Assembly Notları

#dersnotu *Enes Utku Selbes*
Son Güncelleme Tarihi: 26-01-2026 00:19

FÇ alt seviye programlama ders videolarından , slaytlardan vb. kaynaklardan derlenmiştir.

İÇİNDEKİLER

PART 1: 8086 ASSEMBLY

Genel Kültür

FSB(Front Side Bus)

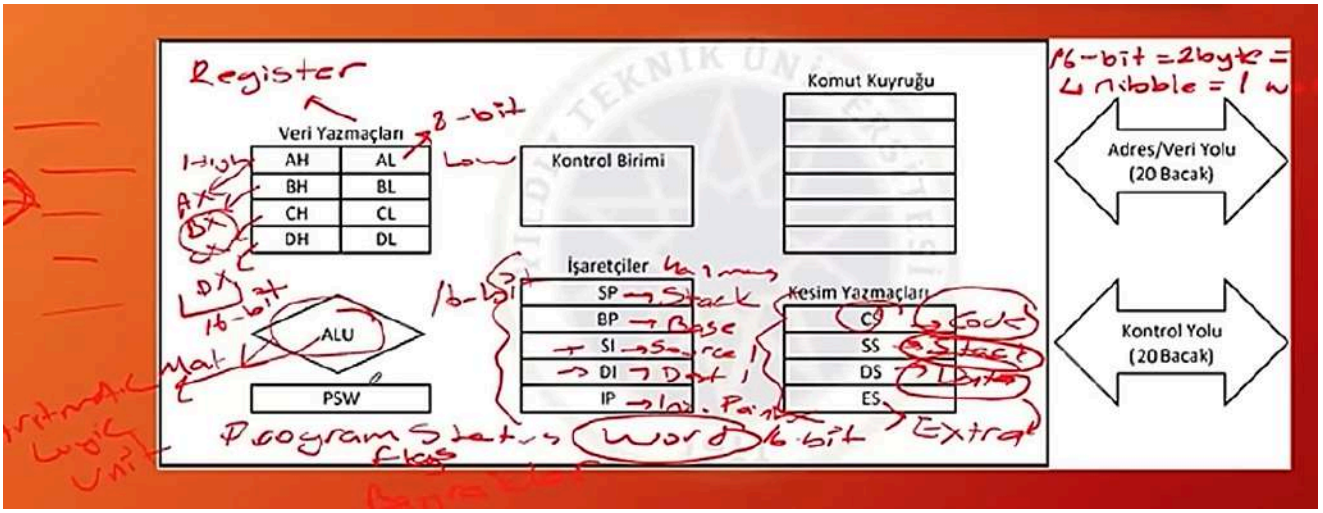
- işlemci ile anakart arasındaki bağlantı

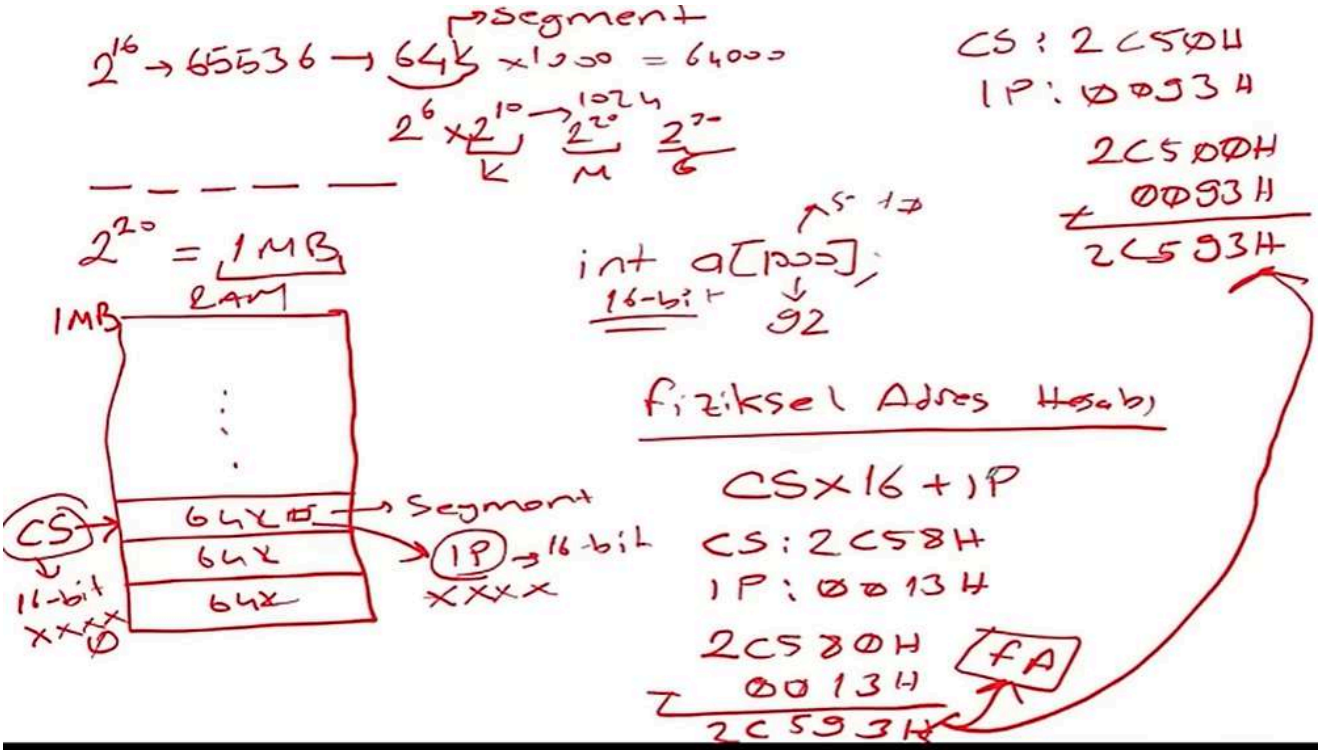
Bellek Bağlantı Yolları

- SDRAM (Synchronous Dynamic RAM)
- DDR-SRAM (Dual Data Rate SDRAM)
- DDR2-SDRAM
- DDR3-SDRAM
- DDR4-SDRAM
- RDRAM (Rambus Dynamic RAM)

- CISC
  Intel, AMD, Cyrix, IBM, TI, NexGen vb.
- RISC

80x86 Ailesi İşlemciler





Fiziksel Adres Hesabı

Fiziksel Adres = Kesim Yazmaç Değeri * 16 + İşaret Yazmaç Değeri
Örnek:
CS *16 + IP ya da CS* 10h + IP

> **Not: Hexadecimal Sayılar**
>
> - A12BH var (variable) bir değişken sayılır.
> - 0A12BH hexadecimal value
>   (her zaman MSB'i (most significant biti) harf olan hex. sayıların başına 0 koy
>   Aynı şekilde binary sayıların sonuna b koyulur 1000111b

> **Not**
>
> assembly büyük küçük harflere duyarlı değildir! (mov = MOV)

Segments/Kesimler

- Code Segment
  - CS:IP → RAM'deki fiziksel adrese ulaşıyoruz
- Stack Segment
  - SS:[SP-BP]
  - LIPO (last in first out)
- Data Segment
  - DS:[SI, DI, BX] → bunlar haricindeki registerları dizi parantezi içinde KULLANAMAZSINIZ!
    - Dizi[i] yi ifade eder
- Extra Segment
  - DS:[SI, DI, BX]

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

Bayraklar (Flags - PSW (Program Status Word))

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–	–	–	–	OF	DF	IF	TP	SF	ZF	–	AF	–	PF	–	CF

Bayrak	Görevi (Set –> 1, Clear, Reset –> 0)
CF: Carry Flag	Elde ödünç durumlarında CF set olur.
PF: Parity Flag	Even parity –> Set, Odd Parity –> Clear (Even: Çift)
AF: Auxilary CF	8 – bit işlemde 4'den 5'e; 16 – bit işlemde 8'den 9'a elde/ödünç aktarımı
ZF: Zero Flag	İşlem sonucu 0 ise Set, 1 ise Clear
SF: Sign Flag	İşlem sonucu negatif ise Set, pozitif ise Clear
TP: Trap Flag	Adım bayrağı. Set ise her komuttan sonra kesme üretilerek prog. izleniyor
IF: Interrupt Flag	Maskelenebilir (Maskable) kesmelerin kontrolü için kullanılır. 1 ise izin verilir.
DF: Direction Flag	Dizgi (String) işlemlerinde işlemin yönünü belirlemek için kullanılır.
OF: Overflow Flag	Aritmetik taşma durumunda Set, aksi durumda Clear.

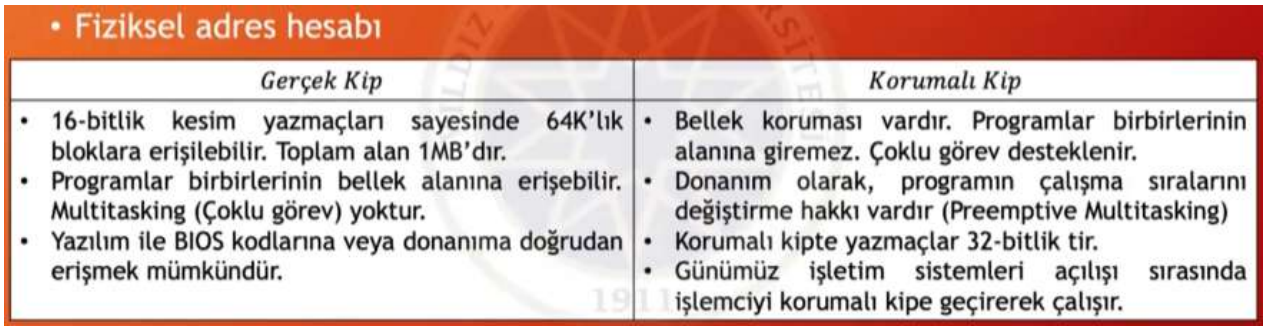
🖊 Kasıntı Bayrak Soruları İçin (Hocalar Sormayı Seviyor)

- CF 16 bitlik iki sayının toplanmasında 8. bitin taşması sırasında 1 olmaz. Auxilary Flag 1 olur.
  ⭐ OF işaretli sayılar ile yapılan işlemlerde elde edilen işaretli sonucun tanımlı alana sığmaması (taşması) durumunu belirler. Burada önemli bir kafa karışıklığı var. OF, sonuç sayısını İŞARETLİ OLARAK YORUMLAR. OF nin set edilmesi için iki pozitif sayı toplamından negatif, iki negatif sayı toplamından pozitif sayı çıkması gerekmektedir. Örneğin 8 bitlik işaretsiz sayılar olan 220 + 40 toplamında OF 0 olur (0000 0100), ancak 8 bitlik işaretsiz
- sayılar olan 100 + 100 toplamında OF 1 olmaktadır! (1011 0100)

Belleğe Erişim

İşlemcinin adresleyebileceği bellek adres bacağı ile değişir. → 20 bacak → 2^20 = 1MB

- İşlemci real modda hafıza adresleme yapar.



8086 ASSEMBLY KOMUTLARI

(komutların ne olduğunu daha iyi anlamak için instruction set dokümanını incele!)

Veri Aktarım Komutları

LEA (Load Effective Address)

LEA regw, mem
LEA SI, dizi → adresi SI ya koy

MOV (Move Data)

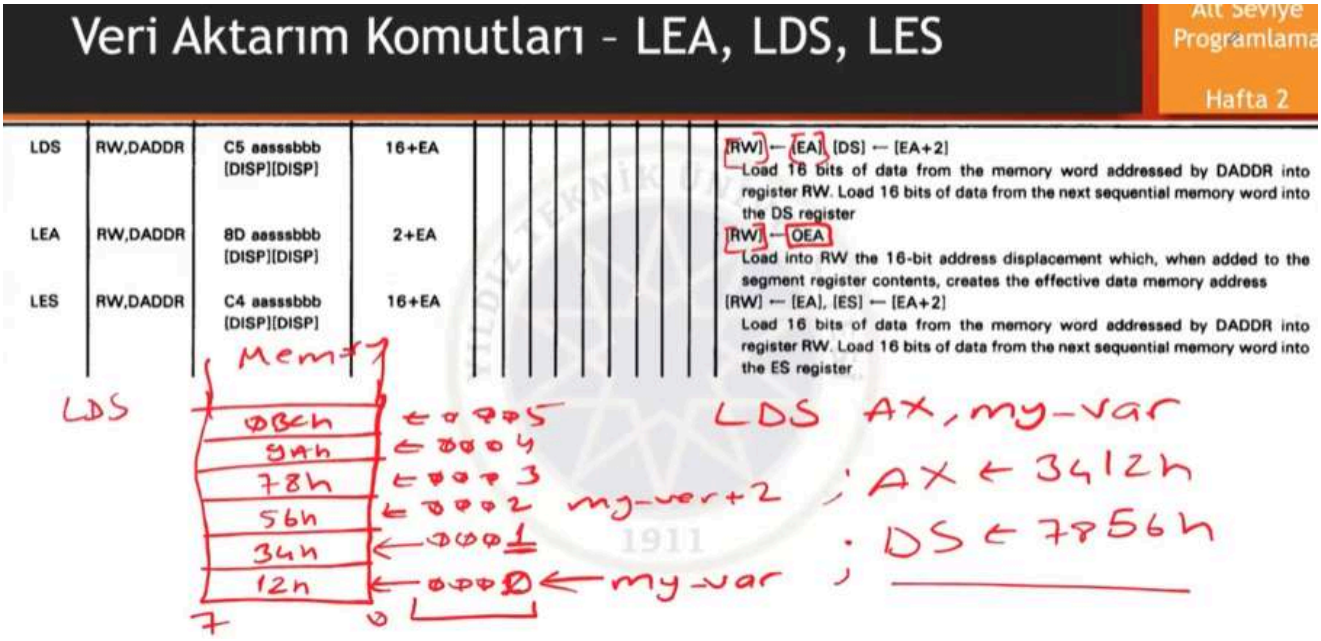
MOV dest, src
MOV SI, data → değeri SI ya koy

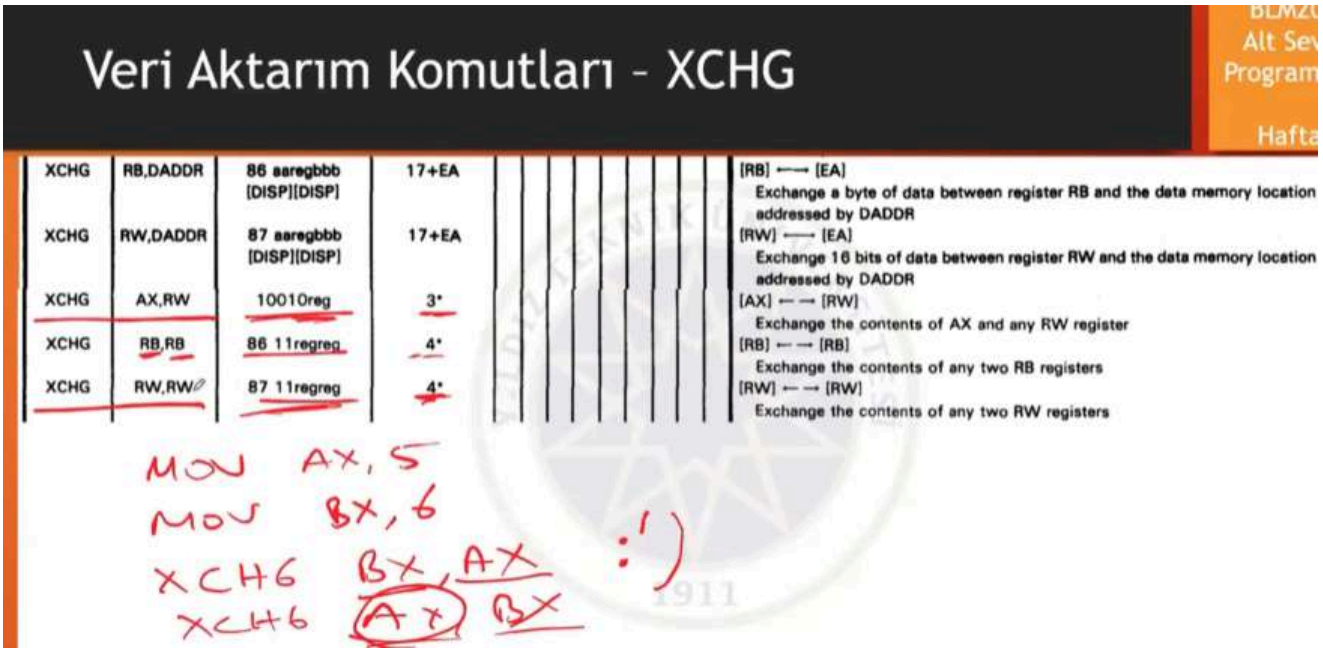
MOV ile İlgili

- ❌MOV komutunun her iki işleneni mem (mem, mem) ve sreg (sreg, sreg) olamaz.
- ❌Yine ilk işlenen sreg ise ikincisi idata da olamaz. (sreg, iata) Bunun gibi herhangi bir komutun alabileceği parametre tiplerinin uygunluğu set instructionu incelenebilir.
- MOV komutunda her iki işlenenin de aynı tipte olması son derece önemlidir. Aksi halde derleyici hata verir. Özellikle işlenenlerden birinin memory olması durumunda veri tiplerinin uyumlu olup olmadığını tespit etmek için *PTR* komutundan yararlanılır.

Veri Aktarım Komutları

⭐Veri Aktarım Komutlarının çalışması *bayrakları etkilemez*.



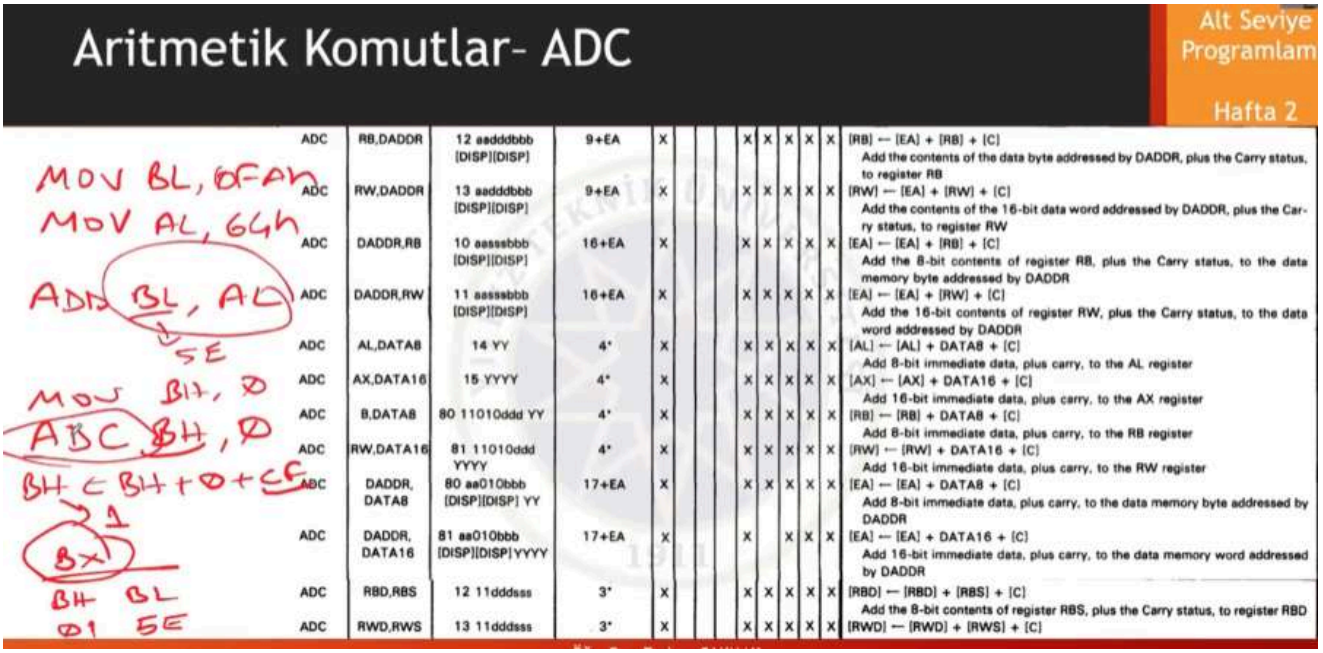


Not: Görüldüğü gibi AX üzerinden Exchange daha hızlı!
Bazı komutlar 8086 tasarımında AX üzerinden daha hızlı yapılabilecek şekilde tasarlanmıştır.

XLAT - XLATB (translate byte)

- Doğrudan operandı olmayan bir komut, AL gizli operand
- AL ← DS:[BX+AL]
- DS:[BX] adresindeki tablonun AL numaralı indisindeki değeri AL yazmacına kopyalar

Aritmetik Komutlar



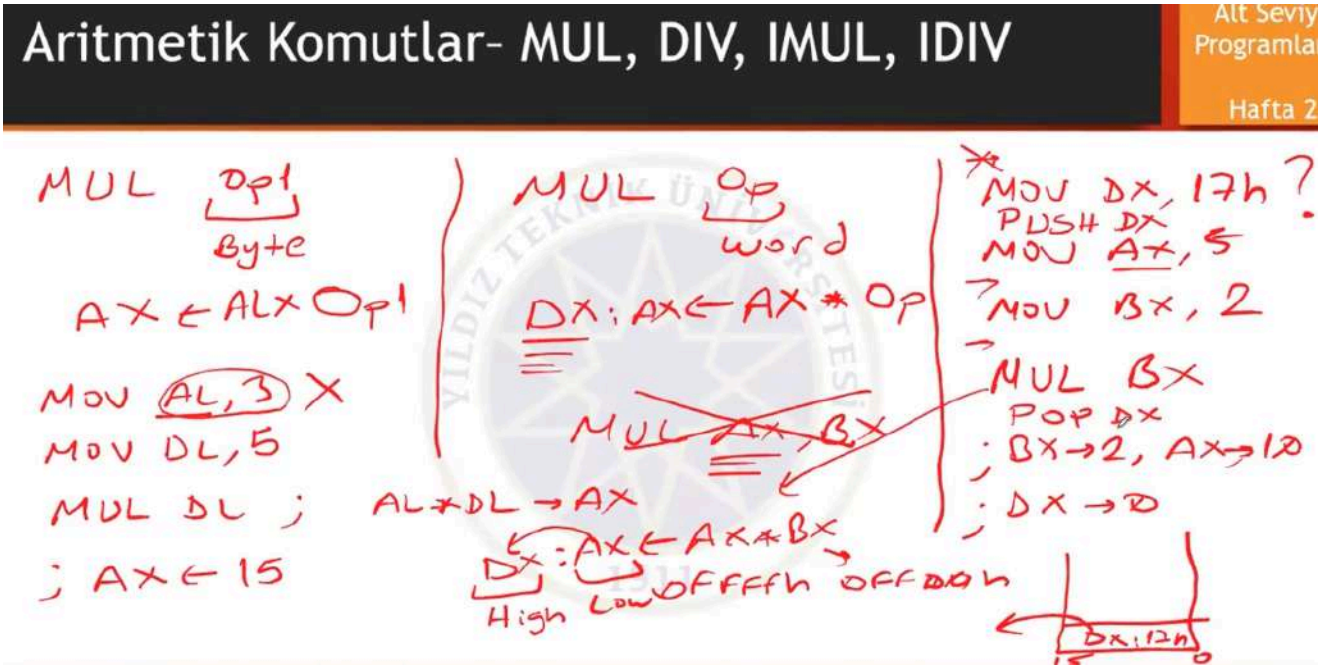
ADC	RB,DADDR	12 aadddbbb [DISP][DISP]	9+EA	X				X	X	X	X	X	[RB] ← [EA] + [RB] + [C] Add the contents of the data byte addressed by DADDR, plus the Carry status, to register RB
ADC	RW,DADDR	13 aadddbbb [DISP][DISP]	9+EA	X				X	X	X	X	X	[RW] ← [EA] + [RW] + [C] Add the contents of the 16-bit data word addressed by DADDR, plus the Carry status, to register RW
ADC	DADDR,RB	10 aassbbbb [DISP][DISP]	16+EA	X				X	X	X	X	X	[EA] ← [EA] + [RB] + [C] Add the 8-bit contents of register RB, plus the Carry status, to the data memory byte addressed by DADDR
ADC	DADDR,RW	11 aassbbbb [DISP][DISP]	16+EA	X				X	X	X	X	X	[EA] ← [EA] + [RW] + [C] Add the 16-bit contents of register RW, plus the Carry status, to the data word addressed by DADDR
ADC	AL,DATA8	14 YY	4*	X				X	X	X	X	X	[AL] ← [AL] + DATA8 + [C] Add 8-bit immediate data, plus carry, to the AL register
ADC	AX,DATA16	15 YYYY	4*	X				X	X	X	X	X	[AX] ← [AX] + DATA16 + [C] Add 16-bit immediate data, plus carry, to the AX register
ADC	B,DATA8	80 11010ddd YY	4*	X				X	X	X	X	X	[RB] ← [RB] + DATA8 + [C] Add 8-bit immediate data, plus carry, to the RB register
ADC	RW,DATA16	81 11010ddd YYYY	4*	X				X	X	X	X	X	[RW] ← [RW] + DATA16 + [C] Add 16-bit immediate data, plus carry, to the RW register
ADC	DADDR, DATA8	80 aa010bbb [DISP][DISP] YY	17+EA	X				X	X	X	X	X	[EA] ← [EA] + DATA8 + [C] Add 8-bit immediate data, plus carry, to the data memory byte addressed by DADDR
ADC	DADDR, DATA16	81 aa010bbb [DISP][DISP] YYYY	17+EA	X				X		X	X	X	[EA] ← [EA] + DATA16 + [C] Add 16-bit immediate data, plus carry, to the data memory word addressed by DADDR
ADC	RBD,RBS	12 11dddsss	3*	X				X	X	X	X	X	[RBD] ← [RBD] + [RBS] + [C] Add the 8-bit contents of register RBS, plus the Carry status, to register RBD
ADC	RWD,RWS	13 11dddsss	3*	X				X	X	X	X	X	[RWD] ← [RWD] + [RWS] + [C]

Mnemonic	Full Name	8086 Operand Usage
ADD - ADC	Add - Add with Carry	`ADD dest, src` - `ADC dest, src` (+CF)
SUB - SBB	Subtract - Subtract with Borrow	`SUB dest, src` - `SBB dest, src` (-CF)
INC - DEC	Increment - Decrement	`INC dest` - `DEC dest`
NEG	Two's Complement (Negate)	`NEG dest`
CMP	Compare	`CMP dest, src` (dest - src is performed to set flags)
MUL - IMUL	Multiply - Signed Multiply	`MUL src` - `IMUL src` (Implicitly uses `AL` or `AX`)
DIV - IDIV	Divide - Signed Divide	`DIV src` - `IDIV src` (Implicitly uses `AX` or `DX:AX`)

ADD AX + 1 ile INC AX farkı ?
→ INC kullanımında CF (carry flag) etkilenmez & INC genelde daha hızlıdır.

NEG - 2'ye tümleyen alır
CMP - SUB gibi davranır ama sonucu etkilemez, bayraklarla oynar:

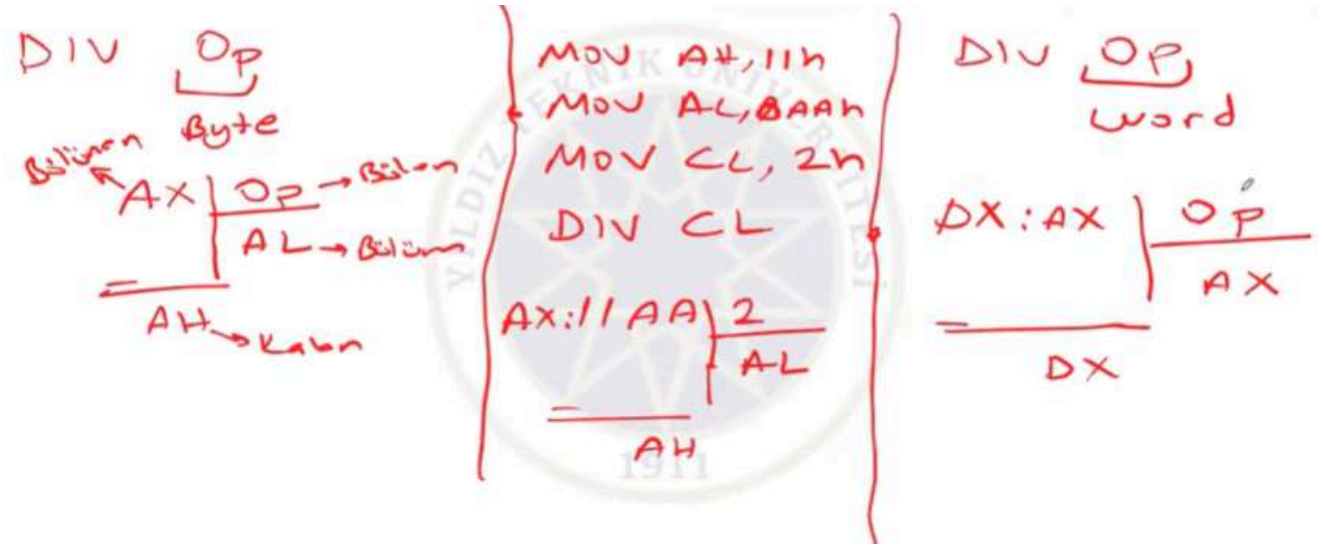
```
CMP AX, 10      ; AX - 10 işlemini yap (AX'in değerini değiştirme)
      JE  EsitlikVar  ; Jump if Equal (Eğer ZF=1 ise, yani AX = 10 ise sıçra)
      ; ... AX 10 değilse bu kod çalışır ...
      JNE EsitlikYok  ; Jump if Not Equal (Eğer ZF=0 ise, yani AX != 10 ise sıçra)

EsitlikVar:     ; ... Eşitlik durumunda yapılacaklar ...
```

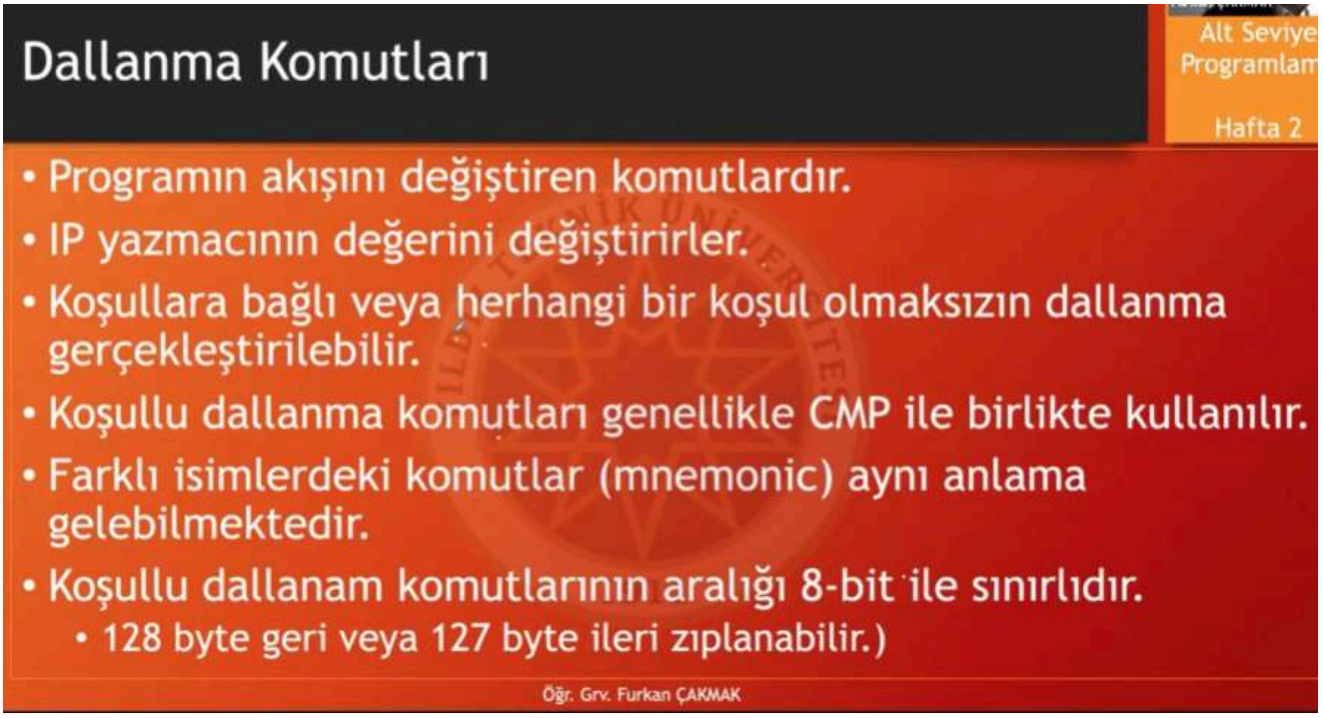


MUL' da tek operand var!

- MUL komutu, sonuç ne olursa olsun, word tipi çarpmada **otomatik olarak** 32-bitlik sonucu oluşturur ve iki hedef kayıtçıya yazar: `DX` (Yüksek Kısım) ve `AX` (Düşük Kısım).

Özellik	MUL (Multiply)	IMUL (Integer Multiply)
Kullanım Amacı	**İşaretsiz** (Unsigned) tamsayıları çarpar.	**İşaretli** (Signed) tamsayıları çarpar.
Negatif Sayılar	Sayıları her zaman pozitif kabul eder. Örneğin, 16-bit'te `0xFFFF`'i 65535 olarak görür.	Sayıları işaret bitini (en soldaki bit) dikkate alarak yorumlar. Örneğin, 16-bit'te `0xFFFF`'i -1 olarak görür.

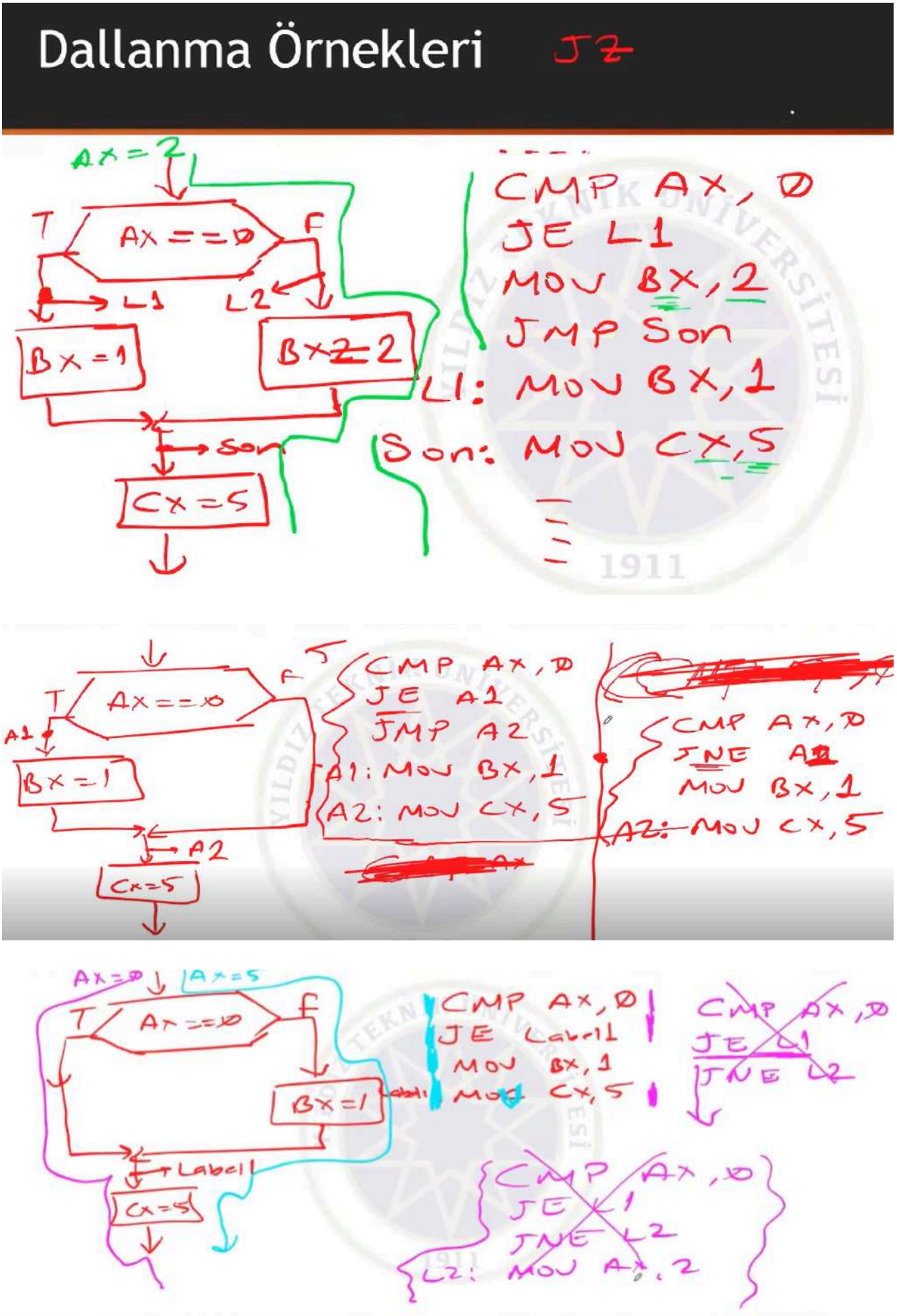




Dallanma Komutları ve Bayrakları

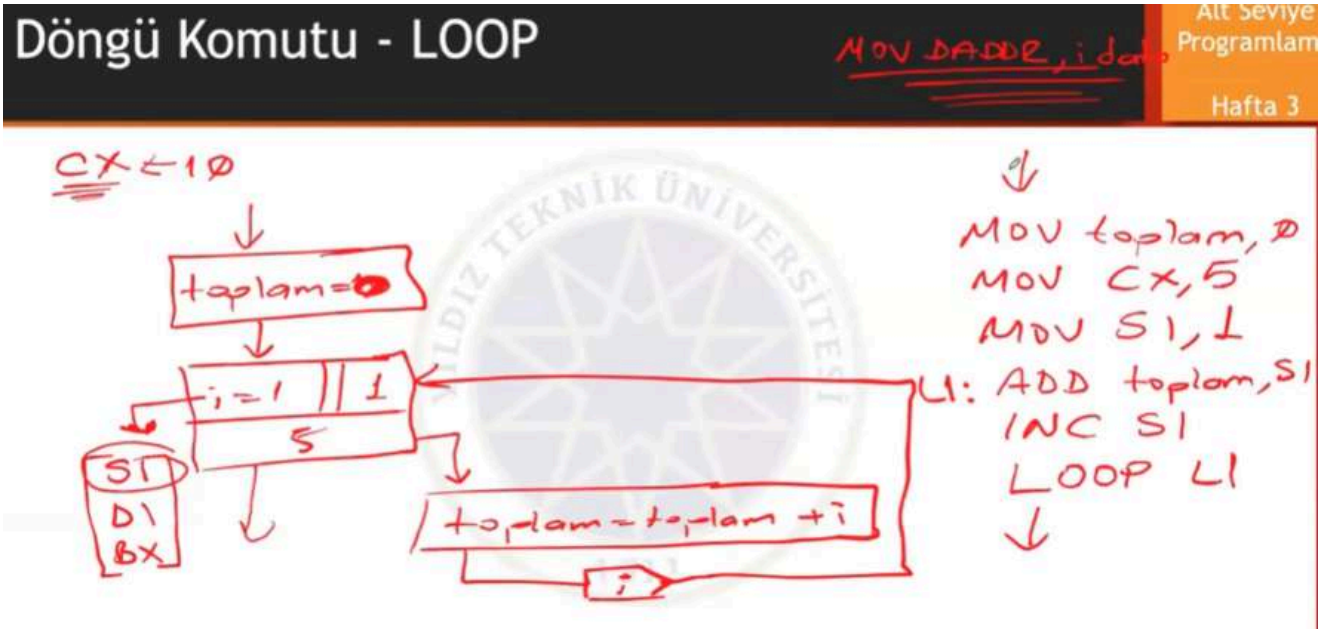
Koşulsuz dallanma → **JMP**
Koşullu Dallanmalar:

Mnemonic	Description	Condition (Flag State)
JZ / JE	Jump if Zero / Jump if Equal	$ZF = 1$
JNZ / JNE	Jump if Not Zero / Jump if Not Equal	$ZF = 0$
JC / JB / JNAE	Jump if Carry / Jump if Below / Jump if Not Above or Equal	$CF = 1$
JNC / JNB / JAE	Jump if No Carry / Jump if Not Below / Jump if Above or Equal	$CF = 0$
JS	Jump if Sign (Negative result)	$SF = 1$
JNS	Jump if No Sign (Positive or zero result)	$SF = 0$
JO	Jump if Overflow	$OF = 1$
JNO	Jump if No Overflow	$OF = 0$
JP / JPE	Jump if Parity / Jump if Parity Even	$PF = 1$
JNP / JPO	Jump if No Parity / Jump if Parity Odd	$PF = 0$
JL / JNGE	Jump if Less / Jump if Not Greater or Equal **(Signed)**	$SF \neq OF$
JGE / JNL	Jump if Greater or Equal / Jump if Not Less **(Signed)**	$SF = OF$
JLE / JNG	Jump if Less or Equal / Jump if Not Greater **(Signed)**	$ZF = 1$ or $SF \neq OF$
JG / JNLE	Jump if Greater / Jump if Not Less or Equal **(Signed)**	$ZF = 0$ and $SF = OF$



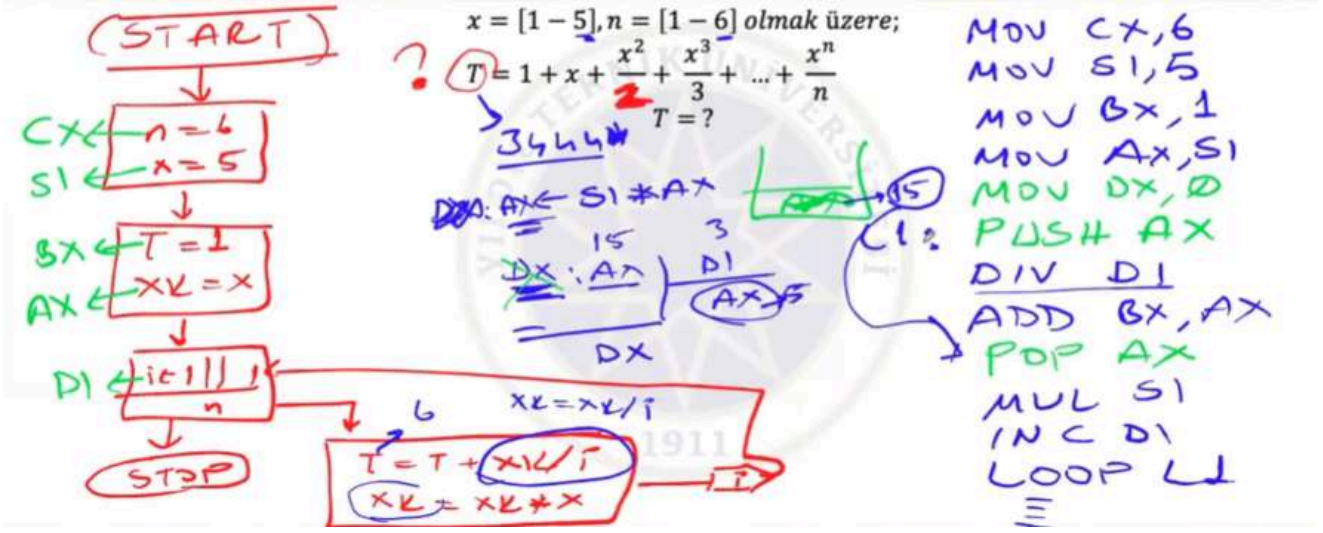
mor kodlardaki gereksizliklere dikkat et

Döngü Komutu - Loop



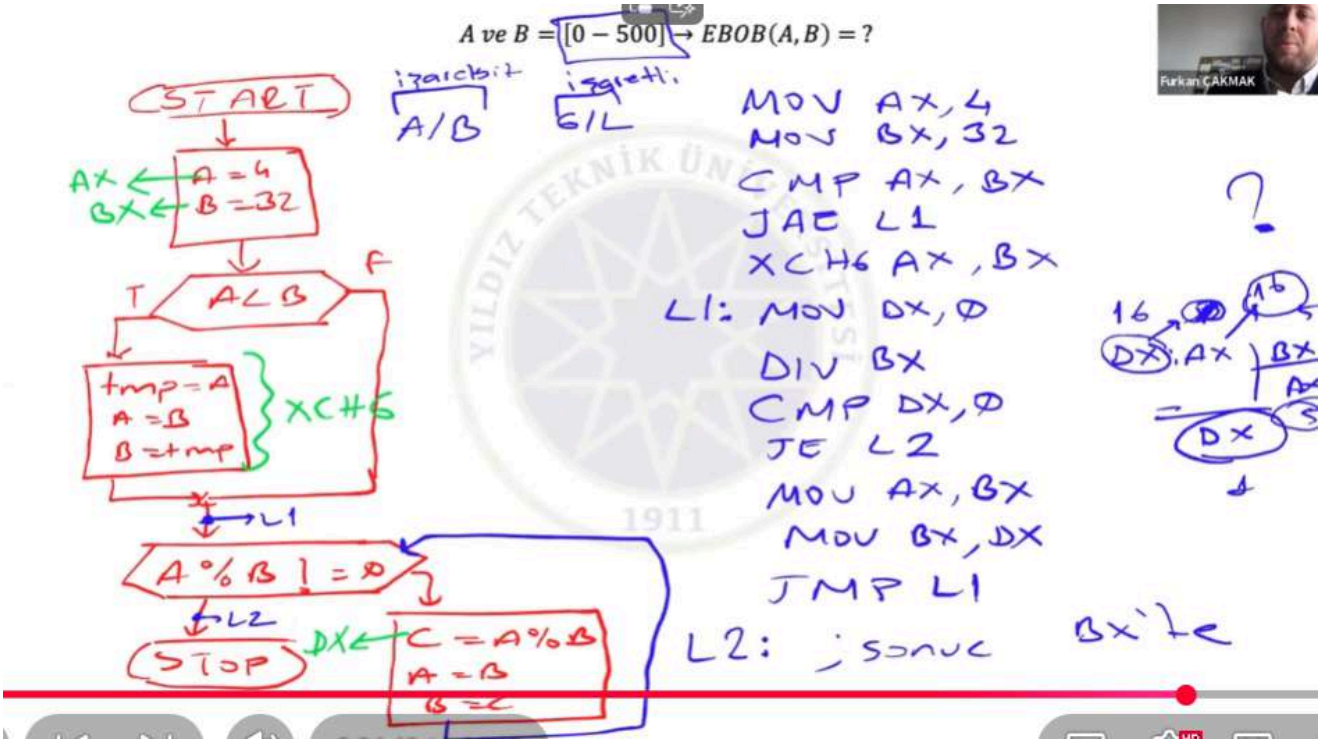
LOOP labela zıplamadan önce CX değerini 1 azaltır.

Örnek 1



(PUSH, POP işlemleri bölmeden önce veriyi korumak için kullanılır!)

Örnek 2: EBOB



Bayraklarla İlgili Komutlar

Bayraklarla İlgili Komutlar		
CLC - Clear CF	**STD** - Set DF	**SAHF** - Store AH in Flag
CMC - Complement CF	**STI** - Set IF	
STC - Set CF	**CLI** - Clear IF	
CLD - Clear DF	**LAHF** - Load AH with Flag	

Mantıksal Komutlar

- NOT, OR, AND, XOR, TEST
  TEST → AND ile aynı, sonuç operand içinde değil bayrakların içinde oluşur

veri sıfırlama yolları

MOV AX, 0
SUB AX, AX
XOR AX, AX

Komut	Açıklama	Bayraklar (Flags) Etkisi	Performans (Genel)
1. MOV AX, 0	AX'e doğrudan sıfır sabit değerini yükler.	**Etkilenmez.** Bayraklar korunur.	İyi, ancak veri yolu (bus) kullanımı nedeniyle en hızlısı değil.
2. SUB AX, AX	AX'i kendisinden çıkarır. ($AX - AX = 0$)	**Etkilenir.** `ZF=1`, `CF=0`, `OF=0`, `SF=0` olur.	Çok hızlı. İşlemci içinde kalır (register-to-register).
3. XOR AX, AX	AX'i kendisiyle XOR (özel veya) işlemine tabi tutar. ($X \oplus X = 0$)	**Etkilenir.** `ZF=1`, `CF=0`, `OF=0`, `SF=0` olur.	**En hızlısı.** Modern işlemcilerde sıfırlama için özel olarak optimize edilmiştir.

Öteleme Komutları

Komut	Tam Adı	İşlevi	Etkilenen Bayraklar
SHL	**Shift Logical Left** (Mantıksal Sola Kaydırma)	Değerin tüm bitlerini sola kaydırır. Sağdaki en düşük değere sahip (LSB) bite **0** eklenir. Sola kayan en yüksek değere sahip (MSB) bit **CF (Carry Flag)** bayrağına kopyalanır.	**CF**, **OF**, **SF**, **ZF**, **PF** güncellenir.
SAL	**Shift Arithmetic Left** (Aritmetik Sola Kaydırma)	**SHL** ile tamamen **aynı** işlevi görür ve aynı komut kodu (opcode) ile derlenir.	**CF**, **OF**, **SF**, **ZF**, **PF** güncellenir.
SHR	**Shift Logical Right** (Mantıksal Sağa Kaydırma)	Değerin tüm bitlerini sağa kaydırır. Soldaki en yüksek değere sahip (MSB) bite **0** eklenir. Sağdan kayan en düşük değere sahip (LSB) bit **CF (Carry Flag)** bayrağına kopyalanır.	**CF**, **OF**, **SF**, **ZF**, **PF** güncellenir.
SAR	**Shift Arithmetic Right** (Aritmetik Sağa Kaydırma)	İşaretli sayıların sağa kaydırılması için kullanılır. Değerin tüm bitlerini sağa kaydırır. Soldaki en yüksek değere sahip (MSB) bit **işaretini korumak** için **kendisiyle doldurulur**. Sağdan kayan en düşük değere sahip (LSB) bit **CF (Carry Flag)** bayrağına kopyalanır.	**CF**, **OF**, **SF**, **ZF**, **PF** güncellenir.

SAR örneği

SHL vs. SAL: Assembly dilinde **SHL** ve **SAL** tamamen aynıdır. Her ikisi de mantıksal sola kaydırma yapar ve bir sayıyı 2^n ile çarpmaya eşdeğerdir.
SHR vs. SAR:

- **SHR** (Mantıksal Sağa Kaydırma) işaretsiz sayılar için uygundur. Sayının başına her zaman **0** ekler.
- **SAR** (Aritmetik Sağa Kaydırma) işaretli sayılar için uygundur. Sayının işaretini (MSB bitini) koruyarak başına **aynı biti** ekler. Bu, işaretli bir sayıyı 2^n ile bölmeye eşdeğerdir.

```
    SHL AX, 1   ; doğru
    SHL AX, 2   ; yanlış!!!

    MOV CL, 2
    SHL AX, CL  ; doğru!!!
```

Tek Çift Sayı Kontrol örneği

```
        SHR sayi, 1
        JCF tek
        MOV AX, 1
        JMP sonuc
tek:    MOV AX, 0
sonuc:  ...
```

(TEST ve JCF komutu ile daha hızlısı yapılabilir)

Döndürme Komutları

Komut (Mnemonic)	Anlamı	Açıklama	Sözdizimi (Syntax)
ROL	**R**otate **L**eft (Sola Döndür)	Operandın bitlerini sola döndürür. En soldaki (MSB) bit hem Taşıma Bayrağı'na (CF) hem de en sağdaki (LSB) bite kopyalanır.	`ROL hedef, sayım`
ROR	**R**otate **R**ight (Sağa Döndür)	Operandın bitlerini sağa döndürür. En sağdaki (LSB) bit hem Taşıma Bayrağı'na (CF) hem de en soldaki (MSB) bite kopyalanır.	`ROR hedef, sayım`
RCL	**R**otate **C**arry **L**eft (Taşıma ile Sola Döndür)	Operandın bitlerini sola, Taşıma Bayrağı (CF) aracılığıyla döndürür. En soldaki bit CF'ye kopyalanır ve CF'nin önceki değeri en sağdaki bite kopyalanır.	`RCL hedef, sayım`
RCR	**R**otate **C**arry **R**ight (Taşıma ile Sağa Döndür)	Operandın bitlerini sağa, Taşıma Bayrağı (CF) aracılığıyla döndürür. En sağdaki bit CF'ye kopyalanır ve CF'nin önceki değeri en soldaki bite kopyalanır.	`RCR hedef, sayım`

- Sadece **bir (1) bit** döndürülecekse, doğrudan 1 yazılır (`ROL AX, 1`)
- Birden fazla bit döndürülecekse, döndürme sayısı **CL** register'ına yüklenmeli ve komutta CL kullanılmalıdır!
  (örneğin, `MOV CL, 4` ardından `ROL AX, CL`)

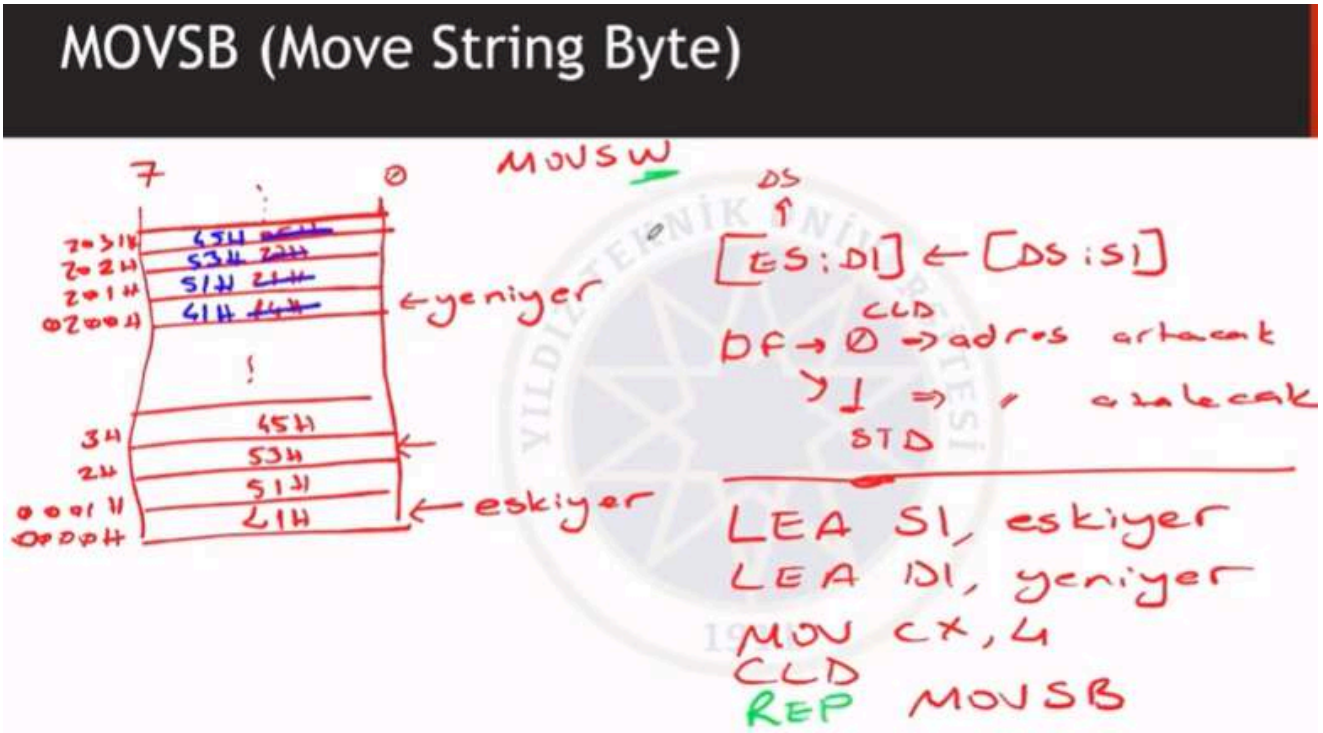
Örnek: 8 bitlik sayının bitlerini tersten yazmak

a = 1110 0010 olsun. istenen sonuç → a = 0100 0111

```
    MOV AL, a
    MOV CX, 8
L1: SHR AL, 1
    RCL AH, 1
    LOOP L1
    MOV a, AH
```

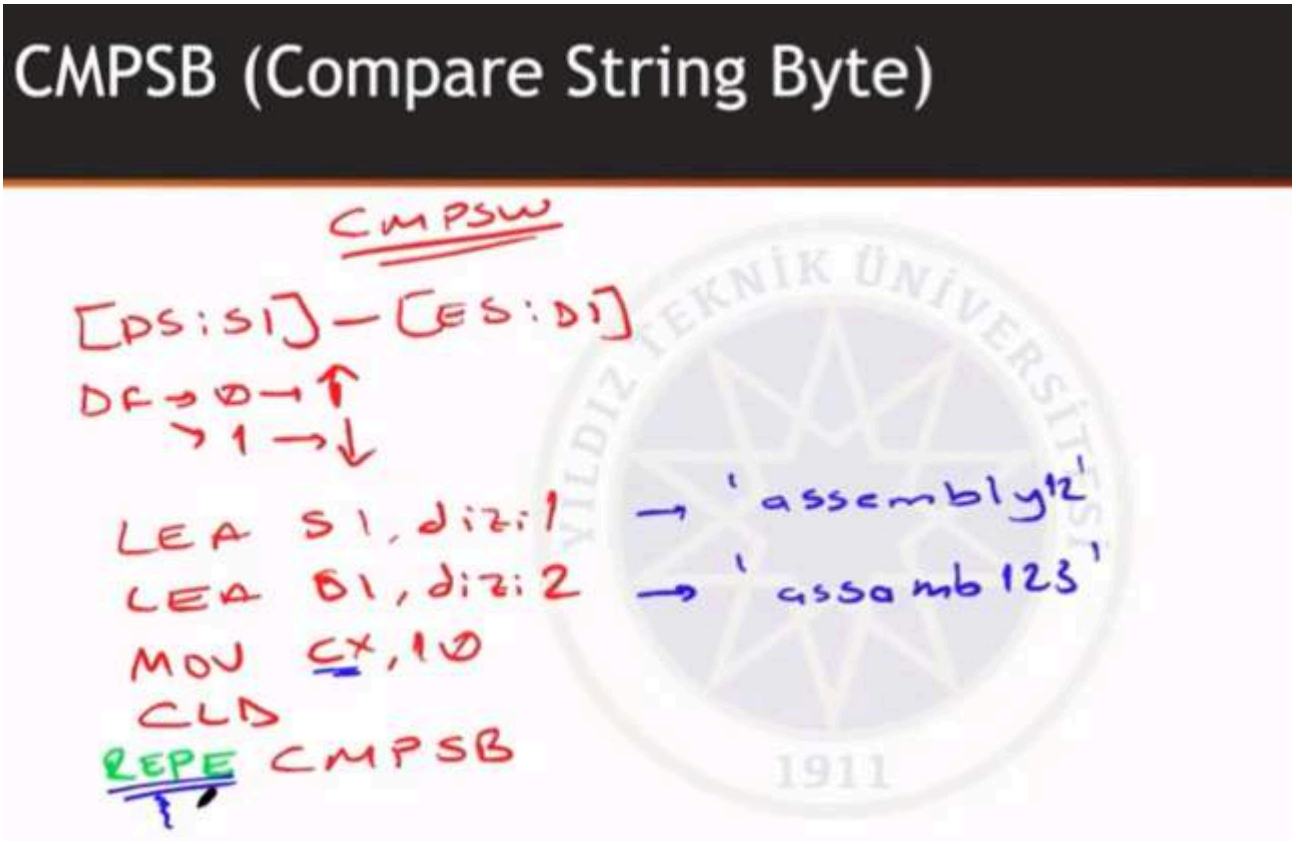
Dizgi(String) Komutları

Komut (Mnemonic)	Anlamı	Açıklama
MOVS / MOVSB / MOVSW	Dizge Taşıma	Bellekteki bir dizgeden diğerine bir **byte (B)** veya **word (W)** taşır. Veriyi `DS:[SI]` adresinden `ES:[DI]` adresine taşır, ardından `SI` ve `DI`'ı günceller.
LODS / LODSB / LODSW	Dizge Yükleme	Bellekteki veriyi (`DS:[SI]`) **Akümülatör'e** (`AL` veya `AX`) yükler ve `SI`'ı günceller.
STOS / STOSB / STOSW	Dizge Saklama	**Akümülatör'deki** veriyi (`AL` veya `AX`) bellekteki bir dizgeye (`ES:[DI]`) saklar ve `DI`'ı günceller.
CMPS / CMPSB / CMPSW	Dizge Karşılaştırma	İki dizgedeki verileri (`DS:[SI]` ve `ES:[DI]`) bir **byte** veya **word** uzunluğunda karşılaştırır ve bayrakları ayarlar. `SI` ve `DI`'ı günceller.
SCAS / SCASB / SCASW	Dizge Tarama	Akümülatör'deki (`AL` veya `AX`) veriyi bellekteki dizgedeki veriyle (`ES:[DI]`) karşılaştırır ve bayrakları ayarlar. `DI`'ı günceller.

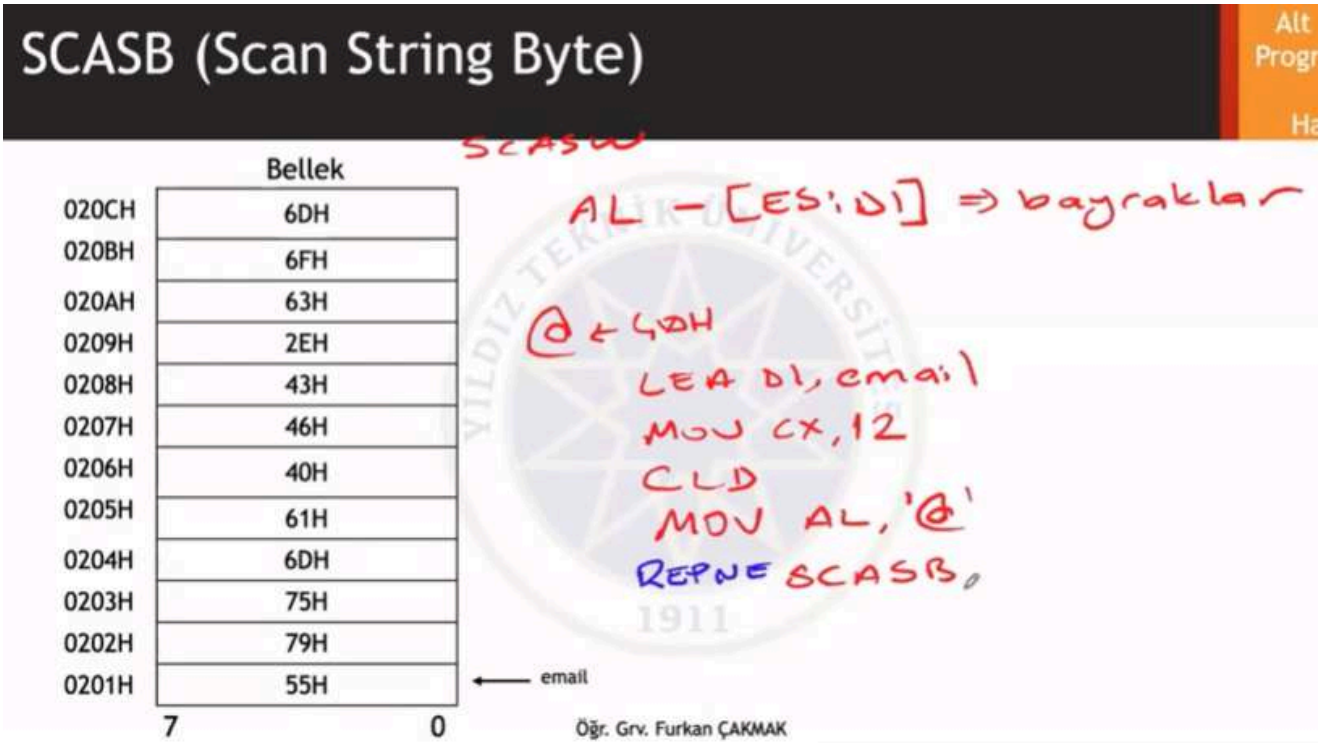


(REP: LOOP'un özel hali. Önündeki komutu tekrarlar)

- Veriyi taşımak istediğimiz ilerideki adres veriyle örtüşüyorsa DF, STD komutuyla değiştirilip dizinin sonundan adres azaltarak taşınmalı!



(REPE: Repeat If Equal)

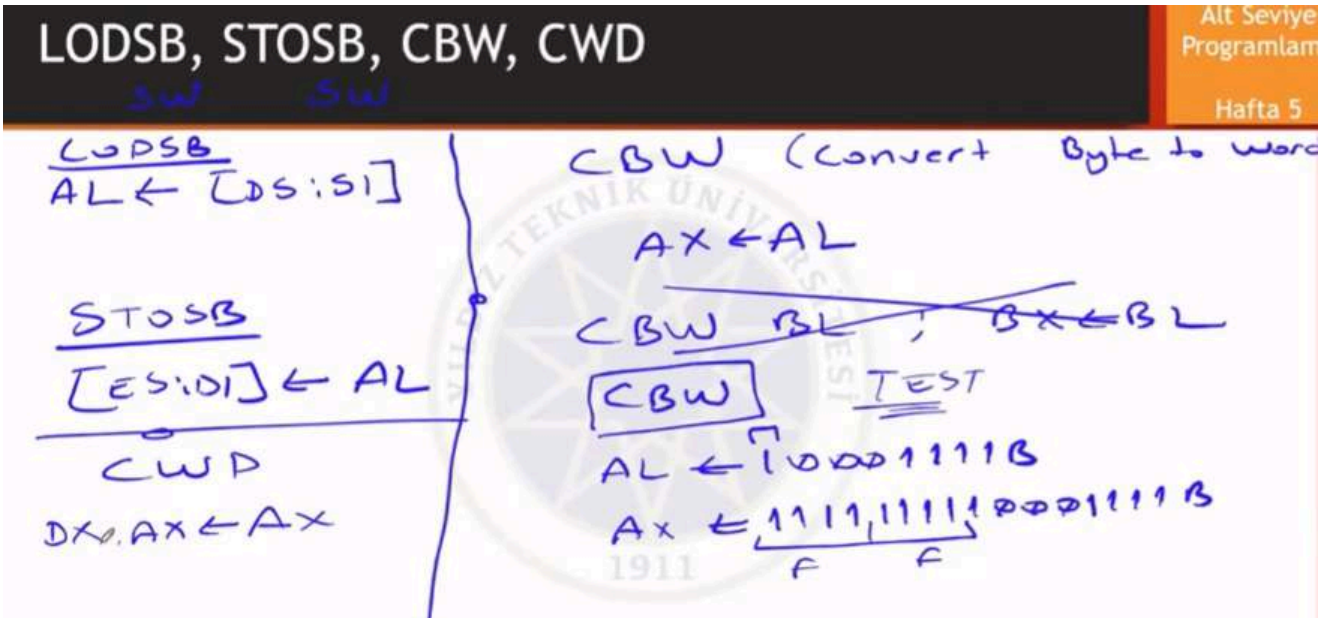


(REPNE: Repeat if NOT Equal)

CBW (Convert Byte to Word)

Operand'i yok! AL' deki sayıyı AX'e büyütür.

CWD (Convert Word to Double Word)



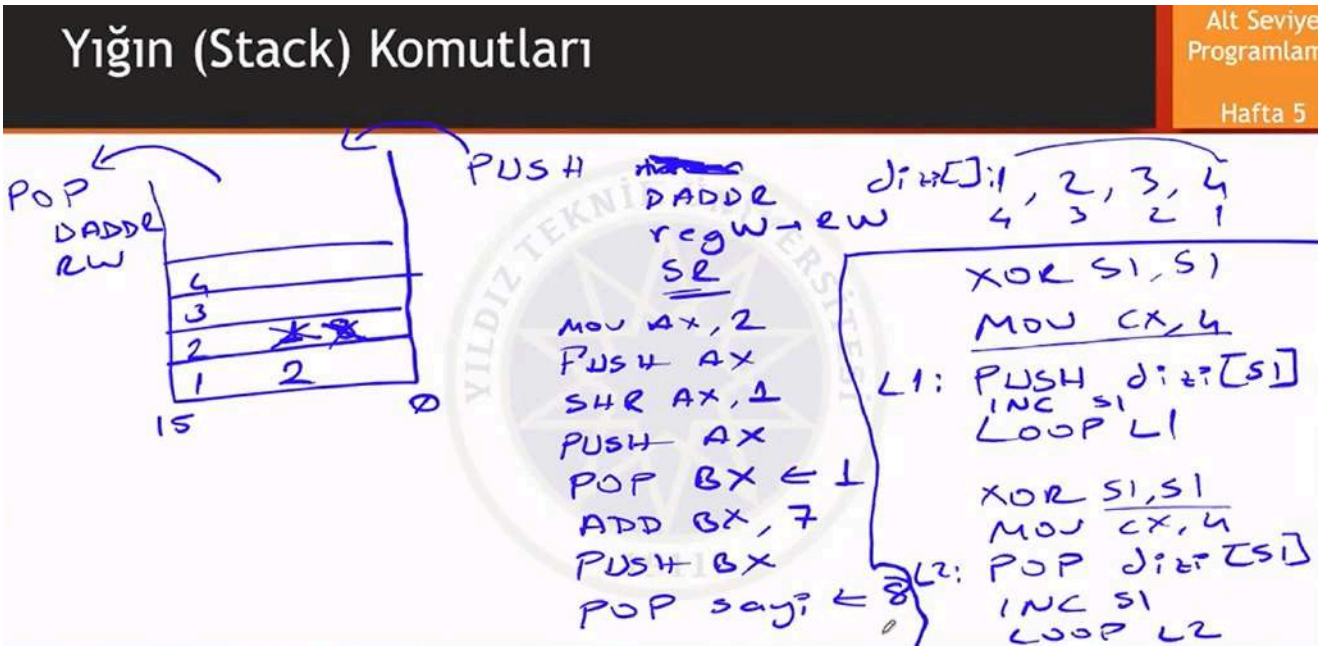
Örnek:

1. **Pozitif Sayı:** `MOV AX, 0005h`.
   - `CWD` komutundan sonra: DX:AX = 0000:0005h
2. **Negatif Sayı:** `MOV AX, 8000h` (1000 0000 0000 0000b -32768).
   - `CWD` komutundan sonra: $\mathbf{DX : AX = FFFF : 8000h}$.

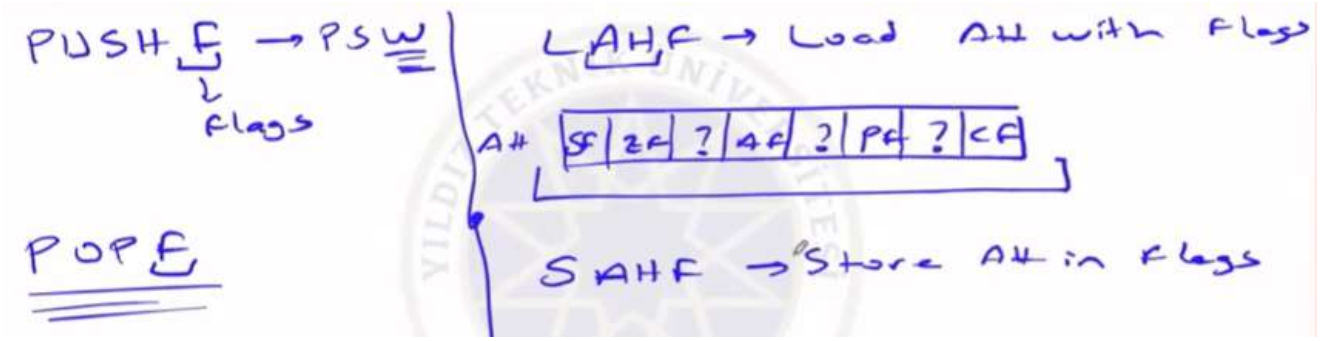
Bu komutlar, özellikle işaretli bölme komutu olan `IDIV`'i kullanmadan hemen önce gereklidir, çünkü `IDIV` bölünen sayının AX veya $DX : AX$ çiftinde olmasını bekler.

Yığın(Stack) Komutları

Komut (Mnemonic)	Anlamı	Açıklama
PUSH	Yığına İtme	Bir kelime (16 bit) uzunluğundaki veriyi yığının tepesine yerleştirir ve **SP**'yi 2 azaltır.
POP	Yığından Çekme	Yığının tepesinden bir kelime (16 bit) uzunluğundaki veriyi alır ve **SP**'yi 2 artırır.
PUSHF	Bayrakları İtme	**Bayraklar (Flags)** yazmacının (16 bit) içeriğini yığına kaydeder.
POPF	Bayrakları Çekme	Yığının tepesinden bir kelimeyi alarak **Bayraklar** yazmacına yükler.
CALL	Çağırma	Prosedüre atlar ve **dönüş adresini (IP'nin sonraki değeri)** yığına kaydeder.
RET	Geri Dönüş	Yığından en son kaydedilen dönüş adresini alır ve program akışını oradan devam ettirir.



PUSHF, POPF, LAHF, SAHF



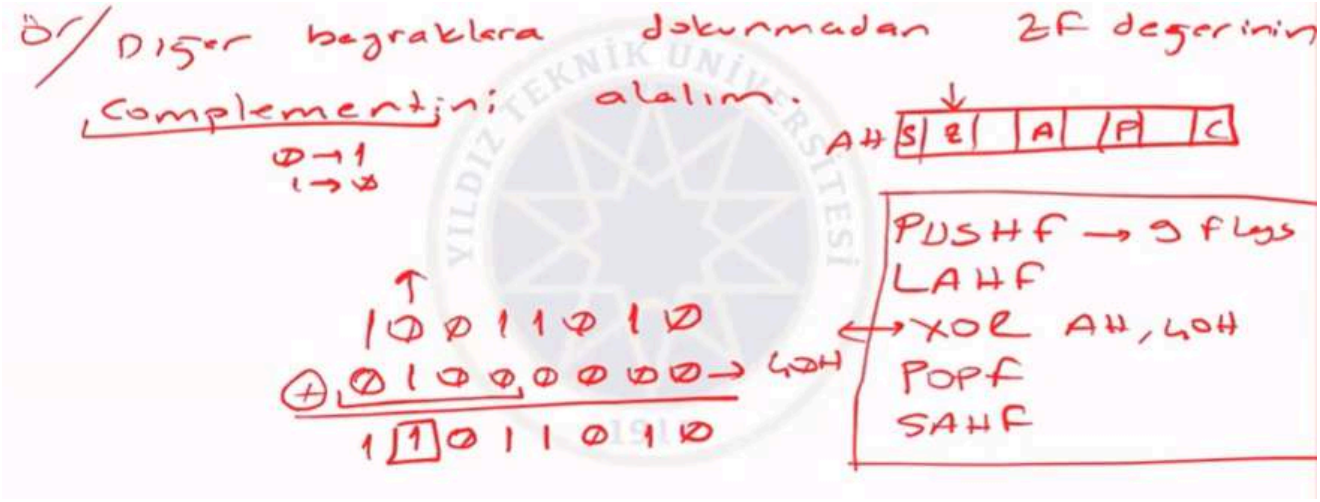
Örnek1:

Diğer bayraklara dokunmadan ZF değerinin complementini alınız.

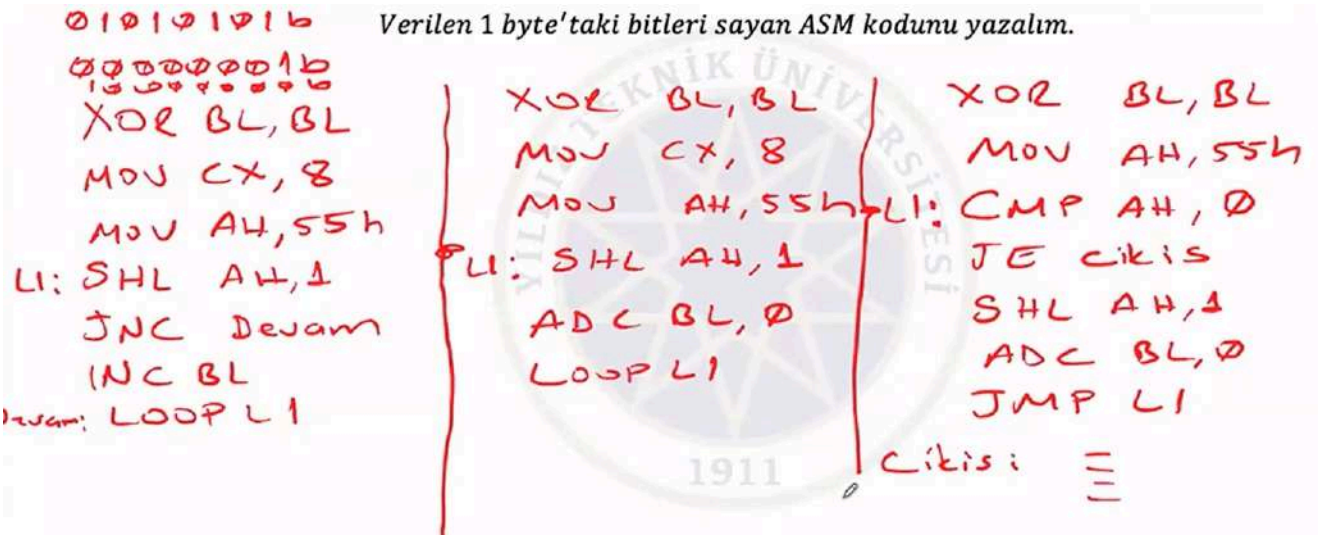
7	6	5	4	3	2	1	0
SF	ZF	-	AF	-	PF	-	CF

Nasıl Yapılır? Değiştirilmek istenen bit 1 diğer bitler 0 olan ikinci bir sayıyla XOR alınır

```
PUSHF
LAHF
XOR AH, 40H
POPF
SAHF
```

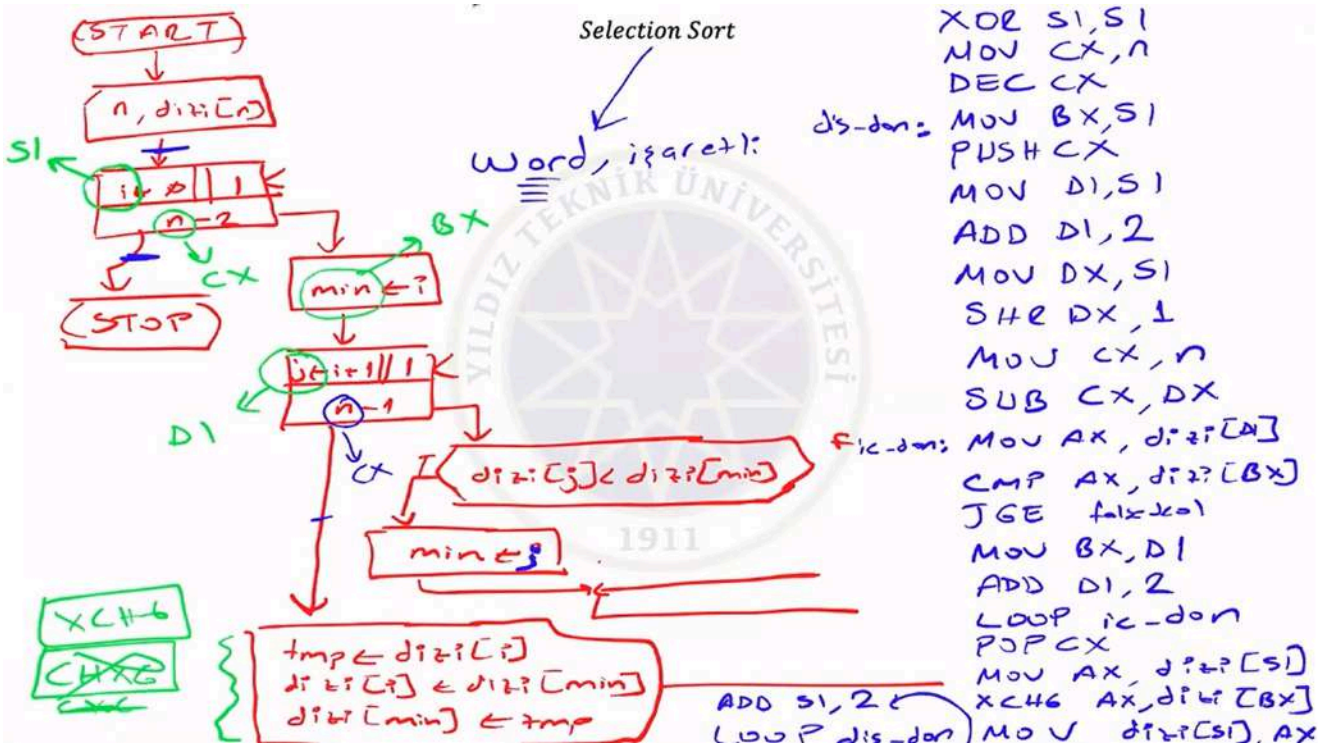


Örnek 2



Bir problem birden farklı şekilde çözülebilir. Önemli olan en optimize halini bulmaktır.

Örnek 3: Selection Sort



Adresleme Kipleri

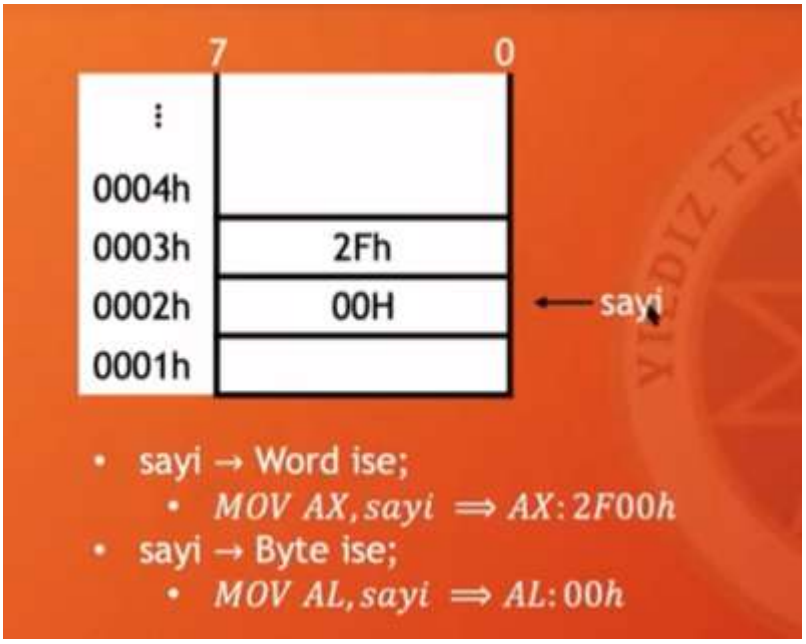
Adresleme: Belleğin içine bir şeyler koyma gibi düşünülebilir.

Anlık Adresleme

Bir register veya değişken içerisine doğrudan sayısal değer ataması yapılan adresleme tipidir.

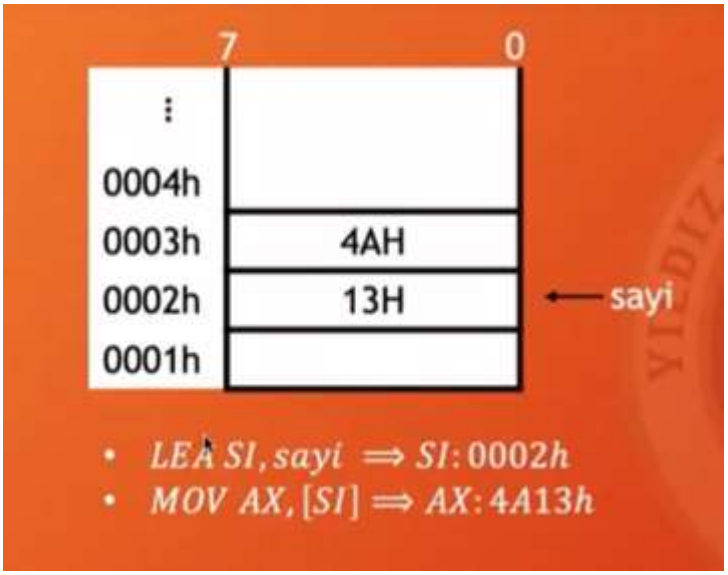
Register(Yazmaç) Adresleme

Direct (Doğrudan) Adresleme

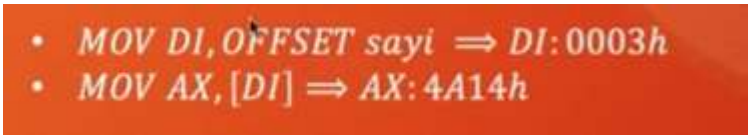


adresin ne yönde arttığına her zaman dikkat et

Register Indirect (Yazmaç Dolaylı) Adresleme

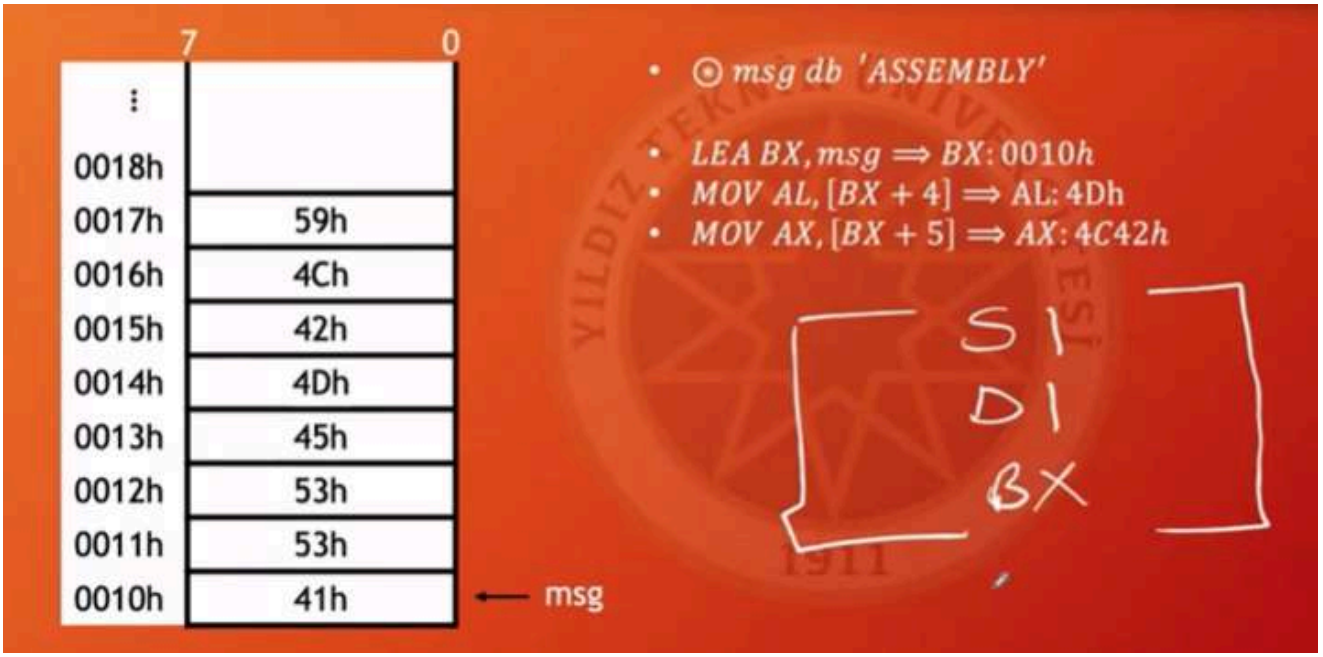


AL yazsaydı sadece 13h alınırdı



OFFSET bir mnemonic değildir. pseoudo komutlardan biridir, derleyici için anlam ifade eder.
(⚠️ resimdeki 0003h hatalı, 0002h olmalı ⚠️)

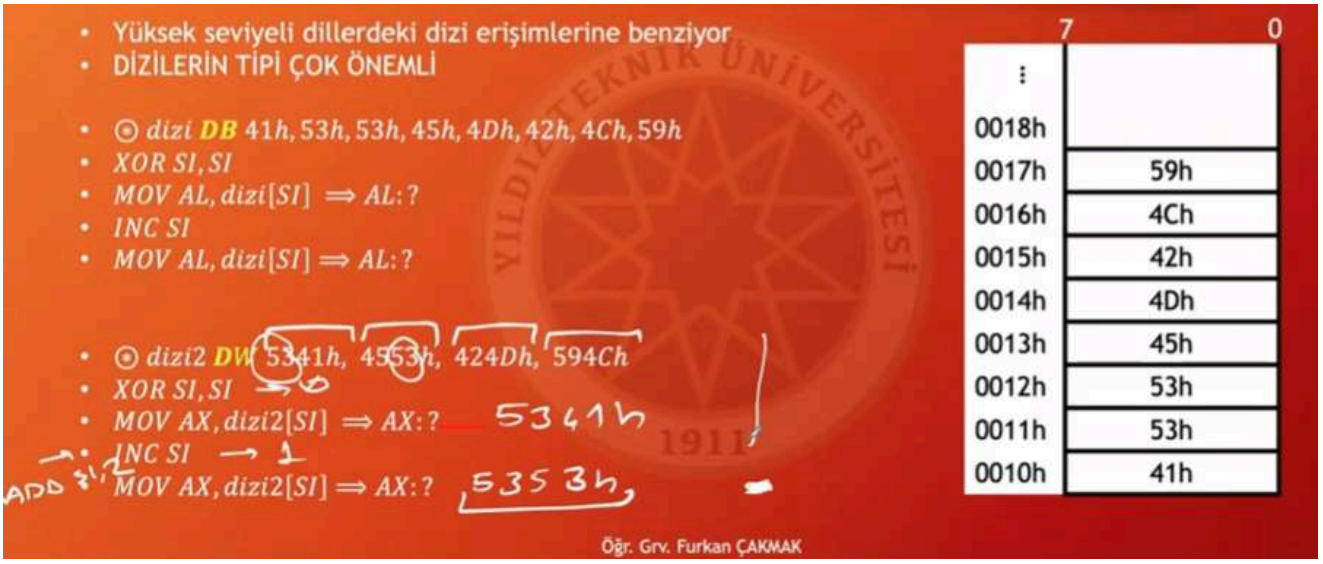
Base Relative (Baz Göreli) Adresleme



msg db 'Assembly' kısmı sonra anlatılacak (db: define byte)

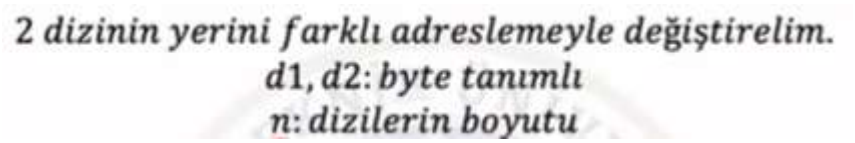
Direct Index (Doğrudan İndisli) Adresleme

Doğrudan İndisli Adresleme yüksek seviyeli dillerde kullanılan dizi adreslemeye benzer.



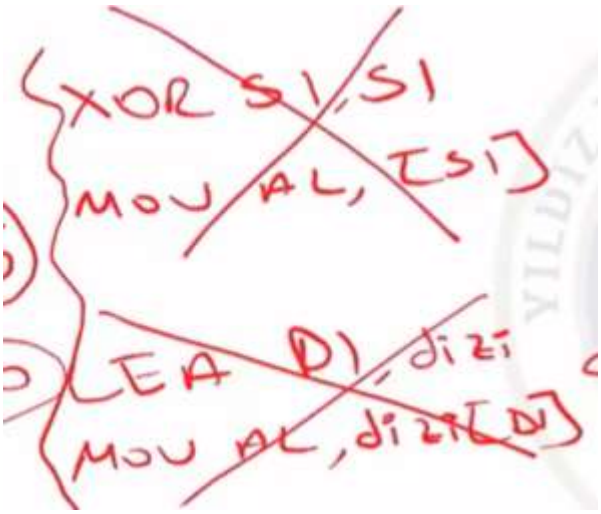
dizi2[SI +1] de hatasız olur ama döngü kurabilmek için INC SI kullanmak daha iyi
not: assembly de verilere erişirken dizi dışına hatasız çıkılabilir.

Örnek 1




```
        MOV CX, n
        XOR SI, SI      ; d1 için-> SI -> 0
        LEA DI, d2      ;           DI -> 1FA2h (örnek)

donus:  MOV AL, d1[SI]
        XCHG AL, [DI]   ; XCHG [DI], d1[SI] KESİNLİKLE YANLIŞ!
                        ; XCHG DADDR, DADDR diye bir instruction yok!
        MOV d1[SI], AL
        INC DI
        INC SI
        LOOP donus:
```

Potansiyel Yanlışlar



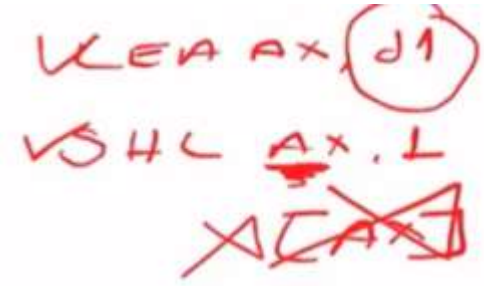
Örnek 2

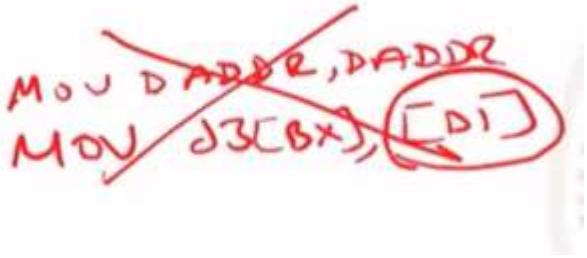
A şirketinde çalışanların doğum yılı d1 dizisinde, B şirketinde çalışanların yaşları d2 dizisinde tutulmaktadır. Önce A şirketinde çalışan a1 kişinin, sonra B şirketinde çalışan b2 kişinin bu bilgileri boyutu a1 + b1 olan d3 dizisine aktarmak için gerekli ASM kodunu yazalım.

doğum yılı: 255te büyük olacağı için WORD tipi olmalı
çalışan yaşları negatif olamaz, 0-255 arasında olabilir (BYTE tipi)

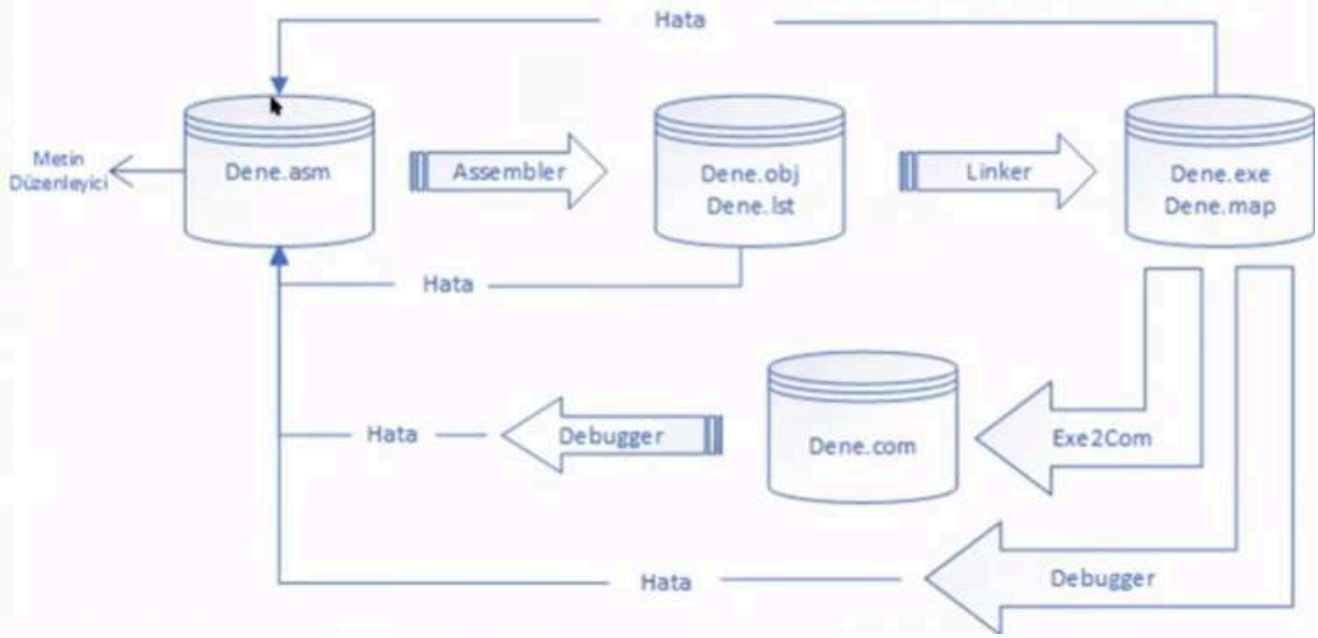
```
        LEA DI, d1
        LEA SI, d2
        XOR BX, BX

        MOV CX, a1
L1:         MOV AX, [DI]
        MOV d3[BX], AX
        ADD DI, 2
        ADD BX, 2
        LOOP L1

        MOV CX, b2
L2:         MOV AL, [SI]
        CBW                 ; MOV d3[BX], AL hatalı olur
        MOV d3[BX], AX      ; çünkü d3 dizisi WORD tipi!
        ADD SI, 1
        ADD BX, 2           ; ARADAKİ FARK ÖNEMLİ
        LOOP L2
```

Potansiyel Yanlışlar





Assembly Programı Derleme ve Çalıştırma



Assember assembly derleyicisine verilen addır.
Hata yoksa obj uzantılı dosyayı oluşturur.

Assemblyde EXE ve COM tipi programlar bulunur.
COM yazılsa bile önce EXE tipi programdan geçirilir.

Pseudo (Sözde) Komutlar

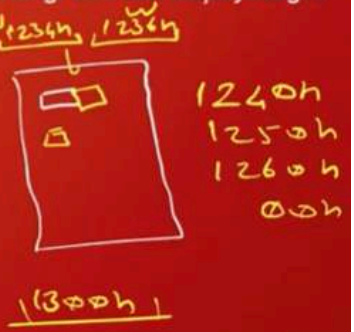
- Makine kodu karşılığı yoktur.
- Bayraklar üzerinde etkisi yoktur.
- Debug ortamında kullanılamıyorlar.
- LST dosya düzeni
  - Program kesim düzeni
- Veri tanımlamaları için kullanılırlar.
- Derleyici tarafından anlamlandırılırlar.

SEGMENT/ENDS

```
kesim_ismi      SEGMENT {seçenekler}
                ..............
                ..............
                ..............
kesim_ismi      ENDS
```

SEÇENEKLER

a) **Hizalama** (Alignment) Tipi:
- Tanımlanan kesimlerin hangi sınırdan başlayacağını belirler.
- BYTE /1
- WORD /2
- PARA /16
- PAGE /256



c. **Sınıf** (Class) Tipi:
- Birleştirme tipi PUBLIC olarak tanımlanmış kesimlerin link işlemi sırasında birbirinin peşi sıra gelmesi için kullanılır.
- Tırnak içinde yazılır.

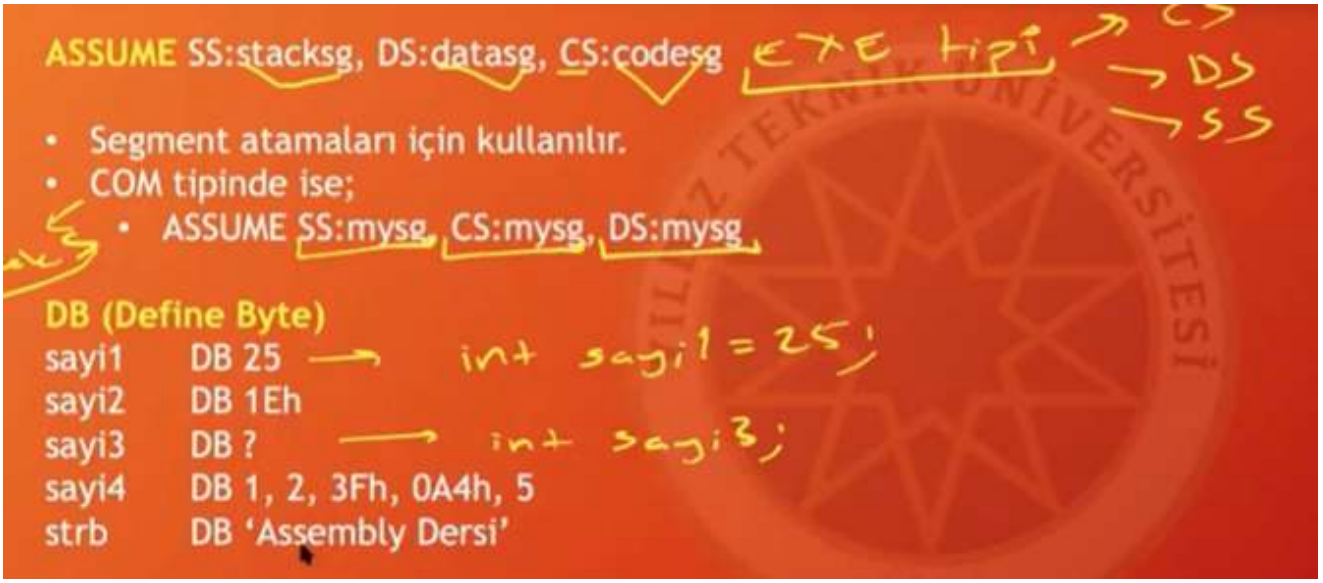
b) **Birleştirme** (Combine) Tipi:
- Link işleminden sonra aynı isim verilmiş kesimlerin birleştirilmesi için kullanılır.
- **PUBLIC**: Aynı isimli kesimler birbirinin devamı olur.
- **COMMON**: Kesimler birbirinin üzerine yazılır.
- **STACK**: LIFO sisteminde çalışır.
- **AT #####**: Belirlenen fiziksel adresten başlar.

```
stacksg     SEGMENT PARA STACK 'yigin'
            ..............
stacksg     ENDS
codesg      SEGMENT PARA 'kod'
            ..............
codesg      ENDS
datasg      SEGMENT PARA 'veri'
            ..............
datasg      ENDS
```

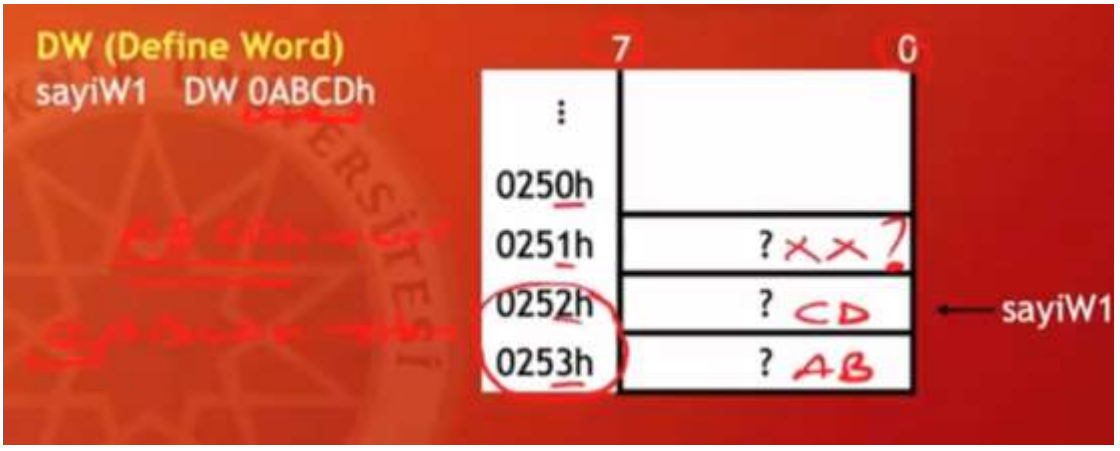
test SEGMENT BYTE AT 0120AH

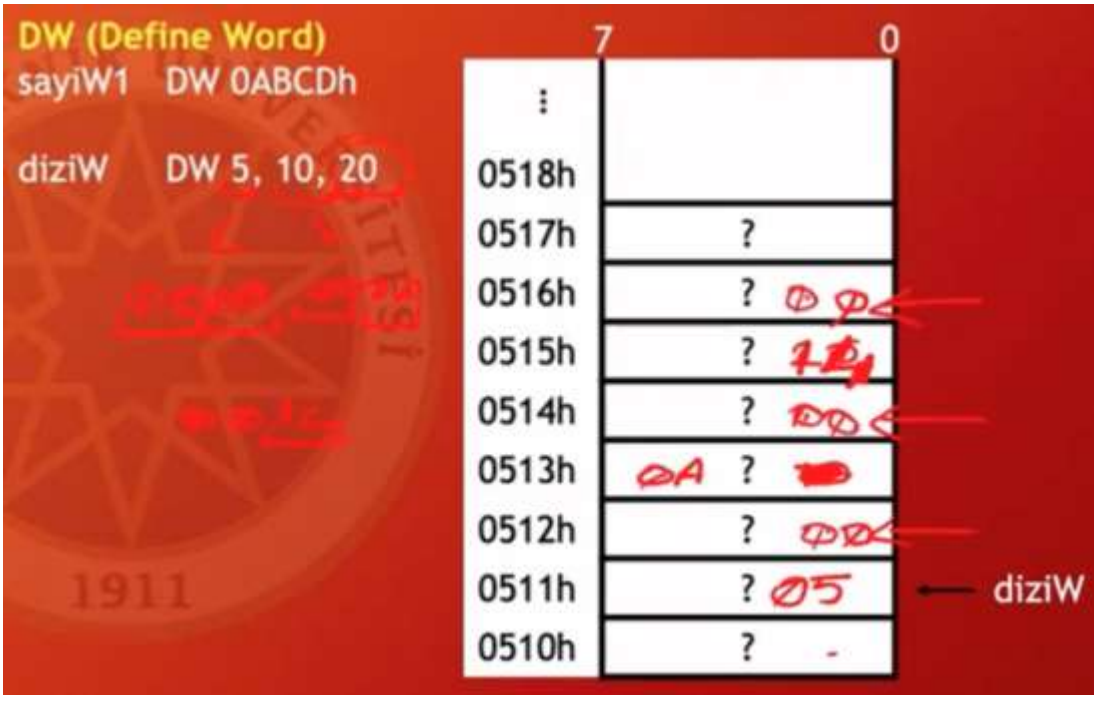
test ENDS

ASSUME, DB, DW, DD, DQ, DT



? (soru işareti) sembolü ilk değerinin ne olduğunun önemi yok demektir.
Virgülle ayırılan elemanlar dizi oluşturur, tırnak içine yazılanlar da string dizisi oluşturur.
COM tipi programda tek bir segment bulunur (bu dönem işlenmedi)

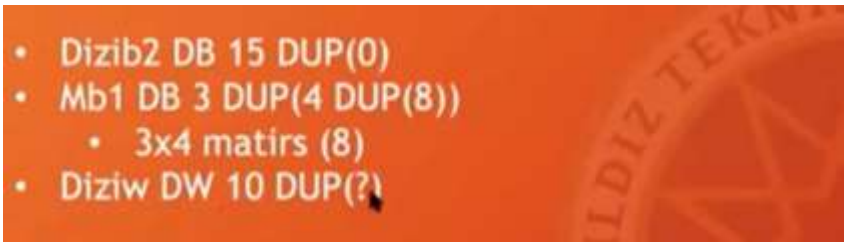





DD (Define Double Word)
DQ (Define Quadro Word)
DT (Define Ten Bytes)

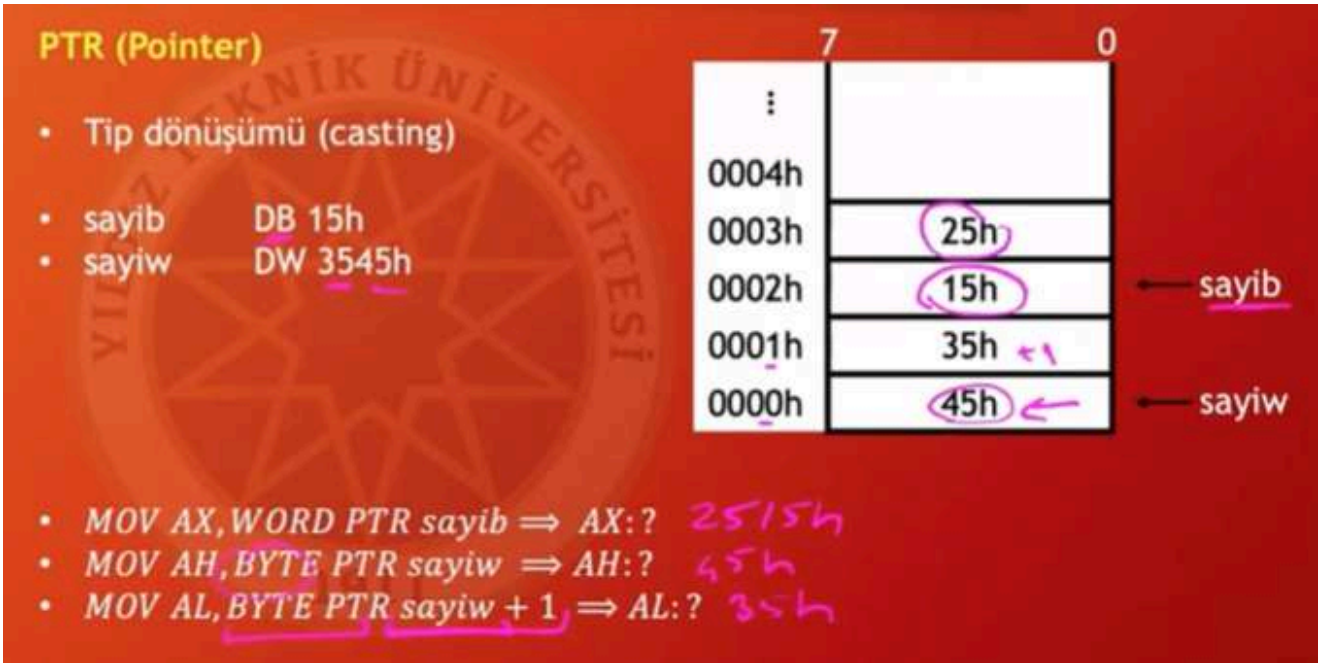
DUP, PTR

Dup (Duplication Factor)



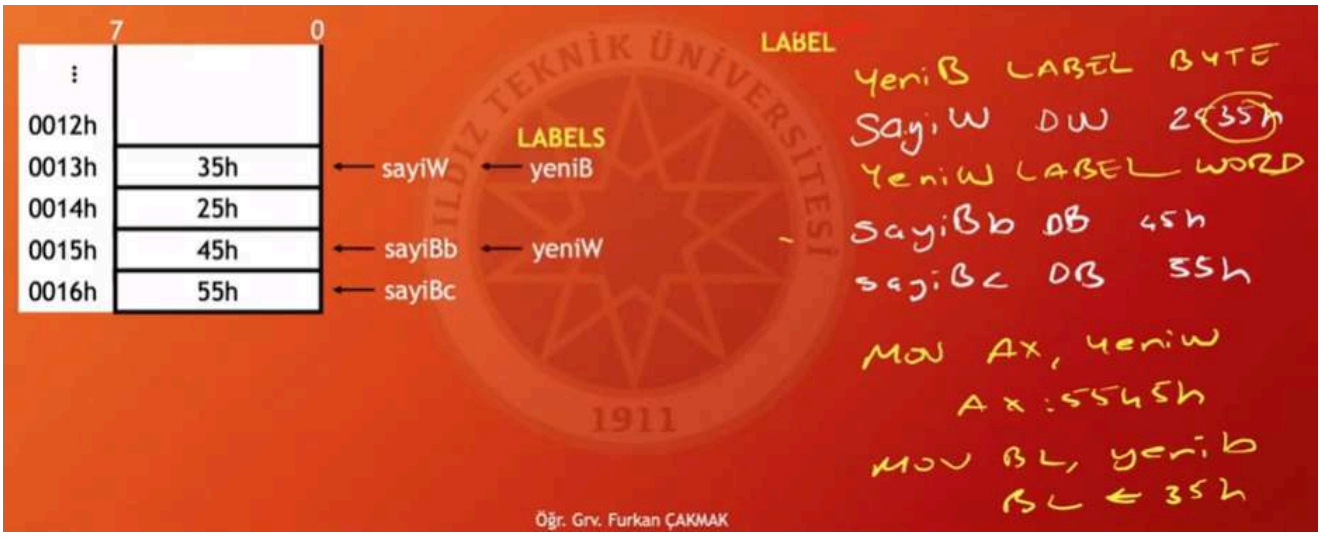
önüne gelen rakam kaç tane duplicate edileceğini gösterir
içindeki ifade o değerin ne olduğunu gösterir

PTR (Pointer)

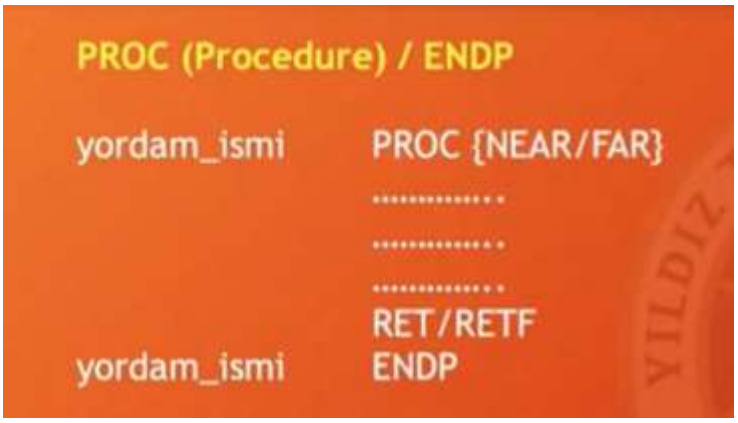


LABEL

tanımlandığı yerden sonraki belirlenen adres büyüklüğündeki veriyi alır

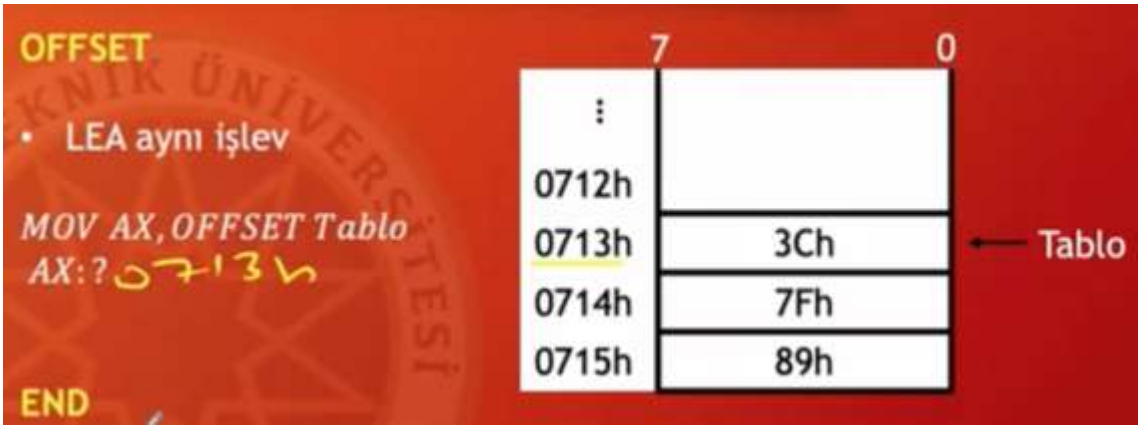


PROC/ENDP, SEG, END



PROC/ENDP: prosedür başlangıcı ve sonunu belirtmek için kullanılır

- NEAR , prosedürün aynı kod segmenti içinde çağrılacağını belirtir. (Varsayılan değerdir)
- FAR , prosedürün farklı bir kod segmentinden çağrılabileceğini belirtir.

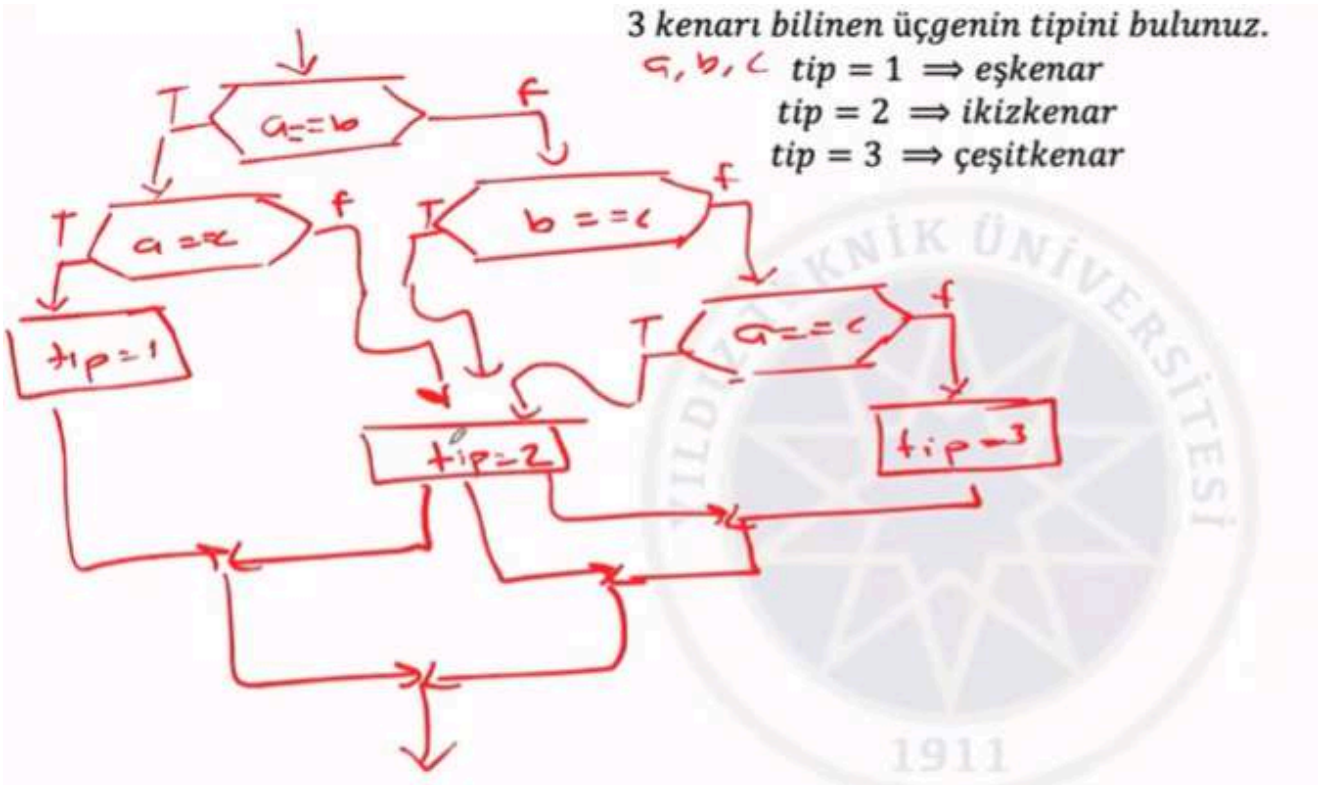


Derleyici END komutunu gördüğünde daha fazla kaynak kod okumayı durdurur.

LENGHT, TYPE, SIZE

Artık baştan sonra tam bir Assembly kodu yazabiliriz!

Örnek 1



```
datasg SEGMENT PARA 'veri'
   tip DB ?
   a   DB 12
   b   DB 13
   c   DB 12
datasg ENDS

; stack boyutu yazılan programda ihtiyaç duyulduğu kadar oluşturulmalı
stacksg SEGMENT PARA STACK 's'
   DW 15 DUP(?)
stacksg ENDS

codesg SEGMENT PARA 'kod'
   ASSUME CS:codesg, DS:datasg, SS: stacksg

ANA PROC FAR
   ; Sıradaki 5 satırdaki ifadeler bir EXE tipi kodda bulunmalı!
   ; İşlevlerine sonra değinilecek
    PUSH DS
    XOR AX, AX
    PUSH AX
    MOV AX, datasg
    MOV DS,AX

    ; Artık program kısmına başlayabiliriz
    MOV AL, a
    MOV BL, b
    MOV CL, c

    CMP AL, BL
    JE J1
    CMP BL, CL
    JE J2
    CMP AL, CL
    JE J2
    MOV tip, 3
    JMP Json
J1:  CMP AL, CL
    JNE J2
    MOV tip, 1
J2:  MOV tip, 2

Json:RETF

ANA ENDP               ; Önce prosedürü kapatıyoruz
```

```
codesg ENDS             ; Sırada kod segmenti var

END ANA                 ; en son END ve programın başlamasını
                        ; istediğimiz prosedür ismi yazılmalı
                        ; zaten 1 prosedür var, o da ANA
```

Başlangıç Ayarları

Code Segmentteki ilk 5 satırın amacı:

PUSH DS
XOR AX, AX - > Geri dönüş adreslerinin ayarlanması
PUSH AX

MOV AX, datasg - > data segment değerinin ayarlanması
MOV DS,AX

Bu ifadeler kodun doğru çalışması için bir EXE programında kesinlikle bulunmalı.
Program bittiğinde atılan veriler geri çekilip sonlandırılır.

DOSBOX (Lab için)

modern işletim sistemleri üzerinde eski DOS (Disk Operating System) ortamını taklit eden bir emülatör programı. Lablarda çıktı, hafıza veya registerları test etmek için kullanılmakta.

Ödev ve Lablar için önemli bazı komutlar:

MOUNT	**Bilgisayarınızdaki bir klasörü, DOSBox'ta sanal bir sürücü (örneğin C:) olarak bağlar.**
C:	Aktif olarak çalışılan sürücüyü, bağlanan **sanal C sürücüsü** olarak değiştirir.
CD	Aktif olarak çalışılan **dizini (klasörü)** değiştirmeye yarar.
MASM	Assembly kaynak kodunu (.ASM) **çalıştırılabilir makine koduna** çeviren derleyici (Assembler) programıdır.
LINK	**Bağlayıcı (Linker)** programıdır. MASM tarafından üretilen .OBJ **nesne kodunu** alıp, gerekli kütüphanelerle birleştirerek **çalıştırılabilir** .EXE veya .COM dosyasını oluşturur.
DEBUG	Bir DOS programıdır. **Hata ayıklama (Debugging)** yapmak, belleği incelemek, yazmaç (Register) içeriklerini görmek ve programı **adım adım** çalıştırmak için kullanılır.
DIR	Mevcut dizindeki dosyaları listeler.

Ayrıca oluşan exe dosyasını çalıştırmak için "*program1.exe*" yazılabilir.
DOSBox büyük/küçük harf duyarlı (case-sensitive) değildir!

Dosya Tipi	Direkt Tanım
OBJ (.OBJ)	MASM'in ürettiği, henüz çalıştırılamayan **nesne kodu**.
EXE (.EXE)	LINK'in ürettiği, doğrudan çalıştırılabilen **yürütülebilir dosya**.
LST (.LST)	Kaynak kodu ve ilgili **makine kodunu** gösteren listeleme dosyası.
CRF (.CRF)	Etiket ve değişkenlerin hangi satırlarda kullanıldığını gösteren **çapraz referans** listesi.

DEBUG Komutları (hepsi önemli değil)

debug ile programın herhangi bir kısmında doğru çalıştığı kontrol edilebilir.

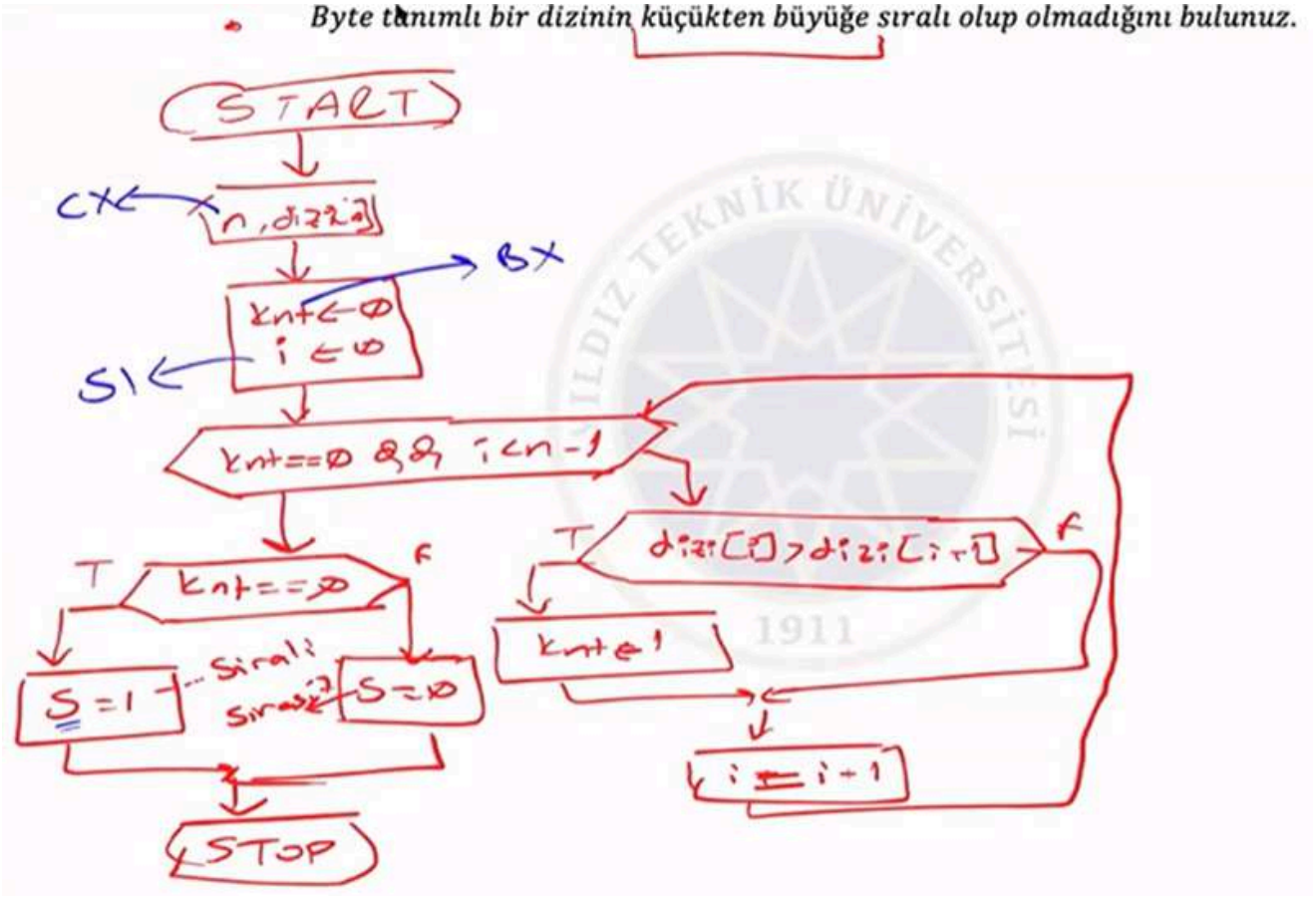
Debug Komutu	Kısa Açıklama
A (Assemble)	Assembly dilinde **doğrudan komut yazıp** belleğe çevirir.
D (Dump/Display)	Belirli bir bellek aralığındaki veriyi **hexadesimal (onaltılık)** ve ASCII formatında görüntüler.
E (Enter)	Belirli bir bellek adresine **veri (byte/word)** girmeye veya değiştirmeye yarar.
F (Fill)	Belirli bir bellek aralığını aynı byte değeriyle **doldurur**.
G (Go)	Programı, verilen bir adrese kadar **kesintisiz** çalıştırır.
H (Hex)	İki hexadesimal sayının **toplamını ve farkını** hesaplar.
L (Load)	Çalıştırılabilir bir dosyayı (örn: `.EXE`) veya diskin belirli bir bölümünü **belleğe yükler**.
M (Move)	Bellekteki bir veri bloğunu bir yerden başka bir yere **taşır**.
N (Name)	Yüklemek veya kaydetmek istediğiniz dosyanın **adını** belirtir.
P (Proceed)	Bir prosedür çağrısını (`CALL`) veya döngüyü (`LOOP`) **tek bir adımda** atlayarak çalıştırır. (`T`'ye benzer, ancak çağrıları atlar.)
Q (Quit)	`DEBUG` oturumundan çıkarak DOSBox komut istemine geri döner.
R (Register)	**Yazmaçların (Register)** mevcut içeriklerini görüntüler ve değiştirmeye izin verir.
S (Search)	Belirli bir bellek aralığında belirli bir **byte dizisini arar**.
T (Trace)	Programı **tek bir makine kodu** talimatı kadar çalıştırır (adım adım ilerleme).
U (Unassemble)	Bellekteki makine kodunu alıp, okunabilir **Assembly diline (mnemonics)** çevirir.
W (Write)	Bellekteki veriyi **diske (dosyaya)** kaydeder.

Ek Notlar:

- Debug ortamında görülen her değer başında h olmasa da hexadecimaldir.
- Farklı ileri seviye dilde asm kodu yazılırken asm kod bloğunun içine SADECE kod segmenti yazılır, veriler dilin kendisinde tanımlanmalıdır.
- Program sonunda yazılan RET stackten 1, RETF stackten 2 word çeker.
- LST dosyası ile kod segmentinin ne kadar yer kapladığı incelenebilir. (örneklerde RETF satırına bakmayı deneyin) Not: Baştaki 512 Bytelık header eklentisini de unutmyın.

Örnek 1

Byte tanımlı bir dizinin küçükten büyüğe sıralı olup olmadığını bulunuz.



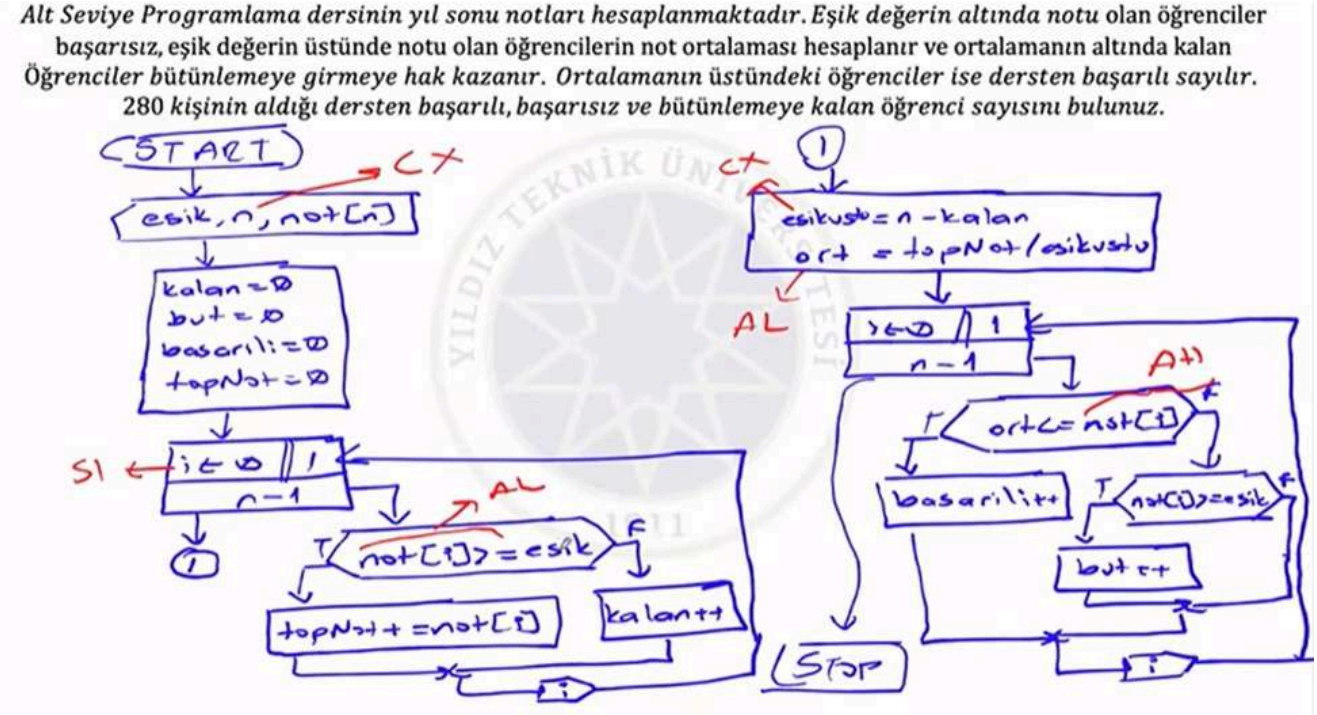

```
myds        SEGMENT PARA 'data'
dizi        DB 1, 2, 3, 4, 5
n           DW 5
S           DB 0
myds        ENDS
```

```
myss        SEGMENT PARA STACK 'stack'
            DW 20 DUP(?)
myss        ENDS
```

```
mycs        SEGMENT PARA 'code'
            ASSUME CS:mycs, SS:myss, DS:myds
MAIN        PROC FAR
            PUSH DS
            XOR AX, AX
            PUSH AX
            MOV AX, myds
            MOV DS, AX

            XOR SI, SI
            MOV BX, 0             ; ya da XOR BX, BX (hatırlatma)
            MOV CS, n
            DEC CX
            CMP BX, 0
            JNE son_if
            CMP SI, CX
            JGE son_if           ; İşaretli sayılar olabilir, Greater Equal
            MOV AL, dizi[SI]
            CMP AL, dizi[SI+1]   ; CMP data, data yapamazsın!
            JLE arttir
            MOV BX, 1
arttir:     INC SI
            JMP don
son_if:     CMP BX, 0
            JNE SON
            MOV S, 1             ; S sadece bir kere kullanıdığından
                                 ; registera atamaya gerek yok
            RETF
MAIN        ENDP
mycs        ENDS
            END MAIN
```

Örnek 2

Alt Seviye Programlama dersinin yıl sonu notları hesaplanmaktadır. Eşik değerin altında notu olan öğrenciler başarısız sayılır, eşik değerin üstünde notu olan öğrencilerin not ortalaması hesaplanır ve ortalamanın altında kalan öğrenciler bütünlemeye girmeye hak kazanır. Ortalamanın üstündeki öğrenciler ise dersten başarılı geçmiş sayılır.
280 kişinin aldığı dersten başarılı, başarısız ve bütünlemeye kalan öğrenci sayısını bulunuz.

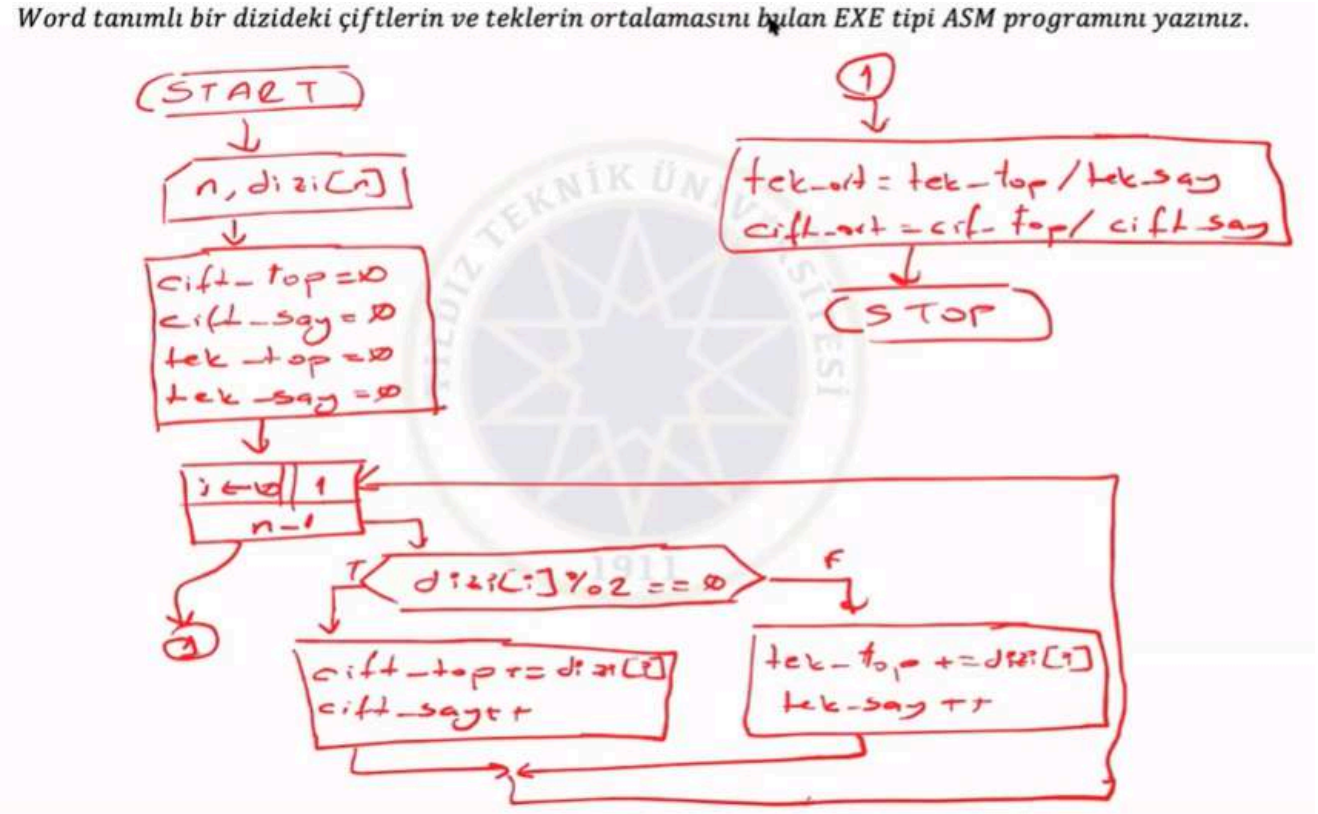


(bunu kendiniz yazmayı deneyin ya da videodan bakın (fç alt seviye prog. 6. video)

ğ

Örnek 3

Word tanımlı bir dizideki çiftlerin ve teklerin ortalamasını bulan EXE tipi ASM programını yazınız.

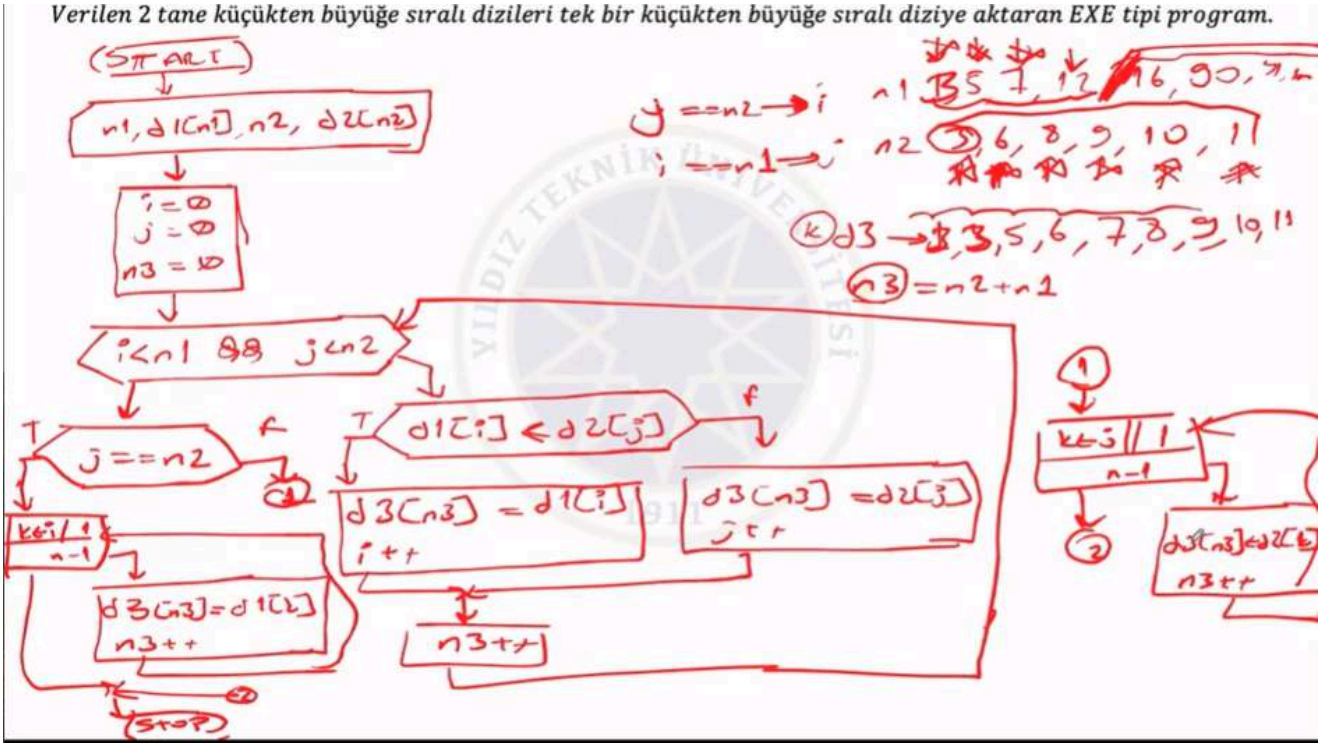


önce kendin dene sonra videoya bak (fç alt seviye prog. 7. video)

ğ

Örnek 4

Verilen 2 tane küçükten büyüğe sıralı dizileri tek bir küçükten büyüğe sıralı diziye aktaran EXE tipi programı yazınız.



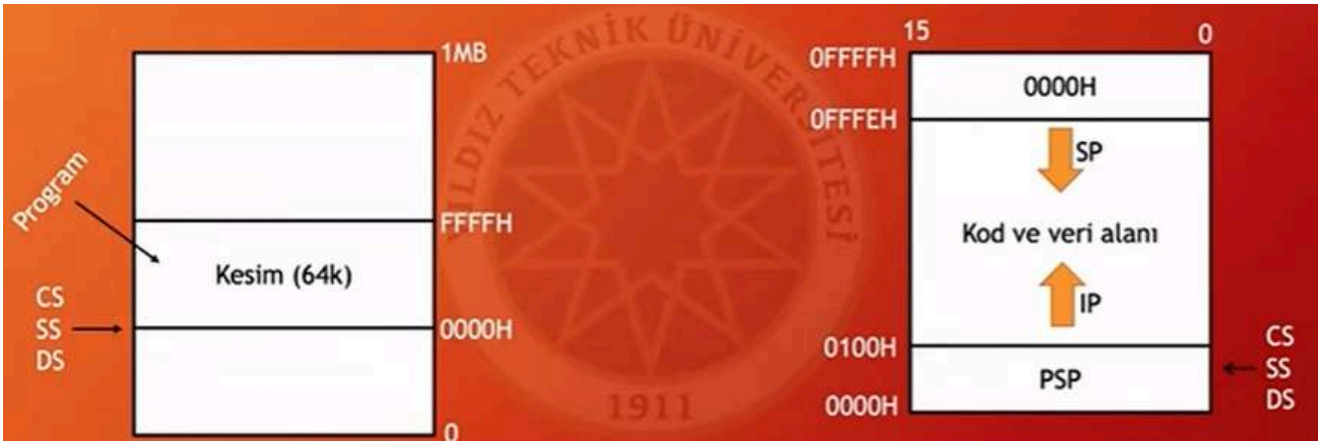
önce kendin dene sonra videoya bak (fç alt seviye prog. 7. video)

ğ

~~COM Tipi Program Yapısı(Sınavda Yok)~~

⚠️ *BU SAYFA SINAVDA YOK*

Compact programı ifade eder, çok ufak bellek yönetimine ihtiyaç duyulduğunda idealdir. 3 Segment de aynı alanı gösterir:



PSP (Program Segment Word)

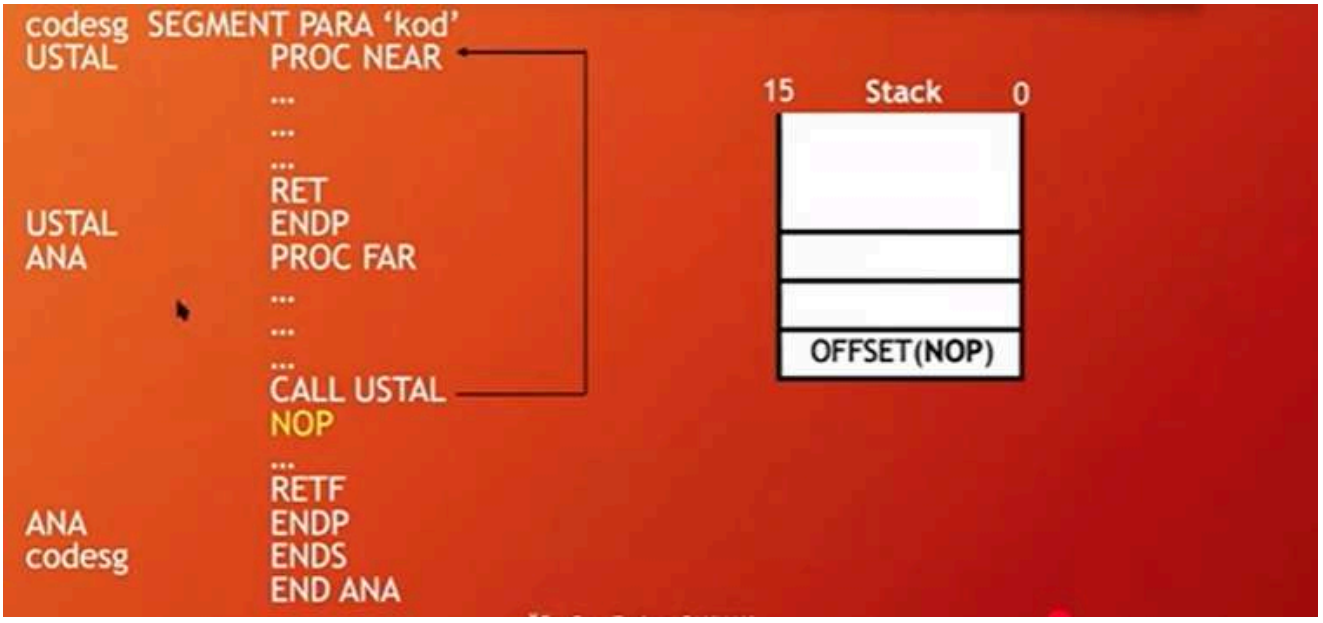
- Parametrelerin tutulduğu alan
- Program sonlandırıldığında gidilecek adreslerin tutulduğu alan
- Yardımcı aktarım değerlerinin tutulduğu alan
  COM tipi programda;
- Ana Yordam NEAR olarak tanımlanır.
- Geri dönüş adresleri yığına YERLEŞTİRİLMEZ.

```
codesg SEGMENT PARA 'kod'
           ORG 100h
           ASSUME CS: codesg, DS: codesg, SS:codesg
Bilgi:      JMP Basla
sayib       DB ?
sayiw       DW ?
           ...
BASLA       PROC NEAR
           ...
           RET
BASLA      ENDP
codesg      ENDS
           END Bilgi
```

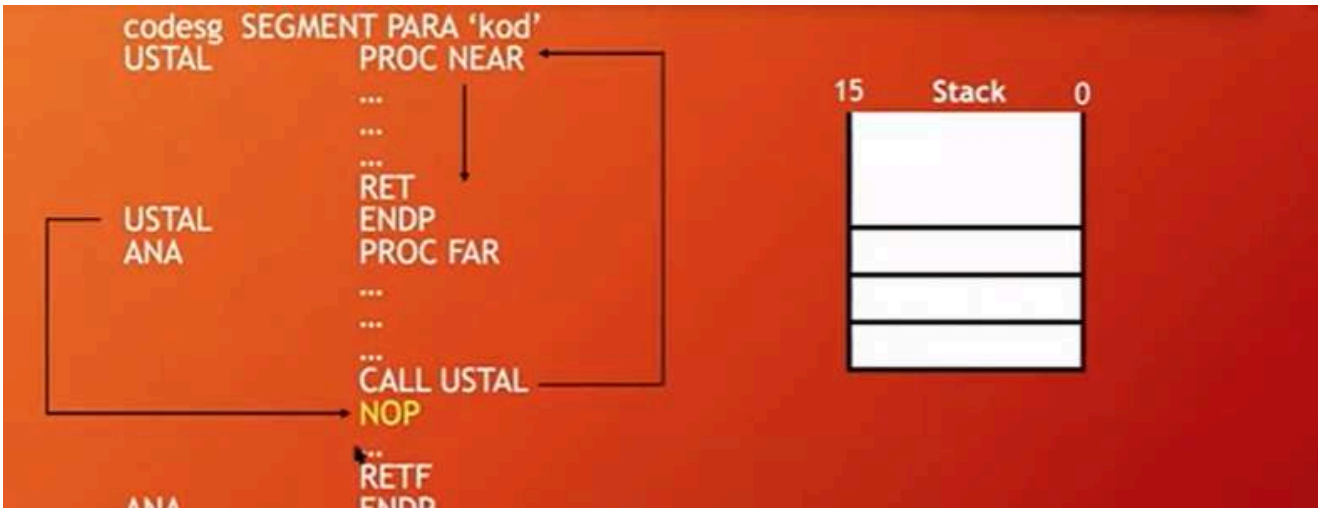
COM tipi programlar modern işletim sistemlerinde veya yazılımlarda kullanılmamaktadır.

Yordam ve Makro Kullanımları

Yordam Tanımı (Syntax)



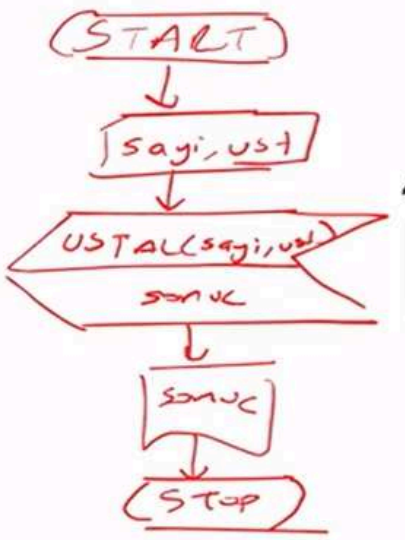
CALL komutu ile yordam çağırıldığında sıradaki satırın OFFSET değeri stacke atılır.



yordam bitince yığına atılan değer POP edilir ve oraya zıplanır.

Örnek 1: Intra-Segment Yordam

Sayı ve üst değerini ana yordamdan USTAL yordamına yollayan ANA yordamı ve üst alma işlemi gerçekleştiren USTAL yordamını EXE tipinde yazınız.



```
myss    SEGMENT PARA STACK 'yigin'
        DW 20 DUP(?)
myss    ENDS
myds    SEGMENT PARA 'veri'
sayi    DW 2
ust     DW 10
sonuc   DW ?
myds    ENDS
mycs    SEGMENT PARA 'kod'
        ASSUME CS:mycs, DS:myds, SS:myss
ANA     PROC FAR
        PUSH DS
        XOR AX, AX
        PUSH AX
        MOV AX, myds
        MOV DS, AX
        MOV BX, sayi
        MOV CX, ust
        CALL USTAL
        MOV sonuc, AX   ; Yordamdan bir değer alacağımızı ve AX e
                        ; kaydedileceğini düşünüyoruz
        RETF
ANA     ENDP
USTAL   PROC NEAR
        MOV AX, 1
L1:     MUL BX          ; AX*BX -> DX:AX
        LOOP L1
        RET
USTAL   ENDP
mycs    ENDS
        END ANA
```

DOSBOX çıktılarını inceleyerek yordam çağırıldığında ne olduğunu inceleyelim:

- SP değeri 0024.
- CALL Çağırıldı
- Call çağırılınca sıradaki komut olan MOV [0004], AX komutunun OFFSET değeri olan 0014h'ı stackte görmemiz lazım.

```
AX=076D  BX=0002  CX=000A  DX=0000  SP=0022  BP=0000  SI=000
DS=076D  ES=075A  SS=076A  CS=076E  IP=0018   NV UP EI PL ZR
076E:0018 B80100        MOV     AX,0001
-d ss:22
076A:0020       14 00 00 00 5A 07-00 00 00 00 00 00 00 00
076A:0030  02 00 0A 00 00 00 00 00-00 00 00 00 00 00 00 00
076A:0040  1E 33 C0 50 B8 6D 07 8E-D8 8B 1E 00 00 8B 0E 02
076A:0050  00 E8 04 00 A3 04 00 CB-B8 01 00 F7 E3 E2 FC C3
076A:0060  E4 40 50 8B C3 8C C2 05-0C 00 52 50 E8 C1 48 83
076A:0070  C4 04 50 8D 86 FA FE 50-E8 17 73 83 C4 06 8B B6
076A:0080  FA FE 81 E6 FF 00 C6 82-FB FE 00 2B C0 50 8D 86
076A:0090  FB FE 50 E8 08 6A 83 C4-04 0B C0 75 03 E9 A6 00
076A:00A0  C7 86
```

- yordam bitince RET çağırılır.
- RET çağırıldığında da stackten POP edilen adrese dönülmesi gerek

```
AX=0400  BX=0002  CX=0000  DX=0000  SP=0022  BP=0000  SI=0000  DI=0000
DS=076D  ES=075A  SS=076A  CS=076E  IP=001F   NV UP EI PL NZ NA PE NC
076E:001F C3            RET
-t

AX=0400  BX=0002  CX=0000  DX=0000  SP=0024  BP=0000  SI=0000  DI=0000
DS=076D  ES=075A  SS=076A  CS=076E  IP=0014   NV UP EI PL NZ NA PE NC
076E:0014 A30400        MOV     [0004],AX                       DS:00
-t
```

görüldüğü gibi yeni komutun OFFSET değeri 0014 oldu ve stack pointer değeri stackten bir değer POP edildiği için yeniden 2 arttı.

Örnek 2: Inter-Segment Yordam

BL ve BH üzerinden aldığı 2 değeri toplayan ve AX üzerinden sonucu döndüren harici yordamı ve bu harici yordamı çağıran ana yordamı EXE tipinde yazınız.

Bu örneği iki dosyayla yapacağız:
dosya1.asm:

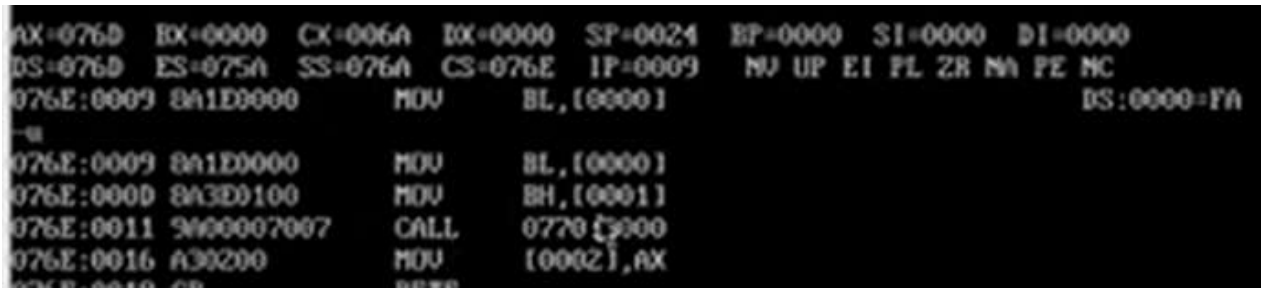
```
EXTRN TOPLAMA:FAR                  ; dışarıdan alınan yordamı belirtmek için
                                   ; koyulmazsa TOPLAMA tanınmaz,
                                   ; masm hata verir
myss    SEGMENT PARA STACK 'yi'
        DW 20 DUP(?)
myss    ENDS

myds    SEGMENT PARA 've'
sayi1   DB 17
sayi2   DB 29
sonuc   DW ?
myds    ENDS

mycs    SEGMENT PARA 'ko'
        ASSUME CS:mycs, DS:myds, SS:myss
ANA     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
        RETP
ANA     ENDP
mycs    ENDS
        END ANA
```

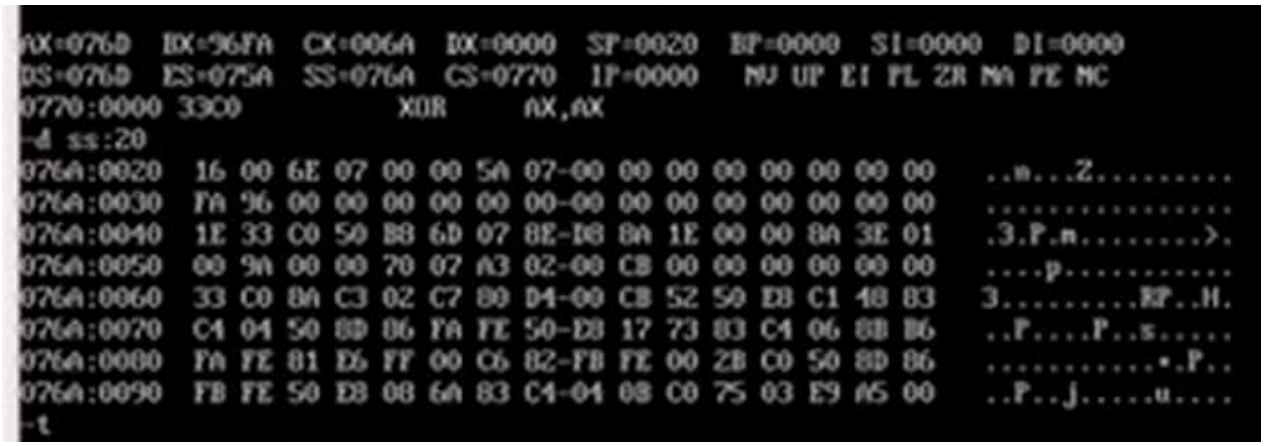
dosya2.asm:

```
PUBLIC TOPLAMA              ; Bunu da link hatası almamak için yazmamız lazım
codesg  SEGMENT PARA 'kod'
        ASSUME CS:codesg
TOPLAMA PROC FAR
        XOR AX, AX
        MOV AL, BL
        ADD AL, BH
        ADC AH, 0
        RETF
TOPLAMA ENDP
codesg  ENDS
        END
```

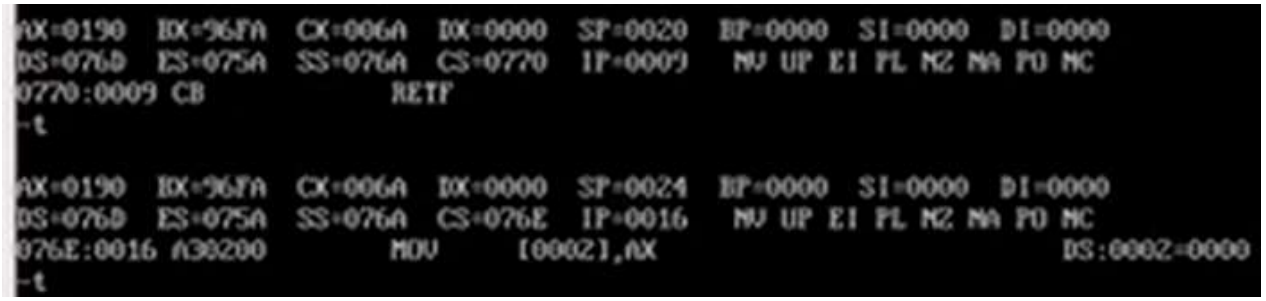
- DOSBOX'da sırayla `masm dosya1.asm` ve `masm dosya2.asm` yazıyoruz.
- `dir dosya*` dersek bu iki dosyayı görüntüleyebiliriz
- link ederken ikisini beraber link etmek gerekir:
- `link dosya1.obj + dosya2.obj;`
- (noktalı virgül koyunca dosya adı sormak yerine direkt ilk yazılan dosyanın adında exe dosyası oluşturur.)

- Bu sefer CALL yapıldığında hem harici kod segmentinin değeri hem de OFFSET değeri değişiceği için yığına iki farklı değer atılacak.



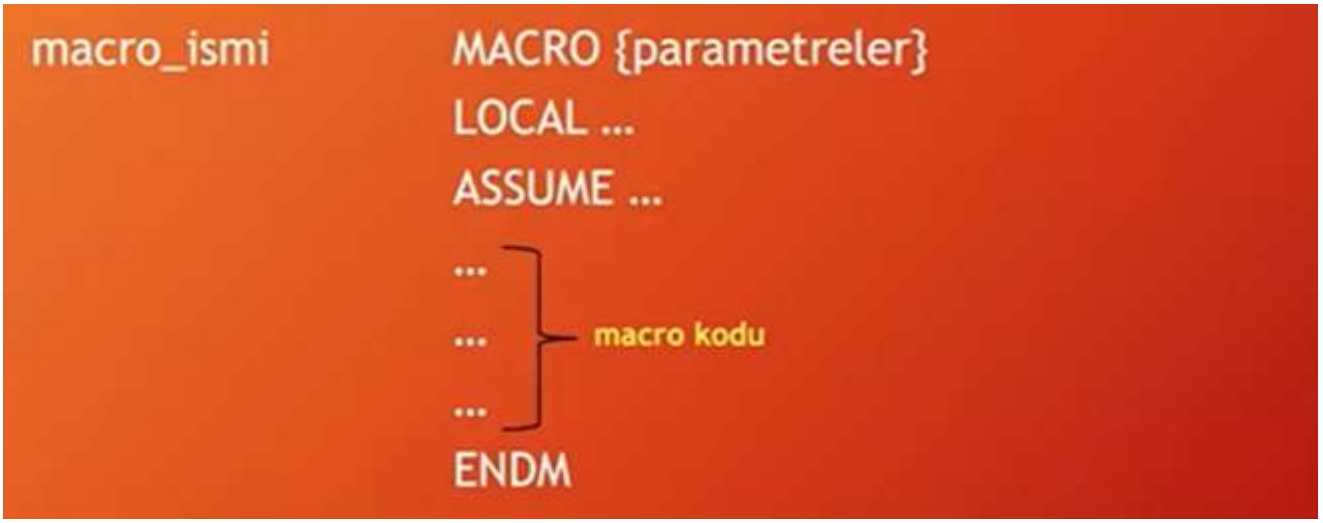


- RETF ile dönüldüğü için de alınan 2 eleman POP edilecek.



SP değişimine ve yeni CS:IP ikilisine dikkat!
(videoları keşke 1080p de izleseymişim...)

Makro Tanımı (Syntax)



Derleme (assembly) aşamasında, makronun çağrıldığı her yere, makronun tanımında yer alan kod bloğu **doğrudan kopyalanıp yapıştırılır**.

- LOCAL: Makro içinde bir etiket tekrar kullanıldığında derleyici hatasını önlemek için, makronun her çağrılışında o etiketin benzersiz sayılmasını sağlar. (Örnek: LOCAL L1 denildiğinde L1 Makro dışında da yeniden kullanılabilir)
- ASSUME makro içinde segment yazmaçlarının varsayımlarını belirlemek için kullanılabilir.
- Makro bittiğinde ENDM kullanılır ve segmentler ile yordamın aksine ENDM başına makronun ismi yazılması gerekmez.
- Makroların kullanılmadan önce tanımlanması gerekir.

⭐⭐⭐⭐⭐	**Makro (Macro)**	**Yordam (Procedure)**
Çalışma Zamanı	**Derleme (Assembly) anında** genişler ve koda eklenir.	**Çalışma (Run Time) anında** `CALL` ile çağrılır.
Program Boyutu	Her çağrıldığında kod kopyalandığı için **program boyutunu artırır.**	Tek bir kod bloğu vardır, `CALL` komutu eklendiği için **boyut daha küçüktür.**
Hız (Performans)	`CALL/RET` yükü olmadığı için **daha hızlıdır.**	`CALL` ve `RET` komutları ek yük getirir, yığın (Stack) kullanılır, **biraz daha yavaştır.**
Veri Aktarımı	Parametreler **metin olarak** doğrudan koda yerleştirilir.	Parametreler genellikle **yazmaçlar (Register)** veya **yığın (Stack)** aracılığıyla aktarılır.

Örnek 3 - Macro

n elemanlı bir dizinin elemanlarını 2'ye bölen MAIN yordamın içinde dizinin en küçük elemanını bulan MACRO'yu COM tipi program için yazınız.

Sınavda yok, COM tipi olduğu için yazmıyorum (FÇ Alt Seviye Prog. 8. Video: 2:18:05)

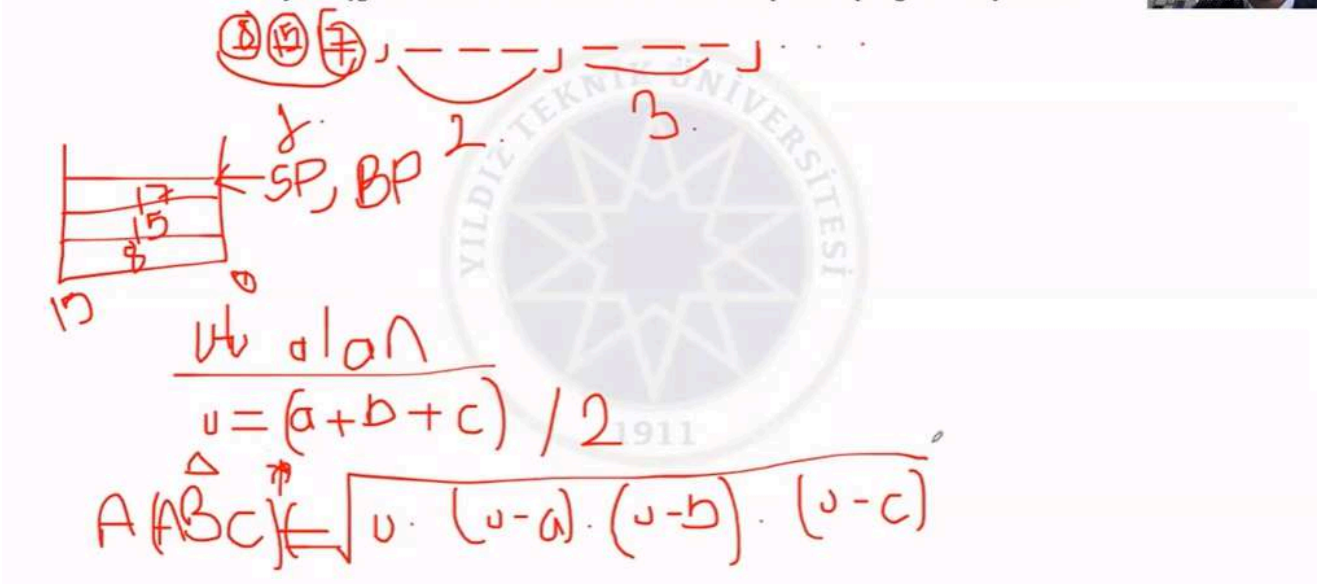
Parametre Aktarma Yöntemleri

yazmaç sayısından fazla parametre kullanmamız gerektiğinde ne yapmalıyız?

- 3 adet yöntem var.
  - Yazmaç üzerinden parametre aktarımı
  - Yığın (stack) üzerinden parametre aktarımı
  - EXTRN/PUBLIC komutları ile parametre aktarımı

Örnek 1 - Stack Üzerinden Parametre Aktarma

Ardışık 3 elemanı üçgen kenarı olan bir dizide n tane üçgenin kenarları tutulmaktadır. Üçgen kenar üzerinden alan ve bu üçgenin alanının karesini AX register'ı üzerinden döndüren bir harici yordam en büyük üçgenin alanının karesini bulan EXE tipi ASM programını yazınız.



Soru okunmuyorsa → Alt Seviye Programlama Dersi - 9. Hafta 30. dakikalara git

main.asm

```
       EXTRN ALAN_BUL
myss    SEGMENT PARA STACK 's'
        DW 20 DUP(?)
myss    ENDS

myds    SEGMENT PARA 'd'
kenarlarn DB 6, 8, 5, 9, 4, 8, 2, 2, 3
n         DW 3
enbykalan DW 0
myds    ENDS

mycs    SEGMENT PARA 'k'
        ASSUME CS:mycs, DS:myds, SS:myss

ANA     PROC FAR
        PUSH DS
        XOR AX, AX
        PUSH AX
        MOV AX, myds
        MOV DS, AX

        MOV CX, n
        XOR SI, SI
L1:     PUSH kenarlar[SI]
        PUSH kenarlar[SI+2]
        PUSH kenarlar[SI+4]
        CALL ALAN_BUL
        CMP AX, enbykalan
        JB kucuk
        MOV enbykalan, AX
kucuk:  ADD SI, 6
        LOOP L1
        RETP

ANA     ENDP
mycs    ENDS
        END ANA
```

ÖNEMLİ

HARİCİ YORDAMLARINIZIN İÇİNDE KULLANILACAK REGISTERLARIN DEĞERLERİNİ KULLANMADAN ÖNCE STACK'E ATMANIZ VE YORDAMDAN DÖNMEDEN HEMEN ÖNCE STACKTEN GERİ ÇEKMENİZ BEKLENİR.

yardimci.asm

```
        PUBLIC ALAN_BUL
mycode  SEGMENT 'kod'
        ASSUME CS:mycode
ALAN_BUL  PROC FAR
        PUSH BX      ; Kullanılacak registerları stacke at
        PUSH CX
        PUSH DI
        PUSH BP
        PUSH DX

        MOV BP, SP
        XOR AX, AX
        ADD AX, [BP+12]
        ADD AX, [BP+14]
        ADD AX, [BP+16]

        SHR AX, 1
        MOV BX, AX
        SUB BX, [BP+14]
        MOV CX, AX
        SUB CX, [BP+16]
        MOV DI, AX
        SUB DI, [BP+18]

        MUL BX       ; AX * BX -> DX:AX
        MUL CX       ; AX * CX -> DX:AX
        MUL DI

        POP DX       ; Kullanılan Registerları geri çek
        POP BP
        POP DI
        POP CX
        POP BX

        RETF 6     ; ÇOK ÖNEMLİ!!! STACKTEN 6 BYTE POP ETTİ
ALAN_BUL  ENDP
mycode    ENDS
        END
```

DAHA DA ÖNEMLİ!

Kodda kullanılan şu satırlara dikkat:
PUSH kenarlar[SI]
PUSH kenarlar[SI+2]
PUSH kenarlar[SI+4]
Stackteki bu değerlerin de program bitmeden önce pop edilmesi gerekir!
Yoksa end komutunun alması gereken adreslere program sonunda erişilemez.

Bunun için iki yol var:

1. Ana program dosyasında bu satırları CALL sonrası kullanılmayan bir registera POP etmek:

```
...nasm
L1:     PUSH kenarlar[SI]
        PUSH kenarlar[SI+2]
        PUSH kenarlar[SI+4]
        CALL ALAN_BUL
        POP BX
        POP BX
        POP BX
        CMP AX, enbykalan
        JB kucuk`
...
```

2. gönderilen yordam içinde RETF komutunun sonuna silinmesi gereken alanı belirterek silmek: (örnekteki gibi)

```
         ...
          POP BX

         RETF 6     ; ÇOK ÖNEMLİ!!! STACKTEN 6 BYTE POP ETTİ
ALAN_BUL  ENDP
mycode    ENDS
          END
```

EK: PUBLIC ve EXTERN unutulursa nolur?

Dosbox aşağıdaki hatayı verecektir:

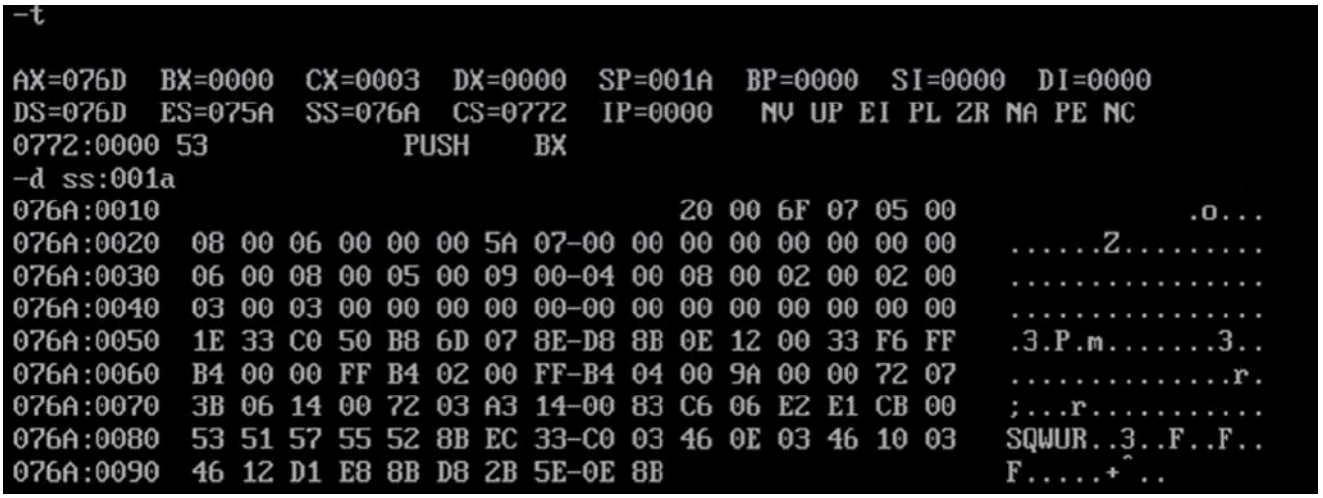
```
Directory of C:\ASM\.
171220A  ASM                531 17-12-2020 10:15
171220B  ASM                487 17-12-2020 10:15
    2 File(s)              1,018 Bytes.
    0 Dir(s)         262,111,744 Bytes free.

C:\ASM>masm 171220a.asm;
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988.  All rights reserved.

171220a.asm(24): error A2009: Symbol not defined: ALAN_BUL

  49918 + 463487 Bytes symbol space free

      0 Warning Errors
      1 Severe  Errors
```

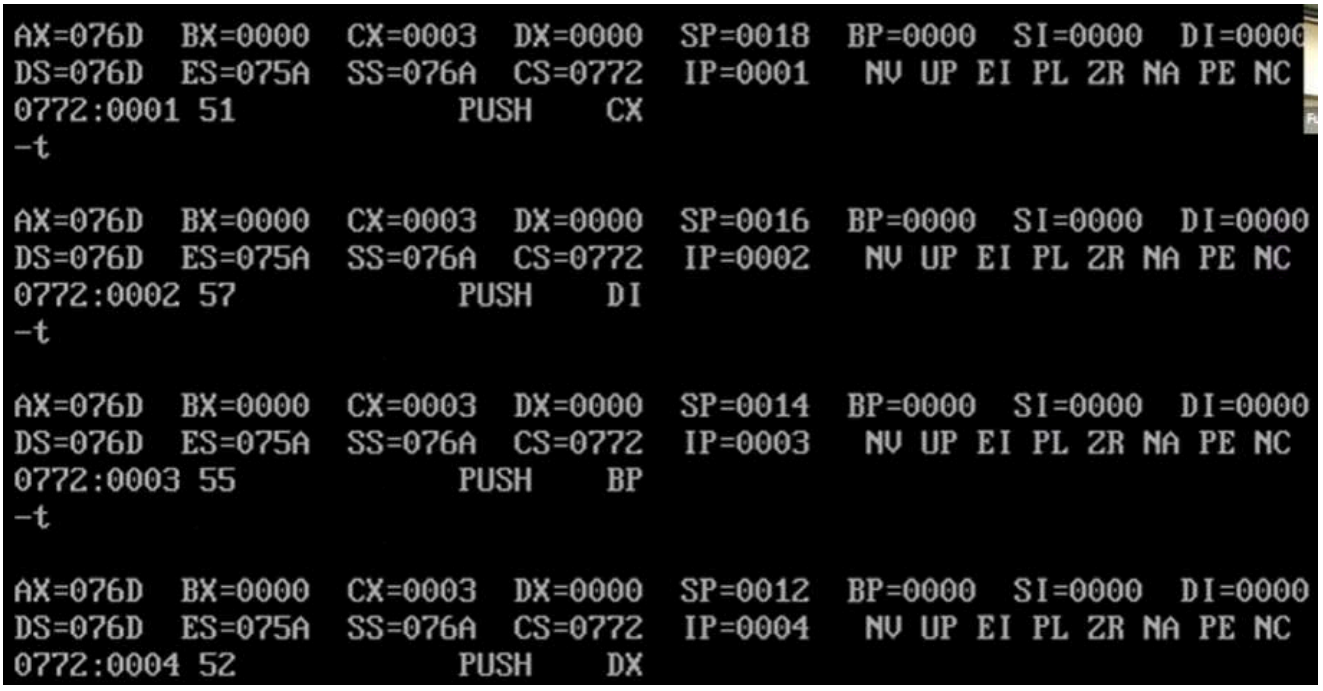
EXTRN ALAN_BUL ve PUBLIC ALAN_BUL satırlarını kodlara eklemeyi unutmayın

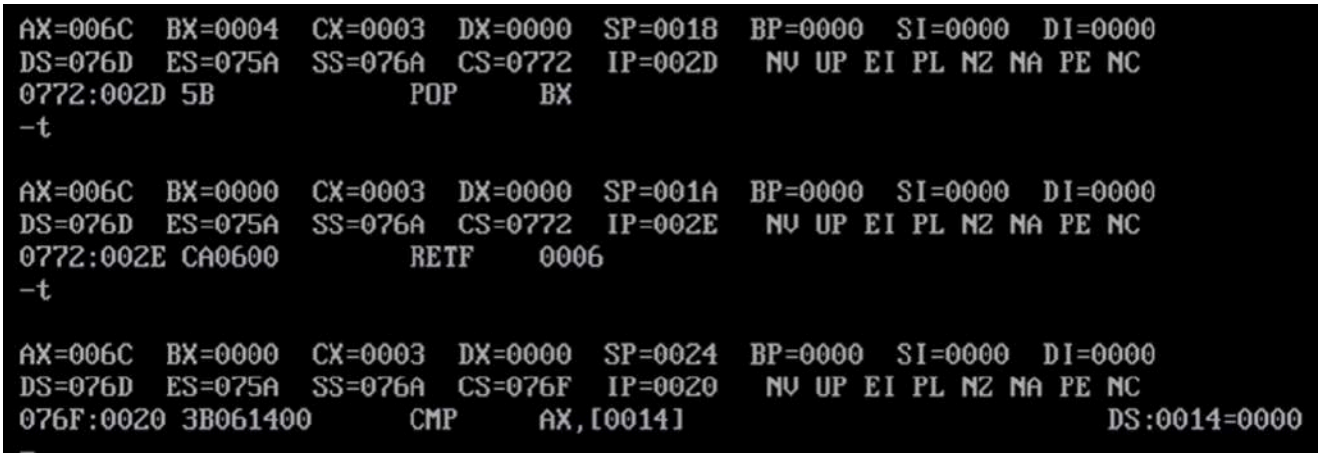
DOSBOX çıktılarını incelemek programı anlamak için önemlidir

```
AX=076D  BX=0000  CX=0003  DX=0000  SP=0024  BP=0000  SI=0000  DI=0000
DS=076D  ES=075A  SS=076A  CS=076F  IP=000F   NV UP EI PL ZR NA PE NC
076F:000F FFB40000      PUSH    [SI+0000]                     DS:0000=0006
-t

AX=076D  BX=0000  CX=0003  DX=0000  SP=0022  BP=0000  SI=0000  DI=0000
DS=076D  ES=075A  SS=076A  CS=076F  IP=0013   NV UP EI PL ZR NA PE NC
076F:0013 FFB40200      PUSH    [SI+0002]                     DS:0002=0008
-t

AX=076D  BX=0000  CX=0003  DX=0000  SP=0020  BP=0000  SI=0000  DI=0000
DS=076D  ES=075A  SS=076A  CS=076F  IP=0017   NV UP EI PL ZR NA PE NC
076F:0017 FFB40400      PUSH    [SI+0004]                     DS:0004=0005
-t

AX=076D  BX=0000  CX=0003  DX=0000  SP=001E  BP=0000  SI=0000  DI=0000
DS=076D  ES=075A  SS=076A  CS=076F  IP=001B   NV UP EI PL ZR NA PE NC
076F:001B 9A00007207    CALL    0772:0000
```

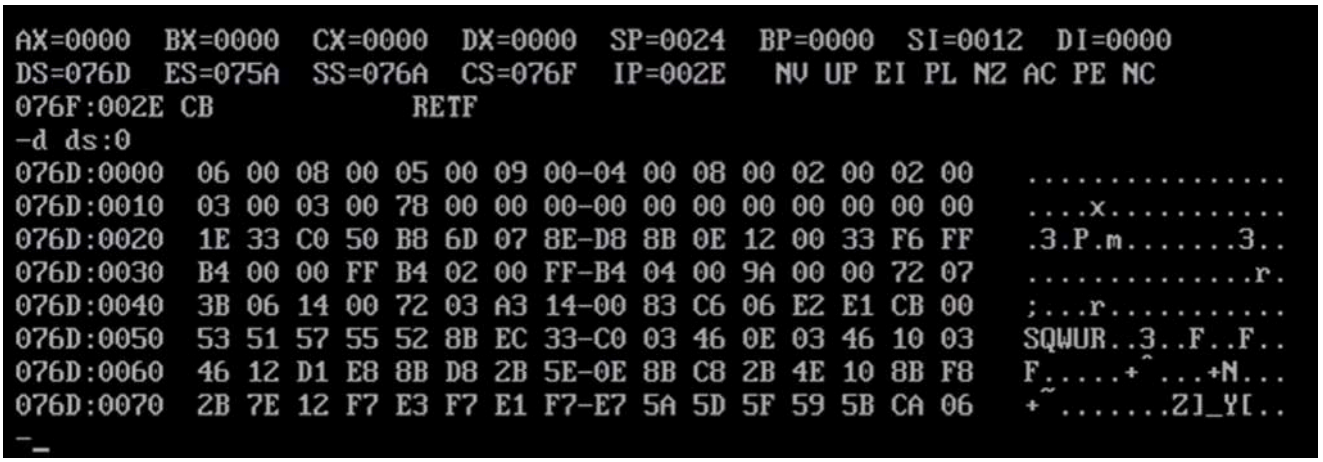
```
-d ss:001e
076A:0010                                           05 00                ..
076A:0020  08 00 06 00 00 00 5A 07-00 00 00 00 00 00 00 00   ......Z.........
076A:0030  06 00 08 00 05 00 09 00-04 00 08 00 02 00 02 00   ................
076A:0040  03 00 03 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  1E 33 C0 50 B8 6D 07 8E-D8 8B 0E 12 00 33 F6 FF   .3.P.m.......3..
076A:0060  B4 00 00 FF B4 02 00 FF-B4 04 00 9A 00 00 72 07   ..............r.
076A:0070  3B 06 14 00 72 03 A3 14-00 83 C6 06 E2 E1 CB 00   ;...r...........
076A:0080  53 51 57 55 52 8B EC 33-C0 03 46 0E 03 46 10 03   SQWUR..3..F..F..
076A:0090  46 12 D1 E8 8B D8 2B 5E-0E 8B C8 2B 4E 10         F.....+^...+N.
```





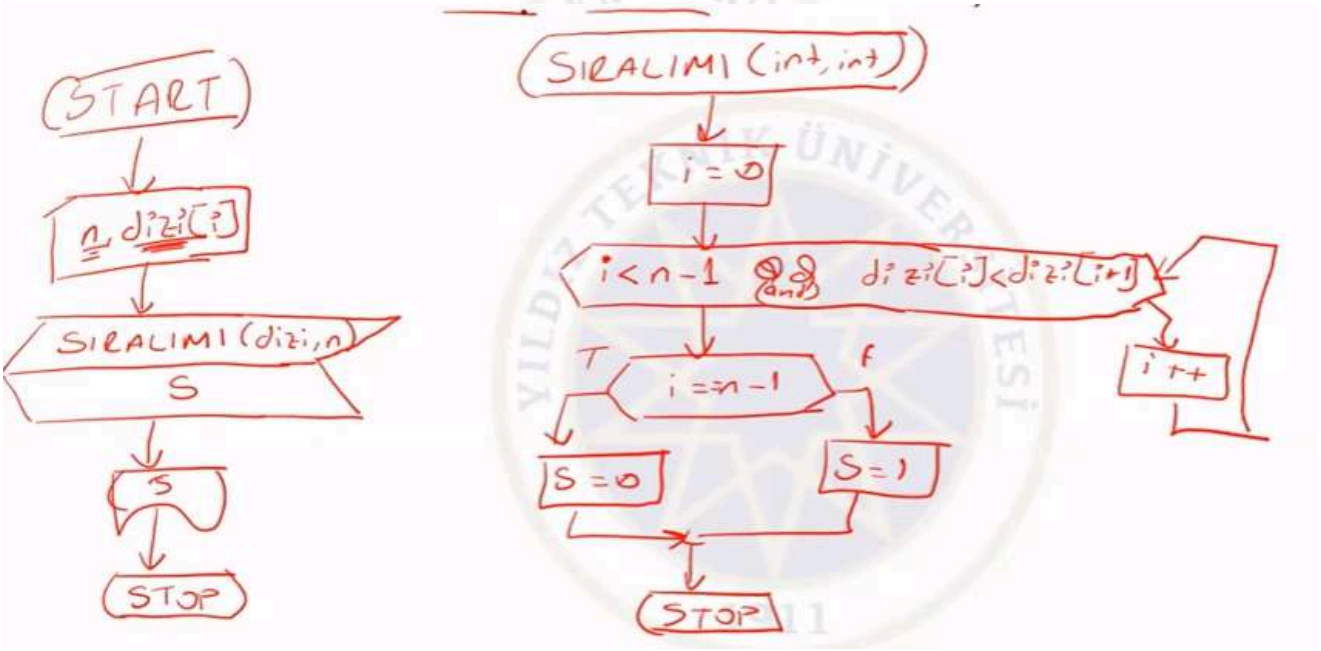


program sonu


AX=0000 BX=0000 CX=0000 DX=0000 SP=0024 BP=0000 SI=0012 DI=0000
DS=076D ES=075A SS=076A CS=076F IP=002E NV UP EI PL NZ AC PE NC
076F:002E CB RETF
-d ds:0
076D:0000 06 00 08 00 05 00 09 00-04 00 08 00 02 00 02 00
076D:0010 03 00 03 00 78 00 00 00-00 00 00 00 00 00 00 00
076D:0020 1E 33 C0 50 B8 6D 07 8E-D8 8B 0E 12 00 33 F6 FF
076D:0030 B4 00 00 FF B4 02 00 FF-B4 04 00 9A 00 00 72 07
076D:0040 3B 06 14 00 72 03 A3 14-00 83 C6 06 E2 E1 CB 00
076D:0050 53 51 57 55 52 8B EC 33-C0 03 46 0E 03 46 10 03
076D:0060 46 12 D1 E8 8B D8 2B 5E-0E 8B C8 2B 4E 10 8B F8
076D:0070 2B 7E 12 F7 E3 F7 E1 F7-E7 5A 5D 5F 59 5B CA 06
-_

Örnek 2 - EXTRN/PUBLIC ile Parametre Aktarma

Verilen bir dizinin küçükten büyüğe sıralı olup olmadığını bulan harici yordamı yazınız. Harici yordam verilere EXTRN ve PUBLIC komutları ile erişmelidir.



```
EXTRN SIRALIMI:FAR
PUBLIC dizi, n

myss    SEGMENT PARA STACK 'sss'     ; SS yazılırsa hata verir
        DW 20 DUP(?)
myss    ENDS

myds    SEGMENT PARA 'd'
n       DB 0
dizi    DB 12, 14, 16, 18, 20, 22, 24
myds    ENDS

mycs    SEGMENT PARA 'k'
        ASSUME CS:mycs, DS:myds, SS:myss

ANA     PROC FAR
        PUSH DS
        XOR AX, AX
        PUSH AX
        MOV AX, myds
        MOV DS, AX

        CALL SIRALIMI
        CMP AL, 0
        JZ sirali
        MOV s, 1
sirali: RET
ANA     ENDP
mycs    ENDS
        END ANA
```

```
PUBLIC SIRALIMI
EXTRN dizi:BYTE, n:WORD

mycode    SEGMENT PARA 'test'
          ASSUME CS:mycode
SIRALIMI  PROC FAR
          PUSH SI
          PUSH CX
          XOR AX, AX
          XOR SI, SI
          MOV CX, n
          DEC CX

don:      CMP SI, CX
          JAE sirali
          MOV AH, dizi[SI]
          CMP AH, dizi[SI+1]
          JG sirasiz
          INC SI
          JMP don

sirasiz:  MOV AL, 1
sirali:   POP CX
          POP SI
          RET
SIRALIMI  ENDP
mycode    ENDS
          END
```

Kriter	1. Yazmaçlar (Registers)	2. Ortak Değişkenler (Shared Data)	3. Yığın (Stack)
Kullanım Senaryosu	En küçük (1-4 adet) ve basit parametreler.	Birden fazla modül arasında büyük veri yapıları veya diziler.	Orta/çok sayıda parametre, yerel değişkenlerin korunması.
Hız / Performans	**En Hızlı.** Bellek erişimi olmadığından çok hızlıdır.	Hızlı. Yordam bir kez DS'i ayarlarsa erişim hızlıdır.	Yavaş. Her parametre için `PUSH/POP` veya `BP` ile dolaylı erişim gerektirir.
Çağıran Yöntemi	Değerleri yazmaçlara kopyalar.	Bellekteki değişkenlerin adını kullanır.	Değerleri yığına iteler.

Çağıran Program Talimatı	`MOV BL, 10H` (1. Parametre)
	`MOV BH, 05H` (2. Parametre)
	`CALL YORDAM`
		`EXTRN dizi:BYTE, n:WORD`
		`CALL YORDAM` (Veri zaten bellekte hazır)
			`PUSH DEGER1`
			`PUSH DEGER2`
			`CALL YORDAM`

Yordamın Erişim Yöntemi	Parametreleri doğrudan yazmaçlardan okur.	**DS** ayarlandıktan sonra değişkenlere isimleriyle erişir.	BP'yi ayarlayarak yığın adreslemesi kullanır.
Çağrılan Talimat	`MOV AL, BL` (1. Parametreyi oku) `ADD AL, BH` (2. Parametreyi kullan)	`MOV CX, n` (Eleman sayısını oku) `MOV AL, dizi[SI]` (Dizi elemanını oku)	`MOV BP, SP` `MOV AX, [BP+6]` (1. Parametreyi oku)
Avantaj	En yüksek performans.	Modüller arası veri paylaşımı kolaydır.	Yordamın durumunu (yazmaçlar, parametreler) korur.

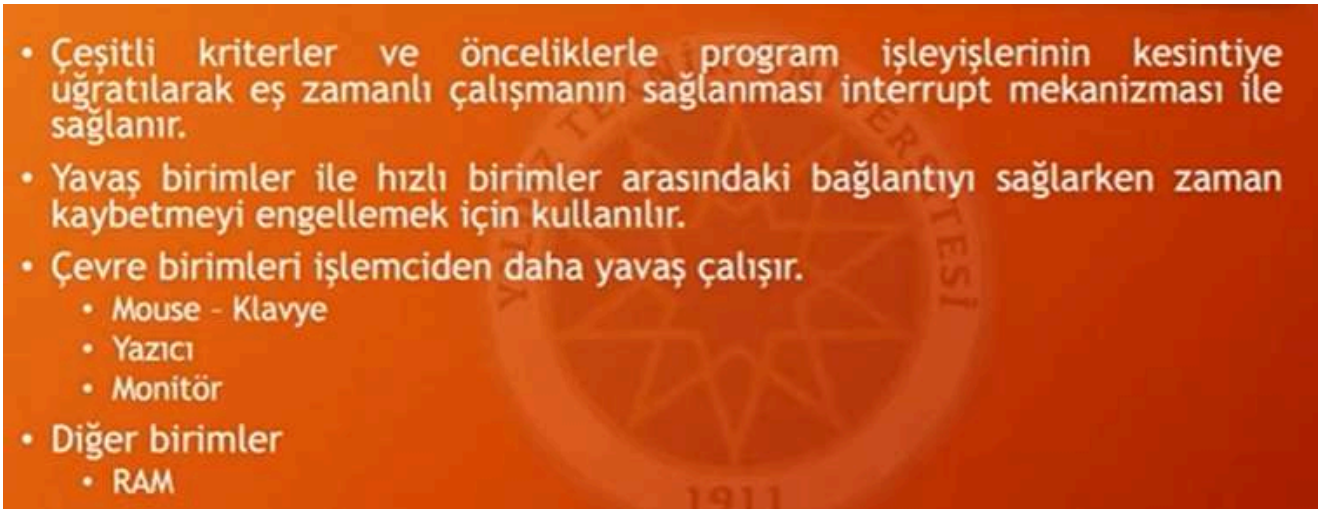
~~Ortak Kesim Kullanımı (Sınavda Yok)~~

(Sınavda Yok)

~~Kesme (Interrupt)(Sınavda Yok)~~

(işlenmedi (?) mikroişlemci kısmında işlenecek)

İşlemcinin zamanını farklı donanımlara bölmek için interrupt mekanizması hayati önem taşır.



Alt Seviye Programlama Dilinin Yüksek Seviyeli Diller ile Kullanılması

- Assembly yordamlarının C fonksiyonu ile benzer kullanılabileceği gibi aynı dosya içerisinde hem assembly hem de C kodu yazılabilmektedir. Ancak platform farklılıklarından kaynaklanan yazım değişiklikleri bilinmeli.
- C ve Assembly dillerinde yazılan kodların aynı tipteki büyüklükte (32-bit, 64-bit, vb.) derlenmesi veya en azından üretecekleri nesne dosyalarının aynı tipte olması sağlanmalıdır.
- Eğer fonksiyon kullanımı tercih edilecekse parametre aktarmanın ne şekilde gerçekleştirileceği önem arz etmektedir:
  ⭐ *C fonksiyonundan Assembly yordamı çağırıldığında aktarılan parametreler; aktarım sırasının tam tersi olacak şekilde (sağdan sola doğru) yığına atılırlar. Yığından aldığı verileri işleyen Assembly yordamı eğer tek bir değer döndürecekse akümülatör yazmacı (kullanılan platforma göre değişiklik göstermekle birlikte AX (16-bit), EAX (32-bit) veya RAX (64-bit) üzerinden değer döndürmektedir. Örnek 1'de integer'ın 4 byte olduğu bir platform için yazılmış bir C fonksiyonu prototipi ve bu prototipin çağırılması sonucu parametrelerin ne şekilde yığına atılacağı gösterilmiştir. Fonksiyon bir tamsayı değeri döndüreceği için bu da EAX yazmacı üzerinden gerçekleştirilecektir.* (E = Extended)

Örnek 1: int Toplama(int* dizi, int boyut, int* sonucdizisi);

- *Çağırılan yordamların "near" tipinde olduğu bilinmeli ve yığına bu sebeple bir göreceli konum (offset) değerinin atıldığı unutulmamalıdır. ❗ C programlama dilinin yapısı gereği yığın üzerinden parametre olarak gönderilen değerler, yine C tarafında yığından kaldırılmaktadır. Bu sebeple Assembly yordamından dönerken bu parametrelerin kaldırılması ile ilgili herhangi bir işlem yapılmamalıdır.*
- Farklı platformlarda değişiklikler göstermekle birlikte unutulmamalıdır ki; bir assembly program (COM tipinde hepsi bir arada olsa bile) veri, yığın ve kod kesimlerinden oluşur. Eğer bir assembly yordamı, bir C fonksiyonu tarafından çağırıldıysa, yığın olarak çağıran programın (yani C programının) yığını kullanılır. Eğer yordamda veri kullanımına ihtiyaç duyulmuşsa, ihtiyaç duyulan veriler, veri kesiminde tanımlandıktan sonra kod kesiminde kullanılabilir.
- Programlar içerisinde kullanılan değişkenler, bellekte birer adresi göstermektedir. Bu sebeple inline assembly kodu yazarken C değişkenleri olduğu gibi kullanılabilir (okunup, yazılabilinir). Yani ortak veri kesimi kullanımı gerçekleştirilmektedir.
- Assembly yordamına int'den büyük değerler (double vb.) gönderilmek istenirse, unutulmamalıdır ki, veri int'den büyük tanımlı olsa bile o verinin bellekteki adresi bir int boyutundadır. Bu da, istenilen her büyüklükteki verinin (kaynakların kapasitesi göz önünde bulundurularak) assembly yordamına gönderilebilmesine imkan sağlamaktadır.

~~Linux (Ubuntu) Ortamında Assembly ve C Dillerinin Birlikte Kullanımı~~

Windows Ortamında Visual Studio 2015 Kullanılarak C Programı İçerisinden Assembly Yordamı Çağırma

.... buna uygulama dokümanından bakmak daha iyi olur, zaten ödev için

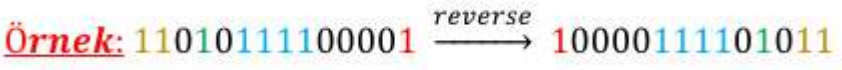
Visual Studio 2015 Ortamında C Dosyası İçerisinde Inline Assembly Komutu Yazma Örneği

Visual Studio ortamında c kodunun içerisinde;

```C++
__asm {
   ;Assembly kodları
}
```

komutu kullanılarak Assembly kodu yazılması sağlanmaktadır. Ancak farklı tür IDE ve platformlarda farklı notasyonların (Intel ve AT&T olarak yayğın kullanılan 2 notasyon vardır. Ders kapsamında Intel notasyonu kullanılmıştır.) kullanılabileceğini unutmamak gerekir.

C'de bir dizide bulunan elemanları dizinin ortasına göre tersine çevirmek kolay bir işlem sayılır. Ancak integer (32-bit) tanımlı bir dizide, dizinin her bir elemanının ikili (binary) gösterimindeki digit'lerin tersine çevrilmesi o kadar basit değildir. Bu tarz Assembly dilinde C diline göre daha kolay olan işlemlerin inline Assembly kodu olarak yazılması işleyişi oldukça kolaylaştıracaktır.
ikili digit'ler üzerinde tersine çevirme (reverse) işleminin nasıl olduğu aşağıda örnek ile verilmiştir.



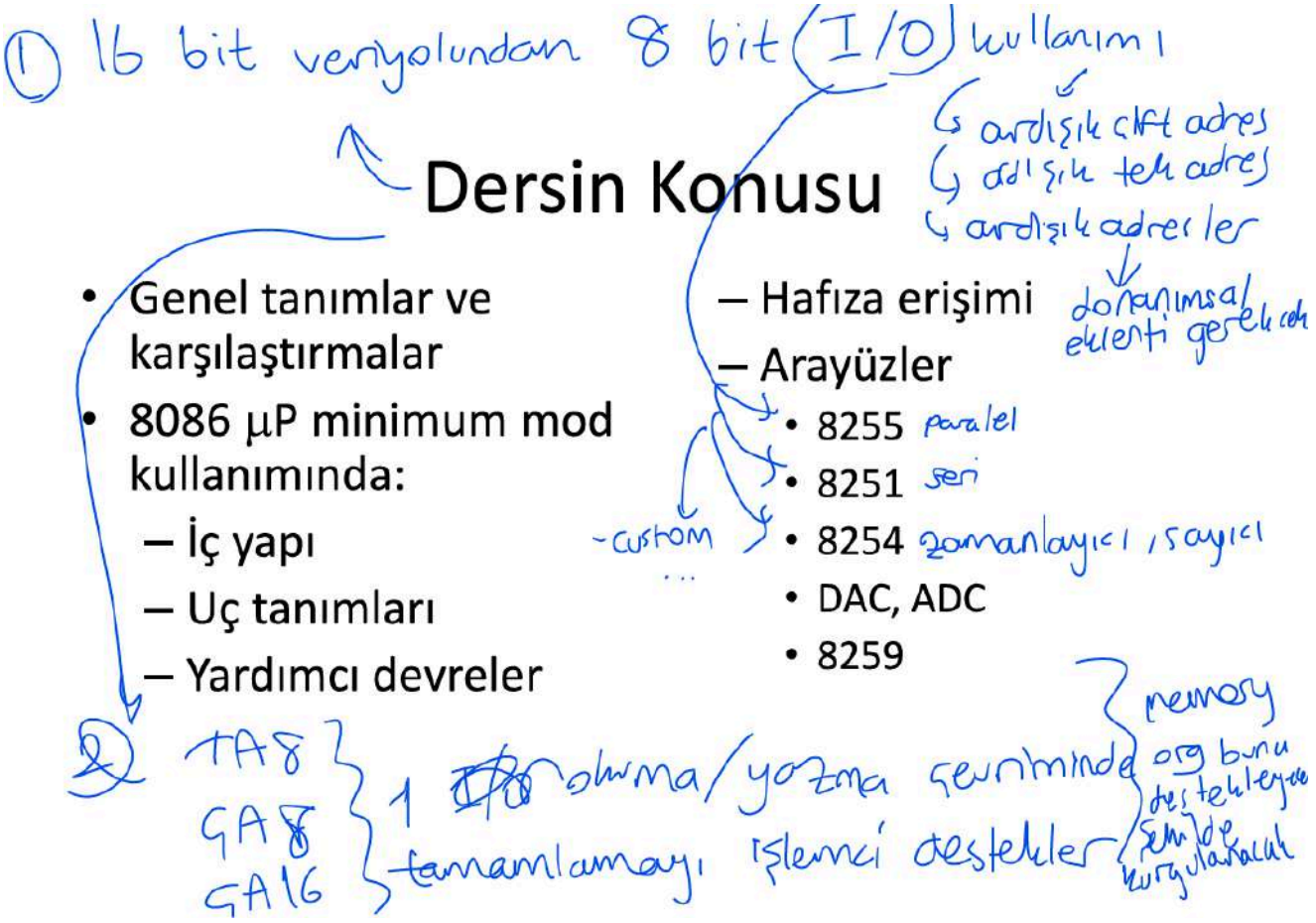
C++

```
#include <stdio.h>
#include <stdlib.h>
const int n = 10;

int main() {
    int i, dizi[n];
    for (i = 0; i < n; i++) {
        dizi[i] = i + 1;
        printf("%8x ", dizi[i]);
    }
    printf("\n\n");

    __asm {
        MOV ECX, n
        XOR ESI, ESI

L2:     MOV EAX, dizi[ESI]
        PUSH ECX
        MOV ECX, 32

L1:     SHR EAX, 1
        RCL EBX, 1
        LOOP L1
        POP ECX

        MOV dizi[ESI], EBX
        ADD ESI, 4
        LOOP L2
    }

    for (i = 0; i < n; i++)
        printf("%8x ", dizi[i]);
    system("PAUSE");
    return 0;
}
```

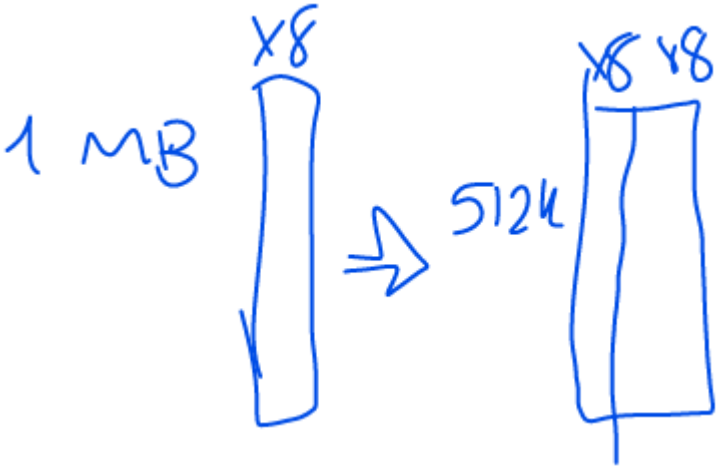
Neticeler

- C programları içerisinde hıza ihtiyaç duyulduğu zaman Assembly dili rahat bir şekilde kullanılabilir.
- Assembly dilinin de yüksek seviyeli dillerde olduğu gibi bir çok kütüphanesi olduğu ve bu kütüphanelerde bir çok ihtiyaç duyulan program, yordam seviyesinde kullanıcıya sunulduğu unutulmamalıdır. C içerisinde kullanılan printf ve scanf gibi fonksiyonların da bir kütüphane tarafından sağlandığı, bu kütüphaneyi kullanmadan ekrana bir yazı yazmanın veya okumanın ne kadar zor olacağı düşünüldüğünde Assembly için hazırlanmış çeşitli kütüphanelerin ihtiyaç duyulduğunda araştırılarak kullanılması belki hayat kurtarabilir.

PART 2: MİKROİŞLEMCİLER



temelde bizim kullanacağımız memory birimleri →
atomik olarak 8 bit hafıza birimleri



Bazı Genel Kavramlar (Tekrar Amaçlı)

Von Neumann mimarisi → program ve veri belleklerine erişim için ortak veri ve adres yollarının kullanıldığı tasarım kalıbı
Harvard Mimarisi → program ve veri belleklerine erişimde ayrı veri ve adres yolları bulunur.

Verinin Bellekteki Yerleşim Formatları:
Little Endian → Verinin düşük anlamlı byte'ından yüksek anlamlı byte'ına dpğru yerleşimi
Big Endian → Tam tersi

Bir işlemci ve çevre birimleri arasında veri aktarımını sağlayan *veri yolu(data bus)*, kontrol işaretlerini aktaran *kontrol yolu (control bus)* ve hangi bellek veya çevre biriminiin hangi içeriğe erişileceğini belirleyen *adres yolu (address bus)* mevcuttur.

ALU (Arithmetic Logic Unit) → İşlemcinin içerisinde yer alan tümdevre
Register (Yazmaç)
Cache (Önbellek)

CPU - central processing unit → Farklı fonksiyonel tümdevrelerin ana kart üzeerinde bir araya gelmesiyle oluşan işlemci yapısı
Mikroişlemci → İşlemcideki fonksiyonel blokları içeren tek bir tümdevre

Offset Memory Model: Örnek CS:IP (segment:offset)

µP → Mikroişlemci
µC → Mikrodenetleyici/Microcontroller

SoP → System on Chip - Yongada sistem

Time Multiplexing: Kısıtlı sayıda kaynağın farklı zamanda farklı görev alması
Coprocessor: Aritmetik İşlemler için Yardımcı İşlemci (günümüzde birleşti)

prefetch queue → önyükleme kuyruğu

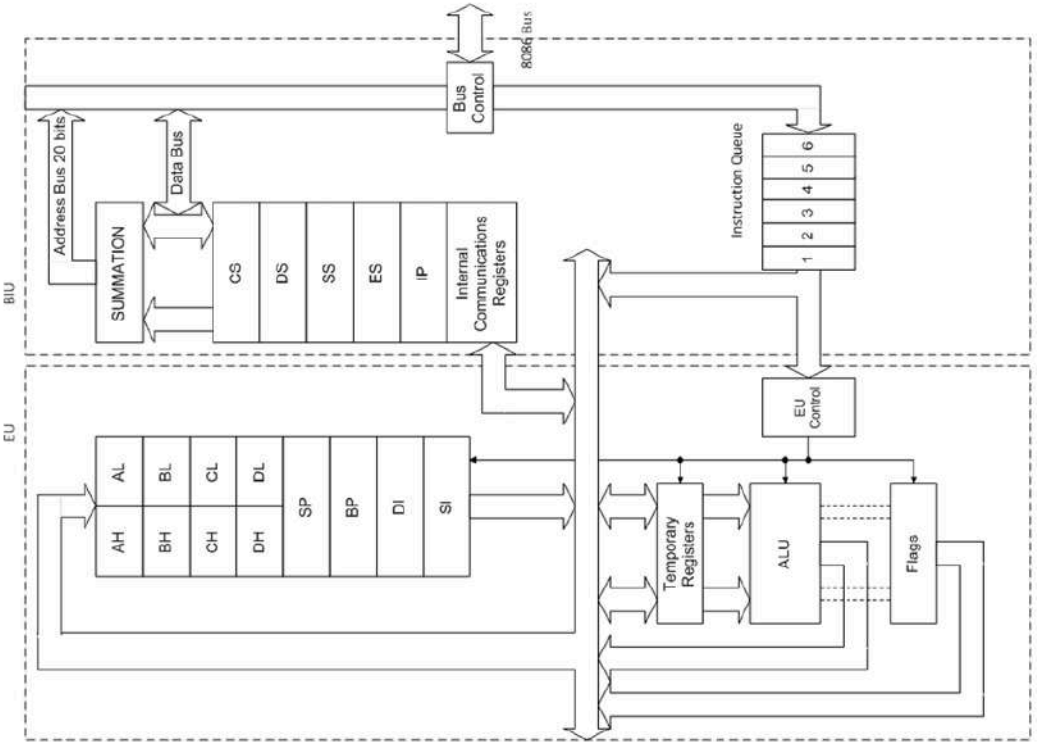
8086 Mikroişlemci Genel Özellikleri

- 1978 yılında Intel firması tarafından piyasaya sürülmüş, 16 bit genişliğinde veri yolu, 20 bit genişliğinde adres yolu ve 16 bit genişliğinde yazmaçlara sahip 16 bit'lik bir mikroişlemcidir.
- *Von Neumann* mimarisine sahiptir
- *CISC* organizasyonunda tasarlanmıştır ve veriyi *küçük sonlu (little endian* formatında saklar

8086 İç Yapısı

İki alt bileşenden oluşur:
bus interface unit- BIU - bellek ve I/O birimlerinden veri okuma ve yazma görevini yerine getirir. Program komutlarını bellekten getirir.
execution unit - EU -- BUI tarafından getirilen komutların çözümlenmesi ve yürütülmesinden sorumludur.

8086 İç Yapısı

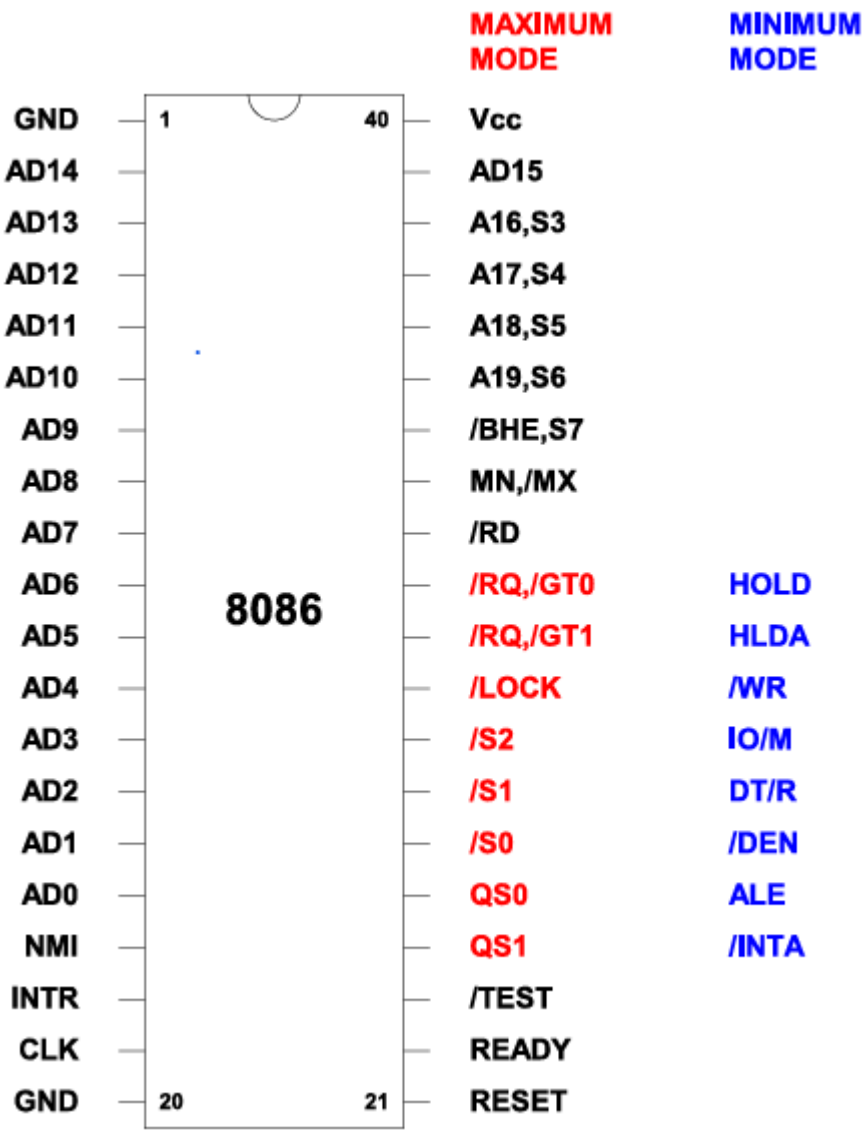


8086 Uç Tanımları

8086, 40 uçlu DIP paket tümleşik olarak üretilmiştir.
Fonksiyonel olarak 20 adet adres ucuna, 16 adet veri ucuna, çeşitli kontrol uçlarına ve besleme uçlarına ihtiyaç duyulmaktadır.
Fiziki 40 uç ile gerekli fonksiyonelliği sağlayabilmek için bazı uçlar *zaman çoğullamalı (time multiplexed* olarak kullanılır. (Örnek: AD (Address/Data) uçları)

NOTASYON

- Uçlar isimlendirilirken kullanılan "X/$\overline{X}$" notasyonu lojik değeriene göre gösterilen fonksiyonu ifade eder:
  0 : $\overline{X}$
  1 : X



(O-3, Tri-State-Buffer tipi mantık kapısını ifade eder. 1 ve 0 değerleri dışında bir de Z değeri alabilir (yüksek empedans)

Uç İsmi (Pin)	G/Ç Durumu	Açıklama ve Görevi
AD15 - AD0	O-3	**Adres/Veri Yolu:** Çoklanmış adres/veri yolunu oluşturur. *ALE*=1 olduğunda adres (bellek adresi/ I/O port numarası), diğer durumlarda veri taşır. AD7-AD0 düşük anlamlı kısmı, AD15-AD8 yüksek anlamlı kısmıdır.
A19/S6 - A16/S3	O-3	**Adres/Durum Yolu:** Adres (A16-A19) ve durum sinyallerini (S3-S6) sağlayabilir. *ALE* ucu 1 iken A19-A16 bitlerini içerir, *ALE* 0 iken (S3-S6) durum bitlerini içerir.S6 biti her zaman 0 değerini içerir. S5 biti IF bayrağının durumunu gösterir. S3/S4 hangi segment kesimine erişildiğini gösterir.
$\overline{RD}$	O-3	**Oku (Read):** Mantık 0 olduğunda, veri yolu bellekten veya I/O cihazlarından veri almaya hazırdır.
READY	I	**Hazır:** READY=0 ise işlemci bekler.
$\overline{TEST}$	I	**Test:** `WAIT` komutu tarafından test edilen giriştir. 0 okunursa NOP komutu yürütülür. 1 okunursa uç her çevrimde tekrar okunmaya devam eder.
NMI	I	**Maskelenemez Kesme (NMI):** INTR'ye benzer, ancak IF bayrağına *bakılmaksızın* INT 02H kesmesini tetikler.
RESET	I	**Sıfırla:** En az 4 saat darbesi boyunca yüksek (high) tutulursa işlemciyi sıfırlar. Programı `FFFF0H` adresinden başlatır ve kesmeleri devre dışı bırakır.
MN/$\overline{MX}$	I	**Mod Seçimi:** Minimum veya Maksimum kip mod seçimi
$\overline{BHE}$ **/ S7**	O-3	**Bus High Enable/Durum:** Yüksek veri baytını (D15-D8) etkinleştirir (Min. Mod). Maks. Modda S7 durum sinyali verir. S7 ucu her zaman lojik 1 değerine sahiptir.
M/$\overline{IO}$	O-3	**Bellek/Giriş-Çıkış Seçimi:** Adres yolunun bir bellek mi yoksa I/O port adresi mi içerdiğini gösterir.
$\overline{WR}$	O-3	**Yaz (Write):** 0 değeri ile 8086'nın bellek veya I/O cihazına veri çıkışı yapması sağlanır.
$\overline{INTA}$	O-3	**Kesme Onayı:** INTR girişine verilen yanıttır. Kesme vektör numarasını veri yoluna aktarmak için kullanılır.
$\overline{INTR}$	I	**Kesme İstek (interrupt request)**
$\overline{M}$**/IO**	O-3	**Bellek/Giriş-çıkış (memory/input-output)**
ALE	O	**(Address Latch Enable):** Adres/veri yolunun adres bilgisi içerdiğini gösterir.
DT/$\overline{R}$	O-3	**Veri İlet/Al (Data Transmit/Receive):** Veri yolunun veri gönderdiğini (transmit) veya aldığını (receive) gösterir. Veri aktarımında kullanılan çift yönlü mandalların aktarım yönünü belirlemek için kullanılır.
$\overline{DEN}$	O-3	**Veri Yolu Etkin (Data Bus Enable):** Harici veri yolunda çift yollu mandalları aktif eder.
HOLD	I	**Tut (Hold):** DMA (Doğrudan Bellek Erişimi) isteği yapar. Aktif olduğunda CPU durur ve yolları yüksek empedansa geçirir.
HLDA	O	**Tutma Onayı (Hold Acknowledge):** 8086'nın hold durumuna girdiğini belirtir.
CLK	-	Saat Darbesi Giriş Ucu.

Uç İsmi (Pin)	G/Ç Durumu	Açıklama ve Görevi
GND	-	Toprak
Vcc	-	Güç Kaynağı

Önemli kontrol uçlarından biri *MN/*$\overline{MX}$ ucudur. 8086 açısından giriş yönlü tanımlıdır, oluşacak lojik 0 seviyesi işlemcinin maksimum kipte , 1 seviyesi de minimum kipte çalışmasını sağlar. (Ders kapsamında minimum kip kullanılıcak, bu kipte G/Ç (I/O) kontrolü için gerekli tüm sinyaller 8086 tarafından üretilir.) Maksimum kip 8086'nın yardımcı işlemcilerle birlikte çalışmasını gerektirir (kontrol işaretlerinin üretimi için yardımcı tümdevre olan yol kontrolcüsü 8288 ile birlikte kullanım) (bu kısmı ders kitabından aldım)

maalesef mikroişlemci kısmı pek obsidian notuna yazılabilir gibi değil. O yüzden assembly notu burada bitiyor. Umarım faydası dokunmuştur. :)

Derste 8086 dışında işlenen çevre birimleri: **8255, 8251, ADC & DAC, 8259, RAM & ROM.** Atlanan konular: **8255 mod1, 8254, DMA**

Furkan Çakmak hocamızın youtube videolarından, hocaların 8086 assembly ders kitabından ve ders slaytlarından derlenmiştir.

Enes Utku Selbes