme Şirketi (SPEC), 68. Ayrıca bkz SPEC belgeleri  
 Sistem yazılımı, 17
- T**
- Etiketler, ön bellek, 140 Görev Değiştirme (TS), 520 Zamansal yerellik, 159-160  
 Test talimatları, 416  
 Texas Instruments (TI) K2H SoC platformu, 669-670  
 Texas Instruments 8800 Yazılım Geliştirme Kartı (SDB), 755-765  
 blok diyagramı, 756  
 bileşenler, 756  
 kontrol işlemleri, 757  
 sayaçlar, 759  
 dış çevre, 762-763  
 mikro talimat formatı, 757-758  
 mikrosequencer, 757-762  
 mikro sıralayıcı mikro komut bitleri, 761 kayıtlı ALU, 762-765  
 kayıtlı ALU komut alanı, 764-765 kayıtlar, 759  
 yoğun işlemleri, 759-760  
 alt alanlar, 760, 761  
 Üçüncü nesil bilgisayarlar, 18-24 DEC P DP- 8, 23-24  
 IBM sistem/360, 22-23  
 mikroelektronik, 19-22  
 32-bit Thumb talimatları, 482 Thrashing, 138, 299  
 İplik, 629, 690  
 İplik blokları, 690  
 İş parçacığı ayrıntı düzeyi, 663  
 İş parçacığı stratejisi  
 kaba taneli, 663  
 ince taneli, 663  
 hibrit, 663  
 eşzamanlı çoklu iş parçacığı (SMT), 667 Valve oyun iş parçacığı, 663-665  
 İş parçacığı düzeyinde paralellik, 662  
 Verim, 71  
 Thumb komut seti, ARM, 479-481  
 Thunderbolt, 263, 265  
 Zaman paylaşımı işletim sistemleri (OS), 296-297

**Zamanlama**

- I/O modülleri, 203, 232-233
- manyetik disk, 203
- bellek sisteminin talimatlar üzerindeki etkileri, 601 TinyOS, 31
- TLP paket montajı, 114-115 Veri alanı, 115
  - uçtan uca CRC alanı, 115 Başlık alanı, 115
- Parçalar, manyetik diskler, 196-197
- İşlem katmanı, 112-114
- Transdüber, I/O, 231
- Kontrol işlemlerinin aktarımı, 433
- Aktarım oranı, 123
- Aktarım süresi, manyetik diskler, 201-202
- Transistörler, 17-18
- Ceviri görünüm arabellegi (TLB), 301-303
- Tuzak bayrağı (TF), 518
- Gerçek veri (akış) bağımlılığı, paralellik, 579-581
- Doğruluk tablosu, 374, 378,
  - 403
  - ikili ekleme, 394
  - salt okunur bellek (ROM) için, 393 64-bit, 393
- Turing, Alan, 11
- İki seviyeli önbellek, 157-164
  - özellikleri, 158
  - yerellik, 158-160
  - işleyisi, 160-161
  - performans parametreleri, 161-164 üst düzey dinamik frekans
  - dil işlemleri, 159
- Tamsayıların ikiye tümleyen işlemi, 331-333, 336, 342-347

**U**

- Ultra Enterprise 2, 71
- Ultra büyük ölçekli entegrasyon (ULSI), 24
- UltraSPARC II işlemci, 71
- Tekli operatör, 417
- Koşulsuz dallanma talimatları, 482 Koşulsuz atlama, 754
- Alt akış, 353, 358 Birleşik önbellek, 149
- Tek tip bellek erişimi (UMA), 640 Tek işlemciler, 615-617, 619, 621 Tek programlama, işletim sistemleri
  - (OS), 280
  - Transfer birimi, 122
- Evrensel Seri Veri Yolu (USB), 263-264
- Yukarı doğru uyumlu, 496
- Bit kullanın, 146
- Kullanıcı/bilgisayar arayüzü, OS, 276-278
- Kullanıcı tarafından görülebilen kayıtlar, 491-493
- Yardımcı Programlar, İşletim Sistemi, 277
- Yardımcı program, 277

**V**

- Vakum tüpleri, gelişimi, 11-17 Valf oyunu dış açma, 663-665
- Değişken uzunluklu komut formatları, 473-477
- Değişken boyutlu bölümler, 295-296
- VAX mimarisi, 300
- VAX komut biçimi tasarımları, 474-477, 479, 537
- Vector, 243
- Vektör kayan nokta (VFP) birimi, 603 Dikey kayıp, 632
- Çok büyük ölçekli tümleştirme (VLSI), 24 Çok uzun komut sözüğü (VLIW), 632 Video görüntü terminaleri (VDT'ler), 230 Sanal adres alanları, 305
- Sanal önbellek, 132 Sanal Kesme Bayrağı (VIF), 519 Sanal bellek, 299-301, 494
  - talep çağrıları, 299
  - sayfa hatası, 299
  - sayfa değiştirme, 299
  - sayfa tablosu, 300-301
  - dayak, 299
- Sanal Mod (VM) biti, 519
- Üçüncü bellek, 32, 124
- Von Neumann mimarisi, 81-82 Von Neumann makineleri, 13

**W**

- Wafer, silikon, 21
- Warps, 696
- Watchdog, 680
- Wi-Fi, 266
- Wilkes kontrolü, 735-739, 746
- Winchester disk biçimi, 199
- Windows, 18
- Kelimeler, 14
  - hafıza, 85, 101, 167, 174, 495
  - paketlenmiş, 441
- Okuma sonrası yazma (WAR) bağımlılığı, 509
- Yazma sonrası yazma (WAW) bağımlılığı, 509
- Geri yazma tekniği, 132, 146, 260, 516, 562, 565
- Yazma işaret/yanlış, 627
- Yazma mekanizmaları, manyetik diskler, 195-196
- Yazma politikası, ön bellek, 145-147
- Yazma Koruması (WP), 521
- Teknik aracılığıyla yazma, 145, 260, 622
- Yazma-güncelleme protokolü, 624

**X**

- X86 ve ARM veri türleri, 422-425 Xeon E5-2600/4600, 255-257
- XOR işlemleri, 430 XU (ceviri birimi), 10

**Z**

- Sıfır adres talimatları, 418
- Bölgeler, tanımlanmış, 198

# KREDILER

---

Sayfa 4: "Dikkat çekici bir şekilde ... tasarım sırasında değil" Siewiorek, D., Bell, C. ve Newell, A. Computer Structures: İlkeler ve Örnekler. New York: McGraw-Hill, 1982.

pp. 12-13: "2.2 İlk: Cihaz öncelikle bir bilgisayar olduğu için ..... Görtüleceği üzere, yine en iyisi M'den (O tarafından) R'ye tüm transferleri yapın ve asla doğrudan C'den yapmayın" Von Neumann, J. First Draft of a Report on the EDVAC. Moore Okulu, Pennsylvania Üniversitesi, 1945.

p. 39: Alıntı: The NIST Definition of Cloud Computing (Bulut Bilişimin NIST Tanımı) (42 kelime). Grance, T., ve Mell, P. "Bulut Bilişimin NIST Tanımı." NIST SP-800-145. Ulusal Standart ve Teknoloji Enstitüsü.

p. 57: Şekil 2.5: Sistem Saati. Resim The Computer Language Company Inc. izniyle, www .computerlanguage.com

p. 269: Şekil 7.20: IBM zEC12 G/C Çerçevevleri-Önden Görünüm IBM, İzinle Yeniden Basılmıştır. IBM zEnterprise EC12 Teknik Kılavuzu, SG24-8049. <http://www.redbooks.ibm.com/abstracts/sg248049.html>

p. 540: Tablo 15.2: Patterson, D. ve Dequin, C. "A VLSI RISC "e dayalı olarak HLL İşlemlerinin Ağırıklı Göreli Dinamik Frekansı. Computer, Eylül 1982.

p. 634: Şekil 17.8: Buyya, R. High Performance Cluster Computing'e dayanan Küme Yapılandırmaları: Mimariler ve Sistemler. Upper Saddle River, NJ: Prentice Hall, 1999.

p. 638: "Aşağıdakileri arzu edilen küme ara katman hizmetleri ve işlevleri olarak listeler ..... " temel alınarak Hwang, K., vd. "Hiyerarşik Kontrol Noktası ve Tek G/C Alanı ile SSI Kümeleri Tasarlama." IEEE Concurrency, Ocak-Mart 1999.

p. 652: Tablo 17.3: S/390 SMP Yapılandırmasında Tipik Önbellek İabet Oranı. MAK97.

p. 670: Şekil 18.8: Texas Instruments 66AK2H12 Heterojen Çok Çekirdekli Çip. Texas Instruments'ın izniyle.

p. 693: Şekil 19.3: CPU ve GPU için Samiye Başına Kayan Nokta İşlemleri. Resim NVIDIA Corporation'ın izniyle.

p. 695: Şekil 19.5: Tek SM Mimarisi. Resim NVIDIA Corporation'ın izniyle.

p. 703: Şekil 19.11: Intel Gen8 Dilimi Intel Corp. Intel Processor Graphics Gen8'in Bilgisayar Mimarisi. Intel Beyaz Kitap, Eylül 2014.

*Bu sayfa kasıtlı olarak boş bırakılmıştır*

# öğrenciler için dijital kaynaklar

Yeni ders kitabınız, VideoNotes (programlama kavramları hakkında adım adım video eğitimleri), kaynak kod, web bölümleri, sınavlar ve daha fazlasını içerebilen dijital kaynaklara 12 aylık erişim sağlar. Kaynakların ayrıntılı bir listesi için ders kitabındaki önsöze bakın.

Stallings'in Bilgisayar Organizasyonu ve Mimarisi, Onuncu Baskı için Yardımcı Web Sitesine kaydolmak için aşağıdaki talimatları izleyin.

1. [www.pearsonhighered.com/cs-resources](http://www.pearsonhighered.com/cs-resources) adresine gidin
2. Ders kitabınızın başlığını girin veya yazar adına göre göz atın.
3. Companion Web Sitesine tıklayın.
4. Kaydol'a tıklayın ve bir oturum açma adı ve parola oluşturmak için ekrandaki talimatları izleyin.

**Kaplamayı kazımak ve erişim kodunuzu ortaya çıkarmak için bir bozuk para kullanın.  
Koda zarar verebileceğinden keskin bir bıçak veya başka bir keskin nesne kullanmayın.**

---

Ders kitabına eşlik eden dijital kaynakları kullanmaya başlamak için kayıt sırasında oluşturduğunuz oturum açma adınızı ve parolayı kullanın.

## Önemli:

Bu erişim kodu yalnızca bir kez kullanılabilir. Bu abonelik etkinleştirildikten sonra 12 ay boyunca geçerlidir ve devredilemez. Erişim kodu zaten açıklanmışsa artık geçerli olmayabilir. Bu durumda, Companion Web sitesinin giriş sayfasından bir abonelik satın alabilirsiniz.

Teknik destek için <http://247pearsoned.custhelp.com> adresine gidin.

*Bu sayfa kasıtlı olarak boş bırakılmıştır*

# AKRONİMLER

ACM	Association for Computing Machinery Aritmetik
ALU	Mantık Birimi
ANSI	Amerikan Ulusal Standartlar Enstitüsü
ASCII	Amerikan Standartları Kodu 1'veya Bilgi Değişimi İkili Kodlu
BCD	Ondalık
CD	Kompakt Disk
CD-ROM	Compaci Disk Salt Okunur Bellek
CISC CPU	Karmaşık Komut Seti Bilgisayarı
DRAM	Merkezi İşlem Birimi
DMA DVD	Dinamik Rastgele Erişimli Bellek
<b>EPROM</b>	Doğrudan Bellek Erişimi
EPIC	Dijital Çok Yönlü Disk
EPROM	Elektrikle Silinebilir Programlanabilir Bellek Salt Okunur Bellek
HLL	Açık Paralel Komut Hesaplama
I/O	Rrasable Programlanabilir Salt Okunur Bellek
1AR	Yüksek Seviye L'unpuage
IC	giriş/çıkış pm
<b>IEEE</b>	Talimat Adres Kaydı Entegre Devresi
ILP IR	Elektrik ve Elektronik Mühendisleri Enstitüsü Talimat
LRU	Seviyesi Paralelliği
LSI	Talimat Kaydi En Son
MAR	Kullanılan Büyük Ölçekli
MBR	Entegrasyon
MESI	Bellek Adres Bellek Tampon
M IC	Kaydedicisi
M MU	Modify-Exclusive-Shared-Invalid
MSI	Many Integrated Core
NUMA	Bellek Yönetimi Birliği Orta
<b>İŞLETİ</b>	Ölçekli Bütünleşik Düzgün
M	Olmayan Bellek Erişimi İşletim
SISTEM	Sistemi
I	Program Sayacı İşlem
PC PCB	Konroiol Blogu
PCI	Pcripheral Componcnt Inlerconnect
PROM	Programmable Read-Only Memory
PSW	Processor Status Word
RAID	Bağımsız Disklerin Yedekli Dizisi
RALU	Regislr/Aritmetik-Mantık Birimi Rastgele
RAM	Erişimli Bellek
RISC	Azaltılmış Talimat Ser Bilgisayar Salt
ROM	Okunur Bellek
SCSi	Küçük Bilgisayar Sistem Arayüzü
SM P	Simetrik M uliprocessors
SRAM	Slatic Random-Access Memory
SS I	Küçük Ölçekli Entegrasyon
VLSI	Ultra Büyük Ölçekli Entegrasyon
<b>VLIW</b>	Çok Büyük Ölçekli lnsruction
VU	Word Çok Büyük Ölçekli Entegrasyon

# **WILLIAM STALLINGS'İN BİLGİSAYAR KİTAPLARI**

## **VERİ VE BİLGİSAYAR İLETİŞİMİ, ONUNCU BASKI**

(1) iletim, medya, sinyal kodlama, bağlantı kontrolü ve çoğullama dahil olmak üzere veri iletişimini; 2) devre ve paket anahtarlamalı, çerçeve rölesi, ATM ve LAN'lar dahil olmak üzere iletişim ağları; (3) IPv6, TCP, MIME ve HTTP dahil olmak üzere TCP/IP protokol paketinin yanı sıra ağ güvenliğinin ayrıntılı bir şekilde ele alınmasını kapsayan, alanında standart haline gelen kapsamlı bir araştırma. **2007 Metin ve Akademik Yazarlar Birliği (TAA) tarafından yılın en iyi Bilgisayar Bilimleri ve Mühendisliği Ders Kitabı ödülünü almıştır.**

## **KABLOSUZ İLETİŞİM AĞLARI VE SİSTEMLERİ** **(Cory Beard ile)**

Kapsamlı, son teknoloji ürünü bir araştırma. Antenler ve yayılım, sinyal kodlama teknikleri, yayılmış spektrum ve hata düzeltme dahil olmak üzere temel kablosuz iletişim konularını kapsar. Bluetooth ve 802.11 dahil olmak üzere uydu, hücresel, kablosuz yerel döngü ağları ve kablosuz LAN'ları inceler. Kablosuz mobil ağları ve uygulamaları kapsar.

## **BİLGİSAYAR GÜVENLİĞİ, ÜÇÜNCÜ BASKI (Lawrie Brown ile birlikte)**

Algoritmalar, protokoller ve uygulamalar dahil olmak üzere bilgisayar güvenliği teknolojisinin kapsamlı bir şekilde ele alınması. Kriptografi, kimlik doğrulama, erişim kontrolü, veritabanı güvenliği, bulut güvenliği, izinsiz giriş tespiti ve önleme, kötü amaçlı yazılımlar, inkâr hizmet, güvenlik duvarları, yazılım güvenliği, fiziksel güvenlik, insan faktörleri, denetim, yasal ve etik yönler ve güvenilir sistemler. **2008 yılında TAA tarafından yılın en iyi Bilgisayar Bilimleri ve Mühendisliği Ders Kitabı ödülünü almıştır.**

## **İŞLETİM SİSTEMLERİ, SEKİZİNCİ BASKI**

İşletim sistemi prensipleri üzerine son teknoloji bir inceleme. Temel teknolojilerin yanı sıra iş parçacıkları, SMP'ler, çok çekirdekli, gerçek zamanlı sistemler, çok işlemcili zamanlama, gömülü işletim sistemleri, dağıtık sistemler, kümeler, güvenlik ve nesne yönelimli tasarım gibi çağdaş konularını kapsar. **Üçüncü, dördüncü ve altıncı baskaları TAA tarafından yılın en iyi Bilgisayar Bilimleri ve Mühendisliği Ders Kitabı ödülünü almıştır.**

## **KRIPTOGRAFİ VE AĞ GÜVENLİĞİ, ALTINCI BASKI**

Ağ güvenliği teknolojisi üzerine bir eğitim ve araştırma. Geleneksel ve açık anahtarlı kriptografi, kimlik doğrulama ve dijital imzalar dahil olmak üzere ağ güvenliğinin temel yapı taşlarının her biri ele alınmaktadır. Ağ güvenliği için kapsamlı bir matematiksel altyapı sağlar.

## ***VE VERİ İLETİŞİM TEKNOLOJİSİ***

AES ve RSA gibi algoritmalar. Kitap, S/MIME, IP Güvenliği, Kerberos, SSL/TLS, ağ erişim kontrolü ve Wi-Fi güvenliği gibi önemli ağ araçlarını ve uygulamalarını kapsamaktadır. Ayrıca, bilgisayar korsanlarına ve virüslere karşı koyma yöntemleri

keşfedildi. **İkinci baskı, 1999 yılının en iyi Bilgisayar Bilimleri ve Mühendisliği Ders Kitabı** da **TAA ödülünü almıştır.**

### ***AĞ GÜVENLİĞİ TEMELLERİ, BEŞİNCİ BASKI***

Ağ güvenliği teknolojisi üzerine bir eğitim ve araştırma. Kitap, S/MIME, IP Güvenliği, Kerberos, SSL/TLS, ağ erişim kontrolü ve Wi-Fi güvenliği gibi önemli ağ güvenliği araçlarını ve uygulamalarını kapsamaktadır. Ayrıca, bilgisayar korsanları ve virüslere karşı koyma yöntemleri de incelemektedir.

### ***İŞ VERİ İLETİŞİMİ, YEDİNCİ BASKI (Tom Case ile birlikte)***

İş perspektifinden veri iletişimi ve telekomünikasyonun kapsamlı bir sunumu. Ses, veri, görüntü ve video iletişim ve uygulama teknolojilerini kapsar ve bir dizi vaka çalışması içerir. İşlenen konular arasında veri iletişimi, TCP/IP, bulut bilişim, İnternet protokollerinin ve uygulamaları, LAN'lar ve WAN'lar, ağ güvenliği ve ağ yönetimi yer almaktadır.

### ***SDN VE QOE ÇERÇEVESİ İLE MODERN AĞ OLUŞTURMA***

Modern ağ teknolojisi ve uygulamalarının kapsamlı ve birleşik bir incelemesi. Yazılım tanımlı ağlar, OpenFlow ve Ağ Fonksiyonu Sanallaştırma (NVF) gibi temel altyapı teknolojilerini, Hizmet Kalitesi (QoS) ve Deneyim Kalitesi sağlama yaratıcı temel araçları ve bulut bilişim ve büyük veri gibi uygulamaları kapsar.

### ***INTERNET PROTOKOLLERİ VE TEKNOLOJİSİ İLE BİLGİSAYAR AĞLARI***

İnternet tabanlı protokoller ve algoritmalar alanındaki gelişmelerin güncel bir incelemesi. Yukarıda aşağıya bir yaklaşım kullanan bu kitap, uygulamalar, taşıma katmanı, İnternet QoS, İnternet yönlendirme, veri bağlantı katmanı ve bilgisayar ağları, güvenlik ve ağ yönetimi konularını kapsamaktadır.