5. BÖLÜM

KORUMA MEKANİZMASI

Korumalı moda ismini veren koruma mekanizması en önemli konulardan birisidir. Önceki bölümlerde koruma mekanizmasına ilişkin çeşitli ip uçları vermiştik. Ancak bu bölüm tamamen bu konuya ayrılmıştır. Bölüm içerisinde koruma mekanizmasının nasıl sağlandığı birçok açıdan ele alarak incelenmiştir. Karmaşık konuları ele alırken verdiğimiz örneklerin konuyu daha fazla açıklayacağını umuyoruz.

5.1 GİRİŞ

Koruma, çokişlemli işletim sistemlerinin güvenli bir biçimde çalışabilmesi için mutlak gerekli bir mekanizmadır. Koruma mekanizması sayesinde programların kendi alanlarının dışına çıkarak birbirlerinin alanlarına erişmesi engellenmektedir. Koruma mekanizması aynı zamanda sistem programcısının ürettiği böcekler (bugs) konusunda da bilgi verir. Aşağı seviyeli işlemlerle uğraşan bir programcı yanlış bir bölgeye eriştiğinde problemle karşılaşacağından programından şüphelenebilir. Koruma mekanizması piyasaya sürülen böcekli programlardan kullanıcıların olumsuz yönde etkilenmemesini de sağlamaktadır. Koruma segmentlere ya da fiziksel sayfalara uygulanabilir. Biz önce segment düzeyindeki korumayı daha sonra ise sayfa düzeyinde korumayı ele alarak inceleyeceğiz. Giriş çıkış portlarına ilişkin koruma mekanizması Görevlerarası Geçiş'in ele alındığı VI. Bölümde açıklanmaktadır.

5.2 SEGMENT DÜZEYİNDE KORUMA

Segment düzeyinde koruma işlemi çeşitli kontrollerin yapılmasıyla sağlanır. Kitabımızda biz bu kontrolleri farklı saat darbelerinde ve seri olarak yapılıyormuş gibi açıklayacağız. Aslında aynı saat darbesi içinde paralel olarak yapılmaktadır. Bu durum kontrollerin CPU çalışma perrformansını olumsuz bir biçimde engellemektedir.

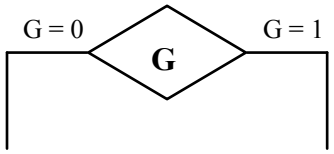
Segment düzeyinde koruma için aşağıdaki kontroller yapılmaktadır:

1. Limit kontrolü
2. Tür kontrolü
3. Öncelik Kontrolü

5.3 LİMİT KONTROLÜ

Mikroişlemci bellek erişim komutlarını çalıştırmadan önce erişilecek olan adresin segment limitleri içerisinde olup olmadığını kontrol eder. Eğer erişilecek adres limit dışındaysa **Genel Koruma Hatası (0D – General Protection Fault)** ile işlemi keser. Limit kontrolünden amaç bir prgramın kendi sınırlarının dışına çıkarak başka bölgelere erişmesini engelleyerek sistem güvenliğini artırmaktır. Bunun yanısıra limit kontrolü programcılar tarafından yapılan gösterici hatalarına karşı bir çeşit debug görevi de görmektedir.

Betimleyici içerisinde, **limit** için 20 ve çözünürlük için 1 bit (**G** biti) ayrıldığını anımsayınız. Limit içerisinde yazılan değerin bir fazlası segment uzunluğunu belirtmektedir. Eğer **G = 1** ise segment sayfa çözünürlüğündedir; bu durumda segment uzunluğu limit içerisinde yazılan değerin bir fazlasının **4096** ile çarpımı kadardır. Eğer **G = 0** ise segment uzunluğu doğrudan limit içerisindeki değerin bir fazlasına eşittir.



Uzunluk = limit + 1 Uzunluk = (limit + 1) * 4096

Segmentlerin en küçük uzunluğunun segment byte çözünürlüğünde ise **1** sayfa çözünürlüğünde ise **4096** olduğuna dikkat ediniz. Limit için betimleyici içerisinde **20** bit ayrıldığına göre eğer segment byte çözünürlüğünde ise en büyük uzunluk **2^{20} * 1 = 1048576 = 1MB**, sayfa çözünürlüğünde ise **2^{20} * 4096 = 4294967296 = 4GB** olacaktır.

	G = 0	**G = 1**
En küçük uzunluk (limit = 0)	1	4096
Erişilecek en küçük offset	0	4095
En büyük uzunluk (limit = FFFFF)	1048576	4294672296
En büyük offset	1048575 (FFFFF)	4294967295 (FFFFFFFF)

Segment sayfa çözünürlüğündeyse doğrusal adresin düşük anlamlı 12 bitinin karşılaştırmaya sokulmasına gerek yoktur; yalnızca yüksek anlamlı 20 bit değerin limit kısmıyla karşılaştırılması yeterlidir. (2^{12} = 4096)

Limit kontrolü yapılırken erişilecek bilginin uzunluğu da önemlidir. Eğer bir byte uzunluğunda bir bilgiye erişilecekse bilginin **limit** ile belirtilen offset adresinde bulunması yeterlidir. Ancak iki byte uzunluğundaki bir bilginin **limit - 1**'de dört byte uzunluğundaki bilginin ise **limit - 3**'de olması gerekir. Örneğin **DS** yazmacına ilişkin betimleyicinin limit kısmında **100_{10}** yazıyor olsun. Bu durumda aşağıdaki ilk komut geçerli olduğu halde diğerleri koruma hatasına yol açar.

```
MOV AL, [100 ]      ; Geçerli!..
MOV AX, [100 ]      ; Koruma hatası!..
MOV EAX, [100 ]     ; Koruma hatası!..
```

Benzer biçimde segment içerisinde atlama yapan **JMP** ve **CALL** komutları ile atlanılan yerdeki makina komutunun uzunluğu limiti aşıyorsa koruma hatası oluşur.

Data / Stack segmentlerine ilişkin betimleyicilerde bulunan **E** biti de **limit** kontrolünde etkili olur. **E = 1** olan segmentlere aşağıya doğru büyüyen segment dendiğini anımsayınız. Aşağıya doğru büyüyen segmentlerde limit kontrolü tersten yapılmaktadır. Bu durum özellikle stack sistemi için düşünülmüştür. Aşağıya doğru büyüyen bir segmentte erişilebilecek bölgeler **(base + limit - 1)**'den **B** bitinin durumuna göre **base + FFFF (B = 0)** ya da **base + FFFFFFFF (B = 1)**'e kadardır.

Toplama işleminde menzil dışına çıkan bitler atılır.

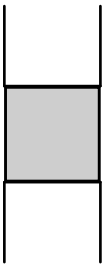
Örneğin aşağıya doğru büyüyen bir segmentin özellikleri şöyle olsun:

- E = 1
- B = 1
- G = 1
- Limit = FFFFF000 (limit kısmının gerçek değeri = FFFFF ve G = 1)
- Base adres = 50000000

Erişilebilecek adreslerlerin en küçük ve en büyük değerleri şunlardır:

En küçük adres değeri : base + (limit - 1) = 50000000 + (FFFFE000 - 1) = 4FFFEFFF

En büyük değeri: base + FFFFFFFF = 4FFFFFFF (menzil dışına çıkan bitlerin atıldığına dikkat ediniz.



4FFFEFFF (base + limit - 1)

4FFFFFFF (base + FFFFFFFF)

Limit değerini küçülttükçe segmentin aşağıya doğru büyüyeceğine dikkat ediniz. Yukarıdaki örnek için limit **FFFFE000** biçiminde azaltılmış olsun. Bu durumda erişilecek adreslerin en büyük ve en küçük değerleri şunlardır:

En küçük adres değeri : base + (limit - 1) = 50000000 + (FFFFE000 - 1) = 4FFFDFFF

En büyük değeri: base + FFFFFFFF = 4FFFFFFF (menzil dışına çıkan bitlerin atıldığına dikkat ediniz.

Bu değerlerin dışına erişilmesi koruma hatasına yol açar. Aşağıya doğru büyüyen segmentlerde **limit** artırıldıkça bellekte daha düşük adreslere erişilebilir. Bu durum özellikle stack segmentlerde stack bölgesini büyütmek amacıyla kullanılmaktadır.

Mikroişlemci segment yazmaçlarına yükleme yapıldığında betimleyici tablolara erişeceği zaman da limit kontrolü yapar. Global betimleyici tablonun limit bilgisi **GDTR** yazmacının, kesme betimleyici tablosunun limit bilgisi ise **IDTR** yazmacının düşük anlamlı 16 bit kısmıdır.

GDTR

TABAN (BASE) (32 bit)	LİMİT (16 bit)

IDTR

TABAN (BASE) (32 bit)	LİMİT (16 bit)

Yerel betimleyici tablo yazamcının yerel betimleyici tablo betimleyicisinin numarasını gösterdiğini anımsayınız. Yerel betimleyici tablonun limiti bu betimleyici içerisinde tutulmaktadır.

YEREL BETİMLEYİCİ TABLO BETİMLEYİCİSİ

SEGMENT BASE 15...0						SEGMENT LİMİT 15...0				
BASE 31...24	G	0	0	AVL	LİMİT 19...16	P	DPL	S	0 0 1 0	BASE 23...16

Limit bilgisinin 64K'dan daha fazla uzayabileceğine dikkat ediniz. Ancak uygulamada 64K'dan daha büyük yerel betimleyici tablolarına rastlanmaz.

Betimleyici tablolarının limit uzunluğu da byte cinsindendir. Limit kısmı için normal olarak 16 bit ayrıldığına dikkat ediniz. Bu durumda tabloların sahip olacağı betimleyici sayısı 65536 / 8 = 8192 ile sınırlıdır. CPU segmentlerde olduğu gibi tablo limitlerinin aşılması durumunda da *Genel Koruma Hatası* ile işlemi keser. **GDT** ve **IDT** uzunlukları genellikle korumalı moda geçmeden önce işletim sisteminin yüklenmesi aşamasında **GDTR** ve **LDTR** yazmaçlarına yükleme yapılarak belirlenmektedir.

LDT'lerin uzunluğu korumalı modda ilgili betimleyicinin değiştirilmesiyle herhangi bir zaman belirlenebilir. **GDTR**, **IDTR** yazmaçlarına yükleme yapan **GIDT** ve **LIDT** makina komutları hem gerçek modda hem de korumalı modda kullanılabilirken, yerel betimleyici tabloya yükleme yapan **LLIDT** yalnızca korumalı modda kullanılabilmektedir. Ayrıca bu makina komutlarını ancak sıfır öncelikli programlar kullanabilirler.

5. 4 TÜR KONTROLÜ

Betimleyicilerin sisteme ilişkin olanlar ve sisteme ilişkin olmayanlar biçiminde iki kısma ayrıldığını anımsayınız. (Bu ayırma işlemi için **S** biti kullanılmaktadır.). Sisteme ilişkin olmayan betimleyiciler (S = 1 olanlar) kendi aralarında **1) Kod 2) Data/Stack** segment betimleyicileri biçiminde yine iki kısma ayrılırlar. Sisteme ilişkin olanların listesini aşağıda bir kez daha veriyoruz:

Tür	**Betimleyici**
0	Reserved
1	Available 80286 TSS
2	LDT
3	Busy 80286 TSS
4	Call Gate
5	Task Gate
6	80286 Interrupt Gate
7	80286 Trap Gate
8	Reserved
9	Available 386/486/Pentium TSS
10	Reserved
11	Busy 386/486/Pentium TSS
12	386/486/Pentium Call Gate
13	Reserved
14	386/486/Pentium Interrupt Gate
15	386/486/Pentium Task Gate

Tür kontrolünü iki aşamada ele alabiliriz:

1. Segment yazmaçlarına yükleme yapıldığı sırada yapılan tür kontrolü. CPU yaptığı kontol sonucunda uygun bir betimleyiciyle karşılaşamazsa *Genel Koruma Hatasıyla* işlemi keser.

- CS yazmacına bir kod segment betimleyicisine ilişkin selektörün yüklenmesi gerekir. Data/stack segment betimleyicileri yüklenemez.

- Okunabilir olmayan bir kod segment betimleyicisine ilişkin bir selektör DS, SS, FS ve GS segment yazmaçlarına yüklenemez. (Kod segment içerisine zaten yazma yapılamıyor; bir de okunamadıktan sonra böyle bir yüklemenin zaten anlamsız olduğuna dikkat ediniz.)

- Yalnızca yazılabilir Data/Stack segment betimleyicilerine ilişkin selektörler SS yazmacına yüklenebilir. (**PUSH** komutu için segmente yazmak gerekir. Yazılabilir olmayan bir segmentin stack amaçlı kullanılmasının bir anlamı olmadığına dikkat ediniz.)

2. Okuma/Yazma iznine göre yapılan tür kontrolü. Data / Stack segmentlerinden her zaman okuma yapılabilir; ancak yazma yapılıp yapılamayacağı betimleyici içerisindeki **W** biti ile belirlenmiştir. Kod segmentlere ise yazma yapılamaz ancak okuma yapılıp yapılamayacağı **R** biti ile belirlenmektedir.

- Kod segment içerisine hiçbir zaman birşey yazılamaz. Yani kod segment betimleyicileri default olarak **read only**'dir. Örneğin aşağıdaki komut her zaman *Genel Koruma Hatasına* yol açar:

MOV CS:[100], AX

Kod içerisne birşey yazılması isteniyorsa yazılabilir eşdeğeri olan bir Data segment betimleyicisi yaratmak gerekir. (Segment alias)

- Yazılabilir olmayan Data/Stack segment betimleyicilerine birşey yazılamaz; ancak Data / Stack betimleyicilerinden her zaman değer okunabilir.

Okunabilir olmayan Kod segment betimleyicilerinden bilgi alınamaz. Örneğin okunabilir olmayan bir kod segment söz konusuysa aşağıdaki komut *Genel Koruma Hatasına* yol açacaktır.

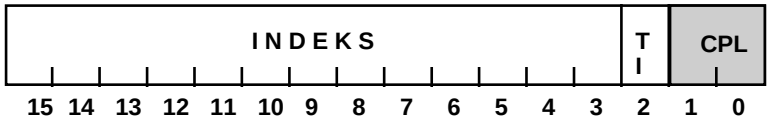
MOV AX, CS:[100]

5.5 ÖNCELİK KONTROLÜ

Koruma mekanizmasının en önemli özelliklerinden birisi öncelik derecesidir (privilege level). Her segment 0, 1, 2, 3 önceliklerinden bir tanesine atanmıştır. Öncelik belirten sayı ne kadar küçükse öncelik o derece yüksektir. Örneğin 1 önceliği 2 önceliğinden daha yüksek bir öncelik düzeyi anlatır. Bir segmentin önceliği betimleyici içerisindeki **DPL** bitleri ile belirlenmektedir. Öncelik derecesi temel olarak işletim sistemine ait olan kodların ve verilerin güvenliğini artırarak güvenliği düşük olan programlardan korunmasını sağlar. Öncelik kontrollerinin nasıl yapıldığını belirtmeden önce bazı kavramları açıklamayı uygun görüyoruz:

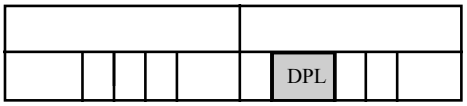
CPL (Current Privilege Level - Geçerli Öncelik Derecesi): Selektörlerin düşük anlamlı 2 bitinin **RPL** (requested privilege level) olarak isimlendirildiğini anımsayınız. **CS** yazmacının **RPL** kısmına **CPL** denilmektedir. **CPL** o anda çalışmakta olan kodun öncelik derecesini belirtir.

CS



DPL (Descriptor privilege level - Betimleyici Öncelik Derecesi): Bir betimleyici içerisindeki **DPL** bitleri segmentin öcelik derecesini belirtir. **DPL** segment yazmaçlarına yükleme sırasında kontrol amacıyla test edilmektedir.

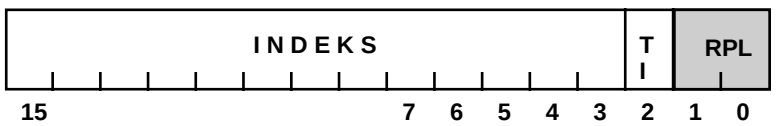
BETİMLEYİCİ



Betimleyici içerisinde **DPL** için iki bit ayrılmıştır. İki bitin toplam 4 öncelik derecesi (0, 1, 2, 3) belirteceğine dikkat ediniz.

RPL (Requested Privile Level): Bir segment yazmacına yüklenecek selektörün düşük anlamlı 2 bitine **RPL** denir.

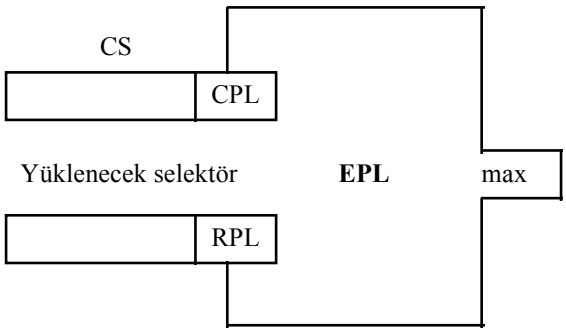
SELEKTÖR



Sistem açısından bir segment yazmacına yüklenecek olan selektöre ilişkin **RPL** değerinin önemi vardır. Bu **RPL** değeri çeşitli biçimlerde test işlemine girer.

EPL (Effective Privilige Level): **CPL** ile segment yazmacına yüklenecek olan selektörün **RPL** değerlerinin en az öncelikli olanına **EPL** denir. **EPL** değerini sayısal olarak aşağıdaki gibi gösterebilirz:

EPL = MAX {CPL, RPL}



5.6 DATA/STACK SEGMENTLERİNE ERİŞİM

Bir program yüklendiğinde segment yazmaçlarına ilkdeğer işletim sistemi tarafından verilir. Programcının başka segmentlere erişebilmesi dolaylı olarak bellekte başka bölgelere erişebilmesi anlamına geleceğine göre bunun belli ölçülerde denetlenmesi gerekmez mi? Örneğin:

```
MOV AX, 1234H
MOV DS, AX
```

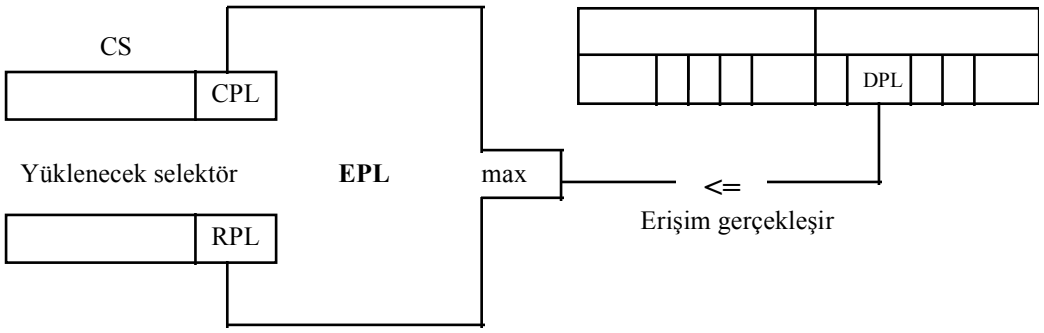
işlemleri ile betimleyici tablodan 1234H ile belirtilen betimleyici yüklenirse, o betimleyici ile belirtilen bölgelere de kolaylıkla erişilebilir. Intel **DS**, **FS**, **GS** ve **SS** yazmaçlarına yükleme yapıldığında bu yüklemenin geçerli olup olmadığını öncelik kontrolü ile test eder. **Kontrol işleminin özeti şudur: Ancak önceliği yüksek programlar önceliği düşük olan segmentlere erişebilirler değilse işlem Genel Koruma Hatası ile kesilir.**

5.6.1 Erişim Kontrolü

DS, **ES, FS**, **GS** segment yazmaçlarına yükleme yapılacağı zaman erişim kontrolü için **CPL**, yüklenecek selektöre ilişkin **RPL** ve yüklenecek segmente ilişkin **DPL** test edilir. Erişimin kabul edilebilmesi için **CPL** ve yüklenecek selektöre ilişkin **RPL** değerlerinin en az öncelikli olanının erişilecek olan segmentin **DPL** değerinden daha öncelikli olması gerekir. Yani sayısal olarak:

max {CPL, RPL} <= DPL

ise erişim gerçekleşir, değilse Genel Koruma Hatası ile işlem kesilir. Öncelik kontrolünü şekilsel olarak da aşağıdaki gibi özetleyebiliriz:



CPL ile **RPL**'nin en büyük değerine **EPL** (Effective Privilege Level) de denilmektedir. Bu durumda erişimin gerçekleşmesi için **EPL <= DPL** gerekliliğini yazabiliriz. Tabi segment yazmaçlarına yükleme yapılabilmesi için erişim kontrollerinin dışında diğer koşulların da sağlanması gerekir. Bu

koşullar yukarıda sıralandı. Gereken koşulların tümünü aşağıda veriyoruz. Koşulların biri bile sağlanmazsa aksi belirtilmediği müddetçe işlem *Genel Koruma Hatası (0D – General Protection Fault)* kesilecektir.

1. Selektörün Global ya da Yerel betimleyici tablosu sınırları içerisinde olması gerekir.
2. Selektöre ilişkin betimleyicinin okunabilir kod segment betimleyicisi ya da Data/Stack segment betimleyicisi olması gerekir.
3. **CPL** ve yüklenecek selektöre ilişkin **RPL'**nin en düşük önceliklisi **DPL**'den daha öncelikli olması gerekir.
4. Yüklenecek segmente ilişkin betimleyicinin **P** biti **1** olmalıdır. Eğer **P = 0** ise *"Segment Bellekte Değil (0B- Segment not in memory)"* kesmesi çağrılır. (Bu durum sanal bellek kullanımının açıklandığı IV. Bölümde ele alınmaktadı.)

SS segment yazmacına yüklemenin yapılabilmesi için **CPL** yüklenecek selektöre ilişkin **RPL** ve yüklenecek segmente ilişkin **DPL** değerlerinin hepsinin aynı olması gerekir. Yani:

CPL = RPL = DPL ise **SS** segment yazmacına yükleme yapılabilir.

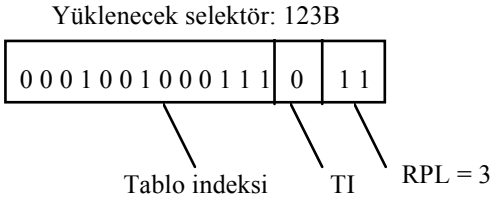
ÖRNEK 5.1

Çalışmakta olan programın öncelik derecesi 0 olsun (CPL = 0). 123BH ile belirtilen Data selektörüne ilişkin betimleyicinin önceliğinin de 3 olduğunu varsalım (DPL = 3). DS yazmacının aşağıdaki gibi yüklenmesi koruma hatasına yol açar mı?

```
MOV   AX, 123BH
MOV   DS, AX
```

ÇÖZÜM

Yüklenecek olan segmente ilişkin selektörün RPL değerini araştıralım.



Şekilden de görüldüğü gibi aranan RPL değeri 3'tür. TI = 0 olduğuna göre betimleyici global betimleyici tabloya ilişkindir. Test işlemine CPL, RPL ve tablo indeksi ile belirtilen yerdeki DPL öncelikleri girer. CPL = 0 ve DPL = 3 olduğu verilmiştir; RPL = 3 olduğunu da yüklenecek selektörü ayrıştırarak biz bulduk. Bu durumda:

max {CPL = 0, RPL = 3} <= {DPL = 3} *koşulunun sağlandığına dikkat ediniz; o halde erişim gerçekleşir. Tablo indeksi ile belirtilen betimleyici mikroişlemcinin tampon yazmaçlarına çekilir. Artık bu segmentin gösterdiği yerlere erişebiliriz.*

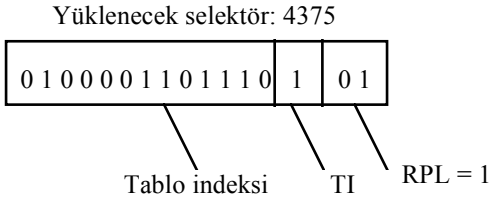
ÖRNEK 5.2

*Çalışmakta olan programın öncelik derecesi **3** olsun (CPL = 3). **4375** ile belirtilen Data selektörüne ilişkin betimleyicinin önceliğinin de 0 olduğunu varsayalım (DPL = 0). **ES** yazmacının aşağıdaki gibi yüklenmesi koruma hatsına yol açar mı?*

```
MOV   AX, 4375H
MOV   ES, AX
```

ÇÖZÜM

Yine önce yüklenecel selektöre ilişkin RPL değerini araştıralım.



RPL değerinin 1 olduğuna dikkat ediniz. TI = 1 olduğuna göre betimleyici Yerel Betimleyici Tablosu'na ilişkindir. Yerel betimleyici tabloya belirtilen indeks değerinde bulunan betimleyicinin DPL biti 0 olarak verilmiş. Bu durumda:

max {CPL = 3, RPL = 1} <= {DPL = 0}

koşulunun sağlanmadığına dikkat ediniz. İşlem Genel Koruma Hatasıyla kesilir.

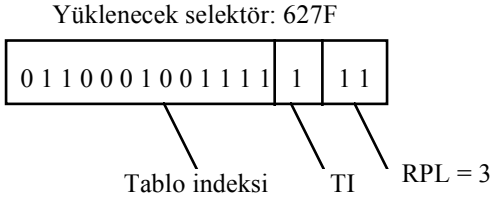
ÖRNEK 5.3

Çalışmakta olan programın öncelik derecesi 3 olsun (CPL = 3). 627FH ile belirtilen Stack selektörüne ilişkin betimleyicinin önceliğinin de 3 olduğunu varsalım (DPL = 3). SS yazmacının aşağıdaki gibi yüklenmesi koruma hatasına yol açar mı?

MOV AX, 627FH
MOV DS, AX

ÇÖZÜM

Yüklenecek selektörün önceliği 3'tür.



TI = 1 olduğuna göre betimleyici Yerel Betimleyici Tablosuna ilişkindir. Bu durumda:

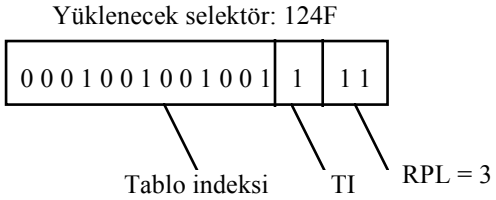
CPL = RPL = DPL = 3 olduğundan erişim gerçekleşir.

5.6.2 RPL Kontrol İşlemine Neden Giriyor?

DS, **ES**, **FS** ve **GS** yazmaçlarına yükleme yapılmak istendiğinde yüklenecek selektöre ilişkin **RPL** değerinin kontrol edilmesi size anlamsız gelebilir. Çünkü **RPL** değeri o anda çalışmakta olan program tarafından tablo indeksi ve **TI** bitleri değiştirilmeksizin ayarlanabilir. Örneğin **CPL = 0** olan bir program **DS** yazmacına **124F** selektörünü yüklemeye çalışsın. **124F** selektörüne ilişkin **DPL** değerinin de **0** olduğunu varsayalım

MOV AX, 124FH
MOV DS, AX

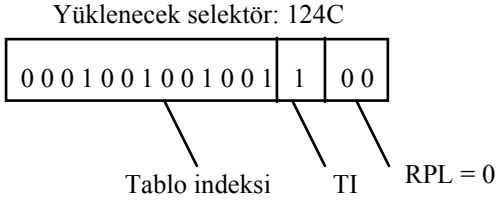
işlemi koruma hatasına yol açar. Çünlü **max {CPL = 0, RPL = 3 } <= DPL** koşulu sağlanmamaktadır. Selektörün parçalarına dikkat edin:



Program tablo indeksini ve TI değerini değiştirmeden 124F selektörünü yeni RPL 0 olacak biçimde ayarlayabilir. Yani aynı selektör şöyle yüklenseydi koruma hatasına yol açmazdı:

```
MOV AX, 124CH
MOV DS, AX
```

Tablo indeksinin ve **TI** bitinin değerleri değişmediğine yalnızca **RPL** değerinin değiştiğine dikkat ediniz.



RPL değerini ayarlamak **ARPL** komutuyla da yapılabilir.

ARPL {Reg/Mem}, Reg_{16}

Intel tasarımcılar gösterici hatalarına karşı koruma mekanizması yoluyla debug olanakları sağlamak ve sistemin güvenliğini biraz daha artırmak amacıyla **RPL** değerini de koruma işlemine dahil etmişlerdir. Böylece eğer dikkatsizlikle ya da bellekteki bilgilerin bozulması sonucunda yanlış bir selektörün yüklenme olasılığı azaltılmaktadır. Örneğin C Programlama dilinde ve sembolik makina dillerinde karşılaşılan gösterici hataları ilk elden işlemci tarafından tespit edilebilir.

Bir noktaya dikkatinizi çekmek istiyoruz: **DS**, **FS**, **GS**, **SS** yazmaçlarına yüklenen **RPL** değerleri bir daha kullanılmamaktadır.

5.7 BAŞKA BİR SEGMENTE DALLANMA

Bir kodun kendinden daha düşük öncelikli kodlara dallanması normal bir durum değildir. Fakat eşit ve daha yüksek öncelikli programlara dallanma birçok durumda istenebilir. Yüksek öncelikli işletim sistemi kodlarının düşük öncelikli uygulama programları tarafından çağrılması bu duruma örnek verilebilir.

Kontrol ancak segmentler arası (uzak) doğrudan ya da dolaylı **jmp** ve **call** makina komutlarıyla başka bir segmente aktarılabilir. Bu durumu 3 biçimde ele alabiliriz

1) Nonconforming kod segmentlere yapılan dallanmalar
2) Conforming kod segmentlere yapılan dallanmalar
3) Kapılar yoluyla yapılan dallanmalar

Önce nonconforming ve conforming kod segmentlere yapılan dallanmaları ele alacağız. Daha sonra kapılar üzerinde durarak kapılar yoluyla yapılan dallanmaları açıklayacağız.

5.7.1 Nonconforming Segmentlere Yapılan Dallanmalar

Kod segment betimleyicisinin **C** biti **0** ise böyle segmentlere nonconforming kod segmentler denir. Bu durumda dallanmanın gerçekleşmesi için programa ilişkin **CPL** değerinin **DPL** değerine eşit olması

ve dallanılacak segmente ilişkin selektörün **RPL** değerinin **CPL**'den küçük ya da ona eşit olması gerekir. Aksi halde işlem *Genel Koruma Hatası* ile kesilir. Daha açık bir ifadeyle dallanmanın gerçekleşmesi için iki koşul vardır.

1. CPL = DPL olmalıdır.
2. RPL <= DPL olmalıdır.

Test işlemi olumlu olarak sonuçlanırsa selektör **CS**'ye yüklenmeden önce **RPL** değeri **CPL** ile değiştirilir. Segmentler arası dallanma sırasında offset değeri için limit kontrolü de yapılacaktır. **JMP** ya da **CALL** komutu ile dallanma arasında bir test farklılığı yoktur.

ÖRNEK 5.3

Çalışmakta olan programın öncelik derecesi 3 olsun (CPL = 3). Program içerisinde öncelik derecesi 3 (DPL = 3) ve limit bilgisi FFFF olan byte çözünürlüğündeki nonconforming bir kod segmente 1234H selektörü ile dallanılmaya çalışılıyor. Aşağıdaki komut sonucu koruma hatası oluşur mu?

JMP FAR 1234: 00000010

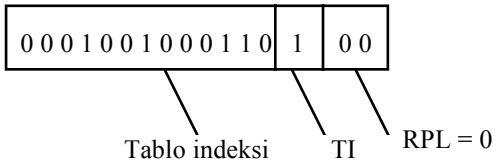
ÇÖZÜM

Test koşullarını inceleyelim.

***1. CPL = DPL** koşulu sağlanıyor.*
***2. RPL <= DPL** koşulu sağlanıyor.*

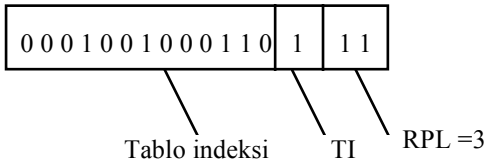
*Her iki koşul da sağlandığına göre dallanma başarılı bir biçimde gerçekleşir. Ayrıca dallanılacak segmente ilişkin **0010** offsetinin de limit içerisinde olduğuna dikkat ediniz. Bir noktayı vurgulamalıyız: Ancak dallanma gerçekleştiğinde **CS** yazmacının içerisindeki değer **1234** olmayacaktır. Çünkü eğer böyle olsaydı **CPL** de değişmiş olurdu. Yüklenecek selektöre ilişkin **RPL** değeri mikroişlemci tarafından gerçek **CPL** ile ayarlandıktan sonra **CS** içerisine yerleştirilecektir*

Yüklenecek selektör: 1234



*Ayarlamadan sonra mikroişlemci tarafından yüklenecek selektöre ilişkin **RPL** değeri **CPL = 3** olduğundan **3** değerine çekilecektir.*

Ayarlamadan sonra: 1237



*Yani **CS** segment yazmacına yüklenecek olan selektör **1234** değil **1237**'dir.*

ÖRNEK 5.4

Çalışmakta olan programın öncelik derecesi 0 olsun (CPL = 0). Program içerisinde öncelik derecesi 3 (DPL = 3) ve limit bilgisi FFFF olan byte çözünürlüğündeki nonconforming bir kod

segmente 2470H selektörü ile dallanılmaya çalışılıyor. Aşağıdaki komut sonucu koruma hatası oluşur mu?

CALL FAR 2470: 00007F00

ÇÖZÜM

Koşulları araştıralım

***1. CPL ≠ DPL** koşulu sağlanmıyor*
***2. RPL <= DPL** koşulu sağlanıyor.*

Birinci koşul sağlanmadığı için işlem Genel Koruma Hatası ile kesilir.

5.7.2 Conforming Segmentlere Yapılan Dallanmalar

Kod segment betimleyicisinin **C** biti **1** ise böyle segmentlere conforming kod segment dendiğini anımsayınız. Conforming kod segmentler daha düşük ya da eşit öncelikli segmentler tarafından çağrılabilirler. Conforming kod segmentlere dallanılabilmesi için **RPL** koşulu aranmaz. Yalnızca **DPL** ile **CPL** karşılaştırılır.

Conforming kod segment DPL <= CPL

Conforming segmentlere yapılan dallanmalarda CPL değeri değişmez. Yani **RPL** ne olursa olsun dallanma sonrasında **CS** yazmacının **CPL** kısmı sabit kalır.

ÖRNEK 5.5

Çalışmakta olan programın öncelik derecesi 3 olsun (CPL = 3). Program içerisinde öncelik derecesi 1 (DPL =1) ve limit bilgisi FFFFF olan sayfa çözünürlüğündeki conforming bir kod segmente 1F15H selektörü ile dallanılmaya çalışılıyor. Aşağıdaki komut sonucu koruma hatası oluşur mu?

CALL 1F15: 50001000

ÇÖZÜM

CPL = 3 ve dallanılacak conforming kod segmente ilişkin **DPL = 1** olduğuna göre

Conforming kod segment DPL <= CPL

dallanma koşulu sağlanır. Ayrıca limit bakımından bir sorun oluşmayacağına da dikkat ediniz. Dallanmadan sonra **CPL** değeri değişmeyecek ve 3 olarak kalacaktır. Bu durumda **CS** içerisndeki selektör **1F17** olarak ayarlanır.

5.8 KAPILAR (GATES)

Bir programın kendinden daha düşük öncelikli segmentlere erişmesi normal ve istenilen bir durum değildir. Ancak tersi faydalı bir durumdur. Örneğin işletim sisteminin aşağı seviyeli fonksiyonları uygulama programlarından daha yüksek öncelikli segmentlerde bulunabilir. Bu durumda sistem çağırmaları yüksek seviyeli segmentlere erişim anlamana gelir. Koruma mekanizması altında genel olarak bir programın kendinden daha yüksek öncelikli segmentlere erişmesi engellendiğine göre Uygulama programlarının kendinden daha öncelikli sistem fonksiyonlarını çağırması iki yolla olabilir.

1) Sistem fonksiyonlarına ilişkin betimleyici conforming kod segmenttir. Böylece herhangi bir zaman uygulama programcısı bu fonksiyonları çağırabilir. Ancak bu çağırma sırasında **CPL** değişmez. Bu durumda çağrılan fonksiyon daha yüksek öncelikli data ve kod segmentlere erişemez. Örneğin 3. Önceliğe sahip bir program 0. Önceliğe sahip conforming bir kod segmente dallansa bile öncelik derecesi yükseltilemediği için 0, 1 ve 2. Öncelikli data segmentler bu kod içerisinde yüklenerek kullanılamazlar.

2) Kapılar yoluyla erişim. Bu durumda program yüksek öncelikli segmentlere işletim sistemi tarafından hazırlanmış özel kapılar yoluyla atlanabilir.

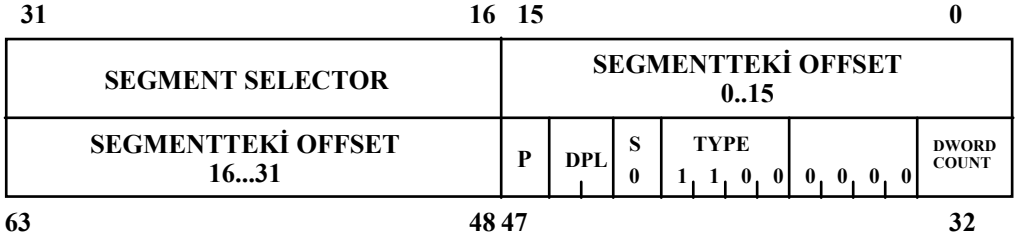
Dört tür kapı vardır:

1) Cağırma kapısı (call gate)
2) Tuzak kapısı (trap gate)
3) Kesme kapısı (interrupt gate)
4) Görev kapısı (task gate)

Çağırma kapısı **CALL** ve **JMP** makina komutuyla kesme ve tuzak kapıları ise **INT** makina komutuyla ya da donanım kesmeleri yoluyla çağrılabilir. Görev kapısı ise **CALL**, **JMP**, ya da **INT** makina komutlarıyla ya da donanım kesmeleriyle çağrılabilmektedir. Daha öncelikli bir fonksiyon kapı yoluyla çağrıldığında çağrılan kodun öncelik derecesi kapının gösterdiği segmentin öncelik derecesine yükseltilir. Örneğin **3** öncelik derecesine sahip olan bir kod bir kapı yoluyla **0** öncelik derecesine sahip bir segmente atlayarak oradaki fonksiyonu **CPL = 0** olacak biçimde çalıştırabilir. Oysa conforming segmente bu biçimde bir çağırma yapılsa çağıran programın öncelik derecesi **CPL = 3** olarak kalacaktır.

Kapı yoluyla daha yüksek öncelikli kodlara dallanmanın koruma açısından stack ile ilgili önemli bir problemi vardır. Eğer çağıran fonksiyonun STACK bölgesi düşük öncelikte kalırsa oluşacak stack problemleri yüksek öncelikli programları ve sistemin bütününü olumsuz yönde etkiler. Örneğin, STACK bölgesinin düşük öncelikte kalması başka düşük öncelikli programların buraya erişebilmesi anlamına gelir ki, çağıran fonksiyonun geri dönüş adresi ve çeşitli parametreler bu bölgede olduğu için bu durum sistemin güvenliğini bozabilir. Ya da işletim sisteminin donanım birimlerini programladığı bir sırada stack ile ilgili bir problem ortaya çıkarsa ve bu problem sırasında bir kesme ile çalışılan programa son verilirse donanımın güvenliği kalmayabilir. Bu nedenlerden dolayı kapı yoluyla yüksek öncelikli kodların çağrılması durumunda stack bölgesinin de yüksek bir önceliğe geçici olarak kopyalanması gerekmektedir. İleride görüleceği gibi her kapı 4 bitlik **COUNT** isimli bir alana sahiptir. Bu alan 32 bitlik stack sözcüklerinden kaç tanesinin yüksek öncelikli stack bölgesine kopyalanacağını belirtir.

Kapıların dört kısma ayrıldığını belirtmiştik. Her kapıda ortak bazı alanlar vardır.Aşağıda örnek bir kapı görüyorsunuz:



Görev kapısının dışında her kapı betimleyicisi içerisinde dallanılacak koda ilişkin bir selektör ve offset bilgisi vardır. Bir kapı çağrıldığında programın akışı selektör ve offset ile belirtilen doğrusal adrese gider.

5.9 KAPILARDAKİ ÖNCELİK KONTROLÜ

Kapı yoluyla dallanma sırasında bir dizi kontrol yapılmaktadır.

1) İşlemci kapının **DPL** alanı ile o anda çalışmakta olan kodun **CPL** alanını ve kapıya ilişkin **RPL** alanını karşılaştırır. Eğer **CPL** ve **RPL**'nin en düşük önceliklisi **, DPL**'ye eşit ya da **DPL**'den öncelikliyse (yani sayısal olarak **MAX {CPL, RPL}** <= **DPL** ise) sonraki aşamaya geçilir; değilse *Genel Koruma Hatası* verilir.

2) Birinci aşamadan başarıyla geçilmişse bu sefer işlemci kapı içerisindeki selektöre ilişkin **DPL** ile o anda çalışmakta olan kodun **CPL**'sini karşılaştırır. Eğer **CPL DPL**'den daha az öncelikli ya da **DPL** ile ya da eşit öncelikliyse (yani sayısal olarak **CPL >= DPL** ise) kesme işlemi onaylanır; değilse *Genel Koruma Hatası* verilir.

Kontrol işlemi iki madde içerisinde aşağıdaki gibi özetlenebilir:

1) **MAX {CPL, RPL} <= DPL (kapıya ilişkin)**
2) **CPL >= DPL (Kapı içerisindeki selektöre ilişkin)**

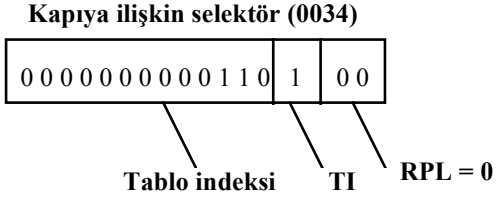
Tabi bu kontrollerin dışında şüphesiz bir de tür ve limit kontrolleri yapılmaktadır. Yani bu kontroller olumlu olsa bile dallanılan segmente ilişkin tür uyuşmazlığı limit yetersizligi *Genel Koruma Hatası'*na yol açar. Kapı yoluyla dallanmada offset bilgisi kullanılmaz. Önemli olan selektör bilgisidir. Gerçek offset değeri komut içerisinde değil kapı içerisinde belirtilmektedir.

ÖRNEK 5.6

CALL 0034:00000000 *makina komutu ile çağırma kapısı yoluyla bir segmente dallanma yapılmak isteniyor. O anda çalışmakta olan programın* ***CPL*** *değeri* ***3****'tür. Kapıya ilişkin* ***DPL = 3*** *ve kapı içersindeki dallanılacak segmente ilişkin* ***DPL = 0*** *ise dallanma gerçekleşir mi?*

ÇÖZÜM

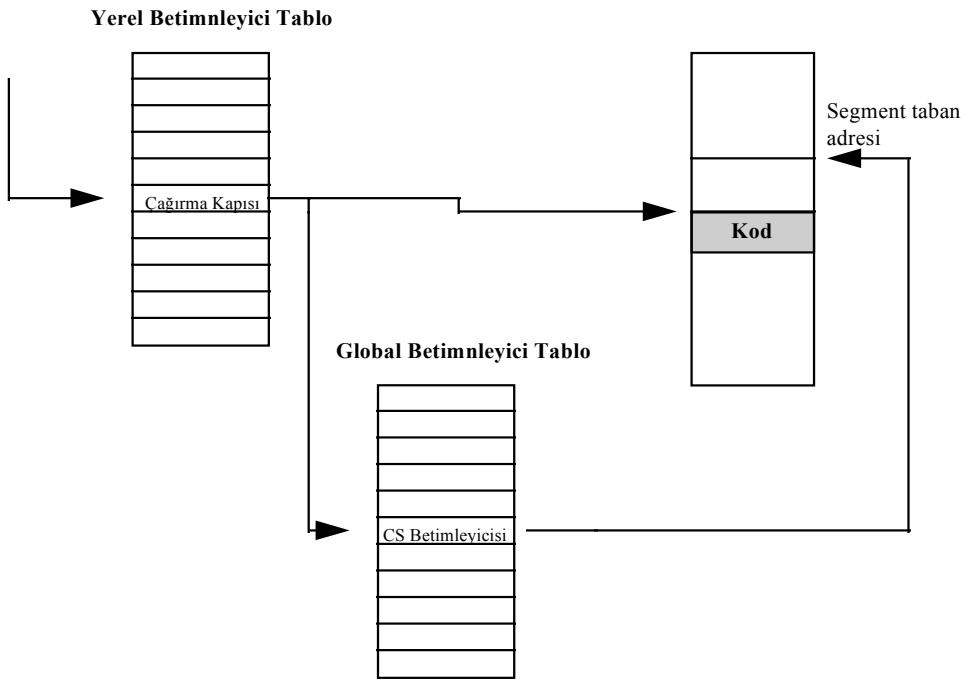
Dallanma yapılan çağırma kapısına ilişkin ***RPL*** *değerini bulabilmek için selektörü bit bit ayrıştıralım.*



Kapının bulunduğu tablo yerel betimleyici tablodur. ***RPL = 0*** *olduğunu görüyorsunuz.*

MAX {CPL, RPL} = MAX {3, 0} = 3*'tür. Bu değer kapıya ilişkin* ***DPL*** *değerine eşittir. Ayrıca* ***CPL = 3*** *dallanılacak segmente ilişkin* ***DPL = 0*** *değerinden daha az öncelikli olduğundan (yani sayısal olarak daha büyük olduğundan) dallanma gerçekleşir. Programın* ***CPL*** *değeri dallanılacak segmentin öncelik derecesine (burada 0) yükseltilir.*

Çağırma, kesme ve tuzak kapılarındaki selektör Yerel ya da Global Betimleyici Tabloya ilişikn olabilir. Örneğin kapı Yerel Betimleyici Tablo içerisinde olduğu halde Kapı içerisindeki selektör Global Betimleyici Tablo içerisinde bulunabilir:



Kapıya dallanma işlemi **CALL** yerine **JMP** komutu ile yapılırsa ya da kapı içerindeki selektöre ilişkin betimleyici conforming kod segment betimleyicisi ise öncelik yükseltilmesi yapılmaz. Daha açık olarak:

1) Düşük öncelikli bir program conforming kod segmente kapı yoluyla **CALL** komutuyla dallansa bile öncelik yükseltilmesi yapılmaz.

2) JMP komutu ile kapı yoluyla yapılan dallanmalarda öncelik yükseltilmesi yapılmaz.
Sonuç olarak **JMP** ile conforming olmayan kod segmente çağırma kapısı yoluyla atlanması sırasında şu kontroller yapılmaktadır:

1) Kapıya ilişkin **DPL** alanı o anda çalışmakta olan kodun **CPL** alanı ve kapıya ilişkin **RPL** alanı ile karşılaştırılır. Eğer **CPL** ve **RPL**'nin en düşük önceliklisi **, DPL**'ye eşit ya da **DPL'**den öncelikliyse (yani sayısal olarak **MAX {CPL, RPL}** <= **DPL** ise) sonraki aşamaya geçilir; değilse *Genel Koruma Hatası* verilir.

2. CPL ile dallanılacak segmente ilişkin **DPL** değerinin eşit olması gerekir. (Yani DPL (dallanılacak koda ilişkin) = CPL). Yoksa *Genel Koruma Hatası* verilir.

Conforming kod segmente **CALL** ya da **JMP** komutları ile yapılan dallanmalar sırasında da iki kontrol yapılmaktadır:

1) İlk ontrol diğerleriyle aynıdır. Yani yapıya ilişkin **DPL** alanı o anda çalışmakta olan kodun **CPL** alanı ve kapıya ilişkin **RPL** alanı ile karşılaştırılır. Eğer **CPL** ve **RPL**'nin en düşük önceliklisi **, DPL**'ye eşit ya da **DPL'**den öncelikliyse (yani sayısal olarak **MAX {CPL, RPL}** <= **DPL** ise) sonraki aşamaya geçilir; değilse *Genel Koruma Hatası* verilir.

2. CPL değerinin dallanılacak segmente ilişkin **DPL** değerinden daha az öncelikli ya da ona eşit öncelikli olması gerekir. (Sayısal olarak DPL <= CPL olmalıdır). Bu koşul da sağlanırsa dallanma yapılır. Ancak **CPL** değeri dallanılacak kodun **DPL** değerine yükseltilmez. Bunun da doğrudan conforming kod segmente dallanmadan bir farkı olmadığından kapı kullanmanın da anlamı kalmaz.

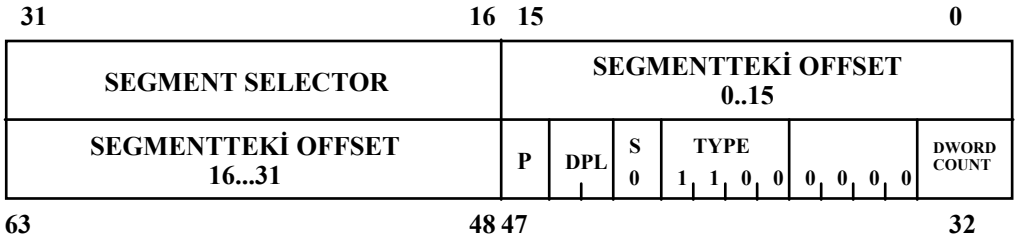
Görüldüğü gibi **JMP** komutu ile gerçek anlamda öncelik derecesi yükseltilerek dallanma yapılamamaktadır. Bu yüzden uygulamada çağırma kapısı ile dallanma CALL makina komutuyla yapılır.

Not: Bu bölümde yalnızca çağırma kapısı açıklanacaktır. Tuzak ve Kesme kapıları Kesme işlemlerinin anlatıldığı VII. Bölümde Görev kapısı ise Çokişlemli çalışmanın anlatıldığı VI. Bölümde ele alınmaktadır.

5.10 ÇAĞIRMA KAPISI (CALL GATE)

Çağırma kapısı Global Betimleyici Tabloda ya da Yerel Betimleyici tabloda bulunabilir ve segmentler arası çağırma (far call) ya da atlama (far jmp) komutuyla kullanılabilir. Çağırma kapısı 16 bit ve 32 bit olmak üzere iki kısma ayrılmaktadır. 16 bit çağırma kapısı 80286'lar zamanında kullanılıyordu. 32 bit çağırma kapıları sonraki modellerde kullanılmaktadır. Aşağıda 32 bit çağırma kapısını görüyorsunuz:

ÇAĞIRMA KAPISI



Çağırma kapısı alanlarını tek tek inceleyelim:

Segment Selector: Atlanılacak segmente ilişkin selektördür. Bu segment Yerel Betimleyici Tabloda ya da Global Betimleyici Tabloda bulunabilir. Buradaki selektörün kod segment betimleyicisine ilişkin olması gerekir.

Segmentteki Offset: Dallanmanın yapılaacağı segmentin offseti. Programın akışı segment selektörü ile belirlenen segmentin burada belirlenen offsetine atlar.

Atlanılacak kodun adresi = Segment selektörü : Segmentteki offset

ile belirlenmektedir.

Dword Count: Çağıran programın stack bölgesinden kaç tane 32 bitlik (double word) bilgi kopyalanacağını gösterir. Bu durum ileride ele alınmaktadır.

S biti: Çağırma kapısı sisteme ilişkin bir kapı olduğundan **S = 0** olmak zorundadır.

TYPE: 32 bit çağırma kapısına ilişkin olduğundan burada **1 1 0 0** olmak zorundadır.

DPL: Çağırma kapısının öncelik derecesidir. Bu öncelik derecesi koruma mekanizması için kontrol işlemine sokulur.

P biti: Çağırma kapısına ilişkin segmentin bellekte olup olmadığını anlatır.

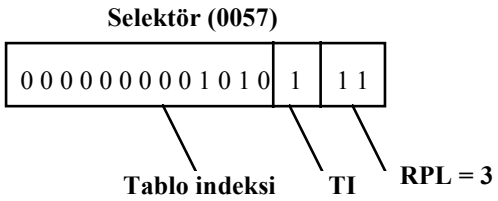
Çağırma kapıları **far CALL** ya da **far JMP** makina komutlarıyla çağrılabilir **JMP** makina komutuyla çağırmada öncelik derecesi yükseltilmez. Gerçek işlevsel çağırma **CALL** komutuyla yapılmaktadır. **JMP** komutu ile kapı çağırması ile normal segment çağırması arasında bir fark yoktur. Bu durumu ileride ele alacağız. Biz şimdi **far CALL** ile atlanılan çağırma kapıları üzerinde durmak istiyoruz.

Öncelikle çağırma kapısının makina komutu olarak kullanılma biçimi üzerinde durmak istiyoruz. **CALL** makina komutuyla segmentler arası atlama sırasında atlanılacak offset adresi çağırma kapısının içerisinde bulunduğundan çağırma sırasındaki offset mikroişlemci tarafından kullanılmaz. Yani bu offset herhangi bir değerde olabilir.

Örneğin:

CALL 0057:00000000

gibi bir çağırma sırasında mikroişlemci önce **0057** selektörüne ilişkin segmentin türünü belirlemek zorundadır. Selektörü bit bit ayrıştıralım:



TI = 1 olduğuna göre betimleyici Yerel Betimleyici Tablo içerisindedir. Tablo indeksi = 1010 = $10_{10.}$ İşlemci Yerel Betimleyici Tablosunun **10** numaralı betimleyicisini okur. Eğer bu betimleyicide **S = 0** ve **TYPE** alanı da **1 1 0 0** biçimindeyse mikroişlemci bunun bir çağırma kapısı olduğuna karar verir. Artık komuttaki offset adresinin ne olduğu hiç önemli değildir. Mikroişlemci bu offset adresini kullanmayacaktır. (Gerçek offsetin kapı içerisinde olduğunu anımsayınız.)

5.11 OTOMATİK STACK DEĞİŞİMİ

Bildiğiniz gibi normal olarak **far CALL** makina komutu **CS** ve **EIP** yazmaçlarını stack bölgesine atarak sakladıktan sonra ilgili fonksiyona dallanır. Fonksiyondan dönüş **RETF** makina komutuyla yapılmaktadır. **RETF** sırasıyla **EIP** ve **CS** yazmaçlarını stack bölgesinden alarak eski değerlerle yükler.

Kapı yoluyla yüksek öncelikli bir fonksiyona dallanıldığında stack bölgesinin düşük öncelikte kalması yukarıda da değindiğimiz gibi bazı problemlerin ortaya çıkmasına yol açabilir. Bu problemlerin kaynağı iki türlü olabilir.

1) Stack bölgesi düşük öncelikli olarak kalırsa düşük öncelikli programlar buraya erişebilir. Stack bölgesinin başka programlar tarafından bozulması dallanılan yüksek öncelikli programların çalışmasını da bozabilir. Yüksek öncelikli programlar çeşitli donanım birimlerini programlamış olabilirler. Bu durumda yüksek öncelikli programların çalışmasının bozulması sistemi olumsuz yönde etkileyebilir.

2) Stack taşması yoluyla çağrılacak bir içsel kesme yüksek öncelikli programın çalışmasını durdurabilir. Bu durumda da sistemin güvenliği bozulacaktır.

Yukarıda açıkladığımız iki tür hata kaynağının ortadan kaldırılması ancak kapı yolula yüksek öncelikli kodlara dallanma sırasında stack bilgilerinin de geçici olarak yüksek öncelikli bir bölgeye otomatik olarak taşınmasıyla mümkün olabilir.

Düşük öncelikli bir program çağırma kapısı kullanarak yüksek öncelikli bir programa dallandığında mikroişlemci otomatik olarak geri dönüş bilgileri ve çağrılacak fonksiyonun kullanacağı parametre bilgileri için yüksek öncelikli bir stack alanı yaratır ve değerleri oraya taşır. Bu işlem için o anda TR yazmacı ile belirtilen Görev Durum Segment (Task State Segment) bölgesi kullanılmaktadır. (TR ve Görev Durum Segmentlerinin işlevi sonraki bölümde ele alınmaktadır.). Yüksek öncelikli programlara kapı yoluyla dallanırken işlemci yaratacağı stack için **SS** ve **ESP** değerlerini görev durum segmenti içerisindeki SS0 – ESP0, SS1 – ESP1, SS2-ESP2 alanlarından elde eder. İşlemci yükseltilecek öncelik hangisiyse o alanın **SSn** ve **ESPn** bölgelerine bakmaktadır. (Bu açıklamalarda altprogram yerine C programlama dilinde kullanılan fonksiyon terimini tercih ediyoruz.)

Çağırma kapısı yoluyla yüksek bir önceliğe dallanıldığında sırasıyla şunlar olur:

1) Çağırma kapısına ilişkin **CALL** komutundan çağırma işini yapan fonksiyon çeşitli parametreleri kendi stack bölgesine atar.

2) Görev durum segmenti içerisinde belitrilen **SSn** ve **ESPn** alanlarıyla beliritlen stack bölgesi stack değişimi için çağıran fonskiyonun en azından **SS** ve **ESP** ve **CS:EIP** değerlerini alacak büyüklükte olmalıdır. Aksi takdirde işlem **"Stack taşması (0B - stack exception)"** kesmesiyle durdurulur.

3) Çağıran programın **SS** ve **ESP** değerleri yüksek öncelikli stack bölgesine kopyalanır.

4) Kapının **COUNT** kısmında belirtilen sayıda DWORD stack bilgisi yeni alana kopyalanır. (Bu alanda sıfır sayısı da olabilir. Bu durum hiçbir bilginin kopyalanmayacağını göstermektedir.)

5) Çağıran fonksiyonda sonraki komuta ilişkin **CS** ve **EIP** değerleri yeni stack bölgesine kopyalanır.

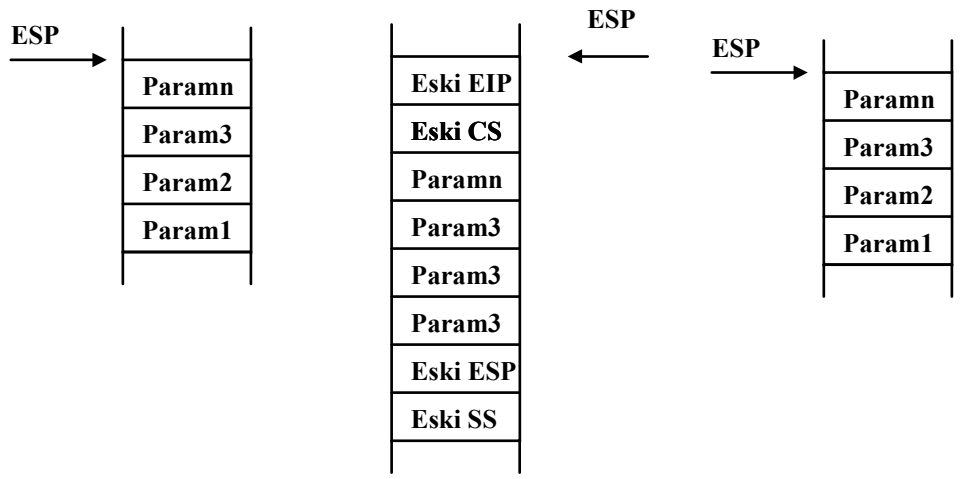
6) İşlemci kapıda belirtilen selektör ve offseti sırasıyla **CS** ve **EIP** yazmaçlarına yükler.

7) Kapı yolula fonksiyon çağrılır.

8) RETF komutuyla karşılaşıldığında yüksek öncelikli bölgede saklanmış olan değerlerle **CS : EIP** ve **SS : ESP** yazmaçlarını yükler.

9) İşlemci sonraki komuttan çalışmaya devam eder.

Aşağıda çağıran fonksiyonun eski stack bölgesii ve kapı yoluyla çağıran fonksiyonun yeni stack bölgesi çizilmiştir; inceleyiniz:



5. 12 SAYFA DÜZEYİNDE KORUMA

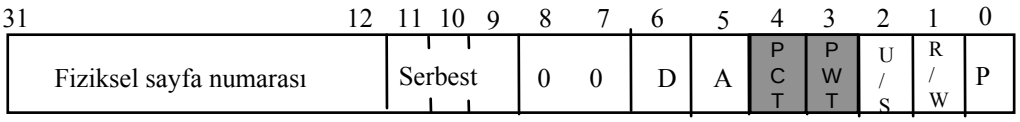
Segment düzeyinde yapılan korumaların dışında sayfa düzeyinde de koruma mekanizması sözkonusudur. Sayfalara ilişkin koruma özellikleri III. Bölümde ele alınmıştı. Bu bölümde bunları özellikle koruma mekanizması içerisinde bir kez daha ele alacağız.

Sayfa düzeyinde koruma iki biçimde yapılabilir.

1) Erişim kontrolleri

2) Tür kontrolü

Erişim kontrolü sayfaya erişen programın öncelik derecesiyle **U/S** bitlerine bağlıdır. **U/S = 0** ise sayfa sistem programcısı modundadır ve yalnızca **CPL = 0** olan programlar bu sayfalara erişebilirler.
U/S = 1 ise sayfa uygulama programcısı modundadır. **CPL = 1, 2, 3** olan programlar erişebilirler.
U/S = 1 ise sistem programcısı her zaman sayfaya erişebilir.



Tür kontrolü sayfaya yazma ya da okuma işleminin sınırlandırılması için kullanılmaktadır. Bu işlem sayfa içerisindeki **R/W** bitleriyle yapılmaktadır. **R/W = 0** ise sayfadan yazlnızca okuma ; R/W ise sayfaya hem okuma hem yazma yapılabilir.

Normal olarak sistem programcısının **R/W** bitlerinden etkilenmez. Ancak sistem programcısını sınırlamak için 486 ve Pentium işlemcilerinde **CR0** yazmacı içerisindeki **WP (write protect)** biti kullanılmaktadır. **WP = 1** ise sistem programcısı **R/W = 0** durumunda sayfaya yazma yapamaz. Aşağıdaki tabloda tüm erişim olasılıklarını görüyorsunuz:

U/S	R/W	WP	Uyulama Programcısı (CPL= 3)	Sistem Programcı (CPL = 0, 1,2)
0	0	0	Erişemez	Okur / Yazar
0	1	0	Erişemez	Okur / Yazar
1	0	0	Yalnız okur	Okur / Yazar
1	1	0	Okur / Yazar	Okur / Yazar
0	0	1	Erişemez	Yalnız okur
0	1	1	Erişemez	Okur / Yazar
1	0	1	Yalnız okur	Yalnız okur
1	1	1	Okur / Yazar	Okur / Yazar

Ayrıntılı bilgi için III. Bölüme başvurabilirisiniz.