

Bölüm 8: Yığın

Mikroişlemciler

Yığın ve LIFO

SS:SP

PUSH / POP

Bayt sırası

Yazmaç saklama

Takas

CALL / RET

Parametre geçirme

INT / IRET

Yığın güvenliği

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Yığın modeli

8086 yığınının SS:SP ile nasıl adreslendiğini ve aşağı doğru büyüdüğünü açıklamak.

PUSH / POP

Geçerli işlenen türlerini bilmek ve her komutta SP'nin nasıl değiştiğini adım adım izlemek.

Yazmaç saklama

Yığını yazmaç değerlerini korumak, geri yüklemek ve takas etmek için doğru sırayla kullanmak.

CALL / RET / INT

Dönüş adresinin ve bayrakların yığında nasıl saklandığını, parametrelerin BP ile nasıl okunduğunu göstermek.

Yığın güvenliği

Dengesiz PUSH/POP, taşma ve yanlış sıralama hatalarını tanımak ve önlemek.

emu8086 izleme

Programları tek adım çalıştırırken yazmaç ve yığın pencerelerini yorumlamak.

Yığın (Stack)

- Yığın, **geçici verileri** saklamak için ayrılmış bir bellek alanıdır.
- **CALL** komutu prosedürün **dönüş adresini** yığına koyar; **RET** bu değeri yığından alır ve o adrese döner.
- **INT** bir kesmeyi çağırdığında benzer bir işlem olur: **bayrak (durum) yazmacı, kod kesimi (CS)** ve **bağlı konum (IP)** yığına saklanır; **IRET** ile kesmeden dönülür.
- **PUSH** : 16 bitlik bir değeri yığına koyar.
- **POP** : 16 bitlik bir değeri yığından alır.

Kim kullanır?

Programcı (PUSH/POP), CPU (CALL, INT), işletim sistemi ve BIOS kesme yordamları.

Birim

8086'da yığına her zaman **sözcük** (2 bayt) yazılır ve okunur; **PUSH AL** geçersizdir.

Yön

Yığın yüksek adresten düşük adrese doğru, yani **aşağı doğru** büyür.

LIFO Algoritması

- LIFO, **Last In First Out** (son giren ilk çıkar) ifadesinin kısaltmasıdır.
- Yığına eklenen son öğe, yığından çıkan ilk öğe olur.
- Yığına konan değerler sırasıyla 1, 2, 3, 4, 5 ise ilk alınan değer **5** olur; ardından 4, 3, 2 ve en son 1 alınır.

Sonuç: yığın, değerleri **ters sırada** geri verir. Yazmaç geri yüklerken POP sırası PUSH sırasının tersi olmalıdır.

PUSH



yığın



POP

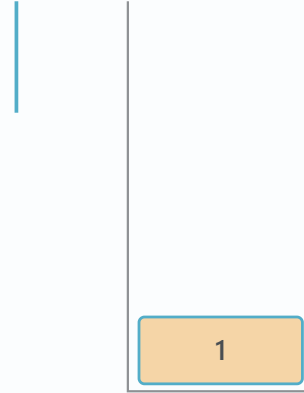
Başlangıçta yığın boş.

LIFO Algoritması

- LIFO, **Last In First Out** (son giren ilk çıkar) ifadesinin kısaltmasıdır.
- Yığına eklenen son öğe, yığından çıkan ilk öğe olur.
- Yığına konan değerler sırasıyla 1, 2, 3, 4, 5 ise ilk alınan değer **5** olur; ardından 4, 3, 2 ve en son 1 alınır.

Sonuç: yığın, değerleri **ters sırada** geri verir. Yazmaç geri yüklerken POP sırası PUSH sırasının tersi olmalıdır.

PUSH



yığın

POP

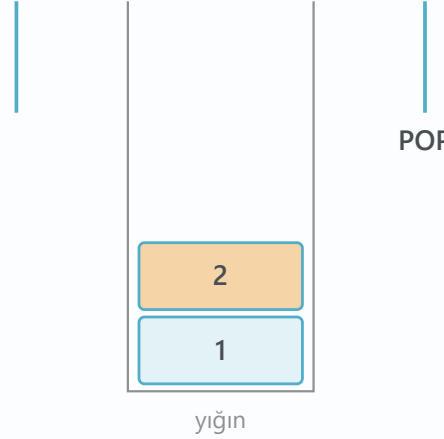
PUSH 1: 1 yığının **üstüne** konur.

LIFO Algoritması

- LIFO, **Last In First Out** (son giren ilk çıkar) ifadesinin kısaltmasıdır.
- Yığına eklenen son öğe, yığından çıkan ilk öğe olur.
- Yığına konan değerler sırasıyla 1, 2, 3, 4, 5 ise ilk alınan değer **5** olur; ardından 4, 3, 2 ve en son 1 alınır.

Sonuç: yığın, değerleri **ters sırada** geri verir. Yazmaç geri yüklerken POP sırası PUSH sırasının tersi olmalıdır.

PUSH



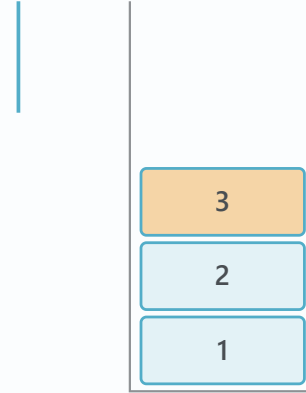
PUSH 2: 2 yığının **üstüne** konur.

LIFO Algoritması

- LIFO, **Last In First Out** (son giren ilk çıkar) ifadesinin kısaltmasıdır.
- Yığına eklenen son öğe, yığından çıkan ilk öğe olur.
- Yığına konan değerler sırasıyla 1, 2, 3, 4, 5 ise ilk alınan değer **5** olur; ardından 4, 3, 2 ve en son 1 alınır.

Sonuç: yığın, değerleri **ters sırada** geri verir. Yazmaç geri yüklerken POP sırası PUSH sırasının tersi olmalıdır.

PUSH



yığın

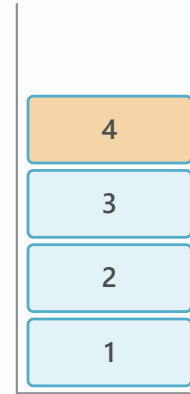
PUSH 3: 3 yığının **üstüne** konur.

LIFO Algoritması

- LIFO, **Last In First Out** (son giren ilk çıkar) ifadesinin kısaltmasıdır.
- Yığına eklenen son öge, yığından çıkan ilk öge olur.
- Yığına konan değerler sırasıyla 1, 2, 3, 4, 5 ise ilk alınan değer **5** olur; ardından 4, 3, 2 ve en son 1 alınır.

Sonuç: yığın, değerleri **ters sırada** geri verir. Yazmaç geri yüklerken POP sırası PUSH sırasının tersi olmalıdır.

PUSH



yığın

POP

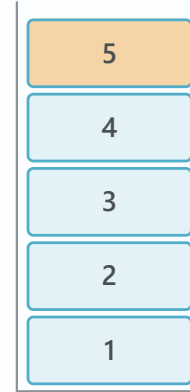
PUSH 4: 4 yığının **üstüne** konur.

LIFO Algoritması

- LIFO, **Last In First Out** (son giren ilk çıkar) ifadesinin kısaltmasıdır.
- Yığına eklenen son öğe, yığından çıkan ilk öğe olur.
- Yığına konan değerler sırasıyla 1, 2, 3, 4, 5 ise ilk alınan değer **5** olur; ardından 4, 3, 2 ve en son 1 alınır.

Sonuç: yığın, değerleri **ters sırada** geri verir. Yazmaç geri yüklerken POP sırası PUSH sırasının tersi olmalıdır.

PUSH



yığın

POP

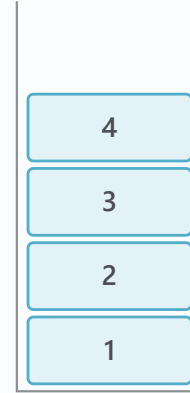
PUSH 5: 5 yığının **üstüne** konur.

LIFO Algoritması

- LIFO, **Last In First Out** (son giren ilk çıkar) ifadesinin kısaltmasıdır.
- Yığına eklenen son öğe, yığından çıkan ilk öğe olur.
- Yığına konan değerler sırasıyla 1, 2, 3, 4, 5 ise ilk alınan değer **5** olur; ardından 4, 3, 2 ve en son 1 alınır.

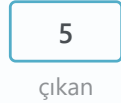
Sonuç: yığın, değerleri **ters sırada** geri verir. Yazmaç geri yüklerken POP sırası PUSH sırasının tersi olmalıdır.

PUSH



yığın

POP



çıkan

POP → **5** çıkar. Çıkış sırası şimdiye kadar: 5

LIFO Algoritması

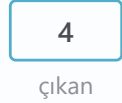
- LIFO, **Last In First Out** (son giren ilk çıkar) ifadesinin kısaltmasıdır.
- Yığına eklenen son öge, yığından çıkan ilk öge olur.
- Yığına konan değerler sırasıyla 1, 2, 3, 4, 5 ise ilk alınan değer **5** olur; ardından 4, 3, 2 ve en son 1 alınır.

Sonuç: yığın, değerleri **ters sırada** geri verir. Yazmaç geri yüklerken POP sırası PUSH sırasının tersi olmalıdır.

PUSH



POP



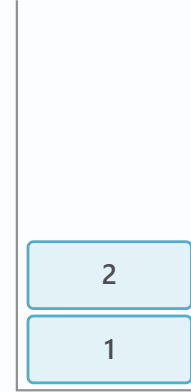
POP → **4** çıkar. Çıkış sırası şimdiye kadar: 5, 4

LIFO Algoritması

- LIFO, **Last In First Out** (son giren ilk çıkar) ifadesinin kısaltmasıdır.
- Yığına eklenen son öge, yığından çıkan ilk öge olur.
- Yığına konan değerler sırasıyla 1, 2, 3, 4, 5 ise ilk alınan değer **5** olur; ardından 4, 3, 2 ve en son 1 alınır.

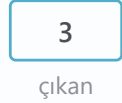
Sonuç: yığın, değerleri **ters sırada** geri verir. Yazmaç geri yüklerken POP sırası PUSH sırasının tersi olmalıdır.

PUSH



yığın

POP



çıkan

POP → **3** çıkar. Çıkış sırası şimdiye kadar: 5, 4, 3

LIFO Algoritması

- LIFO, **Last In First Out** (son giren ilk çıkar) ifadesinin kısaltmasıdır.
- Yığına eklenen son öge, yığından çıkan ilk öge olur.
- Yığına konan değerler sırasıyla 1, 2, 3, 4, 5 ise ilk alınan değer **5** olur; ardından 4, 3, 2 ve en son 1 alınır.

Sonuç: yığın, değerleri **ters sırada** geri verir. Yazmaç geri yüklerken POP sırası PUSH sırasının tersi olmalıdır.

PUSH



yığın



POP



çıkan

POP → **2** çıkar. Çıkış sırası şimdiye kadar: 5, 4, 3, 2

LIFO Algoritması

- LIFO, **Last In First Out** (son giren ilk çıkar) ifadesinin kısaltmasıdır.
- Yığına eklenen son öğe, yığından çıkan ilk öğe olur.
- Yığına konan değerler sırasıyla 1, 2, 3, 4, 5 ise ilk alınan değer **5** olur; ardından 4, 3, 2 ve en son 1 alınır.

Sonuç: yığın, değerleri **ters sırada** geri verir. Yazmaç geri yüklerken POP sırası PUSH sırasının tersi olmalıdır.

PUSH



yığın



POP



çıkan

POP → **1** çıkar. Çıkış sırası şimdiye kadar: 5, 4, 3, 2, 1

Yığın Belleği: SS ve SP

- Yığın belleği **SS** (Stack Segment, yığın kesimi) ve **SP** (Stack Pointer, yığın işaretçisi) yazmaçlarını kullanır.
- SS:SP her zaman yığının **üstündeki** (en son konan) sözcüğü gösterir.
- Başlangıç değerlerini genellikle işletim sistemi (yükleyici) belirler: .COM'da $SS = CS$, $SP = FFFh$; .EXE'de başlıktaki değerler.
- **BP** de varsayılan olarak SS kesimini kullanır; yığındaki verilere erişmek için idealdir.

SS

0700h

Yığın kesimi $SS = 0700h$ (emu8086'da .COM programı için tüm kesim yazmaçları 0700h).

Yığın Belleği: SS ve SP

- Yığın belleği **SS** (Stack Segment, yığın kesimi) ve **SP** (Stack Pointer, yığın işaretçisi) yazmaçlarını kullanır.
- SS:SP her zaman yığının **üstündeki** (en son konan) sözcüğü gösterir.
- Başlangıç değerlerini genellikle işletim sistemi (yükleyici) belirler: .COM'da $SS = CS$, $SP = FFFh$; .EXE'de başlıktaki değerler.
- **BP** de varsayılan olarak SS kesimini kullanır; yığındaki verilere erişmek için idealdir.

SS

0700h

$SS \times 16$

07000h

Kesim değeri 16 ile çarpılır (onaltılıkta sağa bir 0 eklenir): **07000h**.

Yığın Belleği: SS ve SP

- Yığın belleği **SS** (Stack Segment, yığın kesimi) ve **SP** (Stack Pointer, yığın işaretçisi) yazmaçlarını kullanır.
- SS:SP her zaman yığının **üstündeki** (en son konan) sözcüğü gösterir.
- Başlangıç değerlerini genellikle işletim sistemi (yükleyici) belirler: .COM'da $SS = CS$, $SP = FFFh$; .EXE'de başlıktaki değerler.
- **BP** de varsayılan olarak SS kesimini kullanır; yığındaki verilere erişmek için idealdir.

SS	0700h
SS × 16	07000h
+ SP	FFFEh

Bağıl konum olarak yığın işaretçisi $SP = FFFEh$ eklenir.

Yığın Belleği: SS ve SP

- Yığın belleği **SS** (Stack Segment, yığın kesimi) ve **SP** (Stack Pointer, yığın işaretçisi) yazmaçlarını kullanır.
- SS:SP her zaman yığının **üstündeki** (en son konan) sözcüğü gösterir.
- Başlangıç değerlerini genellikle işletim sistemi (yükleyici) belirler: .COM'da SS = CS, SP = FFFh; .EXE'de başlıktaki değerler.
- **BP** de varsayılan olarak SS kesimini kullanır; yığındaki verilere erişmek için idealdir.

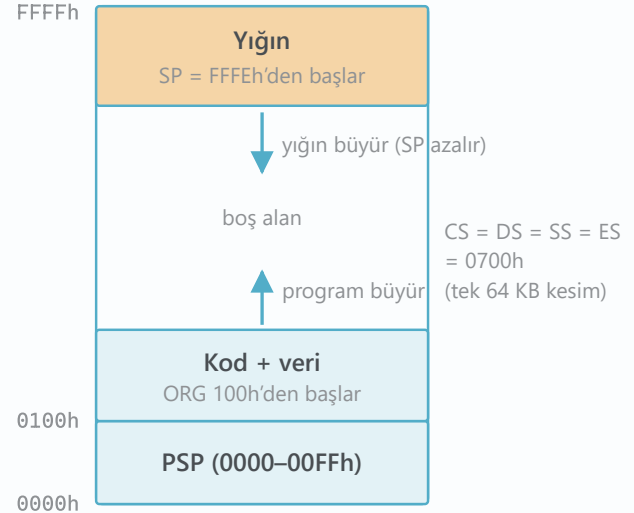
SS	0700h
SS × 16	07000h
+ SP	FFFEh
Fiziksel adres	16FFFEh

Yığının üstü fiziksel olarak **16FFFEh** adresindedir (20 bit).

YIĞIN KAVRAMI

Yığın Aşağı Doğru Büyür

- .COM programında kod, veri ve yığın **aynı 64 KB kesimi** paylaşır.
- Program 0100h'den yukarı doğru, yığın FFFEh'den aşağı doğru yerleşir; aradaki boş alan ikisine de yeter.
- Her PUSH SP'yi **2 azaltır**, her POP **2 artırır**.
- Çok fazla PUSH yapılırsa yığın kodun/verinin üzerine yazabilir: **yığın taşması** (stack overflow).



PUSH VE POP

PUSH Komutu

Biçim	Geçerli işlenenler	Örnek
PUSH REG	AX, BX, CX, DX, DI, SI, BP, SP	PUSH AX
PUSH SREG	DS, ES, SS, CS	PUSH DS
PUSH memory	[BX], [BX+SI+7], 16 bitlik değişken...	PUSH [BX] , PUSH sayı
PUSH immediate	5, -24, 3Fh, 10001101b...	PUSH 3Fh

Dikkat: PUSH immediate komutu 8086/8088'de **yoktur**; 80186 ile gelmiştir. emu8086 bunu kabul eder ama gerçek 8086 için MOV AX, 5 + PUSH AX yazılmalıdır.

Yalnızca **16 bitlik** işlenenler kullanılabilir: PUSH AL , PUSH byte_degisken hatalıdır. Bayraklar için ayrı komut vardır: PUSHF .

Ayrıntı: 8086'da **PUSH SP** , SP azaltıldıktan *sonraki* değeri yığına yazar; 80286 ve sonrası azaltmadan önceki değeri yazar.

PUSH VE POP

POP Komutu

Biçim	Geçerli işlenenler	Örnek
POP REG	AX, BX, CX, DX, DI, SI, BP, SP	POP BX
POP SREG	DS, ES, SS (CS hariç)	POP DS
POP memory	[BX], [BX+SI+7], 16 bitlik değişken...	POP sonuc

Neden CS yok?

CS'yi tek başına değiştirmek, IP aynı kalırken programın akışını rastgele bir yere atar. CS yalnızca RETF, IRET, uzak JMP/CALL ile birlikte değişir.

Sabit yok

POP 5 anlamsızdır: alınan değer bir yere **yazılmalıdır**.

Bayraklar

PUSH/POP bayrakları değiştirmez; yalnızca POPF bayrak yazmacının tamamını yığından yükler.

Yaygın deyim: PUSH CS + POP DS → DS'ye CS'nin değeri kopyalanır (kesim yazmaçları arasında doğrudan MOV yoktur).

PUSH VE POP

PUSH ve POP Nasıl Çalışır?

PUSH kaynak

1. SP yazmacından 2 çıkarılır.
2. Kaynak değeri SS:SP adresine yazılır.

POP hedef

1. SS:SP adresindeki değer hedefe yazılır.
2. SP yazmacına 2 eklenir.

Sıra önemlidir: PUSH önce azaltır sonra yazar; POP önce okur sonra artırır. Böylece SP her an **dolu** olan en üst hücreyi gösterir.



Başlangıç: SP = FFFeh, AX = 1234h.
(.COM'da FFFeh'de DOS'un koyduğu 0000h vardır.)

PUSH VE POP

PUSH ve POP Nasıl Çalışır?

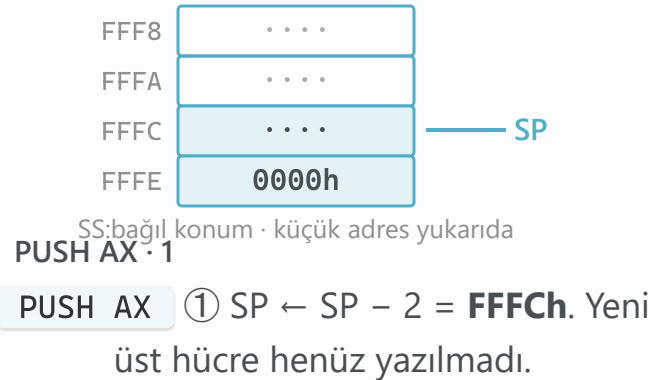
PUSH kaynak

1. SP yazmacından 2 çıkarılır.
2. Kaynak değeri SS:SP adresine yazılır.

POP hedef

1. SS:SP adresindeki değer hedefe yazılır.
2. SP yazmacına 2 eklenir.

Sıra önemlidir: PUSH önce azaltır sonra yazar; POP önce okur sonra artırır. Böylece SP her an **dolu** olan en üst hücreyi gösterir.



PUSH VE POP

PUSH ve POP Nasıl Çalışır?

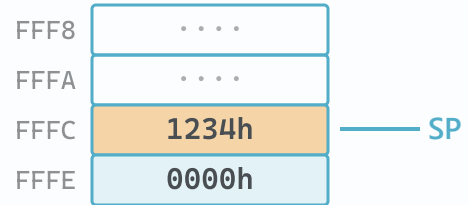
PUSH kaynak

1. SP yazmacından 2 çıkarılır.
2. Kaynak değeri SS:SP adresine yazılır.

POP hedef

1. SS:SP adresindeki değer hedefe yazılır.
2. SP yazmacına 2 eklenir.

Sıra önemlidir: PUSH önce azaltır sonra yazar; POP önce okur sonra artırır. Böylece SP her an **dolu** olan en üst hücreyi gösterir.



SS:bağlı konum · küçük adres yukarıda
PUSH AX · 2

- ② Kaynak değeri SS:SP adresine yazılır:
[FFFCh] ← 1234h.

PUSH ve POP Nasıl Çalışır?

PUSH kaynak

1. SP yazmacından 2 çıkarılır.
2. Kaynak değeri SS:SP adresine yazılır.

POP hedef

1. SS:SP adresindeki değer hedefe yazılır.
2. SP yazmacına 2 eklenir.

Sıra önemlidir: PUSH önce azaltır sonra yazar; POP önce okur sonra artırır. Böylece SP her an **dolu** olan en üst hücreyi gösterir.



PUSH VE POP

PUSH ve POP Nasıl Çalışır?

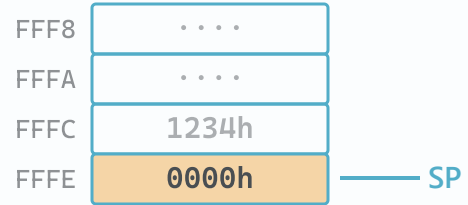
PUSH kaynak

1. SP yazmacından 2 çıkarılır.
2. Kaynak değeri SS:SP adresine yazılır.

POP hedef

1. SS:SP adresindeki değer hedefe yazılır.
2. SP yazmacına 2 eklenir.

Sıra önemlidir: PUSH önce azaltır sonra yazar; POP önce okur sonra artırır. Böylece SP her an **dolu** olan en üst hücreyi gösterir.



SS:bağlı konum · küçük adres yukarıda

POP BX · 2

② $SP \leftarrow SP + 2 = \text{FFFEh}$. FFFCh'deki 1234h **silinmez**; yalnızca artık yığının dışında kalır (soluk).

Adım 5 / 5

PUSH VE POP

Yığında Bayt Sırası (Little-Endian)

Sözcük görünümü

1234h

SS:FFFC

Bayt görünümü (bellek)

34h

FFFC ← düşük bayt (AL)

12h

FFFD ← yüksek bayt (AH)

00h

FFFE

DOS'un 0000h sözcüğü

00h

FFFF

- 8086 bir sözcüğü belleğe **düşük bayt önce** yazar: `PUSH AX` (AX = 1234h) sonrası [FFFCh] = 34h, [FFFDh] = 12h.
- emu8086'nın *stack* penceresi sözcükleri gösterir; *memory* penceresi baytları gösterir. İkisi aynı veriyi farklı biçimde sunar.
- SP her zaman **çift** bir değer olarak tutulmalıdır; tek adreste sözcük erişimi 8086'da ek bir bellek döngüsü harcar.

PUSH VE POP

Ne Değişir, Ne Değişmez?

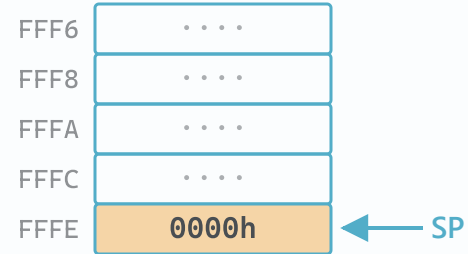
Komut	SP	Bellek	Hedef	Bayraklar
PUSH kaynak	-2	[SS:SP] ← kaynak	—	değişmez
POP hedef	+2	okunur, silinmez	hedef ← [SS:SP]	değişmez
PUSHF	-2	[SS:SP] ← FLAGS	—	değişmez
POPF	+2	okunur	FLAGS ← [SS:SP]	hepsi yüklenir
CALL (yakın)	-2	[SS:SP] ← dönüş IP	IP ← hedef	değişmez
RET (yakın)	+2	okunur	IP ← [SS:SP]	değişmez
INT n	-6	FLAGS, CS, IP	CS:IP ← vektör	IF = TF = 0
IRET	+6	okunur	IP, CS, FLAGS	hepsi yüklenir

Bu yüzden bir karşılaştırmamanın sonucunu (bayrakları) PUSH/POP arasında güvenle taşıyabilirsiniz; bayrağı saklamak gerekiyorsa PUSHF/POPF kullanılır.

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

İlk Örnek: **PUSH AX / POP BX**

```
► ORG    100h
2 MOV    AX, 42      ; AX'e 42 ata
3 PUSH   AX          ; AX'i yığına koy
4 POP    BX          ; yığından BX'e al
5 RET                    ; işletim sistemine
    dön
6 END
```



SS:bağlı konum · küçük adres yukarıda

emu8086 .COM başlangıcı: CS = DS = SS = ES = 0700h,
IP = 0100h, SP = FFFEh.

Adım 1 / 5

DEĞİŞKENLER

AX = 0000h

BX = 0000h

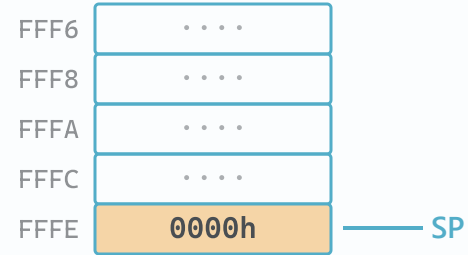
SP = FFFEh

IP = 0100h

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

İlk Örnek: **PUSH AX / POP BX**

1	ORG	100h	
▶	MOV	AX, 42	; AX'e 42 ata
3	PUSH	AX	; AX'i yığına koy
4	POP	BX	; yığından BX'e al
5	RET		; işletim sistemine dön
6	END		



SS:bağlı konum · küçük adres yukarıda

MOV AX, 42 : 42 **onluk** sayıdır → AX = **002Ah** (42h değil!).

Adım 2 / 5

DEĞİŞKENLER

AX = 002Ah

BX = 0000h

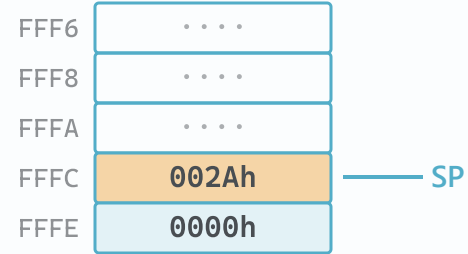
SP = FFFEh

IP = 0103h

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

İlk Örnek: **PUSH AX / POP BX**

```
1 ORG    100h
2 MOV    AX, 42      ; AX'e 42 ata
3 PUSH    AX          ; AX'i yığına koy
4 POP     BX          ; yığından BX'e al
5 RET     ; işletim sistemine
6 END
```



SS:bağlı konum · küçük adres yukarıda

PUSH AX : $SP = FFEh - 2 = \mathbf{FFCh}$; $[FFCh] \leftarrow 002Ah$.

Adım 3 / 5

DEĞİŞKENLER

AX = 002Ah

BX = 0000h

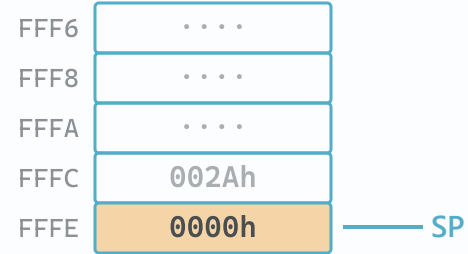
SP = FFFCh

IP = 0104h

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

İlk Örnek: **PUSH AX / POP BX**

```
1 ORG    100h
2 MOV    AX, 42      ; AX'e 42 ata
3 PUSH   AX          ; AX'i yığına koy
4 ► POP   BX          ; yığından BX'e al
5 RET                      ; işletim sistemine
6 END
```



SS:bağlı konum · küçük adres yukarıda

POP BX : $BX \leftarrow [FFFC] = 002Ah$; SP tekrar **FFFEh**.
Değer AX'ten BX'e yığın üzerinden kopyalandı.

Adım 4 / 5

DEĞİŞKENLER

AX = 002Ah

BX = 002Ah

SP = FFEh

IP = 0105h

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

İlk Örnek: **PUSH AX / POP BX**

```
1 ORG    100h
2 MOV    AX, 42      ; AX'e 42 ata
3 PUSH   AX          ; AX'i yığına koy
4 POP    BX          ; yığından BX'e al
▶ RET                ; işletim sistemine
  dön
6 END
```

FFF6	
FFF8	
FFFA	
FFFC	002Ah
FFFE	0000h

SS:bağlı konum · küçük adres yukarıda

RET : yığındaki **0000h** alınır, IP = 0000h → PSP'deki
INT 20h programı sonlandırır.

Adım 5 / 5

DEĞİŞKENLER

AX = 002Ah

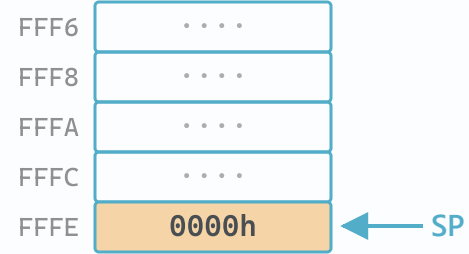
BX = 002Ah

SP = 0000h

IP = 0000h

LIFO Sırası: 1, 2, 3 Girer → 3, 2, 1 Çıkar

```
► MOV    AX, 1      ; AX'e 1 ata
2 PUSH   AX          ; 1'i yığına koy
3 MOV    AX, 2      ; AX'e 2 ata
4 PUSH   AX          ; 2'yi yığına koy
5 MOV    AX, 3      ; AX'e 3 ata
6 PUSH   AX          ; 3'ü yığına koy
7 POP    BX          ; BX ← 3
8 POP    BX          ; BX ← 2
9 POP    BX          ; BX ← 1
```



SS:bağlı konum · küçük adres yukarıda

Başlangıç: SP = FFEh.

Adım 1 / 10

DEĞİŞKENLER

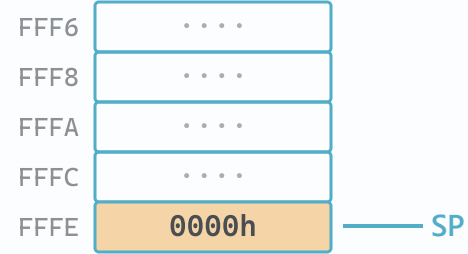
AX = 0000h

BX = 0000h

SP = FFEh

LIFO Sırası: 1, 2, 3 Girer → 3, 2, 1 Çıkar

```
► MOV    AX, 1      ; AX'e 1 ata
2 PUSH   AX          ; 1'i yığına koy
3 MOV    AX, 2      ; AX'e 2 ata
4 PUSH   AX          ; 2'yi yığına koy
5 MOV    AX, 3      ; AX'e 3 ata
6 PUSH   AX          ; 3'ü yığına koy
7 POP    BX          ; BX ← 3
8 POP    BX          ; BX ← 2
9 POP    BX          ; BX ← 1
```



SS:bağlı konum · küçük adres yukarıda

AX = 0001h.

Adım 2 / 10

DEĞİŞKENLER

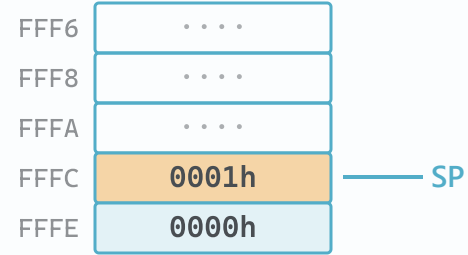
AX = 0001h

BX = 0000h

SP = FFFEh

LIFO Sırası: 1, 2, 3 Girer → 3, 2, 1 Çıkar

1	MOV	AX, 1	; AX'e 1 ata
2	PUSH	AX	; 1'i yığına koy
3	MOV	AX, 2	; AX'e 2 ata
4	PUSH	AX	; 2'yi yığına koy
5	MOV	AX, 3	; AX'e 3 ata
6	PUSH	AX	; 3'ü yığına koy
7	POP	BX	; BX ← 3
8	POP	BX	; BX ← 2
9	POP	BX	; BX ← 1



SS:bağlı konum · küçük adres yukarıda

PUSH: SP = FFFCh, [FFFCh] = 0001h.

Adım 3 / 10

DEĞİŞKENLER

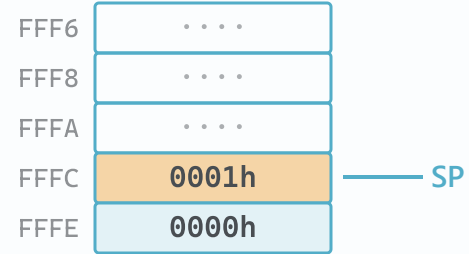
AX = 0001h

BX = 0000h

SP = FFFCh

LIFO Sırası: 1, 2, 3 Girer → 3, 2, 1 Çıkar

1	MOV	AX, 1	; AX'e 1 ata
2	PUSH	AX	; 1'i yığına koy
▶ 3	MOV	AX, 2	; AX'e 2 ata
4	PUSH	AX	; 2'yi yığına koy
5	MOV	AX, 3	; AX'e 3 ata
6	PUSH	AX	; 3'ü yığına koy
7	POP	BX	; BX ← 3
8	POP	BX	; BX ← 2
9	POP	BX	; BX ← 1



SS:bağlı konum · küçük adres yukarıda

AX = 0002h (AX'in eski değeri yığında güvende).

Adım 4 / 10

DEĞİŞKENLER

AX = 0002h

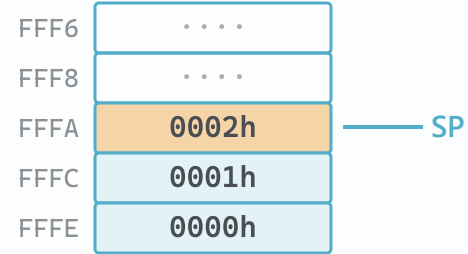
BX = 0000h

SP = FFFCh

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

LIFO Sırası: 1, 2, 3 Girer → 3, 2, 1 Çıkar

1	MOV	AX, 1	; AX'e 1 ata
2	PUSH	AX	; 1'i yığına koy
3	MOV	AX, 2	; AX'e 2 ata
▶	PUSH	AX	; 2'yi yığına koy
5	MOV	AX, 3	; AX'e 3 ata
6	PUSH	AX	; 3'ü yığına koy
7	POP	BX	; BX ← 3
8	POP	BX	; BX ← 2
9	POP	BX	; BX ← 1



SS:bağlı konum · küçük adres yukarıda

PUSH: SP = FFFAh, [FFFAh] = 0002h.

Adım 5 / 10

DEĞİŞKENLER

AX = 0002h

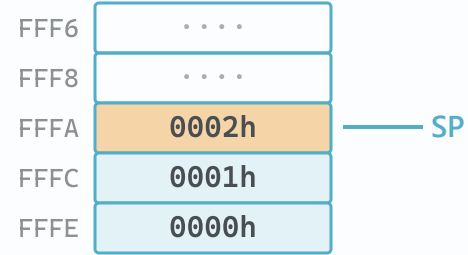
BX = 0000h

SP = FFFAh

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

LIFO Sırası: 1, 2, 3 Girer → 3, 2, 1 Çıkar

1	MOV	AX, 1	; AX'e 1 ata
2	PUSH	AX	; 1'i yığına koy
3	MOV	AX, 2	; AX'e 2 ata
4	PUSH	AX	; 2'yi yığına koy
5	MOV	AX, 3	; AX'e 3 ata
6	PUSH	AX	; 3'ü yığına koy
7	POP	BX	; BX ← 3
8	POP	BX	; BX ← 2
9	POP	BX	; BX ← 1



SS:bağlı konum · küçük adres yukarıda

AX = 0003h.

Adım 6 / 10

DEĞİŞKENLER

AX = 0003h

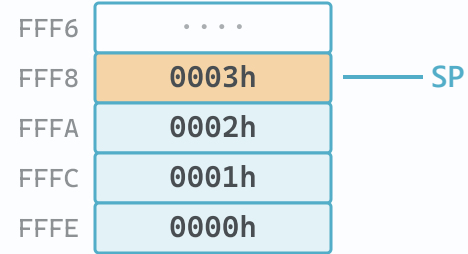
BX = 0000h

SP = FFFAh

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

LIFO Sırası: 1, 2, 3 Girer → 3, 2, 1 Çıkar

1	MOV	AX, 1	; AX'e 1 ata
2	PUSH	AX	; 1'i yığına koy
3	MOV	AX, 2	; AX'e 2 ata
4	PUSH	AX	; 2'yi yığına koy
5	MOV	AX, 3	; AX'e 3 ata
6	PUSH	AX	; 3'ü yığına koy
7	POP	BX	; BX ← 3
8	POP	BX	; BX ← 2
9	POP	BX	; BX ← 1



SS:bağlı konum · küçük adres yukarıda

PUSH: SP = **FFF8h**; yığına üç değer var.

Adım 7 / 10

DEĞİŞKENLER

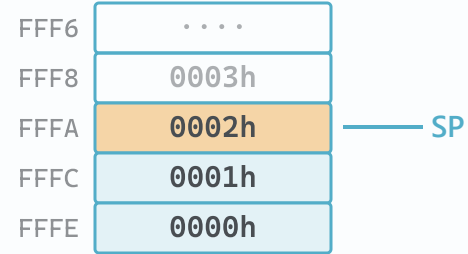
AX = 0003h

BX = 0000h

SP = FFF8h

LIFO Sırası: 1, 2, 3 Girer → 3, 2, 1 Çıkar

1	MOV	AX, 1	; AX'e 1 ata
2	PUSH	AX	; 1'i yığına koy
3	MOV	AX, 2	; AX'e 2 ata
4	PUSH	AX	; 2'yi yığına koy
5	MOV	AX, 3	; AX'e 3 ata
6	PUSH	AX	; 3'ü yığına koy
7	POP	BX	; BX ← 3
8	POP	BX	; BX ← 2
9	POP	BX	; BX ← 1



SS:bağlı konum · küçük adres yukarıda

POP BX → 3 (son giren ilk çıkar). SP = FFFAh.

Adım 8 / 10

DEĞİŞKENLER

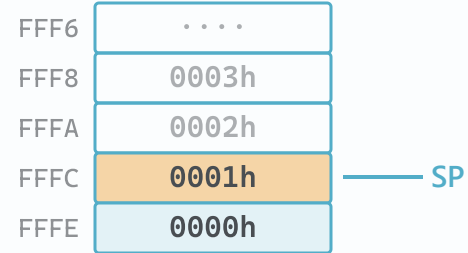
AX = 0003h

BX = 0003h

SP = FFFAh

LIFO Sırası: 1, 2, 3 Girer → 3, 2, 1 Çıkar

1	MOV	AX, 1	; AX'e 1 ata
2	PUSH	AX	; 1'i yığına koy
3	MOV	AX, 2	; AX'e 2 ata
4	PUSH	AX	; 2'yi yığına koy
5	MOV	AX, 3	; AX'e 3 ata
6	PUSH	AX	; 3'ü yığına koy
7	POP	BX	; BX ← 3
8	POP	BX	; BX ← 2
9	POP	BX	; BX ← 1



SS:bağlı konum · küçük adres yukarıda

POP BX → 2. SP = FFFCh.

Adım 9 / 10

DEĞİŞKENLER

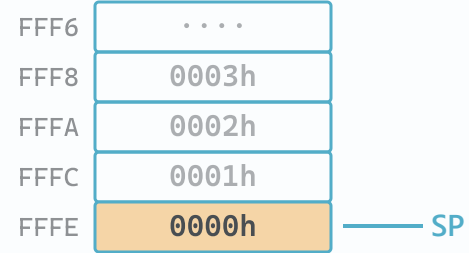
AX = 0003h

BX = 0002h

SP = FFFCh

LIFO Sırası: 1, 2, 3 Girer → 3, 2, 1 Çıkar

```
1 MOV    AX, 1      ; AX'e 1 ata
2 PUSH   AX          ; 1'i yığına koy
3 MOV    AX, 2      ; AX'e 2 ata
4 PUSH   AX          ; 2'yi yığına koy
5 MOV    AX, 3      ; AX'e 3 ata
6 PUSH   AX          ; 3'ü yığına koy
7 POP    BX          ; BX ← 3
8 POP    BX          ; BX ← 2
▶ POP    BX          ; BX ← 1
```



SS:bağlı konum · küçük adres yukarıda

POP BX → 1. SP = FFFEh; yığın başlangıç hâline döndü.

Adım 10 / 10

DEĞİŞKENLER

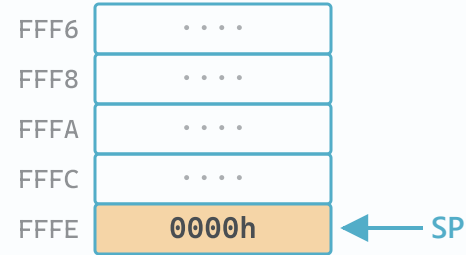
AX = 0003h

BX = 0001h

SP = FFFEh

Bir Yazmacın Değerini Saklamak

```
► ORG    100h
2 MOV    AX, 1234h
3 PUSH   AX          ; AX'in değerini sakla
4 MOV    AX, 5678h   ; AX'i değiştir
5 POP    AX          ; eski değeri geri
   yükle
6 RET
7 END
```



SS:bağlı konum · küçük adres yukarıda

Amaç: AX'i geçici olarak başka bir iş için kullanıp sonra eski değerine döndürmek.

Adım 1 / 6

DEĞİŞKENLER

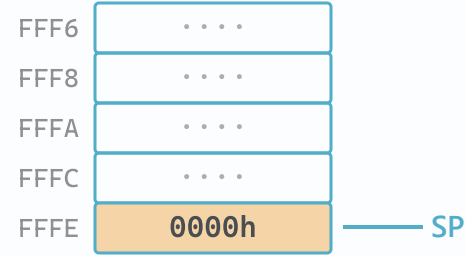
AX = 0000h

SP = FFFEh

Prosedürlerin girişinde kullandıkları yazmaçları PUSH, çıkışta POP etmesi bu desenin aynısıdır.

Bir Yazmacın Değerini Saklamak

```
1  ORG    100h
▶  MOV    AX, 1234h
3  PUSH   AX      ; AX'in değerini sakla
4  MOV     AX, 5678h ; AX'i değiştir
5  POP     AX      ; eski değeri geri
                     yükle
6  RET
7  END
```



SS:bağlı konum · küçük adres yukarıda

AX = 1234h.

Adım 2 / 6

DEĞİŞKENLER

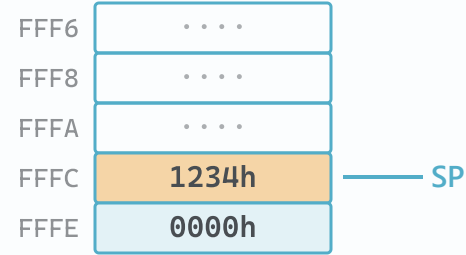
AX = 1234h

SP = FFEh

Prosedürlerin girişinde kullandıkları yazmaçları PUSH, çıkışta POP etmesi bu desenin aynısıdır.

Bir Yazmacın Değerini Saklamak

```
1 ORG    100h
2 MOV    AX, 1234h
3 ► PUSH AX    ; AX'in değerini sakla
4 MOV    AX, 5678h ; AX'i değiştir
5 POP    AX    ; eski değeri geri
           yükle
6 RET
7 END
```



SS:bağlı konum · küçük adres yukarıda

AX yığına saklandı: [FFFC] = 1234h.

Adım 3 / 6

DEĞİŞKENLER

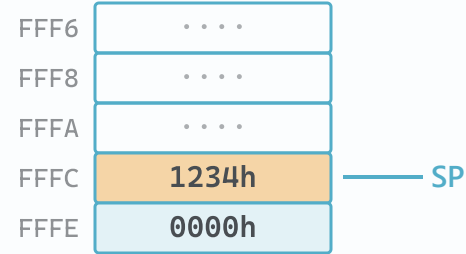
AX = 1234h

SP = FFFCh

Prosedürlerin girişinde kullandıkları yazmaçları PUSH, çıkışta POP etmesi bu desenin aynısıdır.

Bir Yazmacın Değerini Saklamak

```
1 ORG    100h
2 MOV    AX, 1234h
3 PUSH   AX      ; AX'in değerini sakla
4 ► MOV   AX, 5678h ; AX'i değiştir
5 POP    AX      ; eski değeri geri
              yükle
6 RET
7 END
```



SS:bağlı konum · küçük adres yukarıda

AX başka bir iş için kullanıldı: AX = 5678h.

Adım 4 / 6

DEĞİŞKENLER

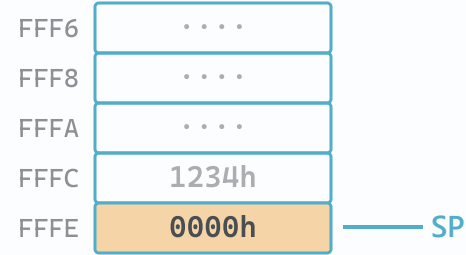
AX = 5678h

SP = FFFCh

Prosedürlerin girişinde kullandıkları yazmaçları PUSH, çıkışta POP etmesi bu desenin aynısıdır.

Bir Yazmacın Değerini Saklamak

```
1 ORG    100h
2 MOV    AX, 1234h
3 PUSH   AX          ; AX'in değerini sakla
4 MOV    AX, 5678h    ; AX'i değiştir
5 ► POP   AX          ; eski değeri geri
   yükle
6 RET
7 END
```



SS:bağlı konum · küçük adres yukarıda

POP AX : AX eski değerine döndü → **1234h**.

Adım 5 / 6

DEĞİŞKENLER

AX = **1234h**

SP = **FFFEh**

Prosedürlerin girişinde kullandıkları yazmaçları PUSH, çıkışta POP etmesi bu desenin aynısıdır.

Bir Yazmacın Değerini Saklamak

```
1 ORG    100h
2 MOV    AX, 1234h
3 PUSH   AX          ; AX'in değerini sakla
4 MOV    AX, 5678h    ; AX'i değiştir
5 POP    AX          ; eski değeri geri
               yükle
▶ RET
7 END
```

FFF6	...
FFF8	...
FFFA	...
FFFC	1234h
FFFE	0000h

SS:bağlı konum · küçük adres yukarıda

RET: 0000h'ye dönüş, program biter.

Adım 6 / 6

DEĞİŞKENLER

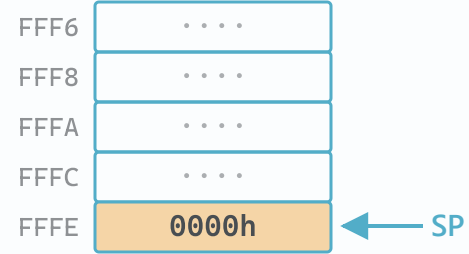
AX = 1234h

SP = 0000h

Prosedürlerin girişinde kullandıkları yazmaçları PUSH, çıkışta POP etmesi bu desenin aynısıdır.

Yığın ile İki Yazmacı Takas Etmek

```
► ORG    100h
2 MOV    AX, 1212h ; AX ← 1212h
3 MOV    BX, 3434h ; BX ← 3434h
4 PUSH   AX        ; AX'i yığına koy
5 PUSH   BX        ; BX'i yığına koy
6 POP    AX        ; AX ← BX'in eski
               değeri
7 POP    BX        ; BX ← AX'in eski
               değeri
8 RET
9 END
```



SS:bağlı konum · küçük adres yukarıda

emu8086'da yükleme sonrası: CX = 000Bh (programın
bayt sayısı: 3+3+1+1+1+1+1+1 = 11), SP = FFFEh.

Adım 1 / 8

DEĞİŞKENLER

AX = 0000h

BX = 0000h

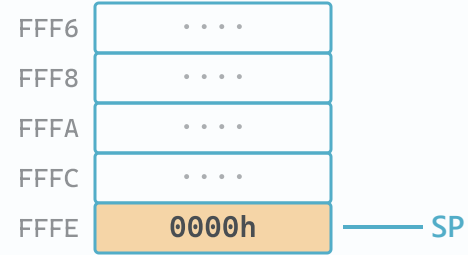
CX = 000Bh

SP = FFFEh

Aynı sırayla POP etseydik (önce BX) değerler yerinde kalırdı. Takasın sırrı LIFO'dur.

Yığın ile İki Yazmacı Takas Etmek

```
1  ORG    100h
▶ 2  MOV   AX, 1212h ; AX ← 1212h
3  MOV   BX, 3434h ; BX ← 3434h
4  PUSH  AX        ; AX'i yığına koy
5  PUSH  BX        ; BX'i yığına koy
6  POP   AX        ; AX ← BX'in eski
                     değeri
7  POP   BX        ; BX ← AX'in eski
                     değeri
8  RET
9  END
```



SS:bağlı konum · küçük adres yukarıda

AX = 1212h.

Adım 2 / 8

DEĞİŞKENLER

AX = 1212h

BX = 0000h

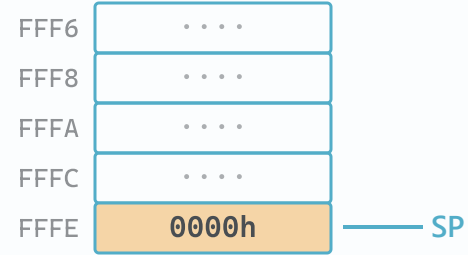
CX = 000Bh

SP = FFEh

Aynı sırayla POP etseydik (önce BX) değerler yerinde kalırdı. Takasın sırrı LIFO'dur.

Yığın ile İki Yazmacı Takas Etmek

```
1  ORG    100h
2  MOV    AX, 1212h ; AX ← 1212h
3  ► MOV    BX, 3434h ; BX ← 3434h
4  PUSH    AX        ; AX'i yığına koy
5  PUSH    BX        ; BX'i yığına koy
6  POP     AX        ; AX ← BX'in eski
                     değeri
7  POP     BX        ; BX ← AX'in eski
                     değeri
8  RET
9  END
```



SS:bağlı konum · küçük adres yukarıda

BX = 3434h.

Adım 3 / 8

DEĞİŞKENLER

AX = 1212h

BX = 3434h

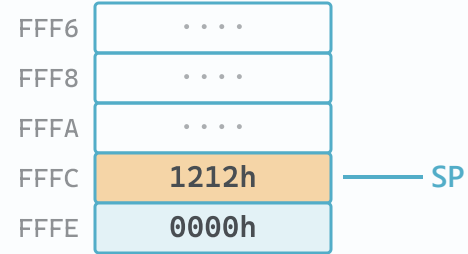
CX = 000Bh

SP = FFEh

Aynı sırayla POP etseydik (önce BX) değerler yerinde kalırdı. Takasın sırrı LIFO'dur.

Yığın ile İki Yazmacı Takas Etmek

```
1  ORG    100h
2  MOV    AX, 1212h ; AX ← 1212h
3  MOV    BX, 3434h ; BX ← 3434h
4  PUSH   AX        ; AX'i yığına koy
5  PUSH   BX        ; BX'i yığına koy
6  POP    AX        ; AX ← BX'in eski
                     değeri
7  POP    BX        ; BX ← AX'in eski
                     değeri
8  RET
9  END
```



SS:bağlı konum · küçük adres yukarıda

PUSH AX: [FFFC] = 1212h, SP = FFFCh.

Adım 4 / 8

DEĞİŞKENLER

AX = 1212h

BX = 3434h

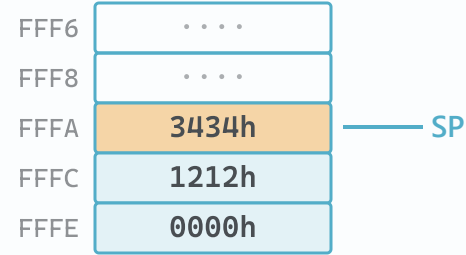
CX = 000Bh

SP = FFFCh

Aynı sırayla POP etseydik (önce BX) değerler yerinde kalırdı. Takasın sırrı LIFO'dur.

Yığın ile İki Yazmacı Takas Etmek

```
1 ORG    100h
2 MOV    AX, 1212h ; AX ← 1212h
3 MOV    BX, 3434h ; BX ← 3434h
4 PUSH   AX        ; AX'i yığına koy
5 PUSH   BX        ; BX'i yığına koy
6 POP    AX        ; AX ← BX'in eski
                     değeri
7 POP    BX        ; BX ← AX'in eski
                     değeri
8 RET
9 END
```



SS:bağlı konum · küçük adres yukarıda

PUSH BX: [FFFAh] = 3434h, SP = FFFAh.

Adım 5 / 8

DEĞİŞKENLER

AX = 1212h

BX = 3434h

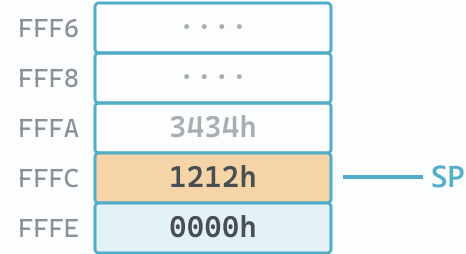
CX = 000Bh

SP = FFFAh

Aynı sırayla POP etseydik (önce BX) değerler yerinde kalırdı. Takasın sırrı LIFO'dur.

Yığın ile İki Yazmacı Takas Etmek

```
1 ORG    100h
2 MOV    AX, 1212h ; AX ← 1212h
3 MOV    BX, 3434h ; BX ← 3434h
4 PUSH   AX        ; AX'i yığına koy
5 PUSH   BX        ; BX'i yığına koy
6 ► POP   AX        ; AX ← BX'in eski
   değeri
7 POP    BX        ; BX ← AX'in eski
   değeri
8 RET
9 END
```



SS:bağlı konum · küçük adres yukarıda

POP AX: üstteki **3434h** AX'e gelir.

Adım 6 / 8

DEĞİŞKENLER

AX = 3434h

BX = 3434h

CX = 000Bh

SP = FFFCh

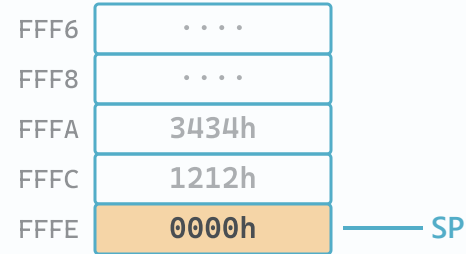
Aynı sırayla POP etseydik (önce BX) değerler yerinde kalırdı. Takasın sırrı LIFO'dur.

Yığın ile İki Yazmacı Takas Etmek

```

1  ORG    100h
2  MOV    AX, 1212h ; AX ← 1212h
3  MOV    BX, 3434h ; BX ← 3434h
4  PUSH   AX        ; AX'i yığına koy
5  PUSH   BX        ; BX'i yığına koy
6  POP    AX        ; AX ← BX'in eski
                     değeri
7  ► POP   BX        ; BX ← AX'in eski
                     değeri
8  RET
9  END

```



SS:bağlı konum · küçük adres yukarıda

POP BX: sıradaki **1212h** BX'e gelir. Takas tamam!

Adım 7 / 8

DEĞİŞKENLER

AX = 3434h

BX = 1212h

CX = 000Bh

SP = FFFEh

Aynı sırayla POP etseydik (önce BX) değerler yerinde kalırdı. Takasın sırrı LIFO'dur.

Yığın ile İki Yazmacı Takas Etmek

```
1 ORG    100h
2 MOV    AX, 1212h ; AX ← 1212h
3 MOV    BX, 3434h ; BX ← 3434h
4 PUSH   AX        ; AX'i yığına koy
5 PUSH   BX        ; BX'i yığına koy
6 POP    AX        ; AX ← BX'in eski
                değeri
7 POP    BX        ; BX ← AX'in eski
                değeri
8 RET
9 END
```

FFF6	...
FFF8	...
FFFA	3434h
FFFC	1212h
FFFE	0000h

SS:bağlı konum · küçük adres yukarıda

RET: program sonu.

Adım 8 / 8

DEĞİŞKENLER

AX = 3434h

BX = 1212h

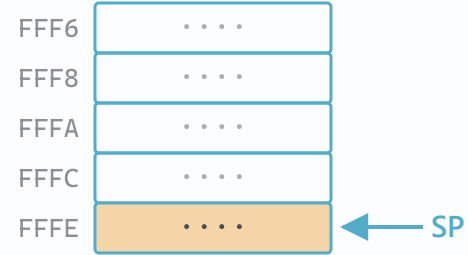
CX = 000Bh

SP = 0000h

Aynı sırayla POP etseydik (önce BX) değerler yerinde kalırdı. Takasın sırrı LIFO'dur.

Yığın Kullanımı: Üç Yazmaç

```
► mov ax, 2
2 mov bx, 4
3 mov cx, 8
4 push ax
5 push bx
6 push cx
7 pop ax
8 pop bx
9 pop cx
```



SS:bağlı konum · küçük adres yukarıda

.BIN olarak yüklendi: CS = SS = 0100h, IP = 0000h, SP = FFEh. Yığın başlangıçta boş.

Adım 1 / 10

DEĞİŞKENLER

AX = 0000h

BX = 0000h

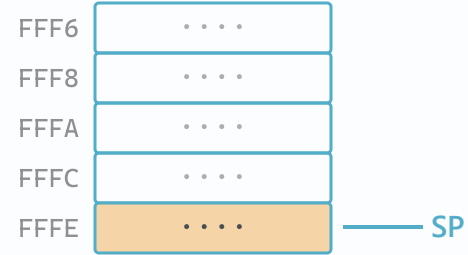
CX = 0000h

SP = FFEh

Kaynak: emu8086 ile tek adım çalıştırma (single step). Yığın penceresi SS:SP çevresini gösterir.

Yığın Kullanımı: Üç Yazmaç

```
► mov ax, 2
2 mov bx, 4
3 mov cx, 8
4 push ax
5 push bx
6 push cx
7 pop ax
8 pop bx
9 pop cx
```



SS:bağlı konum · küçük adres yukarıda

AX = 0002h.

Adım 2 / 10

DEĞİŞKENLER

AX = 0002h

BX = 0000h

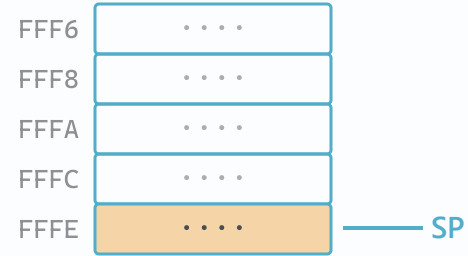
CX = 0000h

SP = FFFEh

Kaynak: emu8086 ile tek adım çalıştırma (single step). Yığın penceresi SS:SP çevresini gösterir.

Yığın Kullanımı: Üç Yazmaç

```
1 mov ax, 2
▶ mov bx, 4
3 mov cx, 8
4 push ax
5 push bx
6 push cx
7 pop ax
8 pop bx
9 pop cx
```



SS:bağlı konum · küçük adres yukarıda

BX = 0004h.

Adım 3 / 10

DEĞİŞKENLER

AX = 0002h

BX = 0004h

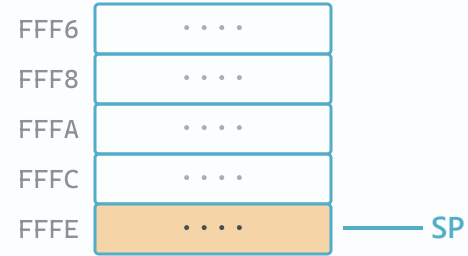
CX = 0000h

SP = FFFEh

Kaynak: emu8086 ile tek adım çalıştırma (single step). Yığın penceresi SS:SP çevresini gösterir.

Yığın Kullanımı: Üç Yazmaç

```
1 mov ax, 2
2 mov bx, 4
3 ► mov cx, 8
4 push ax
5 push bx
6 push cx
7 pop ax
8 pop bx
9 pop cx
```



SS:bağlı konum · küçük adres yukarıda

CX = 0008h.

Adım 4 / 10

DEĞİŞKENLER

AX = 0002h

BX = 0004h

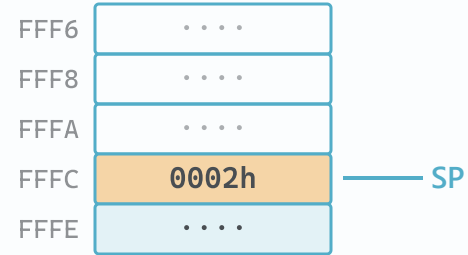
CX = 0008h

SP = FFFEh

Kaynak: emu8086 ile tek adım çalıştırma (single step). Yığın penceresi SS:SP çevresini gösterir.

Yığın Kullanımı: Üç Yazmaç

```
1 mov ax, 2
2 mov bx, 4
3 mov cx, 8
4 ► push ax
5 push bx
6 push cx
7 pop ax
8 pop bx
9 pop cx
```



SS:bağlı konum · küçük adres yukarıda

push ax: SP = FFFCh, [FFFC] = 0002h.

Adım 5 / 10

DEĞİŞKENLER

AX = 0002h

BX = 0004h

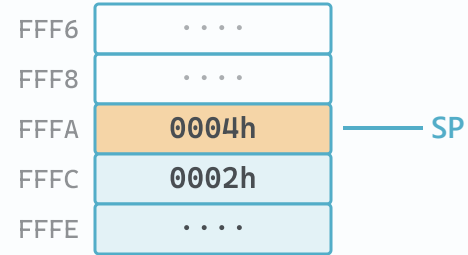
CX = 0008h

SP = FFFCh

Kaynak: emu8086 ile tek adım çalıştırma (single step). Yığın penceresi SS:SP çevresini gösterir.

Yığın Kullanımı: Üç Yazmaç

```
1 mov ax, 2
2 mov bx, 4
3 mov cx, 8
4 push ax
5 push bx
6 push cx
7 pop ax
8 pop bx
9 pop cx
```



SS:bağlı konum · küçük adres yukarıda
push bx: SP = FFFAh, [FFFAh] = 0004h.

Adım 6 / 10

DEĞİŞKENLER

AX = 0002h

BX = 0004h

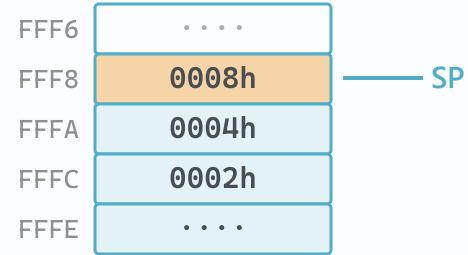
CX = 0008h

SP = FFFAh

Kaynak: emu8086 ile tek adım çalıştırma (single step). Yığın penceresi SS:SP çevresini gösterir.

Yığın Kullanımı: Üç Yazmaç

```
1 mov ax, 2
2 mov bx, 4
3 mov cx, 8
4 push ax
5 push bx
▶ push cx
7 pop ax
8 pop bx
9 pop cx
```



SS:bağlı konum · küçük adres yukarıda

push cx: SP = **FFF8h**, [FFF8h] = 0008h.

Adım 7 / 10

DEĞİŞKENLER

AX = 0002h

BX = 0004h

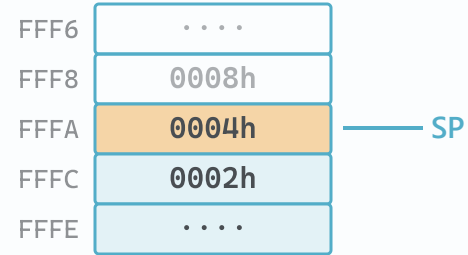
CX = 0008h

SP = **FFF8h**

Kaynak: emu8086 ile tek adım çalıştırma (single step). Yığın penceresi SS:SP çevresini gösterir.

Yığın Kullanımı: Üç Yazmaç

```
1 mov ax, 2
2 mov bx, 4
3 mov cx, 8
4 push ax
5 push bx
6 push cx
7 ► pop ax
8 pop bx
9 pop cx
```



SS:bağlı konum · küçük adres yukarıda
pop ax: AX ← 0008h (üstteki).

Adım 8 / 10

DEĞİŞKENLER

AX = 0008h

BX = 0004h

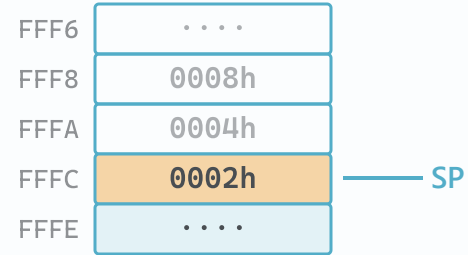
CX = 0008h

SP = FFFAh

Kaynak: emu8086 ile tek adım çalıştırma (single step). Yığın penceresi SS:SP çevresini gösterir.

Yığın Kullanımı: Üç Yazmaç

```
1 mov ax, 2
2 mov bx, 4
3 mov cx, 8
4 push ax
5 push bx
6 push cx
7 pop ax
8 pop bx
9 pop cx
```



SS:bağlı konum · küçük adres yukarıda
pop bx: BX ← 0004h (ortadaki; değişmedi).

Adım 9 / 10

DEĞİŞKENLER

AX = 0008h

BX = 0004h

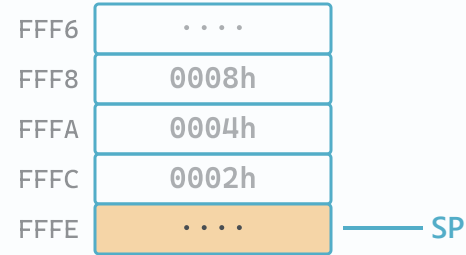
CX = 0008h

SP = FFFCh

Kaynak: emu8086 ile tek adım çalıştırma (single step). Yığın penceresi SS:SP çevresini gösterir.

Yığın Kullanımı: Üç Yazmaç

```
1 mov ax, 2
2 mov bx, 4
3 mov cx, 8
4 push ax
5 push bx
6 push cx
7 pop ax
8 pop bx
▶ pop cx
```



SS:bağlı konum · küçük adres yukarıda

pop cx: CX ← 0002h. Sonuç: AX ile CX'in değerleri **yer değiştirdi**.

Adım 10 / 10

DEĞİŞKENLER

AX = 0008h

BX = 0004h

CX = 0002h

SP = FFFEh

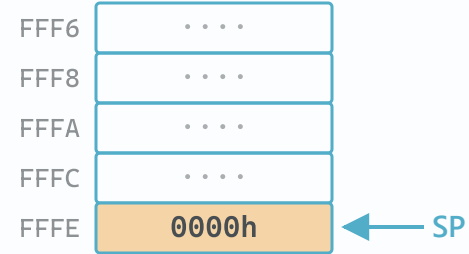
Kaynak: emu8086 ile tek adım çalıştırma (single step). Yığın penceresi SS:SP çevresini gösterir.

Yığınla Diziyi Ters Çevirme

```

► ORG 100h
2 MOV SI, OFFSET metin
3 MOV CX, 3
4 koy: MOV AL, [SI]
5      PUSH AX      ; karakteri sakla
6      INC SI
7      LOOP koy
8 MOV CX, 3
9 yaz: POP DX      ; DL = karakter
10     MOV AH, 2
11     INT 21h      ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC'   ; 0117h

```



SS:bağlı konum · küçük adres yukarıda

metin = 'ABC' bellekte 0117h adresinde. Hedef: ekrana ters sırada yazmak.

Adım 1 / 29

DEĞİŞKENLER

AX = 0000h

CX = 0000h

DX = 0000h

SI = 0000h

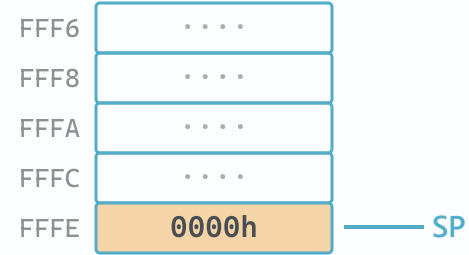
SP = FFEh

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1  ORG  100h
2  ► MOV  SI, OFFSET  metin
3  MOV  CX, 3
4  koy:  MOV  AL, [SI]
5        PUSH AX          ; karakteri sakla
6        INC  SI
7        LOOP koy
8  MOV  CX, 3
9  yaz:  POP  DX          ; DL = karakter
10      MOV  AH, 2
11      INT  21h          ; DL'yi ekrana yaz
12      LOOP yaz
13  RET
14  metin DB 'ABC'      ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

SI = **0117h** (metin dizisinin adresi).

Adım 2 / 29

DEĞİŞKENLER

AX = 0000h

CX = 0000h

DX = 0000h

SI = **0117h**

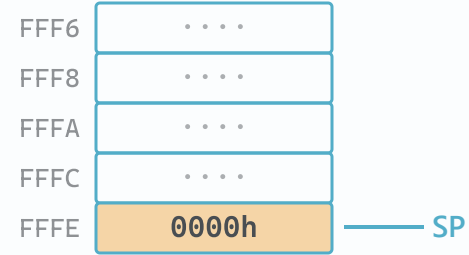
SP = FFEh

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  ► MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX      ; karakteri sakla
6       INC SI
7       LOOP koy
8  MOV CX, 3
9  yaz: POP DX      ; DL = karakter
10      MOV AH, 2
11      INT 21h     ; DL'yi ekrana yaz
12      LOOP yaz
13  RET
14  metin DB 'ABC' ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

Döngü sayacı CX = 3.

Adım 3 / 29

DEĞİŞKENLER

AX = 0000h

CX = 0003h

DX = 0000h

SI = 0117h

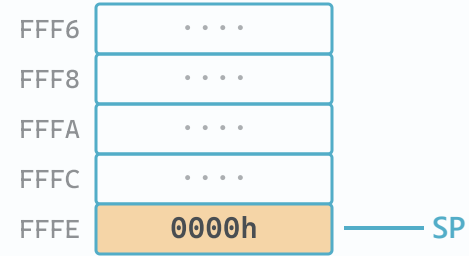
SP = FFEh

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
▶ koy: MOV AL, [SI]
5      PUSH AX      ; karakteri sakla
6      INC SI
7      LOOP koy
8  MOV CX, 3
9  yaz: POP DX      ; DL = karakter
10     MOV AH, 2
11     INT 21h      ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC'  ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

$AL \leftarrow [SI] = 'A' (41h)$.

Adım 4 / 29

DEĞİŞKENLER

AX = 0041h

CX = 0003h

DX = 0000h

SI = 0117h

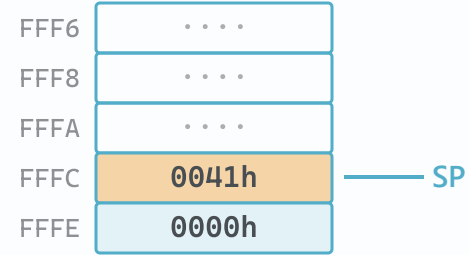
SP = FFEh

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5  ► PUSH AX ; karakteri sakla
6      INC SI
7      LOOP koy
8  MOV CX, 3
9  yaz: POP DX ; DL = karakter
10     MOV AH, 2
11     INT 21h ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC' ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

PUSH AX: 'A' yığına (AH = 00h ile birlikte, 0041h).

Adım 5 / 29

DEĞİŞKENLER

AX = 0041h

CX = 0003h

DX = 0000h

SI = 0117h

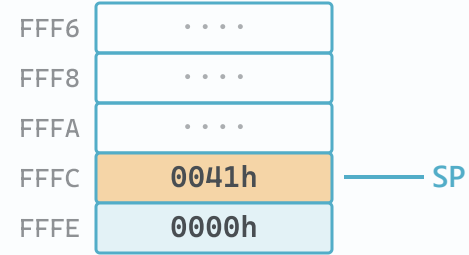
SP = FFFCh

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1 ORG 100h
2 MOV SI, OFFSET metin
3 MOV CX, 3
4 koy: MOV AL, [SI]
5 PUSH AX ; karakteri sakla
6 INC SI
7 LOOP koy
8 MOV CX, 3
9 yaz: POP DX ; DL = karakter
10 MOV AH, 2
11 INT 21h ; DL'yi ekrana yaz
12 LOOP yaz
13 RET
14 metin DB 'ABC' ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

SI = 0118h: sonraki karakter.

Adım 6 / 29

DEĞİŞKENLER

AX = 0041h

CX = 0003h

DX = 0000h

SI = 0118h

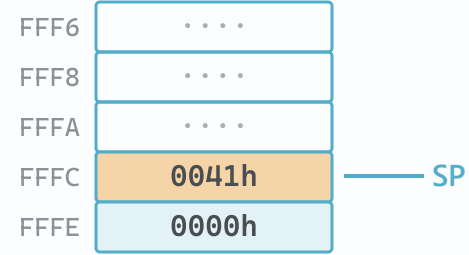
SP = FFFCh

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1 ORG 100h
2 MOV SI, OFFSET metin
3 MOV CX, 3
4 koy: MOV AL, [SI]
5      PUSH AX      ; karakteri sakla
6      INC SI
7      LOOP koy
8 MOV CX, 3
9 yaz: POP DX      ; DL = karakter
10     MOV AH, 2
11     INT 21h      ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC' ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

LOOP: CX = 2 → geri dön.

Adım 7 / 29

DEĞİŞKENLER

AX = 0041h

CX = 0002h

DX = 0000h

SI = 0118h

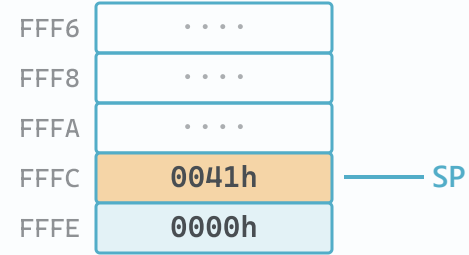
SP = FFFCh

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  ► koy: MOV AL, [SI]
5      PUSH AX      ; karakteri sakla
6      INC SI
7  LOOP koy
8  MOV CX, 3
9  yaz: POP DX      ; DL = karakter
10     MOV AH, 2
11     INT 21h      ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC'  ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

$AL \leftarrow [SI] = 'B' (42h).$

Adım 8 / 29

DEĞİŞKENLER

AX = 0042h

CX = 0002h

DX = 0000h

SI = 0118h

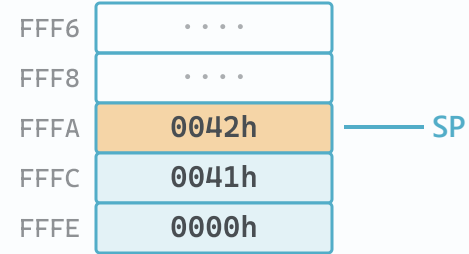
SP = FFFCh

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5  ► PUSH AX ; karakteri sakla
6      INC SI
7      LOOP koy
8  MOV CX, 3
9  yaz: POP DX ; DL = karakter
10     MOV AH, 2
11     INT 21h ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC' ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

PUSH AX: 'B' yığına (AH = 00h ile birlikte, 0042h).

Adım 9 / 29

DEĞİŞKENLER

AX = 0042h

CX = 0002h

DX = 0000h

SI = 0118h

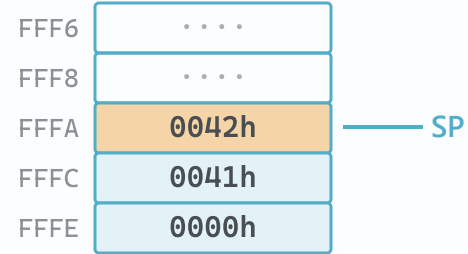
SP = FFFAh

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5      PUSH AX      ; karakteri sakla
6      INC SI
7      LOOP koy
8  MOV CX, 3
9  yaz: POP DX      ; DL = karakter
10     MOV AH, 2
11     INT 21h      ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC'   ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

SI = 0119h: sonraki karakter.

Adım 10 / 29

DEĞİŞKENLER

AX = 0042h

CX = 0002h

DX = 0000h

SI = 0119h

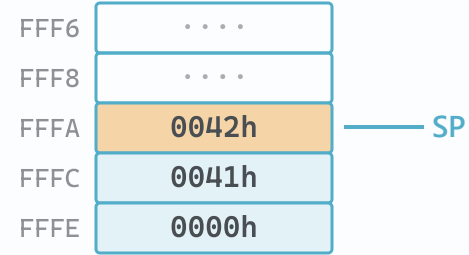
SP = FFFAh

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1 ORG 100h
2 MOV SI, OFFSET metin
3 MOV CX, 3
4 koy: MOV AL, [SI]
5      PUSH AX      ; karakteri sakla
6      INC SI
7      LOOP koy
8 MOV CX, 3
9 yaz: POP DX      ; DL = karakter
10     MOV AH, 2
11     INT 21h      ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC' ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

LOOP: CX = 1 → geri dön.

Adım 11 / 29

DEĞİŞKENLER

AX = 0042h

CX = 0001h

DX = 0000h

SI = 0119h

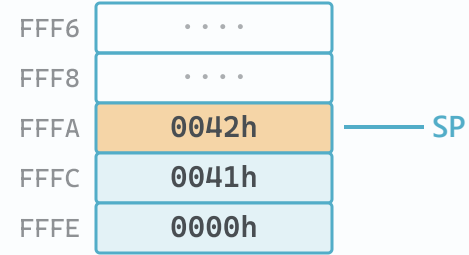
SP = FFFAh

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
▶ koy: MOV AL, [SI]
5    PUSH AX      ; karakteri sakla
6    INC SI
7    LOOP koy
8  MOV CX, 3
9  yaz: POP DX      ; DL = karakter
10     MOV AH, 2
11     INT 21h      ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC'  ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

$AL \leftarrow [SI] = 'C' (43h).$

Adım 12 / 29

DEĞİŞKENLER

AX = 0043h

CX = 0001h

DX = 0000h

SI = 0119h

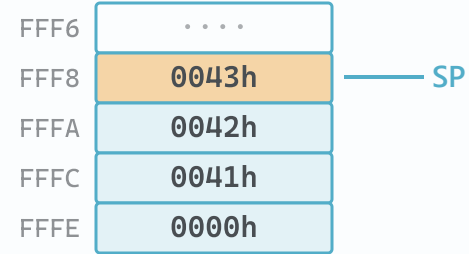
SP = FFFAh

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5  PUSH AX ; karakteri sakla
6  INC SI
7  LOOP koy
8  MOV CX, 3
9  yaz: POP DX ; DL = karakter
10 MOV AH, 2
11 INT 21h ; DL'yi ekrana yaz
12 LOOP yaz
13 RET
14 metin DB 'ABC' ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

PUSH AX: 'C' yığına (AH = 00h ile birlikte, 0043h).

Adım 13 / 29

DEĞİŞKENLER

AX = 0043h

CX = 0001h

DX = 0000h

SI = 0119h

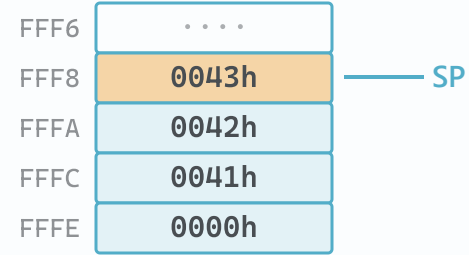
SP = FFF8h

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5      PUSH AX      ; karakteri sakla
6      INC SI
7      LOOP koy
8  MOV CX, 3
9  yaz: POP DX      ; DL = karakter
10     MOV AH, 2
11     INT 21h      ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC'   ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

SI = 011Ah: sonraki karakter.

Adım 14 / 29

DEĞİŞKENLER

AX = 0043h

CX = 0001h

DX = 0000h

SI = 011Ah

SP = FFF8h

ÇIKTI

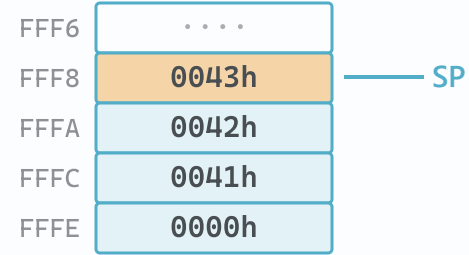


Yığınla Diziyi Ters Çevirme

```

1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX      ; karakteri sakla
6       INC SI
7  ► LOOP koy
8  MOV CX, 3
9  yaz: POP DX      ; DL = karakter
10      MOV AH, 2
11      INT 21h     ; DL'yi ekrana yaz
12      LOOP yaz
13  RET
14  metin DB 'ABC' ; 0117h

```



SS:bağlı konum · küçük adres yukarıda

LOOP: CX = 0 → döngü biter.

Adım 15 / 29

DEĞİŞKENLER

AX = 0043h

CX = 0000h

DX = 0000h

SI = 011Ah

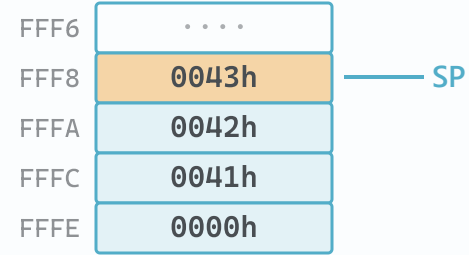
SP = FFF8h

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1 ORG 100h
2 MOV SI, OFFSET metin
3 MOV CX, 3
4 koy: MOV AL, [SI]
5      PUSH AX      ; karakteri sakla
6      INC SI
7      LOOP koy
8
9 MOV CX, 3
10 yaz: POP DX      ; DL = karakter
11      MOV AH, 2
12      INT 21h     ; DL'yi ekrana yaz
13      LOOP yaz
14 RET
15 metin DB 'ABC'   ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

İkinci döngü için CX = 3.

Adım 16 / 29

DEĞİŞKENLER

AX = 0043h

CX = 0003h

DX = 0000h

SI = 011Ah

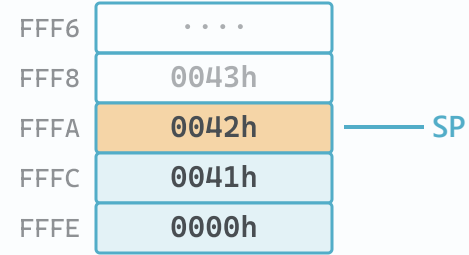
SP = FFF8h

ÇIKTI



Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX      ; karakteri sakla
6       INC SI
7       LOOP koy
8  MOV CX, 3
9  ► yaz: POP DX      ; DL = karakter
10      MOV AH, 2
11      INT 21h      ; DL'yi ekrana yaz
12      LOOP yaz
13  RET
14  metin DB 'ABC'   ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

POP DX: DX = 0043h → DL = 'C' (son giren ilk çıkar).

Adım 17 / 29

DEĞİŞKENLER

AX = 0043h

CX = 0003h

DX = 0043h

SI = 011Ah

SP = FFFAh

ÇIKTI

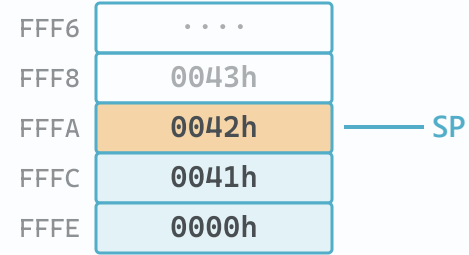


Yığınla Diziyi Ters Çevirme

```

1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX      ; karakteri sakla
6       INC SI
7       LOOP koy
8  MOV CX, 3
9  yaz: POP DX      ; DL = karakter
10     ► MOV AH, 2
11     INT 21h      ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC'   ; 0117h

```



SS:bağlı konum · küçük adres yukarıda

AH = 02h: DOS karakter yazma işlevi.

Adım 18 / 29

DEĞİŞKENLER

AX = 0243h

CX = 0003h

DX = 0043h

SI = 011Ah

SP = FFFAh

ÇIKTI

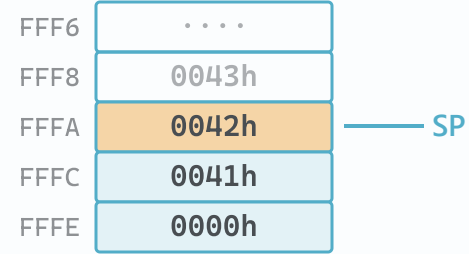


Yığınla Diziyi Ters Çevirme

```

1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX          ; karakteri sakla
6       INC SI
7       LOOP koy
8  MOV CX, 3
9  yaz: POP DX           ; DL = karakter
10     MOV AH, 2
11     ► INT 21h         ; DL'yi ekrana yaz
12     LOOP yaz
13  RET
14  metin DB 'ABC'      ; 0117h

```



SS:bağlı konum · küçük adres yukarıda

INT 21h: ekrana **C** yazılır.

Adım 19 / 29

DEĞİŞKENLER

AX = 0243h

CX = 0003h

DX = 0043h

SI = 011Ah

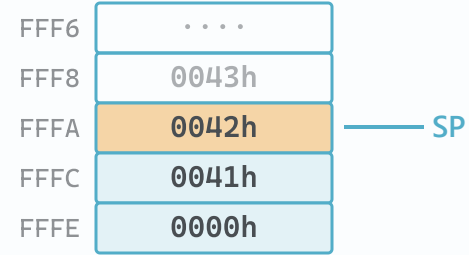
SP = FFFAh

ÇIKTI

C

Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX          ; karakteri sakla
6       INC SI
7       LOOP koy
8  MOV CX, 3
9  yaz: POP DX           ; DL = karakter
10     MOV AH, 2
11     INT 21h          ; DL'yi ekrana yaz
12     LOOP yaz
13  RET
14  metin DB 'ABC'      ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

LOOP: CX = 2 → geri dön.

Adım 20 / 29

DEĞİŞKENLER

AX = 0243h

CX = 0002h

DX = 0043h

SI = 011Ah

SP = FFFAh

ÇIKTI

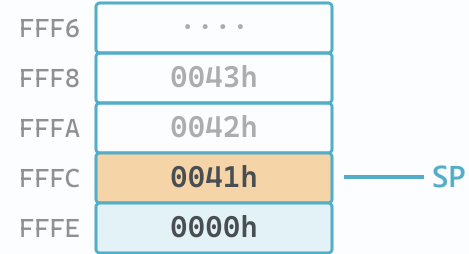
C

Yığınla Diziyi Ters Çevirme

```

1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX      ; karakteri sakla
6       INC SI
7       LOOP koy
8  MOV CX, 3
9  ► yaz: POP DX      ; DL = karakter
10      MOV AH, 2
11      INT 21h      ; DL'yi ekrana yaz
12      LOOP yaz
13  RET
14  metin DB 'ABC'   ; 0117h

```



SS:bağlı konum · küçük adres yukarıda

POP DX: DX = 0042h → DL = 'B' (son giren ilk çıkar).

Adım 21 / 29

DEĞİŞKENLER

AX = 0243h

CX = 0002h

DX = 0042h

SI = 011Ah

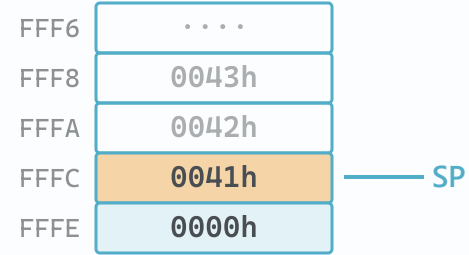
SP = FFFCh

ÇIKTI

C

Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX      ; karakteri sakla
6       INC SI
7       LOOP koy
8  MOV CX, 3
9  yaz: POP DX      ; DL = karakter
10     MOV AH, 2
11     INT 21h      ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC'   ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

AH = 02h: DOS karakter yazma işlevi.

Adım 22 / 29

DEĞİŞKENLER

AX = 0243h

CX = 0002h

DX = 0042h

SI = 011Ah

SP = FFFCh

ÇIKTI

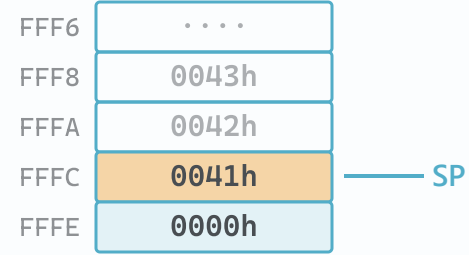
C

Yığınla Diziyi Ters Çevirme

```

1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX          ; karakteri sakla
6       INC SI
7       LOOP koy
8  MOV CX, 3
9  yaz: POP DX           ; DL = karakter
10     MOV AH, 2
11     ► INT 21h         ; DL'yi ekrana yaz
12     LOOP yaz
13  RET
14  metin DB 'ABC'      ; 0117h

```



SS:bağlı konum · küçük adres yukarıda

INT 21h: ekrana **B** yazılır.

Adım 23 / 29

DEĞİŞKENLER

AX = 0243h

CX = 0002h

DX = 0042h

SI = 011Ah

SP = FFFCh

ÇIKTI

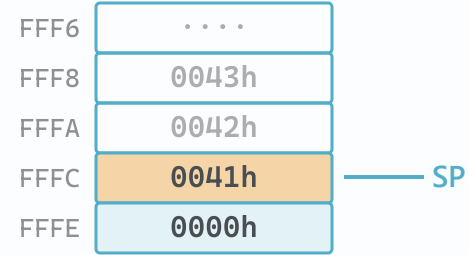
CB

Yığınla Diziyi Ters Çevirme

```

1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5      PUSH AX          ; karakteri sakla
6      INC SI
7      LOOP koy
8  MOV CX, 3
9  yaz: POP DX          ; DL = karakter
10     MOV AH, 2
11     INT 21h          ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC'      ; 0117h

```



SS:bağlı konum · küçük adres yukarıda

LOOP: CX = 1 → geri dön.

Adım 24 / 29

DEĞİŞKENLER

AX = 0243h

CX = 0001h

DX = 0042h

SI = 011Ah

SP = FFFCh

ÇIKTI

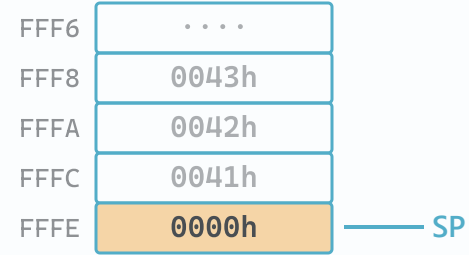
CB

Yığınla Diziyi Ters Çevirme

```

1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX      ; karakteri sakla
6       INC SI
7       LOOP koy
8  MOV CX, 3
9  ► yaz: POP DX      ; DL = karakter
10      MOV AH, 2
11      INT 21h      ; DL'yi ekrana yaz
12      LOOP yaz
13  RET
14  metin DB 'ABC'   ; 0117h

```



SS:bağlı konum · küçük adres yukarıda

POP DX: DX = 0041h → DL = 'A' (son giren ilk çıkar).

Adım 25 / 29

DEĞİŞKENLER

AX = 0243h

CX = 0001h

DX = 0041h

SI = 011Ah

SP = FFEh

ÇIKTI

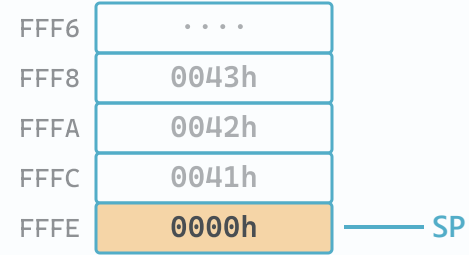
CB

Yığınla Diziyi Ters Çevirme

```

1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX      ; karakteri sakla
6       INC SI
7       LOOP koy
8  MOV CX, 3
9  yaz: POP DX      ; DL = karakter
10     ► MOV AH, 2
11     INT 21h      ; DL'yi ekrana yaz
12     LOOP yaz
13 RET
14 metin DB 'ABC'   ; 0117h

```



SS:bağlı konum · küçük adres yukarıda

AH = 02h: DOS karakter yazma işlevi.

Adım 26 / 29

DEĞİŞKENLER

AX = 0243h

CX = 0001h

DX = 0041h

SI = 011Ah

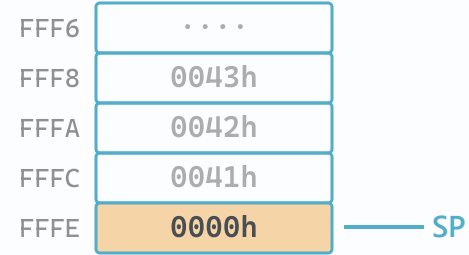
SP = FFEh

ÇIKTI

CB

Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX      ; karakteri sakla
6       INC SI
7       LOOP koy
8  MOV CX, 3
9  yaz: POP DX      ; DL = karakter
10     MOV AH, 2
11     ► INT 21h    ; DL'yi ekrana yaz
12     LOOP yaz
13  RET
14  metin DB 'ABC' ; 0117h
```



SS:bağlı konum · küçük adres yukarıda

INT 21h: ekrana **A** yazılır.

Adım 27 / 29

DEĞİŞKENLER

AX = 0243h

CX = 0001h

DX = 0041h

SI = 011Ah

SP = FFEh

ÇIKTI

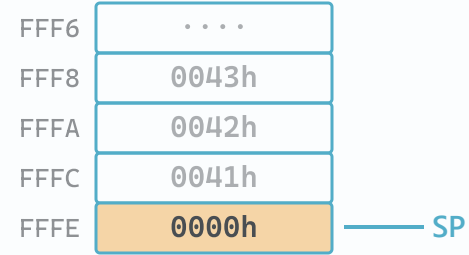
CBA

Yığınla Diziyi Ters Çevirme

```

1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX      ; karakteri sakla
6       INC SI
7       LOOP koy
8  MOV CX, 3
9  yaz: POP DX      ; DL = karakter
10      MOV AH, 2
11      INT 21h      ; DL'yi ekrana yaz
12      LOOP yaz
13  RET
14  metin DB 'ABC'    ; 0117h

```



SS:bağlı konum · küçük adres yukarıda

LOOP: CX = 0 → döngü biter.

Adım 28 / 29

DEĞİŞKENLER

AX = 0243h

CX = 0000h

DX = 0041h

SI = 011Ah

SP = FFEh

ÇIKTI

CBA

Yığınla Diziyi Ters Çevirme

```
1  ORG 100h
2  MOV SI, OFFSET metin
3  MOV CX, 3
4  koy: MOV AL, [SI]
5       PUSH AX      ; karakteri sakla
6       INC SI
7       LOOP koy
8  MOV CX, 3
9  yaz: POP DX      ; DL = karakter
10      MOV AH, 2
11      INT 21h      ; DL'yi ekrana yaz
12      LOOP yaz
▶ RET
14 metin DB 'ABC'    ; 0117h
```

FFF6	
FFF8	0043h
FFFA	0042h
FFFC	0041h
FFFE	0000h

SS:bağlı konum · küçük adres yukarıda

Ekranda **CBA**: dizi yığın sayesinde ters çevrildi. RET ile program biter.

Adım 29 / 29

DEĞİŞKENLER

AX = 0243h

CX = 0000h

DX = 0041h

SI = 011Ah

SP = 0000h

ÇIKTI

CBA

ÖRNEKLER

Tahmin Et: Yazmaçlar Ne Olur?

Kod çalıştıktan sonra AX, BX, CX ve SP ne olur? (Başlangıç: SP = FFFEh)

Adım 1 / 5

```
MOV AX, 1111h
MOV BX, 2222h
MOV CX, 3333h
PUSH AX
PUSH BX
PUSH CX
POP AX
POP CX
PUSH AX
POP BX
POP BX
```

ÖRNEKLER

Tahmin Et: Yazmaçlar Ne Olur?

Kod çalıştıktan sonra AX, BX, CX ve SP ne olur? (Başlangıç: SP = FFFEh)

```
MOV AX, 1111h
MOV BX, 2222h
MOV CX, 3333h
PUSH AX
PUSH BX
PUSH CX
POP AX
POP CX
PUSH AX
POP BX
POP BX
```

PUSH'lardan sonra yığın (üstten alta): 3333h, 2222h, 1111h; SP = FFF8h.

Adım 2 / 5

ÖRNEKLER

Tahmin Et: Yazmaçlar Ne Olur?

Kod çalıştıktan sonra AX, BX, CX ve SP ne olur? (Başlangıç: SP = FFFEh)

```
MOV AX, 1111h
MOV BX, 2222h
MOV CX, 3333h
PUSH AX
PUSH BX
PUSH CX
POP AX
POP CX
PUSH AX
POP BX
POP BX
```

PUSH'lardan sonra yığın (üstten alta): 3333h, 2222h, 1111h; SP = FFF8h.

POP AX → AX = **3333h**; POP CX → CX = **2222h**;
SP = FFFCh.

Adım 3 / 5

ÖRNEKLER

Tahmin Et: Yazmaçlar Ne Olur?

Kod çalıştıktan sonra AX, BX, CX ve SP ne olur? (Başlangıç: SP = FFFEh)

```
MOV AX, 1111h
MOV BX, 2222h
MOV CX, 3333h
PUSH AX
PUSH BX
PUSH CX
POP AX
POP CX
PUSH AX
POP BX
POP BX
```

PUSH'lardan sonra yığın (üstten alta): 3333h, 2222h, 1111h; SP = FFF8h.

POP AX → AX = **3333h**; POP CX → CX = **2222h**;
SP = FFFCh.

PUSH AX (3333h), POP BX → BX = 3333h;
ardından POP BX → BX = **1111h**.

Adım 4 / 5

ÖRNEKLER

Tahmin Et: Yazmaçlar Ne Olur?

Kod çalıştıktan sonra AX, BX, CX ve SP ne olur? (Başlangıç: SP = FFFEh)

```
MOV AX, 1111h
MOV BX, 2222h
MOV CX, 3333h
PUSH AX
PUSH BX
PUSH CX
POP AX
POP CX
PUSH AX
POP BX
POP BX
```

PUSH'lardan sonra yığın (üstten alta): 3333h, 2222h, 1111h; SP = FFF8h.

POP AX → AX = **3333h**; POP CX → CX = **2222h**;
SP = FFFCh.

PUSH AX (3333h), POP BX → BX = 3333h;
ardından POP BX → BX = **1111h**.

Sonuç: AX = 3333h, BX = 1111h, CX = 2222h, SP = **FFFEh** (4 PUSH, 4 POP → dengeli).

Takas Yöntemlerinin Karşılaştırması

Yığın ile

```
PUSH AX  
PUSH BX  
POP AX  
POP BX
```

4 bayt · 4 bellek erişimi. Her yazmaç çifti ve kesim yazmaçları için çalışır.

Geçici yazmaç ile

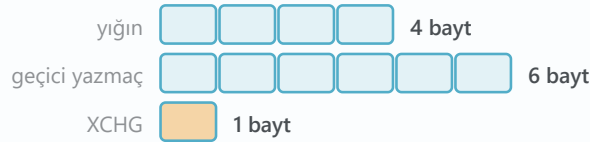
```
MOV CX, AX  
MOV AX, BX  
MOV BX, CX
```

6 bayt · bellek erişimi yok, ama CX'in değeri **bozulur**.

XCHG ile

```
XCHG AX, BX
```

1 bayt (93h) · en kısa ve en hızlı yol. Kesim yazmaçlarıyla kullanılamaz.



Yığın yönteminin asıl değeri takas değil, **saklamadır**: yazmaçları prosedür ya da kesme süresince korumak, ara sonuçları geçici olarak tutmak, kesim yazmaçlarını kopyalamak (PUSH DS / POP ES).

ÖRNEKLER

Etkileşimli Yığın Simülatörü

Yazmaçlar (hex, düzenlenebilir)

AX 1212

BX 3434

CX 0008

SP FFFEh

PUSH AX

PUSH BX

PUSH CX

POP AX

POP BX

POP CX

sıfırla

SS:SP çevresi

FFF4:

FFF6:

FFF8:

FFFA:

FFFC:

FFFE: 0000h ←SP

Çalıştırılan komutlar

; .COM: [FFFEh] = 0000h (DOS)

SP = **FFFEh**. Bir PUSH ya da POP düğmesine basın.

Yığın boşken POP yapılırsa gerçek 8086 bunu engellemez: SP yine 2 artar ve kesimdeki rastgele değer okunur.

CALL ve RET

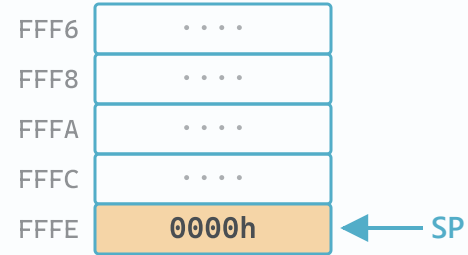
- `CALL hedef` (yakın): **bir sonraki komutun** adresini (dönüş IP'si) yığına koyar, sonra $IP \leftarrow \text{hedef}$.
- `RET` : yığının üstündeki sözcüğü IP'ye alır; program CALL'dan sonraki komuttan devam eder.
- **Uzak** (far) CALL önce CS'yi sonra IP'yi koyar; `RETF` ikisini ters sırada alır.
- `RET n` : dönüşten sonra SP'ye ayrıca n ekler; yığına konmuş parametreleri temizlemek için kullanılır.
- İç içe çağrılarda her CALL yığına yeni bir dönüş adresi ekler; RET'ler bunları **ters sırada** tüketir (LIFO).

Komut	Yığına	SP
<code>CALL near</code>	IP	-2
<code>CALL far</code>	CS, IP	-4
<code>RET</code>	$\leftarrow IP$	+2
<code>RETF</code>	$\leftarrow IP, CS$	+4
<code>RET 4</code>	$\leftarrow IP$	+2+4

CALL / RET · KODU ADIM ADIM ÇALIŞTIR

CALL / RET Adım Adım

```
► ORG 100h      ; 0100h
2 MOV AX, 5      ; 0100h
3 CALL KARE      ; 0103h
4 RET           ; 0106h
5 KARE PROC
6 MUL AX         ; 0107h  DX:AX = AX*AX
7 RET           ; 0109h
8 KARE ENDP
```



SS:bağlı konum · küçük adres yukarıda

Başlangıç: IP = 0100h, SP = FFEh.

Adım 1 / 6

DEĞİŞKENLER

AX = 0000h

DX = 0000h

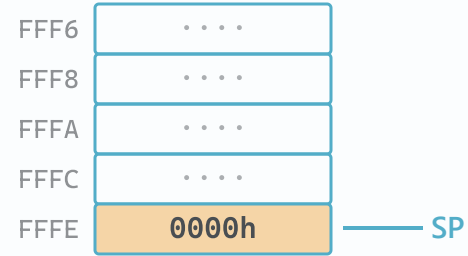
SP = FFEh

IP = 0100h

CALL / RET · KODU ADIM ADIM ÇALIŞTIR

CALL / RET Adım Adım

```
1  ORG  100h      ; 0100h
▶  MOV  AX, 5      ; 0100h
3  CALL KARE      ; 0103h
4  RET           ; 0106h
5  KARE  PROC
6  MUL  AX         ; 0107h  DX:AX = AX*AX
7  RET           ; 0109h
8  KARE  ENDP
```



SS:bağlı konum · küçük adres yukarıda

AX = 0005h; IP bir sonraki komutu (0103h) gösterir.

Adım 2 / 6

DEĞİŞKENLER

AX = 0005h

DX = 0000h

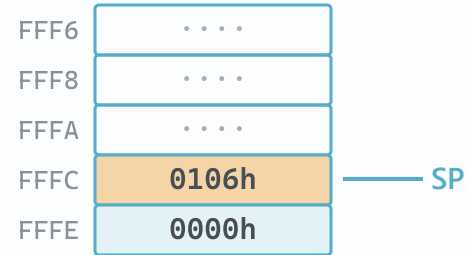
SP = FFFEh

IP = 0103h

CALL / RET · KODU ADIM ADIM ÇALIŞTIR

CALL / RET Adım Adım

```
1  ORG  100h      ; 0100h
2  MOV  AX, 5      ; 0100h
3  ► CALL KARE     ; 0103h
4  RET            ; 0106h
5  KARE PROC
6  MUL  AX          ; 0107h  DX:AX = AX*AX
7  RET            ; 0109h
8  KARE  ENDP
```



SS:bağlı konum · küçük adres yukarıda

CALL KARE : dönüş adresi **0106h** (CALL 3 bayt: 0103h + 3) yığına konur; IP = 0107h.

Adım 3 / 6

DEĞİŞKENLER

AX = 0005h

DX = 0000h

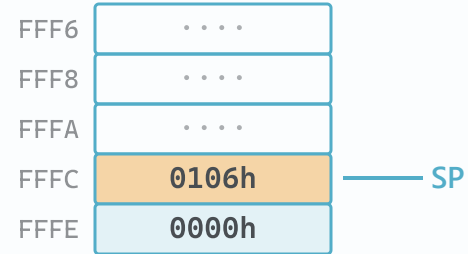
SP = FFFCh

IP = 0107h

CALL / RET · KODU ADIM ADIM ÇALIŞTIR

CALL / RET Adım Adım

```
1  ORG  100h      ; 0100h
2  MOV  AX, 5      ; 0100h
3  CALL KARE      ; 0103h
4  RET            ; 0106h
5  KARE  PROC
▶  MUL  AX        ; 0107h  DX:AX = AX*AX
7  RET            ; 0109h
8  KARE  ENDP
```



SS:bağlı konum · küçük adres yukarıda

MUL AX : DX:AX = $5 \times 5 = 0019h$ (25). DX = 0000h.

Adım 4 / 6

DEĞİŞKENLER

AX = 0019h

DX = 0000h

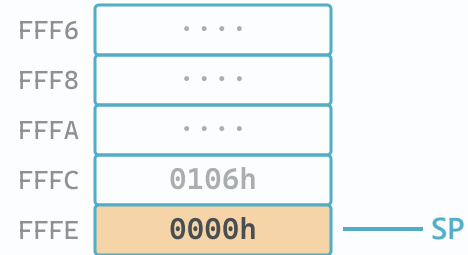
SP = FFFCh

IP = 0109h

CALL / RET · KODU ADIM ADIM ÇALIŞTIR

CALL / RET Adım Adım

```
1 ORG 100h      ; 0100h
2 MOV AX, 5      ; 0100h
3 CALL KARE      ; 0103h
4 RET            ; 0106h
5 KARE PROC
6 MUL AX          ; 0107h  DX:AX = AX*AX
7 RET            ; 0109h
8 KARE ENDP
```



SS:bağlı konum · küçük adres yukarıda

RET : IP ← [FFFC] = **0106h**; SP = FFEh. Ana programa dönüldü.

Adım 5 / 6

DEĞİŞKENLER

AX = 0019h

DX = 0000h

SP = FFEh

IP = 0106h

CALL / RET · KODU ADIM ADIM ÇALIŞTIR

CALL / RET Adım Adım

```
1  ORG  100h      ; 0100h
2  MOV  AX, 5      ; 0100h
3  CALL KARE       ; 0103h
4  ► RET          ; 0106h
5  KARE PROC
6  MUL  AX          ; 0107h  DX:AX = AX*AX
7  RET            ; 0109h
8  KARE  ENDP
```

FFF6	...
FFF8	...
FFFA	...
FFFC	0106h
FFFE	0000h

SS:bağlı konum · küçük adres yukarıda

Ana programın RET 'i 0000h'yi alır → program biter.

Adım 6 / 6

DEĞİŞKENLER

AX = 0019h

DX = 0000h

SP = 0000h

IP = 0000h

Prosedürde Yazmaç Koruma

```
TOPLA_N PROC      ; AX = 1+2+...+CX
    PUSH BX      ; BX'i koru
    PUSH CX      ; CX'i koru
    XOR AX, AX
    MOV BX, 1
dongu:  ADD AX, BX
    INC BX
    LOOP dongu
    POP CX      ; ters sıra!
    POP BX
    RET
TOPLA_N ENDP
```

- Prosedür, kullandığı yazmaçları girişte yığına koyar, çıkışta **ters sırada** geri alır. Çağırın taraf yazmaçlarının bozulmayacağını bilir.
- Dönüş adresi bu PUSH'ların **altında** kalır; RET'e gelindiğinde tüm PUSH'lar POP edilmiş olmalıdır.
- Sonucu döndüren yazmaç (burada AX) korunmaz; aksi hâlde sonuç kaybolur.

RET anında yığının üstünde dönüş adresi yerine CX olsaydı, program CX'in değerindeki adrese **atlardı**.

Kod İzleme: TOPLA_N ve Yazmaç Koruma

► **MOV** BX, 9 ; çağıranın BX'i

► **MOV** CX, 4 ; n = 4

3 **CALL** TOPLA_N ; AX = 1+2+...+n

4 **RET**

5 TOPLA_N **PROC**

6 **PUSH** BX ; BX'i koru

7 **PUSH** CX ; CX'i koru

8 **XOR** AX, AX

9 **MOV** BX, 1

10 dongu: **ADD** AX, BX

11 **INC** BX

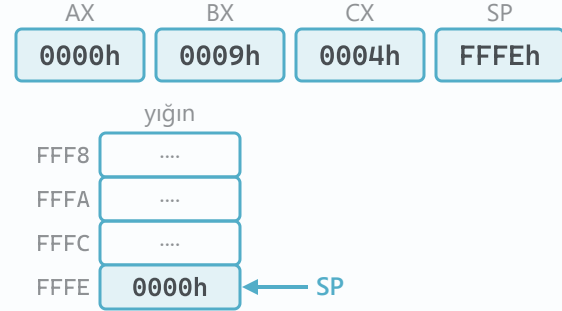
12 **LOOP** dongu

13 **POP** CX ; ters sıra!

14 **POP** BX

15 **RET**

16 TOPLA_N **ENDP**



Çağıranın BX'inde önemli bir değer (9) var; n = CX = 4.

Adım 1 / 10

DEĞİŞKENLER

AX = 0

BX = 9

CX = 4

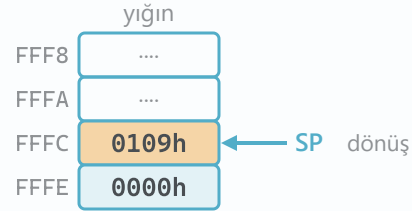
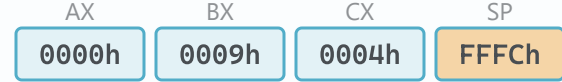
SP = FFFEh

Kod İzleme: TOPLA_N ve Yazmaç Koruma

```

1 MOV BX, 9      ; çağıranın BX'i
2 MOV CX, 4      ; n = 4
▶ CALL TOPLA_N   ; AX = 1+2+...+n
4 RET
5 TOPLA_N PROC
6 PUSH BX        ; BX'i koru
7 PUSH CX        ; CX'i koru
8 XOR AX, AX
9 MOV BX, 1
10 dongu: ADD AX, BX
11 INC BX
12 LOOP dongu
13 POP CX        ; ters sıra!
14 POP BX
15 RET
16 TOPLA_N ENDP

```



CALL TOPLA_N : dönüş adresi 0109h yığına, SP = FFFCh.

Adım 2 / 10

DEĞİŞKENLER

AX = 0

BX = 9

CX = 4

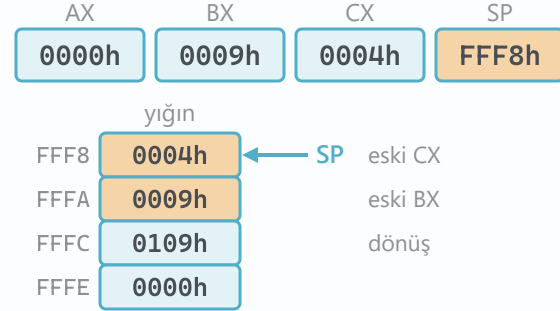
SP = FFFCh

Kod İzleme: TOPLA_N ve Yazmaç Koruma

```

1 MOV BX, 9      ; çağıranın BX'i
2 MOV CX, 4      ; n = 4
3 CALL TOPLA_N   ; AX = 1+2+...+n
4 RET
5 TOPLA_N PROC
▶ PUSH BX      ; BX'i koru
▶ PUSH CX      ; CX'i koru
8 XOR AX, AX
9 MOV BX, 1
10 dongu: ADD AX, BX
11 INC BX
12 LOOP dongu
13 POP CX       ; ters sıra!
14 POP BX
15 RET
16 TOPLA_N ENDP

```



BX (9) ve CX (4) yığına: dönüş adresinin üstünde iki sözcük, SP = FFF8h.

Adım 3 / 10

DEĞİŞKENLER

AX = 0

BX = 9

CX = 4

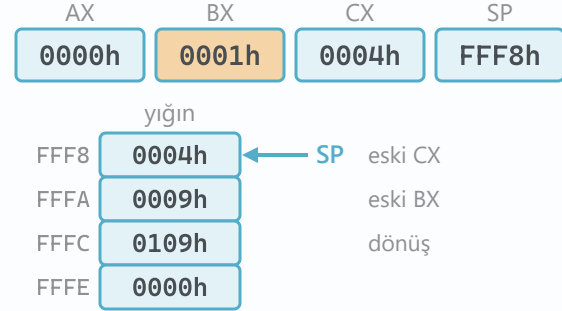
SP = FFF8h

Kod İzleme: TOPLA_N ve Yazmaç Koruma

```

1  MOV  BX, 9      ; çağıranın BX'i
2  MOV  CX, 4      ; n = 4
3  CALL TOPLA_N    ; AX = 1+2+...+n
4  RET
5  TOPLA_N PROC
6  PUSH BX        ; BX'i koru
7  PUSH CX        ; CX'i koru
8  ► XOR  AX, AX
9  ► MOV  BX, 1
10 dongu: ADD  AX, BX
11 INC  BX
12 LOOP dongu
13 POP  CX        ; ters sıra!
14 POP  BX
15 RET
16 TOPLA_N ENDP

```



AX = 0, BX = 1: prosedür BX'i kendi sayacı olarak kullanıyor.

Adım 4 / 10

DEĞİŞKENLER

AX = 0

BX = 1

CX = 4

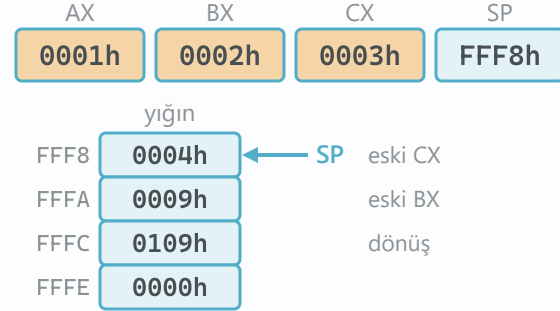
SP = FFF8h

Kod İzleme: TOPLA_N ve Yazmaç Koruma

```

1 MOV BX, 9      ; çağıranın BX'i
2 MOV CX, 4      ; n = 4
3 CALL TOPLA_N   ; AX = 1+2+...+n
4 RET
5 TOPLA_N PROC
6 PUSH BX        ; BX'i koru
7 PUSH CX        ; CX'i koru
8 XOR AX, AX
9 MOV BX, 1
▶ dongu: ADD AX, BX
▶ INC BX
▶ LOOP dongu
13 POP CX        ; ters sıra!
14 POP BX
15 RET
16 TOPLA_N ENDP

```



1. tur: AX = 1, BX = 2, LOOP → CX = 3.

Adım 5 / 10

DEĞİŞKENLER

AX = 1

BX = 2

CX = 3

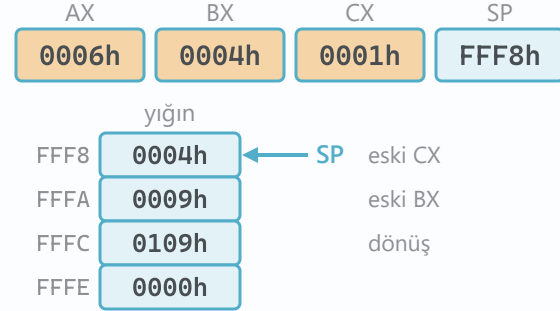
SP = FFF8h

Kod İzleme: TOPLA_N ve Yazmaç Koruma

```

1 MOV BX, 9      ; çağıranın BX'i
2 MOV CX, 4      ; n = 4
3 CALL TOPLA_N   ; AX = 1+2+...+n
4 RET
5 TOPLA_N PROC
6 PUSH BX        ; BX'i koru
7 PUSH CX        ; CX'i koru
8 XOR AX, AX
9 MOV BX, 1
▶ dongu: ADD AX, BX
▶ INC BX
▶ LOOP dongu
13 POP CX        ; ters sıra!
14 POP BX
15 RET
16 TOPLA_N ENDP

```



2. ve 3. tur: $AX = 1 + 2 + 3 = 6$, $BX = 4$, $CX = 1$.

Adım 6 / 10

DEĞİŞKENLER

AX = 6

BX = 4

CX = 1

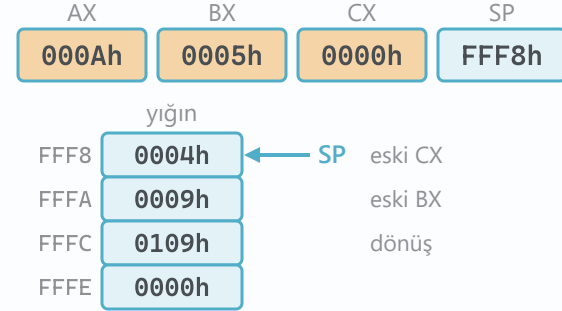
SP = FFF8h

Kod İzleme: TOPLA_N ve Yazmaç Koruma

```

1 MOV BX, 9      ; çağıranın BX'i
2 MOV CX, 4      ; n = 4
3 CALL TOPLA_N   ; AX = 1+2+...+n
4 RET
5 TOPLA_N PROC
6 PUSH BX       ; BX'i koru
7 PUSH CX       ; CX'i koru
8 XOR AX, AX
9 MOV BX, 1
▶ dongu: ADD AX, BX
▶ INC BX
▶ LOOP dongu
13 POP CX        ; ters sıra!
14 POP BX
15 RET
16 TOPLA_N ENDP

```



4. tur: AX = 10, CX = 0 → döngü biter. BX = 5, CX = 0: çağıranın değerleri bozulmuş durumda.

Adım 7 / 10

DEĞİŞKENLER

AX = 10

BX = 5

CX = 0

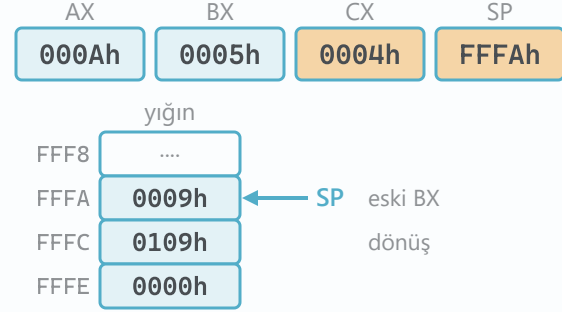
SP = FFF8h

Kod İzleme: TOPLA_N ve Yazmaç Koruma

```

1 MOV BX, 9      ; çağıranın BX'i
2 MOV CX, 4      ; n = 4
3 CALL TOPLA_N   ; AX = 1+2+...+n
4 RET
5 TOPLA_N PROC
6 PUSH BX        ; BX'i koru
7 PUSH CX        ; CX'i koru
8 XOR AX, AX
9 MOV BX, 1
10 dongu: ADD AX, BX
11 INC BX
12 LOOP dongu
13 ► POP CX      ; ters sıra!
14 POP BX
15 RET
16 TOPLA_N ENDP

```



POP CX : son itilen ilk alınır → CX = 4.

Adım 8 / 10

DEĞİŞKENLER

AX = 10

BX = 5

CX = 4

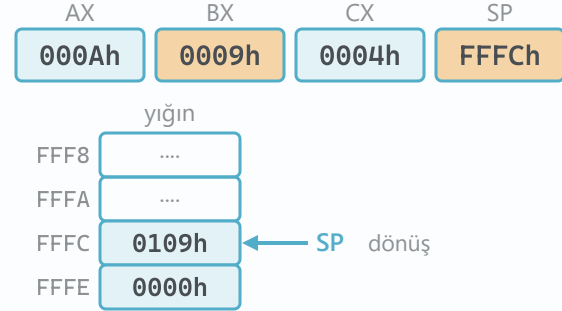
SP = FFFAh

Kod İzleme: TOPLA_N ve Yazmaç Koruma

```

1 MOV  BX, 9      ; çağıranın BX'i
2 MOV  CX, 4      ; n = 4
3 CALL TOPLA_N    ; AX = 1+2+...+n
4 RET
5 TOPLA_N PROC
6 PUSH BX        ; BX'i koru
7 PUSH CX        ; CX'i koru
8 XOR   AX, AX
9 MOV   BX, 1
10 dongu: ADD AX, BX
11 INC  BX
12 LOOP dongu
13 POP  CX        ; ters sıra!
14 ► POP BX
15 RET
16 TOPLA_N ENDP

```



POP BX → BX = 9. SP = FFFCh: yığının üstünde yine dönüş adresi var.

Adım 9 / 10

DEĞİŞKENLER

AX = 10

BX = 9

CX = 4

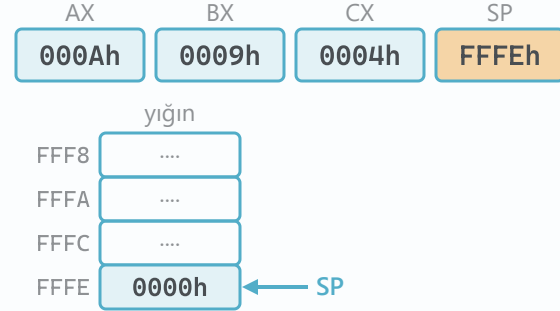
SP = FFFCh

Kod İzleme: TOPLA_N ve Yazmaç Koruma

```

1 MOV  BX, 9      ; çağıranın BX'i
2 MOV  CX, 4      ; n = 4
3 CALL TOPLA_N    ; AX = 1+2+...+n
4 RET
5 TOPLA_N PROC
6 PUSH  BX        ; BX'i koru
7 PUSH  CX        ; CX'i koru
8 XOR   AX, AX
9 MOV   BX, 1
10 dongu: ADD  AX, BX
11 INC  BX
12 LOOP dongu
13 POP  CX        ; ters sıra!
14 POP  BX
15 ► RET
16 TOPLA_N ENDP

```



RET : 0109h'ye dönülür. AX = 10 (sonuç), BX = 9 ve CX = 4 korunmuş.

Adım 10 / 10

DEĞİŞKENLER

AX = 10

BX = 9

CX = 4

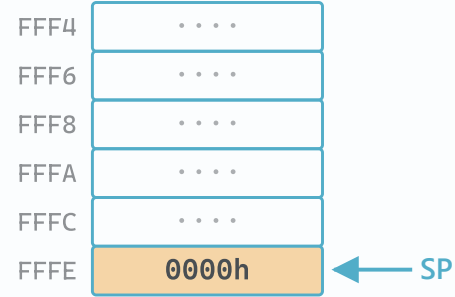
SP = FFFEh

Yığınla Parametre Geçirme (BP ile)

```

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

```



SS:bağlı konum · küçük adres yukarıda

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

Adım 1 / 14

DEĞİŞKENLER

AX = 0000h

BP = 0000h

SP = FFFEh

IP = 0100h

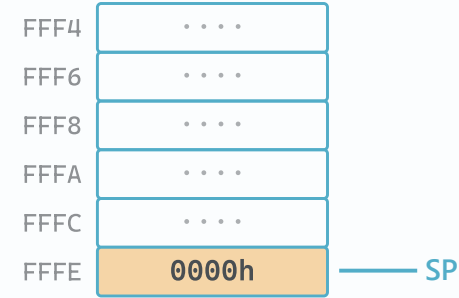
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

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

```



SS:bağlı konum · küçük adres yukarıda

AX = 7.

Adım 2 / 14

DEĞİŞKENLER

AX = 0007h

BP = 0000h

SP = FFEh

IP = 0103h

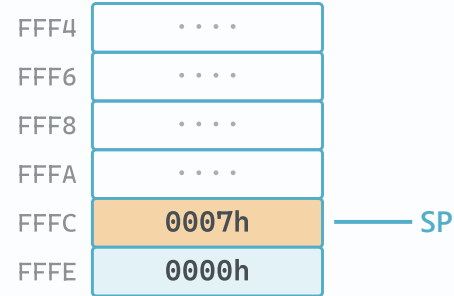
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

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

```



SS:bağlı konum · küçük adres yukarıda

1. parametre yığında: [FFFC] = 0007h.

Adım 3 / 14

DEĞİŞKENLER

AX = 0007h

BP = 0000h

SP = FFFCh

IP = 0104h

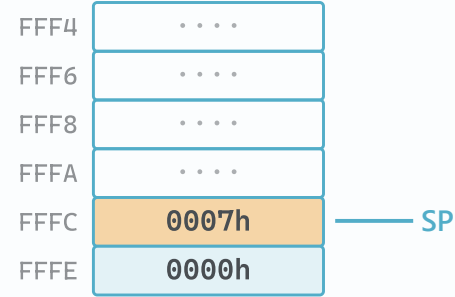
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

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

```



SS:bağlı konum · küçük adres yukarıda

AX = 3.

Adım 4 / 14

DEĞİŞKENLER

AX = 0003h

BP = 0000h

SP = FFFCh

IP = 0107h

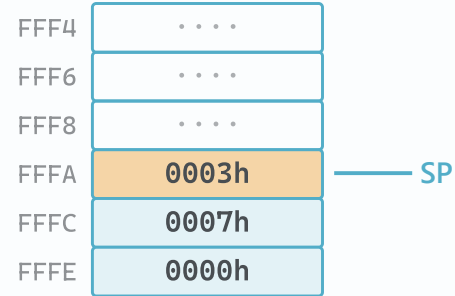
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

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

```



SS:bağlı konum · küçük adres yukarıda

2. parametre: [FFFAh] = 0003h.

Adım 5 / 14

DEĞİŞKENLER

AX = 0003h

BP = 0000h

SP = FFFAh

IP = 0108h

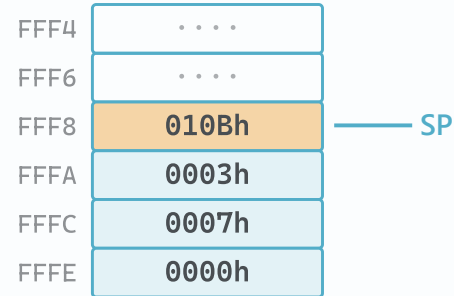
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

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

```



SS:bağlı konum · küçük adres yukarıda

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

Adım 6 / 14

DEĞİŞKENLER

AX = 0003h

BP = 0000h

SP = FFF8h

IP = 010Fh

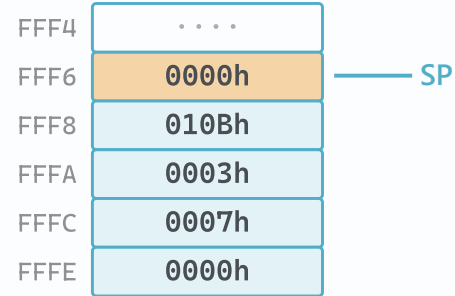
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

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

```



SS:bağlı konum · küçük adres yukarıda

Eski BP saklanır: [FFF6h] = 0000h.

Adım 7 / 14

DEĞİŞKENLER

AX = 0003h

BP = 0000h

SP = FFF6h

IP = 0110h

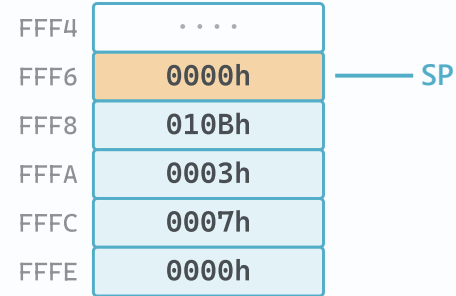
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

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

```



SS:bağlı konum · küçük adres yukarıda

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

Adım 8 / 14

DEĞİŞKENLER

AX = 0003h

BP = **FFF6h**

SP = **FFF6h**

IP = 0112h

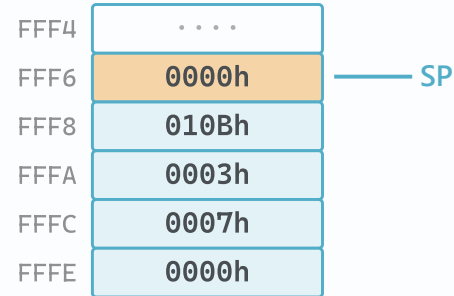
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

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

```



SS:bağlı konum · küçük adres yukarıda

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

Adım 9 / 14

DEĞİŞKENLER

AX = 0007h

BP = FFF6h

SP = FFF6h

IP = 0115h

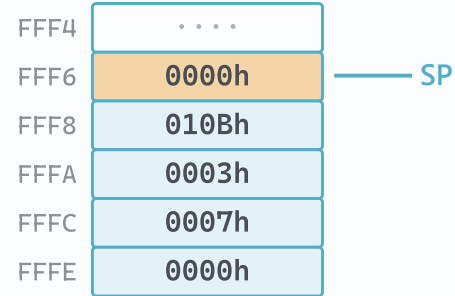
$[BP+0]$ = eski BP, $[BP+2]$ = dönüş adresi, $[BP+4]$ = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

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

```



SS:bağlı konum · küçük adres yukarıda

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

Adım 10 / 14

DEĞİŞKENLER

AX = 000Ah

BP = FFF6h

SP = FFF6h

IP = 0118h

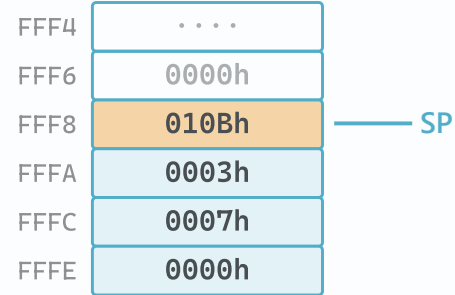
$[BP+0]$ = eski BP, $[BP+2]$ = dönüş adresi, $[BP+4]$ = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

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

```



SS:bağlı konum · küçük adres yukarıda

BP eski değerine döner; SP = FFF8h (dönüş adresi üstte).

Adım 11 / 14

DEĞİŞKENLER

AX = 000Ah

BP = 0000h

SP = FFF8h

IP = 0119h

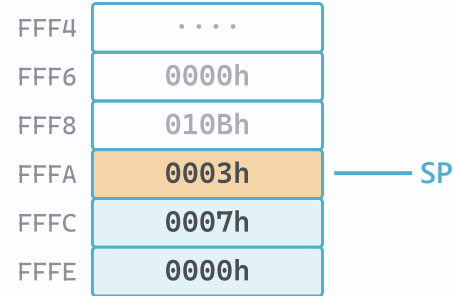
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

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

```



SS:bağlı konum · küçük adres yukarıda

RET: IP = 010Bh, SP = FFFAh.

Adım 12 / 14

DEĞİŞKENLER

AX = 000Ah

BP = 0000h

SP = FFFAh

IP = 010Bh

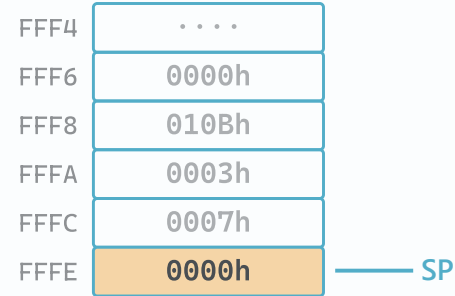
[BP+0] = eski BP, [BP+2] = dönüş adresi, [BP+4] = son konan parametre. ADD SP, 4 yerine prosedürde RET 4 de yazılabilirdi.

Yığınla Parametre Geçirme (BP ile)

```

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

```



SS:bağlı konum · küçük adres yukarıda

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

Adım 13 / 14

DEĞİŞKENLER

AX = 000Ah

BP = 0000h

SP = FFEh

IP = 010Eh

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

Yığınla Parametre Geçirme (BP ile)

```

1  ORG  100h
2  MOV  AX, 7
3  PUSH AX           ; 1. parametre
4  MOV  AX, 3
5  PUSH AX           ; 2. parametre
6  CALL TOPLA
7  ADD  SP, 4         ; parametreleri at
▶  RET

```

```

9  TOPLA  PROC
10  PUSH BP
11  MOV  BP, SP
12  MOV  AX, [BP+6] ; 1. parametre
13  ADD  AX, [BP+4] ; 2. parametre
14  POP  BP
15  RET
16  TOPLA  ENDP

```

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

SS:bağlı konum · küçük adres yukarıda

Program biter.

Adım 14 / 14

DEĞİŞKENLER

AX = 000Ah

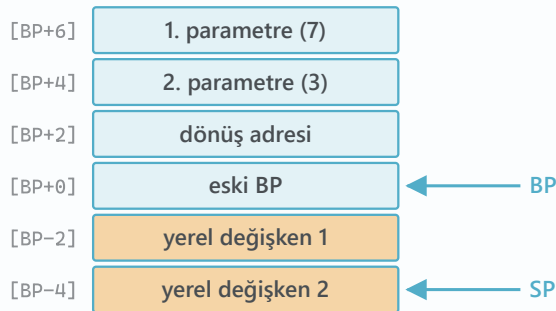
BP = 0000h

SP = 0000h

IP = 0000h

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

Yığın Çerçevesi ve Yerel Değişkenler



küçük adres yukarıda · büyük adres aşağıda

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

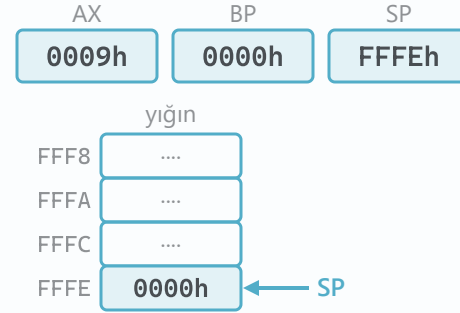
- `SUB SP, 4` yığında iki sözcüklük **yerel alan** açar: [BP-2], [BP-4].
- `MOV SP, BP` yerel alanı tek komutla geri verir.
- Parametreler pozitif, yereller negatif BP bağlı konumundadır.

Kod İzleme: Yığında Yerel Değişken

```

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

```



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

Adım 1 / 10

DEĞİŞKENLER

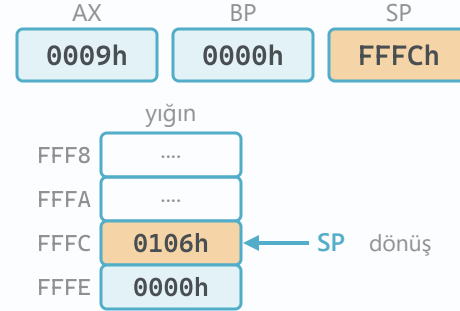
AX = 9

BP = 0000h

SP = FFFEh

Kod İzleme: Yığında Yerel Değişken

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



CALL : dönüş adresi 0106h → [FFFC].

Adım 2 / 10

DEĞİŞKENLER

AX = 9

BP = 0000h

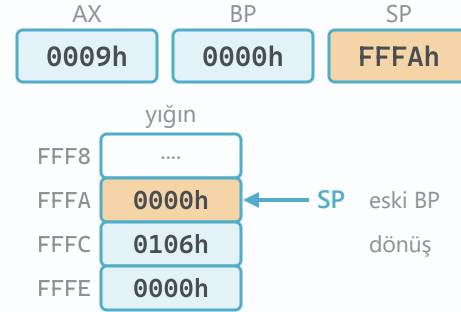
SP = FFFCh

Kod İzleme: Yığında Yerel Değişken

```

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

```



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

Adım 3 / 10

DEĞİŞKENLER

AX = 9

BP = 0000h

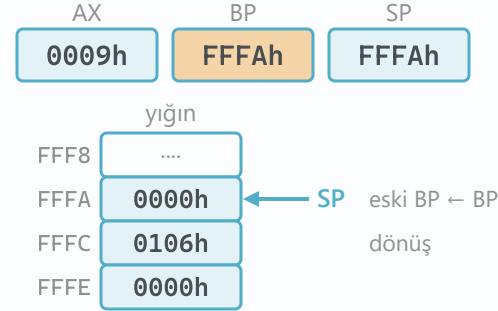
SP = FFFAh

Kod İzleme: Yığında Yerel Değişken

```

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

```



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

Adım 4 / 10

DEĞİŞKENLER

AX = 9

BP = FFFAh

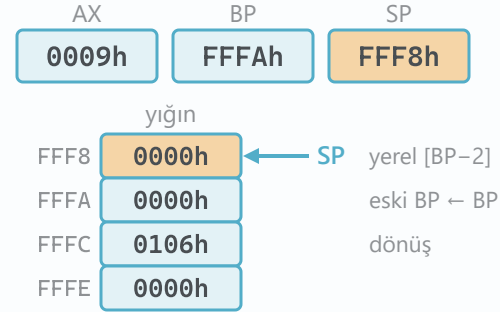
SP = FFFAh

Kod İzleme: Yığında Yerel Değişken

```

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

```



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

Adım 5 / 10

DEĞİŞKENLER

AX = 9

BP = FFFAh

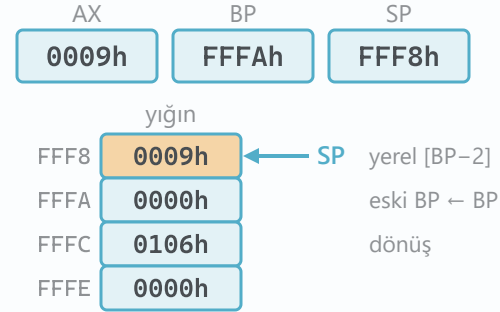
SP = FFF8h

Kod İzleme: Yığında Yerel Değişken

```

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

```



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

Adım 6 / 10

DEĞİŞKENLER

AX = 9

BP = FFFAh

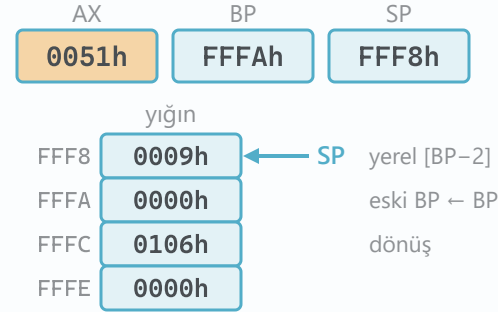
SP = FFF8h

Kod İzleme: Yığında Yerel Değişken

```

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

```



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

Adım 7 / 10

DEĞİŞKENLER

AX = 81

BP = FFFAh

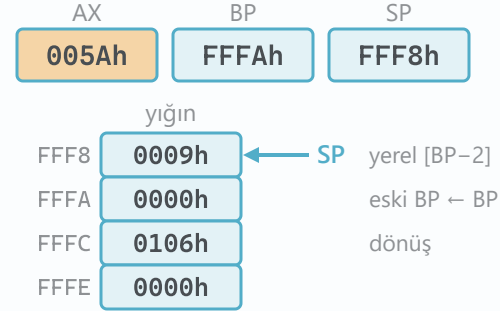
SP = FFF8h

Kod İzleme: Yığında Yerel Değişken

```

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

```



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

Adım 8 / 10

DEĞİŞKENLER

AX = 90

BP = FFFAh

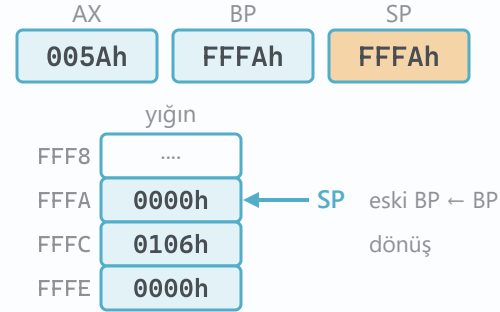
SP = FFF8h

Kod İzleme: Yığında Yerel Değişken

```

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

```



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

Adım 9 / 10

DEĞİŞKENLER

AX = 90

BP = FFFAh

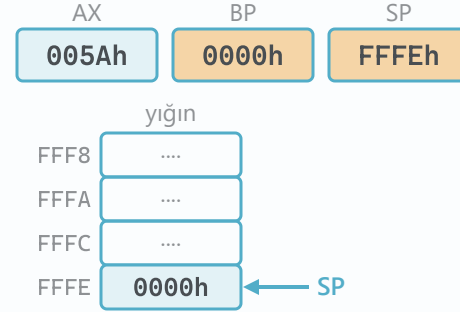
SP = FFFAh

Kod İzleme: Yığında Yerel Değişken

```

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

```



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

Adım 10 / 10

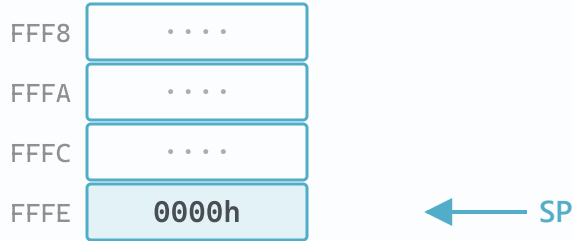
DEĞİŞKENLER

AX = 90

BP = 0000h

SP = FFFEh

INT ve IRET: Yığında Üç Sözcük

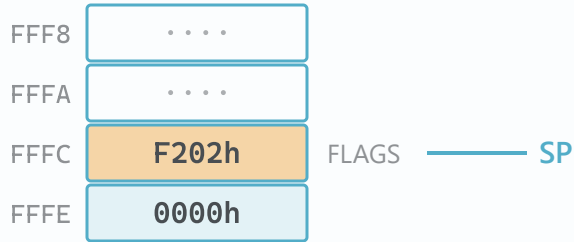


0700:0103h adresindeki **INT 21h** (2 bayt) çalışacak. FLAGS = F202h (IF = 1; 8086'da 12–15. bitler hep 1), SP = FFFEh.

Adım 1 / 6

Donanım kesmelerinde (ör. zamanlayıcı, klavye) CPU aynı üç sözcüğü kendisi yığına koyar. Bu yüzden kesme yordamları kullandıkları yazmaçları mutlaka PUSH/POP ile korumalıdır.

INT ve IRET: Yığında Üç Sözcük

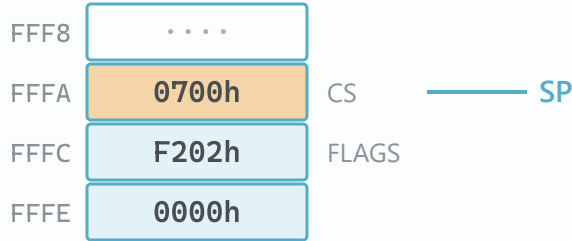


① FLAGS yığına: $SP = FFFCh$, $[FFCh] = \mathbf{F202h}$.

Adım 2 / 6

Donanım kesmelerinde (ör. zamanlayıcı, klavye) CPU aynı üç sözcüğü kendisi yığına koyar. Bu yüzden kesme yordamları kullandıkları yazmaçları mutlaka PUSH/POP ile korumalıdır.

INT ve IRET: Yığında Üç Sözcük

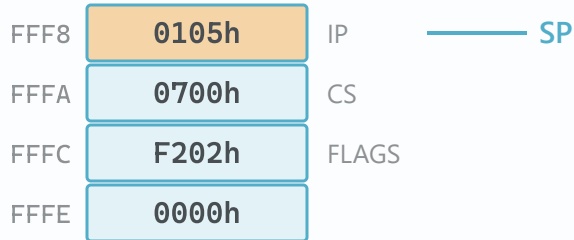


② CS yığına: SP = FFFAh, [FFFAh] = **0700h**.

Adım 3 / 6

Donanım kesmelerinde (ör. zamanlayıcı, klavye) CPU aynı üç sözcüğü kendisi yığına koyar. Bu yüzden kesme yordamları kullandıkları yazmaçları mutlaka PUSH/POP ile korumalıdır.

INT ve IRET: Yığında Üç Sözcük



③ Dönüş IP'si yığına: $SP = FFF8h$, $[FFF8h] = \mathbf{0105h}$ (INT'ten sonraki komut). Ardından $IF = TF = 0$.

Adım 4 / 6

Donanım kesmelerinde (ör. zamanlayıcı, klavye) CPU aynı üç sözcüğü kendisi yığına koyar. Bu yüzden kesme yordamları kullandıkları yazmaçları mutlaka PUSH/POP ile korumalıdır.

INT ve IRET: Yığında Üç Sözcük



- ④ Yeni CS:IP, vektör tablosundaki $21h \times 4 = 84h$ adresinden okunur: $IP \leftarrow [0000:0084h]$, $CS \leftarrow [0000:0086h]$. Kesme yordamı çalışır.

Adım 5 / 6

Donanım kesmelerinde (ör. zamanlayıcı, klavye) CPU aynı üç sözcüğü kendisi yığına koyar. Bu yüzden kesme yordamları kullandıkları yazmaçları mutlaka PUSH/POP ile korumalıdır.

INT ve IRET: Yığında Üç Sözcük

FFF8	0105h	IP
FFFA	0700h	CS
FFFC	F202h	FLAGS
FFFE	0000h	

—— SP

IRET sonrası:

IP = 0105h, CS = 0700h

FLAGS = F202h

- ⑤ Yordamın sonundaki **IRET** : IP, CS ve FLAGS ters sırada geri alınır; SP = FFFEH. Program 0700:0105h'den devam eder.

Adım 6 / 6

Donanım kesmelerinde (ör. zamanlayıcı, klavye) CPU aynı üç sözcüğü kendisi yığına koyar. Bu yüzden kesme yordamları kullandıkları yazmaçları mutlaka PUSH/POP ile korumalıdır.

Kod İzleme: INT 61h ve IRET

```

▶ XOR AX, AX
▶ MOV ES, AX      ; ES = 0000h
3 MOV WORD PTR ES:[184h], OFFSET isr
4 MOV ES:[186h], CS ; vektör 61h
5 CLC              ; CF = 0
6 INT 61h
7 RET
8 isr: STC          ; yordamda CF = 1
9 MOV BX, 1234h
10 IRET            ; FLAGS geri yüklenir

```



Adım 1 / 8

DEĞİŞKENLER

IP = 0104h

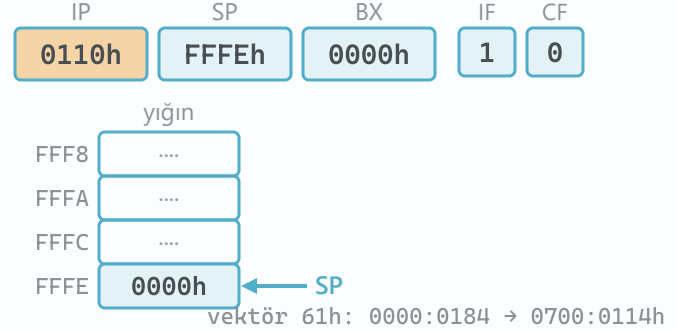
SP = FFEh

IF = 1

CF = 0

Kod İzleme: INT 61h ve IRET

```
1 XOR AX, AX
2 MOV ES, AX      ; ES = 0000h
▶ MOV WORD PTR ES:[184h], OFFSET isr
▶ MOV ES:[186h], CS ; vektör 61h
5 CLC              ; CF = 0
6 INT 61h
7 RET
8 isr: STC          ; yordamda CF = 1
9 MOV BX, 1234h
10 IRET            ; FLAGS geri yüklenir
```



Vektör 61h kuruldu: [0000:0184h] = isr'nin bağlı konumu (0114h), [0000:0186h] = CS (0700h).

Adım 2 / 8

DEĞİŞKENLER

IP = 0110h

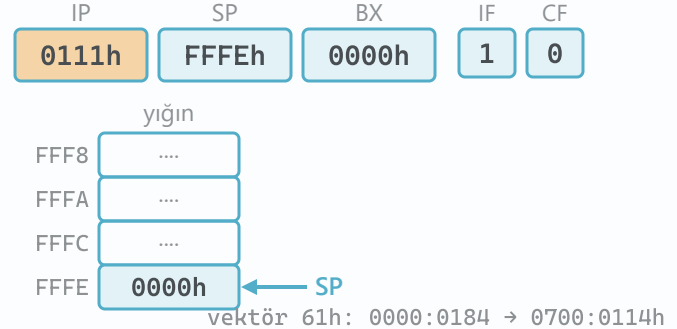
SP = FFFEh

IF = 1

CF = 0

Kod İzleme: INT 61h ve IRET

```
1 XOR AX, AX
2 MOV ES, AX      ; ES = 0000h
3 MOV WORD PTR ES:[184h], OFFSET isr
4 MOV ES:[186h], CS ; vektör 61h
5 ► CLC           ; CF = 0
6 INT 61h
7 RET
8 isr: STC        ; yordamda CF = 1
9 MOV BX, 1234h
10 IRET           ; FLAGS geri yüklenir
```



CLC : CF = 0. FLAGS = F246h (IF = 1; XOR'dan ZF = PF = 1; 8086'da üst 4 bit hep 1).

Adım 3 / 8

DEĞİŞKENLER

IP = 0111h

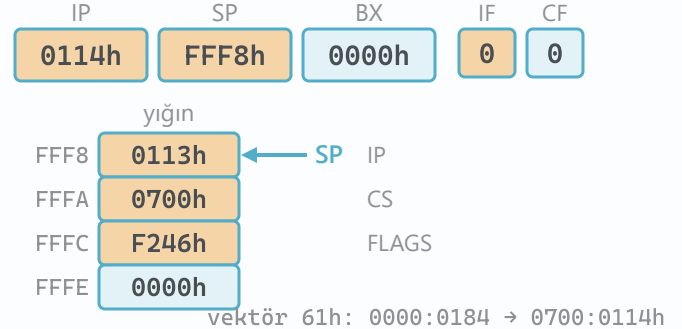
SP = FFEh

IF = 1

CF = 0

Kod İzleme: INT 61h ve IRET

```
1 XOR AX, AX
2 MOV ES, AX      ; ES = 0000h
3 MOV WORD PTR ES:[184h], OFFSET isr
4 MOV ES:[186h], CS ; vektör 61h
5 CLC              ; CF = 0
▶ INT 61h
7 RET
8 isr: STC          ; yordamda CF = 1
9 MOV BX, 1234h
10 IRET            ; FLAGS geri yüklenir
```



INT 61h : FLAGS (F246h), CS (0700h) ve dönüş IP'si (0113h) yığına; SP = FFF8h. IF = 0, CS:IP ← 0700:0114h.

Adım 4 / 8

DEĞİŞKENLER

IP = 0114h

SP = FFF8h

IF = 0

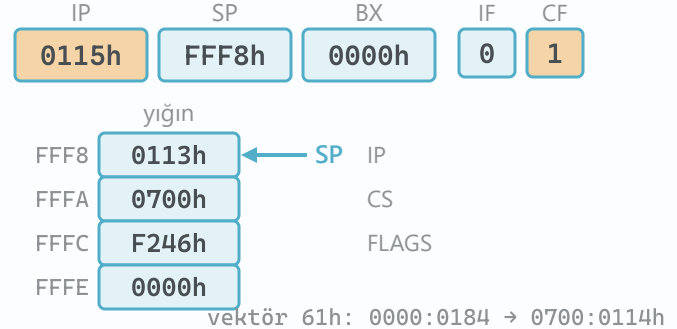
CF = 0

Kod İzleme: INT 61h ve IRET

```

1 XOR  AX, AX
2 MOV  ES, AX      ; ES = 0000h
3 MOV  WORD PTR ES:[184h], OFFSET isr
4 MOV  ES:[186h], CS ; vektör 61h
5 CLC                      ; CF = 0
6 INT  61h
7 RET
▶ isr: STC                ; yordamda CF = 1
9 MOV  BX, 1234h
10 IRET                ; FLAGS geri yüklenir

```



Yordam **STC** ile CF = 1 yapar. Yordam içinde IF = 0: başka INTR kesmesi kabul edilmez.

Adım 5 / 8

DEĞİŞKENLER

IP = 0115h

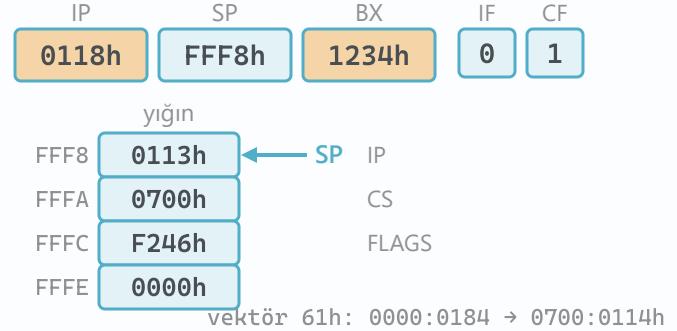
SP = FFF8h

IF = 0

CF = 1

Kod İzleme: INT 61h ve IRET

```
1 XOR AX, AX
2 MOV ES, AX      ; ES = 0000h
3 MOV WORD PTR ES:[184h], OFFSET isr
4 MOV ES:[186h], CS ; vektör 61h
5 CLC              ; CF = 0
6 INT 61h
7 RET
8 isr: STC         ; yordamda CF = 1
9 ► MOV BX, 1234h
10 IRET           ; FLAGS geri yüklenir
```



BX = 1234h: yazmaçlara yapılan değişiklik IRET'ten sonra da kalır.

Adım 6 / 8

DEĞİŞKENLER

IP = 0118h

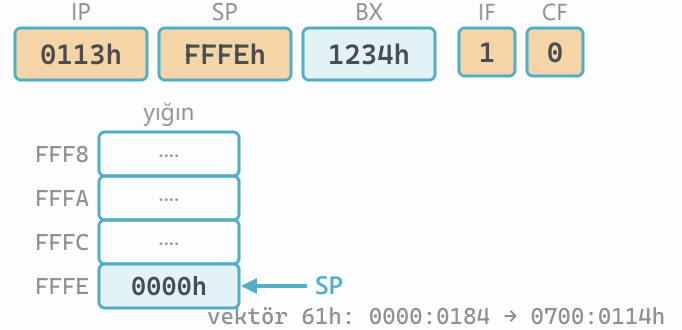
SP = FFF8h

IF = 0

CF = 1

Kod İzleme: INT 61h ve IRET

```
1 XOR AX, AX
2 MOV ES, AX      ; ES = 0000h
3 MOV WORD PTR ES:[184h], OFFSET isr
4 MOV ES:[186h], CS ; vektör 61h
5 CLC             ; CF = 0
6 INT 61h
7 RET
8 isr: STC        ; yordamda CF = 1
9 MOV BX, 1234h
▶ IRET           ; FLAGS geri yüklenir
```



IRET : IP, CS, FLAGS ters sırayla alınır. CF yeniden 0, IF yeniden 1: yordamın bayrak değişikliği kayboldu.

Adım 7 / 8

DEĞİŞKENLER

IP = 0113h

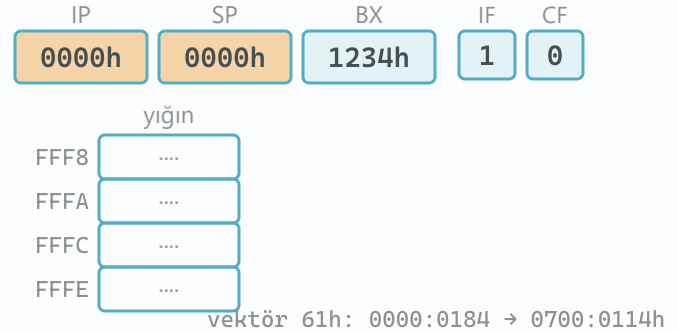
SP = FFFEh

IF = 1

CF = 0

Kod İzleme: INT 61h ve IRET

```
1 XOR AX, AX
2 MOV ES, AX      ; ES = 0000h
3 MOV WORD PTR ES:[184h], OFFSET isr
4 MOV ES:[186h], CS ; vektör 61h
5 CLC              ; CF = 0
6 INT 61h
▶ RET
8 isr: STC          ; yordamda CF = 1
9 MOV BX, 1234h
10 IRET             ; FLAGS geri yüklenir
```



RET ile program biter. Bayrakla sonuç döndürmek isteyen yordam, yığındaki FLAGS kopyasını değiştirmelidir.

Adım 8 / 8

DEĞİŞKENLER

IP = 0000h

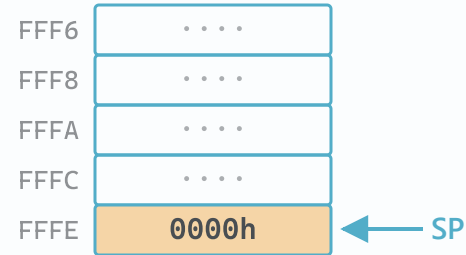
SP = 0000h

IF = 1

CF = 0

Bayrakları Saklamak: PUSHF / POPF

```
► ORG    100h
2 MOV    AX, 0FFFFh
3 ADD    AX, 1      ; CF=1, ZF=1
4 PUSHF                ; bayrakları sakla
5 MOV    BX, 5
6 ADD    BX, 3      ; CF=0, ZF=0
7 POPF                ; bayraklar geri
8 RET
```



SS:bağlı konum · küçük adres yukarıda

Başlangıç: bayraklar temiz (IF = 1).

Adım 1 / 8

DEĞİŞKENLER

AX = 0000h

BX = 0000h

SP = FFFEh

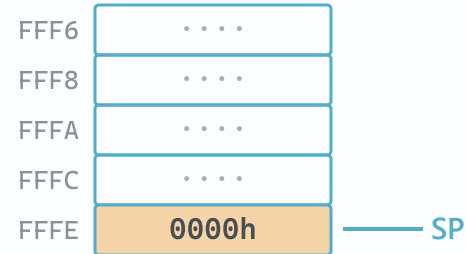
CF = 0

ZF = 0

Kesme yordamları bayrakları kendileri saklamaz; INT bunu otomatik yapar. Normal kodda bir sonucu korumak için PUSHF/POPF gerekir.

Bayrakları Saklamak: PUSHF / POPF

```
1  ORG    100h
▶ 2  MOV   AX, 0FFFFh
3  ADD    AX, 1      ; CF=1, ZF=1
4  PUSHF                ; bayrakları sakla
5  MOV    BX, 5
6  ADD    BX, 3      ; CF=0, ZF=0
7  POPF                ; bayraklar geri
8  RET
```



SS:bağlı konum · küçük adres yukarıda

AX = FFFFh.

Adım 2 / 8

DEĞİŞKENLER

AX = FFFFh

BX = 0000h

SP = FFFEh

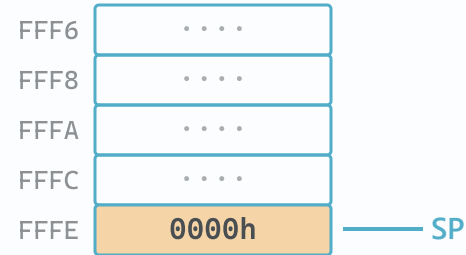
CF = 0

ZF = 0

Kesme yordamları bayrakları kendileri saklamaz; INT bunu otomatik yapar. Normal kodda bir sonucu korumak için PUSHF/POPF gerekir.

Bayrakları Saklamak: PUSHF / POPF

```
1 ORG 100h
2 MOV AX, 0FFFFh
3 ► ADD AX, 1 ; CF=1, ZF=1
4 PUSHF ; bayrakları sakla
5 MOV BX, 5
6 ADD BX, 3 ; CF=0, ZF=0
7 POPF ; bayraklar geri
8 RET
```



SS:bağlı konum · küçük adres yukarıda

$FFFFh + 1 = 1\ 0000h \rightarrow AX = 0000h, CF = 1, ZF = 1.$

Adım 3 / 8

DEĞİŞKENLER

AX = 0000h

BX = 0000h

SP = FFFEh

CF = 1

ZF = 1

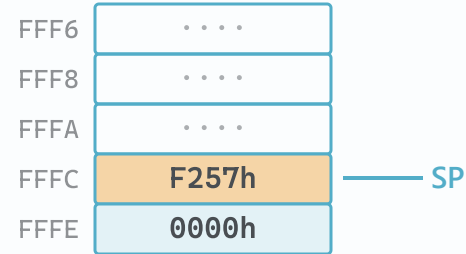
Kesme yordamları bayrakları kendileri saklamaz; INT bunu otomatik yapar. Normal kodda bir sonucu korumak için PUSHF/POPF gerekir.

Bayrakları Saklamak: PUSHF / POPF

```

1  ORG    100h
2  MOV    AX, 0FFFFh
3  ADD    AX, 1      ; CF=1, ZF=1
4  ► PUSHF          ; bayrakları sakla
5  MOV    BX, 5
6  ADD    BX, 3      ; CF=0, ZF=0
7  POPF
8  RET

```



SS:bağlı konum · küçük adres yukarıda

PUSHF: FLAGS sözcüğü (**F257h**: CF, PF, AF, ZF, IF ve 8086'nın hep 1 olan bitleri) yığına.

Adım 4 / 8

DEĞİŞKENLER

AX = 0000h

BX = 0000h

SP = FFFCh

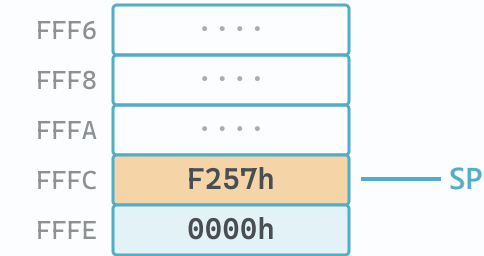
CF = 1

ZF = 1

Kesme yordamları bayrakları kendileri saklamaz; INT bunu otomatik yapar. Normal kodda bir sonucu korumak için PUSHF/POPF gerekir.

Bayrakları Saklamak: PUSHF / POPF

```
1 ORG 100h
2 MOV AX, 0FFFFh
3 ADD AX, 1 ; CF=1, ZF=1
4 PUSHF ; bayrakları sakla
5 MOV BX, 5
6 ADD BX, 3 ; CF=0, ZF=0
7 POPF ; bayraklar geri
8 RET
```



BX = 5.

Adım 5 / 8

DEĞİŞKENLER

AX = 0000h

BX = 0005h

SP = FFFCh

CF = 1

ZF = 1

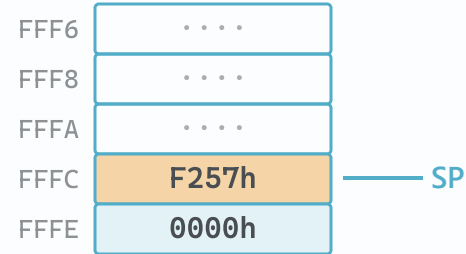
Kesme yordamları bayrakları kendileri saklamaz; INT bunu otomatik yapar. Normal kodda bir sonucu korumak için PUSHF/POPF gerekir.

Bayrakları Saklamak: PUSHF / POPF

```

1  ORG  100h
2  MOV  AX, 0FFFFh
3  ADD  AX, 1      ; CF=1, ZF=1
4  PUSHF          ; bayrakları sakla
5  MOV  BX, 5
6  ► ADD  BX, 3    ; CF=0, ZF=0
7  POPF          ; bayraklar geri
8  RET

```



SS:bağlı konum · küçük adres yukarıda

$5 + 3 = 8 \rightarrow CF = 0, ZF = 0$. Önceki sonuç bayrakları kayboldu...

Adım 6 / 8

DEĞİŞKENLER

AX = 0000h

BX = 0003h

SP = FFFCh

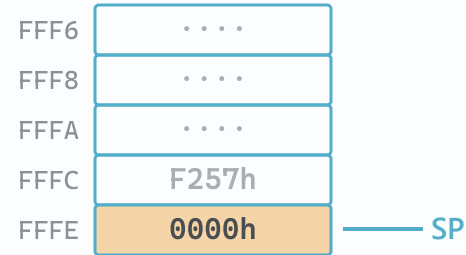
CF = 0

ZF = 0

Kesme yordamları bayrakları kendileri saklamaz; INT bunu otomatik yapar. Normal kodda bir sonucu korumak için PUSHF/POPF gerekir.

Bayrakları Saklamak: PUSHF / POPF

```
1 ORG 100h
2 MOV AX, 0FFFFh
3 ADD AX, 1 ; CF=1, ZF=1
4 PUSHF ; bayrakları sakla
5 MOV BX, 5
6 ADD BX, 3 ; CF=0, ZF=0
7 ► POPF ; bayraklar geri
8 RET
```



SS:bağlı konum · küçük adres yukarıda

...POPF ile geri geldi: **CF = 1, ZF = 1**.

Adım 7 / 8

DEĞİŞKENLER

AX = 0000h

BX = 0003h

SP = FFEh

CF = 1

ZF = 1

Kesme yordamları bayrakları kendileri saklamaz; INT bunu otomatik yapar. Normal kodda bir sonucu korumak için PUSHF/POPF gerekir.

Bayrakları Saklamak: PUSHF / POPF

```
1 ORG    100h
2 MOV    AX, 0FFFFh
3 ADD    AX, 1      ; CF=1, ZF=1
4 PUSHF                ; bayrakları sakla
5 MOV    BX, 5
6 ADD    BX, 3      ; CF=0, ZF=0
7 POPF                ; bayraklar geri
▶ RET
```

FFF6	
FFF8	
FFFA	
FFFC	F257h
FFFE	0000h

SS:bağlı konum · küçük adres yukarıda

Program biter.

Adım 8 / 8

DEĞİŞKENLER

AX = 0000h

BX = 0003h

SP = 0000h

CF = 1

ZF = 1

Kesme yordamları bayrakları kendileri saklamaz; INT bunu otomatik yapar. Normal kodda bir sonucu korumak için PUSHF/POPF gerekir.

Yığın Güvenliği

- PUSH ve POP komutları programın çalışması süresince geçici verileri saklar.
- Yığının bütünlüğü **eşit sayıda PUSH ve POP** ile korunmalıdır (her yolda, her dalda!).
- Eşit sayıda PUSH ve POP yapılmazsa yığın bozulur; işletim sistemine geri dönüş yapılamayabilir.
- **RET** yığının üstünde bir **dönüş adresi** bekler. .COM programında DOS, yığına başlangıçta **0000h** koyar: RET ile CS:0000h'ye (PSP) gidilir; orada duran **INT 20h** programı sonlandırır.

Fazla PUSH

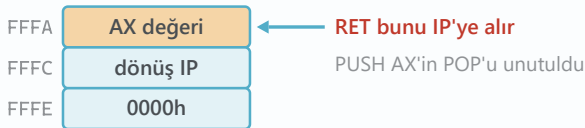
RET, dönüş adresi yerine son konan veriyi alır → rastgele adrese atlama.

Fazla POP

Dönüş adresi bir yazmaca "yenir"; RET yanlış yere döner.

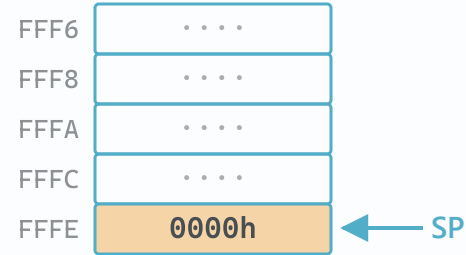
Yanlış sıra

Değerler karışır (takas gibi), program sessizce yanlış çalışır.



Hata Örneği: Dengesiz PUSH

```
► ORG 100h
2 MOV AX, 1234h
3 PUSH AX ; POP unutuldu!
4 RET ; IP ← ?
```



SS:bağlı konum · küçük adres yukarıda

Program sonunda RET ile DOS'a dönmek istiyoruz.

Adım 1 / 4

DEĞİŞKENLER

AX = 0000h

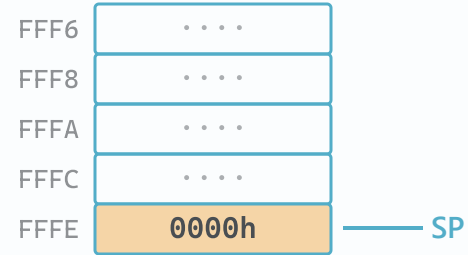
SP = FFFEh

IP = 0100h

Düzeltilme: RET'ten önce POP AX ekleyin ya da hiç PUSH etmeyin. Her dalın (JMP/Jcc) PUSH/POP sayısını ayrı ayrı sayın.

Hata Örneği: Dengesiz PUSH

```
1  ORG  100h
▶  MOV  AX, 1234h
3  PUSH AX      ; POP unutuldu!
4  RET          ; IP ← ?
```



SS:bağlı konum · küçük adres yukarıda

AX = 1234h.

Adım 2 / 4

DEĞİŞKENLER

AX = 1234h

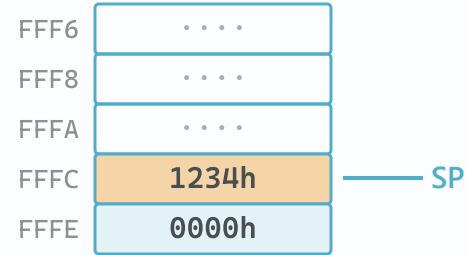
SP = FFFEh

IP = 0103h

Düzeltilme: RET'ten önce POP AX ekleyin ya da hiç PUSH etmeyin. Her dalın (JMP/Jcc) PUSH/POP sayısını ayrı ayrı sayın.

Hata Örneği: Dengesiz PUSH

```
1 ORG 100h
2 MOV AX, 1234h
3 ► PUSH AX ; POP unutuldu!
4 RET ; IP ← ?
```



SS:bağlı konum · küçük adres yukarıda
AX yığına kondu ama hiç POP edilmeyecek.

Adım 3 / 4

DEĞİŞKENLER

AX = 1234h

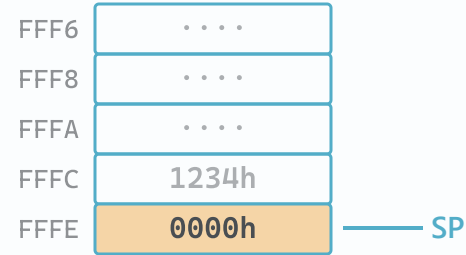
SP = FFFCh

IP = 0104h

Düzeltilme: RET'ten önce POP AX ekleyin ya da hiç PUSH etmeyin. Her dalın (JMP/Jcc) PUSH/POP sayısını ayrı ayrı sayın.

Hata Örneği: Dengesiz PUSH

```
1 ORG 100h
2 MOV AX, 1234h
3 PUSH AX ; POP unutuldu!
▶ RET ; IP ← ?
```



SS:bağlı konum · küçük adres yukarıda

RET üstteki sözcüğü dönüş adresi sanar: IP = **1234h**.
CPU 0700:1234h'deki rastgele baytları komut olarak çalıştırmaya başlar → program çöker ya da donar.

Adım 4 / 4

DEĞİŞKENLER

AX = **1234h**

SP = **FFFEh**

IP = **1234h**

Düzeltilme: RET'ten önce POP AX ekleyin ya da hiç PUSH etmeyin. Her dalın (JMP/Jcc) PUSH/POP sayısını ayrı ayrı sayın.

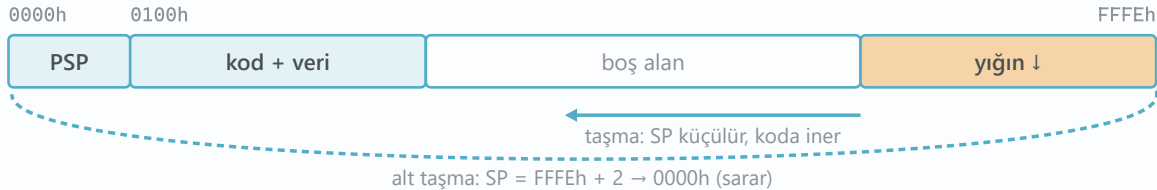
Yığın Taşması ve Alt Taşma

Taşma (overflow)

- Sonsuz özyineleme, döngü içinde POP'suz PUSH.
- SP küçüldükçe yığın aşağı inip **kod ve verinin üzerine** yazar (.COM'da aynı kesim).
- 8086 bunu denetlemez; hata yoktur, yalnızca bozulma vardır.

Alt taşma (underflow)

- PUSH edilmemiş değeri POP etmek.
- SP = FFEh iken POP → SP = **0000h** (16 bitte sarar); sonraki PUSH FFEh'ye yazar.
- Okunan değer çöptür (ya da DOS'un 0000h'si).



Kural: yığın alanının boyutunu programın en derin çağrı zinciri + en çok PUSH sayısına göre seçin. .EXE programlarında .STACK 100h gibi bir direktifle ayrı ve yeterli bir yığın kesimi ayırın.

İyi Uygulamalar

1. Ayna sırası

PUSH AX, BX, CX → POP CX, BX, AX. Kodu alt alta yazınca ayna görüntüsü gibi görünmeli.

2. Tek çıkış

Prosedürde birden çok RET yerine tek bir çıkış etiketi kullanın; POP'lar bir kez yazılır.

3. SP'ye dokunmayın

SP'yi doğrudan değiştirmek yalnızca parametre temizleme (ADD SP, n) veya yerel alan ayırma (SUB SP, n) içindir.

4. BP çerçevesi

Yığındaki verilere [SP+n] ile erişilemez (8086'da SP taban yazmaç değildir); MOV BP, SP kullanın.

5. Kesimleri koruyun

DS/ES değiştiren yordamlar onları da PUSH/POP etmeli.

6. emu8086 ile denetleyin

stack düğmesi yığın penceresini açar; RET'ten önce SP'nin girişteki değerine döndüğünü kontrol edin.



emu8086 ile Yığını İzlemek

1. Derle ve yükle

compile → *emulate*. Yazmaç panelinde CS, IP, SS, SP'nin başlangıç değerlerini not edin (.COM: 0700h, 0100h, 0700h, FFFEh).

2. Tek adım

single step her basıpta bir komut çalıştırır; değişen yazmaçlar vurgulanır. *step back* ile geri alınabilir.

3. stack penceresi

Alttaki *stack* düğmesi SS:SP çevresindeki sözcükleri gösterir; SP'nin gösterdiği satır işaretlidir.

4. Bellek görünümü

Ortakdaki liste baytları (hex, onluk, ASCII) gösterir: B8 12 12 = MOV AX, 1212h . Little-endian burada görülür.

Yükleme sonrası CX, programın bayt sayısını gösterir (ör. takas örneğinde **000Bh** = 11 bayt). Bu bir DOS geleneğidir; programınız CX'e güvenmemelidir.

ÖZET

Özet

Kavram	Hatırlanacak nokta
Adres	Yığının üstü = SS:SP; fiziksel adres $SS \times 16 + SP$.
Büyüme yönü	Aşağı: PUSH SP -= 2, POP SP += 2. Birim her zaman sözcük.
PUSH / POP	REG, SREG, bellek (POP CS yok). PUSH sabit yalnızca 80186+ / emu8086.
LIFO	Değerler ters sırada döner → saklarken ayna sırası, takasta aynı sıra.
CALL / RET	Dönüş adresi = CALL'dan sonraki komutun adresi; RET n parametreleri temizler.
INT / IRET	FLAGS, CS, IP (6 bayt) otomatik saklanır; vektör adresi $n \times 4$.
Parametreler	PUSH ile verilir, [BP+4], [BP+6]... ile okunur.
Güvenlik	Eşit sayıda PUSH/POP; .COM'da son RET DOS'un 0000h'sini alır.

ÖZET

Alıştırmalar

1. SS = 2000h, SP = 0100h iken `PUSH AX` sonrası yazılan baytların fiziksel adresleri nedir?
2. `MOV AX,1 / MOV BX,2 / MOV CX,3 / PUSH AX / PUSH BX / PUSH CX / POP BX / POP CX / POP AX` sonunda AX, BX, CX?
3. Yalnızca PUSH/POP ile DS'nin değerini ES'ye kopyalayın.
4. Bir prosedür girişte `PUSH BX` , `PUSH CX` yapıp çıkışta yalnızca `POP BX` yaparsa ne olur?
5. 0700:0200h'deki `CALL 0300h` (3 bayt) yürütülürken SP = FFF0h ise sonrasında SP ve [SS:SP]?
6. Üç parametrelili bir prosedürde (PUSH BP / MOV BP,SP sonrası) ilk konan parametre hangi adrestedir?

► Cevaplar

Kaynaklar

- Intel, *The 8086 Family User's Manual*, 1979 — PUSH, POP, CALL, RET, INT, IRET komut tanımları.
- B. B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson, 2009 — Bölüm 4 ve 6.
- K. R. Irvine, *Assembly Language for x86 Processors*, 7. baskı, Pearson, 2014 — yığın işlemleri ve prosedürler.
- emu8086 yardım belgeleri: *8086 Assembler Tutorial, Part 9 (The Stack)*.

Son

Bölüm 8: Yığın

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