

Bilgisayar Mimarisi

Lisans: <http://creativecommons.org/licenses/by-nc-nd/3.0/>

## 2. İş Hattı (Pipeline)

**İş hattında (pipeline)** birden fazla iş (örneğin komutlar) paralel olarak aynı anda yürütülür.

Bir iş hattının verimli olarak çalışabilmesi için

- Farklı veriler üzerinde defalarca tekrarlanan işler (task) olması gerekir,
- İşler paralel yürütülebilen küçük alt işlere bölünebilmeli.

**İş hattına örnek:** Bir otomobil fabrikasındaki üretim/montaj bandı Burada iş/görev (task) bir otomobilin montajının yapılmasıdır. Bu iş farklı otomobiller için sürekli tekrar edilir. İş (otomobilin montajı), küçük adımlardan oluşur; kapıların takılması, tekerleklerin montajı, camların takılması. Bu adımların her biri için iş hattında (montaj bandı) bir istasyon oluşturulur. Bu istasyonlarda aynı anda paralel olarak farklı otomobiller üzerinde çalışılır. Örneğin i. işçi bir otomobilin camını takarken aynı anda (i+1). sıradaki işçi bir önceki otomobilin tekerleklerini takmaktadır.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.1

Bilgisayar Mimarisi

### 2.1 Bir iş hattının genel yapısı:

1. Segman (katman) (Segment, stage, layer) 2. Segman k. Segman

- Her katman (işlem birimi) belli, sabit bir işi yapar.
- Her saat çevriminde (clock cycle) işlem birimi farklı veriler (iş) üzerinde çalışır. (Saat işareti konusunda Sayısal Devreler Ders Notları Bölüm 6 'da bilgi bulabilirsiniz.)
- R1, R2 gibi saklayıcılar ara sonuçları tutarlar.
- Tüm segmanlar ortak bir saat işareti ile denetlenirler ve eş zamanlı çalışırlar.
- Bir önceki verinin bütün adımları tamamlanmadan (sonuç üretilmeden) önce iş hattının girişinden yeni veriler alınır.
- İş hattının bütün segmanları dolduktan sonra her saat çevriminde çıkışta yeni bir sonuç üretilir.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.2

Bilgisayar Mimarisi

**Örnek:** A, B ve C dizisinin elemanları önce bellekten okunacak ardından aşağıdaki işlem yapılacaktır.  $A_i * B_i + C_i \quad i=1,2,3, \dots$

1. Katman (layer, segment) Okuma

2. Katman Çarpma ve okuma

3. Katman Toplama

Saat

Sonuç

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.3

Bilgisayar Mimarisi

Bu örnekte görev üç alt işleme bölünmüştür: Okuma, çarpma, toplama

Üç segmanlı olarak tasarlanan iş hattının çalışması:

Saat Çevrimi	1. Segman	2. Segman	3. Segman
1	R1	R2	R3
2	A <sub>1</sub>	B <sub>1</sub>	-
3	A <sub>2</sub>	B <sub>2</sub>	A <sub>1</sub> *B <sub>1</sub>
4	A <sub>3</sub>	B <sub>3</sub>	A <sub>2</sub> *B <sub>2</sub>
5	A <sub>4</sub>	B <sub>4</sub>	A <sub>3</sub> *B <sub>3</sub>
6	A <sub>5</sub>	B <sub>5</sub>	A <sub>4</sub> *B <sub>4</sub>
7			A <sub>1</sub> *B <sub>1</sub> + C <sub>1</sub>
8			A <sub>2</sub> *B <sub>2</sub> + C <sub>2</sub>
9			A <sub>3</sub> *B <sub>3</sub> + C <sub>3</sub>

Not: Verinin önceden hazır olduğu veya bellek okuma süresinin diğer işlemlere göre çok kısa olduğu sistemlerde bellekten okuma ayrı bir alt işlem olarak ele alınmaz. Bu durumda sadece aritmetik işlemi yapan iş hattı 3 yerine 2 katmanlı olarak tasarlanabilir.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.4

Bilgisayar Mimarisi

### 2.2 4 Segmanlı Bir İş Hattının Uzak-Zaman Diyagramı (Space-Time Diagram)

Bir iş hattında belli bir anda hangi işin hangi segmanda işlem gördüğünü göstermek için uzak-zaman diyagramları (zamanlama diyagramı) kullanılır. Aşağıdaki tabloda, saat çevrimleri (adımlar) sütunlara, segmanlar satırlara, o anda yapılan iş (task) (veya işleme giren veriler) de tablonun içine yazılmıştır.

Zaman	Saat Çevrimi (adımlar)						
	1	2	3	4	5	6	7
1	T1	T2	T3	T4	T5	T6	
2		T1	T2	T3	T4	T5	T6
3			T1	T2	T3	T4	T5
4				T1	T2	T3	T4

İnci iş (T1) 4 saat çevrimi (segman sayısı k=4) sonunda tamamlandı.

k.'dan sonraki her saat çevriminde yeni bir iş tamamlanır.

Dört iş (T4) 7 saat çevriminde tamamlanmıştır.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.5

Bilgisayar Mimarisi

### 4 Segmanlı Bir İş Hattının Uzak-Zaman Diyagramı (Space-Time Diagram) Devamı

Uzak-zaman diyagramları farklı şekilde de oluşturulabilir. Aşağıdaki diyagramda saat çevrimleri (adımlar) sütunlara, veriler (işler) satırlara, o anda etkin olan segman da tablonun içine yazılabilir.

Zaman	Saat Çevrimi (adımlar)						
	1	2	3	4	5	6	7
T1	S1	S2	S3	S4*			
T2		S1	S2	S3	S4*		
T3			S1	S2	S3	S4*	
T4				S1	S2	S3	S4*

İnci iş (T1) 4 saat çevrimi (segman sayısı k=4) sonunda tamamlandı.

k.'dan sonraki her saat çevriminde yeni bir iş tamamlanır.

Dört iş (T4) 7 saat çevriminde tamamlanmıştır.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.6

Bilgisayar Mimarisi Lisans: <http://creativecommons.org/licenses/by-nc-nd/3.0/>

### 2.3 İş Hattının sağladığı hızlanma (Speedup):

İş hattındaki tüm segmanlar eşzamanlı (*synchronous*) işlem yaptığından, saat işaretinin periyot uzunluğu (çevrim zamanı) (*cycle time*) **en yavaş segmanın** gerek duyduğu çalışma zamanı (gecikmesi) tarafından belirlenir.

**Çevrim zamanı (cycle time)** ( $t_p$ ) saat işaretinin periyodu  $t_p$  aşağıdaki gibi hesaplanır:

$$t_p = \max(t_i) + d_r = \tau_M + d_r$$

$t_p$ : çevrim zamanı (*cycle time*)  
 $t_i$ : i. katmandaki devrenin gecikmesi  
 $\tau_M$ : en büyük gecikme (en yavaş katman)  
 $d_r$ : saklayıcıların gecikmesi

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.7

Bilgisayar Mimarisi

### Hızlanma (Speedup):

k: İş hattındaki segman sayısı  
 $t_p$ : saat periyodu (En yavaş birime göre ayarlanır)  
n: İş sayısı (işin tekrar sayısı)  
İnci için (T1) tamamlanması için k adet saat darbesi gereklidir.  
Buna göre İnci için tamamlanma süresi:  $T(1) = k \cdot t_p$   
Kalan n-1 işin tamamlanması için (n-1) çevrim gereklidir. Süre:  $(n-1)t_p$   
Tüm işlerin (n adet) toplam süresi:  $(k+n-1)t_p$   
 $t_n$ : İş hattı kullanılsaydı bir işin süresi

Hızlanma (Speedup):  $S = \frac{\text{İş hattı olmadan gereken süre}}{\text{İş hattı ile gerekli olan süre}} \quad S = \frac{n \cdot t_n}{(k+n-1) \cdot t_p}$

İş sayısı çok artarsa:  $n \rightarrow \infty$

$$S = \frac{t_n}{t_p}$$

Eğer  $t_n = k \cdot t_p$  varsayımı yapılırsa  
(ana işi k adet eşit süreli küçük alt işleme bölmek mümkünse):

$$S_{max} = k \quad (\text{Teorik maksimum hızlanma})$$

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.8

Bilgisayar Mimarisi

### Hızlanma ile ilgili yorumlar:

İş hattının verimini arttırmak için bir işi mümkün olduğu kadar **eşit** (en azından yakın) sürelerdeki **küçük** alt işlere bölmek gerekir.

Eğer alt işlemlerin süreleri kısa olursa saat işaretinin çevrim süresi de kısalmır. Hatırlatma: en yavaş birim çevrim süresini belirler.

İş hattındaki katman sayısının etkileri:

Olumlu:

- Eğer iş **çok sayıda kısa süreli** alt işlere bölünebiliyorsa segman sayısını arttırmak saat işareti hızlandırır ve iş hattının verimi artırır.

$$S_{max} = k$$

Olumsuz:

- İş hattının maliyeti artar. Her katmanın sonuna yerleştirilen saklayıcılar maliyet, enerji tüketimi, boyut açısından sisteme yükler getirir.
- İlk baştaki 1. iş için bekleme süresi artar.  $T(1) = k \cdot t_p$
- Dallanma cezaları artar. Dallanma cezaları "2.5 İş Hattında Oluşabilen Sorunlar" bölümünde ele alınacaktır.

Bir iş hattı tasarlanırken bütün bu olumlu ve olumsuz noktalar birlikte dikkate alınmalıdır.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.9

Bilgisayar Mimarisi

### İş alt işlemlere bölmenin hızlanma üzerindeki etkisi:


Eğer ana iş kısa süreli küçük alt işlere bölünebiliyorsa sisteme daha hızlı bir saat işareti uygulanabilir.

Örnek olarak toplam süresi 100 ns olan bir T işini ele alalım.

Bu işin farklı şekillerde alt işlere bölünebildiği varsayılmıştır.

**Durum A:** İş 2 eşit katmana bölünüyor.

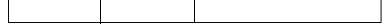
$S1=50\text{ns} \quad S2=50\text{ns}$

T: 

Saklayıcıların gecikmesinin 5 ns olduğu varsayılırsa saat çevrimi  $t_p = 50+5 = 55 \text{ ns}$

**Durum B:** İş 3 adet dengesiz katmana bölünüyor.

$S1=25\text{ns} \quad S2=25\text{ns} \quad S3=50\text{ns}$

T: 

Saat çevrimi  $t_p = 50+5 = 55 \text{ ns}$  (en yavaş katman  $\tau_M = 50\text{ns}$ )

İş hattında daha fazla katman olmasına rağmen Durum A'ya göre bir hızlanma sağlanmamıştır.

Ayrıca iş hattının maliyeti de artmıştır.

İlk işin tamamlanma süresi uzamıştır.  $T(1) = k \cdot t_p$

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info


2005-2014 Dr. Feza BUZLUCA 2.10

Bilgisayar Mimarisi

### İş alt işlemlere bölmenin hızlanma üzerindeki etkisi: (devamı)

**Durum C:** İş 3 adet yakın süreli katmana bölünüyor.

$S1=30\text{ns} \quad S2=30\text{ns} \quad S3=40\text{ns}$

T: 

Saat çevrimi  $t_p = 40+5 = 45 \text{ ns}$  (en yavaş katman  $\tau_M = 40\text{ns}$ )

Saat işareti Durum A ve B'ye göre hızlanmıştır.

**Sonuç:**

İş hattının hızlanma sağlayabilmesi için işi **çok sayıda, kısa süreli ve dengeli** alt işe bölmek gerekir.

Örneğin yukarıdaki örnek iş, her biri 20ns süreli 5 adet alt işleme bölünebilirse saat işaretinin periyodu 25ns olur.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.11

Bilgisayar Mimarisi

### 2.4 Komut İş Hattı (Instruction Pipeline)

#### Komut düzeyinde paralellik (Instruction-Level Parallelism)

Merkezi işlem birimleri her komutu işlerken belli alt işlemleri tekrar ederler.

Bir komutun MİB'te işlenme sürecine komut çevrimi (*instruction cycle*) denir.

Komut çevriminin genel olarak alt çevrimleri: Komut alma ve çözme, operand alma, yürütme, kesme.

En basit iş hattı yapısı iki segmanlı olarak kurulabilir:

1) Komut alma ve çözme 2) Operandları alma ve komut yürütme

Komut yürütme birimi belleğe erişmediği zamanlarda komut alma birimi sıradaki komutu bellekten alarak bir komut saklayıcısına yazar.

Böylece o andaki komut yürütülürken sonraki komut bellekten paralel olarak okunur.

Komutların bu şekilde paralel işlenmesine Komut düzeyinde paralellik (*Instruction-Level Parallelism*) denir.

Hatırlatma: iş hattındaki hızlanmayı arttırmak için iş hattını çok sayıda kısa süreli katmandan oluşturmak gerekir.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.12

Bilgisayar Mimarisi Lisans: <http://creativecommons.org/licenses/by-nc-nd/3.0/>

**Komut İş Hattı (Instruction Pipeline) (devamı)**

Verimi arttırmak için komut işleme daha küçük alt işlemlere bölünerek 6 segmanlı bir iş hattı oluşturulabilir:

1. Komut alma (Fetch instruction) (FI):
2. Komut çözme (Decode instruction) (DI):
3. Operand adresi hesabı (Calculate addresses of operands) (CO)
4. Operand alma (Fetch operands) (FO)
5. Komut yürütme (Execute instruction) (EI)
6. Sonucu yazma (Write operand) (WO)

Bu kadar ayrıntılı bölmeleme aşağıdaki problemler nedeniyle verimli olmaz:

- Segmanların süreleri farklıdır.
- Her komut bütün alt işlemlere gerek duymaz.
- Değişik segmanlar aynı anda bellek erişimine gerek duyar.

Bu nedenle bazı alt işlemler birleştirilerek komut iş hatları daha az (örneğin 4 veya 5), dengeli segmanla oluşturulur.

Örneğin 80486' da 5 segmanlı bir iş hattı bulunmaktaydı.

Daha çok segmana sahip iş hattı içeren işlemciler de bulunmaktadır.

Örneğin Pentium 4 ailesinin işlemcilerinde 20 segmanlı iş hatları bulunmaktadır.

Bu işlemcilerde komut çevrimin alt işlemleri de daha küçük işlemlere bölünmüştür.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.13

Bilgisayar Mimarisi

**Örnek: Dört Segmanlı Komut İş Hattı**

1. FI (Fetch Instruction): Komut alma
2. DA (Decode, Address): Komutu çöz, efektif adresi hesapla
3. FO (Fetch Operand): Operand al
4. EX (Execution): Yürütme (İşlem yapılır, saklayıcılar güncellenir)

Komut alma ve operand alma işlemlerinin aynı anda yapılabilmesi için komut ve veri belleklerinin ayrı oldukları varsayılmıştır.

Komut iş hattının zaman diyagramı (ideal durum):

Saat çevrimi	Adımlar	1	2	3	4	5	6	7	8
1	FI	DA	FO	EX					
2		FI	DA	FO	EX				
3			FI	DA	FO	EX			
4				FI	DA	FO	EX		
5					FI	DA	FO	EX	

İlk komut tamamlandı, 4 çevrim

Bir saat çevrimi sonra ikinci komut tamamlandı.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.14

Bilgisayar Mimarisi

**2.5 İş Hattında Oluşabilen Sorunlar (Pipeline Hazards) (Conflicts)**

**2.5.1 Denetim Sorunları (Control Hazards):**

**Dallanma ve Kesmeler (Branches, Interrupts)**

İş hattında komutlar paralel olarak yürütüldüğünden bir dallanma komutu işlenirken bellekte ondan sonra gelen ancak dallanma nedeniyle yürütülmeyecek olan komut (veya komutlar) da iş hattına alınmış olur.

Eğer önlem alınmazsa programın mantığı gereği yürütülmemesi gereken komutlar da yürütülmüş olur.

**Örnek:**

1. Komut\_1
2. JUMP Hedef
3. Komut\_3
4. Hedef Komut\_4

Koşulsuz dallanma komutu (BRA)

Bellekte dallanmanın peşindeki komut (next instruction). Programa göre yürütülmemesi gerekir.

Dallanmanın hedefi, dallanmadan sonra yürütülecek komut (target instruction).

Koşulsuz dallanma komutu JUMP işlenirken Komut\_3 de iş hattına girmiş olur.

Programın yanlış çalışmasını önlemek için iş hattını durdurmak (stall) ve Komut\_3 çalışmadan önce iş hattını boşaltmak gerekir.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.15

Bilgisayar Mimarisi

**a. Koşulsuz Dallanma (Unconditional Branch)**

Saat çevrimi	Adımlar	1	2	3	4	5	6	7
1	FI	DA	FO	EX				
2		FI	DA	FO	EX			
3			FI	DA	FO	EX		
4				FI	DA	FO	EX	

Komut çözüldüğünde dallanma olduğu anlaşılır.

Dallanılacak adres alınıyor (Mutlak ya da bağıl).

PC (program sayacı) güncelleniyor. PC = Hedef (dallanılacak adres)

Dallanmadan sonra gidilen hedef komut (Dallanmanın hedefi)

**Sorun:** Bu komut boşuna alındı. Bu komut yürütülmemeli. İş hattından silinecek.

Dallanma cezası (Branch penalty) İş hattı durdurulacak ve boşaltılacak.

Koşulsuz dallanma komutu çözüldüğü (anlaşıldığı) anda olası önlemlerden biri iş hattına yeni komut alma işlemini (FI segmanını) durdurmaktadır.

Dallanma komutunun yürütülmesi sonucu hedef komutun adresi hesaplanıp program sayacı (PC) güncellendikten sonra komut alma işlemi tekrar başlar.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.16

Bilgisayar Mimarisi

**b. Koşullu Dallanma (Conditional Branch):**

Koşullu dallanma komutları yürütülürken iki durum oluşur;

1. Koşul yanlış (dallanma olmaz), 2. Koşul doğrudur (dallanma olur)

**b1. Koşullu dallanma (koşul yanlışsa):**

Koşul doğru değilse iş hattını durdurmaya veya boşaltmaya gerek yoktur, çünkü program bir sonraki komut ile devam edecektir.

Saat çevrimi	Adımlar	1	2	3	4	5	6
1	FI	DA	FO	EX			
2		FI	DA	FO	EX		
3			FI	DA	FO	EX	

Önceki komut bayrakları (koşulları) belirliyor.

PC değişmedi. Dallanma gerçekleşmedi.

Dallanmanın peşindeki komut yürütülüyor.

Koşula bakılmaksızın bir sonraki komut alındı.

Dallanma olmadığı için boşaltılmayacak (Ceza yok)

Koşulun doğru olup olmadığı ancak dallanma komutu yürütüldükten sonra belli olur.

Eğer Koşul doğru çıkarsa programın yanlış çalışmasını önlemek için çözüm yöntemleri uygulamak gerekir (Koşulsuz dallanmada olduğu gibi).

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.17

Bilgisayar Mimarisi

**b2. Koşullu dallanma (koşul doğru ise):**

Saat çevrimi	Adımlar	1	2	3	4	5	6	7
1	FI	DA	FO	EX				
2		FI	DA	FO	EX			
3			FI	DA	FO	EX		
4				FI	DA	FO	EX	
5					FI	DA	FO	EX
6						FI	DA	EX

Koşul doğru. Dallanılacak adres (Hedef) PC'ye yazılıyor. PC = Hedef. İş hattı boşaltılmalı.

Bu segmanlar durdurulur.

İş hattı boşaltılıyor.

Dallanma cezası: 3 saat çevrimi

Dallanma ile gidilen komut (hedef)

Dallanma cezasının süresi iş hattındaki segmanların sayısına ve işlevlerine bağlıdır.

Bu örnek iş hattında dallanma cezası 3 saat çevrimidir; ancak başka yapılarıdaki iş hatlarında bu süre farklı olabilir.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.18

Bilgisayar Mimarisi

Lisans: <http://creativecommons.org/licenses/by-nc-nd/3.0/>

### 2.5.2. Kaynak Çatışması (Resource Conflict), Yapısal Sorun (Structural Hazard):

İş hattında aynı anda işlenen iki komut aynı kaynağa (bellek, ALU) gerek duyarsa kaynak çatışması oluşur.

a) Bellek çatışması: İki farklı segmanda aynı bellek modülüne erişilmek istenirse. Örneğin komut alma ile operand okuma/yazma aynı anda olamaz.

Çözümler:

- Komutların belli bölümleri paralel değil, peş peşe seri işlenir. İş hattının belli segmanları durdurulur. Bu çözüm performansı düşürür.
- Harvard mimarisi: Komutlar ve veriler için ayrı bellek
- Komut kuyruğu veya cep bellek: Bir komut işlenirken belleğe erişilmediği anlarda sıradaki komutlar bellekten okunarak bir kuyruğa yazılır.

b) İşlem birimi (ALU, FPU) çatışması: İki farklı segmanda aynı işlem birimine (Arithmetic Logic Unit- ALU, Floating Point Unit- FPU) gerek duyulursa.

Çözümler:

- İşlem birimlerinin sayısı artırılır. Örneğin adres hesabı ve veri işleme için iki ayrı ALU kullanılır.
- İşlem birimleri de iş hattı olarak tasarlanarak paralellik sağlanır. Örnek FPU

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.19

Bilgisayar Mimarisi

### 2.5.3. Veri Çatışması (Data Conflict), Veri Bağımlılığı (Data Dependency):

İki farklı komut aynı bellek bölgesine erişmek istediğinde oluşabilir. Bu sorun çözülmezse program yanlış sonuç üretebilir.

a) **Operand bağımlılığı:**  
Bir komutun kaynak operandı diğer bir komutun sonucuna bağlıdır.

Örnek (68000)

Saat çevrimi	1	2	3	4	5
ADD.W D1, DATA	FI	DA	FO	EX	
MOVE.W DATA, (A0)		FI	DA	FO	EX

DATA güncelleniyor

Operand bağımlılığı

DATA'nın geçerli olmayan eski değeri alınıyor.

b) **Adres Bağımlılığı:**  
Veri çatışması (bağımlılığı) adres saklayıcılarında (işaretçi - pointer) da oluşabilir.

Örnek (68000)

ADDA.W #2, A0  
MOVE.B (A0)+, D0

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.20

Bilgisayar Mimarisi

### 2.5.3. Veri Çatışması (Data Conflict), devamı

Üç farklı tipte veri çatışması (data hazard) oluşabilir:

- Yazmadan sonra okuma (Read after write) (RAW):** Bu türe gerçek bağımlılık (true dependency) da denir.  
Bir komut bir saklayıcıyı veya bellek gözüni değiştirmektedir. Daha sonra gelen bir komut da aynı saklayıcı veya bellek gözüni okumaktadır.  
Eğer iş hattı nedeniyle okuma işlemi yazmadan önce yapılırsa veri çatışması sorunu oluşur.
- Okumadan sonra yazma (Write after read) (WAR):** Anti bağımlılık da denir.  
Bir komut bir saklayıcıyı veya bellek gözüni okumaktadır. Daha sonra gelen bir komut da aynı saklayıcı veya bellek gözüne yazmaktadır.  
Eğer yazma işlemi okumadan önce yapılırsa veri çatışması sorunu oluşur.
- Yazmadan sonra yazma (Write after write) (WAW):** Çıkış bağımlılığı da denir.  
İki komut aynı saklayıcıyı veya bellek gözüne yazmaktadır.  
Eğer yazma işlemleri programda belirtilenden farklı sırada olursa veri çatışması sorunu oluşur.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.21

Bilgisayar Mimarisi

### 2.6 Veri çatışması sorununun çözümleri:

- Operand yönlendirme (Operand forwarding or Bypassing):**  
ALU'nun çıkışı ile girişi arasında ek bir bağlantı (bypass) oluşturulur. Operand yönlendirme yöntemini açıklamak için RISC işlemcilerde sık kullanılan aşağıdaki komut iş hattı yapısı kullanılacaktır.

Örnek iş hattı:

- FI (Fetch Instruction)
- DO (Decode, Operand (register) fetch): Operand (saklayıcı değeri) alınır.
- EX (Execution): Saklayıcılar üzerinde işlem yapılır.
- WO (Write Operand): Sonuç saklayıcılara yazılır.

Örnek:

Saat çevrimi	1	2	3	4	5
ADD R1, R2, R3; R1 ← R2+R3	FI	DO	EX	WO	
SUB R4, R5, R1; R4 ← R5-R1		FI	DO	EX	WO

R1 güncelleniyor

Yazmadan sonra okuma (RAW) bağımlılığı

SUB komutu R1'i henüz ADD komutu tarafından güncellemeden önce okuyor.

R1'in geçerli olmayan önceki değeri okunuyor.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.22

Bilgisayar Mimarisi

### Operand yönlendirme (Bypassing) devamı:

Yönlendirme (Bypass)

s girişi, iş hattının çatışma sezme (hazard detection) birimi tarafından denetlenir. ALU'nun girişine ya saklayıcılardan gelen değeri ya da ALU'nun çıkışından doğrudan gelen değeri (bypass) yönlendirir.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.23

Bilgisayar Mimarisi

### Operand yönlendirme (Bypassing) devamı:

Eğer çatışma sezme birimi bir önceki ALU işleminin hedef saklayıcısının şimdiki ALU işleminin kaynağı olduğunu sezerse denetim birimi ALU'nun girişine saklayıcıdan gelen değeri değil, ALU'nun çıkışından doğrudan gelen değeri (bypass) yönlendirir.

Örnek:

Saat Çevrimi	1	2	3	4	5
ADD R1, R2, R3; R1 ← R2+R3	FI	DO	EX	WO	
SUB R4, R5, R1; R4 ← R5-R1		FI	DO	EX	WO

R1'in geçerli olmayan önceki değeri okunuyor. Bu geçersiz değer EX segmanında kullanılmayacak.

İş hattı denetim birimi ALU'nun girişine saklayıcıdan DO'da alınan geçersiz değeri değil, ALU'nun çıkışından doğrudan gelen değeri (bypass) yönlendirir.

Eğer sorunu operand yönlendirme ile çözmek mümkün olursa iş hattını durdurmaya gerek olmaz ve performans düşmez.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.24

Bilgisayar Mimarisi Lisans: <http://creativecommons.org/licenses/by-nc-nd/3.0/>

## 2.6 Veri çatışması sorununun çözümleri (devamı):

- **Donanım Kilidi (Hardware interlock):**  
Bir donanım tüm komutları izler. Veri bağımlılığı olan komutların iş hattına girmesi geciktirilir.  
İş hattının komut alma segmanı (FI) gerekli saat çevrimi kadar durdurulur.
- **Derleyici Tabanlı Çözümler:**  
Ek donanıma gerek yoktur, yazılım temelli çözümlerdir.  
- **Gecikmeli Yükleme (Delayed Load):**  
Derleyici eğer mümkünse veri çatışmasına neden olan komutların yerini değiştirir. Burada programın algoritmasının değişmemesi sağlanır.  
Bu mümkün değilse bağımlı komutlar arasına NOP (No Operation) komutu koyar.  
  
Derleyici tabanlı çözümler "2.8 RISC İş hattı" bölümünde ele alınmıştır.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.25

Bilgisayar Mimarisi

## 2.7 Dallanma sorunlarının ele alınması:

Programlardaki dallanma komutları nedeniyle iş hatlarının durulması ve boşaltılması gerektiği daha önce gösterilmişti (yansı 2.16, 2.18).

Aksi durumda işlemci program gereği atlanması gerek komutları da yürütür. İş hattını durdurmak veya boşaltmak sistemin performansını düşürür.

Dallanma komutunun bellekte peşinden gelen komut yerine, dallanmanın hedefi olan komutu iş hattına almak performans düşüşünü önlemek için yararlı olur.

Burada temel sorun koşullu dallanmada ortaya çıkar, çünkü komut yürütülünceye kadar dallanmanın gerçekten olup olmayacağı belli değildir (yansı 2.18).

Bu problemi çözmek için **dallanma öngörü** (branch prediction) yöntemleri kullanılır.

Diğer bir problem ise dallanmanın hedef adresinin dallanma komutunun ancak yürütme çevriminde belli olmasıdır.

Bu nedenle hangi hedef komutun iş hattına alınacağı önceden belli değildir. Bu problemi çözmek için **dallanma hedef tablosu** (branch target table) kullanılır.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.26

Bilgisayar Mimarisi

## Dallanma problemlerine ilişkin çözümler:

### 2.7.1 Önceden komut alma (Target Instruction prefetch):

Koşullu dallanmalarda hem dallanmadan sonraki komut (koşul yanlış ise kullanılacak) hem de dallanma ile gidilmesi olası olan hedef komut (koşul doğru ise kullanılacak) iş hattına alınır.

Hedef komutu önceden belirlemek için dallanma hedef tablosu kullanılır.

**Dallanma hedef tablosu (branch target table (buffer)):**  
Son çalışan belli sayıdaki koşullu dallanma komutunun adresleri ve son çalıştıklarında nereye gidildiği (gidilen yerdeki bir kaç komut) bir çağrışımlı bellekte (associative memory) tutulur.  
Böylece gidilecek adres hesaplanmadan önce dallanma komutundan sonraki komutlara erişilebilir.  
Tutulan komut sayısı tablo boyutu ile sınırlıdır.

Dallanma Komutu adresi	Hedef adres

Programda en son çalışan belli sayıdaki her koşullu dallanma komutu için bir satır vardır.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.27

Bilgisayar Mimarisi

## 2.7.2 Dallanma öngörüsü (Branch prediction):

### Statik dallanma öngörüsü stratejileri:

- Her zaman "dallanma yok" öngörüsü: Her zaman dallanma olmayacağı öngörülür ve bellekte dallanmadan sonra gelen komut iş hattına alınır.
- Her zaman "dallanma var" öngörüsü: Her zaman dallanma olacağı öngörülür ve dallanmanın hedefi olan komut iş hattına alınır.

Programların davranışını inceleyen çalışmalar, koşullu dallanmaların %50'sinden fazlasında dallanmanın gerçekleştiğini göstermişlerdir.

Bu nedenle "her zaman dallanma var" öngörüsü performans açısından daha iyi sonuç vermektedir.

### Dinamik dallanma öngörüsü stratejileri:

Dinamik dallanma öngörüsü stratejileri o anda çalışan programdaki tüm koşullu dallanma komutları ile ilgili istatistik tutarak dallanmanın olup olmayacağını öngörmeye çalışırlar.

Programdaki her koşullu dallanma komutu ile bir veya daha fazla sayıda **öngörü biti** (veya sayaç) (prediction bits) ilişkilendirilir.

Komutların geçmişi ile ilgili bilgi (dallanma gerçekleşme oranı) sağlayan bu bitler bir dallanma geçmişi tablosunda (branch history table) tutulur (Bkz. 2.29).

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.28

Bilgisayar Mimarisi

## Dallanma hedef tablosu ve dallanma geçmişi tablosu (branch history table):

Öngörü bitleri hızlı erişilebilen bir bellekte oluşturulan dallanma geçmişi tablosunda (branch history table - BHT) tutulur.

Dallanma geçmişi tablosunda, en son çalışan belli sayıdaki her koşullu dallanma komutu için komutun bellek adresi, hedef adresi ve durum (öngörü) bitleri tutulur. Öngörü bitleri dallanma komutunun her çalışmasında dallanma olup olmamasına göre değer alırlar.

Koşullu dallanma komutu tekrar çalıştığında bu bitler iş hattı denetim birimi tarafından karar vermek için kullanılır.

Eğer "dallanma VAR" öngörüsü yapılırsa dallanma komutu yürütülmeden önce tablodaki hedef adresi kullanılarak gidilecek olan komut iş hattına alınabilir.

Dallanma Komutu adresi	Hedef adres	Durum (öngörü) bitleri

Programda son çalışmış olan koşullu dallanma komutları

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.29

Bilgisayar Mimarisi

## 1 bit dinamik dallanma öngörü yöntemi :

Her koşullu dallanma komutu için bir **öngörü biti** ( $p_i$ ) tutulur.

Öngörü biti, ilgili komutun son çalışmasında dallanma olup olmadığını gösterir.

Eğer komutun son çalışmasında dallanma olduysa bir sonraki çalışmasında da dallanma olacağı varsayılır.

### Algoritma:

- Koşullu dallanma komutunu al
- Eğer ( $p_i = 0$ ) ise öngörü: "dallanma YOK", bellekte sıradaki komutu al
- Eğer ( $p_i = 1$ ) ise öngörü: "dallanma VAR", dallanmanın hedefi olan komutu al
- Eğer dallanma gerçekten olursa  $p_i \leftarrow 1$
- Eğer dallanma gerçekten olmazsa  $p_i \leftarrow 0$

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.30

Bilgisayar Mimarisi

Lisans: <http://creativecommons.org/licenses/by-nc-nd/3.0/>

**Örnek: 1 bitlik öngörü yöntemi ve döngüler**  
 Öngörü yöntemleri özellikle döngülerde yararlı olur.

Örnek:

```

  counter ← 100      ; saklayıcı veya bellek gözü
  LOOP
  ----              ; döngüdeki komutlar
  ----
  Decrement counter
  BNZ LOOP           ; Branch if Not Zero (koşullu dallanma, p biti vardır)
  ----              ; döngüden sonraki komut
  
```

Program çalışmaya başladığında BNZ komutunun p biti 1'dir (dallanma VAR öngörüsü). Döngünün ilk çalışmasında BNZ'de doğru öngörü yapılacak ve döngünün başındaki komut iş hattına alınacak.

p bitinin değeri döngünün son çalışmasına kadar değişmeyecek.

Döngünün son çalışmasında p biti hâlâ 1'dir ve "dallanma VAR" öngörüsü yapılır; ama counter sıfır olduğu için program döngünün başına dallanmaz ve döngüden sonraki komut ile devam eder (yanlış öngörü).

Sonuç olarak 100 defa dönen bir döngüde 99 defa doğru, sadece bir defa yanlış öngörü yapılmış oldu.

Döngüden sonra BNZ'nin p biti 0'dır, çünkü son çalışmada dallanma olmamıştır.

Aynı döngü başka bir döngünün içinde olduğu için tekrar çalıştığında ne olur?

www.akademi.itu.edu.tr/buzluca  
 www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.31

Bilgisayar Mimarisi

1 bit dinamik dallanma öngörü yöntemindeki sorun

Birden fazla defa çalışan (iç içe) döngülerde her defasında iki defa yanlış öngörü yapılır; biri döngü ilk çalıştığında diğeri de döngüden çıkarken.

```

  LOOP_EX
  LOOP
  BNZ LOOP
  BNZ LOOP_EX
  
```

Hatırlayın: önceki örnekte döngüden sonra BNZ'nin p biti 0'dır. Aynı dögüye tekrar gelinirse, ilk çalışmada BNZ'deki öngörü "dallanma YOK" olacaktır.

Ancak program dallanarak döngünün başına dönecektir (birinci hata).

Şimdi p biti 1 olur, çünkü dallanma olmuştur.

Döngünün son çalışmasına kadar öngörüler doğru olacaktır.

Döngünün son çalışmasında önceki örnekte de gösterildiği gibi yine hatalı öngörü yapılır (ikinci hata).

www.akademi.itu.edu.tr/buzluca  
 www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.32

Bilgisayar Mimarisi

**2 bit dinamik dallanma öngörü yöntemi:**  
 Her koşullu dallanma komutuna iki öngörü (durum) biti atanır.  
 Eğer komut 11 veya 10 durumlarındaysa "dallanma VAR" öngörüsü yapılır.  
 Eğer komut 00 veya 01 durumlarındaysa "dallanma YOK" öngörüsü yapılır.

Bu yöntemde ancak **peş peşe iki defa** yanlış öngörü yapılırsa öngörü kararı değişir.

www.akademi.itu.edu.tr/buzluca  
 www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.33

Bilgisayar Mimarisi

**Örnek: 2 bit dinamik dallanma öngörüsü**

V: Dallanma VAR  
 Y: Dallanma YOK

"VAR"dan "YOK"a

"YOK"tan "VAR"a

Durum:	V11	V10	V11	V10	Y00	Y00	Y01	Y00	Y01	V11
Öngörü:	V	V	V	V	Y	Y	Y	Y	Y	V
Olan:	V	V	V	V	Y	Y	Y	Y	Y	V

Gerçekte dallanma oldu  
 Dallanma VAR

Gerçekte dallanma YOK

Öngörü doğru çıktı ✓

Peş peşe 2 yanlış öngörü  
 Durum (karar) değişti

2 yanlış öngörü  
 Durum (karar) değişti

www.akademi.itu.edu.tr/buzluca  
 www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.34

Bilgisayar Mimarisi

**Doyan sayaç (Saturating counter):**  
 Diğer bir 2 bit dinamik dallanma öngörü yöntemi:

Dallanma öngörüsü yöntemlerinin sonlu durumlu makineleri (finite state machine) farklı şekillerde tasarlanabilir.

Doyan sayaç alternatif bir öngörü yöntemidir. Durum geçişleri farklıdır.

Eğer komut 11 veya 10 durumlarındaysa "dallanma VAR" öngörüsü yapılır.

Eğer komut 00 veya 01 durumlarındaysa "dallanma YOK" öngörüsü yapılır.

www.akademi.itu.edu.tr/buzluca  
 www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.35

Bilgisayar Mimarisi

**2.7.3 Derleyici Tabanlı Çözümler:**

**Gecikmeli Dallanma (Delayed branch):**

- Optimize gecikmeli dallanma (Optimized delayed branch):

Derleyici eğer mümkünse, programın davranışını değiştirmeden bazı komutların yerini değiştirerek iş hattını durdurulmasına veya boşaltılmasına gerek kalamadan dallanma sorunlarını çözer.

Bu çözüm mümkün olursa performans kaybı yaşanmaz.

- NOOP (No Operation) komutlarının eklenmesi:

Eğer komutların yerini değiştirmek mümkün değilse, derleyici dallanma komutlarının peşine gerekli sayıda NOOP komutu yerleştirir.

Derleyici tabanlı çözümler "2.8 RISC İş hattı" bölümünde ele alınmıştır.

www.akademi.itu.edu.tr/buzluca  
 www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.36

Bilgisayar Mimarisi

Lisans: <http://creativecommons.org/licenses/by-nc-nd/3.0/>

### 2.8 RISC İş Hattı (RISC Pipeline)

RISC işlemcilerde veri işleme komutları sadece saklayıcılar üzerinde çalışır. Bu komutlar için sadece iki segmanlı bir iş hattı yeterli olurdu:

I: (Instruction fetch) Komut alma, A: saklayıcılarla ALU işlemi (execution) Belleğe sadece okuma/yazma için erişilir. Bu komutlar saklayıcı-bellek arası aktarım yaparlar.

Bu komutların bellek erişimi için ek bir segmana (D) gerek duyulur. Böylece bir RISC işlemci için üç segmanlı bir komut iş hattı tasarlanabilir.

→ I → A → D →

- I: (Instruction Fetch) Komut Alma
- A: (Decode, ALU Operation) Komut çözme, ALU işlemi
- D: (Data, memory access) Veri, bellek erişimi

Performansı arttırmak için daha fazla segmana (4, 5 veya daha fazla) RISC işlemciler de vardır.

Örnekler:

MIPS R3000: 5 katman  
MIPS R4000: 8 katman (superpipelined)  
ARM7: 3 katman, ARM Cortex-A8: 13 katman.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.37

Bilgisayar Mimarisi

### 2.8.1 Örnek: Üç segmanlı bir RISC komut iş hattı

- I (Instruction Fetch): Bellekten PC'nin gösterdiği komut alınır.
- A (Decode, ALU Operation): Komut çözme ve ALU işlemi ALU üç farklı iş için kullanılır.
  - Saklayıcılar üzerindeki aritmetik/lojik işlemlerde
  - Bellek erişimi komutlarında adres hesabı için.  $LDL(R5)\#10, R15$
  - Bağlı adreslemede  $PC \leftarrow PC + Y$  işlemi için. A (ALU) segmanında hem işlem yapılır hem de sonuç varış saklayıcısına (R veya PC) aynı saat çevriminde yazılmış olur.
- D (Data): Bu segman sadece bellek erişimi (load/store) için kullanılır. Bellekten gelen veri saklayıcıya ya da saklayıcıdaki veri belleğe yazılır.

I ve D segmanlarında aynı anda bellek erişimi yapılmaya çalışılır.

Kaynak çatışması sorununu çözmek için komut ve veriler için paralel erişilebilen ayrı belleklerin (ve ayrı yolların) olması gerekir (Harvard mimarisi).

Diğer çözümler komut ve veri kuyrukları veya cep bellek (cache memory) kullanılmaktadır.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.38

Bilgisayar Mimarisi

### 3 katmanlı RISC iş hattında veri bağımlılığı (Data conflict (dependency))

Örnek Program:

```

100 LOAD M[X], R1    R1 ← M[X] ; Bellekten oku
104 LOAD M[Y], R2    R2 ← M[Y]
108 ADD R1, R2, R3    R3 ← R1 + R2
10C STORE R3, M[Z]   M[Z] ← R3 ; Belleğe yaz
110 LOAD M[W], R4    R4 ← M[W]

```

ADD komutu bu segmanı kullanmaz

#### İş hattında veri bağımlılığı

Saat çevrimi

Komutlar	1	2	3	4	5	6	7
LOAD R1	I	A	D				
LOAD R2		I	A	D			
ADD R1, R2, R3			I	A	D		
STORE R3				I	A	D	
LOAD R4					I	A	D

Veri çatışması:  
4. saat çevriminde LOAD komutu R2'ye yazıyor; aynı anda ADD komutu R2'yi kaynak operandı olarak kullanıyor.

Burada R3 ile ilgili bir kaynak çatışması yoktur.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.39

Bilgisayar Mimarisi

### Çözümler: Gecikmeli Yükleme (Delayed Load)

- Çözüm 1: NOOP (No Operation) komutları eklemek

Derleyici LOAD ile sorun çıkaran saklayıcıyı kullanan komut arasına NOOP komutu ekler.

```

100 LOAD M[X], R1
104 LOAD M[Y], R2
108 NOOP
10C ADD R1, R2, R3
110 STORE R3, M[Z]
114 LOAD M[W], R4

```

Saat çevrimi

Komutlar	1	2	3	4	5	6	7	8
LOAD R1	I	A	D					
LOAD R2		I	A	D				
NOOP			I	A	D			
ADD R1, R2, R3				I	A	D		
STORE R3					I	A	D	
LOAD R4						I	A	D

Derleyici tarafından yerleştirildi.

R2 güncelleniyor.

Farklı saat çevrimleri

R2 operand olarak kullanılıyor.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.40

Bilgisayar Mimarisi

### Çözümlerin devamı:

- Çözüm 2: Komutların yerini değiştirmek, optimize edilmiş gecikmeli yükleme

Derleyici programda uygun bir komut yerini değiştirerek bu komutu LOAD ile sorun çıkaran saklayıcıyı kullanan komut arasına yerleştirir.

```

100 LOAD M[X], R1
104 LOAD M[Y], R2
108 LOAD M[W], R4 ←
10C ADD R1, R2, R3
110 STORE R3, M[Z]

```

Derleyici tarafından yer değiştirildi.

R2 güncelleniyor.

Farklı saat çevrimleri

R2 operand olarak kullanılıyor.

Performans artmıştır: Toplam 7 saat çevrimi (çözüm 1'deki 8 çevrim yerine). Programın davranışı değişmemiştir.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.41

Bilgisayar Mimarisi

### Dallanma Problemi

Dallanma komutları (branch, jump) PC'yi ALU (yürütme) katmanında güncellerler. Ancak dallanma komutu yürütülürken bellekte dallanmadan sonra yer alan komut da (dallanmanın hedefi değil) iş hattına alınmış olur. Bu durumda ya bir donanım birimi iş hattını boşaltmalı ya da derleyici tabanlı bir çözüm uygulanmalıdır.

#### Önlem alınmazsa:

#### Örnek: Sanal Kod

```

100 LOAD M[X], R1
104 ADD 1, R2
108 JUMP 200
10C ADD R1, R2
...
200 STORE R1, M[Y]

```

Bu komut yürütülmek istenmiyor.

Ya bir donanım birimi iş hattını boşaltmalı ya da derleyici tabanlı bir çözüm uygulanmalıdır.

PC güncelleniyor.  $PC \leftarrow 200$  (Hedef adres)

Problem: Bu komut gereksiz yere alınmış oldu. Aslında çalışması gerekiyor!

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.42

Bilgisayar Mimarisi Lisans: <http://creativecommons.org/licenses/by-nc-nd/3.0/>

**Çözümler: Gecikmeli Dallanma (Delayed Branch)**

- **Çözüm 1:** NOOP (No Operation) komutları eklemek

Derleyici dallanma komutunun peşine NOOP komutu ekler.

**Sanal Kod**

```

100 LOAD M[X], R1
104 ADD 1, R2
108 JUMP 200
10C NOOP
110 ADD R1,R2
....
200 STORE R1,M[Y]

```

**NOOP ile gecikmeli dallanma**

Saat çevrimi	1	2	3	4	5	6	7
Komutlar							
100 LOAD M[X], R1	I	A	D				
104 ADD 1, R2		I	A				
108 JUMP 200			I	A			
10C NOOP				I	A		
110 ADD R1,R2					I	A	D

Derleyici tarafından yerleştirildi.

PC güncelleniyor.  
PC ← 200 (Hedef adres)

Farklı saat çevrimleri

Şimdi program istenen şekilde çalışır.  
Ancak NOOP komutu eklemek performans düşürecektir.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.43

Bilgisayar Mimarisi

**Çözümler: Gecikmeli Dallanma (Delayed Branch) (devamı)**

- **Çözüm 2:** Komutların yerini değiştirmek, optimize gecikmeli dallanma

Derleyici uygun bir komutu (çoğunlukla dallanmadan önce gelen) dallanmanın peşine kaydırır.

Komut yer değiştirme iş hatında performans düşüşüne neden olmaz.

**Komut yer değiştirme ile gecikmeli dallanma:**

**Sanal Kod**

```

100 LOAD M[X], R1
104 JUMP 200
108 ADD 1, R2
10C ADD R1,R2
....
200 STORE R1,M[Y]

```

**Komut yer değiştirme ile gecikmeli dallanma:**

Saat çevrimi	1	2	3	4	5	6
Komutlar						
100 LOAD M[X], R1	I	A	D			
108 JUMP 200		I	A			
104 ADD 1, R2			I	A		
10C ADD R1,R2				I	A	D

Derleyici tarafından yerleştirildi.

PC güncelleniyor.  
PC ← 200 (Hedef adres)

Bu komut zaten alınmış oldu.

Performans artmıştır: Toplam 6 saat çevrimi (çözüm 1'deki 7 çevrim yerine).  
Programın davranışı değişmemiştir.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.44

Bilgisayar Mimarisi

**Komutların sırasını değiştirmek ile ilgili önemli noktalar:**

Dallanmadan önce gelen bir komut dallanmadan sonraya kaydırılabilir. Dallanmanın koşulu veya hedef adresi kaydırılan komuta bağlı olmamalı. Bu yöntem (eğer mümkünse) her zaman performansı artırır (NOOP'a göre). Özellikle koşullu dallanmalarda bu yöntem dikkatli uygulanmalı. Dallanmanın bağlı olduğu koşulu belirleyen komut dallanmadan sonraya taşınamaz. Bu durumda NOOP eklenir.

Diğer seçenekler:

Derleyici taşımak üzere şu komutları seçebilir:

- **Dallanmanın hedefinden** (gidilecek yerden)
  - Taşınan komut dallanma gerçekleşme de çalışacaktır. Bu programı etkilememeli.
  - Dallanma gerçekleşirse performans artar.
- **Dallanma komutunun peşinden** (dallanma olmazsa devam edilen kol)
  - Taşınan komut dallanma gerçekleşse de çalışacaktır. Bu programı etkilememeli.
  - Dallanma gerçekleşmezse performans artar.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.45

Bilgisayar Mimarisi

**2.8.2 Dört Segmanlı bir RISC İş Hattı**

RISC işlemcilerdeki komutların basitliği ve düzenli yapıları nedeniyle bu işlemcilerin iş hatları üç veya dört segmanlı olarak gerçekleştirilebilmektedir.

Dört segmanlı bir RISC iş hattı aşağıdaki gibi tasarlanabilir:

- **I (Instruction Fetch):** Bellekten PC'nin gösterdiği komut alınır.
- **R (Decode, Read register file):** Komutu çöz ve kaynak saklayıcıları oku
- **A (ALU Operation And register write):** ALU işlemi ve sonuçları yazma
- **D (Data):** LOAD/STORE komutları bellek erişimi için kullanır.

Bir iş hattındaki segman sayısını arttırmak eğer segmanlar küçülüyor ve hızlanıyorsa saat işaretinin de hızlanmasını sağlar (Bkz. 2.10).

Ancak segman sayısındaki artış çatışma durumunda cezaların da artmasına neden olur.

Yukarıda verilen 4 segmanlı örnek iş hatında gecikmeli yükleme çözümünü uygulamak için 2 adet NOOP komutu eklemek gerekir.

Benzer şekilde, gecikmeli dallanma çözümünü uygulamak için 2 adet NOOP komutunu dallanma sonrasına kaydırmak gerekir.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.46

Bilgisayar Mimarisi

**Gecikmeli Yükleme (4 segmanlı iş hattı)**

2 adet NOOP komutu eklenecek.

**Sanal Kod**

```

100 LOAD M[X], R1
104 LOAD M[Y], R2
108 NOOP
10C NOOP
110 ADD R1,R2,R3
....
200 STORE R1,M[Y]

```

**NOOPlar ile gecikmeli yükleme:**

Saat çevrimi	1	2	3	4	5	6	7
Komutlar							
LOAD R1	I	R	A	D			
LOAD R2		I	R	A	D		
NOOP			I	R	A		
NOOP				I	R	A	
ADD R1,R2,R3					I	R	A

Derleyici tarafından yerleştirildi.

R2 güncelleniyor.

Farklı saat çevrimleri

R2 operand olarak kullanılıyor.

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.47

Bilgisayar Mimarisi

**Gecikmeli Dallanma (4 segmanlı iş hattı)**

2 adet NOOP komutu eklenecek.

**Sanal Kod**

```

108 JUMP 200
10C NOOP
110 NOOP
114 ADD 1,R1
118 ADD R1,R2
....
200 STORE R1,M[Y]

```

**NOOPlar ile gecikmeli dallanma:**

Saat çevrimi	1	2	3	4	5	6	7
Komutlar							
108 JUMP 200	I	R	A				
10C NOOP		I	R	A			
110 NOOP			I	R	A		
200 STORE R1,M[Y]				I	R	A	D

Derleyici tarafından yerleştirildi.

PC güncelleniyor.  
PC ← 200 (Hedef adres)

Farklı saat çevrimleri

www.akademi.itu.edu.tr/buzluca  
www.buzluca.info

2005-2014 Dr. Feza BUZLUCA 2.48