

## Ders Hedefi

Bu derste sayısal bilgisayarda kullanılan sayısal bileşenlerden başlayarak, bilgisayar organizasyonunun oluşturulması ve yazılıminin assembler seviyesinde tanıtılması hedeflenmiştir. Lojik kapıların (AND, OR, NOT, XOR, etc) bir üst kümesi olan sayısal bileşenler (dekodeur, enkoder, çoğullayıcı (MUX), hızlı bellekler,...) bilgisayarın temelini oluşturur. Bu elemanları kullanarak aritmetik lojik birimi, hızlı bellekler, anabellek, hat grubu ve giriş-çıkış bileşenlerinin oluşturulması anlatılacaktır. Sonraki adımda bu bileşenler "tek akümülatör" yapısını oluşturmak üzere bir araya getirilecek ve gerekli yapısal "kabuller" ifade edilecektir. Komut seti ve adres modları tasarılanıp, bu komut setinin gerçeklenmesi açıklanacaktır.

Yazılım kısmında ise alt seviyede "assembler dil" komutlarıyla tanıtılıp uygulamalar verilecektir.

Bilgisayarın zamanlama ve kontrolü ile ilgili iki yöntem tanıtılıp detaylandırılacaktır: donanım ile kontrol ve yazılım ile kontrol. Bu iki yöntemin tasarlanan bilgisayarda kullanımı açıklanıp, aralarındaki farklar belirtilecektir.

Son olarak, tasarlanan bilgisayar yapısı, genel hızlı bellek elemanlı bilgisayarlar ve yiğin gibi türler ile genişletilecek, komut setleri ve adres modları üzerinde örnekler verilecektir. Tasarlanan bilgisayarın günümüzde genel bilgisayar yapılarına olan bağlantısı açıklanacaktır.

### BÖLÜM - 1

## SAYISAL LOJİK DEVRELERİN ÖZETİ



Bu bölümde aşağıda sıralanan konular incelenecektir.

- Sayısal bilgisayarın tanımı
- Lojik kapılar
- Boole cebri
- Karnaugh map örneği
- Kombinasyonel devreler örneği
- FF'ler
- Ardışıl devre örneği

Bu dersin her hakkı Ahmet Yesevi Üniversitesi'ne aittir.  
Kopyalanamaz ve ıconsız kullanılabilir.

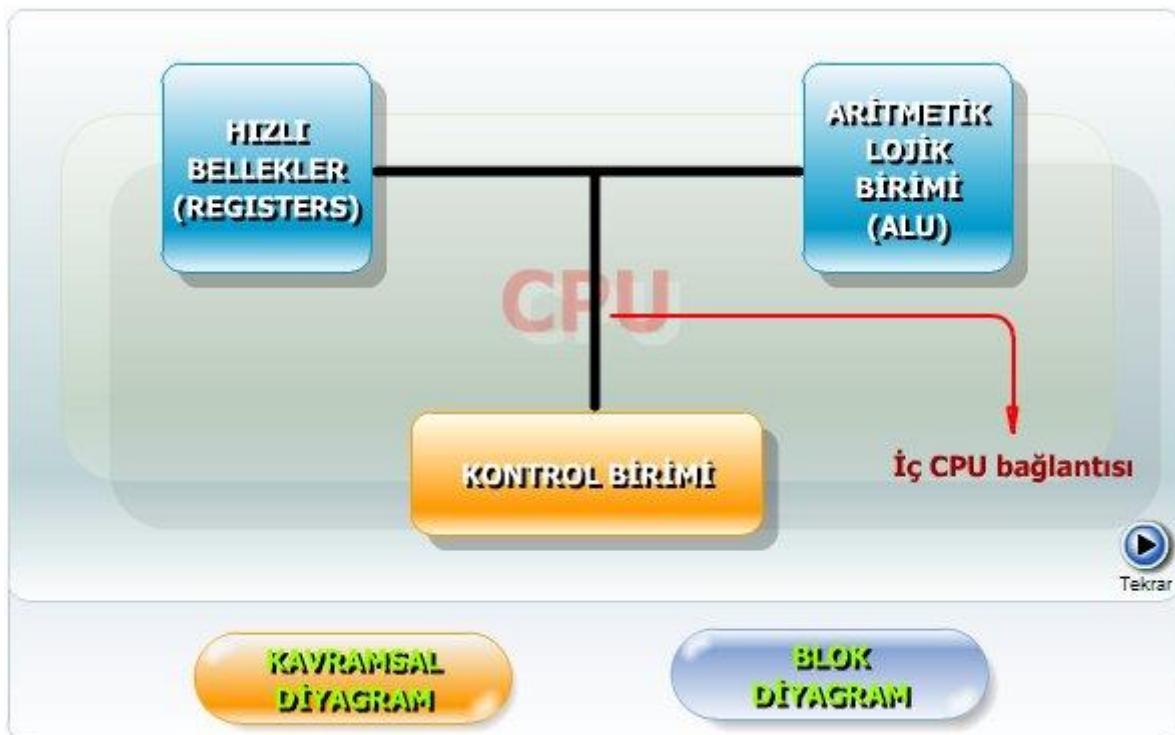
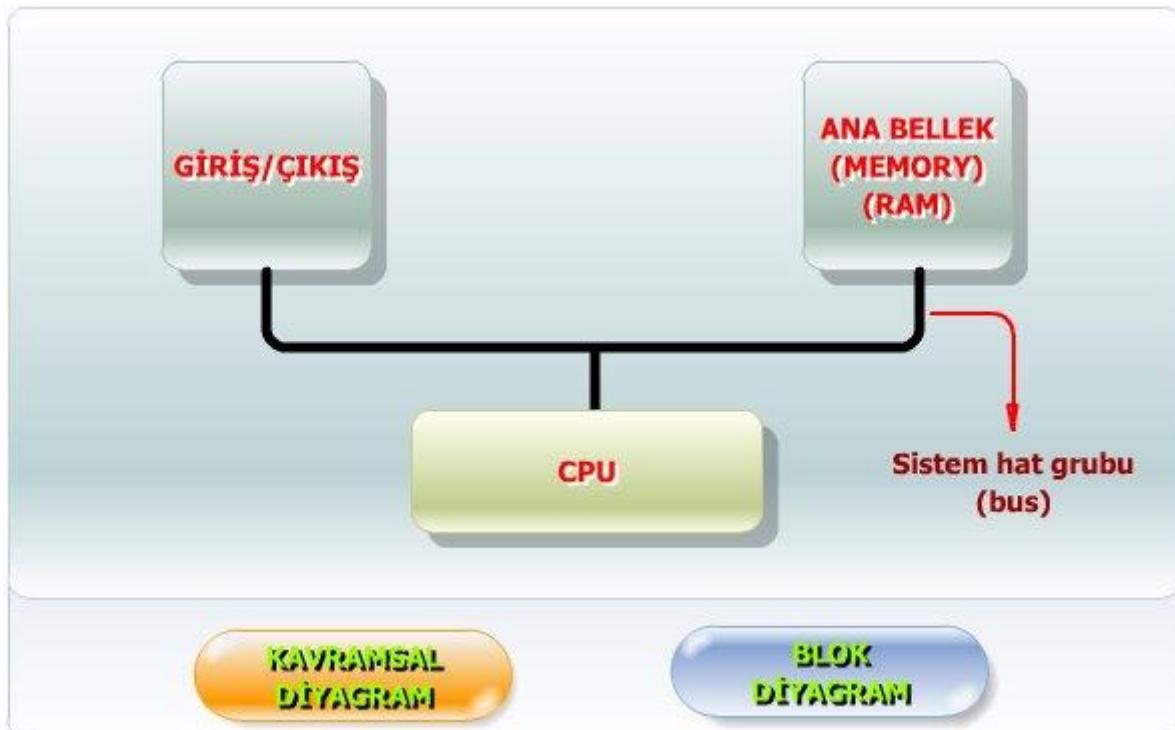
## Bölüm Hedefi

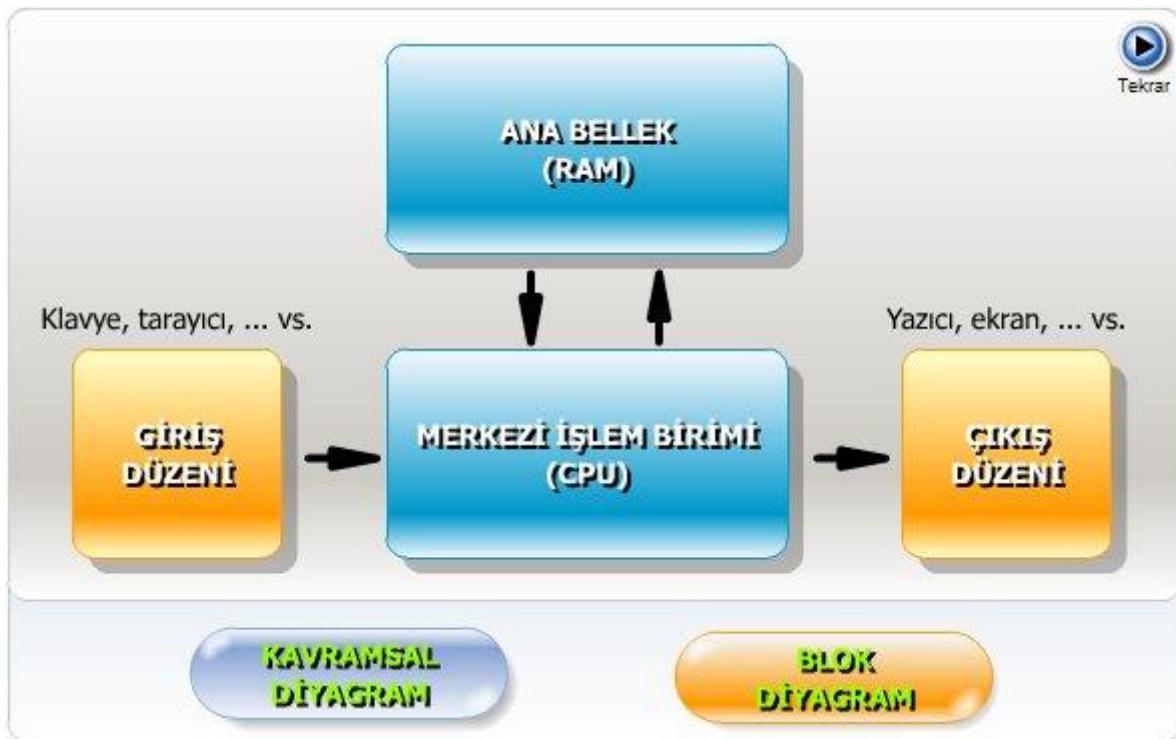
Bu haftaki derste sayısal bilgisayar, lojik kapılar, Boole cebri, Karnaugh haritası basitleştirilmesi özet, kombinasyonel devreler, Flip-Flop (FF) çeşitleri ve FF'li (ardışıl) devrelerin örnekler ile kısa özetü üzerinde durulacaktır. Bu bölümdeki amaç, bazı temel kavramların yüzeysel olarak özetlenmesiyle sonraki derslerde kullanılabilecek birtakım kavramlara geçiş kolaylığını sağlamak olacaktır.

### 1.1 SAYISAL BİLGİSAYAR

Sayısal bilgisayar, basitçe, ikili (0 ve 1) sayılar ile çeşitli hesaplamaları yapan sistemdir. İkili sayı sistemi "0" ve "1"den oluşur. Aşağıda sayısal bir bilgisayarın kavramsal ve blok diyagramı gösterilmiştir (Von Neumann makinası).

! Sayısal bir bilgisayarın kavramsal ve blok diyagramını görmek için düğmelere tıklayınız.





Sayısal bilgisayar merkezi işlem birimi (CPU: central processing unit), bellek birimi (RAM: random access memory) ve giriş-çıkış biriminden oluşur (IOP:giriş-çıkış işlemcisi). CPU aritmetik ve lojik birimi, hızlı bellekleri (registers) ve kontrol devrelerini içerir. RAM komutların ve verinin saklanmasında kullanılır. IOP ise bilgisayar ile dış dünya arasında bilgi, veri alışverişini sağlar.

Bilgisayar, program yoluyla verileri (data) işler. Yazılan program (tanımlanan algoritma), çözülmesi istenen probleme uygun bir çözüm üretir. Diğer adı veri işlemedir (data-processing).

### 1.1.1 Sayısal Bilgisayarların Yapısı

Sayısal bilgisayarı veya sadece bilgisayarı, yazılım ve donanım olmak üzere iki temel kısımdan oluşmuş kabul edebiliriz. Donanım fiziksel gerçeklenebilen elemanlardır: lojik kapılar, FF'ler, sayısal komponentler (tasarlanmış sayısal sistem elemanları, örneğin dekoder, enkoder, multiplekser-çoğullayıcı). Yazılım ise programlardan oluşur.

**! Açıklamaları görmek için düğmelere tıklayınız.**

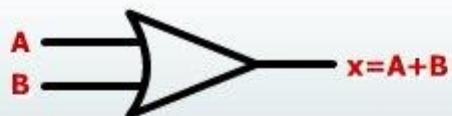


Bilgisayar organizasyonu donanım komponentlerinin interaktif olarak nasıl çalıştığını tanımlar. Bilgisayar mimarisi, yapının kullanıcı tarafından görülen kısmı ile ilgilidir, örneğin komut (instruction) formatları, adres modları (addressing modes).

## 1.2 LOJİK KAPILAR

Lojik kapılar sayısal tasarımın temel elemanlarıdır. Tipik örnekleri arasında AND, OR, NOT, buffer, XOR'u sayabiliyoruz. İkili (0,1) bilgi girişlerinden yeni bir ikili (0,1) çıkış bilgisi üretirler. OR ve XOR örnekleri aşağıda gösterilmiştir.

! OR ve XOR örneklerini görmek için düğmelere tıklayınız.



A	B	x
0	0	0
0	1	1
1	0	1
1	1	1

OR LOJİK  
FONKSİYONU

XOR LOJİK  
FONKSİYONU



A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

OR LOJİK  
FONKSİYONU

XOR LOJİK  
FONKSİYONU

! Uyarı: Diğer fonksiyonları [AND, ileti elemanı (buffer)... vs.] hatırlayınız.

Lojik kapılarının giriş-cıkış ilişkisi, "doğruluk tablosu" (truth table) aracılığıyla gösterilir. Doğruluk tablosu ikili giriş değişkenleriyle çıkış değişkeni arasında dönüşümü sağlayan tablodur.

### 1.3 BOOLE CEBİRİ

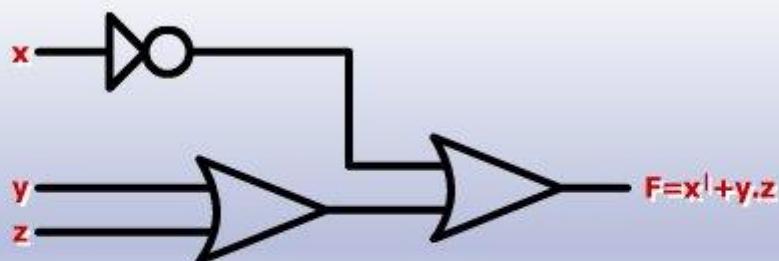
"Boole Cebri" sayısal devrelerin analiz ve tasarımını sağlayan matematiksel teoridir. Sayısal bilgisayar devreleri uygulamasında, ikili değişkenler üzerinde tanımlanan lojik operasyonları gösterir. Örneğin,

$$F = x^{\prime} + y \cdot z$$

**! Hatırlatma:** "+" işaretini OR işlemi, "·" işaretini AND işlemi için kullanılır.

Üç değişkenli ( $x, y, z$ ) bir  $F$  Boole fonksiyonunu tanımlar. Doğruluk tablosu yoluyla  $F$  Boole fonksiyonunun  $x, y, z$  değişkenlerinin çeşitli değerlerine göre alacağı değerler tanımlanır ve lojik diyagram ile lojik kapılar cinsinden şekillendirilebilir.

**!  $F = x^{\prime} + y \cdot z$  'nin lojik diyagramı ve doğruluk tablosunu görmek için düğmelere tıklayınız.**



**$F = x^{\prime} + y \cdot z$  NİN  
LOJİK DİYAGRAMI**

**$F = x^{\prime} + y \cdot z$  NİN  
DOĞRULUK TABLOSU**

x	y	z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$F = x^1 + y.z$  YENİ  
LOJİK DİYAGRAMI

$F = x^1 + y.z$  YENİ  
DOĞRULUK TABLOSU

Boole cebrinin temel eşitlikleri (örn:  $0+1=1$ ,  $x.1=1$ , etc.) ve De Morgan teoremi devrelerin gerçekleştirmesinde kullanılan temel kurallardır.

### 1.3.1 DeMorgan Kuralı

DeMorgan kuralı NDR ve NAND kapılarıyla aşağıdaki dönüşümleri sağlar.

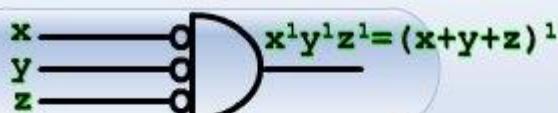
$$\text{NOR} \Leftrightarrow (x + y)^1 = x^1 y^1$$

$$\text{NAND} \Leftrightarrow (x \cdot y)^1 = x^1 y^1$$

Bu durumda;



OR-değil (NOR)

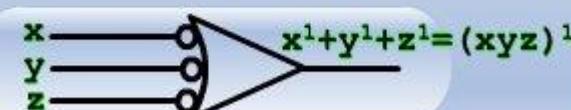


değil-AND

ve



AND-değil (NAND)



değil-OR

bağlantıları elde edilir.

## 1.4 KARNAUGH HARİTASININ (K-map) BASİTLEŞTİRİLMESİ

Boole fonksiyonu küçükterimlerin (minterm) kombinasyonu cinsinden yazılıp basitleştirilebilir. Bu işlem tasarlanan lojik devrelerin minimum elemanlı tasarımları için kullanılır. "n" değişkenli Boole fonksiyonu  $2^n$  küçükterim içerir. Örneğin n=3 değişkenli K-map 8 küçükterim ile (8 kare ile) gösterilir.

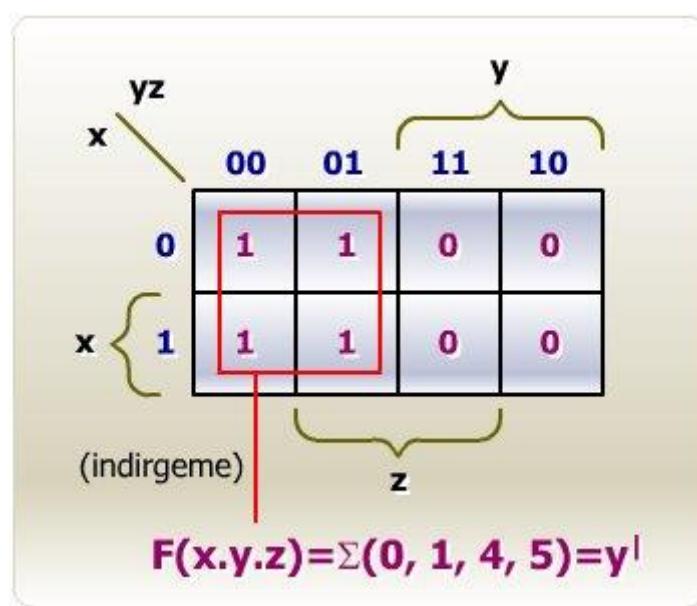
$$F(x, y, z) = \Sigma (0, 1, 4, 5)$$

Komşu kareler Boole fonksiyonunun sadeleştirilmesinde kullanılır. Böylece en az sayıda eleman içeren bir devre tasarlanır.

Örnek olarak 0 ve 1 mintermlerini gözönüne alalım:

000 ve 001 değerleri  $x^1y^1z^1$  ve  $x^1y^1z$  olup sadeleşebilirler. Sadeleşme sonucu ise 00 olup  $x^1y^1$  dir:

$$x^1y^1z^1 + x^1y^1z = x^1y^1 \underbrace{(z^1 + z)}_{=1} = x^1y^1$$



$$F(x,y,z) = \Sigma (0,1,4,5) \text{ K-map gösterimi}$$

## 1.5 KOMBİNASYONEL DEVRELER

Kombinasyonel devre lojik kapılarının, giriş ve çıkışların bağlanmasıından oluşan devredir. Geribesleme ve Flip-Flop elemanı bulunmaz. Çıkışlar sadece giriş elemanlarına bağlıdır.



Flip-Flop bulunmaz.



Kombinasyonel devre yalnızca **AND**, **OR**, **NOT**, **NAND**, vb. lojik kapılardan oluşur.

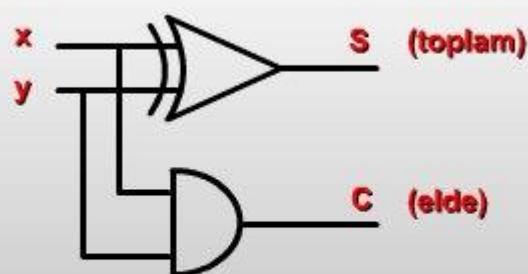
Kombinasyonel devrinin blok diyagramı

Kombinasyonel devrenin analiz ve tasarımları doğruluk tablosu yoluyla yapılabilir. Analiz evresinde, her bir devre çıkışları, giriş değişkenleri cinsinden yazılır. Tasarım evresinde, verilen problem giriş ve çıkış değişkenleri cinsinden tanımlandıktan sonra, doğruluk tablosu ile giriş-çıkış bağıntısı belirlenir. Sonra, K-map yoluyla basitleştirme (minimum elemanla devre gerçekleme) yapılp, devre gerçekleştirir.

## 1.5 KOMBİNASYONEL DEVRELER

Kombinasyonel devre lojik kapıların, giriş ve çıkışlarının bağlanmasıından oluşan devredir. Geribesleme ve Flip-Flop elemanı bulunmaz. Çıkışlar sadece giriş elemanlarına bağlıdır.

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Doğruluk tablosu

Lojik diyagram

Kombinasyonel devrinin blok diyagramı

Kombinasyonel devrenin analiz ve tasarımları doğruluk tablosu yoluyla yapılabilir. Analiz evresinde, her bir devre çıkışları, giriş değişkenleri cinsinden yazılır. Tasarım evresinde, verilen problem giriş ve çıkış değişkenleri cinsinden tanımlandıktan sonra, doğruluk tablosu ile giriş-çıkış bağıntısı belirlenir. Sonra, K-map yoluyla basitleştirme (minimum elemanla devre gerçekleme) yapılp, devre gerçeklenir.

## 1.5.2 Tam-Toplayıcı (FA:Full Adder)

Tam-toplayıcı (FA:full adder) için ise S (toplam) ve C (elde) çıkışları x,y,z girişleri cinsinden aşağıdaki lojik eşitliklerle gösterilir.

$$S = x \oplus y + z$$

$$C = xy + (x \oplus y)z = xy + (x' + (x'y + xy')z$$

## 1.6 FLİP-FLOPLAR (FF)

Bu devrelerde elemanlar hafıza (memory) içerirler. Ardışıl devre olarak adlandırılırlar. Saat darbeleri (clock pulses) ile çalışırlar. Eşzamanlı (senkron) ve eşzamanlı olmayan (asenkron) olmak üzere iki tipleri vardır. Eşzamanlı olanlar saat darbeleri ile çalışırlar, eşzamanlı olmayanlar ise giriş büyüklüğünün devreye ulaşması ile yeni çıkış değerlerini alırlar.

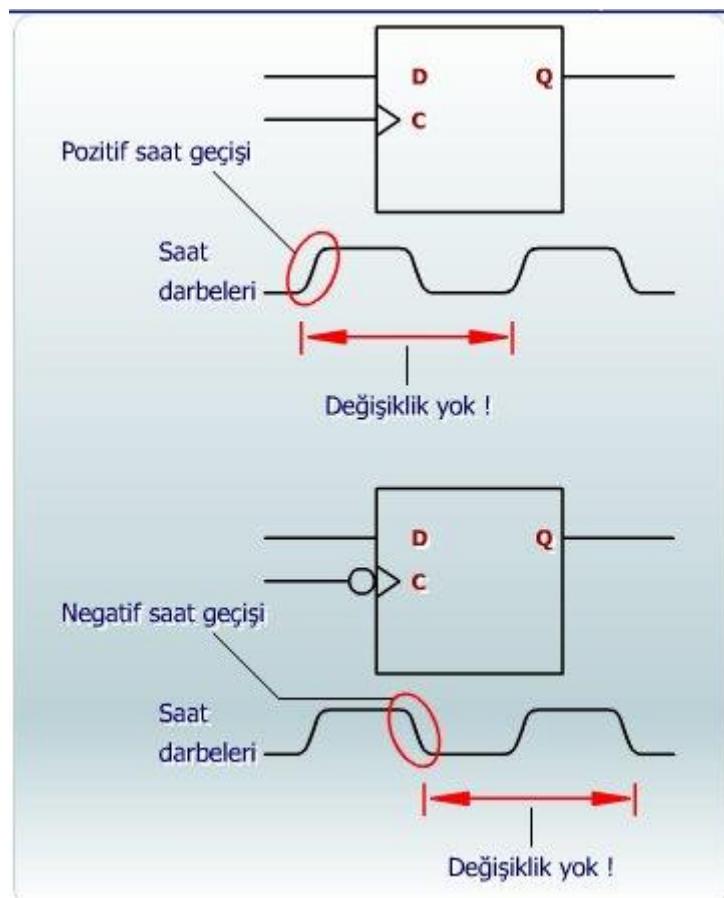
Dört tip FF elemanı vardır: SR, JK, D, T. FF elemanı giriş veya girişlerine uygulanan ikili değer ile şimdiki durumdan (present state) gelecek duruma (next state) geçer. Aşağıda FF'lerin uyarma tabloları verilmiştir.

 <b>SR</b> <input type="checkbox"/> S <input type="checkbox"/> R	 <b>JK</b> <input type="checkbox"/> J <input type="checkbox"/> K																		
<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="2"></th> <th><b>Q (t+1)</b> (gelecek çıkış değeri)</th> </tr> <tr> <th><b>S</b></th> <th><b>R</b></th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td><b>Q (t)</b> değişiklik yok</td> </tr> <tr> <td>0</td> <td>1</td> <td>0 temizle</td> </tr> <tr> <td>1</td> <td>0</td> <td>1 yap</td> </tr> <tr> <td>1</td> <td>1</td> <td>? belirsiz</td> </tr> </tbody> </table>				<b>Q (t+1)</b> (gelecek çıkış değeri)	<b>S</b>	<b>R</b>		0	0	<b>Q (t)</b> değişiklik yok	0	1	0 temizle	1	0	1 yap	1	1	? belirsiz
		<b>Q (t+1)</b> (gelecek çıkış değeri)																	
<b>S</b>	<b>R</b>																		
0	0	<b>Q (t)</b> değişiklik yok																	
0	1	0 temizle																	
1	0	1 yap																	
1	1	? belirsiz																	
<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="2"></th> <th><b>Q (t+1)</b></th> </tr> <tr> <th><b>J</b></th> <th><b>K</b></th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td><b>Q (t)</b> değişiklik yok</td> </tr> <tr> <td>0</td> <td>1</td> <td>0 temizle</td> </tr> <tr> <td>1</td> <td>0</td> <td>1 yap</td> </tr> <tr> <td>1</td> <td>1</td> <td><b>Q'(t)</b> komplement</td> </tr> </tbody> </table>				<b>Q (t+1)</b>	<b>J</b>	<b>K</b>		0	0	<b>Q (t)</b> değişiklik yok	0	1	0 temizle	1	0	1 yap	1	1	<b>Q'(t)</b> komplement
		<b>Q (t+1)</b>																	
<b>J</b>	<b>K</b>																		
0	0	<b>Q (t)</b> değişiklik yok																	
0	1	0 temizle																	
1	0	1 yap																	
1	1	<b>Q'(t)</b> komplement																	
Not: JK FF'da 11 giriş $Q(t) \rightarrow Q'(t+1)$ yapmaktadır.																			
 <b>D</b> <input type="checkbox"/> D <input type="checkbox"/> C	 <b>T</b> <input type="checkbox"/> T <input type="checkbox"/> C																		
<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="2"></th> <th><b>Q (t+1)</b></th> </tr> <tr> <th><b>D</b></th> <th></th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>temizle</td> </tr> <tr> <td>1</td> <td>1</td> <td>1 yap</td> </tr> </tbody> </table>				<b>Q (t+1)</b>	<b>D</b>			0	0	temizle	1	1	1 yap						
		<b>Q (t+1)</b>																	
<b>D</b>																			
0	0	temizle																	
1	1	1 yap																	
<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th><b>T</b></th> <th><b>Q (t+1)</b></th> </tr> </thead> <tbody> <tr> <td>0</td> <td><b>Q (t)</b> değişiklik yok</td> </tr> <tr> <td>1</td> <td><b>Q'(t+1)</b> komplement</td> </tr> </tbody> </table>		<b>T</b>	<b>Q (t+1)</b>	0	<b>Q (t)</b> değişiklik yok	1	<b>Q'(t+1)</b> komplement												
<b>T</b>	<b>Q (t+1)</b>																		
0	<b>Q (t)</b> değişiklik yok																		
1	<b>Q'(t+1)</b> komplement																		

FF'lerin uyarma tabloları

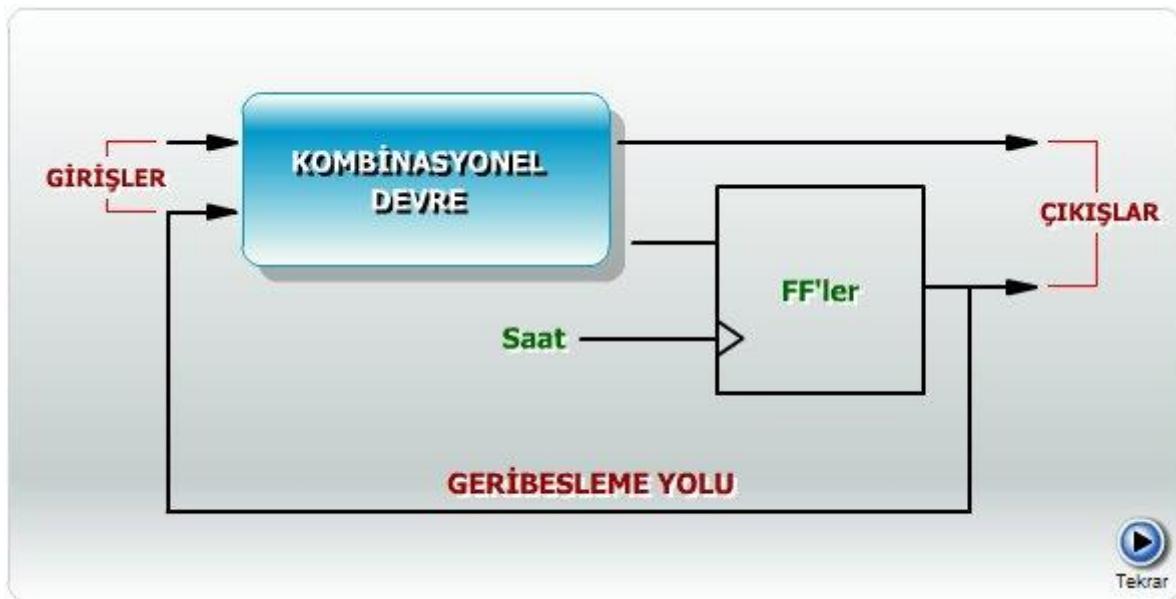
### 1.6.1 Kenar Tetiklemeli ve Efendi-Köle (Master-Slave) Tipi FF'ler

Kenar tetiklemeli ve Efendi-köle (Master-slave) tipi FF'ler yaygın olarak kullanılan tiplerdir. Kenar tetiklemeli FF, yanlışca saat darbelerinin uygulanması ile durum değiştirir. Saat darbelerinin geçişi sırasında durum değişir; diğer anlarda durum sabit kalır. Efendi-köle (master-slave) tipi FF'de giriş değeri ilk saat darbesi geçişinde master çıkışında olur. İkinci darbe geçişinde aynı çıkış köle FF çıkışı olur. Aşağıda kenar tetiklemeli FF'nin çalışması açıklanmaktadır.



### 1.7 ARDİŞİL DEVRELER

Ardışıl devreler FF'ler ve lojik kapıların bağlanmasıından oluşur. Aşağıda ardışıl devrenin blok diyagramı gösterilmiştir. Şekilde FF'nin çıkışından alınan geribeslemenin sisteme giriş olarak uygulandığı izlenebilir.

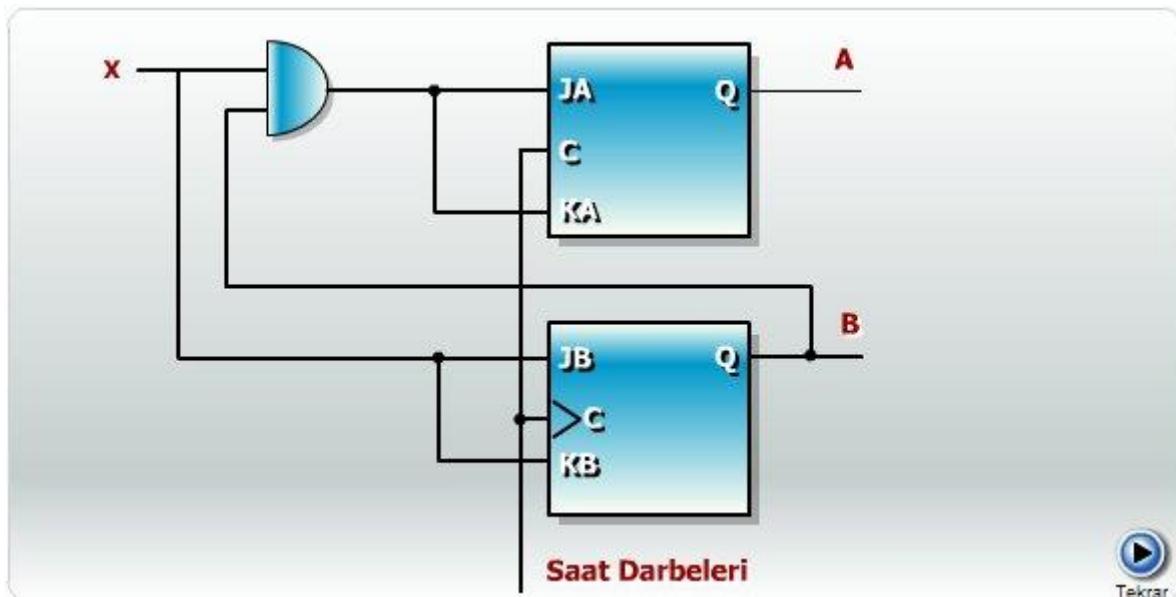


Ardışıl devrenin blok diyagramı

Ardışıl devrenin analizi işlemi, verilen lojik diyagramdan yola çıkarak, giriş eşitliklerinden, uyarma tablosu yoluyla durum diyagramının elde edilmesi olarak özetlenebilir. Tasarım sırasında verilen durum diyagramından lojik diyagramın elde edilmesidir. Tasarımın ara işlemi olarak uyarma tablosu kullanılır. İstenen giriş-çıkış ikili bağıntısını sağlayan FF'lerin giriş eşitlikleri elde edilir.

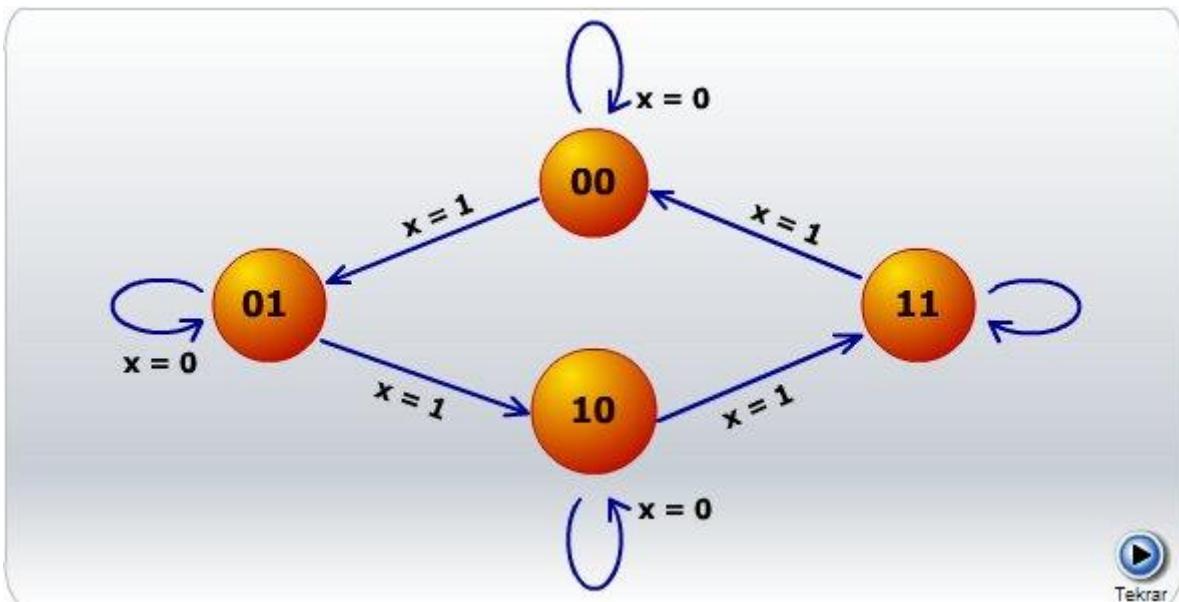
### 1.7.1 Ardışıl Devreler (Devam)

Örnek olarak verilen bir devrenin analizini yapalım.



$$\begin{aligned} J_A &= BX \quad K_A = BX \\ J_B &= X \quad K_B = X \end{aligned}$$

Sonra  $x$  giriş değerleri için durum diyagramı çizelim (durum tablosu da oluşturulabilir). Dört durum arasında geçişler  $x$  değerine göre olacaktır.



**! Sonuç:** Yukarda Lojik diyagramı gösterilen devre bir 2-bit ikili sayıci devresidir.

Analizördeki verilen bir lojik devrenin FF'lerinin giriş eşitliklerinin bulunması ve çıkış eşitliğinin elde edilmesi hatırlatılmıştır. Tasarım örneğinde ise 2-bit eşzamanlı sayıci gösterilmiştir.

## Bölüm Özeti

- Bu bölümde daha önce bilindiği varsayılan temel sayısal sistem kavramları yüzeysel olarak hatırlatılmıştır.

## BÖLÜM - 2

# SAYISAL BİLEŞENLER

Bu bölümde aşağıda sıralanan tümdevre tasarım elemanları inceleneciktir.

- Tümdevrelerin tanıtımı (integrated circuits-IC)
- Dekoder-Enkoder (Decoder-Encoder)
- Çoğullayıcı (multipleksör-MUX)
- Bellek elemanı (Register)
- Kaydırımlı bellek alanı (Shift Register)
- İkili sayıci (Binary Counter)
- Ana bellek birimi (Memory Unit)

## Bölüm Hedefi

Bu haftaki derste, sayısal devre tasarımının daha üst seviyesini oluşturan, sayısal bileşenleri tanıtabileceğiz. Bileşenlerin temel teknolojisi olan tümdevrelerin kısaca tanıtımı

yapılacaktır. Sonra elemanlar sıra ile lojik fonksiyonları ve fiziksel özellikleri göz önüne alınarak tanıtılcaktır. Bu elemanlar dekoder, çoğullayıcı, bellek elemanı (register), kaydıraklı bellek elemanı, ikili sayıcılar ve ana bellek birimi örnekler ile özet olarak tanıtılcaktır. Bu bölümdeki amaç, tümdevre tasarım elemanlarının özetlenmesi ile sonraki derslerde bilgisayar tasarımda blok olarak kullanılacak bu elemanların özelliklerinin gözden geçirilmesini sağlamaktır.

## 2.1 TÜMDEVRELERİN TANITIMI

Tümdevreler (integrated circuits, chip) çok sayıda lojik elemanın gerçeklendiği, küçük, yarı iletken sayısal bileşenlerdir. Küçük bir paket içinde yüzlerce, binlerce lojik kapı, FF gibi elemanı gerçekleyip, bağlantıları bacaklar ile dışarıya aktararak elde edilir.

### TÜMDEVRELERİN KAPASİTELERİNE GÖRE BÜTÜNLEME ÇEŞİTLERİ

- **Küçük ölçekli bütünlere (KÖB)**  
[small scale integration-SSI]
- **Orta ölçekli bütünlere (OÖB)**  
[medium scale integration-MSI]
- **Geniş ölçekli bütünlere (GÖB)**  
[large scale integration-LSI]
- **Çok geniş ölçekli bütünlere (ÇGÖB)**  
[very large scale integration-VLSI]



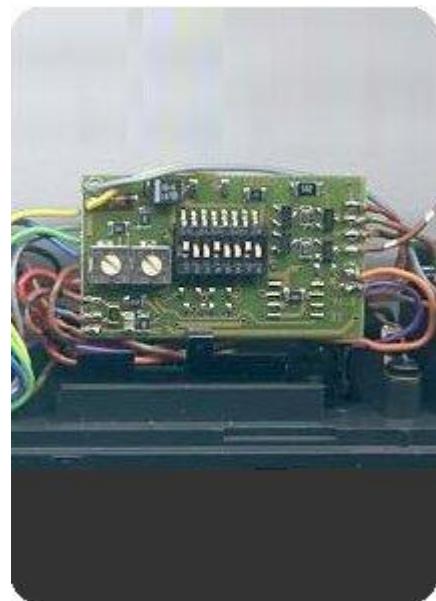
Tümdevreler kapasitelerine göre yukarıdaki gibi çeşitlendirilebilir. KÖB'de 10'dan az sayıda eleman bir paketin içinde bulunur. OÖB'de bu sayı 10 ile 200 arasındadır. GÖB'de ise eleman sayısı 200 ile birkaç binler arasında değişir. Son olarak, ÇGÖB'de ise bir paketteki eleman sayıları kolaylıkla yüzbinler mertebesindedir. Bir fiziksel örnek vermek gerekirse, tırnak yüzeyi boyutundaki yarı iletken üzerinde 400-500 binler mertebesinde lojik kapı gerçeklemektedir.

Tümdevereler birkaç tipik teknikle üretilmektedir: bunlar TTL (transistor, transistor logic-transistor, transistor lojik), ECL (emitter coupled logic-emetör kuplajlı lojik), MOS (metal oxide semiconductor-metal oksit yarı iletken), CMOS (complementary metal oxide semiconductor-komplementer MOS). TTL teknolojisi yıllardır standart olan, 5 volt besleme gerilimi ve lojik 0 ve 1 seviyeleri için 0 volt ve 3.5 volt kullanan, NAND kapıları temel olarak elde edilmiş bir teknolojidir. ECL ise hızın önemli olduğu uygulamalar, örneğin süperbilgisayar ve işaret işleme donanımlarında kullanılan, transistörleri doyuma (saturasyona) girmeyen bir teknolojidir. MOS teknolojisi bileşen yoğunluğunun fazla olduğu bir teknolojidir. n-kanallı (NMOS) ve p-kanallı (PMOS) olmak üzere yanlışca bir çeşit akım taşıyıcının kullanıldığı çeşitleri vardır. CMOS teknolojisi ise bugün yaygın olarak kullanılan, üretimi basit ve düşük güç tüketimi olan bir teknolojidir.

Yukarda belirtilen teknolojilerle üretilen sayısal bileşenler bilgisayar tasarımindan差别 fonksiyonlarının sağlanması sırasında kullanılmaktadır. Aşağıda ise kullanılan bileşenleri daha detaylı olarak inceleyeceğiz.

## 2.2 DEKODER-ENKODER

Dekoder ve enkoder, bilgisayarda ( $n-m$ ) bit dönüşümünü sağlayan kombinasyonel devrelerdir. Bilindiği gibi, bilgisayar ikili sayılar üzerinde hesap yapmaktadır. Giriş ve çıkışta farklı sayıda bit elde etmek önemli bir işlemidir. Bu ise  $n$  bit kodlanmış bilginin  $m=2^n$  bir çıkış bilgisi veya tam tersi bir  $2^n$  bit girişin bir  $n$  bit çıkış bilgisi haline çevrilmesi işlemidir. Burada giriş çıkış kodlamasında  $2^n$  kuvvetlerinin kullanıldığı gözlenmelidir. Ayrıca bu devreler, tümdevre gerçeklemesi ile kullanıldığı için Enable-E girişi ile istenilen devrenin seçimi sağlanır:  $E=0$  tümdevrenin seçilmemesine,  $E=1$  ise seçilip, işleminin kullanılmasına neden olur.

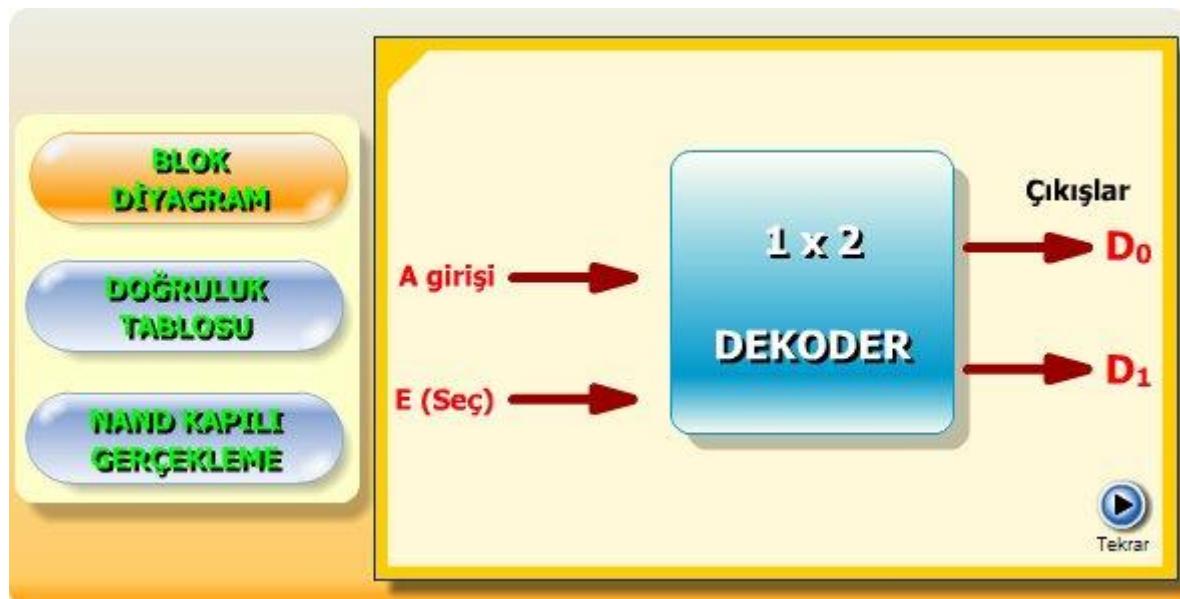


### 2.2.1 Dekoder

Dekoder  $n$  girişli ve  $m=2^n$  çıkışlı bir elemandır.  $n \times m$  dekoder olarak da adlandırılır. Amacı,  $n$  girişte verilen ikili bilgiye göre  $2^n$  çıkıştan birini üretmektir. Aşağıda  $1 \times 2$  dekoderin blok diyagramı, doğruluk tablosu ve NAND kapılı gerçeklemesi (lojik diyagramı) gösterilmiştir. Dekoderin AND, OR, NOR kapıları ile de gerçekleştirilebileceği unutulmamalıdır.

$1 \times 2$  dekoder blok diyagramı, doğruluk tablosu ve NAND kapılı gerçeklemesi girişte uygulanan değere göre (0 veya 1) çıkışlardan birisi seçilir: 0 uygulanırsa 0 çıkışı, 1 uygulanırsa 1 çıkışı seçilir.

! Düğmelere tıklayarak tablo ve şemaları görebilirsiniz.



**BLOK  
DİYAGRAM**

**DOĞRULUK  
TABLOSU**

**NAND KAPILI  
GERÇEKLEME**

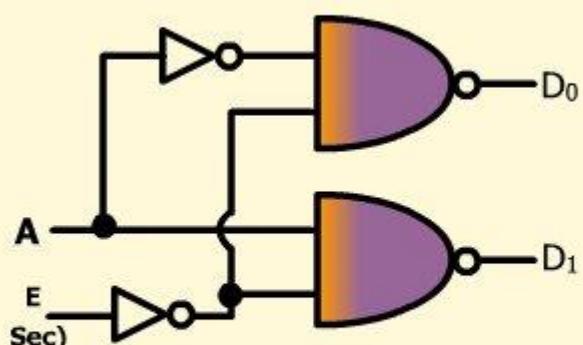
E	A	D <sub>0</sub>	D <sub>1</sub>
0	0	0	1
0	1	1	0
1	X	1	1

0 yada 1

**BLOK  
DİYAGRAM**

**DOĞRULUK  
TABLOSU**

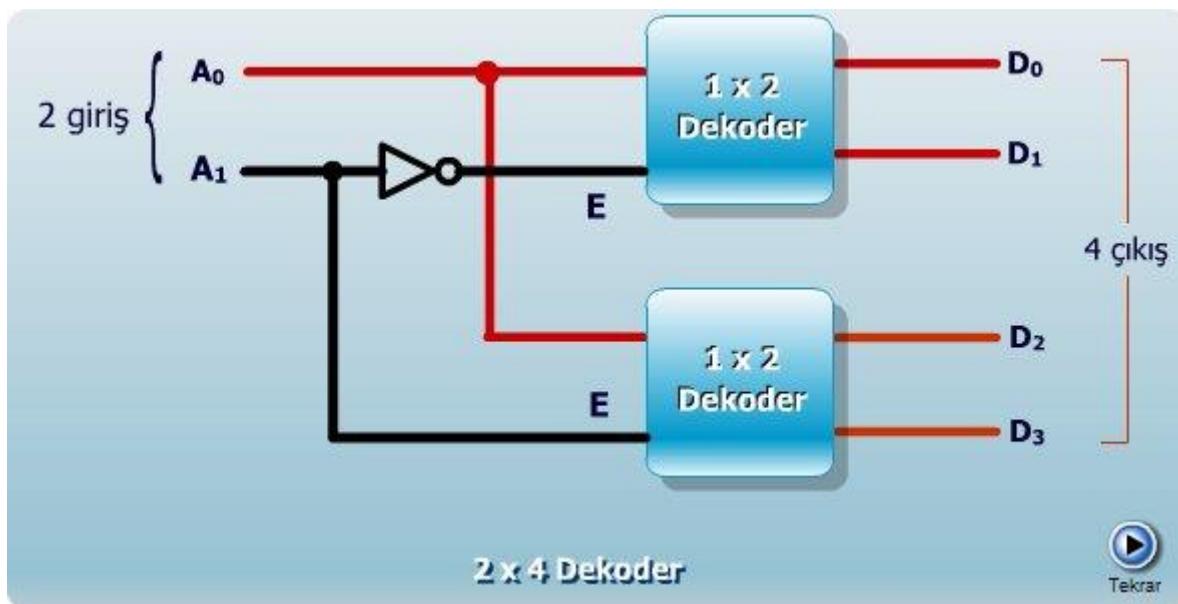
**NAND KAPILI  
GERÇEKLEME**



Tekrar

### 2.2.1.1 Dekoder Uygulaması

Dekoderin aynı zamanda 2x4, 3x8, 4x16 ve daha çok sayıda bit dönüşümü yapabileceği göz önüne alınmalıdır. Bir uygulama olarak 2 adet 1x2 dekoder kullanılarak 2x4 dekoderin fonksiyonunun sağlandığı (kapasite arttırımı) aşağıda gösterilmiştir.



2 adet 1x2 dekoder ile 2x4 dekoder elde edilmesi

2x4 dekoderin çalışması 1x2 dekodere benzer. Bu durumda giriş sayısı 2 çıkış sayısı ise 4'tür. Girişe uygulanan ikili değere karşı düşen çıkış seçilir. Örneğin, giriş değeri 00 ise 00 çıkışı, 10 ise 10 çıkışı seçilir.

## 2.2.2 Enkoder

Enkoder ise  $2^n$  girişi n çıkışa dönüştüren veya dekoderin ters işlemini yapan devredir. Aşağıda 4x2 enkoderin blok diyagramı, doğruluk tablosu ve OR kapıları ile gerçeklemesi gösterilmiştir. Devrede bir zamanda, yalnızca bir giriş "1" olabilir.

**! 4x2 enkoderin blok diyagramı, doğruluk tablosu ve OR kapıları ile gerçeklemesini görmek için düğmelere tıklayınız.**

BLOK  
DİYAGRAMI

DOĞRULUK  
TABLOSU

NAND KAPILI  
GERÇEKLEME

The diagram shows the internal structure of a 4x2 ENDEKODER (Encoder). It has four input lines labeled D<sub>3</sub>, D<sub>2</sub>, D<sub>1</sub>, and D<sub>0</sub> on the left, and two output lines labeled A<sub>1</sub> and A<sub>0</sub> on the right. The central component is a blue rectangular block labeled "ENDEKODER". A "Tekrar" (Repeat) button is located in the bottom right corner of the main panel.

**BLOK  
DİYAGRAM**

**DOĞRULUK  
TABLOSU**

**NAND KAPILI  
GERÇEKLEME**

D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	A <sub>1</sub>	A <sub>0</sub>
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

**BLOK  
DİYAGRAM**

**DOĞRULUK  
TABLOSU**

**NAND KAPILI  
GERÇEKLEME**

Çıklar

Girişler

$$A_0 = D_1 + D_3$$

$$A_1 = D_2 + D_3$$

OR Kapıları



Tekrar

## 2.3 ÇOĞULLAYICI (MUX)

Çoğullayıcı (MUX)  $2^n$  girişli ve 1 çıkışlı kombinasyonel bir devredir. Girişlerden birinin çıkışta seçimi için kullanılır. Bu nedenle enable-E girişi yanında seçim hatları (selection lines) kullanılması gerekmektedir. 2 girişin seçimi için 1 seçim hattı, 4 girişin seçimi için ise 2 seçim hattı gerekeceği hatırlanmalıdır. Aşağıda bir 2x1 çoğullayıcının blok diyagramı, doğruluk tablosu ve gerçeklenmesi gösterilmiştir.

! 2x1 MUX blok diyagramı, doğruluk tablosu ve gerçeklenmesini görmek için düğmelere tıklayınız.

**BLOK  
DİYAGRAMI**

**DOĞRULUK  
TABLOSU**

**LOJİK  
DİYAGRAM**

(seçme girişi)

S

I<sub>1</sub>

I<sub>0</sub>

(girişler)

**2 x 1**

**MUX**

(çıkış)

Y

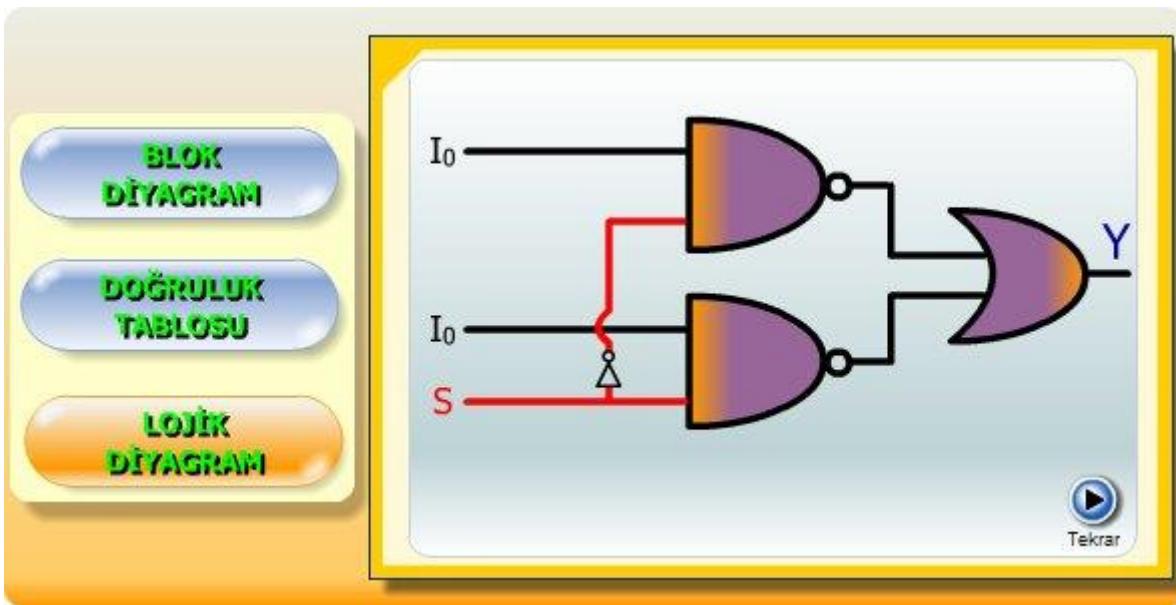
▶  
Tekrar

**BLOK  
DİYAGRAM**

**DOĞRULUK  
TABLOSU**

**LOJİK  
DİYAGRAM**

<b>S</b>	<b>Y</b>
0	I <sub>0</sub>
1	I <sub>1</sub>



2x1 MUX'un seçim hattı hangi girişin çıkışa aktarılacağını tayin eder: Seçim hattı 0 ise 0 girişi çıkışa gönderilir, 1 ise 1 girişi çıkışa gönderilir.

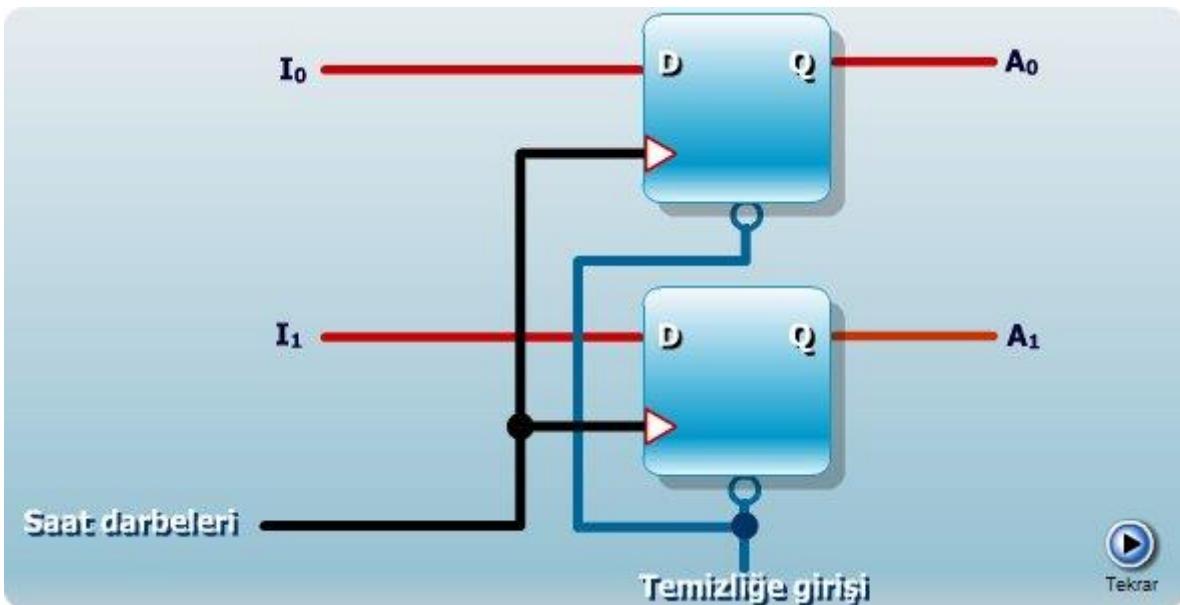
**! Uyarı:** Burada kullanılan 2 "tek" giriş hattı yerine 4'lü veya 8'li giriş hatları aynı sayıda seçim hattı ile kullanılabilir.

## 2.4 BELLEK ELEMANI (REGISTER)

Bellek elemanı ikili bilgi depolama özelliğine sahip bir dizi Flip-Floptan oluşur. Genellikle,  $2^n$  bit FF dizisi bu görevi görür, ikili bilgi kapılarının kontrolü yoluyla FF'lere yazılır veya okunur. Bellek elemanı olarak kullanılabilen çeşitli FF'lerin tipleri arasında, D FF basitliği nedeniyle en sıkılıkla kullanılanlardan birisi olmaktadır. Örneğin, 1011 4-bitlik bir bilgi olup, bunu 4 D FF'sine depo etmek istersek FF girişlerine 1011 uygulamak gereklidir.

Bellek elemanlarının çıkış değerlerinin elde edilmesi okuma (read) işlemi olarak adlandırılır. Bunun için elemanın adresinin bilinmesi ve çıkışının izlenmesi gereklidir. Bellek elemanına yeni bir ikili bilgi yazılması ise yazma (write) veya yükleme (load) olarak adlandırılır. Aşağıdaki şekil 2-bitlik bir  $I_0\ I_1$  ikili bilgisinin 2 D FF'sine doğrudan ve bir yükleme girişi yoluyla kontrollü bir şekilde yüklemesini göstermektedir.

Saat darbelerinin (clock- C) düzenli olarak FF'lere uygulandığı gözlenmelidir. Yükle girişi 1 değerini aldıında, giriş bilgisi FF'lere yüklenir.

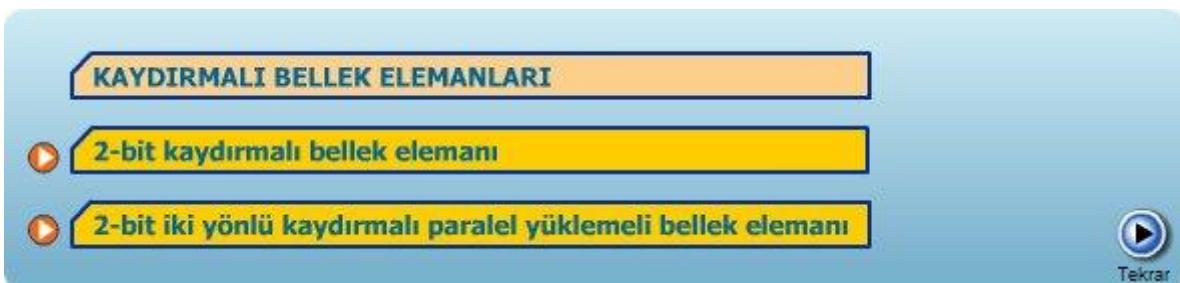


D FF'e  $I_0$   $I_1$  ikili bilgisinin yüklenmesi

Bellek elemanın içeriğini kaybetmemesi için Yükle girişi aktive edilmediği zamanlarda, Saat darbeleri çıkışa ve girişe uygulanır. Her saat darbesinde bellek içeriği "tazelenmiş" (refresh) olur.

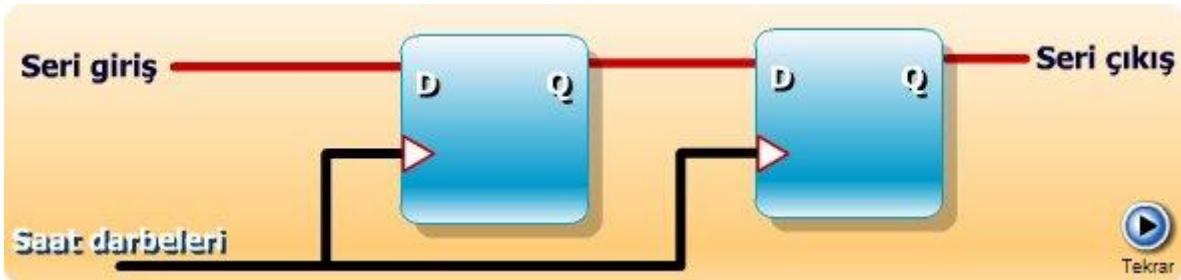
## 2.5 KAYDIRMALI BELLEK ELEMANI

İkili bilgiyi iki yönde kaydırma özelliğine sahip bellek elemanına kaydirmalı bellek adı verilir. Bu başlık altında 2-bit kaydirmalı bellek elemanı ve 2-bit iki yönlü kaydirmalı paralel yüklemeli bellek elemanı ve çalışma prensipleri anlatılacaktır.



### 2.5.1 2-Bit Kaydirmalı Bellek Elemani

İkili bilgiyi iki yönde kaydırma özelliğine sahip bellek elemanına kaydirmalı bellek adı verilir. Bütün bellek elemanlarına ortak saat darbeleri uygulanır ve kaydırma yönüne göre ikili bilgi bir FF ileri veya geriye kayar. Aşağıdaki şekilde 2-bit kaydirmalı bellek elemanı kavramı açıklanmaktadır.

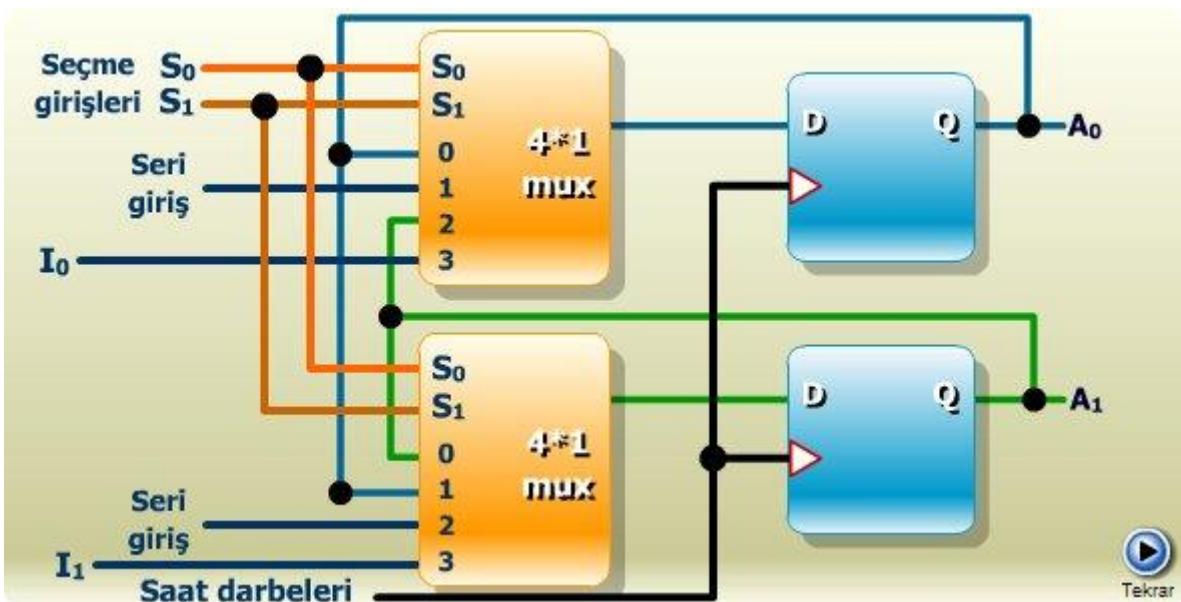


2-bit kaydırmalı bellek elemanı kavramı

Şekile dikkat edilirse, saat darbeleri bellek elemanlarını senkronize (eszamanlı) eder. İlk saat darbesinde ilk FF girişine uygulanan değer, ikinci saat darbesinde ikinci FF'ye üçüncü saat darbesinde ise üçüncü FF'ye uygulanır. Böylece "kaydırma" işlemi gerçekleşmiş olur.

## 2.5.2 2-Bit İki Yönlü Kaydırmalı Paralel Yüklemeli Bellek Elemanı

Aşağıda 2-bit iki yönlü kaydırmalı, paralel yüklemeli bellek elemanı ve çalışma prensibi anlatılmıştır. Girişteki MUX elemanın, 4 girişindeki farklı bilgileri seçim hatları ile seçilerek dört fonksiyonu gerçeklenmiştir: bilgi tazeleme (değişiklik yok), sağ kaydırma, sol kaydırma ve paralel yükleme.  $S_1 S_0$  seçim hatları devrenin fonksiyonunu belirler.



2-bit İki yönlü kaydırmalı, paralel yüklemeli bellek elemanı

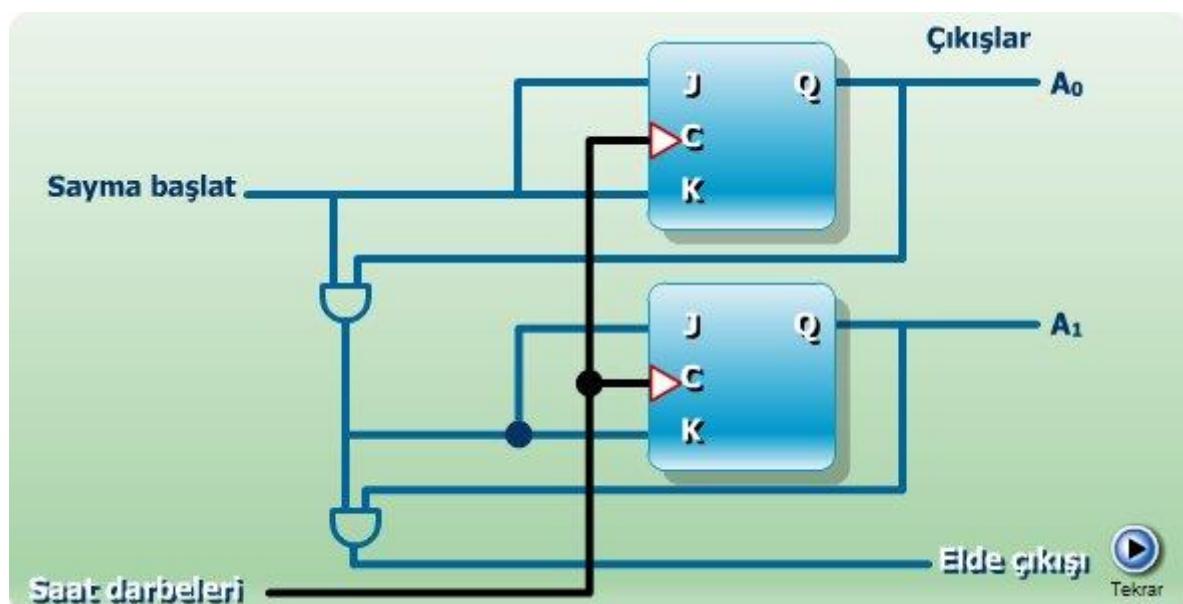
Şekil dikkatle incelenirse sağa ve sola kaydırma işlemi sırasında iki yanda bulunan FF'lerin çıkışlarının seçilen MUX girişine uygulandığı görülebilir. Örneğin,  $A_2$  bitinin sağa kaydırılması sırasında  $A_2$ 'ye ait 4x1 MUX girişine  $A_1$  çıkışı, sola kaydırılması sırasında ise aynı MUX'un girişine  $A_3$  çıkışı uygulanır.

<b>S<sub>1</sub></b>	<b>S<sub>0</sub></b>	<b>Hızlı bellek operasyonu</b>
0	0	<b>Değişiklik yok (aynı çıkış değer)</b>
0	1	<b>Sağ (aşağı) kaydırma</b>
1	0	<b>Sola (yukarı) kaydırma</b>
1	1	<b>Paralel yükleme</b>



## 2.6 İKİLİ SAYICI

İkili sayıçı, önceden belirlenmiş durum dizisini izleyen sayısal devre olarak tanımlanır. Örneğin, 3-bit ikili sayıçı 3 FF'den oluşup, 000, 001, 010, 011, 100, 101, 110, 111 dizisini izler. Sayıcı genel olarak, komplement yapabilen FF'lerden oluşur. Sayıclar, saat darbeli ardışılı sayısal devre yöntemleri ile tasarılanabilirler. Aşağıdaki animasyonda 2-bit eşzamanlı ikili sayıçı görülmektedir.



Yukarıdaki örnekte JK FF'leri kullanılmıştır. Aynı sayıçı paralel load ve eşzamanlı silme (clear) eklenerek tasarlanırsa, JK FF'lerin giriş eşdeğeri aşağıda gösterildiği gibi olur. Aşılurma olarak aşağıdaki şekilde FF giriş eşitlikleri yazılabilir. Örneğin, J<sub>0</sub> girişi için:  $J_0 = I_0 \cdot (\text{LOAD} \cdot \text{Clear}') + \text{INC} \cdot \text{Load}' \cdot \text{Clear}'$  yazılabilir. K<sub>0</sub> girişini aşıtlurma olarak yazınız.

### Paralel yükleme ve eşzamanlı silme özellikleriyle donatılmış 2-bit eşzamanlı ikili sayıcının JK FF tasarım eşitlikleri

	Giriş değeri	Load	Clear	Komplement	Increment
J <sub>k</sub> FF'nin J girişi	$J = I_0 \cdot (\text{yükle} \cdot \text{temizle}') + (\text{temizle}' \cdot \text{yükle}' \cdot \text{arttır})$				
J <sub>k</sub> FF'nin K girişi		$K = I_0' \cdot (\text{yükle} \cdot \text{temizle}') + \text{temizle} + (\text{temizle}' \cdot \text{yükle}' \cdot \text{arttır})$			

Tekrar

NOT: Yukarıdaki deverein "yükle", "temizle", "arttır" ve "değişiklik" yok fonksiyonlarını sağladığını her bir 00,01,10,11 için gösteriniz.

## 2.7 ANA BELLEK ÜNİTESİ

Ana bellek ünitesi, bellek hücreleri ve ikili bilgiyi okumak ve yazmak için kullanılan devrelerden oluşur. İkili bilgiyi, byte (8 bit) ve kelime (2 byte, 4 byte, vb.) formatında saklamak mümkündür. Örneğin, 16 bitlik kelime formatı 2 byte'lık kelimeyi tanımlar.

Ana bellekteki kelimeye adres bilgisi ile ulaşmak mümkün olur. Adres bilgisi ise adres hatları ile belleğe taşınır. k adres hattı ile  $0-2^k$  kelimeye erişmek mümkün olur. Bellek kapasitesi, byte sayısı ile belirlenir. Kullanılan birimleri K:kilo, M:mega ve G: gigadır.  $K=2^{10}$ ,  $M=2^{20}$  ve  $G=2^{30}$  byte kapasitesine karşı gelir. Örneğin, 8G,  $2^{33}$  byte kapasitesine karşı gelir.

### ANA BELLEĞİN TEMEL TIPLERİ

● **Olasılık Erişimli Bellek (random acces memory: RAM)**

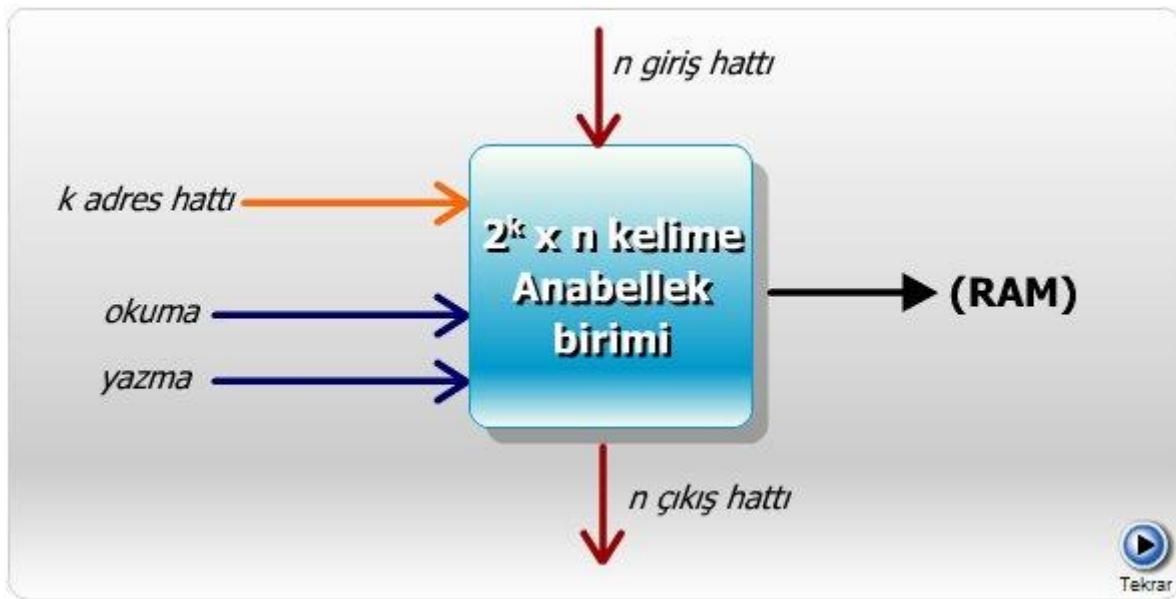
● **Yalnızca Okunabilen Bellek (read only memory: ROM)**

Tekrar

### .7.1 Olasılık Erişimli Bellek (RAM)

Olasılık erişimli belleğin (RAM) istenilen kelimesine olasılık olarak ulaşılabilir, okuma ve yazma yapılabilir. Aşağıda RAM'in blok diyagramı gösterilmiştir. Veri giriş ve çıkış hatları yanında okuma ve yazma girişleri ve adres hatları vardır. Açıklandığı gibi, k adres hattı  $2^k$  kelimenin seçiminde kullanılabilir. Okuma işlemi, adres hatlarına gerekli ikili adres bilgisini gönderdikten sonra, oku girişini aktive ederek yapılır. Yazma işleminde ise, adres bilgisi adres hatlarına, giriş bilgisi giriş hatlarına gönderilir, sonra yaz hattı aktive edilir.

Olasılık erişimli belleğin (RAM) blok diyagramı aşağıdaki gibidir.



Olasılık erişimli belleğin (RAM) blok diyagramı

## 2.7.2 Yalnızca Okunabilir Bellek (ROM)

Yalnızca okunabilen bellekteki (ROM) ikili bilgi okunabilir ama değiştirilemez. RAM de bilgi istenildiğinde değiştirilebilir. İki tip belleğin kullanım alanları bu nedenle fark eder: RAM dinamik bellek olarak kullanılır. ROM ise ikili bilginin nadiren değiştirildiği uygulamalarda kullanılır. Örneğin, mikroprogram kontrolunda mikroprogram belleği olarak ROM kullanılır. ROM'un yapısı, yalnızca okunduğu varsayımlıyla sadece okuma ve adres hattı ile veri çıkış hattından oluşur. Aşağıda yalnızca okunabilen belleğin (ROM) blok diyagramı gösterilmiştir.



Yalnızca okunabilen belleğin (ROM) blok diyagramı

### 2.7.2.1 Yalnızca Okunabilir Bellek (ROM) Çeşitleri

Üç tip ROM kullanılmaktadır:

#### ROM

- Fabrikasyon sırasında programlanıp, kullanıcı tarafından içeriği değiştirilmeden kullanılan yapı.

#### PROM

- Programlanabilen ROM yapısında ROM sigortalarla üretilip kullanıcı tarafından istenildiği gibi bir kez programlanabilir.

#### EPROM-EEPROM

- Elektriksel olarak silinebilen programlanabilen ROM yapısında, kullanıcı gereğinde ROM içeriğini ultraviyole ışığı ile silip, tekrar programlayabilir.

## Bölüm Özeti

- Bu bölümde, sayısal devre tasarıminın daha üst seviyesini oluşturan sayısal bileşenler tanıtılmış, bileşenlerin temel teknolojisi olan tümdevreler üzerinde kısaca durulmuştur.

## BÖLÜM - 3

## VERİ GÖSTERİLİMİ ÖZETİ

Bu bölümde aşağıdaki temel noktalar inceleneciktir:

- Veri tipleri (taban tanımı, onlu, ikili, sekizli, onaltı tabanlı sayı sistemleri ve dönüşümleri, ikili kodlanmış onaltı tabanlı sayılar, alfanümerik gösterim (ASCII))
- Komplement
- Sabit nokta gösterimini (fixed point) ve Hareketli nokta gösterimini (floating point)
- İlgili konular: ikili kodlar ve hata bulan kodlar

## Bölüm Hedefi

Bu haftaki derste, veri gösterimini özetleyeceğiz. Sayısal bilgisayarda kullanılan sayı sistemi olan ikili gösterim ile diğer sayı sistemleri arasındaki ilişkiyi tanıtip, komplement alma çeşitlerini tanımlayacağız. Sabit nokta (fixed point) ve hareketli nokta (floating point) gösterimini vereceğiz. Son olarak, ikili kodlar ve hata bulan kodlardan kısaca bahsederek bölümü tamamlayacağız.

## 3.1 VERİ TİPLERİ

Sayısal bilgisayarın bellek ve işlemcisinde ikili sayılar kullanılır. Bilgisayarın işlemcisindeki hızlı bellek elemanında (register) veri veya kontrol bilgisi bulunur.

**! Bilgisayar kullanılan veri tiplerini görmek için "Devam" düğmesine tıklayınız.**

**Bilgisayar kullanılan veri tipleri:**



- Aritmetik hesaplamalarda kullanılan sayılar
- Veri işlemesinde kullanılan alfabe karakterleri veya kontrol karakterleri
- Özel amaçlı ayrık karakterler (parantez, !,...)

### 3.1.1 Sayı Sistemlerine Ait Örnek

Genel olarak sayı sistemleri,  $r$ -tabanlı (radix, base) olarak veya  $r$  farklı sembolle tanımlanır. Bir sayının gösterdiği miktar, her haneındaki sayının  $r$ 'nin tamsayı kuvveti ile çarpılması ile elde edilir.

**ÖRNEK 1**

**ÖRNEK 2**

- 625.5 sayısını  $r = 10$  tabanında gösteriniz.

**ÇÖZÜM**

**Örnek 1'in Çözümü**

$$625.5 = 6 \times 10^2 + 2 \times 10^1 + 5 \times 10^0 + 5 \times 10^{-1}$$

olarak gösterilir.

**ÖRNEK 1**

**ÖRNEK 2**

- 1110 dizisi ikili sayma sistemindedir. ( $r=2$ ) Bu dizinin  $r=10$  tabanındaki gösterimini elde ediniz.

**ÇÖZÜM**

### Örnek 2'nin Çözümü

$$1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 14$$

olarak gösterilir.

### 3.1.2 Tipik Olarak Kullanılan Tabanlar

! İkili, sekizli ve onaltılı sayı tabanındaki sayıları görmek için  
düğmeye tıklayınız.

**DEVAM**

- ▶ **On tabanlı (onlu,desimal) sayılar:** 0,1,2,3,4,5,6,7,8,9
- ▶ **İki tabanlı (ikili) sayılar:** 0 ve 1
- ▶ **Sekiz tabanlı (sekizli, oktal):** 0,1,2,3,4,5,6,7
- ▶ **Onaltı tabanlı (onaltılı, hegzadesimal):** 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F



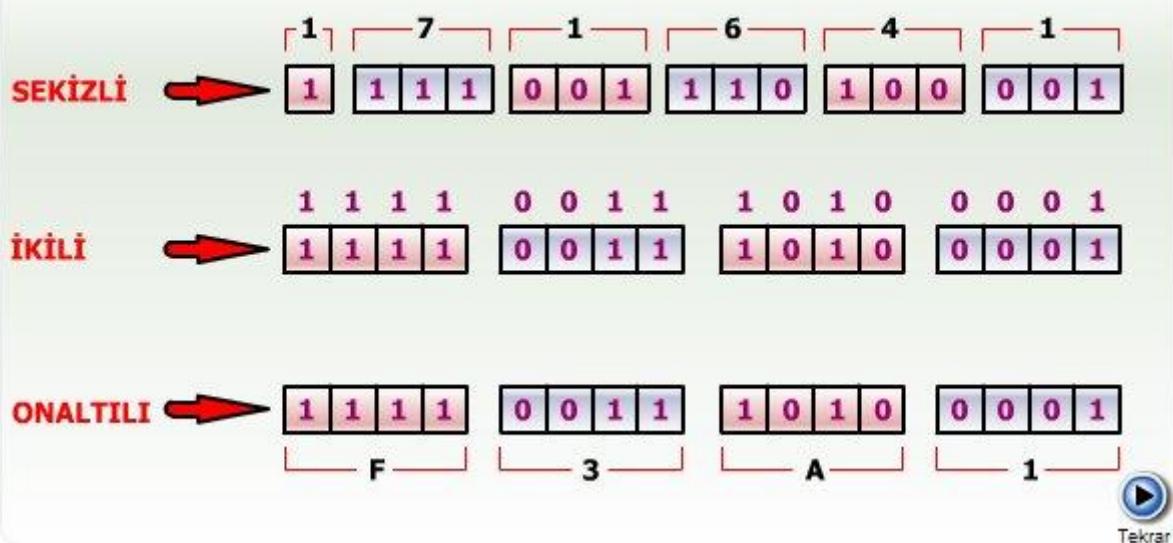
Tekrar

Sayı tabanları arasındaki dönüşümleri, her bir hanedeki kuvvet tabana bölünerek elde edilebilir. Örneğin,

$$(F2)_{16} = F \times 16 + 2 \times 1 = 15 \times 16 + 2 \times 1 = (242)_{10}$$

#### 3.1.2.1 İkili, Sekizli ve Onaltılı Dönüşüm

İkili, sekizli ve onaltılı dönüşüm aşağıda gösterildiği gibi yapılabilir.



### 3.1.3 Kesirli Onlu Sayılar İçin İkili Dönüşüm

Bu dönüşümlerde tamsayı kısmı  $r=2$  ile bölünerek, ondalık kısmı ise  $r=2$  ile çarpılarak yapılır.

#### ÖRNEK

- Onlu 17.6875 kesirli sayısının ikili dönüşümünü elde ediniz.

**ÇÖZÜM**

**Örnek Çözümü**

<b>TAMSAYI=17</b>	<b>KESİRLİ SAYI=0.6875</b>
$  \begin{array}{r}  0.6875 \\  \times 2 \\  \hline  1.3750 \\  \times 2 \\  \hline  0.7500 \\  \times 2 \\  \hline  1.5000 \\  \times 2 \\  \hline  1.0000  \end{array}  $	
$  \begin{array}{c cc}  17 & & \\  \hline  8 & 1 & \\  4 & 0 & \\  2 & 0 & \\  1 & 0 & \\  0 & 1 &  \end{array}  $	$  \begin{array}{r}  (17)_{10} = (10001)_2 \\  (0.6875)_{10} = (0.1011)_2  \end{array}  $

### 3.1.4 İkili Kodlanmış Onlu Sayı Sistemi

Genel olarak, insanlar onlu gösterilime yatkındırlar. Diğer taraftan bilgisayarlar ikili sayı sistemi kullanırlar. Aradaki uyuşumu sağlamak için ikili kodlanmış onlu sayı sistemi (binary coded decimal-BCD) kullanılır. İkili kodlanmış onlu sayıda her bir onlu sayı ikili kodlanmış dört sayı tarafından gösterilir. İkili tabandaki dört sayı 10 farklı elemanı ayırbilir. Örneğin,

$$(0)_{10} = (0000)_2, (5)_{10} = (0101)_2.$$

Onlu tabanlı sayıların ikili tabana dönüştürülmesi ve onlu sayıların ikili kodlanması arasında fark olduğuna dikkat edilmelidir.  $(45)_{10}$  sayısı  $(101101)_2$  sayısı ile gösterilir. Fakat aynı sayı ikili kodlanmış onlu sayı ile gösterildiğinde bu sayı 0100 0101 olarak yazılır.

**(45)<sub>10</sub> SAYISI (101101)<sub>2</sub> İLE GÖSTERİLİR.**

**AYNI SAYI İKİLİ KODLANMIŞ ONLU SAYI İLE GÖSTERİLDİĞİNDE BU SAYI 0100 0101 OLARAK YAZILIR.**

**4 5**

Tekrar 

Pek çok uygulamada sayısal bilgisayarlar yalnızca sayıların değil, aynı zamanda alfabe karakterlerin ve özel karakterlerin de kullanılmasını gerektirir. Alfanümerik karakter seti 10 sayı, 26 harf ve özel karakter setinden oluşur. Böyle bir sette 64 ve 128 eleman (küçük ve büyük harf karakterler) bulunur. Standart alfanümerik ikili kodu ASCII (American Standard Code for Information Interchange) olarak bilinir.

Karakter	İkili Kod
A	100 0001
B	100 0010
C	100 0011
.	.
O	100 1111
P	101 0000
Z	101 1010
.	.
(sıfır) 0	011 0000
1	011 0001
.	.
9	011 1001
boşluk	010 0000
(nokta) :	010 1110

### 3.1.4.1 ASCII Koda Ait Örnekler

ASCII koda ait bir kısım örnek aşağıda verilmiştir.

İkili sayılar sayısal bilgisayarlarda önemli rol oynar. Hızlı bellek yapıları yalnızca ikili bilginin saklanması uygundur. Her türlü işlem bu belleklerde saklanan ikili bilgi üzerinde yapılır.

İkili kodların kullanımı çeşitli uygulamalar için yaygındır. Örneğin, müzik notaları ve satranç taşları için bu kodlar kullanılabilir. Bu kodlar aynı zamanda kontrol bilgisini göstermek için de kullanılır.

## 3.2 KOMPLEMENT

Komplementler verilen bir sayının bilgisayarda tanımlanan çıkarma işlemini kolaylaştırır ve lojik işlemler için kullanılır. Verilen bir sayının  $(r-1)$  komplementi,  $r$  komplementi aşağıdaki gibi bulunabilir.  $N$  sayısı  $r$ -tabanında  $n$  haneden oluşuyorsa

$(r-1)$  komplementi:  $(r^n - 1) - N$

$(r)$  komplementi:  $r^n - N$

olarak tanımlanır. İkili tabanda 1-komplementi (1's complement) ve 2-komplementi (2's complement) olarak elde edilir. 1-komplement, her bir ikili hanenin komplementi alınarak elde edilir. 2-komplement ise her bir hanenin komplementi alındıktan sonra elde edilen sayıya 1 eklenderek elde edilir. Örneğin,  $(1001)_2$  sayısının 1-komplementi  $(0110)_2$  iken 2-komplementi  $(0111)_2$  olarak bulunur.

Yukarıda tanımlanan komplementlerde taban noktası (.) göz önüne alınmamıştır. Eğer sayıda bir nokta varsa komplement almadan önce geçici olarak kaldırılır. Komplement alındıktan sonra taban noktası aynı haneye geri yazılır. Burada, komplement operasyonundan sonra tekrar komplement alınmasının orijinal sayıyı geri oluşturacağından bahsetmek gereklidir:  $r^n - (r^n - N) = N$  orijinal değer olur.

İşaretsiz iki sayının  $r$  tabanında tanımlanan çıkarması aşağıdaki gibi yapılır:  $n$  haneli  $M$  ve  $N$  sayıları  $r$  tabanında işaretsiz iki sayı olsun.  $M-N$  değerinin elde edilmesi için aşağıdaki adımlar uygulanır:

- **M ile N' nin r-komplementini topla:  $M + (r^n - N) = M - N + r^n$**
- **Eğer  $M > N$  ise  $r^n$  eldesi ihmäl edilir.**
- **Eğer  $M < N$  ise,  $r^n$  eldesi oluşmaz ve  $r^n - (N - M)$  elde edilir. Bu ise  $(N - M)$ 'nin r-komplementidir. Sonucu bilinen bir şekilde elde etmek için r-komplement alınır ve önüne negatif işaret eklenir.  $M > N$  örneğinin normal çıkarma işleminden bir farkı yoktur.**

Tekrar

### 3.2.1 $M < N$ Durumu

$M < N$  durumunu bir örnekle inceleyelim:  $M=13250$ ,  $N=72532$  olmak üzere  $13250 - 72532$  işleminde  $-59282$  değeri elde edilir. Bu işlemin adımları aşağıda verilmiştir.

1.  $N$  değerinin 10 komplementi 27468 olarak bulunur.
2.  $M$  ve  $N$ 'nin 10 komplementi toplanır, 40718 elde edilir.
3. Bu toplama işlemi sonunda elde yoktur.
4. Cevap 40718 değerinin 10 komplementi olup, 59282 elde edilir.

Öncelikle her zaman kullandığımız sabit nokta gösterimini göz önüne alalım. Pozitif tamsayıların bilgisayarda gösterilimi sorun olmadığı halde, negatif tamsayıların gösteriminde donanım kısıtlamaları ile karşılaşılır. Tamsayılar, pozitif iseler tek bir gösterimle gösterildikleri halde negatif olduklarında aşağıda yer alan 3 yoldan biri ile gösterilebilirler:

- **İŞARETLİ GENLİK GÖSTERİLİMİ**
- **1-KOMPLEMENT GÖSTERİLİMİ**
- **2-KOMPLEMENT GÖSTERİLİMİ**

Tekrar

Tamsayı, işaretli genlik gösteriminde 0 pozitif, 1 negatif işaret için kullanılıp genlik yanına eklenir. 1- ve 2- komplement gösteriminde, negatif sayı pozitif halinin 1- veya 2- komplementidir. Örneğin,  $+10$  sadece  $(00001010)_2$  olarak gösterilir ama  $-10$  için 3 farklı gösterim vardır:

- **İŞARETLİ GENLİK GÖSTERİLİMİ : 1 0001010**
- **1-KOMPLEMENT GÖSTERİLİMİ : 1 1110101**
- **2-KOMPLEMENT GÖSTERİLİMİ : 1 1110110**

Tekrar

### 3.2.2 Aritmetik Toplama ve Çıkarma

Bu başlık altında 2-komplement yoluyla aritmetik toplama ve çıkarma örneği üzerinde tartışacağız. İki sayının toplamasında, iki sayının işaret bitlerini de göz önüne alarak toplama yapılır. Eğer iki sayının toplamında, negatif sayılar 2-komplement ile

gösterildiğinden sonuç negatif olursa doğrudan 2-komplement şeklindedir. Örneğin, 00000110 (+6) ve 00001101 (+13)'ün toplamı 00010011 (+19)'dur. İşaret biti 0 dır. Sayı pozitif bir sayıdır. Eğer 11111010 (-6) ve 11110011 (-13) sayılarının toplamı 11101101 (-19) olup, 2-komplement şeklindedir. İşaret bitinin (-) olduğuna dikkat edilmelidir.

Çıkarma işlemi 2-komplement sayılar için çok basittir: çıkartılan sayının 2-komplementi alınır ve kendisinden çıkarılan sayıya eklenir. Bu işlem işaret bitlerini de kapsar. Bu kural, çıkarma işleminin aşağıdaki özelliklerinden gelir:

$$(A) - (+B) = (A) + (-B)$$

$$(A) - (-B) = (A) + (+B)$$



Tekrar

Pozitif bir sayının 2-komplementi negatif halini üretir, tersi de doğrudur.

Bir örnekle çıkarma işlemini gösterelim:  $(-6) - (-13) = ?$  işleminin sonucunu bulalım. 2-komplement 8 bit şeklinde sayıları yazarsak  $11111010 - 11110011$  ikili ifadesini elde ederiz. Sonra çıkarma işlemi yukarıda belirttiğimiz kurallarla toplamaya dönüştürülür: çıkarılan (-13) sayısının 2-komplementi (+13) sayısını verir. İkili sayılar bu durumda  $11111010 + 00001101 = 10000111$ . Sonra elde sayısını kaldırırız: sonuç 00000111 (+7) olarak elde edilir.

Taşma (overflow) 2 n bit sayı toplandığında sonuç  $(n+1)$  bit sayı olur. Bu durum taşıma olarak adlandırılır. Sayısal bilgisayarların hızlı bellek kapasiteleri sınırlı olduğu için sonuç n bit hızlı bellekte saklanamaz. Bu durum kullanıcıya bir FF (bayrak) ile belirtilir. Taşma işaret biti pozisyonunda elde girişi ve elde çıkışı değerlerinin gözlenmesi ile anlaşılabilir. Eğer iki elde değeri XOR kapısına uygulandığında, çıkış 1 olursa bir taşıma durumu belirlenmiş olur.

### 3.3 SABİT ve HAREKETLİ NOKTA GÖSTERİMİ

Bilgisayarda sayılar sabit veya hareketli nokta gösterilebilir. Sabit nokta gösteriminde, kesirli sayı, tam ve ondalık kısımlarında baştan kararlaştırılmış hane sayısı ile gösterilir. Bu kural dışı durumlarda değişmez: örneğin, elde edilen sayının ondalık hanesi daha fazla ise kesme (truncation) veya yuvarlatma (roundoff) yollarıyla belirlenen sayı formatına dönüştürülür.

Hareketli nokta gösteriminde, mantis ve üstel kısım kullanılır. Mantis kesirli kısımındır. Üstel kısım ise mantise çarpan olarak gelir. Dolayısıyla, sayı ayarlanabilen bir gösterim ile ifade edilmiş olur:

$m \times r^e$  (örnek:  $+0.5564543 +04$  veya  
 $+0.5564543 \times 10^4$ )

Örnekte, üstel kısmın değeri, desimal noktanın gerçek nokta değerinin dört pozisyon sağında olduğunu gösterir.

İkili hareketli nokta gösterilimi hareketli nokta gösterilimi ile aynıdır yanlışca üstel taban olarak 10 yerine 2 kullanılır. Örnek olarak, + 1001.11 değeri yüzen nokta gösteriliminde 8-bit kesir ve 6-bit üstel ifade ile aşağıdaki gibi gösterilir:

Kesir	Üstel Kismı
01001110	000100

Tekrar

Yüzen nokta sayıları ile yapılan aritmetik işlemler sabit nokta sayıları ile yapılan işlemlerden daha karışiktır. Genel olarak, yüzen nokta işlemleri daha çok zaman alır ve özel donanım gerektirir.

### 3.4 İLGİLİ KONULAR: İKİLİ KODLAR ve HATA BULUNAN KODLAR

Sayısal bilgisayarlar diğer ikili kodları da kullanmaktadır: Gray kodu, 3-fazla (excess-3) kodu, ağırlıklı kod (weighted code), vb. Örneğin, bunlardan Gray kodu analog-sayısal dönüştürmede kullanılmaktadır. Bilindiği gibi sayısal bilgisayarlar ikili sayısal veri üzerinde işlemler yapar. Dönen bir mil göz önüne alalım. Bu milin pozisyonunun sayısal değerlerle gösterilmesi Gray kodu ile çok uygun olur. Analog işaretin, komşu sayısal seviyelere karşı düşen kod gösteriliminde yanlışca bir bitlik değişim olmaktadır. Örnek olarak desimal değerler için Gray kodları aşağıdaki gibi tanımlanır. Görüldüğü gibi her seviyede sadece bir bitlik değişim olmaktadır.

ONLU DEĞER 0 İÇİN GRAY KODU 0000
ONLU DEĞER 1 İÇİN GRAY KODU 0001
ONLU DEĞER 2 İÇİN GRAY KODU 0011
ONLU DEĞER 3 İÇİN GRAY KODU 0010
ONLU DEĞER 4 İÇİN GRAY KODU 0110
ONLU DEĞER 5 İÇİN GRAY KODU 0111

Tekrar

Değişik ikili kodlar çeşitli uygulamalarda kullanılmaktadır: ikili kodlanmış onlu kod (BCD) çok yaygın olarak ikili sistemler ile insan arasındaki bağlantıyi sağlamak için kullanılır. Örneğin 1001 ikili sayısı yerine 9 onlu它的unu kullanmak çok daha anlaşılabilir olmaktadır.

Hata bulan kodlar (aynı zamanda, daha karışık olan hata düzeltken kodlarda) sayısal bilgisayarlarla, iletişim ağlarında, telsiz haberleşme sistemlerinde yaygın olarak kullanılmaktadır. Oluşan hatanın düzeltilmesi değil, varlığı gösterilmektedir. Eğer oluşan hatalar çok sık değilse ve olasılı ise, mesajı tekrar göndermek bir çözüm olabilir.

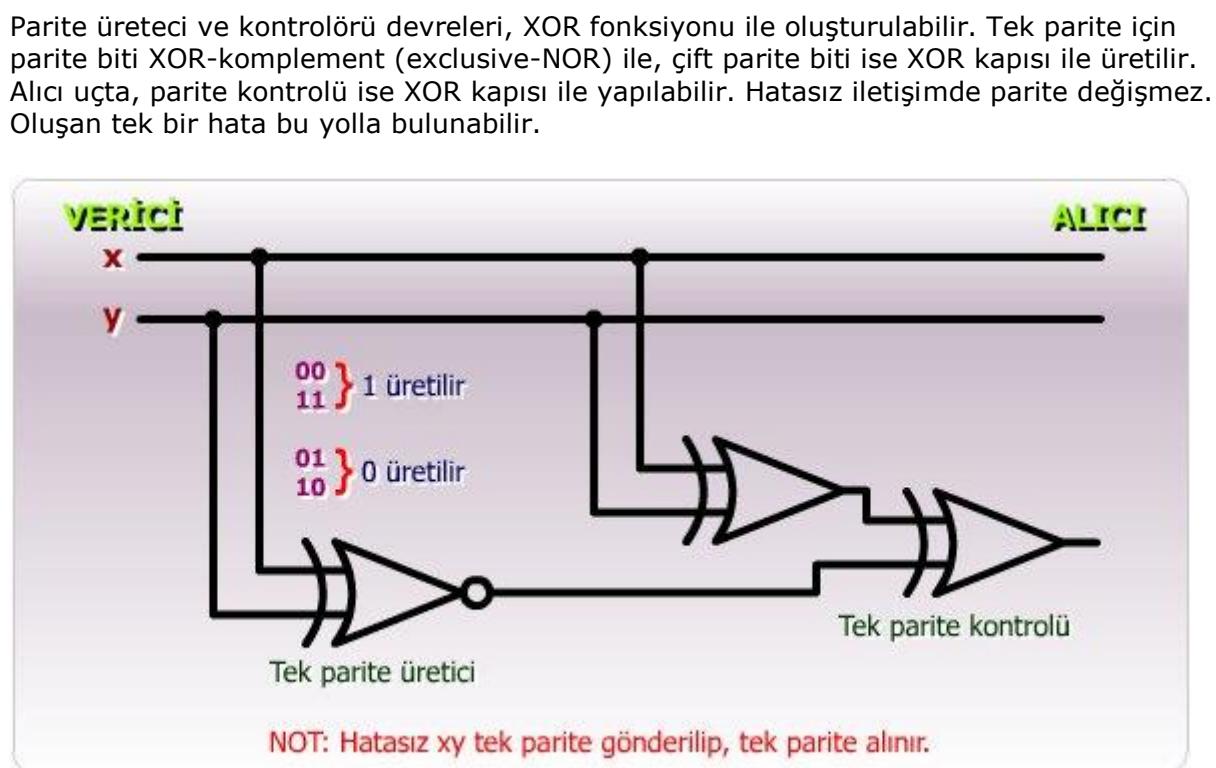
### 3.4.1 Parite Biti

Mesaj:	Parite biti	
ab	P(tek)	P(çift)
00	1	0
01	0	1
10	0	1
11	1	0

Sıklıkla kullanılan hata bulan kod parite biti'dir. Parite biti veriye eklenen extra bittir (redundant) ve görevi mesajdaki bit sayısını tek veya çift yapmaktadır. Mesajda 1 olan bit sayısı sayılır: 1 olan bit sayısı tek ise ve eğer tek parite kullanılıyorsa parite biti  $P(\text{tek})=0$ , 1 olan bit sayısı çift ise,  $P(\text{tek})=1$ . Eğer mesajda çift parite kullanılıyorsa ve mesajdaki bit sayısı tek ise,  $P(\text{çift})=1$ , çift ise  $P(\text{çift})=0$  olur. Parite bitinin tek veya çift değerinden farklı olması mesajda hata olduğunu belirtir. Ayrıca, parite bitinin mesaja eklenebilmesi ve alıcı uça kontrol edilebilmesi için parite üretici ve parite kontrolörü kullanılması gereklidir. Diğer taraftan birden fazla hata olduğunda parite biti bunu bulamaz. 2-bit mesajın parite biti ile hatasının bulunması yan tarafta belirtilen şekilde olur.

İkili bilginin bir yerden diğer bir yere transferi sırasında parite şöyle kullanılır: gönderen tarafta parite üretici ile istenilen parite üretilir ve mesaja eklenir. Alıcı tarafında ise bütün gelen bitler parite kontrolörüne uygulanır. Parite kontrolörü gelen mesajın kullanılan pariteyi sağlayıp sağlamadığını kontrol eder. Eğer doğru parite sağlanmışsa, bu onaylanır. Aksi halde, 1, 3 veya tek hatalar oluşmuşsa bu kestirilir. Çift sayılı hatalar bulunamaz.

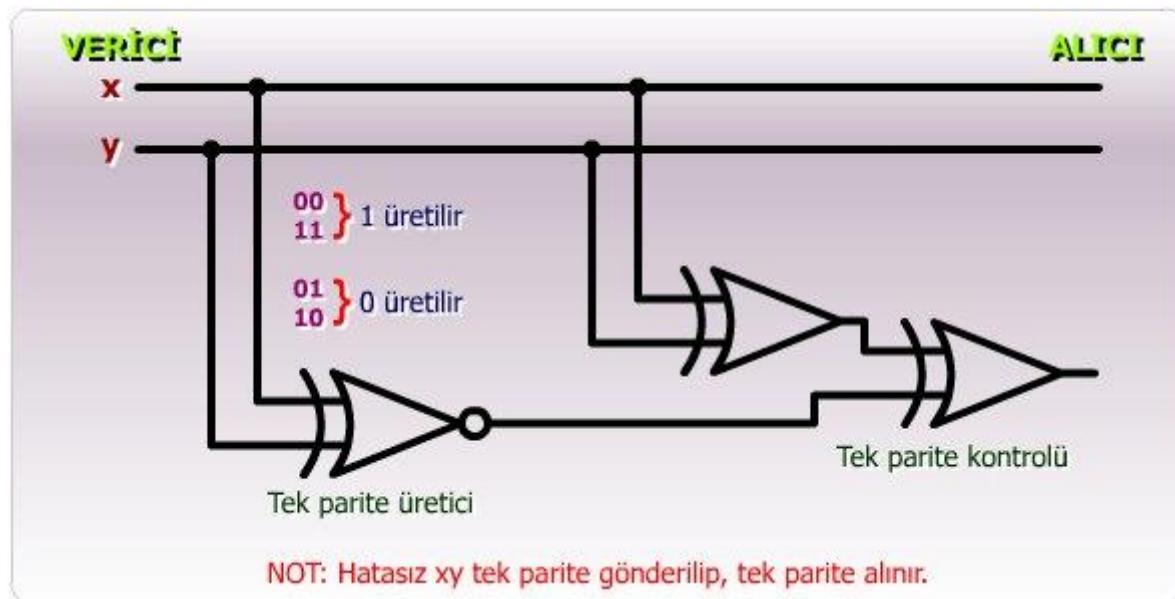
Parite üretici ve kontrolörü devreleri, XOR fonksiyonu ile oluşturulabilir. Tek parite için parite biti XOR-komplement (exclusive-NOR) ile, çift parite biti ise XOR kapısı ile üretilir. Alıcı uça, parite kontrolü ise XOR kapısı ile yapılabilir. Hatasız iletişimde parite değişmez. Oluşan tek bir hata bu yolla bulunabilir.



Tek parite için 2-bitin hatasının bulunması

### 3.4.1.1 Tek Parite İçin 2-bitin Hatasının Bulunması

Parite üretici ve kontrolörü devreleri, XOR fonksiyonu ile oluşturulabilir. Tek parite için parite biti XOR-komplement (exclusive-NOR) ile, çift parite biti ise XOR kapısı ile üretilir. Alıcı ucta, parite kontrolü ise XOR kapısı ile yapılabilir. Hatasız iletişimde parite değişmez. Oluşan tek bir hata bu yolla bulunabilir.



## Bölüm Özeti

- Bu bölümde, veri gösterimini özetlenmiş, sayısal bilgisayarda kullanılan ikili gösterim ile diğer sayı sistemleri arasındaki ilişki tanıtılp, komplement alma çeşitleri tanımlanmıştır. Ayrıca sabit nokta (fixed point) ve hareketli nokta (floating point) gösterimini verilmiş ve son olarak, ikili kodlar ve hata bulan kodlardan kısaca bahsedilmiştir.

## BÖLÜM - 4

# HIZLI BELLEK TRANSFERLERİ ve MİKROİŞLEMLER

Bu bölümde aşağıdaki konular incelenecaktır:

- Sayısal bilgisayarın iç yapısı
- Hızlı bellek transfer dili
- Hat grubu (bus) yoluyla hızlı bellekten veri transferi
- Aritmetik mikroişlemci
- Lojik mikroişlemeler
- Kaydırma mikroişlemleri
- Aritmetik, lojik ve kaydırma (ALU) birimi

## Bölüm Hedefi

Bu bölümde, hızlı bellekler arasındaki transfer işlemleri ve mikroişlemler görecek, hızlı bellek transfer dili ve notasyonu tanımlanacaktır. Hat grubu (bus) yoluyla veri ve hızlı bellek transferleri incelenecaktır. Aritmetik, lojik, kaydırma mikroişlemleri ve aritmetik, lojik, kaydırma (arithmetik logic shift-ALU) birimi tasarımları anlatılacaktır.

## 4.1 SAYISAL BİLGİSAYARIN İÇ YAPISI

Sayısal bilgisayar bir işlevi yerine getirmek için çeşitli modüllerin bağlantılarına gereksinim duyar. Modüller, 2. bölümde tanımlamış olduğumuz sayısal bileşenlerden oluşur. Bu modüller, en iyi olarak, bellek elemanları (register) ve saklanan veri üzerindeki işlemlerle tanımlanabilir. Hızlı bellekte saklanan veriler üzerindeki işlemlere mikroişlemler adı verilir. Bir sayısal bilgisayarın donanım yapısı en iyi aşağıdaki şekilde tanımlanabilir.

- ▶ **Hızlı bellek elemanları (register) ve görevleri**
- ▶ **Hızlı bellek elemanlarında saklanan ikili bilgiye uygulanan mikroişlemlerin dizisi**
- ▶ **Mikroişlemleri başlatacak kontrol bilgisi**

Tekrar 

Bir bilgisayarın mikroişlemleri hızlı bellek elemanı transfer dili yoluyla tanımlanır. Bu dil hatırlanması kolay, yararlı bir dil olup, hızlı bellek seviyesinde tüm aktiviteleri tam olarak tanımlar. Mikroişlemler hızlı bellek seviyesindeki işlemler olup, bu dil mikroişlemlerin gerçekleşmesinde kullanılır. Kontrol fonksiyonları ile birlikte belli koşullarda gerçekleşen hızlı bellek operasyonlarının gösterilimidir.

## 4.2 BELLEK ELEMANI (REGISTER) TRANSFER DİLİ

Bellek elemanı transfer dilinin temel sembollerini aşağıdaki gibi özetlenebilir.

Kullanılan simbol	Tanımı	Örnekler
Harfler	Bellek elemanı	R2, PC, IR
Parantez	Bellek elemanın bir bölümü	R2 (8-15), R2(H)
Ok<=	Bilgi transferi	R2<=R3
Virgül	İki mikroişlemi ayırma	R2<=R3, R4<=R5

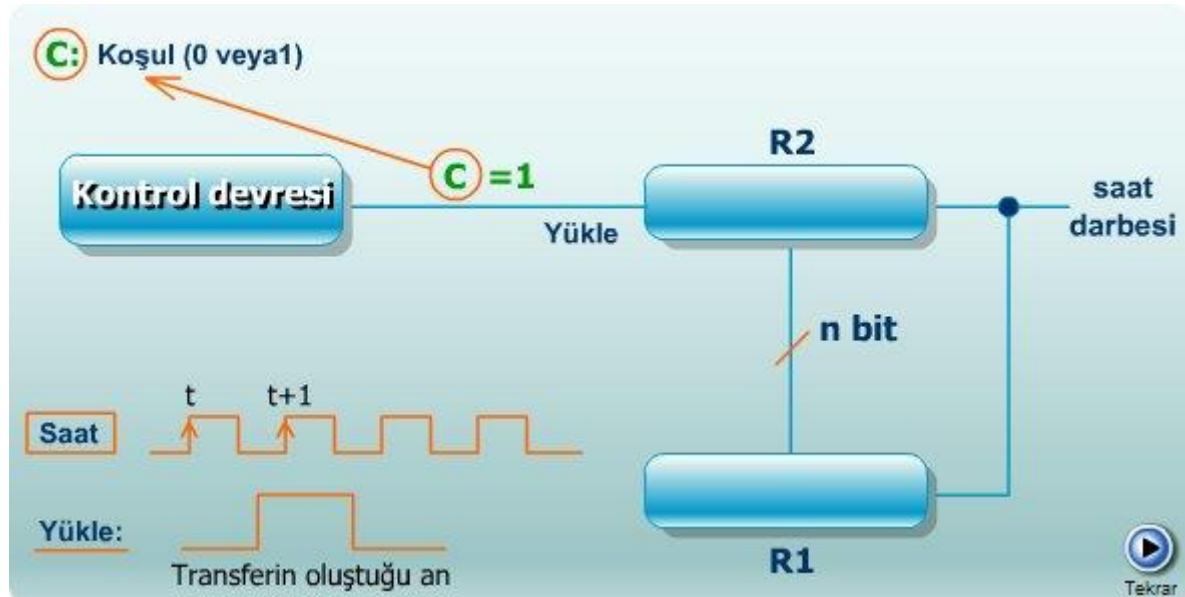
Tekrar 

### 2.1 R3'den R2'ye İkili Bilgi Transferi

Bellek elemanı (Register) transfer dilinin ilk örneğini verip, detayları açıklayalım:

C:R2<=R3

Burada C kontrol fonksiyonu veya koşulu,  $R2 \leq R3$  ise  $R3$ 'ten  $R$ 'ye veri transferini gösterir. Kontrol koşulu iki nokta ile sonlandırılır. Koşul sağlanırsa (if-then), ardından gelen transfer gerçekleştirilir. Aşağıdaki şekilde C koşulu sağlandığında, yükle girişi aktive olur (lojik 1). Saat darbesinin yükselen kısmında (kenar tetikleme) yükle=1 göz önüne alınır ve  $R3$ 'teki ikili bilgi  $R2$ 'ye yüklenir.



$C=1$  koşulunda  $R3$ 'den  $R2$ 'ye ikili bilgi transferi

Virgül ile aynı zamanda gerçekleşen birden fazla sayıda operasyonu ayırmak mümkündür:

D:  $R2 \leq R3$ ,  
 $R4 \leq R5$

İfadesi iki transfer ifadesinin aynı saat darbesinde gerçekleştiğini gösterir. Aynı zamanda gerçekleşebilen operasyonlar kenar tetiklemeli FF'lerle mümkün olabilir. Saat darbelerinin kenarları gerçekleşen işlemlerin zamanlarını belirler.

### 4.3 HAT GRUBU YOLUYLA BELLEKTEN VERİ TRANSFERİ

Tipik bir bilgisayarda bellek elemanları (register) bir grup hattan oluşan bağlantılarla birbirine bağlanılmışlardır. 8 bitlik bir hızlı bellek elemanı, diğer 8 bitlik hızlı bellek elemanına 8 ayrı tel ile bağlanmış olarak düşünülebilir. Bir elemanın her bir biti, diğer elemanında karşı düşen bite bağlanmış olarak düşünülebilir. Böylece elemanlar arasında hat grubu (bus) bağlantısı yapılmış olur. Bu kısımda genel hat grubu bağlantısı oluşturmanın iki yolu bahsedeceğiz:

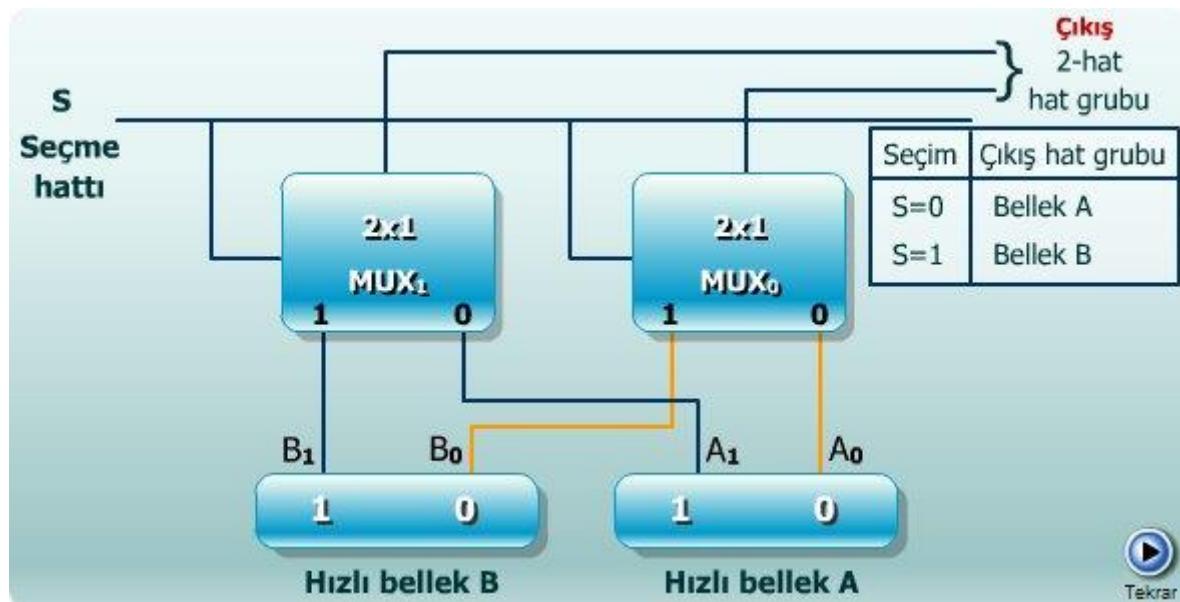
▶ MUX (multiplexer bus) yoluya

▶ Üç durumlu (tri-state gate) kapı yoluya

Tekrar

### 4.3.1 MUX (Multiplexer Bus) Yoluyla Genel Hat Grubu Bağlantısının Yapılması

Aşağıda 2 bellek elemanının 2 MUX ile genel hat grubu bağlantısının yapılması açıklanmıştır. Girişteki belleklerden R1'in bitleri MUX'ların ilk bitlerine bağlanmış olup, seçim hattı S1'in seçilmesi ile genel hat grubuna ulaşırlar.



$S_1$	$S_0$	Seçilen hızlı bellek
0	0	A hızlı belleği
0	1	B hızlı belleği
1	0	C hızlı belleği
1	1	D hızlı belleği

Bellek elemanları arasında ikili bilginin transferinde ara adım olduğuna dikkat edilmelidir: Eğer  $R2 \leq R3$  transferi gerçekleştirilecekse

hat grubu  $\leq R3$ ,  $R2 \leq$  hat grubu

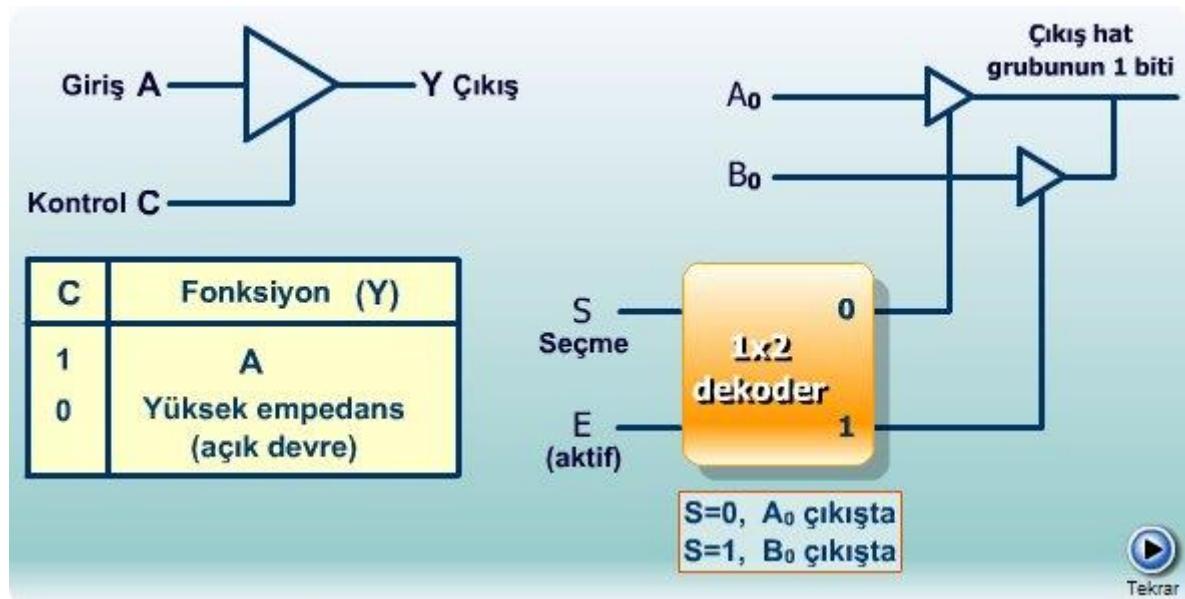
transferleri gerekli olur. 4 adet 4 bitlik hızlı belleği ortak hat grubuna bağlamak için 4 adet 4x1 MUX gereklidir. Bu MUX'ların 2 seçim hatları ortak olarak bağlanır. Aşağıdaki hat grubu fonksiyon tablosuna göre 4 hızlı bellekten birisi seçilir.

Burada dikkat edilirse her 4x1 MUX'un çıkışları ortak hat grubunun bir bitini oluşturur.

### 4.3.2 Üç-Durumlu Kapılar Yoluyla Genel Hat Grubu Bağlantısının Yapılması

Üç-durumlu kapılar, genel hat bağlantılarında yaygın olarak kullanılan elemanlardır. İki durumları normal lojik devre ödevi görür. Eğer eleman bir üç-durumlu koruyucu (buffer) ise girişteki 0 ve 1 lojik seviyeler aynen çıkışa aktarılır. Üçüncü durumu ise "yüksek empedans" durumudur. Bu durumda giriş ve çıkış bağlantısı kesilir, izolasyon oluşturur.

Aşağıda üç-durumlu koruyucu devre elemanı ve 1x2 dekoder kullanılarak oluşturulan 2 giriş bitinin nasıl çıkışa bağlandığı gösterilmiştir.



Üç durumlu önbelleğin ortak hat grubu 1x2 dekoder yardımıyla oluşturulması yukarıda gösterilmiştir. Şekilde 1x2 dekoder üç durumlu önbelleklerden birini seçer ve çıkışında giriş değerinin elde edilmesini sağlar.

Benzer şekilde 2x4 dekoder kullanılırsa girişteki 4 bitten bir tanesinin çıkış biti olarak seçilmesi sağlanır. Bu işlem alıştırma olarak öğrenciye bırakılmıştır.

Çıkış hat grubunun n bitinin oluşturulması için yukarıda anlatılan çekirdek yapının  $(nx2^n)$  dekoderle seçilen üç durumlu önbelleklerin n tanesinin tekrar edilmesi gereklidir.

Ana bellek transfer işlemleri diğer önemli ikili bilgi transfer işlemleridir. Bu işlemler Okuma ve yazma işlemleri olarak tanımlanabilir. Ana bellek işlemleri, hızlı bellek elemanları AR (address register) ve DR (data register) yoluyla yapılır.

Okuma:  $DR \leftarrow M[AR]$

Yazma:  $M[AR] \leftarrow R2$

şeklinde tanımlanır. AR ana belleğin adresini tutar, DR ise bu adrese yazılacak veya okunacak ikili bilgiyi tutar.

## .4 ARİTMETİK MİKROİŞLEMLER

Aritmetik mikroişlemler, Aritmetik operasyonların hızlı bellek elemanları yoluyla gerçekleştirilmesi olarak tanımlanabilir. Bu bölümde, tüm aritmetik işlemler yerine tam ve minimal aritmetik işlemleri göz önüne alacağız. İkili sayıarda, çarpma, toplama ve sola kaydırma ile, bölme ise çıkarma, sağa kaydırma ile gerçekleşebileceğinin burada sadece toplama ve çıkarma gibi temel işlemleri sayısal bileşenlerle gerçekleyeceğiz. Aritmetik mikroişlemler aşağıdaki tabloda gösterilmiştir:

$R4 \leq R2 + R3$	Toplama
$R4 \leq R2 - R3$	$R2$ eksi $R3$
$R4 \leq R4'$	Komplement (1-komplement)
$R4 \leq R4' + 1$	Negatif $R4$ (2-komplement)
$R4 \leq R2 + R3' + 1$	Çıkarma
$R4 \leq R4 + 1$	Arttırma
$R4 \leq R4 - 1$	Eksiltme

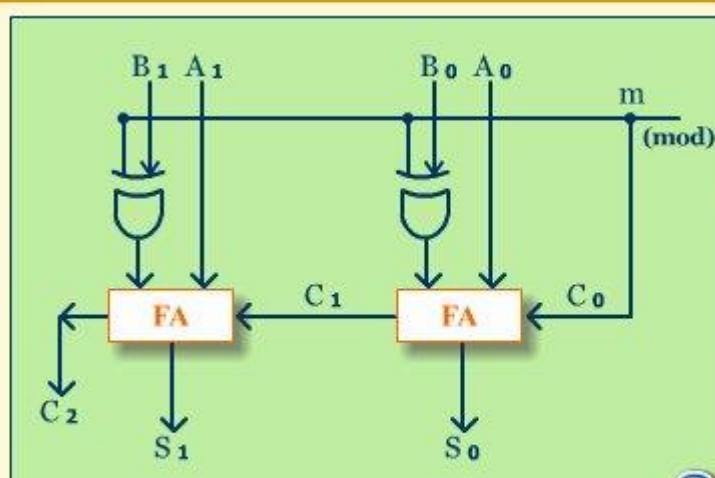
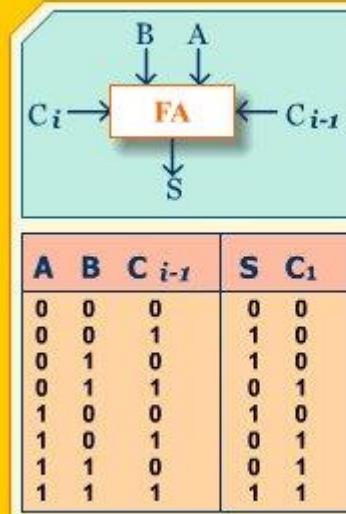


Toplama-çıkarma işlemi yapan bir devre için operandları, sonucu tutan hızlı bellekler ve işlemi yapan işlemci kısmı gereklidir.

Toplama işlemi yapmak için tam toplayıcı (full adder-FA) kullanılır. FA, elde (carry) girişi kullanarak ikili sayıları topladığı için 1-bitlik toplama işlemi yapmak için uygun olur. Yarım toplayıcı (half adder-HA) elde girişi kullanmaz.

#### 4.4.1 2-Bitlik Toplama-Çıkarma İşlemi

2 ikili sayının çıkarma işlemi, bir sayı ile diğer sayının 2-komplementinin toplanması olarak gerçekleştirilebilir. Bu durumda toplama ve çıkarma için tek bir devre kullanılabilir. Aşağıdaki şekilde FA devresinin blok diyagramı ve 2-bitlik toplama-çıkarma devresinin blok diyagramı gösterilmiştir.



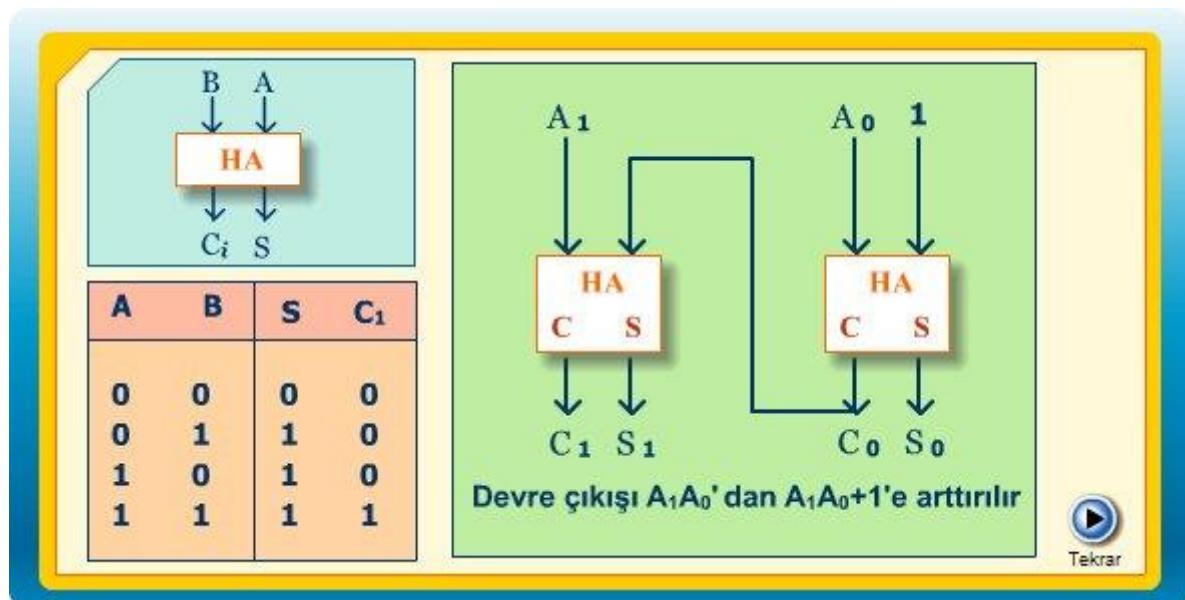
Tekrar

FA devresinin 2 girişinden A<sub>i</sub> ve B<sub>i</sub>'yi, elde C<sub>i-1</sub> ile toplayarak S<sub>i</sub> toplamını ve C<sub>i</sub> eldesini elde etmek mümkün olur. N-bitlik toplama devresi hücreleri yan yana bağlayarak elde edilir. Aynı devreye, çıkarma işlemini eklemek için M (mod) girişi kullanılır. M=0

olduğunda devre toplama işlemi yapar. Çünkü  $B_i \text{ XOR } 0 = B_i$  dir.  $M=1$  olduğunda  $B_i \text{ XOR } 1 = B_i'$  ve  $C_{i-1}=1$  olduğundan  $B_i$ 'nin 2-komplementi üretilir ve devre çıkarma işlemi yapar.

#### 4.4.2 2-Bitlik Arttırma İşlemi

İkili artırma (binary incrementer), hızlı bellek elemanın içeriğini 1 artırmaktır. Örneğin, 4-bit hızlı bellek elemanın içeriği 0101 ise artırma ile 0110 olur. Bu mikroişlem ikili sayıcı ile kolaylıkla gerçekleştirilebilir. Bu durumda bir saat darbesi gecikme yaratır. Bir diğer gerçekleme yöntemi ise kombinasyonel ( FF elemanı kullanmadan) HA devresi kullanmaktadır. Aşağıda 2-bitlik artırma işlemi 2 HA devresi ile gerçekleştirilmiştir. Bu işlemin gecikmesi ihmali edilebilir.

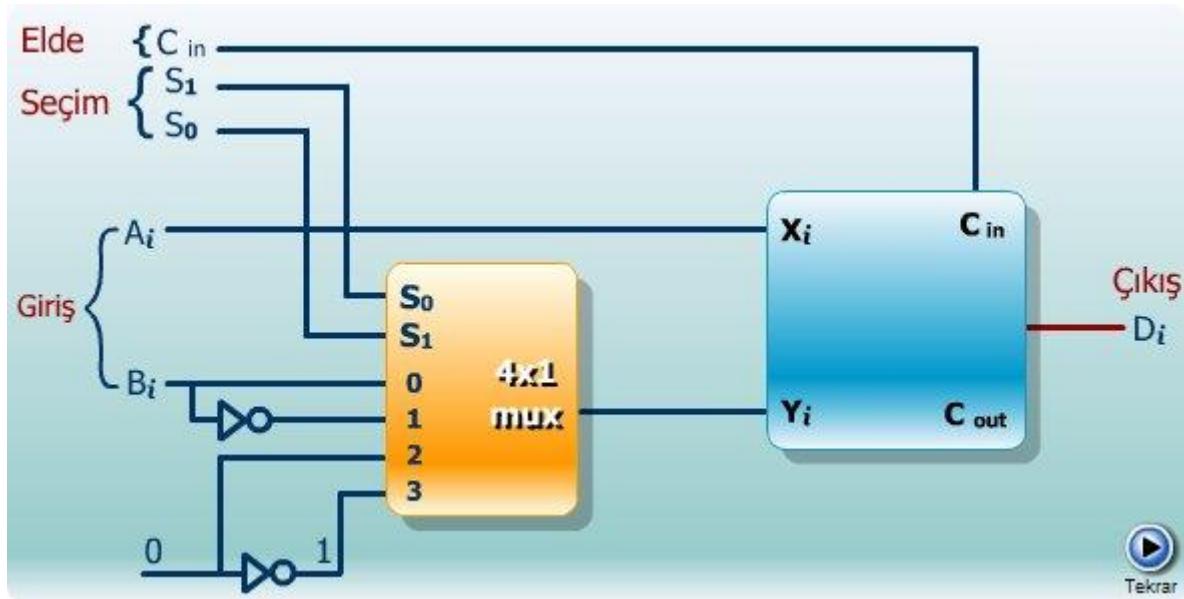


#### 4.4.3 1-Bitlik Aritmetik Devre Tasarımı

Bu bölümün başında tanımlanan aritmetik mikroişlemleri yapabilen 1-bitlik bir devre tasarımı aşağıda gösterilmiştir. Devre bir FA ve bir 4x1 MUX devresinden oluşmaktadır. Aritmetik devre çıkışı aşağıdaki toplam yoluyla oluşturulur:

$$D_i = A_i + Y_i + C_{in}$$

Girişler A<sub>i</sub>, B<sub>i</sub> ve elde C<sub>in</sub>, MUX'un seçme hatları S<sub>1</sub> ve S<sub>0</sub> yoluyla aritmetik mikroişlemler tablosunda tanımlanan işlemlerin gerçeklemesini sağlar.



#### 4.4.3.1.1 Aritmetik Mikroişlemciler Tablosunda Aritmetik Devrenin Gerçekleştirdiği İşlemler

<b>S1</b>	<b>S0</b>	<b>Y<sub>i</sub></b>	<b>D<sub>i</sub> = A<sub>i</sub> + Y<sub>i</sub> + C<sub>in</sub></b>	<b>Mikroişlem</b>
0	0	B <sub>i</sub>	D <sub>i</sub> = A <sub>i</sub> + Y <sub>i</sub>	Toplama ( C <sub>in</sub> =0 ,1)
0	1	B <sub>i</sub>	D <sub>i</sub> = A <sub>i</sub> + Y <sub>i</sub> '	Çıkarma (C <sub>in</sub> =0 ,1)
1	0	0,0	D <sub>i</sub> = A <sub>i</sub> + C <sub>in</sub>	Transfer, arttırma (C <sub>in</sub> =0 ,1)
1	1	1,1	D <sub>i</sub> = A <sub>i</sub> - C <sub>in</sub>	Azaltma, transfer (C <sub>in</sub> =0 ,1)

MUX seçim hatları  $S_1 S_0 = 0 0$ , eldeye ( $C_{in} =0 ,1$  veya eldeli/eldesiz) göre toplama işlemi tanımlar. Seçim hatları  $S_1 S_0 = 0 1$ , ödünçe ( $C_{in} =0 ,1$  veya ödünçle/ödünsüz) göre çıkarma işlemi tanımlar.  $S_1 S_0 = 1 0$ , değeri  $C_{in} =0 ,1$  a göre transfer ve artırma işlemlerini tanımlar. Son olarak da,  $S_1 S_0 = 1 1$ , değeri  $C_{in} =0 ,1$  'e göre azaltma ve transfer işlemlerini tanımlar.

n bitlik aritmetik devre tasarımı, yukarıda açıklanan 1 bitlik aritmetik devrelerin birbirine paralel olarak bağlanmasıyla sağlanır:  $C_{in}$ ,  $S_1$ ,  $S_0$  girişleri tüm bitler için ortak olacaktır. Ayrıca komşu bitlerin  $C_{in}$  ve  $C_{out}$ 'ları zincirleme olarak birbirine bağlanacaktır. En anlamlı bitin  $C_{out}$ 'u tüm devrenin  $C_{out}$ 'u olarak tanımlanacaktır.

Genel olarak, n bit aritmetik devrede 1-bitlik birimler arasında  $C_{in}$  ve  $C_{out}$  değerleri yoluyla ilişki sağlandığı gözlenir.

## 4.5 LOJİK MİKROİŞLEMLER

Lojik mikroişlemeler hızlı bellek elemanlarında depolanan bit dizilerine uygulanan lojik operasyonları tanımlar. Bu operasyonlar her bit için bağımsız olup, her bir bite "bit bazında" uygulanır. Aritmetik mikroişlemelerde elde ve ödünç bitleri ile bir sonraki bite

gönderilen bilgi bu işlemlerde bulunmaz. Örneğin, hızlı bellek R2 ve R3 de 1010 ve 0110 ikili bigileri bulunsun. Aritmetik toplama ve lojik AND işlemleri sonucu aşağıdaki bilgiler elde edilir:

1010	1010
0110	0110
<b>Toplama 1 0000 AND 0010</b>	
(elde)	
	
Tekrar	

Hızlı bellek transfer dilinde tanımladığımız + operasyonu, ifadenin koşul kısmında OR lojik işlemi, bellek transfer kısmında ise toplama işlemi olarak değerlendirilir, bu kısımda lojik OR V ile gösterilir:

S+T: R2 <= R2+R3, R7 <= R2VR5

## 4.5 LOJİK MİKROİŞLEMLER

Lojik mikroişlemler hızlı bellek elemanlarında depolanan bit dizilerine uygulanan lojik operasyonları tanımlar. Bu operasyonlar her bit için bağımsız olup, her bir bite "bit bazında" uygulanır. Aritmetik mikroişlemlerde elde ve ödünc bitleri ile bir sonraki bite gönderilen bilgi bu işlemlerde bulunmaz. Örneğin, hızlı bellek R2 ve R3 de 1010 ve 0110 ikili bigileri bulunsun. Aritmetik toplama ve lojik AND işlemleri sonucu aşağıdaki bilgiler elde edilir:

A	B	F <sub>0</sub>	F <sub>1</sub>	F <sub>2</sub>	F <sub>3</sub>	F <sub>4</sub>	F <sub>5</sub>	F <sub>6</sub>	F <sub>7</sub>	F <sub>8</sub>	F <sub>9</sub>	F <sub>10</sub>	F <sub>11</sub>	F <sub>12</sub>	F <sub>13</sub>	F <sub>14</sub>	F <sub>15</sub>
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	1	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

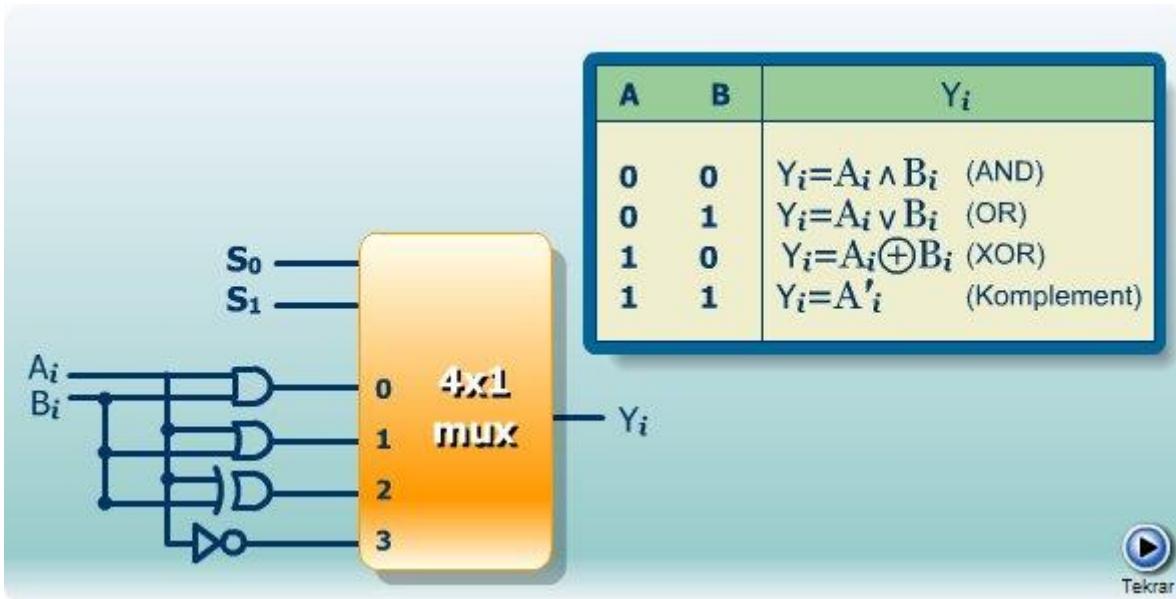
Hızlı bellek transfer dilinde tanımladığımız + operasyonu, ifadenin koşul kısmında OR lojik işlemi, bellek transfer kısmında ise toplama işlemi olarak değerlendirilir, bu kısımda lojik OR V ile gösterilir:

S+T: R2 <= R2+R3, R7 <= R2VR5

### 4.5.2 1-bitlik Lojik Mikroişlem Biriminin Gerçeklenmesi

Boole fonksiyonlarının bilinen bir diğer özelliği ise, sadece bazı temel fonksiyonlarla diğer bütün işlemlerin yapılabilmesidir: örneğin, sadece AND, OR, XOR ve NOT kullanılarak diğer işlemler gerçekleştirilebilir.

Aşağıda 1-bitlik lojik mikroişlem biriminin gerçeklenmesi gösterilmiştir.



Devrede 4x1 MUX'un girişlerindeki AND, OR, XOR ve NOT fonksiyonlar seçme hatları  $S_1$  ve  $S_0$  ile seçilerek çıkışa verilir. n-bit için bu hücrelerden n tanesinin kullanılacağı açıkları.

Yukarıdaki şekilde görüldüğü gibi 1 bitlik Lojik devre dört temel işlemin üretilmesini sağlar. n bitlik Lojik devre için 1 bitlik herbir Lojik devreden n tanenin paralel olarak görülmeli gerekir. Her bir bitlik birim bağımsız bir çıkış üretecektir. Bu durum, aritmetik devrede oluşturulan çıkışın aksine bit bazında elde edilmesi nedeniyle farklıdır. Diğer taraftan, 1-bitlik Lojik devrede üretilen işlemler AND, OR, XOR ve NOT, diğer tüm Lojik işlemlerin (fonksiyonlarının) Boole cebri vasıtasiyla elde edilmesini sağlar.

### 4.5.3 Bit Bazında Uygulanabilen Operasyonlar

Bit bazında uygulanabilen operasyonlar lojik işlemlere yeni bir bakış açısı getirir. Hızlı belleğin seçilen bitlerine istenilen değer verilebilir. Aşağıdaki operasyonları şu şekilde tanımlamak mümkündür:

#### Bellek Referansı

##### Seç ve set (selective set)

Hızlı belleğin seçilen bitini 1 yapmak. Diğer bitler değişmez. Bit seçimi için R kullanılsın. R1'deki bitlerin yalnızca R1'deki 1 bitlerine karşı düşen bitleri seçilir ve 1 yapılır.

##### Seç ve komplement

##### (Selective komplement)

1100 bellek R1

0110 bellek R

1110 sonuçta R1

##### Seç ve temizle (selective clear)

##### Maskele (mask)

##### Yerleştir

##### Temizle

<b>Bellek Referansı</b>	
<b>Seç ve set (selective set)</b>	Bu operasyonda seçilen bitlerin komplementi alınır.
<b>Seç ve komplement (Selective komplement)</b>	1100 bellek R1 0110 bellek R 1010 sonuçta R1
<b>Seç ve temizle (selective clear)</b>	
<b>Maskele (mask)</b>	
<b>Yerleştir</b>	
<b>Temizle</b>	
<b>Bellek Referansı</b>	
<b>Seç ve set (selective set)</b>	Bu operasyonda seçilen bitler 0 yapılır.
<b>Seç ve komplement (Selective komplement)</b>	1100 bellek R1 0110 bellek R 1000 sonuçta R1
<b>Seç ve temizle (selective clear)</b>	
<b>Maskele (mask)</b>	
<b>Yerleştir</b>	
<b>Temizle</b>	
<b>Bellek Referansı</b>	
<b>Seç ve set (selective set)</b>	Seçmeli operasyonların yanında maskele (mask), yerleştir (insert) ve temizle (clear) operasyonları da bit bazında lojik operasyonlardır. Maskele operasyonu seç ve temizle operasyonuna benzer, farkı R belleğinin 0 olan bitterine karşı düşen R1 bitlerinin 0 yapılmasıdır. Diğer R1 bitleri olduğu gibi kalır.
<b>Seç ve komplement (Selective komplement)</b>	
<b>Seç ve temizle (selective clear)</b>	
<b>Maskele (mask)</b>	1100 bellek R1 0110 bellek R 0100 sonuçta R1
<b>Yerleştir</b>	
<b>Temizle</b>	Maskeleme bir AND mikroişlemidir.

<b>Bellek Referansı</b>	
<b>Seç ve set (selective set)</b>	Yerleştir operasyonu bir grup bite yeni değer vermek için kullanılır. İlkin bu grup bitler maskelenip 0 yapılır (AND ile), sonra istenilen değerler OR ile aynı yere yerleştirilir.
<b>Seç ve komplement (Selective komplement)</b>	1010 1110 bellek R1 0000 1111 bellek R (maske)
<b>Seç ve temizle (selective clear)</b>	0000 1110 maskelendikten sonra R1
<b>Maskele (mask)</b>	İkinci kısımda
<b>Yerleştir</b>	0000 1110 bellek R1
<b>Temizle</b>	1110 0000 bellek R (yerleştirme) 1110 1110 yerleştir operasyonundan sonra R1

<b>Bellek Referansı</b>	
<b>Seç ve set (selective set)</b>	Son olarak, temizle operasyonunu gerçeklemanın bir yolu (diğer yolları da vardır) XOR mikroişlemiini kullanmaktadır.
<b>Seç ve komplement (Selective komplement)</b>	1100 bellek R1 1100 bellek R
<b>Seç ve temizle (selective clear)</b>	0000 sonuçta R1 ( $R1 \leq R1 \text{ xor } R$ )
<b>Maskele (mask)</b>	Ayrıca, XOR un 2 ikili sayının eşit olup olmadığını kontrol edebilecek bir mikroişlem olduğuna da dikkat edilmelidir.
<b>Yerleştir</b>	
<b>Temizle</b>	

## 4.6 KAYDIRMA MIKROISLEMLERI

### 1. Lojik kaydırma

$R \ll= shr R$ : lojik sağ kaydırma

$R \ll= shl R$ : lojik sol kaydırma

### 2. Dairesel kaydırma

$R \ll= cir R$ : dairesel sağ kaydırma

$R \ll= cil R$ : dairesel sol kaydırma

### 3. Aritmetik kaydırma

$R \ll= ash R$ : aritmetik sağ kaydırma

$R \ll= ashl R$ : aritmetik sol kaydırma

Kaydırma mikroişlemleri seri veri transferi için kullanılır. Aynı zamanda, diğer mikroişlemlerle birlikte, çarpma, bölme gibi çeşitli operasyon gerçeklemeleri için kullanılırlar. Hızlı bellek içeriği sağ ve sol olmak üzere iki yönde kaydırılabilir.

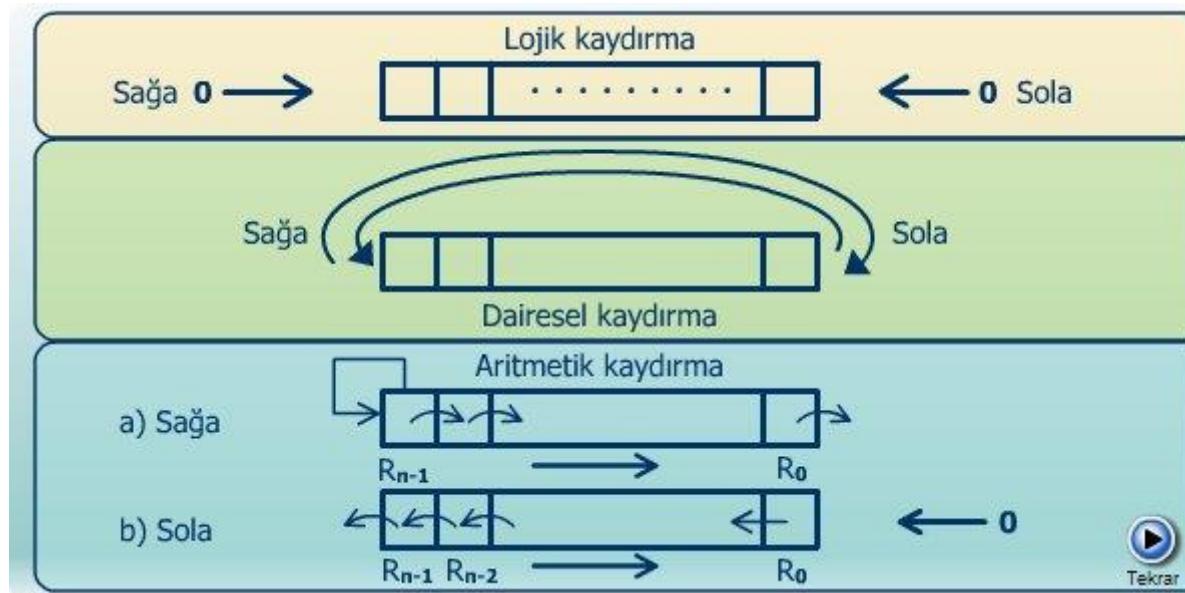
Üç tip kaydırma mikroişlemi yanda verildiği gibi tanımlanabilir.



Tekrar

## 4.6.1 Lojik Dairesel ve Aritmetik Kaydırma Mikroişlemleri

Lojik kaydırında, seri girişten 0 transfer edilerek, hızlı bellek içeriği sağa ve sola kaydırılır. Bu sırada, içeriğin karşı uçtan en son biti kaybolur. Dairesel kaydırında ise karşı uçtan gelen bit, seri giriş olarak kaydırılan uca uygulanır. Böylece, bellek içeriği dairesel olarak hareket etmiş olur.



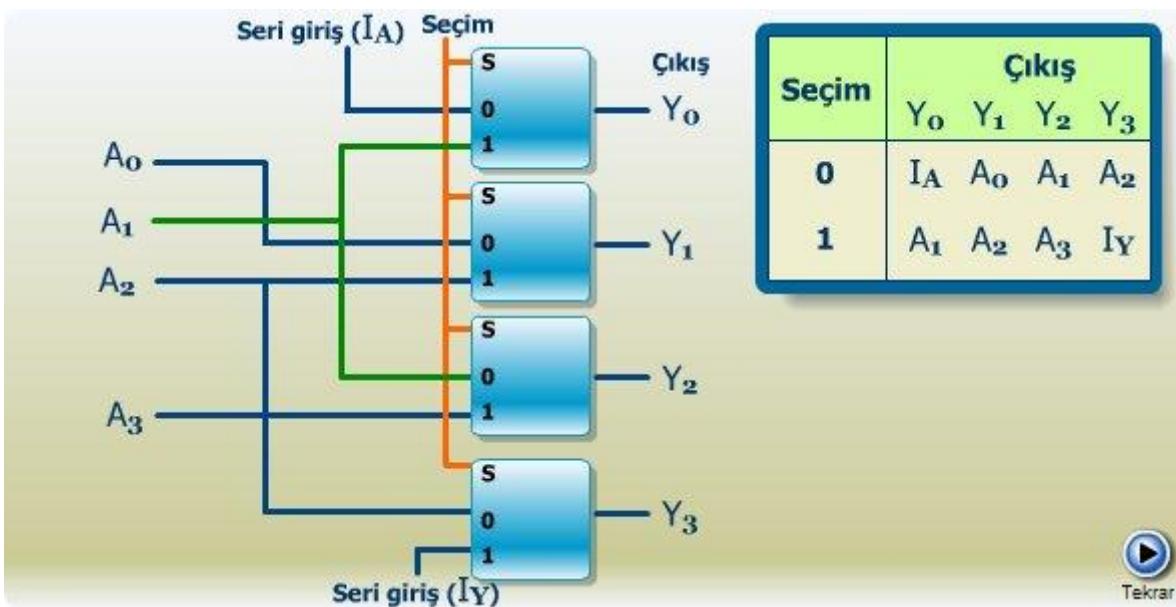
Aritmetik kaydırında, hızlı bellek içeriğine seri girişten 0 uygulanması ve işaretinin değişmemesi önemlidir. Aritmetik sola kaydırma işaretli ikili sayıyı 2 ile çarpmak, sağa kaydırma ise 2 ile bölmektir. Dolayısıyla, en soldaki işaret biti sağ ve sola kaydırında aynı kalmalıdır: sağa kaydırında  $R_{n-1}, R_{n-2}$ 'nin yerine geçer. Sola kaydırında ise  $R_{n-2}, R_{n-1}$  yerine geçeceğiinden, bir taşıma biti V ile bunu göstermek gereklidir:

$$V = R_{n-1} \text{ XOR } R_{n-2}$$

Eğer  $V=0$  ise taşıma yoktur,  $V=1$  ise taşıma olduğundan içeriğin işaretini değiştirmek gereklidir.

### 4.6.1.1 Kaydırma Yapan Hızlı Bellek Elemanının Tasarımı

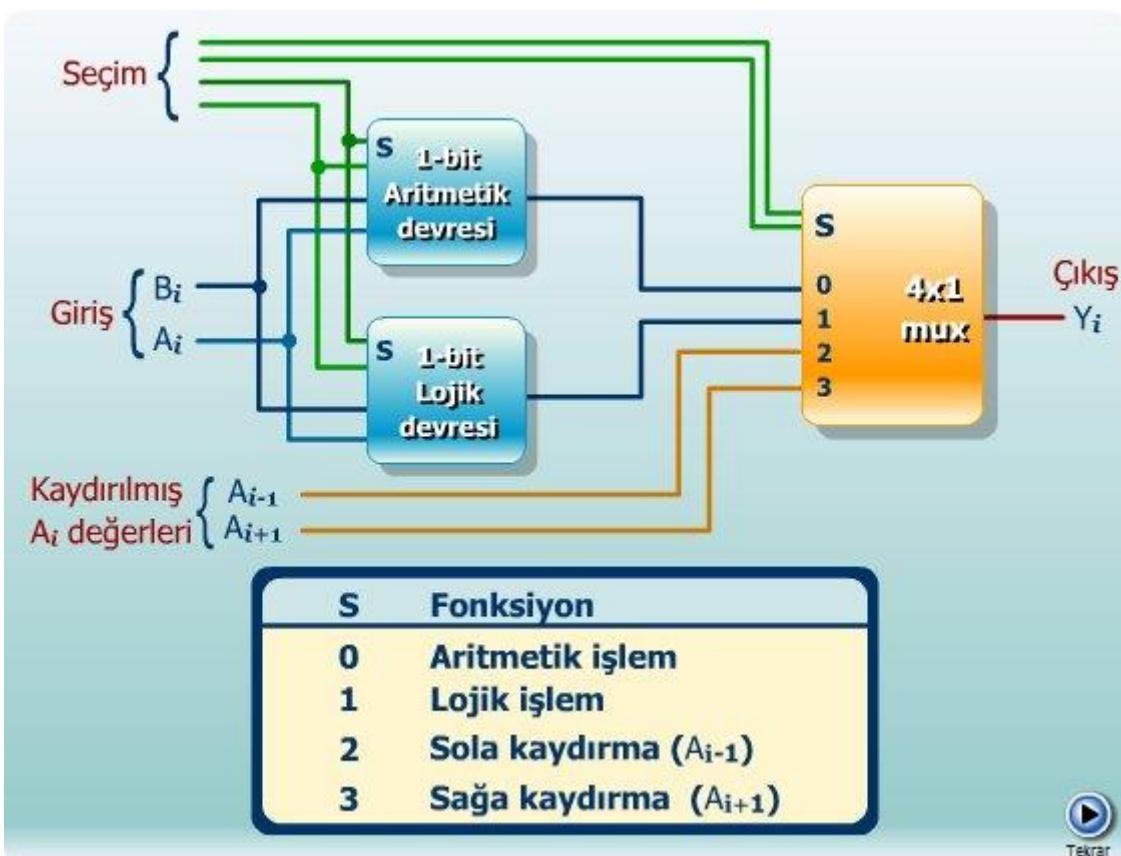
Kaydırma yapan hızlı bellek elemanın iki yoldan tasarımından bahsedebiliriz: Birincisinde, daha önce açıklanan iki-yönlü kaydırmalı bellek elemanını paralel yükleme özelliği ile kullanmak mümkündür. Burada 1 saat darbesi gecikme ile her bir kaydırma operasyonunu gerçekleyebiliriz. İkincisinde ise aşağıda gösterilen 4-bitlik kombinasyonel devreyi kullanabiliriz.



Bu devrede kullanılan dört 2x1 MUX'un uygun girişlerini seçerek, 1 saat darbesi gecikme olmaksızın, iki-yönlü kaydırma işlemi gerçekleşir. Bu tasarımın 4-bitten daha az bit sayısı için gerçeklenemeyeceği gözlenmelidir.

## 4.7 ARİTMETİK, LOJİK, KAYDIRMA (ALK veya ALU) Birimi

Bir bilgisayarda, tüm operasyonların gerçekleştirmesi için aritmetik, lojik ve kaydırma işlemlerini yapan devrelerin, veri depolayan hızlı bellek elemanları ile birleştirilerek ALU oluşturulması gereklidir. Aşağıda her bir işlemi yapan 1-bitlik blokların bir MUX ile birleştirilerek tüm işlemleri yapan bir devre elde edilebileceği gösterilmiştir. S<sub>1</sub> S<sub>0</sub> seçim hatları yoluyla aritmetik ve lojik işlem çeşitleri, S<sub>2</sub> S<sub>3</sub> hatları yoluyla çıkış MUX'dan kaydırma dahil herhangi bir işlem seçimi mümkün olmaktadır. 1-bitlik ALU devresine dikkat edilirse çıkış MUX'da tüm işlemlerin 1 bit bazında elde edilmesi mümkündür. Aritmetik ve Lojik devre blokları gerçekleştirip, çoğaltılarak istenilen n bit ALU devre tasarımı elde edilebilir.



## Bölüm Özeti

- Bu bölümde, hızlı bellekler arasındaki transfer işlemleri ve mikroişlemler anlatılmış, hızlı bellek transfer dili ve notasyonu tanımlanmıştır. Ayrıca, aritmetik, lojik, kaydırma mikroişlemleri ve aritmetik, lojik, kaydırma (arithmetic logic shift-ALU) birimi tasarımları anlatılmıştır.

## BÖLÜM - 5

# TEMEL BİLGİSAYAR YAPISI ve ÖZELLİKLERİ

Bu bölümde aşağıdaki konular incelenecaktır:

- Komut yapısı ve hızlı bellek elemanlarının tanıtımı
- Bilgisayar yapısı ve komutlarının tanıtımı
- Zamanlama ve kontrol
- Komutu oluşturan bileşenler
- Bellek ve hızlı-bellek referanslı komutlar
- Giriş-çıkış komutları ve bilgisayara kesintili veri aktarımı
- Tam bir bilgisayar ve akümülatör lojik tasarımları

## Bölüm Hedefi

Önümüzdeki üç haftada, temel bir bilgisayar yapısını belli önkabullerle tanıtıp, işlemlerinin hızlı bellek transfer işlemleri ile nasıl gösterildiğini inceleyeceğiz. Bilgisayarın tasarımını çok detaya girmeden anlatıp, bugün kullanılan bilgisayarlarla arasındaki bağıntıları kuracağiz. Bilgisayarın iç yapısı ile hızlı belleklerle gerçekleşen mikroişlemler arasında ilişki kurup, işletim detaylarını gözden geçireceğiz.

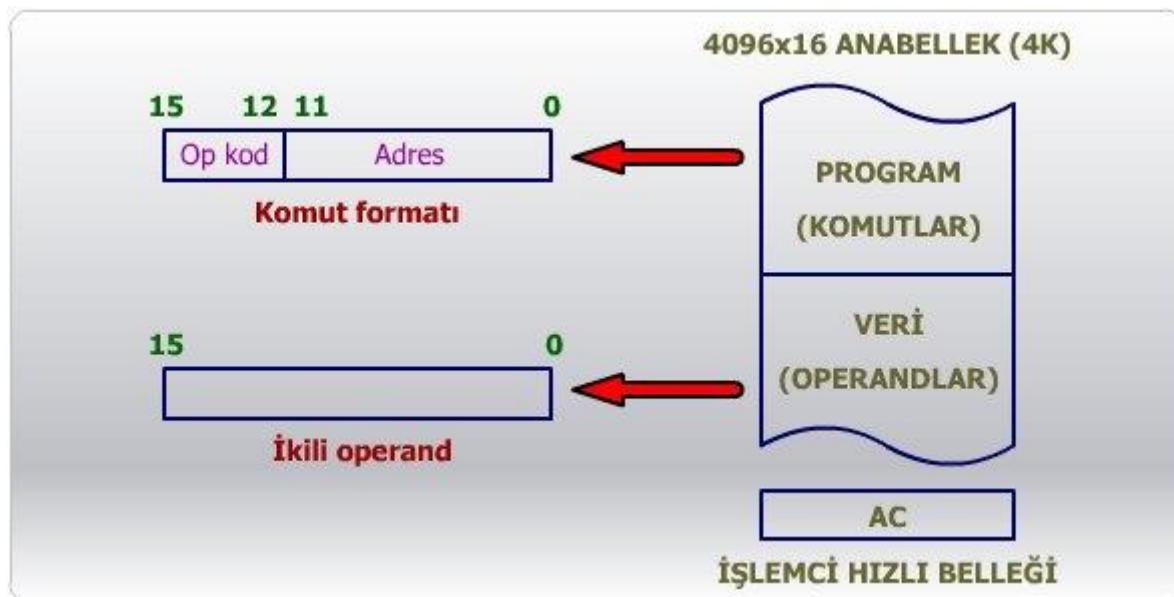
Aynı zamanda, tasarlanan bilgisayarın komutlarını, formatını, hızlı belleklerini, zamanlama ve kontrolünü detaylarla anlatacağız. Tipik bir komutun bileşenleri ve çeşitlerine [bellek-referanslı (memory reference), hızlı bellek-referanslı (register reference) ve giriş-çıkış (input output) komutlarına] değineceğiz. Giriş-çıkış kavramını ve bilgisayara kesintili (interrupt) veri aktarımını tanıtıp, bilgisayarın dış dünyadan nasıl bilgi aldığı anlatacağız. Bu kavramaların tanıtımında hızlı-bellek transfer dili yanında akış diyagramını (flowchart) kullanacağız. Son olarak, tam bir bilgisayar tanımı yapıp, daha önce tanıtılan sayısal bileşenler yardımıyla temel bilgisayar ve akümülatör (accumulator) lojik tasarımını aktaracağız.

## 5.1 TEMEL SAYISAL BİLGİSAYARIN KOMUT YAPISI ve HIZLI BELLEK ELEMANLARININ TANITIMI

Bir sayısal bilgisayarın, kullanıcı tarafından kullanımı "program" ve verilen "veri" ile mümkün olur. Program çeşitli görevleri yapan komutlardan oluşur. Veri-işleme görevi, yeni bir programın yazılımı ve aynı ve farklı veri üzerinde koşturulması olarak tanımlanabilir. Bu işlevi aşağıdaki "saklanmış program yapısı" örneği ile göstermek mümkündür: belli bir formata göre yazılmış ve saklanmış komutlar, saklanmış veri üzerinde koşturulabilir (program execution). Komut kodu, belli bir operasyonu gerçeklemek için tasarlanmış bir grup bittir.

Komut kodu iki parçadan oluşmuştur: operasyon kısmı ve address kısmı. Operasyon kısmı yapılan işlemin tanımlandığı bir grubudur. Adres kısmı ise, tanımlanan moda göre,

operand adresinin tanımlandığı bitlerdir. Operand hesaplanan "efektif adres" değerine göre bellekten okunur. En kullanışlı bilgisayar yapılarından biri, işlemciye ait bir hızlı-bellek elemanı (akümülatör-AC) kullanmaktadır. İşlem görecek operandı ve işlem sonucunu burada saklamaktır.

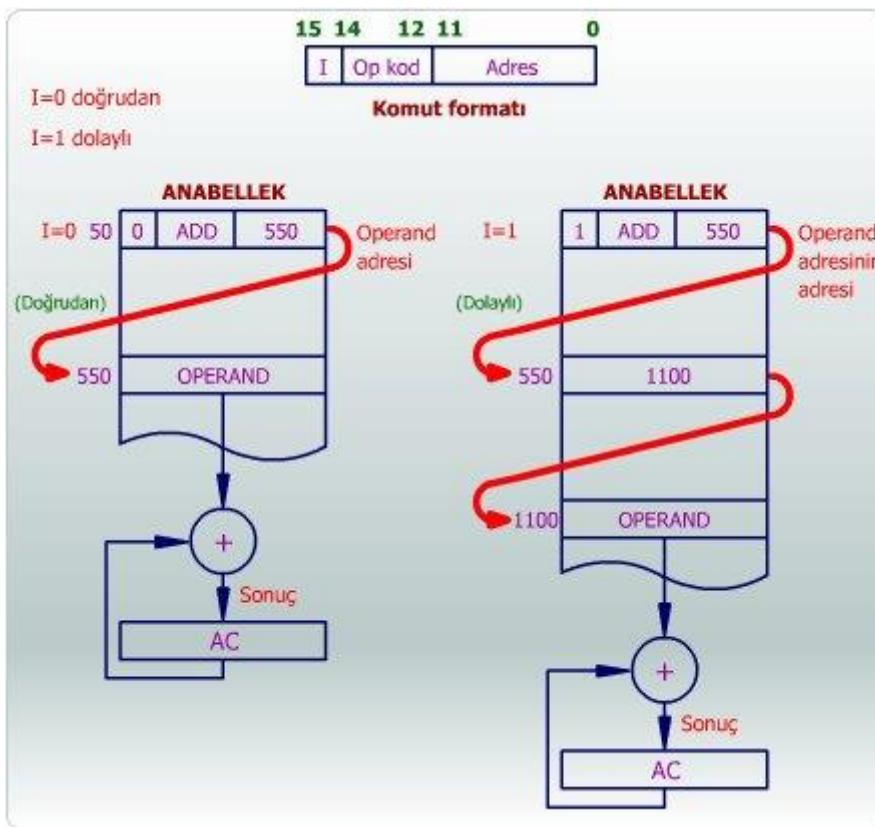


Saklanmış program yapısı

### 5.1.1 Temel Sayısal Bilgisayarın Komut Yapısı ve Hızlı Bellek Elemanlarının Tanıtımı (Devam)

Her bir bilgisayar komutu, bir dizi mikroişlem ile gerçekleşir: bilgisayar komutu bellekten okur ve uygun hızlı-belleğe yerleştirir. Kontrol kısmı komutun ikili bilgisini yorumlar ve hangi mikroişlemlerin yapılacağı kararlaştırılır. Her bilgisayarın kendine özgü bir komut seti vardır. Genel amaçlı bir bilgisayarın en önemli özelliği, yazılan programı veya herhangi bir komut setini koşturabilmesidir.

16 bitlik komut formatında 4 bit operasyon kodu ve kalan 12 bit adres için ayrılmıştır. 12 bitlik adres ile  $2^{12}$  kelimelik bir bellek biriminin tüm adres değerlerine erişilebilir. Adres modları olarak, operandi içeren (immediate), doğrudan (direct) ve dolaylı (indirect) modlar kullanılmıştır. Efektif adres, adres modu tarafından hesaplanan adres olarak tanımlanır. Doğrudan modda, komutun adres kısmı operandın yerini gösterir. Dolaylı adres modunda, komutun adres kısmı "adresin saklandığı yerin" veya adresin adresinin değeridir.



Komut formatı, doğrudan ve dolaylı adres modu

### 5.1.1.1 Adres Modları

Önceki sayfada anlatılan adres modlarını aşağıdaki şekilde özetlemek mümkündür.

Operandı içeren mod: Operand komutta bulunur.

Doğrudan mod : Operandın adresi komutta bulunur.

Dolaylı mod : Operandın adresinin adresi komutta yer alır.

Tekrar

Genel olarak, operandın adresi "efektif adres" olarak tanımlanır. Komuttan hesaplanan adres efektif adresidir.

## 5.2 TEMEL BİLGİSAYARIN YAPISI ve KOMUTLARININ TANITIMI

Bilgisayar komutları, anabellekte ardışır gelen adreslerde saklanır ve sıra ile koşturulur. Bir komut koşturulurken, arkasından gelen komutun adresi hesaplanır. Kontrol birimi, komutu oluşturan mikroişlemleri ardışır gelen saat darbelerinde gerçekler. Mikroişlemlerin koşturulmasında, anabellek ve hızlı bellekler kullanılır. 16 bit olarak tanımlanan komut formatının 4 bitlik kısmının operasyon kodu olduğu hatırlanırsa, geri kalan 12 bitlik adres bilgisinin 4096 kelimelek bir anabellek ile uyumla çalışacağı açıklar.

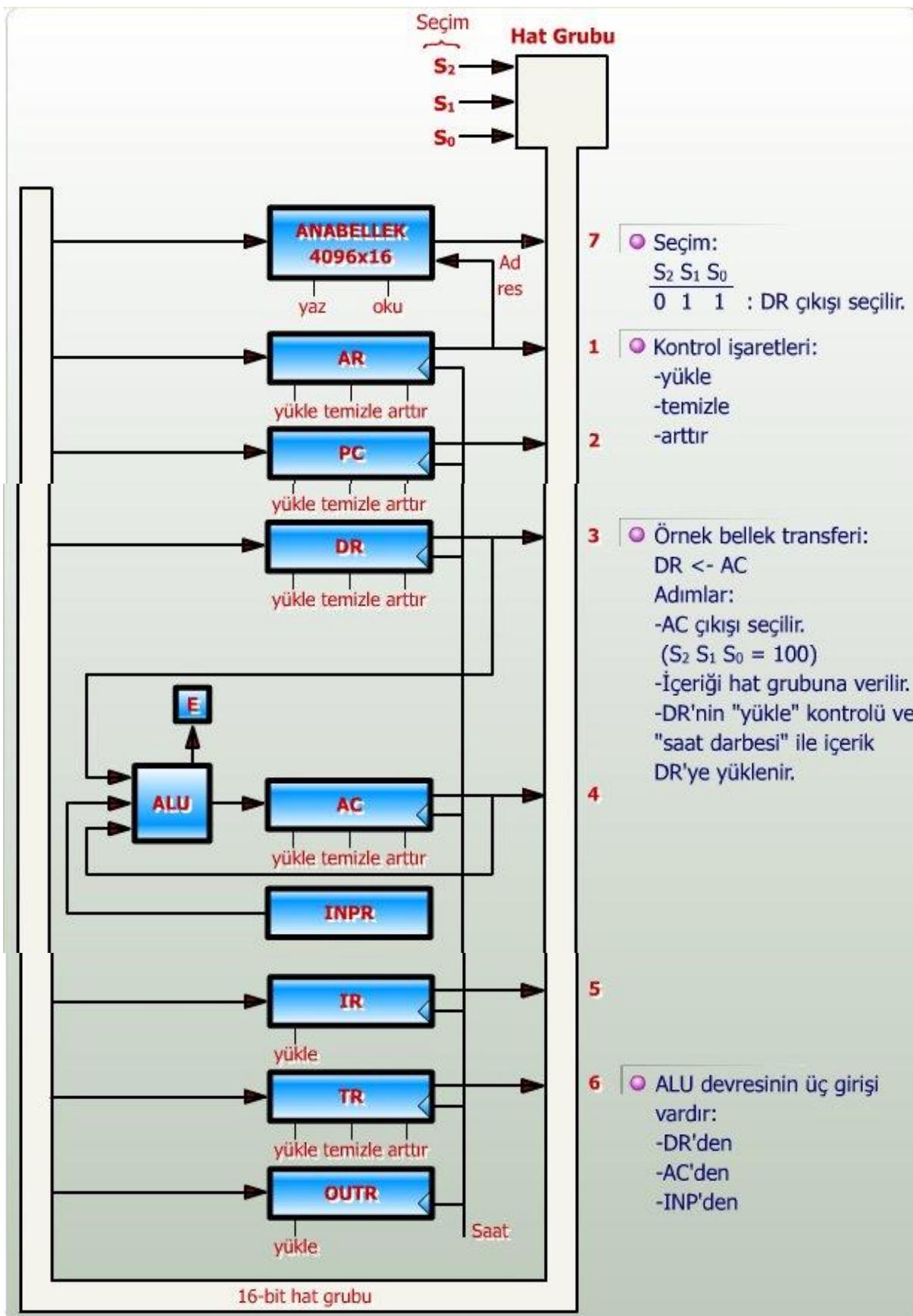
Temel bilgisayar yapısında aşağıdaki tablodaki hızlı bellekler kullanılmaktadır:

Hızlı bellek (h.b.)				
Sembolü	Bit sayısı	Adı	Fonksiyonu	
DR	16	Veri h. b.	Operandı saklar	
AR	12	Adres h. b.	Adresi saklar	
AC	16	Akümülatör	İşlemci h. b.	
IR	16	Komut h.b.	Komutu saklar	
PC	12	Program sayıcı	Komut adresini saklar	
TR	16	Geçici h.b.	Geçici veriyi saklar	
INPR	8	Giriş h.b.	Giriş karakterini saklar	
OUTR	8	Cıkış h.b.	Cıkış karakterini saklar	

DR ve AR, daha önce açıklandığı gibi, anabellek okuma ve yazma işlemlerinde veri ve adres bilgisini tutmakla görevlidir. PC, bilgisayarda saklı bulunan programın komutlarının okunmasıyla görevli olup, koşulan komuttan sonraki komutun adresini gösterir. INPR ve OUTR 8 bitlik karakterin, kesinti işlevi yoluyla, bilgisayar akümülatörüne okunması ve yazılmasıyla görevlidir. TR ise geçici verinin saklanabileceği hızlı bellektir. Son olarak da, IR komutun saklanması, operasyonunun yorumlanması, mikroişlemlerin başlatılması görevini yapar.

### 5.2.1 Temel Bilgisayarın Yapısı

Aşağıda temel bilgisayarın yapısı gösterilmiştir. Temel bilgisayar sekiz hızlı bellek, bir ana bellek ve kontrol biriminden oluşmaktadır. Bu yapıda ikili bilgiler, 16 bit hat grubu yoluyla elemanlar arasında transfer edilmektedir. Ortak hat grubu, genel bir ikili bilgi aktarım metodudur. İkili bilgi aktarımı şu şekilde yapılır: Bellek elemanlarının çıkışları  $S_2 S_1 S_0$  seçim girişleri ile kontrol edilen MUX (3X8) tarafından hat grubuna bağlanır. Örneğin, DR'nin çıkışı 3 ise,  $S_2 S_1 S_0 = 011$  DR'nin çıkışının seçilmesini sağlar. Böylece, DR'nin içeriği hat grubuna verilir. Bu içerik hangi hızlı belleğe yüklenecekse, onun "yükle" kontrolü aktive edilir. Örneğin, AC'nin yükle hattı aktive edilirse AC<=DR transferi sağlanır, 16 bitlik bilgi DR'den AC'ye yazılır. Hızlı belleklerde genel olarak üç kontrol işaret이 vardır: Yükle, arttır ve temizle. Böylece, belleklerin bilgi içeriği kontrol edilir.



Yapıda bulunan dört hızlı bellek DR, AC, IR ve TR 16 bittir. İki hızlı bellek AR ve PC 12 bittir ve geri kalan iki bellek İNPR, OUTR ise 8 bittir. Örneğin, AR ve PC anabellekteki adreslerin tanımlanmasını sağlar. Eğer bu belleklerin içeriğine 16 bitlik hat grubu gerekirse, en anlamlı 4 bit (the most significant 4 bit) 0000 olarak alınır. İNPR ve OUTR ise 8 bit içerik olduğundan, hat grubunun en az anlamlı 8 bitini kullanırlar. Bu hızlı bellekler, giriş düğmeninden veya çıkış düzenine 8 bitlik karakter (ASCII) transferi için kullanılırlar. AC yoluyla bu bilgi transferi gerçekleştirilebilir.

Ana bellek okuma ve yazma operasyonlarının, AR ve DR hızlı bellek elemanları yoluyla yapıldığı hatırlanmalıdır: okuma operasyonunda AR anabellek adresini tutar, anabelleğin oku kontrolü aktive edilir. Son olarak da, bu içerik DR'ye aktarılacağından, onun "yükle" kontrolü aktive edilerek transfer gerçekleştirilebilir. Yazma işleminde ters işlem yapıldığı hatırlanmalıdır.

16 bitlik hat grubu bilgisayar yapısında veri aktarımını sağlamaktadır: Ana bellek ile hızlı bellekler arasında, hızlı belleklerin kendi aralarında aktarımı sağlar. Buna karşın ALU ile AC belleği arasındaki doğrudan bağlantıya dikkat edilmelidir. ALU'da yapılan tüm işlemlerde bir operand AC saklanır, işlem sonucu ise yine AC'ye aktarılır. Dolayısıyla AC'de bulunan operand silinir. Bu yapıya "tek akümülatörlü" yapı adı da verilir.

Gösterilen yapının diğer bir özelliği de aynı saat darbesinde iki mikroişlemenin yapılabilceğinin gözlenmesidir:

DR <= AC , AC <= DR

Yukarıdaki şekilde DR ve AC arasındaki bağlantı bu transferleri sağlar.

ALU'da E bayrağı (flag) toplama işleminin eldesi, çıkarmanın ödünsü, taşıma etc. gibi değerleri göstermekte kullanılır.

### 5.3 TEMEL BİLGİSAYAR KOMUTLARI

Temel bilgisayarın üç komut kodu formatı aşağıda gösterilmiştir.

15 14 12 11 0

I	Op kod	Adres
---	--------	-------

Op kod = 000'dan 011'e

Bellek referanslı komut

15 12 11 0

0111	Hızlı bellek operasyonu
------	-------------------------

Op kod = 111, I=0

Hızlı bellek referanslı komut

15 12 11 0

1111	Giriş/Çıkış operasyonu
------	------------------------

Giriş/Çıkış komutu

Operasyon kodu alanı olan en anlamlı 4 bitin bellek referanslı doğrudan ve dolaylı komutların seçimi I biti ve 000'dan 110'a kadar olan kodlardan elde ettiğine dikkat edilmelidir. Hızlı bellek referanslı komutlara 0111 ve giriş/çıkış komutlarına 1111 kodu atanmış

### 5.3.1 Temel Bilgisayar Komutları (Devam)

Temel bilgisayarın komutları aşağıdaki tabloda gösterilmiştir.

KOMUT ADI	AÇIKLAMA
AND	Bellek kelimesi ile AC lojik "and"
ADD	Bellek kelimesi ile AC'yi "topla"
LDA	Bellek kelimesini AC'ye yükle
STA	AC'ye bellek kelimesini yükle
BUN	Koşulsuz dallanma
BSA	Dallan ve geri dönüş adresini sakla
ISZ	Arttır ve sıfır ise sonraki komuta atla
CLA	AC'yi temizle
CLE	E bayrağını temizle
CMA	komplement AC
CME	komplement E
CIR	AC'yi sağa dairesel kaydır

<b>CIL</b>	AC'yi sola dairesel kaydır
<b>INC</b>	AC'yi arttır
<b>SPA</b>	Eğer AC>0 ise sonraki komuta atla
<b>SNA</b>	Eğer AC<0 ise sonraki komuta atla
<b>SZE</b>	Eğer E=0 ise sonraki komuta atla
<b>HLT</b>	Halt
<b>INP</b>	AC'ye karakter gir (INPR'dan)
<b>OUT</b>	AC'den karakter gönder (OUTR'a)
<b>SKI</b>	Giriş_bayrağı=1 ise sonraki komuta atla
<b>SKO</b>	Çıkış_bayrağı=1 ise sonraki komuta atla
<b>ION</b>	Kesinti (interrupt) geçerli (on)
<b>IOF</b>	Kesinti geçersiz (off)

### 5.3.2 Tam Komut Seti ve Verimliliği

Temel bilgisayarın komutlarını tanıttıktan sonra, bu setin tamlığı ve verimliliğini tartışacağız. Genel olarak bir komut seti aşağıda tanımlanan her bir sınıfından yeteri komut bulunduruyorsa, bu set “tam bir komut seti” dir denilir. Bu sınıflar:

- **Aritmetik, lojik ve kaydırma komutları**
- **İkili bilgiyi bellek ile işlemci arasında taşıyan komutlar**
- **Program kontrol komutları**
- **Giriş/çıkış komutları**

Tekrar

Temel bilgisayar komutları tablosunda tanımlanan komutların her bir sınıfından “minimum” komut içerdigini göstermek mümkündür. Böylece tanımlanan set, bir minimum settir. Örneğin, sadece ADD operasyonu vardır, çıkarma için 2-komplement elde edilip, toplama yapılması gereklidir. Çarpma, bölme vs. komutları için aynı yol izlenmelidir.

Diğer taraftan, bu komut seti verimli bir set değildir. Sık kullanılan komutlar hızlı bir şekilde gerçeklenmemiştir. Çıkarma, çarpma, OR, XOR işlemleri çok yavaş gerçeklenmiştir. Günümüz bilgisayarları hızlı devrelerle bu işlemleri yapabilmektedir, örneğin çarpma devresi.

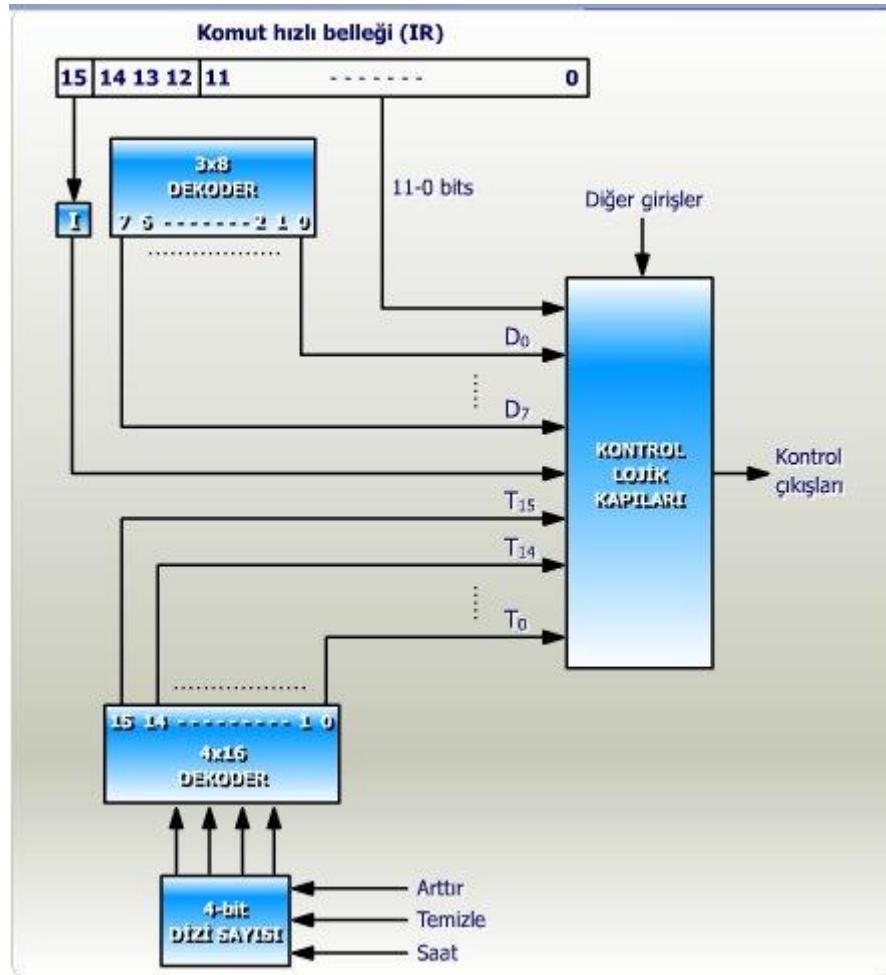
### 5.4 ZAMANLAMA ve KONTROL

Bilgisayarın zamanlaması master saat üretici tarafından oluşturulan saat darbeleriyle sağlanır. Bu devre tipik bir ikili sayıcı ve dekoder devresi olarak tasarlanabilir. Saat

darbeleri ancak kontrol işaretini ile birlikte olunca hızlı belleğin durumunu değiştirebilir. Örneğin, yükle işaretini ile birlikte uygulanan saat darbesi belleğin yüklenmesini sağlar.

İki çeşit kontrol yapısı kullanılmaktadır: donanım yoluyla (hardwired) ve mikroprogramlama veya yazılım yoluyla (microprogrammed) kontrol. Donanım yoluyla kontrolde, lojik kapılar, FF'ler, dekoder ve diğer sayısal devreler kullanılır. Bu kontrol hızlidir, bir kez tasarılanınca bir daha değiştirilemez. Mikroprogramlama yoluyla kontrol esnekdir fakat daha yavaştır. Bu bölümde, donanım yoluyla kontrolden bahsedeceğiz. Mikroprogramlama kontrolüne ise daha sonra detaylı olarak değineceğiz.

Aşağıda donanım yoluyla kontrol blok diyagramı gösterilmiştir.

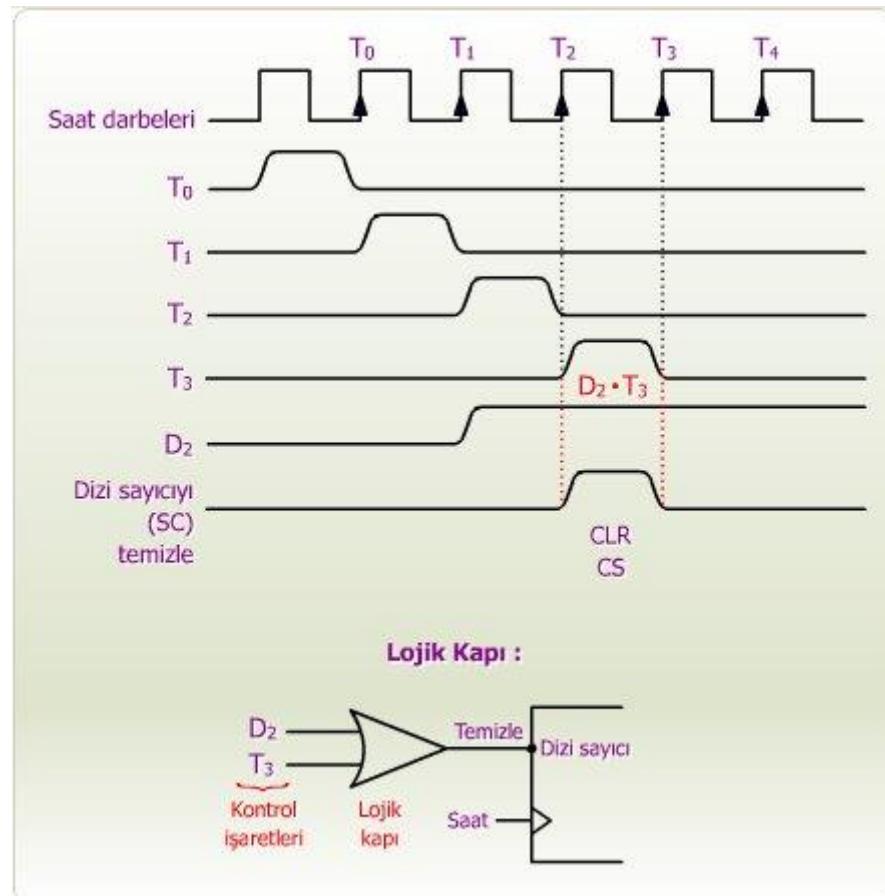


### 5.4.1 Zamanlama Birimi

Kontrol biriminin iki kısmı gözlenmelidir: zamanlama kısmında 4-bit dizi sayacı (sequence counter, SC) 0000'dan 1111'e kadar sıra ile sayarak, önündeki 4x16'lık dekoder devresinin 0'dan 15'e kadar çıkışlarından birini seçmektedir. Seçilen çıkış T<sub>0</sub>'dan T<sub>15</sub>'e kadar bir darbe üretir. Diğer kısmı ise IR'ye bağlantılı olarak, burada bulunan komutun operasyon kodu ve adres bilgisini kontrol lojik kapılarına aktarır. Operasyon kodu 3x8 dekoder yoluyla D<sub>0</sub>' dan D<sub>7</sub>' ye kadar dekode edilmiş çıkış işaretleri üretir. Bu arada I biti de adres bilgisiyle birlikte kontrol devresine giriş oluşturur. Kontrol devresi çıkışı, sonuçta komut ile ilişkili bir kontrol işaretini dizisi oluşturur. Bunu bir örnek ile ve zamanlama diyagramı ile açıklayalım:

### **D<sub>2</sub> T<sub>3</sub> : SC <= 0**

Aşağıda yukarıdaki ifadenin zamanlama diyagramı gösterilmiştir. İfadenin koşulu (D<sub>2</sub> T<sub>3</sub>) sağlandığında, dizi sayıcısının (SC) saat darbesinin pozitif geçişinde “temizle” girişi aktif yapılarak ifade sağlanır.



### **Örnek zamanlama diyagramı**

Saat darbe üreteci, dekoder çıkışından T<sub>0</sub>, T<sub>1</sub>, T<sub>2</sub>, T<sub>3</sub> darbelerini üretir. D<sub>2</sub> komut dekoderinin çıkışı ve T<sub>3</sub> işaretinin ikisi de aktif olduğunda üretilen çıkış, SC'nin “temizle” girişine uygulanarak saat darbe üretecinin T<sub>0</sub>'dan başlaması sağlanır.

Bilgisayarın çalışmasını tam olarak anlayabilmek için, hızlı bellek transfer dilindeki çeşitli ifadelerin zamanlama diyagramlarını incelemek gereklidir.

## **5.5 KOMUT FAZLARI**

Ana bellekte saklanmış olan programın her bir komutu, aşağıdaki fazlarla koşturulur:



Tekrar

- Ana bellekten bir komutun alınması (fetch cycle)
- Komutun dekode edilmesi (decode cycle)
- Efektif adresin hesaplanması ve komutun dolaylı adresinin olup olmadığını araştırılması
- Komutun koşturulması (execution cycle)

Komutun alınması ve dekode edilmesi (fetch and decode cycle), aşağıdaki bellek transfer dili ifadeleri ile tanımlanır:

$T_0 : AR \leq PC$

$T_1 : IR \leq M[AR], PC \leq PC + 1$

$T_2 : D_0, \dots, D_7 \leq \text{Decode } IR(12-14), AR \leq IR(0-11), I \leq IR(15)$

İlk satırda,  $T_0$  zamanlama işaretinin ilişkili olduğu saat geçişinde PC'de saklı olan komut adresi AR'ye transfer edilir.  $T_1$ 'e ilişkin saat darbesinin geçişinde iki transfer gerçekleşir: ilkinde, AR ile gösterilen adresteki içerik (komut) IR'ye aktarılır. Aynı zamanda PC, 1 arttırılır.  $T_2$  zamanında IR' de bulunan operasyon kodu dekode edilir, dolaylı biti I FF'sine aktarılır ve komutun adres kısmı AR' ye transfer edilir. Bu satırlarla yapılan işlem, komutun ana bellekten okunup ne iş yaptığı, doğrudan veya dolaylı olup olmadığı ve adres kısmı öğrenilmiş olur.

### 5.5.1 Örnek Hızlı Bellek Transfer İfadesinin Gerçeklenmesi

PC, AR ve IR hızlı bellekleri kullanıldığında, bu belleklerin "yükle" ve "arttır" kontrol girişlerinin kullanıldığına dikkat edilmelidir. Temel bilgisayar diyagramına ek olarak kullanılan lojik kapılarla bu transferlerin nasıl yapıldığı gözlenmelidir. Örneğin,

$T_1 : IR \leq M[AR], PC \leq PC + 1$

ifadesi aşağıdaki adımlarla gerçekleşir:

Ana belleğin okuma girişi aktive edilir.

Ana bellek içeriği hat grubuna verilip,  $S_2 S_1 S_0 = 111$  (ana bellek çıkış) seçilir.

Hat grubundaki içerik "yükle" kontrolü aktive edilerek IR yazılır.

Aynı zamanda, PC "arttır" kontrolü ile artırılır.



Tekrar

### 5.5.2 Komut Çeşitlerine Göre Dallanma

$T_3$  zamanında ise ana bellekten okunan komutun tipinin belirlenmesi fazı gelir. Üç komut tipine göre dört kola ayrimının gerektiği anlaşılır:

$D'_7 IT_3 : AR \leq M[AR]$ : Bellek referanslı, dolaylı adresli komut a

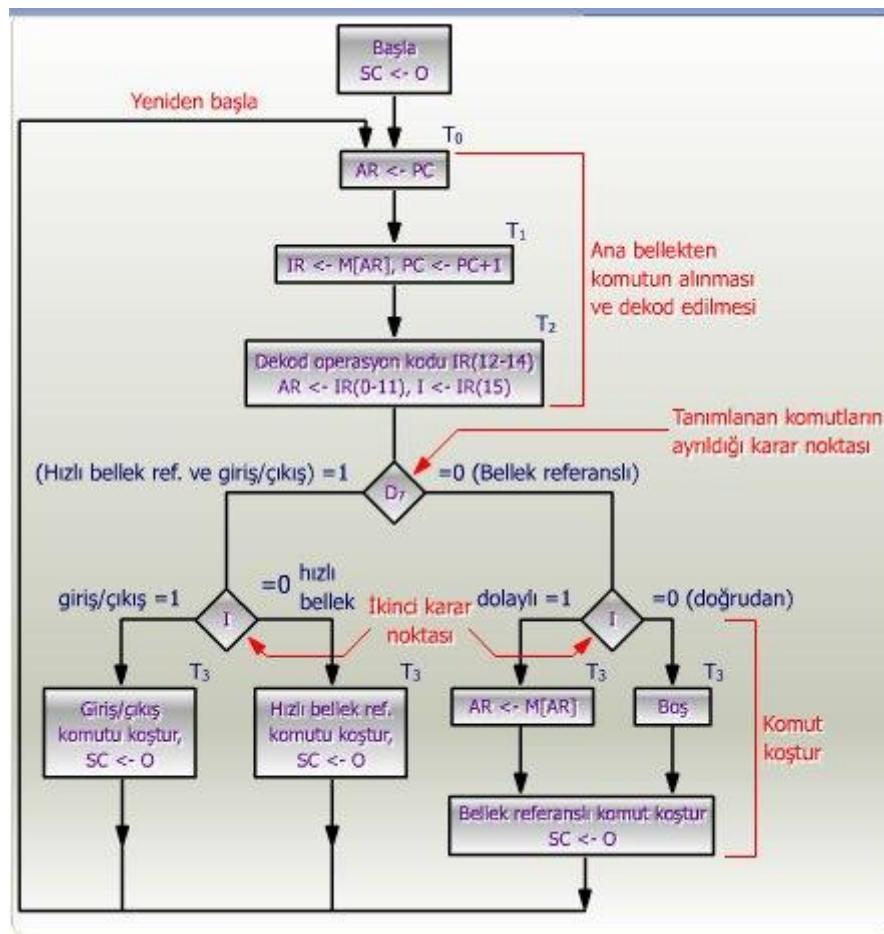
D<sub>7</sub>I'T<sub>3</sub>: İşlem yok: Bellek referanslı, doğrudan adresli komut

D<sub>7</sub>I'T<sub>3</sub> : Hızlı bellek referanslı komut

D<sub>7</sub>IT<sub>3</sub> : Giriş/çıkış komutu

Her bir ayrı koldan, komut çeşitlerine göre tanımlanan mikroişlemler yardımıyla komutlar koşturulur. Burada kapalı olarak SC<=SC+1 ifadesi her bir mikroişlem için geçerlidir. Her bir komutun koşturumundan sonra SC<=0 alınarak komut fazı başa döner.

Komutun yukarıda anlatılan fazları ile koşturulması, "hızlı bellek transfer dili" yanında "akış diyagramı" ile de gösterilebilir. Aşağıda komut fazlarının koşturulumu akış diyagramı ile gösterilmiştir.



#### Akış diyagramı ile komut fazlarının gösterimi

Kısaca akış diyagramı tanımını hatırlamak gerekirse, dikdörtgenler (kutular) belli bir işlemi, oklar sonraki işleme geçişini ve eşkenar dörtgen şekli ise dallanma veya karar verme noktalarını gösterir. Dikdörtgenlerin sağ üst köşesindeki T<sub>0</sub>, T<sub>1</sub>, T<sub>2</sub> vb. yapılan mikroişlemlerin zamanlamasını gösterir. Basılılığı ve öğrenme kolaylığı bu gösterimin sıkılıkla kullanılmasını sağlar.

## 5.6 HIZLI BELLEK REFERANSLI KOMUTLAR

Temel bilgisayarda, hızlı bellek referanslı komutlar D<sub>7</sub>=1 ve I=0 kontrol değerleri ile tasarlanmıştır. Komut kodunun 0' dan 11' e kadar olan değerleri ile tanımlanan 12

komuttan oluşurlar. İlk yedi tanesi temizle, AC ve E bayrağı üzerinde tanımlanmış komplement, dairesel kaydırma ve artırma işlemlerinden oluşurlar. Kalan dört tanesi komuttan sonrakine atlama ve sonucusu ise işlemi durdurma olarak tanımlanmıştır. Bilindiği gibi

$D_7 I'T_3 : d \text{ ve } IR(i) = B_i$  (IR(011) bit değerleri)

ile tanımlanırlar. Aşağıdaki tablo hızlı bellek referanslı komutları tanımlar.

<b>HIZLI BELLEK REFERANSLI KOMUTLAR TABLOSU</b>		
<b>SC&lt;=0 (d ile dizi sayıcı baştan saymaya başlar)</b>		Tekrar
<b>CLA</b>	<b>AC&lt;=0</b>	(d B 11 )
<b>CLE</b>	<b>E&lt;=0</b>	(d B 10 )
<b>CMA</b>	<b>AC&lt;=AC'</b>	(d B 9 )
<b>CME</b>	<b>E&lt;=E'</b>	(d B 8 )
<b>CIR</b>	<b>AC&lt;=shr AC, AC(15)&lt;=E, E&lt;=AC(0)</b>	(d B 7 )
<b>CIL</b>	<b>AC&lt;=shl AC, AC(0)&lt;=E, E&lt;=AC(15)</b>	(d B 6 )
<b>INC</b>	<b>AC&lt;=AC+1</b>	(d B 5 )
<b>SPA</b>	<b>if (AC(15) = 0) then (PC&lt;=PC+1)</b>	(d B 4 )
<b>SNA</b>	<b>if (AC(15) = 1) then (PC&lt;=PC+1)</b>	(d B 3 )
<b>SZA</b>	<b>if (AC = 0) then (PC&lt;=PC+1)</b>	(d B 2 )
<b>SZE</b>	<b>if (E = 0) then (PC&lt;=PC+1)</b>	(d B 1 )
<b>HLT</b>	<b>İşlemi durdur (S &lt;=0. S başlat-durdur)</b>	(d B 0 )

## 5.7 ANA BELLEK REFERANSLI KOMUTLAR

Yedi ana bellek referanslı komut,  $D_i i=0, 1, \dots, 6$  operasyon koduna göre AND, ADD, LDA, STA, BUN, BSA ve ISZ olarak sıralanır.

<b>AND</b>	AC'yi AND'le
<b>ADD</b>	AC ile topla
<b>LDA</b>	AC'ye yükle
<b>STA</b>	AC'yi sakla
<b>BUN</b>	Koşulsuz dallan
<b>BSA</b>	Dallan ve geri dönüş adresini sakla
<b>ISZ</b>	Arttır, sıfır ise sakla



Tekrar

### 5.7.1 Ana Bellek Referanslı Komut Tanımları

AND to AC lojik operasyonu aşağıdaki mikroişlemlerle gösterilir:

D <sub>0</sub> T <sub>4</sub> : DR<=M[AR]
D <sub>0</sub> T <sub>5</sub> : AC<=AC AND DR, SC<=0

D<sub>0</sub> T<sub>5</sub> fazında AND işlemi tamamlanıp, aynı saat darbesinde SC<=0 mikroişlemi ile yeni komutun mikroişlem dizisine başlanır.

ADD to AC lojik operasyonu ise

D <sub>0</sub> T <sub>4</sub> : DR<=M[AR]
D <sub>0</sub> T <sub>5</sub> : AC<=AC + DR, E<=C <sub>out</sub> , SC<=0

olarak tanımlanır. C<sub>out</sub> ise toplama mikroişleminin eldesidir.

LDA veya AC'ye yükle işlemi ise

D <sub>2</sub> T <sub>4</sub> : DR<=M[AR]
D <sub>2</sub> T <sub>5</sub> : AC<= DR, SC<=0

mikroişlemleri ile tanımlanır. STA veya AC içeriğini sakla ise

D <sub>3</sub> T <sub>4</sub> : M[AR]<=AC, SC<=0 dır.
-------------------------------------------------------

BUN veya koşulsuz dallanma

D <sub>4</sub> T <sub>4</sub> : PC<= AR, SC<=0
------------------------------------------------

olup, efektif adres PC'ye aktarılır ve program akışı bu noktadan devam eder.

BSA veya dallan ve geri dönüş adresini sakla komutunun mikroişlemleri ise

D<sub>5</sub> T<sub>4</sub> : M[AR] <= DR, AR <= AR+1

D<sub>5</sub> T<sub>5</sub> : PC <= AR, SC <= 0

şeklinde gösterilir.

ISZ arttı, sıfır ise atla komutu sonraki komutun arttırılmasını sağlar.

D<sub>6</sub> T<sub>4</sub> : DR <= M[AR]

D<sub>6</sub> T<sub>5</sub> : DR <= DR+1

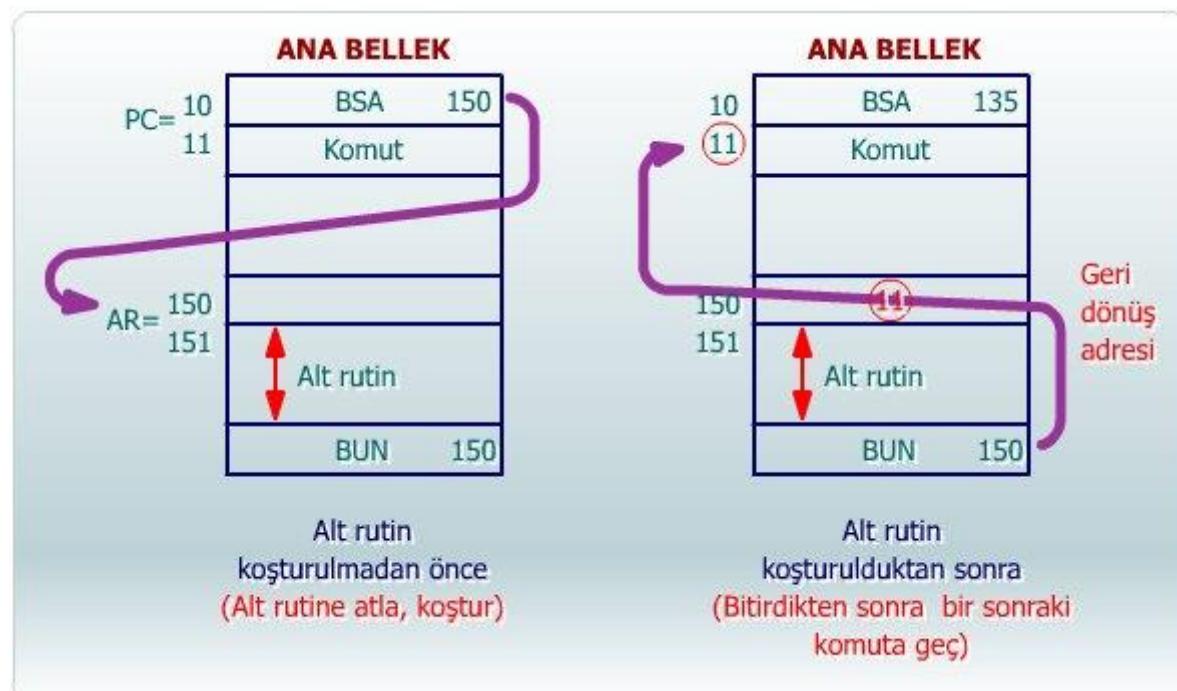
D<sub>6</sub> T<sub>6</sub> : M[AR] <= DR, if (DR=0) then (PC <= PC+1), SC <= 0

şeklinde tanımlanır.

Komut fazlarında verilen akış diyagramında T<sub>3</sub> fazından sonra gelen komut koşturumu kısmına her bir komut için tanımlanan ve T<sub>4</sub>'ten itibaren başlanan koşturulma fazı eklenecek tam bir komut fazı için akış diyagramı elde edilir.

### 5.7.2 BSA Koşturulmasına Ait Örnek

Aşağıdaki şekilde BSA koşturulması örnekle açıklanmıştır. BSA subroutine call olarak da bilinir. Genellikle bir fonksiyon için yazılan subroutine, fonksiyon tamamlandıktan sonra ana programa geri döner.

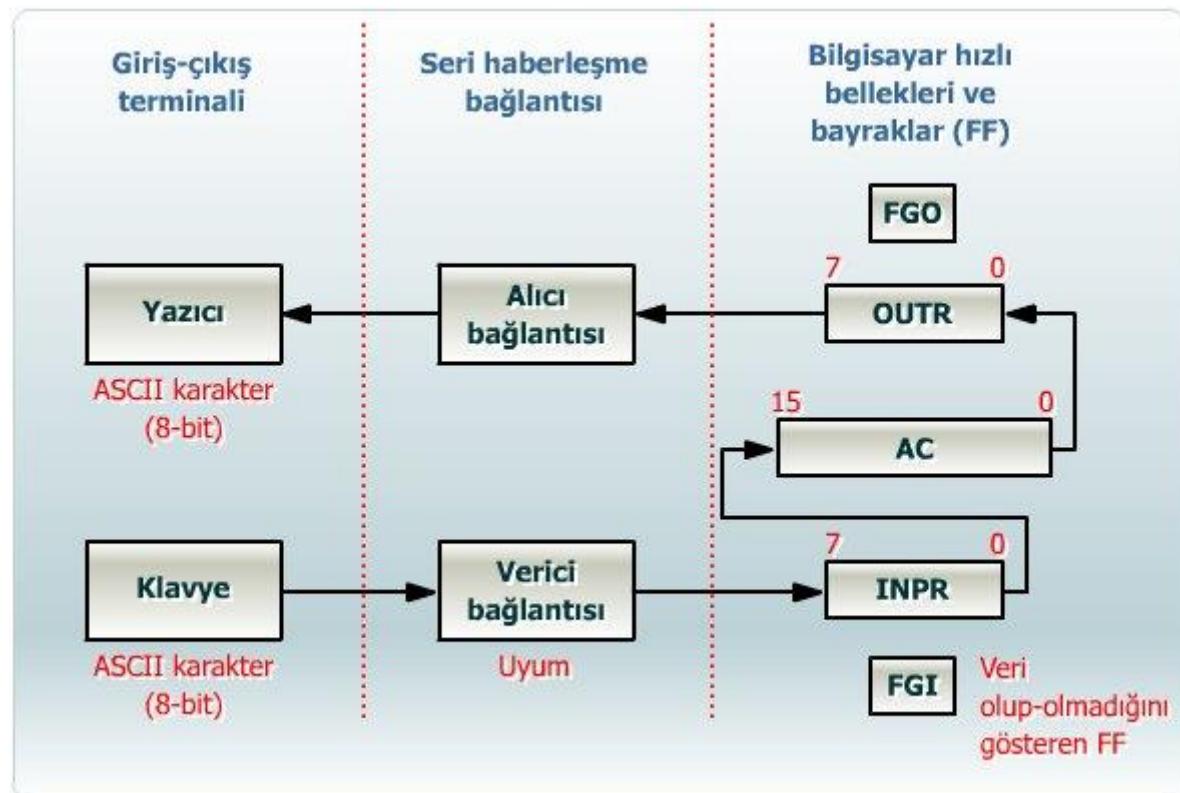


BSA'nın koşturulması örneği

## 5.8 GİRİŞ/ÇIKIŞ ve KESİNTİ KAVRAMI

Bilgisayar dış dünya ile bağlantıyı giriş/çıkış düzenleri ile sağlar. Pek çok giriş/çıkış düzeni arasında en yaygın olanları klavye, tarayıcı, tablet (giriş için) ve yazıcı, ekrandır (çıkış için).

Örnek giriş/çıkışümüz, klavye ve ekran ile temel bilgisayarın bağlantısını "kesinti" kavramı ile işletmektedir. Bunun için giriş düzeni klavyeden basılan her tuşa karşı düşen ASCII karakter seri iletişim bağlantısı yoluyla INPR'ye gelir. FGI bayrağı (FF) yoluyla bu bilgisayara iletilip, kesintinin işlemi yoluyla AC'ye iletilir. FGI bayrağı 0 yapılır. Buradan veri istenilen yere, örneğin ana belleğe transfer edilir. Çıkış düzeni olan ekran, veriyi AC'den alır. Bunun için FGO bayrağı (FF) kullanılır. Veri OUTR'ye gelince bayrak aktive olur, buradan da ekrana transfer edilir. Transfer işlemi bitince bayrak 0 yapılır. Aşağıdaki şekil giriş/çıkış işlemini göstermektedir.



Temel bilgisayarın giriş/çıkış işlemi

Temel bilgisayarın giriş/çıkış komutları aşağıdaki tabloda gösterilmiştir.  $D_7 \ IT_3$  : d bütün giriş/çıkış komutları için ortak koşuludur. Bu koşulda  $D_7 = 1$ ,  $I = 1$ ,  $T_3 = 1$  sonuçta da  $d = 1$  olması gereklidir (AND lojik işlemi).

## GİRİŞ/ÇIKIŞ KOMUTLARI TABLOSU



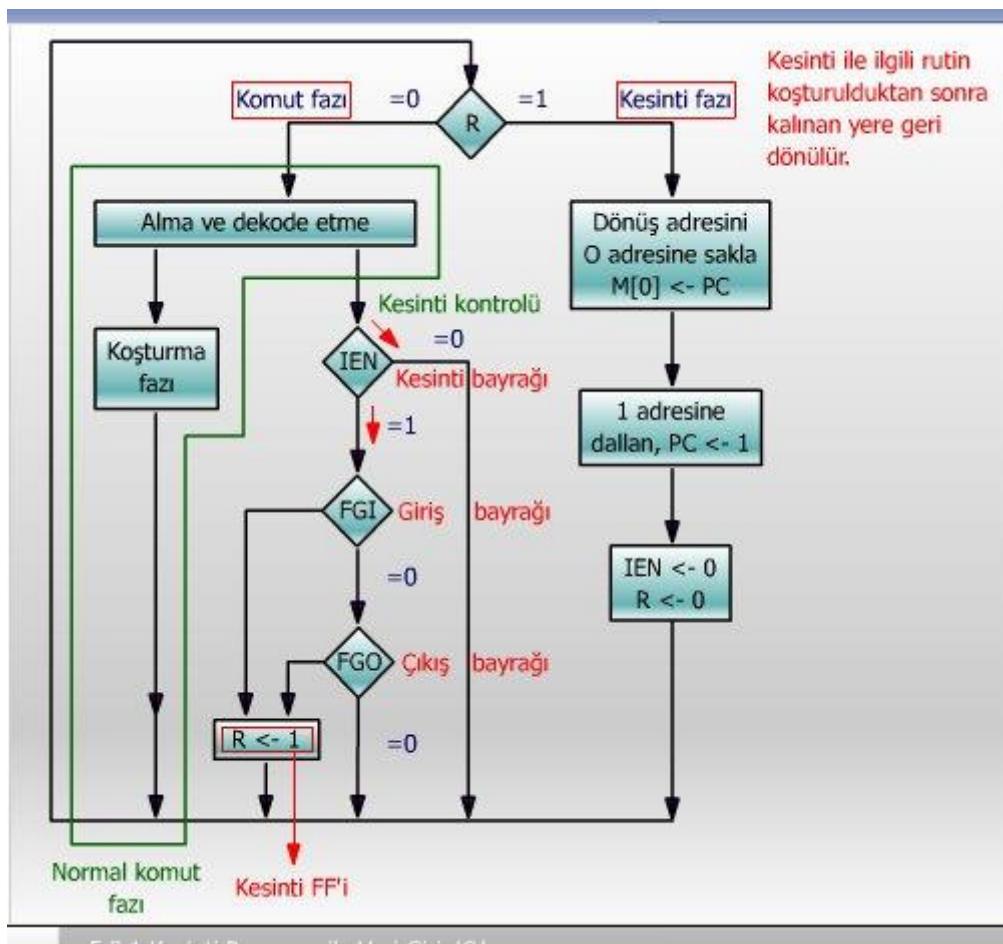
### D7 IT3 : d ve IR(i) = Bi ( IR(011) bit değerleri )

<b>INP</b>	<b>AC(0-7)&lt;=INPR, FGI&lt;=0</b>	<b>( d B<sub>11</sub> )</b>
<b>OUT</b>	<b>OUTR&lt;=AC(0-7), FGO&lt;=0</b>	<b>( d B<sub>10</sub> )</b>
<b>SKI</b>	<b>If (FGI=1) then PC&lt;=PC+1</b>	<b>( d B<sub>9</sub> )</b>
<b>SKO</b>	<b>If (FGO=1) then PC&lt;=PC+1</b>	<b>( d B<sub>8</sub> )</b>
<b>ION</b>	<b>IEN&lt;=1</b>	<b>( d B<sub>7</sub> )</b>
<b>IOF</b>	<b>IEN&lt;=0</b>	<b>( d B<sub>6</sub> )</b>

INP ve OUT komutları veriyi INPR ve OUTR'den AC'ye transfer eder. SKI ve SKO komutları, FGI ve FGO'nun kontrol edilip, INPR ve OUTR'de veri olup, olmadığından anlaşılması sağlar. ION ve IOF kesinti işleminin kontrolü için kullanılır.

### 5.8.1 Kesinti Programı ile Veri Giriş/Çıkışı

Kesinti (interrupt) programı ile veri giriş/çıkışının sağlanması için temel bilgisayarda gerekli yapı değişikliğinin sağlanması gereklidir. Bununla ilgili temel bilgisayarın akış diyagramına bir kesinti fazı eklenmesi gereklidir. Aşağıdaki şekilde kesinti fazı eklenmiş komut fazı görülmektedir.



Kesinti fazı eklenmiş komut fazı

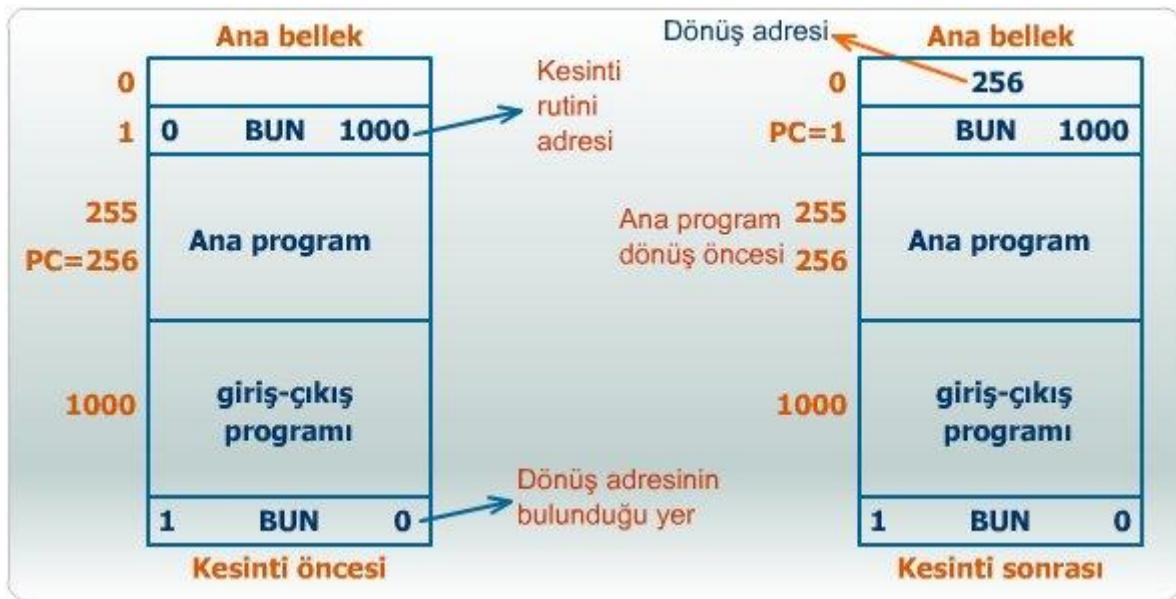
Kesinti fazı son komut fazından sonra başlar. Eğer R FF 'i 1 ise kesinti fazı başlar:

$T_0' T_1' T_2' (IEN)(FGI+FGO): R \leq 1$

R=1 olması IEN=1 veya FGI veya FGO 'nun 1 olmasını bağlıdır. Bu şekilde, kesinti fazı ile komut fazını değiştirmiş oluruz. Kesinti olması durumunda kesinti fazı uygulanır.

### 5.8.1.1 Kesinti Fazı Şeması

Aşağıdaki şekilde kesinti fazının şeması gösterilmiştir.



Kesinti fazının blok şeması

Kesinti fazında dönüş adresini saklayıp, kesinti programını (örneğin, giriş/çıkış programı) koşturup, programın yine eski adresine dönmesini sağlayan mikroişlem yapısı, bu kesinti fazının blok şemasında açıklanan örnek için aşağıdaki gibidir:

$RT_0: AR \leq 0, TR \leq PC$
$RT_1: M[AR] \leq 0, PC \leq 0$
$RT_2: PC \leq PC+1, IEN \leq 0, R \leq 0, SC \leq 0$

İlk zaman işareti sırasında, AR temizlenir, PC içeriği TR (geçici hızlı bellek)'ye aktarılır. İkinci zaman işareti sırasında, PC temizlenir, dönüş adresi bellek adresi 0'da saklanır. Üçüncü zaman işaretinde, PC artırılır, R ve IEN (kesinti aktive bayrağı) 0 yapılır ve SC başa döner.

Kesinti pek çok bilgisayar uygulamasında kullanılan genel ve önemli bir kavramdır. Güç kaynağı kesintisinden program hatalarına (sıfır ile bölmeye) kadar değişik durumlarda kesinti rutinleri kullanılır. Burada bahsedilen giriş/çıkış uygulaması, verimli bir yöntem örneği olmamasına rağmen anlatılmıştır. Giriş/çıkış için günümüz bilgisayarları çok daha verimli yöntemler kullanmaktadır.

## 5.9 TEMEL BİLGİSAYARIN ÖZETLENMESİ

Temel bilgisayar özet olarak aşağıdaki donanım parçalarından oluşmaktadır:



- 4096 kelimeli ve 16 bitlik ana bellek
- Dokuz hızlı bellek elemanı: AR, PC, DR, AC, IR, TR, OUTR, INPR ve SC
- İki dekoder: 3x8 operasyon dekoderi, 4x16 zamanlama dekoderi
- Yedi bayrak (FF): S (başla), I (doğrudan/dolaylı), E (elde, vs.), R (kesinti), IEN (kesinti başlama), FGI ve FGO (giriş/çıkış)
- 16-bit hat grubu
- Kontrol (ve zamanlama) kapıları
- Toplama, lojik ve kaydırma (ALU) devresi ve AC bağlantısı

Temel bilgisayar, tek AC'li bir yapıdır. Daha sonra, genel hızlı bellek yapıları, yiğin yapılarını göreceğiz. Burada tanımlanan hızlı bellek transfer dili veya akış diyagramı tüm yapılar için geçerlidir. Kesinti programı ile değiştirilmiş, komut fazı dış dünya ile ikili bilgi alışverişini sağlayan bir yapıyı sağlamıştır. Temel transfer ve zamanlama kavramlarının öğrenilmiş olması ve diğer karşılaşılan bilgisayar yapılarında benzer kuralların uygulandığı hatırlanmalıdır.

## 5.10 TEMEL BİLGİSAYAR PARÇALARININ TASARIM ÖRNEKLERİ

Temel bilgisayar bileşenlerini tanımladıktan ve bağlantıları yapıldıktan sonra, bu yapının daha önce anlatılan mikroişlemler dizisini verilen zamanlamada koşturması gereklidir. Bununla ilgili kontrol birimi tasarımları önem kazanmaktadır. Burada biz bir örnek olarak, AR 'ye gelen tüm kontrol işaretlerini kullanarak, bu hızlı belleğin  $T_0$ 'dan  $T_{15}$ 'e kadar süren zaman dizisinde hangi transfer ifadelerinde kullanıldığını gösterip, donanım tasarımları örneği verelim. Tüm hızlı bellek ifadeleri ve akış diyagramından yola çıkarak AR'lı ifadeleri bulalım:

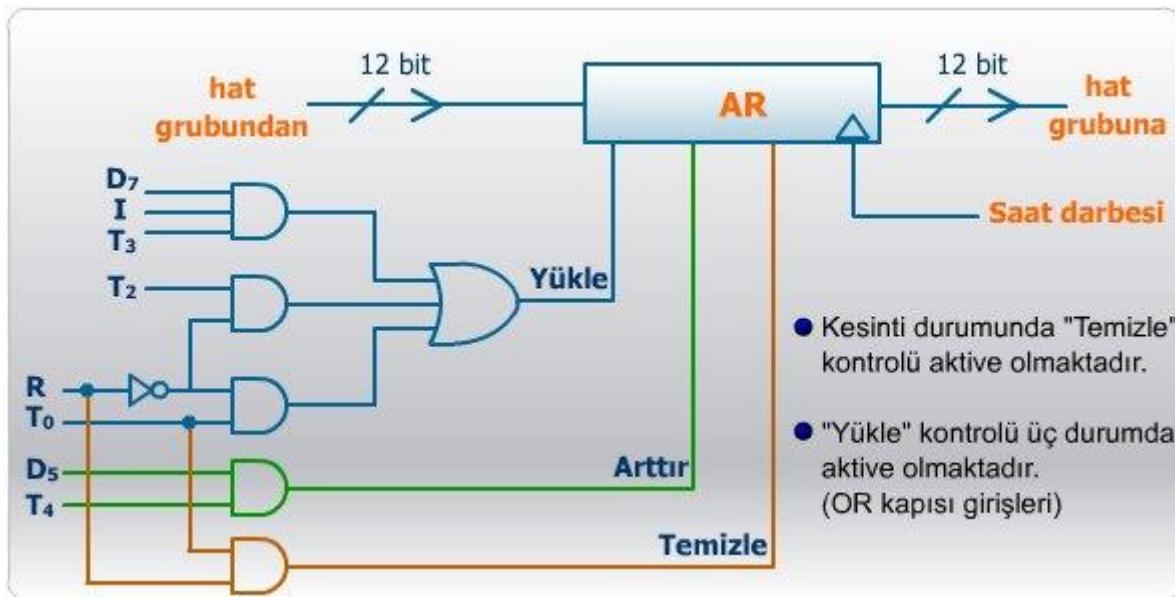
R' $T_0$ : AR <= PC
R' $T_2$ : AR <= IR(0-11)
D <sub>7</sub> 'IT <sub>3</sub> : AR <= M[AR]
RT <sub>0</sub> : AR <= 0
D <sub>5</sub> T <sub>4</sub> : AR <= AR + 1

İlk üç ifadede ana bellek veya hızlı bellekten AR'ye transfer söz konusudur, dolayısıyla AR'nin yükle girişi aktive edilir. Dördüncü giriş AR'nin temizle aktivasyonudur. Sonuncu ise arttır girişinin kullanımını gerektirir.

Yükle (AR) = R' $T_0$ + R' $T_2$ + D <sub>7</sub> 'IT <sub>3</sub>
Temizle (AR) = RT <sub>0</sub>

$$\text{Arttır (AR)} = D_5 T_4$$

Aşağıda AR belleğinin kontrol kapıları gösterilmiştir. Tasarım K-map indirgemesinden sonra minimum sayıda eleman ile yapılır.



## Bölüm Özeti

- Bu bölümde, temel bir bilgisayar yapısı belli önkabullerle tanıtılmış, işlemlerinin hızlı bellek transfer işlemleri ile nasıl gösterildiği anlatılmıştır. Bilgisayarın tasarımları çok detaya girilmeden anlatılmış, bugün kullanılan bilgisayarlarla arasındaki bağlantılar kurulmuştur. Ayrıca bilgisayarın iç yapısı ve hızlı bellekler ile gerçekleşen mikroişlemler arasında ilişki kurulup işletim detayları gözden geçirilmiştir.

## BÖLÜM - 6

# TEMEL BİLGİSAYARDA ASSEMBLER PROGRAMLAMA

Bu bölümde aşağıdaki konular incelenecaktır:

- Başlangıç
- Bilgisayar yapısı ve komutlarının tanıtımı

- Makina dili, onaltı kod, assembler dili ve yüksek seviyeli programlama dili komutlarının tanıtımı
- Assembler dili yazılım detaylarının tanıtılması
- Assembler özellikleri
- Çevrimler, aritmetik ve lojik operasyonlarla programlama örnekleri
- Kaydırma işlemleri
- Alt rutin (subroutine)
- Giriş-çıkış programlaması

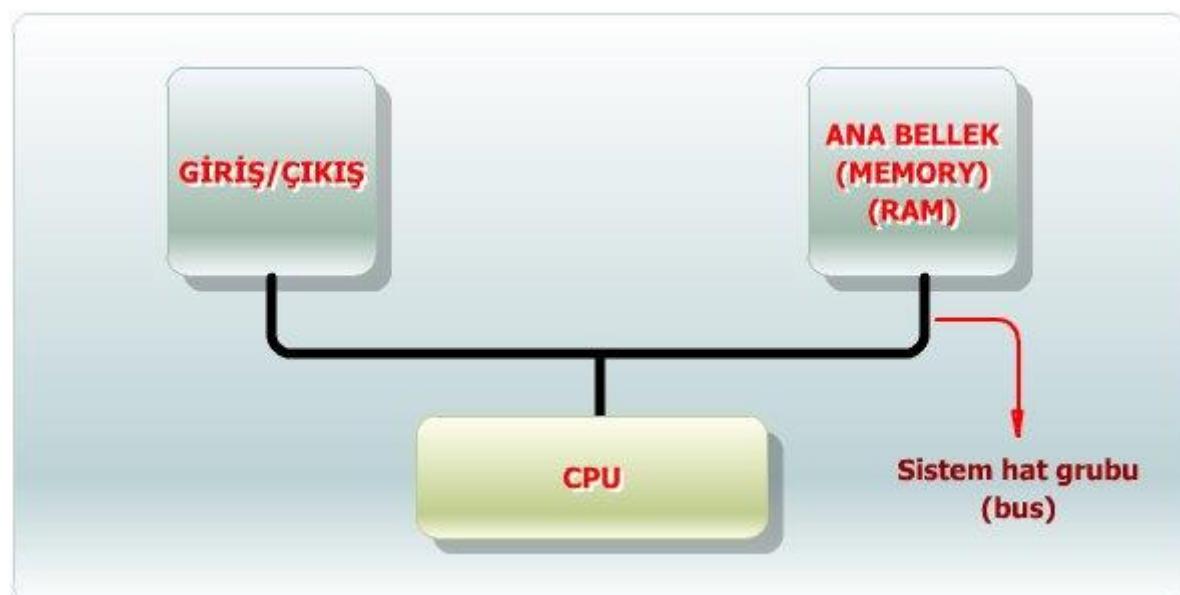
## Bölüm Hedefi

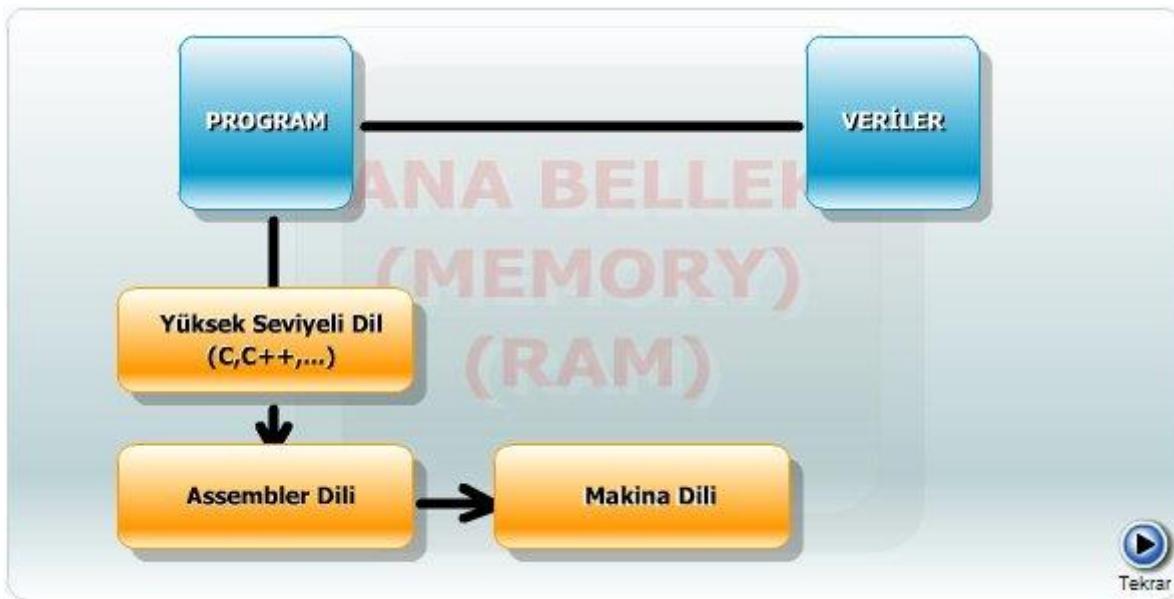
Önümüzdeki iki haftada, temel bir bilgisayarda assembler programlamanın nasıl yapıldığını örneklerle anlatacağız. Bilgisayarın tasarımından bildiğimiz komutların yapılarıyla, makina dili olarak tanıtılan ikili sayıların assembler dili haline getirilmesini açıklayacağız. Sonra çeşitli programlama örnekleri ile bu dildeki programlamanın detaylarını inceleyeceğiz.

Ayrıca, temel bilgisayarda tanımlanan tam (complete) fakat hızlı olmayan komut setinden oluşan assembler dilinin uygulamalarını göstereceğiz. Aritmetik ve lojik operasyonları, çevrim (loop) kavramını tanıtıp alt rutinlerden (subroutine) bahsedeceğiz. Son olarak, giriş ve çıkış programlama örneğini tanıtacağız.

### 6.1 BAŞLANGIÇ

Bir sayısal bilgisayarın, donanım ve yazılım olmak üzere iki vazgeçilmez parça olduğunu belirtmiştik. Kullanılan tüm sayısal bileşenler (MUX, dekoder, hızlı bellek, ana bellek vs.) donanım kısmını ve her türlü yazılan ve saklanan programlar ise yazılım kısmını oluşturmaktadır. Aşağıda temel bilgisayarın donanım ve yazılım ilişkisi açıklanmıştır. Kullanıcı tarafından yazılan ve ana bellekte saklanan program ve veri bilgisayarın işlem yapmasını sağlamaktadır. "Sistem programı" denilen ve bilgisayarın iç çalışmasını sağlayan programlar dışında, kullanıcı tarafından yazılan programlar çeşitli seviyelerde olmaktadır. "Yüksek seviyeli" (high level) programlar C, Fortran, Pascal İngilizce dilinde yazılmış, dolayısıyla, kullanıcıya daha yardımcı olan programlardır. Bugün bu programlar daha da geliştirilip, "nesne programlama" dilleri (C++, Java, Pearl, vb.) haline getirilerek Internet'te modüler olarak kolayca koşturulabilen bir yapıya getirilmiştir.





Temel bilgisayarın donanım ve yazılımı

### 6.1.1 Bilgisayarda Koşturulabilen Makina Kodu Elde Edilmesi

"C" gibi başka bir dil, İngilizce kolayca hatırlanabilen komutlardan oluşmaktadır. Assembler dili ise belli işlemleri yapmaya yarayan "sembol" lerden oluşmaktadır. Daha alt seviyeli bir dil grubudur. Bilgisayarın işlemcisine bağlı olarak, yapımcı firma tarafından tasarım sırasında belirlenmiştir. Makina dilindeki 0 ve 1'lerden oluşan dizileri hatırlamak veya onaltılı kodlar üzerinde çalışmak yerine, assembler dili ile çalışmak çok daha kolay olmaktadır.

Yüksek seviyeli bir dilde yazılan programı koşturabilmek için "koşturulabilen" (executable) programın veya ikili kodun (makine kodu) elde edilmesi gereklidir. Bunun için derleyiciler (compiler) mevcuttur. Örneğin, C derleyicisi ile C programlama dilinde yazılmış olan bir programı koşturulabilecek formata, yani ikili makina diline döndürmek mümkün olur.

Assembler dilinde yazılmış bir program ise "assembler" vasıtıyla makina diline çevrilmektedir. Elde edilen ikili kodlar, bilgisayar işlemcisinde koşturularak sonuçlar elde edilir ve gösterilir. Aşağıda assembler yoluyla makina kodu elde edilmesi gösterilmiştir.



Bilgisayarda koşturulabilen makina kodu elde edilmesi

## 6.2 MAKİNA DİLİ, ONALTILI KOD, ASSEMBLER DİLİ ve YÜKSEK SEVİYELİ PROGRAMLAMA DİLİ KOMUTLARININ TANITILMASI

Bilgisayarda yazılan programlar aşağıdaki kategorilerden birinde olabilir:

- ▶ İkili kod veya makine kodu, komutların 0 ve 1 olarak bilgisayarda saklandığı şekilde gösterilimdir.
- ▶ Onaltılık kod (veya sekizli kod), ikili kod gösterilimine eşit olarak kabul edilip, kullanılmaktadır.
- ▶ Sembolik kod, komutlar için (Çoğunlukla akılda tutulması kolay olan) sembollerin kullanıldığı gösterilimdir.

Tekrar

Sembolik komutlar özel bir program yoluyla ikili koda çevrilir. Bu programa "assembler" denir. Bu sembollerin oluşturduğu programa ise "assembler dili" adı verilir.

Yüksek seviyeli programlama dili, kullanıcının bir problemi çözümünde düşünme yoluna yakın olarak kullanacağı ifadelerden oluşur. Örneğin, C dilinde "for", "while" ve "switch" anahtar kelimeleri düşünme mantığına uygun可以说ablecekel kelimelerdir.

### 6.2.1 Temel Bilgisayarın Assembler Komutları

Aşağıdaki tabloda temel bilgisayarda kullanılan tüm assembler komutları gösterilmiştir. Bu komutlar akılda tutulması çok daha kolay sembollerdir.

KOMUT ADI	AÇIKLAMA
AND	Bellek kelimesi ile AC lojik "and"
ADD	Bellek kelimesi ile AC' yi "topla"
LDA	Bellek kelimesini AC'ye yükle
STA	AC'ye bellek kelimesini yükle
BUN	Koşulsuz dallanma
BSA	Dallan ve geri dönüş adresini sakla
ISZ	Arttır ve sıfır ise sonraki komuta atla
CLA	AC'yi temizle
CLE	E bayrağını temizle
CMA	komplement AC
CME	komplement E
CIR	AC'yi sağa dairesel kaydır

<b>CIL</b>	AC'yi sola dairesel kaydır
<b>INC</b>	AC'yi arttır
<b>SPA</b>	Eğer AC>0 ise sonraki komuta atla
<b>SNA</b>	Eğer AC<0 ise sonraki komuta atla
<b>SZE</b>	Eğer E=0 ise sonraki komuta atla
<b>HLT</b>	Halt
<b>INP</b>	AC'ye karakter gir (INPR'den)
<b>OUT</b>	AC'den karakter gönder (OUTR'ye)
<b>SKI</b>	Giriş_bayrağı=1 ise sonraki komuta atla
<b>SKO</b>	Çıkış_bayrağı=1 ise sonraki komuta atla
<b>ION</b>	Kesinti (interrupt) geçerli (on)
<b>IOF</b>	Kesinti geçersiz (off)

## 6.2.2 İkili Kod ve Assembler Kodu Arasındaki İlişki

İkili kod, onaltılı kod ve assembler kod arasındaki ilişkiyi göstermek için örnek olarak iki komutun üç koddaki değerleri aşağıda gösterilmiştir. Bilgisayar gösterilimi yerine "STA 012" ve "LDA 002" komutlarını kullanmak çok daha uygundur.

KOMUT	İKİLİ KOD	ONALTILI KOD
STA 012	(0011 0000 0000 0110) <sub>2</sub>	(3006) <sub>H</sub>
LDA 002	(0010 0000 0000 0010) <sub>2</sub>	(2002) <sub>H</sub>



Tekrar

## 6.2.3 Karşılaştırmalı Program Örneği

Son olarak C dilinde yazılmış, iki sayının toplamını alan bir programın temel bilgisayar assembler dilinde yazılımını göstererek aralarındaki farkları belirtelim: C dilinde iki sayının toplamı aşağıda gösterilmiştir.

```
main ()
{
    int x=-20, y=40; /* Toplanacak iki sayı ve değerleri */

    int z; /* Sonuç değişkeni */

    z = x + y ; /* Toplama işlemi */
```

}

Aynı sayıları tanımladığımız assemblers dilinde toplarsak, aşağıdaki programı yazmamız gereklidir.

İki dilde yazılan küçük programlar karşılaştırılınca C dilinde yazılan programın ne kadar kısa olduğu gözlenebilir. Bu programın komutları tamamen anlaşılabilen ifadelerdir.

Assembler komutları daha temel (alt seviye) komutlar olup, her bir operasyonu yapabilmek için hızlı bellek transferlerini tanımlamak gereklidir.

<b>ORG 0</b>	/ Programın başlangıcı 0 belirtilmiştir
<b>LDA x</b>	/ x' i AC e yükle
<b>ADD y</b>	/ y' yi topla
<b>STA z</b>	/ Sonucu z'ye sakla
<b>HLT</b>	/ bilgisayarı durdur
<b>x, DEC -20</b>	/ Onlu operand x
<b>y, DEC 40</b>	/ Onlu operand y
<b>z, DEC 0</b>	/ Onlu operand z
<b>END</b>	/ Assembler programının sonu

#### Temel bilgisayar assemblerleri ile iki sayının toplanması

Yukarıdaki assembler programı -20 ile 40 sayısını toplamak için bölüm 5'te tanıtılmış olan AC (akümülatör) hızlı belleğini kullanmaktadır. Burada x=-20 değeri önce AC'ye saklanır, sonra ana bellekteki y=40 değeri ile ADD assembler komutu aracılığı ile toplanır. Sonuçta elde edilen sayı Z adresinde saklanır.

x,y,z etiketleri ile tanımlanan on'luk (DEC) operand değerlerinin assembler programda ayrı bir yerde gösterildiğine dikkat edilmelidir.

## 6.3 ASSEMBLER DİLİ YAZILIM DETAYLARININ TANITILMASI

Tüm programlama dilleri gibi, burada tanıtmış olduğumuz assembler dili de belli kurallardan oluşur. Yazılan programların doğru çevrilmesi (assembling) için, kullanıcının bu kurallara tamamıyla uyması gereklidir.

Burada tanıtılan assemblelerin temel birimi "satır kodu" dur.

- ▶ Etiket (label) alanı: sembolik adresi tanımlar.
- ▶ Komut (instruction) alanı: komut veya psödo-komutu tanımlar.
- ▶ Açıklama (comment) alanı: açıklamayı içerir.

Tekrar

Temel bilgisayar assembleri ile iki sayının toplanması örneğinde END komutundan önceki üç satırda, başlangıç ifadeleri x, y, z etiketler ve etiket alanını, END, LDA, vb. komut alanını ve / (ters kesme) ile başlayan kısım ise açıklama kısmını gösterir.

### 6.3.1 Assembler Diline Ait Örnek

Aşağıda bir örnek satır kodu ve alanları gösterilmiştir.

Etiket	Komut	Açıklama
	STA z	/ Sonucu z'ye sakla
y	DEC 40	/ Onlu operand y

Çeşitli örneklerde tanımlanan semboller de komut alanında kullanılır:

CMA	/ Komplement AC, ana bellek referanslı olmayan komut
AND	OPR / AND operand, doğrudan ana bellek referanslı adres
ADD	PTR I / ADD dolaylı referanslı operand, dolaylı refereanslı adres

Burada kullanılan **OPR**: operand, **PTR**: gösterici (pointer) ve **I**: dolaylı (indirect) anlamında kullanılan anahtar kelimelerdir.

Aşağıdaki tabloda bazı komutlar ve açıklamaları verilmektedir:

ORG M	M onaltılı sayısı, sonraki satırda komut veya bellek adresini belirtir
END	Programın sonunu belirtir
DEC N	İkili hale dönüştürülmüş işaretli onlu sayı
HEX N	İkili hale dönüştürülmüş işaretli onaltılı sayı

#### 6.3.1.1 Assembler Diline Ait Örnek Soru

**ÖRNEK**

- z = (x - 2\*y) işleminin assembler programını yazınız.

**ÇÖZÜM****Örneğin Çözümü**

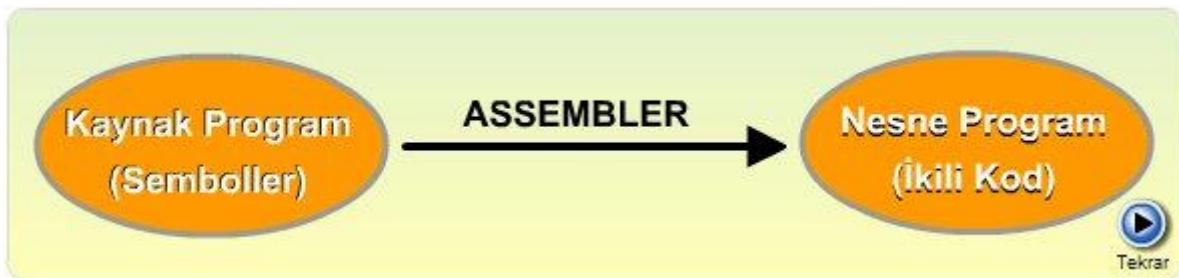
<b>ORG 100</b>	/ Programın başlangıcı 100 belirtilmiştir
<b>LDA y</b>	/ y'yi AC'ye yükle
<b>CIL</b>	/Dairesel sola kaydırma (E bayrağı yoluyla)
<b>CMA</b>	/Komplement AC
<b>INC</b>	/AC'yi arttır
<b>ADD x</b>	/ x'i topla
<b>STA z</b>	/ Sonucu z'ye sakla
<b>HLT</b>	/ Bilgisayarı durdur
<b>x, DEC 20</b>	/ Onlu operand x
<b>y, DEC 40</b>	/ Onlu operand y
<b>z, DEC 0</b>	/ Onlu operand z
<b>END</b>	/Assembler programının sonu

z=(x-2\*y) işleminin assembler programı

Bu programda ORG 100 ve END psödo komutları programın sonunu bașını belirtir. Değerleri 20 ve 40 olan x ve y sayıları etiketlerle HLT operasyonundan sonra tanımlanmışlardır. Yapılan işlem ( $x - 2 * y$ ) işlemi olup, sonuç z'ye saklanır. Burada bir 2 ile çarpması CIL ile tanımlanmıştır. Sonra ise, CMA ve INC komutları ile 2'nin komplementi alınarak, çıkarma işlemi yapılmıştır. Bu işlemin sonucu  $z = (40 - 2 * 20) = 0$  olarak elde edilir.

## 6.4 ASSEMBLER ÖZELLİKLERİ

Assembler sembolik dili programından ikili makina kodu üreten programa verilen isimdir. Girişteki sembolik programa "kaynak program" (source program), çıkışta üretilen ikili koda da "nesne program" (object program) adı verilir. Aşağıda satır kodu (line of code) temelli assembler tanıtılmıştır.



Satır kodu temelli assembler

### 6.4.1 Sembollerden İkili Koda Geçiş

Sembollerden ikili koda geçiş, iki geçişli assembler (two pass assembler) programı ile yapılır. İki geçişli assembler, sembolik programı iki kere gezer veya tarar: birinci geçiş ve ikinci geçiş. Birinci geçişte, kullanıcının (veya programcının) tanımlamış olduğu semboller ile ikili karşılıkları arasında ilişki kuracak bir tablo oluşturulur. İkinci geçişte ise ikili değerlere dönüşüm yapılır.

Birinci geçişte, komutların yerlerinin takip edilmesi için yer sayacı (location counter-LC) kullanılır. Yer sayacı (LC) son komutun yerini gösterir. ORG psödo kodu LC'yi sıfırlar. Her bir "satır kodu" işlem gördükten sonra, sonraki satır için LC bir arttırılır.

Birinci geçişte gerçeklenen işlev aşağıdaki gibi tanımlanabilir: LC başlangıçta 0 yapılır. Sembolik kodun etiket kısmı olup olmadığı kontrol edilir (etiketten sonra virgül olup olmadığı araştırılır). Eğer etiket yoksa, komut alanında ORG veya END psödo komutları olup olmadığı araştırılır. Eğer ORG ise LC=0, eğer END ise birinci geçiş sonlandırılır. Eğer etiket varsa, etiket ikili eşdeğeri ve LC değeri ile birlikte "adres-simbol" tablosuna saklanır.

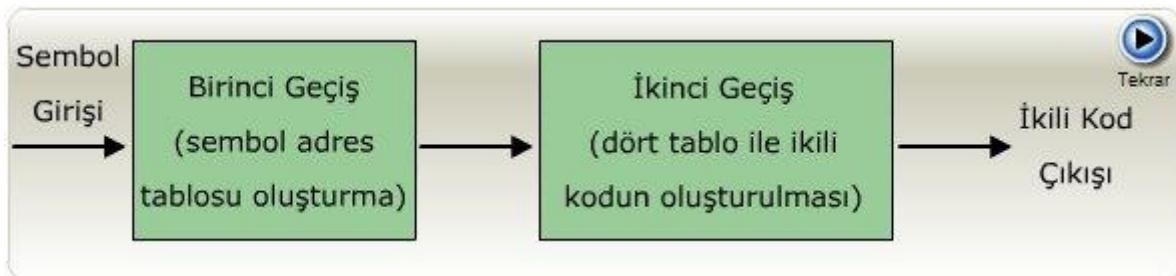
İkinci geçişte, bilgisayar komutları "tablo-bakma" işlemi ile ikili sayılaraya dönüştürülür. Tablo-bakma işlemi, tabloda saklı olan içeriklerin girdi değerleri yoluyla bulunması işlemidir. Assemblerler dört tablo kullanır:

- Ana bellekten bir komutun alınması (fetch cycle)
- Komutun dekode edilmesi (decode cycle)
- Efektif adresin hesaplanması ve komutun dolaylı adresinin olup olmadığını araştırılması
- Komutun koşturulması (execution cycle)



Psödo komut tablosunun girdileri dört sembolden ibarettir: ORG, END, DEC ve HEX. Her bir simbol assemblerlerin ilgili alt rutininin koşturulmasını sağlar. Ana bellek referanslı komutlar tablosu, temel bilgisayar tasarımindan hatırlanacağı gibi, 7 eleman içerir. Ana bellek referanslı olmayan komutlar tablosu ise kalan 18 elemanı içerir. Adres-simbol tablosu ise, bilindiği gibi, birinci geçiş sırasında oluşturulur.

İkinci geçiş sırasında, karşılaşılan her bir simbol, hangi tabloya ait olduğu bütün olasılıklar kontrol edilerek araştırılır ve sonuçta ikili koda veya makina koduna dönüştürme işlemi END psödo komutuna ulaşılana kadar devam eder. Aşağıda Assemblerlerin işlevi blok diyagramla açıklanmaktadır.



Assembler'in işlevi

## 6.4.2 Assembler ile Hataların Belirlenmesi (Error Diagnostic)

Assembler'in bir diğer önemli görevi, sembolik programda oluşabilecek hataları kontrol etmektir.

HATA	AÇIKLAMA
komut tablolarında bulunmayan bir simbolün belirlenmesi	Bu durumda, assembler simbolü ikili sayıya dönüştüremez ve bir "hata mesajı" verir.
programda etiket olarak tanımlanmamış bir simbolik adresin bulunması	Assembler bu durumda satır kodunu uygun olarak dönüştüremeyecektir, çünkü ikili eşdeğer sayı adres-simbol tablosundan bulunamayacaktır

Burada anlatılan assembler'in basit, pratikte ise çok daha karmaşık bir program olduğuna dikkat etmek gerekir. Bugünkü bilgisayarlar kullanıcıya çok daha fazla esneklik vermektedir. Daha akıcı ve daha çok özelliği bulunan assembler'lar kullanıcıya daha kolay assembler program yazma olanağı sağlamaktadır.

Son olarak, derleyici (compiler) ile assembler arasındaki farkı belirlemek gerekir. Derleyici yüksek seviyeli bir dilden (C gibi) makina koduna dönüşüm yapar. Derleyici yapısı ve çalışması daha karmaşıktır. Derleyiciyi daha iyi anlamak için "sistem programlaması" bilmek gereklidir. Bazı derleyiciler ise ara adım olarak assembler kullanmaktadır.

## 6.5 ÇEVİRİMLER, ARİTMETİK, LOJİK OPERASYONLAR ile PROGRAMLAMA

Bu bölümde temel bilgisayar assembler dili ile çeşitli program örneklerini göstereceğiz: Bunlardan ilki, ana bellekte arka arkaya saklanmış 20 sayıyı toplayan programdır. C dilinde 20 tane sayıyı toplamak için yazacağımız program aşağıdaki gibidir.

```

main()
{
    int sum, a[20], i, K=20;

    sum=0;

    for ( i=0; i<=K-1; i++)

        sum= sum+a[i];

}

```

Programda "for" çevrimi 20 defa koşturularak "sum" değeri a[20] dizisinin 20 elemanın toplanmasını sağlar.

Aşağıda aynı programın assembler dilinde yazılımı görülmektedir. Program başlangıcı ORG psödo komutu ile 200 olarak tanımlanmıştır. Ana bellek adresi 250'den itibaren ise 20 operand saklanmıştır. Ana bellek adresi 400'de, bir gösterici (PTR) yer almaktadır. Bu gösterici yoluyla 20 operanda ulaşılmakta ve bir SAYICI ile indeks değerleri kontrol edilmektedir. Çevrim, aynı isimli etiketli satırla başlar ve koşulsuz dalandırma komutu (BUN) ile sonlanır. Çevrim içerisinde, 20 sayı sıra ile "ADD PTR I" komutu ile AC'ye toplanır, arttır ve 0 ise sonraki komuta atla (ISZ) komutlarıyla gerek sayı olarak, gerekse içeriğinin 0'dan farklı olması ile kontrol edilir. Koşullar sağlandığında çevrimden çıkışlıp, AC değeri TOPLAM adresine saklanır.

<b>ORG 200</b>	/ Programın başlangıcı 200 belirtilmiştir
<b>LDA ADR</b>	/ ADR'yi AC'ye yükle
<b>STA PTR</b>	/ PTR'ye sakla
<b>LDA SAYI</b>	/ -20'yi yükle
<b>STA SAYICI</b>	/ SAYICI'ya yükle
<b>CLA</b>	/ AC'yi temizle
<b>ÇEVİRİM, ADD PTR I</b>	/ AC yoluyla operandları topla
<b>ISZ PTR</b>	/ PTR'yi arttır
<b>ISZ CTR</b>	/ CTR'yi arttır
<b>BUN ÇEVİRİM</b>	/ Çevrimi tekrarla

<b>STA TOPLAM</b>	/ Toplam TOPLAM'a saklanır
<b>HLT</b>	/ Bilgisayarı durdur
<b>ADR, HEX 250</b>	/ 20 operandın ilk adresi
<b>PTR, HEX 400</b>	/ Göstericinin (PTR) yeri
<b>SAYI, DEC -20</b>	/ SAYICI'nın başlangıç değeri
<b>SAYICI, HEX 0</b>	/ SAYICI'nın yeri
<b>ORG 250</b>	/ Operandlar 250'den başlayarak saklanmıştır
<b>DEC 18</b>	/ İlk operand
.....	
.....	
<b>DEC 63</b>	/ Son (20'nci) operand
<b>END</b>	/ Assembler programının sonu

Assembler programında tanımlanan ADR, PTR, SAYI ve SAYICI etiketleri çeşitli HEX ve DEC değerlerin tanımlanmasında kullanılmaktadır.

### 6.5.1 Assembler Dilinde Çarpma ve Bölme İşlemi

Daha önce yazılan programda 2 ile çarpma işleminin, temel bilgisayar komutları kullanılarak, sola kaydırma olarak gerçeklendiğini belirttik. 2'ye bölme işlemi için ise sağa kaydırma kullanılır. Kullandığımız assembler dilinde birden fazla komut ile gerçekleşen çarpma ve bölme işlemi, bugün kullanılan kişisel bilgisayarlarda çarpma ve bölme işlemi yapan devreler nedeniyle tek bir komutla (örneğin MUL, DIV gibi) hızlı olarak gerçekleştirilebilmektedir. Ayrıca günümüz bilgisayarlarında çift haneli doğruluklu (double precision) işlemler için donanım yapıları ve komutları tasarlanmıştır. Temel bilgisayar, çift haneli doğruluklu sayıları tek haneli 2 sayı olarak yorumlayıp işlem yapar.

İkinci programlama örneğinde  $(x.y)' + z$  lojik işlemini yapan programı yazalım. Aşağıda  $(x.y)' + z$  lojik işlemlerini gerçekleyen program gösterilmiştir.

<b>ORG 100</b>	/ Programın başlangıcı 100 belirtilmiştir
<b>LDA x</b>	/ x'i AC'ye yükle
<b>AND y</b>	/ Lojik AND x ve y
<b>STA T</b>	/(x.y)'yi T'ye sakla
<b>LDA z</b>	/ z'yi AC'ye yükle
<b>CMA</b>	/ Komplement AC
<b>AND T</b>	/ (x.y).z' ifadesini oluştur
<b>CMA</b>	/ (x.y).z' = (x.y)' + z sonucunu oluşturmak
<b>STA SONUÇ</b>	/ Sonucu SONUÇ'a sakla
<b>HLT</b>	/ Bilgisayarı durdur
<b>x, BIN 1</b>	/ ikili operand x
<b>y, BIN 1</b>	/ ikili operand y
<b>z, BIN 0</b>	/ ikili operand z
<b>T, BIN</b>	/ Geçici bellek elemanı
<b>SONUÇ, BIN</b>	/ikili operand SONUÇ
<b>END</b>	/ Program sonu

Bu programda, komut setinde OR komutunun olmaması nedeniyle ilkin AND işlemi sonrada NOT (komplement) işleminin kullanılıldığı izlenmelidir. Diğer bir nokta ise, ikili sayılar üzerinde tanımlanan bu işlemlerin BIN psödo kodu ile tanımlanan bellek elemanlarına uygulanmasıdır. AND lojik işleminin akümülatör AC'ye yüklenen eleman ile ana bellekte saklanan elemana uygulanmakta olduğu gözlenmelidir (ana bellek referanslı komut).

## 6.6 KAYDIRMA ISLEMLERI

Temel bilgisayarın kaydırma işlemleri CIR: dairesel sağa kaydırma ve CIL: dairesel sola kaydırmadır. Lojik kaydırma ve aritmetik kaydırma için birkaç komut kullanmak gereklidir.

Lojik kaydırma, kaydırılan taraftan 0 ekleyerek yapılır. Bu ise E=0 (bayrak veya FF sıfırlanır) yapılarak sağlanır.

**Lojik sağa kaydırında kullanılan komutlar:**

- **CLE**
- **CIR**

**Lojik sola kaydırında kullanılan komutlar:**

- **CLE**
- **CIL**

### **6.6.1 Aritmetik Kaydırma**

Aritmetik kaydırma, negatif sayıların gösterilimine bağlıdır. Temel bilgisayarda 2'li komplement gösterimi kullanılır. Bu durumda, en soldaki işaret bitinin aynı kalması gereklidir. Aritmetik sağ kaydırında, E FF'sine işaret biti ile aynı değer verilir.

#### **Aritmetik Sağa Kaydırma**

- |            |                                      |
|------------|--------------------------------------|
| <b>CLE</b> | / E=0                                |
| <b>SPA</b> | / Eğer AC>0 (pozitif ise) atla, E=0  |
| <b>CME</b> | / AC>0 (negatif) ve E=1              |
| <b>CIR</b> | / E ve AC' yi dairesel olarak kaydır |

Aritmetik sola kaydırında, en az değerli bit (least significant bit) pozisyonundan 0 eklenmelidir. Bu E=0 yapılarak yapılabilir. İşaret biti kaydırma sırasında değişimmemelidir. İşlem sonrası E biti ve işaret biti karşılaştırılmalıdır. Eğer iki değer aynı ise işlem doğru olarak tamamlanmıştır. Eğer farklı ise taşıma (overflow) olduğunu gösterir. Kaydırma yapılmadan önceki sayı değeri çok büyük olduğundan, kaydırma sonucu akümülatör AC kapasitesi yetersiz kalır.

### **6.7 ALT RUTİN (SUBROUTINE)**

Alt rutin (subroutine), aynı program kodunun defalarca değişik yerlerde tekrarlanmasılığını öner. Ortak komutlar bir kere yazılır, sonra programın değişik yerlerinden birçok defa çağrılabılır. Alt rutin bir görev veya fonksiyonun yapılması için olmuş komutlardan oluşur. Dallanma yoluyla ana programın herhangi bir kısmından bu alt rutine ulaşılabilir. Dönüş adresi bilgisayara saklanır ve alt rutin koşturulduktan sonra ana programa dönülür. Bilgisayarlar özel komutlarla alt rutine geçiş ve geri dönüşü sağlarlar.

Temel bilgisayarda BSA (dallan ve dönüş adresi sakla) komutu ana program ve alt rutin arasında geçişini sağlar. İki örnek ile alt rutinlerin nasıl çalıştığını gösterelim: birinci örnekte AC içeriğinin sola bir kaydırılmasını yapan programı yazalım. Bu işlem, temel bilgisayarda çarpma için sıkılıkla kullanılacağından defalarca kullanılabilecek bir işledir.

Aşağıda alt rutin kullanımı gösterilmiştir. Örnek programda sola kaydırma işlemi iki kez yapılmaktadır: birincisi x, ikincisi ise y iledir. Dönüş adresi ana bellekte 0 yerine saklanmaktadır. Eğer birden fazla sola kaydırma işlemi yapılması gerekirse KAYDIR alt rutini o kadar sayıda çağrılabılır. Son olarak da MSK verisi ile kaydırılan değerin en az anlamlı biti 0 yapılır.

<b>ORG 100</b>	/ Programın başlangıcı 100 belirtilmiştir
<b>LDA x</b>	/ x'i AC'ye yükle
<b>BSA KAYDIR</b>	/ KAYDIR alt rutinine dallan
<b>STA x</b>	/ x'i sakla
<b>LDA z</b>	/ z' yi AC'ye yükle
<b>BSA KAYDIR</b>	/ KAYDIR alt rutinine dallan
<b>STA z</b>	/ z'i sakla
<b>HLT</b>	/ Bilgisayarı durdur
<b>x, HEX ABCD</b>	/ Onaltıli operand x
<b>y, HEX DBDB</b>	/ İkili operand z
<b>KAYDIR, HEX 0</b>	/ Dönüş adresini buraya saklar
<b>CIL</b>	/ Dairesel sola 1 kez kaydır
<b>AND MSK</b>	/ AC (16) yi 0 yap
<b>BUN KAYDIR</b>	/ Ana programa dönüş
<b>MSK, HEX FFFE</b>	/ Mask operandı: en sol biti 0 yapar
<b>END</b>	/ Program sonu

Örnek programda, KAYDIR alt rutinin çağrımasıyla (subroutine call) 0 adresine dönüş adresi saklanır. Alt rutin koşturulduktan sonra, alt rutinden dönüş (subroutine return) bu adresin geri alınması ile sağlanır.

### 6.7.1 Verinin Bir Adresten Diğer Adrese Taşınması

Bir alt rutin çağrıduğunda, ana programın gerekli veriyi alt rutine taşıması gereklidir. Örnekte bu AC yoluyla sağlanır. Alt rutin çağrılmadan önce veri AC'ye yüklenir. İşlem sonrası veri yine AC'dedir. AC bir giriş ve bir çıkış verisi için kullanılabilir ve genel olarak, alt rutin çağrıran programdan birden fazla parametreyi de alabilir. Bu da ana bellek yoluyla olabilir. Ana bellekte birbiri ardı sıra gelen yerler saklanan parametreler yoluyla parametre geçişleri sağlanabilir. İkinci bir örnekle birden çok parametrenin geçişini sağlar. Bu örnekte, ardı sıra 5 elemanlık veriyi 50 adresinden 75 adresine taşıyan anaprogram tarafından çağrılan TAŞI alt rutinini görelim. Aşağıda 5 elemanı bir adresten diğer adrese taşıyan programı inceleyelim.

<b>BSA TAŞI</b>	/ Alt rutine dallan
<b>HEX 50</b>	/ 5 elemanlık verinin ilk adresi
<b>HEX 75</b>	/ 5 elemanın taşınacağı ilk adres
<b>DEC -5</b>	/ Taşınacak eleman sayısı
<b>HLT</b>	
<b>TAŞI, HEX 0</b>	/ Alt rutin TAŞI
<b>LDA TAŞI I</b>	/ Kaynak adresi AC'ye yükle
<b>STA PT1</b>	/ İlk göstericiyi sakla
<b>ISZ TAŞI</b>	/ Dönüş adresini arttır
<b>LDA TAŞI I</b>	/ Taşınacak adresi AC'ye yükle
<b>STA PT2</b>	/ İkinci göstericiyi sakla
<b>ISZ TAŞI</b>	/ Dönüş adresini arttır
<b>LDA TAŞI I</b>	/ Kaynak adresi AC'ye yükle
<b>STA Sayıcı</b>	/ Sayıcıyı sakla
<b>ISZ TAŞI</b>	/ Dönüş adresini arttır
<b>ÇEVİRİM, LDA PT1 I</b>	/ Kaynaktan elemanı AC'ye yükle
<b>STA PT2 I</b>	/ Taşınacak yere sakla
<b>ISZ PT1</b>	/ İlk göstericiyi arttır
<b>ISZ PT2</b>	/ İkinci göstericiyi arttır
<b>ISZ Sayıcı</b>	/ Sayıcıyı arttır
<b>BUN ÇEVİRİM</b>	/ 5 defa tekrarla
<b>BUN TAŞI</b>	/ Ana programa dönüş
<b>PT1, -</b>	
<b>PT2, -</b>	
<b>Sayıçı, -</b>	
<b>END</b>	/ Program sonu

İki gösterici (pointer) yardımıyla veri bir taraftan diğer tarafa transfer edilir. Sayıcı, tanımlanan 5 elemanın bu transferini sağlar. Alt rutin işlevini tamamladıktan sonra veri 75 adresinden başlar, alt rutin ana programa geri döner ve HLT ile bilgisayar durur.

## 6.8 GİRİŞ/ÇIKIŞ PROGRAMLAMASI

Kullanıcılar çeşitli programlar yazarlar, bu programları koştururlar ve çıkışlarını alırlar. INP komutu ile, klavyede basılan tuştan elde edilen ASCII kodlanmış karakterleri AC (0-7)' ye transfer ederler. OUT komutu ise tersi işlem olan, AC (0-7) bitlerini yazıcı gibi, çıkış düzeneğine transfer eder. Aşağıdaki tabloda bir karakterin giriş ve çıkış için program gösterilmiştir. Giriş ve çıkış bayrağını kontrol eden çevrim bayrak=0 ise devam eder, bayrak=1 olduğunda giriş veya çıkış komutlarının bulunduğu satır atlanır. Karakter ya ana belleğe saklanır ya da basılır. Burada bu işlemin verimli bir işlem olmadığına dikkat edilmelidir. Bugün kişisel bilgisayarlar daha verimli yöntemlerle ara bellek (buffer) ve kesinti (interrupt) işlemi ile daha uygun stratejiler uygulamaktadır.

/ Bir karakterin giriş'i	
GİRİŞ, SKI	/ Giriş bayrağını kontrol et
BUN GİRİŞ	/ Bayrak=0, tekrar kontrol et
INP	/ Bayrak=1, karakter gir
OUT	/ Karakteri bastır
STA CHR	/ Karakteri CHR'ye sakla
HLT	
CHR, -	
/ Bir karakterin çıkış'i (basamak)	
LDA CHR	/ Karakteri AC'ye yükle
ÇIKIŞ, SKO	/ Çıkış bayrağını kontrol et
BUN ÇIKIŞ	/ Bayrak=0, tekrar kontrol et
OUT	/ Bayrak=1, karakter gönder
HLT	
CHR, HEX 00--	/ Bir ASCII karakter

ASCII karakterlerden oluşan bir kelimenin bilgisayara giriş veya çıkış için kaydırma işleminin de yukarıdaki alt rutinlere eklenmesi gereklidir. Tipik olarak, daha önce yapısı açıklanan sola kaydırma alt rutini arkaya arkaya çağrılarak tüm kelimenin INPR veya OUTR hızlı belleklerine kaydedilmesi sağlanır.

### 6.8.1 Kesinti İşlemi ile Giriş/Çıkış

Kesinti işlemi ile yapılan giriş ve çıkışın üstünlüğü, giriş ve çıkış düzeninin başlattığı bir kesinti ile verilerin transferinin sağlanmasıdır. Böylece bayrak kontrol etmek suretiyle kaydedilen uzun bekleme zamanının ortadan kaldırıldığı gözlenmelidir. Böylece, bilgisayar diğer işlevlerle meşgul olabilmektedir. Bu şekilde sadece kesinti işlemi sırasında bilgisayar işlemcisi giriş veya çıkış işlemi yapabilecektir. Kesinti işlemi çoklu programlama ortamlarında (birden fazla programın aynı bellekte saklanması durumunda) çok uygun olmaktadır.

Kesinti işlemi koşturulan programın giriş veya çıkış düzeninin bayrağı 1 yapmasına kadar devam eder. Bayrak=1 olduğunda, bilgisayar koşturduğu programı tamamlar ve kesintiyi onaylar. Bu sırada hızlı bellek içerikleri de saklanır. Dönüş adresi 0 adresinde saklanıp, 1 adresinde bulunan (önceden saklanan) giriş veya çıkış işlemi için kesinti servis rutini koşturulmaya başlanır. Bu işlem giriş/çıkış tamamlanana kadar sürer, sonra 0 adresinde saklanan dönüş adresine geri döndürerek, koşturulan programa kalınan yerden devam edilir. İşlemcinin hızlı bellek içerikleri, kesinti öncesi ve sonrası aynı olmalıdır. Aşağıda giriş/çıkış için kesinti işlemi blok diyagramı gösterilmiştir.



## Bölüm Özeti

- Bu bölümde, temel bir bigisayarda assembler programlamanın nasıl yapıldığı örneklerle anlatılmıştır.

## BÖLÜM - 7

# TEMEL BİLGİSAYARIN MİKROPROGRAMLAMA ile KONTROLÜ

Bu bölümde aşağıdaki konular incelenecaktır:

- Mikroprogram ile kontrolde bellek yapısı
- Adres dizileri ile kontrol, kontrol rutinleri
- Mikroprogram ile kontrolde bilgisayar yapısı ve mikrokomutlar
- Mikroprogramlama
- Kontrol birimi tasarım detayları

## Bölüm Hedefi

Önümüzdeki iki haftada, temel bir bigisayarda mikroprogramlama ile kontrolü tanıtacağız. Donanım ile kontrolü 5. bölümde anlatmıştık. Lojik kapılar ve FF'lerle gerçekleşen donanım ile kontrol hızlı fakat esnek olmayan bir yapı sergilemektedir.

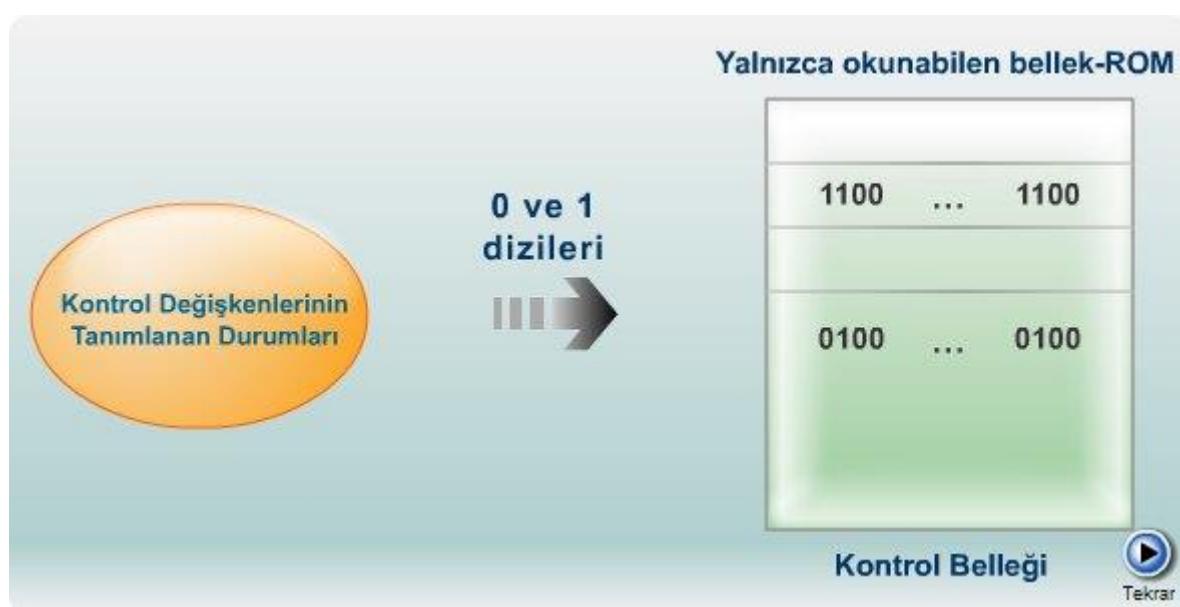
Mikroprogramlama ile kontrol ise diğer seçenek olup, esnek fakat daha yavaş bir yapıya sahiptir ve yazılım ile gerçeklenmektedir.

Bu bölümde, temel bilgisayarda tanımlanan tam (complete) fakat hızlı olmayan komut setinin kontrolü için birbiri ardı sıra saklanan mikroprogram komutlarından yararlanılacaktır. Bununla ilgili olarak yalnızca okunabilen bellek (ROM) ve adres dizileri ile kontrol kavramları tanıtılacaktır. Temel bilgisayar yapısındaki değişiklikler belirtilecek ve değiştirilmiş yapının tasarımı tanıtılacaktır.

## 7.1 MİKROPROGRAM ile KONTROLDE BELLEK YAPISI

Kontrol işlevleri ikili değişkenlerle tanımlanır ve her bir işlev "mikrokomutlar" ile gösterilebilir. Her yeni "durum" için, karşı düşen mikrokomut dizisi koşturulur. Mikrokomutlar bir grup bitten oluşmuştur. Mikrokomutlara karşı düşen bit grularını bellekte saklayıp, sırası gelince koşturma fikri mikroprogram ile kontrolün ana fikridir. Böylece, kontrol değişkenlerinin aldığı değerler, yalnızca-okunabilen bir bellekte (read-only memory-ROM) saklanan 0 ve 1 bit dizileri yoluyla üretilmiş olur.

Her bir bit dizisine "kontrol kelimesi" ve bu kelimelerin ROM'da saklanması yoluyla oluşturulan sisteme mikroprogramlama ile kontrol birimi adı verilir. Mikrokomutlar, mikroprogramlamanın temel elemanı olup, oluşturulan kontrol programlarına "mikroprogram" adı verilir. Kontrol programlarında değişiklik söz konusu olmadığı için bellek birimi olarak, "yalnızca-okunabilen bellek" (ROM) kullanılır. Aşağıda mikroprogramla kontrolün ana fikri açıklanmıştır.



Mikroprogramlı kontrol kullanan bilgisayarda iki bellek bulunur:

- 1. Ana bellek
- 2. Kontrol belleği (ROM)

### 7.1.1 Ana Bellek

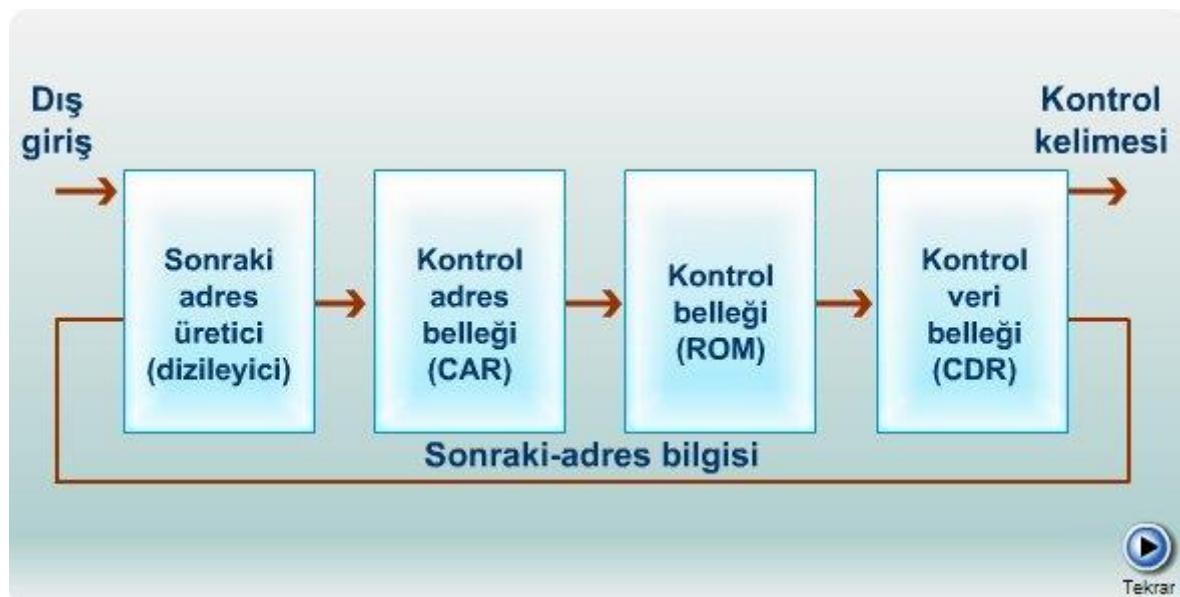
Ana bellek, bilindiği gibi, kullanıcının programlarını ve verisini saklamakta kullanılır. Kontrol belleği ise mikrokomutları saklamak için kullanılır. Mikrokomutlar, hızlı bellek

seviyesinde tanımlanan mikroişlemlerin koşturulması (hızlı bellek operasyonları) için kullanılması gereken kontrol işaretleridir.

Her bilgisayar komutu, kontrol belleğinde saklanmış olan bir dizi mikrokomutu başlatır. Bu mikrokomutlar, komutu ana bellekten alan (fetch cycle), efektif adresi bulan ve komutu koşturan (execution cycle) mikroişlemleri oluştururlar. Sonra kontrol yeni komut için, tekrar başlanacak komut fazına girer.

Mikrokomut ve mikroişlem arasındaki farkı belirlemek gerekir: Mikrokomut kontrol belleğinde saklanan mikroprogram satırlarıdır ve mikroişlemlerin koşturulmasını sağlayan 0 ve 1 dizilerinden oluşur. Mikroişlemler ise hızlı bellek seviyesinde tanımlanan operasyonlardır.

Aşağıda mikroprogramla kontrol organizasyonu gösterilmiştir.



### 7.1.2 Kontrol Belleği



Kontrol belleği yapısı ROM dan olmuştur. Bellekte saklanan mikrokomutlara (0 ve 1 dizilerine) kontrol adres belleği (control address register-CAR) ve kontrol veri belleği (control data register-CDR) yoluyla ulaşılır. CAR kontrol belleğindeki adresin saklandığı yerdir, bu adrese karşı düşen mikrokomut CDR'ye aktarılır. Böylece mikroişlemenin yapılabilmesi için gerekli kontrol işaretleri elde edilir. CDR'ye aynı zamanda "ardışılık işlem belleği" (pipeline register) adı da verilir. Bir komutun koşturulması için gerekli mikroişlem dizisine karşı düşen kontrol işaretleri(mikrokomutlar) dizisine "kontrol rutini" adı verilir. Her bir komut fazı için tanımlanmış olan kontrol rutinleri vardır: komut alma fazı, komut koştirma fazı vb.

Mikroprogramla kontrol organizasyonu için verilen blok diyagramda "bir sonraki mikrokomut adresi"

**1) Sonraki adres bilgisi (mikrokomuttan)**

## 2) Dışardan gelen giriş (komut adreslerinden)

yoluyla bulunur.

Koşturulan mikrokomutu (veya kontrol kelimesi) elde edilirken gelecek adres de dizi üretecinde saklanır. Böylece kontrol rutinlerini sürekli koştururan bir yapı elde edilmiş olunur. Bu işleme kontrol dizilerinin oluşturulması(sequencing) adı verilir.

Mikroprogramla kontrol organizasyonu için verilen blok diyagramı dikkatle inceleyelim. Bir sonraki mikrokomutun adresi, hâlihazırda bulunan adres bilgisi veya dış giriş yoluyla bulunabilir. Bunun için, bir sonraki adres üretici- dizi üreteci(next address generator veya sequencer) kullanılır. Koşturulan mikrokomuta (veya kontrol kelimesi) elde edilirken gelecek adreste dizi üretici girişи olarak kullanılır. Böylece kontrol rutinlerini "sürekli koşturulan" veya program komutları olduğu sürece döngü oluşturan bir yapı elde edilmiş olunur. Bu işleme "kontrol dizilerinin oluşturulması" (sequencing) adı verilir.

Donanımla kontrolde, lojik kapı bağlantıları bir kez yapılınca değiştirilmeden sürekli olarak kullanılır. Hızlı kontrol yapısı oluşturulur. Mikroprogramla kontrolde yazılan program sıkılıkla olmasada değiştirilebilir, esnekdir. Fakat bu yapının hızı, bellek okuma, vs. gibi fazlar nedeniyle daha azdır.

## 7.2 ADRES DİZİLERİ ile KONTROL ve KONTROL RUTİNLERİ

Mikroprogram, kontrol belleğine saklanmış kontrol rutinlerinden oluşur. Komutların gerçekleşmesinde kullanılan mikroişlemler, kontrol belleğine bir "transformasyon" (mapping) ile "adres dizilerine" dönüştürülür. Bu dönüşüm, mikroişlemenin operasyon kodu kısmına yapılan ekleme ile mümkün olur. Aşağıda adres transformasyonu açıklanmaktadır. 4 bitlik operasyon kodu 7 bitlik mikrokomut adresine dönüştürülmektedir. Kontrol belleği  $2^7=128$  adreslik bir kapasiteden oluşur. Mikroişlemler ve bunu gerçeklemek için gerekli olan mikrokomutlar arasında bağlantı kurulmuş olur.



Adres transformasyonu sonucunda, komutlardan kontrol rutinlerine geçmek mümkün olur.

### 7.2.1 Mikrokomut Adreslerinin Seçilmesi

Kontrol belleğinde aşağıdaki adres seçme olanaklarının sağlanması gereklidir:

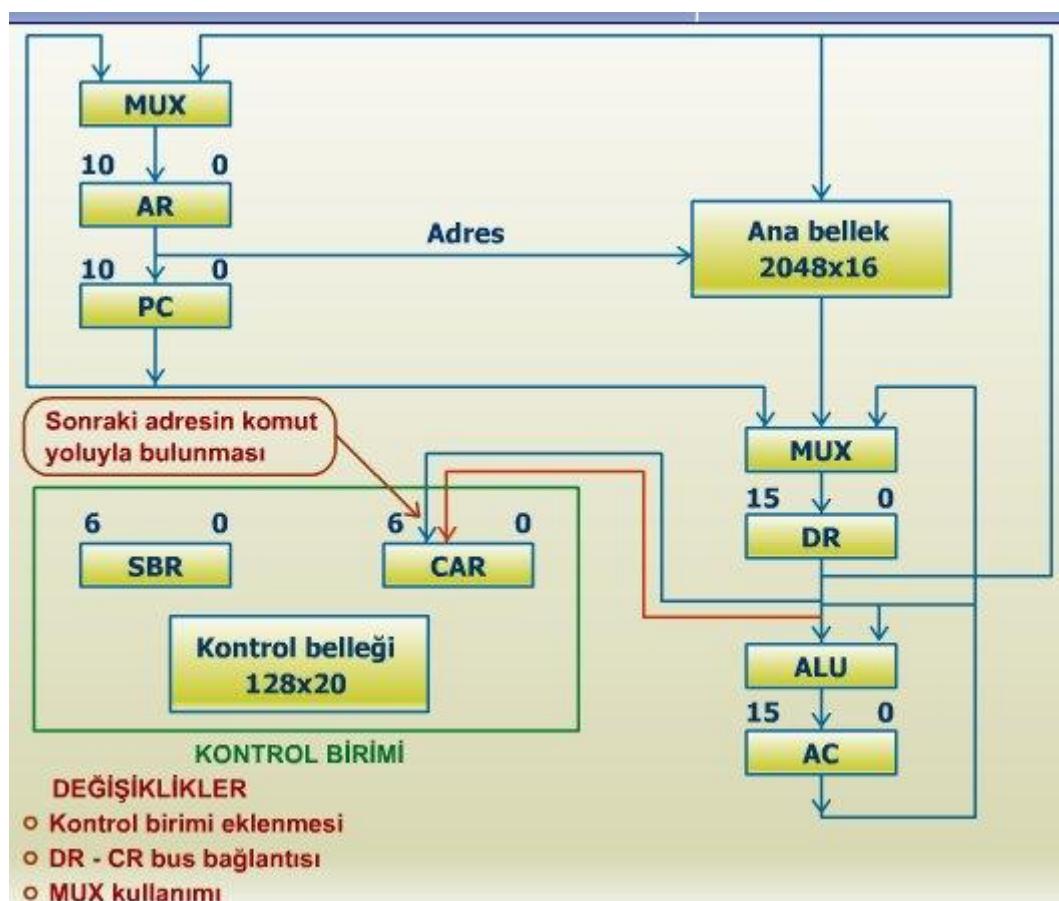
1. CAR'nın artırılması
2. Durum bitlerine (status bit) bağlı olarak koşullu veya koşulsuz dالانma
3. Komutların bitlerinden kontrol belleğine transformasyon
4. Alt rutin çağrıma ve dönüş olanağı



Tekrar

## 7.3 MİKROPROGRAMLA KONTROLDE BİLGİSAYAR YAPISI ve MİKROKOMUTLAR

Aşağıda mikroprogram kontrolü için temel bilgisayar yapısının alması gereken son durum gösterilmiştir.



### 7.3.1 Kontrol Birim Bloğu

Temel bilgisayar yapısına kontrol birimi bloğu eklenmiştir.  
Kontrol belleği, CAR ve SBR sisteme eklenmiştir. Blok diyagramda iki bellek birimi görülmektedir:

**!Açıklamaları görmek için kutuların üzerine tıklayınız.**

<b>Ana Bellek Birimi</b>	<b>komutların ve verinin saklanması için kullanılır</b>
<b>Kontrol Belleği</b>	<b>mikrokomutların saklanması için kullanılır</b>

Tekrar

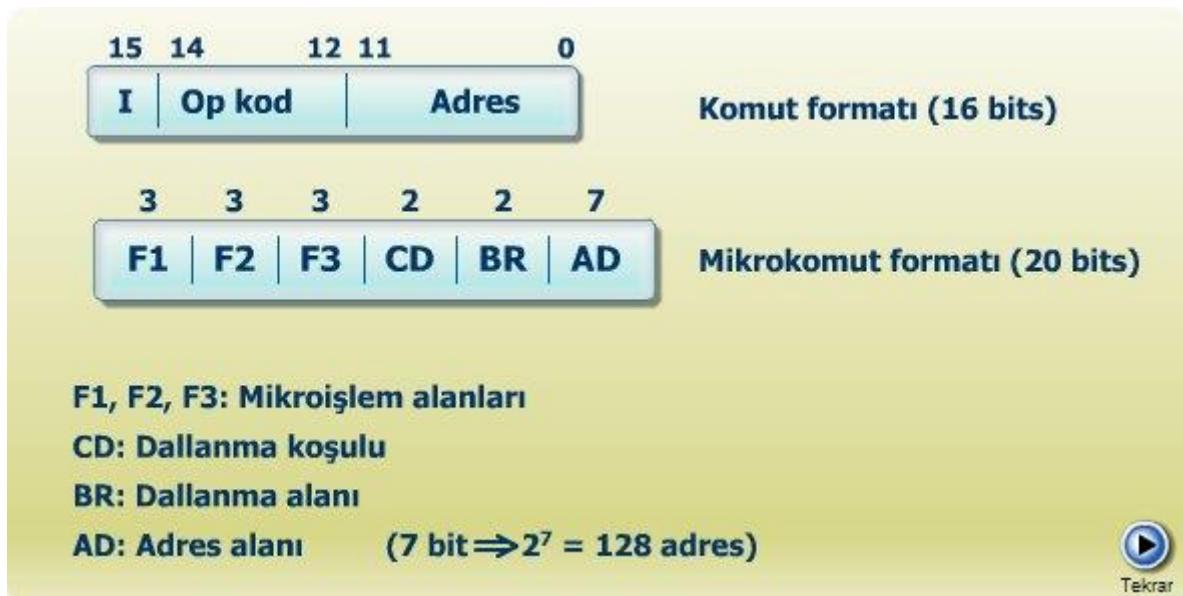
Dört hızlı bellek (AC, DR, AR ve PC) birimi işlemci birimiyle, iki hızlı bellek birimi ise kontrol birimi ile ilişkilidir. İşlemci ile ilgili hızlı bellekler program sayıcı PC, adres belleği AR, veri belleği DR ve akümülatör veya işlemci belleği AC'dir. PC'nin komutların yerini gösterdiği, AR ve DR'nin ise ana belleğe veri transferi için kullanıldığı ve AC'nin işlemciyle ilgili operasyonlarda operand ve sonucu saklamakta kullanıldığı hatırlanmalıdır. Kontrol biriminde ise kontrol adres belleği CAR ve alt rutin belleği SBR bulunmaktadır. CAR kontrol belleğinden (ROM) veri alabilmek için gerekli adresi tutmak, SBR ise alt rutin çağrılmaması sırasında dönüş adresini saklamakta kullanılır.

İşlemci bellekleri arasındaki veri transferi, hat grubu (bus) yerine MUX yapıları ile sağlanmaktadır. Örneğin, DR verileri PC'den, ana bellekten veya AC'den alabilmektedir. AR ise girişlerini PC veya AC'den alabilmektedir.

Kontrol birimi ile işlemci birimi arasında ilişki DR ve CAR bağlantısıyla sağlanmaktadır. Bilgisayar komutuna ilişkin operasyon kodunun 4 biti ile CAR'nın 7 biti arasında transformasyon yoluyla ilişki kurulduğu hatırlanmalıdır. Böylece komutun işlemine (ADD, LDA, etc) ilişkin bilgi kontrol belleğinde saklanmakta olan rutinin adresine dönüştürülmüş olur.

### 7.3.2 Temel Bilgisayarın Komut ve Mikrokomut Formatı

Temel bilgisayarın komut formatı ve buna karşı düşen mikrokomut formatı aşağıda gösterilmiştir.



Tekrar

16-bit komutun 11-15 bitleri arasındaki dört biti işlem (operasyon) kodu için, geri kalan 0-11 bitleri ise adres için ayrılmıştır. Mikrokomut formatı 20 bit olup şekildeki gibi F1, F2,

F3 : mikroişlem alanı (3'er bitten toplam 9 bit), CD: koşul alanı (2 bit), BR: dallanma alanı (2 bit) ve kontrol belleğindeki toplam  $128=2^7$  adrese karşı düşen AD: adres alanından (7 bit) oluşur. Her bir mikrokomut alanı yeni tanımlanacak olan, sembolik mikroprogramlama dilinin sembollerini yoluyla kontrol işaretlerinin depolanması ve üretiminde kullanılacaktır.

### 7.3.3 Üç Bitlik Alanlarının Sembol ve İkili Kodları

F1, F2 ve F3 alanlarının sembol ve ikili kodları son derece basit ve anlaşılabilirdir. 3 bitlik alan nedeniyle her biri 0'dan 7'ye kadar sayar ve tanımlanan semboller kendi kendine açıklayıcıdır. F1 alanının 001, F2=001 ve F3=001 ise ADD, SUB ve XOR operasyonlarını tanımlar. Örneğin ADD 000 olması NOP: operasyon yok (no operation) olarak tanımlanmıştır. F2 ve F3 içinde 000 değeri NOP olarak tanımlanır. F1= için AC<=AC+DR bellek transfer yapılır. Aynı düşünüş yoluyla, F1=011 değeri INCAC veya AC<=AC+1'i tanımlar. Tüm mikrokomutlar aşağıdaki tablodan öğrenilebilir.

F1	Mikroişlem	Sembol	F2	Mikroişlem	Sembol
000	Bos	NOP	000	Bos	NOP
001	AC← AC + DR	ADD	001	AC← AC - DR	SUB
010	AC← 0	CLRAC	010	AC← AC v DR	OR
011	AC← AC+1	INCAC	011	AC← AC ^ DR	AND
100	AC← DR	DRTAC	100	DR← M[AR]	READ
101	AR← DR(0-10)	DRTAR	101	DR← AC	ACTDR
110	AR← PC	PCTAR	110	DR← DR+1	INCDR
111	M[AR]← DR	WRITE	111	DR(0-10)← PC	PCTDR

F3	Mikroişlem	Sembol
000	Bos	NOP
001	AC← AC⊕DR	XOR
010	AC← AC̄	COM
011	AC← shlAC	SHL
100	DR← shrAC	SHR
101	PC← PC+1	INCPC
110	PC← AR	ARTPC
111	rezerv	

### 7.3.4 İki Bitlik Alanlarının Sembol ve İkili Kodları

CD ve BR alanları da aşağıda gösterilmiştir. CD ve BR 2 bit ile tanımlanan değerlerdir. Dört durumu tanımlayabilirler. Örneğin CD=00 değeri U sembolü ile gösterilip koşulsuz dallanma olarak tanımlanır. CD=01, 10 ve 11 değerleri ise I (dolaylı), S ( işaret) ve Z (sıfır) sembollerile gösterilip, bu koşullarda (örneğin I=1 ise) dallanmayı tanımlar. BR değerleri de sırasıyla 00, 01, 10 ve 11 olup, JMP, CALL, RET ve MAP sembollerini tanımlar. Örneğin, eğer JMP (BR=00) ve koşul (I, S, Z den biri)=1 ise CAR<=AD, değilse (koşul=0) CAR<=CAR+1 olarak tanımlanır. CALL ve RET alt rutin çağrıma ile ilgili olup, RET (BR=10) ise CAR<=SBR bellek transferi gerçekleşir.

AD alanı ise sonraki işlemin adresinin tanımı için kullanılır. Mikrokomut "AL"ındıktan sonra (fetch), ne işlem yapılacağı dekode edilip veya öğrenilip, bu işlemin rutinine atlama

(JMP) gerekir. Böylece sürekli olarak çalışan ve kontrol işaretleri üreten bir program (mikroprogram) oluşturulmuş olunur.

CD	Koşul	Sembol	Açılıma
00	daima=1	U	Koşulsuz
01	DR(15)	I	Dolaylı biti(komut)
10	AC(15)	S	İşaret biti (AC)
11	AC=0	Z	Sıfır (AC)

BR	Sembol	Fonksiyon
00	JMP	CAR $\leftarrow$ AD if koşul=1 CAR $\leftarrow$ CAR+1 if koşul=0
01	CALL	CAR $\leftarrow$ AD, SBR $\leftarrow$ CAR+1 if koşul=1 CAR $\leftarrow$ CAR+1 if koşul=0
10	RET	CAR $\leftarrow$ SBR (alt rutinden dönüş)
11	MAP	CAR(2-5) $\leftarrow$ DR(11-14), CAR(0,1,6 $\leftarrow$ 0)

ADD komutunu örnek olarak kullanalım ve mikrokomut alanlarını belirleyelim: ADD komutunun opkodu=0000'dır ve bir efektif adres alanı mevcuttur (EA alanı). Tanımı hatırlanırsa AC<=AC+M[EA] işlemini yapar. Bununla ilgili mikrokomutun alanlarını gösterelim: ADD ile ilgili F1=001, F2=F3=000 mikrooperasyon alanları tanımlanır. CD=00 (U) ve BR=00 (JMP) olarak tanımlanır. AD ise sonraki mikrokomutun alınması için "AL" (fetch) fazının adresine aktarılır. Burada işlem yapıldıktan sonra gidilecek adresin tanımının da yapılması gereğine dikkat edilmelidir.

Sonuç olarak, 0 ve 1'lerden oluşturulan mikrokomut, bir sembol ile isimlendirilerek mikroprogramlama dili oluştururlar ve kontrol belleğine saklanır. Mikroprogram, kullanıcının yazdığı program koştuğu müddetçe çalışarak, komutlara uygun kontrol işaretlerini "yazılım" yoluyla üretir.

## 7.4 MİKROPROGRAMLAMA

Mikroprogramlama, kontrol işaretlerini oluşturan ikili sayıların (0, 1) mikrokomutlar yazılarak oluşturulmasıdır. Bu sembolik mikroprogram ikili kontrol işaretlerine mikroassemblerlere anlamında dönüştürülür. Bu daha önce tanıttığımız assemblerlere anlamında olmaktadır.

Mikroprogramın her satırı 5 alanlı mikrokomutlardan oluşur: etiket, mikrooperasyon, CD, BR ve AD alanlarıdır. Her bir alan aşağıdaki bilgileri içerir.

Etiket	Mikrooperasyon	CD	BR	AD
FETCH:	PCTAR	U	JMP	NEXT



1. Etiket alanı, boş olabilir veya sembolik bir adres içerebilir. Kolon (:) ile biter.

2. Mikrooperasyon alanı, virgül ile ayrılmış bir, iki veya üç sembolden oluşur. Her biri F1, F2, F3 alanlarına karşı düşer ve 9 bit (3 x 3 bit) ile tanımlanır.
3. CD alanı U, I, S veya Z harflerinden birini içerir. Mikrokomuttaki koşulu tanımlar.
4. BR alanı daha önce tanımlanan JMP, CALL, RET ve MAP sembollerini içerir.
5. AD alanı aşağıdaki 3 yoldan birisi olabilecek adresi tanımlar:
  - a. Sembolik bir adres (etiket)
  - b. NEXT sembolü: sonraki satırı tanımlar
  - c. BR alanı RET veya MAP olarak tanımlanmışsa, AD alanı boş bırakılır.

### **7.4.1 AI (Fetch) Rutini**

Mikroprogram rutininin ilk adres veya orijini ORG ile tanımlanır. ORG 64 rutinin 64'üncü adresinden başladığını gösterir.

Kontrol belleği 128 kelime olup, her kelime 20 bit içerir. Rutinler bu bellekte ikili bilgi olarak saklanırlar. Mikrokomutlar için de "al" (fetch) rutini tanımlamak gereklidir. Mikroprogram kontrollü bilgisayarın "al" rutini aşağıdaki gibi tanımlanır:

**AR<=PC**  
**DR<=M[AR], PC<=PC+1**  
**AR<=DR(0-10), CAR(2-5)<=DR(11-14), CAR(0, 1, 6)<=0**


  
Tekrar

Komutun adresi PC'den AR'ye transfer edilir. Sonra komut ana bellekten DR'ye okunur ve PC bir arttırılır. Operasyon kodu DR'den CAR'ye dönüştürülür (mapping), tanımlanmış olan 16 komut için, kontrol belleğinde saklanan 16 rutinden (0-63 adreslerinde yer almaktır) birine yönlendirilir.

Yukarıdaki işlemleri yapabilecek "AI" veya fetch rutini için mikrokomutlar 64'den başlayıp, üç satır sürer. Aşağıda FETCH rutini mikroprogram sembollerini cinsinden tanımlanmaktadır:

FETCH:	ORG 64	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT	
	DRTAR	U	MAP		


  
Tekrar

Böylece ilk mikroprogramımızı yazmış olduk. Bu sembollerin daha önce açıkladığımız ikili karşılıkları kontrol belleğine (ROM) kaydedilir.

Bilindiği gibi sembolik mikroprogram kullanıcıya kolaylık sağlayarak, kontrol belleğine saklanmış olan ikili kodların okunmasını ve anlaşılmasını sağlar. Bu durumun anlaşılması için FETCH rutininin kontrol belleğinde 64 adresinden başlayarak ikili sayılarla oluşan halini görelim:

Adres									
	Onaltılı	İkili	F1	F2	F3	CD	BR	AD	
<b>FETCH:</b>	64	1000000	110	000	000	00	00	1000001	
	65	1000001	000	100	101	00	00	1000010	
	66	1000010	101	000	000	00	11	0000000	



Adres kısmı hem onaltılı hem de ikili olarak tanımlanmıştır. Mikrokomutun her bir alanına karşı düşen alanların ikili kodları tablodan bakılarak okunabilir. Bu hatırlarda tutması zor bir iştir. AD alanı bir sonraki adresi göstermektedir. (66)H nolu satırda ait AD adresi değeri 0 olup, ADD komutuna ilişkin mikroprogram rutininin ilk satırına gider.

Yukarda FETCH rutininin üç farklı gösterimini tanımladık. İlk gösterilim hızlı bellek transfer dili cinsinden yazılmıştır. İkincisi ise yeni tanımlanan mikroprogramlama dili sembollerini kullanarak yazılan mikrokomut satırlarıdır. Son olarak ise, ikili kodlar cinsinden aynı rutin gösterilmiştir. İlk iki gösterilimin okuma ve anlaşılabilirliğinin daha fazla olduğu çok açıklıktır.

### 7.4.2 INDRCT Rutini (Dolaylı Adres Elde Etme Rutini)

Yazacağımız ikinci rutin INDRCT veya dolaylı adres elde etmeye ilişkin rutindir. Hatırlanırsa I=1 olması durumunda dolaylı adres kullanılmaktaydı. INDRCT rutini:

<b>INDRCT:</b>	READ	U	JMP	NEXT
	DRTAR	U	RET	



Temel olarak, INDRCT dolaylı adrese ulaşmak için bir daha AR<=DR(0-10) işleminin yapılmasını sağlar.

### 7.4.3 ADD Rutini

Üçüncü rutin olarak ise ADD rutinini yazalım:

<b>ADD:</b>	ORG 0			
	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH



ADD rutini 0 adresinden başlamaktadır (ORG 0). ADD etiketi rutinin başladığını gösterir. NOP hiçbir operasyon yapılmadığını, I ise dolaylı adres olup olmadığını gösterir. Eğer I=1 ise INDRCT rutini çağrırlıır ve dolaylı adres tanımından efektif adres bulunur. Sonraki

satırda, READ mikrokomutu ile kontrol belleğinden okuma işlemi yapılır. "U JMP NEXT" alanları koşulsuz olarak bir sonraki satıra geçilmesi gerektiğini tanımlar. Son satırda ise ADD mikrokomutu tanımlanmaktadır. AC içeriği ile önceki satırlardan elde edilen operand toplanır. "U JMP FETCH" ise yeni bir mikrokomutun alınabilmesi için FETCH rutinini bir sonraki adres olarak tanımlar.

ADD komutuna ilişkin kontrol işaretlerinin oluşturulması için, kontrol belleğinde saklanmış olan rutinler FETCH, INDRCT ve ADD kullanılır. FETCH ve INDRCT bütün komutlar için ortaktır. ADD komutuna ilişkin kontrol işaretleri ADD rutinini oluşturan mikro komutlar yardımıyla elde edilir.

#### 7.4.4 Dallanma (Branch) Rutini

Sembolik mikroprogramlama için diğer bir örnek olarak dallanma (BRANCH) rutinini yazalım ve inceleyelim. BRANCH rutini ROM'un 4. satırından başlasın.

ORG 4			
BRANCH:	NOP	S	JMP      OVER
	NOP	U	JMP      FETCH
OVER:	NOP	I	CALL     INDRCT
	ARTPC	U	JMP      FETCH

Tekrar

BRANCH rutini S=1 (S işaret biti koşulu) olup olmamasına bağlı olarak OVER rutinine atlar. OVER rutini I=1 koşulu sağlanırsa INDRCT rutinini çağrıır. Burada saklı olan adres ARTPC mikrokomutu yoluyla program sayıcıya (PC) aktarılır ve programın yeni değerden devam etmesi için gerekli kontrol işaretlerini üretir.

BRANCH rutini örneğindeki bir diğer önemli gözlem ise bir rutinden diğer rutinleri çağrımanın mümkün olduğunu. Böylece kontrol işaretleri, yazılan sembolik programın uygun parçalarının bir araya gelmesiyle oluşturulur.

Bu bölümü son olarak açıklamasız olarak vereceğimiz STORE rutini ile bitirelim:

ORG 8			
STORE:	NOP	I	CALL     INDRCT
	ACTDR	U	JMP      NEXT
	WRITE	U	JMP      FETCH

Tekrar

**! NOT:** Kullanıcı tarafından yazılan programın koşturulabilmesi için mikroprogram rutinlerinin durmaksızın (FETCH, INDRCT, ADD, BRANCH komut tipine göre) koşturulduğuna dikkat ediniz.

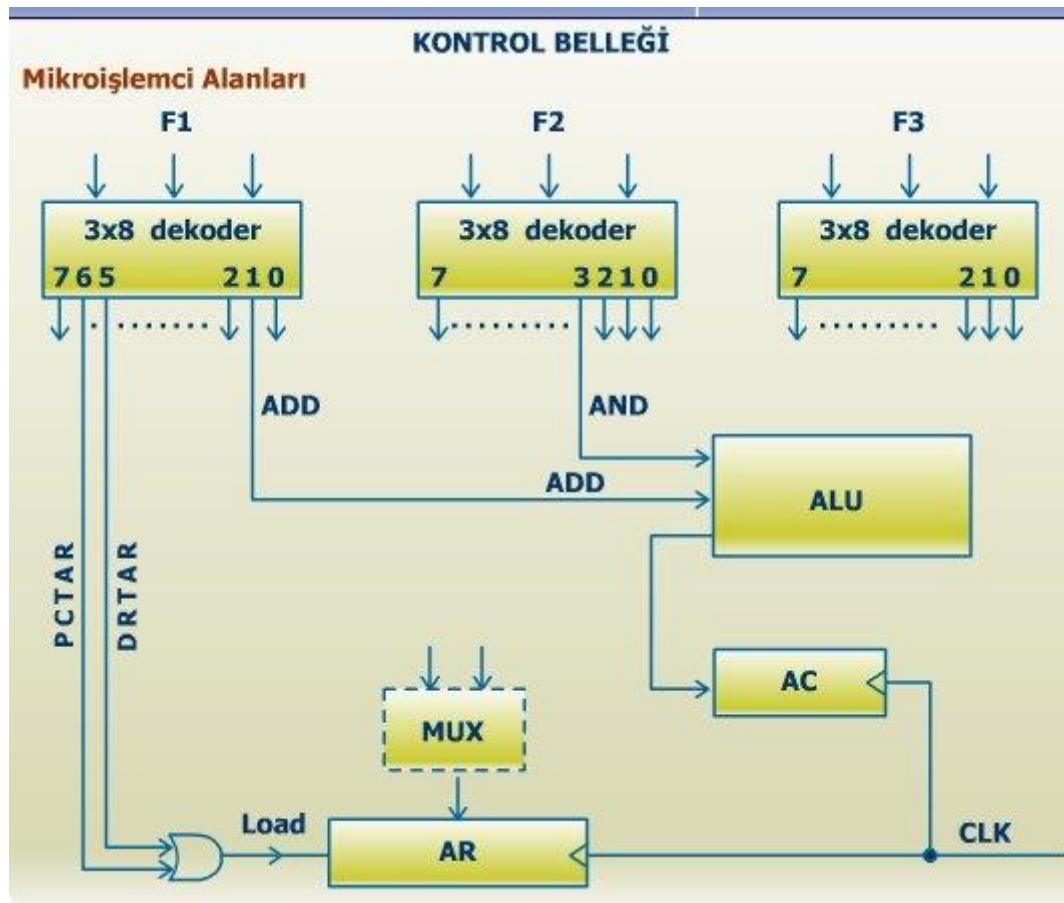
### 7.5 KONTROL BİRİMİ TASARIM DETAYLARI

Mikroprogramlama ile kontrol işlemini özetleyelim: Çeşitli komut alanlarında üretilen kontrol işaretleri, mikroişlemlerin başlaması için kontrol bitlerini üretir. Bu bitler,

mikrokomutların alanlarında birbirinden bağımsız işlevler üretmektedir (F1, F2, F3, CD, BR, AD).

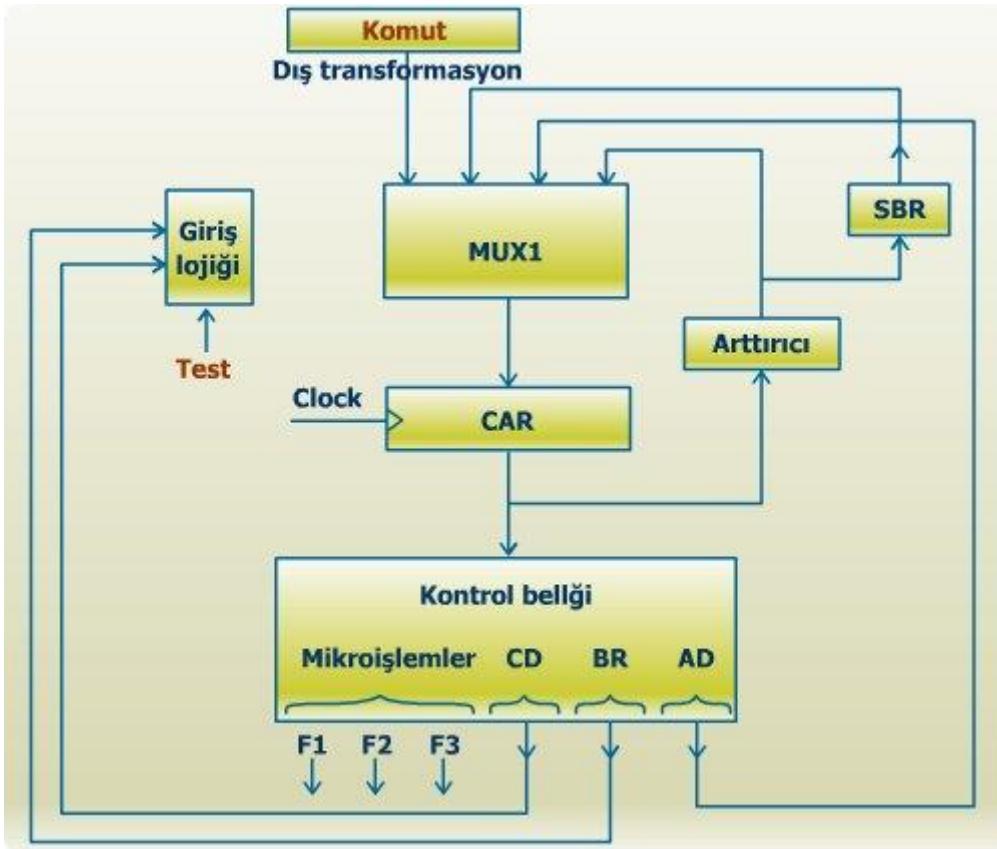
Bu kontrol bitlerinin sayısını azaltabilmek için dekoderler kullanılabilir. Böylece kontrol belleğinden elde edilecek kontrol bitleri için kullanılması gereken ek donanım azaltılmış olunur. F1, F2 ve F3 alanlarının çıkışında kullanılacak her bir  $3 \times 8$  dekoder toplam  $3 \times (8 \text{ çıkış}) = 24$  çıkış işaretini üretebilir. Bu çıkışlar ALU, AC, PC, AR ve DR bellekleri için kontrol işaretleri olarak kullanılırlar.

Aşağıdaki şekilde de daha önce genel olarak çizilen mikrokomut dizileyici ve kontrol belleğinin temel gösterimi göz önüne alınmıştır. Kontrol belleği alanları ve dizinin dört seçimi  $4 \times 1$  MUX bağlantısı ile CAR'ye aktarılması dizinin işleyişinde temel noktadır.



### 7.5.1 Mikroişlem Alanlarının Dekoder ile Bağlantıları

Aşağıdaki şekilde mikroişlem alanlarının dekoder ile bağlantıları gösterilmiştir. Her bir alan ALU işlemleri ve bellek transferlerini kontrol işaretleri olarak dönüştürür; bu işaretler bilgisayarda kullanılmaktadır.



Mikroprogramlama ile kontrolde artırmacı devrenin FF'lerden oluşmuş bir ikili sayıcı olmadığına dikkat etmelidir. Onun yerine, kaskad bağlanmış iki yarımtoplayıcıdan (half adder) oluşan kombinasyonel bir artırmacı kullanılmıştır.

Sonuç olarak, mikroprogramlama ile kontrolün temel kavramı kontrol belleğinde (ROM) sürekli olarak saklanan rutinlerin adreslerinin elde edilmesi ve her bir satırda oluşan mikrokomut dizilerinin takip edilerek arkaya arkaya kontrol işaretlerinin üretilmesi olarak özetlenebilir.

## Bölüm Özeti

- Bu bölümde, temel bir bilgisayarda mikroprogramlama ile kontrol tanıtılmıştır. Temel bilgisayarda tanımlanan tam (complete) fakat hızlı olmayan komut setinin kontrolü için birbiri ardı sıra saklanan mikroprogram komutlarından yararlanılmış, bununla ilgili olarak yalnızca okunabilen bellek (ROM) ve adres dizileri ile kontrol kavramları anlatılmıştır.

## BÖLÜM - 8

# MERKEZİ İŞLEM BİRİMİ (CPU)

Bu bölümde aşağıdaki konular incelenecaktır:

- Giriş
- Genel hızlı bellek organizasyonu
- Son giren ilk çıkar (SGİÇ) (stack) organizasyonu

- Komut formatları
- Adres modları

## Bölüm Hedefi

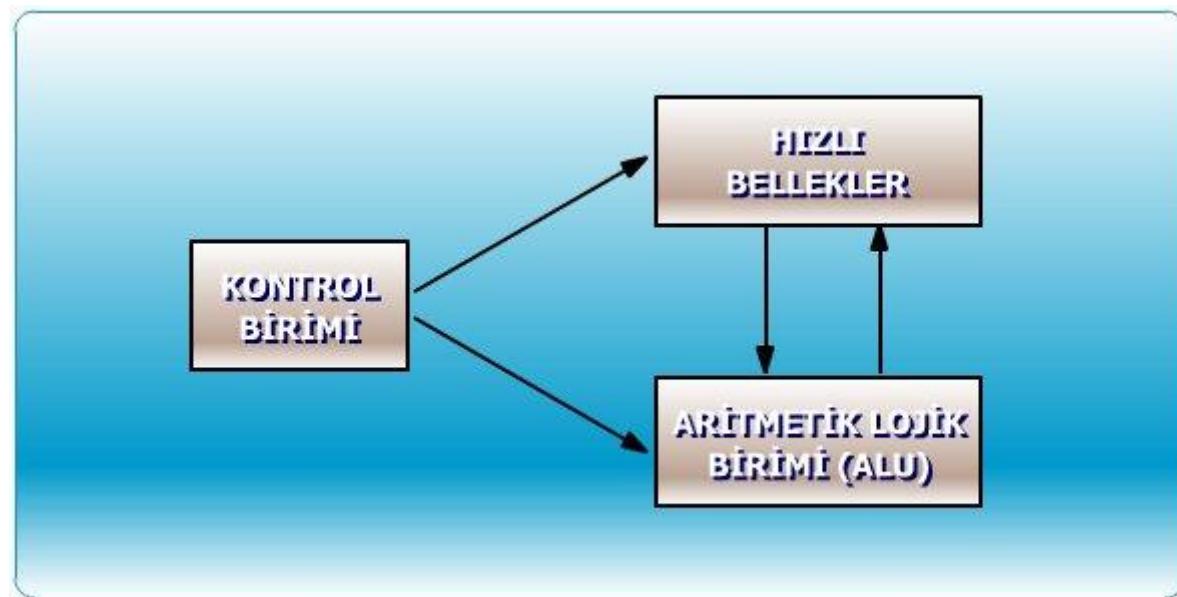
Önümüzdeki iki haftada, merkezi işlem birimi detayları ve çeşitleri ile tanıtılacaktır. Temel bilgisayarda kullanılan tek akümlatör yapısından, genel amaçlı hızlı bellek yapısına geçilecektir. Bilgisayar yapıları, çeşitleri ve farkları açıklanacaktır.

Ayrıca bu bölümde, çeşitli komut formatları ve adres modları detayları ile açıklanacaktır. Daha sonra veri transferi ve program kontrolü ile ilgili komutlar ve özelliklerini özetlenecektir. Son olarak, indirgenmiş komut setli bilgisayarlar (RISC) tanıtılmaktır.

## 8.1 GİRİŞ

Merkezi işlem birimi (CPU) veri işleme birimidir. Aşağıda CPU bileşenleri gösterilmiştir. Temel olarak, veri işleyen ALU ve veri saklayan hızlı bellekler arasında veri alışverişi olur. Kontrol birimi ise işlemlerin yapılmasını sağlayan kontrol işaretlerini üretir.

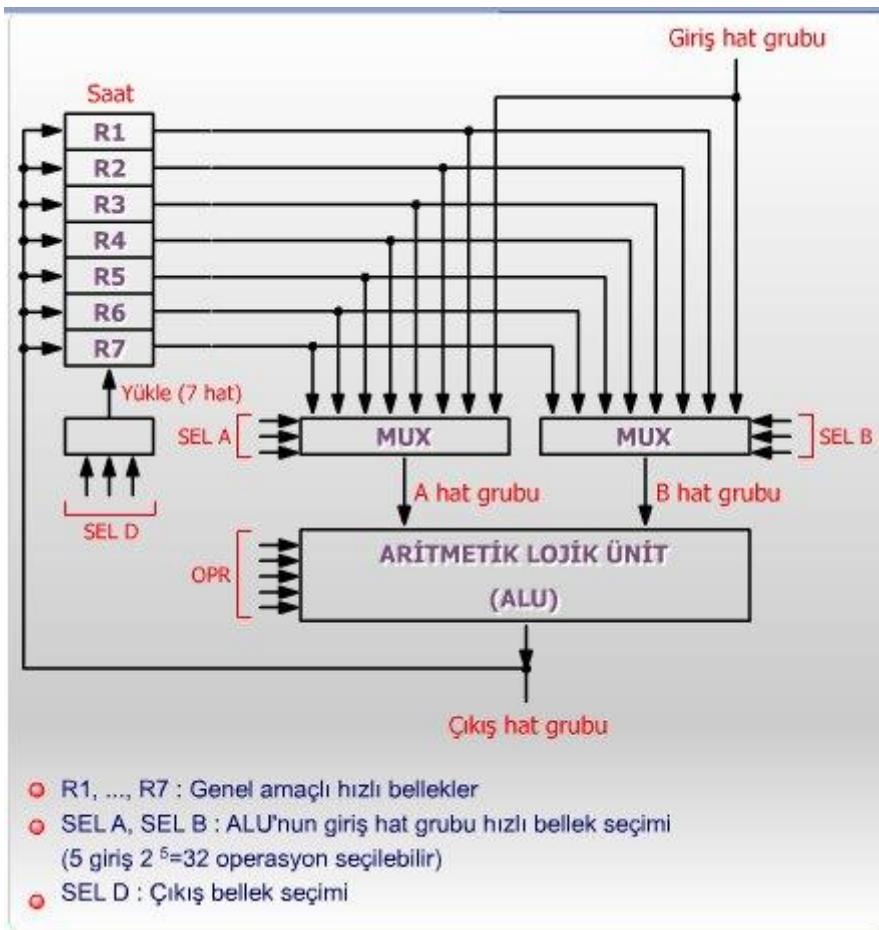
Bu tanıma göre çeşitli CPU konfigürasyonları oluşturulabilecektir. Bunlardan bir tanesi tek akümlatörlü (AC) temel bilgisayar yapısıdır.



CPU organizasyonu

## 8.2 GENEL HIZLI BELLEK ORGANİZASYONU

Temel bilgisayarın tek akümlatör organizasyonunu genelleştirip, tüm belleklere erişebilen ve hepsini işlemci belleği (AC) gibi kullanabilen yeni organizasyonu tanıtalım. Aşağıda genel hızlı bellek organizasyonunun blok diyagramı gösterilmiştir.



Genel hızlı bellek organizasyonunun blok diyagramı

Bu organizasyonda 7 hızlı bellek (R1, R2, ..., R7) kullanılır ve her biri ALU'ya 3x8 MUX'lar yoluyla bağlıdır. Seçme hatları olan SELA, SELB kullanılarak A ve B hat grubuna uygulanacak belleğin seçilmesini; A ve B hat grubunun ALU'ya girişine uygulanmasını sağlar. ALU işlemi sonucunda oluşan değer ise SELD seçme hattıyla istenilen belleğe saklanır. Örneğin,  $R5 \leq R2 + R3$  işlemi SELA'nın R2 girişini, SELB'nin R3 girişini seçmesiyle yapılır. Toplama işlemi sonucu ise SELD yardımıyla seçilen R5 belleğine saklanır. ALU operasyonları ise OPR seçme hatları ile yapılır. 5 hattan oluşan OPR toplam 32 operasyonunun seçimini mümkün kılar. Bağımsız giriş 8'inci giriş olarak, girişteki MUX'lara uygulanır. Hızlı bellek elemanları saat darbeleriyle çalışır.

### 8.2.1 Kontrol Kelimesi ve Ri Bellek Seçimi

Tipik olarak bu yapıdaki bir bilgisayarın kontrol kelimesinde SELA, SELB, SELD ve OPR alanları bulunmalıdır. Aşağıda kontrol kelimesi ve  $R i=1, 2, \dots, 7$  belleklerinin nasıl seçildiği gösterilmiştir.

<b>SEL A</b>	<b>SEL B</b>	<b>SEL D</b>	<b>OPR</b>
<b>3</b>	<b>3</b>	<b>3</b>	<b>5</b>
(bit sayısı)			
<i><b>İkili kod</b></i>	<i><b>SEL A</b></i>	<i><b>SEL B</b></i>	<i><b>SEL D</b></i>
<b>000</b>	<b>giriş</b>	<b>giriş</b>	-
<b>001</b>	<b>R1</b>	<b>R1</b>	<b>R1</b>
<b>010</b>	<b>R2</b>	<b>R2</b>	<b>R2</b>
<b>011</b>	<b>R3</b>	<b>R3</b>	<b>R3</b>
...	...	...	...
<b>111</b>	<b>R7</b>	<b>R7</b>	<b>R7</b>

Kontrol kelimesi ve Ri bellek seçimi

Hızlı bellek seçiminin basit olduğu ve ikili kod değerine bağlı olan belleğin seçildiği gözlenmelidir.

### 8.2.2 OPR Yoluyla Seçilen ALU İşlemleri

ALU'nun gerçekleştirildiği operasyonların seçimi ise OPR değerleriyle olmaktadır. Aşağıda OPR yoluyla seçilen ALU işlemlerinden bazıları gösterilmiştir. Bu organizasyondaki operasyonların sayısı ve çeşitliliği artmıştır. 32 adet işlemin yapılabilmesi için 5 bit ayrılmıştır. ADD işlemi yanında SUB işlemi, AND işlemi yanında ise OR işleminin bulunduğuuna dikkat edilmelidir. Ayrıca, tek operandlı işlemler, örneğin komplement, A girişi yoluyla yapılmaktadır.

<i>OPR kodu</i>	<i>Sembol</i>	<i>Açıklama</i>
00000	TSFA	A girişini transfer et
00001	INCA	A girişini artırır
00010	ADD	A + B
00101	SUB	A - B
00110	DEC	A girişini azaltır
01000	AND	A AND B
01010	OR	A OR B
01100	XOR	A XOR B
01110	COMA	Komplement A giriş
10000	SHRA	A girişini sağa kaydırır
11000	SHLA	A girişini sola kaydırır

#### OPR yoluyla seçilen bazı ALU işlemleri

Sonuçta, genel hızlı bellek organizasyonunda daha esnek ve çok sayıda işlem yapılabilmektedir. Temel bilgisayarın "tam komut seti" (tam fakat verimli olmayan) bu organizasyonda geliştirilmiş ve daha verimli hale getirilmiştir. Bu, assemblers programlamada kullanıcıya daha kolaylık sağlar. Bu tasarımda assemblers programlama örnekleri verilmemiş, programlama da alıştırma olarak öğrenciye bırakılmıştır.

### 8.2.3 XOR Mikroişlemi

Burada verilen tasarımla ilgili olarak bir mikroişlem örneği verilecek ve kontrol kelimesi oluşturulacaktır. XOR "mikroişlemi" aşağıdaki bellek transfer ifadesi ile gerçekleşsin:

R4 <= R3 XOR R6

R3 ve R6 hızlı belleklerinin ikili içeriklerine XOR lojik işlemi uygulanır ve sonuç R4'e saklanır. Buna ilişkin "komut"

XOR R3, R6, R4

olarak yazılabilir. İkili "kontrol kelimesi" ise aşağıdaki gibidir.

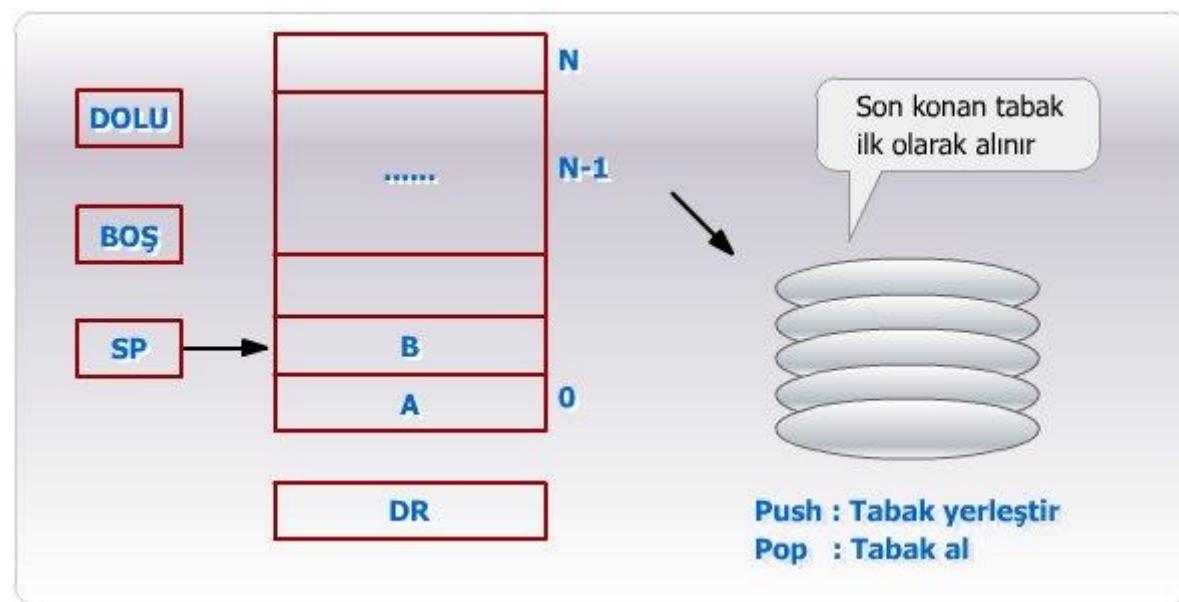
<b>ALAN :</b>	<b>SEL A</b>	<b>SEL B</b>	<b>SEL D</b>	<b>OPR</b>
<b>SEMBOL :</b>	<b>R3</b>	<b>R6</b>	<b>R4</b>	<b>XOR</b>
<b>KONTROL KELİMESİ :</b>	<b>011</b>	<b>110</b>	<b>100</b>	<b>01100</b>

XOR işleminin bir özelliğinin de bellekleri sıfırlamak olduğu hatırlanmalıdır:

R1 XOR R1 =0

### 8.3 YIĞIN (STACK) ORGANİZASYONU

Yığın (stack) bir bellek olup, veri giriş-cıkışı belli bir sırayla olur. Son saklanan veri ilk olarak alınabilir. Bu kavramı, üst üste konan tabaklar örneği ile açıklayabiliriz: Tabakları üst üste yerleştirirken daima yenisini en üste koymak gereklidir. Ama alırken daima en (üstteki) sonuncu tabak en önce alınır. Aşağıda bir yığın yapısı gösterilmiştir.



Bir yığın yapısı

$N$  elemanlı bir yığında iki bayrak (bit)[dolu (full) ve boş (empty)], yığın gösterici (stack pointer-SP) görülmektedir. Bayraklar yığının dolu ve boş olup olmamadığını gösterir. SP ise yeni gelen elemanın adresini tanımlar. Yığın operasyonlarını tanımlayıp, bu elemanların kullanımlarını gösterelim; yığın operasyonları "Push" ve "Pop" olarak tanımlanır.

### 8.3.1 Push Operasyonu

Push operasyonu yanda verilen hızlı bellek transferleri ile tanımlanır:

$SP \Leftarrow SP+1$

$M[SP] \Leftarrow DR$

If ( $SP=0$ ) then ( $DOLU \Leftarrow 1$ )

$BOŞ \Leftarrow 0$

SP 1 arttırılır, DR içeriği SP'nin gösterdiği adrese saklanır. Yiğinin dolu olup olmadığı "if" ifadesi ile kontrol edilip, boş olmadığı BOŞ bayrağı 0 yapılarak belirtilir.

$DR \Leftarrow M[SP]$

$SP \Leftarrow SP-1$

If ( $SP=0$ ) then ( $BOŞ \Leftarrow 1$ )

$DOLU \Leftarrow 0$

### 8.3.2 Pop Operasyonu

Pop operasyonu yanda verilen hızlı bellek transferleri ile tanımlanır:

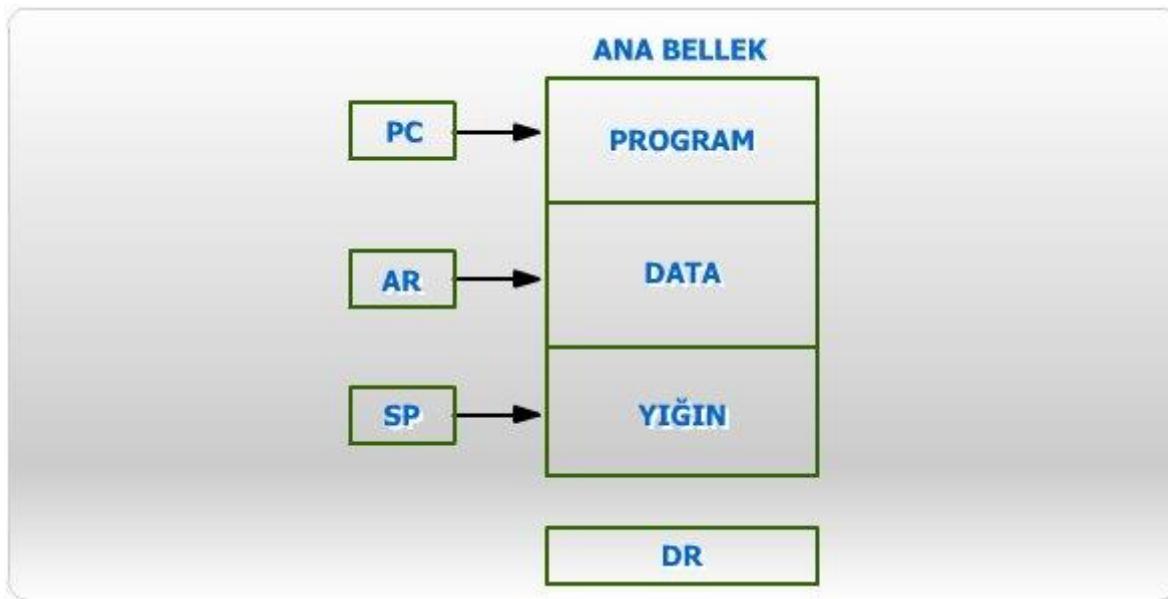
DR'nin içeriği SP'nin gösterdiği adrese saklanır ve SP 1 azaltılır. Yiğinin boş olup olmadığı kontrol edilip, dolu olmadığı DOLU bayrağı 0 yapılarak belirtilir.

### 8.3.3 Yiğinin Yapısı

Pek çok bilgisayar yiğinin boş veya dolu olduğunun kontrol edilmesi kullanıcıya bırakıldığından, bununla ilgili yazılım daha önce belirtildiği gibi yazılarak sınırların belirlenmesi gereklidir.

Yığın tek başına kullanıldığı gibi ana bellekle birlikte bir CPU'ya eklenmiş olarak da kullanılır. CPU'ya eklenmiş yığın yapısı için ana bellek üç segmente ayrılır: Program, veri ve yığın. Aşağıda bilgisayarın ana belleğinin üç segmenti program, veri ve yığın blok diyagramla gösterilmiştir. PC program segmentinde adresleri göstermek için, AR veri segmentinde operandların yerini göstermek için kullanılır. SP ise yığındaki adresleri tutan yerdür. Ana belleğin yanında DR veri alışverişine yardım eder.

SP yiğinin en üst adresini gösterir. Yığın azalan adreslerle büyür. Örneğin, SP 3000 adresini gösterirse yeni gelen eleman 2999 adresinde saklanır.



Bilgisayarın ana belleğinin üç segmenti: program, veri ve yiğin blok diyagramları

### 8.3.4 Yiğine Eleman Giriş Çıkışı

Yiğine elemanların giriş çıkışları PUSH ve POP komutları kullanılarak, DR yoluyla yapılır.

Push operasyonu aşağıda verildiği gibi tanımlanır.

$$\begin{aligned} \text{SP} &\leq \text{SP}-1 \\ \text{M}[\text{SP}] &\leq \text{DR} \end{aligned}$$

Pop operasyonunun tanımı ise,

$$\begin{aligned} \text{DR} &\leq \text{M}[\text{SP}] \\ \text{SP} &\leq \text{SP}+1 \end{aligned}$$

şeklindedir.

### 8.3.5 Aritmetik Operasyonlarda Yiğin Organizasyonu

Yiğin organizasyonu, aritmetik ifadelerin sonuçlarının elde edilmesinde kullanılmaktadır. Aritmetik operasyonlar üç notasyonla tanımlanabilir:



**A+B : Takısız notasyonu (infix notation)**

**+AB : Öntakılı notasyon (prefix-polish notation)**

**AB+ : Sontakılı notasyon (suffix-reverse polish notation)**

Notasyonlar arasında dönüşüm yapmak mümkündür. Örnek olarak takısız ve sontakılı notasyon dönüşümünü ele alıp gösterelim. Takısız ifade

(A-B)\*[C\*(D+E)-F]

sontakılı ifade ise

AB-DE+C\*F-\*

olarak elde edilir.

### 8.3.6 Sontakılı İfadelerde Yığın Organizasyonu

Yığın sontakılı ifadelerin değerlerinin bulunmasında kullanılabilir. Sontakılı ifadede, operandlar operasyon görülene kadar yığına saklanır, operasyon görüldüğünde son iki operand ile işlem yapılarak tekrar yığına aktarılır. Kuralları aşağıdaki gibi belirtebiliriz:

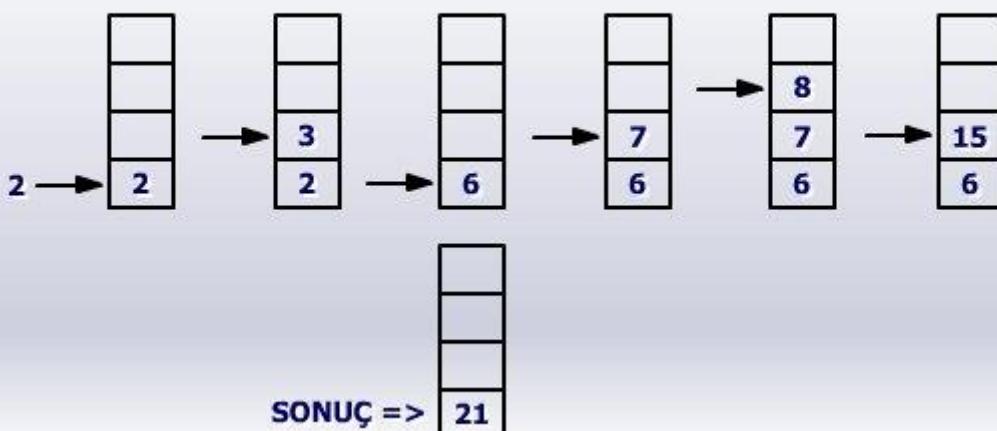
1. En son iki operand görülen operasyon için kullanılır (iki kere pop)
2. Sonuç yığına aktarılır (push)

Şimdi bir örnekle bu operasyonları gösterelim:

**(2\*3)+(7+8) (takısız ifade)**

**23\*78++ (sontakılı ifade)**

Aşağıda  $(2*3)+(7+8)$  ifadesinin yığın ile sonucunun bulunması gösterilmiştir:



## 8.4 KOMUT FORMATLARI

Genel olarak komutların formatları aşağıdaki alanlardan oluşur:

! Aşağıdaki başlıklara tıklayarak açıklamaları görebilirsiniz.

**Operasyon kodu** Operasyonun cinsini tanımlar.

**Adres alanı** Bir ana bellek adresi ve hızlı bellek gösterir.

**Mod alanı** Operandın efektif adresinin elde edilme yolunu gösterir.

Operasyon kodu, bilgisayar komutunun toplama, çıkarma,..vb. olduğunu tanımlayan bir grup bittir. Operasyonun cinsi belirlendikten sonra "mod alanı" yardımıyla efektif adresin nasıl elde edileceği belirlenir. Temel bilgisayar örneğinde, doğrudan ve dolaylı modlar I bitini 0 veya 1 olması yoluyla seçilebilir. Son olarak da, adres alanının adres bilgisini tanımlamasından yola çıkılarak adres bulunur. I=0 efektif adresi, adres alanındaki ikili kod olur (doğrudan), I=1 ise efektif adres ikili adres kodunun gösterdiği yerdeki içerik olmaktadır (dolaylı).

### 8.4.1 Bilgisayarların Kategorileri

Bilgisayarların çoğu aşağıdaki kategorilerden birine düşer:

- Tek akümülatör (AC) organizasyonu
- Genel hızlı bellek organizasyonu
- Yiğin organizasyonu



Tek akümülatör yapısında kullanılan işlemciye ait hızlı bellek AC yerine, genel hızlı bellek yapısında bir grup bellek (verdiğimizörnekte, R1, R2, R3,..., R7) aynı görev için kullanılabilmektedir. Yiğin yapısı kendine özgü ana bellek erişimi ve komutlarıyla üçüncü tip organizasyon türüdür. Örnek olarak, aritmetik toplama ve veri transferi operasyonlarının bilgisayar tiplerinde nasıl gerçekleştiğini görelim: tek AC yapısında ADD X olup, X adresi (ana bellek veya hızlı bellek) operandın yerini gösterir. Veri transferi, bilindiği gibi LDA X (AC'yi yükle) ve STA X (AC'yi sakla) komutlarıyla gerçekleşir. Genel hızlı bellekli organizasyonda,

ADD R1, R2, R3

operasyonu üç hızlı bellek kullanır.  $R1 \leq R2+R3$  olarak tanımlanır. Üç bellekli gösterilim, iki bellekli olarak da gösterilebilir:

ADD R1, R2

Ifadesinde R1 belleği hem operandı, hem de sonucu tutar:  $R1 \leq R1 + R2$ . İki bellekli gösterilimde veri transferi ise MOV komutu yoluyla sağlanır:

MOV R1, R2

olup,  $R1 \leq R2$  olarak sağlanır. Genel hızlı bellek organizasyonunda X veya R1, R2 kullanılabilir: operand ana bellek adresi ve hızlı bellek olabilir.

ADD X, R1

MOV R2, X

Son olarak, yiğin yapısında SP'nin değerinin düzenli olarak yenilenmesi (yiğinin en üstü-top of stack: TOS) özelliğinden yararlanarak PUSH ve POP komutları kullanılır:

PUSH X

Yiğin yapısında operasyon tipi komutların adres bilgisine gerek yoktur. Yiğine saklanmış iki operand ile işlem yapılır:

ADD

Önceden konmuş iki operand POP yapılır, toplama işlemi gerçekleştirilir ve sonuç yine yiğine saklanır.

## 8.4.2 Komut Format Çeşitleri

Komut formatları olarak:

### ÖRNEK

- $Y=(A+B)*(C+D)$  matematiksel ifadesini ADD SUB, MUL ve DIV kullanarak komut format çeşitlerine göre gerçekleyiniz.

MOV ve LOAD, STORE komutları da gerekli yerlerde veri transferi için kullanılacaktır.  
A, B, C, D ve Y ana bellek adresleridir.

İPUCU

ÇÖZÜM

## Örnek Çözümü

Üç adresli komutlar:

```
ADD R1, A, B  R1<=M[A] + M[B]
ADD R2, C, D  R2<=M[C] + M[D]
MUL Y, R1, R2  M[Y]<=R1 * R2
```

Hızlı bellek transferleriyle işlemler açıklanmıştır. İşlemcinin R1 ve R2 hızlı belleklerini kullandığını ve M[Y] ise Y adresiyle tanımlanan ana bellek içeriğini göstermektedir.

İki adresli komutlar:

İki adresli komutlar günümüz bilgisayarlarında çok kullanılmaktadır.

```
MOV R1, A  R1<=M[A]
ADD R1, B  R1<= R1+M[B]
MOV R2, C  R2<=M[C]
ADD R2, D  R2<= R2+M[D]
MUL R1, R2  R1<= R1 * R2
MOV X, R1  M[X]<=R1/td>
```

Her bir satır işlemleri hızlı bellek ifadeleriyle açıklanmıştır.

Bir adresli komutlar:

Bir adresli komutlar AC'yi bir hızlı bellek olarak kullanır. Diğer simbol ikinci hızlı bellek içindir.

```
LOAD  A  AC<=M[A]
ADD   B  AC<=AC + M[B]
STORE T  M[T]<= AC
LOAD  C  AC<= M[C]
ADD   D  AC<= AC +M[D]
MUL   T  AC<=AC*M[T]
STORE X  M[X]<=AC
```

Satırları yukarıdaki işlemi yapar.

Sıfır adresli komutlar:

Yığın organizasyonlu bir bilgisayarda adres alanı sadece MUL ve DIV gibi komutlar için kullanılır. PUSH ve POP komutlarında adres yoktur.

```
PUSH A TOS<=A  
PUSH B TOS<=B  
ADD    TOS<=(A+B)  
PUSH C TOS<=C  
PUSH D TOS<=D  
ADD    TOS<=(C+D)  
MUL    TOS<=(C+D)*(A+B)  
POP   X M[X]<=TOS
```

TOS yığının en üst adresini göstermektedir.

olmak üzere dört çeşit format tanımlanabilir.

## 8.5 ADRES MODLARI

Adres modu komutun adres alanından operandı bulmaya yarayan kurallıdır. Daha önceden açıklandığı gibi efektif adres hesaplamak yoluyla operanda ulaşılır. Adres modu kullanmanın iki önemli sebebinden bahsedebiliriz:

- Kullanıcıya programlamada göstericiler, sayıcılar ve indeksleme gibi üstünlükler sağlamak
- Komutta kullanılan bit sayısını indirmek

### 8.5.1 Komutun Koşturulma Fazları

Hatırlanırsa komutun bilgisayarda koşturulması belli fazların tamamlanmasıyla olmaktadır:

- Komutun ana bellekten alınması (fetch cycle)
- Dekode edilmesi (decode cycle)
- Koşturulması (execution cycle)

Mod alanı bu durumda komut formatına eklenecek, komutun hangi modda olduğunun da belirlenmesi gereklidir.



Mod alanı eklenmiş komut formatı

### 8.5.1.1 Adres Alanı ile Birlikte Kullanılmayan Adres Modu

Hatırlanacağı gibi PC koşturulacak bir sonraki komutun adresini gösterir. Dekod işlemi sırasında komutun adresinde bulunması gereklidir. Bu da adres modunun da öğrenilip, efektif adresin belirlenmesinin bu fazda tamamlanmasını gerektirir. Son fazda ise komut koşturulacaktır.

Bazı bilgisayarlarda operasyon kodu ile adres modu bir olabilir. İki tip adres modunda ise adres alanına gerek yoktur:



Saklı modda, operand komutta belirtilir. Örneğin, CMA: komplement AC komutunda AC komutun tanımından çıkarılır. Anında modda ise operand komutun içinde yer alır. Adres alanı yerine operandı taşıyan bölüm yer alır.

### 8.5.2 Adres Modlarının Assembler Notasyonu

Adres modlarını, bir adresli komut formatında LOAD komutu için assembler notasyonu ile gösterelim. Aşağıda LOAD komutu ile 8 adres modunun tanıtılması gösterilmektedir.

<i>Mod</i>	<i>Assembler notasyonu</i>	<i>Hızlı bellek transferi</i>
Doğrudan	LOAD ADR	$AC \leq M[ADR]$
Dolaylı	LOAD @ADR	$AC \leq M[M[ADR]]$
Bağılı	LOAD \$ADR	$AC \leq M[PC+ADR]$
Operandlı	LOAD #NBR	$AC \leq NBR$
İndeks	LOAD ADR(X)	$AC \leq M[ADR+X]$
Hızlı bellek	LOAD R1	$AC \leq R1$
Hızlı bellek dolaylı	LOAD (R1)	$AC \leq M[R1]$
Kendi-arttırmalı	LOAD (R1)+	$AC \leq M[R1], R1 \leq R1+1$

### LOAD komutu ile 8 adres modunun tanıtılması

Yukarıdaki tabloda ADR: adres ve NBR: sayı olarak tanımlanmıştır. X veya XR indeks için kullanılan hızlı belleği tanımlamaktadır. Şimdi Tabloda gösterilen modların özelliklerini tanımlayalım.

- Doğrudan mod (direct addressing mode): komutun adres alanı efektif adresi verir. Operand bu adreste yer alır. Notasyonunda herhangi bir takı kullanılmaz.
- Dolaylı mod (indirect addressing mode): komutun adres alanı, operandın efektif adresinin saklandığı yerin adresidir. Notasyon olarak LOAD @ADR dolaylı adresi gösterir.  $AC \leq M[M[ADR]]$  ifadesi ile adresin adresinden operandın elde edildiği gösterilmiştir.
- Bağıl adres modu (relative addressing mode): bu modda program sayacı (PC) komutun adres alanı ile toplanır. Elde edilen adres efektif adres olup, operand bu adresden alınır. Elde edilen adresin PC değerine yani sonraki adresin değerine bağlı olduğu gözlenmelidir.  $AC \leq M[PC+ADR]$  ifadesi bu modu net olarak açıklar. \$ notasyonu bu modu tanımlar.
- Operandlı mod (immediate addressing mode): operand komutun adres kısmında olup, herhangi bir efektif adres hesaplamasına gerek yoktur. # notasyonu anında modu tanımlar.  $AC \leq NBR$  ile sayı AC'ye yüklenir.
- İndeks mod (index addressing mode): operandın efektif adresi, komutun adres alanı ve X index hızlı belleğinin toplamı olarak hesaplanır. ADR(X) notasyonu adres ve X belleğini tanımlar.  $AC \leq M[ADR+X]$  ifadesi operandın elde edilip AC'ye yüklediğini açıklar. Komutun adres değeri ve indeks belleği farklıyla operandlara erişilir. Dizi elemanlarına bu yolla kolaylıkla erişilebilir.
- Hızlı bellek modu (register mode): hızlı bellek içeriği operandı oluşturur. Bellek adıyla tanımlanır.
- Dolaylı hızlı bellek modu (register indirect mode): Komut operandın adresini verecek hızlı belleği verir. Parantez (hızlı bellek) ile gösterilir.  $AC \leq M[R1]$  ifadesi ile R1 belleğinin gösterdiği adresin içeriği AC'ye yüklenir.
- Kendi-arttırmalı mod: Dolaylı hızlı bellek modunun benzeri olup, bellek değeri mikrooperasyon sonunda artırılır. LOAD (R1)+ ifadesinde R1 değerinin gösterdiği

adresteği içerik AC'ye yüklenikten sonra R1 değeri 1 arttırılır.  $AC <= M[R1]$ ,  $R1 <= R1+1$  bellek transfer ifadesi bu mikrooperasyonları gösterir.

Yukarıdaki tabloda gösterilmeyen bir diğer adres modu baz bellek adres modudur. Bu modda, baz belleğin içeriği komutun adres kısmına eklenerek efektif adres hesaplanır. Bu mod indeks adres moduna benzer, her iki adres modunda indeks ve baz bellekleri komutun adres kısmına eklenecek sayıyı tutar. Fark olarak, indeks belleğin içeriği her bir hesaplamaada değişerek istenilen dizi veya tabloya erişilir. Baz bellekte ise baz adres sabit kalıp, komutun adres kısmı değişir. Komutun adresindeki değişim yoluyla istenilen içeriye erişilir.

### 8.5.3 Sayısal Örnek

Farklı adres modlarının nasıl çalıştığını göstermek için aşağıdaki örnekte kullanılan komutun sayısal olarak çeşitli modlara göre değerlerini bulalım. 2 kelimelik komut yapısını göz önüne alalım. "AC'ye yükle" komutu 100 ve 101 adreslerinde saklı olsun. İlk kelime komutun op kodu ve adres modunu göstersin. İkinci kelime ise adresi tanımlasın. PC değeri komutu alabilmek için 100 olsun. R1 belleğinin içeriği 300 ve indeks belleği XR'nin içeriği 100 olsun. Aşağıda ana bellek ve hızlı belleklerin verilen adresler için içerikleri gösterilmiştir.

Efektif adres hesaplanması, operandlı mod için söz konusu değildir. Doğrudan ve dolaylı modda ise komutun adres alanı ve bu alandaki adresin içerikleri olmaktadır. Bağıl modda PC ve adres alanı, indeks modunda ise indeks belleği (XR) ve adres alanı kullanılır. Hızlı bellek modlarının özelliği, ana bellek yerine hızlı bellekteki içeriğin ya doğrudan (bellek içeriği) ya da dolaylı bellek (bellek içeriğinin gösterdiği adres) olmasıdır. Kendi artırmalı modda ise içerik AC'ye yüklenikten sonra bellek değeri 1 artırılır.

ADRES ANA BELLEK	
PC=100	100 Load to AC Mod
R1=300	101 Adres=400
XR=100	102
	103
	:
Bilgisayarın bazı hızlı bellekleri ve değerleri	300 750
	:
	400 1000
	:
	500 444
	:
	502 333
	:
	1000 500

2 kelimelik komut

Sonraki komut

Adres modları için sayısal örnek

### 8.5.3.1 Sayısal Örnek İçin Adres Mod Değerleri

Aşağıda sayısal örnek için çeşitli adres mod değerleri gösterilmiştir.

Adres modu	Efektif adres	AC'nin içeriği
Doğrudan adres	400	1000
Operandlı mod	101	400
Dolaylı mod	1000	500
Bağıl adres	502	333
İndeks adres	500	444
Hızlı bellek	-	300
Hızlı bellek dolaylı	300	750
Kendi artırmalı	300	750

Sayısal örnek için adres mod değerleri

Sonuç olarak, burada tanımlanan adres modları programcıya yeni ve çok önemli olanaklar sunmaktadır. Bugünkü teknolojide, kişisel bilgisayarlar (Pentium I-IV, vs.) yukarıda bahsettiğimiz adres modlarını yoğun olarak kullanmaktadır. Programcı adres modlarının kendisine sağladığı kolaylıklarla çeşitli uygulamaları yapabilmektedir.

## Bölüm Özeti

- Bu bölümde, merkezi işlem birimi detayları ve çeşitleri ile tanıtılmıştır. Ayrıca, çeşitli komut formatları ve adres modları detayları ile açıklanmıştır. Veri transferi, program kontrolü ile ilgili komutlar ve özellikleri özetlenmiştir.

## BÖLÜM - 9

## GENEL ÖZET

Bu derste tek akümlatörlü (AC) temel bilgisayar yapısı tanıtılmış, donanım ve yazılım olarak detayları incelenmiştir. Daha sonra bilgisayar yapılarının çeşitleri anlatılmıştır. Bunlar arasında genel hızlı bellek yapılı ve yiğin yapılı bilgisayarlar tanıtılmıştır.

Bilgisayarı oluşturan bileşenler lojik kapı ve FF seviyesinden başlanarak ele alınmış ve donanım yapısındaki kullanımları incelenmiştir. Saatli ardisil lojik devre tasarımının temel kuralları çerçevesinde bileşenler oluşturulmuş ve bilgisayarda kullanıldıkları yerler blok diyagramlarda gösterilmiştir. Sonucta, kağıt kalem ile belli kabuller altında temel bilgisayar tasarılmıştır.

Temel bilgisayarın kontrol ve zamanlaması iki yöntemle yapılmıştır:

1. Donanımla kontrol: lojik kapılarla, hızlı, değiştirilemeyen kontrol yapısı
2. Mikroprogramlamayla kontrol (Yazılımla): ROM' da saklanmış mikrokomutlarla kontrol, daha yavaş ve değiştirilebilir.

## 9.1 GENEL ÖZET (DEVAM)

**Bu dersin içeriğinde üç ayrı yazılımı görmüş olduk, bunlar;**



Tekrar

- Hızlı bellek transfer dili: hızlı belleklerin transferlerinin tanımlandığı semboller (mikroişlemler)
- Assembler dili: alt seviyede işlemlerin yapıldığı komutlar
- Mikroprogramlama dili: mikrokomutlardan (semboller) oluşan ve kontrol belleğinde (ROM'da) saklanan alt rutinlerden oluşan dil

Temel bilgisayar ile günümüz kişisel bilgisayarlarında karşılaştığımız fark, donanım ve yazılım bileşenlerinin artan ölçülerde kullanılması olarak özetlenebilir. Örneğin, temel bilgisayarda, tam komut seti içerisinde sadece toplama (ADD) komutu kullanılır. Diğer işlemler: çıkarma, ikili sayının 2'li komplement ile toplaması, ikili sayının sola kaydırılarak 2 ile çarpımı ve 2'ye bölme ile sağa kaydırılması olarak gerçekleşmektedir. Günümüz bilgisayarlarında, her bir işlem için ayrı devreler kullanılmaktadır.

Burada anlatılan bilgilerin genelleştirilmesi ve yeni bilgilerle desteklenmesi ile çeşitli bilgisayar organizasyonlarını incelemek mümkün olabilecektir.

Günümüz bilgisayarlarında ana bellek, hızlı bellek elemanları işlemci gücünün yanında çok daha büyük değerlerde ve ucuza sağlanmaktadır. Bu ise çoklumeda (Multimedia) uygulamalarını hızla artırmaktadır. Ses, görüntü, grafik ve yazı gibi bileşenler Internet ortamında yayılmaktadır. Diğer bir uygulama ise elektronik ticaret (E-commerce) uygulamalarıdır. Bilgisayar teknolojisi, uzaktan alışveriş, konferans, robot, tıp, vb. alanlardaki uygulamalara hızla yerleşmiştir. Yaygınlaşan diğer bir teknoloji ise telsiz haberleşme (wireless communication) teknolojisidir. Cep telefonları (mobile phone) uygulamaları ve bilgisayar bağlantıları hızla artmaktadır. Hafif düzenlerin verimli olarak kullanılması, bilgisayarla bağlantısının hızlı ve ucuz olarak sağlanması için ise yeni

programlama teknikleri ( light weight programming) geliştirilmektedir. Bununla ilgili uygulamalar başlamış, görüntü alışverişi ve belirli tıp uygulamaları (acil yardım ve bilgilendirme) önem kazanmıştır.

## 9.2 GENEL ÖZET (DEVAM)

Burada degenilmesi gereken diğer bir alan ise C, Pascal gibi yüksek seviyeli programlama dilleri yanında "nesneye dayalı" yüksek seviyeli programlama dillerinin güncellesmesidir: C++, Java, Perl, vb. gibi diller önem kazanmış, web sitelerinden kullanıcı bilgisayarına indirilen veya uzak sitelere yüklenebilen ve koşturulabilen bu programlar internet teknolojilerinde çığır açmıştır.

Bilgisayar teknolojisinin bugün eriştiği nokta, kişisel bilgisayarların, Windows sistemlerinin yaygın olarak kullanıldığı, internet erişiminin gittikçe artan hızla



yaygınlaştiği bir dönemdir. Doğal olarak, VLSI teknolojisinin küçültüğü ve kuvvetlendirdiği yeni modeller pazara çıkarılmaktadır.

Son olarak, bilgisayar, haberleşme ve elektronik teknolojilerinin gittikçe birbirine yaklaşan eğilimlerinden bahsetmek gereklidir. Televizyon cihazının aynı zamanda bir bilgisayar gibi kullanılabileceği veya tersine bir bilgisayarın televizyon olarak yorumlandığı çeşitli uygulamalarda görülmüştür. Sayısal televizyon kanalları bir taraftan seyirciye görüntü verirken, diğer taraftan banka hesabına bağlanma kolaylığı sağlayabilmektedir. Günümüz ihtiyaçlarına cevap verecek sistemler bu teknolojilerin birleşimi veya kesiminden doğacaktır.

## Ders Sonu

**"Bilgisayar Organizasyonu"** dersinin sonuna gelmiş bulunmaktayız. Bu çalışmamızın tüm öğrencilerimize faydalı olması dileğiyle...

Sınavlarınızda başarılar...

Prof. Dr. Fikret Gürgen