

Bölüm 8: Prosedürler

Mikroişlemciler

Prosedür kavramı

PROC / ENDP

CALL ve RET

Yığın ve dönüş adresi

Parametre geçirme

Yığın çerçevesi

Yazmaçların korunması

Örnek programlar

Sık yapılan hatalar

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

Öğrenme Çıktıları

Tanımlama

PROC / ENDP ile prosedür tanımlamak;
direktif ile işlemci komutunu ayırt etmek.

CALL / RET

Dönüş adresinin yığına nasıl itildiğini ve
RET ile nasıl geri alındığını SP ve IP
değerleriyle izlemek.

Parametreler

Yazmaçlar, bellek değişkenleri ve yığınla
(BP çerçevesi) prosedüre veri gönderip
sonuç almak.

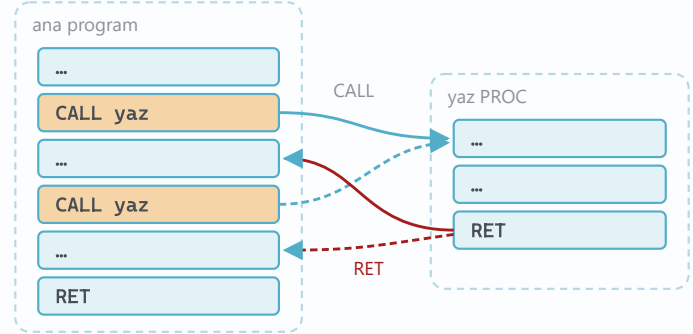
Güvenli prosedür

Yazmaçları PUSH/POP ile korumak, yığını
dengeli tutmak ve tipik hatalardan
kaçınmak.

GİRİŞ

Prosedür Nedir?

- Prosedür, **belirli bir görevi** gerçekleştiren kod parçasıdır (yüksek seviyeli dillerdeki fonksiyon / alt program).
- Programın içinden `CALL` ile çağrılır.
- Görevi tamamlayınca `RET` ile çağrıldığı noktanın **hemen arkasına** geri döner.
- Aynı kod tek kez yazılır, birçok yerden çağrılır.
- Programı daha **yapılandırılmış** (structured) ve anlaşılır hâle getirir.



Tekrar kullanım

Kod tekrarı ve bellek kullanımı azalır.

Okunabilirlik

Program, adı konmuş küçük görevlere bölünür.

Bakım

Hata tek yerde düzeltilir, ayrı ayrı test edilir.

Prosedür Tanımlama

```
ad PROC          ; başlangıç (varsayılan NEAR)
    ; prosedür kodu buraya yazılır
    RET
ad ENDP          ; bitiş
```

- `ad` : prosedürün adıdır; **başta ve sonda aynı** olmalıdır.
- Ad, etiket kurallarına uyar (rakamla başlayamaz).
- Prosedür içinde en az bir `RET` bulunmalıdır.

Prosedür nereye yazılır?

```
ORG 100h
    CALL p1
    RET          ; ana program biter
p1 PROC          ; RET'ten sonra:
    ; ...          ; sırayla yürütülmez
    RET
p1 ENDP
END
```

Prosedürler ana programın `RET` 'inden sonra konursa yanlışlıkla "düşerek" yürütülmez. Önce yazılacaksa üzerinden `JMP` ile atlanmalıdır.

TANIMLAMA

Direktifler ve Komutlar

Öge	Türü	Görevi
PROC / ENDP	Assembler direktifi	Prosedürün başlangıç adresini ve türünü (NEAR/FAR) assembler'a bildirir. Makine koduna çevrilmez , bellekte yer kaplamaz.
CALL	İşlemci komutu	Dönüş adresini yığına iter, prosedürün adresine atlar.
RET	İşlemci komutu	Dönüş adresini yığından alır (POP) ve oraya atlar. Hem prosedürden hem .COM programından dönmek için kullanılır.
END	Assembler direktifi	Kaynak dosyanın sonunu bildirir.

CALL ve RET derleyici direktifi değil, işlemcinin yürüttüğü **komutlardır** (CALL near: E8, RET: C3). Yalnızca PROC, ENDP ve END direktiftir.

derleme: PROC · ENDP · END → bayt yok

çalışma: CALL = E8 · RET = C3

RET neden işletim sistemine döner? DOS bir .COM programını başlatırken yığına 0000h iter. Programın son RET'i bu değeri IP'ye alır; CS:0000h'da (PSP'nin başı) INT 20h bulunur ve program sonlanır.

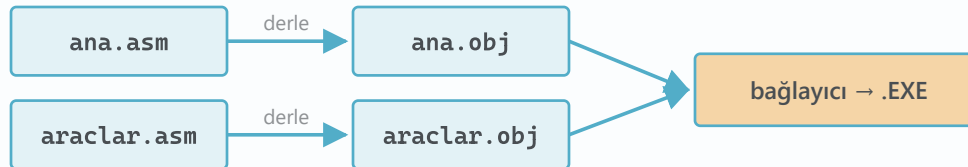
TANIMLAMA

Modüler Programlama

```
; ana.asm
EXTRN yaz:NEAR    ; başka dosyada
...
CALL yaz

; araclar.asm
PUBLIC yaz        ; dışarıya açık
yaz PROC NEAR
...
RET
yaz ENDP
```

- Büyük programlar prosedürlere, prosedürler de ayrı kaynak dosyalarına (modüllere) bölünür.
- **PUBLIC** bir prosedür adını başka modüllere açar; **EXTRN** adın başka bir modülde tanımlı olduğunu ve türünü (NEAR/FAR) bildirir.
- Her modül ayrı derlenir ve bir nesne dosyası (.OBJ) olur; **bağlayıcı** bunları birleştirir ve CALL komutlarındaki adresleri tamamlar.
- Sık kullanılan nesne dosyaları bir kütüphane (.LIB) dosyasında toplanır; bağlayıcı yalnızca gereken prosedürleri programa alır.
- emu8086'da ortak kod genellikle **INCLUDE** ile eklenir: dosyanın metni kaynağa katılır ve tek parça derlenir (emu8086.inc).



CALL VE RET

CALL ve RET Adım Adım

kod (CS)

0100h	CALL m1	← IP
0103h	MOV AX, 2	
0106h	RET	
0107h	m1: MOV BX, 5	
010Ah	RET	

Yığın (SS)

FFF8h	
FFFAh	
FFFCh	
FFFEh	0000h ← SP

yukarı: düşük adresler

Başlangıç: IP = 0100h, SP = FFFEh. Sıradaki komut **CALL m1** (3 bayt: E8 04 00).

Adım 1 / 6

CALL VE RET

CALL ve RET Adım Adım

kod (CS)

0100h	CALL m1	← IP
0103h	MOV AX, 2	
0106h	RET	
0107h	m1: MOV BX, 5	
010Ah	RET	

Yığın (SS)

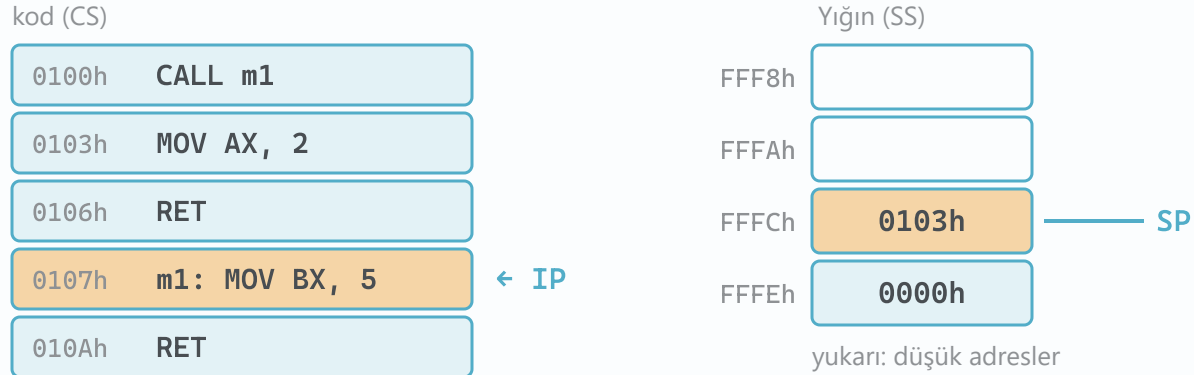
FFF8h		
FFFAh		
FFFCh	0103h	— SP
FFFEh	0000h	

yukarı: düşük adresler

CALL, önce **dönüş adresini** (sonraki komut: 0103h) yığına iter: SP = FFFCh, [FFFCh] = 0103h.

CALL VE RET

CALL ve RET Adım Adım



Sonra $IP \leftarrow 0103h + 0004h = \mathbf{0107h}$: prosedürün ilk komutu.

CALL VE RET

CALL ve RET Adım Adım

kod (CS)

0100h	CALL m1
0103h	MOV AX, 2
0106h	RET
0107h	m1: MOV BX, 5
010Ah	RET

← IP

Yığın (SS)

FFF8h	
FFFAh	
FFFCh	0103h — SP
FFFEh	0000h

yukarı: düşük adresler

MOV BX, 5 yürütüldü, IP = 010Ah. Yığın değişmedi.

CALL VE RET

CALL ve RET Adım Adım

kod (CS)

0100h	CALL m1
0103h	MOV AX, 2
0106h	RET
0107h	m1: MOV BX, 5
010Ah	RET

← IP

Yığın (SS)

FFF8h	
FFFAh	
FFFCh	
FFFEh	0000h

SP

yukarı: düşük adresler

RET: IP ← [SP] = **0103h**, SP = FFFEh. Akış, CALL'ın hemen arkasından sürer.

CALL VE RET

CALL ve RET Adım Adım

kod (CS)

0100h	CALL m1
0103h	MOV AX, 2
0106h	RET
0107h	m1: MOV BX, 5
010Ah	RET

← IP

Yığın (SS)

FFF8h	
FFFAh	
FFFCh	
FFFEh	0000h

SP

yukarı: düşük adresler

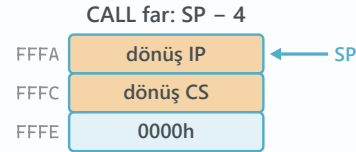
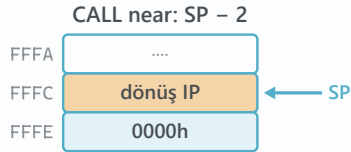
MOV AX, 2 → IP = 0106h. Buradaki RET 0000h'yı alır ve programı sonlandırır.

CALL VE RET

CALL / RET Türleri

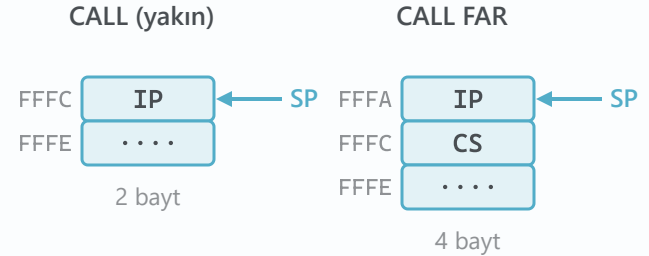
Komut	Kod	Bayt	Yığına / yığından	Açıklama
CALL near	E8 rel16	3	IP	Aynı kesimde doğrudan çağrı (varsayılan).
CALL reg/mem	FF /2	2-4	IP	Dolaylı yakın çağrı: CALL BX , CALL [tablo+SI] .
CALL far	9A ofs seg	5	CS, sonra IP	Başka kesimdeki prosedür; CS:IP birlikte değişir.
RET	C3	1	IP	Yakın dönüş.
RETF	CB	1	IP, sonra CS	Uzak dönüş (FAR prosedür).
RET n	C2 n16	3	IP, SP += n	Dönüşte yığındaki n baytlık parametreyi de atar.

ad PROC NEAR (varsayılan) ya da ad PROC FAR yazıldığında assembler, prosedür içindeki RET 'i uygun biçimde (C3 / CB) kodlar. Tek kesimli .COM örneklerinde prosedürler NEAR'dır.



Uzak Prosedür: Yığın Düzeni

- Uzak (FAR) prosedür başka bir kod kesimindedir; ona ulaşmak için hem CS hem IP değişmelidir.
- `CALL FAR` önce CS'yi, sonra IP'yi iter: yığında 4 bayt. `RET` önce IP'yi, sonra CS'yi çeker.
- Yakın `CALL` yalnızca IP'yi (2 bayt) iter, `RET` yalnızca IP'yi çeker.
- Prosedürün türü ile dönüş komutu uyuşmalıdır: `PROC FAR` yazıldığında assembler içteki RET'i RETF (CB) olarak kodlar.
- .COM programı tek kesimde çalıştığı için yakın çağrı yeterlidir; birden çok kod kesimi olan .EXE programlarında uzak çağrı gerekir.



Yığındaki sıra RETF'in çekme sırasıyla uyumludur: en son itilen IP, yığının tepesindedir ve ilk çekilen odur.

Kod İzleme: PUSH CS + CALL ile RETF

```
1  ORG 100h
2  MOV AX, 5
3  PUSH CS          ; uzak çağrının ilk yarısı
4  CALL uzak        ; IP itilir
5  ADD AX, 1
6  RET
7  uzak PROC
8  SHL AX, 1
9  RETF            ; önce IP, sonra CS
10 uzak ENDP
```

DEĞİŞKENLER

AX = 0005h

IP = 0103h

CS = 0700h

SP = FFFEh

AX	IP	CS	SP
0005h	0103h	0700h	FFFEh



AX = 5. Bu kod, CALL FAR'ın yığına yaptığı iki itmeyi iki ayrı komutla taklit eder.

Kod İzleme: PUSH CS + CALL ile RETF

```
1  ORG 100h
2  MOV AX, 5
3  ► PUSH CS          ; uzak çağrının ilk yarısı
4  CALL uzak          ; IP itilir
5  ADD AX, 1
6  RET
7  uzak PROC
8  SHL AX, 1
9  RETF                ; önce IP, sonra CS
10 uzak ENDP
```

DEĞİŞKENLER

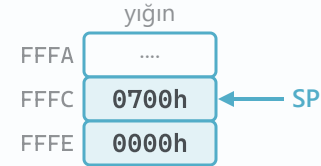
AX = 0005h

IP = 0104h

CS = 0700h

SP = FFFCh

AX	IP	CS	SP
0005h	0104h	0700h	FFFCh



PUSH CS : kod kesimi (bu çalıştırmada 0700h) yığına itilir; uzak çağrının ilk yarısı.

Kod İzleme: PUSH CS + CALL ile RETF

```
1  ORG 100h
2  MOV AX, 5
3  PUSH CS ; uzak çağrının ilk yarısı
4  CALL uzak ; IP itilir
5  ADD AX, 1
6  RET
7  uzak PROC
8  SHL AX, 1
9  RETF ; önce IP, sonra CS
10 uzak ENDP
```

DEĞİŞKENLER

AX = 0005h

IP = 010Bh

CS = 0700h

SP = FFFAh

AX	IP	CS	SP
0005h	010Bh	0700h	FFFAh

yığın	
FFFA	0107h ← SP
FFFC	0700h
FFFE	0000h

Yakın **CALL uzak** dönüş adresini (0107h) iter ve 010Bh'ye atlar. Yığın artık uzak çağrının düzenindedir: tepede IP, altında CS.

Kod İzleme: PUSH CS + CALL ile RETF

```
1  ORG 100h
2  MOV AX, 5
3  PUSH CS          ; uzak çağrının ilk yarısı
4  CALL uzak        ; IP itilir
5  ADD AX, 1
6  RET
7  uzak PROC
8  ▶ SHL AX, 1
9  RETF             ; önce IP, sonra CS
10 uzak ENDP
```

DEĞİŞKENLER

AX = 000Ah

IP = 010Dh

CS = 0700h

SP = FFFAh

AX	IP	CS	SP
000Ah	010Dh	0700h	FFFAh

	yığın	
FFFA	0107h	← SP
FFFC	0700h	
FFFE	0000h	

SHL AX, 1 : AX = 000Ah.

Kod İzleme: PUSH CS + CALL ile RETF

```
1  ORG 100h
2  MOV AX, 5
3  PUSH CS          ; uzak çağrının ilk yarısı
4  CALL uzak        ; IP itilir
5  ADD AX, 1
6  RET
7  uzak PROC
8  SHL AX, 1
9  ► RETF           ; önce IP, sonra CS
10 uzak ENDP
```

DEĞİŞKENLER

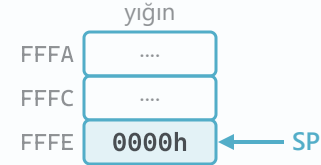
AX = 000Ah

IP = 0107h

CS = 0700h

SP = FFEh

AX	IP	CS	SP
000Ah	0107h	0700h	FFFEh



RETF iki sözcük çeker: IP = 0107h, CS = 0700h; SP 4 artarak FFEh olur.

Kod İzleme: PUSH CS + CALL ile RETF

```
1  ORG 100h
2  MOV AX, 5
3  PUSH CS          ; uzak çağrının ilk yarısı
4  CALL uzak        ; IP itilir
5  ADD AX, 1
6  RET
7  uzak PROC
8  SHL AX, 1
9  RETF            ; önce IP, sonra CS
10 uzak ENDP
```

DEĞİŞKENLER

AX = 000Bh

IP = 010Ah

CS = 0700h

SP = FFFEh

AX	IP	CS	SP
000Bh	010Ah	0700h	FFFEh



Akış `ADD AX, 1` 'den sürer: AX = 000Bh.
Gerçek CALL FAR (9A, 5 bayt) aynı iki itmeyi
tek komutla yapar.

Kod İzleme: Tablo ve Yazmaçla Dolaylı CALL

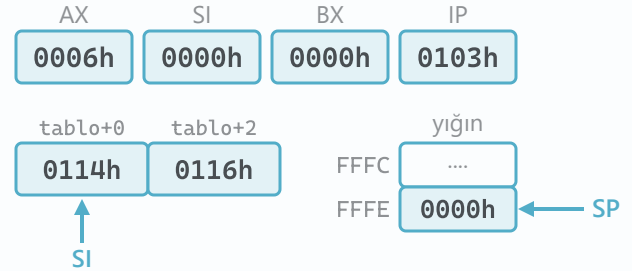
```
► MOV AX, 6
2 MOV SI, 2 ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 MOV BX, OFFSET bir_ekle
5 CALL BX ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 bir_ekle: INC AX
9 RET
10 iki_kat: SHL AX, 1
11 RET
```

DEĞİŞKENLER

AX = 6

IP = 0103h

SP = FFFEh



$\text{bir_ekle} = 0114h \cdot \text{iki_kat} = 0116h$

AX = 6. tablo, iki prosedürün adresini tutan bir sözcük dizisidir (0114h, 0116h).

Kod İzleme: Tablo ve Yazmaçla Dolaylı CALL

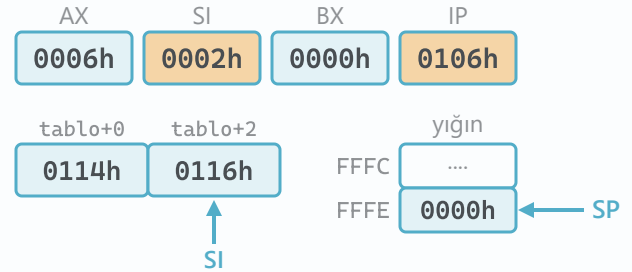
```
1 MOV AX, 6
2 MOV SI, 2 ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 MOV BX, OFFSET bir_ekle
5 CALL BX ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 bir_ekle: INC AX
9 RET
10 iki_kat: SHL AX, 1
11 RET
```

DEĞİŞKENLER

AX = 6

IP = 0106h

SP = FFFEh



$\text{bir_ekle} = 0114\text{h} \cdot \text{iki_kat} = 0116\text{h}$

SI = 2: tablodaki 2. eleman (her eleman 2 bayt).

Kod İzleme: Tablo ve Yazmaçla Dolaylı CALL

```

1 MOV AX, 6
2 MOV SI, 2      ; 2. işlem (× 2)
3 ► CALL tablo[SI] ; bellekle dolaylı
4 MOV BX, OFFSET bir_ekle
5 CALL BX        ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 bir_ekle: INC AX
9 RET
10 iki_kat: SHL AX, 1
11 RET

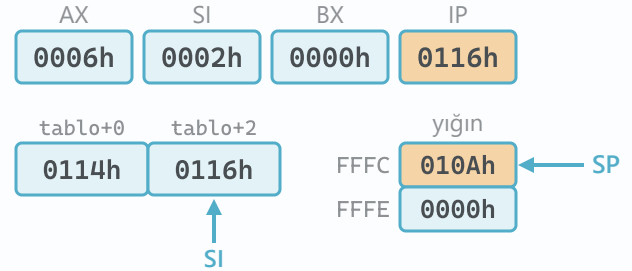
```

DEĞİŞKENLER

AX = 6

IP = 0116h

SP = FFFCh



$\text{bir_ekle} = 0114h \cdot \text{iki_kat} = 0116h$

CALL tablo[SI] : hedef adres bellekteki [tablo+2] = 0116h'den okunur. Dönüş adresi 010Ah yığına, IP = 0116h.

Kod İzleme: Tablo ve Yazmaçla Dolaylı CALL

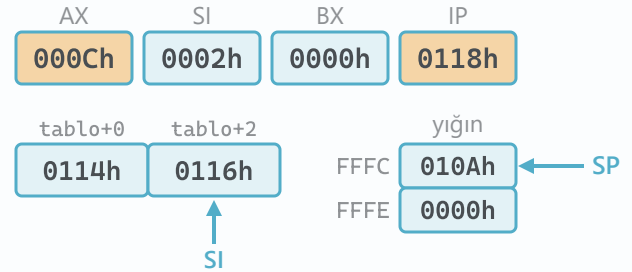
```
1 MOV AX, 6
2 MOV SI, 2 ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 MOV BX, OFFSET bir_ekle
5 CALL BX ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 bir_ekle: INC AX
9 RET
► iki_kat: SHL AX, 1
11 RET
```

DEĞİŞKENLER

AX = 12

IP = 0118h

SP = FFFCh

 $\text{bir_ekle} = 0114\text{h} \cdot \text{iki_kat} = 0116\text{h}$ $\text{iki_kat: SHL AX, 1} \rightarrow \text{AX} = 12.$

Kod İzleme: Tablo ve Yazmaçla Dolaylı CALL

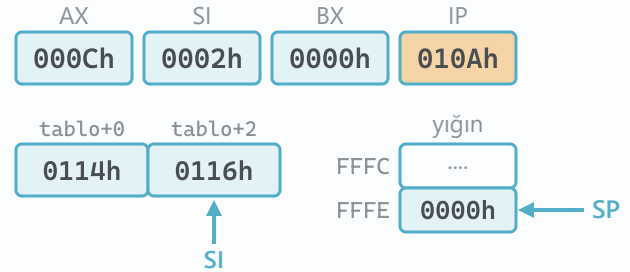
```
1 MOV AX, 6
2 MOV SI, 2 ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 MOV BX, OFFSET bir_ekle
5 CALL BX ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 bir_ekle: INC AX
9 RET
10 iki_kat: SHL AX, 1
▶ RET
```

DEĞİŞKENLER

AX = 12

IP = 010Ah

SP = FFFEh



$\text{bir_ekle} = 0114\text{h} \cdot \text{iki_kat} = 0116\text{h}$

RET : IP = 010Ah, ana programa dönüldü.

Kod İzleme: Tablo ve Yazmaçla Dolaylı CALL

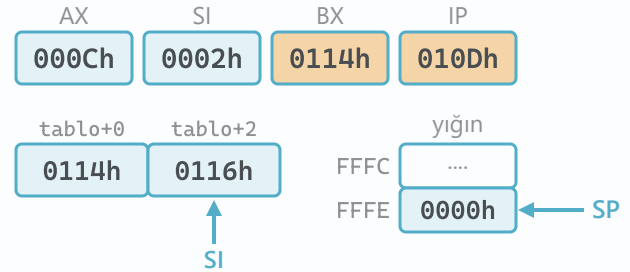
```
1 MOV AX, 6
2 MOV SI, 2 ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 ► MOV BX, OFFSET bir_ekle
5 CALL BX ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 bir_ekle: INC AX
9 RET
10 iki_kat: SHL AX, 1
11 RET
```

DEĞİŞKENLER

AX = 12

IP = 010Dh

SP = FFFEh

 $\text{bir_ekle} = 0114h \cdot \text{iki_kat} = 0116h$ $\text{BX} = \text{OFFSET bir_ekle} = 0114h.$

Kod İzleme: Tablo ve Yazmaçla Dolaylı CALL

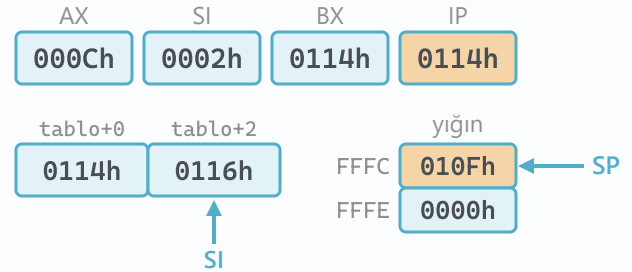
```
1 MOV AX, 6
2 MOV SI, 2 ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 MOV BX, OFFSET bir_ekle
5 ► CALL BX ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 bir_ekle: INC AX
9 RET
10 iki_kat: SHL AX, 1
11 RET
```

DEĞİŞKENLER

AX = 12

IP = 0114h

SP = FFFCh


$$\text{bir_ekle} = 0114\text{h} \cdot \text{iki_kat} = 0116\text{h}$$

CALL BX : hedef yazmaçtan alınır. Dönüş 010Fh yığına, IP = 0114h.

Kod İzleme: Tablo ve Yazmaçla Dolaylı CALL

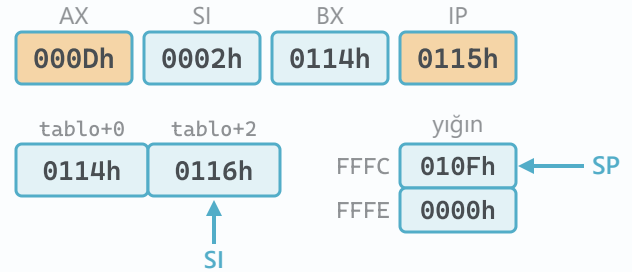
```
1 MOV AX, 6
2 MOV SI, 2 ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 MOV BX, OFFSET bir_ekle
5 CALL BX ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 ► bir_ekle: INC AX
9 RET
10 iki_kat: SHL AX, 1
11 RET
```

DEĞİŞKENLER

AX = 13

IP = 0115h

SP = FFFCh


$$\text{bir_ekle} = 0114\text{h} \cdot \text{iki_kat} = 0116\text{h}$$

bir_ekle: AX = 13.

Adım 8 / 9

Kod İzleme: Tablo ve Yazmaçla Dolaylı CALL

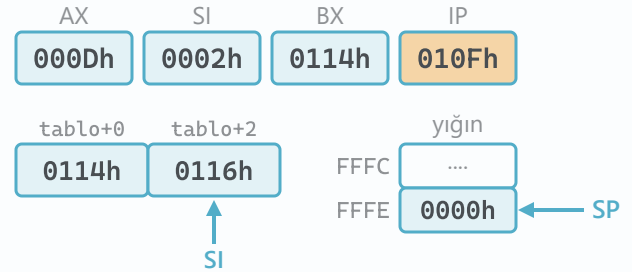
```
1 MOV AX, 6
2 MOV SI, 2      ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 MOV BX, OFFSET bir_ekle
5 CALL BX        ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 bir_ekle: INC AX
9 ► RET
10 iki_kat: SHL AX, 1
11 RET
```

DEĞİŞKENLER

AX = 13

IP = 010Fh

SP = FFFEh



$\text{bir_ekle} = 0114\text{h} \cdot \text{iki_kat} = 0116\text{h}$

RET → 010Fh. Hangi prosedürün çağrılacağı çalışma anında SI ya da BX ile seçildi.

Atlama Tablosuyla Menü

```

MOV AH, 1
INT 21h          ; AL = '1'..'3'
SUB AL, '1'      ; 0..2
CMP AL, 2
JA gecersiz     ; aralık denetimi
MOV BL, AL
MOV BH, 0
SHL BX, 1        ; her adres 2 bayt
CALL islem[BX]
gecersiz: RET
islem DW topla, cikar, carp

```

- Menüdeki her seçenek bir prosedürdür; adresleri **islem** tablosunda sırayla durur.
- Tuş, tablo indisine çevrilir: '1' → 0, '2' → 1, '3' → 2. Adresler 2 bayt olduğundan indis 2 ile çarpılır (**SHL BX, 1**).
- JA** işaretsiz karşılaştırır: '0' tuşu $0 - 1 = \text{FFh}$ verir ve 2'den büyük sayıldığı için o da reddedilir. Tek denetim iki yönü de kapatır.
- Aralık denetimi olmasaydı geçersiz bir tuş tablonun dışındaki baytları adres sanıp oraya çağrı yapardı.

Tuş	BX	Çağrılan
1	0	topla
2	2	cikar
3	4	carp
0 - 9	—	hiçbiri (RET)

Kod İzleme: Basit Prosedür

```
1 ORG      100h
▶ CALL     m1      ; 0100h: E8 04 00
3 MOV      AX, 2    ; 0103h: B8 02 00
4 RET                      ; 0106h: C3 → DOS
5 m1 PROC
6 MOV      BX, 5    ; 0107h: BB 05 00
7 RET                      ; 010Ah: C3 → çağırana
8 m1 ENDP
9 END
```

DEĞİŞKENLER

IP = 0100h

SP = FFFEh

ÇAĞRI YIĞINI

[FFFEh] 0000h

Program 0100h'dan başlar. ORG , PROC ,
ENDP , END hiç bayt üretmez.

Adım 1 / 6

Kod İzleme: Basit Prosedür

```
1 ORG      100h
2 ▶ CALL   m1          ; 0100h: E8 04 00
3 MOV      AX, 2        ; 0103h: B8 02 00
4 RET                               ; 0106h: C3 → DOS
5 m1 PROC
6 MOV      BX, 5        ; 0107h: BB 05 00
7 RET                               ; 010Ah: C3 → çağırana
8 m1 ENDP
9 END
```

DEĞİŞKENLER

IP = 0107h

SP = FFFCh

ÇAĞRI YIĞINI

[FFFCh] 0103h

[FFFEh] 0000h

CALL m1 : 0103h yığına itilir, IP = 0107h.

Kod İzleme: Basit Prosedür

```
1 ORG      100h
2 CALL     m1          ; 0100h: E8 04 00
3 MOV      AX, 2       ; 0103h: B8 02 00
4 RET                      ; 0106h: C3 → DOS
5 m1 PROC
▶ MOV      BX, 5        ; 0107h: BB 05 00
7 RET                      ; 010Ah: C3 → çağırana
8 m1 ENDP
9 END
```

DEĞİŞKENLER

IP = 010Ah

SP = FFFCh

BX = 0005h

ÇAĞRI YİĞİNİ

[FFFCh] 0103h

[FFFEh] 0000h

BX = 0005h.

Adım 3 / 6

Kod İzleme: Basit Prosedür

```
1 ORG      100h
2 CALL     m1          ; 0100h: E8 04 00
3 MOV      AX, 2        ; 0103h: B8 02 00
4 RET                      ; 0106h: C3 → DOS
5 m1 PROC
6 MOV      BX, 5        ; 0107h: BB 05 00
7 ► RET      ; 010Ah: C3 → çağırana
8 m1 ENDP
9 END
```

DEĞİŞKENLER

IP = 0103h

SP = FFEh

BX = 0005h

ÇAĞRI YİĞİNİ

[FFEh] 0000h

RET : 0103h yığından alınır → IP = 0103h,
SP = FFEh.

Adım 4 / 6

Kod İzleme: Basit Prosedür

```
1 ORG      100h
2 CALL     m1          ; 0100h: E8 04 00
3 ► MOV     AX, 2       ; 0103h: B8 02 00
4 RET                      ; 0106h: C3 → DOS
5 m1 PROC
6 MOV      BX, 5        ; 0107h: BB 05 00
7 RET                      ; 010Ah: C3 → çağırana
8 m1 ENDP
9 END
```

DEĞİŞKENLER

IP = 0106h

SP = FFFEh

AX = 0002h

BX = 0005h

ÇAĞRI YIĞINI

[FFFEh] 0000h

AX = 0002h.

Adım 5 / 6

Kod İzleme: Basit Prosedür

```
1 ORG      100h
2 CALL     m1          ; 0100h: E8 04 00
3 MOV      AX, 2       ; 0103h: B8 02 00
4 ► RET          ; 0106h: C3 → DOS
5 m1 PROC
6 MOV      BX, 5       ; 0107h: BB 05 00
7 RET          ; 010Ah: C3 → çağırana
8 m1 ENDP
9 END
```

DEĞİŞKENLER

IP = 0000h

SP = 0000h

AX = 0002h

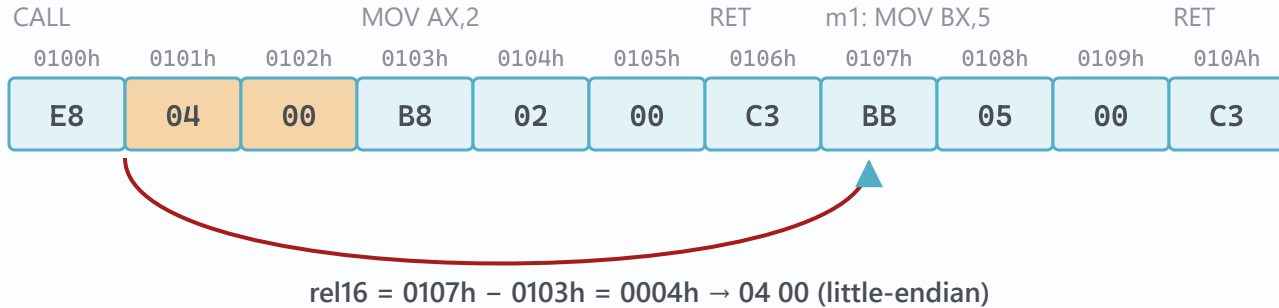
BX = 0005h

ÇAĞRI YİĞİNİ

boş

RET : 0000h alınır → IP = 0000h, SP = FFFeh + 2 = 0000h. PSP'deki INT 20h programı bitirir. Sonuç: AX = 2, BX = 5.

Bellekteki Makine Kodu



- Örnek program bellekte yalnızca **11 bayt** tutar: `E8 04 00 B8 02 00 C3 BB 05 00 C3`.
- `CALL m1` hedefi mutlak adres olarak değil, **sonraki komuta göre uzaklık** (rel16) olarak saklar. Bu yüzden kod bellekte başka bir yere taşınsa da çalışır.
- 16 bitlik değerler bellekte **little-endian** (little-endian) saklanır: önce düşük bayt (04), sonra yüksek bayt (00).
- Prosedürün adı (`m1`) yalnızca assembler için vardır; makine kodunda yalnızca 0107h adresi kalır.

CALL ve RET'in Süresi

Komut (8086)	Saat turu
CALL yakın	19
CALL yazmaç	16
CALL FAR	28
RET	8
RETF	18
PUSH yazmaç	11
POP yazmaç	8

Çağrı yükü hesabı

- İki yazmaç saklayan bir prosedürün işi dışındaki yükü: $19 + 8 + 2 \times (11 + 8) = \mathbf{65 \text{ tur}}$.
- 5 MHz'lik 8086'da bir tur 200 ns: $65 \times 200 \text{ ns} = \mathbf{13 \mu s}$.
- Prosedür 1000 kez çağrılırsa yük 13 ms olur.
- Makro bu yükü taşımaz ama her kullanımda gövdeyi kopyalar.

Kısa ve çok sık çalışan işler (tek karakter yazma) makroya, uzun ya da az çağrılan işler prosedüre uygundur. Tablodaki değerler yazmaç ve doğrudan biçimler içindir; bellek işlenenli biçimlere etkin adres hesabının süresi eklenir.

Parametre Geirme Yolları

1. Yazmalarla

En kolay ve en hızlı yol. r. AL ve BL iki parametre, sonu AX'te. Yazma sayısı sınırlıdır.

2. Bellek deėiřkenleriyle

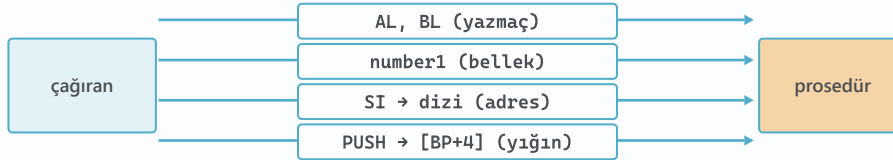
Prosedr `number1`, `result` gibi ortak deėiřkenleri okur/yazar. Kolay ama prosedr o deėiřkenlere baėımlı kalır.

3. Adres (iřareti) ile

Dizinin **adres**i SI/DI/BX'te gnderilir; uzun veriler kopyalanmadan iřlenir (`print_me`).

4. Yıėınla

Parametreler `PUSH` ile itilir, prosedr BP ile okur, `RET n` ile temizlenir. Yksek seviyeli dillerin yntemi.



Hangi yol seilirse seilsin, prosedrn bařına **girdi**, **ıktı** ve **bozulan yazmalar** aıklama olarak yazılmalıdır.

PARAMETRELER

Değerle mi, Adresle mi?

Değerle geçirme

Prosedüre verinin **kopyası** verilir ($AX = 5$).
Prosedür kopyayı değiştirse de bellekteki asıl değişken değişmez. Tek sayı gibi küçük veriler için hızlı ve güvenlidir.

Adresle geçirme

Verinin **adresi** verilir ($SI = \text{OFFSET } a$). Prosedür **[SI]** üzerinden asıl değişkeni okur ve değiştirebilir. Diziler, dizgiler ve prosedürün değiştirmesi gereken değişkenler için.

```
MOV AX, a          ; değer: AX'te a'nın kopyası
CALL iki_kat        ; AX iki katına çıkar, a aynı kalır
LEA SI, a           ; adres: SI = OFFSET a
CALL iki_kat_m      ; [SI] iki katına çıkar, a değişir
```

LEA SI, a ile MOV SI, OFFSET a aynı sonucu verir: a'nın adresi. MOV SI, a ise adresi değil, a'nın **içindeki değeri** yükler; adresle geçirmede sık yapılan bir karışıklıktır.

Kod İzleme: Adresle Takas

```
1  ORG 100h
▶ LEA SI, a
▶ LEA DI, b
4  CALL takas
5  RET
6  takas PROC          ; Girdi: SI, DI (adres)
7  PUSH AX
8  MOV AX, [SI]
9  XCHG AX, [DI]
10 MOV [SI], AX
11 POP AX
12 RET
13 takas ENDP
14 a      DW 1234h
15 b      DW 00FFh
```

DEĞİŞKENLER

SI = 0115h

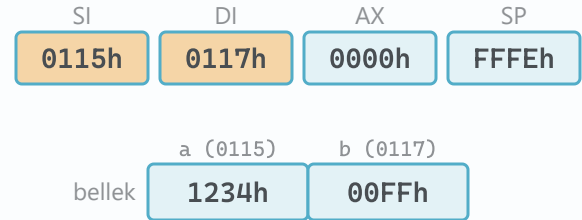
DI = 0117h

AX = 0000h

SP = FFFEh

a = 1234h

b = 00FFh



SI ve DI, a ve b değişkenlerinin adreslerini taşır (0115h, 0117h): prosedüre değerler değil, yerler gönderilir.

Kod İzleme: Adresle Takas

```
1  ORG 100h
2  LEA SI, a
3  LEA DI, b
4  ► CALL takas
5  RET
6  takas PROC ; Girdi: SI, DI (adres)
7  PUSH AX
8  MOV AX, [SI]
9  XCHG AX, [DI]
10 MOV [SI], AX
11 POP AX
12 RET
13 takas ENDP
14 a DW 1234h
15 b DW 00FFh
```

DEĞİŞKENLER

SI = 0115h

DI = 0117h

AX = 0000h

SP = FFFCh

a = 1234h

b = 00FFh

SI	DI	AX	SP
0115h	0117h	0000h	FFFCh

	a (0115)	b (0117)
bellek	1234h	00FFh

CALL takas : dönüş adresi 010Bh yığına, SP = FFFCh.

Adım 2 / 7

Kod İzleme: Adresle Takas

```
1  ORG 100h
2  LEA SI, a
3  LEA DI, b
4  CALL takas
5  RET
6  takas PROC ; Girdi: SI, DI (adres)
7
8  PUSH AX
9  MOV AX, [SI]
10 XCHG AX, [DI]
11 MOV [SI], AX
12 POP AX
13 RET
14 takas ENDP
15 a DW 1234h
16 b DW 00FFh
```

DEĞİŞKENLER

SI = 0115h

DI = 0117h

AX = 0000h

SP = FFFAh

a = 1234h

b = 00FFh

SI	DI	AX	SP
0115h	0117h	0000h	FFFAh

	a (0115)	b (0117)
bellek	1234h	00FFh

takas AX'i kullanacak: önce saklar (SP = FFFAh).

Kod İzleme: Adresle Takas

```
1  ORG 100h
2  LEA SI, a
3  LEA DI, b
4  CALL takas
5  RET
6  takas PROC ; Girdi: SI, DI (adres)
7  PUSH AX
8  ► MOV AX, [SI]
9  XCHG AX, [DI]
10 MOV [SI], AX
11 POP AX
12 RET
13 takas ENDP
14 a DW 1234h
15 b DW 00FFh
```

DEĞİŞKENLER

SI = 0115h

DI = 0117h

AX = 1234h

SP = FFFAh

a = 1234h

b = 00FFh

SI	DI	AX	SP
0115h	0117h	1234h	FFFAh

	a (0115)	b (0117)
bellek	1234h	00FFh

MOV AX, [SI] : a'nın değeri (1234h) okunur.

Adım 4 / 7

Kod İzleme: Adresle Takas

```
1  ORG 100h
2  LEA SI, a
3  LEA DI, b
4  CALL takas
5  RET
6  takas PROC ; Girdi: SI, DI (adres)
7  PUSH AX
8  MOV AX, [SI]
9  ► XCHG AX, [DI]
10 MOV [SI], AX
11 POP AX
12 RET
13 takas ENDP
14 a DW 1234h
15 b DW 00FFh
```

DEĞİŞKENLER

SI = 0115h

DI = 0117h

AX = 00FFh

SP = FFFAh

a = 1234h

b = 1234h

SI	DI	AX	SP
0115h	0117h	00FFh	FFFAh

	a (0115)	b (0117)
bellek	1234h	1234h

XCHG AX, [DI] : AX ile b tek komutta yer değiştirir: b = 1234h, AX = 00FFh.

Adım 5 / 7

Kod İzleme: Adresle Takas

```
1  ORG 100h
2  LEA SI, a
3  LEA DI, b
4  CALL takas
5  RET
6  takas PROC ; Girdi: SI, DI (adres)
7  PUSH AX
8  MOV AX, [SI]
9  XCHG AX, [DI]
10 ► MOV [SI], AX
11 POP AX
12 RET
13 takas ENDP
14 a DW 1234h
15 b DW 00FFh
```

DEĞİŞKENLER

SI = 0115h

DI = 0117h

AX = 00FFh

SP = FFFAh

a = 00FFh

b = 1234h

SI	DI	AX	SP
0115h	0117h	00FFh	FFFAh

	a (0115)	b (0117)
bellek	00FFh	1234h

MOV [SI], AX : a = 00FFh. Takas bellekte tamamlandı.

Adım 6 / 7

Kod İzleme: Adresle Takas

```
1  ORG 100h
2  LEA SI, a
3  LEA DI, b
4  CALL takas
5  RET
6  takas PROC ; Girdi: SI, DI (adres)
7  PUSH AX
8  MOV AX, [SI]
9  XCHG AX, [DI]
10 MOV [SI], AX
11 POP AX
12 RET
13 takas ENDP
14 a DW 1234h
15 b DW 00FFh
```

DEĞİŞKENLER

SI = 0115h

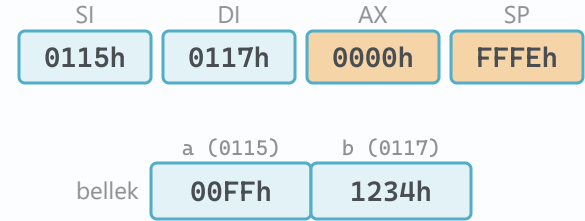
DI = 0117h

AX = 0000h

SP = FFEh

a = 00FFh

b = 1234h



POP AX ve **RET** : AX eski değerine (0000h) döner, SP = FFEh; a ile b'nin değişmiş hâli kalır.

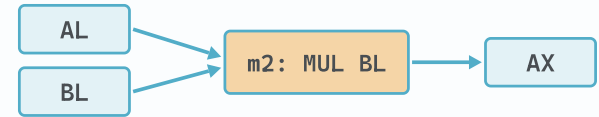
PARAMETRELER

Yazmaçlarla Parametre

- Örnekte **AL** ve **BL** yazmaçları iki parametreyi temsil eder.
- m2** prosedürü AL ve BL'yi **çarpar**, sonucu **AX** yazmacına koyar.
- CALL m2** prosedürü çağırır; **RET** çağrıyı yapan yere döner.
- m2 iki kez çağrılıyor: ikinci çağrıda AL, birinci çağrının **sonucunu** taşır, çünkü MUL AX'i (dolayısıyla AL'yi) değiştirir.

MUL kaynak	İşlem	Sonuç
8 bit	AL × kaynak	AX
16 bit	AX × kaynak	DX:AX

MUL işaretsiz çarpar; sonucun üst yarısı (AH ya da DX) 0 değilse CF = OF = 1 olur. Kaynak sabit olamaz: **MUL 2** geçersizdir.



girdi yazmaçları → prosedür → çıktı yazmacı

Kod İzleme: Yazmaçlarla Parametre

```
1 ORG      100h
2 ► MOV     AL, 1      ; 1. parametre
3 MOV      BL, 2       ; 2. parametre
4 CALL     m2          ; 0104h: E8 04 00
5 CALL     m2          ; 0107h: E8 01 00
6 RET              ; 010Ah: DOS'a dön
7 m2 PROC          ; 010Bh
8 MUL      BL          ; AX = AL * BL
9 RET              ; çağırana dön
10 m2 ENDP
11 END
```

DEĞİŞKENLER

IP = 0102h

SP = FFEh

AL = 01h

ÇAĞRI YİĞİNİ

[FFEh] 0000h

AL = 1 (MOV AL, 1 = B0 01, 2 bayt).

Adım 1 / 9

Kod İzleme: Yazmaçlarla Parametre

```
1 ORG    100h
2 MOV    AL, 1      ; 1. parametre
3 ► MOV   BL, 2      ; 2. parametre
4 CALL   m2         ; 0104h: E8 04 00
5 CALL   m2         ; 0107h: E8 01 00
6 RET     ; 010Ah: DOS'a dön
7 m2 PROC         ; 010Bh
8 MUL     BL        ; AX = AL * BL
9 RET     ; çağırana dön
10 m2 ENDP
11 END
```

DEĞİŞKENLER

IP = 0104h

SP = FFFEh

AL = 01h

BL = 02h

ÇAĞRI YIĞINI

[FFFEh] 0000h

BL = 2.

Adım 2 / 9

Kod İzleme: Yazmaçlarla Parametre

```
1 ORG    100h
2 MOV    AL, 1      ; 1. parametre
3 MOV    BL, 2      ; 2. parametre
4 ► CALL  m2        ; 0104h: E8 04 00
5 CALL   m2         ; 0107h: E8 01 00
6 RET                      ; 010Ah: DOS'a dön
7 m2 PROC                ; 010Bh
8 MUL    BL          ; AX = AL * BL
9 RET                      ; çağırana dön
10 m2 ENDP
11 END
```

DEĞİŞKENLER

IP = 010Bh

SP = FFFCh

AL = 01h

BL = 02h

ÇAĞRI YIĞINI

[FFFCh] 0107h

[FFFEh] 0000h

1. çağrı: dönüş adresi 0107h itilir, IP = 010Bh.

Adım 3 / 9

Kod İzleme: Yazmaçlarla Parametre

```
1 ORG      100h
2 MOV      AL, 1          ; 1. parametre
3 MOV      BL, 2          ; 2. parametre
4 CALL     m2             ; 0104h: E8 04 00
5 CALL     m2             ; 0107h: E8 01 00
6 RET                               ; 010Ah: DOS'a dön
7 m2 PROC                                ; 010Bh
▶ MUL      BL             ; AX = AL * BL
9 RET                               ; çağırana dön
10 m2 ENDP
11 END
```

DEĞİŞKENLER

IP = 010Dh

SP = FFFCh

AL = 02h

BL = 02h

AX = 0002h

CF = 0

ÇAĞRI YİĞİNİ

[FFFCh] 0107h

[FFFEh] 0000h

MUL BL : $AX = 1 \times 2 = 0002h$. AH = 0 → CF = OF = 0.

Kod İzleme: Yazmaçlarla Parametre

```
1 ORG      100h
2 MOV      AL, 1          ; 1. parametre
3 MOV      BL, 2          ; 2. parametre
4 CALL     m2             ; 0104h: E8 04 00
5 CALL     m2             ; 0107h: E8 01 00
6 RET                      ; 010Ah: DOS'a dön
7 m2 PROC                ; 010Bh
8 MUL      BL             ; AX = AL * BL
9 ► RET                      ; çağırana dön
10 m2 ENDP
11 END
```

DEĞİŞKENLER

IP = 0107h

SP = FFFEh

AL = 02h

BL = 02h

AX = 0002h

CF = 0

ÇAĞRI YIĞINI

[FFFEh] 0000h

RET → IP = 0107h: ikinci CALL.

Adım 5 / 9

Kod İzleme: Yazmaçlarla Parametre

```
1 ORG      100h
2 MOV      AL, 1          ; 1. parametre
3 MOV      BL, 2          ; 2. parametre
4 CALL     m2             ; 0104h: E8 04 00
▶ 5 CALL     m2             ; 0107h: E8 01 00
6 RET                               ; 010Ah: DOS'a dön
7 m2 PROC                               ; 010Bh
8 MUL      BL              ; AX = AL * BL
9 RET                               ; çağırana dön
10 m2 ENDP
11 END
```

DEĞİŞKENLER

IP = 010Bh

SP = FFFCh

AL = 02h

BL = 02h

AX = 0002h

CF = 0

ÇAĞRI YİĞİNİ

[FFFCh] 010Ah

[FFFEh] 0000h

2. çağrı: bu kez dönüş adresi **010Ah** itilir. Aynı prosedür, farklı dönüş noktası.

Kod İzleme: Yazmaçlarla Parametre

```
1 ORG      100h
2 MOV      AL, 1          ; 1. parametre
3 MOV      BL, 2          ; 2. parametre
4 CALL     m2             ; 0104h: E8 04 00
5 CALL     m2             ; 0107h: E8 01 00
6 RET                      ; 010Ah: DOS'a dön
7 m2 PROC                ; 010Bh
▶ MUL      BL             ; AX = AL * BL
9 RET                      ; çağırana dön
10 m2 ENDP
11 END
```

DEĞİŞKENLER

IP = 010Dh

SP = FFFCh

AL = 04h

BL = 02h

AX = 0004h

CF = 0

ÇAĞRI YİĞİNİ

[FFFC] 010Ah

[FFFE] 0000h

MUL BL : AL artık 2 → AX = 2 × 2 = 0004h.

Adım 7 / 9

Kod İzleme: Yazmaçlarla Parametre

```
1 ORG      100h
2 MOV      AL, 1          ; 1. parametre
3 MOV      BL, 2          ; 2. parametre
4 CALL     m2             ; 0104h: E8 04 00
5 CALL     m2             ; 0107h: E8 01 00
6 RET                      ; 010Ah: DOS'a dön
7 m2 PROC                ; 010Bh
8 MUL      BL             ; AX = AL * BL
9 ► RET                      ; çağırana dön
10 m2 ENDP
11 END
```

DEĞİŞKENLER

IP = 010Ah

SP = FFFEh

AL = 04h

BL = 02h

AX = 0004h

CF = 0

ÇAĞRI YİĞİNİ

[FFFEh] 0000h

RET → IP = 010Ah.

Adım 8 / 9

Kod İzleme: Yazmaçlarla Parametre

```
1 ORG      100h
2 MOV      AL, 1          ; 1. parametre
3 MOV      BL, 2          ; 2. parametre
4 CALL     m2             ; 0104h: E8 04 00
5 CALL     m2             ; 0107h: E8 01 00
6 ► RET                      ; 010Ah: DOS'a dön
7 m2 PROC                    ; 010Bh
8 MUL      BL             ; AX = AL * BL
9 RET                      ; çağırana dön
10 m2 ENDP
11 END
```

DEĞİŞKENLER

IP = 0000h

SP = 0000h

AL = 04h

BL = 02h

AX = 0004h

CF = 0

ÇAĞRI YİĞİNİ

boş

RET : DOS'a dönülür. Sonuç AX = 0004h.

Adım 9 / 9

Kod İzleme: Sonucu CF ile Döndürmek

```
► MOV AL, '7'
2 CALL rakam ; CF = 0 → AL = değer
3 MOV BL, AL
4 MOV AL, 'x'
5 CALL rakam ; CF = 1 → rakam değil
6 RET
7 rakam PROC ; Girdi AL, Çıktı CF
8 SUB AL, '0'
9 CMP AL, 10 ; 0..9 → ödünç, CF = 1
10 CMC ; rakamsa CF = 0
11 RET
12 rakam ENDP
```

DEĞİŞKENLER

AL = 37h

BL = 00h

CF = 0

AL

37h

BL

00h

SP

FFFEh

CF

0

AL = 55

rakam prosedürü AL'deki karakteri sayıya çevirir; başarıyı **CF** ile bildirir (DOS hizmetleri de böyle yapar). AL = '7' = 37h.

Adım 1 / 9

Kod İzleme: Sonucu CF ile Döndürmek

```
1 MOV AL, '7'
▶ CALL rakam ; CF = 0 → AL = değer
3 MOV BL, AL
4 MOV AL, 'x'
5 CALL rakam ; CF = 1 → rakam değil
6 RET
7 rakam PROC ; Girdi AL, Çıktı CF
▶ SUB AL, '0'
9 CMP AL, 10 ; 0..9 → ödünç, CF = 1
10 CMC ; rakamsa CF = 0
11 RET
12 rakam ENDP
```

DEĞİŞKENLER

AL = 07h

BL = 00h

CF = 0

AL

07h

BL

00h

SP

FFFCh

CF

0

AL = 7

CALL rakam ; SUB AL, '0' : 37h - 30h = 7.

Adım 2 / 9

Kod İzleme: Sonucu CF ile Döndürmek

```
1 MOV AL, '7'
2 CALL rakam ; CF = 0 → AL = değer
3 MOV BL, AL
4 MOV AL, 'x'
5 CALL rakam ; CF = 1 → rakam değil
6 RET
7 rakam PROC ; Girdi AL, Çıktı CF
8 SUB AL, '0'
9 ► CMP AL, 10 ; 0..9 → ödünç, CF = 1
10 CMC ; rakamsa CF = 0
11 RET
12 rakam ENDP
```

DEĞİŞKENLER

AL = 07h

BL = 00h

CF = 1

AL

07h

BL

00h

SP

FFFCh

CF

1

AL = 7

CMP AL, 10 : 7 – 10 ödünç ister → CF = 1
(değer 0...9 aralığında).

Adım 3 / 9

Kod İzleme: Sonucu CF ile Döndürmek

```
1 MOV AL, '7'
2 CALL rakam ; CF = 0 → AL = değer
3 MOV BL, AL
4 MOV AL, 'x'
5 CALL rakam ; CF = 1 → rakam değil
6 RET
7 rakam PROC ; Girdi AL, Çıktı CF
8 SUB AL, '0'
9 CMP AL, 10 ; 0..9 → ödünç, CF = 1
▶ CMC ; rakamsa CF = 0
11 RET
12 rakam ENDP
```

DEĞİŞKENLER

AL = 07h

BL = 00h

CF = 0

AL

07h

BL

00h

SP

FFFCh

CF

0

AL = 7

CMC : CF = 0 → "rakamdır".

Adım 4 / 9

Kod İzleme: Sonucu CF ile Döndürmek

```
1 MOV AL, '7'
2 CALL rakam ; CF = 0 → AL = değer
3 ► MOV BL, AL
4 MOV AL, 'x'
5 CALL rakam ; CF = 1 → rakam değil
6 RET
7 rakam PROC ; Girdi AL, Çıktı CF
8 SUB AL, '0'
9 CMP AL, 10 ; 0..9 → ödünç, CF = 1
10 CMC ; rakamsa CF = 0
11 ► RET
12 rakam ENDP
```

DEĞİŞKENLER

AL = 07h

BL = 07h

CF = 0

AL

07h

BL

07h

SP

FFFEh

CF

0

AL = 7

sonuç: rakam

RET; çağıran CF'ye bakabilir (JC/JNC). BL = 7 saklandı.

Adım 5 / 9

Kod İzleme: Sonucu CF ile Döndürmek

```
1 MOV AL, '7'
2 CALL rakam ; CF = 0 → AL = değer
3 MOV BL, AL
▶ MOV AL, 'x'
▶ CALL rakam ; CF = 1 → rakam değil
6 RET
7 rakam PROC ; Girdi AL, Çıktı CF
▶ SUB AL, '0'
9 CMP AL, 10 ; 0..9 → ödünç, CF = 1
10 CMC ; rakamsa CF = 0
11 RET
12 rakam ENDP
```

DEĞİŞKENLER

AL = 48h

BL = 07h

CF = 0

AL

48h

BL

07h

SP

FFFCh

CF

0

AL = 72

AL = 'x' = 78h; ikinci çağrı: 78h – 30h = 48h (72).

Adım 6 / 9

Kod İzleme: Sonucu CF ile Döndürmek

```
1 MOV AL, '7'
2 CALL rakam ; CF = 0 → AL = değer
3 MOV BL, AL
4 MOV AL, 'x'
5 CALL rakam ; CF = 1 → rakam değil
6 RET
7 rakam PROC ; Girdi AL, Çıktı CF
8 SUB AL, '0'
9 ► CMP AL, 10 ; 0..9 → ödünç, CF = 1
10 CMC ; rakamsa CF = 0
11 RET
12 rakam ENDP
```

DEĞİŞKENLER

AL = 48h

BL = 07h

CF = 0

AL

48h

BL

07h

SP

FFFCh

CF

0

AL = 72

CMP AL, 10 : $72 \geq 10$, ödünç yok → CF = 0.

Adım 7 / 9

Kod İzleme: Sonucu CF ile Döndürmek

```
1 MOV AL, '7'
2 CALL rakam ; CF = 0 → AL = değer
3 MOV BL, AL
4 MOV AL, 'x'
5 CALL rakam ; CF = 1 → rakam değil
6 RET
7 rakam PROC ; Girdi AL, Çıktı CF
8 SUB AL, '0'
9 CMP AL, 10 ; 0..9 → ödünç, CF = 1
▶ CMC ; rakamsa CF = 0
11 RET
12 rakam ENDP
```

DEĞİŞKENLER

AL = 48h

BL = 07h

CF = 1

AL

48h

BL

07h

SP

FFFCh

CF

1

AL = 72

CMC : CF = 1 → “rakam değil”. AL bozuldu (48h); bu durumda AL'nin değeri kullanılmamalıdır.

Adım 8 / 9

Kod İzleme: Sonucu CF ile Döndürmek

```
1 MOV AL, '7'
2 CALL rakam ; CF = 0 → AL = değer
3 MOV BL, AL
4 MOV AL, 'x'
5 CALL rakam ; CF = 1 → rakam değil
6 RET
7 rakam PROC ; Girdi AL, Çıktı CF
8 SUB AL, '0'
9 CMP AL, 10 ; 0..9 → ödünç, CF = 1
10 CMC ; rakamsa CF = 0
▶ RET
12 rakam ENDP
```

DEĞİŞKENLER

AL = 48h

BL = 07h

CF = 1

AL

48h

BL

07h

SP

FFFEh

CF

1

AL = 72

sonuç: rakam değil

RET: sonuç bayrakta. Çağıran JC hata ile ayırt edebilir.

Adım 9 / 9

PARAMETRELER

Birden Çok Sonuç Döndürmek

```
bolkalan PROC ; Girdi: AX, BL (bölen)
CMP BL, 0 ; Çıktı: AL = bölüm
JE hata ; AH = kalan
DIV BL ; CF = 1: bölen 0
CLC
RET
hata: STC
RET
bolkalan ENDP
```

- **Yazmaç çifti:** 32 bitlik sonuç DX:AX'te (16 bitlik MUL gibi), bölüm ile kalan AL ve AH'de.
- **Bayrak + yazmaç:** CF başarıyı, yazmaç değeri bildirir; çağırın önce JC ile bakar.
- **Bellek:** sonuçlar adresi verilen bir alana yazılır; sayıları sınırsızdır.

Girdi	Sonuç
AX = 100, BL = 7	AL = 0Eh (14), AH = 02h, CF = 0
AX = 100, BL = 0	CF = 1, AX değişmez (0064h)
AX = 1000, BL = 2	bölüm 500 AL'ye sığmaz: bölme hatası (INT 0)

Bu prosedürde iki RET vardır; yığına hiçbir şey itilmediği için güvenlidir. PUSH yapan prosedürlerde ise tüm yollar tek bir çıkış etiketinde birleştirilir.

Yazmaçların Korunması

Sorun

Prosedür, çağıranın kullandığı bir yazmacı habersizce değiştirirse (ör. döngü sayacı CX), çağırın kod bozulur.

Belgeleme

```
; m2: AL * BL hesaplar  
; Girdi : AL, BL (işaretsiz)  
; Çıktı : AX = AL * BL  
; Bozar : CF, OF
```

Çözüm: PUSH / POP

```
yaz PROC  
    PUSH AX      ; kullanacaklarını  
    PUSH SI      ; sakla  
    ; ... AX ve SI kullanılır  
    POP  SI      ; TERS sırayla  
    POP  AX      ; geri yükle  
    RET  
yaz ENDP
```

Sonuç döndüren yazmaç (ör. AX) korunmaz; aksi hâlde sonuç kaybolur.

Kod İzleme: PUSH CX ile Sayaç Koruma

```
► MOV CX, 2 ; ana döngü: 2 satır
2 satir: CALL yildiz
3 LOOP satir
4 RET
5 yildiz PROC
6 PUSH CX ; çağırının sayacı
7 MOV CX, 3 ; iç sayaç
8 MOV AX, 0E23h ; AH = 0Eh, AL = '#'
9 ic: INT 10h
10 LOOP ic
11 POP CX ; sayaç geri
12 RET
13 yildiz ENDP
```

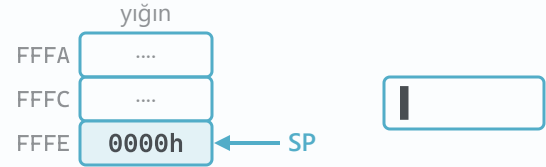
DEĞİŞKENLER

CX = 2

SP = FFFh

ÇIKTI

CX	AX	SP
0002h	0000h	FFh



Ana döngü 2 satır yazacak: sayaç CX = 2.
yildiz prosedürü de iç döngüsünde CX kullanıyor.

Kod İzleme: PUSH CX ile Sayaç Koruma

```
1 MOV CX, 2 ; ana döngü: 2 satır
▶ satir: CALL yildiz
3 LOOP satir
4 RET
5 yildiz PROC
6 PUSH CX ; çağıranın sayacı
7 MOV CX, 3 ; iç sayaç
8 MOV AX, 0E23h ; AH = 0Eh, AL = '#'
9 ic: INT 10h
10 LOOP ic
11 POP CX ; sayaç geri
12 RET
13 yildiz ENDP
```

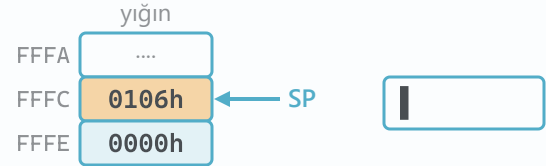
DEĞİŞKENLER

CX = 2

SP = FFFCh

ÇIKTI

CX	AX	SP
0002h	0000h	FFFCh



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

Kod İzleme: PUSH CX ile Sayaç Koruma

```
1 MOV CX, 2          ; ana döngü: 2 satır
2 satir: CALL yildiz
3 LOOP satir
4 RET
5 yildiz PROC
▶ 6 PUSH CX           ; çağıranın sayacı
7 MOV CX, 3           ; iç sayaç
8 MOV AX, 0E23h       ; AH = 0Eh, AL = '#'
9 ic: INT 10h
10 LOOP ic
11 POP CX             ; sayaç geri
12 RET
13 yildiz ENDP
```

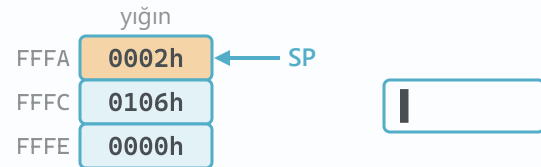
DEĞİŞKENLER

CX = 2

SP = FFFAh

ÇIKTI

CX	AX	SP
0002h	0000h	FFFAh



PUSH CX : çağıranın sayacı (2) yığına saklandı.

Kod İzleme: PUSH CX ile Sayaç Koruma

```
1 MOV CX, 2          ; ana döngü: 2 satır
2 satir: CALL yildiz
3 LOOP satir
4 RET
5 yildiz PROC
6 PUSH CX            ; çağıranın sayacı
7 MOV CX, 3          ; iç sayaç
8 MOV AX, 0E23h      ; AH = 0Eh, AL = '#'
9 ic: INT 10h
10 LOOP ic
11 POP CX            ; sayaç geri
12 RET
13 yildiz ENDP
```

DEĞİŞKENLER

CX = 3

SP = FFFAh

ÇIKTI

CX	AX	SP
0003h	0E23h	FFFAh

yığın

FFFA	0002h
FFFC	0106h
FFFE	0000h

SP →

I

İç sayaç CX = 3; AX = 0E23h (AH = 0Eh teletype, AL = '#').

Kod İzleme: PUSH CX ile Sayaç Koruma

```
1 MOV CX, 2          ; ana döngü: 2 satır
2 satir: CALL yildiz
3 LOOP satir
4 RET
5 yildiz PROC
6 PUSH CX            ; çağıranın sayacı
7 MOV CX, 3          ; iç sayaç
8 MOV AX, 0E23h      ; AH = 0Eh, AL = '#'
▶ ic: INT 10h
▶ LOOP ic
11 POP CX            ; sayaç geri
12 RET
13 yildiz ENDP
```

DEĞİŞKENLER

CX = 0

SP = FFFAh

ÇIKTI

###

CX	AX	SP
0000h	0E23h	FFFAh

yığın	
FFFA	0002h
FFFC	0106h
FFFE	0000h

###

İç döngü üç kez INT 10h : ###. LOOP sonunda CX = 0.

Kod İzleme: PUSH CX ile Sayaç Koruma

```
1 MOV CX, 2          ; ana döngü: 2 satır
2 satir: CALL yildiz
3 LOOP satir
4 RET
5 yildiz PROC
6 PUSH CX            ; çağıranın sayacı
7 MOV CX, 3          ; iç sayaç
8 MOV AX, 0E23h      ; AH = 0Eh, AL = '#'
9 ic: INT 10h
10 LOOP ic
11 ► POP CX          ; sayaç geri
12 RET
13 yildiz ENDP
```

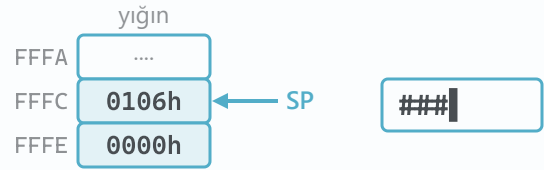
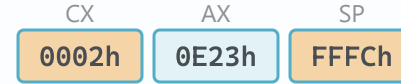
DEĞİŞKENLER

CX = 2

SP = FFFCh

ÇIKTI

###



POP CX : çağıranın sayacı geri geldi, CX = 2.

Kod İzleme: PUSH CX ile Sayaç Koruma

```
1 MOV CX, 2          ; ana döngü: 2 satır
2 satir: CALL yildiz
3 LOOP satir
4 RET
5 yildiz PROC
6 PUSH CX            ; çağıranın sayacı
7 MOV CX, 3          ; iç sayaç
8 MOV AX, 0E23h      ; AH = 0Eh, AL = '#'
9 ic: INT 10h
10 LOOP ic
11 POP CX            ; sayaç geri
12 RET
13 yildiz ENDP
```

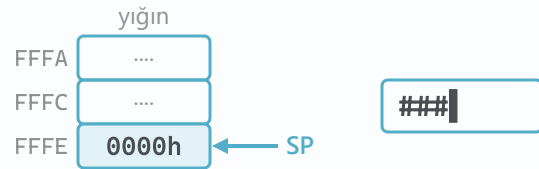
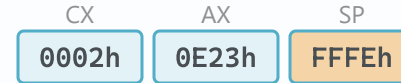
DEĞİŞKENLER

CX = 2

SP = FFFh

ÇIKTI

###



RET : 0106h'ye dönülür, SP = FFFh.

Kod İzleme: PUSH CX ile Sayaç Koruma

```

1 MOV CX, 2          ; ana döngü: 2 satır
2 satir: CALL yildiz
▶ LOOP satir
4 RET
5 yildiz PROC
6 PUSH CX            ; çağıranın sayacı
7 MOV CX, 3          ; iç sayaç
8 MOV AX, 0E23h      ; AH = 0Eh, AL = '#'
9 ic: INT 10h
10 LOOP ic
11 POP CX            ; sayaç geri
12 RET
13 yildiz ENDP

```

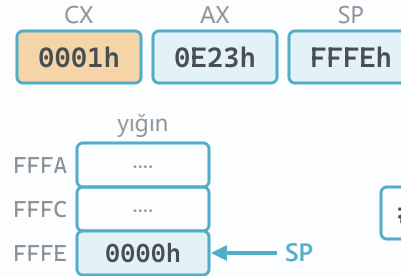
DEĞİŞKENLER

CX = 1

SP = FFFEh

ÇIKTI

###



Ana `LOOP satir` : CX = 1 → ikinci satır için yeniden çağrı.

Kod İzleme: PUSH CX ile Sayaç Koruma

```

1 MOV CX, 2          ; ana döngü: 2 satır
▶ satir: CALL yildiz
3 LOOP satir
4 RET
5 yildiz PROC
▶ PUSH CX             ; çağıranın sayacı
▶ MOV CX, 3           ; iç sayaç
▶ MOV AX, 0E23h       ; AH = 0Eh, AL = '#'
▶ ic: INT 10h
▶ LOOP ic
▶ POP CX              ; sayaç geri
▶ RET
13 yildiz ENDP

```

DEĞİŞKENLER

CX = 1

SP = FFFeh

ÇIKTI

#####

CX	AX	SP
0001h	0E23h	FFFEh

yığın

FFFA	
FFFC	
FFFE	0000h

SP →

#####

İkinci çağrı aynı şekilde: ### yazılır, CX = 1 geri yüklenir.

Kod İzleme: PUSH CX ile Sayaç Koruma

```

1 MOV CX, 2          ; ana döngü: 2 satır
2 satir: CALL yildiz
3 ▶ LOOP satir
4 RET
5 yildiz PROC
6 PUSH CX            ; çağıranın sayacı
7 MOV CX, 3          ; iç sayaç
8 MOV AX, 0E23h      ; AH = 0Eh, AL = '#'
9 ic: INT 10h
10 LOOP ic
11 POP CX            ; sayaç geri
12 RET
13 yildiz ENDP

```

DEĞİŞKENLER

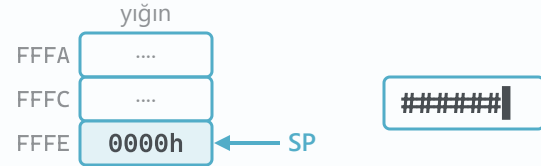
CX = 0

SP = FFFh

ÇIKTI

#####

CX	AX	SP
0000h	0E23h	FFFh



LOOP: CX = 0 → döngü biter. Ekranda 6 karakter. PUSH/POP olmasaydı dönüşte CX = 0 olur, LOOP onu FFFFh yapıp neredeyse sonsuz dönerdi.

PARAMETRELER

Çağırın mı Korur, Çağrılan mı?

Ölçüt	Çağrılan korur (prosedürde PUSH/POP)	Çağırın korur (CALL'ın çevresinde)
Kod nerede?	prosedürde bir kez	her çağrı noktasında
Gereksiz saklama	çağırın o yazmacı kullanmasa da saklanır	yalnızca gerçekten gereken yazmaçlar
Unutma riski	düşük: tek yerde yazılır	yüksek: her çağrıda tekrar gerekir
Uygun olduğu durum	kütüphane prosedürleri	çıkı yazmacındaki eski değerin gerektiği yer

```
dongu: PUSH CX          ; çağırın: sayaç
      CALL sayi_oku      ; sonuç CX'te
      ADD toplam, CX
      POP CX             ; sayaç geri
      LOOP dongu
```

Uygulamadaki kural: prosedür kullandığı yazmaçları korur, çıkı yazmaçlarını korumaz. Çıkı yazmacındaki eski değeri gerekiyorsa onu çağırın saklar.

Kod İzleme: Tek Çıkış Noktası

```

1  ORG 100h
▶ LEA SI, dizi
▶ MOV CX, 5
▶ MOV AL, 7

5  CALL bul          ; CF = 1: bulundu
6  RET
7  bul PROC          ; Girdi: SI, CX, AL
8  PUSH CX
9  PUSH SI
10 XOR BX, BX        ; Çıktı: BX = indis
11 ara: CMP [SI], AL
12 JE buldu
13 INC SI
14 INC BX
15 LOOP ara
16 CLC                ; bulunamadı
17 JMP son
18 buldu: STC
19 son: POP SI        ; tek çıkış noktası
20 POP CX
21 RET
22 bul ENDP
23 dizi DB 3, 9, 7, 1, 4

```

DEĞİŞKENLER

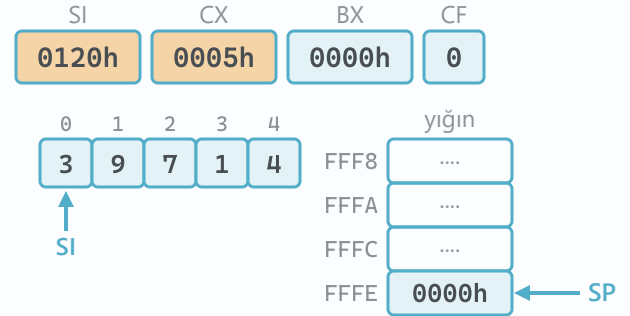
SI = 0120h

CX = 0005h

BX = 0000h

CF = 0

SP = FFFEh



bul prosedürü SI'daki diziden CX eleman içinde AL'yi (7) arar; sonucu CF ve BX ile döndürür.

Kod İzleme: Tek Çıkış Noktası

```

1  ORG 100h
2  LEA SI, dizi
3  MOV CX, 5
4  MOV AL, 7
5  ► CALL bul          ; CF = 1: bulundu
6  RET
7  bul PROC           ; Girdi: SI, CX, AL
8  PUSH CX
9  PUSH SI
10 XOR BX, BX         ; Çıktı: BX = indis
11 ara: CMP [SI], AL
12 JE buldu
13 INC SI
14 INC BX
15 LOOP ara
16 CLC               ; bulunamadı
17 JMP son
18 buldu: STC
19 son: POP SI        ; tek çıkış noktası
20 POP CX
21 RET
22 bul ENDP
23 dizi DB 3, 9, 7, 1, 4

```

DEĞİŞKENLER

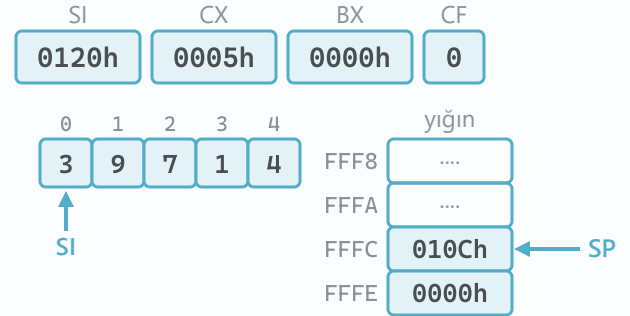
SI = 0120h

CX = 0005h

BX = 0000h

CF = 0

SP = FFFCh



CALL bul : dönüş adresi 010Ch yığına.

Adım 2 / 9

Kod İzleme: Tek Çıkış Noktası

```

1  ORG 100h
2  LEA SI, dizi
3  MOV CX, 5
4  MOV AL, 7
5  CALL bul ; CF = 1: bulundu
6  RET
7  bul PROC ; Girdi: SI, CX, AL
▶ PUSH CX
▶ PUSH SI
▶ XOR BX, BX ; Çıktı: BX = indis
11 ara: CMP [SI], AL
12 JE buldu
13 INC SI
14 INC BX
15 LOOP ara
16 CLC ; bulunamadı
17 JMP son
18 buldu: STC
19 son: POP SI ; tek çıkış noktası
20 POP CX
21 RET
22 bul ENDP
23 dizi DB 3, 9, 7, 1, 4

```

DEĞİŞKENLER

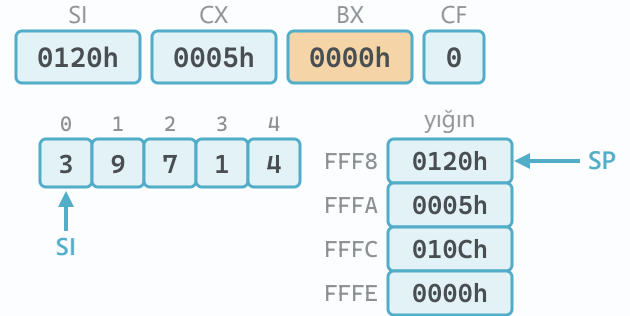
SI = 0120h

CX = 0005h

BX = 0000h

CF = 0

SP = FFF8h



Prosedür CX ve SI'yi değiştireceği için ikisini de saklar; SP = FFF8h. BX = 0 (indis).

Kod İzleme: Tek Çıkış Noktası

```

1  ORG 100h
2  LEA SI, dizi
3  MOV CX, 5
4  MOV AL, 7
5  CALL bul ; CF = 1: bulundu
6  RET
7  bul PROC ; Girdi: SI, CX, AL
8  PUSH CX
9  PUSH SI
10 XOR BX, BX ; Çıktı: BX = indis
    ► ara: CMP [SI], AL
    ► JE buldu
    ► INC SI
    ► INC BX
    ► LOOP ara
16 CLC ; bulunamadı
17 JMP son
18 buldu: STC
19 son: POP SI ; tek çıkış noktası
20 POP CX
21 RET
22 bul ENDP
23 dizi DB 3, 9, 7, 1, 4

```

DEĞİŞKENLER

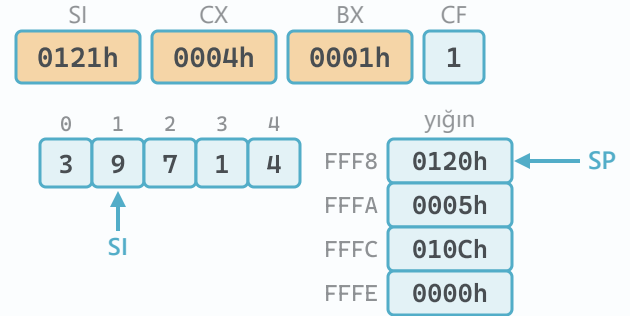
SI = 0121h

CX = 0004h

BX = 0001h

CF = 1

SP = FFF8h



dizi[0] = 3 ≠ 7: SI ve BX ilerler, LOOP CX'i 4'e indirir.

Adım 4 / 9

Kod İzleme: Tek Çıkış Noktası

```

1  ORG 100h
2  LEA SI, dizi
3  MOV CX, 5
4  MOV AL, 7
5  CALL bul          ; CF = 1: bulundu
6  RET
7  bul PROC          ; Girdi: SI, CX, AL
8  PUSH CX
9  PUSH SI
10 XOR BX, BX        ; Çıktı: BX = indis
    ► ara: CMP [SI], AL
    ► JE buldu
    ► INC SI
    ► INC BX
    ► LOOP ara
16 CLC                ; bulunamadı
17 JMP son
18 buldu: STC
19 son: POP SI        ; tek çıkış noktası
20 POP CX
21 RET
22 bul ENDP
23 dizi DB 3, 9, 7, 1, 4

```

DEĞİŞKENLER

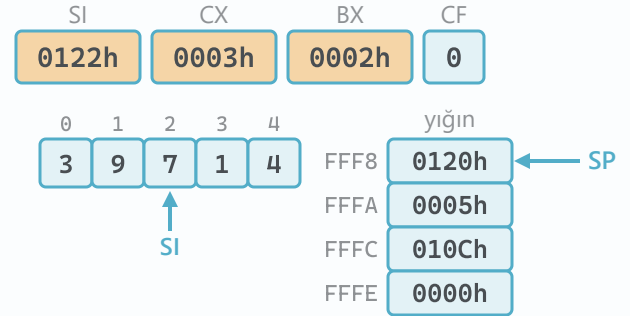
SI = 0122h

CX = 0003h

BX = 0002h

CF = 0

SP = FFF8h



dizi[1] = 9 ≠ 7: BX = 2, CX = 3.

Adım 5 / 9

Kod İzleme: Tek Çıkış Noktası

```

1  ORG 100h
2  LEA SI, dizi
3  MOV CX, 5
4  MOV AL, 7
5  CALL bul          ; CF = 1: bulundu
6  RET
7  bul PROC          ; Girdi: SI, CX, AL
8  PUSH CX
9  PUSH SI
10 XOR BX, BX        ; Çıktı: BX = indis
    ► ara: CMP [SI], AL
    ► JE buldu
13 INC SI
14 INC BX
15 LOOP ara
16 CLC                ; bulunamadı
17 JMP son
18 buldu: STC
19 son: POP SI        ; tek çıkış noktası
20 POP CX
21 RET
22 bul ENDP
23 dizi DB 3, 9, 7, 1, 4

```

DEĞİŞKENLER

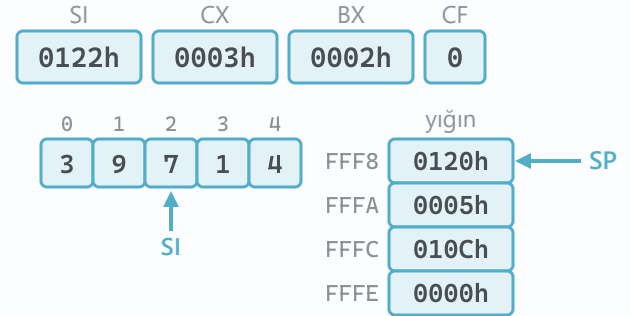
SI = 0122h

CX = 0003h

BX = 0002h

CF = 0

SP = FFF8h



dizi[2] = 7: ZF = 1, **JE buldu** atlar. Döngü erken bitiyor ama yığında hâlâ iki saklı değer var.

Kod İzleme: Tek Çıkış Noktası

```

1  ORG 100h
2  LEA SI, dizi
3  MOV CX, 5
4  MOV AL, 7
5  CALL bul      ; CF = 1: bulundu
6  RET
7  bul PROC      ; Girdi: SI, CX, AL
8  PUSH CX
9  PUSH SI
10 XOR BX, BX    ; Çıktı: BX = indis
11 ara: CMP [SI], AL
12 JE buldu
13 INC SI
14 INC BX
15 LOOP ara
16 CLC          ; bulunamadı
17 JMP son
18 ► buldu: STC
19 son: POP SI   ; tek çıkış noktası
20 POP CX
21 RET
22 bul ENDP
23 dizi DB 3, 9, 7, 1, 4

```

DEĞİŞKENLER

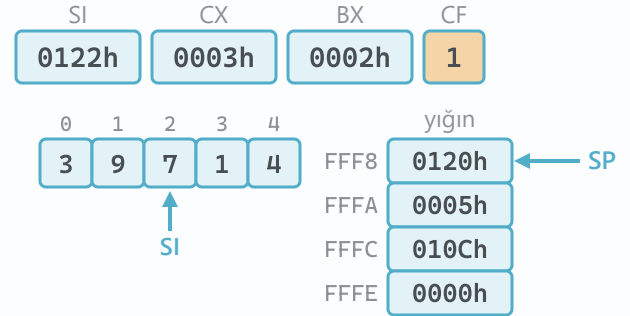
SI = 0122h

CX = 0003h

BX = 0002h

CF = 1

SP = FFF8h



STC : CF = 1 → "bulundu". Bulunamasaaydı CLC ve JMP son aynı yere gelirdi.

Kod İzleme: Tek Çıkış Noktası

```

1  ORG 100h
2  LEA SI, dizi
3  MOV CX, 5
4  MOV AL, 7
5  CALL bul           ; CF = 1: bulundu
6  RET
7  bul PROC           ; Girdi: SI, CX, AL
8  PUSH CX
9  PUSH SI
10 XOR BX, BX         ; Çıktı: BX = indis
11 ara: CMP [SI], AL
12 JE buldu
13 INC SI
14 INC BX
15 LOOP ara
16 CLC               ; bulunamadı
17 JMP son
18 buldu: STC
19 ► son: POP SI      ; tek çıkış noktası
20 ► POP CX
21 RET
22 bul ENDP
23 dizi DB 3, 9, 7, 1, 4

```

DEĞİŞKENLER

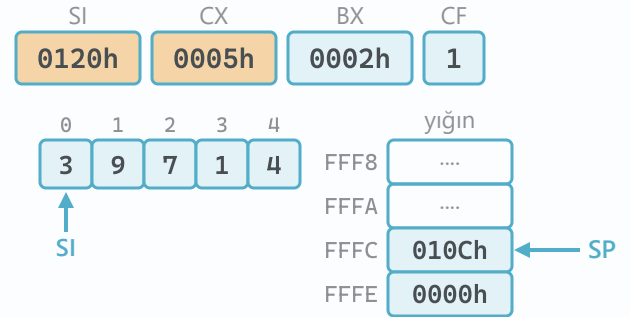
SI = 0120h

CX = 0005h

BX = 0002h

CF = 1

SP = FFFCh



Her iki yol da **son** etiketinde birleşir: POP'lar her durumda çalışır, SI ve CX eski değerlerine döner.

Kod İzleme: Tek Çıkış Noktası

```

1  ORG 100h
2  LEA SI, dizi
3  MOV CX, 5
4  MOV AL, 7
5  CALL bul           ; CF = 1: bulundu
6  RET
7  bul PROC           ; Girdi: SI, CX, AL
8  PUSH CX
9  PUSH SI
10 XOR BX, BX         ; Çıktı: BX = indis
11 ara: CMP [SI], AL
12 JE buldu
13 INC SI
14 INC BX
15 LOOP ara
16 CLC               ; bulunamadı
17 JMP son
18 buldu: STC
19 son: POP SI        ; tek çıkış noktası
20 POP CX
21 RET
22 bul ENDP
23 dizi DB 3, 9, 7, 1, 4

```

DEĞİŞKENLER

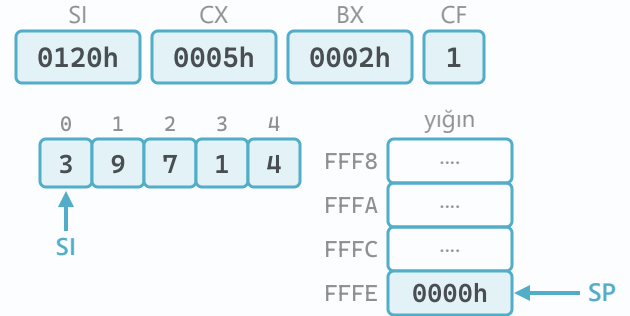
SI = 0120h

CX = 0005h

BX = 0002h

CF = 1

SP = FFFEh



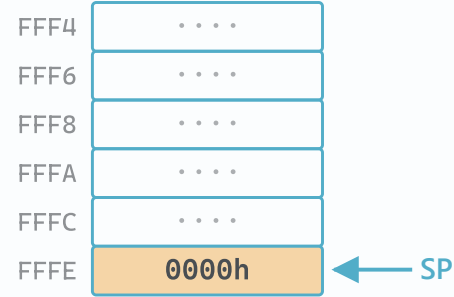
RET doğru adrese (010Ch) döner: CF = 1, BX = 2. JE'nin yerinde doğrudan RET olsaydı tepedeki SI değeri (0120h) dönüş adresi sanılırdı.

Yığınla Parametre Geçirme (BP ile)

```

▶ ORG 100h
2 MOV AX, 7
3 PUSH AX          ; 1. parametre
4 MOV AX, 3
5 PUSH AX          ; 2. parametre
6 CALL TOPLA
7 ADD SP, 4        ; parametreleri at
8 RET
9 TOPLA PROC
10 PUSH BP
11 MOV BP, SP
12 MOV AX, [BP+6]  ; 1. parametre
13 ADD AX, [BP+4]  ; 2. parametre
14 POP BP
15 RET
16 TOPLA ENDP

```



SS:bağlı konum · küçük adres yukarıda

Çağırın, parametreleri yığına koyar; prosedür bunları BP üzerinden okur.

Adım 1 / 14

DEĞİŞKENLER

AX = 0000h

BP = 0000h

SP = FFFEh

IP = 0100h

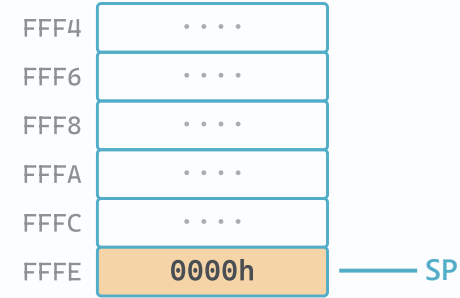
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

1  ORG  100h
▶  MOV  AX, 7
3  PUSH AX          ; 1. parametre
4  MOV  AX, 3
5  PUSH AX          ; 2. parametre
6  CALL TOPLA
7  ADD  SP, 4        ; parametreleri at
8  RET
9  TOPLA PROC
10 PUSH  BP
11 MOV  BP, SP
12 MOV  AX, [BP+6]   ; 1. parametre
13 ADD  AX, [BP+4]   ; 2. parametre
14 POP  BP
15 RET
16 TOPLA ENDP

```



SS:bağlı konum · küçük adres yukarıda

AX = 7.

Adım 2 / 14

DEĞİŞKENLER

AX = 0007h

BP = 0000h

SP = FFEh

IP = 0103h

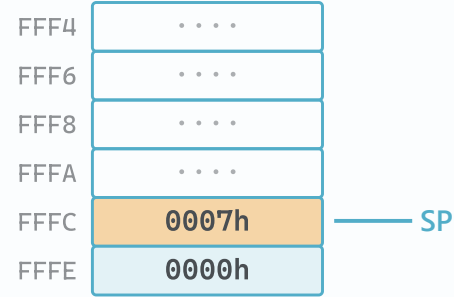
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

1  ORG  100h
2  MOV  AX, 7
▶ 3  PUSH AX          ; 1. parametre
4  MOV  AX, 3
5  PUSH AX          ; 2. parametre
6  CALL TOPLA
7  ADD  SP, 4        ; parametreleri at
8  RET
9  TOPLA PROC
10 PUSH  BP
11 MOV  BP, SP
12 MOV  AX, [BP+6]   ; 1. parametre
13 ADD  AX, [BP+4]   ; 2. parametre
14 POP  BP
15 RET
16 TOPLA ENDP

```



SS:bağlı konum · küçük adres yukarıda

1. parametre yığında: [FFFC] = 0007h.

Adım 3 / 14

DEĞİŞKENLER

AX = 0007h

BP = 0000h

SP = FFFCh

IP = 0104h

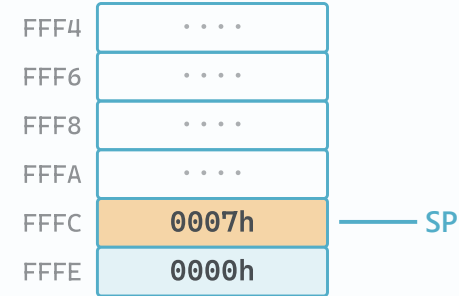
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

1  ORG  100h
2  MOV  AX, 7
3  PUSH AX          ; 1. parametre
4  ► MOV  AX, 3
5  PUSH AX          ; 2. parametre
6  CALL TOPLA
7  ADD  SP, 4        ; parametreleri at
8  RET
9  TOPLA PROC
10 PUSH  BP
11 MOV  BP, SP
12 MOV  AX, [BP+6]   ; 1. parametre
13 ADD  AX, [BP+4]   ; 2. parametre
14 POP  BP
15 RET
16 TOPLA ENDP

```



SS:bağlı konum · küçük adres yukarıda

AX = 3.

Adım 4 / 14

DEĞİŞKENLER

AX = 0003h

BP = 0000h

SP = FFFCh

IP = 0107h

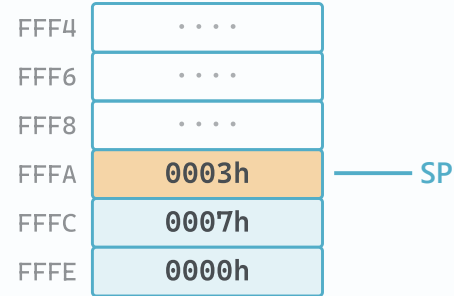
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

1  ORG  100h
2  MOV  AX, 7
3  PUSH AX           ; 1. parametre
4  MOV  AX, 3
5  ► PUSH AX         ; 2. parametre
6  CALL TOPLA
7  ADD  SP, 4         ; parametreleri at
8  RET
9  TOPLA PROC
10 PUSH  BP
11 MOV  BP, SP
12 MOV  AX, [BP+6]    ; 1. parametre
13 ADD  AX, [BP+4]    ; 2. parametre
14 POP  BP
15 RET
16 TOPLA ENDP

```



SS:bağlı konum · küçük adres yukarıda

2. parametre: [FFFAh] = 0003h.

Adım 5 / 14

DEĞİŞKENLER

AX = 0003h

BP = 0000h

SP = FFFAh

IP = 0108h

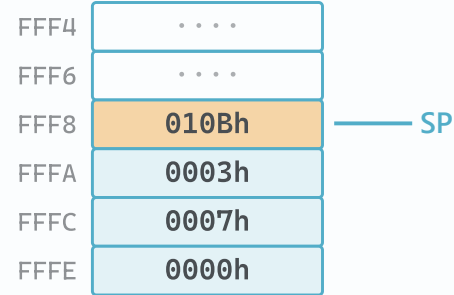
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

1  ORG 100h
2  MOV AX, 7
3  PUSH AX          ; 1. parametre
4  MOV AX, 3
5  PUSH AX          ; 2. parametre
6  ► CALL TOPLA
7  ADD SP, 4        ; parametreleri at
8  RET
9  TOPLA PROC
10 PUSH BP
11 MOV BP, SP
12 MOV AX, [BP+6]   ; 1. parametre
13 ADD AX, [BP+4]   ; 2. parametre
14 POP BP
15 RET
16 TOPLA ENDP

```



SS:bağlı konum · küçük adres yukarıda

CALL: dönüş adresi **010Bh** → [FFF8h]. IP = 010Fh.

Adım 6 / 14

DEĞİŞKENLER

AX = 0003h

BP = 0000h

SP = FFF8h

IP = 010Fh

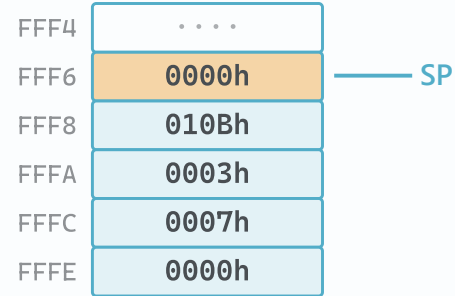
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

1  ORG  100h
2  MOV  AX, 7
3  PUSH AX           ; 1. parametre
4  MOV  AX, 3
5  PUSH AX           ; 2. parametre
6  CALL TOPLA
7  ADD  SP, 4         ; parametreleri at
8  RET
9  TOPLA PROC
▶  PUSH BP
11 MOV  BP, SP
12 MOV  AX, [BP+6]    ; 1. parametre
13 ADD  AX, [BP+4]    ; 2. parametre
14 POP  BP
15 RET
16 TOPLA ENDP

```



SS:bağlı konum · küçük adres yukarıda

Eski BP saklanır: [FFF6h] = 0000h.

Adım 7 / 14

DEĞİŞKENLER

AX = 0003h

BP = 0000h

SP = FFF6h

IP = 0110h

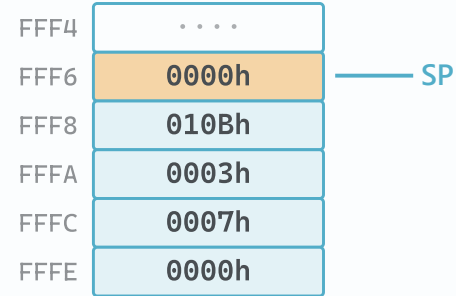
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

1  ORG 100h
2  MOV AX, 7
3  PUSH AX          ; 1. parametre
4  MOV AX, 3
5  PUSH AX          ; 2. parametre
6  CALL TOPLA
7  ADD SP, 4        ; parametreleri at
8  RET
9  TOPLA PROC
10 PUSH BP
11 MOV BP, SP
12 MOV AX, [BP+6]   ; 1. parametre
13 ADD AX, [BP+4]   ; 2. parametre
14 POP BP
15 RET
16 TOPLA ENDP

```



SS:bağlı konum · küçük adres yukarıda

MOV BP, SP : BP = **FFF6h** sabit bir çerçeve noktası olur.

Adım 8 / 14

DEĞİŞKENLER

AX = 0003h

BP = **FFF6h**

SP = **FFF6h**

IP = 0112h

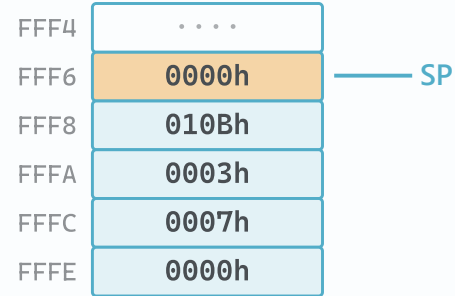
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

1  ORG 100h
2  MOV AX, 7
3  PUSH AX          ; 1. parametre
4  MOV AX, 3
5  PUSH AX          ; 2. parametre
6  CALL TOPLA
7  ADD SP, 4        ; parametreleri at
8  RET
9  TOPLA PROC
10 PUSH BP
11 MOV BP, SP
12 ► MOV AX, [BP+6] ; 1. parametre
13 ADD AX, [BP+4]   ; 2. parametre
14 POP BP
15 RET
16 TOPLA ENDP

```



SS:bağlı konum · küçük adres yukarıda

$[BP+6] = [FFCCh] = 0007h \rightarrow AX = 7.$

Adım 9 / 14

DEĞİŞKENLER

AX = 0007h

BP = FFF6h

SP = FFF6h

IP = 0115h

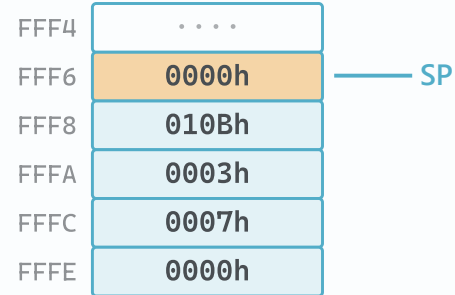
$[BP+0]$ = eski BP, $[BP+2]$ = dönüş adresi, $[BP+4]$ = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

1  ORG 100h
2  MOV AX, 7
3  PUSH AX          ; 1. parametre
4  MOV AX, 3
5  PUSH AX          ; 2. parametre
6  CALL TOPLA
7  ADD SP, 4        ; parametreleri at
8  RET
9  TOPLA PROC
10 PUSH BP
11 MOV BP, SP
12 MOV AX, [BP+6]   ; 1. parametre
13 ► ADD AX, [BP+4] ; 2. parametre
14 POP BP
15 RET
16 TOPLA ENDP

```



SS:bağlı konum · küçük adres yukarıda

$[BP+4] = [FFFAh] = 0003h \rightarrow AX = 000Ah$.

Adım 10 / 14

DEĞİŞKENLER

AX = 000Ah

BP = FFF6h

SP = FFF6h

IP = 0118h

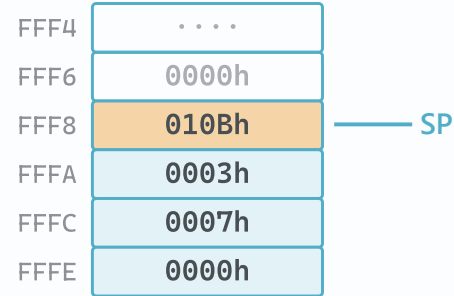
$[BP+0]$ = eski BP, $[BP+2]$ = dönüş adresi, $[BP+4]$ = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

1  ORG  100h
2  MOV  AX, 7
3  PUSH AX           ; 1. parametre
4  MOV  AX, 3
5  PUSH AX           ; 2. parametre
6  CALL TOPLA
7  ADD  SP, 4         ; parametreleri at
8  RET
9  TOPLA PROC
10 PUSH  BP
11 MOV  BP, SP
12 MOV  AX, [BP+6]    ; 1. parametre
13 ADD  AX, [BP+4]    ; 2. parametre
14 ► POP  BP
15 RET
16 TOPLA ENDP

```



SS:bağlı konum · küçük adres yukarıda
BP eski değerine döner; SP = FFF8h (dönüş adresi üstte).

Adım 11 / 14

DEĞİŞKENLER

AX = 000Ah

BP = 0000h

SP = FFF8h

IP = 0119h

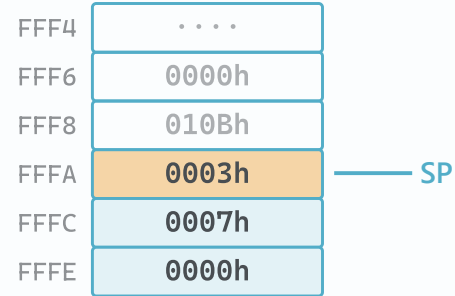
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

1  ORG 100h
2  MOV AX, 7
3  PUSH AX          ; 1. parametre
4  MOV AX, 3
5  PUSH AX          ; 2. parametre
6  CALL TOPLA
7  ADD SP, 4        ; parametreleri at
8  RET
9  TOPLA PROC
10 PUSH BP
11 MOV BP, SP
12 MOV AX, [BP+6]   ; 1. parametre
13 ADD AX, [BP+4]   ; 2. parametre
14 POP BP
15 RET
16 TOPLA ENDP

```



SS:bağlı konum · küçük adres yukarıda

RET: IP = 010Bh, SP = FFFAh.

Adım 12 / 14

DEĞİŞKENLER

AX = 000Ah

BP = 0000h

SP = FFFAh

IP = 010Bh

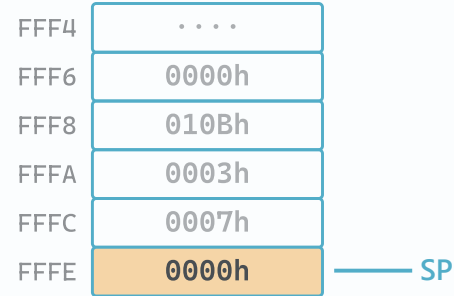
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

1  ORG 100h
2  MOV AX, 7
3  PUSH AX          ; 1. parametre
4  MOV AX, 3
5  PUSH AX          ; 2. parametre
6  CALL TOPLA
7  ► ADD SP, 4      ; parametreleri at
8  RET
9  TOPLA PROC
10 PUSH BP
11 MOV BP, SP
12 MOV AX, [BP+6]   ; 1. parametre
13 ADD AX, [BP+4]   ; 2. parametre
14 POP BP
15 RET
16 TOPLA ENDP

```



SS:bağlı konum · küçük adres yukarıda

ADD SP, 4 : iki parametre atılır, SP = FFEh. Yığın dengelendi.

Adım 13 / 14

DEĞİŞKENLER

AX = 000Ah

BP = 0000h

SP = FFEh

IP = 010Eh

[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

1  ORG 100h
2  MOV AX, 7
3  PUSH AX          ; 1. parametre
4  MOV AX, 3
5  PUSH AX          ; 2. parametre
6  CALL TOPLA
7  ADD SP, 4        ; parametreleri at
▶ RET
9  TOPLA PROC
10 PUSH BP
11 MOV BP, SP
12 MOV AX, [BP+6]   ; 1. parametre
13 ADD AX, [BP+4]   ; 2. parametre
14 POP BP
15 RET
16 TOPLA ENDP

```

FFF4	
FFF6	0000h
FFF8	010Bh
FFFA	0003h
FFFC	0007h
FFFE	0000h

SS:bağlı konum · küçük adres yukarıda

Program biter.

Adım 14 / 14

DEĞİŞKENLER

AX = 000Ah

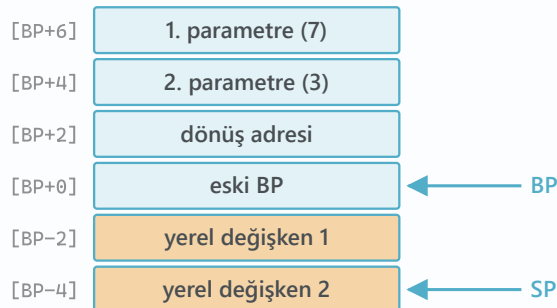
BP = 0000h

SP = 0000h

IP = 0000h

[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığın Çerçevesi ve Yerel Değişkenler



küçük adres yukarıda · büyük adres aşağıda

```
PUSH BP
MOV BP, SP
SUB SP, 4      ; yerel alan
MOV [BP-2], AX ; yerel 1
...
MOV SP, BP     ; yerelleri at
POP BP
RET 4          ; 2 parametreyi at
```

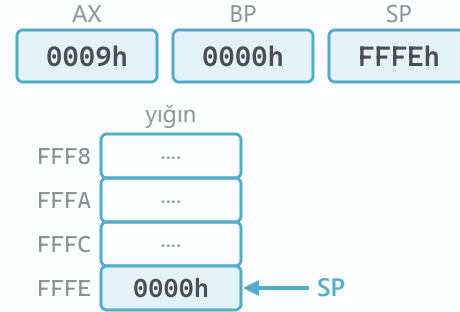
- `SUB SP, 4` yığında iki sözcüklük **yerel alan** açar: [BP-2], [BP-4].
- `MOV SP, BP` yerel alanı tek komutla geri verir.
- Parametreler pozitif, yereller negatif BP bağlı konumundadır.

Kod İzleme: Yığında Yerel Değişken

```

► MOV AX, 9
2 CALL kare_art ; AX = x·x + x
3 RET
4 kare_art PROC
5 PUSH BP
6 MOV BP, SP
7 SUB SP, 2 ; yerel alan [BP-2]
8 MOV [BP-2], AX ; yerel = x
9 MUL AX ; AX = x·x
10 ADD AX, [BP-2] ; + x
11 MOV SP, BP ; yereli at
12 POP BP
13 RET
14 kare_art ENDP

```



Parametre $x = 9$ AX'te. Prosedür x 'i bir yerel değişkende saklayıp $x \cdot x + x$ hesaplayacak.

Adım 1 / 10

DEĞİŞKENLER

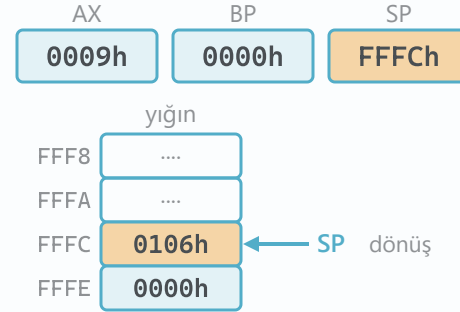
AX = 9

BP = 0000h

SP = FFEh

Kod İzleme: Yığında Yerel Değişken

```
1 MOV AX, 9
2 CALL kare_art ; AX = x·x + x
3 RET
4 kare_art PROC
5 PUSH BP
6 MOV BP, SP
7 SUB SP, 2 ; yerel alan [BP-2]
8 MOV [BP-2], AX ; yerel = x
9 MUL AX ; AX = x·x
10 ADD AX, [BP-2] ; + x
11 MOV SP, BP ; yereli at
12 POP BP
13 RET
14 kare_art ENDP
```



CALL : dönüş adresi 0106h → [FFFC].

Adım 2 / 10

DEĞİŞKENLER

AX = 9

BP = 0000h

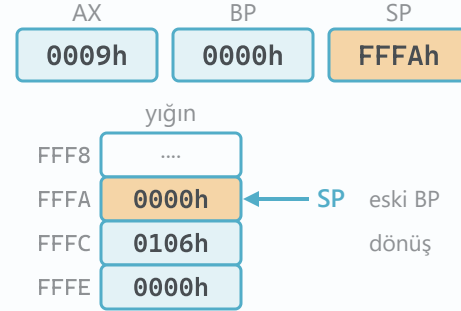
SP = FFFCh

Kod İzleme: Yığında Yerel Değişken

```

1 MOV AX, 9
2 CALL kare_art ; AX = x·x + x
3 RET
4 kare_art PROC
▶ PUSH BP
6 MOV BP, SP
7 SUB SP, 2 ; yerel alan [BP-2]
8 MOV [BP-2], AX ; yerel = x
9 MUL AX ; AX = x·x
10 ADD AX, [BP-2] ; + x
11 MOV SP, BP ; yereli at
12 POP BP
13 RET
14 kare_art ENDP

```



`PUSH BP` : çağıranın BP'si (0000h) saklandı.

Adım 3 / 10

DEĞİŞKENLER

AX = 9

BP = 0000h

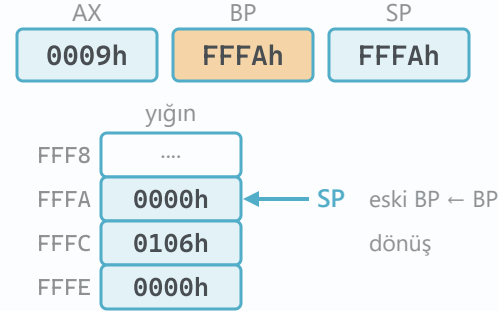
SP = FFFAh

Kod İzleme: Yığında Yerel Değişken

```

1 MOV AX, 9
2 CALL kare_art ; AX = x·x + x
3 RET
4 kare_art PROC
5 PUSH BP
6 MOV BP, SP
7 SUB SP, 2 ; yerel alan [BP-2]
8 MOV [BP-2], AX ; yerel = x
9 MUL AX ; AX = x·x
10 ADD AX, [BP-2] ; + x
11 MOV SP, BP ; yereli at
12 POP BP
13 RET
14 kare_art ENDP

```



MOV BP, SP : BP = FFFAh, çerçevenin sabit noktası.

Adım 4 / 10

DEĞİŞKENLER

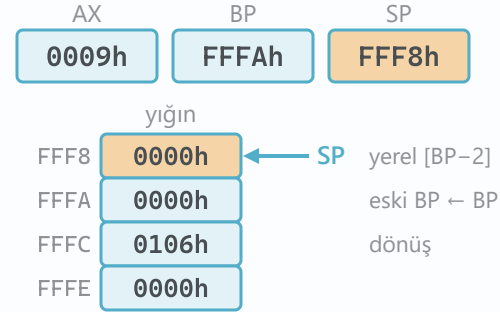
AX = 9

BP = FFFAh

SP = FFFAh

Kod İzleme: Yığında Yerel Değişken

```
1 MOV AX, 9
2 CALL kare_art ; AX = x·x + x
3 RET
4 kare_art PROC
5 PUSH BP
6 MOV BP, SP
7 ► SUB SP, 2 ; yerel alan [BP-2]
8 MOV [BP-2], AX ; yerel = x
9 MUL AX ; AX = x·x
10 ADD AX, [BP-2] ; + x
11 MOV SP, BP ; yereli at
12 POP BP
13 RET
14 kare_art ENDP
```



SUB SP, 2 : bir sözcüklük yerel alan açıldı; [BP-2] = FFF8h.

Adım 5 / 10

DEĞİŞKENLER

AX = 9

BP = FFFAh

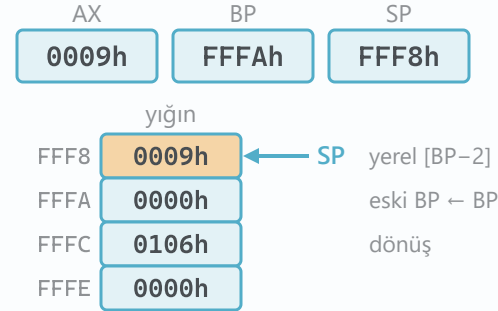
SP = FFF8h

Kod İzleme: Yığında Yerel Değişken

```

1 MOV AX, 9
2 CALL kare_art ; AX = x·x + x
3 RET
4 kare_art PROC
5 PUSH BP
6 MOV BP, SP
7 SUB SP, 2 ; yerel alan [BP-2]
8 ► MOV [BP-2], AX ; yerel = x
9 MUL AX ; AX = x·x
10 ADD AX, [BP-2] ; + x
11 MOV SP, BP ; yereli at
12 POP BP
13 RET
14 kare_art ENDP

```



MOV [BP-2], AX : yerel = 9. [BP...] adresleri varsayılan olarak SS kesimini kullanır.

Adım 6 / 10

DEĞİŞKENLER

AX = 9

BP = FFFAh

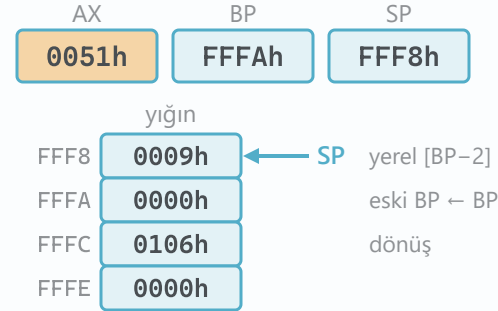
SP = FFF8h

Kod İzleme: Yığında Yerel Değişken

```

1 MOV AX, 9
2 CALL kare_art ; AX = x·x + x
3 RET
4 kare_art PROC
5 PUSH BP
6 MOV BP, SP
7 SUB SP, 2 ; yerel alan [BP-2]
8 MOV [BP-2], AX ; yerel = x
9 ► MUL AX ; AX = x·x
10 ADD AX, [BP-2] ; + x
11 MOV SP, BP ; yereli at
12 POP BP
13 RET
14 kare_art ENDP

```



`MUL AX` : $AX = 81$ ($DX = 0$). AX değişti ama x yerelde duruyor.

Adım 7 / 10

DEĞİŞKENLER

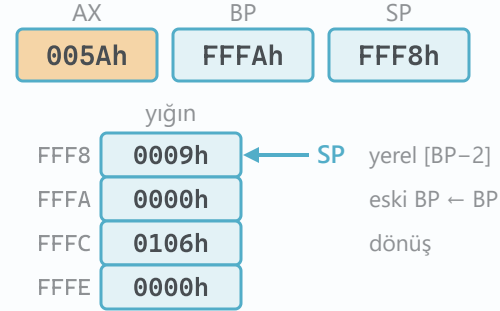
AX = 81

BP = FFFAh

SP = FFF8h

Kod İzleme: Yığında Yerel Değişken

```
1 MOV AX, 9
2 CALL kare_art ; AX = x·x + x
3 RET
4 kare_art PROC
5 PUSH BP
6 MOV BP, SP
7 SUB SP, 2 ; yerel alan [BP-2]
8 MOV [BP-2], AX ; yerel = x
9 MUL AX ; AX = x·x
▶ ADD AX, [BP-2] ; + x
11 MOV SP, BP ; yereli at
12 POP BP
13 RET
14 kare_art ENDP
```



ADD AX, [BP-2] : $81 + 9 = 90 = 005Ah$.

Adım 8 / 10

DEĞİŞKENLER

AX = 90

BP = FFFAh

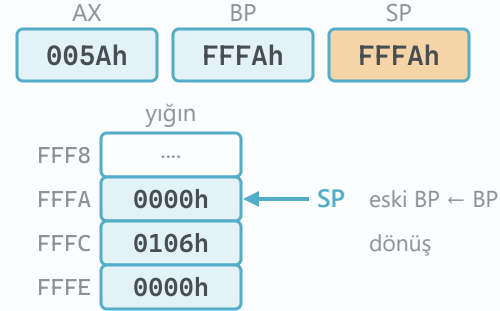
SP = FFF8h

Kod İzleme: Yığında Yerel Değişken

```

1 MOV  AX, 9
2 CALL kare_art    ; AX = x·x + x
3 RET
4 kare_art PROC
5 PUSH BP
6 MOV  BP, SP
7 SUB  SP, 2      ; yerel alan [BP-2]
8 MOV  [BP-2], AX ; yerel = x
9 MUL  AX        ; AX = x·x
10 ADD AX, [BP-2] ; + x
11 ► MOV SP, BP   ; yereli at
12 POP  BP
13 RET
14 kare_art ENDP

```



MOV SP, BP : yerel alan tek komutla bırakıldı, SP = FFFAh.

Adım 9 / 10

DEĞİŞKENLER

AX = 90

BP = FFFAh

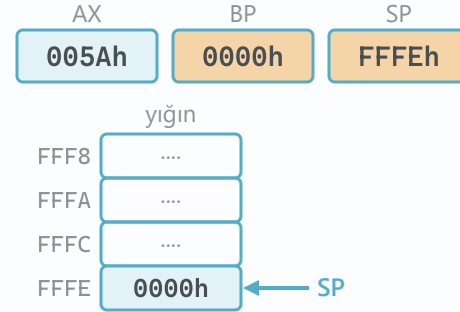
SP = FFFAh

Kod İzleme: Yığında Yerel Değişken

```

1 MOV AX, 9
2 CALL kare_art ; AX = x·x + x
3 RET
4 kare_art PROC
5 PUSH BP
6 MOV BP, SP
7 SUB SP, 2 ; yerel alan [BP-2]
8 MOV [BP-2], AX ; yerel = x
9 MUL AX ; AX = x·x
10 ADD AX, [BP-2] ; + x
11 MOV SP, BP ; yereli at
12 POP BP
13 RET
14 kare_art ENDP

```



POP BP, RET : BP = 0, SP = FFFEh, çağırana dönüldü. Sonuç AX = 90.

Adım 10 / 10

DEĞİŞKENLER

AX = 90

BP = 0000h

SP = FFFEh

ÖRNEK PROGRAMLAR

Merhaba Dünya Mesajı

- Prosedür kullanarak ekrana mesaj yazdırma.
- `LEA SI, msg` : msg adlı dizginin **adresini** SI yazmacına yükler (parametre = adres).
- `CALL print_me` : print_me prosedürünü çağırır.
- `RET` : işletim sistemine geri döner.
- `print_me` , **null (0) ile biten** bir dizgiyi karakter karakter yazdırır.
- `b.[SI]` emu8086'nın `BYTE PTR [SI]` kısaltmasıdır: `CMP [SI], 0` işlem boyutunu belirtmez, bu yüzden boyut açıkça yazılır.

```
ORG    100h
LEA     SI, msg      ; msg adresi → SI
CALL    print_me
RET                                ; DOS'a dön
```

```
print_me PROC
        ; .....
print_me ENDP
```

```
msg DB 'Hello World!', 0
END
```

Kod İzleme: `print_me`

```
1 ORG    100h
▶ 2 LEA    SI, msg      ; SI = msg adresi
3 CALL   print_me
4 RET                                ; DOS'a dön
5 print_me PROC
6 next_char:
7 CMP    b.[SI], 0      ; sıfır mı?
8 JE     stop           ; evet: bitir
9 MOV    AL, [SI]       ; sıradaki ASCII karakter
10 MOV   AH, 0Eh        ; teletype işlevi
11 INT    10h           ; AL'yi ekrana yaz
12 ADD    SI, 1          ; sonraki karakter
13 JMP    next_char     ; başa dön
14 stop:
15 RET                                ; çağırana dön
16 print_me ENDP
17 msg DB 'Hello World!', 0
18 END
```

DEĞİŞKENLER

SI = `msg+0`

SP = `FFFEh`

ÇAĞRI YİĞİNİ

[`FFFEh`] `0000h`

ÇIKTI

SI, dizginin ilk karakterini ('H') gösterir.

Adım 1 / 12

Kod İzleme: `print_me`

```
1 ORG    100h
2 LEA    SI, msg      ; SI = msg adresi
3 ► CALL  print_me
4 RET                                ; DOS'a dön
5 print_me PROC
6 next_char:
7 CMP    b.[SI], 0     ; sıfır mı?
8 JE     stop          ; evet: bitir
9 MOV    AL, [SI]      ; sıradaki ASCII karakter
10 MOV   AH, 0Eh        ; teletype işlevi
11 INT    10h          ; AL'yi ekrana yaz
12 ADD    SI, 1         ; sonraki karakter
13 JMP    next_char     ; başa dön
14 stop:
15 RET                                ; çağırana dön
16 print_me ENDP
17 msg DB 'Hello World!', 0
18 END
```

DEĞİŞKENLER

SI = `msg+0`

SP = `FFFCh`

ÇAĞRI YIĞINI

[`FFFCh`] dönüş → satır 4

[`FFFEh`] `0000h`

ÇIKTI

`CALL print_me` : dönüş adresi (4. satırdaki RET) yığına itilir.

Adım 2 / 12

Kod İzleme: `print_me`

```
1 ORG    100h
2 LEA    SI, msg      ; SI = msg adresi
3 CALL   print_me
4 RET                      ; DOS'a dön
5 print_me PROC
6 next_char:
7 ► CMP  b.[SI], 0      ; sıfır mı?
8 ► JE   stop          ; evet: bitir
9 MOV    AL, [SI]       ; sıradaki ASCII karakter
10 MOV   AH, 0Eh        ; teletype işlevi
11 INT    10h           ; AL'yi ekrana yaz
12 ADD    SI, 1         ; sonraki karakter
13 JMP    next_char     ; başa dön
14 stop:
15 RET                      ; çağırana dön
16 print_me ENDP
17 msg DB 'Hello World!', 0
18 END
```

DEĞİŞKENLER

SI = `msg+0`

ZF = 0

SP = `FFFCh`

ÇAĞRI YİĞİNİ

[`FFFCh`] dönüş → satır 4

[`FFFEh`] `0000h`

ÇIKTI

[SI] = 'H' (48h) ≠ 0 → ZF = 0, JE atlamaz.

Adım 3 / 12

Kod İzleme: `print_me`

```
1 ORG    100h
2 LEA    SI, msg      ; SI = msg adresi
3 CALL   print_me
4 RET                      ; DOS'a dön
5 print_me PROC
6 next_char:
7 CMP    b.[SI], 0     ; sıfır mı?
8 JE     stop          ; evet: bitir
9 ▶ MOV   AL, [SI]      ; sıradaki ASCII karakter
10 ▶ MOV  AH, 0Eh       ; teletype işlevi
11 ▶ INT  10h           ; AL'yi ekrana yaz
12 ADD   SI, 1         ; sonraki karakter
13 JMP   next_char      ; başa dön
14 stop:
15 RET                      ; çağırana dön
16 print_me ENDP
17 msg DB 'Hello World!', 0
18 END
```

DEĞİŞKENLER

SI = `msg+0`

AL = **48h**

AH = **0Eh**

ZF = **0**

SP = **FFFCh**

ÇAĞRI YİĞİNİ

[**FFFCh**] dönüş → **satır 4**

[**FFFEh**] 0000h

ÇIKTI

H

AL = 48h, AH = 0Eh, INT 10h → **H**.

Adım 4 / 12

Kod İzleme: `print_me`

```
1 ORG    100h
2 LEA    SI, msg      ; SI = msg adresi
3 CALL   print_me
4 RET                      ; DOS'a dön
5 print_me PROC
6 next_char:
7 CMP    b.[SI], 0     ; sıfır mı?
8 JE     stop          ; evet: bitir
9 MOV    AL, [SI]       ; sıradaki ASCII karakter
10 MOV   AH, 0Eh        ; teletype işlevi
11 INT    10h           ; AL'yi ekrana yaz
12 ▶ ADD   SI, 1         ; sonraki karakter
13 ▶ JMP   next_char     ; başa dön
14 stop:
15 RET                      ; çağırana dön
16 print_me ENDP
17 msg DB 'Hello World!', 0
18 END
```

DEĞİŞKENLER

SI = msg+1

AL = 48h

AH = 0Eh

ZF = 0

SP = FFFCh

ÇAĞRI YİĞİNİ

[FFFCh] dönüş → satır 4

[FFFEh] 0000h

ÇIKTI

H

SI = msg+1, `next_char` 'a dön.

Adım 5 / 12

Kod İzleme: `print_me`

```
1 ORG    100h
2 LEA    SI, msg      ; SI = msg adresi
3 CALL   print_me
4 RET                      ; DOS'a dön
5 print_me PROC
6 next_char:
7 ▶ CMP  b.[SI], 0      ; sıfır mı?
8 ▶ JE   stop           ; evet: bitir
9 ▶ MOV  AL, [SI]       ; sıradaki ASCII karakter
10 ▶ MOV AH, 0Eh        ; teletype işlevi
11 ▶ INT  10h           ; AL'yi ekrana yaz
12 ▶ ADD SI, 1          ; sonraki karakter
13 ▶ JMP next_char      ; başa dön
14 stop:
15 RET                  ; çağırana dön
16 print_me ENDP
17 msg DB 'Hello World!', 0
18 END
```

DEĞİŞKENLER

SI = msg+2

AL = 65h

AH = 0Eh

ZF = 0

SP = FFFCh

ÇAĞRI YİĞİNİ

[FFFCh] dönüş → satır 4

[FFFEh] 0000h

ÇIKTI

He

Aynı döngü: 'e' yazılır. AL = 65h, SI = msg+2.

Adım 6 / 12

Kod İzleme: `print_me`

```
1 ORG    100h
2 LEA    SI, msg      ; SI = msg adresi
3 CALL   print_me
4 RET                      ; DOS'a dön
5 print_me PROC
6 next_char:
7 ▶ CMP  b.[SI], 0      ; sıfır mı?
8 ▶ JE   stop           ; evet: bitir
9 ▶ MOV  AL, [SI]        ; sıradaki ASCII karakter
10 ▶ MOV AH, 0Eh         ; teletype işlevi
11 ▶ INT  10h            ; AL'yi ekrana yaz
12 ▶ ADD SI, 1           ; sonraki karakter
13 ▶ JMP next_char       ; başa dön
14 stop:
15 RET                      ; çağırana dön
16 print_me ENDP
17 msg DB 'Hello World!', 0
18 END
```

DEĞİŞKENLER

SI = msg+5

AL = 6Fh

AH = 0Eh

ZF = 0

SP = FFFCh

ÇAĞRI YİĞİNİ

[FFFCh] dönüş → satır 4

[FFFEh] 0000h

ÇIKTI

Hello

Döngü 3 tur daha: "llo" yazılır. AL = 6Fh, SI = msg+5.

Adım 7 / 12

Kod İzleme: `print_me`

```
1 ORG    100h
2 LEA    SI, msg      ; SI = msg adresi
3 CALL   print_me
4 RET                      ; DOS'a dön
5 print_me PROC
6 next_char:
7 ▶ CMP  b.[SI], 0      ; sıfır mı?
8 ▶ JE   stop           ; evet: bitir
9 ▶ MOV  AL, [SI]       ; sıradaki ASCII karakter
10 ▶ MOV AH, 0Eh        ; teletype işlevi
11 ▶ INT  10h           ; AL'yi ekrana yaz
12 ▶ ADD SI, 1          ; sonraki karakter
13 ▶ JMP next_char      ; başa dön
14 stop:
15 RET                  ; çağırana dön
16 print_me ENDP
17 msg DB 'Hello World!', 0
18 END
```

DEĞİŞKENLER

SI = msg+11

AL = 64h

AH = 0Eh

ZF = 0

SP = FFFCh

ÇAĞRI YİĞİNİ

[FFFCh] dönüş → satır 4

[FFFEh] 0000h

ÇIKTI

Hello World

Döngü 6 tur daha: " World" yazılır (boşluk da bir karakterdir: 20h). AL = 64h, SI = msg+11.

Adım 8 / 12

Kod İzleme: `print_me`

```
1 ORG    100h
2 LEA    SI, msg      ; SI = msg adresi
3 CALL   print_me
4 RET                      ; DOS'a dön
5 print_me PROC
6 next_char:
7 ▶ CMP  b.[SI], 0      ; sıfır mı?
8 ▶ JE   stop           ; evet: bitir
9 ▶ MOV  AL, [SI]        ; sıradaki ASCII karakter
10 ▶ MOV AH, 0Eh         ; teletype işlevi
11 ▶ INT  10h            ; AL'yi ekrana yaz
12 ▶ ADD SI, 1           ; sonraki karakter
13 ▶ JMP next_char       ; başa dön
14 stop:
15 RET                      ; çağırana dön
16 print_me ENDP
17 msg DB 'Hello World!', 0
18 END
```

DEĞİŞKENLER

SI = msg+12

AL = 21h

AH = 0Eh

ZF = 0

SP = FFFCh

ÇAĞRI YİĞİNİ

[FFFCh] dönüş → satır 4

[FFFEh] 0000h

ÇIKTI

Hello World!

Aynı döngü: '!' yazılır. AL = 21h, SI = msg+12.

Adım 9 / 12

Kod İzleme: `print_me`

```
1 ORG    100h
2 LEA    SI, msg      ; SI = msg adresi
3 CALL   print_me
4 RET                      ; DOS'a dön
5 print_me PROC
6 next_char:
7 ► CMP  b.[SI], 0      ; sıfır mı?
8 ► JE   stop          ; evet: bitir
9 MOV    AL, [SI]       ; sıradaki ASCII karakter
10 MOV   AH, 0Eh        ; teletype işlevi
11 INT    10h           ; AL'yi ekrana yaz
12 ADD   SI, 1          ; sonraki karakter
13 JMP   next_char      ; başa dön
14 stop:
15 RET                      ; çağırana dön
16 print_me ENDP
17 msg DB 'Hello World!', 0
18 END
```

DEĞİŞKENLER

SI = **msg+12**

AL = **21h**

AH = **0Eh**

ZF = **1**

SP = **FFFCh**

ÇAĞRI YİĞİNİ

[**FFFCh**] dönüş → **satır 4**

[**FFFEh**] 0000h

ÇIKTI

Hello World!

[SI] = 0 → ZF = 1, `JE stop` atlar.

Adım 10 / 12

Kod İzleme: `print_me`

```
1 ORG    100h
2 LEA    SI, msg      ; SI = msg adresi
3 CALL   print_me
4 RET                      ; DOS'a dön
5 print_me PROC
6 next_char:
7 CMP    b.[SI], 0     ; sıfır mı?
8 JE     stop          ; evet: bitir
9 MOV    AL, [SI]       ; sıradaki ASCII karakter
10 MOV   AH, 0Eh        ; teletype işlevi
11 INT    10h           ; AL'yi ekrana yaz
12 ADD    SI, 1         ; sonraki karakter
13 JMP    next_char     ; başa dön
14 stop:
15 ▶ RET                ; çağırana dön
16 print_me ENDP
17 msg DB 'Hello World!', 0
18 END
```

DEĞİŞKENLER

SI = `msg+12`AL = `21h`AH = `0Eh`ZF = `1`SP = `FFFEh`

ÇAĞRI YİĞİNİ

[`FFFEh`] `0000h`

ÇIKTI

Hello World!

Prosedürün `RET` 'i dönüş adresini alır → 4. satıra dönülür.

Adım 11 / 12

Kod İzleme: `print_me`

```
1 ORG    100h
2 LEA    SI, msg      ; SI = msg adresi
3 CALL   print_me
▶ 4 RET      ; DOS'a dön
5 print_me PROC
6 next_char:
7 CMP    b.[SI], 0     ; sıfır mı?
8 JE     stop          ; evet: bitir
9 MOV    AL, [SI]       ; sıradaki ASCII karakter
10 MOV   AH, 0Eh        ; teletype işlevi
11 INT    10h           ; AL'yi ekrana yaz
12 ADD   SI, 1          ; sonraki karakter
13 JMP    next_char     ; başa dön
14 stop:
15 RET      ; çağırana dön
16 print_me ENDP
17 msg DB 'Hello World!', 0
18 END
```

DEĞİŞKENLER

SI = `msg+12`

AL = `21h`

AH = `0Eh`

ZF = `1`

SP = `0000h`

ÇAĞRI YİĞİNİ

boş

ÇIKTI

Hello World!

Ana programın `RET` 'i DOS'a döner. Ekranda:
Hello World!

Adım 12 / 12

Kod İzleme: İki Değişken Toplama

```
1 ORG 100h
2 start:
3 ► CALL AddNumbers ; prosedürü çağır
4 RET                ; DOS'a dön
5 AddNumbers PROC
6 MOV AX, number1    ; AX = ilk sayı
7 ADD AX, number2     ; AX = AX + ikinci sayı
8 MOV result, AX      ; sonucu sakla
9 RET                 ; prosedürü bitir
10 AddNumbers ENDP
11 number1 DW 5        ; ilk sayı 5
12 number2 DW 3        ; ikinci sayı 3
13 result DW ?         ; sonucu tutar
```

DEĞİŞKENLER

SP = FFFEh

result = ?

ÇAĞRI YİĞİNİ

[FFFEh] 0000h

Parametreler bellek değişkenlerinde: number1 = 5, number2 = 3.

Adım 1 / 7

Kod İzleme: İki Değişken Toplama

```
1 ORG 100h
2 start:
3 ► CALL AddNumbers ; prosedürü çağır
4 RET ; DOS'a dön
5 AddNumbers PROC
6 MOV AX, number1 ; AX = ilk sayı
7 ADD AX, number2 ; AX = AX + ikinci sayı
8 MOV result, AX ; sonucu sakla
9 RET ; prosedürü bitir
10 AddNumbers ENDP
11 number1 DW 5 ; ilk sayı 5
12 number2 DW 3 ; ikinci sayı 3
13 result DW ? ; sonucu tutar
```

DEĞİŞKENLER

SP = FFFCh

result = ?

ÇAĞRI YİĞİNİ

[FFFCh] 0103h

[FFFEh] 0000h

CALL AddNumbers (0100h, 3 bayt): dönüş
adresi **0103h** itilir.

Adım 2 / 7

Kod İzleme: İki Değişken Toplama

```
1 ORG 100h
2 start:
3 CALL AddNumbers ; prosedürü çağır
4 RET             ; DOS'a dön
5 AddNumbers PROC
6 ► MOV AX, number1 ; AX = ilk sayı
7 ADD AX, number2 ; AX = AX + ikinci sayı
8 MOV result, AX ; sonucu sakla
9 RET             ; prosedürü bitir
10 AddNumbers ENDP
11 number1 DW 5 ; ilk sayı 5
12 number2 DW 3 ; ikinci sayı 3
13 result DW ? ; sonucu tutar
```

DEĞİŞKENLER

SP = FFFCh

AX = 0005h

result = ?

ÇAĞRI YİĞİNİ

[FFFCh] 0103h

[FFFEh] 0000h

AX = number1 = 0005h.

Adım 3 / 7

Kod İzleme: İki Değişken Toplama

```
1 ORG 100h
2 start:
3 CALL AddNumbers    ; prosedürü çağır
4 RET                ; DOS'a dön
5 AddNumbers PROC
6 MOV AX, number1    ; AX = ilk sayı
7 ► ADD AX, number2  ; AX = AX + ikinci sayı
8 MOV result, AX     ; sonucu sakla
9 RET                ; prosedürü bitir
10 AddNumbers ENDP
11 number1 DW 5       ; ilk sayı 5
12 number2 DW 3       ; ikinci sayı 3
13 result DW ?        ; sonucu tutar
```

DEĞİŞKENLER

SP = FFFCh

AX = 0008h

result = ?

ZF = 0

CF = 0

ÇAĞRI YİĞİNİ

[FFFCh] 0103h

[FFFEh] 0000h

AX = 5 + 3 = 0008h; sonuç sıfır değil, elde yok.

Adım 4 / 7

Kod İzleme: İki Değişken Toplama

```
1 ORG 100h
2 start:
3 CALL AddNumbers ; prosedürü çağır
4 RET             ; DOS'a dön
5 AddNumbers PROC
6 MOV AX, number1 ; AX = ilk sayı
7 ADD AX, number2 ; AX = AX + ikinci sayı
8 ► MOV result, AX ; sonucu sakla
9 RET             ; prosedürü bitir
10 AddNumbers ENDP
11 number1 DW 5    ; ilk sayı 5
12 number2 DW 3    ; ikinci sayı 3
13 result DW ?     ; sonucu tutar
```

DEĞİŞKENLER

SP = FFFCh

AX = 0008h

result = 0008h

ZF = 0

CF = 0

ÇAĞRI YİĞİNİ

[FFFCh] 0103h

[FFFEh] 0000h

result = 0008h (bellekte 08 00).

Adım 5 / 7

Kod İzleme: İki Değişken Toplama

```
1 ORG 100h
2 start:
3 CALL AddNumbers ; prosedürü çağır
4 RET             ; DOS'a dön
5 AddNumbers PROC
6 MOV AX, number1 ; AX = ilk sayı
7 ADD AX, number2 ; AX = AX + ikinci sayı
8 MOV result, AX  ; sonucu sakla
9 ► RET           ; prosedürü bitir
10 AddNumbers ENDP
11 number1 DW 5    ; ilk sayı 5
12 number2 DW 3    ; ikinci sayı 3
13 result DW ?     ; sonucu tutar
```

DEĞİŞKENLER

SP = FFFEh

AX = 0008h

result = 0008h

ZF = 0

CF = 0

ÇAĞRI YIĞINI

[FFFEh] 0000h

RET → IP = 0103h.

Adım 6 / 7

Kod İzleme: İki Değişken Toplama

```
1 ORG 100h
2 start:
3 CALL AddNumbers ; prosedürü çağır
4 ► RET           ; DOS'a dön
5 AddNumbers PROC
6 MOV AX, number1 ; AX = ilk sayı
7 ADD AX, number2 ; AX = AX + ikinci sayı
8 MOV result, AX  ; sonucu sakla
9 RET             ; prosedürü bitir
10 AddNumbers ENDP
11 number1 DW 5    ; ilk sayı 5
12 number2 DW 3    ; ikinci sayı 3
13 result DW ?     ; sonucu tutar
```

DEĞİŞKENLER

SP = 0000h

AX = 0008h

result = 0008h

ZF = 0

CF = 0

ÇAĞRI YİĞİNİ

boş

RET → DOS. DW ? başlangıç değeri
belirsiz demektir (emu8086 0 koyar).

Adım 7 / 7

Kod İzleme: Adres ve Sayaçla Parametre

```
1 ORG 100h
▶ 2 LEA SI, vector ; parametre 1: adres
3 MOV CX, 5 ; parametre 2: adet
4 CALL topla
5 MOV m, AL ; dönüş değeri AL'de
6 RET
7 topla PROC ; Girdi: SI, CX
8 MOV AL, 0 ; Çıktı: AL = toplam
9 dongu:
10 ADD AL, [SI]
11 INC SI
12 LOOP dongu
13 RET
14 topla ENDP
15 vector DB 5, 4, 5, 2, 1
16 m DB 0
```

DEĞİŞKENLER

SI = **vector+0**

SP = **FFFEh**

m = **00h**

ÇAĞRI YIĞINI

[**FFFEh**] **0000h**

Dizi toplamı, yeniden kullanılabilir bir prosedür olarak: SI = dizinin adresi.

Adım 1 / 12

Kod İzleme: Adres ve Sayaçla Parametre

```
1 ORG 100h
2 LEA SI, vector      ; parametre 1: adres
3 ► MOV CX, 5          ; parametre 2: adet
4 CALL toplama
5 MOV m, AL            ; dönüş değeri AL'de
6 RET
7 toplama PROC         ; Girdi: SI, CX
8 MOV AL, 0            ; Çıktı: AL = toplam
9 dongu:
10 ADD AL, [SI]
11 INC SI
12 LOOP dongu
13 RET
14 toplama ENDP
15 vector DB 5, 4, 5, 2, 1
16 m DB 0
```

DEĞİŞKENLER

SI = **vector+0**

CX = **0005h**

SP = **FFFEh**

m = **00h**

ÇAĞRI YİĞİNİ

[**FFFEh**] **0000h**

CX = 5 eleman.

Adım 2 / 12

Kod İzleme: Adres ve Sayaçla Parametre

```
1 ORG 100h
2 LEA SI, vector      ; parametre 1: adres
3 MOV CX, 5           ; parametre 2: adet
4 ► CALL toplama
5 MOV m, AL           ; dönüş değeri AL'de
6 RET
7 toplama PROC        ; Girdi: SI, CX
8 MOV AL, 0           ; Çıktı: AL = toplam
9 dongu:
10 ADD AL, [SI]
11 INC SI
12 LOOP dongu
13 RET
14 toplama ENDP
15 vector DB 5, 4, 5, 2, 1
16 m      DB 0
```

DEĞİŞKENLER

SI = **vector+0**

CX = **0005h**

SP = **FFFFh**

m = **00h**

ÇAĞRI YIĞINI

[FFFFh] dönüş → satır 5

[FFFEh] 0000h

CALL toplama : dönüş adresi yığına.

Adım 3 / 12

Kod İzleme: Adres ve Sayaçla Parametre

```
1 ORG 100h
2 LEA SI, vector      ; parametre 1: adres
3 MOV CX, 5           ; parametre 2: adet
4 CALL toplama
5 MOV m, AL           ; dönüş değeri AL'de
6 RET
7 toplama PROC        ; Girdi: SI, CX
▶ 8 MOV AL, 0          ; Çıktı: AL = toplam
9 dongu:
10 ADD AL, [SI]
11 INC SI
12 LOOP dongu
13 RET
14 toplama ENDP
15 vector DB 5, 4, 5, 2, 1
16 m      DB 0
```

DEĞİŞKENLER

SI = **vector+0**

CX = **0005h**

AL = **00h**

SP = **FFFFh**

m = **00h**

ÇAĞRI YİĞİNİ

[FFFFh] dönüş → satır 5

[FFFEh] 0000h

AL = 0.

Adım 4 / 12

Kod İzleme: Adres ve Sayaçla Parametre

```
1 ORG 100h
2 LEA SI, vector      ; parametre 1: adres
3 MOV CX, 5           ; parametre 2: adet
4 CALL toplama
5 MOV m, AL           ; dönüş değeri AL'de
6 RET
7 toplama PROC        ; Girdi: SI, CX
8 MOV AL, 0           ; Çıktı: AL = toplam
9 dongu:
10  ▶ ADD AL, [SI]
11  ▶ INC SI
12  ▶ LOOP dongu
13 RET
14 toplama ENDP
15 vector DB 5, 4, 5, 2, 1
16 m      DB 0
```

DEĞİŞKENLER

SI = **vector+1**

CX = **0004h**

AL = **05h**

SP = **FFFFh**

m = **00h**

ÇAĞRI YİĞİNİ

[**FFFFh**] dönüş → **satır 5**

[**FFFEh**] **0000h**

+5 → AL = 5; SI ilerler, CX = 4.

Adım 5 / 12

Kod İzleme: Adres ve Sayaçla Parametre

```
1 ORG 100h
2 LEA SI, vector      ; parametre 1: adres
3 MOV CX, 5           ; parametre 2: adet
4 CALL toplama
5 MOV m, AL           ; dönüş değeri AL'de
6 RET
7 toplama PROC        ; Girdi: SI, CX
8 MOV AL, 0           ; Çıktı: AL = toplam
9 dongu:
10  ► ADD AL, [SI]
11  ► INC SI
12  ► LOOP dongu
13 RET
14 toplama ENDP
15 vector DB 5, 4, 5, 2, 1
16 m      DB 0
```

DEĞİŞKENLER

SI = **vector+2**

CX = **0003h**

AL = **09h**

SP = **FFFFh**

m = **00h**

ÇAĞRI YİĞİNİ

[FFFFh] dönüş → satır 5

[FFFEh] 0000h

+4 → AL = 9; SI ilerler, CX = 3.

Adım 6 / 12

Kod İzleme: Adres ve Sayaçla Parametre

```
1 ORG 100h
2 LEA SI, vector      ; parametre 1: adres
3 MOV CX, 5           ; parametre 2: adet
4 CALL toplama
5 MOV m, AL           ; dönüş değeri AL'de
6 RET
7 toplama PROC        ; Girdi: SI, CX
8 MOV AL, 0           ; Çıktı: AL = toplam
9 dongu:
10 ► ADD AL, [SI]
11 ► INC SI
12 ► LOOP dongu
13 RET
14 toplama ENDP
15 vector DB 5, 4, 5, 2, 1
16 m DB 0
```

DEĞİŞKENLER

SI = **vector+3**

CX = **0002h**

AL = **0Eh**

SP = **FFFFh**

m = **00h**

ÇAĞRI YIĞINI

[**FFFFh**] dönüş → **satır 5**

[**FFFEh**] **0000h**

+5 → AL = 14; SI ilerler, CX = 2.

Adım 7 / 12

Kod İzleme: Adres ve Sayaçla Parametre

```
1 ORG 100h
2 LEA SI, vector      ; parametre 1: adres
3 MOV CX, 5           ; parametre 2: adet
4 CALL toplama
5 MOV m, AL           ; dönüş değeri AL'de
6 RET
7 toplama PROC        ; Girdi: SI, CX
8 MOV AL, 0           ; Çıktı: AL = toplam
9 dongu:
10  ▶ ADD AL, [SI]
11  ▶ INC SI
12  ▶ LOOP dongu
13 RET
14 toplama ENDP
15 vector DB 5, 4, 5, 2, 1
16 m      DB 0
```

DEĞİŞKENLER

SI = **vector+4**

CX = **0001h**

AL = **10h**

SP = **FFFCh**

m = **00h**

ÇAĞRI YİĞİNİ

[FFFCh] dönüş → **satır 5**

[FFFEh] **0000h**

+2 → AL = 16; SI ilerler, CX = 1.

Adım 8 / 12

Kod İzleme: Adres ve Sayaçla Parametre

```
1 ORG 100h
2 LEA SI, vector      ; parametre 1: adres
3 MOV CX, 5           ; parametre 2: adet
4 CALL toplama
5 MOV m, AL           ; dönüş değeri AL'de
6 RET
7 toplama PROC        ; Girdi: SI, CX
8 MOV AL, 0           ; Çıktı: AL = toplam
9 dongu:
10  ► ADD AL, [SI]
11  ► INC SI
12  ► LOOP dongu
13 RET
14 toplama ENDP
15 vector DB 5, 4, 5, 2, 1
16 m      DB 0
```

DEĞİŞKENLER

SI = **vector+5**

CX = **0000h**

AL = **11h**

SP = **FFFFh**

m = **00h**

ÇAĞRI YİĞİNİ

[FFFFh] dönüş → satır 5

[FFFEh] 0000h

+1 → AL = 17; SI ilerler, CX = 0 → döngü biter.

Adım 9 / 12

Kod İzleme: Adres ve Sayaçla Parametre

```
1 ORG 100h
2 LEA SI, vector      ; parametre 1: adres
3 MOV CX, 5           ; parametre 2: adet
4 CALL toplama
5 MOV m, AL           ; dönüş değeri AL'de
6 RET
7 toplama PROC        ; Girdi: SI, CX
8 MOV AL, 0           ; Çıktı: AL = toplam
9 dongu:
10 ADD AL, [SI]
11 INC SI
12 LOOP dongu
13 ► RET
14 toplama ENDP
15 vector DB 5, 4, 5, 2, 1
16 m DB 0
```

DEĞİŞKENLER

SI = **vector+5**

CX = **0000h**

AL = **11h**

SP = **FFFEh**

m = **00h**

ÇAĞRI YİĞİNİ

[FFFEh] 0000h

RET : çağırana dönülür, sonuç AL'de.

Adım 10 / 12

Kod İzleme: Adres ve Sayaçla Parametre

```
1 ORG 100h
2 LEA SI, vector      ; parametre 1: adres
3 MOV CX, 5           ; parametre 2: adet
4 CALL toplama
5 MOV m, AL           ; dönüş değeri AL'de
6 RET
7 toplama PROC        ; Girdi: SI, CX
8 MOV AL, 0           ; Çıktı: AL = toplam
9 dongu:
10 ADD AL, [SI]
11 INC SI
12 LOOP dongu
13 RET
14 toplama ENDP
15 vector DB 5, 4, 5, 2, 1
16 m DB 0
```

DEĞİŞKENLER

SI = **vector+5**

CX = **0000h**

AL = **11h**

SP = **FFFEh**

m = 11h

ÇAĞRI YİĞİNİ

[**FFFEh**] **0000h**

m = 11h = 17.

Adım 11 / 12

Kod İzleme: Adres ve Sayaçla Parametre

```
1 ORG 100h
2 LEA SI, vector      ; parametre 1: adres
3 MOV CX, 5           ; parametre 2: adet
4 CALL toplama
5 MOV m, AL           ; dönüş değeri AL'de
▶ RET
7 toplama PROC        ; Girdi: SI, CX
8 MOV AL, 0           ; Çıktı: AL = toplam
9 dongu:
10 ADD AL, [SI]
11 INC SI
12 LOOP dongu
13 RET
14 toplama ENDP
15 vector DB 5, 4, 5, 2, 1
16 m      DB 0
```

DEĞİŞKENLER

SI = **vector+5**

CX = **0000h**

AL = **11h**

SP = **0000h**

m = **11h**

ÇAĞRI YİĞİNİ

boş

Program biter. Aynı prosedür başka bir dizi için yalnızca SI ve CX değiştirilerek çağrılabilir.

Adım 12 / 12

Parametre Bloğu

```
LEA BX, blok
CALL hesap      ; sonuç = a × b + c
RET
hesap PROC      ; Girdi: BX = blok adresi
PUSH AX
PUSH DX          ; MUL DX'i de yazar
MOV AX, [BX]      ; a
MUL WORD PTR [BX+2] ; × b
ADD AX, [BX+4]    ; + c
MOV [BX+6], AX    ; sonuç
POP DX
POP AX
RET
hesap ENDP
blok DW 12, 7, 5, ?
```

[BX+0] a	[BX+2] b	[BX+4] c	[BX+6] sonuç
12	7	5	89

BX → blok

- Tek bir yazmaç (BX) istenen sayıda parametreyi taşır: blok, sabit uzaklıklarla ([BX+2], [BX+4]) okunur.
- Sonuç da bloğa yazılır: $12 \times 7 + 5 = 89$ (0059h).
- Aynı prosedür başka bir blokla, yalnızca BX değiştirilerek çağrılır.
- Blok düzeni (hangi uzaklıkta ne var) prosedürün açıklamasında belgelenmelidir.

Prosedürü Test Etmek

Bir prosedür yalnızca “olağan” girdiyle değil, sınır durumlarıyla da çalıştırılarak denir. **bul** prosedürü (dizi: 3, 9, 7, 1, 4) için sonuçlar:

Durum	AL	CX	CF	BX	Değerlendirme
ilk eleman	3	5	1	0	doğru
son eleman	4	5	1	4	doğru
dizide yok	8	5	0	5	doğru; BX anlamsızdır, yalnızca CF'ye bakılır
tek elemanlık arama	9	1	0	1	doğru: 9 ikinci eleman, aranan aralığın dışında
boş dizi	7	0	1	2	yanlış: CX = 0 iken LOOP 65 536 tur döner

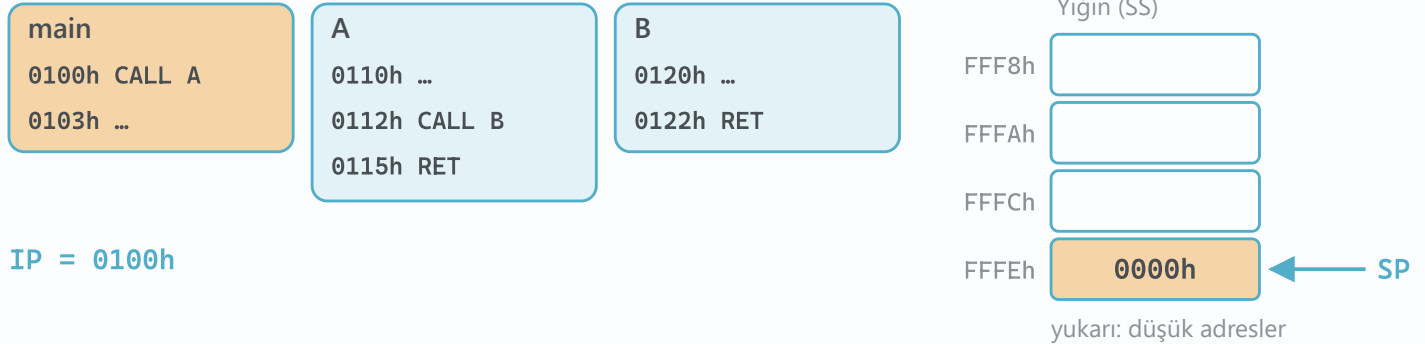
Hatanın nedeni

LOOP önce CX'i azaltır, sonra sıfır mı diye bakar: $0 - 1 = \text{FFFFh}$. Prosedür dizinin ötesindeki baytları da tarar ve oradaki 7'yi “bulur”.

Düzeltilme

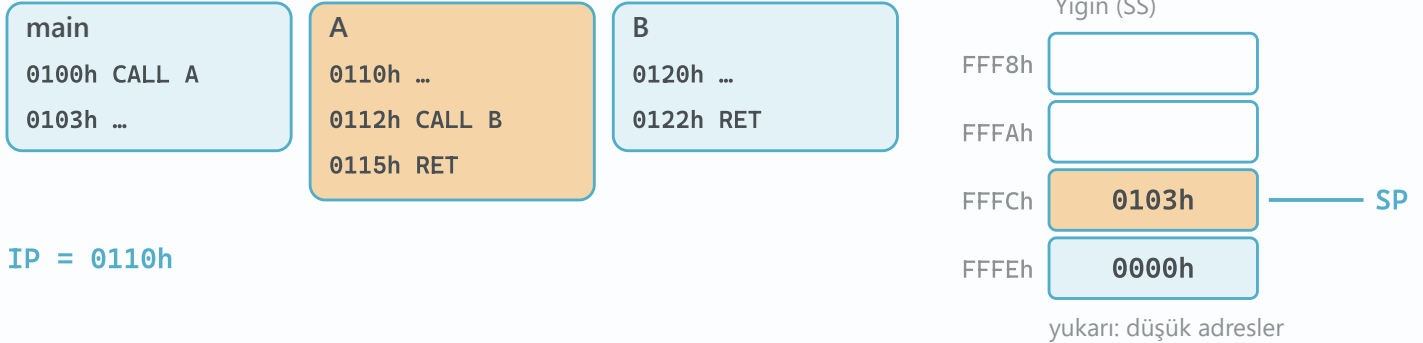
Döngüden önce **JCXZ** ile boş dizi ayrılır: CX = 0 ise doğrudan CLC yapıp **son** etiketine gidilir. Ayrıca CX ve SI'nın korunduğu her testte denetlenir.

İç İçe Çağrılar



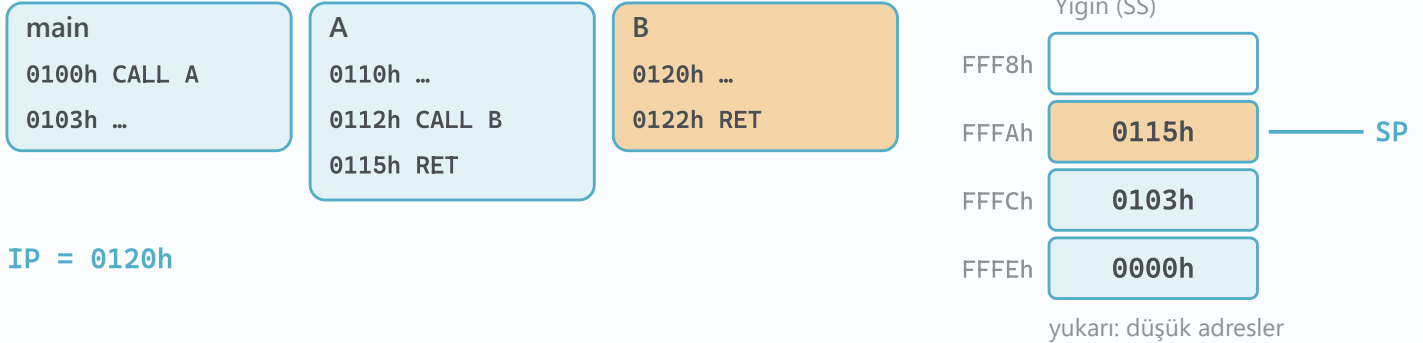
main, A'yı çağırarak üzere. SP = FFEh.

İç İçe Çağrılar



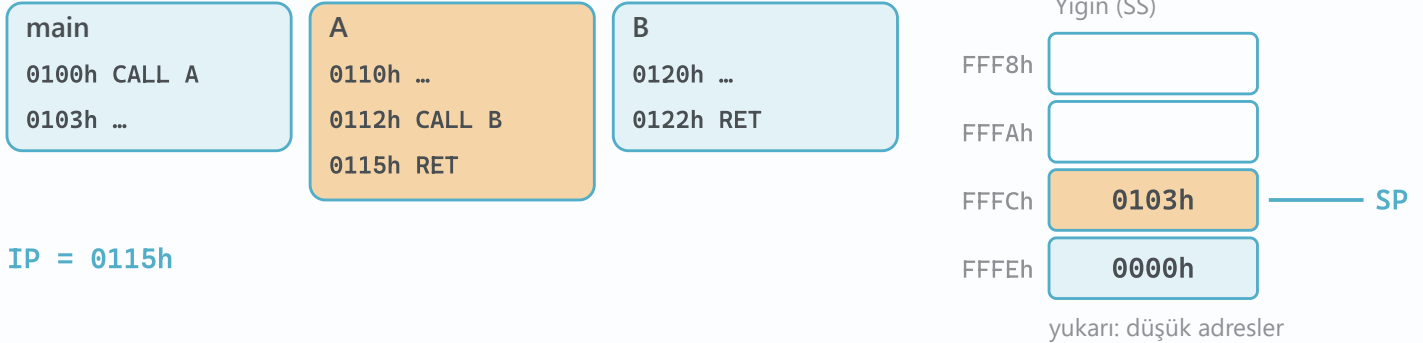
CALL A : 0103h itilir, SP = FFFCh.

İç İçe Çağrılar



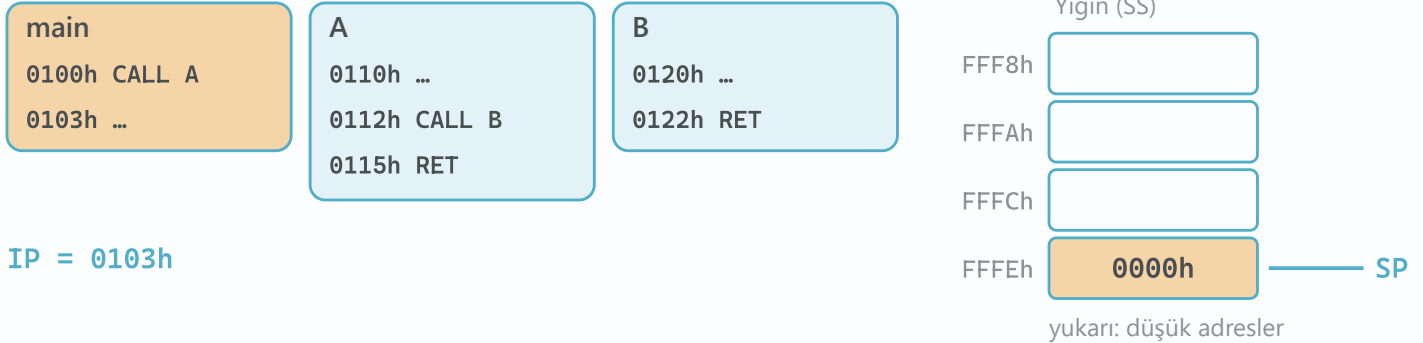
A içinden **CALL B** : 0115h itilir, SP = FFFAh. Yığında iki dönüş adresi var.

İç İçe Çağrılar



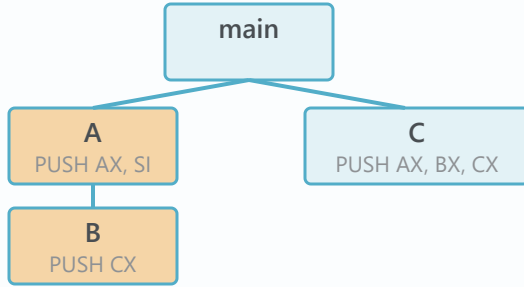
B'nin **RET** 'i en üstteki 0115h'yı alır: A'ya döner (LIFO).

İç İçe Çağrılar



A'nın **RET** 'i 0103h'yı alır: main'e döner. Son çağrılan ilk döner.

Yığının İhtiyacı Hesabı



- Bir programın yığın gereksinimi, çağrı ağacındaki **en derin yol** boyunca biriken baytların toplamıdır.
- Her yakın CALL 2 bayt dönüş adresi, her PUSH 2 bayt ekler.

Yol	Hesap	Bayt
main → A → B	$2 + 4 + 2 + 2$	10
main → C	$2 + 6$	8

- SP = FFFEH ile başlayan programda SP en çok FFF4h'ye iner.
- A ve C hiçbir zaman aynı anda yığında değildir: C çağrıldığında A'nın baytları çoktan geri alınmıştır.

Kesmeler de yığını kullanır (bayraklar, CS ve IP için 6 bayt): bir kesme yığının en derin anında gelebileceği için hesaplanan değerin üstünde pay bırakılır.

CALL / RET Simülatörü

CALL düğmeleri prosedür çağırır, "komut yürüt" IP'yi ilerletir, RET dönüşü yapar; yığındaki dönüş adresleri ve SP her adımda görünür.

CALL yaz (0200h)

CALL topla (0300h)

komut yürüt

RET

sıfırla

IP = 0100h

SP = FFFEh

konum: main

Program 0100h'da başladı. DOS yığına 0000h itti.

FFF4h

FFF6h

FFF8h

FFFAh

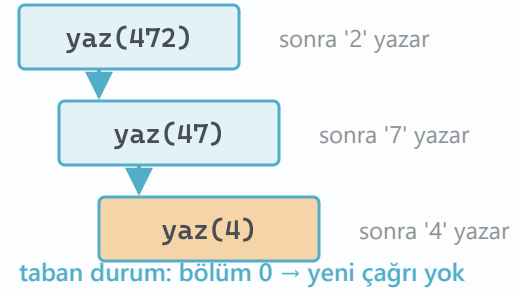
FFFC h

FFFEh 0000h ← SP

Simülasyon bir CALL ile başlar.

Özyineleme

- **Özyineleme:** bir prosedürün kendini (doğrudan ya da başka bir prosedür üzerinden) çağırmasıdır.
- Her çağrı yığına yeni bir dönüş adresi ve kendi sakladığı değerleri koyar: her düzeyin değerleri birbirinden ayrı kalır.
- **Taban durum** (durma koşulu) şarttır; yoksa her çağrı 2 bayt daha yer ve yığın taşar.
- Problem her adımda küçülerek kendine benzer: 472'yi yazmak = 47'yi yazmak + '2' yazmak.
- Sayı yazdırma, faktöriyel, ağaç ve dizin dolaşma özyinelemenin tipik örnekleridir.



Kod İzleme: Özyinelemeli Sayı Yazdırma

```

1  ORG 100h
▶ MOV AX, 472
▶ CALL yaz
4  RET
5  yaz PROC          ; Girdi: AX (işaretsiz)
6  XOR DX, DX
7  MOV BX, 10
8  DIV BX            ; AX = bölüm, DX = basamak
9  PUSH DX
10 OR AX, AX         ; başka basamak var mı?
11 JZ son
12 CALL yaz          ; önce üst basamaklar
13 son: POP AX
14 ADD AL, '0'
15 MOV AH, 0Eh
16 INT 10h
17 RET
18 yaz ENDP

```

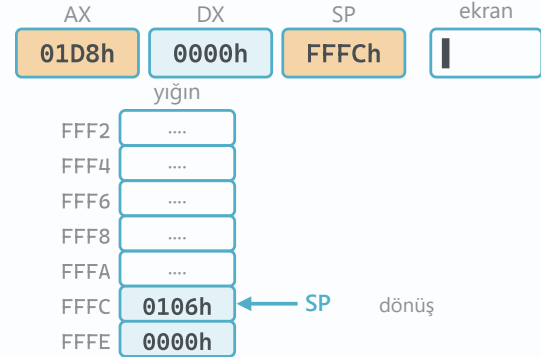
DEĞİŞKENLER

AX = 01D8h

DX = 0000h

SP = FFFCh

ÇIKTI



AX = 472 (01D8h); `CALL yaz` : dönüş adresi 0106h yığına.

Kod İzleme: Özyinelemeli Sayı Yazdırma

```

1  ORG 100h
2  MOV AX, 472
3  CALL yaz
4  RET
5  yaz PROC          ; Girdi: AX (işaretsiz)
▶  XOR DX, DX
▶  MOV BX, 10
▶  DIV BX          ; AX = bölüm, DX = basamak
▶  PUSH DX
10 OR  AX, AX      ; başka basamak var mı?
11 JZ  son
12 CALL yaz        ; önce üst basamaklar
13 son: POP AX
14 ADD AL, '0'
15 MOV AH, 0Eh
16 INT 10h
17 RET
18 yaz ENDP

```

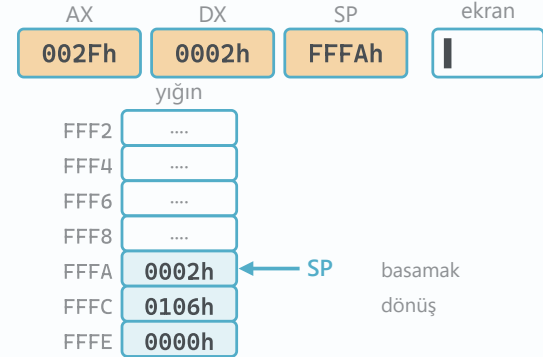
DEĞİŞKENLER

AX = 002Fh

DX = 0002h

SP = FFFAh

ÇIKTI



472 / 10: AX = 47 (2Fh), DX = 2. Son basamak (2) yığına itilir.

Kod İzleme: Özyinelemeli Sayı Yazdırma

```

1  ORG 100h
2  MOV AX, 472
3  CALL yaz
4  RET
5  yaz PROC                ; Girdi: AX (işaretsiz)
6  XOR DX, DX
7  MOV BX, 10
8  DIV BX                  ; AX = bölüm, DX = basamak
9  PUSH DX
10 OR AX, AX              ; başka basamak var mı?
11 JZ son
12 CALL yaz              ; önce üst basamaklar
13 son: POP AX
14 ADD AL, '0'
15 MOV AH, 0Eh
16 INT 10h
17 RET
18 yaz ENDP

```

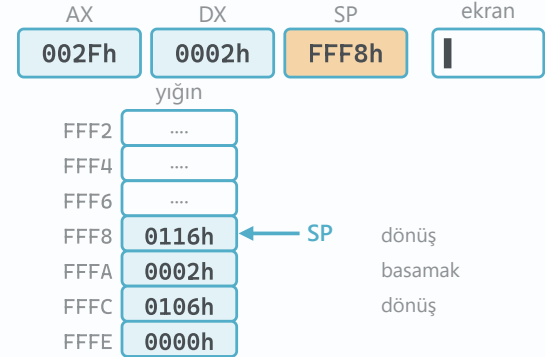
DEĞİŞKENLER

AX = 002Fh

DX = 0002h

SP = FFF8h

ÇIKTI



AX ≠ 0: yaz kendini çağırır; dönüş adresi 0116h (son etiketi) yığına.

Kod İzleme: Özyinelemeli Sayı Yazdırma

```

1  ORG 100h
2  MOV AX, 472
3  CALL yaz
4  RET
5  yaz PROC                ; Girdi: AX (işaretsiz)
▶  XOR DX, DX
▶  MOV BX, 10
▶  DIV BX                  ; AX = bölüm, DX = basamak
▶  PUSH DX
▶  OR AX, AX               ; başka basamak var mı?
▶  JZ son
▶  CALL yaz                ; önce üst basamaklar
13 son: POP AX
14 ADD AL, '0'
15 MOV AH, 0Eh
16 INT 10h
17 RET
18 yaz ENDP

```

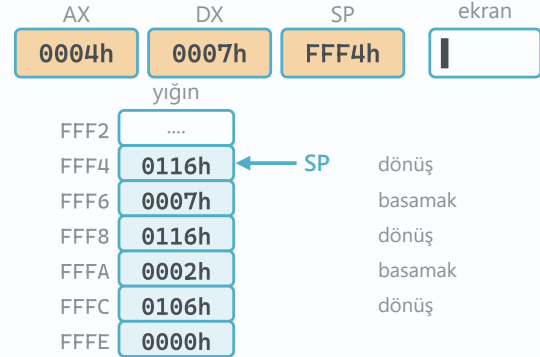
DEĞİŞKENLER

AX = 0004h

DX = 0007h

SP = FFF4h

ÇIKTI



İkinci düzey: $47 / 10 \rightarrow AX = 4$, basamak 7 yığına; yeniden CALL.

Kod İzleme: Özyinelemeli Sayı Yazdırma

```

1  ORG 100h
2  MOV AX, 472
3  CALL yaz
4  RET
5  yaz PROC                ; Girdi: AX (işaretsiz)
▶  XOR DX, DX
▶  MOV BX, 10
▶  DIV BX                  ; AX = bölüm, DX = basamak
▶  PUSH DX
▶  OR AX, AX               ; başka basamak var mı?
▶  JZ son
12 CALL yaz                ; önce üst basamaklar
13 son: POP AX
14 ADD AL, '0'
15 MOV AH, 0Eh
16 INT 10h
17 RET
18 yaz ENDP

```

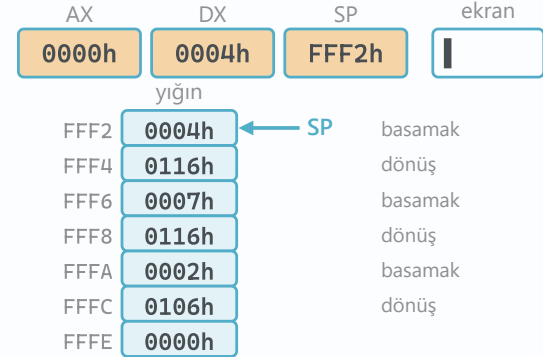
DEĞİŞKENLER

AX = 0000h

DX = 0004h

SP = FFF2h

ÇIKTI



Üçüncü düzey: $4 / 10 \rightarrow AX = 0$, basamak 4 yığına. $AX = 0$: taban durum, **JZ son**. SP en derin noktada: FFF2h.

Kod İzleme: Özyinelemeli Sayı Yazdırma

```

1  ORG 100h
2  MOV AX, 472
3  CALL yaz
4  RET
5  yaz PROC          ; Girdi: AX (işaretsiz)
6  XOR DX, DX
7  MOV BX, 10
8  DIV BX          ; AX = bölüm, DX = basamak
9  PUSH DX
10 OR AX, AX      ; başka basamak var mı?
11 JZ son
12 CALL yaz      ; önce üst basamaklar
    ► son: POP AX
    ► ADD AL, '0'
    ► MOV AH, 0Eh
    ► INT 10h
17 RET
18 yaz ENDP

```

DEĞİŞKENLER

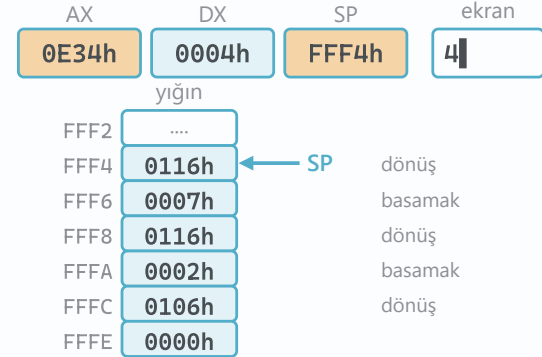
AX = 0E34h

DX = 0004h

SP = FFF4h

ÇIKTI

4



POP AX = 4 → '4' yazılır. En son itilen basamak en önce çıkar.

Kod İzleme: Özyinelemeli Sayı Yazdırma

```

1  ORG 100h
2  MOV AX, 472
3  CALL yaz
4  RET
5  yaz PROC                ; Girdi: AX (işaretsiz)
6  XOR DX, DX
7  MOV BX, 10
8  DIV BX                ; AX = bölüm, DX = basamak
9  PUSH DX
10 OR AX, AX            ; başka basamak var mı?
11 JZ son
12 CALL yaz            ; önce üst basamaklar
    ► son: POP AX
    ► ADD AL, '0'
    ► MOV AH, 0Eh
    ► INT 10h
    ► RET
18 yaz ENDP

```

DEĞİŞKENLER

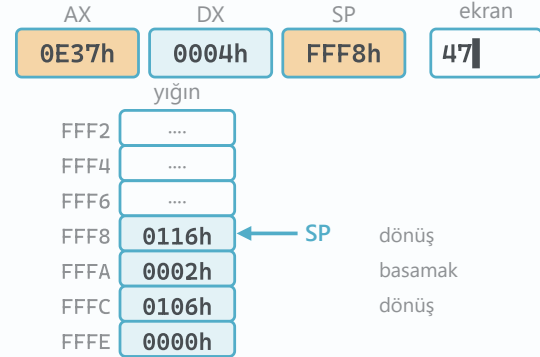
AX = 0E37h

DX = 0004h

SP = FFF8h

ÇIKTI

47



RET 0116h'ye, yani ikinci düzeyin **son** satırına döner:
POP AX = 7 → '7'.

Kod İzleme: Özyinelemeli Sayı Yazdırma

```

1  ORG 100h
2  MOV AX, 472
3  CALL yaz
4  RET
5  yaz PROC          ; Girdi: AX (işaretsiz)
6  XOR DX, DX
7  MOV BX, 10
8  DIV BX            ; AX = bölüm, DX = basamak
9  PUSH DX
10 OR  AX, AX        ; başka basamak var mı?
11 JZ  son
12 CALL yaz          ; önce üst basamaklar
    ► son: POP AX
    ► ADD AL, '0'
    ► MOV AH, 0Eh
    ► INT 10h
    ► RET
18 yaz ENDP

```

DEĞİŞKENLER

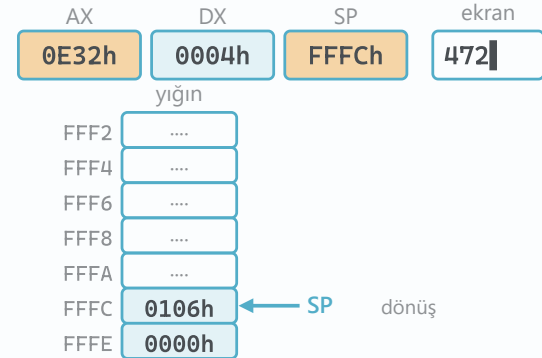
AX = 0E32h

DX = 0004h

SP = FFFCh

ÇIKTI

472



Birinci düzey: POP AX = 2 → '2'. Ekranda 472.

Kod İzleme: Özyinelemeli Sayı Yazdırma

```

1  ORG 100h
2  MOV AX, 472
3  CALL yaz
▶ RET
5  yaz PROC                ; Girdi: AX (işaretsiz)
6  XOR DX, DX
7  MOV BX, 10
8  DIV BX                ; AX = bölüm, DX = basamak
9  PUSH DX
10 OR AX, AX            ; başka basamak var mı?
11 JZ son
12 CALL yaz            ; önce üst basamaklar
13 son: POP AX
14 ADD AL, '0'
15 MOV AH, 0Eh
16 INT 10h
▶ RET
18 yaz ENDP

```

DEĞİŞKENLER

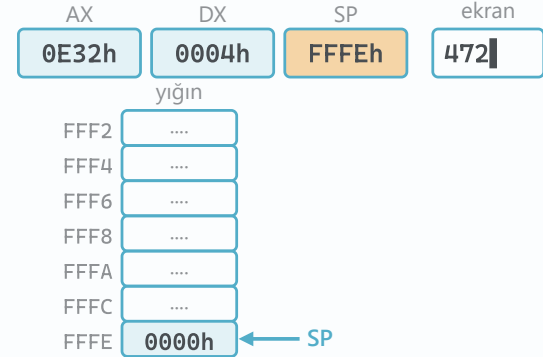
AX = 0E32h

DX = 0004h

SP = FFEh

ÇIKTI

472



Son RET 0106h'ye döner; SP = FFEh. Her düzey kendi basamağını yığında sakladığı için ek değişken gerekmedi.

ÖZYİNELEME

Özyineleme mi, Döngü mü?

Ölçüt	Özyineleme (yaz)	Döngü (bölme + LOOP)
Kod	kısa; problemin tanımına yakın	sayaç ve ayrı bir yazdırma döngüsü gerekir
Yığın	her düzeyde dönüş adresi + basamak: 4 bayt	basamak başına 2 bayt
Süre	her düzeyde CALL + RET (19 + 8 tur)	çağrı yükü yok
Risk	derinlik sınırsız büyürse yığın taşar	yığın kullanımı sınırlı

Derinlik hesabı

yaz için düzey sayısı = basamak sayısı. 65 535
→ 5 düzey: 5×2 (dönüş) + 5×2 (basamak) =
20 bayt; SP en çok FFEAh'ye iner.

Taban durumun yeri

Denetim CALL'dan önce yapılmalıdır. AX = 0
için de bir basamak (0) yazılır: 0 sayısı ek kod
gerektirmeden '0' olarak çıkar.

Kod İzleme: PUSH / POP Dengesizliği

```

1 ORG 100h
▶ MOV AX, 1234h      ; 0100h: B8 34 12
3 CALL hatali        ; 0103h: dönüş = 0106h
4 RET                ; 0106h
5 hatali PROC
6 PUSH AX             ; AX saklandı...
7 ; ... işlemler ...
8 RET                ; POP AX unutuldu!
9 hatali ENDP
    
```

DEĞİŞKENLER

AX = 1234h

IP = 0103h

SP = FFEh

ÇAĞRI YIĞINI

[FFEh] 0000h

AX = 1234h.

Adım 1 / 4

Kod İzleme: PUSH / POP Dengesizliği

```

1 ORG 100h
2 MOV  AX, 1234h      ; 0100h: B8 34 12
3 ► CALL hatali       ; 0103h: dönüş = 0106h
4 RET                 ; 0106h
5 hatali PROC
6 PUSH AX             ; AX saklandı...
7 ; ... işlemler ...
8 RET                 ; POP AX unutuldu!
9 hatali ENDP
    
```

DEĞİŞKENLER

AX = 1234h

IP = hatali

SP = FFFCh

ÇAĞRI YIĞINI

[FFFCh] 0106h

[FFFEh] 0000h

CALL: dönüş adresi 0106h itilir.

Adım 2 / 4

Kod İzleme: PUSH / POP Dengesizliği

```
1 ORG 100h
2 MOV AX, 1234h      ; 0100h: B8 34 12
3 CALL hatali        ; 0103h: dönüş = 0106h
4 RET                ; 0106h
5 hatali PROC
▶ 6 PUSH AX           ; AX saklandı...
7 ; ... işlemler ...
8 RET                ; POP AX unutuldu!
9 hatali ENDP
```

DEĞİŞKENLER

AX = 1234h

IP = hatali

SP = FFFAh

ÇAĞRI YIĞINI

[FFFAh] 1234h

[FFFCh] 0106h

[FFFEh] 0000h

PUSH AX : 1234h, dönüş adresinin **üstüne** itilir.

Kod İzleme: PUSH / POP Dengesizliği

```

1 ORG 100h
2 MOV  AX, 1234h      ; 0100h: B8 34 12
3 CALL hatali         ; 0103h: dönüş = 0106h
4 RET                 ; 0106h
5 hatali PROC
6 PUSH AX              ; AX saklandı...
7 ; ... işlemler ...
8 ► RET                ; POP AX unutuldu!
9 hatali ENDP
    
```

DEĞİŞKENLER

AX = **1234h**

IP = **1234h**

SP = FFFCh

ÇAĞRI YIĞINI

[FFFCh] 0106h

[FFFEh] 0000h

RET yığının tepesindekini alır: IP = **1234h**!
İşlemci rastgele bir adrese atlar, program çöker ya da öngörülemez davranır.

Kod İzleme: Yakın CALL, Uzak Dönüş

```

1  ORG 100h
▶ 2  MOV AX, 5
3  CALL uzak          ; yakın CALL: yalnız IP
4  ADD AX, 1
5  RET
6  uzak PROC
7  SHL AX, 1
8  RETF              ; IP ve CS çeker!
9  uzak ENDP
    
```

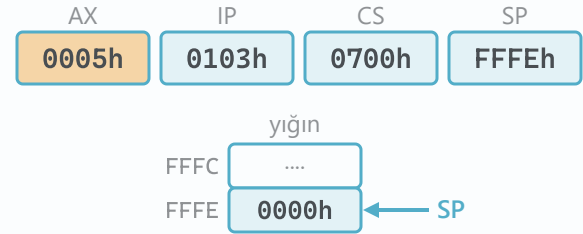
DEĞİŞKENLER

AX = 0005h

IP = 0103h

CS = 0700h

SP = FFFEh



AX = 5. uzak prosedürü RETF ile bitiyor, ama çağrı yakın.

Kod İzleme: Yakın CALL, Uzak Dönüş

```

1  ORG  100h
2  MOV  AX, 5
3  ► CALL uzak      ; yakın CALL: yalnız IP
4  ADD  AX, 1
5  RET
6  uzak PROC
7  SHL  AX, 1
8  RETF              ; IP ve CS çeker!
9  uzak ENDP
    
```

DEĞİŞKENLER

AX = 0005h

IP = 010Ah

CS = 0700h

SP = FFFCh

AX	IP	CS	SP
0005h	010Ah	0700h	FFFCh

	yığın	
FFFC	0106h	← SP
FFFE	0000h	

Yakın **CALL** yalnızca IP'yi (0106h) iter: yığında 2 bayt.

Kod İzleme: Yakın CALL, Uzak Dönüş

```

1  ORG 100h
2  MOV AX, 5
3  CALL uzak ; yakın CALL: yalnız IP
4  ADD AX, 1
5  RET
6  uzak PROC
7  ► SHL AX, 1
8  RTF ; IP ve CS çeker!
9  uzak ENDP
    
```

DEĞİŞKENLER

AX = 000Ah

IP = 010Ch

CS = 0700h

SP = FFFCh



SHL AX, 1 : AX = 000Ah.

Kod İzleme: Yakın CALL, Uzak Dönüş

```

1  ORG 100h
2  MOV AX, 5
3  CALL uzak          ; yakın CALL: yalnız IP
4  ADD AX, 1
5  RET
6  uzak PROC
7  SHL AX, 1
8  RET                ; IP ve CS çeker!
9  uzak ENDP
    
```

DEĞİŞKENLER

AX = 000Ah

IP = 0106h

CS = 0000h

SP = 0000h

AX	IP	CS	SP
000Ah	0106h	0000h	0000h

	yığın
FFFC	
FFFE	

RET iki sözcük çeker: IP = 0106h ve CS olarak yığında bulunduğu değer, burada DOS'un .COM için ittiği 0000h. SP = 0000h.

Kod İzleme: Yakın CALL, Uzak Dönüş

```

1  ORG 100h
2  MOV AX, 5
3  CALL uzak          ; yakın CALL: yalnız IP
4  ADD AX, 1
5  RET
6  uzak PROC
7  SHL AX, 1
8  RET                ; IP ve CS çeker!
9  uzak ENDP
    
```

DEĞİŞKENLER

AX = 000Ah

IP = 0106h

CS = 0000h

SP = 0000h

AX	IP	CS	SP
000Ah	0106h	0000h	0000h

	yığın
FFFC	
FFFE	

İşlemci 0000:0106h'ye atlar: burası kesme vektör tablosudur, program kodu değil. Program çöker; **ADD AX, 1** hiç çalışmaz.

Sık Yapılan Hatalar

RET'i unutmak

İşlemci ENDP'yi görmez; prosedürden sonraki kod ya da veri yürütülür.

PUSH / POP dengesizliği

RET yanlış değeri IP'ye alır. Her PUSH'un bir POP'u olmalı ve ters sırada yapılmalı.

Prosedüre "düşmek"

Prosedür ana kodun ortasına konup üzerinden atlanmazsa CALL edilmeden yürütülür; sonundaki RET programı bitirir.

CALL yerine JMP

JMP dönüş adresi itmez; prosedürün RET'i yığındaki ilgisiz bir değere döner.

Ad uyuşmazlığı

p1 PROC ... p2 ENDP → assembler hatası.

Sonsuz özyineleme

Kendini çağıran prosedürde durma koşulu yoksa her çağrı 2 bayt yer, yığın taşar.

Daha Fazla Prosedür Hatası

Erken RET

PUSH'tan sonra koşullu bir RET, saklanan değeri dönüş adresi sanır. Bütün yollar POP'ların bulunduğu tek çıkış etiketine gider.

LEA yerine MOV

`MOV SI, dizi` adresi değil dizinin ilk sözcüğünü yükler; prosedür ilgisiz bir adresten okur.

Gizlice bozulan DX

16 bitlik MUL ve DIV DX'i de yazar. Açıklamasında "Bozar: DX" yazmayan prosedür çağırının DX'ini siler.

Tür uyuşmazlığı

Yakın CALL + RETF ya da CALL FAR + RET: yığın 2 bayt kayar, dönüş yanlış adrese yapılır.

Bayrağı bozmak

Sonucu CF ile döndüren prosedürde STC/CLC'den sonra ADD, CMP gibi bir komut CF'yi değiştirir. POP ve RET bayraklara dokunmaz.

Taban durumu geç denetlemek

Özyinelemeli prosedürde durma denetimi CALL'dan sonra yapılırsa çağrılar hiç durmaz ve yığın taşar.

ÖZET

Özet

Prosedür

Belirli bir görevi yapan, CALL ile çağrılıp RET ile dönen kod parçası.

PROC / ENDP

Direktif; makine kodu üretmez. CALL ve RET ise işlemci komutudur.

CALL

Sonraki komutun adresini yığına iter ($SP -= 2$), $IP \leftarrow \text{hedef}$.

RET

$IP \leftarrow [SP]$, $SP += 2$. .COM'da son RET 0000h'ya $\rightarrow INT 20h$.

Parametreler

Yazmaç, bellek değişkeni, adres (SI/DI/BX) veya yığın: $[BP+4]$, $[BP+6]$... ile okunur, RET n ile temizlenir.

Güvenlik

Yazmaçlar PUSH/POP ile korunur, yığın dengede tutulur, girdi/çıktı belgelenir.

ÖZET

Alıştırmalar

1. 0120h adresindeki `CALL p` komutu 0100h'daki p'yi çağırıyor. Makine kodu nedir? Yığına ne itilir?
2. m2 örneğinde ikinci `CALL m2` 'den önce `MOV AL, 3` eklenseydi son AX ne olurdu?
3. SP = FFFEh iken art arda 3 CALL yapılıyor (hiç RET yok). SP kaç olur?
4. CX'i bozan bir prosedürü LOOP içinde çağırarak neden hatalıdır? İki çözüm önerin.
5. AL'deki küçük harfi büyük harfe çeviren `buyuk` prosedürünü yazın (girdi/çıktı AL).
6. print_me'yi, yazdığı karakter sayısını CX'te döndürecek şekilde değiştirin.

► İpuçları / cevaplar

Kaynaklar

- emu8086 belgeleri: *8086 Assembler Tutorial for Beginners* (Part 8: Procedures).
- Intel, *The 8086 Family User's Manual*, 1979: CALL, RET ve yığın işlemleri.
- Barry B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson, 2009: Bölüm 6.3 "Procedures".
- Kip R. Irvine, *Assembly Language for x86 Processors*, 7. baskı, Pearson, 2014: Bölüm 5 "Procedures".

Son

Bölüm 8: Prosedürler

Sonraki bölüm: Bölüm 9: Makrolar · [Terim sözlüğü](#)