

\* Bilgisayar sistemi 3 parçadan oluşur.

- 1) Merkezi İşlemci Birimi (Central Processing Unit, CPU): kısaca mikroişlemci olarak isimlendirilir. Bilgisayar içindeki bütün aktivitelerde beyin gibi koordinasyon yapar.
- 2) Hafıza : Program komutları ve veri esas olarak burada saklanır.
- 3) Giriş / Çıkış (Input / Output, I/O) Birimleri: Bilgisayara islemek üzere bilgi girişini sağlayan ve sonra sonuclarını çıkaran, bilgisayar sevce bilimi olarak

billinen parçalarıdır.

\* CPU'yu bulunduran entegre devre cipine mikroişlemci; mikroistemci, hafıza ve I/O kisimlarının tamamını bulunduran bilgisayar Mikrobilgisayar.

## Sayı Sistemleri

## 2. Hafta

- İnsanlar tarafından 10'luk (decimal) sayı sistemi kullanır. İnsanların 10'luk sistemi kullanmasının nedeninin 10 parmağı sahip olmasından ötürü geldiği düşünülmektedir. ( Düşüntür. ☺)
- Bilgisayarlar ise binary olarak 2'lik sayı sistemini kullanır.
- 16'lık kod ve ASCII Kodu da vardır.
- Bilgi sayılarından 2'lik sistemi kullanmanın sebebi 0 ve 1 gibi değerlerin iki voltaj seviyede kullanılmasıdır. ( On yada Off )

ÖR: 25 sayısını 2'lik tabanda ifade ediniz.

$$\begin{array}{r} 25 \\ 24 \end{array} \left| \begin{array}{r} 2 \\ 12 \\ 6 \\ 0 \end{array} \right. \quad \begin{array}{r} 2 \\ 12 \\ 6 \\ 0 \end{array} \left| \begin{array}{r} 2 \\ 3 \\ 1 \end{array} \right. \Rightarrow (11001)_2$$

MSB (Most Significant Bit) = En yüksek değerli bit.  
LSB (Least Significant Bit) = En düşük değerli bit.

MSB: Most significant bit = En yüksek değerli bit.

LSB : Least " " = En düşük " " .

ÖR:  $(110101)_2$  sayısının 10'luksuz sisteme çevrilmesi.

$$\begin{array}{r} 110101 \\ \downarrow \quad \downarrow \quad \downarrow \\ 2^5 \times 1 \quad 2^4 \times 0 \quad 2^3 \times 1 \end{array} \Rightarrow 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 = 53_{10}$$

ÖR:  $(\underbrace{1001}_{9}, \underbrace{1111}_{F}, \underbrace{0101}_{5})_2$  sayısının hexadecimal sayı sisteme çevrilmesi.

$$\begin{array}{r} 1001 \\ 1111 \\ 0101 \\ \hline 9 \quad F \quad 5 \end{array} = 9F5_{16}$$

<u>Hexadecimal</u>	<u>Binary</u>	<u>Decimal</u>
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

ÖRNEK :  $29B_{16}$  Hex  $\rightarrow$  2'lik tabanı çeviriniz

$$0010 \quad 1001 \quad 1011 \rightarrow (\underline{0010} \underline{1001} \underline{1011})_2 = (29B)_{16}$$

$$= 29B H$$

ÖRNEK Decimal koddaki 45 sayısını hexadecimal kodu çeviriniz.

$$\begin{array}{r} 45 \\ 32 \\ \hline 13 \end{array} \quad | \quad \begin{array}{r} 16 \\ 2 \end{array}$$

A red circle highlights the number 13, with a blue arrow pointing down to the letter D at the bottom right.

$$2D = (2D)_{16} = 45$$

ÖRNEK : 6B2 H = (6B2)<sub>16</sub> sayısını decimal kodu çeviriniz.

$$6B2 \Rightarrow 6 \times 16^2 + B \times 16^1 + 2 \times 16^0 = 1714$$

The number 6B2 is shown with arrows indicating the powers of 16:  $6 \times 16^2$ ,  $B \times 16^1$ , and  $2 \times 16^0$ .

İkili Sayılarda Toplama ve Çıkarma

<u><math>A + B</math></u>	<u>Toplam (Sum)</u>	<u>Elde (carry)</u>
0+0	0	0
0+1	1	0
1+0	1	0
1+1	0	1

$$\text{ÖR: } \begin{array}{r} +1 \\ -1 \\ +1 \\ \hline 101100 \end{array}$$

Gitarma işlerini yaparken, ikinci sayının  $2^7$  ye tamlayını alırız ve ilk sayı ile toplazır.

$$\text{ÖR } \begin{array}{r} 11001001 \\ -10011101 \\ \hline \end{array} \quad \begin{array}{l} \text{2}^7 \text{ye tamlayını} \\ \text{Topla} \end{array} \quad \begin{array}{r} 01100010 \\ +1 \\ \hline 01100011 \end{array}$$

$$\begin{array}{r} 11001001 \\ +01100011 \\ \hline 100101100 \end{array}$$

$$\text{ÖR: } \begin{array}{r} 59F \\ -2B8 \\ \hline 2E7 \end{array}$$

ASCII Kodları

<u>Hex</u>	<u>Sembol</u>
41	A
42	B

43 — C

1  
2  
3  
4  
5  
6  
7  
8

5A — z

$$61 \quad - \quad 9$$

b  
c  
d  
e

7A    2

\*  $bit = 0$  veya  $1 \Rightarrow \underline{1 basamak}$

- \* nibble = 0000  $\Rightarrow$  4 basamak 4 bit
- \* 1 byte = 0 000 000<sub>2</sub> = 8 basamak = 8 bit = 2 nibble

\* word = 16 byte = 16 bit = 4 nibble = 2 byte

\* Double word = 32 bits = 2 words = 8 nibbles  
4 bytes.

\* guard word = 64 bits = 64 bytes

1

$$* 1 \text{ kibibyte} = \underline{\underline{2^{10} \text{ byte}}} = 1K = \underline{\underline{1KB}}$$

\*  $1 \text{ Megabyte} = 2^{20} \text{ byte} = 1M = 1MB$

\*  $1 \text{ Giga byte} = 2^{30} \text{ byte} = 1G = 1GB$

\*  $1 \text{ Tera byte} = 2^{40} \text{ byte} = 1T = 1TB$

ÖR: 786 sayısının ikilik ve BCD karşılığı nedir.

$$\begin{array}{r} 786 \\ | \\ 1100010010 \end{array} \rightarrow 2'lik karşılığı$$

$\begin{array}{c} 1100010010 \\ \text{---} \\ 0111 \quad 1000 \\ | \quad \downarrow \\ 0110 \end{array}$

X

$(786)_{10} = (0111 \ 1000 \ 0110) \underline{\underline{BCD}}$

NOT: Binary ile BCD arasındaki fark, 10'luk tabanda bir sayıyı 2'ye bölerken binary halini buluyoruz.

BCD Kodu ise  $10^4$ 'lik tabandaki sayıları

$2^4$ 'lik kodda 4 bit ile ifade ederken bulunur.

ÖR  $25 \rightarrow 2^4$ lik ve BCD = ?

$$\begin{array}{r} 25 \\ \times 2 \\ \hline 125 \\ \times 2 \\ \hline 0625 \\ \times 2 \\ \hline 1 \end{array}$$

$25 = (11001)_2$

$$25 = (0010\ 0101) \text{ BCD}$$

/ \

0010 0101

NOT: Sayının işaretli olduğu söyleyenler ve ilk basamak (MSB)

0 ise sayı pozitif, 1 ise negatifdir.

ÖR:  $(0,6875)_B = (?)_2$

$$0,6875 \times 2 = \underline{\underline{1}}.375 \quad 1 \rightarrow 0,375 \quad (0,1011)_2$$

$$0,375 \times 2 = \underline{\underline{0}}.75 \rightarrow$$

$$0,75 \times 2 = \underline{\underline{1}}.5 \quad 0,5$$

$$0,5 \times 2 = \underline{\underline{1}},0$$

Karılık kismı ortadan  
Valkara Kadar  
2'ye çarpı.

ÖR :  $(\underline{\underline{110}} \underline{\underline{1100}} \underline{11})_2 = (?)_8 \quad (\underline{\underline{663}})_8$

3. Hafta (16.03.2021)

ÖRNEK :  $(35A8DE)_{16} = (?)_8$

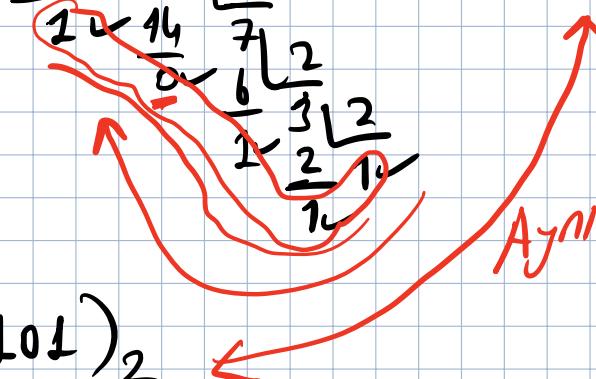
$$(\underline{\underline{0011}} \underline{\underline{0101}} \underline{1010} \underline{1000} \underline{1101} \underline{\underline{1110}})_2 = (15324336)_8$$

1    5    3    2    4    3    3    6

ÖRNEK :  $(35)_8 = (?)_2$

1.yol:  $(35)_8 = 5 + 3 \times 8 = \underline{\underline{29}} \quad \begin{array}{r} 1 \\ 2 \\ \hline 28 \\ 1 \end{array} \quad \begin{array}{r} 2 \\ 4 \\ \hline 4 \\ 2 \\ \hline 2 \end{array} \Rightarrow (\underline{\underline{11101}})_2$

2.yol  $\quad (35)_8$   
 $011 \quad \underline{101} = (\underline{\underline{11101}})_2$



ÖR :  $8 \cdot 2^{24}$  byte kaç MB?

$$\underbrace{2^3}_{} \times \underbrace{2^4}_{} \times \underbrace{2^{20}}_{} = \underline{\underline{2^7}} \times \underline{\underline{2^{20}}} = \underline{\underline{128}} \text{ MB}$$

ÖR:  $2^{20}$  byte kaç kB ?  $\rightarrow \underline{1 \text{ MB}}$

ÖR:  $2^{12}$  byte kaç kB ?  $\rightarrow \underline{\underline{2^2 \times 2^{10} = 4 \text{ kB}}}$

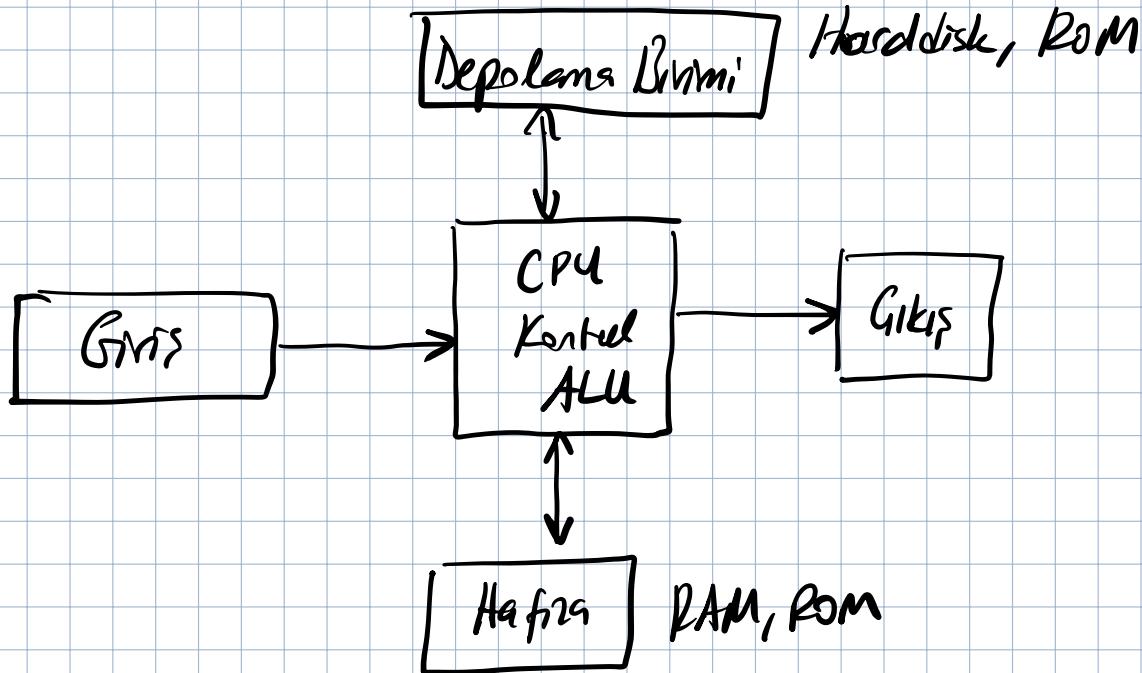
ÖR

$$\begin{array}{r} 1234 \text{ H} \\ \times \quad 22 \text{ H} \\ \hline 2468 \\ + 2468 \\ \hline 26AE8 \checkmark \end{array}$$

## MIKRO BİLGİ SAYARLAR VE MIKROİSLEMÇİLER

Mikrobilgisayar en basit haliyle bir bilgisayardır. Bunu üç temel parçaya ayıririz.

- \* CPU (Central Processing Unit - Merkezi İşleme Birimi)
- \* Hafıza (The Memory)
- \* Donanım (Input / Output Devices) (Giriş / Çıkış Birimleri)



**Mikroişlemci:** CPU'yu bulunduran entegre devre cihapına mikroişlemci denir.

**Mikrobilgisayar:** Mikroişlemci, hafıza ve giriş/cıkış birimlerini bulundurduiği gibi lise mikrobilgisayar denir.

### CPU

→ Bilgisayarlarda değişik birimler arasındaki veri aktığı ve veri işleme görevi yerine getirir.

→ Veri aktığını CPU'nun alt birimi olan "kontrol" birimi yönetir. Hafızadan okunan komutları görür ve komut tarafından

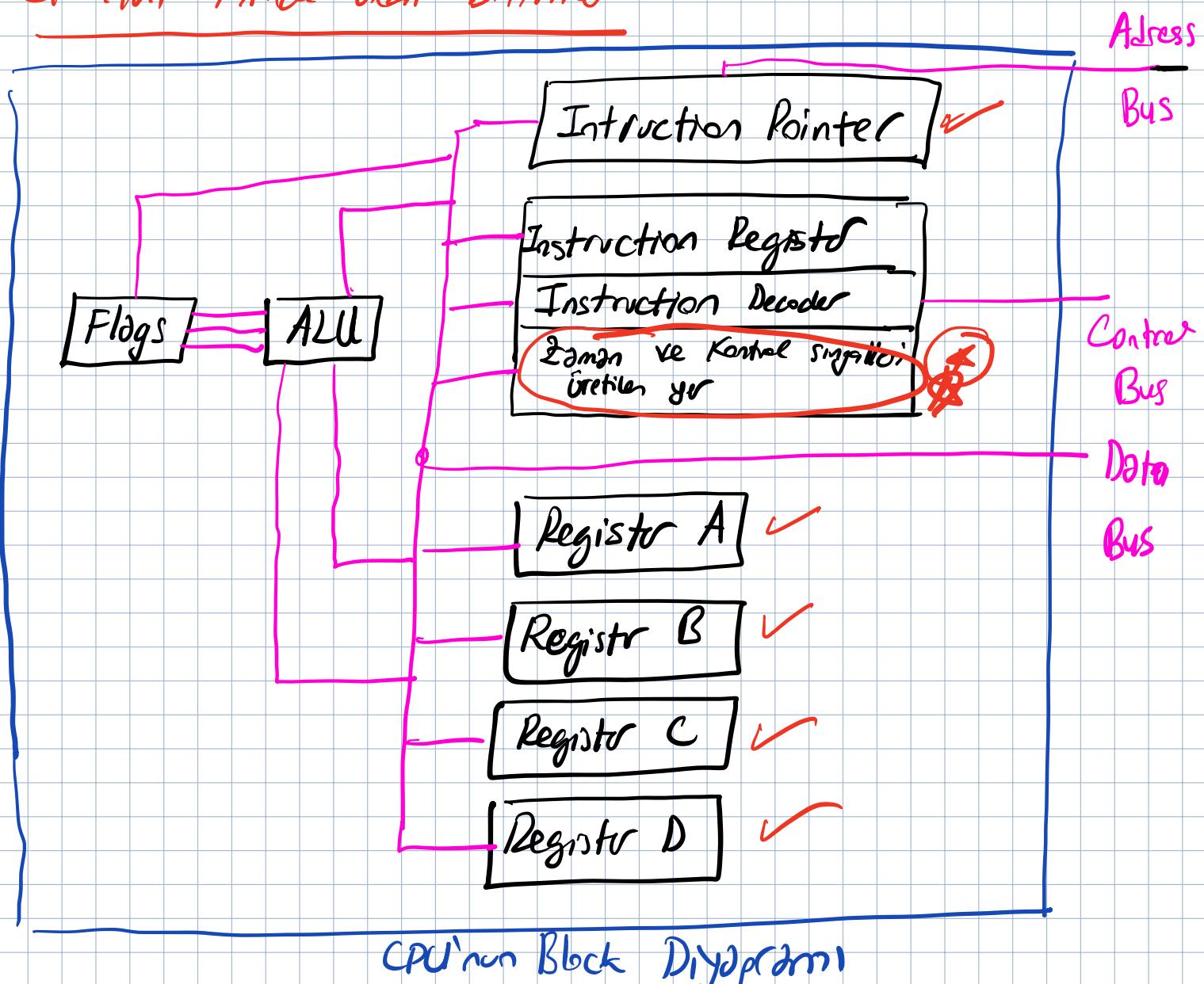
# beşerlenen işlemi yerine getirir ( Execute )

→ Veri işlemininAGO CPU içindeki ALU'da gerçekleştirilir.

→ CPU içinde sunulan işlemler gerçekleştiriliyor

- 1) Sayısal aritmetik işlemler (+, -, /, \*)
- 2) Lojik işlemler (and, or, ...)
- 3) Kontrol

## CPU'nun içinde olan birimler:



- \* → Hafızada bulunan program CPU'ya yapmasını istediyim işlem için komutlar söyle. CPU'nun program komutlarını (Instruction), hafızadan getirmek (fetch) ve olayları yürütmemtedir (execute)
- CPU'lar bilgileri geçici olarak registerlerde depolar. CPU içindeki registerler 8, 16, 32 veya 64 bitlik olabilir.
- CPU'lar ALU biriminin içeri: ALU toplama, çıkarma ve bölme işlemlerinin yanı sıra lojik yani and, or, not gibi işlemleri de yerine getirir.
- Her CPU program sayacına sahiptir. Genelde Instruction Pointer diye isimlendirilir. Fonksiyonu, bir sonraki işlemin konutunun adresini gösterir. İşlemi yerine getirince program sayacı bir artarak bir sonraki konutu adresini gösterir. (Program sayacı = Program counter)
- Konut Gözüçü yani instruction decoder, CPU'ya getirilen komutların anlamını yorumlayan bir sepet sözlük gibidir. Komutun anlamına göre kontrol sinyalleri üretir.
- Program sayacının (sayacı) adres yolu üzerinde istenilen komut okuyacılık ve sıfırçacılık bulucuları yer almaktadır.

→ Denebeli CPU içerisindeki Flags, ALU, Control,

Instruction pointer, Instruction decoder ve registerler yer almaktadır.

## Hafızal (Memory)

CPU'nun dışından enstığı birim "bellek" denir. Bu yerde, harddisk trafiği elemeni deildir. Donanım yanı çevre birimi elemendir. CPU'nun bireysel enstığı trafiği elemeni RAM veya ROM'dur. Programların talimatları ve verilerin ilk olarak yüklenip yer memory'dir. Program komutları ve verilerde olaraka burada saklanır.

### RAM

Random Access Memory  
(Rastgele Erişimli Bellek)

\* Programlar çalışırken bilgisayarlar trafiğinden kullanıcılar geçici trafiği zedir.

\* Bilgisayarlar kapatılınca buradaki veriler kaybolur.

### ROM

Read Only Memory  
(Yazma okunabilir Bellek)

\* ROM'daki bilgi sadece okunabilir. ve gerek kesinleşeyle kaybolmaz.

\* Kullanıcı hafızaya önde; BIOS'tur.

\* PC'nin ilk çalıştığı program

budur.

Input: Dis dünyadan CPU'ya veri transfer eden alt birim  
Veri cihazdır. Giriş ve si'ri msandan gelebilir, cihazın yada  
bir bilgisayardan gelebilir.

Output: CPU'nun dis dünyaya veri transfer eden alt birim

Gıks cihazdır. Cıksımları, yonucular, göstergeler,  
diğer bilgisayarlar veya başka elektronik sistemlerdir.

## BUS

→ CPU'nun trafige ve giriş/gıks cihazlarına bağlantılarını  
"Bus" adı verilen tel sırtları sağlar. Aynı zamanda bilgi-  
sayalarla bilginin bir yerden başka bir yere taşınmasını  
saçır. Her bilgisayar 3 tip Bus'a sahiptir.

- 1) Address Bus
- 2) Data Bus
- 3) Control Bus

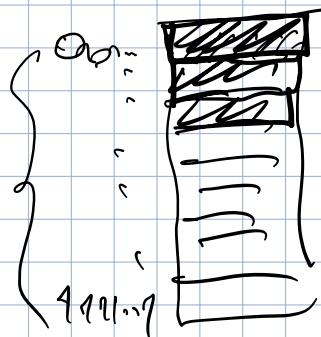
### Address Bus

- Hafıza ve kaynak veinin adresini taşıır.
- Address Bus genişliği sistemin maksimum hafıza kapasite-

sını belirler.

→ Address Bus 16 bit olan bir mikroişlemcide maksimum hafıza  $2^{16} = 2^6 \times 2^4 = \underline{\underline{64 \text{ kB}}}$  'dur.

Address Bus'un genişliği dəlibirken, aynı anda birden fazla işlenen yapılabılabiliridir. Örneğin, adres yolunun genişliği 8 bit olursa bu yolla "2<sup>8</sup>" adet adrese ulaşılabilir.



→ Adres yolu tek yönlüdir.

### Data Bus

→ Mikroişlemci tarafından hafızaya veya çıkış bilimine veya çıkış bilimlerine veri göndermede veya hafızadan veya giriş bilimlerinden veri alımları kullanılır.

→ Komut veya Data olan veiler tasınır.

→ Veri yolu genişliği, bilgisayarın performansı doğrular etür.

→ Intel mikroişlemcilerde veri yolu genişliği 8 bitten 64 byte kadar değişir.

→ Veri yolu iki yönlüdir. ve birden fazla paralel telden oluşan haberleşme konlidır.

## Kontrol Yolu (Control Bus)

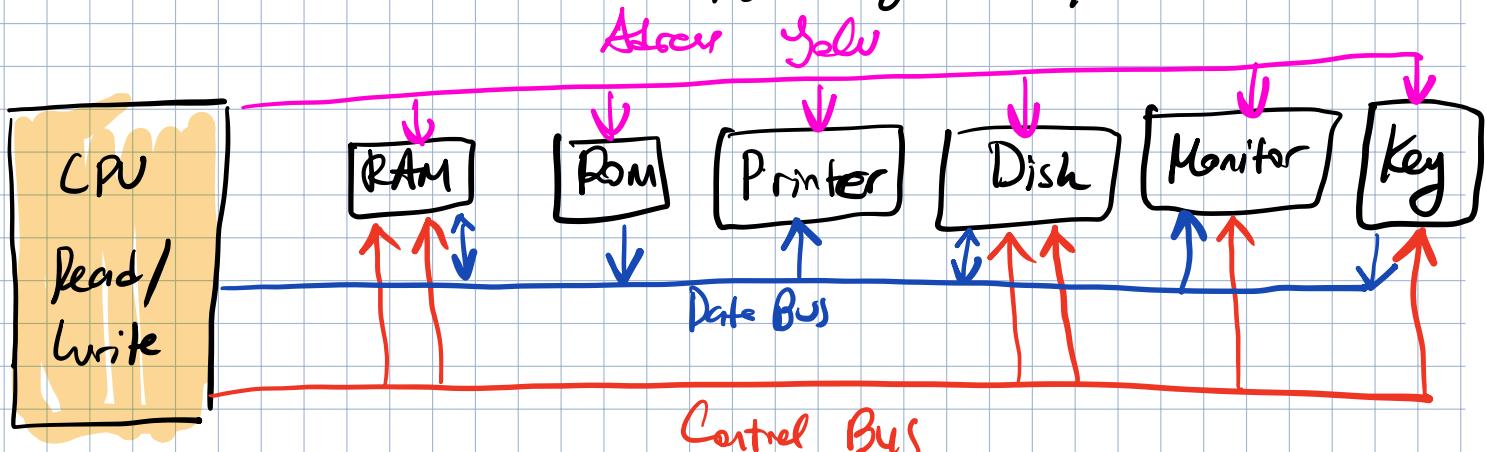
- Yol üzerindeki bir device hafıza adresini I/O birimlerinden biri mi olduğunu Control Bus tarafından tespit eder. Read/Write sinyallerini sağlamak için kullanır.
- CPU'nun I/O veya hafızaya bilgi göndermek mi yoksa onlardan bilgi almak istediyimi belirler.
- Control Bus, 4 kontrol yolu sinyallerinden birini aktif eder:
  - \* Memory Write
  - \* Memory Read
  - \* I/O write
  - \* I/O Read.
- Kontrol Bus tek yönlüdür.

\* **Decode:** Komut çözme

\* **Execute:** İşlemi yerle getirme

\* **Instruction:** Komut

\* **Fetch:** Komutların hafızada getirilmesi / Komut okuma



# Internal Organisation of Microcomputer (Bir bilgisayarın iç organizasyonu)

- Registrler, CPU içerisindeki hafıza bloklarıdır. Ne kadar çok olursa kapasite artar. Genişlikleri CPU ile değiştir.
- İşlemcide transistor sayısı, antenler, fikstür ve tazalar. Mikroişlemeler saat frekansıyla çalışır. Saat frekansı ne kadar yükselse işlemeler o kadar hızlı yapılır.

23.03.2021

4. Ders

A, B, C, D registerleri sırasıyla bir CPU

düşünelim. CPU tarafından, 21H değer A registerine yerleştirilecek, daha sonra A registerine 42H ve 12H sayıları ilave edilecektir. Her sayı bir deponun A registerine koplayacak olur CPU komutu 1011000 (B0H), A registerındaki bir depona koplayacak olan komut 00000100 (04H), ise iki deponun nasıl olur?

Tiplen işlen

Ked

Veri

21H'yi A registerine yaz

B0H — 21H

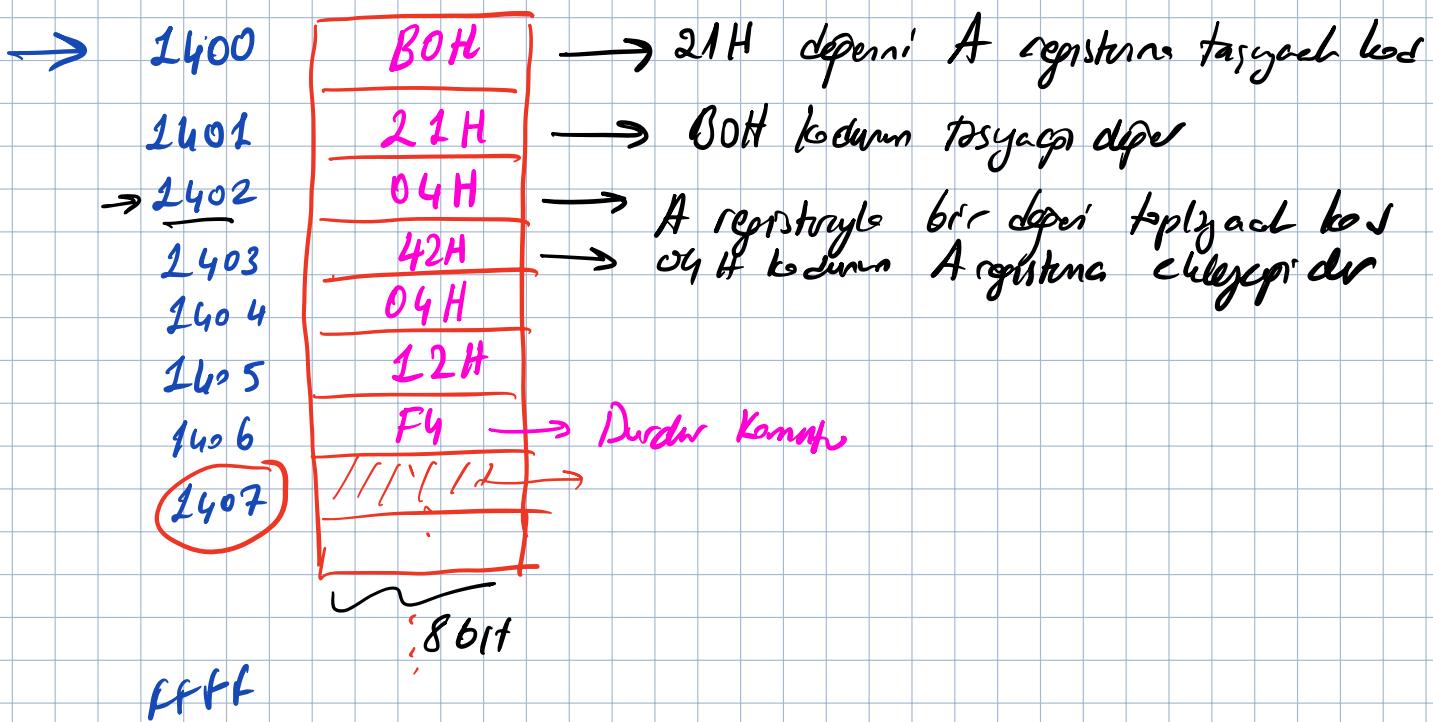
A Registeriyle 42H'yi topla

-04H — 42H

" " 12H "

04H — 12H

## Hafıza Adresi Hafıza Adresinin İşlevi



0000 - FFFF .. (16bitlik işlemenin max kapasitesi)

- CPU Program çağrısı (pointer) 0000H - FFFFH arasıında dğrabilir. Bütün örneğimizde 1400H'ye ayarlanmıştır. Yerine getirilmesi gerken ilk 15, kodu gösteren yerin belirlenmesidir.
- Program çağrısı, ilk komutu 2. byteyle yüklenildikten sonra CPU işlemi gerne getirmeye hazırlanır.

→ CPU, 1400H'ın adresi yoluna yazar ve gönderir. Hafıza anteşi; CPU'nun aktif etkisi "oku Singalı"nın yerini belirler, ve gestesinden hafıza yer 1400H'daki byte'i ister. 1400H hafıza gözündeki Paralel BO'yu yoluyla yole tutturur ve CPU'ya gonderebilir. CPU, BO komutunu komut çözücü yardımıyla çözer. Komutun türüne göre CPU'nun A registerine bir sonrakı hafıza förmüeli

byte'ı giderkeni dilar. Bu nedenle kontrol devresi "bunu yapma" komutu verir. Bu sayede 1400H hafıza girişindeki 21 H depezi A regitronu hizis dura regitr girisini kapatırak ve ramın dönmeye gelmesini sağlar. Bu nedenle 21 H depezi CPU'ya gelinice döndürden A'ya gider

→ Bir komutun tamamlanmasından sonra program sayacı bir sonraki yerine getirilecektir komutun adresini işaretler. 1402 H adresi, adres yolu ile sonraki komutun okunma işlemleri için gondereir. 1402H hafıza girdinden gelen 04H kodu okunur. Kod fırıldaktır sonra CPU bunu bir sonraki adresi gösterdeki veriyi A Registronun içeriği ile toplayıcısını dilar. 42 H depezi A regitronun içeriği ile toplar ve ALU'dan çıkan toplamı sonus A regitrona yarar.

## Bölüm 4

- Mikroislemci Ailesinin Tanitimi
- Pipeling \*
- Cache

## 80X86 Mikroişlemci Aksinin Kısa Tarihi

→ Intel, 8086 mimarisini 1978 yılında üretmiş. Bu 16 bitlik mikroişlemci önceki nesil mikroişlemeler olan 8080 / 8085 serisi mikroişlemcilere göre 6 katlı bir gelişimdir.

### 8086

- 1 MB ( 20 bit Address Bus)
- 26 bitlik Data bus
- Pipelined İşlemci

### 8080 / 8085

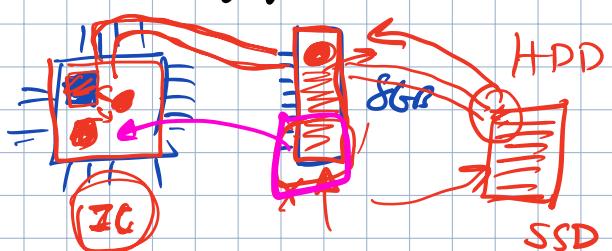
- 64 KB
- 8 bitlik Data bus
- Non-pipelined.

Internal Data Bus: Mikroişlemcinin kendi içinde iletişimini sağlar.

External Data Bus: Mikroişlemcinin dışarıda yapıtlarda iletişimini sağlar.

### Virtual Memory:

RAM'ın yetmezse olsun durumda harddiskin bir bölümünün mikroişlemci tarafından veri girdi - çıktı olarak



kullanılmazdır. Yani harddiskin bir bölümünün içinde  
tarafından RAM gibi algılmazdır.

**Real Mode:** Maksimum 1MB belleğe kadar olan hızıdır.

**Protected Mode:** Kullanıcıdan kaynaklanan bir hata dan  
dopasız istenmeyen durumlara koruma sağlar. Bu

modus daha yavaştır. Çünkü belleğin 16MB lik kısmını kullanır.

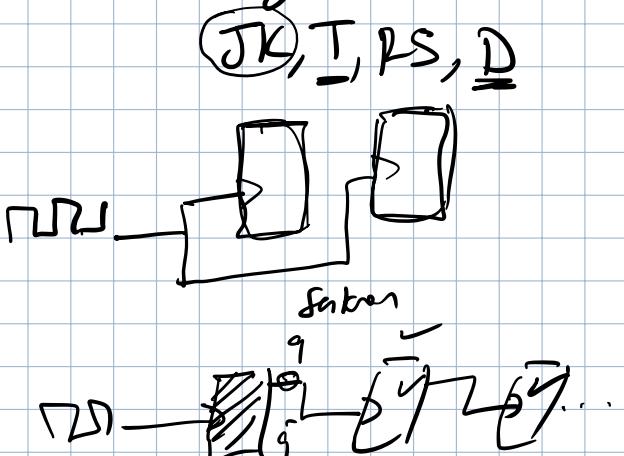
**Pipelining:** 8085 mikroişlemcisinde CPU'ya zyni andası hem  
komut okuma (fetch) türünde komutun gerekni yerde getirme  
(execute) imkanı verir. Burada dikkat edilmesi gereken komu-  
tur okunma ile yerde getirme süreçlerini zyni olmak  
zorunda olmayacağıdır.

Buradaki ilk mesele

zyni andası yapma imkanıdır.

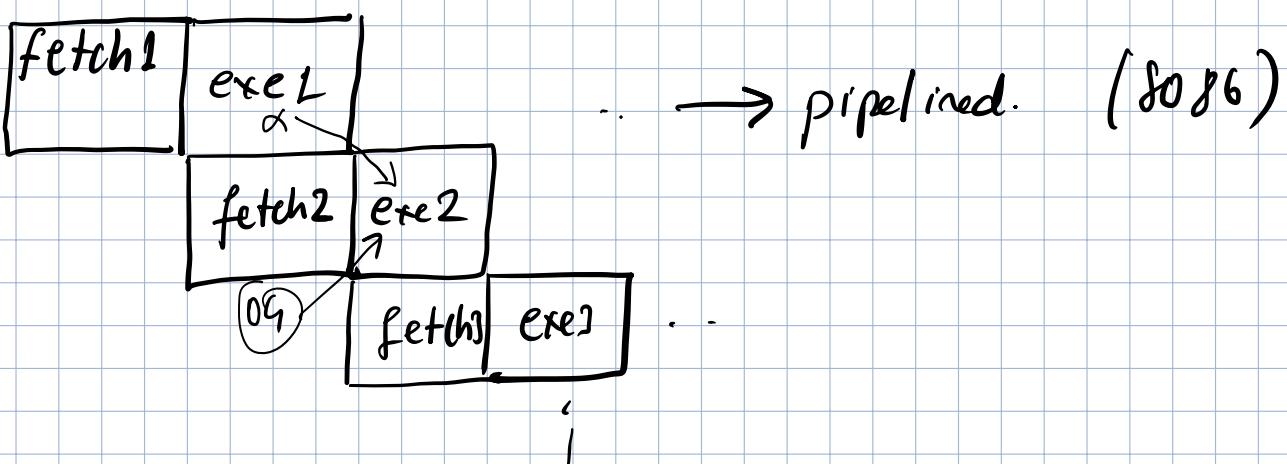
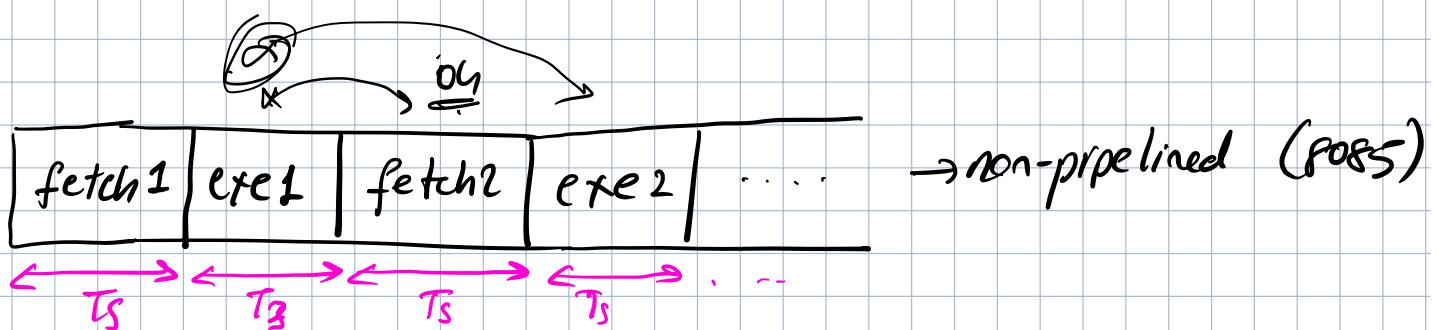
→ Intel 8086 ve 8088

mikroişlemci lerin iş yapısını değiştirecek, 2 kisma ayıracak  
pipelining kuramını tanımlamıştır.

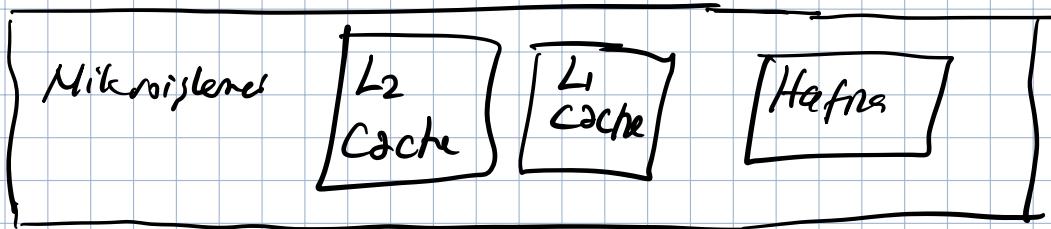


→ Birimler : Execution Unit (EU) (Yürütme birimi) , BUS Interface Unit (BIU) ; Bu 2 kisim aynı anda çalışır. Ell konutları yorumlar ve yürütür. BIU, konutun kodunu okur, operant okur, veri saklar, I/O ilişkileri gibi işlərini yerine getirir.

Bu durum, BIU'ın EU'dan önce olup olmadığı gösterir. Bu nedenle BIU, buffer yani kicik hafizaya sahipdir.

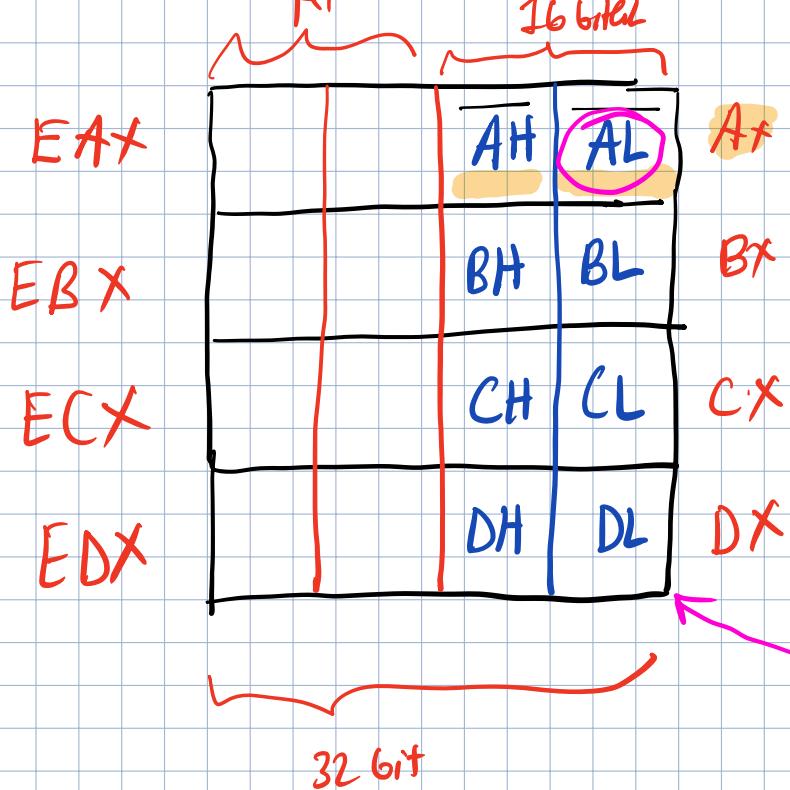


Cache: Mikroişlemci ile hafıza arasında kalan 4KB veya daha büyük bir hafıza elemanıdır. Mikroişlemci bir işlemi yapıp hafızaya gönderirken bu cache'da hafıza elemanında problem sonucunu hafızada tutar. Bir kırış işlem sonucu, işlemci bu işlem sonucuna ihtiyac duyuyorsa hafızadan gizlilik gönlünde daha hızlı olunca da hafıza elemanları yani cache'den alır.



### 8086 Mikroişlemcisinin Registerleri

- CPU'da registerler bilgiyi, geçici olarak saklamak amacıyla kullanırlar. Bu bilgi 1 veya 2 byte'lik bir veya birden fazla bit gibi olabilir.
- 8086'da genel amaçlı registerler 16 bitlik yada 8 bitlik olarak kullanılır.



AH AL  
~~FFFF~~ ~~1111XXXXXX~~

\* 8 bitlik registerlarn diziş. sırası :  $\rightarrow$  AH, AL  
 BH, BL

D7 D6 D5 D4 D3 D2 D1 D0

\* 16 bitlik registerlarn diziş. sırası  $\rightarrow$  AX, BX, CX, DX

D15 D14 . . . D1 D0

$\rightarrow$  A, B, C, D registerleri genel ismeli Registerlerdir.

Genel ismeli Registerler 16 bit olurken AX, BX, CX ve DX

seklinde, 8 bitlik gösterimde ise AH, AL, BH, BL, CH, CL, DH

ve DL seklinde ifade edilebilir.

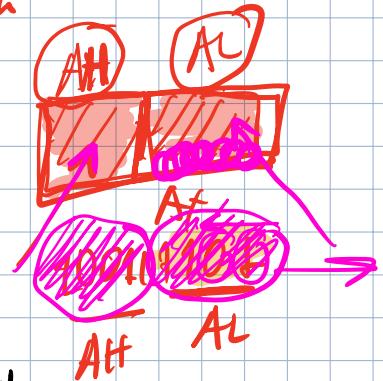
\* AX = Accumulator

\* BX = Base Addressing Register

\* CX = Counter Loop Operation

16 bitlik

\* DX = Data I/O operation



E AX: 32 bit uzunlukonda bir registerdir. İlk 16 bit

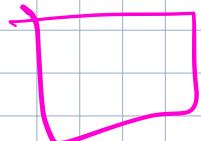
uzunlukundaki kismına AX adı verilir. Bu kısma

verdi içerisinde 8'er bitlik 2 kisma ayrılr. İlk 8

bitlik kısma AL, 2. 8 bitlik kısma AH adı verilir.

Bu registerler çarpma / bölme gibi işlemlerde özel

anakı olarak kullanılır.



E BX: Genel amaçlı olarak kullanılır. Özel amaçlı olarak

memory adresinde offset bilgisini içinde barındırır.

E CX: String (döngüsel) işleri ve loop'larında counter

olarak kullanılır.

E DX: Genel amaçlı bir registerdir. Word boyutunda çarpma /

bölme olarak kullanılır.

30/03/2021

## 8086 Register List

<u>Kategorie</u>	<u>Bits</u>	<u>Register Abb.</u>
Genel	16 8	$AX, BX, CX, DX$ $AH, AL, BH, BL, CH, CL, DH, DL$
Pointer	16	$SP$ (Stack Pointer) - $BP$ (Base Pointer)
Index	16	$SI$ (Source Index) $DI$ (Destination Index)
Segment	16	$CS$ (Code segment) $DS$ (Data segment) $SS$ (Stack segment) - $ES$ (Extra segment)
Instruction	16	$IP$ (Instruction Pointer)
Flag	16	$FR$ (Flag Register)

## ASSEMBLY PROGRAMLAMA YAZMA

- "Makine Dili" I ve O'ları içeren programdır. Assembly dilleri makine kodu komutları gibi (mnemonics) hatırlatıcılarla dillerde kullanır.
- mnemonics, kodların yeri tutan kılmların klasik hatırlamasını sağlayarak kolaylaştırır.
- Assembly dili bir dilsel serviseli dilidir.
- CPU'nu direkt iş yapısıyla başlatır.
- Assembler, Assembly dilini makine diline çevirir.

### Assembly Dili Programa

- Assembly deli program, assembly diliyle yazılmış komutların birbirinden satırlardan oluşur. Komut hatırlatıcı kod ve 2 adet işleniciden (özenle işlenen yapılar da dahil) oluşur.

### "MOV" Komutu

MOV : Kaynak taki işlenen değer hedefe kopyala

hedef, kaynak

Mnemonics

ÖR: (8 byte)      MOV CL, SS : CL=SS

MOV DL, CL : CL'in içeriği DL'ye kopyalandı. DL=CL=SS

(16 bitler)  $Mov CX, 468FH$ ;  $CX'ye 468FH sayisi kopyalandi.$

$CX = 468FH$   
 $CH = 46$        $CL = 8F$

$Mov AX, CX : AX = CX$

$Mov BX, AX$

$Mov DX, BX$

$Mov SI, DI$

$Mov DS, SI$

**NOT:** Veri, **Fbp - Registerler** hanesi, bütün registerlerde kullanılabilir.

\* → Kursak ve Hedef Registerlerin boyutları uyumludur.

\* → Veri sadece synapt olmayan Registerler arasında  
direkt kullanılabilir.

→ Veri **segment** registerlerne direkt tayinmez. \*

**ÖR:**  $Mov BX, 14AFH$  ✓  $BX = 14 AFH$  ( $BH=14, BL=AF$ )

$Mov SI, 2345H$  ✓

$Mov DI, 2238H$  ✓

$Mov CS, 2A3FH$  X "Code Segment'e <sup>öncə</sup> yaramaz."

Bunun yapması için önce normal  
bir Register'a yazılırlar, ardından sonra  
meite aktarlanır. " \*

MOV DS, CS ✓

MOV RR, BX X "Buyruk registerlerin ve tasınması."

MOV DS, 16AF X

MOV AX, 1204H }  
MOV SS, AX

MOV BX, 02H; BX = 0002H ✓  
BH BL

MOV AL, 12JH X ;  
—  
8bit      12bit  
—

→ ADD Konutu:

M. ~ Operands

ADD Destination, Source ; Kaynaktaki Veriyle hedefteki veriyi toplar.

ÖR MOV AL, 24H → AL = 24H

MOV DL, 11H → DL : 11H

ADD AL, DL → AL = AL + DL

ÖR MOV CH, 24H

ADD CH, 11H → CH = CH + 11 = 35 H

## Program Segmentleri

Segment: 64 KB'a kadar ( $2^{16} = 64\text{ KB}$ ) hafıza容量unu sahip, 16 ile bölen birler dörtlərə sahip kismıdır. (16 ile bölen birlerin sayı sonu 0 olmaz)

Assembly dili programlama üç terel segmentlərdir.

- \* **Code Segment:** Program kodlarının (komutlarının) içeriği kod segmentidir.
- \* **Data Segment:** Program trafindən kullanılan varyi sabitlərənək kullanılan veri segmentidir.
- \* **Stack Segment:** Dilçiyi geçici olarak saklandığı källərdir. Yığın segmentidir.

## Lojiksel ve Fiziksel Adres

1) **Fiziksel Adres:** 20 bitlik adresdir. ve Address Bus

İle tanınır. 8086 için 00000-FFFFH'e kadar may by kadar adreslene yoper.

$$2^{20} = 1\text{ MB} \quad \text{dresalene yoper.}$$

2) **OFFSET ADRES:** 64 KB'lik segment içindəki yeri gösterir. 0000-FFFFH sahip olduğunu da bilir.

3) Lojik Adres : Segment adresi ve OFFSET adresini içen.

### Kod Segment Tekni Adresleme

Komutun lojik adresi, kod segment ve IP (Instruction pointer)'ını içen.

CS : IP ile gösterilir.  
Segment → Offset  
Adresi gösterir.

Lojik Adres =  $\left[ \underbrace{250}_{CS} : \underbrace{95F3}_{IP} \right]$  gibiols.

Not: Bir programı yerine getirmek için 8086 komutu kod segmentler yapılır.

\* Fiziksel adres, Kod segmentin 1 hexadecimal sola kaydırılması ve IP ile toplamasyla bulunur.

de: CS:IP =  $\underbrace{250}_{CS} : \underbrace{95F3}_{IP}$  H , Fiziksel adres=?

\* CS'yi sola kaydır.: 2500

IP, ik top'a : + 95F3

FA-adres: 2E5F3

offset : 95F3

Lojik Adres: CS:IP=2500:95F3

DR CS:24F6H, IP = 6J4AH

Lojik Adres, offset adres, Frikkel adres, Kod segmenti 21t  
Sırrı ne ist sırrı nedir?

Lojik Adres: CS:IP = 24F6:6J4AH ✓

Offset : 6J4A ✓

Frikkel Adres: 24F60

$$\begin{array}{r}
 + 6J4A \\
 \hline
 2B2AA
 \end{array} \quad \checkmark$$

Kod Segmenti 21t sırrı:

$$\begin{array}{r}
 24F60 \\
 + 0000 \\
 \hline
 24F60
 \end{array} \quad \checkmark$$

Kod Segmenti 21t sırrı:

$$\begin{array}{r}
 24F60 \\
 + FFFF \\
 \hline
 34F5F
 \end{array} \quad \checkmark$$

## Data Segment Address

Veri segmenti (DS) ve offset degeri kullanır. 8086'da

BX, SI, DI offset adresini tutmamakta kullanır.

ÖR: DS: 7FA2H ve OFFSET: 438EH ise F.A., L.A.,

DS üst. Sıvı, alt sıvı nedir?

Lojik Adres: 7FA2 : 438EH

$$\begin{array}{r}
 \text{FA} : \overbrace{7FA2}^{\text{+1}} \\
 + 438E \\
 \hline
 \text{83 DAE} \checkmark
 \end{array}$$

$$\begin{array}{r}
 \text{DS, üst sıvı: } \overbrace{7FA2}^{\text{+}} \text{ } 0 \\
 + \overbrace{\text{FFFF}}^{\text{+}} \\
 \hline
 \text{8FA1F} \checkmark
 \end{array}$$

Alt sıvı: 7FA2 ✓

## Data Segment Kullanımı

5 sayının toplamını yapalım:

I YOL: MOV AL, 00 H → Burada 15'i silip

→ ADD AL, 25 H ✓      } resetleyen:

ADD AL, 12 H ✓

ADD AL, 15H ✓

ADD AL, 1FH ✓

$$AL = 25H + 12H + 15H + 1FH + 23$$

→ A DD AL, 23 H

II. YOL:

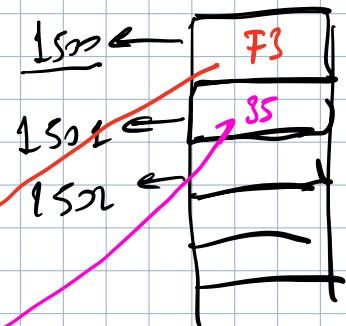
$$\begin{aligned} DS: \underline{\underline{0200}} &= 25 - \\ DS: \underline{0201} &= 12 - \\ DS: 0202 &= 15 - \\ DS: \underline{0203} &= 1f - \\ DS: \underline{0204} &= 2B - \end{aligned}$$

Bu programda obtain dep-  
trilmeni içindeki  
değisme gel yoldur.

$\leftrightarrow$  MOV BX, 200H  
 $\leftrightarrow$  ADD AL, [BX]  
 $\leftrightarrow$  INC BX  
 $\leftrightarrow$  ADD AL, [BX]

$\leftrightarrow$   
 $\leftrightarrow$

ADD AL, [BX]  
 :



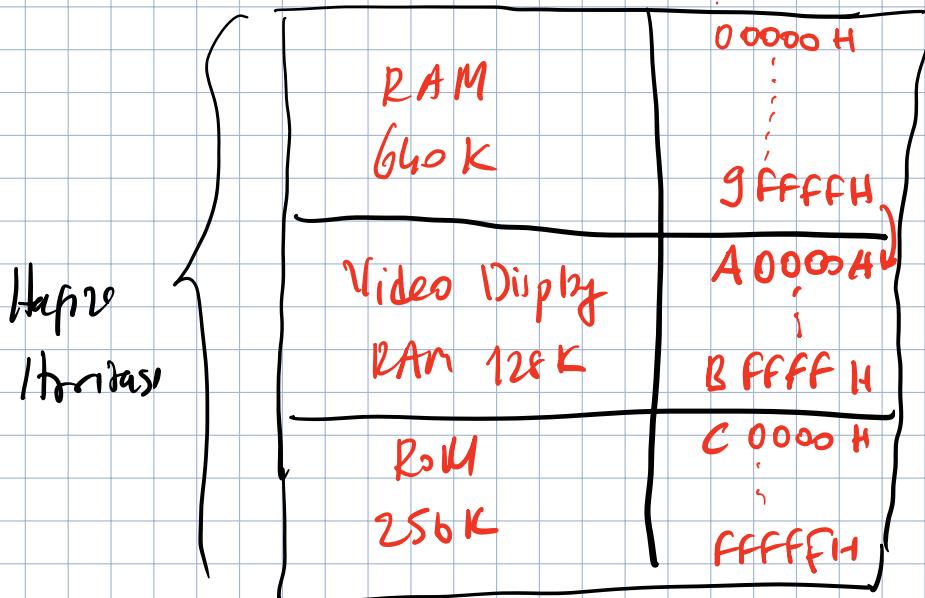
MOV AX, 3SFJH

MOV [1501], AX       $\rightarrow$  ilk 8biti [1501], 2. 8biti [1502]'e  
 8bit 16bit      2. 8biti [1502]'e  
 2. 8biti [1503]'e  
 2. 8biti [1504]'e

$$\left. \begin{array}{l} DS: 1500 = FJ \\ DS: 1501 = 35 \end{array} \right\}$$

2 offset kullanıldı !

## IBM PC'nin Hafızası Hakkında



PC'lerde bir program başlatıldığında ilk olarak DOS ekranı buna RAM'e yaradılır.

## STACK: PUSHING AND POPING OPERATION

Stack: CPU'nun bilgileri geçici olarak saklandığı okunur/yazılır hafızanın bir kısmının CPU, sadece sayıda register'e sahip olduğunu için bu alanı ihtiyac vardır. Stack'in dezavantajı, yavaş olmasıdır. (En fazla 256) Stack RAM'in içindedir ve buraya erişmek registerlarda karşılaştırıldıktan çok zaman alır.

## Stacklara nasıl erişir?

Stack Segment ve stack pointer registerleri Stack'e erişmek için kullanılır. Bu registerler, Stack'e ulaşmak için turhane'ye bir komut uygulamadan önce stack'e yiklenmesi olmalıdır. 8086'daki her register (segment registerleri ve stack pointer)

hərmiş) stack'te saklanabilecek re stack'tan CPU'ya

geliş getirilebilir.

(\*) CPU registerlerinin, stack'te saklanması "push"  
icerisinde CPU registerlerin yüklenmesi "pop" denir.

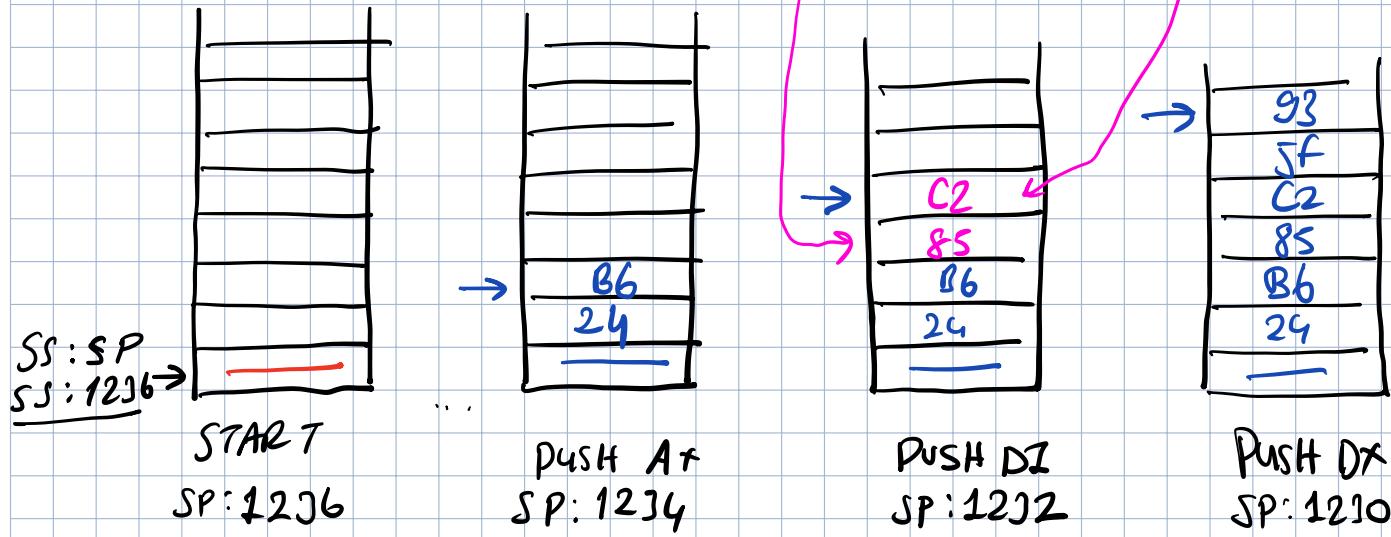
ÖR: SP: 1236, AX = 2hB6, DI = 85C2 ve DX = 5F93

olarak verilmiştir. Aşağıdaki konuları salıstırıldığında  
durumları incelyelim.

PUSH AX ✓

PUSH DI ✓

PUSH DX ✓

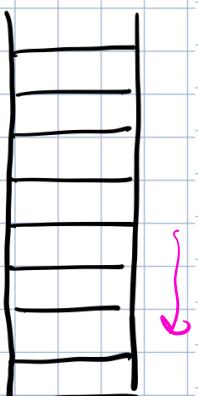
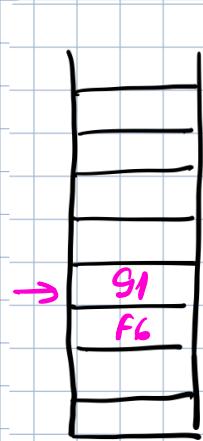
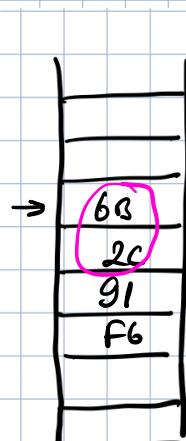


QR: SP: 18FA olup, 4 bytelik kodlar salıverse;

POP CX

POP DX

POP BX



START  
SP = 18FA

POP CX  
CX = 1423  
SP = 18FC

POP DX  
DX = 2C6B  
SP = 18FE

POP BX  
BX = F691  
SP = 1900

→ Stack, LIFO (Last in First Out, son giren ilk çıkar)

Mutlakla gelir.

→ 16 bitlik olmalıdır. Yani PUSH → AX şeklinde olmalıdır. PUSH AL veya PUSH AH şeklinde asla yaramaz. ~~→~~

→ PUSH ve POP işlemleri asla karıştırma. Aslında program göster.

Stack'ta Logik ve Fiziksel Adres:

→ Kod segment ve data segmentin logik ve fiziksel adresini bulma yöntemini ile öğrenir.

ÖR SS: 3500 H .

a) Frikuel adres?

SP: FFFEH

b) Stack'in bit sonu nedir?

c) " üst " " ?

d) Logik Adres?

a) 35000  
FFF E

b)  $35000 + 0000 = 35000$  Altisins.

$$\begin{array}{r} \\ + \\ \hline 44FFE \end{array}$$

F.A ✓ c)  $35000 + FFFF = 44FFF$  ✓

d) SS: SP = 3500 : FFFF ✓

Tek bir frikuel adres bir çok farklı logik adresin  
adı ola bilir mi?

Evet olabilir.

Örnek;

Logik Adres

FA

1000 ; 5020

15020

1500 ; 0020

15020

1502 ; 0020

15020

1400 ; 1020

15020

## THE FLAG REGISTER (FR)

→ 16 bitlik Registerdir. Sayı işlemleri kullanılmaz.

→ 6 bayrak durum bayrapıdır. Herhangi bir konus ugurlan-  
diktan sonra durumu değiştirebilir. Bunlar; CF, PF, AF,

ZF, OF ve SF 'dir.

→ Z bayrah ise kontrol bayrapları. Buralı TF, IF, ve DF 'dir.

R	R	R	R	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF
---	---	---	---	----	----	----	----	----	----	---	----	---	----	---	----

R: Reserved

U: Undefined

**CF : ( Carry Flag )**; Bir aritmetik işlemede, toplamdan sonraki  
eldeyi veya çıkarmada sonraki öðrenci belirtir. Yani  
8 bit ikili  $\text{d}7$ 'yi 16 bitlik ikili  $\text{d}15$ 'i kontrol etmeyi.

**PF ( Parity Flag )**; Dati işlemelerde sade dieçik deðerlilikli  
byte'ta parity kontrolü yapar. Çift sayıda lojik olmak  
"1" varsa; parity bayrağı "1", olursa daimılarda ise  
bayrah 0'dır. Hvis "1" bulunmaması da çift sayıda "1"  
bulunması denektr.

**AF ( Auxiliary Flag )**: Yapılmış işlemin sonucunda 6 bit  
parisyonları 3-4 pozisyonda olan toplamdan sonraki  
eldeyi veya çıkarmadan sonraki öðrenci belirtir.

**ZF (Zero Flag)**: Bir aritmetik veya lojik işlem sonucunda

sonuç "0" olduğunu belirtir. Eğer  $ZF=1$  ise, sonuç 0'dır. Döngülerde kullanılır.

**SF (Sign Flag)**: Bir toplama veya çıkarma işleminden sonra sonuç aritmetik işaretini belirler. Eğer  $SF=1$  ise, işaret negatif "0" ise, işaret pozitiftir.

**OF (Overflow Flag)**: Tanımlı, işaretli sayıların toplanması veya çıkartması sonucu oluşan bir durumdur.

**TF (Trap Flag)**: Eğer  $TF=1$  ise tüm devrede hata takip işlemi devreye girer. Bu zaman dökümleri içermekte söz konusu bir komut yerine getirilir.

**IF (Interrupt Flag)**:  $IF=1$  olupanda, mikroişlemci dışarıdan gelen kesme isteği cevap verir.

**DF (Direct Flag)**: String komutları yürütülürken DI veysa SI słaycılardan değiştirilmesi veya değişikliklerin işleminin sırasının kontrol eder.  $DF=1$  ise değiştirir,  $DF=0$  ise değiştirmez.

ÖR: MOV BH, 38 H      ADD BH, 2F H      } Kod parçası geleceklere bize rakkam  
derum ne olur?

$$BH = \underline{38}$$

$$BH = \underline{38} + \underline{2F} = 67H$$

$$\begin{array}{r}
 & \text{+} \\
 \begin{matrix} & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{matrix} & \hline
 & 0 & 1 & 1 & 0 & 0 & 1 & 1
 \end{array}$$

CF: 0 (Elden yok)

PF: 0 (5 tane 1 var)

AF: 1 (dördüncü)

geniş eldevar

ZF: 0 (Sonuç sıfır değil)

SF: 0 (pozitif, dizi 0)

OF: 0

## 8086 Adreslene Modları

→ CPU verilere deprek yollarından ulaşabilir. Bu yolların adreslene modlarıdır. 8086'da 7 tane adreslene modu vardır.

- 1- Register
- 2- Immediate
- 3- Direct
- 4- Register Indirect
- 5- Based Relative
- 6- Indexed Relative
- 7- Based Indexed Relative

1) Register Adreslene Modu: Registerler arasında veri aktarması meydandır.

Registerlerin boyutları eşittir.

MOV BX, DX ✓

MOV ES, AX ✓

ADD AL, BH ✓

MOV AL, CX ✗

2) Immediate Adreslene Modu: Source byte veya word bir sayı, hedef memory veya register olan adreslene meydandır. (Segment registerler ve flag registerler hariç hepsi direkt veri yonlabilir.)

MOV BX, 1234 H ✓

ADD AL, 40H ✓

MOV DS, 1234 H ✗

3) Direct Adreslene Modu: Byte veya Word Uzunlukundaki  
veri, memory veya registerler arasında kopular.

→ MOV AL, [2400]

MOV AL, 99H

MOV [3518], AL

DS: 2400 → 2'de gelen girer ismini

AL'ye kopyalar.

DS: 3518

99

4.) Register Indirect Adresleme Modu: Verim hafızadaki adresi register tarafından tutulmak tâdus. SI, DI, ve BX registerleri offset adreslerini tutan pointerlar olarak kullanılır. DS ile birleştirilerek 20 bitlik fizikal adresler elde edilir.

MOV AL, [BX] → DS: BX'in içeriğini AL'ye kopolar.

MOV CL, [SI] → DS: SI' içeriğini CL'ye kopalar.

MOV [DX], AH → AH'in içeriğini DS: DI' hedefinin içine kopalar.

5) Based Relative Adresleme Modu: BX ve BP

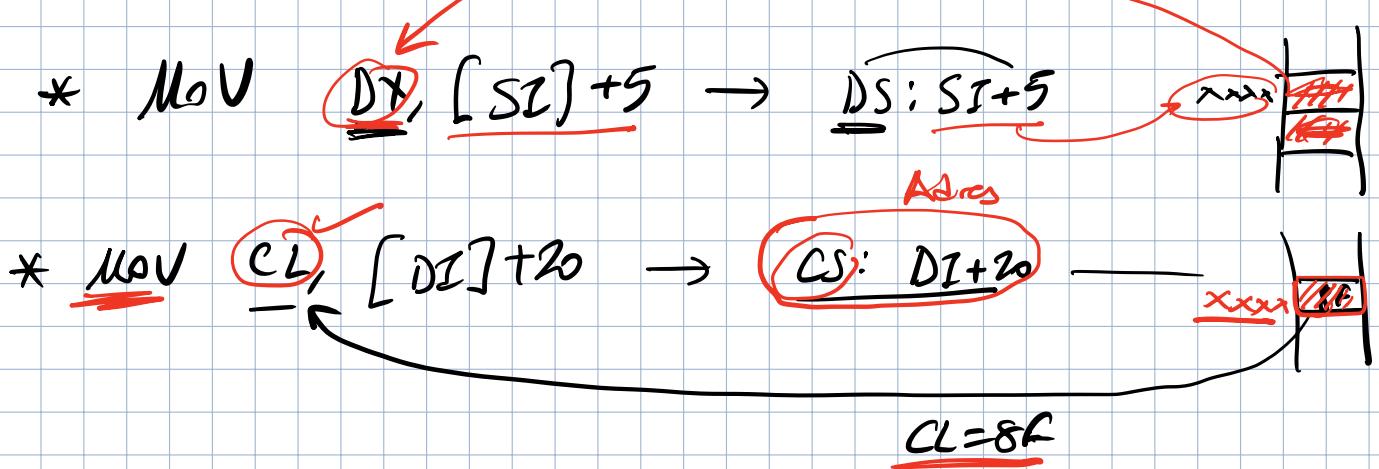
her registerden olarak kullanılır. Bu mode yer deplasmanı

yapıldıkları sonra efektif adres bulunur.

MOV CX, [BX]+10 → DS: BX+10 ve DS: BX+11' in  
icerisini CX'e kopalar.

Moz: MOV CX, [BX]+10 = MOV CX, [BX + 10]  
= MOV CX 10 [BX]

6) Indexed Relative Adreslene Modu: Indexed Relative Adreslene modunda, based relative adreslene modu gibi gelisir. Sadece burada offset adresini tutan register SI ve DI'dir.

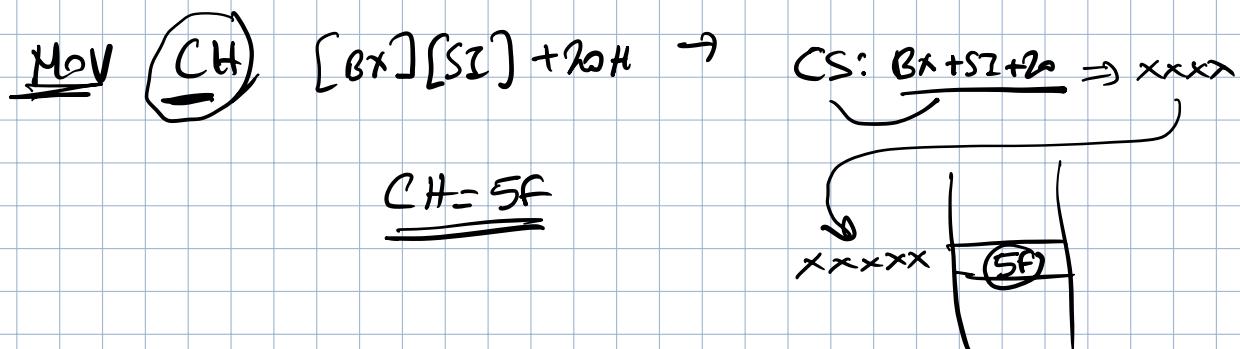


## 7) Based Indexed Adreslene Modu:

Suc 6' daki adreslene modlarının birleşiminde olusur. Bir base registeri ve index register birlikte kullanılır.

$\text{Mov } CL, [Bx][DI] + 8$

↓      ↓  
based      indexed



$$\text{NOT: } \text{Mov } CL, [Bx][DI] + 8 = \text{Mov } CL, [Bx + DX + 8]$$

Ques: DS : 4500

BX : 2100

SS : 2000

SI : 1486

AX : 2512

DI : 8500

BP : 7814

a) Mov [BX]+20, AX

b) Mov [SI]+10, AX

c) Mov [DI]+4, AX

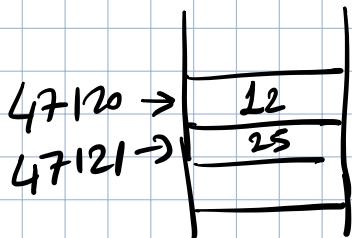
d) Mov [BP]+12, AX



$$DS: \underline{BX+20} = 4500: 2120$$

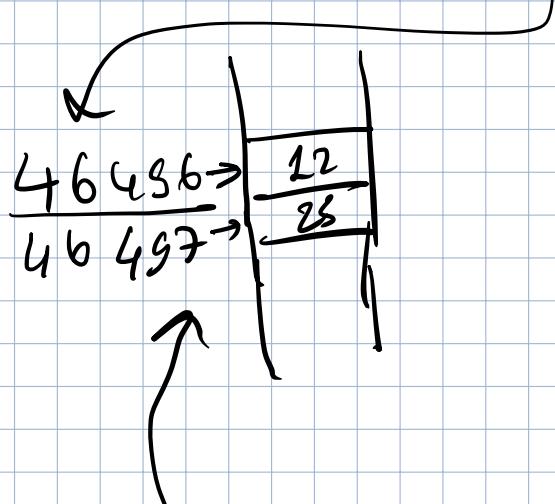
$\xrightarrow{2100+20}$

$$\begin{aligned} FA = & 45000 \\ & + 2120 \\ \hline & 47120 \text{ H} \end{aligned}$$



b)

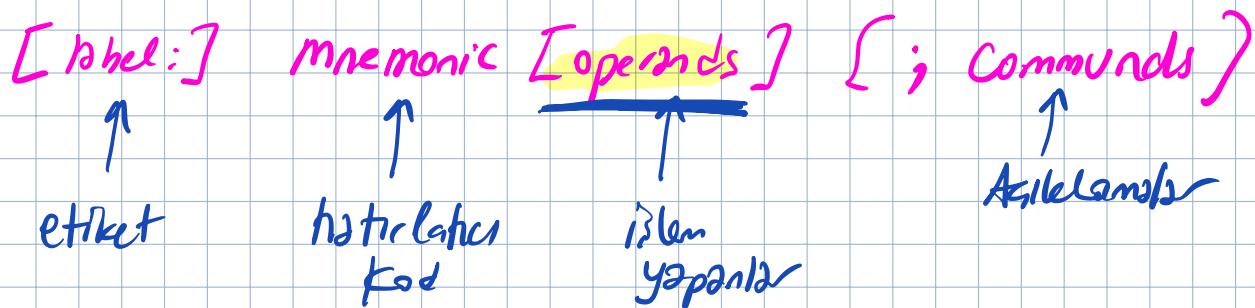
DS: SI+10 → FA : 46496



$$\begin{array}{r} 20000 \\ 7817 \\ + 12 \\ \hline \end{array}$$

## Directives And Sample Program

→ Direktifler Assemblerde, assembly dilindeki komutları makine diline nesil sevireceği konusunda yön gösterir. Assembly dili komutları ekadolu gibi 4 tane türden: etiket, hatırlatıcı kod, işlem yapınlar ve açıklamalar.



→ Direktifleri kullanın bir programda öncelikle segmentler tanımlanır. Yani bir program parçası hangi segmenti ifade ediyorsa onun altına yazılmalıdır. Veri tanımlarının data segmenti içinde programla ilgili kodlar veya kod segmentin altına yazılır.

→ DB b DVP(?) segmentin altında tanımlanan bu direktif 64 bytelik hafıza tahsis eder.

→ Label kimi, bir kod satırına isim ile göndermeye sahip. Eğer label, bir işlen koduna göndermede bulunuyorsa,

Label sonuna ":" koyulmalıdır.

### Temel Tanımlamalar:

Temel tanımlama komutları ile kullanılacek hafızanın boyutuna karar verilir. Kullanılacak direktifler;

**SMALL, MEDIUM, COMPACT, LARGE, HUGE, TINY**

İşbu komutlardır.

**Model Small:** Bu direktifle hafıza modeli olarak small seçilir. En çok kullanılır. Bu model kod için 64KB, Data için 64 KB hafıza ayırar.

**Model Medium:** Data için max 64 KB, Kod için 64 KB'ı, 2x64'ü.

**Model Compact:** Data 64 KB', 2x64'ü. Kod max 64 KB'dır.

**Model Large:** Her iki side 64 KB', 2x64'ü. Fakat tek bir data 64 KB', verebilir.

**Model Huge:** Her iki side 64 KB', 2x64'ü. Tek bir data tipi 64 KB', verebilir.

**Model Tiny:** Kod ve data max 64 KB'dır. COM dosyaları ile kullanılır.

**Segment Tanımlanması** 4 segment vadır. (DS, CS, ES, SS).

Segmentlerin kesc adı;

".CODE" ".DATA" ".STACK" → .STACK 64, direktif.  
Stack'in 64 byte'lik hafızaya ayırr.

**STACK** Stack segmentinin başlangıcıyı belirler.

**DATA** Data " " " "

**CODE** Kod " " " "

→ Data segment 3 satırda tanımlanır. DATA1, DATA2, SUM.

\* Bu türün hizmini DB (define byte) olarak tanımlanır.

Data Segmentte tanımlanın veilen etiketleri (DATA1, DATA2, SUM) ile kod segmenti onları ulaşılabilir.

→ Hafızada 2 byte'lik yer ayırmak istiyorsak DW direktifini kullanırız.

→ DATA1 ve DATA2'ye bir değer verilmesine karşın SUM'a değer vermez.

→ .CODE kod direktifinden sonra segmentin ilk satırı PROC direktifidir. Prosedür direktifidir. Prosedürün bitirilmesi mutlaka ENDP ile yapılır.

- PROC ile ENDP ifadesi zyni etikete sahip olmalıdır.
- PROC direktifi FAR veya NEAR fonksyonu oluşturur.
- Yordamız kodu ekranı yordamınıza tanımlı kod segment ve stack segment de personele sahipdir. Fakat DS'yi mutlaka bireim programınız tarafından belirtmemelidir.
- MOV AX, @DATA yordamında stack segmentin başlangıç adresini gösterir. MOV DS, AX DS'ye depondu yaramazdır.

**Mov AX, @DATA** yordamında stack segmentin başlangıç adresini gösterir. **Mov DS, AX** DS'ye depondu yaramazdır.

→ **Mov AH, 4CH**  
**INT 21H**

Bu kod, işletim sisteminde DS'sin dönmek için gerekli koddur.

## A Sample Assembly Language Program Using

### FULL SEGMENT DEFINITION

- STSEG SEGMENT → Stack segmentin etiketi STSEG'dır.
- DB 64 DB(?) → Stack'a 64 byte'lik yer tahsis eden direktif.
- STSEG ENDS → Stack segment sonlandırılmıştır. Etiketler zyni olmaz.
- DTSEG SEGMENT → Data segmentin etiketi DTSEG'dır.
- DATA1 DB 52H → 8bit (1 byte) boyutunda [OB] türünden DATA1=52
- DATA2 DS 29H → " " " " "
- SUM DB ? → Toplam 13 byte'lik [OB] boyutlu.

DTSEG ENDS

Son rastgi kullanılır.

CDSEG SEGMENT → Kod segmentin etiketi CDSEG'dir.

MAIN PROC FAR → PROC, prosedür direktifi. Prosedür mutlaka ENDP ile sonlandırılmalıdır.

ASSUME CS: CDSEG, DT: DTSEG, SS: STSEG → Programda

MOV AX, DTSEG

Kullanılan segment etiketi -

MOV DS, AX → Data segment degrinden ve işareti belirler.

MOV AL, DATA1

Dosya içi dönmek için gerekli.

MOV BL, DATA2

ADD AL, BL ;  $AL = AL + BL = 52H + 29H = 7B$

MOV SUM, AL ;  $SUM = AL = 7B$

MOV AH, 4CH ; Dosya içi dönmek için gerekli kod.

INT 21H

MAIN ENDP → PROC prosedür direktifi olduğunu için aynı etiketle (MAIN) belirtildiği yerde tanımlanır.

CDSEG ENDS → Kod segmenti sonlandırılmış için yararlı.

END MAIN → Tam program bitirilir.

Aynı işlemi yapın kod: (SIMPLIFIED SEGMENT DEFINITION)

• MODEL SMALL → Kod 16 bit, data 16 bit 64 KB'ye ayrılr.

• STACK 64 → 64 KB'lik yer stack için ayrıldı.

• DATA

DATA1 DB 52H

DATA2 DB 29H

SUM DB ?

• CODE

MAIN: MOV AX, @DATA → Main etiketidir AX'e data segment

MOV DS, AX  
başlangıç adresi gösteren.

MOV AL, DATA1 → DATA1'i AL'ye aldırdı.

MOV BL, DATA2 → DATA2'yi BL'ye aldırdı.

ADD AL, BL

MOV SUM, AL

MOV AH, 4CH } Programı bitirme komutu.

INT 21H }

END MAIN

ÖLÇÜ: 5 Adet byte boyutundaki veriyi toplayın ve toplam sonucunu kaydeden programı yazınız. Veriler, 25H, 12H, 15H, 1FH, 2BH

• MODEL SMALL

• STACK 64

• DATA

DATA-IN DB (25H) 12H, 15H, 1FH, 2BH

SUM DB (?)

• CODE

MAIN PROC FAR ~~(\*)~~

MOV AX, @DATA

MOV DS, AX

MOV CX, 05  
 MOV BX, OFFSET DATA\_IN  
 MOV AL, 0  
AGAIN: ADD AL, [BX]  
 . INC BX, → BX'i bir arttt.  
 . DEC CX, → CX'i bir azatt.  
INZ → AGAIN →  $ZF = 1$  ise devam, değilse AGAIN'e  
 tıkethetile.  
 MOV SUM, AL  
 MOV AH, 4CH  
 ZINC 21H  
 MAIN ENDP



CPU'nnn Sınıflandırması (CISC, RISC)

CISC :

Amaç : Derleyicinin işini kolaylaştırmak: Makine dili yüksek düzeyli dillerde yadaşıdır.

Programın performansını artırmak: Tetikleme komutları ile kısa kısa programlar yazmamız mümkündür.

Temel Özellikler • Göre sayıda konst var. (100-250)

- Karmaşık komutları ve adreslenebilen (dolaylı, adresleneke ile bellek erişimi)

- Dopruden bellek üzerine işlem yapan komutlar şahsiyî.

## B. Özelliklerin dezavantajları:

- Farklı uyguluklarda komutlar vardır. Gözne ve önceden bellekten gelenek daha iyi dir.
- Bazı komutlar çok az kullanılır.
- İşlençinin iş yapılan kümelerdir.

## RISC Özellikleri

Degisik özelliklerin şahsiyî RISC işlençiler bulundurda birlikte aşağıdaki özellikleri ortostır;

- 1) Az sayıda komut vardır. (yaklaşık 35) ve komutların işlenisi basittir.
- 2) Az sayıda, basit öklere kipi (örngün 3)
- 3) Sabit uygulukta komut sayısı (komut Gözneisi' kolaylaşır.)
- 4) Komutlar bellekte üzerinde işlem yapınca, işlemler iş hafızalarada yapılır.
- 5) Bellege sadece okuma / yazma işlemleri için erişilir.
- 6) Tek geriye dönen ve yürütülebilir komutlar.
- 7) Devrildirilmiz (hardwired) deretimi birimdir.

## Diger Özellikler:

- Cole sayıs Repitit (saklayıcı) (128-256)
- Komutları optimize edilmiş iş hattı
- Harvard Mimarisi'ne sahiptir.

## CISC ve RISC işlemci kullanımları:

### • CISC

VAX, PDP-II, intel x86 until Pentium, Motorola 68K

### • RISC

MIPS, SPARC, Alpha, HP-PA, Power PC, i860, i960-

**ARM**, Atmel AVR.

## CPU'nun sınıflarını belirlemek;

Günümüzdeki durumda RISC işlemciler CISC Özellikleri konusunda tasarımlına katarken, bazi CISC işlemcilerde RISC Özelliklerini içermektedir. Daha da fazla genel bazi RISC tasarımları örneğin Power PC, "saf" RISC deildir. Bu yerde bazi CISC işlemcilerde örneğin Pentium II ve sonrası "saf" CISC tasarımları deildir ve RISC işlemcilerin bazi özellikleri tasır.

RISC işlemcilerin kullanımlığı Jüpiter ait örnekler:

- ARM

- Apple , iPad, Apple iPhone, iPad Touch,
- Palm and PocketPC PDA, Smartphone
- RIM BlackBerry smartphone /email device
- Microsoft Windows Mobile
- Nintendo Game Boy Advance

- MIPS

- SGI Computer, PlayStation, PlayStation2.

- Power Architecture ( IBM Freescale (eski Motorola) )

- IBM super Computer, workstations
- Apple Power PC tabanlı Macintosh

- Motorola Gamecube, wii

- Microsoft Xbox 360
- Sony PlayStation 3

- Atmel AVR

BMW otomobillerde mikrodosythici olarak kullanır.

## \* CPU'nun Sınıflandırılması:

Gesitli özelliklerin göre depository yapıları dört tip mikroişlemci bulunmaktadır.

### 1) Operand sayısına göre;

- \* Sıfır operandlı (zero operand) makinalar. Tipin yapılışı makinanın olasılıkla 2'dir.
- \* Bir adresli makinalar
- \* İki operandlı / adresli makinalar (Saklayıcı-Saklayıcı, saklayıcı-bellek, bellek-bellek)
- \* Üç operandlı makinalar.

### 2) Komut sayıları, sayıları adreslere kiplome göre;

- CISC (Complex Instruction Set Computer)
- RISC (Reduced Instruction Set Computer)

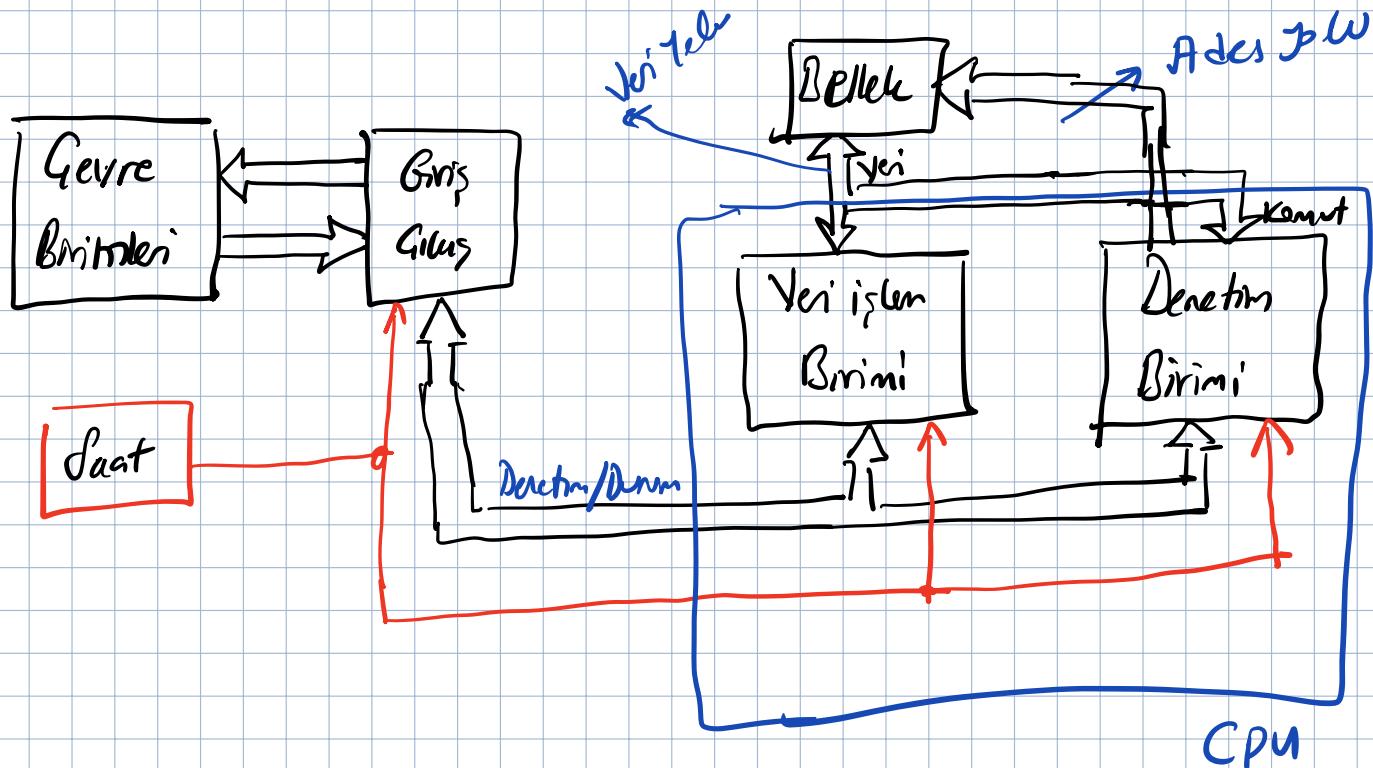
### 3) Komut ve Veri Belleklerine göre;

- Von Neumann Mimarisii
- Harvard Mimarisii

# 1) Von Neumann Mimarisi

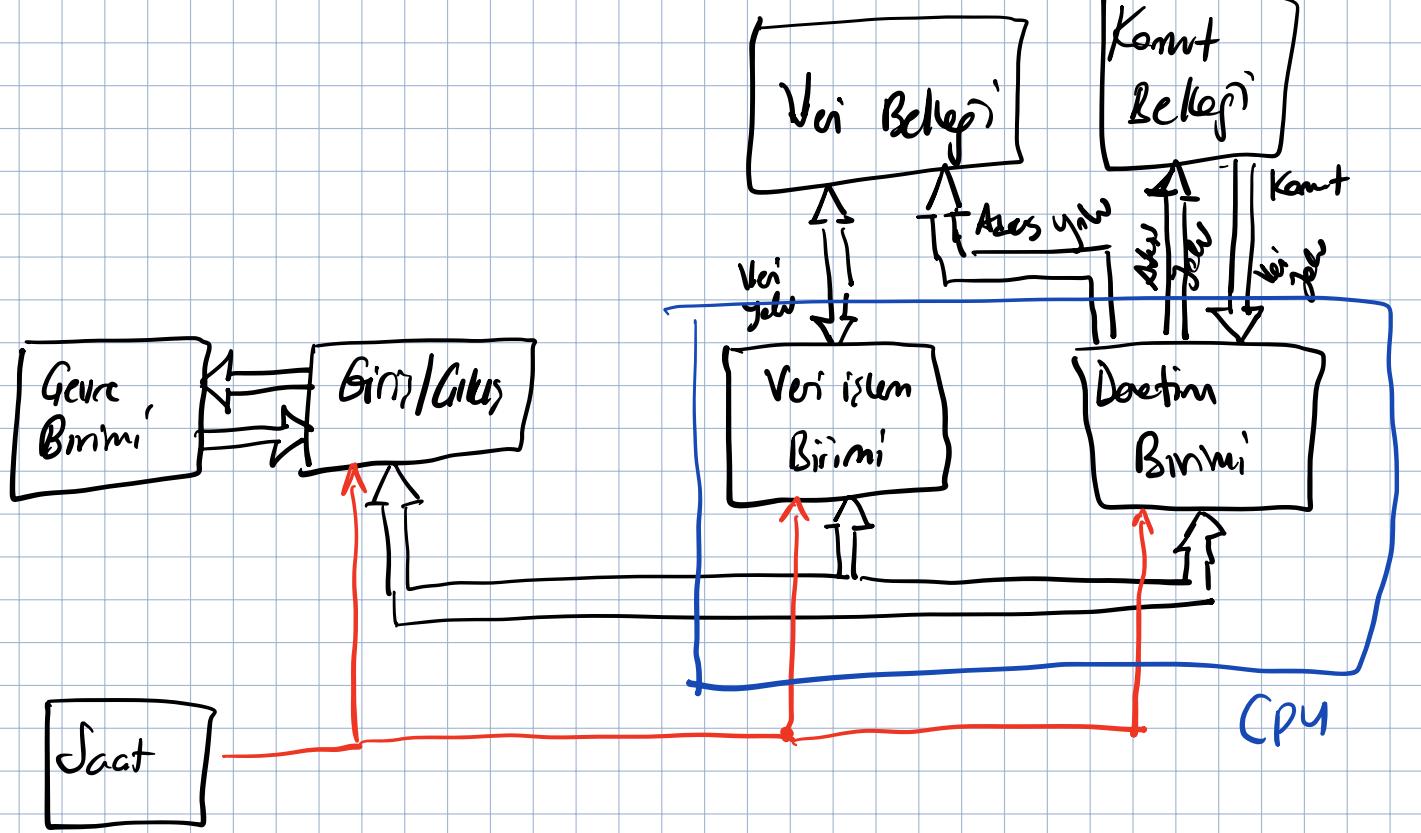
(John von Neumann 1903-1957)

Komutlar ve veriler aynı bellekte yer alır.



# 2) Harvard Mimarisi

Komutlar ve veriler farklı belleklere yer alır. Adres ve veri yolları farklıdır. Böylece aynı anda komut ve operasyon erişimi yapılabilir.



**FAR ve NEAR ( given CS:IP )**

→ NEAR'da kontrol  $\ominus$  arka kod segmentini içerindedir.  
 (sadece IP deplzir.)

→ FAR'da kontrol kod segmentini dışındadır. Hm CS  
 hem de IP deplzir.

### Kosullu Dallanmalar

JC  $\rightarrow$  CF = 1 ise etikete atla

JNC  $\rightarrow$  CF = 1 değilse yani CF = 0 ise etikete atla.

JZ  $\rightarrow$  ZF = 1 ise etikete atla

JNZ  $\rightarrow$   $ZF=0$  ise etikete atla.

JS  $\rightarrow$   $SF=1$  ise etikete atla

JNS  $\rightarrow$   $SF=0$  " " "

JNB/E/JA  $\rightarrow$   $CF=0$  ve  $ZF=0$

JAE/JNB  $\rightarrow$   $CF=0$  ( ~~CF=0~~ ) ~~\*~~ } ~~\*~~

JBE/JNA  $\rightarrow$   $CF$  veya  $ZF=1$

:

Düzenli tablo ekranda bulunur...

\* Dörtler otomatik, gerçeli şart yerine getirilince kontrol yeni yer tasınır.

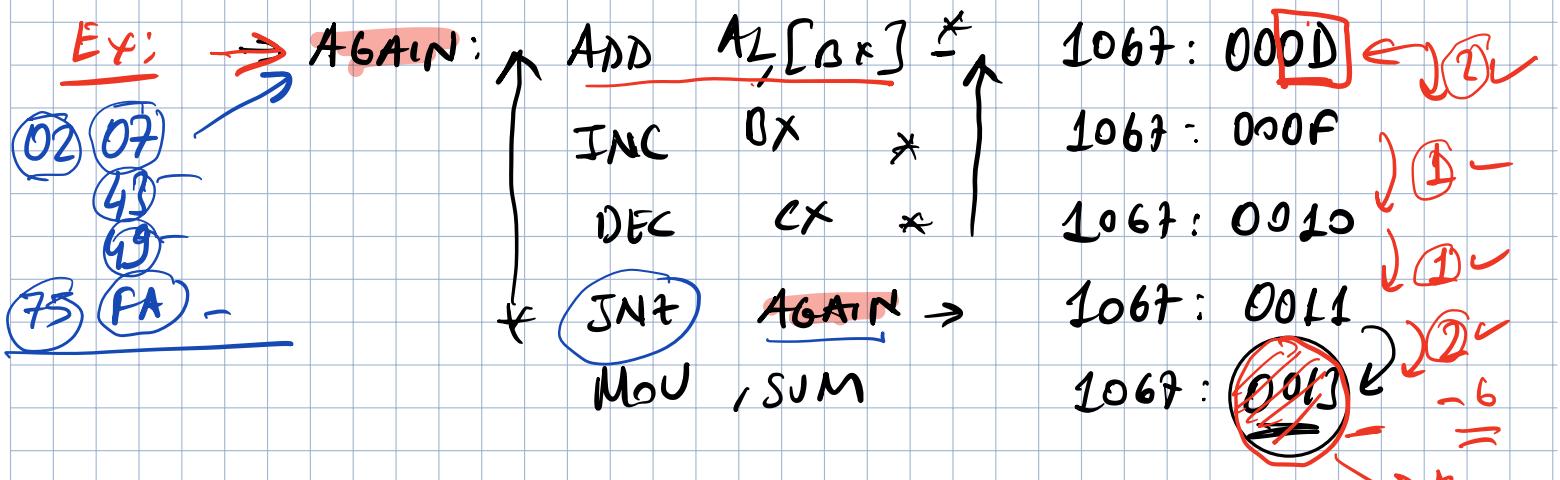
### Kısa Atlamalar

$\rightarrow$  Bütün şartlı atlamalar, kısa atlamalar. Atlamaların  
hedef adres o daki hafızanın -128 ile 127'ye  
sayıları arasında olmalıdır.

\* Koşullu dallanma 2 byte'lik konuttur. 1. byte',  
dallanma konutunun hex. karşılığı; 2. byte', ise  
00 ile FF arasında deðer alan bir sayıdır. Bu sayı

256 farklı adres depo'yu vermektedir. Bu 256 adreslik dilim genelde depo (-128) ve ikinci depo (127) olmak üzere 2 parçaya ayrılmıştır.

→ Genelde depo dallanmalarla 2 byte, 2'ni tüketenini şekilde belirtmektedir.



0013 — JNT 13'üncü adresinde IP depo'

$$+ \underline{\text{FA}} \rightarrow (-6)$$

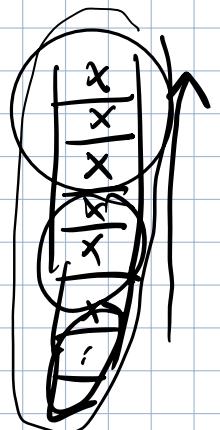
0100 → Tasla bin'i ifade ediyor. Esnek dallanma 00-FF arasındadır.

F (-) olduğu için dallanma genelde depoluks.

FA dallanma miktarını gösterir. Neçye

grideğini bulmak için IP ile dallanma miktarını toplar.

Again ek'ten fitinini form



02 : 0005 ✓ ✓ - ... AGAIN: Mov AL, [BX]+2  
0008 3C 61 - - - - Cmp AL, 61H  
000A 72 06 - - - - JB NEXT ✓  
000C 3C 7A - - - - Cmp AL, 7AH  
 ↓

000E 77 02 - - - - JA NEXT ✓  
0010 24 DF → - - - - AND AL, DFH :

0012 88 04 - NEXT : Mov [SI], AL  
 ↗

✓ 000C - → JB NEXT  
06 ✓

0012 → NEXT Etiketi ✓

0010 → JA NEXT

0002

0012 → NEXT Etiketi ✓

Partisi Atlamözlü

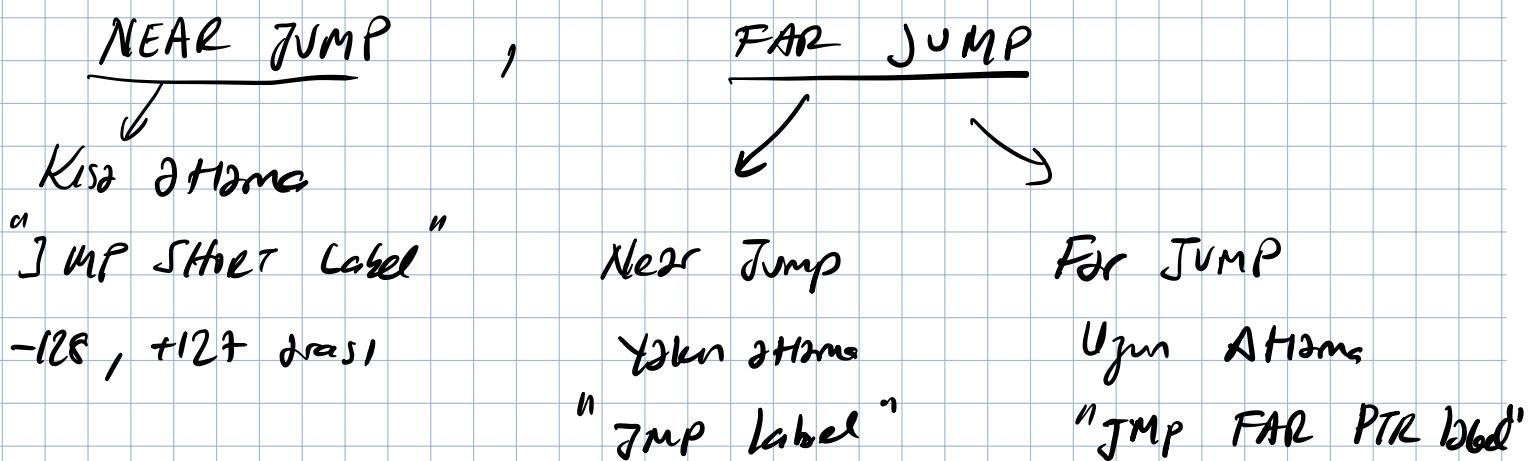
"JMP Label"

Verilen etiketin hedefe direkt gider. Partisi atla-

Ler daki gibi kısa adıma olursa " -128 " ile "+127" arası

sında bir yere, NEAR JUMP olursa dyn. kod segment

icinde herhangi bir yerde, FAR JUMP olursa bulunduğu kod segment desindaki bir yerde olur.



## CALL STATEMENTS (Görmek ifadeleri)

Baska bir kontrol transfer komutu da call 'dur. Bir prosedürin görselini kullanır. Yine dyn. seklide dyn. segment içinde veya dışında çağrılmasına göre NEAR veya FAR olarak gösterilir. Görselin alt program yerine getirildikten sonra mikroişlemci neye geri dönerini bilir. Çünkü sıradaki adresi oto-

math olacak stack'e kaydeder. Eger bunu kullana-

Cakscok su önenli: Gidonan programin son komutu  
"return" kisaltilmasi olun "RET" olmali. Bylece

CPU, direkt buradan stacktan 2idigi adresi  
gectir yapar.

DL:

```
CDSEG SEGMENT
MAIN PROC FAR
ASSUME ...
MOV AX, ...
MOV BA, AX
CALL SUBR1 -
CALL SUBR2 -
CALL SUBR3 -
MOV AH, 4CH
INT 21H
MAIN ENDP
```

SUBR1 PROC ..

RET

SUBR1 ENDP

SUBR2 PROC ..

SUBR2 ENDP

SUB R3 PROC ..

.

SUBR3 ENDP

CDSEG ENDS

END

## DATA TYPES AND DATA DEFINITION

⇒ Assembler Data Directives

★ ORG (Origin)

★ EQU (Equate)

★ DB (Define byte)

★ DD (Define doubleword)

★ DUP (Duplicate)

★ DQ (Define quad word)

★ DW (Define word)

★ DT (define ten bytes)

ORG (Origin): ORG, Offset adresinin başlığını göstermek

için kullanılır. ORG'de sonra hexadecimal

veya 10'luk formda sayı yazılabilir. Değer

cıktı ver segmentinde kullanılır. Eger

Saydırın dona ~~H~~ gelmediyse decimal'dır.

ve assembler onu hexa'ya çevirir.

DL

DT SEG SEGMENT

DATA - IN DW 234DH, 1DE6H, 1BCAH, 566AH

ORG 10 H

SUM DW ?

DT SEG ENDS

DQ ( Define Quad word ) = 8 byte  
( 4 word )

İsaretsiz sayılarıda Çıkarma işlemi

→ SUB hedef, kaynak → hedef = hedef - kaynak

→ Çıkarma işleminde 2'ye tımlayan kullanılır.

→ Çıkan sayının (kaynak) 2'ye tımlayanını al, bunu hedef sayı ile topla. Eldeyi ters çeviriceğiz. Yani C=0 ise 1, 1 ise 0 yap!

ÖR: MOV AL, 3FH

MOV BH, 23H

SUB AL, BH

$$\begin{array}{r} 3F H \\ - 23 H \\ \hline 1C H \end{array}$$

$$\begin{array}{r} 0011\ 1111 \\ - 0010\ 0011 \\ \hline 0001\ 1100 \end{array}$$

11011101

$\rightarrow C=O$  sıkları negative olaca-  
ğında 2'ye tamdeğindi 2'li olur  
fırçalı 0 Sayı olurdu.

$$\begin{array}{r}
 & 00111111 \\
 + & 11011101 \\
 \hline
 & 100011100
 \end{array}$$

Ters Geviv:

$C=1 \rightarrow C=0$   $y_{p11v}$

Song : 00011100

Serus  
dew.  
★

OR: DATA 1 DB 4CH  
DATA 2 DB 6EH  
RESULT DB ?  
:  
:

MOV DH, DATA1 DH= DATA1

SUB DH, DATA2 → DH = DH - DATA2

JNC NEXT → If CF=0 jump to NEXT

$\text{NOT DH} \rightarrow \text{if CF=1 take the 1's complement}$

INC DH → increment get 2's complement

NEXT: MOV RESULT, DH → Save DH in Result.

# İşaretler Sözlük Toplancı

$\text{ADD}$      $\text{hedef}, \text{kaynak} \rightarrow \text{hedef} = \text{hedef} + \text{kaynak}$

OR: MOV AL, FS H → 11110101

$$\begin{array}{r} \text{ADD} \quad A_2, \quad 0BH \quad \longrightarrow \\ + \quad 0000 \quad 1011 \\ \hline 1 \quad 0000 \quad 0000 \\ \hline \quad \quad \quad 0 \quad 0 \quad H \end{array}$$

$AL = 00H \rightarrow$  Elde göz öncü

2 linmeyer.

Eldayı göz öncüne alır  
kont ADC

$ADC = Add\ with\ Carry = Elde\ ile\ tpls.$

ADC hedef, kaynak ; hedef = hedef + kaynak + CF

ÖRNEK 5 Byk veriyi toplayan bir program yazınız.  
Datalar 10'luk tabanda : 125, 235, 197, 91, 48

Hexaya çevirelim;

$$125 = 7DH$$

$$197 = C5H$$

$$48 = 30H$$

$$235 = EBH$$

$$91 = 5BH$$

Kodu yazalım;

- MODEL SMALL
- STACK 64
- DATA

COUNT EQU 05

DATA-IN DB 7DH, EBH, C5H, 5BH, 30H

→ ORG 0008H

SUM DW ?

• CODE

MAIN PROC FAR

```

MOV AX, @ DATA-IN
MOV DS, AX
MOV CX, COUNT
MOV SI, OFFSET DATA-IN
MOV AX
BACK: ADD AL, [SI] → CF ← →
      ADC AH, 00 → → AH = AH + 0 + CF
      INC SI
      DEC CX - - 5(4,1,2,1,0)
      INT BACK

```

```

MOV SUM, AX
MOV AH, 4C H
INT 21H
MAIN ENDP
END MAIN

```

OR: 5 word datayn top layen bir program yonunit.

Data Br: 27345, 28521, 29533, 30105, 32175

- MODEL SMALL

- STACK 64

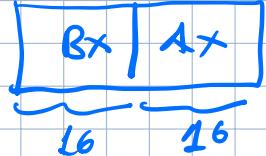
- DATA

COUNT EQU 05

DATA-IN DW 27345, 28521, 29533, 30105, 32175

ORG 0010

SUM DW 2 DUP(?) → 2 byte Ax tanımlayıcı. →



• CODE

MAIN PROC FAR

MOV AX, @DATA\_IN

MOV DS, AX

MOV CX, COUNT

MOV SI, OFFSET DATA\_IN

MOV AX, 00 ✓

MOV BX, AX ✓

BACK: ADD AX, [SI]

ADC BX, 00

INC SI

INC SI

DEC CX

JNZ BACK

MOV SUM, AX

MOV SUM+2, BX → SUM :

MOV AH, 4CH

INT 21H

MAIN ENDP

END MAIN



## Multiword (bir kag word boyutunda) sayıların toplanması

Registers, 16 bit boyutunda olduğundan 16 tane daha büyük sayıları toplamak için onları kucuk parçalara biseip, sonra top layout.

ÖR: DATA1 : 548FB996]CEF H, DATA2 : 3FC04FA23B8D H

• MODEL SMALL

• STACK 64

• DATA

DATA1 DQ 548FB996]CEF H

ORG 0020H

DATA2 DQ 3FC04FA23B8D H

ORG 0020H → offset

DATA3 DQ ?

• CODE

MAIN PROC FAR

MOV AX, @DATA

MOV DS, AX

CLC ; clear carry

MOV SI, OFFSET DATA1

MOV DI, OFFSET DATA2



MOV BX, OFFSET DATA1

MOV CX, 04

BACK: MOV AX, [SI]

ADC AX, [DI]

MOV [BX], AX

INC SI

INC SI

INC DI

INC DI

INC DX

ZNC BX

Loop BACK

= DEC AX  
JNZ BACK

MOV AH, 4CH

INT 21H

MAIN ENDP

END MAIN

## SBB

SBB hedef, kaynak  $\Rightarrow$  hedef = hedef - kaynak - CF

→ Multi byte ve multiword için kullanılır.

→ Eğer CF=0 ise SBB, SUB ile aynı gibi girer. ✓

→ Eğer CF=1 ise SBB, sonraki 1 silinirler.

ÖR: DATA-A DD 62562FAH  $\Rightarrow$   $0625 = \text{DATA-A} + 2$   
 DATA-B DD 412963BH  $\Rightarrow$   $0412 = \text{DATA-B} + 2$

→ MOV AX, WORD PTR DATA-A

→ SUB AX, WORD PTR DATA-B

MOV WORD PTR RESULT, AX

MOV AX, WORD PTR DATA-A+2 ✓

SBB AX, WORD PTR DATA-B+2

MOV WORD PTR RESULT+2, BX

Yüksek sayılarla  
direct silinme yapın  
komut olmalıdır için  
WORD PTR kullanılsın

→ Öncelikle AX'e WORD PTR DATA-A diyecek 62562FAH

sayısının son 4 digitini (düşük olun) yerlestiriyor. AX=62FA

\* SUB AX, 963B (WORD PTR DATA-B)

62FAH  
- 963B H  $\rightarrow$  0110 0010 1111 1010  
                    $\rightarrow$  - 1001 0110 0011 1011

$$\begin{array}{r}
 0110\ 0010\ 1111\ 1010 \\
 + 0110\ 1001\ 1100\ 0101 \\
 \hline
 0\ 1100\ 1100\ 1011\ 1111
 \end{array}$$

2' ✓

$C=0 \rightarrow C'=1$  Negatif. Sonra tenevî ol ( $2'$ )

0011 001 10100 0001 ✓

\* SBB AX, 0412 ✓

$$\begin{array}{r}
 0625 \xrightarrow{-1} \\
 - 0412 \\
 \hline
 0212
 \end{array}$$

0110 0010 0101  
 - 0100 0001 0010

→

$$\begin{array}{r}
 0110\ 0010\ 0101 \\
 + 1011\ 1110\ 1110 \\
 \hline
 10010\ 0001\ 0011
 \end{array}$$

2' ✓

$CF = 1$     $CF = 0$

$- CF = 1 = 0212$

★ Onelikle AX'e WORD PTR DATA\_A çalıştır 62567FAH

Sayımları son 4 bitini kaydulk.

AX = 62 FAH

\*  $Ax = Ax - \text{WORD PTR DATA\_B} \rightarrow 62FAH$   
 $\underline{- 963BH}$

$$\begin{array}{rcl}
 Ax = CC\text{ BF} & = & 0110\ 0010\ 1111\ 0101 \\
 2' \rightarrow & + & \underline{0110\ 1001\ 1100\ 0101} - \\
 & & \begin{array}{c} 0 \\ | \\ 1100 \end{array} \quad \begin{array}{c} 1 \\ | \\ 100 \end{array} \quad \begin{array}{c} 1 \\ | \\ 100 \end{array} \quad \begin{array}{c} 1 \\ | \\ 011 \end{array} \quad \begin{array}{c} 1 \\ | \\ 111 \end{array} \\
 & & C \quad C \quad B \quad F
 \end{array}$$

$\hookrightarrow CF = 0 \rightarrow \text{terminal} \quad CF = 1$

\* Daha sonra  $Ax'$  den çıkış sonucu WORD PTR RESULT'a yararlı (Düşük 4 bitini yararlı)

\*  $Ax'$  e Simdi DATA\_A'nın yükselt 4 bitini WORD PTR DATA\_A+2 ile atarlık.

$Ax = 0625$

\* SBR  $\rightarrow Ax = Ax - 0412 - CF$

$$\begin{array}{rcl}
 0625 \rightarrow 0110\ 0010\ 0101 & \longrightarrow & 0110\ 0010\ 0101 \\
 0412 - 0100\ 0001\ 0010 & + & \underline{1011\ 1110\ 1111} - \\
 & & \begin{array}{c} ① \\ | \\ 0010 \end{array} \quad \begin{array}{c} 0001 \\ | \\ 1 \end{array} \quad \begin{array}{c} 001 \\ | \\ 1 \end{array}
 \end{array}$$

$CF = 1 \rightarrow \text{terminal}$

$CF$  yukarıda 1 bulunmuştur. Simdi bu eldeyi dikkate almalıyız.  $2^{13}-1=212$  olur.

Sonuç = 02 12 CC BF H

Result → 02 | 12 | CC | BF

İsaretsiz Sayıları Çarpımı

→ Byte × Byte

Operatörlərin bir fənəsi kesinlikle AL'də olmalıdır. Daha sonra operand registrda veya hafıza olabilir. Aşağıda sonradan AX'ın təsviri dedir.

ÖR: RESULT DW ?

!

MOV AL, 25H

MOV BL, 65H

MUL BL ;  $AL = AL \times BL = 25 \times 65$

MOV RESULT, AL

02; From Data Segment

DATA1 DD 25A

DATA2 DB 65H

RESULT DW ?

; Code

MOV AL, DATA1

}

MOV BL, DATA2 } Registre Adreslene Modu.

MUL BL

MOV RESULT, AX

ÖR

MOV AL, DATA1 ]

MUL DATA2 ]

MOV RESULT, AX ]

Direkt Adreslene Modu...

→ WORD X WORD

Operatörlerden bir tanesi kesinlikle AX'in içinde olmalıdır.

Diger operand registrede veya hafızada olabilir. Sırpılmışken-  
de sonra döşük ikinci word (16 bitlik kismi) AX'e,  
yuksek ikinci word (yani 16 bitlik kismi) DX'de olacaktr.



ÖR: DATA3 DW 2278H

DATA4 DW 2F74H

RESULT1 DW 2 DUP(?) →

;

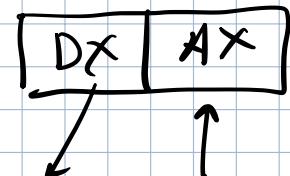
MOV AX, DATA3

MUL DATA4

MOV RESULT1, AX

MOV RESULT1+2, DX

RESULT1



RESULT1+2 RESULT1

## → WORD X BYTE

WORD x WORD çarpımı islemiğe birleştirilir. Fakat AL, byte olm operatör içermelidir. ve AH kendiyle 0 olmalıdır.

ÖR: DATA5 DB 6BH

DATA6 DW 12C3H

RESULT DW 2 DUP(?)

:  
:

MUL AL, DATA5

SUB AH, AH ; AH sıfırlandı.

MUL DATA6

MOV [BX], AX } Yandelenir. \*

MOV [BX]+2, DX }

İsaretçi Sayıda Bölme:

## → Byte / Byte Bölme

Bölmenin sağ kısmı kendiyle AL'de olmalıdır. AH da 0'a setlenmelidir. Bölün sayı immediyat (sayı) olmalıdır. Registr veya hafıza olabilir.

Bölme işleminden sonra quotient AL'de olarak ve kalan da

AH'da olacaktır.

ÖR: DATA7 DB 95  
 DATA8 DB 10  
 QUOT1 DB ? ✓  
 REMAIN1 DO ? ✓

; Using direct mode

```
MOV AL, DATA7
SUB AH, AH
DIV DATA8
MOV QUOT1, AL
MOV REMAIN1, AH
```

; Using register address mode

```
MOV AL, DATA7
SUB AH, AH
MOV BH, DATA8
DIV BH
MOV QUOT1, AL
MOV REMAIN1, AH
```

→ ~~MOV AL, DATA7~~  
~~SUB AH, AH~~  
~~DIV 10~~ → Aslı olmasın. Çünkü tam sayı  
 immediate olmasın... !

→ WORD / WORD BÖLME

Poj, AX'de olmalıdır ve DX sıfır olmalıdır.

~~(X)~~ ← Poj  
~~(D)~~ ← pojda

Pojda hafızaya veya Register olabilir. Bölme işlemlerinden sonra

AX bölüm, DX kaladır.

ÖR: MOV AX, 10050

SUB DX, DX

MUL BX, 100

DIV BX

MOV QUOT2, AX

MOV REMAIN2, DX

→ WORD / BYTE BÖLME

Pay AX'de olmalıdır. Payda hafızası veya register olabilir.

Bölme işleminden sonra AL bölüm, AH kalır.

ÖR: MOV AX, 2005

MOV CL, 100

DIV CL

MOV QUO, AL

MOV REM, AH

→ DOUBLE WORD / WORD BÖLME

Pay AX ve DX'de olmalıdır. En düşük 24 tane word

AX, en yüksek 24 tane word DX'dedir. Bölme işleminden

sonra AX bölüm, DX kalır.



→ Hafızadaki sayı bu şekilde dir.

ÖR: DATA1 DD 105432

DATA2 DW 1000

QUOT DW ?

REMAZN DW ?

MOV AX, WORD PTR DATA1

MOV DX, WORD PTR DATA1+2

MOV BX, DATA2

DIV BX

MOV QUOT, AX

MOV REMAZN, DX

## LOJIK KOMUTLAR

### AND

AND hedef, kaynak  $\rightarrow$  Hedef ile kaynaktaki' sayılar AND

İşlemire tabii tutular ve sonus

hedef yapar.

ÖR

MOV BL, 35H

AND BL, OFA

35H  $\rightarrow$  0011 0101

OFH  $\rightarrow$  0000 1111

$\rightarrow$  Bütün bitler 0 olursa OF=1  
CF=0, ZF=0, OF=0, SF=0, PF=1

X	Y	AND
0	0	0
0	1	0
1	0	0
1	1	1

→ AND Komutu bir operatör belli bitlerini sıfırlamak için kullanılır. Ayrica bir operatör sıfır olup olmadığını test etmek için kullanılır.

ÖR: AND DH, DH → bu isimden sonra sonuc sıfır ise DH sıfırdır, dolayısıyla gelir.

OR Komutu

X	Y	OR
0	0	0
0	1	1
1	0	1
1	1	1

OR hedef, kaynak → Hedef ile kaynaktaki sayılar OR işlemi tabii tutulur. ve sonuc hedefe gelir.

→ OR bir operatör sıfır olup olmadığını kontrol etmek için kullanılır. Dİrgin OR BL, 0 işlemi bunu belirtir. Eğer BL sıfır ise sonuc sıfır olacaktır. ve ZF = 1 olur.

ÖR: MOV AX, 0504H → 0000 0101 0000 0100  
 OR A+, DA68H → 1101 1010 0110 1000

$$\begin{array}{r} 1101 \\ + 1010 \\ \hline 1101 \end{array}$$

$$CF=0 \quad SF=0 \rightarrow (11 \text{ direk 1 var.})$$

$$OF=0 \quad ZF=0$$

A⊕B

## XOR Komutu

XOR hedef, kaynak

X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

XOR komutu iki bitin farklılığı  
dönüşde sonuc 1 yapar. Diğer  
dönüşlerde sonuc sıfır olur.

→ XOR komutu regitrların içeriğini kendisigile XOR la-  
yarak tanzimede kullanılır. XOR, iki regitren  
oynu deşinde olduğunda da entegre edilebilir.

→ XOR işlemi yapıldığında terminlerden birini 0'lardan  
oluşan bitlerden seçerek diğer terminde hangi bitin  
tersini alınsın isteyse orun 1 yaparak deşirebilir  
yapabilmiz. (Toggle)

D<sub>2</sub> XOR B<sub>1</sub> 0<sub>4</sub> H → yani B<sub>1</sub>'yi 0000 0100 ile

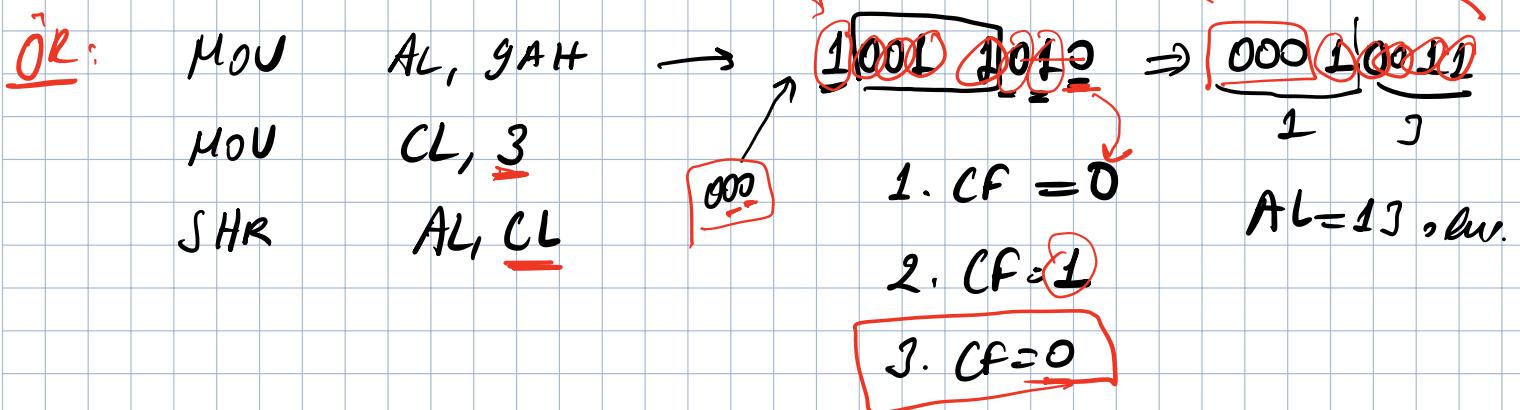
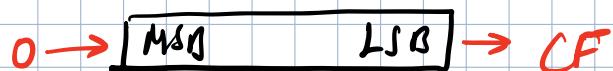
XOR isenmiş tabii tutuyor. burada sadece 1 bitin olmasına  
D<sub>2</sub> biti B<sub>1</sub>'de deşireni diğer deşire oynuyor.

## SHIFT KOMUTU

SHR

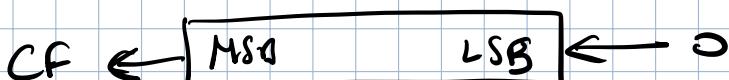
→ SHR hedef, kaynak → sağa kaydırır. (Shift Right)

→ Bu lojiksel olarak sağa kaydırma işlemidir. Terim sağa doğru her ilerde bir bit kayar, ter kayma sonrasında en düşük değerli bitin değeri CF'a gizler ve en yüksek değerli biti 0 gizler.



### SHL Komutu

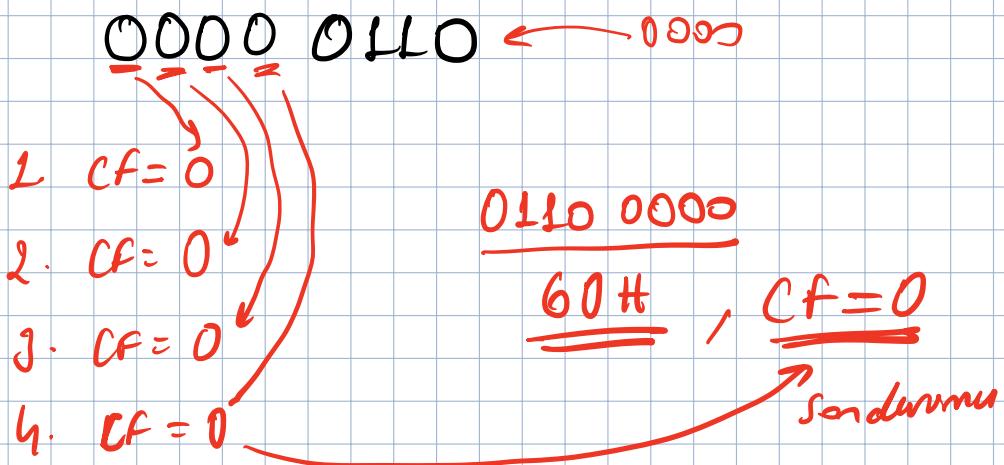
→ SHL hedef, kaynak ⇒ sola kaydırır. (Shift Left)



ÖR: MOV DH, 6

MOV CL, 4 → Kaydırma sayısı

SHL DH, CL



→ Not: Kaydrome yapılmış hedef terim register yada hafızas  
gözdelenilen bir bit bir deej olmalıdır. Dırect says kaydetlamaz.

→ Eğer sadece L deej kaydrolaması, direkt STH AL,L  
yazılabilir, L'den farklı kaydrolaması sayın CL registeri  
yazılmasını gerektir. ve iżleni STH AL,CL şeklinde yaz-  
malyız.

SHR iñ de geçerlidir. ✘

İsretsiz Sayılarda Karşılaştırma

## CMP

→ CMP hedef, kaynak  $\Rightarrow$  COMPARE

→ Terimler değişmet. Olduğu gibi kalsın.

→ Hedef terim register yada hafızanın gözdelenilen bir deej olmalıdır. Kaynak terimi ise registerdeki, hafızanın gözdelenilen bir deej olmalıdır.

sayı okunabilecegi gibi direkt bir sayı da okunabilir.

→ CΜΤ komutu 2 terimi karıştırır. ve durusu şı-

halaraklar değişir. Burada bu işi içinden veren bayraklar,

Cf ve Zf 'dır.

→ Hedef > Kynnak Cf=0 Zf=0 —

Hedef = Kynnak Cf=0 Zf=1 —

Hedef < Kynnak Cf=1 Zf=0 —

BCD ve ASCII

⇒ BCD sayı sistemleri 0'dan 9'a kadar olan

rakamların ikili sisteme gösterimini BCD olarak

adlandırır. Bilgisayar literatüründe BCD ile ilgili iki  
terme karıştırır.

1) Un packed BCD (Paketlenmiş BCD)

2) Packed BCD (Paketlenmiş BCD)

Unpacked BCD: Unpacked BCD'de sayılar deðulk dört biti BCD saysnır. gosterir ve diper bitlerini sıfırdır.

Örnek; "0000 1001" ve "0000 0101" 9 ve 5 iki  
unpacked BCD'dir.

Packed BCD: Packed BCD'de tek byte iki BCD  
saysnır iken. Bir dosya 4 bitlerde, diper yilek  
4 bitlerde olur. Örnek; "0101 1001" ifadesi  
59 sayının ifade eder

<u>Digit</u>	<u>BCD</u>	<u>ASCII Sıfırlar</u>			
		<u>Key</u>	<u>ASCII (Hex)</u>	<u>Binary</u>	<u>BCD (Unpacked)</u>
0	0 0 0 0	0	30	0110000	0000 0000
1	0 0 0 1	1	31	0110001	0000 0001
2	0 0 1 0	2	32		
3	0 0 1 1	3	33		
4	0 1 0 0	4	34		
5	0 1 0 1	5	35		
6	0 1 1 0	6	36		
7	0 1 1 1				
8	1 0 0 0				

9

2001

7

37

1

8

38

9

39

011 1001 0000 1001

⇒ ASCII kodyla gösterilen rakamları BCD'ye çevirmek için yüksek tepsilikli 4 bitin bulunduğu yerdeki 0011 bulunmalıdır. Bu nın yapması için ASCII sayı 0000 1111 (0FH) ile AND işlemi yapılabilir.

ASCII'yi BCD'ye Çevirme

→ ASCII'yi Unpacked BCD'ye Çevirme:

→ ilk olarak 011'ler kurtulmalıdır.

→ Daha sonra 0000 1111 (0FH) ile AND'lenmelidir.

ÖR: ASC DB '9562481273'

ORG 0010 H

UNPACK PB TO DUP (?)

:

MOV CX,5 → Döşen sayacı

MOV BX, OFFSET ASC → BX, ASCII'ın belirleyen  
isn't ediyor.

MOV DI, OFFSET UNPACK → DI, UNPACKED BCD'inin  
bulunduğu yer işaret ediyor.

AGAIN MOV Ax, WORD PTR [BX] → ASCII numarası Ax'e  
taynacak.

AND Ax, 0F0F → 011'de kurtarılır.

MOV WORD PTR [DI], Ax

ADD DI, 2

ADD BX, 2

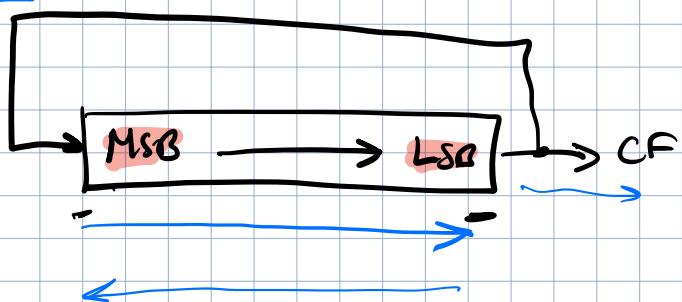
Loop AGAIN

→ ASCII'yi Packed BCD'ye çeviri ✓

→ Packed BCD'ler ASCII'ye dönüştürme ✓

### Döndürme Komutları

ROR ✓



→ Burada en düşük degerdeki bit hem en yüksek degerdeki biti  
geliş hem de CF' a kopyalanır.

→ CL döndürme sayımı tutucu sadecə 1 deyə döndür-  
keçise CL'ye geri yoxdur.

ÖR: MOV AL, 36H

ROR AL, 1

ROR AL, 1

ROR AL, 1

veya

MOV AL, 36H

MOV CL, 3

ROR AL, CL

36 H →

0011 0110 10

1. Döndürme;

000110011, CF = 0

2. Döndürme; 10001101, CF = 1

3. Döndürme, 11000110, CF = L

AL = C6H

CF = 10

ÖR: MOV BX, CFESTH

MOV CL, 6

ROR BX, CL

BX = 1100 0111 1110 0101

1. kez döndürme; 1110 0011 1111 0010

CF = 1

CF = 0

2. Kır döndürme : 0111 0001 1111 1001 CF=0

3. Kır döndürme : 1011 1000 1111 1100 CF=1

4. Kır döndürme : 0101 1100 0111 1110 CF=0

5. Kır döndürme : 0010 1110 0011 1111 CF=0

6. Kır döndürme : 1001 0111 0001 1111 CF=1

$$BX = \underbrace{1001}_g \quad \underbrace{0111}_7 \quad \underbrace{0001}_1 \quad \underbrace{1111}_F = 971F \text{ H}$$

## ROL Komutu

rotate left = Sila kaydırma



→ En yüksek değerlikli bit en düşük değerlikli bite geçer

ve aynı zamanda CF'a kopuları.

→ CL döndürme sayısını tutar. Sadece 1 defa döndür -  
teke CL'ye geçer yaktır.

ÖR:  $\text{MOV AL, } 47\text{H}$

$\text{ROL AL, 1}$

$\text{ROL AL, 1}$

$\text{ROL AL, 1}$

$\text{ROL AL, 1}$

$AL = \underline{\underline{0100}} \quad \underline{\underline{0110}}$

1. döndürme:  $1000 \quad 110$   $CF=0$

2. döndürme:  $0001 \quad 1101$   $CF=1$

3. döndürme:  $0011 \quad 1010$   $CF=0$

4. döndürme:  $0111 \quad 0100$   $CF=0$

$AL = 0111 \quad 0100, \quad CF=0$

ÖR: Bir byte'lık 1'lerin sayısını bulan programı yazınız.

DATA DB  $97\text{H}$

COUNT DB ?

SUB BL, BL  $\rightarrow$  BL 'yi sıfırladık.

MOV DL, 8  $\rightarrow$   $DL = 08\text{H}$

MOV AL, DATA  $\rightarrow AL = \underline{\underline{97\text{H}}}$

5 tane 1 var.

AGAIN:  $\rightarrow \text{ROL AL, 1} \Rightarrow \underline{\underline{1001}} \quad \underline{\underline{0110}} \rightarrow \underline{\underline{0010}} \quad \underline{\underline{1111}} \quad CF=1$

JNC NEXT  $\rightarrow CF=1$  değil ise ( $CF=0$ ) de NEXT'i git

INC BL  $\rightarrow$  BL'yi artır.  $BL = \underline{\underline{01}}$

NEXT

DEC DL → DL' yeri aralıktır DL=07

JNZ AGAIN → ZF=1 değilse ZF=0 ise AGAIN'e git

MOV COUNT, BL

2. Tur. 0010 1111 → ROL ⇒ 0101 1110 CF=0

CF=1 değil, CF=0 ise NEXT'e git . NEXT'i gireriz.

DL'yi aralıktır ⇒ DL=06

ZF=1 değil , ZF=0 ise AGAIN'e git . Gireriz.

3. Tur. . . .

'  
'  
'

6. Tur. 1100 1011 ⇒ ROL ⇒ 1001 0111 CF=1

CF≠1 in CF=0 ise NEXT'e git . Gitmeyiz !

BL'yi 1 25'tür. BL=05

RCR

⇒ RCR = Sayı tarafları carry üzerinde doldurur



⇒ En düşük depoluklu bit CF'ye gider. CF'deki depoluk  
en yüksek depoluklu bite gider.

⇒ CL döndürme sayımı tutar. Sadece 1 sefa döndürmekte  
CL'ye gerek yoktur.

ÖR : CLC → CF=0 yapar.

MOV AL, 26H

RCR AL,L      veya

RCR AL,L

RCR AL,L

CLC

MOV AL, 26H

MOV CL,J

RCR AL, CL

26 H → 0010 0110      CF=0

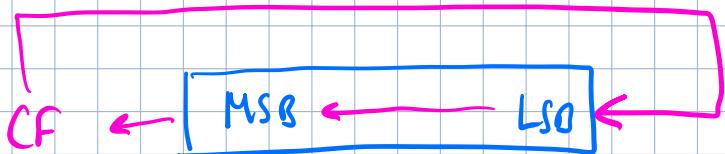
1. Kır döndürme = 0001 0011      CF=0

2. Kır döndürme = 0000 1001      CF=1

3. Kır döndürme = 1000 0100      CF=1

AL = 1000 0100      CF=1

# RCL Konutu



$\Rightarrow$  En yikesi deperlikli bit CF'ye geser. CF'deki  
deperde en dusun deperlikli bite yasulur.

$\Rightarrow$  CL döndürme sayın, tufar. Sadece 1 deya döndürmekle se  
CL'ye gel yahut.

OR : STC → CF = 1 yapaar. }  
MOV BL, 15H } very  
RCL BL, 1  
RCL BL, 1

STC  
MoV BL, 15H  
MoV CL, 2  
RCC BL, CL

$$BL = LS H = \underline{\underline{0001 \ 0101}} \quad CF = 1$$

## 1. Kategori Döndürme:

$$0010 \quad 1011 \quad CF=0$$

## 2. Ket döndürme :

0101 0110  $CF = 0$

$$BL = 0101 \quad 0110, \quad CF \Rightarrow$$

ÖR:

CLC

MOV Ax, LGIC H

MOV CL, 5

RCL Ax, CL

.

,

,

↓  
istem yaparsa

↓

Ax = 0010 0011 1000 0001 , Cf = 1

\_\_\_\_\_

\_\_\_\_\_

Soru Çözümü

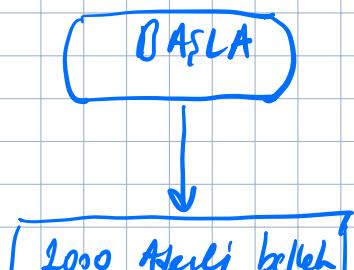
① 1000 ve 2003 adresli bellek单元indeki 2 sayıya toplayın  
ve sonuc FFH'dan büyük ise (faaliyet olursa) 300 adresine

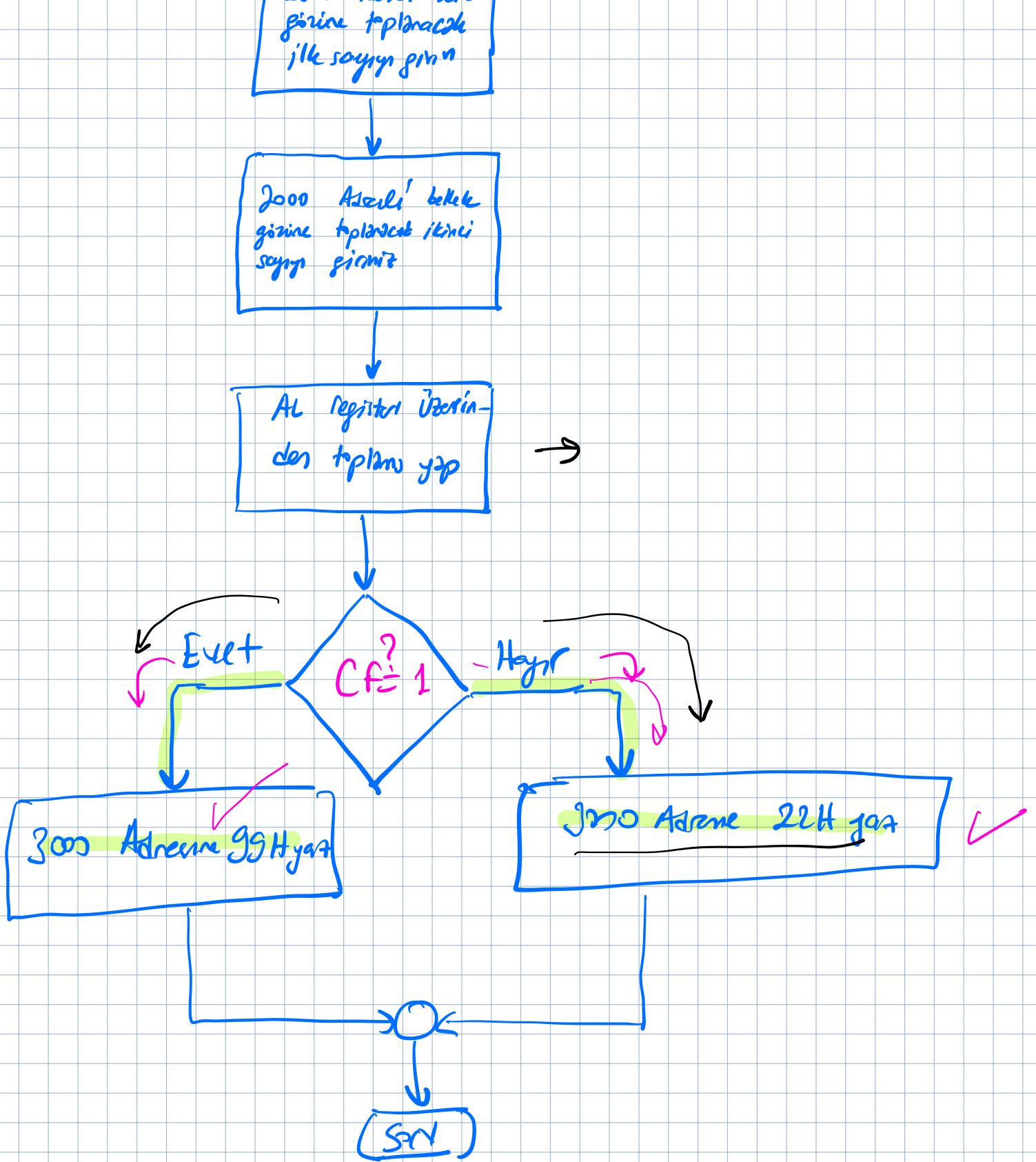
95H ; küçükse (elde olumsuzsa) 22H yazar programı

a) Aşırı log programı

b) Kodun yazdır.

a)





## b) Assembly Programı

Org 100H

Mov [100], 22H ; Toplanacak ilk sayı

Mov [200], 88H ; Toplanacak ikinci sayı

MOV AX, 0H  
 MOV BX, 0H  
 MOV AL, [1000H]  
 ADD AL, [2000H]  
 JC greater ↓  
 MOV [3000], 22H  
 Jmp stop

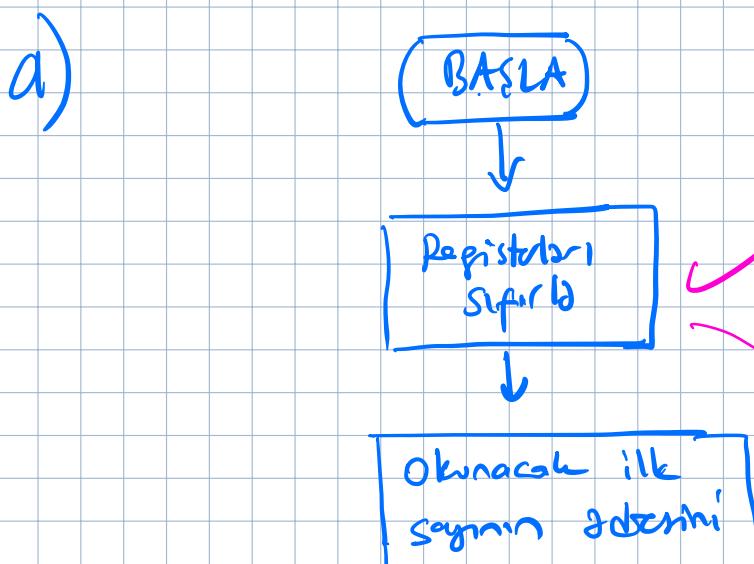
greater: mov [3000], 9SH

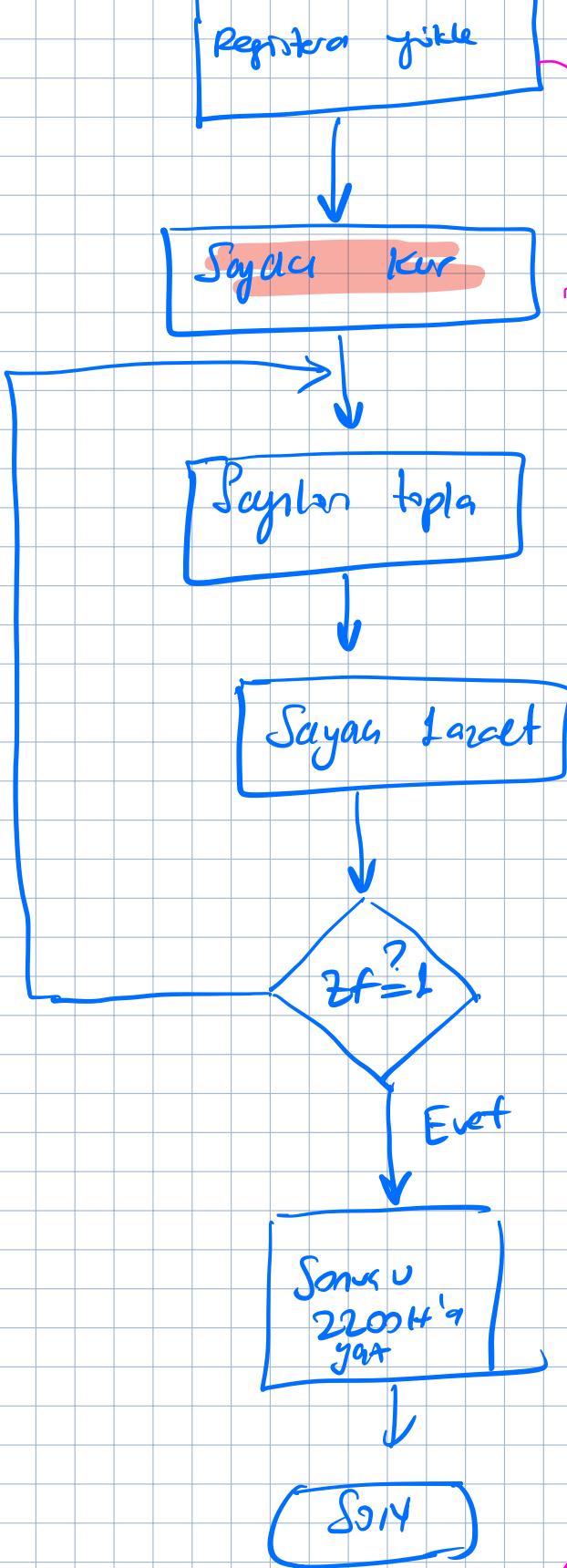
stop: ret

(2) 1100H ile 110EH adreslerinde tüm veriler toplayın

ve sonra 2200H'a yazın programı;

- a) Aşağıdaki programı çiziniz.
- b) Assembly Programı yazınız.





b)

ORG 100 H

MOV AX, 0

MOV BX, P



Sonrakı 1003H adresine, elde olunmuşsa işlem sonucu  
denilen 1003H'a yazan ve duran programı yarınıt.

→ İDEU      300H - 300BH      bellek gösteri aralındaki  
sayılardan 2'ye bölüp ilk bölümdeki sayıların  
sayısı 3100H belleğin yanın ve sonra devam  
programı yarınıt.

## Cevap 1:

ORG 100H

MOV [1000H], LEH ; 1000 numaralı hafıza gizine 5000 istenilen  
ilk sayıyı girdi.  
MOV [1001H], 20H ; 1001 numaralı hafıza gizine 5000 istenilen  
ikinci sayıyı girdi.

MOV AX, 0H ; AX registerini sıfırlaştı.

MOV BX, 0H ; BX registerini sıfırlaştı.

MOV BL, 20H ; Gördüğün ikinci sayıyı BL'ye attıktı.

TOPLA: ADD AX, LFH ; AX'ye ilk girdiğin sayısını attıktı.

DEC BL ; BL'yi 1 azaltı

JNT TOPLA

$$\underbrace{LE + LE + LE + \dots + LE}_{20+1} = LE \times 20H$$

AX  
LE AL PA

→ OR AH, 00H ; AH' ile orlayip ikinci kontrol ederiz

JNT ABC ; sonus sıfır depise ABC dengirm after

JZ SON

ABC MOV [1002H], AH

SON : MOV [1003H], AL

1111 2nd 0000  
X X 1111

1000 X 00010 1111  
D000 X D000

STOP:

RET

## Cevap 2

ORG 100H

MOV [3000H], 15

MOV [3001H], 22

MOV [3002H], 30

MOV [3003H], 40

MOV [3004H], 26

MOV [3005H], 5

MOV [3006H], 44

MOV [3007H], 11

MOV [3008H], 10

MOV [3009H], 27

MOV [300AH], 19

MOV [9000H], 1F

MOV Si, 9000 H

MOV CL, 0H → CL; 2'ye bölen ve kalan  
bolunmayan sayıları add etti

JMP TEKRAR

(A)

MUS:

INC SI

CMP SI, 900CH

JZ STOP

TEKRAR: MOV BL, 2

MOV AH, 00H

MOV AL, [Si] AL=15

DIU BL

$$AL \mid BL = 15 \mid 2 = 7.5$$

CMP AH, 0H

JNZ MUS

AL=22

$$AL \mid BL = 22 \mid 2 = 11 \quad AL=11$$

AH=0

DÖRDÜ BÖLÜ:

MOV AL, [Si]

MOV BL, 4

MOV AH, 00H → AH=00

DN BL → 22 / 4 → 5, 2

CMP AH, 0H

JNT ART

JZ MUS

(B) CL

ART

INC CL → CL=1

AL  
AH

$$40 / 4 = 10 \mid 0 \rightarrow AH=2$$

AL=10

# JMP MUS

STOP: MOV [3100H], CL

RET

ÖRNEK:

2000H - 200AH adresleri arasındaki bellek gösterimdeki sayıların

ilk 3 bitini kontrol eden eger ikinci bit de sıfırsa kontrol

istemine deşti, ilk 4<sup>th</sup> bitten en az biri 1'e duran

ve 2100H bellek pozisyonu 01 yarın programı yarın.

ORG 100H

MOV [2007H], F0H

MOV BX, 2000H

TEKRAR: MOV AL, [BX]

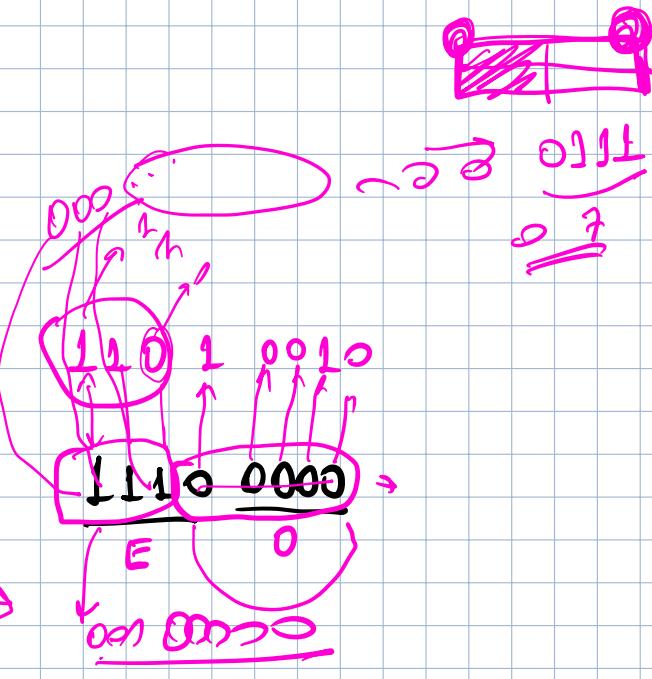
AND AL, E0H

If YOK  $x \Rightarrow f=1$

MOV CX, 02

MOV [2100H], CX

JMP S1N



YOK, INC BX

CMP BX, 200B H

JZ SON4

JMP TEKAR

SON : RET

2000 H - 200A

200B

ÖRNEK: 1000H adresi'ndeki gizendeki sayıya

1001H bellek gizendeki sayıya bıçır, bıçın sonucunu 1002H

adresi'ndeki gizine, bıçın sonucu halen okusmamış 2003H'de

adresi'ndeki gizine, bıçın sayı sıfır ise bıçın ikinci inde

hazır olusugundan bu hizmet durumunu belirtmek için

1004H adresi'ndeki gizine 1 sayısını yazan programı  
yazınız.

AKİS

(BASLA)

Düzenleme

DİYAGRAM



Bölen sayın 1000'te yinkle



Bölen sayın 1001'te yinkle



Bölen sayın Kontrol et



EVET

HAYIR

1004'e T'ye git

Son

BÖLME İZLEMI



BÖLME 1002'ye git



Kalani 1003'te yinkle

