Bölüm 8.1: Yığın

Mikroişlemciler

Yığın ve LIFO | SS:SP | PUSH / POP | Bayt sırası | Yazmaç saklama | Takas | PUSHF / POPF | 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.

PUSHF / POPF

Bayrak yazmacını yığında saklamak ve bir işlemin bozduğu bayrakları geri yüklemek.

Yığın güvenliği

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

emu8086 izleme

Programları tek adım çalıştırırken yazmaç ve yığın pencerelerini yorumlamak.

YIĞIN KAVRAMI

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 **bağıl 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

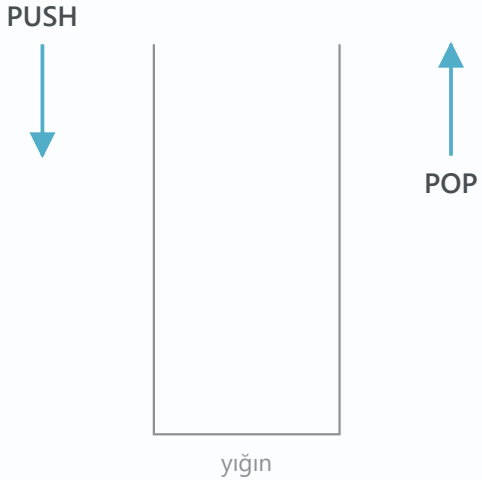
Yığın yüksek adresten düşük adrese doğru, yani **aşağı doğru** büyür.

YIĞIN KAVRAMI

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.



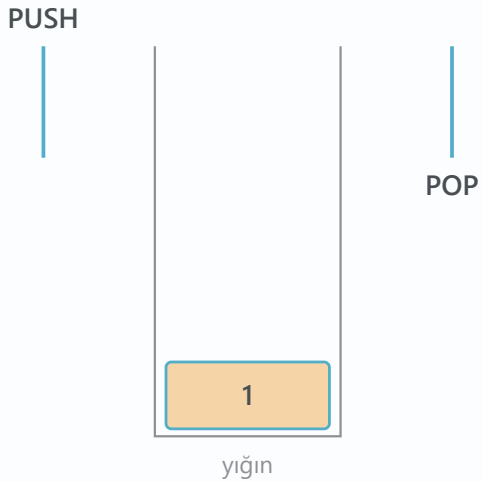
Başlangıçta yığın boş.

YIĞIN KAVRAMI

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.

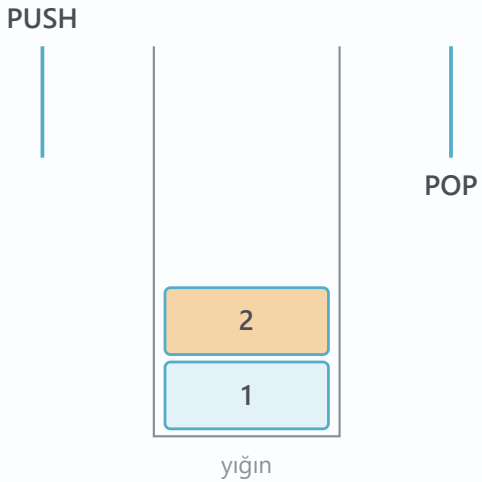


PUSH 1: 1 yığının **üstüne** konur.

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.

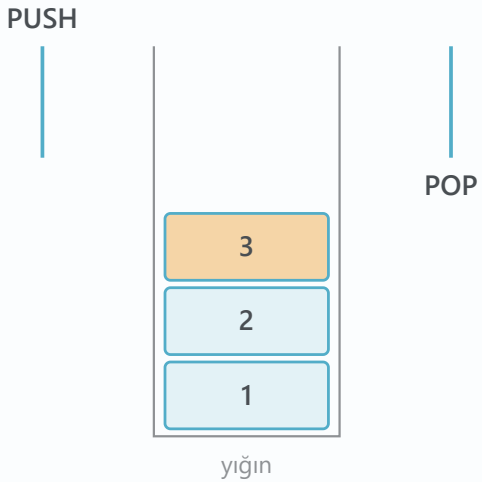


PUSH 2: 2 yığının **üstüne** konur.

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.

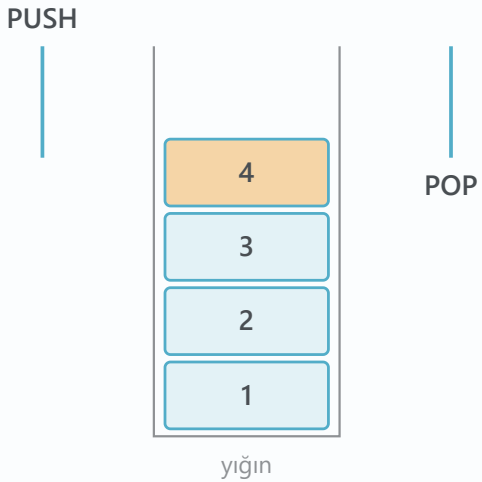


PUSH 3: 3 yığının **üstüne** konur.

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.

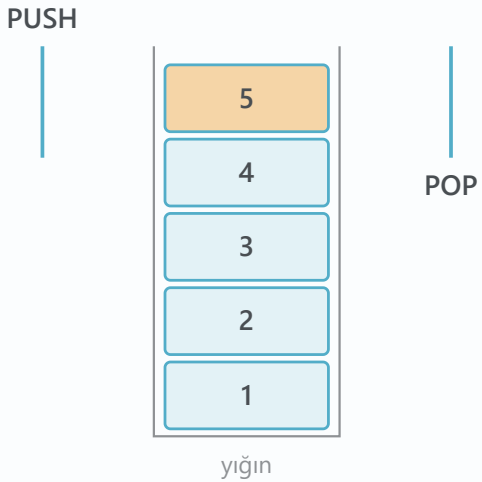


PUSH 4: 4 yığının **üstüne** konur.

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.



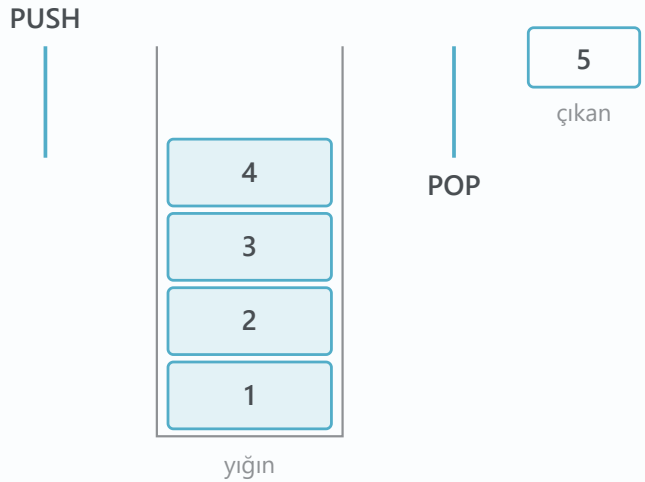
PUSH 5: 5 yığının **üstüne** konur.

YIĞIN KAVRAMI

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.

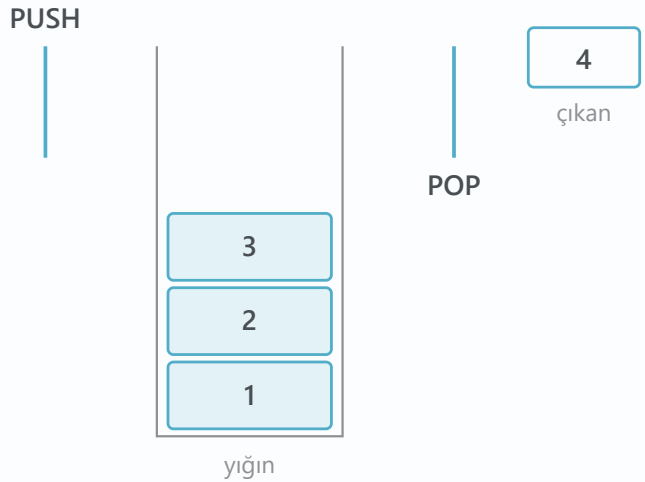


POP → **5** çıkar. Çıkış sırası şimdiye kadar: 5

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.

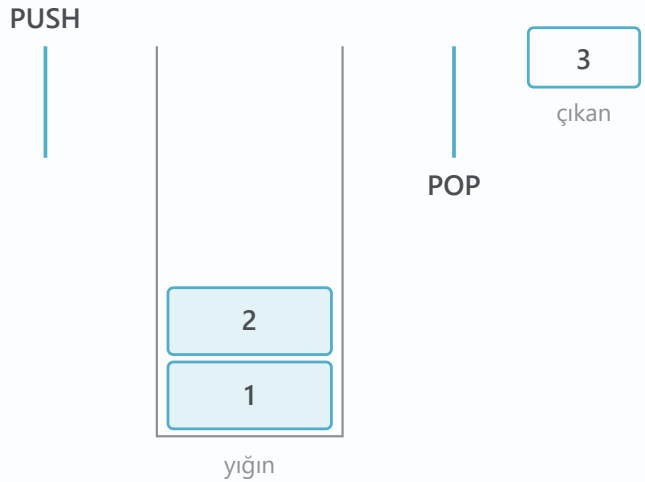


POP → **4** çıkar. Çıkış sırası şimdiye kadar: 5, 4

LIFO Algoritması

- LIFO, **Last In First Out** (son giren ilk çıkar) ifadesinin kısaltmasıdır.
- Yığına eklenen son 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.

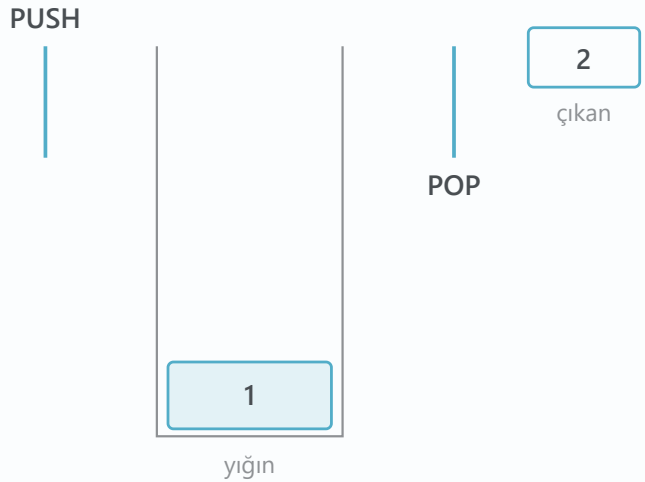


POP → **3** çıkar. Çıkış sırası şimdiye kadar: 5, 4, 3

LIFO Algoritması

- LIFO, **Last In First Out** (son giren ilk çıkar) ifadesinin kısaltmasıdır.
- Yığına eklenen son 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.



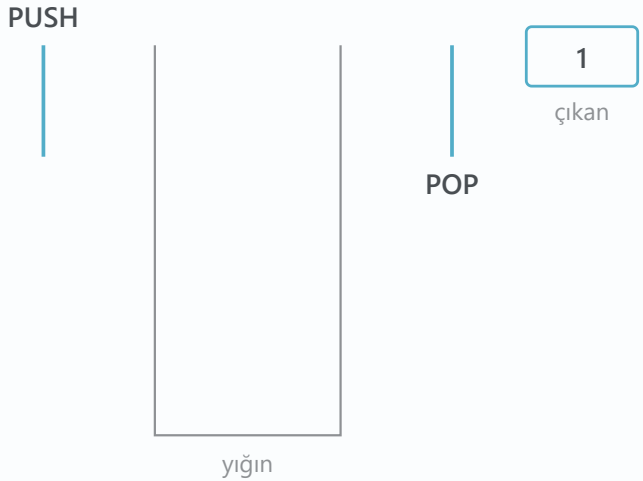
POP → **2** çıkar. Çıkış sırası şimdiye kadar: 5, 4, 3, 2

YIĞIN KAVRAMI

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.



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

YIĞIN KAVRAMI

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 = FFFEh; .EXE'de başlıktaki değerler.
- **BP** de varsayılan olarak SS kesimini kullanır; yığındaki verilere erişmek için idealdir.

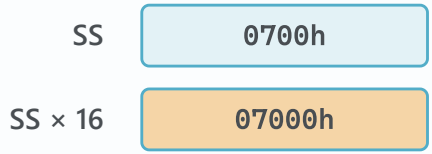


Yığın kesimi SS = **0700h** (emu8086'da .COM programı için tüm kesim yazmaçları 0700h).

YIĞIN KAVRAMI

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 = FFFEh; .EXE'de başlıktaki değerler.
- **BP** de varsayılan olarak SS kesimini kullanır; yığındaki verilere erişmek için idealdir.

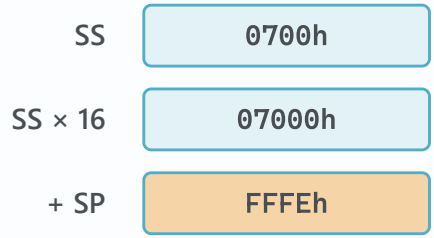


Kesim değeri 16 ile çarpılır (onaltılıkta sağa bir 0 eklenir): **07000h**.

Adım 2 / 4

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 = FFFEh; .EXE'de başlıktaki değerler.
- **BP** de varsayılan olarak SS kesimini kullanır; yığındaki verilere erişmek için idealdir.

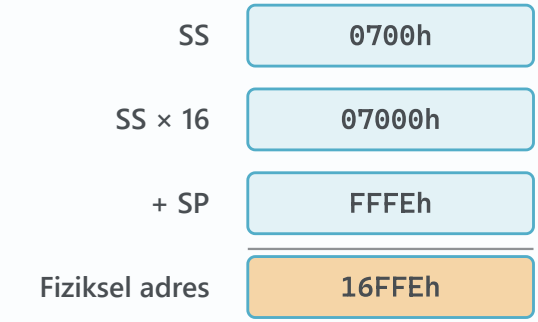


Bağıl konum olarak yığın işaretçisi SP = **FFFEh** eklenir.

YIĞIN KAVRAMI

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 = FFFEh; .EXE'de başlıktaki değerler.
- **BP** de varsayılan olarak SS kesimini kullanır; yığındaki verilere erişmek için idealdir.



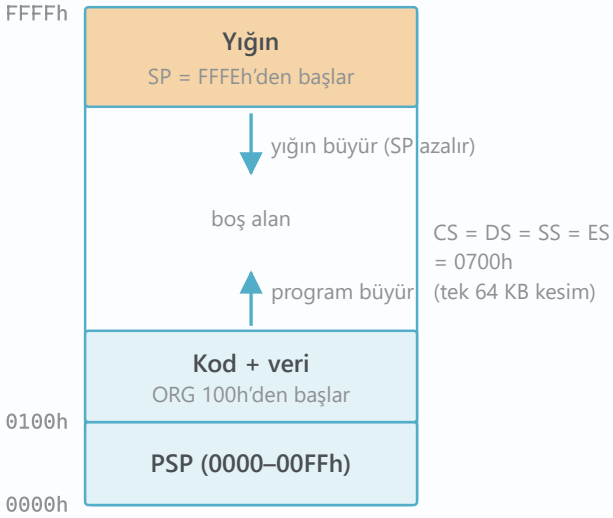
Yığının üstü fiziksel olarak **16FFEh** adresindedir (20 bit).

Adım 4 / 4

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 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 sayi`
`PUSH immediate`	5, −24, 3Fh, 10001101b...	`PUSH 3Fh`

`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.

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 Nasıl Çalışır?

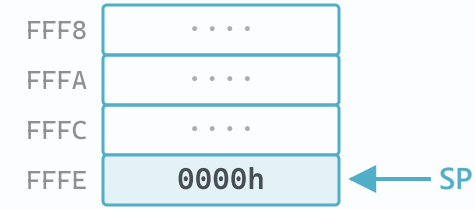
PUSH kaynak

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:bağıl konum · küçük adres yukarıda

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

Adım 1 / 5

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:bağıl konum · küçük adres yukarıda

PUSH AX · 1

`PUSH AX` ① SP ← SP − 2 = **FFFCh**. Yeni üst hücre henüz yazılmadı.

PUSH ve POP Nasıl Çalışır?

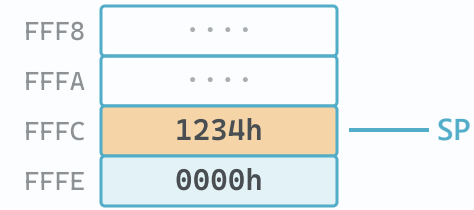
PUSH kaynak

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:bağıl konum · küçük adres yukarıda

PUSH AX · 2

② Kaynak değer SS:SP adresine yazılır: [FFFCh] ← **1234h**.

PUSH ve POP Nasıl Çalışır?

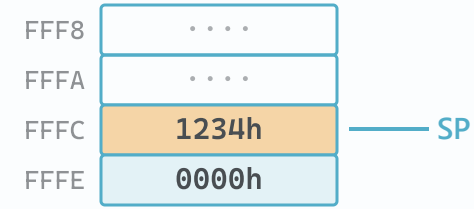
PUSH kaynak

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.



SS:bağıl konum · küçük adres yukarıda

POP BX · 1

`POP BX` ① SS:SP'deki değer hedefe yazılır: BX ← **1234h**.

PUSH ve POP Nasıl Çalışır?

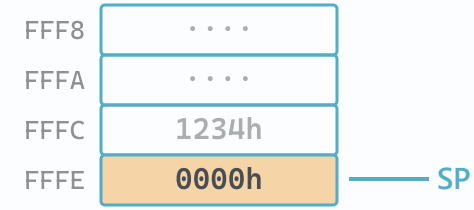
PUSH kaynak

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.

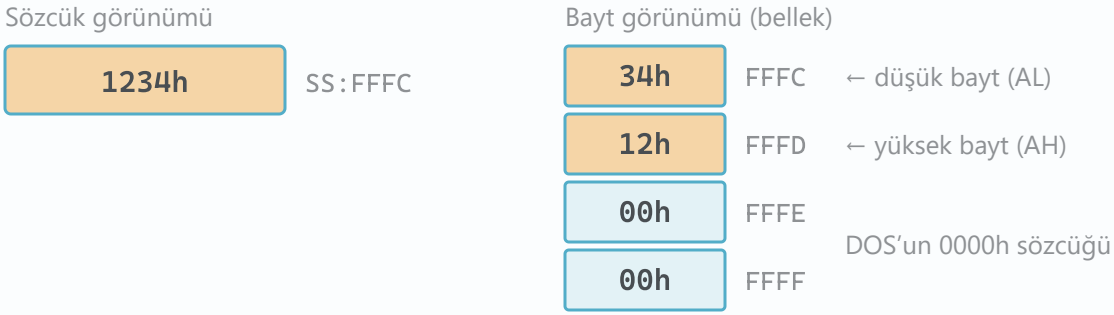


POP BX · 2

② SP ← SP + 2 = **FFFEh**. FFFCh'deki 1234h **silinmez**; yalnızca artık yığının dışında kalır (soluk).

PUSH VE POP

Yığında Bayt Sırası (Little-Endian)



- 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

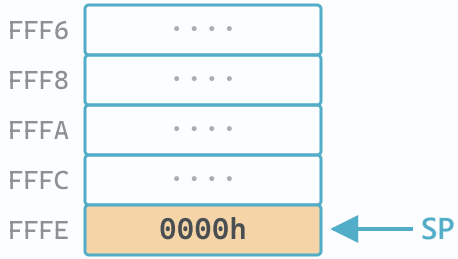
Ne Değişir, Ne Değişmez?

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 karşılaştırmanın sonucu (bayraklar) PUSH/POP arasında güvenle taşınabilir; bayrağı saklamak gerekiyorsa PUSHF/POPF kullanılır.

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
  dön
6 END
```



SS:bağıl konum · küçük adres yukarıda

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

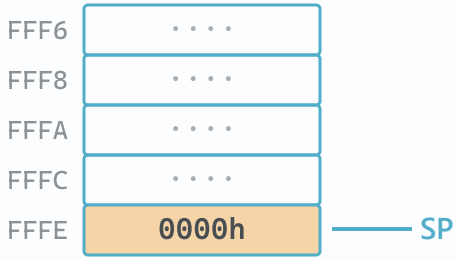
Adım 1 / 5

DEĞİŞKENLER

`AX = 0000h` `BX = 0000h` `SP = FFFEh` `IP = 0100h`

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:bağıl konum · küçük adres yukarıda

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

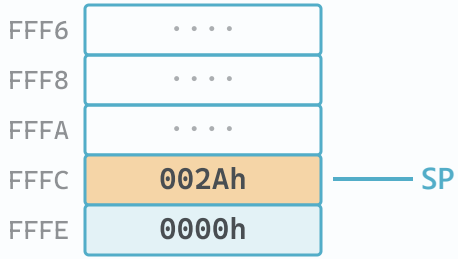
Adım 2 / 5

DEĞİŞKENLER

`AX = 002Ah` `BX = 0000h` `SP = FFFEh` `IP = 0103h`

İlk Örnek: `PUSH AX` / `POP BX`

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



SS:bağıl konum · küçük adres yukarıda

`PUSH AX` : SP = FFFEh − 2 = **FFFCh**; [FFFCh] ← 002Ah.

Adım 3 / 5

DEĞİŞKENLER

`AX = 002Ah` `BX = 0000h` `SP = FFFCh` `IP = 0104h`

İlk Örnek: `PUSH AX` / `POP BX`

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



SS:bağıl konum · küçük adres yukarıda

`POP BX`: BX ← [FFFCh] = 002Ah; SP tekrar **FFFEh**. Değer AX'ten BX'e yığın üzerinden kopyalandı.

Adım 4 / 5

DEĞİŞKENLER

`AX = 002Ah` `BX = 002Ah` `SP = FFFEh` `IP = 0105h`

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
```



SS:bağıl konum · küçük adres yukarıda

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

Adım 5 / 5

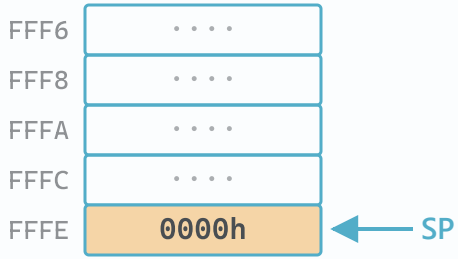
DEĞİŞKENLER

`AX = 002Ah` `BX = 002Ah` `SP = 0000h` `IP = 0000h`

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

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:bağıl konum · küçük adres yukarıda

Başlangıç: SP = FFFEh.

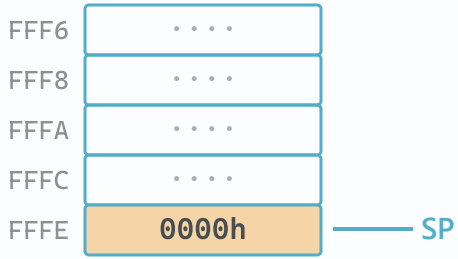
Adım 1 / 10

DEĞİŞKENLER

AX = 0000h BX = 0000h SP = FFFEh

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:bağıl konum · küçük adres yukarıda

AX = 0001h.

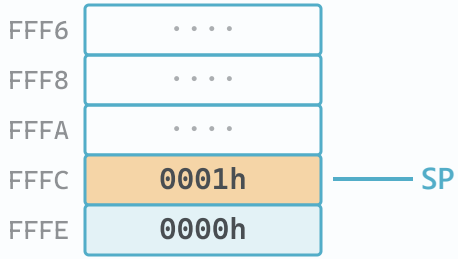
Adım 2 / 10

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
▶  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:bağıl konum · küçük adres yukarıda

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

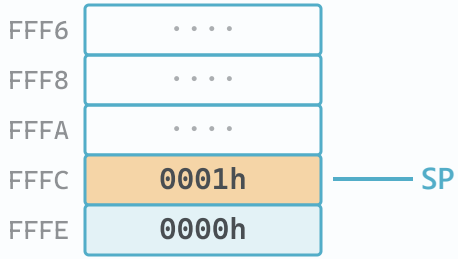
Adım 3 / 10

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
▶ 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:bağıl konum · küçük adres yukarıda

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

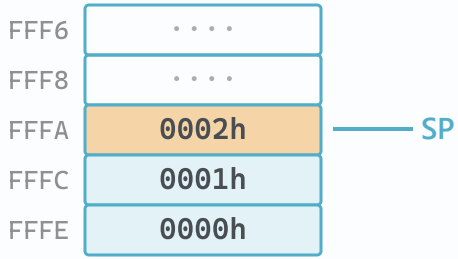
Adım 4 / 10

DEĞİŞKENLER

AX = 0002h 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
▶ 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:bağıl konum · küçük adres yukarıda

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

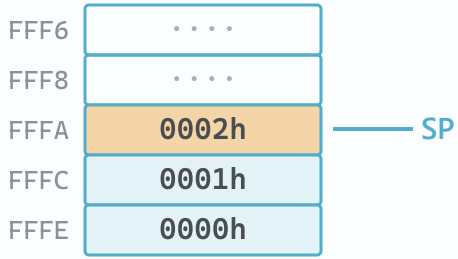
Adım 5 / 10

DEĞİŞKENLER

AX = 0002h | BX = 0000h | 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
▶ 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:bağıl konum · küçük adres yukarıda

AX = 0003h.

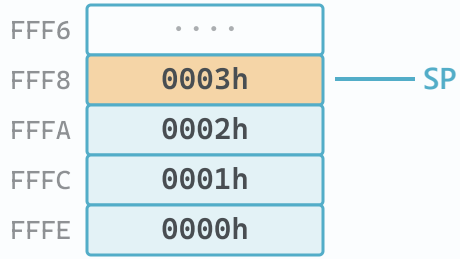
Adım 6 / 10

DEĞİŞKENLER

AX = 0003h | BX = 0000h | 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
▶ PUSH  AX         ; 3'ü yığına koy
7 POP   BX         ; BX ← 3
8 POP   BX         ; BX ← 2
9 POP   BX         ; BX ← 1
```



SS:bağıl konum · küçük adres yukarıda

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

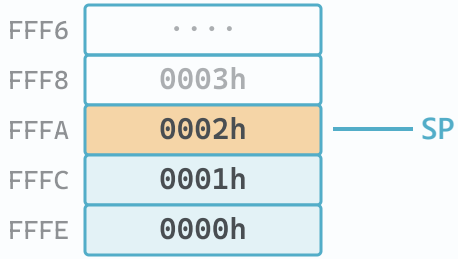
Adım 7 / 10

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
▶ POP   BX         ; BX ← 3
8 POP   BX         ; BX ← 2
9 POP   BX         ; BX ← 1
```



SS:bağıl konum · küçük adres yukarıda

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

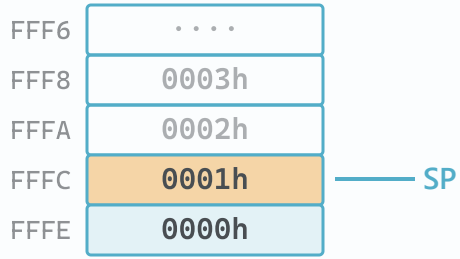
Adım 8 / 10

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
▶ POP   BX         ; BX ← 2
9 POP   BX         ; BX ← 1
```



SS:bağıl konum · küçük adres yukarıda

POP BX → **2**. SP = FFFCh.

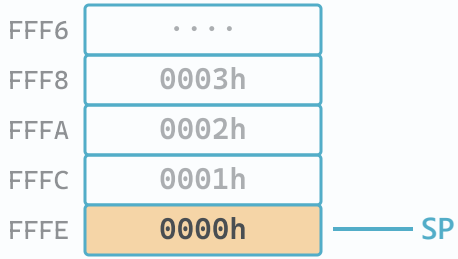
Adım 9 / 10

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:bağıl konum · küçük adres yukarıda

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

Adım 10 / 10

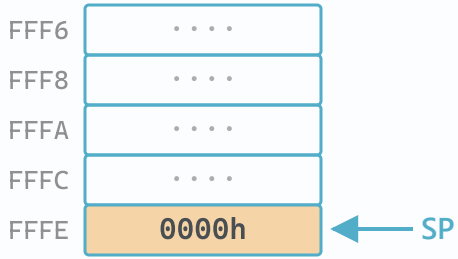
DEĞİŞKENLER

AX = **0003h** BX = **0001h** SP = **FFFEh**

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

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:bağıl konum · küçük adres yukarıda

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

Adım 1 / 6

DEĞİŞKENLER

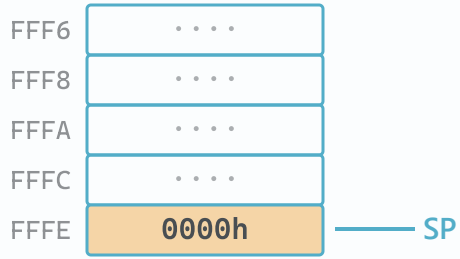
AX = 0000h SP = FFFEh

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

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

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:bağıl konum · küçük adres yukarıda

AX = 1234h.

Adım 2 / 6

DEĞİŞKENLER

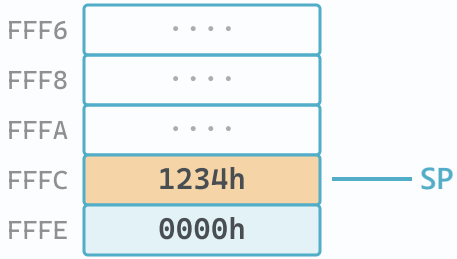
AX = 1234h SP = FFFEh

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

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Bir Yazmacın Değerini Saklamak

```
1 ORG   100h
2 MOV   AX, 1234h
▶ 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:bağıl konum · küçük adres yukarıda

AX yığına saklandı: [FFFCh] = 1234h.

Adım 3 / 6

DEĞİŞKENLER

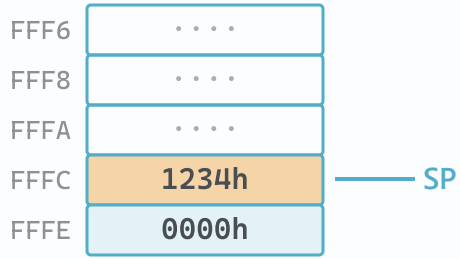
AX = 1234h SP = FFFCh

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

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Bir Yazmacın Değerini Saklamak

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



SS:bağıl konum · küçük adres yukarıda

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

Adım 4 / 6

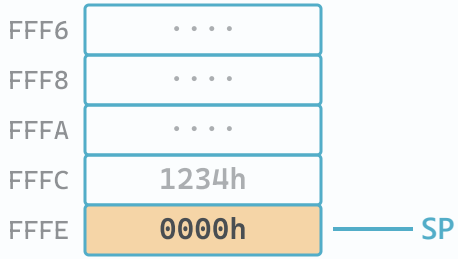
DEĞİŞKENLER

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
▶ POP   AX          ; eski değeri geri
  yükle
6 RET
7 END
```



SS:bağıl konum · küçük adres yukarıda

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

Adım 5 / 6

DEĞİŞKENLER

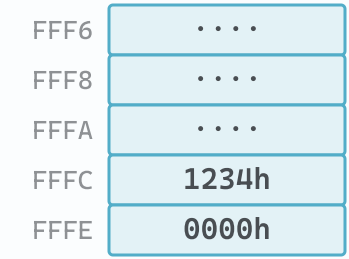
`AX = 1234h` `SP = FFFEh`

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

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

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
```



SS:bağıl konum · küçük adres yukarıda

RET: 0000h'ye dönüş, program biter.

Adım 6 / 6

DEĞİŞKENLER

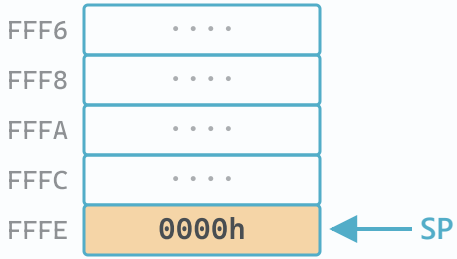
AX = 1234h SP = 0000h

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

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

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:bağıl konum · küçük adres yukarıda

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

Adım 1 / 8

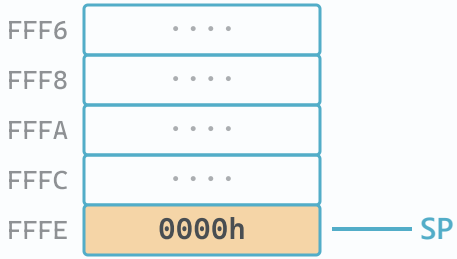
DEĞİŞKENLER

AX = 0000h BX = 0000h CX = 000Bh SP = FFFEh

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

Yığın ile İki Yazmacı Takas Etmek

```
1 ORG   100h
▶ 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:bağıl konum · küçük adres yukarıda

AX = 1212h.

Adım 2 / 8

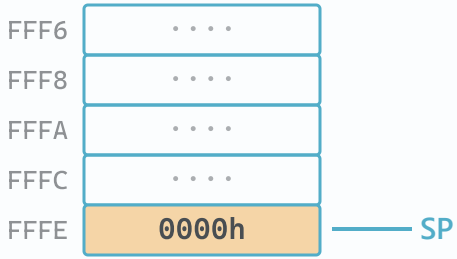
DEĞİŞKENLER

AX = 1212h | BX = 0000h | CX = 000Bh | SP = FFFEh

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

Yığın ile İki Yazmacı Takas Etmek

```
1 ORG   100h
2 MOV   AX, 1212h  ; AX ← 1212h
▶ 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:bağıl konum · küçük adres yukarıda

BX = 3434h.

Adım 3 / 8

DEĞİŞKENLER

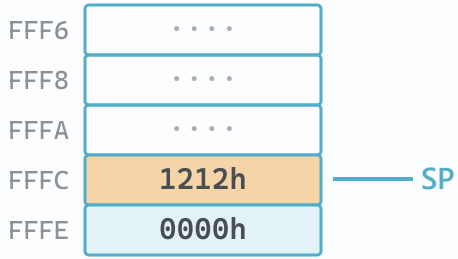
AX = **1212h** | BX = **3434h** | CX = **000Bh** | SP = **FFFEh**

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

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Yığın ile İki Yazmacı Takas Etmek

```
1 ORG   100h
2 MOV   AX, 1212h  ; AX ← 1212h
3 MOV   BX, 3434h  ; BX ← 3434h
▶ 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:bağıl konum · küçük adres yukarıda

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

Adım 4 / 8

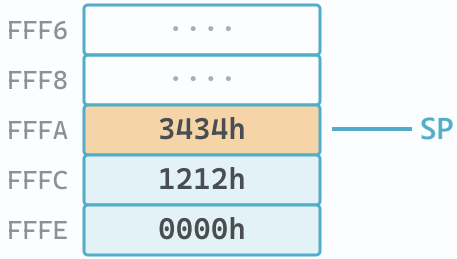
DEĞİŞKENLER

AX = 1212h BX = 3434h CX = 000Bh SP = FFFCh

Aynı sırayla POP edilseydi (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
▶ 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:bağıl konum · küçük adres yukarıda

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

Adım 5 / 8

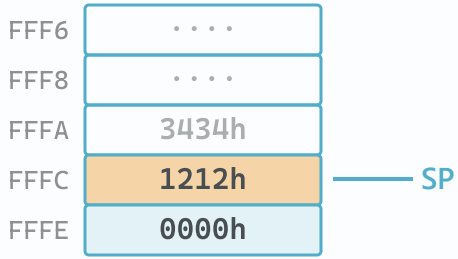
DEĞİŞKENLER

AX = **1212h** BX = **3434h** CX = **000Bh** SP = **FFFAh**

Aynı sırayla POP edilseydi (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
▶ POP   AX         ; AX ← BX'in eski
  değeri
7 POP   BX         ; BX ← AX'in eski
  değeri
8 RET
9 END
```



SS:bağıl konum · küçük adres yukarıda

POP AX: üstteki **3434h** AX'e gelir.

Adım 6 / 8

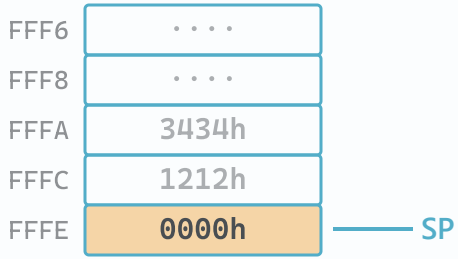
DEĞİŞKENLER

AX = **3434h** BX = **3434h** CX = **000Bh** SP = **FFFCh**

Aynı sırayla POP edilseydi (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
▶ POP   BX         ; BX ← AX'in eski
  değeri
8 RET
9 END
```



SS:bağıl konum · küçük adres yukarıda

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

Adım 7 / 8

DEĞİŞKENLER

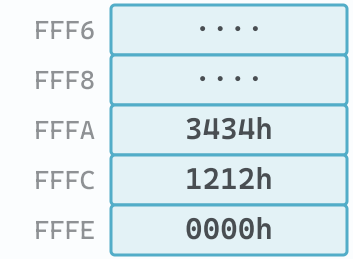
AX = **3434h** BX = **1212h** CX = **000Bh** SP = **FFFEh**

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

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

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
▶ RET
9 END
```



SS:bağıl konum · küçük adres yukarıda

RET: program sonu.

Adım 8 / 8

DEĞİŞKENLER

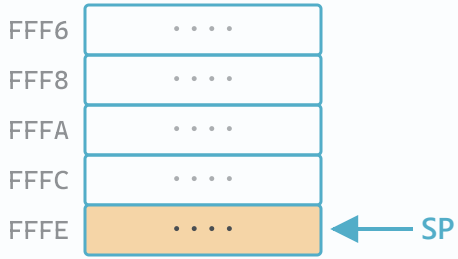
AX = 3434h BX = 1212h CX = 000Bh SP = 0000h

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

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

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:bağıl konum · küçük adres yukarıda

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

Adım 1 / 10

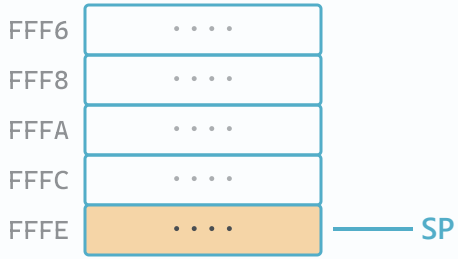
DEĞİŞKENLER

AX = **0000h** BX = **0000h** CX = **0000h** SP = **FFFEh**

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:bağıl konum · küçük adres yukarıda

AX = 0002h.

Adım 2 / 10

DEĞİŞKENLER

AX = **0002h** BX = **0000h** CX = **0000h**

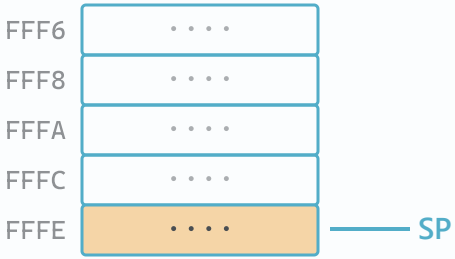
SP = **FFFEh**

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

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

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:bağıl konum · küçük adres yukarıda

BX = 0004h.

Adım 3 / 10

DEĞİŞKENLER

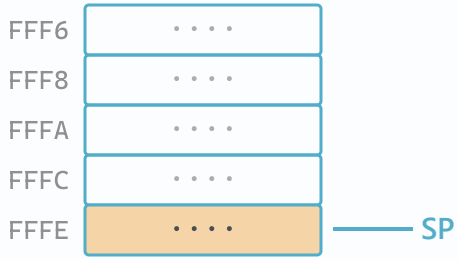
AX = 0002h BX = 0004h CX = 0000h SP = FFFEh

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

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Yığın Kullanımı: Üç Yazmaç

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



SS:bağıl konum · küçük adres yukarıda

CX = 0008h.

Adım 4 / 10

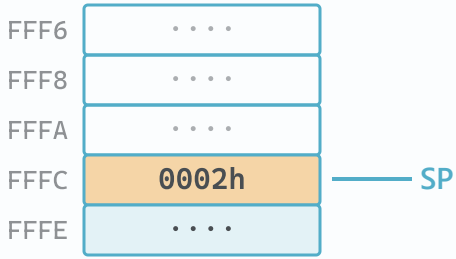
DEĞİŞKENLER

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
▶ push ax
5 push bx
6 push cx
7 pop  ax
8 pop  bx
9 pop  cx
```



SS:bağıl konum · küçük adres yukarıda

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

Adım 5 / 10

DEĞİŞKENLER

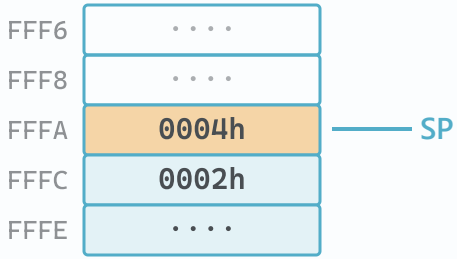
AX = **0002h** BX = **0004h** CX = **0008h**

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
▶ push bx
6 push cx
7 pop  ax
8 pop  bx
9 pop  cx
```



SS:bağıl konum · küçük adres yukarıda

push bx: SP = FFFAh, [FFFAh] = 0004h.

Adım 6 / 10

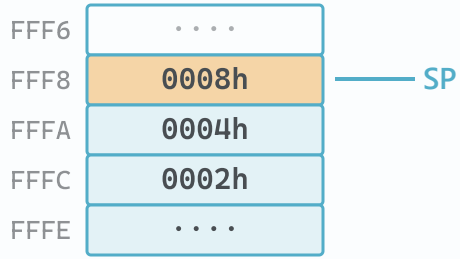
DEĞİŞKENLER

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:bağıl konum · küçük adres yukarıda

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

Adım 7 / 10

DEĞİŞKENLER

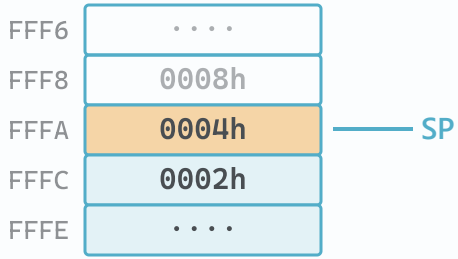
AX = **0002h** BX = **0004h** CX = **0008h**

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
▶ pop  ax
8 pop  bx
9 pop  cx
```



SS:bağıl konum · küçük adres yukarıda

pop ax: AX ← 0008h (üstteki).

Adım 8 / 10

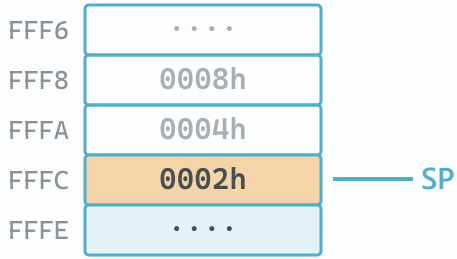
DEĞİŞKENLER

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
▶ pop  bx
9 pop  cx
```



SS:bağıl konum · küçük adres yukarıda

pop bx: BX ← 0004h (ortadaki; değişmedi).

Adım 9 / 10

DEĞİŞKENLER

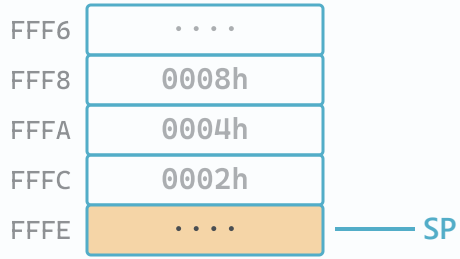
AX = **0008h** BX = **0004h** CX = **0008h**

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:bağıl konum · küçük adres yukarıda

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

Adım 10 / 10

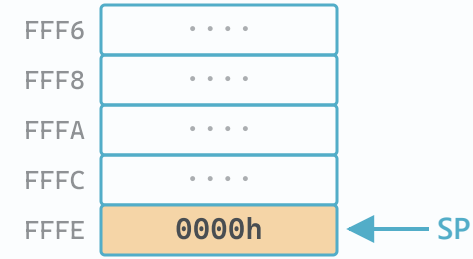
DEĞİŞKENLER

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

```
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:bağıl konum · küçük adres yukarıda

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

Adım 1 / 29

DEĞİŞKENLER

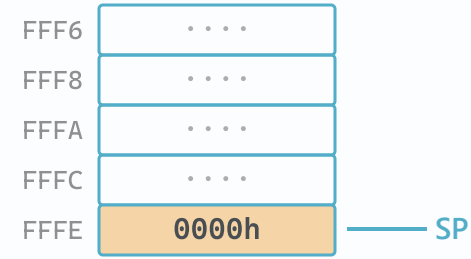
AX = 0000h CX = 0000h DX = 0000h

SI = 0000h SP = FFFEh

ÇIKTI

Yığınla Diziyi Ters Çevirme

```
1  ORG  100h
▶  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:bağıl konum · küçük adres yukarıda

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

Adım 2 / 29

DEĞİŞKENLER

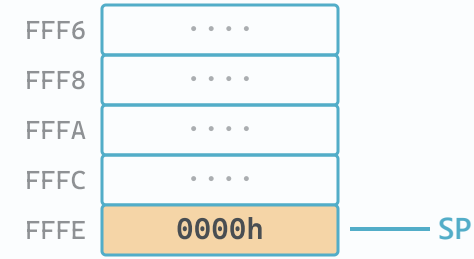
AX = **0000h** CX = **0000h** DX = **0000h**

SI = **0117h** SP = **FFFEh**

ÇIKTI

Yığınla Diziyi Ters Çevirme

```
1  ORG  100h
2  MOV  SI, OFFSET metin
▶  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:bağıl konum · küçük adres yukarıda

Döngü sayacı CX = 3.

Adım 3 / 29

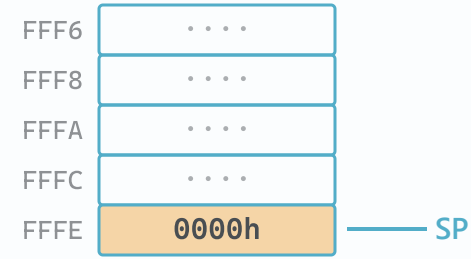
DEĞİŞKENLER

AX = 0000h | CX = 0003h | DX = 0000h | SI = 0117h | SP = FFFEh

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:bağıl konum · küçük adres yukarıda

AL ← [SI] = 'A' (41h).

Adım 4 / 29

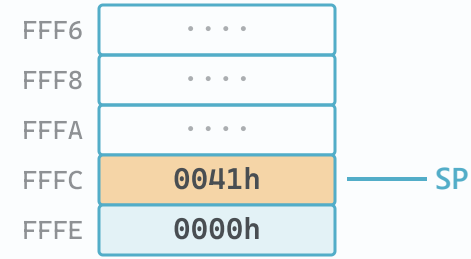
DEĞİŞKENLER

AX = 0041h | CX = 0003h | DX = 0000h | SI = 0117h | SP = FFFEh

ÇIKTI

Yığınla Diziyi Ters Çevirme

```
1  ORG  100h
2  MOV  SI, OFFSET metin
3  MOV  CX, 3
4  koy: MOV  AL, [SI]
▶       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:bağıl konum · küçük adres yukarıda

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

Adım 5 / 29

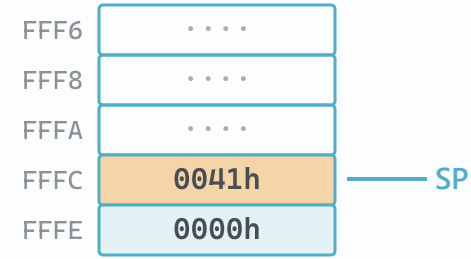
DEĞİŞKENLER

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
▶       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:bağıl konum · küçük adres yukarıda

SI = 0118h: sonraki karakter.

Adım 6 / 29

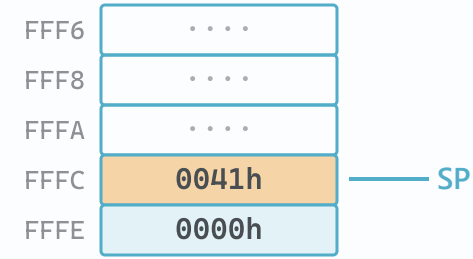
DEĞİŞKENLER

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
▶       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:bağıl konum · küçük adres yukarıda

LOOP: CX = 2 → geri dön.

Adım 7 / 29

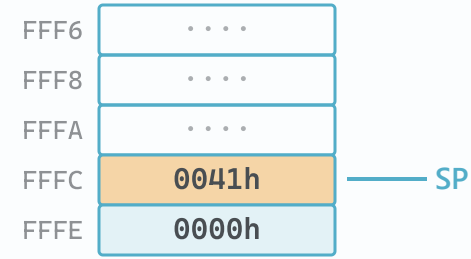
DEĞİŞKENLER

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
▶ 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:bağıl konum · küçük adres yukarıda

AL ← [SI] = 'B' (42h).

Adım 8 / 29

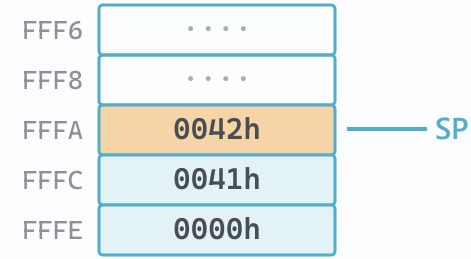
DEĞİŞKENLER

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]
▶       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:bağıl konum · küçük adres yukarıda

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

Adım 9 / 29

DEĞİŞKENLER

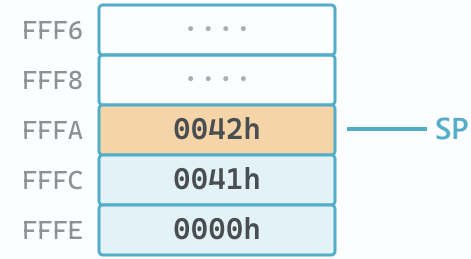
AX = 0042h CX = 0002h DX = 0000h

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
▶       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:bağıl konum · küçük adres yukarıda

SI = 0119h: sonraki karakter.

Adım 10 / 29

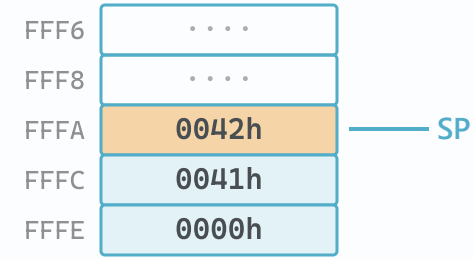
DEĞİŞKENLER

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
▶       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:bağıl konum · küçük adres yukarıda

LOOP: CX = 1 → geri dön.

Adım 11 / 29

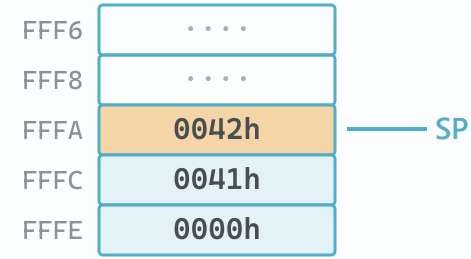
DEĞİŞKENLER

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:bağıl konum · küçük adres yukarıda

AL ← [SI] = 'C' (43h).

Adım 12 / 29

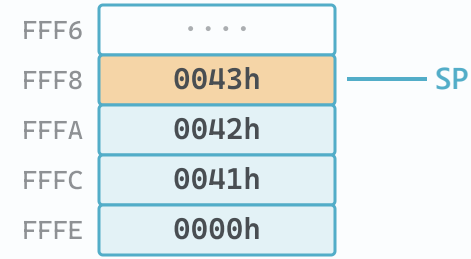
DEĞİŞKENLER

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]
▶       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:bağıl konum · küçük adres yukarıda

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

Adım 13 / 29

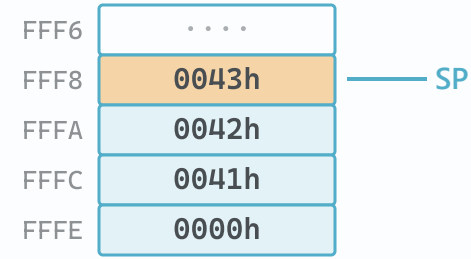
DEĞİŞKENLER

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:bağıl konum · küçük adres yukarıda

SI = 011Ah: sonraki karakter.

Adım 14 / 29

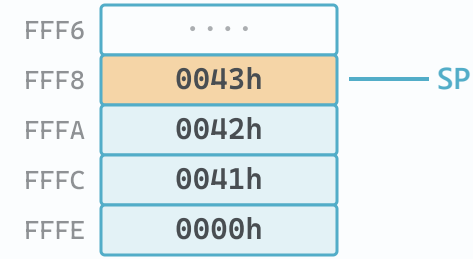
DEĞİŞKENLER

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
▶       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:bağıl konum · küçük adres yukarıda

LOOP: CX = 0 → döngü biter.

Adım 15 / 29

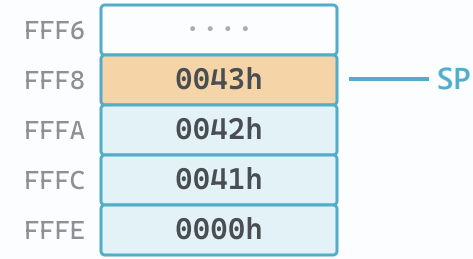
DEĞİŞKENLER

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
▶  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:bağıl konum · küçük adres yukarıda

İkinci döngü için CX = 3.

Adım 16 / 29

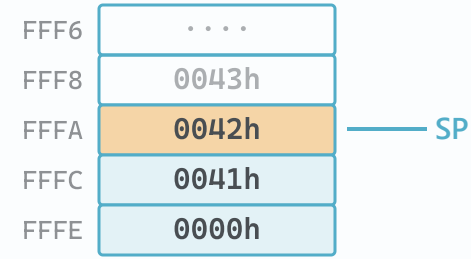
DEĞİŞKENLER

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
▶  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:bağıl konum · küçük adres yukarıda

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

Adım 17 / 29

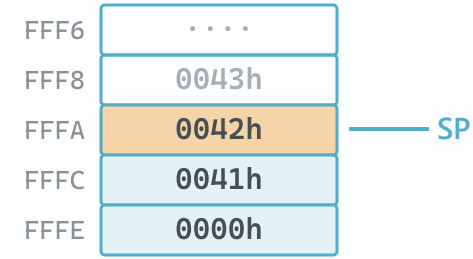
DEĞİŞKENLER

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
▶       MOV  AH, 2
11      INT  21h       ; DL'yi ekrana yaz
12      LOOP yaz
13 RET
14 metin DB 'ABC'    ; 0117h
```



SS:bağıl konum · küçük adres yukarıda

AH = 02h: DOS karakter yazma işlevi.

Adım 18 / 29

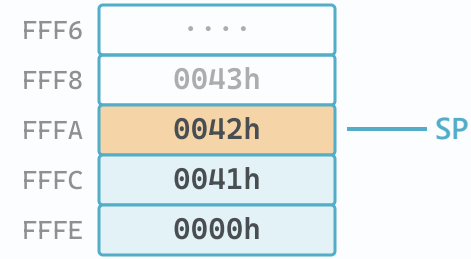
DEĞİŞKENLER

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
▶       INT  21h       ; DL'yi ekrana yaz
12      LOOP yaz
13 RET
14 metin DB 'ABC'    ; 0117h
```



SS:bağıl konum · küçük adres yukarıda

INT 21h: ekrana **C** yazılır; AL = DL = 43h olur.

Adım 19 / 29

DEĞİŞKENLER

AX = **0243h** CX = **0003h** DX = **0043h**

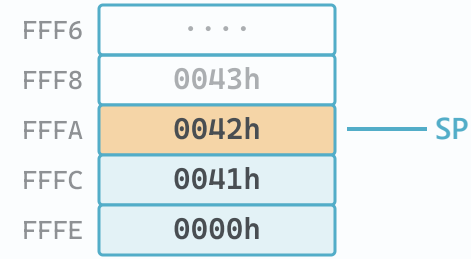
SI = **011Ah** SP = **FFFAh**

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
▶       LOOP yaz
13 RET
14 metin DB 'ABC'    ; 0117h
```



SS:bağıl konum · küçük adres yukarıda

LOOP: CX = 2 → geri dön.

Adım 20 / 29

DEĞİŞKENLER

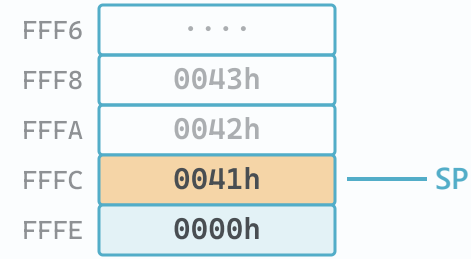
AX = 0243h | CX = 0002h | DX = 0043h | SI = 011Ah | SP = FFFAh

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
▶  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:bağıl konum · küçük adres yukarıda

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

Adım 21 / 29

DEĞİŞKENLER

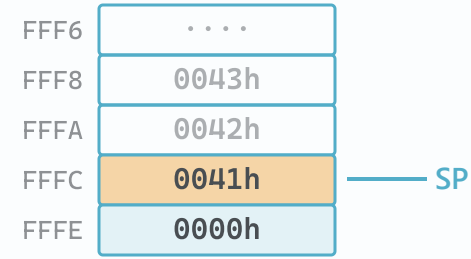
AX = 0243h | CX = 0002h | DX = 0042h | SI = 011Ah | SP = FFFCh

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
▶       MOV  AH, 2
11      INT  21h       ; DL'yi ekrana yaz
12      LOOP yaz
13 RET
14 metin DB 'ABC'    ; 0117h
```



SS:bağıl konum · küçük adres yukarıda

AH = 02h: DOS karakter yazma işlevi.

Adım 22 / 29

DEĞİŞKENLER

AX = **0243h** CX = **0002h** DX = **0042h**

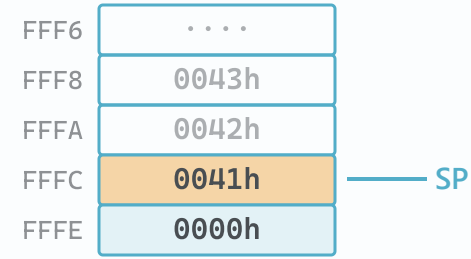
SI = **011Ah** SP = **FFFCh**

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
▶       INT  21h       ; DL'yi ekrana yaz
12      LOOP yaz
13 RET
14 metin DB 'ABC'    ; 0117h
```



SS:bağıl konum · küçük adres yukarıda

INT 21h: ekrana **B** yazılır; AL = DL = 42h olur.

Adım 23 / 29

DEĞİŞKENLER

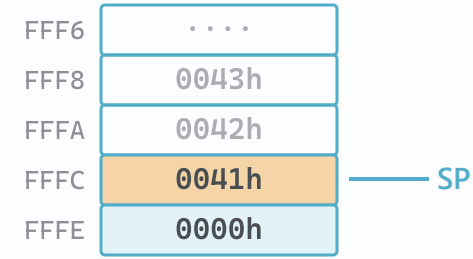
AX = **0242h** CX = **0002h** DX = **0042h**
SI = **011Ah** SP = **FFFCh**

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
▶       LOOP yaz
13 RET
14 metin DB 'ABC'    ; 0117h
```



SS:bağıl konum · küçük adres yukarıda

LOOP: CX = 1 → geri dön.

Adım 24 / 29

DEĞİŞKENLER

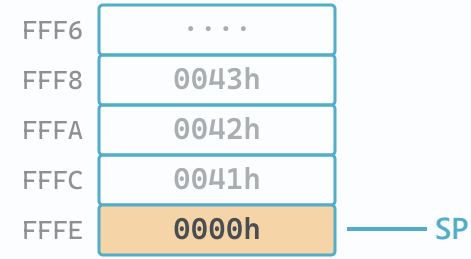
AX = 0242h | CX = 0001h | DX = 0042h | SI = 011Ah | SP = FFFCh

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
▶  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:bağıl konum · küçük adres yukarıda

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

Adım 25 / 29

DEĞİŞKENLER

AX = **0242h** CX = **0001h** DX = **0041h**

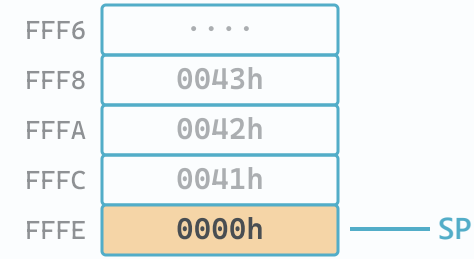
SI = **011Ah** SP = **FFFEh**

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
▶       MOV  AH, 2
11      INT  21h       ; DL'yi ekrana yaz
12      LOOP yaz
13 RET
14 metin DB 'ABC'    ; 0117h
```



SS:bağıl konum · küçük adres yukarıda

AH = 02h: DOS karakter yazma işlevi.

Adım 26 / 29

DEĞİŞKENLER

AX = **0242h** CX = **0001h** DX = **0041h**

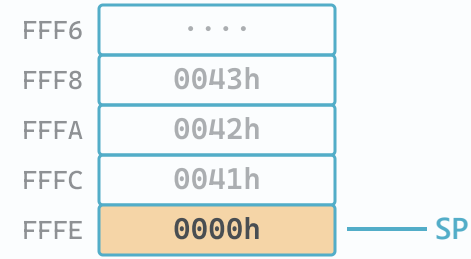
SI = **011Ah** SP = **FFFEh**

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
▶       INT  21h       ; DL'yi ekrana yaz
12      LOOP yaz
13 RET
14 metin DB 'ABC'    ; 0117h
```



SS:bağıl konum · küçük adres yukarıda

INT 21h: ekrana **A** yazılır; AL = DL = 41h olur.

Adım 27 / 29

DEĞİŞKENLER

AX = **0241h** CX = **0001h** DX = **0041h**

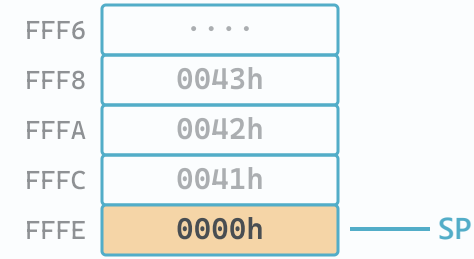
SI = **011Ah** SP = **FFFEh**

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
▶       LOOP yaz
13 RET
14 metin DB 'ABC'    ; 0117h
```



SS:bağıl konum · küçük adres yukarıda

LOOP: CX = 0 → döngü biter.

Adım 28 / 29

DEĞİŞKENLER

AX = **0241h** | CX = **0000h** | DX = **0041h**

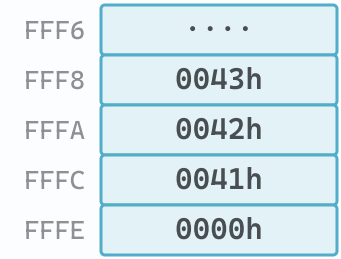
SI = **011Ah** | SP = **FFFEh**

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
```



SS:bağıl konum · küçük adres yukarıda

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

Adım 29 / 29

DEĞİŞKENLER

AX = 0241h | CX = 0000h | DX = 0041h | SI = 011Ah | SP = 0000h

ÇIKTI

```
CBA
```

ÖRNEKLER

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
```

Adım 1 / 5

ÖRNEKLER

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.

ÖRNEKLER

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.

ÖRNEKLER

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**.

ÖRNEKLER

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).

ÖRNEKLER

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

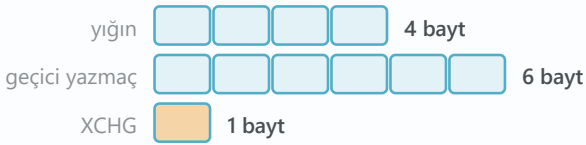
```
MOV CX, AX
MOV AX, BX
MOV BX, CX
```

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

XCHG ile

```
XCHG AX, BX
```

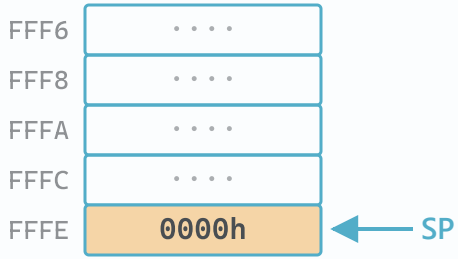
1 bayt (93h) · en kısa ve en hızlı yol. Kesim yazmaçlarıyla kullanılamaz.



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

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:bağıl konum · küçük adres yukarıda

Başlangıç: bayraklar temiz (IF = 1).

Adım 1 / 8

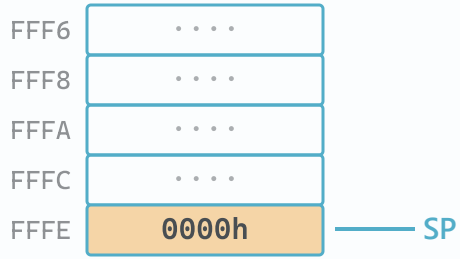
DEĞİŞKENLER

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
▶ 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:bağıl konum · küçük adres yukarıda

AX = FFFFh.

Adım 2 / 8

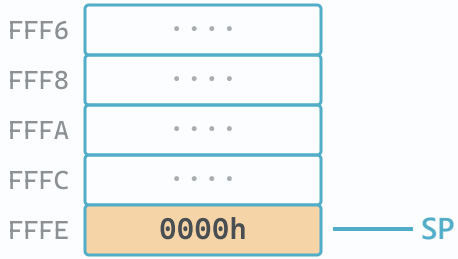
DEĞİŞKENLER

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
▶ 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:bağıl konum · küçük adres yukarıda

FFFFh + 1 = 1 0000h → AX = 0000h, **CF = 1, ZF = 1**.

Adım 3 / 8

DEĞİŞKENLER

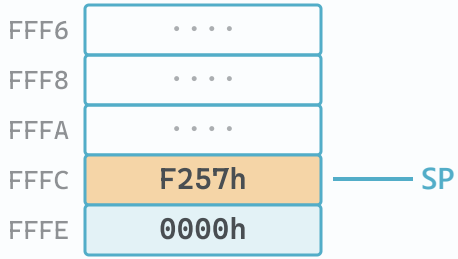
AX = **0000h** BX = **0000h** SP = **FFFEh**

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
▶ PUSHF             ; bayrakları sakla
5 MOV   BX, 5
6 ADD   BX, 3       ; CF=0, ZF=0
7 POPF              ; bayraklar geri
8 RET
```



SS:bağıl konum · küçük adres yukarıda

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

Adım 4 / 8

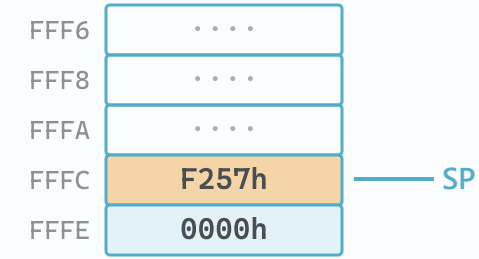
DEĞİŞKENLER

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
▶ MOV   BX, 5
6 ADD   BX, 3       ; CF=0, ZF=0
7 POPF              ; bayraklar geri
8 RET
```



SS:bağıl konum · küçük adres yukarıda

BX = 5.

Adım 5 / 8

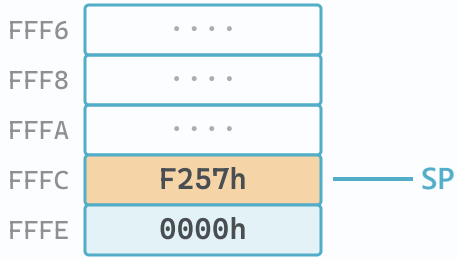
DEĞİŞKENLER

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
▶ ADD   BX, 3       ; CF=0, ZF=0
7 POPF              ; bayraklar geri
8 RET
```



SS:bağıl konum · küçük adres yukarıda

5 + 3 = 8 → CF = 0, ZF = 0. Önceki sonuç bayrakları kayboldu...

Adım 6 / 8

DEĞİŞKENLER

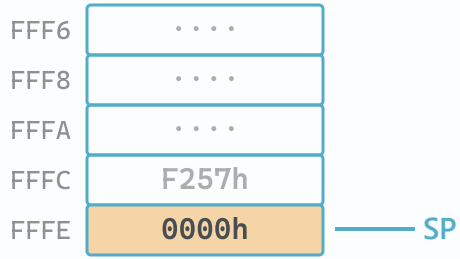
AX = **0000h** BX = **0008h** SP = **FFFCh**

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
▶ POPF              ; bayraklar geri
8 RET
```



SS:bağıl konum · küçük adres yukarıda

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

Adım 7 / 8

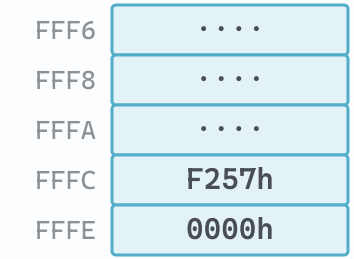
DEĞİŞKENLER

AX = 0000h | BX = 0008h | 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              ; bayraklar geri
▶ RET
```



SS:bağıl konum · küçük adres yukarıda

Program biter.

Adım 8 / 8

DEĞİŞKENLER

AX = 0000h | BX = 0008h | 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

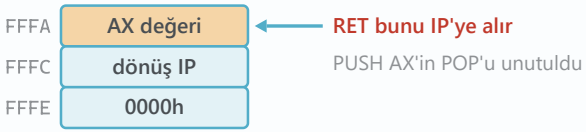
RET, dönüş adresi yerine son konan veriyi alır → rastgele adrese atlama.

Fazla POP

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

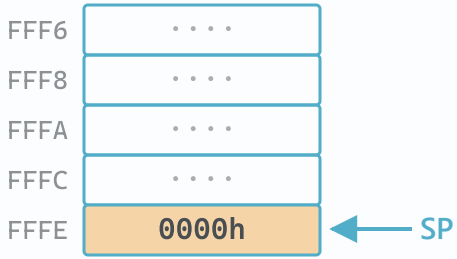
Yanlış sıra

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



Hata Örneği: Dengesiz PUSH

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



SS:bağıl konum · küçük adres yukarıda

Program sonunda RET ile DOS'a dönülmesi bekleniyor.

Adım 1 / 4

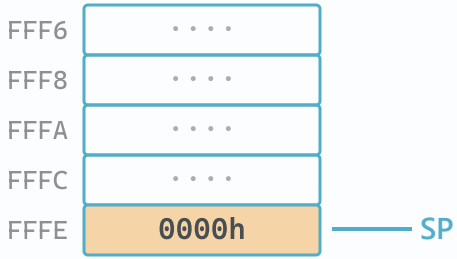
DEĞİŞKENLER

AX = 0000h SP = FFFEh IP = 0100h

Düzeltme: RET'ten önce POP AX eklenir ya da hiç PUSH yapılmaz. Her dalın (JMP/Jcc) PUSH/POP sayısı ayrı ayrı sayılır.

Hata Örneği: Dengesiz PUSH

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



SS:bağıl konum · küçük adres yukarıda

AX = 1234h.

Adım 2 / 4

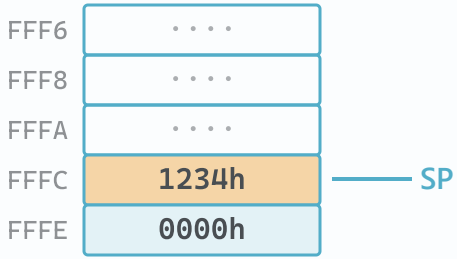
DEĞİŞKENLER

AX = 1234h | SP = FFFEh | IP = 0103h

Düzeltme: RET'ten önce POP AX eklenir ya da hiç PUSH yapılmaz. Her dalın (JMP/Jcc) PUSH/POP sayısı ayrı ayrı sayılır.

Hata Örneği: Dengesiz PUSH

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



SS:bağıl konum · küçük adres yukarıda

AX yığına kondu ama hiç POP edilmeyecek.

Adım 3 / 4

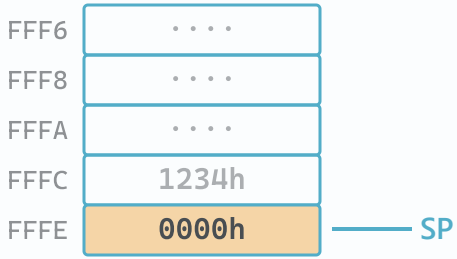
DEĞİŞKENLER

`AX = 1234h` `SP = FFFCh` `IP = 0104h`

Düzeltme: RET'ten önce POP AX eklenir ya da hiç PUSH yapılmaz. Her dalın (JMP/Jcc) PUSH/POP sayısı ayrı ayrı sayılır.

Hata Örneği: Dengesiz PUSH

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



SS:bağıl 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`

Düzeltme: RET'ten önce POP AX eklenir ya da hiç PUSH yapılmaz. Her dalın (JMP/Jcc) PUSH/POP sayısı ayrı ayrı sayılır.

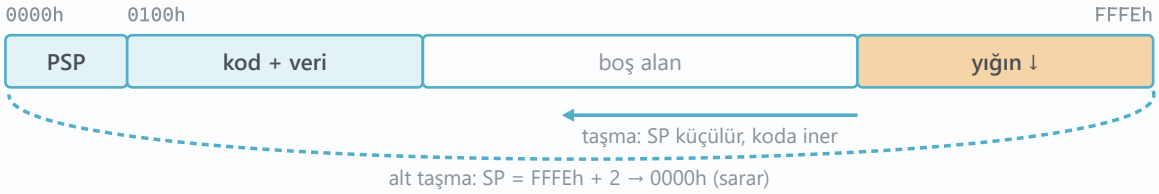
Yığın Taşması ve Alt Taşma

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



Kural: yığın alanının boyutu programın en derin çağrı zinciri + en çok PUSH sayısına göre seçilir. .EXE programlarında `.STACK 100h` gibi bir direktifle ayrı ve yeterli bir yığın kesimi ayrılır.

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 kullanılır; POP'lar bir kez yazılır.

3. SP'ye dokunmamak

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

4. BP çerçevesi

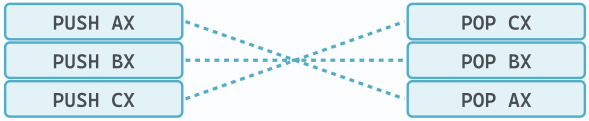
Yığındaki verilere [SP+n] ile erişilemez (8086'da SP taban yazmaç değildir); `MOV BP, SP` kullanılır.

5. Kesimleri korumak

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

6. emu8086 ile denetlemek

stack düğmesi yığın penceresini açar; RET'ten önce SP'nin girişteki değerine döndüğü denetlenir.



ayna görüntüsü

YIĞIN GÜVENLİĞİ

emu8086 ile Yığını İzlemek

1. Derle ve yükle

compile → emulate. Yazmaç panelinde CS, IP, SS, SP'nin başlangıç değerleri görülür (.COM: 0700h, 0100h, 0700h, FFFEh).

2. Tek adım

single step her basışta 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

Ortadaki 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, DOS'un DEBUG programından kalma bir gelenektir; program CX'e güvenmemelidir.

Laboratuvar: Yığın

PUSH AX / POP BX

KOD · **6 bayt**

```
1 ORG   100h
2 MOV   AX, 42      ; AX'e 4…
3 PUSH  AX          ; AX'i y…
4 POP   BX          ; yığınd…
5 RET               ; işleti…
6 END
```

YAZMAÇLAR

AX	0000	AH 00	AL 00		
BX	0000	BH 00	BL 00		
CX	0006	CH 00	CL 06		
DX	0000	DH 00	DL 00		
SI	0000	DI 0000	BP 0000		
SP	FFFE	CS 0700	IP 0100		
DS	0700	ES 0700	SS 0700		

BAYRAKLAR

O	D	I	T	S	Z	A	P	C
0	0	1	0	0	0	0	0	0

CS:IP = 0700:0100 · adım 0

BELLEK **DS = 0700h**

```
0100 B8 2A 00 50 5B C3 00 00
0108 00 00 00 00 00 00 00 00
0110 00 00 00 00 00 00 00 00
0118 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00
```

YIĞIN **SS:SP**

```
FFF6 0000
FFF8 0000
FFFA 0000
FFFC 0000
FFFE 0000 ◂ SP
```

ÇIKTI

Program yüklendi: 6 bayt, CS:IP = 0700:0100h. emu8086'daki gibi CX başlangıçta program boyutunu (0006h) gösterir. Adım düğmesiyle komutlar tek tek yürütülür; vurgulu satır sıradaki komuttur.

ÖZET

Özet

Kavram	Hatırlanacak nokta
Adres	Yığının üstü = SS:SP; fiziksel adres SS × 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.
PUSHF / POPF	Bayrak yazmacını saklar ve geri yükler; PUSH ve POP bayraklara dokunmaz.
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. SP = FFFEh iken `PUSH AX / PUSH BX / PUSHF / POP CX` yürütülüyor. SP kaç olur, CX'e ne gelir?
5. AX'i ve bayrakları bir işlem boyunca koruyan PUSH/POP dizisini yazın.

▶ **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 8.1: Yığın

Sonraki bölüm: Bölüm 8.2: Prosedürler · Terim sözlüğü