

Bölüm 8: Makrolar

Mikroişlemciler

Makro kavramı

MACRO / ENDM

Genişletme

Parametre tuzakları

Makro ve prosedür

Boyut hesabı

LOCAL

PUTC

POWER

emu8086.inc

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Makro tanımı

`MACRO` / `ENDM` ile parametrelili makro yazmak.

Genişletme

Derleyicinin makro çağrısını nasıl metin olarak açtığını izlemek.

Makro mu, prosedür mü?

Boyut, hız ve kullanım açısından iki yöntemi karşılaştırıp doğru olanı seçmek.

LOCAL

Etiket içeren makrolarda *Duplicate declaration* hatasını önlemek.

Uygulama

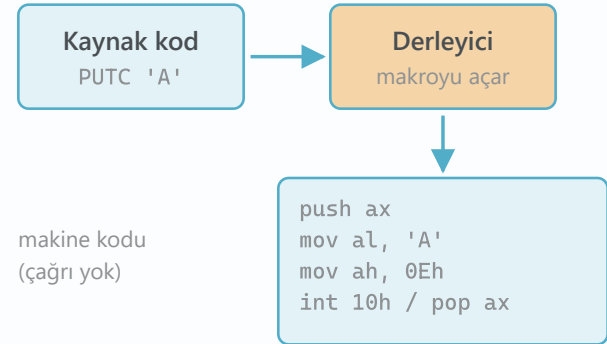
PUTC (ekrana karakter) ve POWER (üs alma) makrolarını adım adım çalıştırmak.

Tuzaklar

Parametre yan etkileri, yazmaç bozulması ve sınır durumlarını fark etmek.

Makrolar

- Makrolar **prosedürlere benzer**: tekrar eden bir kod parçasına ad verir.
- Ancak çalışma zamanında çağrılmazlar; **derleme aşamasında** gerçek komutlarla değiştirilen geçici yapılardır.
- Kod içinde kullanılmayan makrolar görmezden gelinir: makine koduna hiçbir bayt eklemeyiz.
- Makrolar, prosedürlerden farklı olarak, **kullanıldıkları kodun üzerinde** (önce) tanımlanmalıdır; derleyici makroyu tanımadan çağrıyı açamaz.



Makro Tanımlama

```
name    MACRO [parametreler,...]  
        <komutlar>  
        ENDM
```

name

Makronun adı; çağırırken komut gibi yazılır.

parametreler

Virgülle ayrılmış biçimsel adlar (isteğe bağlı). Çağırdaki değerler bunların **yerine metin olarak** konur.

ENDM

Makro gövdesini bitirir. Prosedürdeki gibi ad tekrar yazılmaz.

MyMacro p1, p2, p3 tanım: biçimsel adlar
 ↓ ↓ ↓
MyMacro 4, 5, DX MOV CX, p3 →
 çağrı: yerine konan metin

MOV CX, DX

Parametreler yazmaç, bellek ya da sabit olabilir: derleyici yalnızca metni değiştirir; sonuçta oluşan komutun geçerli olup olmadığına **genişletmeden sonra** bakılır.

Makro Genişletme Adım Adım

```
MyMacro  MACRO  p1, p2, p3
          MOV  AX, p1
          MOV  BX, p2
          MOV  CX, p3
ENDM
```

Önce **tanım**: derleyici MyMacro'yu ve p1, p2, p3 parametrelerini öğrenir; henüz kod üretmez.

Makro Genişletme Adım Adım

MyMacro 1, 2, 3

```
MOV AX, p1  
MOV BX, p2  
MOV CX, p3
```

MyMacro 1, 2, 3 : p1 = 1, p2 = 2, p3 = 3.

Makro Genişletme Adım Adım

MyMacro 1, 2, 3

```
MOV AX, 1  
MOV BX, p2  
MOV CX, p3
```

1. satır: MOV AX, p1 → MOV AX, 1 .

Adım 3 / 10

Makro Genişletme Adım Adım

MyMacro 1, 2, 3

```
MOV AX, 1  
MOV BX, 2  
MOV CX, p3
```

2. satır: MOV BX, p2 → MOV BX, 2 .

Makro Genişletme Adım Adım

MyMacro 1, 2, 3

```
MOV AX, 1  
MOV BX, 2  
MOV CX, 3
```

3. satır: `MOV CX, p3` → `MOV CX, 3` .

Makro Genişletme Adım Adım

MyMacro 4, 5, DX

```
MOV AX, p1  
MOV BX, p2  
MOV CX, p3
```

MyMacro 4, 5, DX : p1 = 4, p2 = 5, p3 = DX.

Makro Genişletme Adım Adım

MyMacro 4, 5, DX

```
MOV AX, 4  
MOV BX, p2  
MOV CX, p3
```

1. satır: MOV AX, p1 → MOV AX, 4 .

Makro Genişletme Adım Adım

MyMacro 4, 5, DX

```
MOV AX, 4  
MOV BX, 5  
MOV CX, p3
```

2. satır: MOV BX, p2 → MOV BX, 5 .

Makro Genişletme Adım Adım

MyMacro 4, 5, DX

```
MOV AX, 4  
MOV BX, 5  
MOV CX, DX
```

3. satır: MOV CX, p3 → MOV CX, DX .

Makro Genişletme Adım Adım

```
MOV AX, 00001h  
MOV BX, 00002h  
MOV CX, 00003h  
MOV AX, 00004h  
MOV BX, 00005h  
MOV CX, DX  
RET
```

Sonuç (emu8086'nın gösterdiği gibi): iki çağrı **6 komuta** açıldı. Sabitler onaltılık olarak gösterilir.

Açılmış Kodu Çalıştırmak

► **ORG 100h**

```
2 MOV AX, 00001h ; MyMacro 1, 2, 3
3 MOV BX, 00002h
4 MOV CX, 00003h
5 MOV AX, 00004h ; MyMacro 4, 5, DX
6 MOV BX, 00005h
7 MOV CX, DX
8 RET
```

Yükleme sonrası CX = **0012h**: programın boyutu $5 \times 3 + 2 + 1 = 18$ bayt. SP = FFEh.

Adım 1 / 8

DEĞİŞKENLER

AX = 0000h

BX = 0000h

CX = 0012h

DX = 0000h

SP = FFEh

SP hiç değişmedi: makro yığın kullanmaz (gövdesinde PUSH/POP yoksa). Prosedürde CALL/RET SP'yi değiştirirdi.

Açılmış Kodu Çalıştırmak

```
1 ORG 100h
2 MOV AX, 00001h ; MyMacro 1, 2, 3
3 MOV BX, 00002h
4 MOV CX, 00003h
5 MOV AX, 00004h ; MyMacro 4, 5, DX
6 MOV BX, 00005h
7 MOV CX, DX
8 RET
```

AX = 0001h.

Adım 2 / 8

DEĞİŞKENLER

AX = 0001h

BX = 0000h

CX = 0012h

DX = 0000h

SP = FFFeh

SP hiç değişmedi: makro yığın kullanmaz (gövdesinde PUSH/POP yoksa). Prosedürde CALL/RET SP'yi değiştirirdi.

Açılmış Kodu Çalıştırmak

```
1 ORG 100h
2 MOV AX, 00001h ; MyMacro 1, 2, 3
3 MOV BX, 00002h
4 MOV CX, 00003h
5 MOV AX, 00004h ; MyMacro 4, 5, DX
6 MOV BX, 00005h
7 MOV CX, DX
8 RET
```

BX = 0002h.

Adım 3 / 8

DEĞİŞKENLER

AX = 0001h

BX = 0002h

CX = 0012h

DX = 0000h

SP = FFFh

SP hiç değişmedi: makro yığın kullanmaz (gövdesinde PUSH/POP yoksa). Prosedürde CALL/RET SP'yi değiştirirdi.

Açılmış Kodu Çalıştırmak

```
1 ORG 100h
2 MOV AX, 00001h ; MyMacro 1, 2, 3
3 MOV BX, 00002h
4 MOV CX, 00003h
5 MOV AX, 00004h ; MyMacro 4, 5, DX
6 MOV BX, 00005h
7 MOV CX, DX
8 RET
```

CX = 0003h. Birinci çağrı bitti; hiçbir CALL/RET yok.

Adım 4 / 8

DEĞİŞKENLER

AX = 0001h

BX = 0002h

CX = 0003h

DX = 0000h

SP = FFFeh

SP hiç değişmedi: makro yığın kullanmaz (gövdesinde PUSH/POP yoksa). Prosedürde CALL/RET SP'yi değiştirirdi.

Açılmış Kodu Çalıştırmak

```
1 ORG 100h
2 MOV AX, 00001h ; MyMacro 1, 2, 3
3 MOV BX, 00002h
4 MOV CX, 00003h
5 MOV AX, 00004h ; MyMacro 4, 5, DX
6 MOV BX, 00005h
7 MOV CX, DX
8 RET
```

AX = 0004h.

Adım 5 / 8

DEĞİŞKENLER

AX = 0004h

BX = 0002h

CX = 0003h

DX = 0000h

SP = FFFh

SP hiç değişmedi: makro yığın kullanmaz (gövdesinde PUSH/POP yoksa). Prosedürde CALL/RET SP'yi değiştirirdi.

Açılmış Kodu Çalıştırmak

```
1 ORG 100h
2 MOV AX, 00001h ; MyMacro 1, 2, 3
3 MOV BX, 00002h
4 MOV CX, 00003h
5 MOV AX, 00004h ; MyMacro 4, 5, DX
6 MOV BX, 00005h
7 MOV CX, DX
8 RET
```

BX = 0005h.

Adım 6 / 8

DEĞİŞKENLER

AX = 0004h

BX = 0005h

CX = 0003h

DX = 0000h

SP = FFFh

SP hiç değişmedi: makro yığın kullanmaz (gövdesinde PUSH/POP yoksa). Prosedürde CALL/RET SP'yi değiştirirdi.

Açılmış Kodu Çalıştırmak

```
1 ORG 100h
2 MOV AX, 00001h ; MyMacro 1, 2, 3
3 MOV BX, 00002h
4 MOV CX, 00003h
5 MOV AX, 00004h ; MyMacro 4, 5, DX
6 MOV BX, 00005h
7 MOV CX, DX
8 RET
```

CX ← DX = 0000h (DX başlangıçta sıfır).

Adım 7 / 8

DEĞİŞKENLER

AX = 0004h

BX = 0005h

CX = 0000h

DX = 0000h

SP = FFFeh

SP hiç değişmedi: makro yığın kullanmaz (gövdesinde PUSH/POP yoksa). Prosedürde CALL/RET SP'yi değiştirirdi.

Açılmış Kodu Çalıştırmak

```
1 ORG 100h
2 MOV AX, 00001h ; MyMacro 1, 2, 3
3 MOV BX, 00002h
4 MOV CX, 00003h
5 MOV AX, 00004h ; MyMacro 4, 5, DX
6 MOV BX, 00005h
7 MOV CX, DX
▶ RET
```

RET: program biter.

Adım 8 / 8

DEĞİŞKENLER

AX = 0004h

BX = 0005h

CX = 0000h

DX = 0000h

SP = 0000h

SP hiç değişmedi: makro yığın kullanmaz (gövdesinde PUSH/POP yoksa). Prosedürde CALL/RET SP'yi değiştirirdi.

Tuzak: Parametreler Metindir

► **ORG** 100h

```
2 MOV AX, 1111h
3 MOV BX, 2222h
4 MOV AX, BX ; MyMacro BX,AX,0
5 MOV BX, AX ; p2 = AX (bozuk!)
6 MOV CX, 0
7 RET
```

MyMacro BX, AX, 0 çağrısının amacı: $AX \leftarrow BX$, $BX \leftarrow AX$ (takas) sanılıyor.

Adım 1 / 7

DEĞİŞKENLER

AX = 0000h

BX = 0000h

CX = 0000h

Makro parametresi bir değişken değil, yerine konan bir metindir: değeri, kullanıldığı satırda okunur. Makro gövdesinin değiştirdiği yazmaçları parametre olarak geçirmeyin.

Tuzak: Parametreler Metindir

```
1 ORG 100h
2 MOV AX, 1111h
3 MOV BX, 2222h
4 MOV AX, BX ; MyMacro BX,AX,0
5 MOV BX, AX ; p2 = AX (bozuk!)
6 MOV CX, 0
7 RET
```

AX = 1111h.

Adım 2 / 7

DEĞİŞKENLER

AX = 1111h

BX = 0000h

CX = 0000h

Makro parametresi bir değişken değil, yerine konan bir metindir: değeri, kullanıldığı satırda okunur. Makro gövdesinin değiştirdiği yazmaçları parametre olarak geçirmeyin.

Tuzak: Parametreler Metindir

```
1 ORG 100h
2 MOV AX, 1111h
3 MOV BX, 2222h
4 MOV AX, BX ; MyMacro BX,AX,0
5 MOV BX, AX ; p2 = AX (bozuk!)
6 MOV CX, 0
7 RET
```

BX = 2222h.

Adım 3 / 7

DEĞİŞKENLER

AX = 1111h

BX = 2222h

CX = 0000h

Makro parametresi bir değişken değil, yerine konan bir metindir: değeri, kullanıldığı satırda okunur. Makro gövdesinin değiştirdiği yazmaçları parametre olarak geçirmeyin.

Tuzak: Parametreler Metindir

```
1 ORG 100h
2 MOV AX, 1111h
3 MOV BX, 2222h
4 ▶ MOV AX, BX ; MyMacro BX,AX,0
5 MOV BX, AX ; p2 = AX (bozuk!)
6 MOV CX, 0
7 RET
```

Makronun 1. satırı: `MOV AX, p1` → `MOV AX, BX` →
AX = 2222h.

Adım 4 / 7

DEĞİŞKENLER

AX = 2222h

BX = 2222h

CX = 0000h

Makro parametresi bir değişken değil, yerine konan bir metindir: değeri, kullanıldığı satırda okunur. Makro gövdesinin değiştirdiği yazmaçları parametre olarak geçirmeyin.

Tuzak: Parametreler Metindir

```
1 ORG 100h
2 MOV AX, 1111h
3 MOV BX, 2222h
4 MOV AX, BX ; MyMacro BX,AX,0
5 MOV BX, AX ; p2 = AX (bozuk!)
6 MOV CX, 0
7 RET
```

2. satır: `MOV BX, AX`. Programcı “eski AX” (1111h) bekliyordu ama AX bir önceki satırda değişti → BX = **2222h**.

Adım 5 / 7

DEĞİŞKENLER

AX = **2222h**

BX = **2222h**

CX = **0000h**

Makro parametresi bir değişken değil, yerine konan bir metindir: değeri, kullanıldığı satırda okunur. Makro gövdesinin değiştirdiği yazmaçları parametre olarak geçirmeyin.

Tuzak: Parametreler Metindir

```
1 ORG 100h
2 MOV AX, 1111h
3 MOV BX, 2222h
4 MOV AX, BX ; MyMacro BX,AX,0
5 MOV BX, AX ; p2 = AX (bozuk!)
▶ MOV CX, 0
7 RET
```

CX = 0.

Adım 6 / 7

DEĞİŞKENLER

AX = **2222h**

BX = **2222h**

CX = **0000h**

Makro parametresi bir değişken değil, yerine konan bir metindir: değeri, kullanıldığı satırda okunur. Makro gövdesinin değiştirdiği yazmaçları parametre olarak geçirmeyin.

Tuzak: Parametreler Metindir

```
1 ORG 100h
2 MOV AX, 1111h
3 MOV BX, 2222h
4 MOV AX, BX ; MyMacro BX,AX,0
5 MOV BX, AX ; p2 = AX (bozuk!)
6 MOV CX, 0
▶ RET
```

Sonuç: AX = BX = 2222h. 1111h kayboldu.

Adım 7 / 7

DEĞİŞKENLER

AX = **2222h**

BX = **2222h**

CX = **0000h**

Makro parametresi bir değişken değil, yerine konan bir metindir: değeri, kullanıldığı satırda okunur. Makro gövdesinin değiştirdiği yazmaçları parametre olarak geçirmeyin.

Etkileşimli Makro Genişletici

Makro

MyMacro p1, p2, p3 ▼

1.

1

2.

2

3.

3

kullanım sayısı

1

Çağrı

MyMacro 1, 2, 3

Derleyicinin ürettiği kod

```
MOV AX, 00001h
MOV BX, 00002h
MOV CX, 00003h
```

Bir kopya $\approx$ **9 bayt**. 1 kullanım: makro $\approx$ **9 bayt**, prosedür $\approx 10 + 1 \times 3 =$ **13 bayt**.

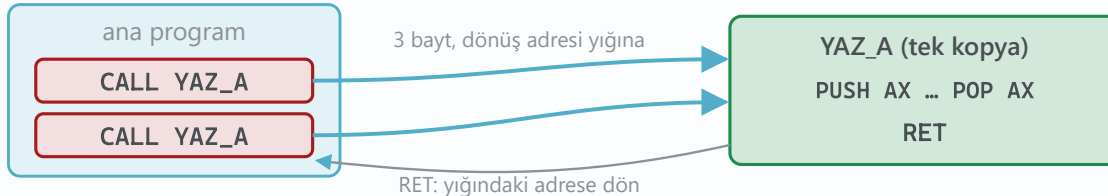
Boyut karşılaştırması: aynı gövde bir prosedür olsaydı gövde + RET bir kez yazılır, her kullanım yalnızca 3 baytlık bir CALL olurdu.

Prosedür Özellikleri

- Prosedür `CALL` komutu ile çağrılır.
- Prosedür belleğin belirli bir adresinde **tek kopya** olarak bulunur.
- Çağrıldığında CPU kontrolü bu bölgeye aktarır; `RET` ile kontrol programın ana kısmına geri döner.
- Dönüş adresini saklamak için **yığın** kullanılır.
- Aynı prosedür 100 kez çağrılırsa dosya boyutu çok az artar: yakın `CALL` komutu yalnızca **3 bayt** yer kaplar.
- Parametre geçirmek için yığın veya yazmaçlar kullanılır.
- Prosedür, adı ve `ENDP` direktifiyle sonlandırılır.

```
YAZ_A PROC
    PUSH AX
    MOV AL, 'A'
    MOV AH, 0Eh
    INT 10h
    POP AX
    RET
YAZ_A ENDP

CALL YAZ_A ; 3 bayt
CALL YAZ_A ; 3 bayt
```

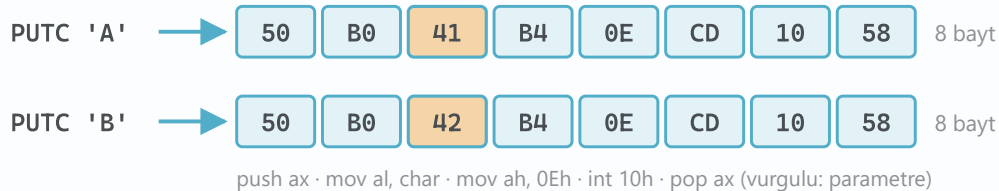


Makro Özellikleri

- Makro sadece **adı yazılarak** kullanılır (CALL yok).
- Makrolar program kodunda **doğrudan genişletilir**.
- Aynı makro 100 kez kullanılırsa derleyici makroyu 100 kez genişletir; çalıştırılabilir dosyanın boyutu giderek büyür.
- Parametre geçirmek için makro adının ardından değerler yazılır.
- Makroyu sonlandırmak için **ENDM** direktifi yeterlidir.
- Makro tanımında etiketler varsa ve makro birden fazla kez kullanılırsa **Duplicate declaration** hatası alınabilir; değişken, etiket ve prosedür adları için **LOCAL** direktifi kullanılır.

```
PUTC MACRO char
    PUSH AX
    MOV AL, char
    MOV AH, 0Eh
    INT 10h
    POP AX
ENDM
```

```
PUTC 'A' ; 8 bayt
PUTC 'B' ; 8 bayt
```



MAKRO VE PROSEDÜR

Makro mu, Prosedür mü?

Ölçüt	Makro	Prosedür
Ne zaman işlenir?	Derleme zamanı (metin değiştirme)	Çalışma zamanı (CALL/RET)
Kullanım	PUTC 'A'	CALL YAZ_A
Kod boyutu	Her kullanımda gövde kadar artar	Gövde 1 kez + her çağrı 3 bayt
Hız	Daha hızlı (CALL/RET ve yığın erişimi yok)	Her çağrıda CALL + RET maliyeti
Parametre	Ad sonrası değerler, metin olarak	Yazmaç ya da yığın
Yığın	Kullanmaz (gövde kullanmadıkça)	Dönüş adresi için kullanır
Tanım yeri	Kullanımdan önce	Herhangi bir yerde
Bitiş	ENDM	ad ENDP
Etiketler	LOCAL gerekir	Sorun yok (tek kopya)

Kural: kısa ve sık kullanılan, hız gerektiren kod → makro; uzun ve çok yerde kullanılan kod → prosedür. İkisi birlikte de kullanılabilir: makro, parametreleri yazmaçlara koyup bir prosedürü çağırabilir.

Boyut Hesabı: 1, 10 ve 100 Kullanım

Kod boyutu (bayt) · PUTC gövdesi 8 bayt; prosedür = aynı 8 bayt + RET (1 bayt)

1 kullanım

makro: $1 \times 8 = 8$

prosedür: $9 + 1 \times 3 = 12$

1 kullanım: makro 8 bayt; prosedür 8 bayt gövde + 1 bayt RET + 3 bayt CALL = 12 bayt.

Burada makro **daha küçük**.

Adım 1 / 3

Hız tarafında tersi geçerlidir: her CALL + RET çifti 8086'da yaklaşık $19 + 8 = 27$ saat turu ve iki yığın erişimi ekler. Makroda bu maliyet yoktur.

Boyut Hesabı: 1, 10 ve 100 Kullanım

Kod boyutu (bayt) · PUTC gövdesi 8 bayt; prosedür = aynı 8 bayt + RET (1 bayt)

1 kullanım	 makro: $1 \times 8 = 8$
	 prosedür: $9 + 1 \times 3 = 12$
10 kullanım	 makro: $10 \times 8 = 80$
	 prosedür: $9 + 10 \times 3 = 39$

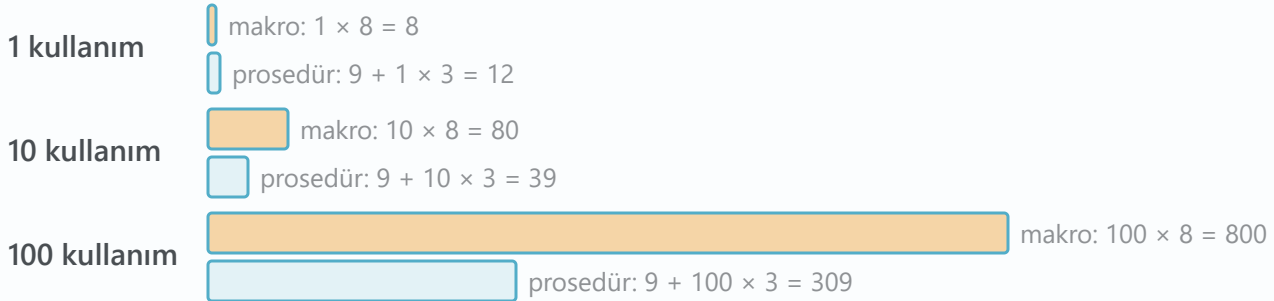
10 kullanım: makro 80 bayt; prosedür $9 + 30 = 39$ bayt.

Adım 2 / 3

Hız tarafında tersi geçerlidir: her CALL + RET çifti 8086'da yaklaşık $19 + 8 = 27$ saat turu ve iki yığın erişimi ekler. Makroda bu maliyet yoktur.

Boyut Hesabı: 1, 10 ve 100 Kullanım

Kod boyutu (bayt) · PUTC gövdesi 8 bayt; prosedür = aynı 8 bayt + RET (1 bayt)



100 kullanım: makro 800 bayt; prosedür $9 + 300 = 309$ bayt. Fark kullanım sayısıyla büyür.

Adım 3 / 3

Hız tarafında tersi geçerlidir: her CALL + RET çifti 8086'da yaklaşık $19 + 8 = 27$ saat turu ve iki yığın erişimi ekler. Makroda bu maliyet yoktur.

Karar Alıştırması: Makro mu, Prosedür mü?

Beş durum: her birinde kod boyutu (kaç kopya?) ve hız (CALL/RET maliyeti) birlikte tartılır.

Adım 1 / 6

Karar Alıştırması: Makro mu, Prosedür mü?

Beş durum: her birinde kod boyutu (kaç kopya?) ve hız (CALL/RET maliyeti) birlikte tartılır.

- 1. Ekrana tek karakter yazmak, 200 yerde** → kısa gövde (8 bayt) ama 200 kopya = 1600 bayt. Prosedür $9 + 600 = 609$ bayt. Boyut önemliyse **prosedür**; hız önemliyse makro.

Karar Alıştırması: Makro mu, Prosedür mü?

Beş durum: her birinde kod boyutu (kaç kopya?) ve hız (CALL/RET maliyeti) birlikte tartılır.

- 1. Ekranı tek karakter yazmak, 200 yerde** → kısa gövde (8 bayt) ama 200 kopya = 1600 bayt. Prosedür $9 + 600 = 609$ bayt. Boyut önemliyse **prosedür**; hız önemliyse makro.
- 2. İki yazmacı takas eden 1 satırlık iş** → `XCHG` zaten tek komut; makro okunabilirlik için yeterli, prosedür gereksiz yük (**makro**).

Karar Alıştırması: Makro mu, Prosedür mü?

Beş durum: her birinde kod boyutu (kaç kopya?) ve hız (CALL/RET maliyeti) birlikte tartılır.

- 1. Ekrana tek karakter yazmak, 200 yerde** → kısa gövde (8 bayt) ama 200 kopya = 1600 bayt. Prosedür $9 + 600 = 609$ bayt. Boyut önemliyse **prosedür**; hız önemliyse makro.
- 2. İki yazmacı takas eden 1 satırlık iş** → `XCHG` zaten tek komut; makro okunabilirlik için yeterli, prosedür gereksiz yük (**makro**).
- 3. 300 baytlık sıralama algoritması, 3 yerde** → makro 900 bayt, prosedür $300 + 9$ bayt: açıkça **prosedür**.

Karar Alıştırması: Makro mu, Prosedür mü?

Beş durum: her birinde kod boyutu (kaç kopya?) ve hız (CALL/RET maliyeti) birlikte tartılır.

- 1. Ekrana tek karakter yazmak, 200 yerde** → kısa gövde (8 bayt) ama 200 kopya = 1600 bayt. Prosedür $9 + 600 = 609$ bayt. Boyut önemliyse **prosedür**; hız önemliyse makro.
- 2. İki yazmacı takas eden 1 satırlık iş** → `XCHG` zaten tek komut; makro okunabilirlik için yeterli, prosedür gereksiz yük (**makro**).
- 3. 300 baytlık sıralama algoritması, 3 yerde** → makro 900 bayt, prosedür $300 + 9$ bayt: açıkça **prosedür**.
- 4. Sıkı bir döngü içinde her turda yapılan 2 komutluk iş** → CALL/RET maliyeti için kendisinden büyük: **makro**.

Karar Alıştırması: Makro mu, Prosedür mü?

Beş durum: her birinde kod boyutu (kaç kopya?) ve hız (CALL/RET maliyeti) birlikte tartılır.

- 1. Ekranı tek karakter yazmak, 200 yerde** → kısa gövde (8 bayt) ama 200 kopya = 1600 bayt. Prosedür $9 + 600 = 609$ bayt. Boyut önemliyse **prosedür**; hız önemliyse makro.
- 2. İki yazmacı takas eden 1 satırlık iş** → `XCHG` zaten tek komut; makro okunabilirlik için yeterli, prosedür gereksiz yük (**makro**).
- 3. 300 baytlık sıralama algoritması, 3 yerde** → makro 900 bayt, prosedür $300 + 9$ bayt: açıkça **prosedür**.
- 4. Sıkı bir döngü içinde her turda yapılan 2 komutluk iş** → CALL/RET maliyeti için kendisinden büyük: **makro**.
- 5. Parametreleri yazmaçlara koyup ortak kodu çağırmak** → ikisi birlikte: makro
`YAZ_SAYI x` → `MOV AX, x` + `CALL print_num` (emu8086.inc de böyle çalışır).

Makro Parametreyi Hazırlar, Prosedür İşi Yapar

```

1 org 100h
2 ► mov al, 4 ; YAZ_RAKAM 4
3 call rakam_yaz
4 mov al, 7 ; YAZ_RAKAM 7
5 call rakam_yaz
6 ret
7 rakam_yaz:
8 add al, '0' ; rakam → ASCII
9 mov ah, 0Eh
10 int 10h
11 ret

```



Makro `YAZ_RAKAM d`, derlemede iki satıra açıldı: `mov al, d` + `call rakam_yaz`. Gövdenin asıl işi prosedürde, **tek kopya**. İlk kopya: AL = 04h.

Adım 1 / 9

DEĞİŞKENLER

AL = 04h

AH = 00h

SP = FFFEh

IP = 0102h

Makro: `YAZ_RAKAM MACRO d` / `mov al, d` / `call rakam_yaz` / `ENDM` . 10

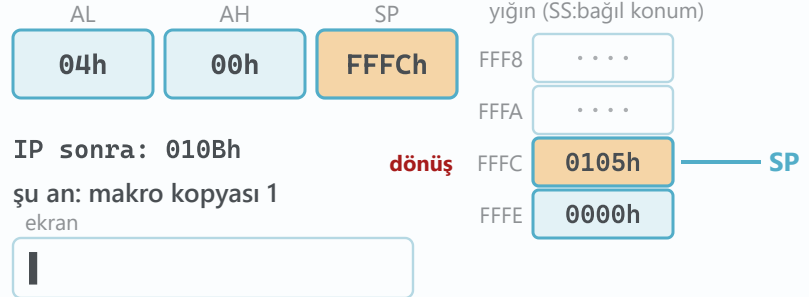
kullanımda kod $10 \times 5 + 7 = 57$ bayt; gövdenin tamamı makroda olsaydı $10 \times 8 = 80$ bayt.

Makro Parametreyi Hazırlar, Prosedür İşi Yapar

```

1 org 100h
2 mov al, 4      ; YAZ_RAKAM 4
3 ► call rakam_yaz
4 mov al, 7      ; YAZ_RAKAM 7
5 call rakam_yaz
6 ret
7 rakam_yaz:
8 add al, '0'     ; rakam → ASCII
9 mov ah, 0Eh
10 int 10h
11 ret

```



CALL, dönüş adresini (CALL'dan sonraki komut, **0105h**) yığına iter: SP = FFFCh. IP ← rakam_yaz.

Adım 2 / 9

DEĞİŞKENLER

AL = 04h

AH = 00h

SP = FFFCh

IP = 010Bh

Makro: YAZ_RAKAM MACRO d / mov al, d / call rakam_yaz / ENDM . 10

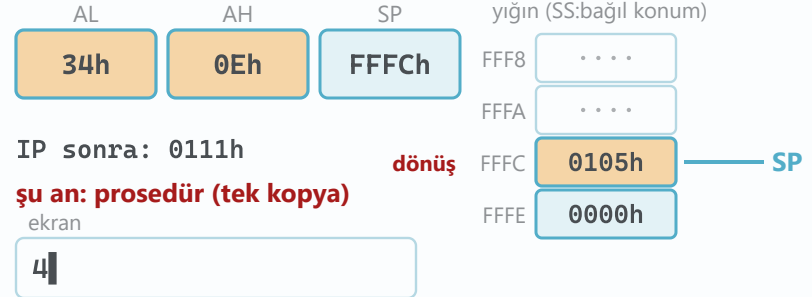
kullanımda kod $10 \times 5 + 7 = 57$ bayt; gövdenin tamamı makroda olsaydı $10 \times 8 = 80$ bayt.

Makro Parametreyi Hazırlar, Prosedür İşi Yapar

```

1 org 100h
2 mov al, 4      ; YAZ_RAKAM 4
3 call rakam_yaz
4 mov al, 7      ; YAZ_RAKAM 7
5 call rakam_yaz
6 ret
7 rakam_yaz:
▶ add al, '0'    ; rakam → ASCII
▶ mov ah, 0Eh
▶ int 10h
11 ret

```



Prosedür gövdesi: AL + 30h = **34h** ('4'), AH = 0Eh, INT 10h
 ekrana **4** yazar.

Adım 3 / 9

DEĞİŞKENLER

AL = 34h

AH = 0Eh

SP = FFFCh

IP = 0111h

Makro: `YAZ_RAKAM MACRO d / mov al, d / call rakam_yaz / ENDM` . 10

kullanımda kod $10 \times 5 + 7 = 57$ bayt; gövdenin tamamı makroda olsaydı $10 \times 8 = 80$ bayt.

Makro Parametreyi Hazırlar, Prosedür İşi Yapar

```

1 org 100h
2 mov al, 4      ; YAZ_RAKAM 4
3 call rakam_yaz
4 mov al, 7      ; YAZ_RAKAM 7
5 call rakam_yaz
6 ret
7 rakam_yaz:
8 add al, '0'    ; rakam → ASCII
9 mov ah, 0Eh
10 int 10h
▶ ret

```



RET yığından 0105h'yi çeker: SP = FFEh, yürütme makronun ilk kopyasından sonraki satırda sürer.

Adım 4 / 9

DEĞİŞKENLER

AL = 34h

AH = 0Eh

SP = FFEh

IP = 0105h

Makro: `YAZ_RAKAM MACRO d / mov al, d / call rakam_yaz / ENDM` . 10

kullanımda kod $10 \times 5 + 7 = 57$ bayt; gövdenin tamamı makroda olsaydı $10 \times 8 = 80$ bayt.

Makro Parametreyi Hazırlar, Prosedür İşi Yapar

```

1 org 100h
2 mov al, 4      ; YAZ_RAKAM 4
3 call rakam_yaz
4 ► mov al, 7    ; YAZ_RAKAM 7
5 call rakam_yaz
6 ret
7 rakam_yaz:
8 add al, '0'    ; rakam → ASCII
9 mov ah, 0Eh
10 int 10h
11 ret

```



Makronun ikinci kopyası: AL = **07h**. Her kopya yalnız 5 bayt (2 bayt MOV + 3 bayt CALL).

Adım 5 / 9

DEĞİŞKENLER

AL = 07h

AH = 0Eh

SP = FFEh

IP = 0107h

Makro: `YAZ_RAKAM MACRO d / mov al, d / call rakam_yaz / ENDM` . 10

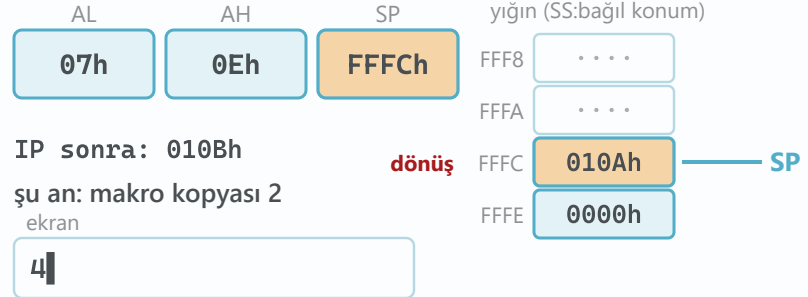
kullanımda kod $10 \times 5 + 7 = 57$ bayt; gövdenin tamamı makroda olsaydı $10 \times 8 = 80$ bayt.

Makro Parametreyi Hazırlar, Prosedür İşi Yapar

```

1 org 100h
2 mov al, 4      ; YAZ_RAKAM 4
3 call rakam_yaz
4 mov al, 7      ; YAZ_RAKAM 7
5 ► call rakam_yaz
6 ret
7 rakam_yaz:
8 add al, '0'    ; rakam → ASCII
9 mov ah, 0Eh
10 int 10h
11 ret

```



Yine CALL: bu kez dönüş adresi **010Ah**. Aynı prosedür, farklı dönüş adresi.

Adım 6 / 9

DEĞİŞKENLER

AL = 07h

AH = 0Eh

SP = FFFCh

IP = 010Bh

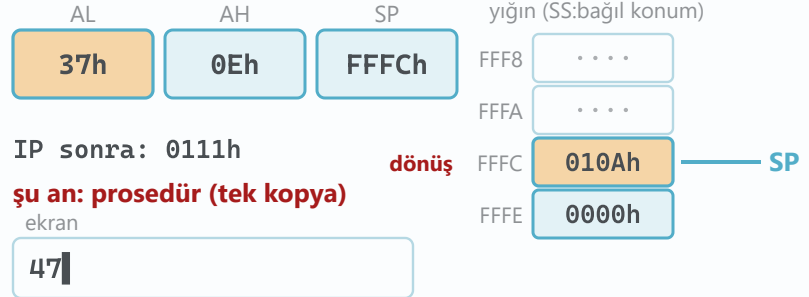
Makro: `YAZ_RAKAM MACRO d / mov al, d / call rakam_yaz / ENDM` . 10

kullanımda kod $10 \times 5 + 7 = 57$ bayt; gövdenin tamamı makroda olsaydı $10 \times 8 = 80$ bayt.

Makro Parametreyi Hazırlar, Prosedür İşi Yapar

```

1 org 100h
2 mov al, 4      ; YAZ_RAKAM 4
3 call rakam_yaz
4 mov al, 7      ; YAZ_RAKAM 7
5 call rakam_yaz
6 ret
7 rakam_yaz:
▶ add al, '0'    ; rakam → ASCII
▶ mov ah, 0Eh
▶ int 10h
11 ret
    
```



Aynı tek prosedür kopyası çalışır: AL = 37h ('7'). Ekranda **47**.

Adım 7 / 9

DEĞİŞKENLER

AL = 37h

AH = 0Eh

SP = FFFCh

IP = 0111h

Makro: YAZ_RAKAM MACRO d / mov al, d / call rakam_yaz / ENDM . 10

kullanımda kod $10 \times 5 + 7 = 57$ bayt; gövdenin tamamı makroda olsaydı $10 \times 8 = 80$ bayt.

Makro Parametreyi Hazırlar, Prosedür İşi Yapar

```

1 org 100h
2 mov al, 4      ; YAZ_RAKAM 4
3 call rakam_yaz
4 mov al, 7      ; YAZ_RAKAM 7
5 call rakam_yaz
6 ret
7 rakam_yaz:
8 add al, '0'    ; rakam → ASCII
9 mov ah, 0Eh
10 int 10h
▶ ret
    
```



RET → 010Ah. Yığın yine boşaldı; çağırının SP'si korunmuş oldu.

Adım 8 / 9

DEĞİŞKENLER

AL = 37h

AH = 0Eh

SP = FFEh

IP = 010Ah

Makro: `YAZ_RAKAM MACRO d / mov al, d / call rakam_yaz / ENDM` . 10

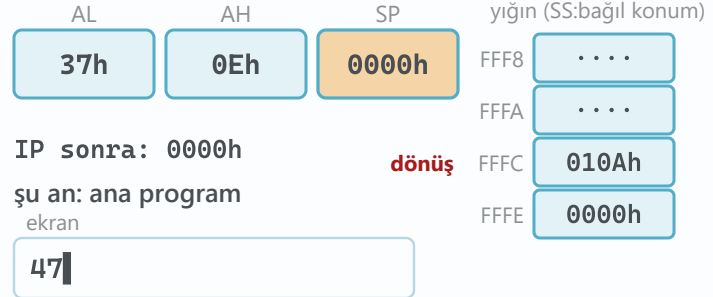
kullanımda kod $10 \times 5 + 7 = 57$ bayt; gövdenin tamamı makroda olsaydı $10 \times 8 = 80$ bayt.

Makro Parametreyi Hazırlar, Prosedür İşi Yapar

```

1 org 100h
2 mov al, 4      ; YAZ_RAKAM 4
3 call rakam_yaz
4 mov al, 7      ; YAZ_RAKAM 7
5 call rakam_yaz
6 ► ret
7 rakam_yaz:
8 add al, '0'    ; rakam → ASCII
9 mov ah, 0Eh
10 int 10h
11 ret

```



Program biter. Makro parametreyi yazmaca koydu, ortak kod prosedürde: emu8086.inc'deki `DEFINE_ ...` + makro ikilisi de böyle çalışır.

Adım 9 / 9

DEĞİŞKENLER

AL = 37h

AH = 0Eh

SP = 0000h

IP = 0000h

Makro: `YAZ_RAKAM MACRO d / mov al, d / call rakam_yaz / ENDM` . 10

kullanımda kod $10 \times 5 + 7 = 57$ bayt; gövdenin tamamı makroda olsaydı $10 \times 8 = 80$ bayt.

Etiket İçeren Makrolar: Sorun

Gövdesinde etiket olan bir makro iki kez kullanılırsa, genişletme sonucu aynı etiket **iki kez** tanımlanmış olur:

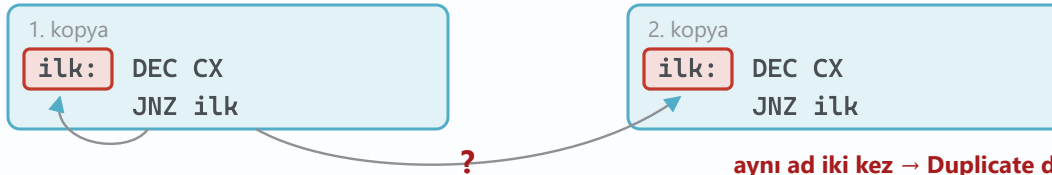
```
BEKLE MACRO  
ilk: DEC CX  
      JNZ ilk  
ENDM
```

```
MOV CX, 5  
BEKLE  
MOV CX, 9  
BEKLE
```

Derleyicinin gördüğü:

```
MOV CX, 5  
ilk: DEC CX      ; 1. kopya  
      JNZ ilk  
MOV CX, 9  
ilk: DEC CX      ; 2. kopya !  
      JNZ ilk
```

emu8086: **Duplicate declaration** hatası.
Hangi ilk: 'e atlanacağı belirsizdir.



Çözüm: LOCAL Direktifi

```
MyMacro2 MACRO
    LOCAL label1, label2
    CMP AX, 2
    JE label1
    CMP AX, 3
    JE label2
label1:
    INC AX
label2:
    ADD AX, 2
ENDM
```

- `LOCAL` , makro gövdesinin **ilk satırı** olmalıdır (MACRO satırından hemen sonra).
- Listelenen adlar her genişletmede derleyici tarafından **benzersiz** adlarla değiştirilir (MASM'da `??0000` , `??0001` , ...).
- Etiketler, değişkenler ve makro içindeki prosedür adları için kullanılabilir.

Kullanım	label1	label2
1. MyMacro2	??0000	??0001
2. MyMacro2	??0002	??0003

MyMacro2'yi Çalıştırmak (AX = 2)

► **ORG** 100h

```
2 MOV AX, 2      ; MyMacro2 öncesi
3 CMP  AX, 2      ; MyMacro2 başlar
4 JE   ??0000
5 CMP  AX, 3
6 JE   ??0001
7 ??0000:
8 INC  AX
9 ??0001:
10 ADD AX, 2      ; MyMacro2 biter
11 RET
```

Etiketler LOCAL ile benzersiz adlara dönüştürülmüş hâli.

Adım 1 / 7

DEĞİŞKENLER

AX = 0000h

ZF = 0

CF = 0

AX = 3 olsaydı label2'ye atlar, AX = 5 olur. AX = 7 → hiçbir atlama yok, label1'e düşer: INC + ADD → AX = 10.

MyMacro2'yi Çalıştırmak (AX = 2)

```
1  ORG 100h
2  ► MOV AX, 2      ; MyMacro2 öncesi
3  CMP AX, 2      ; MyMacro2 başlar
4  JE   ??0000
5  CMP AX, 3
6  JE   ??0001
7  ??0000:
8  INC AX
9  ??0001:
10 ADD AX, 2      ; MyMacro2 biter
11 RET
```

AX = 2.

Adım 2 / 7

DEĞİŞKENLER

AX = 0002h

ZF = 0

CF = 0

AX = 3 olsaydı label2'ye atlar, AX = 5 olur. AX = 7 → hiçbir atlama yok, label1'e düşer: INC + ADD → AX = 10.

MyMacro2'yi Çalıştırmak (AX = 2)

```
1 ORG 100h
2 MOV AX, 2 ; MyMacro2 öncesi
3 CMP AX, 2 ; MyMacro2 başlar
4 JE ??0000
5 CMP AX, 3
6 JE ??0001
7 ??0000:
8 INC AX
9 ??0001:
10 ADD AX, 2 ; MyMacro2 biter
11 RET
```

CMP AX, 2: $2 - 2 = 0 \rightarrow \mathbf{ZF = 1}$.

Adım 3 / 7

DEĞİŞKENLER

AX = 0002h

ZF = 1

CF = 0

AX = 3 olsaydı label2'ye atlar, AX = 5 olur. AX = 7 → hiçbir atlama yok, label1'e düşer: INC + ADD → AX = 10.

MyMacro2'yi Çalıştırmak (AX = 2)

```
1  ORG 100h
2  MOV AX, 2      ; MyMacro2 öncesi
3  CMP AX, 2      ; MyMacro2 başlar
4  JE  ??0000
5  CMP AX, 3
6  JE  ??0001
7  ??0000:
8  INC AX
9  ??0001:
10 ADD AX, 2      ; MyMacro2 biter
11 RET
```

JE: ZF = 1 olduğundan **??0000**'a (label1) atlanır.

Adım 4 / 7

DEĞİŞKENLER

AX = 0002h

ZF = 1

CF = 0

AX = 3 olsaydı label2'ye atlar, AX = 5 olur. AX = 7 → hiçbir atlama yok, label1'e düşer: INC + ADD → AX = 10.

MyMacro2'yi Çalıştırmak (AX = 2)

```
1 ORG 100h
2 MOV AX, 2          ; MyMacro2 öncesi
3 CMP AX, 2          ; MyMacro2 başlar
4 JE   ??0000
5 CMP AX, 3
6 JE   ??0001
7 ??0000:
8 INC AX
9 ??0001:
10 ADD AX, 2          ; MyMacro2 biter
11 RET
```

INC AX → AX = 3.

Adım 5 / 7

DEĞİŞKENLER

AX = 0003h

ZF = 0

CF = 0

AX = 3 olsaydı label2'ye atlar, AX = 5 olur. AX = 7 → hiçbir atlama yok, label1'e düşer: INC + ADD → AX = 10.

MyMacro2'yi Çalıştırmak (AX = 2)

```
1 ORG 100h
2 MOV AX, 2      ; MyMacro2 öncesi
3 CMP AX, 2      ; MyMacro2 başlar
4 JE   ??0000
5 CMP AX, 3
6 JE   ??0001
7 ??0000:
8 INC AX
9 ??0001:
▶ ADD AX, 2      ; MyMacro2 biter
11 RET
```

ADD AX, 2 → AX = 5.

Adım 6 / 7

DEĞİŞKENLER

AX = 0005h

ZF = 0

CF = 0

AX = 3 olsaydı label2'ye atlar, AX = 5 olur. AX = 7 → hiçbir atlama yok, label1'e düşer: INC + ADD → AX = 10.

MyMacro2'yi Çalıştırmak (AX = 2)

```
1 ORG 100h
2 MOV AX, 2      ; MyMacro2 öncesi
3 CMP AX, 2      ; MyMacro2 başlar
4 JE   ??0000
5 CMP AX, 3
6 JE   ??0001
7 ??0000:
8 INC AX
9 ??0001:
10 ADD AX, 2      ; MyMacro2 biter
    ► RET
```

Program biter.

Adım 7 / 7

DEĞİŞKENLER

AX = 0005h

ZF = 0

CF = 0

AX = 3 olsaydı label2'ye atlar, AX = 5 olur. AX = 7 → hiçbir atlama yok, label1'e düşer: INC + ADD → AX = 10.

LOCAL ile İki Kopya, İki Ayrı Etiket

```
1 org 100h
```

```
▶ mov cx, 3
```

```
3 ??0000: dec cx ; BEKLE (1. kopya)
```

```
4 jnz ??0000
```

```
5 mov cx, 2
```

```
6 ??0001: dec cx ; BEKLE (2. kopya)
```

```
7 jnz ??0001
```

```
8 ret
```

CX

0003h

ZF
0

IP sonra: 0103h

1. kopya · ilk → ??0000

etiket adresi 0103h

2. kopya · ilk → ??0001

etiket adresi 0109h

Makroda **LOCAL ilk** var. Derleyici her açılımda **ilk** yerine yeni bir ad koydu: 1. kopyada **??0000** (0103h), 2. kopyada **??0001** (0109h). CX = 3.

Adım 1 / 8

DEĞİŞKENLER

CX = 0003h

ZF = 0

IP = 0103h

Makro: **BEKLE MACRO** / **LOCAL ilk** / **ilk: DEC CX** / **JNZ ilk** / **ENDM** ; ana programda **mov cx, 3** / **BEKLE** / **mov cx, 2** / **BEKLE** .

LOCAL ile İki Kopya, İki Ayrı Etiket

```

1 org 100h
2 mov cx, 3
▶ ??0000: dec cx ; BEKLE (1. kopya)
▶ jnz ??0000
5 mov cx, 2
6 ??0001: dec cx ; BEKLE (2. kopya)
7 jnz ??0001
8 ret

```

CX

0002h

ZF
0

IP sonra: 0103h

1. kopya · ilk → ??0000
etiket adresi 0103h

JNZ: atla

2. kopya · ilk → ??0001
etiket adresi 0109h

1. tur: DEC CX → CX = 2, ZF = 0; JNZ ??0000'a (0103h) döner.

Adım 2 / 8

DEĞİŞKENLER

CX = 0002h

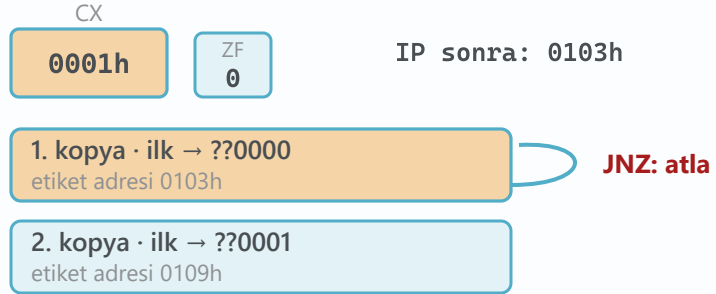
ZF = 0

IP = 0103h

Makro: BEKLE MACRO / LOCAL ilk / ilk: DEC CX / JNZ ilk / ENDM ; ana
programda mov cx, 3 / BEKLE / mov cx, 2 / BEKLE .

LOCAL ile İki Kopya, İki Ayrı Etiket

```
1 org 100h
2 mov cx, 3
▶ ??0000: dec cx ; BEKLE (1. kopya)
▶ jnz ??0000
5 mov cx, 2
6 ??0001: dec cx ; BEKLE (2. kopya)
7 jnz ??0001
8 ret
```



2. tur: CX = 1, ZF = 0 → yine 0103h.

Adım 3 / 8

DEĞİŞKENLER

CX = 0001h

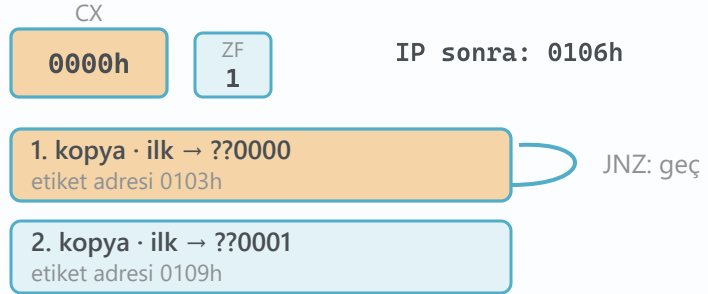
ZF = 0

IP = 0103h

Makro: BEKLE MACRO / LOCAL ilk / ilk: DEC CX / JNZ ilk / ENDM ; ana
programda mov cx, 3 / BEKLE / mov cx, 2 / BEKLE .

LOCAL ile İki Kopya, İki Ayrı Etiket

```
1 org 100h
2 mov cx, 3
▶ ??0000: dec cx ; BEKLE (1. kopya)
▶ jnz ??0000
5 mov cx, 2
6 ??0001: dec cx ; BEKLE (2. kopya)
7 jnz ??0001
8 ret
```



3. tur: CX = 0 → ZF = 1; JNZ atlamaz, 1. kopya bitti.

Adım 4 / 8

DEĞİŞKENLER

CX = 0000h

ZF = 1

IP = 0106h

Makro: BEKLE MACRO / LOCAL ilk / ilk: DEC CX / JNZ ilk / ENDM ; ana
programda mov cx, 3 / BEKLE / mov cx, 2 / BEKLE .

LOCAL ile İki Kopya, İki Ayrı Etiket

```
1 org 100h
2 mov cx, 3
3 ??0000: dec cx ; BEKLE (1. kopya)
4 jnz ??0000
▶ mov cx, 2
6 ??0001: dec cx ; BEKLE (2. kopya)
7 jnz ??0001
8 ret
```

CX

0002h

ZF
1

IP sonra: 0109h

1. kopya · ilk → ??0000
etiket adresi 0103h

2. kopya · ilk → ??0001
etiket adresi 0109h

CX = 2: makronun ikinci kullanımı için sayaç.

Adım 5 / 8

DEĞİŞKENLER

CX = 0002h

ZF = 1

IP = 0109h

Makro: BEKLE MACRO / LOCAL ilk / ilk: DEC CX / JNZ ilk / ENDM ; ana
programda mov cx, 3 / BEKLE / mov cx, 2 / BEKLE .

LOCAL ile İki Kopya, İki Ayrı Etiket

```

1 org 100h
2 mov cx, 3
3 ??0000: dec cx    ; BEKLE (1. kopya)
4 jnz ??0000
5 mov cx, 2
▶ ??0001: dec cx    ; BEKLE (2. kopya)
▶ jnz ??0001
8 ret

```

CX

0001h

ZF
0

IP sonra: 0109h

1. kopya · ilk → ??0000
etiket adresi 0103h

2. kopya · ilk → ??0001
etiket adresi 0109h

JNZ: atla

2. kopya: DEC CX → CX = 1; JNZ **??0001**'e (0109h) döner, 1. kopyanın etiketine değil.

Adım 6 / 8

DEĞİŞKENLER

CX = 0001h

ZF = 0

IP = 0109h

Makro: BEKLE MACRO / LOCAL ilk / ilk: DEC CX / JNZ ilk / ENDM ; ana programda mov cx, 3 / BEKLE / mov cx, 2 / BEKLE .

LOCAL ile İki Kopya, İki Ayrı Etiket

```
1 org 100h
2 mov cx, 3
3 ??0000: dec cx ; BEKLE (1. kopya)
4 jnz ??0000
5 mov cx, 2
6 ??0001: dec cx ; BEKLE (2. kopya)
7 jnz ??0001
8 ret
```

CX

0000h

ZF

1

IP sonra: 010Ch

1. kopya · ilk → ??0000
etiket adresi 0103h

2. kopya · ilk → ??0001
etiket adresi 0109h

JNZ: geç

CX = 0, ZF = 1: döngü biter. Aynı etiket adı iki kez tanımlanmadığı için *Duplicate declaration* hatası yok.

Adım 7 / 8

DEĞİŞKENLER

CX = 0000h

ZF = 1

IP = 010Ch

Makro: BEKLE MACRO / LOCAL ilk / ilk: DEC CX / JNZ ilk / ENDM ; ana programda mov cx, 3 / BEKLE / mov cx, 2 / BEKLE .

LOCAL ile İki Kopya, İki Ayrı Etiket

```
1 org 100h
2 mov cx, 3
3 ??0000: dec cx ; BEKLE (1. kopya)
4 jnz ??0000
5 mov cx, 2
6 ??0001: dec cx ; BEKLE (2. kopya)
7 jnz ??0001
▶ ret
```

CX

0000h

ZF
1

IP sonra: 0000h

1. kopya · ilk → ??0000
etiket adresi 0103h

2. kopya · ilk → ??0001
etiket adresi 0109h

RET: program biter. Toplam 3 + 2 = 5 DEC çalıştı.

Adım 8 / 8

DEĞİŞKENLER

CX = 0000h

ZF = 1

IP = 0000h

Makro: BEKLE MACRO / LOCAL ilk / ilk: DEC CX / JNZ ilk / ENDM ; ana
programda mov cx, 3 / BEKLE / mov cx, 2 / BEKLE .

PUTC: Ekrana Karakter Yazdırma

```
; karakter yazar, imleci ilerletir
PUTC  MACRO  char
    push  ax
    mov   al, char
    mov   ah, 0eh
    int   10h
    pop   ax
ENDM

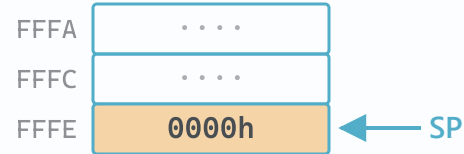
ORG 100h
START:
    PUTC  'A'
    PUTC  'B'
    RET
```

- Makro, `char` parametresindeki karakteri yazar ve imleci bir sağa ilerletir.
- `INT 10h` , `AH = 0Eh` : BIOS **teletype** çıkışı. AL'deki karakter imleç konumuna yazılır; 0Dh, 0Ah, 07h, 08h gibi kontrol karakterleri yorumlanır.
- `push ax` / `pop ax` : makro AX'i kullandığı için çağırının AX değerini korur. Makronun "görünmez" yan etkisi kalmaz.
- Bu makro `emu8086.inc` kütüphanesindeki PUTC makrosunun aynısıdır.

PUTC 'B' 'den sonraki RET programı bitirir; o olmasa işlemci program sonundaki rastgele baytları çalıştırmaya devam ederdi.

PUTC 'A' ve PUTC 'B' Çalışırken

```
► ORG 100h
2 push ax          ; PUTC 'A'
3 mov al, 'A'
4 mov ah, 0eh
5 int 10h
6 pop ax
7 push ax          ; PUTC 'B'
8 mov al, 'B'
9 mov ah, 0eh
10 int 10h
11 pop ax
12 RET
```



SS:bağlı konum · küçük adres yukarıda

Yükleme sonrası CX = **0011h**: 2 kopya × 8 bayt + RET = 17 bayt. Ekran boş.

Adım 1 / 12

DEĞİŞKENLER

AX = 0000h

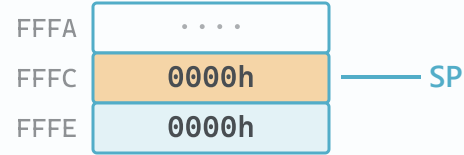
SP = FFEh

ÇIKTI

emu8086 ekran penceresinde (80×25) çıktı: AB. Her PUTC kopyası 8 bayt: 50 B0 41 B4 0E CD 10 58.

PUTC 'A' ve PUTC 'B' Çalışırken

```
1 ORG 100h
2 ► push ax ; PUTC 'A'
3 mov al, 'A'
4 mov ah, 0eh
5 int 10h
6 pop ax
7 push ax ; PUTC 'B'
8 mov al, 'B'
9 mov ah, 0eh
10 int 10h
11 pop ax
12 RET
```



SS:bağlı konum · küçük adres yukarıda

AX (0000h) yığına saklanır; SP = FFFCh.

Adım 2 / 12

DEĞİŞKENLER

AX = 0000h

SP = FFFCh

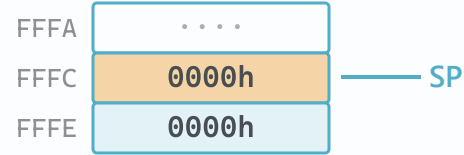
ÇIKTI

emu8086 ekran penceresinde (80×25) çıktı: AB. Her PUTC kopyası 8 bayt: 50 B0 41 B4 0E CD 10 58.

PUTC MAKROSU · KODU ADIM ADIM ÇALIŞTIR

PUTC 'A' ve PUTC 'B' Çalışırken

```
1 ORG 100h
2 push ax ; PUTC 'A'
3 ► mov al, 'A'
4 mov ah, 0eh
5 int 10h
6 pop ax
7 push ax ; PUTC 'B'
8 mov al, 'B'
9 mov ah, 0eh
10 int 10h
11 pop ax
12 RET
```



SS:bağlı konum · küçük adres yukarıda

AL = 41h ('A').

Adım 3 / 12

DEĞİŞKENLER

AX = 0041h

SP = FFFCh

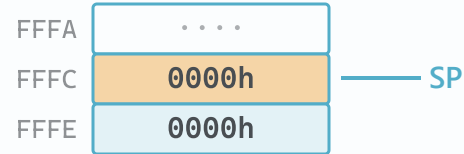
ÇIKTI

emu8086 ekran penceresinde (80×25) çıktı: AB. Her PUTC kopyası 8 bayt: 50 B0 41 B4 0E CD 10 58.

PUTC MAKROSU · KODU ADIM ADIM ÇALIŞTIR

PUTC 'A' ve PUTC 'B' Çalışırken

```
1 ORG 100h
2 push ax          ; PUTC 'A'
3 mov al, 'A'
4 mov ah, 0eh
5 int 10h
6 pop ax
7 push ax          ; PUTC 'B'
8 mov al, 'B'
9 mov ah, 0eh
10 int 10h
11 pop ax
12 RET
```



SS:bağlı konum · küçük adres yukarıda

AH = 0Eh → AX = 0E41h.

Adım 4 / 12

DEĞİŞKENLER

AX = 0E41h

SP = FFFCh

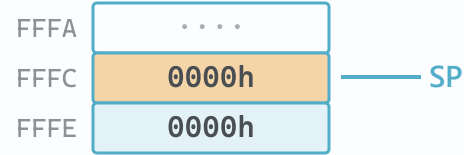
ÇIKTI

emu8086 ekran penceresinde (80×25) çıktı: AB. Her PUTC kopyası 8 bayt: 50 B0 41 B4 0E CD 10 58.

PUTC MAKROSU · KODU ADIM ADIM ÇALIŞTIR

PUTC 'A' ve PUTC 'B' Çalışırken

```
1 ORG 100h
2 push ax          ; PUTC 'A'
3 mov al, 'A'
4 mov ah, 0eh
5 int 10h
6 pop ax
7 push ax          ; PUTC 'B'
8 mov al, 'B'
9 mov ah, 0eh
10 int 10h
11 pop ax
12 RET
```



SS:bağlı konum · küçük adres yukarıda

INT 10h: ekrana **A** yazılır, imleç ilerler.

Adım 5 / 12

DEĞİŞKENLER

AX = 0E41h

SP = FFFCh

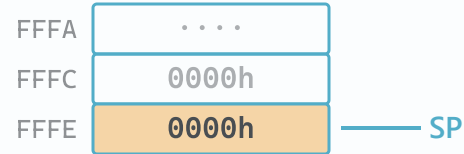
ÇIKTI

A

emu8086 ekran penceresinde (80×25) çıktı: AB. Her PUTC kopyası 8 bayt: 50 B0 41 B4 0E CD 10 58.

PUTC 'A' ve PUTC 'B' Çalışırken

```
1 ORG 100h
2 push ax          ; PUTC 'A'
3 mov al, 'A'
4 mov ah, 0eh
5 int 10h
6 pop ax
7 push ax          ; PUTC 'B'
8 mov al, 'B'
9 mov ah, 0eh
10 int 10h
11 pop ax
12 RET
```



SS:bağlı konum · küçük adres yukarıda

AX eski değerine (0000h) döner; SP = FFFEh.

Adım 6 / 12

DEĞİŞKENLER

AX = 0000h

SP = FFFEh

ÇIKTI

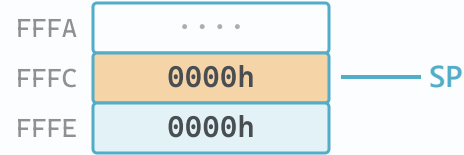
A

emu8086 ekran penceresinde (80×25) çıktı: AB. Her PUTC kopyası 8 bayt: 50 B0 41 B4 0E CD 10 58.

PUTC MAKROSU · KODU ADIM ADIM ÇALIŞTIR

PUTC 'A' ve PUTC 'B' Çalışırken

```
1 ORG 100h
2 push ax          ; PUTC 'A'
3 mov al, 'A'
4 mov ah, 0eh
5 int 10h
6 pop ax
7 push ax          ; PUTC 'B'
8 mov al, 'B'
9 mov ah, 0eh
10 int 10h
11 pop ax
12 RET
```



SS:bağlı konum · küçük adres yukarıda

İkinci kopya: AX yine saklanır.

Adım 7 / 12

DEĞİŞKENLER

AX = 0000h

SP = FFFCh

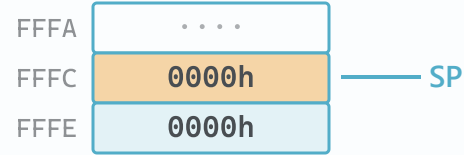
ÇIKTI

A

emu8086 ekran penceresinde (80×25) çıktı: AB. Her PUTC kopyası 8 bayt: 50 B0 41 B4 0E CD 10 58.

PUTC 'A' ve PUTC 'B' Çalışırken

```
1 ORG 100h
2 push ax          ; PUTC 'A'
3 mov al, 'A'
4 mov ah, 0eh
5 int 10h
6 pop ax
7 push ax          ; PUTC 'B'
8 mov al, 'B'
9 mov ah, 0eh
10 int 10h
11 pop ax
12 RET
```



SS:bağlı konum · küçük adres yukarıda

AL = 42h ('B').

Adım 8 / 12

DEĞİŞKENLER

AX = 0042h

SP = FFFCh

ÇIKTI

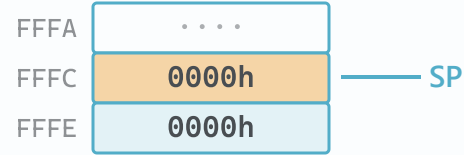
A

emu8086 ekran penceresinde (80×25) çıktı: AB. Her PUTC kopyası 8 bayt: 50 B0 41 B4 0E CD 10 58.

PUTC MAKROSU · KODU ADIM ADIM ÇALIŞTIR

PUTC 'A' ve PUTC 'B' Çalışırken

```
1 ORG 100h
2 push ax          ; PUTC 'A'
3 mov al, 'A'
4 mov ah, 0eh
5 int 10h
6 pop ax
7 push ax          ; PUTC 'B'
8 mov al, 'B'
9 mov ah, 0eh
10 int 10h
11 pop ax
12 RET
```



SS:bağlı konum · küçük adres yukarıda

AX = 0E42h.

Adım 9 / 12

DEĞİŞKENLER

AX = 0E42h

SP = FFFCh

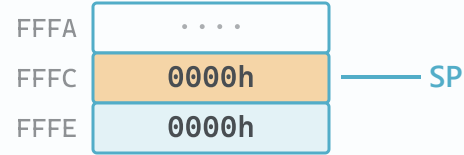
ÇIKTI

A

emu8086 ekran penceresinde (80×25) çıktı: AB. Her PUTC kopyası 8 bayt: 50 B0 41 B4 0E CD 10 58.

PUTC 'A' ve PUTC 'B' Çalışırken

```
1 ORG 100h
2 push ax          ; PUTC 'A'
3 mov al, 'A'
4 mov ah, 0eh
5 int 10h
6 pop ax
7 push ax          ; PUTC 'B'
8 mov al, 'B'
9 mov ah, 0eh
10 int 10h
11 pop ax
12 RET
```



SS:bağlı konum · küçük adres yukarıda

INT 10h: ekranda **AB**.

Adım 10 / 12

DEĞİŞKENLER

AX = 0E42h

SP = FFFCh

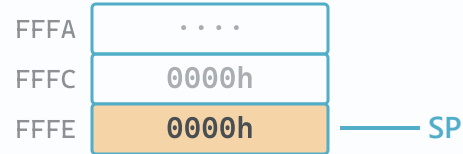
ÇIKTI

AB

emu8086 ekran penceresinde (80×25) çıktı: AB. Her PUTC kopyası 8 bayt: 50 B0 41 B4 0E CD 10 58.

PUTC 'A' ve PUTC 'B' Çalışırken

```
1 ORG 100h
2 push ax          ; PUTC 'A'
3 mov al, 'A'
4 mov ah, 0eh
5 int 10h
6 pop ax
7 push ax          ; PUTC 'B'
8 mov al, 'B'
9 mov ah, 0eh
10 int 10h
11 pop ax
12 RET
```



SS:bağlı konum · küçük adres yukarıda

AX = 0000h; makro çağırının AX'ini bozmadı.

Adım 11 / 12

DEĞİŞKENLER

AX = 0000h

SP = FFEh

ÇIKTI

AB

emu8086 ekran penceresinde (80×25) çıktı: AB. Her PUTC kopyası 8 bayt: 50 B0 41 B4 0E CD 10 58.

PUTC 'A' ve PUTC 'B' Çalışırken

```
1 ORG 100h
2 push ax          ; PUTC 'A'
3 mov al, 'A'
4 mov ah, 0eh
5 int 10h
6 pop ax
7 push ax          ; PUTC 'B'
8 mov al, 'B'
9 mov ah, 0eh
10 int 10h
11 pop ax
▶ RET
```

FFFA

....

FFFC

0000h

FFFE

0000h

SS:bağlı konum · küçük adres yukarıda

RET: program biter.

Adım 12 / 12

DEĞİŞKENLER

AX = 0000h

SP = 0000h

ÇIKTI

AB

emu8086 ekran penceresinde (80×25) çıktı: AB. Her PUTC kopyası 8 bayt: 50 B0 41 B4 0E CD 10 58.

POWER MAKROSU

Üs Alma: POWER Makrosu

```
POWER macro b, e, r
    mov ax, 1          ; geçici sonuç = 1
    mov cx, e          ; üs → CX
    mov bx, b          ; taban → BX
powerloop:
    mul bx             ; DX:AX = AX * BX
    loop powerloop     ; e kez tekrarla
    mov r, ax          ; sonucu yaz
endm
```

```
ORG 100h
start:
    POWER base, exponent, result
    ret
base    dw 2          ; taban
exponent dw 3          ; üs
result  dw ?          ; sonuç
```

- Sonuç $r = b^e$: AX'i 1 ile başlatıp e kez taban ile çarpar.
- **MUL BX** : $DX:AX \leftarrow AX \times BX$ (işaretsiz). **DX de değişir!**
- **LOOP** : CX'i 1 azaltır, $CX \neq 0$ ise etikete atlar.

POWER 2, 3 Adım Adım

► ORG 100h

```
2 mov ax, 1          ; POWER açılımı
3 mov cx, exponent
4 mov bx, base
5 powerloop:
6 mul bx
7 loop powerloop
8 mov result, ax
9 ret
10 base      dw 2      ; 0113h
11 exponent  dw 3      ; 0115h
12 result    dw ?      ; 0117h
```

Veriler kodun hemen arkasında: base = 0113h, exponent = 0115h, result = 0117h (kod 19 bayt).

Adım 1 / 12

DEĞİŞKENLER

AX = 0000h

BX = 0000h

CX = 0000h

DX = 0000h

[0117h] = 0000h

CF = 0

MUL sonrası DX = 0 olduğundan CF = OF = 0. Sonuç 16 biti aşıyadı (ör. 2^{16}) DX $\neq$ 0 ve CF = 1 olurdu; makro yalnızca AX'i sakladığı için sonuç yanlış olurdu.

POWER 2, 3 Adım Adım

```
1 ORG 100h
▶ mov ax, 1      ; POWER açılımı
3 mov cx, exponent
4 mov bx, base
5 powerloop:
6 mul bx
7 loop powerloop
8 mov result, ax
9 ret
10 base      dw 2      ; 0113h
11 exponent dw 3      ; 0115h
12 result    dw ?      ; 0117h
```

AX = 1 (geçici sonuç).

Adım 2 / 12

DEĞİŞKENLER

AX = 0001h

BX = 0000h

CX = 0000h

DX = 0000h

[0117h] = 0000h

CF = 0

MUL sonrası DX = 0 olduğundan CF = OF = 0. Sonuç 16 biti aşıyadı (ör. 2^{16}) DX $\neq$ 0 ve CF = 1 olurdu; makro yalnızca AX'i sakladığı için sonuç yanlış olurdu.

POWER 2, 3 Adım Adım

```
1 ORG 100h
2 mov ax, 1      ; POWER açılımı
3 mov cx, exponent
4 mov bx, base
5 powerloop:
6 mul bx
7 loop powerloop
8 mov result, ax
9 ret
10 base      dw 2      ; 0113h
11 exponent dw 3       ; 0115h
12 result    dw ?      ; 0117h
```

$CX \leftarrow [0115h] = 3$ (üs).

Adım 3 / 12

DEĞİŞKENLER

AX = 0001h

BX = 0000h

CX = 0003h

DX = 0000h

[0117h] = 0000h

CF = 0

MUL sonrası $DX = 0$ olduğundan $CF = OF = 0$. Sonuç 16 biti aşıyadı (ör. 2^{16}) $DX \neq 0$ ve $CF = 1$ olurdu; makro yalnızca AX'i sakladığı için sonuç yanlış olurdu.

POWER 2, 3 Adım Adım

```
1 ORG 100h
2 mov ax, 1      ; POWER açılımı
3 mov cx, exponent
4 mov bx, base
5 powerloop:
6 mul bx
7 loop powerloop
8 mov result, ax
9 ret
10 base      dw 2      ; 0113h
11 exponent dw 3      ; 0115h
12 result    dw ?      ; 0117h
```

$BX \leftarrow [0113h] = 2$ (taban).

Adım 4 / 12

DEĞİŞKENLER

AX = 0001h

BX = 0002h

CX = 0003h

DX = 0000h

[0117h] = 0000h

CF = 0

MUL sonrası $DX = 0$ olduğundan $CF = OF = 0$. Sonuç 16 biti aşıyadı (ör. 2^{16}) $DX \neq 0$ ve $CF = 1$ olurdu; makro yalnızca AX'i sakladığı için sonuç yanlış olurdu.

POWER 2, 3 Adım Adım

```
1 ORG 100h
2 mov ax, 1      ; POWER açılımı
3 mov cx, exponent
4 mov bx, base
5 powerloop:
6 mul bx
7 loop powerloop
8 mov result, ax
9 ret
10 base      dw 2      ; 0113h
11 exponent dw 3      ; 0115h
12 result    dw ?      ; 0117h
```

1. tur: $DX:AX = 1 \times 2 = \mathbf{0002h}$.

Adım 5 / 12

DEĞİŞKENLER

AX = **0002h**

BX = **0002h**

CX = **0003h**

DX = **0000h**

[0117h] = **0000h**

CF = **0**

MUL sonrası $DX = 0$ olduğundan $CF = OF = 0$. Sonuç 16 biti aşıyadı (ör. 2^{16}) $DX \neq 0$ ve $CF = 1$ olurdu; makro yalnızca AX'i sakladığı için sonuç yanlış olurdu.

POWER 2, 3 Adım Adım

```
1 ORG 100h
2 mov ax, 1      ; POWER açılımı
3 mov cx, exponent
4 mov bx, base
5 powerloop:
6 mul bx
7 loop powerloop
8 mov result, ax
9 ret
10 base      dw 2      ; 0113h
11 exponent dw 3      ; 0115h
12 result    dw ?      ; 0117h
```

LOOP: CX = 2 $\neq$ 0 $\rightarrow$ powerloop.

Adım 6 / 12

DEĞİŞKENLER

AX = 0002h

BX = 0002h

CX = 0002h

DX = 0000h

[0117h] = 0000h

CF = 0

MUL sonrası DX = 0 olduğundan CF = OF = 0. Sonuç 16 biti aşıyadı (ör. 2^{16}) DX $\neq$ 0 ve CF = 1 olurdu; makro yalnızca AX'i sakladığı için sonuç yanlış olurdu.

POWER 2, 3 Adım Adım

```
1 ORG 100h
2 mov ax, 1      ; POWER açılımı
3 mov cx, exponent
4 mov bx, base
5 powerloop:
6 ▶ mul bx
7 loop powerloop
8 mov result, ax
9 ret
10 base      dw 2      ; 0113h
11 exponent dw 3      ; 0115h
12 result    dw ?      ; 0117h
```

2. tur: $AX = 2 \times 2 = \mathbf{0004h}$.

Adım 7 / 12

DEĞİŞKENLER

AX = **0004h**

BX = 0002h

CX = 0002h

DX = 0000h

[0117h] = 0000h

CF = 0

MUL sonrası $DX = 0$ olduğundan $CF = OF = 0$. Sonuç 16 biti aşıyadı (ör. 2^{16}) $DX \neq 0$ ve $CF = 1$ olurdu; makro yalnızca AX'i sakladığı için sonuç yanlış olurdu.

POWER 2, 3 Adım Adım

```
1 ORG 100h
2 mov ax, 1      ; POWER açılımı
3 mov cx, exponent
4 mov bx, base
5 powerloop:
6 mul bx
7 loop powerloop
8 mov result, ax
9 ret
10 base      dw 2      ; 0113h
11 exponent dw 3      ; 0115h
12 result    dw ?      ; 0117h
```

LOOP: CX = 1 → powerloop.

Adım 8 / 12

DEĞİŞKENLER

AX = 0004h

BX = 0002h

CX = 0001h

DX = 0000h

[0117h] = 0000h

CF = 0

MUL sonrası DX = 0 olduğundan CF = OF = 0. Sonuç 16 bit'i aşıyadı (ör. 2^{16}) DX ≠ 0 ve CF = 1 olurdu; makro yalnızca AX'i sakladığı için sonuç yanlış olurdu.

POWER 2, 3 Adım Adım

```
1 ORG 100h
2 mov ax, 1      ; POWER açılımı
3 mov cx, exponent
4 mov bx, base
5 powerloop:
6 mul bx
7 loop powerloop
8 mov result, ax
9 ret
10 base      dw 2      ; 0113h
11 exponent dw 3      ; 0115h
12 result    dw ?      ; 0117h
```

3. tur: $AX = 4 \times 2 = \mathbf{0008h}$.

Adım 9 / 12

DEĞİŞKENLER

AX = **0008h**

BX = **0002h**

CX = **0001h**

DX = **0000h**

[0117h] = **0000h**

CF = **0**

MUL sonrası $DX = 0$ olduğundan $CF = OF = 0$. Sonuç 16 biti aşıyadı (ör. 2^{16}) $DX \neq 0$ ve $CF = 1$ olurdu; makro yalnızca AX'i sakladığı için sonuç yanlış olurdu.

POWER 2, 3 Adım Adım

```
1 ORG 100h
2 mov ax, 1      ; POWER açılımı
3 mov cx, exponent
4 mov bx, base
5 powerloop:
6 mul bx
7 loop powerloop
8 mov result, ax
9 ret
10 base      dw 2      ; 0113h
11 exponent dw 3      ; 0115h
12 result    dw ?      ; 0117h
```

LOOP: CX = 0 → döngüden çık.

Adım 10 / 12

DEĞİŞKENLER

AX = 0008h

BX = 0002h

CX = 0000h

DX = 0000h

[0117h] = 0000h

CF = 0

MUL sonrası DX = 0 olduğundan CF = OF = 0. Sonuç 16 biti aşıyadı (ör. 2^{16}) DX ≠ 0 ve CF = 1 olurdu; makro yalnızca AX'i sakladığı için sonuç yanlış olurdu.

POWER 2, 3 Adım Adım

```
1 ORG 100h
2 mov ax, 1      ; POWER açılımı
3 mov cx, exponent
4 mov bx, base
5 powerloop:
6 mul bx
7 loop powerloop
8 mov result, ax
9 ret
10 base      dw 2      ; 0113h
11 exponent dw 3      ; 0115h
12 result    dw ?      ; 0117h
```

result ← AX: [0117h] = **0008h** = 2^3 .

Adım 11 / 12

DEĞİŞKENLER

AX = **0008h**

BX = **0002h**

CX = **0000h**

DX = **0000h**

[0117h] = **0008h**

CF = **0**

MUL sonrası DX = 0 olduğundan CF = OF = 0. Sonuç 16 biti aşıyadı (ör. 2^{16}) DX ≠ 0 ve CF = 1 olurdu; makro yalnızca AX'i sakladığı için sonuç yanlış olurdu.

POWER 2, 3 Adım Adım

```
1 ORG 100h
2 mov ax, 1      ; POWER açılımı
3 mov cx, exponent
4 mov bx, base
5 powerloop:
6 mul bx
7 loop powerloop
8 mov result, ax
9 ret
10 base      dw 2      ; 0113h
11 exponent dw 3      ; 0115h
12 result    dw ?      ; 0117h
```

RET: program biter.

Adım 12 / 12

DEĞİŞKENLER

AX = 0008h

BX = 0002h

CX = 0000h

DX = 0000h

[0117h] = 0008h

CF = 0

MUL sonrası DX = 0 olduğundan CF = OF = 0. Sonuç 16 biti aşıyadı (ör. 2^{16}) DX $\neq$ 0 ve CF = 1 olurdu; makro yalnızca AX'i sakladığı için sonuç yanlış olurdu.

POWER Makrosunun Tuzakları

1. Üs = 0

CX = 0 iken `LOOP` önce CX'i azaltır: FFFFh → döngü **65 536 kez** döner; $b^0 = 1$ olmalıydı.
Çözüm: `JCXZ`.

2. Etiket

`powerloop` `LOCAL` değil: ikinci kullanımda *Duplicate declaration*.

3. Yazmaç bozulması

AX, BX, CX ve (MUL yüzünden) DX değişir.

4. Taşma

Sonuç 65 535'i aşarsa yüksek kısım DX'te kalır ve yazılmaz.

Düzeltilmiş sürüm:

```
POWER MACRO b, e, r
    LOCAL p_loop, p_end
    PUSH BX
    PUSH CX
    PUSH DX
    MOV AX, 1
    MOV CX, e
    MOV BX, b
    JCXZ p_end ; e = 0 → 1
p_loop: MUL BX
    LOOP p_loop
p_end: MOV r, AX
    POP DX
    POP CX
    POP BX
ENDM
```

Düzeltilmiş POWER 7, 2 Adım Adım

```
1 org 100h
```

```
▶ push bx           ; POWER açılımı
```

```
▶ push cx
```

```
▶ push dx
```

```
5 mov ax, 1
```

```
6 mov cx, us
```

```
7 mov bx, taban
```

```
8 jcxz ??0001      ; us = 0 → 1
```

```
9 ??0000: mul bx
```

```
10 loop ??0000
```

```
11 ??0001: mov sonuc, ax
```

```
12 pop dx
```

```
13 pop cx
```

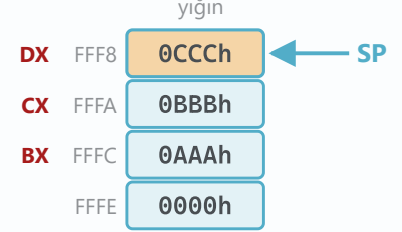
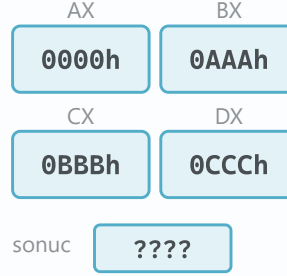
```
14 pop bx
```

```
15 ret
```

```
16 taban dw 7
```

```
17 us     dw 2
```

```
18 sonuc dw ?
```



Çağıranın yazmaçları BX = 0AAAh, CX = 0BBBh, DX = 0CCCh.
Düzeltilmiş makro önce üçünü de yığına saklar: SP = FFF8h.

Adım 1 / 10

DEĞİŞKENLER

AX = 0000h

BX = 0AAAh

CX = 0BBBh

DX = 0CCCh

SP = FFF8h

sonuc = ?

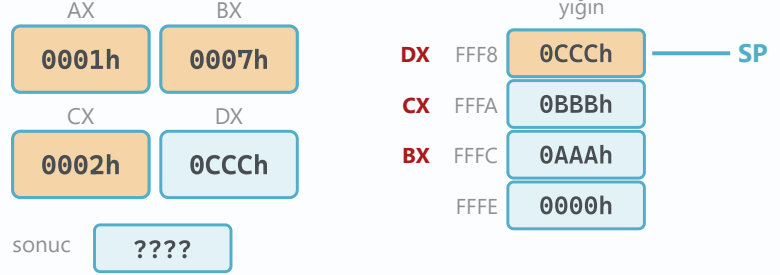
Kod, düzeltilmiş makronun **POWER taban, us, sonuc** açılımıdır; LOCAL adları derleyicinin ürettiği biçimde gösterildi.

Düzeltilmiş POWER 7, 2 Adım Adım

```

1 org 100h
2 push bx          ; POWER açılımı
3 push cx
4 push dx
5 ▶ mov ax, 1
6 ▶ mov cx, us
7 ▶ mov bx, taban
8 jcxz ??0001      ; us = 0 → 1
9 ??0000: mul bx
10 loop ??0000
11 ??0001: mov sonuc, ax
12 pop dx
13 pop cx
14 pop bx
15 ret
16 taban dw 7
17 us     dw 2
18 sonuc dw ?

```



AX = 1, CX ← us = 2, BX ← taban = 7. Yazmaçlar artık makronun.

Adım 2 / 10

DEĞİŞKENLER

AX = 0001h

BX = 0007h

CX = 0002h

DX = 0CCCh

SP = FFF8h

sonuc = ?

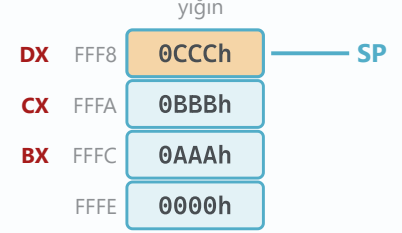
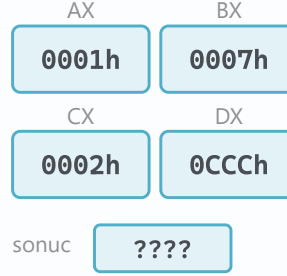
Kod, düzeltilmiş makronun **POWER taban, us, sonuc** açılımıdır; LOCAL adları derleyicinin ürettiği biçimde gösterildi.

Düzeltilmiş POWER 7, 2 Adım Adım

```

1 org 100h
2 push bx          ; POWER açılımı
3 push cx
4 push dx
5 mov ax, 1
6 mov cx, us
7 mov bx, taban
8 jcxz ??0001      ; us = 0 → 1
9 ??0000: mul bx
10 loop ??0000
11 ??0001: mov sonuc, ax
12 pop dx
13 pop cx
14 pop bx
15 ret
16 taban dw 7
17 us dw 2
18 sonuc dw ?

```



JCXZ: CX = 2 ≠ 0, atlama yok. us = 0 olsaydı doğrudan ??0001'e gidilir, sonuç 1 olurdu (LOOP'un 65 536 tur tuzağı yok).

Adım 3 / 10

DEĞİŞKENLER

AX = 0001h

BX = 0007h

CX = 0002h

DX = 0CCCh

SP = FFF8h

sonuc = ?

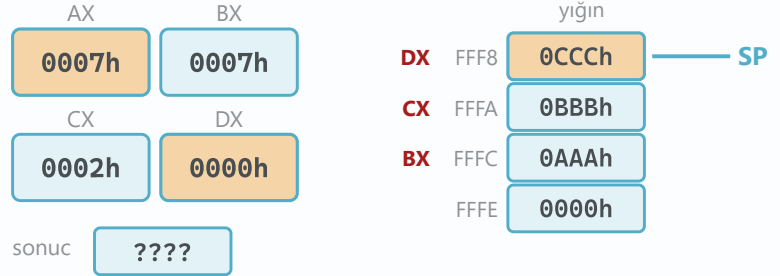
Kod, düzeltilmiş makronun **POWER taban, us, sonuc** açılımıdır; LOCAL adları derleyicinin ürettiği biçimde gösterildi.

Düzeltilmiş POWER 7, 2 Adım Adım

```

1 org 100h
2 push bx          ; POWER açılımı
3 push cx
4 push dx
5 mov ax, 1
6 mov cx, us
7 mov bx, taban
8 jcxz ??0001      ; us = 0 → 1
9 ► ??0000: mul bx
10 loop ??0000
11 ??0001: mov sonuc, ax
12 pop dx
13 pop cx
14 pop bx
15 ret
16 taban dw 7
17 us     dw 2
18 sonuc dw ?

```



1. MUL: $DX:AX = 1 \times 7 = 0007h$. MUL, DX'i de yazar (DX = 0).

Adım 4 / 10

DEĞİŞKENLER

AX = 0007h

BX = 0007h

CX = 0002h

DX = 0000h

SP = FFF8h

sonuc = ?

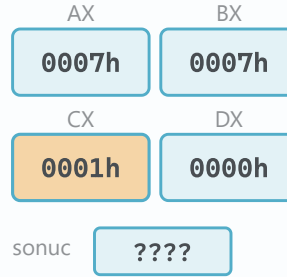
Kod, düzeltilmiş makronun **POWER taban, us, sonuc** açılımıdır; LOCAL adları derleyicinin ürettiği biçimde gösterildi.

Düzeltilmiş POWER 7, 2 Adım Adım

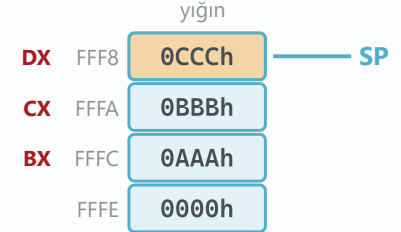
```

1 org 100h
2 push bx          ; POWER açılımı
3 push cx
4 push dx
5 mov ax, 1
6 mov cx, us
7 mov bx, taban
8 jcxz ??0001      ; us = 0 → 1
9 ??0000: mul bx
▶ loop ??0000
11 ??0001: mov sonuc, ax
12 pop dx
13 pop cx
14 pop bx
15 ret
16 taban dw 7
17 us     dw 2
18 sonuc dw ?

```



LOOP: CX = 1 ≠ 0 → ??0000.



Adım 5 / 10

DEĞİŞKENLER

AX = 0007h

BX = 0007h

CX = 0001h

DX = 0000h

SP = FFF8h

sonuc = ?

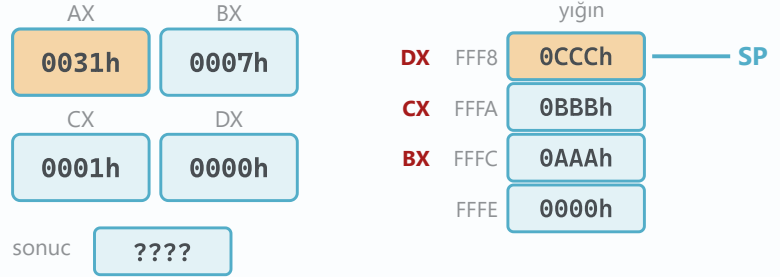
Kod, düzeltilmiş makronun **POWER taban, us, sonuc** açılımıdır; LOCAL adları derleyicinin ürettiği biçimde gösterildi.

Düzeltilmiş POWER 7, 2 Adım Adım

```

1 org 100h
2 push bx          ; POWER açılımı
3 push cx
4 push dx
5 mov ax, 1
6 mov cx, us
7 mov bx, taban
8 jcxz ??0001      ; us = 0 → 1
9 ► ??0000: mul bx
10 loop ??0000
11 ??0001: mov sonuc, ax
12 pop dx
13 pop cx
14 pop bx
15 ret
16 taban dw 7
17 us     dw 2
18 sonuc dw ?

```



2. MUL: DX:AX = $7 \times 7 = 0031h = 49$.

Adım 6 / 10

DEĞİŞKENLER

AX = 0031h

BX = 0007h

CX = 0001h

DX = 0000h

SP = FFF8h

sonuc = ?

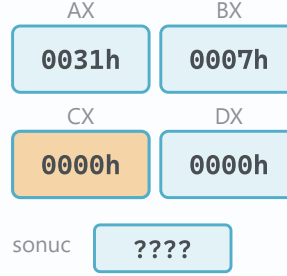
Kod, düzeltilmiş makronun **POWER taban, us, sonuc** açılımıdır; LOCAL adları derleyicinin ürettiği biçimde gösterildi.

Düzeltilmiş POWER 7, 2 Adım Adım

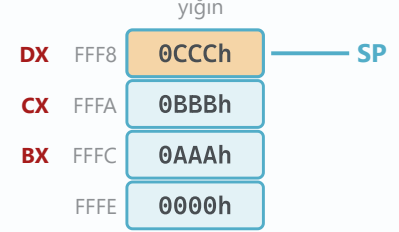
```

1 org 100h
2 push bx          ; POWER açılımı
3 push cx
4 push dx
5 mov ax, 1
6 mov cx, us
7 mov bx, taban
8 jcxz ??0001      ; us = 0 → 1
9 ??0000: mul bx
▶ loop ??0000
11 ??0001: mov sonuc, ax
12 pop dx
13 pop cx
14 pop bx
15 ret
16 taban dw 7
17 us     dw 2
18 sonuc dw ?

```



LOOP: CX = 0 → döngüden çık.



Adım 7 / 10

DEĞİŞKENLER

AX = 0031h

BX = 0007h

CX = 0000h

DX = 0000h

SP = FFF8h

sonuc = ?

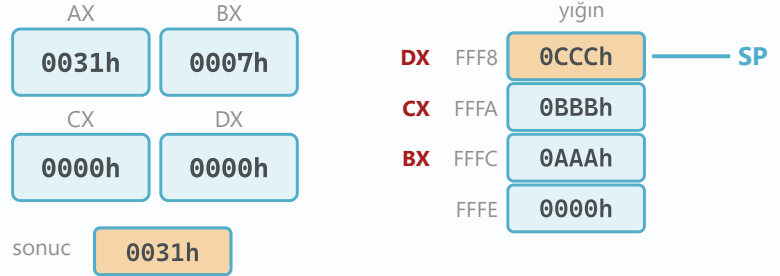
Kod, düzeltilmiş makronun **POWER taban, us, sonuc** açılımıdır; LOCAL adları derleyicinin ürettiği biçimde gösterildi.

Düzeltilmiş POWER 7, 2 Adım Adım

```

1 org 100h
2 push bx          ; POWER açılımı
3 push cx
4 push dx
5 mov ax, 1
6 mov cx, us
7 mov bx, taban
8 jcxz ??0001      ; us = 0 → 1
9 ??0000: mul bx
10 loop ??0000
11 ► ??0001: mov sonuc, ax
12 pop dx
13 pop cx
14 pop bx
15 ret
16 taban dw 7
17 us     dw 2
18 sonuc dw ?

```



sonuc ← AX: [011Fh] = **0031h** = 7^2 . Etiketler LOCAL ile ??0000 / ??0001 adını aldı.

Adım 8 / 10

DEĞİŞKENLER

AX = 0031h

BX = 0007h

CX = 0000h

DX = 0000h

SP = FFF8h

sonuc = 0031h

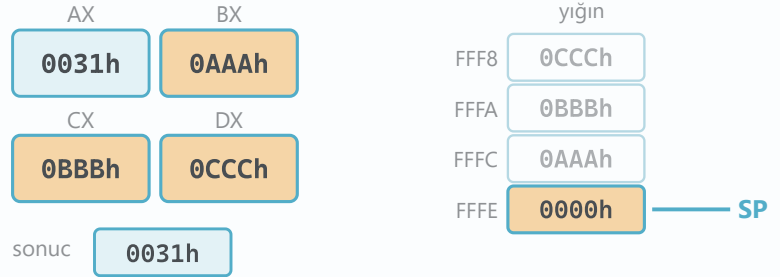
Kod, düzeltilmiş makronun **POWER taban, us, sonuc** açılımıdır; LOCAL adları derleyicinin ürettiği biçimde gösterildi.

Düzeltilmiş POWER 7, 2 Adım Adım

```

1 org 100h
2 push bx          ; POWER açılımı
3 push cx
4 push dx
5 mov ax, 1
6 mov cx, us
7 mov bx, taban
8 jcxz ??0001      ; us = 0 → 1
9 ??0000: mul bx
10 loop ??0000
11 ??0001: mov sonuc, ax
12 pop dx
13 pop cx
14 pop bx
15 ret
16 taban dw 7
17 us     dw 2
18 sonuc dw ?

```



Üç POP, PUSH'ların tersi sırada: DX = 0CCCh, CX = 0BBBh, BX = 0AAAh geri geldi; SP = FFEh. Makro yalnız AX'i değiştirdi.

Adım 9 / 10

DEĞİŞKENLER

AX = 0031h

BX = 0AAAh

CX = 0BBBh

DX = 0CCCh

SP = FFEh

sonuc = 0031h

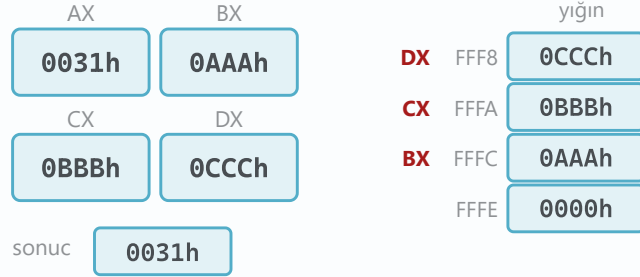
Kod, düzeltilmiş makronun **POWER taban, us, sonuc** açılımıdır; LOCAL adları derleyicinin ürettiği biçimde gösterildi.

Düzeltilmiş POWER 7, 2 Adım Adım

```

1 org 100h
2 push bx          ; POWER açılımı
3 push cx
4 push dx
5 mov ax, 1
6 mov cx, us
7 mov bx, taban
8 jcxz ??0001      ; us = 0 → 1
9 ??0000: mul bx
10 loop ??0000
11 ??0001: mov sonuc, ax
12 pop dx
13 pop cx
14 pop bx
15 ► ret
16 taban dw 7
17 us     dw 2
18 sonuc dw ?

```



RET: program biter.

Adım 10 / 10

DEĞİŞKENLER

AX = 0031h

BX = 0AAAh

CX = 0BBBh

DX = 0CCCh

SP = 0000h

sonuc = 0031h

Kod, düzeltilmiş makronun **POWER taban, us, sonuc** açılımıdır; LOCAL adları derleyicinin ürettiği biçimde gösterildi.

POWER MAKROSU

İç İçe Makrolar

```
PRINT2 MACRO c1, c2  
    PUTC c1  
    PUTC c2  
  
ENDM
```

```
PRINT2 'O', 'K'
```

```
; 1. düzey: PRINT2 açılır  
PUTC 'O'  
PUTC 'K'
```

Önce dıştaki makro açılır: gövdedeki iki satır da birer **makro çağırısıdır**.

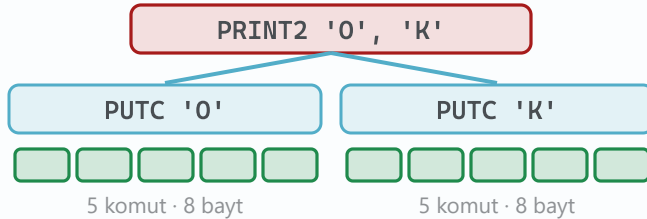
Adım 1 / 2

PUTC, PRINT2'den önce tanımlanmış olmalıdır. Bir makro kendini çağırırsa (özyineleme) koşullu derleme olmadan genişletme hiç bitmez.

İç İçe Makrolar

```
PRINT2 MACRO c1, c2  
    PUTC c1  
    PUTC c2  
ENDM
```

```
PRINT2 'O', 'K'
```



; 2. düzey: PUTC'ler açılır

```
push ax  
mov al, 'O'  
mov ah, 0eh  
int 10h  
pop ax  
push ax  
mov al, 'K'  
mov ah, 0eh  
int 10h  
pop ax
```

Derleyici iç makroları da açar: tek satırlık çağrı **10 komut** (16 bayt) üretir. Ekranda: OK.

Adım 2 / 2

PUTC, PRINT2'den önce tanımlanmış olmalıdır. Bir makro kendini çağırırsa (özyineleme) koşullu derleme olmadan genişletme hiç bitmez.

Makro Kütüphaneleri: emu8086.inc

- Sık kullanılan makrolar ayrı bir dosyada toplanır ve `INCLUDE` ile programa eklenir.
- Kullanılmayan makrolar kod üretmediği için büyük bir kütüphaneyi eklemek dosyayı büyütmez.
- Makrolar başka makroları çağırabilir (iç içe genişletme).
- Kütüphane, makroların yanında `DEFINE_ ...` makrolarıyla prosedürler de sunar (ör. `DEFINE_SCAN_NUM`): ikisi birlikte kullanılır.

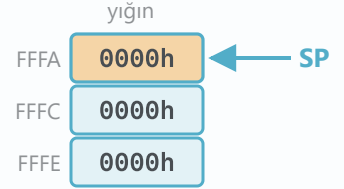
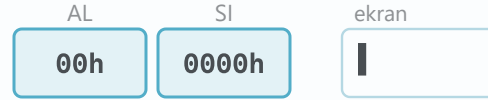
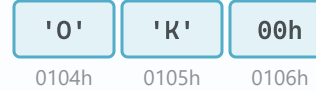
```
include 'emu8086.inc'  
ORG 100h  
    GOTOXY 10, 5  
    PRINT 'Merhaba'  
    PUTC '!'   
RET
```

Makro	İşlev
<code>PUTC ch</code>	Tek karakter yazar
<code>PRINT 'metin'</code>	Metin yazar
<code>PRINTN 'metin'</code>	Metin + yeni satır
<code>GOTOXY x, y</code>	İmleci taşır
<code>CURSOROFF</code>	İmleci gizler

emu8086.inc: PRINT 'OK' Açılımı

```
1 org 100h
▶ push ax          ; PRINT 'OK'
▶ push si
4 jmp ??0003       ; veriyi atla
5 ??0001 db 'OK', 0
6 ??0003: lea si, ??0001
7 ??0000: mov al, cs:[si]
8 cmp al, 0
9 jz  ??0002
10 inc si
11 mov ah, 0Eh
12 int 10h
13 jmp ??0000
14 ??0002: pop si
15 pop ax
16 ret
```

kod kesimindeki dizgi



PRINT 'OK' açıldı. Makro AX ve SI'yı kullanacağı için önce ikisini yığına saklar.

Adım 1 / 10

DEĞİŞKENLER

AL = 00h

SI = 0000h

SP = FFFAh

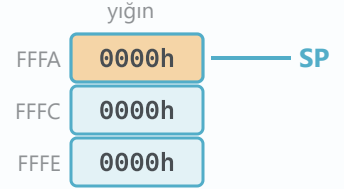
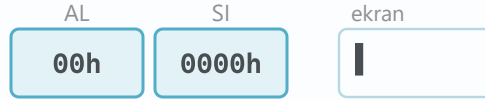
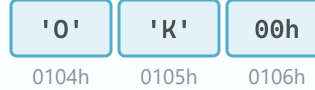
ZF = 0

emu8086.inc'deki PRINT makrosunun açılımı (LOCAL adları derleyicinin ürettiği biçimde). PRINTN aynı işi yapıp sonuna CR LF ekler.

emu8086.inc: PRINT 'OK' Açılımı

```
1 org 100h
2 push ax          ; PRINT 'OK'
3 push si
4 ► jmp ??0003     ; veriyi atla
5 ??0001 db 'OK', 0
6 ??0003: lea si, ??0001
7 ??0000: mov al, cs:[si]
8 cmp al, 0
9 jz  ??0002
10 inc si
11 mov ah, 0Eh
12 int 10h
13 jmp ??0000
14 ??0002: pop si
15 pop ax
16 ret
```

kod kesimindeki dizgi



Dizgi, makronun içinde **kodun arasına** gömülü (0104h: 'O', 'K', 0). JMP, işlemcinin bu baytları komut sanmaması için üzerinden atlar.

Adım 2 / 10

DEĞİŞKENLER

AL = 00h

SI = 0000h

SP = FFFAh

ZF = 0

emu8086.inc'deki PRINT makrosunun açılımı (LOCAL adları derleyicinin ürettiği biçimde). PRINTN aynı işi yapıp sonuna CR LF ekler.

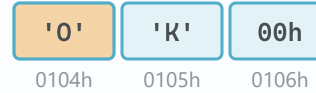
emu8086.inc: PRINT 'OK' Açılımı

```

1 org 100h
2 push ax          ; PRINT 'OK'
3 push si
4 jmp ??0003       ; veriyi atla
5 ??0001 db 'OK', 0
6
7 ??0003: lea si, ??0001
8 mov al, cs:[si]
9 cmp al, 0
10 jz ??0002
11 inc si
12 mov ah, 0Eh
13 int 10h
14 jmp ??0000
15 ??0002: pop si
16 pop ax
17 ret

```

kod kesimindeki dizgi



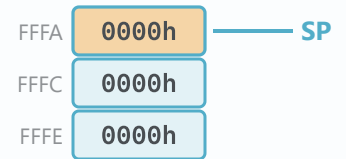
SI



ekran



yığın



SI ← dizginin adresi 0104h. LOCAL adları (??0000...) her PRINT için yenidir.

Adım 3 / 10

DEĞİŞKENLER

AL = 00h

SI = 0104h

SP = FFFAh

ZF = 0

emu8086.inc'deki PRINT makrosunun açılımı (LOCAL adları derleyicinin ürettiği biçimde). PRINTN aynı işi yapıp sonuna CR LF ekler.

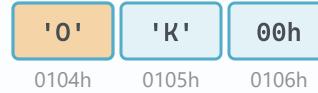
emu8086.inc: PRINT 'OK' Açılımı

```

1 org 100h
2 push ax          ; PRINT 'OK'
3 push si
4 jmp ??0003       ; veriyi atla
5 ??0001 db 'OK', 0
6 ??0003: lea si, ??0001
7   ??0000: mov al, cs:[si]
8   ??0001: cmp al, 0
9   ??0002: jz  ??0002
10 inc si
11 mov ah, 0Eh
12 int 10h
13 jmp ??0000
14 ??0002: pop si
15 pop ax
16 ret

```

kod kesimindeki dizgi



SI

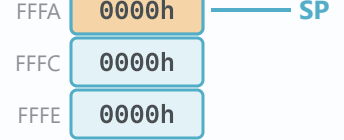
AL

SI

ekran



yığın



AL ← CS:[SI] = 4Fh ('O'). Dizgi kod kesiminde olduğu için **CS:** önceki kullanılır. 0 değil → devam.

Adım 4 / 10

DEĞİŞKENLER

AL = 4Fh

SI = 0104h

SP = FFFAh

ZF = 0

emu8086.inc'deki PRINT makrosunun açılımı (LOCAL adları derleyicinin ürettiği biçimde). PRINTN aynı işi yapıp sonuna CR LF ekler.

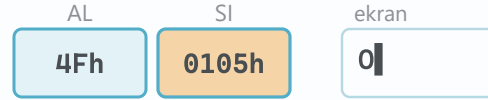
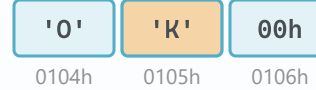
emu8086.inc: PRINT 'OK' Açılımı

```

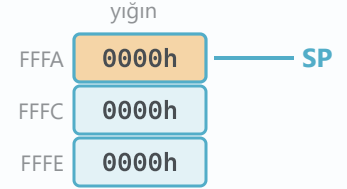
1 org 100h
2 push ax          ; PRINT 'OK'
3 push si
4 jmp ??0003       ; veriyi atla
5 ??0001 db 'OK', 0
6 ??0003: lea si, ??0001
7 ??0000: mov al, cs:[si]
8 cmp al, 0
9 jz  ??0002
▶ inc si
▶ mov ah, 0Eh
▶ int 10h
▶ jmp ??0000
14 ??0002: pop si
15 pop ax
16 ret

```

kod kesimindeki dizgi



SI bir ilerler, INT 10h/0Eh ekrana **O** yazar, JMP döngü başına döner.



Adım 5 / 10

DEĞİŞKENLER

AL = 4Fh

SI = 0105h

SP = FFFAh

ZF = 0

emu8086.inc'deki PRINT makrosunun açılımı (LOCAL adları derleyicinin ürettiği biçimde). PRINTN aynı işi yapıp sonuna CR LF ekler.

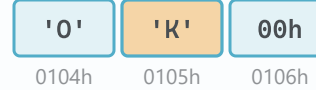
emu8086.inc: PRINT 'OK' Açılımı

```

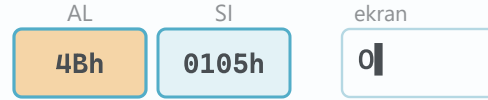
1 org 100h
2 push ax          ; PRINT 'OK'
3 push si
4 jmp ??0003       ; veriyi atla
5 ??0001 db 'OK', 0
6 ??0003: lea si, ??0001
7 ??0000: mov al, cs:[si]
8 ??0000: cmp al, 0
9 ??0000: jz ??0002
10 inc si
11 mov ah, 0Eh
12 int 10h
13 jmp ??0000
14 ??0002: pop si
15 pop ax
16 ret

```

kod kesimindeki dizgi

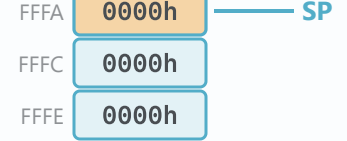


SI



AL = 4Bh ('K'); sonlandırıcı değil.

yığın



Adım 6 / 10

DEĞİŞKENLER

AL = 4Bh

SI = 0105h

SP = FFFAh

ZF = 0

emu8086.inc'deki PRINT makrosunun açılımı (LOCAL adları derleyicinin ürettiği biçimde). PRINTN aynı işi yapıp sonuna CR LF ekler.

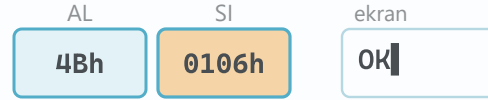
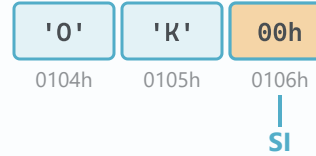
emu8086.inc: PRINT 'OK' Açılımı

```

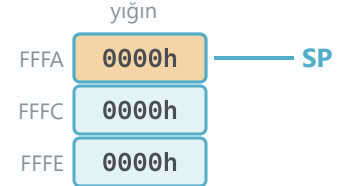
1 org 100h
2 push ax          ; PRINT 'OK'
3 push si
4 jmp ??0003       ; veriyi atla
5 ??0001 db 'OK', 0
6 ??0003: lea si, ??0001
7 ??0000: mov al, cs:[si]
8 cmp al, 0
9 jz  ??0002
▶ inc si
▶ mov ah, 0Eh
▶ int 10h
▶ jmp ??0000
14 ??0002: pop si
15 pop ax
16 ret

```

kod kesimindeki dizgi



Ekranada **OK**.



Adım 7 / 10

DEĞİŞKENLER

AL = 4Bh

SI = 0106h

SP = FFFAh

ZF = 0

emu8086.inc'deki PRINT makrosunun açılımı (LOCAL adları derleyicinin ürettiği biçimde). PRINTN aynı işi yapıp sonuna CR LF ekler.

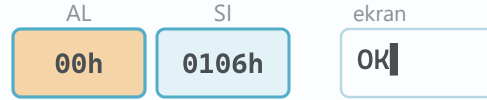
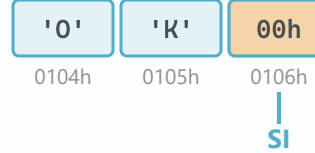
emu8086.inc: PRINT 'OK' Açılımı

```

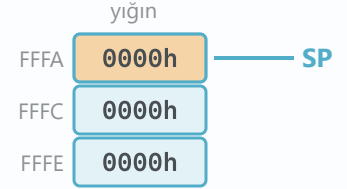
1 org 100h
2 push ax          ; PRINT 'OK'
3 push si
4 jmp ??0003       ; veriyi atla
5 ??0001 db 'OK', 0
6 ??0003: lea si, ??0001
7 ??0000: mov al, cs:[si]
8 ??0001: cmp al, 0
9 ??0002: jz ??0002
10 inc si
11 mov ah, 0Eh
12 int 10h
13 jmp ??0000
14 ??0002: pop si
15 pop ax
16 ret

```

kod kesimindeki dizgi



AL = 00h: CMP → ZF = 1, JZ çıkışa atlar.



Adım 8 / 10

DEĞİŞKENLER

AL = 00h

SI = 0106h

SP = FFFAh

ZF = 1

emu8086.inc'deki PRINT makrosunun açılımı (LOCAL adları derleyicinin ürettiği biçimde). PRINTN aynı işi yapıp sonuna CR LF ekler.

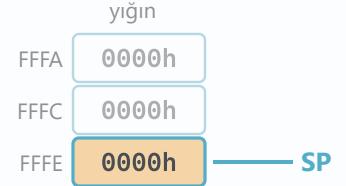
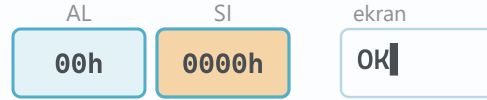
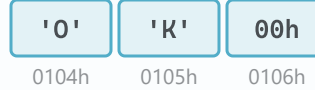
emu8086.inc: PRINT 'OK' Açılımı

```

1 org 100h
2 push ax          ; PRINT 'OK'
3 push si
4 jmp ??0003       ; veriyi atla
5 ??0001 db 'OK', 0
6 ??0003: lea si, ??0001
7 ??0000: mov al, cs:[si]
8 cmp al, 0
9 jz  ??0002
10 inc si
11 mov ah, 0Eh
12 int 10h
13 jmp ??0000
14 ??0002: pop si
15 pop ax
16 ret

```

kod kesimindeki dizgi



SI ve AX yığından geri alınır: makro çağırının yazmaçlarını bozmadı.

Adım 9 / 10

DEĞİŞKENLER

AL = 00h

SI = 0000h

SP = FFEh

ZF = 1

emu8086.inc'deki PRINT makrosunun açılımı (LOCAL adları derleyicinin ürettiği biçimde). PRINTN aynı işi yapıp sonuna CR LF ekler.

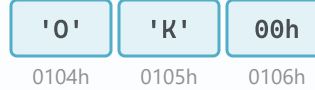
emu8086.inc: PRINT 'OK' Açılımı

```

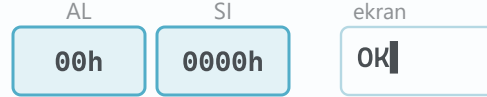
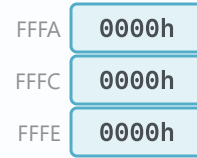
1 org 100h
2 push ax          ; PRINT 'OK'
3 push si
4 jmp ??0003       ; veriyi atla
5 ??0001 db 'OK', 0
6 ??0003: lea si, ??0001
7 ??0000: mov al, cs:[si]
8 cmp al, 0
9 jz  ??0002
10 inc si
11 mov ah, 0Eh
12 int 10h
13 jmp ??0000
14 ??0002: pop si
15 pop ax
▶ ret

```

kod kesimindeki dizgi



yığın



RET: program biter. Tek satırlık PRINT 'OK', 27 baytlık kod + veri üretti.

Adım 10 / 10

DEĞİŞKENLER

AL = 00h

SI = 0000h

SP = 0000h

ZF = 1

emu8086.inc'deki PRINT makrosunun açılımı (LOCAL adları derleyicinin ürettiği biçimde). PRINTN aynı işi yapıp sonuna CR LF ekler.

ÖZET

Özet

Konu	Hatırlanacak nokta
Tanım	<code>ad MACRO p1, p2 ... ENDM</code> ; kullanımdan önce tanımlanır.
Genişletme	Derleme zamanında metin değiştirme; çalışma zamanında CALL yok.
Parametreler	Metin olarak yerine konur; değerleri kullanıldıkları satırda okunur.
Boyut / hız	Makro: her kullanımda gövde kadar büyür, daha hızlıdır. Prosedür: 3 bayt CALL, CALL/RET maliyeti.
LOCAL	Etiketlerin her genişletmede benzersiz olması için gövdenin ilk satırı.
Yan etkiler	Makronun değiştirdiği yazmaçları PUSH/POP ile koruyun (PUTC gibi).
Sınır durumları	LOOP'tan önce CX = 0 kontrolü (JCXZ).

ÖZET

Alıştırmalar

1. `MyMacro AX, BX, CX` çağrısının genişletilmiş hâlini yazın. Hangi satır gereksizdir?
2. İki sözcük değişkeni toplayıp üçüncüsüne yazan `TOPLA a, b, c` makrosunu yazın.
3. PUTC makrosu 50 kez, aynı işi yapan prosedür 50 kez çağrılırsa kod boyutları ne olur?
4. POWER makrosu `POWER base, exponent, result` ile iki kez kullanıldığında hangi hata alınır? Nasıl düzeltilir?
5. `exponent = 0` iken orijinal POWER makrosu kaç kez MUL çalıştırır?
6. Bir makronun AX dışında hiçbir yazmacı bozmaması için hangi komutlar eklenmelidir?

► Cevaplar

Kaynaklar

- emu8086 yardım belgeleri: *8086 Assembler Tutorial, Part 10 (Macros)* ve `emu8086.inc`.
- Microsoft, *MASM Programmer's Guide* — MACRO, ENDM, LOCAL direktifleri.
- K. R. Irvine, *Assembly Language for x86 Processors*, 7. baskı, Pearson, 2014 — Bölüm 10 (Makrolar).
- B. B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson, 2009.

Son

Bölüm 8: Makrolar

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