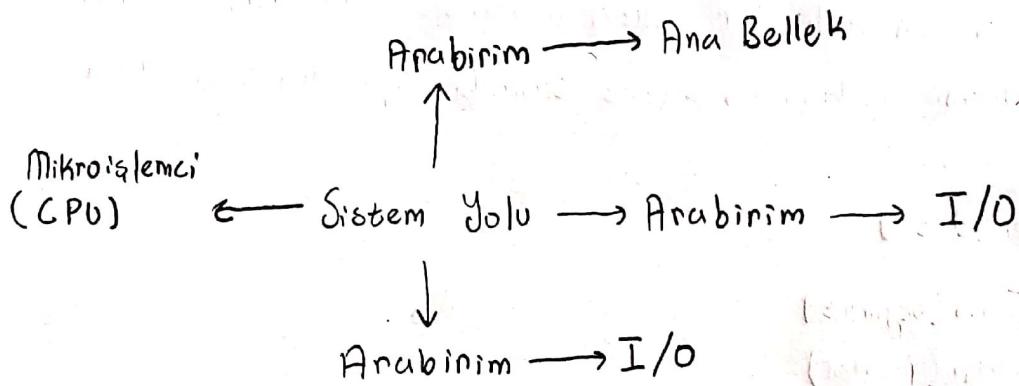


X86 ASM

şafak bilici

Sistem Organizasyonu



Merkezi İşlem Birimi / İşlemci / CPU : Tüm aritmetik-matiksal işlemlerin yapıldığı yer. Komutlar ve üzerinde işlem yapılacak data işlemci tarafından isteninceye kadar bellekte tutulmaktadır. Von Neuman mimarisindeki birimlerin bağlantıları bus ile sağlanır.

- bus → address bus
- data bus
- control bus

Front Side Bus : işlemci ile ana kart arasındaki bağlantıyı sağlayan birimi.

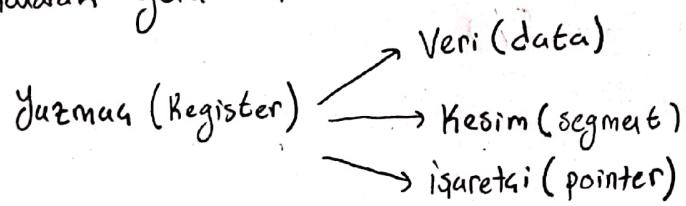
Yaygın Olarak Kullanan Bellek Tipleri → SDRAM

- DDR-SDRAM
- DDR2-SDRAM
- DDR3-SDRAM
- DDR4-SDRAM
- RDRAM

8086 işlemcisinin yapı yapısı

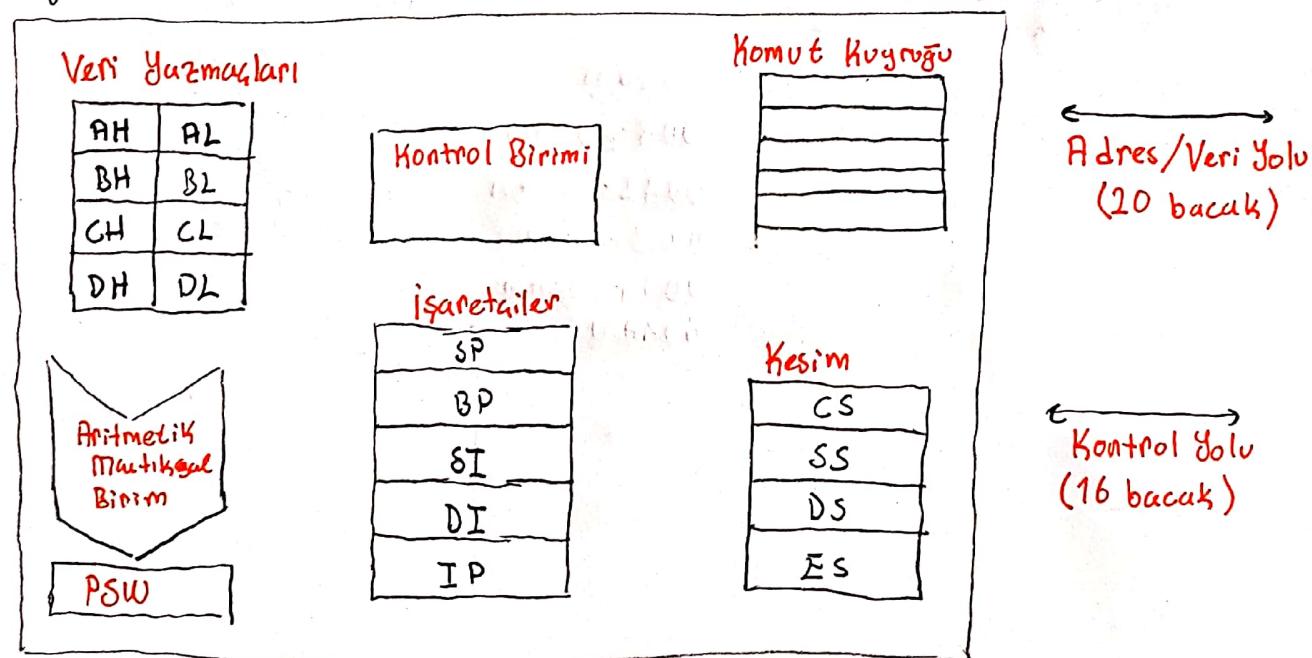
8086 işlemci, işlemleri 3 temel grup altında toplanmış yuzmeleri (register) kullanarak gerçekleştirmektedir.

Yazma, küçük kapasiteli ancak çok hızlı bir bellek birimidir. İşlemci, işlemlerini bu özel nitelikli bellek birimlerini kullanmak üzere tasarlanmış olan komutları kullanarak gerçekleştirir.



İşlemci içinde 6 byte uzunluğundaki Komut Kuyruğunu (instruction queue) yanı sıra aritmetik ve matematiksel işlemleri yapan bir ünite (ALU=Arithmetic-Logic Unit) ve genel işleyişten sorumlu olan Kontrol birimi mevcuttur.

Yapıda işlemlerin sonuçlarına göre farklı durumları ifade etmek amacıyla kullanılan ve bayrak (flag) olarak adlandırılan değerler, PSW (Program Status Word) isimli yazmaya saklanmaktadır.



Kesim (Segment) Yapısı

8086 işlemcisinde 20 adet adres bacağı olmasına rağmen en büyük yazmacı 16 bit Bir tek yazmacı üzerindeki değer ile en fazla 64 KB'lık bir alanı adresleyebilmektedir.

Kesim yazmaları ve Kesim içi işaretçileri ile 8086 işlemcileri belleğin tümünü kullanabilmektedir.

Herhangi bir Kesime erişmek ancak o alanın Kesim yazmacını kullanmakla mümkündür. Bu kesim yazmaları CS, SS, DS, ES olarak adlandırılır. Kesim içi verilere Kesim içi işaretçilerle ulaşılır.

- Kod (Code) Kesimi: Bilgisayarın herhangi bir uygulanmayı galistirabilmesi için derlemiş Kodun bellek üzerinde yerleştirilmiş olması gereklidir. Buna göre galistirılması istenilen komutların bulunduğu bellek alanı Kod Kesimi olarak değerlendirilir.

Kod Kesiminin başlangıç adresi CS yazmacı ile belirlenir.

Bu alan içinde hangi Komutun işleneceğini gösteren işaretçi IP (instruction pointer) dir.

Galistirılması istenen Komutun yerini CS:IP çifti belirler.

- Yığın (Stack) Kesimi: Yığın, son giren ilk çıkar (LIFO-Last In First Out) kurallına göre galışan bir bellek alanı olarak tanımlanabilir. Yığın Kesiminin başlangıç adresi SS yazmacı ile belirlenmektedir. Bu Kesim, işletim sisteme dönüştürülmesi gereklili döngü parametrelerinin saklanması ve ve programın galışı sırasında kullanılan PUSH Komutlarının kullanım sonucu oluşan geçici veri saklama ihtiyacını karşılamak üzere ayrılmıştır.

Bu Kesim içine konulacak verinin nereye yerleştirileceğini veya nerede alınacağını belirleyen işaretçiye SP (stack pointer) adı verilmektedir. Yığın boyut adresten kaack adrese ilerlediği iain PUSH Komutu ile yığınca bir word boyutluğundaki veri yerleştirildiğinde SP yazmacı 2 byte azalacaktır.

Eğer yığın Kesimi içerisinde, yığın içi dizilisi engellemeksiz herhangi bir yerdeki veriye erişilmek isteniyorsa bunun iain BP (base pointer) yazmacı kullanılır.

Veri (Data) Kesimi: Programların üzerinde işlem yapacağı veri, bellekteki:

Veri kesiminde bulunur.

Veri kesiminin başlangıç adresi DS yazmacı ile belirlenmekte, kesim içi adresleme ise; SI, DI ve gerekmesi durumunda BX yazmacalarından yararlanılmaktadır.

EK (Extra) Kesim: Kullanıcının isteyeceği verinin mevcut veri kesiminin (DS) kapasitesinden fazla veya aynı anda iki farklı veri kesimine erişimin gereklili olduğu durumlarda ek kesimler yararlanılmalıdır. Bu başlangıç adresi ES yazmacı ile belirlenmektedir.

Yazmaç (Register)

Yazmaç; işlemci içinde, belirli bir cihaz için tasarlanmış, sınırlı kapasiteli hızlı veri saklama ve/veya devresidir. En hızlı bellek birimidir.

Byte veya word olarak kullanılır.

15	8 7	0
AH	AL	
BH	BL	
CH	CL	
DH	DL	

Veri Grubu

SP
BP
IP

İşaretçi Grubu

SI
DI

Sıra Grubu

CS
DS
SS
ES

Kesim Grubu

Veri Grubu Yazmaçları

Veri grubu yazmaçları 8 bit veya 16 bit olarak kullanılabılırler. Yazmaçların hangi şekilde kullanıldığı ihtiyaca göre değişir. 8 bit işlemlerde AL, AH, BL, BH, CL, CH, DL, DH olarak; 16 bit işlemlerde AX, BX, CX, DX olarak; 32 bit işlemlerde EAX, EBX, ECX, EDX kullanılır.

AX (Accumulator): I/O işlemlerinde; aritmetik, string işlemlerinde kullanılır.

BX (Base Register): Genel amaçlı hesaplamalarda ve dizi şeklindeki veri yapılarına erişmek için indis olarak da kullanılır. → [BX]

CX (Count Register): Gevrim sayılarını belirtmek üzere kullanılır. Kaydırma (shift) ve döndürme (rotate) işlemlerinde tekrar sayısını belirler. → LOOP

DX (Data Register): I/O işlemlerinde; çarpma gibi büyük sayıların oluşabileceği (DX:AX ikilisi olarak), bölme gibi bölüm ve kalan şeklinde iki ayrı sonucun olduğu işlemlerde (AX bölüm, DX kalan değerini saklayacak şekilde) kullanılır. 16 bit tek parça.

İsaret Grubu Yazmaçları

Tanımlı oldukları kesim içindeki görelî konum değerini göstermek amacıyla kullanılır. Kullanılırlar komutlar ile otomatik olarak değer değiştirirler (BP hizası).

SP (Stack Pointer): SP yazmacı ^(stack) yığın üzerinde yapılan işlemlerde, verinin nereye yerleştirileceğini veya nereden alınacağını belirler. (PUSH / POP)

BP (Base Pointer): BP yazmacı yığın dan istenen verilere erişimi kolaylaştırır. İzin Kullanılırlar. Degeri kullanıcı tarafından belirlenir.

IP (Instruction Pointer): IP yazmacı işleminin o anda çalıştıracağı komutun Kod Kesimi içindeki yerini belirlemek için kullanılır.

Sıra Grubu Yazmaları

Sıra grubu yazmaları veri kesiminde kullanılır. işaretin grubundan farklılığı değerleri kullanıcı tarafından belirlenir.

SI (Source Index): Veri kesiminde taşınmış veriye ulaşmak için gerekli konum değeri olarak kullanılmaktadır. [SI]

DI (Destination Index): SI'nın özelliklerinin yanı sıra eğer var ise ek kesim içindeki veriye de erişmek için kullanılmaktadır.

Kesim Grubu Yazmaları

Tanımlı kesimlerin başlangıç adreslerini göstermek üzere kullanılan 16 bitlik yazmalar. Herhangi bir verinin bellek üzerindeki yeni kesim ve gerekli konum değerleri ile belirleniyor.

	Kesim	Göreli Konum
Konut Adresleme	CS	IP
Yığın Adresleme	SS	SP
	SS	BP
Veni Adresleme	DS	SI, DI, BX
	ES	DI, SI, BX

Bayraklar (Flags)

Bayraklar bazı işlemlerin sonucunda değer değiştirerek sonucu yansıtırlar.
1 bit ile ifade edilirler, bu sebeple 0 veya 1 değerlerini alabilirler.

1 → Set

0 → Clear

SF(Carry Flag): Elde/ödönç durumlarında Set.

PF(Parity Flag): İşlem sonucunda even parity (1'lerin sayısı çift) ise Set.

AF(Auxiliary Flag): 8'li işlemlerde düşük çukullu 4'lüde (nibble) yüksek çukullu dörtlüğe elde/ödönç, 16'lı işlemlerde düşük çukullu 8'lüde yüksek çukullu 8'lige elde/ödönç aktarılmasında Set.

ZF(Zero Flag): İşlem sonucunun sıfır çıkması durumunda Set.

SF(Sign Flag): MSB 1 ise Set.

TF(Trap Flag): Debug mode, her satırda kesme.

IF(Interrupt Flag): Maskable interrupt isteklerinin izinli olup olmadığını belirlemek üzere kullanılır.

DF(Direction Flag): String işlemlerinin yönünü belirler.

OF(Overflow Flag): Aritmetik taşıma durumunda Set.

İşlemcinin Komutları Adım Adım Galistirması

- 1- Komutu belirleyen byte'in bellek üzerindeki kod akınlarda alınması (instruction fetch)
- 2- IP yazmacının bir sonraki byte'si göstererek şekilde değiştirilmesi.
- 3- Alınan komutun ne komutu olduğunu ve ne tür parametreler ile çalışacığının tespit edilmesi (instruction decode)
- 4- Gerekli olması durumunda kullanıacak parametrelerin bellek akınlarda alınması (operand fetch)
- 5- IP yazmacının alınan parametre sayısı ve tiplerine de bağlı olarak gerekiyor ise bir sonraki komutu göstererek şekilde değiştirilmesi
- 6- Parametreler kullanılarak işlemin gerçekleştirilmesi (execute)
- 7- Elde edilen sonucun gerekli olağane yerleştirilmesi (store)

Fiziksel Adres Hesabı

Fiziksel Adres = Kesim yazmacı değeri $\times 16$ + işaret yazmacı değeri

$CS \leftarrow 2C58H$ ve $IP \leftarrow 0018H$ ise $CS:IP$ ikilisi hangi fiziksel adresi belirler?

16 ile çarpma = 4 bit sola kaydirmak

$$\begin{array}{r} 2C580H \\ + 0013H \\ \hline 2C593H \end{array}$$

MNEMONIC

{Label}: Mnemonic {{op1}, {op2}} ; Comment

Veri Aktarım Komutları

Veri aktarım komutlarının çalışması boyutlarını etkilemez.

MOV

- MOV RB, DADDR

↓
Register
Byte

→ bellek alanı
(variable)
↓
1 byte

- MOV RW, DADDR

↓
Register

- MOV DADDR, RB

- MOV AL, LABEL

- MOV DADDR, RW

- MOV AX, LABEL

- MOV LABEL, AL

- MOV LABEL, AX

- MOV SR, DADDR

- MOV DADDR, SR

↓
Segment register

Örnek

0 | MOV AX, sayı1

2 | MOV BX, sayı2

19 | Ortalaması MOV AX, BX

MOV AX, ortalama

19

④ Byte Type Registers

AX → AH AL

BX → BH BL

CX → CH CL

DX → DH DL

④ Register Words

AX SI

BX DI

CX BP

DX SP

④ Segment Registers

CS ES

DS SS

- MOV DADDR, DATA8

- MOV RB, DATA8

- MOV DADDR, DATA16

- MOV RW, DATA8

- MOV RBD, RES

- MOV RWD, RWS

- MOV SR, RW

- MOV RW, SR

LEA

Load Effective Address

- LEA RW, DADDR

adresini RW'e atar

LEA RB, DADDR $\rightarrow \text{sigma 8}$

MOV AX, BL

MOV RW, RB \rightarrow sigma 8 bile hata

LDS

Load Data Segment Register

- LDS RW, DADDR

Örnek

LDS SI, [0000] ^{myData}

SI $\leftarrow 2312h$ \rightarrow üst ilk 2
DS $\leftarrow 6745h$ \rightarrow Jaha üst ilk 2

0ABh	0006
89h	0005
67h	0004
45h	0003
23h	0002
12h	0001
	0000

→ my Data

LES

Load Extra Segment Register

- LES RW, DADDR

RW \leftarrow [DADDR]

ES \leftarrow [DADDR+2]

Örnek

LES DI, [0001] \rightarrow my Data

DI $\leftarrow 5351h$

SI $\leftarrow 4D48h$

4Dh	0005h
48h	0004h
53h	0003h
51h	0002h
41h	0001h
	0000h

→ My Data

XCHG

- XCHG RB, DADDR
- XCHG RW, DADDR
- XCHG AX, RW → 1 byte ve 5 clock cycle / karlı
- XCHG RBD, RBS
- XCHG RWD, RWS → 2 byte 4 clock cycle

İstelenen değerlerin değiş-tokus edilmesini sağlayan komut.

- XCHG BX, AX

- XCHG AX, BX

↳ aynı iş ama
1 clock cycle ve
1 byte uz.

Aritmetik Komutlar

ADD

- ADD op1, op2 ; $op1 \leftarrow op1 + op2$

- ADD RB, DADDR

- ADD RW, DADDR

- ADD AL, DATA8 → 2 byte 4 clock cycle

- ADD RB, DATA8 → 3 byte 4 clock cycle

- ADD DADDR, DADDR → Illegal

ADC

Add with Carry

ADD'a ek olarak CF bayragının değeri de toplama işlemine dahil edilir.

- ADC op1, op2 \sim Taşma durumu varsa bunu kullanı $op1 \leftarrow op1 + op2 + CF$

XADD

Exchange and ADD

- XADD op1, op2 →

$temp \leftarrow dest$
 $dest \leftarrow dest + src$
 $src \leftarrow temp$

SUB

- SUB op1, op2 $op1 \leftarrow op1 - op2$

SBB

Subtraction with borrow

- SBB op1, op2 $op1 \leftarrow op1 - op2 - CF$

DX%AX
BX%CX > 32 bit
→ SUB AX, CX
→ SBB DX, BX

INC

Increment

- INC op1 ; $op1 \leftarrow op1 + 1$

- INC DADDR

- INC RB → 3 clock cycle

- INC RW → 2 clock cycle

DEC

- DEC op1 ; $op1 \leftarrow op1 - 1$

NEG

Two's Complement

- NEG op1 ; $op1 \leftarrow \emptyset - op1$

CMP

Compare

- CMP op1, op2 ; $OP1 - OP2$

MUL

Unsigned Multiplication

- MUL op1

- MUL DADDR

- MUL RBS \longrightarrow MUL CL ; $AX \leftarrow AL * CL$

- MUL RWS \longrightarrow MUL BX ; $DX:AX \leftarrow AX * BX$

IMUL

Integer signed Multiplication

- IMUL RBS

- IMUL reg, Reg ; $opr1 \leftarrow opr2 * opr1$

- IMUL RWS

- IMUL reg, reg, idata ; $opr1 \leftarrow opr2 * opr3$

- IMUL DADDR

DIV

Division

- DIV RBS \longrightarrow DIV CH

$$\begin{array}{r} AX \\ \hline CH \\ | \\ AL \end{array}$$

- DIV RWS \longrightarrow DIV CX

$$\begin{array}{r} DX:AX \\ \hline CX \\ | \\ AX \end{array}$$

IDIV

- IDIV DADDR

DIV ile aynı galisir.

- IDIV RBS

- IDIV RWS

Dallanma Komutları

Koşulsuz Dallanma

- JMP

Koşullu Dallanma

Basit Koşullu Dallanma

- JZ (Jump Zero) / JE (Jump Equal)

- JNZ / JNE

- JS (Jump if Sign)

- JNS (Jump No Sign)

- JP (Jump on Parity)

- JNP (Jump no parity)

- JNO (Jump no overflow)

Above/Below/Equal

İsüretsiz Sayılar

- JB (Jump Below) / JNBE (Jump Not Above Equal)

- JA (Jump Above) / JNBE (Jump Not Below Equal)

- JAE (Jump Above Equal) / JNB (Jump Not Below)

- JBE (Jump Below Equal) / JNA (Jump Not Above)

Less/Greater/Equal

İsüretli Sayılar

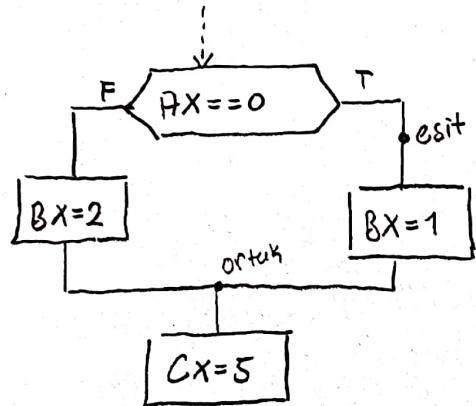
- JL (Jump Less) / JNGE

- JNL / JGE

- JLE / JNG

- JG / JNLE

Örnek



\equiv
CMP AX, 0

JE esit

MOV BX, 2

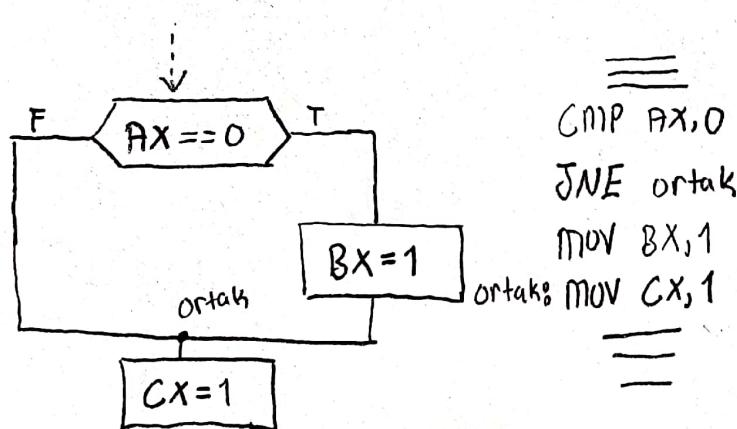
JMP ortak

esit: MOV BX, 1

ortak: MOV CX, 5

\equiv

Örnek



\equiv
CMP AX, 0

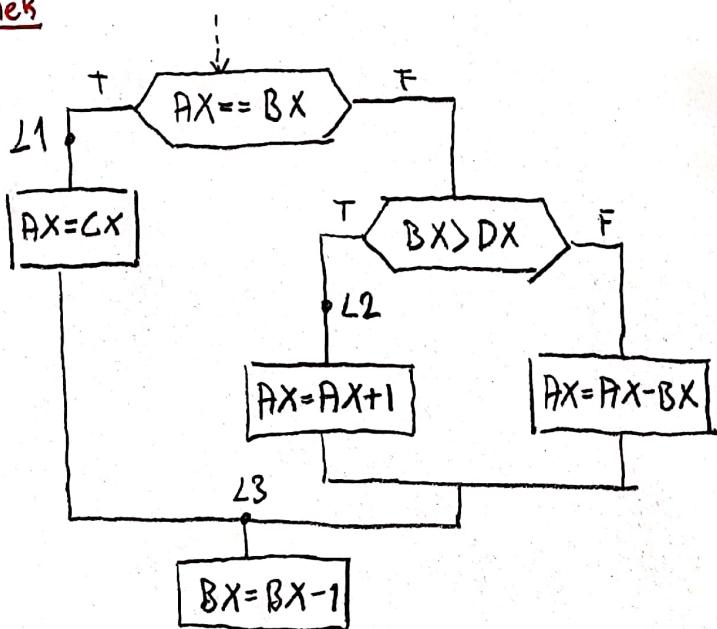
JNE ortak

MOV BX, 1

ortak: MOV CX, 1

\equiv

Örnek



\equiv
CMP AX, BX

JE L1

CMP BX, DX

JG L2

SUB AX, BX

L2: INC AX

JMP L3

L1: MOV AX, CX

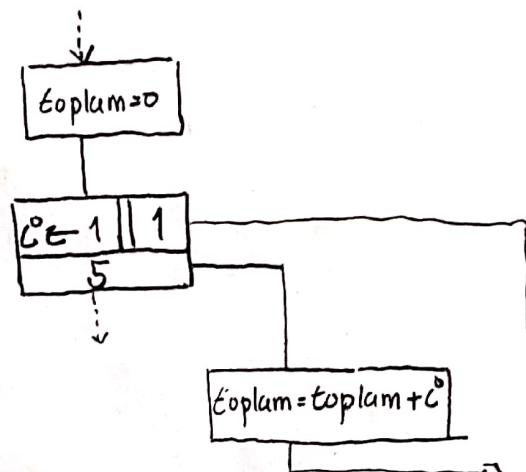
L3: DEC BX

LOOP

→ CX içerisindeki değer kadar döner

Gevrim değişkenleri → SI, DI, BX

Örnek

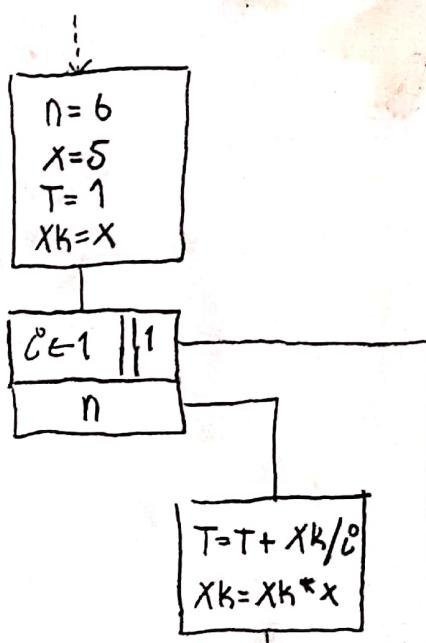


≡
 MOV toplam, 0
 MOV CX, 5
 MOV SI, 1
 L1: ADD toplam, SI
 INC SI
 LOOP L1
 ≡

Örnek

$$X \in [1,5], n \in [1,6]$$

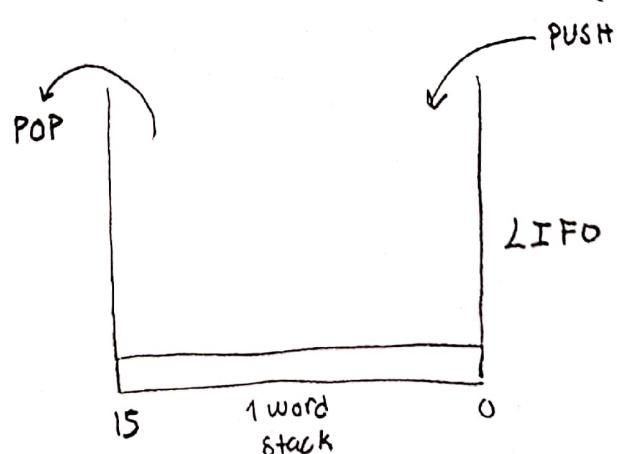
$$T = 1 + X + \frac{X^2}{2} + \frac{X^3}{3} + \dots + \frac{X^n}{n}, T = ?$$



≡
 MOV CX, 6
 MOV SI, 5
 MOV BX, 1
 MOV AX, SI
 MOV DI, 1
 L1: MOV DX, 0
 PUSH AX
 DIV DI
 ADD BX, AX
 POP AX
 MUL SI
 INC DI
 LOOP L1 ; sonuc = BX

$n \rightarrow CX$
 $x \rightarrow SI$
 $T \rightarrow BX$
 $Xk \rightarrow AX$
 $C \rightarrow DI$

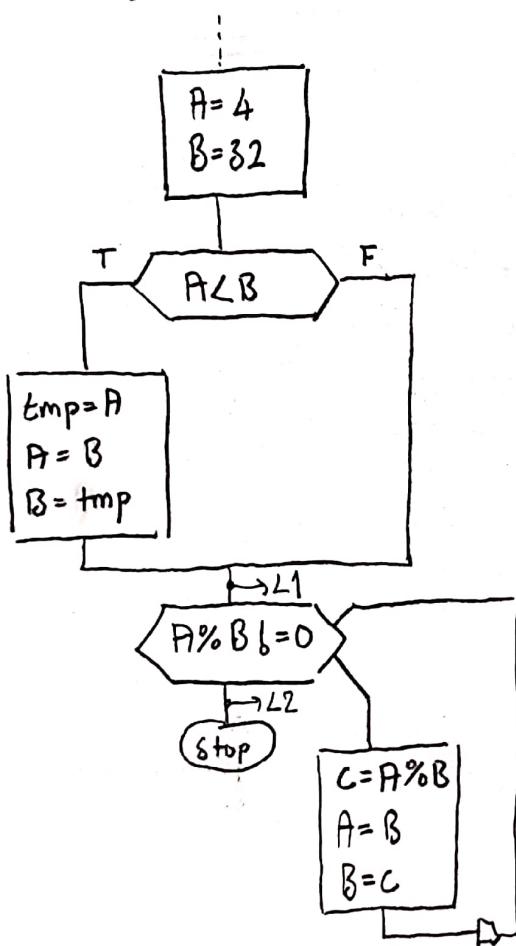
$\frac{DX:AX}{DX}$
 $DX:AX \leftarrow AX * SI$



Örnek

$A, B \in [0, 5000]$

$E80B(A, B) = ?$



$A \rightarrow AX$
 $B \rightarrow BX$
 $C \rightarrow DX$

L
 JAE
 JNB

\equiv
 $MOV AX, 4$

$MOV BX, 32$

$CMP AX, BX$

$JAE L1$

$XCHG AX, BX$

$L1: DIV BX ;$ kalan $DX'te$

$CMP DX, 0$

$JE L2$

$MOV AX, BX$

$MOV BX, DX$

$JMP L1$

$L2: \equiv$

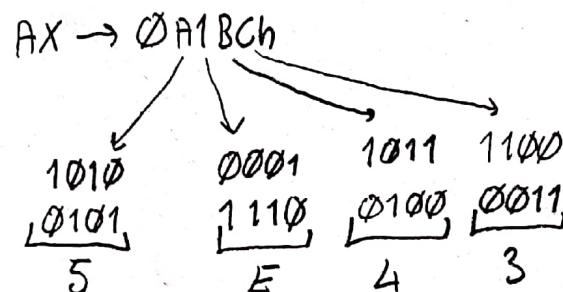
$; sonuc = BX$

Mantıksal Komutlar

NOT

- NOT op1

- NOT AX ; $\emptyset \emptyset FFh \rightarrow \emptyset \emptyset \emptyset \emptyset \emptyset \emptyset 11111111$
 $\emptyset FF\emptyset \emptyset h \quad 1111110000000000$



OR

- OR op1, op2

AL $\leftarrow 12h$

OR AL, 80h

$$\begin{array}{r}
 \emptyset \emptyset \emptyset 1 \emptyset \emptyset \emptyset 1 \emptyset \\
 \text{OR} \quad 1 \emptyset \emptyset \emptyset \emptyset \emptyset \emptyset \emptyset \\
 \hline
 1001 \emptyset \emptyset \emptyset 1 \emptyset
 \end{array}$$

AND

- AND op1, op2

AND'lıyor , sonuc Op1'de

TEST

- TEST op1, op2

AND'lıyor , sonuc buyraklarda

XOR

- XOR op1, op2

↑
sonuc

XOR op1, op2 = 0

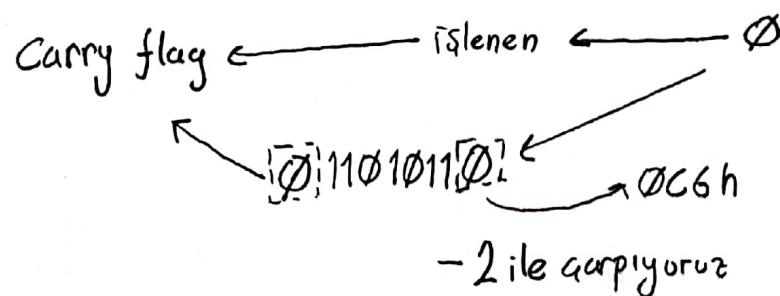
Öteleme Komutları

SHL

Shift Left Logical

MOV AL, 6Bh

- SHL AL, 1

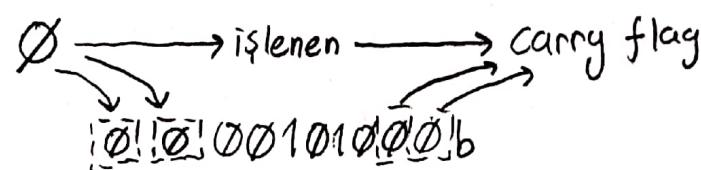


SHR

Shift Right Logical

MOV AL, 0010101000b

- SHR AL, 2



Örnek

"Saye" değişkenindeki değer çift ise $AX=1$, tek ise $AX=0$ yapan ASM Kodu.

```

SHR saye, 1
JC tek
MOV AX, 0
JMP ortak
tek: MOV AX, 1
    
```

JC → 1'e eşitler
0 ise eşitmez

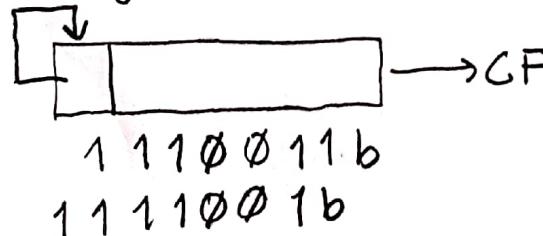
SAL

Shift Arithmetic Left

SAL=SHL

SAR

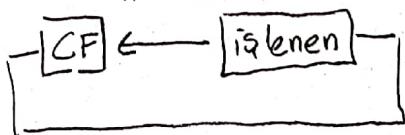
Shift Arithmetic Right



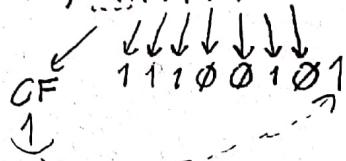
Döndürme Komutları

[RCL]

Rotate Left

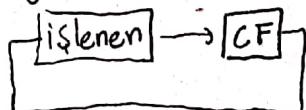


MOV BL, 0F2h ; 11110010b



[RCR]

Rotate Right

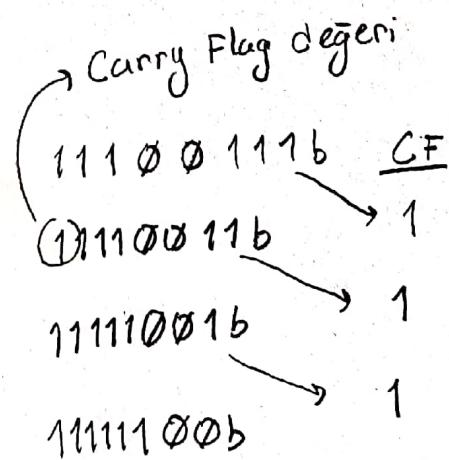


; CF ← 1

MOV BL, 11100111b

MOV CL, 3

BCR, BL, CL

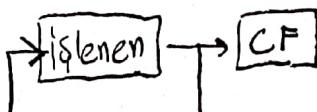


[ROL]



CF = MSB ← 1011
0111

[ROB]



Bayrak Komutları

CLC

Clear CF

$$CF \leftarrow \emptyset$$

STC

Set CF

$$CF \leftarrow 1$$

STD

Set Direction Flag

$$DF \leftarrow 1$$

CMC

Complement CF

$$CF \leftarrow \neg CF$$

CLD

Clear Direction Flag

$$DF \leftarrow \emptyset$$

STI

Set Interrupt Flag

$$IF \leftarrow 1$$

CLI

Clear Interrupt Flag

$$IF \leftarrow \emptyset$$

L AHF

Load AH with flag

SF	ZF	? AF	? PF	? CF
----	----	------	------	------

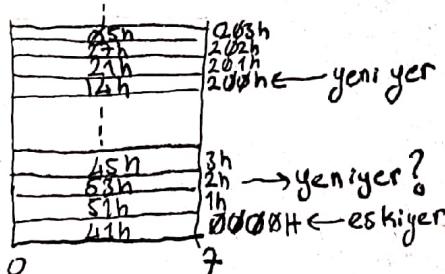
flag \rightarrow AH

SAHF

Store AH in flag

$$AH \rightarrow \text{flag}$$

String (Dizgi) Komutları (w)



MOVSB

Move String Byte

$$[ES:DI] \leftarrow [DS:SI]$$

DF \rightarrow \emptyset \rightarrow adresler artarak
 $\rightarrow 1$ \rightarrow adresler atalacak

$$05h \leftarrow 45h$$

$$27h \leftarrow 53h$$

$$21h \leftarrow 51h$$

$$14h \leftarrow 41h$$

LEA SI, eskiyer

LEA DI, yeniyer

MOV CX, 4

CLD

REP MOVSB

ayni
Komut
4 kez
repeat

CMPSB

ES

$$[DS:SI] - [DS:DI] \quad 2 \text{ dizigi karşılaştırılmış } [10]$$

$$\begin{matrix} DF \rightarrow \emptyset \\ DF \rightarrow 1 \end{matrix}$$

Repeat
Equal

LEA SI, dizii1

LEA DI, dizii2

MOV CX, 10

CLD

REPE CMPSB

SCASB

Scan String Byte

AL register ile [ES:DI]

adresindeki byte'ı karşılaştırıyor.

Bayraklar etkileniyor.

AL - [ES:DI]

DF → 0↑
→ 1↓

204h	6DH
208h	6Ph
20Fh	63h
20Ah	27h
208h	48h
207h	46h
206h	40h
205h	61h
204h	60h
208h	75h
202h	70h
201h	55h

← email

② → 40h

LEA DI, email

MOV CX, 12

CLD

MOV AL, '@'

REPNE SCASB

LODSB

AL ← [DS:SI]

LEA SI, eskiyer

LODSB

STOSB

[ES:DI] ← AL

CBW

Convert byte to word

AX ← AL

AL ← 10001111b

CBW ; 1111 1111 10001111b → AX
AH AL

CWD

DX:AX ← AX

Örnek

Diger bayraklara dokunmadan Zero Flag'in complementini alalım

$$ZF : \emptyset \rightarrow 1 \\ 1 \rightarrow \emptyset$$

PUSHF

LAHF

XOR AH, 40h

POPF

SAHF

SF	ZF	?	AF	?	PF	?	CF
0	1	0	1	0	1	1	1
XOR	0	1	0	0	0	0	0
				0	1	0	1

LAHF
SAHF
PUSHF
POPF

Örnek

Verilen bir byte'daki bitleri sayalım.

XOR BL, BL ①

CF ← $\begin{array}{c} 01010101 \\ \hline s \quad s \end{array}$ h

MOV CX, 8

MOV AH, 55h

L1: SHL AH, 1

$\begin{array}{c} 01101000 \\ 11010000 \\ 10100001 \end{array}$ b

JNC L2

INC BL

L2: LOOP L1

JNC → Jump not carry

②

XOR BL, BL

MOV CX, 8

MOV AH, 0ABh

L1: SHL AH, 1

ADC BL, 0

LOOP L1

③

XOR BL, BL

MOV AH, 55h

L1: CMP AH, 0

JE Cikis

SHL AH, 1

ADC BL, 0

JMP L1

Cikis: \equiv

Adresleme Kipleri

1) Anlık (Immediate) Adresleme

MOV {reg, mem}, cdata

MOV AL, 16

MOV sayi, FFFFh

2) Register Adresleme

MOV reg, reg

MOV DS, AX

MOV DH, AL

5) Doğrudan İndisli (Direct Index) Adr.

Dizi DB 41h, 53h, 53h, 45h, 4Dh, 42h, 4Ch, 59h

XOR SI, SI

MOV AL, dizi[SI]; AL ← 41h

INC SI

MOV AH, dizi[SI]; AX ← 5341h

Dizi2DW DW 5841h, 4553h, 424Dh, 594Ch

XOR BX, BX

MOV DX, dizi2[BX]

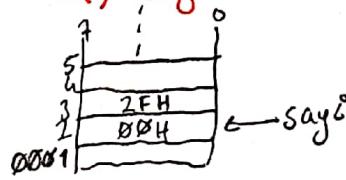
ADD BX, 2

MOV AX, dizi2[BX]

7) String Adresleme

8) istekle (PORT) Adresleme

4) Doğrudan (Direct) Adresleme



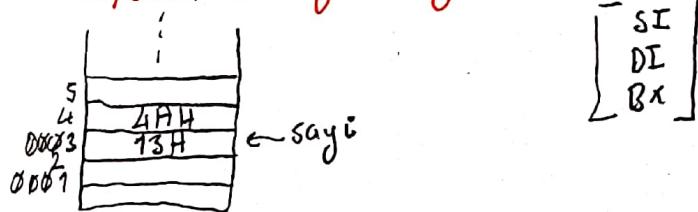
sayi → byte

MOV AL, sayi ; AL : 00H

sayi → word

MOV CX, sayi ; CX : 2F00H ; CL : 00 ; CH : 2F

4) Yazmaq Dolaylı (Register Indirect) Adr.



LEA SI, sayi ; SI ← 0003h

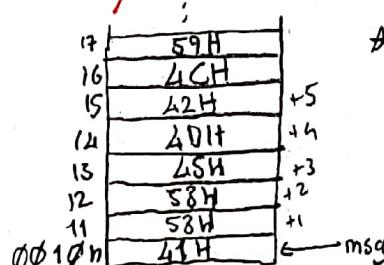
MOV AX, SI ; AX ← 0003h

MOV AX, [SI] ; AX ← 4A13h

MOV DI, OFFSET sayi

MOV CX, [DI] ; CX ← 4A13h

6) Baz GÖRELİ (Base Relative) Adresleme



* msg db 'ASSEMBLY'

LEA BX, msg

MOV AL, [BX+4] ; AL ← 4Dh

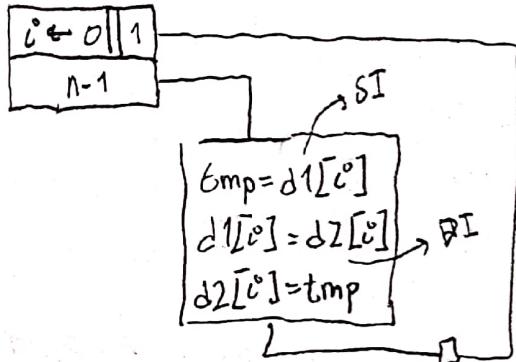
MOV AX, [BX+5] ; AX ← 4C42h

Örnek

2 dizinin yerini farklı adresleme tipleri kullanarak değiştirelim.

$d_1, d_2 \rightarrow$ byte tanımlılar

$n \rightarrow$ her iki dizinin de boyutu



$\text{MOV CX, } n$
 XOR SI, SI
 $\text{LEA DI, } d_2$
 $\text{MOV AL, } d_1[\text{SI}]$
 $L1: \text{XCHG AL, DI}$
 $\text{MOV } d_1[\text{SI}], AL$
 INC SI
 INC DI
 LOOP L1

$\nearrow 12$

~~$\text{XCHG } d_1[\text{SI}], [DI]$~~

Örnek

A şirketinde çalışanların doğum yolları d_1 , B şirketinde çalışanların yaşları d_2 dizisinde saklanıyor. Önce A şirketinde çalışan al kışının, sonra B şirketinde çalışan b1 kışının bu bilgileri boyutlu a_1+b_1 olar d_3 dizisine aktarılması isteniyor.

$d_1 \rightarrow$ word
 $d_2 \rightarrow$ byte
 $d_3 \rightarrow$ word

$\text{LEA DI, } d_1$
 $\text{LEA SI, } d_2$
 XOR BX, BX
 $\text{MOV CX, } a_1$
 $L1: \text{MOV AX, [DI]}$
 $\text{MOV } d_3[BX], AX$
 $\text{ADD BX, } 2$
 $\text{ADD DI, } 2$
 LOOP L1
 $\text{MOV CX, } b_1$
 $L2: \text{MOV AL, [SI]}$
 CBW
 $\text{MOV } d_3[BX], AX$
 $\text{ADD BX, } 2$
 INC SI
 LOOP L2

$\text{DI word adrese erişecek}$
 $\text{SI byte adrese erişecek}$
 $\text{BX word adrese erişecek}$

~~MOV mem, mem~~
 ~~$\text{MOV DI, DI}, \text{DX, DX}$~~
 $\left\{ \text{MOV } d_3[BX], [DI] \right.$

$\text{MOV } d_3[BX], AL \longrightarrow$
 \downarrow
 word by+8
 \checkmark
 olmaz
 $\longrightarrow \text{XOR AH, AH}$
 $\text{MOV } d_3[BX], AX$
 \downarrow
 olur una
 MSB ye bükmez
 signed olabilir

Sözde (Pseudo) Komutlar

- Makine Kodu Karşılığı yok
- Bayraklara etkileri yok
- Debug ortamında kullanılmıyor
- LST dosya düzeni
- Kesimleri düzeltmek
- Veri taşımalarını gerçekleştirmek

SEGMENT/ENDS

Kesim_ismi SEGMENT {seçenekler}

≡
≡
≡

Kesim_ismi ENDS

XXXX = YY00 01 02 03 04 05 06 07 08 - 09 0A 0B 0C 0D 0E 0F
XXXX = YY10 - - - - - - - -
P PARA

Byte word

* Seçenekler

a) Hizalama (Alignment) Tipi

- BYTE /1
- WORD /2
- PARA /16
- PAGE

b) Birleştirme (combine) Tipi

- PUBLIC
- COMMON
- STACK → LIFO
- AT #####

c) Sınıf (Class) Tipi

Stacksgmt SEGMENT PARA STACK "yığın"

≡
≡

Stacksgmt ENDS

codesgmt SEGMENT WORD "Kod"

≡

codesgmt ENDS

myds SEGMENT PAGE PUBLIC "veri"

≡

myds ENDS

ASSUME

EXE tipi → ASSUME SS: stacksgmt, CS: codesgmt, DS: myds

Com tipi → ASSUME SS: myseg, DS: myseg, CS: myseg

DB (Define Byte)

sayc1 DB 25
 sayc2 DB 1EH
 sayc3 DB ?
 dizit1 DB 1,2,0111@101b,1Fh,12
 strb DB 'Assembly'

DW (Define Word)

sayı1 DW 0ABCDh

DD → 4 byte

DA → 8 byte

DT → 10 byte

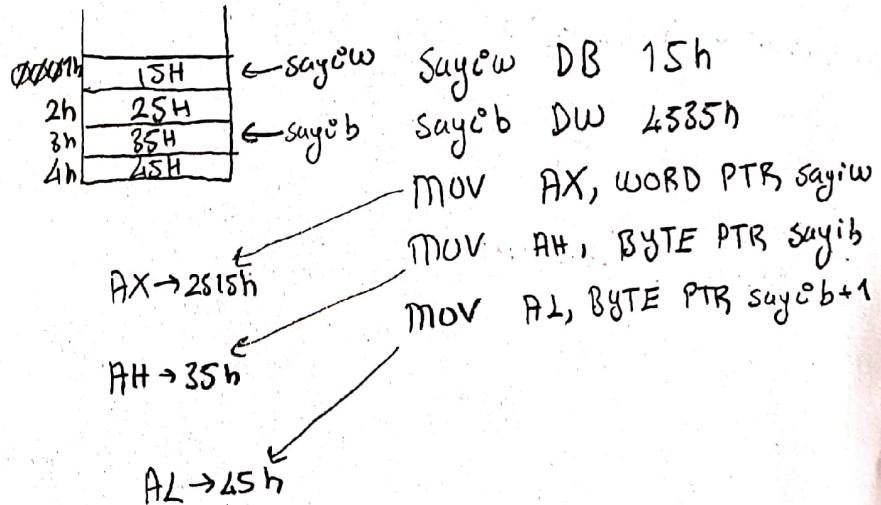
DUP (Duplication Factor)

dizib2 DW 15 DUP(0)
 m61 DW 3 DUP(4 DUP(8))

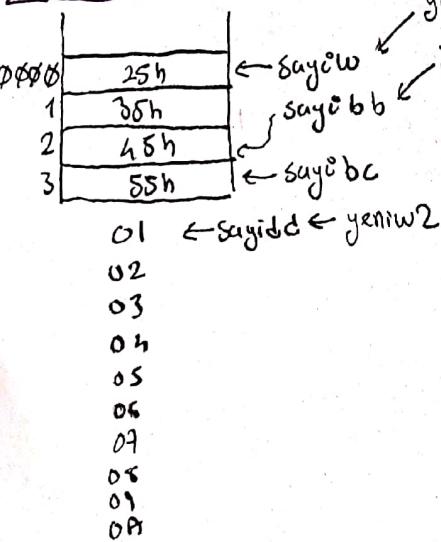
3x4 (8 8 8 8
 8 8 8 8
 8 8 8 8)

PTR (Pointer)

* Tip dönüşümü için kullanıyor (casting)



LABEL



yenid LABEL WORD
 sayıdd DT 0A090807060504030201

yeniw2 LABEL WORD
 sayıdd DT 0A090807060504030201

PROC (Procedure) / ENDP

yordam-ismi PROC {NEAR/FAR\$
 |
 ||
 |||
 RET/RETF

yordam-ismi ENDP

TYPE

sizeof();
MOV AX, TYPE Tablo
 |
 0002h

SIZE

LENGTH * SIZE
MOV AX, SIZE Tablo
 |
 80

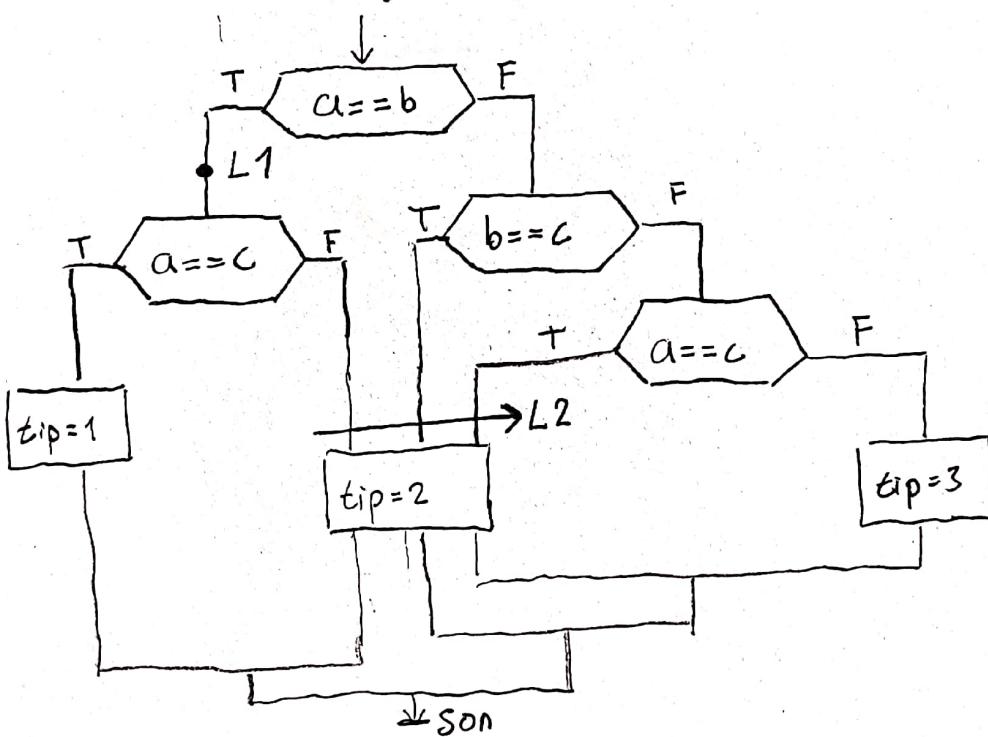
Tablo DW 15 DUP(?)

-MOV AX, LENGTH Tablo
 |
 000Fh

EXE Tipi 8

Örnek

3 kenar bilinen bir üçgenin tipini bulalıma



```

datasg SEGMENT PARA 'Veri'
a DB 12
b DB 17
c DB 12
tip DB ?
datasg ENDS
stacksg SEGMENT PARA STACK 'Yığın'
DW 12 DUP(?)
stacksg ENDS
codesg SEGMENT PARA 'Kod'
ASSUME CS:codesg, DS:datasg, SS:stacksg
AIFA PROC FAR
PUSH DS
XOR AX, AX } → Geniş döner adreslerinin stack'te saklanması → DS:0000
PUSH AX
MOV AX, datasg } DS'ye erişmek için ayar yapıyor.
MOV DS, AX
MOV AL, a
MOV BL, b
MOV CL, c
  
```

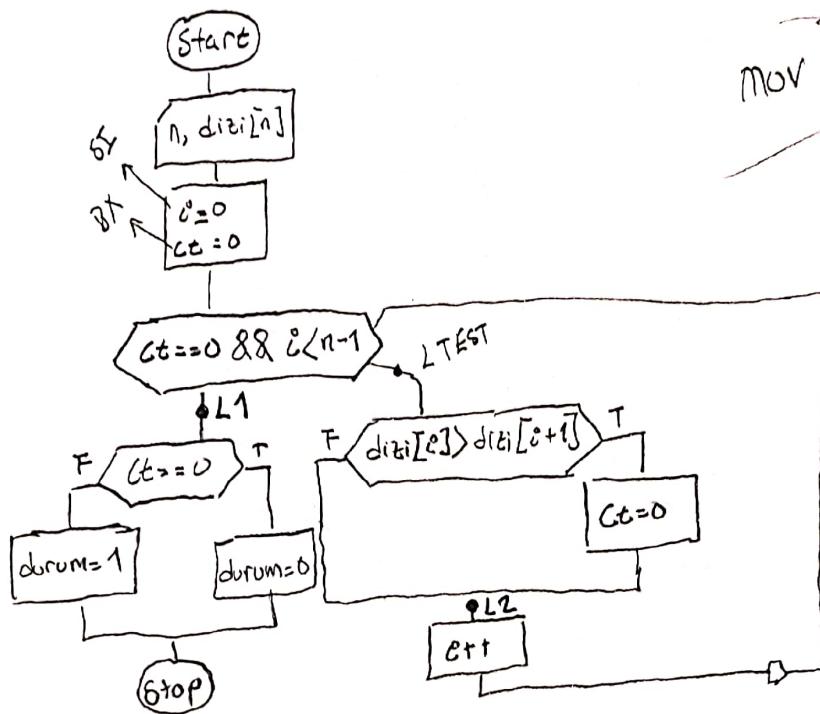
①

①

```
CMP AL, BL  
JE L1  
CMP BL, CL  
JE L2  
CMP AL, CL  
JE L2  
MOV tip, 3  
JMP son  
L1: CMP AL, CL  
JNE L2  
MOV tip, 1  
JMP son  
L2: MOV tip, 2  
son: RETF ; RETURN FAR  
ANIA ENDP  
codesy ENDS  
END ANIA
```

ÖRNEK

byte tamlı bir dizinin kırıktan büyük sıralı olup olmadığı, bulunuz.



~~MOV dizi[SI], dizi[SI+1]~~

```

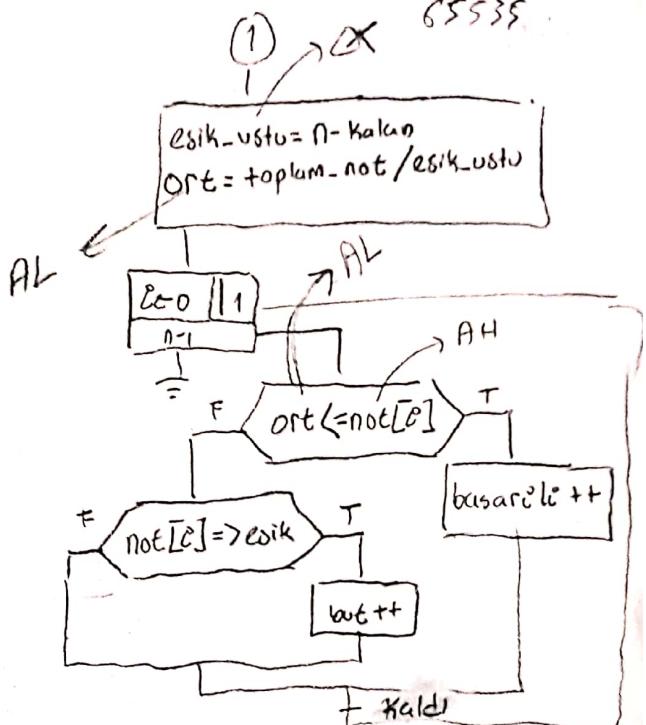
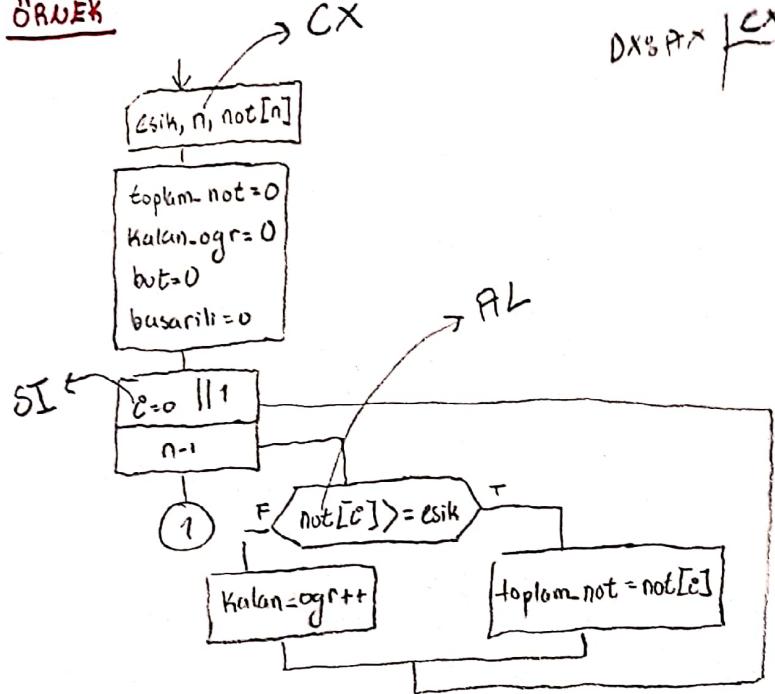
myds SEGMENT PARA 'data'
dizi DB 1, 2, 3, 4, 5
n DW 5
durum DB ?
myds ENDS
myss SEGMENT PARA STACK 'stack'
DW 12 DUP(?)
myss ENDS
mycs SEGMENT PARA 'code'
ASSUME CS:mycs, SS:myss, DS:myds
SIRALIMI PROC FAR
    PUSH DS
    XOR AX, AX
    PUSH AX
    MOV AX, myds
    MOV DS, AX
    MOV BX, 0
    XOR SI, SI
    MOV CX, n
    DEC CX
    while : CMP BX, 0
    JNE L1
    CMP SI, CX
    JAE L1
  
```

(1)

Ltest :	MOV AL, dizi[SI]
	CMP AL, dizi[SI+1]
JLE L2	MOV BX, 1
	INC SI
JMP while	CMP BX, 0
L1 :	JNE sirasiz
	Mov durum, 1
	JMP L3
sirasiz :	Mov durum, 0
L3 :	RETF
SIRALIMI ENDP	
mycs ENDS	

END SIRALIMI

ÖRNEK



myds SEGMENT PARA 'data'

not DB 280 DUP(3)

ogr-say DW 280

Kalan DW Ø

but DW Ø

basarili DW Ø

toplum-not DW Ø

esik DB 40

myds ENDS

myss SEGMENT PARA STACK 'y'

DW 20 DUP(3)

myss ENDS

mycs SEGMENT PARA 'code'

ASSUME CS:mycs, DS:myds, SS:myss

CAN PROC FAR

PUSH DS

XOR AX, AX

PUSH AX

MOV AX, myds

MOV DS, AX

XOR SI, SI

MOV CX, ogr-say

L2: MOV AL, not[SI]

CMP AL, esik

JBE gesti

esikustu

INC Kalan

JMP L1

esikustu CBW

ADD toplam-not, AX

L1: INC SI

LOOP L2

MOV CX, ogr-say

SUB CX, Kalan

XOR DX, DX

MOV AX, toplam-not

DIV CX

XOR SI, SI

MOV CX, ogr-say

L2: MOV AH, not[SI]

CMP AL, AH

JBE gesti

CMP AH, esik

JB Kaldi

INC but

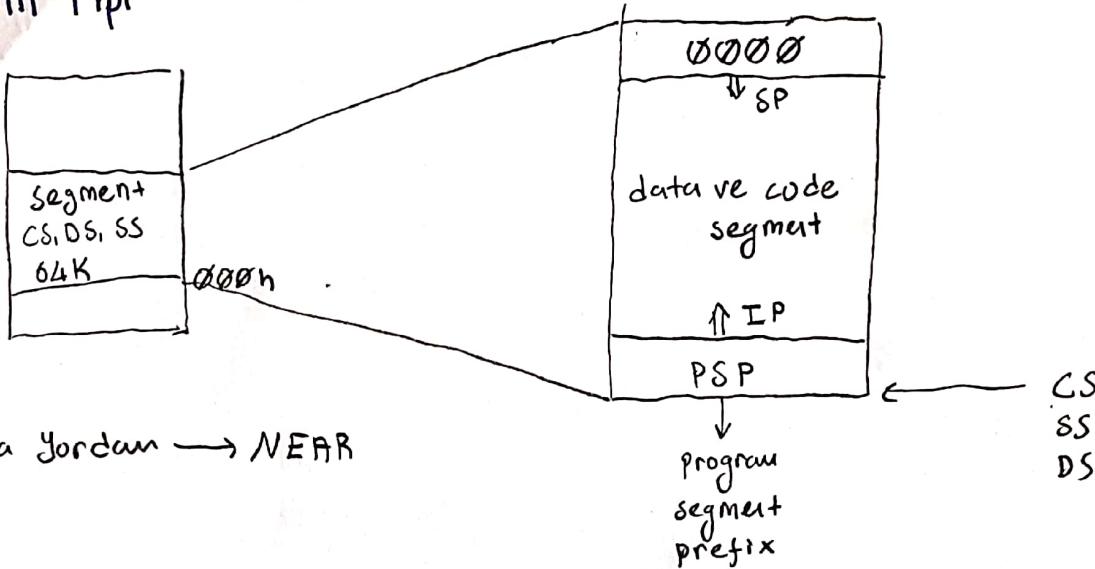
JMP Kaldi

gesti: INC basarili

Kaldi: INC SI

LOOP L2

COM Tipi



Syntax Model 1

```
codesg SEGMENT PARA 'a'  
    ORG 100H  
    ASSUME CS:codesg, SS:codesg, DS:codesg  
  
datasg: JMP main  
    sayi1 DB 17  
    sayi2 DW 100  
  
main: PROC NEAR  
      
    RET  
main ENDP  
  
codesg ENDS  
END main datasg
```

Syntax Model 2

```
codesg SEGMENT PARA 'b'  
    ORG 100H  
    ASSUME CS:codesg, SS:codesg, DS:codesg  
  
main PROC NEAR  
      
    RET  
main ENDP  
    sayi1 DB 17  
    sayi2 DW 100  
codesg ENDS  
END main
```

EXE Tipi

Syntax model

myds SEGMENT PARA 'a'

≡

myds ENDS

myss SEGMENT PARA STACK 'b'

DW 20 DUP(?)

myss ENDS

mycs SEGMENT PARA 'c'

ASSUME CS:mycs, DS:myds, SS:myss

main PROC FAR

PUSH DS

XOR AX,AX

PUSH AX

Mov AX, myds

Mov DS, AX

≡

RETF

main ENDP

mycs ENDS

END main

COM/EXE

	Com	EXE
Bellek Miktarı :	64 KB	Bos bellek ile sınırlı.
CS Baslangic Degeri :	PSP'nin bulunduğu adres	END'i takip eden etiket ile Kodun kesim adresi.
IP // // :	100H	END ile belirlenen göreli konum adresi.
DS // // :	PSP'nin bulunduğu adres	Dönüş i̇in gerekli olan Kesim adresi.
Tanimli Degiskenlere : Enisim icin Yapilmali :	yok	MOV AX,myds , MOV DS, AX
ES:	PSP'nin bulunduğu adres	PSP'nin bulunduğu adres.
SS:	PSP	Tanimlanan yigin Kesimi adresi
SP:	0FFF Eh	Tanimluan yigin Kesiminin boyutu ile belirlenen deger.
Yigindaki ilk deger :	0000h	Bos.
Ana Yordam Tipi:	NEAR	FAR
Alt // // :	NEAR	NEAR/FAR
Programi Bitirmek i̇in :	RET / INT21 \$4CHF	RETF / INT 21 \$4CHF
Boyut :	Programin byte cinsinden Karsiligi	program boyutu + 512 byte EXE header
Calisma Onceligi :	EXE'den once	COM'dan sonra

Yordamlar

PROC pseudo komutu ile başlar ENDP sözde komutu ile sonlanır.

NEAR → Kesim içi

FAR → Kesimler arası

RET → NEAR] → Gağırlıkları yere dönebilmesi için.
RETF → FAR]

Yordam Tanımı

yordam_ismi PROC {NEAR|FAR}

↳ yordamın kodu

RET

yordam_ismi ENDP

DX:AX ← AX^SI

MUL SI

Kesim içi Yordam Kullanımı

Kullanımda gağırlar ve gagnları aynı Kesim içinde yer alır.

myss SEGMENT PARA STACK 'ss'
DW 20 DUP(?)

myss ENDS

myds SEGMENT PARA 'veri'

Sayı DB2

UST DW10

SOME DW?

myds ENDS

mycs SEGMENT PARA 'code'

ASSUME DS:myds, CS:mycs, SS:myss

ANA PROC, FAR

PUSH DS

XOR AX, AX

PUSH AX

MOV AX, myds

MOV DS, AX

XOR BX, BX

MOV BL, Sayı

MOV CX, UST

CALL POW

MOV Sonuc, AX

RETF

ANA ENDP

①
POW PROC NEAR

PUSH DX

MOV AX, 1

L18 MUL BX

LOOP L1

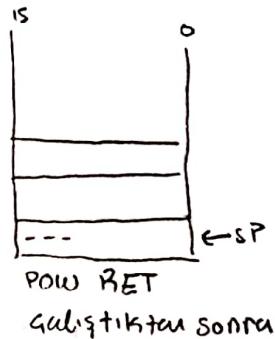
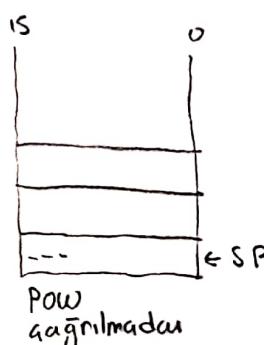
POP DX

RET

ENDP

mycs ENDS

END ANA



①

Kesimler Arası Yardım Kullanımı

Yardamlar aynı Kesimlerde bulunmaya bilir.

- Bir Kesim ile sınırlı kalmayıp program kodunu büyütme.
- I/O bilinen birkaçı tarafından yazılımsız yardım edilebilir.

Örnek

pseudo

```
EXTRN TOPLAMA: FAR
myss SEGMENT PARA STACK 'ss'
DW 20 DUP(?)
my6S ENDS
myds SEGMENT PARA 'ds'
sayi1 DB 17
sayi2 DB 29
sonuc DW ?
myds ENDS
mycs SEGMENT PARA 'cs'
ASSUME CS:mycs, DS:myds, SS:myss
MAIN PROC FAR
    PUSH DS
    XOR AX, AX
    PUSH AX
    MOV AX, myds
    MOV DS, AX
    MOV BL, sayi1
    MOV BH, sayi2
    CALL TOPLAMA
    MOV sonuc, AX
    RETF
MAIN ENDP
END MAIN
```

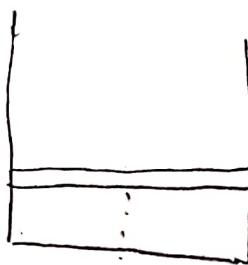
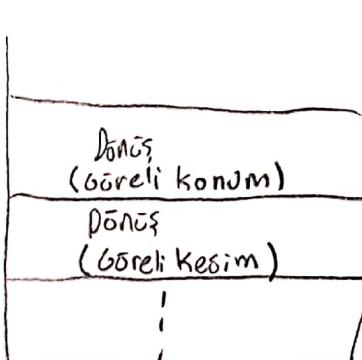
pseudo

```
PUBLIC TOPLAMA,
codes SEGMENT PARA 'cs'
ASSUME CS:codes
TOPLAMA PROC FAR
    XOR AX, AX
    MOV AL, BL
    ADD AL, BH
    CBW ; ADD AH, 0
    RETF
TOPLAMA ENDP
codes ENDS
END
```

farklı yardımlar varsa

Bu segmentte farklı yardımlar da tanımlayabilirsin.

Farklı
segmentler



Makrolar

ismi geçtiği her yerde kendilerine ait kod bloğunu kopyalarlar.
Makro kullanılırken sadece macro-name & parameters ifadesi kod içine
yazılmalıdır. Önemli olan makronun adının geçtiği yerde önce tanımlanmış
olmasıdır. Eğer istenirse makrolar ayrı bir metin dosyası olarak yazılarak
programa INCLUDE sözde komutıyla da dahil edilebilirler.

Makro Tanımı

```
macro-name MACRO &parameters  
LOCAL ---  
ASSUME ---  
; makro kodu  
ENDM
```

→ Label

LOCAL → Kullanılan etiketlerin unique olma durumu.
ASSUME → Kesim eslemelerinde değişiklik.

Power MACRO

```
LOCAL L1  
MOV AX, 1  
L1: MUL BX  
LOOP L1  
ENDM
```

Main PROC FAR

```
MOV CX, ust  
XOR BX, BX  
MOV BL, sayı1  
power  
MOV Sonuc, AX  
RET  
Main ENDP
```

Örnek

```
mycs SEGMENT PARA 'code'  
    ORG 100H  
    ASSUME CS:mycs, SS:mycs, DS:mycs  
min MACRO dizi, n  
    LOCAL L1  
    XOR SI, SI  
    MOV AL, dizi[SI]  
    INC SI  
    MOV CX, n  
    DEC CX  
L1: CMP AL, dizi[SI]  
    JL son  
    MOV AL, dizi[SI]  
son: INC SI  
    LOOP L1  
    ENDM  
main PROC NEAR  
    XOR SI, SI  
    MOV CX, n  
L1: SAR dizi[SI], 1  
    INC SI  
    LOOP L1  
    min dizi, n  
    MOV KCK, AL  
    RET  
main ENDP  
dizi DB 10, -2, 10, 3, 3, 17, 9  
n DW 8  
KCK DB ?  
mycs ENDS  
END main
```

Type Casting

• ADD DD-var, AX

• CBW

ADC WORD PTR DD-var[2], 0

• CWD

• MOV DX, DD-var[2]

MOV AX, DD-var[0]

MNEMONICS

ADC → ADC op₁, op₂; op₁ ← op₁ + op₂ + CF

DX:AX
+ CX:BX

ADD AX,BX
ADC DX,CX

DX:AX
- CX:BX

SUB AX,BX
SBB DX,CX

SBB → SBB op₁, op₂; op₁ ← op₁ - op₂ - CF

MUL → MUL op₁; AX ← AL * CL

MUL BX; DX:AX ← AX * BX

DIV CH; AX | CH

DIV CX; AX | CX

TEST → TEST op₁, op₂ → AND'liyor ve sonua bayraklarda.

SHL → MOV CL, 2 ~ SHL AL, CL → 2 ile çarpma

SAR → 2 ile bölmek

LEA SI, eski;
LEA DI, yeni
MOV CX, 4
CLD
REP MOVEB

CMPSB → Compare string byte → LEA SI, digit1
LEA DI, digit2
MOV CX, 10
CLD
REPE CMPSB

SCASB → Scan string byte → LEA DI, Email
MOV CX, 12
CLD
MOV AL, '@'
REPNE SCASB

● Direct Indexing →

XOR SI, SI
MOV AL, [SI]
INC SI

● Register Indirect →

LEA SI, sayi
MOV AX, [SI] | MOV DI, offset sayi
MOV AX, [DI]

● Base Relative →

LEA BX, sayi
MOV AL, [BX+4]
MOV CL, [BX+5]