BİL 362 Mikroişlemciler: Veri Aktarım Komutları

Ahmet Burak Can
abc@hacettepe.edu.tr

İşlenen Tipleri

- Üç temel işlenen tipi vardır:
  - Anlık veri – sabit bir sayı (8, 16, or 32 bit)
  - Yazmaç – yazmaç adı
  - Bellek – bellekteki bir konuma referans

Doğrudan Bellek İşlenenleri

- Doğrudan bellek işleneni, bellekteki bir konuma referans eden isimdir.
  - Referans isim (etiket), derleyici tarafından otomatik olarak ters-referans edilir.

```
.data
var1 BYTE 10h

.code
mov al,var1                    ; AL = 10h
mov al,[var1]                  ; AL = 10h
```


MOV Komutu

- Veriyi kaynaktan hedefe taşır.
- Sözdizimi: MOV *hedef, kaynak*
  - Birden çok bellek işleneni kullanılamaz.
  - CS, EIP ve IP hedef olamaz.
  - Bölüt adres atamalarında anlık veri kullanılamaz.

```
.data
count BYTE 100
wVal  WORD 2

.code
    mov bl, count
    mov ax, wVal
    mov count, al

    mov al, wVal      ; hata: boyut uyumsuz
    mov ax, count     ; hata: boyut uyumsuz
    mov eax, count    ; hata: boyut uyumsuz
```

Yanlış MOV Kullanımları

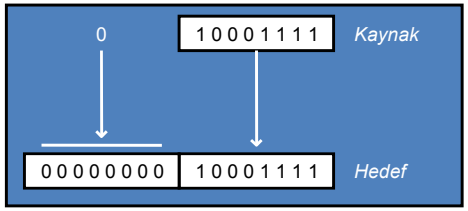
Aşağıdaki MOV deyimlerinin neden yanlış olduğunu açıklayın.

```
.data
bVal  BYTE   100
bVal2 BYTE   ?
wVal  WORD   2
dVal  DWORD  5

.code
    mov ds,45                ;DS yazmacina anlik veri atanamaz.
    mov esi,wVal             ;Büyüklük uyumsuz.
    mov eip,dVal             ;EIP hedef olamaz
    mov 25,bVal              ;Anlik veri hedef olamaz.
    mov bVal2,bVal           ;Bellekten bellege tasima yapilamaz.
```


Sıfır ile Genişletme (“Zero Extension”)

Küçük bir değeri daha büyük bir hedefe kopyalamak istediğinizde, MOVZX komutu hedefin üst yarısını sıfırla doldurur (genişletir).

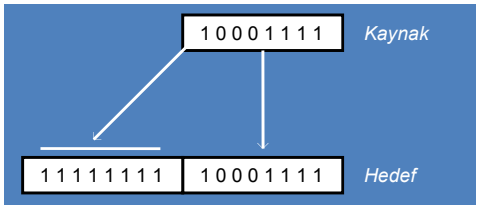



```
mov bl,10001111b
movzx ax,bl                  ;sifir ile genisletme
```

Hedef bir yazmaç olmalıdır !

İşaret ile Genişletme (“Sign Extension”)

MOVSX komutu, hedefin üst yarısını, kaynağın işaret biti ile doldurur (genişletir).




```
mov bl,10001111b
movsx ax,bl                  ;isaret ile genisletme
```

Hedef bir yazmaç olmalıdır !

XCHG Komutu

XCHG komutu, iki işlenenin değerlerini birbirleriyle değiştirir.

```
.data
var1 WORD 1000h
var2 WORD 2000h
.code
xchg ax,bx        ;16-bit yazmaçlarin degerlerini degistir
xchg ah,al        ;8-bit yazmaçlarin degerlerini degistir
xchg var1,bx      ;bellek ve yazmaç içeriklerini değiştir
xchg eax,ebx      ;32-bit yazmaçlarin degerlerini degistir

xchg var1,var2    ;hata:iki bellek isleneni degistirilemez
```

İşlenenlerden en az biri yazmaç olmalıdır.

Anlık veri işlenen olamaz.

Doğrudan-Ofset ("Direct-Ofset") İşlenenler - 1

Ofset değeri veri etiketine eklenerek etkin adres ("effective address" – EA) elde edilir. Adres, bellek konumundaki veriye ulaşmak için ters-referans edilir.

```
.data
arrayB BYTE 10h, 20h, 30h, 40h
.code
mov al, arrayB+1                ; AL = 20h
mov al, [arrayB+1]              ; alternatif gösterim
```


Doğrudan-Ofset ("Direct-Ofset") İşlenenler - 2

```
.data
arrayW  WORD 1000h,2000h,3000h
arrayD  DWORD 1,2,3,4
.code
mov ax,[arrayW+2]               ; AX = 2000h
mov ax,[arrayW+4]               ; AX = 3000h
mov eax,[arrayD+4]              ; EAX = 00000002h
```

Soru: Aşağıdaki ifadeler derlenebilir mi?

```
mov ax,[arrayW-2]               ; ??
mov eax,[arrayD+16]             ; ??
```

Soru: Çalıştırılırlarsa ne olur?

Alıştırma - 1

Aşağıdaki dizideki çift-sözcük değerlerini "3,1,2" şeklinde düzenleyen Assembly kodunu yazınız.

```
.data
arrayD DWORD 1,2,3
```

- Adım-1: İlk değeri EAX yazmacına kopyala ve ikinci pozisyondaki değerle yer değiştir.

```
mov eax,arrayD
xchg eax,[arrayD+4]
```

- Adım-2: EAX yazmacını üçüncü pozisyondaki değerle yer değiştir ve EAX yazmaç değerini ilk pozisyona kopyala.

```
xchg eax,[arrayD+8]
mov  arrayD,eax
```


Alıştırma - 2

- Aşağıdaki 3 baytı toplayan bir program yazmak istiyoruz.

```
.data
myBytes BYTE 80h,66h,0A5h
```

- Çözüm 1:

```
mov al,myBytes
add al,[myBytes+1]
add al,[myBytes+2]
```

- Çözüm 2: (derlemeden geçmez)

```
mov ax,myBytes        ;hata:boyut uyumsuz
add ax,[myBytes+1]    ;hata:boyut uyumsuz
add ax,[myBytes+2]    ;hata:boyut uyumsuz
```

Alıştırma - 3

```
.data
myBytes BYTE 80h,66h,0A5h
```

- Aşağıdaki kodu nasıl değerlendirirsiniz? Bir eksik var mı?

```
movzx ax,myBytes
mov   bl,[myBytes+1]
add   ax,bx
mov   bl,[myBytes+2]
add   ax,bx                ; AX = toplam
```

MOVZX komutundan önce, BX yazmacına sıfır atanmalıydı.

Örnek: Hareket.asm

```
.data
val1  WORD 1000h
val2  WORD 2000h
arrayB BYTE  10h,20h,30h,40h,50h
arrayW WORD  100h,200h,300h

.code
main PROC
;  Veri bölüt adresini ilklendir
    mov ax, @data
    mov ds, ax

;  Bellekten belleğe değişim:
    mov  ax, val1    ; AX = 1000h
    xchg ax, val2    ; AX = 2000h, val2 = 1000h
    mov  val1, ax    ; val1 = 2000h

;  Doğrudan-ofset adresleme (bayt dizisi):
    mov al, arrayB           ; AL = 10h
    mov al, [arrayB+1]       ; AL = 20h
    mov al, [arrayB+2]       ; AL = 30h

;  Doğrudan-ofset adresleme (sözcük dizisi):
    mov ax, arrayW           ; AX = 100h
    mov ax, [arrayW+2]       ; AX = 200h

; Komutların sonu
    .exit

main ENDP
END main
```


LAFH ve SAHF Komutları

LAHF ("Load Status Falgs into AH") komutu, EFLAGS yazmacının düşük baytını AH yazmacına kopyalar.

SAHF ("Store AH into Status Flags") komutu, AH yazmacının değerini EFLAGS yazmacının düşük baytına kopyalar.

("Overflow", "Sign", "Zero", "Auxiliary Carry", "Parity", "Carry" bayrakları)

```
.data
saveflags byte ?
.code
lahf              ; bayraklari AH yazmacina kopyala
mov saveflags, ah ; bir degiskende sakla

mov ah, saveflags ; saklanan bayraklari AH yazmacina yukle
sahf              ; bayrak yazmacina kopyala
```


Yığıt Komutları

- PUSH: Parametre olarak verilen 16 bitlik işleneni yığıta saklar.
  - Örnek: push ax
    push SI
- POP: Yığıtın başından aldığı 16 bitlik veriyi, işlenene aktarır.
  - Örnek: pop bx
    pop DI
- PUSHF: Bayrak yazmacını yığıta saklar. İşleneni yoktur.
  - Örnek: pushf
- POPF: Yığıtın başından aldığı 16 bitlik veriyi bayrak yazmacına aktarır.
  - Örnek: popf

LEA komutu

- LEA komutu, iki işlenen alır. Birinci işlenen bir yazmaçken, ikinci işlenen bir bellek adresidir. Birinci işlenenle belirtilen yazmaca, ikinci işlenenle belirtilen bölüt içi göreli (offset) adresi aktarır.
- Örnek:

```
veri  db 34H
...
lea ax, veri            ;ax içerisine verinin
                        ;adresini saklar
mov bx, 0200H
mov di, 0003H
lea bx, [bx+di]         ; bx=203H
```


IN Komutu

- IN komutu, bir giriş/çıkış kapısındaki (I/O port) veriyi, AL veya AX yazmacına aktarır.
- İki işlenen alır. Birinci işlenen AL veya AX yazmacı iken, ikinci işlenen kapı numarasıdır.
  - 255'den büyük kapı numaraları için ikinci işlenen DX yazmacıdır.

Örnek :

```
IN AX, 137H
MOV DX,  2300H
IN AL, DX
```


OUT Komutu

- OUT komutu, AL veya AX yazmacıdaki değeri, biri giriş/cıkış kapısına aktarır.
- İşlenenler IN komutundaki ile aynıdır, fakat tersi sıradadır.
  - Örnek :

```
MOV AX, 1234H
OUT 138H, AL

MOV DX,  2300H
OUT DX, AX
```