

Furkan EKİCİ 2017555017

# MICROPROCESSORS

3 state gate: Örneğin and kapısını ele alalım:



Bu kapının akışı (c) için 3 durum vardır.

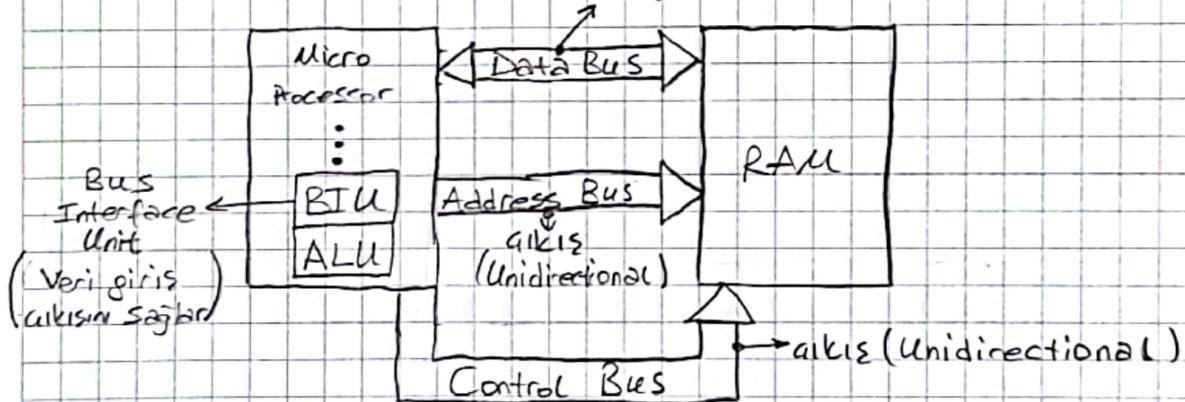
Bunlar klasik olarak 0 ve 1'in yanı sıra çıkışın yüksek empedansı (dirence) sahip olması durumudur.

$z \rightarrow 0$  ise and kapısı normal çalışır.

$z \rightarrow 1$  ise and kapısı  $z$  durumuna geçer. ( $z$  durumunu akış direncinin çok büyük olma durumudur.). Bu yüzden kapı hiç orada yokmuş gibi davranışır.

Micro Processor and RAM:

giriş/çıkış (haft yönlü) (Bidirectional)



• Adres gönderilir (Address Bus)

) Bu işlemlerin hepsini

•hangi işlemi yapacağını söyler (Control Bus)

micro işlemci yapar.

• Veriyi yollar veya alır (Data Bus)

\* Bu işlemler kablolar aracılığı ile yapılır. (Bus'ların içinde kablo var)

BIOS (Basic Input Output System): Giriş ve çıkış aygıtlarını kontrol ederek sistemin açılmasını sağlar. BIOS, kalıcı bir yazılımdır ve ROM (Read Only Memory) 'de bulunur.  
 ↳ (Yazılma olmaz sadece okunur)

EPROM (Programmable Read Only Memory): içi boş olan ROM'dur tek seferle mühşus kodlanabilir.

EPRom (Erasable Programmable Read Only Memory): Bu belleğin içine hem bir seyler yazılabılır hem bu yazıları seyler silinebilir hem de bu bilgiler silinebilir.



→ Entegre üzerinde cam bölmeye vardır ve burası kapalı olduğunda bir seyler yazılabılır. Açıktı olduğunda ve ışık alındıktan sonra zaman içindeki bilgiyi siler.

EEPROM (Electrically Erasable Programmable Read Only Memory):  
 Yukarıdaki (EPRom'daki) işi elektrik ile yapar.

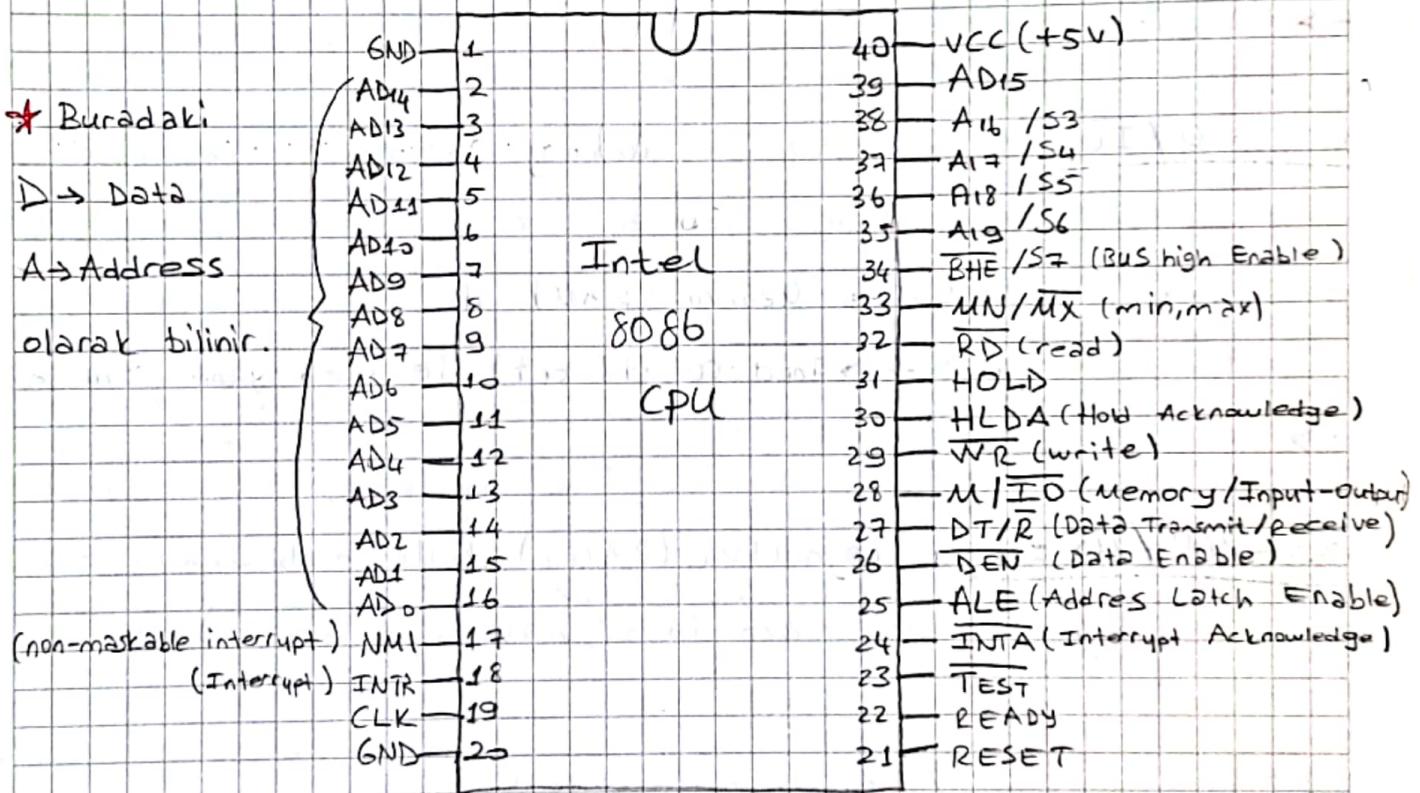
\* Flash belleklerin atası olarak bilinir.

RAM: içine verilen bilgileri enerji olmadığı sürece saklar.

↓ Buradaki random: Rastgele istediğim yere yazıp okuyabilirim veya silinebilirim anlamlarına geliyor.

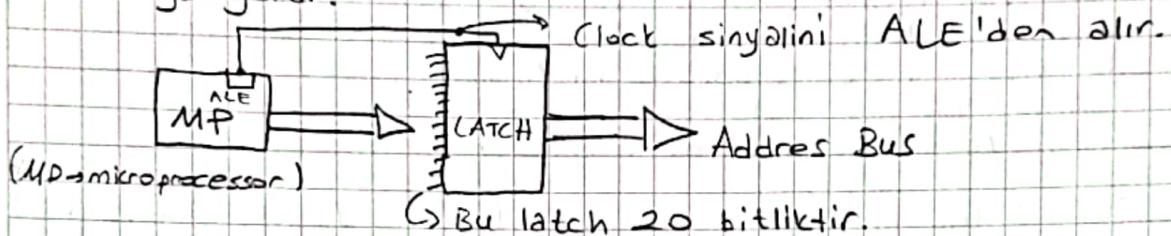
## 8086 PINOUT

- ★ 8086 işlemcisi 16 bitlik bir işlemcidir.
  - ↳ (Bir anda alıp okuyabileceği bit sayısı 16'dır)
  - ↳ (Veri yolunda 16 tane kablo vardır)
  - ↳ (Bacaklarının 16 tanesi data bus'a gider)
- ★ Maksimum 1 MB 'lık RAM'i destekler ( $2^{20} \rightarrow 1.048.576$  byte)
  - ↳ 20 bitlik adres hattı gereklidir ( $2^{20}$  den dolayı)
- ★ 40 bacaklı bir entegredir.



- ★ Hem adres hem veri yolları aynı bacakları, zaman paylaşımımlı olarak kullanırlar.
- ★ İşlemci hızı 5-10 MHz'dır.

25-ALE : Adres yolundaki adresi Latch yardımı ile saklamaya yarar.



ALE, gecerli bir adres gelirse bu adresin Latch'le yazılması için clock sinyali yollar.

→ Giristir.

28-M/I/O : Bu bacak, verinin memory ile mi yoksa input output ile mi işlem yapılacağını belirtir.

Bu bacak 1 olursa → Memory (RAM) ile

Bu bacak 0 olursa → Input-Output (Port) ile işlem yapılacağını söyler.

→ Giristir.

33-MUX : Aynı kaynakları (RAM'i) kullanan birden fazla işlemcinin (grafik işlemci, mikro işlemci, matematik işlemcisi) olup olmadığını söyler.

0 → Başka işlemcinin/İşlemcilerin olduğunu söyler.

1 → 8086 işlemcisinin tek olduğunu söyler.

→ Giristir.

31 - HOLD: DMA(Directly Memory Access) adında memoryden veri okuyan ve memory'e veri yazan unitler bulunur. Yolu kullanma talebi oluştururlar. (Min moddayken galler)

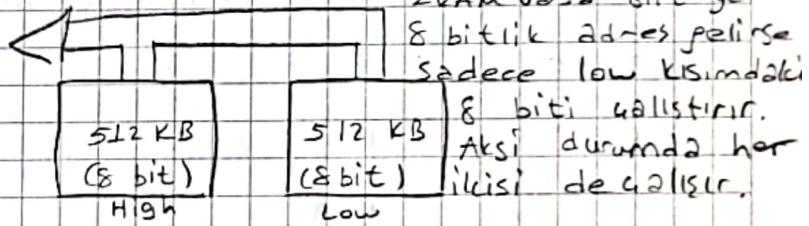
→ Gerekstir.

30 - HLDA: Oluşturan talebin kabul edildiğini bildirir.

→ Gerekstir.

34 - BHE / S7: Veri yolunun yüksek değerli baytını izintemek için kullanılır. (D15-D8).  
S7 daima 1'dir.

→ Gerekstir.



27 - DT/R: İşlemcinin veri gönderdiğini veya aldığıni bildirir.

1 olursa → Mikro işlemci veri gönderiyor

0 olursa → Mikro işlemci veri alıyor.

→ Gerekstir.

\* 1. CLK sinyalinde ne yapılacağı ve nereye yapılacağı (memory or IO) belli olur.

2. CLK sinyalinde hangi adres olduğu belli olur.

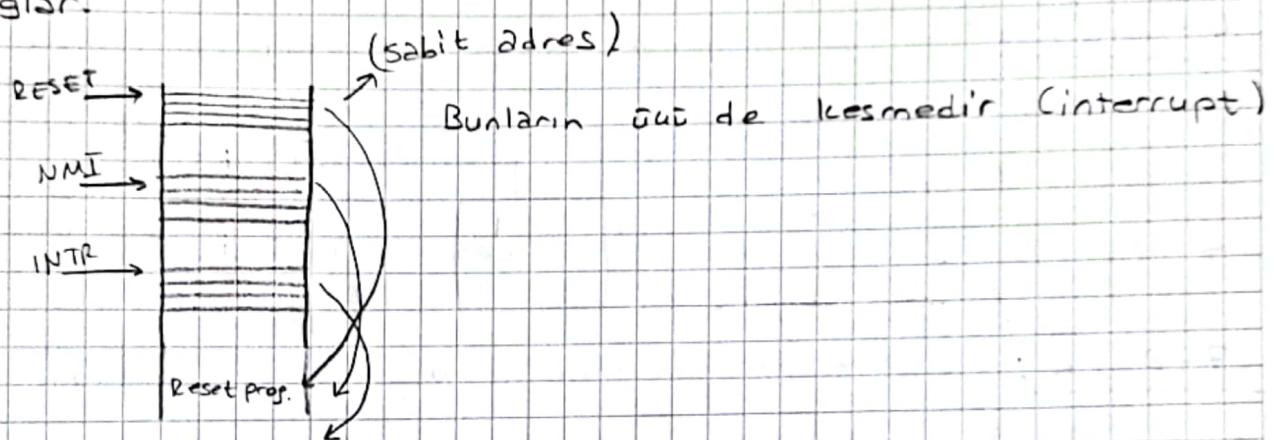
3. CLK sinyalinde RAM'den data alınır.

→ (Bu işlemlerden sonra AD barakları data bus olur)

26 - DEN: External data bus bufferlerini izlemek için kullanılır.

→ Giriş

21 - RESET: işlemci sıfırlanır. Kesme isteği (interrupt) sabit bir adresi okur ve o adresde gider bir program çalıştırır. Bu program işlemciin başlangıç durumuna gelmesini sağlar.



→ Giriş.

18 - INTR: Bunu bir örnekle anlatıcaz olursak:

Bir yazıcıya işlemci olarak 8086 taktığımızı düşünelim. Bu yazıcıdan bir şey çıkartırken kağıt sıkışırsa kesme yollar ve çıkarma işlemi durur.

INTR, RESET gibi sabit bir adres sahiptir. (Aslında bunlar pointer görevi görür). INTA'ya sürekli bir sıkıntı olduğunu ve elindeki işi bırakıp kendisine dönmesi gerektiğini bilmeli.

→ Giriş.

24 - INTA: INT<sub>R</sub> girişinin yanıtının verildiği kısımdır. INT<sub>R</sub>'ye daha fazla kesme isteğinde bulunmaması gerektiğini söyler.

→ Giristir.

17 - NMI: Yapılması gereken çok önemli bir iş yapılmakken bir hata oluştuğunda normal bir kesme ile (INT<sub>R</sub>) bu engellenemeyebilir. NMI yapılan kesmeler iş her ne olursa olsun o işlemi bitirir.

→ Giristir.

23 - TEST: "WAIT" komutu geldiğinde bu bacağı bakar. 1'den 0'a geçiste galır.

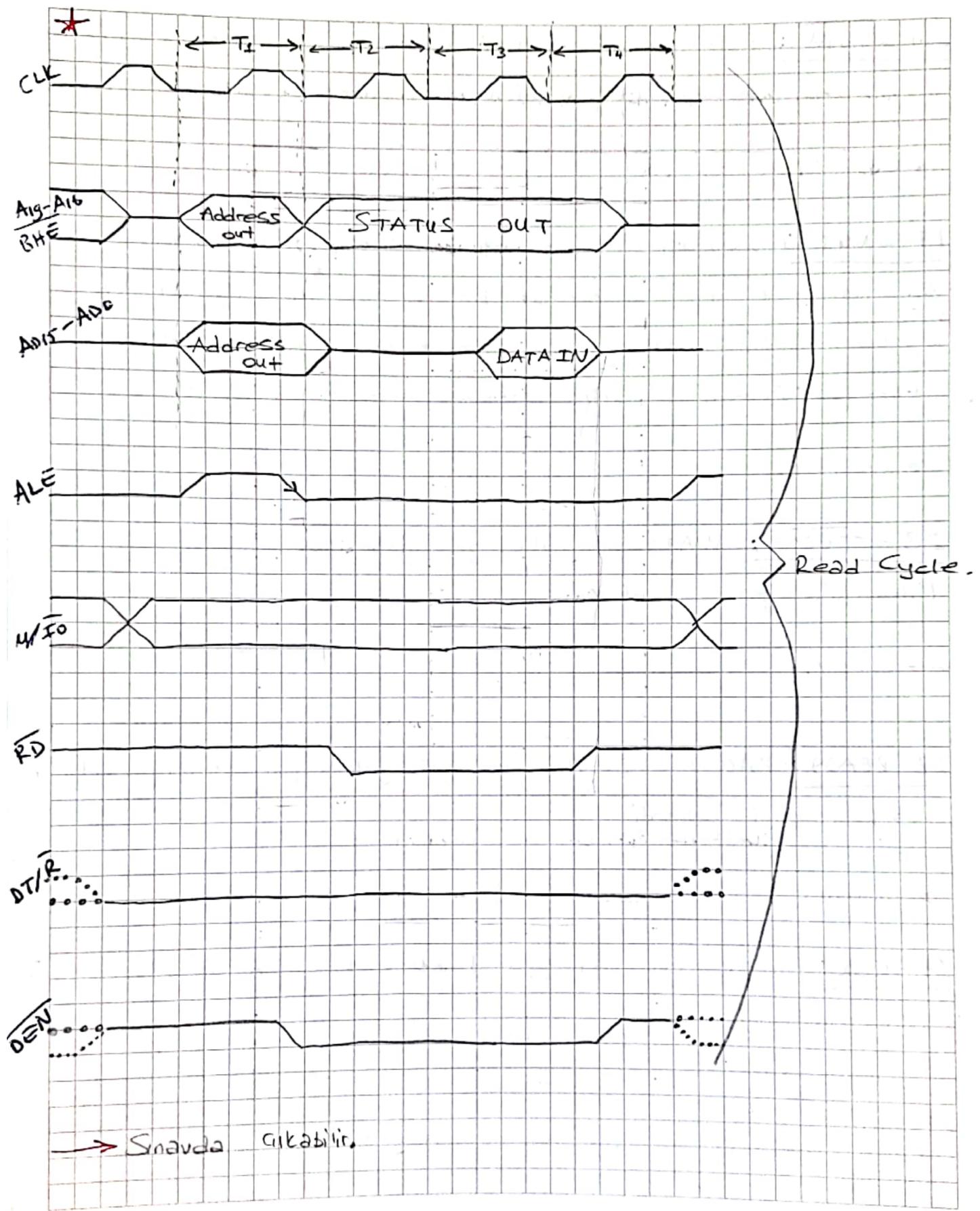
→ Giristir.

22 - READY: RAM veya IO'dan veri gelip gelmediğini işlemciye bildirir. 0 → henüz bilgi gelmedi

→ Giristir.

19 - CLK: işlemci ve veri yolu denetleyicisi için gerekli clock dördesini sağlar.

→ Giristir.



## Ortak Sinyaller (Common Signals) :

Hem maksimum hem de minimum modda kullanılabilirlerdir.

Name	Function	Type
A <sub>D5</sub> - A <sub>D0</sub>	Address/Data Bus	Bidirectional, 3-state
A <sub>19</sub> /S <sub>6</sub> - A <sub>16</sub> /S <sub>3</sub>	Address / Status	Output, 3-state
BHE/S <sub>7</sub>	Bus high enable / Status	Output, 3-state
MN/MX	Minimum / Maximum Mode control	Input
RD	Read Control	Output, 3-state
TEST	Wait on test control	Input
READY	Wait state control	Input
RESET	System reset	Input
NMI	Non-maskable interrupt request	Input
INTR	Interrupt request	Input
CLK	System clock	Input
V <sub>CC</sub>	+5V	Input
GND	Ground	

### Minimum Mod Sinyalleri:

Bu sinyaller sadece minimum madda ( $M_N/M_X = Vcc$ ) calisir.

Name	Function	Type
HOLD	Hold request	Input
HLDA	Hold Acknowledge	Output
WR	Write Control	Output, 3-state
M/I $\overline{O}$	Memory / IO Control	Output, 3-state
DT/R	Data Transmit / Receive	Output, 3-state
DEN	Data Enable	Output, 3-state
ALE	Address Latch Enable	Output
INTA	Interrupt Acknowledge	Output

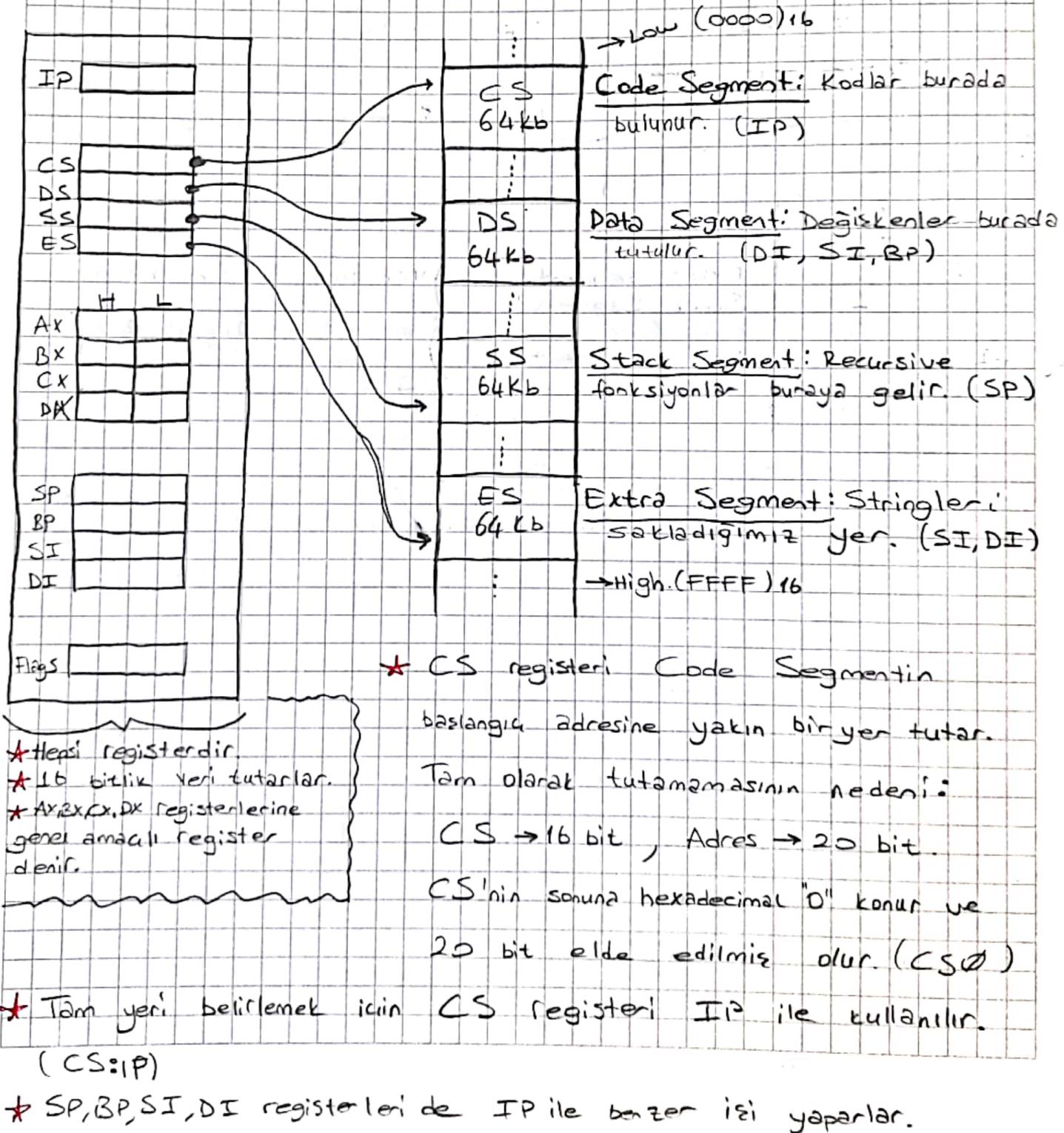
### Maksimum Mod Sinyalleri:

Bu sinyaller sadece maksimum madda ( $M_N/M_X = GND$ ) calisir.

Name	Function	Type
RQ / GT $I_O$	Request / Grant Bus Access Control	Bidirectional
LOCK	Bus Priority Lock Control	Output, 3-state
S <sub>2</sub> - S <sub>0</sub>	Bus Cycle Status	Output, 3-state
QS <sub>1</sub> , QS <sub>0</sub>	Instruction Queue Status	Output

→ Sınavda açıkabilir (min, max, common signals). Hoca tabloyu baktıktan sonra doldurmanızı isteyebilir veya direkten sorumlu olarak (hangisi min mode sinyalidir gibi) soru sorabilir.

## 8086 Kayıtları



\* CS registeri Code Segment'in başlangıç adresine yakın bir yer tutar.

Tam olarak tutamamasının nedeni:

CS → 16 bit, Adres → 20 bit.

CS'nin sonuna hexadecimal "0" konur ve 20 bit elde edilmiş olur. (CS0)

\* Tam yeri belirtmek için CS registeri IP ile kullanılır.

(CS:IP)

\* SP,BP,SI,DI registerleri de IP ile benzer işi yaparlar.

Örneğini Tam adresiniz = 7A2C2 olsun.

IP=0082 olsun. bu durumda CS = 7A24 olur.

Gözüm:

$7A24 \rightarrow 16$  bitlidir. Bunu 20 bit yapmak için  $7A24\phi$

$$\begin{array}{r} 7A24\phi \\ + 0082 \\ \hline 7A2C2 \end{array} \rightarrow \text{IP}$$

\* Buradan da anlayacağımız gibi  
IP, CS'ye hassasiyet sağlıyor.

Status

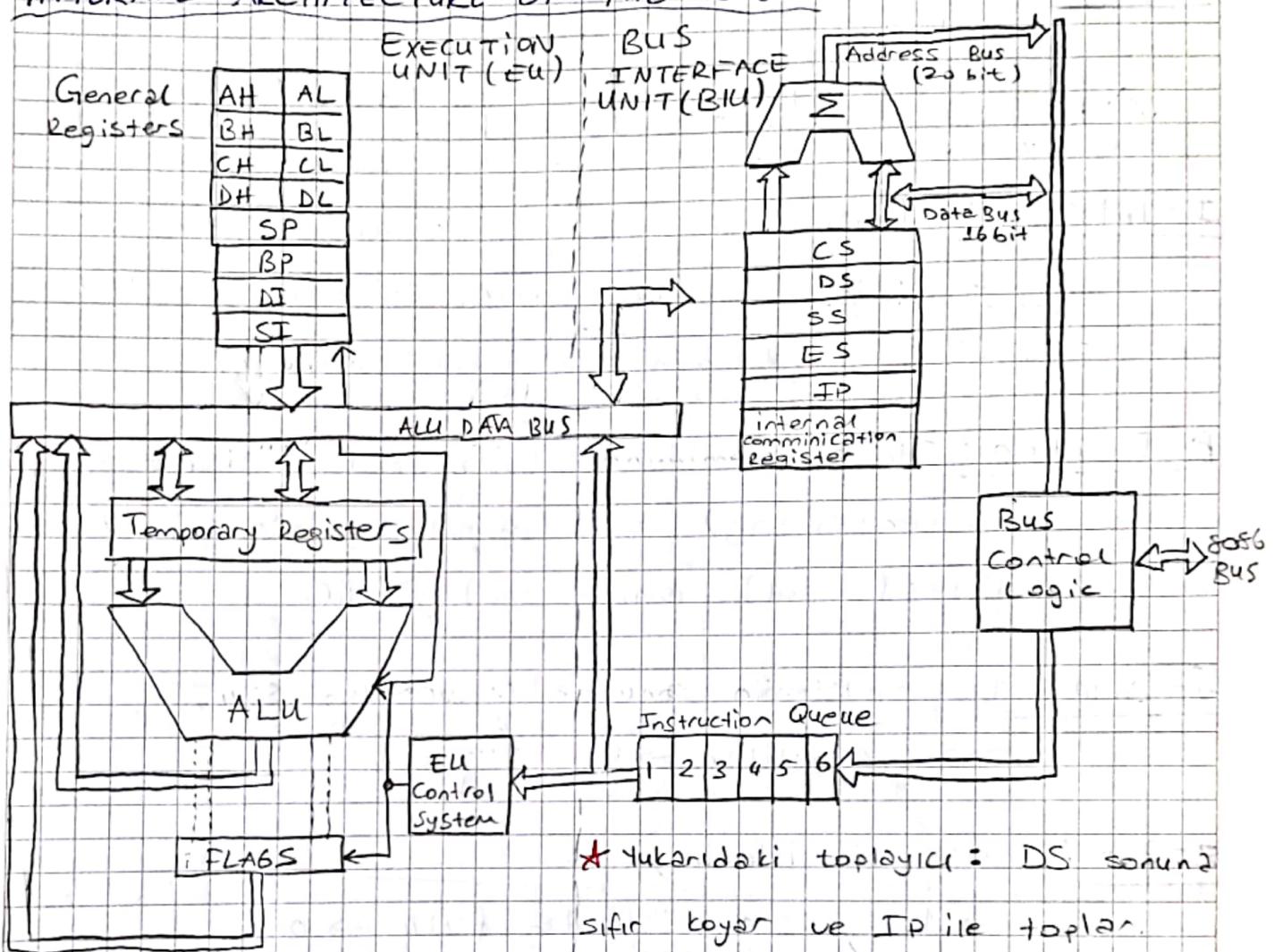
S4	S3
0	0
0	1
1	0
1	1

Extra segment  
Stack Segment  
Code/none segment  
Data Segment.

) Var olan adresin hangi  
segmentten üretildiğini  
gösterir.

3-hafta.

## INTERNAL ARCHITECTURE OF THE 8086



\* Yukarıdaki toplayıcı : DS sonunda sıfır koyar ve IP ile toplar.

\* Bayrak kaydedicisi 16 bit tir.

\* Her yürütülen komutun sonunda işlem sonucuna göre bayraklar update edilir.

\* EU Control System: Gelen kodu çözümler, işlem sırasını belirler...

Kısaca tüm organizasyonu yapar

$$\begin{array}{r} \text{DS} \\ + \text{IP} \\ \hline \text{PA} \end{array} \rightarrow \text{fiziksel adres (20 bit)}$$

\* Instruction Queue: 6 byte veri saklayabilir. EU control system datanın ne olduğunu 43zmeye çalışırken kodlar read cycle ile alınarak buraya getirilir.

Normalde bilgi aldıktan sonra EU

43zmeye çalışırken data bus boş kalır. O kodları üzerinde gelen sürede

buraya sonraki komutlar alınır ve zamanında sarruf edilmiş olur.  
8086'dan önce yok!

## FLAGS

X	X	X	X	O	D	I	T	S	z	X	A	X	P	X	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

I-Interrupt: "I" ise, sisteme bağlı harici cihazlardan gelen kesme taleplerine izin verir. "O" olduğunda NMI kesmesini durduramaz. Onun hâlicindekileri durdurur.

D-Direction: String işlemlerinde, bu bayrat "1" ise en son indisten (en sağ) başa (en sol) yönelir. "0" ise en baş indisten (en sol) sona (en sağ) yönelir.

S-Sign: Yapılan işlemin sonucunda üretilen sayı; pozitifse  $\rightarrow 0$ , negatifse  $\rightarrow 1$ .

Z-Zero: Yapılan bir işlemin, sonucu "0"  $\rightarrow 1$ , sonucu sıfırdan farklı  $\rightarrow 0$ . örneğin bir işlem sonunda Ax kaydedicisindeki değer 0000 ise zero flag 1 olur.

C-Carry: İzretsiz sayılarında yapılan toplama sonucu ekle düşmisse bu bayrat 1 olur.

\* işaretli sayılarında toplama yaparken CF etkilenebilir. Ama bunun bir anlamı yoktur.

OFFFOh ; <sup>isaretli sayılarla işlem yaparken elde olursa</sup>  
 08000h ; <sup>isaretli sayılarla işlem yaparken elde olursa</sup>  
 ↗  
7FFD ; 0111 1111 1111 0000  
 ↗  
 Sonuçta farklı operanın DF = 1, CF = 0 olur.

O-Overflow : işaretli sayılarla işlem yaparken elde olursa bu bayrağ "1" olur. İki sayı arasında işlem yaparken sayıların işaret bitleri farklı ise  $\rightarrow$  0 olur. Aynı ise ve sonucunda farklı ise +1 olur.  
A-Auxiliary Carry : 2 tane 8 bitlik sayıyı toplarken.



Sonuçta, işaretli kısımdan ok yönüne doğru elde ettiğimizde bu bayrağ "1" olur.

P-Parity : işlem sonucunda kaydedildiği mantıksal birlerin sayısı çift  $\rightarrow P=1$ , tek  $\rightarrow P=0$  olur. P flagi genelde veri iletişiminde karşılıklı verilerin güvenli iletilip, iletilmediği kontrolünde kullanılır.

T-Trap : Hata ayıklama işlemlerinde komutların adım adım işlemesi matematiksel kullanılır. "1"  $\rightarrow$  single step mode.

★Float sayıları nasıl tutulur araştır. Kesin bir soru gelecek (IEEE 754)



## ADDRESSING MODES

### - Register Addressing :

MOV AX, BX olduğunda registerden register'e işlem yapıldığından burada register addressing var denir.

Memory content

01000	8B
01001	C3
01002	XX
01003	XX

→ Bunun kod karşılığı:

MOV AX, BX

\* 8B, C3 görüldüğünde EU control unit bunu  
MOV komutu olduğunu anlar.

### Örnek değerler:

IP : 0000

CS : 0100

DS : 3100

AX : 2222

BX : 1122

$$\begin{array}{r} \text{CS:IP} \Rightarrow 01000 \\ + 0000 \\ \hline 01000 \end{array} \rightarrow \begin{array}{r} \text{CS} \\ + \text{IP} \\ \hline \text{PA} \end{array}$$

Bu adres

RAM'deki

fiyksel adresidir.

\* MOV AX, BX  
komutu okunduktan  
sonra BX değeri Axle  
kopyalanır.

komutu okunduktan  
BX değeri Axle

### - Immediate Addressing :

0000 { Sifir  
15 } 015H

Mov AL, 015H

→ Buradaki sıfırın matin  
için hiçbir önemi yoktur.  
(Immediate adres modu olduğunu gösteri)

\* MOV BX, 81234H yazarsak hata verir. ( $Bx \rightarrow 16\text{ bit}$   
 $81234H \rightarrow 20\text{ bit}$ )

\* MOV AX, 01234H → Bu immediate olduğunu gösterir. ve  
1234H'den farklıdır.

\* MOV AX, 1234H } Efektif adresidir.  
7683H } (Immediate değil)

→ [lab dersinde ] arasında yazılırlar gibi.

### - Direct Addressing:

→ constant (bir adresi tutuyor olabilir) ( $BETA = 1234H$ )

MOV CX, BETA  
 → opcode  
 {  
 }  
 → operand.

\* MOV 1234H, AX → doğru kullanımdır. (1234H eteklif adresi geçerir)

↳ AX değerini al DS=1234H adresine yaz.

DS → A123H olsun

$$\begin{array}{r} A123\emptyset \\ + 1234 \\ \hline A2464 \end{array} \rightarrow DS$$

→ Adres

A2464 → adresine AX'ı yaz.

→ (direct addressing mode)

\* MOV AX, 6817H

A2464	AL	17	yazılmaz. Çünkü böyle olması için MOV AX, 6817H
A2465	AH	68	

olması gerekiirdi (immediate addressing)

### Ex:

→ 1234H (adres)

IP: 0000	01000	8B	MOV CX, BETA
CS: 0100	01001	0E	
DS: 0200	01002	34	
	01003	12	
	01004	:	
02000			02000\emptyset
02001			+ 1234
03234	ED		03234
03235	BE		→ Bu adresdeki değerleri al CX'ın içine yaz.
		CX	
		BE	CL
			ED

Bu durumda CX = BEED olur.

## - Register Indirect Addressing:

MOV AX, [SI] → S I  
D I  
B R  
B P

Ex:

IP: 0000	01000	8B	Mov AX, [SI]
CS: 0100	01001	34	
DS: 0200	01002	12	
AX: FEED	01003		
SI: 1234			
	02000	AB	
	02001	72	
	03234	ED	
	03235	BE	

Bir komut adımı geçince değerler nasıl değişir?

$$\begin{array}{r} CS\emptyset \rightarrow 0100\emptyset \\ + IP \quad + 0000 \\ \hline PA \quad 01000 \end{array} \rightarrow \text{Bu adresten okunmaya başlayacak.}$$

(fiziksel adres)

$$\begin{array}{r} DS\emptyset \rightarrow 0200\emptyset \\ + SI \quad + 1234 \\ \hline PA \quad 03234 \end{array} \rightarrow \text{Bu adreseki değeri AL AX'in içine yaz.}$$

Daha sonra (fiziksel adres)

Bir komut satırı ilerledikten sonra son durum şu şekilde dir:

IP: 0002  
CS: 0100  
DS: 0200  
AX: BEED  
SI: 1234

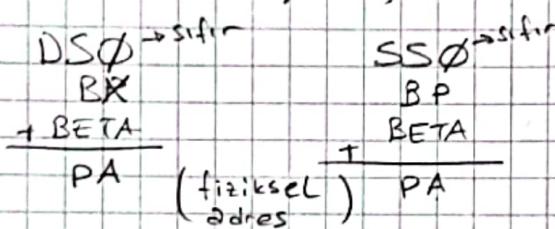
## + Based Addressing :

$\begin{array}{c} BX \\ BP \end{array}$  constant.

MOV [BX].BETA, AL (AL değerini belirtilen yere aktar)

\* [] içinde base registerleri olabilir. (BX, BP)

\* BX  $\rightarrow$  DS ile, BP  $\rightarrow$  SS ile gelir.



\* Aslında burada array işlemi yapılmaktadır.

## Ex:

IP: 0000	01000	88	} MOV [BX].BETA, AL
CS: 0100	01001	07	
DS: 0200	01002	34	
AX: 16723	01003	12	
BX: 1000	01004	XX	Next instruction
		:	
	02000	72	
	02001	64	
		:	
	03234		
		:	
	04234	3623	$\rightarrow$ Arrayin başlangıç adresi ( $\frac{DS\phi}{+ BETA}$ )
	04235	15	
		:	

BX olduğu için DS ile kullanacağız;

$$\begin{array}{c} DS\phi \\ BX \\ + BETA \end{array} \quad \begin{array}{c} 0200\phi \\ 1000 \\ + 1234 \end{array} \quad \boxed{04234}$$

MOV [BX].BETA, AL  
 $\rightarrow$  AL 'deki' değeri ( $\frac{AH}{AL}$  67 23) al  
 04234 adresine yaz.

## - Indexed Addressing:

MOV AL, BETA[SI]

SI  
DI

\* DI veya SI'yi, DS ve BETA değerleri ile toplayınca fiziksel adresi ulaşıyoruz. (PA)

\* Efektif Adres = (SI) + ARRAY

Ex:

IP: 0000  
CS: 0100  
DS: 0200  
SI: 2000  
AX: 0723 BE

01000	8A
01001	44
01002	34
01003	12
01004	XX
05234	BE
05235	ED

MOV AL, BETA[SI]

$$\begin{array}{r} \text{DS} \oplus \rightarrow 0200 \oplus \\ \text{SI} \quad 2000 \\ \text{BETA} \quad + 1234 \\ \hline \text{PA} \quad \boxed{05234} \end{array}$$

→ Bu adressteki değeri AL'ye yaz.

Bir komut adımı geçtikten sonra son durum şu şekilde dir,

IP: 0004  
CS: 0100  
DS: 0200  
SI: 2000  
AX: 07BE

\* Hexadecimal sayıları toplamayı öğren!

(1) elde → Toplamları 17 yapıyor.  
 Toplayınca 8F  
 15 yap + 62  
 $\frac{+ 62}{F\ 1}$   
 (16+1) elde

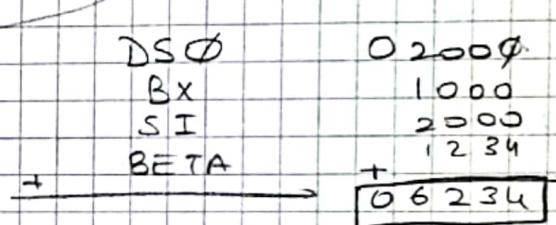
## - Based Indexed Addressing:

MOV AH,[BX].BETA[SI]  
 ↓  
 Sadece  
 BX olabilir.

★ İki boyutlu arraydir.

Ex:

IP: 0000	01000	8A	} MOV AH,[BX].BETA[SI]
CS: 0100	01001	20	
DS: 0200	01002	34	
AX: 1234	01003	12	
BX: 1000	01004	XX	
SI: 2000			
	06234	BE	



→ Bu adressteki değeri AH'ye ata.

Bir işlem yürütülüğünde son durum şu şekilde olur.

IP : 0004  
 CS : 0100  
 DS : 0200  
 AX : BE34  
 BX : 1000  
 SI : 2000

String Addressing:

MOV\$ Komutu olduğunda string addressing olur.

Port Addressing:

IN OUT Komutları varsa port addressing olur.

## INSTRUCTION SET OF 8086 (8086 Komut Seti)

Instruction: (Talimat Komut): Mikroişlemcinin içinde tanımlanmış ve beli görevleri yapan fonksiyonlardır.

Opcode: Operasyon kod anlamına gelir. CPU tarafından gerçekleştirilecek işlem türünü belirtir. Örneğin;

08, MOV X,Y 'nin opcode'udur. (Yani yazılan kodun sayıya çevrilmiş hali)

Operand: Komutların alacağı değerlerdir. 8 bit veya 16 bit olabilir.

Assembler: Komutu İtilik tabana çevirir. Böylece kod işlenici tarafından okunabilir.

Mnemonics: Bunlar fonksiyonların sembolik kodlarıdır. Örneğin;

ADD, SUB, XCHG, MOV ...

### -MOV instruction : (Move)

Kullanım: MOV D,S

Algoritma: (S) → (D)

Source                    destination

Etkilediği Bayraklar: Highbini

İşi: S değerini D değerinin içine kopyalamak.

- MOV register/memory      MOV BX, 0210H
- MOV register/register    MOV AL, BL
- MOV memory/Immediate    MOV [DI], 01231H
- MOV register/Immediate    MOV AL, 011H
- MOV accumulator/Memory    MOV AL, [SI]
- MOV segment/memory        MOV SS, [DI]
- MOV register / segment     MOV DX, SS

\*

MOV CS, DS X (Bu kullanım yanlışdır).

MOV BL, BX X (Yanlıştır, BL → 8 bit, BX, 16 bit)

MOV [SI], [BX] X (Yanlıştır, Memoryden memorye aktarım yoktur!)

\* MOV DS, 05000H X (Direkt olarak aktarım yapılamaz).

Bunun yerine;

MOV AX, 05000H

MOV DS, AX ; kullanılır

Örnek:

IP: 0100	01100	8C	})	MOV DX, CS
CS: 0100	01101	CA		
DS: 0200	01102	XX		next instr.
DX: XXXX				

\* CS0+IP yapılarak kodun nereden oturmayıza başlayacağına karar verilir.

→ Bir komut adımı islandığında ;

IP: 0102 → 2 artmasının nedeni: MOV komut 2 byte'lidir.  
CS: 0100  
DS: 0200  
DX: 0100

- XCHG instruction : (Exchange)

Kullanım: XCHG D, S

Algoritma: (D)  $\leftrightarrow$  (S)

Etkilediği Bayraklar: Hiçbiri

isi: S değerini D'ye, D değerini S'ye aktarır.

- XCHG accumulator/register 16 XCHG AX, DX
- XCHG register/register XCHG AL, CL
- XCHG memory/register XCHG [BX], DX

\* Bu komutta da Mov'da olduğu gibi iki memory arasında işlem yapılamaz.

\*

XCHG register/memory XCHG AL, [BX] ✓ (Doğru Kullanım)

Örnek:

IP: 0101	01101	87	} XCHG SUM, BX → SUM → Önce CS0+IP yapılarak kodun nereden okunmaya başlanacağı bulunur. Daha sonra DS0+1234 yapılarak fiziksel adres bulunur. Bu fiziksel adrese gidilenek oradaki değer BX'e, BX'teki değer de bu fiziksel adrese kopyalanır.
CS: 0100	01102	1F	
DS: 0200	01103	34	
BX: 11 AA	01104	12	
Adres Adresindeki değeri ↓ SUM: [1234H] [1234H]: 00FF	01105	XX	
	03234	FF	
	03235	00	

→ Bir komut adımı istendiğinde;

IP: 0105  
CS: 0100  
DS: 0200  
BX: 00FF  
SUM: [1234H]  
[1234H]: 11 AA  
1235H 1234H

\*  $CS0+IP = 01000 + 0101 = 01101 \rightarrow$  kod buradan okunur.

$DS0+SUM = 02000 + 1234 = 03234 \rightarrow$  fiziksel adres.

Bu fiziksel adresteki değer BX' e kopyalanacağından 03235 adresindeki değer de alınır. Çünkü BX → 16 bit her bir hafıza balmesi → 8 bit'tir.

## -XLAT instruction : (Translate)

Kullanım: XLAT

Algoritma:  $((AL) + (BX) + (DS)O) \rightarrow (AL)$

Etkilediği Bayraklar: Hiçbiri

isi: AL'deki, BX'teki değerleri ve DS'nin sonuna sıfır ekleyerek oluşturulan sonucu topla ve bir fiziksel adres elde et. Bu fiziksel adresdeki değeri al ve AL'ye yaz.

Binary ve Graycode:

Binary bir sayıyı graycode'a çevirirken;

Örneğin; (6)<sub>10</sub>

$6 \rightarrow (0110)_2$  birinci biti aynen yaz. Sonra 1. bit ve 2. biti xor işlemine koyup 2. bit'i elde et. 2 ve 3'ü xorlayarak 3. biti, 3. ve 4.'ü xorlayarak 4. bit'i elde et.

$$\begin{array}{ccccccc}
 0 & \oplus & 1 & \oplus & 1 & \oplus & 0 \\
 \downarrow & & \downarrow & & \downarrow & & \downarrow \\
 0 & 1 & 0 & 1 & & & \\
 \end{array} \quad \text{Binary (6)}$$

$$\begin{array}{ccccccc}
 & & & & & & \\
 & & & & & & \\
 \end{array} \quad \text{Graycode (6)}$$

Graycode bir sayıyı binary yaparken;

Birinci biti aynen yaz. Binary sayının ilk biti ile graycode'un 2. bitini xorlayıp binarynin 2. biti olarak yaz. Binary 2. bit, graycode 3. biti yolla 3. binary bitini yaz. Binary 3. bit ve graycode 4. biti yolla 4. binary bitini yaz.

$$\begin{array}{ccccc}
 0 & 1 & 0 & 1 & \rightarrow \text{graycode (6)} \\
 \downarrow \text{xor} \quad \downarrow \text{xor} \quad \downarrow \text{xor} \quad \downarrow \text{xor} \\
 0 & 1 & 1 & 0 & \rightarrow \text{binary (6)}
 \end{array}$$

**★ BCD code 'da sayıların binary değerleri aynı kalır.**

Örneğin ;  $(9)_{10} \rightarrow (1001)_2$

$(1001)$  <sub>binary</sub>  $\rightarrow (1001)$  <sub>BCD</sub>

**★ Excess-3 code 'da sayıları 3 ile toplandıktan sonra binary olarak yazılır.**

Örneğin ;  $(4)_{10} \rightarrow (100)_2$

$4+3=7$  olacağinden.

$(\overbrace{0+00}^4)$  <sub>binary</sub>  $\rightarrow (\overbrace{0111}^7)$  <sub>Excess-3</sub> olur.

**★ 7 Segment code 'da .**

f | b  
e | g | c  
d |  
  
Örneğin ;  $(3)_{10}$  yazmak isterseniz.  
0 a b c d e f g  
0 1 1 1 1 0 0 1  
7 9

7 Segment 'te 3'ün karşılığı  $(79)$  <sub>Hexadecimal(16)</sub> 'dir.

**→ Bu ters olarak yazılabilir. (3 için) (Bu daha çok tercih ediliyor)**

0 g f e d c b 2  
0 1 0 0 1 1 1 1  
4 F

0 zaman 3'ün değeri  $(4F)$  <sub>Hexadecimal(16)</sub> olur.

Örnek:

10100	00	10200	03	10300	3F
10101	01	10201	04	10301	06
10102	03	10202	05	10302	5B
10103	02	10203	06	10303	4F
10104	06	10204	07	10304	66
10105	07	10205	08	10305	6D
10106	05	10206	09	10306	7D
10107	04	10207	OA	10307	07
10108	0C	10208	OB	10308	7F
10109	OD	10209	OC	10309	6F
1010A	OF	1020A		1030A	
1010B		1020B		1030B	

GRAY

EXCESS-3

7 SEGMENT  
(Ters Atanır)

MOV AX, 01000H  
 MOV DS, AX  
 MOV DI, 02000H  
 MOV BX, 00100H  
 MOV AL, 007H  
 XLAT  
 MOV [DI], AL  
 INC DI

AH: 10 AL: 00  
 DS: 1000H  
 DI: 2000H  
 BX: 0100H  
 AL: 07H

10000 + 0100 + 07 = 10107 → AL: 04  
 10000 + 2000 = 12000 adresine AL değerini yaz. (graycode)  
 DI'yi 1 artır. DI: 2001H

adresdeki  
değerin 111D  
değerinin 111D  
AL'ye aktar.

MOV BX, 00300H  
 MOV AL, 007H  
 XLAT  
 MOV [DI], AL  
 INC DI

BX: 0300H  
 AL: 07H

10000 + 0300 + 07 = 10307 → AL: 07  
 10000 + 2001 = 12001 adresine AL değerini yaz (7 segment)  
 DI = DI + 1 • DI: 2002H

adresdeki  
değerin 111D  
değerinin 111D  
AL'ye aktar.

MOV BX, 00200H  
 MOV AL, 007H  
 XLAT  
 MOV [DI], AL

BX: 0200H  
 AL: 07H

10000 + 0200 + 07 = 10207 → AL: OA  
 10000 + 2002 = 12002 adresine AL değerini yaz. (Excess3)

adresdeki  
değerin 111D  
değerinin 111D  
AL'ye aktar.

\* Sırayla (7)H'un gray code, 7 segment ve excess 3 değerleri bulunur.

\* Renkli kısımdan sonrakiler cevaplıdır.

Soru: GRAY, EXCESS-3, 7 SEGMENT ve kod kısımlıdır.

-LEA instruction: (Load effective address)

Kullanım: LEA register(16bit), Efektif Adres

Algoritma: EA  $\rightarrow$  register (16 bit)

Etkilediği Bayraklar: Hiçbiri

İşİ: Normalde efektif adres olan bir değeri normal adresmiş gibi 16 bitlik register'a aktarır.

Örnekler:  $\rightarrow$  Direct addressing mode.

1  $\rightarrow$  MOV AX, 1234H demek 1234H' a git oradaki değeri al (2 byte olacak şekilde) sonra bu değeri AX'e aktar.

2  $\rightarrow$  LEA AX, 1234H yazarsak  $AX = 1234H$  oluyor.

Yani bir bakıma direct addressing modu, immediate addressing mode çeviriyoruz. Yani bu MOV AX, 01234H ile aynı şey)

\* emu8086'da ;

MOV AX, [1234H] yazmak 1 (İşlem sonunda AX:0000)

LEA AX, [1234H] yazmak 2. (İşlem sonunda AX:1234)

Örnek:

MOV BX, 035H

MOV DI, 012H

LEA SI, [BX+DI]

İşlem sonunda SI = 47H olur.

-LDS instruction : ( Load register and DS )

Kullanım : LDS register(16), memory(32 bit)

Algoritma : memory(32)  $\rightarrow$  register(16)  
memory(32+2)  $\rightarrow$  DS

Etkilediği Bayraklar: Hiçbir!

İşİ: Verilen hafıza alanındaki 16 bitlik kismı (2 bölmeli)  
( $DS \oplus$  hafıza alanı)  
16 bitlik register'la aktarır. 2 adımla ilerleyerek diğer 16  
bitlik alanı DS'ye aktarır.

Örnek :

IP: 0100	01100	C5	LDS SI, [200H]
CS: 0100	01101	36	
DS: 0200	01102	02	
SI: ????	01103	00	
...	01104	XX	Next inst.

02200	20
02201	00
02202	00
02203	03

$\Rightarrow$  Öncelikle CS:IP yaparak kodun okunacağı yeri belirle.

$$CS \oplus IP = 0100 \oplus 100 = \boxed{01100}$$

$\Rightarrow$  DS : [200H] adresini bul

$0200 \oplus 200 = \boxed{02200}$  fiziksel adres buradan başlayarak 2 byte'lık  
kismı SI'ya, SI:0020, diğer 2 byte'lık kismı DS'ye  
DS:0300 yazar.

$\Rightarrow$  Bu işlemler bittiğinde

IP: 0104  
CS: 0100  
DS: 0300  
SI: 0020

- LES instruction: (Load register and ES)

Kullanım: LES register(16), memory(32)

Algoritma: memory(32) → register(16)  
memory(32+2) → ES

Etkilediği Bağraklar: Hıtabı

isi: Verilen hafıza alanındaki 2 byte'lik kısmı 16 bitlik  
(DS0+hafıza alanı)  
register'e, diğer 2 byte'lik kısmı (toplamda 32 bitlik kism)  
ES'ye aktarır.

\* Kullanımı LDS ile aynıdır.

### Arithmetic Instructions

- Addition ADD, ADC (AAA, DAA)
- Subtraction SUB, SBB (AAS, DAS)
- Multiplication MUL, IMUL (AAM)
- Division DIV, IDIV (AAD, CBW, CWD)
- Negation NEG
- Increment INC
- Decrement DEC

## - ADD instruction: (Addition)

Kullanım: ADD D, S

Algoritma: (S)+(D) → (D) (Elde varsa Carry → (CF))

Etkilediği Bayraklar! Hepsı (C, Z, S, O, P, A)

isi: S değeri ve D değerini topla, D'ye yaz.

\* Burada yine ADD memory, memory yapamıyoruz.

\* MOV komutu için olası kurallar burada da geçerli.

• ADD register, register    ADD AL, BL    ( $AL = AL + BL$ )

• ADD register, memory    ADD CL, [BP]    ( $DS0 + BP$  üzerindeki değeri ile CL'yi topla sonuc CL'ye yaz) ( $CL'$ yi topla sonuc  $CL'$ ye yaz)

• ADD memory, register    ADD [BX], AL    (Yukarıdaki gibi yap amma bu kez  $DS0 + BX$ 'e yaz sonucu)

• ADD register, immediate    ADD BX, 013H    ( $BX = BX + 13H$ )

• ADD memory, immediate    ADD [DI], 024H    ( $DS0 + DI$  üzerindeki değeri 24H'e toplayıp  $DS0 + DI$ 'ya yaz)

\* ADD BX, [SI+2]

↳  $DS0 + SI + 2$  yapıp fiziksel adresle git. O fiziksel adresdeki değeri BX'yi topla sonucu BY'e yaz.

\* ADD AL, ARRAY[SI]

↳  $DS0 + ARRAY + SI$  yapıp fiziksel adresle git. O adresdeki değeri AL'ile toplayıp sonucu AH'ye yaz.

\* ADD komutu half-adder olarak adlandırılabilir.

\* 8 bitlik bir register ile 16 bitlik bir register toplanamaz !!! (ADD BL, CX X yanlışlık)

## - ADC instruction (Add with carry)

Kullanım: ADC (D), (S)

Algoritma: (D) + (S) + CF  $\rightarrow$  D (Ede varsa Carry  $\rightarrow$  (CF))

Etillediği Bayraklar: Hepsı (CF SOFA)

isi: D, S ve Carry Flag'i toplar ve D'ye yazır sonucu.

\* ADD'deki kurallar burada da geçerli.

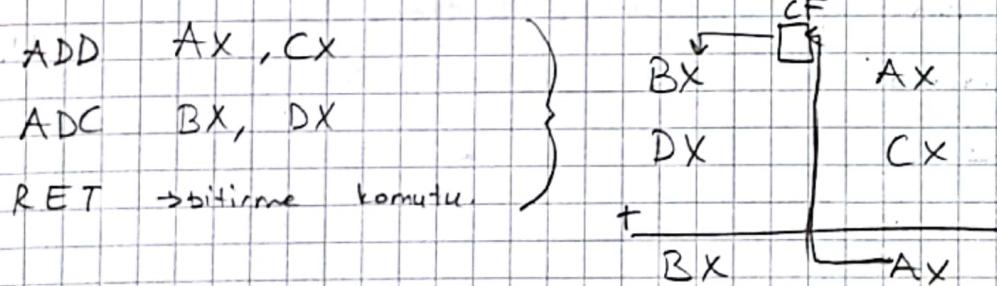
• ADC AL, AH      AL = AL + AH + CF

- ADC BX, [BP+2]

↳ SS $\oplus$  + BP + 2 fiziksel adresindeki değer, BX değeri ve CF'yi topla BY'e yaz. (BP olduğunu için SS)

Örnek:

\* 2 tane 32 bitlik sayıyı topla ((BX, AX) ve (DX, CX) ölmek üzere 2 ayrı 32 bitlik sayı) (Sonucu BX, AX'te tut)



\*

MOV AX, 0111H  
MOV BX, 0222H  
MOV CX, 0333H  
MOV DX, 0444H

İçerideki 32 bitlik sayı: 22221111  
İkinci 32 bitlik sayı: 44443333

+  
6666 4444

ADD AX, CX ; bu izlem sonunda CF=0 (ede yok)

ADC BX, DX

RET

## -INC instruction: (Increment by 1)

Kullanım: INC D

Algoritma:  $(D) + 1 \rightarrow (D)$

Etkilediği Bayraklar: ZSDPA etkilenir CF etkilenmez.

isi: D yi bir artırıp D'ye kaydeder (C'deki ++ operatörü)

• INC BL       $BL = BL + 1$

• INC SP       $SP = SP + 1$

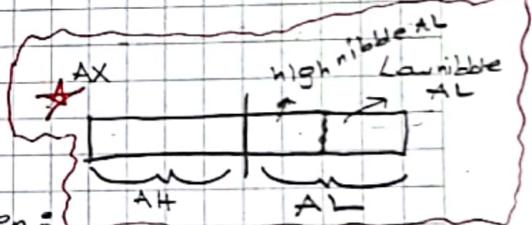
• INC DATA1       $DS0 + DATA1$  fiziksel adresine git. Bu adres-teki değeri 1 artır ve bu adres'e kaydet.

\* D → register(8), register(16) ve memory olabilir.

## -AAA instruction: (ASCII adjust for addition)

Kullanım: AAA

Algoritma: if low nibble of AL > 9 or AF = 1 then



•  $AL = AL + 6$

•  $AH = AH + 1$

•  $AF = 1$

•  $CF = 1$

else:

•  $AF = 0$

•  $CF = 0$

→ler iki durumda da AL'nin high nibble'i silinir.

Etkilediği Bayraklar: AF, CF degisebilir. Diğerleri de degisebilir ama

dikkate alınmaz (undefined)

isi: AL registerinin Low nibble'ı ile bir sayıyı decimal(10)

olarak toplamaya yarar. Eğer toplam 9 dan büyükse ilk

haneyi AH'ye, ikinci haneyi AL'ye yazar.

Örnek:

Mov AH, 03H

AH      AL  
high nibble  
↓  
0 0 | 3 | 7

ADD AL, 056H

AL  
low nibble  
↓  
0 0 | 8 | D

AAA

; Bu yazıldıkten sonra aşağıdaki adımlar uygulanır:

AL'ın low nibble',  $\rightarrow 7$  (high nibble = 3)

AL'ye eklenen sayının low'u  $\rightarrow 6$

$$\begin{array}{r} 10 \\ + 1 \\ \hline 11 \end{array}$$

Sonuç  $> 9$  olduğundan.

- $AL = DH + BH = 13H \rightarrow$  Bunun high nibble'ini atıp AL'ye yaz.
- $AH = AH + 1$
- $CF = 1$
- $AF = 1$

$\rightarrow$  Bu işlemlerin sonucunda  $AH = 01$   $AL = 03$  olur.

\* Yukarıdaki kodda AAA'nın altına ADD AX, 03030H yazılırsa AH ve AL'de sayıların ASCII karşılıkları bulunur.

\* Bu işlem için AL registeri kullanılır.

-DAA instruction: (Decimal adjust for addition)

Kullanım: DAA

Algoritma: if low nibble of AL > 9 or AF = 1 then :

- $AL = AL + 6$
- $AF = 1$

if  $AL > 9Fh$  or  $CF = 1$  then :

- $AL = AL + 60h$
- $CF = 1$

Etkilediği Bayraklar: C, Z, S, P, A etkilenir. OF etkilemeyebilir

fakat dikkate alınmaz. (undefined)

isi: Decimal toplama yapar.

high nibble  
low nibble  
 $\Rightarrow C$        $AL < 9Fh$   
 $C > 9:$        $\Rightarrow C$   
 $AL = AL + 6$   
 $\Rightarrow 82$

Örnek:

Mov AX, 053H ; AH:00      AL: 53  
ADD AL, 029H ; AH:00      AL: 70  
DAA ; AH:00      AL: 82       $(53+29)$

Örnek:

Mov AX, 099H ; AH:00      AL: 99  
ADD AL, 034H ; AH:00      AL: CD  
DAA ; AH:00      AL: 33       $\begin{array}{l} \text{high} \\ \text{low} \\ \Rightarrow C(D>9) : \\ \text{AL} = \text{AL} + 6 \Rightarrow D3 \\ \text{AL} > 9Fh \\ \text{AL} = \text{AL} + 60h \rightarrow D3+60h \\ \text{AL} = 33 \end{array}$

\* ikinci if'yi kontrol etmemi unutma ( $AL > 9Fh$ )

\* Unpacked BCD number

59      direct olarak  
binary karşılığı  
(0011 1011)  
11

Packed BCD number

59  
↓ ↓  
(0101 1001)

## -SUB instruction: (Subtract)

Kullanım: SUB D,S

Algoritma:  $(D) - (S) \rightarrow (D)$  (Borrow  $\rightarrow CF$ )

Ettilediği Bayraklar: Hepsisi (CF SOPA)

isi:  $(D)$  'den  $(S)$  'yi çıkarıp  $D$  'ye yazar. \* (Sıralarına dikkat et)

\* D ve S değerleri ADD ile aynıdır.

Örnek:

Mov CH, 022H  
SUB CH, 044H  
; CH: DE oluyor.

$$\begin{array}{r}
 \begin{array}{c} 2 & 2 \\ 00100010 \\ 40100100 \\ \hline 1101110 \end{array} \\
 \text{Sign bit} \quad \text{D} \quad \text{E} \\
 \text{bit} \quad \text{bit} \quad \text{bit}
 \end{array}$$

sign (PF'ye dikkate alma!)

C = 1 (borrow)  
 Z = 0 (result not zero)  
 S = 1 (result negative)  
 O = 0 (no overflow)  
 P = 1 (even parity)  
 A = 1 (half borrow)

## -SBB instruction: (Subtract with borrow)

Kullanım: SBB D,S

Algoritma:  $(D) - (S) - (CF) \rightarrow (D)$

Ettilediği Bayraklar: Hepsisi (CF SOPA)

isi: D 'den S ve CF 'yi çıkarıp D 'ye yazar.

Örnek:

STC  
 MOV AL, 005H  
 SBB AL, 003H

Set CF yani  $CF=1$  yapar  
 ; AH:00 AL:05  
 ;  $AL = 05 - 03 - 1 = 1 \Rightarrow AH:00 AL:01$

\* D ve S değerleri ADD ile aynı

## -DEC instruction: (Decrement by 1)

Kullanım: DEC D

Algoritma:  $(D) - 1 \rightarrow (D)$

Etkilediği Bayraklar: ZSOPA etkilendir. CF etkilenemez!

İşii: D değerini bir azaltıp D'ye atar. (C'deki -- operatörü)

### \* DEC NUMB

→ DS0+NUMB adresine git. Oradaki değeri bir azalt ve tekrar oraya yaz.

\* (D) → Reg(16), Reg(8), Memory olabilir.

## -NEG instruction: (Negate)

Kullanım: NEG D

Algoritma:  $0 - (D) \rightarrow (D) \quad (1 \rightarrow (CF))$

Etkilediği Bayraklar: Hepsи (CZSOPA)

İşii: D'nin negatifini alır. ( $2^8$  li komplement)

↳ bitlerin tensini yazıp 1 ekler.

\* (D) → Reg ve Memory olur.

### Örnek:

Mov AL, 005H ; AL=05h → 0000 0101 ) I. NEG  
 NEG AL ; AL=OFBh  
 NEG AL ; AL=05h

$$\begin{array}{r}
 0 \qquad \overline{5} \\
 \begin{array}{r}
 0000 \quad 0101 \\
 + \quad \quad \quad \downarrow \\
 1111 \quad 1010 \\
 \hline
 \end{array} \\
 \begin{array}{c}
 F \\
 \hline
 \end{array} \quad \begin{array}{c}
 B \\
 \hline
 \end{array} \quad ) \text{II. NEG}
 \end{array}$$

\*  $-5 = 1011$   
bu nun basına gelen türün  
önesi yok

$$1111 \quad 1011 = 1011$$

## - DAS instruction: (Decimal adjust for subtraction)

Kullanım: DAS

Algoritma: if low nibble of AL > 9 or AF = 1 then:

- $AL = AL - 6$
- $AF = 1$

if  $AL > 9Fh$  or  $CF = 1$  then:

- $AL = AL - 60h$
- $CF = 1$

Etkilediği Bayraklar: C2, S, P, A etkilenir. OF → undefined (dikkate alınmaz)

İşİ: Decimal çıkarma yapar. (2 tanesi packed BCD sayı ile)

★ DAA komutu ile benzer.

Örnek:

```

MOV AX, 051H ; AH:00   AL: 51
SUB AL, 017H ; AH:00   AL: 3A
DAS           ; AH: 00   AL: 34

```

$A - 6 = 4$

low nibble > 9

## - AAS instruction: (ASCII adjust for subtraction)

Kullanım: AAS

Algoritma: if low nibble of AL > 9 or AF = 1 then:

- $AL = AL - 6$
- $AH = AH - 1$
- $AF = 1$
- $CF = 1$

else

- $AF = 0$
- $CF = 0$

→ Her iki durumda da AL'nin high nibble'ı atılır.

Etkilediği Bayraklar: AF, CF etkilenir. Diğerleri undefined. (≠ SOP)

İşİ: AAA ile benzer bir işi yapar.

Örnek:

```

MOV AX, 002FFh ; AH:02   AL: FF
AAS           ; AH: 01   AL: 09

```

$\cancel{FF}$

$$F > 9 \text{ so } AL = F - 6 = 9 \\ AH = AH - 1$$

## - MUL instruction : (Multiply (unsigned))

Kullanım: MUL S (Reg 8, Reg 16, Mem8, Mem16)

Algoritma: (AL). (S8)  $\rightarrow$  (AX)   
 (AX). (S16)  $\rightarrow$  (DX), (AX)

Etkilediği Bayraklar: OF, CF  $\rightarrow$  etkilendir. ZSPA undefined

isi: S  $\rightarrow$  8 bitlik ise : S ile AL'yi carp AX'e yaz.  
 S  $\rightarrow$  16 bitlik ise : S ile AX'yi carp DX ve AX'e yaz.

$\hookrightarrow$  MUL CL ; AL ile CL'yi carp AX'e yaz.

$\rightarrow$  MUL CX ; AX ile CX'i carp DX-AX'e yaz.

$\rightarrow$  MUL TEMP ; AL ile TEMP adresindeki değeri carp AX'e yaz.

Örnek:

MOV AL, 00Ah ; AH:00 AL:0A  
MOV BL, 003h ; BH:00 BL:03  
MUL BL ; & bit multiplication  
; AX:00 AL:1E

$$\begin{array}{r}
 \begin{matrix} \textcircled{1} \text{ normalde } 30 \\ \times 03 \\ \hline 1E \end{matrix} \\
 \begin{matrix} 0A \\ \uparrow 30 - 16 = 14 \rightarrow E \\ \times 03 \\ \hline 1E \end{matrix}
 \end{array}$$

## - IMUL instruction: (Integer multiply (signed))

Kullanım: IMUL S (Reg 8, Reg 16, Mem8, Mem16)

Algoritma: (AL). (S8)  $\rightarrow$  AX  
 (AX). (S16)  $\rightarrow$  (DX), (AX)

Etkilediği Bayraklar: OF, CF  $\rightarrow$  etkilendir. Diğerleri undefined.

isi: MUL ile aynı isi yapar.

$\hookrightarrow$  IMUL DI ; AX ile DI'yi carp. Sonucu DX-AX'e yaz.

IMUL DH ; AL ile DH'yi carp. Sonucu AX'e yaz.

## -AAM instruction: (Adjust AX for multiplication)

Kullanım: AAM

Algoritma:  $AL / 10 \rightarrow AH$  (bölm)  $AL \% 10 \rightarrow AL$  (kalan)

Etkilediği Bayraklar: SF, ZF, PF  $\rightarrow$  etkilenir. CF, OF, AF  $\rightarrow$  undefined.

İşleme: AL'deki değeri decimal yap -  $(AL / 10)$  değerini

(bölm) AH'ye,  $(AL \% 10)$  değerini (kalan) AL'ye yaz.

Örnek:

MOV AL, 005h	$; AH:00$	AL: 05
MOV CL, 005h	$; CH:00$	CL: 05
MUL CL	$; AH:00$	AL: 19
AAM	$; AH:02$	AL: 05

$$\begin{array}{r}
 05 \\
 \times 05 \\
 \hline
 +00 \\
 \hline
 019h = 25
 \end{array}
 \quad \text{MUL} \\
 \boxed{25 / 10 = 2} \\
 \boxed{25 \% 10 = 5} \quad \text{AAM}$$

## -DIV instruction: (Division (unsigned))

Kullanım: DIV S (Reg8, Reg16, Mem8, Mem16)

Algoritma:  $(AX) \% (S8) \rightarrow (AH)$  (kalan)  
 $(AX) / (S8) \rightarrow (AL)$  (bölm)

$(DX, AX) / (S16) \rightarrow (AX)$  (bölm)  
 $(DX, AX) \% (S16) \rightarrow (DX)$  (kalan)

\* Sifira bölüm durumunda hata verir. (type 0 interrupt)

Etkilediği Bayraklar: Herpsi  $\rightarrow$  undefined.

İşleme: S 8 bit ise: AX'i S'ye bööl bölümü AX'ye kalanı AL'ye yaz.

S 16 bit ise: AX'i S'ye bööl bölümü DX'ye kalanı DX'ye yaz.

Örnek:

MOV AX, 203d ;  $AX = 00CBh$

MOV BL, 4d ;  $BL = 4h$

DIV BL ;  $AL = 50(32h)$ , AH = 3      Emanatörde görünen  
 $AH = 32$        $AL = 3$

- IDIV instruction: (Integer divide (signed))

Kullanım: IDIV S (Reg8, Reg16, Mem8, Mem16)

Algoritma:  $AX \div 58 \rightarrow AH$  (kalan)  
 $A \div 58 \rightarrow AL$  (bölm)

$DX, AX \div 516 \rightarrow AX$  (bölm)  
 $DX, AX \div 516 \rightarrow DX$  (kalan)

Etkilediği Bayraklar: Hepsi  $\rightarrow$  undefined.

isi: DN ile aynı isi yapar.

- AAD instruction: (Adjust AX for division)

Kullanım: AAD

Algoritma:  $AL = (AH * 10) + AL$   
 $AH = 0$

Etkilediği Bayraklar: SF, ZF, PF  $\rightarrow$  etkilendir. OF, AF, CF  $\rightarrow$  undefined.

isi: AAM komutunun yaptığı işin tam tersini yapar.

Örnek:

MOV AX, 0D10SH ; AH:01 AL:05  
AAD ; Formülü uygula  $AL = (AH * 10) + AL$ , AH=0  
; AH:00 AL:0F  $(01 * 10) + 05 \rightarrow AL = 15(Fh)$   
AH=0.

\* Bu önemli dedi hoca.

\* Diğer adjust'lar after, bu before.

\* ASCII adjust before division

Örnek:

MOV BL, 009 ; BH:00 BL:09  
MOV AX, 00702h ; AH:07 AL:02  
AAD ; AH:00 AL:48 (72 decimal)  
DIV BL ; AH:00 AL:08

- CBW instruction: (Convert byte to word)

Kullanım: CBW

Algoritma: if high bit of AL=1 then:

- AH = 255 (0FFh)

else

- AH = 0

Etkilediği Bayraklar: Hıghbitisi.

İşti: AL'nin en yüksek değerli bitine bak. 1 ise AH'ye FF'ye yaz.  
değilse AH'ye 00'yaz.

Örnek:

MOV AX, 0 ; AL:00 AL:00  
MOV AL, -5 ; AL:00 AL:F3 → 1111 1011  
CBW ; AH:FF AL:FB → AX:0FFF3h (-5)

\* AL'deki sayıyı AX'ye taşıır. (8 bitliyi 16 bitlik yapar)

\* 8 bitte -5 → 0000 0101 (5) { high bit of AL.  
1111 1010  
+  
1111 1011 → (-5) } 0000 0000 0000 0101  
F B F F 1111 1111 1111 1010  
1111 1011 -5

- CWD instruction: (convert word to double word)

Kullanım: CWD

Algoritma: if high bit of AX=1 then:

- DX=65535 (0FFFFh)

else

- DX = 0

Etkilediği Bayraklar: Hıghbitisi

İşti: AX'in en yüksek değerli biti 1 ise DX'yi FFFF yap.  
değilse DX=0 yap.

\* CBW'ye benzerdir. Burada 16 bitlik sayı 32 bitlige (DX, AX) konur.

## - AND instruction : (Logical AND)

Kullanım: AND D, S

Algoritma: (S) . (D) → (D)

Etkilediği Bayraklar: OF, SF, ZF, PF, CF → etkilenir. AF → undefined.

İşleme: S ve D'nin bitleri arasında mantıksal AND işlemi yapar.

\* Burada yine memory to memory yok!

Örnek:

MOV AL, 0B3h ; AL = <u>1011</u> MOV BL, 02Fh ; BL = <u>0010</u> AND AL, BL ; AL = <u>0010</u> , → Son durumda	<u>0011</u> <u>1111</u> <u>0010</u> , <u>0011</u> , <u>AL = 23</u>
2      3	

\* XXXX XXXX → unknown number  
0000 1111 → Mask  
0000 XXXX → Result (Boyle yaparak 8 bitlik register'in low nibble'ini alıp high nibble'ini değiştiririz.)

\* Bu işlemi kullanarak ASCII kodlarını BCD kodlarına çevirebiliriz.

MOV BX, 03135h ; BH:31      BL:35 AND BX, 0F0Fh ; BH:01      BL:05	ASCII 30 - 39 BCD 0 - 9
---	----------------------------

## - OR instruction : (Logical inclusive -OR)

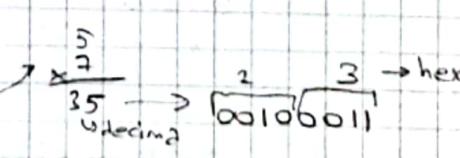
Kullanım: OR D, S

Algoritma: (S)+(D) → (D)

Etkilediği Bayraklar: OF, SF, ZF, PF, CF → etkilenir. AF → undefined.

İşleme: S ve D'nin bitlerini or işlemine tabi tut. Sonucu D'ye yaz.

Örnek:

MOV AL, 005h ; AL = 05 MOV BL, 007h ; BL = 07 MUL BL ; AL = 23 AAM ; AH = 02      AL = 03 OR AX, 03030h ; AH = 32      AL = 33	
--	--

- XOR instruction: (Logical Exclusive - OR)

Kullanım: XOR D, S

Algoritma: (S) ⊕ (D) → D

Etkilediği Bayraklar: OF, SF, ZF, PF, CF → etkilenir. AF → undefined.

isi: S ve D'nin bitlerini xor'la. Sonucu D'ye yazar.

★ S ⊕ D		Sonuç	{	★ XXXX ×××× → unknown 8-bit binary num	
Girdi (	0 0	0 ) Aynı		0 0 0 0	1 1 1 1 → pattern for inverting low 4 bits.
Girdi (	1 0	1 ) Farklı		X X X X	XXXX → result inverted bits.
	1 1	0			

- NOT instruction: (Logical NOT)

Kullanım: NOT D (Reg, Mem)

Algoritma: (D)  $\xrightarrow{\text{reverse}}$  (D)

Etkilediği Bayraklar: Hızbırımı

isi: Mantıksal not işlemi yapar. (0' → 1, 1 → 0)

Örnek:

Mov AL, 06Ch ; AL =  $\begin{array}{l} 0110 \\ \downarrow \quad \uparrow \\ 01 \quad 10 \end{array}$  C  
NOT AL ; AL =  $\begin{array}{l} 1001 \\ \downarrow \quad \uparrow \\ 10 \quad 01 \end{array}$  → AL = 93

Örnek: (Final için +5 bonus sorusu). AX en son durumda nedir?

MOV AX, 033CCh ; AH = 33 AL = CC  
AND AX, 0FF00h ; AH = 33 AL = 00  
XOR BX, BX ; BX = 0000 (Kadisiyle xor larsak 0⊕0→0 olacağından BX = 0000)  
NOT BX ; BX = FF FF  
OR AY, BX ; AY = FF FF

Cevap: AX son durumda FFFF'dir.

- SAL / SHL instructions : (Shift arithmetic left / shift, logical left)

Kullanım : SAL / SHL D, Count

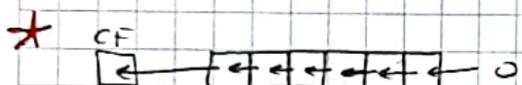
Algoritma : Tüm bitleri count kadar sola kaydırır. Sağdaki yerlere 0 doldur.

Etkilediği Bayrakları : C Z SP → ettilenir. AF → undefined. OF → undefined if count ≠ 1

isi : Bitlerin count kadar sola kaydırılır.

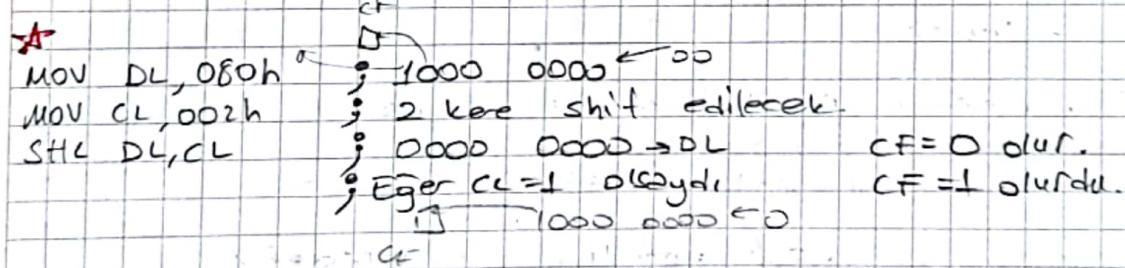
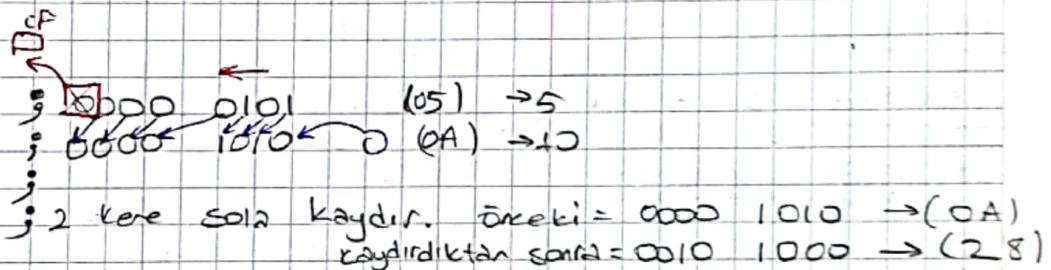
D	Count
Reg	1
Reg	CL
Nem	+
Nem	CL

- \* Kaydırma sayısı (count), 1'den büyükse CL registerine kaydedilmeli.
- \* SAL → shift arithmetic left  
SHL → shift logical left } ikisi de aynı işi yapar.



Örnek :

MOV DL, 005H  
SHL DL, 1  
MOV CL, 002H  
SDL DL, CL



\* Burada dikkat edilmesi gereken konu eğer DL 2 kez sola kaydırılırsa en yüksek öncelikli bitin bir sağındaki bit CF'ye yazılır ve DL'nin sonunda 2 tane sıfır eklenir. En yüksek öncelikli bit atılır.

## - SHR instruction: (Shift logical right)

Kullanım: SHR D, Count

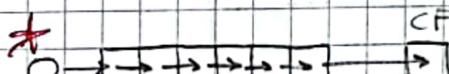
Algoritma: Tüm bitleri sağa doğru count kadar kaydırır. Boş kalan yerlere 0 doldur.

Etkilediği Bayraklar: C, P, S, Z, O → etkilenir. AF → undefined, OF → undefined if count ≠ 0

isi: Sayının bitlerini count kadar sağa kaydırır.

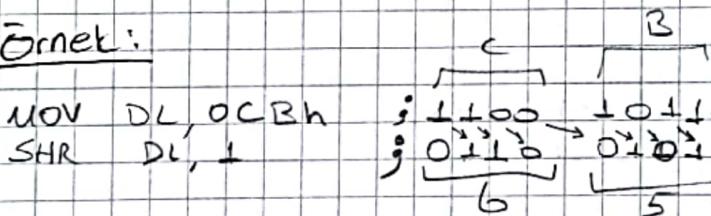
\* D count \* Count 1'den büyükse CL kullanımları zorunluudur.

Reg 1  
Reg CL  
Mem 1  
Mem CL



\* Buradaki kaydırma mantığı SALISH ile aynıdır.

Örnek:



$$\rightarrow 65, CF = 1$$

## - SAR instruction: (Shift arithmetic right)

Kullanım: SAR D, count

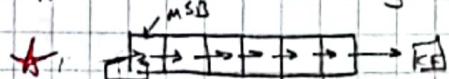
Algoritma: D'nin bitlerini count kadar sağa kaydırır. Boş kalan soldaiki yerlere en yüksek enverlikli bit ile aynı bitleri yerlestirir.

Etkilediği Bayraklar: C, Z, S, O, P → etkilenir. AF → undefined.

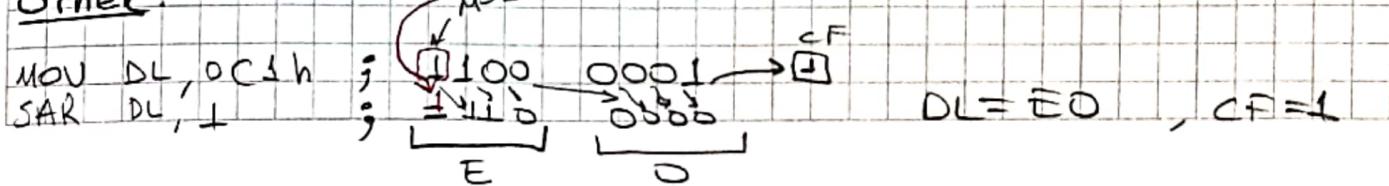
isi: D'yi count kadar sağa kaydırır.

\* D count \* Count 1'den büyükse CL kullanımları zorunluudur.

Reg 1  
Reg CL  
Mem 1  
Mem CL



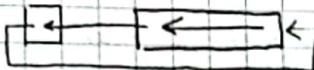
Örnek:



$$DL = EO, CF = 1$$

## Rotate Instructions

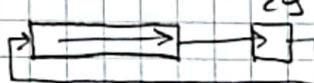
- RCL (Rotate Left Through Carry)



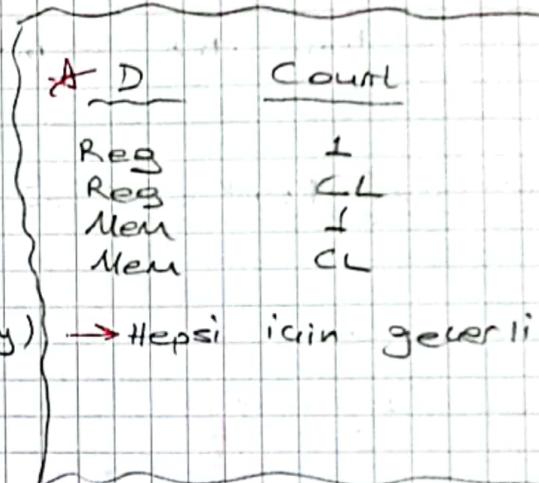
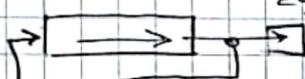
- ROL (Rotate Left)



- RCR (Rotate Right Through Carry)



- ROR (Rotate right)



### ROL instruction: (Rotate Left)

Kullanım: ROL D, Count

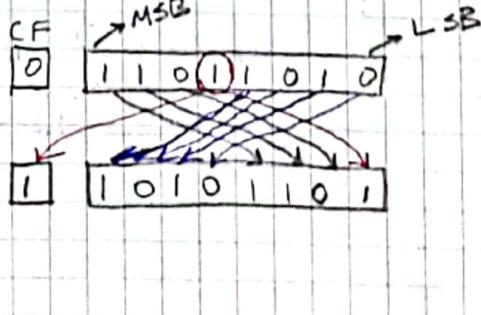
Algoritma: Tüm bitleri count kadar sola kaydır. Sağda boş kalan yerleri soldan çıkan bitlerle doldur. CF'ye en yüksek değerli basamağı yaz.

Etkilediği Bayraklar: CF, etkilenir. OF → count ≠ 1 ise undefined. aksi halde etkilenir.  
isi: Algoritmada verilen kaydırma isini yapar.

\* Bit kaybı olmaz. (CY ile sayı ayırdır. Bir sayıya bakıyoruz.)

Örnek:

MOV DL, 0DAh  
MOV CL, 004h  
ROL DL, CL



4 kez yapıldığı için MSB işaretli yer oluyor ve CF'ye ve LSB'ye kopyalandı. (soldan kaymaya başlıyor)

## - ROR instruction: (Rotate right)

Kullanım: ROR D, Count

Algoritma: Tüm bitler count kadar sağa kaydırılır. Solda boş kalan yerlere sağdan çıkan bitler konur. CF'ye en düşük değerlikli bit (LSB) yerleştirilir.

Etkilediği Bayraklar: (F etkilenir. OF  $\rightarrow$  count  $\neq 1$  ise undefined aksi halde etkilenir.)

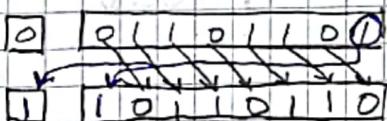
i̇i̇: Algoritmada verilen i̇i̇ yapar.

\* Bit kaybı olmaz.

Örnek:

CF

Mov DL, 06D  
Ror DL, 1



## - RCL instruction: (Rotate left through carry)

Kullanım: RCL D, Count

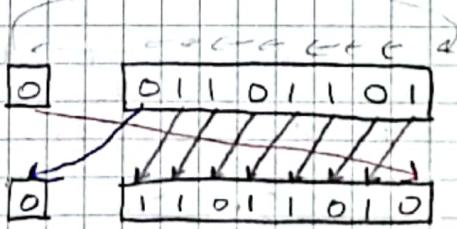
Algoritma: CF'de D ile birleşenek count kadar sola kaydırılır.

Etkilediği Bayraklar: CF  $\rightarrow$  etkilenir. OF  $\rightarrow$  count  $\neq 1$  ise undefined aksi halde etkilenir.  
i̇i̇: Algoritmada verilen i̇i̇ yapar.

\* Tam bir daire uzayımous gibi düşünülebilir.

Örnek:

Mov DL, 06Dh  
Rcl DL, 1



- RCR instruction: (Rotate right through carry)

Kullanım: RCR D, Count

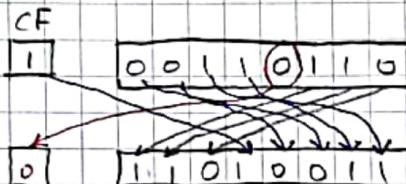
Algoritma: CF'yi de D'ye katarak count kadar sağa kaydırma yapılır.

Etkilediği Bayraklar: CF → etkilenir. OF → count + 1 ise undefined aksi halde etkilenir.

İşii: Algoritmada verilen işi yapar.

Örnek:

MOV DL, 036h  
MOV CL, 004h  
RCR DL, CL



Sağdan 4 say.

Örnek: MUL ve DIV kullanmadan  $7(Ax) - 5(Bx) - \frac{1}{8}(By) \rightarrow (Ax)$

İşlemi yap. (SHL → 2 ile çarpma, SHR → 2'ye bölmeye denktir.)

MOV CL, 003h ; CL'ye 3 tane 2'ye bölmeye hazırla

MOV DX, AX ; Ax değerini DX'e yükle.

SHL AX, CL ; Ax'i 3 defa sola kaydır. ( $8'le çarp$ ) ( $8(Ax)$ )

SUB AX, DX ; DX'in içinde Ax vardı.  $8(Ax) - (Ax) = 7(Ax)$

MOV CL, 002h ; Bir sayıyı 4 ile çarpmak için CL'yi hazırla.

MOV DX, BX ; BX değerini DX'e yükle.

SHL BX, CL ; BX'i 2 defa sola kaydır. ( $4'le çarp$ ) ( $4(Bx)$ )

ADD BX, DX ;  $4(Bx) + (Bx) = 5(Bx)$

MOV CL, 003h ; Bir sayıyı 8'e bölmek için CL'yi hazırla

SHR DX, CL ; DX'de zaten BX tutuluyordu.  $\frac{1}{8}(Bx)$

SUB AX, BX ;  $7(Ax) - 5(Bx) - \frac{1}{8}(By) \rightarrow (Ax)$

SUB AX, DX ;  $7(Ax) - 5(Bx) - \frac{1}{8}(By) \rightarrow Ax$

## Flag Control Instructions:

<u>Kullanım</u>	<u>Algoritma</u>	<u>Etkilediği Bayraklar</u>	<u>İşİ</u>
- <b>L AHF</b> (load AH from flags)	$(AH) \leftarrow (Flags)$	Hibirisı	Bayrakları AH'ye yükler. $SF \quad ZF \quad AF \quad PF \quad CF \rightarrow AH$
- <b>SAHF</b> (Store AH into flags)	$(Flags) \leftarrow (AH)$	SF, ZF, AF, PF, CF	AH'yi bayraklara yükler.
- <b>CLC</b> (Clear carry flag)	$(CF) \leftarrow 0$	CF	CF'yi temizler.
- <b>STC</b> (Set carry flag)	$(CF) \leftarrow 1$	CF	CF'yi set eder.
- <b>C MC</b> (Complement carry flag)	$(CF) \leftarrow (\overline{CF})$	CF	CF'nin komplementini alır. (tersini)
- <b>CLI</b> (Clear interrupt flag)	$(IF) \leftarrow 0$	IF	IF'yi temizler.
- <b>STI</b> (Set interrupt flag)	$(IF) \leftarrow 1$	IF	IF'yi set eder.
- <b>CLD</b> (Clear direction flag)	$(DF) \leftarrow 0$	DF	DF'yi temizler.
- <b>STD</b> (Set direction flag)	$(DF) \leftarrow 1$	DF	DF'yi set eder.

## CMP instruction: (Compare)

Kullanım CMP D,S

→ Memory-memory yok!

Algoritma:  $(D) - (S)$

Etkilediği Bayraklar: Hepsí (CF, ZF, SF, AF, PF, DF)

İşİ: D-S yapar ve böylece bayrak değerlerini etkiler.

\* (D)-(S)'nin sonucu hiçbir yerde tutulmaz.

\* D ve S ADD komutundaki gibidir.

\* Slaytta bundan sonra flagler geliyor. Ama defterin

başında işlemistik

Örnek:

Mov AL,09H

; 1001 1001 → -103d

Mov BL,01BH

; 0001 1011 → +27d

CMP AL,BL

: -103d - (+27)d

$$\begin{array}{r}
 \text{Emulator'da} \\
 \begin{array}{r}
 \text{C:1} \quad 0 \\
 \text{Z:0} \quad 0 \\
 \text{S:0} \quad 0 \\
 \text{O:0} \quad 0 \\
 \text{P:1} \quad 1 \\
 \text{A:0} \quad 1
 \end{array}
 \end{array}
 \begin{array}{r}
 \xrightarrow{\quad} 0001 \quad 1011 \quad \rightarrow +7d \\
 + 1110 \quad 0100 \\
 \hline
 1110 \quad 0101
 \end{array}
 \begin{array}{l}
 \rightarrow 2^7 d \\
 \text{2's complement}
 \end{array}$$
  

$$\begin{array}{r}
 \text{card+} \\
 \begin{array}{r}
 \text{C:1} \quad 0 \\
 \text{Z:0} \quad 0 \\
 \text{S:0} \quad 0 \\
 \text{O:0} \quad 0 \\
 \text{P:1} \quad 1 \\
 \text{A:0} \quad 1
 \end{array}
 \end{array}
 \begin{array}{r}
 \xrightarrow{\quad} 1001 \quad 1001 \quad \rightarrow -103d \quad (98h) \\
 + 1110 \quad 0101 \\
 \hline
 0111 \quad 1110
 \end{array}
 \begin{array}{l}
 \rightarrow -27d. \quad (E5h) \\
 \rightarrow \text{Sonuç.}
 \end{array}$$

## JMP instruction: (Unconditional Jump)

Kullanım: JMP operand (Labels, memptr16, memptr32, regptr16)  
(short near far)

Algoritma: Belirtilen etikete koşulsuz olarak geç.

### Etkilediği Bayraklar: Hibirisı

iş: Etikete koşul olmadan gider ve program o etiketten devam eder.

\* 3 tane jump çeşidi vardır:

→ Short Jump: (+127, -128 byte öteye) CS ve IP degismez.)

→ Near JUMP: (+32 kB öteye) IP değişir, CS değişmez. } intrasegment Jump

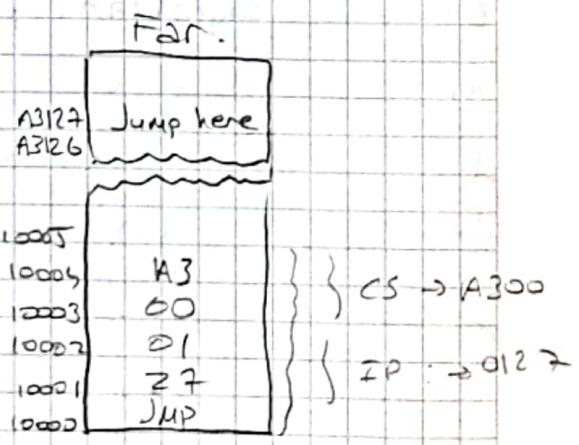
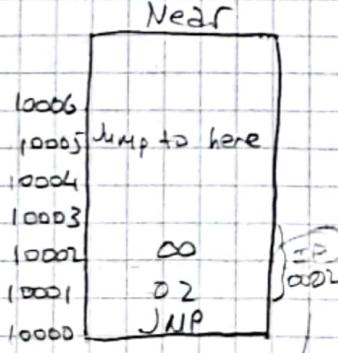
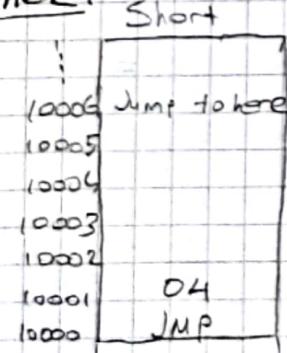
→ Far JUMP: Başka bir bellek alanına geçer. IP ve CS değişir } intersegment Jump

\* Pozitif yer değiştirmeye genelde su anlamına gelir:

Sırasına öne doğrudur. Negatif yer değiştirmeye genelde su anlamına gelir; sırasına geriye doğrudur. (Genelde denmesin).

Sebebi örneğin; 00 olsun ve birer birer 0'dan başlayıp birer bire artıralım (ileri doğru). 11'e geldiğinde geri 00'a döncepi için (geri doğru) her zaman yerine genelde ifadesi kullanılır.)

### Örnek:



$$CS = 1000$$

$$IP = 0002$$

$$\text{New IP} = IP + 4 \\ = 0006$$

$$CS = 1000$$

$$IP = 0003 + 2$$

$$\text{New IP} = 0005$$

$$CS = 1000$$

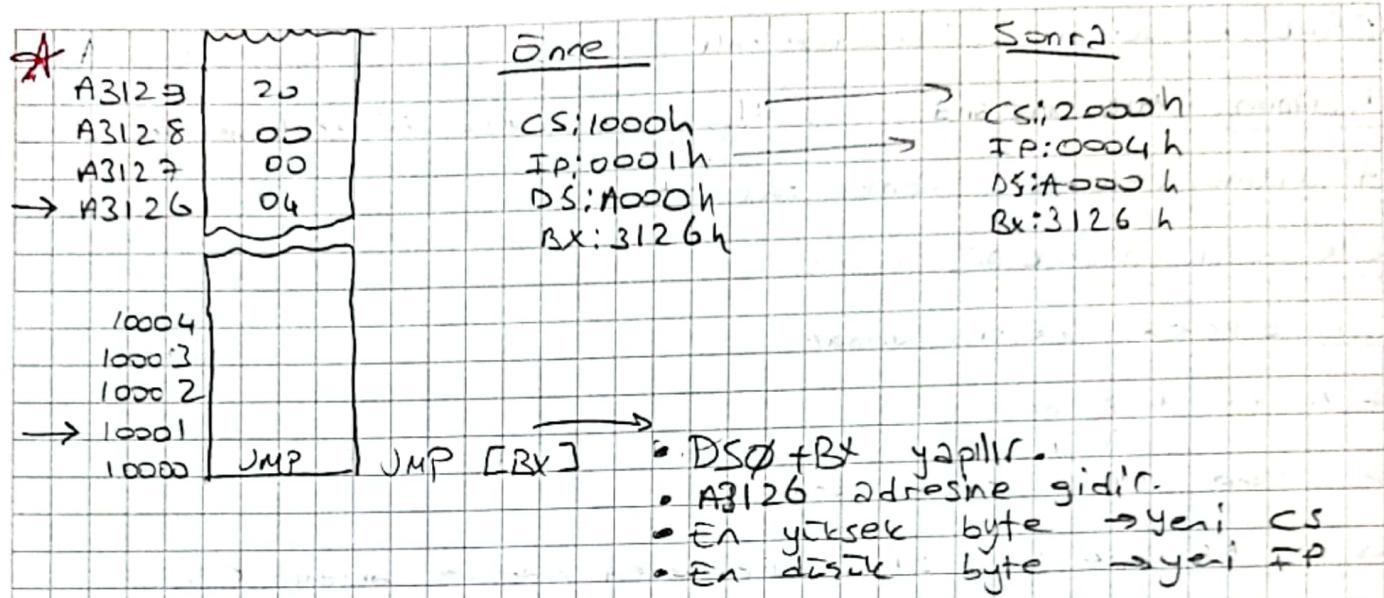
$$IP = 0005$$

$$\text{New CS} = A300 \\ \text{New IP} = 0127$$

$$A300 = 0 \\ 0127$$

$$+ 0127 \\ + A3127$$

\* JMP BX → Near jump, JMP BL → short jump, Far jump DS0 + BX → far jump here.



### Conditional Jump Instructions:

Kullanılmış cc operand (labels, memptr 16, memptr 32, regptr 16)

Etkilediği Bayraklar & Hıgbiri

İşleri: Koşullu Skırama yapmak

İsim	Skırama	Ertel	Algoritma	İzaretli / izanetsiz
JA	Z=0 and C=0		Jump if above	Unsigned
JAE	C=0		Jump if above or equal	Unsigned
JB	C=1		Jump if below	Unsigned
JBE	Z=1 or C=1		Jump if below or equal	Unsigned
JC	C=1		Jump if carry set	-
JE or JZ	Z=1		Jump if eq. or jmp if zero JE → itisi, JZ → -	
JG	Z=0 and S=OF		Jump if greater than	Signed
JGE	S=OF		Jump if greater than or eq	Signed
JL	S#OF		Jump if less than	Signed
JLE	S#OF or ZF=1		Jump if less than or eq	Signed
JNC	C=0		Jump if no carry	-
JNE or JNZ	Z=0		Jump if not eq or not zero JNE → itisi, JNZ → -	
JNO	OF=0		Jump if no overflow	-
JNS	S=0		Jump if no sign	-
JNP or JPO	P=0		Jump if no parity or odd parity	-
JO	OF=1		Jump if overflow set	-
JP or JPE	P=1		Jump if parity set or even parity	-
JS	S=1		Jump if sign is set	-
JCXZ	CX=0		Jump if CX=0	-

## - CALL instruction: (Subroutine call)

Kullanım: CALL operand

( Proc, memptr16, Regptr16, memptr32 )

Algoritma: Belirtilen fonksiyona git.  
near far

Etkilediği Bayraklar: Hızbırısı

İş: Fonksiyon çağrısı yapar.

\* JMP komutuna benzerdir.

\* 2 tane CALL çeşidi vardır:

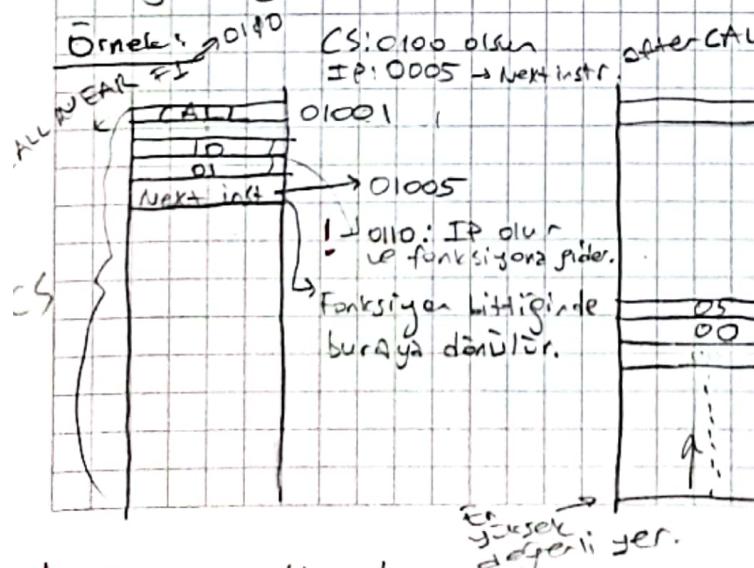
→ Near Procedure : ( $\pm 32\text{ kB}$ ) IP değeri stackte saklanır. CS değişmez  
↔ (SS)

→ Far Procedure : (CS'den farklı bir yer) IP ve CS stackte saklanır.

\* SS'de saklanmaların nedeni: Fonksiyon bittiğinden sonra nereye dönmeyi bilmesi gerekiyor.

\* Near: CALL komutu okunduğunda SP 2 azalır. IP'nin sonraki komutu göstereceği yer stack'e aktarılır. Fonksiyon bittiğten sonra, program kaldığı yerden devam eder.

\* Far: CALL komutu okunduğunda SP 2 azalır. IP değeri (nextinst.) stack'e kopyalanır. Tekrar SP 2 azaltılır ve CS stack'e kopyalanır.



! SP : SP basken SS'in en başını yanı son elemanı gösterir.

! Normalde yukarıdan aşağıya sıralanır. Ama stack'te tam tersi.

! SP 2 azalır ve yukarıda gitken önce dökük öncelikli kismi yanı 05'i döküp sonra yükseli öncelikli kismı 00'yi stack'e getirir.

\* Far proc'da benzer mantıkta olur.

## - RET instruction : (Return)

Kullanım: RET

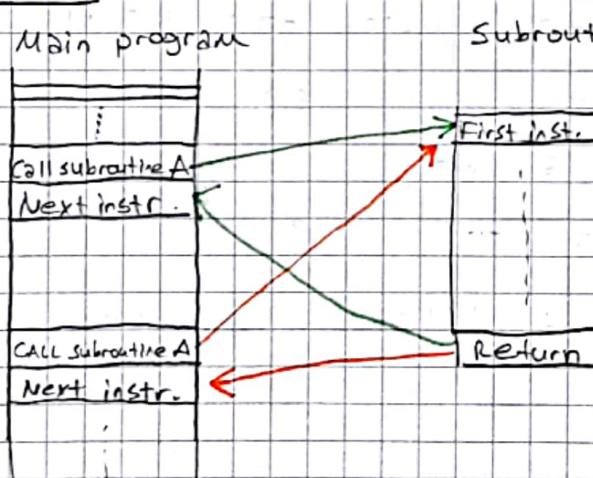
Algoritma: Near proc  $\rightarrow$  Next instr. değerini IP'ye aktar.  
Far proc  $\rightarrow$  CS ve next inst. değerini CS ve IP'ye aktar.  
Etkilediği Bayraklar: Hicibinisi.

İzi: Fonksiyon bittiğinden sonra programın kaldığı yerden devam etmesi için IP ve CS (eger far ise) türü yükler.  
(Stackteki)

\*Near: SP 2 artır ve stackteki next instr. IP'ye yazılır.

\*Far: SP 2 artır and programın CS değeri stackten çıkarılır ve CS'ye yazılır. Tekrar SP 2 artır ve next instruction IP'ye yazılır.

Örnek:



\* Fonksiyonlar bir değer döndürür.

Procedure'ler bir değer döndürmez.

\* Buradakiler procedure'dır.

## Stack Instructions

- PUSH instruction: (Push word onto stack)

Kullanım: PUSH S

(Register, Seg-Reg, Memory)  
(16) ((S hanesi))

Algoritma:  $((SP) \leftarrow (S))$

$(SP) = (SP) - 2 \rightarrow 2$  olmasının nedeni word (16 bit) ile çalışmasıdır.  
Etkilediği Bayraklar: Hiçbirisi

isi: S'yi stack'e eklemek. ( $S \rightarrow$  16 bitlik olmalıdır)

\* Stack Segment, geçici dataları ve geri dönüş adresini (call) tutar. LIFO (last in first out) (son giren ilk çıkar) mantığında göre çalışır. PUSH ile stack'e sokulan datalar, POP ile çıkarılır. SS:SP, stack'in en üstüne (yani son giren eleməni) aynı zamanda bu elemən ilk çıkarık eleməndir gösterir.

Stack, yüksek adresden düşük adres'e doğru dolar. (Tersten)  
Higher byte önce lower byte sonra eklenir.

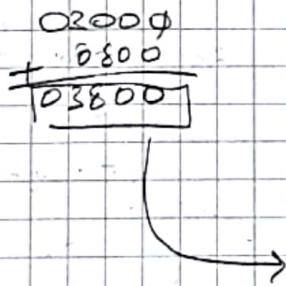
Örnek:

AX: 06AB3H

SS: 0300H

SP: 0800H

PUSH AX

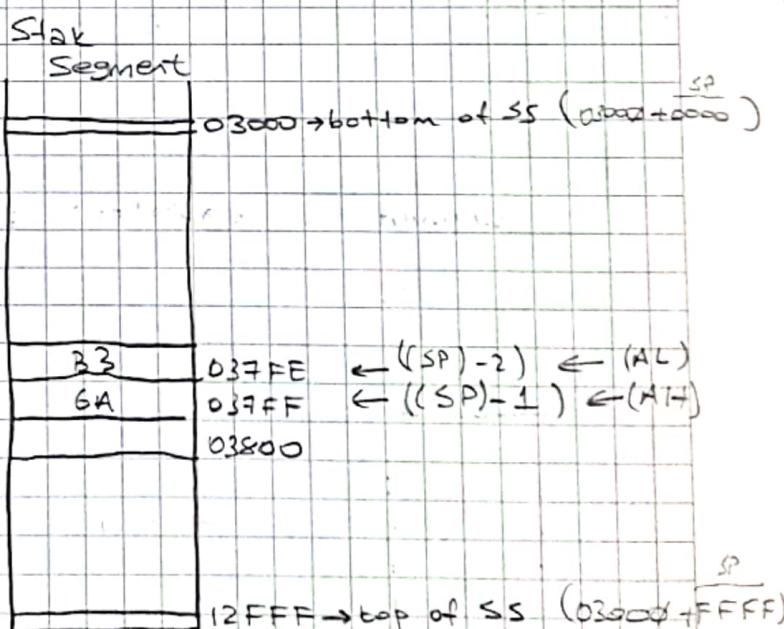


Son durumda

AX: 06AB3H

SS: 0300H

SP: 07FEH



• Başka bir değer ekleneceği zaman 037FD ve 037FC'ye eklenir. Bu böyle devam eder.

- POP instruction: (Pop word off stack)

Kullanım: POP D (Reg, Seg\_Reg, Memory)  
(CS, DS, SS, ES, FS, GS)

Algoritma:  $(D) \leftarrow ((SP))$

$$(SP) = (SP) + 2$$

Etkilediği Bayraklar: Hiabirisi.

İşİ: SS:SP'ının gösterdiği yerdeki 2 byte'lik (word) veriyi D'ye aktarır.

Örnek: Geçen sayfadaki örneğin POP versiyonu

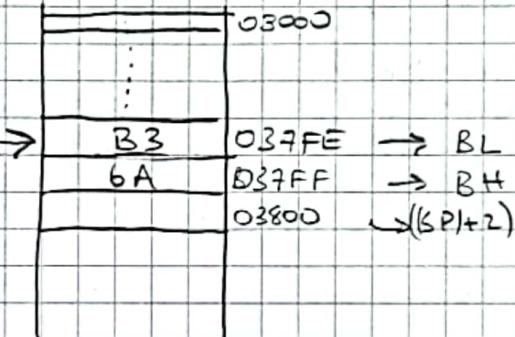
BX:00000H

SS:0300H

SP:07FEH

POP BX

$$\begin{array}{r} 0300\phi \\ + 07FE \\ \hline 1037FE \end{array}$$



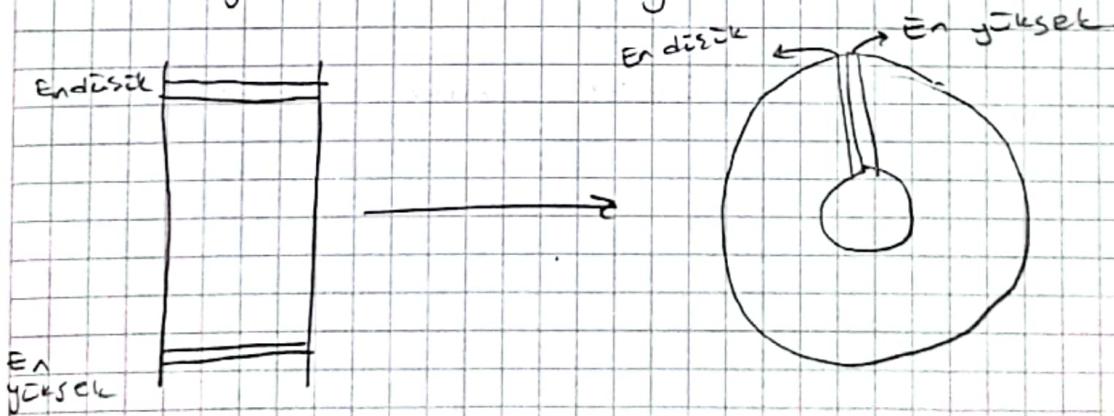
İşlem bittiğinde:

BX:6AB3H

SS:0300H

SP:0800H

!!! **A** Tom segmentler döngüseldir yani en düşük öncelikli yeri ve en yüksek öncelikli yeri komesudur.



## - PUSHF instruction: (Push flags onto stack)

Kullanım: PUSHF

Algoritma:  $((SP)) \leftarrow (Flags)$

$$(SP) = (SP) - 2$$

Etkilediği Bayraklar: Hibiris!

isi: Bayrakları stack'e yerlestirir.

## - POPF instruction: (Pop flags from stack)

Kullanım: POPF

Algoritma:  $(Flags) \leftarrow ((SP))$

$$(SP) = (SP) + 2$$

Etkilediği Bayraklar: OF, DF, IF, TF, SF, ZF, AF, PF, CF → Hepsı.

isi: Stack'teki 2 byte'lık değerin bitlerini bayraklara yerlestirir.

## Loop Instructions

### Kullanım

- LOOP short-label  
(Loop)

- LOOPZ/LOOPNZ short-label  
(Loop while equal/zero)

- LOOPNE/LOOPNZE short-label  
(Loop while not equal/not zero)

### Algoritma

$CX \neq 0$  ise etikete git  
 $CX = CX - 1$

$CX \neq 0$  ve  $ZF \neq 0$  etikete git  
 $CX = CX - 1$

$CX \neq 0$  ve  $ZF = 0$  etikete git  
 $CX = CX - 1$

### Etkilediği Bayraklar

Hibiris!

Hibiris!

Hibiris!

\* Belli sayıda tekrar eden komutlar ikerir. Loop'un tekrar sayısını CX registeri belirler. CX=0 olduğunda döngiden çıkış bir sonraki komuttan devam eder. Her LOOP komutu okunduğunda CX, 1 azalır. Short-label (-128 byte, +127 byte) değerini alır.

### Örnek:

Next:

MOV CX, count

;

;

;

LOOP Next

Bu işlemi en  
Sonraki de yaparız.

\* ikisi de aynı.

MOV CX, count

;

;

;

DEC CX

JNZ Next

;

\* Loop komutunu  
yapmak için  
diğer instr. kullanı  
yapabiliriz.

Örnek: Aşağıdaki kodun ne iş yaptığıni bul.

```
XOR BL, BL  
MOV AX, 01234H  
MOV CX, 00010H
```

L1:

```
SHL AX, 1  
JNC Bypass  
INC BL  
Bypass:  
LOOP L1  
HLT
```

BL: 00, AX = 0001 0010 0011 0100  
CX = 10H → 16 decimal

CF 0001 0010 0011 0100

\* Her seferinde + kaydırıp CF'nin üzerine bakıyor (JNC → Jump not CF). Eğer CF'ye + gelirse BL'yi 1 artırıyor. Kodun yaptığı iş:

Ax' registerinin içindeki bitlere bakarak kaç tane + olduğunu söylüyor. Kod tamamlandığında BL, 5 olur.

Örnek: Aşağıdaki kodun ne iş yaptığıni bul.

```
MOV DL, 005H  
XOR AX, AX  
MOV DS, AX  
MOV SI, 0A000H  
MOV CX, 0FH  
AGAIN:  
INC SI  
CMP [SI], DL  
LOOPNE AGAIN  
HLT
```

DL = 05H, AX: 0000H, DS: 0000H, SI: A000H, CX: 0FH

\* 0A001 adresinden itibaren 15 defa adres artırıp o adresler aralığında 05H değeri var mı diye bakıyor.

\* Bir virüsün imzasını biliyorsak o imzayı (DL, yerine) ve bu kodu yazarsak (yani imzayı memari arama kodu) bir antivirus programı yazmış oluruz. (Hədə söyledi)

## String Instructions

- MOVSB instruction (Move string (byte or word))

Kullanım: MOVSB veya MOVSW (B → Byte (8 bit), W → Word (16 bit))

Algoritma: ((ES)O + (DI)) ← ((DS)O + (SI))

(SI) = (SI) ± 1 veya 2 } ± olmasının nedeni DF = 0 ise +  
(DI) = (DF) ± 1 veya 2 } DF = 1 ise -  
"B" olursa +, "W" olursa 2

Etkilediği Bayraklar: Hiçbirisi

isi: Memoryden memorge aktarım yapmak. (DS:SI'den ES:DI'ya)

unique for 8086 µP

! DF'ye dikkat et O ise normal  
1 ise tersten.

- **CMPSB** instructions: (Compare string (byte or word))

Kullanım: CMPSB veya CMPSW

Algoritma:  $((DS)O + (SI)) - ((ES)O + (DI))$

Bu işlem sonucu bayraklar etkilenir.

$$\begin{aligned} (SI) &= (SI) \pm 1 \text{ veya } 2 && \left. \begin{array}{l} \text{B ise } 1, \text{ W ise } 2 \\ DF = 0 \text{ ise } +, DF = 1 \text{ ise } - \end{array} \right. \\ (DF) &= (DI) \pm 1 \text{ veya } 2 && \end{aligned}$$

Etkilediği Bayraklar: (CZSOPA)  $\rightarrow$  Hepsı.

isi: Memory'ler arası karşılaştırma yapmak. (CMP ile aynı mantık)

\* Diğer komutlar kullanılarak da yapılabilir bu komut.

- **SCASB** instruction: (Scan string (byte or word))

Kullanım: SCASB veya SCASW

Algoritma:  $(AL \text{ or } AX) - ((ES)O + (DI))$

Bu işlem sonucu bayraklar etkilenir.

$$(DI) = (DI) \pm 1 \text{ veya } 2 \rightarrow \left. \begin{array}{l} \text{B ise } 1, \text{ W ise } 2 \\ DF = 0 \text{ ise } +, DF = 1 \text{ ise } - \end{array} \right.$$

Etkilediği Bayraklar: (CZSOPA)  $\rightarrow$  Hepsı

isi: AL veya AX ile ES:DI'yi karşılaştırır.

- **LODSB** instruction: (Load string (byte or word))

Kullanım: LODSB veya LODSW

Algoritma:  $(AL \text{ veya } AX) \leftarrow (DS)O + (SI)$

$$(SI) = (SI) \pm 1 \text{ veya } 2$$

Etkilediği Bayraklar: Hiçbirisi.

isi: Memorydeki (DS:SI) bitileri (B veya W) AL veya AX'le yüklemek.

- **STOSB** instruction: (Store string (byte or word))

Kullanım: STOSB veya STOSW

Algoritma:  $((ES)O + (DI)) \leftarrow (AL \text{ veya } AX)$

$$(DI) = (DI) \pm 1 \text{ veya } 2$$

Etkilediği Bayraklar: Hiçbirisi.

isi: AL veya AX'teki bilgiyi (ES:DI)'da depolamak.

- REP instruction: (Repeat)

Kullanımı: REP instr. → (MOVSB, LODSB, STOSB)

Algoritma: If CX ≠ 0 repeat instruction.

$$CX = CX - 1$$

Etkilediği Bayraklar: ZF etkilenir. Diğerlerini etkilenmez.

işii: MOVSB, LODSB, STOSB komutlarını CX içere tekrar eder ve her seferinde CX'yi bir azaltır.

- REPE/REPZ instructions: (Repeat while equal/zero)

Kullanımı: REPE/REPZ instr. → (CMPSB, SCASB)

Algoritma: If CX ≠ 0 and ZF = 1 repeat instruction.

$$CX = CX - 1$$

Etkilediği Bayraklar: ZF etkilenir. Diğerlerini etkilenmez.

işii: CMPSB, SCASB komutlarını CX ≠ 0 ve ZF = 1 olduğunda yap.

- REPNE/REPNZ instructions: (Repeat while not equal/not zero)

Kullanımı: REPNE/REPNZ instr. → (CMPSB, SCASB)

Algoritma: If CX ≠ 0 and ZF = 0 repeat instruction.

$$CX = CX - 1$$

Etkilediği Bayraklar: ZF etkilenir. Diğerlerini etkilenmez.

işii: CMPSB, SCASB komutlarını CX ≠ 0 ve ZF = 0 olduğunda yap.

Örnek: 1b tane veriyi (byte türünde) taşıyın. (00200'den 00300'e)

MOV AX, 07000H

MOV DS, AX

MOV SI, 00200H

MOV DI, 00300H

MOV CX, 010H ; 1b dönsüz

BACK: MOV AL, [SI]

MOV [DI], AL

INC SI

INC DI

LOOP BACK

HLT

MOV AX, 07000H

MOV DS, AX

MOV ES, AX

MOV SI, 00200H

MOV DI, 00300H

MOV CX, 010H ; decimal 16

CLD ; DF = 0 (Clear)

REP MOVS B

HLT

\* İki kodda aynı işi yapar. Soldakı ile yapmak daha basit.

### Örnek:

```

MOV AX, 0ABCDH
MOV CL, 011H ; 17 decimal
BACK:
    RCL AX, 1
    CMC
LOOP BACK
HLT

```

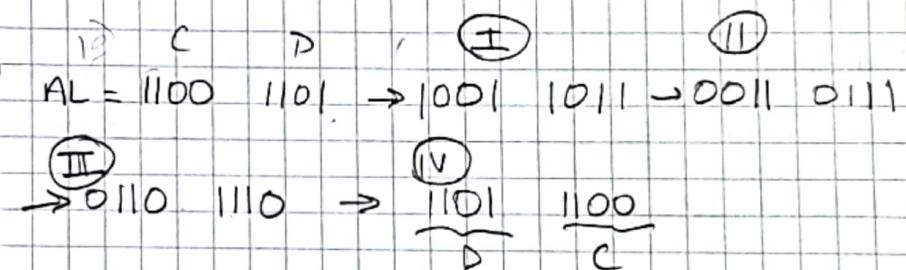
\* Hoca bunu verip bununla aynı işi yapan kod nedir diyebilir.

### Örnek:

```

MOV AX, 0ABCDH
MOV CL, 004
ROL AL, CL
ROL AH, CL
XCHG AH, AL
HLT

```



XCHG yapınca AH = DC, AL = BA olur.

Son durumda AX:DCBA olur. Basamaklarının yerini ters olur.

SON