Bölüm 7: Akış Kontrolü

Mikroişlemciler

IP ve sıralı yürütme · Koşulsuz atlama (JMP) · Tek bayrağı test eden atlamalar · CMP ile karşılaştırma · İşaretli / işaretsiz atlamalar · LOOP ailesi · İç içe döngüler · Örnek programlar

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Atlama mekanizması

JMP ve etiketlerle programın akışını yönlendirmek; IP'nin nasıl değiştiğini ve bağıl konumun (offset) nasıl hesaplandığını açıklamak.

Koşullu atlamalar

Her koşullu atlamanın hangi bayrağa (ZF, CF, SF, OF, PF) baktığını söylemek ve zıt komutunu bulmak.

Doğru karşılaştırma

CMP sonrasında işaretsiz sayılar için JA/JB, işaretli sayılar için JG/JL ailesini doğru seçmek.

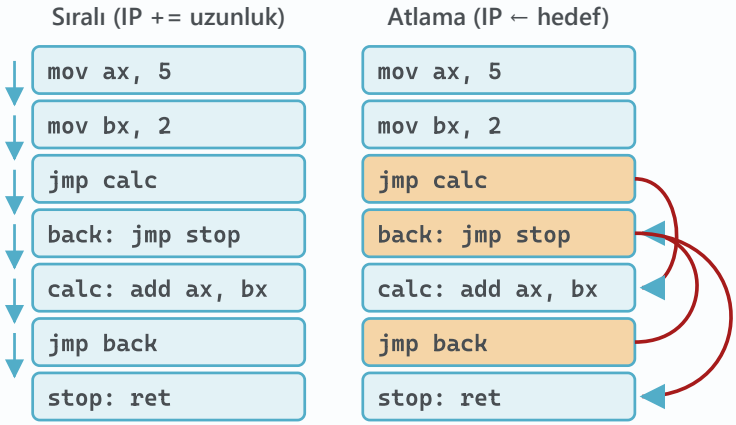
Döngüler

LOOP, LOOPE/LOOPZ, LOOPNE/LOOPNZ ve JCXZ ile sayaçlı döngüler yazmak; örnek programları adım adım izlemek.

GİRİŞ

Program Akış Kontrolü

- İşlemci normalde komutları bellekteki sırasıyla yürütür: her komuttan sonra **IP** (Instruction Pointer) komutun uzunluğu kadar artar.
- **Akış kontrol komutları** IP'ye yeni bir değer yükleyerek yürütmeyi başka bir noktaya taşır.
- Böylece karar yapıları (if/else) ve döngüler (for/while) makine düzeyinde gerçekleştirilir.



Koşulsuz atlama

`JMP etiket`: her zaman atlar.

Koşullu atlamalar

Tek bayrağı test edenler, işaretli ve işaretsiz sayılar için olanlar.

Döngüler

`LOOP`, `LOOPE/LOOPZ`, `LOOPNE/LOOPNZ`, `JCXZ`.

JMP: Koşulsuz Atlama

- `JMP etiket` kontrolü programın başka bir noktasına **her koşulda** aktarır; bayrakları değiştirmez.
- Hem **ileri** hem **geri** atlayabilir.
- Yakın (near) atlama, mevcut kod kesimi içinde (64 KB = 65.536 bayt, bağıl konum 0000h–FFFFh) her yere ulaşır; yalnızca IP değişir.
- Uzak (far) atlama CS ve IP'yi birlikte değiştirerek başka bir kesime geçer.
- Assembler, hedef yakınsa otomatik olarak 2 baytlık **kısa** (short) biçimi seçebilir.

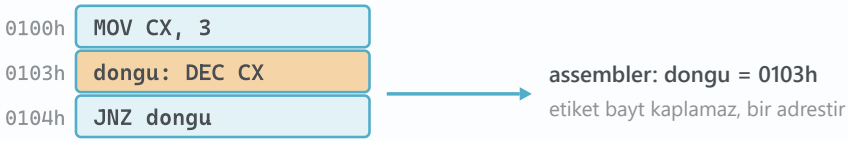
Biçim	Kod	Bayt	Menzil
`JMP SHORT`	EB rel8	2	−128… +127
`JMP NEAR`	E9 rel16	3	kesim içi
`JMP FAR`	EA ofs seg	5	CS:IP
`JMP reg/mem`	FF /4	2–4	dolaylı

rel8 / rel16: hedefin, atlama komutundan **sonraki** komutun adresine göre işaretli uzaklığı.

KOŞULSUZ ATLAMA

Etiketler

- Etiket tanımlamak için adı yazılır ve sonuna `:` eklenir: `dongu:`
- Harf, rakam ve `_` gibi karakterlerden oluşabilir, ancak **rakamla başlayamaz**.
- Ayrı bir satırda ya da bir komutun önünde tanımlanabilir: `back: jmp stop`
- Etiket bellekte yer kaplamaz; assembler onu bir **adrese** (bağıl konum) dönüştürür.
- Komut adları ve yazmaç adları (`mov` , `ax` ...) etiket olarak kullanılamaz; her etiket bir kez tanımlanır.



Geçerli

+ `dongu:`
+ `next_char:`
+ `k1:` `calc:`

Geçersiz

– `1basla:` rakamla başlıyor
– `mov:` komut adı
– `iki kelime:` boşluk içeriyor

KOŞULSUZ ATLAMA · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: `JMP` ile İleri ve Geri

```
1  org     100h
2  mov     ax, 5           ; ax = 5
3  mov     bx, 2           ; bx = 2
4  jmp     calc            ; 'calc' etiketine git
5  back:   jmp stop        ; 'stop' etiketine git
6  calc:
7  add     ax, bx          ; ax = ax + bx
8  jmp     back            ; 'back' etiketine dön
9  stop:
10 ret                     ; işletim sistemine dön
```

DEĞİŞKENLER

`IP = 0100h`

Program 0100h adresinden başlar (`org 100h` , .COM programı). IP = 0100h.

Adım 1 / 8

Kod İzleme: `JMP` ile İleri ve Geri

```
1  org   100h
▶  mov   ax, 5          ; ax = 5
3  mov   bx, 2          ; bx = 2
4  jmp   calc           ; 'calc' etiketine git
5  back: jmp stop       ; 'stop' etiketine git
6  calc:
7  add   ax, bx         ; ax = ax + bx
8  jmp   back           ; 'back' etiketine dön
9  stop:
10 ret                  ; işletim sistemine dön
```

DEĞİŞKENLER

`AX = 0005h` `IP = 0103h`

`mov ax, 5` (3 bayt) yürütüldü: AX = 0005h, IP = 0103h.

Adım 2 / 8

Kod İzleme: `JMP` ile İleri ve Geri

```
1  org   100h
2  mov   ax, 5          ; ax = 5
▶  mov   bx, 2          ; bx = 2
4  jmp   calc           ; 'calc' etiketine git
5  back: jmp stop       ; 'stop' etiketine git
6  calc:
7  add   ax, bx         ; ax = ax + bx
8  jmp   back           ; 'back' etiketine dön
9  stop:
10 ret                  ; işletim sistemine dön
```

DEĞİŞKENLER

`AX = 0005h` `BX = 0002h` `IP = 0106h`

`mov bx, 2` : BX = 0002h. Sıradaki komut 0106h adresindeki `jmp calc` .

Adım 3 / 8

Kod İzleme: `JMP` ile İleri ve Geri

```
1  org   100h
2  mov   ax, 5          ; ax = 5
3  mov   bx, 2          ; bx = 2
▶  jmp   calc           ; 'calc' etiketine git
5  back: jmp stop       ; 'stop' etiketine git
6  calc:
7  add   ax, bx         ; ax = ax + bx
8  jmp   back           ; 'back' etiketine dön
9  stop:
10 ret                  ; işletim sistemine dön
```

DEĞİŞKENLER

`AX = 0005h` `BX = 0002h` `IP = 010Ah`

`jmp calc` : IP ← calc = 010Ah. Arada kalan `back:` satırı atlanır.

Adım 4 / 8

Kod İzleme: `JMP` ile İleri ve Geri

```
1  org   100h
2  mov   ax, 5          ; ax = 5
3  mov   bx, 2          ; bx = 2
4  jmp   calc           ; 'calc' etiketine git
5  back: jmp stop       ; 'stop' etiketine git
6  calc:
▶  add   ax, bx         ; ax = ax + bx
8  jmp   back           ; 'back' etiketine dön
9  stop:
10 ret                  ; işletim sistemine dön
```

DEĞİŞKENLER

`AX = 0007h` `BX = 0002h` `IP = 010Ch`

`add ax, bx` : AX = 5 + 2 = 0007h.

Adım 5 / 8

Kod İzleme: `JMP` ile İleri ve Geri

```
1  org    100h
2  mov    ax, 5          ; ax = 5
3  mov    bx, 2          ; bx = 2
4  jmp    calc           ; 'calc' etiketine git
5  back:  jmp stop       ; 'stop' etiketine git
6  calc:
7  add    ax, bx         ; ax = ax + bx
▶  jmp    back           ; 'back' etiketine dön
9  stop:
10 ret                   ; işletim sistemine dön
```

DEĞİŞKENLER

AX = **0007h** | BX = **0002h** | IP = **0108h**

`jmp back` : **geriye** atlama, IP ← 0108h.

Adım 6 / 8

Kod İzleme: `JMP` ile İleri ve Geri

```
1  org   100h
2  mov   ax, 5          ; ax = 5
3  mov   bx, 2          ; bx = 2
4  jmp   calc           ; 'calc' etiketine git
▶  back: jmp stop       ; 'stop' etiketine git
6  calc:
7  add   ax, bx         ; ax = ax + bx
8  jmp   back           ; 'back' etiketine dön
9  stop:
10 ret                  ; işletim sistemine dön
```

DEĞİŞKENLER

`AX = 0007h` `BX = 0002h` `IP = 010Eh`

`back: jmp stop` : tekrar ileri atlanır, IP ← 010Eh.

Adım 7 / 8

Kod İzleme: `JMP` ile İleri ve Geri

```
1  org   100h
2  mov   ax, 5          ; ax = 5
3  mov   bx, 2          ; bx = 2
4  jmp   calc           ; 'calc' etiketine git
5  back: jmp stop       ; 'stop' etiketine git
6  calc:
7  add   ax, bx         ; ax = ax + bx
8  jmp   back           ; 'back' etiketine dön
9  stop:
▶  ret                  ; işletim sistemine dön
```

DEĞİŞKENLER

`AX = 0007h` `BX = 0002h` `IP = 0000h`

`ret` : IP ← 0000h, program sonlanır. Sonuç AX = 0007h. Akış: 1 → 2 → 3 → calc → back → stop.

Adım 8 / 8

Bağıl Konum Hesabı

adres	makine kodu	komut
0100h	B8 05 00	mov ax, 5
0103h	BB 02 00	mov bx, 2
0106h	EB 02	jmp calc
0108h	EB 04	back: jmp stop
010Ah	01 D8	calc: add ax, bx
010Ch	EB FA	jmp back
010Eh	C3	stop: ret

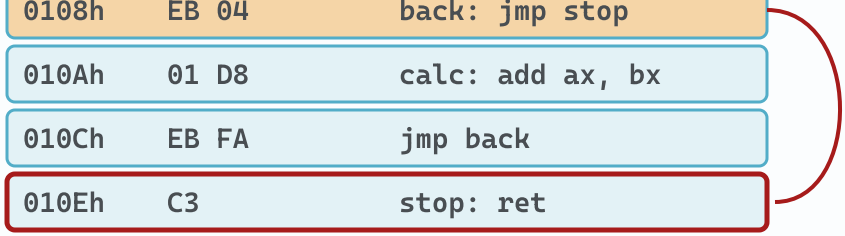


```
jmp calc @ 0106h (2 bayt)
sonraki IP = 0108h
hedef calc = 010Ah
rel8 = 010A − 0108 = +2
kod: EB 02
```

İleri atlama: uzaklık, atlama komutundan **sonraki** adrese göre hesaplanır → **+2**.

Bağıl Konum Hesabı

adres	makine kodu	komut
0100h	B8 05 00	mov ax, 5
0103h	BB 02 00	mov bx, 2
0106h	EB 02	jmp calc
0108h	EB 04	back: jmp stop
010Ah	01 D8	calc: add ax, bx
010Ch	EB FA	jmp back
010Eh	C3	stop: ret



```
jmp stop @ 0108h
sonraki IP = 010Ah
hedef stop = 010Eh
rel8 = 010E − 010A = +4
kod: EB 04
```

İşlemci yürütürken IP ← sonraki IP + rel8 = 010Ah + 4 = **010Eh**.

Bağıl Konum Hesabı

adres	makine kodu	komut
0100h	B8 05 00	mov ax, 5
0103h	BB 02 00	mov bx, 2
0106h	EB 02	jmp calc
0108h	EB 04	back: jmp stop
010Ah	01 D8	calc: add ax, bx
010Ch	EB FA	jmp back
010Eh	C3	stop: ret

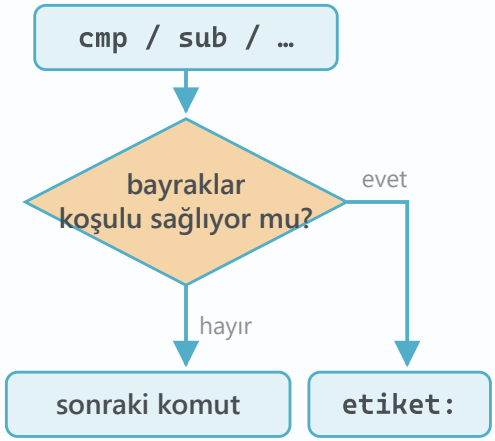


```
jmp back @ 010Ch
sonraki IP = 010Eh
hedef back = 0108h
rel8 = 0108 − 010E = −6
−6 = FAh (ikiye tümleyen)
kod: EB FA
```

Geri atlama: uzaklık negatiftir ve ikiye tümleyenle saklanır: −6 → **FAh**. 010Eh + FFFAh = 0108h (taşan bit atılır).

Koşullu Atlama Fikri

- `Jcc etiket` : koşul (cc) doğruysa IP ← etiket, değilse bir sonraki komut yürütülür.
- Koşul, önceki bir komutun (çoğunlukla `CMP` , `SUB` , `ADD` , `TEST` , `DEC`) ayarladığı **bayraklara** bakılarak değerlendirilir.
- Atlama komutlarının kendisi bayrakları **değiştirmez**.



ZF sonuç sıfır	**CF** elde / ödünç	**SF** sonuç negatif
OF işaretli taşma	**PF** 1 sayısı çift	**AF** atlamada kullanılmaz

KOŞULLU ATLAMALAR

Tek Bir Bayrağı Test Eden Atlamalar

Komut	Anlamı	Koşul	Zıt komut
`JZ, JE`	Jump if Zero (Equal): sıfırsa / eşitse	ZF = 1	JNZ, JNE
`JC, JB, JNAE`	Jump if Carry (Below, Not Above or Equal): elde varsa	CF = 1	JNC, JNB, JAE
`JS`	Jump if Sign: sonuç negatifse	SF = 1	JNS
`JO`	Jump if Overflow: işaretli taşma varsa	OF = 1	JNO
`JPE, JP`	Jump if Parity Even: eşlik çiftse	PF = 1	JPO, JNP
`JNZ, JNE`	Jump if Not Zero (Not Equal)	ZF = 0	JZ, JE
`JNC, JNB, JAE`	Jump if Not Carry (Not Below, Above or Equal)	CF = 0	JC, JB, JNAE
`JNS`	Jump if Not Sign: negatif değilse	SF = 0	JS
`JNO`	Jump if Not Overflow	OF = 0	JO
`JPO, JNP`	Jump if Parity Odd (No Parity)	PF = 0	JPE, JP

JC/JB adındaki "C" elde (carry) bayrağıdır; "taşma" (overflow) ise OF ile ilgilidir: ikisi farklı kavramlardır.

KOŞULLU ATLAMALAR

Kodlama ve Eş Anlamlılar

- 8086'da koşullu atlamalar sabit uzunluktadır: **2 bayt** = işlem kodu + 1 baytlık bağıl konum (rel8).
- rel8 her zaman **işaretli** sayıdır: sonraki komuta göre **−128 bayt geriye** veya **+127 bayt ileriye** atlanabilir.
- Aynı koşulun farklı adları (eş anlamlılar) **aynı makine koduna** derlenir: JE = JZ, JNE = JNZ, JC = JB = JNAE, JNC = JNB = JAE.
- 16 bitlik uzaklıklı koşullu atlamalar (0F 8x) ancak 80386 ile gelmiştir; 8086'da yoktur.

`74` JE / JZ	`75` JNE / JNZ	`72` JB / JC / JNAE	`73` JAE / JNB / JNC
`78` JS	`79` JNS	`70` JO	`71` JNO
`7A` JP / JPE	`7B` JNP / JPO	`77` JA / JNBE	`76` JBE / JNA
`7F` JG / JNLE	`7D` JGE / JNL	`7C` JL / JNGE	`7E` JLE / JNG
`E2` LOOP	`E1` LOOPE / LOOPZ	`E0` LOOPNE / LOOPNZ	`E3` JCXZ

Hepsi 2 bayttır: işlem kodu + rel8. Kısa JMP'nin kodu EBh'dir.

Kod İzleme: `JNC` = `JNB` = `JAE`

```
▶ org 100h
2 jnc a          ; 0100h: 73 07
3 jnb a          ; 0102h: 73 05
4 jae a          ; 0104h: 73 03
5 mov ax, 4      ; 0106h: B8 04 00
6 a: mov ax, 5   ; 0109h: B8 05 00
7 ret            ; 010Ch: C3
```

DEĞİŞKENLER

`CF = 0` `AX = 0000h` `IP = 0100h`

Üç komut da aynı koşulu (CF = 0) sınar ve aynı işlem kodunu (**73h**) üretir; yalnızca rel8 farklıdır, çünkü her biri farklı adreste.

Adım 1 / 4

Kod İzleme: `JNC` = `JNB` = `JAE`

```
1  org 100h
▶  jnc a          ; 0100h: 73 07
3  jnb a          ; 0102h: 73 05
4  jae a          ; 0104h: 73 03
5  mov ax, 4      ; 0106h: B8 04 00
6  a: mov ax, 5   ; 0109h: B8 05 00
7  ret            ; 010Ch: C3
```

DEĞİŞKENLER

`CF = 0` `AX = 0000h` `IP = 0109h`

CF = 0 olduğundan `jnc a` atlar: IP ← 0102h + 07h = **0109h**. Aradaki `jnb`, `jae` ve `mov ax, 4` yürütülmez.

Adım 2 / 4

Kod İzleme: `JNC` = `JNB` = `JAE`

```
1 org 100h
2 jnc a          ; 0100h: 73 07
3 jnb a          ; 0102h: 73 05
4 jae a          ; 0104h: 73 03
5 mov ax, 4      ; 0106h: B8 04 00
▶ a: mov ax, 5   ; 0109h: B8 05 00
7 ret            ; 010Ch: C3
```

DEĞİŞKENLER

`CF = 0` `AX = 0005h` `IP = 010Ch`

`a: mov ax, 5` → AX = 0005h.

Adım 3 / 4

Kod İzleme: `JNC` = `JNB` = `JAE`

```
1 org 100h
2 jnc a          ; 0100h: 73 07
3 jnb a          ; 0102h: 73 05
4 jae a          ; 0104h: 73 03
5 mov ax, 4      ; 0106h: B8 04 00
6 a: mov ax, 5   ; 0109h: B8 05 00
▶ ret            ; 010Ch: C3
```

DEĞİŞKENLER

`CF = 0` `AX = 0005h` `IP = 0000h`

`ret` : IP ← 0000h. CF = 1 olsaydı üç atlama da gerçekleşmez, AX önce 4 sonra 5 olurdu.

Adım 4 / 4

KOŞULLU ATLAMALAR

Atlama Kodu Hesaplayıcı

Atlama komutunun adresi ve hedef adres (onaltılık) girilince rel8 / rel16 değeri ve makine kodu hesaplanır.

Komut: JMP SHORT (EB)

adres: 010C

hedef: 0108

```
sonraki IP = 010Ch + 2 = 010Eh
 uzaklık = 0108h − 010Eh = −6
rel8 = FAh → makine kodu EB FA
```

JMP SHORT 010Ch adresinde, hedef 0108h.

Kural: `uzaklık = hedef − (adres + komut uzunluğu)` . Kısa biçimde sonuç −128…+127 aralığında olmalıdır; değilse assembler "*relative jump out of range*" hatası verir.

KARŞILAŞTIRMA

CMP Komutu

- Sayısal değerleri karşılaştırmak için `CMP hedef, kaynak` kullanılır.
- CMP, **SUB** (çıkarma) işlemini yapar: `hedef − kaynak`. Ancak sonucu **hiçbir yere yazmaz**; yalnızca bayrakları (CF, ZF, SF, OF, PF, AF) ayarlar.
- Ardından gelen koşullu atlama bu bayraklara bakarak karar verir.
- İşlenenler: yazmaç–yazmaç, yazmaç–bellek, bellek–yazmaç, yazmaç/bellek–sabit. İki bellek işleneni **olamaz**.

CMP 05h, 02h

`5 − 2 = 3 → 03h`

ZF = 0, CF = 0, SF = 0, OF = 0

CMP 07h, 07h

`7 − 7 = 0 → 00h`

ZF = 1, CF = 0, SF = 0, OF = 0

Örnek 1: 5 − 2 = 3 → sonuç sıfır değil, **ZF = 0**. Örnek 2: 7 − 7 = 0 → sonuç sıfır, **ZF = 1** ve `JE / JZ` atlar.

KARŞILAŞTIRMA

`CMP AL, BL` Bayrakları Nasıl Ayarlar?

AL	0000 0101	05h	işaretsiz 5 · işaretli 5
BL	0000 0010	02h	işaretsiz 2 · işaretli 2
AL−BL	0000 0011	03h	sonuç saklanmaz, yalnız bayraklar

CF=0 ZF=0 SF=0 OF=0 **PF=1** AF=0

işaretsiz: AL > BL

CF=0, ZF=0 → JA atlar

işaretli: AL > BL

SF=0, OF=0, ZF=0 → JG atlar

Pozitif fark: hiçbir bayrak kalkmaz (PF = 1 çünkü 03h = 0000 0011 iki adet 1 içerir).

KARŞILAŞTIRMA

`CMP AL, BL` Bayrakları Nasıl Ayarlar?

AL	0000 0111	07h	işaretsiz 7 · işaretli 7
BL	0000 0111	07h	işaretsiz 7 · işaretli 7
AL−BL	0000 0000	00h	sonuç saklanmaz, yalnız bayraklar

CF=0 ZF=1 SF=0 OF=0 PF=1 AF=0

işaretsiz: AL = BL

CF=0, ZF=1 → JE atlar

işaretli: AL = BL

SF=0, OF=0, ZF=1 → JE atlar

Eşit sayılar: sonuç 0 → **ZF = 1**. Hem işaretli hem işaretsiz bakışta eşitlik.

KARŞILAŞTIRMA

`CMP AL, BL` Bayrakları Nasıl Ayarlar?

AL	0000 0101	05h	işaretsiz 5 · işaretli 5
BL	0000 0111	07h	işaretsiz 7 · işaretli 7
AL−BL	1111 1110	FEh	sonuç saklanmaz, yalnız bayraklar

`CF=1` `ZF=0` `SF=1` `OF=0` `PF=0` `AF=1`

işaretsiz: AL < BL

CF=1, ZF=0 → JB atlar

işaretli: AL < BL

SF=1, OF=0, ZF=0 → JL atlar

Küçükten büyük çıkarılınca ödünç alınır → **CF = 1**; sonuç FEh = −2 → **SF = 1**, taşma yok.

KARŞILAŞTIRMA

`CMP AL, BL` Bayrakları Nasıl Ayarlar?

AL	`0111 1111`	`7Fh`	işaretsiz 127 · işaretli 127
BL	`1000 0000`	`80h`	işaretsiz 128 · işaretli -128
AL−BL	`1111 1111`	`FFh`	sonuç saklanmaz, yalnız bayraklar

`CF=1` `ZF=0` `SF=1` `OF=1` `PF=1` `AF=0`

işaretsiz: AL < BL

CF=1, ZF=0 → JB atlar

işaretli: AL > BL

SF=1, OF=1, ZF=0 → JG atlar

127 − (−128) = 255 işaretli 8 bite sığmaz → **OF = 1**. İşaretsiz bakışta 127 < 128 (CF = 1), işaretli bakışta 127 > −128 (SF = OF).

KARŞILAŞTIRMA

`CMP AL, BL` Bayrakları Nasıl Ayarlar?

AL	`1111 1011`	`FBh`	işaretsiz 251 · işaretli -5
BL	`0000 0011`	`03h`	işaretsiz 3 · işaretli 3
AL−BL	`1111 1000`	`F8h`	sonuç saklanmaz, yalnız bayraklar

`CF=0` `ZF=0` `SF=1` `OF=0` `PF=0` `AF=0`

işaretsiz: AL > BL

CF=0, ZF=0 → JA atlar

işaretli: AL < BL

SF=1, OF=0, ZF=0 → JL atlar

FBh işaretsiz 251, işaretli −5. İşaretsiz: 251 > 3 (CF = 0); işaretli: −5 < 3 (SF ≠ OF). Aynı bitler, farklı karar!

KARŞILAŞTIRMA

Karşılaştırma Hesaplayıcı

İki bayt girilince (onluk −128…255 ya da `7Fh` gibi onaltılık) `CMP A, B` sonucu, bayraklar ve hangi atlamaların gerçekleşeceği gösterilir.

A: FBh

B: 3

A = 1111 1011 (FBh: 251 / −5)

B = 0000 0011 (03h: 3 / 3)

A − B = 1111 1000 = **F8h**

CF=0 ZF=0 SF=1 OF=0 PF=0 AF=0

~~JE/JZ~~ JNE/JNZ JA JAE/JNC ~~JB/JC~~ ~~JBE~~ ~~JG~~ ~~JGE~~ JL JLE JS ~~JNS~~ ~~JO~~ JNO ~~JP~~ JNP

işaretsiz: 251 > 3 · işaretli: -5 < 3

KARŞILAŞTIRMA

İşaretli Sayılar İçin Atlamalar

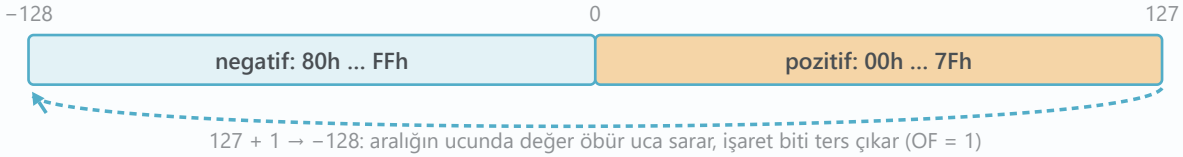
Komut	Anlamı	Koşul	Zıt komut
`JE, JZ`	Jump if Equal (=) · Jump if Zero	ZF = 1	JNE, JNZ
`JNE, JNZ`	Jump if Not Equal (<>) · Jump if Not Zero	ZF = 0	JE, JZ
`JG, JNLE`	Jump if Greater (>) · Not Less or Equal	ZF = 0 ve SF = OF	JNG, JLE
`JL, JNGE`	Jump if Less (<) · Not Greater or Equal	SF ≠ OF	JNL, JGE
`JGE, JNL`	Jump if Greater or Equal (>=) · Not Less	SF = OF	JNGE, JL
`JLE, JNG`	Jump if Less or Equal (<=) · Not Greater	ZF = 1 veya SF ≠ OF	JNLE, JG

`<>` işareti "eşit değil" anlamına gelir. **G**reater / **L**ess sözcükleri, sayıların **ikiye tümleyen** gösterimde işaretli olduğu varsayımıyla kullanılır.

KARŞILAŞTIRMA

Neden SF = OF?

- CMP A, B → A − B hesaplanır. Taşma yoksa sonucun işareti doğrudur: **SF = 1 ⇔ A < B**.
- Taşma varsa (OF = 1) sonuç 8/16 bite sığmamış, **işaret biti ters** çıkmıştır.
- Bu yüzden gerçek işaret = SF XOR OF: **SF ≠ OF ⇔ A < B**, **SF = OF ⇔ A ≥ B**.



A	B	A − B (8 bit)	SF	OF	Sonuç
5	7	FEh (−2)	1	0	SF ≠ OF → 5 < 7
127 (7Fh)	−128 (80h)	FFh (−1) ✗ gerçek: +255	1	1	SF = OF → 127 > −128
−128 (80h)	1	7Fh (+127) ✗ gerçek: −129	0	1	SF ≠ OF → −128 < 1
−5 (FBh)	3	F8h (−8)	1	0	SF ≠ OF → −5 < 3

Taşmalı satırlarda SF tek başına yanlış cevap verirdi; OF ile birlikte değerlendirilince karar doğru olur.

KARŞILAŞTIRMA

İşaretsiz Sayılar İçin Atlamalar

Komut	Anlamı	Koşul	Zıt komut
`JE, JZ`	Jump if Equal (=) · Jump if Zero	ZF = 1	JNE, JNZ
`JNE, JNZ`	Jump if Not Equal (<>) · Jump if Not Zero	ZF = 0	JE, JZ
`JA, JNBE`	Jump if Above (>) · Not Below or Equal	CF = 0 ve ZF = 0	JNA, JBE
`JB, JNAE, JC`	Jump if Below (<) · Not Above or Equal · Carry	CF = 1	JNB, JAE, JNC
`JAE, JNB, JNC`	Jump if Above or Equal (>=) · Not Below · Not Carry	CF = 0	JNAE, JB, JC
`JBE, JNA`	Jump if Below or Equal (<=) · Not Above	CF = 1 veya ZF = 1	JNBE, JA

İşaretsiz çıkarmada A < B ise ödünç alınır ve **CF = 1** olur. Bu nedenle "aşağıda" (below) testi doğrudan elde bayrağıdır: JB ≡ JC.

KARŞILAŞTIRMA

Above/Below mı, Greater/Less mi?

Karşılaştırma	İşaretsiz (A/B)	İşaretli (G/L)
A = B	JE / JZ	JE / JZ
A ≠ B	JNE / JNZ	JNE / JNZ
A > B	JA / JNBE	JG / JNLE
A ≥ B	JAE / JNB	JGE / JNL
A < B	JB / JNAE	JL / JNGE
A ≤ B	JBE / JNA	JLE / JNG

Hatırlatıcı

Above / **B**elow → işaretsiz (adresler, sayaçlar, ASCII kodları). **G**reater / **L**ess → işaretli (sıcaklık, fark, negatif olabilen değerler).

Tipik hata

AL = FBh, BL = 03h için `JA` atlar (251 > 3) ama `JG` atlamaz (−5 < 3). Verinin türüne göre doğru aile seçilmelidir.

Kod İzleme: Aynı Bitler, JA ve JG

```
▶ MOV  AL, 0FBh     ; 251 ya da -5
▶ MOV  BL, 03h
3 CMP  AL, BL       ; FBh - 03h
4 JA   a1           ; işaretsiz: 251 > 3 ?
5 MOV  CL, 1        ; JA atlamasaydı
6 a1:  JG  g1       ; işaretli: -5 > 3 ?
7 MOV  CH, 1        ; JG atlamadı
8 g1:  MOV AL, 7Fh  ; 127
9 CMP  AL, 80h      ; 127 - (-128)
10 JG  g2           ; SF = OF ?
11 MOV DL, 1        ; JG atlamasaydı
12 g2: RET
```

DEĞİŞKENLER

AL = FBh | CF = 0 | SF = 0 | OF = 0

AL	BL	CH	DL
FBh	03h	00h	00h

CF	ZF	SF	OF
0	0	0	0

AL = 251
işaretli -5

AL = FBh: işaretsiz 251, işaretli −5. BL = 3. CL, CH, DL hangi atlamanın atlamadığını işaretleyecek.

Adım 1 / 9

Kod İzleme: Aynı Bitler, JA ve JG

```
1  MOV  AL, 0FBh     ; 251 ya da -5
2  MOV  BL, 03h
▶  CMP  AL, BL       ; FBh - 03h
4  JA   a1           ; işaretsiz: 251 > 3 ?
5  MOV  CL, 1        ; JA atlamasaydı
6  a1:  JG  g1       ; işaretli: -5 > 3 ?
7  MOV  CH, 1        ; JG atlamadı
8  g1:  MOV AL, 7Fh  ; 127
9  CMP  AL, 80h      ; 127 - (-128)
10 JG   g2           ; SF = OF ?
11 MOV  DL, 1        ; JG atlamasaydı
12 g2:  RET
```

DEĞİŞKENLER

`AL = FBh` `CF = 0` `SF = 1` `OF = 0`

AL	BL	CH	DL
FBh	03h	00h	00h

CF	ZF	SF	OF
0	0	1	0

AL = 251
işaretli -5

`CMP AL, BL` : FBh − 03h = F8h. Ödünç yok → CF = 0; sonuç negatif → SF = 1; taşma yok → OF = 0.

Kod İzleme: Aynı Bitler, JA ve JG

```
1  MOV  AL, 0FBh     ; 251 ya da -5
2  MOV  BL, 03h
3  CMP  AL, BL       ; FBh - 03h
▶  JA   a1           ; işaretsiz: 251 > 3 ?
5  MOV  CL, 1        ; JA atlamasaydı
6  a1:  JG  g1       ; işaretli: -5 > 3 ?
7  MOV  CH, 1        ; JG atlamadı
8  g1:  MOV AL, 7Fh  ; 127
9  CMP  AL, 80h      ; 127 - (-128)
10 JG   g2           ; SF = OF ?
11 MOV  DL, 1        ; JG atlamasaydı
12 g2:  RET
```

DEĞİŞKENLER

`AL = FBh` `CF = 0` `SF = 1` `OF = 0`

AL	BL	CH	DL
FBh	03h	00h	00h

CF	ZF	SF	OF
0	0	1	0

AL = 251
işaretli -5

JA → atlar

`JA` : CF = 0 ve ZF = 0 → atlar (251 > 3). CL = 1 satırı atlandı.

Adım 3 / 9

Kod İzleme: Aynı Bitler, JA ve JG

```
1  MOV  AL, 0FBh     ; 251 ya da -5
2  MOV  BL, 03h
3  CMP  AL, BL       ; FBh - 03h
4  JA   a1           ; işaretsiz: 251 > 3 ?
5  MOV  CL, 1        ; JA atlamasaydı
▶ a1: JG  g1         ; işaretli: -5 > 3 ?
7  MOV  CH, 1        ; JG atlamadı
8  g1:  MOV AL, 7Fh  ; 127
9  CMP  AL, 80h      ; 127 - (-128)
10 JG   g2           ; SF = OF ?
11 MOV  DL, 1        ; JG atlamasaydı
12 g2:  RET
```

DEĞİŞKENLER

AL = **FBh** | CF = **0** | SF = **1** | OF = **0**

AL	BL	CH	DL
FBh	03h	00h	00h

CF	ZF	SF	OF
0	0	1	0

AL = 251
işaretli -5

JG → atlamaz

`JG` : ZF = 0 ama SF ≠ OF → atlamaz (−5 < 3).

Adım 4 / 9

Kod İzleme: Aynı Bitler, JA ve JG

```
1  MOV  AL, 0FBh     ; 251 ya da -5
2  MOV  BL, 03h
3  CMP  AL, BL       ; FBh - 03h
4  JA   a1           ; işaretsiz: 251 > 3 ?
5  MOV  CL, 1        ; JA atlamasaydı
6  a1:  JG  g1       ; işaretli: -5 > 3 ?
▶  MOV  CH, 1        ; JG atlamadı
8  g1:  MOV AL, 7Fh  ; 127
9  CMP  AL, 80h      ; 127 - (-128)
10 JG   g2           ; SF = OF ?
11 MOV  DL, 1        ; JG atlamasaydı
12 g2:  RET
```

DEĞİŞKENLER

AL = **FBh** | CF = **0** | SF = **1** | OF = **0**

AL	BL	CH	DL
FBh	03h	01h	00h

CF	ZF	SF	OF
0	0	1	0

AL = 251
işaretli -5

CH = 1: işaretli karşılaştırmada AL büyük değil. Aynı CMP, iki farklı karar.

Kod İzleme: Aynı Bitler, JA ve JG

```
1  MOV  AL, 0FBh     ; 251 ya da -5
2  MOV  BL, 03h
3  CMP  AL, BL       ; FBh - 03h
4  JA   a1           ; işaretsiz: 251 > 3 ?
5  MOV  CL, 1        ; JA atlamasaydı
6  a1:  JG  g1       ; işaretli: -5 > 3 ?
7  MOV  CH, 1        ; JG atlamadı
▶  g1:  MOV AL, 7Fh  ; 127
9  CMP  AL, 80h      ; 127 - (-128)
10 JG   g2           ; SF = OF ?
11 MOV  DL, 1        ; JG atlamasaydı
12 g2:  RET
```

DEĞİŞKENLER

AL = 7Fh | CF = 0 | SF = 1 | OF = 0

AL	BL	CH	DL
7Fh	03h	01h	00h

CF	ZF	SF	OF
0	0	1	0

AL = 127
işaretli 127

AL = 7Fh = 127. Şimdi 127 ile 80h (−128) karşılaştırılacak.

Adım 6 / 9

Kod İzleme: Aynı Bitler, JA ve JG

```
1  MOV  AL, 0FBh     ; 251 ya da -5
2  MOV  BL, 03h
3  CMP  AL, BL       ; FBh - 03h
4  JA   a1           ; işaretsiz: 251 > 3 ?
5  MOV  CL, 1        ; JA atlamasaydı
6  a1:  JG  g1       ; işaretli: -5 > 3 ?
7  MOV  CH, 1        ; JG atlamadı
8  g1:  MOV AL, 7Fh  ; 127
▶  CMP  AL, 80h      ; 127 - (-128)
10 JG   g2           ; SF = OF ?
11 MOV  DL, 1        ; JG atlamasaydı
12 g2:  RET
```

DEĞİŞKENLER

AL = **7Fh** | CF = **1** | SF = **1** | OF = **1**

AL	BL	CH	DL
7Fh	03h	01h	00h

CF	ZF	SF	OF
1	0	1	1

AL = 127
işaretli 127

7Fh − 80h = FFh: işaretli 127 − (−128) = 255 sığmaz → OF = 1, sonucun işaret biti SF = 1. Ödünç → CF = 1.

Kod İzleme: Aynı Bitler, JA ve JG

```
1  MOV  AL, 0FBh     ; 251 ya da -5
2  MOV  BL, 03h
3  CMP  AL, BL       ; FBh - 03h
4  JA   a1           ; işaretsiz: 251 > 3 ?
5  MOV  CL, 1        ; JA atlamasaydı
6  a1:  JG  g1       ; işaretli: -5 > 3 ?
7  MOV  CH, 1        ; JG atlamadı
8  g1:  MOV AL, 7Fh  ; 127
9  CMP  AL, 80h      ; 127 - (-128)
▶  JG   g2           ; SF = OF ?
11 MOV  DL, 1        ; JG atlamasaydı
12 g2:  RET
```

DEĞİŞKENLER

AL = **7Fh** | CF = **1** | SF = **1** | OF = **1**

AL	BL	CH	DL
7Fh	03h	01h	00h

CF	ZF	SF	OF
1	0	1	1

AL = 127
işaretli 127

JG → atlar

`JG` : ZF = 0 ve SF = OF (1 = 1) → atlar: 127 > −128 doğru. SF tek başına “küçük” derdi.

Adım 8 / 9

Kod İzleme: Aynı Bitler, JA ve JG

```
1  MOV  AL, 0FBh     ; 251 ya da -5
2  MOV  BL, 03h
3  CMP  AL, BL       ; FBh - 03h
4  JA   a1           ; işaretsiz: 251 > 3 ?
5  MOV  CL, 1        ; JA atlamasaydı
6  a1:  JG  g1       ; işaretli: -5 > 3 ?
7  MOV  CH, 1        ; JG atlamadı
8  g1:  MOV AL, 7Fh  ; 127
9  CMP  AL, 80h      ; 127 - (-128)
10 JG   g2           ; SF = OF ?
11 MOV  DL, 1        ; JG atlamasaydı
▶  g2:  RET
```

DEĞİŞKENLER

AL = **7Fh** | CF = **1** | SF = **1** | OF = **1**

AL	BL	CH	DL
7Fh	03h	01h	00h

CF	ZF	SF	OF
1	0	1	1

AL = 127
işaretli 127

DL = 0 kaldı. Sonuç: CH = 1 (−5 > 3 değil), DL = 0 (127 > −128).

Adım 9 / 9

Kod İzleme: `CMP` + `JE`

```
▶ mov ax, 5
2 mov bx, 5
3 cmp ax, bx
4 je  equal_message
5 jmp not_equal_message
6 equal_message:        ; eşitse
7 jmp end_program
8 not_equal_message:    ; eşit değilse
9 end_program:
```

DEĞİŞKENLER

`AX = 0005h`

AX = 0005h.

Adım 1 / 7

Kod İzleme: `CMP` + `JE`

```
1 mov ax, 5
▶ mov bx, 5
3 cmp ax, bx
4 je  equal_message
5 jmp not_equal_message
6 equal_message:        ; eşitse
7 jmp end_program
8 not_equal_message:    ; eşit değilse
9 end_program:
```

DEĞİŞKENLER

AX = **0005h** BX = **0005h**

BX = 0005h.

Adım 2 / 7

Kod İzleme: `CMP` + `JE`

```
1 mov ax, 5
2 mov bx, 5
▶ cmp ax, bx
4 je  equal_message
5 jmp not_equal_message
6 equal_message:        ; eşitse
7 jmp end_program
8 not_equal_message:    ; eşit değilse
9 end_program:
```

DEĞİŞKENLER

`AX = 0005h` `BX = 0005h` `ZF = 1` `CF = 0` `SF = 0` `OF = 0`

`cmp ax, bx` : 5 − 5 = 0 → **ZF = 1**. AX değişmez.

Adım 3 / 7

Kod İzleme: `CMP` + `JE`

```
1 mov ax, 5
2 mov bx, 5
3 cmp ax, bx
▶ je  equal_message
5 jmp not_equal_message
6 equal_message:        ; eşitse
7 jmp end_program
8 not_equal_message:    ; eşit değilse
9 end_program:
```

DEĞİŞKENLER

AX = **0005h** | BX = **0005h** | ZF = **1**

CF = **0** | SF = **0** | OF = **0**

ZF = 1 olduğundan `je` atlar.

Adım 4 / 7

Kod İzleme: `CMP` + `JE`

```
1 mov ax, 5
2 mov bx, 5
3 cmp ax, bx
4 je  equal_message
5 jmp not_equal_message
▶ equal_message:        ; eşitse
7 jmp end_program
8 not_equal_message:    ; eşit değilse
9 end_program:
```

DEĞİŞKENLER

`AX = 0005h` `BX = 0005h` `ZF = 1`
`CF = 0` `SF = 0` `OF = 0`

`equal_message:` eşitlik durumundaki işlemler burada yapılır.

Adım 5 / 7

Kod İzleme: `CMP` + `JE`

```
1 mov ax, 5
2 mov bx, 5
3 cmp ax, bx
4 je  equal_message
5 jmp not_equal_message
6 equal_message:        ; eşitse
▶ jmp end_program
8 not_equal_message:    ; eşit değilse
9 end_program:
```

DEĞİŞKENLER

AX = **0005h** | BX = **0005h** | ZF = **1**

CF = **0** | SF = **0** | OF = **0**

`jmp end_program` : `not_equal_message` bloğu atlanır. Bu JMP olmasaydı her iki blok da yürütülürdü.

Adım 6 / 7

Kod İzleme: `CMP` + `JE`

```
1 mov ax, 5
2 mov bx, 5
3 cmp ax, bx
4 je  equal_message
5 jmp not_equal_message
6 equal_message:        ; eşitse
7 jmp end_program
8 not_equal_message:    ; eşit değilse
▶ end_program:
```

DEĞİŞKENLER

`AX = 0005h` `BX = 0005h` `ZF = 1`
`CF = 0` `SF = 0` `OF = 0`

Program sonu. Eşit olmasaydı: `je` atlamaz, 5. satırdaki `jmp` ile 8. satıra gidilirdi.

Adım 7 / 7

Kod İzleme: `CMP` + `JA`

```
▶ mov ax, 5
2 mov bx, 7
3 cmp ax, bx
4 ja  jump_above
5 jmp not_jump_above
6 jump_above:        ; ax büyükse yapılacaklar
7 jmp end_program
8 not_jump_above:    ; ax küçük veya eşitse
9 end_program:
```

DEĞİŞKENLER

`AX = 0005h`

AX = 0005h.

Adım 1 / 7

Kod İzleme: `CMP` + `JA`

```
1 mov ax, 5
▶ mov bx, 7
3 cmp ax, bx
4 ja  jump_above
5 jmp not_jump_above
6 jump_above:        ; ax büyükse yapılacaklar
7 jmp end_program
8 not_jump_above:    ; ax küçük veya eşitse
9 end_program:
```

DEĞİŞKENLER

AX = **0005h** BX = **0007h**

BX = 0007h.

Adım 2 / 7

Kod İzleme: `CMP` + `JA`

```
1 mov ax, 5
2 mov bx, 7
▶ cmp ax, bx
4 ja  jump_above
5 jmp not_jump_above
6 jump_above:        ; ax büyükse yapılacaklar
7 jmp end_program
8 not_jump_above:    ; ax küçük veya eşitse
9 end_program:
```

DEĞİŞKENLER

AX = **0005h** | BX = **0007h** | ZF = **0** | CF = **1** | SF = **1** | OF = **0**

`cmp ax, bx` : 5 − 7 = FFFEh (−2). Ödünç alındı → **CF = 1**, ZF = 0.

Adım 3 / 7

Kod İzleme: `CMP` + `JA`

```
1 mov ax, 5
2 mov bx, 7
3 cmp ax, bx
▶ ja  jump_above
5 jmp not_jump_above
6 jump_above:        ; ax büyükse yapılacaklar
7 jmp end_program
8 not_jump_above:    ; ax küçük veya eşitse
9 end_program:
```

DEĞİŞKENLER

`AX = 0005h` `BX = 0007h` `ZF = 0` `CF = 1` `SF = 1` `OF = 0`

`ja` için CF = 0 **ve** ZF = 0 gerekir. CF = 1 → atlamaz.

Adım 4 / 7

Kod İzleme: `CMP` + `JA`

```
1 mov ax, 5
2 mov bx, 7
3 cmp ax, bx
4 ja  jump_above
▶ jmp not_jump_above
6 jump_above:        ; ax büyükse yapılacaklar
7 jmp end_program
8 not_jump_above:    ; ax küçük veya eşitse
9 end_program:
```

DEĞİŞKENLER

`AX = 0005h` `BX = 0007h` `ZF = 0`
`CF = 1` `SF = 1` `OF = 0`

`jmp not_jump_above` ile "küçük veya eşit" bloğuna gidilir.

Adım 5 / 7

Kod İzleme: `CMP` + `JA`

```
1 mov ax, 5
2 mov bx, 7
3 cmp ax, bx
4 ja  jump_above
5 jmp not_jump_above
6 jump_above:       ; ax büyükse yapılacaklar
7 jmp end_program
▶ not_jump_above:   ; ax küçük veya eşitse
9 end_program:
```

DEĞİŞKENLER

AX = **0005h** | BX = **0007h** | ZF = **0** | CF = **1** | SF = **1** | OF = **0**

`not_jump_above:` AX ≤ BX durumundaki işlemler.

Adım 6 / 7

Kod İzleme: `CMP` + `JA`

```
1 mov ax, 5
2 mov bx, 7
3 cmp ax, bx
4 ja  jump_above
5 jmp not_jump_above
6 jump_above:        ; ax büyükse yapılacaklar
7 jmp end_program
8 not_jump_above:    ; ax küçük veya eşitse
▶ end_program:
```

DEĞİŞKENLER

`AX = 0005h` `BX = 0007h` `ZF = 0`
`CF = 1` `SF = 1` `OF = 0`

Program sonu. Aynı kodda `ja` yerine `jg` kullanılsaydı da (SF ≠ OF) atlanmazdı; değerler pozitif olduğu için iki aile aynı sonucu verir.

Adım 7 / 7

Kod İzleme: Karşılaştır ve Yaz

```
▶ org     100h
2 mov     al, 25      ; al = 25
3 mov     bl, 10      ; bl = 10
4 cmp     al, bl      ; al - bl karşılaştır
5 je      equal       ; al = bl ise (zf = 1) atla
6 mov     dl, 'n'     ; al <> bl: 'n' yazılacak
7 jmp     yaz         ; ortak yazma satırına git
8 equal:
9 mov     dl, 'y'     ; al = bl: 'y' yazılacak
10 yaz:
11 mov    ah, 2       ; DOS: DL'deki karakteri
   yaz
12 int    21h
13 ret                ; DOS'a dön
```

DEĞİŞKENLER

—

ÇIKTI

[redacted]

Karakter DOS hizmetiyle yazılır: `INT 21h`, AH = 02h, DL'deki karakteri ekrana yazar.

Adım 1 / 10

Kod İzleme: Karşılaştır ve Yaz

```
1  org     100h
▶  mov     al, 25      ; al = 25
3  mov     bl, 10      ; bl = 10
4  cmp     al, bl      ; al - bl karşılaştır
5  je      equal       ; al = bl ise (zf = 1) atla
6  mov     dl, 'n'     ; al <> bl: 'n' yazılacak
7  jmp     yaz         ; ortak yazma satırına git
8  equal:
9  mov     dl, 'y'     ; al = bl: 'y' yazılacak
10 yaz:
11 mov     ah, 2       ; DOS: DL'deki karakteri
   yaz
12 int     21h
13 ret                 ; DOS'a dön
```

DEĞİŞKENLER

AL = **19h**

ÇIKTI

[redacted]

AL = 25 = **19h**.

Adım 2 / 10

Kod İzleme: Karşılaştır ve Yaz

```
1  org   100h
2  mov   al, 25      ; al = 25
▶  mov   bl, 10      ; bl = 10
4  cmp   al, bl      ; al - bl karşılaştır
5  je    equal       ; al = bl ise (zf = 1) atla
6  mov   dl, 'n'     ; al <> bl: 'n' yazılacak
7  jmp   yaz         ; ortak yazma satırına git
8  equal:
9  mov   dl, 'y'     ; al = bl: 'y' yazılacak
10 yaz:
11 mov   ah, 2       ; DOS: DL'deki karakteri
   yaz
12 int   21h
13 ret               ; DOS'a dön
```

DEĞİŞKENLER

AL = **19h** BL = **0Ah**

ÇIKTI

[redacted]

BL = 10 = **0Ah**.

Adım 3 / 10

Kod İzleme: Karşılaştır ve Yaz

```
1  org     100h
2  mov     al, 25      ; al = 25
3  mov     bl, 10      ; bl = 10
▶  cmp     al, bl      ; al - bl karşılaştır
5  je      equal       ; al = bl ise (zf = 1) atla
6  mov     dl, 'n'     ; al <> bl: 'n' yazılacak
7  jmp     yaz         ; ortak yazma satırına git
8  equal:
9  mov     dl, 'y'     ; al = bl: 'y' yazılacak
10 yaz:
11 mov     ah, 2       ; DOS: DL'deki karakteri
   yaz
12 int     21h
13 ret                 ; DOS'a dön
```

DEĞİŞKENLER

AL = **19h** BL = **0Ah** ZF = **0** CF = **0** SF = **0** OF = **0**

ÇIKTI

`cmp al, bl` : 25 − 10 = 15 (0Fh) → ZF = 0.

Adım 4 / 10

Kod İzleme: Karşılaştır ve Yaz

```
1  org    100h
2  mov    al, 25      ; al = 25
3  mov    bl, 10      ; bl = 10
4  cmp    al, bl      ; al - bl karşılaştır
▶  je     equal       ; al = bl ise (zf = 1) atla
6  mov    dl, 'n'     ; al <> bl: 'n' yazılacak
7  jmp    yaz         ; ortak yazma satırına git
8  equal:
9  mov    dl, 'y'     ; al = bl: 'y' yazılacak
10 yaz:
11 mov    ah, 2       ; DOS: DL'deki karakteri
   yaz
12 int    21h
13 ret                ; DOS'a dön
```

DEĞİŞKENLER

`AL = 19h` `BL = 0Ah` `ZF = 0` `CF = 0` `SF = 0` `OF = 0`

ÇIKTI

ZF = 0 → `je` atlamaz.

Adım 5 / 10

Kod İzleme: Karşılaştır ve Yaz

```
1  org    100h
2  mov    al, 25      ; al = 25
3  mov    bl, 10      ; bl = 10
4  cmp    al, bl      ; al - bl karşılaştır
5  je     equal       ; al = bl ise (zf = 1) atla
▶  mov    dl, 'n'     ; al <> bl: 'n' yazılacak
7  jmp    yaz         ; ortak yazma satırına git
8  equal:
9  mov    dl, 'y'     ; al = bl: 'y' yazılacak
10 yaz:
11 mov    ah, 2       ; DOS: DL'deki karakteri
   yaz
12 int    21h
13 ret                ; DOS'a dön
```

DEĞİŞKENLER

AL = **19h** | BL = **0Ah** | DL = **6Eh** | ZF = **0** | CF = **0** | SF = **0** | OF = **0**

ÇIKTI

`mov dl, 'n'` : yazılacak karakter DL'de (6Eh).

Adım 6 / 10

Kod İzleme: Karşılaştır ve Yaz

```
1  org    100h
2  mov    al, 25      ; al = 25
3  mov    bl, 10      ; bl = 10
4  cmp    al, bl      ; al - bl karşılaştır
5  je     equal       ; al = bl ise (zf = 1) atla
6  mov    dl, 'n'     ; al <> bl: 'n' yazılacak
▶  jmp    yaz         ; ortak yazma satırına git
8  equal:
9  mov    dl, 'y'     ; al = bl: 'y' yazılacak
10 yaz:
11 mov    ah, 2       ; DOS: DL'deki karakteri
   yaz
12 int    21h
13 ret                ; DOS'a dön
```

DEĞİŞKENLER

`AL = 19h` `BL = 0Ah` `DL = 6Eh`

`ZF = 0` `CF = 0` `SF = 0` `OF = 0`

ÇIKTI

`jmp yaz` : 'y' satırı atlanır; iki dal `yaz` etiketinde birleşir.

Adım 7 / 10

Kod İzleme: Karşılaştır ve Yaz

```
1  org   100h
2  mov   al, 25      ; al = 25
3  mov   bl, 10      ; bl = 10
4  cmp   al, bl      ; al - bl karşılaştır
5  je    equal       ; al = bl ise (zf = 1) atla
6  mov   dl, 'n'     ; al <> bl: 'n' yazılacak
7  jmp   yaz         ; ortak yazma satırına git
8  equal:
9  mov   dl, 'y'     ; al = bl: 'y' yazılacak
10 yaz:
▶  mov   ah, 2       ; DOS: DL'deki karakteri
   yaz
12 int   21h
13 ret               ; DOS'a dön
```

DEĞİŞKENLER

AL = **19h** | BL = **0Ah** | DL = **6Eh** | AH = **02h** | ZF = **0** | CF = **0** | SF = **0** | OF = **0**

ÇIKTI

AH = 02h: DOS karakter yazma hizmeti.

Adım 8 / 10

Kod İzleme: Karşılaştır ve Yaz

```
1  org    100h
2  mov    al, 25      ; al = 25
3  mov    bl, 10      ; bl = 10
4  cmp    al, bl      ; al - bl karşılaştır
5  je     equal       ; al = bl ise (zf = 1) atla
6  mov    dl, 'n'     ; al <> bl: 'n' yazılacak
7  jmp    yaz         ; ortak yazma satırına git
8  equal:
9  mov    dl, 'y'     ; al = bl: 'y' yazılacak
10 yaz:
11 mov    ah, 2       ; DOS: DL'deki karakteri
   yaz
▶  int    21h
13 ret                ; DOS'a dön
```

DEĞİŞKENLER

AL = **6Eh** | BL = **0Ah** | DL = **6Eh** | AH = **02h** | ZF = **0** | CF = **0** | SF = **0** | OF = **0**

ÇIKTI

```
n
```

`int 21h` : ekrana **n** yazılır. DOS, yazdığı karakteri AL'ye de koyar: AL = 6Eh.

Adım 9 / 10

Kod İzleme: Karşılaştır ve Yaz

```
1  org    100h
2  mov    al, 25      ; al = 25
3  mov    bl, 10      ; bl = 10
4  cmp    al, bl      ; al - bl karşılaştır
5  je     equal       ; al = bl ise (zf = 1) atla
6  mov    dl, 'n'     ; al <> bl: 'n' yazılacak
7  jmp    yaz         ; ortak yazma satırına git
8  equal:
9  mov    dl, 'y'     ; al = bl: 'y' yazılacak
10 yaz:
11 mov    ah, 2       ; DOS: DL'deki karakteri
   yaz
12 int    21h
▶  ret                ; DOS'a dön
```

DEĞİŞKENLER

AL = **6Eh** BL = **0Ah** DL = **6Eh**
AH = **02h** ZF = **0** CF = **0** SF = **0**
OF = **0**

ÇIKTI

```
n
```

Programın sonunda `ret` bulunmalıdır; yoksa işlemci sonraki rastgele baytları komut sanıp yürütürdü.

Adım 10 / 10

KARŞILAŞTIRMA

if / else Kalıbı

Yüksek Düzey

```
if (ax == bx) {
    ; A bloğu
} else {
    ; B bloğu
}
while (cx > 0) {      ; işaretsiz
    ; gövde
}
```

Assembly

```
    cmp ax, bx
    jne degil       ; koşulun TERSİ
    ; A bloğu
    jmp son
degil:
    ; B bloğu
son:
bas: cmp cx, 0
    jbe bitti       ; cx <= 0 → çık
    ; gövde
    jmp bas
bitti:
```

Koşulun **tersiyle** blok atlanırsa kod, yüksek düzeydeki sırayı korur. A bloğunun sonundaki `jmp son` unutulursa B bloğu da çalışır.

Kod İzleme: if / else ve while

```
▶ MOV  AX, 12
▶ MOV  BX, 30
3  CMP  AX, BX
4  JNE  degil        ; koşulun TERSİ
5  MOV  DX, 1        ; A bloğu: AX = BX
6  JMP  son
7  degil: MOV DX, 2  ; B bloğu: AX ≠ BX
8  son: MOV CX, 2    ; while (CX > 0)
9  bas: CMP CX, 0
10 JBE  bitti        ; CX ≤ 0 → çık
11 ADD  DX, CX       ; gövde
12 DEC  CX
13 JMP  bas
14 bitti: RET
```

DEĞİŞKENLER

AX = 12 | BX = 30 | CX = 0 | DX = 0

AX	BX	CX	DX
000Ch	001Eh	0000h	0000h

ZF	CF
0	0

A bloğu | B bloğu

döngü gövdesi

if (AX == BX) DX = 1; else DX = 2; ardından CX = 2 ile while (CX > 0) DX += CX. AX = 12, BX = 30.

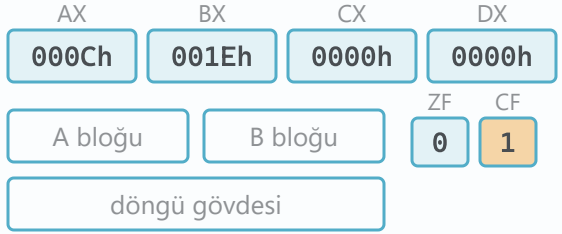
Adım 1 / 10

Kod İzleme: if / else ve while

```
1  MOV  AX, 12
2  MOV  BX, 30
▶  CMP  AX, BX
4  JNE  degil        ; koşulun TERSİ
5  MOV  DX, 1        ; A bloğu: AX = BX
6  JMP  son
7  degil: MOV DX, 2  ; B bloğu: AX ≠ BX
8  son: MOV CX, 2    ; while (CX > 0)
9  bas: CMP CX, 0
10 JBE  bitti        ; CX ≤ 0 → çık
11 ADD  DX, CX       ; gövde
12 DEC  CX
13 JMP  bas
14 bitti: RET
```

DEĞİŞKENLER

AX = 12 | BX = 30 | CX = 0 | DX = 0



`CMP AX, BX` : 12 − 30 → ZF = 0 (eşit değil), ödünç → CF = 1.

Adım 2 / 10

Kod İzleme: if / else ve while

```
1  MOV  AX, 12
2  MOV  BX, 30
3  CMP  AX, BX
▶ JNE  degil        ; koşulun TERSİ
5  MOV  DX, 1       ; A bloğu: AX = BX
6  JMP  son
7  degil: MOV DX, 2 ; B bloğu: AX ≠ BX
8  son: MOV CX, 2   ; while (CX > 0)
9  bas: CMP CX, 0
10 JBE  bitti       ; CX ≤ 0 → çık
11 ADD  DX, CX      ; gövde
12 DEC  CX
13 JMP  bas
14 bitti: RET
```

DEĞİŞKENLER

AX = 12 | BX = 30 | CX = 0 | DX = 0

AX	BX	CX	DX
000Ch	001Eh	0000h	0000h

A bloğu | B bloğu

döngü gövdesi

ZF	CF
0	1

JNE → atlar

`JNE degil` : koşulun tersi sağlandı → A bloğu atlanır.

Adım 3 / 10

Kod İzleme: if / else ve while

```
1  MOV  AX, 12
2  MOV  BX, 30
3  CMP  AX, BX
4  JNE  degil        ; koşulun TERSİ
5  MOV  DX, 1        ; A bloğu: AX = BX
6  JMP  son
▶ degil: MOV DX, 2 ; B bloğu: AX ≠ BX
8  son: MOV CX, 2    ; while (CX > 0)
9  bas: CMP CX, 0
10 JBE  bitti        ; CX ≤ 0 → çık
11 ADD  DX, CX       ; gövde
12 DEC  CX
13 JMP  bas
14 bitti: RET
```

DEĞİŞKENLER

AX = 12 | BX = 30 | CX = 0 | DX = 2

AX	BX	CX	DX
000Ch	001Eh	0000h	0002h

ZF	CF
0	1

A bloğu | **B bloğu**

döngü gövdesi

B bloğu: DX = 2. Sıradaki satır `son` ; A bloğundaki `JMP son` bu yolda gerekmedi.

Adım 4 / 10

Kod İzleme: if / else ve while

```
1  MOV  AX, 12
2  MOV  BX, 30
3  CMP  AX, BX
4  JNE  degil        ; koşulun TERSİ
5  MOV  DX, 1        ; A bloğu: AX = BX
6  JMP  son
7  degil: MOV DX, 2  ; B bloğu: AX ≠ BX
▶  son: MOV CX, 2    ; while (CX > 0)
▶  bas: CMP CX, 0
▶  JBE  bitti        ; CX ≤ 0 → çık
11 ADD  DX, CX       ; gövde
12 DEC  CX
13 JMP  bas
14 bitti: RET
```

DEĞİŞKENLER

AX = 12 | BX = 30 | CX = 2 | DX = 2

AX	BX	CX	DX
000Ch	001Eh	0002h	0002h

ZF	CF
0	0

A bloğu | B bloğu | döngü gövdesi

JBE → atlamaz

CX = 2. `CMP CX, 0` : 2 > 0 → CF = 0, ZF = 0; `JBE` atlamaz, gövdeye girilir.

Adım 5 / 10

Kod İzleme: if / else ve while

```
1  MOV  AX, 12
2  MOV  BX, 30
3  CMP  AX, BX
4  JNE  degil        ; koşulun TERSİ
5  MOV  DX, 1        ; A bloğu: AX = BX
6  JMP  son
7  degil: MOV DX, 2  ; B bloğu: AX ≠ BX
8  son: MOV CX, 2    ; while (CX > 0)
9  bas: CMP CX, 0
10 JBE  bitti        ; CX ≤ 0 → çık
▶  ADD  DX, CX       ; gövde
▶  DEC  CX
13 JMP  bas
14 bitti: RET
```

DEĞİŞKENLER

AX = 12 | BX = 30 | CX = 1 | DX = 4

AX	BX	CX	DX
000Ch	001Eh	0001h	0004h

ZF	CF
0	0

A bloğu | B bloğu

döngü gövdesi

Gövde: DX = 2 + 2 = 4; CX = 1.

Adım 6 / 10

Kod İzleme: if / else ve while

```
1  MOV  AX, 12
2  MOV  BX, 30
3  CMP  AX, BX
4  JNE  degil        ; koşulun TERSİ
5  MOV  DX, 1        ; A bloğu: AX = BX
6  JMP  son
7  degil: MOV DX, 2  ; B bloğu: AX ≠ BX
8  son: MOV CX, 2    ; while (CX > 0)
▶  bas: CMP CX, 0
▶  JBE  bitti        ; CX ≤ 0 → çık
11 ADD  DX, CX       ; gövde
12 DEC  CX
▶  JMP  bas
14 bitti: RET
```

DEĞİŞKENLER

AX = **12** | BX = **30** | CX = **1** | DX = **4**

AX	BX	CX	DX
000Ch	001Eh	0001h	0004h

A bloğu | B bloğu | döngü gövdesi

ZF	CF
0	0

JBE → atlamaz

`JMP bas` : koşul yeniden sınanır; 1 > 0 → JBE atlamaz.

Adım 7 / 10

Kod İzleme: if / else ve while

```
1  MOV  AX, 12
2  MOV  BX, 30
3  CMP  AX, BX
4  JNE  degil        ; koşulun TERSİ
5  MOV  DX, 1        ; A bloğu: AX = BX
6  JMP  son
7  degil: MOV DX, 2  ; B bloğu: AX ≠ BX
8  son: MOV CX, 2    ; while (CX > 0)
9  bas: CMP CX, 0
10 JBE  bitti        ; CX ≤ 0 → çık
▶  ADD  DX, CX       ; gövde
▶  DEC  CX
13 JMP  bas
14 bitti: RET
```

DEĞİŞKENLER

AX = 12 | BX = 30 | CX = 0 | DX = 5

AX	BX	CX	DX
000Ch	001Eh	0000h	0005h

ZF	CF
1	0

A bloğu | B bloğu

döngü gövdesi

DX = 4 + 1 = 5; CX = 0 → DEC ZF = 1 yapar.

Adım 8 / 10

Kod İzleme: if / else ve while

```
1  MOV  AX, 12
2  MOV  BX, 30
3  CMP  AX, BX
4  JNE  degil        ; koşulun TERSİ
5  MOV  DX, 1        ; A bloğu: AX = BX
6  JMP  son
7  degil: MOV DX, 2  ; B bloğu: AX ≠ BX
8  son: MOV CX, 2    ; while (CX > 0)
▶  bas: CMP CX, 0
▶  JBE  bitti        ; CX ≤ 0 → çık
11 ADD  DX, CX       ; gövde
12 DEC  CX
▶  JMP  bas
14 bitti: RET
```

DEĞİŞKENLER

AX = 12 | BX = 30 | CX = 0 | DX = 5

AX	BX	CX	DX
000Ch	001Eh	0000h	0005h

ZF	CF
1	0

A bloğu | B bloğu

döngü gövdesi

JBE → atlar

Başa dönülür: CX = 0 → ZF = 1, `JBE bitti` atlar. Döngü koşulu başta sınandığı için CX = 0 ile hiç girilmezdi.

Adım 9 / 10

Kod İzleme: if / else ve while

```
1  MOV  AX, 12
2  MOV  BX, 30
3  CMP  AX, BX
4  JNE  degil        ; koşulun TERSİ
5  MOV  DX, 1        ; A bloğu: AX = BX
6  JMP  son
7  degil: MOV DX, 2  ; B bloğu: AX ≠ BX
8  son: MOV CX, 2    ; while (CX > 0)
9  bas: CMP CX, 0
10 JBE  bitti        ; CX ≤ 0 → çık
11 ADD  DX, CX       ; gövde
12 DEC  CX
13 JMP  bas
▶  bitti: RET
```

DEĞİŞKENLER

AX = 12 | BX = 30 | CX = 0 | DX = 5

AX	BX	CX	DX
000Ch	001Eh	0000h	0005h

ZF	CF
1	0

A bloğu | B bloğu

döngü gövdesi

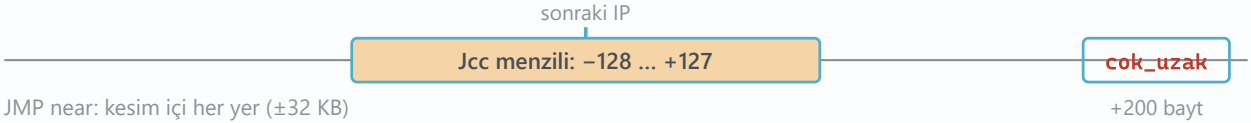
Sonuç DX = 5: B bloğu (2) + 2 + 1.

Adım 10 / 10

KARŞILAŞTIRMA

Menzil Dışı Koşullu Atlama

- 8086'da koşullu atlama yalnızca −128…+127 bayt uzağa gidebilir.
- Hedef daha uzaktaysa assembler hata verir. Çözüm: koşulu **ters çevirip** kısa atlama ile hemen sonraki komutu atlamak, uzak hedefe ise **JMP** (near, ±32 KB) ile gitmek.
- emu8086 bu dönüşümü kendisi yapar: menzil dışı Jcc'yi ters koşullu kısa atlama ve JMP çiftine çevirir; MASM gibi assembler'lar ise hata verir.



Hatalı (hedef çok uzak)

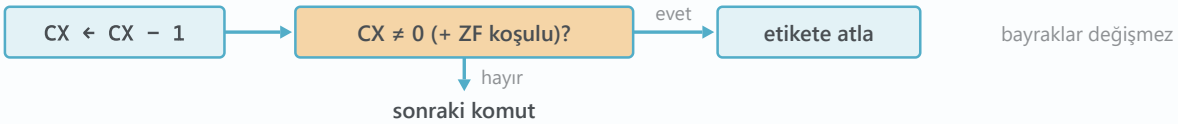
```
cmp al, 0
je  cok_uzak    ; out of range!
; ... 200 baytlık kod ...
cok_uzak:
```

Doğru

```
cmp al, 0
jne devam       ; ters koşul, kısa
jmp cok_uzak    ; near JMP (E9)
devam:
; ... 200 baytlık kod ...
cok_uzak:
```

Döngü Komutları

Komut	İşlem ve atlama koşulu
LOOP	CX'i 1 azalt; CX ≠ 0 ise etikete atla.
LOOPE	CX'i 1 azalt; CX ≠ 0 **ve** eşitse (ZF = 1) atla.
LOOPZ	LOOPE ile aynı: CX ≠ 0 ve ZF = 1 ise atla.
LOOPNE	CX'i 1 azalt; CX ≠ 0 **ve** eşit değilse (ZF = 0) atla.
LOOPNZ	LOOPNE ile aynı: CX ≠ 0 ve ZF = 0 ise atla.
JCXZ	CX = 0 ise etikete atla (CX azaltılmaz).



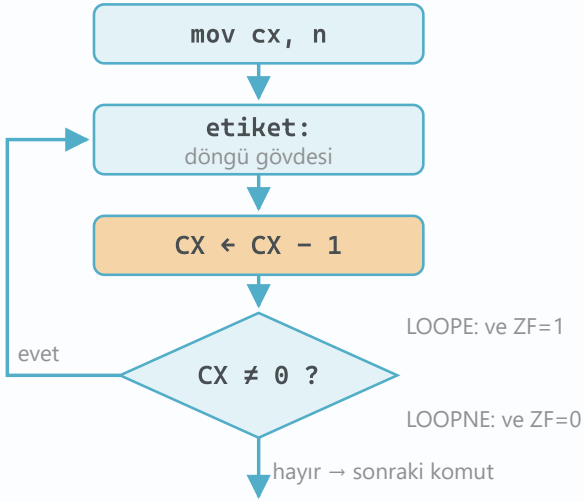
Sayaç
Sayaç her zaman **CX** yazmacıdır.

Bayraklar
LOOP ailesi CX'i azaltırken bayrakları **değiştirmez**.

Menzil
Hepsi 2 bayt, rel8: geriye en fazla 128 bayt.

LOOP Nasıl Çalışır?

- Döngüler, bir kod bloğunun bir koşula bağlı olarak **tekrarlanmasını** sağlar.
- `LOOP etiket` : önce CX azaltılır, CX sıfır değilse etikete atlanır. `DEC CX` + `JNZ` ile aynı etki, ama bayraklar korunur.
- `LOOPE / LOOPZ` : CX ≠ 0 ve ZF = 1 olduğu sürece; `LOOPNE / LOOPNZ` : CX ≠ 0 ve ZF = 0 olduğu sürece atlar.
- `JCXZ` : CX sıfır olduğunda etikete atlar.
- CX = 0 iken döngüye girilirse ilk azaltma CX'i FFFFh yapar → gövde **65.536** kez çalışır.



Kod İzleme: LOOP

```
1 org 100h
▶ mov cx, 5          ; CX = tekrar sayısı (5)
3 dongu:
4 ; döngü içeriği
5 loop dongu         ; CX ≠ 0 ise dongu'ye atla
6 jmp dur            ; döngü bitince dur'a atla
7 dur:
8 ret
```

DEĞİŞKENLER

`CX = 0005h` `tur = 0`

CX = 0005h: gövde 5 kez çalışacak.

Adım 1 / 13

Kod İzleme: LOOP

```
1  org 100h
2  mov cx, 5          ; CX = tekrar sayısı (5)
3  dongu:
▶  ; döngü içeriği
5  loop dongu         ; CX ≠ 0 ise dongu'ye atla
6  jmp dur            ; döngü bitince dur'a atla
7  dur:
8  ret
```

DEĞİŞKENLER

`CX = 0005h` `tur = 1`

1. tur: döngü içeriği yürütülür.

Adım 2 / 13

Kod İzleme: LOOP

```
1 org 100h
2 mov cx, 5          ; CX = tekrar sayısı (5)
3 dongu:
4 ; döngü içeriği
▶ loop dongu         ; CX ≠ 0 ise dongu'ye atla
6 jmp dur            ; döngü bitince dur'a atla
7 dur:
8 ret
```

DEĞİŞKENLER

`CX = 0004h` `tur = 1`

`loop` : CX = 0004h ≠ 0 → `dongu` 'ya geri atla.

Adım 3 / 13

Kod İzleme: LOOP

```
1  org 100h
2  mov cx, 5          ; CX = tekrar sayısı (5)
3  dongu:
▶  ; döngü içeriği
5  loop dongu         ; CX ≠ 0 ise dongu'ye atla
6  jmp dur            ; döngü bitince dur'a atla
7  dur:
8  ret
```

DEĞİŞKENLER

`CX = 0004h` `tur = 2`

2. tur: döngü içeriği yürütülür.

Adım 4 / 13

Kod İzleme: LOOP

```
1 org 100h
2 mov cx, 5          ; CX = tekrar sayısı (5)
3 dongu:
4 ; döngü içeriği
▶ loop dongu         ; CX ≠ 0 ise dongu'ye atla
6 jmp dur            ; döngü bitince dur'a atla
7 dur:
8 ret
```

DEĞİŞKENLER

`CX = 0003h` `tur = 2`

`loop` : CX = 0003h ≠ 0 → `dongu` 'ya geri atla.

Adım 5 / 13

Kod İzleme: LOOP

```
1  org 100h
2  mov cx, 5          ; CX = tekrar sayısı (5)
3  dongu:
▶  ; döngü içeriği
5  loop dongu         ; CX ≠ 0 ise dongu'ye atla
6  jmp dur            ; döngü bitince dur'a atla
7  dur:
8  ret
```

DEĞİŞKENLER

`CX = 0003h` `tur = 3`

3. tur: döngü içeriği yürütülür.

Adım 6 / 13

Kod İzleme: LOOP

```
1 org 100h
2 mov cx, 5          ; CX = tekrar sayısı (5)
3 dongu:
4 ; döngü içeriği
▶ loop dongu         ; CX ≠ 0 ise dongu'ye atla
6 jmp dur            ; döngü bitince dur'a atla
7 dur:
8 ret
```

DEĞİŞKENLER

`CX = 0002h` `tur = 3`

`loop` : CX = 0002h ≠ 0 → `dongu` 'ya geri atla.

Adım 7 / 13

Kod İzleme: LOOP

```
1  org 100h
2  mov cx, 5          ; CX = tekrar sayısı (5)
3  dongu:
▶  ; döngü içeriği
5  loop dongu         ; CX ≠ 0 ise dongu'ye atla
6  jmp dur            ; döngü bitince dur'a atla
7  dur:
8  ret
```

DEĞİŞKENLER

`CX = 0002h` `tur = 4`

4. tur: döngü içeriği yürütülür.

Adım 8 / 13

Kod İzleme: LOOP

```
1 org 100h
2 mov cx, 5          ; CX = tekrar sayısı (5)
3 dongu:
4 ; döngü içeriği
▶ loop dongu         ; CX ≠ 0 ise dongu'ye atla
6 jmp dur            ; döngü bitince dur'a atla
7 dur:
8 ret
```

DEĞİŞKENLER

`CX = 0001h` `tur = 4`

`loop` : CX = 0001h ≠ 0 → `dongu` 'ya geri atla.

Adım 9 / 13

Kod İzleme: LOOP

```
1  org 100h
2  mov cx, 5          ; CX = tekrar sayısı (5)
3  dongu:
▶  ; döngü içeriği
5  loop dongu         ; CX ≠ 0 ise dongu'ye atla
6  jmp dur            ; döngü bitince dur'a atla
7  dur:
8  ret
```

DEĞİŞKENLER

`CX = 0001h` `tur = 5`

5. tur: döngü içeriği yürütülür.

Adım 10 / 13

Kod İzleme: LOOP

```
1 org 100h
2 mov cx, 5          ; CX = tekrar sayısı (5)
3 dongu:
4 ; döngü içeriği
▶ loop dongu         ; CX ≠ 0 ise dongu'ye atla
6 jmp dur            ; döngü bitince dur'a atla
7 dur:
8 ret
```

DEĞİŞKENLER

`CX = 0000h` `tur = 5`

`loop` : CX = 0000h → atlama yok, döngüden çıkılır.

Adım 11 / 13

Kod İzleme: LOOP

```
1  org 100h
2  mov cx, 5          ; CX = tekrar sayısı (5)
3  dongu:
4  ; döngü içeriği
5  loop dongu         ; CX ≠ 0 ise dongu'ye atla
▶  jmp dur            ; döngü bitince dur'a atla
7  dur:
8  ret
```

DEĞİŞKENLER

`CX = 0000h` `tur = 5`

`jmp dur` : `dur` zaten sonraki satır olduğundan bu atlama gereksizdir ama zararsızdır.

Adım 12 / 13

Kod İzleme: LOOP

```
1 org 100h
2 mov cx, 5          ; CX = tekrar sayısı (5)
3 dongu:
4 ; döngü içeriği
5 loop dongu         ; CX ≠ 0 ise dongu'ye atla
6 jmp dur            ; döngü bitince dur'a atla
7 dur:
▶ ret
```

DEĞİŞKENLER

`CX = 0000h` `tur = 5`

`ret` : gövde toplam 5 kez çalıştı.

Adım 13 / 13

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
▶  mov bx, 0        ; toplam adım
3  mov cx, 5        ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h         ; '1' yaz
9   push cx         ; dış sayacı sakla
10  mov cx, 5       ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h        ; '2' yaz
16   loop k2        ; iç döngü
17  pop cx          ; dış sayacı geri al
18 loop k1          ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = **0000h** SP = **FFFEh**

ÇAĞRI YIĞINI

`[FFFEh] 0000h`

ÇIKTI

BX toplam adım sayacı: 0. .COM programında SP = FFFEh.

Adım 1 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
▶  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h        ; '1' yaz
9   push cx        ; dış sayacı sakla
10  mov cx, 5      ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h       ; '2' yaz
16   loop k2       ; iç döngü
17  pop cx         ; dış sayacı geri al
18 loop k1         ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = **0000h** | CX = **0005h** | SP = **FFFEh**

ÇAĞRI YIĞINI

`[FFFEh] 0000h`

ÇIKTI

Dış döngü 5 kez dönecek: CX = 0005h.

Adım 2 / 16

Kod İzleme: İç İçe LOOP

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
▶   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h         ; '1' yaz
9   push cx         ; dış sayacı sakla
10  mov cx, 5       ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h        ; '2' yaz
16   loop k2        ; iç döngü
17  pop cx          ; dış sayacı geri al
18 loop k1          ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = 0001h | CX = 0005h | SP = FFFEh

ÇAĞRI YIĞINI

[FFFEh] 0000h

ÇIKTI

Dış tur 1: BX = 1.

Adım 3 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
▶   mov al, '1'
▶   mov ah, 0eh
▶   int 10h         ; '1' yaz
9   push cx         ; dış sayacı sakla
10  mov cx, 5       ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h        ; '2' yaz
16   loop k2        ; iç döngü
17  pop cx          ; dış sayacı geri al
18 loop k1          ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = **0001h** | CX = **0005h** | AL = **31h** | AH = **0Eh** | SP = **FFFEh**

ÇAĞRI YIĞINI

`[FFFEh] 0000h`

ÇIKTI

```
1
```

AL = '1' (31h), AH = 0Eh → INT 10h teletype ile **1** yazılır. (0Eh işlevi BH'yi sayfa no. olarak kullanır; BX < 256 olduğundan BH = 0.)

Adım 4 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h        ; '1' yaz
▶   push cx        ; dış sayacı sakla
10  mov cx, 5      ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h       ; '2' yaz
16   loop k2       ; iç döngü
17  pop cx         ; dış sayacı geri al
18 loop k1         ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = **0001h** | CX = **0005h** | AL = **31h** | AH = **0Eh** | SP = **FFFCh**

ÇAĞRI YIĞINI

```
[FFFCh] 0005h
[FFFEh] 0000h
```

ÇIKTI

```
1
```

`push cx` : iç döngü de CX kullanacağı için dış sayaç (5) yığına saklanır; SP = FFFCh.

Adım 5 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h        ; '1' yaz
9   push cx        ; dış sayacı sakla
▶   mov cx, 5      ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h       ; '2' yaz
16   loop k2       ; iç döngü
17  pop cx         ; dış sayacı geri al
18 loop k1         ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = **0001h** | CX = **0005h** | AL = **31h** | AH = **0Eh** | SP = **FFFCh**

ÇAĞRI YIĞINI

- **[FFFCh] 0005h**
- [FFFEh] 0000h

ÇIKTI

```
1
```

İç döngü sayacı: CX = 5.

Adım 6 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h         ; '1' yaz
9   push cx         ; dış sayacı sakla
10  mov cx, 5       ; iç döngü sayacı
11 k2:
▶    add bx, 1
▶    mov al, '2'
▶    mov ah, 0eh
▶    int 10h         ; '2' yaz
16   loop k2         ; iç döngü
17  pop cx          ; dış sayacı geri al
18 loop k1          ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = **0002h** | CX = **0005h** | AL = **32h** | AH = **0Eh** | SP = **FFFCh**

ÇAĞRI YIĞINI

- **[FFFCh] 0005h**
- [FFFEh] 0000h

ÇIKTI

```
12
```

İç tur 1: BX = 2, **2** yazılır.

Adım 7 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h        ; '1' yaz
9   push cx        ; dış sayacı sakla
10  mov cx, 5      ; iç döngü sayacı
11 k2:
12    add bx, 1
13    mov al, '2'
14    mov ah, 0eh
15    int 10h      ; '2' yaz
▶     loop k2      ; iç döngü
17  pop cx         ; dış sayacı geri al
18 loop k1         ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = **0002h** | CX = **0004h** | AL = **32h** | AH = **0Eh** | SP = **FFFCh**

ÇAĞRI YIĞINI

`[FFFCh] 0005h`

`[FFFEh] 0000h`

ÇIKTI

```
12
```

`loop k2` : CX = 4 → k2'ye dön.

Adım 8 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h        ; '1' yaz
9   push cx        ; dış sayacı sakla
10  mov cx, 5      ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h       ; '2' yaz
▶    loop k2       ; iç döngü
17  pop cx         ; dış sayacı geri al
18 loop k1         ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = **0006h** | CX = **0000h** | AL = **32h** | AH = **0Eh** | SP = **FFFCh**

ÇAĞRI YIĞINI

- **[FFFCh] 0005h**
- [FFFEh] 0000h

ÇIKTI

```
122222
```

4 iç tur daha: toplam 5 adet **2** yazıldı, BX = 6, CX = 0 → iç döngü biter.

Adım 9 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h        ; '1' yaz
9   push cx        ; dış sayacı sakla
10  mov cx, 5      ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h       ; '2' yaz
16   loop k2       ; iç döngü
▶   pop cx         ; dış sayacı geri al
18 loop k1         ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = **0006h** | CX = **0005h** | AL = **32h** | AH = **0Eh** | SP = **FFFEh**

ÇAĞRI YIĞINI

`[FFFEh] 0000h`

ÇIKTI

```
122222
```

`pop cx` : dış sayaç geri gelir: CX = 5, SP = FFFEh.

Adım 10 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h        ; '1' yaz
9   push cx        ; dış sayacı sakla
10  mov cx, 5      ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h       ; '2' yaz
16   loop k2       ; iç döngü
17  pop cx         ; dış sayacı geri al
▶ loop k1          ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = **0006h** | CX = **0004h** | AL = **32h**

AH = **0Eh** | SP = **FFFEh**

ÇAĞRI YIĞINI

`[FFFEh] 0000h`

ÇIKTI

```
122222
```

`loop k1` : CX = 4 → k1'e dön (dış tur 2).

Adım 11 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h         ; '1' yaz
9   push cx         ; dış sayacı sakla
10  mov cx, 5       ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h         ; '2' yaz
16   loop k2         ; iç döngü
17  pop cx          ; dış sayacı geri al
▶ loop k1          ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = 000Ch | CX = 0003h | AL = 32h | AH = 0Eh | SP = FFFEh

ÇAĞRI YIĞINI

`[FFFEh] 0000h`

ÇIKTI

```
122222122222
```

Dış tur 2 bitti: BX = 12, CX = 3.

Adım 12 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h         ; '1' yaz
9   push cx         ; dış sayacı sakla
10  mov cx, 5       ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h         ; '2' yaz
16   loop k2         ; iç döngü
17  pop cx          ; dış sayacı geri al
▶ loop k1          ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = **0012h** CX = **0002h** AL = **32h**
AH = **0Eh** SP = **FFFEh**

ÇAĞRI YIĞINI

`[FFFEh] 0000h`

ÇIKTI

```
122222122222122222
```

Dış tur 3 bitti: BX = 18, CX = 2.

Adım 13 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h         ; '1' yaz
9   push cx         ; dış sayacı sakla
10  mov cx, 5       ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h         ; '2' yaz
16   loop k2         ; iç döngü
17  pop cx          ; dış sayacı geri al
▶ loop k1          ; dış döngü
19 ret
```

DEĞİŞKENLER

BX = **0018h** | CX = **0001h** | AL = **32h** | AH = **0Eh** | SP = **FFFEh**

ÇAĞRI YIĞINI

`[FFFEh] 0000h`

ÇIKTI

```
122222122222122222122222
```

Dış tur 4 bitti: BX = 24, CX = 1.

Adım 14 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h        ; '1' yaz
9   push cx        ; dış sayacı sakla
10  mov cx, 5      ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h        ; '2' yaz
16   loop k2        ; iç döngü
17  pop cx         ; dış sayacı geri al
▶ loop k1          ; dış döngü
19 ret
```

DEĞİŞKENLER

`BX = 001Eh` `CX = 0000h` `AL = 32h` `AH = 0Eh` `SP = FFFEh`

ÇAĞRI YIĞINI

```
[FFFEh] 0000h
```

ÇIKTI

```
12222212222212222212222212222
```

Dış tur 5 bitti: CX = 0 → dış döngüden çıkılır. BX = 30 = **001Eh** (5 + 5×5 adım).

Adım 15 / 16

Kod İzleme: İç İçe `LOOP`

```
1  org 100h
2  mov bx, 0       ; toplam adım
3  mov cx, 5       ; dış döngü sayacı
4  k1:
5   add bx, 1
6   mov al, '1'
7   mov ah, 0eh
8   int 10h         ; '1' yaz
9   push cx         ; dış sayacı sakla
10  mov cx, 5       ; iç döngü sayacı
11 k2:
12   add bx, 1
13   mov al, '2'
14   mov ah, 0eh
15   int 10h         ; '2' yaz
16   loop k2         ; iç döngü
17  pop cx          ; dış sayacı geri al
18 loop k1          ; dış döngü
▶  ret
```

DEĞİŞKENLER

BX = **001Eh** | CX = **0000h** | AL = **32h** | AH = **0Eh** | SP = **0000h**

ÇAĞRI YIĞINI

boş

ÇIKTI

```
12222212222212222212222212222
```

`ret` . Ekranda 5 kez "122222" vardır. push/pop dengesi sayesinde ret doğru adrese (0000h) döner; SP = 0000h.

Adım 16 / 16

Kod İzleme: `LOOPNE` ile Dizide Arama

```
1  org 100h
▶  mov cx, 5          ; eleman sayısı
3  mov si, -1         ; ilk inc ile 0 olur
4  ara:
5  inc si             ; sonraki eleman
6  cmp dizi[si], 3    ; aranan değer 3 mü?
7  loopne ara         ; CX≠0 ve ZF=0 ise sürdür
8  jne yok            ; ZF=0 → bulunamadı
9  mov al, 1          ; bulundu, konum SI'da
10 ret
11 yok:
12 mov al, 0          ; bulunamadı
13 ret
14 dizi db 7, 1, 3, 9, 4
```

DEĞİŞKENLER

`CX = 0005h`

[0]	[1]	[2]	[3]	[4]
7	1	3	9	4

En fazla 5 eleman incelenecek.

Adım 1 / 13

Kod İzleme: `LOOPNE` ile Dizide Arama

```
1  org 100h
2  mov cx, 5          ; eleman sayısı
▶  mov si, -1         ; ilk inc ile 0 olur
4  ara:
5  inc si             ; sonraki eleman
6  cmp dizi[si], 3    ; aranan değer 3 mü?
7  loopne ara         ; CX≠0 ve ZF=0 ise sürdür
8  jne yok            ; ZF=0 → bulunamadı
9  mov al, 1          ; bulundu, konum SI'da
10 ret
11 yok:
12 mov al, 0          ; bulunamadı
13 ret
14 dizi db 7, 1, 3, 9, 4
```

DEĞİŞKENLER

`CX = 0005h` `SI = FFFFh`

[0]	[1]	[2]	[3]	[4]
7	1	3	9	4

SI = −1 (FFFFh); döngüdeki ilk `inc` onu 0 yapar.

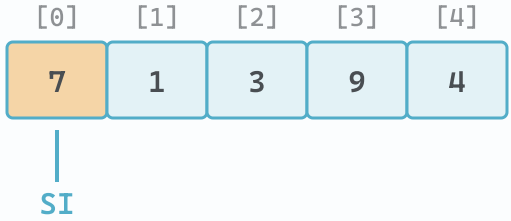
Adım 2 / 13

Kod İzleme: `LOOPNE` ile Dizide Arama

```
1  org 100h
2  mov cx, 5          ; eleman sayısı
3  mov si, -1         ; ilk inc ile 0 olur
4  ara:
▶  inc si             ; sonraki eleman
6  cmp dizi[si], 3    ; aranan değer 3 mü?
7  loopne ara         ; CX≠0 ve ZF=0 ise sürdür
8  jne yok            ; ZF=0 → bulunamadı
9  mov al, 1          ; bulundu, konum SI'da
10 ret
11 yok:
12 mov al, 0          ; bulunamadı
13 ret
14 dizi db 7, 1, 3, 9, 4
```

DEĞİŞKENLER

`CX = 0005h` `SI = 0000h`



SI = 0.

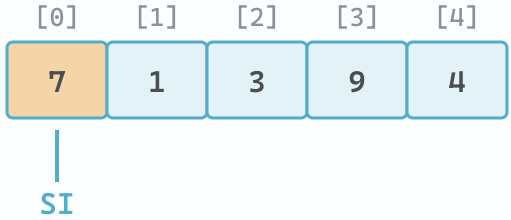
Adım 3 / 13

Kod İzleme: `LOOPNE` ile Dizide Arama

```
1  org 100h
2  mov cx, 5          ; eleman sayısı
3  mov si, -1         ; ilk inc ile 0 olur
4  ara:
5  inc si             ; sonraki eleman
▶  cmp dizi[si], 3    ; aranan değer 3 mü?
7  loopne ara         ; CX≠0 ve ZF=0 ise sürdür
8  jne yok            ; ZF=0 → bulunamadı
9  mov al, 1          ; bulundu, konum SI'da
10 ret
11 yok:
12 mov al, 0          ; bulunamadı
13 ret
14 dizi db 7, 1, 3, 9, 4
```

DEĞİŞKENLER

`CX = 0005h` `SI = 0000h` `ZF = 0`



dizi[0] = 7; 7 − 3 → ZF = 0.

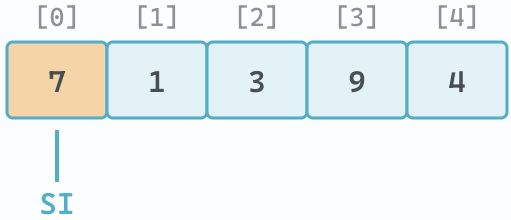
Adım 4 / 13

Kod İzleme: `LOOPNE` ile Dizide Arama

```
1  org 100h
2  mov cx, 5          ; eleman sayısı
3  mov si, -1         ; ilk inc ile 0 olur
4  ara:
5  inc si             ; sonraki eleman
6  cmp dizi[si], 3    ; aranan değer 3 mü?
▶  loopne ara         ; CX≠0 ve ZF=0 ise sürdür
8  jne yok            ; ZF=0 → bulunamadı
9  mov al, 1          ; bulundu, konum SI'da
10 ret
11 yok:
12 mov al, 0          ; bulunamadı
13 ret
14 dizi db 7, 1, 3, 9, 4
```

DEĞİŞKENLER

`CX = 0004h` `SI = 0000h` `ZF = 0`



`loopne` : CX = 4 ≠ 0 ve ZF = 0 → aramaya devam.

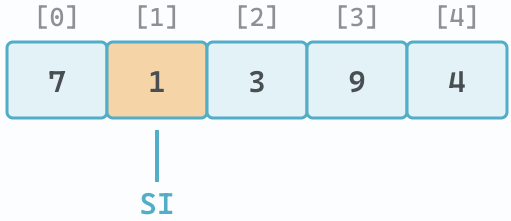
Adım 5 / 13

Kod İzleme: `LOOPNE` ile Dizide Arama

```
1  org 100h
2  mov cx, 5          ; eleman sayısı
3  mov si, -1         ; ilk inc ile 0 olur
4  ara:
▶  inc si             ; sonraki eleman
6  cmp dizi[si], 3    ; aranan değer 3 mü?
7  loopne ara         ; CX≠0 ve ZF=0 ise sürdür
8  jne yok            ; ZF=0 → bulunamadı
9  mov al, 1          ; bulundu, konum SI'da
10 ret
11 yok:
12 mov al, 0          ; bulunamadı
13 ret
14 dizi db 7, 1, 3, 9, 4
```

DEĞİŞKENLER

`CX = 0004h` `SI = 0001h` `ZF = 0`



SI = 1.

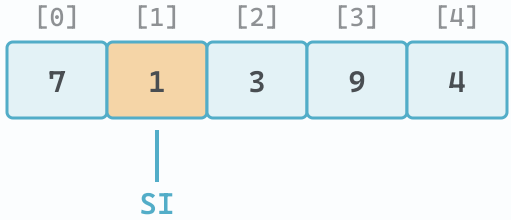
Adım 6 / 13

Kod İzleme: `LOOPNE` ile Dizide Arama

```
1  org 100h
2  mov cx, 5          ; eleman sayısı
3  mov si, -1         ; ilk inc ile 0 olur
4  ara:
5  inc si             ; sonraki eleman
▶  cmp dizi[si], 3    ; aranan değer 3 mü?
7  loopne ara         ; CX≠0 ve ZF=0 ise sürdür
8  jne yok            ; ZF=0 → bulunamadı
9  mov al, 1          ; bulundu, konum SI'da
10 ret
11 yok:
12 mov al, 0          ; bulunamadı
13 ret
14 dizi db 7, 1, 3, 9, 4
```

DEĞİŞKENLER

`CX = 0004h` `SI = 0001h` `ZF = 0`



dizi[1] = 1; 1 − 3 → ZF = 0.

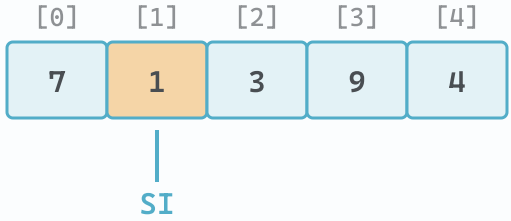
Adım 7 / 13

Kod İzleme: `LOOPNE` ile Dizide Arama

```
1  org 100h
2  mov cx, 5          ; eleman sayısı
3  mov si, -1         ; ilk inc ile 0 olur
4  ara:
5  inc si             ; sonraki eleman
6  cmp dizi[si], 3    ; aranan değer 3 mü?
▶  loopne ara         ; CX≠0 ve ZF=0 ise sürdür
8  jne yok            ; ZF=0 → bulunamadı
9  mov al, 1          ; bulundu, konum SI'da
10 ret
11 yok:
12 mov al, 0          ; bulunamadı
13 ret
14 dizi db 7, 1, 3, 9, 4
```

DEĞİŞKENLER

`CX = 0003h` `SI = 0001h` `ZF = 0`



`loopne` : CX = 3 ≠ 0 ve ZF = 0 → aramaya devam.

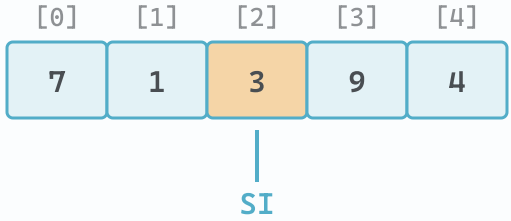
Adım 8 / 13

Kod İzleme: `LOOPNE` ile Dizide Arama

```
1  org 100h
2  mov cx, 5          ; eleman sayısı
3  mov si, -1         ; ilk inc ile 0 olur
4  ara:
5  inc si             ; sonraki eleman
6  cmp dizi[si], 3    ; aranan değer 3 mü?
7  loopne ara         ; CX≠0 ve ZF=0 ise sürdür
8  jne yok            ; ZF=0 → bulunamadı
9  mov al, 1          ; bulundu, konum SI'da
10 ret
11 yok:
12 mov al, 0          ; bulunamadı
13 ret
14 dizi db 7, 1, 3, 9, 4
```

DEĞİŞKENLER

CX = **0003h** | SI = **0002h** | ZF = **0**



SI = 2.

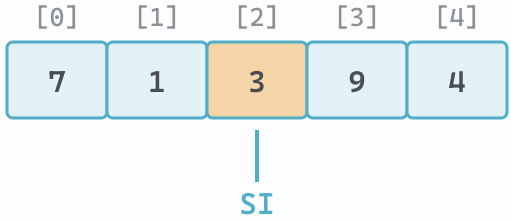
Adım 9 / 13

Kod İzleme: `LOOPNE` ile Dizide Arama

```
1  org 100h
2  mov cx, 5          ; eleman sayısı
3  mov si, -1         ; ilk inc ile 0 olur
4  ara:
5  inc si             ; sonraki eleman
▶  cmp dizi[si], 3    ; aranan değer 3 mü?
7  loopne ara         ; CX≠0 ve ZF=0 ise sürdür
8  jne yok            ; ZF=0 → bulunamadı
9  mov al, 1          ; bulundu, konum SI'da
10 ret
11 yok:
12 mov al, 0          ; bulunamadı
13 ret
14 dizi db 7, 1, 3, 9, 4
```

DEĞİŞKENLER

`CX = 0003h` `SI = 0002h` `ZF = 1`



dizi[2] = 3; 3 − 3 → ZF = 1.

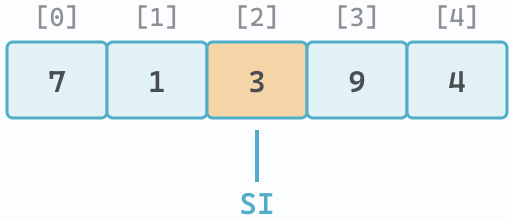
Adım 10 / 13

Kod İzleme: `LOOPNE` ile Dizide Arama

```
1  org 100h
2  mov cx, 5          ; eleman sayısı
3  mov si, -1         ; ilk inc ile 0 olur
4  ara:
5  inc si             ; sonraki eleman
6  cmp dizi[si], 3    ; aranan değer 3 mü?
▶  loopne ara         ; CX≠0 ve ZF=0 ise sürdür
8  jne yok            ; ZF=0 → bulunamadı
9  mov al, 1          ; bulundu, konum SI'da
10 ret
11 yok:
12 mov al, 0          ; bulunamadı
13 ret
14 dizi db 7, 1, 3, 9, 4
```

DEĞİŞKENLER

`CX = 0002h` `SI = 0002h` `ZF = 1`



`loopne` : CX = 2 ama **ZF = 1** → döngüden çıkılır. LOOPNE bayrakları değiştirmediği için ZF = 1 korunur.

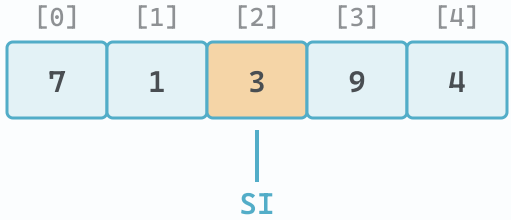
Adım 11 / 13

Kod İzleme: `LOOPNE` ile Dizide Arama

```
1  org 100h
2  mov cx, 5          ; eleman sayısı
3  mov si, -1         ; ilk inc ile 0 olur
4  ara:
5  inc si             ; sonraki eleman
6  cmp dizi[si], 3    ; aranan değer 3 mü?
7  loopne ara         ; CX≠0 ve ZF=0 ise sürdür
▶  jne yok            ; ZF=0 → bulunamadı
9  mov al, 1          ; bulundu, konum SI'da
10 ret
11 yok:
12 mov al, 0          ; bulunamadı
13 ret
14 dizi db 7, 1, 3, 9, 4
```

DEĞİŞKENLER

`CX = 0002h` `SI = 0002h` `ZF = 1`



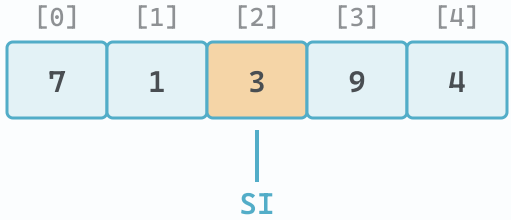
ZF = 1 → `jne yok` atlamaz: değer bulundu.

Kod İzleme: `LOOPNE` ile Dizide Arama

```
1  org 100h
2  mov cx, 5          ; eleman sayısı
3  mov si, -1         ; ilk inc ile 0 olur
4  ara:
5  inc si             ; sonraki eleman
6  cmp dizi[si], 3    ; aranan değer 3 mü?
7  loopne ara         ; CX≠0 ve ZF=0 ise sürdür
8  jne yok            ; ZF=0 → bulunamadı
▶  mov al, 1          ; bulundu, konum SI'da
▶  ret
11 yok:
12 mov al, 0          ; bulunamadı
13 ret
14 dizi db 7, 1, 3, 9, 4
```

DEĞİŞKENLER

`CX = 0002h` `SI = 0002h` `ZF = 1` `AL = 01h`



AL = 1 (bulundu), SI = 2 konumu gösterir. Döngü 5 yerine 3 turda bitti.

Adım 13 / 13

JCXZ ve LOOP / DEC–JNZ

Sıfır turlu döngüye karşı koruma

```
mov cx, n          ; n çalışma anında
jcxz atla          ; CX = 0 ise girme
tekrar:
    ; gövde
    loop tekrar
atla:
```

JCXZ olmadan CX = 0 iken LOOP önce CX'i FFFFh yapar ve gövde 65.536 kez çalışır.

LOOP mu, DEC + JNZ mi?

	LOOP	DEC CX + JNZ
Boyut	2 bayt	1 + 2 = 3 bayt
Bayraklar	değişmez	ZF, SF, OF, PF, AF değişir (CF korunur)
Sayaç	yalnız CX	herhangi bir yazmaç
Ek koşul	LOOPE / LOOPNE	farklı Jcc seçilebilir

Kod İzleme: JCXZ Koruması ve DEC–JNZ

```
1  MOV  CX, 0        ; n = 0
2  MOV  AX, 0
3  JCXZ atla         ; CX = 0 → girme
4  tekrar: INC AX
5  LOOP tekrar
6  atla: MOV BX, 3   ; n = 3, sayaç BX
7  STC               ; CF = 1
8  tekrar2: INC AX
9  DEC  BX           ; CF korunur
10 JNZ  tekrar2
11 RET
```

DEĞİŞKENLER

AX = 0 | BX = 0 | CX = 0 | CF = 0

AX	BX	CX
0000h	0000h	0000h

ZF	CF
0	0

Tur sayısı n = 0 olarak CX'te. Korumasız LOOP önce CX'i FFFFh yapar ve gövde 65.536 kez çalışırdı.

Adım 1 / 9

Kod İzleme: JCXZ Koruması ve DEC–JNZ

```
1  MOV  CX, 0        ; n = 0
2  MOV  AX, 0
▶  JCXZ atla         ; CX = 0 → girme
4  tekrar: INC AX
5  LOOP tekrar
6  atla: MOV BX, 3   ; n = 3, sayaç BX
7  STC               ; CF = 1
8  tekrar2: INC AX
9  DEC  BX           ; CF korunur
10 JNZ  tekrar2
11 RET
```

DEĞİŞKENLER

`AX = 0` `BX = 0` `CX = 0` `CF = 0`

AX	BX	CX
0000h	0000h	0000h

ZF	CF
0	0

JCXZ → atlar

`JCXZ atla`: CX = 0 → döngüye hiç girilmez. AX = 0 kaldı.

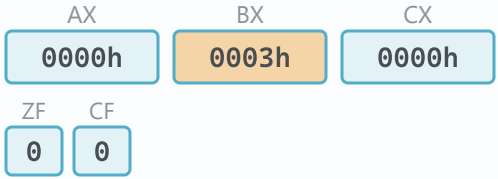
Adım 2 / 9

Kod İzleme: JCXZ Koruması ve DEC–JNZ

```
1  MOV  CX, 0        ; n = 0
2  MOV  AX, 0
3  JCXZ atla         ; CX = 0 → girme
4  tekrar: INC AX
5  LOOP tekrar
▶  atla: MOV BX, 3   ; n = 3, sayaç BX
7  STC               ; CF = 1
8  tekrar2: INC AX
9  DEC  BX           ; CF korunur
10 JNZ  tekrar2
11 RET
```

DEĞİŞKENLER

AX = 0 | BX = 3 | CX = 0 | CF = 0



İkinci döngü: sayaç BX = 3 (DEC–JNZ istenen yazmacı kullanabilir).

Adım 3 / 9

Kod İzleme: JCXZ Koruması ve DEC–JNZ

```
1  MOV  CX, 0        ; n = 0
2  MOV  AX, 0
3  JCXZ atla         ; CX = 0 → girme
4  tekrar: INC AX
5  LOOP tekrar
6  atla: MOV BX, 3   ; n = 3, sayaç BX
▶  STC               ; CF = 1
8  tekrar2: INC AX
9  DEC  BX           ; CF korunur
10 JNZ  tekrar2
11 RET
```

DEĞİŞKENLER

AX = 0 | BX = 3 | CX = 0 | CF = 1

AX	BX	CX
0000h	0003h	0000h

ZF	CF
0	1

`STC` : CF = 1. Döngü boyunca korunması gereken bir bilgi gibi düşünülebilir.

Adım 4 / 9

Kod İzleme: JCXZ Koruması ve DEC–JNZ

```
1  MOV  CX, 0        ; n = 0
2  MOV  AX, 0
3  JCXZ atla         ; CX = 0 → girme
4  tekrar: INC AX
5  LOOP tekrar
6  atla: MOV BX, 3   ; n = 3, sayaç BX
7  STC               ; CF = 1
▶  tekrar2: INC AX
▶  DEC  BX           ; CF korunur
▶  JNZ  tekrar2
11 RET
```

DEĞİŞKENLER

`AX = 1` `BX = 2` `CX = 0` `CF = 1`

AX	BX	CX
0001h	0002h	0000h

ZF	CF
0	1

JNZ → atlar

Tur 1: AX = 1. `DEC BX` → 2, ZF = 0 → `JNZ` geri atlar. CF hâlâ 1.

Adım 5 / 9

Kod İzleme: JCXZ Koruması ve DEC–JNZ

```
1  MOV  CX, 0        ; n = 0
2  MOV  AX, 0
3  JCXZ atla         ; CX = 0 → girme
4  tekrar: INC AX
5  LOOP tekrar
6  atla: MOV BX, 3   ; n = 3, sayaç BX
7  STC               ; CF = 1
▶  tekrar2: INC AX
▶  DEC  BX           ; CF korunur
▶  JNZ  tekrar2
11 RET
```

DEĞİŞKENLER

AX = 2 | BX = 1 | CX = 0 | CF = 1

AX	BX	CX
0002h	0001h	0000h

ZF	CF
0	1

JNZ → atlar

Tur 2: AX = 2, BX = 1.

Kod İzleme: JCXZ Koruması ve DEC–JNZ

```
1  MOV  CX, 0        ; n = 0
2  MOV  AX, 0
3  JCXZ atla         ; CX = 0 → girme
4  tekrar: INC AX
5  LOOP tekrar
6  atla: MOV BX, 3   ; n = 3, sayaç BX
7  STC               ; CF = 1
▶  tekrar2: INC AX
▶  DEC  BX           ; CF korunur
10 JNZ  tekrar2
11 RET
```

DEĞİŞKENLER

AX = 3 | BX = 0 | CX = 0 | CF = 1

AX	BX	CX
0003h	0000h	0000h

ZF	CF
1	1

Tur 3: AX = 3. `DEC BX` → 0, ZF = 1. INC/DEC CF'ye dokunmadı: CF = 1.

Kod İzleme: JCXZ Koruması ve DEC–JNZ

```
1  MOV  CX, 0        ; n = 0
2  MOV  AX, 0
3  JCXZ atla         ; CX = 0 → girme
4  tekrar: INC AX
5  LOOP tekrar
6  atla: MOV BX, 3   ; n = 3, sayaç BX
7  STC               ; CF = 1
8  tekrar2: INC AX
9  DEC  BX           ; CF korunur
▶  JNZ  tekrar2
11 RET
```

DEĞİŞKENLER

`AX = 3` `BX = 0` `CX = 0` `CF = 1`

AX	BX	CX
0003h	0000h	0000h

ZF	CF
1	1

JNZ → atlamaz

`JNZ` : ZF = 1 → atlamaz, döngü biter.

Adım 8 / 9

Kod İzleme: JCXZ Koruması ve DEC–JNZ

```
1  MOV  CX, 0        ; n = 0
2  MOV  AX, 0
3  JCXZ atla         ; CX = 0 → girme
4  tekrar: INC AX
5  LOOP tekrar
6  atla: MOV BX, 3   ; n = 3, sayaç BX
7  STC               ; CF = 1
8  tekrar2: INC AX
9  DEC  BX           ; CF korunur
10 JNZ  tekrar2
▶  RET
```

DEĞİŞKENLER

AX = 3 | BX = 0 | CX = 0 | CF = 1

AX	BX	CX
0003h	0000h	0000h

ZF	CF
1	1

Sonuç AX = 3, CF = 1 korundu. LOOP da bayrak değiştirmez ama yalnız CX ile çalışır.

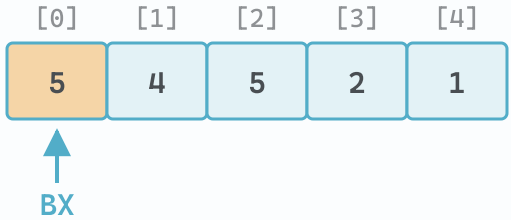
Adım 9 / 9

Kod İzleme: Dizi Elemanları Toplamı

```
1  org 100h
▶ mov cx, 5            ; eleman sayısı
▶ mov al, 0            ; toplam (AL) = 0
▶ mov bx, 0            ; BX = indis
5  next:
6  add al, vector[bx]  ; elemanı topla
7  inc bx              ; sonraki eleman
8  loop next           ; CX = 0 olana kadar dön
9  mov m, al           ; sonucu m'ye yaz
10 ret
11 vector db 5, 4, 5, 2, 1
12 m db 0
```

DEĞİŞKENLER

CX = 0005h BX = 0000h AL = 00h m = 00h



CX = 5 eleman, toplam AL = 0, indis BX = 0.

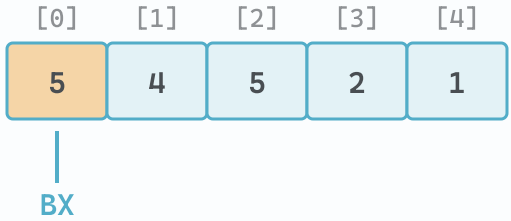
Adım 1 / 13

Kod İzleme: Dizi Elemanları Toplamı

```
1  org 100h
2  mov cx, 5            ; eleman sayısı
3  mov al, 0            ; toplam (AL) = 0
4  mov bx, 0            ; BX = indis
5  next:
▶  add al, vector[bx]   ; elemanı topla
7  inc bx               ; sonraki eleman
8  loop next            ; CX = 0 olana kadar dön
9  mov m, al            ; sonucu m'ye yaz
10 ret
11 vector db 5, 4, 5, 2, 1
12 m db 0
```

DEĞİŞKENLER

`CX = 0005h` `BX = 0000h` `AL = 05h` `m = 00h`



`add al, vector[0]` : AL = 0 + 5 = 5 (05h).

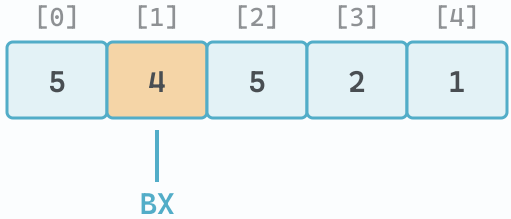
Adım 2 / 13

Kod İzleme: Dizi Elemanları Toplamı

```
1  org 100h
2  mov cx, 5            ; eleman sayısı
3  mov al, 0            ; toplam (AL) = 0
4  mov bx, 0            ; BX = indis
5  next:
6  add al, vector[bx]   ; elemanı topla
▶  inc bx               ; sonraki eleman
▶  loop next            ; CX = 0 olana kadar dön
9  mov m, al            ; sonucu m'ye yaz
10 ret
11 vector db 5, 4, 5, 2, 1
12 m db 0
```

DEĞİŞKENLER

CX = 0004h | BX = 0001h | AL = 05h | m = 00h



BX = 1, CX = 4 → next'e dön.

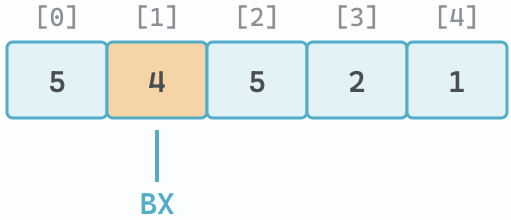
Adım 3 / 13

Kod İzleme: Dizi Elemanları Toplamı

```
1  org 100h
2  mov cx, 5            ; eleman sayısı
3  mov al, 0            ; toplam (AL) = 0
4  mov bx, 0            ; BX = indis
5  next:
▶  add al, vector[bx]   ; elemanı topla
7  inc bx               ; sonraki eleman
8  loop next            ; CX = 0 olana kadar dön
9  mov m, al            ; sonucu m'ye yaz
10 ret
11 vector db 5, 4, 5, 2, 1
12 m db 0
```

DEĞİŞKENLER

`CX = 0004h` `BX = 0001h` `AL = 09h` `m = 00h`



`add al, vector[1]` : AL = 5 + 4 = 9 (09h).

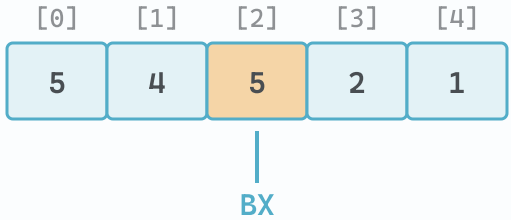
Adım 4 / 13

Kod İzleme: Dizi Elemanları Toplamı

```
1  org 100h
2  mov cx, 5            ; eleman sayısı
3  mov al, 0            ; toplam (AL) = 0
4  mov bx, 0            ; BX = indis
5  next:
6  add al, vector[bx]   ; elemanı topla
▶  inc bx               ; sonraki eleman
▶  loop next            ; CX = 0 olana kadar dön
9  mov m, al            ; sonucu m'ye yaz
10 ret
11 vector db 5, 4, 5, 2, 1
12 m db 0
```

DEĞİŞKENLER

CX = **0003h** BX = **0002h** AL = **09h** m = **00h**



BX = 2, CX = 3 → next'e dön.

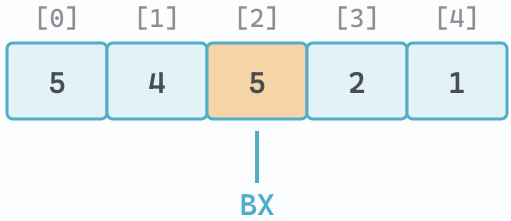
Adım 5 / 13

Kod İzleme: Dizi Elemanları Toplamı

```
1  org 100h
2  mov cx, 5            ; eleman sayısı
3  mov al, 0            ; toplam (AL) = 0
4  mov bx, 0            ; BX = indis
5  next:
▶  add al, vector[bx]   ; elemanı topla
7  inc bx               ; sonraki eleman
8  loop next            ; CX = 0 olana kadar dön
9  mov m, al            ; sonucu m'ye yaz
10 ret
11 vector db 5, 4, 5, 2, 1
12 m db 0
```

DEĞİŞKENLER

`CX = 0003h` `BX = 0002h` `AL = 0Eh` `m = 00h`



`add al, vector[2]` : AL = 9 + 5 = 14 (0Eh).

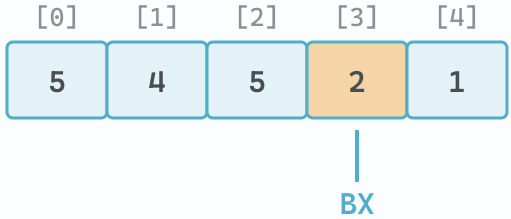
Adım 6 / 13

Kod İzleme: Dizi Elemanları Toplamı

```
1  org 100h
2  mov cx, 5            ; eleman sayısı
3  mov al, 0            ; toplam (AL) = 0
4  mov bx, 0            ; BX = indis
5  next:
6  add al, vector[bx]   ; elemanı topla
▶  inc bx               ; sonraki eleman
▶  loop next            ; CX = 0 olana kadar dön
9  mov m, al            ; sonucu m'ye yaz
10 ret
11 vector db 5, 4, 5, 2, 1
12 m db 0
```

DEĞİŞKENLER

CX = 0002h | BX = 0003h | AL = 0Eh | m = 00h



BX = 3, CX = 2 → next'e dön.

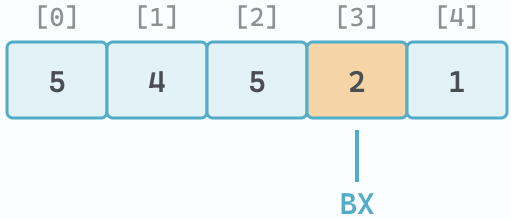
Adım 7 / 13

Kod İzleme: Dizi Elemanları Toplamı

```
1  org 100h
2  mov cx, 5            ; eleman sayısı
3  mov al, 0            ; toplam (AL) = 0
4  mov bx, 0            ; BX = indis
5  next:
▶  add al, vector[bx]   ; elemanı topla
7  inc bx               ; sonraki eleman
8  loop next            ; CX = 0 olana kadar dön
9  mov m, al            ; sonucu m'ye yaz
10 ret
11 vector db 5, 4, 5, 2, 1
12 m db 0
```

DEĞİŞKENLER

`CX = 0002h` `BX = 0003h` `AL = 10h` `m = 00h`



`add al, vector[3]` : AL = 14 + 2 = 16 (10h).

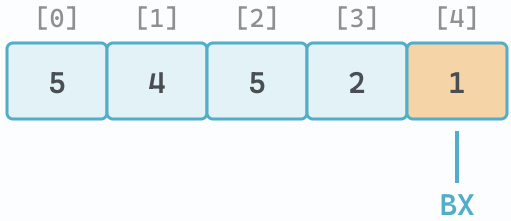
Adım 8 / 13

Kod İzleme: Dizi Elemanları Toplamı

```
1  org 100h
2  mov cx, 5            ; eleman sayısı
3  mov al, 0            ; toplam (AL) = 0
4  mov bx, 0            ; BX = indis
5  next:
6  add al, vector[bx]   ; elemanı topla
▶  inc bx               ; sonraki eleman
▶  loop next            ; CX = 0 olana kadar dön
9  mov m, al            ; sonucu m'ye yaz
10 ret
11 vector db 5, 4, 5, 2, 1
12 m db 0
```

DEĞİŞKENLER

CX = 0001h BX = 0004h AL = 10h m = 00h



BX = 4, CX = 1 → next'e dön.

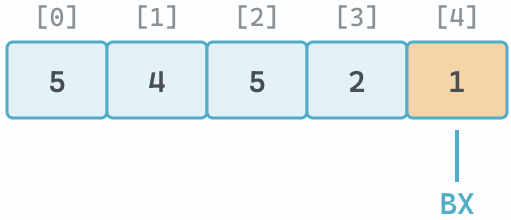
Adım 9 / 13

Kod İzleme: Dizi Elemanları Toplamı

```
1  org 100h
2  mov cx, 5            ; eleman sayısı
3  mov al, 0            ; toplam (AL) = 0
4  mov bx, 0            ; BX = indis
5  next:
▶  add al, vector[bx]   ; elemanı topla
7  inc bx               ; sonraki eleman
8  loop next            ; CX = 0 olana kadar dön
9  mov m, al            ; sonucu m'ye yaz
10 ret
11 vector db 5, 4, 5, 2, 1
12 m db 0
```

DEĞİŞKENLER

`CX = 0001h` `BX = 0004h` `AL = 11h` `m = 00h`



`add al, vector[4]` : AL = 16 + 1 = 17 (11h).

Adım 10 / 13

Kod İzleme: Dizi Elemanları Toplamı

```
1  org 100h
2  mov cx, 5            ; eleman sayısı
3  mov al, 0            ; toplam (AL) = 0
4  mov bx, 0            ; BX = indis
5  next:
6  add al, vector[bx]   ; elemanı topla
▶  inc bx               ; sonraki eleman
▶  loop next            ; CX = 0 olana kadar dön
9  mov m, al            ; sonucu m'ye yaz
10 ret
11 vector db 5, 4, 5, 2, 1
12 m db 0
```

DEĞİŞKENLER

CX = **0000h** BX = **0005h** AL = **11h** m = **00h**

[0]	[1]	[2]	[3]	[4]
5	4	5	2	1

BX = 5, CX = 0 → döngü biter.

Adım 11 / 13

Kod İzleme: Dizi Elemanları Toplamı

```
1  org 100h
2  mov cx, 5            ; eleman sayısı
3  mov al, 0            ; toplam (AL) = 0
4  mov bx, 0            ; BX = indis
5  next:
6  add al, vector[bx]   ; elemanı topla
7  inc bx               ; sonraki eleman
8  loop next            ; CX = 0 olana kadar dön
▶  mov m, al            ; sonucu m'ye yaz
10 ret
11 vector db 5, 4, 5, 2, 1
12 m db 0
```

DEĞİŞKENLER

`CX = 0000h` `BX = 0005h` `AL = 11h` `m = 11h`



`mov m, al` : m = 11h = **17** = 5 + 4 + 5 + 2 + 1.

Adım 12 / 13

Kod İzleme: Dizi Elemanları Toplamı

```
1  org 100h
2  mov cx, 5            ; eleman sayısı
3  mov al, 0            ; toplam (AL) = 0
4  mov bx, 0            ; BX = indis
5  next:
6  add al, vector[bx]   ; elemanı topla
7  inc bx               ; sonraki eleman
8  loop next            ; CX = 0 olana kadar dön
9  mov m, al            ; sonucu m'ye yaz
▶  ret
11 vector db 5, 4, 5, 2, 1
12 m db 0
```

DEĞİŞKENLER

`CX = 0000h` `BX = 0005h` `AL = 11h` `m = 11h`



`ret`. Veriler `ret`'ten sonra tanımlandığı için komut olarak yürütülmez.

Adım 13 / 13

Kod İzleme: Ekrana Hello Yazdırma

```
1  org     100h
▶ jmp     start
3  msg db 'Hello', 0
4  start:
5  mov     si, 0           ; kaynak indisi = 0
6  next_char:
7  mov     al, msg[si]     ; karakteri al
8  cmp     al, 0           ; 0 ile karşılaştır
9  je      stop            ; 0 ise yazmayı durdur
10 mov     ah, 0eh         ; teletype işlevi
11 int     10h             ; AL'deki karakteri
   yaz
12 inc     si              ; sonraki karakter
13 jmp     next_char       ; başa dön
14 stop:
15 ret
```

DEĞİŞKENLER

—

ÇIKTI

48h	65h	6Ch	6Ch	6Fh	00h
'H'	'e'	'l'	'l'	'o'	0

`jmp start` : veri (msg) kodun arasında olduğu için üzerinden atlanır; yoksa 'H' (48h) gibi baytlar komut sanılırdı.

Adım 1 / 13

Kod İzleme: Ekrana Hello Yazdırma

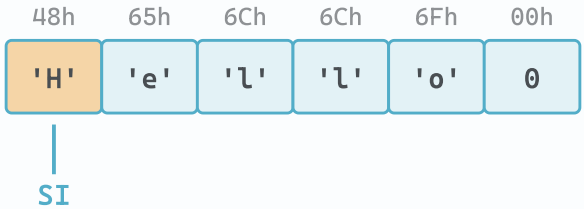
```
1  org     100h
2  jmp     start
3  msg db 'Hello', 0
4  start:
▶  mov     si, 0           ; kaynak indisi = 0
6  next_char:
7  mov     al, msg[si]     ; karakteri al
8  cmp     al, 0           ; 0 ile karşılaştır
9  je      stop            ; 0 ise yazmayı durdur
10 mov     ah, 0eh         ; teletype işlevi
11 int     10h             ; AL'deki karakteri
   yaz
12 inc     si              ; sonraki karakter
13 jmp     next_char       ; başa dön
14 stop:
15 ret
```

DEĞİŞKENLER

SI = 0000h

ÇIKTI





SI = 0: ilk karakter.

Adım 2 / 13

Kod İzleme: Ekrana Hello Yazdırma

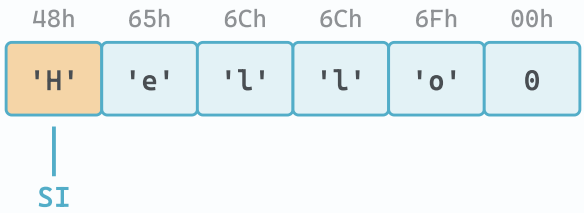
```
1  org     100h
2  jmp     start
3  msg db 'Hello', 0
4  start:
5  mov     si, 0           ; kaynak indisi = 0
6  next_char:
▶  mov     al, msg[si]     ; karakteri al
8  cmp     al, 0           ; 0 ile karşılaştır
9  je      stop            ; 0 ise yazmayı durdur
10 mov     ah, 0eh         ; teletype işlevi
11 int     10h             ; AL'deki karakteri
   yaz
12 inc     si              ; sonraki karakter
13 jmp     next_char       ; başa dön
14 stop:
15 ret
```

DEĞİŞKENLER

SI = 0000h AL = 48h

ÇIKTI





AL = msg[0] = 'H' = 48h.

Adım 3 / 13

Kod İzleme: Ekrana Hello Yazdırma

```
1  org     100h
2  jmp     start
3  msg db 'Hello', 0
4  start:
5  mov     si, 0           ; kaynak indisi = 0
6  next_char:
7  mov     al, msg[si]     ; karakteri al
▶  cmp     al, 0           ; 0 ile karşılaştır
▶  je      stop            ; 0 ise yazmayı durdur
10 mov     ah, 0eh         ; teletype işlevi
11 int     10h             ; AL'deki karakteri
   yaz
12 inc     si              ; sonraki karakter
13 jmp     next_char       ; başa dön
14 stop:
15 ret
```

DEĞİŞKENLER

SI = **0000h** | AL = **48h** | ZF = **0**

ÇIKTI

48h	65h	6Ch	6Ch	6Fh	00h
'H'	'e'	'l'	'l'	'o'	0

SI

48h − 0 ≠ 0 → ZF = 0, `je` atlamaz.

Adım 4 / 13

Kod İzleme: Ekrana Hello Yazdırma

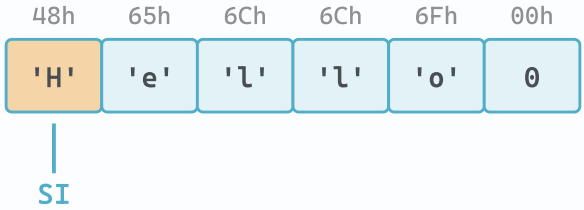
```
1  org     100h
2  jmp     start
3  msg db 'Hello', 0
4  start:
5  mov     si, 0           ; kaynak indisi = 0
6  next_char:
7  mov     al, msg[si]     ; karakteri al
8  cmp     al, 0           ; 0 ile karşılaştır
9  je      stop            ; 0 ise yazmayı durdur
▶  mov     ah, 0eh         ; teletype işlevi
▶  int     10h             ; AL'deki karakteri
   yaz
12 inc     si              ; sonraki karakter
13 jmp     next_char       ; başa dön
14 stop:
15 ret
```

DEĞİŞKENLER

SI = **0000h** | AL = **48h** | AH = **0Eh** | ZF = **0**

ÇIKTI

```
H
```



AH = 0Eh, INT 10h → **H** yazılır.

Adım 5 / 13

Kod İzleme: Ekrana Hello Yazdırma

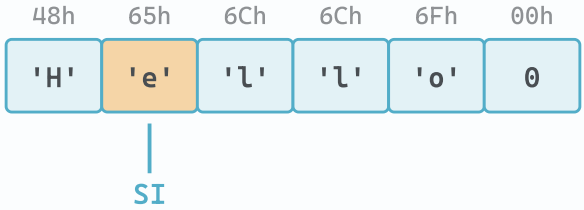
```
1  org     100h
2  jmp     start
3  msg db 'Hello', 0
4  start:
5  mov     si, 0           ; kaynak indisi = 0
6  next_char:
7  mov     al, msg[si]     ; karakteri al
8  cmp     al, 0           ; 0 ile karşılaştır
9  je      stop            ; 0 ise yazmayı durdur
10 mov     ah, 0eh         ; teletype işlevi
11 int     10h             ; AL'deki karakteri
   yaz
▶ inc     si              ; sonraki karakter
▶ jmp     next_char       ; başa dön
14 stop:
15 ret
```

DEĞİŞKENLER

SI = 0001h | AL = 48h | AH = 0Eh | ZF = 0

ÇIKTI

```
H
```



SI = 1, `jmp next_char` ile başa dönülür.

Adım 6 / 13

Kod İzleme: Ekrana Hello Yazdırma

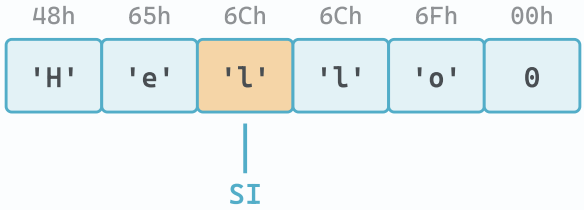
```
1  org     100h
2  jmp     start
3  msg db 'Hello', 0
4  start:
5  mov     si, 0           ; kaynak indisi = 0
6  next_char:
▶  mov     al, msg[si]     ; karakteri al
▶  cmp     al, 0           ; 0 ile karşılaştır
▶  je      stop            ; 0 ise yazmayı durdur
▶  mov     ah, 0eh         ; teletype işlevi
▶  int     10h             ; AL'deki karakteri
   yaz
▶  inc     si              ; sonraki karakter
▶  jmp     next_char       ; başa dön
14 stop:
15 ret
```

DEĞİŞKENLER

SI = **0002h** AL = **65h** AH = **0Eh** ZF = **0**

ÇIKTI

```
He
```



Aynı tur: AL = 'e' (65h), ZF = 0 → **e** yazılır, SI = 2.

Kod İzleme: Ekrana Hello Yazdırma

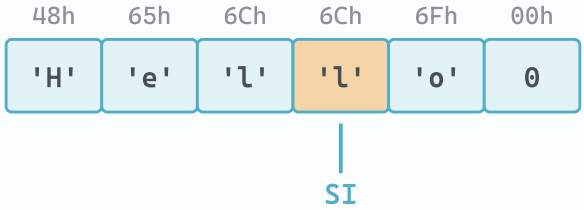
```
1  org     100h
2  jmp     start
3  msg db 'Hello', 0
4  start:
5  mov     si, 0           ; kaynak indisi = 0
6  next_char:
▶  mov     al, msg[si]     ; karakteri al
▶  cmp     al, 0           ; 0 ile karşılaştır
▶  je      stop            ; 0 ise yazmayı durdur
▶  mov     ah, 0eh         ; teletype işlevi
▶  int     10h             ; AL'deki karakteri
   yaz
▶  inc     si              ; sonraki karakter
▶  jmp     next_char       ; başa dön
14 stop:
15 ret
```

DEĞİŞKENLER

SI = 0003h | AL = 6Ch | AH = 0Eh | ZF = 0

ÇIKTI

```
Hel
```



Aynı tur: AL = 'l' (6Ch), ZF = 0 → **l** yazılır, SI = 3.

Kod İzleme: Ekrana Hello Yazdırma

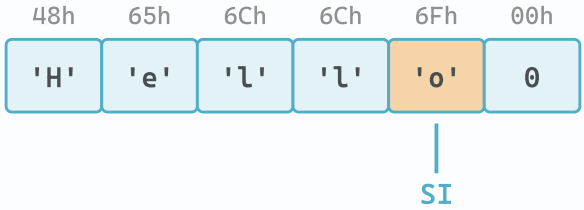
```
1  org     100h
2  jmp     start
3  msg db 'Hello', 0
4  start:
5  mov     si, 0           ; kaynak indisi = 0
6  next_char:
▶  mov     al, msg[si]     ; karakteri al
▶  cmp     al, 0           ; 0 ile karşılaştır
▶  je      stop            ; 0 ise yazmayı durdur
▶  mov     ah, 0eh         ; teletype işlevi
▶  int     10h             ; AL'deki karakteri
   yaz
▶  inc     si              ; sonraki karakter
▶  jmp     next_char       ; başa dön
14 stop:
15 ret
```

DEĞİŞKENLER

SI = 0004h | AL = 6Ch | AH = 0Eh | ZF = 0

ÇIKTI

```
Hell
```



Aynı tur: AL = 'l' (6Ch), ZF = 0 → **l** yazılır, SI = 4.

Kod İzleme: Ekrana Hello Yazdırma

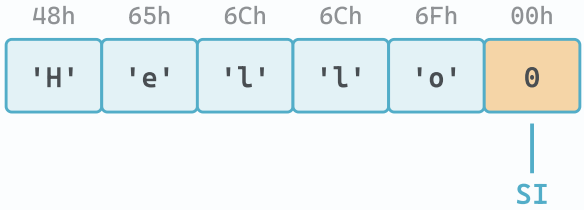
```
1  org     100h
2  jmp     start
3  msg db 'Hello', 0
4  start:
5  mov     si, 0           ; kaynak indisi = 0
6  next_char:
▶  mov     al, msg[si]     ; karakteri al
▶  cmp     al, 0           ; 0 ile karşılaştır
▶  je      stop            ; 0 ise yazmayı durdur
▶  mov     ah, 0eh         ; teletype işlevi
▶  int     10h             ; AL'deki karakteri
   yaz
▶  inc     si              ; sonraki karakter
▶  jmp     next_char       ; başa dön
14 stop:
15 ret
```

DEĞİŞKENLER

SI = **0005h** AL = **6Fh** AH = **0Eh** ZF = **0**

ÇIKTI

```
Hello
```



Aynı tur: AL = 'o' (6Fh), ZF = 0 → **o** yazılır, SI = 5.

Kod İzleme: Ekrana Hello Yazdırma

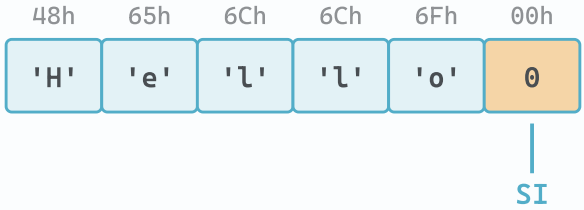
```
1  org     100h
2  jmp     start
3  msg db 'Hello', 0
4  start:
5  mov     si, 0           ; kaynak indisi = 0
6  next_char:
▶  mov     al, msg[si]     ; karakteri al
8  cmp     al, 0           ; 0 ile karşılaştır
9  je      stop            ; 0 ise yazmayı durdur
10 mov     ah, 0eh         ; teletype işlevi
11 int     10h             ; AL'deki karakteri
   yaz
12 inc     si              ; sonraki karakter
13 jmp     next_char       ; başa dön
14 stop:
15 ret
```

DEĞİŞKENLER

SI = **0005h** | AL = **00h** | AH = **0Eh** | ZF = **0**

ÇIKTI

```
Hello
```



AL = msg[5] = 0: dizginin sonu (null sonlandırıcı).

Adım 11 / 13

Kod İzleme: Ekrana Hello Yazdırma

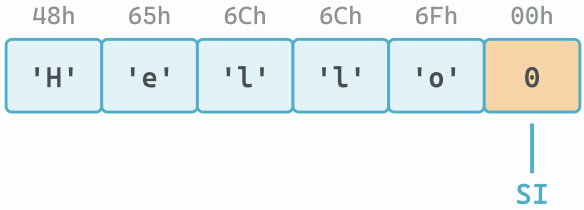
```
1  org     100h
2  jmp     start
3  msg db 'Hello', 0
4  start:
5  mov     si, 0           ; kaynak indisi = 0
6  next_char:
7  mov     al, msg[si]     ; karakteri al
▶  cmp     al, 0           ; 0 ile karşılaştır
▶  je      stop            ; 0 ise yazmayı durdur
10 mov     ah, 0eh         ; teletype işlevi
11 int     10h             ; AL'deki karakteri
   yaz
12 inc     si              ; sonraki karakter
13 jmp     next_char       ; başa dön
14 stop:
15 ret
```

DEĞİŞKENLER

`SI = 0005h` `AL = 00h` `AH = 0Eh` `ZF = 1`

ÇIKTI

```
Hello
```



0 − 0 = 0 → ZF = 1, `je stop` atlar.

Adım 12 / 13

Kod İzleme: Ekrana Hello Yazdırma

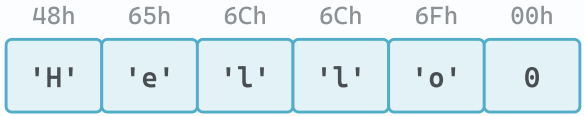
```
1  org     100h
2  jmp     start
3  msg db 'Hello', 0
4  start:
5  mov     si, 0           ; kaynak indisi = 0
6  next_char:
7  mov     al, msg[si]     ; karakteri al
8  cmp     al, 0           ; 0 ile karşılaştır
9  je      stop            ; 0 ise yazmayı durdur
10 mov     ah, 0eh         ; teletype işlevi
11 int     10h             ; AL'deki karakteri
   yaz
12 inc     si              ; sonraki karakter
13 jmp     next_char       ; başa dön
14 stop:
▶  ret
```

DEĞİŞKENLER

SI = **0005h** | AL = **00h** | AH = **0Eh** | ZF = **1**

ÇIKTI

```
Hello
```



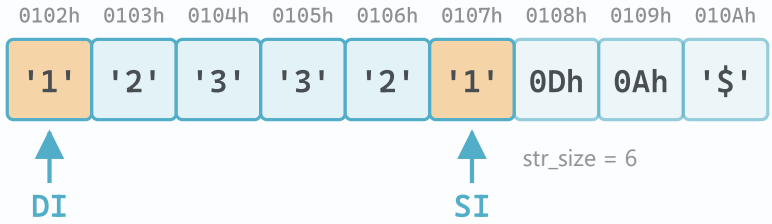
`ret` : ekranda **Hello**.

Adım 13 / 13

ÖRNEK PROGRAMLAR

Palindrom Kontrolü: Fikir

- Palindrom: baştan ve sondan okunuşu aynı olan dizi (`123321` , `kayak`).
- **DI** dizinin başını, **SI** sonunu gösterir; karakterler karşılaştırılır, DI ileri, SI geri gider.
- Uzunluğun **yarısı** kadar karşılaştırma yeterlidir (`shr cx, 1` → CX / 2). Tek uzunlukta ortadaki karakter karşılaştırılmaz.
- `str_size = $ - msg` : `$` o anki adrestir; `msg` ile `str` aynı adreste olduğundan sonuç dizinin uzunluğu olan **6** sabitidir (bellekte yer kaplamaz).
- INT 21h / AH = 09h: DS:DX adresindeki dizgiyi `'$'` karakterine kadar yazar; 0Dh, 0Ah satır başı ve yeni satırdır.



Adresler `org 100h` ve 2 baytlık kısa `jmp start` (0100h) varsayımıyla verilmiştir: str 0102h'den başlar.

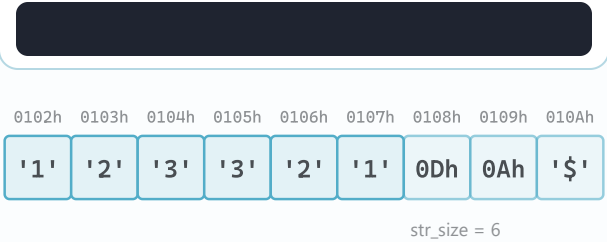
Kod İzleme: Palindrom (1/2) Hazırlık

```
1  org 100h
▶  jmp start
3  msg:
4  str db '123321'
5  str_size = $ - msg      ; = 6
6  db 0Dh,0Ah,'$'
7  start:
8  mov ah, 9               ; '$'e kadar yaz
9  mov dx, offset str
10 int 21h
11 lea di, str             ; DI = str adresi
12 mov si, di
13 add si, str_size        ; SI = son + 1
14 dec si                  ; SI = son karakter
15 mov cx, str_size
16 cmp cx, 1               ; tek karakter mi?
17 je is_palindrome        ; tek karakter
   palindromdur
18 shr cx, 1               ; CX = uzunluk / 2
```

DEĞİŞKENLER

—

ÇIKTI



`jmp start` : veri bölümünün üzerinden atlanır.

Adım 1 / 11

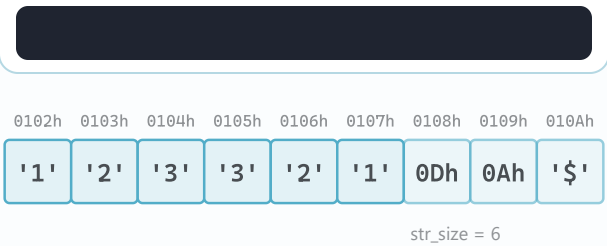
Kod İzleme: Palindrom (1/2) Hazırlık

```
1  org 100h
2  jmp start
3  msg:
4  str db '123321'
5  str_size = $ - msg      ; = 6
6  db 0Dh,0Ah,'$'
7  start:
▶  mov ah, 9               ; '$'e kadar yaz
9  mov dx, offset str
10 int 21h
11 lea di, str             ; DI = str adresi
12 mov si, di
13 add si, str_size        ; SI = son + 1
14 dec si                  ; SI = son karakter
15 mov cx, str_size
16 cmp cx, 1               ; tek karakter mi?
17 je is_palindrome        ; tek karakter
   palindromdur
18 shr cx, 1               ; CX = uzunluk / 2
```

DEĞİŞKENLER

AH = 09h

ÇIKTI



AH = 09h: DOS dizgi yazdırma işlevi.

Adım 2 / 11

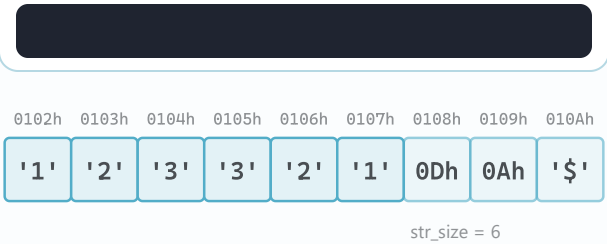
Kod İzleme: Palindrom (1/2) Hazırlık

```
1  org 100h
2  jmp start
3  msg:
4  str db '123321'
5  str_size = $ - msg      ; = 6
6  db 0Dh,0Ah,'$'
7  start:
8  mov ah, 9               ; '$'e kadar yaz
▶  mov dx, offset str
10 int 21h
11 lea di, str             ; DI = str adresi
12 mov si, di
13 add si, str_size        ; SI = son + 1
14 dec si                  ; SI = son karakter
15 mov cx, str_size
16 cmp cx, 1               ; tek karakter mi?
17 je is_palindrome        ; tek karakter
   palindromdur
18 shr cx, 1               ; CX = uzunluk / 2
```

DEĞİŞKENLER

AH = 09h DX = 0102h

ÇIKTI



DX = offset str = 0102h.

Adım 3 / 11

Kod İzleme: Palindrom (1/2) Hazırlık

```
1  org 100h
2  jmp start
3  msg:
4  str db '123321'
5  str_size = $ - msg      ; = 6
6  db 0Dh,0Ah,'$'
7  start:
8  mov ah, 9               ; '$'e kadar yaz
9  mov dx, offset str
▶  int 21h
11 lea di, str             ; DI = str adresi
12 mov si, di
13 add si, str_size        ; SI = son + 1
14 dec si                  ; SI = son karakter
15 mov cx, str_size
16 cmp cx, 1               ; tek karakter mi?
17 je is_palindrome        ; tek karakter
   palindromdur
18 shr cx, 1               ; CX = uzunluk / 2
```

DEĞİŞKENLER

`AH = 09h` `DX = 0102h`

ÇIKTI

```
123321
```

0102h	0103h	0104h	0105h	0106h	0107h	0108h	0109h	010Ah
'1'	'2'	'3'	'3'	'2'	'1'	0Dh	0Ah	'$'

str_size = 6

INT 21h: **123321** ve satır sonu yazılır ('$' yazılmaz).

Adım 4 / 11

Kod İzleme: Palindrom (1/2) Hazırlık

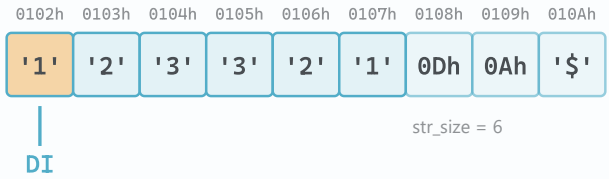
```
1  org 100h
2  jmp start
3  msg:
4  str db '123321'
5  str_size = $ - msg      ; = 6
6  db 0Dh,0Ah,'$'
7  start:
8  mov ah, 9               ; '$'e kadar yaz
9  mov dx, offset str
10 int 21h
▶ lea di, str              ; DI = str adresi
12 mov si, di
13 add si, str_size        ; SI = son + 1
14 dec si                  ; SI = son karakter
15 mov cx, str_size
16 cmp cx, 1               ; tek karakter mi?
17 je is_palindrome        ; tek karakter
   palindromdur
18 shr cx, 1               ; CX = uzunluk / 2
```

DEĞİŞKENLER

AH = **09h** | DX = **0102h** | DI = **0102h**

ÇIKTI

```
123321
```



`lea di, str` : DI = 0102h (ilk karakter).

Adım 5 / 11

Kod İzleme: Palindrom (1/2) Hazırlık

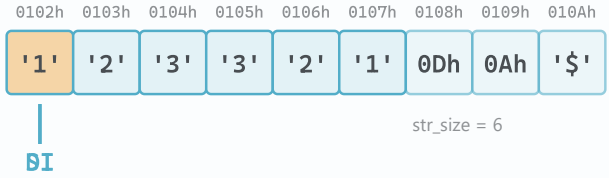
```
1  org 100h
2  jmp start
3  msg:
4  str db '123321'
5  str_size = $ - msg      ; = 6
6  db 0Dh,0Ah,'$'
7  start:
8  mov ah, 9               ; '$'e kadar yaz
9  mov dx, offset str
10 int 21h
11 lea di, str             ; DI = str adresi
▶  mov si, di
13 add si, str_size        ; SI = son + 1
14 dec si                  ; SI = son karakter
15 mov cx, str_size
16 cmp cx, 1               ; tek karakter mi?
17 je is_palindrome        ; tek karakter
   palindromdur
18 shr cx, 1               ; CX = uzunluk / 2
```

DEĞİŞKENLER

AH = **09h** DX = **0102h** DI = **0102h** SI = **0102h**

ÇIKTI

```
123321
```



SI = DI = 0102h.

Adım 6 / 11

Kod İzleme: Palindrom (1/2) Hazırlık

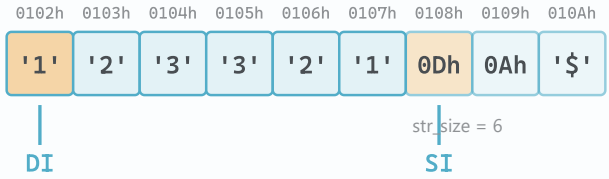
```
1  org 100h
2  jmp start
3  msg:
4  str db '123321'
5  str_size = $ - msg      ; = 6
6  db 0Dh,0Ah,'$'
7  start:
8  mov ah, 9               ; '$'e kadar yaz
9  mov dx, offset str
10 int 21h
11 lea di, str             ; DI = str adresi
12 mov si, di
▶ add si, str_size         ; SI = son + 1
14 dec si                  ; SI = son karakter
15 mov cx, str_size
16 cmp cx, 1               ; tek karakter mi?
17 je is_palindrome        ; tek karakter
   palindromdur
18 shr cx, 1               ; CX = uzunluk / 2
```

DEĞİŞKENLER

AH = **09h** DX = **0102h** DI = **0102h** SI = **0108h**

ÇIKTI

```
123321
```



SI = 0102h + 6 = 0108h: dizinin bir sonrası (0Dh).

Adım 7 / 11

Kod İzleme: Palindrom (1/2) Hazırlık

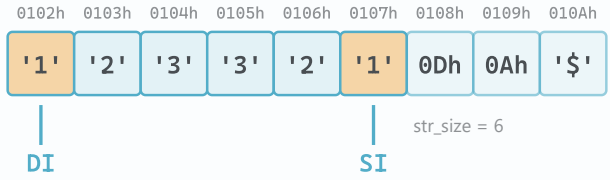
```
1  org 100h
2  jmp start
3  msg:
4  str db '123321'
5  str_size = $ - msg      ; = 6
6  db 0Dh,0Ah,'$'
7  start:
8  mov ah, 9               ; '$'e kadar yaz
9  mov dx, offset str
10 int 21h
11 lea di, str             ; DI = str adresi
12 mov si, di
13 add si, str_size        ; SI = son + 1
▶ dec si                   ; SI = son karakter
15 mov cx, str_size
16 cmp cx, 1               ; tek karakter mi?
17 je is_palindrome        ; tek karakter
   palindromdur
18 shr cx, 1               ; CX = uzunluk / 2
```

DEĞİŞKENLER

AH = **09h** DX = **0102h** DI = **0102h** SI = **0107h**

ÇIKTI

```
123321
```



`dec si` : SI = 0107h, son karakter '1'.

Adım 8 / 11

Kod İzleme: Palindrom (1/2) Hazırlık

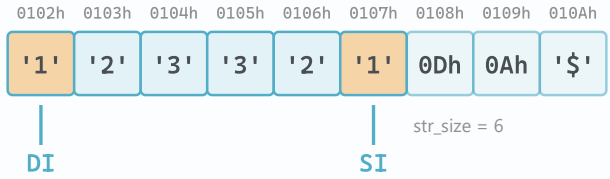
```
1  org 100h
2  jmp start
3  msg:
4  str db '123321'
5  str_size = $ - msg      ; = 6
6  db 0Dh,0Ah,'$'
7  start:
8  mov ah, 9               ; '$'e kadar yaz
9  mov dx, offset str
10 int 21h
11 lea di, str             ; DI = str adresi
12 mov si, di
13 add si, str_size        ; SI = son + 1
14 dec si                  ; SI = son karakter
▶  mov cx, str_size
16 cmp cx, 1               ; tek karakter mi?
17 je is_palindrome        ; tek karakter
   palindromdur
18 shr cx, 1               ; CX = uzunluk / 2
```

DEĞİŞKENLER

AH = **09h** | DX = **0102h** | DI = **0102h** | SI = **0107h** | CX = **0006h**

ÇIKTI

```
123321
```



CX = str_size = 6.

Adım 9 / 11

Kod İzleme: Palindrom (1/2) Hazırlık

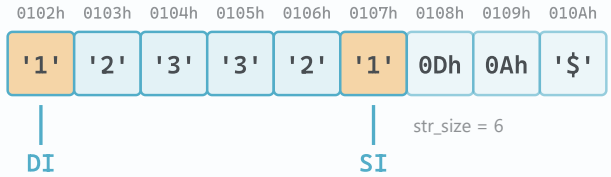
```
1  org 100h
2  jmp start
3  msg:
4  str db '123321'
5  str_size = $ - msg      ; = 6
6  db 0Dh,0Ah,'$'
7  start:
8  mov ah, 9               ; '$'e kadar yaz
9  mov dx, offset str
10 int 21h
11 lea di, str             ; DI = str adresi
12 mov si, di
13 add si, str_size        ; SI = son + 1
14 dec si                  ; SI = son karakter
15 mov cx, str_size
▶  cmp cx, 1               ; tek karakter mi?
▶  je is_palindrome        ; tek karakter
   palindromdur
18 shr cx, 1               ; CX = uzunluk / 2
```

DEĞİŞKENLER

AH = **09h** DX = **0102h** DI = **0102h**
SI = **0107h** CX = **0006h** ZF = **0**
CF = **0**

ÇIKTI

```
123321
```



6 − 1 ≠ 0 → ZF = 0, `je` atlamaz.

Adım 10 / 11

Kod İzleme: Palindrom (1/2) Hazırlık

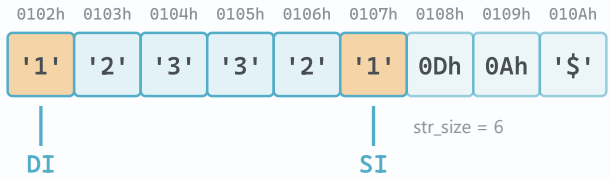
```
1  org 100h
2  jmp start
3  msg:
4  str db '123321'
5  str_size = $ - msg      ; = 6
6  db 0Dh,0Ah,'$'
7  start:
8  mov ah, 9               ; '$'e kadar yaz
9  mov dx, offset str
10 int 21h
11 lea di, str             ; DI = str adresi
12 mov si, di
13 add si, str_size        ; SI = son + 1
14 dec si                  ; SI = son karakter
15 mov cx, str_size
16 cmp cx, 1               ; tek karakter mi?
17 je is_palindrome        ; tek karakter
   palindromdur
▶  shr cx, 1               ; CX = uzunluk / 2
```

DEĞİŞKENLER

AH = **09h** | DX = **0102h** | DI = **0102h** | SI = **0107h** | CX = **0003h** | ZF = **0** | CF = **0**

ÇIKTI

```
123321
```



`shr cx, 1` : 0110b → 0011b, CX = 3. Dışarı kayan bit 0 → CF = 0.

Adım 11 / 11

Kod İzleme: Palindrom (2/2) Karşılaştırma

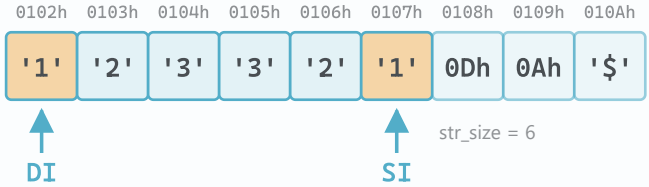
```
▶ next_char:
2  mov al, [di]          ; baştaki karakter
3  mov bl, [si]          ; sondaki karakter
4  cmp al, bl
5  jne not_palindrome    ; farklıysa değil
6  inc di                ; baştan bir ileri
7  dec si                ; sondan bir geri
8  loop next_char
9  is_palindrome:
10 mov ah, 9             ; "palindrome!" yaz
11 mov dx, offset msg1
12 int 21h
13 jmp stop
14 not_palindrome:
15 mov ah, 9             ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
19 ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

DI = **0102h** SI = **0107h** CX = **0003h**

ÇIKTI

```
123321
```



Döngü başı: DI = 0102h, SI = 0107h, CX = 3.

Adım 1 / 14

Kod İzleme: Palindrom (2/2) Karşılaştırma

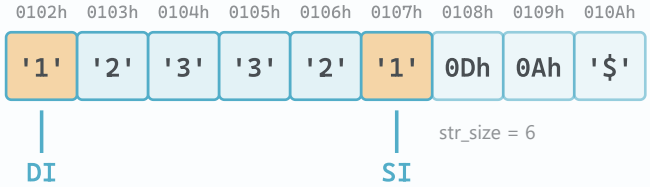
```
1  next_char:
▶ mov al, [di]          ; baştaki karakter
▶ mov bl, [si]          ; sondaki karakter
4  cmp al, bl
5  jne not_palindrome   ; farklıysa değil
6  inc di               ; baştan bir ileri
7  dec si               ; sondan bir geri
8  loop next_char
9  is_palindrome:
10 mov ah, 9            ; "palindrome!" yaz
11 mov dx, offset msg1
12 int 21h
13 jmp stop
14 not_palindrome:
15 mov ah, 9            ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
19 ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

AL = 31h | BL = 31h | DI = 0102h | SI = 0107h | CX = 0003h

ÇIKTI

```
123321
```



AL = [DI] = '1', BL = [SI] = '1'.

Adım 2 / 14

Kod İzleme: Palindrom (2/2) Karşılaştırma

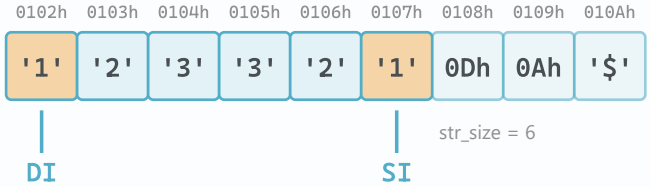
```
1  next_char:
2  mov al, [di]          ; baştaki karakter
3  mov bl, [si]          ; sondaki karakter
▶  cmp al, bl
▶  jne not_palindrome    ; farklıysa değil
6  inc di                ; baştan bir ileri
7  dec si                ; sondan bir geri
8  loop next_char
9  is_palindrome:
10 mov ah, 9             ; "palindrome!" yaz
11 mov dx, offset msg1
12 int 21h
13 jmp stop
14 not_palindrome:
15 mov ah, 9             ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
19 ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

AL = 31h | BL = 31h | DI = 0102h | SI = 0107h | CX = 0003h | ZF = 1

ÇIKTI

```
123321
```



Eşit → ZF = 1, `jne` atlamaz.

Adım 3 / 14

Kod İzleme: Palindrom (2/2) Karşılaştırma

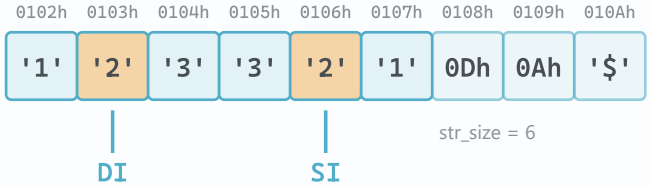
```
1  next_char:
2  mov al, [di]          ; baştaki karakter
3  mov bl, [si]          ; sondaki karakter
4  cmp al, bl
5  jne not_palindrome    ; farklıysa değil
▶  inc di                ; baştan bir ileri
▶  dec si                ; sondan bir geri
▶  loop next_char
9  is_palindrome:
10 mov ah, 9             ; "palindrome!" yaz
11 mov dx, offset msg1
12 int 21h
13 jmp stop
14 not_palindrome:
15 mov ah, 9             ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
19 ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

AL = 31h | BL = 31h | DI = 0103h | SI = 0106h | CX = 0002h | ZF = 0

ÇIKTI

```
123321
```



DI = 0103h, SI = 0106h (INC/DEC sonucu sıfır değil → ZF = 0), CX = 2 → next_char.

Adım 4 / 14

Kod İzleme: Palindrom (2/2) Karşılaştırma

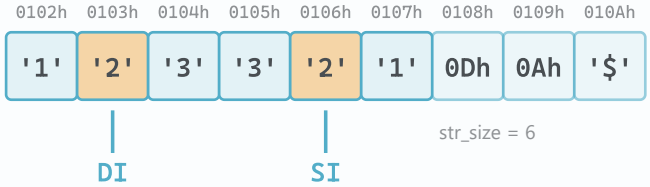
```
1  next_char:
▶  mov al, [di]          ; baştaki karakter
▶  mov bl, [si]          ; sondaki karakter
4  cmp al, bl
5  jne not_palindrome    ; farklıysa değil
6  inc di                ; baştan bir ileri
7  dec si                ; sondan bir geri
8  loop next_char
9  is_palindrome:
10 mov ah, 9             ; "palindrome!" yaz
11 mov dx, offset msg1
12 int 21h
13 jmp stop
14 not_palindrome:
15 mov ah, 9             ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
19 ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

AL = 32h | BL = 32h | DI = 0103h | SI = 0106h | CX = 0002h | ZF = 0

ÇIKTI

```
123321
```



AL = [DI] = '2', BL = [SI] = '2'.

Adım 5 / 14

Kod İzleme: Palindrom (2/2) Karşılaştırma

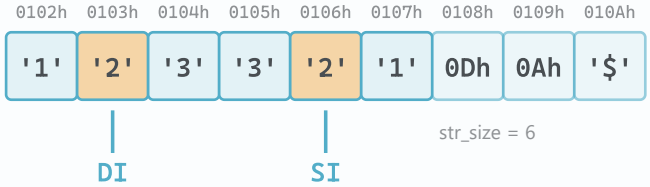
```
1  next_char:
2  mov al, [di]          ; baştaki karakter
3  mov bl, [si]          ; sondaki karakter
▶  cmp al, bl
▶  jne not_palindrome    ; farklıysa değil
6  inc di                ; baştan bir ileri
7  dec si                ; sondan bir geri
8  loop next_char
9  is_palindrome:
10 mov ah, 9             ; "palindrome!" yaz
11 mov dx, offset msg1
12 int 21h
13 jmp stop
14 not_palindrome:
15 mov ah, 9             ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
19 ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

AL = 32h | BL = 32h | DI = 0103h | SI = 0106h | CX = 0002h | ZF = 1

ÇIKTI

```
123321
```



Eşit → ZF = 1, `jne` atlamaz.

Adım 6 / 14

Kod İzleme: Palindrom (2/2) Karşılaştırma

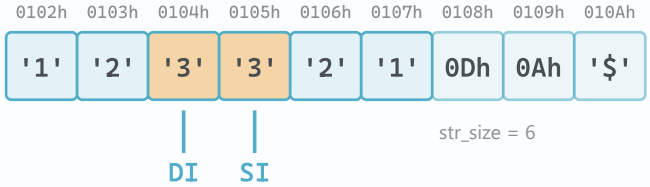
```
1  next_char:
2  mov al, [di]          ; baştaki karakter
3  mov bl, [si]          ; sondaki karakter
4  cmp al, bl
5  jne not_palindrome    ; farklıysa değil
▶  inc di                ; baştan bir ileri
▶  dec si                ; sondan bir geri
▶  loop next_char
9  is_palindrome:
10 mov ah, 9             ; "palindrome!" yaz
11 mov dx, offset msg1
12 int 21h
13 jmp stop
14 not_palindrome:
15 mov ah, 9             ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
19 ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

AL = 32h | BL = 32h | DI = 0104h | SI = 0105h | CX = 0001h | ZF = 0

ÇIKTI

```
123321
```



DI = 0104h, SI = 0105h, CX = 1 → next_char.

Kod İzleme: Palindrom (2/2) Karşılaştırma

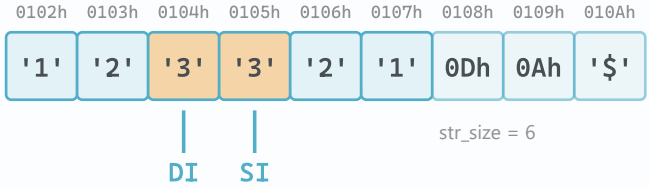
```
1  next_char:
▶  mov al, [di]          ; baştaki karakter
▶  mov bl, [si]          ; sondaki karakter
4  cmp al, bl
5  jne not_palindrome    ; farklıysa değil
6  inc di                ; baştan bir ileri
7  dec si                ; sondan bir geri
8  loop next_char
9  is_palindrome:
10 mov ah, 9             ; "palindrome!" yaz
11 mov dx, offset msg1
12 int 21h
13 jmp stop
14 not_palindrome:
15 mov ah, 9             ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
19 ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

AL = 33h | BL = 33h | DI = 0104h | SI = 0105h | CX = 0001h | ZF = 0

ÇIKTI

```
123321
```



AL = [DI] = '3', BL = [SI] = '3'.

Adım 8 / 14

Kod İzleme: Palindrom (2/2) Karşılaştırma

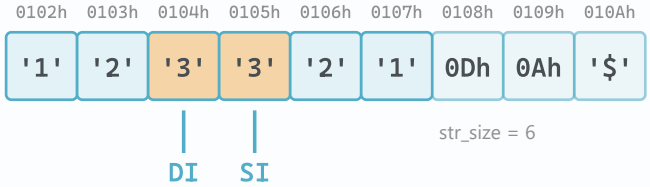
```
1  next_char:
2  mov al, [di]          ; baştaki karakter
3  mov bl, [si]          ; sondaki karakter
▶  cmp al, bl
▶  jne not_palindrome    ; farklıysa değil
6  inc di                ; baştan bir ileri
7  dec si                ; sondan bir geri
8  loop next_char
9  is_palindrome:
10 mov ah, 9             ; "palindrome!" yaz
11 mov dx, offset msg1
12 int 21h
13 jmp stop
14 not_palindrome:
15 mov ah, 9             ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
19 ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

AL = 33h | BL = 33h | DI = 0104h | SI = 0105h | CX = 0001h | ZF = 1

ÇIKTI

```
123321
```



Eşit → ZF = 1, `jne` atlamaz.

Adım 9 / 14

Kod İzleme: Palindrom (2/2) Karşılaştırma

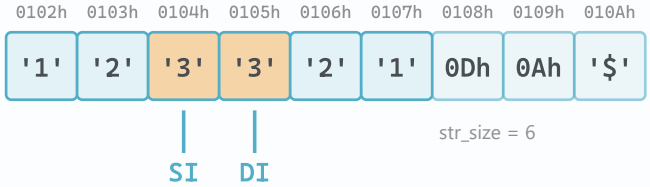
```
1  next_char:
2  mov al, [di]          ; baştaki karakter
3  mov bl, [si]          ; sondaki karakter
4  cmp al, bl
5  jne not_palindrome    ; farklıysa değil
▶  inc di                ; baştan bir ileri
▶  dec si                ; sondan bir geri
▶  loop next_char
9  is_palindrome:
10 mov ah, 9             ; "palindrome!" yaz
11 mov dx, offset msg1
12 int 21h
13 jmp stop
14 not_palindrome:
15 mov ah, 9             ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
19 ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

AL = 33h | BL = 33h | DI = 0105h | SI = 0104h | CX = 0000h | ZF = 0

ÇIKTI

```
123321
```



CX = 0 → döngü biter: tüm çiftler eşleşti.

Adım 10 / 14

Kod İzleme: Palindrom (2/2) Karşılaştırma

```
1  next_char:
2  mov al, [di]          ; baştaki karakter
3  mov bl, [si]          ; sondaki karakter
4  cmp al, bl
5  jne not_palindrome    ; farklıysa değil
6  inc di                ; baştan bir ileri
7  dec si                ; sondan bir geri
8  loop next_char
9  is_palindrome:
▶  mov ah, 9             ; "palindrome!" yaz
▶  mov dx, offset msg1
12 int 21h
13 jmp stop
14 not_palindrome:
15 mov ah, 9             ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
19 ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

AL = 33h | BL = 33h | DI = 0105h | SI = 0104h | CX = 0000h | ZF = 0 | AH = 09h | DX = 0142h

ÇIKTI

```
123321
```

0102h	0103h	0104h	0105h	0106h	0107h	0108h	0109h	010Ah
'1'	'2'	'3'	'3'	'2'	'1'	0Dh	0Ah	'$'

str_size = 6

AH = 09h, DX = offset msg1 = 0142h.

Kod İzleme: Palindrom (2/2) Karşılaştırma

```
1  next_char:
2  mov al, [di]          ; baştaki karakter
3  mov bl, [si]          ; sondaki karakter
4  cmp al, bl
5  jne not_palindrome    ; farklıysa değil
6  inc di                ; baştan bir ileri
7  dec si                ; sondan bir geri
8  loop next_char
9  is_palindrome:
10 mov ah, 9             ; "palindrome!" yaz
11 mov dx, offset msg1
▶ int 21h
13 jmp stop
14 not_palindrome:
15 mov ah, 9             ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
19 ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

AL = 24h | BL = 33h | DI = 0105h | SI = 0104h | CX = 0000h | ZF = 0 | AH = 09h | DX = 0142h

ÇIKTI

```
123321
palindrome!
```

0102h	0103h	0104h	0105h	0106h	0107h	0108h	0109h	010Ah
'1'	'2'	'3'	'3'	'2'	'1'	0Dh	0Ah	'$'

str_size = 6

INT 21h: **palindrome!** yazılır. (DOS bu işlevden AL = 24h, yani '$' ile döner.)

Kod İzleme: Palindrom (2/2) Karşılaştırma

```
1  next_char:
2  mov al, [di]          ; baştaki karakter
3  mov bl, [si]          ; sondaki karakter
4  cmp al, bl
5  jne not_palindrome    ; farklıysa değil
6  inc di                ; baştan bir ileri
7  dec si                ; sondan bir geri
8  loop next_char
9  is_palindrome:
10 mov ah, 9             ; "palindrome!" yaz
11 mov dx, offset msg1
12 int 21h
▶  jmp stop
14 not_palindrome:
15 mov ah, 9             ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
19 ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

AL = **24h** | BL = **33h** | DI = **0105h**
SI = **0104h** | CX = **0000h** | ZF = **0**
AH = **09h** | DX = **0142h**

ÇIKTI

```
123321
palindrome!
```

0102h	0103h	0104h	0105h	0106h	0107h	0108h	0109h	010Ah
'1'	'2'	'3'	'3'	'2'	'1'	0Dh	0Ah	'$'

str_size = 6

`jmp stop` : "not palindrome!" bloğu atlanır.

Adım 13 / 14

Kod İzleme: Palindrom (2/2) Karşılaştırma

```
1  next_char:
2  mov al, [di]          ; baştaki karakter
3  mov bl, [si]          ; sondaki karakter
4  cmp al, bl
5  jne not_palindrome    ; farklıysa değil
6  inc di                ; baştan bir ileri
7  dec si                ; sondan bir geri
8  loop next_char
9  is_palindrome:
10 mov ah, 9             ; "palindrome!" yaz
11 mov dx, offset msg1
12 int 21h
13 jmp stop
14 not_palindrome:
15 mov ah, 9             ; "not palindrome!" yaz
16 mov dx, offset msg2
17 int 21h
18 stop:
▶  ret
20 msg1 db "palindrome!$"
21 msg2 db "not palindrome!$"
```

DEĞİŞKENLER

AL = **24h** | BL = **33h** | DI = **0105h**
SI = **0104h** | CX = **0000h** | ZF = **0**
AH = **09h** | DX = **0142h**

ÇIKTI

```
123321
palindrome!
```

0102h	0103h	0104h	0105h	0106h	0107h	0108h	0109h	010Ah
'1'	'2'	'3'	'3'	'2'	'1'	0Dh	0Ah	'$'

str_size = 6

`ret`. Veriler `stop:` etiketinin hemen ardından geldiği için `ret` olmadan işlemci "palindrome!$" baytlarını komut sanıp yürütürdü.

Adım 14 / 14

ÖRNEK PROGRAMLAR

Palindrom: Farklı Girişler

str	Uzunluk	CX (shr sonrası)	Karşılaştırmalar	Çıktı
`'123321'`	6	3	1=1, 2=2, 3=3	palindrome!
`'123421'`	6	3	1=1, 2=2, 3≠4 → jne	not palindrome!
`'12321'`	5	2	1=1, 2=2 (ortadaki 3 atlanır)	palindrome!
`'7'`	1	—	cmp cx,1 → je is_palindrome	palindrome!
`'ab'`	2	1	a≠b → jne	not palindrome!

Neden `cmp cx, 1`? Uzunluk 1 iken `shr cx, 1` CX'i 0 yapar; LOOP'lu döngüye CX = 0 ile girilirse karşılaştırma 65.536 kez çalışırdı. Uzunluk 0 da aynı tehlikeyi taşır: `jcxz` ile korunabilir.

Laboratuvar: Atlamalar ve Döngüler

JMP örneği

KOD · **15 bayt**

```
 1 org    100h
 2 mov    ax, 5              ; ax…
 3 mov    bx, 2              ; bx…
 4 jmp    calc               ; 'c…
 5 back:  jmp stop           ; 's…
 6 calc:
 7 add    ax, bx             ; ax…
 8 jmp    back               ; 'b…
 9 stop:
10 ret                       ; iş…
```

YAZMAÇLAR

AX	0000	AH 00	AL 00		
BX	0000	BH 00	BL 00		
CX	000F	CH 00	CL 0F		
DX	0000	DH 00	DL 00		
SI	0000	DI 0000	BP 0000		
SP	FFFE	CS 0700	IP 0100		
DS	0700	ES 0700	SS 0700		

BAYRAKLAR

O	D	I	T	S	Z	A	P	C
0	0	1	0	0	0	0	0	0

CS:IP = 0700:0100 · adım 0

BELLEK DS = 0700h

```
0100 B8 05 00 BB 02 00 EB 02
0108 EB 04 03 C3 EB FA C3 00
0110 00 00 00 00 00 00 00 00
0118 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00
```

YIĞIN SS:SP

```
FFF6 0000
FFF8 0000
FFFA 0000
FFFC 0000
FFFE 0000 ◂ SP
```

ÇIKTI

Program yüklendi: 15 bayt, CS:IP = 0700:0100h. emu8086'daki gibi CX başlangıçta program boyutunu (000Fh) gösterir. Adım düğmesiyle komutlar tek tek yürütülür; vurgulu satır sıradaki komuttur.

ÖZET

Özet

JMP
Koşulsuz; IP ← hedef. Kısa (EB, ±128), yakın (E9, kesim içi), uzak (EA, CS:IP).

Bağıl konum
hedef − sonraki komutun adresi; negatifse ikiye tümleyenle saklanır.

Jcc
Bayraklara bakar, bayrakları değiştirmez; 8086'da yalnız kısa (2 bayt).

CMP
Sonucu saklamayan SUB; ardından gelen Jcc kararı verir.

İşaretsiz
JA, JAE, JB, JBE → CF ve ZF.

İşaretli
JG, JGE, JL, JLE → ZF, SF ve OF.

LOOP ailesi
CX azalır, bayraklar korunur; LOOPE/LOOPNE ZF'ye de bakar.

Tuzaklar
CX = 0 ile LOOP, iç içe döngüde CX'i saklamamak, veriyi kod gibi yürütmek.

ÖZET

Alıştırmalar

1. 0120h adresindeki `JNE` komutu 0100h adresine atlamak istiyor. Makine kodu nedir?
2. `mov al, 80h` / `cmp al, 1` sonrasında `JA` ve `JG` atlar mı?
3. `mov cx, 0` / `d: loop d` kaç kez döner? Nasıl önlenir?
4. AX'teki işaretsiz sayı 100'den büyükse BX = 1, değilse BX = 0 yapan kodu yazın.
5. Dizi toplamı örneğinde dizinin elemanları 200, 100 olsaydı AL ve CF ne olurdu? Çözüm önerin.
6. Hello örneğinde `jmp start` satırı silinirse ne olur?

▶ **İpuçları / cevaplar**

Kaynaklar

- emu8086 belgeleri: *8086 Assembler Tutorial for Beginners* (Part 7: Program Flow Control).
- Intel, *The 8086 Family User's Manual*, 1979: komut kümesi ve kodlama tabloları.
- Barry B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson, 2009: Bölüm 6 "Program Control Instructions".
- Kip R. Irvine, *Assembly Language for x86 Processors*, 7. baskı, Pearson, 2014: koşullu işlem ve döngüler.

Son

Bölüm 7: Akış Kontrolü

Sonraki bölüm: Bölüm 8.1: Yığın · Terim sözlüğü