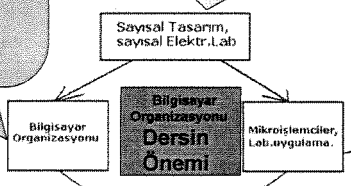
BMÜ-202
Bilgisayar Organizasyonu

2010-2011 Bahar Dönemi

Ders 1: Giriş

Bir bilgisayarı oluşturan temel birimler ve bunların birlikte kullanılabilmesi için kullanıcı-makine etkileşimi ***(üst seviye programlama dili - makine dili)*** seviyesinde tasarım adımları . Bunun için organizasyon aşamalarının öğrenilmesi.
" Bilgisayarlar nasıl çalışır?" sorusuna cevap arar.

Sayısal sistemlerin temel yapı taşları ***Bağlaçlar ve FF'ların*** çalışması ve elektronik yapıları, bunlarla sayısal devre tasarımı, Ardışıl devreler. ***(Toplayıcı, Mux'lar, Sayıcılar , S.register v.b)*** fonksiyonel devrelerin yapıları. Sayısal bigisayarların bu birimlerden oluşumunun mantığının kavranması.



Bilgisayarı kendi diliyle programlayarak, donanımsal olarak istenilen kontrol işaretlerinin gerçek zamanda oluşturulması v.b gerekli ve faydalı uygulamaları yapabilmek.

Bilgisayar mı tasarlayacağız ki?
Belki !!!!! AMA.....
1- Eğer gömülü sistemler üzerinde çalışacaksanız.
2-İşletim sistemleri , Derleyici, aygıt sürücüleri v.b. Sistem yazılımlarını performanslı bir şekilde gerçekleştirecekseniz
3- Yazılımlarınızdan iyi performans istiyorsanız.
Bu durumların hepsinde, donanımı anlamanız gerekir!

Yazılımsal ve donanımsal olarak optimize bir şekilde çalışabilecek bir bilgisayarın tasarlanması için gereken aşamaların (yüksek seviye dil-assembler dili- makine dili (komutun binary kelime karşılığı) ve komutları işleyen ünitelerin kontrolu) öğrenilmesi ve tasarımının yapılması.
"Bilgisayar nasıl tasarlanır?" sorusunu cevaplar.

2

Neler anlatılacak?

Hafta	İçeriği
1	Giriş, bilgisayar kuramsalları ve teknolojisi (Bölüm-1)
2	Komutlar: Makine dili (Bölüm-2 ve Ek A)
3	Komutlar: Makine dili (Bölüm-2 ve Ek A)
4	Bilgisayar aritmetiği (Bölüm-3 ve Ek B)
5	Bilgisayar aritmetiği (Bölüm-3 ve Ek B)
6	Bilgisayar performansını anlamak ve değerlendirmek (Bölüm 4)
7	1. Arasınav
8	İşlemci: Veriyolu ve kontrol (Bölüm 5 and Ek C)
9	İşlemci: Veriyolu ve Kontrol (Bölüm 5 and Ek C)
10	Boru hattı ile performans iyileştirilmesi (Bölüm 6)
11	Hafıza hiyerarşisi (Bölüm-7)
12	Hafıza hiyerarşisi (Bölüm-7)
13	Veri depolama, ağlar ve diğer çevrebirimler (Bölüm-8)
15	uygulamalar

Kaynak :Computer Organization and Design (David A.Patterson)
Yardımcı kaynak : Bilgisayar Sistemleri Mimarisi (M. Morris Mano)

3

Kazancınız ne olacak?

Bahsedilen konuları başarılı bir şekilde anlatabilirsek ve sizlerde öğrenmek için gerekli çabayı gösterirseniz; elde edeceğiniz bilgi kazanımları aşağıda sıralanmıştır.

–Komut kümesi mimarisi tasarımı ve uygulaması

–Bilgisayar aritmetiğinin öğrenilmesi (fix point ve floating point sayı sistemleri)

–İşlemci uygulamalarında veri yolu ve kontrol mekanizmaları

–İşlemci uygulama alternatifleri (tek çevrim, çok çevrim ve boru hattı uygulamaları)

–Bellek hiyerarşisi tasarımı

–Bilgisayar performansının bellek-hız değerlendirilmesinin yapılması

–Çok işlemcili aritmetik algoritmalar ve gerçek zamanlı planlama

–Önbellek tasarımı

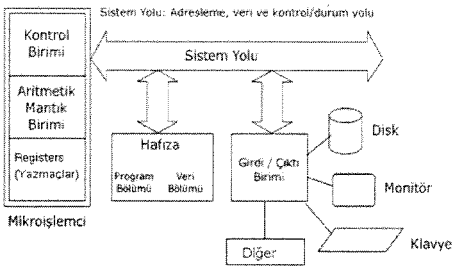
–Sanal bellek

–Bellek arabirim ve çevrebirimleri

4

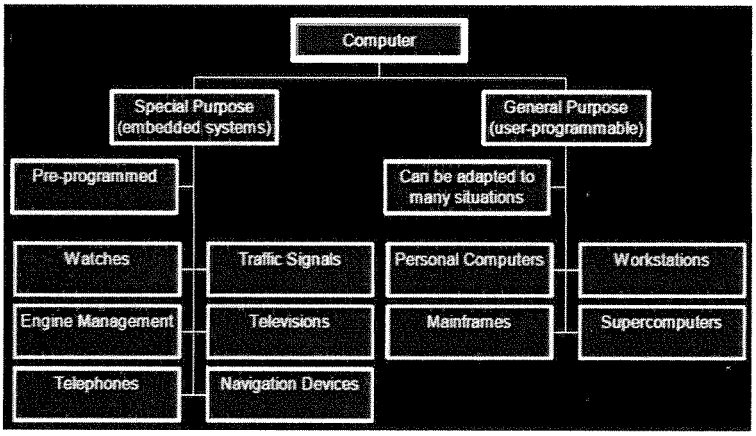
Bilgisayar nedir? Nerelerde kullanılır?

" Yüksek hızda aritmetik ve mantıksal işlemleri ard arda yapabilen ve karar verebilme yetisine sahip hafızalı bir elektronik cihazdır "



5

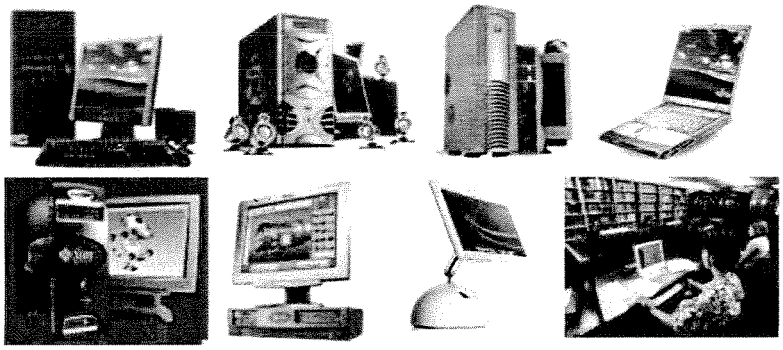
Bilgisayar nedir? Nerelerde kullanılır?



6

Kişisel Bilgisayarlar

- Düşük maliyetli, tek bir kullanıcı için iyi bir performans sunan kişisel bilgisayarlar
- Masaüstü bilgisayarlar sadece son 30 yıllık bir geçmişe rağmen, günümüzde ,bilgisayar pazarlarındaki en büyük paya sahiptir.
- Kişisel bilgisayarlarda;
  - Microprocessor
  - Memory - Synchronous DRAM
  - Hard disk(s), CDROM/DVD, etc.
  - I/O - mouse, klavye, video kart, monitor, network kartı, v.b)

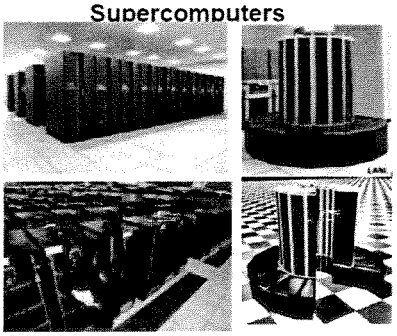


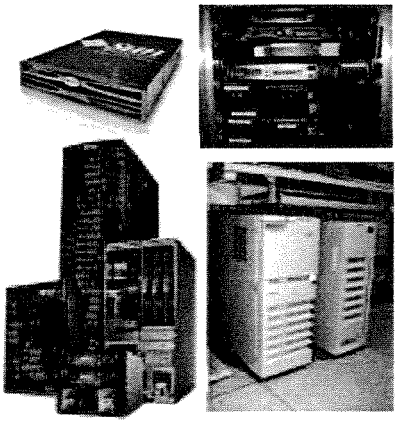
7

Sunucular

- Büyük iş yüklerini taşıyan sunucularda
- Tek karmaşık uygulamalar (bilimsel hesaplama veya mühendislik uygulaması gibi) ya da çok küçük işler (web sunucusu gibi).
- Sunucularda;
  - Microprocessor(s)
  - Hard disks
  - Network Interface(s)

Supercomputers





Gömülü sistemler

- Kullanıcının hiç ummadığı cihaz ve birimlerde, bunları yönetmek için.
- Aşağı yukarı her cihaz ve aletin içine girmiştir.
- Gömülü sistemlerde
  - -Microcontroller / Microprocessor
  - -Hafıza: RAM, ROM; Disk
  - -Özel amaçlı I/O (including analog stuff) vardır.

Bilgisayarın Tarihsel gelişimi

- Mevcut bilgisayar yapısını anlamak için kronolojik gelişim evrelerini bilmek yerinde olur.
- Bilgi işlem makinelerinin evrimi birkaç yüzyıldır devam etmektedir.
- Günümüz referans alındığında, bilgisayarların evrimi, radikal teknoloji değişimine göre dört kuşağa ayrılır.

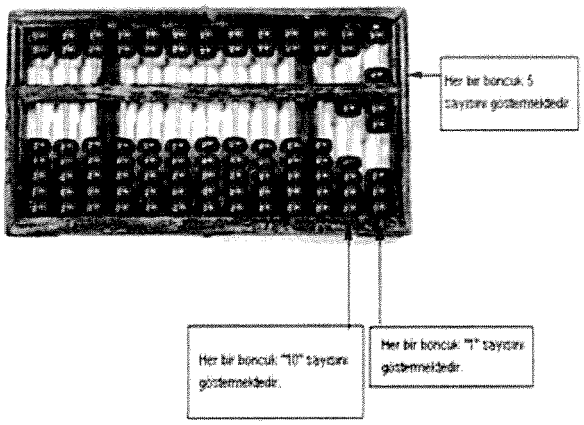
Aşağıda verilen tarihlerin yaklaşık tarihler olduğunu unutmayınız.

10

0.kuşak öncesi

ABAKÜS: Aritmetik işlemleri gerçekleştirmek için kullanılan sayma tablosudur. Sayıları temsil etmek için pozisyonel gösterimi kullanır. M.Ö 1200'lü yıllarda Çin'de ortaya çıkmıştır

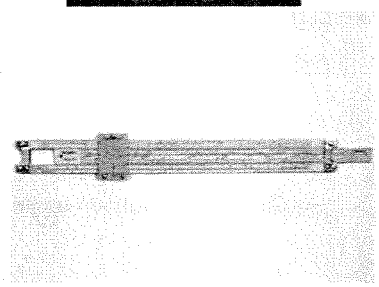
ALGORİTMA: Bir görevi gerçekleştirmek için resmi prosedürler olarak tarif edilen algoritmayı, Muhammed b. Musa el- Khwarizmi 800'lü yıllarda geliştirildi.



11

Sürgülü Hesap Cetveli

- 1621'de, Willliam Oughtred isimli bir İngiliz matematikçi, Napier'in logaritmalarını kullanarak ilk Sürgülü Hesap Cetveli'ni oluşturmuştur.
- Bu hesap cetveli, 1960lı yılların sonlarına kadar okullarda yaygın olarak kullanım alanı bulmuştur.



12

0.kuşak: Mekanik hesaplama makinaları (1642 – 1945)

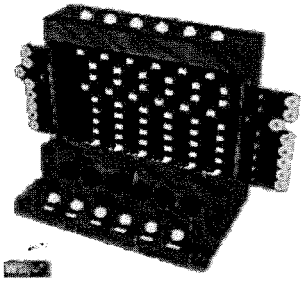
- Hesapsal saat (computing clock) - Wilhelm Schickard (1592 - 1635).
- Pascaline - Blaise Pascal (1623 - 1662).
- (Fark makinası) Difference Engine - Charles Babbage (1791 - 1871), also designed but never built the Analytical Engine.
- Delikli kart tabulating machines - Herman Hollerith (1860 - 1929).

Hollerith kartları (Delikli kart) yaygın olarak1970'lerde , bilgisayar girişi için kullanılmıştır.

13

İlk Mekanik Hesaplayıcılar

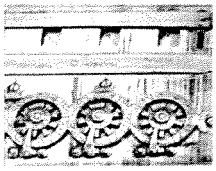
- Wilhelm Schickard'ın 1623 yılında birbiri ile bağlı dişlilerden oluşan bir cihazı geliştirmesi ile ilk mekanik hesaplayıcı ortaya çıkmıştır.

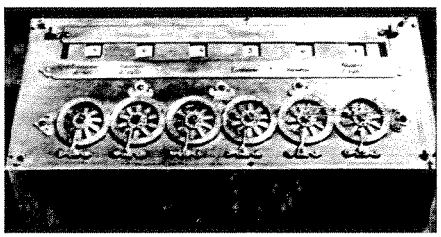


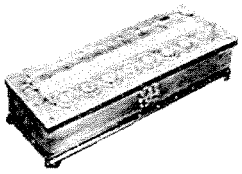
14

Pascaline

- 1642 yılında Blaise Pascal adlı bir Fransız tarafından geliştirilen ve "*Pascaline*" olarak adlandırılan cihaz ile mekanik olarak toplama, çıkarma, çarpma ve bölme işlemleri gerçekleştirilebilmekteydi.
- Dişli mekanizmaları marifetiyle, işlenecek sayılar aranıp , uygun şekilde dizilip hesap yapılırdı.



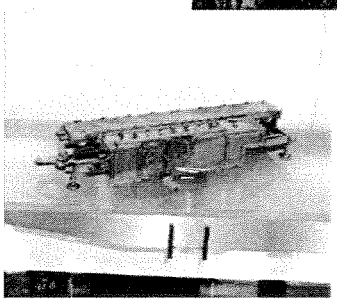




15

Leibniz Hesaplayıcısı

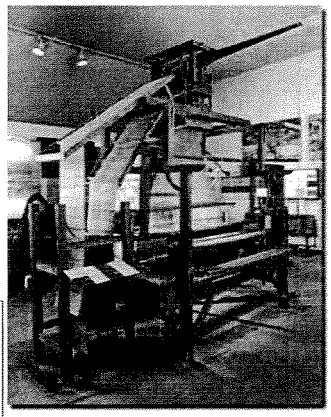
- 1673 yılında, bir Alman Baronu, Gottfried Wilhelm von Leibniz tarafından , Pascalinin birazdaha gelişkini "*Leibniz Hesaplayıcısı*" olarak adlandırılan bir diğer mekanik hesaplayıcı üretilmiştir.

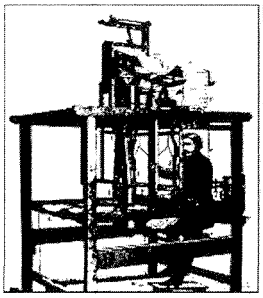


16

Jacquard'ın Dokuma Makinesi

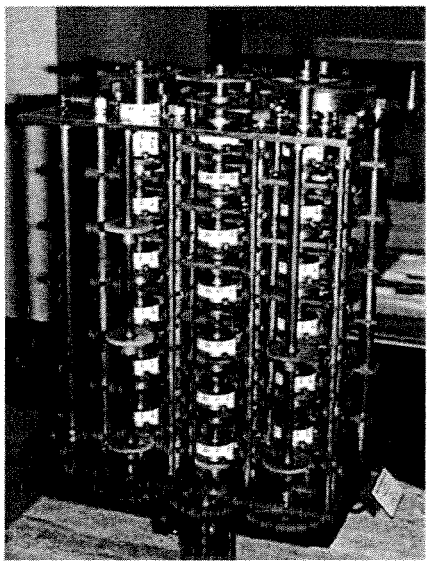
- Bir Fransız dokumacı Joseph Jacquard, delikli kartlarla (dokuma deseninin hafızaya alınmış şekli) dokuma işlemini gerçekleştiren bir dokuma makinesi üretmiştir.





17

Difference Engine - Charles Babbage (1791 – 1871) Fark Makinası

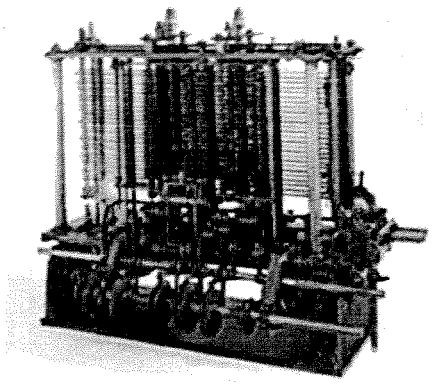


- 1822 yılında bir İngiliz matematikçi olan Charles Babbage , Daha karmaşık hesaplamalar için fark makinası tasarlamıştır.
- 4,000'den fazla dişli ve diğer parçalardan oluşan bu makine, yeterli hassasiyette dişliler üretilemediğinden, çalışır duruma getirilememiştir.

18

Babbage-Analitik Makina

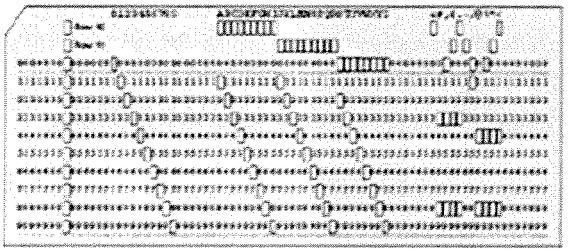
- 1834 yılında Babbage, "Analitik Makine (Analytical Engine)" adı verilen yeni bir genel amaçlı hesaplayıcı tasarlamıştır.
- Bu makinenin tasarımı, modern bilgisayarların tasarımlarının arkasında yatan hafıza, programlanabilen bir işlemci, bir çıktı aygıtı ve kullanıcı-tanımlı programlar ve veri gibi birçok kavramı tanımlamaya yaramıştır.



19

Babbage-Delikli Kartlar

- Babbage, programları ve veriyi, daha önce Jacquard'ın dokuma makinesi için geliştirdiğine benzer şekilde delikli kartlarda saklamayı önermiştir.
- Delikli kartlar, ilk elektronik bilgisayar dönemlerinde de (1970'lerde)yaygın olarak kullanılmıştır.



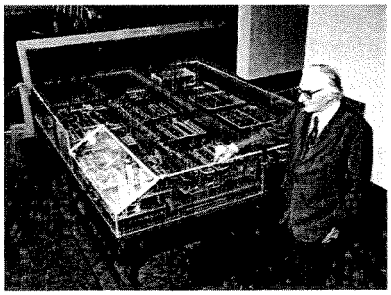
20

Tabulating Machine

- 1890 yılında,Amerikalı Herman Hollerith nüfus sayımı sonuçlarını hızlı çabucak belirleyebilmek için delikli kartlarla çalışan makineyi yaptı.
- Hollerith 'in gelistirdigi bu makine J.M.Jaquard'in 1806 yilinda kullandigi kart sistemini kullaniyordu. Mark-1, kartlari verilen kodlara göre delerek bilgiyi kaydediyor, delikli karttaki bilgiyi tekrar okuyabiliyor ve bu bilgiyi kullanabiliyordu. Mark- I insan müdahalesi ile islem gördügü için yari otomatik çalisiyordu.
- Mark –I ile sayimin degerlendirme süresi dörtte bire düstü.
- *Makine üretimini arttrmak isteyen H. Hollerithin kurduğu Tabulating MachineCompany şirketi 1924 yılında IBM (International Business Machine Corporation) adını aldı.*
-

21

Elektromekanik Bilgisayarlar

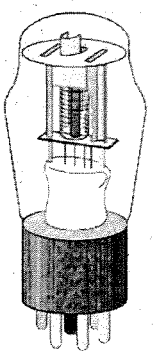


Alman Elektrik mühendisi ZUSE'nin bu eserinin tasarımı İkinci Dünya Savaşı nedeniyle gizli tutulmuştur. Bu tasarımla ilgili bilgiler savaştan sonra ortaya çıkmıştır.

- Elektriksel röleler (hareketli kontağı olan kontrollu elektrik anahtarı) kullanılarak Conrad Zuse tarafından geliştirildi.
- İlk defa binary aritmetik kullanılan makinadır.
- Z3 (1941) fonksiyonel-programlama kullanılan ilk bilgisayardır.
  - 2,000 relays
  - 5-10 Hz

22

1. kuşak Bilgisayarlar - Vacuum Tube Bilgisayarlar (1945 - 1953)



- Vakum Tüp Lambalı elektronik anahtar elemanlarının kullanıldığı bilgisayarlardır.
- İlki; Atanasoff- Berry Bilgisayar (ABC) (1937 - 1938) 'dır. Lineer denklem takımlarını çözmek için kullanıldı.
- John Atanasoff ve Clifford Berry , Iowa State Universitesi elemanlarındandır.
- ABC, mekanik anahtarlar yerine vakum tüplerinin kullanıldığı ilk bilgisayardır.
- İlk sayısal bilgisayardır
- Hesaplama işlemlerinin temeli ikili sayı sistemine dayanmaktadır

23

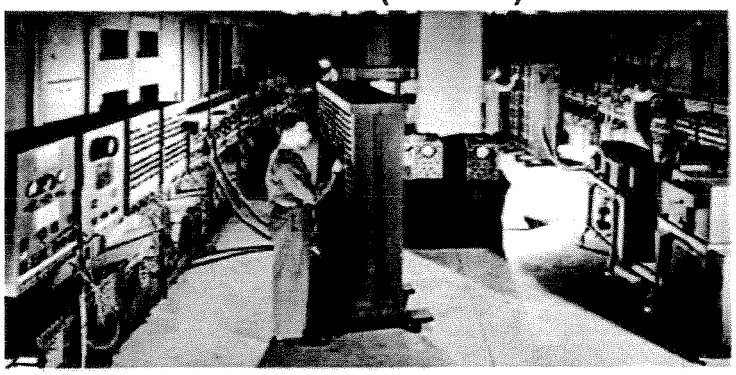
Birinci kuşak bilgiayar :ENİAC:

- John Mauchly and J. Presper Eckert tarafından...
- University of Pennsylvania, 1946
- Electronic Numerical Integrator and Computer (ENIAC)
- ENIAC, Birleşik Devletler ordusunun savaş sırasındaki işlemleri için geliştirilmeye çalışılıyordu.
- Savaşın bitiminden üç ay sonra, Kasım 1945'de tamamlanabildi.

- The ENIAC ilk genel amaçlı elektronik bilgisayardır.

24

ENIAC (1945)



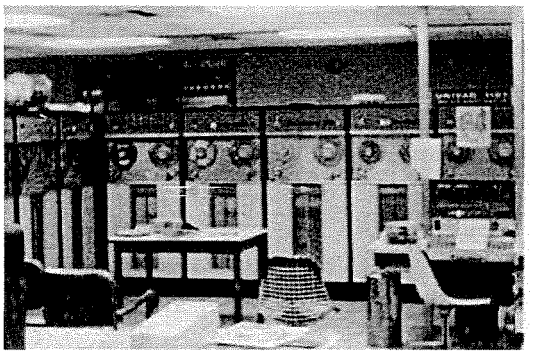
- 18000 vacuum tubes & 1500 röle kullanılmıştır. 10.000 direnç, 7.000 kondansatör ....
- 30 ton ağırlığında, 140 kW güç harcıyor (kışın soba ihtiyacını da görüyor,)
- 6000 anahtar ve kablo üzerinden Programlanıyor.
- Bir saniyede 5000 toplama, 3500 çarpma işlemi gerçekleştiriyor. Çalışma hızı 100 hz.(Şimdilerde GHZ'ler mertebesindeyiz)

- <u>Birinci kuşak bilgisayarların seri üretimi;</u>
  - The IBM 650 ilk seri üretim bilgisayar. (1955)
    - Ömrü kısa sürdü. Aşamalı olarak 1969 yılında devre dışı.
  - Bu periyottaki diğer önemli bilgisayar üreticileri ise UNIVAC, Engineering Research Associates (ERA), ve Computer Research Corporation (CRC).
    - UNIVAC ve ERA, Unisys CorporatioIn kurucusu Remington tarafından satın alınır..
    - CRC , Underwood (typewriter) Corporation tarafından satın alınır...
    - Dikkat bu yıllarda Billy Gates henüz doğmamış!!!!!!!!!

26

UNIVAC

- Ticari olarak kullanılabilen ilk sayısal bilgisayar UNIVAC (Universal Automatic Computer)'dır.
- Eckert-Mauchly Computer tarafından tasarlanmıştır.
- Remington Rand firması tarafından satın alınmıştır.
- Amerika Başkanlık seçimlerini tahmin edebilmek amacıyla kullanılmıştır.
- 1951-1958 yılları arasında 48 adet UNIVAC bilgisayar Remington-Rand firmasının müşterilerine teslim edilmiştir



UNIVAC'ın Özellikleri

- UNIVAC, 14.5 feet (4.5 m) uzunluk, 7.5 feet (2.3m) yükseklik ve 9 feet (2.7 m) genişliğe sahipti.
- ENIAC'dan daha küçük, ancak daha güçlü idi.
- Saniyede 7,200 karakter okuyabiliyor ve 2.25 milyon deyimi işleyebiliyordu
- 12,000 karakter (12k) belleğe (RAM) sahipti.
- Verilerin depolanabilmesi ve sonradan okunabilmesi için manyetik bantları kullanıyordu.
- En düşük resmi satış fiyatı yaklaşık 930,000$ idi.

28

Birinci Kuşaktaki Programlama Yöntemleri

- İlk kuşak bilgisayarların her biri yalnızca belirli işlemleri yapmak üzere programlanabilirlerdi.
- Programların değiştirilmesi, uzun süren kablo ve anahtar bağlantıları yapılarak mümkün olabilirdi.
- Birinci kuşak bilgisayarların döneminin sonlarına doğru Assembly adı verilen bir programlama dili geliştirildi
- Programcılar bu dilde programlarını daha kolay, ancak yalnızca o bilgisayarda kullanabilmek üzere yazabilme imkanına kavuştular.

29

1947'li yıllar;

- O yıllarda dünyaca çok önemli bir akademik dergi olan *Popular Mechanis* şöyle bir yorum yapıyor.

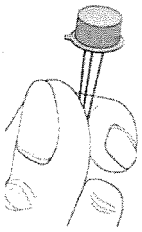
  "Öyle umuyoruz ki; Elektronikçilerin icadı ENİAC çok hızlı bir gelişmeyle 10 yıl içerisinde 10 ton ağırlığa düşer. 10Kw güç harcar, İşlem hızı iki katına çıkar"

Tabiidir ki bu yorum makinacı gözüyledir.

Oysa 1948'li yılların ortasına doğru Transistör denenen statik elektronik eleman icat edilir.

30

2. Kuşak: Transistörlü Bilgisayarlar (1954 -1965)



- –Transistörün icadıyla vakum tüp devri kapanıyor.
- –Transistör; Vakum tüpten çok daha küçük (1/13), daha az enerji harcıyor, daha hızlı anahtarlıyor.
- –IBM'in 7094 (bilimsel amaçlı) ve 1401 (iş amaçlı)
- –Digital Equipment Corporation (DEC) PDP-1
- –Univac 1100
- –Control Data Corporation 1604.
- –. . . Ve diğer üreticiler transistörlü bilgisayar seri üretimini gerçekleştiriyorlar..

Bu sistemler birkaç mimari benzerlikler vardı,

31

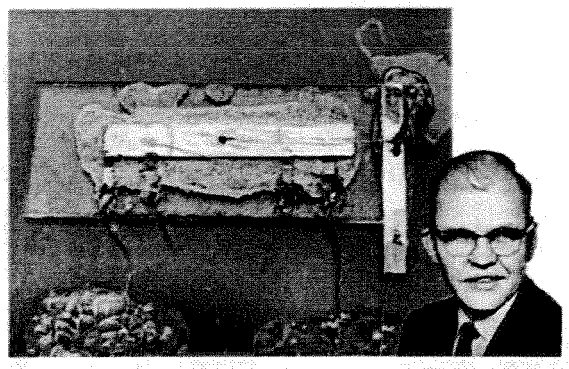
Transistor- temelli Makinalar

- IBM 704 (1955)'de transistörler kullanarak bir mainframe bilgisayar geliştiriyor.
- The 704'e değişik terminallerin bağlanabilmesi, merkezi bilgisayar fikrinin ve dağıtılmış giriş/çıkışlı uygulamaların ilkidir.
- FLOATİNG –Point sayı sisteminin uygulandığı ilk makinadır.

32

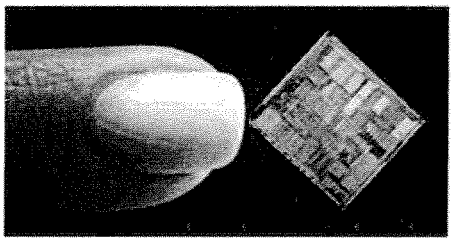
3.Kuşak: Entegre Devreli Bilgisayarlar

- Texas Instruments'da çalışan Jack Kilby ve Fairchild Semiconductor'da çalışan Robert Noyce'un birbirlerinden bağımsız olarak çalışmaları sonucunda "entegre devre" (Integrated Circuits –I C) ortaya çıktı.



Entegre Devre Teknolojisi

- Entegre devre teknolojisi, transistör, diyod v.b statik elektronik devre elemanının yüzlercesinin bir silisyum veya germanyum yüzeye kimyasal yollarla yerleştirilmesi şeklindedir.
- Böylece bilgisayar gibi cihazların fiziksel boyut, ağırlık ve güç ihtiyaçlarını büyük ölçüde azaltmıştır.
- "Entegre Devreler" üçüncü kuşak bilgisayarların gerçekleştirilmelerini sağlamıştır.



3.Kuşak-Integre devre'li Bilgisayarlar (1965-1980)

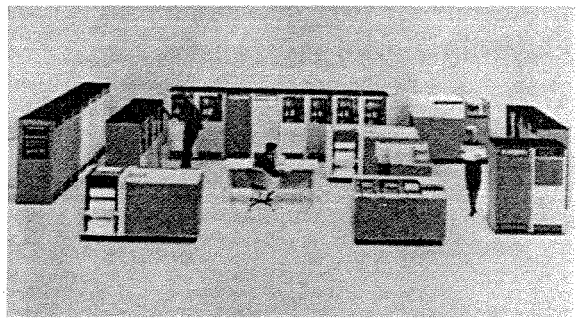
- IBM 360
- DEC PDP-8 and PDP-11
- Cray-1 supercomputer
- . . . and many others.

- Bu süreçte IBM diğer markalara ezici bir üstünlük sağlamıştır..
  - Bu dönemin Bilgisayar üreticileri IBM ve diğerleri (Burroughs, Unisys, NCR, Control Data, ve Honeywell) olarak karakterize edilmiştir.

35

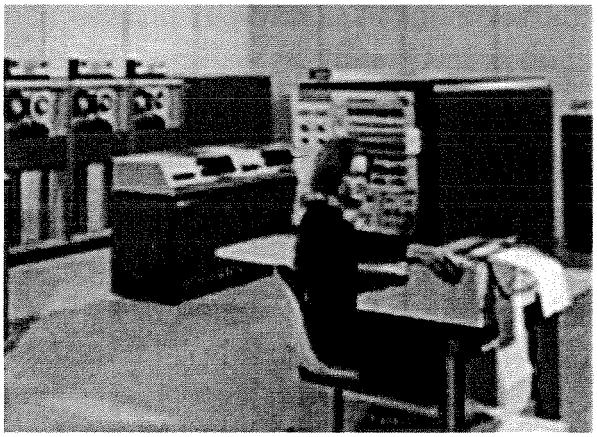
RCA Spectra 70

- Entegre devre teknolojisi kullanılarak üretilen bilgisayarlardan ilk ikisi RCA Spectra 70 ve uzun yıllar kullanımda kalan bir bilgisayar olan IBM 360'dır.



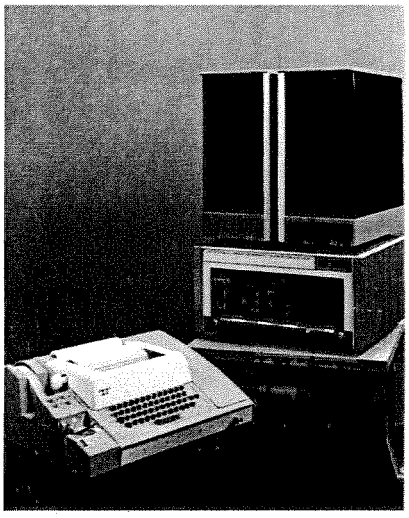
36

IBM 360



37

Minicomputers



- Çok daha büyük ve hızlı makinelere odaklanmadan, Digital Equipment Inc(DEC), 1965 yılında, ticari ilk minibilgisayar olan

  DEC PDP-8'i üretmiştir
- PDP-8, birçok üretim yerinde, küçük işletmelerde ve bilimsel laboratuarlarda yaygın olarak kullanılmıştır.

38

İlk Üçüncü Kuşak Programlama Dilleri

- Programcıların istedikleri işlemleri kolay yapabilmelerine imkan sağlayacak programlama dil arayışları da sürmekte idi.
- İlk olarak FORTRAN (Formula Translator) programlama dili mühendislik hesapları için ortaya çıktı
- Ticari uygulamalarda kullanılmak üzere COBOL (Common Business Oriented Language) programlama dili ortaya çıktı.
- Diğer programlama dilleri bu dilleri takip etti.

39

İşletim Sistemi'nin Doğuşu

- İlk kuşak bilgisayarlarda her bilgisayar için özel yazılımlar geliştirilmekteydi.
- Bir programda kullanılan bir altprogram, geliştirilen başka bir programda da yeniden yazılmaktaydı.
- Her programda standart olan altprogramları bir kütüphane yapısı içinde toplamak ve ihtiyaç duyulduğunda o altprogramları oradan çağırıp işlemleri gerçekleştirebilmenin yollarını aramaya başladılar.
- "İşletim Sistemi" yazılımları ortaya çıktı.
- Her bilgisayar üreticisi kendi işletim sistemini geliştirerek kullanıcılara daha rahat program yazabilecekleri ortamlar sağlamaya başladılar.

40

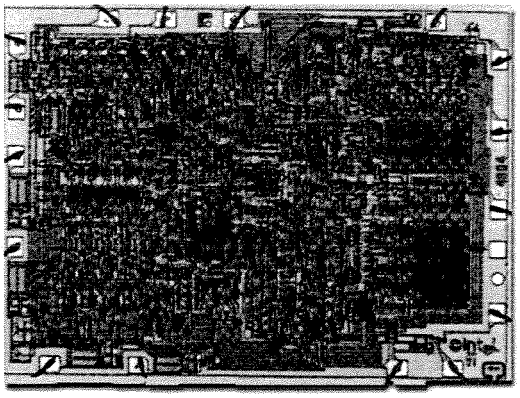
4. Kuşak
VLSI entegre devreli Bilgisayarlar
(1980 - ????)
(Microprocessors- Mikroişlemciler)

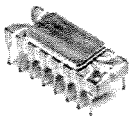
41

4. Kuşak – VLSI entegre devreli Bilgisayarlar (1980 - ????) (Microprocessors- Mikroişlemciler)

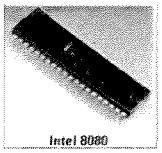
- Çok büyük çaplı İntegre devrelerin içinde en az 10.000 adet bağlaca eşdeğer eleman vardır.
- Mikroişlemci gelişimi bu dönemde çok hız kazanmıştır.
- İlk 4 bit işlemci olan INTELT 4004 gerçekleşmiştir.
- 8080, 8086 gibi daha sonraki sürümleri, ve 8088 "kişisel bilgisayar" fikrinin uygulaması.

42





Intel 4004 chip

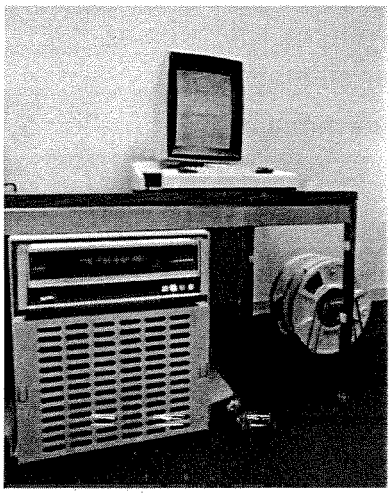


Intel 8080

- Intel'den Ted Hoff ilk mikroişlemci olan 4004'ü, Bir Japon hesap makinası üretim firması (BUSİCOM) için geliştirdi (1971).
- 4004, transistörlerin tek tek bağlandığı devreler yerine IC şeklinde gerçekleştirilmiş ilk Mikroişlemci örneğidir.
- Tek bir yonga üzerinde 2,800 transistör bulunmakta, saniyede 60,000 deyimi işleyebilmekteydi.
- Intel, 4004 modelinden sonra ilk 8-bit mikroişlemci olan 8008 modelini piyasaya sürmüştür.

43

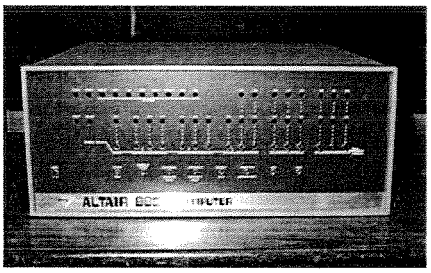
Xerox Alto (1975)



- Kullanıcı için grafiksel arayüzlü ilk bilgisayardır.
- Ayrıca bizim bildiğimiz, fare, klavye, özellikli ve "masaüstü" monitorlü ilk yapıdır.

44

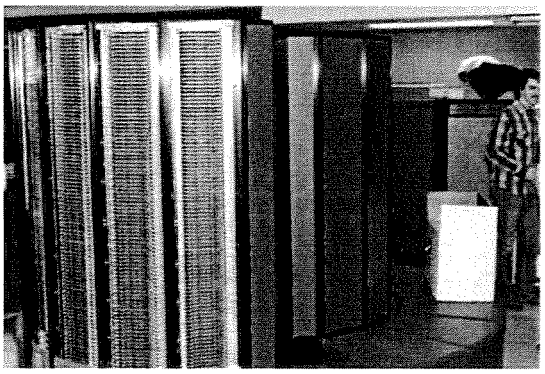
Personal computers (Kişisel Bilgisayarlar)



- MITS Altair 8800 (1975) ilk kişisel bilgisayardır.
- 256 bytes hafızalı, monitor, keyboard, harici hafızası yok.
- Microsoftun sahiplerinden Paul Allen ve Bill Gates Altair için ilk programı (BASIC derleyici) yazdı.

45

Supercomputers



- Cray XMP (1986) 4 işlemcili özel bir bilgisayar.
- Fazla ısındığından su soğutmalı olraka çalışıyor.
- Maksimum performansı 840 megaflops ...

46

1.5 Historical Development

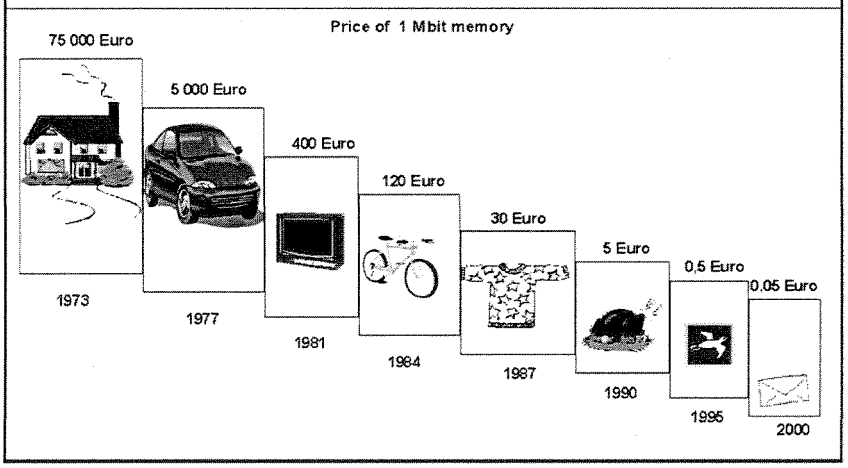
- Moore Yasası (1965)
  - Gordon Moore, Intel'in kurucusu
- "Entegre devredeki transistör yoğunluğu her yıl iki katına çıkacak."
- Günümüz versiyonu:
  "silikon çip yoğunluğu her 18 ayda iki katına çıkar."
- Nisan 2005'te Intel firması 19 Nisan 1965 tarihli *Electronics Magazine* dergisi sayısını satın almak için 10.000 ABD Doları teklif etmiştir.

Fakat bu yasa sonsuza kadar geçeli olamaz.

47

MOORE yasasına değişik bir bakış

Moore's Law: Capacity doubles every 18 months

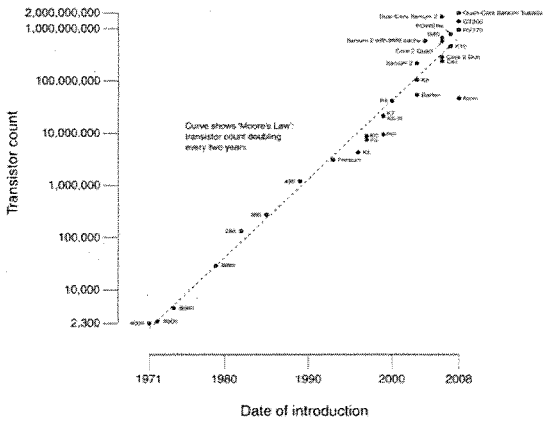


48

İNTEL İŞLEMCİLERİN GELİŞİMİ

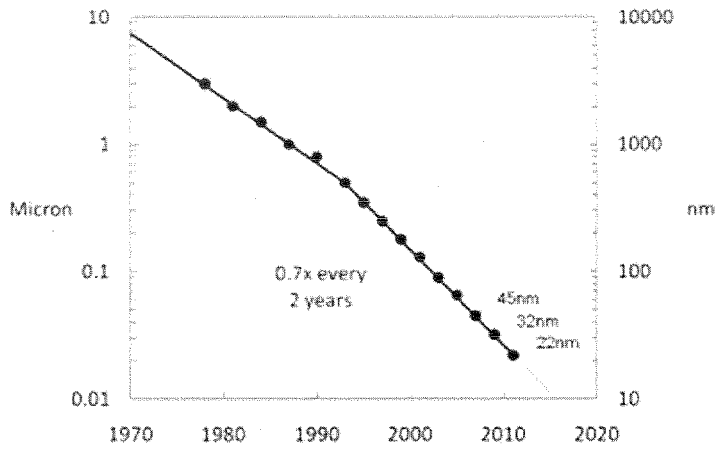
Name	Date	Transistors	Microns	Clock speed	Data width	MIPS
8080	1974	6,000	6	2 MHz	8 bits	0.64
8088	1979	29,000	3	5 MHz	16 bits 8-bit bus	0.33
80286	1982	134,000	1.5	6 MHz	16 bits	1
80386	1985	275,000	1.5	16 MHz	32 bits	5
80486	1989	1,200,000	1	25 MHz	32 bits	20
Pentium	1993	3,100,000	0.8	60 MHz	32 bits 64-bit bus	100
Pentium II	1997	7,500,000	0.35	233 MHz	32 bits 64-bit bus	~300
Pentium III	1999	9,500,000	0.25	450 MHz	32 bits 64-bit bus	~510
Pentium 4	2000	42,000,000	0.18	1.5 GHz	32 bits 64-bit bus	~1,700
Pentium 4 "Prescott"	2004	125,000,000	0.09	3.6 GHz	32 bits 64-bit bus	~7,000

CPU Transistor Counts 1971-2008 & Moore's Law



Core i7 (Quad)	731,000,000
Six-Core Core i7 (Gulftown)	1,170,000,000
10-Core Xeon Westmere -EX	2,600,000,000

Yıllara göre entegre devre teknolojisi

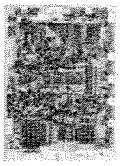


51

İntel işlemcilerin gelişimi(1971-74)

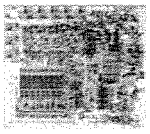
Intel® 4004 processor
Introduced 1971
Initial clock speed
108 KHz
Number of transistors
2,300
Manufacturing technology
10µ

The good news was Intel's 4004 processor was introduced with the same computing power as ENIAC.



Intel® 8008 processor
Introduced 1972
Initial clock speed
500-800 KHz
Number of transistors
3,500
Manufacturing technology
10µ

The Intel 8008 processor was twice as powerful as the Intel 4004 processor.



Intel® 8080 processor
Introduced 1974
Initial clock speed
2 MHz
Number of transistors
4,500
Manufacturing technology
6µ

The Intel 8080 processor made very small and home computers possible.

52

İntel işlemcilerin gelişimi(1978-79)

Intel® 8086 processor
Introduced 1978
Initial clock speed
5 MHz
Number of transistors
29,000
Manufacturing technology
3µ

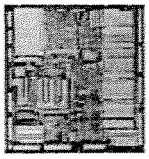
Intel® 8088 processor
Introduced 1979
Initial clock speed
5 MHz
Number of transistors
29,000
Manufacturing technology
3µ

The Intel® 8086 processor was the first 16 bit processor and delivered about ten times the performance of its predecessor.

A spin-off sale to IBM's new personal computer division made the Intel 8088 processor the brains of IBM's new hit product—the IBM PC.

53

İntel işlemcilerin gelişimi(1982-89)



Intel® 286 processor
Introduced 1982
Initial clock speed
6 MHz
Number of transistors
134,000
Manufacturing technology
1.5µ

Intel386™ processor
Introduced 1985
Initial clock speed
16 MHz
Number of transistors
275,000
Manufacturing technology
1.5µ

Intel486™ processor
Introduced 1989
Initial clock speed
25 MHz
Number of transistors
1,200,000
Manufacturing technology
1µ

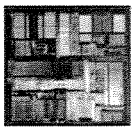
The Intel® 286 was the first x86 processor that could run all the software written for its predecessor.

The Intel386™ processor could run multiple software programs at once and hosted 275,000 transistors—more than 100 times as many as the original Intel® 4004.

The Intel486™ introduced the integrated floating point unit. This generation of consumer-ready chips could run from a connected local computer into point and click computing.

54

İntel islemcilerin gelisimi(1993-95)



Intel® Pentium® processor
Introduced 1993
Initial clock speed
66 MHz
Number of transistors
3,100,000
Manufacturing technology
0.8µ

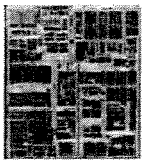
Intel® Pentium® Pro processor
Introduced 1995
Initial clock speed
200 MHz
Number of transistors
5,500,000
Manufacturing technology
0.6µ

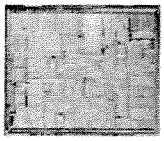
The Intel® Pentium® processor, executing 112 million operations per second, allowed computers to more easily incorporate "real world" data such as speech, sound, handwriting and photographic images.

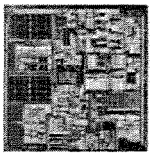
The Pentium® Pro processor delivered more performance than previous generation processors through an innovation called Dynamic Execution. This made possible the advanced 3D visualization and interactive capabilities.

55

İntel işlemcilerin gelişimi(1997-2000)







Intel® Pentium® II processor
Intel® Pentium® II Xeon® processor
Introduced 1997
Initial clock speed
300 MHz
Number of transistors
7,500,000
Manufacturing technology
0.25µ

Intel® Pentium® III processor
Intel® Pentium® III Xeon® processor
Introduced 1999
Initial clock speed
500 MHz
Number of transistors
9,500,000
Manufacturing technology
0.18µ

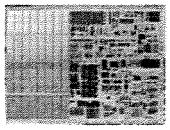
Intel® Pentium® 4 processor
Introduced 2000
Intel® Xeon® processor
Introduced 2001
Initial clock speed
1.5 GHz
Number of transistors
42,000,000
Manufacturing technology
0.18µ

The Intel® Pentium® II processor's significant performance improvement over previous Intel® Celeron® processors was based on the superior combination of the P6 microarchitecture and Intel MMX media enhancement technology.

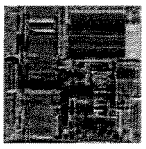
The Intel® Pentium® III processor executed Internet Streaming SIMD Extensions, extension of the product of processor identification and added multiple low-power states to conserve power during idle times.

The Intel® Pentium® 4 processor refers to the advent of the nanotechnology age.

İntel işlemcilerin gelişimi(2002-2005)



Intel® Pentium® M processor
Introduced - 2002
Initial Clock Speed
1.7 GHz
Number of transistors
55,000,000
Manufacturing technology
90nm

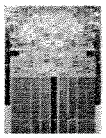


Intel® Itanium® 2 processor
Introduced 2002
Initial clock speed
1 GHz
Number of transistors
220,000,000
Manufacturing technology
0.13µ

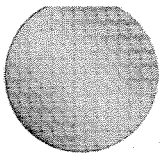
Intel® Pentium® D processor
Introduced 2005
Initial clock speed
3.2 GHz
Number of transistors
291,000,000
Manufacturing technology
65nm

57

İntel işlemcilerin gelişimi(2006-2006)



Intel® Core™ 2 Duo processor
Intel® Core™2 Extreme processor
Dual-Core Intel® Xeon® processor
Introduced 2006
Initial clock speed
2.93 GHz
Number of transistors
291,000,000
Manufacturing technology
65nm

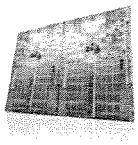


Dual-Core Intel® Itanium® 2 processor 9000 series
Introduced 2006
Initial clock speed
1.66 GHz
Number of transistors
1,720,000,000
Manufacturing technology
90nm

58

İntel işlemcilerin gelişimi(2006-2007)

Quad-Core Intel® Xeon® processor
Quad-Core Intel® Core™2 Extreme processor
Introduced 2006
Intel® Core™2 Quad processors
Introduced 2007
Initial clock speed
2.66 GHz
Number of transistors
582,000,000
Manufacturing technology
65nm



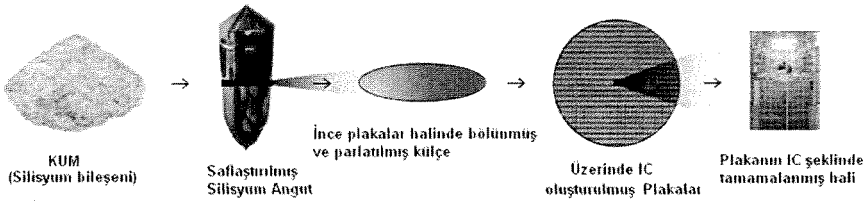
Quad-Core Intel® Xeon® processor (Penryn)
Dual-Core Intel® Xeon® processor (Penryn)
Quad-Core Intel® Core™2 Extreme processor (Penryn)
Introduced 2007
Initial clock speed
> 3 GHz
Number of transistors
820,000,000
Manufacturing technology
45nm

59

- 4-bit processors (intel 4004)
- 8-bit processors (intel 8080)
- 16-bit processors: origin of x86 (80286)
- 32-bit processors: (80386DX)
- 32-bit processors: the 80486 range (80486DX)
- 32-bit processors: P6/Pentium M microarchitecture (Pentium III)
- 32-bit processors: NetBurst microarchitecture (Pentium 4)
- 64-bit processors: (Itanium)
- 64-bit processors: Intel 64 – Core microarchitecture (Intel Core 2)
- 64-bit processors: Intel 64 – Nehalem microarchitecture (Core i3, Core i5)
- 64-bit processors: Intel 64 – Sandy Bridge microarchitecture (Core i7)

Bu teknoloji nereye kadar?

Hammadesi kum olduktan sonra, sonunu hayal edemiyoruz....



61

BÖLÜM-1

- Biz şu ana kadar, bilgisayarların ana ünitesi olan Mikroişlemcilerin (İşlemci – Microprocessors – Central Processing Unit(CPU) – Merkezi İşlem Birimi (MİB)) yapısal gelişmelerinin kronolojisini özetledik.
- Artık Bilgisayar yapısını tanımaya başlayabiliriz.

62

Bilgisayarların Bileşenleri

- ENİAC'ta programlama digital logic devre seviyesinde yapılırdı..
- Bilgisayarın programlanması, fişlerin ve tellerin farklı bir şekilde bağlanmasıyla gerçekleşirdi.
- Her bir farklı problemi çözmek için farklı donanım konfigürasyonları gerekiyordu.

Farklı bir basit problemi çözmek için, ENIAC vasıflı teknisyenler tarafından günlerce emek verilerek donanımsal konfigüre edilirdi.

63

- Aslında ENİAC'ın mucitleri, John Mauchley ve J. Presper Eckert, hafızasında komutların saklandığı bir bilgisayar tasavvur etmişlerdi.
- Bu fikrin asıl yaratıcısı o çağlarda yaşayan Matematikçi John von Neumann'dır.
- Kayıtlı-programlı bilgisayarların von Neumann Mimarisi sistemleri olarak da bilinmesi bu yüzdendir.
- Bu mimari günümüzdeki bilgisayarların temel taşıdır.

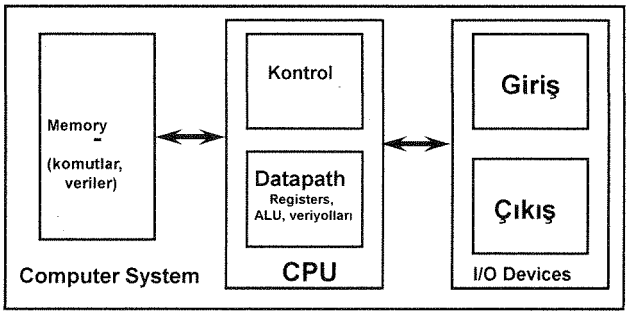
64

Von Neumann Bilgisayar Modeli

- **von Neumann mimarisi** veri ve komutları tek bir veri depolama biriminde barındıran bilgisayar tasarı örneğidir. Belleğin, işlemciden ayrılması Von Neumann mimarisi olarak ifade edilir.
- İlk hesap makineleri değişmez yazılımlara (veya programlara) sahiptiler. Bir makine hangi işlemi gerçekleştirmesi için tasarlandıysa sadece onu yerine getirebiliyordu. Bu tür bilgisayarların yeni bir görevi yerine getirebilmesi için donanımlarının tekrar tasarlanması zorunludur.
- Bu zorlukların aşılması "saklı yazılım bilgisayarı" ilkesi ile gerçekleşmiştir. Makinenin yürütebileceği bir dizi komutun tanımlanması ile bilgisayarlar çok daha esnek bir yapıya kavuştular.
- Komut ve verinin aynı biçimde saklanması ise, bu ilke sayesinde yazılımın kolayca değiştirilebilmesine olanak tanımıştır.
- Saklı yazılım tasarımı, ayrıca yazılımların çalıştırılmaları esnasında kendilerini değiştirebilmelerini mümkün kılar.

Von Neumann Bilgisayar Modeli

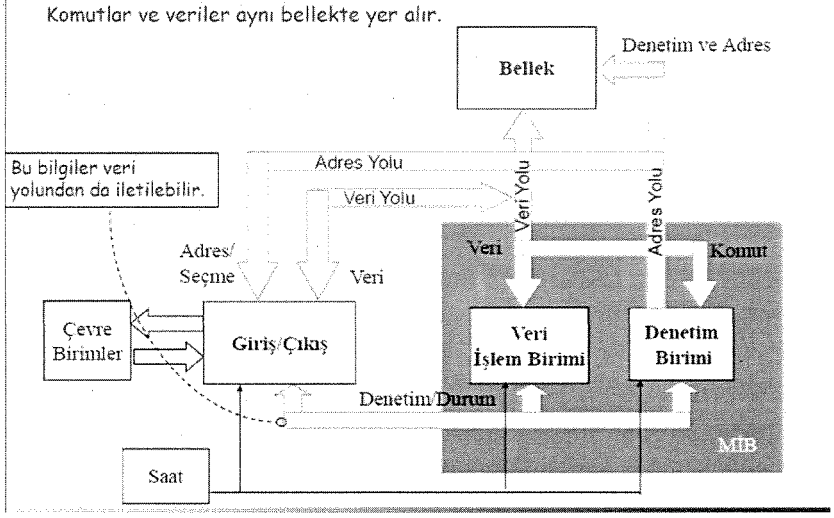
- Bilgisayarı oluşturan Bileşenler
- **von Neumann mimarisi** veri ve komutları tek bir veri depolama biriminde barındıran bilgisayar tasarı örneğidir.
  - Central Processing Unit (CPU - MİB): Kontrol Birimi (komut kodlarının çözülmesi, İşlemlerin sıraya konması v.b), Datapath (Registers'lar, Aritmetik ve lojik işlem birimi , Veri yolları) alt birimleri vardır.
  - Memory: Komutların ve verilerin saklandığı ortam.
  - Input/Output (I/O) sub-system: I/O BUS'ları, Arayüzler, Elemanlar.
  - The stored program concept: Bir zaman diliminde Komut setinin hafızalandığı bir ortak bellekten çağrılan komutlar icra edilir.



Major CPU Performance Limitation: The Von Neumann computing model implies sequential execution one instructio at

Von Neumann Mimarisi: John von Neumann (1903 - 1957)

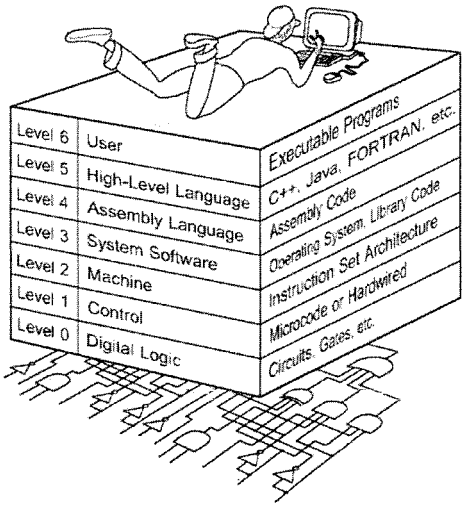
Komutlar ve veriler aynı bellekte yer alır.



67

Bilgisayarın hiyerarşik yapısı

- **Bilgisayarların çalışmasını anlamak için sanal katmanlı bir yapı modeli uygundur.**
- **Her bir katman belirli görevleri gerçekleştirmek içindir ve kendi komutları vardır. Katmanlar gerektiğinde alt seviyelerdeki komutları çağırabilirler.**
- **Esas işi enalt tabakadaki sayısal devreler yapacaktır.**



68

- **Seviye 6: Kullanıcı seviyesi**
  - Kullanıcı ile bilgisayarın etkileştiği arayüz. Bu seviye birçok bilgisayar familyası için aynıdır.
- **Seviye 5: Yüksek-seviyeli dil seviyesi**
- C, Pascal, Java v.b programlas dilleri ile yazdığımız programlarla bilgisayarla etkileştiğimiz seviyedir.
- **Seviye 4: Assemler dili seviyesi**
  - 5.seviyedeki yüksek-seviye dilinde yazılmış program assemler diline çevrilir. as well as instructions programmed directly at this level.
- **Level 3: Sistem yazılımı seviyesi**
  - Sistemde işlenen prosesleri kontrol eder. Sistem kaynaklarını korur.
  - Assemler dili komutları fazla değişikliğe uğramadan bu seviyeye geçer.
- **Seviye 2: Makine dili seviyesi**
  - Instruction Set Architecture (ISA – Komut Seti Mimarisi) seviyesi olarak bilinir.
  - Makine mimarisine özel komutların oluşturulduğu seviyedir. Makine dilinde yazılmış programlar hiçbir şekilde derleyiciye, yorumlayıcıya veyea assemler'a ihtiyaç duymazlar.
- Seviye 1: Kontrol Seviyesi
  - Bir kontrol ünitesi komutları decode eder ve işler. Verilerisistem üzerinde hareket ettirir.
  - Kontrol üniteleri mikroprogramlanabilir olabilir veya sadece devrelendirilmiş(Hardwired) yapıdadır.
  - Bir mikroprogram donanım tarafından uygulanabilen, bir düşük seviyeli bir dilde yazılmış bir programdır.
  - Devrelendirilmiş(Hardwired) üniteler, doğrudan makine komutlarını çalıştırır donanımlardır.
- **Seviye 0: Digital Logic Seviyesi**
  - Bu seviyede biribirine uygun bir şekilde bağlanmış sayısal lojik devreler bulunur (IC'nin içinde)
  - Bu komponentler, diğer üst sevilerdeki matematiksel ve Lojik işlemleri gerçekleştirir.

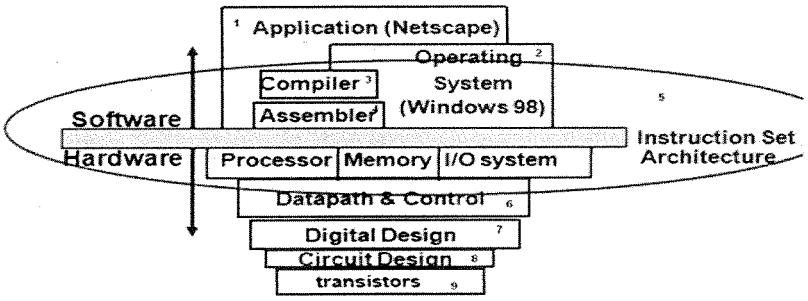
69

Hiyerarşik yapının şeması

1-Uygulama Yazılımları (Applıcatıon SW): kullanıcı ve sistem yazılımı arasındaki etkileşimi sağlar. Yüksek seviyeli dildir. C++, Web Browser v.b

2,3,4 Sistem yazılımları: (2) İşletim Sistemi: Bilgisayarın kaynaklarını yönetir, birden fazla programın beraber çalışmasını düzenler. **(3) Compiler :**Yüksek seviyeli dilde yazılmış deyimleri assemler diline dönüştürür. **(4) Assemler:** Sembolik deyimleri makinenin anlayacağı komutlara, makine diline çevirir.

5- komut Seti Mimarisi (ISA): Yazılımdan donanıma geçiş arayüzüdür. Bu derste uğraşacağımız en önemli konudur. Kapsadığı birimlere dikkat!!!!!!!!!!!!!!



70

Yüksek seviyeli dilden- Sayısal gerçeklemeye

High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Compiler

Assembly language program (for MIPS)

```
swap:
      mult $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```

Assembler

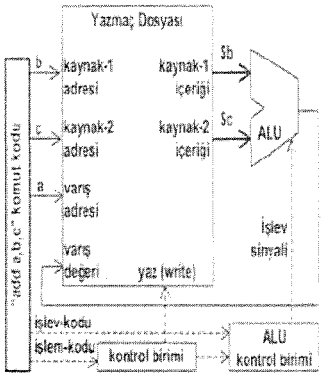
Binary machine language program (for MIPS)

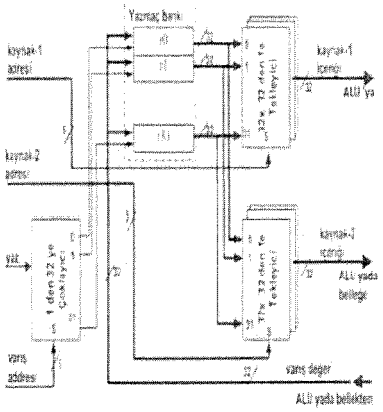
```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

71

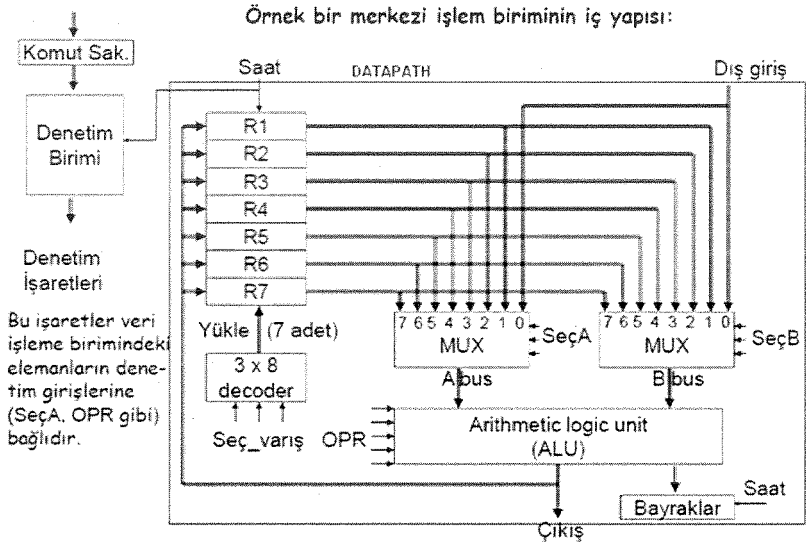
Özet

a = b+c ----add a,b,c-------01101110





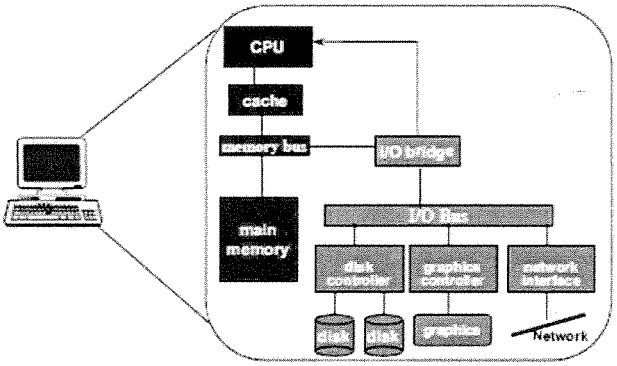
Örnek bir merkezi işlem biriminin iç yapısı:



Bu işaretler veri işleme birimindeki elemanların denetim girişlerine (SeçA, OPR gibi) bağlıdır.

73

BİLGİSAYAR SİSTEM ORGANİZASYONU

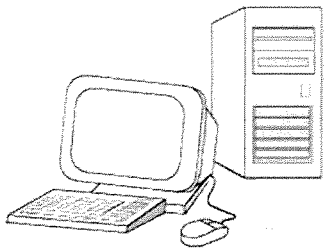


74

Bir reklam düşünün:

MHz??

For Sale: Obsolete Computer – Cheap! Cheap! Cheap!



- xxx Motherboard, xx23 Chipset
- Pentium 4.20GHz
- 400MHz 256MB DDR SDRAM
- 32KB L1 cache, 256KB L2 cache
- 80GB serial ATA hard drive (7200 RPM)
- 8 USB ports, 1 serial port, 1 parallel port
- Monitor 19" .24mm AG, 1280 x 1024 at 75Hz
- 48x CD-RW drive
- 128MB PCI express video card
- 56K PCI data/fax modem
- 64-bit PCI sound card
- Integrated 10/100 Ethernet card

MB??

USB??

Bunların hepsi ne demek?

75

1.3 An Example System

Kapasite ve hız ölçü birimlerini hatırlayalım.

- Kilo- (K) = 1 thousand = 10^3 and 2^{10}
- Mega- (M) = 1 million = 10^6 and 2^{20}
- Giga- (G) = 1 billion = 10^9 and 2^{30}
- Tera- (T) = 1 trillion = 10^{12} and 2^{40}
- Peta- (P) = 1 quadrillion = 10^{15} and 2^{50}
- Exa- (E) = 1 quintillion = 10^{18} and 2^{60}
- Zetta- (Z) = 1 sextillion = 10^{21} and 2^{70}
- Yotta- (Y) = 1 septillion = 10^{24} and 2^{80}

76

1.3 An Example System

- Hertz = Bir saniyedeki clock sayısı (frequency)
  - 1MHz = 1,000,000Hz
  - İşlemci hızı MHz veya GHz. Birimi ile ölçülür.
- Byte = Hafıza kapasite birimidir.
  - 1KB = 2^{10} = 1024 Bytes
  - 1MB = 2^{20} = 1,048,576 Bytes
  - Ana hafıza (RAM) genellikle MB birimi ile ifade edilir.
  - Hard disklerin hafıza birimi genellikle GB veya TB 'dır.

77

1.3 An Example System

Zaman ve yer ölçme birimleri:

- Milli- (m) = 1 thousandth = 10^{-3}
- Micro- (μ) = 1 millionth = 10^{-6}
- Nano- (n) = 1 billionth = 10^{-9}
- Pico- (p) = 1 trillionth = 10^{-12}
- Femto- (f) = 1 quadrillionth = 10^{-15}
- Atto- (a) = 1 quintillionth = 10^{-18}
- Zepto- (z) = 1 sextillionth = 10^{-21}
- Yocto- (y) = 1 septillionth = 10^{-24}

78

1.3 An Example System

- Milisaniye = 1/100.000 saniye
  - Bir Hard disk sürücünün erişim zamanı 5 - 20 ms'dir.
- Nanosaniye = 1/1.000.000.000 saniye
  - Bir RAM'ın erişim zamanı 30 ile 70 nanosaniyedir.
- Micron (micrometre) = 1 millionth of a meter
  - Bir işlemci teknolojisini belirtmek için iki komponent arasındaki mesafeyi verir. Örnek 1mikron teknolojisi.

79

1.3 An Example System

- Bir clock işaretinin, Frekansıyla periyodu(cycle) arasındaki f=1/T ilişkisini unutmayalım.
- 133MHz'lik bir veriyolu hızının bir periyotluk zamanı :

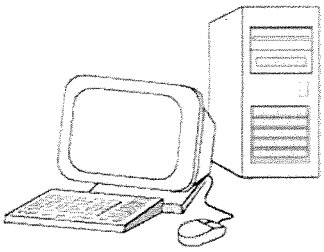
133,000,000 cycles/second = 7.52 ns/cycle

Bu tanımlardan sonra tekrar reklama dönelim.....

80

Ana kart: İşlemci; Hafıza birimleri, çevre birimleri ile etkileşimi sağlayan veriyolları ve arayüzlerin üzerinde bulunduğu elektronik karttır. Üzerindeki cipset entegre bu trafiği yönetmek içindir.

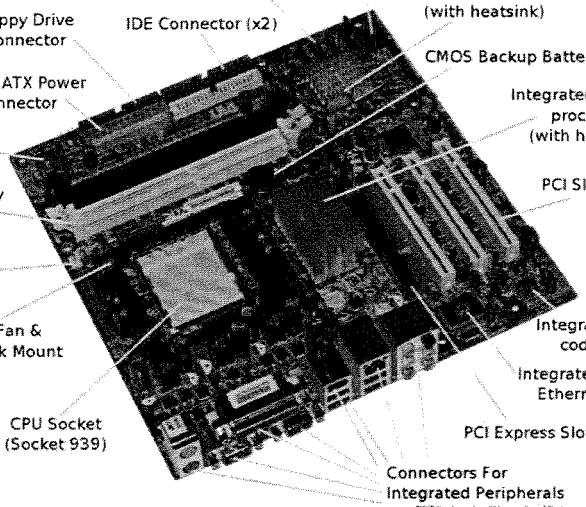
Cheap!



- xxx Motherboard, xx23 Chipset
- Pentium 4.20GHz
- 400MHz 256MB DDR SDRAM
- 32KB L1 cache, 256KB L2 cache
- 80GB serial ATA hard drive (7200 RPM)
- 8 USB ports, 1 serial port, 1 parallel port
- Monitor 19" .24mm AG, 1280 x 1024 at 75Hz
- 48x CD-RW drive
- 128MB PCI express video card
- 56K PCI data/fax modem
- 64-bit PCI sound card
- Integrated 10/100 Ethernet card

81

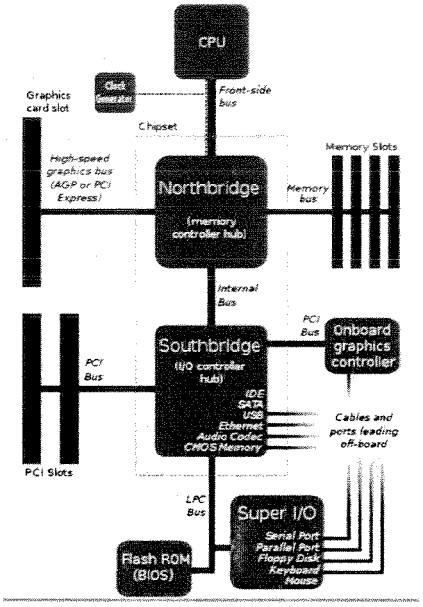
2000'li yıllardaki ana kart örneği



1/3C

82

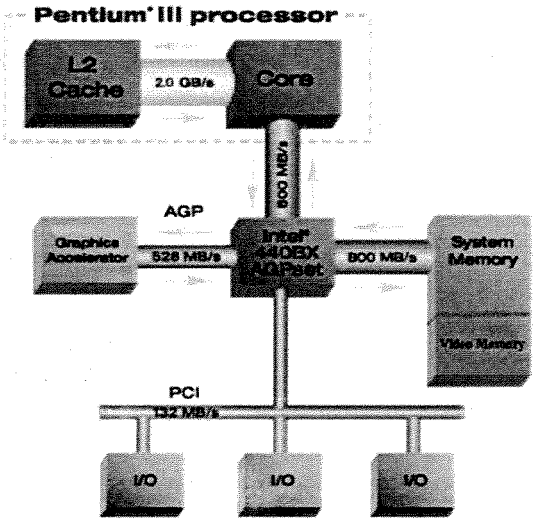
Chipset(Ana kart trafiğini yöneten birim)

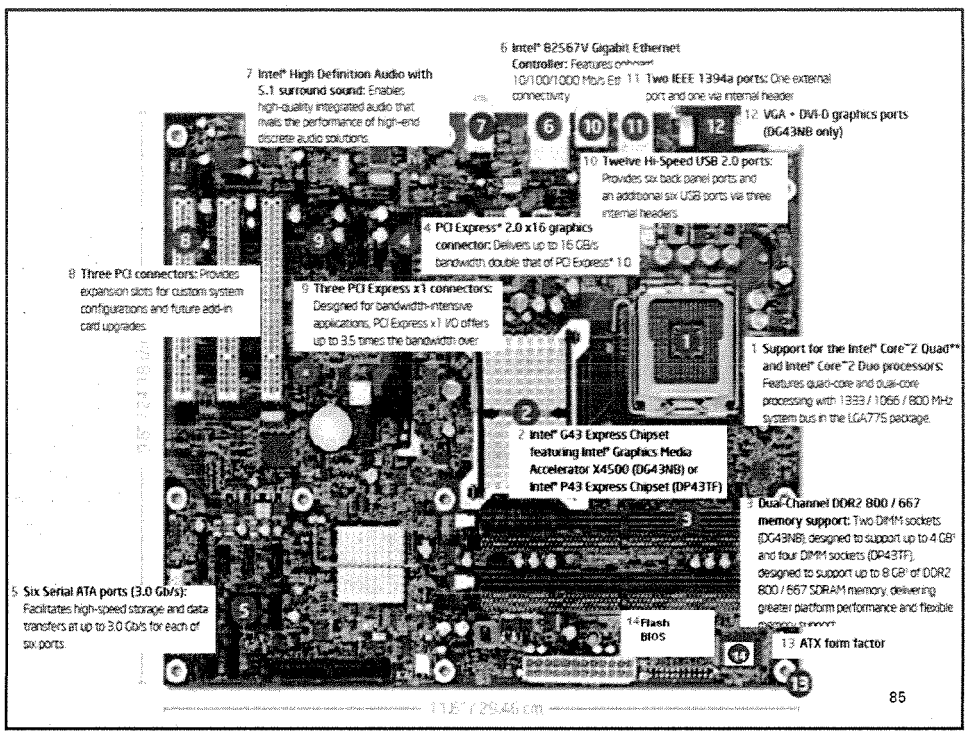


- Bilgisayar donanımlarını birbirlerine bağlamak ve koordineli halde çalışmalarını sağlamak için kullanılan bir donanımdır. Parçaların aralarında iletişim kurmasını ve gerekli işlemlerin gerçekleşmesini sağlamak <u>için anakartlar üzerinde harici işlemci yanında kart işlemcileri de bulunmaktadır. Chipset olarak</u> ta adlandırılan bu işlemcilerin bilgisayar performansı üzerinde çok etkili oldukları bilinmektedir. Chipset'lerdeki gelişmeler işlemcilerdeki gelişmelere paralel olarak ilerlemektedir. Yeni bir RAM ya da bus geliştirildiği zaman bunu işlemciye aktaracak olan Chipsetler de geliştirilir
- **Kuzey Köprüsü (North Bridge)**
- Anakart üzerinde bulunan, bellek ve AGP ve Güney Köprüsünü işlemciye bağlayan yongadır.
- **Güney Köprüsü (South Bridge)**
- North Bridgearacılığı ile işlemciye bağlanarak paralel ,seri,usb,ps2 gibi düşük hızlı çıkış ve arabirimlerin işlemciye bağlanmasını sağlayan çipin ismidir.

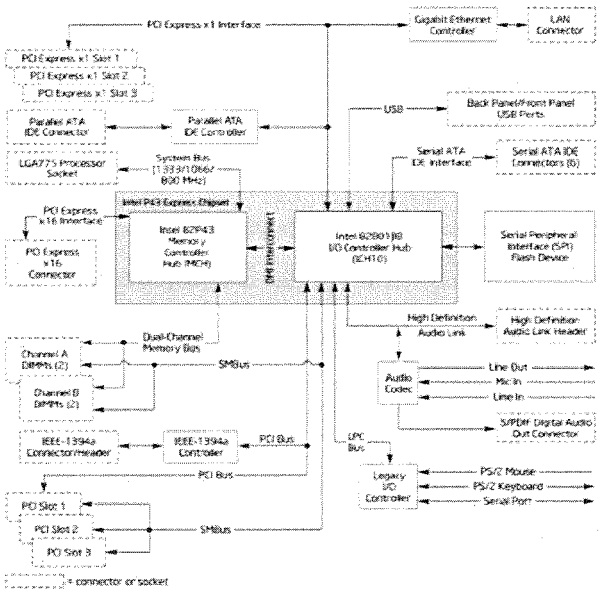
83

İşlemci ve chip set




1 Support for the Intel® Core™2 Quad** and Intel® Core™2 Duo processors: Features quad-core and dual-core processing with 1333 / 1066 / 800 MHz system bus in the LGA775 package.
2 Intel® G43 Express Chipset featuring Intel® Graphics Media Accelerator X4500 (DG43NB) or Intel® P43 Express Chipset (DP43TF)
3 Dual-Channel DDR2 800 / 667 memory support: Two DIMM sockets (DG43NB), designed to support up to 4 GB* and four DIMM sockets (DP43TF), designed to support up to 8 GB* of DDR2 800 / 667 SDRAM memory, delivering greater platform performance and flexible
4 PCI Express* 2.0 x16 graphics connector: Delivers up to 16 GB/s bandwidth, double that of PCI Express* 1.0
5 Six Serial ATA ports (3.0 Gb/s): Facilitates high-speed storage and data transfers at up to 3.0 Gb/s for each of six ports
6 Intel® 82567V Gigabit Ethernet Controller: Features
7 Intel® High Definition Audio with 5.1 surround sound: Enables high-quality integrated audio that rivals the performance of high-end discrete audio solutions.
8 Three PCI connectors: Provides expansion slots for custom system configurations and future add-in card upgrades
9 Three PCI Express x1 connectors: Designed for bandwidth-intensive applications, PCI Express x1 I/O offers up to 3.5 times the bandwidth over
10 Twelve Hi-Speed USB 2.0 ports: Provides six back panel ports and an additional six USB ports via three internal headers
11 Two IEEE 1394a ports: One external port and one via internal header
12 VGA + DVI-D graphics ports (DG43NB only)
13 ATX form factor
14 Flash BIOS

85


PCI Express x1 Interface
Gigabit Ethernet Controller
LAN Connector
PCI Express x1 Slot 1
PCI Express x1 Slot 2
PCI Express x1 Slot 3
Parallel ATA IDE Connector
Parallel ATA IDE Controller
USB
Back Panel/Front Panel USB Ports
LGA775 Processor Socket
System Bus (1333/1066/800 MHz)
Serial ATA IDE Interface
Serial ATA IDE Connectors (6)
Intel P43 Express Chipset
PCI Express x16 Interface
PCI Express x16 Connector
Intel 82P43 Memory Controller Hub (MCH)
DMI Interconnect
Intel 82801IB I/O Controller Hub (ICH10)
Serial Peripheral Interface (SPI) Flash Device
High Definition Audio Link
High Definition Audio Link Header
Dual-Channel Memory Bus
Channel A DIMMs (2)
Channel B DIMMs (2)
SMBus
Audio Codec
Line Out
Mic In
Line In
S/PDIF Digital Audio Out Connector
IEEE-1394a Connector/Header
IEEE-1394a Controller
PCI Bus
LPC Bus
Legacy I/O Controller
PS/2 Mouse
PS/2 Keyboard
Serial Port
PCI Slot 1
PCI Slot 2
PCI Slot 3
= connector or socket

86

ANA kart ile ilgili

Bus (Veri yolu)

Bilgisayardaki veriyoluna verilen isimdir. Paralel iletken yollardan oluşur. Bayt'ı oluşturan bitleri; paralel olarak 0 veya 1 sevyiesinde elektrisel işaretler şeklinde taşır.

Anakart üzerinde bir çok çeşit bus yapısı vardır. Bu farklılık her bileşenin farklı ihtiyacından kaynaklanmaktadır. Örneğin hafıza veriyolu (memory bus), kuzey köprü (hafıza kontrolcusu) ile hafıza arasındaki veriyoludur.

Yeni nesil bilgisayarlarda(günümüzdeki) hafıza ile CPU arasındaki veriyoluna FSB (front side bus), CPU ile level2 bellek arasındaki veriyoluna BSB(back side bus) ismi verilmektedir. Günümüzde **PCI, AGP, PCI Express, USB,SCSI, SATA, firewire ve PCMCIA** gibi veri yolları vardır. Verileri bazıları seri bazıları ise paralel olarak iletir. Her bir veriyolu yapısı kendisine has bir slot veya soket yapısına sahiptir. Veriyolları aygıt sürücüleri yardımıyla kontrol edilir.

SCSI(Small Computer System Interface): Birçok farklı donanım birimini(yazıcı, cd-rom, sabit disk, tarayıcı...) destekleyen, aynı veriyoluna birçok donanım biriminin bağlanmasına imkan veren bir veriyolu yapısıdır. SCSI veriyoluna bağlanan her bir donanımın tekil bir numarası (ID) olması gerekmektedir. SCSI-3 versiyonunda Maksimum veri yolu genişliği 16 bit, veriyolu hızı 80Mhz ve desteklenen donanım sayısı 16 dır.

IEEE1394 Firewire: Daha çok video, ses, görüntü işleme ve DVD dünyasında kullanılan yüksek hızlarda seri veri aktaran bir yapıdır. **İ-Link(Sony) ve DV(Panasonic)** olarak ta isimlendirilir. SCSI'nın daha da gelişmiş şeklidir. Günümüz bilgisayarlarında paralel SCSI'nin yerini almıştır. Firewire bant genişliği 400, 800 ve 3200 Mbit değerlerinde olabilmektedir. Firewire HotPlugging yapıya sahiptir.

87

Ana kart ile ilgili -II

PCI(Peripheral Component Interconnect): Ses, ekran, tv ve ağ kartları bağlanabilir. Paralel iletişimi kullanılır. Anakart üzerine onboard olarak veya slotlar yardımıyla bağlanan donanım yapılarını destekler. 33/66Mhz frekans değerini ve 32/64 bit band genişliğini kullanan modelleri vardır.

AGP(Accelerated Graphics Port): Hızlandırılmış grafik port'u anlamına gelen AGP, ekran kartları ve video kartları için kullanılan yeni bir veri yoludur. Paralel iletişimi kullanır. 32 bit genişliğe sahiptir. **1x, 2x, 4x** ve **8x** gibi sürümleri vardır. AGP yapısı bir veriyolu olmasına rağmen noktadan noktaya iletimi sağlar. Sabit 32 bit genişliğin her bir çarpana 66Mhz eklenip çarpılmasıyla band genişliği hesaplanır.

PCI veri yolu ile aralarındaki temel fark: AGP'ler 128 KB'a varan büyük grafik dokularını (texture) ekran kartı belleğinin dışında, sistem belleğinden de yararlanarak işler. Bu sayede performansta artış sağlanır. AGP veri yolunun performansta bu şekilde bir artış sağlamasına "Doğrudan Bellek Kullanımı" DIME (Direct Memory Execute) denir.

PCMCIA veya PC Card (Personal Computer Memory Card International Association): Dizüstü bilgisayarlarda kullanılan genişleme yuvalarının bağlandığı veriyoludur. **31-1067 Mbit** bant genişliğine sahiptir.

USB(Universal Serial Bus): Evrensel seri veriyolu anlamındadır. PnP özelliğe sahiptir. Tek bir bilgisayara 127 adet donanım bağlamaya izin veren bir yapıdır. Günümüzde monitör, klavye, fare, TV kartı, Ses kartı, sabit disk gibi birçok donanım birimi bu veriyolunu kullanmaktadır. **HotPlugging** yapıya sahiptirler.**USB3.0** Çok yüksek hızlı olup 4.8Gbit bant genişliğine sahiptir.

88

Anakart ile ilgili -III

PCI Express: En yeni bus yapısıdır. PCI Express PCI veri yolunda kullanılan paralel veri iletimi mimarisinin yerine seri çalışan ve noktadan noktaya iletişim mimarisini kullanan bir teknoloji getiriyor Çarpanları sahip olduğu hat sayısını gösterir. Veriler paketler halinde iletilir. Bus yapısından ziyade ağ mantığı ile çalışır. Veri seri olarak birkaç hattan gönderilip alınabilir. Bu hatların her birine kanal denmektedir ve çarpanla gösterilir. Veriler anahtarlama yöntemiyle istenen noktalara kanalize edilerek band genişliğinden bağımsız iletim oluşturulur. Yani her bir kanal için adanmış bir yol anahtarlanarak sağlanır. Her bir hat sayısı **250MB/s band genişliğine** sahiptir. Haberleşme şekli seridir. Kartlara sağladığı elektriksel güç AGP den yaklaşık iki kat fazladır. Slot uzunluğu, çarpan sayısı ile doğru

PCI-Express veriyolunun çalışma mimarisi aynı donanım birimlerini ortak çalıştırmak için elverişlidir.

Serial ATA(SATA), masaüstü bilgisayarlardaki, bazı sunuculardaki ve ağa bağlı depolama cihazlarındaki(HD) paralel ATA fiziksel depolama arabiriminin yenilenmiş versiyonudur.

89

Anakartla ilgil -IV

- **Portlar**
  Bilgisayarın dış dünya(modem, kalvye, yazıcı, kamera...) ile iletişimini sağlayan giriş çıkış kapılarına port denir. Aslında bunlarda birer veriyoludur. Seri, paralel, USB, PS/2... gibi çeşitleri vardır.

- **BIOS (Basic Input Outpu System)** Temel giriş çıkış sistemi anlamına gelir. Bilgisayar ilk açıldığında bu sistemin sahip olduğu program kodları ile başlatılır. Bu kodlar **eeprom** veya **flash hafıza** denilen yapılarda saklanır. Biosun ilk işi sistem elemanlarını(ekran kartı, sabit disk, Ram...) tanımlamak ve onları kontrol etmektir. Donanım birimleri denetlendikten ve başlangıç parametreleri yüklendikten sonra **BIOS** disk üzerindeki MBR (Master Boot Record) alanında işletim sistemi açılış dosyalarını arar. İşletim sistemini çağırarak kontrolü ona bırakır.
  Chipset
  Anakart üzerinde faklı donanımların birbiri ile iletişimini sağlayan chiplerdir. Norhbridge(kuzey köprü) ve **Soutbridge**(güney köprü) adında iki bileşenden(chip) oluşur. **Norhbridge** hızlı bileşenleri(CPU, RAM, PCI Express ve AGP) birbirine bağlar. SouthBridge ise yavaş bileşenleri birbirine bağlar(IDE, USB, PCI...).

90

Veriyolu karşılaştırmaları

Adı **Band genişliği**

AGP 1x 266MB/s
AGP 2x 533MB/s
AGP 4x 1067MB/s
AGP 8x 2133MB/s
PCI(32Bit, 33Mhz) 132MB/s
PCI(64Bit, 33Mhz) 266MB/s
PCI(64Bit, 66Mhz) 533MB/s
PCI Express 1x 250MB/s
PCI Express 2x 500MB/s
PCI Express 4x 1000MB/s
PCI Express 8x 2000MB/s
PCI Express 16x 4000MB/s
PCI Express 32x 8000MB/s
EISA 32MB/s

91

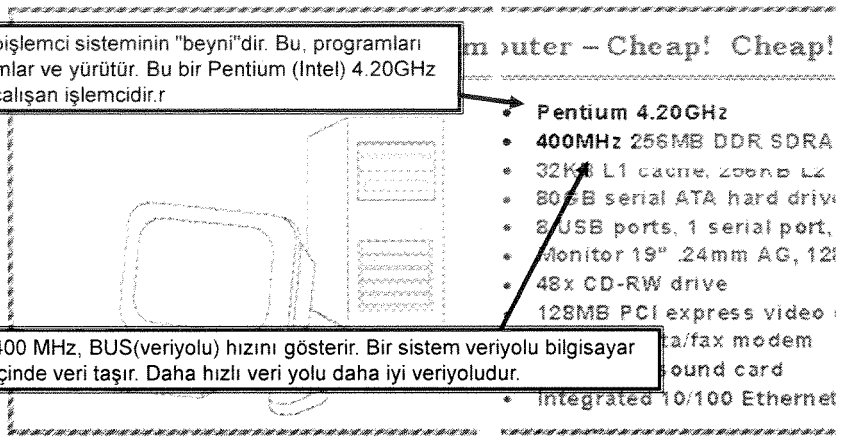
ANAKART BİLGİLERİ- Örnek

Form Factor	ATX (294.64 millimeters [11.60 inches] x 243.84 millimeters [9.60 inches])
Processor	Support for an Intel® processor in the LGA775 package
Main Memory	• Four 240-pin, DDR2 1.8 V SDRAM Dual Inline Memory Module (DIMM) sockets • 800/667 MHz single or dual channel DDR2 SDRAM interface • Support for up to 8 GB of system memory
Chipset	Intel® P43 Express Chipset consisting of: • Intel P43 Express Chipset Memory Controller Hub (MCH) • Intel® 82801JB I/O Controller Hub (ICH10)
Graphics	One PCI Express* 2.0 x16 connector supporting PCI Express graphics add-in cards
Audio	Onboard subsystem, featuring: • Independent 6-channel (5.1) audio streams • 2-channel stereo audio streams via an onboard header • Intel® High Definition Audio (Intel® HD Audio) interface • RealTek* ALC888VC audio codec • Onboard 3-pin S/PDIF output connector
Expansion Capabilities	• One PCI Express 2.0 x16 connector • Three PCI Express 1.1 x1 connectors • Three PCI* connectors
Legacy I/O Support	Legacy I/O Controller that provides: • PS/2* keyboard and mouse ports • One serial port via an onboard header
Peripheral Interfaces	• Up to 12 USB 2.0 ports: — Six ports routed to the back panel — Six ports routed to three USB headers • Up to two IEEE 1394a ports: — One port routed to the back panel — One port routed to an onboard IEEE 1394a header • Six Serial ATA (SATA) channels (3.0 Gb/s) via ICH10 • One IDE interface with ATA-66/100 support (two devices)
BIOS	• Intel® Platform Innovation Framework for EFI • Intel® Rapid BIOS Boot • Intel® Express BIOS Update
Power Management	• Support for Advanced Configuration and Power Interface (ACPI) • Suspend to RAM (STR) • Wake on USB, PCI, PCI Express, PS/2, LAN, and front panel • ENERGY STAR* capable
Hardware Management	Hardware monitor with: • Four fan sensing inputs used to monitor fan activity • Intel® Quiet System Technology (Intel® QST) fan speed control • Voltage sensing to detect out of range values
LAN Support	Intel® 82567LV Gigabit (10/100/1000 Mb/s) Ethernet LAN controller

92

1.3 An Example System

Mikroişlemci sisteminin "beyni"dir. Bu, programları yorumlar ve yürütür. Bu bir Pentium (Intel) 4.20GHz 'de çalışan işlemcidir.r



400 MHz, BUS(veriyolu) hızını gösterir. Bir sistem veriyolu bilgisayar içinde veri taşır. Daha hızlı veri yolu daha iyi veriyoludur.

93

İŞLEMCİLER

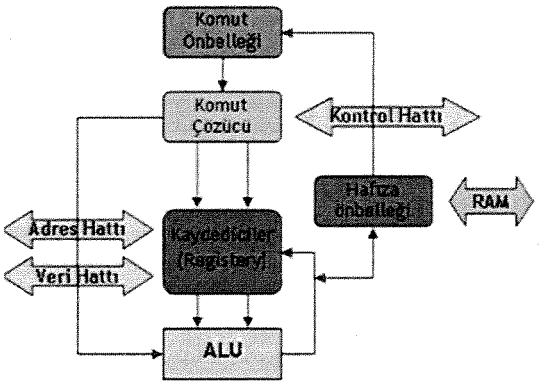
1. ALU(Aritmetik ve Mantıksal İşlem Birimi): Toplama çıkarma, çarpma, bölme, mantıksal ve, veya, değil komutları ve kaydırma komutları.

2. Komut Çözücü(Instruction Decoder): İşlemcinin yapması gereken kodların icrası için gerekli işlemleri başlatır ve komutun çalıştırılması için gerekli işlemleri belirler.

3. Kaydediciler(Registery): İşlemci içerisinde sayıları depolamak için kullanılan hafıza çeşididir. İşlemci veri uzunluğu kadar genişliğe(32, 64 bit) sahiptirler. Literatürde test, EBX, EAX, BX, ES, IP gibi isimler alan kaydedici hafıza gözleri vardır.

4. Bayraklar(Flags): İşlemlerin sonucuna göre 1 ya da 0 değerlerini alan 1 bit genişliğe sahip hafıza gözleridir. Sıfır, işaret, elde, eşlik, taşma gibi çeşitleri vardır. Örneğin bir çıkarma işleminde sonuç sıfır çıkarsa sıfır bayrağı 1 değerini alır.

5. Veriyolları(Buses): İşlemcinin diğer donanım birimleri ile bağlantısını sağlayan iletken elektriksel yollardır. Üç adet veriyolu bulunur. Bunlar veri(data), adres(address) ve kontrol(control) veriyollarıdır.



94

- Bilgisayarda genelde üç tip bilgi iletişimi yapılır.
- Bunlar; veri (data) iletişimi, işlemin kontrol (control) edilmesi ve durumun bir bilgi ile (Status data) belirlenmesidir.
- Kontrol işaretleri bilgisayarın farklı üniteleri arasında bir işlem yapıldığı sırada kullanılır. Status (durum) işaretleri ile merkezi işlemcinin durumu ve ne iş yaptığı açıklanır. yani veri giriş mi yaptığı yoksa veri çıkışı mı oluyor ya da kontrol mu yapılıyor gibi.
- Bir merkezi işlemci işleyeceği bilgileri bellekten alarak işler, daha sonra işlenen bilgileri belleğe ya da portlara bilgi olarak geri gönderir. Veri işlemlerinin dışında ayrıca mantıksal ve matematiksel operasyonları gerçekleştirir.
- CPU bir işlemi gerçekleştirirken kontrol, adres ve veri yolarını kullanır ve CPU'nun o anki durumu; işlem durum kaydedicisinde yani Status Register'da görülür.
- Bir mikroişlemcinin iki tane çalışma frekansı vardır. Bunlardan biri; iç frekans (internal), diğeri ise;dış frekans (external). İç frekans, işlemcinin saniyede yaptığı işlem miktarını tanımlar. Dış frekans ise; işlemcinin bellek ve çipset gibi diğer bileşenlerle haberleştiği frekanstır. Buna veri yolu hızı ya da FSB de denilmektedir. İç frekans her zaman dış frekanstan çok daha hızlıdır.

İşlemciler

- 1- Klasik İşlemciler: Komutlar zaman üzerinde sırayla işlenir. Bir komutun işlenmesi biter. Sonraki komut işlenmeye başlar.
- 2- Pipe-Line İşlemciler: Bir komutun işlenmesi süreci devam ederken sonraki komutun işlenmesi için hazırlık yapılır.
- Bir komutun icra süreci:Fetch, Decode, Execute, write back

Saat Frek	1	2	3	4	5	6	7	8	9	10	11	12
1. Komut	F	D	E	WB								
2. Komut					F	D	E	WB				
3. Komut									F	D	E	WB

Pipeline olmayan yapıda iş akışı

1. Komut	F	D	E	WB			
2. Komut		F	D	E	WB		
3. Komut			F	D	E	WB	
4. Komut				F	D	E	WB

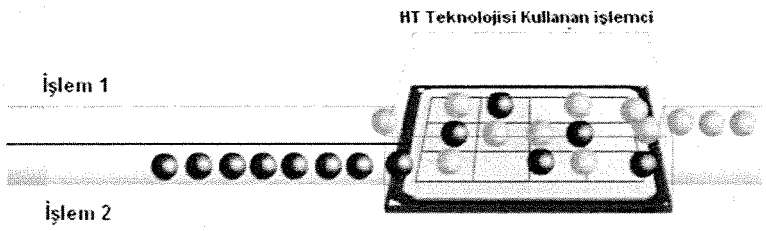
Pipeline yapıda iş akışı

- Örneğin 5 ve 6 sayılarının toplanması ve ekrana yansıtılması işleminin işlemci birimleri tarafından şu şekilde gerçekleşir:
  1. Komut alınır : örneğin 5 rakamı belleğin 12345 adresinden alınır.
  2. Komut çözülür.
  3. Komut çalıştırılır : ALU sayıyı bulur.
  4. Komut saklanır : 5 rakamı belleğe geçici olarak saklanır.
  5. 1, 2, 3, 4 adımları aynen 6 rakamı için de uygulanır.
  6. Komut alınır : toplama işlemi çalıştırılır.
  7. Komut çözülür.
  8. Komut çalıştırılır : ALU 5 ve 6 sayılarını toplar
  9. Komut saklanır : hesaplanan sonuç geçici olarak belleğe saklanır.
  10. Komut çalıştırılır : ekranda görüntüleme emri iletilir.
  11. Komut çözülür.
  12. Komut çalıştırılır : sonuçta hesaplanan sonuç ekranda gösterilir.

İŞLEMCİLER

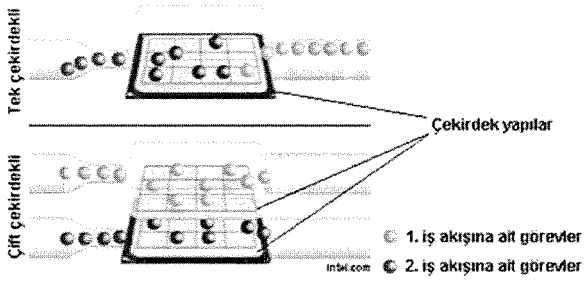
3-HT (Hyper threading) İşlemciler: Hyper-Threading (Çoklu iş parçacıkları) teknolojisi, tek bir fiziksel işlemcinin çok sayıda komut zincirini (farklı programları iş parçacıkları şeklinde bölerek) eş zamanlı olarak işlemesi ile performans artışı sağlamasıdır. Hyper-Threading teknolojisine sahip olan bir işlemci, mantıksal olarak iki adet işlemciden oluşmaktadır. Çip üzerinde tek bir işlemci bulunmasına rağmen programları zaman paylaşımlı olarak çalıştırabilir.

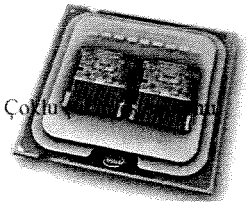
Not: HT özelliğine izin veren işletim sistemleri ile HT özelliği olmayan işlemcilerle de farklı programlar eşzamanlı olarak çalıştırılabilir. Fakat performans çok iyi değildir.



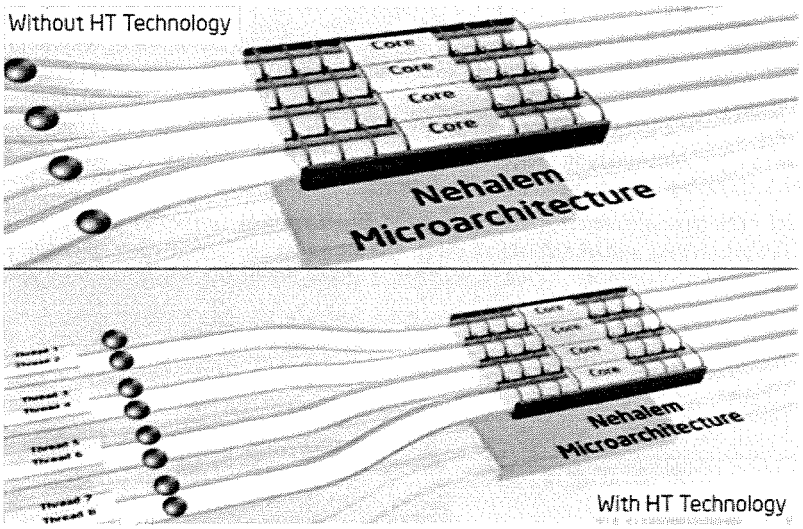
İşlemciler

4- Çok çekirdekli İşlemciler (Örn:Dual Core İşlemciler): İşlemci paketi içerisinde birbirinden bağımsız olarak komutları çalıştırabilen her yapıya çekirdek ismi verilmektedir. Gerçek zamanlı olarak kendisine verilen iş akışı görevlerini aynı anda yerine getirerek performansı artırır. Her çekirdek birbirinden bağımsız FSB ye sahiptir. Tüm çekirdekler L2 yi ortak kullanırlar. Günümüzde 2 ve 4 çekirdekli işlemciler bulunmaktadır. Intel Pentium Dual Core, Intel Pentium Core 2 Duo, Intel Quad Core ve AMD Athlon X2 serisi çok çekirdekli yapıya sahiptir.





99



100

İşlemci ile ilgili Parametreler

- **1. Hız:** Birimi frekans olarak GHz katsayısı(10003) ile değerlendirilir. İşlemcinin hızlı olması işlemlerini daha kısa sürede tamamlaması anlamına gelmektedir. Günümüzde 1.8, 2.0, 2.2, 2.4, 2.6, 3.0, 3.2, 3.6, 3.8 GHz hızlarında olanları vardır.
  2. Bit Genişliği: İşlem yapabilme boyutunu gösterir. Günümüzde 64 ve 32 bit işlemciler vardır. İşlemcinin sahip olduğu kaydediciler, veri hattı ve adres hattının genişliğini gösterir.
  3. FSB Hızı: İşlemcinin, kuzey köprüsü ile iletişim hızını gösterir. Günümüzde 1333, 1066, 800, 533 Mhz değerlerine sahip işlemciler vardır.
  4. Level2(L2) Cache: İşlemciye yakınlığından dolayı bu isim verilir. En yakın olana L1, diğerine L2, L3 gibi isimlendirmeler kullanılır. Yapısı SRAM hafıza tipindedir. SRAM hafızalar daha hızlı fakat maliyetleri yüksek hafıza çeşitleridir. Yavaş olan RAM erişimlerini azaltmak için işlemci içerisinde yer alan daha hızlı fakat küçük boyutlardaki hafızaya verilen isimdir. İşlemcinin hafıza kontrol devresinden(MCH) istekte bulunduğu her kod bu belleğe yazılır. İşlemci aynı kodu RAM yerine daha hızlı olan bu bellekten alır.
  2x1MB, 2MB, Core2 Dualarda 4MB, Core2 Quad larda 8MB ve Celeron larda 1MB veya 512 KB L2 bellek miktarı vardır. RAM'den tipik olarak 4 kat daha hızlı çalışmaktadır.
  5-Silikon teknolojisi: Bir nanometre metrenin milyarda biridir. Piyasada şu anda bulunan işlemcileri oluşturan komponentler arasındaki mesafe bir metrenin 45 milyarda biri genişliğindedir (45nm teknolojisi). Transistörlerden 2000 tanesini yan yana koyduğunuzda bir insan saç telinin çapı (0.09 mm) kadar bir genişliğe ulaşılır. Silikon dioksit (1960'lardan bu yana kullanılmaktadır) yerine hafniyum oksit kullanımıyla yeni transistörlerde daha az enerji kaybı, daha az ısınma ve daha hızlı geçişler sağlanır.
-

101

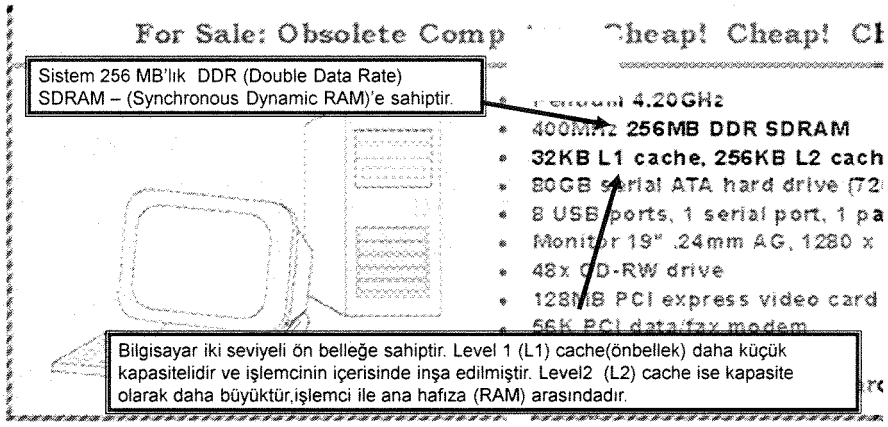
İşlemci temel özellikleri -örnek

	INTEL® CORE™ i7-980X PROCESSOR EXTREME EDITION
Number of Processor Cores	6
Number of Simultaneous Threads with Intel® Hyper-Threading Technology[2]	12
Intel® Smart Cache	12 MB
Processor Base Frequency	3.33 GHz
Integrated Memory Controller	Yes
Intel® Turbo Boost Technology[1]	Yes
Number of DDR3 Memory Channels	3
Intel® Express Chipset	X58
Overclocking Enabled[3]	Yes

102

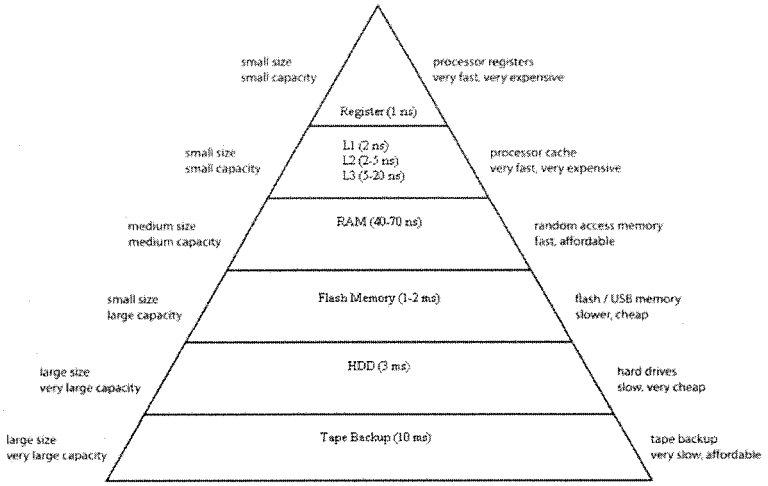
- Büyük ana bellek kapasitesi olan bilgisayarlar, daha büyük programları, daha küçük ana bellekli bilgisayarlara göre daha hızlı çalıştırırılar.
- RAM rastgele erişimli bellek için kullanılan bir kısaltmadır. Rasgele erişim, hafıza uzayındaki herhangi bir yerdeki bilgiye aynı sürede erişilebilir anlamındadır.
- Önbellek (Cache), RAM'den daha hızlı erişilebilir geçici bir bellek türüdür.

103



104

Computer Memory Hierarchy



RAM (Random Access Memory- Rasgele Erişimli Bellek)

- Bilgisayarda verilerin geçici olarak depolandığı hafıza birimidir. **CPU**'nun ihtiyaç duyduğu kodları sakladığı için hızlı ve kapasitesinin çok olması performansa doğrudan etki edecektir. Rasgele erişimli olması sayesinde belleğin sıra gözetmeksizin istenen adresindeki veriler okunup yazılabilir.
- Disk, **CDROM** ve **I/O** portlarından gelen ve giden veriler geçici olarak RAM üzerinde işlenmek üzere tutulur. RAMler üzerindeki bilgiyi tutabilmeleri için elektrik enerjisine ihtiyaç duyarlar. Elektrik enerjisi kesildiğinde üzerlerindeki verilerde kaybolur.
- **RAM**, her bir hücresi bir transistor ve kapasitordan oluşan 2 boyutlu(satır ve sütunlardan oluşan) matris yapıya sahiptir. Hücreler şarj edilirken hücre hücre değil, satır satır edilir. Verileri tazeleme oranı(refresh rate) satır bazında değerlendirilir. Örneğin **2K tazeleme oranına** sahip bir bellekte 2048 adet satır tazeleniyor demektir.
- Günümüzde **DDR SDRAM(double-data-rate synchronous dynamic random access memory)**, DDR2 SDRAM, DDR3 SDRAM ve SDRAM yapıda hafıza ürünleri kullanılmaktadır.
- Daha **ucuz ve küçük boyutta olduğundan** bilgisayar **ana belleğinde DRAM** kullanımı tercih edilirken **SRAM hızından** dolayı öncelikle **cache bellek** için kullanılır.

- **SDRAM(Synchronous Dynamic RAM):** 64 bit veri genişliğine sahiptir. Günümüz bellek yapılarından en az band genişliğine sahiptir. DDR çeşit RAM' lerin temelini oluşturur ve onlardan en az iki kat yavaş çalışır. Dinamik bellek yapısındadır. 168 pine sahiptir.

  DDR SDRAM: 64 bit veri genişliğine sahiptir. Veri transferi için, saat(clock) işaretinin alçalan ve yükselen kenarlarını kullanan yapıya sahip, SDRAM çeşididir. Dolayısıyla SDRAM' e göre iki kat hızlıdır. 184pin yapıdadır.

- **DDR2 SDRAM:** 64 bit veri genişliğine sahiptir. DDR SDRAM ile aynı yapıda olup aynı saat hızında çalışırlar. Aralarındaki fark latency(istenen adrese ulaşmak için harcanan zaman) değerinin **DDR** da daha büyük olması ve daha fazla güç gereksinimidir. Ayrıca burada belleğin, I/O bus frekansı DDR'a göre iki katı hızda çalışmaktadır. 240 pin yapıya sahiptir.
- **DDR3 SDRAM:** 64 bit veri genişliğine sahiptir. DDR yapıya sahiptir. Fakat en az güç gereksinimine sahiptir. DDR2 ye göre dahili geçici hafıza miktarı büyüktür. DDR3' ün, I/O bus frekansı **DDR2**'ye göre iki katı hızda çalışmaktadır. 240 pin yapıya sahiptir.

107

RAM Parametreleri

- **Kapasite:** Bir RAM'ın kapasitesi diye, xbaytlık kaç tane farklı adresşeme olduğudur.

  Hız: Bir işlemci RAM üzerindeki veriye ihtiyaç duyduğunda hafıza kontrol devresine(MCH=Memory Control Hub) istekte bulunur. MCH bu isteği RAM'e aktarır ve MCH veri okunmaya hazır olduğunda bunu CPU'ya rapor eder. RAM'in aldığı bi talebe karşılık vermesi için geçen süreye Access Time (Erişim zamanı) denir. Hafıza modülleri 80-50ns arasında değişen Access Time sürelerine sahiptir. Sürenin kısa olması modülün hızını gösterir.

108

6. ÖNBELLEK (Cach)

- Bilgisayar hızını belirleyen en önemli faktörlerden biri dir.
- SRAMlerden oluşmuş bellek topluluğudur.
- Çalışma şekli; işlenecek olan bilgi alınmak istendiğinde ilk defa buraya uğranır.
- Kullanım amacı; işlenecek olan bilgi işlemeden önce buraya getirilerek bilgilerin işlenmeye hazır hale getirilmesidir.
- Genellikte işlemcilerde kullanılır ve işlemcinin hızını belirleyen en önemli faktörlerden birisidir.
- Normalde işlenecek bilgilerin hepsi bilgisayarın ANA BELLEĞİN (RAM) 'de bulunur.
- İşlemci verileri işlemek için yol sistemleri ile ANA BELLEĞE gider, oradan işleyeceği bilgiyi alır ve işler. ANA BELLEĞİN ve yolların hızı işlemcinin hızına yetişemez. İşlemcinin her veri işleme ihtiyacında ANA BELLEĞE gitmesi zaman kaybına yol açar ve işlemci yeteri kadar verimli kullanılmaz. İşte bu yavaşlığı önlemek için önbellekler kullanılır.

1.3 An Example System

Sabit disk kapasitesi, saklayabileceğiniz veri ve programlarınızın boyutunu belirler.

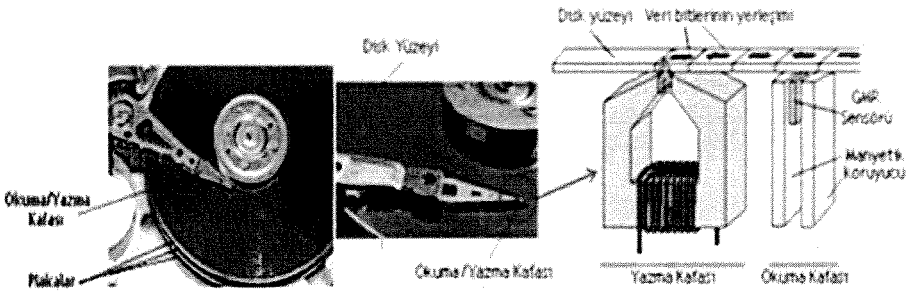
- 400MHz 256MB DDR SDRAM
- 32KB L1 cache, 256KB L2 cache
- **80GB serial ATA hard drive (7200 RPM)**
- 8 USB ports, 1 serial port, 1 parallel port
- Monitor 19" .24mm AG, 1280 x 1024 at 75Hz
- 48x CD-RW drive

Bu 80GB'lık veriyi saklayabilir.. 7200 RPM, diskin dönüş hızıdır. Genelde,disk dönüş hızı, daha hızlı işlemdir. Bu her zaman doğru değildir..

110

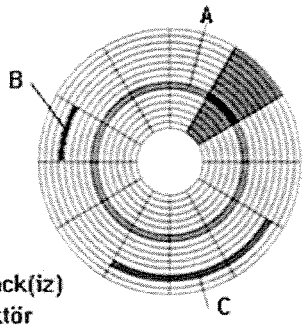
Sabit diskler (Harddiskler)

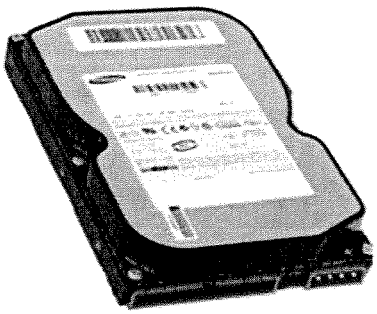
- Sabit diskler, üzerlerine kalıcı olarak bilgi depolanabilen ve depo edilmiş verilerin okunabildiği manyetik ortamlardır. Sabit disklerin bilgi depolama kapasitesi disketlerden çok yüksektir, Günümüzde 80 -120 , 256 512 GB, 1-2 TB.lık sabit diskler yaygın olarak kullanılmaktadır.
- Sabit disklerde bilgi kaydedilen bölüm (disk plakaları-bunların üsüste gelmesiyle silindir oluşur), mıknatıslanmayan metal bir yüzey (alüminyum) üzerine kaplanmış demir grubu (demir, nikel, kobalt vb.) oksitlerden oluşan bir tabakadır.



111

- Sabit disk üretildikten sonra biçimlenmeden (formatlanmadan) kullanılamaz. *Biçimleme işlemi yapılınca disk yüzeyinde iç içe daireler şeklinde* ***izler (track)*** *oluşturulur*. Bunlar gerçekte disk üzerine çizilmiş izler olmayıp bilgilerin adresleme biçimidir. Disk üzerin oluşturulan izler bilgilerin kolayca bulunmasını sağlamak için **sektör (dilim)** lere bölünür.
- **Kümeler(Cluster)** ise İşletim sisteminin disk yönetimi ile alakalıdır. Dosya ve dizinlerin yerleştirildiği en küçük disk alanına denir. Boyutu dosya sistemine göre değişir.





A. Track(iz)
B. Sektör
C. Cluster

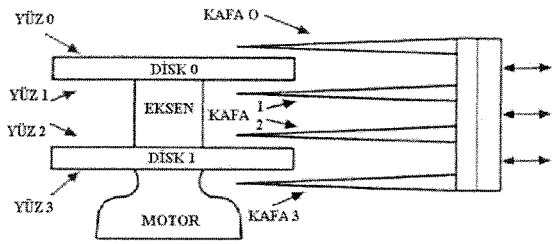
112

Disk Parametreleri

- **Disk Dönüş Hızı(RPM=Rotate Per Minute):** Diskin dakikadaki dönüş hızını gösteren bir parametredir. Günümüzde 15.000, 10.000, 7200, 5400rpm değerlere sahip diskler bulunmaktadır.
  Tampon Bellek(Cache veya Buffer): Disk erişimi bellek erişiminden daha yavaş olduğu için disk performansını artırmak için HDD üzerine bellek hafıza birimleri yerleştirilmiştir. Amaç erişilmesi öngörülen verileri bellekte hazır bekletmektir. Eğer istenen veri bellekte varsa disk erişimi olmadan veriler doğrudan bellek üzerinden gönderilir.

  Konumlanma Süresi(Seek Time): Disk üzerinde okuma yazma kafasının, istenen adrese yazma veya okuma amaçlı ulaşmak için harcadığı süredir. Kafanın disk üzerindeki konumuna göre bu süre kısalıp uzayabilir. Bunun için ortalama konumlanma süresinden(average seek time) bahsedilir. Kısa olması diskin okuma ve yazma performansının daha iyi olduğunu gösterir. HDDLerde erişim süresi ms'ler mertebesindedir.

  Kapasite: Depolayacağı veri miktarını gösterir. Günümüzde 750 GB, 1TB ve hatta 4TB kapasiteli diskler bulunmaktadır.



Şekil : Okuma-Yazma Kafaları ve Diskler

- Spin Up: işletme hızına diski hızlandırma zamanı.
- Seek Time: İstenen Track'ı bulma zamanı
- Rotational Time: istenen sektörü bulma zamanı
- Transfer Time: Veri okuma veya yazma zamanı.
- Bir sabit diskin kapasitesi şu şekilde hesaplanır.
- Silindir sayısı*Sektör Sayısı*kafa sayısı*512'dir. 1024 silindir, 256 kafa ve 63 sektör parametrelerine sahip bir sabit diskin kapasitesi: 1024*256*63*512=845571864 Byte'dır. Bu da yaklaşık 8.4 Gigabyte'dır.

IDE(integrated Drive Electronics), işlemciyle Hard disk arasında veri akışının kontrolunu sağlayan eski bir standarttır. SATA (Serial *Advanced Technology Attachment)* ise IDE'nin yerini alan bir hafıza arayüz standartıdır.

- 32KB L1 cache, 25
- **80GB serial ATA** hard drive (7200 RPM)
- 8 USB ports, 1 serial port, 1 parallel port
- Monitor 19" .24mm AG, 1280 x 1024 at 75Hz
- **48x CD-RW drive**
- 128MB PCI expres eo card
- 56K PCI data/fax n n

A CD yaklaşık 650MB datayı hafızalar. Bu sürücü yazılabilir CDs, CD-RW'leri destekleyebilir. 48x sürücünün hızını açıklar. X her bir saniyede153,000 byte'lık data demektir.

115

1.3 An Example System

Port'lar bir sistem ve onun çevresel birimleri arasında verinin alınıp verilmesini sağlar.

- Cheap! Cheap! Cheap!
- um 4.20GHz
- 400MHz 256MB DDR SDRAM
- 32KB L1 cache, 256KB L2 cache
- 80GB serial ATA hard drive (7200 RPM)
- **8 USB ports, 1 serial port, 1 parallel port**
- Monitor 19" .24mm AG, 1280 x 1024 at 75Hz
- 48x CD-RW drive
- 128MB PCI express video card
- 56K PCI data/fax modem
- 64-bit PCI sound card
- Integrated 10/100 Ethernet card

Bu sistem 10 tane porta sahiptir.

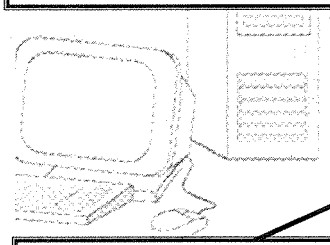
116

- Serial portlar ile, veriyi oluşturan bitleri 1,0 sinyal katarı ardarda (herbir clok süresince tek bit) olarak iki nokta arasında tek bir iletişim ortamı ile gönderebilirsiniz.
- Parallel portlar ile bir baytlık(veya daha fazla) bilgiyi tekbir zaman diliminde karşı tarafa gönderebilirsiniz. Bit sayısı kadar iletişim yolu olmalıdır.
- USB (Universal Serial Bus) : Kendisini konfigüre edebilen akıllı bir seri arayüzdür. Tak-Çalıştır'ı destekler.

117

1.3 An Example System

Sistem BUS'ları, I/O BUS'ları olarak düzenlenebilir. PCI, *(Peripheral Component İnterface)*, bunlardan biridir.



ap! Cheap! Cheap!

0GHz
MB DDR SDRAM
che, 256KB L2 cache
80GB serial ATA hard drive (7200 RPM)
8 USB ports, 1 serial port, 1 parallel port
Monitor 19" .24mm AG, 1280 x 1024 at 75Hz
48x CD-RW drive
- **128MB PCI express video card**
- **56K PCI data/fax modem**
- **64-bit PCI sound card**
- Integrated 10/100 Ethernet card

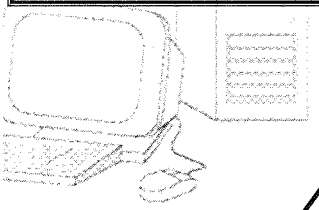
Bu sistem 3 adet PCI BUS'a sahiptir. Devices: Bir video kart, Bir ses kartı, ve bir adet data/fax modem.

118

Bir monitorde, bir görüntünün saniyede kaç kez tekrar oluşturulma sayısı onun yenileme oranını *(refresh rate) ifade eder.* Bir monitorun piksel (nokta) aralığı, görüntünün netliği ile orantılıdır.

Cheap!

Bu monitörün piksel(dot) aralığı .24mm'dir. Görüntü yenileme oranı 75Hz'dir.



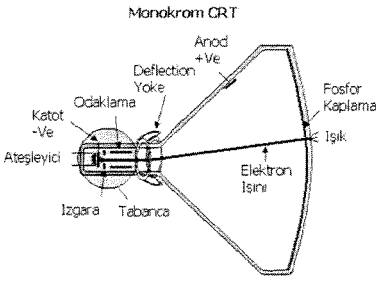
- 8 USB ports, 1 serial port, 1 parallel port
- **Monitor 19" .24mm AG, 1280 x 1024 at 75Hz**
- **48x CD-RW drive**
- **128MB PCI express video card**
- 56K PCI data/fax modem
- 64-bit PCI sound card
- Integrated 10/100 Ethernet card

Video Kartı 128Mb hafıza bulundurur.

119

Monitör

- Katot Işınlı Tüp (CRT, Cathode Ray Tube) denilen bu tüp, havası boşaltılmış mühürlenmiş bir cam konidir.
- Koninin geniş tarafı düz ve dikdörtgen biçimindedir. Bu, monitörün ekranını oluşturan taraftır. Koninin öbür tarafı dardır ve katot levhaları ile küçük tel ızgaraları içerir. Bu katot levhaları ısıtıldığında tüpün içinde serbestçe dolaşan elektron bulutları oluştururlar.
- Katot negatif olarak yüklenirken,dış kısmına pozitif bir yüksek gerilim (26000V) uygulanarak katot ışınlı tüpün anodu veya pozitif kutbu oluşturulur. Anot ve katot arasındaki büyük gerilim farkı, bu serbest elektronların ekrana doğru fırlatılmalarına neden olur.



- **Çözünürlük (Resolation):** Çözünürlük, ekran kartının gönderdiği görüntüyü dikey ve yatay olarak ifade ederken kullandığı piksel sayısıdır.
- **Boyut:** Ekranın fiziksel uzunluğunu ekranın boyutları belirler. 14 inc`lik bir monitörün genişliği yaklaşık olarak 11 inc`tir (28 cm).
- **Tazeleme Oranı (Refresh Rate):**Elektron tabancası, CRT`nin noktacıklarını üst sıradan başlayarak en alt sıraya kadar tarar. Bundan sonra elektron tabancası, grafik kartından kendisine gelen diğer ekran görüntüsünü CRT üzerinde oluşturmak üzere tarama işlemine devam eder. Bir tam ekran tarandıktan sonra yeni bir görüntü oluşturmak üzere tarama işleminin başlamasına ekranın tazelenmesi denir. Bu tazeleme olayının saniyede kaç kez tekrarlandığı ise tazeleme hızıyla ölçülür. Saniyede 60 Hz`lik tazeleme hızına sahip olan bir monitör, o sırada saniyede 60 ekran taramaktadır.
- Bir monitörün satır sayısı ile düşey tarama frekansının çarpımı yatay tarama frekansını verir. Bu, elektron demetinin, ekranın solundan sağına saniyede kaç defa gidileceğini gösterir. Buna göre 480 satır çözünürlüğünde ve 70 Hz düşey tarama frekansına sahip monitörün; 480*70 veya 33600 Hz (33.6 KHz)`dir. Bu durumda elektron demeti saniyede 33600 satırı tarayacaktır.

- Bir monitörün satır sayısı ile düşey tarama frekansının çarpımı yatay tarama frekansını verir. Bu, elektron demetinin, ekranın solundan sağına saniyede kaç defa gidileceğini gösterir. Buna göre 480 satır çözünürlüğünde ve 70 Hz düşey tarama frekansına sahip monitörün; 480*70 veya 33600 Hz (33.6 KHz)`dir. Bu durumda elektron demeti saniyede 33600 satırı tarayacaktır.
- **Nokta Aralığı (Dot Pitch):**Aynı renkteki iki nokta (dot)`nın, merkezleri arasındaki uzaklık nokta aralığı (dot pitch) olarak adlandırılır. Nokta aralığının bugünkü değerleri 0.25 mm ve 0.28 mm arasında değişir. "Nokta aralığı ne kadar küçük olursa, daha net görüntü anlamına gelir."

LCD Liquid Crystal Display

- LCD monitörlerde görüntü sıvı kristal diyotlar yardımıyla sağlanmaktadır. Bu diyotlara gerilim uygulandığında, içlerindeki moleküllerin polarizasyonu değişmekte ve beraberinde de diyodun geçirgenliği değişmektedir.
- Normalde şeffaf olan bu diyotlara gerilim uygulandığında geçirgenliklerini kaybederler ve siyaha dönerler. Renkli LCD monitörlerde ise çok ufak ve birden fazla diyot kamanı kullanılarak görüntü alınmaktadır.
- LCD monitörler DSTN ve TFT olmak üzere ikiye ayrılmaktadır. Ucuz olan ve "passive matrix" teknolojisini kullanan DSTN (Dual-Scan Twisted Nematic)'ler çözünürlükleri ve görüş açıları TFT'lerden düşük olan monitörlerdir. Bu monitörler genelde dizüstü bilgisayarlarda kullanılmaktadır.
- TFT (Thin Film Transistor)'ler ise "active matrix" adı verilen ve görüntüyü daha parlak ve keskin gösteren bir teknoloji kullanırlar. TFT'lerde her piksel bir ya da dört transistör tarafından kontrol edilir ve bu sayede flat panel ekranlar arasında en iyi çözünürlüğü sunarlar.

Peki bu bileşenleri üreticiler kendi standartlarına göre üretebilirler mi?

-Bilgisayar bileşenlerinin, beklediğimiz gibi çalışacağının bir güvencesi varmı?

-Bilgisayar bileşenlerinin birlikte çalışacağının ne güvencesi var?

Bu belirsizliği ortadan kaldırmak ve Bilgisayar bileşenlerinin birlikte çalışabilirliğini sağlamak için - bilgisayar donanım standartlarını belirleyen birkaç önemli organizasyon vardır.

-The Institute of Electrical and Electronic Engineers (IEEE)

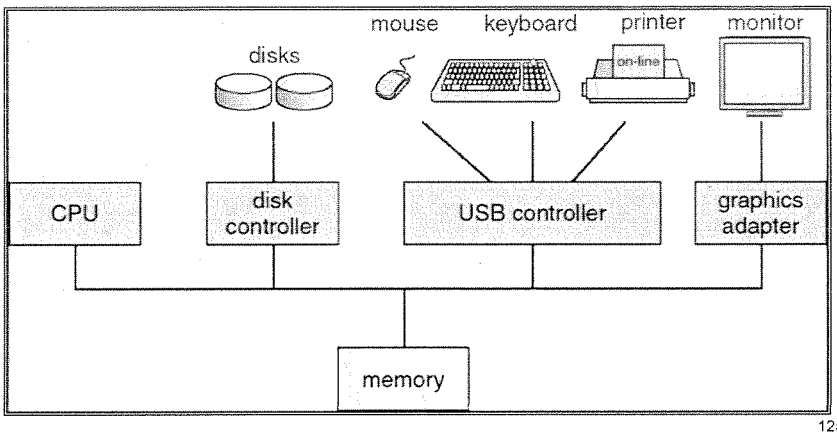
-The International Telecommunications Union (ITU)

-The American National Standards Institute (ANSI)

-The British Standards Institution (BSI)

-The International Organization for Standardization (ISO)

Bu bileşenlerin nasıl çalıştığını ve bunların biribirleriyle nasıl etkileştiği ve etkileşim yazılımı ile komple bilgisayar sistemlerinin nasıl kontrol edilebileceği, organize edilebileceği bundan sonraki konularımızda zaman zaman açıklanacaktır.



125

2.Bölüm
Komutlar: Bilgisayar dili
Instructions: Language of Computers

"Ben, Tanrıyla İspanyolca, kadınlarla İtalyanca, erkeklerle Fransızca, karımla Almanca konuşurum."

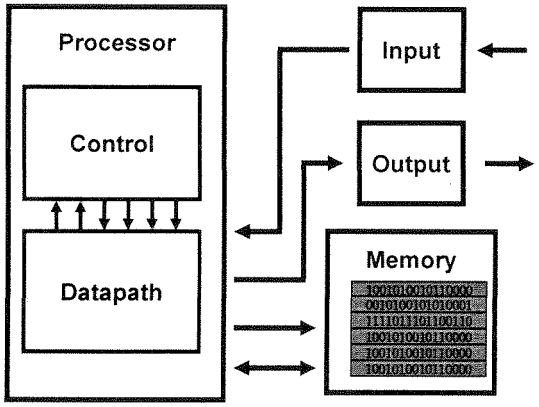
V.Charles – Fransa Kralı 1337-1380

"Geç bunları kralım; ben tek dil konuşurum anlayan geçer. Anlamayan seneye gelir"

D.A.B.D.Başkanı Hayrettin CAN – 2011

Bilgisayar Sistemi

- Bir bilgisayarın işlemcisi büyük bir lojik devredir.



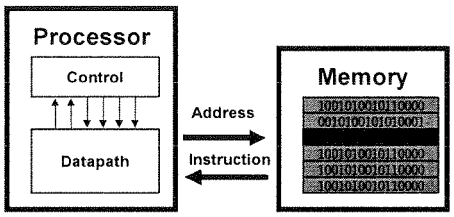
2

İşlemci işlemi nasıl yapar?

- The "fetch/execute- (Alma(getirme) / yürütme)" süreci
  - İşlemci komutu, hafızadan getirir (fetches).
  - İşlemci komutu "makine dilinde " yürütür (executes).

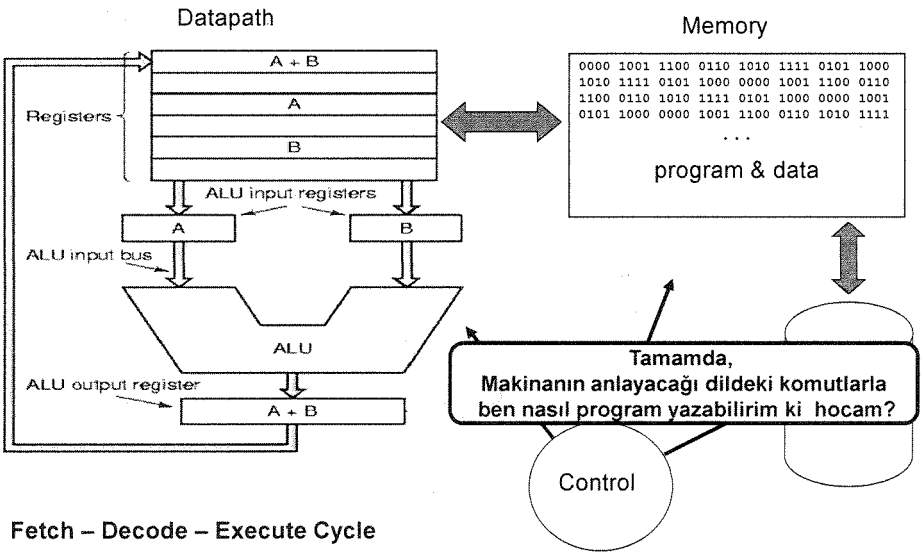
Yeni komut

Komutun yüklenmesi ve çözülmesi
İşlenenin (Operand) getirilmesi
Operand'ın işlenmesi
Sonuçların kaydedilmesi



3

İşlemin yapılması -II



Fetch – Decode – Execute Cycle

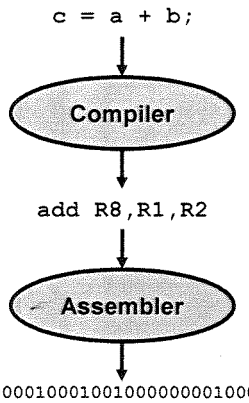
Bilgisayar sistemlerinde Soyutlama (biribirinden ayırma)
Abstractions in Computer Systems

- Yüksek seviyeli bir dilde kodlanmış komutları makinanın işlemesi ve tekrar geri döndürmesi süreci oldukça karmaşık bir süreçtir.

- Tasarımcılar; karmaşıklığı yönetmek için soyutlama (ayrıştırma) tekniğini kullanırlar.
  - Sadece ilgili bilgilere odaklanılır.
  - Gereksiz ayrıntılara hiç bakılmaz.

Yazılımsal Ayrıştırma - Diller

Yüksek Seviyeli dil (C)



Compiler: Yüksek seviyeli dilde yazılmış programı assemler sembolik dil komutlarına çevirir. Yüksek seviyeli dildeki birkaç satırlık program assemler dilinde çok daha fazla sayıda komut içerir.

Assembler dili : Bir sembolik komut kodunu binary forma çeviren bir programdır. (Binary koddaki komutların sembolik gösterimini yapan dil)

Makine dili : Bilgisayar sistemiyle haberleşebilmek için , komutların binary formda oluşturulmuş şeklidir.

6

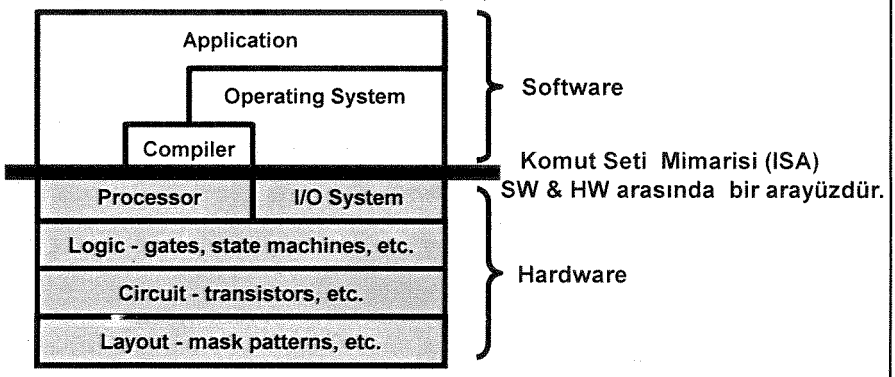
Yazılımsal ayrıştırma - Sistem Yazılımı

- İşletim Sistemi
  - Alt-seviye detayları programcıdan izole eder.
    - Sistem kaynaklarını yönetir.
    - Dosya sistemlerini yönetir.
  - Birden fazla programın işletilmesini koordine eder.
  - Kullanıcı programlarının sisteme zarar vermesini önler.
- **Kütüphaneler**
  - Üst düzey primitiv programlara , programcının erişimini sağlar.
  - İyi tanımlanmış arayüzler ile programlara erişim sağlanır(API).
- **Uygulamalar:** kullanıcı için özel fonksiyonları başarmak içindir.
  - Web Browser
  - Tablo
  - Word Processor

7

Komut Seti Mimarisi - Instruction Set Architecture (ISA)
(Donanım-Yazılım Arayüzü - The Hardware-Software Interface)

- Bilgisayarın en önemli ayrışımıdır



8

2.1. Giriş

- Bir bilgisayarı, donanımsal (hardware) olarak yorumlamak için onun dilini bilmelisiniz. Bir bilgisayar dilinin kelimelerine komut *(instruction)* denir. Ve genellikle bu kelimelerin bütününe *komut seti* diyeceğiz. Bu komutlar;

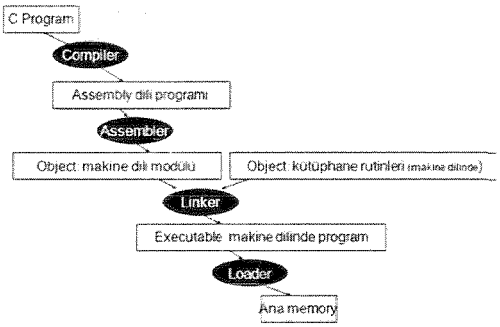
  - Kullanıcı tarafında: assembler dili
  - Bilgisayar tarafında : Makine dili

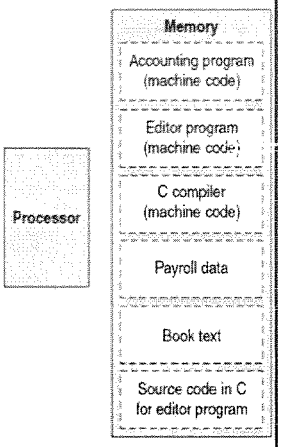
- Bir komut setinin öğrenilmesi demek;
  -Yüksek seviyeli programlama dilleri (C dili) ile ilişkisi nedir?
  -Yüksek seviyeli dille yazılmış komutu makine nasıl anlar?
  -Donanımsal olarak komutun gereği nasıl yapılır?
- Örneklerimizde yüksek seviyeli dil olarak C programlama dilini kullanacağız.

Komut seti mimarisi (ISA) SW/HW arayüzü olarak tanımlanabilir.

- 1980'lerden beri popular olarak kullanılan *MIPS (Mıllıon Instructıon of per Second)* komut seti mimarisini (Instruction Set Architecture – ISA) kullanacağız.
- MIPS komutlarını zamana yayarak adım adım öğreneceğiz.
- Komutların ve farklı tipten verilerin, hafıza biriminde sayı değerleri olarak saklandığını varsayan "Stored program consept" konseptini kullanacağız.

Stored program consept(Hafızalanmış Program Konsepti)





Günümüz bilgisayarları 2 önemli özelliği gözeterek inşa edilirler.

1-Komutlar, sayı sembolleri ile gösterilir.

2- Programlar ana hafızada kaydedilir. Ordan sayı sembolleri olarak okunur veya yazılır. Bu prensipler "stored-program" konseptidir.

Özel olarak; Hafıza, makine kodu oluşturan bir editör programının kaynak kodlarını ihtiva eder. Bu editör programı , ilgili yüksek seviye dilde yazılmış kodların makine kodu karşılığını text olarak oluşturur. Hatta bu editörde derlenmiş makine kodları doğrudan kullanılabilir.

ISA'nın temel prensipleri

- Komut seti mimarisinde (ISA) önemli dizayn prensipleri her zaman ön plandadır. Bunlara uyulunca donanımsal mimari optimize bir şekilde oluşur.

1-Basitlik düzenlilikten yanadır. (*simplicity favors regularity)*: Komutların belirli bir düzene göre işlenmesi Çözümü basitleştirir. Donanımın basitleşmesi için komutların tipini ve biçimini basitleştirtmek için kullanırız.

2- En küçük en hızlıdır (*smaller is faster):* İşlemciler sadece basit ilkellerin (primitive) uygulanmasını gözetmeli. Yani basit birimlerden oluşmalı. Register sayısı, ALU işlemlerinin sayısı, v.b donanım birimlerini doğru boyutta seçerken önemlidir.

3- *Make the common case fast : Çok kullanılanları hızlı yapın.*

4: Good design demands good compromises : İyi tasarım , iyi tavizler (uzlaşmalar) verilerek oluşur.

2.2 Bilgisayar donanımının İşlemesi
MIPS Komut Seti

- Herbir bilgisayar muhakkak olarak aritmetik işlemleri gerçekleştirebilmelidir.
  C kodu: a = b + c ;
- **Assembler kodu: (İnsanın anlayacağı makine komutları formu)**
- add a, b, c # b ve c'yi toplayıp a operand'ına koyar.
- **Makine kodu: (Donanımın anlayacağı makine komut formu)**
- 00000010001100100100000000100000
- ***MIPS'te her komut 3 operand'a (işlenen değer) sahiptir. Ve bir birim zamanda sadece 1 işlem başarılır.***
- ***Operandların sırası sabittir . (İlk operand hedef operand'dır.)***
- ***a, b, c aslında birer register (Kayıtcı)'dır. Registerlerin bit olarak kayıt kapasitesi, işlenenlere yazılacak değerler için çok önemlidir.***

- **Aşağıdaki C kodunu assembler koduna çevirelim.**

  a = b + c + d + e
- Bu işlem 2'den fazla sayının toplanması işlemidir. MIPS'te aritmetik komutlar 3 operand'la işlendiğinden (En basit işleme şekli budur.) bu toplama işlemi ard arda toplamalar ile gerçekleşir.
- MIPS kodu :

```
add a, b, c    # a=b+c
add a, a, d    # a=b+c+d
add a, a, e    # a=b+c+d+e
```

- #-- yorum sembolüdür. Satır sonuna kadar etkilidir.
- Bir aritmetik işlem için, operand sayısı üçtür. Olması gereken budur ve daha az, daha fazla olmaması, "donanımın basitliği" felsefesine uyar.

Örnek:

Aşağıdaki 2 program parçası için C Compiler'ın çıkışı ne olur. (t geçici operand'ı ifade eder)

- C deyimi > C Compiler > MIPS instructions

a = b + c;
d = a –e;

```
add a, b, c
sub d, a, e
```

f = (g + h) – (i + j);

```
add t0, g, h
add t1, i, j
sub f, t0, t1
```

2.3 Donanımın İşlenenleri(Operands)

- Yüksek seviyeli programlama dillerinden farklı olarak MIPS'de herbir satır sadece bir komutun icra edilmesi içindir. Yani n+1 adet toplama için n tane komut icra edilmelidir (Simplicity favors regularitiy).
- Aritmetik komutlarda, İşlenenler (Operand) kaydedilmelidir. Bunun için MIPS'e yalnızca 32 register'in sağlanması yeterlidir. (Buna karşılık intel İşlemcilerde komutların hafızadaki verilerle direkt çalışılmasına izin verilir.)
- Register'lar ALU'ya en yakın küçük kapasiteli hafızalardır. İşlemcinin içindedirler.
- MIPS'te herbir register 32 bitliktir. Ve 32 register vardır.

  Bu ***MIPS-32***'dir. (*smaller is faster- 32 register'lı yapı 33 registerLI yapıdan daha hızlıdır.)*
- MIPS-64 ise 32 adet 64-bit register anlamındadır.

REGİSTER'ler Neye Var?

• C programlamada, her bir operand (değişken) ana hafızada tutulur.

• Donanımsal olarak, her seferinde ana hafızaya erişim zaman açısından pahalıdır.

•Eğer bir a operandına tekrar tekrar erişilecekse, bunu işlemci içerisindeki bir registere bir sefer getirip üzerinde işlem yapmak daha hızlı olur.

• Komutların icrasının kolaylaşması için, her bir komutu (add, sub v.b) yalnızca bir register'da işletmemiz gerekir.

• Not: C'de işlenenlerin sayısı çok fazla olabilir. Ancak assemler'da registerlerin sayısı sabittir. Fakat çok fazla değişken, geçici registerler ile işlenebilir.

17

MIPS'te Register isimleri için kural

- 32 adet registerleri numaralarını kodlayarak komutlarda belirginleştirebiliriz.
- *$s0, $s1, $s2, ...Yüksek seviyeli programlama dillerindeki operand registerlerine karşılık gelir.*
- *$t0, $t1, $t2, ... Yüksek-seviye dil kodlarını MIPS'te kodlarken kullanılan geçici değişkenler (temporary variables) için kullanılan registerlere karşılık gelir.*

Register Standart İsimleri (MIPS-32)

Name	Register Number	Usage	Preserved on call (Çağrılarda içeriğinkorunması)
$zero	0	the constant value 0	n.a.
$at	1	reserved for the assembler	n.a.
$v0-$v1	2-3	value for results and expressions	no
$a0-$a3	4-7	arguments (procedures/functions)	yes
$t0-$t7	8-15	temporaries	no
$s0-$s7	16-23	saved	yes
$t8-$t9	24-25	more temporaries	no
$k0-$k1	26-27	reserved for the operating system	n.a.
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return address	yes

Örnek:

$$f = (g + h) - (i + j)$$

Örneğindeki f, g, h, i, j değişkenleri $s0, $s1,$s2,$s3,$s4 registerlerine atanmış olsun. Bu işlemin MIPS kodu karşılığı nedir?

Cevap:

```
add $t0,$s1,$s2 # register $t0 contains g + h
add $t1,$s3,$s4 # register $t1 contains i + j
sub $s0,$t0,$t1  # f gets $t0 - $t1, which is (g + h)-(i + j)
```

Hafıza İşlenenleri(Operands)

- *Şimdiye kadarki örneklerde birkaç değişken üzerinde işlem yapıyorduk. Bu durumda da registerlerde problem çıkmıyordu.*
- Fakat öyle işlemler vardır ki çok sayıda verinin kullanılması gerekir. Yani karmaşık ve kapasiteli bir veri bloğuyla çalışılması sözkonudur. Bu durumda işlenen sayısı register sayısından çok çok fazla olabilir.
- Bu durumda milyonlarca tane veriden oluşan veri yapılarını *ana hafızalar* üzerlerinde bulundururlar.
- Ana hafıza; yüksek seviye-programlama dili komutlarının,verilerin ve sonuçların saklandığı yerdir.
- Registerler ise, MIPS komutlarının çalıştırdığı aritmetiksel işlemlerin yapıldığı dataları bulundurur.

- Ana hafızadaki veriler, registerlare MIPS'ın içerdiği transfer komutlarıyla çağrılır. Bu komutlara *"veri transfer komutları (**lw, sw**)* denir.
- Hafızadaki bir veri kelimesine ulaşabilmek için; komut, ulaşılacak verinin hafıza adresini beslemelidir. Çünkü hafıza tek boyutlu çok büyük bir dizi şeklinde, adreslenmiş bir yapıdır.
- *Bir hafıza adresi, dizi içerisinde bir indextir (0,1,2...).*
- *"Adreslenmiş byte" anlamında, index hafızadaki 1 baytlık yeri işaret eder.*

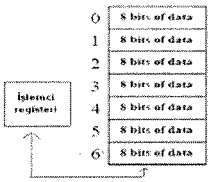
Ana hafızadaki bir operand'ın atanmasının derlenmesi- Örnek (Compiling an assignment when an operand is in memory)

- Ana hafıza dizi şeklindeki operandlar'ı (İşlenenler - veri) registerler ile ilişkilendirmek için , derleyici bellekte bir yer ayırır. Ve data transfer komutu için bir başlangıç adresini de belirler.
- A , 100 baytlık bir dizi olsun. g için $s1 registeri, h için $s2 registeri , adres bloğunun başlangıç adresi için için $s3 (başlangıç registeri) kullanılsın (Veriler 1 bayt'lık olsun).

g = h + A[8]; işlemi için

lw $t0, 8($s3) *# temporary register $to'a A[8]'in get,rilmesi.*

add $s1, $s2, $t0



Başlangıç registeri *(Base register): Dizinin başlangıç adresi*

offset: *Data transferi için sabit(8).*

***** Çoğu veri ögeleri 1 baytlık olabildiği gibi, 4 baytlık(32 bitlik) 64 baytLık veri ögesi uzunlukları da söz konusudur. 32bitlik veri elemanlarının transfer edilmesi ise aşağıdaki gibidir.*

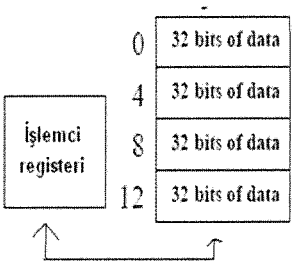
•Bir önceki problemi 32bitlik kelime uzunluğu için kodlayalım.

- Çoğu veri ögeleri 1 bayttan dah uzun kelimeler kullanır.
- MIPS'te registerler 32bit (4 bayt) veri tutarlar.

0.kelimenin byte adresi 0'dan başlar

1.kelimenin byte adresi 4

2.kelimenin byte adrei 8...



lw $t0, 32($s3) *#8x4=32*

add $s1, $s2, $t0

load ve Store (lw ve sw) kullanımının derlenmesi (Compiling Using Load and store)- örnek

C kodu: **A[12] = h + A[8];**

Bu işlem C kodunda tek bir işlem olmasına rağmen ana hafızada 2 tane işlenen (değişken) vardır. Dolayısıyla 1'den fazla MIPS komutuna ihtiyaç vardır.

MIPS kodu :

```
lw   $t0, 32($s3)    #32 = 8x4
add  $t0, $s2, $t0
sw   $t0, 48($s3)    # toplama sonucu nu ana hafızaya kaydet (store word) .
```

• Derleyicinin atamaları:
–$s3'ü başlangıç adresi için (A dizisinin başlangıç adresi)
–$s2' yi h değişkeni için.

• Hatırla!.Aritmetik işlemler registerlardaki verilerle yapılır. Hafızadakilerle değil.

Bu komut çalışmaz : **add 48($s3), $s2, 32($s3) niye?**

Sabit veya doğrudan işlenenler (değişkenler) (Constant or Immediate Operands)

- Çoğu zaman programlar, İşlemlerin bir çoğunda sabit değerler kullanır. Dolayısıyla MIPS aritmetik komutlarının yarıdan fazlası sabitlerle yapılan işlemlerdir.
- ***Açıklama Notu: Bu konuyla ilgili Donanım-Yazılım etkileşimi***

✓ ***Birçok programlarda çok sayıdaki değişken sayısı, bilgisayarın sahip olduğu sınırlı sayıdaki registerlardan fazladır. Dolayısıyla derleyici; en sık kullanılan değişkenleri registerlar'da tutmaya çalışır. Register'ler ve hafıza arasında yer değiştiren (Load ve store komutları ile) değişkenleri ise hafızanın belirli bir kısmına yerleştirir. Daha az kullanılan değişkenlerin konulduğu hafızanın belirli bölgesine " Sıçrama registerleri- spilling registers " denir.***

✓ ***Donanım prensibinden dolayı, hız ve kapasite ters orantılıdır. Dolayısıyla ana hafızanın hızı registerlere göre yavaştır.***

✓ ***Registerdaki veri daha kullanışlı, faydalı bir veridir. MIPS aritmetik komutları iki registeri okuyabilir, onlarla işlem yapabilir, onlara yazabilir. MIPS'in data transfer komutları ise sadece bir işleneni (operand) okuyabilir veya yazabilir . Onlarla işlem yapamaz.***

✓ ***Performanslı bir derleyici, registerleri etkili olark kullanan derleyicidir.***

- Biz şu ana kadar sadece komutları kullanmayı gördük. Bu komutları kullanmak için, ana hafızadan bir sabit yüklenmelidir. (Sabitler, program yüklenirken ana hafızaya yüklenirler).

• Örneğin ; C dilinde , *AddrConstant4 (programın başlangıcında* $s1 ile ilişkili hafıza yerinde olduğunu varsayalım*) adresindeki 4 sabitini* , $s3 registerine ekleyelim:

```
lw      $t0, AddrConstant4($s1)      # $t0 = constant 4
add     $s3, $s3, $t0                # $s3 = $s3+ $t0
```

Veya;

```
addi    $s3, $s3, 4                  # $s3 = $s3+ 4
```

addi *"immediate instruction - Doğrudan işlenen komut"* olarak adlandırılır. Bu komut sabitlerle çalışmaya müsaade eder. Bu komutlar donanımsal 3.dizayn prensibini gösterir.

•*Dizayn prensibi 3:* ***"Make the common case fast"***

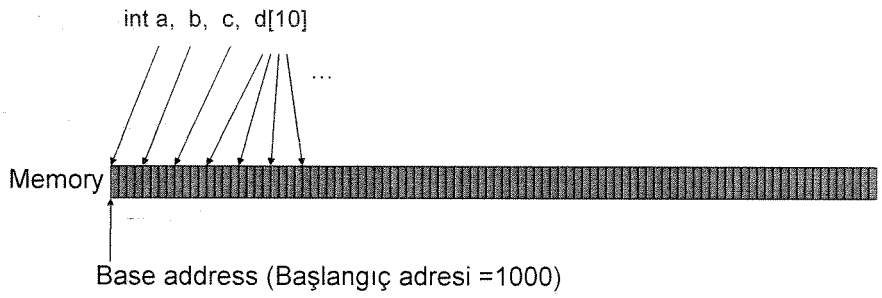
Özet

MIPS
— Adreslenmiş baytları loading/storing yapar.
— Sadece registerlerda aritmetik işlemler yapar.

	Komutlar	Anlamı
add	**$s1, $s2, $s3**	**$s1 = $s2 + $s3**
addi	**$s1, $s2, 100**	**$s1 = $s2 + 100**
sub	**$s1, $s2, $s3**	**$s1 = $s2 –$s3**
lw	**$s1, 100($s2)**	**$s1 = Memory[$s2+100]**
sw	**$s1, 100($s2)**	**Memory[$s2+100] = $s1**

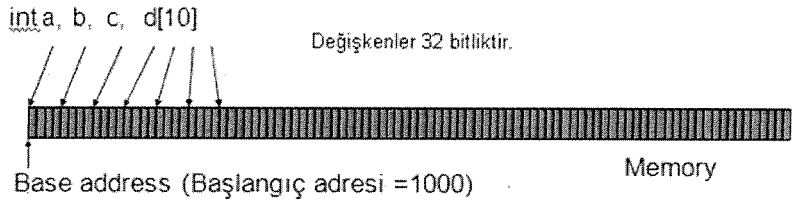
HAFIZA Adresleri (3.hafta ders başlangıcı)

- Compiler datayı(veriyi) hafızada organize eder. ...
- O her operand'ın lokasyonunu (nereye kaydedildiğini) bilir.
- Compiler, load/store komutlarının gereği için hafızanın adresi belli bölgelerini ayırıp kullanır (*spilling registers*).



29

Immediate Operands - Örnek



- Bir komut sabit bir giriş değerine ihtiyaç duyar.
- Bir immediate komut,girişlerden birini sabit bir değer olarak kullanır (Bir register operand'ı yerine).

```
addi  $s0, $zero, 1000   # programın başlangıç adresinin 1000 olduğu, bunun $s0
                         # 'a kayıt edilmesini, $zero ise devamlı 0'a eşit olan bir
                         # register olduğunun belirtilmesi.
addi  $s1, $s0, 0        # Bu a operand'ının adresidir.
addi  $s2, $s0, 4        # Bu b operand'ının adresidir.
addi  $s3, $s0, 8        # Bu c operand'ının adresidir.
addi  $s4, $s0, 12       # Bu dizi operandı d[0] ın adresidir.
```

30

Memory Instruction Format
(Hafızadan transfer komutunun formatı)

- Hafızadan reg'e transfer komutunun formatı;

destination register (Hedef registeri)

source address (kaynak adresi)

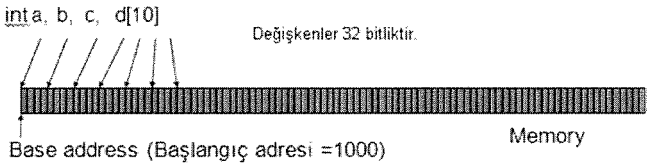
```
lw   $t0,   8($t3)
```

any register

a constant that is added to the register in brackets
(Parantez içindeki registerin içeriğine eklenen sabit değer)

31

Örnek



Aşağıdaki C kodunu Assemler koduna dönüştür. Bir önceki hafıza yapısını ve register transferlerini kullan.

C kodu: **d[3] = d[2] + a;**

Çözüm.

```
addi  $s0, $zero,1000
addi  $s1, $s0, 0        # Bu a değişkeninin adresidir.
addi  $s2, $s0, 4        # Bu b değişkeninin adresidir.
addi  $s3, $s0, 8        # Bu c değişkeninin adresidir.
addi  $s4, $s0, 12       # Bu d[0] değişkeninin adresidir.

lw    $t0, 8($s4)        # d[2] , nin içeriği $t0 registerine getirilir.
lw    $t1, 0($s1)        # a'nın içeriği $t1 registerine getirilir.
add   $t0, $t0, $t1      # Toplam $t0'nin içindedir.
sw    $t0, 12($s4)       # $t0'ln içeriği d[3] nolu hafızadaki adrese yazılır.
```

32

Özet tablo

MIPS operands

Name	Example	Comments
32 registers	$s0, $s1, ... , $t0, $t1, . . .	Registerlar DATA için hızlı yerleşimlerdir. MIPS'de, aritmetiksel işlemler için Data registerlarıdır olmalıdır
2^{30} memory words	Memory[0], Memory[4], ... Memory[4294967292]	MIPS'te hafızadaki DATA'ya sadece data transfer komutlarıyla erişilir. MIPS, Bayt adreslemi kullanır. Bu yüzden 32 bitlik bir kelimenin adresi ardış 4 bayt adresidir. Hafıza, dizi ve "spilled register" şeklinde veri (data) yapılarını tutar

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; data in registers
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; data in registers
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	Used to add constants
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Data from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Data from register to memory

Örnek:

a) Aşağıdaki işlemin sonucunu elde ediniz. Data hafızasının başlangıç adresi 0'dır. A dizisinin başlangıç adresi 8'dir.

b) Bu işlemlerin yüksek seviyeli dil kodu nedir.

```
addi   $s0, $zero,0
addi   $s1, $s0,8
lw     $s2, 0($s0)
lw     $s3, 20($s1)
add    $s4, $s2, $s3
sw     $s4, 28($s1)
```

Data Memory

19	0	1	0	1	0	1	0	1
18	1	1	0	0	1	0	1	0
17	0	0	1	1	1	1	0	1
16	1	0	0	1	1	1	0	1
15	0	0	1	1	0	0	0	1
14	0	1	0	0	1	1	0	1
13	1	1	1	1	0	1	1	0
12	0	1	0	0	1	1	0	1
11	1	0	1	1	1	0	1	0
10	0	1	0	1	0	1	0	1
9	0	1	0	1	1	0	1	1
8	1	1	1	0	0	1	0	0
7	1	0	1	0	1	1	0	1
6	0	1	0	1	1	1	0	1
5	0	0	1	0	1	0	1	0
4	0	1	0	1	0	1	0	1
3	1	0	0	0	0	0	1	1
2	0	1	1	1	1	1	1	0
1	1	0	0	0	0	0	1	1
0	0	1	0	0	0	1	1	1

39	1	1	1	0	1	0	1	1
38	1	0	0	0	1	1	0	0
37	0	0	0	1	1	0	1	0
36	1	0	0	0	0	1	1	0
35	1	0	1	1	0	1	0	1
34	1	1	0	0	1	0	1	1
33	0	1	0	0	1	1	0	0
32	0	0	0	1	1	1	0	0
31	0	1	1	0	1	0	0	0
30	0	0	1	1	0	0	1	1
29	0	0	1	1	0	0	0	1
28	0	1	1	1	0	1	0	1
27	1	1	0	0	1	0	0	0
26	1	0	0	1	1	0	1	0
25	0	1	1	0	0	0	0	1
24	0	1	0	0	1	1	0	1
23	1	1	1	1	0	1	0	1
22	0	1	0	1	1	0	1	0
21	1	0	0	0	1	1	0	1
20	0	1	1	0	1	0	1	1

2.4 Komutların Bilgisayarda temsil edilmesi (representing instructions the Computers)

- Bilgisayar sistemler binary sayı sisteminde çalışırlar. Bu sayı sisteminde 0 ve 1 olmak üzere 2 rakam vardır.
- Bu rakamların donanımsal olarak tanınması ve işlenmesi, sadece 2 durumda (düşük seviye = Lojik 0, ve yüksek seviye = lojik 1) bulunabilen elektriksel gerilim seviyeleri ile olur.
- Sözcükler 1 ve 0'ların yanyana dizilmesiyle oluşur. Sözcüğü oluşturan her rakama (0 veya 1) (binary digit) BİT denir.
- Komutlar ve sabitler ve değişkenler binary sözcükler şeklinde ifade edilir.
- MIPS assembler dilinde, *$s0* dan *$s7* ye kadar olan sembolleri 16 dan 23 'e kadarki registerleri, *$t0* dan *$t7* arasındaki semboller *8* den 15' e kadarki registerleri haritalar (MIPS-32 32 adet register'la işlem yapar). Burada $s0 16. registerdir. $t0 ise 8.registerdir.
- 32 tane registerin geri kalanları ise sonra açıklanacaktır.

Nostalji

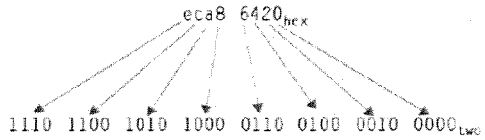
Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary
0_{hex}	0000_{two}	4_{hex}	0100_{two}	8_{hex}	1000_{two}	c_{hex}	1100_{two}
1_{hex}	0001_{two}	5_{hex}	0101_{two}	9_{hex}	1001_{two}	d_{hex}	1101_{two}
2_{hex}	0010_{two}	6_{hex}	0110_{two}	a_{hex}	1010_{two}	e_{hex}	1110_{two}
3_{hex}	0011_{two}	7_{hex}	0111_{two}	b_{hex}	1011_{two}	f_{hex}	1111_{two}

Binary-to-Hexadecimal and Back

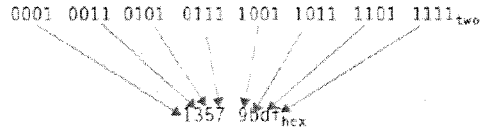
Convert the following hexadecimal and binary numbers into the other base:
$eca8\ 6420_{hex}$

$0001\ 0011\ 0101\ 0111\ 1001\ 1011\ 1101\ 1111_{two}$

Just a table lookup one way:



And then the other direction too:



MIPS assembler komutunun Makine komutuna dönüşümü

- Komutlar, veri kelimelerinin uzunluğu ve registerler 32 bit uzunluğundadır.
  - Örnek

    add $t1, $s1, $s2
  - kullanılan registerler: $t1=9, $s1=17, $s2=18
- Bir komutun formatı

0	17	18	9	0	32	10 tabanlı gösterimi
000000	10001	10010	01001	00000	100000	Makine Dili
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	
op	rs	rt	rd	shamt	funct	

- op (6 bits): Komutun temel işlevini belirtir (aritmetik, transfer, lojik, v.b), *opcode*
- rs (5 bits): 1.operand (Işlenen) için kaynak registeri belirtir.
- rt (5 bits): 2.operant için kaynak registerini belirtir.
- rd (5 bits): Sonucun yerleştirildiği hedef (varış yeri) registerini belirtir.
- shamt (5 bits): Kayma miktarı (Daha sonra açıklanacak) shift amount
- funct (6 bits): Başarılacak fonksiyon, fonksiyonun kodu (toplama, çıkrama v.b)

•Register gösterimi için niye 5 bitlik yer ayırıyoruz? Niçin
•Başarılabilecek fonksiyonların sayısı kaçtır? Neden

MIPS alanları

- Bir komut daha büyük bir alan ihtiyaç duyabilir. Örneğin ***lw komutu*** için , 2 tane registere ve bir tane sabite ihtiyaç vardır. Eğer sabiti ifade etmek için 5 bitlik adres alanı kullanılacaksa siz sabit bir değer olarak en fazla 32'yi ifade edebilirsiniz. Ancak, bir diziden veya veri yapısından eleman seçmek için kullanılan bu sabit değer çoğu zaman 32'den çok büyüktür. Bu nedenle, tüm komutları aynı uzunlukta tutmak ve tek bir komut formatı kullanmak oldukça zordur.. Bu durum bizi 4. donanım tasarım prensibine götürür.
- *Design principle 4: Good design demands good compromises.*

  (iyi tasarım , iyi tavizler (uzalaşmalar) verilerek oluşur)

  MIPS tasarımcıları, komut uzunluklarının benzer olması için, farklı gurublardaki komutlar için farklı komut formalrı oluşturmak yoluna gitmişlerdir.

Örnek;

R-Format'lı komutlar (registerler için)

I-Format'lı komutlar (sabitler veya veri transferi için)

J-Formatlı komutlar (Şartsız Dallanmalar için)

R-tipi, I-Tipi komut formatları

I- tipi komut formatında, 16 bitlik alanın anlamı, başlangıç adres registerine 2**15 = 32.768 tane bayt adresi veya sabit değerin yüklenebileceğidir.

I- tipi komut formatında; ***rt alanı*** hedef register olarak değerlendirilir. Yani yüklemenin sonucunun yazıldığı registerdir.

R-Tipi (Registerlerdaki işlem komutları için)

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

I -Tipi (Sabitler veya veri transfer komutları için)

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

Örnek `lw $t0, 32($s2)`

35	18	8	32
6 bits	5 bits	5 bits	16 bits
op	rs	rt	16 bit number

UNUTMA !!!!!!!!

Formats:

R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	16 bit address		
J	op	26 bit address				

** I-tipi format immediate(doğrudan) ve data transfer komutları içindir
** R-tipi formak Register komutları içindir.
** J-tipi format şartsız dallanma komutları içindir.

MIPS komutlarının kodlanması

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32_{ten}	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34_{ten}	n.a.
add immediate	I	8_{ten}	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	35_{ten}	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	43_{ten}	reg	reg	n.a.	n.a.	n.a.	address

FIGURE 2.6 MIPS instruction encoding.

•Dikkat !! ***subi*** komutu yoktur. Sadece ***addi*** komutu vardır. Sabit bir negatif değer olabilir.

•Formatın opcode kısmı komut kümesini belirtir (0: toplama veya çıkarma v.b aritmetik işlem komutlar, 35: lw komutu)

•Func kısmı ise komut gurubundaki özel işlem komutunu belirtir (toplama:32 v.b)

MIPS assemblerden, makine dili koduna çevrimine örnek (S.65):

$t1 A dizisinin başlangıç adresini, $s2 h operandını tutar.

```
A[300] = h + A[300];

lw   $t0,1200($t1)  # Temporary reg $t0 gets A[300]
add  $t0,$s2,$t0    # Temporary reg $t0 gets h + A[300]
sw   $t0,1200($t1)  # Stores h + A[300] back into A[300]
```

Bu üç komutun MIPS makine dili kodu nedir?

Komutların desimal karşılığı

op	rs	rt	rd	address/ shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

Komutların binary karşılığı

101011	01001	01000	0000 0100 1011 0000			8D2404B0
000000	10010	01000	01000	00000	100000	02484020
101011	01001	01000	0000 0100 1011 0000			AD2804B0

ÖĞRETİLENLERİN ÖZETİ

MIPS operands

Name	Example	Comments
32 registers	$s0, $s1, ..., $s7 $t0, $t1, ..., $t7	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. Registers $s0-$s7 map to 16-23 and $t0-$t7 map to 8-15.
2^{30} memory words	Memory[0], Memory[4], ..., Memory[4294967292]	Accessed only by data transfer instructions in MIPS. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, arrays, and spilled registers.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; data in registers
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3	Three operands; data in registers
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Data from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Data from register to memory

MIPS machine language

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
addi	I	8	18	17	100			addi $s1,$s2,100
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer format

LOJİK İşlemler, operatörler, (2.5. Logical Operations)

- İlk bilgisayarlardaki komutlar veri sözcüklerini bir bütün olarak ele alırdı. Ancak; kelime içindeki bit alanları ve hatta bireysel bitlerin üzerinde çalışmasınında yararlı olduğu ortaya çıktı. Her biri 8 bit olarak saklanır bir kelim içinde karakter incelenmesi böyle bir operasyona örnektir.
- Bu tip komutlar; paketler veya kelimeler şeklindeki verilerin içeriğini, bit açma, değerlendirme için kullanılır.
- Bu komutlar mantıksal(Lojik) işlemleri olarak bilinir.

Logical Operations	C operator	Java operator	MIPS instruction
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit NOT	~	~	nor

- Shift operations (kaydırma İşlemi):

sll $t2, $s0, 4 # $t2 = $s0 << 4

- $s0 registerinin içeriğini 4 bit sola kaydır. (Sağdan 0 doldurarak), ve sonucu $t2 registerine yaz.
- $s0'ın içeriği; 00000000 00000000 00000000 00001001 = 9
  Komut işlendikten sonraki değer.
- $t2'nin içeriği; 00000000 00000000 0000000 10010000 = 144
- 4 , "shift amount - shamt" olarak isimlendirilir ve *R-tipi komut formatında, kaydırma alanını oluşturur.*
- Hatırlayınızki bir bitlik sola kaydırma, sözcüğün 2 ile çarpılmasına eşdeğerdir.
- *Bu komut R-tipi bir komut'tur. Op ve func alanı 0'dır. Rs alanı kullanılmaz. 0 yazılır.*

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0

$t2 0000 0000 0000 0000 0000 1101 0000 0000 ise

$t1 0000 0000 0000 0000 0011 1100 0000 0000 ise

and $t0, $t1, $t2 # $t0 = $t1 & $t2

$t0 0000 0000 0000 0000 0000 1100 0000 0000 olur.

or $t0, $t1, $t2 # $t0 = $t1 | $t2

$t0 0000 0000 0000 0000 00011 110010000 0000 olur.

<u>Önemli:</u> MIPS'te NOT işlemi, nor komutu ile başarılır. Çünkü;

A Nor 0 = NOT (A OR 0) = NOT (A)

nor $t0, $t1, $t3 # $t0 = ~($t1 | $t3)

$t3 0000 0000 0000 0000 0000 0000 0000 0000 ise

$t0 1111 1111 1111 1111 11000011 1111 1111 olur.

MIPS operands

Name	Example	Comments
32 registers	$s0, $s1, . . . , $s7 $t0, $t1, . . . , $t7	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. Registers $s0-$s7 map to 16–23 and $t0-$t7 map to 8–15.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, arrays, and spilled registers.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow detected
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3	Three operands; overflow detected
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow detected
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $$s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory

FIGURE 2.10 MIPS architecture revealed thus far.

- Tablodan da görüldüğü gibi MIPS'in lojik komutları sabit değerler (immediate) ile de işlem yapabilirler. Bu lojik komutlar *andi* ve *ori* dir.

andi $s1, $s2, 100 # $s1 = $s2 & 100

ori $s1, $s2, 100 # $s1 = $s2 | 100

2.6. instructions for making decisions (Karar Komutları)

- Bir bilgisayarı basit hesap makinesinden ayıran en önemli özellik karar verebilme yeteneğidir. Giriş verileri ve değişkenler arasına istenen işlemlerin yapılması sürecinde birçok farklı komutlar çalışır.
- Pororamlama dillerinde karar verme deyimi *if* dir. Bu deyim bazen de *go* deyimi ve bir etiket ile kombine edilir.
- MIPS assembler dili iki tane karar verme komutu içerir. Bunlar *if.....go* deyimlerine benzerdir (Şartlı dallanma komutları).

1-Branch if equal (beq - Eşitse dallan):

```
beq   register1, register2, L1   # register1'in içeriği register2'nin
                                 #   içeriğine eşit ise L1'e dallan.
```

2- Branch if not equal (bne - Eşit değilse dallan):

```
bne   register1, register2, L1   # register1'in içeriği register2'nin
                                 # içeriğine eşit değilse L1'e dallan.
```

Örnek: Aşağıdaki C kodunun MIPS kodu karşılığını bulunuz.

if(i==j) f= g + h; else f = g - h;

- Buradaki f, g, h, i, j değişkenleri; MIPS' te sırasıyla f→$s0 , g → $s1, h → $s2, i → $s3, j → $s4 registerleri ile ilişkilendirilmişlerdir.

MIPS kodu aşağıdadır.

```
        bne  $s3, $s4, Else   # eğer i ≠ j ise Else etiketli yere git
        add  $s0, $s1, $s2    # h = g + j , eğer i ≠ j ise atlanır (işlenmez)
        j    exit             # Exit etiketli yere atla. (Şartsız dallan).
Else:   sub  $s0, $s1, $s2    # h=g-j , eğer i=j ise atlanır(işlenmez)
Exit:
```

Not: 1-) Hafızadaki değişkenlerin registerlere transfer edilmesi gösterilmemiştir.

2-) Atlama yerlerinin adresleri compiler ve assemler dili programlayıcısı tarafından belirlenir.

MIPS *şartsız dallanma komutu*

j etiket

- Aynı örneği başka şekilde gerçekleştirelim

```
if (i!=j);          beq  $s4, $s5, Else
h=i+j;              add  $s3, $s4, $s5
else                  j  Exit;
h=i-j;        Else: sub  $s3, $s4, $s5
              Exit:..
```

Döngüler (LOOPS)

- Kararlar hem, iki alternatif arasında seçim yapmak için (if deyimi ile karar), hemde bir hesaplamadaki iterasyon için (döngü şeklinde bulunur).
- Her iki durum için benzer assemler komutları kullanılır.

Örnek: C'de kodlanmış aşağıdaki *while* döngüsünün MIPS assemler kodu nedir?

while (save[i] == k)

i += 1;

varsayılım ki; *i* ve *k* sırasıyla $s3 ve $s5 registerlerindedir ve dizinin başlangıç değeri ise $s6 registerine kayıt edilmiştir.

İlk adım, **save[i]** operandını bir geçici (temporary) registere yüklemektir.

- **save[i]** operandını geçici registere yükleyebilmek için önce onun adresini bilmeye ihtiyaç vardır.

- *i* değişkenini , save dizisinin başlangıç adresine eklemeden önce , i değişkenini 4 ile çarpmalıyız (her bir kelime 32 bit olduğu için). Bu işlemi i sözcüğünü 2 defa sola kaydıran komut (sll) ile yapabiliriz.

-Biz programa bir LOOP isimli döngü etiketi eklemeliyiz. Böylece döngünün sonunda, dallandığımız yere geri dönebiliriz.

Loop: sll $t1, $s3,2 # $t1 = 4*i yükle

Save[i] dizi indexsinin adresini elde edelim.(dizinin başlangıç adresine $t1'in içeriğini yükleyerek)

add $t1, $t1 ,$s6 # $t1 = save[i] adresi

Elde edilen bu adres değeri ile, save[i] içeriğinin $t0 register'ine e transferi.

lw $t0, 0 ($t1) # $t0 = save[i]

Save[i] ≠k 'ya göre LOOP testi

bne $t0, $s5, Exit # eğer Save[i] ≠k ise, Exit'e git

i'ye 1 eklenme işlemi

addi $s3, $s3,1 # i = i + 1

Loop dönüsünün başına şartsız dallanma işlemi

j Loop # Loop döngüsüne git.

Exit:

save[ ]

0	k
1	k
2	k
3	j
.	
.	
.	
n	x

- Karşılaştırılanların eşit olması veya olmaması testi değişik şekillerde başarılabilir. Görme testi (to see) de bunlardan birisidir.
- MIPS'te , bu işlem karşılaştırma komutu ile yapılır. Bu komut ***"set on less than"*** *veya* ***slt*** komutudur. Bu komut iki registerin içeriğini karşılaştırır; eğer 1. registerin içeriği 2.registerin içeriğinden küçük ise 3. bir registeri 1' set eder. Tersi ise 0'a set eder.

  slt $t0, $s3, $s4 # $s3 < $s4 ise $t0 = 0
- Sabit operand'ların karşılaştırmasına çok sıkça rastlanır. ***$zero*** registerinde devamlı 0 yüklü olduğundan 0 ile karşılaştırmalar bununla yapılır. Diğer sabit değerleri karşılaştırmak için; ***slti*** komutu kullanılır. Örneğin, $s2 registerinin içeriğinin 10 sayısıyla karşılaştırmak için;

  slti $t0, $s2,10 # eğer $s2 < 10 ise, $t0 = 1

MIPS operands

Name	Example	Comments
32 registers	$s0, $s1, . . . , $s7 $t0, $t1, . . . , $t7, $zero	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. Registers $s0–$s7 map to 16–23 and $t0–$t7 map to 8–15. MIPS register $zero always equals 0.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions in MIPS. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, arrays, and spilled registers.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; data in registers
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; data in registers
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Data from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Data from register to memory
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $$s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,L	if ($s1 == $s2) go to L	Equal test and branch
	branch on not equal	bne $s1,$s2,L	if ($s1 != $s2) go to L	Not equal test and branch
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; used with beq, bne
	set on less than immediate	slti $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare less than immediate; used with beq, bne
Unconditional jump	jump	j L	go to L	Jump to target address

FIGURE 2.12 MIPS architecture revealed through Section 2.6. Highlighted portions show MIPS structures introduced in Section 2.6.

MIPS Machine Language (ex. in decimal)

Name	Format	Example						Comments
Add	R	0	2	3	1	0	32	add $s1, $s2, $s3
Sub	R	0	2	3	1	0	34	sub $s1, $s2, $s3
Addi	I	8	2	1	100			addi $s1, $s2, 100
Lw	I	35	2	1	100			lw $s1, 100($s2)
Sw	I	43	2	1	100			sw $s1, 100($s2)
Beq	I	4	1	2	100			beq $s1, $s2, 100
Bne	I	5	1	2	100			bne $s1, $s2, 100
Slt	R	0	2	3	1	0	42	slt $s1, $s2, $s3
Slti	I	10	2	1	100			slti $s1, $s2, 100
J	J	2	10000					j 10000
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
Format R	R	op	rs	rt	rd	Shamt	funct	Arithmetic Instruction format
Format I	I	op	rs	rt	Address / immediate			Transfer, branch, Imm. Format
Format J	J	op	Target address					Jump instruction format

Registerleri tekrar hatırlayalım !!!!!!! Bunlar bi-li-ne-cek

- The 32 MIPS registerleri guruplara göre:
  - Register 0 : $zero devamlı olarak 0 değeri ile yüklüdür.
  - Regs 2-3 : $v0, $v1 bir prosedürün döndürdüğü değer yüklüdür.
  - Regs 4-7 : $a0-$a3 prosedüre giriş argümanları yüklüdür.
  - Regs 8-15 : $t0-$t7 geçici değerler
  - Regs 16-23: $s0-$s7 Değişkenler için
  - Regs 24-25: $t8-$t9 daha fazla geçici değerler
  - Reg 28 : $gp global pointer
  - Reg 29 : $sp stack pointer
  - Reg 30 : $fp frame pointer
  - Reg 31 : $ra Dönüş adresi yüklüdür.

57

2.7.Bilgisayar donanımında prosedürlerin (Fonksiyonların) desteklenmesi

- Prosedür; özel bir görevi başarmak içi kullanılan altprogramlar (subroutine v.B) gibi düşünülebilir. Bunlar kendilerine tanımlanan parametrelerle ilgili işlemi başarıp sonucu verirler.
- C, Java v.b üste seviye programlarda , tool olarak ta adlandırılır .
- Prosedürler, programcıya karmaşık bir problemi çözerken, kompleks programı parçalara ayırıp, her parçanın ayrı parametreler kullanılarak sonuçlarının diğer bölümlerde kullanılması kolaylığını sağlar.
- Prosedürler sadece kendisi için tanımlanmış işi yapabilir, programda , programcı onu sık sık çağırarak başvurabilir. Yani programın farklı noktalarında yapılması gereken aynı işlemler için de prosedürlere başvurulur.

Bir prosedürün yürütülmesi için yazılan bir program aşağıdaki altı adımı yerine getirmelidir.

1- Kullanılacak parametrelerin, prosedürün ulaşabileceği bir yere yerleştirilmesi.
2- Prosedüre transferin kontrolu.
3- Prosedür için gereken hafıza kaynaklarının oluşturulması.
4- İstenen görevin yapılması.
5- Sonuçların, çağıran programın erişebileceği yerde hafızalanması.
6-Bir programda, bir prosedür değişik değişik noktaklardan çağrılabilir. Dolayısıyla bu merkez noktaların kontrol edilebilir olması gerekir. Çünkü prosedür tamamalandıktan program tekrar o noktadan devam edebilsin diye.

- Daha önce de belirtildiği gibi, registerler en hızlı veri yerleştirme ve tutma yerleridir. Biz onları mümkün olduğunca verimli kullanmak isteriz.
- MIPS-32'nin procedurlar için kullandığı registerlerin standart isimleri aşağıdadır.
- *$a0 - $a3* :Kullanılacak parametreler için 4 tane argüman registeri
- *$v0 - $v1:* döndürülen değerler için 2 tane değer registeri.
- *$ra :* Dönülecek orijinal noktayı bilmek için bir tane dönüş adresi registeri (return address tarifi).
- Bu registerlara ek olarak; MIPS assemler dili; prosedürler için bir atlama komutu da bulundurur. Bu komut istenen adrese (prosedür adresine) atlar ve aynı zamanda, takip eden komutun adresini de *$ra ' ya* kaydeder. Bu komuta *jump-and-link (jal)* komutu denir.

  jal Prosedür adresi

- Jal komutunun *prosedür adresi* kısmı, çağrılacak prosedürün adresidir. Bu adres o anda PC'deki aktüel adrestir. Bu fonksiyon aynı zamanda , tamamlandıktan sonra ana programda dönülecek yerin adresini; (dönüş adresini), *$ra* registerine depolar. Bu adres "program Counter -PC" registerindeki *adres + 4* 'dür.
- *Şu anda yürütülmekte olan komutun adresinin saklandığı özel bir register (register dosyasının parçası olmayan) vardır. Bu program sayacı (PC)'dır.*
- Jal komutu; *$ra* registerına geri dönüş komut adresi olarak *PC(Program Counter) + 4* adresini set eder.
- Prosedürün sonuna eklenen;

  Jr $ra

  (jump register- jr) komutu, $ra registerindeki belirtilen adrese şartsız dallanmayı sağlamanın garantisidir.

- Böylece çağıran program (calling program - caller - ana program v.b olabilir), ilgili parametre değerlerini $a0-$a3 registerlerine koyar ve, jal X ' ile X prosedürüne bu parametre değerleri ile dallanır. *(Prosedür, değişik komutların işlendiği bir program parçasıdır. Bazen callee diye isimlendirilir.)*
- Calle, yani çağrılan program yani, prosedür; kendisinden istenen hesaplamaları başarır ve sonuçlarını *$v0 - $v1* registerlerine koyup, çağıran programda kaldığı yere *jr $ra* komutu ile geri döner.

ÖZET

Caller (= Ana program veya diğer Prosedür)

- –stores parameters in **$a0 -$a3**
- –performs a **jal ProcedureAddress to call the procedure**

- Callee(= Prosedür)
- –performs calculations
- –saves the results in **$v0 -$v1**
- –performs a **jr $rato return to the Caller**

Bilgisayar Organizasyonu
4.Hafta:

1

2.7. Bilgisayar donanımında prosedürlerin (Fonksiyonların) desteklenmesi ***(4.hafta başlangıcı)***

- *Prosedür;* özel bir görevi başarmak için kullanılan altprogram (function - subroutine – tool) gibi düşünülebilir.

- Bunlar kendileri için tanımlanmış parametrelerle ilgili işlemi başarıp sonucu verirler.

- Prosedürler, programcıya karmaşık bir problemi çözerken, programı parçalara ayırıp, her parçanın ayrı parametreler kullanılarak sonuçlarının diğer bölümlerde kullanılması kolaylığını sağlar.

- Prosedürler sadece kendisi için tanımlanmış işi yapabilir, programda , programcı onu sık sık çağırarak başvurabilir. Yani programın farklı noktalarında yapılması gereken aynı işlemler için de prosedürlere başvurulur.

Bir prosedürün yürütülmesi için yazılan bir program aşağıdaki altı adımı yerine getirmelidir.

1- Kullanılacak parametrelerin, prosedürün ulaşabileceği bir yere yerleştirilmesi.

2- Prosedüre transferin kontrolu.

3- Prosedür için gereken hafıza kaynaklarının oluşturulması.

4- İstenen görevin yapılması.

5- Sonuç değerlerinin, *çağıran programın* erişebileceği yerde hafızalanması için bir yer.

6-Bir programda, bir prosedür değişik değişik noktalardan çağrılabilir. Dolayısıyla bu orijin noktaların kontrol edilebilir olması gerekir. Çünkü prosedür tamamlandıktan sonra program tekrar o noktadan devam edebilsin diye.

- Daha önce de belirtildiği gibi, registerler en hızlı veri yerleştirme ve tutma yerleridir. Biz onları verimli kullanmak isteriz.
- MIPS-32'nin ***procedurlar*** için kullandığı registerlerin standart isimleri aşağıdadır.

▪ ***$a0 - $a3*** :Kullanılacak parametreler için 4 tane argüman registeri; ***Ana programdan (Çağıran -Caller) veya başka bir prosedürden, çağrılan prosedüre (callee) gönderilen giriş argümanları*** içindir.

▪ ***$v0 - $v1: Prosedürden döndürülen değerler için 2 tane*** register.

▪ ***$ra :*** Dönülecek orijinal noktayı bilmek için bir tane dönüş adresi registeri (return address tarifi).

• Bu registerlar'a ek olarak; MIPS assemler dili; prosedürler için bir atlama komutu da bulundurur. Bu komut istenen adrese (prosedür adresine) atlar ve aynı zamanda, takip eden komutun adresini de *$ra 'ya* kaydeder. Bu komuta *jump-and-link (jal)* komutu denir.

jal Prosedür adresi

- ***Yürütülmekte olan komutun adresinin saklandığı özel bir register (register dosyasının parçası olmayan) vardır. Buna program sayacı (PC – Program counter) diyoruz.***
- **Jal** komutunun ***prosedür adresi*** kısmı, çağrılacak prosedürün adresidir (prosedürün ilk satırıdır). Bu adres o anda PC'deki aktüel adrestir.
- Bu fonksiyon aynı zamanda , tamamlandıktan sonra ana programda dönülecek yerin adresini; (dönüş adresini), ***$ra*** registerine depolar.
- **Jal komutu**; ***$ra*** registerına geri dönüş komut adresi olarak

  PC (Program Counter) + 4 adresini set eder.
- Prosedürün sonuna eklenen;

  Jr $ra

  (**jump register - jr**) komutu, $ra registerindeki belirtilen adrese şartsız dallanmayı sağlamanın garantisidir.

Dikkat!! Jr $ra → PC= R[ra] (ra reg'inin içeriğini PC'ye koyar.)

- Böylece çağıran program *(calling program - caller - ana program v.b olabilir),* ilgili parametre değerlerini $a0-$a3 registerlerine koyar ve, jal X ' ile X prosedürüne bu parametre değerleri ile dallanır. *(Prosedür, değişik komutların işlendiği bir program parçasıdır. Bazen callee (çağrılan) diye isimlendirilir.)*

- Calle, yani çağrılan program yani, prosedür; kendisinden istenen hesaplamaları başarır ve sonuçlarını *$v0 - $v1* registerlerine koyup, çağıran programda kaldığı yere *jr $ra* komutu ile geri döner.

Prosedürün ihtiyaç duyduğu register sayısı fazla ise ne yapılmalı?

- Prosedürün attığı ilk adım, ana program tarafından kullanılan registerlerin içeriklerini kaydetmek olur. Çünkü bu registerları kendisi de kullanacağından bir önceki içerikler kaybolmamalıdır. Ana programa geri dönmeden önce de registerlerin ilk içeriklerini ilgili registerlere tekrar kaydetmelidir.

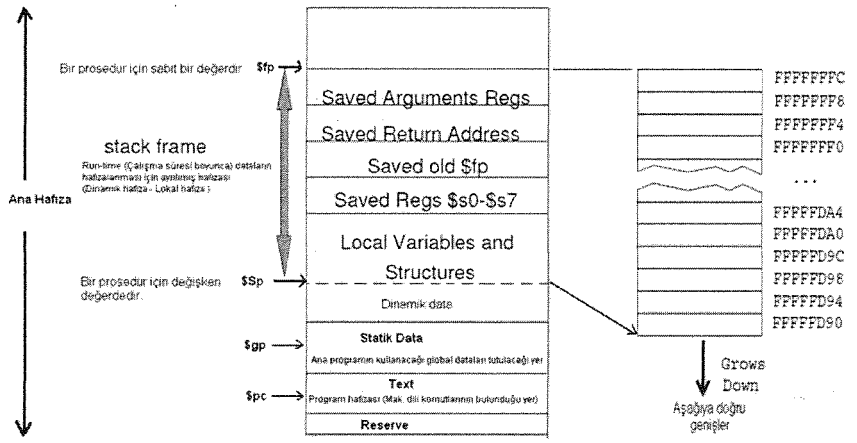
- Registerların kaydedildiği hafıza kısmı spilling memory olarak isimlendirilir.

- *stack = LIFO yapısında, registerlerin saklanma biçimini tarif eden veri yapısıdır.*

- *Stack Pointer ($sp) : Stack'teki en son işlenenin adresini işaret eder.*

- Stack'e yazma için push (itme) okuma için pop (çekme) işlemleri kullanılır.

Bir program için hafıza yapısının basit resmi



8

Aşağıdaki C prosedürünü inceleyelim

```
int leaf_example (int g, int h, int i, int j)
{
        int f;
        f = (g + h) –(i + j);
        return f;
}
```

İlgili operandlar aşağıdaki registerlardadır. Bu reg'lerin bir özelliği varmıdır?

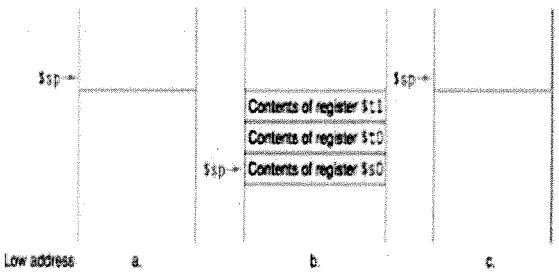
g →$a0, h → $a1, I → $a2, j → $a3, f → $s0

Bu prosedür derlendiğinde MIPS kodu nedir?

<u>Bu prosedürün yapacağı 1.İşlem :</u> Ana programda kullanılabilen reg'leri stack'e depolamaktır. Bu prosedürün, $to, $t1, $s0 reg'lerini kullandığını farz edelim. Stack'e atma işlemi aşağıdaki gibidir.

Register spilling in stack (push)

```
Leaf_example:     # prosedür bu etiketle başlar
addi  $sp, $sp, -12   # Stakc'te  3 birimlik yer açmak
sw    $t1, 8($sp)     # t1'in içeriğini stack'in içine kaydetmek (push)
sw    $t0, 4($sp)     # to'ın içeriğini stack'in içine kaydetmek
sw    $s0, 0($sp)     # s0'ın içeriğini stack'ın içine ittirmek.
```



Şekil: stack ve stack pointer'in değerleri;

a) Prosedür çağrılmadan önce
b) Prosedürün çalışma süresince
c) Prosedür sonlandıktan sonra

Prosedürün MIPS kodu

Register spilling in stack (push)

```
addi $sp, $sp, -12
sw   $t1, 8($sp)
sw   $t0, 4($sp)
sw   $s0, 0($sp)
```

Body or procedure

```
add $t0, $a0, $a1      # $t0 = g + h
add $t1, $a2, $a3      # $t1 = i + j
sub $s0, $t0, $t1      # f = (g + h) - (i + j)
```

Save return value

```
add $v0, $s0, $zero    # to return $v0 = f = $s0 + 0
```

Restore Registers (pop)

```
lw   $s0, 0($sp)       # restore $s0 from stack
lw   $t0, 4($sp)       # restore $t0 from stack
lw   $t1, 8($sp)       # restore $t1 from stack
addi $sp, $sp,12       # stack pointer'i eski yerine
```

Return

```
jr $ra                 # return = jump back to caller
```

ÖZET

Caller (= Ana program veya diğer Prosedür)

- – Parametreli, **$a0 -$a3 registerlarına kayıt eder.**
- – **jal ProcedureAddress** prosedürün çağrılmasını sağlar.

• *Callee(= Prosedür)*

- – kendinden beklenen hesaplamaları başarır.
- – Elde mettiğ sonuçları **$v0 -$v1** geri dönüş reg'lerine kayıt eder.
- – **jr $ra** komutu ile çağıran programa geri döner.
- ***Eğer bir prosedür, s ile başlayan reg'leri kullanacaksa, bunların içerikleri muhakkak korunmalı (stack'e atılabilir).***
- ***Eğer bir prosedür, t ile başlayan reg'leri kullanacaksa, bunların içerikleri korunmayabilir.***

özet

• Jal komutu prosedüre dallanmak için kullanılır ve PC içeriğine 4 ekleyerek geri dönüş adres registerine (ra) yazar.

• Argümanlar $a0-$a3 registerlerine, Geri dönüş değerleri, $v0-$v1 registerlerine atanır.

• Callee,callerin registerlerinin üzerine yazabildiği için, İlişkili değerler hafızaya(Stack) kopyalanmalıdır.

•Her bir procedur, yerel değişkenler için bir hafıza yerine İhtiyaç duyar (Stackt'e yer ayrılması).

•Stack; procedürler için ihtiyaç duyulan organize edilmiş hafıza bölgesidir.

13

İç İçe prosedürler (Nested Procedures)

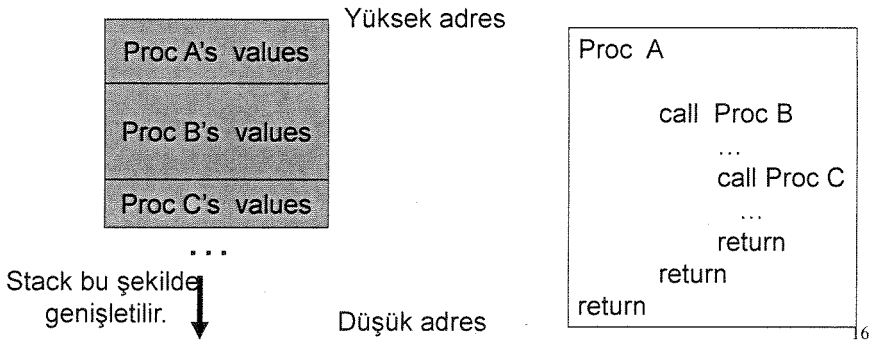
- Başka bir prosedür tarafından çağrılmayan prosedürlere yaprak prosedür (leaf prosedür) denir.
- Kullanılan prosedürler yaprak prosedür olsaydı işler çok kolay olurdu.
- Oysa; bir prosedür tanımlanmış bir işi kolay bir şekilde yapmak için, içerisinde başka bir prosedürün (veya prosedürlerin) çağrılmasına gerek duyabilir.
- Hatta; kendi kendilerini çağıran recursive (Yinelemeli) prosedürler bile mevcuttur.
- Bu şekilde çalışmalarda , kullanılan registerlardaki bilgilerin kaydedilmesi ve geri çağrılması durumları çok önemlidir.

Çalışılan prosedür yaprak prosedür değilse çok dikkat edilmelidir.!!!!!!! *Örneğin;*

- *ProsedürA, 5 parametresi ile ($a0 = 5, $ra 'da set edilerek) ana program tarafından çağrılmış olsun.*
- *–ProsedürB ise 10 parametresi ile ($a0 = 10, $ra dönüş adresi tekrar set edilerek) ProsedürA tarafından çağrılmış olsun.*
- – A prosedürü doğru data ile devam edebilecek midir?
- – A prosedür'ü ana programa doğru bir şekilde dönebilecek midir?
- En son çağrılan prosedür, registerlerin üzerine veri yazabileceği için; ilgili reg'lerdeki değerlerin belleğe (stack) kaydedilmesi gerekir !!!!!TAMAM'MI?
- Önemli not: İç-içe çağrılan prosedürlerde veya rekürsif programlarda her çağırıdan önce bir önceki dallanma yeri ve parametreler stac'te kaydedilmelidir.

Stack (Yığıt)

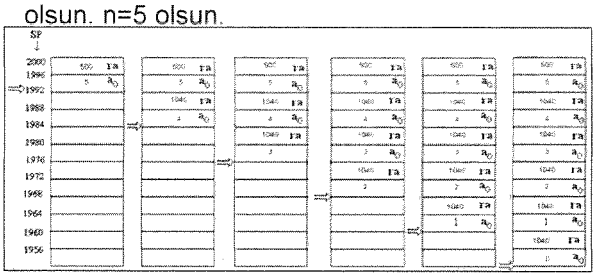
Procedür'ün kullandığı her bir register değerleri stack'ta yedeklenmelidir. Aşağıda iç içe prosedürler için stack'taki değerlerin yerleşimi gösterilmiştir.



Örnek: Aşağıdaki rekürsif fonksiyonunun MIPS komu seti kodunu Yazınız.

```
int  fact (int n)
{
    if (n < 1) return (1);
        else return (n * fact(n-1));
}
```

Ana programda; çağrılan fact(n) fonksiyonundan sonraki komutun adresi 500 olsun. n=5 olsun.



```
    1000  fact:  addi  $sp,$sp,-8
1   1004         sw    $ra,4($sp)
    1008         sw    $a0,0($sp)
    1012         slti  $t0,$a0,1
2   1016         beq   $t0,$zero,L1
    1020         addi  $v0,$zero,1
3   1024         addi  $sp,$sp,8
    1028         jr    $ra
    1032  L1:    addi  $a0,$a0,-1
4   1036         jal   fact
    1040         lw    $a0,0($sp)
5   1044         lw    $ra,4($sp)
    1048         addi  $sp,$sp,8
    1052         mul   $v0,$a0,$v0
6   1056         jr    $ra
```

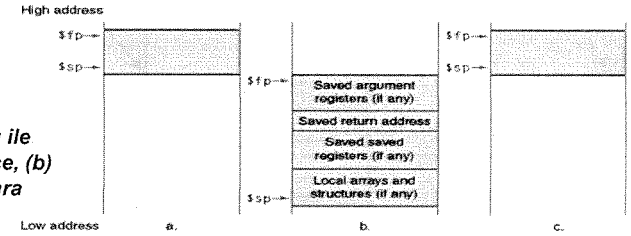
STACK'te yeni veriler için yer ayrılması

•Bir prosedür tarafından stack'ta tahsis edilen yer, *"prosedür frame"* veya "aktivasyon kayıtı" olarak adlandırılır. (Bu ayrılan yer,Prosedür için yerel verileri ve kaydedilen değerleri içerir).

•Frame Pointer; Verilen bir prosedür için yerel değişkenleri ve kaydedilen registerlerin yerini gösteren bir değerdir

• $fp registeri , stac'teki kayıtın başını, $sp sonunu gösterir.

•$fp prosedürün yürütülmesi boyunca değişmez.. $sp Prosedürün yürütmesi boyunca değişebilir.



Prosedürün çağrılması ile Stack yerleşim, (a) önce, (b) süreç boyunca, (c) sonra

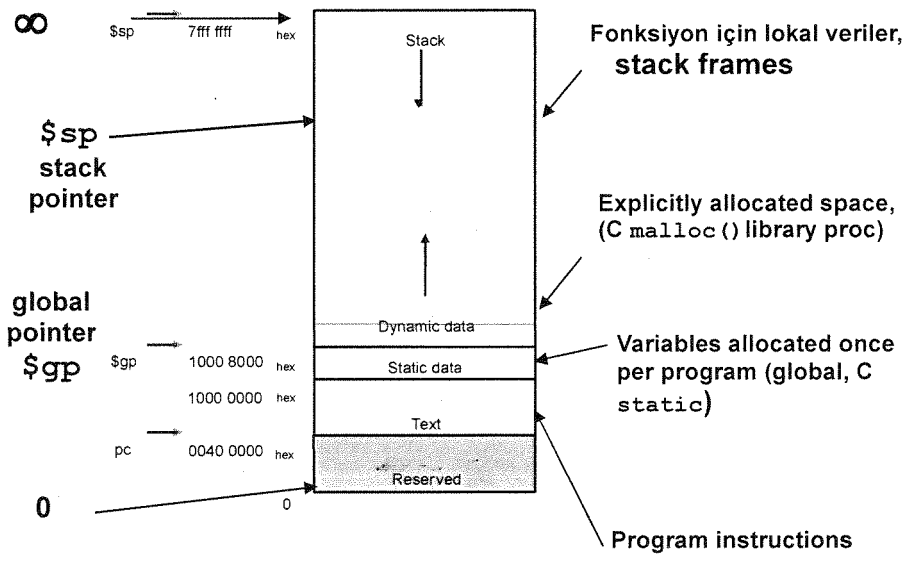
•$fp Frame'in 1.kelimesini gösterir.
•$sp ; stac'in en üstünü gösterir.
•$sp programın çalışması boyunca değişebilir.
•$fp sabit kalır, böylece $fp kullanılarak referans veri adresi tespit edilir.
•$fp kullanıldığında, $sp başlangıç noktasına bu sayede getirilir.

18

- Global pointer: Hafıza alanına kaydedilen global değişkenlerin (ana program ve prosedürler tarafından kullanılabilen) kaydedildiği alanı gösterir. Bu değer $gp registerinde tutulur.
- Dinamik olarak tahsis edilen hafıza (C'de malloc() fonksiyonu ile ayrılır) **heap** alanıdır.

19

Yürütülebilir programlarda gerçek zamanda hafıza-Organizasyonu



20

Name	Register number	Usage	Preserved on call?
$zero	0	the constant value 0	n.a.
$v0–$v1	2–3	values for results and expression evaluation	no
$a0–$a3	4–7	arguments	no
$t0–$t7	8–15	temporaries	no
$s0–$s7	16–23	saved	yes
$t8–$t9	24–25	more temporaries	no
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return address	yes

21

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3	three register operands
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Data from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Data from register to memory
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 |$s3)	three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $$s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,L	if ($s1 == $s2) go to L	Equal test and branch
	branch on not equal	bne $s1,$s2,L	if ($s1 != $s2) go to L	Not equal test and branch
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; used with beq, bne
	set on less than immediate	slt $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare less than immediate; used with beq, bne
Unconditional jump	jump	j L	go to L	Jump to target address
	jump register	jr $ra	go to $ra	For procedure return
	jump and link	jal L	$ra = PC + 4; go to L	For procedure call

22

MIPS machine language

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
and	R	0	18	19	17	0	36	and $s1,$s2,$s3
or	R	0	18	19	17	0	37	or $s1,$s2,$s3
nor	R	0	18	19	17	0	39	nor $s1,$s2,$s3
andi	I	12	18	17	100			andi $s1,$s2,100
ori	I	13	18	17	100			ori $s1,$s2,100
sll	R	0	0	18	17	10	0	sll $s1,$s2,10
srl	R	0	0	18	17	10	2	srl $s1,$s2,10
beq	I	4	17	18	25			beq $s1,$s2,100
bne	I	5	17	18	25			bne $s1,$s2,100
slt	R	0	18	19	17	0	42	slt $s1,$s2,$s3
j	J	2	2500					j 10000 (see Section 2.9)
jr	R	0	31	0	0	0	8	jr $ra
jal	J	3	2500					jal 10000 (see Section 2.9)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer, branch format

23

Karakterlerle çalışma için temel komutlar

•Çoğu bilgisayarlar, text prosesleri için karakter gösteriminde 8-bitlik kelime kullanır (ASCII kodlamada olduğu gibi) .

•Bazı komutlar, 32 bitlik bir kelimeden 8 bitlik bir kelime elde edebilir. Bunlardan **lb** ve **sb** transfer komutları 32 bitlik kelimeler yerine 8 bitlik kelimeleri transfer edebilir.

•MIPS bunu desteklemelidir. Çünkü bazı programlar 8 bit çalışmak içindir.
• lb (load byte) Komutu, ana hafızadaki 8 bitlik veriyi registerin en sağdaki 8 bitine yükler.
•Sb(Store byte) komutu ise bunun tersini yapar.
  • lb $t0, 0($sp)
  • sb $t0, 0($sp)

24

Örnek

String (dizi-karakter topluluğu) karakterlerden oluşur. Örneğin C'de tanımlı *"Cal"* stringi 4 byte ile gösterilir (67 (C), 97 (a),108 (l),0(null) olur.

Aşağıdaki kodu assemlere dönüştürün.

```
void strcpy (char x[], char y[])
{
   int i;
   i=0;
   while  ((x[i] = y[i]) != '\0')
   i + = 1;
}
```

<u>Varsayım:</u> x ve y dizilerinin başlangıç adresleri $a0 ve $a1'de olsun. i için $so reg'i kullanılsın. $so reg'i muhakkak stakc'te yedeklenmelidir.

```
strcpy:
 addi    $sp, $sp, -4
 sw      $s0, 0($sp)
 add     $s0, $zero, $zero
L1: add  $t1, $s0, $a1
 lb      $t2, 0($t1)
 add     $t3, $s0, $a0
 sb      $t2, 0($t3)
 beq     $t2, $zero, L2
 addi    $s0, $s0, 1
 j       L1
L2: lw   $s0, 0($sp)
 addi    $sp, $sp, 4
 jr      $ra
```

25

- MIPS komut seti half-word (16 bitlik) yükleme ve depolama (transfer) işlemlerini de destekler.
- lh (load half) : hafızadan bir half word'ü registerin en ağırlıksız 16 bitine yükleme işini yapar.
- Sh(Store half) : Bu komut ise tersi işlemi yapar.

```
lh $t0,0($sp)  # Read halfword (16 bits) from source
sh $t0,0($gp)  # Write halfword (16 bits) to destination
```

26

32 bit immediate operandlar

- Immediate komutlar ile sadece 16-bit sabitler belirtilebilir.
- lui (load upper immediate) komutu: 16 bitlik sabiti, registerin upper register (en ağırlıkılı 16 bit) alanına kaydetmek için kullanılır.

The machine language version of `lui $t0, 255` `# $t0 is register 8:`

001111	00000	01000	0000 0000 1111 1111

Contents of register `$t0` after executing `lui $t0, 255:`

0000 0000 1111 1111	0000 0000 0000 0000

27

Örnek: 32 bitlik sabit sayı oluşturma

32 bitlik sabit tanımlamak için iki immediate komut kullanılır.

What is the MIPS assembly code to load this 32-bit constant into register `$s0`?

```
0000 0000 0011 1101 0000 1001 0000 0000
```

First, we would load the upper 16 bits, which is 61 in decimal, using `lui`:

```
lui $s0, 61   # 61 decimal = 0000 0000 0011 1101 binary
```

The value of register `$s0` afterward is

```
0000 0000 0011 1101 0000 0000 0000 0000
```

The next step is to add the lower 16 bits, whose decimal value is 2304:

```
ori $s0, $s0, 2304 # 2304 decimal = 0000 1001 0000 0000
```

The final value in register `$s0` is the desired value:

```
0000 0000 0011 1101 0000 1001 0000 0000
```

MIPS'te , assemler $at registerinde saklanmış değeri uzun sabit değerlerini veya adres değerlerini oluşturmaktan sorumludur.

28

Dallanma ve atlamada adreslemenin önemi

`bne $t4,$t5,Label` $t4 ≠ $t5 ise yeni komut LABEL' etiketinin gösterdiği adrestir.
`beq $t4,$t5,Label` $t4 = $t5 ise yeni komut LABEL' etiketinin gösterdiği adrestir.
`j Label` yeni komut LABEL' etiketinin gösterdiği adrestir.

I	op	rs	rt	16 bit address
J	op	26 bit address		

- Şartlı dallanmada, dallanılacak adres 16 bitlik sabit bir değerdir. Dallanılacak adres 16 bitden daha büyük ise ne olacaktır?
- Aynı şekilde şartsız dallanmada (atlama) da dallanılacak adres 26 bit'tir. Daha büyük bir değerde ne olacaktır?

1- Dallanma adresi, 32bitlik bir reg'in içeriği ile toplanıp PC'ye yazılır . Buna relative adresleme denir. Ama hangi registeri kullanalım?

PC = register + dallanma komutundaki adres

Şartlı dallanmalarda genellikle yakın komutlara dallanıldığından (İstatistiksel elde edilmiştir) , mevcut komutun adresinin tutulduğu PC reg'in kullanıma çok uygundur. Buna PC-Relativ adresleme denir.

29

2- PC'nin içeriği ile dallanma komutundaki adres toplanıp PC'ye yazılır. Buna PC-Relative adresleme denir. Genellikle bu kullanılır .

PC = PC + dallanma komutundaki adres

Böylece biz mevcut komuttan $\pm 2^{15}$ word adresi kadar uzağa dallanabiliriz.

- Şartlı dallanmada, belirtilen 16 bitlik sabit değer, hafızadaki word (32 bit – 4 byte) adresleridir. Oysa ana hafızada 1 baytlık adresleme söz konusuydu. Bu durumda dallanılacak doğru adresi bulmak için 16 bitlik sabit değeri 4 ile çarpıp PC + 4'e eklemek gerekir.

PC = (PC+4) + (dallanma komutundaki adres*4)

30

Örnek:

```
Loop:   sll $t1, $s3, 2      # $t1 = 4*I
        add $t1, $t1, $s6    # $t1 = address of save[i]
        lw  $t0, 0($t1)      # $t0 = save[i]
        bne $t0, $s5, Exit   # go to Exit if save[i] ≠ k
        addi $s3, $s3, 1     # i++
        j Loop               # loop again
Exit:                        # out of the loop
```

Assembled instructions and their addresses

80000	0	0	19	9	4	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024	...					

31

bne $t0, $s5, Exit komutu için PC-relative adresleme;

- PC = (PC+4) + (dallanma adresi*4)
- **→ PC = (80012+4) + (2*4) = 80024 (Exit etiketinin adresi)**
- Not: 80012, bne komutunun adresidir. Sonraki komutun adresi 80012 + 4 dür. bne komutundaki dallanılacak exit etiketinin adresi 2 olarak komutta verilmiştir. Bu değer word olarak adres değeridir.
- Oysa biz 32bit çalıştığımızdan bu değer 4 ile çarpılıp word'e dönüştürülerek dallanılacak gerçek adres elde edilir. (80024 adresi)

Assembled instructions and their addresses

80000	0	0	19	9	4	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024	...					

32

j Loop (Atlama komutu) için Pseudodirect adresleme

Şartsız dallanma komutundaki 26 bitlik alan word (32 bitlik) olarak atlama adresini belirtir. Bunu byte olarak ifade etmek için 4 ile çarpmak gerekir.

Dolayısıyla bu alan PC'de 28 bit olarak ortaya çıkar. Niye?

PC = (PC'nin en ağ.4 biti) +(jump addresi*4)

$\rightarrow$PC = $(0000)_2$+ $(20000*4)_{10}$= 80000_{10}(Loop'un adresi)

33

Uzaklara Dallanma (Branching Away)

```
        beq  $s0, $s1, L1
```

L1 etiketinin çok uzak bir adres olduğunu varsayın. Budurumda iki komut kullanılarak çok uzak adreslere dallanma yapılabilir.

```
        bne  $s0, $s1, L2
        j    L1    # uzaklara dallanabilir. Niye?
L2:
```

34

MIPS'de Adresleme modları

1- Register adresleme : Burada operand register'dır.

2- Base veya yerdeğiştirme adresleme: Burada operand ana hafızadaki lokasyondur.Ki onun adresi; komutaki sabit ile registerin toplamından elde edilir.

3- İmmediate adresleme: Buradaki operand komutun kendisindeki sabittir.

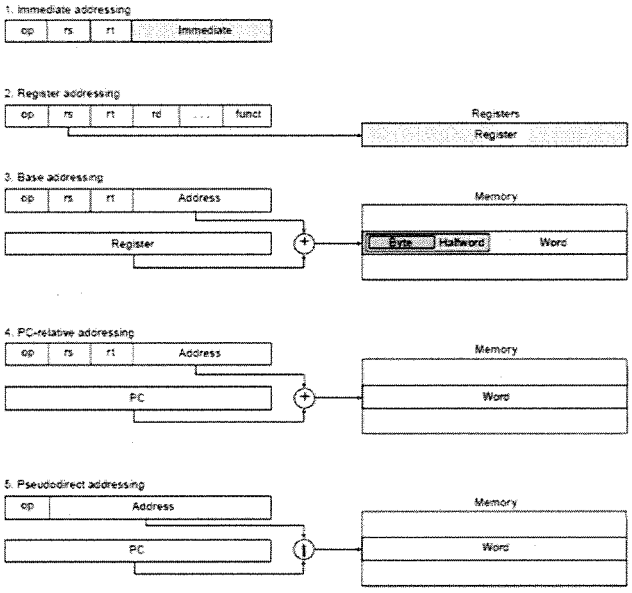
4-PC-Relative adresleme: Burada adres, komuttaki sabit ile PC'deki değerin toplanmasıyla elde edilir.

5- Pseudodirect adresleme: Buradaki atlama adresi PC ağırlıklı bitleri ile birleştirilmiş komutunun 26 bit'idir.

Bunları şematik olarak gösterebiliriz.

35

Adresleme modları şeması



36

Makine dilinin dekodlanması

•Makine dilinden ,tersine giderek orijinal assemler komutlarnı yaratmaktır.

- Örnek:

  00af8020 hex

Makine dilindeki komutun assemler dili karşılığı nedir?

Çözüm:

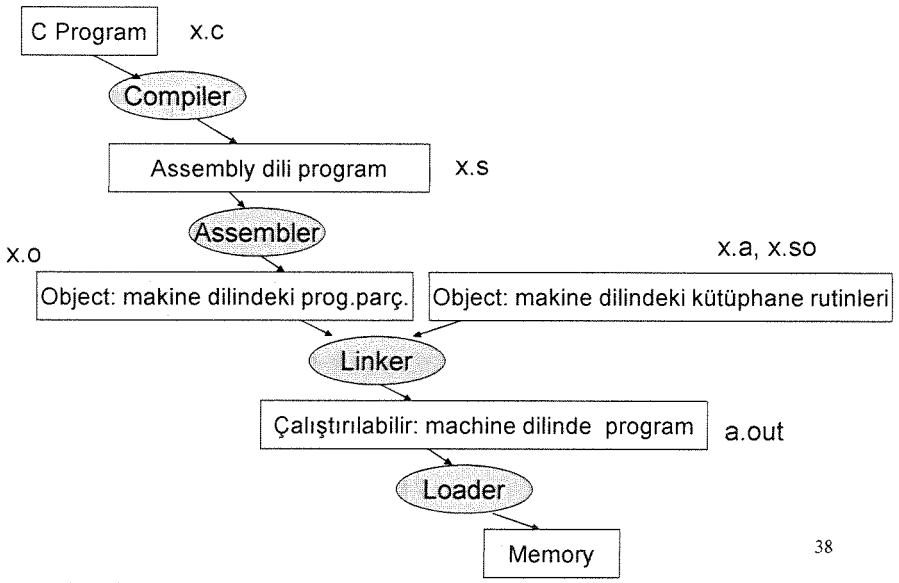
```
(Bits:  31 28 26                          5  2  0)
        0000 0000 1010 1111 1000 0000 0010 0000

op        rs      rt      rd      shamt   funct
000000    00101   01111   10000   00000   100000

add $s0,$a1,$t7
```

37

Bir Programın Başlatılması



38

Assembler'in Rolü

• Pseudo(sözde) kodları gerçek donanım komutlarına dönüştürmek.
  – Pseudo-kodlar assembly programa kolayca dönüştürür örnekler: "move", "blt", 32-bit immediate operandlar gibi.

• Assembly kodlar machine kodlarına dönüştürülür.
Ayrı object file (x.o) her bir C dosyası için oluşturulur (x.c) komut etiketleri için gerçek değerleri hesaplar.
Dış referanslar ve hata ayıklama bilgileri hakkında bilgi sağlamak için

39

Linker'in Rolü

- patch iç ve dış kaynaklar
- veri ve komut etiketlerinin adreslerini hesapla
- hafızada veri ve kod modüllerini organize et.

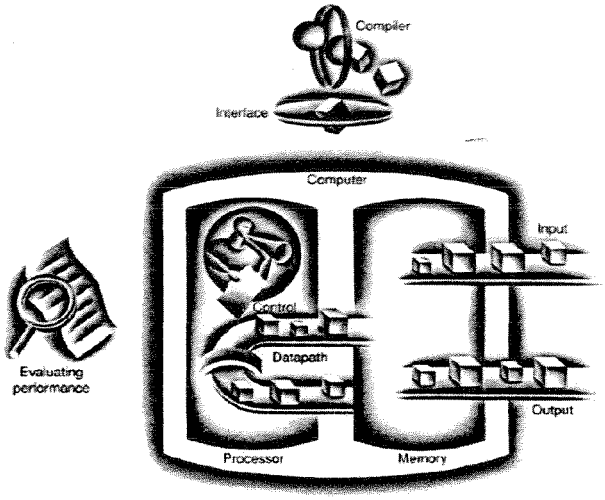
• Bazı kütüphaneler(DLLs) dinamik olarak bağlanır(linked)
Dummy rutinleri yürütülebilir nokatalara– Bu dummy rutinleri dinamik linker ve yükleme cağırır. Bu yüzden onlar doğru rutinlere atlama için yürütülebilir güncellenebilir.

40

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three register operands
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	Used to add constants
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Word from memory to regis
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Word from register to mem
	load half	lh $s1,100($s2)	$s1 = Memory[$s2 + 100]	Halfword memory to register
	store half	sh $s1,100($s2)	Memory[$s2 + 100] = $s1	Halfword register to memory
	load byte	lb $s1,100($s2)	$s1 = Memory[$s2 + 100]	Byte from memory to registe
	store byte	sb $s1,100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memor
	load upper immed.	lui $s1,100	$s1 = 100 * 2^{16}	Loads constant in upper 16
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-b
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-b
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 |$s3)	Three reg. operands; bit-by-b
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND reg with const
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR reg with consta
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branc
	branch on not equal	bne $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for be
	set less than immediate	slti $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare less than constant
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure retu
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

3. Bölüm
BİLGİSAYAR ARİTMETİĞİ



1

İşaretli ve işaretsiz sayılar

Sayılar herhangi bir taban kullanılarak temsil edilebilir. Günlük hayatımızda kullandığımız sayı sistemi 10'luk (Desimal) sayı sistemiyken bilgisayar 2'lik (Binary)sayı sistemini kullanmaktadır. Herhangi bir tabanda bir sayının oluşumu aşağıdaki gibidir. A_i basamak rakamı, T taban, i basamak değerini ifade eder. Ai basamak rakamı, 0, 1..T-1'e kadarki ardışıl rakamlardan herhangibiri olabilir.

$$(\text{Sayı})_T = \sum_{i=0}^{i=n-1} A_i \cdot T^i + \sum_{j=-m}^{j=-1} A_j \cdot T^j$$

(Tamsayı kısmı) + (kesirli kısmı)

Soru: Bilgisayarlarda niye 2 tabanlı sayı sistemi kullanılır?
Cevap: Bu tabanda kullanılan rakamlar 0 ve 1'dir. Bu rakamların elektriksel işaretler olarak tanınması çok kolaydır.(Transistörün anahtarlama modunda çalıştırılması ile)

2

1011_{two} sayısının desimal sayı sistemindeki karşılığını hesaplayalım;

$$(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)_{ten}$$
$$= (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1)_{ten}$$
$$= 8 + 0 + 2 + 1_{ten}$$
$$= 11_{ten}$$

Bu sayıyı ikilik tabanda 32 bit ile ifade edecek olursak yeni sayı;

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 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 0 1 1

(32 bits wide)

3

32 bitlik işaretsiz sayılarda yazılabilecek en küçük sayı;

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$$

En büyük sayı;

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 4{,}294{,}967{,}295_{ten}$$

Bu sayıların desimal karşılıklarının hesaplanması;

$$(x31 \times 2^{31}) + (x30 \times 2^{30}) + (x29 \times 2^{29}) + \ldots + (x1 \times 2^1) + (x0 \times 2^0)$$

4

İşaretli sayılarda ikilik tabanda diğer sayı tabanlarında olduğu gibi ± işaretleri, sayının pozitif ya da negatif olduğunu belirlemez ikilik tabanda işaretli bir sayının işaretini en anlamlı biti (MSB) vermektedir. En anlamlı bit işareti, diğer bitler ise sayıyı ifade etmektedir.

En anlamlı bit '0' ise sayı pozitiftir ve desimal karşılığı işaretsiz sayılarda olduğu gibi hesaplanır. En anlamlı bit '1' ise sayı negatiftir ve desimal karşılığını hesaplayabilmek için 2'ye tümleyeninin alınması gerekmektedir.

5

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = +\ 1_{10}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = +\ 2_{10}$
...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = +\ 2{,}147{,}483{,}646_{10}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = +\ 2{,}147{,}483{,}647_{10}$ *maxint*
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -\ 2{,}147{,}483{,}648_{10}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = -\ 2{,}147{,}483{,}647_{10}$ *minint*
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = -\ 2{,}147{,}483{,}646_{10}$
...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = -\ 3_{10}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = -\ 2_{10}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = -\ 1_{10}$

Yukarıda 32 bitlik bazı işaretli sayılar ve desimal karşılıkları verilmektedir.

32 bitlik işaretli sayılar ile ifade edilebilecek;

En büyük sayı (2^{31}-1) 2,147,483,647

En küçük sayı (-2^{31}) -2,147,483,648 dir.

6

- İkilik tabanda verilen işaretli bir sayının desimal karşılığı aşağıdaki şekilde hesaplanmaktadır.

$$(x31 \times -2^{31}) + (x30 \times 2^{30}) + (x29 \times 2^{29}) + \ldots + (x1 \times 2^{1}) + (x0 \times 2^{0})$$

Buradaki -2^{31} işaret bitini temsil etmektedir.

Örnek: 5 bitlik işaretli sayı için

$(+13)_{10} = 0\ 1101 = 0x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 8 + 4 + 1 = +13$

$(-13)_{10} = 1\ 0011 = -(1x2^4) + (0x2^3 + 0x2^2 + 1x2^1 + 1x2^0) = -16 + 3 = -13$

7

İkilik tabandan onluk tabana dönüşüm işlemi

Verilen 32 bitlik işaretli sayının desimal karşılığını bulalım.

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$$

Sayının tüm bitlerini formülde yerine koyarsak;

$$(1 \times -2^{31}) + (1 \times 2^{30}) + (1 \times 2^{29}) + \ldots + (1 \times 2^{2}) + (0 \times 2^{1}) + (0 \times 2^{0})$$
$$= -2^{31} + 2^{30} + 2^{29} + \ldots + 2^{2} + 0 + 0$$
$$= -2{,}147{,}483{,}648_{ten} + 2{,}147{,}483{,}644_{ten}$$
$$= -4_{ten}$$

8

MIPS

- `lb, lbu`
  - (lb - load byte) 32 bitlik işaretli sayının en anlamlı 24 bitini işarete göre doldurur.
  - (lbu- load byte unsigned) işaretsiz sayılarda anlamlı bitlere 0 atar.
- `lh, lhu`
  - (lh - Load half) işaretli sayılarda en anlamlı 16 biti işarete göre doldurur.
  - (lhu - Load half word unsigned) işaretsiz sayılarda
- `slt & slti`
  - (slt - set on less than) işaretli sayılarda karşılaştırma
  - (slti - set on less than immediate) işaretli tamsayılarda karşılaştırma
- `sltu & sltiu`
  - (sltu - set on less than unsigned) işaretsiz sayılarda karşılaştırma
  - (sltiu -set on less than immediate unsigned) işaretsiz tamsayılarda karşılaştırma

9

İşaretli ve işaretsiz sayıların karşılaştırılması

$s0 register'ında aşağıdaki sayı olsun.

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two}$

$s1 register'ında aşağıdaki sayı olsun.

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two}$

Aşağıdaki tanımlamaya göre $t0 ve $t1 register'larının değerleri ne olur?

```
slt     $t0, $s0, $s1 # signed comparison
sltu    $t1, $s0, $s1 # unsigned comparison
```

10

Cevap;

$s1 değeri 1'dir. $s0 işaretli bit tamsayı ise değeri -1, işaretsiz bir tamsayı ise değeri 4294967295 tir.

slt $t0, $s0, $s1 $t0=1 (-1<1)

sltu $t1, $s0, $s1 $t1=0 (4294967295>1)

11

İşaret değiştirme

- 2 sayısını 32 bitlik -2'ye çeviriniz.

$2_{ten} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two}$

12

Bir sayının negatifi alınacağı zaman öncelikle tüm bitlerinin tersi alınır, ardından 1 ile toplanarak karşılığı bulunur.

$$\begin{array}{ll} & 1111\,1111\,1111\,1111\,1111\,1111\,1111\,1101_{two} \\ + & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 1_{two} \\ \hline = & 1111\,1111\,1111\,1111\,1111\,1111\,1111\,1110_{two} \\ = & -2_{ten} \end{array}$$

Aynı işlem negatif bir sayının pozitif karşılığını bulmak için de kullanılır.

$$\begin{array}{ll} & 0000\,0000\,0000\,0000\,0000\,0000\,0000\,0001_{two} \\ + & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 1_{two} \\ \hline = & 0000\,0000\,0000\,0000\,0000\,0000\,0000\,0010_{two} \\ = & 2_{ten} \end{array}$$

13

İkilik tabanda işaretli 16 bitlik 2'yi 32 bit ile ifade edelim.

$0000\,0000\,0000\,0010_{two} = 2_{ten}$

$0000\,0000\,0000\,0000\,0000\,0000\,0000\,0010_{two} = 2_{ten}$

İkilik tabanda işaretli 16 bitlik -2'yi 32 bit ile ifade edelim.

$$\begin{array}{l} 1111\,1111\,1111\,1101_{two} \\ + \qquad\qquad\qquad\qquad 1_{two} \\ \hline = 1111\,1111\,1111\,1110_{two} \end{array}$$

16 bitlik -2'yi 32 bitlik sayıya çevireceğimiz zaman anlamlı bitlerine '0' değil '1' eklememiz gerekmektedir.

$1111\,1111\,1111\,1111\,1111\,1111\,1111\,1110_{two} = -2_{ten}$

14

MIPS operands

Name	Example	Comments
32 registers	$s0-$s7, $t0-$t9, $sp, $gp, $fp, $zero, $ra, $at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zero always equals 0. Register $at is reserved for the assembler to handle large constants.
2^{30} memory words	Memory[0], Memory[4], ..., Memory[4294967293]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

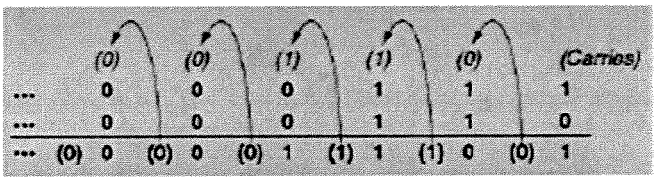
MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3	Three operands
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	+ constant
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory
	load half unsigned	lhu $s1,100($s2)	$s1 = Memory[$s2 + 100]	Halfword memory to register
	store half	sh $s1,100($s2)	Memory[$s2 + 100] = $s1	Halfword register to memory
	load byte unsigned	lbu $s1,100($s2)	$s1 = Memory[$s2 + 100]	Byte from memory to register
	store byte	sb $s1,100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	lui $s1,100	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND with constant
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; two's complement
	set less than immediate	slti $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; two's complement
	set less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; unsigned numbers
	set less than immediate unsigned	sltiu $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; unsigned numbers
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 20000	For procedure call

15

Toplama

6 ile 7 sayılarının ikilik sayı tabanında toplanması;



Basamaklar sağdan sola elde biti de göz önüne alınarak eklenerek soldaki basamağa geçer.

$$
\begin{array}{ll}
 & 0000\,0000\,0000\,0000\,0000\,0000\,0000\,0111_{two} = 7_{ten} \\
+ & 0000\,0000\,0000\,0000\,0000\,0000\,0000\,0110_{two} = 6_{ten} \\
\hline
= & 0000\,0000\,0000\,0000\,0000\,0000\,0000\,1101_{two} = 13_{ten}
\end{array}
$$

16

Çıkarma

Aynı örnek üzerinde çıkarma işlemini yapalım;

	0000 0000 0000 0000 0000 0000 0000 0111_{two}	$= 7_{ten}$
-	0000 0000 0000 0000 0000 0000 0000 0110_{two}	$= 6_{ten}$
=	0000 0000 0000 0000 0000 0000 0000 0001_{two}	$= 1_{ten}$

Çıkarma işleminde yukarıdaki gibi doğrudan çıkarma yapılabileceği gibi çıkarılan sayının 2'ye tümleyeni alınarak toplamada yapılabilir.

	0000 0000 0000 0000 0000 0000 0000 0111_{two}	$= 7_{ten}$
+	1111 1111 1111 1111 1111 1111 1111 1010_{two}	$= -6_{ten}$
=	0000 0000 0000 0000 0000 0000 0000 0001_{two}	$= 1_{ten}$

17

MIPS

- **`Add, addu`**
  - **İşaretli sayılarda toplama**
  - **İşaretsiz sayılarda toplama**
- **`Addi, addiu`**
  - **İşaretli sayılarda bir tamsayı ile toplama**
  - **İşaretsiz sayılarda**
- **`Sub, subu`**
  - **İşaretli sayılarda çıkarma**
  - **İşaretsiz sayılarda çıkarma**
  - **MIPS istisna nedeninin talimat adresini içeren EPC(exception program counter) olarak adlandırılan bir register içerir. sistem kontrolünden ayrılma komutu(MFC0-move from system control) EPCyi genel amaçlı bir register içine kopyalamak için kullanılır.**

18

Toplama ve çıkarma için overflow durumları;

Operation	Operand A	Operand B	Aşağıdaki sonuçlar da taşma oluşur
$A + B$	≥ 0	≥ 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	≥ 0	< 0	< 0
$A - B$	< 0	≥ 0	≥ 0

- **Bir işlem sonucunda elde edilen sayısal değer, sözcük uzunluğu ile tanımlanmış değerlerden büyükse, taşma (overflow) vardır denir.**
- **32 bitlik 2 sayı toplandığında/çıkarıldığında sonuç 33 bit ise;**
- **İki pozitif sayı toplandığında sonuç negatif ise (sonucun İşaret biti 1 ise);**
- **İki negatif sayı toplandığında sonuç pozitif ise (sonucun işaret biti 0 ise);**
- **Bir pozitiften bir negatif değer çıkartıldığında sonuç negatif ise (sonucun işaret biti 1 ise)**
- **Bir negatiften bir pozitif değer çıkartıldığında sonuç pozitif ise (sonucun işaret biti 0 ise)**

  overflow (taşma) veya underflow (borç alma) oluştuğu anlaşılır.
- **Bir negatif sayı bir pozitif sayıya eklendiğinde hiçbir zaman taşma olmaz.**
- **Çıkarılan sayılar aynı işaretli ise taşma olmaz.**

19

Taşma oluşumunun tanınması

– add, addi veya sub komutlarının sebep olduğu bir taşma interrupt 'ı (kesme – exception-istisna) oluşur.

•Kontrol, önceden tanımlanmış adresteki interrupt rutinine atlar.

•Interrupted addresi, mümkün yeniden başlatamalar için kaydedilir.

• MIPS **$epc (exception program counter) registeri içerir.** Bu register interrupt' a sebep olan komutun adresini tutar.
• Move from system control (Sistem kontrolundan ayrılma) komutu ile;

mfco $t0, $epc # copy $epcto register

$epc reg'in içeriği $t0 reg'e aktarılır.

• İşaretsiz sayılarda oluşan (**addu, addiu, subu komutları ile**) overflow'lardaki interrup'lar sürekli olarak sezilmek istenmez. Bu durumdaki taşma yok sayılır. Çünkü işaretsiz sayılarla işlemler genellikle ana hafıza adreslemeleri içindir.

Tavsiye:
QtSPIM'de : EPC ve ilgili reg'leri incele.

20

MIPS overflowu yakalayabilir, ancak birçok bilgisayarda overflow testi için dallanma şartı bulunmamaktadır. Overflow'u yakalayabilmek için MIPS kodu aşağıdaki gibi yazılır;

```
addu $t0, $t1, $t2 # $t0 = sum, but don't trap
xor  $t3, $t1, $t2 # Check if signs differ
slt  $t3, $t3, $zero # $t3 = 1 if signs differ
bne  $t3, $zero, No_overflow # $t1, $t2 signs ≠,
so no overflow
xor  $t3, $t0, $t1 # signs =; sign of sum match too?
                   # $t3 negative if sum sign different
slt  $t3, $t3, $zero # $t3 = 1 if sum sign different
bne  $t3, $zero, Overflow # All three signs ≠; go to
overflow
```

21

İşaretsiz sayılarda toplama için ($t0= $t1+$t2) overflow'u yakalayabilmek için yazılan MIPS kodu;

```
addu $t0, $t1, $t2 # $t0 = sum
nor  $t3, $t1, $zero # $t3 = NOT $t1
         # (2's comp - 1: 2^32 - $t1 - 1)
sltu $t3, $t3, $t2 # (2^32 - $t1 - 1) < $t2
        #⇒ 2^32 - 1 < $t1 + $t2
bne $t3,$zero,Overflow # if(2^32-1 < $t1 + $t2) go to
overflow
```

22

MIPS assembly language

Category	Instruction	Example		Meaning	Comments
Arithmetic	add	add	$s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow detected
	subtract	sub	$s1,$s2,$s3	$s1 = $s2 - $s3	Three operands; overflow detected
	add immediate	addi	$s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow detected
	add unsigned	addu	$s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow undetected
	subtract unsigned	subu	$s1,$s2,$s3	$s1 = $s2 - $s3	Three operands; overflow undetected
	add immediate unsigned	addiu	$s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow undetected
	move from coprocessor register	mf[illegible]	$s1,$[illegible]	$s1 = [illegible]	Used to copy floating-point or other special register
Data transfer	load word	lw	$s1,100($s2)	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	sw	$s1,100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory
	load half unsigned	lhu	$s1,100($s2)	$s1 = Memory[$s2 + 100]	Halfword memory to register
	store half	sh	$s1,100($s2)	Memory[$s2 + 100] = $s1	Halfword register to memory
	load byte unsigned	lbu	$s1,100($s2)	$s1 = Memory[$s2 + 100]	Byte from memory to register
	store byte	sb	$s1,100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	lui	$s1,100	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and	$s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or	$s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor	$s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi	$s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND with constant
	or immediate	ori	$s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR with constant
	shift left logical	sll	$s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl	$s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq	$s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne	$s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt	$s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; two's complement
	set less than immediate	slti	$s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; two's complement
	set less than unsign	sltu	$s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; unsigned
	set less than immediate unsigned	sltiu	$s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; unsigned
Unconditional jump	jump	j	2500	go to 10000	Jump to target address
	jump register	jr	$ra	go to $ra	For switch, procedure return
	jump and link	jal	2500	$ra = PC + 4; go to 10000	For procedure call

23

Çarpma

```
Multiplicand        1000
Multiplier        x 1001
                    1000
                   0000
                  0000
                 1000
Product        01001000
```

m bit x n bit = m+n bit çarpım (product)

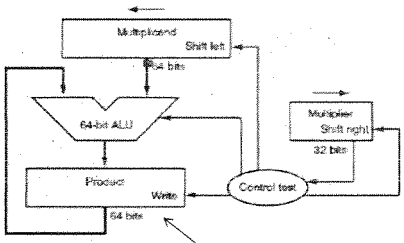
Binary 2 sayıyı çarpma işlemi;

- çarpan(multiplier) bit 1 => çarpılanı (multiplicand) kopyala (1 x multiplicand)
- çarpan bit 0 => yerine 0 koy (0 x multiplicand)

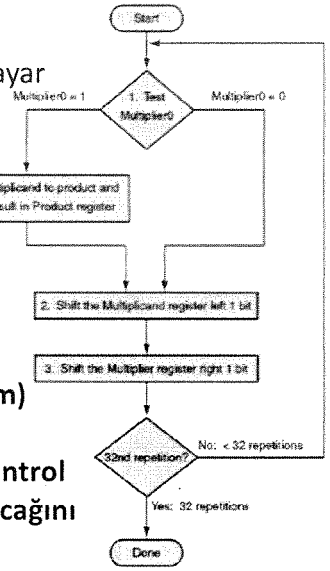
24

ÇARPMA İŞLEMİNİN DONANIMININ İLK VERSİYONU VE AKIŞ ŞEMASI

32-bit multiplicand(çarpılan) register'ın en anlamsız bitlerindedir ve her adımda sola kayar



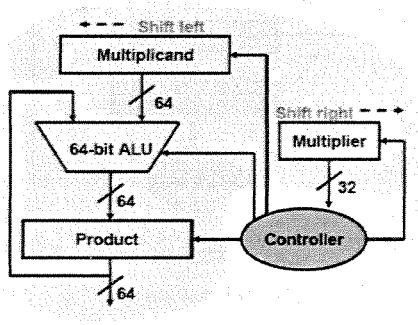
Çarpımın başlangıç değeri 0'dır

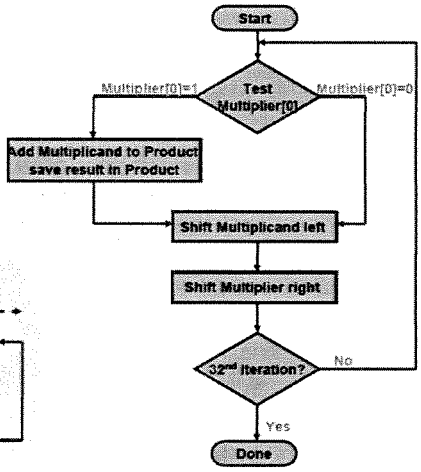


Multiplicand(çarpılan) register, product(çarpım) register ve ALU 64-bit genişliğindeyken; multiplier(çarpan) ise 32-bit uzunlukludur. Kontrol bloğu çarpılan ile çarpanın ne zaman kaydırılacağını ve sonucun ne zaman yenileneceğini belirtir.

25

-Çarpma işlemi, bir çok kez kaydırma ve 64 bit toplama işlemiyle başarılır.
-Basit fakat yavaş çalışır.
- Bu işlemi DATAPATh'te hangi parça yapar?

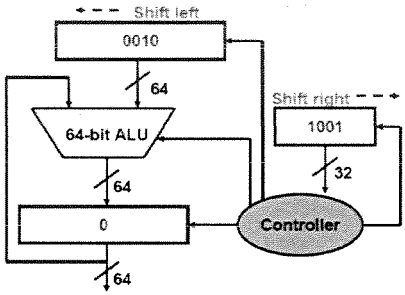


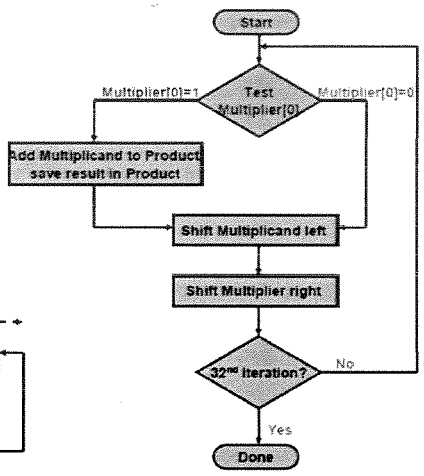


Control Flow

26

```
    0010   (multiplicand)
  x 1001   (multiplier)
   0010
  0000
 0000
+ 0010
 0010010   (product)
```

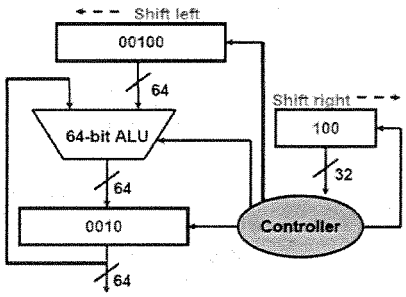


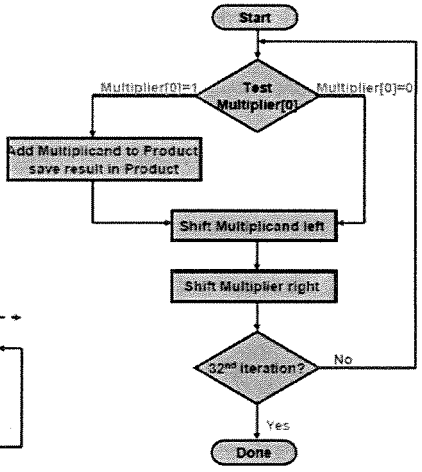


Control Flow
(Βοή Μονάδας Ελέγχου)

27

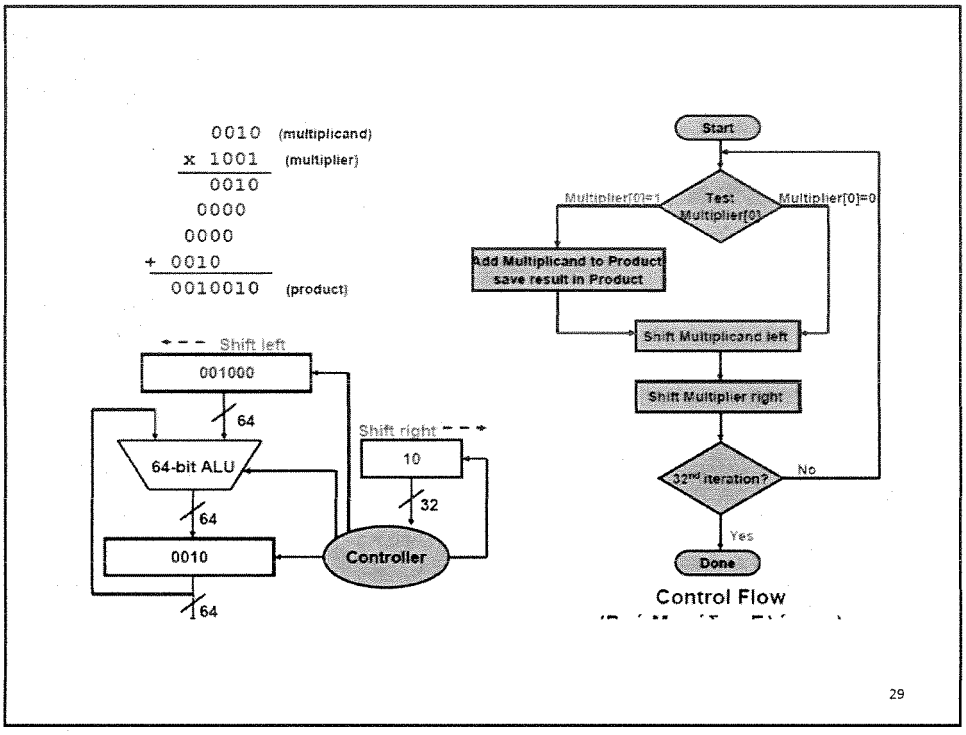
```
    0010   (multiplicand)
  x 1001   (multiplier)
   0010
  0000
 0000
+ 0010
 0010010   (product)
```

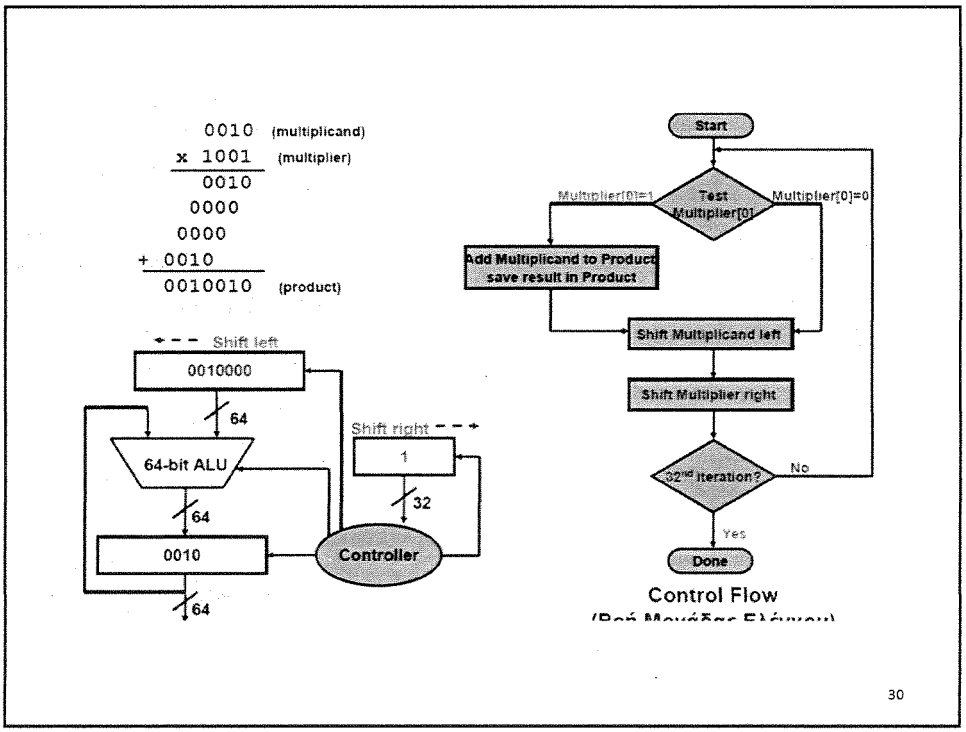


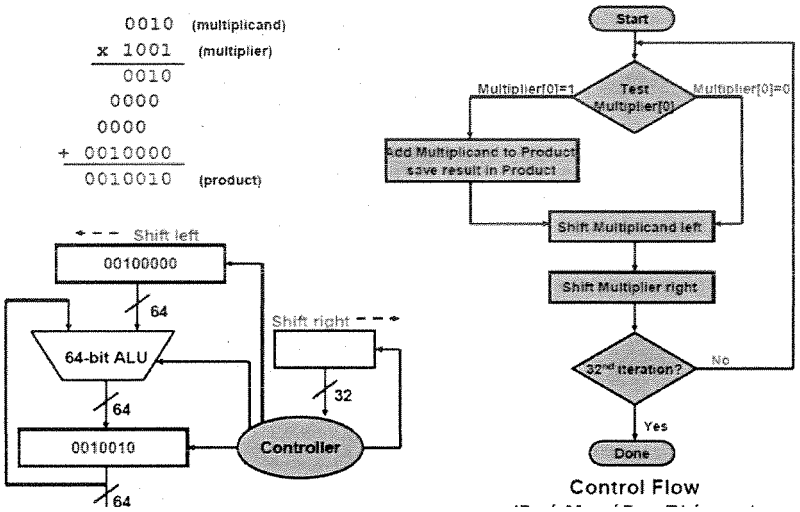


Control Flow
(Βοή Μονάδας Ελέγχου)

28


0010 (multiplicand)
x 1001 (multiplier)
0010
0000
0000
+ 0010
0010010 (product)
Shift left
001000
64
64-bit ALU
64
0010
64
Shift right
10
32
Controller
Start
Test Multiplier[0]
Multiplier[0]=1
Multiplier[0]=0
Add Multiplicand to Product save result in Product
Shift Multiplicand left
Shift Multiplier right
32nd iteration?
No
Yes
Done
Control Flow
29


0010 (multiplicand)
x 1001 (multiplier)
0010
0000
0000
+ 0010
0010010 (product)
Shift left
0010000
64
64-bit ALU
64
0010
64
Shift right
1
32
Controller
Start
Test Multiplier[0]
Multiplier[0]=1
Multiplier[0]=0
Add Multiplicand to Product save result in Product
Shift Multiplicand left
Shift Multiplier right
32nd iteration?
No
Yes
Done
Control Flow
30



31

Fix-point çarpma işlemi

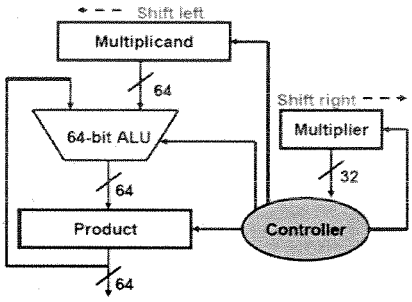
Multiplicand 0010
Multiplier x 0011

```
        0010
       0010
      0000
     0000
Product 00000110
```

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 ⇒ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 ⇒ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 ⇒ no operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 ⇒ no operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

32

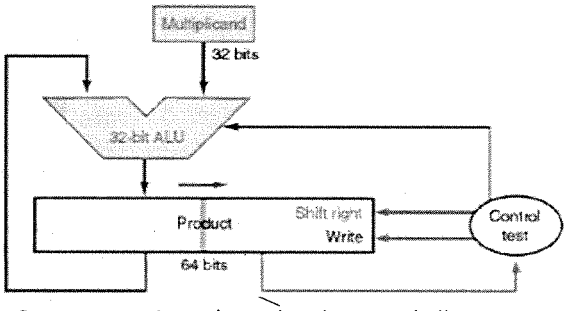
Bu çarpma devresi işaretli sayıların Çarpılması için kullanılabilir mi?



İşaretli sayıların çarpılması işlemi; işaretsiz sayıların çarpılması şeklindedir. Yani negatif sayı pzitifsayıya dönüştürülüp iki işaretsiz ssayının çarpılması şeklinde gerçekjleştirilir. Sonucun işareti ise işaretli sayılarınişaret bitlerine göre belirlenir.

33

ÇARPMA İŞLEMİNİN İŞLENMİŞ VERSİYONU



Çarpım register'ının başlangıç değeri 0

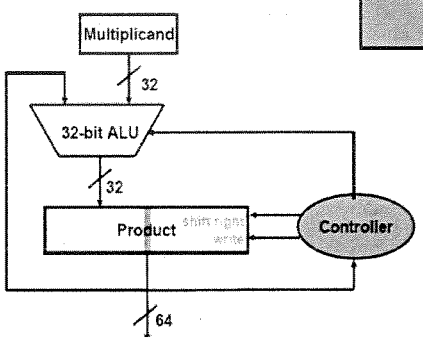
Multiplicand register, multiplier register ve ALU 32-bit uzunluğunda; product register ise 64-bit uzunluğundadır. Çarpım sonucu sağa kaydırılarak işlem devam eder.Çarpan bulunmadığı için multiplicand (çarpılan) çarpım register'ının en anlamlı 32 biti ile toplanır.

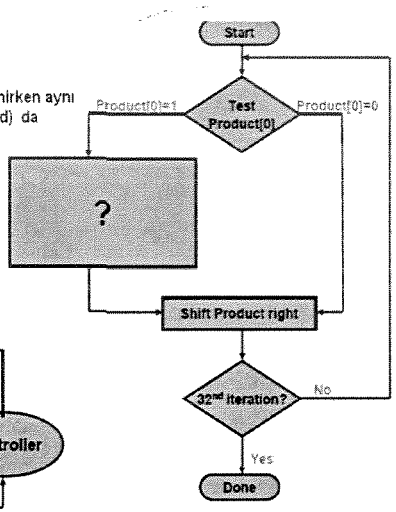
34

** Her adım(step) 1Cp'ma ihtiyaç duyar.Bu durumda 1.Çarpma devresine göre performans artmıştır.

**İşlem paralel başarılır.Çarpılan, Product reg'e eklenirken aynı zamanda Çarpan (multipler) ve çarpılan (multiplicand) da kaydırılır.

** Çarpıcı, Product reg'in ağırlıksız yarım 32 bitine yerleştirilmiştir.





Control Flow

35

İşlenmiş versiyona göre çarpma işlemi

Multiplicand 0010
Multiplier x 0011

```
        0010
       0010
      0000
     0000
```

Product 00000110

İterasyon	Step	Multiplier	Multiplicand	Product
0	init values	0011	0010	0000 0000
1	1a	0011	0010	0010 0000
	2	0011	0010	0001 0000
	3	0001	0010	0001 0000
2	1a	0001	0010	0011 0000
	2	0001	0010	0001 1000
	3	0000	0010	0001 1000
3	1a	0000	0010	0001 1000
	2	0000	0010	0000 1100
	3	0000	0010	0000 1100
4	1a	0000	0010	0000 1100
	2	0000	0010	0000 0110
	3	0000	0010	0000 0110

36

MIPS

- 64 bitlik çarpımı ifade edebilmek için `Hi` ve `Lo` adı verilen 2 adet 32-bitlik register kullanılır.
- `mult`, `multu`

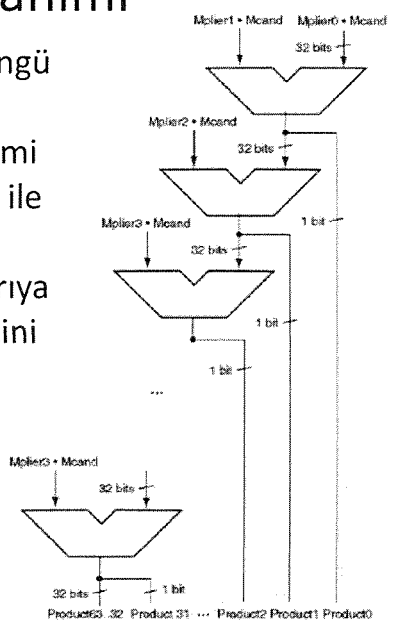
  İşaretli ve işaretsiz sayılarda çarpma işlemi
- `mflo`, `mfhi`

  bu komutlar sanal (pseudoinstruction)komutlar olup yukarıda verilen mult ve multu komutları çalıştırıldığında bu komutlar 64 bitlik sonucu 32 bitlik Hi ve Lo olmak üzere 2 register'a taşır .

37

Hızlı çarpma donanımı

32 bitlik bir toplayıcı ile döngü kurarak 32 adet toplama yapmak yerine çarpma işlemi döngü açılarak 32 toplayıcı ile yandaki şekilde yapılabilir. Buradaki her toplayıcı dışarıya 32 bitlik toplam ile elde bitini göndermektedir.



38

Bölme

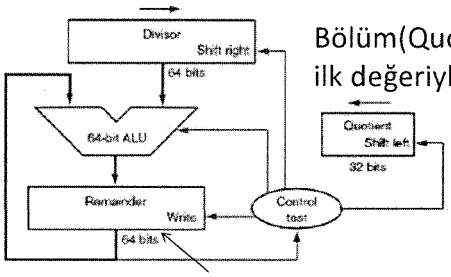
```
                        1001     Quotient(Bölüm)
Divisor(Bölen) 1000  |1001010    Dividend (Bölünen)
                    -1000
                        10
                        101
                        1010
                       -1000
                          10     Remainder(Kalan)
```

Dividend = (Quotient * Divisor) + Remainder

39

Bölme donanımının ilk versiyonu

32 bitlik bölen (Divisor), bölen register'ının sol yarısından başlar ve her adımda sağa kaydırılır.



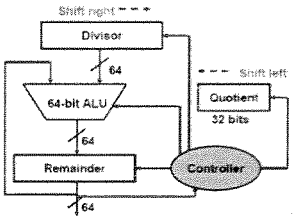
Bölüm(Quotient) registeri 0 ilk değeriyle başlamaktadır.

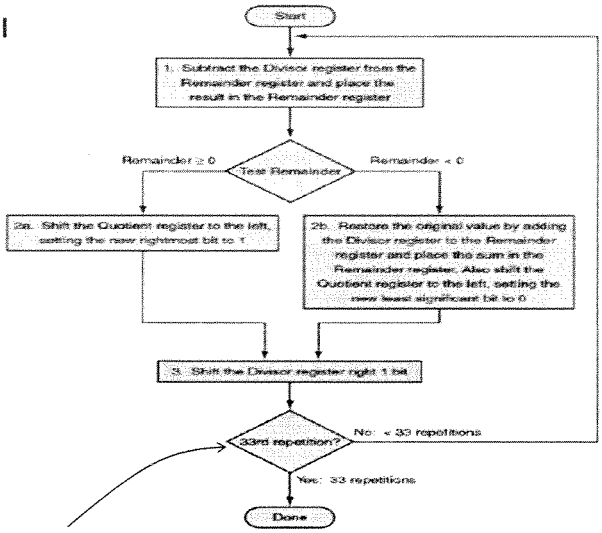
Kalan register'ı bölünen registerının değeriyle başlar.

Divisor(bölen) register, remainder(kalan) register ve ALU 64-bit uzunluğunda; quotient(bölüm) registeri 32-bit uzunluğundadır. Kontrol bölen ve bölümün ne zaman kaydırılacağı ve kalan registerına yeni değerin ne zaman yazılacağını belirler.

40

Bölme Algoritması



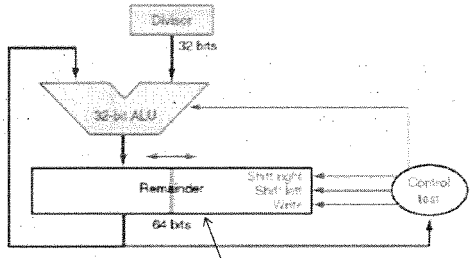


Neden 33?

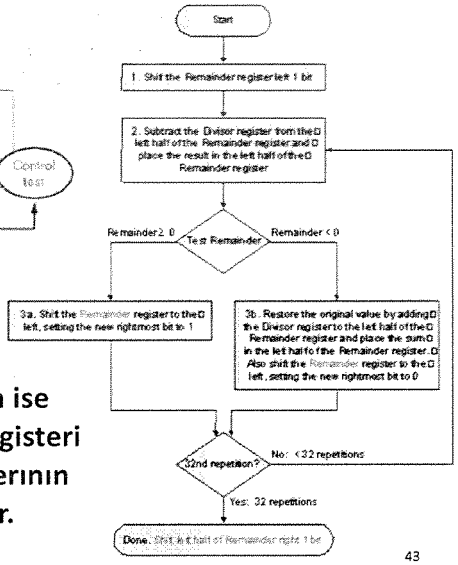
00000111_2 (7)/ 0010_2 (2)

Iteration	Step	Bölüm Quotient	Bölen Divisor	Kalan Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	1110 0111
	2b: Rem < 0 ⇒ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	1111 0111
	2b: Rem < 0 ⇒ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	1111 1111
	2b: Rem < 0 ⇒ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	0000 0011
	2a: Rem ≥ 0 ⇒ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	0000 0001
	2a: Rem ≥ 0 ⇒ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

Bölme donanımının gelişmiş versiyonu



Kalan registerının anlamlı bitlerinde bölüm registerının değeri bulunur.



Bölen, ALU ve bölüm 32 bitlik, kalan ise 64 bitliktir. Bu donanımda bölüm registeri bulunmamakta, bölüm kalan registerının en anlamlı bitlerinde bulunmaktadır.

43

MIPS

- **div, divu**

  iki tane 32 bitlik operand registeri ile işlenen içerik bölünür, kalan Hi registerına bölüm ise Lo registerına aktarılır taşma her iki durumda da göz ardı edilir.

  Sözde talimatlar kullanılarak div (signed with overflow), divu (unsigned without overflow) 3 tane 32 bitlik register ile iki registerın bölümü 3 . registera aktarılır.

44

MIPS assembly language

Category	Instruction	Example		Meaning	Comments
Arithmetic	add	add	$s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow detected
	subtract	sub	$s1,$s2,$s3	$s1 = $s2 - $s3	Three operands; overflow detected
	add immediate	addi	$s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow detected
	add unsigned	addu	$s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow undetected
	subtract unsigned	subu	$s1,$s2,$s3	$s1 = $s2 - $s3	Three operands; overflow undetected
	add immediate unsigned	addiu	$s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow undetected
	move from coprocessor register	mfc0	$s1,$epc	$s1 = $epc	Copy Exception PC + special regs
	multiply	mult	$s2,$s3	Hi, Lo = $s2 * $s3	64-bit signed product in Hi, Lo
	multiply unsigned	multu	$s2,$s3	Hi, Lo = $s2 * $s3	64-bit unsigned product in Hi, Lo
	divide	div	$s2,$s3	Lo = $s2 / $s3, Hi = $s2 % $s3	Lo = quotient, Hi = remainder
	divide unsigned	divu	$s2,$s3	Lo = $s2 / $s3, Hi = $s2 % $s3	Unsigned quotient and remainder
	move from Hi	mfhi	$s1	$s1 = Hi	Used get high part of 64
	move from Lo	mflo	$s1	$s1 = Lo	Used to get high part of 64
Data transfer	load word	lw	$s1,100($s2)	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	sw	$s1,100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory
	load half unsigned	lhu	$s1,100($s2)	$s1 = Memory[$s2 + 100]	Halfword memory to register
	store half	sh	$s1,100($s2)	Memory[$s2 + 100] = $s1	Halfword register to memory
	load byte unsigned	lbu	$s1,100($s2)	$s1 = Memory[$s2 + 100]	Byte from memory to register
	store byte	sb	$s1,100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	lui	$s1,100	$s1 = 100 * $2^16	Loads constant in upper 16 bits
Logical	and	and	$s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or	$s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor	$s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi	$s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND with constant
	or immediate	ori	$s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR with constant
	shift left logical	sll	$s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl	$s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq	$s1,$s2,25	if($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne	$s1,$s2,25	if($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt	$s1,$s2,$s3	if($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; two's complement
	set less than immediate	slti	$s1,$s2,100	if($s2 < 100) $s1 = 1; else $s1=0	Compare < constant; two's complement
	set less than unsigned	sltu	$s1,$s2,$s3	if($s2 < $s3) $s1 = 1; else $s1=0	Compare less than; natural numbers
	set less than immediate unsigned	sltiu	$s1,$s2,100	if($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; natural numbers
Unconditional jump	jump	j	2500	go to 10000	Jump to target address
	jump register	jr	$ra	go to $ra	For switch, procedure return
	jump and link	jal	2500	$ra = PC + 4; go to 10000	For procedure call

45

FLOATING POINT SAYILAR

46

Floating Point Sayılar

- IEEE 754 floaing point formatı günümüzde gerçek sayıları temsil etmek üzere kullanılan en yaygın gösterim şeklidir.
- Bilgisayarda gerçek sayıları temsil etmenin birkaç farklı yolu vardır.Örneğin fixed point sayı gösterimi ile her sayı 2 integer sayının oranı şeklinde gösterilebilir.
- IEEE 754 floating point formatında ise gerçek sayılar bilimsel bir notasyonla gösterilir.

47

Floating Point

Aşağıda verilen sayıları tanımlayabilmek için floating point sayılar kullanılır;

- Kesirli sayılar, örneğin 3.1416
- Çok küçük sayılar, örneğin .00000000023
- Ve çok büyük sayılar, örneğin $-3.15576 * 10^{46}$

48

Floating point sayı gösterimi

Floating point sayıların gösterimi 3 bölümden oluşur. *Bu bölümler:* sign(işaret), exponent(üs) ve significand(fraction,mantissa-kesir) olarak ifade edilir.

Sign Biti: Sign biti pozitif sayılar için 0, negatif sayılar için 1 değerini alır.

Exponent: Exponent alanı hem negatif hem de pozitif üsleri temsil edebilmektedir.Bunu gerçekleştirmek için bir **bias değeri** gerçek üs değeriyle toplanıp exponent kısmı oluşturulur.

IEEE 32 bitlik gösterimi için ***bias değeri*** 127, 64 bitlik gösterim için de 1023 tür.Buna göre gerçek üssün 0 olması exponent alanında saklanan değerin 127 olacağı anlamına gelir.

Mantissa: Mantissa sayıyı ifade eden bitleri gösterir. Bu,sayının tam ve kesir(fraction) kısımlarını gösteren bitlerden oluşur.

49

Floating point sayılar aşağıdaki gibi gösterilir:

$(-1)^{sign} * significand * 2^{exponent}$

Örneğin; $-101.001101 * 2^{111001}$

- *significand* bölümü ne kadar bit ile ifade edilirse sayı gerçeğe o kadar yakın olur.
- *exponent* bölümü ne kadar bitle ifade edilirse, değer aralığı o kadar artar.

Floating point sayılar $\mathbf{1 \leq significand < 10_{two}}(=2_{ten})$ aralığında ifade edilir. Bu aralıkta olmayan sayıların normalize edilmeleri gerekmektedir.Örneğin; $\mathbf{-101.001101 * 2^{111001}}$ sayısının floating point olarak gösterilişi (normalize edilmiş hali), $\mathbf{-1.01001101 * 2^{111011}}$ şeklindedir.

50

Floating point standartında gösterilecek olan sayıların normalize edilmesi zorunludur. Sayıların normalize edilmesindeki amaç aynı sayıyı daha az bit ile ifade etmektir. Bu nedenle binary formda ifade edilecek her sayı aşağıdaki formatta olmak zorundadır.

$$1.xxxxxxx_2 x\ 2^{yyyyy}$$

Örneğin binary bir sayı olan

11110000 11001100 10101010 00000000;

+ 1.1110000 11001100 10101010 x 2^{31}

şeklinde gösterilir.

51

IEEE 754 Floating-point Standard

IEEE 754 floating point standartında 32 bitlik ve 64 bitlik olmak üzere 2 tür gösterim mevcuttur.Her 2 gösterim için sign, exponent ve mantissanın durumları aşağıda verilmiştir.

- Single precision (tek duyarlıklı): tek kelime (32 bit)

31	bits 30 to 23	bits 22 to 0
sign	8-bit exponent	23-bit fraction

- Double precision (çift duyarlıklı): iki kelime (64 bit)

31	bits 30 to 20	bits 19 to 0
sign	11-bit exponent	upper 20 bits of 52-bit fraction

bits 31 to 0
lower 32 bits of 52-bit fraction

52

Representation of Floating Point Numbers in Single Precision *IEEE 754 Standard*

$$\text{Value} = N = (-1)^S \times 2^{E-127} \times (1.M)$$

	1	8	23
sign	S	E	M

$0 < E < 255$
Actual exponent is:
$e = E - 127$

exponent: excess 127 binary integer added

mantissa: sign + magnitude, normalized binary significand with a hidden integer bit: 1.M

Example: 0 = 0 00000000 0 . . . 0 -1.5 = 1 01111111 10 . . . 0

Magnitude of numbers that can be represented is in the range: $2^{-126}(1.0)$ to $2^{127}(2 - 2^{-23})$

Which is approximately: 1.8×10^{-38} to 3.40×10^{38}

53

Representation of Floating Point Numbers in Double Precision *IEEE 754 Standard*

$$\text{Value} = N = (-1)^S \times 2^{E-1023} \times (1.M)$$

	1	11	52
sign	S	E	M

$0 < E < 2047$
Actual exponent is:
$e = E - 1023$

exponent: excess 1023 binary integer added

mantissa: sign + magnitude, normalized binary significand with a hidden integer bit: 1.M

Example: 0 = 0 00000000000 0 . . . 0 -1.5 = 1 01111111111 10 . . . 0

Magnitude of numbers that can be represented is in the range: $2^{-1022}(1.0)$ to $2^{1023}(2 - 2^{-52})$

Which is approximately: 2.23×10^{-308} to 1.8×10^{308}

54

IEEE 754 Floating-point Standard

- Sayı normalize kabul edilir ve sayıda noktanın sol tarafında yer alan 1 biti fraction kısmında gösterilmez.
  - Yani significand kısmı 1.1001... olan sayı 1001... şeklinde ifade edilir.
  - Sayıların temsili:

sayı değeri = $(-1)^{sign} * (1 + fraction) * 2^{exponent\ value}$

- Sayının değerini daha kolay bir şekilde bulabilmek için exponent biaslanır.
  - Exponent tamamen 0lardan oluşmuşsa en küçük, 1'lerden oluşmuşsa en büyüktür.
  - single precision için bias değeri 127, double precision ise 1023'tür.
  - Sayı temsili;

sayı değeri = $(-1)^{sign} * (1 + fraction) * 2^{(exponent - bias)}$

55

Örnek 1;

-0.75_{ten} sayısını IEEE 754 standartına göre single precision ve double precision formatında ifade edelim.

Desimal değer: $-0.75 = -3/4 = -3/2^2$

Binary karşılığı: $-11/100 = -.11 = -1.1 \times 2^{-1}$

IEEE single precision floating point exponent

= bias + exponent değeri

$= 127 + (-1) = 126_{ten}$

$= 01111110_{two}$

56

IEEE 754 single precision formunda 0.75:

$(-1)^1 * (1 + .1000\ 0000\ 0000\ 0000\ 0000\ 0000) * 2^{(126-127)}$

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
1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 bit 8 bits 23 bits

IEEE 754 double precision formunda 0.75:

$(-1)^1 * (1 + .1000\ 0000\ 0000\ 0000\ 0000...0000) * 2^{(1022-1023)}$

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
1	0	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 bit 11 bits 20 bits

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

32 bits

57

Örnek 2; -2345.125_{10} desimal sayısını IEEE 754 standartında 32 bitlik formatta ifade edelim:

$-2345.125_{10} = -100100101001.001_2$ (binary karşılığı)

$= -1.00100101001001_2 \times 2^{11}$ (normalize binary)

yazılmaz

-Mantissa negatif olduğu için işaret biti 1 olacaktır. (S=1)

-32 bit için bias değeri 127 idi, binary sayının exponenti=11

$E=127+11=138_{10} = 10001010_2$ (8 bit)

-Kesir kısmı (fraction)= .00100101001001000000000 (23 bit)

Sign	Exponent	Fraction
1	10001010	00100101001001000000000
1bit	8bit	23bit

58

Örnek 3; Aşağıda verilen ikilik tabandaki floating point sayıyı desimal sayıya çevirelim.

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
1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.	.	.

İşaret biti:1, exponent alanının değeri:129,
fraction alanı: $1x2^{-2}$ = ¼=0.25

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})} = (-1)^1 \times (1 + 0.25) \times 2^{(129-127)}$$
$$= -1 \times 1.25 \times 2^2$$
$$= -1.25 \times 4$$
$$= -5.0$$

59

IEEE 754 formatında sayı gösterimi

Single precision		Double precision		Object represented
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	0
0	nonzero	0	nonzero	± denormalized number
1-254	anything	1-2046	anything	± floating-point number
255	0	2047	0	± infinity
255	nonzero	2047	nonzero	NaN (Not a Number)

0 özel durumu:

- Exponent ve fraction tamamen 0'lardan oluşmuşsa, işaret biti ne olursa olsun sayının değeri 0'dır.
- Exponent 0 ve fraction 0'dan farklı ise sayının değeri;

$(-1)^{sign} * (1 + \text{fraction}) * 2^{-127}$ 'dir.

60

Matlab ortamında floating-point sayılar

```
>> q= quantizer('single','nearest','saturate');
```

Floating-point sayının özelliklerinin belirlendiği fonksiyon

```
>> num2hex(single(-0.75))
ans =  bf400000
```

Desimal sayının floating point karşılığını veren komut

1 01111110 10000000000000000000000

floating point sayının desimal karşılığını veren komut

```
>> hex2num(q,'C5129200')
```

1 10001010 00100101001001000000000

```
ans = -2.3451e+003
```

61

Floating Point Toplama

Örnek; $1{,}234{,}823.333_{10} + .0011_{10} = ?$

Floating point sayıların toplanabilmesi için öncelikle kesir kısımlarının toplanması gerekir ancak toplamaya başlanılmadan önce sayıların exponentlerinin aynı olması sağlanmalıdır. Bu nedenle toplama işlemine geçilmeden önce üslerin eşitlenmesi gerekir.

$1{,}234{,}823.333 = 1.234823333 \times 10^6$

$.0011 = 1.1 \times 10^{-3} = 0.0000000011 \times 10^6$

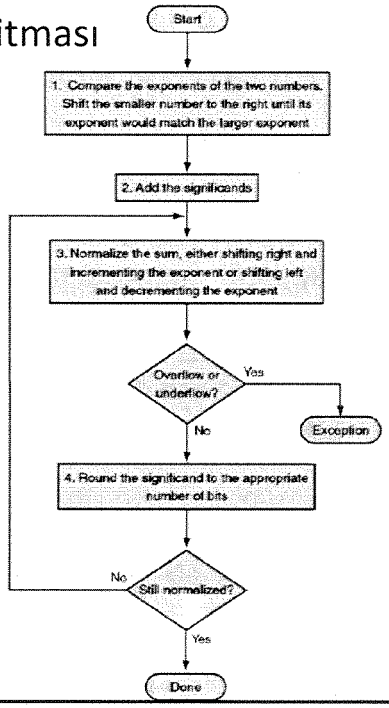
62

Toplanacak sayıların üsleri eşitlendikten sonra önce kesir kısmı ardından tam kısımları toplanır.

$$
\begin{array}{r}
1.2348233330 \times 10^6 \\
+\ \underline{0.0000000011 \times 10^6} \\
1.2348233341 \times 10^6
\end{array}
$$

63

Toplama işleminin Algoritması



64

Verilen algoritmaya göre toplama işleminin adımları;

- **1. adım:** toplanacak olan sayıların exponentleri eşitlenir. küçük olan sağa kaydırılarak büyük olan exponente eşitlenir.
- **2.adım:** sayıların fraction(mantisa) kısımları toplanır.
- **3. adım:** elde edilen toplam normalize edilir.
- **4. adım:** elde edilen toplam değeri olması istenen uzunluğa(4 bit) getirilir.

65

Örnek 2; 0.5 + (-0.4375) =?

Toplam 4 dijitle ifade edilsin.

Öncelikle sayıların binary karşılıklarını hesaplayalım:

$$
\begin{aligned}
0.5_{ten} &= 1/2_{ten} &&= 1/2^{1}{}_{ten} \\
&= 0.1_{two} &&= 0.1_{two} \times 2^{0} && = 1.000_{two} \times 2^{-1} \\
-0.4375_{ten} &= -7/16_{ten} &&= -7/2^{4}{}_{ten} \\
&= -0.0111_{two} &&= -0.0111_{two} \times 2^{0} && = -1.110_{two} \times 2^{-2}
\end{aligned}
$$

66

1. adım: exponentlerin eşitlenmesi

$$-1.110_{two} \times 2^{-2} = -0.111_{two} \times 2^{-1}$$

2. adım: fraction kısımlarının toplanması

$$1.000_{two} \times 2^{-1} + (-0.111_{two} \times 2^{-1}) = 0.001_{two} \times 2^{-1}$$

3. adım: toplamın normalize edilmesi

$$\begin{aligned} 0.001_{two} \times 2^{-1} &= 0.010_{two} \times 2^{-2} = 0.100_{two} \times 2^{-3} \\ &= 1.000_{two} \times 2^{-4} \end{aligned}$$

67

4. adım: overflow,underflow oluşmuş mu?

Üs değeri 127≥-4≥-126 arasında olduğu sürece overflow veya underflow oluşmaz.

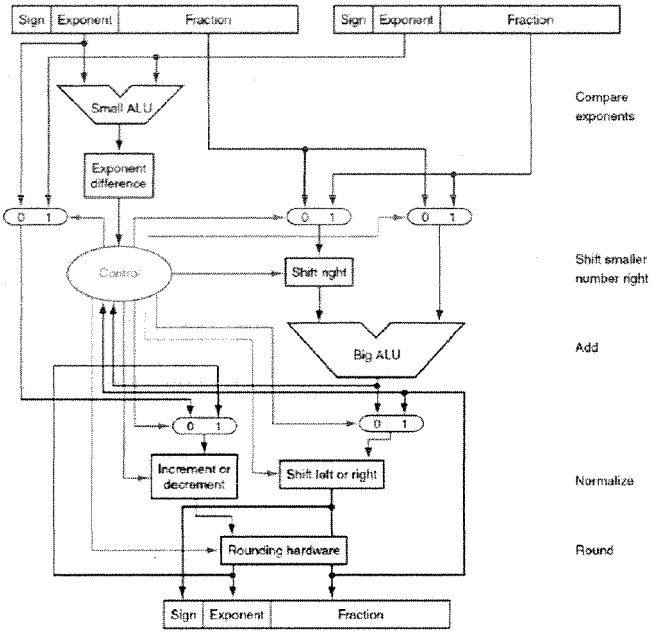
$$1.000_{two} \times 2^{-4}$$

Sonuç olarak elde edilen değerin desimal karşılığı;

$$\begin{aligned} 1.000_{two} \times 2^{-4} &= 0.0001000_{two} = 0.0001_{two} \\ &= 1/2^{4}{}_{ten} = 1/16_{ten} = 0.0625_{ten} \end{aligned}$$

68

Floating point toplama için donanımın blok şeması



69

Örnek; 2345.125_{10} sayısı ile $.75_{10}$ sayısını IEEE 754 standartında 32 bitlik formatta toplayınız.

$X = 2345.125_{10}$

Sign	Exponent	Fraction
0	10001010	00100101001001000000000

$Y = 0.75_{10}$

Sign	Exponent	Fraction
0	01111110	10000000000000000000000

1.adım: Exponentlerin büyük olana eşitlenmesi

$E_x > E_y$ $\quad E = 10001010 = 138_{10}$

$E_x - E_y = 10001010 - 01111110 = 00000110 = 12_{10}$

Fy 12 bit sağa kaydırılacak.

$F_y \times 2^{-12} = 0.000000000000110000000000$

70

2.adım: kesirlerin toplanması

$Fx + (Fy \times 2^{-12}) = 1.00100101001001000000000$

$+ 0.00000000000110000000000$

$= 1.00100101001111000000000$

3. adım: toplam normalize mi? Evet

4.adım: overflow veya underflow oluşmuş mu?hayır

5.adım: sonuç 0 mı? Hayır

Sonuç:

Sign	Exponent	Fraction
0	10001010	00100101001111000000000
1bit	8bit	23bit

71

Floating point sayılarda Çarpma

Örnek 1; $1.110_{10} \times 10^{10} \times 9.200_{10} \times 10^{-5}$ dört dijitlik sayıları çarpalım (IEEE 754 single precision).

1.adım: toplamadan farklı olarak üsler toplanır. Öncelikle biaslanmış exponentleri hesaplamalıyız.

10+127=137 ve (-5)+127= 122

Yeni exponent= 137+122= 259

259 sayısını 8 bitle nasıl ifade edeceğiz?

72

Biaslanmış sayıları toplarken bias değerini çıkarmayı unutmamamız gerekir, dolayısıyla;

yeni exponent= 137+122-127 = 132

ya da yeni exponent= 10 + (-5)+ 127 = 132 olur.

2.adım: significand kısımları çarpılır.

$$
\begin{array}{r}
1.110 \\
\times\ 9.200 \\
\hline
0000 \\
0000 \\
2220 \\
+\ 9990 \\
\hline
10212000_{10}
\end{array}
$$

Çarpım sonucu= $10.212000_{10}=10.212_{10}x10^5$

73

3. adım: sonucun normalize edilmesi

$10.212_{10}x10^5=1.0212_{10}x10^6$

4.adım: significant kısmı 4 dijit olduğu için sayının yuvarlanması gerekir.

$1.0212_{10}x10^6=1.021_{10}x10^6$

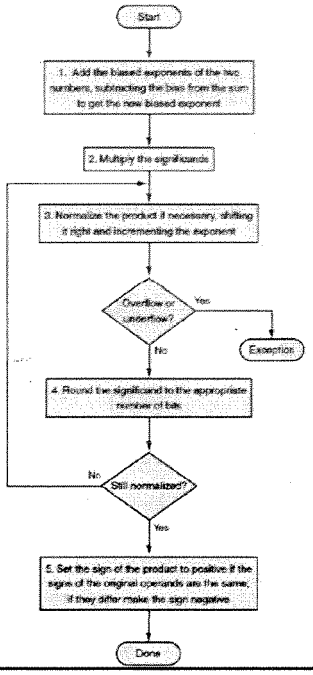
5.adım: işaret bitinin değerinin bulunması. Çarpılan sayıların işaretleri aynı ise sonuç pozitif(S=0), farklı ise sonuç negatif (S=1) olur.

Sonuç= $+1.021_{10}x10^6$

74

Çarpma Algoritması

Genel olarak 3 ve 4. adımlar birer kez gerçekleştirilir, ancak normalizasyon yapılmamışsa 3. adım tekrarlanmalıdır.



75

Örnek 2; Verilen algoritmanın adımlarını takip ederek 0.5_{10} ile -0.4375_{10} sayılarını çarpalım.

Sayıların binary karşılıkları; $1.000_2 x 2^{-1}$ ve $-1.110_2 x 2^{-2}$

1. Adım: exponentlerin toplanması

Bias olmadan -1 +(-2)= -3

Bias ile; (-1+127)+(-2+127)-127=124

2. Adım: significand kısımlarının çarpımı

$$
\begin{array}{r}
1.000_2 \\
\times\ 1.110_2 \\
\hline
0000 \\
1000 \\
1000 \\
+\ 1000 \\
\hline
1110000_2
\end{array}
$$

çarpım= $1.110000 x 2^{-3}$

76

3. Adım: çarpım sonucu normalize mi? **Evet**

İşlem sonucunda overflow oluşmuş mu?

127≥-3≥-126 aralığında olduğundan **hayır**

Bias edilen için 254≥124≥1 aralığında olduğundan **hayır**

Underflow oluşmuş mu? **Hayır**

4. Adım: çarpım değerini 4 bit ile ifade etmemiz gerekiyor. Dolayısı ile çarpım= $1.110000 \times 2^{-3} = 1.110 \times 2^{-3}$

5. Adım: çarpılan sayıların işaretleri farklı olduğu için sonuç negatif olacaktır.

Sonuç = -1.110×2^{-3}

Sonucu onluk tabana çevirerek değerini kontrol edelim;

Sonuç = $-1.110 \times 2^{-3} = -0.001110_2 = -0.00111_2$

$= -7/2^5{}_{10} = -7/32_{10} = \mathbf{-0.21875_{10}}$

77

Örnek 3; -18_{10} ve 9.5_{10} sayılarını IEEE 754 single precision formatında çarpalım.

X= -18 = 10010_2

1	10000011	00100000000000000000000

Y=9.5 = $1001{,}1_2$

0	10000010	00110000000000000000000

(1) Exponentleri topla

$E_x + E_y - 127_{10} = 1000\ 0011 + 1000\ 0010 - 01111111$

$= 1000\ 0110$

(2) Fraction kısımlarının çarpımı

1.0010 0000... 0000 x 1.0011 0000 ... 0000

$= 01.\underbrace{0101\ 0110\ 0000 \ldots 0000\ 0000}_{\text{46 bit uzunluğunda}}$

78

Sonuç single precision olacağı için fraction kısmı 24 bit uzunluğuna indirilmelidir.

Fraction = **1.0101 0110 0000 0000 0000 000**

(3) Fraction kısmının normalize edilmesi gerekmiyor.

(4) overflow? **Yok** underflow? **Yok**

(5) İşaret biti

Sx XOR Sy = 1 xor 0 = 1

Sonuç;

S	E	F
1	1000 0110	0101 0110 0000 0000 0000 000

79

MIPS

MIPS'te otuziki tane 32bitlik register($f0 - $f31) içeren *floating point coprocessor(yardımcı işlemci)* bulundurmaktadır. IEEE 754 single precision ve double precision formatlarında işlem yapmak için aşağıdaki kodlar registerları ile birlikte kullanılmalıdır.

- **Toplama, single (add. s), double (add . d)**
- **Çıkarma, single (sub. s), double (sub .d)**
- **Çarpma, single (mul. s), double (mul .d)**
- **Bölme, single (div. s), double (div .d)**
- **Karşılaştırma, single (c .x. s), double (c. x .d)**

x değişkeni equal(eq), not equal(neq), less than(lt), less than or equal(le), greater than(gt) veya greater than or equal(ge) olabilir.

- **branch, true (bclt), false (bclf)**

80

Floating-point comparison(karşılaştırma) ve Branch şarta bağlı olarak bir biti true veya false yapar.

lwcl ve swcl komutları ile floating point single ve double precision registerlarına ($f*0,* $f1, $f2, ...)ayrılarak yazılmasını sağlayan komutlardır. Verilen MIPS kodu, 2 single precision sayıyı hafızadan alıp(load), topladıktan sonra hafızaya yollamaktadır(store).

```
lwcl   $f4,x($sp)      #   Load 32-bit F.P. number into F4
lwcl   $f6,y($sp)      #   Load 32-bit F.P. number into F6
add.s  $f2.$f4.$f6     #   F2 = F4 + F6 single precision
swcl   $f2.z($sp)      #   Store 32-bit F.P. number from F2
```

81

MIPS floating-point **operands**

Name	Example	Comments
32 floating-point registers	$f0, $f1, $f2, . . . , $f31	MIPS floating-point registers are used in pairs for double precision numbers.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS floating-point **assembly language**

Category	Instruction	Example	Meaning	Comments
Arithmetic	FP add single	add.s $f2,$f4,$f6	$f2 = $f4 + $f6	FP add (single precision)
	FP subtract single	sub.s $f2,$f4,$f6	$f2 = $f4 - $f6	FP sub (single precision)
	FP multiply single	mul.s $f2,$f4,$f6	$f2 = $f4 × $f6	FP multiply (single precision)
	FP divide single	div.s $f2,$f4,$f6	$f2 = $f4 / $f6	FP divide (single precision)
	FP add double	add.d $f2,$f4,$f6	$f2 = $f4 + $f6	FP add (double precision)
	FP subtract double	sub.d $f2,$f4,$f6	$f2 = $f4 - $f6	FP sub (double precision)
	FP multiply double	mul.d $f2,$f4,$f6	$f2 = $f4 × $f6	FP multiply (double precision)
	FP divide double	div.d $f2,$f4,$f6	$f2 = $f4 / $f6	FP divide (double precision)
Data transfer	load word copr. 1	lwc1 $f1,100($s2)	$f1 = Memory[$s2 + 100]	32-bit data to FP register
	store word copr. 1	swc1 $f1,100($s2)	Memory[$s2 + 100] = $f1	32-bit data to memory
Conditional branch	branch on FP true	bc1t 25	if (cond == 1) go to PC + 4 + 100	PC-relative branch if FP cond.
	branch on FP false	bc1f 25	if (cond == 0) go to PC + 4 + 100	PC-relative branch if not cond.
	FP compare single (eq, ne, lt, le, gt, ge)	c.lt.s $f2,$f4	if ($f2 < $f4) cond = 1; else cond = 0	FP compare less than single precision
	FP compare double (eq, ne, lt, le, gt, ge)	c.lt.d $f2,$f4	if ($f2 < $f4) cond = 1; else cond = 0	FP compare less than double precision

82

MIPS floating-point **machine language**

Name	Format	Example						Comments
add.s	R	17	16	6	4	2	0	add.s $f2,$f4,$f6
sub.s	R	17	16	6	4	2	1	sub.s $f2,$f4,$f6
mul.s	R	17	16	6	4	2	2	mul.s $f2,$f4,$f6
div.s	R	17	16	6	4	2	3	div.s $f2,$f4,$f6
add.d	R	17	17	6	4	2	0	add.d $f2,$f4,$f6
sub.d	R	17	17	6	4	2	1	sub.d $f2,$f4,$f6
mul.d	R	17	17	6	4	2	2	mul.d $f2,$f4,$f6
div.d	R	17	17	6	4	2	3	div.d $f2,$f4,$f6
lwc1	I	49	20	2	100			lwc1 $f2,100($s4)
swc1	I	57	20	2	100			swc1 $f2,100($s4)
bc1t	I	17	8	1	25			bc1t 25
bc1f	I	17	8	0	25			bc1f 25
c.lt.s	R	17	16	4	2	0	60	c.lt.s $f2,$f4
c.lt.d	R	17	17	4	2	0	60	c.lt.d $f2,$f4
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits

op(31:26):

28–26 31–29	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	Rfmt	Blt/gez	j	jal	beq	bne	blez	bgtz
1(001)	addi	addiu	slti	sltiu	andi	ori	xori	lui
2(010)	TLB	FIPs						
3(011)								
4(100)	lb	lh	lwl	lw	lbu	lhu	lwr	
5(101)	sb	sh	swl	sw			swr	
6(110)	lwc0	lwc1						
7(111)	swc0	swc1						

83

op(31:26) = 010001 (FIPs), rs(16:16) = 0 => f = s, rs(16:16) = 1 => f = d, rs(25:21):

23–21 25–24	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(00)	mfc1		cfc1		mtc1		ctc1	
1(01)	bc1..							
2(10)	f = single	f = double						
3(11)								

op(31:26) = 010001 (FIPs), (f above: 10000 => f = s, 10001 => f = d), funct(5:0):

2–0 5–3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	add.f	sub.f	mul.f	div.f		abs.f	mov.f	neg.f
1(001)								
2(010)								
3(011)								
4(100)	cvt.s.f	cvt.d.f			cvt.w.f			
5(101)								
6(110)	c.f.f	c.un.f	c.eq.f	c.ueq.f	c.olt.f	c.ult.f	c.ole.f	c.ule.f
7(111)	c.sf.f	c.ngle.f	c.seq.f	c.ngl.f	c.lt.f	c.nge.f	c.le.f	c.ngt.f

84

Floating-Point C Programının MIPS Assembly kodunda derlenmesi

Örnek; Fahrenheit (F°)'ı Celsius(C°)'a çevirelim:

float f2c (float fahr)
{return((5.0/9.0) * (fahr- 3 2.0));}

Sıcaklığın(Fahr) $f12'de, 5.0, 9.0 ve 32.0 sayılarının floating point değerlerin hafızada tutulan sabitler (sabitlere erişim pointer $gp) olduğunu varsayalım. Elde edilen sonucun(C°) $ f0 registerına yazılması isteniliyorsa, MIPS assembly kodu:

85

f2c:

Hafızadan sabitlerin alınması

lwcl Sfl6.const5(tgp) # $f 16 - 5.0 (5.0 In memory)

lwcl Sfl8.const9($gp> # $f19 - 9.0 (9.0 in memory)

Fraction kısmını oluşturan 5.0/9.0

div.s $f16. $f16. If18 # $fl6 = 5.0 / 9.0

32.0 sabitinin hafızadan alınıp, sıcaklıktan (fahr =$f12) çıkarılması:

lwcl If18. const32(Sgp) *9* Sfl8 - 32.0

sub.s $fl8. $f 12. If 18 y/ If 18 = fahr - 32.0

Elde edilen 2 sonucun çarpılarak $f0 register'ına gönderilmesi

mul.s$fO, $f16, $f18 # $f0 = (5/9>*<fahr - 32.0)

jr $ra # return

86

C type	Java type	Data transfers	Operations
int	int	lw, sw, lui	addu, addiu, subu, mult, div, and, andi, or, ori, nor, slt, slti
unsigned int	—	lw, sw, lui	addu, addiu, subu, multu, divu, and, andi, or, ori, nor, sltu, sltiu
char	—	lb, sb, lui	addu, addiu, subu, multu, divu, and, andi, or, ori, nor, sltu, sltiu
—	char	lh, sh, lui	addu, addiu, subu, multu, divu, and, andi, or, ori, nor, sltu, sltiu
float	float	lwcl, swcl	add.s, sub.s, mult.s, div.s, c.eq.s, c.lt.s, c.le.s
double	double	l.d, s.d	add.d, sub.d, mult.d, div.d, c.eq.d, c.lt.d, c.le.d

87

Encodings of $\pm 2^{k+1-N} n$ into Binary Fields :

Number Type	Sign Bit	K+1 bit Exponent	Nth bit	N-1 bits of Significand
NaNs:	?	binary 111...111	1	binary 1xxx...xxx
SNaNs:	?	binary 111...111	1	nonzero binary 0xxx...xxx
Infinities:	±	binary 111...111	1	0
Normals:	±	$k-1+2^K$	1	nonnegative $n - 2^{N-1} < 2^{N-1}$
Subnormals:	±	0	0	positive $n < 2^{N-1}$
Zeros:	±	0	0	0

88

Performans Değerlendirmesi ve Anlaşılması

- Amaç aşağıda belirtilen sorulara cevap bulmaktır?
- Performansı ölçme, raporlama ve özetleme ?
- Performansı hesaplamaka için ana parametrelerin belirlenmesi..

- *Niçin bazı donanımlar farklı programlar için diğerlerinden daha iyidir?*
- *Sistem performansının hangi parametreleri donanımla alakalıdır.?*
- *Komut seti, performansı nasıl etkiler?*

- 747, Concord ile karşılaştırıldığında ne kadar hızlıdır?
- 747, Douglas DC-8 den ne kadar büyüktür?
- Tabloya bakarak performans için birçok faktör belirlenebilir.
- 1 yolcu taşıma zamanı
- N yolcu taşıma zamanı (Verim -througput)

Airplane Model	Passengers Capacity	Range (miles)	Speed (m.p.h.)	Throughput (passengers x m.p.h.)
Boeing 737-100	101	630	598	60,398
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Concorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,424

- Response time (Cevap Zamanı), Latency (Gecikme) veya Execution (Elapsed) time (Yürütme zamanı): Bir işi tamamlamak için harcanması gerekli toplam zaman (Hafıza erişim zamanı, CPU time, I/O , v.b. gibi donanımlarla alakalıdır).

İşimi çalıştırmak ne kadar sürer?

Bir işi yürütmek ne kadar sürer?

Veri tabanını sorgulamak için ne kadar beklemeliyim?

Gibi sorular bu terimlerle ilgilidir.

- Throughput (Veri - çıktı): Verilen bir zamanda yapılan toplam iş miktarıdır.

Bir seferde kaç tane iş makinada çalıştırılabilir?
Ortalama yürütme oranı nedir?
Ne kadar iş yapıldı?

Gibi sorular bu terimle ilgilidir

.

- Eğer yeni bir işlemci ile makinayı upgrade yaparsak neyi arttırmış oluruz?
- Eğer laboratuvara yeni bir makine eklersek neyi arttırmış oluruz.

- Yürütme (Execution) veya Geçen(Elapsed) zaman: Bir işte kullanılan herşey için harcanan zamanın toplamı.

- Sık sık karşılaştırma amaçları için uygun değildir.

- CPU time: Giriş çıkış sayısı veya başka programların harcadığı zaman değildir.
- CPU'nu bir program için hesaplamaya harcadığı zamandır. Bu terim bekleme sürelerini veya giri,ş/çıkış işlemleri için harcanan zamanı içermez.
- Kullanıcı CPU zamanı: Bir program için CPU'nun harcadığı zamandır.
- Sistem CPU zamanı: Bir programın çalışma sürecinde, işletim sistemi için harcadığı zamanadır.
- Biz kullanıcı(user) CPU time' a odaklanacağız.

Verim ve response time ile ilgili

- X ve Y makinalarında çalışan programlar için verim,

$$\text{Performance}_X = 1 / \text{Execution time}_X$$

$$\text{Performance}_x > \text{Performance}_y$$
$$1 / \text{Execution time}_x > 1 / \text{Execution time}_y$$
$$\text{Execution time}_y > \text{Execution time}_x$$

- X, Y den n kez daha hızlıdır.

$$\text{Performance}_X / \text{Performance}_Y = n$$
$$\text{Execution time}_y / \text{Execution time}_x = n$$

Problem: A makinası bir programı 20 sn'de çalıştırıyor. Aynı programı B 25 saniyede çalıştırıyor. A, B'den ne kadar hızlıdır.

Çözüm:

$$\text{Performance}_A / \text{Performance}_B = \text{Execution time}_B / \text{Execution time}_A = n$$

Buna göre performans oranı = 25 / 20 = 1.25 = n

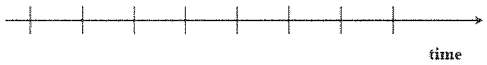
$$\text{Performance}_A = \text{Performance}_B \times 1.25$$

Yürütme zamanı yerine sık sık biz cycle terimini kullanırız. (h/w tasarımcının görüşü)

CPU execution time = CPU clock cycles x clock cycle time

$$Time = \frac{\text{seconds}}{\text{program}} = \frac{\text{clock cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{clock cycle}}$$

- "Ticks" aktiviteler başladığı zamanki clock' u gösterir.



- **cycle time (period) = time between ticks = seconds per cycle**
- **clock rate (frequency) = cycles per second = 1/ cycle time**

- (1 Hz. = 1 cycle/sec), 4 Ghz' 'lik clock'un cycle periyodu

$$\frac{1}{4\times10^9}\text{seconds} = \frac{1}{4\times10^9}\times10^{12} = 250\,\text{picoseconds (ps)}$$

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- Performansı geliştirmek için yukarıdaki bağıntıya göre neler yapılabilir.?

  1- Bir program için gerekli Cycle'ların sayısının azaltılması
  2- Clock cycle time (Periyot) azlatmak
  3- Diğer ifadeyle clock rate (Frekans) arttırmak.

Örnek

- Programımızı A bilgisayarı 10 sn'de çalıştırıyor. A bilgisayarı 4 Ghz lik Clock'a sahiptir. B bilgisayarı programı 6 sn de çalıştırıyor. B bilgisayarı A'nın clock cycles'ları sayısının 1.2 katına ihtiyaç duymaktadır. B ne kadarlık bir clock zamanına (veya frekansına) sahiptir?
- Çözüm:

$$\text{CPU time}_A = \frac{\text{CPU clock cycles}_A}{\text{Clock rate}_A}$$

$$10 \text{ seconds} = \frac{\text{CPU clock cycles}_A}{4 \times 10^9 \frac{\text{cycles}}{\text{second}}}$$

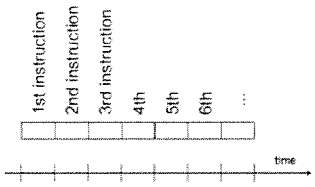
$$\text{CPU clock cycles}_A = 10 \text{ seconds} \times 4 \times 10^9 \frac{\text{cycles}}{\text{second}} = 40 \times 10^9 \text{cycles}$$

$$\text{CPU time}_B = \frac{1.2 \times \text{CPU clock cycles}_A}{\text{Clock rate}_B}$$

$$6 \text{ seconds} = \frac{1.2 \times 40 \times 10^9 \text{cycles}}{\text{Clock rate}_B}$$

$$\text{Clock rate}_B = \frac{1.2 \times 40 \times 10^9 \text{cycles}}{6 \text{ seconds}} = \frac{8 \times 10^9 \text{cycles}}{\text{second}} = 8 \text{ GHz}$$

Komutların sayısını clock'ların sayısına eşit farz edebilirmiyiz?



Bu düşünce yanlıştır.Çünkü farklı komutlar farklı makinalarda farklı zaman alırlar.
Niçin? Makine kodlarını hatırlayın!

- Çarpma toplamadan daha çok zaman alır.
- Floating point sayılarla işlem tamsayılarla işlemlerden daha uzundur.
- Hafızaya erişim registerlere erişimden daha uzun zaman alır.

Ortalama Clock Cycle'ı

Clock cycles per instruction (CPI); Bir program için bir komut başına clock cycle'ının ortalama sayısıdır. Buna göre bir programın harcayacağı clock sayısı:

CPU Clock cycles = Programdaki komutların sayısı x CPI

- Bir programda execution time (yürütme zamanı) doğrudan doğruya komutların sayısına bağımlıdır.
- Buna göre, bir programın yürütme zamanı hesabı için başka bir yolda; yürütülen komutların sayısı ile, her komut için harcanan ortalama zamanın çarpılmasıdır.

CPU execution time = CPU clock cycles x clock cycle time

(CPU yürütme zamanı = CPU clock sayısı x clock periyodu)

Veya;

CPU execution time = Programdaki komut sayısı x CPI x clock periyodu

- Verilen bir program için
  Bazı komutların sayısı (makine komutları)
  Bazı cycle'lar sayıları
  Bazı seconds sayıları gereklidir.
- Bu nicelikler ile ilgili kelimeler;
  cycle zamanı (cycle başına saniye)
  clock rate (saniye başına cycle)
  CPI (komut başına ortalama cycle'lar)
- *Yoğun floating point uygulamaları daha yüksek CPI gerektirir.*

- Performans, yürütme (execution) zamanı tarafından belirlenir.

  Diğer değişkenlerin hangisi performansa eşittir.
- Yürütülen program için Cycle'lerin sayısı?
- Programda komutların sayısı?
- Saniye başına cycle'ların sayısı (Frekans)?
- Komut başına cycle'ların ortalama sayısı?(CPI)
- Saniye başına komutların ortalama sayısı?(MIPS)

Yaygın hata: değişkenlerin birini düşünmek performansı gösterir ancak gerçekçi değildir.

CPI aynı ISA (komut seti)' nin farklı gerçekleştirmelerini karşılaştırmak için iyi bir ölçümdür.

- Farzedin ki aynı komut seti mimarisinin iki farklı gerçekleştirmesine sahibiz.
- A makinasının Clock cycle süresi 250 ps ve aynı program için 2.0 CPI ya sahiptir.
- B makinasının Clock cycle süresi 500 ps ve 1.2 CPI ya sahiptir.
- Bu program için hangi makine daha hızlıdır. Ne kadar hızlıdır?
- Eğer iki makine aynı ISA'ya sahipse (clock rate, CPI, yürütme zamanı, komutların sayısı, MIPS) daima benzer olacaklarmıdır?

Çözüm:

- Veriler göre her iki bilgisayarda, aynı program için benzer sayıda komut işler. Bu sayıya I diyelim.
- İlk olarak her bir bilgisayar için *clock cycle* sayısını bulalım.
  - **CPU clock cycles$_A$ = I x 2.0**
  - **CPU clock cycles$_B$ = I x 1.2**

  Buradan her bir bilgisayar için CPU time'i hesaplayabiliriz.

  CPU time$_A$ = CPU clock cycles$_A$ x Clock cycles time$_A$
  = I x 2.0 x 250 ps = I x 500 ps

  CPU time$_B$ = CPU clock cycles$_B$ x Clock cycles time$_B$
  = I x 1.2 x 500 ps = I x 600 ps

  Anlaşılacağı gibi A bilgisayarı daha hızlıdır.

Performance$_A$ / Performance$_B$= Execution time$_B$ / Execution time$_A$= n
= Ix600ps / Ix500ps = 1.2

CPU time = komut sayısı x CPI x Clock cycle time

Kod Segmentlerinin karşılaştırılması

- Bir derleyici tasarımcısı iki kod dizileri arasında karar vermeyi deniyor. Donanım aşağıdaki mimarileri desteklemektedir.
- 3 farklı komut sınıfı vardır. A, 1 cycle, B, 2 cycle, C 3 cycle ihtiyaç duyuyor.
- İlk kod dizisi 5 komuta,(A=2, B=1,ve C=2)
- İkinci kod dizisi 6 komuta sahiptir. (A=4, B=1 ve C=1.)
- Hangi size göre daha hızlıdır. Ne kadar?
- Herbir dizi için CPI?

	CPI for this instruction class		
	A	B	C
CPI	1	2	3

Code sequence	Instruction counts for instruction class		
	A	B	C
1	2	1	2
2	4	1	1

Çözüm:

- 1.kod dizisi (2 + 1 + 2 = 5 komut)
- 2.kod dizisi (4 + 1 + 1 = 6 komut) kullanır.

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (CPI_i \times C_i),$$

- n komut sınıflarının sayısı, C_i i. Sınıfın komutlarının sayısı ise yukarıdaki formüle göre, her iki komut dizisine göre CPU clok cycles'ı (Clock periyodu sayısını) bulabiliriz.

$$\text{CPU clock cycles}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10 \text{ cycles}$$

$$\text{CPU clock cycles}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9 \text{ cycles}$$

Buna göre 2.kod dizisi daha hızlı olsa bile fazladan 1 komut işleyecektir. Dolayısıyla daha düşük bir CPI'ye sahip olmalıdır.

$$CPI = \frac{\text{CPU clock cycles}}{\text{Instruction count}}$$

$$CPI_1 = \frac{\text{CPU clock cycles}_1}{\text{Instruction count}_1} = \frac{10}{5} = 2$$

$$CPI_2 = \frac{\text{CPU clock cycles}_2}{\text{Instruction count}_2} = \frac{9}{6} = 1.5$$

Örneğe devam:

- Bu örnek gösterir ki; performans değerlendirme için yalnızca bir kıstas (Burada komut sayısı kullanılsaydı) kullanmak yanıltıcı olabilir.
- İki bilgisayarı performans açısından karşılaştırmak için, (yürütme zamanı- execution time) her üç bileşeni (Komut sayısı, CPI, Clock Cycle time (periyot)) dikkate almak gerekir.
- Eğer bileşenlerden bir kısmı özdeş ise, performans karşılaştırması özdeş olmayan bileşenlere göre yapılır. Örneğin her iki bilgisayarın frkansları ayni ise karşılaştırma; komut sayısı ve CPI'ya göre yapılabilir.
- Performas'a ve CPI'ye etki eden en büyük faktör isePİPELİNE mimarisidir ki bu konu daha ileride detaylandırılacaktır.

Performans değerlendirilmesi

- İki veya daha fazla bilgisayar arasındaki performans değerlendirmesini nasıl yaparız.
- Workload (işyükü):A set of programs run on a computer that is either the *actual* collection of applications run by a user or is constructed from real programs to*approximate*such a mix (usually, a set of *benchmark* programs).

- Benchmarks: Specifically chosen to measure performance. Nowadays, they are real applications that the user will use more often (for example, scientific applications for engineers,compiler and text editingfor s/wengineers)

- Performans en iyi gerçek bir uygulamanın çalıştırılmasıyla belirlenir.
- Küçük kriterler

Mimarlar ve tasrımcılar için güzel

Standardize etmek kolay

Kötüye kullanılabilir.

- SPEC (System Performance Evaluation Cooperative)

şirketler gerçek program ve giriş kümeleri hakkında anlaşmaya varmışlardır.

CPU-intensive kriterleri kullanarak sunucular ve iş istasyonları üzerinde duruluyor.

- Şimdilerde CPU performansı, grafik, yüksek performanslı hesaplama, nesneye yönelik hesaplama,Java uygulamaları, client server modelleri, mail sistemleri , Dosya sistemleri ve web serverler. İçerir.

Hala kötüye kullanılabilir.

- *An embarrassed Intel Corp. acknowledged Friday that a bug in a*
- *software program known as a compiler had led the company to*
- *overstate the speed of its microprocessor chips on an industry*
- *benchmark by 10 percent. However, industry analysts said the*
- *coding error…was a sad commentary on a common industry*
- *practice of "cheating"on standardized performance tests…The*
- *error was pointed out to Intel two days ago by a competitor,*
- *Motorola …came in a test known as SPECint92…Intel*
- *acknowledged that it had "optimized"its compiler to improve its*
- *test scores.The company had also said that it did not like the*
- *practice but felt to compelled to make the optimizations because*
- *its competitors were doing the same thing…At the heart of Intel's*
- *problem is the practice of "tuning"compiler programs to recognize*
- *certain computing problems in the test and then substituting*
- *special handwritten pieces of code…*
- *Saturday, January 6, 1996 New York Times*

- Performans Raporu
- Hazırlıktan sonra performans ölçümleri seçilen uygun kriterler kümesinden elde edilir.
- Tekrarlanabilirlik :rapordaki diğer herşey dahil, deneyler rapor edilen sonuçları çoğaltmak gerekir. (O/S version, derleyici, giriş, bilgisayar konfigürasyonu, gibi)
- Girişlerin seçimi çok önemlidir. Büyük girişler bellek sistemini değerlendirmek için gereklidir. Ayrıca büyük iş yükü gereklidir.

- Kriterlerin bir grubunun performansını özetlemek için nasıl karar verildiği kritiktir.
- Özet hesaplama nasıl yapılmalıdır.
- Aşağıdaki örneği göz önüne alınız.

Program	Computer A execution time	Computer B execution time
1	1 sec	10 secs
2	1000 secs	100 secs
Total	1001 secs	110 secs

- Program 1 için A, B'den 10 kat daha hızlıdır.
- Program 2 için B, A'dan 10 kat daha hızlıdır.
- Toplam sonucu nedir?

- Sürekli özet ölçümü
- Böylece bağıl performansı özetlemek için 2 programın toplam yürütme zamanını kullanabiliriz.

$$\frac{\text{Performance}_B}{\text{Performance}_A} = \frac{\text{Execution Time}_A}{\text{Execution Time}_B} = \frac{1001}{110} = 9.1$$

- Böylece B, A'dan 9.1 kat daha hızlıdır.
- Yürütme zamanının ortalaması toplam yürütme zamanına oranıdır. Bu ise aritmetik ortalamadır.

$$\text{AM} = \frac{1}{n} \sum_{i=1}^{n} \text{Execution Time}_i$$

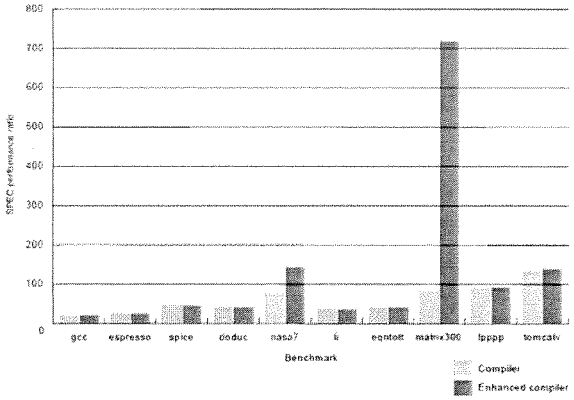
- Weighted AM: iş yükünde programın frekansını yansıtmak için her bir programa ağırlık faktörü atanır.

- Farzedinki 4 farklı desktop seçiyorsunuz. (apple Macintosh, Pentium 4, AMD, Pentium 5)
- En hızlı bilgisayar en yüksek clock rate' sahip olan olacaktır.
- Bütün bilgisayarlar aynı komut setini kullandığı için(intel ISA) komutların sayısı aynıdır. En hızlısı en yüksek clock rate sahip olan olacaktır.
- AMD işlemci farklı CPI'lara sahip olabilir. Ancak biz clock rate bakarak iki pentium işlemcinin daha hızlı olduğunu söyleyebiliriz.
- Benzer işler için kriterlerin sonuçlarına bakarak tam bir performans elde edebiliriz.

- Aşağıdakilerden hangisi doğrudur.
- Program 1 için A B'den daha hızlıdır.
- Program 2 için A B'dan daha hızlıdır.
- Program 1 ve 2'nin yürütmelerinin sayısı eşit iş yükü için A B'den daha hızlıdır.
- Program 2'nin program 1'den 2 kat daha iş yükü için A B'den daha hızlıdır.

Program	Computer A execution time	Computer B execution time
1	2 secs	4 secs
2	5 secs	2 secs

- Performans ve Compiler (Geliştirme)

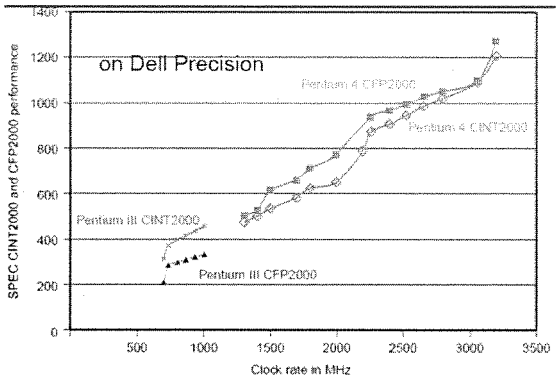


- 12 integer 14 floating point program
- Herbir grup için farklı özetler.
- Clock rapor edilir ancak Çoğunlukla harcanan zaman CPU'dadır. CPU performansı ölçülür.

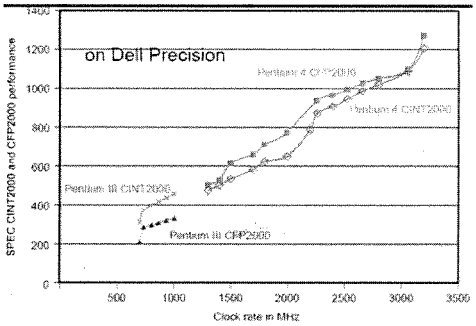
Integer benchmarks		FP benchmarks	
Name	Description	Name	Description
gzip	Compression	wupwise	Quantum chromodynamics
vpr	FPGA circuit placement and routing	swim	Shallow water model
gcc	The GNU C compiler	mgrid	Multigrid solver in 3-D potential field
mcf	Combinatorial optimization	applu	Parabolic/elliptic partial differential equation
crafty	Chess program	mesa	Three-dimensional graphics library
parser	Word processing program	galgel	Computational fluid dynamics
eon	Computer visualization	art	Image recognition using neural networks
perlbmk	Perl application	equake	Seismic wave propagation simulation
gap	Group theory, interpreter	facerec	Image recognition of faces
vortex	Object-oriented database	ammp	Computational chemistry
bzip2	Compression	lucas	Primality testing
twolf	Place and route simulator	fmad5	Crash simulation using finite-element method
		sixtrack	High-energy nuclear physics accelerator design
		apsi	Meteorology: pollutant distribution

FIGURE 4.5 The SPEC CPU2000 benchmarks. The 12 integer benchmarks in the left half of the table are written in C and C++, while the floating-point benchmarks in the right half are written in Fortran (77 or 90) and C. For more information on SPEC and on the SPEC benchmarks, see www.spec.org. The SPEC CPU benchmarks use wall clock time as the metric, but because there is little I/O, they measure CPU performance.

- Performans skalası clock rate ile hemen hemen lineerdir.
- Diğer gözlemler?



- Bir clock rate tarafından bölünen CFP2000 ve CINT2000'in ortalama değerlerine bakınız.
- CINT2000/Clock ratePentium III = 0.47Pentium 4 = 0.36
- CFP2000/Clock ratePentium III = 0.34Pentium 4 = 0.39



SPECweb99

- Verime odaklanır.Bir web server bağlantılarının maksimum sayısının ölçümünü destekleyebilir.
- Çoklu işlemciler sık sık kullanılır.
- Web server s/w sistem ölçümlerinin bir bölümüdür.
- SPECweb99 performansı bir çok sistem karakteristiğine bağlıdır. (disk sürücülerinin sayısı, CPU'ların sayısı, ağların sayısı, clock rate)
- Örnek: Pentium III Xeon w/ 7 disk drives, 8 CPUs, 8 networks, at 0.7GHz, Pentium Xeon 4 w/ 5 disk drives, 2 CPUs, 4 networks, at 3.06GHz 'dan çok daha hızlıdır.

- Güç işlemci performansı için anahtar sınırlama oluyor.
- Batarya güçleri yavaş gelişiyor
- İşlemciler etkili işletilebilmek için tasarımlanmalıdır. Farklı clock rate'ler arasında anahtarlanabilmeli ve güç korunabilmelidir.
- 3 işletim modu varsayılır.

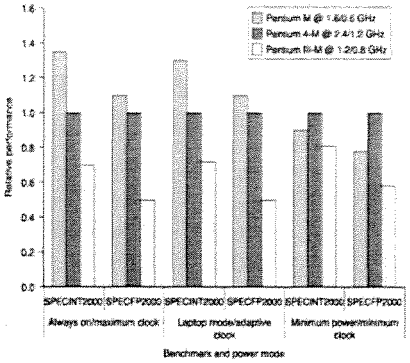
Maximum güç: maximum clock rate→ en iyi performans

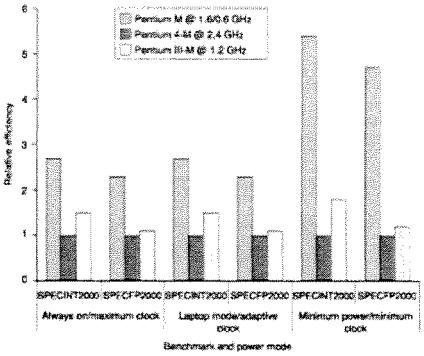
Laptop mode: adaptif clock rate

Minimum güç: minimum clock rate → en iyi güç verimliliği

- Enerji verimliliği:kriterler çalışıyorken performans ortalama güç tüketimi tarafından bölünür

- Hangi makinedaha düşük clock rate ile daha yüksek performansa sahiptir





- Performans özel programlara özgüdür.
- Toplam yürütme zamanı performansın tutarlı bir özetidir.
- Mimari performansı arttırmak için

Clock rate'yi arttırmak

İşlemci organizasyonunu geliştirmek,düşük CPI

Compiler geliştirme,düşük CPI ve düşük komut sayısı

Komut sayısını etkileyecek algoritma/dil seçmek

- Birçok tuzak,

Makine performansının bir yönünü geliştirmeyi ummak toplam performansı etkiler. (Amdahl kanunu)

Performans metriği olarak bir alt performans denklemi kullanılarak (MIPS ölçümleri)

$$\text{Execution Time After Improvement} = \text{Execution Time Unaffected} + \frac{\text{Execution Time Affected}}{\text{Amount of Improvement}}$$

- Örnek; bir makinada, bir program 100 saniyede çalıştığını farzedelim. Bu zamanın 80 sn'si çarpma içindir. Biz programı 4 kat daha hızlı çalıştırmak istersek çarpma hızını nekadar çok geliştirmeliyiz.
- Programı, 5 kat daha hızlı nasıl yapabiliriz.?

- MIPS = Million Instructions Per Second
- Anlamak kolaydır,belli varsayımlar altında performans belirlemekte kullanılır.
- "whole picture" vermediği için Çoğu kez yanlış olabilir.
- Yürütme oranı tanımlar ancak komut yetenekleri görünmez
- Farklı ISA'lar ile bilgisayar karşılaştırmada kullanılamaz. Niçin?

- İki farklı compiler 4 Ghz lik bir makinada 3 farklı komut sınıfı için test ediliyor. Class A, Class B, ve C Class, sırasıyla 1, 2, ve 3 cycle gereklidir.
- Her iki compiler yazılımın büyük bir parçası için kod üretmede kullanılır.
- İlk compilerin kodu 5 million Class A komut, 1 million Class B komut, and 1 million Class C komut kullanıyor.
- İkinci compiler kodu 10 million Class A komut,1 million Class B komut, ve 1 million Class C komut kullanıyor.
- Hangi dizi MIPS'e göre daha hızlıdır.
- Yürütme zamanına göre hangi dizi daha hızlıdır.

Bölüm: 5
İşlemci: Datapath ve Kontrol

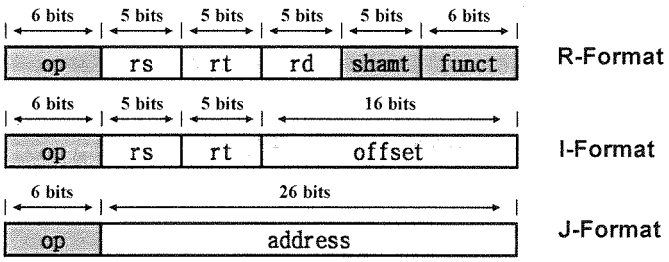
Slide Kaynağı: Patterson & Hennessy COD book website (copyright Morgan Kaufmann) adapted and supplemented

Önemli bir problemin çözümünde, hiçbir detay önemsiz değildir - **French Proverb**

Adam haklı - **Hayrettin Can**

MIPS'in gerçekleştirilmesi

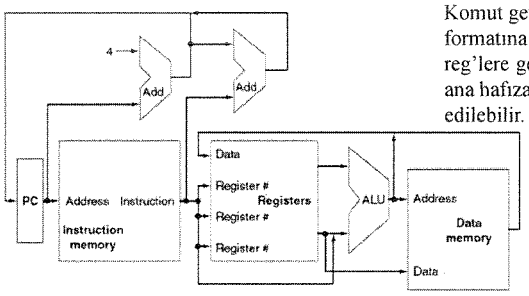
- Biz şimdi bir MIPS komut setinin gerçekleştirilmesini incelemeye hazırız.
- Komut yapılarını ana hatlarıyla hatırlayalım.
  - Aritmetik-Lojik komutlar: `add, sub, and, or, slt`
  - Hafıza-tabanlı komutlar: `lw, sw`
  - Kontrol-Akış komutları: `beq, j`



MIPS'in gerçekleştirilmesi: Fetch/Execute Çevrimi

- Fetch/execute gerçekleştirme işlemine yüksek seviye ayrıştırmayla bakarsak:
  - Komut adresini okumak için program Counter (PC) kullanılır.
  - Komutun hafızadan alınması ve PC'nin arttırılması.
  - Okunacak registerleri seçmek için için komut yapısındaki alanları kullan. Örneğin transfer komutu (Load, store) sadece 1 registerin okunmasına gerek duyar fakat diğer komutlar 2 reg'in okunmasına gerek duyar.
  - Komut sınıfına bağlı yürütme *(execute)* işlemi.
  - Tekrar.....

Dikkat ediniz ki çoğu komutlar doğrudan icra edilir ve bir sonraki komutun adresi PC+4 ile yazılır. Fakat dallanma komutların da ise ALU dallanılacak yeni komut adresini hesap edebilir. Veya şartsız dallanma komutlarında ise PC+4 ikinci bir toplama ile dallanılacak komutun adresi elde edilir.



Komut getirildikten sonra, formatına göre operandlar ilgili reg'lere getirilip işlenir. Veya ana hafıza ya load/store edilebilir.

İşlemcilerde gerçekleştirme stilleri

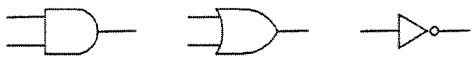
- Tek Cycle
  - Herbir komut 1 clock cycle'da icra edilir.
  - Clock en yavaş komutlar için yeterince uzun olmalıdır.; Bu yüzden,
  - Dez avantaj: sistem en yavaş komut kadar hızlıdır.
- Çoklu-Cycle
  - Çoklu-cycle adımlarında fetch (Alma)/execute(Yürütme) yapılır
  - Her bir clock cycle'da 1 adım icra edilir.
  - Avantajı: Her bir komut ihtiyacı kadar birkaç cycle kullanır.
- Pipeline
  - Herbir komut çoklu adımlarda yürütülür.
  - Her bir clock cycle'da 1 adım/komut icra edilir.
  - Paralel olarak çoklu komutlar işlenir – assembly line

Functional Elemanlar

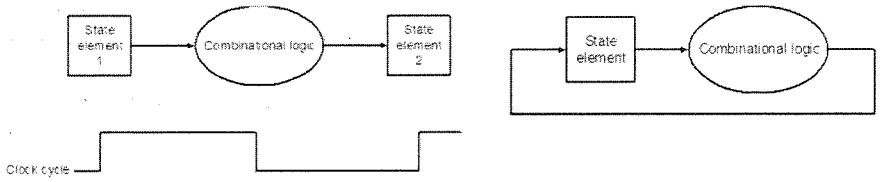
- Donanımda 2 tip fonksiyonel eleman vardır.
  - Veri üzerinde işlem yapan elemanları : (*Kombinasyonal elemanlar olarak adlandırılır – Toplayıcılar v.b.*)
  - Veri içeren elemanlar: (*state* veya *ardışıl elemanlar olarak adlandırılır. Hafıza elemanları v.b*)

Kombinasyonal Elemanlar

- giriş ⇒ *çıkış fonksiyonları gibi çalışır* . örneğin; ALU
- Kombinasyonal lojik, bir registerden okur veya bir registere veya okuduğu registere yazar.
  - *read/write bir cycle'da meydana gelir.* – kombinasyonal eleman gelecekte kullanabilmek için veri depolayamaz



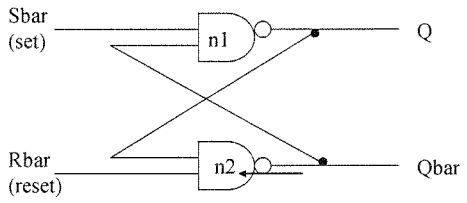
`Combinational logic` donanım birimleri



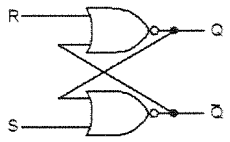
State (Durum) Elemanlar

- State elemanlar dahili hafızaları içerir. Ör: *registers* ve hafıza.
- Bütün state elemanlar makinanın durumunu tanımlar
  - *Bunun anlamı nedir?* Tekrar açmayı ve kapatmayı düşün.
- *Flipflops* ve *latches* are 1 bitlik state elemanlardır, Eşdeğeri 1 bitlik hafızalardır.
- Flip-flop veya Latch'in çıkış(ları) daima saklanan bit değerine bağlıdır. Onların durumu 0/1 veya yüksek/düşük veya doğru/yanlış olarak adlandırılır.
- Bir flip-flop veya latch'in girişi onun durumuna bağlı olarak clock olup olmadığına göre değişebilir.

Set-Reset (SR-) latch (unclocked)

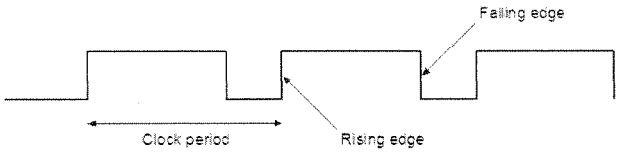


NOR latch



Senkronize Lojik Logic: Clock girişli FF'lar veya LATCH'ler

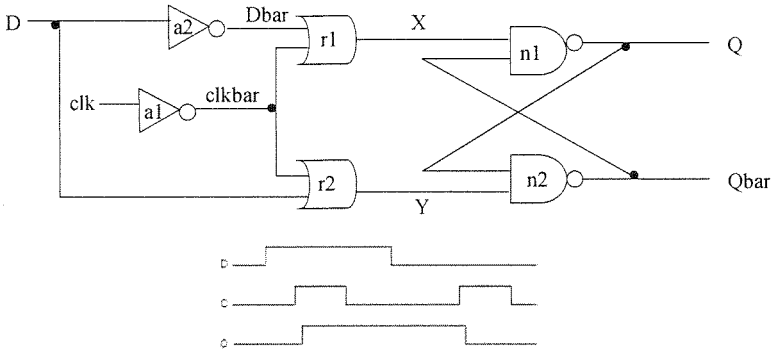
- Clock işareti; bir state elemanını (FF'lar barındıran) senkronlamak veya ona yaz/oku yapmayı denetlemek için kullanılır.
  - Genellikle kenar tetiklemeli (Yükselen/Düşen) state elemanları kullanılır.



- Latch'ler seviye tetiklemelidir.
- Flip-flops are kenar teteiklemelidir.

Clock girişli D-latch

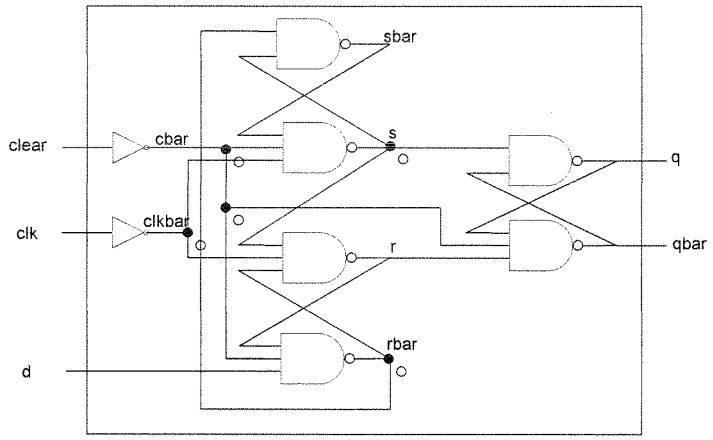
- Durum (Q) sadece clk'nın 1 olduğu zamanda değişebilir.
- Tek bir data (D) girişi vardır. (SR-latch ile karşılaştırıldığında)
- *Tanımlanmamamış davranışı yoktur.*



D-LATCH'in zamanlama diyagramı

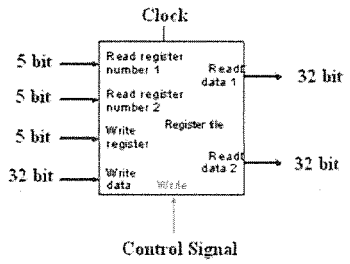
Clock girişli D-flipflop

- *Negatif kenar tetiklemeli FF'dır.*
- 3 tane S-R Mandal'la yapılır.



Data path'de durum(State) elemanları: Register File

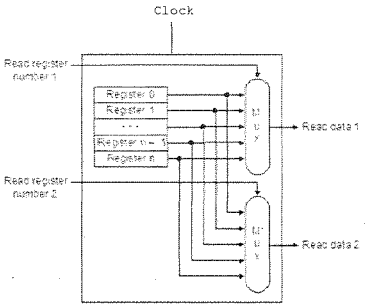
- Registerler D-flipflop'ların dizisi ile gerçekleştirilir.



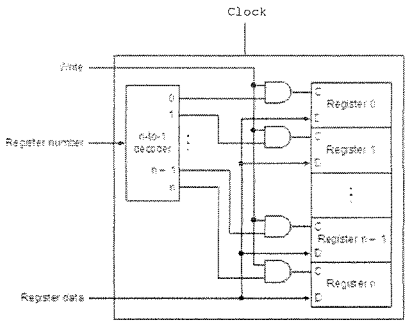
Register file iki okuma portu ve 1 yazma portuna sahiptir.

DATA PATH'de durum(state) elemanları: Register File

- Port gerçekleştirme:



32 register için oku portları 5 bit mux'ların çifti ile gerçekleştirilir.



Yazma portu bir decoder kullanılarak gerçekleştirilir. 32 register için 5'e 32'lik decoderdir. Yazma işlemi Clock ile ilişkilidir. (Yükselen veya düşen kenar)

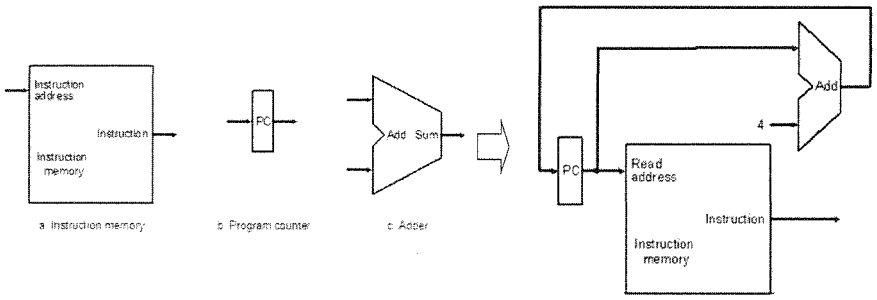
Verilog - VHDL

- Donanım tanımlama dili, Bütün donanımlar Verilog veya VHDL donanım tanımlama dili kullanılarak gerçekleştirilebilir.
- Detaylı bilgi ve örnekler için Verilog, VHDL slide'larına bakınız

MIPS'in tek Cycle'da gerçekleştirme

- MIPS komut seti mimarisinin ilk gerçekleştirilmesinde her bir komut için yeterince uzun tek bir clock cycle kullanacağız.
- Herbir komut yükselen veya düşen clock kenarı ile başlar ve yükselen veya düşen kenar ile sonlanır.
- Bu yaklaşım pratik değildir. Multicycle işlemeden daha yavaştır.Burada farklı komut sınıfları farklı cycle zamanları alabilir.
  - Herbir komut bir tek cycle gerçekleştirmede en yavaş komut kadar zaman alır.
  - multicycle gerçekleştirmede bir kaç cycle kullanılarak bu problemden kaçınılır.
- Tek cycle yaklaşımı pratik olmamasına rağmen konuyu anlamak için yararlı ve önemlidir.
- *Note* : we shall implement `jump` at the very end

Veri yolu: komut Store/Fetch & PC arttırma

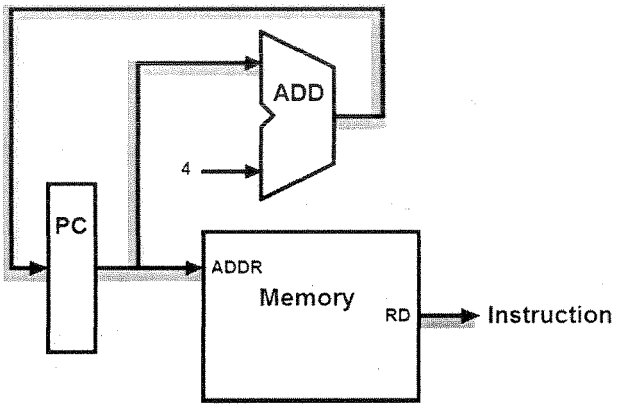


Fetch ve depolama ve PC' yi arttırmak için 3 elaman kullanılır

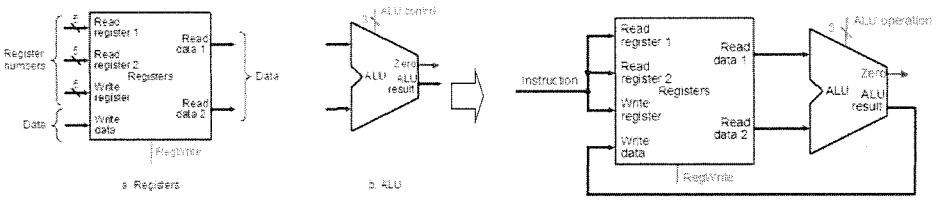
DATAPATH

Datapath Animasyonu

Instruction <- MEM[PC]
PC <- PC + 4



Datapath: R-Tipi Komut



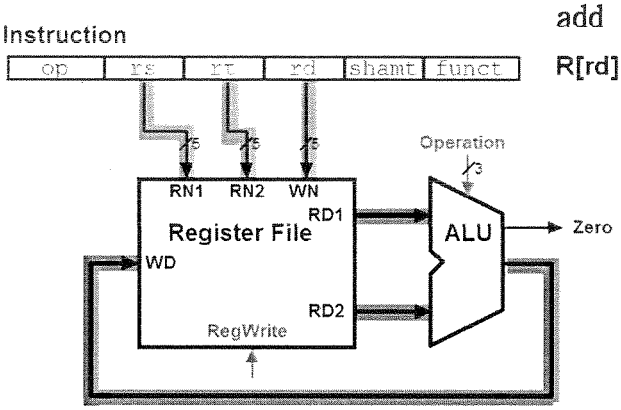
R-type komutları gerçekleştirmek için 2 eleman kullanıldı

Datapath

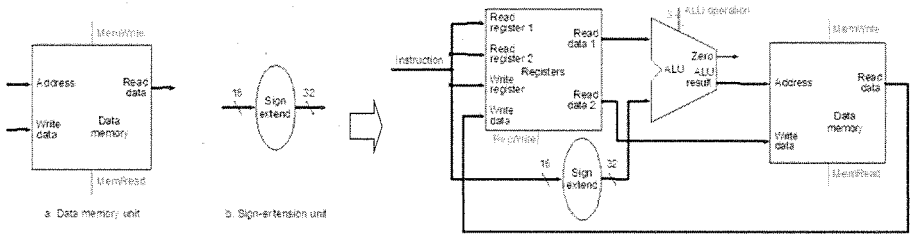
Datapath Animasyonu ...

add rd, rs, rt

R[rd] <- R[rs] + R[rt];



Datapath: Load/Store Komutu



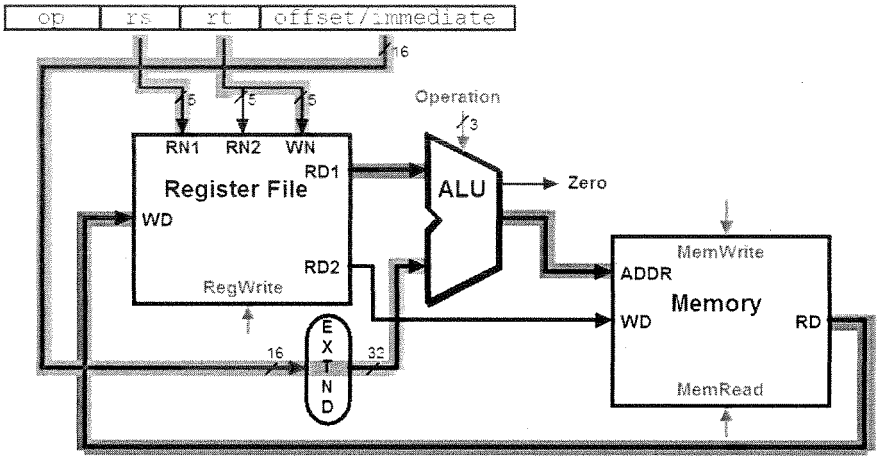
load/store komutunu gerçekleştirmek için İki ek eleman kullanıldı

Datapath

Datapath Animasyonu.....

lw rt, offset(rs)

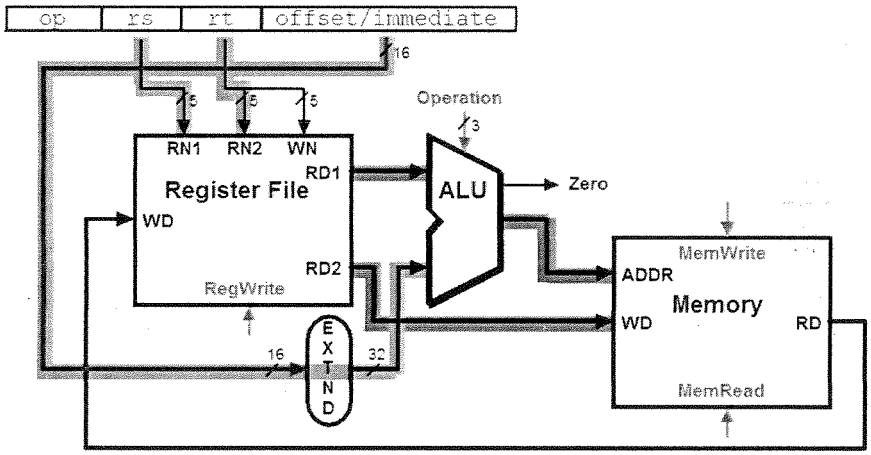
R[rt] <- MEM[R[rs] + s_extend(offset)];



Datapath Animasyonu.....

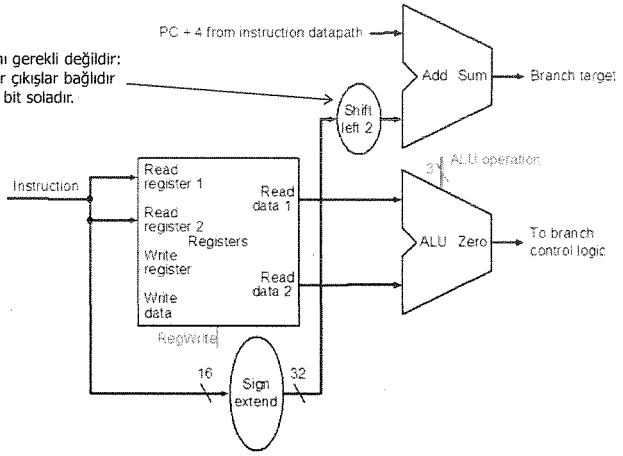
sw rt, offset(rs)

MEM[R[rs] + sign_extend(offset)] <- R[rt]



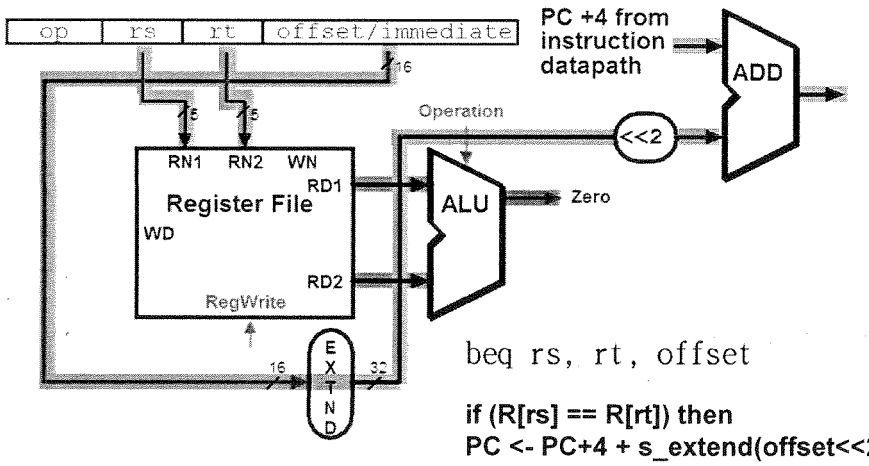
Datapath: Branch Instruction (Dallanma komutları)

Kaydırma donanımı gerekli değildir:
Basit olarak girişler çıkışlar bağlıdır
Herbir kaydırma 2 bit soladır.



Datapath

Animating the Datapath

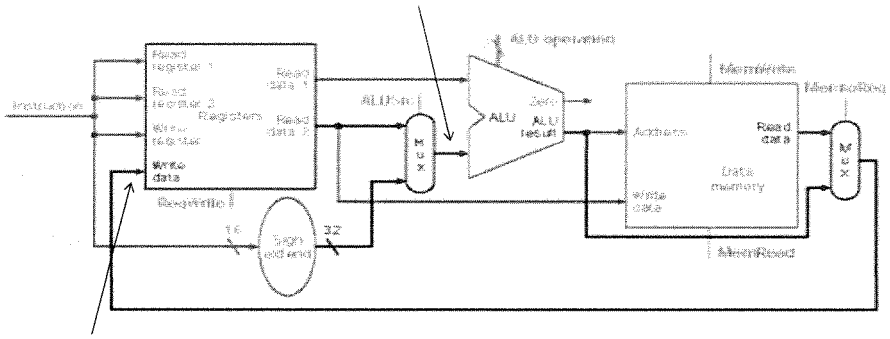


beq rs, rt, offset

if (R[rs] == R[rt]) then
PC <- PC+4 + s_extend(offset<<2)

MIPS Veriyolu I: Single-Cycle

Giriş ya register (R-type) yada komutun alt yarısı

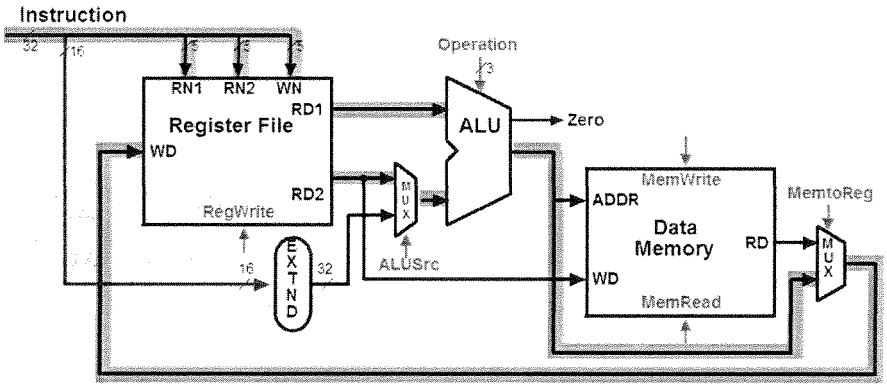


Veri ya ALU'dan (R-type)
Yada hafızadan (load)

R-tip komutlar ve load/store için veriyolu oluşturmak için 2 mux kullanılır

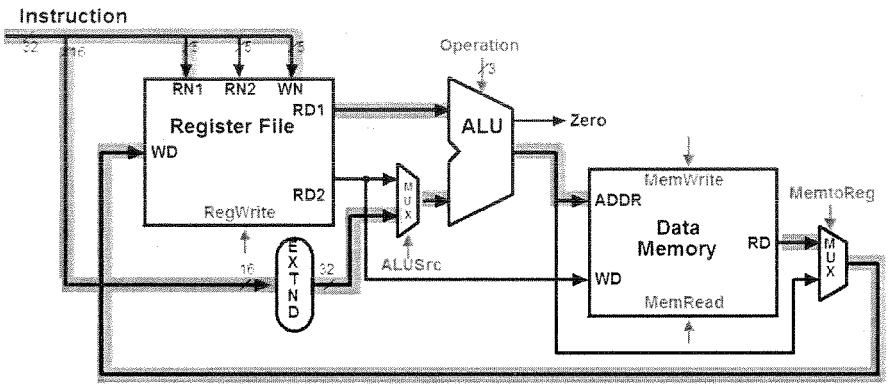
R-tipi komutun çalışması

`add rd,rs,rt`



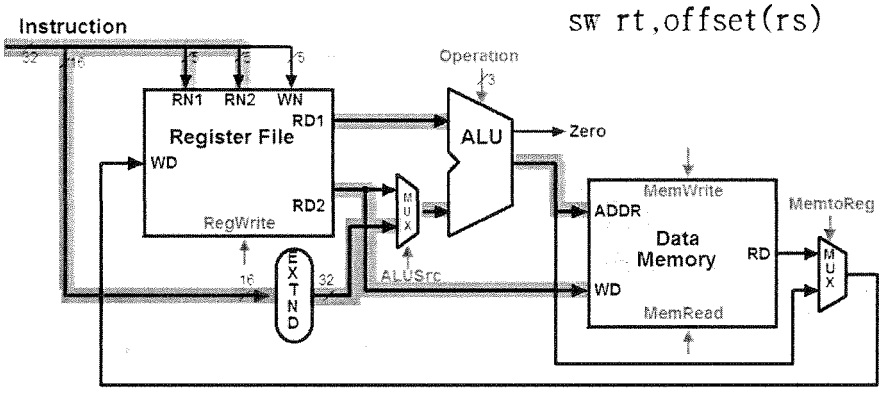
Load Komutunun Çalışması

lw rt,offset(rs)

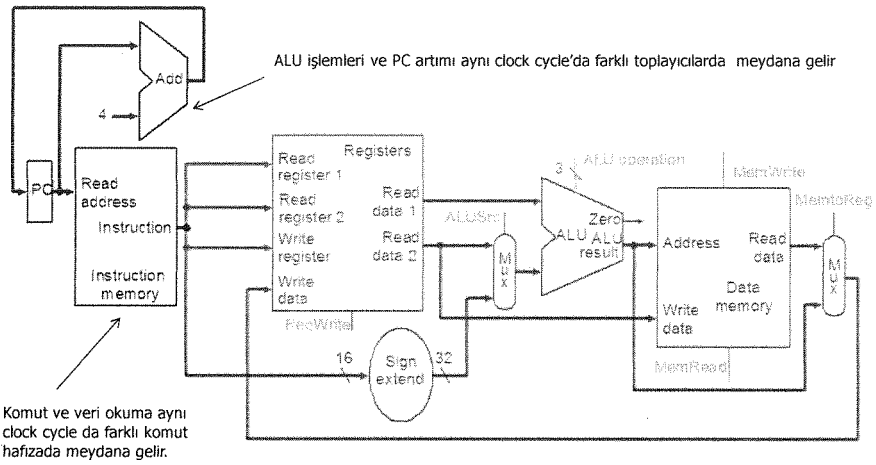


Store komutunun Çalışması

sw rt,offset(rs)

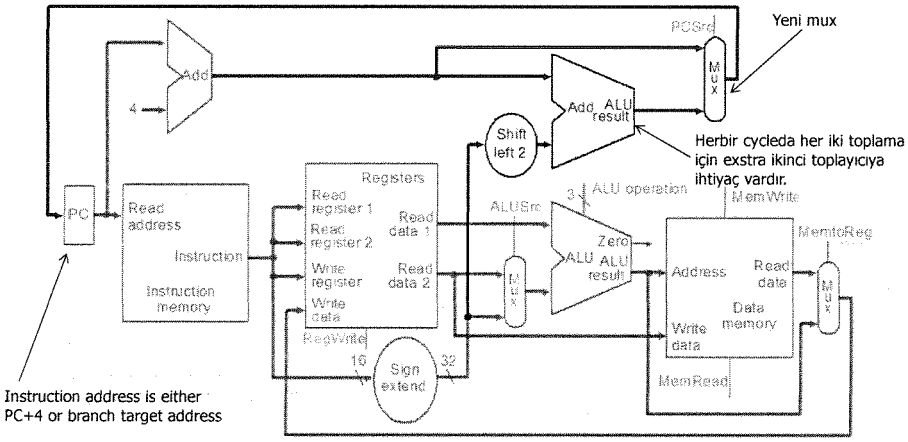


MIPS Veri yolu II: Single-Cycle



komut fetch'i ekleme

MIPS DATA PATH III: Single-Cycle

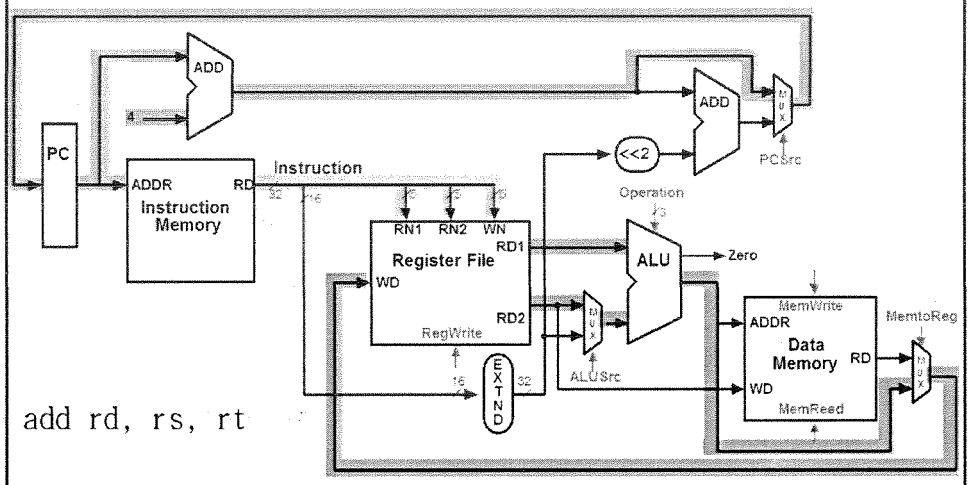


dallanma yeteneğinin ve diğer mux'un eklenmesi

Önemli not: single-cycle gerçekleştirmede veri komut boyunca depolanamaz. Veri yalnızca kombinasyonal lojik yoluyla taşınır

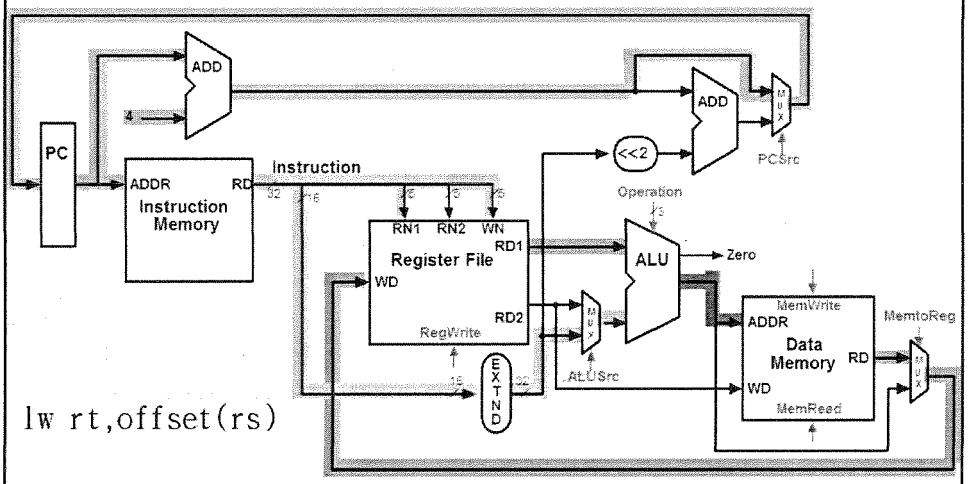
Soru: Memread sinyaline gerçekten ihtiyaç varmıdır. RegWrite' ı düşünün

DATAPATH: Executing `add`



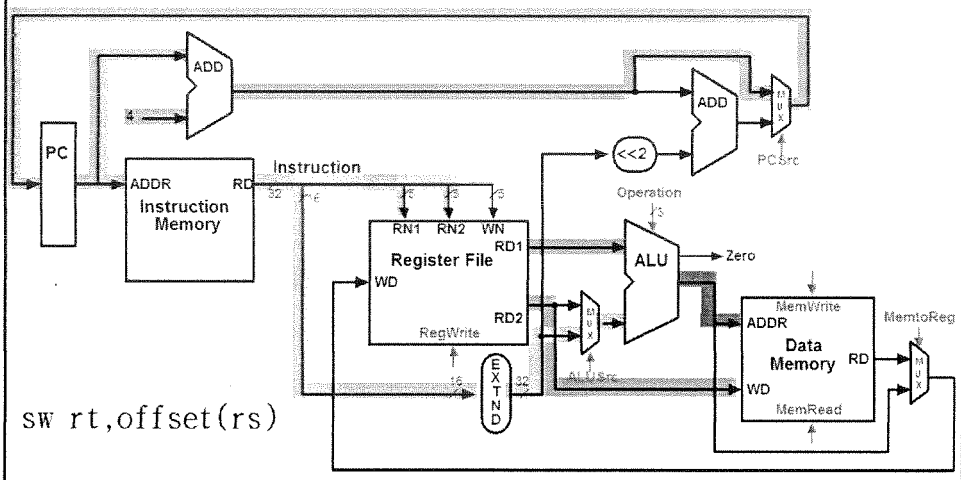
add rd, rs, rt

DATAPATH Executing `lw`



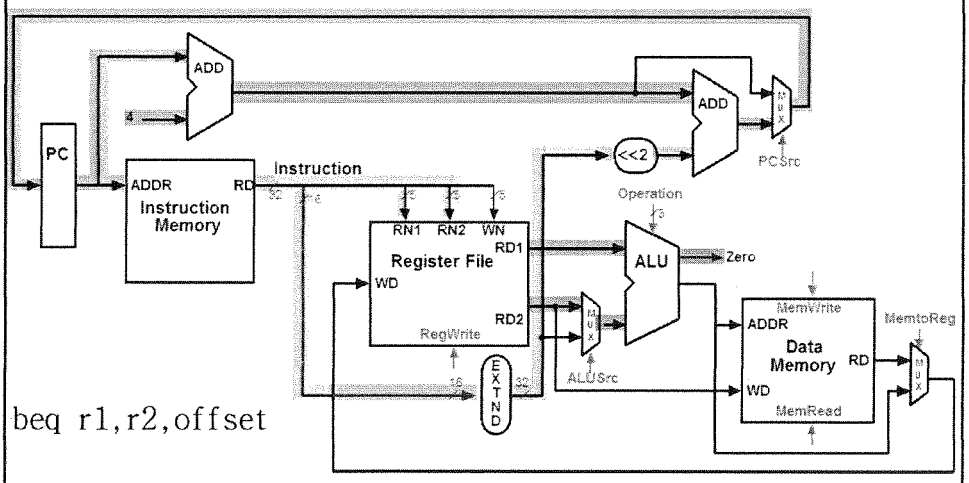
lw rt,offset(rs)

DATAPATH Executing sw



sw rt,offset(rs)

DATAPATH Executing beq



beq r1,r2,offset

DATA PATH, Kontrol

- Kontrol biriminin girişleri;
- Komut kodu formatındaki opcode bitleri (5 bit).

- Control birimi aşağıdaki çıkışları üretir.
  - ALU kontrol girişleri (2 bit input)
  - Yazma enable (olası olarak, okuma enable) her bir depolama elemanları için sinyal.
  - Herbir mux için seçme kontrolü

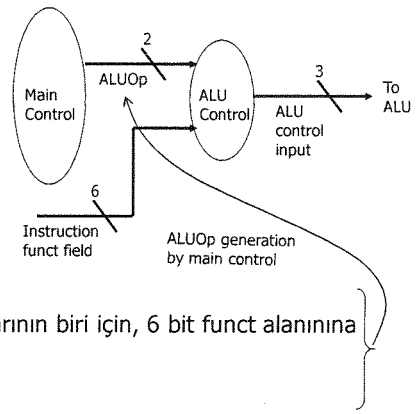
ALU Kontrol

- ALU kontrol planı: Ana kontrol birimi; ALU kontrol birimi girişine iki bit (ALUop kontrol) gönderir. ALU kontrol birimi girişine ayrıca komutun 6 bitlik *funct* bitleri de uygulanır. Çıkışta, ALU kontrolu için 3 bitli oluşur.

Recall from Ch. 4

-

ALU kontrol alanı	Fonksiyon
000	and
001	or
010	add
110	sub
111	slt



- ALU icra etmelidir.
  - load/stores için add (ALUOp 00)
  - *Dallanma için sub* (ALUOp 01)
  - *and, or, add, sub, slt* R-type komutlarının biri için, 6 bit funct alanınına bağlıdır. (ALUOp 10)

ALU kontrol bit ayarları

komut opcode	AluOp	komut işlemi	Funct Alanı	İstenen ALU etkisi	ALU girişi
LW	00	load word	xxxxxx	add	010
SW	00	store word	xxxxxx	add	010
Branch eq	01	branch eq	xxxxxx	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	set on less	101010	set on less	111

*, eğer X olursa, ozaman satır 2 ile 3-7 arası çatışma Vardır.

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0*	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

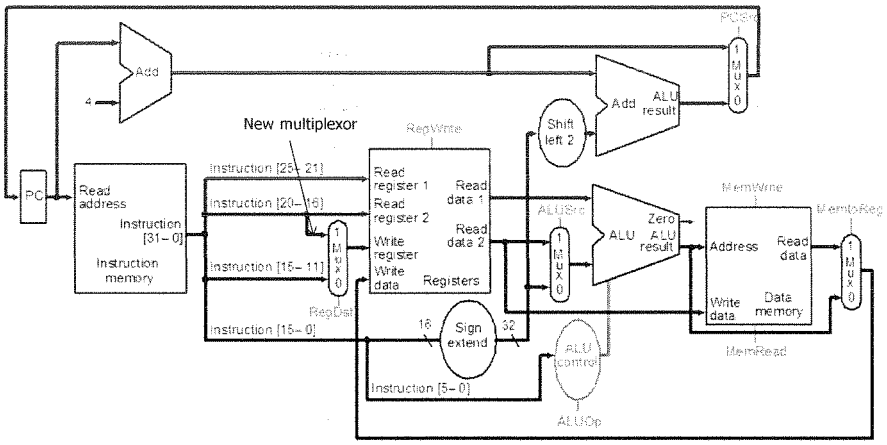
ALU kontrol bitleri için doğruluk tablosu

Ana kontrol Birimi tasarımı

R-type	opcode	rs	rt	rd	shamt	funct
	31-26	25-21	20-16	15-11	10-6	5-0
Load/store or branch	opcode	rs	rt	address		
	31-26	25-21	20-16	15-0		

- MIPS komut formatları hakkında gözlemler
  - opcode daima 31-26 bitlerindedir.
  - Okunabilen 2 register daima rs (bits 25-21) ve rt (bits 20-16) dir.
  - load/stores için base register daima rs (bits 25-21) dir.
  - Dallanma ve load/store için 16-bit offset daima 15-0 bitleridir.
  - Load için hedef register 20-16 (rt) bitlerindedir. R-type komutlarda 15-11 (rd) bitleridir. (bunun için mux ile seçmek gereklidir)

DATA PATH ve kontrol I



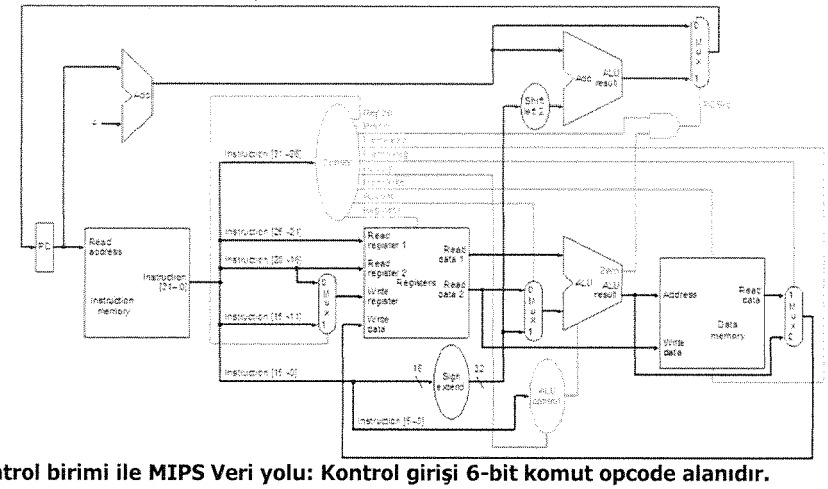
MIPS DATAPATH III'e kontrol ekleme (ve özel hedef register için alan seçmek için yeni bir mux): 9 kontrol sinyallerinin fonksiyonları nedir?

Kontrol sinyalleri

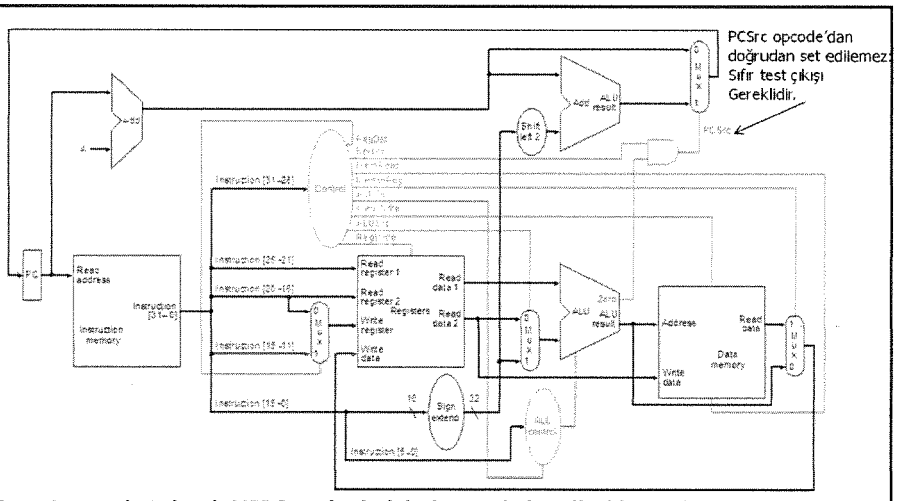
Signal Name	Effect when deasserted (Lojik 0)	Effect when asserted (Lojik 1)
RegDst	yazma registeri için hedef register sayısı rt alanı (bits 20-16)	yazma registeri için hedef register sayısı rd alanı (bits 15-11)
RegWrite	None	Write register girişindeki register Write data input girişindeki değere yazılır.
ALUSrc	ikinci ALU operandı ikinci register file çıkışından gelir(Read data 2)	İkinci ALU operandı is the sign-extended, komutun düşük 16 bitine sign-extended dir
PCSrc	PC toplayıcının çıkışı tarafından yerdeğiştirilir. bu PC + 4 ile hesaplanır	PC Toplatıcının çıkışı tarafından yer değiştirilir. bu hedef dallanmayla hesaplanır.
MemRead	None	Data memory içeriği giriş adresine göre belirlenir bu giriş ilk ilk Read data çıkışına konur
MemWrite	None	Data memory içeriği giriş adresine göre belirlenir bu giriş Write data girişinin değeri tarafından değiştirilir.
MemtoReg	ALU'dan gelen değeri Write data giriş registerine yaz	Data memory'den gelen değeri Write data giriş registerine yaz

Yedi kontrol sinyalinin etkileri

Datapath with Control II



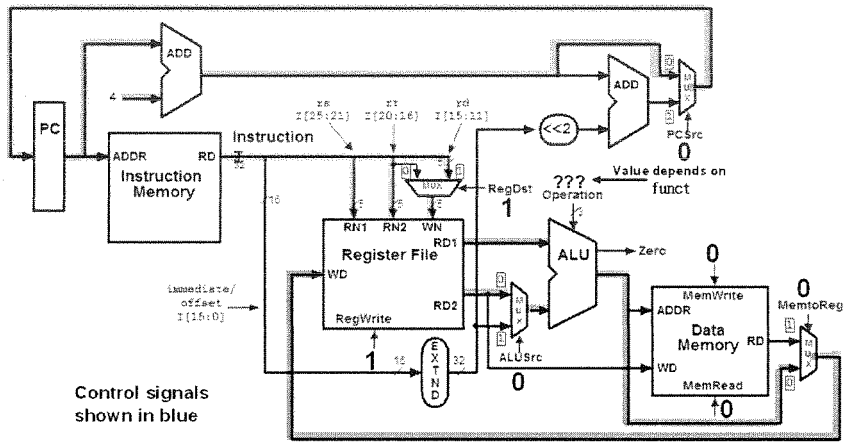
Kontrol birimi ile MIPS Veri yolu: Kontrol girişi 6-bit komut opcode alanıdır. Çıkış 7, 1-bit signal ve 2-bit ALUOp sinyalidir.



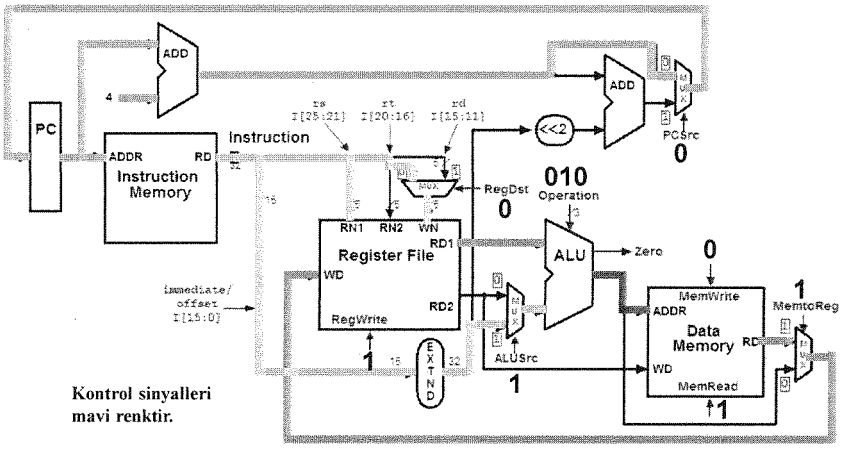
Komut opcode tabanlı MIPS veriyolu için kontrol sinyalleri hesaplama

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

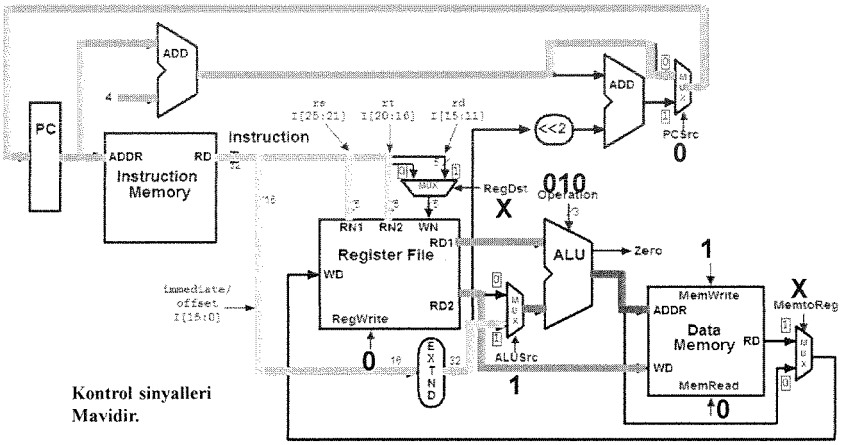
Control Signals: R-Type Instruction



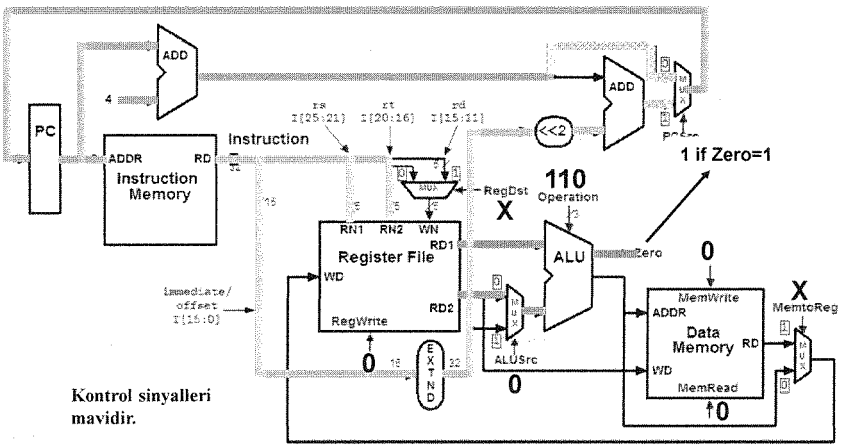
Kontrol sinyalleri: `lw` komutu



Kontrol sinyalleri: `sw` komutu



Kontrol Sinyalleri: `beq` komutu

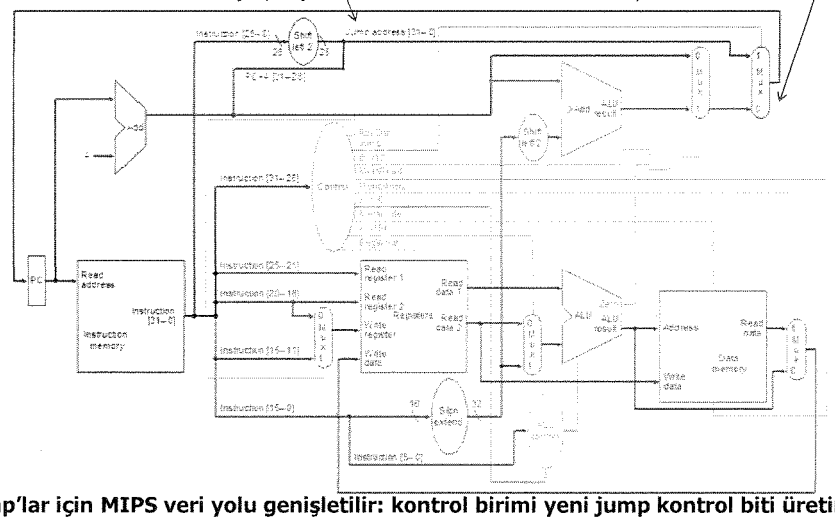


Veri yolu ile kontrol III

Jump	opcode	address
	31-26	25-0

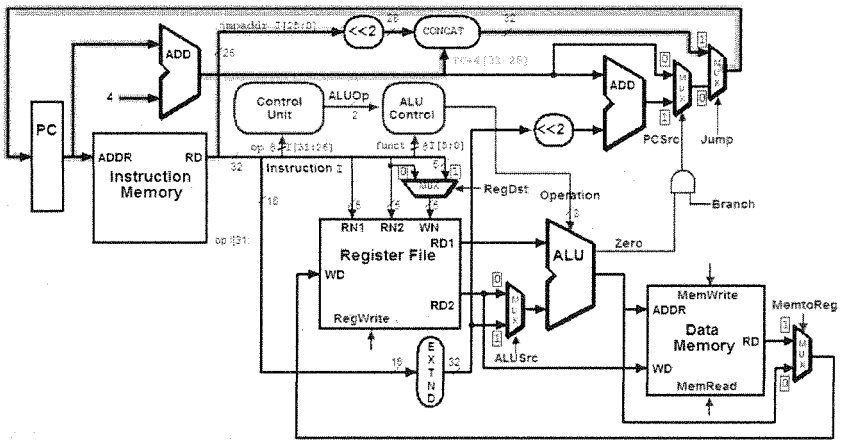
Hedef adrese jump oluşturma

İlave jump kontrol biti ile yeni mux

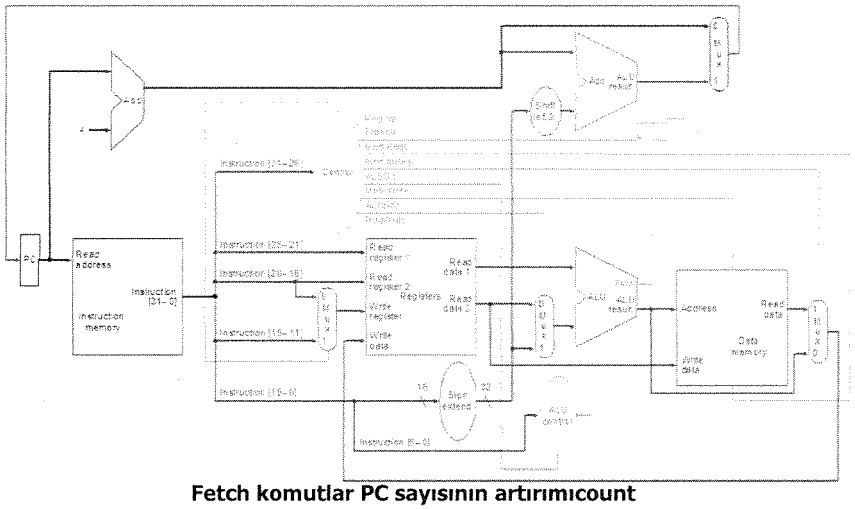


Jump'lar için MIPS veri yolu genişletilir: kontrol birimi yeni jump kontrol biti üretir.

Veri yolu Executing j

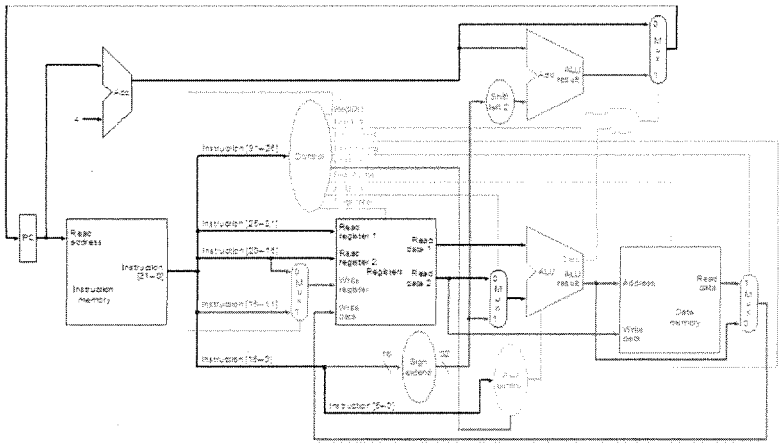


R-type Komutlar: Step 1
add $t1, $t2, $t3 (active = bold)



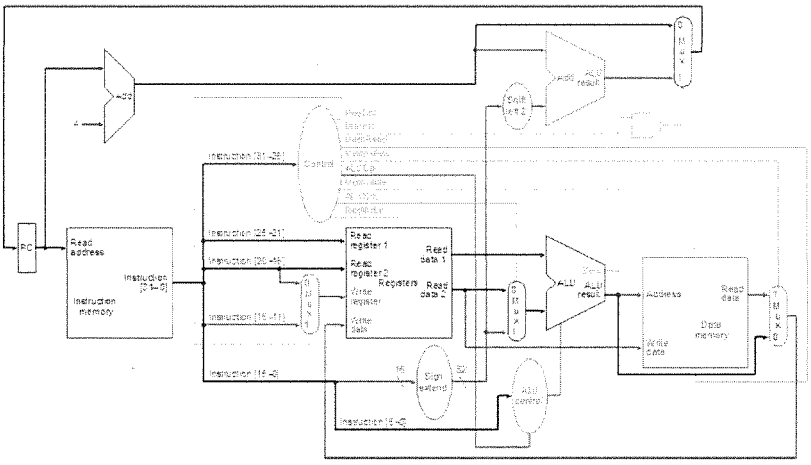
Fetch komutlar PC sayısının artırımıcount

R-type Komut: Step 2
add $t1, $t2, $t3 (active = bold)



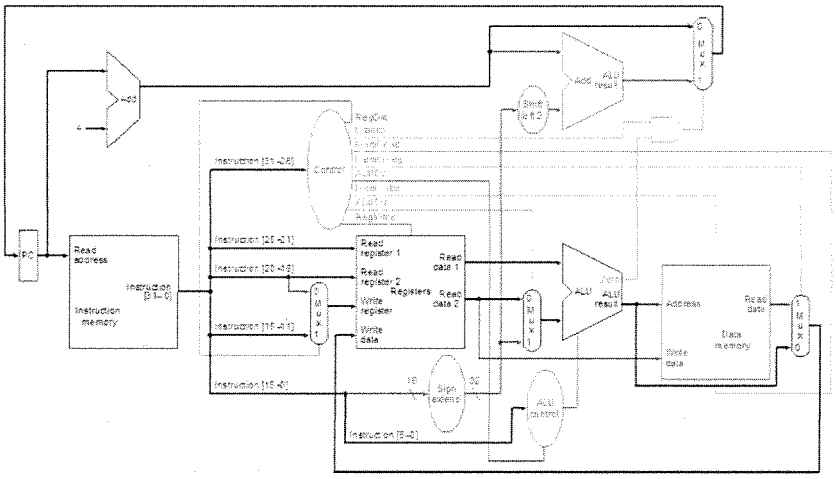
Register file'dan 2 kaynak register oku

R-type komut: Step 3
add $t1, $t2, $t3 (active = bold)



ALU 2 register operandı işler

R-type komut: Step 4
add $t1, $t2, $t3 (active = bold)



Sonuç registere yazılır.

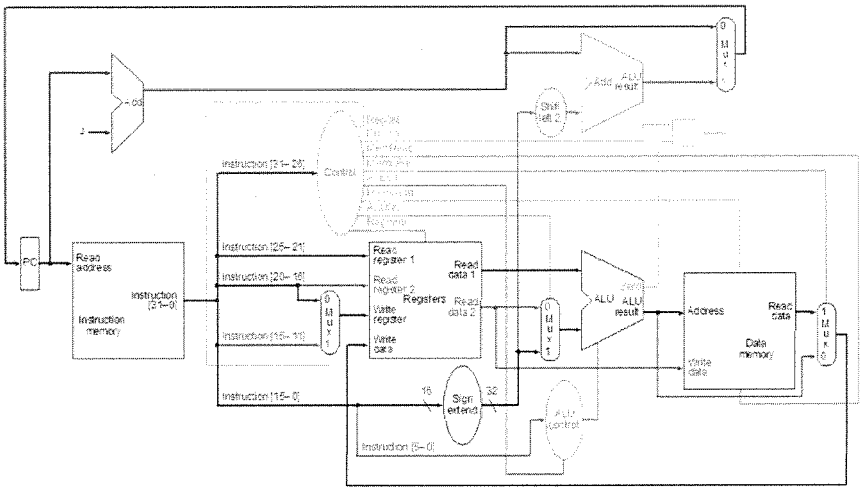
Single-cycle Gerçekleştirme Notları

- *Adımlar gerçekten ayrı değildir. Tamamen bir clock cycle'daherbir komut tamamlanır.* – Basitçe veriyolu boyunca veri akışının dizisi görülür.
- *Bir cycle boyunca veriyolunun işletimi kombinasyoneldir. Hiçbir şey bir clock cycle boyunca depolanmaz*
- Buyüzden, makine bir cycle başlangıcında stabildir. Ve cycle sonunda makine yeni bir stabil duruma geçer.
- *single-cycle işlemeyi anlamak için çok önemli*:
  bizim Verilogdaki SimpleSingleCycleComputer örneklerine bak.

Load komut adımları lw $t1, offset($t2)

1. Komutu Fetch et ve PC' yi arttır.
2. Register file'dan base registeri oku: base register ($t2) komutun 25-21 bitleritle verilir.
3. ALU 16 bitlik lower sign-extended ve register file'dan değerlerin toplamını hesaplar
4. ALU'dan toplam data memory için adres olarak kullanılır.
5. Hafıza biriminden data register file'a yazılır. Hedef register ($t1) komutun 20-16 bitleriyle verilir

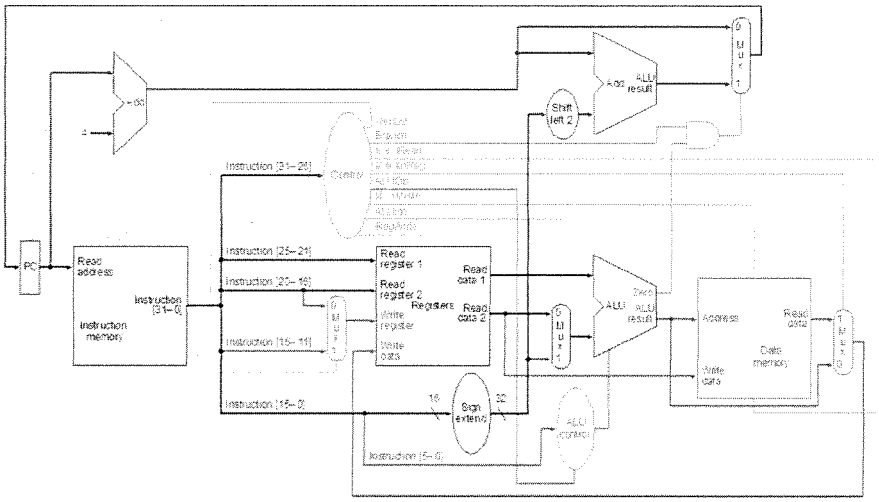
Load komutu
lw $t1, offset($t2)



Branch komut adımları
beq $t1, $t2, offset

1. Komutu Fetch et ve PC'yi arttır.
2. Register file'dan iki registeri ($t1 and $t2) oku
3. ALU register file'dan veri değerlerini çıkararak icra eder; PC+4' ün değeri Hedef dallanma adresini vermek için 2 sola kaydırılan komutun sign-extended lower 16 bit (offset)' eklenir.
4. ALU'dan Zero sonuçları karar vermek için kullanılır PC'de depolamak için sonuçlar toplanır. (1 veya 3. adımdan)

Branch komutu
beq $t1, $t2, offset

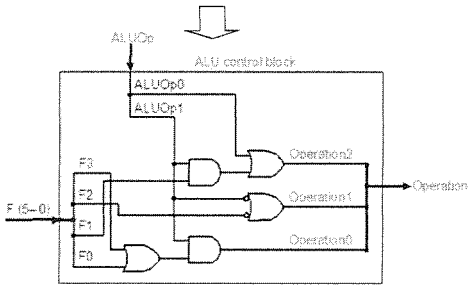


Gerçekleştirme: ALU kontrol Bloğu

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0 *	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

Truth table for ALU control bits

•Eğer X olursa satır-2 ve satır3-7 arası karışıklık olur.

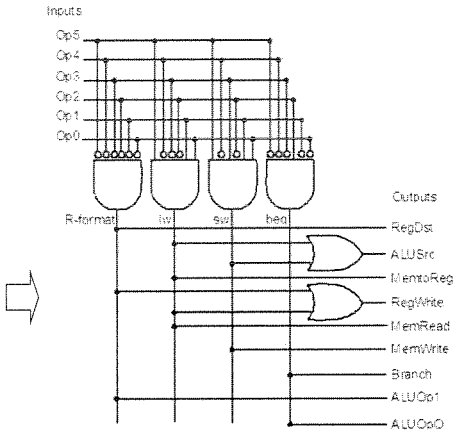


ALU Kontrol logic

Gerçekleştirme: Ana kontrol bloğu

	Signal name	R- Format	lw	sw	beq
Girişler	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Çıkışlar	RegDst	1	0	x	x
	ALUSrc	0	1	1	0
	MemtoReg	0	1	x	x
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOP2	0	0	0	1

Ana kontrol sinyalleri için doğruluk tablosu



Ana kontrol PLA (programmable logic array): LA altında yatan prensip Herhangi bir mantıksal gösterim Ürünlerin toplamı olarak yazılabilir.

Single-Cycle Tasarım Problemleri

- Farzedinki fixed-period clock her komut datapath bir clock cycle kullanır.
  - CPI = 1
  - cycle time en uzun komutun uzunluğu tarafından belirlenir.
    - Ancak birkaç komut kısa clock cycle'da çalışabilir. Zaman israfı
    - Düşünün Daha komplike komutlara floating point gibi sahip olursak!
  - resources used more than once in the same cycle need to be duplicated
    - *Donanım ve chip alan israfı*

Örnek:Tek cycle gerçekleştirmede Fixed-period clock , variable-period clock kullanımı

- Kayan nokta ünitesi eklenmiş bir makine düşünün. Aşağıdaki gibi işlevsel(fonksiyonel) gecikmelerin olduğunu varsayalım:
  - *memory*: 2 ns., *ALU* and *adders*: 2 ns., *FPU add*: 8 ns., *FPU multiply*: 16 ns., *register file access* (read or write): 1 ns.
  - *multiplexors, control unit, PC accesses, sign extension, wires*: gecikme yok.
- Farzedinki komutlar aşağıdaki gibi karıştırılmıştır.
  - Tüm yüklemeler %31'lik benzer zaman almaktadır.
  - Tüm store'lar 21%'lik benzer zamanda oluşur.
  - R-format komutlar 27% de oluşmakta.
  - Dallanmalar 5% 'lik zamanda oluşmakta.
  - Şartsız dallanmalar 2% Lik zaman almakta.
  - FP toplama ve çıkarma benzer olarak toplam 7%'lik zamanda oluşmakta.
  - FP çarpma ve bölme benzer olarak toplam 7%'lik bir zaman almakta.
- *(a) a single-cycle implementation using a fixed-period clock with*
- *(b) one using a variable-period clock where each instruction executes in one clock cycle that is only as long as it needs to be (not really practical but pretend it's possible!)*

Çözüm

Instruction class	Instr. mem.	Register read	ALU oper.	Data mem.	Register write	FPU add/ sub	FPU mul/ div	Total time ns.
Load word	2	1	2	2	1			8
Store word	2	1	2	2				7
R-format	2	1	2	0	1			6
Branch	2	1	2					5
Jump	2							2
FP mul/div	2	1			1		16	20
FP add/sub	2	1			1	8		12

- Clock period for fixed-period clock = longest instruction time = 20 ns.
- Average clock period for variable-period clock =
  $8 \times 31\% + 7 \times 21\% + 6 \times 27\% + 5 \times 5\% + 2 \times 2\% + 20 \times 7\% + 12 \times 7\% = 7.0$ ns.
- Bu şekilde, $\text{performance}_{\text{var-period}} / \text{performance}_{\text{fixed-period}} = 20/7 = 2.9$

Tek çevrim (Single-Cycle) tasarımın sorunlarına çözüm

Çözüm Alternatifi: Herbir komut sınıfı için farklı çevrim süreli değişken-periyodlu clock kullanmak.

- İmkansız bir çözüm.Değişken hızlı(periyotlu) bir clock uygulama olarak teknik olarak zor

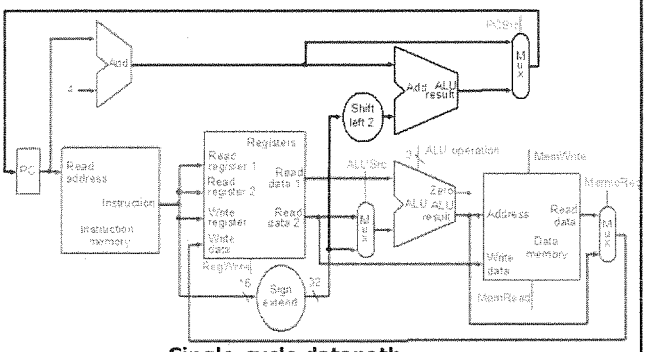
- Diğer Çözüm:
  - Daha küçük bir clock cycle kullanmak.
  - Farklı komutları, farklı sayıda clock cycleri ile gerçekleştirmek. Yani her komutun değişik fazlarınının herbirini bir cyle'lık çevrimde çözebilmek.
  - *Bu mümkündür : Bunun ismi Multicyle uygulamadır.*

Multicycle Uygulama

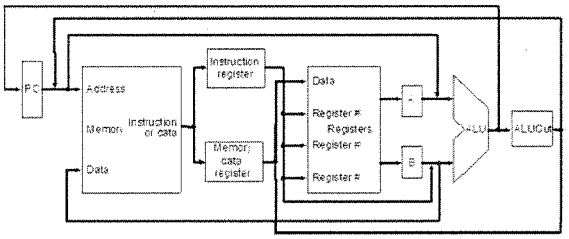
- Komutlar adımlara bölünür.
  - Herbir adım birtek clock cycle'ında yapılır.
  - Komut çevriminde herbir adım/cycle'da yapılan iş miktarı eşitlenerek dengelenir.
  - restrict each cycle to use at most once each major functional unit so that such units do not have to be replicated
  - Bir komutun farklı adımlarında, fonksiyonel birimler paylaşılarak kullanılabilir.
- Adım/Cycle arasındaki ilişki.
  - Bir cycle(çevrim) sonunda store edilen data, aynı komutun daha sonraki cycle'ında kullanılır.
  - Bu amaçla ek dahili (programcı-görünmez) reg tanıtmak gerekir
  - Programcının, store edilmiş programda; Daha sonraki komutlarda kullanılabilmesi için saklanacak data için durum elemanları, reg'ler, Memory kullanılır.

Multicycle Uygulama

- Multicycle ve singlecycle özelliklerini gösteren diyagramlar.
- Tek bir ALU kullanılır. Fazladan toplayıcı gerektirmez.
- Clock cycel'ları arasındaki kullanılacak datayı tutmak için fazladan reg'ler gerekir.

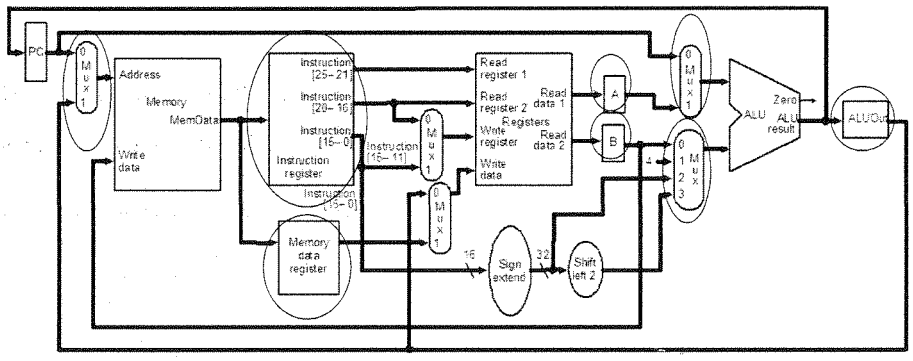


Single-cycle datapath



Multicycle datapath (high-level view)

Multicycle Datapath



Temel multicycle MIPS Datapath yapısı; R-tipi ve yük / store komutları için. kırmızı elips yeni dahili reg, mavi elips yeni mux'lardır.

Komutların adımlar halinde parçalanması

- Biz aşağıdaki avantajlar için komutları, adımlar halinde parçalayacağız.
  - Her bir adım bir clock cycle'da oluşur.
  - Herbir adımda yapılan iş miktarı yaklaşık birirbirine eşittir.
  - Her bir cycle, herbir önemli fonksiyonel birimi en fazla 1 kez kullanır. Bu yüzden böyle birimler çoğaltılamaz.
  - Fonksiyonel üniteler, aynı komutun farklı çevrimlerinde kullanılabilir.
- Bir çevrimin sonunda elde edilen data bir sonraki çevrimde kullanılabileceğinden saklanmalıdır.

Komutların adımlara bölünmesi

- Biz genel olarak bir komutu aşağıdaki yürütme adımlarına parçalayacağız. – Not1: Bütün komutların hepsi aşağıdaki adımların hepsine birden ihtiyaç duymayabilir.–Not2: Her bir adımbir clock cyle'da gerçekleşir.

1. Komutların getirilmesi ve PC'nin arttırılması (IF)
   (Instruction fetch and PC increment (**IF**))
2. Komutların çözülmesi ve register getirilmesi (ID)
   (Instruction decode and register fetch (**ID**))
3. Yürütme, hafıza adres hesabı, veya dallanmayı tamamlama (EX)
   (Execution, memory address computation, or branch completion (**EX**))
4. Hafızaya erişim veya R tipi komutun tamamlanması (MEM).
   (Memory access or R-type instruction completion (**MEM**))
5. Hafızaya okumanın tamamlanması (WB)
   (Memory read completion (**WB**))

- Herbir MIPS komutu 3-5 Cycle'da (adım) oluşur.

Adım 1: Instruction Fetch & PC Increment (**IF**)
(Komutun getirilmesi & PC'nin arttırılması)

- Komutu alıp Instruction Reg (IR)'e getirmek için PC'yi kullan. Yeni komut için PC+4 yap bunu PC'ye yaz.
- RTL kullanılarak kısaca tarif edilebilir.*RTL* (*Register-Transfer Language*):

```
IR = Memory[PC];Komut Reg (IR)'e PC'de yazılı adresteki komutu getir.

PC = PC + 4; PC'ye 4 ekle
```

Adım 2: Instruction Decode and Register Fetch (**ID**)
(Komutun decode edilmesi ve Registerlerin getirilmesi)

- Onlara İhtiyaç duyulduğunda rs ve rt reg'lerini oku.
  Komut bir dallanma komutu ise dallanma adresini hesap et.
- RTL:

```
A = Reg[IR[25-21]];

B = Reg[IR[20-16]];

ALUOut = PC + (sign-extend(IR[15-0]) << 2);
```

Adım 3: Execution, Address Computation or Branch Completion (**EX**)
(Yürütme, Adress hesabı veya dallanmanın tamamlanması)

- ALU komut tipine bağımlı olarak aşağıdaki 4 fonksiyondan birisini başarır (yürütür).
  - memory reference (Hafıza referansını hazırlar):
    `ALUOut = A + sign-extend(IR[15-0]);`
  - R-type:
    `ALUOut = A op B;`
  - branch (komutu tamamlıyor):
    `if (A==B) PC = ALUOut;`
  - jump (komutu tamamalıyor):
    `PC = PC[31-28] || (IR(25-0) << 2)`

Adım 4: Memory access or R-type Instruction Completion (**MEM**)
(Hafızaya erişim veya R-tipi komutun tamamalanması)

- Komut tipine bağımlı olarak yeniden:
- Load ve store için hafızaya erişim sağlar.
  - load
    `MDR = Memory[ALUOut];`
  - store (instruction *completes*)
    `Memory[ALUOut] = B;`
- R-type (instructions *completes*)

  `Reg[IR[15-11]] = ALUOut;`

Adım 5: Memory Read Completion (**WB**) (Hafıza Okumanın tamamlanması)

- Komut tipine bağımlı olarak yeniden;
- Yüklenen değeri geri yazar.(Komut tamamlama)

```
Reg[IR[20-16]]= MDR;
```

Önemli: There is no reason from a datapath (or control) point of view that Step 5 cannot be eliminated by performing

```
Reg[IR[20-16]]= Memory[ALUOut];
```

for loads in Step 4. This would eliminate the MDR as well.

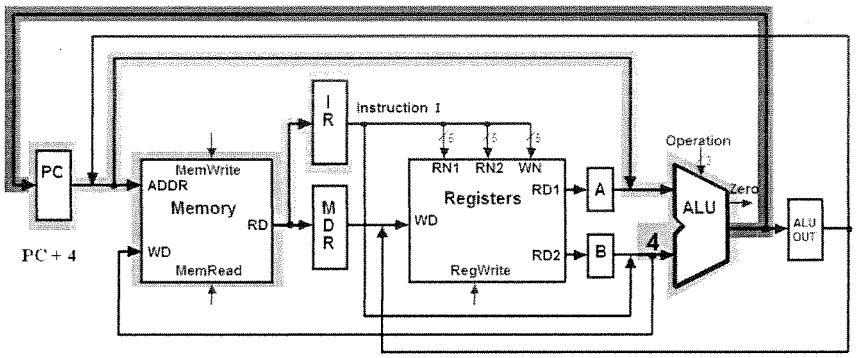
The reason this is not done is that, to keep steps balanced in length, the design restriction is to allow each step to contain *at most* one ALU operation, or one register access, or one memory access.

Bir komutun icra edilme sürecinin özeti

Step	Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
1: IF	Instruction fetch	IR = Memory[PC] PC = PC + 4			
2: ID	Instruction decode/register fetch	A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
3: EX	Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	If (A ==B) then PC = ALUOut	PC = PC [31-28] || (IR[25-0]<<2)
4: MEM	Memory access or R-type completion	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
5: WB	Memory read completion		Load: Reg[IR[20-16]] = MDR		

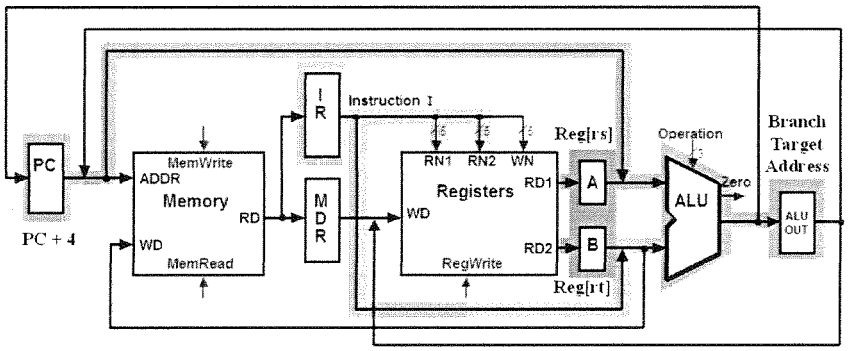
Multicycle Yürütme Adım (1): Instruction Fetch (Komutun getirilmesi)

```
IR = Memory[PC];
PC = PC + 4;
```



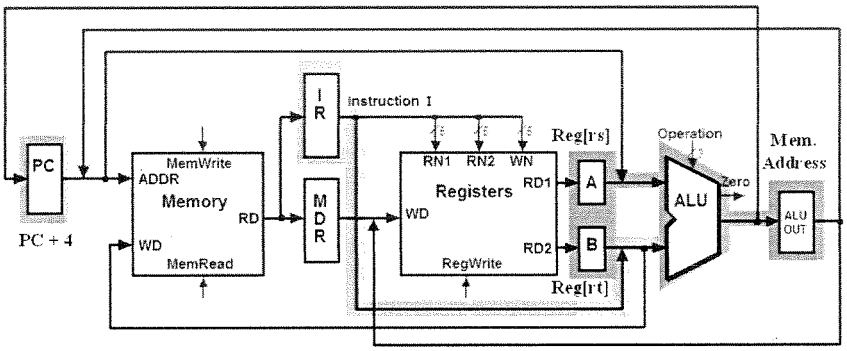
Multicycle Execution Step (2): Instruction Decode & Register Fetch

```
A = Reg[IR[25-21]];              (A = Reg[rs])
B = Reg[IR[20-15]];              (B = Reg[rt])
ALUOut = (PC + sign-extend(IR[15-0]) << 2)
```



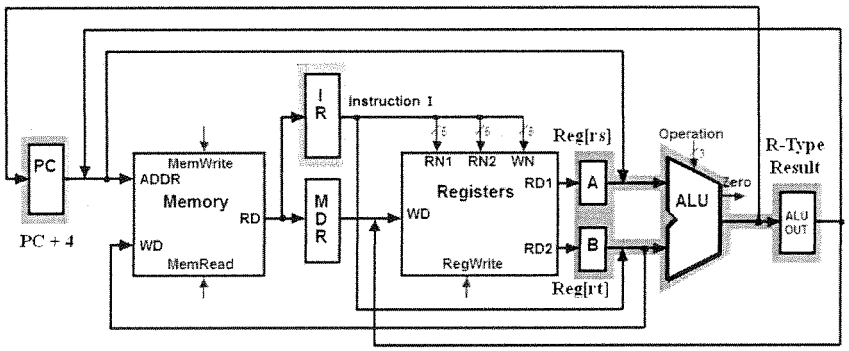
Multicycle Execution Step (3): Memory Reference Instructions

```
ALUOut = A + sign-extend(IR[15-0]);
```



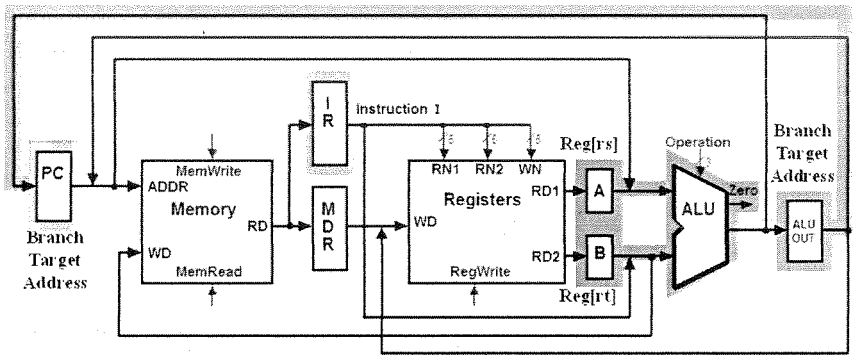
Multicycle Execution Step (3): ALU Instruction (R-Type)

```
ALUOut = A op B
```



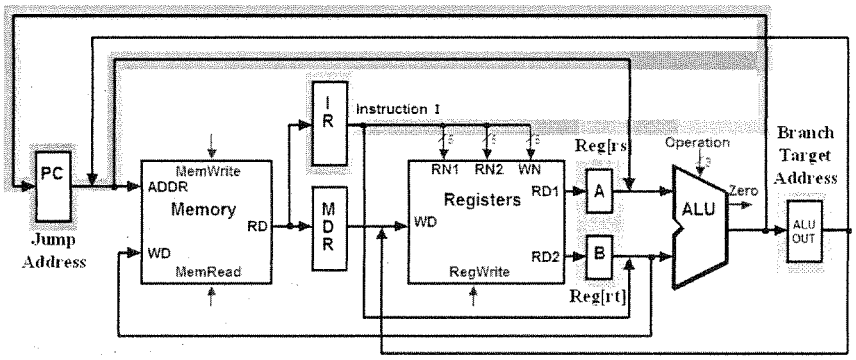
Multicycle Execution Step (3): Branch Instructions

```
if (A == B) PC = ALUOut;
```



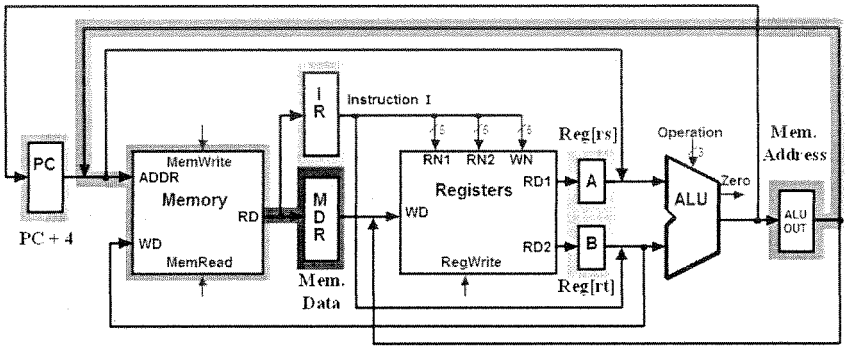
Multicycle Execution Step (3): Jump Instruction

```
PC = PC[31-28] concat (IR[25-0] << 2)
```



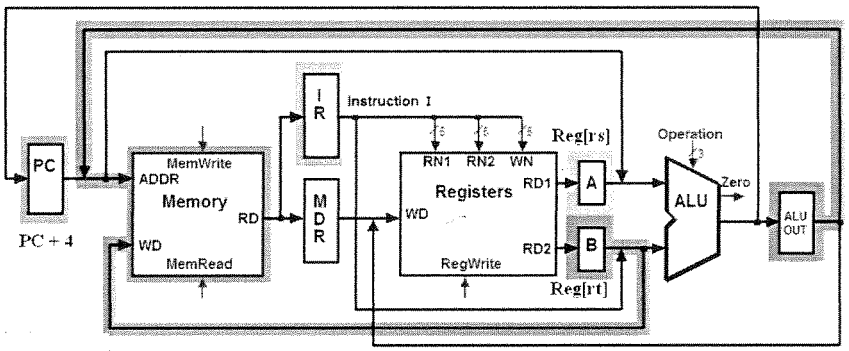
Multicycle Execution Step (4): Memory Access - Read (`lw`)

```
MDR = Memory[ALUOut];
```



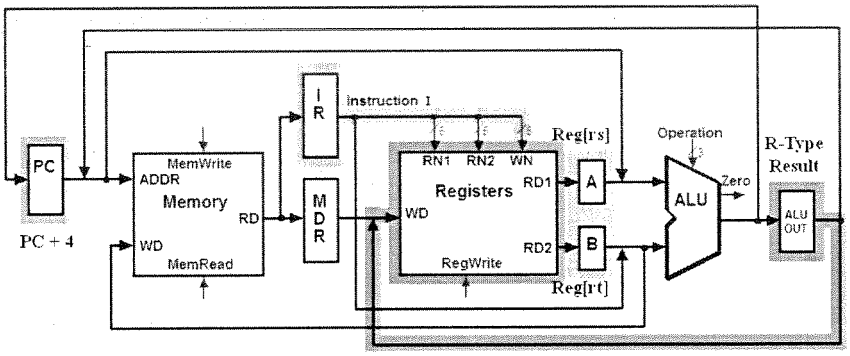
Multicycle Execution Step (4): Memory Access - Write (`sw`)

```
Memory[ALUOut] = B;
```



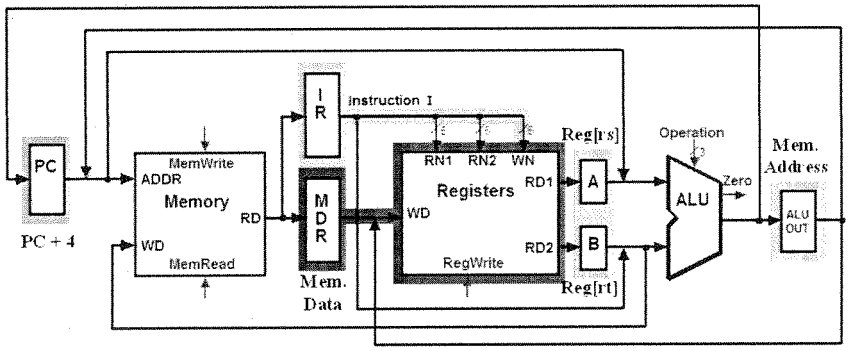
Multicycle Execution Step (4): ALU Instruction (R-Type)

```
Reg[IR[15:11]] = ALUOUT
```

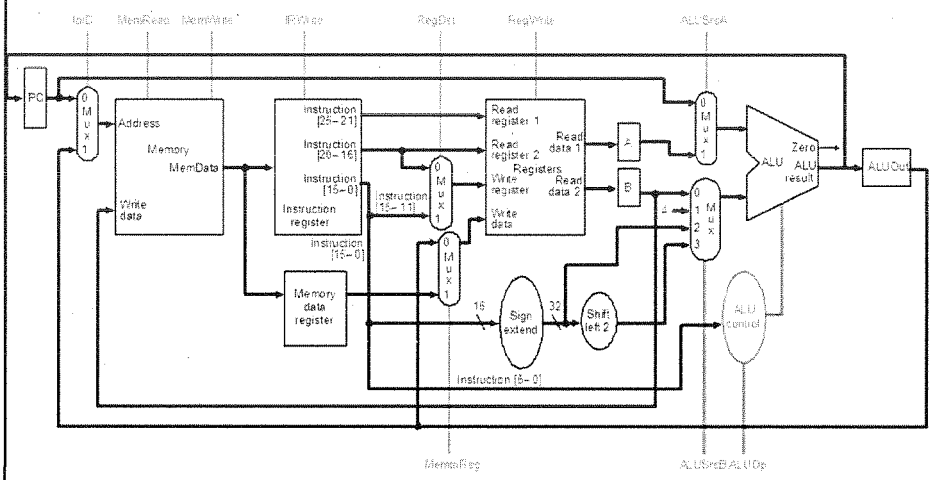


Multicycle Execution Step (5): Memory Read Completion (`lw`)

```
Reg[IR[20-16]] = MDR;
```

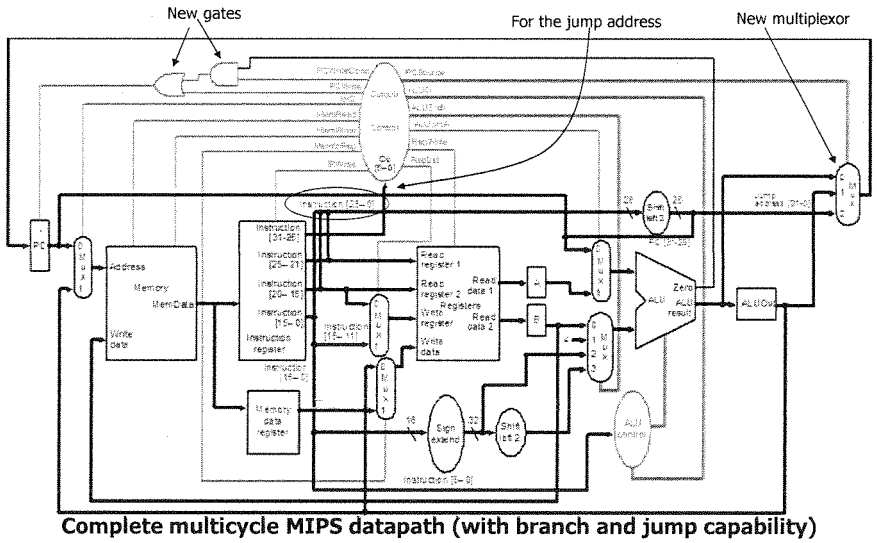


Multicycle Datapath with Control I



... with control lines and the ALU control block added – *not all* control lines are shown

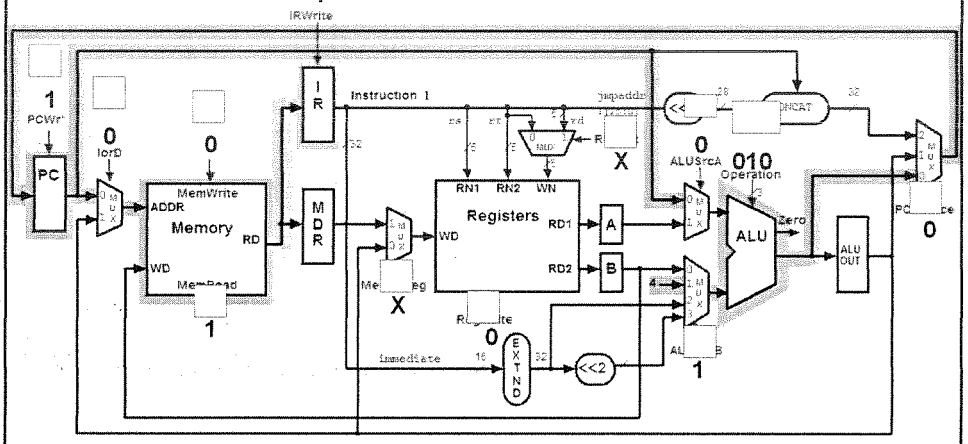
Multicycle Datapath with Control II



Complete multicycle MIPS datapath (with branch and jump capability) and showing the main control block and all control lines

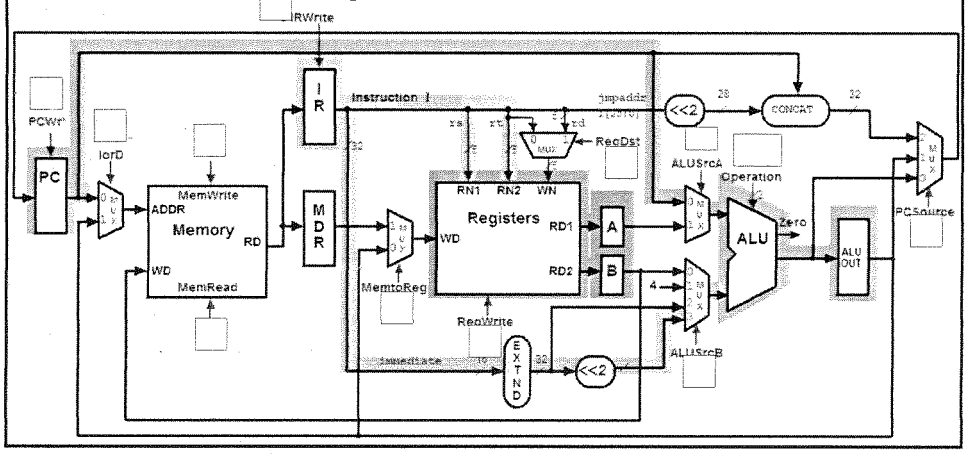
Multicycle Control Step (1): Fetch

IR = Memory[ ];
PC = PC + 4;



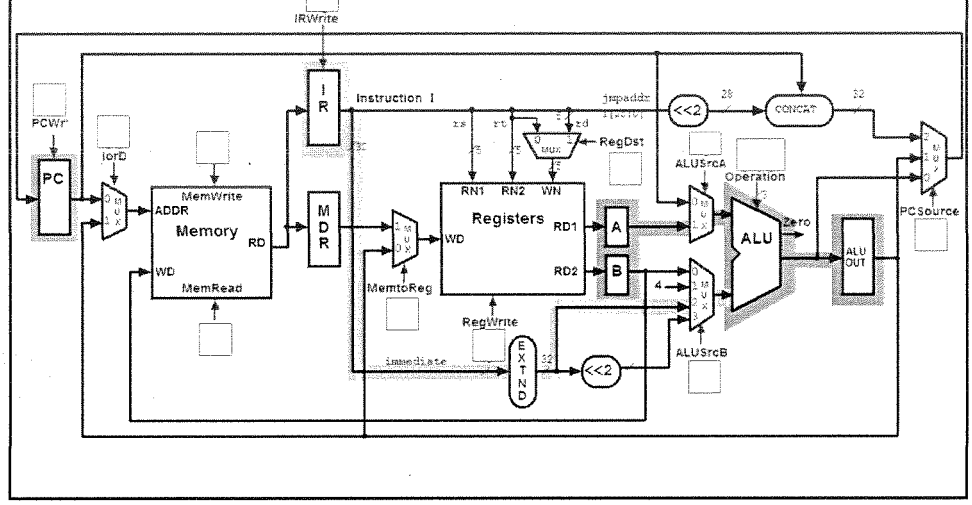
Multicycle Control Step (2): Instruction Decode & Register Fetch

A = Reg[IR[25-21]]; (A = Reg[rs])
B = Reg[IR[20-15]]; (B = Reg[rt])
ALUOut = (PC + sign-extend(IR[15-0]) << 2);



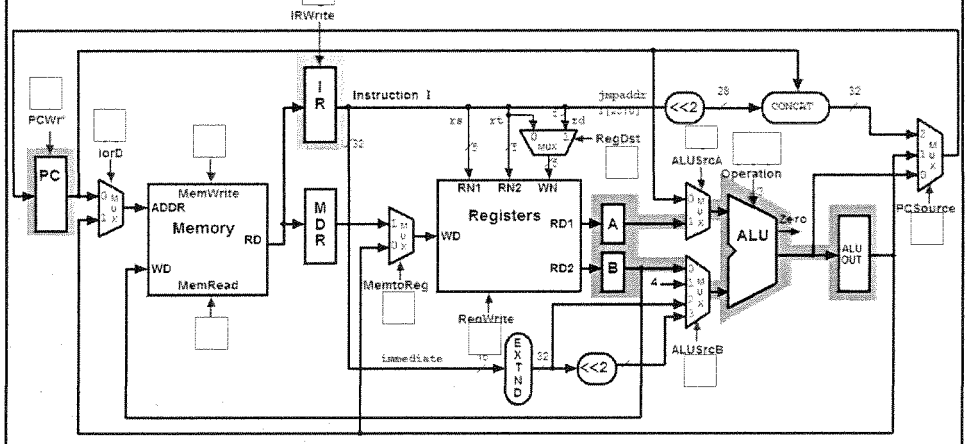
Multicycle Control Step (3): Memory Reference Instructions

ALUOut = A + [illegible]gn-extend(IR[15-0]);



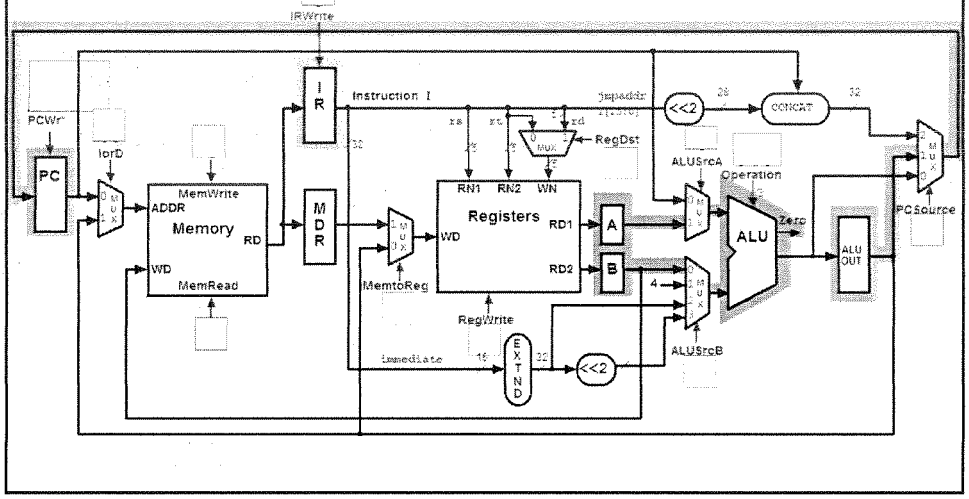
Multicycle Control Step (3): ALU Instruction (R-Type)

ALUOut = A o[illegible]B;



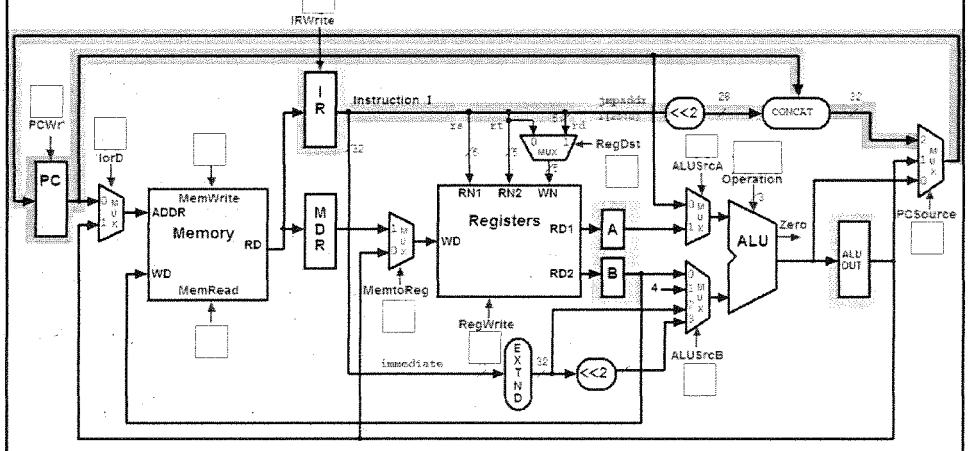
Multicycle Control Step (3): Branch Instructions

if (A == B) PC = ALUOut;



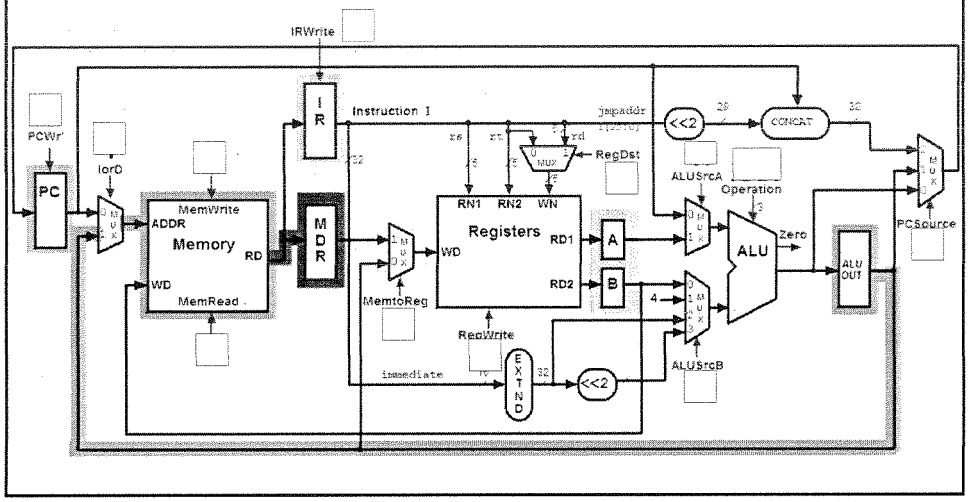
Multicycle Execution Step (3): Jump Instruction

PC = PC[21-28] concat (IR[25-0] << 2);



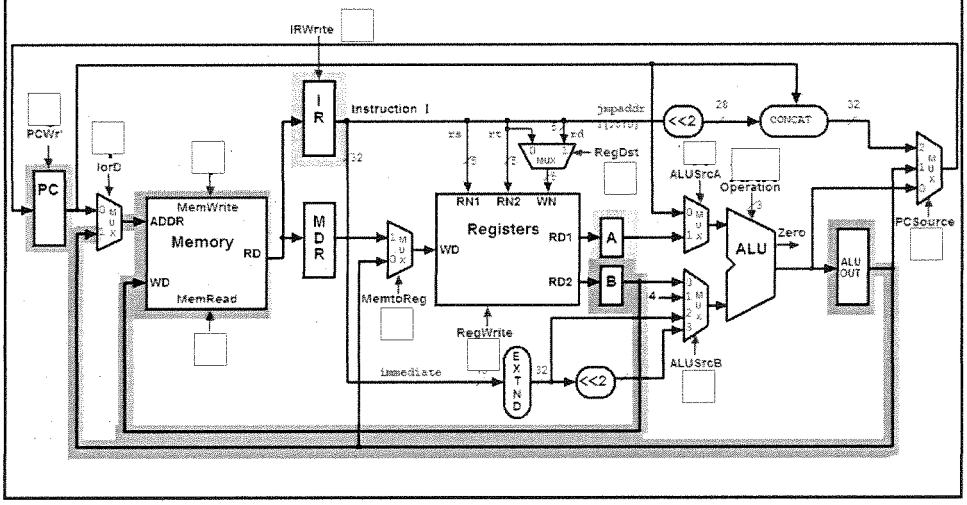
Multicycle Control Step (4): Memory Access - Read (lw)

MDR = Memory[ALUOut];



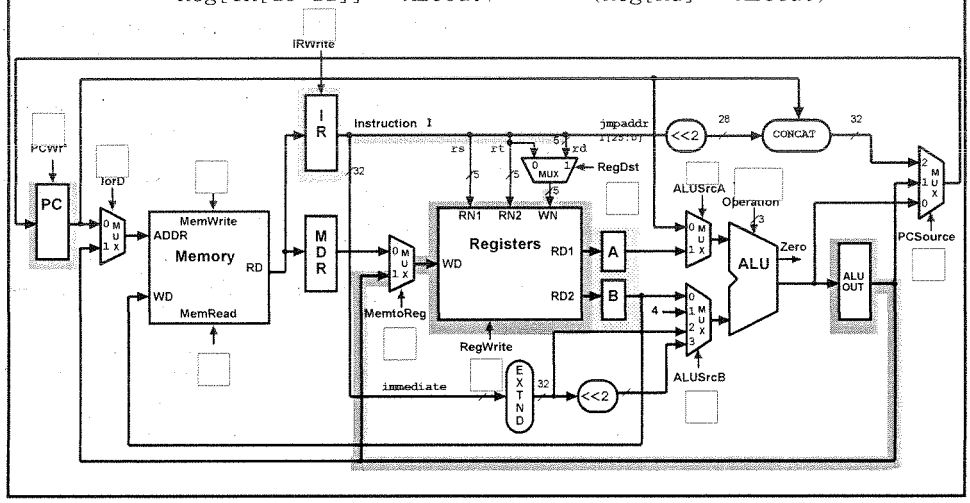
Multicycle Execution Steps (4) Memory Access - Write (sw)

Memory[ALUOut] = B;



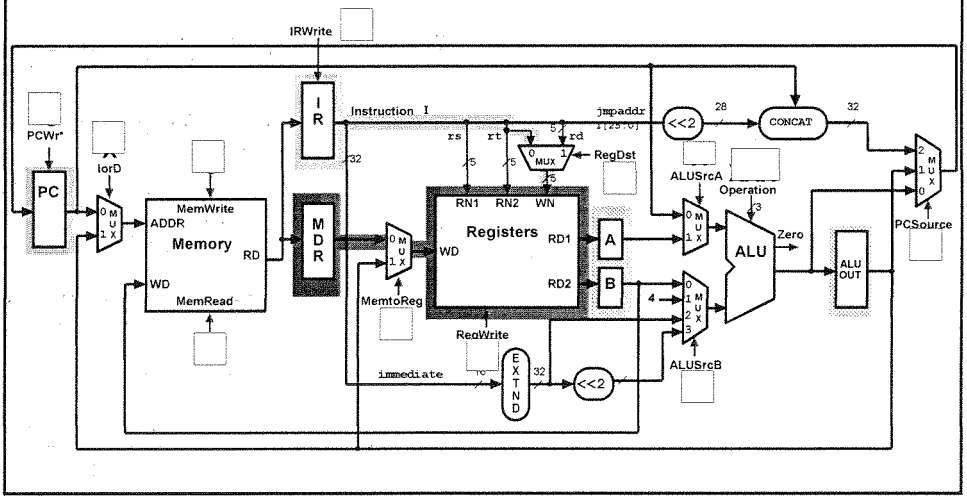
Multicycle Control Step (4): ALU Instruction (R-Type)

Reg[IR[15:11]] = ALUOut; (Reg[Rd] = ALUOut)



Multicycle Execution Steps (5) Memory Read Completion (lw)

Reg[IR[20-16]] = MDR;



Basit Sorular

- *Bu kodların yürütülmesi kaç cycle da tamamalanır?*

```
        lw $t2, 0($t3)
        lw $t3, 4($t3)
        beq $t2, $t3, Label #assume not equal
        add $t5, $t2, $t3
        sw $t5, 8($t3)
Label:  ...
```

- *Yürütülen 8. cycle'da ne oluşur?*

Clock time-line

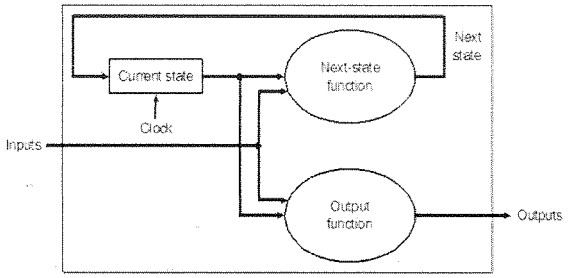
- *In what cycle does the actual addition of $t2 and $t3 takes place?*

Kontrol Gerçekleştirme

- Kontrol sinyallerinin değeri aşağıdakilere bağlıdır.
  - Hangi komut yürütülüyor
  - Hangi adım icra ediliyor
- Bir sonlu durum makinası kullanmak için bilgilerinizi kullanın
  - finite state machine grafiksel olarak belirtin, veya
  - Microprogramming kullanın
- Gerçekleştirme belirtimlerden(tanımlardan) elde edilir.

Finite State Machines

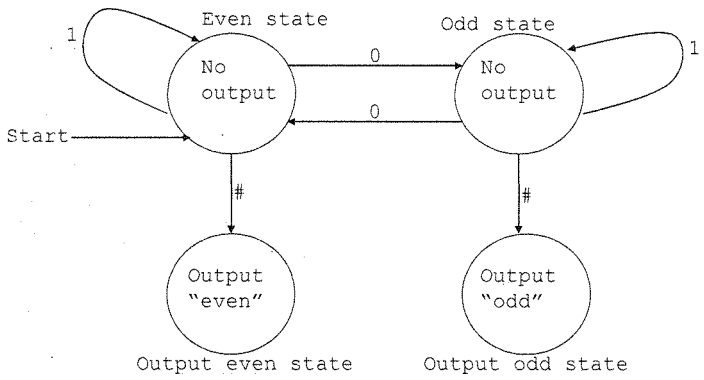
- Finite state machines (FSMs):
  - Durumlar kümesi ve
  - Sonraki durum fonksiyonu, giriş ve şimdiki durum tarafından hesaplama
  - Çıkış fonksiyonu, olası giriş ve şimdiki durum tarafından hesaplama



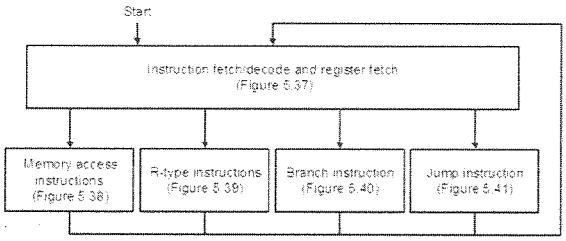
  - *Moore machine'sını kullanacağız.* Çıkış sadece şimdiki duruma bağlıdır.

Örnek: Moore Makinası

- Moore makinası aşağıdadır. # tarafından sonlandırılan giriş binary stringi ile çıkış; eğer stringdeki 0'ların sayısı çift ise "odd değilse "even olacaktır.

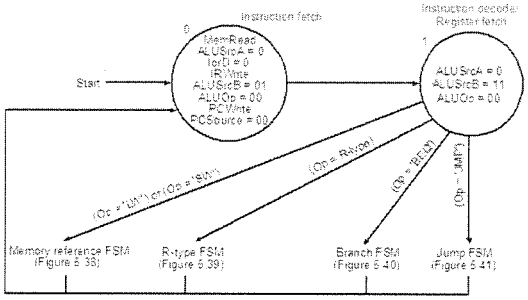


FSM Kontrol: yüksek Seviye bakış



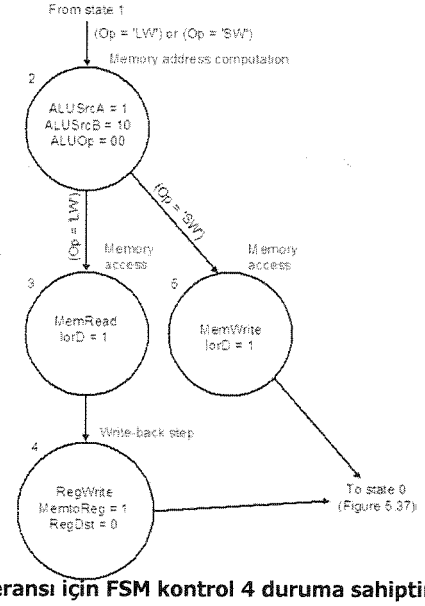
FSM kontrolün yüksek seviye bakışı

Sinyaller durum dairesi içinde gösterilmektedir.



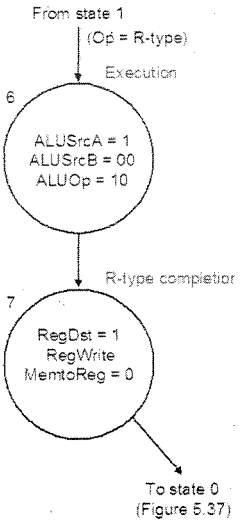
Her komutun komut yakalama ve decode işlemi benzerdir.

FSM Kontrol: Hafıza referens



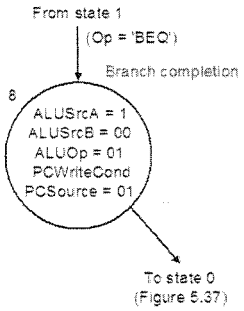
Hafıza-referansı için FSM kontrol 4 duruma sahiptir.

FSM Control: R-Tip Komut



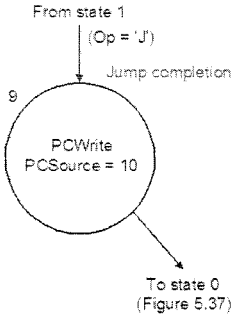
R tip komut gerçekleştirmek için FSM kontrol 2 duruma sahiptir.

FSM Control: Branch Instruction



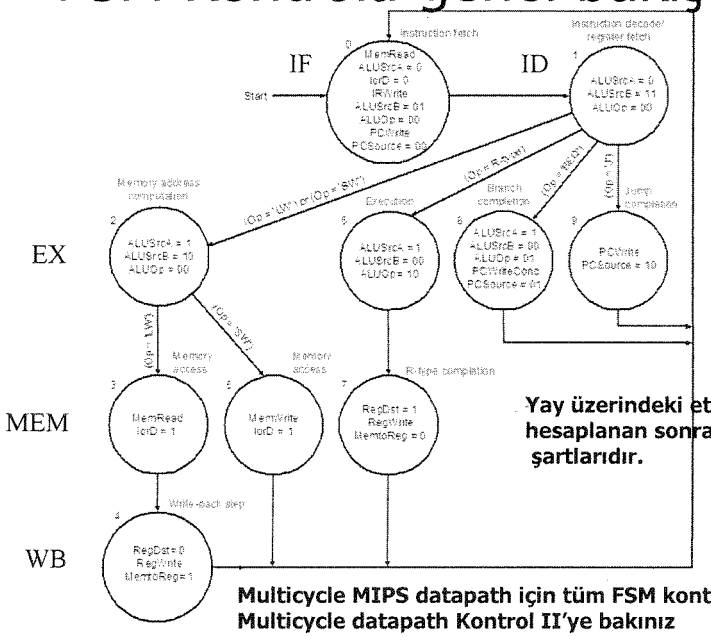
Dallanma komutları için FSM 1 duruma sahiptir.

FSM Kontrol: Jump Komutu



Jump komutu için FSM kontrol 1 duruma sahiptir.

FSM Kontrola genel bakış



Yay üzerindeki etiketler hesaplanan sonraki durum şartlarıdır.

Multicycle MIPS datapath için tüm FSM kontrolü: Multicycle datapath Kontrol II'ye bakınız

Örnek: multicycle CPU'da CPI

- Farzedelim
  - Önceki stayt'ın kontrol tasarımı
  - 22% yükleme , 11% depolama, 49% *R-tip işlemler*, 16% dallanma, and 2% *jump komutları olsun*
- Herbir adım 1 clock cycle gerektirirse CPI nedir.
- Çözüm:
  - Her bir komut sınıfı için önceki slayttan clock cycle'ların sayısı:
    - *yükleme* 5, depolama 4, *R-tip komutlar* 4, dallanma 3, jump 3
  - CPI = CPU clock cycles / komut sayısı count

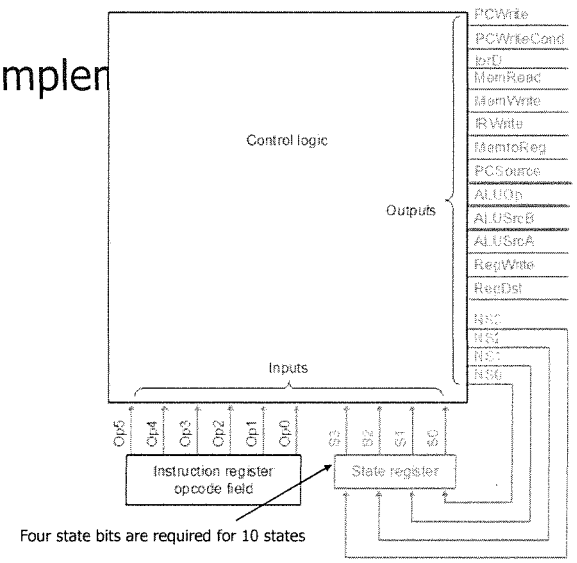
    $= \Sigma$ (komut sayısı$_{class\ i}$ $\times$ CPI$_{class\ i}$) / komut sayısı

    $= \Sigma$ (komut sayısı$_{class\ I}$ / komut sayısı) $\times$ CPI$_{class\ I}$

    $= 0.22 \times 5 + 0.11 \times 4 + 0.49 \times 4 + 0.16 \times 3 + 0.02 \times 3$

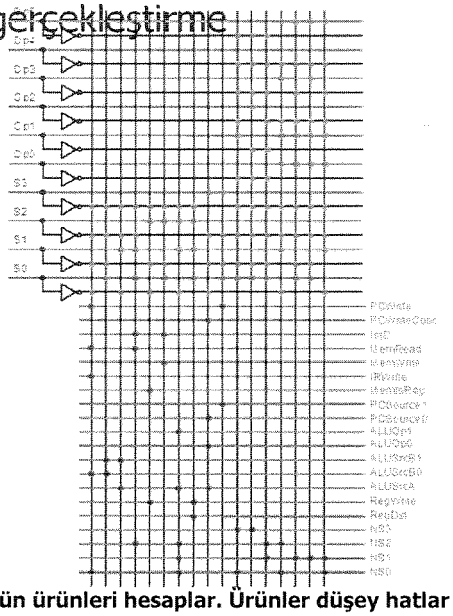
    $= 4.04$

FSM Control:Impler



FSM gerçekleştirmeye yüksek seviye bakış: Kombinasyonal lojik bloklara girişler Şimdiki durum sayısı ve komut opcode bitleridir. Çıkışlar sonraki durum sayısı ve şimdiki durum için iddia edilen kontrol sinyalleridir.

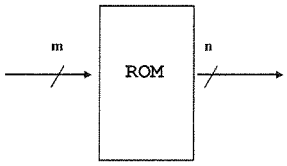
FSM Kontrol:PLA gerçekleştirme



Üst yarı AND düzlemidir ve bütün ürünleri hesaplar. Ürünler düşey hatlar tarafından düşük OR düzlemiyle taşınır. Herbir çıkış için toplam terim ilgili yatay düzlem tarafından verilir. Örnek IorD = S0.S1.S2.S3 + S0.S1.S2.S3

FSM Kontrol:ROM Gerçekleştirme

- ROM (Read Only Memory)
  - Hafıza yerlerinin değerleri zaman öncesinde sabittir.
- Bir rom doğruluk tablosu gerçekleştirmek için kullanılabilir.
  - Eğer adres m bit ise biz 2^m alanını adresleyebiliriz.
  - Çıkışlar giriş adresindeki bitlerdir.



m = 3
n = 4

address	output
0 0 0	0 0 1 1
0 0 1	1 1 0 0
0 1 0	1 1 0 0
0 1 1	1 0 0 0
1 0 0	0 0 0 0
1 0 1	0 0 0 1
1 1 0	0 1 1 0
1 1 1	0 1 1 1

m-giriş n-çıkış ise ROM 2^m x n bittir. – böyle bir ROM n bitlik 2^m dizisi olarak düşünülebilir.

FSM Kontrol: ROM vs. PLA

- ROM geliştirmek:
  - 4 durum bitleri 16 çıkış sinyali verir.ROM'un 2^4 x 16 bitleri
  - Bütün 10 bit 4 sonraki durum bitlerini verir.ROM'un 2^{10} x 4 bitleri
  - Toplam – ROM 4.3K bit olur
- PLA daha küçüktür.
  - Ürün terimleri paylaşılabilir.
  - Aktif bir çıkış üretmek için yanlızca girişlere ihtiyaç vardır
- PLA boyut = (#girişler × #ürünler terimi) + (#çıkışlar × #ürünler terimi)
  - FSM Kontrol PLA = (10x17)+(20x17) = 460 PLA hücresi
- PLA hücreleri bir ROM hücre boyutundadır. (biraz daha büyük)

Microprogramming

- Microprogramming iözel FSM kontrol methodudur. (Bir programlama dilini andıran yada grafikten ziyade text)
  - Eğer komut seti büyük veya her komutun cycle sayısı büyük ise FSM çok büyük olduğundan bu uygundur.
  - Böyle grafiksel gösterim durumlarında zor olabilir. Binlerce durum olduğu ve onlarla bağlantılı yaylar olduğunda
  - Bir microprogram özelliği: Gerçekleştirme ROM veya PLA ile gerçekleştirmedir.
- Bir *microprogram microkomutlar dizisidir.*
  - Herbir microinstruction8 alana sahiptir. (Etiket + 7 fonksiyon)
    - Label: Microcode dizisini kontrol etmede kullanılır.
    - ALU Kontrol:ALU tarafından yapılacak işlemleri belirtir.
    - SRC1: İlk ALU operandı için kaynağı belirtir.
    - SRC2: ikinci ALU operandı için kaynağı belirtir.
    - Register Kontrol: Register file için read/write belirtir.
    - Hafıza: Hafıza için read/write belirtir.
    - PCWrite Kontrol :PC'nin yazmasını belirtir.
    - Sequencing: sonraki komutu seçmeyi belirtir.

Microprogramming

- The *Sequencing alan değeri* microprogramın yürütme sırasını belirler.
  - value *Seq* : *Sonraki microkomuta geçme kontrolü*
  - value *Fetch* : sonraki başlanacak ilk microkomuta dallanma ör: mikroprogramdaki ilk microkomut gibi
  - value *Dispatch i* : gönderme tablo girişi (*dispatching olarak adlandırılır*) ve kontrol girişi tabanlı microkomuta dallanma :
    - Dispatching bir tablo oluşturma aracı tarafından gerçekleştirilir, *dispatch tablo olarak adlandırılır*, onun girişleri microkomut etiketleridir ve o kontrol girişi tarafından indexlenir. Çoklu dispatch tabloları olabilir. –sıralama alanındaki *Dispatch i* değeri göstermeye *i ninci* dispatch tablo kullanılır.

Kontrol Microprogram

- The microprogram corresponding to the FSM control shown graphically earlier:

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

Microprogram 10 mikrokomut içerir.

Dispatch ROM 1		
Op	Opcode name	Value
000000	R-format	Rformat1
000010	jmp	JUMP1
000100	beq	BEQ1
100011	lw	Mem1
101011	sw	Mem1

Gönderme Tablo 1

Dispatch ROM 2		
Op	Opcode name	Value
100011	lw	LW2
101011	sw	SW2

Gönderne Table 2

Microcode: Trade-offs

- Özellikleri Avantajları
  - Tasarlamak ve yazmak kolaydır.
  - Tipik olarak imalatcı mimari ve microkodu paralel tasarlar
- Gerçekleştirme Avantajları
  - Hafızadaki değeri değiştirmek kolaydır (e.g., off-chip ROM)
  - Diğer mimariler taklit edilebilir.
  - Dahili registerlerden yararlanılabilir.
- Gerçekleştirme dezavantajları
  - Kontrol şimdilerde aynı chip üzerinde gerçekleştirilir. Off Chip ROM'un sahip olduğu avantaja sahip değildir.
  - ROM on-board cache'den hızlı değildir.
  - Microkodu değiştirmeye az ihtiyaç vardır. Genel amaçlı bilgisayarlar özel uygulamalar için tasarlanmış bilgisayarlardan çok daha fazla kullanılmaktadır.

Özet

- *Bu bölümde datapath tasarımı ve bütün modern bilgisayar mimarisi tanımlandı*
- Multicycle datapath'ler single-cycle'a göre 2 büyük avantaj sunar.
  - Fonksiyonel birimler bir tek komutla yeniden kullanılabilir.
  - Daha kısa execution path'ler ile komutlar birkaç cycle tüketilmesiyle daha hızlı tamamlanabilir.
- Modern bilgisayarlar, gerçekte, take the multicycle paradigm to a higher level to achieve greater instruction throughput:
  - *pipelining* (bir sonraki komut) birçok komutu(farklı cycle'a sahip olsalar bile) eşzamanlı yürütme
  - *MIPS mimarisi pipeline olarak tasarlandı.*

PİPELİNE (BORU HATTI) İşlemi ile Performansın Geliştirilmesi

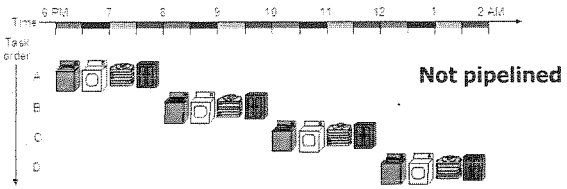
Paralel İşleme ve PİPELİNE

- Aynı zamanda, birden fazla veya çok sayıda işlemin yapılabilmesi bilgisayar sisteminin hızını (Throuhput) arttırır.
- Paralel işlemede ise aynı anda birden fazla komut icra edilir. Örneğin, bir zaman diliminde bir komut ALU'da yürütülürken(execute), aynı zaman diliminde daha sonraki komut ise bellekten okunabilir. Bu işlem tek işlemcili paralel çalışmaya örnektir.
- Paralel çalışmaya diğer örnek , birden fazla işlemcisi bulunan sistemlerdir.
- Paralel işleme aşağıdaki başlıklarda incelenebilir.
- 1- PİPELİNE (Boru Hattı) İşlemleri
- 2-VEKTÖR İŞLEMLERİ
- 3-DİZİ İŞLEMCİLERİ

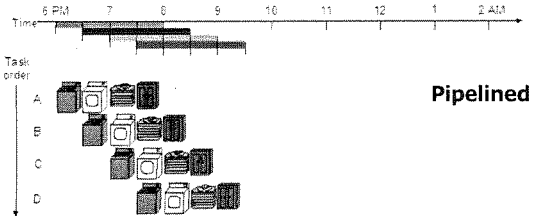
Pipeline bir yürütme tekniği olup, artmetik alt işlemlerde veya bir komutun fazlarının (evrelerinin) işlenmesi sırasında üstüste gelmesidir. Vektör işlemleri büyük boyutlu vektörel veya matris işlemlerinin yapılması içindir. Dizi İşlemcileri ise büyük çaplı diziler üzerinde işlem yaparlar.

Pipelining

- Bir işe başladıktan sonra en kısa sürede bitir.!! Zaman kaybetme!

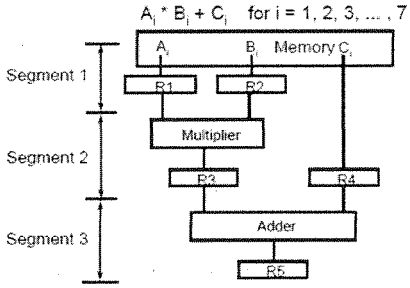


Farz etki her bir görev (yıkama-kurutma- katlama- dolaba yerleştirme) 30 dakikadır. Ve ayrı görevler için ayrı donanımlar kullanılıyorsa, farklı görevler aynı zamanda üstüste çakışabilir.



PİPELİNE İŞLEMLERİNİN ANLAŞILMASI

- Pipeline işlemi birtakım işlemlerin topluluğudur ve boru hattını oluşturan her bir kesimde (segmentte) herbir farklı işin farklı alt işlemleri aynı zamanda gerçekleşir. Bir kesimde elde edilen sonuç aynı iş için sonraki kesimde kullanılmak üzere saklanır.



$R1 \leftarrow A_i$, $R2 \leftarrow B_i$	Load A_i and B_i
$R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$	Multiply and load C_i
$R5 \leftarrow R3 + R4$	Add

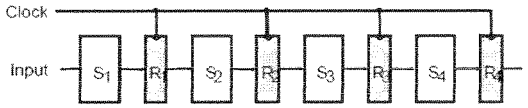
Her bir kesimde icra edilecek alt işlemler şekilde verilmiştir. 5 Registerden herbiri, her clock cyce'lında yeni veriler ile yüklenir.

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A1	B1			
2	A2	B2	A1 * B1	C1	
3	A3	B3	A2 * B2	C2	A1 * B1 + C1
4	A4	B4	A3 * B3	C3	A2 * B2 + C2
5	A5	B5	A4 * B4	C4	A3 * B3 + C3
6	A6	B6	A5 * B5	C5	A4 * B4 + C4
7	A7	B7	A6 * B6	C6	A5 * B5 + C5
8			A7 * B7	C7	A6 * B6 + C6
9					A7 * B7 + C7

Reg'lerin içeriklerinin değişimi

Aynı biçimde alt işlemlere parçalanabilen ve aynı karışıklığa sahip her işlem PİPELİNE olarak işlenebilir.

Herbir alt işlem segment(kesimde) yapılır. Bir önceki işlemde elde edilen sonucun bir sonraki işlemde kullanılabilmesi için Reg'lere ihtiyaç vardır. Aşağıdaki durum 4 kesimli bir PİPELiNE işlemidir.



PİPELİNE davranışı uzay-zaman diyagramı ile gösterilebilir. 4 kesimli bir PİPELİNE işlemi ile 6 tane görevin icra edilmesi aşağıdaki şemada verilmiştir.

Segment	1	2	3	4	5	6	7	8	9
1	T1	T2	T3	T4	T5	T6			
2		T1	T2	T3	T4	T5	T6		
3			T1	T2	T3	T4	T5	T6	
4				T1	T2	T3	T4	T5	T6

Clock cycles

PİPELİNE işlemenin hızı (Kazancımız ne?)

n: Görev Sayısı (Bir assemler programdaki işlenecek komut sayısı diyebiliriz.)

PİPELİNE çalışmayan bir makine için;

tn: Clock cycle (her bir görevi bitirmek için gerekli zaman)

t1: n tane görevi tamamlamak için gerekli harcanacak zaman.

→ **t1 = n * tn**

PİPELİNE bir makine için. (k kesimli (durumlu))

tp: Clock cycle (Herbir alt işlemin tamamlanması için gerekli zaman)

tk: n adet görevin başarılması için gerekli zaman

→ **tk = (k + n - 1) * tp**

- **Hızlanma (Speedup)**
- **Sk:** Speedup

→ **Sk = t1 / tk = n*tn / (k + n - 1)*tp**

n görev sayısı arttıkça (k + n -1) terimi yaklaşık n olur. Buna göre aşağıdaki bağıntı ortaya çıkar.

Sk= tn / tp

Eğer; pipeline ve pipeline olmayan bilgisayarlarda bir görevin tamamlanması için geçen zaman eşit ise, Bu durumda; pipeline toplam hızlandırması k (kesim sayısı) kadar olabilir.

$$\lim_{n \to \infty} S_k = \frac{t_n}{t_p} \quad (= k, \text{ if } t_n = k * t_p)$$

Örnek:
4-kesimli pipeline'da, bir alt işlemin her bir kesimde işlenebilmesi için gerekli zaman;

tp = 20nS olsun.

İcra edilecek görev sayısı 100 olsun. Sk hızlanma değerini hesaplayınız.

Çözüm:

- Pipeline olmayan bir sistemde, tekbir görev için harcanacak zaman 20*4 = 80nS
  Tüm görevin icra edilmesi için harcanacak zaman: 100*20*4 =8000 ns.

- Pipeline makine için toplam süre ; (k + n - 1)*tp =(4+100 -1)*20=2060 ns

- Buna göre;

  Sk = 8000 / 2060 = 3.88

Elde edilir.
Önemli not: Bütün alt işlemlerin icra sürelerinin aynı olduğunu kabul ediyoruz.

- PİPELİNE İşlemenin 2 temel uygulaması vardır.
- 1- Aritmetik işlemlerde PİPELİNE
- 2-Komut Yürütmede PİPELİNE

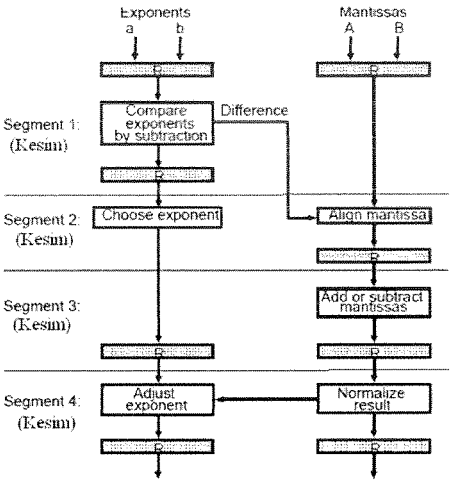
Aritmetik İşlemlerde PİPELİNE

PİPELİNE aritmetik birimleri, FP işlemler için ve çarpma işlemleri için önemli bir hızlandırıcı yapıdır.

Floating-point Toplama

$X = A \times 2^a$
$Y = B \times 2^b$

[1] Exponentlerin karşılaştırılması
[2] Mantisanın hizalanması
[3] Mantisaların toplanması/Çıkarılması
[4] Sonuç Normalizasyonu



Örnek:

16 tabanlı uzaydaki x ve y FP sayılarını toplayınız.

$x = 0.188 \times 16^2$
$y = 0.C80 \times 16^1$

Kesim 1 : üsler eşit mi? 2 - 1 = 1 çıktığından eşit değil.

Kesim 2 : Kesirlerin hizalanması. y sayısını 1 defa sağa kaydırarak üsler eşitlenir.

$x = 0.1880 \times 16^2$
$y = 0.0C80 \times 16^2$

Kesim 3: Kesirlerin toplanması/çıkarılması

$Z = 0.25 \times 16^2$

Kesim 4: Sonucun Normalizasyonu

Normalizeye gerek yok.

$Z = 0.25 \times 16^2$

4 kesim için harcanan zamanlar: t1=60ns, t2=70ns,t3=100ns, t4=80ns , fiziksel gecikmeler 10ns olsun. Buna göre;

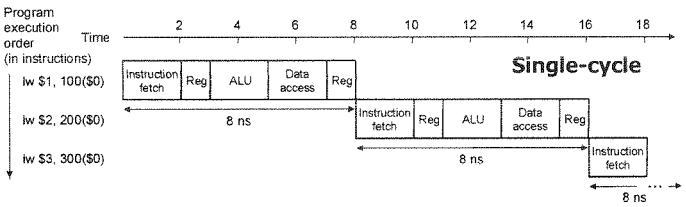
pipeline'sız bir makinada 1 toplama işlemi için: 60+70+100+80+10 = 320ns

Aritmetik işlem için 4 kesimli pipeline 'da ise herbir kesim de hrcanan süre farklı olduğundan en kötü durum olarak pipeline clock cycel'ını 100ns alarak tp = 100+10 =110 ns elde ederiz. Buna göre;

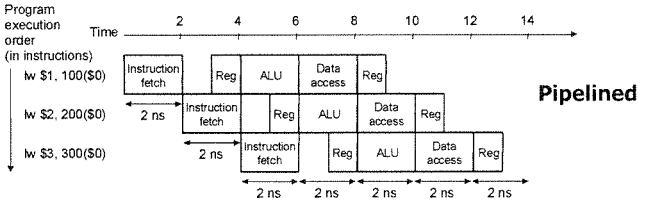
Sk= tn / tp = 320 / 110 = 2.9

Komut Yürütme için Pipeline

Single-Cycle Komut Yürütme: Planlama



Farzedin ki memorye erişim ve ALU işlemleri 2 ns ; register erişimi 1 ns. Buna göre, single cycle clock 8 ns'dir. Pipelined clock cycle ise 2 ns.



Pipelining: Unutma

- Pipeline işlemi tek bir görev (veya komut işlemesi) için harcanacak zamanı (latency) azaltmaz fakat görevin bütünü (bir programın sıralı komutları v.b) için harcanan zamanı azaltır (Verilen sabit bir zamanda yapılan iş miktarını –throughput- arttırır).
- Pipeline hızı en uzun durum ile sınırlıdır.
  - *potential* hızlandırıcı= Boru durumları sayısıdır (kesim sayısı).
  - *Dengesiz boru durumları hızı azaltır (her kesim için farklı süreler).*
- Pipeline sürecinde herhangi bir fazda boş kalmak pipeline'ı yavaşlatır.
- En verimli pipeline işlemi, komutların parçalanmasıyla oluşan alt işlem sürelerinin (fetch, decode v.b) eşit sürelerde işlenmesi ile olur.

Örnek Problem

- *Problem: Çamaşır yıkama örneği için aşağıdaki tabloyu doldurmak isteyelim.*
  1. *Her bir durumun uzunluğu(stage lengths) 30, 30, 30 30 dak. olsun.*
  2. *Durumların uzunluğu (stage lengths) , 20, 20, 60, 20 dak. olsun.*

Person	Unpipelined finish time	Pipeline 1 finish time	Ratio unpipelined to pipeline 1	Pipeline 2 finish time	Ratio unpiplelined to pipeline 2
1	120 d.	120 d.	120/120 = 1	120 d.	120/120 = 1
2	2x120=240d	120+30=150d	240/150 = 1.6	180 d	240/180 = 1.33
3					
4					
n			n x 120/(120+((n-1)x30)		

- *Pipeline'ı hızlandırmak için bir formül ortaya çıkaralım?*

Dört kesimli (Segmentli) Komut İçin Pipeline

Dört aşamaya (faz - Kesim) bölünmüş komutlar için kesimler ;

FI: Komutu getir
DA: Komutu çöz, etkin adresi hesapla
FO: Veriyi getir
EX: Komutu yürüt

Not: komut ve data hafızalarına ayrı ayrı erişildiğini düşününüz.

Dallanma komutu olmadığı sürece, hiçbir kesim boş geçilmez. Farklı işlemler yapar. Ancak 3. komutun dallanma olduğu durumu inceleyelim. Neler olur ve neden olur? Niye 5.ve 6. adımlarda işlem yapılmaz?

Dallanma komutunun işleyip dallanılacak adres ortaya çıkıncaya kadar bir sonraki komut ile ilgili sadece Fetch işlemi yapılabilir. 5.6. adımlarda işlem yapılmaz.

Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Komut	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
Dallanma	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

MIPS'te PİPELİNE

- MIPS için Pipeline işlem neden kolaydır?
  - **Bütün komutlar benzer uzunluktadır.**
    - Tüm komutlar için fetch ve decode durumları benzer şekilde işlenir.
  - **Yalnızca birkaç komut formatları vardır.**
    - Basit bir şekilde, Komutların decodlanması ve tek durumda yapılması mümkün hale gelir.
  - ***Memory işlemleri sadece load/store komutlarında oluşur.***

  Böylece memory erişimi tam olarak bir sonraki duruma ertelenmiş olabilir
  - ***Operandlar (İşlenenler) memory'de dizili sıralıdır.***
    - Bir data transfer komutu yalnızca bir memory'e erişim durumuna ihtiyaç duyar.

MIPS'te Pipeline

- Neler zorlaştırır?

- *Structural hazards (Yapısal tehlikeler):* Pipeline işleminde, farklı komutlarda , farklı durumlarda aynı donanım kaynaklarının kullanılması durumu oluşabilir (Kaynak Çatışması).

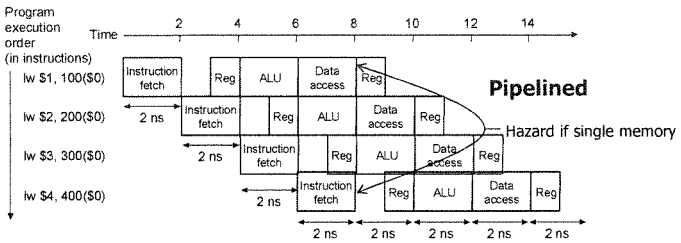
- *Control hazards (Kontrol tehlikeleri):* Bir önceki dallanma komutunun sonucuna bağlı olarak, izleyen komutu mevcut pipeline içine alma sorunu (Dallanma zorluğu).

- *Data hazards (Data tehlikeleri):* Pipeline'daki bir komutun, aynı pipelin'daki önceki komutun hesapladığı datayı kullanması gerektiğinde (Veri Beklentisi).

- Biz, pipeLine'lı datapath'ı inclemeden önce yukarıda sıralanan olası tehlikeleri tek tek incelemeliyiz.

Structural Hazards (Yapısal Tehlikeler)

- *Structural hazard* : Pipeline işleminde aynı clock cyclında bütün komutların eşzamanlı işlemesi için gereken donanımın yetersizliği.
- Örnek: Varsayınız ki, komut ve data hafızası tek olan, yani tekbir veriyolu kullanan, bir yapıda pipeline işlemi yapılsın.
  - O zaman 1. ve 4. lw komutları arasında bir structural hazard oluşabilir.

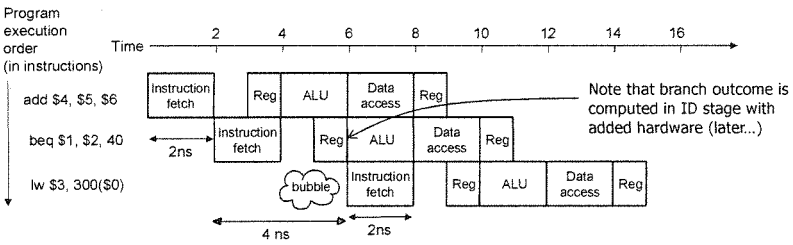


- *Korkmayın MIPS PİPELİNE'ı destekler şekilde dizayn edilmiştir. Yapısal tehlikelerden kolayca kaçınılır.*

Control Hazards

- *Control hazard (Kontrol Tehlikesi)*: PİPELİNE'in yürütülmesinde bir önceki komutun sonucuna göre karar vermek gerekebilir. Bu problem genellikle şartlı ve şartsız dallanma komutlarının işlenmesi durumunda çok önemlidir.

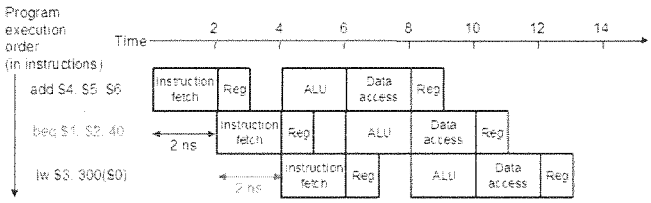
- <u>Çözüm 1</u> *Stall* the pipeline (Pipeline'ı geciktirme)



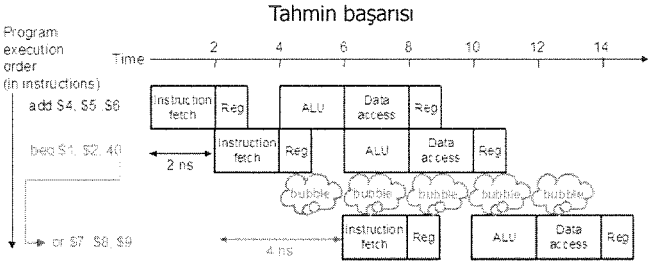
Pipeline stall (Pipeline gecikmesi)

Control Hazards

- Çözüm 2 *Predict* branch outcome (dallanma sonucunun önceden tahmin edilmesi veya dallanmanın olmayacağının tahmini)



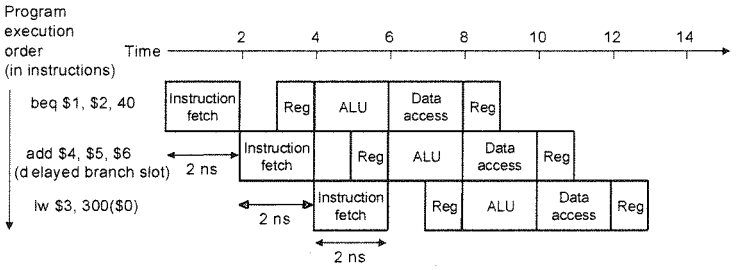
Tahmin başarısı



Tahmin hatası: lw'ü geri al (=flush)

Control Hazards (Kontrol Tehlikesi)

- Çözüm 3: Gecikmiş dallanma (*Delayed branch*) :Derleyici , dallanmaları yeniden düzenleyerek, zamanında pipeline içine olmalarını sağlar. Bu yöntemde derleyici dallanma komutundan önceki ve sonraki komutları analiz ederek geciktireceği komutların önüne faydalı komutlar yazarak program akışını yeniden düzenler. Yani dallanma komutuna sıra gelmeden önündeki komutları, akışı bozmamak şartıyla gecikme adımlarına alıp bu esnada dallanma komutunuda diğer kesimlerde işlemeye devam eder. Böylece dallanmanın gecikmesi önlenmiş olur.



Delayed branch `beq` is followed by `add` that is independent of branch outcome

Gecikmiş Dallanmaya Örnek:

5 komutluk bir program olsun. Komutlar sırayla;
1- Hafızadan R1'e yükle
2-R2'yi arttır.
3-R3 e R4'ü topla
4-R5'i R6'dan çıkar
5-X adresine dallan

Aşağıda derleyicinin yapabileceği 2 gerçekleştirme verilmiştir.

Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to X					I	A	E			
6. NOP						I	A	E		
7. NOP							I	A	E	
8. Instr. in X								I	A	E

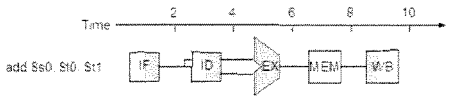
İşlem yok komutu kullanılarak

Clock cycles:	1	2	3	4	5	6	7	8
1. Load	I	A	E					
2. Increment		I	A	E				
3. Branch to X			I	A	E			
4. Add				I	A	E		
5. Subtract					I	A	E	
6. Instr. in X						I	A	E

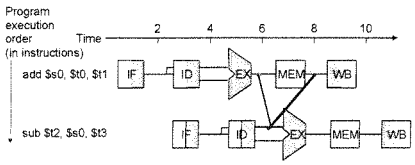
Komutları tekrar sıralayarak

Data Hazards (Veri Tehlikeleri)

- *Data hazard :* Bir sonraki komut, yürütmesi bitmemiş bir önceki komutun datasına ihtiyaç duyabilir.
- Çözüm : *Mümkünse datayı ilerlet....*



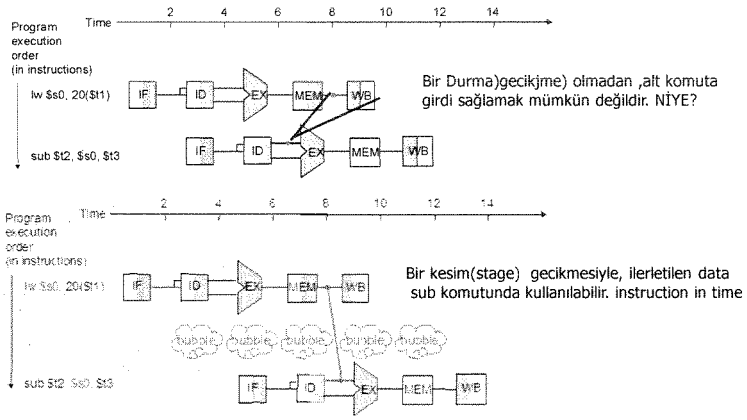
Komutun Pipeline diyagramı: shade indicates use – left=write, right=read



İlerletme yoksa – mavi çizgi – Veri zamanda geri gitmek durumunda. Yani 2.komut 6,8 arasında bekleyecek.; İlerletme varsa– kırmızı çizgi – Veri, zamanında hazır edilmiş. 2.komut 6,8 süresinde işlem yapar.

Data Hazards

- Datayı ilerletmeninde yetmeyeceği durumlar olabilir.
- Örneğin, R-tipi bir komut yükleme komutunu takip ediyorsa, yüklenen değeri kullanacaksa – *load-use data hazard'ı* olarak bilinir.



PİPELİNE gecikmesinden (Stall) kaçınmak için kodlamayı yeniden düzenleme.(Software Solution)

- Example:

```
lw $t0, 0($t1)
lw $t2, 4($t1)
sw $t2, 0($t1)
sw $t0, 4($t1)
```

Data hazard

- Reordered code:

```
lw $t0, 0($t1)
lw $t2, 4($t1)
sw $t0, 4($t1)
sw $t2, 0($t1)
```

Interchanged

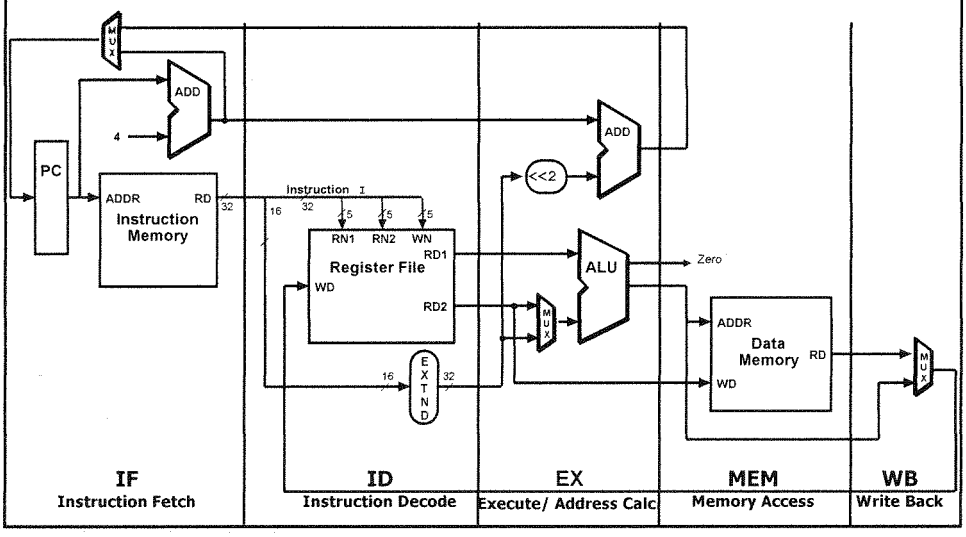
Pipeline çalışan Datapath

- Şimdi bir pipeline datapath yapısını açıklayabiliriz.
- İlk önce bir komutun tamamlanmasındaki 5 adımı hatırlayalım.

1. Instruction Fetch & PC Increment (IF)
2. Instruction Decode and Register Read (ID)
3. Execution or calculate address (EX)
4. Memory access (MEM)
5. Write result into register (WB)

- Hatırlatma: single-cycle işlemci için
  - 5 adımın hepsi birden single clock cycle'ında yapılır.
  - Herbir adım için atanmış bir donanıma (fonsiyonel yapıya) ihtiyaç vardır.
- *Biz single cyle datapath donanımını değiştirmeden, bir komutun işlenmesini birden fazla cycle'da gerçekleşecek şekilde parçalasaydık neler oluşurdu?*
- *What happens if we break the execution into multiple cycles, but keep the extra hardware?*

Hatırlatma - Single-Cycle Datapath "Steps"



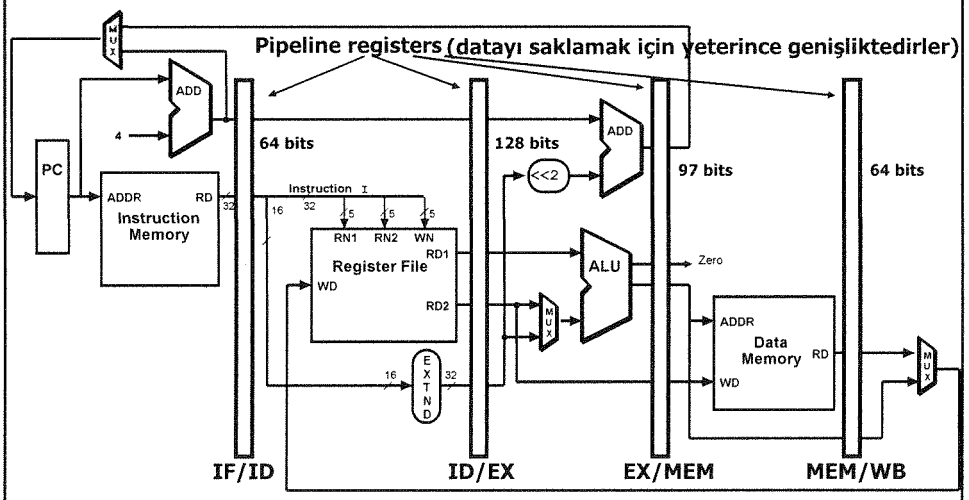
Pipelined Datapath – Temel Fikirler

- *Extra donanımları korumak şartıyla, bir yürütme işlemini bir çok cycle'a bölersek ne olur?*
- *What happens if we break the execution into multiple cycles, but keep the extra hardware?*
  - Cevap: Her bir clok çevriminde yeni bir komutu işlemeye başlayabiliriz.
  - *We may be able to start executing a new instruction at each clock cycle* - pipelining

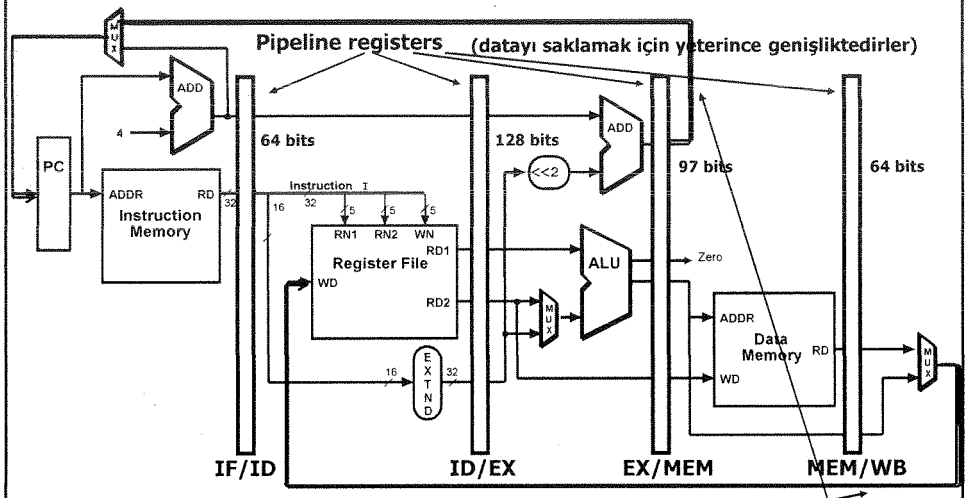
- İşte bu Pipeline işlemidir.
- Fakat; bu cycle'lar arasındaki süreçte (segment-Kesim) datayı tutmak için extra registerlara ihtiyaç vardır. Bunlar *pipeline register'larıdır.*

- *Önemli not: Pipeline yürütmede herbir kesimde(segmentte) elde edilen sonuç bir sonraki kesime aktarılabilir. Aynı kesimde farklı komutlar için eş zamanlı olarak farklı datalar ile farklı işlemler yapılabilir.*
  - *Dolayısıyla her kesim için bir pipeline register kullanılmalıdır.*

Pipelined Datapath

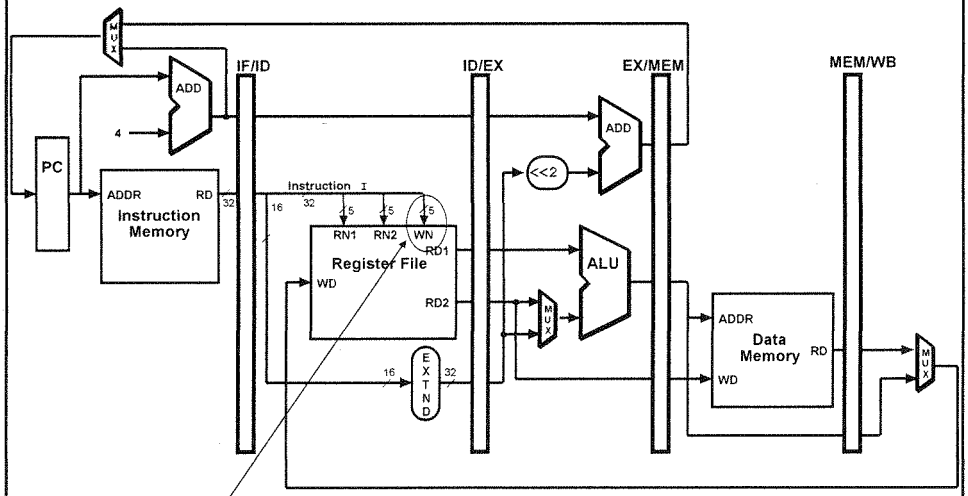


Pipelined Datapath



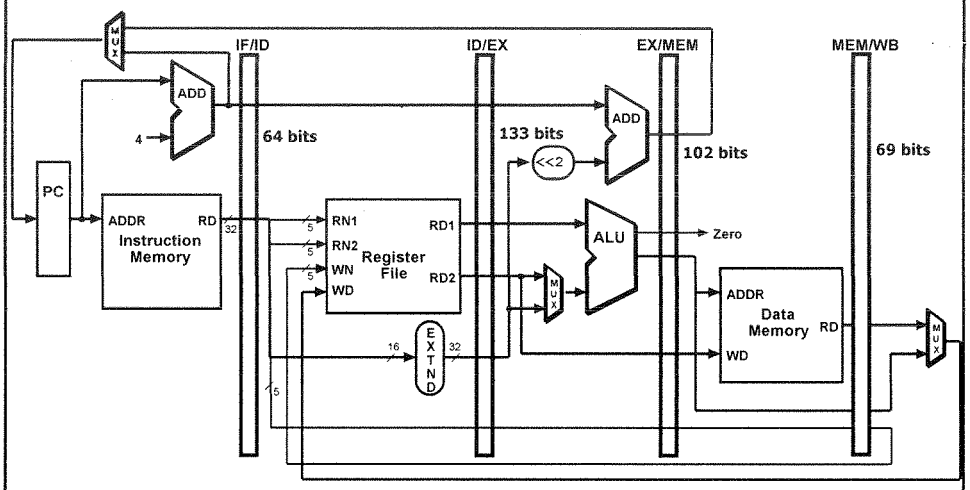
Sadece soldan sağa doğru akan veri durumunda tehlikeler oluşabilir? neden?

Datapath'de hatalar



Hedef reg'e yazılma süreci, sonraki bir komutun başka bir işlem fazından sonraya denk gelebilir.

Düzeltilmiş DATAPATH



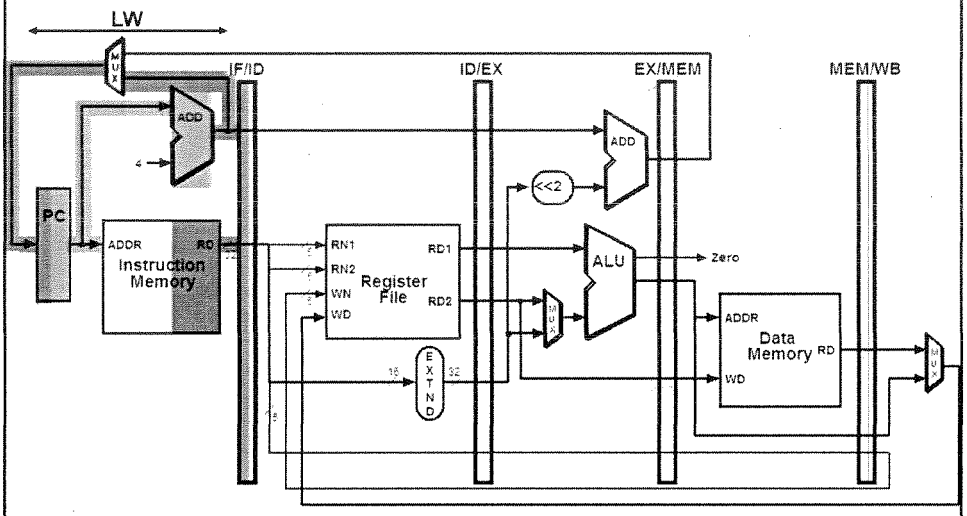
Hedef Reg'in adresi (5 bitlik) ID/EX, EX/MEM ve MEM/WB reg'lerinede yazılır. Şimdi bu reg'lerin genişlikleri 5 bit artmıştır.

Pipeline Örnekleri

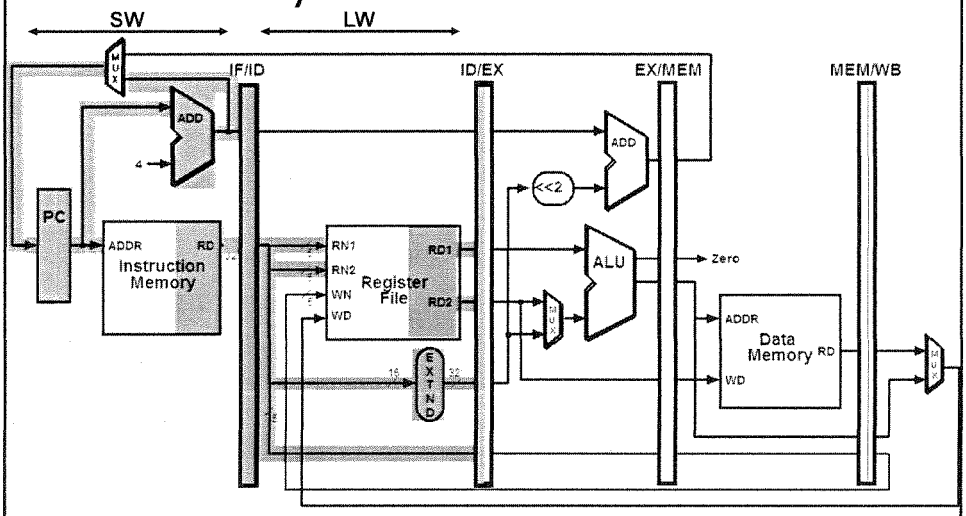
- Aşağıdaki komut dizisini gözönünde bulundur.

```
lw  $t0,  10($t1)
sw  $t3, 20($t4)
add $t5,  $t6,  $t7
sub $t8,  $t9,  $t10
```

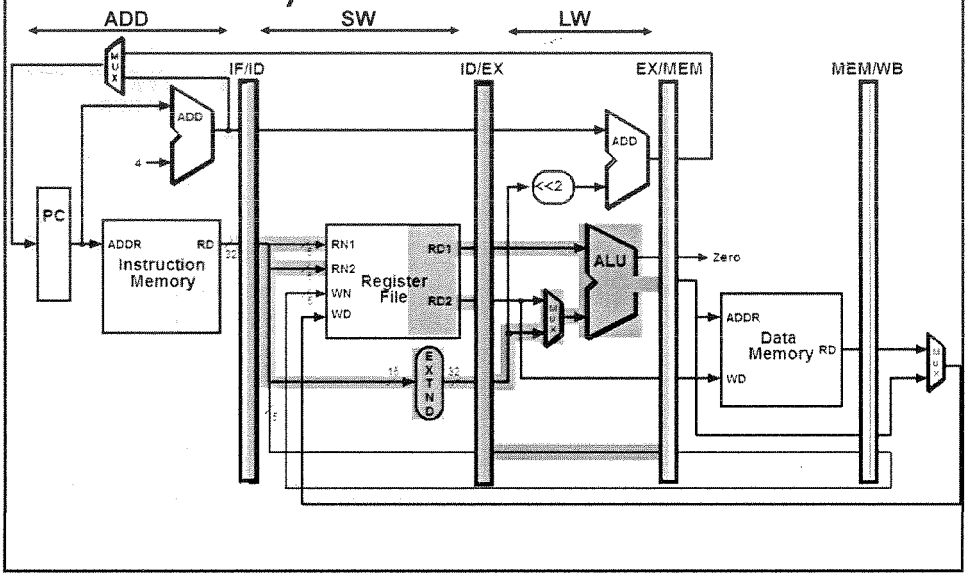
Single-Clock-Cycle Diagram: Clock Cycle 1



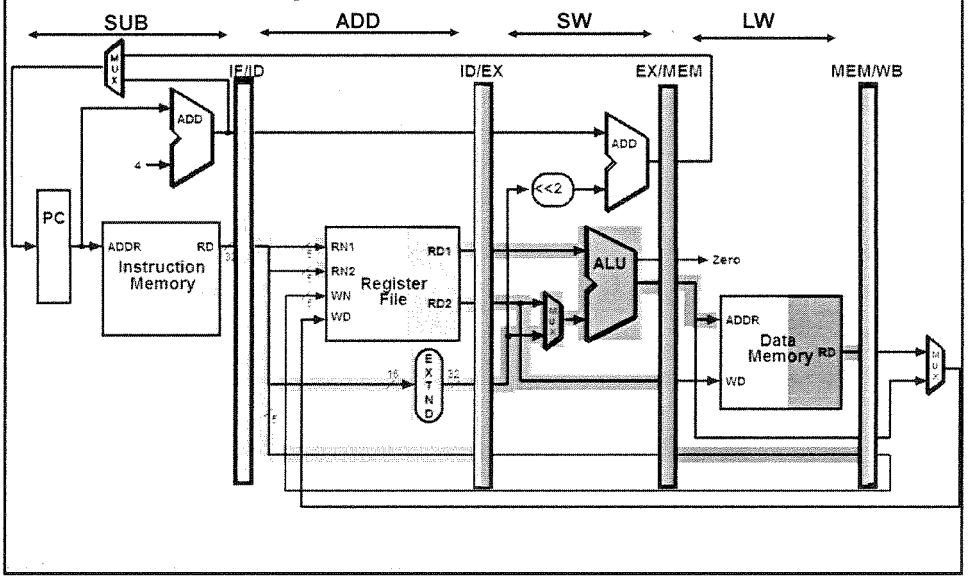
Single-Clock-Cycle Diagram: Clock Cycle 2



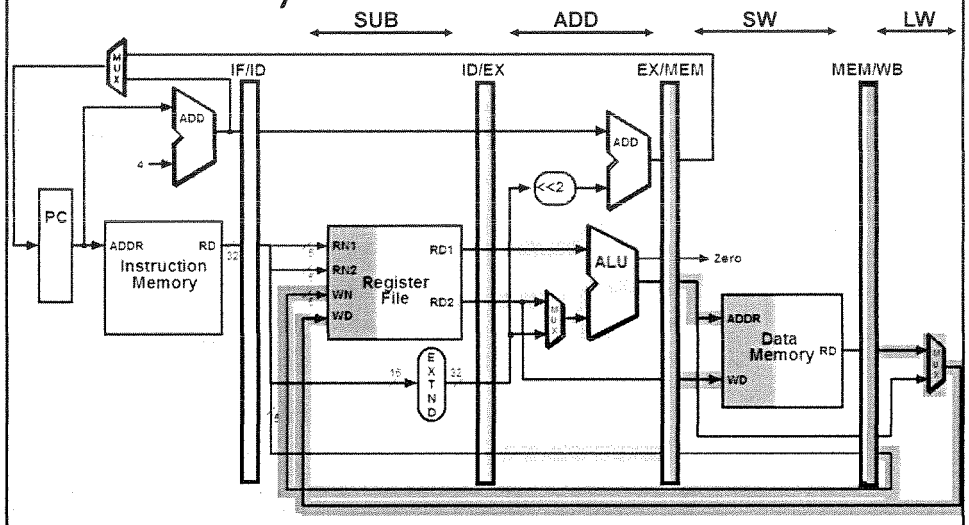
Single-Clock-Cycle Diagram: Clock Cycle 3



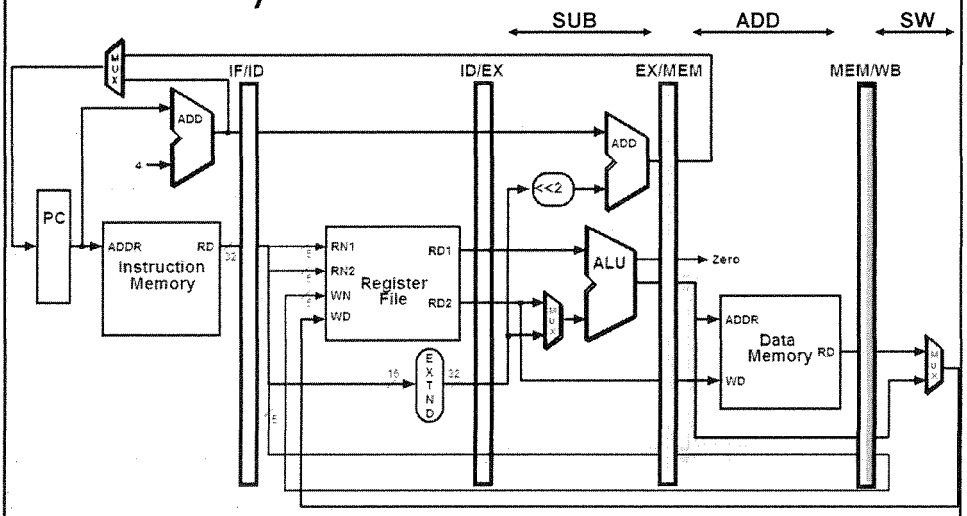
Single-Clock-Cycle Diagram: Clock Cycle 4



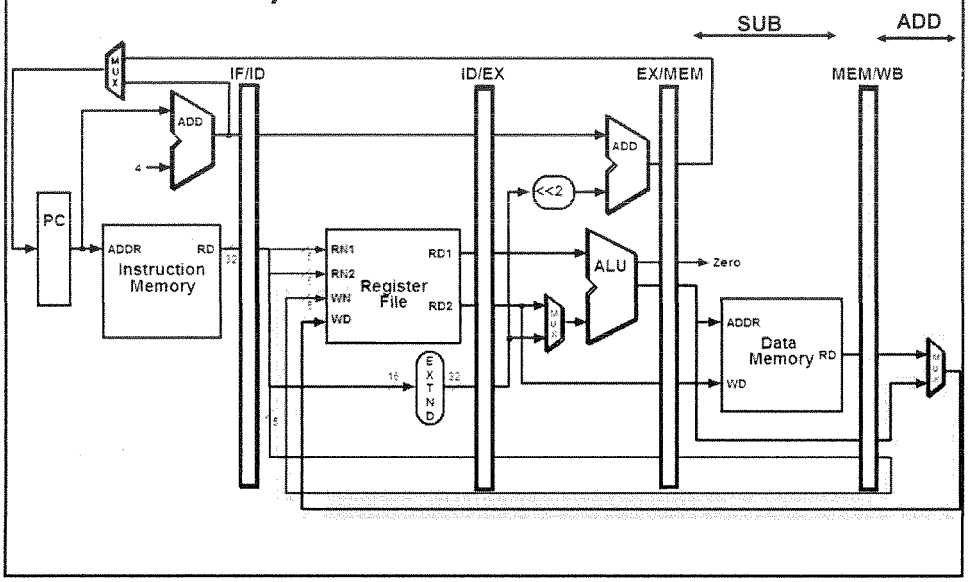
Single-Clock-Cycle Diagram: Clock Cycle 5



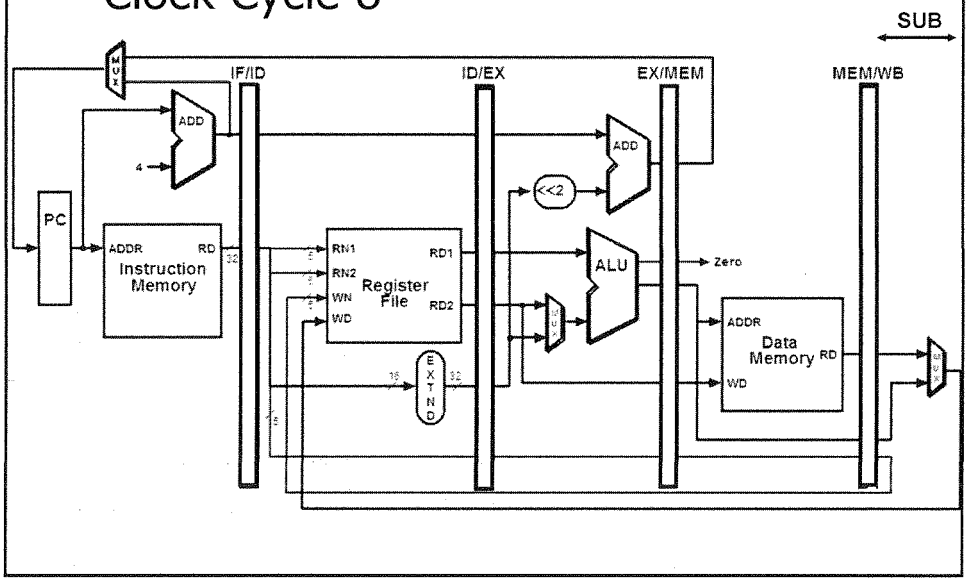
Single-Clock-Cycle Diagram: Clock Cycle 6



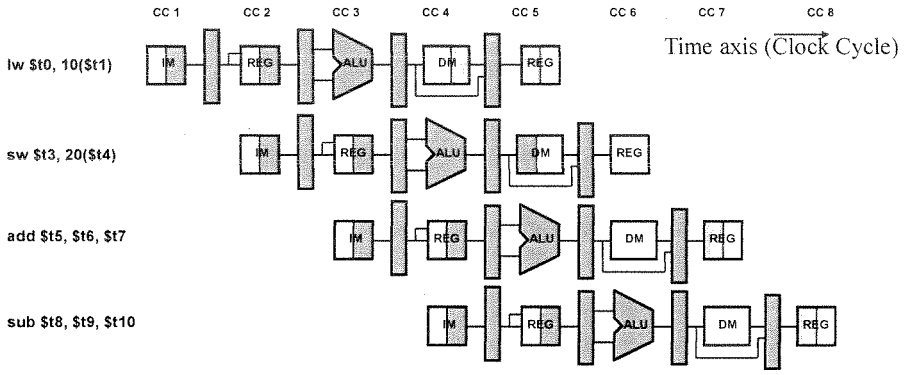
Single-Clock-Cycle Diagram: Clock Cycle 7



Single-Clock-Cycle Diagram: Clock Cycle 8



Alternative View – Multiple-Clock-Cycle Diagram



Not:

- Bir R-tipi komutunun yürütülmesi sürecinde; Multicycle ve pipeline gerçekleştirmeler arasında önemli fark:
  - R tipi bir komutta, registere write-back işlemi 5. pipeline kesimindedir (last write-back segmentindedir). Oysa multicycle gerçekleştirmede ise bu işlem 4. durumda gerçekleşir. Niyen?
  - Reg.file'a yazarken yapısal tehlikelerden dolayı olabilir.
- *Tekrar etmeye değer:* Boru hattı ve multicycle uygulamalar arasındaki temel fark, 5 durumu biribirinden ayrıştırmak için pipeline register yerleştirilmesidir.
- İdeal Pipeline (Duraksama-gecikme yosa) için CPI =1 'dir. Niye?

Simple Example: Comparing Performance

- *Compare performance for single-cycle, multicycle, and pipelined datapaths using the gcc instruction mix*
  - assume 2 ns for memory access, 2 ns for ALU operation, 1 ns for register read or write
  - assume gcc instruction mix 23% loads, 13% stores, 19% branches, 2% jumps, 43% ALU
  - for pipelined execution assume
    - 50% of the loads are followed immediately by an instruction that uses the result of the load
    - 25% of branches are mispredicted
    - branch delay on misprediction is 1 clock cycle
    - jumps always incur 1 clock cycle delay so their average time is 2 clock cycles

Simple Example: Comparing Performance

- *Single-cycle* (p. 373): average instruction time 8 ns
- *Multicycle* (p. 397): average instruction time 8.04 ns
- *Pipelined*:
  - loads use 1 cc (clock cycle) when no load-use dependency and 2 cc when there is dependency – given 50% of loads are followed by dependency the average cc per load is 1.5
  - stores use 1 cc each
  - branches use 1 cc when predicted correctly and 2 cc when not – given 25% misprediction average cc per branch is 1.25
  - jumps use 2 cc each
  - ALU instructions use 1 cc each
  - therefore, average CPI is

  $1.5 \times 23\% + 1 \times 13\% + 1.25 \times 19\% + 2 \times 2\% + 1 \times 43\% = 1.18$
  - therefore, average instruction time is $1.18 \times 2 = 2.36$ ns

Vektör İşlemleri

Birçok yazılım uygulaması (Görüntü işleme, insan genetiği, Sismik veri analizi v.b), büyük sayı dizileri üzerinde çok büyük sayıda aritmetik işlem yapmayı gerektirir. Bu sayılar çoğu zaman Floating Point sayı düzlemindeki tek boyutlu veya çok boyutlu diziler şeklinde toplanmışlardır. . Sıradan bilgisayarlar bu çalışmaları çok uzun sürede sonlandırabilir. Sayılan işlemlerin yapılabilmesi için iyi kaliteli (vektör ve paralel işleme özelliği olan) bilgisayarlara gerek vardır.

Vektör: Tek Boyutlu veri dizisidir.

Satır veya sütün vektörleri şeklinde ifade edilebilirler. V = [V1 V2 V3.....Vn] . Elemanları V olan bir satır vektörü V(I) şeklinde I indisi ile gösterilir ve bir hafızada dizi şeklinde saklanır.

Bir vektör olmayan işlemci ile vektör işlemci arasındaki farkı anlamak için aşağıdaki döngüyü göz önüne alalım.

```
      DO 20 I = 1, 100
20    C(I) = B(I) + A(I)
```

- Conventional computer

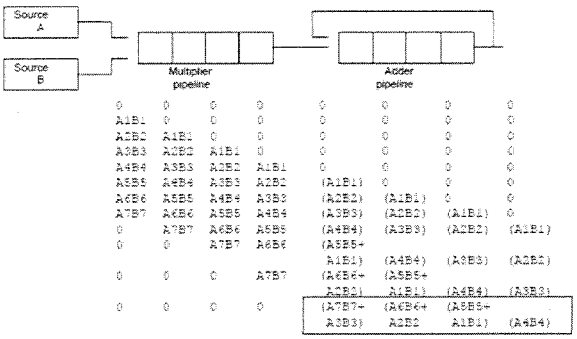
```
      Initialize I = 0
20    Read A(I)
      Read B(I)
      Store C(I) = A(I) + B(I)
      Increment I = I + 1
      If I ≤ 100 goto 20
```

- Vector computer

```
C(1:100) = A(1:100) + B(1:100)
```

Operation code	Base address source 1	Base address source 2	Base address destination	Vector length

Vektör işlemci için komut formatı



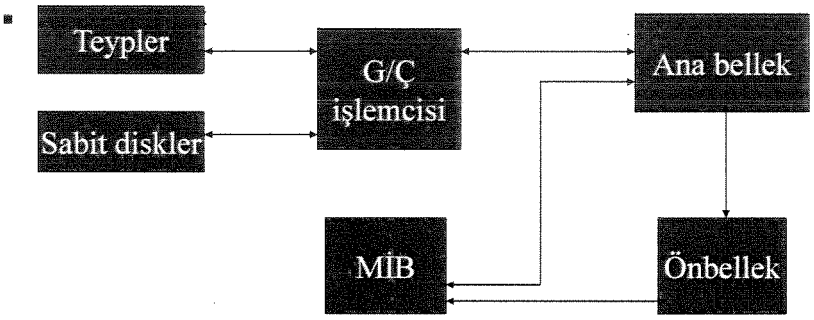
İç çarpım hesabı için PİPELiNE

BELLEK TASARIMI

Bellek Hiyerarşisi
Ana Bellek
Yardımcı Bellek
İçerik adreslemeli Bellek
Ön Bellek
Sanal bellek
Bellek Yönetim Donanımı

Bellek Hiyerarşisi

- Bellek herhangi bir bilgisayarın temel bileşenlerinden biridir.Veri ve programlar burada tutulur.
- MİB ile doğrudan haberleşebilen belleğe ana bellek denir.Yedekleme görevi yapan aletlere ise yardımcı bellek denir.(sabit disk ve teypler).



Bir bilgisayar sisteminde bellek hiyerarşisi

- Bellek hiyerarşisi yavaş geniş yardımcı bellek,daha hızlı ana bellek ve küçük fakat hızlı ön bellekten oluşur.Bu hiyerarşinin en altında manyetik teypler vardır.
- Ön bellek çok hızlı bir bellek parçasıdır.Ön bellek ana belleğe erişim zamanı ile MİB arasındaki hız farkını ortadan kaldırmak için kullanılır.

- G/Ç işlemcisi yardımcı bellek ile ana bellek arasında veri aktarımı yaparken ön bellek de ana bellek ile MİB arasında veri aktarımını gerçekleştirir.
- İşletim sistemlerinde MİB birbirinden bağımsız programları bitişik zamanda icra eder.Bu işlem çoklu programlamadır.

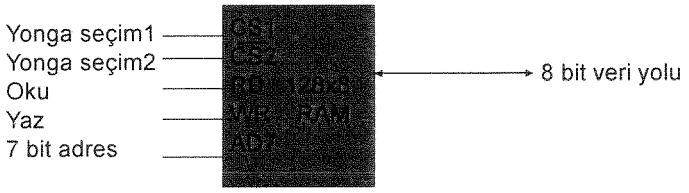
Ana Bellek

- Bilgisayar sisteminin merkezi depolama birimidir.
- RAM bellekte belleğin herhangi bir yerinde bulunan verilere bir sıra takip edilmeksizin yani rasgele ve aynı sürede erişilir. RAM'de saklanan bilgiler değiştirilebilir. RAM'deki bilgilere erişim, disk ya da disket sürücülerindeki erişimle karıştırılmayacak kadar hızlıdır.

☻ Programların kalıcı olarak durduğu sadece okunabilen bellek tipidir. Veri sadece ROM bellekten elde edilebilir. Hiçbir bilgi ROM belleğe yazılamaz. ROM yapımcı veya kullanıcı tarafından bir daha değiştirilmemek üzere konulan program komutlarını içerir. Örneğin BIOS ROM belleğe konulur. ROM bellek uçucu (non-volatile) değildir. Yani bilgisayar sisteminin enerjisi kesildiği zaman ROM'da depolanan bilgi kaybedilmez.

Ram ve Rom Yongaları

- Ram yongaları MİB ile haberleşmek için tasarlanmıştır.
- Ram yongasında çift yönlü veri yolu vardır.Veriyi okuma işlemiyle MİB'ye veya yazma işlemiyle MİB'den belleğe aktarır.
- İki yönlü veri yolu 3 zamanlı buffer larla yapılır.

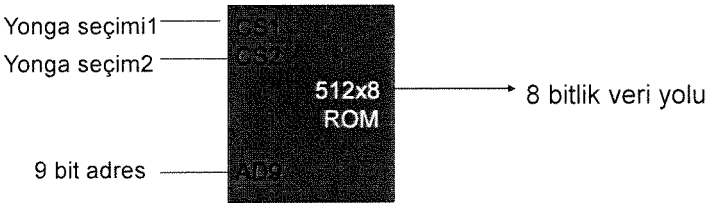


a)Blok şeması

CS1	$\overline{CS2}$	RD	WR	Bellek fonksiyonu	Veri yolu durumu
0	0	x	x	Tutma	Yüksek direnç
0	1	x	x	Tutma	Yüksek direnç
1	0	0	0	Tutma	Yüksek direnç
1	0	0	1	Yaz	RAM'a giriş bilgisi
1	0	1	x	Oku	RAM'den çıkış bilgisi
1	1	x	x	tutma	Yüksek direnç

b)Fonksiyon çizelgesi

Tipik RAM yongası



Tipik ROM yongası

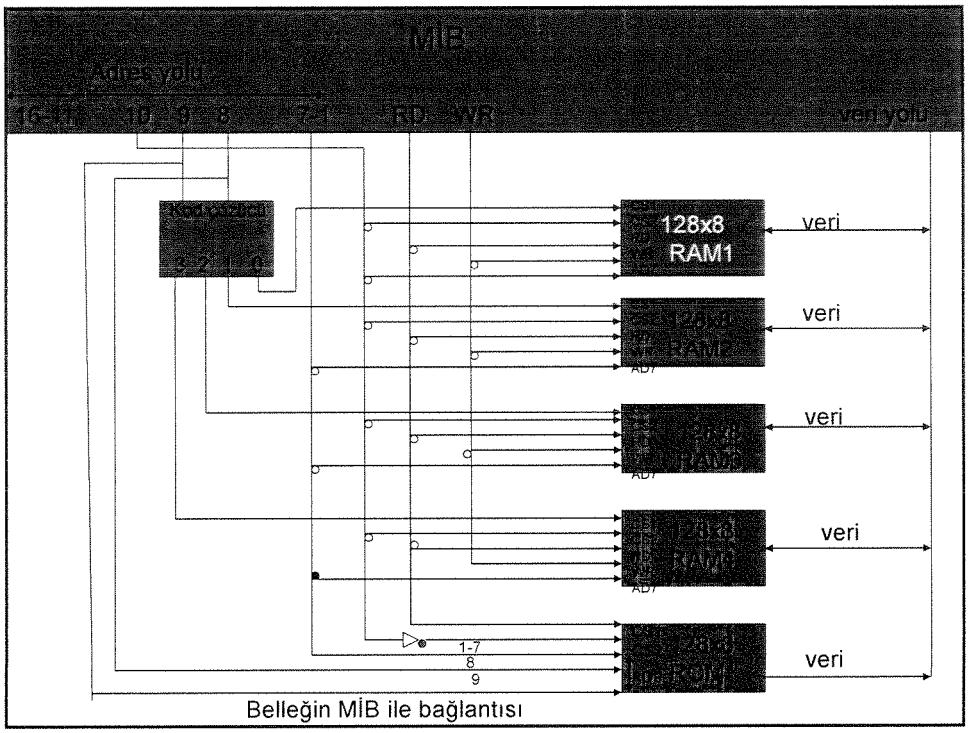
ROM sadece okunabilirdir.Veri yolu çıkış kipindedir.Aynı büyüklükteki ROM yongası RAM yongasından daha fazla bit içerir.

Mikrobilgisayar için bellek adres haritası

		Adres yolu										
Bileşen	onaltılık adres	10	9	8	7	6	5	4	3	2	1
RAM1	0000-07F	0	0	0	x	x	x	x	x	x	x
RAM2	0080-07F	0	0	1	x	x	x	x	x	x	x
RAM3	0100-17F	0	1	0	x	x	x	x	x	x	x
RAM4	0180-1FF	0	1	1	x	x	x	x	x	x	x
ROM	0200-3FF	1	x	x	x	x	x	x	x	x	x

Bellekle MİB Bağlantısı

- Ram ve Rom yongaları MİB'e veri ve adres yollarıyla bağlanmıştır.
- Adres yolundaki düşük önemdeki bitler yonga içindeki byte seçimi için diğer bitler yonga seçimi için kullanılır.



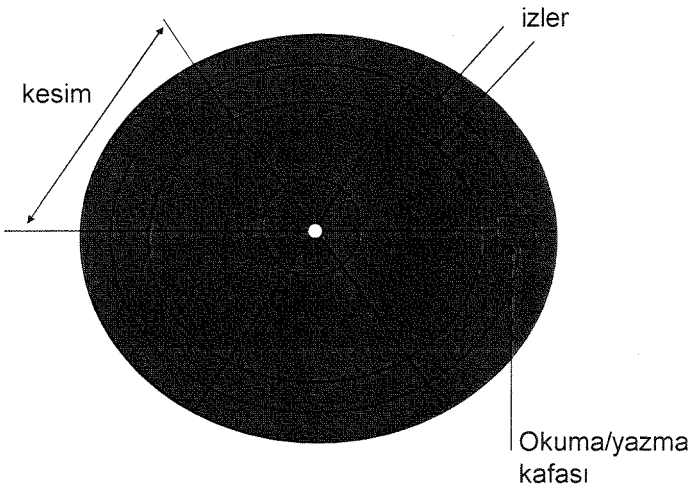
Belleğin MİB ile bağlantısı

Yardımcı Bellek

- Bilgisayar sistemlerinde en çok kullanılan yardımcı bellekler sabit disk ve teyplerdir.
- Bellekte bir adrese erişmek ve onun içeriğini almak için geçen ortalama süreye erişim süresi denir.

Manyetik Disk ve Teypler

- Manyetik disk dairesel bir tabakadır.Manyetik alanlı plastik bir madde ile kaplanmıştır.
- Disk sistemi adres bitleri ile belli olur.
- Manyetik teypler durdurulabilir ileri veya geri sarılabilir.

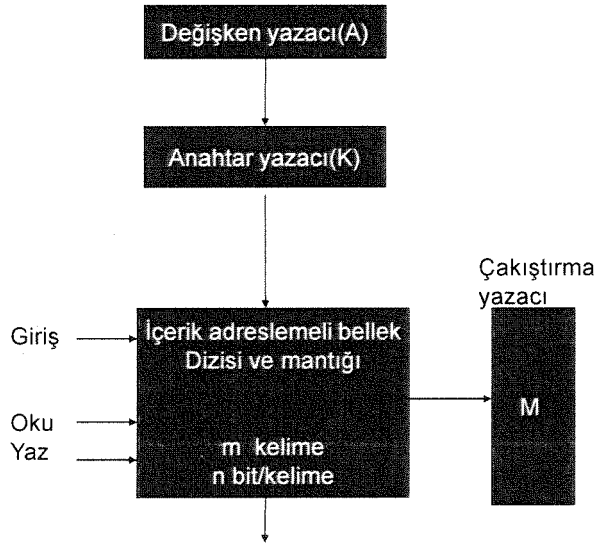


Manyetik Disk

İçerik Adreslemeli Bellek

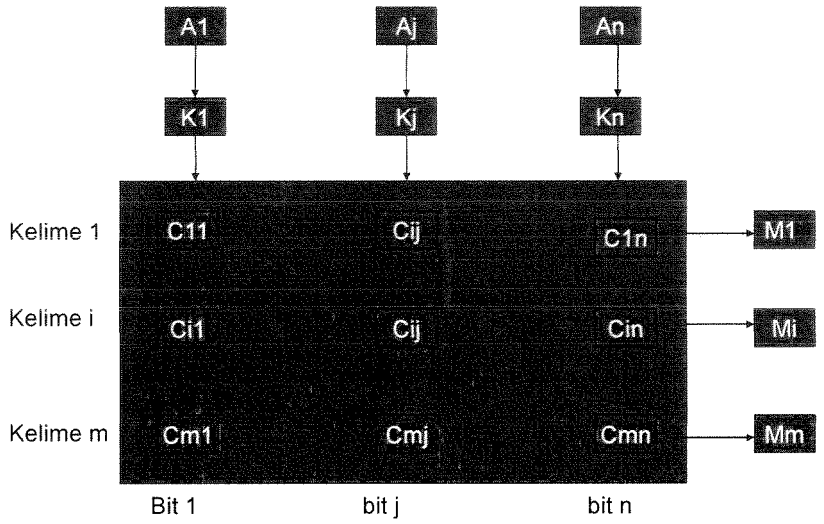
- Bellekte bulunan bir değerin aranıp bulunması ,adres yerine verinin içeriği ile olursa arama süresi çok kısaltılabilir.Buna içerik adreslemeli bellek CAM denir.
- Bu belleğe aynı anda ve paralel olarak erişilebilir.

Donanım Tasarımı



İçerik adreslemeli bellek şeması

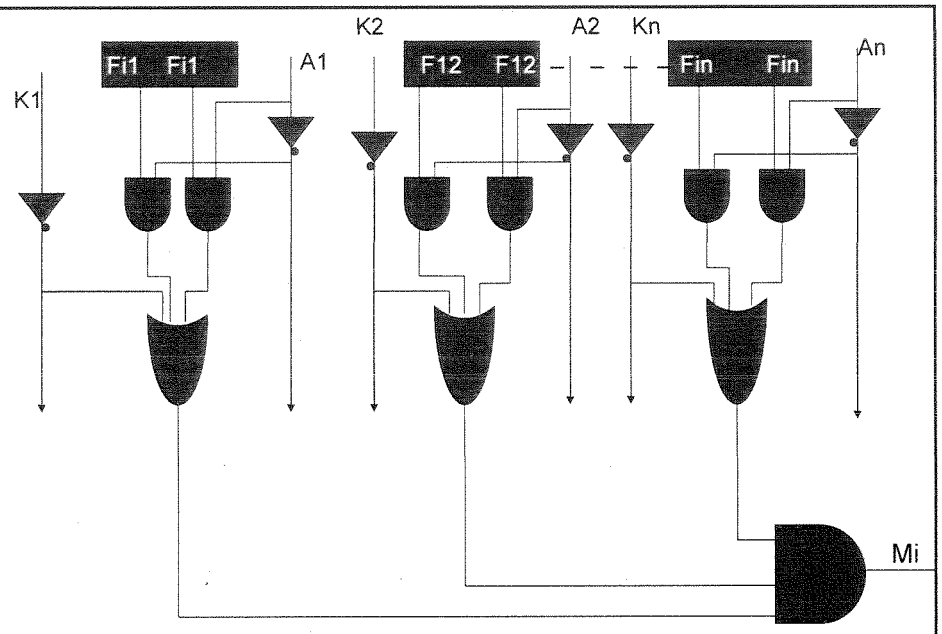
Her kelime n bitlik içerik adreslemeli bellek



Bellek dizisi ile dış registerlar arasındaki bağlantı.Dizi içindeki hücreler C ve iki indis ile gösterilmiştir.Birinci indis kelime numarasını ,ikinci indis bit numarasını gösterir.

Çakıştırma Mantığı

- Her bir kelime için karşılaştırma algoritmasından türetilir.
- Önce anahtar bitler ihmal edilerek A içindeki değişken hücrelerdeki kelimelerle karşılaştırma yapılır.

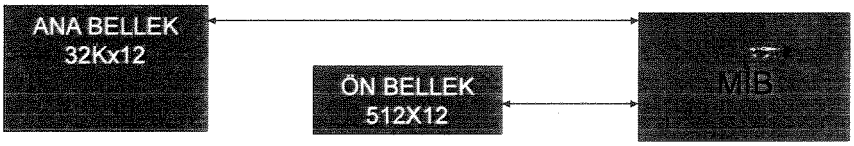


İçerik adreslemeli bellekte bir kelime için çakıştırma mantığı

Ön Bellek

- Ön-bellekler, işlemci tarafından bellek işlemlerinin hızlandırması için tasarlanmış özel yüksek hızlı belleklerdir. İşlemci, ön-bellekte bulunan komut ve verilere, ana bellekte bulunan komut ve verilere göre çok daha hızlı ulaşabilir.
- Ön-bellekler, birincil ön-bellek (Level 1, L1) ve ikincil ön-bellek (Level 2, L2) olarak ayrılırlar.

Haritalama



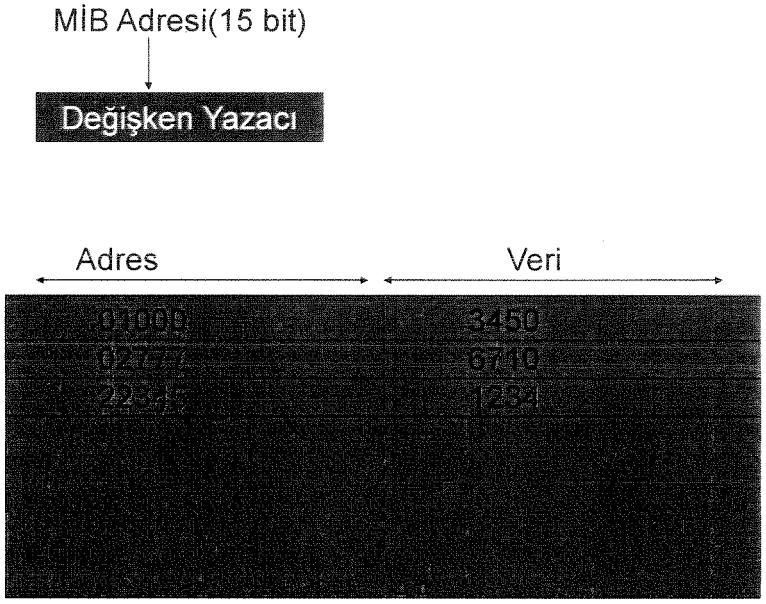
Ön bellek örneği

Ana bellekten ön belleğe verilerin aktarılmasına haritalama denir.3 tip haritalama işlemi kullanılır.

- İçerik adreslemeli haritalama
- Doğrudan Haritalama
- Kümelenmiş içerik adreslemeli haritalama

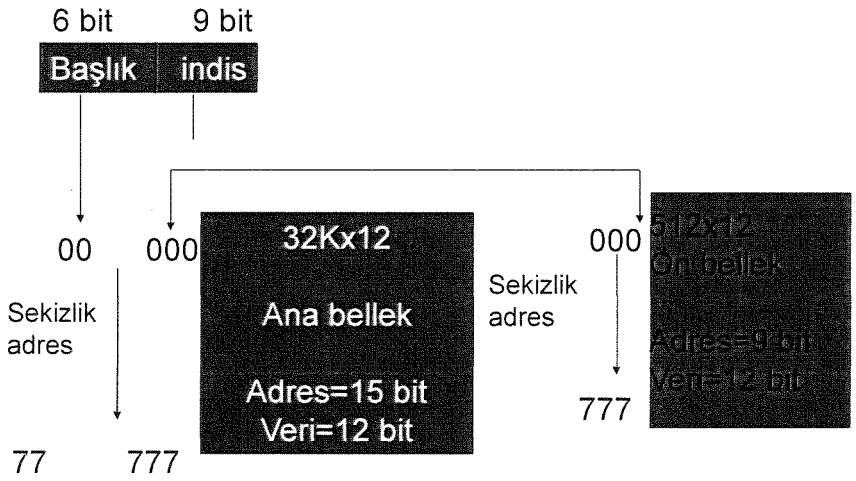
İçerik Adreslemeli Haritalama

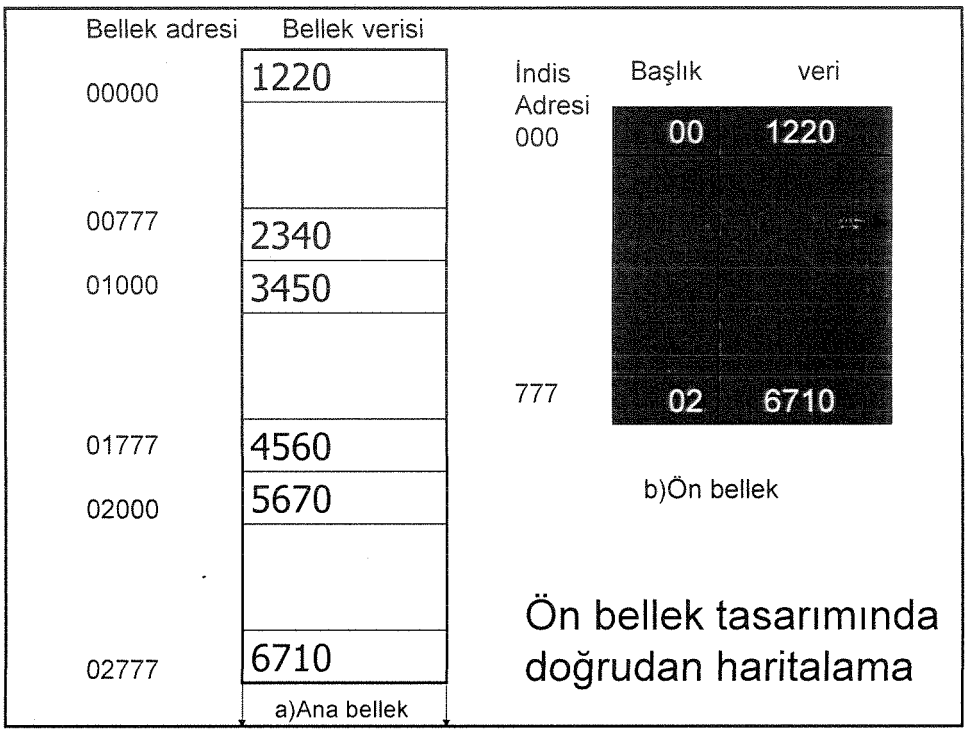
- En hızlı ve çok yönlü kullanılabilen ön bellek tasarımı içerik adreslemeli (CAM) bellek alanıdır.CAM bellek hem adresi hem veriyi tutar.



Önbelleğin içerik adreslemeli haritalanması

Ana ve ön bellek adresleri arasındaki ilişki





Ön bellek tasarımında doğrudan haritalama

Kümelenmiş İçerik Adreslemeli Haritalama

- Direk adreslemenin geliştirilmiş şeklidir.
- Aynı ön bellek kelimesi içinde birden fazla aynı indis adresli bellek kümesi bulunabilir.
- Her bir veri kelimesi başlığı ile birlikte depolanır ve bir küme oluşturur.

İndis	Başlık	Veri	Başlık	Veri
000	01	3450	02	5650
777	02	6710	00	2340

İki yollu kümelenmiş içerik adreslemeli haritalamalı ön bellek

Ön Belleğe Yazma

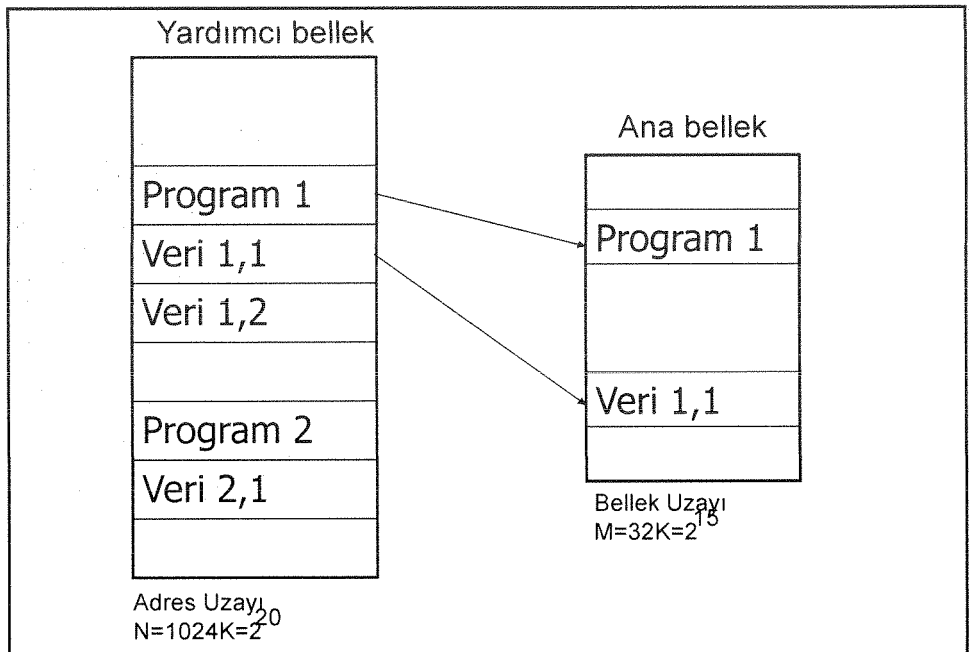
- Yazma işlemi iki şekilde yapılır.
  - Her yazma isteminde ana belleğe yazmak.Bu yazma işlemi doğrudan doğruya yazmadır.
  - Geri yazma yöntemi.Burada yazma sadece ön belleğe yapılır.

Sanal Bellek

- Sanal bellek büyük bilgisayar sistemlerinde kullanılan bir kavramdır.
- Kullanıcı tarafından kullanılan adrese sanal adres denir.Ana bellekteki bir adrese fiziksel adres denir.Bunların kümesine bellek uzayı denir.
- Bellek uzayı gerçek bellek adresleri olup işlemleriçin doğrudan verilmeli.

☺ Çok programlı bir bilgisayar sisteminde programlar ve veri yardımcı bellek ile ana bellek arasında aktarılır.

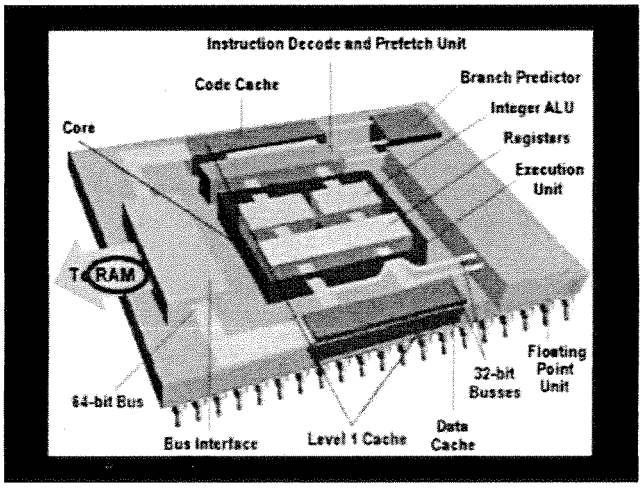
☺ Program ve veri bölümleri,bellekte birbirine komşu ve birbirini takip eden adreslerde bulunmazlar.



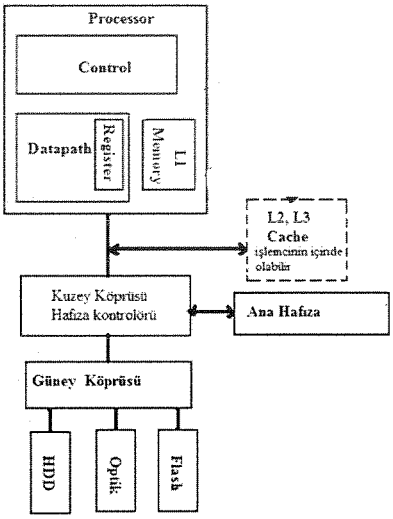
Sanal belleklerde adres ve bellek uzayı

HAFIZA Hiyerarşisi

Bilgisayar organizasyonu dersinde, işlemcilerin (Mikroişlemi-Mıcroprocessor) çalışma prensiplerini Fonksiyonel blok yapısı yaklaşımıyla anlatmaya çalıştık.



Bilgisayarların önemli ünitelerinden biriside Hafızalar (Memory)dır.



Yardımcı Hafızalar
sadece hafıza birimleri gösterilmiştir.

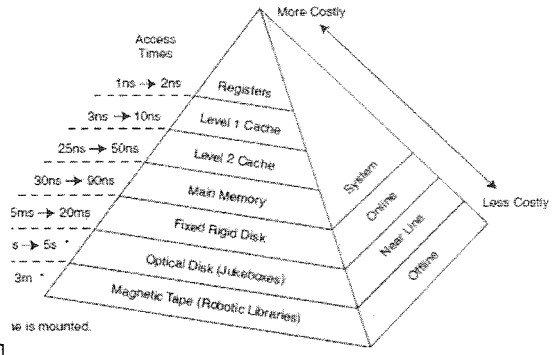
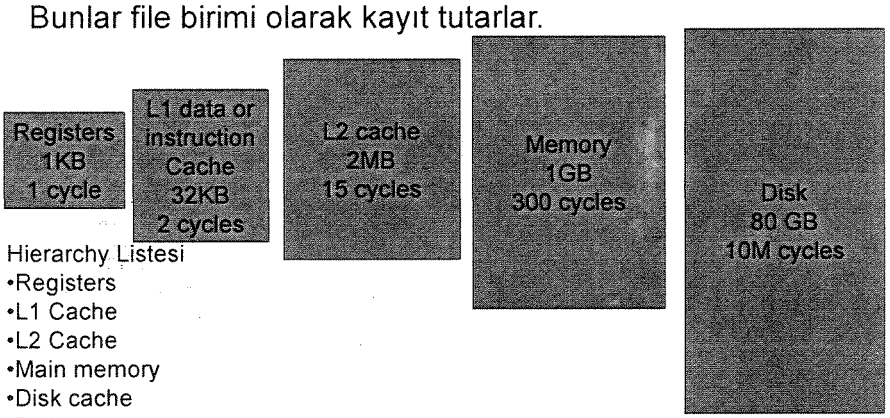


FIGURE 6.1 The Memory Hierarchy

Hafıza hiyerarşisi

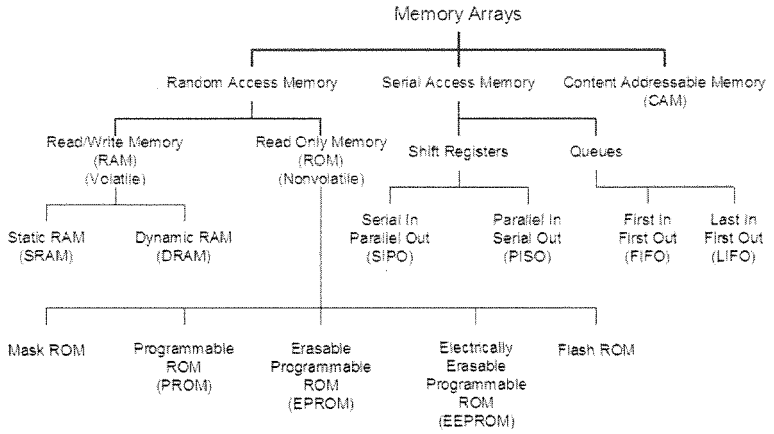
Hiyerarşinin en altında mekanik parçaları olan kalıcı (ikincil – Yardımcı) HD, Manyetik teyp v.b hafızalar bulunur. Bunlar file birimi olarak kayıt tutarlar.



Hierarchy Listesi
- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

4

Yarıiletken Hafıza çeşitleri
(Ana hafıza ve Cache hafızalar yarı iletken hafızalardır)



Bellek (Hafıza – Memory) Terimleri

Kapasite : Bellek aygıtının bir parçasında veya bellek sisteminin tümünde ne kadar bitin depolanabileceğini belirtmek için kullanılır. 8192 tane 8-bit kelime depolayacak şekilde düzenlenmiş bir bellek birimi 65536-bit veri depolar ve bellek kapasitesi 8192 x 8-bit veya 8192-Bayt olarak gösterilir.

Adres : Kelimenin bellek içindeki yerini tanımlar. Bellek içinde bulunan her bir kelimenin tek bir adresi vardır.

Okuma İşlemi : Bellek biriminin içindeki belirli bir adresteki veri kelimesinin okuma komutu alındıktan sonra bellek dışına transfer edilmesidir. Bellek içindeki bilgi değişmez, korunur.

Yazma İşlemi : Bellek biriminin dışında bulunan veri kelimesinin yazma komutu alındıktan sonra bellek içindeki belirli bir adresteki veri kelimesine transfer edilmesidir. Bellek birimine veri depolamak anlamına gelir. Bu durumda, bellek adresinde önceden bulunan veri üzerine yazıldığı için kaybolur.

Erişim Süresi : Bellek hızını belirleme için kullanılan özel bir terimdir. Bellek biriminden okuma işlemi yapmak için gerek duyulan süredir.

Dikkat!!!!!! İşlemciler genellikle Bellek erişim süresinden daha hızlı çalışırlar

Example 3-1

A given memory chip has 12 address pins and 4 data pins. Find:
(a) the organization (b) the capacity

Solution:

(a) This memory chip has 4096 locations ($2^{12} = 4096$), and each location can hold 4 bits of data. This gives an organization of 4096 x 4, often represented as 4Kx4.
(b) The capacity is equal to 16K bits since there is a total of 4K locations and each location can hold 4 bits of data.

Example 3-2

A 512K memory chip has 8 pins for data. Find:
(a) the organization (b) the number of address pins for this memory chip

Solution:

(a) A memory chip with 8 data pins means that each location within the chip can hold 8 bits of data. To find the number of locations within this memory chip, divide the capacity by the number of data pins. 512K/8 = 64K; therefore, the organization for this memory chip is 64Kx8.
(b) The chip has 16 address lines since $2^{16} = 64K$.

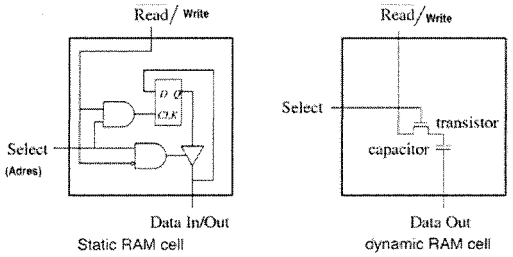
Yarı İletken Bellek Özellikleri

Bellek Tipi	Özellikleri	Okuma/ Yazma	Kalıcı	Hızı	Fiyat/bit
Flip-Flop	Bir bit yazmaç. Genellikle sayısal devrelerde temel tasarım bloğu olarak kullanılır.	Evet	Hayır	Çok çok Hızlı	Çok Yüksek
Yazmaç Register	Bayt, kelime veya çift kelime tutan FF setidir. Mikroişlemciler gibi karmaşık yongaların içinde kullanılır.	Evet	Hayır	Çok çok Hızlı	Çok Yüksek
SRAM	Adreslenebilir FF dizisidir. Verinin geçici olarak saklanması için kullanılır.	Evet	Hayır	Çok Hızlı	Yüksek
DRAM	Adreslenebilir depolama hücre dizisidir. Ana belleklerde veri saklamak için kullanılır.	Evet	Hayır	Hızlı	Orta
ROM	Adreslenebilir donanımla bağlanmış hücre dizisidir. Programlaması yonga üretimi sırasında yapılır.	Hayır	Evet	Çok Hızlı	Düşük
PROM	Adreslenebilir sigorta dizisidir. Programlaması kullanıcı tarafından bir kez yapılır.	Bir kez yazılır	Evet	Çok Hızlı	Yüksek
EPROM	Silinebilir ve yazılabilir ROM. Silme işlemi özelliklı morötesi (ultravıolet) ışıması ile yapılır.	Birden çok yazılır	Evet	Orta	Orta
OTPROM	Bir kez programlanabilir ROM. Temelde EPROM 'a benzer fakat penceresi yoktur.	Bir kez yazılır	Evet	Orta	Orta
EEPROM	Elektrik ile silinebilir ROM. Yazma çevrimi sayıları sınırlıdır	Evet	Evet	Düşük	Yüksek
FLASH ROM	Bölgesel silinebilir ROM. Yazma çevrimi sayıları sınırlıdır	Evet	Evet	Orta	Orta
NOVRAM	Batarya destekli SRAM veya SRAM/EEPROM melez teknoloji	Evet	Evet	Orta	Yüksek

RAM-SRAM-DRAM

Bizim ilgileneceğimiz hafızalar, Cache ve Ana hafıza birimleridir. Bunlar yarıiletken bellek yapılarıdır. SRAM genellikle Cache bellek, DRAM ana hafıza birimlerinde kullanılır. Bir bitlik memory Cell'lerin birleştirilmesiyle oluşurlar.

	Tran. per bit	Access time	Persist?	Sensitive?	Cost	Applications
SRAM	6	1X	Yes	No	100x	cache memories
DRAM	1	10X	No	Yes	1X	Main memories, frame buffers



Cache (Ön-bellek)

Cache (Ön) Bellek : Sistem belleğinden (Ana hafızadan) gelen veriler çoğunlukla İşlemcinin (CPU) hızına yetişemezler .Bu problemi çözmek için CPU içinde yüksek hızlı hafızalar bulunur.

• Ön-bellek çalışmakta olan programa ait komutların ve verilerin geçici olarak saklandığı yüksek hızlı hafızalardır.
•İşlemci, ön-bellekte bulunan komut ve verilere, Ana Bellekte bulunan komut ve verilere göre çok daha hızlı bir şekilde ulaşabilir.

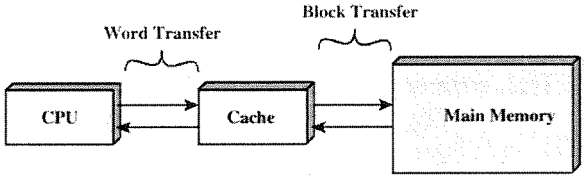
•Cache bellekler, program ve verilerin en yoğun kullanılan kısımlarını tutar.

• Cache bellekler, birincil ön-bellek (Level 1, L1) ve ikincil ön-bellek (Level 2, L2) ve son olarak üçüncül ön-bellek (Level3, L3) olarak ayrılırlar.

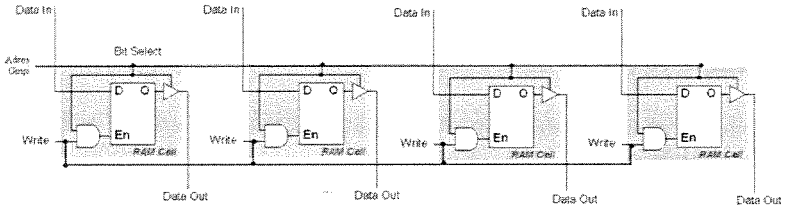
•İşlemcinin Cache belleğe ve ana belleğe doğrudan erişim yolu vardır.

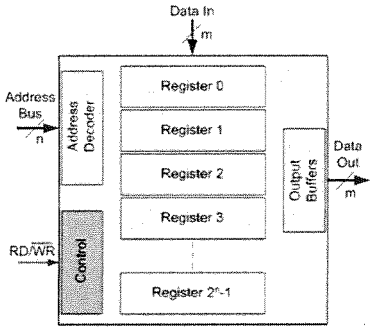
•İşlemci tarafından o anda kullanılmayan büyük verilerin tutulduğu ikincil (yardımcı) belleklere işlemcinin doğrudan bağlantısı yoktur.

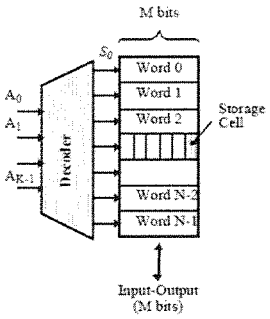
- Harici ön-bellekler işlemcinin dışındadır.
- Dahili ön-bellekler işlemcinin içindedir. Dahili ön bellek, CPU'nun en son işlediği verilerin bir kopyasını saklar.
- Aranan veri dahili ön bellekte bulunamazsa harici ön belleğe bakılır.
- Harici ön bellekte de bulunamayan veriler ana bellekten alınır.Bu,önbelleğe göre daha yavaş bir süreçtir.
- İşlemci cache ile word bazında, Ana hafıza Cache ile blok data olarak alış veriş yapar.

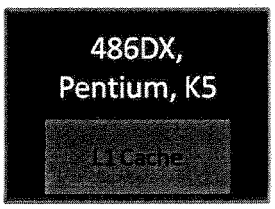


Register blok yapısı



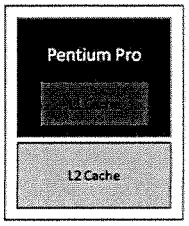


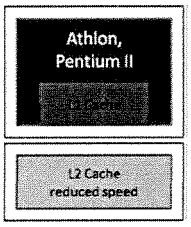


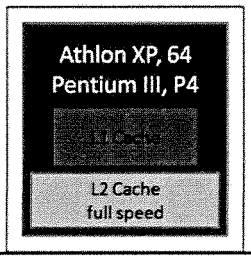


L1 Cache bellekler işlemcinin içinde olup, 486DX işlemcilerden bu yana kullanılıyor.
Şu an için AMD işlemciler çekirdek başına standart olarak 64 KB L1 önbelleklerle gelirken Intel işlemciler 32 KB veri ve komut L1 Cache'lere sahipler.

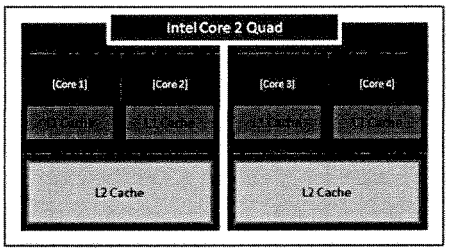
İkinci seviye önbellekler (L2) Pentium III işlemcilerden sonra genel olarak kullanılmaya başladılar (Daha önce Pentium Pro da bu önbellek vardı ancak çekirdeğin içinde değildi.Dolayısıyla çekirdek hızı ile çalışmıyordu).
180 Nm üretim sürecinin geliştirilmesiyle L2 önbellekler en sonunda çekirdeğin içine girdiler ve aynı hızda çalışmaya başladılar.



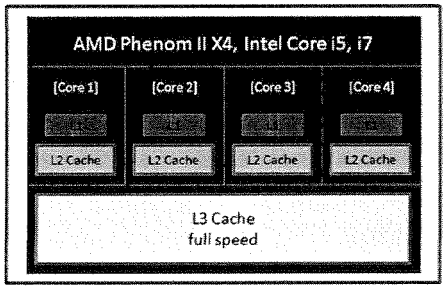




Günümüzde işlemciler çekirdeğe tümleşik 6 MB L2 önbelleğe sahip olarak geliyorlar.



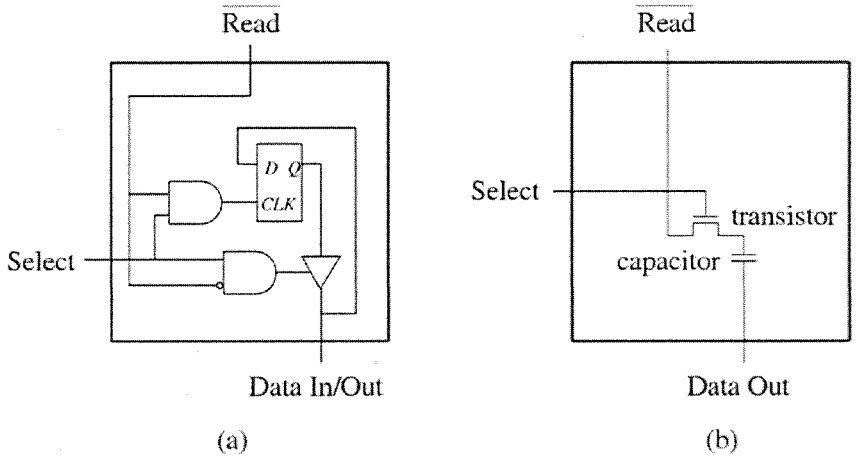
Intel'in Core 2 Duo işlemcilerinde bu önbellek çekirdekler tarafından paylaşımlı olarak kullanılıyor. Genellikle L2 önbellek çekirdek başına 512 KB veya 1 MB büyüklüğe sahip oluyor. L2 önbelleği düşük olan işlemciler daha çok giriş seviyesi modeller oluyorlar.



İlk başlarda sadece ek bir seviye olarak kullanılırken günümüzde L3 önbellekler çekirdekler arasında iletişimi de sağlayan geniş ve paylaşımlı yapılar halini aldılar. L3 önbelleği ilk kez masaüstüne getiren Phenom ailesiyle AMD oldu. 65 NM Phenom X4, 2 MB paylaşımlı L3 önbelleğe sahipti. Şu andaysa 45 NM Phenom II X4 işlemciler 6 MB paylaşımlı L3 önbelleğe sahip olarak geliyorlar. Intel'in Core i5 ve i7 işlemcileri ise 8 MB L3 önbelleğe sahipler.

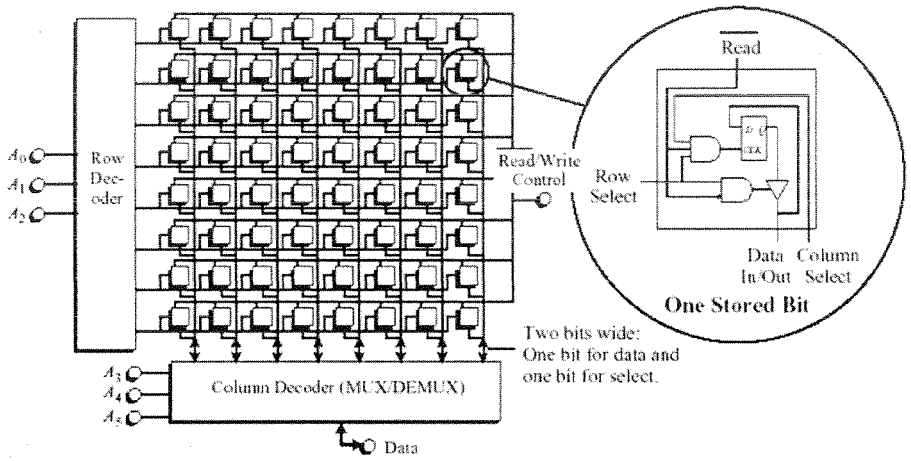
ANA BELLEK

- Bilgisayar sistemin merkezi depolama birimidir. Büyük ve hızlıdır.
- Komut ve dataların depolandığı birimdir
- Statik ve dinamik olmak üzere iki türlüdür.
- Statik hafıza birimlerinde bilgi yaz bozlardan oluşur. Akım verildiği süre boyunca bilgi depolanır.
- Dinamik RAM bilgileri elektrik akımı şeklinde tutar. Tutan devre kondansatörlerdir.
- Üzerlerindeki yük zamanla azalır. Bundan dolayı belli sürelerle yenilenmelidir.
- Yenileme birkaç milisaniyede bir tüm kelimelere uygulanır.
- Okuma ve yazma süreleri kısadır.

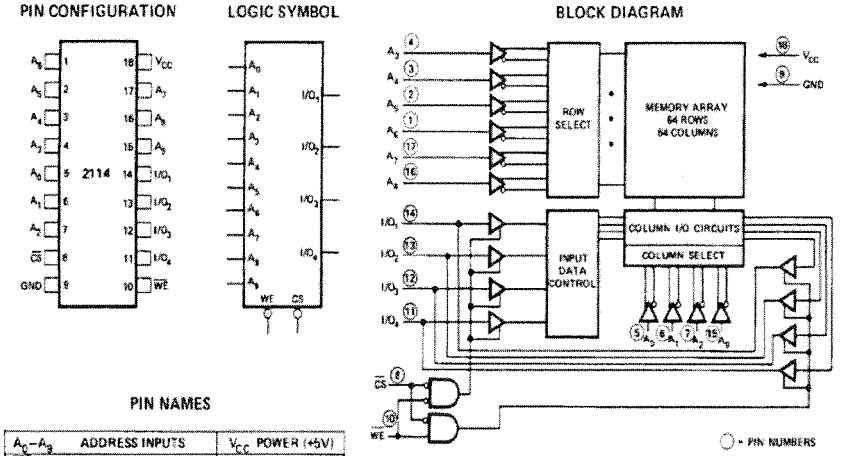


Static RAM cell (a) and dynamic RAM cell (b).

One-Bit RAM



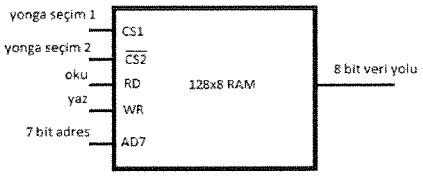
1K X 4 SRAM (Part Number 2114N) Blok yapısı

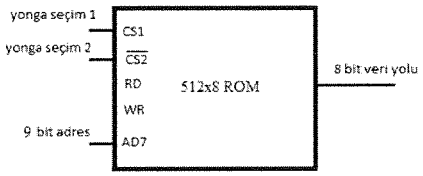


PIN NAMES

A_0–A_9	ADDRESS INPUTS	V_{CC} POWER (+5V)
$\overline{WE}$	WRITE ENABLE	GND GROUND
$\overline{CS}$	CHIP SELECT	
I/O_1–I/O_4	DATA INPUT/OUTPUT	

CS1	CS2	RD	WR	Bellek fonk.	Veri Yolu Durumu
0	0	X	X	Tutma	Yüksek direnç
0	1	X	X	Tutma	Yüksek direnç
1	0	0	0	Tutma	Yüksek direnç
1	0	0	1	Yaz	Ram'a giriş bilgisi
1	0	1	X	Oku	RAM'den çıkış bilgisi
1	1	X	X	Tutma	Yüksek direnç

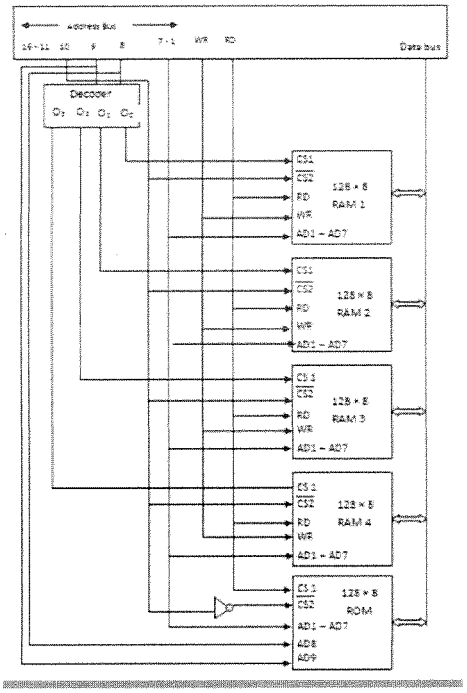




Bellek Adres Haritası

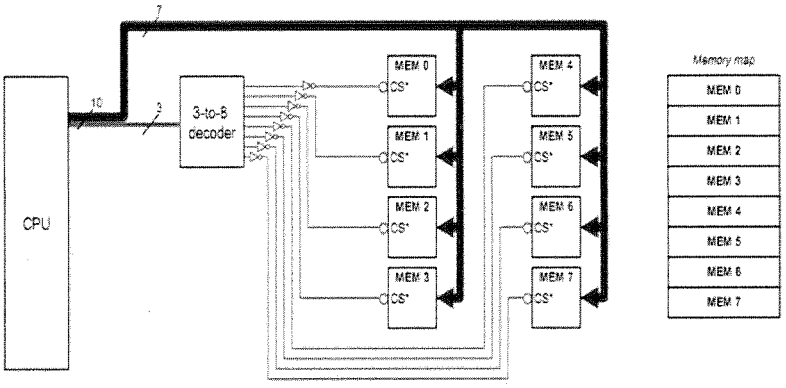
- Bilgisayar tasarımcısı sisteme gerekli bellek miktarının hesaplamalı ve RAM ve ROM olarak tahsis etmelidir.
- Bellek adreslemesi çizelge hazırlanarak yapılır. Bu çizelgeye bellek adres haritası denir.
- Sistemimiz 512 byte RAM ve 512 byte ROM gerektirsin. RAM yongaları 128 byte olsun buna göre bellek adres haritası aşağıdaki gibi olacaktır.
-

Bileşen		Adres yolu									
		10	9	8	7	6	5	4	3	2	1
RAM1	0000-07f	0	0	0	x	x	x	x	x	x	x
RAM2	0080-0ff	0	0	1	x	x	x	x	x	x	x
RAM3	0100-17f	0	1	0	x	x	x	x	x	x	x
RAM4	0180-1ff	0	1	1	x	x	x	x	x	x	x
ROM	0200-3ff	1	x	x	x	x	x	x	x	x	x



Bellek ile MIB Bağlantısı

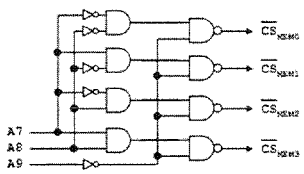
- 128x8 bayt hafıza organizasyonu

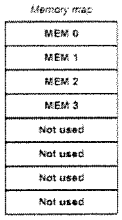


Örnek

- 10 adres hattı için 4 adet 128 bayt toplam 512 bayt hafıza için adres hafızasını tasarlayınız.

Device	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
MEM 0	0	0	0	X	X	X	X	X	X	X
MEM 1	0	0	1	X	X	X	X	X	X	X
MEM 2	0	1	0	X	X	X	X	X	X	X
MEM 3	0	1	1	X	X	X	X	X	X	X





İçerik Adreslemeli Bellek
Content Adressable Memory (CAM)

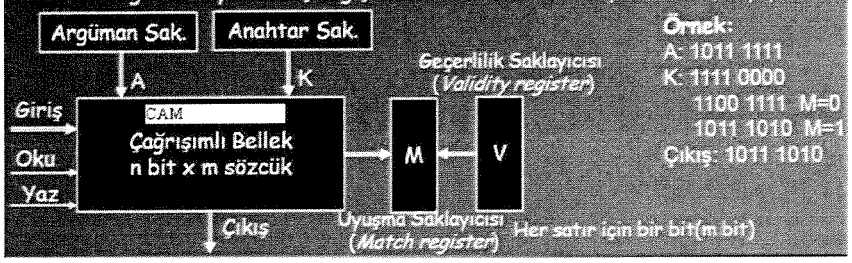
- Bir çok veri işleme uygulamalarında bellekte bulunan bir çizelgeden veri aranır.
- Bellekte bulunan bir değerin aranıp bulunması adres yerine verinin içeriği ile olursa arama süresi kısaltılabilir. Buna içerik adreslemeli bellek CAM denir. Bu belleğe aynı anda paralel olarak erişilebilir.
- Adres veya yer belirtmeye gerek yoktur.
- Bellek kullanılmayan boş bir alanı bularak kelimeyi buraya yerleştirir.
- Eğer kelime okunacaksa kelimenin içeriği veya onun bir kısmı belirtilir.
- Bellek bu verilene uygun ve onunla çakışan bütün kelimeleri belirler ve bunları okumak için işaret koyar.
- Tasarımından dolayı bu bellek paralel aramalar için uygundur.
- Bu bellek RAM belleklerden çok daha pahalıdır. Çünkü her hücre saklama yeteneği dışında içeriği karşılaştıracak devreler içerir.

Donanım Tasarımı

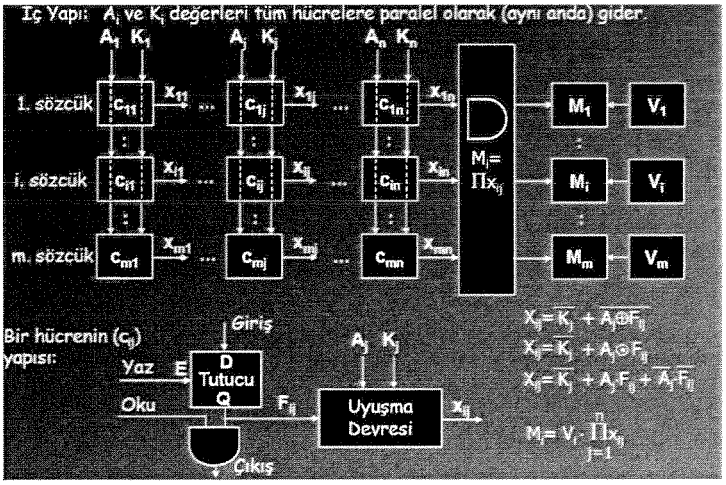
- İçinde bir bellek dizisi ve mantık devreleri vardır.
- Bellek kelime başına n bit olarak m kelimeden oluşur.
- Değişken yazacı A ve anahtar yazacı K n bit içerir.
- Çakıştırma yazacında m bit vardır.
- Herbir bellek kelimesi değişken yazacının içeriği ile paralel olarak karşılaştırılabilir.
- Değişken yazacının bitleriyle çakışan kelimeler çakıştırma yazacında karşılık gelen biti 1 yapar.
- Çakıştırma işleminden sonra çakıştırma yazacı içinde 1 yapılan bitler karşılık gelen kelimelerin çakıştığını belirtir.
- Okuma işlemi çakıştırma yazacında karşılık gelen bitleri 1 olan kelimelerde sıralı olarak yapılır.
- Anahtar yazacı değişken kelimesindeki belli bir alanı seçmek için bir maskeleme oluşturur. Anahtar yazacı 1'lerden oluşuyorsa değişken yazacının tamamı her bir bellek kelimesi ile karşılaştırılır.
- Maskeleme bellekte aranacak yazacı bir maskeleme oluşturur. Bu maskeleme bellekte aranacak bilgi parçasını gösterir.

CAM bellek yapısı

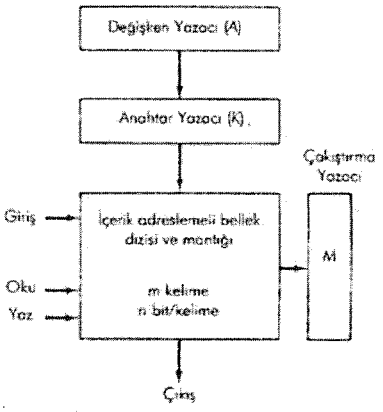
- Hızlı arama yapabilmek için kullanılır.
- Bellek ve paralel arama için kullanılan sayısal devrelerden oluşur.
- Girişine verinin kendisi (Argüman – A) ve bu verinin ne kadarlık kısmının aramada kullanılacağı (Anahtar *Key*-K) verilir.
- Eğer aranan veri bellekte varsa uyuşma saklayıcısının ilgili biti "1" olur, bellekteki veri çıkıştan alınabilir.
- İlk elektrik verildiğinde bellekte rasgele değerler olacağından geçerlilik (*valid*) bitlerine de gerek duyulur. Başlangıçta V=0'dır, i. satıra veri yazılınca V_i=1 yapılır.



CAM BELLEK yapısı



CAM BELLEK ŞEMASI

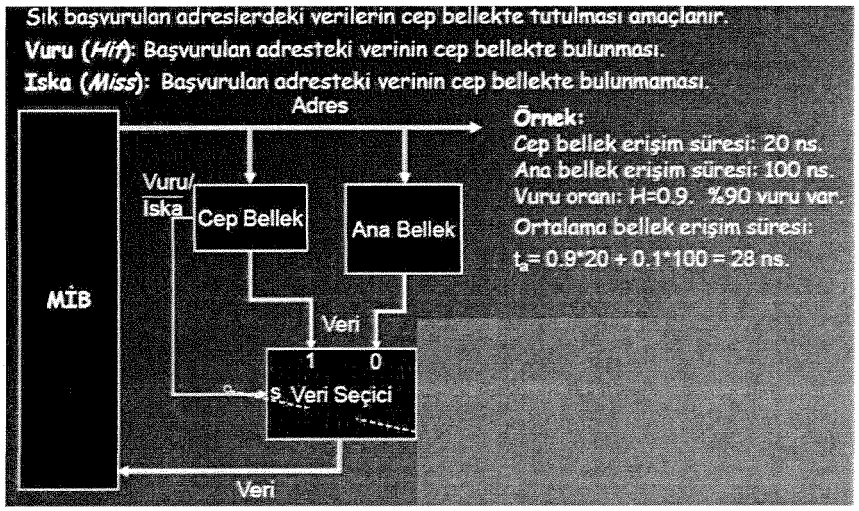


Şekil 12.6 İçerik adreslemeli bellek şeması.

CACHE (Ön bellek-CEP Bellek)Bellek Organizasyonu

- Programın çalışan kısımları ve verilerin o anda kullanılan kısımları küçük ve hızlı belleğe konursa ortalama bellek erişim süresi kısaltılabilir.
- Böylece programın icra süreside kısaltılabilir.
- Böyle bir belleğe ön bellek denir.
- Bu bellek küçük ve hızlıdır.
- Erişim süresi ana belleğin 5 te biri ile 10 da biri arasındadır.
- MIB ile ana bellek arasında olur. Bellek hiyerarşisinde en hızlı bellektir. Hızı MİB bileşinlerine yakındır.
- Ön bellek tasarımını temel fikri en sık kullanılan buyruk ve verileri bu bellekte tutmaktır.
- Böylece ortalama bellek erişim süresi ön bellek erişim süresine yaklaşır.

Cache (Cep-Önbellek) bellek çalışma prensibi



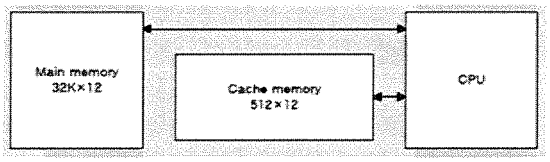
- MIB belleğe erişmek istendiğinde ön belleği inceler. İstenen ön bellekte ise buradan okunur.
- Eğer bu adres ön bellekte bulunmazsa o zaman ana belleğe gidilerek okunur. Sonra bu okunan bilgini bulunduğu blok ön belleğe aktarılır. Blok büyüklüğü erişilen kelimeye komşu yaklaşık 16 kelimeden oluşur. Böylece erişilen yakın verilere yeniden erişmek kolay hale gelir.
- **Ön belleğin verimi, bulma oranı denilen bir sayı ile ölçülür.**
- MIB belleğe erişmek ister ve aradığını ön bellekte bulursa bu bir bulmadır. Eğer aranan kelime ön bellekte değilse ve bellekte bulunursa buna bir bulamama denir.

- **Oran=bulma sayısı / Toplam erişim sayısı**
- **Oran=Önbellek erişim sayısı/(ön bellek erişim sayısı+ana bellek erişim sayısı)**

Ana bellekten ön belleğe verilerin aktarılmasına haritalama işlemi denir.

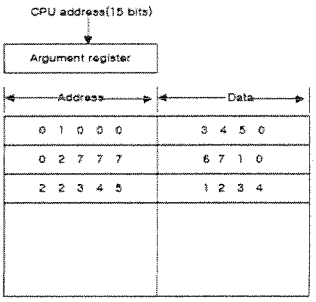
- İçerik adreslemeli haritalama
- Doğrudan haritalama
- Kümelenmiş içerik adreslemeli haritalama

- Ana bellek 12 bitlik 32 K kelimeden oluşsun. Ön bellek ise bunların 512 tanesini taşıyor olsun. Ön bellekte bulunan her kelimenin ana bellekte bir kopyası bulunur. MIB her iki belleğede erişebilir.
- Ön belleğe 15 bit adres yollar. Eğer aranan kelime ön bellekte varsa buradan 12 bit veriyi okur. Aranan kelime ön bellekte bulunmazsa MIB kelimeyi ana bellekten okur ve kelimeyi ön belleğe aktarır.



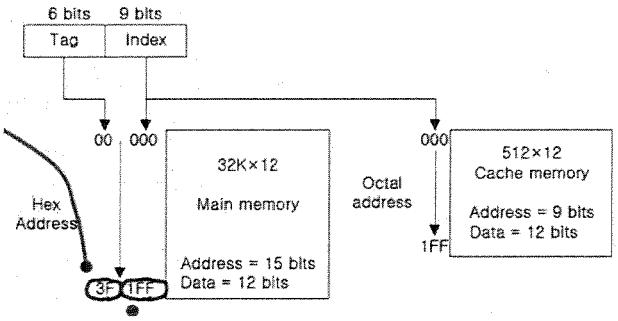
İçerik Adreslemeli Haritalama

- En hızlı ve çok yönlü kullanılabilen ön bellek tasarımı içerik adreslemeli bellek alanıdır. Buradaki **CAM bellek** hem adresi hemde veriyi tutar.
- Böylece belleğin herhangi bir alanındaki veriler veya buyruklar ön belleğin herhangi bir yerine yazılabilir.
- Şekilde bellekte duran 3 adres, beş basamak sekizlik düzende (15 bit) sayılarla ve bunlara karşılık gelen 12 bit lik vriler ise 4 sekizli düzende sayılarla verilmiştir. 15 bit MIB adresi değişken yazacına yazılır ve çakıştırma yapılır.
- Adres varsa veri okunur sonra bulunduğu adres ve veri ön belleğe aktarılır.



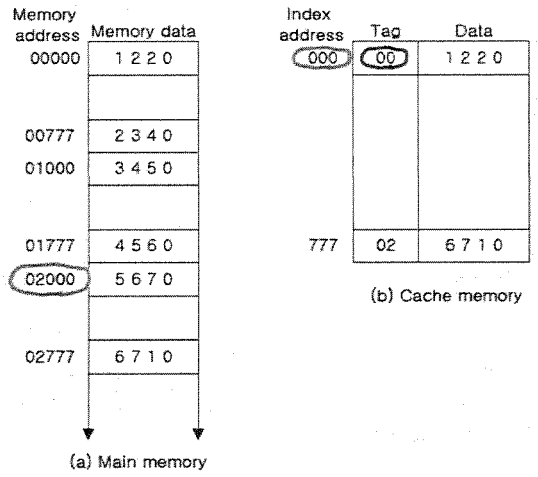
Doğrudan Haritalama

- CAM bellekler RAM'a göre pahalıdır. Çünkü hücrelere bir çok devre eklenmiştir.
- 15 bit CPU adresi 2 alana bölünür. 9 bit indis alanı 6 bit başlık alanıdır. Aslında Önbellek için 9 bit yeterlidir.
- Cache içindeki herbir kelime (12 bitlik) veri ve (6 bitlik) başlık (Tag) bilgisini tutar.



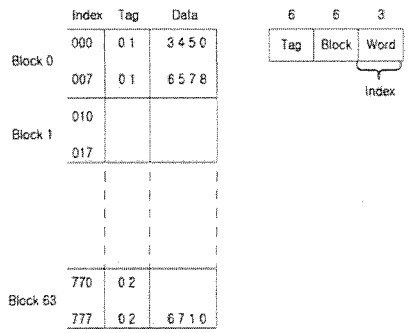
- n bitlik bellek adres alanı k ve n-k olarak ikiye ayrılır. k indis olup ön belleği adresler.
- MIB bir bellek isteminde bulunduğunda ön belleğe erişmek için indis alanı kullanır. MIB adresinin başlık alanı ön bellekten okunan kelimenin başlık alanı ile karşılaştırılır. İki başlık çakışırsa istenen kelime ön bellektedir. Çakışma yoksa istenen kelime ana bellekten okunur. Bu kelime daha sonra başlığıyla beraber ön belleğe yazılır eski kelime atılır.
- Doğrudan haritalamada bit sayısı değişebilir.
- Çünkü iki veya daha fazla kelimenin indis alanları aynı ise ve başlık alanları farklı ise ve bunlara tekrar tekrar erişiliyorsa bu durum ortaya çıkar.

Şekilde görüldüğü gibi ana hafızadaki 00000 adresindeki 1220 verisi Cache bellek içindedir. Ve indisi 000 ve Tag (başlığı) 00'dır.
Örneğin; CPU'nun 02000 adresindeki veriyi istediği varsayılsın. Bunun indisi 000'dır. Tag'i 02'dir. Fakat TAG'ler çakışmadığından bu veri önbellekte your. Veri (5670) Ana bellekten alınıp ön belleğe atılır.



- Bir önceki örnekte 1 kelimelik blok için doğrudan erişim yapılmaktadır.
- Aynı işlem 8 kelimelik bloklar içinde yapılabilir. Bunun için indis alanı 2 parçaya bölünür.
- **Blok alanı**
- **kelime alanı .**

•512 kelimelik bir ön bellekte her biri 8 kelimelik 64 blok vardır (64x8). Dolayısıyla Blok numarası 6 bitlik bir alanla verilir. Kelime alanı ise blok içinde 3 bit yer kaplar. Ön bellek içindeki başlık(tag) alanı blok içindeki 8 kelime için aynıdır.
• Bir bulamama olduğunda 8 kelimenin tamamı bellekten ön belleğe aktarılır. Buda zaman alır.
•Daha büyük blok büyüklüğü tanımalamakla bulma oranı arttırılabilir.



Kümelenmiş İçerik adreslemeli haritalama

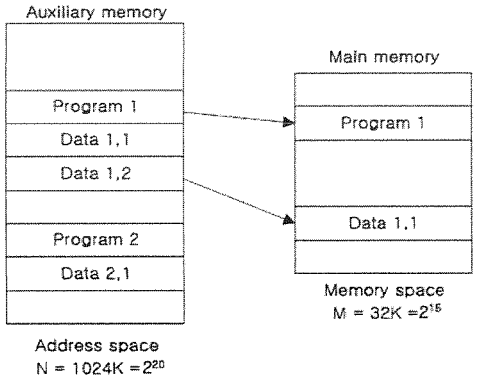
- Doğrudan erişimli ön bellek tasarımında , adreslerinde aynı indisli fakat farklı başlık değeri bulunan iki kelime aynı anda ön bellekte bulunamazlar.
- Bu direkt adreslemenin dezavantajıdır.
- Kümelenmiş içerik adreslemeli haritalama direkt haritalamanın gelişmiş şeklidir.
- Aynı ön bellek kelimesi içinde birden fazla aynı indis adresli bellek kelimesi bulunabilir.
- Her bir veri kelimesi başlığı ile beraber depolanır ve bunlar bir küme oluşturur.
- Küme sayısı birden fazladır.

Index	Tag	Data	Tag	Data
000	0 1	3 4 5 0	0 2	5 6 7 0
777	0 2	6 7 1 0	0 0	2 3 4 0

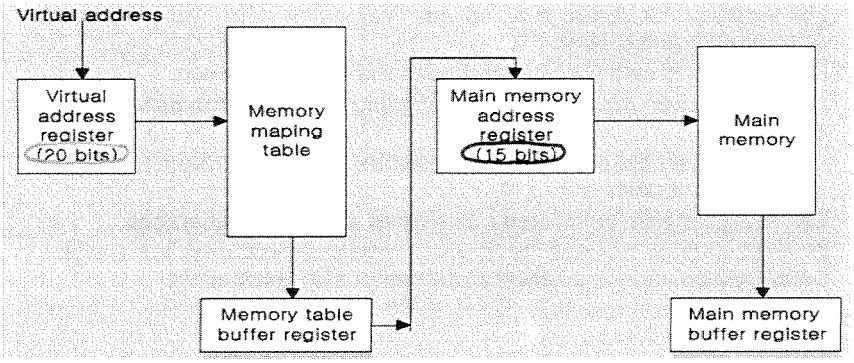
Sanal bellek

- Sanal bellek büyük bilgisayar sistemlerinde kullanılan bir kavramdır.
- Bununla kullanıcılar yardımcı belleğin kapasitesine eşit belleği olan sanal ve bireysel makinede çalışıyormuş gibi olurlar.
- MB tarafından verilen her adres bir haritaya gider ve sanal adres fiziksel adrese dönüştürülür.
- Sanal bellek adres sistemi ile programdaki adresler gerçek ana bellek adreslerine dönüştürülür.
- Bu iş MIB de çalışırken dinamik olarak yapılır.
- Dönüşüm veya haritalama donanım tarafından bir haritalama çizelgesi yardımıyla otomatik olarak yapılır.
- Kullanıcı tarafından kullanılan adrese sanal adres denir.
- Bu adreslerin kümesine adres uzayı denir.
- Ana bellekteki bir adrese fiziksel adres veya yer adı verilir. Bunların kümesine bellek uzayı denir.
- Dolayısı ile adres uzayı programlar tarafından oluşturulan buyrukların ve verilerin adresleridir.
- Bellek uzayı ise gerçek bellek adresleri olup işlemler için doğrudan verilmelidir.

- Bir bilgisayar düşünün belleği 32K olsun. Bu bilgisayarın 1024 K kelimelik bir yardımcı belleği olduğunu varsayalım.
- 2^20 =1024 olduğundan yardımcı belleğin kapasitesi 32 ana bellek eder. Adres uzayı N ve bellek uzayı M ile gösterilirse N=1024 ve M=32 K dır.
- Çok programlı bir bilgisayar sisteminde programlar ve veri yardımcı bellek ile ana bellek arasındadır.
- Farzedelim ki program 1 MIB tarafından icra edilmekte olsun.
- Program ve onun datasının bir kısmı yardımcı bellekten ana belleğe aktarılır.
- Program ve veri bölümleri bellekte birbirine komşu ve birbirini takip eden adreste bulunmazlar.
- Çünkü bilgiler gidip gelmektedir. Boş yerler ise dağınık yerlerdedir.
- Örnekte yardımcı bellekten tek tek veri veya buyruk okumak zaman alır. Dolayısıyla bunların ana bellekten okunması daha avantajlıdır.



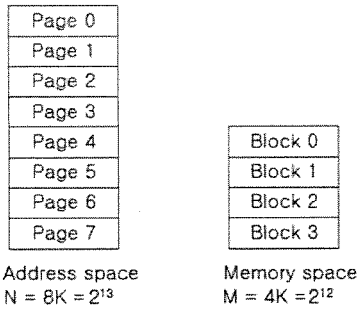
- 20 bit adres 15 bite dönüştürülmelidir. sanal adresin fiziksel adrese dönüşümü şekildeki gibi olacaktır.



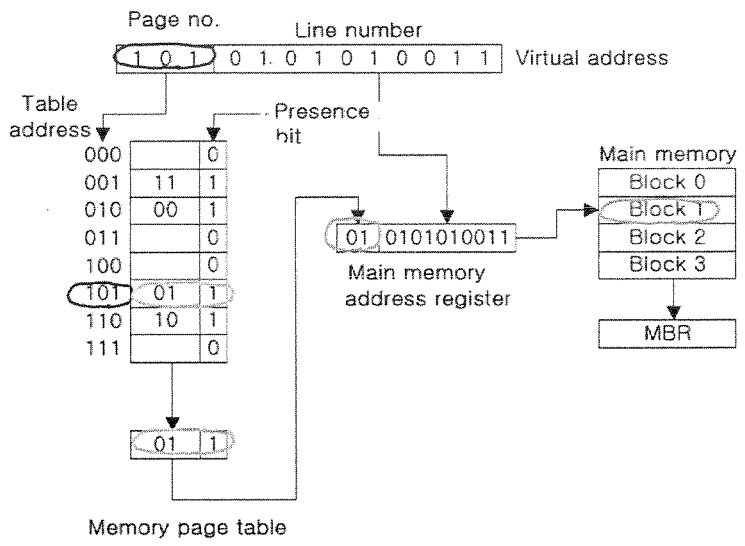
Sayfa kullanarak adres haritalama

- Adres haritalama işlemi adres ve bellek uzaylarının her biri sabit uzunlukta gruplara bölünürse çok basit hale gelebilir.
- Fiziksel bellek blok adı verilen eşit büyüklükte gruplara bölünmüştür.
- Bu büyüklük 64 ile 4096 arasında olabilir. Sayfa deyimi adres uzayında eşit büyüklükteki grupların adıdır.
- Adres uzayı 8 K
- Bellek uzayı 4 K
- Eğer herbiri 1 K gruplara

Bölünürse 8 sayfa 4 blok eder.

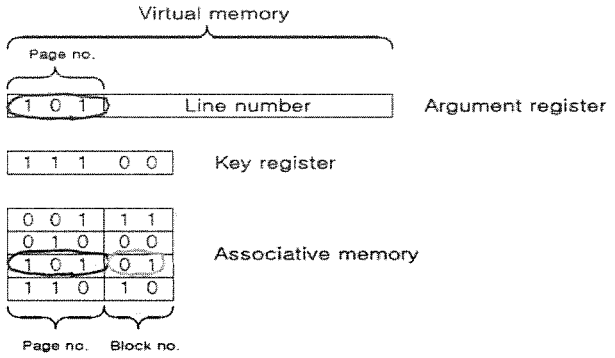


- Bellek sayfa çizelgesinde 8 kelime vardır her sayfa için bir kelime sayfa çizelgesindeki adres sayfa numarasını gösterir.
- Çizelgede 1,2,5 ve 6 numaralı sayfaların ana bellekte 3,0,1,2 numaralı bloklarda olduğunu gösterir.
- Çizelgedeki 1 bit varlık biti olarak tanımlanır. Ve sayfanın yardımcı bellekten ana belleğe aktarılmış olup olmadığını gösterir. 0 aktarılmadığını gösterir.
- MIB bellekten bir kelimeyi 13 bitlik sanal adres ile arar.
- Sanal adresin önemli 3 biti sayfa numarasını ve doğal olarak bellek sayfa çizelgesinde bir adres gösterir.
- Bellek sayfa çizelgesindeki bir kelimenin içeriği sayfanın ana bellekteki blok numarasını verir.
- Bu içrik ana bellek adres yazacındaki önemli bitleri oluşturur. Satır sayısı ise sanal adresten doğrudan doğruya ana bellek adres yazacına aktarılarak ana bellek adresi hazırlanır.
- Ana belleğe gönderilen oku sinyali ile ana bellekteki kelimenin içeriği MBR de CP nin kullanımına hazır olur. Varlık biti 0 ise sanal adresin gösterdiği kelime bellekte değildir.
- İşletim sistemi çağrılarak istenen sayfanın ana belleğe aktarılması ve bu işin hesaplamadan önce yapılması sağlanır.



İçerik adreslemeli bellek sayfa çizelgesi

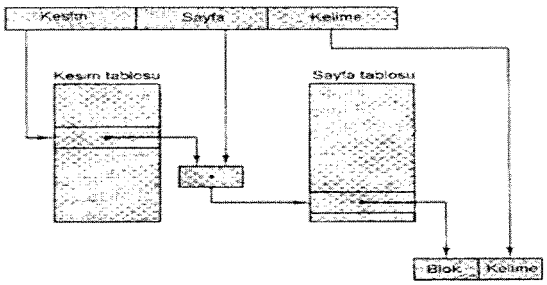
- RAM sayfa çizelgesi kullanışsızdır. Önceki örnekte 8 kelimesine gerek vardı. Her sayfa 1 tane. Bunlardan 4 tanesi daima boş kalır.
- Çünkü bellek 4 blok olmaktadır.
- Genel olarak n sayfa m bloktan oluşan bir sistemde n kelime gerekir.
- Bunlardan sadece m tanesi doludur. Diğerleri boştur.
- 1024 K kelimelik adres uzayı ve 32 kelimelik bellek düşünelim. Hersayfa ve blok 1 kelime ise sayfa sayısı 1024 ve blok sayısı 32 olur. O halde bellek sayfa çizelgesi 1024 kelime olacak ve sadece 32 tanesinde varlık biti 1 olabilecektir. Herhangi bir anda 992 yer boş olacaktır.
- Çizelgeyi oluşturmak için daha etkin bir yol kelime sayısının bellekteki blok sayısına eşit alınmasıdır.
- Böylece çizelge büyüklüğü küçülür ve tamamen kullanılır.
- Bu çizelge CAM bellekte tutulabilir.
- Bu çizelgedeki her kelimede sayfa numarası ve buna karşılık gelen blok numarası bulunur. Kelimedeki sayfa alanı ile sanal adresteki sayfa alanı karşılaştırılır.
- Eğer çakışma olursa kelime bellekten okunur. Ve karşılık gelen blok numarası elde edilir.



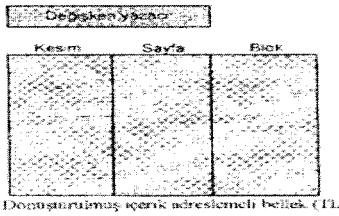
Bellek Yönetim Donanımı

- Çoklu işletim sisteminde bir çok program bellekte bulunur.
- Ayrıca programların ve verilerin sürekli gidip gelmesi gerekir. Bir programın kaplayacağı alan değişebilir. Bu programların idare edilmesi gerekir.
- Bellek yönetim biriminin parçaları şunlardır.
- Mantıksal bellek adreslerini fiziksel adreslere dönüştüren dinamik bir dönüşüm parçası
- Bellekte bulunan programların farklı kullanıcılara kullanılmasını sağlayan bir parça
- Bilginin yetkisiz kullanıcılardan korunması ve kullanıcıların işletim sisteminin değişen fonksiyonlarda korunması

- Mantıksal adres 3 bölümden oluşur.
- Kesim alanı kesim numarasını belirler. Sayfa alanı kesim içindeki sayfayı kelime alanı sayfa içindeki kelimeyi belirtir.
- Mantıksal adresin fiziksel adrese dönüşümü iki çizelge ile yapılır. Mantıksal adresin kesim numarası kesim çizelgesi için olan adresi belirler kesim çizelgesindeki giriş sayfa çizelgesi için göstergedir. Toplam sayfa çizelgesi içinde bir giriş bir gösterge adresidir.
- Sayfa çizelgesinde bulunan değer fiziksel bellekte bir blok numarasıdır. Blok alanı ile kelime alanının arka arkaya yazılarak birleştirilmesi son fiziksel adresi verir.
- Bu sistem için CAM bellek kullanımı hızı arttırır. Son bellek erişimlerinin adresleri burada tutulur.
- Bu tip belleğe dönüştürülmüş içerik adresli bellek TLB denir.

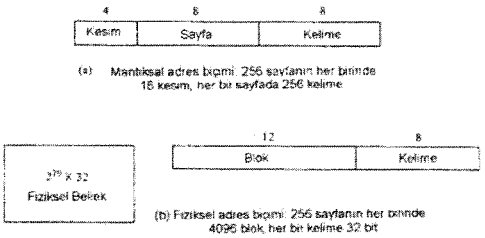


(a) Mantıksaldan fiziksel adrese haritalama

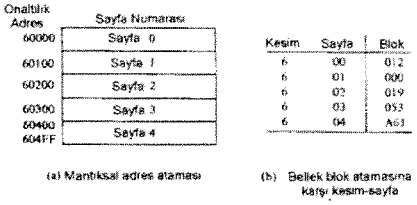
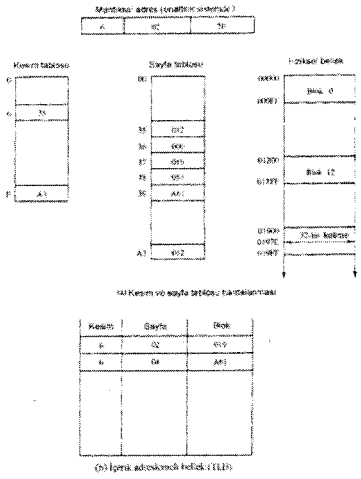


(b) Dönüştürülmüş içerik adreslemeli bellek (TLB)

Şekil 12.21 Kesimlenmiş sayfa bellek yönetim biriminin haritalaması



Şekil 12.22 Mantıksal ve fiziksel adrese bir örnek.



Şekil 12.23 Mantıksal ve fiziksel adres atama örneği