Bölüm 5.1: Kesmeler

Mikroişlemciler

Kesme kavramı | Kesme türleri | Vektör tablosu | INT ve IRET | İstisnalar: INT 0, 1, 3, 4 | Donanım kesmeleri | 8259A ve maskeleme | Kendi kesme yordamı

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Kavram

Kesmenin ne olduğunu, yoklamadan (polling) farkını ve kesme türlerini açıklamak.

Mekanizma

Kesme vektör tablosunu kullanarak `INT n` ile hangi adrese gidileceğini hesaplamak.

Yığın

`INT` ve `IRET` sırasında FLAGS, CS ve IP'nin yığına nasıl yazılıp geri alındığını izlemek.

Donanım kesmeleri

INTR kabulünü ve 8259A ile maskelemeyi izlemek; kendi kesme yordamını kurup eski vektörü geri yüklemek.

KESME KAVRAMI

Kesme Nedir?

- **Kesme (interrupt)**, işlemcinin o anki işini geçici olarak bırakıp özel bir yordamı çalıştırmasını sağlayan olaydır.
- Çalıştırılan yordama **kesme hizmet yordamı** denir: **ISR** (interrupt service routine).
- ISR bitince işlemci, kaldığı komuttan devam eder; ana program kesildiğini fark etmez.
- Yazılım kesmeleri, programlamayı kolaylaştıran **hazır hizmetlerdir**: karakter yazdırmak için kod yazmak yerine bir kesme çağrılır.
- Ekran, klavye, disk sürücüsü, bellek yönetimi gibi donanımlarla çalışan kesme işlevleri vardır.
- Kesmeler **donanım** tarafından da tetiklenebilir (klavyeye basılması, zamanlayıcı).

Günlük hayattan

Kitap okurken telefon çalar: sayfa işaretlenir (dönüş adresini saklama), telefon açılır (ISR), konuşma bitince işaretli sayfadan okumaya devam edilir (IRET).



KESME KAVRAMI

Yoklama mı, Kesme mi?



Yoklama

- Basit, ek donanım gerekmez.
- CPU zamanı boşa harcanır.
- Olay, sorgu aralığına bağlı olarak geç algılanır.

Kesme

+ CPU yalnızca olay olduğunda meşgul edilir.
+ Olaya hızlı tepki verilir.
+ Vektör tablosu, yığın ve ISR düzeni gerekir.

KESME KAVRAMI

Kesme Türleri

Yazılım kesmesi

Program içindeki `INT n` komutuyla bilerek çağrılır. BIOS ve DOS hizmetleri bu yolla kullanılır. `INT 3` (kesme noktası) ve `INTO` (OF=1 ise INT 4) de yazılım kesmesidir.

Donanım kesmesi

Dış aygıt bir pini sürer. **INTR** maskelenebilir (IF=0 iken dikkate alınmaz), **NMI** maskelenemez (ör. bellek eşlik hatası, güç kaybı).

İç kesme (istisna)

İşlemcinin kendisi üretir: sıfıra bölme / bölüm taşması → **INT 0**, TF=1 iken her komuttan sonra tek adım → **INT 1**.

Tip	Kaynak	Açıklama
`00h`	iç	Bölme hatası (`DIV` / `IDIV` sıfıra bölme veya bölüm sığmıyor)
`01h`	iç	Tek adım (TF = 1) — hata ayıklayıcılar kullanır
`02h`	donanım	NMI pini (maskelenemez)
`03h`	yazılım	Kesme noktası; tek baytlık `INT 3` (CCh)
`04h`	yazılım	`INTO` : OF = 1 ise taşma kesmesi
`05h–FFh`	—	Intel 05h–1Fh'yi ayırmıştır; PC'de BIOS, DOS ve kullanıcı programları kullanır

KESME KAVRAMI

Üç Kaynağın Karşılaştırması

	Yazılım (INT n)	Donanım (INTR / NMI)	İç (istisna)
Tetikleyen	programdaki komut	dış aygıt, pin	işlemcinin kendisi
Zamanlama	eş zamanlı: hep aynı yerde	eş zamansız: herhangi bir komuttan sonra	eş zamanlı: hatalı komutta
Tip numarası	komutun içinde	INTR: 8259A, INTA ile; NMI: hep 2	sabit: 0, 1
IF = 0 iken	yine çalışır	INTR bekler, NMI çalışır	yine çalışır
Tipik kullanım	BIOS/DOS hizmeti, INT 3, INTO	klavye, zamanlayıcı, disk	bölme hatası, tek adım

Ortak yol

Tip numarası bir kez belirlendikten sonra üçü de aynı adımları izler: FLAGS, CS, IP yığına; IF = TF = 0; CS:IP ← vektör.

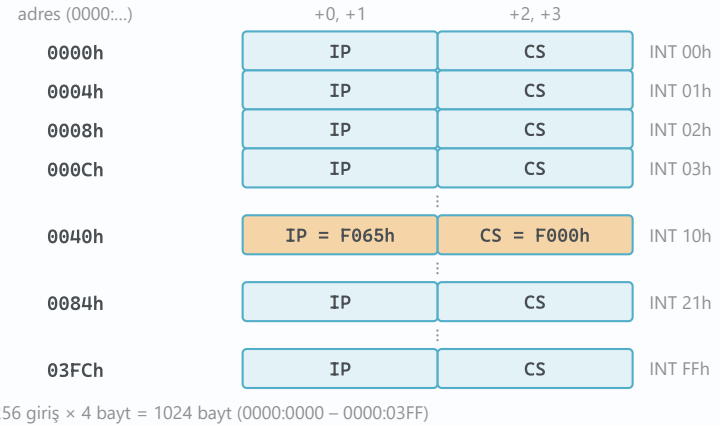
Fark nerede?

Yalnızca tip numarasının nereden geldiği ve kesmenin ne zaman oluştuğu farklıdır.

CLI yalnızca INTR girişini kapatır; yazılım kesmeleri ve istisnalar IF'den etkilenmez.

Kesme Vektör Tablosu (IVT)

- Belleğin en başındaki **1 KB**: `00000h–003FFh`.
- 256 kesme tipinin her biri için **4 baytlık** bir **vektör** (ISR'nin adresi) tutulur.
- Düşük adresli sözcük **IP** (bağıl konum), yüksek adresli sözcük **CS** (kesim) değeridir; ikisi de little-endian.
- Tip *n* için vektörün adresi: **n × 4**.
- Vektörü değiştirmek, kesmeyi başka bir yordama yönlendirmek demektir.



INT 10h vektörü için gösterilen F000:F065 değeri, orijinal IBM PC BIOS'undaki video yordamının adresidir (örnek).

KESME VEKTÖR TABLOSU

Vektör Adresi Adım Adım

INT 10h

kesme tipi n = 10h = 16

Program `INT 10h` komutunu çalıştırır. Tip numarası **n = 10h** (16).

Adım 1 / 4

KESME VEKTÖR TABLOSU

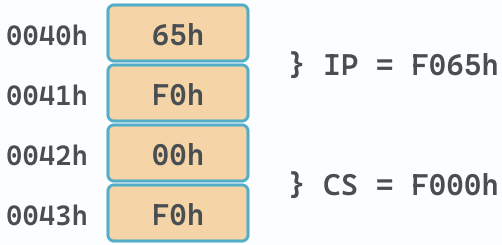
Vektör Adresi Adım Adım

$$10h \times 4 = 40h$$

16 × 4 = 64 → adres 0000:0040h

Vektör adresi **n × 4**: 10h × 4 = **40h**. (Onaltılıkta ×4, iki bit sola kaydırmaktır.)

Vektör Adresi Adım Adım



Bu 4 bayt okunur: düşük sözcük **IP = F065h**, yüksek sözcük **CS = F000h** (little-endian: önce düşük bayt).

KESME VEKTÖR TABLOSU

Vektör Adresi Adım Adım

fiziksel adres = CS × 16 + IP

`F0000h + F065h = FF065h`

Yığına dönüş bilgisi yazıldıktan sonra CS:IP = F000:F065 yüklenir; ISR, fiziksel **FF065h** adresinden çalışmaya başlar.

Demo: Vektör Hesaplayıcı

Bir kesme tipi (00h–FFh) girilince tablodaki konumu ve kesmenin bilinen görevi gösterilir.

INT 21 h **hesapla** **00h** **10h** **16h** **FFh**

Hesap	`21h × 4 = 0084h` (33 × 4 = 132)
IP	`0000:0084h` ve `0000:0085h`
CS	`0000:0086h` ve `0000:0087h`
Görev	DOS: genel hizmetler (AH ile seçilir)

`INT 21h` → vektör `0000:0084h`

Hesap her zaman aynıdır: `adres = n × 4`. IP bu adreste, CS ise `n × 4 + 2` adresindedir.

INT n Nasıl Çalışır?

1. FLAGS yazmacı yığına itilir: `SP ← SP − 2`, `[SS:SP] ← FLAGS`.

Adım 1 / 6

Yığına toplam **6 bayt** (3 sözcük) yazılır. IF'nin eski değeri FLAGS ile birlikte saklandığı için IRET onu da geri yükler.

INT n Nasıl Çalışır?

1. FLAGS yazmacı yığına itilir: `SP ← SP − 2`, `[SS:SP] ← FLAGS`.

2. IF = 0 ve **TF = 0** yapılır: ISR başka maskelenebilir kesmelerle ve tek adımla kesilmez.

Yığına toplam **6 bayt** (3 sözcük) yazılır. IF'nin eski değeri FLAGS ile birlikte saklandığı için IRET onu da geri yükler.

INT n Nasıl Çalışır?

1. FLAGS yazmacı yığına itilir: `SP ← SP − 2`, `[SS:SP] ← FLAGS`.
2. IF = 0 ve **TF = 0** yapılır: ISR başka maskelenebilir kesmelerle ve tek adımla kesilmez.
3. CS yığına itilir.

Yığına toplam **6 bayt** (3 sözcük) yazılır. IF'nin eski değeri FLAGS ile birlikte saklandığı için IRET onu da geri yükler.

INT n Nasıl Çalışır?

1. FLAGS yazmacı yığına itilir: `SP ← SP − 2`, `[SS:SP] ← FLAGS`.
2. **IF = 0** ve **TF = 0** yapılır: ISR başka maskelenebilir kesmelerle ve tek adımla kesilmez.
3. CS yığına itilir.
4. IP (INT'ten *sonraki* komutun adresi) yığına itilir.

Yığına toplam **6 bayt** (3 sözcük) yazılır. IF'nin eski değeri FLAGS ile birlikte saklandığı için IRET onu da geri yükler.

INT n Nasıl Çalışır?

1. FLAGS yazmacı yığına itilir: `SP ← SP − 2`, `[SS:SP] ← FLAGS`.
2. **IF = 0** ve **TF = 0** yapılır: ISR başka maskelenebilir kesmelerle ve tek adımla kesilmez.
3. CS yığına itilir.
4. IP (INT'ten *sonraki* komutun adresi) yığına itilir.
5. `IP ← [n×4]`, `CS ← [n×4+2]`: ISR başlar.

Yığına toplam **6 bayt** (3 sözcük) yazılır. IF'nin eski değeri FLAGS ile birlikte saklandığı için IRET onu da geri yükler.

INT VE IRET

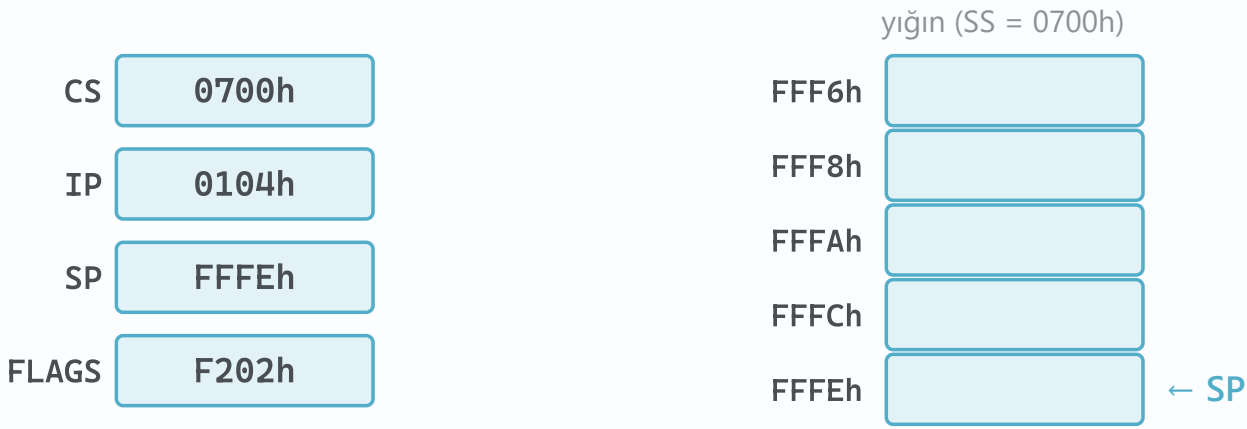
INT n Nasıl Çalışır?

1. FLAGS yazmacı yığına itilir: `SP ← SP − 2`, `[SS:SP] ← FLAGS`.
2. IF = 0 ve **TF = 0** yapılır: ISR başka maskelenebilir kesmelerle ve tek adımla kesilmez.
3. CS yığına itilir.
4. IP (INT'ten *sonraki* komutun adresi) yığına itilir.
5. `IP ← [n×4]`, `CS ← [n×4+2]` : ISR başlar.
6. ISR sonunda **IRET**: IP, CS ve FLAGS sırasıyla yığından geri alınır, program kaldığı yerden sürer.

Adım 6 / 6

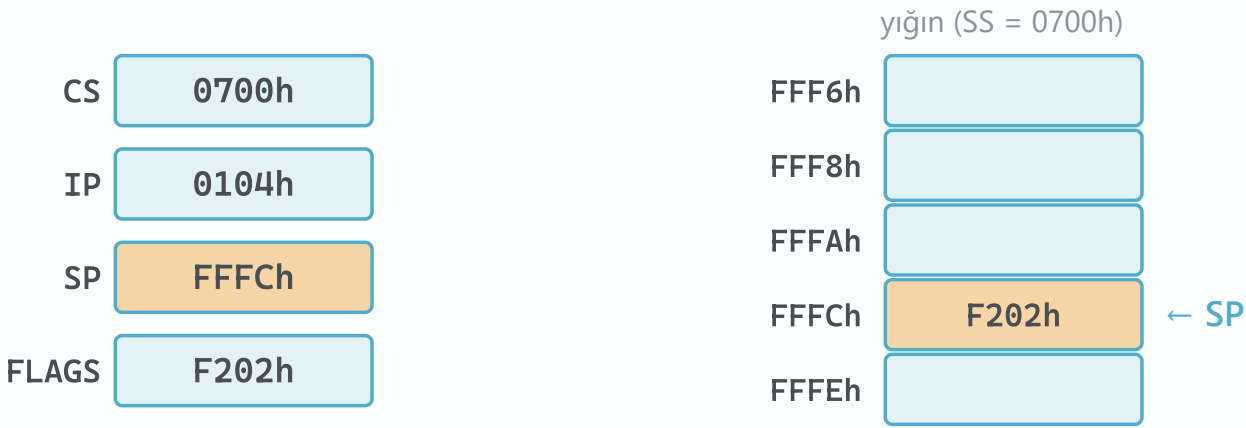
Yığına toplam **6 bayt** (3 sözcük) yazılır. IF'nin eski değeri FLAGS ile birlikte saklandığı için IRET onu da geri yükler.

INT ve IRET: Yığın Adım Adım



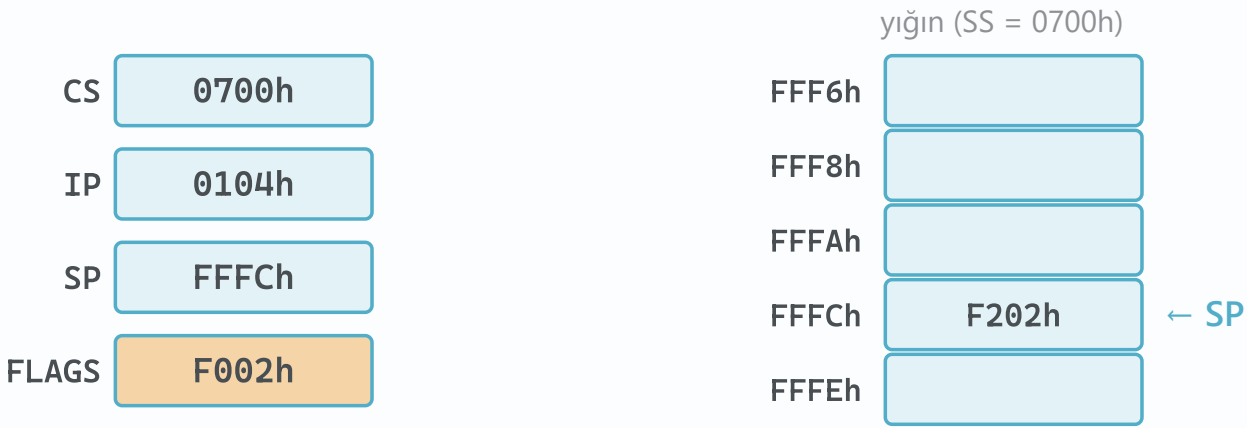
`INT 10h` 0700:0104'te. SP = FFFEh, FLAGS = F202h (IF = 1; 8086'da bit 1 ve 12–15. bitler hep 1'dir).

INT ve IRET: Yığın Adım Adım



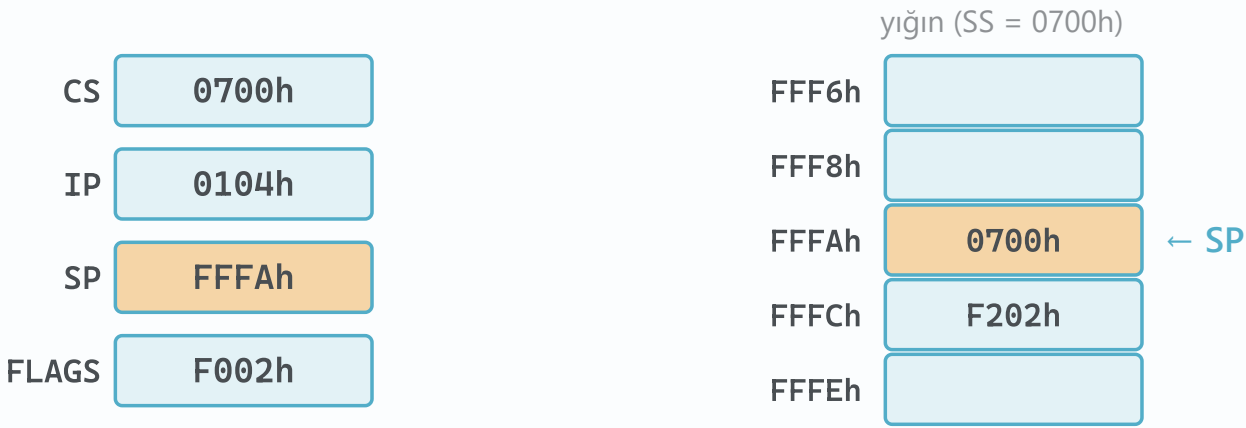
FLAGS itilir: SP = FFFCh, [FFFCh] = F202h.

INT ve IRET: Yığın Adım Adım



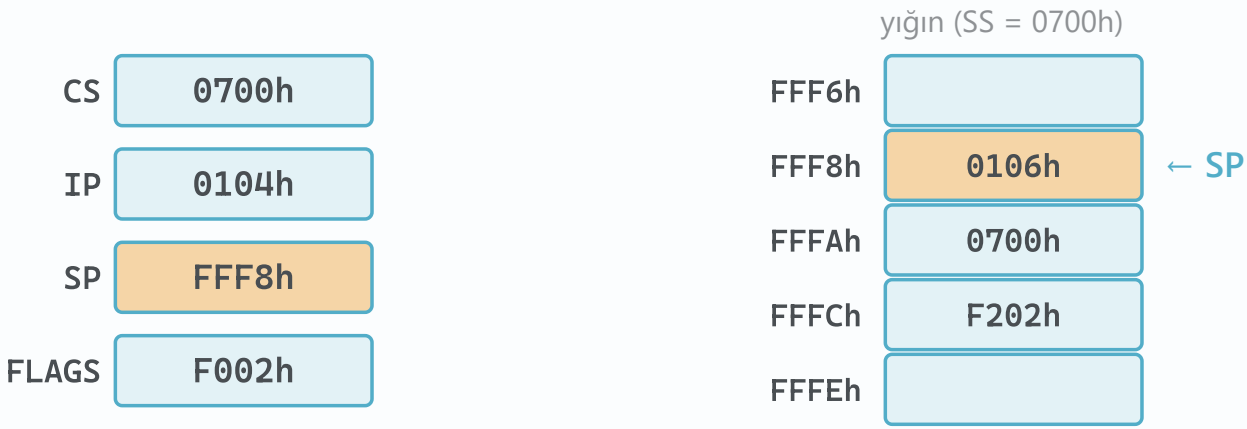
IF = 0, TF = 0 → FLAGS = F002h. Eski değer yığında güvende.

INT ve IRET: Yığın Adım Adım



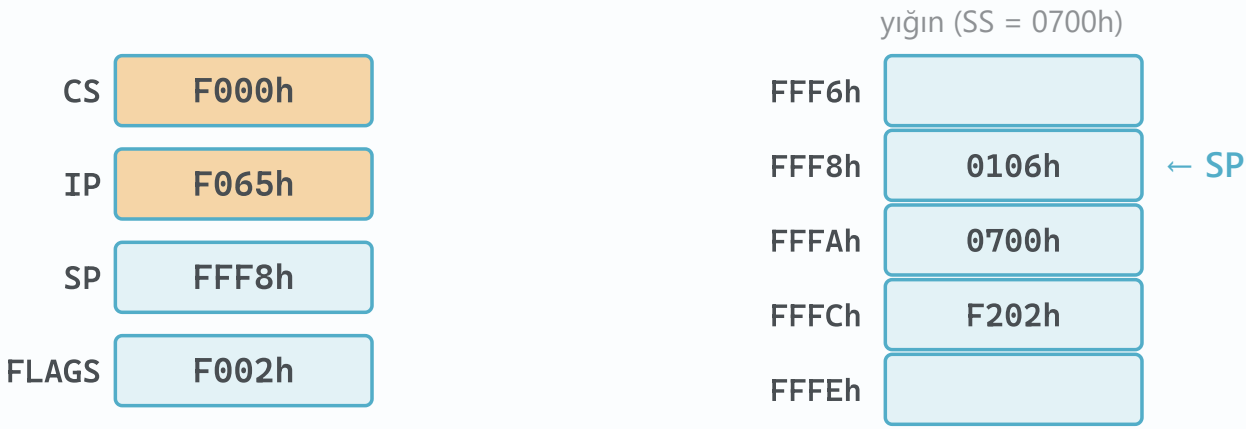
CS itilir: SP = FFFAh, [FFFAh] = 0700h.

INT ve IRET: Yığın Adım Adım



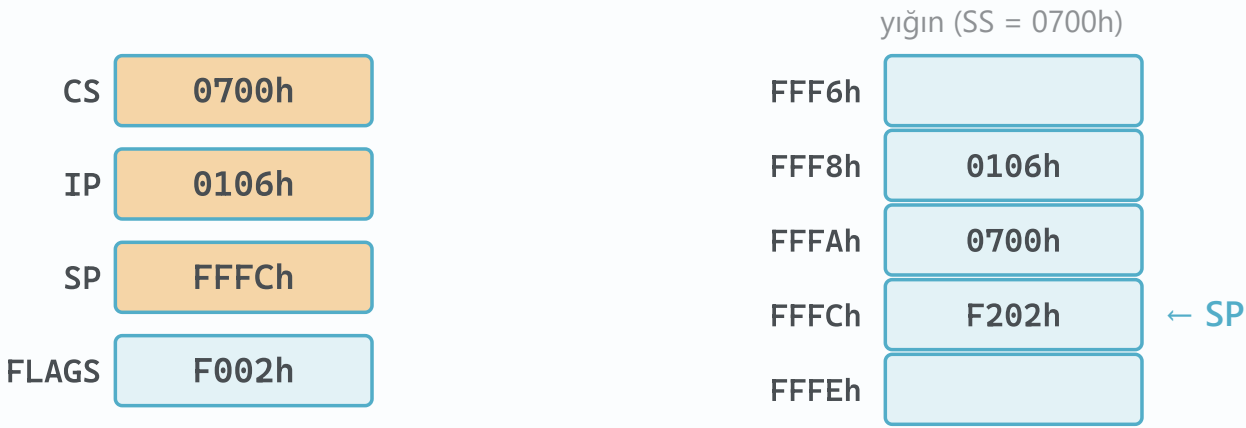
Dönüş adresi itilir: INT 10h 2 bayt olduğu için **0106h**. SP = FFF8h.

INT ve IRET: Yığın Adım Adım



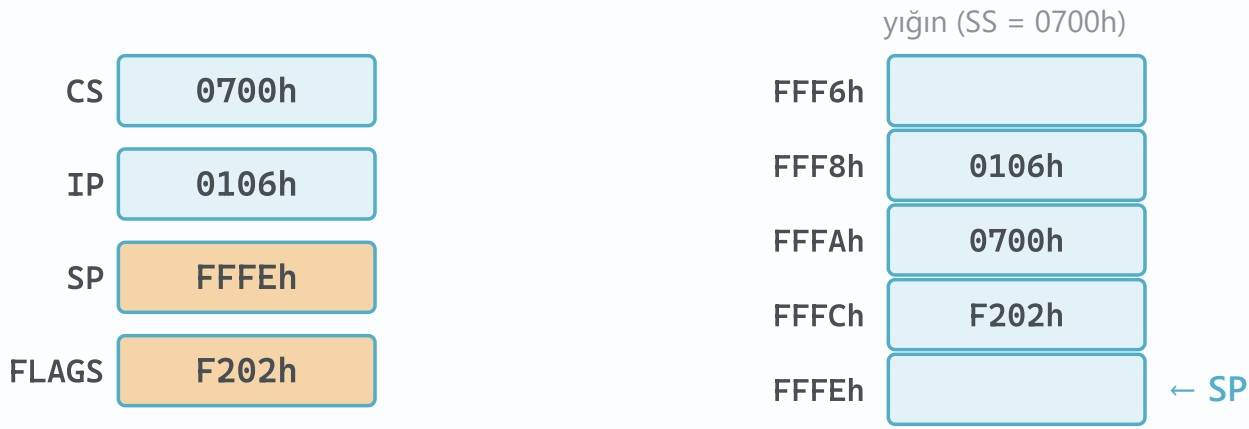
CS:IP ← vektör (F000:F065). ISR çalışır ve karakteri yazar.

INT ve IRET: Yığın Adım Adım



IRET: IP ← 0106h, CS ← 0700h (SP = FFFCh).

INT ve IRET: Yığın Adım Adım



FLAGS ← F202h, SP = FFFEh. Program 0106h'den devam eder. (Yığındaki eski değerler silinmez, yalnızca SP geri çekilir.)

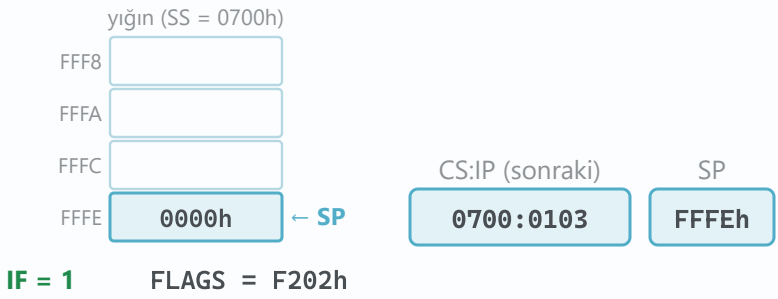
INT/IRET ile CALL/RET

	`CALL` / `RET`	`INT n` / `IRET`
Hedef adres	komutun içinde (etiket)	vektör tablosundan (n × 4)
Yığına yazılan	IP (yakın) veya CS + IP (uzak)	FLAGS + CS + IP (6 bayt)
Bayraklar	değişmez	IF = 0, TF = 0 yapılır
Dönüş	`RET` : IP (ve CS) geri alınır	`IRET` : IP, CS, FLAGS geri alınır
Komut boyu	3 bayt (yakın CALL)	2 bayt (`INT 3` : 1 bayt)
Kim çağırır?	yalnız program	program, işlemci (istisna) veya donanım

ISR'yi `RET` ile bitirmek hatadır: yakın RET yalnız IP'yi alır; CS ve FLAGS yığında kalır (SP 4 bayt kayar) ve IF eski hâline dönmez. ISR'ler daima **IRET** ile biter.

Kod İzleme: IRET Yerine RET

```
1 org 100h
▶ mov ax, 0
3 mov es, ax
4 mov word ptr es:[61h*4], offset isr
5 mov word ptr es:[61h*4+2], cs
6 int 61h
7 hlt                ; SP = ?
8 isr:
9 ret                ; HATA!
```



AX = 0: IVT kesimi.

Adım 1 / 7

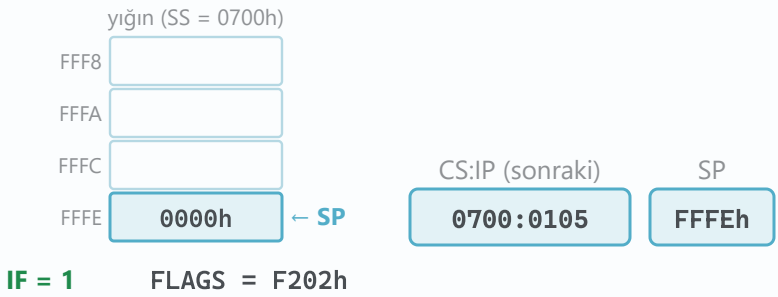
DEĞİŞKENLER

`CS = 0700h` `IP = 0103h` `SP = FFFEh` `IF = 1`

RETF bile yetmez: IP ve CS alınır ama FLAGS (2 bayt) yığında kalır ve IF eski hâline dönmez.

Kod İzleme: IRET Yerine RET

```
1 org 100h
2 mov ax, 0
▶ mov es, ax
4 mov word ptr es:[61h*4], offset isr
5 mov word ptr es:[61h*4+2], cs
6 int 61h
7 hlt                ; SP = ?
8 isr:
9 ret                ; HATA!
```



ES = 0000h.

Adım 2 / 7

DEĞİŞKENLER

`CS = 0700h` `IP = 0105h` `SP = FFFEh` `IF = 1`

RETF bile yetmez: IP ve CS alınır ama FLAGS (2 bayt) yığında kalır ve IF eski hâline dönmez.

Kod İzleme: IRET Yerine RET

```
1 org 100h
2 mov ax, 0
3 mov es, ax
▶ mov word ptr es:[61h*4], offset isr
5 mov word ptr es:[61h*4+2], cs
6 int 61h
7 hlt                ; SP = ?
8 isr:
9 ret                ; HATA!
```



IF = 1 FLAGS = F202h

Tip 61h vektörünün IP'si: [184h] = `offset isr` = 0114h.

Adım 3 / 7

DEĞİŞKENLER

`CS = 0700h` `IP = 010Ch` `SP = FFFEh` `IF = 1`

RETF bile yetmez: IP ve CS alınır ama FLAGS (2 bayt) yığında kalır ve IF eski hâline dönmez.

Kod İzleme: IRET Yerine RET

```
1 org 100h
2 mov ax, 0
3 mov es, ax
4 mov word ptr es:[61h*4], offset isr
▶ mov word ptr es:[61h*4+2], cs
6 int 61h
7 hlt                ; SP = ?
8 isr:
9 ret                ; HATA!
```



CS: [186h] = 0700h. Vektör 0700:0114.

Adım 4 / 7

DEĞİŞKENLER

`CS = 0700h` `IP = 0111h` `SP = FFFEh` `IF = 1`

RETF bile yetmez: IP ve CS alınır ama FLAGS (2 bayt) yığında kalır ve IF eski hâline dönmez.

Kod İzleme: IRET Yerine RET

```
1 org 100h
2 mov ax, 0
3 mov es, ax
4 mov word ptr es:[61h*4], offset isr
5 mov word ptr es:[61h*4+2], cs
▶ int 61h
7 hlt                ; SP = ?
8 isr:
9 ret                ; HATA!
```



IF = 0 FLAGS = F002h

`int 61h` : yığına FLAGS (F202h), CS (0700h), IP (0113h) — **6 bayt**; SP = FFF8h, IF = 0.

Adım 5 / 7

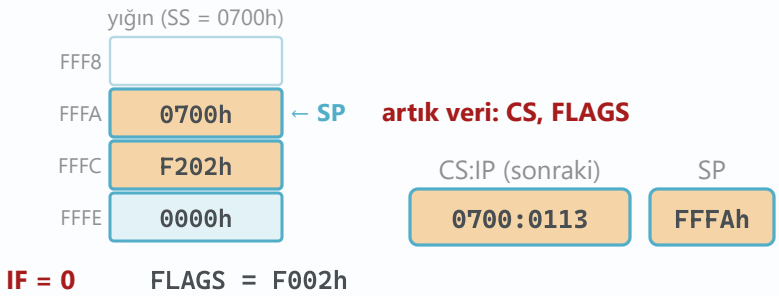
DEĞİŞKENLER

CS = 0700h | IP = 0114h | SP = FFF8h | IF = 0

RETF bile yetmez: IP ve CS alınır ama FLAGS (2 bayt) yığında kalır ve IF eski hâline dönmez.

Kod İzleme: IRET Yerine RET

```
1 org 100h
2 mov ax, 0
3 mov es, ax
4 mov word ptr es:[61h*4], offset isr
5 mov word ptr es:[61h*4+2], cs
6 int 61h
7 hlt                ; SP = ?
8 isr:
▶ ret                ; HATA!
```



ISR'deki yakın `ret` (IRET olmalıydı) yalnız **IP**'yi alır (0113h): CS ve FLAGS yığında kaldı. SP = **FFFAh**, beklenen FFFEh'den 4 bayt eksik; IF = 0 kaldı.

Adım 6 / 7

DEĞİŞKENLER

`CS = 0700h` `IP = 0113h` `SP = FFFAh` `IF = 0`

RETF bile yetmez: IP ve CS alınır ama FLAGS (2 bayt) yığında kalır ve IF eski hâline dönmez.

Kod İzleme: IRET Yerine RET

```
1 org 100h
2 mov ax, 0
3 mov es, ax
4 mov word ptr es:[61h*4], offset isr
5 mov word ptr es:[61h*4+2], cs
6 int 61h
▶ hlt                ; SP = ?
8 isr:
9 ret                ; HATA!
```



IF = 0 FLAGS = F002h

`hlt`'e gelindi ama kesmeler kapalı (IF = 0) ve yığın kaymış: bu noktadan sonra bir `ret` 0700h'yi dönüş adresi sanırdı. ISR daima **IRET** ile biter.

Adım 7 / 7

DEĞİŞKENLER

CS = 0700h | IP = 0114h | SP = FFFAh | IF = 0

RETF bile yetmez: IP ve CS alınır ama FLAGS (2 bayt) yığında kalır ve IF eski hâline dönmez.

İstisnalar: Bölme Hatası ve INTO

INT 0: bölme hatası

DIV ya da IDIV iki durumda INT 0 üretir: bölen 0 ise ya da bölüm hedefe sığmıyorsa. `AX = 1000`, `DIV BL` (BL = 2): bölüm 500, AL'ye (en çok 255) sığmaz → INT 0. Bölen sıfır olmasa da hata oluşur.

Dönüş adresi

8086 yığına DIV'den **sonraki** komutun adresini iter; IRET programı bölmeyi atlayarak sürdürür. 80286 ve sonrasında ise hatalı komutun kendi adresi itilir (komut yeniden denenebilir).

INTO

Tek baytlık komut (CEh). OF = 1 ise INT 4'ü çağırır, OF = 0 ise hiçbir şey yapmaz. İşaretli toplama ya da çıkarmadan hemen sonra yazılır: `ADD AL, BL` · `INTO`.

DOS ne yapar?

DOS, INT 0 vektörüne kendi yordamını koyar: "Divide overflow" yazıp programı sonlandırır. Program kendi yordamını kurarak bu davranışı değiştirebilir.

Bölmeden önce bölenin sıfır olmadığını ve bölümün sığacağını denetlemek (ör. 16 bitlik bölme için DX:AX kullanmak) INT 0'ı baştan önler.

Kod İzleme: Kendi INT 0 Yordamımız

```
1  org 100h
▶  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  mov word ptr es:[0*4], offset hata
5  mov word ptr es:[0*4+2], cs
6  mov ax, 100
7  mov bl, 0
8  div bl                ; INT 0!
9  mov dl, al            ; dönüş buraya
10 ret
11 hata:
12 mov ax, 0FFFFh        ; hata işareti
13 iret
```



AX = 0: IVT kesimi.

Adım 1 / 11

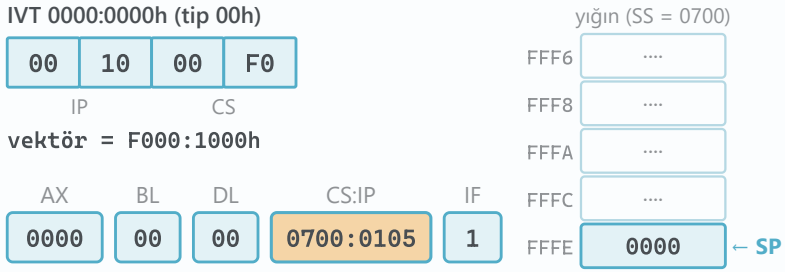
DEĞİŞKENLER

AX = **0000h** BX = **0000h** DX = **0000h** SP = **FFFEh** IF = **1**

Kısalık için eski vektör saklanmadı; gerçek bir programda çıkıştan önce geri yüklenir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: Kendi INT 0 Yordamımız

```
1  org 100h
2  mov ax, 0
▶  mov es, ax            ; ES = 0: IVT
4  mov word ptr es:[0*4], offset hata
5  mov word ptr es:[0*4+2], cs
6  mov ax, 100
7  mov bl, 0
8  div bl                ; INT 0!
9  mov dl, al            ; dönüş buraya
10 ret
11 hata:
12 mov ax, 0FFFFh        ; hata işareti
13 iret
```



ES = 0000h: ES:[0] tip 0'ın vektörüdür (0 × 4 = 0).

Adım 2 / 11

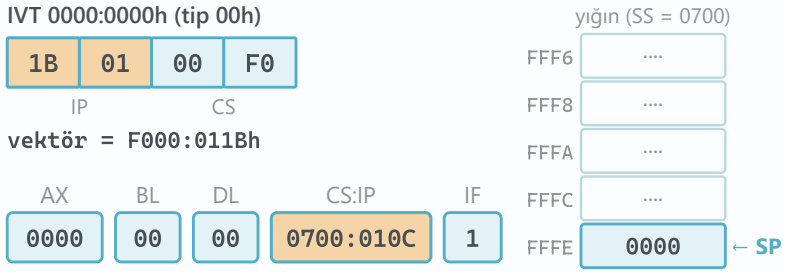
DEĞİŞKENLER

AX = **0000h** BX = **0000h** DX = **0000h** SP = **FFFEh** IF = **1**

Kısalık için eski vektör saklanmadı; gerçek bir programda çıkıştan önce geri yüklenir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: Kendi INT 0 Yordamımız

```
1  org 100h
2  mov ax, 0
3  mov es, ax              ; ES = 0: IVT
▶  mov word ptr es:[0*4], offset hata
5  mov word ptr es:[0*4+2], cs
6  mov ax, 100
7  mov bl, 0
8  div bl                  ; INT 0!
9  mov dl, al              ; dönüş buraya
10 ret
11 hata:
12 mov ax, 0FFFFh          ; hata işareti
13 iret
```



Düşük sözcük (IP) = `offset hata` = **011Bh**. Önceki vektör öykünücünün varsayılan yordamını gösteriyordu.

Adım 3 / 11

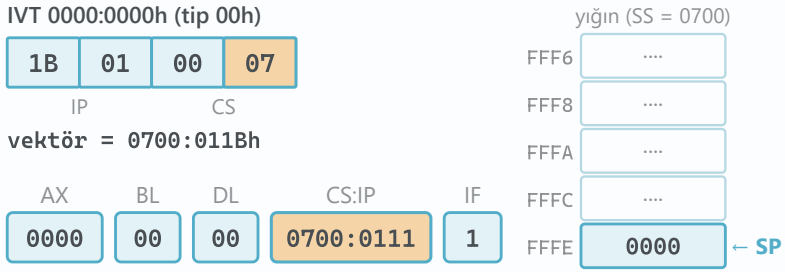
DEĞİŞKENLER

AX = **0000h** · BX = **0000h** · DX = **0000h** · SP = **FFFEh** · IF = **1**

Kısalık için eski vektör saklanmadı; gerçek bir programda çıkıştan önce geri yüklenir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: Kendi INT 0 Yordamımız

```
1  org 100h
2  mov ax, 0
3  mov es, ax              ; ES = 0: IVT
4  mov word ptr es:[0*4], offset hata
▶  mov word ptr es:[0*4+2], cs
6  mov ax, 100
7  mov bl, 0
8  div bl                  ; INT 0!
9  mov dl, al              ; dönüş buraya
10 ret
11 hata:
12 mov ax, 0FFFFh          ; hata işareti
13 iret
```



Yüksek sözcük (CS) = 0700h. Tip 0 artık bizim yordamımıza gider.

Adım 4 / 11

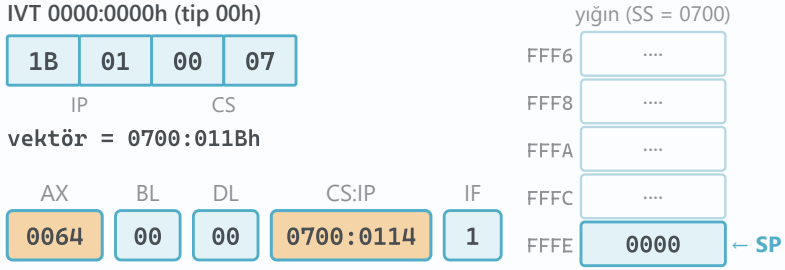
DEĞİŞKENLER

AX = **0000h** BX = **0000h** DX = **0000h** SP = **FFFEh** IF = **1**

Kısalık için eski vektör saklanmadı; gerçek bir programda çıkıştan önce geri yüklenir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: Kendi INT 0 Yordamımız

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  mov word ptr es:[0*4], offset hata
5  mov word ptr es:[0*4+2], cs
▶  mov ax, 100
7  mov bl, 0
8  div bl                ; INT 0!
9  mov dl, al            ; dönüş buraya
10 ret
11 hata:
12 mov ax, 0FFFFh        ; hata işareti
13 iret
```



AX = 100 = 0064h: bölünen.

Adım 5 / 11

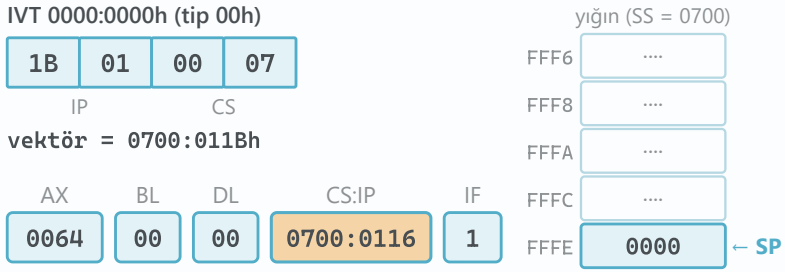
DEĞİŞKENLER

AX = 0064h | BX = 0000h | DX = 0000h | SP = FFFEh | IF = 1

Kısalık için eski vektör saklanmadı; gerçek bir programda çıkıştan önce geri yüklenir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: Kendi INT 0 Yordamımız

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  mov word ptr es:[0*4], offset hata
5  mov word ptr es:[0*4+2], cs
6  mov ax, 100
▶  mov bl, 0
8  div bl                ; INT 0!
9  mov dl, al            ; dönüş buraya
10 ret
11 hata:
12 mov ax, 0FFFFh        ; hata işareti
13 iret
```



BL = 0: bölen sıfır.

Adım 6 / 11

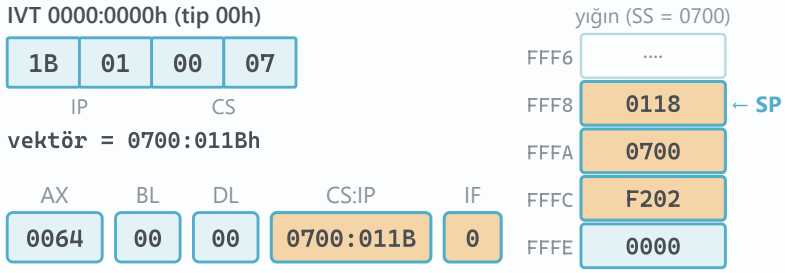
DEĞİŞKENLER

AX = **0064h** BX = **0000h** DX = **0000h** SP = **FFFEh** IF = **1**

Kısalık için eski vektör saklanmadı; gerçek bir programda çıkıştan önce geri yüklenir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: Kendi INT 0 Yordamımız

```
1  org 100h
2  mov ax, 0
3  mov es, ax              ; ES = 0: IVT
4  mov word ptr es:[0*4], offset hata
5  mov word ptr es:[0*4+2], cs
6  mov ax, 100
7  mov bl, 0
▶ div bl                   ; INT 0!
9  mov dl, al              ; dönüş buraya
10 ret
11 hata:
12 mov ax, 0FFFFh          ; hata işareti
13 iret
```



`div bl` → **INT 0**. Yığına FLAGS, CS ve dönüş IP'si **0118h** (DIV'den sonraki komut) itilir; SP = FFF8h, IF = 0.

Adım 7 / 11

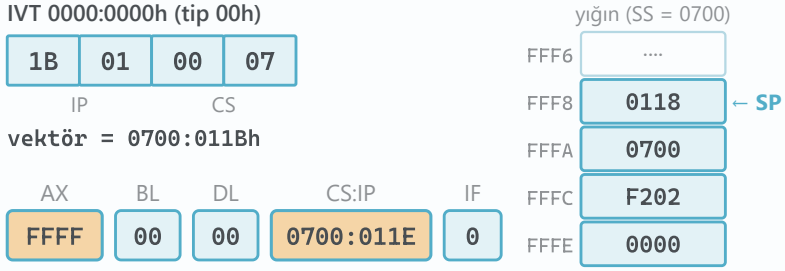
DEĞİŞKENLER

AX = **0064h** · BX = **0000h** · DX = **0000h** · SP = **FFF8h** · IF = **0**

Kısalık için eski vektör saklanmadı; gerçek bir programda çıkıştan önce geri yüklenir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: Kendi INT 0 Yordamımız

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  mov word ptr es:[0*4], offset hata
5  mov word ptr es:[0*4+2], cs
6  mov ax, 100
7  mov bl, 0
8  div bl                ; INT 0!
9  mov dl, al            ; dönüş buraya
10 ret
11 hata:
▶ mov ax, 0FFFFh        ; hata işareti
13 iret
```



Yordam çalışıyor: AX = **FFFFh**, programa "sonuç geçersiz" işareti.

Adım 8 / 11

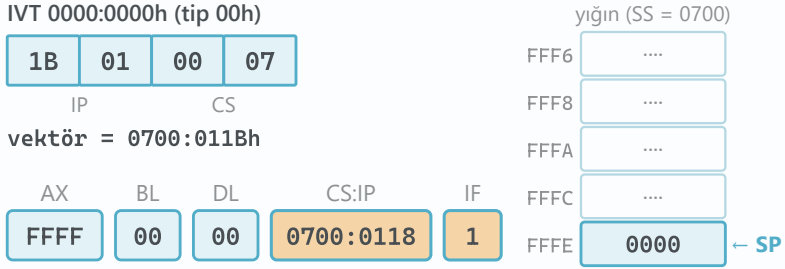
DEĞİŞKENLER

AX = **FFFFh** | BX = **0000h** | DX = **0000h** | SP = **FFF8h** | IF = **0**

Kısalık için eski vektör saklanmadı; gerçek bir programda çıkıştan önce geri yüklenir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: Kendi INT 0 Yordamımız

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  mov word ptr es:[0*4], offset hata
5  mov word ptr es:[0*4+2], cs
6  mov ax, 100
7  mov bl, 0
8  div bl                ; INT 0!
9  mov dl, al            ; dönüş buraya
10 ret
11 hata:
12 mov ax, 0FFFFh        ; hata işareti
▶  iret
```



`iret` : IP = 0118h, CS, FLAGS geri alınır; SP = FFFEh, IF = 1. DIV yeniden çalışmaz.

Adım 9 / 11

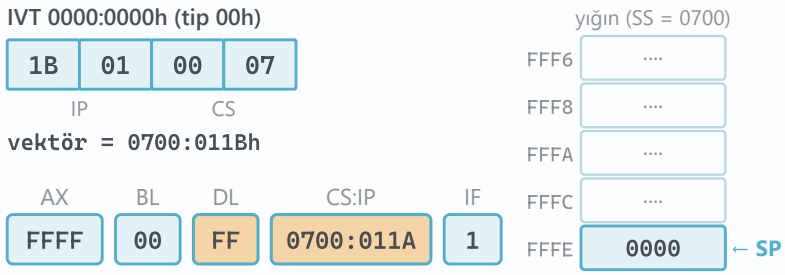
DEĞİŞKENLER

AX = **FFFFh** · BX = **0000h** · DX = **0000h** · SP = **FFFEh** · IF = **1**

Kısalık için eski vektör saklanmadı; gerçek bir programda çıkıştan önce geri yüklenir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: Kendi INT 0 Yordamımız

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  mov word ptr es:[0*4], offset hata
5  mov word ptr es:[0*4+2], cs
6  mov ax, 100
7  mov bl, 0
8  div bl                ; INT 0!
▶  mov dl, al            ; dönüş buraya
10 ret
11 hata:
12 mov ax, 0FFFFh        ; hata işareti
13 iret
```



DL = AL = **FFh**: ana program hatayı işaretten anlar.

Adım 10 / 11

DEĞİŞKENLER

AX = **FFFFh** | BX = **0000h** | DX = **00FFh** | SP = **FFFEh** | IF = **1**

Kısalık için eski vektör saklanmadı; gerçek bir programda çıkıştan önce geri yüklenir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: Kendi INT 0 Yordamımız

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  mov word ptr es:[0*4], offset hata
5  mov word ptr es:[0*4+2], cs
6  mov ax, 100
7  mov bl, 0
8  div bl                ; INT 0!
9  mov dl, al            ; dönüş buraya
▶ ret
11 hata:
12 mov ax, 0FFFFh        ; hata işareti
13 iret
```

IVT 0000:0000h (tip 00h)

1B	01	00	07
IP		CS	

vektör = 0700:011Bh

AX	BL	DL	CS:IP	IF
FFFF	00	FF	0700:0000	1

yığın (SS = 0700)

FFF6	....
FFF8	....
FFFA	....
FFFC	....
FFFE	....

`ret` : program DOS'a döner.

Adım 11 / 11

DEĞİŞKENLER

AX = **FFFFh** · BX = **0000h** · DX = **00FFh** · SP = **0000h** · IF = **1**

Kısalık için eski vektör saklanmadı; gerçek bir programda çıkıştan önce geri yüklenir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Hata Ayıklayıcının Araçları: INT 1 ve INT 3

Tek adım (TF): TF = 1 iken her komuttan sonra işlemci INT 1 üretir. TF'yi doğrudan değiştiren komut yoktur; FLAGS yığın üzerinden düzenlenir:

```
PUSHF
POP  AX
OR   AH, 01h      ; bit 8 = TF
PUSH AX
POPF              ; TF = 1
```

INT 1 yordamı TF = 0 ile çalışır (INT, TF'yi temizler); kendi komutları adım adım kesilmez. IRET, yığındaki TF = 1'i geri yükler.

Kesme noktası (INT 3): tek baytlık özel kodlaması vardır: **CCh**. Hata ayıklayıcı:

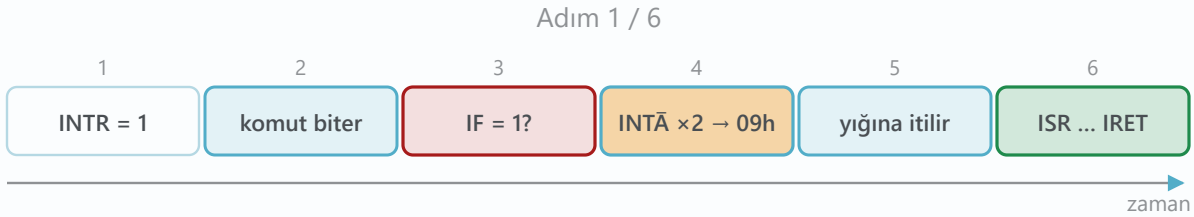
1. Durulacak komutun ilk baytını saklar, yerine CCh yazar.
2. Program oraya gelince INT 3 çalışır; yazmaçlar gösterilir.
3. Saklanan bayt geri yazılır, yığındaki dönüş IP'si 1 azaltılır.
4. IRET ile komut baştan çalıştırılır.

Tek bayt olduğu için en kısa komutun yerine bile sığar; `INT 03h` yazılsa iki baytlık CD 03 olurdu.

emu8086'nın "single step" düğmesi aynı işi öykünücü içinde yapar; gerçek işlemcide DEBUG gibi araçlar INT 1 ve INT 3 vektörlerini kullanır.

INTR Kabul Süreci

1. Aygıt (ör. klavye) IRQ1 hattını etkinleştirir; 8259A **INTR** pinini 1 yapar.

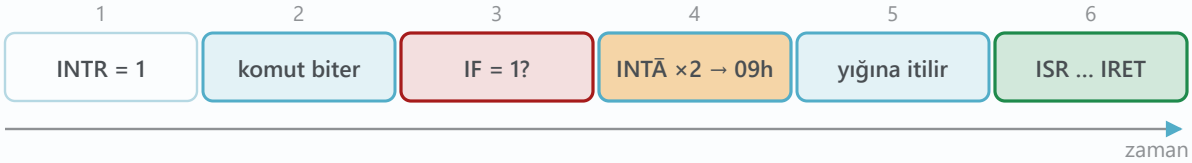


INTR Kabul Süreci

1. Aygıt (ör. klavye) IRQ1 hattını etkinleştirir; 8259A **INTR** pinini 1 yapar.

2. 8086 o anki komutu **bitirir**; kesmeler komut ortasında kabul edilmez.

Adım 2 / 6

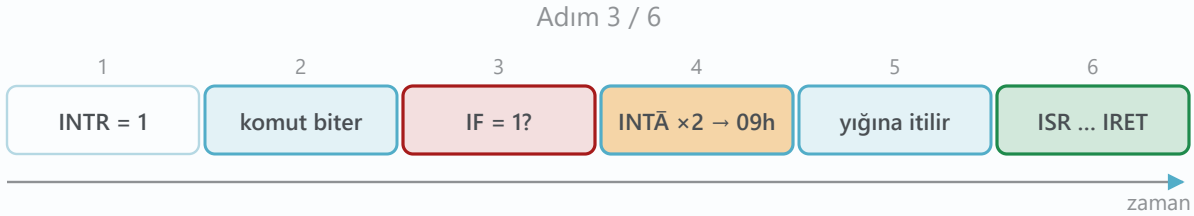


INTR Kabul Süreci

1. Aygıt (ör. klavye) IRQ1 hattını etkinleştirir; 8259A **INTR** pinini 1 yapar.

2. 8086 o anki komutu **bitirir**; kesmeler komut ortasında kabul edilmez.

3. IF = 0 ise istek bekletilir. IF = 1 ise kabul edilir.



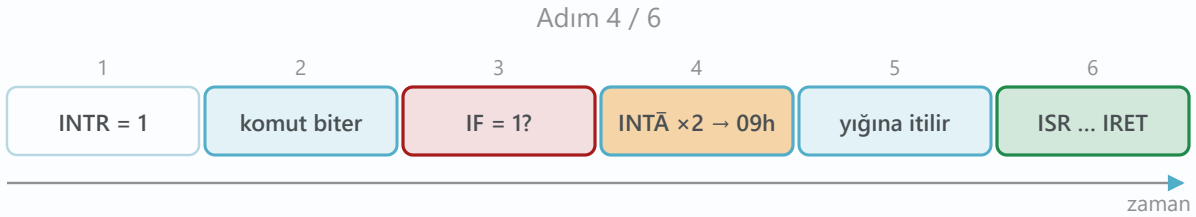
INTR Kabul Süreci

1. Aygıt (ör. klavye) IRQ1 hattını etkinleştirir; 8259A **INTR** pinini 1 yapar.

2. 8086 o anki komutu **bitirir**; kesmeler komut ortasında kabul edilmez.

3. IF = 0 ise istek bekletilir. IF = 1 ise kabul edilir.

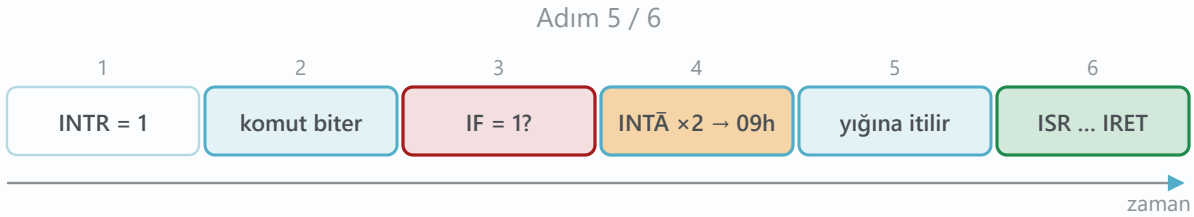
4. CPU iki **INTĀ** darbesi üretir; ikinci darbede 8259A tip numarasını (PC'de klavye için `09h`) veri yoluna koyar.



DONANIM KESMELERİ

INTR Kabul Süreci

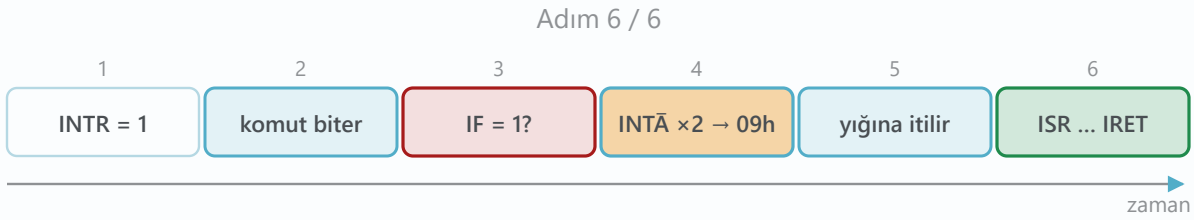
1. Aygıt (ör. klavye) IRQ1 hattını etkinleştirir; 8259A **INTR** pinini 1 yapar.
2. 8086 o anki komutu **bitirir**; kesmeler komut ortasında kabul edilmez.
3. IF = 0 ise istek bekletilir. IF = 1 ise kabul edilir.
4. CPU iki **INTĀ** darbesi üretir; ikinci darbede 8259A tip numarasını (PC'de klavye için `09h`) veri yoluna koyar.
5. Bundan sonrası `INT n` ile aynıdır: FLAGS, CS, IP yığına; IF = TF = 0; CS:IP ← [n×4].



DONANIM KESMELERİ

INTR Kabul Süreci

1. Aygıt (ör. klavye) IRQ1 hattını etkinleştirir; 8259A **INTR** pinini 1 yapar.
2. 8086 o anki komutu **bitirir**; kesmeler komut ortasında kabul edilmez.
3. IF = 0 ise istek bekletilir. IF = 1 ise kabul edilir.
4. CPU iki **INTĀ** darbesi üretir; ikinci darbede 8259A tip numarasını (PC'de klavye için `09h`) veri yoluna koyar.
5. Bundan sonrası `INT n` ile aynıdır: FLAGS, CS, IP yığına; IF = TF = 0; CS:IP ← [n×4].
6. ISR işini bitirince 8259A'ya **EOI** (kesme sonu) gönderir ve `IRET` ile döner.



Öncelik ve PC Kesme Haritası

8086 öncelik sırası

Sıra	Kaynak
1 (en yüksek)	Bölme hatası, `INT n`, `INTO`
2	NMI
3	INTR
4 (en düşük)	Tek adım (TF)

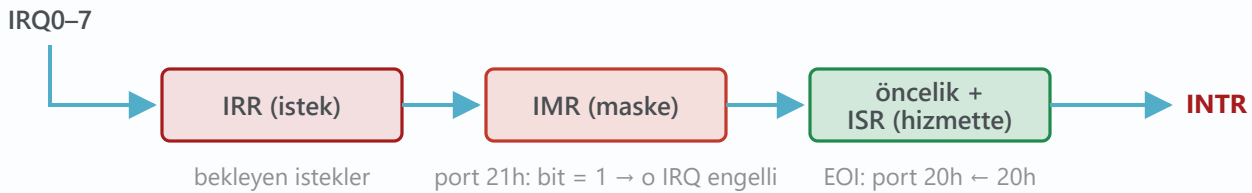
Aynı anda birden çok istek varsa, yüksek öncelikli olan önce hizmet görür.

IBM PC donanım kesmeleri

IRQ	Aygıt	Tip
0	Zamanlayıcı (18,2 Hz)	`08h`
1	Klavye	`09h`
3 / 4	Seri portlar	`0Bh / 0Ch`
6	Disket	`0Eh`
7	Yazıcı	`0Fh`

Klavyeye basılınca önce donanım kesmesi `INT 09h` çalışır ve tuşu arabelleğe koyar; programın çağırdığı `INT 16h` bu arabellekten okur.

8259A İçinde Üç Yazmaç



IRR

Gelen ama henüz hizmet görmeyen istekleri tutar.

IMR

Kesme maske yazmacı: her IRQ için bir bit. IF tüm INTR'yi, IMR tek tek hatları kapatır.

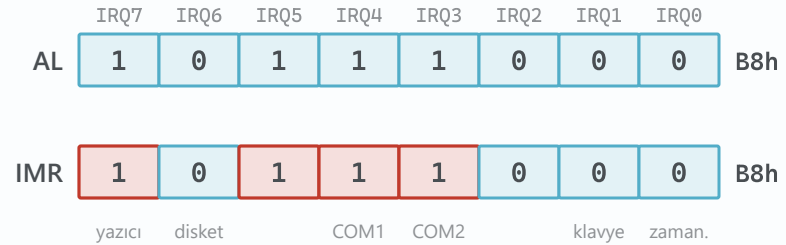
ISR (yazmaç)

O an hizmet gören IRQ'yu işaretler; EOI gelene kadar daha düşük öncelikliler bekler.

Bir ana 8259A'ya sekiz yardımcı bağlanarak 64 IRQ'ya çıkılır; PC/AT ikinci yongayı IRQ2'ye bağlar (IRQ8–15, tip 70h–77h). Yazmaç adı ISR, kesme hizmet yordamı ile karıştırılmamalıdır.

Kod İzleme: Tek Bir IRQ'yu Maskelemek

```
1 org 100h
▶ in al, 21h            ; IMR'yi oku
3 or al, 00000010b      ; IRQ1 maskele
4 out 21h, al
5 mov cx, 3             ; kritik iş
6 in al, 21h
7 and al, 11111101b     ; IRQ1 aç
8 out 21h, al
9 ret
```



kırmızı = maskeli (1) · port 21h = IMR

`in al, 21h` : IMR okunur, AL = **B8h** = 1011 1000b. IRQ3, 4, 5, 7 zaten maskeli (örnek değer).

Adım 1 / 8

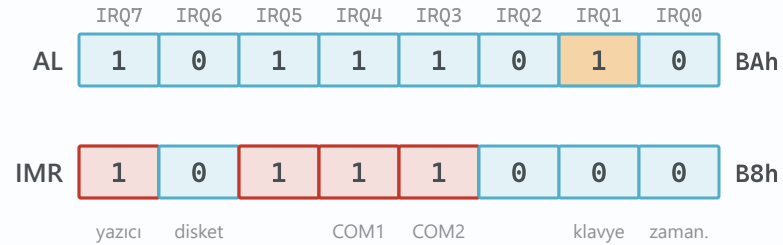
DEĞİŞKENLER

`AL = B8h` `CX = 0000h` `IMR = B8h`

Oku–değiştir–yaz kalıbı: OR ile maskelenir, AND ile açılır, böylece diğer bitler korunur. CLI'den farkı: yalnızca seçilen aygıt bekletilir.

Kod İzleme: Tek Bir IRQ'yu Maskelemek

```
1 org 100h
2 in al, 21h            ; IMR'yi oku
▶ or al, 00000010b      ; IRQ1 maskele
4 out 21h, al
5 mov cx, 3             ; kritik iş
6 in al, 21h
7 and al, 11111101b     ; IRQ1 aç
8 out 21h, al
9 ret
```



kırmızı = maskeli (1) · port 21h = IMR

`or al, 02h` : yalnızca bit 1 kurulur → AL = **BAh**. Diğer bitler korunur; IMR'ye henüz yazılmadı.

Adım 2 / 8

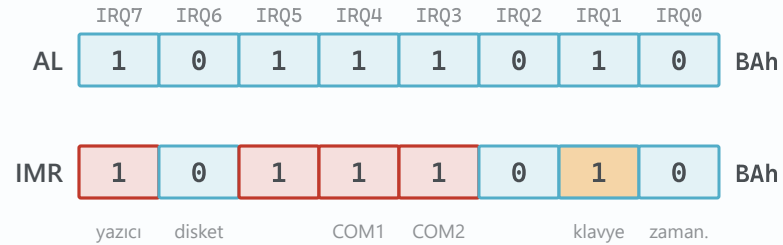
DEĞİŞKENLER

`AL = BAh` `CX = 0000h` `IMR = B8h`

Oku–değiştir–yaz kalıbı: OR ile maskelenir, AND ile açılır, böylece diğer bitler korunur. CLI'den farkı: yalnızca seçilen aygıt bekletilir.

Kod İzleme: Tek Bir IRQ'yu Maskelemek

```
1 org 100h
2 in al, 21h            ; IMR'yi oku
3 or al, 00000010b      ; IRQ1 maskele
▶ out 21h, al
5 mov cx, 3             ; kritik iş
6 in al, 21h
7 and al, 11111101b     ; IRQ1 aç
8 out 21h, al
9 ret
```



kırmızı = maskeli (1) · port 21h = IMR

`out 21h, al` : IMR = BAh. **IRQ1 (klavye)** artık INTR'ye ulaşamaz; IF ve diğer IRQ'lar etkilenmedi.

Adım 3 / 8

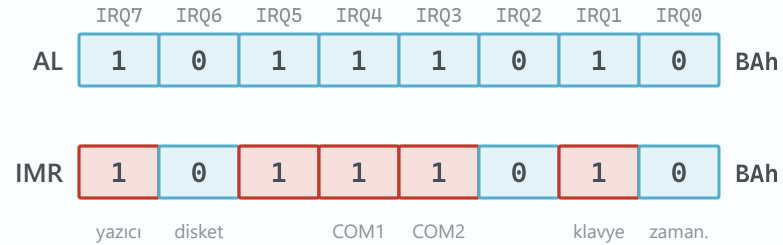
DEĞİŞKENLER

`AL = BAh` `CX = 0000h` `IMR = BAh`

Oku–değiştir–yaz kalıbı: OR ile maskelenir, AND ile açılır, böylece diğer bitler korunur. CLI'den farkı: yalnızca seçilen aygıt bekletilir.

Kod İzleme: Tek Bir IRQ'yu Maskelemek

```
1 org 100h
2 in al, 21h            ; IMR'yi oku
3 or al, 00000010b      ; IRQ1 maskele
4 out 21h, al
▶ mov cx, 3             ; kritik iş
6 in al, 21h
7 and al, 11111101b     ; IRQ1 aç
8 out 21h, al
9 ret
```



kırmızı = maskeli (1) · port 21h = IMR

Kesilmemesi gereken iş (burada yalnızca bir yer tutucu). Klavyeye basılırsa istek IRR'de bekler, kaybolmaz.

Adım 4 / 8

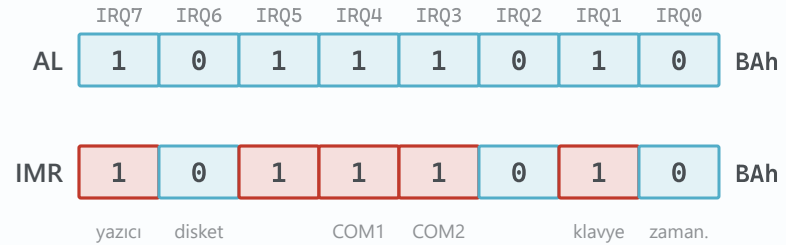
DEĞİŞKENLER

`AL = BAh` `CX = 0003h` `IMR = BAh`

Oku–değiştir–yaz kalıbı: OR ile maskelenir, AND ile açılır, böylece diğer bitler korunur. CLI'den farkı: yalnızca seçilen aygıt bekletilir.

Kod İzleme: Tek Bir IRQ'yu Maskelemek

```
1 org 100h
2 in al, 21h            ; IMR'yi oku
3 or al, 00000010b      ; IRQ1 maskele
4 out 21h, al
5 mov cx, 3             ; kritik iş
▶ in al, 21h
7 and al, 11111101b     ; IRQ1 aç
8 out 21h, al
9 ret
```



kırmızı = maskeli (1) · port 21h = IMR

IMR yeniden okunur: AL = BAh.

Adım 5 / 8

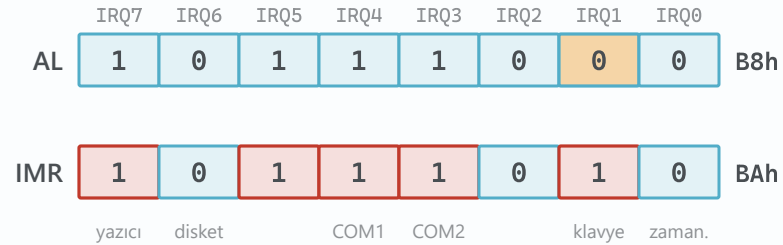
DEĞİŞKENLER

`AL = BAh` `CX = 0003h` `IMR = BAh`

Oku–değiştir–yaz kalıbı: OR ile maskelenir, AND ile açılır, böylece diğer bitler korunur. CLI'den farkı: yalnızca seçilen aygıt bekletilir.

Kod İzleme: Tek Bir IRQ'yu Maskelemek

```
1 org 100h
2 in al, 21h            ; IMR'yi oku
3 or al, 00000010b      ; IRQ1 maskele
4 out 21h, al
5 mov cx, 3             ; kritik iş
6 in al, 21h
▶ and al, 11111101b     ; IRQ1 aç
8 out 21h, al
9 ret
```



kırmızı = maskeli (1) · port 21h = IMR

`and al, 0FDh` (1111 1101b): yalnızca bit 1 silinir → AL = **B8h**.

Adım 6 / 8

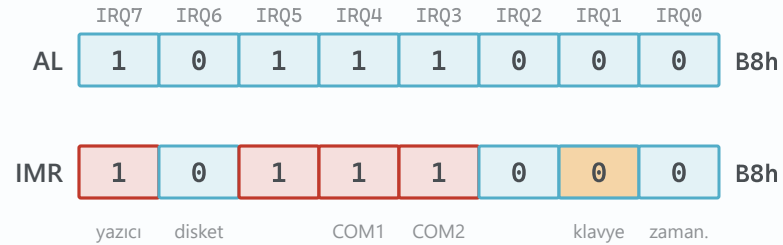
DEĞİŞKENLER

`AL = B8h` `CX = 0003h` `IMR = BAh`

Oku–değiştir–yaz kalıbı: OR ile maskelenir, AND ile açılır, böylece diğer bitler korunur. CLI'den farkı: yalnızca seçilen aygıt bekletilir.

Kod İzleme: Tek Bir IRQ'yu Maskelemek

```
1 org 100h
2 in al, 21h            ; IMR'yi oku
3 or al, 00000010b      ; IRQ1 maskele
4 out 21h, al
5 mov cx, 3             ; kritik iş
6 in al, 21h
7 and al, 11111101b     ; IRQ1 aç
▶ out 21h, al
9 ret
```



kırmızı = maskeli (1) · port 21h = IMR

`out 21h, al`: IMR = B8h, IRQ1 açıldı; bekleyen klavye isteği şimdi hizmet görür.

Adım 7 / 8

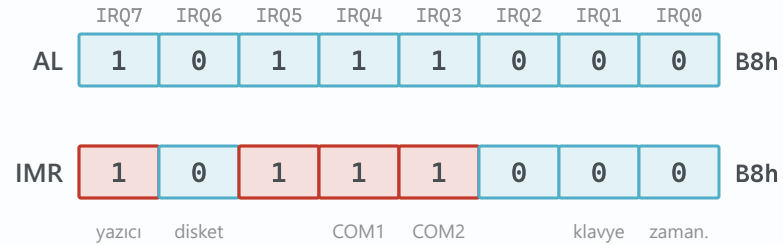
DEĞİŞKENLER

AL = **B8h** CX = **0003h** IMR = **B8h**

Oku–değiştir–yaz kalıbı: OR ile maskelenir, AND ile açılır, böylece diğer bitler korunur. CLI'den farkı: yalnızca seçilen aygıt bekletilir.

Kod İzleme: Tek Bir IRQ'yu Maskelemek

```
1 org 100h
2 in al, 21h            ; IMR'yi oku
3 or al, 00000010b      ; IRQ1 maskele
4 out 21h, al
5 mov cx, 3             ; kritik iş
6 in al, 21h
7 and al, 11111101b     ; IRQ1 aç
8 out 21h, al
▶ ret
```



kırmızı = maskeli (1) · port 21h = IMR

`ret` : maske ilk hâline döndü. Başkasının bitlerine dokunulmadı.

Adım 8 / 8

DEĞİŞKENLER

`AL = B8h` `CX = 0003h` `IMR = B8h`

Oku–değiştir–yaz kalıbı: OR ile maskelenir, AND ile açılır, böylece diğer bitler korunur. CLI'den farkı: yalnızca seçilen aygıt bekletilir.

DONANIM KESMELERİ

INTR Ne Zaman Kabul Edilmez?

Komut ortasında

İstek, yürüyen komut bitene kadar bekler. Önekler (LOCK, REP, kesim öneki) komutun parçası sayılır; önekle komut arasında kesme kabul edilmez.

SS yüklendikten ve STI'den sonra

`MOV SS, AX` ya da `POP SS`'den sonraki komut da bitene kadar kesme gelmez: `MOV SP, …` güvenle tamamlanır. STI'den sonra da bir komut daha yürütülür; `sti` / `hlt` ya da `sti` / `iret` arasına kesme giremez.

REP'li dizgi komutu

Uzun bir `REP MOVSB` turlar arasında kesilebilir; dönüş adresi öneki gösterir, IRET'ten sonra kalan CX kadar sürer. 8086 yalnızca bir öneki hatırlar: `REP ES: MOVSB` kesilirse ES: öneki kaybolur, bu yüzden böyle bir satır CLI/STI arasına alınır.

HLT ve WAIT

`HLT` işlemciyi durdurur; bir INTR ya da NMI onu uyandırır ve IRET HLT'den sonraki komuta döner. `WAIT` sırasında gelen kesme de kabul edilir.

Kısacası 8086 kesmeyi yalnızca komut sınırlarında kabul eder; birkaç durumda bu sınır bilerek bir komut ileri kaydırılır.

Kesme Gecikmesi (Latency)

İşlem (8086)	Saat turu
`INT n`	51
`INT 3`	52
`INTO` (OF = 1 / OF = 0)	53 / 4
INTR kabulü (iki INTA dahil)	61
`IRET`	24
16 bitlik `IDIV` (yazmaç)	165–184

Kesme gecikmesi: isteğin gelmesiyle ISR'nin ilk komutunun başlaması arasındaki süre.

- En kötü durumda önce yürüyen komut bitmelidir: uzun bir IDIV ≈ 184 tur.
- Ardından kabul: 61 tur. Toplam ≈ 245 tur.
- 5 MHz'de bir tur 200 ns → ≈ **49 µs**.
- CLI ile kapalı geçen süre bu gecikmeye doğrudan eklenir; CLI–STI arası kısa tutulur.

Saniyede 18,2 kez gelen zamanlayıcı kesmesi için bu süre önemsizdir; yüksek hızlı seri port gibi aygıtlarda ISR'nin kısa olması veri kaybını önler.

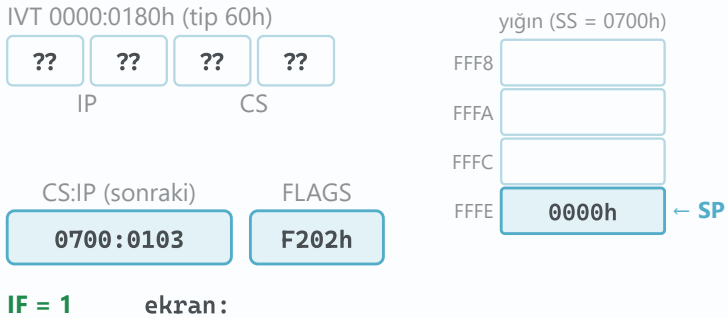
Kendi Kesme Yordamımız

```
org 100h
mov ax, 0
mov es, ax          ; ES = 0: IVT kesimi
cli                 ; kesmeleri kapat
mov word ptr es:[90h*4], offset isr  ; IP
mov word ptr es:[90h*4+2], cs        ; CS
sti                 ; kesmeleri aç
int 90h             ; kendi kesmemiz
ret
isr:
mov ah, 0Eh
mov al, '*'
int 10h
iret                ; RET değil!
```

- Kullanılmayan bir tip seçilir: **90h**. Vektörü `90h × 4 = 240h` adresindedir.
- IVT 0000 kesimindedir; ES = 0 yapılır.
- Vektör yazılırken `CLI` / `STI` ile korunur: yarım yazılmış vektörle kesme gelmemelidir.
- `INT 90h` ekrana * yazar ve program devam eder.
- DOS altında `INT 21h`, AH = `25h` (vektör ata) / `35h` (vektör oku) daha güvenli yoldur.

Kod İzleme: INT 60h ile Kendi ISR'miz

```
1  org 100h
▶ 2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  cli
5  mov word ptr es:[60h*4], offset isr
6  mov word ptr es:[60h*4+2], cs
7  sti
8  mov al, '#'
9  int 60h               ; kendi kesmemiz
10 ret
11 isr:
12 mov ah, 0Eh
13 int 10h
14 iret
```



AX = 0. Vektör tablosu (IVT) 0000:0000'dadır; tip 60h'nin vektörü **60h × 4 = 180h** adresinde.

Adım 1 / 12

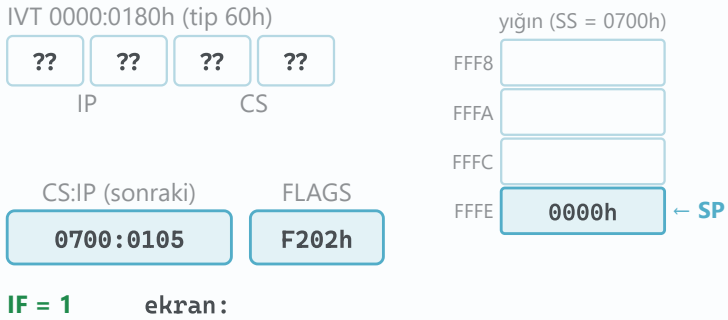
DEĞİŞKENLER

`AX = 0000h` `ES = 0700h` `SP = FFFEh` `FLAGS = F202h` `IF = 1`

Tip 60h kullanıcı programlarına ayrılmış tiplerdendir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: INT 60h ile Kendi ISR'miz

```
1  org 100h
2  mov ax, 0
▶  mov es, ax            ; ES = 0: IVT
4  cli
5  mov word ptr es:[60h*4], offset isr
6  mov word ptr es:[60h*4+2], cs
7  sti
8  mov al, '#'
9  int 60h               ; kendi kesmemiz
10 ret
11 isr:
12 mov ah, 0Eh
13 int 10h
14 iret
```



ES = 0000h: ES:[180h] artık IVT'deki 0000:0180h baytını gösterir.

Adım 2 / 12

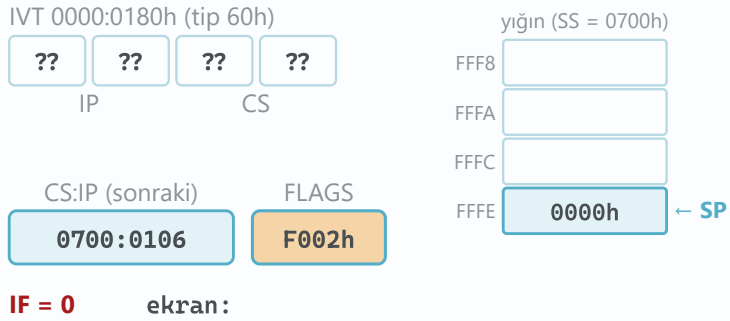
DEĞİŞKENLER

`AX = 0000h` `ES = 0000h` `SP = FFFEh` `FLAGS = F202h` `IF = 1`

Tip 60h kullanıcı programlarına ayrılmış tiplerdendir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: INT 60h ile Kendi ISR'miz

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
▶  cli
5  mov word ptr es:[60h*4], offset isr
6  mov word ptr es:[60h*4+2], cs
7  sti
8  mov al, '#'
9  int 60h               ; kendi kesmemiz
10 ret
11 isr:
12 mov ah, 0Eh
13 int 10h
14 iret
```



`cli` : IF = 0. Vektör yarı yazılmışken donanım kesmesi gelmesin. FLAGS = F002h.

Adım 3 / 12

DEĞİŞKENLER

AX = **0000h** · ES = **0000h** · SP = **FFFEh** · FLAGS = **F002h** · IF = **0**

Tip 60h kullanıcı programlarına ayrılmış tiplerdendir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: INT 60h ile Kendi ISR'miz

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  cli
▶  mov word ptr es:[60h*4], offset isr
6  mov word ptr es:[60h*4+2], cs
7  sti
8  mov al, '#'
9  int 60h               ; kendi kesmemiz
10 ret
11 isr:
12 mov ah, 0Eh
13 int 10h
14 iret
```

IVT 0000:0180h (tip 60h)

18	01	??	??
IP		CS	

yığın (SS = 0700h)

FFF8	
FFFA	
FFFC	
FFFE	0000h ← SP

CS:IP (sonraki)	FLAGS
0700:010D	F002h

IF = 0 ekran:

Düşük sözcük = IP = `offset isr` = **0118h**: bellekte 18 01 (little-endian).

Adım 4 / 12

DEĞİŞKENLER

`AX = 0000h` `ES = 0000h` `SP = FFFEh` `FLAGS = F002h` `IF = 0`

Tip 60h kullanıcı programlarına ayrılmış tiplerdendir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: INT 60h ile Kendi ISR'miz

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  cli
5  mov word ptr es:[60h*4], offset isr
▶  mov word ptr es:[60h*4+2], cs
7  sti
8  mov al, '#'
9  int 60h               ; kendi kesmemiz
10 ret
11 isr:
12 mov ah, 0Eh
13 int 10h
14 iret
```

IVT 0000:0180h (tip 60h)

18	01	00	07
IP		CS	

vektör = 0700:0118h

CS:IP (sonraki)	FLAGS
0700:0112	F002h

IF = 0

ekran:

yığın (SS = 0700h)

FFF8	
FFFA	
FFFC	
FFFE	0000h ← SP

Yüksek sözcük = CS = **0700h**: 00 07. Vektör artık **0700:0118**.

Adım 5 / 12

DEĞİŞKENLER

AX = **0000h** · ES = **0000h** · SP = **FFFEh** · FLAGS = **F002h** · IF = **0**

Tip 60h kullanıcı programlarına ayrılmış tiplerdendir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: INT 60h ile Kendi ISR'miz

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  cli
5  mov word ptr es:[60h*4], offset isr
6  mov word ptr es:[60h*4+2], cs
▶  sti
8  mov al, '#'
9  int 60h               ; kendi kesmemiz
10 ret
11 isr:
12 mov ah, 0Eh
13 int 10h
14 iret
```

IVT 0000:0180h (tip 60h)

18	01	00	07
IP		CS	

vektör = 0700:0118h

CS:IP (sonraki)	FLAGS
0700:0113	F202h

IF = 1

ekran:

yığın (SS = 0700h)

FFF8	
FFFA	
FFFC	
FFFE	0000h ← SP

`sti` : IF = 1, kesmeler yeniden açık. FLAGS = **F202h** (8086'da bit 1 ve 12–15. bitler hep 1).

Adım 6 / 12

DEĞİŞKENLER

AX = **0000h** · ES = **0000h** · SP = **FFFEh** · FLAGS = **F202h** · IF = **1**

Tip 60h kullanıcı programlarına ayrılmış tiplerdendir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: INT 60h ile Kendi ISR'miz

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  cli
5  mov word ptr es:[60h*4], offset isr
6  mov word ptr es:[60h*4+2], cs
7  sti
▶  mov al, '#'
9  int 60h               ; kendi kesmemiz
10 ret
11 isr:
12 mov ah, 0Eh
13 int 10h
14 iret
```

IVT 0000:0180h (tip 60h)

18	01	00	07
IP		CS	

vektör = 0700:0118h

CS:IP (sonraki)	FLAGS
0700:0115	F202h

IF = 1

ekran:

yığın (SS = 0700h)

FFF8	
FFFA	
FFFC	
FFFE	0000h ← SP

AL = '#' (23h): ISR'ye yazmaç ile iletilen parametre.

Adım 7 / 12

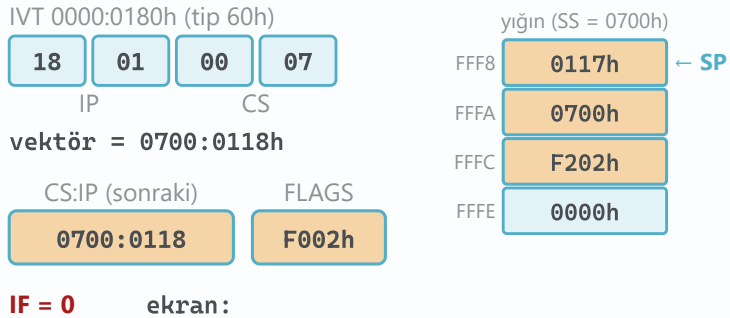
DEĞİŞKENLER

AX = 0023h | ES = 0000h | SP = FFFEh | FLAGS = F202h | IF = 1

Tip 60h kullanıcı programlarına ayrılmış tiplerdendir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: INT 60h ile Kendi ISR'miz

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  cli
5  mov word ptr es:[60h*4], offset isr
6  mov word ptr es:[60h*4+2], cs
7  sti
8  mov al, '#'
▶ int 60h                ; kendi kesmemiz
10 ret
11 isr:
12 mov ah, 0Eh
13 int 10h
14 iret
```



`int 60h` : FLAGS (F202h), CS (0700h), dönüş IP'si (**0117h**) yığına; IF = 0; CS:IP ← [180h] = 0700:0118.

Adım 8 / 12

DEĞİŞKENLER

AX = **0023h** · ES = **0000h** · SP = **FFF8h** · FLAGS = **F002h** · IF = **0**

Tip 60h kullanıcı programlarına ayrılmış tiplerdendir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: INT 60h ile Kendi ISR'miz

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  cli
5  mov word ptr es:[60h*4], offset isr
6  mov word ptr es:[60h*4+2], cs
7  sti
8  mov al, '#'
9  int 60h               ; kendi kesmemiz
10 ret
11 isr:
▶ mov ah, 0Eh
13 int 10h
14 iret
```

IVT 0000:0180h (tip 60h)

18	01	00	07
IP		CS	

vektör = 0700:0118h

CS:IP (sonraki)	FLAGS
0700:011A	F002h

IF = 0 ekran: ISR içinde

yığın (SS = 0700h)

FFF8	0117h	← SP
FFFA	0700h	
FFFC	F202h	
FFFE	0000h	

ISR çalışıyor: AH = 0Eh (teletype).

Adım 9 / 12

DEĞİŞKENLER

AX = 0E23h | ES = 0000h | SP = FFF8h | FLAGS = F002h | IF = 0

Tip 60h kullanıcı programlarına ayrılmış tiplerdendir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: INT 60h ile Kendi ISR'miz

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  cli
5  mov word ptr es:[60h*4], offset isr
6  mov word ptr es:[60h*4+2], cs
7  sti
8  mov al, '#'
9  int 60h               ; kendi kesmemiz
10 ret
11 isr:
12 mov ah, 0Eh
▶ int 10h
14 iret
```

IVT 0000:0180h (tip 60h)

18	01	00	07
	IP		CS

vektör = 0700:0118h

CS:IP (sonraki)	FLAGS
0700:011C	F002h

yığın (SS = 0700h)

FFF8	0117h	← SP
FFFA	0700h	
FFFC	F202h	
FFFE	0000h	

IF = 0 **ekran: #** **ISR içinde**

`int 10h` : ekrana # yazılır. ISR içinden başka bir kesme çağrılabilir.

Adım 10 / 12

DEĞİŞKENLER

AX = **0E23h** · ES = **0000h** · SP = **FFF8h** · FLAGS = **F002h** · IF = **0**

Tip 60h kullanıcı programlarına ayrılmış tiplerdendir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: INT 60h ile Kendi ISR'miz

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  cli
5  mov word ptr es:[60h*4], offset isr
6  mov word ptr es:[60h*4+2], cs
7  sti
8  mov al, '#'
9  int 60h               ; kendi kesmemiz
10 ret
11 isr:
12 mov ah, 0Eh
13 int 10h
▶  iret
```

IVT 0000:0180h (tip 60h)

18	01	00	07
IP		CS	

vektör = 0700:0118h

CS:IP (sonraki)	FLAGS
0700:0117	F202h

yığın (SS = 0700h)

FFF8	
FFFA	
FFFC	
FFFE	0000h ← SP

IF = 1 **ekran: #** **ISR içinde**

`iret` : IP = 0117h, CS = 0700h, FLAGS = F202h geri alınır; IF yeniden 1, SP = FFFEh.

Adım 11 / 12

DEĞİŞKENLER

AX = **0E23h** ES = **0000h** SP = **FFFEh** FLAGS = **F202h** IF = **1**

Tip 60h kullanıcı programlarına ayrılmış tiplerdendir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: INT 60h ile Kendi ISR'miz

```
1  org 100h
2  mov ax, 0
3  mov es, ax            ; ES = 0: IVT
4  cli
5  mov word ptr es:[60h*4], offset isr
6  mov word ptr es:[60h*4+2], cs
7  sti
8  mov al, '#'
9  int 60h               ; kendi kesmemiz
▶ ret
11 isr:
12 mov ah, 0Eh
13 int 10h
14 iret
```

IVT 0000:0180h (tip 60h)

18	01	00	07
IP		CS	

vektör = 0700:0118h

CS:IP (sonraki)	FLAGS
0700:0000	F202h

IF = 1 **ekran: #**

yığın (SS = 0700h)

FFF8	
FFFA	
FFFC	
FFFE	

`ret` : ana program kaldığı yerden sürdü ve DOS'a döner.

Adım 12 / 12

DEĞİŞKENLER

AX = **0E23h** · ES = **0000h** · SP = **0000h** · FLAGS = **F202h** · IF = **1**

Tip 60h kullanıcı programlarına ayrılmış tiplerdendir. Değerler emu8086 benzeri .COM yüklemesi (CS = 0700h) için.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax            ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]     ; yeni IP ⇔ eski
 6 mov eski, ax
 7 mov ax, cs
 8 xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
10 mov ax, 1234h
11 int 3                 ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx      ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h         ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```

IVT 0000:000Ch (tip 03h)

03	10	00	F0
	IP		CS

vektör = F000:1003h

AX	BX	CS:IP	IF
0000	0000	0700:0104	1

çıktı:

yığın (SS = 0700)

FFF6	....
FFF8	....
FFFA	....
FFFC	....
FFFE	0000 ← SP

ES = 0000h: IVT. Tip 3'ün vektörü 3 × 4 = **000Ch** adresinde.

Adım 1 / 17

DEĞİŞKENLER

AX = **0000h** | BX = **0000h** | SP = **FFFEh** | IF = **1**

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
1  org 100h
2  xor ax, ax
3  mov es, ax           ; ES = 0: IVT
▶ mov ax, offset isr3
5  xchg ax, es:[3*4]    ; yeni IP ⇔ eski
6  mov eski, ax
7  mov ax, cs
8  xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
9  mov eski+2, ax
10 mov ax, 1234h
11 int 3                ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h        ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```

IVT 0000:000Ch (tip 03h)

03	10	00	F0
	IP		CS

vektör = F000:1003h

AX	BX	CS:IP	IF
0130	0000	0700:0107	1

yığın (SS = 0700)

FFF6	....
FFF8	....
FFFA	....
FFFC	....
FFFE	0000 ← SP

çıktı:

AX = `offset isr3` = **0130h**.

Adım 2 / 17

DEĞİŞKENLER

AX = **0130h** | BX = **0000h** | SP = **FFFEh** | IF = **1**

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
1  org 100h
2  xor ax, ax
3  mov es, ax           ; ES = 0: IVT
4  mov ax, offset isr3
▶  xchg ax, es:[3*4]    ; yeni IP ⇔ eski
6  mov eski, ax
7  mov ax, cs
8  xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
9  mov eski+2, ax
10 mov ax, 1234h
11 int 3                ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h        ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```

IVT 0000:000Ch (tip 03h)

30	01	00	F0
IP		CS	

vektör = F000:0130h

AX	BX	CS:IP	IF
1003	0000	0700:010C	1

yığın (SS = 0700)

FFF6	....
FFF8	....
FFFA	....
FFFC	....
FFFE	0000 ← SP

çıktı:

`xchg ax, es:[12]` : tek komutta yeni IP yazılır, eski IP (**1003h**) AX'e gelir.

Adım 3 / 17

DEĞİŞKENLER

AX = **1003h** | BX = **0000h** | SP = **FFFEh** | IF = **1**

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax           ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]    ; yeni IP ⇔ eski
▶6 mov eski, ax
 7 mov ax, cs
 8 xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
10 mov ax, 1234h
11 int 3                ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h        ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```

IVT 0000:000Ch (tip 03h)

30	01	00	F0
IP		CS	

vektör = F000:0130h

AX	BX	CS:IP	IF
1003	0000	0700:010F	1

yığın (SS = 0700)

FFF6	....
FFF8	....
FFFA	....
FFFC	....
FFFE	0000 ← SP

çıktı:

Eski IP `eski` değişkenine saklandı.

Adım 4 / 17

DEĞİŞKENLER

`AX = 1003h` `BX = 0000h` `SP = FFFEh` `IF = 1`

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax           ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]    ; yeni IP ⇔ eski
 6 mov eski, ax
▶ 7 mov ax, cs
 8 xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
10 mov ax, 1234h
11 int 3                ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h        ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```

IVT 0000:000Ch (tip 03h)

30	01	00	F0
IP		CS	

vektör = F000:0130h

AX	BX	CS:IP	IF
0700	0000	0700:0111	1

yığın (SS = 0700)

FFF6	....
FFF8	....
FFFA	....
FFFC	....
FFFE	0000 ← SP

çıktı:

AX = CS = 0700h.

Adım 5 / 17

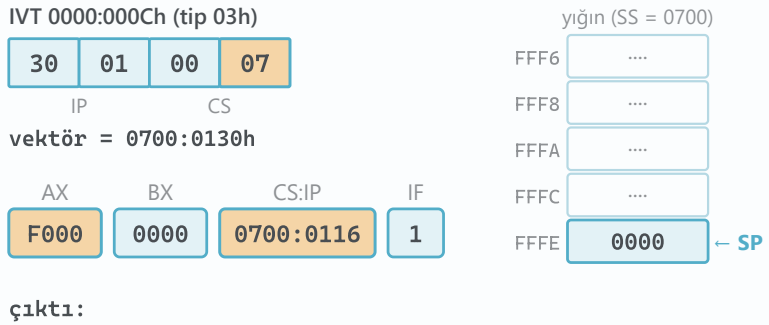
DEĞİŞKENLER

AX = **0700h** | BX = **0000h** | SP = **FFFEh** | IF = **1**

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax           ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]    ; yeni IP ⇔ eski
 6 mov eski, ax
 7 mov ax, cs
 ▶ xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
10 mov ax, 1234h
11 int 3                ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h        ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```



`xchg` : yeni CS yazıldı, eski CS = **F000h** AX'te. Vektör artık 0700:0130.

Adım 6 / 17

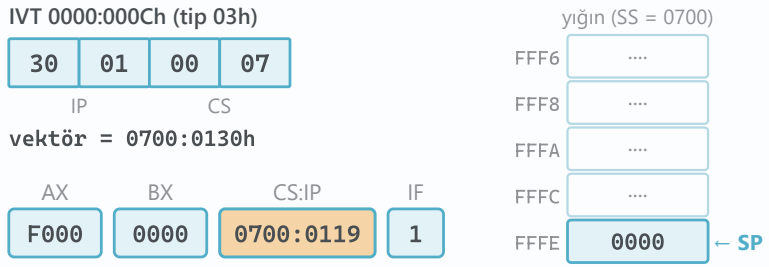
DEĞİŞKENLER

`AX = F000h` `BX = 0000h` `SP = FFFEh` `IF = 1`

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
1  org 100h
2  xor ax, ax
3  mov es, ax           ; ES = 0: IVT
4  mov ax, offset isr3
5  xchg ax, es:[3*4]    ; yeni IP ⇔ eski
6  mov eski, ax
7  mov ax, cs
8  xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
▶ mov eski+2, ax
10 mov ax, 1234h
11 int 3                ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h        ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```



çıktı:

Eski CS de saklandı: eski vektör F000:1003. Tip 3'ü yalnızca program çağırdığı için CLI gerekmedi; donanım tiplerinde bu satırlar CLI/STI arasına alınır.

Adım 7 / 17

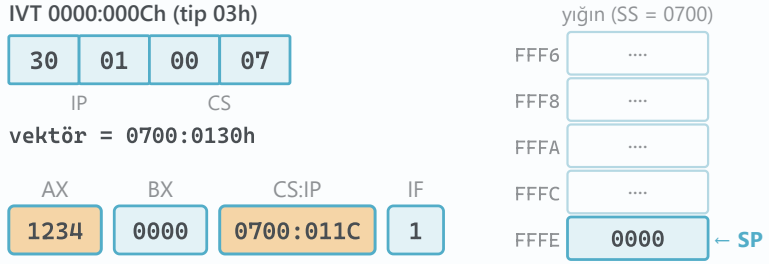
DEĞİŞKENLER

AX = **F000h** | BX = **0000h** | SP = **FFFEh** | IF = **1**

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax           ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]    ; yeni IP ⇔ eski
 6 mov eski, ax
 7 mov ax, cs
 8 xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
▶10 mov ax, 1234h
11 int 3                ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h        ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```



AX = 1234h: ana programın değeri. ISR bunu bozmamalı.

Adım 8 / 17

DEĞİŞKENLER

AX = **1234h** | BX = **0000h** | SP = **FFFEh** | IF = **1**

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax           ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]    ; yeni IP ⇔ eski
 6 mov eski, ax
 7 mov ax, cs
 8 xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
10 mov ax, 1234h
▶ int 3                 ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h        ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```

IVT 0000:000Ch (tip 03h)

30	01	00	07
IP		CS	

vektör = 0700:0130h

AX	BX	CS:IP	IF
1234	0000	0700:0130	0

çıktı:

yığın (SS = 0700)

FFF6	....	
FFF8	011D	← SP
FFFA	0700	
FFFC	F246	
FFFE	0000	

`int 3` (tek bayt, CCh): FLAGS, CS, dönüş IP'si **011Dh** yığına; CS:IP ← 0700:0130.

Adım 9 / 17

DEĞİŞKENLER

AX = 1234h | BX = 0000h | SP = FFF8h | IF = 0

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax           ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]    ; yeni IP ⇔ eski
 6 mov eski, ax
 7 mov ax, cs
 8 xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
10 mov ax, 1234h
11 int 3               ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx    ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
▶ isr3: push ax
18 mov ax, 0E21h       ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```

IVT 0000:000Ch (tip 03h)

30	01	00	07
IP		CS	

vektör = 0700:0130h

AX	BX	CS:IP	IF
1234	0000	0700:0131	0

yığın (SS = 0700)

FFF6	1234	← SP
FFF8	011D	
FFFA	0700	
FFFC	F246	
FFFE	0000	

çıktı:

`push ax` : ISR, kullanacağı AX'i önce saklar (1234h yığında).

Adım 10 / 17

DEĞİŞKENLER

AX = **1234h** | BX = **0000h** | SP = **FFF6h** | IF = **0**

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax           ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]    ; yeni IP ⇔ eski
 6 mov eski, ax
 7 mov ax, cs
 8 xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
10 mov ax, 1234h
11 int 3                ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
▶ mov ax, 0E21h        ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```

IVT 0000:000Ch (tip 03h)

30	01	00	07
IP		CS	

vektör = 0700:0130h

AX	BX	CS:IP	IF
0E21	0000	0700:0134	0

yığın (SS = 0700)

FFF6	1234	← SP
FFF8	011D	
FFFA	0700	
FFFC	F246	
FFFE	0000	

çıktı:

AX = 0E21h: AH = 0Eh (teletype), AL = 21h ('!').

Adım 11 / 17

DEĞİŞKENLER

AX = 0E21h | BX = 0000h | SP = FFF6h | IF = 0

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax           ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]    ; yeni IP ⇔ eski
 6 mov eski, ax
 7 mov ax, cs
 8 xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
10 mov ax, 1234h
11 int 3                ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h        ; AH=0Eh, AL='!'
▶ int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```

IVT 0000:000Ch (tip 03h)

30	01	00	07
IP		CS	

vektör = 0700:0130h

AX	BX	CS:IP	IF
0E21	0000	0700:0136	0

yığın (SS = 0700)

FFF6	1234	← SP
FFF8	011D	
FFFA	0700	
FFFC	F246	
FFFE	0000	

çıktı: !

`int 10h` : ekrana **!** yazılır.

Adım 12 / 17

DEĞİŞKENLER

AX = **0E21h** | BX = **0000h** | SP = **FFF6h** | IF = **0**

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax           ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]    ; yeni IP ⇔ eski
 6 mov eski, ax
 7 mov ax, cs
 8 xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
10 mov ax, 1234h
11 int 3                ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h        ; AH=0Eh, AL='!'
19 int 10h
▶ pop ax
21 iret
22 eski dw ?, ?
```

IVT 0000:000Ch (tip 03h)

30	01	00	07
IP		CS	

vektör = 0700:0130h

AX	BX	CS:IP	IF
1234	0000	0700:0137	0

çıktı: !

yığın (SS = 0700)

FFF6	....	
FFF8	011D	← SP
FFFA	0700	
FFFC	F246	
FFFE	0000	

`pop ax` : AX = **1234h**, ana programın değeri geri geldi.

Adım 13 / 17

DEĞİŞKENLER

AX = **1234h** | BX = **0000h** | SP = **FFF8h** | IF = **0**

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax           ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]    ; yeni IP ⇔ eski
 6 mov eski, ax
 7 mov ax, cs
 8 xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
10 mov ax, 1234h
11 int 3                ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h        ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
▶ iret
22 eski dw ?, ?
```

IVT 0000:000Ch (tip 03h)

30	01	00	07
IP		CS	

vektör = 0700:0130h

AX	BX	CS:IP	IF
1234	0000	0700:011D	1

yığın (SS = 0700)

FFF6	....
FFF8	....
FFFA	....
FFFC	....
FFFE	0000 ← SP

çıktı: !

`iret` : IP, CS, FLAGS geri alınır; SP = FFFEh.

Adım 14 / 17

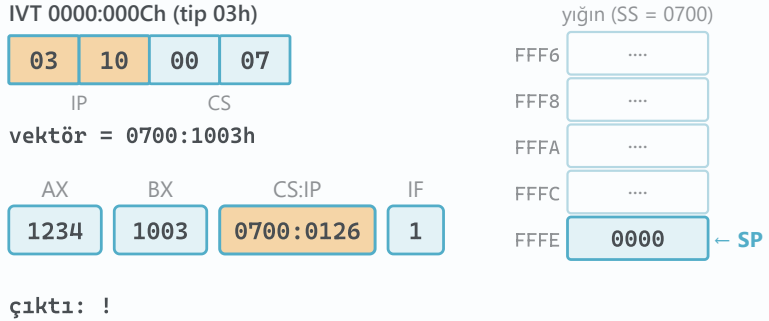
DEĞİŞKENLER

AX = **1234h** | BX = **0000h** | SP = **FFFEh** | IF = **1**

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax           ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]    ; yeni IP ⇔ eski
 6 mov eski, ax
 7 mov ax, cs
 8 xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
10 mov ax, 1234h
11 int 3                ; bizim ISR
▶ mov bx, eski
▶ mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h       ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```



Eski IP vektöre geri yazıldı.

Adım 15 / 17

DEĞİŞKENLER

AX = **1234h** | BX = **1003h** | SP = **FFFEh** | IF = **1**

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax           ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]    ; yeni IP ⇔ eski
 6 mov eski, ax
 7 mov ax, cs
 8 xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
10 mov ax, 1234h
11 int 3                ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
▶ mov bx, eski+2
▶ mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h        ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```



Eski CS de geri yazıldı: tip 3 yine eski yordamı gösteriyor.

Adım 16 / 17

DEĞİŞKENLER

AX = **1234h** | BX = **F000h** | SP = **FFFEh** | IF = **1**

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

Kod İzleme: Vektörü Sakla, Değiştir, Geri Yükle

```
 1 org 100h
 2 xor ax, ax
 3 mov es, ax           ; ES = 0: IVT
 4 mov ax, offset isr3
 5 xchg ax, es:[3*4]    ; yeni IP ⇔ eski
 6 mov eski, ax
 7 mov ax, cs
 8 xchg ax, es:[3*4+2] ; yeni CS ⇔ eski
 9 mov eski+2, ax
10 mov ax, 1234h
11 int 3                ; bizim ISR
12 mov bx, eski
13 mov es:[3*4], bx     ; geri yükle
14 mov bx, eski+2
15 mov es:[3*4+2], bx
16 ret
17 isr3: push ax
18 mov ax, 0E21h        ; AH=0Eh, AL='!'
19 int 10h
20 pop ax
21 iret
22 eski dw ?, ?
```

IVT 0000:000Ch (tip 03h)

03	10	00	F0
	IP		CS

vektör = F000:1003h

AX	BX	CS:IP	IF
1234	F000	0700:0000	1

yığın (SS = 0700)

FFF6	....
FFF8	....
FFFA	....
FFFC	....
FFFE	....

çıktı: !

`ret` : AX = 1234h korundu, vektör tablosu programdan önceki hâlinde.

Adım 17 / 17

DEĞİŞKENLER

AX = **1234h** | BX = **F000h** | SP = **0000h** | IF = **1**

Eski vektör F000:1003h öykünücünün varsayılan yordamıdır. DOS altında: INT 21h, AH = 35h / 25h.

ISR Yazım Kuralları ve Sık Yapılan Hatalar

Kural	Uyulmazsa
Kullanılan her yazmaç PUSH/POP ile korunur (FLAGS'i işlemci korur).	Ana program, değeri kendiliğinden değişmiş bir yazmaçla devam eder; hata rastgele anlarda görülür.
Donanım ISR'si 8259A'ya EOI gönderir: `MOV AL, 20h` · `OUT 20h, AL`.	8259A ISR bitini silmez; aynı ve daha düşük öncelikli IRQ'lar bir daha gelmez.
ISR kısa tutulur; uzun iş bir bayrak değişkeniyle ana programa bırakılır.	Kesme gecikmesi büyür, hızlı aygıtların verisi kaybolur.
Donanım ISR'sinden INT 21h çağrılmaz.	DOS yeniden girişli (reentrant) değildir; kesilen DOS çağrısının verileri bozulur.
Vektör CLI ile ya da AH = 25h ile yazılır, çıkışta eskisi geri yüklenir.	Program bittikten sonra kesme, artık başka bir şeyin bulunduğu belleğe atlar.
ISR `IRET` ile biter.	Yığın 4 bayt kayar, IF kapalı kalır.

Uzun bir ISR içinde STI ile daha öncelikli kesmelere izin verilebilir; bu durumda ISR'nin kendisi de yeniden girişli yazılmalıdır.

Laboratuvar: Kesme Mekanizması

INT 90h: kendi kesme yordamımız

KOD · **29 bayt**

```
 1 org 100h
 2 mov ax, 0
 3 mov es, ax              ; ES =…
 4 cli                     ; kesm…
 5 mov word ptr es:[90h*4], o…
 6 mov word ptr es:[90h*4+2],…
 7 sti                     ; kesm…
 8 int 90h                 ; kend…
 9 ret
10 isr:
11 mov ah, 0Eh
12 mov al, '*'
13 int 10h
14 iret                    ; RET …
```

YAZMAÇLAR

AX	0000	AH 00	AL 00		
BX	0000	BH 00	BL 00		
CX	001D	CH 00	CL 1D		
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 00 00 8E C0 FA 26 C7
0108 06 40 02 16 01 26 8C 0E
0110 42 02 FB CD 90 C3 B4 0E
```

YIĞIN SS:SP

```
FFF2 0000
FFF4 0000
FFF6 0000
FFF8 0000
FFFA 0000
FFFC 0000
FFFE 0000 ◂ SP          DOS dönüşü
```

ÇIKTI

Program yüklendi: 29 bayt, CS:IP = 0700:0100h. emu8086'daki gibi CX başlangıçta program boyutunu (001Dh) gösterir. Adım düğmesiyle komutlar tek tek yürütülür; vurgulu satır sıradaki komuttur.

ÖZET

Özet

Kesme
İşlemci işini bırakır, ISR'yi çalıştırır, `IRET` ile kaldığı yere döner.

Vektör tablosu
0000:0000–03FF, 256 × 4 bayt. Tip n → adres n × 4 (IP), n × 4 + 2 (CS).

INT n
FLAGS, CS, IP yığına (SP −6), IF = TF = 0, CS:IP ← vektör.

Donanım kesmesi
INTR yalnız IF = 1 iken, komut sınırında kabul edilir; tip numarası ikinci INTA darbesinde 8259A'dan gelir. NMI hep INT 2.

8259A
IMR tek tek IRQ'ları maskeler (port 21h); ISR sonunda EOI: `MOV AL, 20h` · `OUT 20h, AL`.

Kendi ISR
Vektör CLI/STI arasında yazılır, kullanılan yazmaçlar korunur, `IRET` ile biter; çıkışta eski vektör geri yüklenir.

ÖZET

Alıştırmalar

1. `INT 13h` vektörü hangi fiziksel adreslerde bulunur?
2. `INT 21h` çalışmadan önce SP = FFFEh ise ISR içindeyken SP kaçtır? Yığında ne vardır?
3. IMR = B8h iken yalnız IRQ0'ı (zamanlayıcı) da maskeleyen komutları yazın. IMR kaç olur?
4. Tip 61h'nin vektörü 0700:0120h yapılacak. Hangi fiziksel adreslere hangi baytlar yazılır?
5. 5 MHz'lik 8086'da INTR geldiğinde `MUL BX` (en çok 133 tur) yeni başlamışsa kesme gecikmesi en çok ne kadardır?
6. ISR'nin `IRET` yerine `RET` ile bitmesi ne gibi sorunlara yol açar?

▶ **İpuçları / cevaplar**

Kaynaklar

- A. S. Tanenbaum, *Modern Operating Systems*, 3. baskı, Pearson, 2008.
- Intel, *The 8086 Family User's Manual*, 1979 — Interrupts bölümü.
- B. B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson, 2009 — Bölüm 6 ve 12.
- emu8086 yardım belgeleri: “Interrupts” ve “Supported interrupts” sayfaları.
- R. Brown, *Ralf Brown's Interrupt List* (BIOS/DOS kesme başvurusu).

Son

Bölüm 5.1: Kesmeler

Sonraki bölüm: Bölüm 5.2: Kesme Fonksiyonları · Terim sözlüğü