Bölüm 10.2: Aygıt Kontrolü

Mikroişlemciler

IN / OUT · Yoklama · Trafik ışıkları · INT 15h bekleme · Yazılımla gecikme · 8255 PPI · Step motor · LED ekran · 7 bölütlü gösterge · Termometre · Yazıcı · Robot · Test aygıtı

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Port tabanlı G/Ç

`IN` ve `OUT` komutlarının biçimlerini ve sınırlarını bilmek.

Bit düzeyinde denetim

Bir porttaki her bitin bir lambaya, mıknatısa ya da duruma karşılık geldiğini çözümlemek.

Yoklama (polling)

Aygıtın hazır bitini bekleyen döngüler yazmak.

Zamanlama

INT 15h / AH = 86h ile mikrosaniye cinsinden bekleme süresi hesaplamak.

emu8086 aygıtları

Trafik ışıkları, step motor, LED ekran, termometre, yazıcı, robot ve test aygıtını programlamak.

Gömülü sistem bakışı

Programın ROM'dan RAM'e yüklenip çalıştırılma ilkesini açıklamak.

GİRİŞ

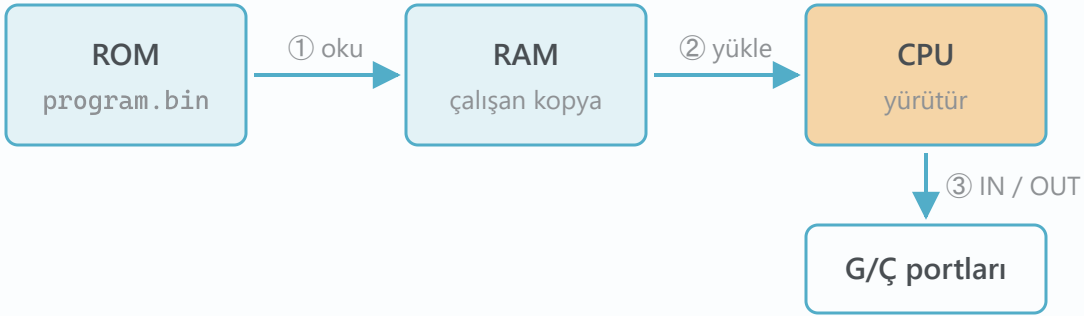
emu8086'ya Bağlı 7 Sanal Aygıt

Aygıt	Port(lar)	Yön	Veri
Trafik lambaları (traffic lights)	4	OUT	sözcük: 12 lamba biti
Step motor (stepper motor)	7	OUT / IN	bit 0–2 mıknatıslar, bit 7 hazır
Robot	9, 10, 11	OUT / IN	komut, veri, durum
Test aygıtı (simple io test)	110, 112	IN / OUT	bayt (110), sözcük (112)
Termometre ve ısıtıcı	125, 127	IN / OUT	sıcaklık, ısıtıcı aç/kapa
Yazıcı (printer)	130	OUT	karakter (INT 21h/05h ile)
LED ekran (LED display)	199	OUT	sözcük: işaretli sayı

Aygıtlar emu8086'da *virtual devices* menüsünden açılır; programın başına `#start=Traffic_Lights.exe#` gibi bir direktif yazılırsa öykünücü aygıtı kendisi başlatır.

Aygıt Kontrolü

- x86 ailesi CPU'lar geniş bir aygıt yelpazesini denetleyebilir. Temel G/Ç **port numaraları** sistemden sisteme farklılık gösterebilir ve uygun bir donanımla (ör. adres çözücü devresi) değiştirilebilir.
- Gömülü bir sistemde derlenen `.bin` dosyası **Salt Okunur Bellek** (ROM) yongasına yazılır.
- Sistem açılınca programı ROM'dan okur, RAM'e yükler ve yürütür. Bu ilke birçok modern cihazda (çamaşır makinesi, yazıcı, modem…) kullanılır.



IN ve OUT Komutları

Biçim	Anlamı	Port aralığı
`IN AL, port`	porttan 1 bayt oku	0–255 (sabit)
`IN AX, port`	porttan 1 sözcük oku	0–255 (sabit)
`IN AL, DX` / `IN AX, DX`	port numarası DX'te	0–65 535
`OUT port, AL` / `OUT port, AX`	porta yaz	0–255 (sabit)
`OUT DX, AL` / `OUT DX, AX`	port numarası DX'te	0–65 535



Veri **yalnızca AL ya da AX** üzerinden geçer: `OUT 4, BX` veya `IN CL, 7` geçersizdir.

255'ten büyük port için önce `MOV DX, 300` yazılır, sonra `OUT DX, AL`. Bu derste tüm emu8086 portları 255'in altındadır.

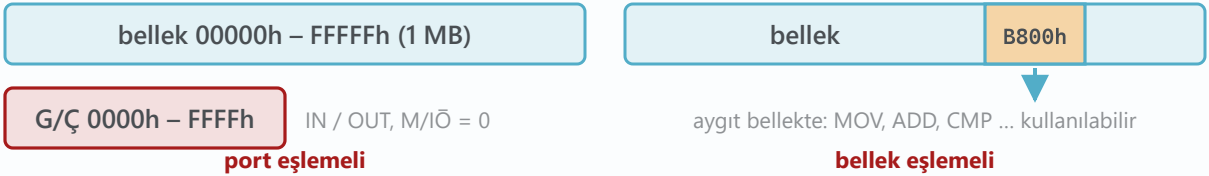
Port Eşlemeli ve Bellek Eşlemeli G/Ç

Port eşlemeli (isolated) G/Ç

- 8086'nın ayrı bir 64 K'lık G/Ç adres uzayı vardır.
- Erişim yalnızca `IN` / `OUT` ile; M/IO̅ pini G/Ç döngüsünü belirtir.
- Bellek adreslerini tüketmez.
- emu8086'nın sanal aygıtlarının hepsi böyledir.

Bellek eşlemeli (memory-mapped) G/Ç

- Aygıt yazmaçları bellek adreslerinde durur.
- Her MOV/ADD/CMP komutu kullanılabilir.
- Örnek: PC metin ekranı B800:0000'dan başlar; oraya yazılan bayt ekranda görünür.

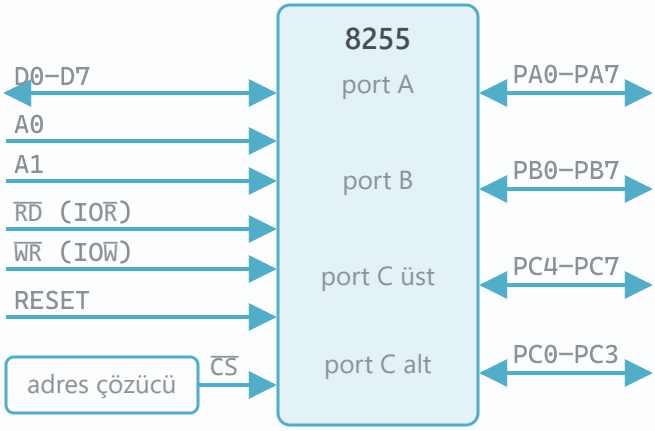


Yoklama (polling): CPU aygıtın durum bitini döngü içinde okuyarak "hazır" olmasını bekler. Basittir ama beklerken CPU başka iş yapamaz. Alternatifi aygıtın CPU'ya **kesme** göndermesidir.

8255 Programlanabilir Çevre Arayüzü (PPI)

- Üç 8 bitlik port (**A**, **B**, iki 4 bitlik yarıya ayrılabilen **C**) ve bir **kontrol yazmacı**; yazmaç **A1, A0** ile seçilir, $\overline{CS}$ adres çözücüden gelir.
- RESET sonrası tüm portlar **giriş**tir; önce kontrol sözcüğü yazılır.

A1 A0	Yazmaç	Adres (taban 310h)
0 0	port A	310h
0 1	port B	311h
1 0	port C	312h
1 1	kontrol yazmacı	313h



8086'da çift adresler D0–D7'den (alt bank) aktarılır; yazmaçlar çift adrese düşsün diye 8255.A0 ← A1, A1 ← A2 bağlanır (310h, 312h, 314h, 316h). 8088'de A0 ← A0, A1 ← A1 yeterlidir. Port > 255: erişim `DX` ile.

8255 Kontrol Sözcüğü (Mod 0)

D7	D6 D5	D4	D3	D2	D1	D0
1 = mod seçimi	A grubu modu 00 = mod 0	port A	PC7–PC4	B grubu modu 0 = mod 0	port B	PC3–PC0
		yön bitleri:	**1 = giriş**, **0 = çıkış** (D4, D3, D1, D0)			

1 00 0 0 0 1 0b = **82h**

İstenen: A çıkış (LED/röle), B giriş (anahtar), C'nin iki yarısı çıkış, hepsi mod 0.

Adım 1 / 4

```
mov dx, 313h   ; kontrol yazmacı
mov al, 82h    ; A çık, B gir, C çık
out dx, al
mov dx, 311h   ; port B
in  al, dx     ; anahtarları oku
mov dx, 310h   ; port A
out dx, al     ; LED'lere yaz
mov dx, 313h
mov al, 0Fh    ; BSR: PC7 = 1
out dx, al
```

8255 Kontrol Sözcüğü (Mod 0)

D7	D6 D5	D4	D3	D2	D1	D0
1 = mod seçimi	A grubu modu 00 = mod 0	port A	PC7–PC4	B grubu modu 0 = mod 0	port B	PC3–PC0
		yön bitleri:	**1 = giriş**, **0 = çıkış** (D4, D3, D1, D0)			

D7 = 1: mod seçimi · D6 D5 = 00 ve D2 = 0: mod 0

Mod 0 el sıkışmasız basit G/Ç'dir; mod 1 ve 2 el sıkışmalı (strobe) aktarım içindir.


```
mov dx, 313h    ; kontrol yazmacı
mov al, 82h     ; A çık, B gir, C çık
out dx, al
mov dx, 311h    ; port B
in  al, dx      ; anahtarları oku
mov dx, 310h    ; port A
out dx, al      ; LED'lere yaz
mov dx, 313h
mov al, 0Fh     ; BSR: PC7 = 1
out dx, al
```

8255 Kontrol Sözcüğü (Mod 0)

D7	D6 D5	D4	D3	D2	D1	D0
1 = mod seçimi	A grubu modu 00 = mod 0	port A	PC7–PC4	B grubu modu 0 = mod 0	port B	PC3–PC0
		yön bitleri:	**1 = giriş**, **0 = çıkış** (D4, D3, D1, D0)			

D4 = 0 (A çıkış) · D3 = 0 (C üst çıkış)
D1 = 1 (B giriş) · D0 = 0 (C alt çıkış)

Sonuç 1000 0010b = 82h. Örneğin A giriş, B çıkış, C giriş için 1001 1001b = 99h olur.


```
mov dx, 313h    ; kontrol yazmacı
mov al, 82h     ; A çık, B gir, C çık
out dx, al
mov dx, 311h    ; port B
in  al, dx      ; anahtarları oku
mov dx, 310h    ; port A
out dx, al      ; LED'lere yaz
mov dx, 313h
mov al, 0Fh     ; BSR: PC7 = 1
out dx, al
```

8255 Kontrol Sözcüğü (Mod 0)

D7	D6 D5	D4	D3	D2	D1	D0
1 = mod seçimi	A grubu modu 00 = mod 0	port A	PC7–PC4	B grubu modu 0 = mod 0	port B	PC3–PC0
		yön bitleri:	**1 = giriş**, **0 = çıkış** (D4, D3, D1, D0)			

D7 = 0 ise **bit kur/sil** (BSR):
D3–D1 = C'nin bit numarası, D0 = yeni değer

0Fh (0000 1111b) → PC7 = 1, 0Eh → PC7 = 0. Diğer C bitleri değişmez.


```
mov dx, 313h    ; kontrol yazmacı
mov al, 82h     ; A çık, B gir, C çık
out dx, al
mov dx, 311h    ; port B
in  al, dx      ; anahtarları oku
mov dx, 310h    ; port A
out dx, al      ; LED'lere yaz
mov dx, 313h
mov al, 0Fh     ; BSR: PC7 = 1
out dx, al
```

8255: Kontrol Sözcüğü, Okuma, Yazma, BSR

```
1  org 100h
▶  mov dx, 313h   ; kontrol yazmacı
▶  mov al, 82h    ; A,C çık; B gir
4  out dx, al
5  mov dx, 311h   ; port B
6  in  al, dx     ; anahtarları oku
7  mov dx, 310h   ; port A
8  out dx, al     ; LED'lere yaz
9  mov dx, 313h
10 mov al, 0Fh    ; BSR: PC7 = 1
11 out dx, al
12 ret
```

DX: 0313h

AL: 82h

G/Ç kaydı:

8255 — kontrol = yok (hepsi giriş)

A 310h — giriş — (yazılmadı)

B 311h — giriş ← anahtar — 1 0 0 1 0 1 1 0

C 312h — giriş — (yazılmadı)

RESET sonrası üç port da giriş. Port B'ye bağlı anahtarlar 1001 0110b = **96h** konumunda (port sözlüğünde 311h = 96h). DX = 313h, AL = 82h.

Adım 1 / 8

DEĞİŞKENLER

DX = **0313h** AL = **82h**

Port değerleri yorumlayıcının G/Ç kaydından (anahtar girişi 96h örnek). BSR sözcüğü yalnız seçilen C bitini değiştirir.

8255: Kontrol Sözcüğü, Okuma, Yazma, BSR

```
1  org 100h
2  mov dx, 313h    ; kontrol yazmacı
3  mov al, 82h     ; A,C çık; B gir
▶  out dx, al
5  mov dx, 311h    ; port B
6  in  al, dx      ; anahtarları oku
7  mov dx, 310h    ; port A
8  out dx, al      ; LED'lere yaz
9  mov dx, 313h
10 mov al, 0Fh     ; BSR: PC7 = 1
11 out dx, al
12 ret
```

DX: **0313h** AL: **82h**

OUT → kontrol

G/Ç kaydı:
`OUT 313h ← 82h`

8255 — kontrol = 82h

A 310h — (yazılmadı)
çıkış → LED

B 311h
giriş ← anahtar

1	0	0	1	0	1	1	0

C 312h — (yazılmadı)
çıkış

OUT 313h ← **82h** = 1 00 0 0 0 1 0b: mod 0; A çıkış, C üst çıkış, B giriş, C alt çıkış.

Adım 2 / 8

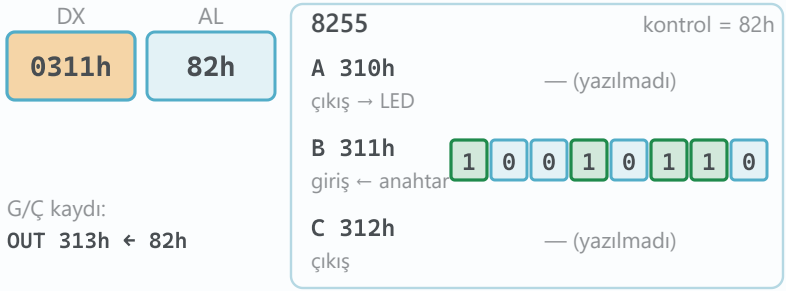
DEĞİŞKENLER

DX = **0313h** AL = **82h**

Port değerleri yorumlayıcının G/Ç kaydından (anahtar girişi 96h örnek). BSR sözcüğü yalnız seçilen C bitini değiştirir.

8255: Kontrol Sözcüğü, Okuma, Yazma, BSR

```
1  org 100h
2  mov dx, 313h    ; kontrol yazmacı
3  mov al, 82h     ; A,C çık; B gir
4  out dx, al
▶  mov dx, 311h    ; port B
6  in  al, dx      ; anahtarları oku
7  mov dx, 310h    ; port A
8  out dx, al      ; LED'lere yaz
9  mov dx, 313h
10 mov al, 0Fh     ; BSR: PC7 = 1
11 out dx, al
12 ret
```



DX = 311h: port B. 311h > 255 olduğundan port numarası DX'te verilir.

Adım 3 / 8

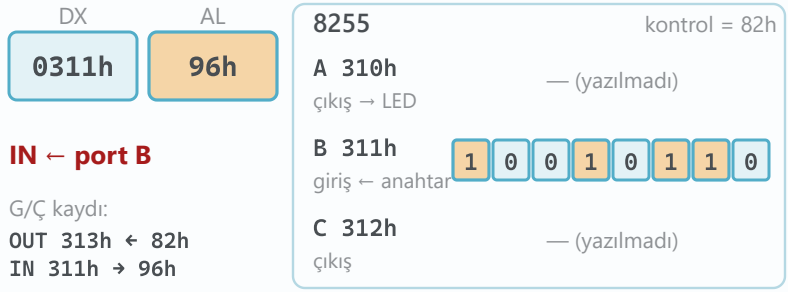
DEĞİŞKENLER

DX = 0311h AL = 82h

Port değerleri yorumlayıcının G/Ç kaydından (anahtar girişi 96h örnek). BSR sözcüğü yalnız seçilen C bitini değiştirir.

8255: Kontrol Sözcüğü, Okuma, Yazma, BSR

```
1  org 100h
2  mov dx, 313h    ; kontrol yazmacı
3  mov al, 82h     ; A,C çık; B gir
4  out dx, al
5  mov dx, 311h    ; port B
▶  in  al, dx      ; anahtarları oku
7  mov dx, 310h    ; port A
8  out dx, al      ; LED'lere yaz
9  mov dx, 313h
10 mov al, 0Fh     ; BSR: PC7 = 1
11 out dx, al
12 ret
```



IN AL, DX: anahtarlar okunur → AL = **96h**.

Adım 4 / 8

DEĞİŞKENLER

DX = **0311h** AL = **96h**

Port değerleri yorumlayıcının G/Ç kaydından (anahtar girişi 96h örnek). BSR sözcüğü yalnız seçilen C bitini değiştirir.

8255: Kontrol Sözcüğü, Okuma, Yazma, BSR

```
1  org 100h
2  mov dx, 313h    ; kontrol yazmacı
3  mov al, 82h     ; A,C çık; B gir
4  out dx, al
5  mov dx, 311h    ; port B
6  in  al, dx      ; anahtarları oku
▶  mov dx, 310h    ; port A
8  out dx, al      ; LED'lere yaz
9  mov dx, 313h
10 mov al, 0Fh     ; BSR: PC7 = 1
11 out dx, al
12 ret
```

DX: 0310h

AL: 96h

G/Ç kaydı:
`OUT 313h ← 82h`
`IN 311h → 96h`

8255 — kontrol = 82h

- **A 310h** — çıkış → LED — (yazılmadı)
- **B 311h** — giriş ← anahtar — 1 0 0 1 0 1 1 0
- **C 312h** — çıkış — (yazılmadı)

DX = 310h: port A.

Adım 5 / 8

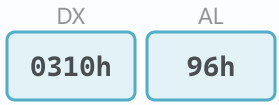
DEĞİŞKENLER

DX = 0310h | AL = 96h

Port değerleri yorumlayıcının G/Ç kaydından (anahtar girişi 96h örnek). BSR sözcüğü yalnız seçilen C bitini değiştirir.

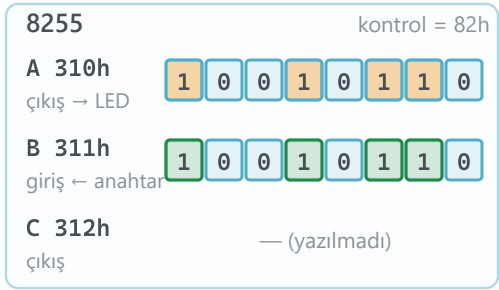
8255: Kontrol Sözcüğü, Okuma, Yazma, BSR

```
1  org 100h
2  mov dx, 313h    ; kontrol yazmacı
3  mov al, 82h     ; A,C çık; B gir
4  out dx, al
5  mov dx, 311h    ; port B
6  in  al, dx      ; anahtarları oku
7  mov dx, 310h    ; port A
▶  out dx, al      ; LED'lere yaz
9  mov dx, 313h
10 mov al, 0Fh     ; BSR: PC7 = 1
11 out dx, al
12 ret
```



OUT → port A

G/Ç kaydı:
`OUT 313h ← 82h`
`IN 311h → 96h`
`OUT 310h ← 96h`



OUT 310h ← 96h: port A çıkış olduğundan LED'ler anahtarları kopyalar (yanan: bit 7, 4, 2, 1).

Adım 6 / 8

DEĞİŞKENLER

`DX = 0310h` `AL = 96h`

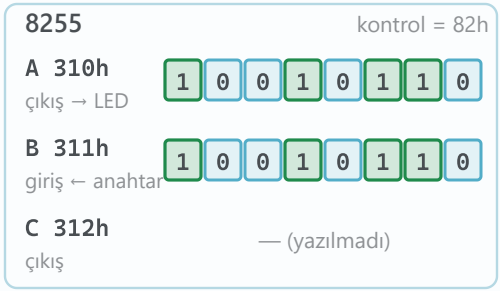
Port değerleri yorumlayıcının G/Ç kaydından (anahtar girişi 96h örnek). BSR sözcüğü yalnız seçilen C bitini değiştirir.

8255: Kontrol Sözcüğü, Okuma, Yazma, BSR

```
1  org 100h
2  mov dx, 313h    ; kontrol yazmacı
3  mov al, 82h     ; A,C çık; B gir
4  out dx, al
5  mov dx, 311h    ; port B
6  in  al, dx      ; anahtarları oku
7  mov dx, 310h    ; port A
8  out dx, al      ; LED'lere yaz
▶  mov dx, 313h
▶  mov al, 0Fh     ; BSR: PC7 = 1
11 out dx, al
12 ret
```

DX: **0313h** AL: **0Fh**

G/Ç kaydı:
`OUT 313h ← 82h`
`IN 311h → 96h`
`OUT 310h ← 96h`



DX = 313h, AL = **0Fh** = 0000 111 1b: D7 = 0 → mod sözcüğü değil, **bit kur/sil** (BSR).

Adım 7 / 8

DEĞİŞKENLER

DX = **0313h** AL = **0Fh**

Port değerleri yorumlayıcının G/Ç kaydından (anahtar girişi 96h örnek). BSR sözcüğü yalnız seçilen C bitini değiştirir.

8255: Kontrol Sözcüğü, Okuma, Yazma, BSR

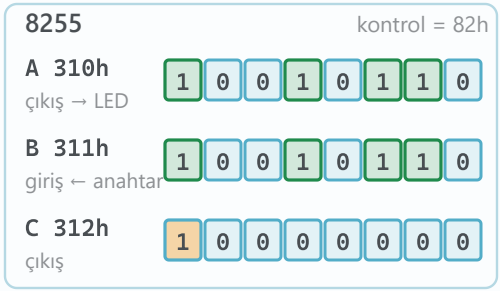
```
1  org 100h
2  mov dx, 313h    ; kontrol yazmacı
3  mov al, 82h     ; A,C çık; B gir
4  out dx, al
5  mov dx, 311h    ; port B
6  in  al, dx      ; anahtarları oku
7  mov dx, 310h    ; port A
8  out dx, al      ; LED'lere yaz
9  mov dx, 313h
10 mov al, 0Fh     ; BSR: PC7 = 1
▶  out dx, al
12 ret
```

DX: 0313h AL: 0Fh

OUT → kontrol (BSR)

G/Ç kaydı:

```
IN 311h → 96h
OUT 310h ← 96h
OUT 313h ← 0Fh
```



OUT 313h ← 0Fh: D3–D1 = 111 → PC7, D0 = 1 → **PC7 = 1**. Port yönleri (82h) değişmez, diğer C bitleri korunur.

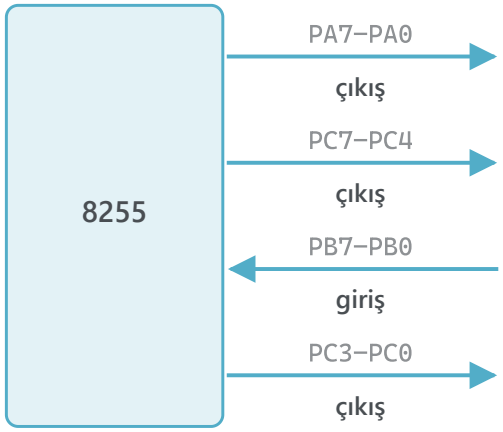
Adım 8 / 8

DEĞİŞKENLER

DX = 0313h AL = 0Fh

Port değerleri yorumlayıcının G/Ç kaydından (anahtar girişi 96h örnek). BSR sözcüğü yalnız seçilen C bitini değiştirir.

Etkileşimli 8255 Kontrol Sözcüğü



Port A (PA7–PA0)	çıkış (0)
Port C üst (PC7–PC4)	çıkış (0)
Port B (PB7–PB0)	giriş (1)
Port C alt (PC3–PC0)	çıkış (0)

Kontrol sözcüğü (D7...D0) **82** h

1	0	0	0	0	0	1	0
mod	A mod	A mod	PA	PC üst	B mod	PB	PC alt

```
mov dx, 313h   ; kontrol yazmacı
mov al, 82h    ; 10000010b
out dx, al
; Port A (DX = 310h): out dx, al
; Port C (DX = 312h): out dx, al
; Port B (DX = 311h): in  al, dx
```

Port A **çıkış** · Port C üst **çıkış** · Port B **giriş** · Port C alt **çıkış**

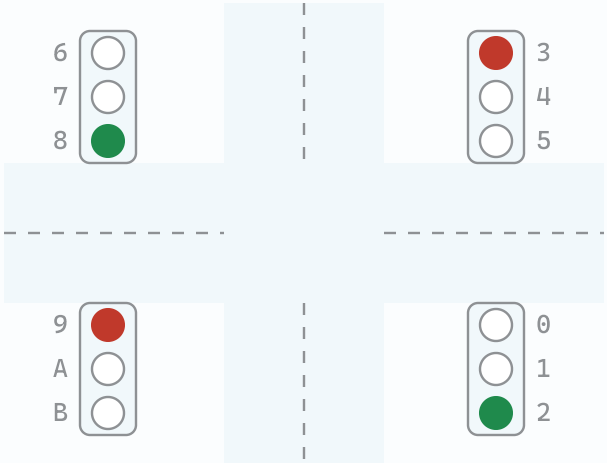
Port yönüne tıklanınca yön değişir; kutuya doğrudan bir değer de yazılabilir (ör. 9B, 80, 0F).

TRAFİK IŞIKLARI

Trafik Lambaları (Port 4)

- Kavşakta 4 direk, her direkte 3 lamba: toplam **12 lamba** = port 4'e yazılan sözcüğün **bit 0–11**'i.
- Bit = 1 → lamba yanar, 0 → söner. Bit 12–15 kullanılmaz.
- Her direkte (k = 0…3): bit 3k **kırmızı**, 3k+1 **sarı**, 3k+2 **yeşil**.
- Örnek: `0000_0011_0000_1100b` → bit 2 ve 8 (yeşil), bit 3 ve 9 (kırmızı).

Direk	Bitler
sağ alt	0 1 2
sağ üst	3 4 5
sol üst	6 7 8
sol alt	9 A B



Durum (Situation) Tablosu

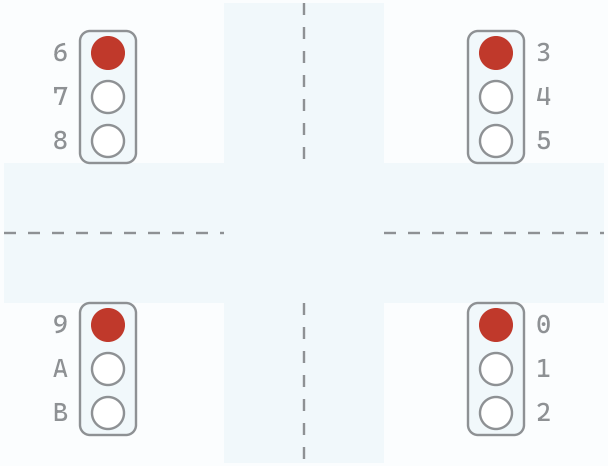
```
;                FEDC_BA98_7654_3210
situation     dw  0000_0011_0000_1100b
s1            dw  0000_0110_1001_1010b
s2            dw  0000_1000_0110_0001b
s3            dw  0000_1000_0110_0001b
s4            dw  0000_0100_1101_0011b
sit_end = $
all_red       equ 0000_0010_0100_1001b
```

Ad	Hex	Lambalar
`situation`	030Ch	sağ alt: yeşil; sağ üst: kırmızı; sol üst: yeşil; sol alt: kırmızı
`s1`	069Ah	sağ alt: sarı; sağ üst: kırmızı + sarı; sol üst: sarı; sol alt: kırmızı + sarı
`s2`	0861h	sağ alt: kırmızı; sağ üst: yeşil; sol üst: kırmızı; sol alt: yeşil
`s3`	0861h	sağ alt: kırmızı; sağ üst: yeşil; sol üst: kırmızı; sol alt: yeşil
`s4`	04D3h	sağ alt: kırmızı + sarı; sağ üst: sarı; sol üst: kırmızı + sarı; sol alt: sarı
`all_red`	0249h	sağ alt: kırmızı; sağ üst: kırmızı; sol üst: kırmızı; sol alt: kırmızı

s2 ve s3 aynıdır: yan yol yeşili iki aralık (10 s) sürer. s1 ve s4 sarı geçişleridir (kırmızı + sarı = “hazırlan”).

TRAFİK IŞIKLARI

Durumlar Adım Adım



FEDC BA98 7654 3210

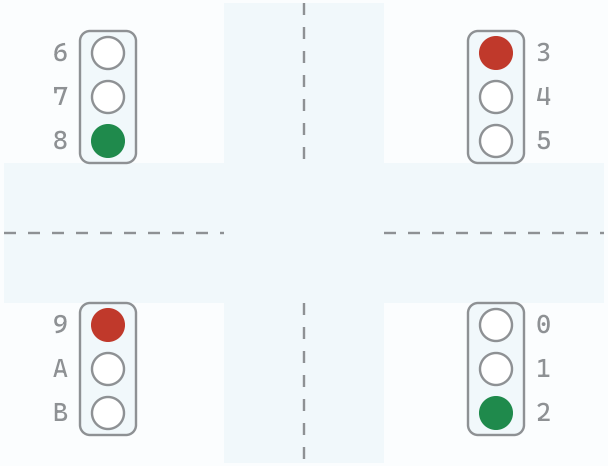
0000 0010 0100 1001

all_red = 0249h → sağ alt: kırmızı; sağ üst: kırmızı; sol üst: kırmızı; sol alt: kırmızı

Adım 1 / 6

Program önce tüm yönleri kırmızı yapar, sonra situation → s1 → s2 → s3 → s4 → situation... diye 5 saniyelik aralıklarla döner.

Durumlar Adım Adım



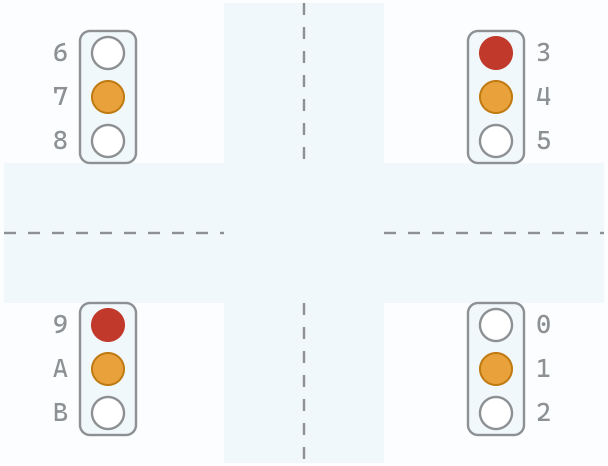
FEDC BA98 7654 3210

0000 0011 0000 1100

situation = 030Ch → sağ alt: yeşil; sağ üst: kırmızı; sol üst: yeşil; sol alt: kırmızı

Program önce tüm yönleri kırmızı yapar, sonra situation → s1 → s2 → s3 → s4 → situation... diye 5 saniyelik aralıklarla döner.

Durumlar Adım Adım



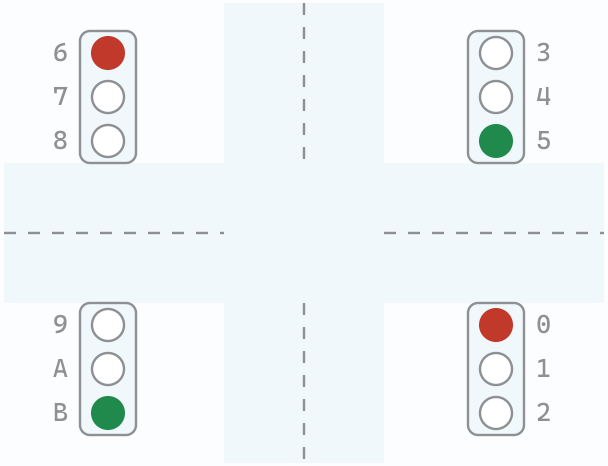
FEDC BA98 7654 3210

0000 0110 1001 1010

s1 = 069Ah → sağ alt: sarı; sağ üst: kırmızı + sarı; sol üst: sarı; sol alt: kırmızı + sarı

Program önce tüm yönleri kırmızı yapar, sonra situation → s1 → s2 → s3 → s4 → situation... diye 5 saniyelik aralıklarla döner.

Durumlar Adım Adım



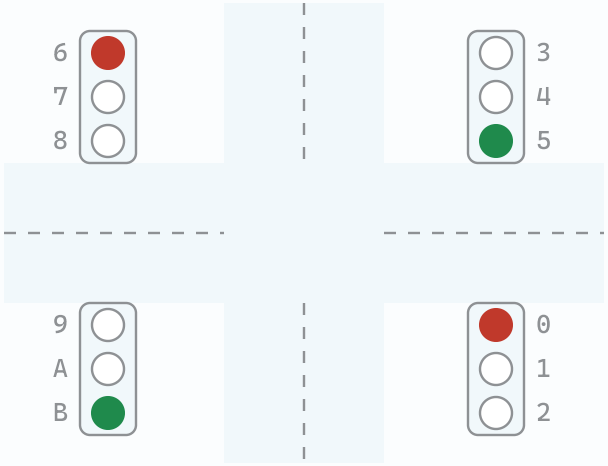
FEDC BA98 7654 3210

0000 1000 0110 0001

s2 = 0861h → sağ alt: kırmızı; sağ üst: yeşil; sol üst: kırmızı; sol alt: yeşil

Program önce tüm yönleri kırmızı yapar, sonra situation → s1 → s2 → s3 → s4 → situation... diye 5 saniyelik aralıklarla döner.

Durumlar Adım Adım



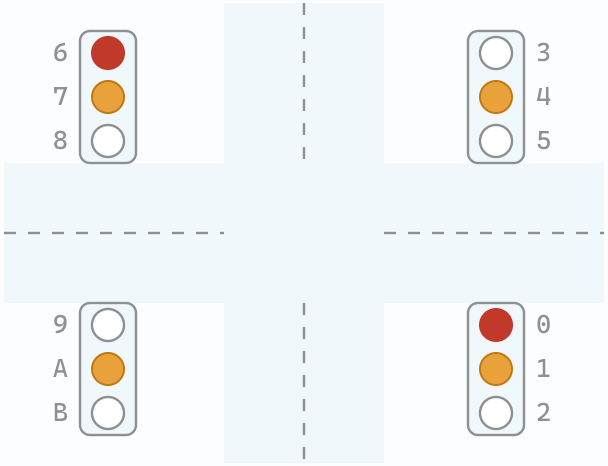
FEDC BA98 7654 3210

0000 1000 0110 0001

s3 = 0861h → sağ alt: kırmızı; sağ üst: yeşil; sol üst: kırmızı; sol alt: yeşil

Program önce tüm yönleri kırmızı yapar, sonra situation → s1 → s2 → s3 → s4 → situation… diye 5 saniyelik aralıklarla döner.

Durumlar Adım Adım



FEDC BA98 7654 3210

0000 0100 1101 0011

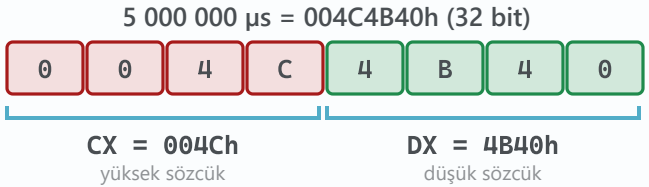
s4 = 04D3h → sağ alt: kırmızı + sarı; sağ üst: sarı; sol üst: kırmızı + sarı; sol alt: sarı

Program önce tüm yönleri kırmızı yapar, sonra situation → s1 → s2 → s3 → s4 → situation… diye 5 saniyelik aralıklarla döner.

INT 15h / AH = 86h ile Bekleme

```
; 5 saniye bekle (5 milyon µs)
mov cx, 4Ch     ; 004C4B40h = 5 000 000
mov dx, 4B40h
mov ah, 86h
int 15h
```

- BIOS işlevi **INT 15h, AH = 86h**: CX:DX mikrosaniye bekler.
- CX yüksek sözcük, DX düşük sözcük: 32 bitlik bekleme süresi.



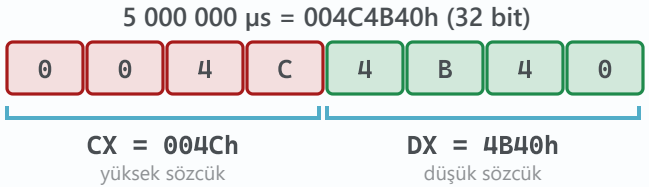
5 s = 5 × 1 000 000 = **5 000 000 µs**

Süre mikrosaniyeye çevrilir.

INT 15h / AH = 86h ile Bekleme

```
; 5 saniye bekle (5 milyon µs)
mov cx, 4Ch     ; 004C4B40h = 5 000 000
mov dx, 4B40h
mov ah, 86h
int 15h
```

- BIOS işlevi **INT 15h, AH = 86h**: CX:DX mikrosaniye bekler.
- CX yüksek sözcük, DX düşük sözcük: 32 bitlik bekleme süresi.



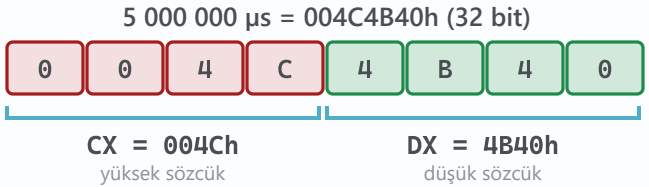
5 000 000 ÷ 16 = 312 500 kalan **0**
312 500 ÷ 16 = 19 531 kalan **4**
19 531 ÷ 16 = 1 220 kalan **11 (B)**
1 220 ÷ 16 = 76 kalan **4**
76 ÷ 16 = 4 kalan **12 (C)**
4 ÷ 16 = 0 kalan **4**

Ardışık 16'ya bölme: kalanlar sondan başa okunur → **4C4B40h**.

INT 15h / AH = 86h ile Bekleme

```
; 5 saniye bekle (5 milyon µs)
mov cx, 4Ch     ; 004C4B40h = 5 000 000
mov dx, 4B40h
mov ah, 86h
int 15h
```

- BIOS işlevi **INT 15h, AH = 86h**: CX:DX mikrosaniye bekler.
- CX yüksek sözcük, DX düşük sözcük: 32 bitlik bekleme süresi.



`0 0 4 C` | `4 B 4 0`

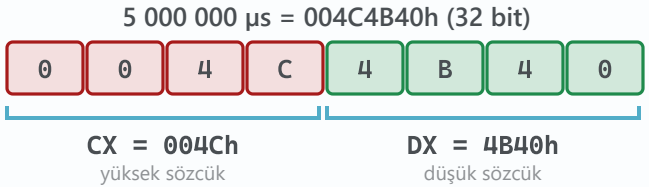
CX = **004Ch** · DX = **4B40h**

32 bitlik değer 0000 0000 0100 1100 0100 1011 0100 0000b ikiye bölünür.

INT 15h / AH = 86h ile Bekleme

```
; 5 saniye bekle (5 milyon µs)
mov cx, 4Ch     ; 004C4B40h = 5 000 000
mov dx, 4B40h
mov ah, 86h
int 15h
```

- BIOS işlevi **INT 15h, AH = 86h**: CX:DX mikrosaniye bekler.
- CX yüksek sözcük, DX düşük sözcük: 32 bitlik bekleme süresi.



1 s → 1 000 000 = 000F4240h → CX = 000Fh, DX = 4240h

Başka bir örnek: 1 saniyelik bekleme.

Yazılımla Gecikme: LOOP ve Saat Turu

```
; ≈ 1 ms gecikme (4,77 MHz 8086)
mov cx, 281     ; 4 tur
bekle:
loop bekle      ; 280 × 17 + 1 × 5 tur
```

Komut	Saat turu
`MOV CX, sabit`	4
`LOOP` (CX ≠ 0, atlar)	17
`LOOP` (CX = 0, atlamaz)	5

Toplam = 4 + 17(N − 1) + 5 = **17N − 8** saat turu.

4,77 MHz: 1 saat turu ≈ 0,2096 µs
1 ms = **4 770** saat turu

Süre, saat turu sayısının saat frekansına bölümüdür.

Hesap yaklaşıktır: kesmeler, bellek bekleme durumları ve komut kuyruğu süreyi uzatabilir. emu8086 komut zamanlamasını modellemez; döngü bilgisayarın ve adım gecikmesi ayarının hızında koşar. emu8086'da süreli bekleme INT 15h/86h ile yapılır.

Yazılımla Gecikme: LOOP ve Saat Turu

```
; ≈ 1 ms gecikme (4,77 MHz 8086)
mov cx, 281     ; 4 tur
bekle:
loop bekle      ; 280 × 17 + 1 × 5 tur
```

Komut	Saat turu
`MOV CX, sabit`	4
`LOOP` (CX ≠ 0, atlar)	17
`LOOP` (CX = 0, atlamaz)	5

Toplam = 4 + 17(N − 1) + 5 = **17N − 8** saat turu.

17N − 8 = 4 770 → N ≈ 281,06

N = **281** → 4 769 saat turu ≈ 0,9998 ms

CX'e tam sayı yazılabildiği için en yakın değer seçilir.

Hesap yaklaşıktır: kesmeler, bellek bekleme durumları ve komut kuyruğu süreyi uzatabilir. emu8086 komut zamanlamasını modellemez; döngü bilgisayarın ve adım gecikmesi ayarının hızında koşar. emu8086'da süreli bekleme INT 15h/86h ile yapılır.

Yazılımla Gecikme: LOOP ve Saat Turu

```
; ≈ 1 ms gecikme (4,77 MHz 8086)
mov cx, 281     ; 4 tur
bekle:
loop bekle      ; 280 × 17 + 1 × 5 tur
```

Komut	Saat turu
`MOV CX, sabit`	4
`LOOP` (CX ≠ 0, atlar)	17
`LOOP` (CX = 0, atlamaz)	5

Toplam = 4 + 17(N − 1) + 5 = **17N − 8** saat turu.

5 MHz: 1 ms = 5 000 saat turu → N ≈ 294,6
N = **295** → 5 007 saat turu ≈ 1,0014 ms

Aynı N = 281 5 MHz'de yalnızca 0,954 ms sürer: gecikme **saat hızına bağlıdır**.

Hesap yaklaşıktır: kesmeler, bellek bekleme durumları ve komut kuyruğu süreyi uzatabilir. emu8086 komut zamanlamasını modellemez; döngü bilgisayarın ve adım gecikmesi ayarının hızında koşar. emu8086'da süreli bekleme INT 15h/86h ile yapılır.

Yazılımla Gecikme: LOOP ve Saat Turu

```
; ≈ 1 ms gecikme (4,77 MHz 8086)
mov cx, 281     ; 4 tur
bekle:
loop bekle      ; 280 × 17 + 1 × 5 tur
```

Komut	Saat turu
`MOV CX, sabit`	4
`LOOP` (CX ≠ 0, atlar)	17
`LOOP` (CX = 0, atlamaz)	5

Toplam = 4 + 17(N − 1) + 5 = **17N − 8** saat turu.

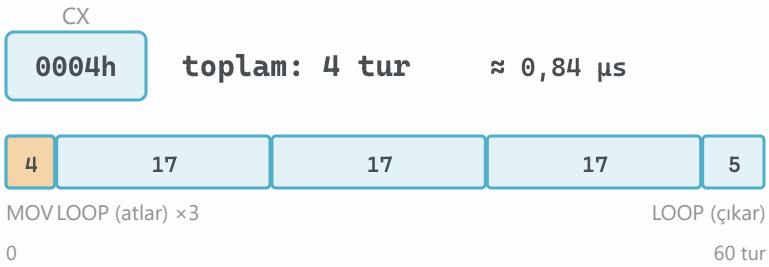
En uzun: CX = 0 → 65 536 tur
17 · 65 536 − 8 = 1 114 104 saat turu ≈ **233,6 ms**

Daha uzun bekleme için iç içe döngü gerekir. Prosedür olarak çağrılırsa CALL (19) ve RET (8) saat turu da eklenir.

Hesap yaklaşıktır: kesmeler, bellek bekleme durumları ve komut kuyruğu süreyi uzatabilir. emu8086 komut zamanlamasını modellemez; döngü bilgisayarın ve adım gecikmesi ayarının hızında koşar. emu8086'da süreli bekleme INT 15h/86h ile yapılır.

Gecikme Döngüsünde Saat Turlarını Saymak

```
1 org 100h
▶ mov cx, 4       ; 4 saat turu
3 bekle:
4 loop bekle      ; 17 tur, sonuncu 5
5 ret
```



CX = 4. `mov cx, sabit` 4 saat turu sürer; 4,77 MHz'de 1 tur ≈ 0,2096 µs.

Adım 1 / 6

DEĞİŞKENLER

`CX = 0004h` `tur = 4`

Tur sayıları 8086 komut sürelerinden; yorumlayıcı yalnız CX ve atlama davranışını doğrular (emu8086 zamanlamayı modellemez).

Gecikme Döngüsünde Saat Turlarını Saymak

```
1 org 100h
2 mov cx, 4       ; 4 saat turu
3 bekle:
▶ loop bekle      ; 17 tur, sonuncu 5
5 ret
```



1. LOOP: CX = 3 ≠ 0 → atlar (aynı komuta döner): **17** tur. Toplam 21.

Adım 2 / 6

DEĞİŞKENLER

`CX = 0003h` `tur = 21`

Tur sayıları 8086 komut sürelerinden; yorumlayıcı yalnız CX ve atlama davranışını doğrular (emu8086 zamanlamayı modellemez).

Gecikme Döngüsünde Saat Turlarını Saymak

```
1 org 100h
2 mov cx, 4       ; 4 saat turu
3 bekle:
▶ loop bekle      ; 17 tur, sonuncu 5
5 ret
```



2. LOOP: CX = 2, yine 17 tur. Toplam 38.

Adım 3 / 6

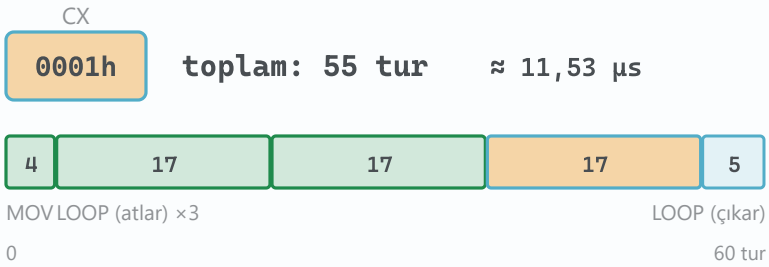
DEĞİŞKENLER

`CX = 0002h` `tur = 38`

Tur sayıları 8086 komut sürelerinden; yorumlayıcı yalnız CX ve atlama davranışını doğrular (emu8086 zamanlamayı modellemez).

Gecikme Döngüsünde Saat Turlarını Saymak

```
1 org 100h
2 mov cx, 4       ; 4 saat turu
3 bekle:
▶ loop bekle      ; 17 tur, sonuncu 5
5 ret
```



3. LOOP: CX = 1, 17 tur. Toplam 55.

Adım 4 / 6

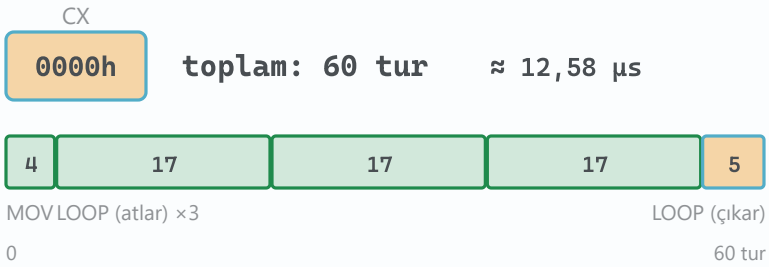
DEĞİŞKENLER

`CX = 0001h` `tur = 55`

Tur sayıları 8086 komut sürelerinden; yorumlayıcı yalnız CX ve atlama davranışını doğrular (emu8086 zamanlamayı modellemez).

Gecikme Döngüsünde Saat Turlarını Saymak

```
1 org 100h
2 mov cx, 4       ; 4 saat turu
3 bekle:
▶ loop bekle      ; 17 tur, sonuncu 5
5 ret
```



4. LOOP: CX = 0 → atlamaz: yalnız **5** tur. Toplam **60** = 17 · 4 − 8 tur.

Adım 5 / 6

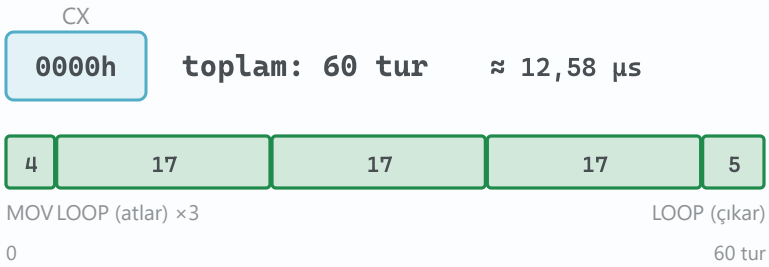
DEĞİŞKENLER

`CX = 0000h` `tur = 60`

Tur sayıları 8086 komut sürelerinden; yorumlayıcı yalnız CX ve atlama davranışını doğrular (emu8086 zamanlamayı modellemez).

Gecikme Döngüsünde Saat Turlarını Saymak

```
1 org 100h
2 mov cx, 4       ; 4 saat turu
3 bekle:
4 loop bekle      ; 17 tur, sonuncu 5
▶ ret
```



60 tur ÷ 4,77 MHz ≈ **12,6 µs**. CX'i N yapınca süre 17N − 8 turdur; 1 ms için N = 281 bulunmuştu.

Adım 6 / 6

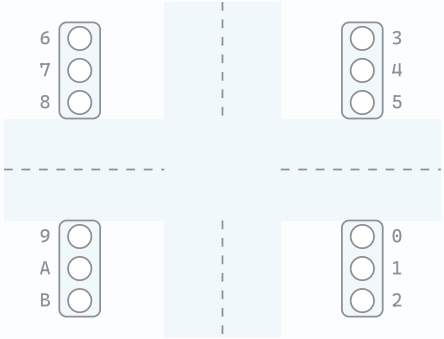
DEĞİŞKENLER

`CX = 0000h` `tur = 60`

Tur sayıları 8086 komut sürelerinden; yorumlayıcı yalnız CX ve atlama davranışını doğrular (emu8086 zamanlamayı modellemez).

Trafik Işıkları Programı

```
#start=Traffic_Lights.exe#
mov ax, all_red
out 4, ax
mov si, offset situation
next:
mov ax, [si]
out 4, ax
mov cx, 4Ch       ; 5 s bekle
mov dx, 4B40h
mov ah, 86h
int 15h
add si, 2         ; sonraki durum
cmp si, sit_end
jb  next
mov si, offset situation
jmp next
```



Tablo 0124h'de varsayıldı (ORG 100h; adres derleyiciye göre değişebilir). sit_end = 012Eh.

Adım 1 / 22

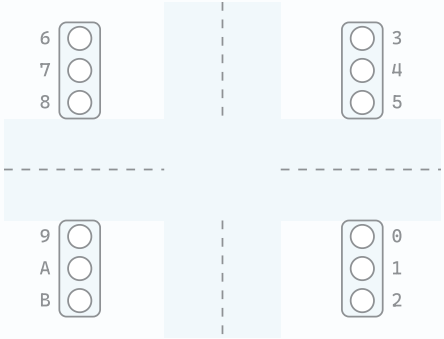
DEĞİŞKENLER

AX = **0000h** SI = **0000h** CX = **0000h** DX = **0000h** CF = **0**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



AX = all_red = 0249h (sabit, EQU).

Adım 2 / 22

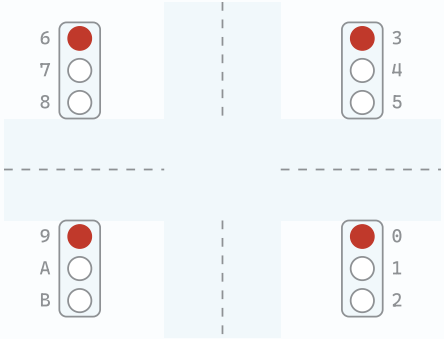
DEĞİŞKENLER

AX = 0249h | SI = 0000h | CX = 0000h | DX = 0000h | CF = 0

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
▶  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



Port 4'e yaz: tüm yönler **kırmızı**.

Adım 3 / 22

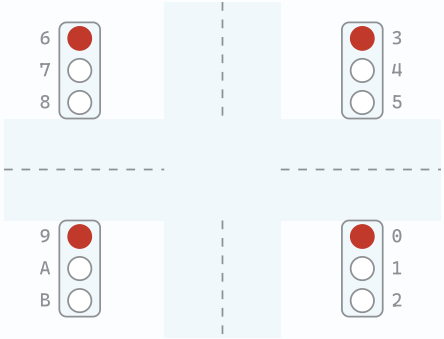
DEĞİŞKENLER

`AX = 0249h` `SI = 0000h` `CX = 0000h` `DX = 0000h` `CF = 0`

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



SI = 0124h: tablonun başı (situation).

Adım 4 / 22

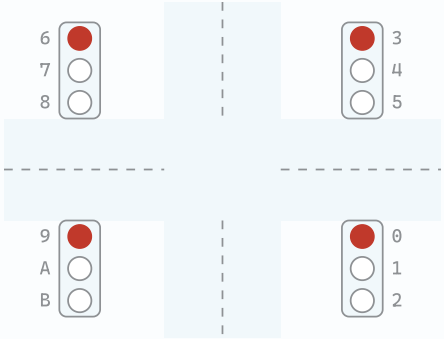
DEĞİŞKENLER

AX = **0249h** | SI = **0124h** | CX = **0000h** | DX = **0000h** | CF = **0**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
▶  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



AX ← [SI] = 030Ch.

Adım 5 / 22

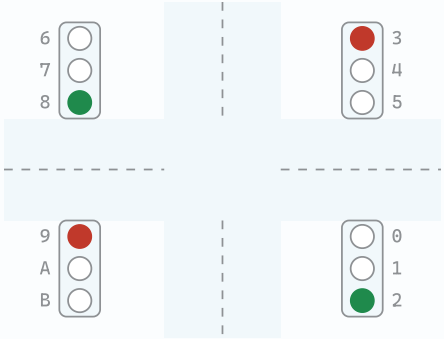
DEĞİŞKENLER

AX = **030Ch** | SI = **0124h** | CX = **0000h** | DX = **0000h** | CF = **0**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



OUT 4, AX: lambalar yeni duruma geçer (SI = 0124h).

Adım 6 / 22

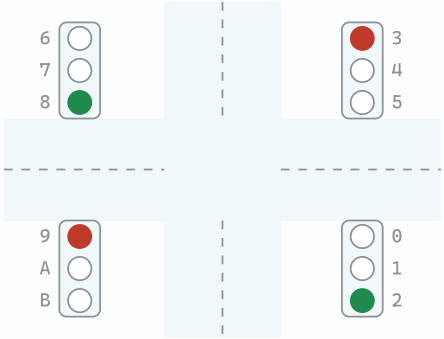
DEĞİŞKENLER

AX = **030Ch** SI = **0124h** CX = **0000h** DX = **0000h** CF = **0**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
▶  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0011 0000 1100

CX = 004Ch (bekleme süresinin yüksek sözcüğü).

Adım 7 / 22

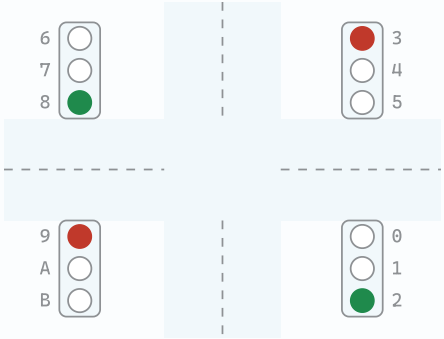
DEĞİŞKENLER

AX = **030Ch** | SI = **0124h** | CX = **004Ch** | DX = **0000h** | CF = **0**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0011 0000 1100

DX = 4B40h (düşük sözcük).

Adım 8 / 22

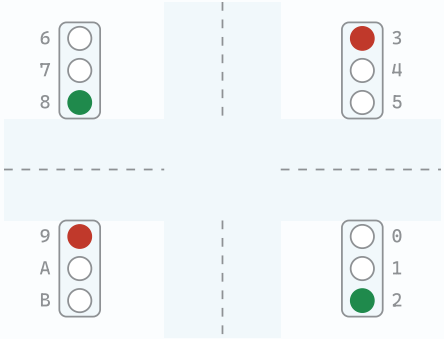
DEĞİŞKENLER

AX = **030Ch** | SI = **0124h** | CX = **004Ch** | DX = **4B40h** | CF = **0**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
▶  mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0011 0000 1100

AH = 86h: BIOS bekleme işlevi.

Adım 9 / 22

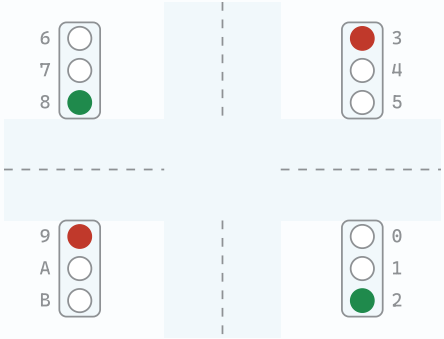
DEĞİŞKENLER

AX = 860Ch | SI = 0124h | CX = 004Ch | DX = 4B40h | CF = 0

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
▶ int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



INT 15h: 5 saniye bekle.

Adım 10 / 22

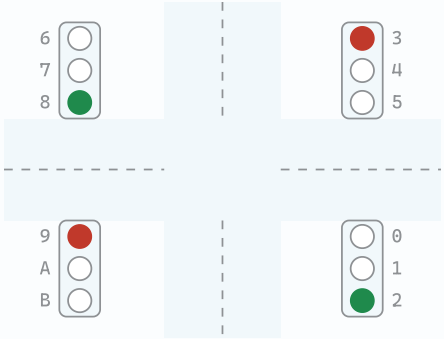
DEĞİŞKENLER

AX = 860Ch | SI = 0124h | CX = 004Ch | DX = 4B40h | CF = 0

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
▶ add si, 2          ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



SI = 0126h: sonraki sözcük.

Adım 11 / 22

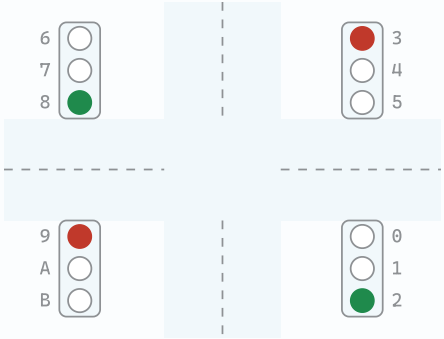
DEĞİŞKENLER

AX = 860Ch | SI = 0126h | CX = 004Ch | DX = 4B40h | CF = 0

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
▶ cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0011 0000 1100

CMP SI, sit_end (012Eh): SI < sit_end → CF = 1.

Adım 12 / 22

DEĞİŞKENLER

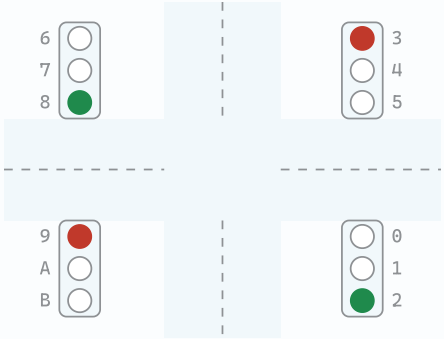
AX = **860Ch** SI = **0126h** CX = **004Ch** DX = **4B40h**

CF = **1**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
▶ jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0011 0000 1100

JB: CF = 1, next'e dön.

Adım 13 / 22

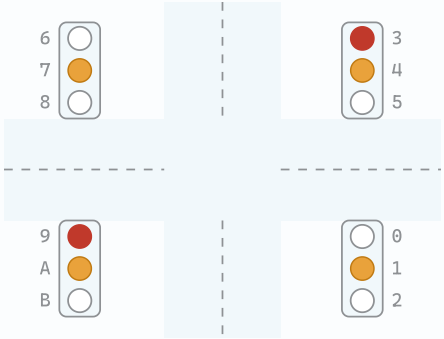
DEĞİŞKENLER

AX = **860Ch** SI = **0126h** CX = **004Ch** DX = **4B40h** CF = **1**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
▶  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0110 1001 1010

... bekleme ve SI artırma ... **OUT 4, AX**: SI = 0126h → port 4 = 069Ah.

Adım 14 / 22

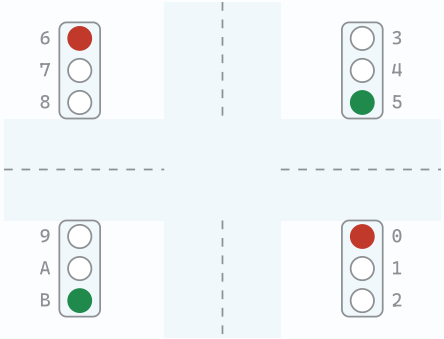
DEĞİŞKENLER

AX = **069Ah** | SI = **0126h** | CX = **004Ch** | DX = **4B40h** | CF = **1**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
▶  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 1000 0110 0001

… bekleme ve SI artırma … **OUT 4, AX**: SI = 0128h → port 4 = 0861h.

Adım 15 / 22

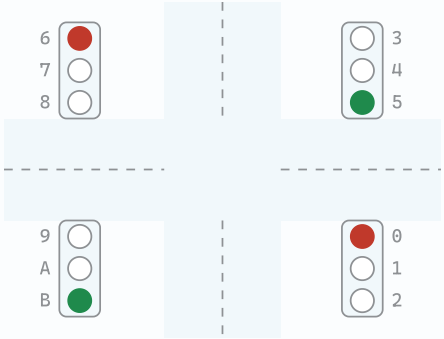
DEĞİŞKENLER

AX = **0861h** SI = **0128h** CX = **004Ch** DX = **4B40h** CF = **1**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
▶  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 1000 0110 0001

... bekleme ve SI artırma ... **OUT 4, AX**: SI = 012Ah → port 4 = 0861h.

Adım 16 / 22

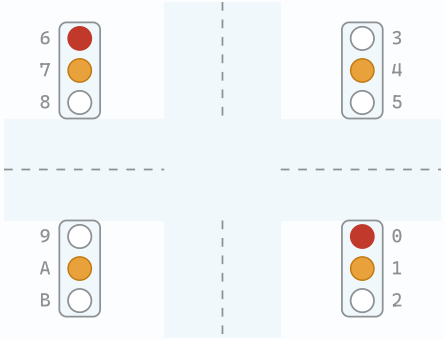
DEĞİŞKENLER

AX = **0861h** | SI = **012Ah** | CX = **004Ch** | DX = **4B40h** | CF = **1**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
▶  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0100 1101 0011

... bekleme ve SI artırma ... **OUT 4, AX**: SI = 012Ch → port 4 = 04D3h.

Adım 17 / 22

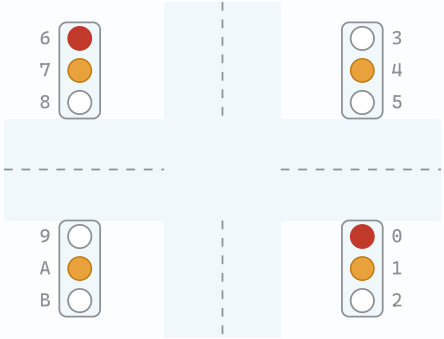
DEĞİŞKENLER

AX = **04D3h** | SI = **012Ch** | CX = **004Ch** | DX = **4B40h** | CF = **1**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
   cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0100 1101 0011

CMP SI, sit_end (012Eh): SI = sit_end → CF = 0.

Adım 18 / 22

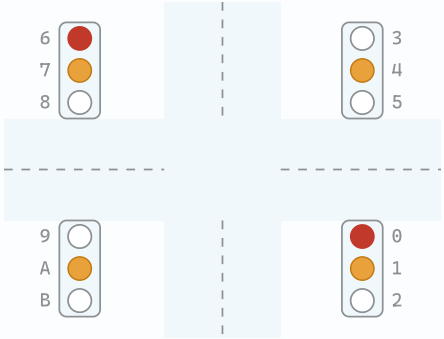
DEĞİŞKENLER

AX = **86D3h** SI = **012Eh** CX = **004Ch** DX = **4B40h** CF = **0**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



FEDC BA98 7654 3210
0000 0100 1101 0011

JB: CF = 0, atlama yok: tablo bitti.

Adım 19 / 22

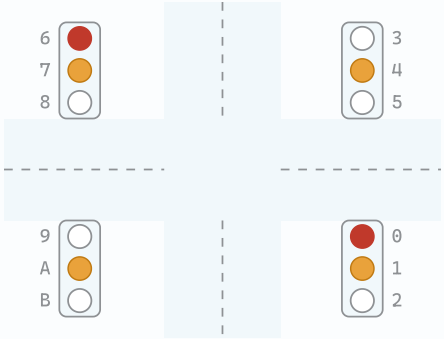
DEĞİŞKENLER

AX = **86D3h** SI = **012Eh** CX = **004Ch** DX = **4B40h** CF = **0**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
   mov si, offset situation
16 jmp next
```



SI = 0124h: tabloya baştan başla.

Adım 20 / 22

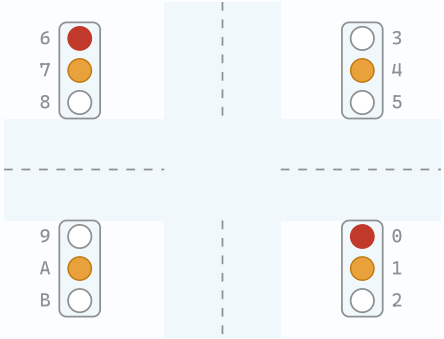
DEĞİŞKENLER

AX = 86D3h | SI = 0124h | CX = 004Ch | DX = 4B40h | CF = 0

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
7  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
▶  jmp next
```



JMP next: sonsuz döngü.

Adım 21 / 22

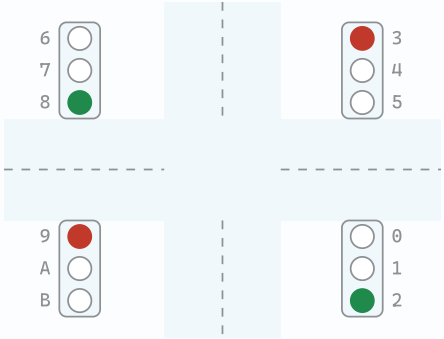
DEĞİŞKENLER

AX = **86D3h** SI = **0124h** CX = **004Ch** DX = **4B40h** CF = **0**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

Trafik Işıkları Programı

```
1  #start=Traffic_Lights.exe#
2  mov ax, all_red
3  out 4, ax
4  mov si, offset situation
5  next:
6  mov ax, [si]
▶  out 4, ax
8  mov cx, 4Ch       ; 5 s bekle
9  mov dx, 4B40h
10 mov ah, 86h
11 int 15h
12 add si, 2         ; sonraki durum
13 cmp si, sit_end
14 jb  next
15 mov si, offset situation
16 jmp next
```



... bekleme ve SI artırma ... **OUT 4, AX**: SI = 0124h → port 4 = 030Ch.

Adım 22 / 22

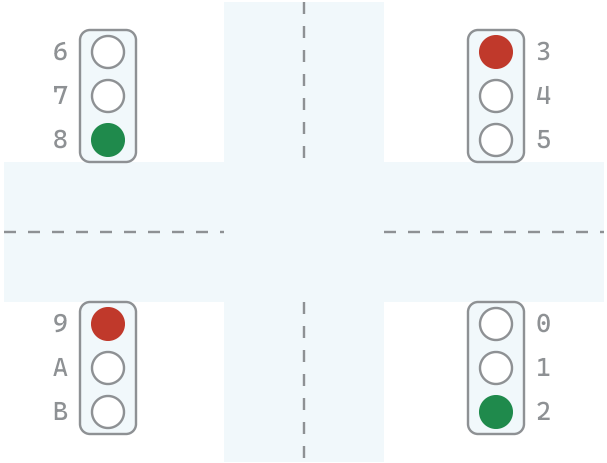
DEĞİŞKENLER

AX = **030Ch** | SI = **0124h** | CX = **004Ch** | DX = **4B40h** | CF = **0**

Döngü hiç bitmez: gömülü sistemlerde ana program genellikle sonsuz bir döngüdür.

TRAFİK IŞIKLARI

Etkileşimli Trafik Bit Düzenleyici



Bir bite tıklanınca o lamba yanar ya da söner (F...0). Bit 12–15 kullanılmaz.

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0

all_red **situation** **s1** **s2 / s3** **s4**

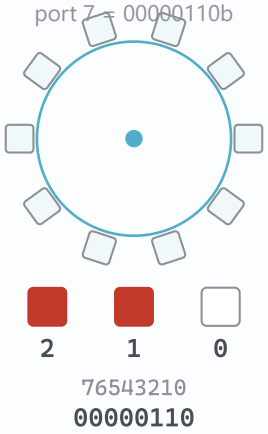
0000h

```
mov ax, 0000_0011_0000_1100b   ; 030Ch
out 4, ax
```

sağ alt: **yeşil** · sağ üst: **kırmızı** · sol üst: **yeşil** · sol alt: **kırmızı**

Step Motor (Port 7)

- Motor hem **saat yönünde** (CW) hem de **tersine** (CCW) döndürülebilir.
- Port 7'ye yazılan baytın **bit 0, 1, 2**'si üç elektromıknatısı açar/kapatır.
- **Yarı adım** 11,25°, **tam adım** 22,5°'dir.
- Yarı adım: çift mıknatıs, ardından tek mıknatıs açılarak döndürülür.
- Tam adım: tek, çift, çift mıknatıs ve “hepsi kapalı” bayt sırayla gönderilir.
- Motor yeni komut almaya hazır olduğunda porttan okunan baytın **bit 7**'sini 1 yapar.



Step Motor Veri Tabloları

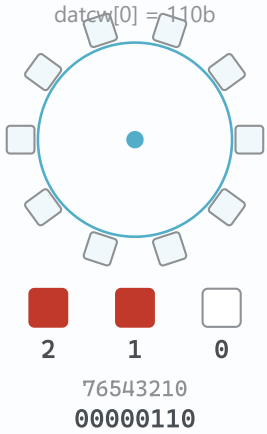
Tablo	Anlam	1	2	3	4
`datcw`	CW yarı adım	110	100	011	010
`datccw`	CCW yarı adım	011	001	110	010
`datcw_fs`	CW tam adım	001	011	110	000
`datccw_fs`	CCW tam adım	100	110	011	000



- Tablolar bellekte arka arkaya durur: her biri **4 bayt**. Bu yüzden `add bx, 4` bir sonraki tabloya geçirir.
- Program 32 (20h) tur sonra tabloyu değiştirir: CW yarı → CCW yarı → CW tam → CCW tam → başa.
- Bayt değerindeki bitler mıknatısları gösterir: `110b` = mıknatıs 2 ve 1 açık; `000b` = hepsi kapalı.
- Kaynak koddaki biçim: `datcw db 0000_0110b` ve alt satırlarda `db 0000_0100b` … (alt çizgi yalnızca okunabilirlik içindir).

Mıknatıs Dizileri Adım Adım

- Yarı adım dizisi (datcw): **çift** → **tek** → **çift** → **tek** mıknatıs.
- Tam adım dizisi (datcw_fs): tek, çift, çift ve “hepsi kapalı” bayt.
- Her OUT komutundan sonra motor yeni konumuna gidene kadar **meşgul**dur; bu yüzden program her bayttan önce bit 7'yi yoklar.

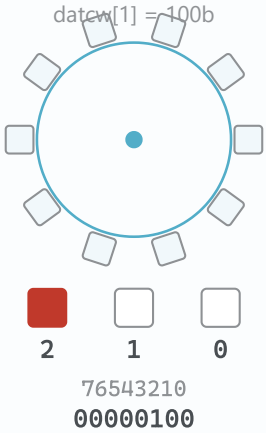


CW yarı adım, 1. bayt: açık mıknatıslar → 2, 1

Adım 1 / 8

Mıknatıs Dizileri Adım Adım

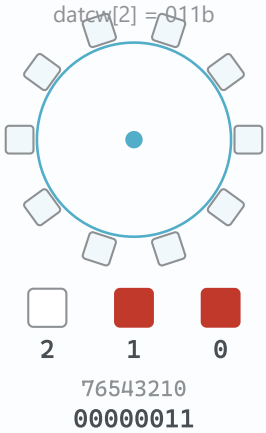
- Yarı adım dizisi (datcw): **çift** → **tek** → **çift** → **tek** mıknatıs.
- Tam adım dizisi (datcw_fs): tek, çift, çift ve “hepsi kapalı” bayt.
- Her OUT komutundan sonra motor yeni konumuna gidene kadar **meşgul**dur; bu yüzden program her bayttan önce bit 7'yi yoklar.



CW yarı adım, 2. bayt: açık mıknatıslar → 2

Mıknatıs Dizileri Adım Adım

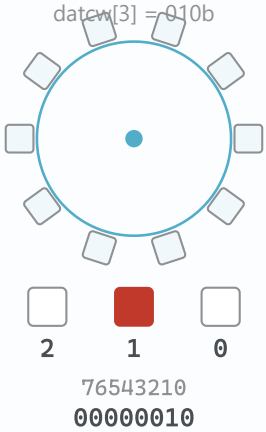
- Yarı adım dizisi (datcw): **çift** → **tek** → **çift** → **tek** mıknatıs.
- Tam adım dizisi (datcw_fs): tek, çift, çift ve “hepsi kapalı” bayt.
- Her OUT komutundan sonra motor yeni konumuna gidene kadar **meşgul**dur; bu yüzden program her bayttan önce bit 7'yi yoklar.



CW yarı adım, 3. bayt: açık mıknatıslar → 1, 0

Mıknatıs Dizileri Adım Adım

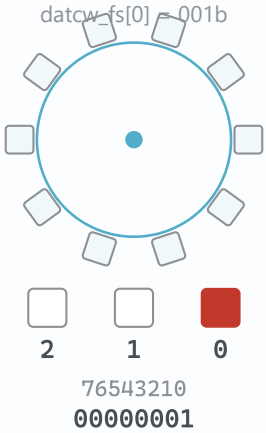
- Yarı adım dizisi (datcw): **çift** → **tek** → **çift** → **tek** mıknatıs.
- Tam adım dizisi (datcw_fs): tek, çift, çift ve “hepsi kapalı” bayt.
- Her OUT komutundan sonra motor yeni konumuna gidene kadar **meşgul**dur; bu yüzden program her bayttan önce bit 7'yi yoklar.



CW yarı adım, 4. bayt: açık mıknatıslar → 1

Mıknatıs Dizileri Adım Adım

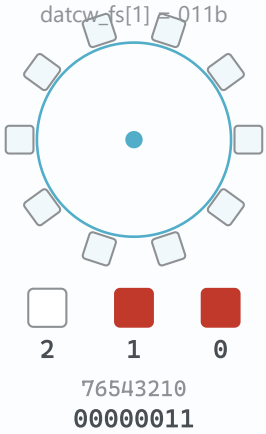
- Yarı adım dizisi (datcw): **çift** → **tek** → **çift** → **tek** mıknatıs.
- Tam adım dizisi (datcw_fs): tek, çift, çift ve “hepsi kapalı” bayt.
- Her OUT komutundan sonra motor yeni konumuna gidene kadar **meşgul**dur; bu yüzden program her bayttan önce bit 7'yi yoklar.



CW tam adım, 1. bayt: açık mıknatıslar → 0

Mıknatıs Dizileri Adım Adım

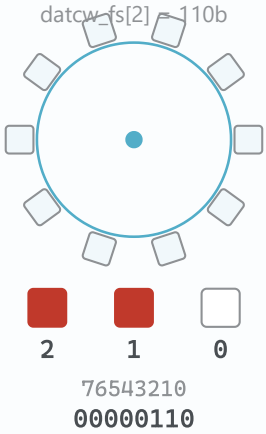
- Yarı adım dizisi (datcw): **çift** → **tek** → **çift** → **tek** mıknatıs.
- Tam adım dizisi (datcw_fs): tek, çift, çift ve “hepsi kapalı” bayt.
- Her OUT komutundan sonra motor yeni konumuna gidene kadar **meşgul**dur; bu yüzden program her bayttan önce bit 7'yi yoklar.



CW tam adım, 2. bayt: açık mıknatıslar → 1, 0

Mıknatıs Dizileri Adım Adım

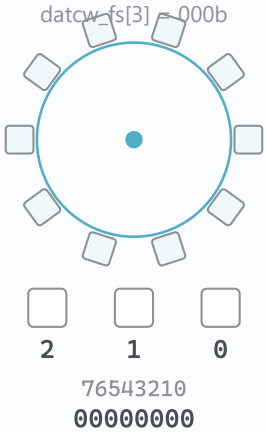
- Yarı adım dizisi (datcw): **çift** → **tek** → **çift** → **tek** mıknatıs.
- Tam adım dizisi (datcw_fs): tek, çift, çift ve “hepsi kapalı” bayt.
- Her OUT komutundan sonra motor yeni konumuna gidene kadar **meşgul**dur; bu yüzden program her bayttan önce bit 7'yi yoklar.



CW tam adım, 3. bayt: açık mıknatıslar → 2, 1

Mıknatıs Dizileri Adım Adım

- Yarı adım dizisi (datcw): **çift** → **tek** → **çift** → **tek** mıknatıs.
- Tam adım dizisi (datcw_fs): tek, çift, çift ve “hepsi kapalı” bayt.
- Her OUT komutundan sonra motor yeni konumuna gidene kadar **meşgul**dur; bu yüzden program her bayttan önce bit 7'yi yoklar.



CW tam adım, 4. bayt: açık mıknatıslar → hiçbiri

Step Motor Kontrol Programı

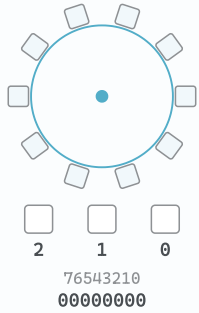
```
ORG 100h
steps_before_direction_change = 20h
jmp start
datcw     db 110b,100b,011b,010b
datccw    db 011b,001b,110b,010b
datcw_fs  db 001b,011b,110b,000b
datccw_fs db 100b,110b,011b,000b
start:
mov bx, offset datcw
mov si, 0
mov cx, 0   ; tur sayacı
```

- **Yoklama** (wait): bit 7 = 0 ise motor meşgul, tekrar oku.
- `[bx][si]` = [BX + SI]: BX tablonun başı, SI sıra (0–3).
- 4 bayt gönderilince SI = 0, CX +1; CX = 20h olunca BX 4 artar, son tablodan sonra başa döner.
- `steps_before_direction_change = 20h` bir **sabittir** (EQU gibi), bellekte yer kaplamaz.

```
next_step:
wait: in al, 7
test al, 10000000b
jz wait
mov al, [bx][si]
out 7, al
inc si
cmp si, 4
jb next_step
mov si, 0
inc cx
cmp cx, steps_before_direction_change
jb  next_step
mov cx, 0
add bx, 4   ; sonraki tablo
cmp bx, offset datccw_fs
jbe next_step
mov bx, offset datcw
jmp next_step
```

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



ORG 100h, steps_before_direction_change = 20h. Port 7 önce 00h (meşgul), sonra 80h (hazır) döndürecek.

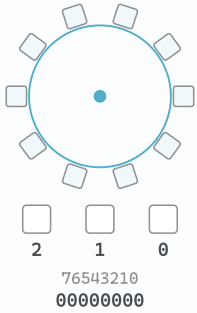
Adım 1 / 44

DEĞİŞKENLER

AX = 0000h | BX = 0000h | SI = 0000h | CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



Veri tablolarının üzerinden atla.

Adım 2 / 44

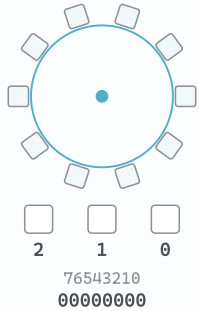
DEĞİŞKENLER

AX = 0000h | BX = 0000h | SI = 0000h

CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



BX = 0102h: datcw tablosu (CW yarı adım).

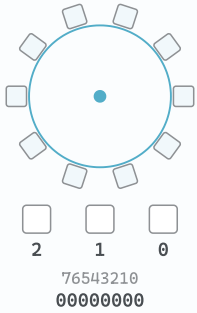
Adım 3 / 44

DEĞİŞKENLER

AX = 0000h | BX = 0102h | SI = 0000h | CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
▶  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



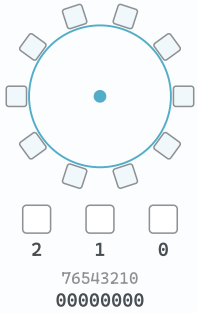
SI = 0: tablonun ilk baytı.

DEĞİŞKENLER

AX = 0000h | BX = 0102h | SI = 0000h | CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
▶  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



CX = 0: tur sayacı.

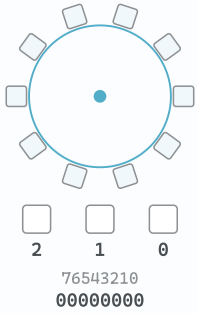
Adım 5 / 44

DEĞİŞKENLER

AX = 0000h | BX = 0102h | SI = 0000h | CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
▶  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



IN AL, 7 → AL = 00h (bit 7 = 0, motor meşgul).

Adım 6 / 44

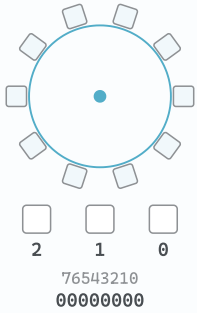
DEĞİŞKENLER

AX = **0000h** | BX = **0102h** | SI = **0000h**

CX = **0000h** | ZF = **0** | CF = **0**

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
▶  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



TEST AL, 80h → ZF = 1.

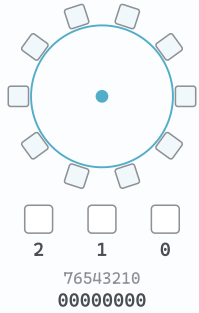
Adım 7 / 44

DEĞİŞKENLER

AX = 0000h | BX = 0102h | SI = 0000h | CX = 0000h | ZF = 1 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
▶  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JZ: ZF = 1 → tekrar bekle.

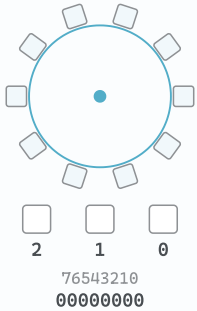
Adım 8 / 44

DEĞİŞKENLER

AX = 0000h | BX = 0102h | SI = 0000h | CX = 0000h | ZF = 1 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
▶  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



IN AL, 7 → AL = 80h (bit 7 = 1, hazır).

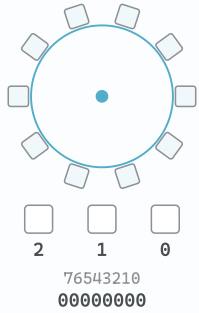
Adım 9 / 44

DEĞİŞKENLER

AX = 0080h | BX = 0102h | SI = 0000h | CX = 0000h | ZF = 1 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
▶  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



TEST AL, 80h → ZF = 0.

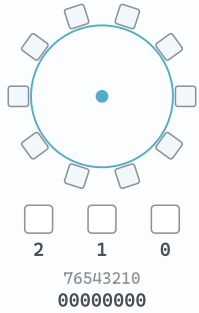
Adım 10 / 44

DEĞİŞKENLER

AX = 0080h | BX = 0102h | SI = 0000h | CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
▶  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JZ: ZF = 0 → devam.

Adım 11 / 44

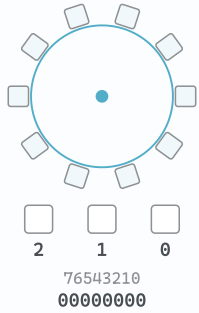
DEĞİŞKENLER

AX = **0080h** | BX = **0102h** | SI = **0000h**

CX = **0000h** | ZF = **0** | CF = **0**

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
▶  mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



AL ← [BX+SI] = [0102h] = 110b.

Adım 12 / 44

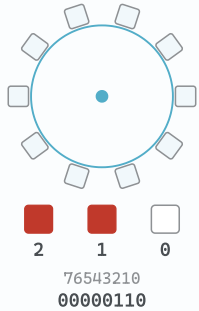
DEĞİŞKENLER

AX = 0006h | BX = 0102h | SI = 0000h

CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
▶  out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



OUT 7, AL: mıknatıslar 110b → motor yarım adım (11,25°) döner.

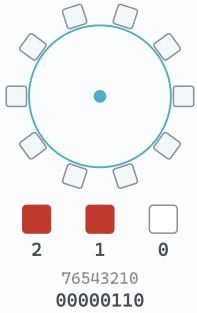
DEĞİŞKENLER

AX = **0006h** | BX = **0102h** | SI = **0000h**

CX = **0000h** | ZF = **0** | CF = **0**

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
▶ 12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



SI = 1.

Adım 14 / 44

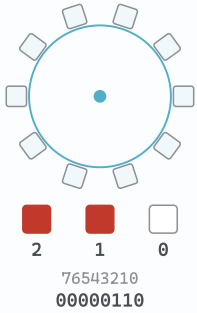
DEĞİŞKENLER

AX = 0006h | BX = 0102h | SI = 0001h

CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
▶ cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



CMP SI, 4 → CF = 1.

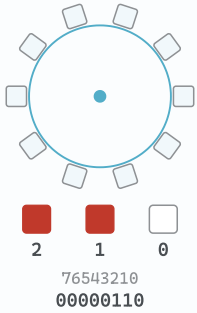
Adım 15 / 44

DEĞİŞKENLER

AX = 0006h | BX = 0102h | SI = 0001h | CX = 0000h | ZF = 0 | CF = 1

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
▶ jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JB: SI < 4 → next_step.

Adım 16 / 44

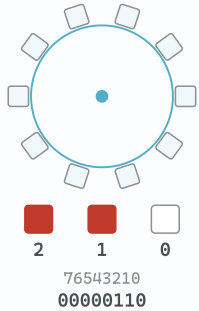
DEĞİŞKENLER

AX = **0006h** | BX = **0102h** | SI = **0001h**

CX = **0000h** | ZF = **0** | CF = **1**

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
▶  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



IN AL, 7 → AL = 80h (bit 7 = 1, hazır).

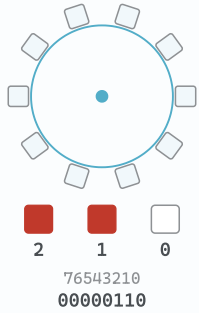
Adım 17 / 44

DEĞİŞKENLER

AX = 0080h | BX = 0102h | SI = 0001h | CX = 0000h | ZF = 0 | CF = 1

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
▶  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



TEST AL, 80h → ZF = 0.

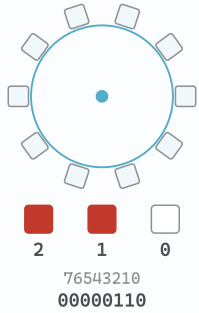
Adım 18 / 44

DEĞİŞKENLER

AX = 0080h | BX = 0102h | SI = 0001h | CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
▶  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JZ: ZF = 0 → devam.

Adım 19 / 44

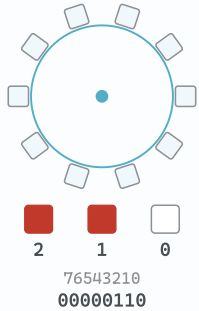
DEĞİŞKENLER

AX = **0080h** | BX = **0102h** | SI = **0001h**

CX = **0000h** | ZF = **0** | CF = **0**

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
▶  mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



AL ← [BX+SI] = [0103h] = 100b.

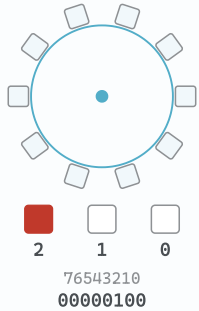
Adım 20 / 44

DEĞİŞKENLER

AX = 0004h | BX = 0102h | SI = 0001h | CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
▶  out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



OUT 7, AL: mıknatıslar 100b → motor yarım adım (11,25°) döner.

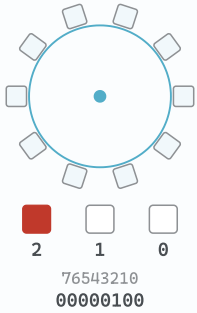
DEĞİŞKENLER

AX = **0004h** | BX = **0102h** | SI = **0001h**

CX = **0000h** | ZF = **0** | CF = **0**

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
▶ inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



SI = 2.

Adım 22 / 44

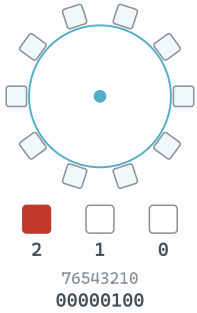
DEĞİŞKENLER

AX = 0004h | BX = 0102h | SI = 0002h

CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
▶ cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



CMP SI, 4 → CF = 1.

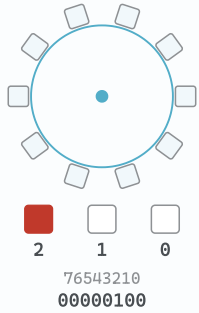
Adım 23 / 44

DEĞİŞKENLER

AX = 0004h | BX = 0102h | SI = 0002h | CX = 0000h | ZF = 0 | CF = 1

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
▶ jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JB: SI < 4 → next_step.

Adım 24 / 44

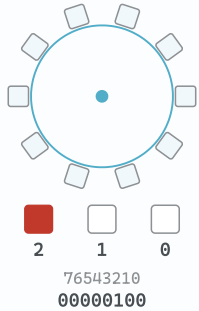
DEĞİŞKENLER

AX = 0004h | BX = 0102h | SI = 0002h

CX = 0000h | ZF = 0 | CF = 1

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
▶  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



IN AL, 7 → AL = 80h (bit 7 = 1, hazır).

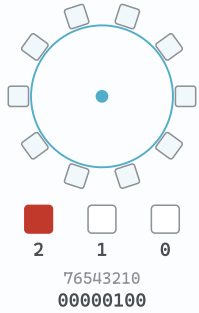
Adım 25 / 44

DEĞİŞKENLER

AX = 0080h | BX = 0102h | SI = 0002h | CX = 0000h | ZF = 0 | CF = 1

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
▶  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



TEST AL, 80h → ZF = 0.

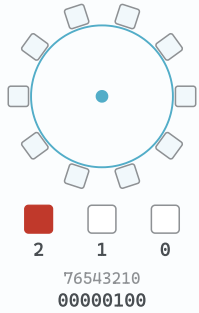
Adım 26 / 44

DEĞİŞKENLER

AX = 0080h | BX = 0102h | SI = 0002h | CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
▶  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JZ: ZF = 0 → devam.

Adım 27 / 44

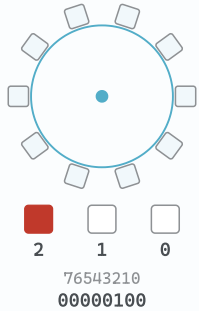
DEĞİŞKENLER

AX = **0080h** | BX = **0102h** | SI = **0002h**

CX = **0000h** | ZF = **0** | CF = **0**

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
▶  mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



AL ← [BX+SI] = [0104h] = 011b.

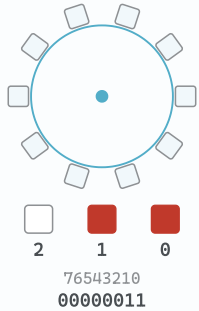
Adım 28 / 44

DEĞİŞKENLER

AX = 0003h | BX = 0102h | SI = 0002h | CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
▶  out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



OUT 7, AL: mıknatıslar 011b → motor yarım adım (11,25°) döner.

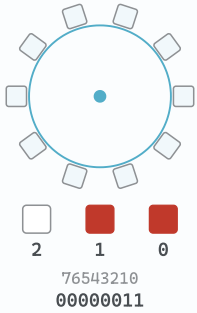
DEĞİŞKENLER

AX = **0003h** | BX = **0102h** | SI = **0002h**

CX = **0000h** | ZF = **0** | CF = **0**

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
▶ inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



SI = 3.

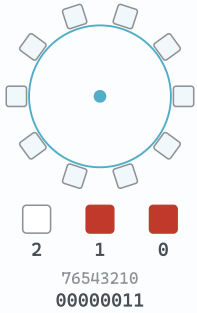
Adım 30 / 44

DEĞİŞKENLER

AX = 0003h | BX = 0102h | SI = 0003h | CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
▶ cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



CMP SI, 4 → CF = 1.

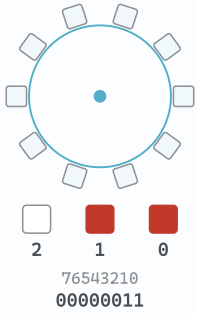
Adım 31 / 44

DEĞİŞKENLER

AX = 0003h | BX = 0102h | SI = 0003h | CX = 0000h | ZF = 0 | CF = 1

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
▶ jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JB: SI < 4 → next_step.

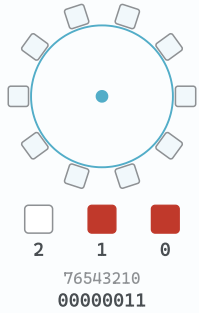
Adım 32 / 44

DEĞİŞKENLER

AX = 0003h | BX = 0102h | SI = 0003h | CX = 0000h | ZF = 0 | CF = 1

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
▶  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



IN AL, 7 → AL = 80h (bit 7 = 1, hazır).

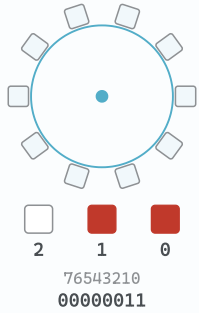
Adım 33 / 44

DEĞİŞKENLER

AX = 0080h | BX = 0102h | SI = 0003h | CX = 0000h | ZF = 0 | CF = 1

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
▶  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



TEST AL, 80h → ZF = 0.

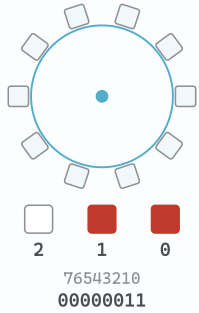
Adım 34 / 44

DEĞİŞKENLER

AX = 0080h | BX = 0102h | SI = 0003h | CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
▶  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JZ: ZF = 0 → devam.

Adım 35 / 44

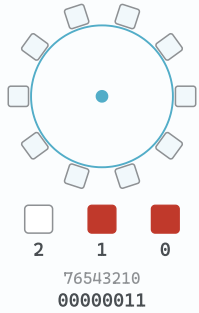
DEĞİŞKENLER

AX = **0080h** | BX = **0102h** | SI = **0003h**

CX = **0000h** | ZF = **0** | CF = **0**

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
▶  mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



AL ← [BX+SI] = [0105h] = 010b.

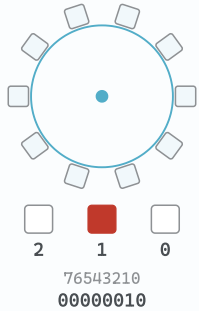
Adım 36 / 44

DEĞİŞKENLER

AX = 0002h | BX = 0102h | SI = 0003h | CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
▶  out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



OUT 7, AL: mıknatıslar 010b → motor yarım adım (11,25°) döner.

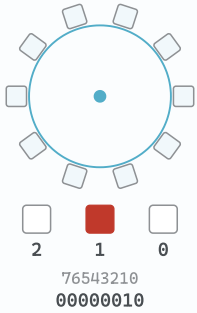
DEĞİŞKENLER

AX = **0002h** | BX = **0102h** | SI = **0003h**

CX = **0000h** | ZF = **0** | CF = **0**

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
▶ inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



SI = 4.

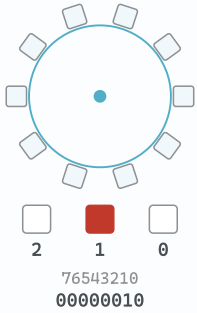
Adım 38 / 44

DEĞİŞKENLER

AX = 0002h | BX = 0102h | SI = 0004h | CX = 0000h | ZF = 0 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
▶  cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



CMP SI, 4 → CF = 0.

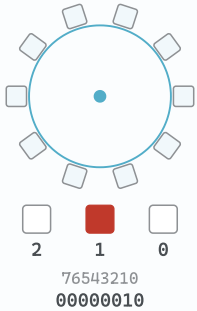
Adım 39 / 44

DEĞİŞKENLER

AX = 0002h | BX = 0102h | SI = 0004h | CX = 0000h | ZF = 1 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
▶ jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



JB: SI = 4: tablo bitti.

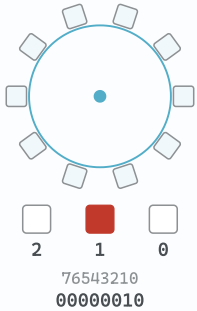
Adım 40 / 44

DEĞİŞKENLER

AX = 0002h | BX = 0102h | SI = 0004h | CX = 0000h | ZF = 1 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
▶ mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



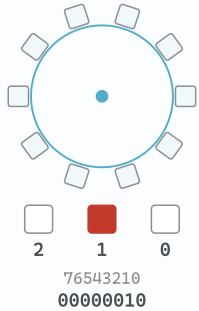
SI = 0: tabloyu baştan oku.

DEĞİŞKENLER

AX = 0002h | BX = 0102h | SI = 0000h | CX = 0000h | ZF = 1 | CF = 0

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
▶ inc cx
17 cmp cx, steps_before_direction_change
18 jb  next_step
```



CX = 1: bir tur (4 adım) tamamlandı.

Adım 42 / 44

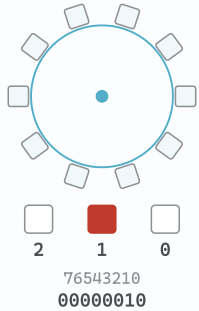
DEĞİŞKENLER

AX = **0002h** | BX = **0102h** | SI = **0000h**

CX = **0001h** | ZF = **0** | CF = **0**

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
▶ cmp cx, steps_before_direction_change
18 jb  next_step
```



CMP CX, 20h → CF = 1 (henüz 32 tur olmadı).

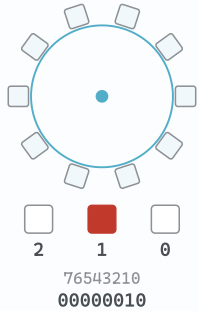
Adım 43 / 44

DEĞİŞKENLER

AX = 0002h | BX = 0102h | SI = 0000h | CX = 0001h | ZF = 0 | CF = 1

Step Motor: İlk Tur

```
1  jmp start   ; tablolar 0102h-0111h
2  start:
3  mov bx, offset datcw
4  mov si, 0
5  mov cx, 0   ; tur sayacı
6  next_step:
7  wait: in al, 7
8  test al, 10000000b
9  jz wait
10 mov al, [bx][si]
11 out 7, al
12 inc si
13 cmp si, 4
14 jb next_step
15 mov si, 0
16 inc cx
17 cmp cx, steps_before_direction_change
▶  jb  next_step
```



JB next_step: aynı tabloyla devam. 32 turdan sonra `add bx, 4` sonraki tabloya geçer.

Adım 44 / 44

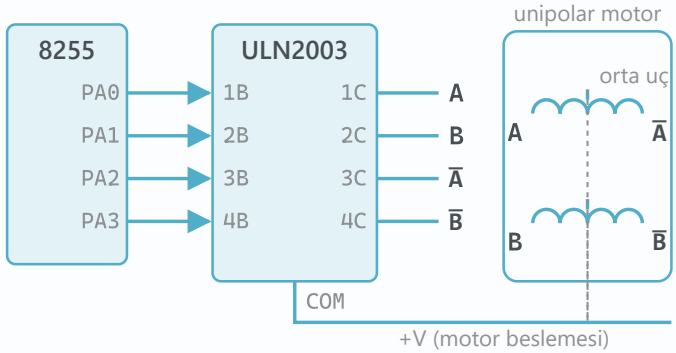
DEĞİŞKENLER

AX = **0002h** | BX = **0102h** | SI = **0000h**

CX = **0001h** | ZF = **0** | CF = **1**

Gerçek Donanım: Unipolar Step Motor ve ULN2003

- Unipolar motorda iki **orta uçlu** sargı vardır: dört sargı yarısı A, B, $\bar{A}$, $\bar{B}$; orta uçlar +V'ye bağlanır (5 ya da 6 tel).
- Sargı akımı (yüzlerce mA) port ucunun sürebileceğinden çok büyüktür; araya **ULN2003** konur: 7 Darlington kanalı, kanal başına 500 mA'e kadar.
- Giriş = 1 → çıkış toprağa çeker → o sargı yarısından akım geçer.
- Akım kesilince sargıda oluşan ters gerilim, COM ucu +V'ye bağlanarak ULN2003'ün iç diyotlarıyla sönümlenir.
- Motor açık çevrimdir: “hazır” biti yoktur, hızı adımlar arası bekleme belirler. Çok kısa beklemede motor **adım kaçırır**.



Gerçek Motor: Uyarma Dizileri

Sürüş	Port A dizisi ($\bar{B}$ $\bar{A}$ B A = bit 3…0)
Tek faz (dalga), tam adım	01h 02h 04h 08h
İki faz, tam adım	03h 06h 0Ch 09h
Yarım adım	01h 03h 02h 06h 04h 0Ch 08h 09h

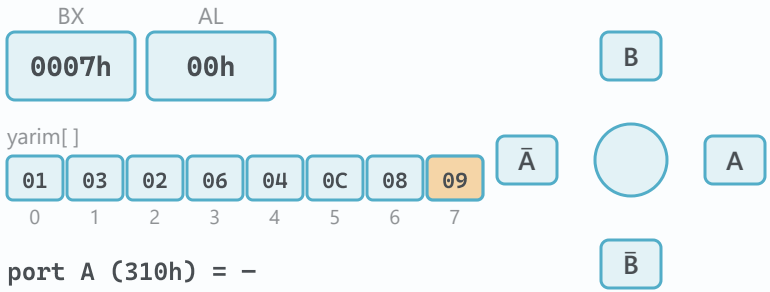
- İki faz sürüşte her an iki sargı yarısı akım çeker: tork daha yüksektir.
- Yarım adım iki diziyi sırayla karıştırır: adım açısı yarıya iner (ör. 1,8° → 0,9°).
- Dizi **tersten** uygulanırsa motor ters yönde döner.
- emu8086'nın sanal motoru 3 mıknatıslı basit bir modeldir (bit 0–2, bit 7 hazır); onun tabloları (110b, 100b, …) 4 fazlı gerçek motora uygulanmaz.

```
; 82h ile port A çıkış yapılmış olsun
yarim db 01h,03h,02h,06h
      db 04h,0Ch,08h,09h
adim  dw 0

ileri:
inc adim          ; geri için: dec adim
mov bx, adim
and bx, 7         ; dizin 0–7
mov al, yarim[bx]
mov dx, 310h      ; port A
out dx, al
call bekle        ; adım aralığı = hız
jmp ileri
```

Gerçek Motor: Yarım Adım Dizisini Sürmek

```
1  org 100h
2  ileri:
▶  inc adim          ; geri: dec adim
▶  mov bx, adim
▶  and bx, 7         ; dizin 0-7
6  mov al, yarim[bx]
7  mov dx, 310h      ; port A
8  out dx, al
9  call bekle        ; adım aralığı
10 jmp ileri
11 bekle: ret        ; (gecikme)
12 yarim db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 adim  dw 6
```



Sargılar: bit 0 = A, bit 1 = B, bit 2 = Ā, bit 3 = B̄. adim = 6 iken INC → **7**; AND BX, 7 dizini 0–7'de tutar → BX = 7.

Adım 1 / 8

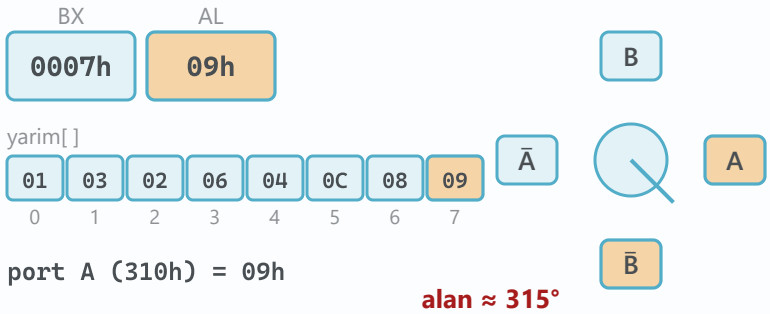
DEĞİŞKENLER

`BX = 0007h` `AL = 00h` `port310h = –`

Port 310h'e yazılan baytlar (09h, 01h, 03h …) yorumlayıcının G/Ç kaydından alındı; `bekle` burada yalnız RET'tir.

Gerçek Motor: Yarım Adım Dizisini Sürmek

```
1  org 100h
2  ileri:
3  inc adim          ; geri: dec adim
4  mov bx, adim
5  and bx, 7         ; dizin 0–7
▶  mov al, yarim[bx]
▶  mov dx, 310h      ; port A
▶  out dx, al
9  call bekle        ; adım aralığı
10 jmp ileri
11 bekle: ret        ; (gecikme)
12 yarim db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 adim  dw 6
```



AL ← yarim[7] = **09h** = 1001b → B̄ ve A sargıları: alan ikisinin arasına (≈ 315°) döner. Port A'ya yazılır.

Adım 2 / 8

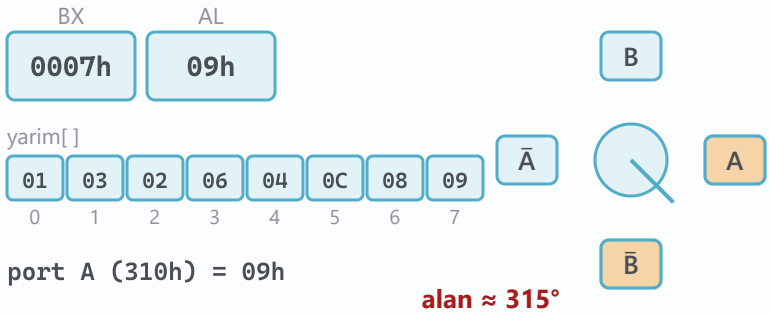
DEĞİŞKENLER

`BX = 0007h` `AL = 09h` `port310h = 09h`

Port 310h'e yazılan baytlar (09h, 01h, 03h …) yorumlayıcının G/Ç kaydından alındı; `bekle` burada yalnız RET'tir.

Gerçek Motor: Yarım Adım Dizisini Sürmek

```
1  org 100h
2  ileri:
3  inc adim          ; geri: dec adim
4  mov bx, adim
5  and bx, 7         ; dizin 0-7
6  mov al, yarim[bx]
7  mov dx, 310h      ; port A
8  out dx, al
▶  call bekle        ; adım aralığı
▶  jmp ileri
▶  bekle: ret        ; (gecikme)
12 yarim db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 adim  dw 6
```



CALL bekle / RET: adımlar arası bekleme motorun hızını belirler. JMP ileri.

Adım 3 / 8

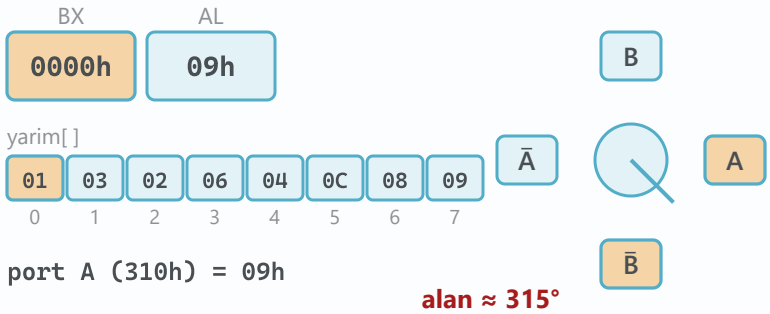
DEĞİŞKENLER

`BX = 0007h` `AL = 09h` `port310h = 09h`

Port 310h'e yazılan baytlar (09h, 01h, 03h …) yorumlayıcının G/Ç kaydından alındı; `bekle` burada yalnız RET'tir.

Gerçek Motor: Yarım Adım Dizisini Sürmek

```
1  org 100h
2  ileri:
▶  inc adim          ; geri: dec adim
▶  mov bx, adim
▶  and bx, 7         ; dizin 0-7
6  mov al, yarim[bx]
7  mov dx, 310h      ; port A
8  out dx, al
9  call bekle        ; adım aralığı
10 jmp ileri
11 bekle: ret        ; (gecikme)
12 yarim db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 adim  dw 6
```



adim = 8 → AND 7 → BX = **0**: dizi başa sarar.

Adım 4 / 8

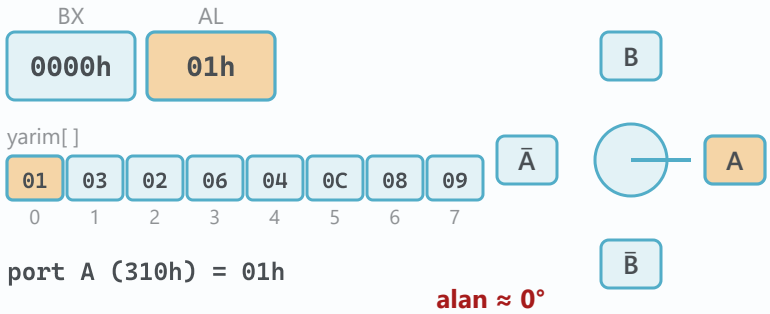
DEĞİŞKENLER

`BX = 0000h` `AL = 09h` `port310h = 09h`

Port 310h'e yazılan baytlar (09h, 01h, 03h …) yorumlayıcının G/Ç kaydından alındı; `bekle` burada yalnız RET'tir.

Gerçek Motor: Yarım Adım Dizisini Sürmek

```
1  org 100h
2  ileri:
3  inc adim          ; geri: dec adim
4  mov bx, adim
5  and bx, 7         ; dizin 0–7
▶  mov al, yarim[bx]
▶  mov dx, 310h      ; port A
▶  out dx, al
9  call bekle        ; adım aralığı
10 jmp ileri
11 bekle: ret        ; (gecikme)
12 yarim db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 adim  dw 6
```



yarim[0] = **01h** → yalnız A: alan 0°'ye geldi (yarım adım).

Adım 5 / 8

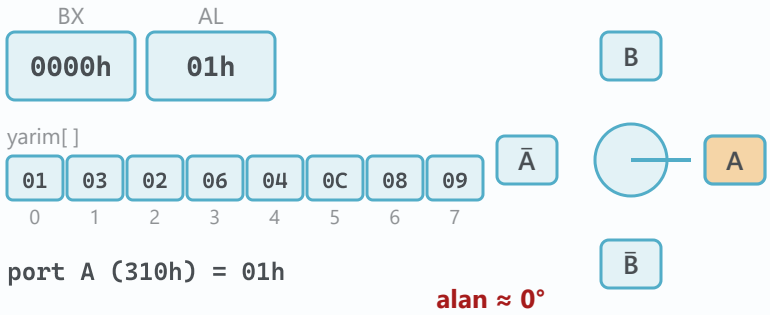
DEĞİŞKENLER

`BX = 0000h` `AL = 01h` `port310h = 01h`

Port 310h'e yazılan baytlar (09h, 01h, 03h …) yorumlayıcının G/Ç kaydından alındı; `bekle` burada yalnız RET'tir.

Gerçek Motor: Yarım Adım Dizisini Sürmek

```
1  org 100h
2  ileri:
3  inc adim          ; geri: dec adim
4  mov bx, adim
5  and bx, 7         ; dizin 0–7
6  mov al, yarim[bx]
7  mov dx, 310h      ; port A
8  out dx, al
▶  call bekle        ; adım aralığı
▶  jmp ileri
▶  bekle: ret        ; (gecikme)
12 yarim db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 adim  dw 6
```



Bekle ve tekrar.

Adım 6 / 8

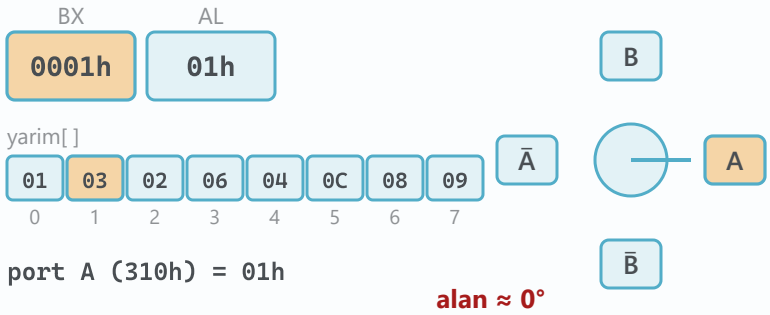
DEĞİŞKENLER

`BX = 0000h` `AL = 01h` `port310h = 01h`

Port 310h'e yazılan baytlar (09h, 01h, 03h …) yorumlayıcının G/Ç kaydından alındı; `bekle` burada yalnız RET'tir.

Gerçek Motor: Yarım Adım Dizisini Sürmek

```
1  org 100h
2  ileri:
▶  inc adim          ; geri: dec adim
▶  mov bx, adim
▶  and bx, 7         ; dizin 0-7
6  mov al, yarim[bx]
7  mov dx, 310h      ; port A
8  out dx, al
9  call bekle        ; adım aralığı
10 jmp ileri
11 bekle: ret        ; (gecikme)
12 yarim db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 adim  dw 6
```



adim = 9 → BX = 1.

Adım 7 / 8

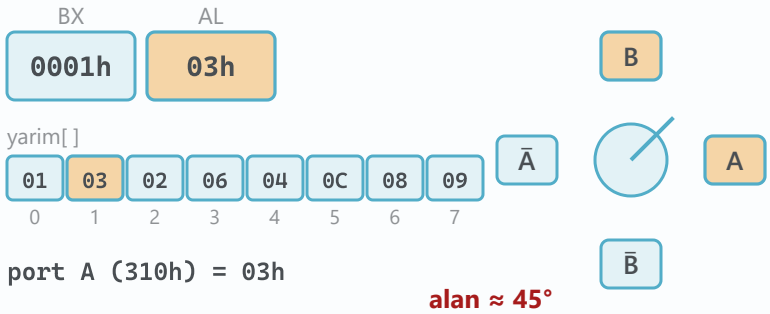
DEĞİŞKENLER

BX = 0001h | AL = 01h | port310h = 01h

Port 310h'e yazılan baytlar (09h, 01h, 03h …) yorumlayıcının G/Ç kaydından alındı; `bekle` burada yalnız RET'tir.

Gerçek Motor: Yarım Adım Dizisini Sürmek

```
1  org 100h
2  ileri:
3  inc adim          ; geri: dec adim
4  mov bx, adim
5  and bx, 7         ; dizin 0–7
▶  mov al, yarim[bx]
▶  mov dx, 310h      ; port A
▶  out dx, al
9  call bekle        ; adım aralığı
10 jmp ileri
11 bekle: ret        ; (gecikme)
12 yarim db 01h,03h,02h,06h
13       db 04h,0Ch,08h,09h
14 adim  dw 6
```



yarim[1] = **03h** → A + B: 45°. Her OUT alanı bir yarım adım (45° sargı açısı) ilerletir; dizi tersten okunursa motor geri döner.

Adım 8 / 8

DEĞİŞKENLER

`BX = 0001h` `AL = 03h` `port310h = 03h`

Port 310h'e yazılan baytlar (09h, 01h, 03h …) yorumlayıcının G/Ç kaydından alındı; `bekle` burada yalnız RET'tir.

LED Ekran (Port 199)

- Port 199'a yazılan **sözcük** 5 basamaklı onluk sayı olarak gösterilir.
- Değer **işaretli** (ikiye tümleyen) yorumlanır: −32 768 … 32 767.
- −5678 = 10000h − 162Eh = **E9D2h**; ekranda **-5678** görünür.
- Yalnızca `OUT 199, AX` kullanılır (199 ≤ 255 olduğundan sabit port yeterli).
- `#make_bin#` : DOS'a bağlı olmayan düz ikili (.bin) dosya üretilir.



port 199 (2 bayt) · AX = 0163h

LED Ekran Kontrolü

```
#start=led_display.exe#
mov ax, 1234
out 199, ax
mov ax, -5678
out 199, ax
; port'a sonsuz döngüde yaz:
mov ax, 0
x1:
out 199, ax
inc ax
jmp x1
hlt                ; hiç çalışmaz
```



port 199 (2 bayt) · AX = 0000h

Ekran başlangıçta 00000 gösterir.

Adım 1 / 15

DEĞİŞKENLER

AX = **0000h**

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; port'a sonsuz döngüde yaz:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; hiç çalışmaz
```



port 199 (2 bayt) · AX = 0000h

AX = 1234 (onluk) = **04D2h**.

Adım 2 / 15

DEĞİŞKENLER

AX = 04D2h

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; port'a sonsuz döngüde yaz:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; hiç çalışmaz
```



port 199 (2 bayt) · AX = 04D2h

OUT 199, AX: ekranda **1234**.

Adım 3 / 15

DEĞİŞKENLER

AX = **04D2h**

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; port'a sonsuz döngüde yaz:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; hiç çalışmaz
```



port 199 (2 bayt) · AX = 04D2h

AX = −5678 = **E9D2h** (ikiye tümleyen).

Adım 4 / 15

DEĞİŞKENLER

AX = **E9D2h**

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
▶  out 199, ax
6  ; port'a sonsuz döngüde yaz:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; hiç çalışmaz
```



port 199 (2 bayt) · AX = E9D2h

OUT 199, AX: ekranda **-5678**.

Adım 5 / 15

DEĞİŞKENLER

AX = **E9D2h**

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; port'a sonsuz döngüde yaz:
   mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; hiç çalışmaz
```



port 199 (2 bayt) · AX = E9D2h

AX = 0: sayaç.

Adım 6 / 15

DEĞİŞKENLER

AX = 0000h

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; port'a sonsuz döngüde yaz:
7  mov ax, 0
8  x1:
▶  out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; hiç çalışmaz
```



port 199 (2 bayt) · AX = 0000h

OUT 199, AX: ekranda **0**.

Adım 7 / 15

DEĞİŞKENLER

`AX = 0000h`

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; port'a sonsuz döngüde yaz:
7  mov ax, 0
8  x1:
9  out 199, ax
▶  inc ax
11 jmp x1
12 hlt              ; hiç çalışmaz
```



port 199 (2 bayt) · AX = 0000h

AX = 1.

Adım 8 / 15

DEĞİŞKENLER

AX = 0001h

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; port'a sonsuz döngüde yaz:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
▶  jmp x1
12 hlt              ; hiç çalışmaz
```



port 199 (2 bayt) · AX = 0000h

JMP x1: sonsuz döngü (alttaki HLT'ye hiç ulaşılmaz).

Adım 9 / 15

DEĞİŞKENLER

AX = **0001h**

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; port'a sonsuz döngüde yaz:
7  mov ax, 0
8  x1:
▶ out 199, ax
10 inc ax
11 jmp x1
12 hlt                ; hiç çalışmaz
```



port 199 (2 bayt) · AX = 0001h

OUT 199, AX: ekranda **1**.

Adım 10 / 15

DEĞİŞKENLER

`AX = 0001h`

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; port'a sonsuz döngüde yaz:
7  mov ax, 0
8  x1:
9  out 199, ax
▶  inc ax
11 jmp x1
12 hlt              ; hiç çalışmaz
```



port 199 (2 bayt) · AX = 0001h

AX = 2.

Adım 11 / 15

DEĞİŞKENLER

AX = 0002h

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; port'a sonsuz döngüde yaz:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
▶  jmp x1
12 hlt              ; hiç çalışmaz
```



port 199 (2 bayt) · AX = 0001h

JMP x1: sonsuz döngü (alttaki HLT'ye hiç ulaşılmaz).

Adım 12 / 15

DEĞİŞKENLER

AX = **0002h**

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; port'a sonsuz döngüde yaz:
7  mov ax, 0
8  x1:
▶ out 199, ax
10 inc ax
11 jmp x1
12 hlt              ; hiç çalışmaz
```



port 199 (2 bayt) · AX = 0002h

OUT 199, AX: ekranda **2**.

Adım 13 / 15

DEĞİŞKENLER

`AX = 0002h`

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; port'a sonsuz döngüde yaz:
7  mov ax, 0
8  x1:
9  out 199, ax
▶  inc ax
11 jmp x1
12 hlt              ; hiç çalışmaz
```



port 199 (2 bayt) · AX = 0002h

AX = 3.

Adım 14 / 15

DEĞİŞKENLER

AX = 0003h

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

LED Ekran Kontrolü

```
1  #start=led_display.exe#
2  mov ax, 1234
3  out 199, ax
4  mov ax, -5678
5  out 199, ax
6  ; port'a sonsuz döngüde yaz:
7  mov ax, 0
8  x1:
9  out 199, ax
10 inc ax
▶  jmp x1
12 hlt              ; hiç çalışmaz
```



port 199 (2 bayt) · AX = 0002h

JMP x1: sonsuz döngü (alttaki HLT'ye hiç ulaşılmaz); sayaç 32767'den sonra −32768'e geçer.

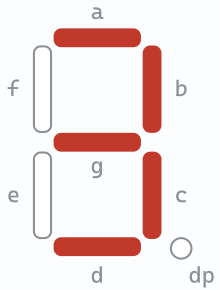
Adım 15 / 15

DEĞİŞKENLER

AX = **0003h**

emu8086'da adım gecikmesi (step delay) 0 iken sayılar çok hızlı akar; döngüye INT 15h/86h ile bekleme eklenebilir.

Gerçek Donanım: 7 Bölütlü Gösterge



PA7 ... PA0 → dp g f e d c b a

D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	0	1	1	1	1
dp	g	f	e	d	c	b	a

ortak katot '3': 4Fh

ortak anot '3': B0h (tersi)

- **Ortak katot** (CC): ortak uç toprakta, bit = 1 bölütü yakar.
- **Ortak anot** (CA): ortak uç +5 V'ta, bit = 0 yakar: kod CC kodunun tersidir (`NOT AL`).
- Her bölüte akım sınırlayıcı direnç bağlanır.
- emu8086'nın port 199 LED ekranı sayıyı kendisi çözer; gerçek göstergede kodu program üretir.

Rakam	Yanan bölütler	CC	CA
0	a b c d e f	3Fh	C0h
1	b c	06h	F9h
2	a b d e g	5Bh	A4h
3	a b c d g	4Fh	B0h
4	b c f g	66h	99h
5	a c d f g	6Dh	92h
6	a c d e f g	7Dh	82h
7	a b c	07h	F8h
8	a b c d e f g	7Fh	80h
9	a b c d f g	6Fh	90h

Kod Tablosu, XLAT ve Çoklamalı Tarama

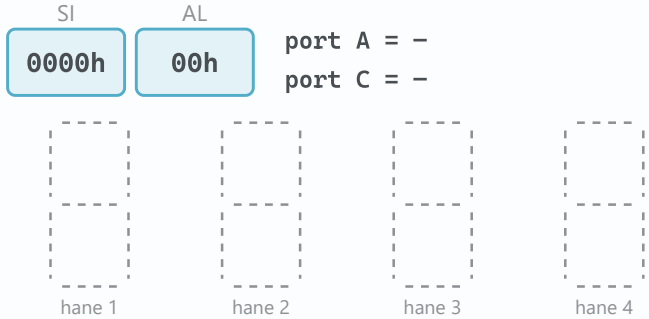
```
; rakamı (0–9) bölüt koduna çevir
mov bx, offset kod
mov al, 3          ; rakam
xlat               ; AL = [BX + AL] = 4Fh
; not al           ; ortak anot ise
mov dx, 310h       ; 8255 port A
out dx, al
kod db 3Fh,06h,5Bh,4Fh,66h
    db 6Dh,7Dh,07h,7Fh,6Fh
```

- **Çoklama:** 4 basamağın bölüt uçları ortak port A'ya, ortak uçları ayrı transistörlerle port C'ye bağlanır.
- Her an tek basamak yanar; basamak başına ≈ 4 ms → tam tur 16 ms ≈ **62 Hz**. Göz titremeyi görmez (≈ 50 Hz üstü).
- Basamağı değiştirmeden önce tümü kapatılır; yoksa önceki kod yan basamakta soluk görünür (gölge).

```
tara:
mov si, 0
hane:
mov dx, 312h       ; port C: hane seçimi
mov al, 0
out dx, al         ; önce tüm haneler sönük
mov al, rakam[si]
mov bx, offset kod
xlat
mov dx, 310h       ; port A: bölütler
out dx, al
mov al, secim[si] ; 01h, 02h, 04h, 08h
mov dx, 312h
out dx, al         ; yalnız bu hane yanar
call bekle_4ms
inc si
cmp si, 4
jb  hane
jmp tara
rakam db 1, 2, 3, 4
secim db 01h, 02h, 04h, 08h
```

Çoklamalı Tarama: 2026 Gösterimi

```
 1 org 100h
 2 mov bx, offset kod
 3 mov si, 0
 4 hane:
 5 mov dx, 312h  ; port C (hane)
 6 mov al, 0
 7 out dx, al    ; tümü sönük
 8 mov al, rakam[si]
 9 xlat          ; bölüt kodu
10 mov dx, 310h  ; port A (bölüt)
11 out dx, al
12 mov al, secim[si]
13 mov dx, 312h
14 out dx, al    ; tek hane yanar
15 inc si
16 cmp si, 4
17 jb  hane
18 ret
19 kod   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 rakam db 2, 0, 2, 6
22 secim db 01h, 02h, 04h, 08h
```



BX = tablo başı, SI = 0. Gösterilecek sayı **2026** (rakam dizisi); 4 hanenin bölüt uçları ortak port A'ya bağlı.

Adım 1 / 10

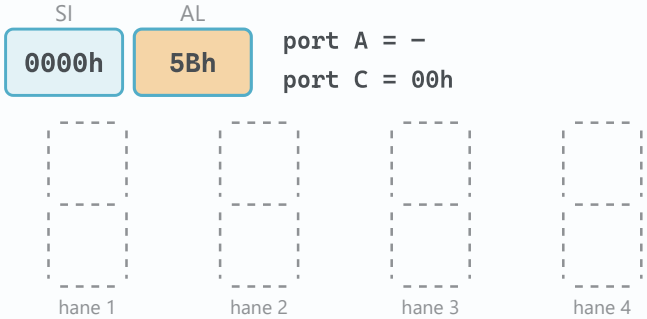
DEĞİŞKENLER

`SI = 0000h` `AL = 00h` `A = –` `C = –`

Ortak katot kodları; port değerleri yorumlayıcının G/Ç kaydından. Gerçekte her hane ≈ 4 ms yanık tutulur.

Çoklamalı Tarama: 2026 Gösterimi

```
 1 org 100h
 2 mov bx, offset kod
 3 mov si, 0
 4 hane:
 5 mov dx, 312h  ; port C (hane)
 6 mov al, 0
 7 out dx, al    ; tümü sönük
 8 mov al, rakam[si]
 9 xlat          ; bölüt kodu
10 mov dx, 310h  ; port A (bölüt)
11 out dx, al
12 mov al, secim[si]
13 mov dx, 312h
14 out dx, al    ; tek hane yanar
15 inc si
16 cmp si, 4
17 jb  hane
18 ret
19 kod   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 rakam db 2, 0, 2, 6
22 secim db 01h, 02h, 04h, 08h
```



1. hane: önce port C ← 00h (hepsi sönük, gölge olmasın). AL = rakam[0] = 2; XLAT → AL = **5Bh**.

Adım 2 / 10

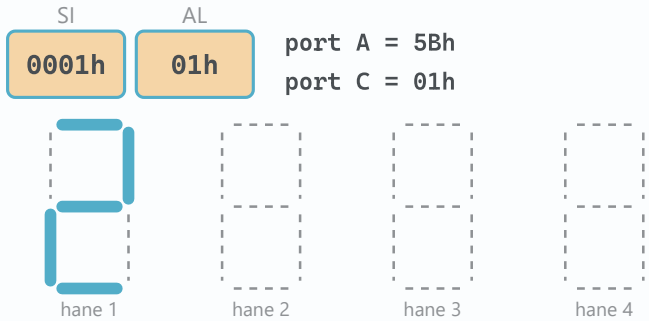
DEĞİŞKENLER

SI = 0000h | AL = 5Bh | A = – | C = 00h

Ortak katot kodları; port değerleri yorumlayıcının G/Ç kaydından. Gerçekte her hane ≈ 4 ms yanık tutulur.

Çoklamalı Tarama: 2026 Gösterimi

```
 1 org 100h
 2 mov bx, offset kod
 3 mov si, 0
 4 hane:
 5 mov dx, 312h  ; port C (hane)
 6 mov al, 0
 7 out dx, al    ; tümü sönük
 8 mov al, rakam[si]
 9 xlat          ; bölüt kodu
10 mov dx, 310h  ; port A (bölüt)
11 out dx, al
12 mov al, secim[si]
13 mov dx, 312h
14 out dx, al    ; tek hane yanar
15 inc si
16 cmp si, 4
17 jb  hane
18 ret
19 kod   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 rakam db 2, 0, 2, 6
22 secim db 01h, 02h, 04h, 08h
```



Port A ← 5Bh (bölütler), port C ← 01h: yalnız 1. hane yanar. INC SI, CMP, JB → sonraki hane.

Adım 3 / 10

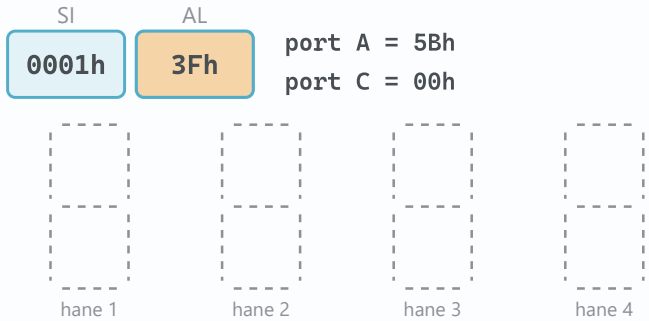
DEĞİŞKENLER

`SI = 0001h` `AL = 01h` `A = 5Bh` `C = 01h`

Ortak katot kodları; port değerleri yorumlayıcının G/Ç kaydından. Gerçekte her hane ≈ 4 ms yanık tutulur.

Çoklamalı Tarama: 2026 Gösterimi

```
 1 org 100h
 2 mov bx, offset kod
 3 mov si, 0
 4 hane:
 5 mov dx, 312h  ; port C (hane)
 6 mov al, 0
 7 out dx, al    ; tümü sönük
 8 mov al, rakam[si]
 9 xlat          ; bölüt kodu
10 mov dx, 310h  ; port A (bölüt)
11 out dx, al
12 mov al, secim[si]
13 mov dx, 312h
14 out dx, al    ; tek hane yanar
15 inc si
16 cmp si, 4
17 jb  hane
18 ret
19 kod   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 rakam db 2, 0, 2, 6
22 secim db 01h, 02h, 04h, 08h
```



2. hane: önce port C ← 00h (hepsi sönük, gölge olmasın). AL = rakam[1] = 0; XLAT → AL = **3Fh**.

Adım 4 / 10

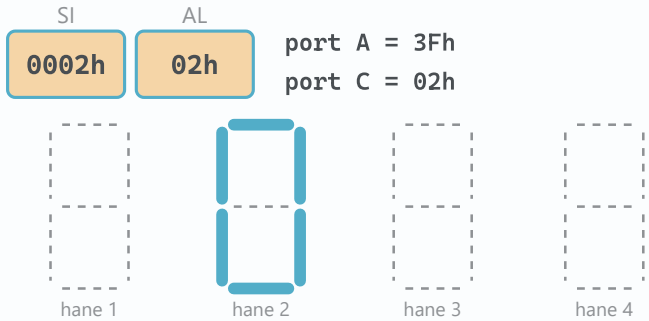
DEĞİŞKENLER

`SI = 0001h` `AL = 3Fh` `A = 5Bh` `C = 00h`

Ortak katot kodları; port değerleri yorumlayıcının G/Ç kaydından. Gerçekte her hane ≈ 4 ms yanık tutulur.

Çoklamalı Tarama: 2026 Gösterimi

```
 1 org 100h
 2 mov bx, offset kod
 3 mov si, 0
 4 hane:
 5 mov dx, 312h  ; port C (hane)
 6 mov al, 0
 7 out dx, al    ; tümü sönük
 8 mov al, rakam[si]
 9 xlat          ; bölüt kodu
10 mov dx, 310h  ; port A (bölüt)
11 out dx, al
12 mov al, secim[si]
13 mov dx, 312h
14 out dx, al    ; tek hane yanar
15 inc si
16 cmp si, 4
17 jb  hane
18 ret
19 kod   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 rakam db 2, 0, 2, 6
22 secim db 01h, 02h, 04h, 08h
```



Port A ← 3Fh (bölütler), port C ← 02h: yalnız 2. hane yanar. INC SI, CMP, JB → sonraki hane.

Adım 5 / 10

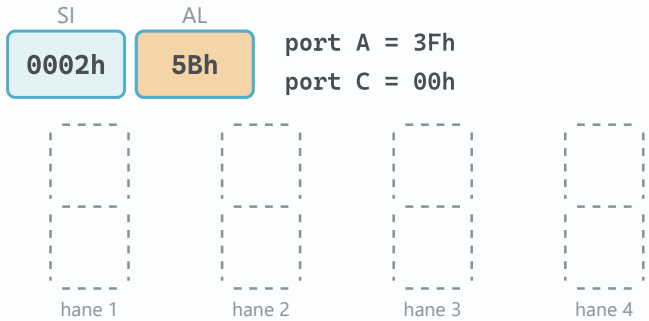
DEĞİŞKENLER

`SI = 0002h` `AL = 02h` `A = 3Fh` `C = 02h`

Ortak katot kodları; port değerleri yorumlayıcının G/Ç kaydından. Gerçekte her hane ≈ 4 ms yanık tutulur.

Çoklamalı Tarama: 2026 Gösterimi

```
 1 org 100h
 2 mov bx, offset kod
 3 mov si, 0
 4 hane:
 ▶ mov dx, 312h  ; port C (hane)
 ▶ mov al, 0
 ▶ out dx, al    ; tümü sönük
 ▶ mov al, rakam[si]
 ▶ xlat          ; bölüt kodu
10 mov dx, 310h  ; port A (bölüt)
11 out dx, al
12 mov al, secim[si]
13 mov dx, 312h
14 out dx, al    ; tek hane yanar
15 inc si
16 cmp si, 4
17 jb  hane
18 ret
19 kod   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 rakam db 2, 0, 2, 6
22 secim db 01h, 02h, 04h, 08h
```



3. hane: önce port C ← 00h (hepsi sönük, gölge olmasın). AL = rakam[2] = 2; XLAT → AL = **5Bh**.

Adım 6 / 10

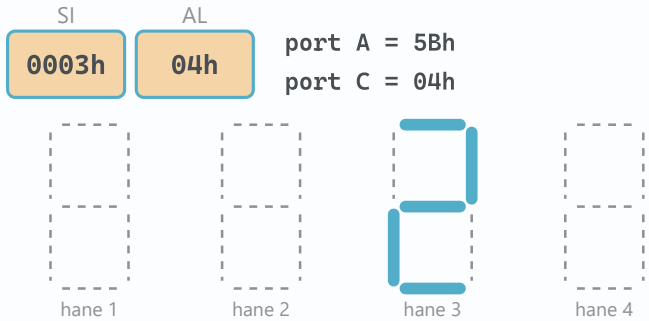
DEĞİŞKENLER

`SI = 0002h` `AL = 5Bh` `A = 3Fh` `C = 00h`

Ortak katot kodları; port değerleri yorumlayıcının G/Ç kaydından. Gerçekte her hane ≈ 4 ms yanık tutulur.

Çoklamalı Tarama: 2026 Gösterimi

```
 1 org 100h
 2 mov bx, offset kod
 3 mov si, 0
 4 hane:
 5 mov dx, 312h  ; port C (hane)
 6 mov al, 0
 7 out dx, al    ; tümü sönük
 8 mov al, rakam[si]
 9 xlat          ; bölüt kodu
10 mov dx, 310h  ; port A (bölüt)
11 out dx, al
12 mov al, secim[si]
13 mov dx, 312h
14 out dx, al    ; tek hane yanar
15 inc si
16 cmp si, 4
17 jb  hane
18 ret
19 kod   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 rakam db 2, 0, 2, 6
22 secim db 01h, 02h, 04h, 08h
```



Port A ← 5Bh (bölütler), port C ← 04h: yalnız 3. hane yanar. INC SI, CMP, JB → sonraki hane.

Adım 7 / 10

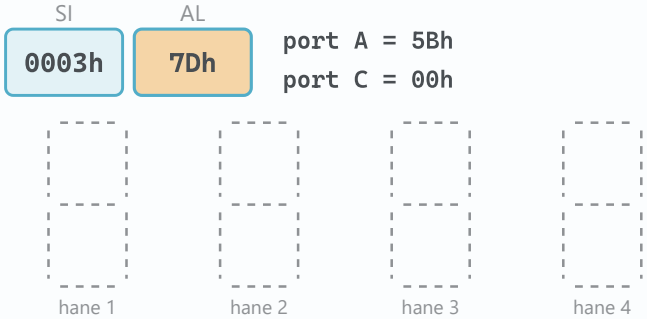
DEĞİŞKENLER

`SI = 0003h` `AL = 04h` `A = 5Bh` `C = 04h`

Ortak katot kodları; port değerleri yorumlayıcının G/Ç kaydından. Gerçekte her hane ≈ 4 ms yanık tutulur.

Çoklamalı Tarama: 2026 Gösterimi

```
 1 org 100h
 2 mov bx, offset kod
 3 mov si, 0
 4 hane:
 5 mov dx, 312h  ; port C (hane)
 6 mov al, 0
 7 out dx, al    ; tümü sönük
 8 mov al, rakam[si]
 9 xlat          ; bölüt kodu
10 mov dx, 310h  ; port A (bölüt)
11 out dx, al
12 mov al, secim[si]
13 mov dx, 312h
14 out dx, al    ; tek hane yanar
15 inc si
16 cmp si, 4
17 jb  hane
18 ret
19 kod   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 rakam db 2, 0, 2, 6
22 secim db 01h, 02h, 04h, 08h
```



4. hane: önce port C ← 00h (hepsi sönük, gölge olmasın). AL = rakam[3] = 6; XLAT → AL = **7Dh**.

Adım 8 / 10

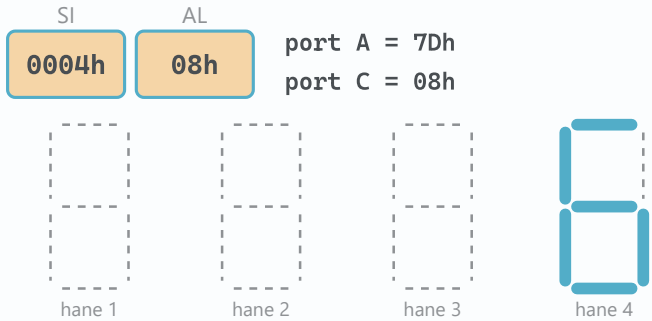
DEĞİŞKENLER

`SI = 0003h` `AL = 7Dh` `A = 5Bh` `C = 00h`

Ortak katot kodları; port değerleri yorumlayıcının G/Ç kaydından. Gerçekte her hane ≈ 4 ms yanık tutulur.

Çoklamalı Tarama: 2026 Gösterimi

```
 1 org 100h
 2 mov bx, offset kod
 3 mov si, 0
 4 hane:
 5 mov dx, 312h  ; port C (hane)
 6 mov al, 0
 7 out dx, al    ; tümü sönük
 8 mov al, rakam[si]
 9 xlat          ; bölüt kodu
10 mov dx, 310h  ; port A (bölüt)
11 out dx, al
12 mov al, secim[si]
13 mov dx, 312h
14 out dx, al    ; tek hane yanar
15 inc si
16 cmp si, 4
17 jb  hane
18 ret
19 kod   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 rakam db 2, 0, 2, 6
22 secim db 01h, 02h, 04h, 08h
```



Port A ← 7Dh (bölütler), port C ← 08h: yalnız 4. hane yanar. INC SI, CMP, JB → SI = 4, döngü biter.

Adım 9 / 10

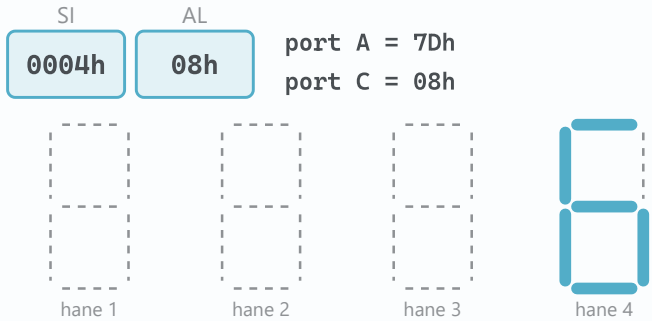
DEĞİŞKENLER

`SI = 0004h` `AL = 08h` `A = 7Dh` `C = 08h`

Ortak katot kodları; port değerleri yorumlayıcının G/Ç kaydından. Gerçekte her hane ≈ 4 ms yanık tutulur.

Çoklamalı Tarama: 2026 Gösterimi

```
 1 org 100h
 2 mov bx, offset kod
 3 mov si, 0
 4 hane:
 5 mov dx, 312h  ; port C (hane)
 6 mov al, 0
 7 out dx, al    ; tümü sönük
 8 mov al, rakam[si]
 9 xlat          ; bölüt kodu
10 mov dx, 310h  ; port A (bölüt)
11 out dx, al
12 mov al, secim[si]
13 mov dx, 312h
14 out dx, al    ; tek hane yanar
15 inc si
16 cmp si, 4
17 jb  hane
▶ ret
19 kod   db 3Fh,06h,5Bh,4Fh,66h
20       db 6Dh,7Dh,07h,7Fh,6Fh
21 rakam db 2, 0, 2, 6
22 secim db 01h, 02h, 04h, 08h
```



Dört hane sırayla birer kez yandı. Her hane ≈ 4 ms yansa tam tur 16 ms (≈ 62 Hz): göz dördünü birlikte görür.

Adım 10 / 10

DEĞİŞKENLER

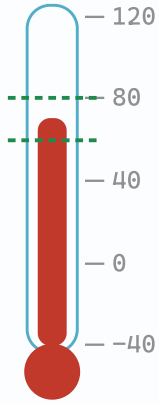
`SI = 0004h` `AL = 08h` `A = 7Dh` `C = 08h`

Ortak katot kodları; port değerleri yorumlayıcının G/Ç kaydından. Gerçekte her hane ≈ 4 ms yanık tutulur.

TERMOMETRE

Termometre ve Isıtıcı (Port 125, 127)

- `IN AL, 125` : anlık hava sıcaklığını (°C, **işaretli bayt**) okur.
- `OUT 127, AL` : AL = 1 ısıtıcıyı açar, AL = 0 kapatır.
- Isıtıcı açıkken sıcaklık yavaşça artar, kapalıyken düşer.
- Amaç: sıcaklığı **60–80 °C** arasında tutan bir **termostat**.
- Sıcaklık eksi olabileceği için karşılaştırmalar **işaretli** atlamalarla (`JL` , `JLE` , `JG`) yapılır.



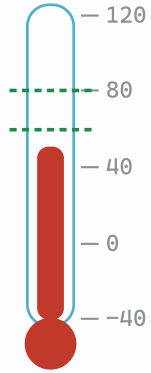
port 125: `70` °C

ısıtıcı KAPALI

port 127 = 0
hedef aralık: 60–80 °C

Termostat Programı

```
; DS'yi CS'ye eşitle:
mov ax, cs
mov ds, ax
start:
in  al, 125
cmp al, 60
jl  low
cmp al, 80
jle ok
jg  high
low:
mov al, 1
out 127, al   ; ısıtıcı açık
jmp ok
high:
mov al, 0
out 127, al   ; ısıtıcı kapalı
ok:
jmp start     ; sonsuz döngü
```



port 125: 50 °C

ISITICI KAPALI

port 127 = 0
hedef aralık: 60–80 °C

Port 125 sırayla 55, 70 ve 85 °C döndürecek. Isıtıcı başlangıçta kapalı.

Adım 1 / 25

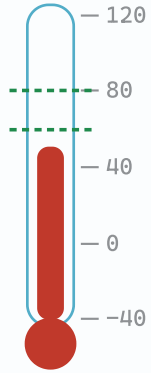
DEĞİŞKENLER

AX = 0000h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 50 °C

ISITICI KAPALI

port 127 = 0
hedef aralık: 60–80 °C

AX = CS = 0700h.

Adım 2 / 25

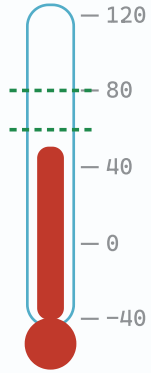
DEĞİŞKENLER

AX = 0700h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
▶  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 50 °C

ISITICI KAPALI

port 127 = 0
hedef aralık: 60–80 °C

DS = 0700h (kesim yazmacına sabit doğrudan yazılamaz).

Adım 3 / 25

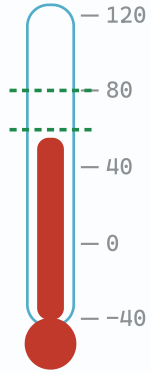
DEĞİŞKENLER

AX = 0700h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
▶ in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 55 °C

ISITICI KAPALI

port 127 = 0
hedef aralık: 60–80 °C

IN AL, 125 → sıcaklık **55 °C**.

Adım 4 / 25

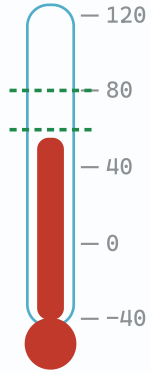
DEĞİŞKENLER

AX = 0737h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 55 °C

ISITICI KAPALI

port 127 = 0
hedef aralık: 60–80 °C

CMP AL, 60: 55 − 60 → SF = 1, OF = 0.

Adım 5 / 25

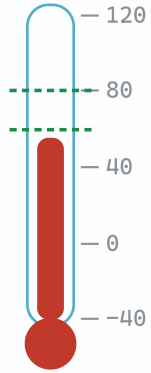
DEĞİŞKENLER

AX = 0737h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
▶  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 55 °C

ISITICI KAPALI

port 127 = 0
hedef aralık: 60–80 °C

JL (SF ≠ OF): **55 < 60** → low.

Adım 6 / 25

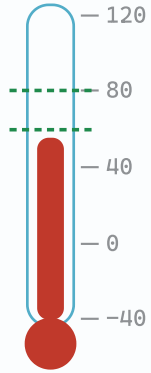
DEĞİŞKENLER

AX = 0737h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
▶ mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 55 °C

ISITICI KAPALI

port 127 = 0
hedef aralık: 60–80 °C

AL = 1.

Adım 7 / 25

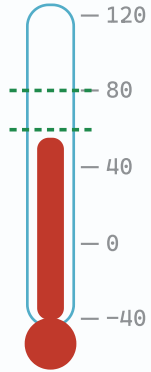
DEĞİŞKENLER

AX = 0701h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
▶ out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 55 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

OUT 127, AL: ısıtıcı **açıldı**.

Adım 8 / 25

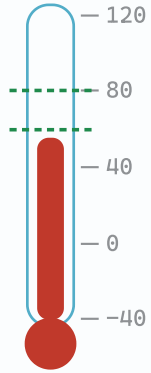
DEĞİŞKENLER

AX = 0701h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
▶ jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 55 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

JMP ok.

Adım 9 / 25

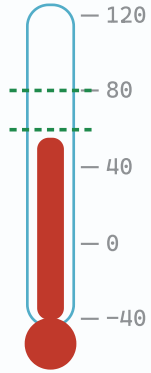
DEĞİŞKENLER

AX = 0701h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
▶  jmp start     ; sonsuz döngü
```



port 125: 55 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

JMP start: yeni ölçüm.

Adım 10 / 25

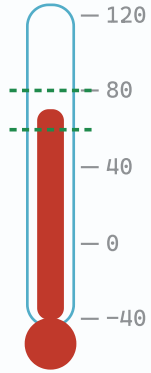
DEĞİŞKENLER

AX = 0701h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
▶ in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: **70 °C**

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

IN AL, 125 → sıcaklık **70 °C**.

Adım 11 / 25

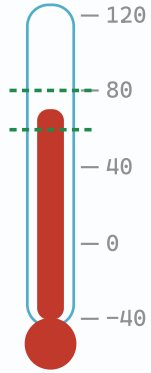
DEĞİŞKENLER

AX = 0746h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 70 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

CMP AL, 60: 70 − 60 → SF = 0, OF = 0.

Adım 12 / 25

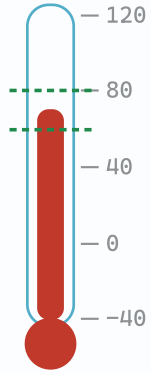
DEĞİŞKENLER

AX = 0746h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
▶  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 70 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

JL (SF ≠ OF): atlama yok.

Adım 13 / 25

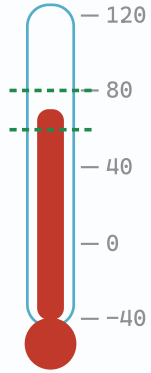
DEĞİŞKENLER

AX = 0746h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
▶  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 70 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

CMP AL, 80 → ZF = 0, SF = 1.

Adım 14 / 25

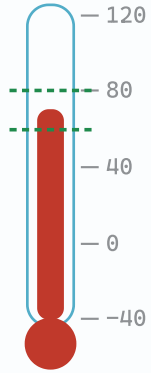
DEĞİŞKENLER

AX = 0746h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
▶  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 70 °C

ısıtıcı AÇIK

port 127 = 1
hedef aralık: 60–80 °C

JLE: 60 ≤ 70 ≤ 80 → ok (ısıtıcıya dokunma).

Adım 15 / 25

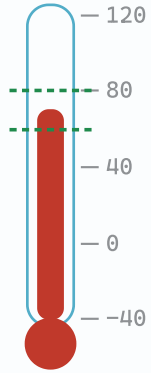
DEĞİŞKENLER

AX = 0746h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
▶  jmp start     ; sonsuz döngü
```



port 125: 70 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

JMP start: yeni ölçüm.

Adım 16 / 25

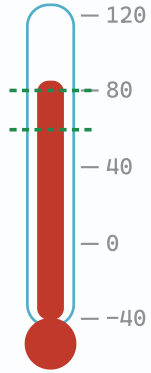
DEĞİŞKENLER

AX = 0746h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
▶ in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 85 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

IN AL, 125 → sıcaklık **85 °C**.

Adım 17 / 25

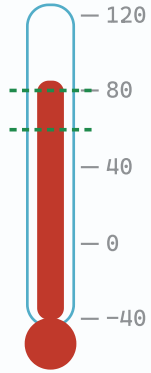
DEĞİŞKENLER

AX = 0755h | DS = 0700h | ZF = 0 | SF = 1 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 85 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

CMP AL, 60: 85 − 60 → SF = 0, OF = 0.

Adım 18 / 25

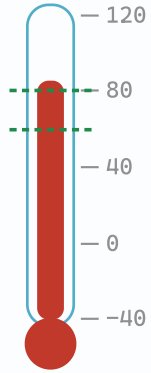
DEĞİŞKENLER

AX = 0755h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
▶ 7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 85 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

JL (SF ≠ OF): atlama yok.

Adım 19 / 25

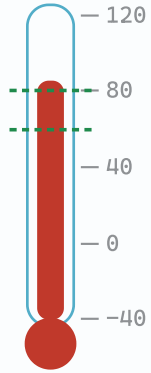
DEĞİŞKENLER

`AX = 0755h` `DS = 0700h` `ZF = 0` `SF = 0` `OF = 0`

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
▶  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 85 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

CMP AL, 80 → ZF = 0, SF = 0.

Adım 20 / 25

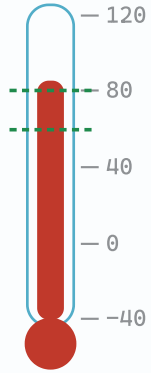
DEĞİŞKENLER

AX = 0755h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
▶  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 85 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

JLE: atlama yok.

Adım 21 / 25

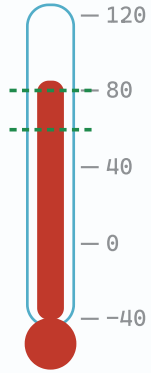
DEĞİŞKENLER

AX = 0755h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
▶  jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 85 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

JG: 85 > 80 → high.

Adım 22 / 25

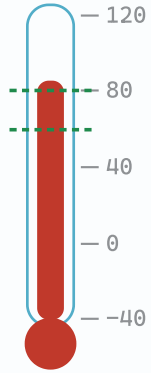
DEĞİŞKENLER

AX = 0755h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
▶ mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 85 °C

ISITICI AÇIK

port 127 = 1
hedef aralık: 60–80 °C

AL = 0.

Adım 23 / 25

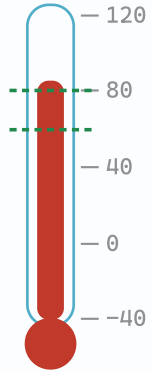
DEĞİŞKENLER

AX = 0700h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
▶ 17 out 127, al   ; ısıtıcı kapalı
18 ok:
19 jmp start     ; sonsuz döngü
```



port 125: 85 °C

ISITICI KAPALI

port 127 = 0
hedef aralık: 60–80 °C

OUT 127, AL: ısıtıcı **kapatıldı**.

Adım 24 / 25

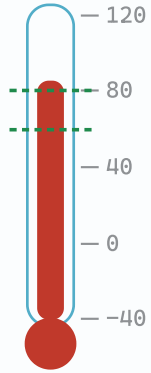
DEĞİŞKENLER

AX = 0700h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

Termostat Programı

```
1  ; DS'yi CS'ye eşitle:
2  mov ax, cs
3  mov ds, ax
4  start:
5  in  al, 125
6  cmp al, 60
7  jl  low
8  cmp al, 80
9  jle ok
10 jg  high
11 low:
12 mov al, 1
13 out 127, al   ; ısıtıcı açık
14 jmp ok
15 high:
16 mov al, 0
17 out 127, al   ; ısıtıcı kapalı
18 ok:
▶  jmp start     ; sonsuz döngü
```



port 125: 85 °C

ISITICI KAPALI

port 127 = 0
hedef aralık: 60–80 °C

JMP start: yeni ölçüm.

Adım 25 / 25

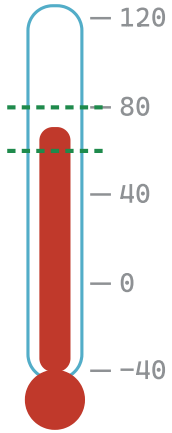
DEĞİŞKENLER

AX = 0700h | DS = 0700h | ZF = 0 | SF = 0 | OF = 0

`jg high` yerine `jmp high` da yazılabilirdi: JLE atlamadıysa sıcaklık zaten 80'den büyüktür.

TERMOMETRE

Etkileşimli Termostat



port 125: **70** °C

ısıtıcı KAPALI

port 127 = 0

Sıcaklık: **70** °C

▶ zamanı akıt

```
in al, 125
cmp al, 60
jl low
cmp al, 80
jle ok
jg high
low: mov al, 1
out 127, al
high: mov al, 0
out 127, al
ok: jmp start
```

60 ≤ 70 ≤ 80 → JLE alındı, ısıtıcıya dokunulmadı (kapalı kalır).

AL = 46h

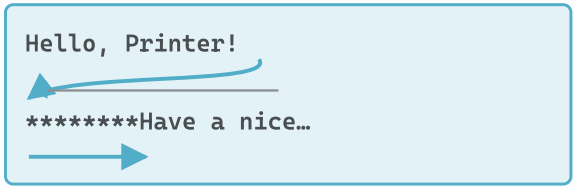
Yazıcı (Printer, Port 130)

- emu8086 yazıcısı port 130'a bağlıdır; ama program porta doğrudan yazmak yerine **DOS işlevini** kullanır:
- `INT 21h` , `AH = 05h` : DL'deki karakteri yazıcıya (PRN) gönderir.
- Kontrol karakterleri yazıcıyı yönetir:

Kod	Ad	Etki
0Ah (10)	LF	bir satır aşağı
0Dh (13)	CR	satır başına dön
09h (9)	HT	yatay sekme (tab)
0Ch (12)	FF	yeni sayfa (form feed)

`db 13, 9` : 13 = CR (satır başı), 9 = **yatay** sekme (HT). Dikey sekme (VT) 11 = 0Bh'dir.

`0Ah, 0Dh` sırası LF + CR'dir; alışılmış sıra CR + LF'tir, ama ikisi birlikte aynı sonucu verir.



0Ah, 0Dh: alt satırın başı

CR (13): aynı satırın başına dön

HT (9): 8. sütuna atla → üzerine bas

FF (12): sayfayı çıkar, yeni sayfa

Yazıcı Programı: Veri

```
org 100h
jmp start
msg db "Hello, Printer!", 0Ah, 0Dh
    db "***************"
    db 13, 9    ; CR ve yatay sekme (HT)
    db "Have a nice printing day!"
msg_end db 0
msg2 db "press any key to eject the page.$"
```

Uzunluk

`offset msg_end - offset msg` derleme zamanında hesaplanır: 15 + 2 + 15 + 2 + 25 = **59** = 3Bh bayt.

msg_end

Son karakterden hemen sonraki adres; değeri (0) yazdırılmaz, yalnızca bir **işaret**tir.

msg2

`'$'` ile biter: INT 21h / AH = 09h ekrana yazdırırken `$`'a kadar gider.

YAZICI · KODU ADIM ADIM ÇALIŞTIR

Yazıcı Programı: Yazdırma Döngüsü

```
▶ start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

Çıktı paneli yazıcı kâğıdını gösterir: LF satır atlar, CR satır başına döner, TAB 8 sütunluk sekmeye gider.

Adım 1 / 33

DEĞİŞKENLER

`CX = 0000h` `DX = 0000h` `SI = 0000h`

ÇIKTI

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
▶  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

DL = 12 (0Ch): form feed.

Adım 2 / 33

DEĞİŞKENLER

CX = 0000h | DX = 000Ch | SI = 0000h

ÇIKTI



CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

AH = 05h: DOS yazıcı çıkışı.

Adım 3 / 33

DEĞİŞKENLER

`CX = 0000h` `DX = 000Ch` `SI = 0000h`

ÇIKTI

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

INT 21h: yazıcı yeni sayfaya geçer.

Adım 4 / 33

DEĞİŞKENLER

CX = 0000h | DX = 000Ch | SI = 0000h

ÇIKTI



CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
▶  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

SI = 0102h: msg'nin adresi.

Adım 5 / 33

DEĞİŞKENLER

CX = 0000h | DX = 000Ch | SI = 0102h

ÇIKTI



CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
▶  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

CX = 59 (003Bh): karakter sayısı.

Adım 6 / 33

DEĞİŞKENLER

CX = 003Bh | DX = 000Ch | SI = 0102h

ÇIKTI

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
▶  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

DL ← [SI] = 'H'.

Adım 7 / 33

DEĞİŞKENLER

CX = 003Bh | DX = 0048h | SI = 0102h

ÇIKTI

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
▶  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

AH = 05h: DOS yazıcı çıkışı.

Adım 8 / 33

DEĞİŞKENLER

CX = **003Bh** DX = **0048h** SI = **0102h**

ÇIKTI



CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
▶  int 21h
11 inc si           ; sonraki karakter
12 loop print
```

INT 21h: 'H' yazıcıya gider.

Adım 9 / 33

DEĞİŞKENLER

CX = **003Bh** DX = **0048h** SI = **0102h**

ÇIKTI

```
H
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
▶  inc si           ; sonraki karakter
12 loop print
```

SI = 0103h.

Adım 10 / 33

DEĞİŞKENLER

CX = 003Bh | DX = 0048h | SI = 0103h

ÇIKTI

```
H
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
▶  loop print
```

LOOP: CX = 58 → devam.

Adım 11 / 33

DEĞİŞKENLER

CX = 003Ah | DX = 0048h | SI = 0103h

ÇIKTI

```
H
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
▶  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

DL ← [SI] = 'e'.

Adım 12 / 33

DEĞİŞKENLER

CX = 003Ah | DX = 0065h | SI = 0103h

ÇIKTI

```
H
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
▶  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

AH = 05h: DOS yazıcı çıkışı.

Adım 13 / 33

DEĞİŞKENLER

CX = **003Ah** DX = **0065h** SI = **0103h**

ÇIKTI

```
H
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
▶  int 21h
11 inc si           ; sonraki karakter
12 loop print
```

INT 21h: 'e' yazıcıya gider.

Adım 14 / 33

DEĞİŞKENLER

`CX = 003Ah` `DX = 0065h` `SI = 0103h`

ÇIKTI

```
He
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
▶  inc si           ; sonraki karakter
12 loop print
```

SI = 0104h.

Adım 15 / 33

DEĞİŞKENLER

CX = 003Ah | DX = 0065h | SI = 0104h

ÇIKTI

```
He
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
▶  loop print
```

LOOP: CX = 57 → devam.

Adım 16 / 33

DEĞİŞKENLER

CX = 0039h | DX = 0065h | SI = 0104h

ÇIKTI

```
He
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
▶  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

DL ← [SI] = 'l'.

Adım 17 / 33

DEĞİŞKENLER

CX = 0039h | DX = 006Ch | SI = 0104h

ÇIKTI

```
He
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
▶  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

AH = 05h: DOS yazıcı çıkışı.

Adım 18 / 33

DEĞİŞKENLER

`CX = 0039h` `DX = 006Ch` `SI = 0104h`

ÇIKTI

```
He
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
▶  int 21h
11 inc si           ; sonraki karakter
12 loop print
```

INT 21h: 'l' yazıcıya gider.

Adım 19 / 33

DEĞİŞKENLER

`CX = 0039h` `DX = 006Ch` `SI = 0104h`

ÇIKTI

```
Hel
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
▶  inc si           ; sonraki karakter
12 loop print
```

SI = 0105h.

Adım 20 / 33

DEĞİŞKENLER

CX = 0039h | DX = 006Ch | SI = 0105h

ÇIKTI

```
Hel
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
▶  loop print
```

LOOP: CX = 56 → devam.

Adım 21 / 33

DEĞİŞKENLER

CX = 0038h | DX = 006Ch | SI = 0105h

ÇIKTI

```
Hel
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
▶  int 21h
11 inc si           ; sonraki karakter
12 loop print
```

... INT 21h: 'l' yazıldı (CX = 56).

Adım 22 / 33

DEĞİŞKENLER

CX = **0038h** DX = **006Ch** SI = **0105h**

ÇIKTI

```
Hell
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
▶  int 21h
11 inc si           ; sonraki karakter
12 loop print
```

... INT 21h: 't' yazıldı (CX = 48).

Adım 23 / 33

DEĞİŞKENLER

CX = 0030h | DX = 0074h | SI = 010Dh

ÇIKTI

```
Hello, Print
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
▶  int 21h
11 inc si           ; sonraki karakter
12 loop print
```

... INT 21h: '*' yazıldı (CX = 40).

Adım 24 / 33

DEĞİŞKENLER

CX = 0028h DX = 002Ah SI = 0115h

ÇIKTI

```
Hello, Printer!
***
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
▶  int 21h
11 inc si           ; sonraki karakter
12 loop print
```

... INT 21h: '*' yazıldı (CX = 32).

Adım 25 / 33

DEĞİŞKENLER

CX = 0020h | DX = 002Ah | SI = 011Dh

ÇIKTI

```
Hello, Printer!
**********
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
▶  int 21h
11 inc si           ; sonraki karakter
12 loop print
```

... INT 21h: 'a' yazıldı (CX = 24).

Adım 26 / 33

DEĞİŞKENLER

CX = 0018h DX = 0061h SI = 0125h

ÇIKTI

```
Hello, Printer!
********Ha*****
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
▶  int 21h
11 inc si           ; sonraki karakter
12 loop print
```

... INT 21h: 'c' yazıldı (CX = 16).

Adım 27 / 33

DEĞİŞKENLER

CX = 0010h DX = 0063h SI = 012Dh

ÇIKTI

```
Hello, Printer!
********Have a nic
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
▶  int 21h
11 inc si           ; sonraki karakter
12 loop print
```

... INT 21h: 'i' yazıldı (CX = 8).

Adım 28 / 33

DEĞİŞKENLER

CX = 0008h DX = 0069h SI = 0135h

ÇIKTI

```
Hello, Printer!
********Have a nice printi
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
▶  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

DL ← [SI] = '!'.

Adım 29 / 33

DEĞİŞKENLER

CX = 0001h DX = 0021h SI = 013Ch

ÇIKTI

```
Hello, Printer!
********Have a nice printing day
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
▶  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
12 loop print
```

AH = 05h: DOS yazıcı çıkışı.

Adım 30 / 33

DEĞİŞKENLER

CX = **0001h** DX = **0021h** SI = **013Ch**

ÇIKTI

```
Hello, Printer!
********Have a nice printing day
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
▶  int 21h
11 inc si           ; sonraki karakter
12 loop print
```

... INT 21h: '!' yazıldı (CX = 1).

Adım 31 / 33

DEĞİŞKENLER

CX = **0001h** DX = **0021h** SI = **013Ch**

ÇIKTI

```
Hello, Printer!
********Have a nice printing day!
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
▶  inc si           ; sonraki karakter
12 loop print
```

SI = 013Dh.

Adım 32 / 33

DEĞİŞKENLER

CX = **0001h** DX = **0021h** SI = **013Dh**

ÇIKTI

```
Hello, Printer!
********Have a nice printing day!
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Yazıcı Programı: Yazdırma Döngüsü

```
1  start:
2  mov dl, 12       ; form feed: yeni sayfa
3  mov ah, 5
4  int 21h
5  mov si, offset msg
6  mov cx, offset msg_end - offset msg
7  print:
8  mov dl, [si]
9  mov ah, 5        ; DOS yazıcı işlevi
10 int 21h
11 inc si           ; sonraki karakter
▶  loop print
```

LOOP: CX = 0 → **59 karakter bitti**.

Adım 33 / 33

DEĞİŞKENLER

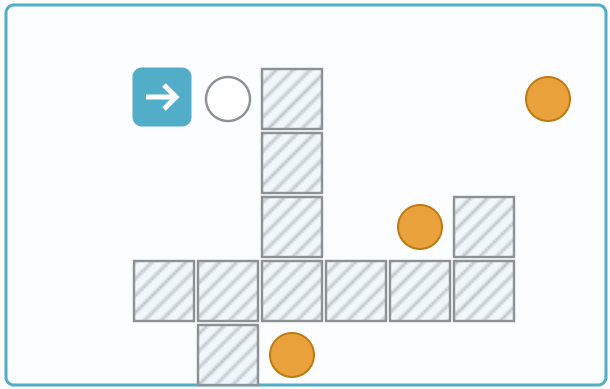
CX = 0000h | DX = 0021h | SI = 013Dh

ÇIKTI

```
Hello, Printer!
********Have a nice printing day!
```

CR (13) LF olmadan gelince üçüncü metin yıldızların **üzerine** basılır (panelde üzerine yazma olarak gösterildi). Döngüden sonra: `mov dx, offset msg2` / `mov ah, 9` / `int 21h` ekrana mesaj yazar; `int 16h` (AH = 0) bir tuş bekler; son form feed sayfayı yazıcıdan çıkarır.

Robot (Port 9, 10, 11)



```
command (9): 00000001
data (10): 00000000
status (11): 00000000
```

yanan lamba

sönük lamba

duvar

- Robot bir ızgara dünyasında hareket eder; duvarları ve lambaları algılayabilir, lambaları yakıp söndürebilir.
- Port **9** komut (OUT), port **10** veri (IN), port **11** durum (IN). Dünya, aygıt penceresindeki araç kutusuyla düzenlenir.

Robot Komutları ve Durum Bitleri

Komut (port 9)	Etki
0	hiçbir şey yapma
1	ileri git
2	sola dön
3	sağa dön
4	önündekini incele (examine)
5	lambayı yak
6	lambayı söndür

Veri (port 10)	Önündeki
255 (FFh)	duvar
0	boş
7	yanan lamba
8	sönük lamba

Durum (port 11)	1 ise
bit 0	veri yazmacında yeni veri var
bit 1	robot meşgul (komut sürüyor)
bit 2	son komutta hata

Robot Programı: Lambayı Bul ve Yak

```
#start=robot.exe#
dongu: CALL hazir_bekle
       MOV  AL, 4       ; incele
       OUT  9, AL
       CALL veri_bekle
       IN   AL, 10
       CMP  AL, 0FFh    ; duvar?
       JE   don
       CMP  AL, 8       ; sönük lamba?
       JNE  ilerle
       CALL hazir_bekle
       MOV  AL, 5       ; yak
       OUT  9, AL
       JMP  dongu
don:   MOV  AL, 3       ; sağa dön
       JMP  gonder
ilerle: MOV AL, 1       ; ileri
gonder: PUSH AX         ; komutu koru
       CALL hazir_bekle
       POP  AX
       OUT  9, AL
       JMP  dongu
```

- Her komuttan önce **meşgul** biti (bit 1) sıfır olana kadar beklenir.
- İncele (4) komutundan sonra **veri hazır** biti (bit 0) beklenir, sonra port 10 okunur.
- Duvarsa sağa dön, sönük lambaysa yak, yoksa ilerle.
- Bekleme yordamları `CALL` ile çağrılır: dönüş adresi yığında.

Robot: Bekleme Yordamları ve Dikkat Edilecekler

```
hazir_bekle PROC
b1:     IN   AL, 11
        TEST AL, 010b     ; meşgul?
        JNZ  b1
        RET
hazir_bekle ENDP

veri_bekle PROC
v1:     IN   AL, 11
        TEST AL, 001b     ; yeni veri?
        JZ   v1
        RET
veri_bekle ENDP
```

AL korunmalı

`hazir_bekle` AL'yi IN ile ezer. `gonder` yolunda AL komut değerini taşıdığı için yordamdan önce `PUSH AX`, sonra `POP AX` yapılır; aksi hâlde porta durum baytı gönderilirdi.

Hata biti

Duvara doğru “ileri” komutu hata (bit 2) üretir; durum okunup kontrol edilmelidir.

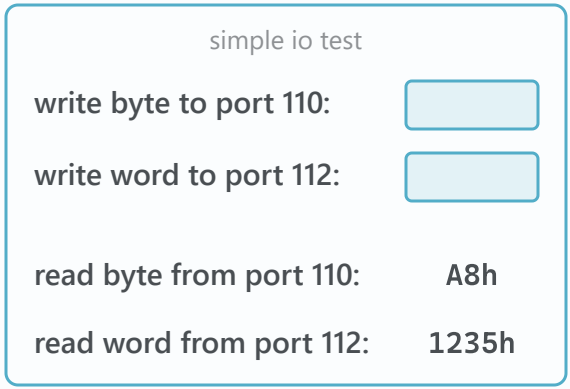
Yoklama maliyeti

CPU bekleme döngüsünde tamamen meşguldür; gerçek sistemlerde kesmeler tercih edilir.

Basit G/Ç Test Aygıtı (Port 110, 112)

- Port **110**: bir **bayt**; port **112**: bir **sözcük**.
- Pencereye yazılan değerler programa giriş olur (hex için `h` soneki), programın yazdıkları pencerede görünür.
- Giriş portları her 100 ms'de güncellenir.
- IN/OUT'u denemek ve hata ayıklamak için idealdir.

```
IN   AL, 110    ; ör. A7h okunur
INC  AL
OUT  110, AL    ; A8h yazılır
IN   AX, 112    ; ör. 1234h
ADD  AX, 1
OUT  112, AX    ; 1235h
```



Test Aygıtı: Bayt ve Sözcük G/Ç

```
1 org 100h
▶ in   al, 110      ; bayt oku
3 inc  al
4 out  110, al      ; bayt yaz
5 in   ax, 112      ; sözcük oku
6 add  ax, 1
7 out  112, ax      ; sözcük yaz
8 ret
```

AX

00FFh	ZF	SF	OF	CF
	0	0	0	0

port 110 (bayt)	giriş: FFh	çıkış: –
port 112 (sözcük)	giriş: 7FFFh	çıkış: –

Test aygıtı penceresine girilen değerler: port 110 = **FFh**, port 112 = **7FFFh**. IN AL, 110 → AL = FFh.

Adım 1 / 7

DEĞİŞKENLER

AX = 00FFh | ZF = 0 | SF = 0 | OF = 0 | CF = 0

Giriş değerleri yorumlayıcının port sözlüğüne verildi; çıkışlar G/Ç kaydından okundu.

Test Aygıtı: Bayt ve Sözcük G/Ç

```
1 org 100h
2 in   al, 110     ; bayt oku
▶ inc  al
4 out  110, al     ; bayt yaz
5 in   ax, 112     ; sözcük oku
6 add  ax, 1
7 out  112, ax     ; sözcük yaz
8 ret
```

AX

0000h	ZF	SF	OF	CF
	1	0	0	0

port 110 (bayt)	giriş: FFh	çıkış: –
port 112 (sözcük)	giriş: 7FFFh	çıkış: –

INC AL: FFh + 1 = **00h** (8 bitte taşar). ZF = 1; INC, CF'yi **değiştirmez**.

Adım 2 / 7

DEĞİŞKENLER

AX = 0000h | ZF = 1 | SF = 0 | OF = 0 | CF = 0

Giriş değerleri yorumlayıcının port sözlüğüne verildi; çıkışlar G/Ç kaydından okundu.

Test Aygıtı: Bayt ve Sözcük G/Ç

```
1 org 100h
2 in   al, 110     ; bayt oku
3 inc  al
▶ out  110, al     ; bayt yaz
5 in   ax, 112     ; sözcük oku
6 add  ax, 1
7 out  112, ax     ; sözcük yaz
8 ret
```

AX

0000h	ZF	SF	OF	CF
	1	0	0	0

port 110 (bayt)	giriş: FFh	çıkış: 00h
port 112 (sözcük)	giriş: 7FFFh	çıkış: –

OUT 110, AL: pencerede port 110 = 00h.

Adım 3 / 7

DEĞİŞKENLER

AX = 0000h | ZF = 1 | SF = 0 | OF = 0 | CF = 0

Giriş değerleri yorumlayıcının port sözlüğüne verildi; çıkışlar G/Ç kaydından okundu.

Test Aygıtı: Bayt ve Sözcük G/Ç

```
1 org 100h
2 in   al, 110      ; bayt oku
3 inc  al
4 out  110, al      ; bayt yaz
▶ in   ax, 112      ; sözcük oku
6 add  ax, 1
7 out  112, ax      ; sözcük yaz
8 ret
```

AX

7FFFh

ZF	SF	OF	CF
1	0	0	0

port 110 (bayt)	giriş: FFh	çıkış: 00h
port 112 (sözcük)	giriş: 7FFFh	çıkış: –

IN AX, 112: 16 bit okuma → AX = **7FFFh** (+32 767).

Adım 4 / 7

DEĞİŞKENLER

AX = **7FFFh** | ZF = **1** | SF = **0** | OF = **0** | CF = **0**

Giriş değerleri yorumlayıcının port sözlüğüne verildi; çıkışlar G/Ç kaydından okundu.

Test Aygıtı: Bayt ve Sözcük G/Ç

```
1 org 100h
2 in   al, 110      ; bayt oku
3 inc  al
4 out  110, al      ; bayt yaz
5 in   ax, 112      ; sözcük oku
▶ add  ax, 1
7 out  112, ax      ; sözcük yaz
8 ret
```

AX **8000h**

ZF	SF	OF	CF
0	1	1	0

port 110 (bayt)	giriş: FFh	çıkış: 00h
port 112 (sözcük)	giriş: 7FFFh	çıkış: –

ADD AX, 1 → **8000h**: iki pozitifin toplamı negatif oldu → OF = 1, SF = 1; işaretsiz taşma yok, CF = 0.

Adım 5 / 7

DEĞİŞKENLER

AX = **8000h** ZF = **0** SF = **1** OF = **1** CF = **0**

Giriş değerleri yorumlayıcının port sözlüğüne verildi; çıkışlar G/Ç kaydından okundu.

Test Aygıtı: Bayt ve Sözcük G/Ç

```
1 org 100h
2 in   al, 110     ; bayt oku
3 inc  al
4 out  110, al     ; bayt yaz
5 in   ax, 112     ; sözcük oku
6 add  ax, 1
▶ out  112, ax     ; sözcük yaz
8 ret
```

AX

8000h

ZF	SF	OF	CF
0	1	1	0

port 110 (bayt)	giriş: FFh	çıkış: 00h
port 112 (sözcük)	giriş: 7FFFh	çıkış: 8000h

OUT 112, AX: pencerede port 112 = 8000h.

Adım 6 / 7

DEĞİŞKENLER

AX = **8000h** ZF = **0** SF = **1** OF = **1** CF = **0**

Giriş değerleri yorumlayıcının port sözlüğüne verildi; çıkışlar G/Ç kaydından okundu.

Test Aygıtı: Bayt ve Sözcük G/Ç

```
1 org 100h
2 in   al, 110     ; bayt oku
3 inc  al
4 out  110, al     ; bayt yaz
5 in   ax, 112     ; sözcük oku
6 add  ax, 1
7 out  112, ax     ; sözcük yaz
▶ ret
```

AX

8000h

ZF	SF	OF	CF
0	1	1	0

port 110 (bayt)	giriş: FFh	çıkış: 00h
port 112 (sözcük)	giriş: 7FFFh	çıkış: 8000h

RET: program biter.

Adım 7 / 7

DEĞİŞKENLER

AX = **8000h** ZF = **0** SF = **1** OF = **1** CF = **0**

Giriş değerleri yorumlayıcının port sözlüğüne verildi; çıkışlar G/Ç kaydından okundu.

Laboratuvar: Port Aygıtları

Trafik ışıkları (port 4)

KOD · **46 bayt**

```
1  #start=Traffic_Lights.exe#
2  org 100h
3  all_red equ 0000_0010_0100…
4  mov ax, all_red
5  out 4, ax
6  mov si, offset situation
7  next:
8  mov ax, [si]
9  out 4, ax
10 mov cx, 4Ch       ; 5 s bek…
11 mov dx, 4B40h
12 mov ah, 86h
13 int 15h
14 add si, 2         ; sonraki…
```

YAZMAÇLAR

AX	0000	AH	00	AL	00
BX	0000	BH	00	BL	00
CX	002E	CH	00	CL	2E
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

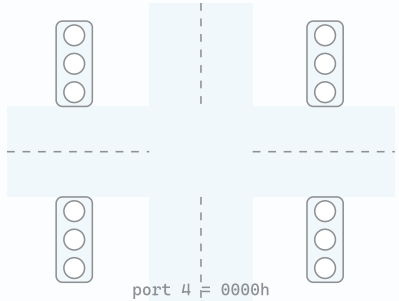
O	D	I	T	S	Z	A	P	C
0	0	1	0	0	0	0	0	0

CS:IP = 0700:0100 · adım 0

BELLEK **DS = 0700h**

```
0120 24 01 EB E4 0C 03 9A 06
0128 61 08 61 08 D3 04 00 00
0130 00 00 00 00 00 00 00 00
```

AYGITLAR



henüz port erişimi yok

Program yüklendi: 46 bayt, CS:IP = 0700:0100h. Saat sanaldır: Adım ve Çalıştır kiplerinde yalnız INT 15h beklemeleriyle ilerler, Oynat kipinde gerçek zamanla akar (hız ayarıyla ölçekli). Adım düğmesiyle komutlar tek tek yürütülür; vurgulu satır sıradaki komuttur.

ÖZET

Özet

Aygıt	Port	Anahtar fikir
Trafik lambaları	4	Her bit bir lamba; durum tablosu + INT 15h/86h bekleme.
Step motor	7	Bit 0–2 mıknatıs, bit 7 hazır; yarı adım 11,25°, tam adım 22,5°.
LED ekran	199	Sözcük, işaretli onluk gösterim.
Termometre	125, 127	Okuma + işaretli karşılaştırma + ısıtıcı aç/kapa.
Yazıcı	130	INT 21h/05h; kontrol karakterleri LF, CR, HT, FF.
Robot	9, 10, 11	Komut, veri, durum; meşgul/veri bitlerini yokla.
Test aygıtı	110, 112	Bayt ve sözcük G/Ç denemesi.
8255 PPI (gerçek)	310h–313h	Kontrol sözcüğü (ör. 82h) port yönlerini belirler; port > 255 → DX.

Ortak desen: durum oku → bit test et → hazırsa veri yaz/oku → döngü. IN/OUT yalnızca AL/AX ile, port > 255 ise DX ile.

ÖZET

Alıştırmalar

1. Tüm yönlerde yalnızca sarı lambaları yakan port 4 değeri nedir?
2. 2,5 saniyelik bekleme için CX ve DX ne olmalıdır?
3. LED ekranda −1 göstermek için AX'e hangi onaltılık değer yazılır?
4. Termostatı 20–25 °C aralığına göre değiştirin; sıcaklık −10 °C iken `JB` kullanılsaydı ne olurdu?
5. Step motor programında motoru yalnızca saat yönünde, tam adımla döndürmek için hangi satırlar değişmeli?

▶ **Cevaplar**

Alıştırmalar (devam)

6. Yazıcıya “Merhaba” yazıp sayfayı çıkaran en kısa programı yazın.
7. 8 MHz'lik bir 8086'da `mov cx, N` / `loop` ile yaklaşık 1 ms gecikme için N kaçtır?
8. 8255'te A giriş, B çıkış, C'nin tamamı giriş (mod 0) için kontrol sözcüğü nedir?
9. Ortak anotlu göstergede '5' ve 'H' harfinin kodları nedir (dp g f e d c b a sırası)?

▶ **Cevaplar**

Kaynaklar

- emu8086 yardım belgeleri: *I/O ports and hardware interrupts*, *Virtual devices* ve örnek programlar (traffic_lights.asm, stepper_motor.asm, led_display_test.asm, thermometer.asm, printer.asm, robot.asm).
- Intel, *The 8086 Family User's Manual*, 1979 — IN/OUT komutları, G/Ç adres uzayı ve komut süreleri.
- Intel, *8255A Programmable Peripheral Interface* veri sayfası; Texas Instruments, *ULN2003A* veri sayfası.
- M. Bodur, *Mikroişlemcilere Giriş: Assembler ile Yazılım ve Arayüz*, EMO, 2014.
- R. Brown, *Ralf Brown's Interrupt List* — INT 15h/86h, INT 21h/05h, INT 16h/00h.
- B. B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson, 2009 — Bölüm 11 (Temel G/Ç arayüzü).

Son

Bölüm 10.2: Aygıt Kontrolü

Sonraki bölüm: Bölüm 11: Komut Kodlama ve Dizgi Komutları · Terim sözlüğü