Bölüm 9.2: emu8086.inc Kütüphanesi

Mikroişlemciler

INCLUDE direktifi · Makrolar · PUTC · GOTOXY · PRINT · PRINTN · İmleç makroları · Prosedürler · Sayı okuma / yazma · PTHIS ve yığın · Örnek programlar

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

INCLUDE

Bir kütüphane dosyasını kaynağa eklemeyi ve derleyicinin dosyayı nerede aradığını açıklamak.

Makrolar

`PUTC`, `GOTOXY`, `PRINT`, `PRINTN` gibi makroların neye genişlediğini göstermek.

Prosedürler

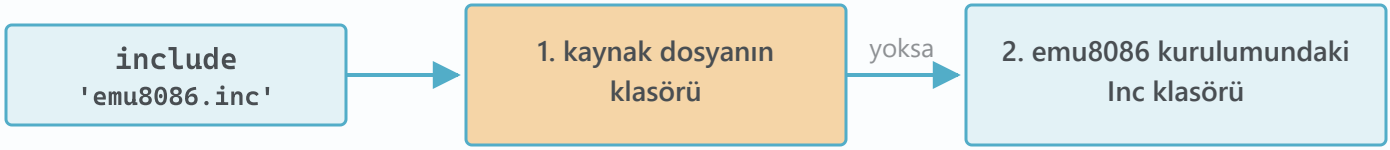
`DEFINE_…` ve `CALL` ile hazır prosedürleri (sayı okuma/yazma, dizgi yazdırma) kullanmak.

Örnek programlar

Kütüphane makro ve prosedürleriyle sayı okuyan, hesaplayan ve ekranı düzenleyen programlar yazmak.

Ortak Fonksiyonlar Kütüphanesi

- Programlamayı kolaylaştırmak için sık kullanılan bazı işlevler hazır olarak bulunur.
- Başka dosyalarda tanımlanan işlevler **INCLUDE** direktifi ile kullanılır.
- `emu8086.inc` içindeki herhangi bir işlevi kullanmak için kaynak dosyanın başına `include 'emu8086.inc'` yazılır.
- Bu kütüphane, emu8086 platformunda sık kullanılan ekran, klavye ve sayı işlevlerini içerir.



bulunan dosyanın içeriği, include satırının yerine eklenir

INCLUDE bir *direktiftir*: işlemci komutu üretmez, derleme sırasında metin ekler.

INCLUDE Sonrası Kaynak

```
; include satırının yerine:
PUTC MACRO char      ; şablon: 0 bayt
  ...
ENDM
GOTOXY MACRO col, row
  ...
ENDM
DEFINE_SCAN_NUM MACRO
  ...                ; prosedür şablonu
ENDM
; programın kendisi:
ORG  100h
PUTC 'A'             ; burada 8 bayt
RET
```

- Assembler `include` satırını dosyanın metniyle değiştirir; ardından kaynağı tek parça olarak derler.
- Makro tanımı yalnızca bir **şablondur**: kullanılmayan makro programa tek bayt eklemez.
- Kod, makronun **kullanıldığı yerde** oluşur: `PUTC 'A'` açılınca 8 baytlık komut dizisi üretir.
- Prosedürler de makroların içinde saklanır (`DEFINE_…`): DEFINE_ satırı yazılmayan prosedürün kodu programa hiç girmez.
- Kütüphanenin tamamını eklemek programı büyütmez; program boyutunu yalnızca kullanılan makrolar ve tanımlanan prosedürler belirler.

Çalışan programda “emu8086.inc” diye bir şey yoktur: bellekte yalnızca açılmış komutlar ve tanımlanmış prosedürler bulunur.

KÜTÜPHANE VE INCLUDE

Kütüphanede Neler Var?

Makrolar · doğrudan kullanılır

Makro	Görev
`PUTC char`	tek karakter yaz
`GOTOXY col, row`	imleci konumlandır
`PRINT string`	dizgi yaz
`PRINTN string`	dizgi yaz + yeni satır
`CURSOROFF`	imleci gizle
`CURSORON`	imleci göster

Prosedürler · DEFINE_ + CALL

Prosedür	Görev
`PRINT_STRING`	DS:SI'deki dizgiyi yaz
`PTHIS`	CALL'dan sonraki dizgiyi yaz
`GET_STRING`	klavyeden dizgi oku
`CLEAR_SCREEN`	ekranı temizle
`SCAN_NUM`	sayı oku → CX
`PRINT_NUM`	AX'i işaretli yaz
`PRINT_NUM_UNS`	AX'i işaretsiz yaz

Makrolar kullanıldıkları yere **kopyalanır**; prosedürler bir kez tanımlanır ve `CALL` ile **çağrılır**.

Ekran Makroları

Makro	Ne yapar?	Arka planda
`PUTC char`	Geçerli imleç konumuna bir ASCII karakter yazar.	`INT 10h`, AH = 0Eh
`GOTOXY col, row`	İmleci verilen sütun ve satıra konumlandırır.	`INT 10h`, AH = 02h
`PRINT string`	Verilen dizgiyi yazar.	0Eh ile karakter karakter
`PRINTN string`	PRINT ile aynı; sonuna CR + LF (13, 10) ekleyerek yeni satıra geçer.	0Eh ile karakter karakter
`CURSOROFF`	Metin imlecini kapatır (gizler).	`INT 10h`, AH = 01h
`CURSORON`	Metin imlecini açar.	`INT 10h`, AH = 01h

Makro parametreleri sabit, yazmaç ya da değişken olabilir: `PUTC 65`, `PUTC 'B'`, `PUTC AL`.

PUTC ve GOTOXY Neye Genişler?

PUTC 'A'

```
PUSH AX         ; AX'i koru
MOV  AL, 'A'    ; parametre
MOV  AH, 0Eh    ; teletype
INT  10h
POP  AX         ; AX geri
```

5 komut, **8 bayt**. Yazmaçlar korunur; çağıran kod etkilenmez.

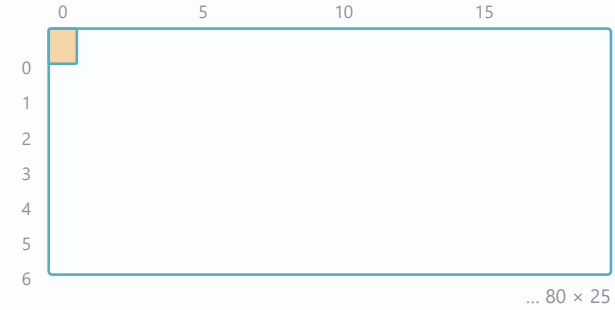
GOTOXY 10, 5

```
PUSH AX
PUSH BX
PUSH DX
MOV  AH, 02h    ; imleç konumla
MOV  DH, 5      ; satır (row)
MOV  DL, 10     ; sütun (col)
MOV  BH, 0      ; sayfa 0
INT  10h
POP  DX
POP  BX
POP  AX
```

Parametre sırası **sütun, satır**dır (x, y); BIOS'ta sütun **DL**'ye, satır **DH**'ye gider. Sol üst köşe (0, 0), sağ alt köşe (79, 24).

Kod İzleme: PRINT, GOTOXY, PUTC

```
▶ include emu8086.inc
2 ORG    100h
3 PRINT 'Hello World!'
4 GOTOXY 10, 5
5 PUTC 65           ; 65 - is an ASCII code for 'A'
6 PUTC 'B'
7 RET               ; return to operating system.
8 END               ; directive to stop the compiler.
```



`include` : makro tanımları derleme sırasında eklenir; kod üretilmez.

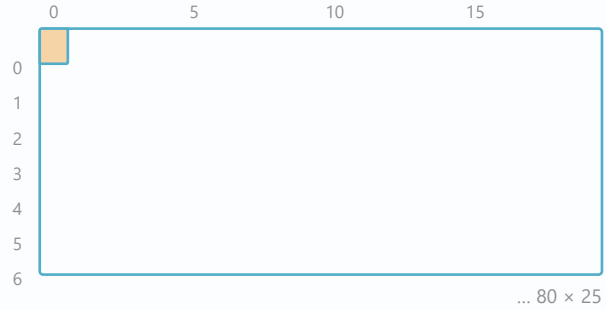
Adım 1 / 7

DEĞİŞKENLER

`sütun = 0` `satır = 0`

Kod İzleme: PRINT, GOTOXY, PUTC

```
1 include emu8086.inc
▶ ORG    100h
3 PRINT 'Hello World!'
4 GOTOXY 10, 5
5 PUTC 65           ; 65 - is an ASCII code for 'A'
6 PUTC 'B'
7 RET               ; return to operating system.
8 END               ; directive to stop the compiler.
```



`ORG 100h` : COM programı 100h'den başlar.

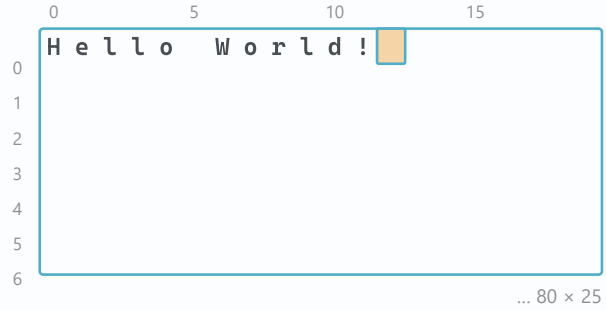
Adım 2 / 7

DEĞİŞKENLER

`sütun = 0` `satır = 0`

Kod İzleme: PRINT, GOTOXY, PUTC

```
1 include emu8086.inc
2 ORG    100h
3 PRINT 'Hello World!'
4 GOTOXY 10, 5
5 PUTC 65           ; 65 - is an ASCII code for 'A'
6 PUTC 'B'
7 RET               ; return to operating system.
8 END               ; directive to stop the compiler.
```



`PRINT` : 12 karakter (0–11. sütun) yazılır; imleç 12. sütunda.

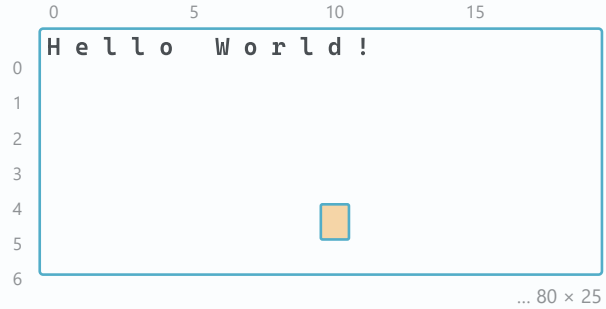
Adım 3 / 7

DEĞİŞKENLER

`sütun = 12` `satır = 0`

Kod İzleme: PRINT, GOTOXY, PUTC

```
1 include emu8086.inc
2 ORG    100h
3 PRINT 'Hello World!'
▶ GOTOXY 10, 5
5 PUTC 65           ; 65 - is an ASCII code for 'A'
6 PUTC 'B'
7 RET               ; return to operating system.
8 END               ; directive to stop the compiler.
```



`GOTOXY 10, 5` : imleç 10. sütun, 5. satıra taşınır. Ekrana bir şey yazılmaz.

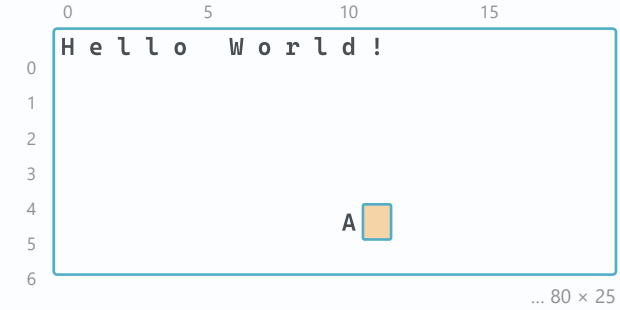
Adım 4 / 7

DEĞİŞKENLER

`sütun = 10` `satır = 5`

Kod İzleme: PRINT, GOTOXY, PUTC

```
1 include emu8086.inc
2 ORG    100h
3 PRINT 'Hello World!'
4 GOTOXY 10, 5
▶ PUTC 65           ; 65 - is an ASCII code for 'A'
6 PUTC 'B'
7 RET               ; return to operating system.
8 END               ; directive to stop the compiler.
```



`PUTC 65` : 65 = 41h = 'A' yazılır, imleç 11. sütuna geçer.

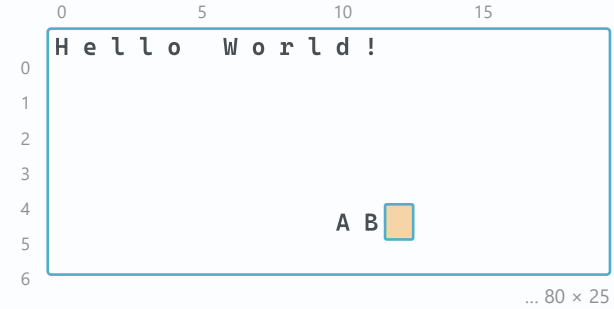
Adım 5 / 7

DEĞİŞKENLER

`sütun = 11` `satır = 5`

Kod İzleme: PRINT, GOTOXY, PUTC

```
1 include emu8086.inc
2 ORG    100h
3 PRINT 'Hello World!'
4 GOTOXY 10, 5
5 PUTC 65           ; 65 - is an ASCII code for 'A'
▶ PUTC 'B'
7 RET               ; return to operating system.
8 END               ; directive to stop the compiler.
```



`PUTC 'B'` : karakter sabiti doğrudan verilebilir.

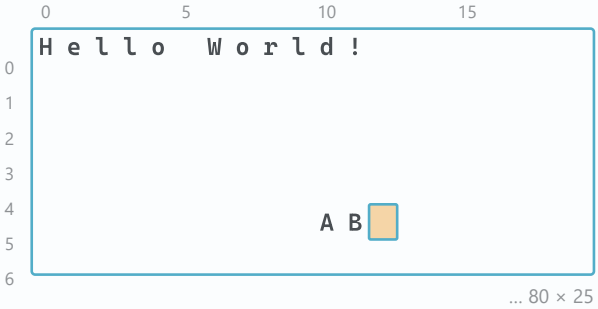
Adım 6 / 7

DEĞİŞKENLER

`sütun = 12` `satır = 5`

Kod İzleme: PRINT, GOTOXY, PUTC

```
1 include emu8086.inc
2 ORG    100h
3 PRINT 'Hello World!'
4 GOTOXY 10, 5
5 PUTC 65           ; 65 - is an ASCII code for 'A'
6 PUTC 'B'
▶ RET               ; return to operating system.
8 END               ; directive to stop the compiler.
```



`RET` : DOS'a dönüş. `END` yalnızca derleyiciye “kaynak bitti” der.

Adım 7 / 7

DEĞİŞKENLER

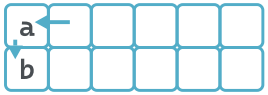
`sütun = 12` `satır = 5`

PRINTN ve İmleç Makroları

PRINT ile PRINTN

`PRINT 'a'` ardından `PRINT 'b'` → `ab` aynı satırda.
`PRINTN 'a'` ardından `PRINT 'b'` → `a` ve alt satırda `b`.

CR (13) imleci satır başına, **LF** (10) bir alt satıra götürür; yeni satır için ikisi birden gerekir.



CR (13): satır başı

LF (10): alt satır

CURSOROFF / CURSORON

İmleç şeklini ayarlayan `INT 10h`, AH = 01h kullanılır: CH = başlangıç, CL = bitiş tarama satırı.
CH'nin 5. biti 1 yapılınca imleç **gizlenir**; CURSORON normal şekli geri yükler.

Oyun veya menü ekranlarında yanıp sönen imleci saklamak için kullanılır.

```
PRINTN 'Ad Soyad:'   ; yazıp alt satıra geç
CURSOROFF
GOTOXY 30, 12
PRINT 'Merhaba'
CURSORON
```

MAKROLAR

PUTC ile Kontrol Karakterleri

Kod	Ad	PUTC ile verilince
`07h`	BEL	Hoparlörden kısa bir bip; imleç yerinden oynamaz.
`08h`	BS (geri)	İmleç bir sola gider; karakteri **silmez**. 0. sütunda etkisizdir.
`0Ah`	LF (satır besleme)	İmleç bir alt satıra iner, **sütun değişmez**. Son satırda ekran bir satır yukarı kayar.
`0Dh`	CR (satır başı)	İmleç aynı satırın 0. sütununa döner; satırdaki yazı yerinde kalır.

Yeni satır = CR + LF

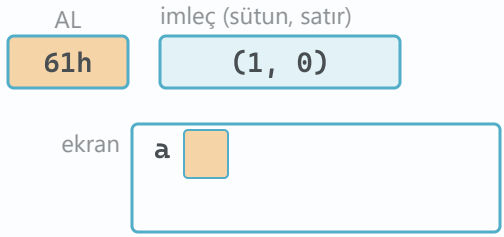
PRINTN'in eklediği `13, 10` çifti bu iki hareketin birleşimidir: önce satır başı, sonra alt satır. Sıra ters olsa da (10, 13) imleç aynı yere varır.

Ekranda silme

Bir karakteri silmek için üçlü kullanılır: `PUTC 8` , `PUTC ' '` , `PUTC 8` . Sola git, boşlukla üstüne yaz, yine sola dön. Kütüphanenin geri silme işleyen prosedürleri de ekranı böyle düzeltir.

Kod İzleme: CR ve LF Ayrı Ayrı

```
1  ORG  100h
▶  MOV  AH, 0Eh     ; teletype
▶  MOV  AL, 'a'
▶  INT  10h
5  MOV  AL, 'b'
6  INT  10h
7  MOV  AL, 10      ; yalnız LF
8  INT  10h
9  MOV  AL, 'c'
10 INT  10h
11 MOV  AL, 13      ; yalnız CR
12 INT  10h
13 MOV  AL, 'd'
14 INT  10h
15 RET
```



AH = 0Eh bu örnekte bir kez yüklenir; `INT 10h` AL'deki 'a'yı yazar ve imleci bir sağa, (1, 0) konumuna götürür.

Adım 1 / 6

DEĞİŞKENLER

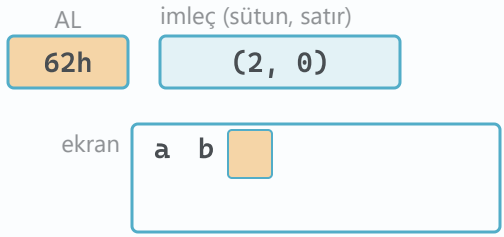
`AL = 61h` `imleç = (1, 0)`

ÇIKTI

```
a
```

Kod İzleme: CR ve LF Ayrı Ayrı

```
1  ORG  100h
2  MOV  AH, 0Eh      ; teletype
3  MOV  AL, 'a'
4  INT  10h
▶  MOV  AL, 'b'
▶  INT  10h
7  MOV  AL, 10       ; yalnız LF
8  INT  10h
9  MOV  AL, 'c'
10 INT  10h
11 MOV  AL, 13       ; yalnız CR
12 INT  10h
13 MOV  AL, 'd'
14 INT  10h
15 RET
```



'b' yazılır; imleç (2, 0).

Adım 2 / 6

DEĞİŞKENLER

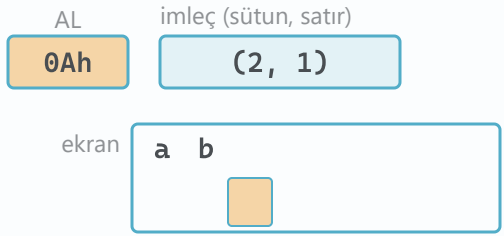
`AL = 62h` `imleç = (2, 0)`

ÇIKTI

```
ab
```

Kod İzleme: CR ve LF Ayrı Ayrı

```
1  ORG  100h
2  MOV  AH, 0Eh      ; teletype
3  MOV  AL, 'a'
4  INT  10h
5  MOV  AL, 'b'
6  INT  10h
▶  MOV  AL, 10       ; yalnız LF
▶  INT  10h
9  MOV  AL, 'c'
10 INT  10h
11 MOV  AL, 13       ; yalnız CR
12 INT  10h
13 MOV  AL, 'd'
14 INT  10h
15 RET
```



Yalnız LF (10): imleç bir alt satıra iner ama sütun 2'de kalır: (2, 1).

Adım 3 / 6

DEĞİŞKENLER

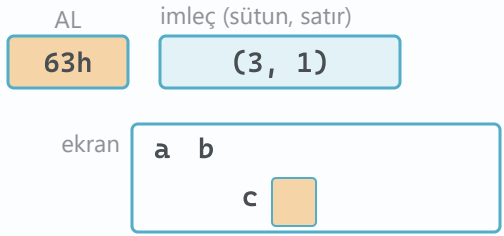
`AL = 0Ah` `imleç = (2, 1)`

ÇIKTI

```
ab
```

Kod İzleme: CR ve LF Ayrı Ayrı

```
1  ORG  100h
2  MOV  AH, 0Eh      ; teletype
3  MOV  AL, 'a'
4  INT  10h
5  MOV  AL, 'b'
6  INT  10h
7  MOV  AL, 10       ; yalnız LF
8  INT  10h
▶  MOV  AL, 'c'
▶  INT  10h
11 MOV  AL, 13       ; yalnız CR
12 INT  10h
13 MOV  AL, 'd'
14 INT  10h
15 RET
```



'c' alt satırın 2. sütununa yazılır: satırlar “merdiven” gibi kayar.

Adım 4 / 6

DEĞİŞKENLER

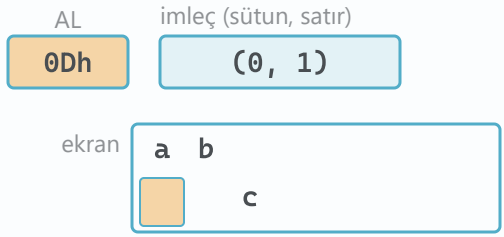
`AL = 63h` `imleç = (3, 1)`

ÇIKTI

```
ab
  c
```

Kod İzleme: CR ve LF Ayrı Ayrı

```
1  ORG  100h
2  MOV  AH, 0Eh      ; teletype
3  MOV  AL, 'a'
4  INT  10h
5  MOV  AL, 'b'
6  INT  10h
7  MOV  AL, 10       ; yalnız LF
8  INT  10h
9  MOV  AL, 'c'
10 INT  10h
▶ MOV  AL, 13       ; yalnız CR
▶ INT  10h
13 MOV  AL, 'd'
14 INT  10h
15 RET
```



Yalnız CR (13): imleç aynı satırın başına döner, (0, 1). Satırdaki c silinmez.

Adım 5 / 6

DEĞİŞKENLER

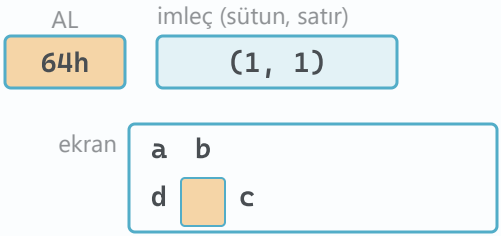
`AL = 0Dh` `imleç = (0, 1)`

ÇIKTI

```
ab
  c
```

Kod İzleme: CR ve LF Ayrı Ayrı

```
1  ORG  100h
2  MOV  AH, 0Eh      ; teletype
3  MOV  AL, 'a'
4  INT  10h
5  MOV  AL, 'b'
6  INT  10h
7  MOV  AL, 10       ; yalnız LF
8  INT  10h
9  MOV  AL, 'c'
10 INT  10h
11 MOV  AL, 13       ; yalnız CR
12 INT  10h
▶ MOV  AL, 'd'
▶ INT  10h
15 RET
```



'd' 0. sütuna yazılır. Ekranda “ab” ve altında “d c”. Düzgün yeni satır için 13 ve 10 birlikte gerekir.

Adım 6 / 6

DEĞİŞKENLER

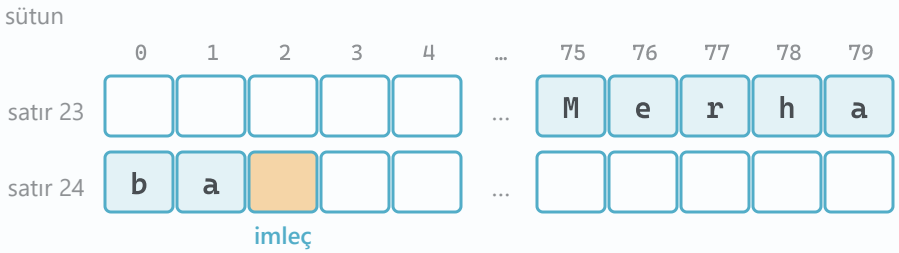
`AL = 64h` `imleç = (1, 1)`

ÇIKTI

```
ab
d c
```

Satır Sonu ve Ekran Kayması

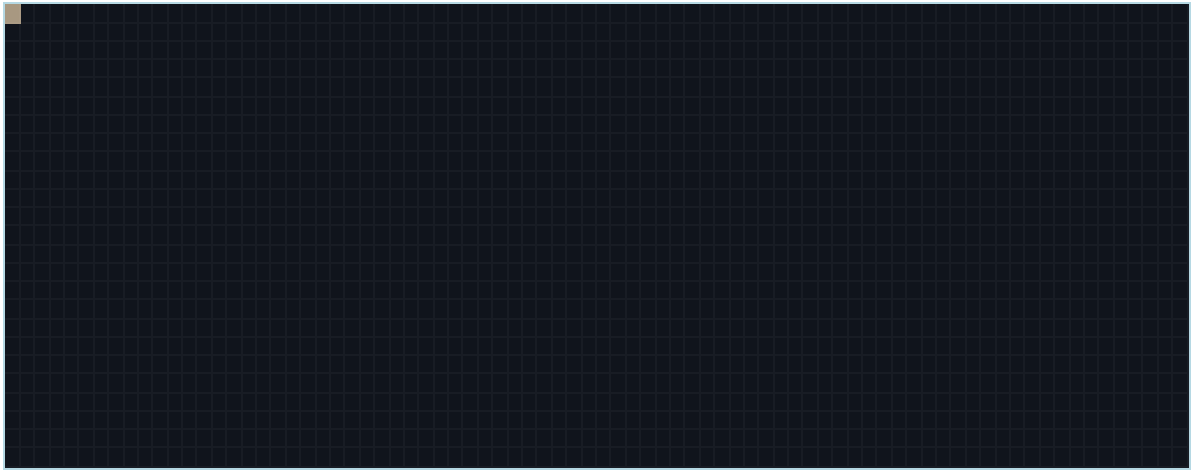
- 0Eh hizmeti 79. sütuna yazdıktan sonra imleci kendiliğinden alt satırın başına geçirir; PRINT uzun bir dizgiyi bu yüzden kesmeden sürdürür.
- İmleç son satırdayken (24) alt satıra geçmek gerekirse ekranın tamamı **bir satır yukarı kayar**: en üst satır kaybolur, en alta boş bir satır gelir.
- Örnek: `GOTOXY 75, 24` ardından `PRINT 'Merhaba'`. "Merha" 75–79. sütunlara yazılır; "ba" için ekran kayar. Sonuçta "Merha" 23. satırda, "ba" 24. satırın başında, imleç (2, 24) konumundadır.
- GOTOXY ve PUTC sınır denetimi yapmaz: konumlar 0–79 (sütun) ve 0–24 (satır) aralığında tutulmalıdır.



Demo: GOTOXY Ekranı

80 × 25 metin ekranında bir hücre seçilince imleci oraya götüren makro, BIOS yazmaçları ve video belleğindeki karşılığı gösterilir.

PUTC karakteri A temizle

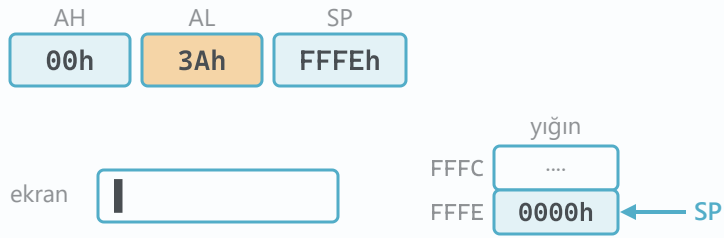


GOTOXY 0, 0 → MOV DL,0 · MOV DH,0 · AH=02h · INT 10h

video belleği: B800:0000h (karakter), B800:0001h (renk)

Kod İzleme: AAM ve PUTC ile İki Basamak

```
1  ORG  100h
▶  MOV  AL, 58
3  AAM              ; AH = 5, AL = 8
4  ADD  AX, 3030h   ; '5', '8'
5  PUSH AX          ; PUTC AH açılımı
6  MOV  AL, AH
7  MOV  AH, 0Eh
8  INT  10h
9  POP  AX          ; AX geri: 3538h
10 PUSH AX          ; PUTC AL açılımı
11 MOV  AL, AL
12 MOV  AH, 0Eh
13 INT  10h
14 POP  AX
15 RET
```



AL = 58 (3Ah). Hedef: sayıyı iki onluk basamak olarak yazmak.

Adım 1 / 7

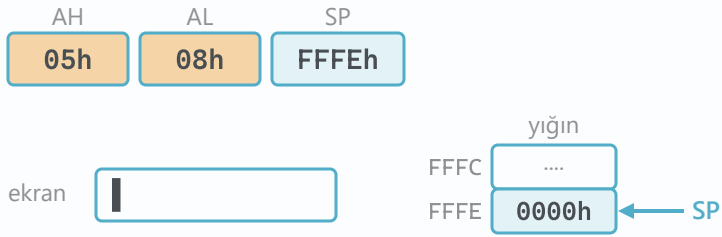
DEĞİŞKENLER

AH = **00h** AL = **3Ah** SP = **FFFEh**

ÇIKTI

Kod İzleme: AAM ve PUTC ile İki Basamak

```
1  ORG  100h
2  MOV  AL, 58
▶  AAM              ; AH = 5, AL = 8
4  ADD  AX, 3030h   ; '5', '8'
5  PUSH AX          ; PUTC AH açılımı
6  MOV  AL, AH
7  MOV  AH, 0Eh
8  INT  10h
9  POP  AX          ; AX geri: 3538h
10 PUSH AX          ; PUTC AL açılımı
11 MOV  AL, AL
12 MOV  AH, 0Eh
13 INT  10h
14 POP  AX
15 RET
```



`AAM` AL'yi 10'a böler: AH = 5 (onlar), AL = 8 (birler). Yalnızca AL bölündüğü için sayı 0–99 aralığında olmalıdır.

Adım 2 / 7

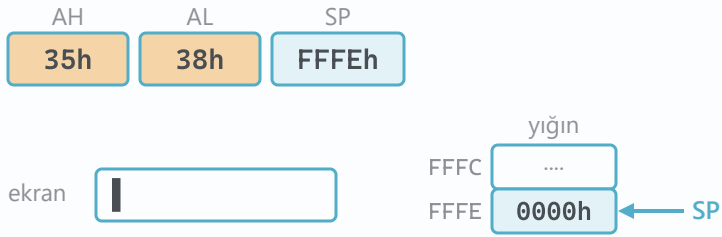
DEĞİŞKENLER

AH = **05h** AL = **08h** SP = **FFFEh**

ÇIKTI

Kod İzleme: AAM ve PUTC ile İki Basamak

```
1  ORG  100h
2  MOV  AL, 58
3  AAM              ; AH = 5, AL = 8
▶ ADD  AX, 3030h    ; '5', '8'
5  PUSH AX          ; PUTC AH açılımı
6  MOV  AL, AH
7  MOV  AH, 0Eh
8  INT  10h
9  POP  AX          ; AX geri: 3538h
10 PUSH AX          ; PUTC AL açılımı
11 MOV  AL, AL
12 MOV  AH, 0Eh
13 INT  10h
14 POP  AX
15 RET
```



`ADD AX, 3030h` iki bayta birden 30h ekler: AH = '5' (35h), AL = '8' (38h).

Adım 3 / 7

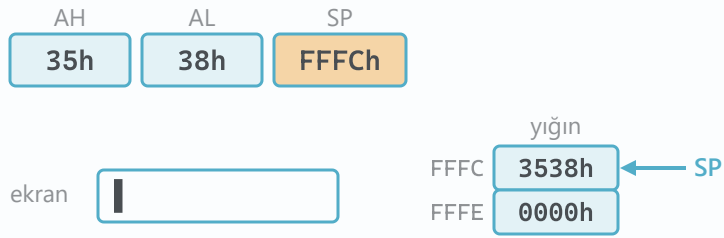
DEĞİŞKENLER

AH = **35h** AL = **38h** SP = **FFFEh**

ÇIKTI

Kod İzleme: AAM ve PUTC ile İki Basamak

```
1  ORG  100h
2  MOV  AL, 58
3  AAM              ; AH = 5, AL = 8
4  ADD  AX, 3030h   ; '5', '8'
5  PUSH AX          ; PUTC AH açılımı
6  MOV  AL, AH
7  MOV  AH, 0Eh
8  INT  10h
9  POP  AX          ; AX geri: 3538h
10 PUSH AX          ; PUTC AL açılımı
11 MOV  AL, AL
12 MOV  AH, 0Eh
13 INT  10h
14 POP  AX
15 RET
```



`PUTC AH` açılımı: önce AX (3538h) yığına, SP = FFFCh.

Adım 4 / 7

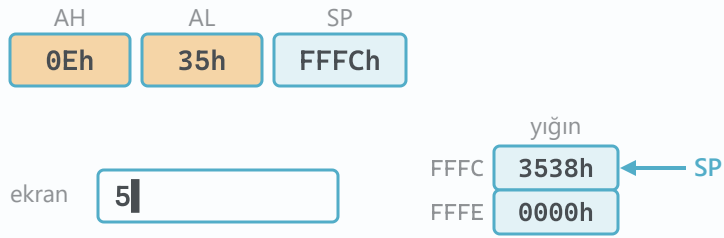
DEĞİŞKENLER

AH = **35h** AL = **38h** SP = **FFFCh**

ÇIKTI

Kod İzleme: AAM ve PUTC ile İki Basamak

```
1  ORG  100h
2  MOV  AL, 58
3  AAM              ; AH = 5, AL = 8
4  ADD  AX, 3030h   ; '5', '8'
5  PUSH AX          ; PUTC AH açılımı
▶  MOV  AL, AH
▶  MOV  AH, 0Eh
▶  INT  10h
9  POP  AX          ; AX geri: 3538h
10 PUSH AX          ; PUTC AL açılımı
11 MOV  AL, AL
12 MOV  AH, 0Eh
13 INT  10h
14 POP  AX
15 RET
```



`MOV AL, AH` ve `MOV AH, 0Eh` : AX = 0E35h, birler basamağı yazmaçtan silindi. `INT 10h` ekrana 5 yazar.

Adım 5 / 7

DEĞİŞKENLER

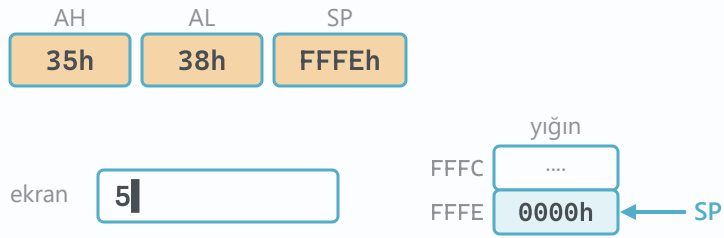
`AH = 0Eh` `AL = 35h` `SP = FFFCh`

ÇIKTI

```
5
```

Kod İzleme: AAM ve PUTC ile İki Basamak

```
1  ORG  100h
2  MOV  AL, 58
3  AAM              ; AH = 5, AL = 8
4  ADD  AX, 3030h   ; '5', '8'
5  PUSH AX          ; PUTC AH açılımı
6  MOV  AL, AH
7  MOV  AH, 0Eh
8  INT  10h
▶  POP  AX          ; AX geri: 3538h
10 PUSH AX          ; PUTC AL açılımı
11 MOV  AL, AL
12 MOV  AH, 0Eh
13 INT  10h
14 POP  AX
15 RET
```



`POP AX` : AX = 3538h geri geldi; '8' yeniden AL'de.

Adım 6 / 7

DEĞİŞKENLER

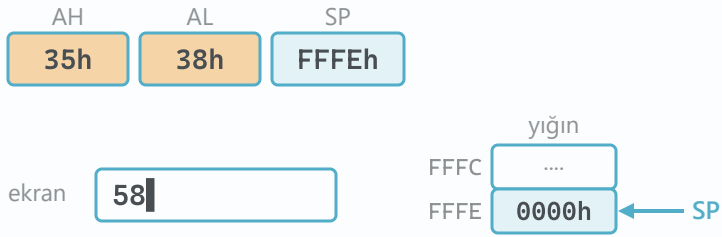
AH = **35h** AL = **38h** SP = **FFFEh**

ÇIKTI

```
5
```

Kod İzleme: AAM ve PUTC ile İki Basamak

```
1  ORG  100h
2  MOV  AL, 58
3  AAM              ; AH = 5, AL = 8
4  ADD  AX, 3030h   ; '5', '8'
5  PUSH AX          ; PUTC AH açılımı
6  MOV  AL, AH
7  MOV  AH, 0Eh
8  INT  10h
9  POP  AX          ; AX geri: 3538h
▶ PUSH AX          ; PUTC AL açılımı
▶ MOV  AL, AL
▶ MOV  AH, 0Eh
▶ INT  10h
▶ POP  AX
15 RET
```



`PUTC AL` açılımı aynı kalıpla çalışır (`MOV AL, AL` zararsızdır). Ekranda 58; AX yine 3538h. PUTC AX'i koruduğu için iki basamak ayrı bir yazmaca kopyalanmadan yazılabildi.

Adım 7 / 7

DEĞİŞKENLER

AH = **35h** AL = **38h** SP = **FFFEh**

ÇIKTI

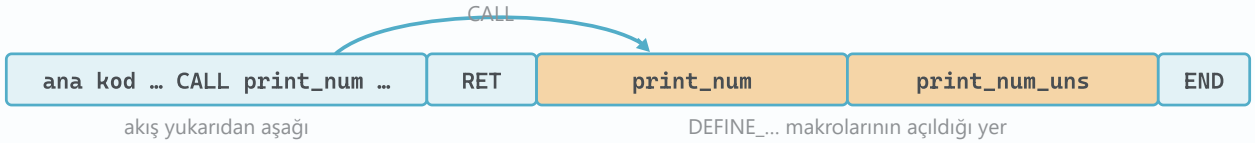
```
58
```

Prosedür Kullanımı: DEFINE_ ve CALL

```
include 'emu8086.inc'
ORG 100h
    ...
    CALL print_num    ; kullan
    ...
RET

DEFINE_PRINT_NUM     ; tanımla
DEFINE_PRINT_NUM_UNS
END
```

- Her prosedür, `END` direktifinden önce kendi `DEFINE_...` makrosuyla **bir kez** tanımlanmalıdır.
- `DEFINE_...` makrosu, prosedürün kodunu bulunduğu yere açar.
- Tanımlar genellikle `RET` ile `END` arasına konur. Her DEFINE_ makrosu gövdenin üstünden atlayan bir `JMP` ile başlar; akış prosedüre kendiliğinden girmez.
- Kodda prosedür adıyla `CALL` yapılır; adlar küçük harfle yazılır (`print_num`).
- `PRINT_NUM` , işini yaparken `PRINT_NUM_UNS` 'yi çağırır; ikisi birlikte tanımlanmalıdır.



Girdi ve Çıktı Yazmaçları

Prosedür	Girdi	Çıktı	Ayrıntı
`PRINT_STRING`	DS:SI = dizgi adresi	—	dizgi 0 ile bitmeli
`PTHIS`	dizgi CALL'ın hemen ardında	—	dönüş adresini kendisi ilerletir
`GET_STRING`	DS:DI = tampon, DX = boyut	tamponda 0 ile biten dizgi	en fazla DX − 1 karakter
`CLEAR_SCREEN`	—	—	imleç (0, 0) konumuna gider
`SCAN_NUM`	klavye	**CX** (işaretli)	CX'teki eski değer kaybolur
`PRINT_NUM`	AX (işaretli)	—	PRINT_NUM_UNS de tanımlı olmalı
`PRINT_NUM_UNS`	AX (işaretsiz)	—	0 … 65 535

Kütüphane prosedürleri kullandıkları diğer yazmaçları girişte yığına saklar, çıkışta geri yükler; değişen tek yazmaç çıktı yazmacıdır. CX'te bir döngü sayacı tutan program, SCAN_NUM'ı çağırmadan önce sayacı kendisi saklamalıdır (`PUSH CX` … `POP CX`).

PROSEDÜRLER

Dizgi Prosedürleri

PRINT_STRING

- Null (0) ile sonlanan dizgiyi geçerli imleç konumuna yazar.
- Dizginin adresi **DS:SI**'den alınır.
- Tanım: `DEFINE_PRINT_STRING`.

```
LEA  SI, msg
CALL print_string
msg DB 'Merhaba', 0
```

PTHIS

- Null ile sonlanan dizgiyi yazar.
- Dizginin adresini **yığından** alır: dizgi, `CALL` 'dan hemen sonra tanımlanır.
- Tanım: `DEFINE_PTHIS`.

```
CALL pthis
DB 'Merhaba', 0
; buradan devam eder
```

PRINT makrosu her kullanımda kodu kopyalar; PRINT_STRING ve PTHIS ise tek kopya kodla istenen sayıda dizgi yazar.

Dizgi Yazdırma Yolları

Yöntem	Dizgi nerede?	Sonlandırıcı	Adres nasıl verilir?	Uygun olduğu durum
`PRINT 'metin'`	kodun içinde (makro açılımı)	0 (makro ekler)	makro kendisi alır (CS:SI)	bir kez yazılan sabit ileti
`PRINT_STRING`	veri alanında	0	DS:SI	birden çok yerde yazılan ya da çalışırken oluşan dizgi
`PTHIS`	CALL'ın hemen ardında	0	yığındaki dönüş adresi	çok sayıda kısa sabit ileti
`INT 21h`, AH = 09h	veri alanında	`'$'`	DS:DX	kütüphanesiz DOS programı

Sonlandırıcı yazılamaz

'$' ile biten dizgide '$' karakteri, 0 ile biten dizgide 0 baytı yazdırılamaz; dizgi orada biter.

Tampon ve yöntem uyumu

GET_STRING'in doldurduğu tampon 0 ile biter: doğrudan PRINT_STRING ile yazılır. AH = 09h ile yazılırsa '$' bulunana kadar bellekteki baytlar ekrana dökülür.

PROSEDÜRLER

GET_STRING ve CLEAR_SCREEN

GET_STRING

- Kullanıcıdan null ile sonlanan bir dizgi alır.
- Dizgi **DS:DI**'deki tampona (buffer) yazılır.
- **DX** = tampon boyutu; sona 0 eklendiği için en fazla DX − 1 karakter alınır.
- **Enter** tuşuna basılınca giriş durur.
- Tanım: `DEFINE_GET_STRING` .

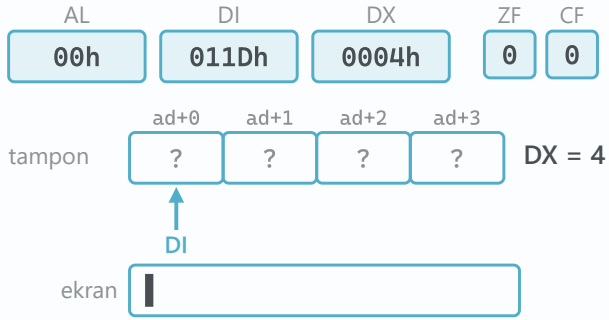
CLEAR_SCREEN

- Ekranı temizler ve imleci en üste (0, 0) taşır.
- Tanım: `DEFINE_CLEAR_SCREEN` .

```
LEA  DI, ad          ; tampon
MOV  DX, 20          ; boyut
CALL get_string
CALL clear_screen
LEA  SI, ad
CALL print_string
RET
ad DB 20 DUP(0)
```

Kod İzleme: GET_STRING Mantığı (DX = 4)

```
▶ LEA  DI, ad       ; tampon adresi
▶ MOV  DX, 4        ; tampon boyutu
3  oku: CMP DX, 1   ; 0 için 1 bayt kalsın
4  JBE  bitti
5  MOV  AH, 01h
6  INT  21h         ; AL = tuş (yankılı)
7  CMP  AL, 13      ; Enter mi?
8  JE   bitti
9  MOV  [DI], AL    ; tampona yaz
10 INC  DI
11 DEC  DX
12 JMP  oku
13 bitti: MOV BYTE PTR [DI], 0
14 RET
15 ad   DB 4 DUP(?)
```



GET_STRING okuma döngüsünün sadeleştirilmiş eşdeğeri (kütüphanedeki sürüm geri silmeyi de işler). DI = tampon, DX = 4 bayt; kullanıcı “Ece” yazacak.

Adım 1 / 8

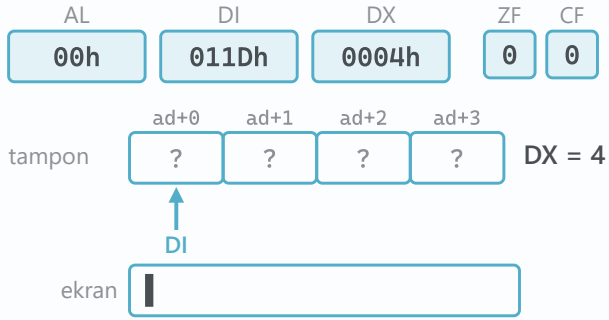
DEĞİŞKENLER

AL = **00h** DI = **ad+0** DX = **4**

ÇIKTI

Kod İzleme: GET_STRING Mantığı (DX = 4)

```
1  LEA  DI, ad       ; tampon adresi
2  MOV  DX, 4        ; tampon boyutu
▶ oku: CMP DX, 1     ; 0 için 1 bayt kalsın
▶ JBE  bitti
5  MOV  AH, 01h
6  INT  21h          ; AL = tuş (yankılı)
7  CMP  AL, 13       ; Enter mi?
8  JE   bitti
9  MOV  [DI], AL     ; tampona yaz
10 INC  DI
11 DEC  DX
12 JMP  oku
13 bitti: MOV BYTE PTR [DI], 0
14 RET
15 ad   DB 4 DUP(?)
```



4 > 1: tamponda karakter ve sondaki 0 için yer var → `JBE` atlamaz.

Adım 2 / 8

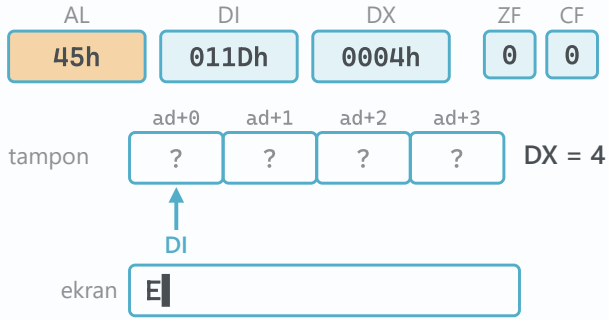
DEĞİŞKENLER

`AL = 00h` `DI = ad+0` `DX = 4`

ÇIKTI

Kod İzleme: GET_STRING Mantığı (DX = 4)

```
1  LEA  DI, ad       ; tampon adresi
2  MOV  DX, 4        ; tampon boyutu
3  oku: CMP DX, 1    ; 0 için 1 bayt kalsın
4  JBE  bitti
▶  MOV  AH, 01h
▶  INT  21h          ; AL = tuş (yankılı)
▶  CMP  AL, 13       ; Enter mi?
▶  JE   bitti
9  MOV  [DI], AL     ; tampona yaz
10 INC  DI
11 DEC  DX
12 JMP  oku
13 bitti: MOV BYTE PTR [DI], 0
14 RET
15 ad   DB 4 DUP(?)
```



`INT 21h`, AH = 01h: 'E' okunup ekrana yankılanır. Enter (13) değil → `JE` atlamaz.

Adım 3 / 8

DEĞİŞKENLER

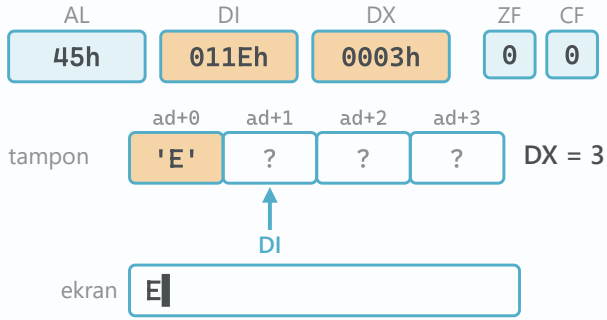
AL = **45h** | DI = **ad+0** | DX = **4**

ÇIKTI

```
E
```

Kod İzleme: GET_STRING Mantığı (DX = 4)

```
1  LEA  DI, ad       ; tampon adresi
2  MOV  DX, 4        ; tampon boyutu
3  oku: CMP DX, 1    ; 0 için 1 bayt kalsın
4  JBE  bitti
5  MOV  AH, 01h
6  INT  21h          ; AL = tuş (yankılı)
7  CMP  AL, 13       ; Enter mi?
8  JE   bitti
▶  MOV  [DI], AL     ; tampona yaz
▶  INC  DI
▶  DEC  DX
12 JMP  oku
13 bitti: MOV BYTE PTR [DI], 0
14 RET
15 ad   DB 4 DUP(?)
```



'E' tampona yazılır; DI bir ilerler, kalan yer DX = 3.

Adım 4 / 8

DEĞİŞKENLER

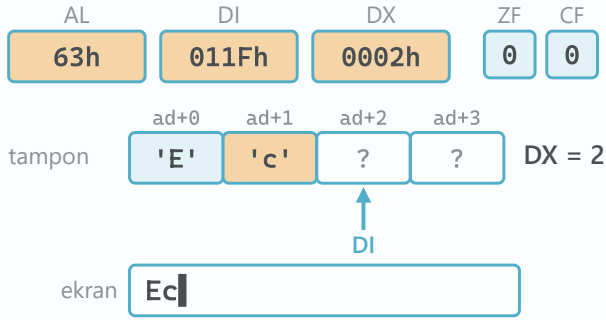
AL = 45h | DI = ad+1 | DX = 3

ÇIKTI

```
E
```

Kod İzleme: GET_STRING Mantığı (DX = 4)

```
1  LEA  DI, ad       ; tampon adresi
2  MOV  DX, 4        ; tampon boyutu
   oku: CMP DX, 1    ; 0 için 1 bayt kalsın
   JBE  bitti
   MOV  AH, 01h
   INT  21h          ; AL = tuş (yankılı)
   CMP  AL, 13       ; Enter mi?
   JE   bitti
   MOV  [DI], AL     ; tampona yaz
   INC  DI
   DEC  DX
   JMP  oku
13 bitti: MOV BYTE PTR [DI], 0
14 RET
15 ad   DB 4 DUP(?)
```



İkinci tur: 'c' okunur ve ad+1'e yazılır; DX = 2.

Adım 5 / 8

DEĞİŞKENLER

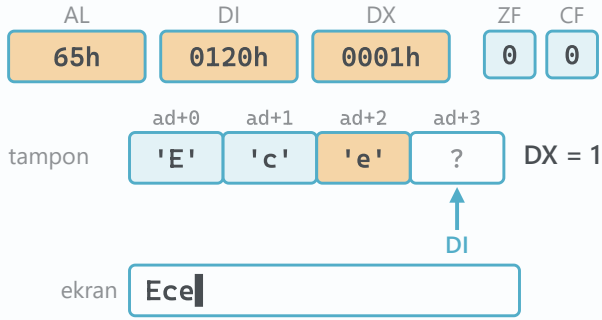
`AL = 63h` `DI = ad+2` `DX = 2`

ÇIKTI

```
Ec
```

Kod İzleme: GET_STRING Mantığı (DX = 4)

```
1  LEA  DI, ad       ; tampon adresi
2  MOV  DX, 4        ; tampon boyutu
   oku: CMP DX, 1    ; 0 için 1 bayt kalsın
   JBE  bitti
   MOV  AH, 01h
   INT  21h          ; AL = tuş (yankılı)
   CMP  AL, 13       ; Enter mi?
   JE   bitti
   MOV  [DI], AL     ; tampona yaz
   INC  DI
   DEC  DX
   JMP  oku
13 bitti: MOV BYTE PTR [DI], 0
14 RET
15 ad   DB 4 DUP(?)
```



Üçüncü tur: 'e' ad+2'ye; DX = 1.

Adım 6 / 8

DEĞİŞKENLER

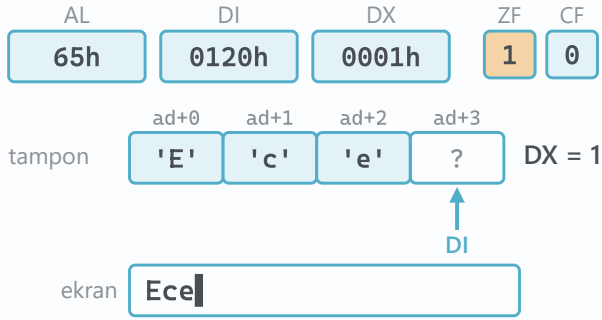
AL = 65h | DI = ad+3 | DX = 1

ÇIKTI

```
Ece
```

Kod İzleme: GET_STRING Mantığı (DX = 4)

```
1  LEA  DI, ad       ; tampon adresi
2  MOV  DX, 4        ; tampon boyutu
▶ oku: CMP DX, 1     ; 0 için 1 bayt kalsın
▶ JBE  bitti
5  MOV  AH, 01h
6  INT  21h          ; AL = tuş (yankılı)
7  CMP  AL, 13       ; Enter mi?
8  JE   bitti
9  MOV  [DI], AL     ; tampona yaz
10 INC  DI
11 DEC  DX
▶ JMP  oku
13 bitti: MOV BYTE PTR [DI], 0
14 RET
15 ad   DB 4 DUP(?)
```



DX = 1: yalnızca sondaki 0 için yer kaldı → `JBE bitti`. Kullanıcı daha yazsa da okunmaz: en fazla DX − 1 = 3 karakter.

Adım 7 / 8

DEĞİŞKENLER

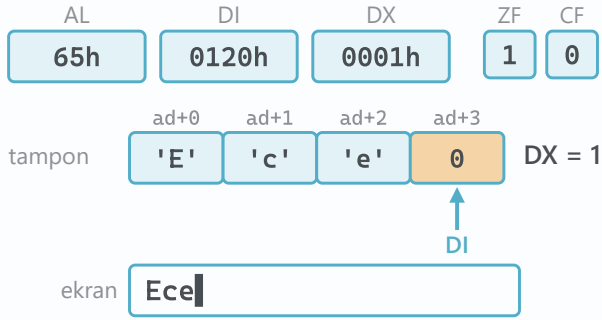
AL = **65h** | DI = **ad+3** | DX = **1**

ÇIKTI

```
Ece
```

Kod İzleme: GET_STRING Mantığı (DX = 4)

```
1  LEA  DI, ad       ; tampon adresi
2  MOV  DX, 4        ; tampon boyutu
3  oku: CMP DX, 1    ; 0 için 1 bayt kalsın
4  JBE  bitti
5  MOV  AH, 01h
6  INT  21h          ; AL = tuş (yankılı)
7  CMP  AL, 13       ; Enter mi?
8  JE   bitti
9  MOV  [DI], AL     ; tampona yaz
10 INC  DI
11 DEC  DX
12 JMP  oku
▶ bitti: MOV BYTE PTR [DI], 0
14 RET
15 ad   DB 4 DUP(?)
```



Dizgi 0 ile sonlandırılır: tampon 'E','c','e',0 → PRINT_STRING ile yazdırılabilir.

Adım 8 / 8

DEĞİŞKENLER

AL = **65h** | DI = **ad+3** | DX = **1**

ÇIKTI

```
Ece
```

PROSEDÜRLER

GET_STRING Ayrıntıları

Yankı denetimi

Kütüphanedeki sürüm tuşu yankısız okur (`INT 16h`, AH = 0) ve ekrana yalnızca tampona aldığı karakteri yazar. Böylece kabul edilmeyen tuşlar ekranda iz bırakmaz.

Dolu tampon

DX − 1 karakter alındıktan sonra yeni harfler yok sayılır; prosedür Enter ya da geri silme bekler. Sadeleştirilmiş izlemedeki gibi kendiliğinden bitmez.

Geri silme (8)

Tamponda harf varsa DI bir geri alınır ve ekrana 8, ' ', 8 yazılır. Tampon boşsa tuş yok sayılır: imleç girdinin başından geri gitmez.

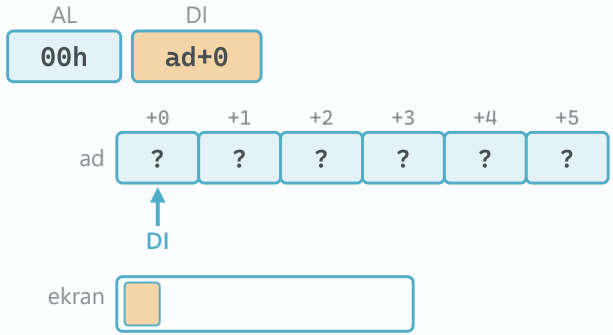
Enter (13)

Tampona yazılmaz; yerine 0 konur ve prosedür döner. DX ≤ 1 verilirse hiç okuma yapılmaz, çünkü yalnızca sondaki 0'a yer vardır.

GET_STRING'den sonra imleç girilen metnin hemen sağında kalır; sonraki çıktı aynı satırda başlar. Alt satıra geçmek için `PUTC 13` ve `PUTC 10` eklenir.

Kod İzleme: Geri Silmeyi İşleyen Okuma

```
 1 ORG  100h
 2 LEA  DI, ad
 3 oku: MOV AH, 0
 4 INT  16h          ; AL = tuş (yankısız)
 5 CMP  AL, 13       ; Enter mi?
 6 JE   bitti
 7 MOV  AH, 0Eh
 8 CMP  AL, 8        ; geri silme mi?
 9 JNE  ekle
10 CMP  DI, OFFSET ad
11 JE   oku          ; silinecek harf yok
12 DEC  DI
13 INT  10h          ; 8: imleç sola
14 MOV  AL, ' '
15 INT  10h          ; harfin yerine boşluk
16 MOV  AL, 8
17 INT  10h          ; yine sola
18 JMP  oku
19 ekle: MOV [DI], AL
20 INC  DI
21 INT  10h          ; yankı
22 JMP  oku
23 bitti: MOV BYTE PTR [DI], 0
24 RET
25 ad   DB 6 DUP(?)
```



Tampon `ad` 6 bayt; DI ilk baytı gösterir. Kullanıcı sırasıyla A, b, geri silme, c ve Enter tuşlarına basacak.

Adım 1 / 9

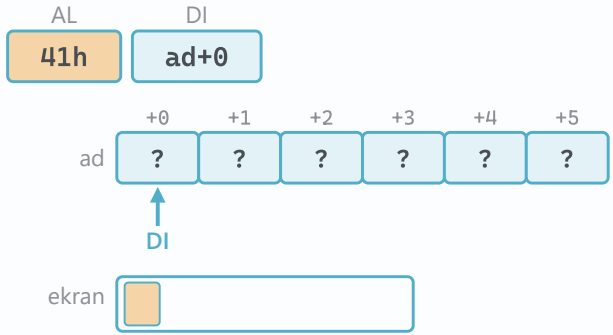
DEĞİŞKENLER

`AL = 00h` `DI = ad+0` `imleç = 0`

ÇIKTI

Kod İzleme: Geri Silmeyi İşleyen Okuma

```
 1 ORG  100h
 2 LEA  DI, ad
 ▶ oku: MOV AH, 0
 ▶ INT  16h          ; AL = tuş (yankısız)
 ▶ CMP  AL, 13       ; Enter mi?
 ▶ JE   bitti
 ▶ MOV  AH, 0Eh
 ▶ CMP  AL, 8        ; geri silme mi?
 ▶ JNE  ekle
10 CMP  DI, OFFSET ad
11 JE   oku          ; silinecek harf yok
12 DEC  DI
13 INT  10h          ; 8: imleç sola
14 MOV  AL, ' '
15 INT  10h          ; harfin yerine boşluk
16 MOV  AL, 8
17 INT  10h          ; yine sola
18 JMP  oku
19 ekle: MOV [DI], AL
20 INC  DI
21 INT  10h          ; yankı
22 JMP  oku
23 bitti: MOV BYTE PTR [DI], 0
24 RET
25 ad   DB 6 DUP(?)
```



`INT 16h` : AL = 'A' (41h); tuş yankısız okunduğu için ekranda henüz bir şey yok. Enter (13) ve geri silme (8) değil: `ekle` .

Adım 2 / 9

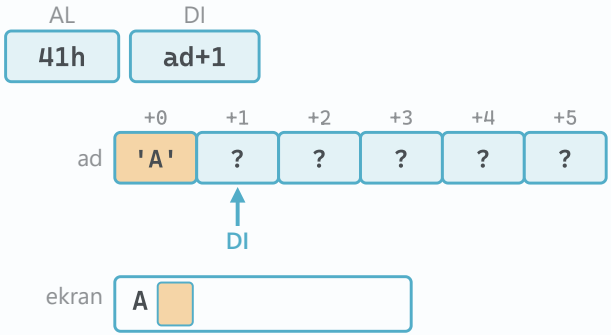
DEĞİŞKENLER

`AL = 41h` `DI = ad+0` `imleç = 0`

ÇIKTI

Kod İzleme: Geri Silmeyi İşleyen Okuma

```
 1 ORG  100h
 2 LEA  DI, ad
 3 oku: MOV AH, 0
 4 INT  16h          ; AL = tuş (yankısız)
 5 CMP  AL, 13       ; Enter mi?
 6 JE   bitti
 7 MOV  AH, 0Eh
 8 CMP  AL, 8        ; geri silme mi?
 9 JNE  ekle
10 CMP  DI, OFFSET ad
11 JE   oku          ; silinecek harf yok
12 DEC  DI
13 INT  10h          ; 8: imleç sola
14 MOV  AL, ' '
15 INT  10h          ; harfin yerine boşluk
16 MOV  AL, 8
17 INT  10h          ; yine sola
18 JMP  oku
   ekle: MOV [DI], AL
   INC  DI
   INT  10h          ; yankı
22 JMP  oku
23 bitti: MOV BYTE PTR [DI], 0
24 RET
25 ad   DB 6 DUP(?)
```



'A' tampona yazılır, DI ilerler; ardından yankı: ekranda A.

Adım 3 / 9

DEĞİŞKENLER

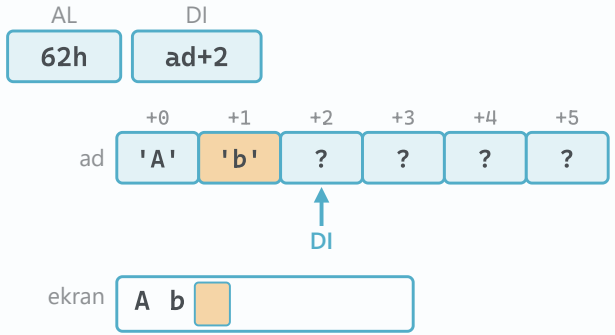
AL = 41h | DI = ad+1 | imleç = 1

ÇIKTI

```
A
```

Kod İzleme: Geri Silmeyi İşleyen Okuma

```
 1 ORG  100h
 2 LEA  DI, ad
 ▶ oku: MOV AH, 0
 ▶ INT  16h          ; AL = tuş (yankısız)
 ▶ CMP  AL, 13       ; Enter mi?
 ▶ JE   bitti
 ▶ MOV  AH, 0Eh
 ▶ CMP  AL, 8        ; geri silme mi?
 ▶ JNE  ekle
10 CMP  DI, OFFSET ad
11 JE   oku          ; silinecek harf yok
12 DEC  DI
13 INT  10h          ; 8: imleç sola
14 MOV  AL, ' '
15 INT  10h          ; harfin yerine boşluk
16 MOV  AL, 8
17 INT  10h          ; yine sola
18 JMP  oku
 ▶ ekle: MOV [DI], AL
 ▶ INC  DI
 ▶ INT  10h          ; yankı
 ▶ JMP  oku
23 bitti: MOV BYTE PTR [DI], 0
24 RET
25 ad   DB 6 DUP(?)
```



Aynı yol 'b' için: tampon A, b; DI = ad+2; ekranda Ab.

Adım 4 / 9

DEĞİŞKENLER

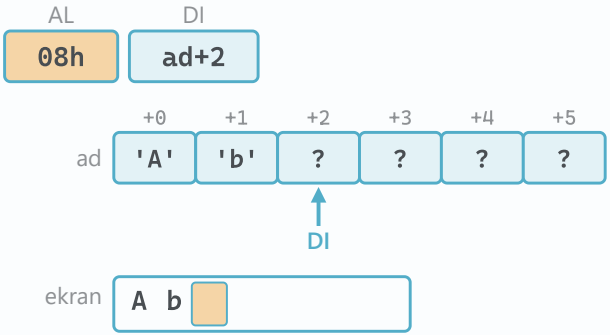
AL = 62h | DI = ad+2 | imleç = 2

ÇIKTI

```
Ab
```

Kod İzleme: Geri Silmeyi İşleyen Okuma

```
 1 ORG  100h
 2 LEA  DI, ad
 3 oku: MOV AH, 0
 4 INT  16h          ; AL = tuş (yankısız)
 5 CMP  AL, 13       ; Enter mi?
 6 JE   bitti
 7 MOV  AH, 0Eh
 8 CMP  AL, 8        ; geri silme mi?
 9 JNE  ekle
10 CMP  DI, OFFSET ad
11 JE   oku          ; silinecek harf yok
12 DEC  DI
13 INT  10h          ; 8: imleç sola
14 MOV  AL, ' '
15 INT  10h          ; harfin yerine boşluk
16 MOV  AL, 8
17 INT  10h          ; yine sola
18 JMP  oku
19 ekle: MOV [DI], AL
20 INC  DI
21 INT  10h          ; yankı
22 JMP  oku
23 bitti: MOV BYTE PTR [DI], 0
24 RET
25 ad   DB 6 DUP(?)
```



Geri silme: AL = 08h. `CMP AL, 8` → ZF = 1, `JNE ekle` atlamaz.

Adım 5 / 9

DEĞİŞKENLER

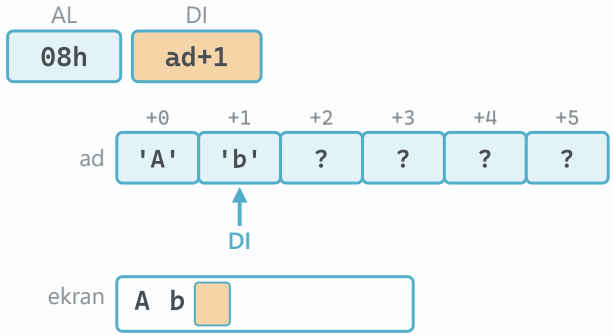
AL = **08h** | DI = **ad+2** | imleç = **2**

ÇIKTI

```
Ab
```

Kod İzleme: Geri Silmeyi İşleyen Okuma

```
 1 ORG  100h
 2 LEA  DI, ad
 3 oku: MOV AH, 0
 4 INT  16h          ; AL = tuş (yankısız)
 5 CMP  AL, 13       ; Enter mi?
 6 JE   bitti
 7 MOV  AH, 0Eh
 8 CMP  AL, 8        ; geri silme mi?
 9 JNE  ekle
▶ CMP  DI, OFFSET ad
▶ JE   oku          ; silinecek harf yok
▶ DEC  DI
13 INT  10h          ; 8: imleç sola
14 MOV  AL, ' '
15 INT  10h          ; harfin yerine boşluk
16 MOV  AL, 8
17 INT  10h          ; yine sola
18 JMP  oku
19 ekle: MOV [DI], AL
20 INC  DI
21 INT  10h          ; yankı
22 JMP  oku
23 bitti: MOV BYTE PTR [DI], 0
24 RET
25 ad   DB 6 DUP(?)
```



DI tamponun başında değil (ad+2 ≠ ad): silinecek harf var. `DEC DI` ile b'nin baytı yeniden boş sayılır; bayt bellekte durur ama dizginin dışında kalır.

Adım 6 / 9

DEĞİŞKENLER

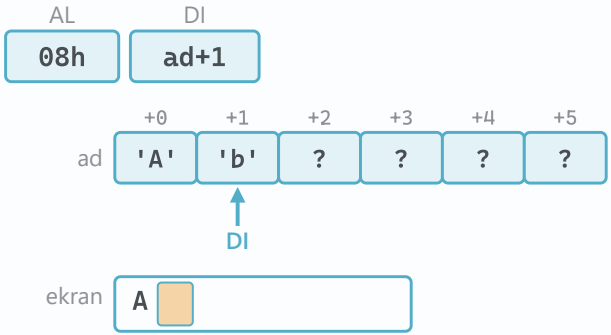
`AL = 08h` `DI = ad+1` `imleç = 2`

ÇIKTI

```
Ab
```

Kod İzleme: Geri Silmeyi İşleyen Okuma

```
 1 ORG  100h
 2 LEA  DI, ad
 3 oku: MOV AH, 0
 4 INT  16h          ; AL = tuş (yankısız)
 5 CMP  AL, 13       ; Enter mi?
 6 JE   bitti
 7 MOV  AH, 0Eh
 8 CMP  AL, 8        ; geri silme mi?
 9 JNE  ekle
10 CMP  DI, OFFSET ad
11 JE   oku          ; silinecek harf yok
12 DEC  DI
▶ INT  10h          ; 8: imleç sola
▶ MOV  AL, ' '
▶ INT  10h          ; harfin yerine boşluk
▶ MOV  AL, 8
▶ INT  10h          ; yine sola
18 JMP  oku
19 ekle: MOV [DI], AL
20 INC  DI
21 INC  10h         ; yankı
22 JMP  oku
23 bitti: MOV BYTE PTR [DI], 0
24 RET
25 ad   DB 6 DUP(?)
```



Ekranda üçlü: 8 imleci sola alır, boşluk b'nin üstüne yazılır, ikinci 8 imleci yine sola getirir. Görünen yalnızca A.

Adım 7 / 9

DEĞİŞKENLER

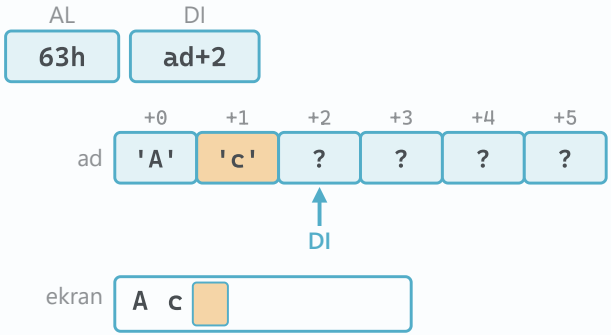
AL = **08h** | DI = **ad+1** | imleç = **1**

ÇIKTI

```
A
```

Kod İzleme: Geri Silmeyi İşleyen Okuma

```
 1 ORG  100h
 2 LEA  DI, ad
 ▶ oku: MOV AH, 0
 ▶ INT  16h          ; AL = tuş (yankısız)
 ▶ CMP  AL, 13       ; Enter mi?
 ▶ JE   bitti
 ▶ MOV  AH, 0Eh
 ▶ CMP  AL, 8        ; geri silme mi?
 ▶ JNE  ekle
10 CMP  DI, OFFSET ad
11 JE   oku          ; silinecek harf yok
12 DEC  DI
13 INT  10h          ; 8: imleç sola
14 MOV  AL, ' '
15 INT  10h          ; harfin yerine boşluk
16 MOV  AL, 8
17 INT  10h          ; yine sola
 ▶ JMP  oku
 ▶ ekle: MOV [DI], AL
 ▶ INC  DI
 ▶ INT  10h          ; yankı
22 JMP  oku
23 bitti: MOV BYTE PTR [DI], 0
24 RET
25 ad   DB 6 DUP(?)
```



'c', b'nin bulunduğu bayta yazılır: tampon A, c; ekranda Ac.

Adım 8 / 9

DEĞİŞKENLER

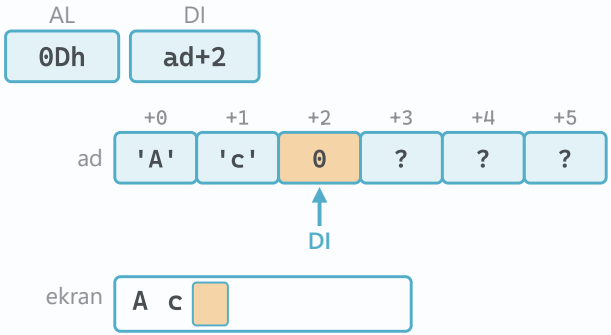
AL = 63h DI = ad+2 imleç = 2

ÇIKTI

```
Ac
```

Kod İzleme: Geri Silmeyi İşleyen Okuma

```
 1 ORG  100h
 2 LEA  DI, ad
 ▶ oku: MOV AH, 0
 ▶ INT  16h          ; AL = tuş (yankısız)
 ▶ CMP  AL, 13       ; Enter mi?
 ▶ JE   bitti
 7 MOV  AH, 0Eh
 8 CMP  AL, 8        ; geri silme mi?
 9 JNE  ekle
10 CMP  DI, OFFSET ad
11 JE   oku          ; silinecek harf yok
12 DEC  DI
13 INT  10h          ; 8: imleç sola
14 MOV  AL, ' '
15 INT  10h          ; harfin yerine boşluk
16 MOV  AL, 8
17 INT  10h          ; yine sola
18 JMP  oku
19 ekle: MOV [DI], AL
20 INC  DI
21 INT  10h          ; yankı
 ▶ JMP  oku
 ▶ bitti: MOV BYTE PTR [DI], 0
24 RET
25 ad   DB 6 DUP(?)
```



Enter: `JE bitti` ; DI'nin gösterdiği bayta 0 yazılır. Tampon 'A', 'c', 0: geri silinen harf dizgide yoktur.

Adım 9 / 9

DEĞİŞKENLER

AL = **0Dh** | DI = **ad+2** | imleç = **2**

ÇIKTI

```
Ac
```

PROSEDÜRLER

Sayı Prosedürleri

Prosedür	Ne yapar?	Girdi / çıktı	Tanım
`SCAN_NUM`	Klavyeden çok basamaklı, **işaretli** bir sayı okur (Enter ile biter).	çıktı: **CX**	`DEFINE_SCAN_NUM`
`PRINT_NUM`	AX'teki **işaretli** sayıyı onluk olarak yazar.	girdi: **AX**	`DEFINE_PRINT_NUM` + `DEFINE_PRINT_NUM_UNS`
`PRINT_NUM_UNS`	AX'teki **işaretsiz** sayıyı yazar.	girdi: **AX**	`DEFINE_PRINT_NUM_UNS`

İşaretli aralık

16 bit: −32 768 … 32 767

İşaretsiz aralık

16 bit: 0 … 65 535

Aynı bitler

`FFE7h` = −25 (işaretli) = 65 511 (işaretsiz)

Kod İzleme: SCAN_NUM Mantığı (girdi 307)

```
1  ORG  100h
▶ XOR  CX, CX       ; sonuç = 0
3  oku: MOV AH, 01h
4  INT  21h         ; AL = tuş (yankılı)
5  CMP  AL, 13      ; Enter mi?
6  JE   bitti
7  SUB  AL, '0'     ; ASCII → rakam
8  CBW              ; AX = rakam
9  XCHG AX, CX      ; AX=sonuç, CX=rakam
10 MOV  BX, 10
11 MUL  BX          ; AX = sonuç × 10
12 ADD  CX, AX      ; CX = sonuç×10 + rakam
13 JMP  oku
14 bitti: RET
```

AX	CX	BX	ZF
0000h	0000h	0000h	1

CX onluk: 0

ekran

SCAN_NUM'un özü: rakamlar soldan sağa okunur, sonuç = sonuç × 10 + rakam (kütüphanedeki sürüm eksi işaretini ve taşmayı da denetler). CX = 0.

Adım 1 / 11

DEĞİŞKENLER

AL = 00h CX = 0000h BX = 0000h

ÇIKTI

Kod İzleme: SCAN_NUM Mantığı (girdi 307)

```
1  ORG  100h
2  XOR  CX, CX       ; sonuç = 0
▶  oku: MOV AH, 01h
▶  INT  21h          ; AL = tuş (yankılı)
5  CMP  AL, 13       ; Enter mi?
6  JE   bitti
7  SUB  AL, '0'      ; ASCII → rakam
8  CBW               ; AX = rakam
9  XCHG AX, CX       ; AX=sonuç, CX=rakam
10 MOV  BX, 10
11 MUL  BX           ; AX = sonuç × 10
12 ADD  CX, AX       ; CX = sonuç×10 + rakam
13 JMP  oku
14 bitti: RET
```

AX	CX	BX	ZF
0133h	0000h	0000h	1

CX onluk: 0

ekran: 3

`INT 21h`, AH = 01h: '3' tuşu, AL = 33h; karakter ekrana yankılanır.

Adım 2 / 11

DEĞİŞKENLER

AL = 33h | CX = 0000h | BX = 0000h

ÇIKTI

```
3
```

Kod İzleme: SCAN_NUM Mantığı (girdi 307)

```
1  ORG  100h
2  XOR  CX, CX       ; sonuç = 0
3  oku: MOV AH, 01h
4  INT  21h          ; AL = tuş (yankılı)
▶  CMP  AL, 13       ; Enter mi?
▶  JE   bitti
▶  SUB  AL, '0'      ; ASCII → rakam
8  CBW               ; AX = rakam
9  XCHG AX, CX       ; AX=sonuç, CX=rakam
10 MOV  BX, 10
11 MUL  BX           ; AX = sonuç × 10
12 ADD  CX, AX       ; CX = sonuç×10 + rakam
13 JMP  oku
14 bitti: RET
```

AX	CX	BX	ZF
0103h	0000h	0000h	0

CX onluk: 0

ekran: 3

Enter değil. `SUB AL, '0'` : 33h − 30h = 3.

Adım 3 / 11

DEĞİŞKENLER

AL = 03h | CX = 0000h | BX = 0000h

ÇIKTI

```
3
```

Kod İzleme: SCAN_NUM Mantığı (girdi 307)

```
1  ORG  100h
2  XOR  CX, CX       ; sonuç = 0
3  oku: MOV AH, 01h
4  INT  21h          ; AL = tuş (yankılı)
5  CMP  AL, 13       ; Enter mi?
6  JE   bitti
7  SUB  AL, '0'      ; ASCII → rakam
▶  CBW               ; AX = rakam
▶  XCHG AX, CX       ; AX=sonuç, CX=rakam
10 MOV  BX, 10
11 MUL  BX           ; AX = sonuç × 10
12 ADD  CX, AX       ; CX = sonuç×10 + rakam
13 JMP  oku
14 bitti: RET
```

AX	CX	BX	ZF
0000h	0003h	0000h	0

CX onluk: 3

ekran: 3

`CBW` AX = 0003h yapar; `XCHG` ile AX = eski sonuç (0), CX = rakam (3).

Adım 4 / 11

DEĞİŞKENLER

AL = 00h | CX = 0003h | BX = 0000h

ÇIKTI

```
3
```

Kod İzleme: SCAN_NUM Mantığı (girdi 307)

```
 1 ORG  100h
 2 XOR  CX, CX       ; sonuç = 0
 3 oku: MOV AH, 01h
 4 INT  21h          ; AL = tuş (yankılı)
 5 CMP  AL, 13       ; Enter mi?
 6 JE   bitti
 7 SUB  AL, '0'      ; ASCII → rakam
 8 CBW               ; AX = rakam
 9 XCHG AX, CX       ; AX=sonuç, CX=rakam
▶ MOV  BX, 10
▶ MUL  BX           ; AX = sonuç × 10
▶ ADD  CX, AX       ; CX = sonuç×10 + rakam
13 JMP  oku
14 bitti: RET
```

AX	CX	BX	ZF
0000h	0003h	000Ah	0

CX onluk: 3

ekran: 3

`MUL BX`: 0 × 10 = 0; `ADD CX, AX` → CX = 3.

Adım 5 / 11

DEĞİŞKENLER

AL = 00h | CX = 0003h | BX = 000Ah

ÇIKTI

```
3
```

Kod İzleme: SCAN_NUM Mantığı (girdi 307)

```
 1 ORG  100h
 2 XOR  CX, CX       ; sonuç = 0
▶ oku: MOV AH, 01h
▶ INT  21h          ; AL = tuş (yankılı)
▶ CMP  AL, 13       ; Enter mi?
▶ JE   bitti
▶ SUB  AL, '0'      ; ASCII → rakam
▶ CBW               ; AX = rakam
▶ XCHG AX, CX       ; AX=sonuç, CX=rakam
10 MOV  BX, 10
11 MUL  BX          ; AX = sonuç × 10
12 ADD  CX, AX      ; CX = sonuç×10 + rakam
▶ JMP  oku
14 bitti: RET
```

AX	CX	BX	ZF
0003h	0000h	000Ah	1

CX onluk: 0

ekran: 30

'0' okunur: rakam 0; XCHG sonrası AX = 3, CX = 0.

Adım 6 / 11

DEĞİŞKENLER

AL = 03h | CX = 0000h | BX = 000Ah

ÇIKTI

```
30
```

Kod İzleme: SCAN_NUM Mantığı (girdi 307)

```
1  ORG  100h
2  XOR  CX, CX       ; sonuç = 0
3  oku: MOV AH, 01h
4  INT  21h          ; AL = tuş (yankılı)
5  CMP  AL, 13       ; Enter mi?
6  JE   bitti
7  SUB  AL, '0'      ; ASCII → rakam
8  CBW               ; AX = rakam
9  XCHG AX, CX       ; AX=sonuç, CX=rakam
▶  MOV  BX, 10
▶  MUL  BX           ; AX = sonuç × 10
▶  ADD  CX, AX       ; CX = sonuç×10 + rakam
13 JMP  oku
14 bitti: RET
```

AX	CX	BX	ZF
001Eh	001Eh	000Ah	0

CX onluk: **30**

ekran: **30**

3 × 10 = 30 (1Eh) → CX = 30 + 0 = 30.

Adım 7 / 11

DEĞİŞKENLER

AL = 1Eh | CX = 001Eh | BX = 000Ah

ÇIKTI

```
30
```

Kod İzleme: SCAN_NUM Mantığı (girdi 307)

```
1  ORG  100h
2  XOR  CX, CX       ; sonuç = 0
   oku: MOV AH, 01h
   INT  21h          ; AL = tuş (yankılı)
   CMP  AL, 13       ; Enter mi?
   JE   bitti
   SUB  AL, '0'      ; ASCII → rakam
   CBW               ; AX = rakam
9  XCHG AX, CX       ; AX=sonuç, CX=rakam
10 MOV  BX, 10
11 MUL  BX           ; AX = sonuç × 10
12 ADD  CX, AX       ; CX = sonuç×10 + rakam
   JMP  oku
14 bitti: RET
```

AX	CX	BX	ZF
0007h	001Eh	000Ah	0

CX onluk: 30

ekran: 307

'7' okunur: rakam 7 AX'e (CBW), sonuç 30 hâlâ CX'te.

Adım 8 / 11

DEĞİŞKENLER

AL = 07h | CX = 001Eh | BX = 000Ah

ÇIKTI

```
307
```

Kod İzleme: SCAN_NUM Mantığı (girdi 307)

```
 1 ORG  100h
 2 XOR  CX, CX       ; sonuç = 0
 3 oku: MOV AH, 01h
 4 INT  21h          ; AL = tuş (yankılı)
 5 CMP  AL, 13       ; Enter mi?
 6 JE   bitti
 7 SUB  AL, '0'      ; ASCII → rakam
 8 CBW               ; AX = rakam
▶ XCHG AX, CX        ; AX=sonuç, CX=rakam
▶ MOV  BX, 10
▶ MUL  BX            ; AX = sonuç × 10
▶ ADD  CX, AX        ; CX = sonuç×10 + rakam
13 JMP  oku
14 bitti: RET
```

AX	CX	BX	ZF
012Ch	0133h	000Ah	0

CX onluk: **307**

ekran: 307

XCHG: AX = 30, CX = 7. 30 × 10 = 300 (012Ch) → CX = 300 + 7 = 307 = 0133h.

Adım 9 / 11

DEĞİŞKENLER

AL = 2Ch | CX = 0133h | BX = 000Ah

ÇIKTI

```
307
```

Kod İzleme: SCAN_NUM Mantığı (girdi 307)

```
 1 ORG  100h
 2 XOR  CX, CX       ; sonuç = 0
▶ oku: MOV AH, 01h
▶ INT  21h          ; AL = tuş (yankılı)
▶ CMP  AL, 13       ; Enter mi?
▶ JE   bitti
 7 SUB  AL, '0'      ; ASCII → rakam
 8 CBW               ; AX = rakam
 9 XCHG AX, CX       ; AX=sonuç, CX=rakam
10 MOV  BX, 10
11 MUL  BX           ; AX = sonuç × 10
12 ADD  CX, AX       ; CX = sonuç×10 + rakam
▶ JMP  oku
14 bitti: RET
```

AX	CX	BX	ZF
010Dh	0133h	000Ah	1

CX onluk: **307**

ekran: 307

Enter: AL = 0Dh = 13 → ZF = 1, `JE bitti` atlar.

Adım 10 / 11

DEĞİŞKENLER

AL = 0Dh | CX = 0133h | BX = 000Ah

ÇIKTI

```
307
```

Kod İzleme: SCAN_NUM Mantığı (girdi 307)

```
1  ORG  100h
2  XOR  CX, CX       ; sonuç = 0
3  oku: MOV AH, 01h
4  INT  21h          ; AL = tuş (yankılı)
5  CMP  AL, 13       ; Enter mi?
6  JE   bitti
7  SUB  AL, '0'      ; ASCII → rakam
8  CBW               ; AX = rakam
9  XCHG AX, CX       ; AX=sonuç, CX=rakam
10 MOV  BX, 10
11 MUL  BX           ; AX = sonuç × 10
12 ADD  CX, AX       ; CX = sonuç×10 + rakam
13 JMP  oku
▶  bitti: RET
```

AX	CX	BX	ZF
010Dh	0133h	000Ah	1

CX onluk: **307**

ekran: 307

Sonuç SCAN_NUM'daki gibi CX'te: 0133h = 307.

Adım 11 / 11

DEĞİŞKENLER

AL = 0Dh | CX = 0133h | BX = 000Ah

ÇIKTI

```
307
```

PROSEDÜRLER

SCAN_NUM'ın Denetimleri

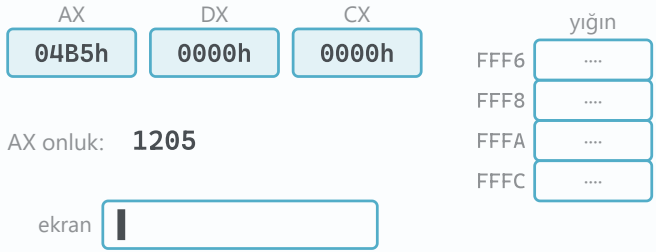
- `-` tuşu bir işaret bayrağı kurar; Enter'da bayrak kuruluysa `NEG CX` uygulanır.
- Rakam olmayan tuş ekrandan silinir (8, ' ', 8) ve sayıya katılmaz.
- Geri silme son basamağı atar: CX ← CX / 10.
- Sayı 16 bite sığmayacaksa (× 10 ya da toplama taşarsa) son basamak kabul edilmez.
- Enter de yankılanır: imleç satır başına döner (CR) ama alt satıra inmez. Örnek programdaki PTHIS dizgisi bu yüzden 13, 10 ile başlar.

Girdi	CX	PRINT_NUM
307	`0133h`	307
-25	`FFE7h`	-25
12a3	`007Bh`	123
40000	`9C40h`	-25536
70000	`1B58h`	7000
-40000	`63C0h`	25536

Basamaklar işaretsiz 16 bit olarak biriktirilir, işaret en sonda uygulanır: −32 768 … 32 767 dışındaki girdiler hata vermeden yanlış işaretli sonuç üretir. (Tablo, kütüphanedeki yordamın eşdeğer kodu çalıştırılarak elde edildi.)

Kod İzleme: PRINT_NUM_UNS Mantığı (AX = 1205)

```
1  ORG  100h
▶  MOV  AX, 1205     ; yazdırılacak sayı
▶  MOV  BX, 10
▶  XOR  CX, CX       ; basamak sayacı
▶  bol: XOR DX, DX
6  DIV  BX           ; AX=bölüm, DX=kalan
7  PUSH DX           ; basamak yığına
8  INC  CX
9  OR   AX, AX       ; bölüm 0 mı?
10 JNZ  bol
11 yaz: POP AX       ; önce en anlamlı
12 ADD  AL, '0'
13 MOV  AH, 0Eh
14 INT  10h
15 LOOP yaz
16 RET
```



Aynı çıktıyı veren eşdeğer kod (kütüphanedeki sürüm 10000, 1000, … ile böler). AX = 1205 = 04B5h, bölen BX = 10.

Adım 1 / 10

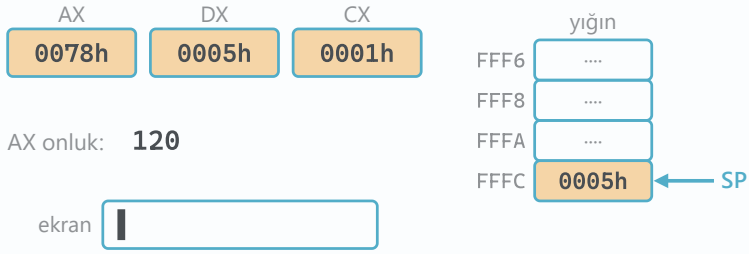
DEĞİŞKENLER

AX = **04B5h** DX = **0000h** CX = **0000h** SP = **FFFEh**

ÇIKTI

Kod İzleme: PRINT_NUM_UNS Mantığı (AX = 1205)

```
1  ORG  100h
2  MOV  AX, 1205     ; yazdırılacak sayı
3  MOV  BX, 10
4  XOR  CX, CX       ; basamak sayacı
5  bol: XOR DX, DX
▶  DIV  BX           ; AX=bölüm, DX=kalan
▶  PUSH DX           ; basamak yığına
▶  INC  CX
9  OR   AX, AX       ; bölüm 0 mı?
10 JNZ  bol
11 yaz: POP AX       ; önce en anlamlı
12 ADD  AL, '0'
13 MOV  AH, 0Eh
14 INT  10h
15 LOOP yaz
16 RET
```



`DIV BX`: DX:AX / 10 → AX = 120, kalan DX = 5. Son basamak yığına, CX = 1.

Adım 2 / 10

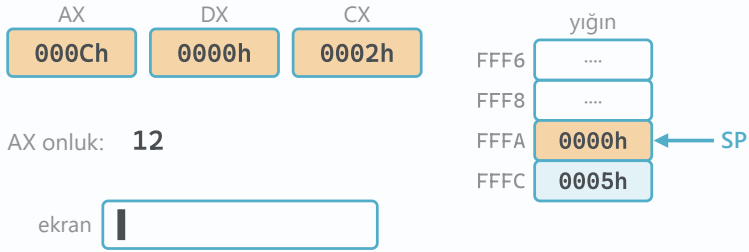
DEĞİŞKENLER

AX = 0078h DX = 0005h CX = 0001h SP = FFFCh

ÇIKTI

Kod İzleme: PRINT_NUM_UNS Mantığı (AX = 1205)

```
 1 ORG  100h
 2 MOV  AX, 1205     ; yazdırılacak sayı
 3 MOV  BX, 10
 4 XOR  CX, CX       ; basamak sayacı
▶bol: XOR DX, DX
▶DIV  BX            ; AX=bölüm, DX=kalan
▶PUSH DX            ; basamak yığına
▶INC  CX
▶OR   AX, AX        ; bölüm 0 mı?
▶JNZ  bol
11 yaz: POP AX      ; önce en anlamlı
12 ADD  AL, '0'
13 MOV  AH, 0Eh
14 INT  10h
15 LOOP yaz
16 RET
```



AX ≠ 0 → döngü sürer: 120 / 10 = 12, kalan 0 yığına; CX = 2.

Adım 3 / 10

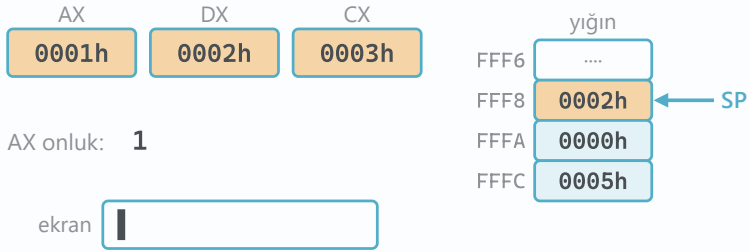
DEĞİŞKENLER

AX = 000Ch DX = 0000h CX = 0002h SP = FFFAh

ÇIKTI

Kod İzleme: PRINT_NUM_UNS Mantığı (AX = 1205)

```
 1 ORG  100h
 2 MOV  AX, 1205     ; yazdırılacak sayı
 3 MOV  BX, 10
 4 XOR  CX, CX       ; basamak sayacı
▶bol: XOR DX, DX
▶DIV  BX            ; AX=bölüm, DX=kalan
▶PUSH DX            ; basamak yığına
▶INC  CX
▶OR   AX, AX        ; bölüm 0 mı?
▶JNZ  bol
11 yaz: POP AX      ; önce en anlamlı
12 ADD  AL, '0'
13 MOV  AH, 0Eh
14 INT  10h
15 LOOP yaz
16 RET
```



12 / 10 = 1, kalan 2 yığına; CX = 3.

Adım 4 / 10

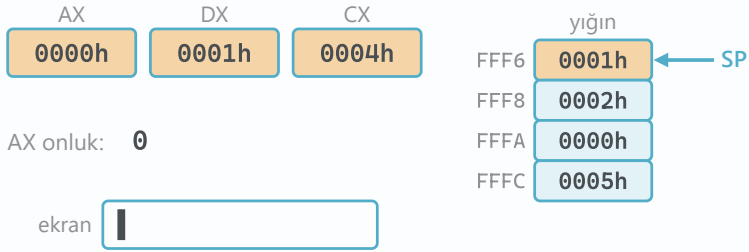
DEĞİŞKENLER

AX = 0001h DX = 0002h CX = 0003h SP = FFF8h

ÇIKTI

Kod İzleme: PRINT_NUM_UNS Mantığı (AX = 1205)

```
 1 ORG  100h
 2 MOV  AX, 1205     ; yazdırılacak sayı
 3 MOV  BX, 10
 4 XOR  CX, CX       ; basamak sayacı
▶bol: XOR DX, DX
▶DIV  BX            ; AX=bölüm, DX=kalan
▶PUSH DX            ; basamak yığına
▶INC  CX
▶OR   AX, AX        ; bölüm 0 mı?
▶JNZ  bol
11 yaz: POP AX      ; önce en anlamlı
12 ADD  AL, '0'
13 MOV  AH, 0Eh
14 INT  10h
15 LOOP yaz
16 RET
```



1 / 10 = 0, kalan 1 yığına; CX = 4 basamak.

Adım 5 / 10

DEĞİŞKENLER

AX = 0000h DX = 0001h CX = 0004h SP = FFF6h

ÇIKTI

Kod İzleme: PRINT_NUM_UNS Mantığı (AX = 1205)

```
1  ORG  100h
2  MOV  AX, 1205     ; yazdırılacak sayı
3  MOV  BX, 10
4  XOR  CX, CX       ; basamak sayacı
5  bol: XOR DX, DX
6  DIV  BX           ; AX=bölüm, DX=kalan
7  PUSH DX           ; basamak yığına
8  INC  CX
▶  OR   AX, AX       ; bölüm 0 mı?
▶  JNZ  bol
11 yaz: POP AX       ; önce en anlamlı
12 ADD  AL, '0'
13 MOV  AH, 0Eh
14 INT  10h
15 LOOP yaz
16 RET
```



`OR AX, AX` : AX = 0 → ZF = 1, `JNZ` atlamaz. Yığının üstünde en anlamlı basamak (1) var.

Adım 6 / 10

DEĞİŞKENLER

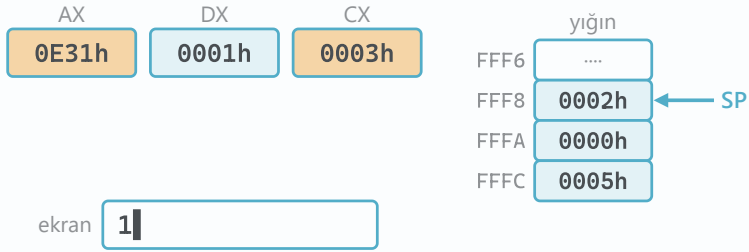
AX = **0000h** DX = **0001h** CX = **0004h**

SP = **FFF6h**

ÇIKTI

Kod İzleme: PRINT_NUM_UNS Mantığı (AX = 1205)

```
1  ORG  100h
2  MOV  AX, 1205     ; yazdırılacak sayı
3  MOV  BX, 10
4  XOR  CX, CX       ; basamak sayacı
5  bol: XOR DX, DX
6  DIV  BX           ; AX=bölüm, DX=kalan
7  PUSH DX           ; basamak yığına
8  INC  CX
9  OR   AX, AX       ; bölüm 0 mı?
10 JNZ  bol
▶  yaz: POP AX       ; önce en anlamlı
▶  ADD  AL, '0'
▶  MOV  AH, 0Eh
▶  INT  10h
▶  LOOP yaz
16 RET
```



POP AX = 1; + '0' = 31h; `INT 10h` '1' yazar. LOOP: CX = 3.

Adım 7 / 10

DEĞİŞKENLER

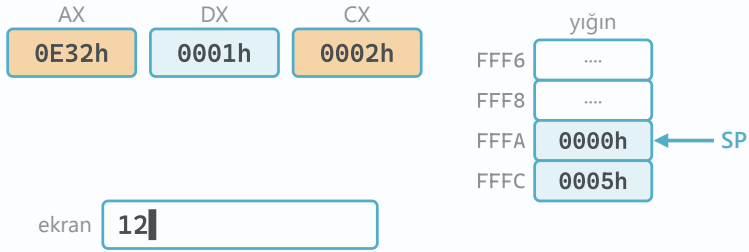
AX = 0E31h DX = 0001h CX = 0003h SP = FFF8h

ÇIKTI

```
1
```

Kod İzleme: PRINT_NUM_UNS Mantığı (AX = 1205)

```
1  ORG  100h
2  MOV  AX, 1205     ; yazdırılacak sayı
3  MOV  BX, 10
4  XOR  CX, CX       ; basamak sayacı
5  bol: XOR DX, DX
6  DIV  BX           ; AX=bölüm, DX=kalan
7  PUSH DX           ; basamak yığına
8  INC  CX
9  OR   AX, AX       ; bölüm 0 mı?
10 JNZ  bol
   yaz: POP AX       ; önce en anlamlı
   ADD  AL, '0'
   MOV  AH, 0Eh
   INT  10h
   LOOP yaz
16 RET
```



Sonraki basamak 2 → '2'. CX = 2.

Adım 8 / 10

DEĞİŞKENLER

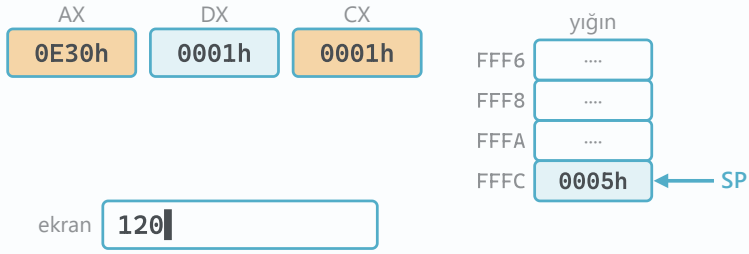
AX = 0E32h DX = 0001h CX = 0002h SP = FFFAh

ÇIKTI

```
12
```

Kod İzleme: PRINT_NUM_UNS Mantığı (AX = 1205)

```
1  ORG  100h
2  MOV  AX, 1205     ; yazdırılacak sayı
3  MOV  BX, 10
4  XOR  CX, CX       ; basamak sayacı
5  bol: XOR DX, DX
6  DIV  BX           ; AX=bölüm, DX=kalan
7  PUSH DX           ; basamak yığına
8  INC  CX
9  OR   AX, AX       ; bölüm 0 mı?
10 JNZ  bol
▶  yaz: POP AX       ; önce en anlamlı
▶  ADD  AL, '0'
▶  MOV  AH, 0Eh
▶  INT  10h
▶  LOOP yaz
16 RET
```



0 → '0'. CX = 1.

Adım 9 / 10

DEĞİŞKENLER

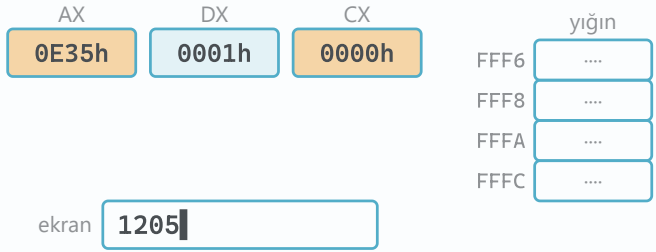
AX = 0E30h DX = 0001h CX = 0001h SP = FFFCh

ÇIKTI

```
120
```

Kod İzleme: PRINT_NUM_UNS Mantığı (AX = 1205)

```
 1 ORG  100h
 2 MOV  AX, 1205     ; yazdırılacak sayı
 3 MOV  BX, 10
 4 XOR  CX, CX       ; basamak sayacı
 5 bol: XOR DX, DX
 6 DIV  BX           ; AX=bölüm, DX=kalan
 7 PUSH DX           ; basamak yığına
 8 INC  CX
 9 OR   AX, AX       ; bölüm 0 mı?
10 JNZ  bol
▶ yaz: POP AX        ; önce en anlamlı
▶ ADD  AL, '0'
▶ MOV  AH, 0Eh
▶ INT  10h
▶ LOOP yaz
16 RET
```



5 → '5'. CX = 0, döngü biter; yığın FFFEh'ye döndü. Ekranda 1205.

Adım 10 / 10

DEĞİŞKENLER

AX = 0E35h DX = 0001h CX = 0000h SP = FFFEh

ÇIKTI

```
1205
```

PROSEDÜRLER

Kütüphanenin Yöntemi: Azalan Bölen

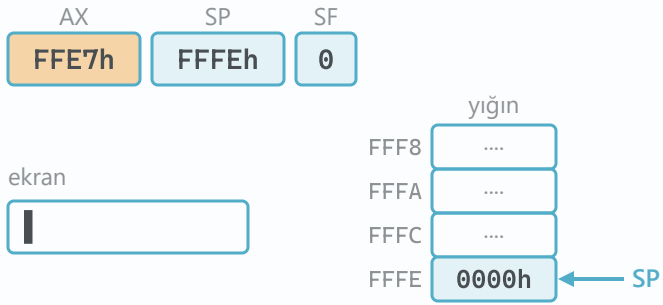
- Bölen 10 000'den başlar, her turda 10'a bölünür; bölen 0 olunca yazdırma biter.
- Bölüm (0–9) bir basamaktır: `ADD AL, 30h` ile yazılır, kalan bir sonraki turun sayısı olur.
- Bir bayrak (CX = 1) baştaki sıfırları atlar; ilk basamak yazılınca bayrak 0 olur ve aradaki sıfırlar artık yazılır.
- AX = 0 için ayrı bir dal doğrudan '0' yazar.
- 65 535 ÷ 10 000 = 6 olduğundan bölüm her zaman AL'ye sığar.

Bölen	Sayı	İşlem	Ekran
10000	3007	sayı < bölen, henüz basamak yok: atla	—
1000	3007	3007 ÷ 1000 = 3, kalan 7	3
100	7	7 ÷ 100 = 0, kalan 7	30
10	7	7 ÷ 10 = 0, kalan 7	300
1	7	7 ÷ 1 = 7, kalan 0	3007

Basamaklar soldan sağa, hesaplandıkları sırada yazılır; yığına itmeye ve ters çevirmeye gerek kalmaz.

Kod İzleme: PRINT_NUM ve Eksi İşareti

```
1  ORG  100h
▶  MOV  AX, -25        ; AX = FFE7h
3  CALL print_num
4  RET
5  print_num PROC
6  PUSH AX
7  OR   AX, AX         ; SF = işaret biti
8  JNS  pozitif
9  NEG  AX             ; AX = 0019h
10 PUSH AX             ; PUTC '-' açılımı
11 MOV  AL, '-'
12 MOV  AH, 0Eh
13 INT  10h
14 POP  AX
15 pozitif: CALL print_num_uns
16 POP  AX             ; AX = FFE7h geri
17 RET
18 print_num ENDP
```



AX = −25; ikiye tümleyen gösterimiyle FFE7h. Gösterilen kod, kütüphanedeki PRINT_NUM'ın eşdeğeridir.

Adım 1 / 8

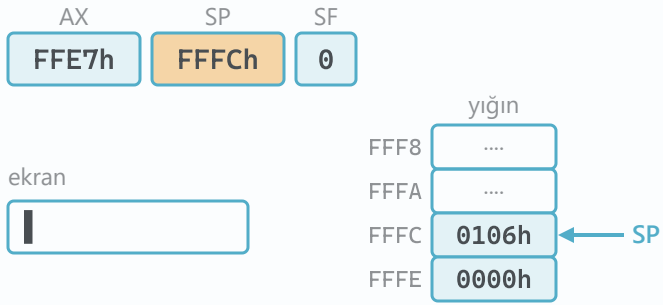
DEĞİŞKENLER

AX = **FFE7h** SP = **FFFEh** SF = **0**

ÇIKTI

Kod İzleme: PRINT_NUM ve Eksi İşareti

```
1  ORG  100h
2  MOV  AX, -25        ; AX = FFE7h
▶  CALL print_num
4  RET
5  print_num PROC
6  PUSH AX
7  OR   AX, AX         ; SF = işaret biti
8  JNS  pozitif
9  NEG  AX             ; AX = 0019h
10 PUSH AX             ; PUTC '-' açılımı
11 MOV  AL, '-'
12 MOV  AH, 0Eh
13 INT  10h
14 POP  AX
15 pozitif: CALL print_num_uns
16 POP  AX             ; AX = FFE7h geri
17 RET
18 print_num ENDP
```



`CALL print_num` : dönüş adresi 0106h yığına, SP = FFFCh.

Adım 2 / 8

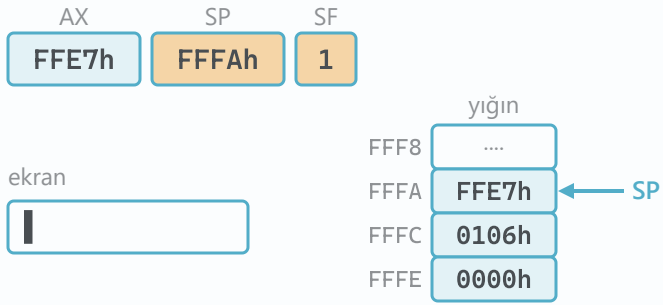
DEĞİŞKENLER

AX = **FFE7h** SP = **FFFCh** SF = **0**

ÇIKTI

Kod İzleme: PRINT_NUM ve Eksi İşareti

```
1  ORG  100h
2  MOV  AX, -25        ; AX = FFE7h
3  CALL print_num
4  RET
5  print_num PROC
6  PUSH AX
7  OR   AX, AX         ; SF = işaret biti
8  JNS  pozitif
9  NEG  AX             ; AX = 0019h
10 PUSH AX             ; PUTC '-' açılımı
11 MOV  AL, '-'
12 MOV  AH, 0Eh
13 INT  10h
14 POP  AX
15 pozitif: CALL print_num_uns
16 POP  AX             ; AX = FFE7h geri
17 RET
18 print_num ENDP
```



`PUSH AX` çağıranın değerini saklar. `OR AX, AX` değeri değiştirmez ama bayrakları kurar: en anlamlı bit 1 → SF = 1.

Adım 3 / 8

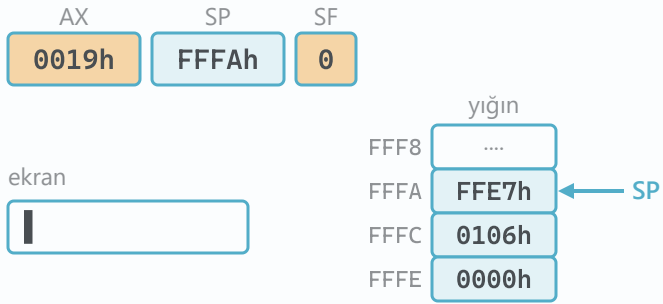
DEĞİŞKENLER

AX = **FFE7h** SP = **FFFAh** SF = **1**

ÇIKTI

Kod İzleme: PRINT_NUM ve Eksi İşareti

```
1  ORG   100h
2  MOV   AX, -25        ; AX = FFE7h
3  CALL  print_num
4  RET
5  print_num PROC
6  PUSH  AX
7  OR    AX, AX         ; SF = işaret biti
▶  JNS   pozitif
▶  NEG   AX             ; AX = 0019h
10 PUSH  AX             ; PUTC '-' açılımı
11 MOV   AL, '-'
12 MOV   AH, 0Eh
13 INT   10h
14 POP   AX
15 pozitif: CALL print_num_uns
16 POP   AX             ; AX = FFE7h geri
17 RET
18 print_num ENDP
```



`JNS` atlamaz (sayı negatif). `NEG AX` : 0 − FFE7h = 0019h = 25, sayının mutlak değeri.

Adım 4 / 8

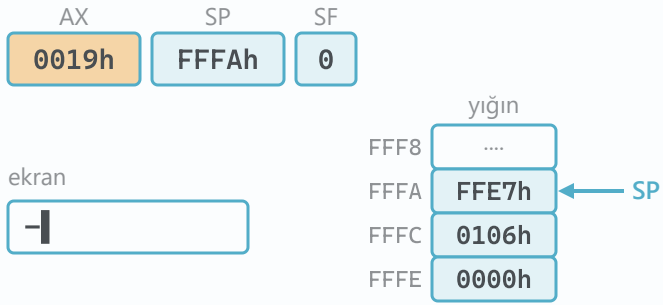
DEĞİŞKENLER

AX = 0019h | SP = FFFAh | SF = 0

ÇIKTI

Kod İzleme: PRINT_NUM ve Eksi İşareti

```
1  ORG  100h
2  MOV  AX, -25        ; AX = FFE7h
3  CALL print_num
4  RET
5  print_num PROC
6  PUSH AX
7  OR   AX, AX         ; SF = işaret biti
8  JNS  pozitif
9  NEG  AX             ; AX = 0019h
   PUSH AX             ; PUTC '-' açılımı
   MOV  AL, '-'
   MOV  AH, 0Eh
   INT  10h
   POP  AX
15 pozitif: CALL print_num_uns
16 POP  AX             ; AX = FFE7h geri
17 RET
18 print_num ENDP
```



`PUTC '-'` açılımı: AX yığına, '-' yazılır, AX = 0019h geri gelir.

Adım 5 / 8

DEĞİŞKENLER

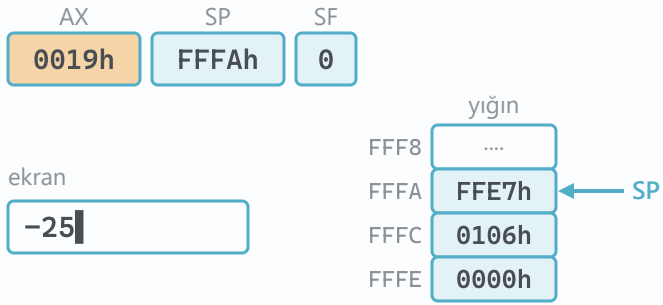
AX = **0019h** | SP = **FFFAh** | SF = **0**

ÇIKTI

```
-
```

Kod İzleme: PRINT_NUM ve Eksi İşareti

```
1  ORG  100h
2  MOV  AX, -25        ; AX = FFE7h
3  CALL print_num
4  RET
5  print_num PROC
6  PUSH AX
7  OR   AX, AX         ; SF = işaret biti
8  JNS  pozitif
9  NEG  AX             ; AX = 0019h
10 PUSH AX             ; PUTC '-' açılımı
11 MOV  AL, '-'
12 MOV  AH, 0Eh
13 INT  10h
14 POP  AX
▶  pozitif: CALL print_num_uns
16 POP  AX             ; AX = FFE7h geri
17 RET
18 print_num ENDP
```



`CALL print_num_uns` : 25'in basamakları yazılır (basamak döngüsü PRINT_NUM_UNS izlemesindekiyle aynıdır; prosedür kullandığı yazmaçları saklar).

Adım 6 / 8

DEĞİŞKENLER

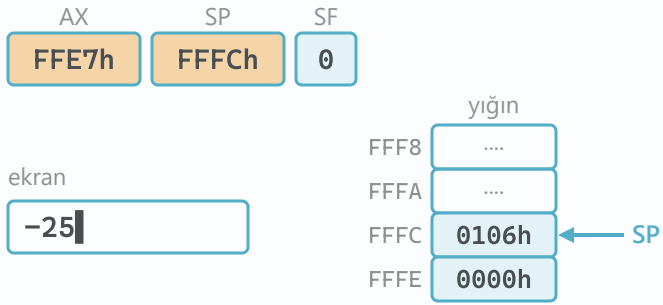
AX = **0019h** | SP = **FFFAh** | SF = **0**

ÇIKTI

```
-25
```

Kod İzleme: PRINT_NUM ve Eksi İşareti

```
1  ORG  100h
2  MOV  AX, -25        ; AX = FFE7h
3  CALL print_num
4  RET
5  print_num PROC
6  PUSH AX
7  OR   AX, AX         ; SF = işaret biti
8  JNS  pozitif
9  NEG  AX             ; AX = 0019h
10 PUSH AX             ; PUTC '-' açılımı
11 MOV  AL, '-'
12 MOV  AH, 0Eh
13 INT  10h
14 POP  AX
15 pozitif: CALL print_num_uns
▶  POP  AX             ; AX = FFE7h geri
17 RET
18 print_num ENDP
```



`POP AX` : AX yeniden FFE7h. Sayı yazdırıldıktan sonra da AX'te durur.

Adım 7 / 8

DEĞİŞKENLER

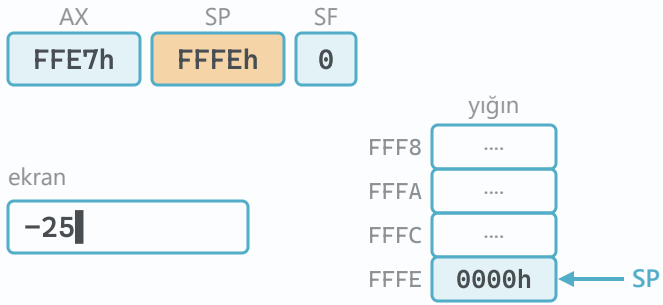
AX = FFE7h SP = FFFCh SF = 0

ÇIKTI

```
-25
```

Kod İzleme: PRINT_NUM ve Eksi İşareti

```
1  ORG  100h
2  MOV  AX, -25        ; AX = FFE7h
3  CALL print_num
4  RET
5  print_num PROC
6  PUSH AX
7  OR   AX, AX         ; SF = işaret biti
8  JNS  pozitif
9  NEG  AX             ; AX = 0019h
10 PUSH AX             ; PUTC '-' açılımı
11 MOV  AL, '-'
12 MOV  AH, 0Eh
13 INT  10h
14 POP  AX
15 pozitif: CALL print_num_uns
16 POP  AX             ; AX = FFE7h geri
▶ RET
18 print_num ENDP
```



RET'ler: SP = FFFEh. Kütüphanedeki PRINT_NUM ayrıca AX = 0 için doğrudan '0' yazar.

Adım 8 / 8

DEĞİŞKENLER

AX = **FFE7h** | SP = **FFFEh** | SF = **0**

ÇIKTI

```
-25
```

PROSEDÜRLER

Basamak Ayırma: İki Yöntem

Ölçüt	Yığınlı yöntem (10'a böl, it, ters sırayla çek)	Azalan bölen (10 000, 1000, ...)
Bölme sayısı	basamak sayısı kadar	beşe kadar, ayrıca böleni küçültmek için birer bölme
Yığın kullanımı	basamak başına 2 bayt	yalnızca saklanan yazmaçlar
Baştaki sıfırlar	kendiliğinden oluşmaz	bayrakla atlanır
Sabit genişlik (`00042`)	ek kod gerekir	bayrak kaldırılınca kendiliğinden
Başka taban	bölen 2, 8 ya da 16 yapılır; 16'da 9'dan büyük basamaklar 'A'–'F' olur	bölen dizisi tabana göre değişir (16'da 4096, 256, 16, 1)
Çıkış sırası	basamaklar önce yığına, sonra ters sırayla ekrana	soldan sağa, hesaplandığı anda

İki yöntem de aynı çıktıyı verir: 1205 için iki yöntem de 1, 2, 0, 5 basamaklarını yazar. Seçimi yığın kullanımı, sabit genişlik ve taban gereksinimi belirler.

PROSEDÜRLER

Demo: PRINT_NUM ve PRINT_NUM_UNS

AX'e verilen değer (onluk, negatif ya da `h` ekli onaltılık) için iki prosedürün ekrana ne yazacağı gösterilir.

AX = -25 | −25 | 32767 | 8000h | FFFFh

AX
FFE7h
1111 1111 1110 0111

PRINT_NUM
−25

PRINT_NUM_UNS
65511

Bit 15 = 1: işaretli yorumda negatif (65511 − 65536 = -25).

ÖRNEK PROGRAM · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Sayı Oku ve Yazdır

```
1  include 'emu8086.inc'
2  ORG     100h
▶  LEA     SI, msg1        ; ask for the number
4  CALL    print_string    ;
5  CALL    scan_num        ; get number in CX.
6  MOV     AX, CX          ; copy the number to AX.
7  CALL    pthis           ; print the following string:
8  DB  13, 10, 'You have entered: ', 0
9  CALL    print_num       ; print number in AX.
10 RET                     ; return to operating system.
11 msg1    DB  'Enter the number: ', 0
12 DEFINE_SCAN_NUM
13 DEFINE_PRINT_STRING
14 DEFINE_PRINT_NUM
15 DEFINE_PRINT_NUM_UNS    ; required for print_num.
16 DEFINE_PTHIS
17 END                     ; directive to stop the compiler.
```

`LEA SI, msg1` : SI ← msg1 dizgisinin bağıl konumu (offset).

Adım 1 / 9

DEĞİŞKENLER

SI = **msg1** | CX = **0000h** | AX = **0000h** | SP = **FFFEh**

ÇAĞRI YIĞINI

main

ÇIKTI

ÖRNEK PROGRAM · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Sayı Oku ve Yazdır

```
1  include 'emu8086.inc'
2  ORG     100h
3  LEA     SI, msg1        ; ask for the number
▶ CALL    print_string    ;
5  CALL    scan_num        ; get number in CX.
6  MOV     AX, CX          ; copy the number to AX.
7  CALL    pthis           ; print the following string:
8  DB  13, 10, 'You have entered: ', 0
9  CALL    print_num       ; print number in AX.
10 RET                     ; return to operating system.
11 msg1    DB  'Enter the number: ', 0
12 DEFINE_SCAN_NUM
13 DEFINE_PRINT_STRING
14 DEFINE_PRINT_NUM
15 DEFINE_PRINT_NUM_UNS   ; required for print_num.
16 DEFINE_PTHIS
17 END                     ; directive to stop the compiler.
```

`CALL print_string` : dönüş adresi yığına itilir (SP = FFFCh); prosedür DS:SI'deki dizgiyi 0'a kadar yazar.

Adım 2 / 9

DEĞİŞKENLER

SI = **msg1** | CX = **0000h** | AX = **0000h** | SP = **FFFCh**

ÇAĞRI YIĞINI

- **print_string**
- main

ÇIKTI

```
Enter the number:
```

ÖRNEK PROGRAM · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Sayı Oku ve Yazdır

```
1  include 'emu8086.inc'
2  ORG     100h
3  LEA     SI, msg1        ; ask for the number
4  CALL    print_string    ;
▶  CALL    scan_num        ; get number in CX.
6  MOV     AX, CX          ; copy the number to AX.
7  CALL    pthis           ; print the following string:
8  DB  13, 10, 'You have entered: ', 0
9  CALL    print_num       ; print number in AX.
10 RET                     ; return to operating system.
11 msg1    DB  'Enter the number: ', 0
12 DEFINE_SCAN_NUM
13 DEFINE_PRINT_STRING
14 DEFINE_PRINT_NUM
15 DEFINE_PRINT_NUM_UNS    ; required for print_num.
16 DEFINE_PTHIS
17 END                     ; directive to stop the compiler.
```

`CALL scan_num` : kullanıcı **-25** yazıp Enter'a basar; karakterler ekrana yankılanır.

Adım 3 / 9

DEĞİŞKENLER

`SI = msg1` `CX = 0000h` `AX = 0000h` `SP = FFFCh`

ÇAĞRI YIĞINI

- **scan_num**
- main

ÇIKTI

```
Enter the number: -25
```

Kod İzleme: Sayı Oku ve Yazdır

```
1  include 'emu8086.inc'
2  ORG     100h
3  LEA     SI, msg1        ; ask for the number
4  CALL    print_string    ;
▶ 5  CALL    scan_num        ; get number in CX.
6  MOV     AX, CX          ; copy the number to AX.
7  CALL    pthis           ; print the following string:
8  DB  13, 10, 'You have entered: ', 0
9  CALL    print_num       ; print number in AX.
10 RET                     ; return to operating system.
11 msg1    DB  'Enter the number: ', 0
12 DEFINE_SCAN_NUM
13 DEFINE_PRINT_STRING
14 DEFINE_PRINT_NUM
15 DEFINE_PRINT_NUM_UNS   ; required for print_num.
16 DEFINE_PTHIS
17 END                     ; directive to stop the compiler.
```

Dönüş: **CX = FFE7h** = −25 (ikiye tümleyen). SP yeniden FFFEh.

Adım 4 / 9

DEĞİŞKENLER

SI = **msg1** | CX = **FFE7h** | AX = **0000h** | SP = **FFFEh**

ÇAĞRI YIĞINI

main

ÇIKTI

```
Enter the number: -25
```

ÖRNEK PROGRAM · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Sayı Oku ve Yazdır

```
1  include 'emu8086.inc'
2  ORG     100h
3  LEA     SI, msg1        ; ask for the number
4  CALL    print_string    ;
5  CALL    scan_num        ; get number in CX.
▶ MOV     AX, CX          ; copy the number to AX.
7  CALL    pthis           ; print the following string:
8  DB  13, 10, 'You have entered: ', 0
9  CALL    print_num       ; print number in AX.
10 RET                     ; return to operating system.
11 msg1    DB  'Enter the number: ', 0
12 DEFINE_SCAN_NUM
13 DEFINE_PRINT_STRING
14 DEFINE_PRINT_NUM
15 DEFINE_PRINT_NUM_UNS   ; required for print_num.
16 DEFINE_PTHIS
17 END                     ; directive to stop the compiler.
```

`MOV AX, CX` : PRINT_NUM sayıyı AX'te bekler.

Adım 5 / 9

DEĞİŞKENLER

SI = **msg1** | CX = **FFE7h** | AX = **FFE7h** | SP = **FFFEh**

ÇAĞRI YIĞINI

main

ÇIKTI

```
Enter the number: -25
```

Kod İzleme: Sayı Oku ve Yazdır

```
1  include 'emu8086.inc'
2  ORG     100h
3  LEA     SI, msg1        ; ask for the number
4  CALL    print_string    ;
5  CALL    scan_num        ; get number in CX.
6  MOV     AX, CX          ; copy the number to AX.
▶  CALL    pthis           ; print the following string:
8  DB  13, 10, 'You have entered: ', 0
9  CALL    print_num       ; print number in AX.
10 RET                     ; return to operating system.
11 msg1    DB  'Enter the number: ', 0
12 DEFINE_SCAN_NUM
13 DEFINE_PRINT_STRING
14 DEFINE_PRINT_NUM
15 DEFINE_PRINT_NUM_UNS    ; required for print_num.
16 DEFINE_PTHIS
17 END                     ; directive to stop the compiler.
```

`CALL pthis` : yığına itilen “dönüş adresi” aslında bir sonraki satırdaki **dizginin adresidir**.

Adım 6 / 9

DEĞİŞKENLER

`SI = msg1` `CX = FFE7h` `AX = FFE7h` `SP = FFFCh`

ÇAĞRI YIĞINI

- **pthis**
- main

ÇIKTI

```
Enter the number: -25
```

ÖRNEK PROGRAM · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Sayı Oku ve Yazdır

```
1  include 'emu8086.inc'
2  ORG     100h
3  LEA     SI, msg1        ; ask for the number
4  CALL    print_string    ;
5  CALL    scan_num        ; get number in CX.
6  MOV     AX, CX          ; copy the number to AX.
7  CALL    pthis           ; print the following string:
▶  DB  13, 10, 'You have entered: ', 0
9  CALL    print_num       ; print number in AX.
10 RET                     ; return to operating system.
11 msg1    DB  'Enter the number: ', 0
12 DEFINE_SCAN_NUM
13 DEFINE_PRINT_STRING
14 DEFINE_PRINT_NUM
15 DEFINE_PRINT_NUM_UNS    ; required for print_num.
16 DEFINE_PTHIS
17 END                     ; directive to stop the compiler.
```

PTHIS dizgiyi yazar: 13, 10 (yeni satır) ve “You have entered: ”. Sonra dönüş adresini 0'ın ötesine taşır.

Adım 7 / 9

DEĞİŞKENLER

SI = **msg1** | CX = **FFE7h** | AX = **FFE7h** | SP = **FFFCh**

ÇAĞRI YIĞINI

- **pthis**
- main

ÇIKTI

```
Enter the number: -25
You have entered:
```

ÖRNEK PROGRAM · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Sayı Oku ve Yazdır

```
1  include 'emu8086.inc'
2  ORG     100h
3  LEA     SI, msg1        ; ask for the number
4  CALL    print_string    ;
5  CALL    scan_num        ; get number in CX.
6  MOV     AX, CX          ; copy the number to AX.
7  CALL    pthis           ; print the following string:
8  DB  13, 10, 'You have entered: ', 0
▶ CALL    print_num       ; print number in AX.
10 RET                     ; return to operating system.
11 msg1    DB  'Enter the number: ', 0
12 DEFINE_SCAN_NUM
13 DEFINE_PRINT_STRING
14 DEFINE_PRINT_NUM
15 DEFINE_PRINT_NUM_UNS    ; required for print_num.
16 DEFINE_PTHIS
17 END                     ; directive to stop the compiler.
```

Dönüş, dizginin **sonrasına** yapılır. `CALL print_num` : AX = FFE7h işaretli yazılır.

Adım 8 / 9

DEĞİŞKENLER

`SI = msg1` `CX = FFE7h` `AX = FFE7h` `SP = FFFCh`

ÇAĞRI YIĞINI

- **print_num**
- main

ÇIKTI

```
Enter the number: -25
You have entered: -25
```

ÖRNEK PROGRAM · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Sayı Oku ve Yazdır

```
1  include 'emu8086.inc'
2  ORG     100h
3  LEA     SI, msg1        ; ask for the number
4  CALL    print_string    ;
5  CALL    scan_num        ; get number in CX.
6  MOV     AX, CX          ; copy the number to AX.
7  CALL    pthis           ; print the following string:
8  DB  13, 10, 'You have entered: ', 0
9  CALL    print_num       ; print number in AX.
▶ RET                      ; return to operating system.
11 msg1    DB  'Enter the number: ', 0
12 DEFINE_SCAN_NUM
13 DEFINE_PRINT_STRING
14 DEFINE_PRINT_NUM
15 DEFINE_PRINT_NUM_UNS    ; required for print_num.
16 DEFINE_PTHIS
17 END                     ; directive to stop the compiler.
```

`RET` : yığındaki 0000h IP'ye alınır, SP = 0000h; DOS'a dönüş. Satır 11–17 veri ve prosedür tanımlarıdır, akışta çalıştırılmaz.

Adım 9 / 9

DEĞİŞKENLER

`SI = msg1` `CX = FFE7h` `AX = FFE7h` `SP = 0000h`

ÇAĞRI YIĞINI

`main`

ÇIKTI

```
Enter the number: -25
You have entered: -25
```

ÖRNEK PROGRAM

PTHIS Nasıl Çalışır?

kod kesimi (bellek, soldan sağa artan adres)



yığın:  ← SP = FFFEh

Bellekte `CALL pthis` 'in hemen ardından dizgi baytları gelir; dizgi 0 ile biter.

Adım 1 / 5

Bu teknik, dönüş adresinin yığında saklandığı bilgisine dayanır. Benzer şekilde PRINT makrosu da veriyi kod içine gömer ama `JMP` ile atlar.

ÖRNEK PROGRAM

PTHIS Nasıl Çalışır?

kod kesimi (bellek, soldan sağa artan adres)



yığın: `adres(0Dh)` ← SP = FFFCh

`CALL` , “sonraki komutun” adresini iter. Bu adres, dizginin ilk baytı olan **0Dh**'nin adresidir.

Adım 2 / 5

Bu teknik, dönüş adresinin yığında saklandığı bilgisine dayanır. Benzer şekilde PRINT makrosu da veriyi kod içine gömer ama `JMP` ile atlar.

ÖRNEK PROGRAM

PTHIS Nasıl Çalışır?

kod kesimi (bellek, soldan sağa artan adres)



yığın: `adres(0Dh)` ← SP = FFFCh

PTHIS yığındaki adresi okur ve oradan itibaren baytları 0'a kadar ekrana yazar.

Bu teknik, dönüş adresinin yığında saklandığı bilgisine dayanır. Benzer şekilde PRINT makrosu da veriyi kod içine gömer ama `JMP` ile atlar.

ÖRNEK PROGRAM

PTHIS Nasıl Çalışır?

kod kesimi (bellek, soldan sağa artan adres)



yığın: `adres(CALL print_num)` ← SP = FFFCh

Yığındaki adres, 0 baytının bir ötesiyle değiştirilir: artık `CALL print_num` 'u gösterir.

Bu teknik, dönüş adresinin yığında saklandığı bilgisine dayanır. Benzer şekilde PRINT makrosu da veriyi kod içine gömer ama `JMP` ile atlar.

ÖRNEK PROGRAM

PTHIS Nasıl Çalışır?

kod kesimi (bellek, soldan sağa artan adres)



yığın:  ← SP = FFFEh

`RET` : işlemci dizgiyi **atlayıp** `CALL print_num` 'dan devam eder. Veri komut sanılıp çalıştırılmaz.

Adım 5 / 5

Bu teknik, dönüş adresinin yığında saklandığı bilgisine dayanır. Benzer şekilde PRINT makrosu da veriyi kod içine gömer ama `JMP` ile atlar.

ÖRNEK PROGRAM

Örnek: İki Sayının Toplamı

```
include 'emu8086.inc'
ORG  100h
CALL scan_num      ; 1. sayı → CX
MOV  BX, CX        ; sakla
PUTC 13
PUTC 10            ; alt satır
CALL scan_num      ; 2. sayı → CX
PUTC 13
PUTC 10
MOV  AX, BX
ADD  AX, CX        ; işaretli toplam
JO   tasma
CALL print_num
RET
tasma: PUTC '?'
RET
DEFINE_SCAN_NUM
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS
END
```

- İkinci SCAN_NUM da sonucu CX'e yazar: birinci sayı önce BX'e alınır.
- Enter yalnızca satır başına döndürür; `PUTC 13` , `PUTC 10` sonraki girdiyi alt satıra taşır.
- `JO` işaretli taşmayı yakalar: 30 000 + 5000 = 35 000, 16 bitlik işaretli aralığı (32 767) aşar.

Girdiler	Ekran (son satır)
120, -45	75
-7, -8	-15
30000, 5000	?

ÖRNEK PROGRAM

Örnek: Çerçeveli Başlık

```
include 'emu8086.inc'
ORG  100h
CALL clear_screen
GOTOXY 10, 2
CALL cizgi
GOTOXY 10, 3
PRINT '|  BASLIK  |'
GOTOXY 10, 4
CALL cizgi
GOTOXY 0, 6       ; imleç alta
RET
cizgi PROC        ; +----------+
PUTC '+'
MOV  CX, 10
tire: PUTC '-'
LOOP tire
PUTC '+'
RET
cizgi ENDP
DEFINE_CLEAR_SCREEN
END
```



- Kendi prosedürümüz (`cizgi`) kütüphane makrolarıyla birlikte kullanılır; tanımı RET'ten sonradır.
- Döngüdeki `PUTC '-'` bir kez açılır, 10 kez çalışır: kod 10 kez kopyalanmaz.
- PUTC yalnızca AX'i kullanır ve geri yükler; CX'e dokunmadığı için döngü sayacı güvendedir.

ÖRNEK PROGRAM

Sık Yapılan Hatalar

DEFINE_ unutmak

`CALL scan_num` var ama `DEFINE_SCAN_NUM` yoksa derleyici etiketi bulamaz: *undefined* hatası.

DEFINE_'nin yeri

Her DEFINE_ makrosu `JMP skip_proc_…` ile başlar; programın başına konsa da işlemci gövdeyi atlar. Alışılmış yer yine `RET` ile `END` arasıdır.

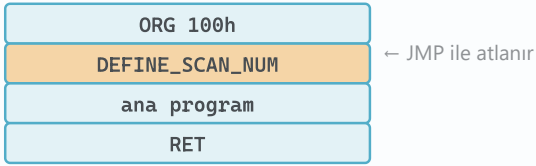
Dizgiyi 0 ile bitirmemek

PRINT_STRING ve PTHIS 0 görene kadar yazar; 0 yoksa bellekteki rastgele baytlar ekrana dökülür.

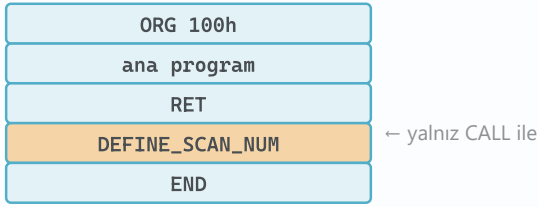
Yazmaçları varsaymak

Prosedürler sonuçlarını belirli yazmaçlarda döndürür (SCAN_NUM → CX). Başka yazmaçtaki değerin korunduğu belgede yazmıyorsa varsayılmamalıdır.

Başta: gereksiz JMP



Alışılmış yer



ÖRNEK PROGRAM

Sayı ve Ekran Hataları

Sonucu AX'te aramak

SCAN_NUM sonucu CX'e yazar. PRINT_NUM'dan önce `MOV AX, CX` unutulursa AX'teki eski değer yazılır.

İşaretsizi işaretli yazmak

AX = 40 000 (9C40h) PRINT_NUM ile −25536 görünür. 32 767'yi aşabilen işaretsiz değerler PRINT_NUM_UNS ile yazılır.

Enter'dan sonra alt satır beklemek

SCAN_NUM'dan sonra imleç girdinin satır başındadır; sonraki çıktı girilen sayının üstüne yazılır. Araya CR + LF gerekir.

Satır ile sütunu karıştırmak

`GOTOXY 5, 10` imleci 5. sütun, 10. satıra götürür; tersi değil.

Açılımın bozduğu yazmacı vermek

`GOTOXY AL, AH` : makro AH'ye önce 02h yazdığı için satır 02h olur. Konum önce başka yazmaçlara alınır.

Tampon boyunu büyük vermek

DX, tamponun gerçek boyundan büyük verilirse GET_STRING tamponun ardındaki değişkenlerin üstüne yazar.

Laboratuvar: emu8086.inc Kütüphanesi

Makrolarla ekran (PRINT, GOTOXY, PUTC)

KOD · **69 bayt** Açılım

```
  1 include emu8086.inc
  2 ORG    100h
  3 PRINT 'Hello World!'
+ PUSH    AX
+ PUSH    SI
+ JMP     ??0003
+ ??0001 DB 'Hello World!', 0
+ ??0003:
+ LEA     SI, ??0001
+ ??0000:
+ MOV     AL, CS:[SI]
+ CMP     AL, 0
+ JZ      ??0002
+ INC     SI
```

YAZMAÇLAR

AX	0000	AH 00	AL 00		
BX	0000	BH 00	BL 00		
CX	0045	CH 00	CL 45		
DX	0000	DH 00	DL 00		
SI	0000	DI 0000	BP 0000		
SP	FFFE	CS 0700	IP 0100		
DS	0700	ES 0700	SS 0700		

BAYRAKLAR

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

CS:IP = 0700:0100 · adım 0

BELLEK DS = 0700h

```
0100 50 56 EB 0D 48 65 6C 6C
0108 6F 20 57 6F 72 6C 64 21
0110 00 BE 04 01 2E 8A 04 3C
```

EKRAN

Program yüklendi: 69 bayt, CS:IP = 0700:0100h. emu8086'daki gibi CX başlangıçta program boyutunu (0045h) gösterir. Adım düğmesiyle komutlar tek tek yürütülür; vurgulu satır sıradaki komuttur.

ÖZET

Özet

INCLUDE

`include 'emu8086.inc'`: önce kaynak klasörü, sonra Inc klasörü aranır.

Makrolar

PUTC, GOTOXY (col, row), PRINT, PRINTN, CURSOROFF, CURSORON: INT 10h üzerine kurulu, kullanıldıkları yere açılır.

Prosedürler

END'den önce `DEFINE_…`, kullanırken `CALL`. Girdi/çıktı yazmaçları: SI, DI, DX, CX, AX.

Sayılar

SCAN_NUM → CX (işaretli); PRINT_NUM AX'i işaretli, PRINT_NUM_UNS işaretsiz yazar.

PTHIS

Dizgi CALL'dan hemen sonra; prosedür dönüş adresini dizginin sonrasına taşır.

Dizgiler

PRINT dizgiyi koda gömer; PRINT_STRING ve PTHIS tek kopya kodla 0 ile biten dizgi yazar; GET_STRING tamponu 0 ile bitirir.

ÖZET

Alıştırmalar

1. Ekranın ortasına (sütun 35, satır 12) “MERKEZ” yazan programı makrolarla yazın.
2. Kullanıcıdan iki sayı okuyup toplamını yazdıran programı yazın.
3. `PRINT` makrosu 10 kez kullanılırsa dizgiler ve döngü kodu kaç kez bulunur? PRINT_STRING ile karşılaştırın.
4. AX = 8000h iken PRINT_NUM ve PRINT_NUM_UNS ne yazar?
5. `DEFINE_PTHIS` , `ORG 100h` 'nin hemen altına yazılırsa ne olur?
6. Kullanıcının adını GET_STRING ile alıp “Merhaba, ad!” yazdırın.

▶ **İpuçları / cevaplar**

ÖZET

Kaynaklar

- emu8086 yardım belgeleri: “emu8086.inc — library of common functions” ve “Macros” sayfaları.
- emu8086 kurulumundaki `Inc/emu8086.inc` dosyası (makro ve prosedürlerin kaynak kodu).
- B. B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson, 2009 — prosedürler ve makrolar.
- A. S. Tanenbaum, *Modern Operating Systems*, 3. baskı, Pearson, 2008.

Son

Bölüm 9.2: emu8086.inc Kütüphanesi

Sonraki bölüm: Bölüm 10.1: 8086 Pinleri · Terim sözlüğü