4 Kesme (*Interrupt*)

Donanım Kesmesi:

Bir dış birimde (örneğin G/Ç arabirimi) belli bir koşul oluştuğunda bu birim MİB'e kesme isteği gönderir.

Tasarım problemleri:

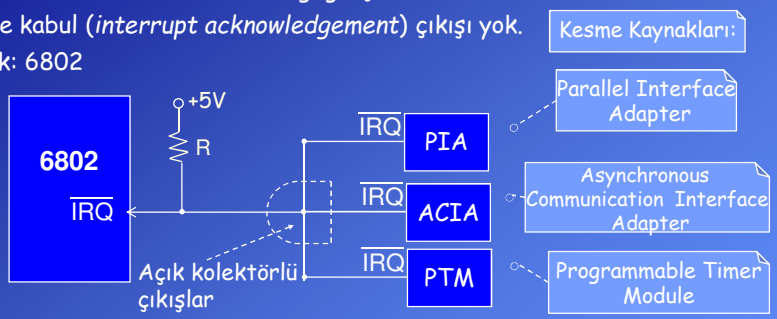
- **Kaynak:** Bir MİB'ne birden fazla kesme kaynağı bağlı olabilir. MİB kesme isteğinin hangi kaynaktan geldiğini nasıl belirler?
- **Öncelik:** Aynı anda birden fazla kaynak kesme isteğinde bulunabilir. MİB hangi isteği yerine getireceğine (öncelik) nasıl karar verir?
- **Başlangıç adresi:** Kesme isteği kabul edilen kaynağa ilişkin kesme hizmet programının (KHP) (*interrupt service routine ISR*) başlangıç adresi nasıl belirlenir (Vektörlü / otovektörlü)?

4.1 MİB'lerin Kesme düzenleri:

a) MİB'in sadece bir kesme isteği girişi var.

Kesme kabul (*interrupt acknowledgement*) çıkışı yok.

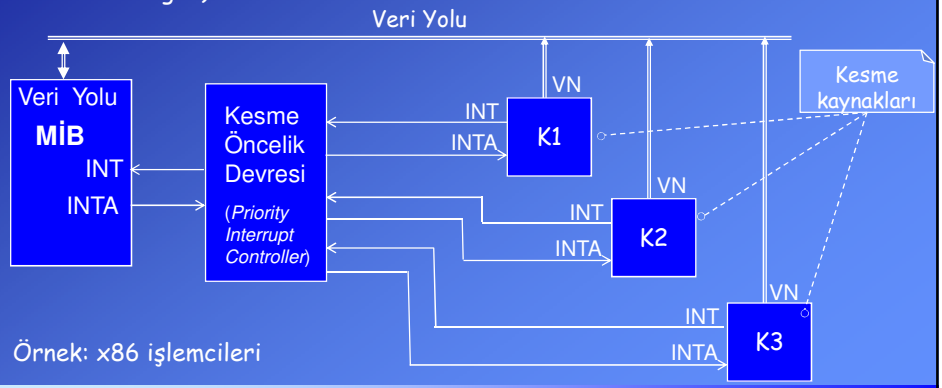
Örnek: 6802



- Tek bir kesme hizmet programı (KHP) (*interrupt service routine - ISR*) var.
- Kesme kaynağı, kesme hizmet programının içinde yoklama (*polling*) ile belirlenir.
- Öncelik sırası da yoklama sırası ile belirlenir.
- Kesme isteğinden vazgeçirme (kesme kabul) yazılım ile yapılır. Kaynak cihazın iskelesini (*port*) okuma/yazma, denetim saklayıcısına yazma gibi.

b) Bir **kesme isteği girişi** (INT) var, bir **kesme kabul çıkışı** (*Interrupt Acknowledge*) (INTA) var, **vektörlü** çalışıyor.

Kesme kaynakları MİB'e kesme öncelik devreleri üzerinden bağlanırlar.
Aynı anda birden fazla istek gelirse "kesme kabul" (INTA) işaretinin hangi kaynağa gönderileceğine kesme öncelik devresi "karar" verir.
İsteği kabul edilen kaynak, vektör numarasını (VN) veri yolu üzerinden MİB'e bildirir.
MİB hizmet programının başlangıç adresini (başlangıç adresinin vektör tablosunda nereden alınacağını) vektör numarasını kullanarak belirler.



Örnek: x86 işlemcileri

4.2 Vektör adresi:

MİB'ler bir kesme isteği kabul edildiğinde hangi hizmet programının çalıştırılacağına ilişkin bilgileri bir kesme **vektör tablosunda** tutarlar.

Bu bilgiler iki farklı şekilde tutulabilir:

1. Vektör tablosunda kesme hizmet programlarının **başlangıç adresi** bulunur.

Kesme isteği kabul edilen kaynak, MİB'e vektör numarasını yani hizmet programının başlangıç adresinin tablodaki satır numarasını gönderir.

MİB ilgili satırdan hizmet programının başlangıç adresini alır ve program sayacına (PC) yazar.

Örnek: MC 68000.

2. Tabloda kesme hizmet programlarının **kendisi (ilk satırı)** bulunur.

Pratikte tüm programları tek tabloya sığdırmak mümkün olmayacağından bu satırlara asıl hizmet programına gitmeyi sağlayan "**JMP hizmet_programi**" komutları yerleştirilir.

Vektörlü ve Otovektörlü Kesmeler:

Vektörlü kesmede kesme isteği kabul edilen kaynak, MİB'e vektör numarasını yani hizmet programının başlangıç adresi ile ilgili bilgiyi gönderir.

Sabit vektörlü (**otovektörlü**) çalışmada vektör numarası kaynak birimden alınmaz.

Her kesme kaynağı (veya düzeyi) için önceden belirlenmiş sabit bir vektör numarası (tabloda belli bir satır) vardır.

Örnek: 6802'de IRQ, NMI, SWI kesmeleri için vektör adresleri sabittir.

Tasarımcı ilgili kesme hizmet programlarının başlangıç adreslerini tablolardaki ilgili gözlere önceden yazar.

MC68000 her iki yönteme göre de (vektörlü ve otovektörlü) çalışabilmektedir.

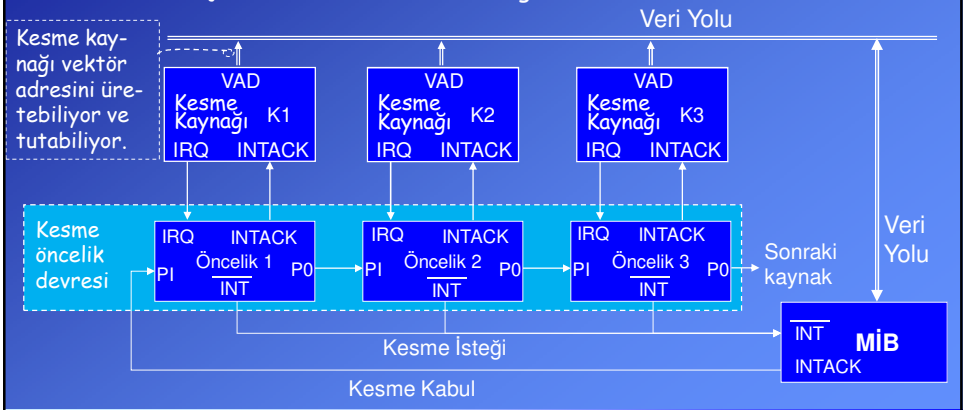
MC 68000 Vektör Tablosu:

Vectors Numbers		Address			
Hex	Decimal	Dec	Hex	Space[6]	Assignment
0	0	0	000	SP	Reset: Initial SSP[2]
1	1	4	004	SP	Reset: Initial PC[2]
2	2	8	008	SD	Bus Error
3	3	12	00C	SD	Address Error
4	4	16	010	SD	Illegal Instruction
5	5	20	014	SD	Zero Divide
6	6	24	018	SD	CHK Instruction
7	7	28	01C	SD	TRAPV Instruction
8	8	32	020	SD	Privilege Violation
9	9	36	024	SD	Trace
A	10	40	028	SD	Line 1010 Emulator
B	11	44	02C	SD	Line 1111 Emulator
C	12[1]	48	030	SD	(Unassigned, Reserved)
D	13[1]	52	034	SD	(Unassigned, Reserved)
E	14	56	038	SD	Format Error[5]
F	15	60	03C	SD	Uninitialized Interrupt Vector
10–17	16–23[1]	64	040	SD	(Unassigned, Reserved)
		92	05C		—
18	24	96	060	SD	Spurious Interrupt[3]
19	25	100	064	SD	Level 1 Interrupt Autovector
1A	26	104	068	SD	Level 2 Interrupt Autovector
1B	27	108	06C	SD	Level 3 Interrupt Autovector
1C	28	112	070	SD	Level 4 Interrupt Autovector
1D	29	116	074	SD	Level 5 Interrupt Autovector
1E	30	120	078	SD	Level 6 Interrupt Autovector
1F	31	124	07C	SD	Level 7 Interrupt Autovector
20–2F	32–47	128	080	SD	TRAP Instruction Vectors[4]
		188	0BC		—
30–3F	48–63[1]	192	0C0	SD	(Unassigned, Reserved)
		255	0FF		—
40–FF	64–255	256	100	SD	User Interrupt Vectors
		1020	3FC		—

4.3 Kesme Öncelik Devreleri (*Priority Interrupt Hardware*)

a) Seri Öncelik Devresi (Papatya Zinciri "*Daisy Chain*") 1

- Öncelik devresi (zincir) değişik önceliklere sahip "halka"lardan oluşur.
- Her kesme kaynağı bu halkalardan birine (*Interrupt Request* - IRQ) bağlıdır.
- Zincirin başındaki halka (öncelik 1) en yüksek önceliğe sahiptir.
- Tüm birimler işlemcinin tek bir kesme isteği hattını kullanır.



Öncelik zincirinin çalışması:

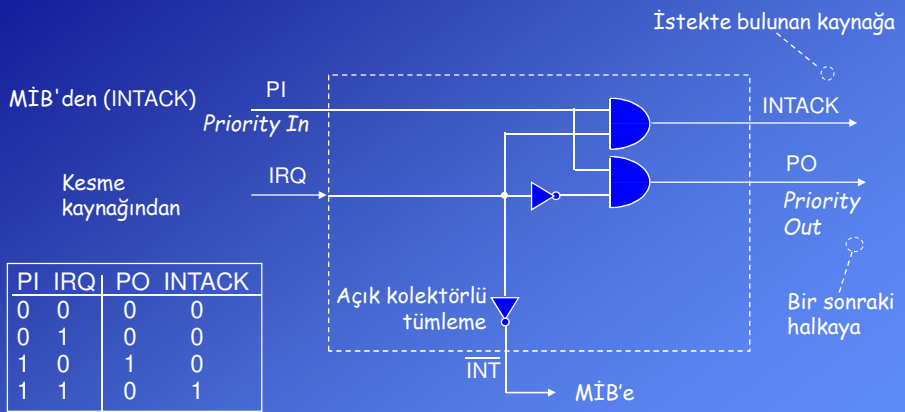
- MİB'den çıkan "kesme kabul" işareti tüm halkalara seri olarak bağlıdır.
- MİB'den "kesme kabul" işareti önce baştaki halkaya (Priority in: PI) gelir.

a. Eğer bu halkaya bağlı olan cihaz istekte bulunduysa
   1. Halka kendisine bağlı olan kesme kaynağına INTACK işaretini gönderir.
   2. Böylece halkaya bağlı olan kaynak vektör adresini (VAD) veri yoluna çıkartır.
   3. Halka (Priority out: PO) çıkışını etkisiz yaparak bir sonraki halkaya (kaynağa) "kesme kabul" işaretinin gitmesini engeller.

b. Eğer kesme kabul işaretini alan halkaya bağlı olan cihaz istekte bulunmadıysa
   1. Halka (Priority out: PO) çıkışını etkin yaparak "kesme kabul" işaretinin bir sonraki halkaya geçmesini sağlar. Böylece sıradaki kaynağın kesme isteğinin olup olmadığına bakılır.

- Bu yönteme **donanım yoklaması** (***hardware poll***) da denir.
- Kesme isteği karşılanan cihaz isteğinden vazgeçer (IRQ) etkisiz olur.
- Kesme kaynağı karşılanmayan cihaz IRQ çıkışını etkin tutarak isteğini sürdürür.

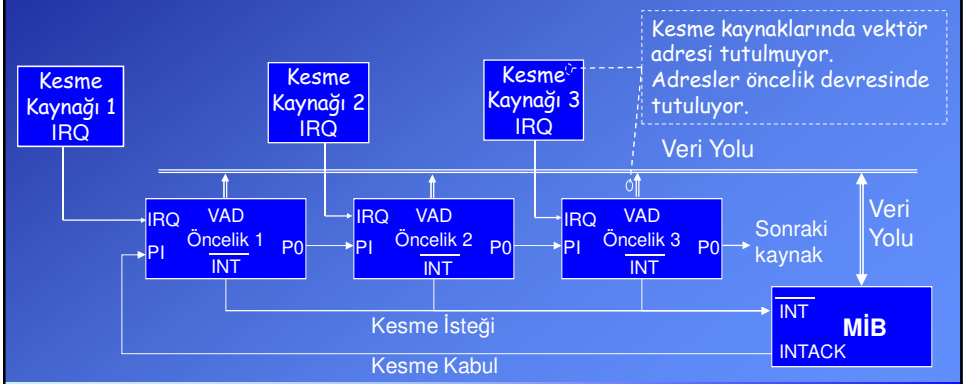
Papatya Zincirinin bir halkasının iç yapısı:



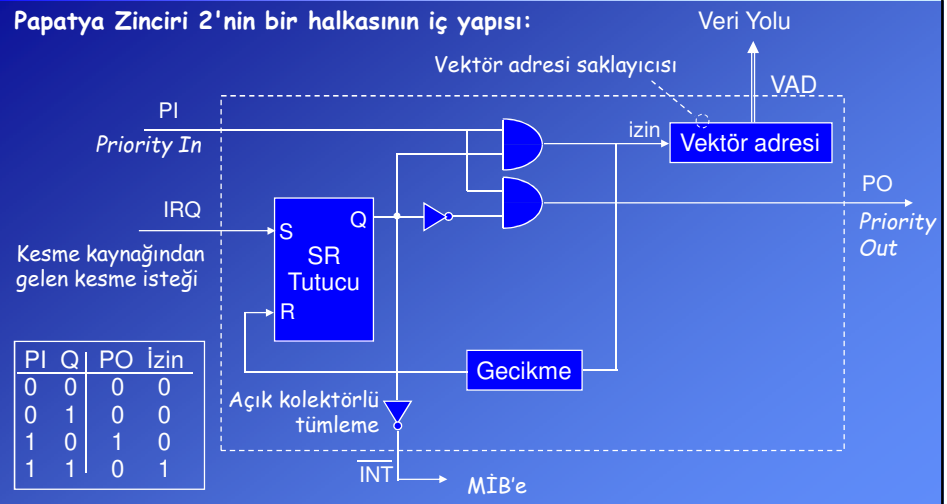
PI	IRQ	PO	INTACK
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

Seri Öncelik Devresi (Papatya Zinciri "*Daisy Chain*") 2

- Önceki devrede kesme kaynakları kendi vektör numaralarına sahiptirler ve INTACK işareti geldiğinde bu adresi veri yoluna çıkartabilmektedirler.
- Aşağıdaki devrede ise kesme kaynakları vektör numarasına sahip değildirler. Bu adresi tutma işi öncelik devresine verilmiştir.
- Ayrıca kesme kaynakları kesme isteğini kabul işareti gelinceye kadar sürdürmemektedirler. İsteği tutma (sürdürme) görevi de öncelik devresindedir.



Papatya Zinciri 2'nin bir halkasının iç yapısı:

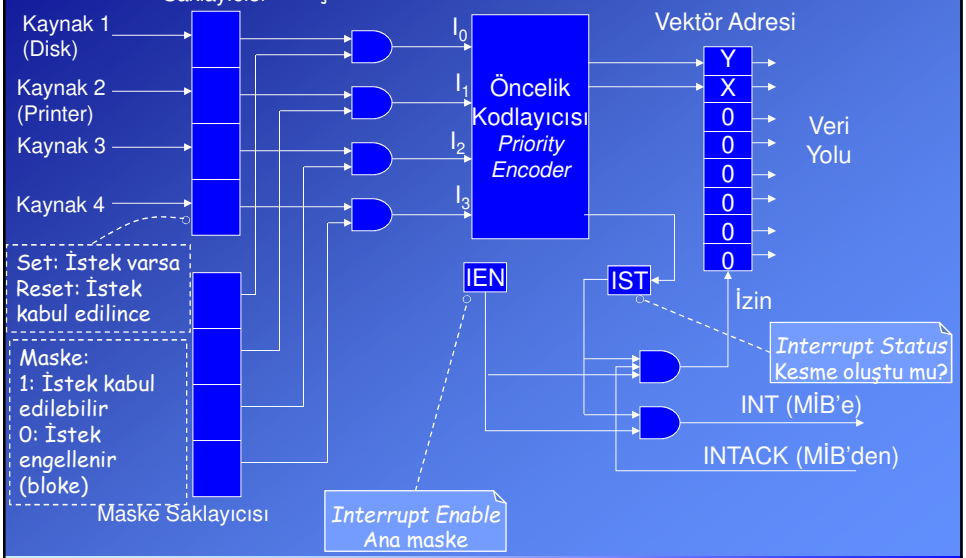


PI	Q	PO	İzin
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

- Kaynaktan gelen kesme isteği SR tutucuyu 1 yapar.
- MİB'den "kesme kabul" işaretini alan halka (Priority in: PI) eğer kendisine bağlı cihaz istekte bulunduysa vektör adresi saklayıcısını etkin yaparak kendi vektör adresini (VAD) veri yoluna koyar.

b) Paralel Öncelik Devresi

Aşağıda **örnek bir devre** gösterilmiştir. Öncelik devreleri başka şekillerde de tasarlanabilir.



Bu örnek sistemde kaynakların vektör adresleri:

Kaynak 1: 0000 0000
Kaynak 2: 0000 0001
Kaynak 3: 0000 0010
Kaynak 4: 0000 0011

Öncelik Kodlayıcısının doğruluk tablosu:

Girişler				Çıkışlar		
I_0	I_1	I_2	I_3	x	y	IST
1	x	x	x	0	0	1
0	1	x	x	0	1	1
0	0	1	x	1	0	1
0	0	0	1	1	1	1
0	0	0	0	Φ	Φ	0

Lojik ifadeler:

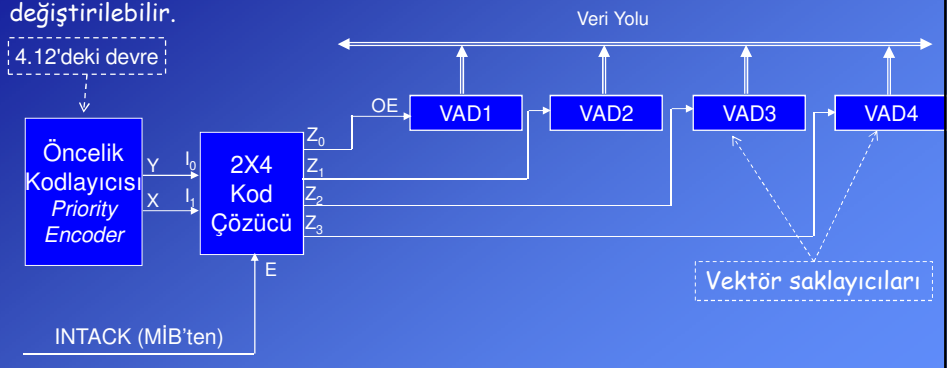
$x = I_0' I_1'$

$y = I_0' I_1 + I_0' I_2'$

$(IST) = I_0 + I_1 + I_2 + I_3$

Önceki örnek sistemde (4.12) kaynakların vektör numaraları önceden belirlenmiştir ve sabittir (Örneğin, Kaynak 1 (disk): 0000 0000).

Sistemi daha esnek yapmak için her kaynak için ayrı bir vektör adres saklayıcısı yerleştirilebilir. Böylece her kaynağın vektör numarası esnek olarak değiştirilebilir.



Kesme isteği kabul edilen kaynağa ilişkin vektör adresi saklayıcısına çıkış izni (*Output enable* OE) verilir.

Bu saklayıcılar MİB tarafından yazma yapılabilecek şekilde sistemdeki uygun bir bellek bölgesine yerleştirilirler. Saklayıcılara sistem programlarında yazılır.

4.4 Kesme Hizmet Programına gidilip dönülürken yapılan işler:

Kesme Hizmet Programından (KHP) Önce:

Hatırlatma; MİB Komut yürütme çevriminden sonra kesme isteklerini değerlendirir.

Eğer bir kesme isteği kabul edilirse MİB kesme çevrimine girer (yansı 1.18).

MİB kesme çevriminde aşağıdaki işleri gerçekleştirir:

(Bu işler MİB'in denetim birimi tarafından iç işlem olarak yapılır; programı ile yapılmazlar.)

```
SP ← SP-1       (Yığın işaretçisi gerektiği kadar –adres kaç sekizli ise- azaltılır)
M[SP] ← PC      (Geri dönüş adresi yığına)
INTACK ← 1      (Kesme kabul)
PC ← VAD        (Vektör adresi)   ya da   PC ← Tablo[VNo]   (Vektör tablosundan)
SP ← SP-1
M[SP] ← SR      (Durum saklayıcısı yığına)
IEN ← 0         (Diğer kesmeler maskelendi, bu bayrak SR'nin içindedir)
```

KHP başlangıç adresi

Bundan sonraki komut alma çevriminde MİB KHP'nin ilk komutunu alır, çünkü PC bu komuta işaret etmektedir (PC ← VAD).

Bazı MİB'ler iç saklayıcılarını da yığında saklarlar. Bazıları bu işlemi programcıya bırakır.

Kesme Hizmet Programından (KHP) Dönülürken:

Hatırlatma Kesme hizmet programları özel bir komut ile sonlandırılır; "*return from interrupt*" (RTI).

Bu komut KHP'den kesilen programa dönüşte yapılması gereken işeri gerçekleştirir.

```
SR ← M[SP]      (Durum saklayıcısı geri alındı)
SP ← SP+1
PC ← M[SP]      (Geri dönüş adresi alındı)
SP ← SP+1       (Gerektiği kadar –adres kaç sekizli ise- arttırılır)
```

Kesme çevrimine sadece KHP'den önce girilir:

Dönüşteki işlemler KHP'nin son komutu olan RTI içinde yapılır.

Yorumlar:

Kesme hazırlık ve geri dönüş işlemleri **zaman alıcıdır** (çok sayıda bellek erişimi).

Bu nedenle çok sık gelen kesmeler sistemin performansını düşürür.

Örneğin, çok sık veri aktarımı yapılan uygulamalarda (dosya aktarımı) kesmeli G/Ç yönteminin kullanılması uygun değildir.

Örnek: Kesmeli G/Ç

Problem:

Bir MİB'in komut çevrimi aşağıda süreleri verilen 5 adımdan oluşmaktadır:

1. Komut alma: 60 ns, 2. Komut çözme: 20 ns, 3. Operand alma: 60 ns, 4. Yürütme: 30 ns, 5. Kesme: 200 ns.

MİB'in kesme çevrimindeki hazırlık işlemleri (geri dönüş adresini saklama, vektör adresini alma vb) 200 ns sürmektedir.

G/Ç arabiriminde belleğe 10 adet sözcüğün aktarılması için kesmeli yöntem kullanılmaktadır.

Toplam 500ns süren kesme hizmet programı her çalıştığında sadece bir sözcük aktarmaktadır (Kesmeye gidilirken yapılan hazırlık işlemleri hariç).

MİB programı koşmaya başladığı anda zamanı başlattığımızı varsayalım (Zaman = 0).

MİB ilk komutun alma çevrimindeyken (Zaman = 5ns) G/Ç arabiriminden ilk kesme isteğinin geldiğini varsayınız.

a. G/Ç arabiriminden belleğe ilk sözcük ne zaman aktarılır (Zaman= ?)? Neden?

b. G/Ç arabiriminin her zaman aktarıma hazır olduğu varsayılırsa 10 sözcüğün tamamının aktarımını ne zaman tamamlanır (Zaman= ?) ?

Örnek: Kesmeli G/Ç (devamı)

Çözüm:

Hatırlatma; MİB Komut yürütme çevriminden sonra kesme isteklerini değerlendirir.

Eğer bir kesme isteği kabul edilirse MİB kesme çevrimine girer.

Veri aktarımı KHP'de yapılır.

Geri dönüş işlemleri, KHP'nin bir parçası olan RTI komutunda yapılır.

a. İlk sözcük:

Komut alma + Çözme + Operand + Yürütme + Hazırlık + KHP

Zaman = 60 + 20 + 60 + 30 + 200 + 500 = 870ns

b. 10 sözcük:

Her KHP'de tek sözcük aktarılıyor.

KHP'den ana programa dönülüyor, bir komut yürütülüyor, tekrar KHP'ye gidiliyor.

Zaman = 10 x 870 = 8700ns (Uzun sürüyor. Hazırlık işlemlerinin bedeli yüksek.)

4.5 Sıra Dışı Durumlar (*Exceptions*)

Sıra dışı durumlar bir komuttan veya dış birimlerde oluşan olağan dışı olaylardan kaynaklanabilirler.

Bu durumlarda MİB o anda yürüttüğü programdan bir **sıra dışı durum hizmet programına (*exception handler*)** dallanır ve o durumla ilgili işlemi yürüttükten sonra kaldığı yere geri döner.

Örnek: MC68000

Dış Kaynaklı:
- Reset
- Yol Hatası (*Bus Error* - BERR)
- Kesme İstekleri (*Interrupts*) Vektörlü, otovektörlü

İç Kaynaklı:
- İzleme (*trace*) Adım adım komut yürütme (hata ayıklama için)
- Adres hatası : Tek adreslere word/long erişimi
- Yazılım kesmesi (TRAP 0 -15), TRAPV (*Trap on overflow*), CHK
- Geçersiz komut: Makine dili karşılığı olmayan işlem kodu (*op code*)
- Komut benzetimi (*emulation*) ($A=1010 ve $F=1111 ile başlayan komutlar)
- Yetkili komut çalıştırma
- Sıfıra bölme

Sıra dışı durum oluştuğunda yapılanlar:
- SR → Temp (SR'nin kopyası çıkartılır.)
- S←1, T←0 (İşlemci yönetici (*supervisor*) konumuna geçiyor, izleme kapatılıyor.)
- PC (geri dönüş adresi) yığına yazılır.
- SR nin Temp'teki kopyası yığına (S ve T'nin değişmemiş önceki değerleri) yazılır.
- Sıra dışı hizmet programın adresi vektör tablosundan alınır.
- Saklayıcılar yığında işlemci tarafından saklanmaz.
  Programcı, sıra dışı hizmet programı içinde kullandığı saklayıcıları yığında kendisi korumalıdır.

Geri dönüşte:
- Programcı yığında koruduğu verileri (eğer varsa) kendisi çeker.
- Sıra dışı hizmet programı, RTE (***Return from Exception***) komutu ile sonlandırılır.
  RTE komutu yürütülürken
  - Durum saklayıcısı SR yığından alınır.
  - Geri dönüş adresi yığından alınır.

RESET durumunda bu işlemlerin hepsi yapılmaz.

Bazı sıra dışı durumlarda ise (BERR, kesme gibi) ek işler de yapılır.

4.5.1 Ayrıcalık Konumları (*Privilege Modes*)

68K iki ayrıcalık konumundan birinde bulunur: **yönetici** (*supervisor*) konumu veya **kullanıcı** (*user*) konumu.

Ayrıcalık durumu o anda hangi komutların çalıştırılabileceğini belirler (kısıtlar).

Konum bilgisi işlemcinin FCO (Function Codes Output) çıkışlarında da belirtilir ve gerek duyulduğunda bellek yönetim devresi tarafından belli bellek bölgelerine erişimi denetlemek için kullanılabilir (Yansılar 3.23-24).

İşlemci, konumuna göre iki farklı yığın işaretçisi kullanır: "supervisor stack pointer" (SSP) ve "user stack pointer" (USP).

Yönetici konumu (*Supervisor mode*):

Yönetici konumu daha yüksek önceliklere sahiptir. Sistem programları için kullanılır.

İşlemcinin konumu durum saklayıcısındaki S biti ile belirlenir; S=1 : yönetici

Yönetici konumundayken işlemci tüm komutları yürütebilir.

Kullanıcı Konumu (*User mode*):

Eğer durum saklayıcısındaki S biti sıfırlanırsa işlemci kullanıcı konumuna geçer.

Sistemin çalışmasını etkileyen bazı ayrıcalıklı komutlar sadece yönetici konumunda çalışırlar. Örneğin: STOP, RESET.

Kullanıcı programlarının işlemciyi denetim dışı yönetici konumuna geçirememesi için durum saklayıcısına yazma komutları da kullanıcı konumunda çalışmazlar.

Ayrıcalık durumları arası geçişler

İşlemci kullanıcı konumundayken sadece sıra dışı durum (*exception*) gerçekleşirse işlemci yönetici konumuna geçer.

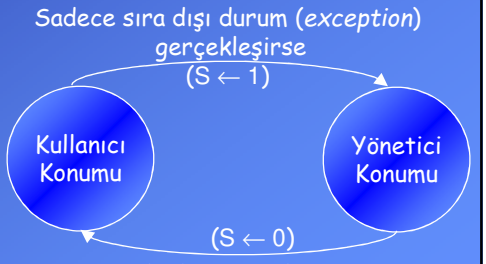
Sıra dışı durum oluştuğunda o andaki S değeri durum saklayıcısının (SR) içinde yığına yazılır ve S←1 yapılarak işlemci yönetici (*supervisor*) konumuna geçirilir.

Bu nedenle hizmet programları yönetici (*supervisor*) konumunda yürütülür.

Eğer sıra dışı durum işlemci kullanıcı (*user*) konumundayken oluştuysa hizmet programından geri dönüldüğünde tekrar kullanıcı konumuna geçilir, çünkü RTE komutu SR'nin orijinal değerini (S bitini içerir) yığından çeker.

Yönetici konumundan kullanıcı konumuna geçmek SR'ye yazan komutlar ile de mümkündür.

MOVE to SR,
ANDI to SR, EORI to SR.



RTE ya da durum saklayıcısına yazan komutlar ile MOVE, ANDI, EORI

4.5.2 Yol Hatası (BERR) ve Adres Hatası:

68000'nin BERR' girişi etkin (lojik 0) yapıldığında **yol hatası** sıra dışı durumu oluşur. Bkz. yansı 3.19: Sonsuz Beklemeyi Önleme

Bir program tek numaralı bir adrese word (16 bit) veya long (32 bit) erişimi yapmak isterse **adres hatası** sıra dışı durumu oluşur.

Adres hatası işlemcinin içinden kaynaklanan yol hatası gibidir.

Kesmelerden farklı olarak o andaki komutun tamamlanması beklenmez.

Yol çevrimi (örneğin bellek erişimi) kesilerek sıra dışı durum işlemlerine geçilir.

Diğer sıra dışı durumlardan farklı olarak yığında daha fazla bilgi saklanır.

BERR, yol hatası veya RESET ile ilgili sıra dışı durum (*exception*) işlemleri yapılırken tekrar BERR oluşursa işlemci durur (HALT konumuna geçer) ve kendini sistemden yalıtır (yüksek empedans).

HALT konumu bellekteki verilerin bozulmasını önler.

Bu konumdan RESET ile çıkılır.

Yol Hatası (BERR) veya Adres Hatası oluştuğunda sistem (yönetici) yığınına (*supervisor stack*) yazılanlar

Bu hatalardan biri oluştuğunda işlemci komut ya da operand alma/yazma işlemlerinden birini yapmaktadır.

Sistemin çalışmasını etkileyen bu hatalar ile daha fazla bilgi alabilmek için diğer sıra dışı durumlardan farklı olarak yığında daha fazla bilgi saklanır.

4.5.3 MC68000'de Kesmeler (*Interrupts*):

68000'nin kesme istekleri için üç bitlik bir girişi vardır (IPL2, IPL1, IPL0).

Bu girişlerden gelen üç bitlik değer (1-7) kesme isteğinin düzeyini belirtir.

Girişlerin hepsi etkisizse kesme isteği yok demektir.

Eğer kesme isteğinin düzeyi SR'deki maskelerin düzeyinden büyükse kesme isteği kabul edilir ve "sıra dışı durum hizmeti" işlemleri (*exception handler*) başlar.

Kesme Maskeleri (I_0 I_1 I_2)

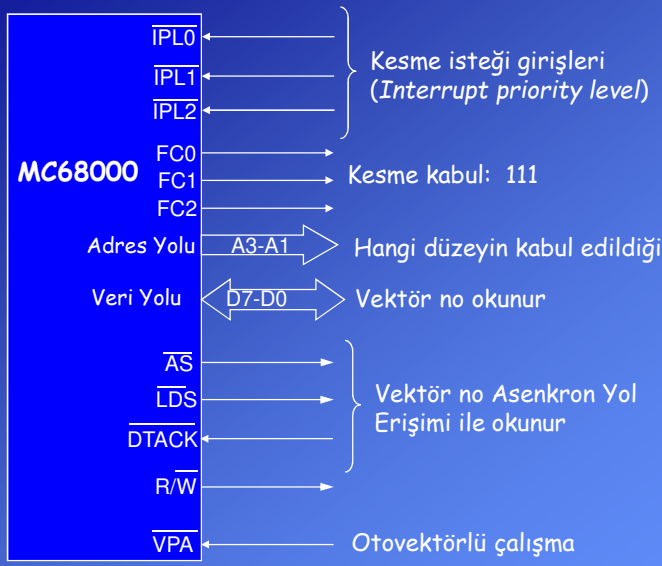
IPL2, IPL1, IPL0 > I2, I1, I0 ise kesme isteği kabul edilir.

IPL2, IPL1, IPL0 ≤ I2, I1, I0 ise kesme isteği kabul edilmez.

Bunun istisnası 7. düzeyden gelen kesmedir.

7. düzeyden gelen kesmeler **maskelenemez**.

MC68000'nin Kesmelerde kullanılan hatları:



68000'de kesme kabul edildiğinde yapılan işlemler:

- SR → Temp (SR'nin kopyası çıkartılır.)
- S ←1, T ←0
- PC yığına yazılır.
- SR'nin Temp'teki kopyası yığına (S ve T'nin değişmemiş önceki değerleri) yazılır.
- I2, I1, I0 ← IPL2, IPL1, IPL0 Kabul edilen düzey maskeye yazılır. Böylece eşit ve (Maske ← Kesme Düzeyi) daha düşük öncelikli kesmeler maskelenmiş olur.
- FC2, FC1, FC0 ← 111 (Kesme Kabul)
- A3, A2, A1 ← Kabul edilen kesme isteğinin düzeyi

a) Vektörlü çalışma:

- Kesmesi kabul edilen cihaz vektör numarasını veri yoluna koyar ve DTACK' hattını etkin yaparak işlemciyi uyarır.
- 68000 asenkron yol erişimi ile 8 bitlik vektör numarasını (D7-D0)'den okur.
- Vektör numarası, hizmet programının başlangıç adresinin vektör tablosunun hangi satırında olduğunu gösterir.
- Vektör tablosunun her satırı 4 sekizli (*byte*) olduğundan vektör tablosunun ilgili satırını bulmak için vektör numarası 4 ile çarpılır (Satır adr. = vno x 4).
- 68000 Vektör tablosu için bakınız yansı 4.6.

68000'de kesme kabul edildiğinde yapılan işlemler (devamı):

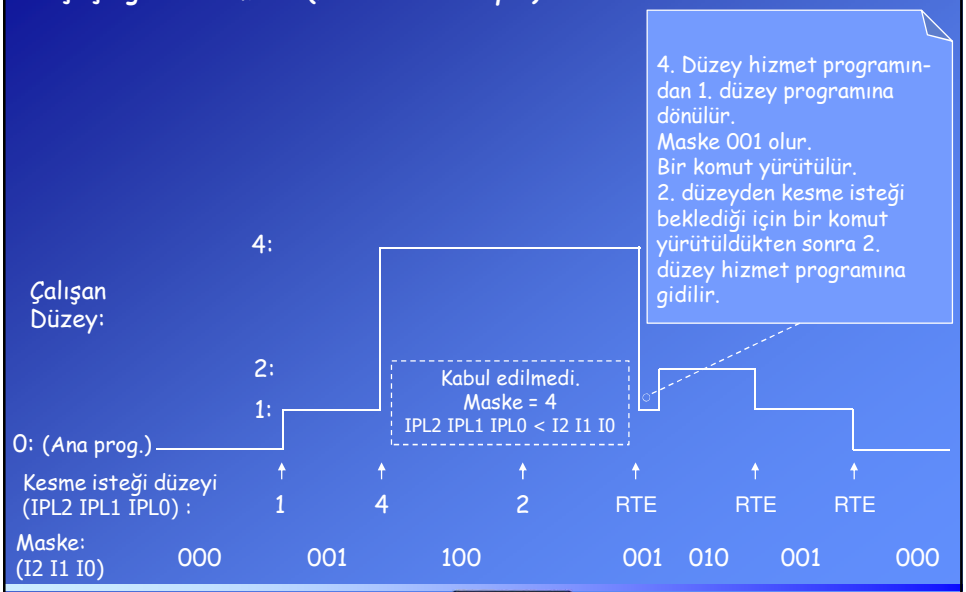
b) Otovektörlü çalışma:

- Kesme üreten birim kesme kabulü sırasında DTACK yerine 68000'in VPA' girişini etkin (sıfır) yaparsa dış birim vektör numarası göndermeyecek demektir.
- Bu durumda 68000 hizmet programının başlangıç adresini kabul edilen düzeye göre vektör tablosunda belirli ve sabit bir yerden kesme alınır.
- 68000'in vektör tablosunda 25-31 numaralı satırlar (7 adet) otovektörlü kesmeler için ayrılmıştır (her düzey için bir satır). (Bkz. 4.6)

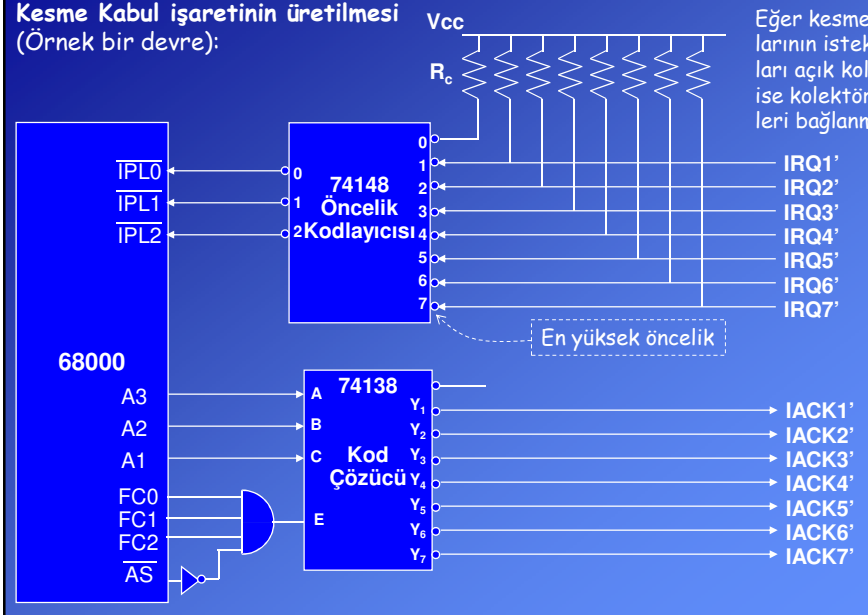
Programcının sorumlulukları:

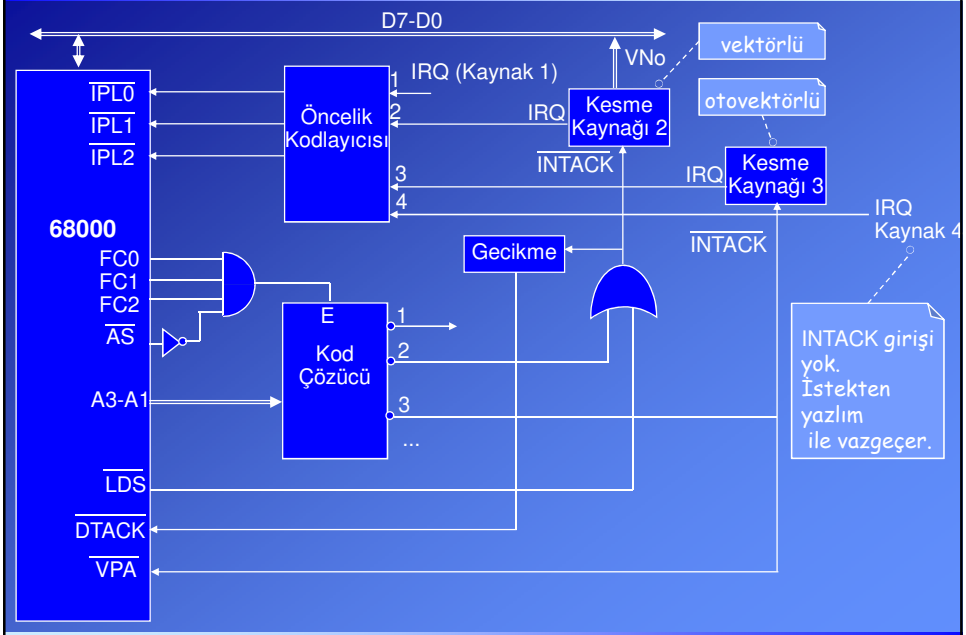
- Kesme hizmet programında kullanılan saklayıcılar yığında saklanmalı.
- Geri dönmeden önce aynı saklayıcılar yığından çekilmeli.
- Bu yığın işlemleri için (MOVEM – "move multiple") komutu kullanılabilir.
- Hizmet programını yazan programcı gerek duyarsa keseme maskelerinin değerini değiştirebilir. Hatırlatma; hizmet programları yönetici konumunda koşar.
- Böylece programcı o andaki hizmet programından daha düşük öncelikli kesmelere izin verebilir veya (7 hariç) daha yüksek düzeydekileri engelleyebilir.
- Kesme hizmet programının sonuna RTE (*return from exception*) komutu yazılmalıdır.

İç içe gelen kesmeler (*Nested Interrupts*):



Kesme Kabul işaretinin üretilmesi (Örnek bir devre):





4.5.4 Yazılım Kesmeleri (*Software Interrupts*)

MC68000'de 16 farklı yazılım kesmesi oluşturmak mümkündür:

TRAP #0 – TRAP #15

Bu komutlar iç kesmeler yaratırlar ve sıra dışı durum (*exception*) işlemlerinin yapılmasını sağlarlar.

Her TRAP komutu için vektör tablosunda bir satır vardır.

Sistemi tasarlayanlar, kullanıcının gerek duyacağı işlemleri (örneğin PIA gibi sistem kaynaklarını kullanmak), yazılım kesmesi hizmet programları şeklinde önceden hazırlarlar.

Kullanıcı programlarını yazanlar ilgili TRAP komutunu çalıştırarak gerek duyduğu sistem kaynaklarına (örneğin G/Ç birimleri) erişebilir.

Hizmet programları yönetici (*supervisor*) konumunda çalıştığından, kullanıcı doğrudan erişemeyeceği sistem kaynaklarını gerek duyduğunda yazılım kesmeleri ile kontrollü olarak kullanmış olur.

Örneğin sistemin tasarımında işlemci kullanıcı konumundayken PIA'ya erişim engellenmiş olur. Bu nedenle kullanıcı programları PIA'nın saklayıcılarına doğrudan erişemez.

Yazılım kesmeleri, kullanıcı programlarından işletim sistemine dönüş için de kullanılır.

4.5.5 Komut Benzetimi (Öykünüm)
(*Instruction Emulation (Unimplemented instructions)*)

MC68000'in makine dilinde $A (1010) ve $F (1111) ile başlayan komut yoktur.

1010 (Line A) ve 1111(Line F) ile başlayan komut kodları (*opcode*) "henüz gerçeklenmemiş komutlar" (*unimplemented*) olarak değerlendirilirler ve sıra dışı durum oluşmasına neden olurlar.

Her iki kodun vektör tablosunda ayrı satırları vardır.

Tasarımcılar kendi makine dili komutlarını bu kodlarla başlayacak şekilde oluşturabilirler ve bunları programlarda diğer komutlar ile kullanabilirler.

Sistemin belleğine bu değerler ile başlayan bir "komut" yerleştirilirse 68000 bu komutu alıp çözmeye çalıştığında sıra dışı bir durum (*exception*) oluşur ve bir hizmet programı çalıştırılır.

İlgili kod için yazılan hizmet programının içinde o komutun yapması gerekenler program ile gerçekleştirilir.

Sıra dışı durum oluştuğunda hizmet programına gidilirken yığına yazılan PC değeri sıra dışı duruma neden olan komutun başına işaret eder.

Örnek:

MC68000 tabanlı bir bilgisayar sisteminde komut benzetimi olanağından yararlanılarak aşağıda açıklanan komutlar gerçekleştirilecektir.

- ADD.B adres1,adres2,adres3 (adres3)←(adres1)+(adres2)

Bu komut adres1'deki ve adres2'deki **8** bitlik sayıları toplayarak sonucu adres3'e yazmaktadır. Adresler 32 bit uzunluğundadır.

- ADD.W adres1,adres2,adres3 (adres3)←(adres1)+(adres2)

Bu komut ise aynı işlemleri **16** bitlik sayılar üzerinde yapmaktadır. Burada da adresler 32 bit uzunluğundadır.

Çözüm:

Önce komutun makine dilindeki yapısını oluşturmak gerekir.

Örnek bir yapı:

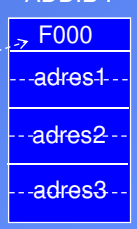
ADD.B adres1,adres2,adres3 $F000 adres1,adres2,adres3

ADD.W adres1,adres2,adres3 $F001 adres1,adres2,adres3

Komut sözcüğünün (*Op word*) son biti boyut (*size*) için kullanılmıştır. 0:B, 1:W

ADD.B : 1111 0000 0000 0000 = $F000

ADD.W: 1111 0000 0000 0001 = $F001



Komutlarda istenen işleri yapacak hizmet programını yazmak gerekir.

Hizmet programından önce onu test etmek için kullanılacak bir ana program aşağıdaki gibi oluşturulur:

```
main  lea     stack,a7                  // Yığın işaretçisi başlangıç değeri
      adda.l  #40,a7                    // Yığın azalan adreslere doğru ilerler
      move.l  #service,($2C)            // Hizmet programının adresi tabloya
      dc.w    $f000,0,$1000,0,$1100,0,$1200      //ADD.B $1000,$1100,$1200
      dc.w    $f001,0,$2000,0,$2100,0,$2200      //ADD.W $2000,$2100,$2200
      ....
      org     $500
stack ds.b    40                        // Yığın için bellekte yer ayrılıyor
```

Line F sıra dışı durumunun vektöror tablosundaki yeri 11. satırdır. Bu satırın adresi ($2C)'dir.

Hizmet programının başlangıç adresinin bu satıra (adrese) yazılması gerekir.

Hizmet programının başında, gerekli saklayıcıları yığına yazmak gerekir.

Bunların hangi saklayıcılar olduğu program yazıldıktan sonra belli olur.

```
service  movem.l d0/a0-a3,-(a7)      D0, A0, A1, A2, A3 yığına yazılıyor
```


Hizmet programına gidilirken SR ve PC mikroişlemci tarafından yığına yazılmıştı.

Hizmet programının başında ise beş adet saklayıcı programla yığına yazıldı.

Buna göre yığındaki bilgiler yandaki gibidir:

```
service movem.l  d0/a0-a3,-(a7)
        movea.l  22(a7),a0      Komuta işaret eden PC → a0
        move.w   (a0)+,d0       Komutun ilk 16 biti OpCode ->d0
        movea.l  (a0)+,a1       Adres1 → a1
        movea.l  (a0)+,a2       Adres2 → a2
        movea.l  (a0)+,a3       Adres3 → a3
        tst.b    d0             B/W?
        bne      word
        move.b   (a1),d0        Byte işlemleri
        add.b    (a2),d0
        move.b   d0,(a3)
        bra      ret
word    move.w   (a1),d0        Word işlemleri
        add.w    (a2),d0
        move.w   d0,(a3)
ret     move.l   a0,22(a7)      Yığındaki PC güncelleniyor (update)
        movem.l  (a7)+,d0/a0-a3
        rte
```

Komut Alma

Komut çözme (*Decoding*)

Operand Alma ve Komut Yürütme

Yığındaki PC bir sonraki komuta işaret ettirilmeli.

SP →	A3_H
+2	A3_L
+4	A2_H
+6	A2_L
+8	A1_H
+10	A1_L
+12	A0_H
+14	A0_L
+16	D0_H
+18	D0_L
+20	SR
+22	PC_H
	PC_L