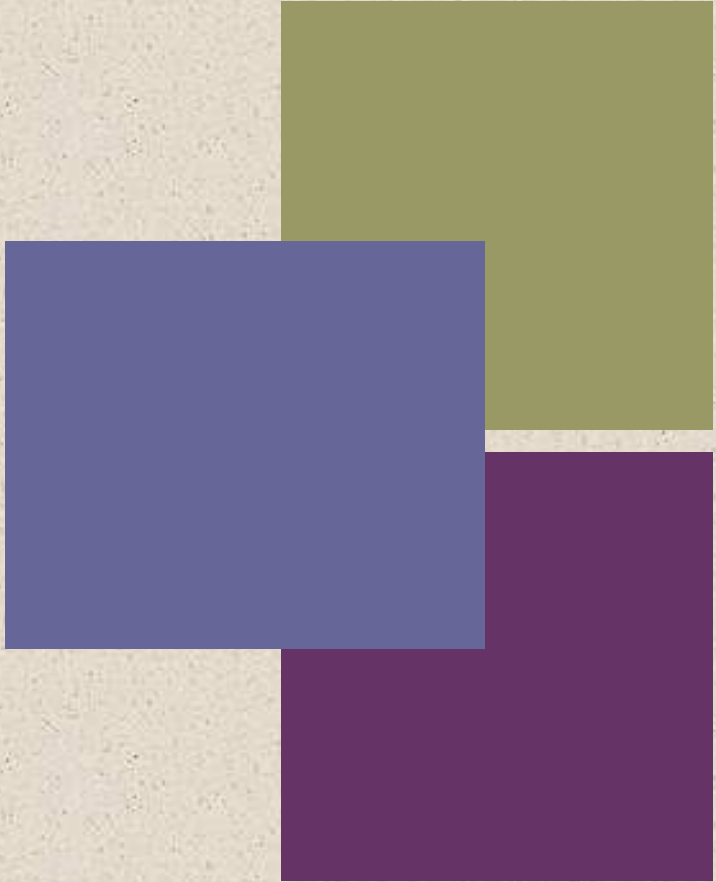


William Stallings Computer Organization and Architecture 10th Edition

Orijinal slaytların
çevirisidir.

Bu bölüm, işlemcinin Chapter 3'te ele alınmayan yönlerini tartışır ve sonraki bölümlerde RISC ve superscalar mimarinin tartışılması için zemin hazırlar.

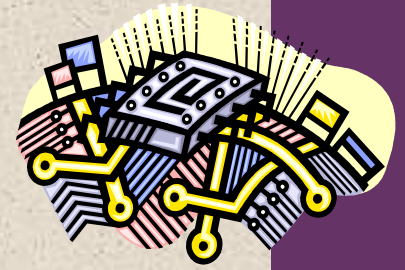


+ Chapter 14

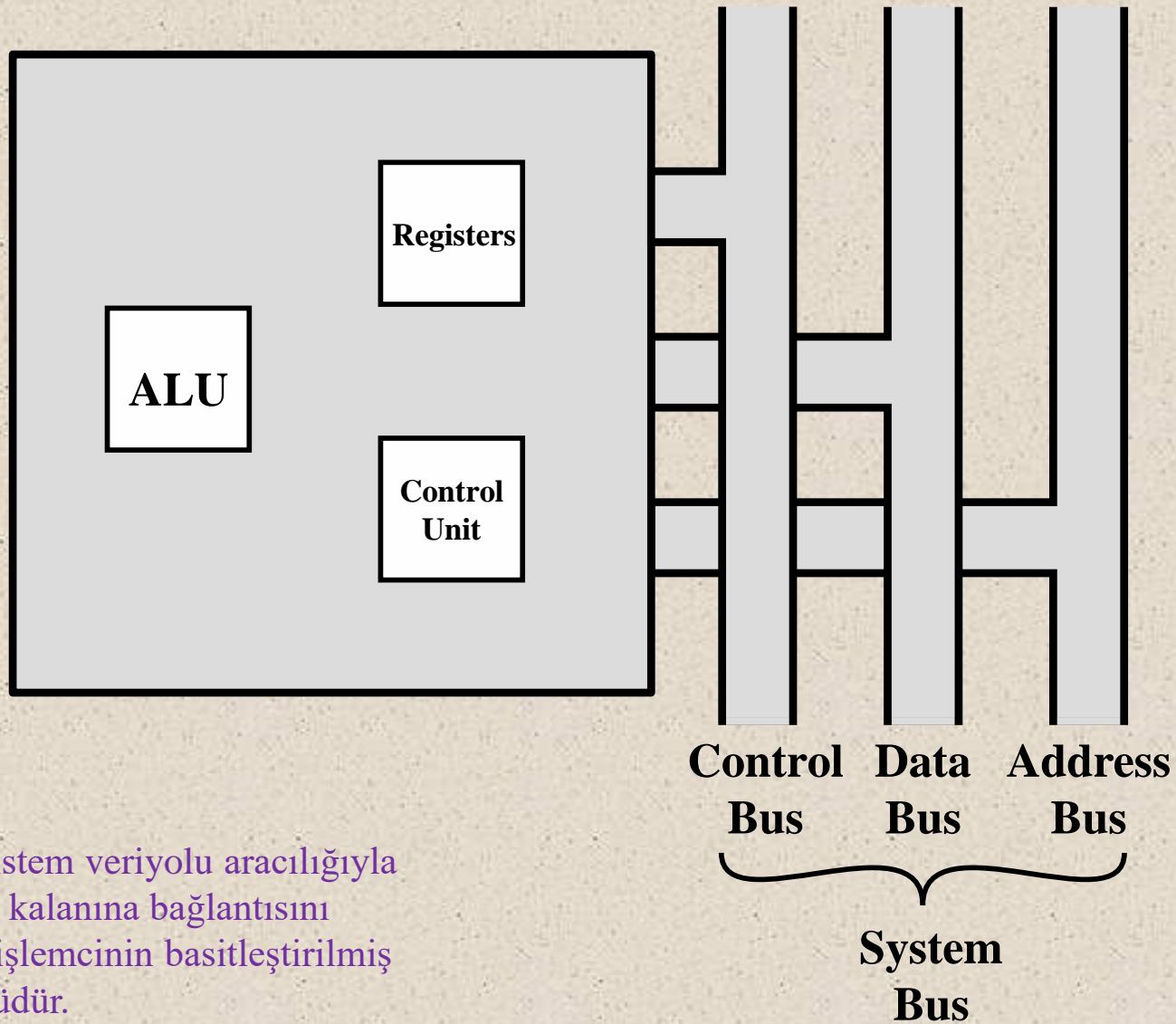
Processor Structure and Function

+ Processor Organization

Processor Requirements:

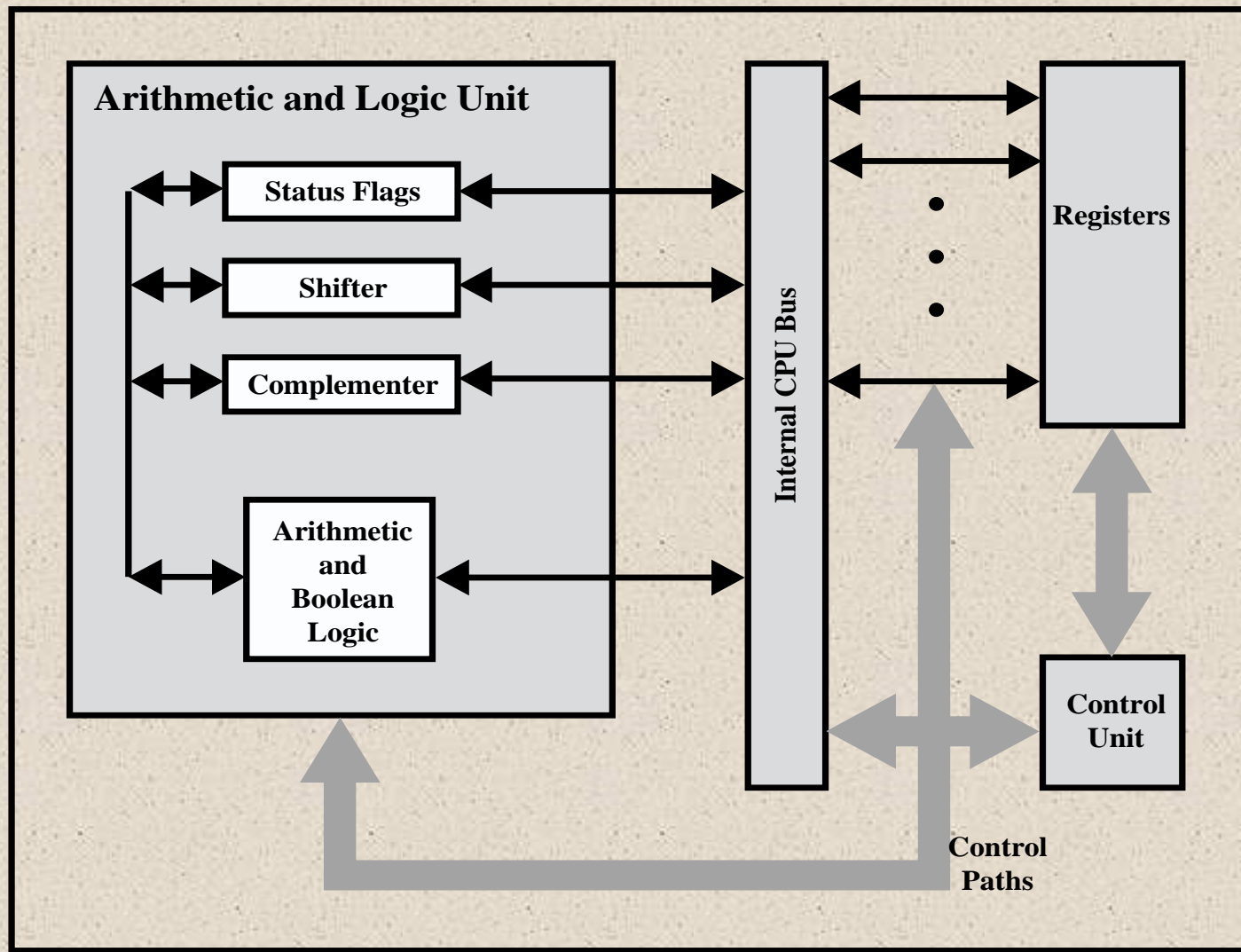


- Fetch instruction
 - İşlemci bellekten bir komut okur (register, cache, main memory)
- Interpret instruction
 - Hangi eylemin gerekli olduğunu belirlemek için komutun kodu çözülür
- Fetch data
 - Bir komutun yürütülmesi, bellekten veya bir I/O modülünden veri okumayı gerektirebilir.
- Process data
 - Bir komutun yürütülmesi, veriler üzerinde bazı aritmetik veya mantıksal işlemlerin gerçekleştirilmesini gerektirebilir.
- Write data
 - Bir uygulamanın sonuçları, verilerin belleğe veya bir I/O modülüne yazılmasını gerektirebilir.
- Bunları yapmak için işlemcinin bazı verileri geçici olarak saklaması ve bu nedenle küçük bir dahili belleğe ihtiyacı vardır.



Şekil 14.1, sistem veriyolu aracılığıyla sistemin geri kalanına bağlantısını gösteren bir işlemcinin basitleştirilmiş bir görünümüdür.

Figure 14.1 The CPU with the System Bus



Şekil 14.2, işlemcinin biraz daha ayrıntılı bir görünümü

Figure 14.2 Internal Structure of the CPU

veri transfer ve kontrol yolları «*internal processor bus*» adlı eleman gösterilmiştir. Bu eleman, çeşitli registerlar ve ALU arasında veri aktarımı için gereklidir çünkü ALU aslında yalnızca dahili işlemci belleğindeki veriler üzerinde çalışır.



Register Organization



- İşlemcinin içinde, hiyerarşide ana belleğin ve ön belleğin üzerinde bir bellek seviyesi olarak işlev gören bir dizi register vardır. İşlemcideki registerlar iki rol oynar:

User-Visible Registers

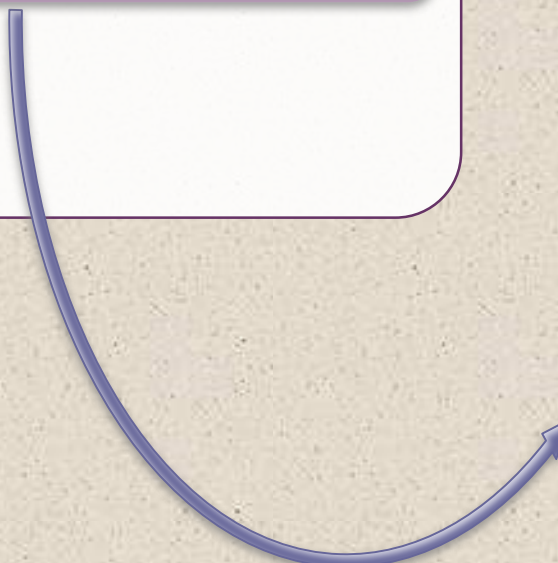
- Registerların kullanımını optimize ederek ana bellek referanslarını en aza indirmek için makine veya assembly dili programcısının işini kolaylaştırır

Control and Status Registers

- Kontrol ünitesi tarafından işlemcinin çalışmasını kontrol etmek için ve ayrıcalıklı (*privileged*) işletim sistemi programları tarafından programların yürütülmesini kontrol etmek için kullanılır

User-Visible Registers

İşlemcinin
yürüttüğü/icra ettiği
makine dili aracılığıyla
başvurulur



Categories:

- **General purpose**
 - Programcı tarafından çeşitli işlemlere atanabilir
- **Data**
 - Yalnızca verileri tutmak için kullanılabilir ve bir operand adresinin hesaplanmasında kullanılamaz
- **Address**
 - Bazen genel amaçlı da kullanılabilir veya belirli bir adresleme moduna adanabilir
 - Examples: segment pointers, index registers, stack pointer
- **Condition codes**
 - Bayrak (*flags*) olarak da anılır
 - İşlemler sonucunda işlemci donanımı tarafından belirlenen bitler

Table 14.1

Condition Codes

IA-64 mimarisine ve MIPS işlemcilere dayalı olanlar da dahil olmak üzere birçok işlemci, koşul kodlarını hiç kullanmaz. Daha ziyade, koşullu dallanma komutları, bir koşul kodunu kaydetmeden, yapılacak bir karşılaştırmayı belirtir ve karşılaştırmının sonucuna göre hareket eder.

Advantages	Disadvantages
Koşul kodları, normal aritmetik ve veri transferi komutlarıyla ayarlandığından, gereken COMPARE ve TEST komutlarının sayısını azaltmalıdır.	Koşul kodları hem donanıma hem de yazılıma karmaşıklık katar. Koşul kodu bitleri genellikle farklı komutlarla farklı şekillerde modifiye edilerek hem mikro programlayıcı hem de derleyiciyi yazan programcı için hayatı zorlaştırır.
DALLANMA («BRANCH») gibi koşullu komutlar, «TEST AND BRANCH» gibi bileşik komutlara göre basitleştirilmiştir.	Koşul kodları düzensizdir; bunlar tipik olarak ana veri yolunun parçası değildirler, bu nedenle ekstra donanım bağlantıları gerektirirler.
Koşul kodları, çok yollu dallanmaları (<i>multiway branches</i>) kolaylaştırır. Örneğin, bir TEST komutunun ardından biri sıfırdan küçük veya sıfıra eşit ve diğeri sıfırdan büyük olmak üzere iki dallanma gelebilir.	Çoğu zaman koşul kodlu makineler, bit kontrolü, döngü kontrolü ve atomik semafor işlemleri gibi özel durumlar için yine de özel koşul kodu olmayan komutlar (<i>special non-condition-code instructions</i>) eklemelidir.
	Boruhattı yapısına sahip bir uygulamada (<i>In a pipelined implementation</i>), koşul kodları çakışmalarını önlemek için özel senkronizasyon gerektirir.



Control and Status Registers


Komutun yürütülmesi için dört register gereklidir:

- Program counter (PC)
 - Alınacak bir komutun adresini içerir
- Instruction register (IR)
 - En son getirilen komutu içerir
- Memory address register (MAR)
 - Bellekteki bir yerin/hücrenin adresini içerir
- Memory buffer register (MBR)
 - Belleğe yazılacak bir veri kelimesini veya en son okunan kelimeyi içerir

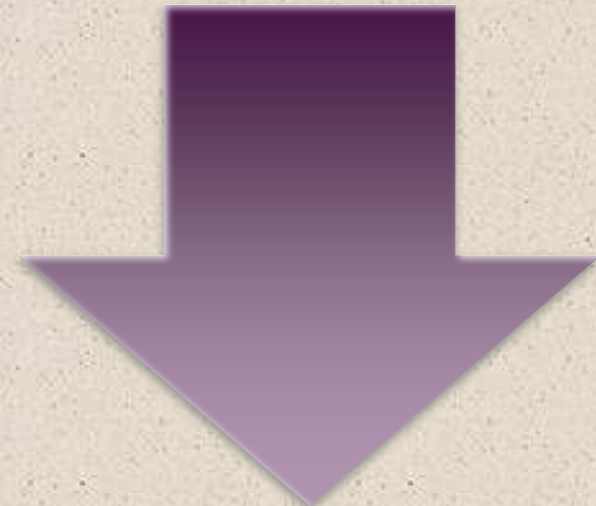


+ Program Status Word (PSW)

Çoğu işlemci tasarımında, durum bilgilerini içeren ve genellikle program durum kelimesi (PSW) olarak bilinen bir register veya register seti bulunur.



Durum bilgilerini içeren register veya register seti



Genel alanlar veya bayraklar şunları içerir:

- Sign
- Zero
- Carry
- Equal
- Overflow
- Interrupt Enable/Disable
- Supervisor

Supervisor : İşlemcinin yönetici modunda mı yoksa kullanıcı modunda mı çalıştığını gösterir. Belirli ayrıcalıklı komutlar yalnızca yönetici modunda yürütülebilir ve belirli bellek alanlarına yalnızca yönetici modunda erişilebilir.

Instruction Cycle

Includes the following stages:

Fetch

Bellekten işlemciye
doğru sıradaki komutu
okur

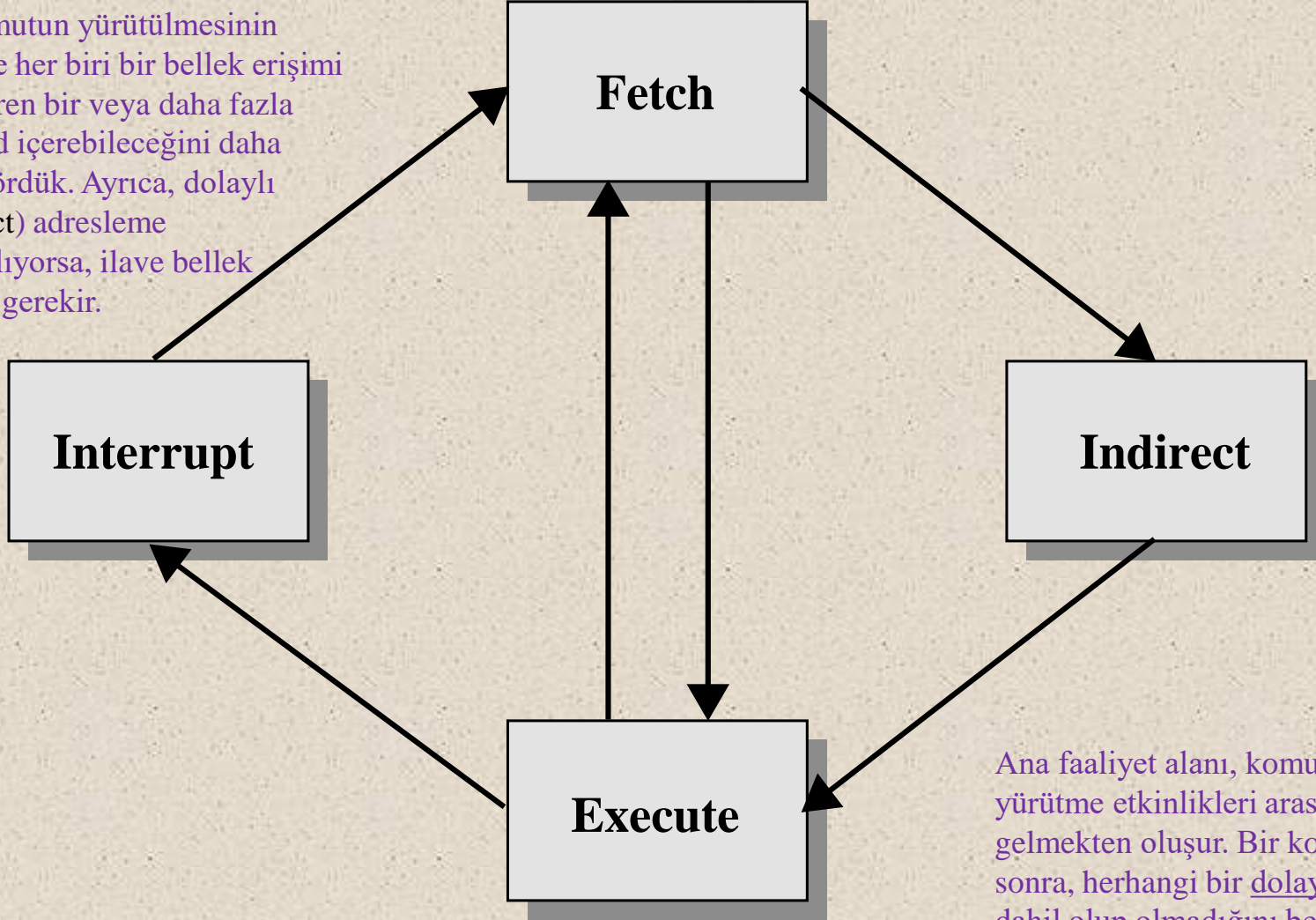
Execute

İşlem kodunu yorumlar
ve belirtilen işlemi
gerçekleştirir

Interrupt

Kesmeler
etkinleştirilirse ve bir
kesme meydana gelirse,
mevcut işlem durumu
kaydedilir ve kesme
hizmeti sağlanır.

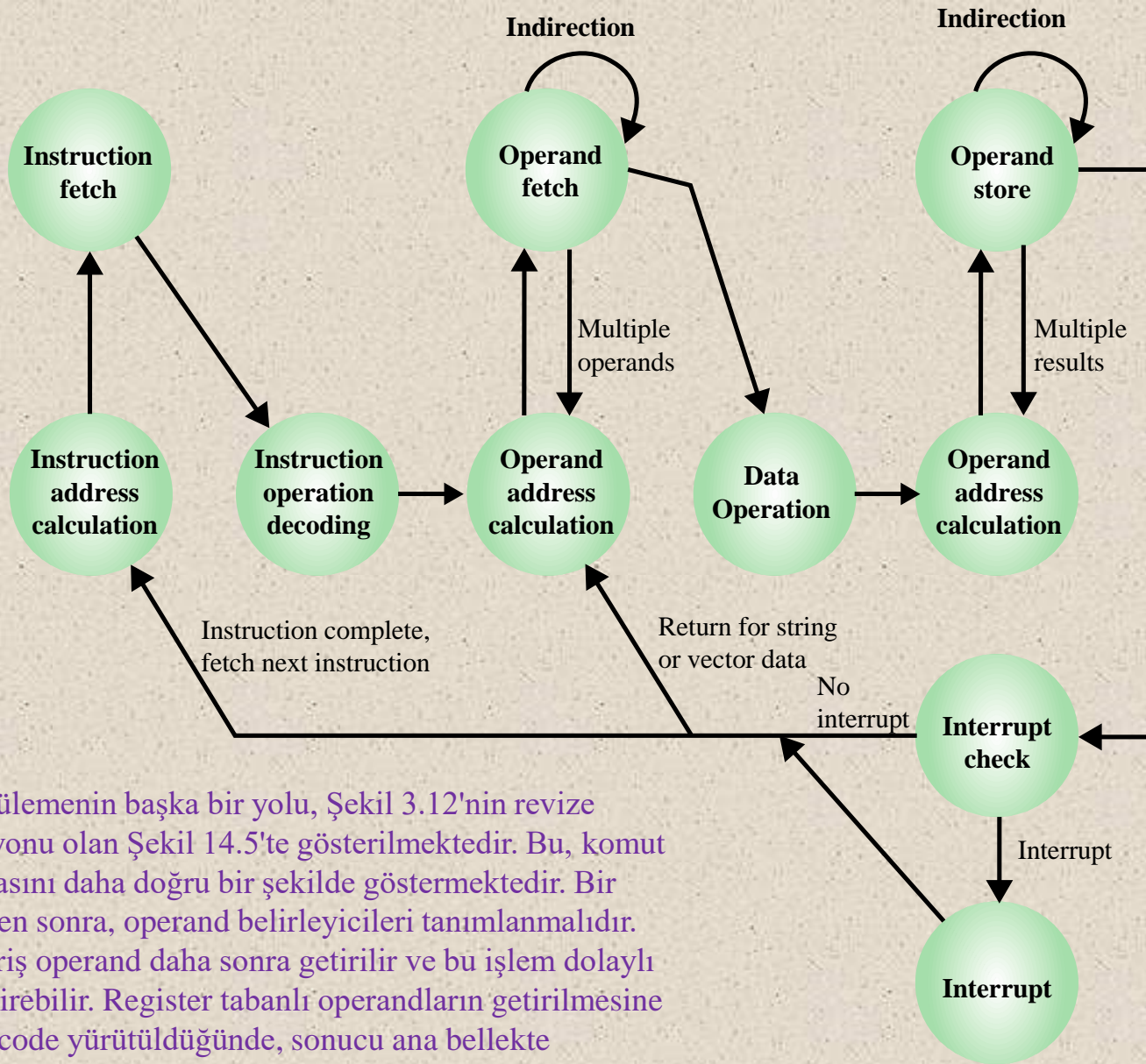
Bir komutun yürütülmesinin bellekte her biri bir bellek erişimi gerektiren bir veya daha fazla operand içerebileceğini daha önce gördük. Ayrıca, dolaylı (indirect) adresleme kullanılıyorsa, ilave bellek erişimi gerekir.



Ana faaliyet alanı, komut getirme ve komut yürütme etkinlikleri arasında gidip gelmekten oluşur. Bir komut getirildikten sonra, herhangi bir dolaylı adreslemenin dahil olup olmadığını belirlemek için incelenir. Eğer öyleyse, gerekli operandlar dolaylı adresleme kullanılarak getirilir. Yürütmenin ardından, bir sonraki komut getirmeden önce bir kesme işlenebilir.

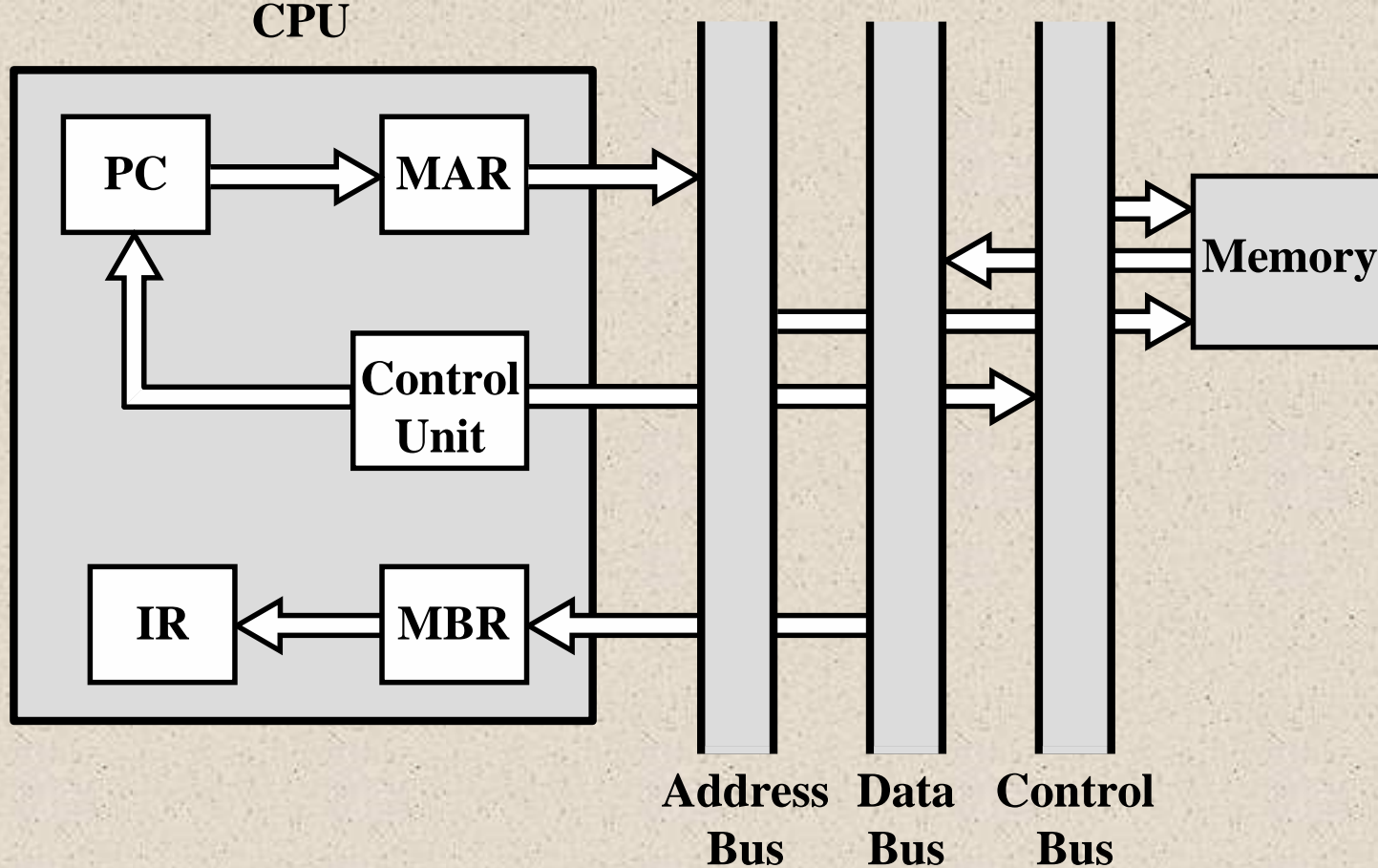
Figure 14.4 The Instruction Cycle

Dolaylı adreslerin getirilmesini bir başka komut aşaması olarak düşünebiliriz. Sonuç, Şekil 14.4'te gösterilmektedir.



Bu süreci görüntülemenin başka bir yolu, Şekil 3.12'nin revize edilmiş bir versiyonu olan Şekil 14.5'te gösterilmektedir. Bu, komut döngüsünün doğasını daha doğru bir şekilde göstermektedir. Bir komut getirildikten sonra, operand belirleyicileri tanımlanmalıdır. Bellekteki her giriş operand daha sonra getirilir ve bu işlem dolaylı adresleme gerektirebilir. Register tabanlı operandların getirilmesine gerek yoktur. Opcode yürütüldüğünde, sonucu ana bellekte saklamak için benzer bir işlem gerekebilir.

Figure 14.5 Instruction Cycle State Diagram



MBR = Memory buffer register
 MAR = Memory address register
 IR = Instruction register
 PC = Program counter

«*fetch cycle*» sırasında, bellekten bir komut okunur. Şekil 14.6, bu döngü sırasında veri akışını göstermektedir. PC, getirilecek bir sonraki komutun adresini içerir. Bu adres MAR'a taşınır ve adres veri yoluna yerleştirilir. Kontrol ünitesi bir bellek okuması talep eder ve sonuç veri yoluna yerleştirilir ve MBR'ye kopyalanır ve ardından IR'ye taşınır. Bu arada, PC bir sonraki getirme için hazırlık olarak artırılır.

Figure 14.6 Data Flow, Fetch Cycle

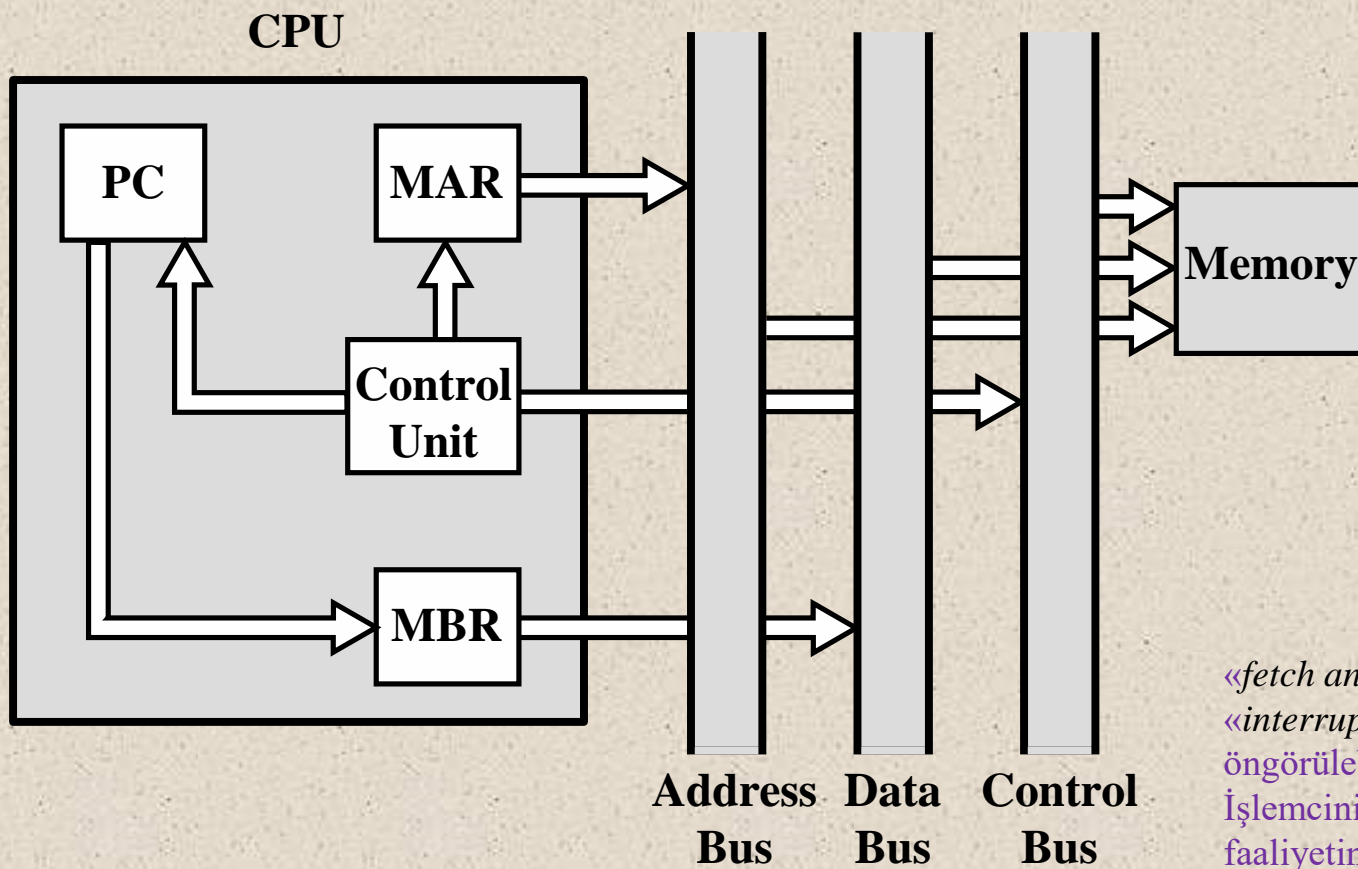


Figure 14.8 Data Flow, Interrupt Cycle

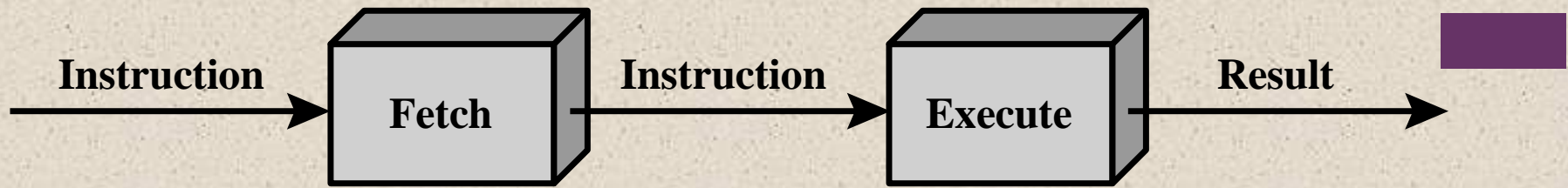
«*fetch and indirect cycles*» gibi, «*interrupt cycle*» de basit ve öngörülebilirdir (Şekil 14.8). İşlemcinin kesmeden sonra normal faaliyetine devam edebilmesi için program sayacının mevcut içeriği kaydedilmelidir. Böylelikle PC'nin içeriği belleğe yazılmak üzere MBR'ye aktarılır. Bu amaç için ayrılan özel bellek konumu, kontrol ünitesinden MAR'a yüklenir. Örneğin, bir yığın işaretçisi (stack pointer) olabilir. PC, kesme rutininin adresi ile yüklenir. Sonuç olarak, bir sonraki komut döngüsü uygun komutu getirerek başlayacaktır.

Pipelining Strategy

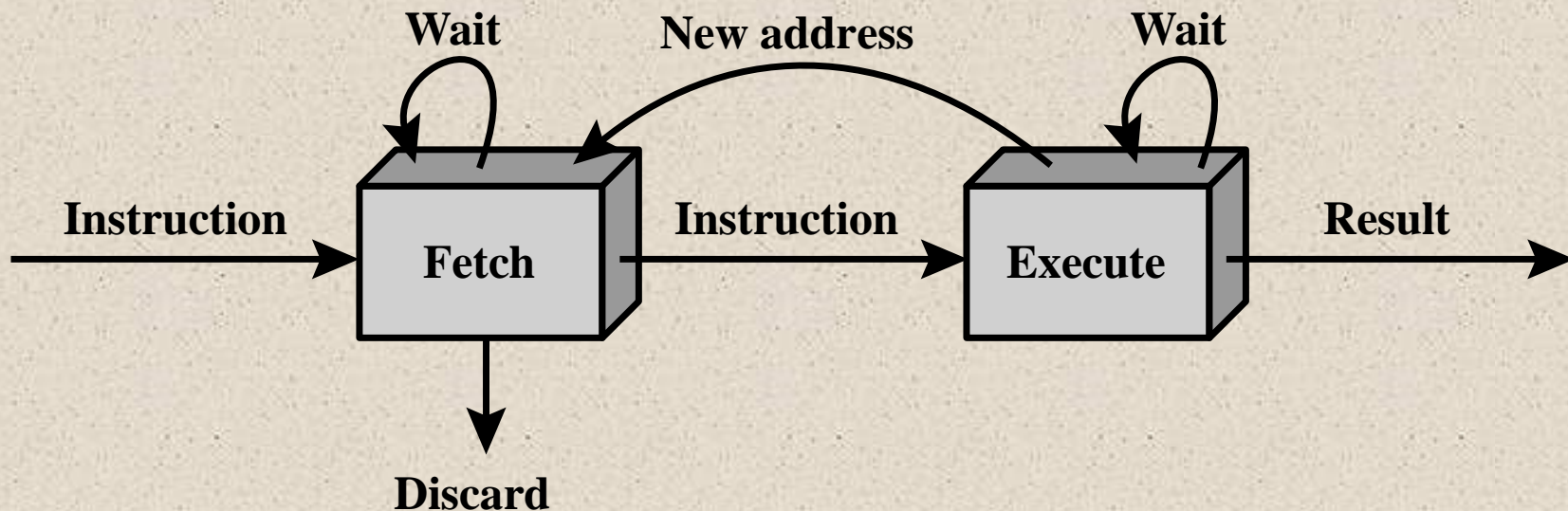
Bir üretim tesisinde
bir montaj hattının
kullanımına benzer

Bu kavramı komutun
yürütülmesine
uygulamak için bir
komutun birkaç
aşaması olduğunu
kabul etmeliyiz.

Daha önce kabul
edilen girdiler diğer
uçta çıktı olarak
görünmeden önce
bir uçta yeni
girdiler kabul edilir.



(a) Simplified view



(b) Expanded view

Figure 14.9 Two-Stage Instruction Pipeline

- Basit bir yaklaşım olarak, komut işlemeyi iki aşamaya ayırmayı düşünün: komutu getirme (fetch instruction) ve komutu yürütme (execute instruction).
- Bir komutun yürütülmesi sırasında ana belleğe erişilmediği zamanlar vardır. Bu süre, mevcut olanın yürütülmesine paralel olarak bir sonraki komutu almak için kullanılabilir.
- Şekil 14.9a bu yaklaşımı göstermektedir.
- Boru hattının iki bağımsız aşaması vardır.
 - İlk aşama bir komut getirir ve onu tamponlar.
 - İkinci aşama serbest olduğunda, ilk aşama ona tampondaki komutu iletir. İkinci aşama komutu yürütürken, birinci aşama bir sonraki komutu almak ve tamponlamak için kullanılmayan bellek döngülerinden yararlanır.
- Buna «*instruction prefetch*» veya «*fetch overlap*» denir. Komut tamponlamasını içeren bu yaklaşımın daha fazla register gerektirdiğini unutmayın. Genel olarak, boru hattı, registerların aşamalar arasında veri depolamasını gerektirir.

+ Additional Stages

- Fetch instruction (FI)
 - Bir sonraki beklenen komutu bir arabelleğe okur
- Decode instruction (DI)
 - İşlem kodunu ve operand belirleyicilerini belirler
- Calculate operands (CO)
 - Her kaynak operandın etkin adresini hesaplar
 - Bu, «displacement», «indirect», «register indirect», veya diğer adres hesaplama biçimlerini içerebilir.
- Fetch operands (FO)
 - Her operandı bellekten alır
 - Registerlardaki operandların getirilmesine gerek yoktur
- Execute instruction (EI)
 - Belirtilen işlemi gerçekleştirir ve sonucu varsa belirtilen hedef operand konumunda saklar
- Write operand (WO)
 - Sonucu bellekte saklar

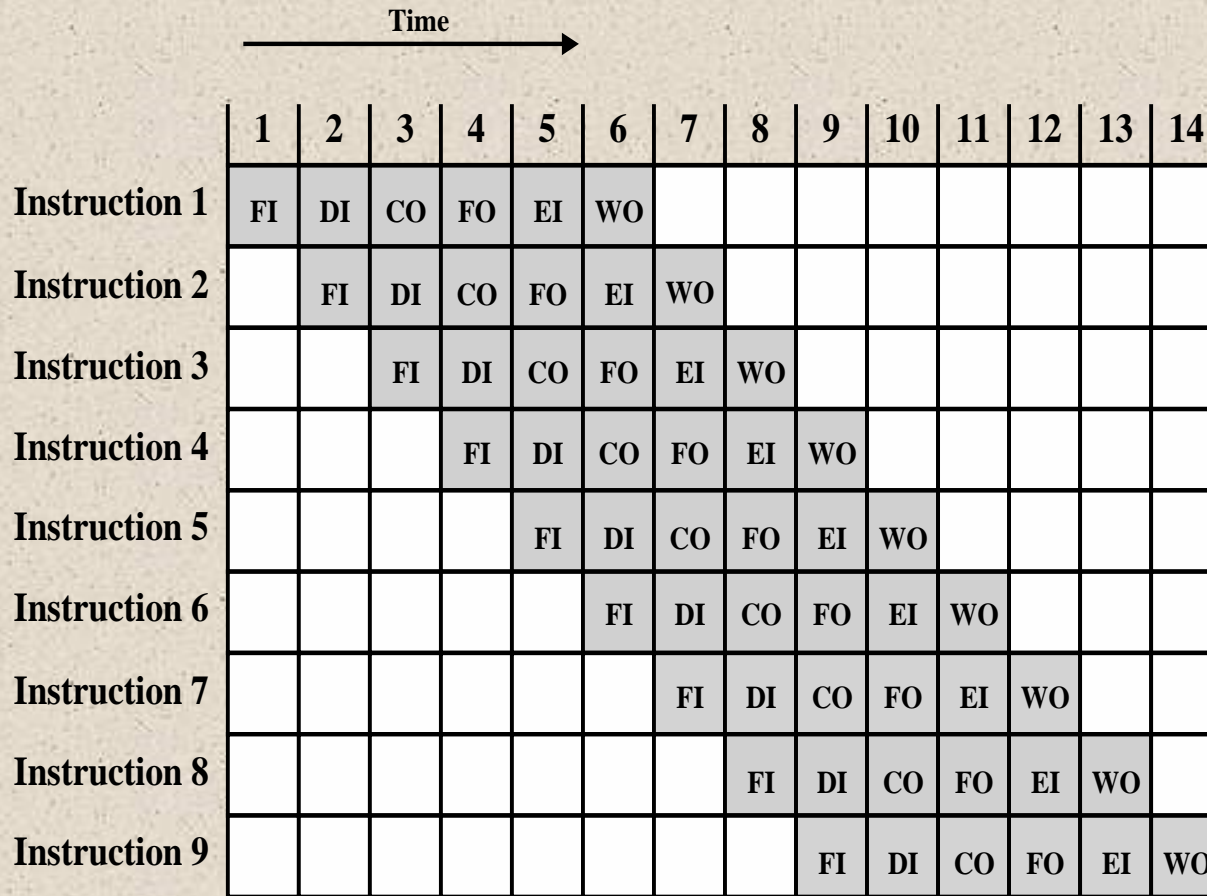


Figure 14.10 Timing Diagram for Instruction Pipeline Operation

Açıklamayı kolaylaştırmak adına, her aşamanın eşit süre olduğunu varsayalım. Bu varsayımı kullanarak, Şekil 14.10, altı aşamalı bir boru hattının 9 komut için yürütme süresini 54 zaman biriminden 14 zaman birimine düşürebileceğini göstermektedir.

Yandaki diyagram, her komutun boruhattının altı aşamasından geçtiğini varsayar. Bu her zaman böyle değildir. Örneğin, bir MOVE komutunun WO aşamasına ihtiyacı yoktur.

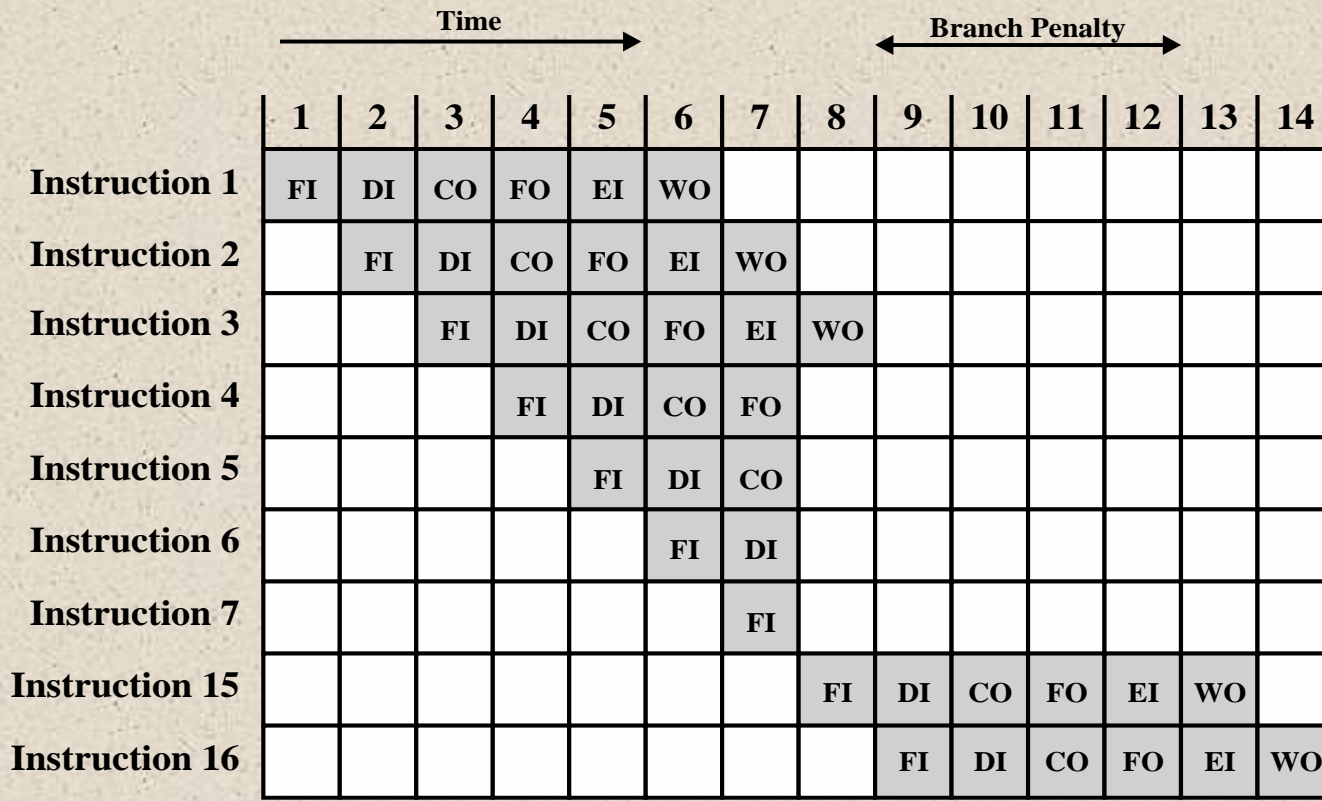
Bununla birlikte, boruhattı donanımını basitleştirmek için, zamanlama, her komutun altı aşamayı da gerektirdiği varsayılarak ayarlanır.

Ayrıca diyagram, tüm aşamaların paralel olarak gerçekleştirilebileceğini varsayar. Özellikle, bellek çatışmalarının olmadığı (no memory conflicts) varsayılır.

- Örneğin, FI, FO ve WO aşamaları bir bellek erişimini içerir.

Şema, tüm bu erişimlerin aynı anda gerçekleştirilebileceğini ima etmektedir. Çoğu bellek sistemi buna izin vermeyecektir.

Bununla birlikte, istenen değer önbellekte olabilir veya FO veya WO aşaması boş olabilir. Bu nedenle, çoğu zaman bellek çatışmaları boru hattını yavaşlatmaz.



Diğer birkaç etken, performans iyileşmesini sınırlandırır.

Altı aşama eşit süreye sahip değilse, daha önce iki aşamalı boruhattı için tartışıldığı gibi, çeşitli boruhattı aşamalarında bir miktar bekleyiş olacaktır.

Diğer bir zorluk, birkaç komut getirmeyi geçersiz kılabilen koşullu dallanma komutlarıdır.

Benzer bir öngörülemez olay, bir kesmedir.

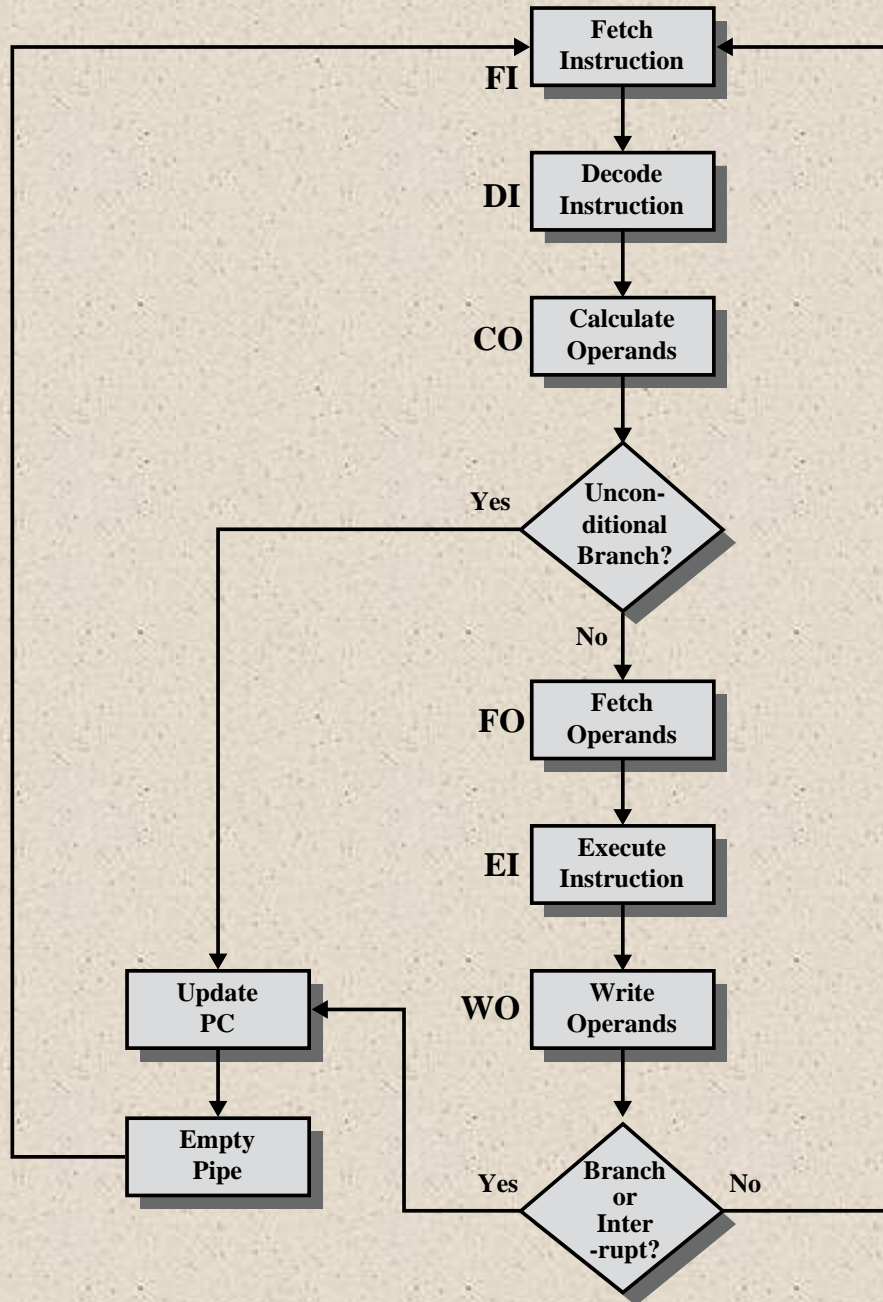
Figure 14.11 The Effect of a Conditional Branch on Instruction Pipeline Operation

Şekil 14.11, Şekil 14.10 ile aynı programı kullanarak koşullu dallanmanın etkilerini gösterir.

- Komut 3'ün, komut 15'e koşullu bir dallanma olduğunu varsayalım.
- Komut yürütülene kadar, bir sonraki komutun ne geleceğini bilmenin bir yolu yoktur.
- Bu örnekte boruhattı, bir sonraki komutu sırayla (Komut 4) yükler ve devam eder.
- Şekil 14.10'da dallanma yoktur ve iyileştirmenin tam performans avantajını elde ederiz.

- Şekil 14.11'de dallanma vardır.
- Bu, 7. zaman diliminin sonuna kadar belirlenmez.
- Bu noktada, boru hattı faydasız/lüzumsuz komutlardan arındırılmalıdır.
- 8. zaman dilimi sırasında, Komut 15 boru hattına girer. 9'dan 12'ye kadar olan zaman dilimlerinde tamamlanmış komut yok; bu, dallanmayı önceden tahmin edemediğimiz için maruz kalınan performans cezasıdır (performance penalty).

Şekil 14.12, boruhattı yapısında dallanmalar ve kesmeleri hesaba katması için gereken mantığı gösterir.



Basit iki aşamalı organizasyonda ortaya çıkmayan başka sorunlar baş gösterir. CO aşaması, hala boruhattında olan önceki bir komut tarafından değiştirilebilecek bir registerin içeriğine bağlı olabilir. Bu tür diğer register ve bellek çatışmaları meydana gelebilir. Sistem, bu tür bir çatışmayı (*conflict*) hesaba katan lojik içermelidir.

Figure 14.12 Six-Stage Instruction Pipeline

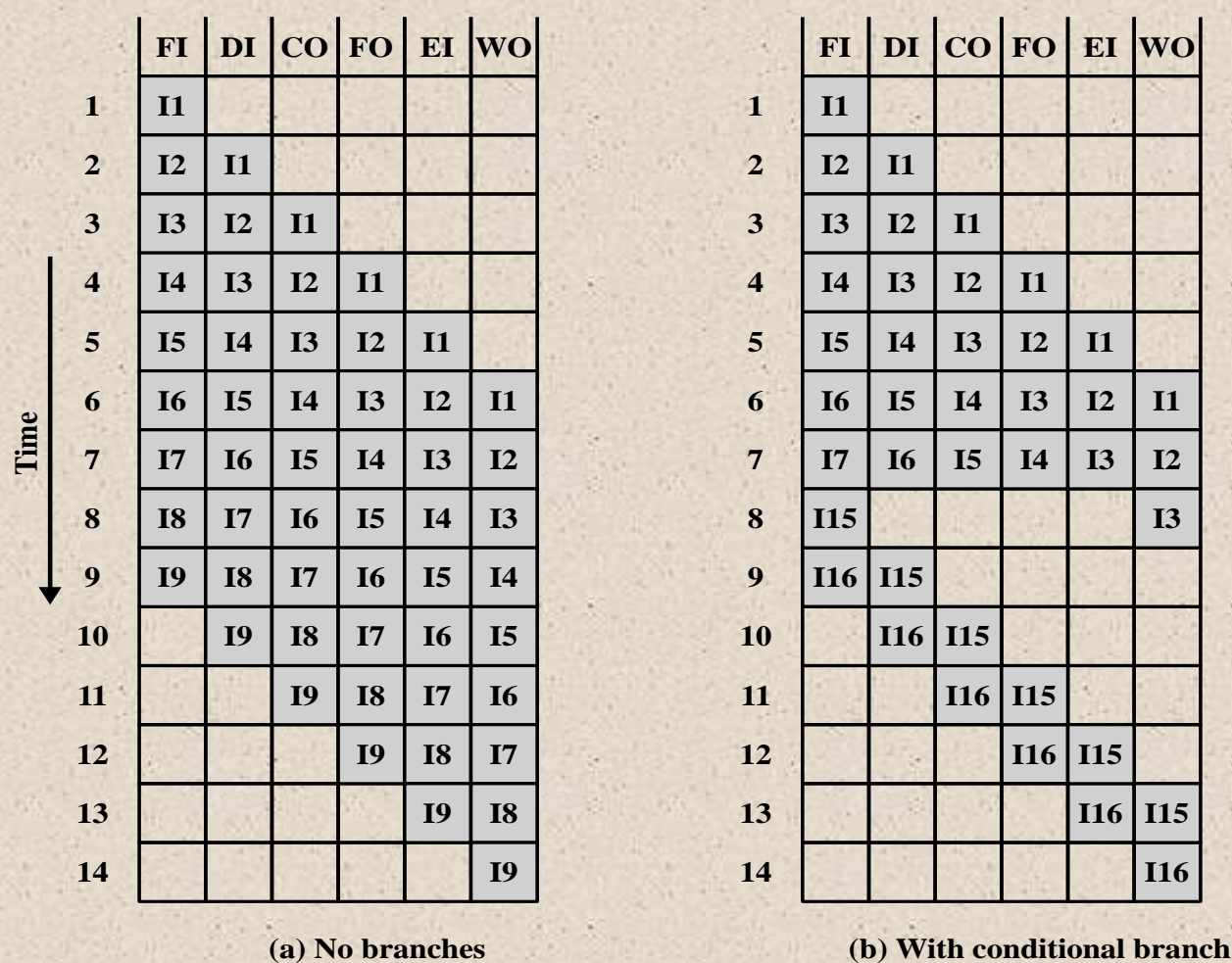


Figure 14.13 An Alternative Pipeline Depiction

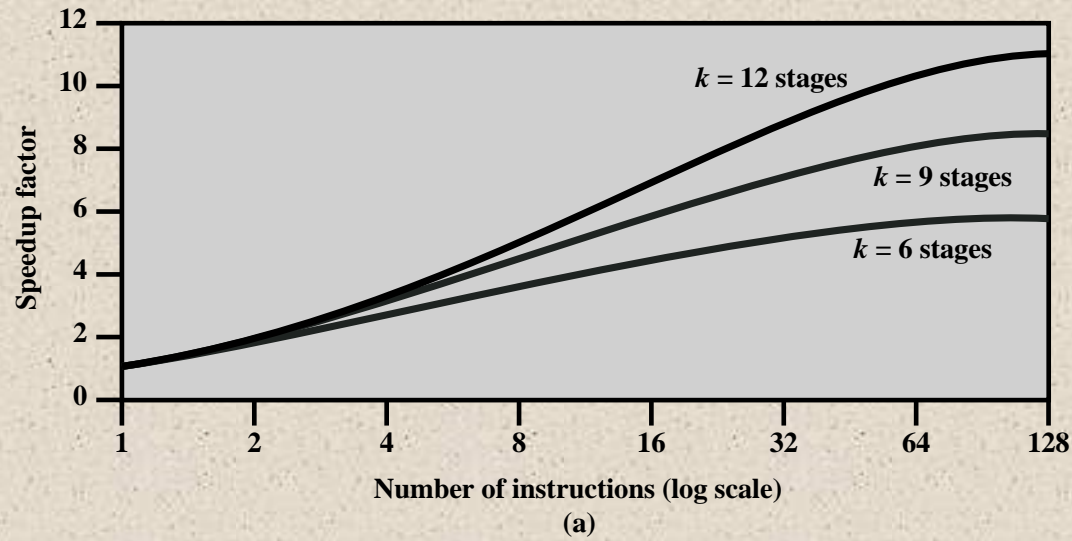
Boruhattı operasyonunu açıklığa kavuşturmak için alternatif bir tasvire bakmak faydalı olabilir.

Şekil 14.10 ve 14.11, her satır ayrı bir komutun ilerlemesini göstererek, şekiller boyunca yatay olarak zamanın ilerlemesini göstermektedir. Şekil 14.13, zamanın dikey olarak aşağıya doğru ilerlediği aynı olay sırasını ve her satırın belirli bir zamandaki boruhattının durumunu gösterdiği aynı olaylar dizisini göstermektedir.

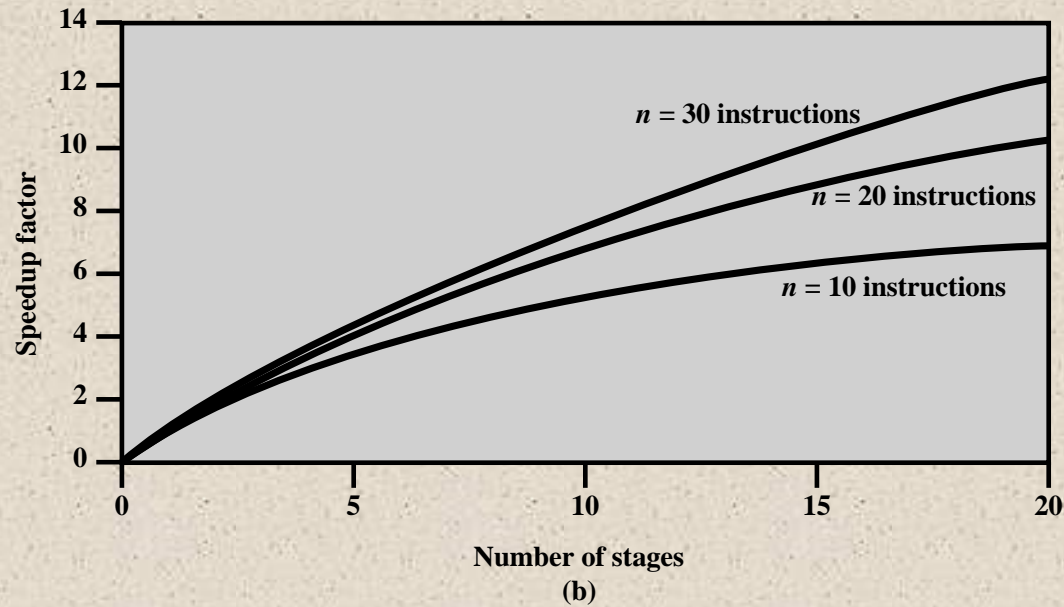
Şekil 14.13a'da, boru hattı 6. zaman diliminden 9. zaman dilimine kadar doludur (çeşitli yürütme aşamalarında 6 farklı komut vardır)

Şekil 14.13b'de, boru hattı 6 ve 7. zaman dilimlerinde doludur.

7 zaman diliminde, I3 yürütme aşamasındadır ve I15'e bir dallanma gerçekleştirir. Bu noktada, I4 ile I7 arasındaki komutlar boru hattından temizlenir, böylece 8 zaman diliminde, boruhattında yalnızca iki komut vardır, I3 ve I15.



Dolayısıyla, boru hattı aşamalarının sayısı arttıkça hızlanma potansiyeli de artar.



Bununla birlikte, pratik bir mesele olarak, ilave boru hattı aşamalarının potansiyel kazanımları; maliyet artışları, aşamalar arasındaki gecikmeler ve boru hattının boşaltılmasını gerektiren dallanmalarla karşılaşılacağı gerçeğiyle karşılanmaktadır.

Figure 14.14 Speedup Factors with Instruction Pipelining

Pipeline Hazards

Koşullar
yürütmenin/icranın
devamına izin
vermediği için boru
hattının veya boru
hattının bir kısmının
durması (*stall*)
gerektiğinde oluşur.

There are three types
of hazards:

- Resource
- Data
- Control

Boru hattı balonu
(*pipeline bubble*)
olarak da adlandırılır



		Clock cycle								
		1	2	3	4	5	6	7	8	9
Instrucion	I1	FI	DI	FO	EI	WO				
	I2		FI	DI	FO	EI	WO			
	I3			FI	DI	FO	EI	WO		
	I4				FI	DI	FO	EI	WO	

(a) Five-stage pipeline, ideal case

		Clock cycle								
		1	2	3	4	5	6	7	8	9
Instrucion	I1	FI	DI	FO	EI	WO				
	I2		FI	DI	FO	EI	WO			
	I3			Idle	FI	DI	FO	EI	WO	
	I4					FI	DI	FO	EI	WO

(b) I1 source operand in memory

Figure 14.15 Example of Resource Hazard

Kaynak çatışmasına başka bir örnek, birden çok komutun komut yürütme aşamasına (*execute instruction phase*) girmeye hazır olduğu ve tek bir ALU'nun olduğu bir durumdur. Bu tür kaynak tehlikelerine bir çözüm, ana belleğe birden çok bağlantı noktası ve birden çok ALU birimi olması gibi mevcut kaynakları artırmaktır.

Halihazırda boruhattında bulunan iki (veya daha fazla) komutun aynı kaynağa ihtiyacı olduğunda bir kaynak tehlikesi (*resource hazard*) ortaya çıkar. Sonuç, komutların, boru hattının bir bölümü için paralel yerine seri olarak yürütülmesi gerektiridir. Bir kaynak tehlikesi bazen yapısal tehlike (*structural hazard*) olarak adlandırılır.

Kaynak tehlikesinin basit bir örneğini ele alalım. Her aşamanın bir saat döngüsünü aldığı, basitleştirilmiş beş aşamalı bir boruhattı düşünün. Şekil 14.15a, her saat döngüsünde boruhattına yeni bir komutun girdiği ideal durumu göstermektedir.

Şimdi, ana belleğin tek bir bağlantı noktasına sahip olduğunu ve tüm komut getirmelerinin (*instruction fetches*), veri okumalarının ve yazmalarının teker teker gerçekleştirilmesi gerektiğini varsayalım.

Ayrıca, ön belleği yoksayın. Bu durumda, bir komut getirme işlemine paralel olarak bellekten operand okuma veya belleğe operand yazma gerçekleştirilemez.

Bu, komut I1 için kaynak operandının (source operand) bir register yerine bellekte olduğunu varsayan Şekil 14.15b'de gösterilmektedir. Bu nedenle, boruhattının komut getirme aşaması, komut I3 için komut getirmeye başlamadan önce bir döngü boyunca boşa kalmalıdır. Şekil, diğer tüm operandların registerlarda olduğunu varsayar.



		Clock cycle									
		1	2	3	4	5	6	7	8	9	10
ADD EAX, EBX		FI	DI	FO	EI	WO					
			FI	DI	Idle		FO	EI	WO		
	I3			FI			DI	FO	EI	WO	
	I4						FI	DI	FO	EI	WO

Bir operand konumuna erişimde bir çakışma olduğunda bir veri tehlikesi (data hazard) oluşur. Genel olarak tehlikeyi şu şekilde ifade edebiliriz: Bir programdaki iki komut sırayla yürütülmeli ve her ikisi de belirli bir belleğe veya register operandına erişmelidir. İki komut katı bir sırayla (*in strict sequence*) yürütülürse, herhangi bir sorun oluşmaz.

Bununla birlikte, komutlar bir boruhattı yapısında yürütülürse, operand değerinin, katı sıralı yürütme ile ortaya çıkacak olandan farklı bir sonuç üretecek şekilde güncellenmesi mümkündür. Başka bir deyişle, program, boruhattı kullanımı nedeniyle yanlış bir sonuç üretir.

Figure 14.16 Example of Data Hazard

+ Types of Data Hazard

- Read after write (RAW), or true dependency
 - Bir komut, bir register veya bellek konumunu değiştirir
 - Bir sonraki komut, bellekteki veya register konumundaki verileri okur
 - Okuma, yazma işlemi tamamlanmadan gerçekleşirse tehlike (*hazard*) oluşur
- Write after read (WAR), or antidependency
 - Bir komut, bir register veya bellek konumunu okur
 - Sıradaki (takip eden) komut bu konuma yazar
 - Okuma işlemi gerçekleşmeden önce yazma işlemi tamamlanırsa tehlike oluşur
- Write after write (WAW), or output dependency
 - İki komut da aynı konuma yazar
 - Yazma işlemleri amaçlanan sıranın tersi sırada gerçekleşirse tehlike oluşur



Control Hazard

Bir komut boruhattı tasarlamadaki en büyük sorunlardan biri, boruhattının ilk aşamalarına düzenli bir komut akışı sağlamaktır. Görüldüğü gibi birincil engel, koşullu dallanma komutudur. Komut fiilen yerine getirilinceye kadar dallanmanın yapılıp yapılmayacağını belirlemek mümkün değildir.

- «*branch hazard*» olarak da bilinir
- Boruhattı bir dallanma tahmininde yanlış karar verdiğinde gerçekleşir
- Sonradan atılması gereken komutları boru hattına getirmiş olur
 - Koşullu dallanmalarla başa çıkmak için çeşitli yaklaşımlar benimsenmiştir:
- Dealing with Branches:
 - Multiple streams
 - Prefetch branch target
 - Loop buffer
 - Branch prediction
 - Delayed branch



Multiple Streams

Basit bir boruhattı, bir sonraki getirme için iki komuttan birini seçmesi gerektiğinden ve yanlış seçim yapabileceğinden, bir dallanma komutu için cezaya katlanır

Bir kaba kuvvet yaklaşımı (*brute-force approach*), boru hattının ilk kısımlarını çoğaltmak ve boru hattının iki akıştan yararlanarak her iki komutu da almasına izin vermektedir.

Bu dezavantajlara rağmen, bu strateji performansı artırabilir. İki veya daha fazla boruhattı akışına sahip makineler örnek olarak IBM 370/168 ve IBM 3033 verilebilir.

Drawbacks:

- Birden çok boruhattında, registerlara ve belleğe erişim için çekişme gecikmeleri (*contention delays*) vardır.
- Orijinal dallanma kararı çözülmeden önce ilave dallanma komutları boruhattına girebilir

Prefetch Branch Target

- Koşullu bir dallanmayla karşılaşıldığında, dallanma komutunu takip eden komuta ek olarak dallanmanın hedefi önceden getirilir (*prefetched*).
- Bu hedef daha sonra dallanma komutu yürütülene kadar kaydedilir.
- Dallanma gerçekleşirse, hedef zaten önceden getirilmişti
- IBM 360/91 bu yaklaşımı kullanır





Loop Buffer

- Boruhattının komut getirme aşamasında bulunan ve sırayla en son getirilen n tane komutu içeren küçük, çok yüksek hızlı bellek
- Faydaları:
 - Sırayla getirilen komutlar, normal bellek erişim süresi olmadan kullanılabilir olacaktır.
 - Dallanma komutunun adresinin sadece birkaç konum ilerisindeki bir hedefe bir dallanma meydana gelirse, hedef zaten arabellekte olacaktır.
 - Bu, IF-THEN ve IF-THEN-ELSE dizilerinin oldukça yaygın oluşumu için kullanışlıdır.
 - Bu strateji özellikle döngülerle ve yinelenmelerle uğraşmak için çok uygundur.
 - dolayısıyla adı döngü tamponu/arabelleği (*loop buffer*)
 - döngü arabelleği bir döngüdeki tüm komutları içerecek kadar büyükse, bu komutların ilk yinelenme için bellekten yalnızca bir kez alınması gerekir. Sonraki yinelenmeler için gerekli tüm komutlar zaten arabellektedir.
- Prensipte komutlara ayrılmış bir ön belleğe benzer:
 - Farkı:
 - Döngü tamponu, yalnızca peşpeşe gelen komutları tutar
 - Boyut olarak çok daha küçüktür ve dolayısıyla maliyeti daha düşüktür

Branch address

Şekil 14.17, bir döngü tampon örneğini gösterir. Tampon 256 bayt içeriyorsa ve bayt adresleme kullanılıyorsa, arabelleği indekslemek için en düşük değerli 8 bit kullanılır. Kalan en değerli bitler, dallanma hedefinin arabellek tarafından yakalanan çevre içinde olup olmadığını belirlemek için kontrol edilir.

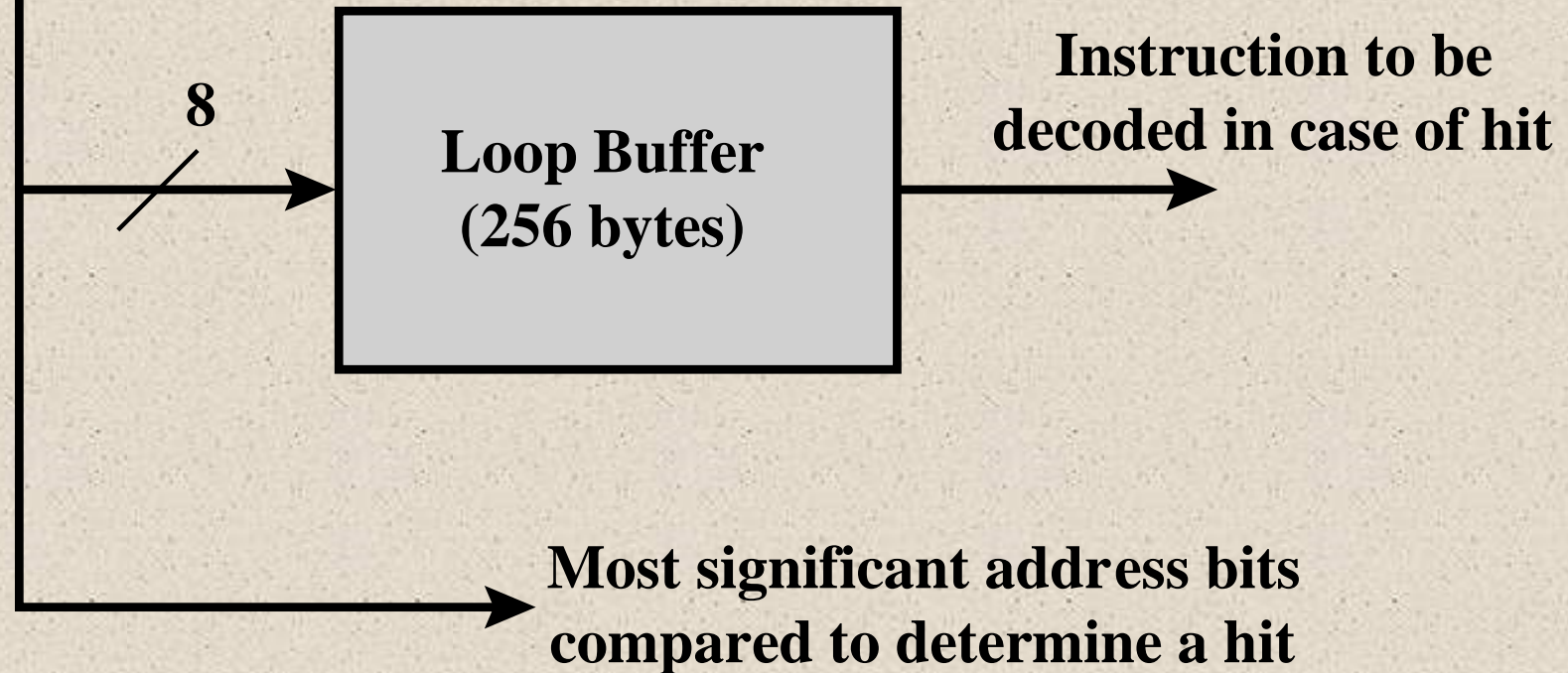


Figure 14.17 Loop Buffer



Branch Prediction

- Bir dallanmanın yapılıp yapılmayacağını tahmin etmek için çeşitli teknikler kullanılabilir:

1. Predict never taken

2. Predict always taken

3. Predict by opcode

- Bu yaklaşımlar statiktir

- Koşullu dallanma komutunun zamanına kadar yürütme geçmişine bağlı değildirler.

1. Taken/not taken switch

2. Branch history table

- Bu yaklaşımlar dinamiktir

- Yürütme geçmişine (*execution history*) bağlıdır

«Predict never taken» yaklaşımı, tüm dallanma tahmin yöntemleri arasında en popüler olanıdır.

Program davranışını analiz eden çalışmalar, koşullu dallanmaların % 50'sinden fazlasının dallandığını göstermiştir [LILJ88] ve bu nedenle, her iki yoldan da önceden getirme maliyeti aynıysa, her zaman dallanma hedef adresinden önceden getirme, sıralı yoldan önceden getirme seçeneğinden daha iyi performans vermelidir. Ancak, sayfalamalı bir makinede (in a paged machine), dallanma hedefinin önceden getirilmesinin, bir sonraki komutun sırayla önceden getirilmesinden daha fazla bir sayfa hatasına neden olması daha olasıdır ve bu nedenle bu performans cezası dikkate alınmalıdır. Bu cezayı azaltmak için bir kaçınma mekanizması kullanılabilir.



Branch Prediction

Son statik yaklaşım, kararı dallanma komutunun işlem koduna göre verir. İşlemci, dallanmanın belirli dallanma işlem kodları için yapılacağını, diğerleri için yapılmayacağını varsayar. [LILJ88], bu strateji ile %75'in üzerinde başarı oranları bildirmektedir.

Dinamik dallanma stratejileri, bir programdaki koşullu dallanma komutlarının geçmişini kaydederek tahmin doğruluğunu iyileştirmeye çalışır. Örneğin, komutun yakın geçmişini yansıtan her koşullu dallanma komutu ile bir veya daha fazla bit ilişkilendirilebilir. Bu bitler, işlemciyi komutla bir sonraki karşılaşıldığında belirli bir karar vermeye yönlendiren «taken/ not taken switch» olarak adlandırılır. Tipik olarak, bu geçmiş bitleri, ana bellekteki komutla ilişkilendirilmez. Bunun yerine, geçici yüksek hızlı depolama alanında tutulurlar. Bir olasılık, bu bitleri bir önbellekteki herhangi bir koşullu dallanma komutu ile ilişkilendirmektir. Komut önbellekte değiştirildiğinde geçmişi de kaybolur. Diğer bir olasılık, her «entry»de bir veya daha fazla geçmiş biti ile yakın zamanda yürütülen dallanma komutları için küçük bir tablo tutmaktır. İşlemci, tabloya bir önbellek gibi çağrışimli olarak veya dallanma komutu adresinin düşük dereceli bitlerini kullanarak erişebilir.

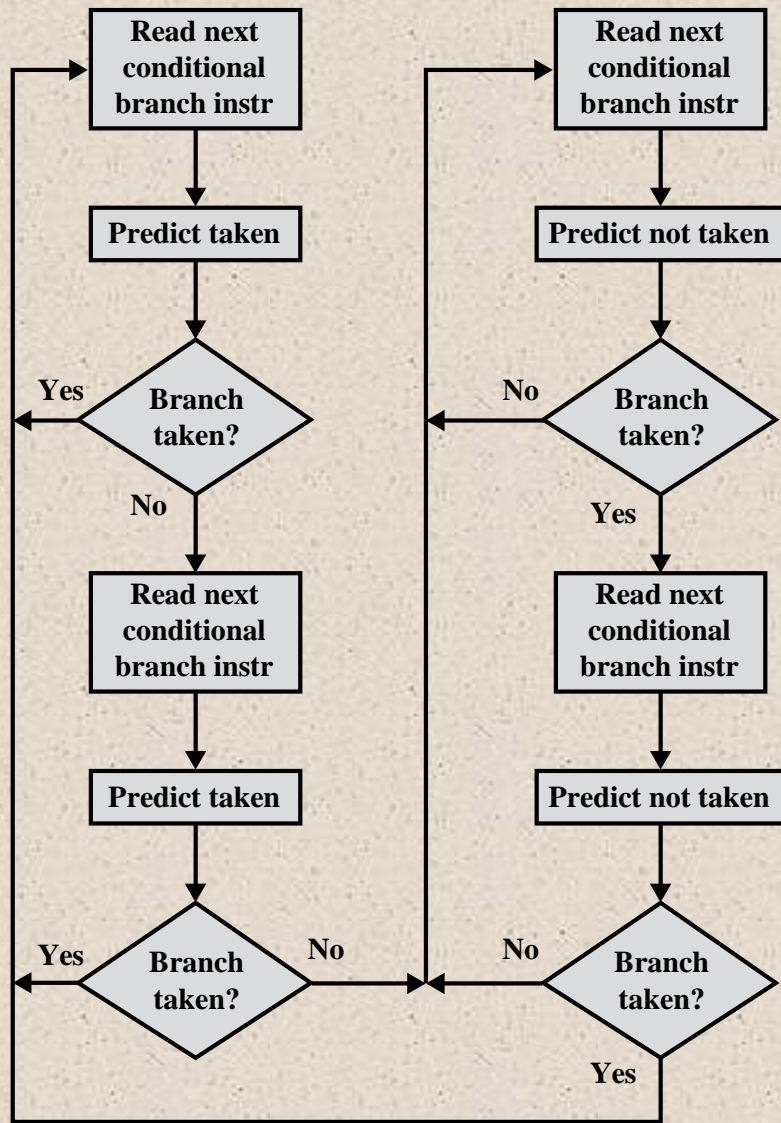


Figure 14.18 Branch Prediction Flow Chart

İki bit kullanılırsa, ilgili komutun yürütülmesinin son iki örneğinin sonucunu kaydetmek veya başka bir şekilde bir durumu kaydetmek için kullanılabilirler. Şekil 14.18 tipik bir yaklaşımı göstermektedir. Algoritmanın akış şemasının sol üst köşesinde başladığını varsayalım. Karşılaşılan her ardışık koşullu dallanma komutu dallandığı sürece, karar süreci bir sonraki dallanmanın yapılacağını tahmin eder. Tek bir tahmin yanlışsa, algoritma bir sonraki dallanmanın gerçekleştiğini tahmin etmeye devam eder. Yalnızca ardışık iki dallanma gerçekleşmezse, algoritma akış şemasının sağ tarafına geçer. Daha sonra, algoritma, arka arkaya iki dallanma gerçekleşene kadar dallanmanın gerçekleşmeyeceğini tahmin edecektir. Bu nedenle, algoritma, tahmin kararını değiştirmek için iki ardışık yanlış tahmin gerektirir.

Karar süreci, Şekil 14.19'da gösterilen bir sonlu durum makinesi (finite-state machine) ile daha kompakt bir şekilde temsil edilebilir.

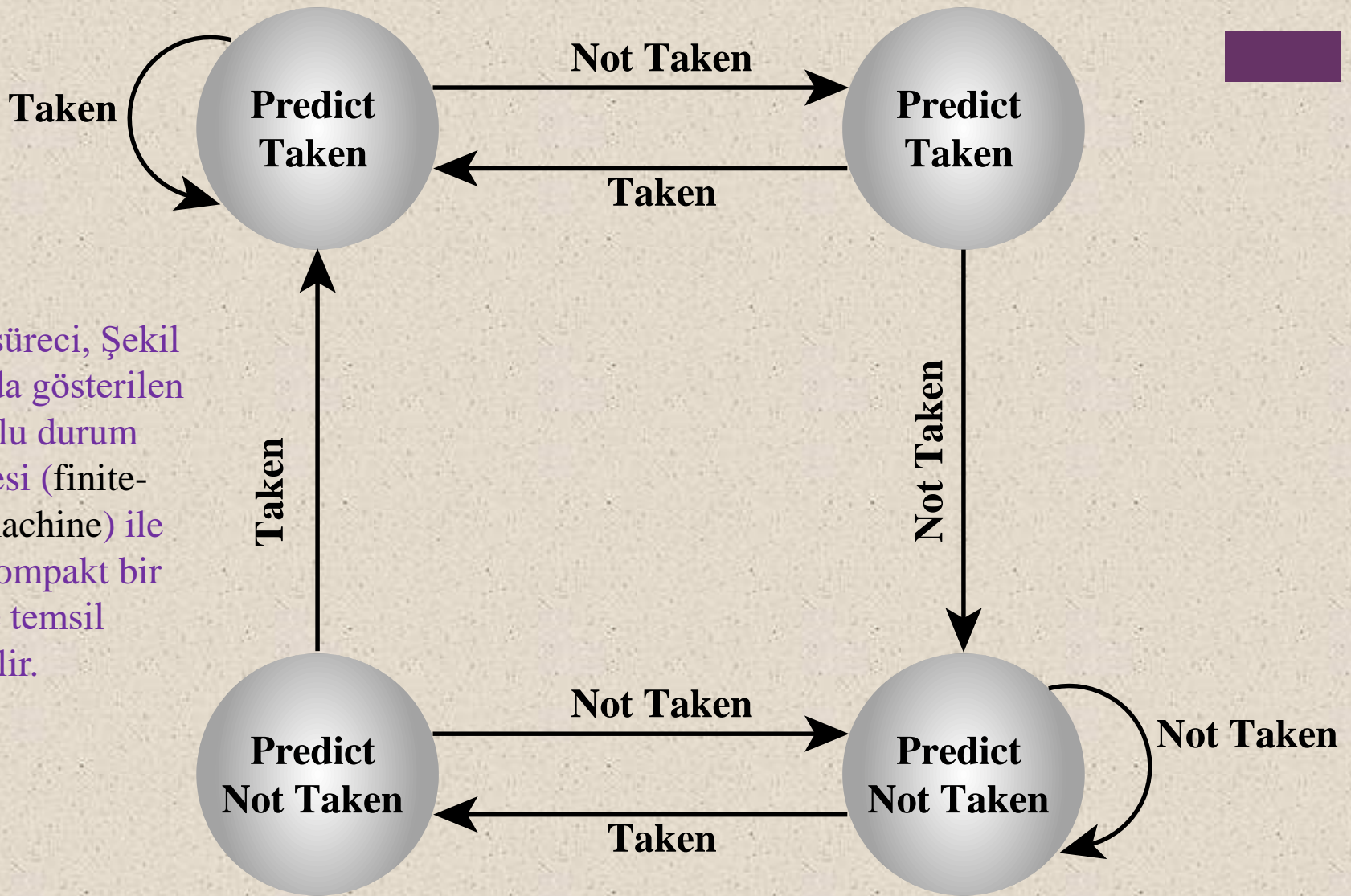
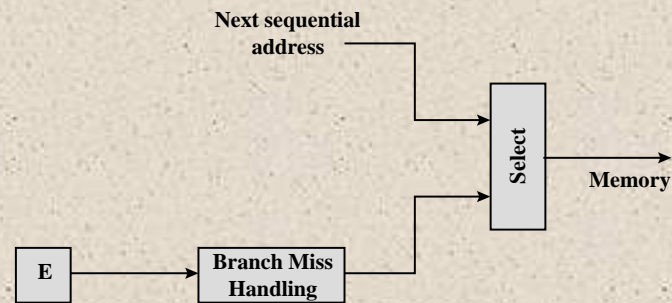
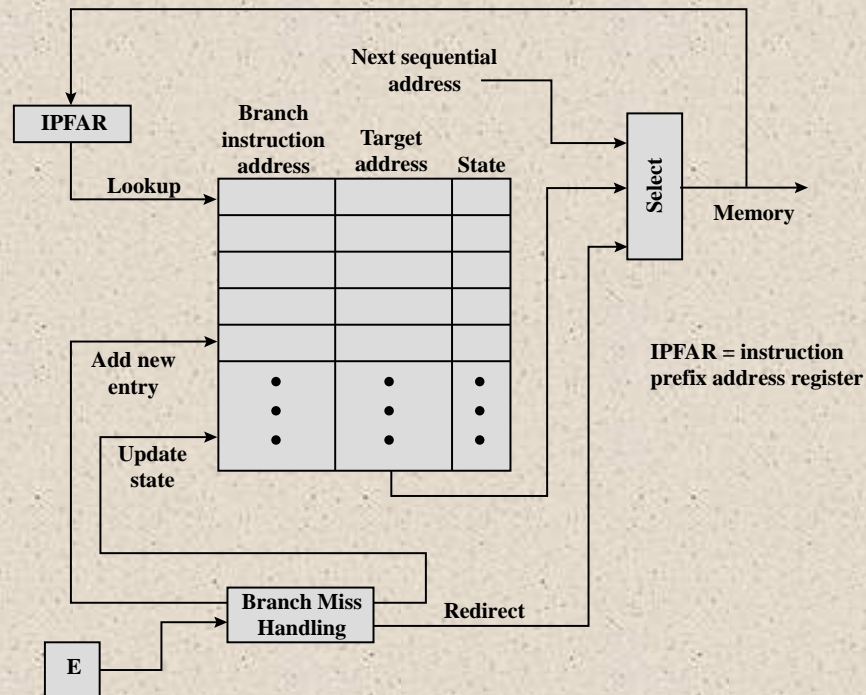


Figure 14.19 Branch Prediction State Diagram



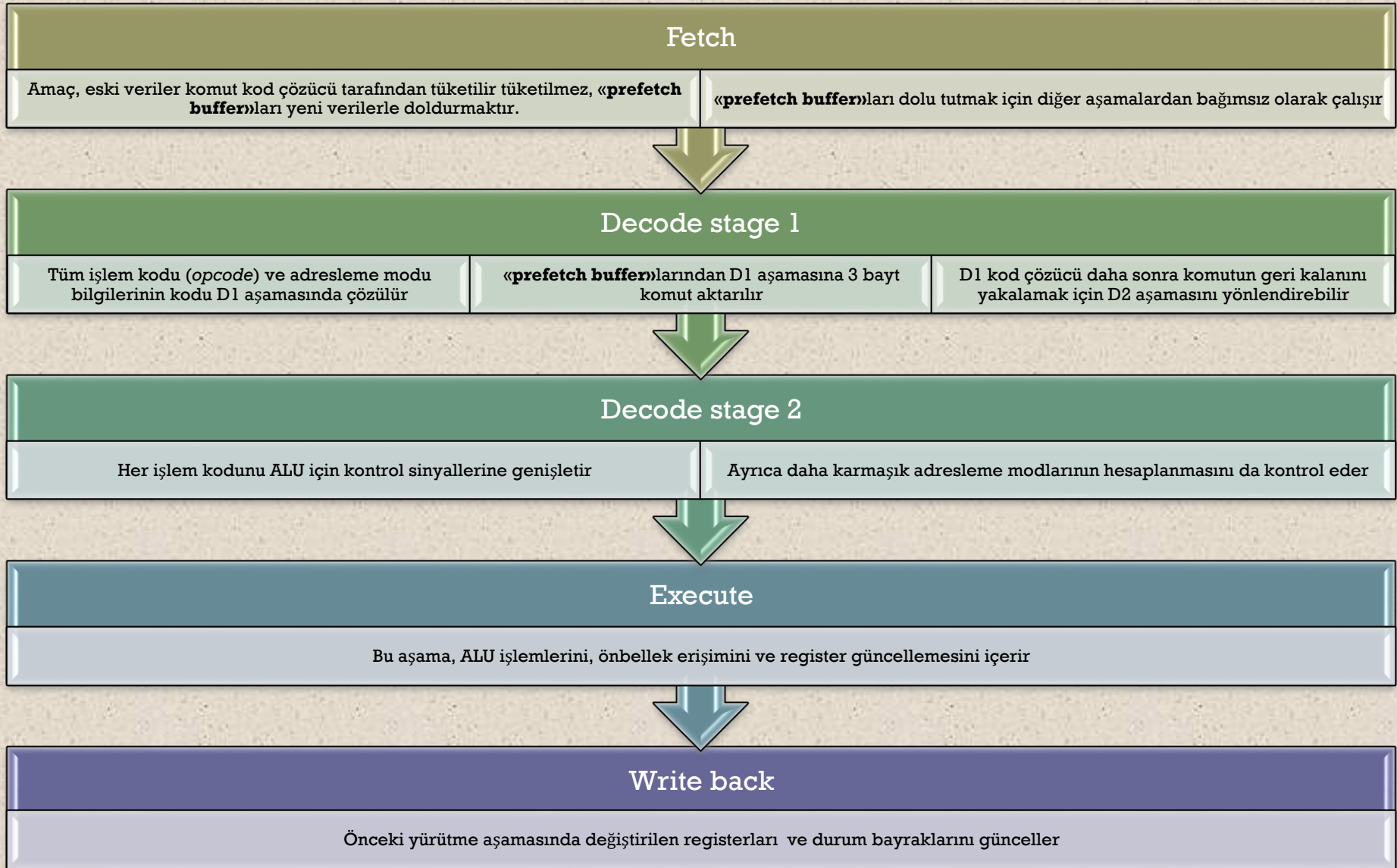
(a) Predict never taken strategy



(b) Branch history table strategy

Figure 14.20 Dealing with Branches

Intel 80486 Pipelining



Şekil 14.21a, bir bellek erişimi gerektiğinde boruhattına herhangi bir gecikmenin getirilmediğini göstermektedir.

Ancak, Şekil 14.21b'de gösterildiği gibi, bellek adreslerini hesaplamak için kullanılan değerler için bir gecikme olabilir. Yani, bellekten bir registra bir değer yüklenirse ve bu register daha sonra bir sonraki komutta temel register olarak kullanılırsa, işlemci bir periyot (cycle) için durur.

Şekil 14.21c, dallanmanın gerçekleştiğini varsayılarak (assuming that the branch is taken) dallanma komutunun zamanlamasını göstermektedir. Karşılaştırma komutu, WB aşamasında durum kodlarını günceller ve bypass yolları, bunu aynı anda dallanma (*Jump*) komutunun EX aşaması için kullanılabilir hale getirir. Paralel olarak, işlemci dallanma komutunun EX aşaması sırasında dallanmanın hedefine spekülatif bir getirme döngüsü (speculative fetch cycle) çalıştırır. İşlemci yanlış bir dallanma koşulu belirlerse, bu önceden getirmeyi atar ve sonraki sıralı komutla (zaten getirilmiş ve kodu çözülmüş) yürütmeye devam eder.

Figure 14.21 80486 Instruction Pipeline Examples

Şekil 14.21, boru hattının işleyişine ilişkin örnekleri göstermektedir.

(a) Integer Unit in 32-bit Mode

Type	Number	Length (bits)	Purpose
General	8	32	General-purpose user registers
Segment	6	16	Contain segment selectors
EFLAGS	1	32	Status and control bits
Instruction Pointer	1	32	Instruction pointer

(b) Integer Unit in 64-bit Mode

Type	Number	Length (bits)	Purpose
General	16	32	General-purpose user registers
Segment	6	16	Contain segment selectors
RFLAGS	1	64	Status and control bits
Instruction Pointer	1	64	Instruction pointer

(c) Floating-Point Unit

Type	Number	Length (bits)	Purpose
Numeric	8	80	Hold floating-point numbers
Control	1	16	Control bits
Status	1	16	Status bits
Tag Word	1	16	Specifies contents of numeric registers
Instruction Pointer	1	48	Points to instruction interrupted by exception
Data Pointer	1	48	Points to operand interrupted by exception

Table 14.2

x86

Processor

Registers

Tagword: Bu 16 bitlik register, her bir kayan noktalı sayısal register için, karşılık gelen registerin içeriğinin doğasını gösteren 2 bitlik bir etiket içerir. Dört olası değer geçerli, sıfır, özel (NaN, sonsuz, denormalize) ve boşdur (empty). Bu «tag»ler, programların, registerdaki gerçek verilerin karmaşık kodunu çözmeden sayısal bir registerin içeriğini kontrol etmelerini sağlar. Örneğin, bir «context switch » yapıldığında, işlemcinin boş olan kayan nokta registerlarını kaydetmesine gerek yoktur.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	I D	V I P	V I F	A C	V M	R F	0	N T	I O P L	O F	D F	I F	T F	S F	Z F	0	A F	0	P F	1	C F	

X ID = Identification flag

X VIP = Virtual interrupt pending

X VIF = Virtual interrupt flag

X AC = Alignment check

X VM = Virtual 8086 mode

X RF = Resume flag

X NT = Nested task flag

X IOPL = I/O privilege level

S OF = Overflow flag

C DF = Direction flag

X IF = Interrupt enable flag

X TF = Trap flag

S SF = Sign flag

S ZF = Zero flag

S AF = Auxiliary carry flag

S PF = Parity flag

S CF = Carry flag

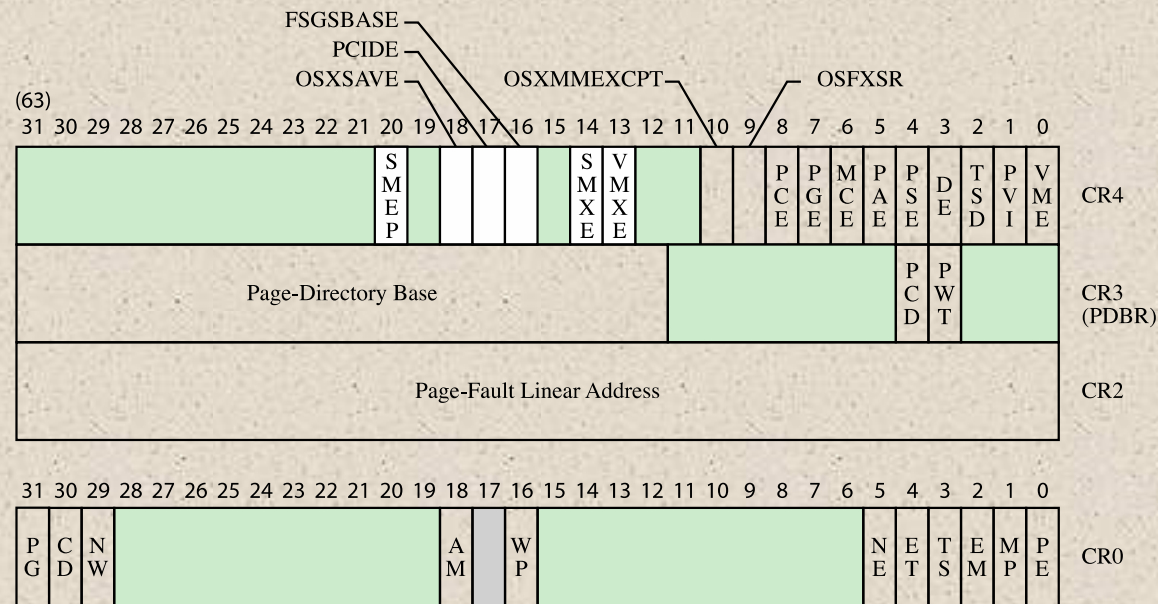
S Indicates a Status Flag

C Indicates a Control Flag

X Indicates a System Flag

Shaded bits are reserved

Figure 14.22 x86 EFLAGS Register

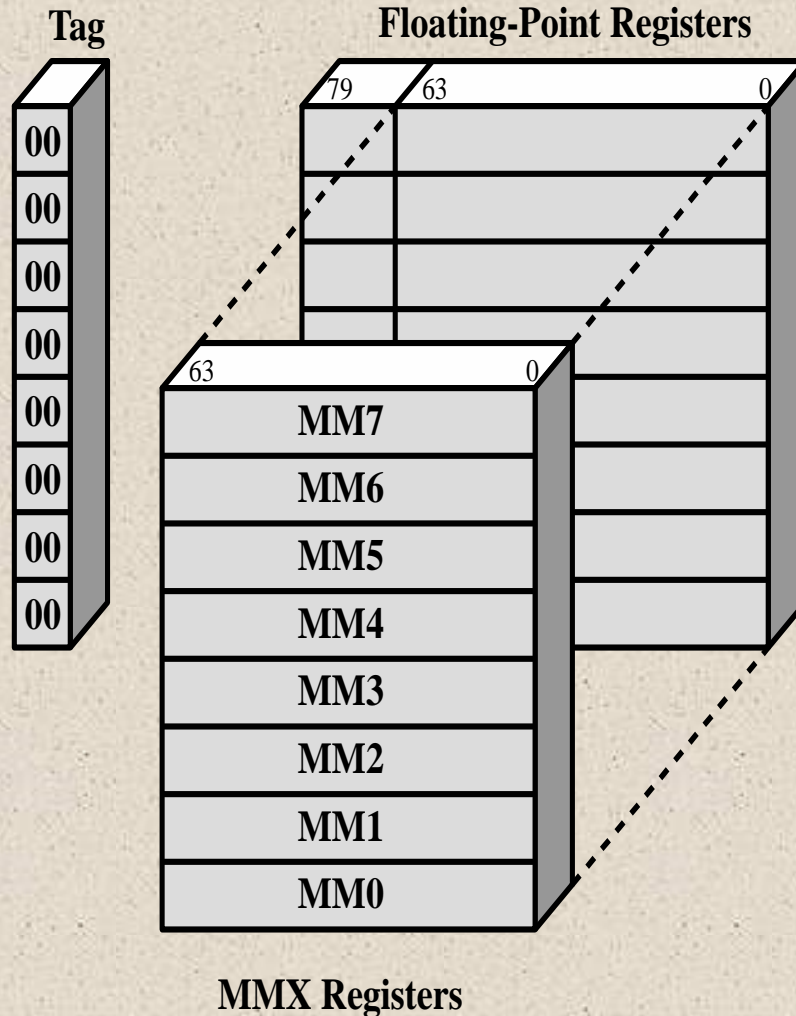


shaded area indicates reserved bits

OSXSAVE	=	XSAVE enable bit	VME	=	Virtual 8086 Mode Extensions
PCIDE	=	Enables process-context identifiers	PCD	=	Page-level Cache Disable
FSGSBASE	=	Enables segment base instructions	PWT	=	Page-level Writes Transparent
SMXE	=	Enable Safer mode extensions	PG	=	Paging
VMXE	=	Enable virtual machine extensions	CD	=	Cache Disable
OSXMMEXCPT	=	Support unmasked SIMD FP exceptions	NW	=	Not Write Through
OSFXSR	=	Support FXSAVE, FXSTOR	AM	=	Alignment Mask
PCE	=	Performance Counter Enable	WP	=	Write Protect
PGE	=	Page Global Enable	NE	=	Numeric Error
MCE	=	Machine Check Enable	ET	=	Extension Type
PAE	=	Physical Address Extension	TS	=	Task Switched
PSE	=	Page Size Extensions	EM	=	Emulation
DE	=	Debug Extensions	MP	=	Monitor Coprocessor
TSD	=	Time Stamp Disable	PE	=	Protection Enable
PVI	=	Protected Mode Virtual Interrupt			

Figure 14.23 x86 Control Registers

Floating-Point



x86 MMX yeteneğinin birkaç 64-bit veri türünü kullandığını hatırlayın. MMX komutları, sekiz MMX registerinin desteklenmesi için 3 bitlik register adres alanlarını kullanır. Aslında, işlemci spesifik MMX registerlarını içermez. Bunun yerine, işlemci bir «aliasing» tekniği kullanır (Şekil 14.24).

Mevcut kayan nokta (floating-point) registerları, MMX değerlerini depolamak için kullanılır. Spesifik olarak, her kayan nokta registerinin düşük değerli 64 bit (mantissa), sekiz MMX registeri oluşturmak için kullanılır. Böylece, eski 32-bit x86 mimarisi, MMX özelliğini desteklemek için kolayca genişletilir. Bu registerların MMX kullanımının bazı temel özellikleri şunlardır:

- Kayan noktalı registerların, kayan nokta işlemleri için bir yığın olarak ele alındığını hatırlayın. MMX işlemleri için, bu aynı registerlara doğrudan erişilir (accessed directly).
- Herhangi bir kayan nokta işleminden sonra bir MMX komutu ilk kez yürütüldüğünde, FP etiket kelimesi (tag word) geçerli olarak işaretlenir. Bu, yığın işleminden doğrudan register adreslemeye (direct register addressing) geçişi yansıtır.

Figure 14.24 Mapping of MMX Registers to Floating-Point Registers

+ Interrupt Processing

Interrupts and Exceptions

■ Interrupts

- Donanımdan gelen bir sinyal tarafından üretilir ve bir programın yürütülmesi sırasında rastgele zamanlarda ortaya çıkabilir.
- Maskable
 - İşlemcinin INTR pininden alınır. İşlemci, kesme etkinleştirme bayrağı (IF) set edilmedikçe maskelenebilir bir kesmeyi tanımaz/dikkate almaz.
- Nonmaskable
 - İşlemcinin NMI pininden alınır. Bu tür kesmelerin tanınması engellenemez.

■ Exceptions

- Yazılım tarafından üretilir ve bir komutun yürütülmesiyle tetiklenir
- **Processor-detected exceptions**
 - İşlemci tarafından algılanan istisnalar: İşlemci bir komutu yürütmeye çalışırken bir hatayla karşılaştığında ortaya çıkar.
- **Programmed exceptions**
 - Bunlar bir istisna oluşturan komutlardır (örneğin, INTO, INT3, INT ve BOUND).

■ Kesme vektör tablosu (Interrupt vector table)

- Her tür kesmeye bir numara atanır
- Bu numara, kesme vektör tablosunu indekslemek için kullanılır

Vector Number	Description
0	Divide error; division overflow or division by zero
1	Debug exception; includes various faults and traps related to debugging
2	NMI pin interrupt; signal on NMI pin
3	Breakpoint; caused by INT 3 instruction, which is a 1-byte instruction useful for debugging
4	INTO-detected overflow; occurs when the processor executes INTO with the OF flag set
5	BOUND range exceeded; the BOUND instruction compares a register with boundaries stored in memory and generates an interrupt if the contents of the register is out of bounds.
6	Undefined opcode
7	Device not available; attempt to use ESC or WAIT instruction fails due to lack of external device
8	Double fault; two interrupts occur during the same instruction and cannot be handled serially
9	Reserved
10	Invalid task state segment; segment describing a requested task is not initialized or not valid
11	Segment not present; required segment not present
12	Stack fault; limit of stack segment exceeded or stack segment not present
13	General protection; protection violation that does not cause another exception (e.g., writing to a read-only segment)
14	Page fault
15	Reserved
16	Floating-point error; generated by a floating-point arithmetic instruction
17	Alignment check; access to a word stored at an odd byte address or a doubleword stored at an address not a multiple of 4
18	Machine check; model specific
19-31	Reserved
32-255	User interrupt vectors; provided when INTR signal is activated



Table 14.3

x86

Exception

and

Interrupt

Vector Table

Unshaded: exceptions
Shaded: interrupts

NMI donanım kesmesi Tip 2'dir. INTR donanım kesmelerine 32 ila 255 aralığında sayılar atanır; bir INTR kesmesi oluşturulduğunda, bu kesme için kesme vektörü numarasıyla veri yolunda buna eşlik edilmelidir. Kalan vektör numaraları istisnalar (exceptions) için kullanılır.

+ The ARM Processor

ARM is primarily a RISC system with the following attributes:

- Orta seviyede uniform register dizisi
 - Bazı CISC sistemlerinde bulunandan daha fazla, ancak birçok RISC sisteminde bulunandan daha az
- İşlemlerin doğrudan bellekte değil, yalnızca registerlardaki operandlar üzerinde gerçekleştirildiği bir «**load/store**» veri işleme modeli
- Standart set için 32 bitlik ve Thumb komut seti için 16 bitlik tek tip sabit uzunlukta komutlar
- Ayrı aritmetik mantık birimi (ALU) ve kaydırma (*shifter*) birimleri
- Registerlardan ve komut alanlarından belirlenen tüm load/store adreslerine sahip az sayıda adresleme modu
- Program döngülerinin çalışmasını iyileştirmek için otomatik artış ve otomatik azalış (Auto-increment and auto-decrement) adresleme modları kullanılır
- Komutların koşullu yürütülmesi, koşullu dallanma komutlarına olan ihtiyacı en aza indirir, böylece boru hattının boşaltılması (pipeline flushing) azaltıldığından boru hattı verimliliğini artırır

ARM işlemci organizasyonu, özellikle ARM mimarisinin farklı sürümlerine dayandığında, bir implementasyondan diğerine büyük ölçüde farklılık gösterir. Şekil 14.25 basitleştirilmiş, genel bir ARM organizasyonu görünümü sunmaktadır. Bu şekilde, oklar veri akışını gösterir. Her kutu, işlevsel bir donanım birimini veya bir depolama birimini temsil eder.

ARM veri işleme komutları tipik olarak Rn ve Rm olmak üzere iki kaynak registerine ve tek bir sonuç veya hedef registeri olan Rd 'ye sahiptir.

Bir işlemin sonuçları, hedef registerine geri beslenir. Load/store komutları, bu iş için bellek adresini oluşturmak amacıyla aritmetik birimlerin çıktısını da kullanabilir.

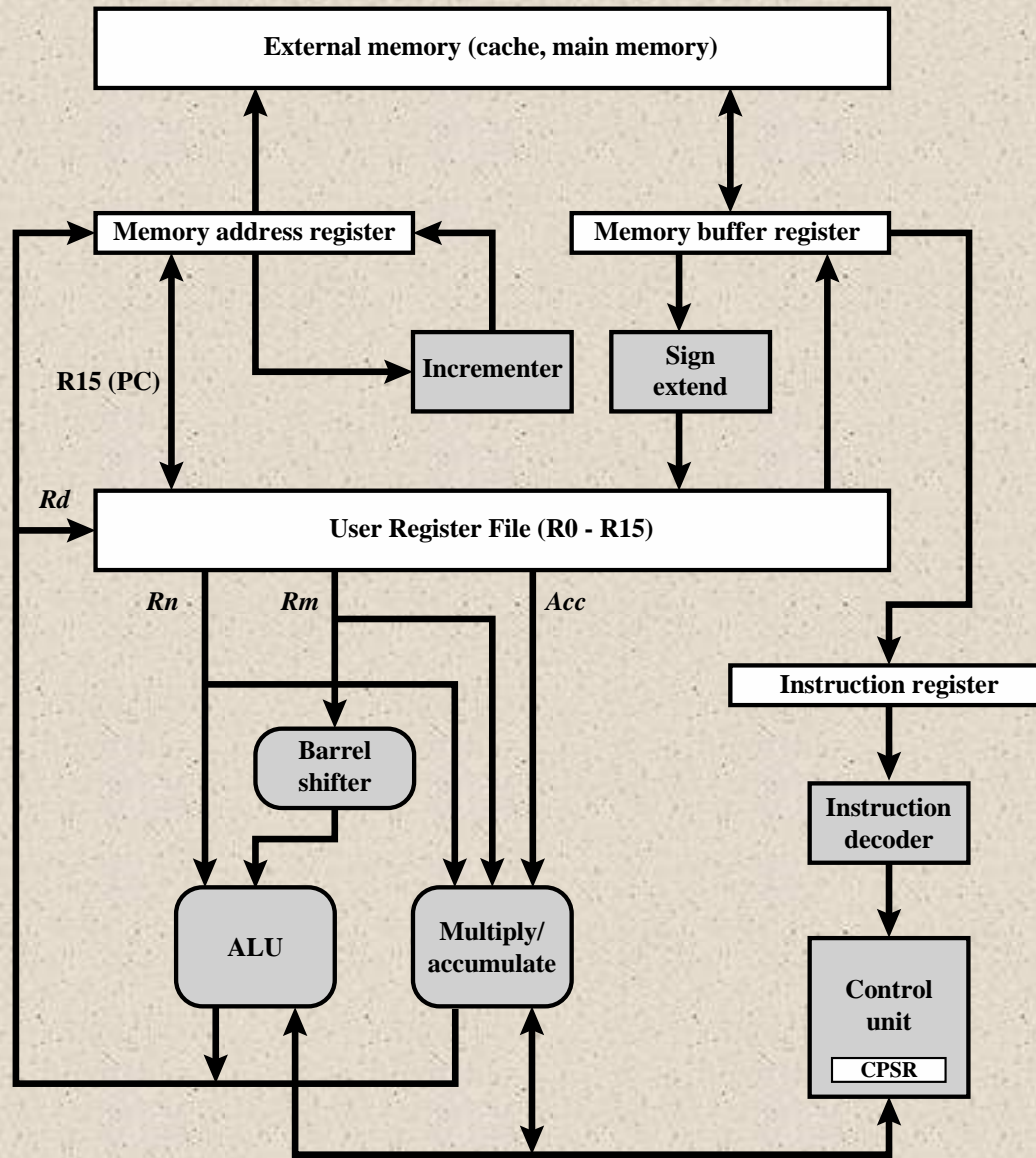


Figure 14.25 Simplified ARM Organization

Veriler, bir veri yolu aracılığıyla harici bellek ile işlemci arasında aktarılır. Aktarılan değer, bir «load» veya «store» komutunun bir sonucu olarak bir veri ögesidir veya bir komut getirir. Getirilen komutlar, bir kontrol ünitesinin kontrolü altında yürütülmeden önce bir komut kod çözücünden geçer. Kontrol ünitesi, boruhattı lojigini içerir ve işlemcinin tüm donanım elemanlarına kontrol sinyalleri sağlar. Veri ögeleri, 32 bitlik registerlardan oluşan register file'a yerleştirilir. 2'ye tümleyen sayılar olarak kabul edilen bayt veya yarım kelime ögeleri işaret bitiyle 32 bit'e genişletilir (sign-extended to 32 bits).

Processor Modes



Exception Modes

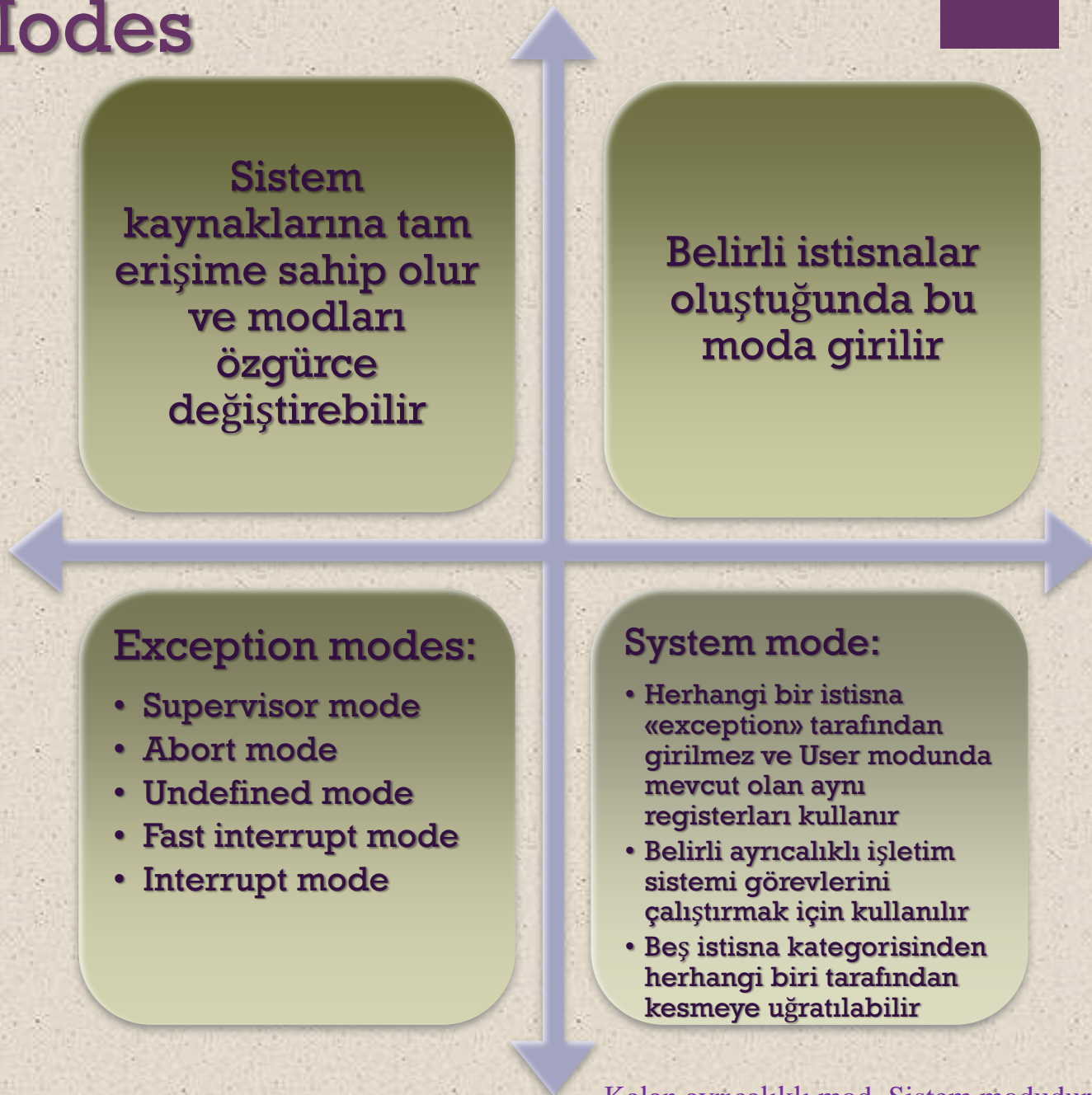


Supervisor mode: Genellikle işletim sisteminin çalıştığı şeydir. İşlemci bir yazılım kesme komutu ile karşılaştığında girilir. Yazılım kesmeleri, ARM'da işletim sistemi servislerini çağırmanın standart bir yoludur.

Abort mode : Bellek hatalarına yanıt olarak girilir.

Undefined mode : İşlemci, ana integer çekirdeği veya yardımcı işlemcilerden biri tarafından desteklenmeyen bir komutu yürütmeye çalıştığında girilir.

Fast interrupt mode : İşlemci belirlenen hızlı kesme kaynağından bir kesme sinyali aldığında girilir. Hızlı kesme kesintiye uğratılamaz, ancak hızlı bir kesme normal bir kesmeyi kesebilir.



Kalan ayrıcalıklı mod, Sistem modudur.



Modes						
User	Privileged modes					
	System	Exception modes				
		Supervisor	Abort	Undefined	Interrupt	Fast Interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13 (SP)	R13 (SP)	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14 (LR)	R14 (LR)	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

Shading indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode.

SP = stack pointer

CPSR = current program status register

LR = link register

SPSR = saved program status register

PC = program counter

Figure 14.26 ARM Register Organization

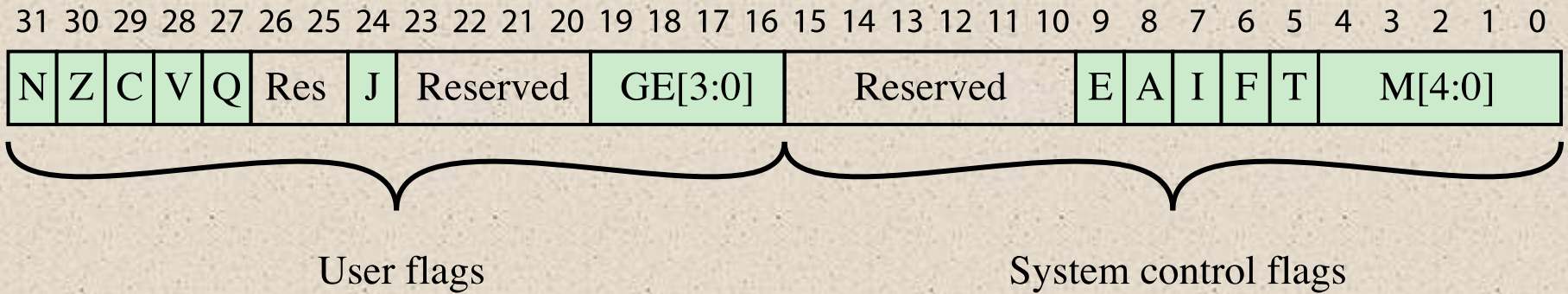


Figure 14.27 Format of ARM CPSR AND SPSR



Table 14.4

ARM Interrupt Vector

Exception type	Mode	Normal entry address	Description
Reset	Supervisor	0x00000000	Occurs when the system is initialized.
Data abort	Abort	0x00000010	Occurs when an invalid memory address has been accessed, such as if there is no physical memory for an address or the correct access permission is lacking.
FIQ (fast interrupt)	FIQ	0x0000001C	Occurs when an external device asserts the FIQ pin on the processor. An interrupt cannot be interrupted except by an FIQ. FIQ is designed to support a data transfer or channel process, and has sufficient private registers to remove the need for register saving in such applications, therefore minimizing the overhead of context switching. A fast interrupt cannot be interrupted.
IRQ (interrupt)	IRQ	0x00000018	Occurs when an external device asserts the IRQ pin on the processor. An interrupt cannot be interrupted except by an FIQ.
Prefetch abort	Abort	0x0000000C	Occurs when an attempt to fetch an instruction results in a memory fault. The exception is raised when the instruction enters the execute stage of the pipeline.
Undefined instructions	Undefined	0x00000004	Occurs when an instruction not in the instruction set reaches the execute stage of the pipeline.
Software interrupt	Supervisor	0x00000008	Generally used to allow user mode programs to call the OS. The user program executes a SWI instruction with an argument that identifies the function the user wishes to perform.

+ Summary

Chapter 14

- Processor organization
- Register organization
 - User-visible registers
 - Control and status registers
- Instruction cycle
 - The indirect cycle
 - Data flow
- The x86 processor family
 - Register organization
 - Interrupt processing

Processor Structure and Function

- Instruction pipelining
 - Pipelining strategy
 - Pipeline performance
 - Pipeline hazards
 - Dealing with branches
 - Intel 80486 pipelining
- The Arm processor
 - Processor organization
 - Processor modes
 - Register organization
 - Interrupt processing