



Microprocessor

Assembly Instructions

Dr. Cahit Karakuş
Esenyurt Üniversitesi

CPU - Mimarisi

- X86 (1978) dan Pentium ve multicore
- Bilgisayar: Donanım + Yazılım.
- Veri analitiđi 2010'lı yılların başından itibaren yapay zekaya dönüştü.

Mikroişlemci - ilki 4004
- 1978 - x86

Transistor - Yarı iletken - Quantum (Atom altı parçacık - elektron akışı)
Elektron akışını kontrol eder, 1947, boyut: Atomik. (4 nm)
(Lojik kapı, Bellek) $10^{-9}m$

Bilgi çağı - Transistor - CPU

Donanım Yazılım
Bilgisayar müh. - 90'lı yıllar.

- Robotlar (Robot kolları)
- Akıllı telefonlar
- Otomasyon sistemleri
- Veri analizi (Veri toplama, Veri hayranı, Veri tabanı yönetimi)

Yazılım: Matlab, Python, C++, Java
Bilimsel Matematik (AI) Donanım (Web) Scripti

Esenyurt Üniversitesi
Cahit hoca emelli - başkaldırı
Suriye, Arap karıştı, dünya karıştı,
İstanbul - Esenyurt.

Öğrenci: 0% 20 - İTÜ, Boğaziçi, ODTÜ
0% 50 - Orta → Şaban gösteren (!!!)

0% 70 - ? (İstanbul → Avrupa, is. → parası yok salıverdi)
? Aritmetik.

Yeni Meydan - Bilgi Müh.

metre, m
1 mm - $10^{-3}m$
1 μm - $10^{-6}m$
1 nm - $10^{-9}m$
1 pm - $10^{-12}m$

Sanatchalar
Bilim.

Robot akıllı (Sensör, aktuatörler)
Akıllı makineler. Elektrik enerjisini her türlü mekanik enerjiye dönüştürür (motor, piston, makaralar)
(İnternet arabalar) %60

İkili sayı sistemi (Binary)

- $0/1$: Bit.

- Her bir hata üzerinde $0/1$ vardır.
 n " bellek göründe 2^n

- 4 bit \rightarrow Hex.
 8 bit \rightarrow byte.
 16 bit \rightarrow word.

$$(1876)_d = (?)_b = (?)_h$$

2^{10}

$$\begin{array}{r} 1024 \\ \hline 852 \end{array}$$

2^9

$$\begin{array}{r} 512 \\ \hline 340 \end{array}$$

2^8

$$\begin{array}{r} 256 \\ \hline 84 \end{array}$$

2^6

$$\begin{array}{r} 64 \\ \hline 20 \end{array}$$

2^4

$$\begin{array}{r} 16 \\ \hline 4 \\ \hline 4 \\ \hline 0 \end{array}$$

2^2

$$\begin{array}{cccccccc} 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ (1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0)_b \end{array}$$

$$(7 \quad 5 \quad 4)_h$$

İkili sayı sisteminde toplama

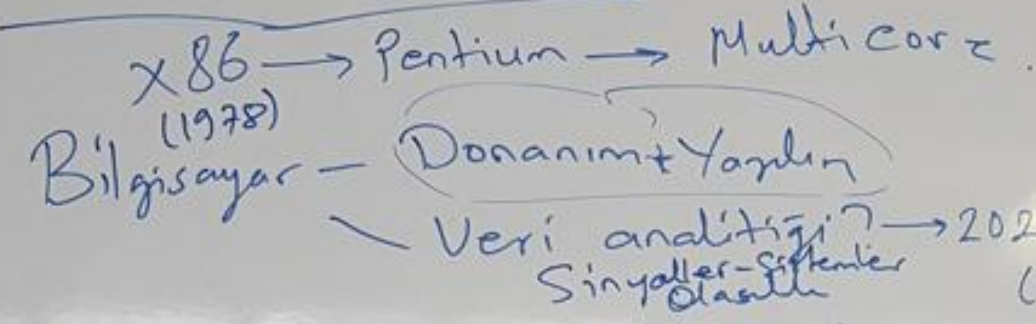
$$\begin{array}{cccccc} 0 & 0 & 1 & 1 & 1 & = 10 \\ +0 & +1 & +0 & +1 & +1 & = 11 \\ \hline 0 & 1 & 1 & 0 & 1 & \\ \text{elde} & & & & & \text{elde.} \end{array}$$

İkili sayı sisteminde bit ötelene (Ne tarafta sağ mı? Sol mu?)

$(10110101)_b$ ötele sağdan sola
 sıfır koyarak ötele 2) byte.
 8 bit
 16 bit
 ...

$$(101101010) = (16A)_h$$

CPU iç mimarisi



Philips
↓
ASML
Entegre/chip
%80

Registers: Geçici kayıt alanı.

1) - Genel amaçlı: 16 bit Ax, Bx, Cx, Dx

8 bit AH, AL, BH, BL, CH, CL, DH, DL

- Aritmetik, Transfer, cmp

- Sayıcı: CX - loop

- İndis: BX

- I/O: AX, DX

ES: BX, SI, DI

SS: BP, SP

CS: IP

2) - Segment Register (Belleğin Başlangıç adresi)

DS, SS, ES, CS: 16 bit → Fiziksel Adres

3) - İndis/Gösterge

20 bitlik segment register ile toplanarak belleğin ve bellek yönünün yeri belirlenir

4) - Flags. (Durum değişimleri: sıfır +/-, çift/tek, Karne, Karne, ...)

CPU-içi

Registers

ALU

Control Birimi:

Ön Bellek

Timing & Clock

System Bus. (A, D, C)

↓ ↓ ↓

Segment Register içeriği x 10h (16d)

Sağına 4 sifere ek

Intel Assembly

Assembly Programming

- Machine Language
 - binary
 - hexadecimal
 - machine code or object code
- Assembly Language
 - mnemonics
 - assembler
- High-Level Language
 - Python, Java Script, C++, Matlab
 - compiler

32-bit Instructions

- Instructions are represented in memory by a series of “opcode bytes.”
- A variance in instruction size means that disassembly is position specific.
- Most instructions take zero, one, or two arguments:

instruction destination, source

For example: `add eax, ebx`

is equivalent to the expression $eax = eax + ebx$

Program Segments

- A segment is an area of memory that includes up to 64K bytes
- Begins on an address evenly divisible by 16
- 8088/86 stayed compatible with 8085
 - Range of 1MB of memory, it has 20 address pins ($2^{20} = 1 \text{ MB}$)
 - Can handle 64KB of code, 64KB of data, 64KB of stack
- A typical Assembly language program consist of three segments:
 - Code segments
 - Data segment
 - Stack segment
 - Extra Segment

Program Segments

The 8086 fetches the instructions (opcodes and operands) from the code segments.

The 8086 address types:

- Physical address
 - Offset address
 - Logical address
- Physical address
 - Segment register içeriği 16 bit.
 - 20-bit address that is actually put on the address pins of 8086
 - Decoded by the memory interfacing circuitry
 - A range of 00000H to FFFFFH
 - It is the actual physical location in RAM or ROM within 1 MB mem. range
 - Offset address (indis)
 - A location within a 64KB segment range
 - A range of 0000H to FFFFH
 - Logical address
 - consist of a segment value and an offset address

Program Segments...*example*

Define the addresses for the 8086 when it fetches the instructions (opcodes and operands) from the code segments.

- Logical address:
 - Consist of a **CS** (code segment) and an **IP** (instruction pointer)
format is **CS:IP**
- Offset address
 - **IP** contains the offset address
- Physical address
 - generated by shifting the **CS** left one hex digit and then adding it to the **IP**
 - the resulting 20-bit address is called the physical address

Program Segments...*example*

Suppose we have:

CS	2500
IP	95F3

- Logical address:
 - Consist of a **CS** (code segment) and an **IP** (instruction pointer)
format is **CS:IP** **2500:95F3H**
- Offset address
 - **IP** contains the offset address which is **95F3H**
- Physical address
 - generated by shifting the **CS** left one hex digit and then adding it to the **IP**
25000 + 95F3 = 2E5F3H

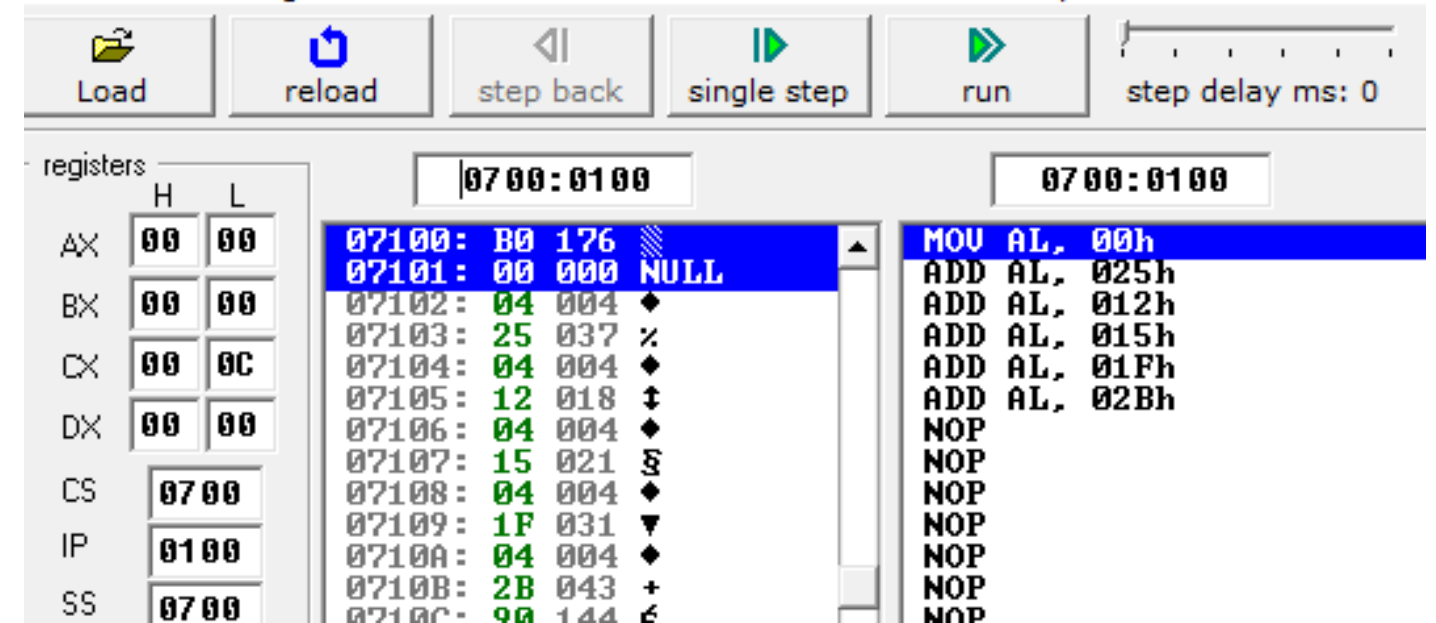
Example:

Add 5 bytes of data: 25H, 12H, 15H, 1FH,
2BH

Not using data segment

org 100h

```
MOV     AL,00H      ;clear AL
ADD     AL,25H      ;add 25H to AL
ADD     AL,12H
ADD     AL,15H
ADD     AL,1FH
ADD     AL,2BH
ret
```



Format:

CS:IP ; 0700:0100

Fiziksel adres: 07100h (07000+0100)

Fiziksel adres hesaplanırken sağ tarafa 0h ya da (0000)b, 4 bit 0 eklenir.

Program Segments

Data segment

Example: Önce 25h, 12h, 15h, 1fh, 2bh değerlerini DS: 4000h olan indis başlangıç adresi ise 200h olan bellek gözlerine yazınız.

Add 5 bytes of data: 25H, 12H, 15H, 1FH, 2BH

using data segment with a constant offset

Data location in memory:

```
Mov ax,1000h
```

```
Mov ds,ax
```

```
MOV [020h], 25h
```

```
MOV [021h], 12h
```

```
MOV [022h], 15h
```

```
MOV [023h], 1Fh
```

```
MOV [024h], 2Bh
```

Program Segments

Data segment

Example:

Add 5 bytes of data: 25H, 12H, 15H, 1FH, 2BH

```
org 100h
Mov ax,1000h
Mov ds,ax
MOV [020h], 25h
MOV [021h], 12h
MOV [022h], 15h
MOV [023h], 1Fh
MOV [024h], 2Bh
```

```
mov al,00h
add al, [20h]
add al, [21h]
add al, [22h]
add al, [23h]
add al, [24h]
ret
```

Program Segments

Data segment

Example: Önce 1234h, 5678h, 9abch, 0defh, 0abcdh değerlerini DS: 4000h olan indis başlangıç adresi ise 200h olan bellek gözlerine yazınız. Bellek gözünün fiziksel adresi 40200h den başlar. Neden? DS:4000h, DS register 16bit. Indis 200h. Fiziksel adres hesaplanır Segment register içeriği 10h ile çarpılır.

Yazılım ekranında aaaa:bbbb

aaaa: segment register

bbbb: indis

using data segment with a constant offset

Data location in memory:

MOV [0200h], 1234h

MOV [0202h], 5678h

MOV [0204h], 9abch

MOV [0206h], 0defh

MOV [0208h], 0abcdh

```
01 org 100h
02 Mov Ax, 4000h
03 Mov ds, Ax
04
05 MOV [0200h], 1234h
06 MOV [0202h], 5678h
07 MOV [0204h], 9abch
08 MOV [0206h], 0defh
09 MOV [0208h], 0abcdh
10
11 ret
12
13
```

emulator: noname.com_

file debug view virtual devices virtual drive help

LOAD reload step back single step run step delay ms: 0

registers	H	L
AX	40	00
BX	00	00
CX	00	24
DX	00	00
CS	07	00
IP	01	23
SS	07	00
SP	FF	FE
BP	00	00
SI	00	00
DI	00	00
DS	40	00
ES	07	00

4000:0200h	0700:0110
40200: 34 052 4	MOV AX, 04000h
40201: 12 018 ↑	MOV DS, AX
40202: 78 120 x	MOV w. [00200h],
40203: 56 086 V	MOV w. [00202h],
40204: BC 188 ↓	MOV w. [00204h],
40205: 9A 154 ü	MOV w. [00206h],
40206: EF 239 'ü	MOV w. [00208h],
40207: 0D 013 CI	RET
40208: CD 205 =	NOP
40209: AB 171 ½	NOP
4020A: 00 000 NI	NOP
4020B: 00 000 NI	NOP
4020C: 00 000 NI	NOP
4020D: 00 000 NI	NOP
4020E: 00 000 NI	NOP
4020F: 00 000 NI	...

screen source reset aux vars debug stack flags

Program Segments

Stack segment

Example:

Add 5 bytes of data: 25H, 12H, 15H, 1FH, 2BH

using stack segment with an offset register

Program:

```
org 100h
Mov AX, 1000h
MOV SS,AX
Mov BP, 200h
MOV SS:[BP], 25h
inc BP
MOV SS:[BP], 12h
inc BP
MOV SS:[BP], 15h
inc BP
MOV SS:[BP], 1Fh
inc BP
MOV SS:[BP], 2Bh

MOV AL,0
Mov BP, 200h
ADD AL,SS:[BP]
inc BP
ADD AL,SS:[BP]
inc BP
ADD AL,SS:[BP]
inc BP
ADD AL,SS:[BP]
inc BP
ADD AL,SS:[BP]
MOV SS:[BP+2], AL
```

registers		1000:200	
	H	L	
AX	10	96	10200: 25 037 %
BX	00	00	10201: 12 018 †
CX	00	46	10202: 15 021 §
DX	00	00	10203: 1F 031 ▼
CS	07	00	10204: 2B 043 +
IP	01	5A	10205: 00 000 NULL
SS	10	00	10206: 96 150 û
SP	FF	FE	10207: 00 000 NULL
BP	02	04	10208: 00 000 NULL
SI	00	00	10209: 00 000 NULL
DI	00	00	1020A: 00 000 NULL
DS	07	00	1020B: 00 000 NULL
ES	07	00	1020C: 00 000 NULL
			1020D: 00 000 NULL
			1020E: 00 000 NULL
			1020F: 00 000 NULL
			10210: 00 000 NULL
			10211: 00 000 NULL
			10212: 00 000 NULL
			10213: 00 000 NULL
			10214: 00 000 NULL
			10215: 00 000 NULL

Program Segments

Stack segment

Stack

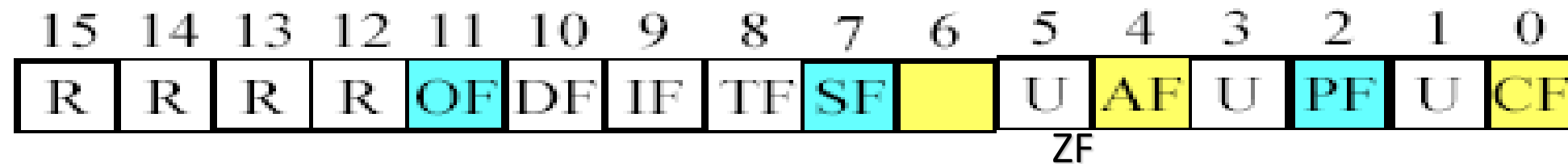
A section of RAM memory used by the CPU to store information temporarily.

- **Registers:** SS (Stack Segment) and SP (stack Pointer)
- **Operations: PUSH and POP**
 - **PUSH** – the storing of a CPU register in the stack
 - **POP** – loading the contents of the stack back into the CPU
- **Logical and offset address format:** SS:SP

Flag Register

- Flag Register (status register)

- 16-bit register
- Conditional flags: CF, PF, AF, ZF, SF, OF
- Control flags: TF, IF, DF



R = reserved

U = undefined

OF = overflow flag

DF = direction flag

IF = interrupt flag

TF = trap flag

SF = sign flag

ZF = zero flag

AF = auxiliary carry flag

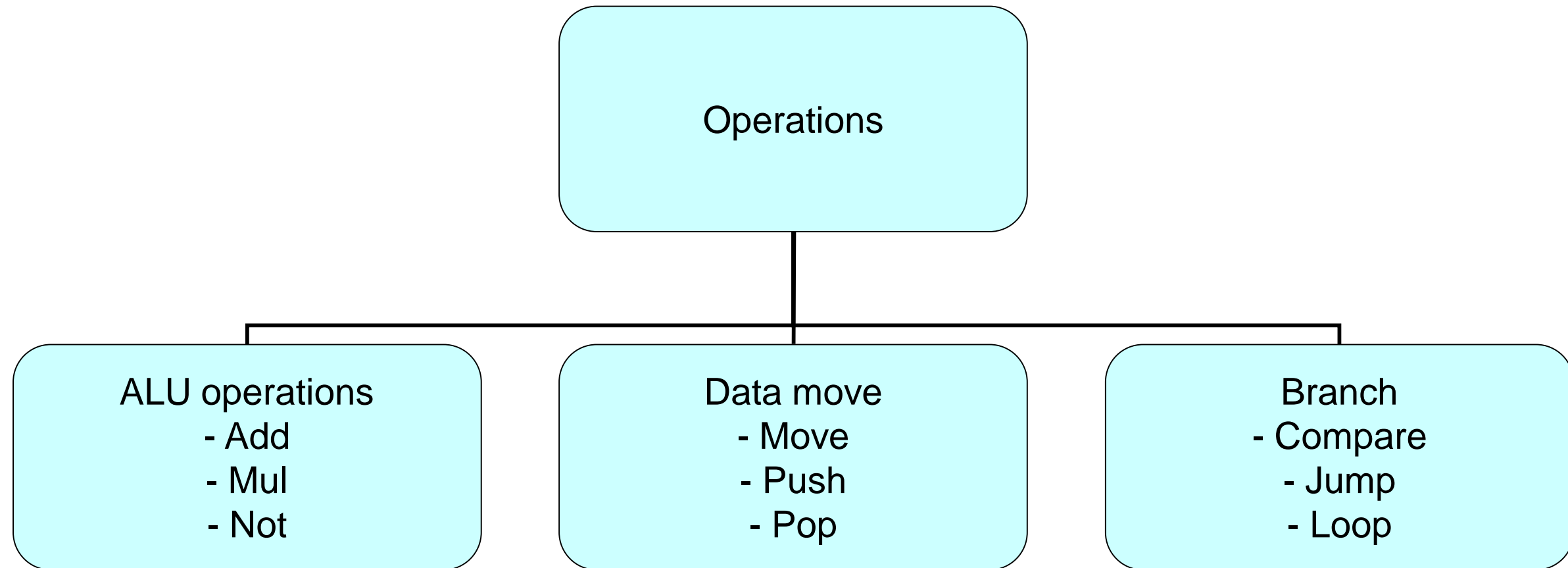
PF = parity flag

CF = carry flag



Assembly Komutlar

Instructions



Instruction Set

- 1. Data Transfer Instructions**
- 2. Arithmetic Instructions**
- 3. Logical Instructions**
- 4. Shift and Rotate Instructions**
- 5. Flow of Control**
- 6. String manipulation Instructions**
- 7. Process Control Instructions**
- 8. Control Transfer Instructions**

Data Transfer Instructions

- MOV dest, source dest ← source

- Stack instructions

PUSH byte ;increment stack pointer,
;move byte on stack

POP byte ;move from stack to byte,
;decrement stack pointer

Aritmetik

1 - T, ^{Carry} F, ^{Fi} Fikartma, B.

$$y = ax^2 + bx + c$$

$a, b, c, x \rightarrow y = ?$

2 - Ürsel

3 - Kompleks

4 - Oran - Yüzde

5 - Compare

6 - Mantılsal.

7 - Bit

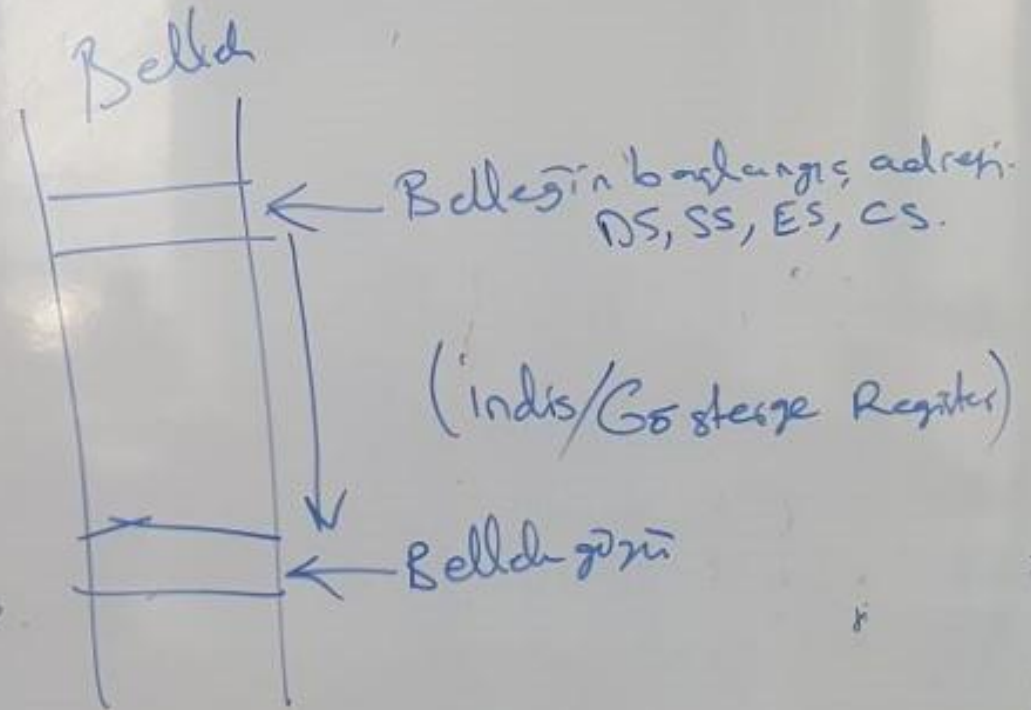
Transfer (Nereden nereye)

R → R
R ← B
B ← R
R ← S

! Veri transferinde hangi Register kullanılır

- Genel Amacı (Ar, B, ex, Px)

- Hepsini.



Section B.1: The 8086 Instruction Set

Page 865

MOV Move

Flags: Unchanged

Format: **MOV dest, source ; copy source to dest**

Function: Copy a word or byte from a register, memory location, or immediate number to a register or memory location. Source and destination must be of the same size and **cannot both be memory locations.**

Assembly Instruction format

General format



Example:

MOV DX,CX

AH	AL
BH	BL
CH	CL
DH	DL

org 100h

Mov cl,55h

Mov dl,cl

Mov ah,dl

Mov al,ah

Mov bh, cl

Mov ch, bh

What if ...

MOV AL,DX

Rule #1:

moving a value that is too large into a register will cause an error

```
MOV    BL,7F2H      ;Illegal: 7F2H is larger than 8 bits
MOV    AX,2FE456H   ;Illegal
```

Rule #2:

Data can be moved **directly** into **nonsegment registers only**

(Values cannot be loaded directly into any segment register.

To load a value into a segment register, first load it to a nonsegment register and then move it to the segment register.)

```
MOV    AX,2345H      MOV    DI,1400H
MOV    DS,AX         MOV    ES,DI
```

Rule #3:

If a value less than FFH is moved into a 16-bit register, the rest of the bits are assumed to be all zeros.

```
MOV BX, 5
```

```
BX = 0005
BH = 00, BL = 05
```

Stack.
2h den sonra

Mov Ax, Bx

Mov AL, Bx
↓ ↓
8bit 16bit

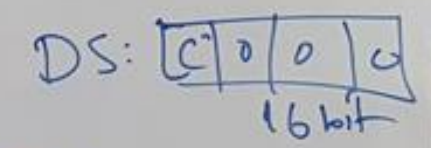
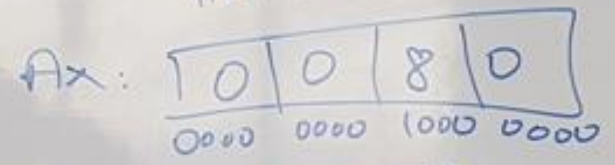
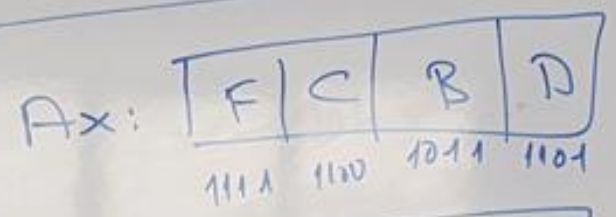
Mov Ax, 0FCBDh

Mov Ax, 80h

Mov AL, 80h, Ah: deşirme

Fiziksel adresi:
0201Eh → belleğine 2sh yapınız.
Ektra segment.
20bit.

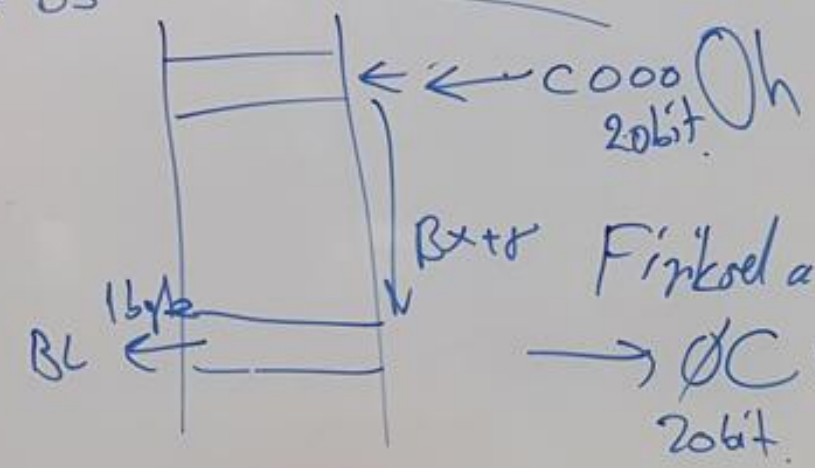
Mov Ax, 0000h
Mov Bx, 2000h
Mov [Bx+1E], 2sh.
ES



Mov BL, [Bx+8], Default DS

Mov Ax, 0000h
Mov DS, Ax

Mov [Bx+8], 80h



Arithmetic instructions

Arithmetic Instructions

- **Add**
- **Subtract**
- **Increment**
- **Decrement**
- Multiply
- Divide
- Decimal adjust

ADD ve SUB instructions

- Flag Register that may be affected
- Conditional flags: **CF**, PF, AF, **ZF**, **SF**, **OF**
- ADD - add second operand to first.
- SUB - Subtract second operand to first.

Arithmetic Instructions

Mnemonic	Description
ADD A, byte	add A to byte, put result in A
ADDC A, byte	add with carry
SUBB A, byte	subtract with borrow
INC A	increment A
INC byte	increment byte in memory
INC DPTR	increment data pointer
DEC A	decrement accumulator
DEC byte	decrement byte
MUL AB	multiply accumulator by b register
DIV AB	divide accumulator by b register
DA A	decimal adjust the accumulator

There are 3 groups of instructions. First Group

First group: ADD, SUB

These types of operands are supported: Add Varış, Kaynak; Varış=Varış+Kaynak

- REG, memory
- memory, REG
- REG, REG
- memory, immediate
- REG, immediate
- REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
- DS, ES; offset registers are Bx, DI, SI
- SS: offset registers are BP, DP
- Memory: [], If you see square bracket, this is memory address.
- ES:[...] this is extra segment
- memory: [BX], ES:[BX+SI+7]
- immediate: 5, -24, 3Fh, 0FAh, 10001101b, etc...
- İşlenenler arasındaki işlemden sonra, sonuç her zaman ilk işlenende saklanır.
- **CMP ve TEST komutları yalnızca bayrakları etkiler ve bir sonucu saklamaz** (bu komutlar programın yürütülmesi sırasında karar vermek için kullanılır).

ADD Signed or Unsigned ADD

Flags: Affected: OF,SF,ZF,AF,PF,CF

Format: **ADD dest, source ; dest=dest+source**

Function: Adds source operand to destination operand and places the result in destination. Both source and destination operands must match (e.g.,both byte size or word size) and only one of them can be in memory.

ADD Operand1, Operand2

Operand1, Operand2

ADD

REG, memory
memory, REG
REG, REG
memory,
immediate
REG,
immediate

Algorithm:

$operand1 = operand1 + operand2$

Example:

MOV AL, 5 ; AL = 5

ADD AL, -3 ; AL = 2

RET

C	Z	S	O	P	A
Γ	Γ	Γ	Γ	Γ	Γ

Add destination, source

destination=destination+source

- Org 100h
- Mov al, 25h
- Mov bl, 34h
- Add al, bl
- ret

- Org 100h
- Mov dh, 25h
- Add dh, 34h
- ret

ADD

- ADD Operand1, Operand2
- Algorithm: operand1 = operand1 + operand2

Operand1, Operand2:

- REG, memory
- memory, REG
- REG, REG
- memory, immediate
- REG, immediate

- **Example:**

```
Org 100h
MOV AL, 5 ; AL = 5
mov bl, -5
ADD AL, -3 ; AL = 2
add al,bl ; AL = -3
RET
```

Soru-1

- $(-5)_d = (?)_b$
- $(5)_d = (0000\ 0101)_b$
- İndis: 7 6 5 4 3 2 1 0
- $(0000\ 0101)_b$ ifadesinde 1'lerin yerine 0; 0'ların yerine 1 konur.
- $(1111\ 1010)_b + 1$
- $(1111\ 1011)_b = (FB)_h$

Soru-2

- $(FD)_h = (-?)_d$
- $(FD)_h = (1111\ 1101)_b$
- $(1111\ 1101)_b - (1)_b = (1111\ 1100)_b$
- $(1111\ 1100)_b$ ifadesinde 1'lerin yerine 0; 0'ların yerine 1 konur.
- $(0000\ 0011)_b$
- İndis: 7 6 5 4 3 2 1 0
- $(0000\ 0011)_b = (-3)_d$ ($3 = 2^1 + 2^0$)

Simple arithmetic instructions

- Both types of overflow occur **independently** and are signaled separately by CF

- Örnek:

```
Org 100h
mov al, 0FFh
add al, 1           ; AL=00h, OF=0, CF=1
```

Örnek:

```
Org 100h
mov ax, 0FFh
add ax, 1
```

Sonuç: ax=(0100)h= (0000 0001 0000 0000)b=(2⁸)d=256d

Simple arithmetic instructions

- Both types of overflow occur **independently** and are signaled separately by CF and OF

Örnek:

```
mov al,7Fh
```

```
add al, 1 ; AL=80h, OF=1, CF=0
```

```
mov al,80h
```

```
add al,80h ; AL=00h, OF=1, CF=1
```

- Hence: we can have either type of overflow or both of them at the same time

ADC:Add with Carry.

- ADC Operand1, Operand2
- Algorithm: operand1 = operand1 + operand2 + CF

Operand1, Operand2:

- REG, memory
- memory, REG
- REG, REG
- memory, immediate
- REG, immediate

Al:254d x

Bl: 6d

Add al, bl

Al:260 olması gerekirken al 8bitlik register olduğu için 4d değer geldi; fakat

CF: 1 olduğu için

Sonuc: 104h

$(0001\ 0000\ 0100)_b = (2^2 + 2^8)_d = (4 + 256)_d = (260)_d$

• Example:

Org 100h

Mov al, 254

mov bl, 6

add al,bl

RET

; registerlar 8bitlik yerine 16bit alınırsa CF'e gerek kalmadan toplama yapılır.

Org 100h

mov ax, 254

mov bx, 6

add ax,bx

RET

ADC Operan1, Operand2

ADC

REG, memory
memory, REG
REG, REG
memory,
immediate
REG,
immediate

Algorithm:

$\text{operand1} = \text{operand1} + \text{operand2} + \text{CF}$

Example:

```
STC      ; set CF = 1
MOV AL, 5 ; AL = 5
ADC AL, 1 ; AL = 7
RET
```

C	Z	S	O	P	A
r	r	r	r	r	r

```
org 100h
Mov al,01h
stc
adc al,02h
ret
```

SUB Operan1, Operand2

SUB

REG, memory
memory, REG
REG, REG
memory,
immediate
REG,
immediate

Subtract.

Algorithm:

$\text{operand1} = \text{operand1} - \text{operand2}$

Example:

MOV AL, 5

SUB AL, 1 ; AL = 4

RET

C	Z	S	O	P	A
r	r	r	r	r	r

SUB: Subtract

- SUB Operand1, Operand2
- Algorithm: $\text{operand1} = \text{operand1} - \text{operand2}$

Operand1, Operand2:

- REG, memory
- memory, REG
- REG, REG
- memory, immediate
- REG, immediate

- **Example:**

```
Org 100h
```

```
MOV AL, 5
```

```
Mov BL, -20h
```

```
SUB AL, BL ; AL = AL-BL
```

```
RET
```

SBB Operand1, Operand2

SBB

REG, memory
memory, REG
REG, REG
memory,
immediate
REG,
immediate

Subtract with Borrow.

Algorithm:

$\text{operand1} = \text{operand1} - \text{operand2} - \text{CF}$

Example:

STC

MOV AL, 5

SBB AL, 3 ; $\text{AL} = 5 - 3 - 1 = 1$

RET

C	Z	S	O	P	A
r	r	r	r	r	r

SBB: Subtract with Borrow

- SBB Operand1, Operand2
- Algorithm: $\text{operand1} = \text{operand1} - \text{operand2} - \text{CF}$

Operand1, Operand2:

- REG, memory
- memory, REG
- REG, REG
- memory, immediate
- REG, immediate

- **Example:**

```
STC
```

```
MOV AL, 5
```

```
SBB AL, 3 ; AL = 5 - 3 - 1 = 1
```

```
RET
```

Examples

```
org 100h  
mov al,5  
mov bl, -3  
mov cl, 10  
sub bl,al  
add cl, bl  
ret
```

Dizi toplamını ve ortalamasını bulan assembly program

```
org 100h
    lea bx, vec1 ; dizinin baslangic adresini bx'e aktarir.
    mov cl, N ; dizideki eleman sayisi
    mov al, 0
sum:
    add al, [bx] ; Segment register: DS, ofset reg bx,
    bellek cell: ds*10h+bx; bellek gozundeki deger ile al
    icerigini toplar
    inc bx      ; bx=bx+1
    loop sum   ; cl=cl-1
    div N      ; al=al/n
    ret

vec1 db 2, 4, 6, 8 ,10
N db 5
```

Loop komutu CL register'ın içeriği sıfır olana dek aradaki işlemleri yapar. CL'nin içeriği her döngüde 1 azalır.

```
org 100h

    mov cl, 6 ; dizideki eleman sayisi
    mov al,0
sum:

    inc al ; al=al+1
    loop sum ; cl=cl-1

    mov bh, 55h
    mov bl, 0cch

    ret
```

Fiziksel adres=Segment adres * 10h + Ofset adres

Segment(16bit): Ofset (16 bit)

The screenshot shows an emulator window titled "emulator: noname.com_". The interface includes a menu bar (file, debug, view, virtual devices, virtual drive, help), a toolbar with buttons for LOAD, reload, step back, single step, run, and a step delay ms: 0 field. Below the toolbar is a "registers" section with a table of CPU registers. To the right of the registers is a memory dump window showing addresses, hex values, and assembly instructions. The assembly window is currently displaying instructions starting at address 07113.

registers	H	L
AX	00	00
BX	01	13
CX	00	19
DX	00	00
CS	0700	
IP	0103	
SS	0700	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

Address	Hex	Hex	Symbol
07113:	02	002	⊖
07114:	04	004	◆
07115:	06	006	♠
07116:	08	008	B
07117:	0A	010	NI
07118:	05	005	♣
07119:	90	144	⌵
0711A:	90	144	⌵
0711B:	90	144	⌵
0711C:	90	144	⌵
0711D:	90	144	⌵
0711E:	90	144	⌵
0711F:	90	144	⌵
07120:	90	144	⌵
07121:	90	144	⌵
07122:	90	144	⌵


```
ADD AL, [SI]
PUSH ES
OR [BP + SI], CL
ADD AX, 09090h
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

Dizi ortalamasını bulan assembly program

```
ORG 100h
```

```
mov cl,N
```

```
lea dx,a
```

```
mov bx, offset a
```

```
mov al,0
```

```
mov si,0
```

```
don:
```

```
add al,a[si]
```

```
inc si
```

```
loop don
```

```
div N
```

```
ret
```

```
a db 5,6,3,2,7,8,5,4,5
```

```
N db 9
```

org 100h

lea bx, vec1 ; vec1 dizisinin baslangic adresini bx'e aktarir.

lea si, vec2; vec2 dizisinin baslangic adresini si'e aktarir.

mov cl, N ; dizideki eleman sayisi

mov al, 0

sum:

add al, [si] ; Segment register: DS, ofset reg si, bellek cell fiziksel adresi: ds*10h+si;

inc si ; si=si+1

loop sum ; cl=cl-1

div N ; al=al/n

ret

vec1 db 2, 4, 6, 8 ,10 ; vec1[0]=2, vec1[1]=4,

vec2 db 5, 15, 16, 18 ,20

N db 5

INC, DEC

- These types of operands are supported: REG, memory
- REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
- memory: [BX], [BX+SI+7], variable, etc...
- INC, DEC instructions affect these flags only: ZF, SF, OF, PF, AF.

INC & DEC Instructions

- INC AX legal
- DEC BL legal
- INC [BX] illegal
- INC Byte PTR [BX] legal
- DEC I legal
- INC DS illegal

INC

REG
memory

operand = operand + 1

Example:

MOV AL, 4

INC AL ; AL = 5

RET

Z	S	O	P	A
r	r	r	r	r

CF - unchanged!

DEC

REG
memory

operand = operand - 1

Example:

MOV AL, 255 ; AL = 0FFh (255 or -1)

DEC AL ; AL = 0FEh (254 or -2)

RET

Z	S	O	P	A
r	r	r	r	r

CF - unchanged!

INC Operand (Increment)

- **INC Operand**
- Algorithm: operand = operand + 1

Operand:

- REG
- Memory

Example:

```
ORG 100h
```

```
MOV AL, 4
```

```
INC AL ; AL = 5
```

```
RET
```

INC

INC

REG
memory

Increment.

Algorithm:

$\text{operand} = \text{operand} + 1$

Example:

MOV AL, 4

INC AL ; AL = 5

RET

Z	S	O	P	A
r	r	r	r	r

CF - unchanged!

DEC Operand (Decrement)

- DEC Operand
- Algorithm: $\text{operand} = \text{operand} - 1$

Operand:

- REG
- Memory

- Example:

```
MOV AL, 255 ; AL = 0FFh (255 or -1)
```

```
DEC AL ; AL = 0FEh (254 or -2)
```

```
RET
```

Arithmetic Instructions

- Add
- Subtract
- Increment
- Decrement
- **Multiply**
- **Divide**
- Decimal adjust

Second group: MUL, IMUL, DIV, IDIV

- These types of operands are supported: REG, memory
- REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
- memory: [BX], [BX+SI+7], variable, etc...
- MUL and IMUL instructions affect these flags only: CF, OF
- When result is over operand size these flags are set to 1, when result fits in operand size these flags are set to 0.

MUL / IMUL – DIV - IDIV

- MUL - Unsigned multiply: «Mul operand»
 - when operand is a byte: $AX = AL * \text{operand}$.
 - when operand is a word: $(DX AX) = AX * \text{operand}$.
- IMUL - Signed multiply: Imul operand
 - when operand is a byte: $AX = AL * \text{operand}$.
 - when operand is a word: $(DX AX) = AX * \text{operand}$.
- DIV - Unsigned divide: Div Operand
 - when operand is a byte: $AL = AX / \text{operand}$; $AH = \text{remainder (modulus)}$. .
 - when operand is a word: $AX = (DX AX) / \text{operand}$; $DX = \text{remainder (modulus)}$.
- IDIV - Signed divide:
 - when operand is a byte: $AL = AX / \text{operand}$; $AH = \text{remainder (modulus)}$. .
 - when operand is a word: $AX = (DX AX) / \text{operand}$; $DX = \text{remainder (modulus)}$. .

MUL Operand (Unsigned multiply)

- MUL Operand

Algorithm:

- when operand is a byte: $AX = AL * \text{Operand}$.
- when operand is a word: $(DX AX) = AX * \text{Operand}$.

$$\begin{aligned}(320)h &= (0011\ 0010\ 0000)b \\ &= (2^5 + 2^8 + 2^9)d = (32 + 256 + 512)d \\ &= (32 + 768)d = (800)d\end{aligned}$$

Operand:

Flags: CF, OF deđiřir.

- REG
- memory

- **Example:**

```
ORG 100h
```

```
MOV Ax, 8000h
```

```
MOV Bx, 1000h
```

```
MUL Bx ; DX, AX = BX*AX, DX:0800, AX:0000, Sonu=(800 0000)h
```

```
RET
```

Example:

```
Org 100h
```

```
Mov al,10h
```

```
Mov bl,12h
```

```
Mul bl
```

```
ret
```

IMUL Operand (Signed multiply)

- **IMUL Operand**

Algorithm:

- when operand is a byte: $AX = AL * \text{operand}$.
- when operand is a word: $(DX\ AX) = AX * \text{operand}$.

Operand:

- REG
- memory

- **Example:**

```
Org 100h
MOV AL, -2
MOV BL, -4
IMUL BL ; AX = 8
RET
```

IMUL instruction

- IMUL (signed integer multiply) multiplies an 8-, 16-, or 32-bit signed operand by either AL, AX, or EAX (there are one/two/three operand format)
- Preserves the sign of the product by sign-extending it into the upper half of the destination register

Example: multiply $48 * 4$, using 8-bit operands:

```
mov    al,48
mov    bl,4
imul  bl    ; AX = 00C0h, OF=1
```

OF=1 because AH is not a sign extension of AL.

Örnek

when operand is a word: $(DX AX) = AX * \text{operand}$.

In this example, Operand is Bx

Org 100h

Mov Bx, 2000h

Mov Ax, 3000h

Mul Bx

Sonuç= DxAx: (06 00 00 00)h

	H	L	
AX	00	00	07100: BB 187 7
BX	20	00	07101: 00 000 NULL
CX	00	08	07102: 20 032 SPA
DX	06	00	07103: B8 184 7
CS	07	00	07104: 00 000 NULL
IP	01	08	07105: 30 048 0
SS	07	00	07106: F7 247 ≈
SP	FF	FE	07107: E3 227 0
BP	00	00	07108: 90 144 É
SI	00	00	07109: 90 144 É
DI	00	00	0710A: 90 144 É
DS	07	00	0710B: 90 144 É
ES	07	00	0710C: 90 144 É
			0710D: 90 144 É
			0710E: 90 144 É
			0710F: 90 144 É
			07110: 90 144 É
			07111: 90 144 É
			07112: 90 144 É
			07113: 90 144 É
			07114: 90 144 É
			07115: 90 144 É

screen source reset

Örnek

when operand is a byte: $AX = AL * \text{operand}$.

In this example, Operand is Bl

Org 100h

Mov Bl, 20h

Mov Al, 30h

Mul Bl

Sonuç= Ax:(06 00)h

registers		07 00 : 01 06	
	H	L	
AX	06	00	
BX	00	20	
CX	00	06	
DX	00	00	
CS	07 00		
IP	01 06		
SS	07 00		
SP	FF FE		
BP	00 00		
SI	00 00		
DI	00 00		
DS	07 00		
ES	07 00		
07100:	B3	179	
07101:	20	032	SPA
07102:	B0	176	⌘
07103:	30	048	0
07104:	F6	246	÷
07105:	E3	227	π
07106:	90	144	É
07107:	90	144	É
07108:	90	144	É
07109:	90	144	É
0710A:	90	144	É
0710B:	90	144	É
0710C:	90	144	É
0710D:	90	144	É
0710E:	90	144	É
0710F:	90	144	É
07110:	90	144	É
07111:	90	144	É
07112:	90	144	É
07113:	90	144	É
07114:	90	144	É
07115:	90	144	É

DIV Operand

DIV

REG
memory

Unsigned divide.

Algorithm:

when operand is a **byte**:

AL = AX / operand

AH = remainder (modulus)

when operand is a **word**:

AX = (DX AX) / operand

DX = remainder (modulus)

Example:

MOV AX, 203 ; AX = 00CBh

MOV BL, 4

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

RET

C	Z	S	O	P	A
?	?	?	?	?	?

IDIV Operand

IDIV

REG
memory

Signed divide.

Algorithm:

when operand is a **byte**:

$AL = AX / \text{operand}$

AH = remainder (modulus)

when operand is a **word**:

$AX = (DX\ AX) / \text{operand}$

DX = remainder (modulus)

Example:

MOV AX, -203 ; AX = 0FF35h

MOV BL, 4

IDIV BL ; AL = -50 (0CEh), AH = -3 (0FDh)

RET

C	Z	S	O	P	A
?	?	?	?	?	?

Örnek

when operand is a word: $AX = (DX\ AX) / \text{operand};$
 $DX = \text{remainder (modulus)}. .$

In this example, Operand is Bx

```
Org 100h
```

```
Mov Bx, 2000h
```

```
Mov Ax, 8000h
```

```
Div Bx
```

```
Sonuç= Ax:(00 004)h
```

DIV Operand

Algorithm:

- when operand is a byte: $AL = AX / \text{operand}$, $AH = \text{remainder (modulus)}$
- when operand is a word: $AX = (DX AX) / \text{operand}$, $DX = \text{remainder (modulus)}$

Operand

- REG
- memory
- Unsigned divide.

- **Example:**

```
Org 100h
```

```
MOV AX, 255d ;
```

```
MOV BL, 10d
```

```
DIV BL ; Bolum: AL, Kalan:AH
```

```
RET
```

DIV Operand

Algorithm:

- when operand is a byte: $AL = AX / \text{operand}$, $AH = \text{remainder (modulus)}$
- when operand is a word: $AX = (DX AX) / \text{operand}$, $DX = \text{remainder (modulus)}$

Operand

- REG
- memory
- Unsigned divide.

- **Example:**

```
Org 100h ; DXAX/BX
```

```
Mov DX, 20000d
```

```
MOV AX, 10011d ;
```

```
MOV BX, 1000d
```

```
DIV BX ;
```

```
RET
```

IDIV Operand (Signed divide)

IDIV Operand

Algorithm:

- when operand is a byte: $AL = AX / \text{operand}$; $AH = \text{remainder (modulus)}$
- when operand is a word: $AX = (DX AX) / \text{operand}$; $DX = \text{remainder (modulus)}$

Operand:

- REG,
- memory

- **Example:**

```
Org 100h
```

```
MOV AX, -203 ; AX = 0FF35h
```

```
MOV BL, 4
```

```
IDIV BL ; AL = -50 (0CEh), AH = -3 (0FDh)
```

```
RET
```

IDIV Operand

- IDIV (signed divide) performs signed integer division
- Uses same operands as DIV

Example: 8-bit division of -48 by 5

```
mov al,-48
cbw          ; extend AL into AH
mov bl,5
idiv bl     ; AL = -9, AH = -3
```

Logical Operations

Sayısal Mantığın Temelleri

- Sayısal mantık değerlerini temsil etmek için ikili sayı sistemi kullanılır.
- Bit:1/0 (Doğru /Yanlış, İyi /Kötü, Gece/Gündüz, 0V /5V)
- Sayısal mantık değerlerinin matematiksel işlemleri, Boole cebirinin kurallarında belirtilen yasalara tabidir.
- Boole cebirinin kurallarında belirtilen yasaların matematiksel girdileri ve çıktıları ikili sayı (1/0) sistemi ile temsil edilir.
- Her türlü aritmetiksel işlemlerin girdileri ve çıktıları ya 1 ya da 0'dır.
- Bilgisayar sistemlerinin donanımını oluşturan mantıksal kapıların girişler, 1/0; çıkışları: 1/0
 - AND, OR, NOT, NAND, NOR, XOR, XNOR.
- Mantık kapıları, transistörler kullanılarak oluşturulur.
 - NOT gate can be implemented by a single transistor
 - AND gate requires 3 transistors
- Transistörler bilgisayar sistemlerinin temel devre elemanlarıdır.
 - Pentium consists of 3 million transistors
 - Compaq Alpha consists of 9 million transistors
 - Now we can build chips with more than 100 million transistors

Boole Cebri Teoremleri

1. a) $a+b=b+a$ Değişme Özelliği
b) $a \cdot b = b \cdot a$
2. a) $a+b+c=a+(b+c)$ Birleşme Özelliği
b) $a \cdot b \cdot c = a \cdot (b \cdot c)$
3. a) $a+b \cdot c = (a+b) \cdot (a+c)$ Dağılma Özelliği
b) $a \cdot (b+c) = a \cdot b + a \cdot c$
4. a) $a+a=a$ Değişkende Fazlalık Özelliği
b) $a \cdot a = a$
5. a) $a+a \cdot b = a$ Yutma Özelliği
b) $a \cdot (a+b) = a$
6. a) $(a)^n = a$ işlemde Fazlalık Özelliği
b) $(a \times n) = a$
7. a) $\overline{(a + b)} = \bar{a} \cdot \bar{b}$ De Morgan Kuralı
b) $\overline{(a \cdot b)} = \bar{a} + \bar{b}$

8. a) $a+\bar{a} = 1$ Sabit Özelliği
b) $a \cdot \bar{a} = 0$
9. a) $0+a=a$ Etkisizlik Özelliği
b) $1 \cdot a = a$
10. a) $1+a=1$ Yutan Sabit Özelliği
b) $0 \cdot a = 0$
11. a) $(a+b) \cdot b = b$
b) $a \cdot b + b = b$

- The 12 Rules of Boolean Algebra

- $A + 0 = A$
- $A + 1 = 1$
- $A \cdot 0 = 0$
- $A \cdot 1 = A$
- $A + A = A$
- $A + \bar{A} = 1$
- $A \cdot A = A$
- $A \cdot \bar{A} = 0$
- $\overline{\bar{A}} = A$
- $A + AB = A$
- $A + \bar{A}B = A + B$
- $(A + B)(A + C) = A + BC$

Mantıksal işlemlerde tüm değişkenler 1/0 iki sayı sisteminde çalışır.

Giriş ya da çıkışlar: 1/0

Boole Cebirinin Kuralları ve Yasaları

Tek değişkenli temel kurallar:

- Boole Cebirinin Kurallarının ve Yasalarının her birinin bir kanıtı, değişkenin yalnızca iki bit(0/1) değere sahip olabileceği gerçeğinden yararlanılarak kolayca ispat edilebilir.
- Not: $A=0$ ya da 1 olur.
- $A + A + A + A + A \dots + A + 1 = 1$; OR kapısında girişlerden herhangi biri 1 ise çıkış birdir. Diğer ispat etme yöntemi, 1 ve 0 değerleri verilerek doğruluk aranır.
- $A + A + A + \dots + A = A$ (Neden? İki değişkenli 0 ya da 1 girişler mevcuttur)
- $(a+b) \times b = a \times b + b \times b = a \times b + b = (a+1) \times b = b$; $b \times b = b$, $a+1=1$
- $1 + 1 + 1 + 1 + 1 = 1$
- $A A A \dots A = A$
 - If $A = 0$ then $0 + 1 = 1$
 - If $A = 1$ then $1 + 1 = 1$

$$\begin{array}{ll} A + 0 = A & A \cdot 0 = 0 \\ A + 1 = 1 & A \cdot 1 = A \\ A + A = A & A \cdot A = A \\ A + \bar{A} = 1 & A \cdot \bar{A} = 0 \end{array}$$

It should be noted that $\bar{\bar{A}} = A$

Boole Cebirinin Kuralları ve Yasaları

Mantıksal matematiği ikili (0/1) sayı sisteminde hesaplama yapılır.

Soru: Boole cebirini kullanarak aşağıdaki işlemi yapınız. $A+1+1+1=?$

a) 0 b)1 c)10 d) 8 A+3

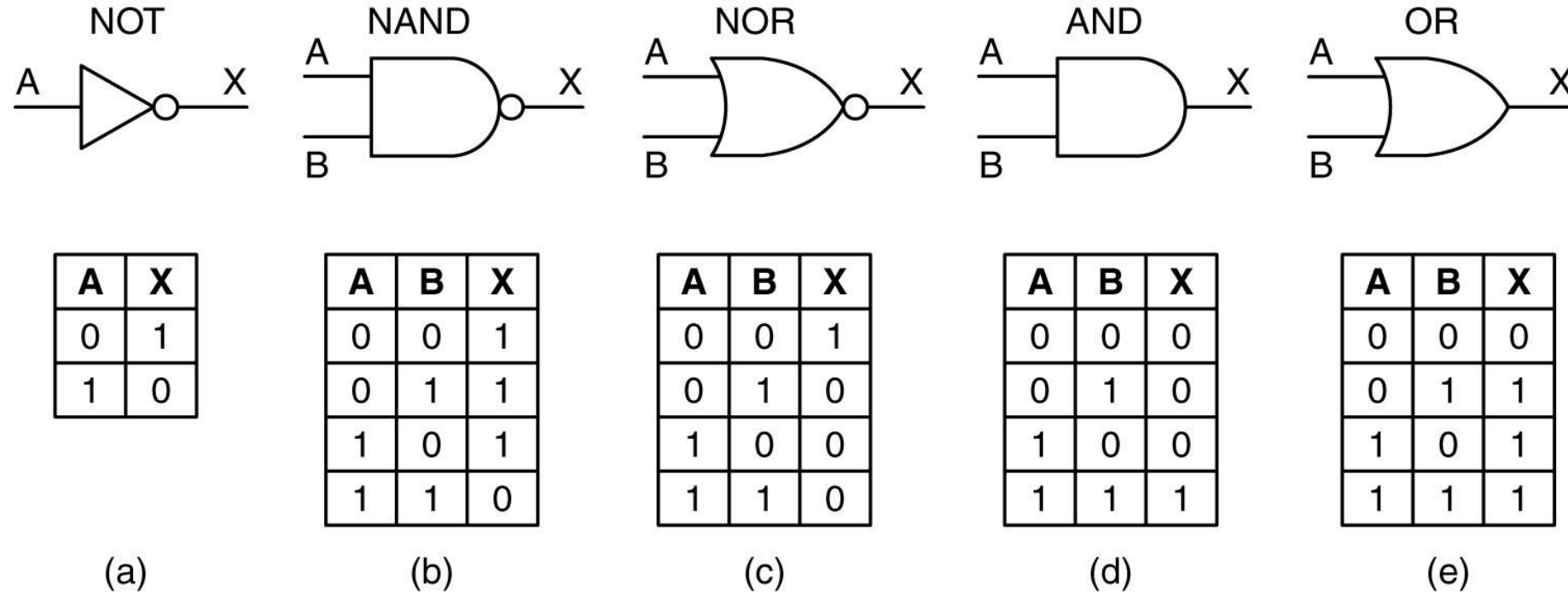
Soru: Boole cebirinde A hangi değerleri alır? a) 0 b)1 c) 0/1 d) 0,1,2, ..., 9 d)Her değeri e)hiçbir değeri

Soru: Boole cebirinde $A=1$ ise, $A+A+A+A+A=?$ A)1 B)0 C)A D)5 D)5A E) Hiçbiri

Soru: Boole Cebirinde $A*A*A*A=?$ A)A B) A^4

Soru: $1+a+b+c+ \dots + z=?$

Mantık Kapıları için semboller ve işlevsel davranış



AND Kapısı: Girişlerden herhangi biri 0 ise çıkış 0 dır. Girişlerin tümü 1 ise çıkış 1 dir.

OR Kapısı: Girişlerden herhangi biri 1 ise çıkış 1 dır. Girişlerin tümü 0 ise çıkış 0 dir.

NOT Kapısı: girişin evriğini alır.

Mantık kapısı, bir Boole işlevini uygulayan idealleştirilmiş veya fiziksel bir devredir, yani bir veya daha fazla mantık girişinde mantıksal bir işlem gerçekleştirir ve tek bir mantık çıkışı üretir.

Toplama

Logic Gates

Karşılaştırma

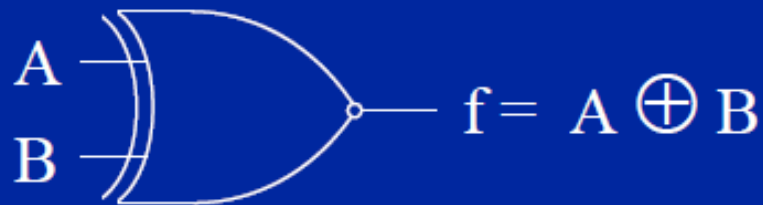
The **EXCLUSIVE OR** Truth Table

A	B	f = A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

The **EXCLUSIVE NOR** Truth Table

A	B	f = A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

The **XOR** Gate



EXCLUSIVE NOR Gate



This is called the equivalence gate

Karşılaştırma ve Aritmetik toplama işlemlerinde XOR kapıları kullanılır.

Girişlerin tümü birbirine eşit (0 ya da 1) çıkış sıfır ise XOR, çıkış 1 ise XNOR kapısıdır.

Logic Instructions

- Bitwise logic operations
 - ❖ (AND, OR, XOR, NOT)
- Clear
- Rotate
- Swap

Logic instructions do **NOT** affect the flags in PSW

Bitwise Logic

AND → AND

OR → OR

XOR → XOR

CP → Complement

Examples:

	00001111
AND	<u>10101100</u>
	00001100

	00001111
OR	<u>10101100</u>
	10101111

	00001111
XOR	<u>10101100</u>
	10100011

	<u>10101100</u>
CP	01010011

Boolean Operation Instructions

- NOT
- AND
- OR
- XOR

- Example: OR EAX, EBX

These instructions affect these flags only:

CF, ZF, SF, OF, PF, AF.

- AND - Logical AND between all bits of two operands.
- These rules apply: $1 \text{ AND } 1 = 1$; $1 \text{ AND } 0 = 0$; $0 \text{ AND } 1 = 0$; $0 \text{ AND } 0 = 0$. As you see we get 1 only when both bits are 1.
- TEST - The same as AND but for flags only.
- OR - Logical OR between all bits of two operands. These rules apply: $1 \text{ OR } 1 = 1$; $1 \text{ OR } 0 = 1$; $0 \text{ OR } 1 = 1$; $0 \text{ OR } 0 = 0$
- **XOR** - Logical XOR (exclusive OR) between all bits of two operands. These rules apply: $1 \text{ XOR } 1 = 0$; $1 \text{ XOR } 0 = 1$; $0 \text{ XOR } 1 = 1$; $0 \text{ XOR } 0 = 0$. As you see we get **1** every time when bits are different from each other.

AND Operand1, Operand2 (Operand1=Operand1 and Operand2)

AND

REG, memory

memory, REG

REG, REG

memory,

immediate

REG,

immediate

Logical AND between all bits of two operands. Result is stored in operand1.

These rules apply:

1 AND 1 = 1

1 AND 0 = 0

0 AND 1 = 0

0 AND 0 = 0

Example:

MOV AL, 'a' ; AL = 01100001b

AND AL, 11011111b ; AL = 01000001b ('A')

RET

C	Z	S	O	P
0	r	r	0	r

AND: Logical AND between all bits of two operands.

- AND Operand1, Operand2

Operand1, Operand2:

- REG, memory
- memory, REG
- REG, REG
- memory, immediate
- REG, immediate

Not: Logical AND between all bits of two operands. Result is stored in operand1.

- These rules apply:
- 1 AND 1 = 1
- 1 AND 0 = 0
- 0 AND 1 = 0
- 0 AND 0 = 0

- **Example:**

```
org 100h
MOV AL, 01100001b
Mov Ah, 01000000b
```

```
Mov Bh, 00111000b
Mov BL, 00001010b
```

```
AND AL,AH
OR BH,BL
NOT BH
```

```
RET
```

```
org 100h
MOV AL, 27h
Mov Ah, 0cch
or AL,Ah
RET
```

OR operand1, operand2

OR

REG, memory
memory, REG
REG, REG
memory,
immediate
REG,
immediate

Logical OR between all bits of two operands. Result is stored in first operand.

These rules apply:

1 OR 1 = 1

1 OR 0 = 1

0 OR 1 = 1

0 OR 0 = 0

Example:

MOV AL, 'A' ; AL = 01000001b

OR AL, 00100000b ; AL = 01100001b ('a')

RET

C	Z	S	O	P	A
0	r	r	0	r	?

OR: Logical OR

- OR Operand1, Operand2

Operand:

- REG, memory
- memory, REG
- REG, REG
- memory, immediate
- REG, immediate

Not: Logical OR between all bits of two operands. Result is stored in first operand.

- These rules apply:
- $1 \text{ OR } 1 = 1$
- $1 \text{ OR } 0 = 1$
- $0 \text{ OR } 1 = 1$
- $0 \text{ OR } 0 = 0$

- **Example:**

```
MOV AL, 'A' ; AL = 01000001b
```

```
OR AL, 00100000b ; AL = 01100001b ('a')
```

```
RET
```

XOR (Aritmetik Toplama)

XOR

REG, memory
memory, REG
REG, REG
memory,
immediate
REG,
immediate

Logical XOR (Exclusive OR) between all bits of two operands. Result is stored in first operand.

These rules apply:

1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0

Example:

```
MOV AL, 00000111b
XOR AL, 00000010b ; AL = 00000101b
RET
```

C	Z	S	O	P	A
0	r	r	0	r	?

XOR: Logical XOR (Exclusive OR) between all bits of two operands.

- **XOR Operand1, Operand2**

Operand1, Operand2:

- REG, memory
- memory, REG
- REG, REG
- memory, immediate
- REG, immediate

Not: Logical XOR (Exclusive OR) between all bits of two operands. Result is stored in first operand.

- These rules apply:
- $1 \text{ XOR } 1 = 0$
- $1 \text{ XOR } 0 = 1$
- $0 \text{ XOR } 1 = 1$
- $0 \text{ XOR } 0 = 0$

- **Example:**

```
MOV AL, 00000111b
```

```
XOR AL, 00000010b ; AL = 00000101b
```

```
RET
```

NOT, NEG

- These types of operands are supported: REG, memory
- REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
- memory: [BX], [BX+SI+7], variable, etc...
- NOT instruction does not affect any flags!
- NEG instruction affects these flags only: CF, ZF, SF, OF, PF, AF.
- NOT - Reverse each bit of operand.
- NEG - Make operand negative (two's complement). Actually it reverses each bit of operand and then adds 1 to it. For example 5 will become -5, and -2 will become 2.

NOT Operand

NOT

REG
memory

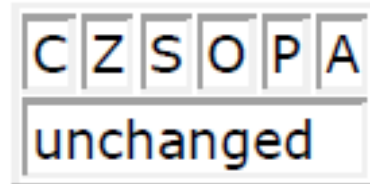
Invert each bit of the operand.

Algorithm:

- if bit is 1 turn it to 0.
- if bit is 0 turn it to 1.

Example:

```
MOV AL, 00011011b
NOT AL ; AL = 11100100b
RET
```



```
org 100h
MOV AL, 'a' ; AL = 01100001b
Mov Ah, 'A'
Mov Bh, '8'
Mov BL, 8
```

```
NOT AL
NOT AH
NOT BL
NOT BH
```

```
RET
```

```
AL: (61)h=(0110 0001)b
NOT AL= (1001 1110)b=(9E)h
```

NOT: Invert each bit of the operand.

- NOT Operand

Algorithm:

- if bit is 1 turn it to 0.
- if bit is 0 turn it to 1.

Operand:

- REG
- memory

- **Example:**

```
MOV AL, 00011011b
```

```
NOT AL ; AL = 11100100b
```

```
RET
```

NEG

NEG

REG
memory

Negate. Makes operand negative (two's complement).

Algorithm:

- Invert all bits of the operand
- Add 1 to inverted operand

Example:

```
MOV AL, 5 ; AL = 05h  
NEG AL ; AL = 0FBh (-5)  
NEG AL ; AL = 05h (5)  
RET
```

C	Z	S	O	P	A
r	r	r	r	r	r

TEST

TEST

REG, memory
memory, REG
REG, REG
memory,
immediate
REG,
immediate

Logical AND between all bits of two operands for flags only. These flags are effected: **ZF, SF, PF**. Result is not stored anywhere.

These rules apply:

1 AND 1 = 1
1 AND 0 = 0
0 AND 1 = 0
0 AND 0 = 0

Example:

```
MOV AL, 00000101b
TEST AL, 1      ; ZF = 0.
TEST AL, 10b   ; ZF = 1.
RET
```

C	Z	S	O	P
0	r	r	0	r

Mantıksal işlemlere ait örnek

- Aşağıda kodu yazılmış assemble programın sonucunda AL register içeriği ne olur?

```
MOV AL, 10  
MOV BL, 8  
OR AL, BL
```

$(10)_d = (1010)_b$

$(8)_d = (1000)_b$

$(1010)_b \text{ OR } (1000)_b = (1010)_b = (10)_d$

- Aşağıda kodu yazılmış assemble programın sonucunda AL register içeriği ne olur?

```
MOV AL, 12  
NOT AL
```

$(12)_d = (1100)_b$

$\text{NOT}(1100)_b = (0011)_b = (3)_d$

- Aşağıda kodu yazılmış assemble programın sonucunda AL register içeriği ne olur?

```
MOV AL, 12  
MOV BL, 9  
AND AL, BL
```

$(12)_d = (1100)_b$

$(9)_d = (1001)_b$

$(1100)_b \text{ AND } (1001)_b = (1000)_b = (8)_d$

Shift and Rotate

(Çarpma ve Bölme)

Shift and Rotate Instructions

- Logical vs Arithmetic Shifts
 - SHL Instruction
 - SHR Instruction
 - SAL and SAR Instructions
 - ROL Instruction
 - ROR Instruction
 - RCL and RCR Instructions
 - SHLD/SHRD Instructions

Shift instruction

□ **TABLE 10-5**
Typical Shift Instructions

Name	Mnemonic	Diagram
Logical shift right	SHR	0 — [-----] → C
Logical shift left	SHL	C ← [-----] — 0
Arithmetic shift right	SHRA	[-----] → C ↑
Arithmetic shift left	SHLA	C ← [-----] — 0
Rotate right	ROR	[-----] → C ↑
Rotate left	ROL	C ← [-----] ← C
Rotate right with carry	RORC	[-----] → C ↑
Rotate left with carry	ROLC	C ← [-----] ← C

ROL: Rotate operand1 left

- **ROL Operand1, Operand2**
- Algorithm: Tüm bitleri sola kaydırılır, giden bit CF'ye ayarlanır ve aynı bit en sağdaki konuma eklenir.
- Not: Operand1'i sola döndürülür. Döndürme sayısı Operand2 tarafından belirlenir.

Operand:

- memory, immediate
- REG, immediate
- memory, CL
- REG, CL

Example:

```
MOV AL, 1Ch ; AL = 00011100b
```

```
ROL AL, 1 ; AL = 00111000b, CF=0.
```

```
RET
```

```
org 100h
```

```
MOV AL, 80h
```

```
ROL AL,1
```

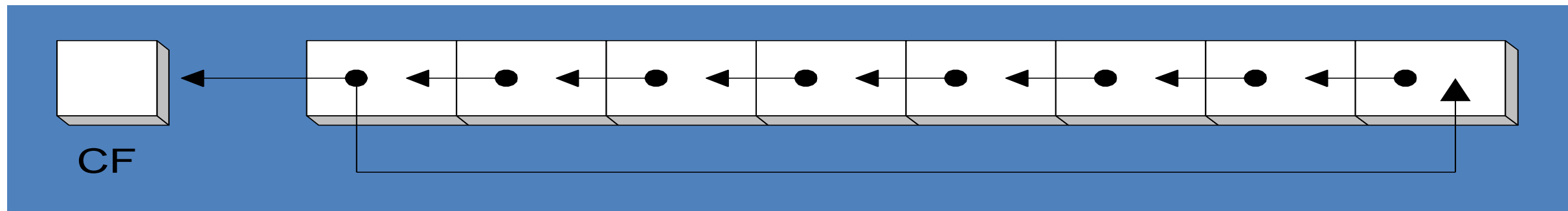
```
MOV BL, 11110000b
```

```
ROL BL,1
```

```
RET
```

ROL instruction

- ROL (rotate) shifts each bit to the left
- The highest bit is copied into both the Carry flag and into the lowest bit
- No bits are lost

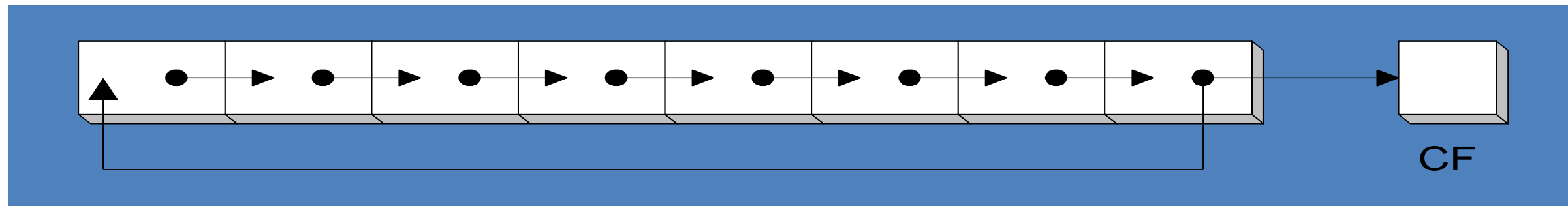


```
mov al,11110000b  
rol al,1; AL = 11100001b
```

```
mov dl,3Fh  
rol dl,4; DL = F3h
```

ROR instruction

- ROR (rotate right) shifts each bit to the right
- The lowest bit is copied into both the Carry flag and into the highest bit
- No bits are lost



```
mov al,11110000b
ror al,1          ; AL = 01111000b

mov dl,3Fh
ror dl,4         ; DL = F3h
```

ROR: Rotate operand1 right

- **ROR Operand1, Operand2**
- Algorithm: Tüm bitleri sağa kaydırılır, giden bit CF'ye ayarlanır ve aynı bit en soldaki konuma eklenir.
- Not: Operand1'i sağa döndürülür. Döndürme sayısı Operand2 tarafından belirlenir.

Operand1, Operand2:

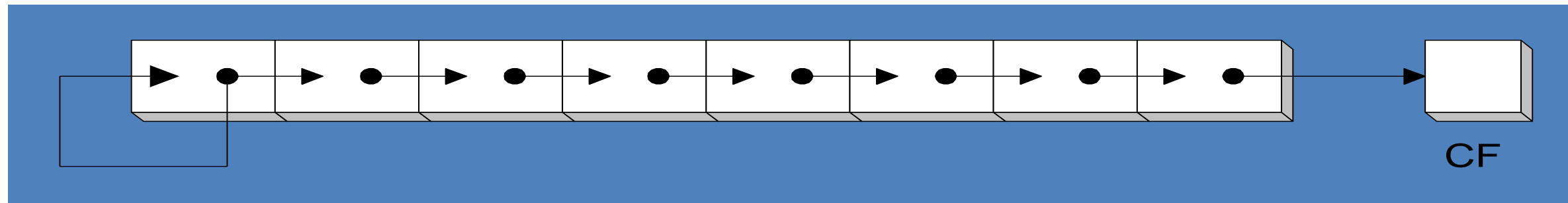
- memory, immediate
- REG, immediate
- memory, CL
- REG, CL

• **Example:**

```
MOV AL, 1Ch ; AL = 00011100b
ROR AL, 1 ; AL = 00001110b, CF=0.
RET
```

SAL and SAR instructions

- SAL (shift arithmetic left) is identical to SHL.
- SAR (shift arithmetic right) performs a right arithmetic shift on the destination operand.



An arithmetic shift preserves the number's sign.

```
mov dl,-80
sar dl,1; DL = -40
sar dl,2; DL = -10
```

SAL: Shift Arithmetic operand1 Left.

- **SAL Operand1, Operand2**
- Algorithm: Tüm bitleri sola kaydırın, giden bit CF'ye ayarlanır. Sıfır biti en sağdaki konuma eklenir.
- Not: Aritmetik olarak Operand1 Sola kaydırılır. Kaydırma sayısı Operand2 tarafından belirlenir

Operand1, Operand2:

- memory, immediate
- REG, immediate
- memory, CL
- REG, CL

• **Example:**

```
MOV AL, 0E0h ; AL = 11100000b  
SAL AL, 1 ; AL = 11000000b, CF=1.  
RET
```

SAR: Shift Arithmetic operand1 Right

- **SAR Operand1, Operand2**
- Algorithm: Tüm bitleri sağa kaydırılır, giden bit CF'ye ayarlanır. Sıfır biti en soldaki konuma eklenir.
- Not: Aritmetik olarak Operand1 sağa kaydırılır. Kaydırma sayısı Operand2 tarafından belirlenir

Operand1, Operand2

- memory, immediate
- REG, immediate
- memory, CL
- REG, CL

- **Example:**

```
MOV AL, 0E0h ; AL = 11100000b
```

```
SAR AL, 1 ; AL = 11110000b, CF=0.
```

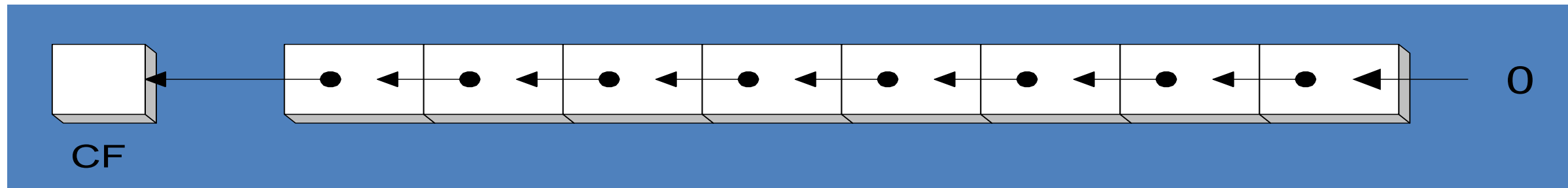
```
MOV BL, 4Ch ; BL = 01001100b
```

```
SAR BL, 1 ; BL = 00100110b, CF=0.
```

```
RET
```

SHL instruction

- The SHL (shift left) instruction performs a logical left shift on the destination operand, filling the lowest bit with 0.



- Operand types: SHL
destination, count

SHL *reg, imm8*

SHL *mem, imm8*

SHL *reg, CL*

SHL *mem, CL*

Örnek:

MOV AL, 8

SHL AL, 1

(8)_d=(0000 1000)_b

SHL (0000 1000)_b= (0001 0000)_b=(16)_d

SHL: Shift operand1 Left.

- **SHL Operand1, Operand2**
- Algorithm: Shift all bits left, the bit that goes off is set to CF. Zero bit is inserted to the right-most position.
- Not: Shift operand1 Left. The number of shifts is set by operand2.

Operand1, Operand2:

- memory, immediate
- REG, immediate
- memory, CL
- REG, CL

• **Example:**

```
MOV AL, 11100000b
```

```
SHL AL, 1 ; AL = 11000000b, CF=1.
```

```
RET
```

SHR: Shift operand1 Right.

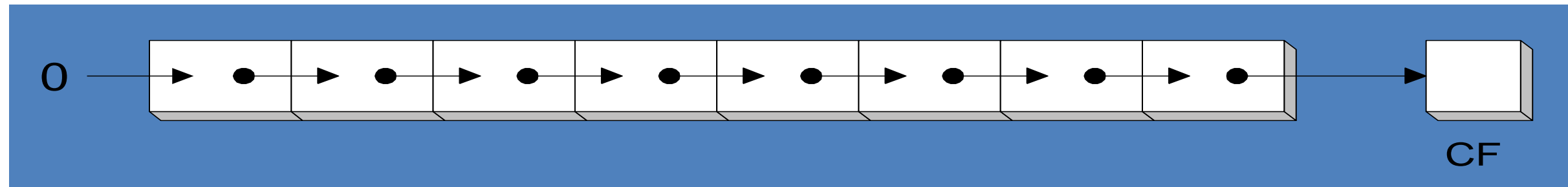
- **SHR Operand1, Operand2**
- Algorithm: Shift all bits right, the bit that goes off is set to CF. Zero bit is inserted to the left-most position.
- Not: Shift operand1 Right. The number of shifts is set by operand2.

- **Operand1, Operand:**
- memory, immediate
- REG, immediate
- memory, CL
- REG, CL

- **Example:**
MOV AL, 00000111b
SHR AL, 1 ; AL = 00000011b, CF=1.
RET

SHR instruction

- The SHR (shift right) instruction performs a logical right shift on the destination operand. The highest bit position is filled with a zero.



Shifting right n bits divides the operand by 2^n

```
mov dl, 80
shr dl, 1; DL = 40
shr dl, 2; DL = 10
```

Örnek:

```
MOV AL, 8
SHR AL, 1
```

$(8)d = (0000\ 1000)b$

$SHR\ (0000\ 1000)b = (0000\ 0100)b = (4)d$

Fast multiplication

Shifting left 1 bit multiplies a number by 2

```
mov dl,5  
shl dl,1
```

Before: `0 0 0 0 0 1 0 1` = 5
After: `0 0 0 0 1 0 1 0` = 10

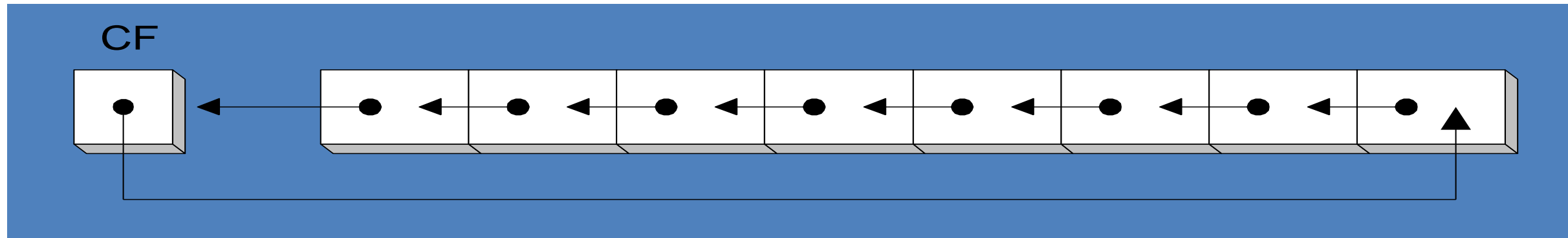
Shifting left n bits multiplies the operand by 2^n

For example, $5 * 2^2 = 20$

```
mov dl,5  
shl dl,2 ; DL = 20
```

RCL instruction

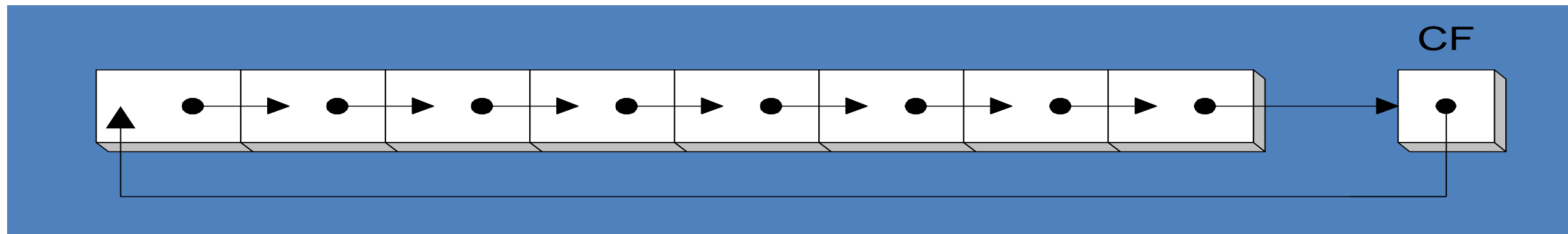
- RCL (rotate carry left) shifts each bit to the left
- Copies the Carry flag to the least significant bit
- Copies the most significant bit to the Carry flag



```
clc          ; CF = 0
mov bl, 88h  ; CF, BL = 0 10001000b
rcl bl, 1    ; CF, BL = 1 00010000b
rcl bl, 1    ; CF, BL = 0 00100001b
```

RCR instruction

- RCR (rotate carry right) shifts each bit to the right
- Copies the Carry flag to the most significant bit
- Copies the least significant bit to the Carry flag



```
stc          ; CF = 1
mov ah,10h   ; CF,AH = 00010000 1
rcr ah,1     ; CF,AH = 10001000 0
```

$$\begin{array}{r} 25 \\ \underline{24} \\ 100 \end{array}$$

$$\begin{array}{r} 100 \\ \underline{100} \\ 1100d \end{array}$$

$$2^{10} \rightarrow \begin{array}{r} 1024 \\ \underline{76} \\ 64 \\ \underline{12} \\ 8 \\ \underline{4} \\ 4 \\ \underline{4} \\ 0 \end{array}$$

? binary.

$$2^9 + 2^6 + 2^3 + 2^2$$

$$\begin{array}{r} 10 \ 9 \ 8 \ 7 \ 6 \ 5 \ 4 \ 3 \ 2 \ 1 \ 0 \\ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \end{array}$$

$$\begin{array}{r} 44ch \\ \underline{\quad} \\ (44c)h \end{array}$$

Örnek:

27h ile CCh değerinin

- a) AND
- b) OR
- c) NOT.

$$\begin{array}{r} 0010 \ 0111 \\ 1100 \ 1100 \\ \hline \text{AND} \ 0000 \ 0100 \\ \hline (04)h \end{array}$$

$$\begin{array}{r} 0010 \ 0111 \\ 1100 \ 1100 \\ \hline \text{OR: } (1110 \ 1111)_b \\ \hline (EF)h \end{array}$$

$$(0010 \ 0111) \xrightarrow{\text{NOT}} (1101 \ 1000) \\ (08)h$$

Comparison Instructions Program Flow Control (CMP)

CMP Operand1, Opernad2

Sadece bayraklar etkilenir. Bir sonraki komut bayrak karşılaştırma işlemidir.

CMP

REG, memory
memory, REG
REG, REG
memory,
immediate
REG,
immediate

operand1 - operand2

result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.

Example:

MOV AL, 5

MOV BL, 5

CMP AL, BL ; AL = 5, ZF = 1 (so equal!)

RET

C	Z	S	O	P	A
r	r	r	r	r	r

Short Conditional Jumps

Instruction	Description	Flags
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if sign	SF = 1
JNS	Jump if not sign	SF = 0
JE	Jump if equal	ZF = 1
JZ	Jump if zero	
JNE	Jump if not equal	ZF = 0
JNZ	Jump if not zero	
JB	Jump if below	CF = 1
JNAE	Jump if not above or equal	
JC	Jump if carry	
JNB	Jump if not below	CF = 0
JAE	Jump if above or equal	
JNC	Jump if not carry	
JBE	Jump if below or equal	CF = 1 or ZF = 1
JNA	Jump if not above	

Short Conditional Jumps

Instruction	Description	Flags
JA JNBE	Jump if above Jump if not below or equal	CF = 0 and ZF = 0
JL JNGE	Jump if less Jump if not greater or equal	SF <> OF
JGE JNL	Jump if greater or equal Jump if not less	SF = OF
JLE JNG	Jump if less or equal Jump if not greater	ZF = 1 or SF <> OF
JG JNLE	Jump if greater Jump if not less or equal	ZF = 0 and SF = OF
JP JPE	Jump if parity Jump if parity even	PF = 1
JNP JPO	Jump if not parity Jump if parity odd	PF = 0
JCXZ JECXZ	Jump if CX register is 0 Jump if ECX register is 0	CX = 0 ECX = 0

CMP: Compare

- CMP Operand1, Operand2
- Algorithm: operand1 - operand2
- Not: result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result. Bu komuttan sonra JMP ya da j.. (JE, JNE, ...) Komutlarından biri gelir.

- **Operand1, Operand1:**
- REG, memory
- memory, REG
- REG, REG
- memory, immediate
- REG, immediate

- **Example:**
Org 100h
MOV AL, 5
MOV BL, 5
CMP AL, BL ; AL = 5, ZF = 1 (so equal!)
RET

Örnek: JE – JG

Ekranaya yazı yazma

Kalıbı

```
ORG 100h
MOV AL, 5
MOV BL, 6
CMP AL, BL
JE label1
call m1
JMP exit
label1: call m2
exit:
RET
```

```
m1 PROC
mov dx, offset msg1
mov ah, 9
int 21h
ret
m1 ENDP
```

```
m2 PROC
mov dx, offset msg2
mov ah, 9
int 21h
ret
m2 ENDP
msg1 db "AL is not equal to
BL $"
msg2 db "AL is equal to BL
$"
END
```

Jump instructions for signed numbers

- JE, JZ
- JNE, JNZ
- JA, JNBE
- JB, JNAE, JC
- JAE, JNB, JNC
- JBE, JNA

Jump instructions for unsigned numbers

- JE, JZ
- JNE, JNZ
- JG, JNLE
- JL, JNGE
- JGE, JNL
- JLE, JNG

Örnek: Dizi Toplamını Bulan Yazılım

```
ORG 100h
```

```
mov si,0h
```

```
mov cx,0h
```

```
atla:  cmp si,12
```

```
je stop
```

```
add cl, dizi[si]
```

```
inc si
```

```
jmp atla
```

```
stop:  ret
```

```
dizi db 1h, 2h, 3h, 4h, 5h,4h,7h,8h,4h,0ah,0bh, 0ch, 0dh
```

Set

STC, CLC

- STC No operands
- Set Carry flag.
- Algorithm:
- $CF = 1$

- CLC No operands
- Clear Carry flag.
- Algorithm:
- $CF = 0$

Kesme

Kesme

- Kesme geldiğine
- AX, BX, CX, DX, SI, DI, SP, BP, CS, SS, DS, ES, IP, Flags register'ların içeriklere SS belleğe kaydedilir.
- Kesme isteğine ilişkin alt programını çalıştırır.
- İşlev tamamlandığında, belleğe kayıt edilen değerler yeniden register'lara aktarılır.
- CPU kaldığı yerden çalışmasına devam eder.

Loop

Loop: CX=0 olana dek çevrimsel işlemleri yapar.

- **LOOP label;** Decrease CX, jump to label if CX not zero.

- **Algorithm:**

CX = CX - 1

if CX <> 0 then

jump

else

no jump, continue

- Example:

```
ORG 100h
```

```
MOV CX, 5
```

```
label1:
```

```
    mov dx, offset msg
```

```
    mov ah, 9
```

```
    int 21h
```

```
    LOOP label1
```

```
RET
```

```
msg db "Esenyurt University $"
```

```
END
```

1'den N'e kadar sayıların toplamını bulan assembly program, $TOP = N * (N + 1) / 2$

```
ORG 100h
    mov cx,20
mov ax,0

dongu: add ax,cx
        loop dongu
ret
```

Örnek: Dizide benzer sayının bulunması

```
org 100h
```

```
    lea bx, vec1  
    mov cl,7  
    mov dl,39h
```

```
ali: mov al,[bx]  
     cmp dl,al  
     je TIK  
     inc bx  
     loop ali
```

```
TIK:  ret
```

```
vec1 db 23h, 26h, 22h, 35h, 39h, 0ach,0bch
```

Looping using zero flag

- The zero flag is set (ZF=1), when the counter becomes zero (CX=0)
- *Example:* add 5 bytes of data

```
MOV    CX,05        ;CX holds the loop count
MOV    BX,0200H     ;BX holds the offset data address
MOV    AL,00        ;initialize AL
ADD_LP: ADD  AL,[BX] ;add the next byte to AL
INC    BX           ;increment the data pointer
DEC    CX           ;decrement the loop counter
CMP    CX,0
JNZ    ADD_LP       ;jump to next iteration if counter
                    ;not zero
```

The loop instruction

instruction	operation and jump condition	opposite instruction
LOOP	decrease CX, jump to label if CX not zero.	DEC CX and JCXZ
LOOPE	decrease cx, jump to label if cx not zero and equal (zf = 1).	LOOPNE
LOOPNE	decrease cx, jump to label if cx not zero and not equal (zf = 0).	LOOPE
LOOPNZ	decrease cx, jump to label if cx not zero and zf = 0.	LOOPZ
LOOPZ	decrease cx, jump to label if cx not zero and zf = 1.	LOOPNZ
JCXZ	jump to label if cx is zero.	OR CX, CX and JNZ

Example

```
org 100h; Finds the array
    lea bx, vec1
    mov cx, 4
    mov al, 0
sum:
    add al, [bx]
    inc bx
    loop sum
    ret
vec1 db 3, 6, 1, 5
```

```
ORG 100h ; TOP= N*(N+1)/2
mov cx,20
mov ax,0
dongu: add ax,cx
loop dongu
ret
```

FOR LOOP

Example:

For 80 times DO

Display '*'

END_IF

Assembly :

```
MOV CX, 80
```

```
MOV AH, 2
```

```
MOV DL, '*'
```

```
Next: INT 21H
```

```
Loop Next
```

While Loop

Example:

```
Initialize count to 0  
Read a character  
While character <> Carriage Return DO  
Count = Count + 1  
Read a character  
END_While
```

Assembly:

```
MOV DX, 0  
MOV AH, 1  
INT 21H  
While_: CMP AL, 0DH  
JE End_While  
INC DX  
INT 21H  
JMP While_  
End_While:
```

Repeat Loop

Example:

Repeat

Read a character

Until character is blank

Assembly:

```
MOV AH, 1
```

Repeat:

```
INT 21H
```

; until

```
CMP AL, ''
```

```
JNE Repeat
```

Application of Loope

- **Example: Search for a number in a Table**

Table DB 1,2,3,4,5,6,7,8,9

XOR SI, SI

MOV CX, 9

Next: INC SI

CMP Table[SI-1], 7

Loopne Next

Example

org 100h ; Is there same number in the given array?

```
    lea bx, vec1
    mov cl,7
    mov dl,39h
```

```
ali: mov al,[bx]
     cmp dl,al
     je TIK
     inc bx
     loop ali
```

```
TIK:  ret
```

```
vec1 db 23h, 26h, 22h, 35h, 39h, 0ach,0bch
```

org 100h; Is there number in the given array?

```
    lea bx, vec1
    mov cl,7
    mov dl,39h
```

```
ali: mov al,[bx]
     cmp dl,al
     je TIK
     inc bx
     loop ali
```

```
TIK:  ret
```

```
vec1 db 23h, 26h, 22h, 35h, 39h, 0ach,0bch
```

Example

```
ORG 100h; teletype function number.
```

```
LEA SI, msg ; DS: aaaa0 + SI
```

```
next_char:
```

```
    CMP [SI], 0
```

```
    JE stop
```

```
    MOV AL, [SI] ; next get ASCII char.
```

```
    MOV AH, 0Eh ; teletype function number.
```

```
    INT 10h ; using interrupt to print a char in AL.
```

```
    inc SI
```

```
    JMP next_char
```

```
stop:    ENDP
```

```
msg DB "Hello World!", 0
```

Examples

```
org 100h; Find the big element in the array
mov cl,1
mov ch,N ; dizi uzunlugu
lea bx,vec ; dizinin baslangic adresi
mov dh,[bx] ; max=vec[1]
    L3: inc cl
        inc bx
        mov dl,[bx]
        cmp dh,dl ; max > vec[i]
        ja L1
        mov dh,dl
        L1:
        cmp ch,cl
        jne L3
endp
N db 8
vec db 43h, 1h, 3h,75h, 9h,7h,2h,66h
```

```
ORG 100h; Average program
mov cx,9
mov si,cx
dec si
mov ax,0
mov dx,0
git:  mov dl, a[si]
      add ax,dx
      cmp si,0
      je cik
      dec si
      jmp git
cik:  div cl
      ret
a db 5,6,3,2,7,8,5,4,5
```

String Instructions

String Instructions

- **Five categories**
 - MOVS, MOVSB, MOVSW
 - CMPS, CMPSB, CMPSW
 - SCAS, SCASB, SCASW
 - LODS, LODSB, LODSW
 - STOS, STOSB, STOSW
- **Source is always in DS:[SI]**
- **Destination is always in ES:[DI]**
- **If DF=0, SI and DI are incremented**
- **If DF=1, SI and DI are decremented**
- **To clear direction flag: CLD**
- **To set direction flag: STD**

String Instructions – Cont.

- **MOVSB**
 - $ES:[DI] \leftarrow DS:[SI]$
 - $DI \leftarrow DI+1; SI \leftarrow SI+1$ (if DF=0)
 - $DI \leftarrow DI-1; SI \leftarrow SI-1$ (if DF=1)
- **MOVSW**
 - $ES:[DI+1:DI] \leftarrow DS:[SI+1:SI]$
 - $DI \leftarrow DI+2; SI \leftarrow SI+2$ (if DF=0)
 - $DI \leftarrow DI-2; SI \leftarrow SI-2$ (if DF=1)
- **MOVS destination, source**
 - Replaced by either MOVSB or MOVSW depending on operands size

String Instructions – Cont.

■ CMPSB

- DS:[SI] - ES:[DI]
- $DI \leftarrow DI+1$; $SI \leftarrow SI+1$ (if DF=0)
- $DI \leftarrow DI-1$; $SI \leftarrow SI-1$ (if DF=1)

■ CMPSW

- DS:[SI+1:SI] - ES:[DI+1:DI]
- $DI \leftarrow DI+2$; $SI \leftarrow SI+2$ (if DF=0)
- $DI \leftarrow DI-2$; $SI \leftarrow SI-2$ (if DF=1)

■ CMPS destination, source

- Replaced by either CMPSB or CMPSW depending on operands size

String Instructions – Cont.

■ SCASB

- AL - ES:[DI]
- $DI \leftarrow DI+1$; (if DF=0)
- $DI \leftarrow DI-1$ (if DF=1)

■ SCASW

- AX - ES:[DI+1:DI]
- $DI \leftarrow DI+2$; (if DF=0)
- $DI \leftarrow DI-2$; (if DF=1)

■ SCAS destination

- Replaced by either SCASB or SCASW depending on operands size

String Instructions – Cont.

■ LODSB

- $AL \leftarrow DS:[SI]$
- $SI \leftarrow SI+1$; (if $DF=0$)
- $SI \leftarrow SI-1$ (if $DF=1$)

■ LODSW

- $AX \leftarrow DS:[SI+1:SI]$
- $SI \leftarrow SI+2$; (if $DF=0$)
- $SI \leftarrow SI-2$; (if $DF=1$)

■ LODS destination

- Replaced by either LODSB or LODSW depending on operands size

String Instructions – Cont.

■ STOSB

- $ES:[DI] \leftarrow AL$
- $DI \leftarrow DI+1$; (if $DF=0$)
- $DI \leftarrow DI-1$ (if $DF=1$)

■ STOSW

- $ES:[DI+1:DI] \leftarrow AX$
- $DI \leftarrow DI+2$; (if $DF=0$)
- $DI \leftarrow DI-2$ (if $DF=1$)

■ STOS destination

- Replaced by either STOSB or STOSW depending on operands size

Copying a String to another

.DATA

```
String1 DB "Hello"  
String2 DB 5 dup(?)
```

.CODE

```
MOV AX, @DATA  
MOV DS, AX  
MOV ES, AX  
CLD  
MOV CX, 5  
LEA SI, String1  
LEA DI, String2  
REP MOVSB
```

Copying a String to another in Reverse Order

.DATA

```
String1 DB "Hello"  
String2 DB 5 dup(?)
```

.CODE

```
MOV AX, @DATA  
MOV DS, AX  
MOV ES, AX  
STD  
MOV CX, 5  
LEA SI, String1+4  
LEA DI, String2  
Next: MOVSB  
ADD DI, 2  
LOOP Next
```

String Instructions

Simple Byte Transfers

LODSB	Load String Byte	AL ← [SI] SI ← SI+1
STOSB	Store String Byte	[DI] ← AL DI ← DI+1

Simple Word Transfers

LODSW	Load String Word	AL ← [SI] AH ← [SI+1] SI ← SI+2
STOSW	Store String Word	[DI] ← AL [DI+1] ← AH DI ← DI+2

String Example

```
org 0x100
section .data
    s_1 db "0123456789",0
    v_1 times 11 db 0
section .text
    MOV SI, s_1      ; SI ← pointer to s_1
    MOV DI, v_1      ; DI ← pointer to v_1
L1:  LODSB          ; AL ← [SI]
      ; SI ← SI + 1

    CMP AL, 0
    JZ end
    SUB AL, 30      ; AL ← AL - 30 = numerical value of ASCII char
    STOSB          ; [DI] ← AL
      ; DI ← DI + 1

    JMP L1          ; loop
end: STOSB          ; store \null
mov ax,4C00h
int 21h
```

Processor Control Instructions

Processor Control Instructions

Mnemonics

STC

Explanation

Set $CF \leftarrow 1$

CLC

Clear $CF \leftarrow 0$

CMC

Complement carry $CF \leftarrow CF'$

STD

Set direction flag $DF \leftarrow 1$

CLD

Clear direction flag $DF \leftarrow 0$

STI

Set interrupt enable flag $IF \leftarrow 1$

CLI

Clear interrupt enable flag $IF \leftarrow 0$

NOP

No operation

HLT

Halt after interrupt is set

WAIT

Wait for TEST pin active

ESC opcode mem/ reg

Used to pass instruction to a coprocessor which shares the address and data bus with the 8086

LOCK

Lock bus during next instruction

ALT Programlama

Control Transfer Instructions

- Transfer the control to a specific destination or target instruction
- Do not affect flags

Mnemonics

CALL reg/ mem/ disp16

RET

JMP reg/ mem/ disp8/ disp16

Explanation

Call subroutine

Return from subroutine

Unconditional jump

Procedures

```
ORG 100h
MOV AL, 1
MOV BL, 2
CALL m2
CALL m2
CALL m2
CALL m2
RET ; return to operating system.
```

```
m2 PROC
MUL BL ; AX = AL * BL.
RET ; return to caller.
m2 ENDP
END
```

Örnek: Call

Örnek-1:

```
ORG 100h  
CALL m1  
MOV AX, 2  
ADD AX, BX  
RET ; return to operating system.
```

m1 PROC

```
MOV BX, 5  
RET ; return to caller.
```

m1 ENDP

END

Örnek-2:

```
ORG 100h  
MOV AL, 1  
MOV BL, 2  
CALL m2  
CALL m2  
CALL m2  
CALL m2  
RET ; return to operating system.
```

m2 PROC

```
MUL BL ; AX = AL * BL.  
RET ; return to caller.
```

m2 ENDP

END

Örnek: Call and Loop

```
ORG 100h
MOV AL, 1
MOV BL, 2
Mov CX,4
don:
CALL m2

LOOP don
RET ; return to operating system.
m2 PROC
MUL BL ; AX = AL * BL.
RET ; return to caller.
m2 ENDP
END
```

Applications

Örnek: Ekran Mesaj Yazma

```
org 100h                                ; write msg3
; write msg1                             mov dx, offset a3
mov dx, offset a1                       mov ah, 9
mov ah, 9                                int 21h
int 21h                                  ret

; write msg2                             a1 db " Cahit Karakus", 0Dh,0Ah, "$"
mov dx, offset a2                       a2 db " Computer Engineering" , 0Dh,0Ah,
mov ah, 9                                "$"
int 21h                                  a3 db " Esenyurt University" , 0Dh,0Ah, "$"
```

Örnek: Ekrandan Karakter Okunması

```
org 100h
    ; write msg1
    mov dx, offset a1
    mov ah, 9
    int 21h
; read character in al:
    mov ah, 1
    int 21h
    ret
    ; yazilan karakter AL icerisindedir.

a1 db " Bir Tusa Basiniz", 0Dh,0Ah, "$"
```

Örnek: Ekrandan 0 ile 99 arasında sayı girilmesi

```
org 100h
; write msg1
    mov dx, offset a1
    mov ah, 9
    int 21h
; read character in al:
git1:  mov ah, 1
        int 21h
        cmp al, 30h
        jb git1
        cmp al, 39h
        ja git1
        mov bh, al
        ; write msg1
        mov dx, offset a1
        mov ah, 9
        int 21h
```

```
; read character in al:
git2:  mov ah, 1
        int 21h
        cmp al, 30h
        jb git2
        cmp al, 39h
        ja git2
                mov bl, al
                sub bh, 30h
        sub bl, 30h
        mov al, bh
        mov cl, 10
        mul cl
        add al, bl
        ret
```

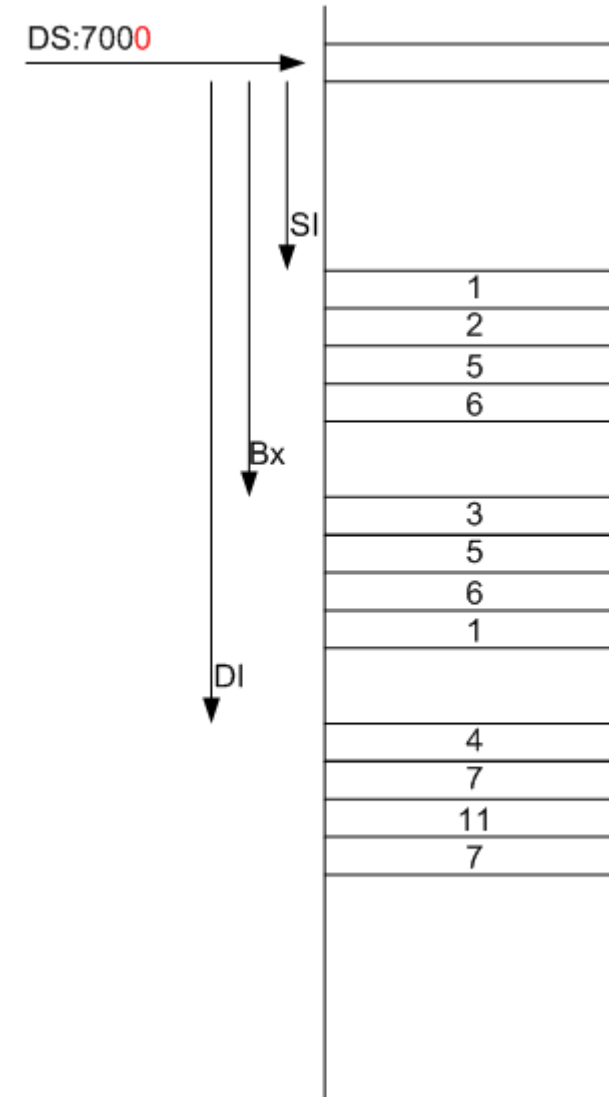
a1 db " 0 ile 99 arasinda bir sayi giriniz: \$"

Example: the sum of a vector with; another vector and saves result in third vector.

```
org 100h
jmp start
vec1 db 1, 2, 5, 6
vec2 db 3, 5, 6, 1
vec3 db ?, ?, ?, ?

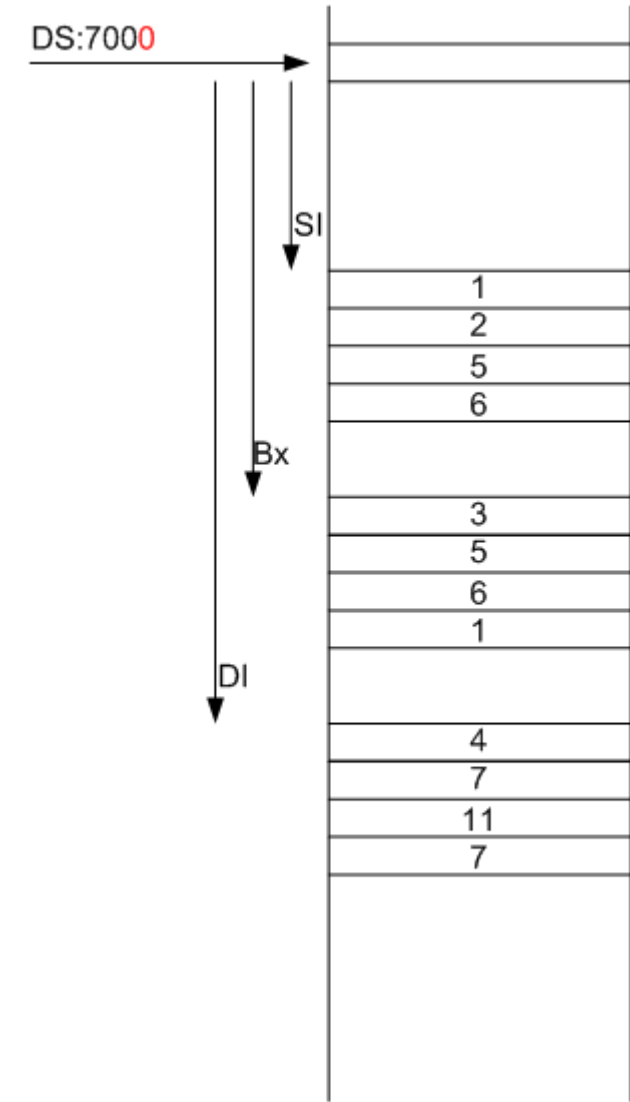
start:  lea si, vec1
        lea bx, vec2
        lea di, vec3
        mov cx, 4

sum:    mov al, [si]
        add al, [bx]
        mov [di], al
        inc si
        inc bx
        inc di
        loop sum
        ret
```



Example: add al, [bx]

- 1- AA: [indis] ifadesinde AA: segment register tanımlar: CS, DS, SS, ES. 16 bittir: (YYYY)h. Eğer AA yok ise default olarak DS tanımlar. 20 bit fiziksel adrese dönüşür: (YYYY0). İndis ,16 bitlik register tanımlar: BX, SI, DI, BP, SP.
- 2- datanın yeri: (YYYY0)+indis
- 3- Al olmasından dolayı 1 byte veri okunacak ve AL registerine transfer edilecektir.



Example: counts the number of characters of a zero terminated string.

```
name "counter"
org 100h
jmp start
    str1 db " Cahit Karakus – Computer Engineering – Esenyut University ", 0
start:    lea bx, str1 ; load address of string.
          mov ax, 0 ; reset counter.
compare: cmp [bx], 0 ; is it end of string?
          je done ; if zero, then it's the end.
          inc ax ; count char.
          inc bx ; next memory position in string.
          jmp compare
done:    ret
```

Example: 1 den N'e kadar sayıların toplanması

```
name "calc-sum"
```

```
org 100h ; directive make tiny com file.
```

```
mov cl, 15 ; number of elements:
```

```
mov al, 0 ; al will store the sum:
```

```
next: add al, cl ; sum elements:
```

```
loop next ; loop until cx=0:
```

```
mov bh, al ; store result in bl:
```

```
ret
```

Example: Dizideki sayıların toplanması ve ortalamasının bulunması

```
ORG 100h  
mov cl,N  
lea dx,a  
mov bx, offset a  
mov al,0  
mov si,0
```

don:

```
add al,a[si]  
inc si  
loop don
```

```
div N  
ret
```

```
a db 5,6,3,2,7,8,5,4,5
```

```
N db 9
```

Example: 0 dan 99 kadar iki basamaklı sayı okunması

```
org 100h
; write msg1
    mov dx, offset a1
    mov ah, 9
    int 21h
; read character in al:
git1: mov ah, 1
    int 21h
    cmp al, 30h
    jb git1
    cmp al, 39h
    ja git1
    mov bl, al
```

```
git2: mov ah, 1
    int 21h
    cmp al, 30h
    jb git2
    cmp al, 39h
    ja git2

    sub al,30h
    sub bl,30h
    xchg al, bl
    mov cl,10
    mul cl
    add al,bl
    ret
```

a1 db " 0 ile 9 arasinda bir sayi giriniz: \$"

Example: Mantıksal İşlemler

Org 100h

MOV AL, 01010101b

MOV CL, AL

MOV BL, 10101010b

AND AL, BL

MOV CH, BL

OR CL, CH

NOT AL

NOT BL

RET

Example: Dizide aynı olan sayıyı bulma

```
org 100h
```

```
    lea bx, vec1
```

```
    mov cl,7
```

```
    mov dl,39h
```

```
don1: mov al,[bx]
```

```
    cmp dl,al
```

```
    je git
```

```
    inc bx
```

```
    loop don1
```

```
git:  ret
```

```
vec1 db 23h, 26h, 22h, 35h, 39h, 0ach,0bch
```

Example: Dizide büyük elamanı bulma

```
org 100h
mov cl,N    ; dizi uzunlugu
dec cl
lea bx, vec ; dizinin baslangic adresi
mov al, [bx] ; max=vec[1]

L3:  inc bx
     mov ah, [bx]
     cmp al, ah ; max > vec[i]
     jae L1
     mov al, ah

L1:  loop L3
     ret

N    db 8
vec  db 43h, 1h, 3h,75h, 9h,7h,0AAh,66h
end
```

Arrays

- Arrays can be seen as chains of variables. A text string is an
- example of a byte array, each character is presented as an ASCII
- code value (0..255).
- Here are some array definition examples:
- a DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h
- b DB 'Hello', 0
- b is an exact copy of the a array, when compiler sees a string inside
- quotes it automatically converts it to set of bytes. This chart shows
- a part of the memory where these arrays are declared:

Usage Notes

- A lot of slides are adopted from the presentations and documents published on internet by experts who know the subject very well.
- I would like to thank who prepared slides and documents.
- Also, these slides are made publicly available on the web for anyone to use
- If you choose to use them, I ask that you alert me of any mistakes which were made and allow me the option of incorporating such changes (with an acknowledgment) in my set of slides.

Sincerely,

Dr. Cahit Karakuş

cahitkarakus@esenyurt.edu.tr