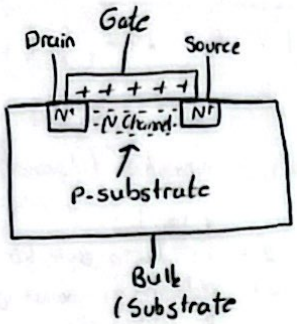
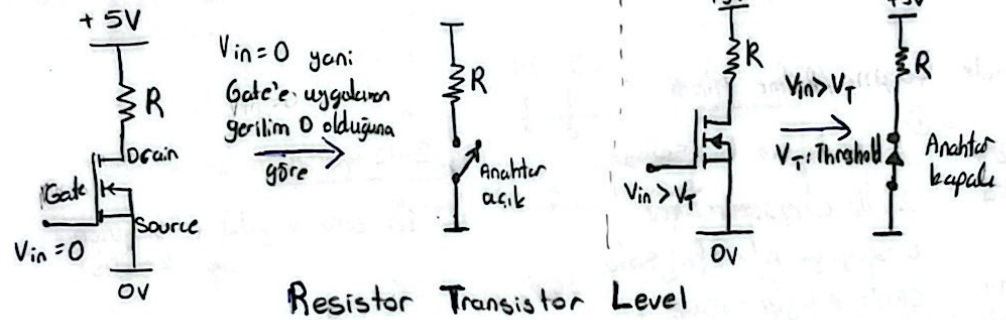
Mikroişlemci Hafta-1

Mosfet bir transistör türüdür.

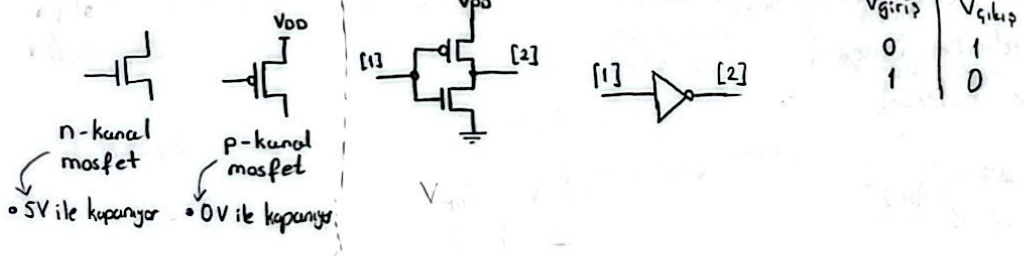


- Gate'e 5 V gerilim uyguladık ve akım akmıyordu. Gate'e verilen gerilim elektrik alan oluşturur ve "+" yükler toplanır. P substratı bölgesinden "−" yükler N channel'a çekilir. Böylece drain source arasındaki kanal iletime geçmiş olur.

Mosfet ile anahtarlama yapılır.



Resistor Transistor Level



Transistor Transistor Level

Mikro işlemci

- ALU, register, kontrol birimlerinin bulunduğu bir çiptir
- RAM, ROM, input, output ilişkilerini sağlayacak ara birimler bulunmuyor

Mikro denetleyici

- input, output, RAM, ROM ve diğer ara birimler bulunmaktadır bize sadece kod yazmak kalır.

FPGA

- içinde hiçbir şey yok sadece logic kapıların olan ham bir çiptir.
- İçinde ALU, register hiçbir şey yok.

	Number of Bits	Common Representation Time
	1	Bit / Digit / Flag
	4	Nybble / Nibble
1 byte	8	Byte / Octet / Character
2 byte	16	Double Byte / Word
4 byte	32	Double Word / Long
8 byte	64	Very Long Word

$2 + 8 = 10$

$(1+1)_2 = 10$

$(3+5)_8 = 10$

$(9+7)_{16} = 10$

2'lik Sayı Sisteminde Toplama (İşaretsiz)

```
  1 0 1 1 1 0 0 1    185
+ 0 0 0 1 1 0 1 1     27
-----------------------
  1 1 0 1 0 1 0 0    212
```

2'lik Sayı Sisteminde Negatif (İşaretli)

```
  1 1 1 1 1 1 1
  0 0 0 1 1 0 1 1    27
  1 1 1 0 0 1 0 1   -27
  0 0 0 0 0 0 0 0
```

1. yöntem: Ben buna öyle bir sayı ekliycem ki sonuç 0 olucak Bu eklediğim sayı -27 olur

2. yöntem: Sağdan başla ilk 1'i gördüğünde onu yaz. diğerlerinin hepsinin tümleyenini alacaksınız.

2'lik Sayı Sisteminde Çarpma/Bölme İşlemi

```
(1)
x2 ↓  0 0 0 1 1 0 1 1    27
      0 0 1 1 0 1 1 0    54
```

Örneğin yukarda 1 defa sola kaydırılmış

① Eğer ki siz bir sayıyı 2^n ile çarpacaksanız o sayıyı n defa sola shift ediyorsunuz

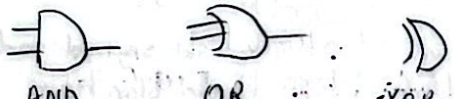
Bölmede ise 2^n ile bölünecekse sayı n defa sağa kaydırılır. Örneğin 27 sayısını 2^1 ile bölersek

```
  0 0 0 1 1 0 1 1    27
  0 0 0 0 1 1 0 1    13   ↓ Böl 2^1
```

3 ile çarpma

1 defa sola kaydır + sayının kendisi

AND, OR, XOR



AND ve

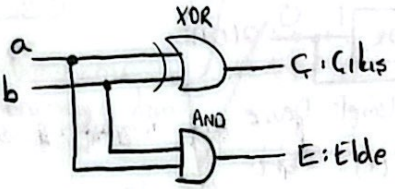
OR veya

XOR ya da

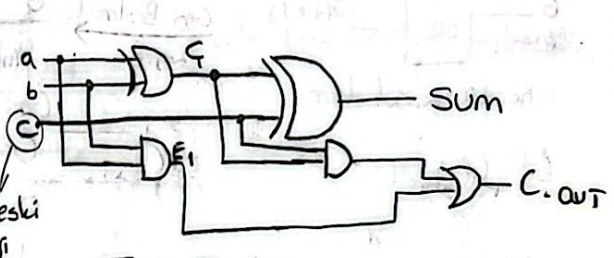
Yarı ve Tam Toplayıcı

a	b	Ç
0	0	0
0	1	1
1	0	1
1	1	0

- Görüleceği üzere XOR toplayıcı devre için oldukça uygun
- a:1 b:1 için elde tutmak gerekir AND kapısı bunun için uygundur.

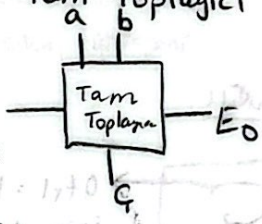


Yarı Toplayıcı

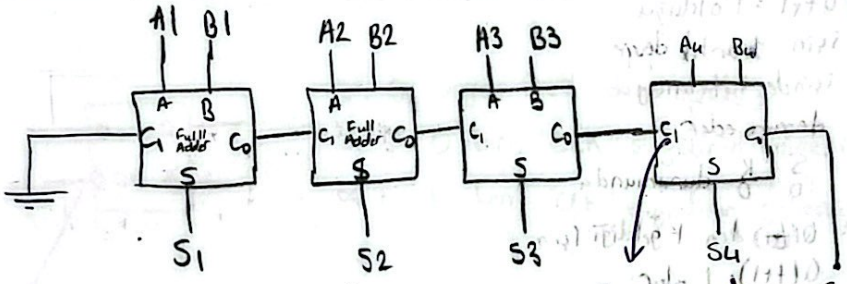


bu eski eldeyi temsil etmektedir

Tam Toplayıcı



4 bit tam toplayıcı: 4 bitlik iki sayı bu şekilde toplanabilir.



$2^4 = 16$, 16'ya kadar olan sayılar toplanabilir

Carry Bit elde biti bir sonraki tam toplayıcıya aktarıldı

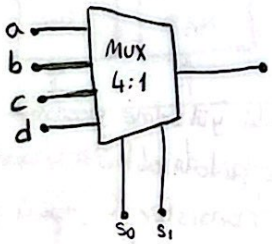
Co: Bu ilk elde, genel olarak ilk toplama işleminde "0" kabul edilir

Bu çıkışa verilen $\frac{S_1}{1}\ \frac{S_2}{2}\ \frac{S_3}{3}\ \frac{S_4}{4}$

Transistör → Kapılar → Hesaplayıcılar (Tam Toplayıcı, Çarpma, Bölme) → Aritmatic Logic Unit (ALU)

Çoğullayıcı (Multiplexer (MUX)) Seçici

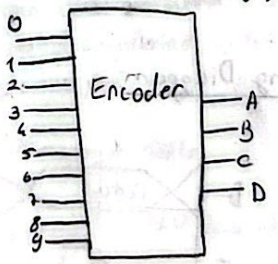
- Paralel olan veriyi seri hale getiriyor



S_0	S_1	Output
0	0	a
0	1	b
1	0	c
1	1	d

DEC → BCD Kodlayıcı (Encoder)

- Desimal (Onluk) sayı, ikilik sayıya dönüştürür



A B C D

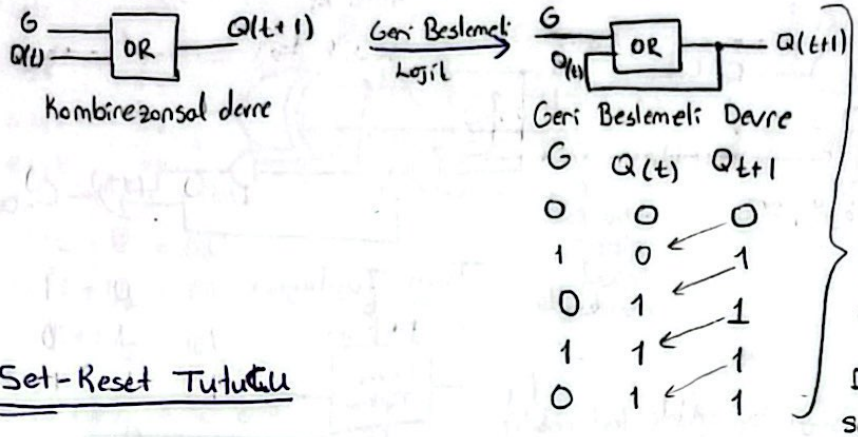
9 → 1 0 0 1

7 → 0 1 1 1

∝ Bu gördüğümüz devreler ileri beslemeli devrelerdir ve bunlara kombinezonsal devre denir

∝ Geri beslemeye sahip olmayan devrelere kombinezonsal devre denir. ALU'nun merkezinde bulunurlar.

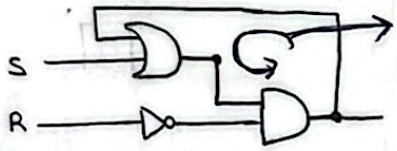
Geri Beslemeli Lojik



G	Q(t)	Qt+1
0	0	0
1	0	1
0	1	1
1	1	1
0	1	1

Bunu reset yapamıyoruz Qt+1 0 olamıyor artık Dolayısıyla 1 bitlik bilgiyi sürekli tutar, değişmiyor

Set-Reset Tutucu



Qt+1 = 1 olduğu için 1 artık devre içinde tutulmaya devam eder.

S R durumunda Q(t) den 1 geldiği için Q(t+1) 1 olur

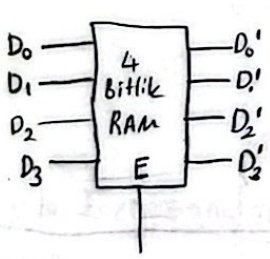
S	R	Qt	Qt+1
0	0	0	0
1	0	0	1
1	0	1	1
0	0	1	1
Reset 0	1	1	0

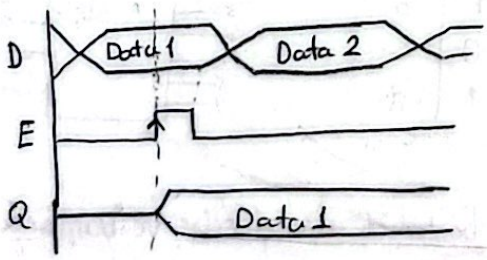
Böylece bir bitlik bilgi tutulur.

1 Bitlik RAM

- Write Enable 1 olduğunda DATA IN verisi alınır ve Data Out'a verilir.
- Write Enable 0 olduğunda DATA OUT korunur, değişmez.

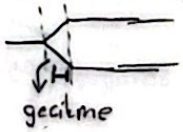
4 Bitlik RAM - Timing Diagram



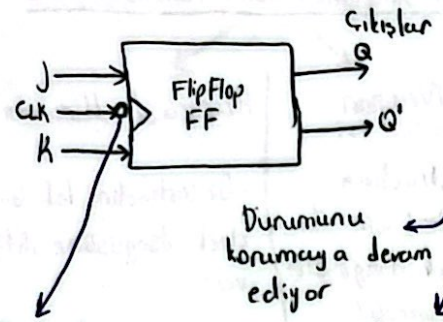


timing diagram

Enable daki yükselme gecikme ifadesini anlatmaktadır hardware kaynaklı transistör kaynaklı gecikmeler olabilir bundan dolayı Q çizimi aşağıdaki şekildedir



J-K Flip Flop Devresi



Girişler			Çıkış
J	K	CLK	Q
0	0	↓	Q_0
0	1	↓	0
1	0	↓	1
1	1	↓	Toggle

- CLK düşen kenar "↓" ya da çıkan kenar "↑" tetiklemeli olabilir.
- Toggle "ters çevirme" işlemidir.

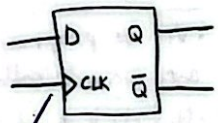
Durumunu korumaya devam ediyor

Çıkış 0'lanıyor

▽ Burada top 0 olması demek bunun düşük kenar tetiklemeli olduğu anlamına gelir.

O zamanki durum ne ise onun tümleyeni Q_0'

Data Flip Flop Devresi (D Tipi Flip Flop)



▽ Top yok, yükselen kenar tetiklemelidir.

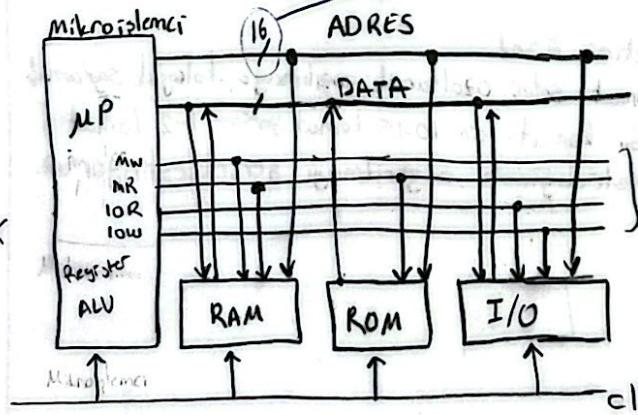
Q = 0 veya Q = 1 durumunda;

∝ D = 0 iken, "CLK" sinyalinin gelmesi ile çıkış Q = 0 değeri olur.

∝ D = 1 iken, "CLK" sinyalinin gelmesi ile çıkış Q = 1 değeri olur.

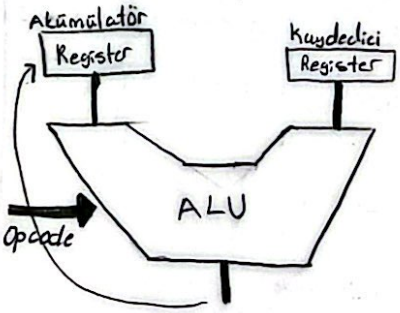
Mikrobilgisayar Yapısı

→ Bunun anlamı 16 tel vardır, 16 yol veriler demek



MW: Memory Write MR: Memory Read
IOR: Input Output Read
IOW: Input Output Write

∝ Adres ve datanın akışının düzenlenebilmesi için kontrol sinyallerine ihtiyacımız var.



∝ Akümülatör özel bir register, ALU'dan elde edilen sonuçlar akümülatör üzerine kaydedilebilir; a++, a+=3, a=a+3 durumlarında

Opcode	Mnemonic	Açıklama
1 0 1 1	INC A	A registerının içeriğini 1 artır

! Sınavda grafik çıkabilir ok falan

Bilgisayar Mimarileri ve 8086

Kelimeler

Instruction Set : Komut Kümesi
Fetch : Getirmek çekmek (RAM'den veri)
Execute : Çalıştırmak, işlemek
Pipeline : Ardışıl Çalışma
Decode : Komut Çözme (kod çözme)
Interface : Arayüz
Opcode : İşlemcinin çalıştıracağı komut
Operand : İşlenen kısım
Bus : Yol

CPU Sınıflandırma

> Donanımsal Mimari → Von Neuman Mimarisi
> Harward Mimarisi
> Komut Kümesi → Reduced Instruction Set Computer (RISC)
> Complex Instruction Set Cumputer (CISC)
> Bit Sayısı → 32 Bit µP
> 64 Bit µP

CPU Sınıflandırma: Donanımsal Mimari

Von Neuman Mimarisi	Harvard Mimarisi
• Bir instruction çalıştırabilmek için iki tane clock döngüsüne ihtiyaç vardır.	• Bir instruction tek bir clock döngüsüne ihtiyaç var
• Daha ucuz	• Daha pahalı
• CPU instruction'a erişme ve read/write işlemini aynı anda yapamaz	• Yapar
• İşlemler için aynı yollar kullanılır	• Yollar ayrıdır
• Veri ve program aynı birimde saklanır	• Veri ve program ayrı birimde saklanır. • Hız iki kat çıkar

CPU Sınıflandırma: Komut Kümesi

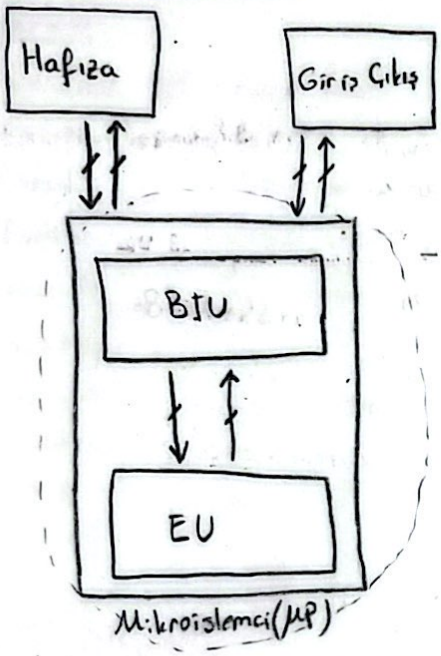
RISC
"Komut Çalışma Süresini Azaltma"
• Amaç çok hızlı çalışan küçük komutlar ortaya koymak
• Yani tek bir cycle'da birden çok komutun çalışmasını istiyoruz
• Complex fonksiyon için bir sürü komut
• Küp alma işlemi için 3 kere çarp
• Çok hızlı çalışıyor

CISC
"Komut Setini Azal
• Amaç komut setini azaltarak yazılımcıya kolaylık sağlamak
• Complex bir komut için 10-15 komut yerine 1-2 komut yazarak istediğimiz algoritmayı gerçekleştiriyoruz.

Farkları nedir Sınavda Çıkıyor

Mikroişlemci - Hafta 3

EU ve BIU Birimleri



Mikroişlemci temel olarak dışarıdan bilgiyi alır, işler ve bir yerde saklar.

Bus Interface Unit (BIU): EU'nun sekreteryası
- > Gelen bilgiyi sıralar
- > Gelen komutları diziyor
- > EU üzerinden gelen emirleri icraa ediyor
- > EU portlarına veri okuma/yazma işini yapıyor.

Execution Unit (EU):
- > Bütün emirler buradan çıkıyor, BIU bu emirler doğrultusunda bilgileri organize edip EU'ya getirir.

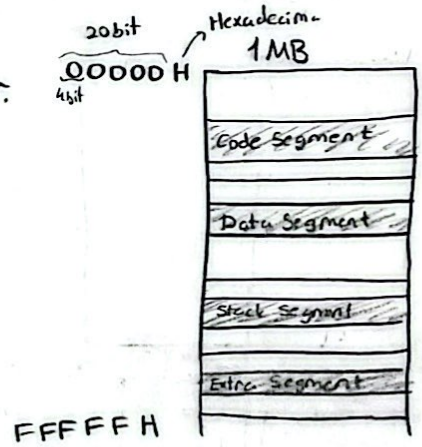
BIU 3 fonksiyonel bölümden oluşur
- Komut İşaretçisi (Instruction Pointer (IP))
- Bölüt Yazmacı (Segment Register (SG))
- Komut Sıralama (Instruction Queue)

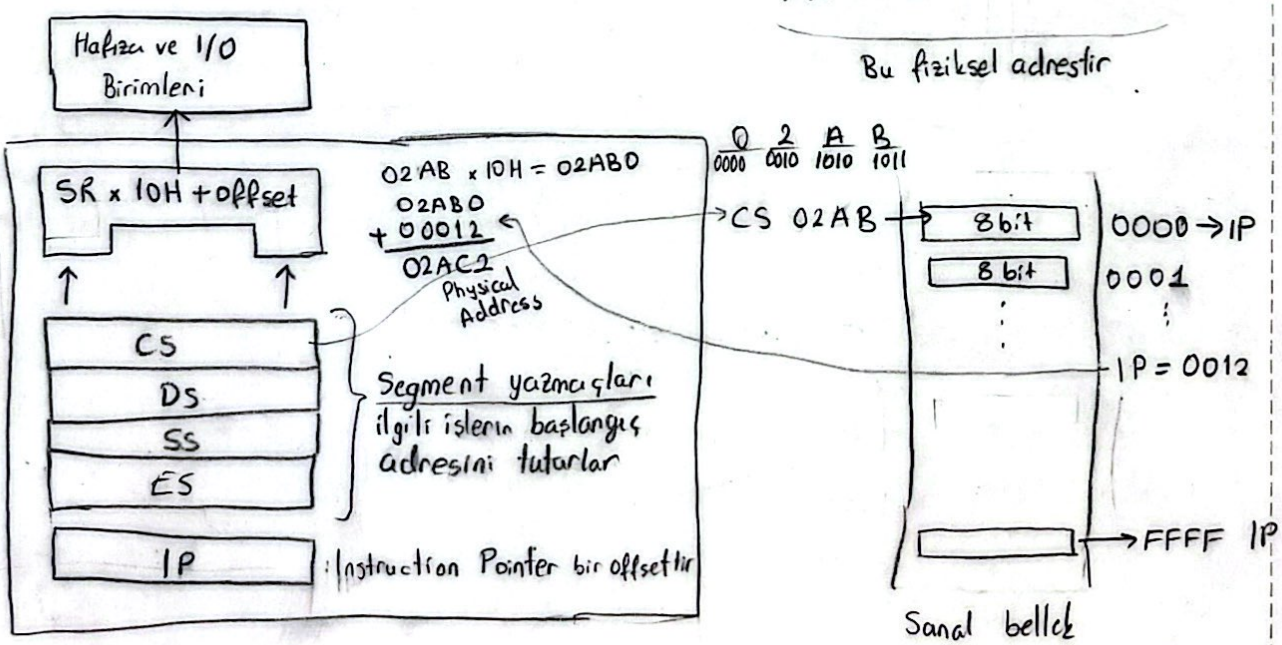
∝ 8086 Mikroişlemcisi 20 bit fiziksel adres yoluna sahiptir.

∝ 2^{20} = 1 MB yer adreslenebilir.

∝ İşlemci içindeki register 16 bit uzunluğundadır.

? 20 bit'i biz direk işleyemiyoruz. Registerlarımız 16 bit. Dolayısıyla sanal bellek kullanmalıyız.





SORU: CS= 1234H , IP= 0005H Fiziksel adres nedir?

```
 12340
+00005
-------
 12345 H
```

S2B, a) IP: 0019H olarak verilmiş RAM gözünün mantıksal (logical) ve fiziksel (physical) adresini hesaplayınız. Segmentin başlangıç ve bitiş fiziksel adreslerini bulunuz

```
 01230
+00019
-------
PA: 01249
```

LA : 0123 : 0019

lower range of PA
0123 : 0000

LP of PA : 01230

upper range of PA
0123 : FFFF

```
 01230
+FFFF
-------
 1122F
```

Data Segment (DS) "BIU içerisinde" "Segment Register"

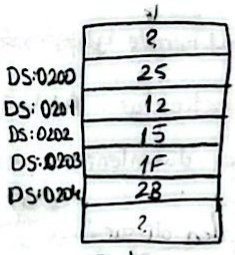
- DS'nin 3 tane offset'i bulunmaktadır.

DS : BX
DS : SI
DS : DI
} Bu offsetler BIU içerisinde değiller. EU içerisindeler.

- Verinin tutulduğu, dışarıdan gelen verilerin saklandığı, üretilen verilerin barındırıldığı yerdir.

AH (8bit) AL (8bit) } AX register → AX akümülatör olarak kullanılabiliyor.

```
MOV AL; clear AL
ADD AL, [0200]; add the contents of DS:200 to AL
ADD AL, [0201]; add " " " DS:201 " "
ADD AL, [0202]; add " " " DS:202 " "
ADD AL, [0203]; add " " " DS:203 " "
ADD AL, [0204]; add " " " DS:204 " "
```

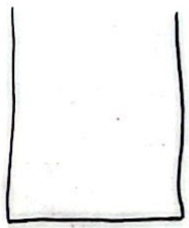


Data Segment

00 + 25 → 25
25 + 12 → 37
37 + 15 → 4C

Stack Segment (SS) "BIU içerisinde" "Segment Register"

- Yığın bölütü (Stack Segment), CPU tarafından bilgileri geçici olarak depolamak için kullanılan bir RAM bölümüdür.
- CPU registerları sınırlı olduğundan bu alan kullanılır
- PUSH ve POP komutları ile veri yığılır ve çağırılır
- Mantıksal Adresi SS:SP


Stack Segment

LIFO

S2B.b) SI = 025A H ile gösterilmiş RAM gözünün fiziksel adresi PA = 443EA H olarak verilmiştir. İlgili segment register'ının başlangıç adresini bulunuz. Yazmacın ne olduğunu belirtiniz

```
  4419
  abcd0
+ 0025A
-------
  443EA
```

DS
başlangıç adresi : 44190

Extra Segment (ES) "BIU içerisinde" "Segment Register"

- Stack Segmentinde artık yer kalmadıysa ihtiyaç duyulması halinde devreye giren segment

Instruction Queue (Komut Sıralama)

- Komut çekme ve çalıştırma işinin daha verimli yapılmasını sağlar.
- 8086'da <<pipeline>> Instruction Queue sayesinde olur
- Toplam 6 byte kapasitesi vardır
- <<FIFO>> prensibine göre çalışır.

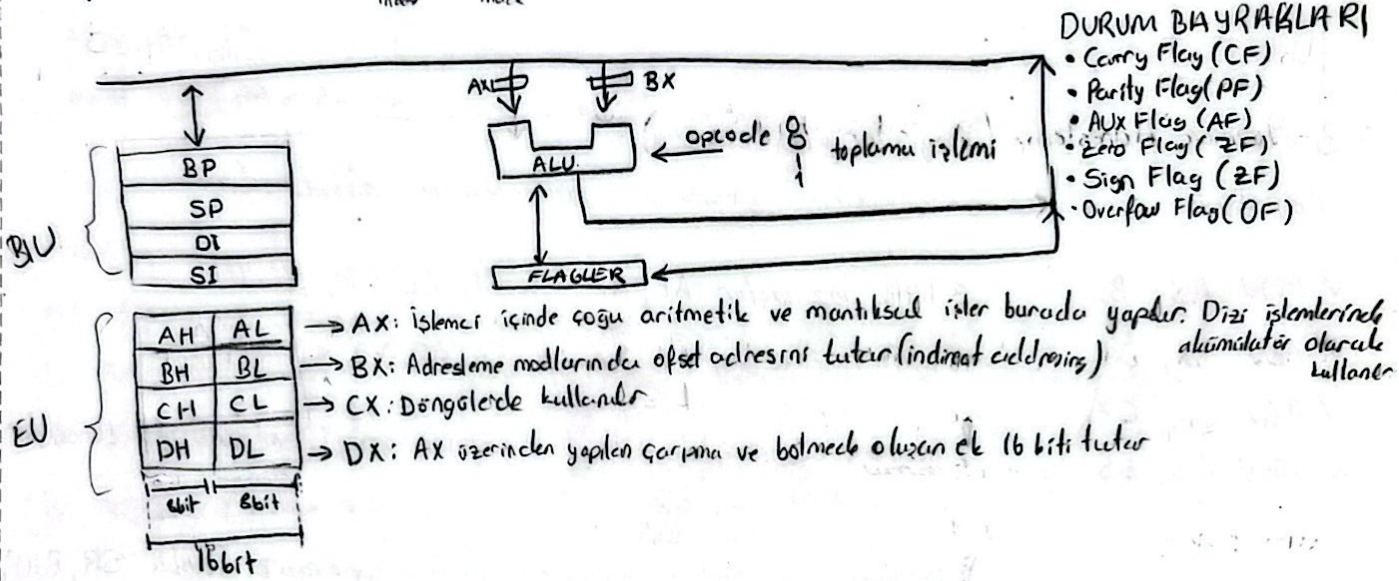
Execution Unit (EU) (içinde ALU var)

∝ Verinin ve komutun nereden çekileceğini BIU'ya söyler
∝ Komut çözme işi EU içinde yapılmaktadır
∝ Elde edilen instructionlar ALU içinde çalıştırılır.
∝ EU ayrıca iç işleyişin düzenlenmesi için kontrol birimi içerir

EU 5 fonksiyonel bölümden oluşur:

- Genel amaçlı registerlar (General Proposed Registers (GPR)) AX BX CX DX
- İşaret ve İndex registerları (Pointer Index Register)
- Aritmetik mantık birimi (Arithmetic Logic Unit (ALU))
- Bayrak yazmacı (Flag Register (FR))
- Zamanlama ve Kontrol Birimi (Timing and Control Unit)

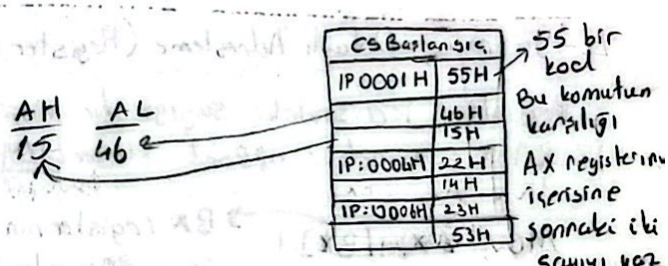
Segment	Offset Registers	Function
CS	IP	Address of the next instruction
DS	BX, DI, SI	Address of data
SS	SP, BP (StackPointer, Base Pointer)	Address in the stack
ES	BX, DI, SI (Data Index, Source Index)	Address of destination data



Mikroişlemci - Hafta 4

→ Buradaki MOV Mnemonic'tir. Assembly dilini oluşturur, Makine dili sadece 1 ve 0'lardan oluşur

```
MOV AX, 1546H  :MOV 16 bitlik AX registerına sabit 1546 sayısını atıyor
MOV BH, 14H
```



Adresleme Modları

BH BL
14

22 sayısı: BH registerının içerisine sonraki bir sayıyı alıcı

∝ Her bir işlem için farklı opcode olmaması için adresleme modlarına ihtiyaç duyarız

1: Hemen Adresleme

• Bir register ve sabit sayı

MOV AX, 1546H (register, sabit sayı)

MOV BH, 14H (register, sabit sayı)

MOV BL, 0053 53H (register, sabit sayı)

2- Doğrudan Adresleme (Direct Addressing)

Doğrudan bir RAM gözünün içindeki sayı register'a atanır ya da registerdan RAM gözüne atama yapılması durumu da olur.

ğrudan a: MOV AX, [1546H]
emen a: ADD AX, 08H
emen a: MOV AH, EDH
ğrudan a: MOV [1548], AL

10010
2102
12112

3- Yazmaç Adresleme (Register Addressing)

Registerlar arasında gerçekleşen atamalar, adreslemeler dikkate alınır.

✓ MOV AX, BX
✓ MOV BX, DI
✓ MOV SI, CX
× MOV DS, SS

8 bitlik yazmaçlar AL, AH, BL, BH, CL, CH, DL, DH
16 bitlik yazmaçlar AX, BX, CX, DX, SP, BP, SI, DI

! Yazmaç adreslemede kullanılan yazmaç genişlikleri uyumlu olmalıdır.
X MOV AL, BP
8 bit 16 bit

! Segment registerları birbirine atama yapamaz. Çünkü SR, BIU'nun içerisinde yer alıyor. MOV işleminin çalışabilmesi için sayının EU'nun içerisindeki registerlardan birine aktarılması gerekiyor.

! mov [BP], [SI] iki tane RAM gözü arasında doğrudan atama yapılamaz

4- Yazmaç Dolaylı Adresleme (Register Indirect Addressing)

Registerın içerisindeki sayıyı bir registera atamıyoruz. Registerin içerisindeki sayı bir RAM gözündeki offset değerindedir.

MOV AX, [BX] → BX registerının içerisindeki sayı, RAM adreslerinde aranır. RAM adresinin gösterdiği değer AX'e atanır.

5- ~~Register Dolaylı Adresleme~~ Taban Mod Adresleme (Based Mod Addressing)

B ile başlamış → sayı var

MOV AX, [BX+52H]
MOV BL, [BP+14H]

1453:0A0B
2535

6-Index Mod Adresleme

(el sayı)

MOV AX, [SI + 52H] Stack Index + displacement

MOV AL, [DI + 14H] Data Index + displacement

7- Taban Index Modlu Adresleme

MOV AX, [BX+SI] Base Register + Stack Index

MOV AL, [BP+DI] Base Pointer + Data Index

Base + Index + Displacement

1. RISC mimarisinin özelliği nedir?

A) EU içinde hesaplama yapılan yer. (ALU)

B) Geliştirilmiş komut kümesidir ve tek komut kullanılarak birçok iş yapılabilir (RISC)

C) BIU içinde sıralamayı sağlayan birim (Queque

D) İndirgenmiş komut seti olup çok hızlı çalışır

2. Hatalı olan atamayı seçiniz

☐ MOV BX, CX

☐ MOV [2203], AL

☒ MOV AH, SI (AH 8bit, SI 16 bit. Hata verir)

☐ MOV BL, [SI]

Mikroişlemci - Hafta - 6

Opcode	Direction	Width	MOD	REG	R/M

03
B5
35
12
} Bunlardan hangisi kod? Ne neyi ifade ediyor?

Opcode (6 bits)	D (1 bit)	W (2 bits)	MOD (2 bits)	REG (3 bits)	R/M (3 bits)	Lower order bits of displacement	Higher order bits of displacement

> Opcode operasyon kodu anlamına gelmektedir. Her instruction (genellikle) 6-bit opcode'a sahiptir. Örneğin MOV 100010 dur.

> D dediğimiz bit işlemin register'a doğru mu? D=1
register'dan mı? D=0

> W elimizdeki işlemin 8 bit mi? W=0
16 bit mi? W=1
olduğunu söyler

Alıştırma

Mnemonic	OPCODE
MOV	100010
ADD	000000
SUB	110011
MUL	111000

	opcode	D	W
• MOV (AX) 1546H	100010	1	1
• SUB (AH), AL	110011	1	0
• ADD (BX), [0233]	000000	1	1
• MUL [BP+SI+03], [0233]	✗ Bu şekilde işlem olmaz		
• ADD [BP+SI+03], (CX)	000000	0	1
MOV [DI], AX	100010	0	1

∝ Atamalarda 2 farklı etkileşim olabilir

1. etkileşim: Yazmaç ile yazmaç etkileşimi
2. etkileşim: Yazmaç ile hafıza bölgesi

MOD field:

Code	Mode	Meaning	
00	Memory	Yer değiştirme yoksa ADD AX, [BX]	no displacement
01	Memory	8-bit yer değiştirme MOVE [BX+08H], CL	displacement 8bit
10	Memory	16bitlik yer değiştirme SUB [BP+1642H], AX	disp 16 bit
11	Register	Register Register modu	

Örnek// Makina kodlarını oluştur.

	Opcode	D	W	MOD	
MOV (AX), [BX]	100010	1 (registera)	1 (16bit)	00 (no-disp)	→ 2 register yok → Memory'de yer değiştirme yok
ADD [SI+02], CL	000000	0 (RAM'e)	0 (8bit)	01 (8bit)	ADD işlem sonucu Memory'e yazılmış
SUB SI, [Bx+SI+1632] (16bitlik yer değiştir)	110011	1 (registera)	1 (16bit)	10 (16 bit)	

Register Bilgisi Register'ın ne olduğunu ifade etmeniz gerekiyor

REG or R/M when MOD=11		
REG R/M	W=0	W=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Örnek register to register MOD 11 adressing

MOV (AX) BX (AX'e bakılır, BX'e bak)

opcode	D (register)	W (16bit register)	MOD (register to register)	REG	R/M
10010	1	1	11	000	011

mov [SI], (AX) MOV (BX), [BP+01]

Örnek:11

	opcode	D	W	MOD	REG	R/M
ADD SI, BP	000000	1	1	11	110	101

0000 0011 1111 0101

03F5

Hafızadan MODE türü için R/M tablosu (MOD ≠ 11 olacak)

R/M	MOD = 00 (no displacement)	MOD = 01 (8-bit displacement)	MOD = 10 (16-bit displacement)
000	BX+SI	BX+SI+D8	BX+SI+D16
001	BX+DI	BX+DI+D8	BX+DI+D16
010	BP+SI	BP+SI+D8	BP+SI+D16
011	BP+DI	BP+DI+D8	BP+DI+D16
100	SI	SI+D8	SI+D16
101	DI	DI+D8	DI+D16
110	direct	BP+D8	BP+D16
111	BX	BX+D8	BX+D16

Örnek:11 Protokolünü oluşturunuz.

	opcode	D	W	MOD	REG	R/M	
SUB AX, [BX+04H]	100011	1	1	01	000	111	1100 1111 0100 0111 (C F 4 7)
ADD AX, BX	000000	1	1	11	000	011	0000 0011 1100 0011 (0 3 C 3)
MOV AX, [BX+DI]	100010	1	1	00	000	001	1000 1011 0000 0001 (8 B 0 1)
MOV AX, [BX+DI+2H]	100010	1	1	01	000	001	1000 1011 0100 0001 (8 B 4 1) Displa... 0000 00... (0 2)
MOV AX, [BX+DI+1234H]	100010	1	1	10	000	001	1000 1011 1000 0001 (8 B 8 1) 0011 0100 (3 4)

0001 00... (1 2)

ADD SI, [DI+1235H] → 03 B5 35 12 (düşük byte önce iletilir)

	opcode	D	W	MOD	REG	R/M	Displacement
MOV [BP+SI+0233H], AX	100010	0	1	10	000	010	0011 0011 0000 0010 (3 3 0 2)

1000 1001 1000 0001 0011 0011 0000 0010

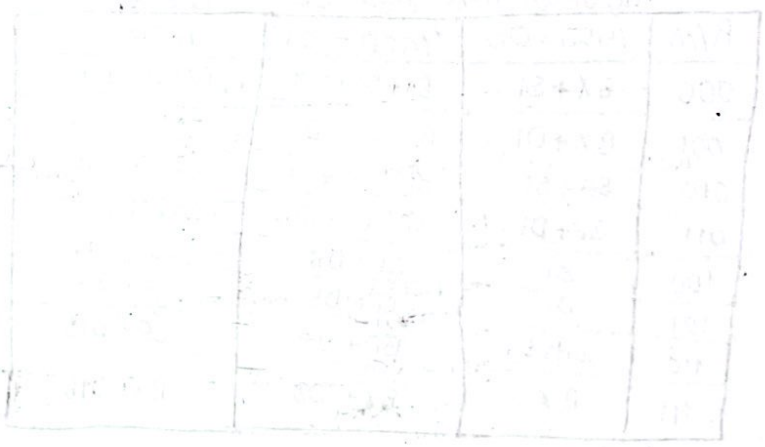
Örnek 89 Bu makina kodunun mnemonic versiyonunu yazınız
47
02

1000 1001 0100 0111 0000 0010

8bit

opcode-b	D	W	Mod	REG	R/M	displacement
100010	0	1	01	000	111	0000 0010

MOV [BX+02H], AX



Örn:

SUB [3342H], SI

opcode	D	W (1 bit)	MOD	REG	R/M
100100	0	1	00	110	110

1001 0001 0011 0110
9 1 3 6

9136

Örn:

Sıra	Komut	Operand		Adres Türü
1	MOV	AX	1122H	2 1
2	SUB	SS:[0007]	BH	2
3	ADD	[DI]	DH	4
4	ADD	[BP+SI+4]	AL	7
5	MOV	SP	DX	3
6	MOV	CX	5891H	1
7	SUB	[0007H]	CH	2
8	MOV	BX	4455H	1
9	MOV	DS	BX	3
10	MOV	[BP-03H]	DX	5

AL AH BL BH CL CH
15H 05 00H FFH 22 0A
22 11 55 44 91 58

DL DH
32 05

Stack Segment

DI SI
0002 0003

SP BP
0001 0004
0532

Data Segment

DS
4455

1 Hemen Adresleme

2 Doğrudan Adr

3 Yazmaç Adr

4 Yazmaç Dolaylı Adr

5 Taban (Base) Mod Adr

6 İndeks Mod Adr

7 Taban İndeks Adr

22
6

16
F F
5 8
B 8

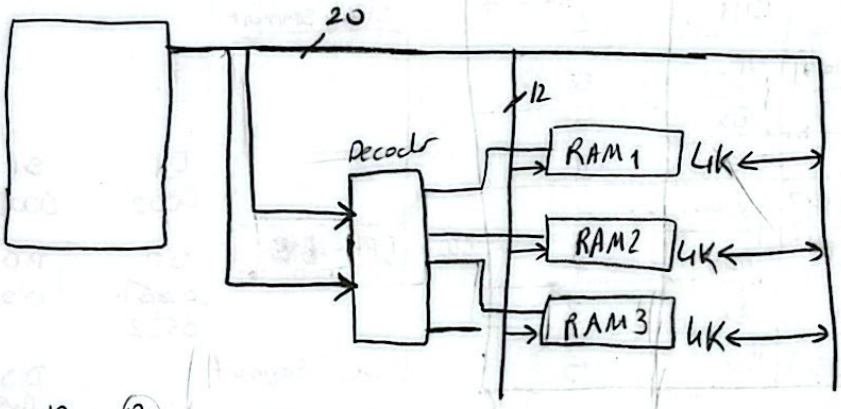
$2^{10} = 1024$ 2^{12} 4096

SORU: 4k üç tane RAM için bellek organizasyonu yapınız

A19	A18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RAM 1

02FFF
02000
01FFF
01000
00FFF
00000



2^3 2^{10} 2^{13}

8K iki tane RAM için bellek organizasyonu 00000H başlangıç adresini kullanarak yapınız

00000 H

1 1 1 1 1 1 1 1 1 1 1 1 1
F F 1F

01FFF H
02000 H
3FFF H

8K

8k

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

RAM 1 8K

S.Reg				
CS	1234H			
DS	1453H	4455H		
SS	1A2FH			
ES	4000H			

I.Reg				
DI	0002H			
SI	0003H			

P.Reg				
SP	0001H	0532H		
BP	0004H			

GPR			
AL	15H	22H	
AH	03H	11H	
BL	00H	55H	
BH	0FH	44H	
CL	A2H	91H	
CH	0A	58H	
DL	32		
DH	05		

DATA Segment (1453)

0000	03	
0001	12	
0002	25	2A
0003	06	28
0004	63	
0005	02	
0006	00	
0007	FF	A7

Stack Segment

0000	03	
0001	12	
0002	25	
0003	4F	
0004	06	
0005	02	
0006	00	
0007	FF	F0

Data Segment (4455)

$$\begin{array}{r} FF \\ -\,0F \\ \hline F0 \end{array} \qquad \begin{array}{r} 25 \\ +\,05 \\ \hline 2A \end{array} \qquad \begin{array}{r} 0004 \\ +\,0003 \\ \hline 0007 \\ -\,4 \\ \hline 0003 \end{array} \qquad \begin{array}{r} 06 \\ +\,22 \\ \hline 28 \end{array} \qquad \begin{array}{r} {}^{15}\not{F}\ {}^{15}\not{F} \\ -\,58 \\ \hline A7 \end{array} \qquad \begin{array}{r} 0004 \\ -\,0003 \\ \hline 0001 \end{array}$$

1. MOV AX 1122 : AX registerına $\overset{H}{11}\overset{L}{22}$ değeri atanır,
2. SUB SS:[0007] BH : Stack Segmentin 0007 adresinden BH registerının değeri çıkarılır
3. ADD [DI] DH : DI registerının içindeki adres Data Segmentte aranır ve DH eklenir.
4. ADD [BP+SI-4] AL : (BP+SI-4) Data Segmentteki adrese AL registerının değeri eklenir
5. MOV CX 5891H : $\overset{H}{58}\overset{L}{91}$H değeri CX registerına atanır
6. MOV SP DX : DX'in içerisindeki değer $\overset{0532}{}$ SP registerına atanır
7. SUB [0007H], CH : Data Segmentindeki 0007 adresli bölümünden CH registerının değer çıkarılır.
8. MOV BX 4455H : BX registerına $\overset{H}{44}\overset{L}{55}$ değeri atanır
9. MOV DS BX : Data Segmentinin değerine BX register'ının değeri 4455 atanır
10. MOV [BP-03H] DX : Data Segmentindeki [BP-03] adresine DX registerının değeri atanır

Mikroişlemci Hafta-9

Assembly Dili Yapısı

[:Label:] Mnemonic [Operands] [;Comment]

↓
Döngüde
Koşullarda
Kullanılacak

ADD AX, BX

Opcode — Destination Operand — Source Operand

Direktifler Sadece derleyici tarafından algılanan kullanıcıya kolay kod yazmasını sağlayan bileşen

∝Direktifler: Derleyici tarafından algılanan fakat I.S içerisinde bulunmayan komutlardır
Makine dilinde direktiflerin opcode şeklinde bir karşılığı bulunmamaktadır

1. Model Direktifleri: Yazılacak olan kodun büyüklüğünü tanımlar

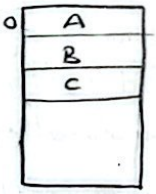
2. Sembol Direktifleri: Geleneksel programlama dilindeki direktiflere denk gelir. Semboller doğrudan veriyi içermezler. Verinin başlangıç adresini içerirler.

3. Segment Direktifleri: Derleyici yazılan kodların düzenlenmesini sağlar

Artık ~~Değişken~~ demiyoruz ⟶ Sembol diyoruz

Karakter ve String Sabitleri: Sembollerin içine string ya da char tanımlayabiliyorsunuz.

```
yazi db 'ABC'
  /
 define byte
```



Sembol Tanımlama

Define Byte (DB), Define Word (DW)

sayi (db) 12H
sayilar (dw) 0012H, FA22H, 2B54H
harf (db) 'A'
kelimeler (db) 'selam'

Lea ⇒ Sembollerin adres bilgisini bir yazmaç içerisine aktarır.
Offset'de aynı işlemi yapmaktadır.

Veri db 5 DUP (12H) →

8 bit ile tanımlıyoruz

12
12
12
12
12

DUP ifadesi (duplicate) veriyi kopyalamaya yarar. Burada 5 kere kopyalamış oluyoruz.

Veri db DUP(12H, 5AH)

8 bit ile tanımlıyoruz

12
5A
12
5A
12
5A
12
5A
12
5A

} 5 tane

Sayi dw 5 DUP(01H)

0001

BYTE PTR WORD PTR
↳ RAM bölgesinden ne kadarlık veri çekileceğini belirler.

01
00
01
00
01
00
01
00

```
Data SEGMENT  ;Data Segmenti açtık
x db 01H, 02H, 03H, 04H, 05H   ;X sembolüne byte boyutunda 5 değer tanımla
Data ENDS

Code SEGMENT
mov ax, @Data   } Buraya gerekli bir alan
mov ds, ax      }
mov bl, x
mov bl, x+1
mov bl, x+2
Code ENDS
```

→

```
Code SEGMENT
mov ax, @Data
mov ds, ax

lea di, x
mov bl, [di]
mov bh, [di+1]
mov cl, [di+2]
CODE ENDS
```

→ Aynısı

```
Code SEGMENT
mov ax, @Data
mov ds, ax

mov ah, 0
mov si, 0

add ah, x[si]
inc si

add ah, x[si]
inc si

add ah x[si]
inc si

add ah x[si]
inc si

add ah x[si]
inc si

hlt
Code ENDS
```

Burada si yerine dx register'ını kullansam x[dx] hata verir. Biz adresleme modlarının içerisinde dx li bir şey görmedik.

Adresleme modlarındaki registerlar SP, BP, DI, SI, BX → BL ve BH kullanılmaz

BP, DI, SI, BX indisleme için kullanılır

Dizi yazdırma 16 bit dw word

```
Data SEGMENT
x dw 01H, 02H, 03H, 04H, 05H
Data ENDS

Code SEGMENT
mov ax, @Data
mov ds, ax

mov ax, 0
mov si, 0

add ax, x[si]
inc si
inc si

add ax, x[si]
inc si
inc si

add ax, x[si]
inc si
inc si

add ax, x[si]
inc si
inc si

add ax, x[si]
inc si
inc si

hlt
Code ENDS
```

FİBONACCİ SERİSİ

```
Code SEGMENT
mov ax, @Data
mov ds, ax

mov ax, 0
mov bl, 0

mov al, 0
mov ah, 1

mov bl, ah
add ah, al
mov al, bl

mov bl, ah
add ah, al
mov al, bl

hlt
Code ENDS
```

∝ Döngülerin ve koşulların yapılabilmesi için mutlaka etiketlerin kullanılması gerekiyor.

∝ Etiketlerin kullanılabilmesi için bazı dallanma komutlarına ihtiyaç var.

Instruction	Description	Condition	Opposite Instruction
JZ, JE	Jump if Zero (Equal)	ZF = 1	JNZ, JNE
JC, JB, JNAE	Jump if Carry (Below, Above Equal)	CF = 1	JNC, JNB, JAE
JS	Jump if Sign	SF = 1	JNS
JO	Jump if Overflow	OF = 1	JNO
JPE, JP	Jump if Parity Even	PF = 1	JPO
JNZ, JNE	Jump if Not Zero (Not Equal)	ZF = 0	JZ, JE

1 1 2 3 5 8 13 21

Fibonacci Serisi Etiket ile Döngü

```
Code SEGMENT
mov ax, @Data
mov ds, ax

mov ax, 0
mov bl, 0

mov al, 0
mov ah, 1

mov cl, 0
FIB:
   mov bl, ah
   add ah, al
   mov al, bl

   dec cl
   jnz FIB

hlt
Code ENDS
```

Dizideki değerleri toplama döngüsü

```
.MODEL SMALL
Data SEGMENT
x db 01H, 02H, 03H, 04H, 05H
Data ENDS

Code SEGMENT
mov ax, @Data
mov ds, ax

mov si, 0
mov ax, 0
mov cl, 5

TPL:
   add ax, x[si]
   inc si
   dec cl
   jnz TPL
hlt
Code ENDS
```

Mikroişlemci: Hafta 10 - 11

Data Segment, Data Ends demenin daha basit bir yolu var

Basit Segment Tanımlama

```
.MODEL SMALL

.DATA
x db 1,2,3,4,5
toplam db ?

.CODE
mov ax, @data
mov ds, ax

mov si, 0
mov cx, 5
mov al, 0

TPL: add al, x[si]
     inc si
     dec cx
     jnz TPL
```

→ `mov toplam, al`

`hlt`

EQU direktifi

S equ 5 dediğimizde

S sembolününü 8 ya da 16 bitlik registera atabiliriz. Hata vermez.

```
S equ 5
mov cx, S   Doğru ✓
mov dl, S   Doğru ✓
```

Yani constant olarak tanımlıyoruz ve istediğimiz her yere atayabiliyoruz.

$ dolar işareti

Dolar işareti RAM gözlerini doldurduktan sonra en sonuncu RAM gözünün ne olduğunu içerisinde otomatik olarak tutan bir değişken

```
.MODEL SMALL
.DATA
x db 1, 2, 15, 33, 66, 5, 3, 8, 9, 25, 34, 22,
Lx equ $-x
ts db 25

U db ?
V db ?

.CODE
mov ax, @data
mov ds, ax
```

↑ dolar işareti şuan buranın adresini tutuyor

TEK İNDİSLERİ BİR YERE ÇİFT İNDİSLERİ BİR YERE TOPLAYAN KOD

```
.MODEL SMALL
.DATA
x db 1,2,3,4,5,6,7,8,9,10
tc db ?
tt db ?

.CODE
mov ax, @Data
mov ds, ax

mov ax, 0
mov si, 0
mov cx, 5

TPL:
    add al, x[si]
    inc si
    add ah, x[si]
    inc si
    dec cx
    jnz TPL

mov tc, al
mov tt, ah

hlt
```

CMP (Compare)

∝ Register içerisini değiştirmeden sadece sonuç üzerinden flagleri kontrol etmek istiyorsak compare komutunu kullanıyoruz.

CMP register, register → CMP AX, BX

CMP register, memory → CMP AX, SONUC

CMP register, immediate → CMP AX, 5
(sabit sayı)

CMP memory, register → CMP SONUC, AX

CMP memory, immediate → CMP SONUC, 5

1. 2.'den büyükse — Sign Flag SF = 0 — pozitif
1. 2.'den küçükse — SF = 1 — negatif — JL SF=1 ise çalışır

JE, JG, JL (Jump Equel) (Jump Great) (Jump Less)

Belli bir sayıdan küçük olanları toplamak

```
.MODEL SMALL
.DATA
x db 1, 2, 15, 33, 66, 5, 3, 8, 9, 25, 34, 22
Lx db $-x
ts db 10
TP db 0

.CODE
mov ax, @Data
mov ds, ax

mov ax, 0
mov bl, 0
mov cx, Lx

KAR:
    mov bl, x[si]
    cmp bl, ts
    jl TPL
    inc si
    dec cx
    jnz KAR

SON:
    mov TP, al
    hlt

TPL:
    add al, bl
    inc si
    dec cx
    jz SON
    jmp KAR
```

```
.Code
mov ax, @Data
mov ds, ax

mov cx, 0
mov si, 0
mov ax, 0
mov CL, Lx
KAR: mov ah, x[si]
     CMP ts, ah
     JGE TOP
     inc si
     dec CL
     JNZ KAR
SON: mov Tp, dL
hLt

TOP: add al, ah
     inc si
     dec CL
     jz SON
     Jmp KAR
```

indis değişkenler	
si	bp
di	sp
bx	

Dizi içindeki elemanlar belli bir sayıdan büyükse U'da değilse V'de toplansın.

```
.MODEL SMALL
.DATA
x db 1,2,15,33,66,5,3,8,9,25,34,22
lx equ $-x
ts db 25
U db ?
V db ?

.CODE
```

```
mov ax,@Data
mov ds, ax

mov ax, 0
mov si, 0
mov cx, 0
mov bx, 0

mov cl, lx

KAR: mov al, x[si]
     CMP al, ts
     jg TPU
     jump TPV

SON: mov U, bh
     mov V, bl

hlt

TPU: add bh, al
     inc si
     dec cl
     jz SON
     jump KAR

TPV: add bl, al
     inc si
     dec cl
     jz SON
     jump KAR
```

TEST : Test instructionu aslında AND yapıyor tıpkı compare'in aslında SUB yapması gibi ve registerın içini değiştirmiyor.

```
0000 0001
0000 0101
---------
0000 0001
     tek
```
> TEST ile AND'le

```
      0000 0001
TEST  0000 1000
---------------
      0000 0000
      çift
```

Bir sayıyı 01H maskesi ile TEST ettiğimiz zaman, sayının tek mi çift mi olduğu anlaşılır.

Çift sayı ile 01 maskesi TEST edildiğinde sonuç 0 olduğundan ZF=1 (ZF aktif) olur
Tek sayı ile 01 maskesi TEST edildiğinde sonuç 0 olmadığından ZF=0 olur

AND register'a kaydeder
TEST register'a kaydetmez

TEST Kullanımı

```
.MODEL SMALL
.DATA
A db 8
B db 3
.CODE
mov ax, @Data
mov ds, ax

test A, 01H
test B, 01H

HLT
```

flags
ZF: 1 (çift)
0000 1000
0000 0001 TEST
0000 0000

flags
ZF: 0 (tek)
0000 0011
TEST 0000 0001
0000 0001

Sayıların çift olduğunu kontrol edip toplama yapmak

```
.MODEL SMALL
.DATA
x db 1, 2, 15, 33, 66, 5, 8, 9, 25, 34, 22
Lx equ $-x
C db 0
T db 0
.CODE
mov ax, @Data
mov ds, ax

mov si, Lx
DON: mov al, x[si]
     test al, 01H
     jz TEK
     jmp CIFT
SON: HLT

TEK: add T, al
     inc si
     cmp si, Lx
     jz SON
     jmp DON
CIFT: add C, al
     inc si
     cmp si, Lx
     jz SON
     jmp DON
```

PROC

> Prosedür. bizim programlama dillerindeki fonksiyonlara denk geliyor.

> Eğer prosedür kullanıyorsanız ana, main prosedür ile diğer prosedürleri ayırarak yazmanız gerekmektedir.

```
.MODEL SMALL
.DATA
.CODE
main proc far
    mov ax, @Data
    mov ds, ax

    mov ax, 0
    mov bx, 0

    call F1
    mov cx, bx
    HLT
main endp

F1 proc
    mov bx, 1255H
    ret
F1 endp
```

LOOP

Loop komutu bizi cx'i bir azaltıp kontrol etme derdinden kurtarıyor. (CMP)

loop L ⟺ dec cx
jnz L

LOOP kullanarak dizideki elemanları toplayan fonksiyon

```
.MODEL SMALL
.DATA
    x db 1,2, 15, 33, 66, 5, 3, 8, 9, 25, 34, 22
    Lx equ $-x
    T db ?

.CODE
main proc far
    mov ax, @Data
    mov ds, ax

    call F1
    mov T, ax
    HLT
main endp
F1 proc
    mov cx, Lx
    mov ax, 0
    lea si, x   Sembol x'in işaret ettiği adresi si'ye atar

    L:
        add ax, [si]
        inc si
        loop L

    ret
F1 endp
```

Mikroişlemci : Hafta - 12

Çalışma Soruları

İndisin tek/çift olmasına göre toplama

```
.MODEL SMALL
.DATA
x db 1, 3, 5, 2, 2, 1, 8, 0, 4, 7
lx equ $-x
T db 0
C db 0
.CODE

mov ax, @Data
mov ds, ax

mov ax, 0
mov si, 0
mov bx, 0

KAR:
      mov al, x[si]
      test si, 01H
      jz CIFT
      jmp TEK
SON:
HLT
CIFT: add C, al
      inc si
      cmp si, lx
      jz SON
      jmp KAR
TEK   add T, al
      inc si
      cmp si, lx
      jz SON
      jmp KAR
```

Mutlak Değerlerin M dizisine dizilmesi

|x(i) - y(i)| toplama

```
.MODEL SMALL
.DATA
x db 1,2,4,5,7,1

y db 3,1,5,2,2,4
len equ $-y

M db len DUP(0)

.CODE

mov ax, @Data
mov ds, ax

mov si, 0
mov ax, 0

DON: al, x[si]
     ah, y[si]
     cmp al, ah
     jl NEG

Pozitif:
     mov bl, al
     sub bl, ah
     mov M[si], bl
     inc si
     cmp si, len
     jz SON
     jmp DON

NEG:
     mov bl, ah
     sub bl, al
     mov M[si], bl
     inc si
     cmp si, len
     jz SON
     jmp DON

SON:
  HLT
```

Sıralama Algoritması

```
.MODEL SMALL
.DATA
x db 1,3,5,2,2,1,8,0,4,7
Lx equ $-x-1

.CODE
main proc far
   mov ax, @Data
   mov ds, ax
L0:
   mov ax, 0
   mov cx, Lx
   mov si, 0
   mov dl, 0

L:
   mov al, x[si]
   mov ah, x[si+1]
   call swp
   inc si
   loop L

   cmp dl, 0
   jnz L0

HLT
main endp

swp proc
   cmp al, ah
   jle not_swap
   mov [si+1], al
   mov [si], ah
   inc dl
not_swap:
   ret
swp endp
```

ÇARPMA ve BÖLME A ve C yi çarp

ÇARPMA

```
.MODEL SMALL
.DATA
A equ 15
C equ 4
Z dw 0
.CODE
main proc far
    mov ax, @Data
    mov ds, ax

    P0:
        mov al, A
        mov ah, C
        mov cx, 0
        mov si, A

    mov cx, A
    P1:
        add Z, ah
        loop P1

main endp end
```

BÖLME PROSEDÜR KULLAN

```
.MODEL SMALL
.DATA
X equ 53
B equ 7

Z dw 0

.CODE
main proc far
    mov ax, @Data
    mov ds, ax

    mov ah, X
    mov al, B
    call bol

    hlt
main endp
bol proc
LO:
    cmp ah, al
    jl PO
    sub ah, al
    inc bx
    jmp LO:
PO:
    ret
bol endp
```

reg, reg
reg, mem
MUL

Çarpılan
AL veya Ax

Çarpan
BL, Cl, DL
BX, CX, DX

MUL 01H gibi bir ifade yanlış

Multipication
MUL: Çarpma yapıyor ama AX (akümülatör) yazmacı ile çarpıyor

```
mov ax, 08H
mov bl, 03H
mul bl
```

Bu multipication komutu otomatik olarak yazdığımız register (bl) ile ax'i çarpmakta ve sonucu ax'in içine yazmaktadır.

bl 8bit olduğu için burada al ve bl çarpılacaktır ve sonuç ax içerisine yazılacaktır.

```
mov ax, 1227H
mov dx,  17H
mul dx
```

Bunun sonucu [dx ax] yazmacı üzerine yazılmıştır.

```
mov ax, 1203H
mov dl,  2
mul dl
```

Bunun sonucu | ax | H: 00 | L: 06 | olacaktır

Faktöriyel KODU

```
.MODEL SMALL
.DATA
X equ 6
S dw 0

.CODE
mov ax,@Data
mov ds, ax

mov bl, X
mov ax, 1

CARP:
    mul bl
    dec bl
    jnz CARP
mov S, ax
hlt
```

DIV

reg, reg

reg, mem

Bölünen
AL veya AX

Bölen
BL, CL, DL
BX, CX, DX

```
MOV AL, 27H ; 39 (Decimal)
MOV DL, 17H ; 23
DIV DL
```

DL, AL'yi bölecek, AL'de bölüm, AH'da kalan olur

Sonuç

AH: 9 AL: 27

AH: 10 AL: 01

Kalan Bölüm

```
39 | 23
-23 | 1
 16
```

X SAYISI ASAL MI?

```
.MODEL SMALL
.DATA
X equ 53
y equ X/2^26

FL db 1
```

FL 1 asal
FL 0 asal-deil

```
.CODE
main proc far
    mov ax, @Data
    mov ds, ax

    call asal-mi

    HLT
main endp
```

```
asal-mi proc
    mov ax, 0
    mov bx, 0
    mov cx, 0

    mov bl, y

P1:
    mov ax, X
    div bL
    cmp ah, 00H
    jz asal_deil
    dec bL
    cmp bL, 01H
    jz asal
    loop P1

asal_deil:
    mov FL, 0
    ret

asal:
    mov FL, 1
    ret
asal-mi endp
```

İÇ İÇE DÖNGÜLER PUSH! POP

Satırları topla

```
.MODEL SMALL
.DATA

x db  1,2,3,4,   5,6,7,8,   9,10,11,12
Lx equ 3
Ly equ 4
T db Lx dup(0)
.CODE

mov ax, @Data
mov ds, ax

mov ax, 0
mov bx, 0
mov cx, 0
mov si, 0
mov di, 0

mov cx, Lx

L0:
    mov ax, 0
    push cx
    mov cx, Ly

    L1:
      add al, x[si]
      inc si
      loop L1
    mov T[di], al
    inc di
    pop cx
    loop L0

HLT
```

PUSH ve POP komutu sadece 16 bit registerlar ile çalışır

KESMELER

> Kesmeler Assembly dili ile kullanıcının etkileşime girdiği kısım diyebiliriz.

> Bir program icra edilirken kesme gönderilirse programın akışı orada durdurulur ve gönderilen kesme doğrultusunda işlemler yapılır.

Yazılımsal kesme olarak INT21 dediğimiz DOS komutlarını çağıran bir kesme var.

```
.MODEL SMALL
.DATA
   msg db "Hello World $"

.CODE
main proc far
   mov ax, @Data
   mov ds, ax
   mov dx, offset msg      ; mesajın adres bilgisi dx'e atanır
   mov ah, 9               ; DOS fonksiyonlarında 9: write string to the STDOUT
   int 21H                 ; fonksiyonu çalıştır

   mov ah, 4CH             ; int 21H
   int 21H
```

MİKROİŞLEMCİ : Hafta-13

Bir dizinin max elemanı

Palindrom

Dizi ortalaması

```
.MODEL SMALL
.DATA
   msg db "Bir sayı giriniz: $"

.CODE
main proc far
   mov ax, @Data
   mov ds, ax

   mov dx, offset msg
   mov ah, 9
   int 21H

   mov ah, 01H             ; yazdığımız karakter ASSCI olarak al'ye yazar
   int 21H

   mov ah, 4CH
   int 21H
main endp
```

div equ ile çalışmaz
div sabit sayı ile çalışmaz

equ bir değişkeni sabit sayı yapar

Matris içinde histogram 1, 2, 3 sayılarına sahibiz

```
.MODEL SMALL
.DATA
xdb 1,2,3,2, 3,3,2,1, 2,1,3,2
Lx equ 3
Ly equ 4
H db 3 dup(0)
.CODE
main proc far
```

Diagonal Toplam

```
.MODEL SMALL.
.DATA
x db 1,2,3,2, 3,3,2,1, 2,1,3,2, 2,1,3,2
Lx equ 4
Ly equ 4

.CODE
main proc far
```

Dışarıdan 5 sayı alacağız. ASCII dönüş
x'in içerisine dizeceğiz.

```
mov ah, 01H
int 21
mov x[si],al
```
} 5 defa yapmalıyım

```
.MODEL SMALL
.DATA
  x db 5 dup(0)

.CODE
main proc far
mov ax, @Data
mov ds, ax

mov ax, 0
mov si, 0
mov cx, 5

L0:
    mov ah, 01H
    int 21
    and al, 0FH
    mov x[si], al
    inc si
    loop L0
```

numerik sayılar 30 - 39

F0 ile maskelersem sonucu
30 ise numerik

Dışarıdan karakter al → 5' değilse sayı
→ 5' ise sonlandır

MAKROLAR

> Makroları assembly dilinin kütüphanesi olarak düşünebilirsiniz.

> Programın başına emu8086.inc kütüphanesini koyduktan sonra

```
PRINT string : String bastır
PAINTN string : ?
PUTC char : Karakter bastır
GOTOXY col,row : Col,row olarak verilen yerlere götür
CURSOROFF : Cursor kontrolü
CURSORON
   :
```

bu makroları kullanabiliyoruz. Prosedür ve Makrolar farklıdır.

```
include 'emu8080.inc'
.model small
.data
.code
main proc far
    mov cl, 10
    mov ch, 0
    mov bl, 0
    mov bh, 8

    mov al, 2
    mov ah, 8

    gotoxy ch, cl
```

```
GOTOXY MACRO col, row
   :
   :
   :
ENDM
```

Derleyici kod parçasını buraya gömüyor.

String Sayma 'ab' 'ac' say s kapat ys db 0AH, 0DH, '$'
newline

```
include 'emu8086.inc'

.model small
.data
cnt db 0

.code
main proc far
mov ax, @Data
mov ds, ax

mov ax, 0
L:
  mov ah 01H   } dizaridan karakter okuma
  int 21H

  and al, 0FH
H:
  cmp ah, "a"
  je A

  cmp ah, "s"
  jne L
HLT

A:
  mov ah 01H
  int 21H
  and al, 0FH

B:
  cmp aH, "b"
  je C

  cmp ah "c"
  je C
  jmp H

C:
  inc cnt
  Jmp L
```

Şekil Oluşturma

bir sayı : 9
bir sembol: /

```
/
//
///
////
 :
```

```
.MODEL SMALL
.DATA

msg1 db "Bir sayı giriniz $
msg2 db 0AH, 0DH, "bir sembol giriniz $"

.CODE
main proc far
  mov ax, @Data
  mov ds, ax

  lea dx, msg1
  mov ah, 09H
  int 21H

  mov ah, 01H
  int 21H
  and dl, 0FH
  mov bl, al
  lea dx, msg2          bl: sayı
  mov ah, 09H           bh: sembol
  int 21H

  mov ah, 01H
  int 21H
  and al, 0FH
  mov bh, al
  mov dx, bl
  L0:
     mov cx, bl

     L1:
        gotoxy(cx, bl)
        putc bh
        Loop L1
     dec bx
     jnz L0

hlt
```