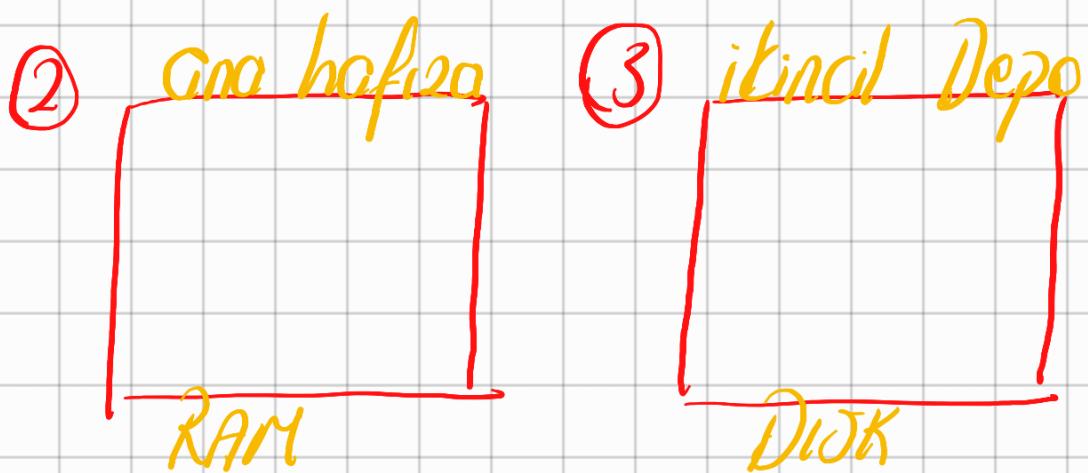


→ yazmaç register



Birim üretim məbləyəti bu ərafdan
dəprə ortası

hafıza təqərisi
ortası

Ümumi dərinliyi TPU mikroişlemcinin 5 indek vəzifələri

Yazma, okunup yazılabilir hale getirilen tek tek adreslenebilen ve erişilebilen hafıza birimleridir

Mikroişlemcinin içinde yazma, kontrol birimi, ALU ve bu boy- bantları yapıyon yollarır

Yazmalar yüksek hızlı hafıza birimleridir

A_x

AL

AH

C_x

C_L

C_H

B_x

B_L

B_H

D_x

D_L

D_H

JI

DI

J_P

DP

JS

CS

DS

ES

IP

Flags

CF

PF

AF

2F
SF
TF
IF
DF
OF

32 BIT μP yarmoclar (80386)

• EAX

• AX

• AL
• AH

• EBX

• BX

• BL

• BH

• ECX

• CX

• CL

• CH

• EDX

• DX

• DL

• DH

ESP

EIP

EDP

ESI

EDI → Extended

EFLAGS

SS

CX

DX

ES

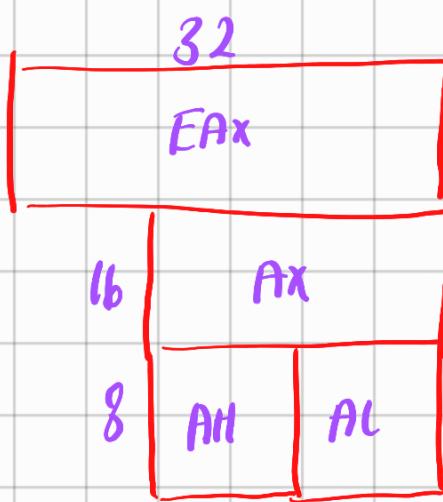
EIP

F5

G5

Bunlar yeni yaradılar

→ 32 Bitlik mimarisde DI ve DI
yok EDI ve EDI var



64 bit NP yaradıları

RAX → 64

EAX → 32

AX → 16

AL

SS 32

CX 32

DX 32

ES 32

AH

FJ 32
GS 32

RBX

RIP

EBX

R8-R15

BX

RFLAGS

BL

BH

X_{mm}
y_{mm}
z_{mm}

Bunlarda
vektörel
yazmalar

RDI

64

RDI

ECX 32

CX 16

CL 8

CH 8

RDX

EDX

DX

DC

DH

RSP

64

RBP

64

[BX]

$\rightarrow [EBX] [EDI]$

64

RAX

32

EAX

16

AX

8

AH

AL

8086 yazmacaları

genel amaçlı yazmacalar → Bütlerde kod içinde erişebiliriz

AX → etkileşimci yazmac
(accumulator 16 bitlik)

AL → 8 bitlik accumulator

Accumulator için özel donanım özel instructionları var oynı
işlemi AX ve AL üzerinden gerçekleştirsem daha hızlı
gerçekleştiriliyor ve daha da sayıda bit ile ifade edilen
komutlar karsımıza çıkıyor genel kullanımını Daha
işlemlerinde veri işlemlerinde kullanabileceğimiz bir
yazmacalar

AX ve AL , DIV ve MUL gibi işlemler-
de gizli operand gibi solşır ve 110 (input
output işlemlerinde varsayılan operand) ola-
rat kullanılır

Bx (Base Register) → Hafıza üzerinde bir tablo, arrayin başlangıcını gösterir BL ve BH'in böyle bir şey yok dedece BX'in var

[] → Bu işaret içinde hafıza gösterine ulaşabiliriz

[BX] → Böyle olabilir

[DI]

[DI]

[BL] [BH]

Böyle yazamayız

Ax'leri de yazamayız

[DL] [DH] [DX]

Böyle yazamayız

JI → Jource index

DI → Definition index

yazmak register dedigimiz şey bir bellek elemani olsun-
da

BX bir tablonun arayın başlangıç adresini tutuyor ise
 SI re DI 'da bir pozü indexlemek için kullanılır
Başlangıçtan itibaren ne kadar ötede bunu söyleyebiliriz

Ben bir tablonun belidi bir gelenek erişmek istedim
 $[BX+SI]$ şeklinde erişebiliriz

BX ile başlayan tablonun SI indexindeki, indisindeki yeridir SI yerine DI 'da yazabilirim
bunlar index yapmag

$CX \rightarrow$ count register (döngü işlemlerinde
kullanılır ve
 CL re CH 'da işlemleri de
belirli döngü işlem
lerinde kullanırız)

$DX \rightarrow$ Data register (Yeri işlemlerinde kullanılır)

DX re MUL işlemlerinde gizli operand
 $1/0$ (input output işlemlerinde varsa yılan operand
olarak kullanılıyor)

$SP \rightarrow$ Stack pointer

BP → Base pointer

Bunlar yığın ile ilgili yazmacaları

Push komutu çalıştırıldığında POP, CALL, INT, RET herhangi birini çalıştırıldığında yığınındaki en üsteki onlamlı veriyi göstermek adına Stack pointer kullanılır

→ Herhangi bir yığın işlemi

BP yığınla ilişkili yazmacı SP'yi mikroişlemci degistirirken SP'yi değiştirmiyice dolayısıyla yığın içerisinde beli offset'teki değere ulaşmak için sabit bir referans gaza gerek görür. BP etki içinde bir şey diye bir veriye ulaşırız

2. grup yazmacaları (yığın yazmacaları)

8086 - 16 bitlik yazmacalar var 20 bit adres yolu var
16 bitlik yazmacalar 20 bit adres yolu nasıl olur (optimizasyon yaporak olur)

Biçim yazmacaları

veri iletiminde kullanıcımız yazmacaları 2'ye ayrıralım

Segment yazmacı



16 bit

offset yazmacı



16 bit

Segment grubundan kullanıcınız tarafından X10H ile karşılık gelen 10 hexadecimal ile çarpmak sonuna bir 0 koymak demek

Segment X10H + offset

ve bu şekilde 20 bitlik değer oluşturuyor

Segment grubu yazınalar
ile ilişkilendiriliyor

SS
CS
{ DS
ES }

Herhangi bir segment yazınası sadece belirli tip offset yazınaları ile colaydır

DS → Veriye ilişkin
segment

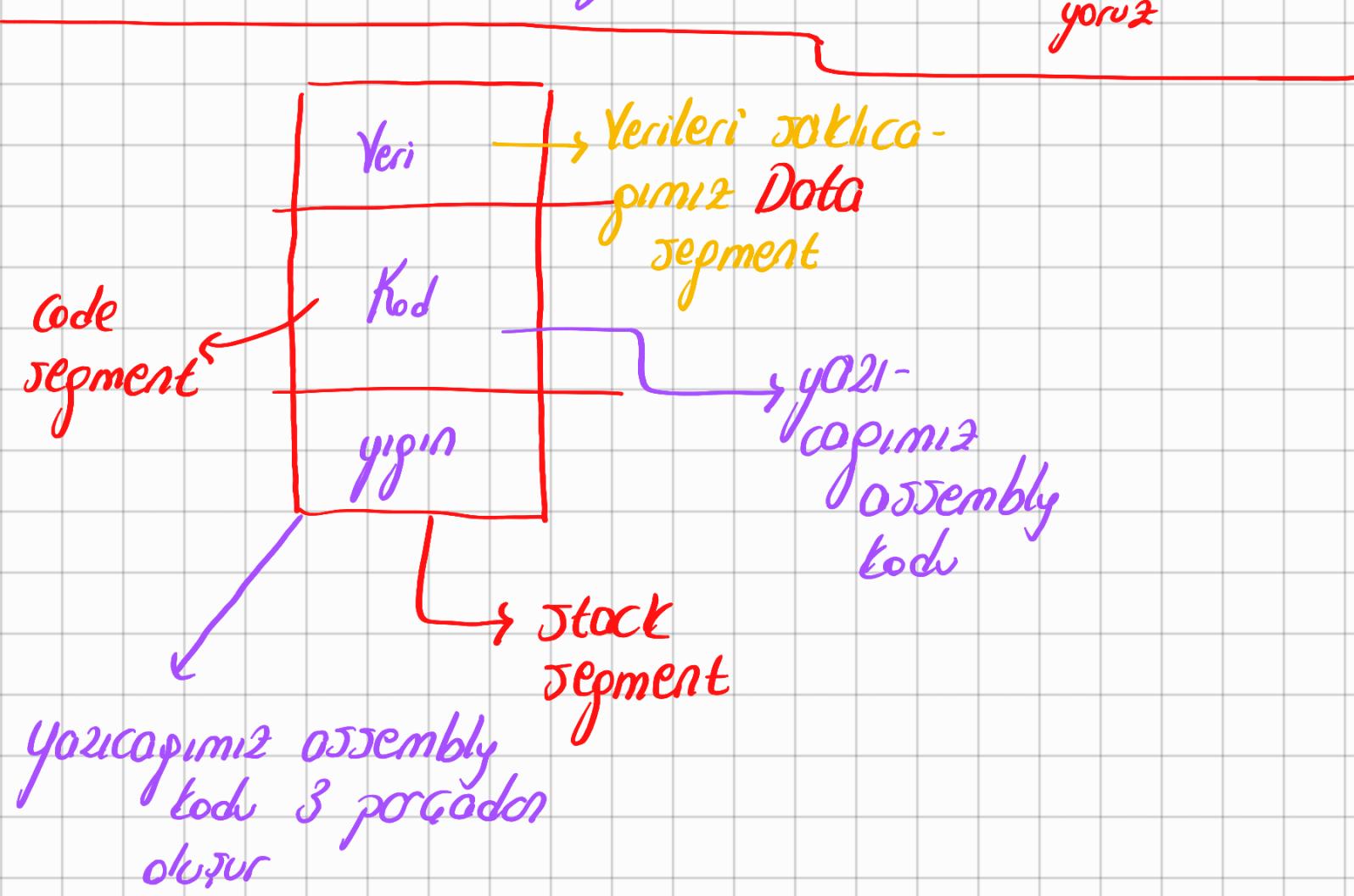
Bunlar sadece BX, SI ve DI ile colaydır

Veri kesiminde
değerinin
adresini bu
şekilde oluştur
biliriz

CS → IP (Bir sonraki adım
göstertilecek olan
komutun adresini
sayılır)

SS → BP, SP ile colaydır
(Stack Segment)
iyi işlemelerinde
yığının en üst adresini gösterir)

Bunlarla 20
bitlik adres
değerlerini
oluşturabiliriz



JJ
 CJ
 DJ
EJ
 IP

(instruction pointer)

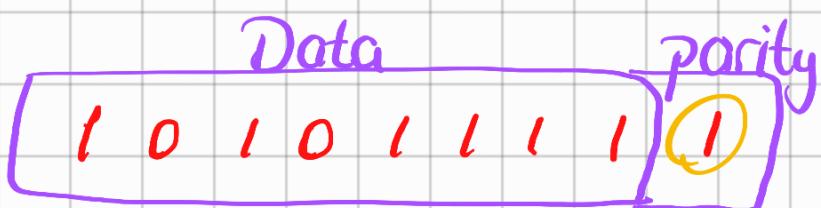
temelde bir offset gösteren CJ ile birlikte kullanılır
 (Bu itidi birlikte sıradaki yürütülecek olan kodun instructionun adresini taşıyor)

IP degeri otomatik olarak mikroişlemci tarafından güncellenir

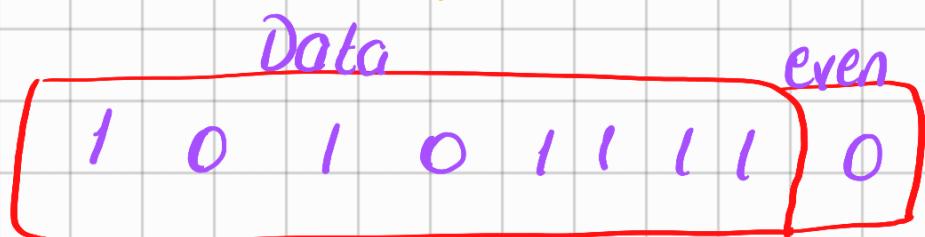
Yürüttüğümüz komut 3 byte ise o komut yürütüldükten sonra IP degeri 3 artar

Carry flag \rightarrow CF \rightarrow İşaretçil işlemlerde toplama yarasa carry flag 1 değerini olur

PF \rightarrow parity flag \rightarrow odd
 \searrow even



Eğer odd parity kullanıyor
Sadece parity bitiyle
beraber 1'lerin
sayısı tek olmalı
o zaman 1 gizdir



Even de ise çift olmalı
Bu seri haberleşmede kullanılır

AF : Auxiliary flag

AF 4bit

AF (elde olup
olmadığını



8086 natively olarak 8 bit ve 16 bitlik işlemleri destekliyor
bu ne demek yani 8 bitlik çarpma yapabilir bir de 16 bitlik
çarpma yapabilir

Auxiliary flag 4 bitlik veri grupları arasında herhangi
bir işlemin sonucunda elde tozması var mı yok
mu onu kontrol ediyor

ZF : zero flag \rightarrow işlem sonucu 0 ise
1 olur

SF : sign flag \rightarrow 8 bit ve 16 bitlik değerin en
yüksek ondalı ubiti direkt
sign flop oluşturur

TF : trap flag \rightarrow Debug omacıyla kullanılır
 $TF=1$ i̇se her bir instruction
dan sonra belirli tipte bir
kesme olur ben kesme içeriğine
gidiip yazmaları ekranla görs-
terebilirim

IF : interrupt flag $IF=1$ i̇se bu donanım JOL kesmeler
possible edilmesi olur

Disaraktaki donanım ile aracın
bize şenleme teşviki istekleri
değerlendirilebilir

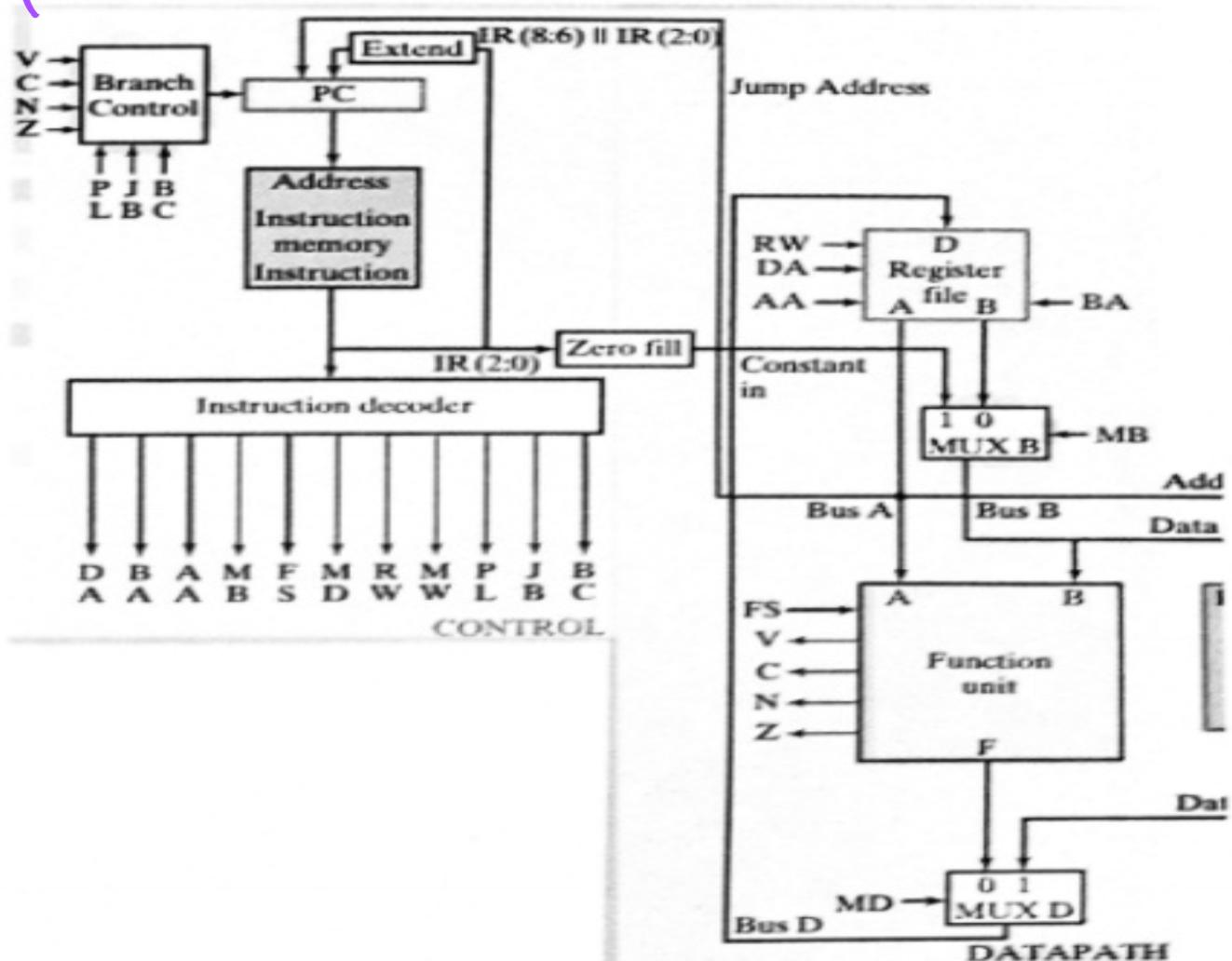
DF - Direction flop

Katar işlemlerinde kullanılır. Katar işlemi yükselt adresten doğrudır. Adrese doğru mu yoksa aşağı adresenin yükselt adresi doğru mu. Azaltı azaltı reya orttura orttura mı yapılıcak onu belirleyen yapısıdır.

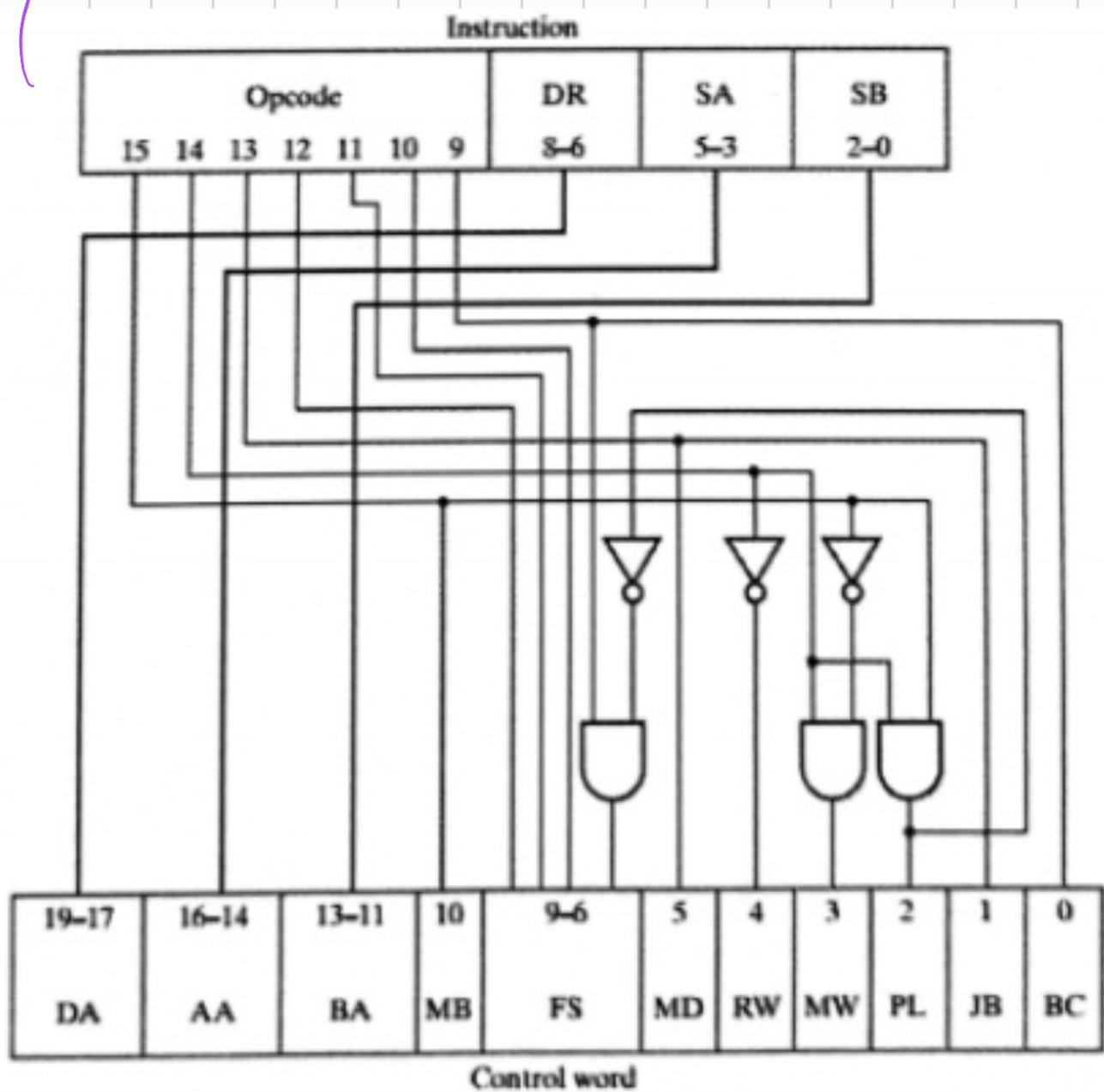
OF → overflow flag

İşarelli sayı işlemlerindeki fazla tasmayı overflow bayrağı tutar

Boşit bir işlemci ia yapıası



*Başit bir işlemciye iliskin komut kodlaşım -
leme derresi*



Protected Mod Hafıza Odresleme

- ↳imb'tan daha geniş hafızaayı adreslemede kullanılır
- ↳ Segment adresi bir tablonun (descriptor table) bir satırını gösterir
- ↳ Bu tablonun her bir pozisyonunda kullanım için ayrılmış segmentin boyu yeri, erişim izinleri yazılır
- ↳ tablolardan global ve yerel versiyonları vardır
- ↳ global descriptor tablosunda tüm programlar için kullanılabilecek segment bilgisi yer alır
- ↳ yerel descriptor tablosunda ise uygulanmaya özel segment bilgisi yer alır

FK Nokut nasıl değerlendirilir

Real Mod hafıza odresleme

Real modda segment adres ve offset adreslerinin birleşimile hafızada istenen alana erişilir

Bir segment deperi 64KB'lik olanı gösterir (Neden?)

offset deperi 64 KB'lik olan içinde bir yeri gösterir

ÖR → segment deperi 1000H, offset deperi 2000H ise mikroişlemcide erişilen fiziki adres ne olur?

$$1000H \times 10H + 2000H \rightarrow 12000H \text{ (1000H; 2000H)}$$

Real Mod Hafıza Odresleme

↳ Yarışılın segment ve offset yazarları

↳ program hafızasında erişimde CS:IP birlikte kullanılır

↳ yığın (stack) erişiminde SS:SP veya DS:BP kullanılır

↳ Veri erişiminde DS:BX, DS:DI, DS:SI kullanılır

↳ String işlemlerinde ES:DI ile kullanılır

↳ program hafızasında → CS:IP

↳ yığın erişiminde → SS:SP SS:BP

↳ String işlemlerinde → ES:DI

Not → DS, ES → Bunlar sadece BX, SI ve DI ile kullanılır.

Real - Protected Mod

Real Mod hafıza adresleme

8086 real mode hafıza adresleme yapar

Real Modda sadece 1 MB olan adreslenebilir

8086 hafıza uzayı 1 MB (20 adres ucu \rightarrow 1MB)

\hookrightarrow tüm bilgisayarlar aynı anda real mode açılır

\hookrightarrow 8086 real mode çalışır

\hookrightarrow real mod \rightarrow yazıcamız herhangi bir program
8086'nın 1 MB kapasitesinin
deli hafıza uzayının herhangi
bir noktasına erişebilir

8086 Komut yapısı

{ Etiket : } Mnemonic

{ } Operand1, { } Operand2 { }

{ ; Açıklamalar }

8086 Komut yapısı

80x86'da komutlar:

Operand olmayan
tek operand olan
iki operand olan
üç operand (80x86 ailesi öst servislerinde)

Operand1 \leftarrow Operand1 işlem operand2

Operatörler aynı tipte olmalı veya uygun türde komut dönüşümü yapılmalıdır.

genel olarak işlemler sağdan sola doğru tanımlı

ADD Ax, Bx



Operand1

Operand2 :

Ax \leftarrow

Ax + Bx

\hookrightarrow Depişme 2



\hookrightarrow Ax ikisinin toplamı olacak şekilde güncellenecektir

ADD r/m

ADD → toplama

INC → increment

Mov → Veri taşıma

Her bir komuta konsi MNEMONIC gösterimimi^z olucat

0 operand

1 operand

2 operand

8086 memori-
sinde bunlar
olur

alon ko-
mutlar
var

{label} : MNEMONIC
(etiket)

{operand1} {ope-
rand2}

Bu parantez
opsiyonel demek

Herhangi bir daturda nokteli virgülden sonra
yazicipm hersey yorum olarak degerlendirilir
(;)

AI :

ADD AX [J1]

INC J1

etiket JMP AI

AI : ADD AX, [SI] ; AX üzerinde toplama

ADD BX, AX → Bu kod colisir
16 bit 16 bit

ADD AL, BX → type mismatch
8 16 hatalı verir

Vart DB 35H
Depistkenin
İsmi
Depistkenin
degeri
Degistkenin
tipi }
chor Vart → 35H

Mov AX, Vart → type mismatch
16 bit 8 bit
yazmag

[DI], [SI] [BX + SI] [SI + DI]

$[BX + DI]$

$[SI + DI] [BX + SI + DI + 100H]$

~~[AX]~~ ~~[DX]~~

$[BX]$ → ilgili kesim yazmanın BX kadar ötedeki veriyi al demek

$[100H]$ ilgili kesim yazmanın gösterdiği odresten 100H hexa decimal byte daha ötedeki veri demek

Komutlardaki Kısıtlamalar

acc : 'Okunabilir' → 8 bitlik yerlerde AL
16 bitlik yerlerde AX olur

reg : 8/16 bitlik yazma → 8 bitlik (AL, BL, BH...)
16 bitlik (CS, DS, SS, ES, DS, SS, CS, ES)
↳ Herhangi bir yazmadan bahseder

regb : 8 bitlik yazma → AL, AH, BL, BH...

regw ; 16 bitlik yazmaç \rightarrow AX, BX, CX, DX, CS, DS -

seg ; Segment (kesim) yazmaç \rightarrow CS, SS, DS, FS

mem ; bellek adresinin içeriği

iData ; 8/16 bitlik sabit değer immediate

disp8 / disp16 : $[-128 \dots 0 \dots 127] / [32768 \dots 0 \dots 32767]$ data

dest / scr ; hedef / kaynak

igerde ofset olur

\nearrow \hookrightarrow Bir segment yazmacıyla beraber kullanılır
[]

\hookrightarrow Küseli parantez hafıza erişimine karşılık gelir

\hookrightarrow Veriyle ilişkiliyse

DS
yazınla ilişkiliyse

SS ofset kodu
iceride demet

, mov AX [100H]) , \rightarrow mem işlemi

`mov AX, 100H` / ikisi farklı işlemleri → immediate işlemi

$AX \leftarrow 100H$ Burda Ax'e 100H atonur

$AX \leftarrow [100H]$ Burda Ax'e 100H adresinin içeriği atonur

$i = a [100H]$

$i = 100H$

AI ! ADD Ax [δ_1]

Displacement

INC δ_1

adres1

JMP AI (-10)

10 byte perideki
komuto git
etiket onde olucydi
20 byte diiyip 20
byte ilerideli komu-
ta git diyebiliriz

adres2

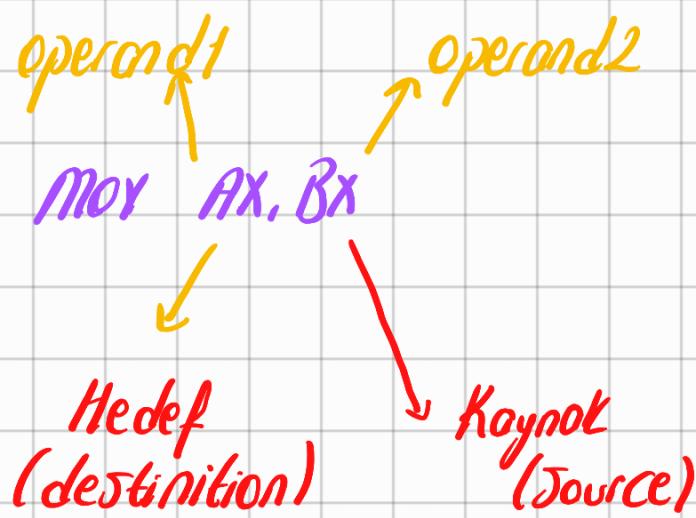
Aralarindaki fark
yazdirir

near → (8 bit işaretli sayıya dair -128, 127)

far \rightarrow (16 bit işaretli soyoyo işaret)

-32768 : 32767)

Displacement



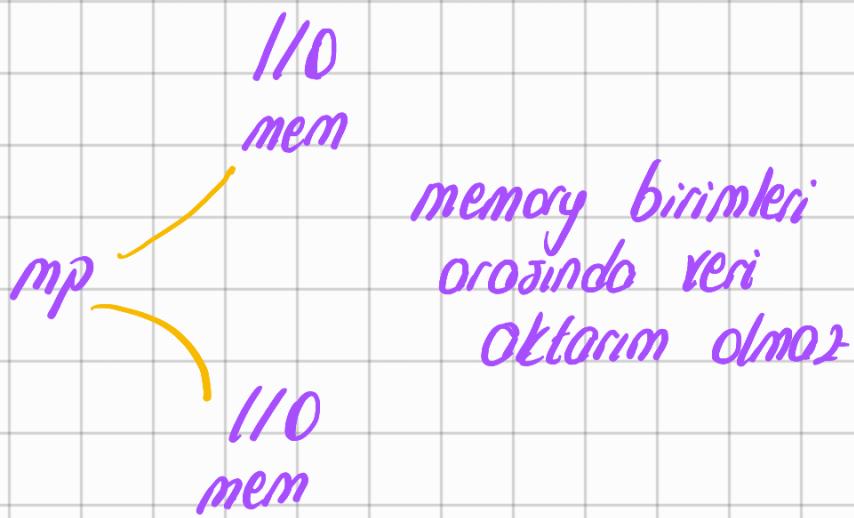
$AX \leftarrow BX$
Hedef Kaynak

8086'da 3. operando geret yok

Veri ottorun Komutları

Mov
LEA
LDS
LES
XCHG
XLAT / XLATB

Beyrolar etkilenmez



Veri aktarım komutları

Mov ; move data

İki tane parametre olur

Mov ; dest src

Jource'deki içeriği destination'a aktarır

dest ← src

dest = mem ve src = mem olamaz

dest = reg ve src = reg olamaz

dest = reg ve src = idata olamaz

dest ve src aynı tipte olmalıdır

mov [100H] [200H]

[offset]

memory adresine denk gelir
yani ben memory ile
ilgili bir işlem yapmak
tam tözeli poronter2
ullanicom

polling } 3 tone
interrupt } veri
DMA } oktarım
 } tipi

↳ Mikroişlemci
derrede değil

Mov ~~SS, CS~~

Valid değil
(gecerli değil)

Mov ~~AL, CS~~

→ AL 8 bitlik bu
yüzden kullanı-
maya iz

Mov ~~SS, AL~~

Mov AX, CS ✓

Mov SS, AX

~~mov DS~~X~~ 100H~~

mov AX, 100H
mov DS, AX ✓

~~mov AX, DX~~ → 16 bit
8 bit → type mismatch
hataşı oluruz

Yarlı DB 12 } Veri kesiminde
 } karşılaştırınız

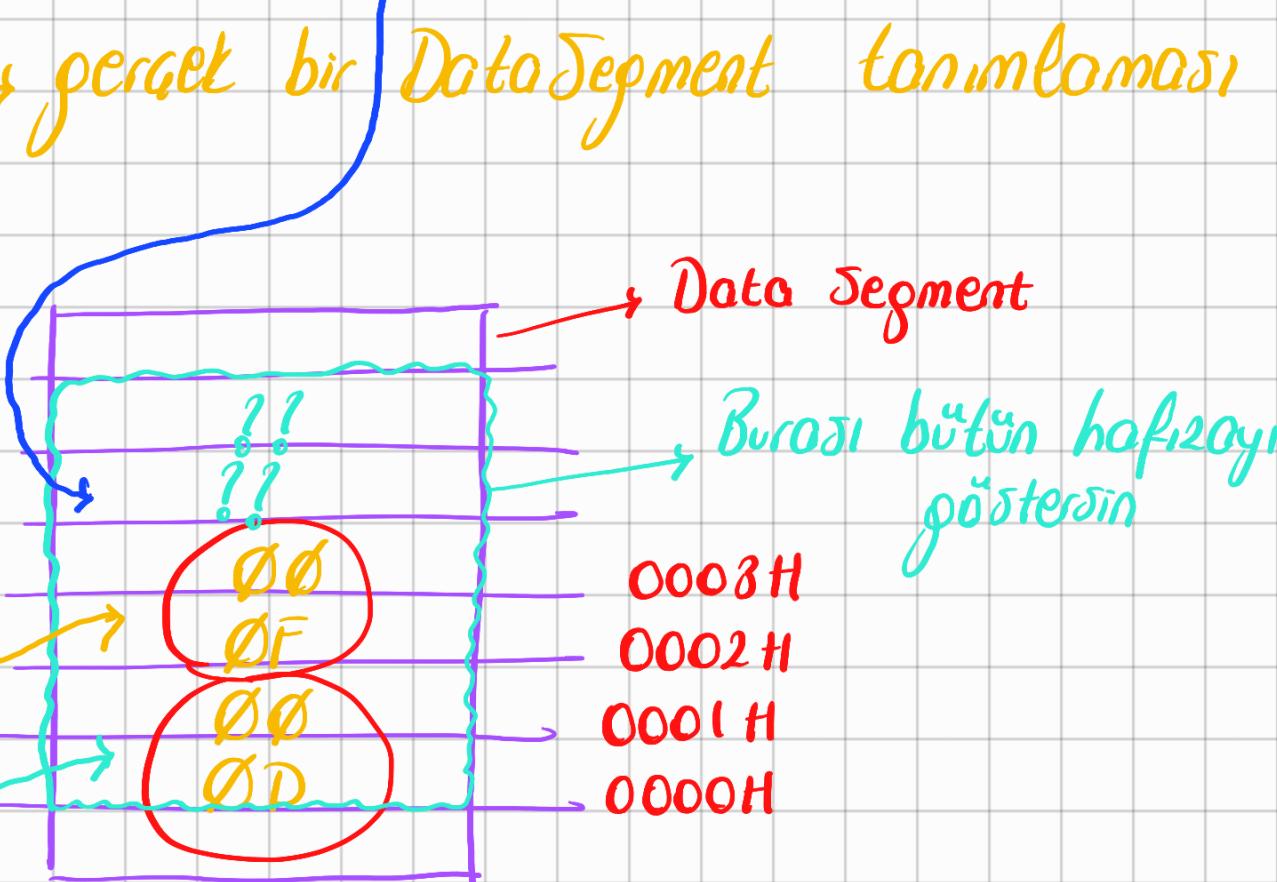
mov BX Yarlı } Kod
 16 bit 8bit } (type mismatch
 hataşı oluruz)

Depisten tip uyumu-
nda dikkat etmek
gerekir

DATASEG JEGMENT PARA PUBLIC 'DATA'

A Sayısı DW 13 → (Bu asagidaki data segmente
 nodil yazılır)
 B Sayısı DW 15
 GCD DW ?
 DENEME DW -1
 DATASG ENDS

Benim data seg-
 ment deperim
 ldiğinden
 belirleniyor



01001 H
01000 H

fiziki 20 bit

DS : 0100H

01000

0001

t
01001 H

0007H	FF	{	Deneme
0006H	FF	}	
0005H	!!	}	6D
0004H	!!	}	
0008H	ØØ	{	B soyisi
0002H	ØF	}	
0001H	ØØ	{	→ A soyisi
0000H	ØØ	}	

LEA BX , A soyisi ; BX = ?

LEA SI , B soyisi ; SI = ?

LEA DI , Deneme ; DI = ?

BX = 0000H

LEA degiskenin tamamlı oldugu adreste segment icerisinde baslangicdan itibaren ne kadar uzaktan olduguunu bulur

SI → 0002H

DI \rightarrow 0006H

Veri Altmı̄m Komutları

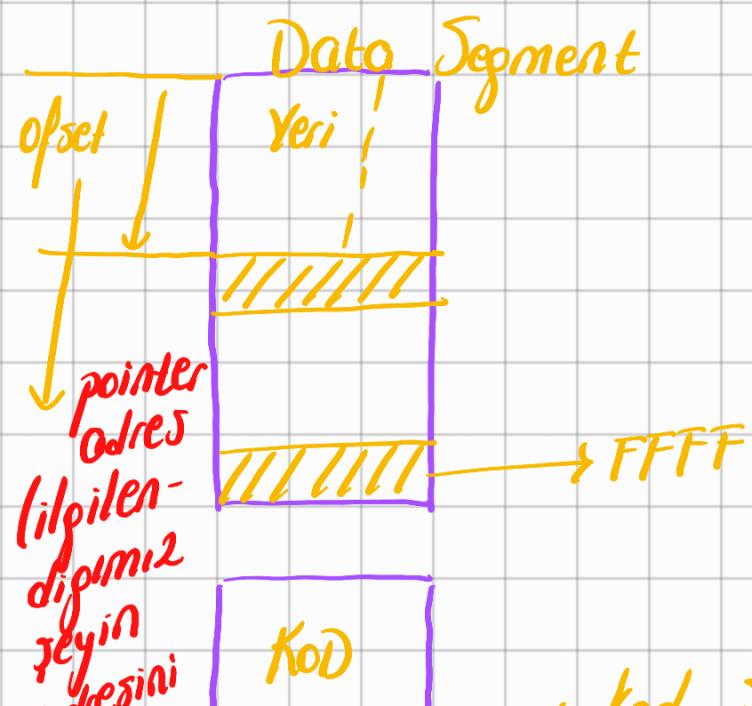
LEA \rightarrow Load effective address

LEA \rightarrow regw, mem \rightarrow adres, deposten
 \rightarrow 16 bitlik
yazmag

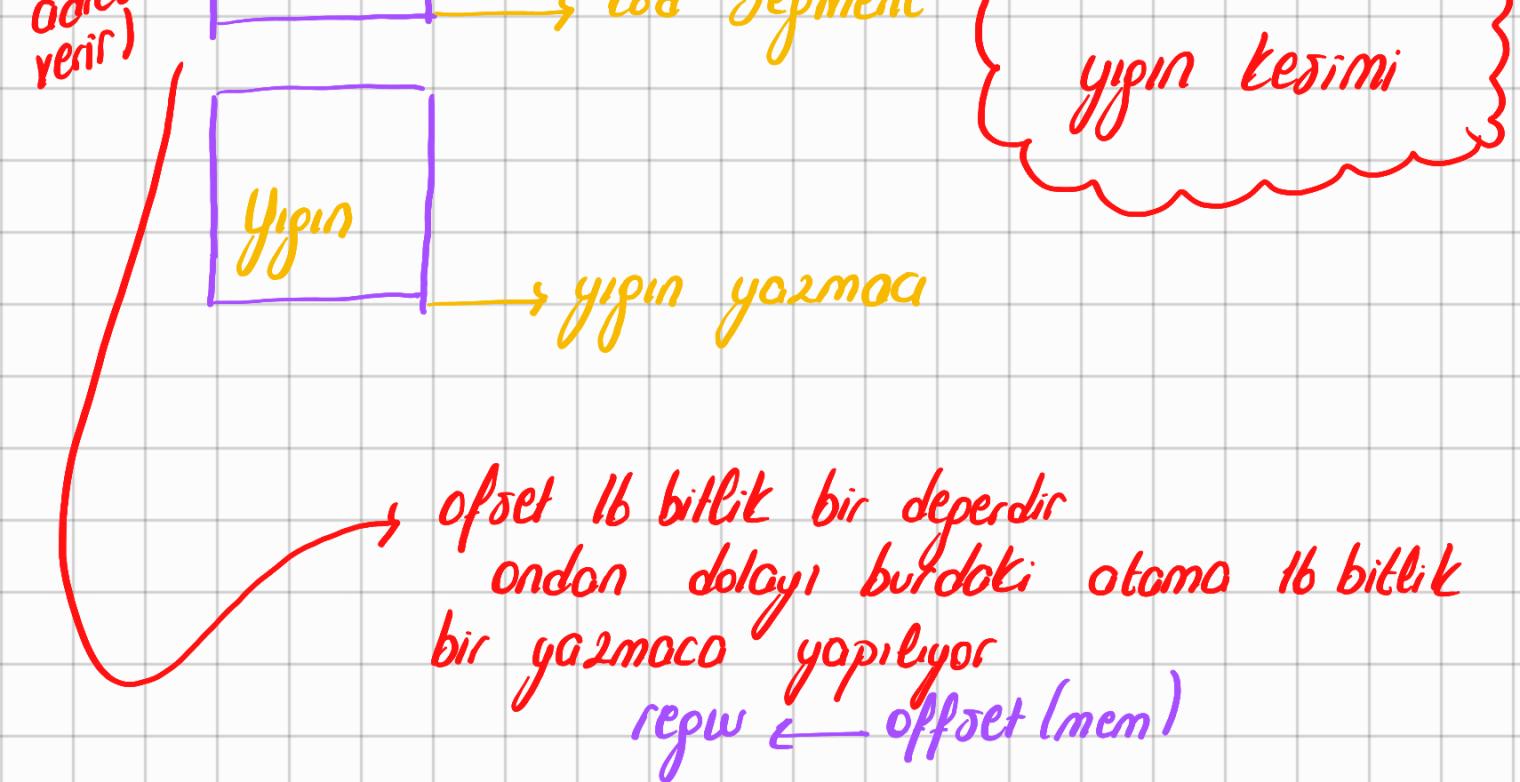
regw : \leftarrow offset(mem)

1 lane mem içeriği olsaydı bu adres olabilecegi gibi depoisten de olabilir

parametre olarak vermiş olduğumuz depoistenin
kendi segmentindeki offset değerini hesapla-
yıp bu verdipimiz 1. operanda oluyor



programcısının
3 segmente
ihtiyacı var
Yeri kesimi
Kod kesimi



offset 16 bitlik bir değerdir
ondan dolayı birdeki otomat 16 bitlik
bir yazmacı yapılıyor
 $\text{repw } \leftarrow \text{offset(mem)}$

Dolayısıyla LEA ile bir depoşkenin adresini
oldığımız zaman bunu olacağınıza
depoşkenin BX, SI, DI gibi koseeli paron-
teri içine yazılabilir

LEA ile oldığımız değer bir adres olduğu
için biz tetrafı buna ulaşmak için
koseeli parantez içinde kullanıcaz

Veri aktarım Komutları

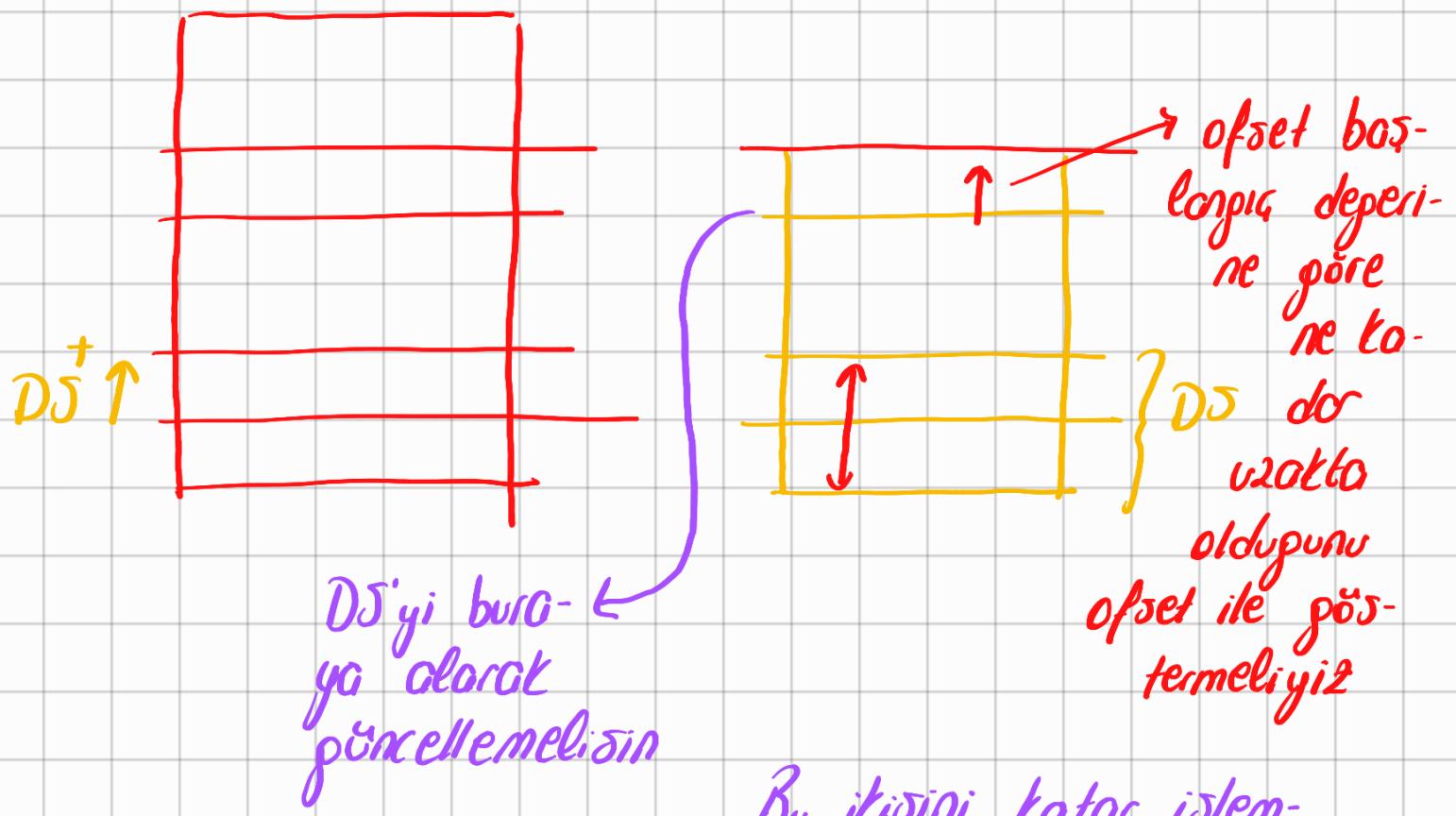
LDS: load data segment register

LDS! repw,mem

$\text{repw } \leftarrow [\text{mem}]$

$DS \leftarrow [mem+2]$

Bizim kullanmak istedipimiz veri bloğu bir veri kesimindiyse döto segment deperini depistirip uzaktaki segmenti işaret etmemiz lazım



Bu işinizi kotor islemelerinde bir kaynaktan diperine aktarm yapmak istersen olur

Veri Altınlık Komutları

LES : load extra segment register

LES regw, mem

$\text{regw} \leftarrow [\text{mem}]$

$\bar{E}\bar{S} \leftarrow [\text{mem}+2]$

Segment range'inde olmayan Gök uzaktaki veriler için
LDS kullanılır

$\text{regw} \leftarrow [\text{mem}] \rightarrow$ Yerini oldugumuz adres
degerinin içeriğini gorma
otor

$\text{DS} \leftarrow [\text{mem}+2] \rightarrow$ Depolunden sonra gelin
wordu 'de DS atar

Yeri { Uzak - OFFF Dw 1234H
Uzak - Seg Dw ABCDH

|
|
|
|

LDS BX, uzak OFF ; BX \leftarrow 1234H
DJ \leftarrow ABCDH

moy AL [BX] \rightarrow ABCDH ; BX

Mov AL[Bx] → ilgili data segmentin Bx offsetine git yani data segmentin 0 offsetine git AL oldugu icin 1 byte'lik işlem yap



Her bir hafıza
gözü 1 byte
olar

Mov AH,[BX+1]

→ 0001H

AH ←

Mov AL,[BX]

Mov DX,[DI]

→ Burda Data
segmentin DI'ina
ulaşıyoruz

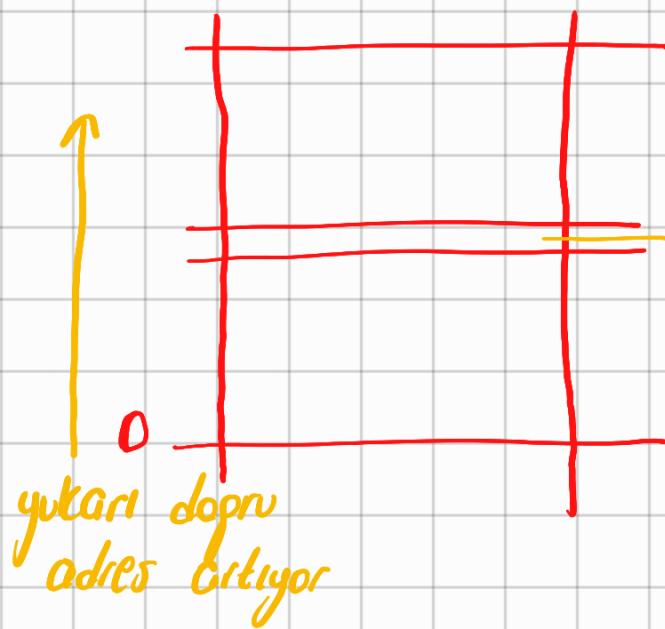
16bit
16bA

DS:DI

0006H'tan itibaren 16 bit
git demek oluyor bu

$Dx \leftarrow FFFFH$

LEA \rightarrow 2 tane operand aliyor



ilgili depisten kendi
segmenti içerisinde

Bu segment içinde
bu depisten seg-
mentin baslangicin-
dan itibaren ne
kadar uzaktayaa
o deperi verir

LEA bir more işlemi
değil

\rightarrow Baslangictan
itibaren ne ka-
dar ötede
olduguunu bulur

İçerik: Seçimli Konular

Mov reg, idata

✓ mov Ax, Bx

Mov mem, idata

✓ mov DL, CL

Mov reg, reg

X mov AH, CS

8 16

Mov reg, mem

Mov mem, reg

Mov SI rep
16 bit 8 bit
16 bit mi
16 bit mi

Mov Sreg, reg

16 bit tello-
nur 8 bit
tello namoyiz

Mov Sreg, mem

✓ mov SI, AX

Mov reg, Sreg

Mov mem, Sreg

Mov SI, CS

Mov BX [250H]

Mov CS, SI

Registers

Segment
yazmaa

16 bitlik
islem

252 H

34H

251 H

12H

250 H

ABH

ileri yönde ve
peri yönde otomotiv
globilir

$Bx \rightarrow 12ABH$

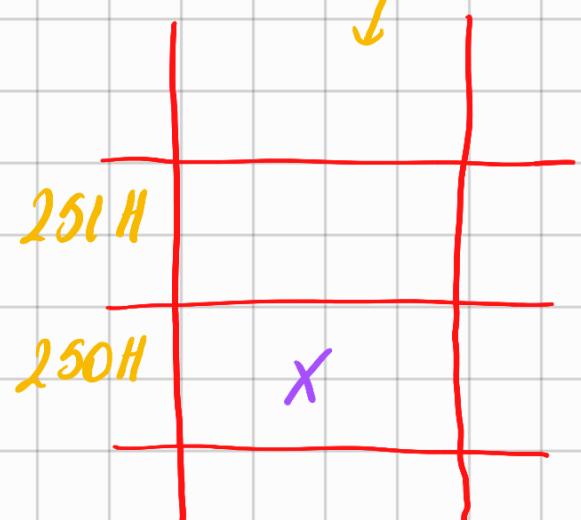
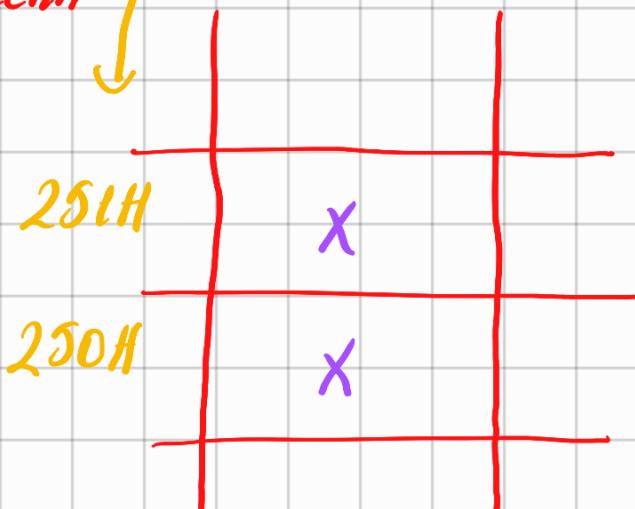
$mov AL [250H]$

$AL \leftarrow ABH$

$mov [250H], BX$

$mov [250H], AL$

Bu komut
golisi 00
251H
250H
güncellenir



Bu komut
golisi 00
250H güncel-
lenir

$mov Jreg, mem$

$mov DS, [20H] \rightarrow 16$ bitlik
işlem

Mov mem, reg

mov [20H] DS 16 bitlik işlem

Veri Atılarım Komutları

LEA : load effective address

LEA regw, mem

regw ← offset(mem)

Mov CH, 300

Mov AX, 1234H

Mov BL, 'A'

Mov DX, -1

DX ← FFFFH
(-1'in two's)

Yolda olup
olmadıklarına
bak

300 8 bite düşmediyi
için uygun bir otomo

Complement kırıldı)

→ BL ← 65

AX ← 1234H

Mov AX, 1 ← AX ← 0001H 16 bit

Mov AL, 1 j AL ← 01H 8 bit

değil çünkü 8 bit
0-255 arası değer
olsın

Mov BL, 100H

0001 0000 0000

Bu do 8 biti asıyo bu
do uygun bir kod değil

Mov [100H], 12H

)

0012H
12H

Mov WORD PTR

Mov BYTE PTR

hangisi olduğunu
bilmiyoruz
byte mi word mu
oldığını bilmiyoruz

8 bitlik mi 16 bitlik
mi bilmiyoruz

Bunları toplayıp işlem tipini

16 bit

belirtmemiz gerekiyor

1234H

word için

101H	00H
100H	12H

101H	12H
100H	34H

Byte için

101H	FF
100H	12H

Bir hafıza
gözünün kapasiti
lesi 1 byte
yani 8 bit

Veri aktarım komutları

XCHG : exchange

Swap işlemlerinde
kullanılır

XCHG dest, src

XCHG AX, BX ✓

dest \leftarrow src

XCHG AL, BX X

XCHG reg, reg

XCHG reg, mem

XCHG mem, reg

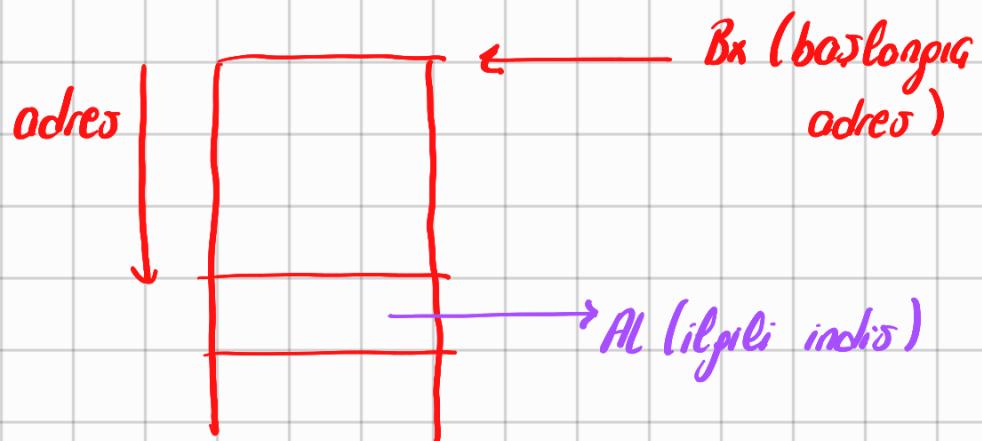
Veri Aktarım Komutları

XLAT / XLATB : translate byte (XLAT = XLATB)

Dogrudan operandı olmayan bir komut AL gizli operand

AL \leftarrow DS:[BX+AL]

DS:[BX] adresindeki tablonun AL numaralı indisindeki değeri AL yorma ve kopyalar
AL, BX



XLAT \rightarrow operand
olmaz

look of table'dan veri olmaz için
tıklanıp pmi2 komut

Aritmetik Komutlar

ADD toplama

ADC etdeli toplama (add with carry)

SUB çıkartma 2's complement

SBB etdeli çıkartma (sub with borrow)

INC t1

DEC -1

NEG sayıının negatifini hesaplar two's complementi (-)

CMP sonucu SUB ile aynı ama bunda işlem sonucu
sıklanmaz

MUL (içaretsız) çarpma

IMUL (içaretli) çarpma

DIV (içaretsız) bölme

IDIV (içaretli) bölme

Sub işlem sonucu saklanır Bayraklar güncellenir

Cmp işlem sonucu saklanmaz Bayraklar güncellenir

CF toplama işlemlerinde Carry algoritma işlemlerinde borrow olur

ADD Ax Bx

$$Ax \leftarrow Ax + Bx$$

ADC Ax ,Bx

$$Ax \leftarrow Ax + Bx + \boxed{CF}$$

1 bit

Sub Ax ,Bx

$$Ax + (-Bx)$$

Carry flag 1 bitlik 1 değer

Aritmetik Komutları

ADD ;addition

ADD dest ,src

$\text{dest} \leftarrow \text{dest} + \text{src}$

ADD reg, idata

ADD mem, idata ; PTR gerectli

ADD reg, reg

ADD reg, mem

ADD mem, reg

LEA Bx, Var3

LEA SI, Var1

LEA DI, Var2

Mov Ax [SI]; Ax \leftarrow ABCDH

↳ Mov Dx,[SI+2]; Dx \leftarrow Var1 yükseli wordü

ADD Ax,[DI]; Ax \leftarrow ABCDH + 1234H ; CF ?

↳ en düşük onluk wordlere dışarıdan bir ekle gelmez o yüzden ADC değil ADD ile yaptık

Aritmetik Komutları

ADC ; add with carry

ADC dest ,src

dest \leftarrow dest + src + CF

ADC reg ,idata

ADC mem ,idata ; PTR gerekli

ADC reg ,reg

ADC reg ,mem

ADC mem ,reg

ADC Dx , [DI+2] ; Dx \leftarrow CF

+ Var1
yüksek wordü

+ Var2
yüksek wordü

Mov [Bx], Ax
Mov [Bx+2], Dx

} Burda bayraklar
etkilenmez
korunur

ADC BYTE PTR [Bx+4], 0

8086 8 bit ve 16 bitlik işlemleri destekler

Assembly büyük ve küçük harflere duyarlı değil

ADD AX, 1

ADD BL, 300

Word
byte

ADD [100H], 12H

ADD word PTR [100H] 12H

ADD Bl, 300 [100H] 12H

ADD byte PTR [100H] 12H

ADD Dx, Ax

ADD [100H], Ax, word

8086 mimarisinde 8 bit ve 16 bitlik toplama yapabilir

32 bitlik işlem yapmak isterseniz ne yapılabilir

32 bitlik bir toplama yapıcısı olacak

Var1 DD 6789ABC0DH } 4 byte Define double word DD

Var2 DD ABCD1234H } 4 byte Define quad word DQ

Var3 DQ Ø

32 bit

+ 64 bit

32 bitlik var1 ve Var2 degistirkenini toplayarak 64 bitlik sonucu var3 degistirkenine atayarak belli et

Var1	CDH	DI
	ABH	adres artar
	89H	
	67H	DI
Var2	34H	
	12H	
	CDH	
Var3	A3H	Bx

	00	
	00	
	00	
	00	
	00	
	00	
	00	
	00	

Aritmetik Komutlar

Sub : subtraction \rightarrow Carry re borrow'u dikkate almas

Sub : dest , src

$dest \leftarrow dest - src$ \rightarrow yaptigi islemler two's complement aritmetikte gerekli sayilar

Sub mem, iobto ; PTR gerekti

Sub reg, reg

Sub reg, mem

Sub mem, reg

Aritmetik Komutlar

SBB ; subtraction with borrow

SBB ; dest, src

dest \leftarrow dest - src - CF

Borrowu dikkate
olar

SBB reg, idata

SBB mem, idata ; PIR gereklidir

SBB reg, reg

SBB reg, mem

SBB mem, reg

Aritmetik Komutlar

INC ; increment

INC dest

dest \leftarrow dest + 1

INC reg

INC mem ; PTR geretri

→ memory üzerinde
istem yopicak-
sat PTR geretri

00H	101H
FFH	100H

INC [100+1]

INC word PTR [100+1]

01H	101H
00H	100H

CF=0

INC BYTE PTR [100+1]

00H	101H
00H	100H

00FF+1

→ 0100

CF=1

Aritmetik Komutları

Dec : decrement

Dec dest

dest \leftarrow dest - 1

Dec rep

Dec mem ; PTR gerekli

memory üzerinde işlem varsa PTR gerekli

Aritmetik Komutları

NEG : negate / two's Complement

NEG dest

dest \leftarrow 0 - dest

NEG rep

NEG mem ; PTR gerekli

Aritmetik Komutlar

CMP : compare

CMP dest, src

dest - src

Sonuç saklanmasa, bayraklar etkilenir

Komut sonrasında koşullu döllenme komutları kullanılır.

dest > src , dest ≥ src

dest = src

Koşulları oluşturmak için kullanılır

dest ≤ src , dest < src

