

Bölüm 7: Prosedürler

Mikroişlemciler

Prosedür kavramı

PROC / ENDP

CALL ve RET

Yığın ve dönüş adresi

Parametre geçirme

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 ve bellek değişkenleriyle
prosedüre veri gönderip sonuç almak.

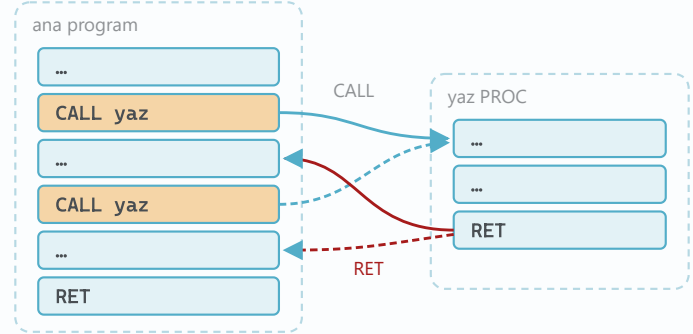
Güvenli prosedür

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.

Not: CALL ve RET derleyici direktifi değil, işlemcinin yürüttüğü **komutlardır** (CALL near: E8, RET: C3). Yalnızca PROC, ENDP ve END direktiftir.

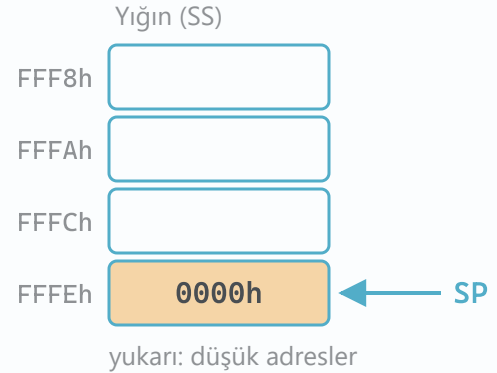
derleme: PROC · ENDP · END → bayt yok

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.

Yığın Hatırlatma

- Yığın, **SS:SP** ile gösterilen ve LIFO çalışan bir bellek bölgesidir; **aşağı doğru** (düşük adreslere) büyür.
- **PUSH kaynak** : $SP \leftarrow SP - 2$, sonra 16 bitlik değer $[SS:SP]$ 'ye yazılır.
- **POP hedef** : $[SS:SP]$ 'deki değer okunur, sonra $SP \leftarrow SP + 2$.
- 8086'da yığına her zaman **sözcük** (2 bayt) itilir.
- .COM programı başladığında $SP = FFFEh$ ve yığının tepesinde DOS'un ittiği **0000h** vardır.
- CALL ve RET, dönüş adresini saklamak için bu yığını kullanır.



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

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
FFFEh	0000h

SP

yukarı: düşük adresler

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

Adım 3 / 6

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.

Adım 4 / 6

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

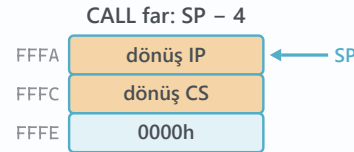
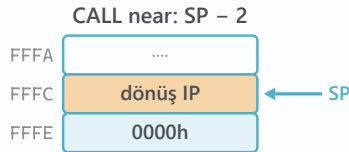
MOV AX, 2 → IP = 0106h. Buradaki RET 0000h'yı alır ve programı sonlandırır.

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. Bu bölümdeki örneklerin hepsi NEAR'dır.



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

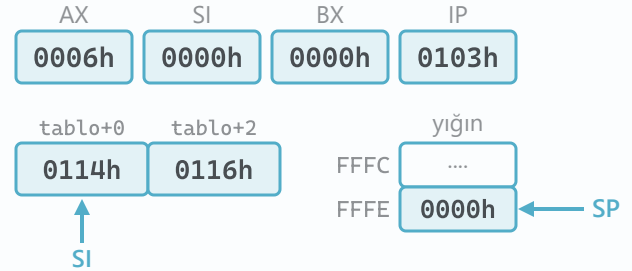
```

DEĞİŞKENLER

AX = 6

IP = 0103h

SP = FFFEh



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

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

Kod İzleme: Tablo ve Yazmaçla Dolaylı CALL

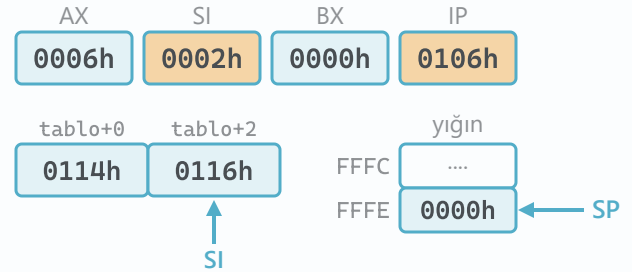
```
1 MOV AX, 6
2 MOV SI, 2 ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 MOV BX, OFFSET bir_ekle
5 CALL BX ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 bir_ekle: INC AX
9 RET
10 iki_kat: SHL AX, 1
11 RET
```

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

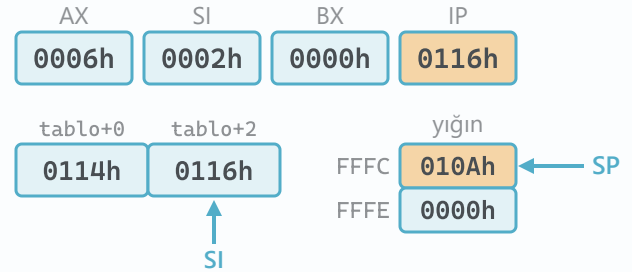
```

DEĞİŞKENLER

AX = 6

IP = 0116h

SP = FFFCh



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

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

Kod İzleme: Tablo ve Yazmaçla Dolaylı CALL

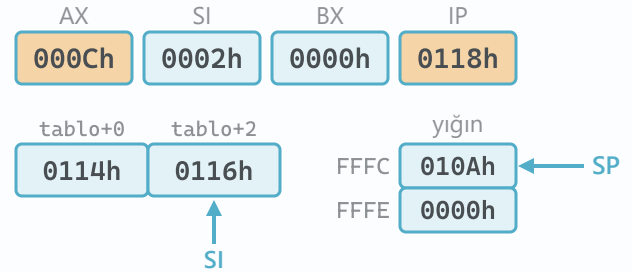
```
1 MOV AX, 6
2 MOV SI, 2 ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 MOV BX, OFFSET bir_ekle
5 CALL BX ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 bir_ekle: INC AX
9 RET
► iki_kat: SHL AX, 1
11 RET
```

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

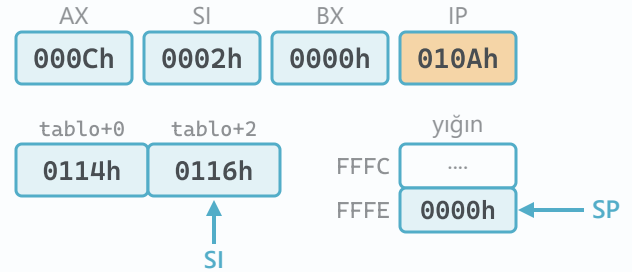
```
1 MOV AX, 6
2 MOV SI, 2 ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 MOV BX, OFFSET bir_ekle
5 CALL BX ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 bir_ekle: INC AX
9 RET
10 iki_kat: SHL AX, 1
▶ RET
```

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

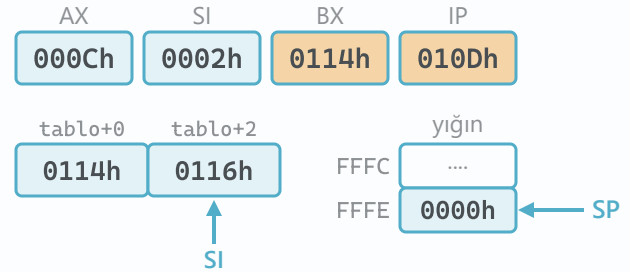
```
1 MOV AX, 6
2 MOV SI, 2 ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 ► MOV BX, OFFSET bir_ekle
5 CALL BX ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 bir_ekle: INC AX
9 RET
10 iki_kat: SHL AX, 1
11 RET
```

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

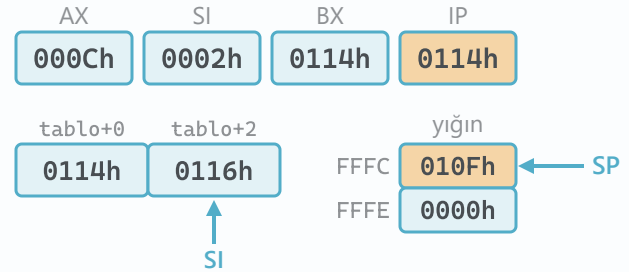
```
1 MOV AX, 6
2 MOV SI, 2 ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 MOV BX, OFFSET bir_ekle
5 ► CALL BX ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 bir_ekle: INC AX
9 RET
10 iki_kat: SHL AX, 1
11 RET
```

DEĞİŞKENLER

AX = 12

IP = 0114h

SP = FFFCh



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

Kod İzleme: Tablo ve Yazmaçla Dolaylı CALL

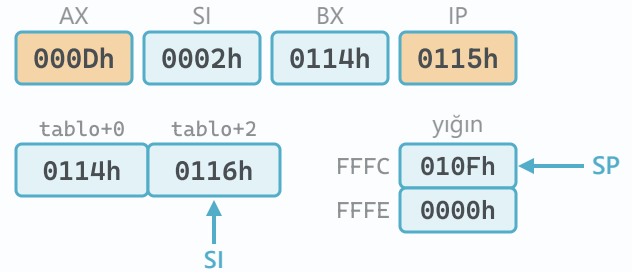
```
1 MOV AX, 6
2 MOV SI, 2 ; 2. işlem (× 2)
3 CALL tablo[SI] ; bellekle dolaylı
4 MOV BX, OFFSET bir_ekle
5 CALL BX ; yazmaçla dolaylı
6 RET
7 tablo DW bir_ekle, iki_kat
8 ▶ bir_ekle: INC AX
9 RET
10 iki_kat: SHL AX, 1
11 RET
```

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

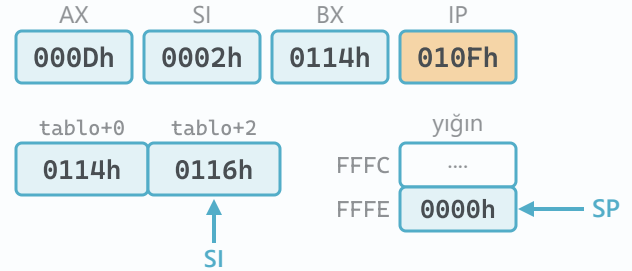
```

DEĞİŞKENLER

AX = 13

IP = 010Fh

SP = FFFEh



bir_ekle = 0114h · iki_kat = 0116h

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

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

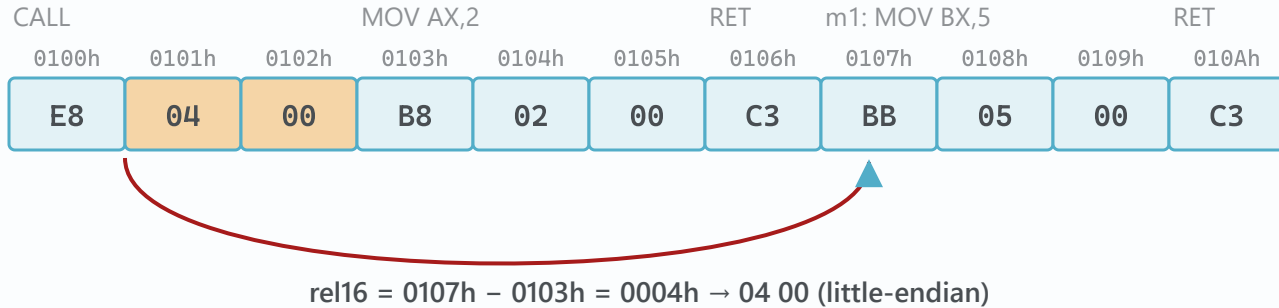
BX = 0005h

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.

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

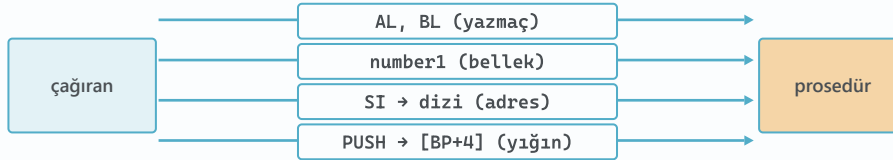
Prosedr `number1`, `result` gibi ortak deėiřkenleri okur/yazar. Kolay ama prosedr o deėiřkenlere baėımlı kalır.

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.

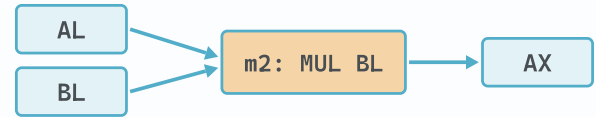
PARAMETRELER

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 işaretli 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 YİĞİNİ

[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 YİĞİNİ

[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 YIĞINI

[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

[FFFCh] 010Ah

[FFFEh] 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 YIĞINI

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; çağırın 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

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

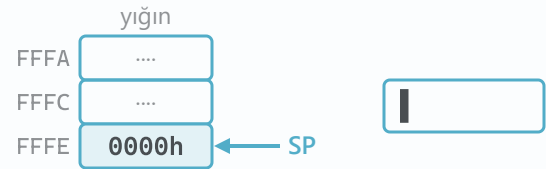
```
► MOV CX, 2 ; ana döngü: 2 satır
2 satir: CALL yildiz
3 LOOP satir
4 RET
5 yildiz PROC
6 PUSH CX ; çağıranın sayacı
7 MOV CX, 3 ; iç sayaç
8 MOV AX, 0E23h ; AH = 0Eh, AL = '#'
9 ic: INT 10h
10 LOOP ic
11 POP CX ; sayaç geri
12 RET
13 yildiz ENDP
```

DEĞİŞKENLER

CX = 2 SP = FFFEH

ÇIKTI

CX	AX	SP
0002h	0000h	FFFEh



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
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

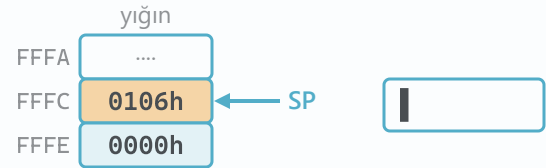
CX = 2

SP = FFFCh

ÇIKTI



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

yığın		
FFFA	0002h	← SP
FFFC	0106h	
FFFE	0000h	

I

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
```

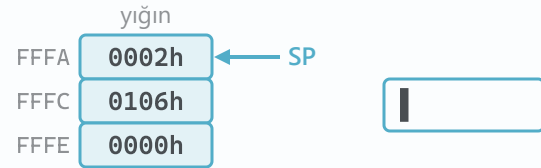
DEĞİŞKENLER

CX = 3

SP = FFFAh

ÇIKTI

CX	AX	SP
0003h	0E23h	FFFAh



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

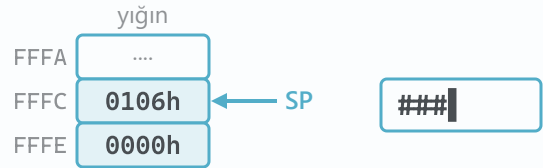
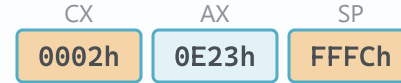
```

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

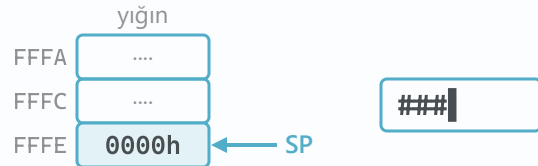
CX = 2

SP = FFFh

ÇIKTI

###

CX	AX	SP
0002h	0E23h	FFh



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
3
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
```

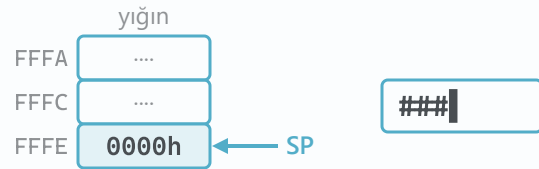
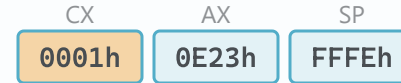
DEĞİŞKENLER

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
```

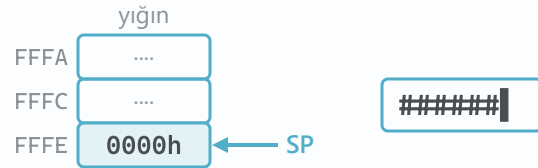
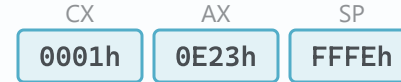
DEĞİŞKENLER

CX = 1

SP = FFEh

ÇIKTI

#####



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

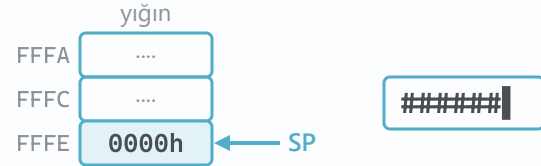
CX = 0

SP = FFFh

ÇIKTI

#####

CX	AX	SP
0000h	0E23h	FFh



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

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 YIĞINI

[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 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**

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 = **FFFCh**

m = **00h**

ÇAĞRI YIĞINI

[FFFCh] 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 YİĞİNİ

[**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:
▶ ADD AL, [SI]
▶ INC SI
▶ 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 topla
5  MOV  m, AL           ; dönüş değeri AL'de
▶ 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+5**

CX = **0000h**

AL = **11h**

SP = **0000h**

m = **11h**

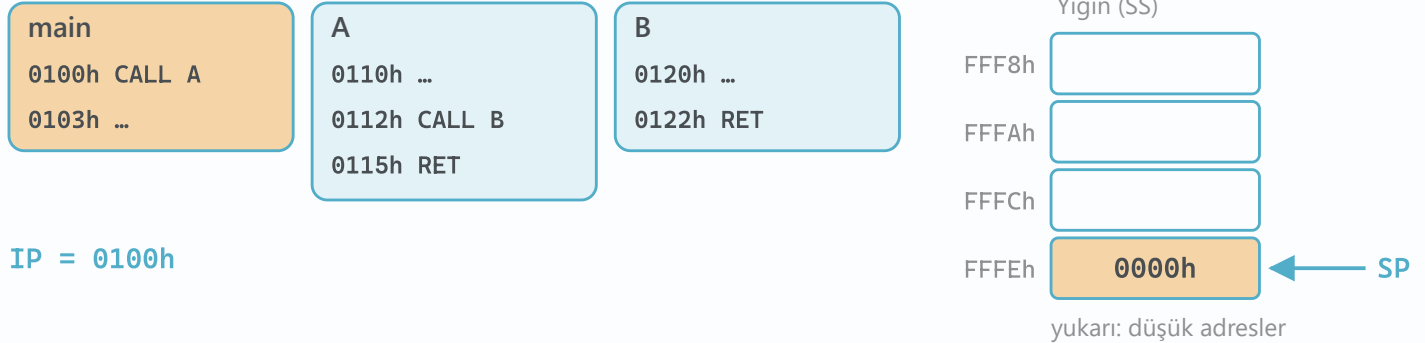
ÇAĞRI YIĞINI

boş

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

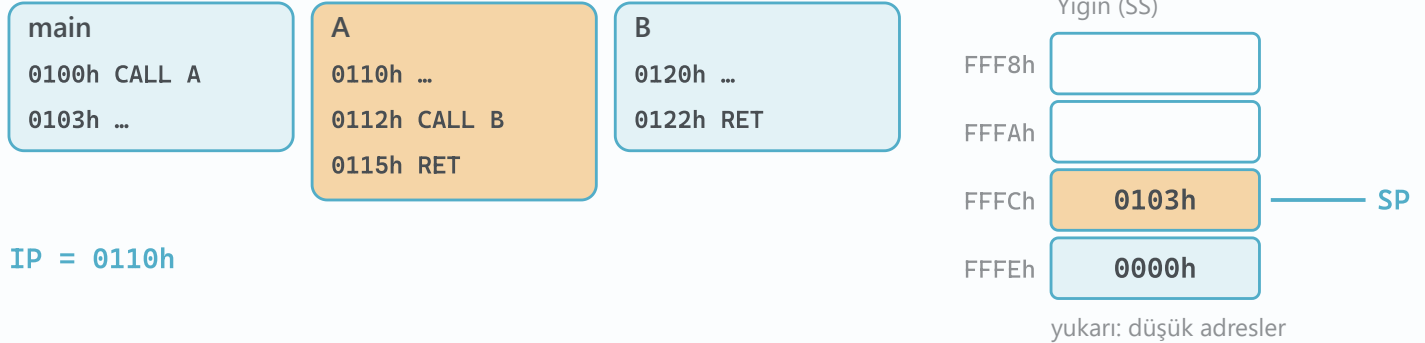
Adım 12 / 12

İç İçe Çağrılar



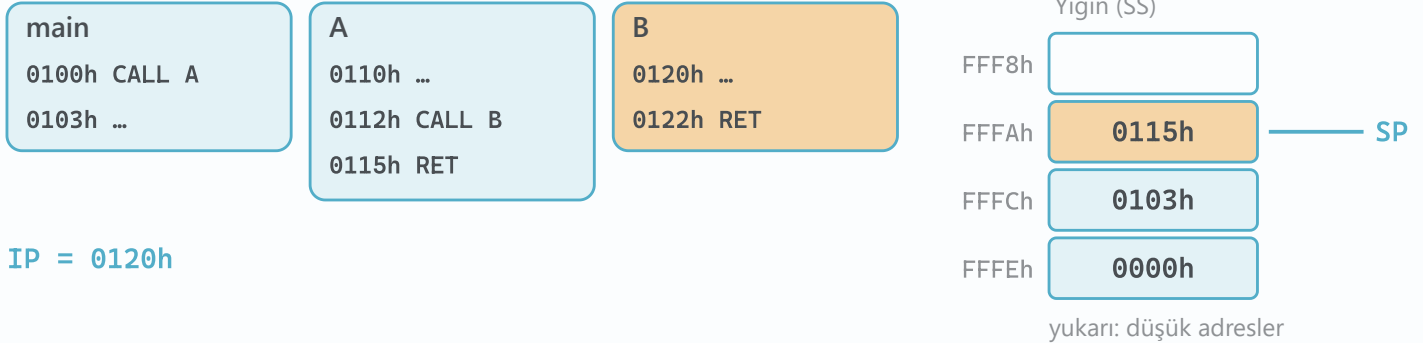
main, A'yı çağırarak üzere. SP = FFFEh.

İç İçe Çağrılar



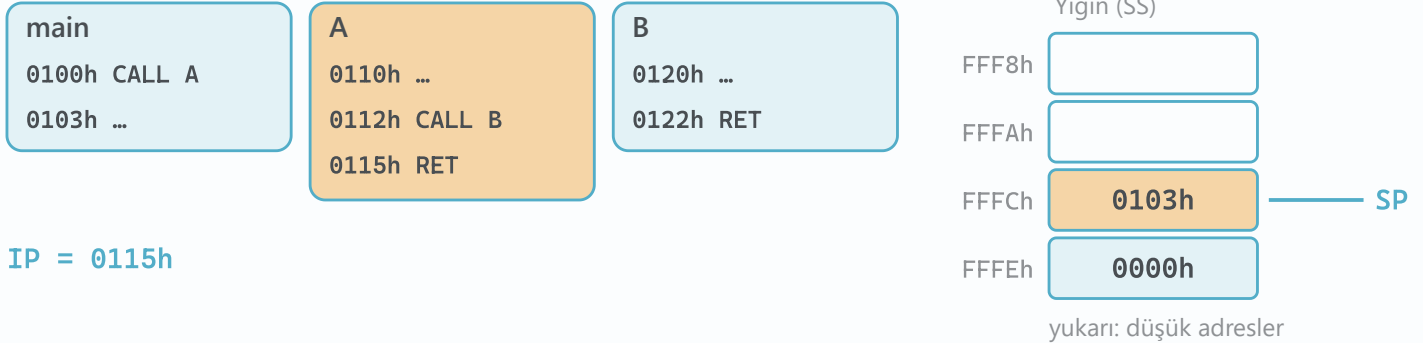
CALL A : 0103h itilir, SP = FFFCh.

İç İçe Çağrılar



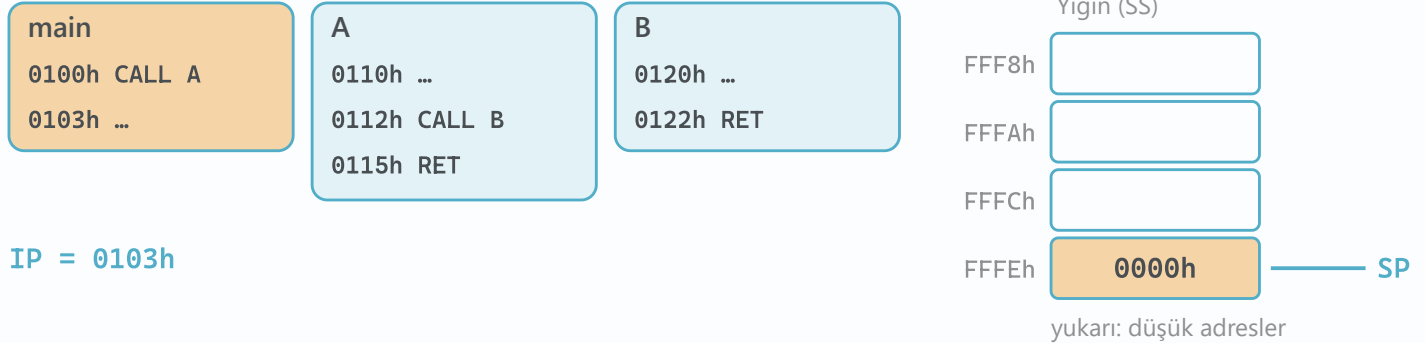
A içinden **CALL B** : 0115h itilir, SP = FFAh. Yığında iki dönüş adresi var.

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.

CALL / RET Simülatörü

CALL ile prosedür çağırın, "komut yürüt" ile IP'yi ilerletin, RET ile dönün. Yığındaki dönüş adreslerini ve SP'yi izleyin.

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

FFFCCh

FFFEh0000h← SP

Bir CALL ile başlayın.

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

[FFFEh] 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.

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
▶ 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.

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.

Prosedür mü, Makro mu?

	Prosedür (CALL)	Makro
Kod	bellette bir kez bulunur	her kullanıldığı yere kopyalanır
Çalışma	CALL / RET ve yığın kullanır	çağrı maliyeti yok, doğrudan yürütölür
Boyut	küçük program	çok kullanılırsa büyük program
Hız	CALL + RET ek süre	daha hızlı
Parametre	yazmaç, bellek, yığın	assembler düzeyinde metin parametreleri

putc , print gibi emu8086.inc öğeleri makrodur; print_string , scan_num gibileri ise DEFINE_... ile eklenen prosedürlerdir ve CALL ile çağrılır.

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.

Güvenlik

Yazmaçları PUSH/POP ile koru, yığını dengede tut, girdi/çıktıyı belgele.

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 7: Prosedürler

Sonraki bölüm: Bölüm 8: Yığın · [Terim sözlüğü](#)