

Bölüm 5: emu8086.inc Kütüphanesi

Mikroişlemciler

INCLUDE direktifi

Makrolar

PUTC · GOTOXY

PRINT · PRINTN

İmleç makroları

Prosedürler

Sayı okuma / yazma

PTHS ve yığın

CALL ve RET

Makro mu, prosedür mü?

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

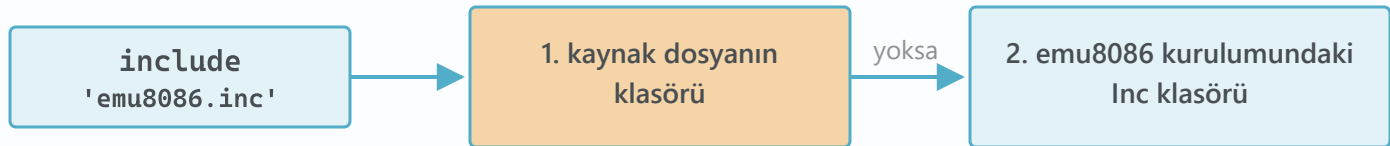
`DEFINE_...` ve `CALL` ile hazır prosedürleri (sayı okuma/yazma, dizgi yazdırma) kullanmak.

CALL / RET

Yığın üzerinden dönüş adresinin nasıl saklandığını izlemek; makro ile prosedürü boyut ve hız açısından karşılaştırmak.

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.

Kütüphanede Neler Var?

Makrolar · doğrudan kullanılır

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

Prosedürler · `DEFINE_` + `CALL`

Prosedür	Görev
<code>PRINT_STRING</code>	DS:SI'deki dizgiyi yaz
<code>PTTHIS</code>	CALL'dan sonraki dizgiyi yaz
<code>GET_STRING</code>	klavyeden dizgi oku
<code>CLEAR_SCREEN</code>	ekranı temizle
<code>SCAN_NUM</code>	sayı oku → CX
<code>PRINT_NUM</code>	AX'i işaretli yaz
<code>PRINT_NUM_UNSI</code>	AX'i işaretsiz yaz

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

MAKROLAR

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 .

MAKROLAR

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).

PRINT Neye Genişler?

```
PUSH AX
PUSH SI
JMP skip_dcl          ; veriyi atla
s_dcl DB 'Hello World!', 0
skip_dcl:
LEA SI, s_dcl
next_char:
MOV AL, CS:[SI]       ; karakteri al
CMP AL, 0
JZ printed            ; 0 → bitti
INC SI
MOV AH, 0Eh
INT 10h
JMP next_char
printed:
POP SI
POP AX
```

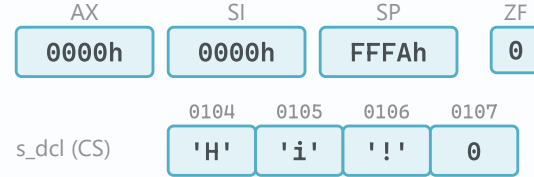
- Dizgi, makronun açıldığı yere, **kodun içine** gömülür; `JMP` ile üzerinden atlanır.
- Sonuna **0** eklenir (null ile sonlanan dizgi).
- Etiketler `LOCAL` tanımlıdır: makro her açılışta yeni adlar alır, çakışma olmaz.
- Veri kod kesiminde olduğu için `CS:` `[SI]` kesim öneki kullanılır.
- PRINTN'de dizgi `DB sdat, 13, 10, 0` olarak tanımlanır.

Kod İzleme: PRINT 'Hi!' Açılımı

► PUSH AX

► PUSH SI

```
3 JMP skip_dcl ; veriyi atla
4 s_dcl DB 'Hi!', 0
5 skip_dcl: LEA SI, s_dcl
6 next_char: MOV AL, CS:[SI]
7 CMP AL, 0
8 JZ printed ; 0 → bitti
9 INC SI
10 MOV AH, 0Eh
11 INT 10h
12 JMP next_char
13 printed: POP SI
14 POP AX
15 RET
```



ekran

PRINT 'Hi!' derlemede bu koda açılır (emu8086.inc'teki makronun eşdeğeri). Önce AX ve SI yığına: SP = FFFAh.

Adım 1 / 9

DEĞİŞKENLER

AL = 00h

SI = 0000h

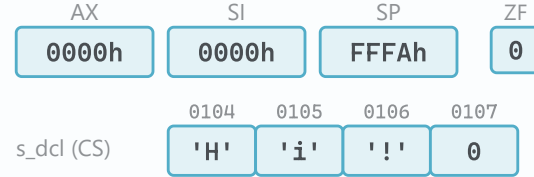
SP = FFFAh

ZF = 0

ÇIKTI

Kod İzleme: PRINT 'Hi!' Açılımı

```
1 PUSH AX
2 PUSH SI
▶ 3 JMP skip_dcl ; veriyi atla
4 s_dcl DB 'Hi!', 0
5 skip_dcl: LEA SI, s_dcl
6 next_char: MOV AL, CS:[SI]
7 CMP AL, 0
8 JZ printed ; 0 → bitti
9 INC SI
10 MOV AH, 0Eh
11 INT 10h
12 JMP next_char
13 printed: POP SI
14 POP AX
15 RET
```



ekran

JMP skip_dcl : kodun içine gömülü dizgi baytları (0104h-0107h) komut sanılmasın diye atlanır.

Adım 2 / 9

DEĞİŞKENLER

AL = 00h

SI = 0000h

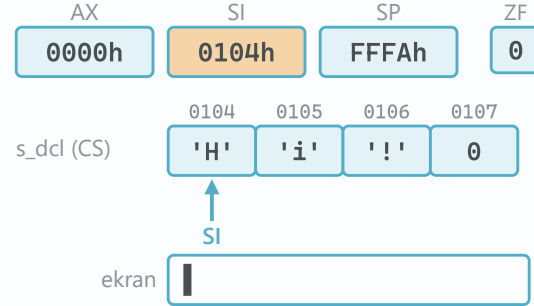
SP = FFFAh

ZF = 0

ÇIKTI

Kod İzleme: PRINT 'Hi!' Açılımı

```
1 PUSH AX
2 PUSH SI
3 JMP skip_dcl ; veriyi atla
4 s_dcl DB 'Hi!', 0
▶ skip_dcl: LEA SI, s_dcl
6 next_char: MOV AL, CS:[SI]
7 CMP AL, 0
8 JZ printed ; 0 → bitti
9 INC SI
10 MOV AH, 0Eh
11 INT 10h
12 JMP next_char
13 printed: POP SI
14 POP AX
15 RET
```



LEA SI, s_dcl : SI = 0104h, dizginin ilk baytı.

Adım 3 / 9

DEĞİŞKENLER

AX = 0000h

SI = 0104h

SP = FFFAh

ZF = 0

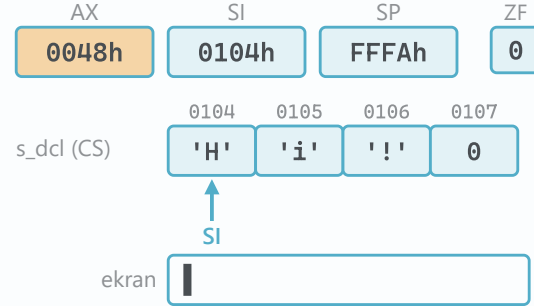
ÇIKTI

Kod İzleme: PRINT 'Hi!' Açılımı

```

1 PUSH AX
2 PUSH SI
3 JMP skip_dcl ; veriyi atla
4 s_dcl DB 'Hi!', 0
5 skip_dcl: LEA SI, s_dcl
▶ next_char: MOV AL, CS:[SI]
▶ CMP AL, 0
▶ JZ printed ; 0 → bitti
9 INC SI
10 MOV AH, 0Eh
11 INT 10h
12 JMP next_char
13 printed: POP SI
14 POP AX
15 RET

```



AL = CS:[SI] = 'H' (48h). $48h - 0 \neq 0 \rightarrow ZF = 0$, JZ atlamaz.

Adım 4 / 9

DEĞİŞKENLER

AL = 48h

SI = 0104h

SP = FFFAh

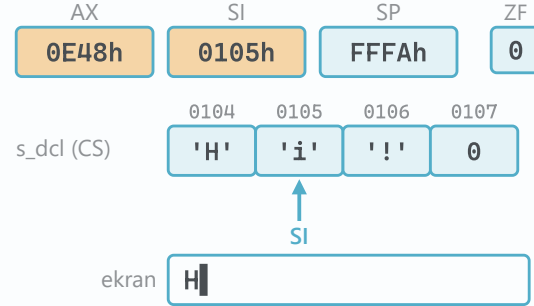
ZF = 0

ÇIKTI



Kod İzleme: PRINT 'Hi!' Açılımı

```
1 PUSH AX
2 PUSH SI
3 JMP skip_dcl ; veriyi atla
4 s_dcl DB 'Hi!', 0
5 skip_dcl: LEA SI, s_dcl
6 next_char: MOV AL, CS:[SI]
7 CMP AL, 0
8 JZ printed ; 0 → bitti
▶ INC SI
▶ MOV AH, 0Eh
▶ INT 10h
12 JMP next_char
13 printed: POP SI
14 POP AX
15 RET
```



SI = 0105h; AH = 0Eh ile INT 10h AL'deki 'H'yi yazar.

Adım 5 / 9

DEĞİŞKENLER

AL = 48h

SI = 0105h

SP = FFFAh

ZF = 0

ÇIKTI

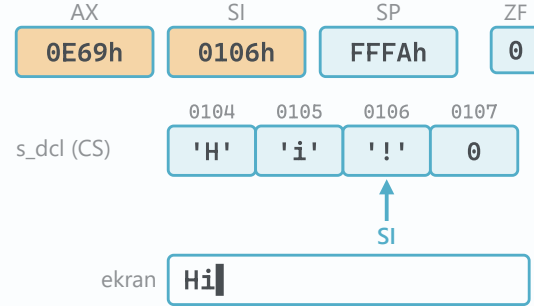
H

Kod İzleme: PRINT 'Hi!' Açılımı

```

1 PUSH AX
2 PUSH SI
3 JMP skip_dcl ; veriyi atla
4 s_dcl DB 'Hi!', 0
5 skip_dcl: LEA SI, s_dcl
▶ next_char: MOV AL, CS:[SI]
▶ CMP AL, 0
▶ JZ printed ; 0 → bitti
▶ INC SI
▶ MOV AH, 0Eh
▶ INT 10h
▶ JMP next_char
13 printed: POP SI
14 POP AX
15 RET

```



JMP next_char : aynı döngü 'i' (69h) için çalışır; SI = 0106h.

Adım 6 / 9

DEĞİŞKENLER

AL = 69h

SI = 0106h

SP = FFFAh

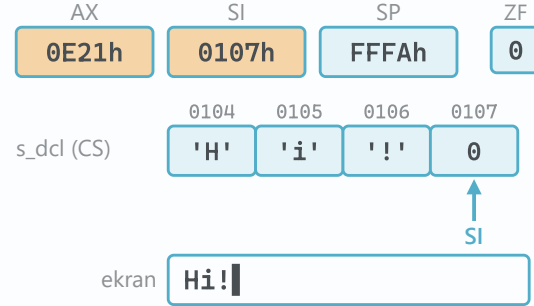
ZF = 0

ÇIKTI

Hi

Kod İzleme: PRINT 'Hi!' Açılımı

```
1 PUSH AX
2 PUSH SI
3 JMP skip_dcl ; veriyi atla
4 s_dcl DB 'Hi!', 0
5 skip_dcl: LEA SI, s_dcl
▶ next_char: MOV AL, CS:[SI]
▶ CMP AL, 0
▶ JZ printed ; 0 → bitti
▶ INC SI
▶ MOV AH, 0Eh
▶ INT 10h
▶ JMP next_char
13 printed: POP SI
14 POP AX
15 RET
```



Üçüncü tur: '!' yazılır, SI = 0107h (0 baytı).

Adım 7 / 9

DEĞİŞKENLER

AX = 21h

SI = 0107h

SP = FFFAh

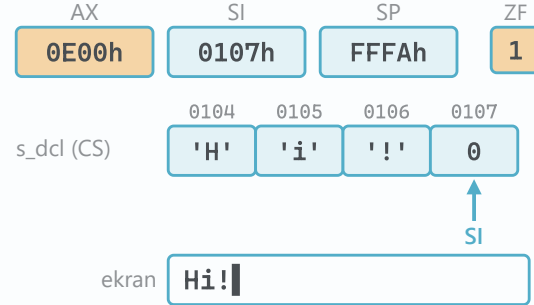
ZF = 0

ÇIKTI

Hi!

Kod İzleme: PRINT 'Hi!' Açılımı

```
1 PUSH AX
2 PUSH SI
3 JMP skip_dcl ; veriyi atla
4 s_dcl DB 'Hi!', 0
5 skip_dcl: LEA SI, s_dcl
▶ next_char: MOV AL, CS:[SI]
▶ CMP AL, 0
▶ JZ printed ; 0 → bitti
9 INC SI
10 MOV AH, 0Eh
11 INT 10h
▶ JMP next_char
13 printed: POP SI
14 POP AX
15 RET
```



AL = 0 → ZF = 1: JZ printed atlar; döngü biter.

Adım 8 / 9

DEĞİŞKENLER

AL = 00h

SI = 0107h

SP = FFFAh

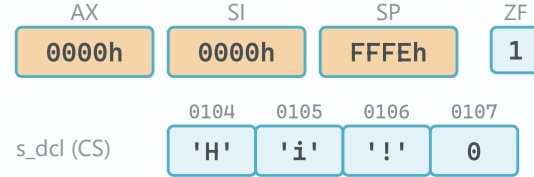
ZF = 1

ÇIKTI

Hi!

Kod İzleme: PRINT 'Hi!' Açılımı

```
1 PUSH AX
2 PUSH SI
3 JMP skip_dcl ; veriyi atla
4 s_dcl DB 'Hi!', 0
5 skip_dcl: LEA SI, s_dcl
6 next_char: MOV AL, CS:[SI]
7 CMP AL, 0
8 JZ printed ; 0 → bitti
9 INC SI
10 MOV AH, 0Eh
11 INT 10h
12 JMP next_char
▶ printed: POP SI
▶ POP AX
15 RET
```



ekran

Hi!

POP SI, POP AX ters sırayla: SI ve AX makrodan önceki değerlerine döner, SP = FFEh.

Adım 9 / 9

DEĞİŞKENLER

AL = 00h

SI = 0000h

SP = FFEh

ZF = 1

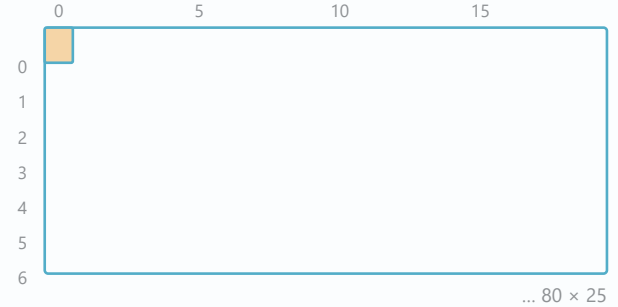
ÇIKTI

Hi!

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

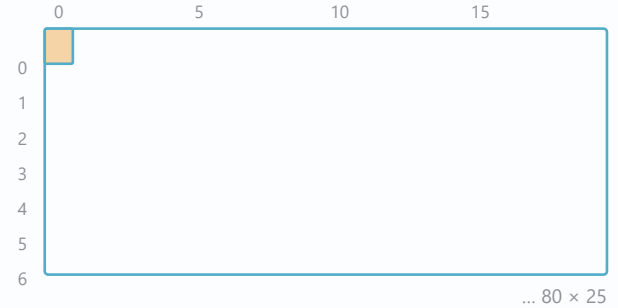
DEĞİŞKENLER

sütun = 0

satır = 0

Kod İzleme: PRINT, GOTOXY, PUTC

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



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

Adım 2 / 7

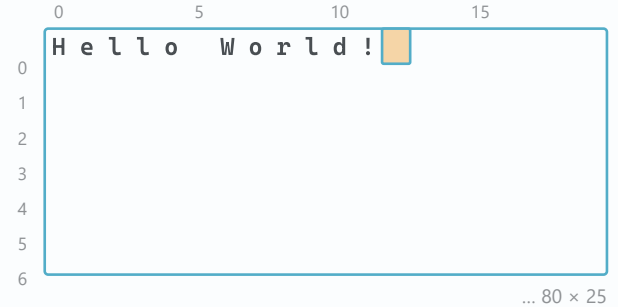
DEĞİŞKENLER

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

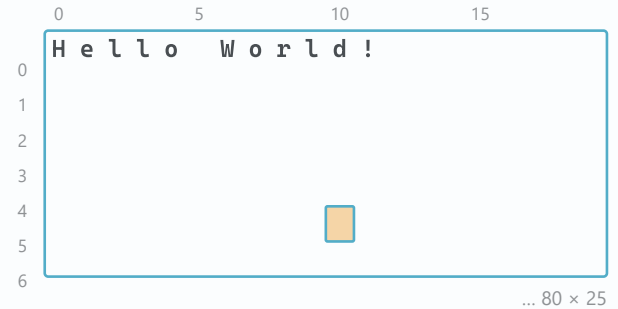
sütun = 12

satır = 0

Kod İzleme: PRINT, GOTOXY, PUTC

```

1 include emu8086.inc
2 ORG 100h
3 PRINT 'Hello World!'
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.
    
```



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

Adım 4 / 7

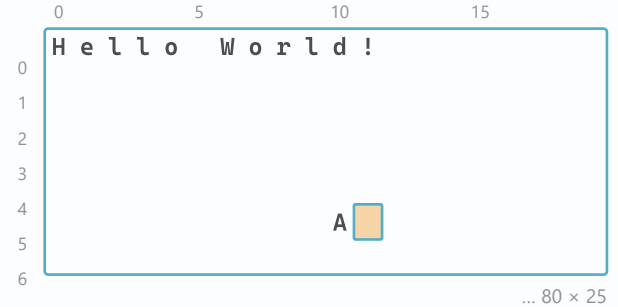
DEĞİŞKENLER

sütun = 10

satır = 5

Kod İzleme: PRINT, GOTOXY, PUTC

```
1 include emu8086.inc
2 ORG 100h
3 PRINT 'Hello World!'
4 GOTOXY 10, 5
▶ 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

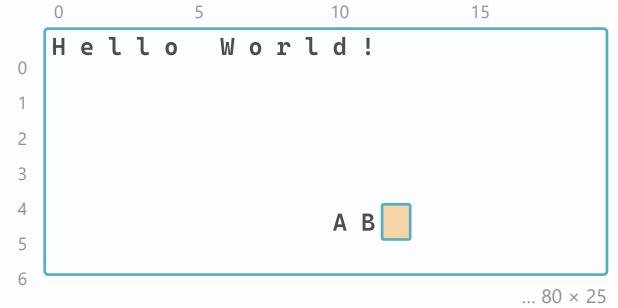
sütun = 11

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

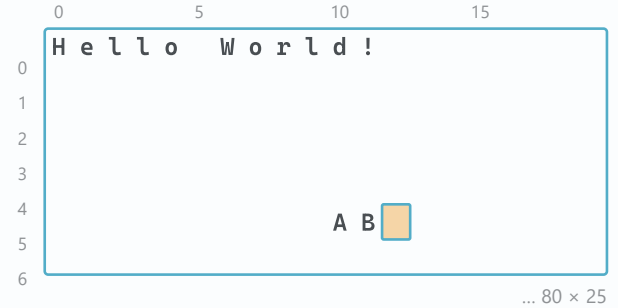
sütun = 12

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'
7 ► 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

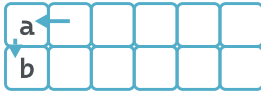
PRINTN ve İmleç Makroları

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

CUSOROFF / CUSORON

İmleç şeklini ayarlayan `INT 10h`, AH = 01h kullanılır: CH = başlangıç, CL = bitiş tarama satırı.

CH'nin 5. biti 1 yapılıncı imleç **gizlenir**; CUSORON normal şekli geri yükler.

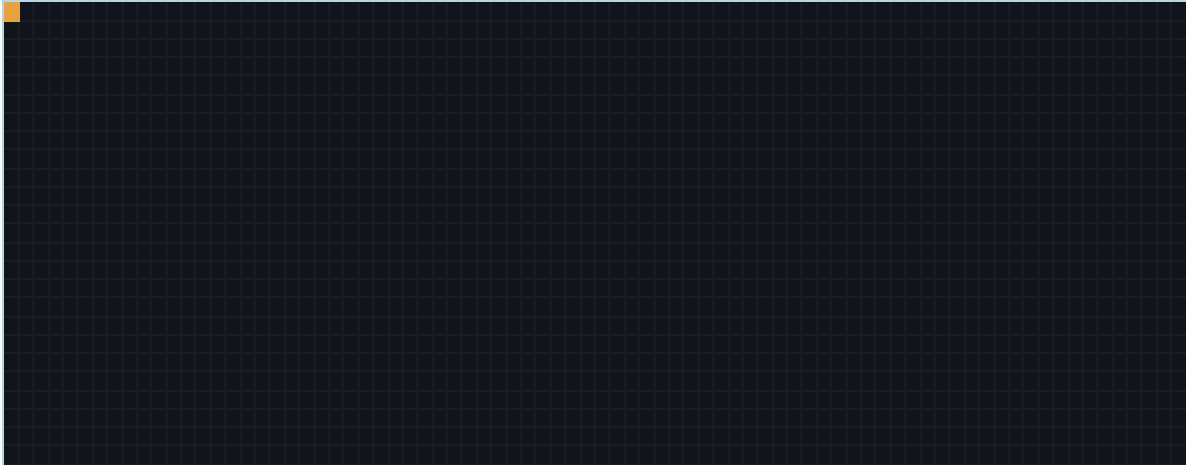
Oyun veya menü ekranlarında yanıp sönen imleci saklamak için kullanılır.

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

Demo: GOTOXY Ekranı

80 × 25 metin ekranında bir hücreye tıklayın: imleci oraya götüren makro, BIOS yazmaçları ve video belleğindeki karşılığı gösterilir.

PUTC
karakteri

temizle

GOTOXY 0, 0 → MOV DL,0 · MOV DH,0 · AH=02h · INT 10h
video belleği: B800:0000h (karakter), B800:0001h (renk)

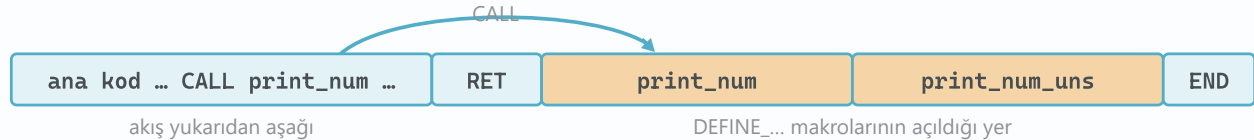
PROSEDÜRLER

Prosedür Kullanımı: DEFINE_ ve CALL

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

DEFINE_PRINT_NUM ; tanımla
DEFINE_PRINT_NUM_UN
END
```

- Her prosedür, `END` direktifinden önce kendi `DEFINE_...` makrosuyla **bir kez** tanımlanmalıdır.
- `DEFINE_...` makrosu, prosedürün kodunu bulunduğu yere açar.
- Tanımlar genellikle `RET` 'ten sonra konur: aksi hâlde işlemci akış içinde prosedür koduna girer.
- Kodda prosedür adıyla `CALL` yapılır; adlar küçük harfle yazılır (`print_num`).
- `PRINT_NUM` , işini yaparken `PRINT_NUM_UN` 'yi çağırır; ikisi birlikte tanımlanmalıdır.



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 **yiğindan** 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.

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` .

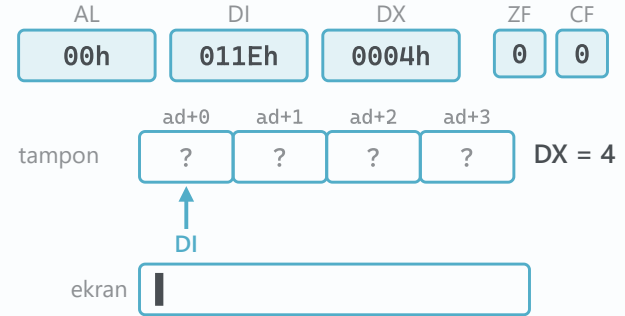
```
LEA DI, ad           ; tampon
MOV DX, 20           ; boyut
CALL get_string
CALL clear_screen
LEA SI, ad
CALL print_string
RET
ad DB 20 DUP(0)
```

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

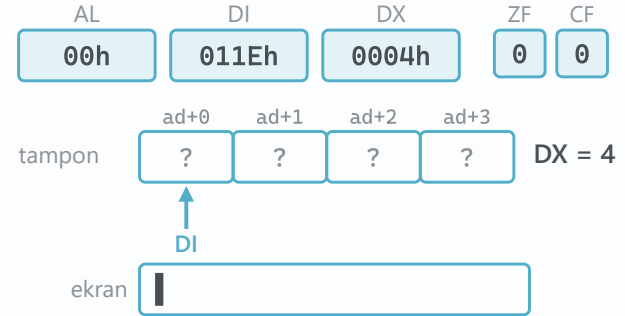
DI = ad+0

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

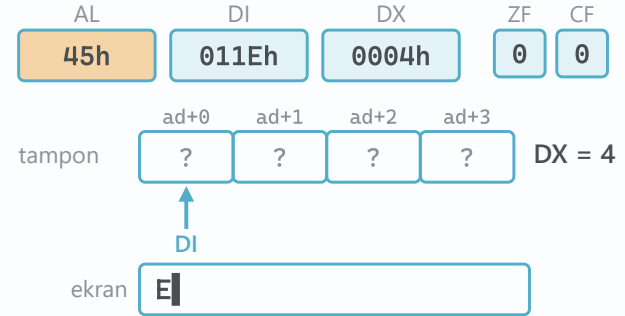


Kod İzleme: GET_STRING Mantığı (DX = 4)

```

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

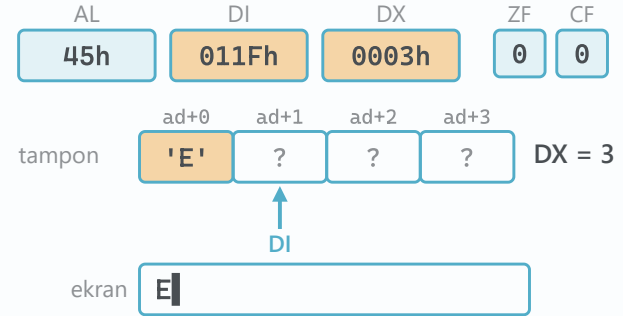
E

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

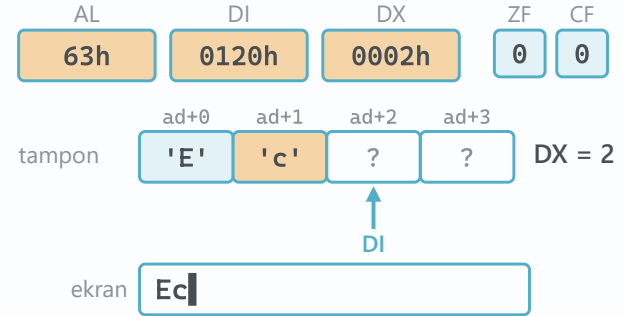
E

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

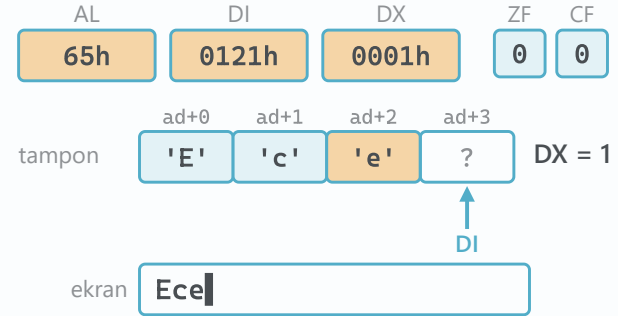
Ec

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

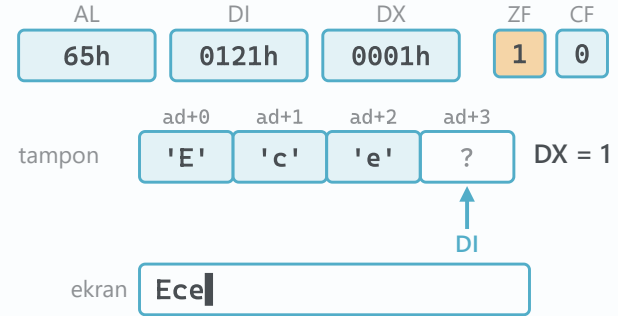
Ece

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

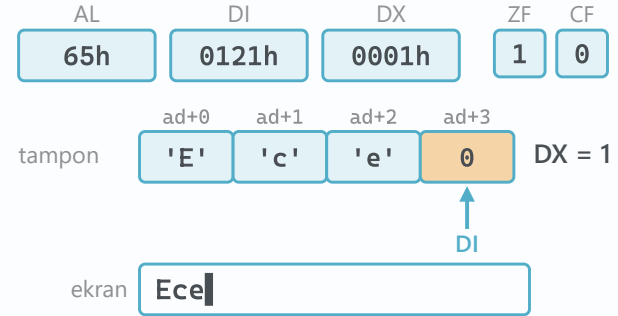
DX = 1

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

Sayı Prosedürleri

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

işaretli aralık

16 bit: -32 768 ... 32 767

işaretsiz aralık

16 bit: 0 ... 65 535

Aynı bitler

FFE7h = -25 (işaretli) =
65 511 (işaretsiz)

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
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
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	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 : $0 \times 10 = 0$; 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

 $3 \times 10 = 30 \text{ (1Eh)} \rightarrow CX = 30 + 0 = 30.$

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. $30 \times 10 = 300$ (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

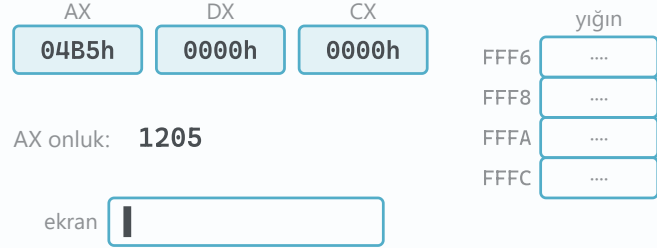
Kod İzleme: PRINT_NUM_UN\$ Mantığı (AX = 1205)

```

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_UN\$ Mantiğı (AX = 1205)

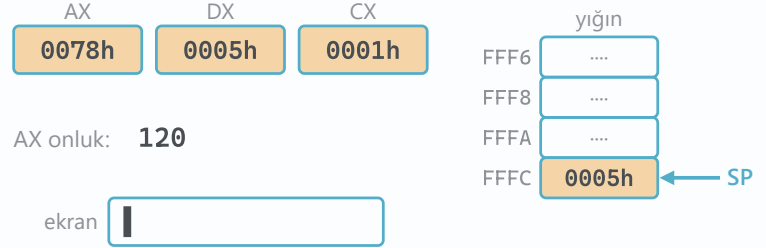
```

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_UN\$ 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 DX CX

000Ch 0000h 0002h

AX onluk: 12

ekran

I

yığın

FFF6	
FFF8	
FFFA	0000h ← SP
FFFC	0005h

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_UN\$ Mantiğı (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	DX	CX
0001h	0002h	0003h

AX onluk: 1

ekran

I

12 / 10 = 1, kalan 2 yığına; CX = 3.

yığın	
FFF6	
FFF8	0002h ← SP
FFFA	0000h
FFFC	0005h

Adım 4 / 10

DEĞİŞKENLER

AX = 0001h

DX = 0002h

CX = 0003h

SP = FFF8h

ÇIKTI

Kod İzleme: PRINT_NUM_UN\$ Mantiğı (AX = 1205)

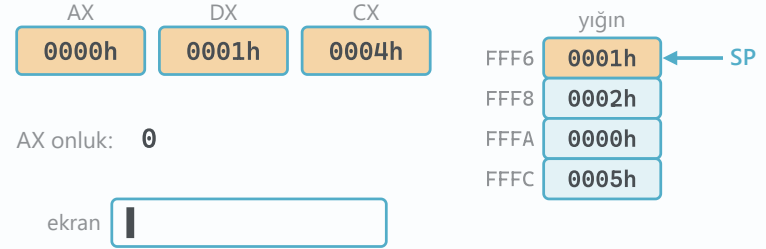
```

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_UN\$ Mantiğı (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
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

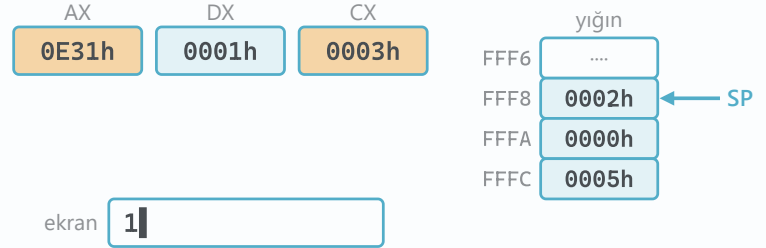


Kod İzleme: PRINT_NUM_UN\$ Mantığı (AX = 1205)

```

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_UN\$ Mantiğı (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

```

AX DX CX

0E32h 0001h 0002h

ekran

12

Sonraki basamak 2 → '2'. CX = 2.

Adım 8 / 10

DEĞİŞKENLER

AX = 0E32h

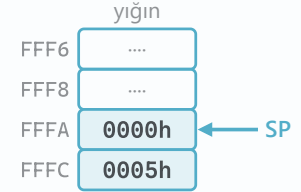
DX = 0001h

CX = 0002h

SP = FFFAh

ÇIKTI

12

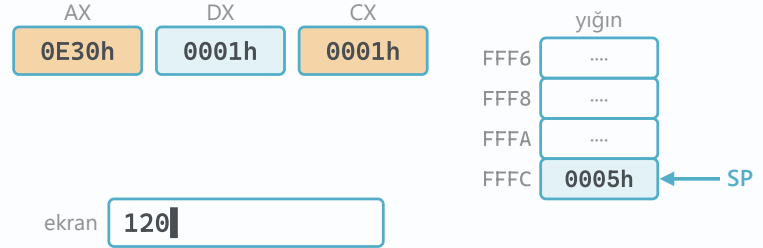


Kod İzleme: PRINT_NUM_UN\$ Mantiğı (AX = 1205)

```

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

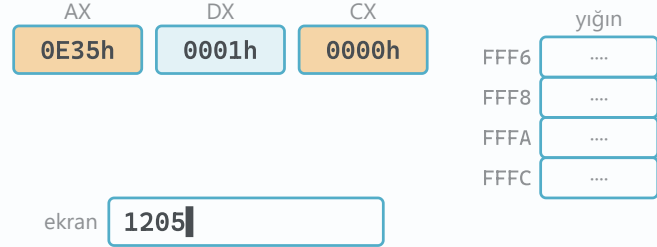
120

Kod İzleme: PRINT_NUM_UN\$ Mantiğı (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 FFF6h'ye döndü.
Ekranda 1205.

Adım 10 / 10

DEĞİŞKENLER

AX = 0E35h

DX = 0001h

CX = 0000h

SP = FFF6h

ÇIKTI

1205

PROSEDÜRLER

Demo: PRINT_NUM ve PRINT_NUM_UN

AX'e bir değer verin (onluk, negatif ya da **h** ekli onaltılık): iki prosedürün ekrana ne yazacağını görün.

AX
=

-25

-25

32767

8000h

FFFFh

AX

FFE7h

1111 1111 1110 0111

PRINT_NUM

-25

PRINT_NUM_UN

65511

Bit 15 = 1: işaretli yorumda negatif (65511 - 65536 = -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.
```

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

Adım 1 / 9

DEĞİŞKENLER

SI = msg1

CX = 0000h

AX = 0000h

SP = FFFEh

ÇAĞRI YİĞİNİ

main

ÇIKTI

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.
```

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 YİĞİNİ

print_string

main

ÇIKTI

Enter the number:

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.
```

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 YİĞİNİ

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 FFEh.

Adım 4 / 9

DEĞİŞKENLER

SI = msg1

CX = FFE7h

AX = 0000h

SP = FFEh

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.
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_UN$ ; 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 YİĞİNİ

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_UN$ ; 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

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 YİĞİNİ

pthis

main

ÇIKTI

Enter the number: -25

You have entered:

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

Adım 8 / 9

DEĞİŞKENLER

SI = msg1

CX = FFE7h

AX = FFE7h

SP = FFFCh

ÇAĞRI YİĞİNİ

print_num

main

ÇIKTI

Enter the number: -25
You have entered: -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_UN$ ; required for print_num.
16 DEFINE_PTHIS
17 END ; directive to stop the compiler.
```

RET : DOS'a dönüş. Satır 11–17 veri ve prosedür tanımlarıdır, akışta çalıştırılmaz.

Adım 9 / 9

DEĞİŞKENLER

SI = msg1

CX = FFE7h

AX = FFE7h

SP = FFFEh

ÇAĞRI YİĞİNİ

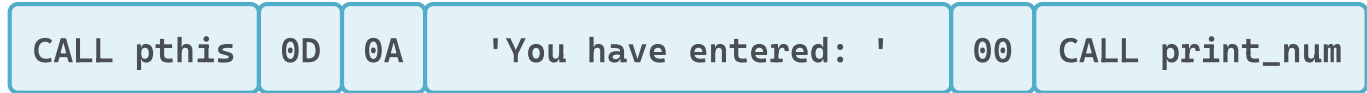
main

ÇIKTI

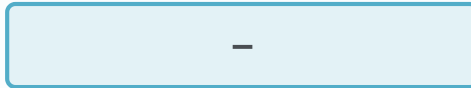
Enter the number: -25
You have entered: -25

PTHS Nasıl Çalışır?

kod kesimi (bellek, soldan sağa artan adres)



yığın:



← SP = FFFEh

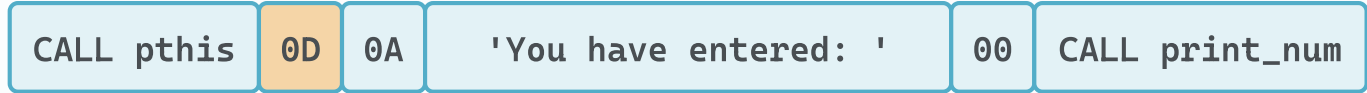
Bellekte `CALL pthis` 'in hemen ardından dizgi baytları gelir; dizgi 0 ile biter.

Adım 1 / 5

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

PTHS 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

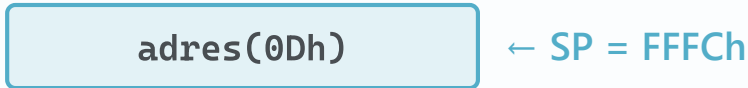
Bu teknik, dönüş adresinin yığında saklandığı bilgisine dayanır. Benzer şekilde PRINT makrosu da veriyi kod içine gömer ama **JMP** ile atlar.

PTHis Nasıl Çalışır?

kod kesimi (bellek, soldan sağa artan adres)



yığın:



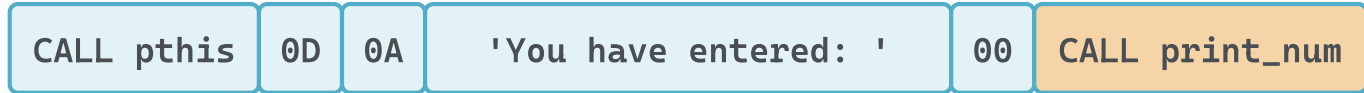
PTHis yığındaki adresi okur ve oradan itibaren baytları 0'a kadar ekrana yazar.

Adım 3 / 5

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

PTHS Nasıl Çalışır?

kod kesimi (bellek, soldan sağa artan adres)



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

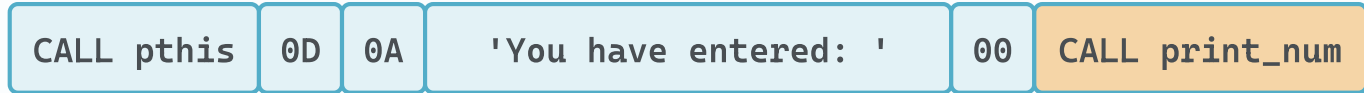
Yığındaki adres, 0 baytının bir ötesiyle değiştirilir: artık **CALL print_num** 'u gösterir.

Adım 4 / 5

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

PTHS 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.

Sık Yapılan Hatalar

DEFINE_unutmak

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

DEFINE_'yi akışa koymak

Tanımlar `ORG 100h` 'nin hemen altına konursa işlemci prosedür kodunu ana program sanarak çalıştırır. Yeri: `RET` ile `END` arası.

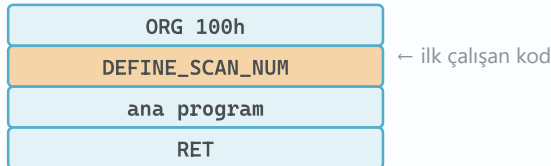
Dizgiyi 0 ile bitirmemek

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

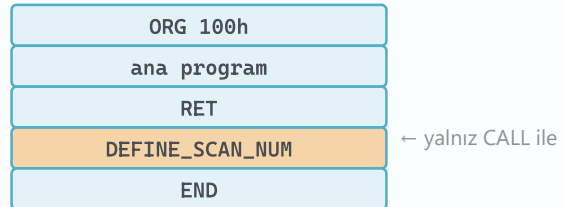
Yazmaçları varsaymak

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

Yanlış



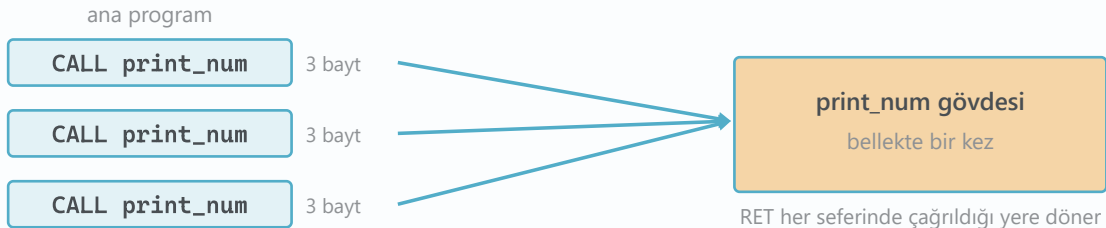
Doğru



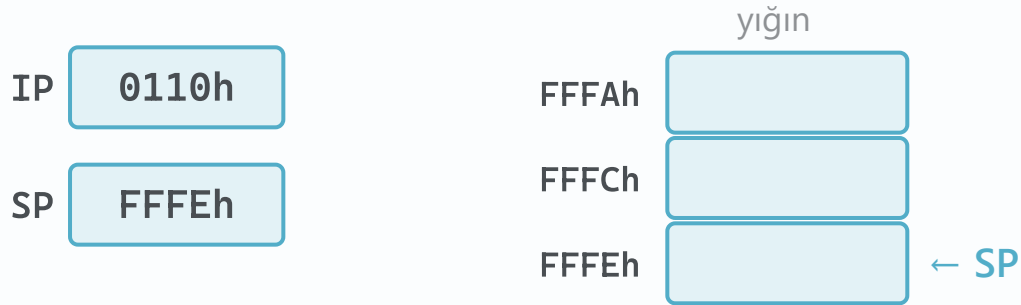
CALL KOMUTU

CALL Komutu

- Derleyici ilk aşamada `DEFINE_...` tanımlarını işler; bunlar **prosedürlere genişleyen** normal makrolardır.
- `CALL` komutu prosedürleri çağırmak için kullanılır.
- Derleyici `CALL` 'daki prosedür adını, prosedürün bulunduğu **adrese** dönüştürür.
- `CALL` yürütüldüğünde dönüş adresi yığına itilir ve kontrol prosedüre aktarılır; prosedür `RET` ile geri döner.
- Prosedür kodu dosyada **bir kez** bulunur: aynı prosedür 100 kez çağrılrsa da gövdesi çoğalmaz; her çağrı yalnızca 3 baytlık bir `CALL` ekler.
- Bu yaklaşım çalıştırılabilir dosyanın boyutunu küçük tutar.



CALL ve RET Adım Adım

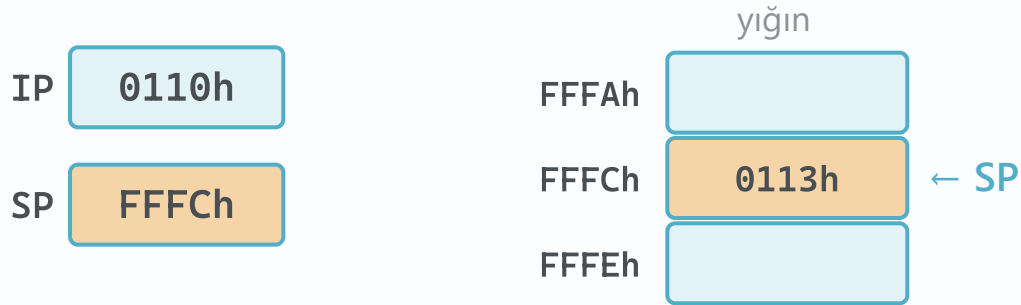


`CALL print_num` 0110h adresinde, 3 bayt (E8 + 16 bit bağıl uzaklık). Sonraki komut 0113h'de. (Örnek adresler.)

Adım 1 / 4

Prosedür içinde PUSH edilen her değer RET 'ten önce POP edilmelidir; yoksa RET yanlış adrese döner.

CALL ve RET Adım Adım

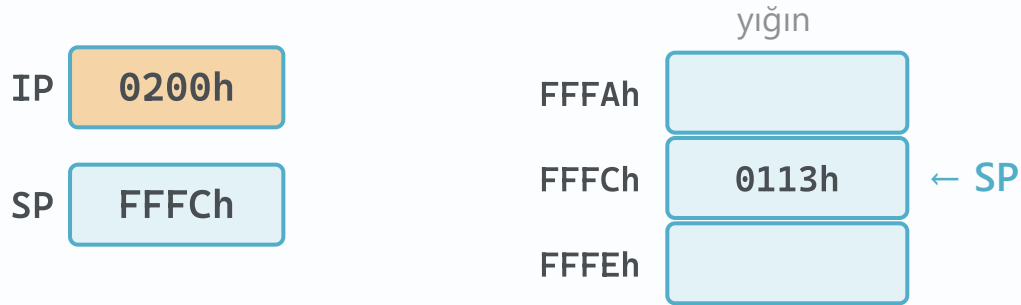


1) $SP \leftarrow SP - 2$, dönüş adresi **0113h** yığına yazılır.

Adım 2 / 4

Prosedür içinde PUSH edilen her değer RET 'ten önce POP edilmelidir; yoksa RET yanlış adrese döner.

CALL ve RET Adım Adım

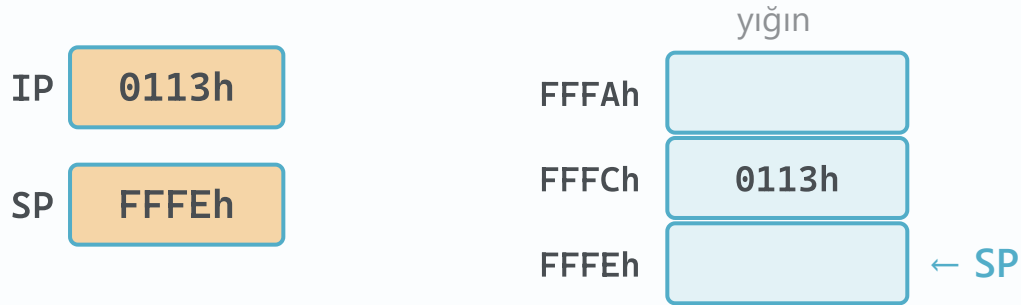


2) IP ← prosedürün adresi (ör. 0200h). Prosedür çalışır.

Adım 3 / 4

Prosedür içinde PUSH edilen her değer RET 'ten önce POP edilmelidir; yoksa RET yanlış adrese döner.

CALL ve RET Adım Adım



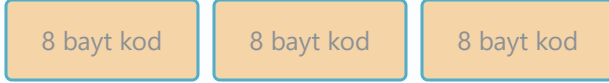
RET : $IP \leftarrow [SP] = 0113h$, $SP \leftarrow SP + 2$. Ana program kaldığı yerden sürer.

Adım 4 / 4

Prosedür içinde PUSH edilen her değer RET 'ten önce POP edilmelidir; yoksa RET yanlış adrese döner.

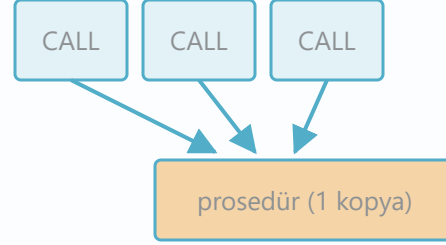
Makro mu, Prosedür mü?

3 × PUTC (makro)



toplam 24 bayt; her kullanım +8

3 × CALL (prosedür)



her kullanım +3 bayt

Ölçüt	Makro	Prosedür
Ne zaman işlenir?	derleme sırasında metin olarak açılır	çalışma sırasında CALL ile çağrılır
Kod boyutu	her kullanımda kopya	tek kopya + çağrı başına 3 bayt
Hız	CALL/RET yükü yok, biraz daha hızlı	her çağrıda yığın işlemi
Parametre	makro argümanları	yazmaçlar veya yığın
emu8086.inc	PUTC, GOTOXY, PRINT ...	PRINT_NUM, SCAN_NUM ... (DEFINE_ gerekli)

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_UNUS işaretli yazdır.

PTTHIS

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

CALL / RET

CALL dönüş adresini iter ve atlar; RET adresi çekip döner. Prosedür kodu bir kez bulunur.

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_UN` 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 5: emu8086.inc Kütüphanesi

Sonraki bölüm: Bölüm 6: Aritmetik ve Mantıksal İşlemler · [Terim sözlüğü](#)