BİL 362 Mikroişlemciler: Akış Denetim Komutları

Ahmet Burak Can
abc@hacettepe.edu.tr

JMP Komutu

- JMP komutu, bir komut etiketine koşulsuz atlamayı sağlar.
- Sözdizimi: JMP *hedef*
  - IP ← *hedef*
- Örnek:

```
top:
    .
    .
    jmp top
```


LOOP Komutu

- LOOP komutu sayılı bir döngü yaratır.
- Sözdizim: LOOP *hedef*
  - CX ← CX – 1 (kendiliğinden azalır)
  - CX sıfır değilse hedefe atla
- *Uygulama:*
  - *Derleyici, bir sonraki komutun ofseti ile hedef etiketin ofseti arasındaki uzaklığı bayt olarak hesaplar (göreceli adres – "relative address").*
  - *Göreceli adresin ofseti IP değerine eklenir.*

LOOP Komutu: Örnek

Aşağıdaki döngü 5 + 4 + 3 +2 + 1 tamsayılarının toplamını hesaplar.

```
ofset      makine kodu              kaynak kod
00000000   66 B8 0000              mov  ax,0
00000004   B9 00000005             mov  ecx,5

00000009   66 03 C1           L1: add  ax,cx
0000000C   E2 FB                  loop L1
0000000E
```

Döngüden sonraki konum = 0000000E (sonraki komutun ofseti)

Mevcut konuma –5 (FBh) eklenerek 00000009 adresine atlanır:

00000009 ← 0000000E + FB

Alıştırma

AX'in son değeri nedir?

10

```
    mov ax,6
    mov cx,4
L1:
    inc ax
    loop L1
```

Döngü kaç kez çalışır?

4,294,967,296

```
    mov cx,0
X2:
    inc ax
    loop X2
```


İçiçe Döngü

Döngü içinde döngü oluşturmak istediğinizde, dış döngü sayacını saklayıp geri okumalısınız.

Aşağıdaki örnekte dış döngü 100 kez, iç döngü 20 kez işletilir.

```
.data
count DWORD ?
.code
    mov cx,100              ; dis dongu sayisini belirle
L1:
    mov count,cx            ; dis dongu sayisini sakla
    mov cx,20               ; ic dongu sayisini belirle
L2: .
    .
    loop L2
    mov cx,count            ; dis dongu sayisini geri al
    loop L1
```


Örnek: Tamsayı Dizisini Toplamak

16-bit tamsayı dizisinin elemanlarını, dolaylı işlenen kullanarak toplayan Assembly programını yazın.

```
.data
intarray DW 100h,200h,300h,400h

.code
    mov di, OFFSET intarray     ; intarray adresi
    mov cx, 4                   ; dongu sayaci
    mov ax, 0                   ; ax yazmacini sifirla
L1:
    add ax, [di]                ; tamsayiyi ekle
    add di, 2                   ; sonraki tamsayiyi goster
    loop L1                     ; CX = 0 olana kadar tekrar et
```


Örnek: Dizgiyi Kopyalamak

Aşağıdaki dizgiyi kaynaktan hedefe, dizinli işlenen kullanarak kopyalayan Assembly programını yazın.

```
.data
source  BYTE  "This is the source string",0
target  BYTE  SIZEOF source DUP(0)

.code
    mov  si,0                   ; dizin yazmaci
    mov  cx, SIZEOF source      ; dongu sayaci
L1:
    mov  al, source[si]         ; kaynaktan karakteri al
    mov  target[si], al         ; karakteri hedefe yaz
    inc  si                     ; bir sonraki karaktere gec
    loop L1                     ; tum dizgi icin tekrarla
```

Koşullu Atlama İfadeleri

J*cond* Komutu

- Bir koşullu atlama komutu, belirli bir yazmaç veya bayrak koşulu sağlandığında etikete atlamayı sağlar.
- Örnekler:
  - JB, JC: Elde (“Carry”) bayrağı 1 ise etikete atlar.
  - JE, JZ: Sıfır (“Zero”) bayrağı 1 ise etikete atlar.
  - JS: İşaret (“Sign”) bayrağı 1 ise etikete atlar.
  - JNE, JNZ: Sıfır (“Zero”) bayrağı 0 ise etikete atlar.
  - JECXZ: ECX yazmacı 0 ise etikete atlar.

Bayraklara Özel Koşullu Atlamalar

Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

Eşitliğe Dayalı Koşullu Atlamalar

Mnemonic	Description
JE	Jump if equal (*leftOp* = *rightOp*)
JNE	Jump if not equal (*leftOp* ≠ *rightOp*)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0

İşaretsiz Karşılaştırmalara Dayalı Koşullu Atlamalar

Mnemonic	Description
JA	Jump if above (if *leftOp* > *rightOp*)
JNBE	Jump if not below or equal (same as JA)
JAE	Jump if above or equal (if *leftOp* >= *rightOp*)
JNB	Jump if not below (same as JAE)
JB	Jump if below (if *leftOp* < *rightOp*)
JNAE	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if *leftOp* <= *rightOp*)
JNA	Jump if not above (same as JBE)

İşaretli Karşılaştırmalara Dayalı Koşullu Atlamalar

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* >= *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* <= *rightOp*)
JNG	Jump if not greater (same as JLE)

Uygulamalar - 1

- Görev: İşaretsiz EAX, EBX'den büyükse etikete atla.

  → JA ("jump if above")

```
cmp eax,ebx
ja  Larger
```

- Görev: İşaretli EAX, EBX'den büyükse etikete atla.

  → JG ("jump if greater")

```
cmp eax,ebx
jg  Greater
```


Uygulamalar - 2

- İşaretsiz EAX, Val1'den küçük veya Val1'e eşitse etikete atla.

  → JBE ("jump if below or equal")

```
cmp eax,Val1
jbe L1              ; küçük veya esitse
```

- İşaretli EAX, Val1'den küçük veya Val1'e eşitse etikete atla.

  → JLE ("jump if less than or equal")

```
cmp eax,Val1
jle L1
```

Uygulamalar - 3

- İşaretsiz AX'i BX ile karşılaştır ve büyük olanı "Large" veri etiketi ile gösterilen alana kopyala. → JNA ("jump if not above")

```
    mov Large,bx
    cmp ax,bx
    jna Next
    mov Large,ax
Next:
```

- İşaretli AX'i BX ile karşılaştır ve küçük olanı "Small" veri etiketi ile gösterilen alana kopyala. → JNL ("jump if not less")

```
    mov Small,ax
    cmp bx,ax
    jnl Next
    mov Small,bx
Next:
```


Uygulamalar - 4

- SI tarafından gösterilen bellek içeriği sıfıra eşitse L1 etiketine atla.
  → JE ("jump if equal")

```
    cmp WORD PTR [si],0
    je  L1
```

- DI tarafından gösterilen çift-sözcük boyutundaki bellek içeriği çift sayı ise L2 etiketine atla.
  → JZ ("jump if zero")

```
test DWORD PTR [di],1
jz   L2
```


Örnekler

```
mov edx,-1
cmp edx,0
jnl L5      ; atlama gerceklesmez
jnle L5     ; atlama gerceklesmez
jl  L1      ; atlama gerceklesir


mov bx,+34
cmp bx,-35
jng L5      ; atlama gerceklesmez
jnge L5     ; atlama gerceklesmez
jge L1      ; atlama gerceklesir
```

```
mov ecx,0
cmp ecx,0
jg L5       ; atlama gerceklesmez
jnl L1      ; atlama gerceklesir


mov ecx,0
cmp ecx,0
jl L5       ; atlama gerceklesmez
jng L1      ; atlama gerceklesir
```


Alıştırma

- Aşağıdaki üç işaretsiz tamsayıyı karşılaştırarak en küçüğünü AX yazmacına kopyalayan programı yazın.

```
.data
V1 WORD ?
V2 WORD ?
V3 WORD ?
```

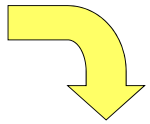
```
.code
main PROC
    mov ax,V1
    cmp ax,V2
    jbe L1
    mov ax,V2
L1: cmp ax,V3
    jbe L2
    mov ax,V3
L2:
main ENDP
END main
```

Örnek: Bir Diziyi Taramak

```
TITLE Bir Diziyi Taramak  (DiziTara.asm)
; Bir diziyi icindeki ilk sifirdan farkli eleman icin tarar.
.data
intArray SWORD 0,0,0,0,1,20,35,-12,66,4,0
.code
main PROC
   mov bx, OFFSET intArray
   mov cx, LENGTHOF intArray
L1:
   cmp WORD PTR [bx],0            ; elemani sifirla karsilastir
   jnz found                      ; sifir degilse etikete etla
   add bx, 2                      ; bir sonraki elemana git
   loop L1                        ; donguye devam et
   jmp notFound                   ; dongu bittiyse eleman bulunamadi
found:
   mov ax,[bx]                    ; bulunan elemani ax yazmacina kopyala
   jmp quit                       ; bitir
notFound:
   mov eax,0                      ; eleman bulunamadi ise ax = 0
quit:
   .exit                          ; program sonu
main ENDP
END main
```


Blok-Yapılı IF Deyimleri

```
if( op1 == op2 )
  X = 1;
else
  X = 2;
```



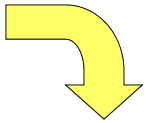
```
    mov ax,op1
    cmp ax,op2
    jne L1
    mov X,1
    jmp L2
L1: mov X,2
L2:
```


Alıştırma - 1

Aşağıdaki kod bloğunu Assembly'de kodlayın.

(Tüm değerlerin işaretsiz olduğunu varsayın.)

```
if( ebx <= ecx )
{
  eax = 5;
  edx = 6;
}
```



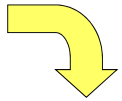
```
    cmp ebx,ecx
    ja  next
    mov eax,5
    mov edx,6
next:
```


Alıştırma - 2

Aşağıdaki kod bloğunu Assembly'de kodlayın.

(Tüm değerlerin 16-bit işaretli tamsayı olduğunu varsayın.)

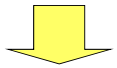
```
if( var1 <= var2 )
  var3 = 10;
else
{
  var3 = 6;
  var4 = 7;
}
```



```
    mov ax,var1
    cmp ax,var2
    jle L1
    mov var3,6
    mov var4,7
    jmp L2
L1: mov var3,10
L2:
```

AND ile Birleştirilen İfadeler - 1

```
if (al > bl) AND (bl > cl)
  X = 1;
```



```
    cmp al,bl               ; ilk ifade...
    ja  L1
    jmp next
L1:
    cmp bl,cl               ; ikinci ifade...
    ja  L2
    jmp next
L2:                         ; her ikisi dogru ise
    mov X,1                 ; X = 1
next:
```


AND ile Birleştirilen İfadeler - 2

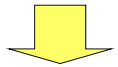
```
if (al > bl) AND (bl > cl)
  X = 1;
```

Aşağıdaki çözüm %29 oranında az kod kullanır.

```
    cmp al,bl               ; ilk ifade...
    jbe next                ; yanlissa çik
    cmp bl,cl               ; ikinci ifade...
    jbe next                ; yanlissa çik
    mov X,1                 ; her ikisi dogru ise
next:
```


OR ile Birleştirilen İfadeler

```
if (al > bl) OR (bl > cl)
  X = 1;
```



```
    cmp al,bl               ; AL > BL?
    ja  L1                  ; evet
    cmp bl,cl               ; hayir: BL > CL?
    jbe next                ; hayir: sonraki komutu atla
L1: mov X,1                 ; X = 1
next:
```


WHILE Döngüsü

WHILE döngüsü, IF deyimini izleyen döngü gövdesi ve döngünün başına koşulsuz atlama ile kodlanır.

Örnek:

```
while( ax < bx)
    ax = ax + 1;
```

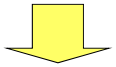
Bir çözüm:

```
top: cmp ax,bx              ; döngü kosulunu kontrol et
     jae next               ; yanlis ise döngüden çık
     inc ax                 ; döngünün gövdesi
     jmp top                ; döngüyü tekrar et
next:
```

Alıştırma

Aşağıdaki döngüyü 16-bit tamsayıları kullanarak kodlayın.

```
while( bx <= val1)
{
    bx = bx + 5;
    val1 = val1 - 1
}
```



```
top: cmp bx,val1              ; döngü kosuulunu kontrol et
     ja  next                 ; yanlis ise döngüden çık
     add bx,5                 ; döngü gövdesi
     dec val1
     jmp top                  ; döngüyü tekrarla
next:
```