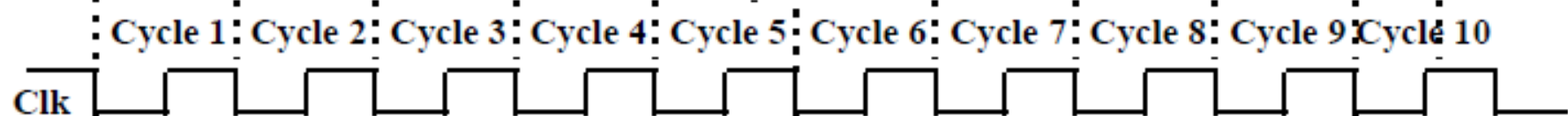


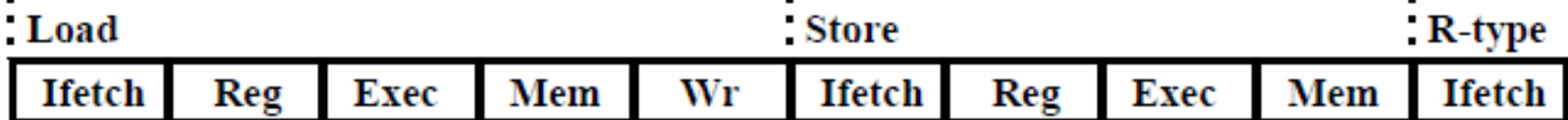
PIPELINE (BORU HATTI) İşlemi ile Performansın Geliştirilmesi



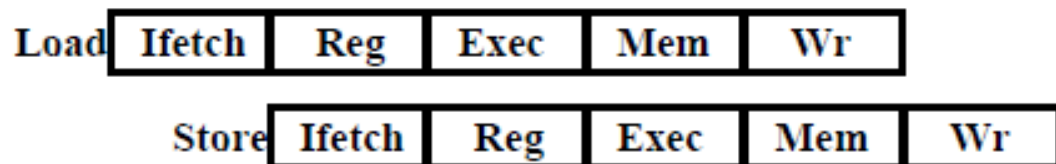
Single Cycle Implementation:



Multiple Cycle Implementation:

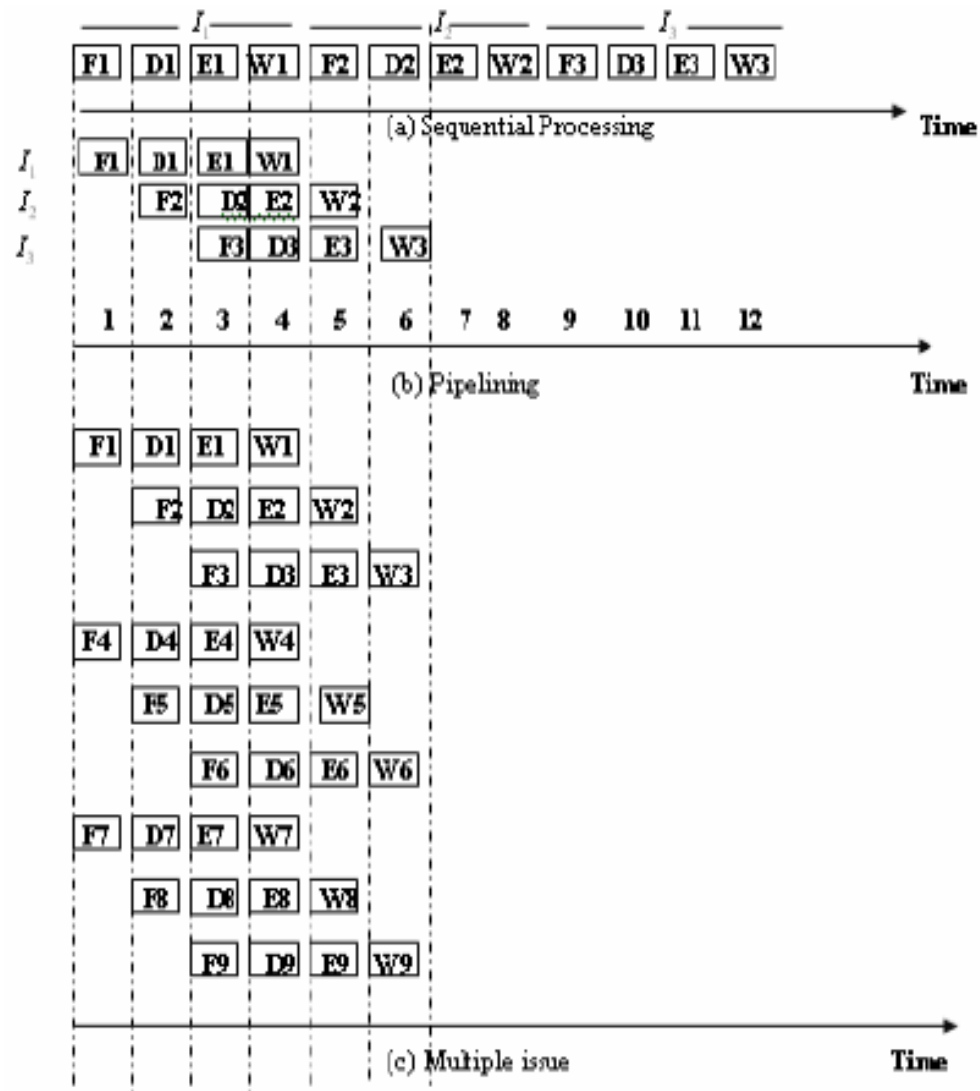


Pipeline Implementation:



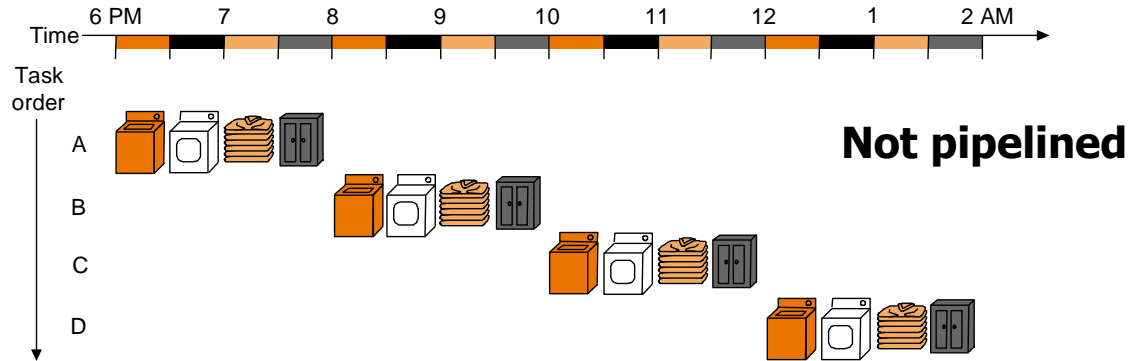
Paralel İşleme ve PIPELINE

- Aynı zamanda, birden fazla veya çok sayıda işlemin yapılabilmesi bilgisayar sisteminin hızını (Throuhput) arttırır.
- Paralel işlemede ise aynı anda birden fazla komut icra edilir. Örneğin, bir zaman diliminde bir komut ALU'da yürütülürken (execute), aynı zaman diliminde daha sonraki komut ise bellekten okunabilir. Bu işlem tek işlemcili paralel çalışmaya örnektir.
- Paralel çalışmaya diğer örnek , birden fazla işlemcisi bulunan sistemlerdir.
- Paralel işleme aşağıdaki başlıklarda incelenebilir.
- 1- PIPELINE (Boru Hattı) İşlemleri
- 2-VEKTÖR İŞLEMLERİ
- 3-DİZİ İŞLEMCİLERİ
- x Pipeline bir yürütme tekniği olup, artmetik alt işlemlerde veya bir komutun fazlarının (evrelerinin) işlenmesi sırasında üstüste gelmesidir.
- x Vektör işlemleri büyük boyutlu vektörel veya matris işlemlerinin yapılması içindir.
- x Dizi İşlemcileri ise büyük çaplı diziler üzerinde işlem yaparlar.

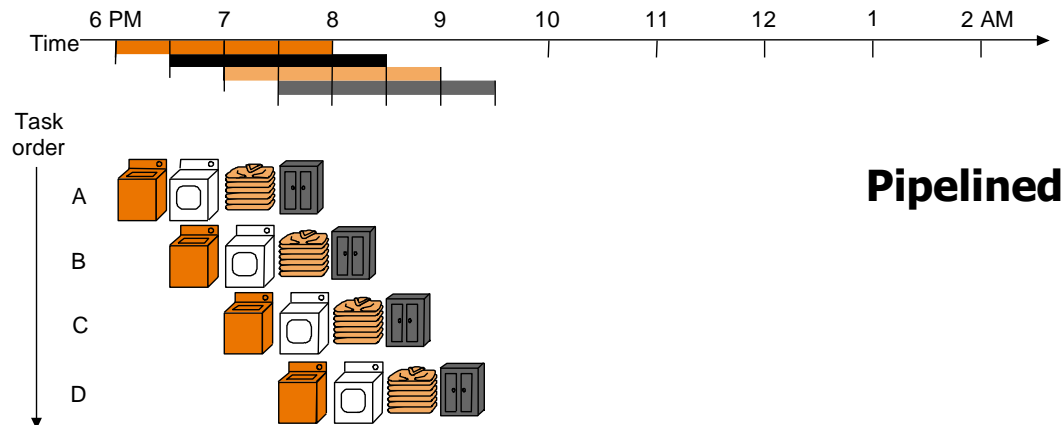


Pipelining

- Bir işe başladıktan sonra en kısa sürede bitir.!! Zaman kaybetme!

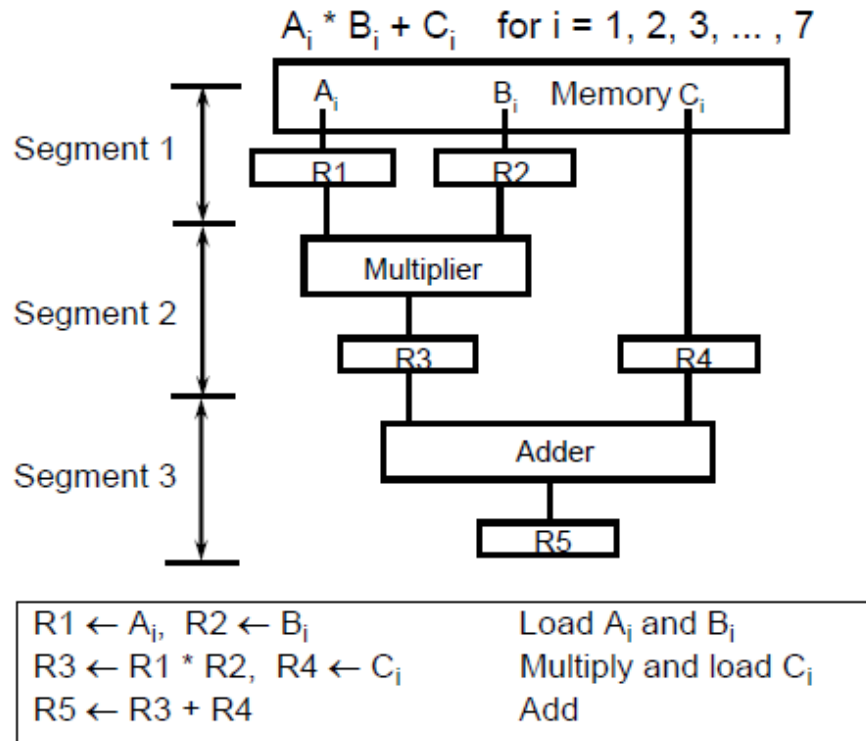


Farz etki her bir görev (yıkama-kurutma- katlama- dolaba yerleştirme) 30 dakikadır. Ve ayrı görevler için ayrı donanımlar kullanılıyorsa, farklı görevler aynı zamanda üstüste çakışabilir.



PIPELINE İŞLEMLERİNİN ANLAŞILMASI

- Pipeline işlemi birtakım işlemlerin topluluğudur ve boru hattını oluşturan her bir kesimde (segmentte) her bir farklı işin farklı alt işlemleri aynı zamanda gerçekleşir. Bir kesimde elde edilen sonuç aynı iş için sonraki kesimde kullanılmak üzere saklanır.



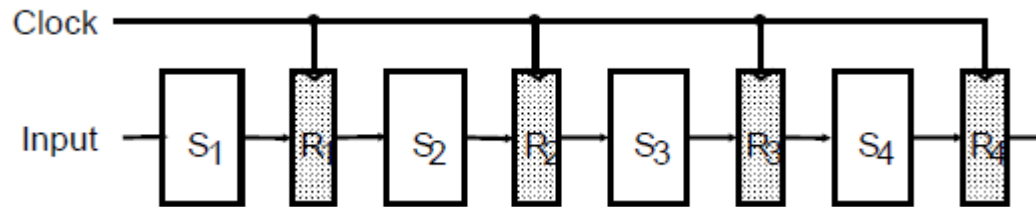
Her bir kesimde icra edilecek alt işlemler şekilde verilmiştir. 5 Registerden herbiri, her clock cyce'ında yeni veriler ile yüklenir.

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A1	B1			
2	A2	B2	$A1 * B1$	C1	
3	A3	B3	$A2 * B2$	C2	$A1 * B1 + C1$
4	A4	B4	$A3 * B3$	C3	$A2 * B2 + C2$
5	A5	B5	$A4 * B4$	C4	$A3 * B3 + C3$
6	A6	B6	$A5 * B5$	C5	$A4 * B4 + C4$
7	A7	B7	$A6 * B6$	C6	$A5 * B5 + C5$
8			$A7 * B7$	C7	$A6 * B6 + C6$
9					$A7 * B7 + C7$

Reg'lerin içeriklerinin değişimi

Aynı biçimde alt işlemlere parçalanabilen ve aynı karışıklığa sahip her işlem PIPELINE olarak işlenebilir.

Herbir alt işlem segment(kesimde) yapılır. Bir önceki işlemde elde edilen sonucun bir sonraki işlemde kullanılabilmesi için Reg'lere ihtiyaç vardır. Aşağıdaki durum 4 kesimli bir PIPELINE işlemidir.



PIPELINE davranışı uzay-zaman diyagramı ile gösterilebilir. 4 kesimli bir PIPELINE işlemi ile 6 tane görevin icra edilmesi aşağıdaki şemada verilmiştir.

		1	2	3	4	5	6	7	8	9	
Segment	1	T1	T2	T3	T4	T5	T6				→ Clock cycles
	2		T1	T2	T3	T4	T5	T6			
	3			T1	T2	T3	T4	T5	T6		
	4				T1	T2	T3	T4	T5	T6	

PIPELINE işlemenin hızı (Kazancımız ne?)

n: Görev Sayısı (Bir assembler programdaki işlenecek komut sayısı diyebiliriz.)

PIPELINE çalışmayan bir makine için;

tn: her bir görevi bitirmek için gerekli zaman- (Örn. Komut cyle'i)

t1: n tane görevi tamamlamak için gerekli harcanacak zaman.

$$t1 = n * tn$$

PIPELINE bir makine için (k kesimli (segmenli - durumlu)

tp: Herbir kesimin (alt işlemin-durum-kesim-segment) tamamlanması için gerekli zaman.

tk: n adet görevin (k kesimli) başarılmaları için gerekli zaman

$$tk = (k + n - 1) * tp$$

- **Hızlanma (Speedup)**

- **Sk:** Speedup

$$\rightarrow Sk = t1 / tk = n * tn / (k + n - 1) * tp$$

n görev sayısı arttıkça (k + n -1) terimi yaklaşık n olur. Buna göre aşağıdaki bağıntı ortaya çıkar.

$$Sk = tn / tp$$

Eğer; pipeline ve pipeline olmayan bilgisayarlarda bir görevin tamamlanması için geçen zaman eşit ise, Bu durumda; pipeline toplam hızlandırması k (kesim sayısı) kadar olabilir.

$$\lim_{n \rightarrow \infty} S_k = \frac{t_n}{t_p} \quad (= k, \text{ if } t_n = k * t_p)$$

Örnek:

k=4 kesimli pipeline'da, bir alt işlemin her bir kesimde işlenebilmesi için gerekli zaman;

$$t_p = 20\text{ns olsun.}$$

İcra edilecek görev sayısı $n=100$ olsun. S_k hızlanma değerini hesaplayınız.

Çözüm:

- Pipeline olmayan bir sistemde, tek bir görev için gereken zaman $t_n = 20 * 4 = 80\text{ns}$
Tüm görevin icra edilmesi için harcanacak zaman $t_1 = 100 * 20 * 4 = 8000\text{ ns.}$

$$S_k = t_1 / t_k = n * t_n / (k + n - 1) * t_p \text{ idi}$$

- Pipeline makine için toplam süre ; $(k + n - 1) * t_p = (4 + 100 - 1) * 20 = 2060\text{ ns}$

- Buna göre;

$$S_k = 8000 / 2060 = 3.88$$

Elde edilir.

Önemli not: **Bütün alt işlemlerin icra sürelerinin aynı olduğunu kabul ediyoruz.**

- PIPELINE İşlemenin 2 temel uygulaması vardır.
- 1- Aritmetik işlemlerde PIPELINE
- 2-Komut Yürütmede PIPELINE

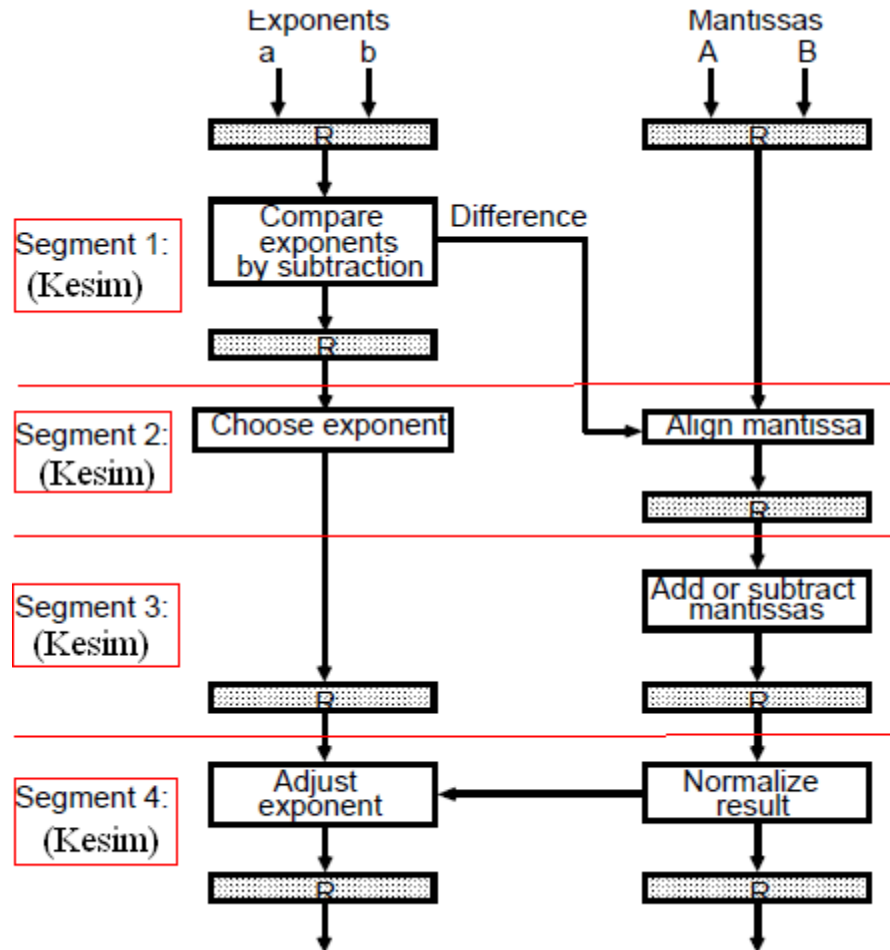
Aritmetik İşlemlerde PIPELINE

PIPELINE aritmetik birimleri, FP işlemler için ve çarpma işlemleri için önemli bir hızlandırıcı yapıdır.

Floating-point Toplama

$$X = A \times 2^a$$
$$Y = B \times 2^b$$

- [1] Exponentlerin karşılaştırılması
- [2] Mantisanın hizalanması
- [3] Mantisaların toplanması/Çıkarılması
- [4] Sonuç Normalizasyonu



Örnek:

16 tabanlı uzaydaki x ve y FP sayılarını toplayınız.

$$x = 0.188 \times 16^2$$

$$y = 0.C80 \times 16^1$$

Kesim 1 : üsler eşit mi? $2 - 1 = 1$ çıktığından eşit değil.

Kesim 2 : Kesirlerin hizalanması. y sayısını 1 defa sağa kaydırarak üsler eşitlenir.

$$x = 0.1880 \times 16^2$$

$$y = 0.0C80 \times 16^2$$

Kesim 3: Kesirlerin toplanması/çıkarılması

$$Z = 0.25 \times 16^2$$

Kesim 4: Sonucun Normalizasyonu

Normalizeye gerek yok.

$$Z = 0.25 \times 16^2$$

4 kesim için harcanan zamanlar: $t_1 = 60\text{ns}$, $t_2 = 70\text{ns}$, $t_3 = 100\text{ns}$, $t_4 = 80\text{ns}$, fiziksel gecikmeler 10ns olsun. Buna göre;

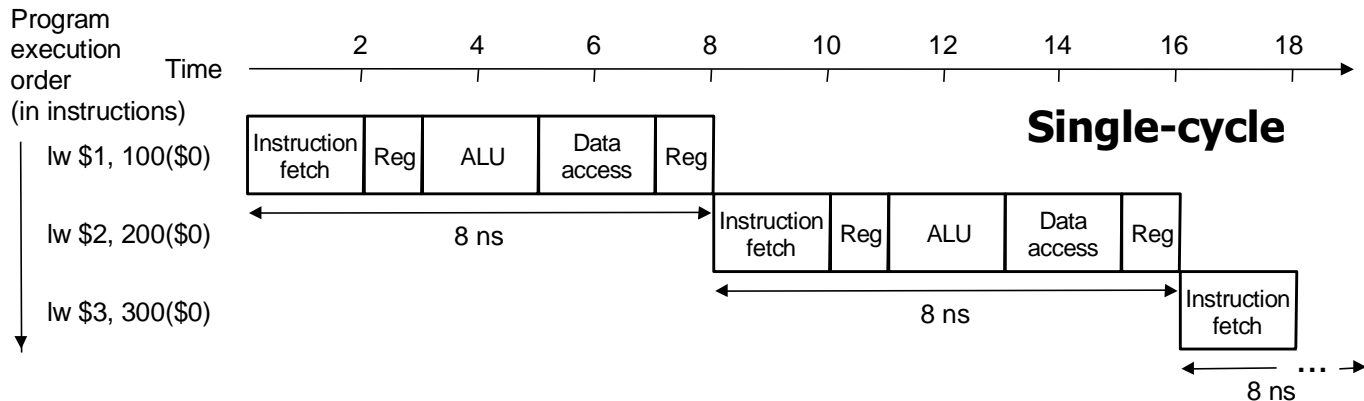
pipeline'sız bir makinada 1 toplama işlemi için: $60 + 70 + 100 + 80 + 10 = 320\text{ns}$

Aritmetik işlem için 4 kesimli pipeline 'da ise her bir kesim de harcanan süre farklı olduğundan en kötü durum olarak pipeline clock cycle'ını 100ns alarak $t_p = 100 + 10 = 110\text{ns}$ elde ederiz. Buna göre;

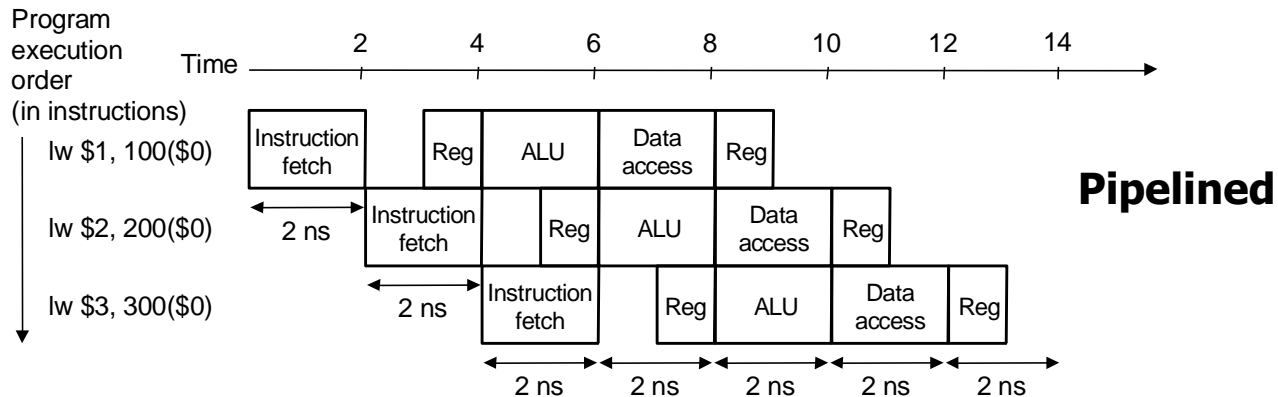
$$S_k = t_n / t_p = 320 / 110 = 2.9$$

Komut Yürütme için Pipeline

Single-Cycle Komut Yürütme: Planlama



Farzedin ki fetch, memorye erişim ve ALU işlemleri 2 ns ; register erişimi 1 ns. Buna göre, single cycle clock 8 ns'dir. Pipelined clock cycle ise 2 ns.



Pipelining: Unutma

- Pipeline işlemi tek bir görev (veya komut işlemesi) için harcanacak zamanı (latency) azaltmaz fakat görevin bütünü (bir programın sıralı komutları v.b) için harcanan zamanı azaltır (Verilen sabit bir zamanda yapılan iş miktarını –throughput- arttırır).
- Pipeline hızı en uzun durum ile sınırlıdır.
 - *potential* hızlandırıcı= Boru durumları sayısıdır (kesim sayısı).
 - *Dengesiz boru durumları hızı azaltır (her kesim için farklı süreler).*
- Pipeline sürecinde herhangi bir fazda boş kalmak pipeline'ı yavaşlatır.
- En verimli pipeline işlemi, komutların parçalanmasıyla oluşan alt işlem sürelerinin (fetch, decode v.b) eşit sürelerde işlenmesi ile olur. Yani kesim veya faz sürelerinin eşit olması ile olur.

Örnek Problem

- *Problem: Çamaşır yıkama örneği için aşağıdaki tabloyu doldurmak isteyelim.*
 1. *Her bir durumun uzunluğu(stage lengths) 30, 30, 30 30 dak. olsun.*
 2. *Durumların uzunluğu (stage lengths) , 20, 20, 60, 20 dak. olsun.*

Person	Unpipelined finish time	Pipeline 1 finish time	Ratio unpipelined to pipeline 1	Pipeline 2 finish time	Ratio unpipelelined to pipeline 2
1	120 d.	120 d.	$120/120 = 1$	120 d.	$120/120 = 1$
2	$2 \times 120 = 240d$	$120 + 30 = 150d$	$240/150 = 1.6$	180 d	$240/180 = 1.33$
3					
4					

n (durum sayısı) $n \times 120 / (120 + ((n-1) \times 30))$

- *Pipeline'ı hızlandırmak için bir formül ortaya çıkaralım?*

Dört kesimli (Segmentli) Komut İçin Pipeline

Dört aşamaya (faz – Kesim-segment) bölünmüş komutlar için kesimler ;

FI: Komutu getir

DA: Komutu çöz, etkin adresi hesapla

FO: Veriyi getir

EX: Komutu yürüt

Not: komut ve data hafızalarına ayrı ayrı erişildiğini düşününüz.

Dallanma komutu olmadığı sürece, hiçbir kesim boş geçilmez. Farklı işlemler yapar. Ancak 3. komutun dallanma olduğu durumu inceleyelim. Neler olur ve neden olur? Niye 5.ve 6. adımlarda işlem yapılmaz?

Dallanma komutunun işleyip dallanılacak adres ortaya çıkıncaya kadar bir sonraki komut ile ilgili sadece Fetch işlemi yapılabilir. 5.6. adımlarda işlem yapılmaz.

Step:	1	2	3	4	5	6	7	8	9	10	11	12	13
Komut	1	FI	DA	FO	EX								
	2		FI	DA	FO	EX							
(Branch)	3			FI	DA	FO	EX						
Dallanma	4				FI	-	-	FI	DA	FO	EX		
	5					-	-	-	FI	DA	FO	EX	
	6								FI	DA	FO	EX	
	7									FI	DA	FO	EX

Örnek: 4 kesimli bir komutun kesimleri F(20ns), D(20ns), E(20ns), WB(25ns) dir.

a)Aşağıdaki pipeline diyagramı doğrumudur? Yanlışmıdır? Neden?

b)Diyagram yanlışsa doğrusunu çiziniz.

c)n komut için harcanan gerekli süreyi (th)hesap eden bir formül bulunuz

d)135 komutluk bir çalışma için seri çalışmaya göre pipeline kazancını hesap ediniz?

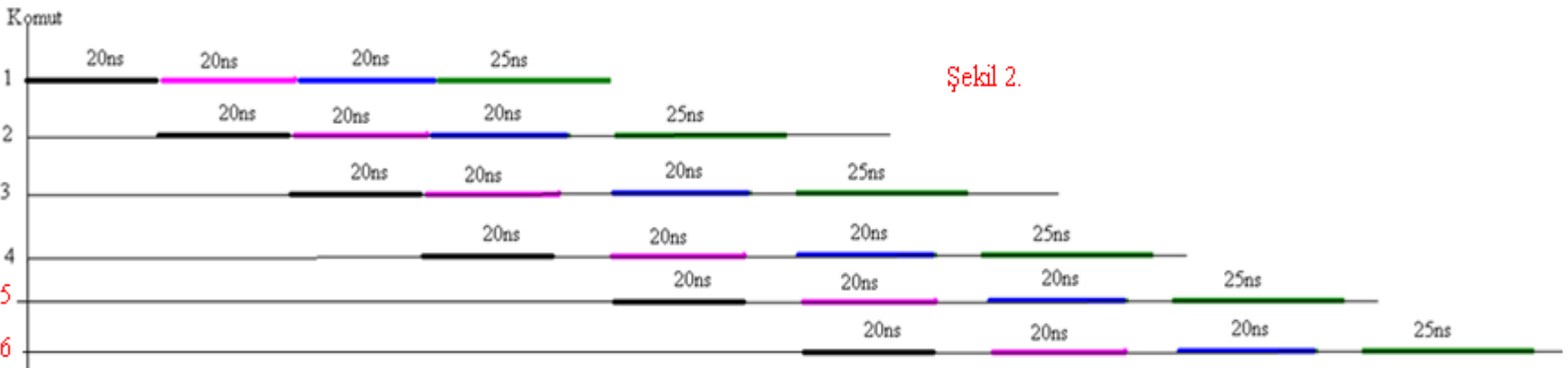
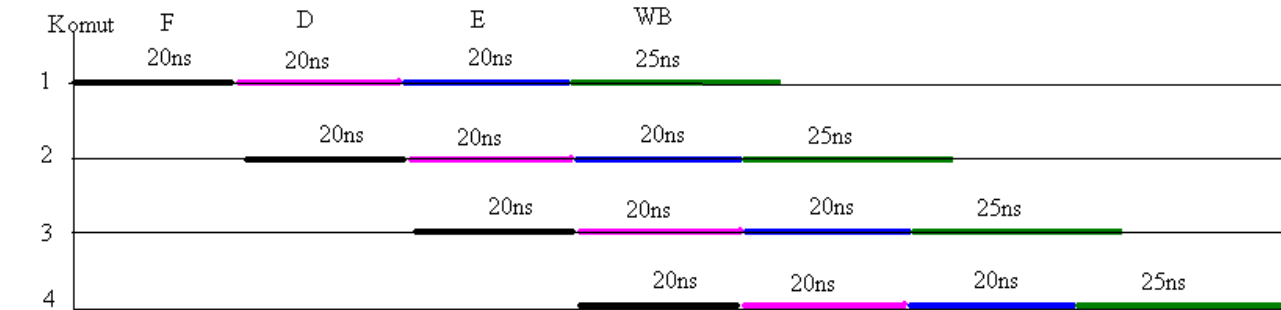
Çözüm: a) Yanlıştır. **b)** doğrusu Şekil 2'deki gibi olabilir.

c)Th = Komut süresi + (Komut sayısı -1)*25ns (şekil 2'ye göre) = 85ns + (Komut sayısı -1) * 25 ns

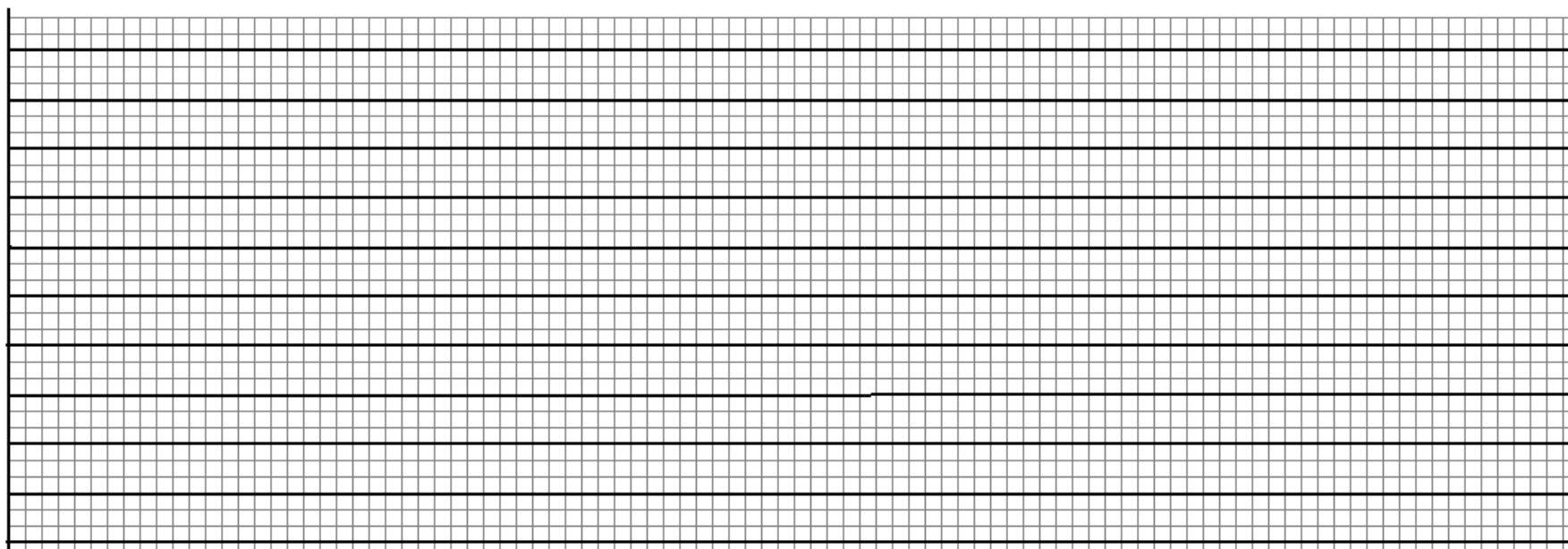
d)(Seri çalışma için) Th = 85ns * 135 = 11.475 ns

(Pipeline çalışma için)Th = 85ns + (135 -1)* 25ns = 85ns + 3350ns = 3435 ns

Verim = 11475 / 3435 = 3.34



Şekil 2.



Komut

[illegible]

MIPS'te PIPELINE

- MIPS için Pipeline işlem neden kolaydır?
 - **Bütün komutlar benzer uzunluktadır.**
 - Tüm komutlar için fetch ve decode durumları benzer şekilde işlenir.
 - **Yalnızca birkaç komut formatları vardır.**
 - Basit bir şekilde, Komutların decodlanması ve tek durumda yapılması mümkün hale gelir.
 - ***Memory işlemleri sadece load/store komutlarında oluşur.***

Böylece memory erişimi tam olarak bir sonraki duruma ertelenmiş olabilir
 - ***Operandlar (İşlenenler) memory'de dizili sıralıdır.***
 - Bir data transfer komutu yalnızca bir memory'e erişim durumuna ihtiyaç duyar.

Bir MIPS
komutunun
yürütülmesi beş
adımda
tamamalanabilir.

Step	Name	Description
Instruction Fetch	IF	Read an instruction from memory.
Instruction Decode	ID	Read source registers and generate control signals.
Execute	EX	Compute an R-type result or a branch outcome.
Memory	MEM	Read or write the data memory.
Writeback	WB	Store a result in the destination register.

Instruction	Steps required				
beq	IF	ID	EX		
R-type	IF	ID	EX		WB
sw	IF	ID	EX	MEM	
lw	IF	ID	EX	MEM	WB

Bütün komutların hepsi beş adımlık süre gerektirmeyebilir.

(Instruction Fetch)

IR := Memory[PC]; PC := PC + 4

(Instruction decode and Register fetch)

A := Reg[IR[25:21]], B := Reg[IR[20:16]]

ALUout := PC + sign-extend(IR[15:0])

(Execute | Memory address | Branch completion)

Memory refer: ALUout := A + IR[15:0]

R-type (ALU): ALUout := A op B

Branch: if A = B then PC := ALUout

(Memory access | R-type completion)

LW: MDR := Memory[ALUout]

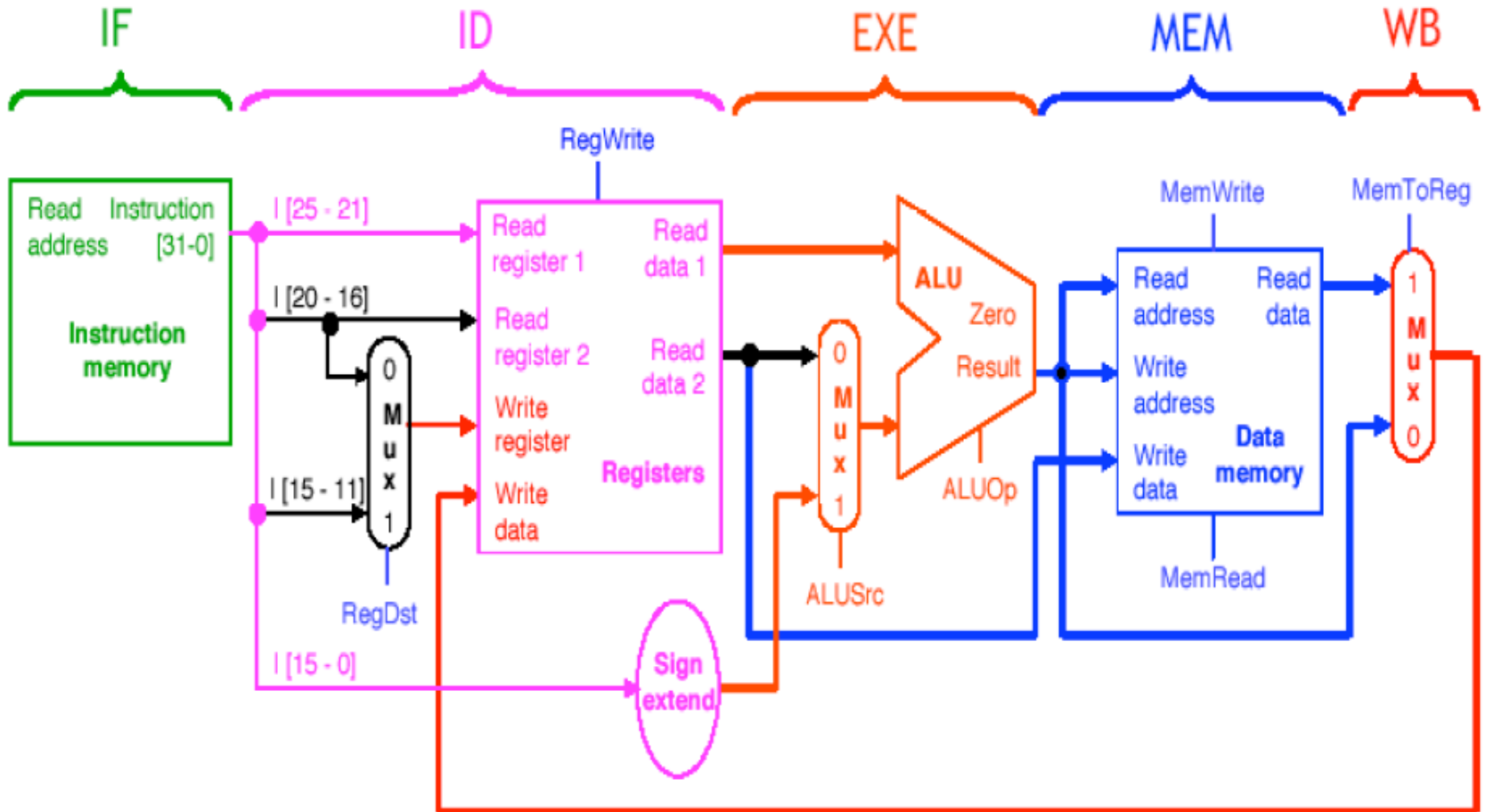
SW: Memory[ALUout] := B

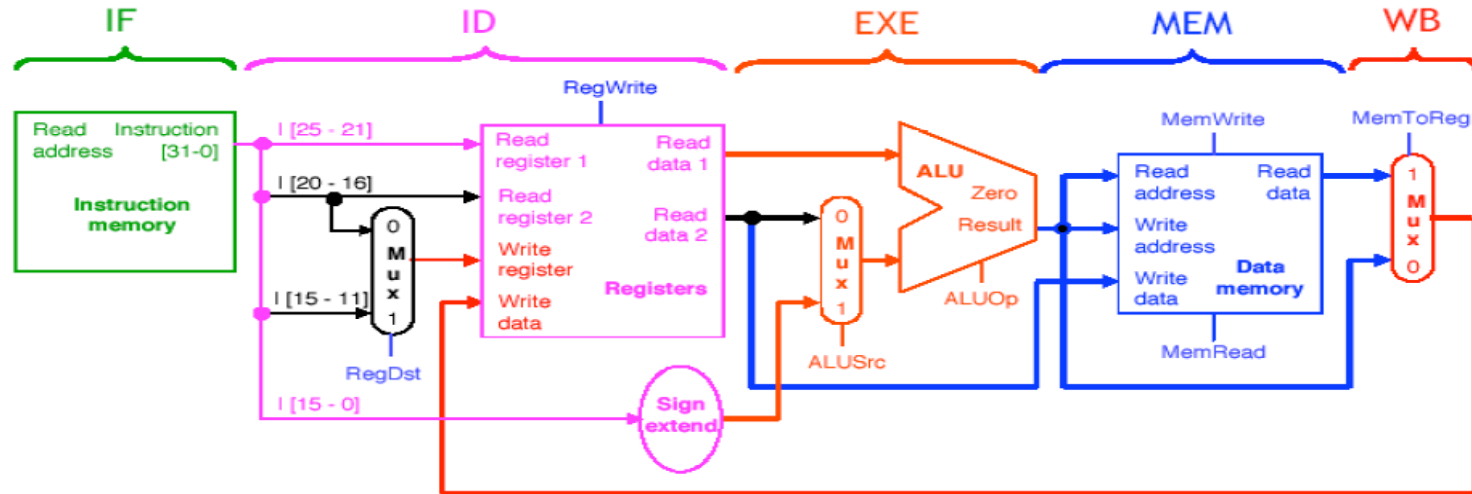
R-type: Reg[IR[15:11]] := ALUout

(Write back)

LW: Reg[[20:16]] := MDR

- Herbir kesim(segment) kendi fonksiyonel birimine sahiptir.
- Aşağıda satage bufferları gösterilmemiş bir pipeline datapath yapısı görülmektedir.





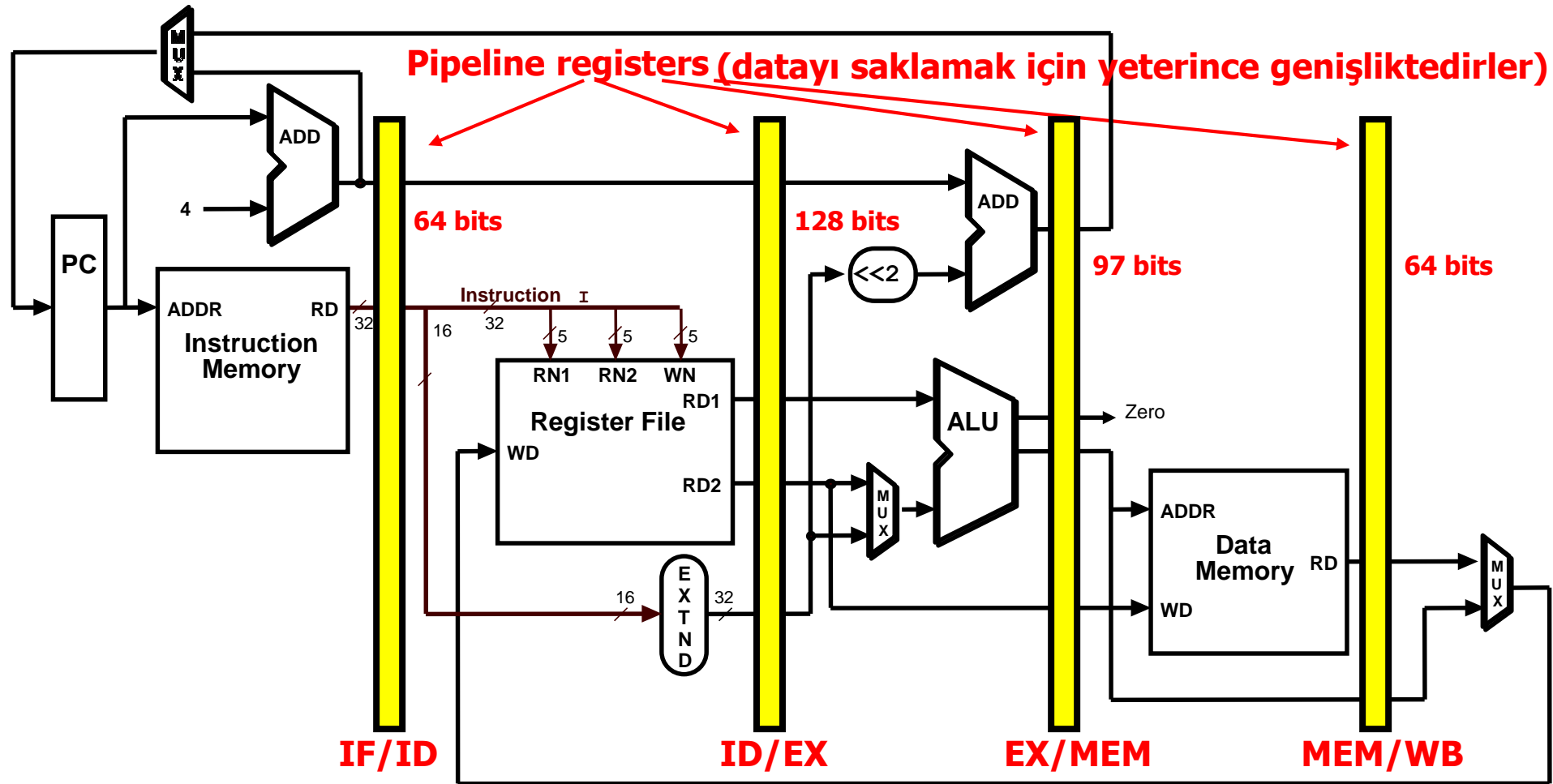
Clock cycle

	1	2	3	4	5	6	7	8	9
lw \$t0, 4(\$sp)	IF	ID	EX	MEM	WB				
lw \$t1, 8(\$sp)		IF	ID	EX	MEM	WB			
lw \$t2, 12(\$sp)			IF	ID	EX	MEM	WB		
lw \$t3, 16(\$sp)				IF	ID	EX	MEM	WB	
lw \$t4, 20(\$sp)					IF	ID	EX	MEM	WB

Dikkat: 3.clock cycyle'daki gibi Ex segmenti ve IF segmenti ALU'ya ihtiyaç duymaktadır.Fazladan bir toplayıcı devreyede ihtiyaç duyulacaktır.

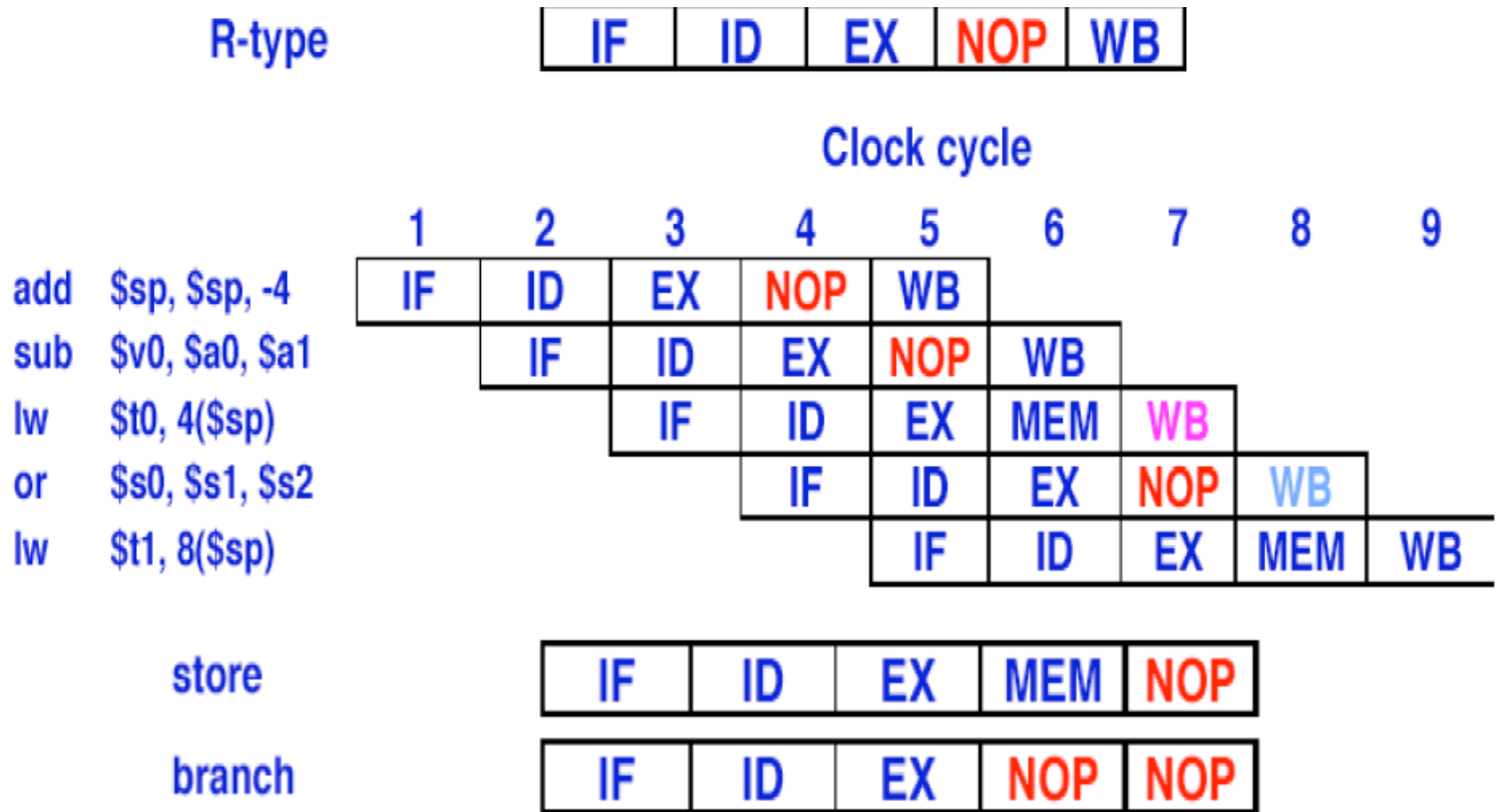
- Biz kontrol devresinin kolaylığı açısından her komutu 5 segmentli olarak (stage) göstereceğiz.
- Aynı zaman sürecinde değişik komutların segmentleri hafızayla etkileşime ihtiyaç duyabilir (5.clock sürecini incele). Bunun için Pipeline datapath **harward** mimarisi ile yapılır.
- Dikkat Single cycle datapath, harward mimarisi, multicycle datapath van neuman mimariye sahiptir.

Pipelined Datapath



UNIFORM'luk basitlik getirir. MIPS'te bütün komutları 5 state olarak düşünelim.

- Bütün komutlar 5 periyotluk zaman alır. Bütün durumlar için benzer uzunlukta clock kullanılır.
- Bu durumda bazı komutlar, bazı stage'leri iş yapmadan geçirecektir.

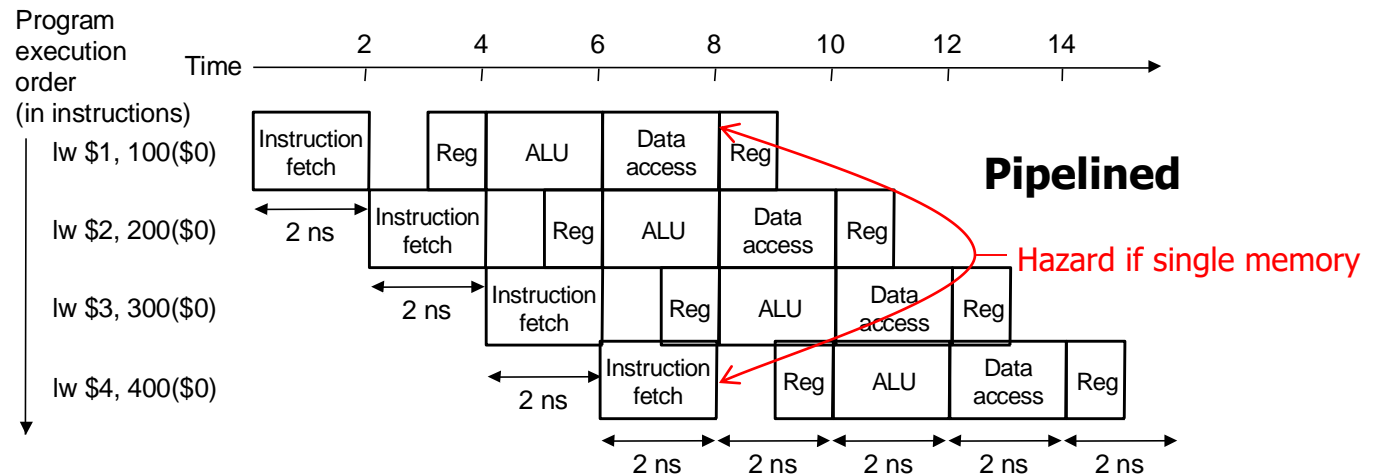


MIPS'te Pipeline

- Tehlikeler (Hazard), bir pipeline yürütülme sürecindeki çatışmaları (aynı birimi farklı fazların kullanma isteği) işaret eder. Bunlar nelerdir?
- Structural hazards (Yapısal tehlikeler): Pipeline işleminde, farklı komutlarda , farklı durumlarda aynı donanım kaynaklarının kullanılması durumu oluşabilir (Kaynak Çatışması).
- Control hazards (Kontrol tehlikeleri): Bir önceki dallanma komutunun sonucuna bağlı olarak, izleyen komutu mevcut pipeline içine alma sorunu (Dallanma zorluğu).
- Data hazards (Data tehlikeleri): Pipeline'daki bir komutun, aynı pipelin'daki önceki komutun hesapladığı datayı kullanması gerektiğinde (Veri Beklentisi).
- Biz, pipeLine'lı datapath'ı incelemeden önce yukarıda sıralanan olası tehlikeleri tek tek incelemeliyiz.

Structural Hazards (Yapısal Tehlikeler)

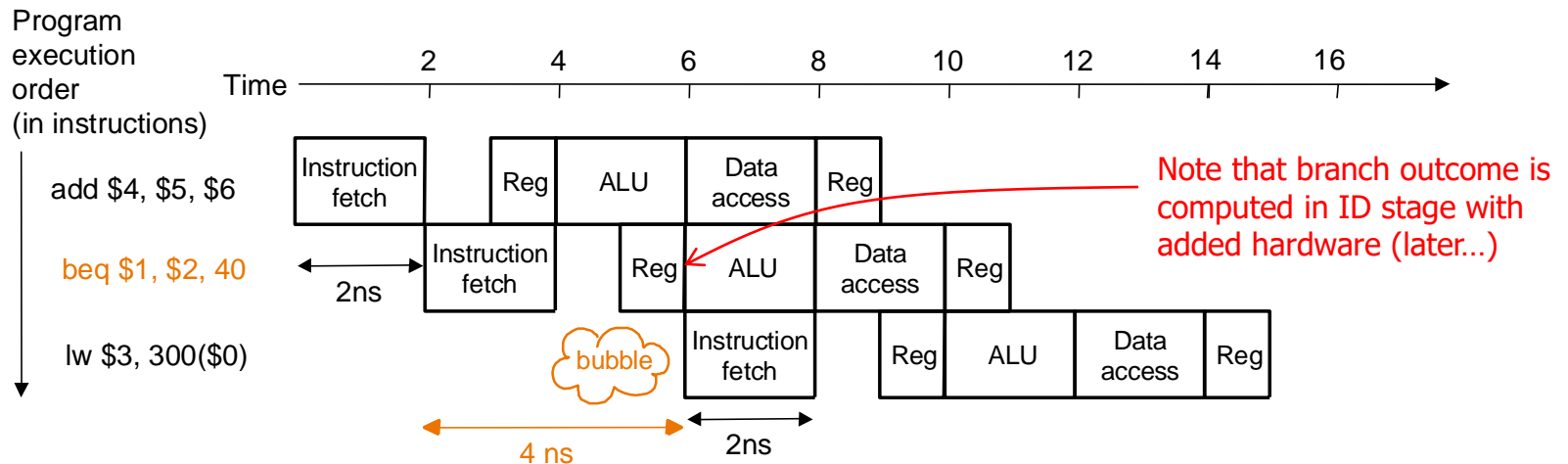
- **Structural hazard** : Pipeline işleminde aynı clock cyclında bütün komutların eşzamanlı işlemesi için gereken donanımın yetersizliği.
- Örnek: Varsayınız ki, komut ve data hafızası tek olan, yani tek bir veriyolu kullanan, bir yapıda pipeline işlemi yapılsın.
 - O zaman 1. ve 4. lw komutları arasında bir structural hazard oluşabilir.
 - Veya farklı komutların farklı segmentleri aynı toplayıcıyı kullanmak durumunda olabilir.



- *Korkmayın MIPS PIPELINE'ı destekler şekilde dizayn edilmiştir. Yapısal tehlikelerden kolayca kaçınılır.*

Control Hazards

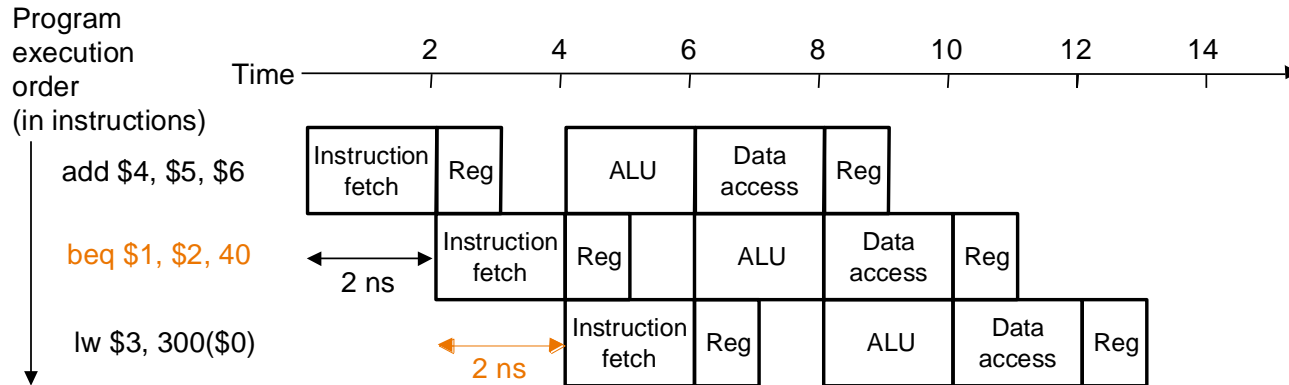
- *Control hazard (Kontrol Tehlikesi)*: PIPELINE'in yürütülmesinde bir önceki komutun sonucuna göre karar vermek gerekebilir. Bu problem genellikle şartlı ve şartsız dallanma komutlarının işlenmesi durumunda çok önemlidir.
- Çözüm 1 *Stall* the pipeline (Pipeline'ı geciktirme)



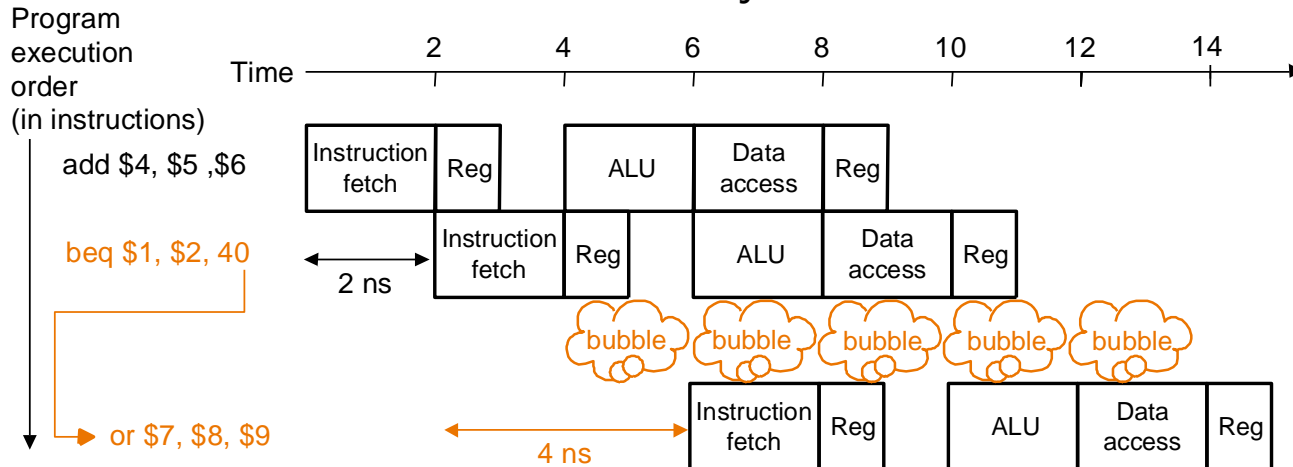
Pipeline stall (Pipeline gecikmesi)

Control Hazards

- Çözüm 2 *Predict* branch outcome (dallanma sonucunun önceden tahmin edilmesi veya dallanmanın olmayacağını tahmini)



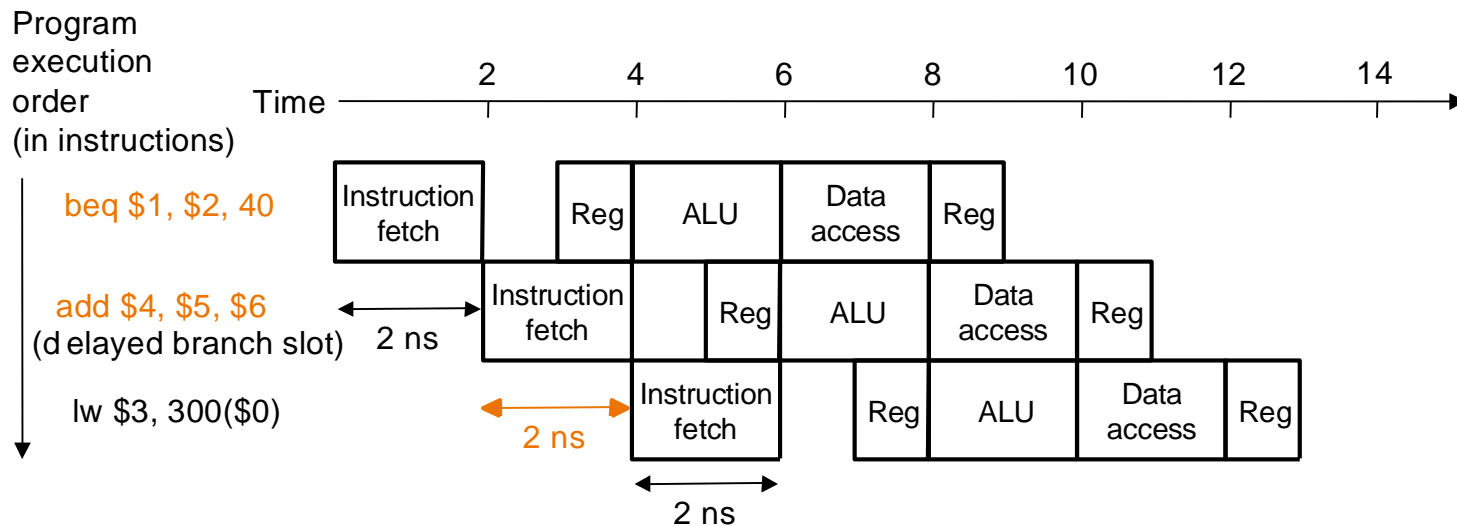
Tahmin başarısı



Tahmin hatası: lw'ü geri al (=flush)

Control Hazards (Kontrol Tehlikesi)

- Çözüm 3: Gecikmiş dallanma (*Delayed branch*) :Derleyici , dallanmaları yeniden düzenleyerek, zamanında pipeline içine olmalarını sağlar. Bu yöntemde derleyici dallanma komutundan önceki ve sonraki komutları analiz ederek geciktireceği komutların önüne faydalı komutlar yazarak program akışını yeniden düzenler. Yani dallanma komutuna sıra gelmeden önündeki komutları, akışı bozmamak şartıyla gecikme adımlarına alıp bu esnada dallanma komutunuda diğer kesimlerde işlemeye devam eder. Böylece dallanmanın gecikmesi önlenmiş olur.



Delayed branch beq is followed by add that is independent of branch outcome

Gecikmiş Dallanmaya Örnek:

5 komutluk bir program olsun. Komutlar sırayla;

1- Hafızadan R1'e yükle

2-R2'yi arttır.

3-R3 e R4'ü topla

4-R5'i R6'dan çıkar

5-X adresine dallan

Aşağıda derleyicinin yapabileceği 2 gerçekleştirme verilmiştir.

Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to X					I	A	E			
6. NOP						I	A	E		
7. NOP							I	A	E	
8. Instr. in X								I	A	E

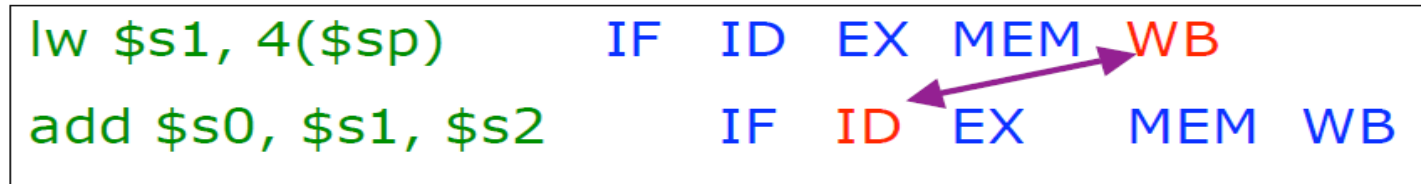
İşlem yok komutu kullanılarak

Clock cycles:	1	2	3	4	5	6	7	8
1. Load	I	A	E					
2. Increment		I	A	E				
3. Branch to X			I	A	E			
4. Add				I	A	E		
5. Subtract					I	A	E	
6. Instr. in X						I	A	E

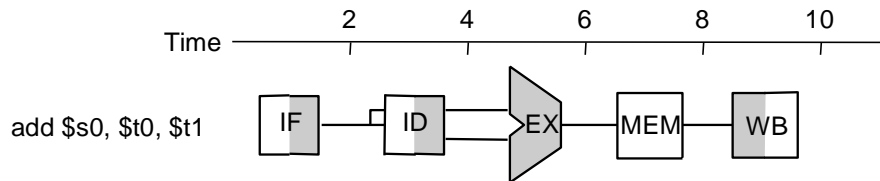
Komutları tekrar sıralayarak

Data Hazards (Veri Tehlikeleri)

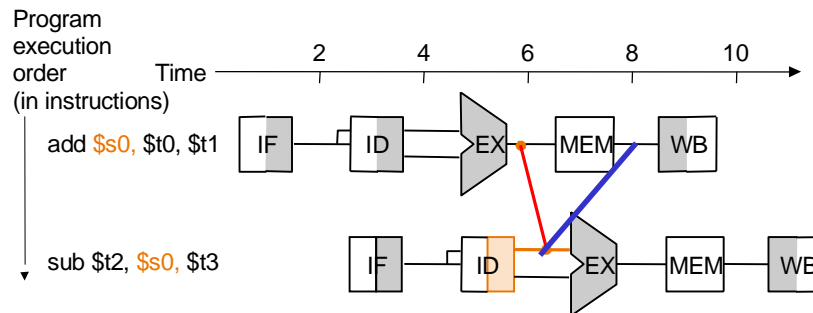
- Data hazard : Bir sonraki komut, yürütmesi bitmemiş bir önceki komutun datasına ihtiyaç duyabilir.



- Çözüm : *Mümkünse datayı ilerlet....*



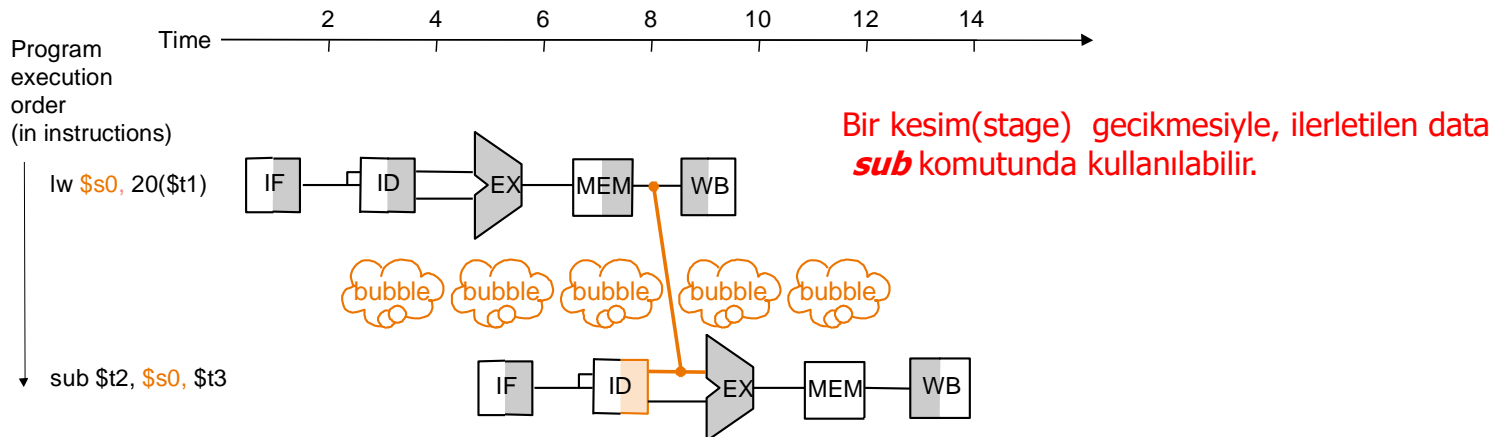
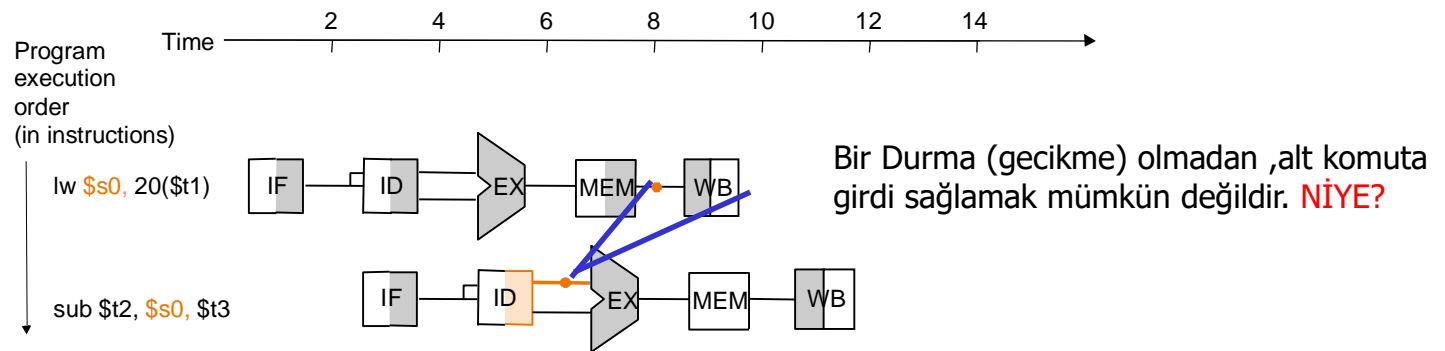
Komutun Pipeline diyagramı:
shade indicates use –
left=write, right=read



İlerletme yoksa – mavi çizgi –
Veri zamanda geri gitmek
durumunda. Yani 2.komut 6,8
arasında bekleyecek.;
İlerletme varsa– kırmızı çizgi –
Veri, zamanında hazır edilmiş.
2.komut 6,8 süresinde işlem
yapar.

Data Hazards


- Datayı ilerletmeninde yetmeyeceği durumlar olabilir.
- Örneğin, R-tipi bir komut yükleme komutunu takip ediyorsa, yüklenen değeri kullanacaksa – *load-use data hazard*'i olarak bilinir.



PIPELINE gecikmesinden (Stall) kaçınmak için kodlamayı yeniden düzenleme.(Software Solution)

■ Example:

```
lw $t0, 0($t1)
lw $t2, 4($t1)
sw $t2, 0($t1)
sw $t0, 4($t1)
```




A red curved arrow points from the `$t1` register in the second instruction (`lw $t2, 4($t1)`) to the `$t1` register in the third instruction (`sw $t2, 0($t1)`), indicating a data hazard.

Data hazard

■ Reordered code:

```
lw $t0, 0($t1)
lw $t2, 4($t1)
sw $t0, 4($t1)
sw $t2, 0($t1)
```



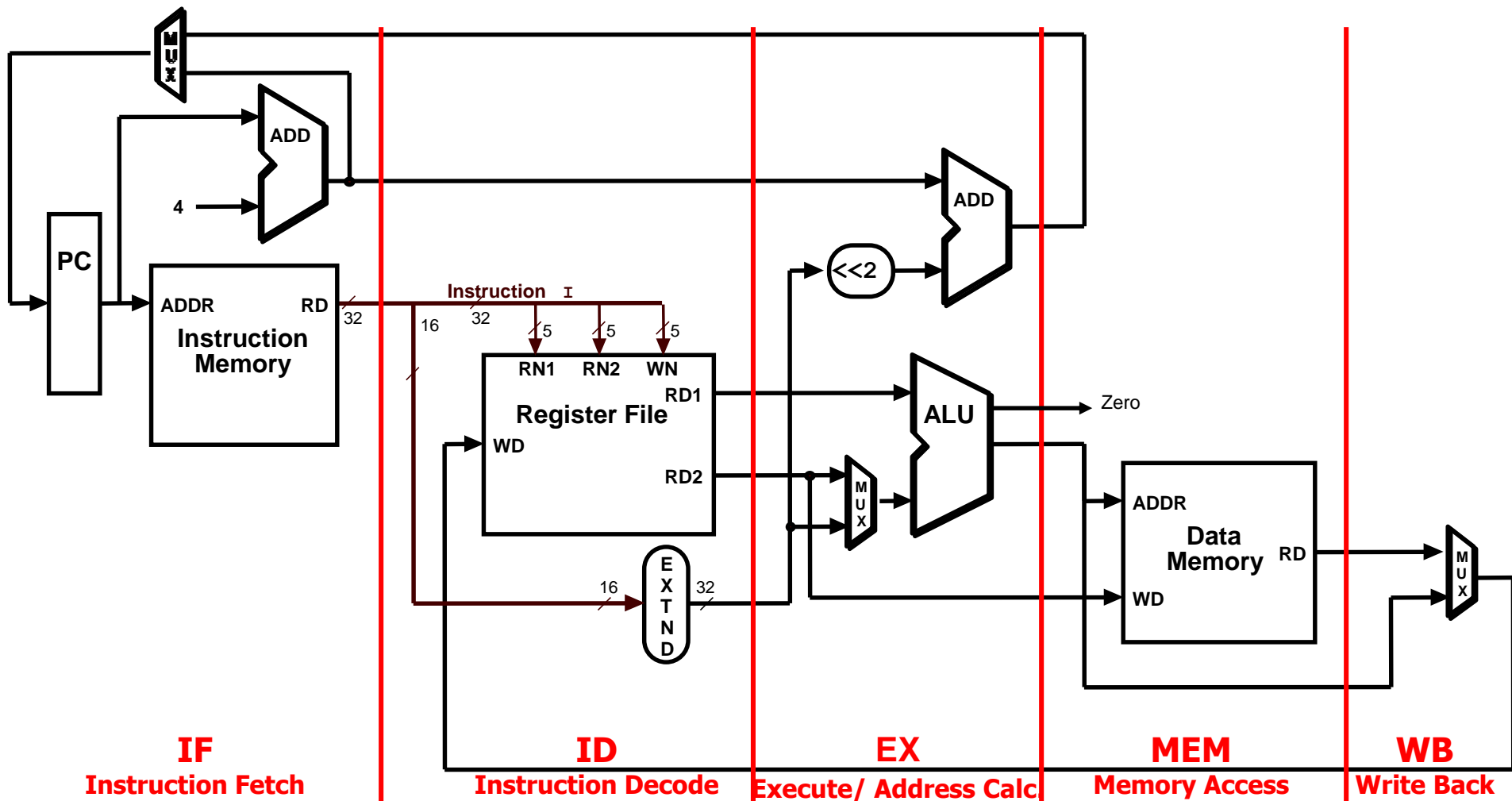
A red curved arrow points from the `$t1` register in the second instruction (`lw $t2, 4($t1)`) to the `$t1` register in the third instruction (`sw $t0, 4($t1)`), indicating an interchange.

Interchanged

Pipeline çalışan Datapath

- Şimdi bir pipeline datapath yapısını açıklayabiliriz.
- İlk önce bir komutun tamamlanmasındaki 5 adımı hatırlayalım.
 1. Instruction Fetch & PC Increment (IF)
 2. Instruction Decode and Register Read (ID)
 3. Execution or calculate address (EX)
 4. Memory access (MEM)
 5. Write result into register (WB)
- Hatırlatma: single-cycle işlemci için
 - 5 adımın hepsi birden single clock cycle'ında yapılır.
 - Herbir adım için atanmış bir donanıma (fonsiyonel yapıya) ihtiyaç vardır.
- *Biz single cyle datapath donanımını değiştirmeden, bir komutun işlenmesini birden fazla cycle'da gerçekleşecek şekilde parçalasaydık neler oluşurdu?*
- *What happens if we break the execution into multiple cycles, but keep the extra hardware?*

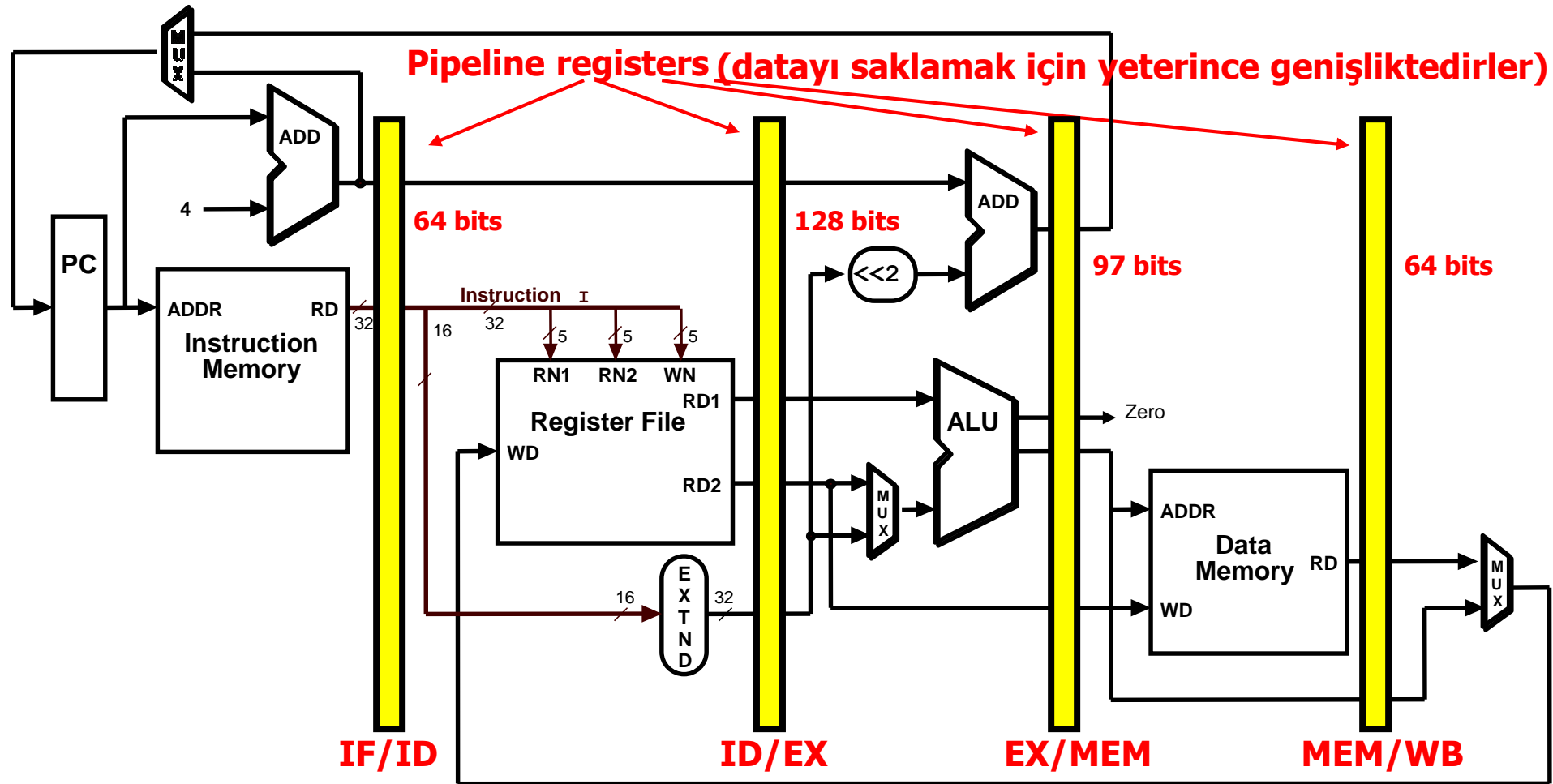
Hatırlatma - Single-Cycle Datapath "Steps"



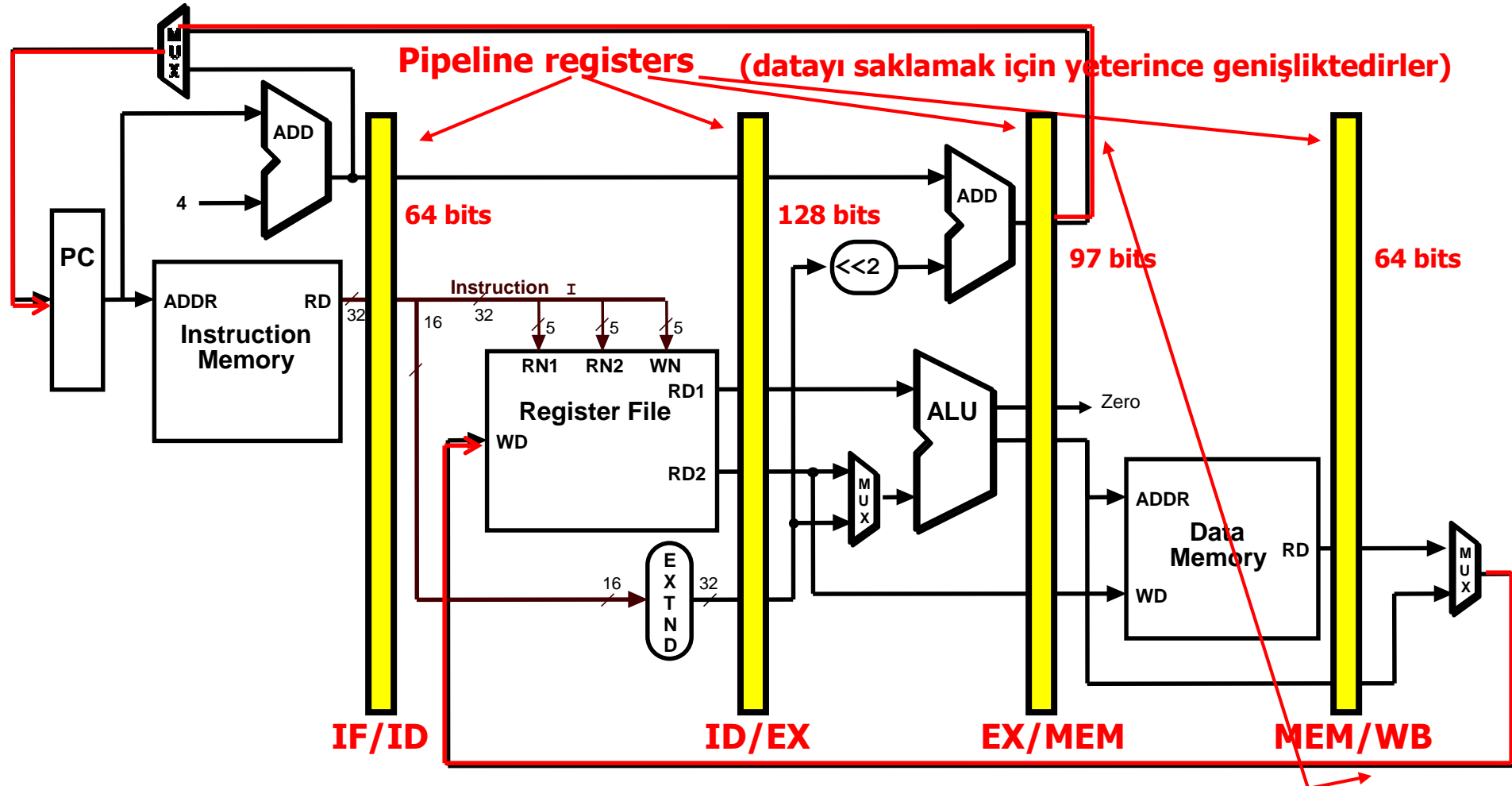
Pipelined Datapath – Temel Fikirler

- *Extra donanımları korumak şartıyla, bir yürütme işlemini bir çok cycle'a bölersek ne olur?*
- *What happens if we break the execution into multiple cycles, but keep the extra hardware?*
 - Cevap: Her bir klok çevriminde yeni bir komutu işlemeye başlayabiliriz.
 - *We may be able to start executing a new instruction at each clock cycle - pipelining*
- İşte bu Pipeline işlemidir.
- Fakat; bu cycle'lar arasındaki süreçte (segment-Kesim) datayı tutmak için extra registerlara ihtiyaç vardır. Bunlar *pipeline register'larıdır*.
- *Önemli not:* *Pipeline yürütmede herbir kesimde(segmentte) elde edilen sonuç bir sonraki kesime aktarılabilir. Aynı kesimde farklı komutlar için eş zamanlı olarak farklı datalar ile farklı işlemler yapılabilir.*
 - Dolayısıyla her kesim için bir *pipeline register* kullanılmalıdır.

Pipelined Datapath

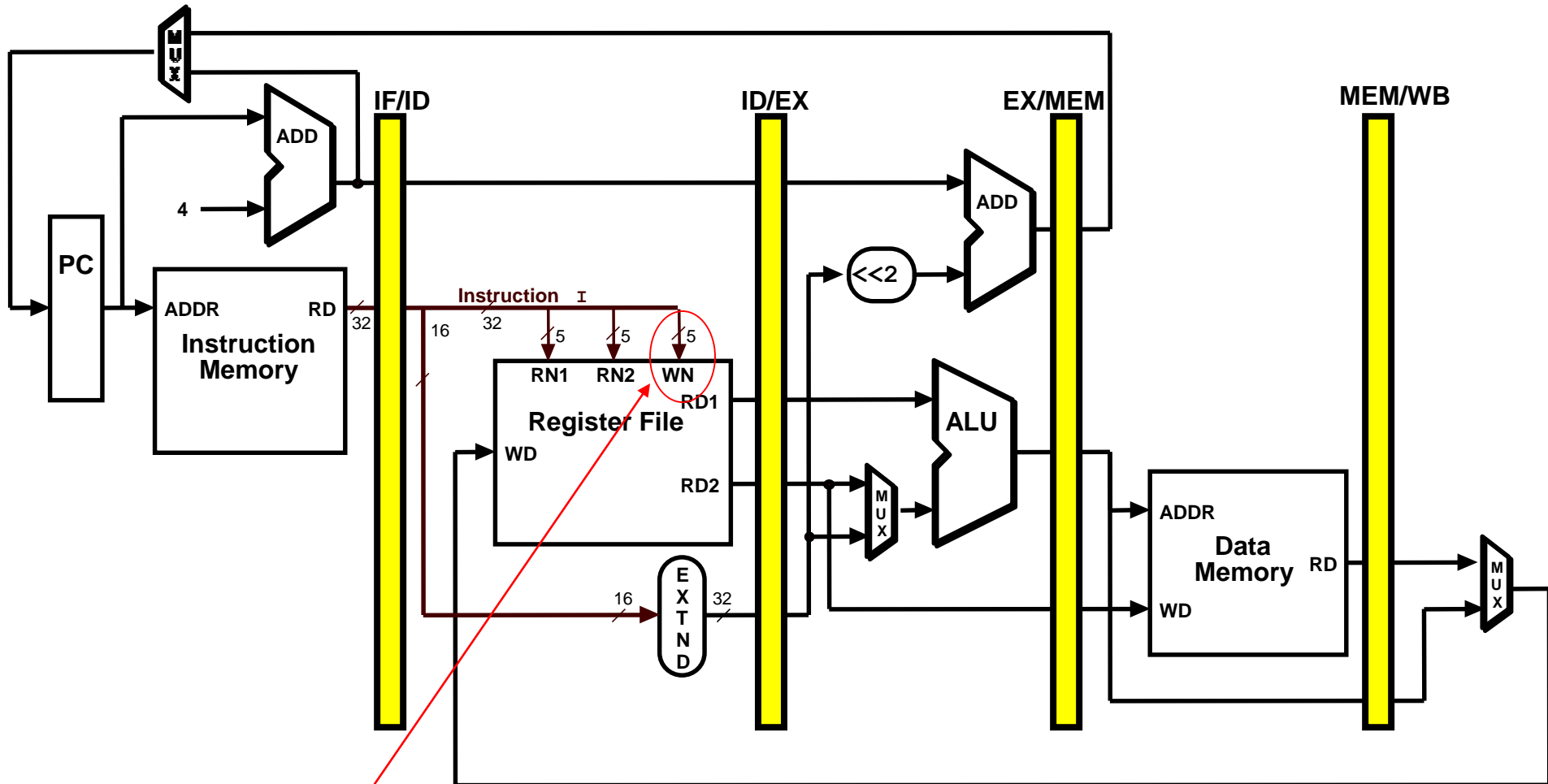


Pipelined Datapath



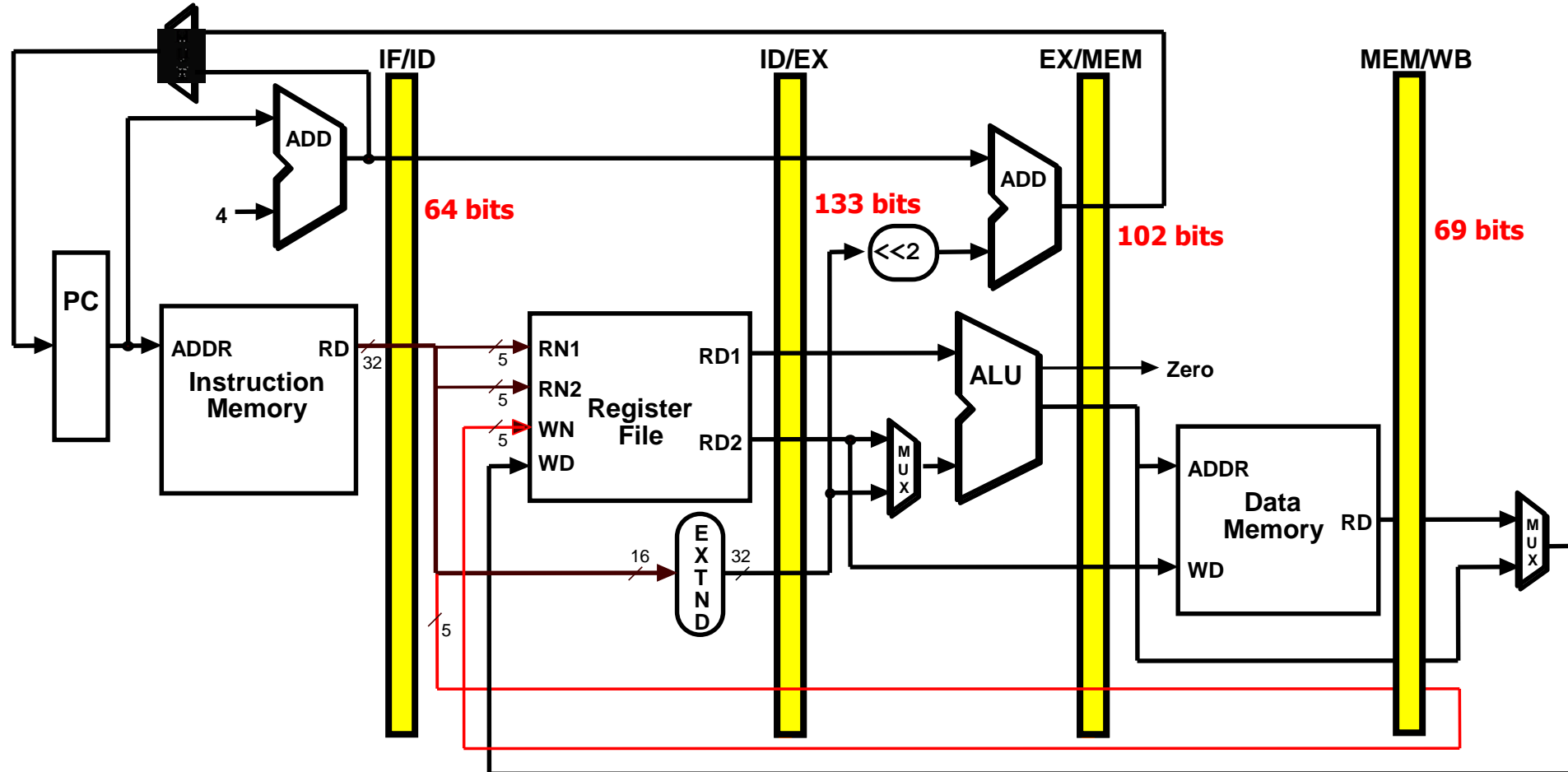
Sadece soldan sağa doğru akan veri durumunda tehlikeler oluşabilir? neden?

Datapath'de hatalar



Hedef reg'e yazılma süreci, sonraki bir komutun başka bir işlem fazından sonraya denk gelebilir.

Düzeltilmiş DATAPATH



Hedef Reg'in adresi (5 bitlik) ID/EX, EX/MEM ve MEM/WB reg'lerinede yazılır. Şimdi bu reg'lerin genişlikleri 5 bit artmıştır.

Pipeline Örnekleri

- Aşağıdaki komut dizisini gözönünde bulundur.

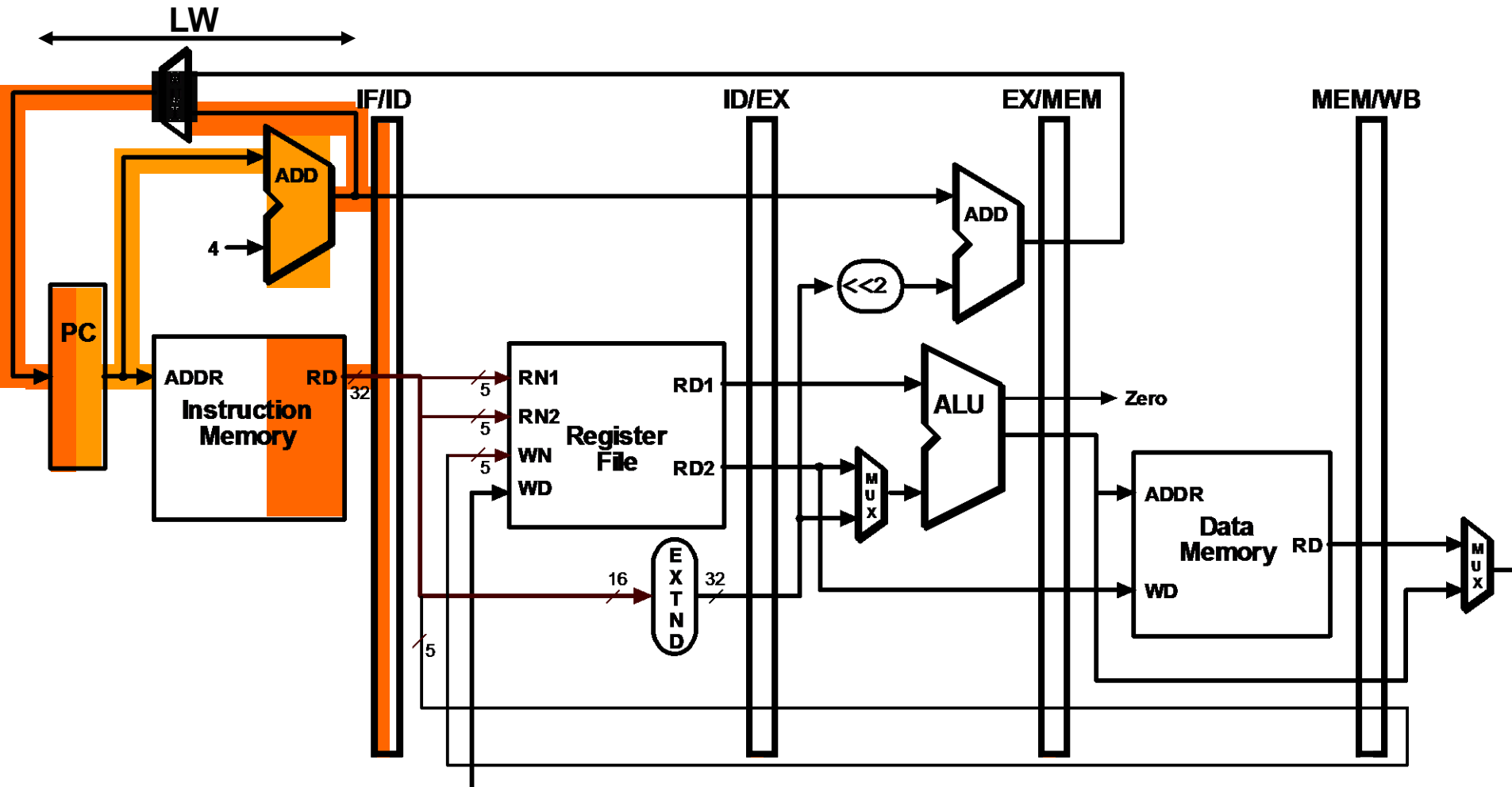
```
lw    $t0, 10($t1)
```

```
sw    $t3, 20($t4)
```

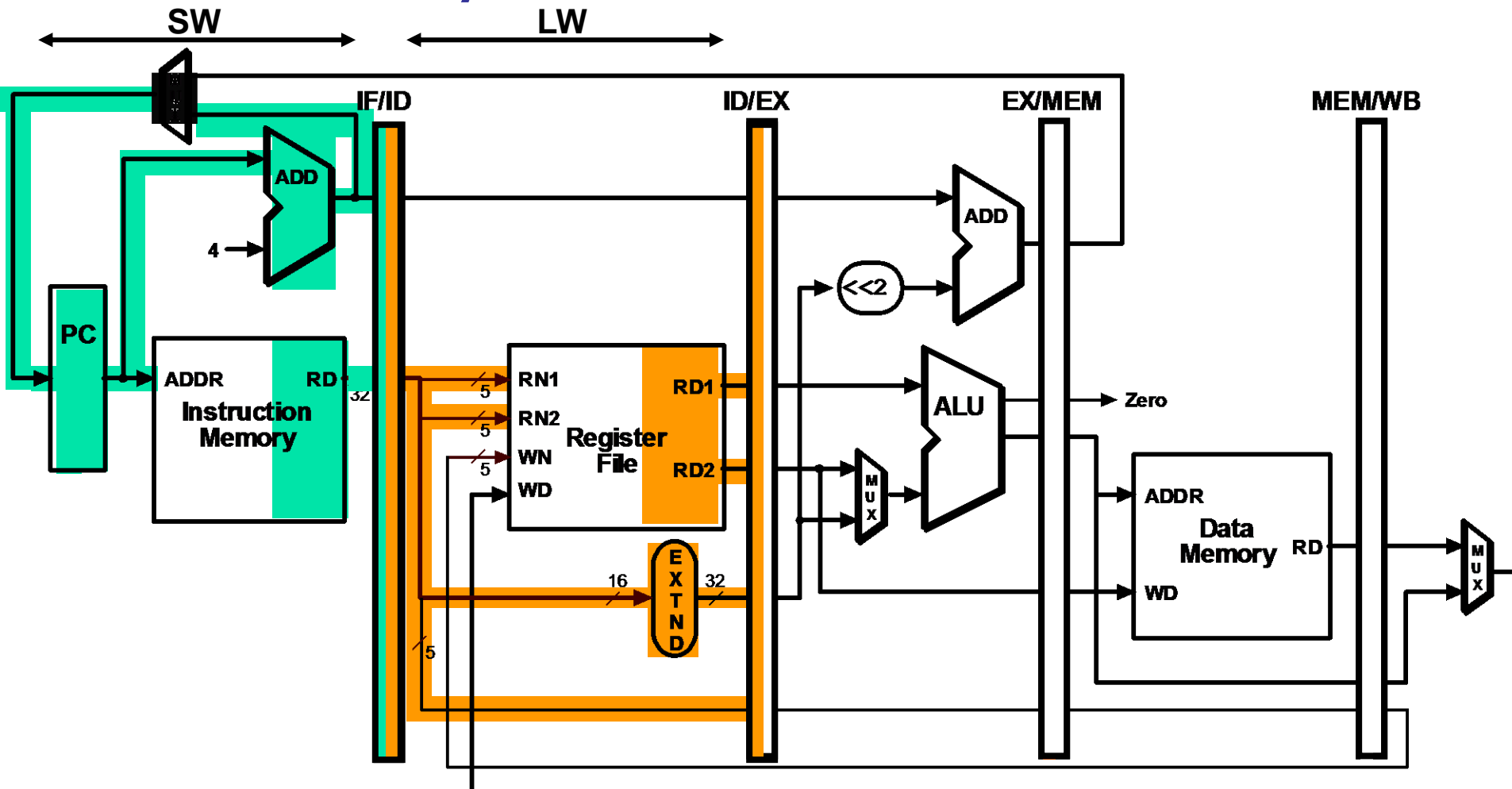
```
add   $t5, $t6, $t7
```

```
sub   $t8, $t9, $t10
```

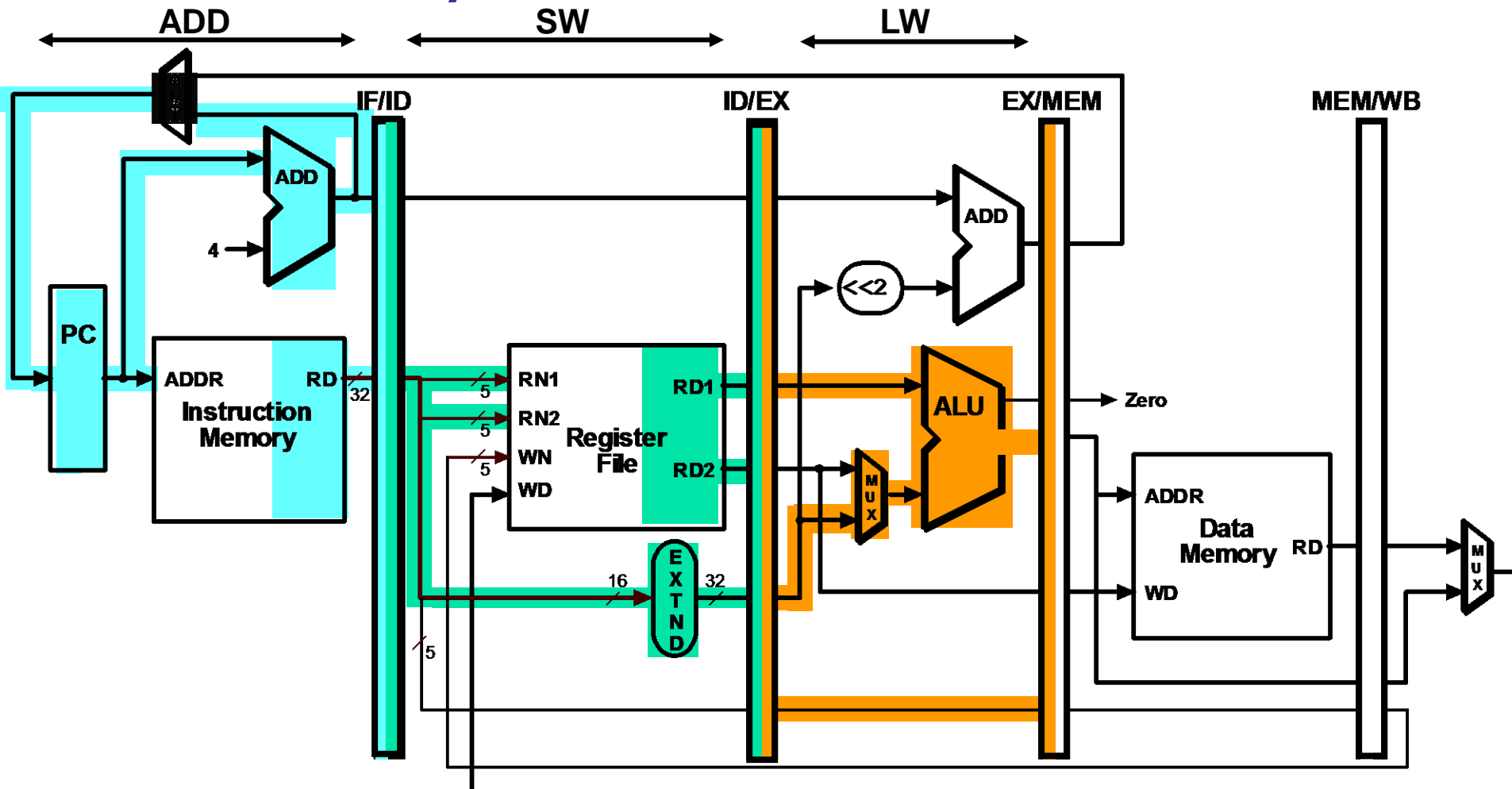
Single-Clock-Cycle Diagram: Clock Cycle 1



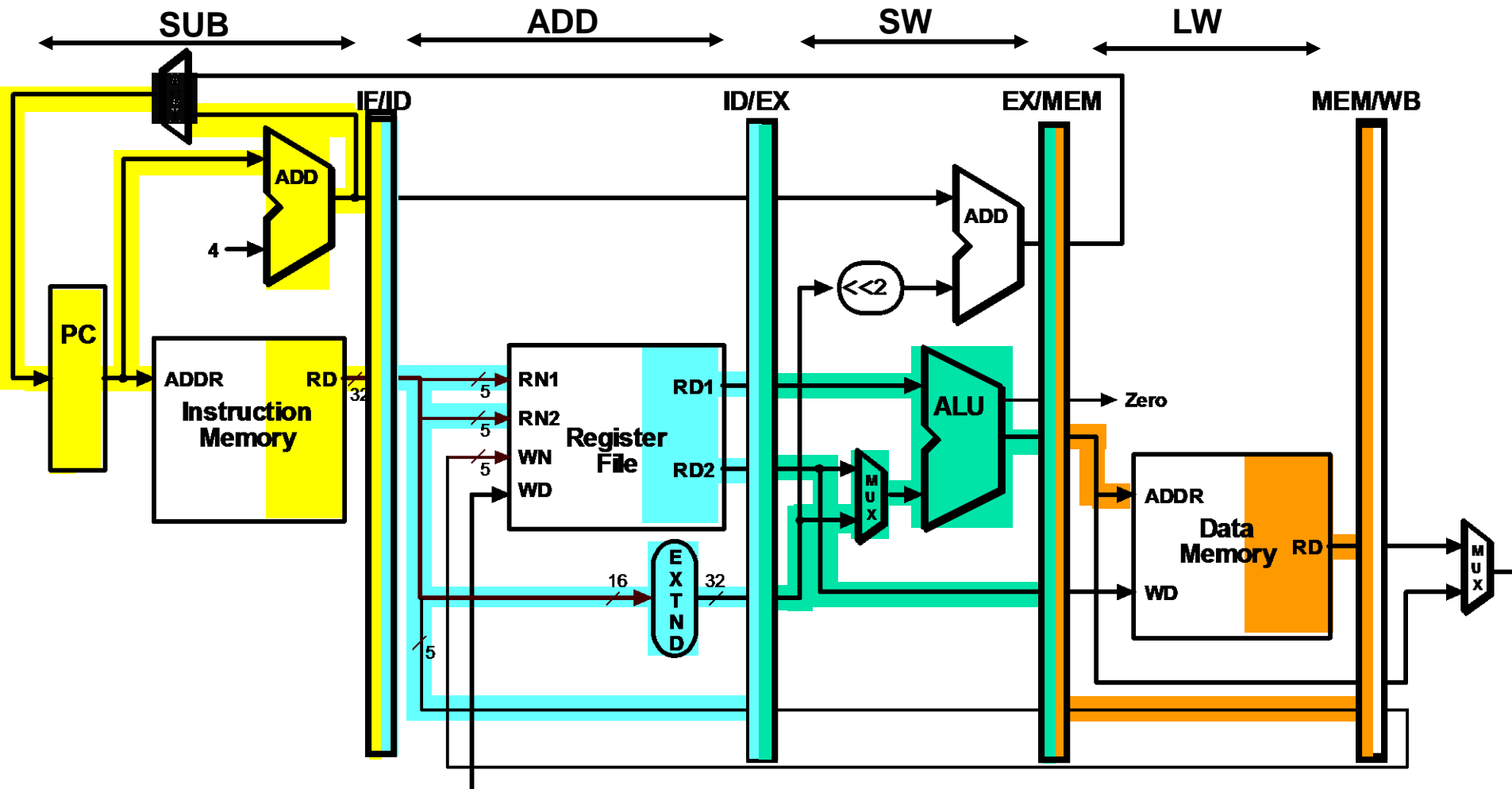
Single-Clock-Cycle Diagram: Clock Cycle 2



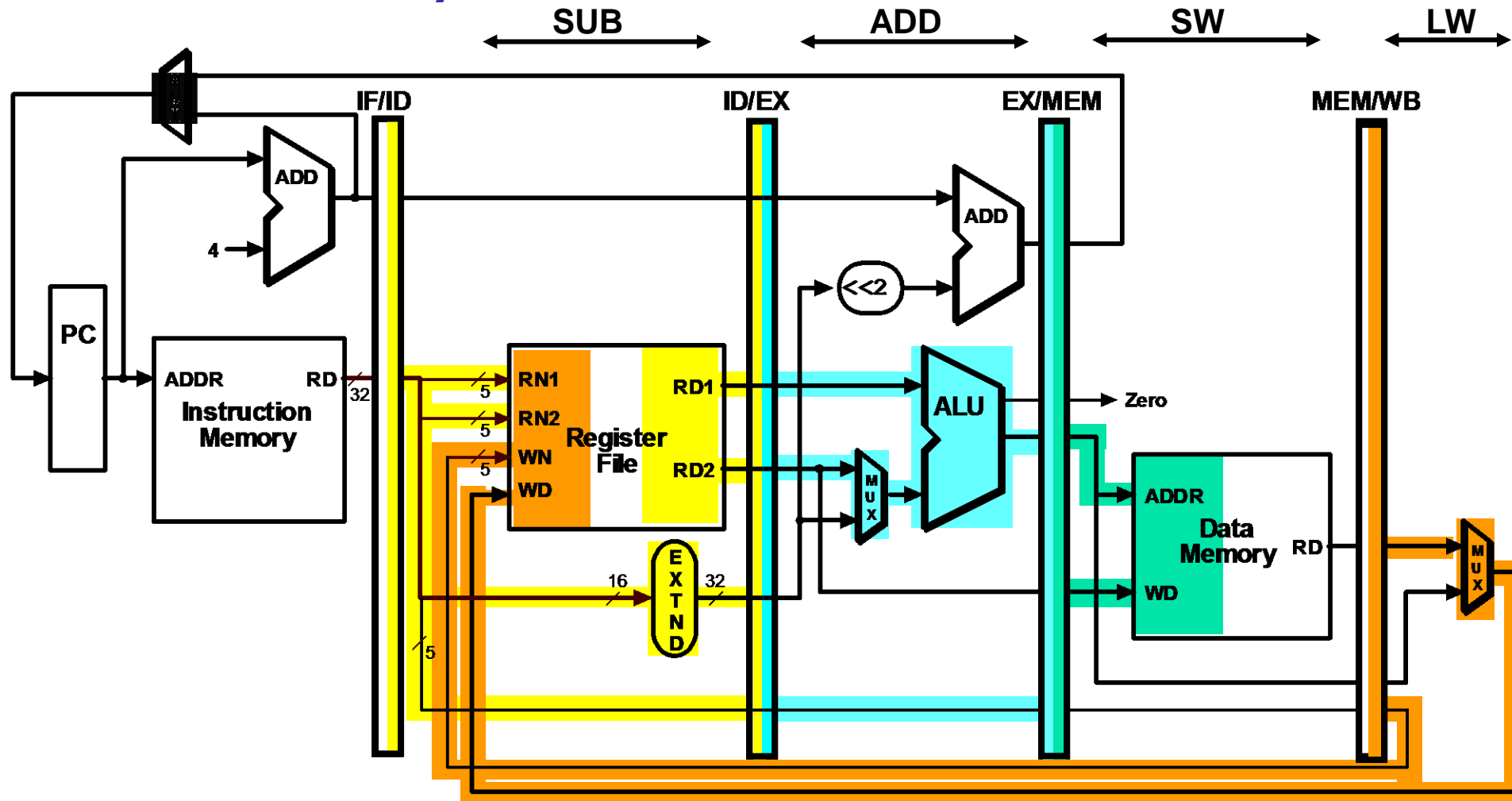
Single-Clock-Cycle Diagram: Clock Cycle 3



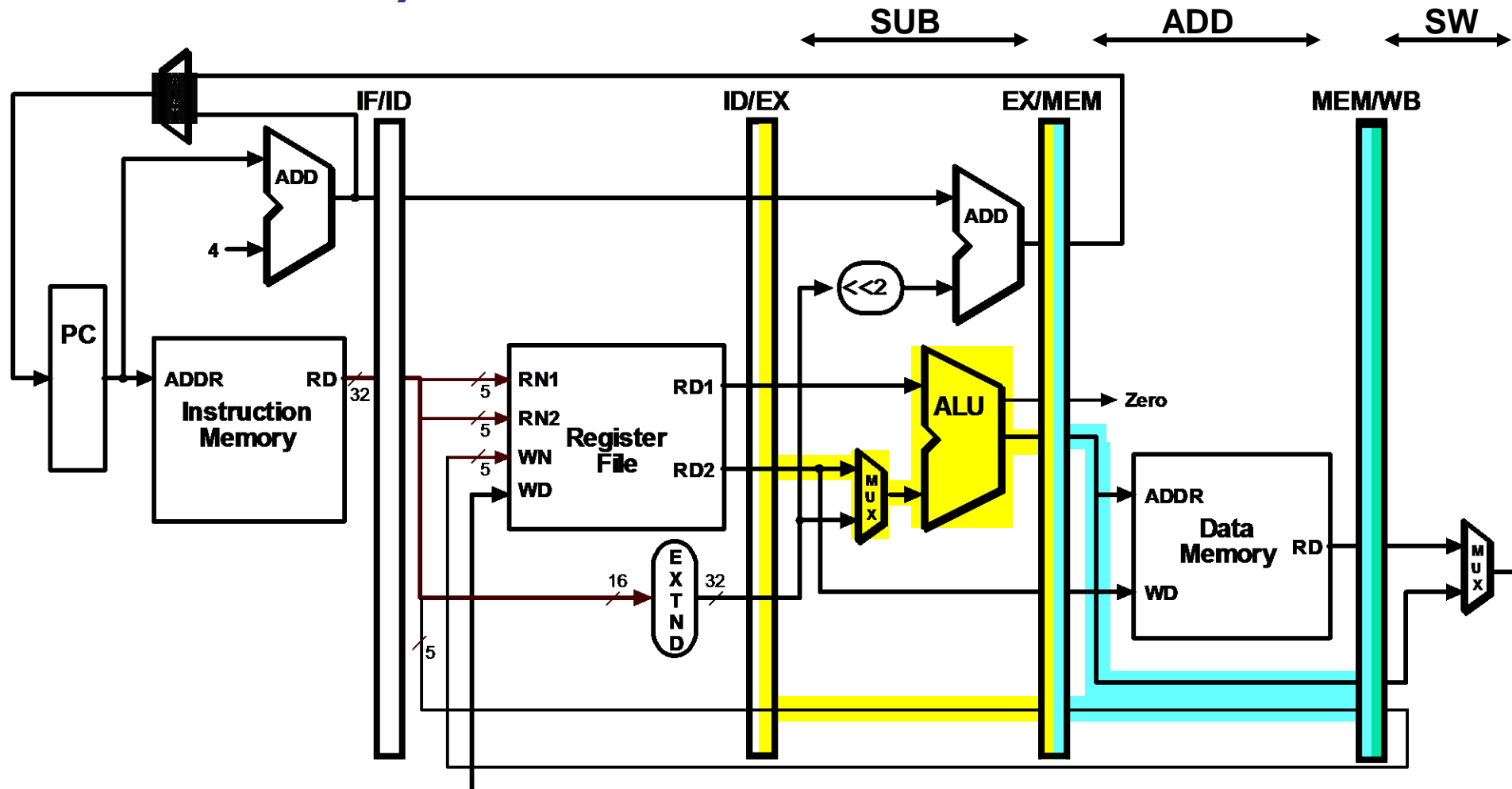
Single-Clock-Cycle Diagram: Clock Cycle 4



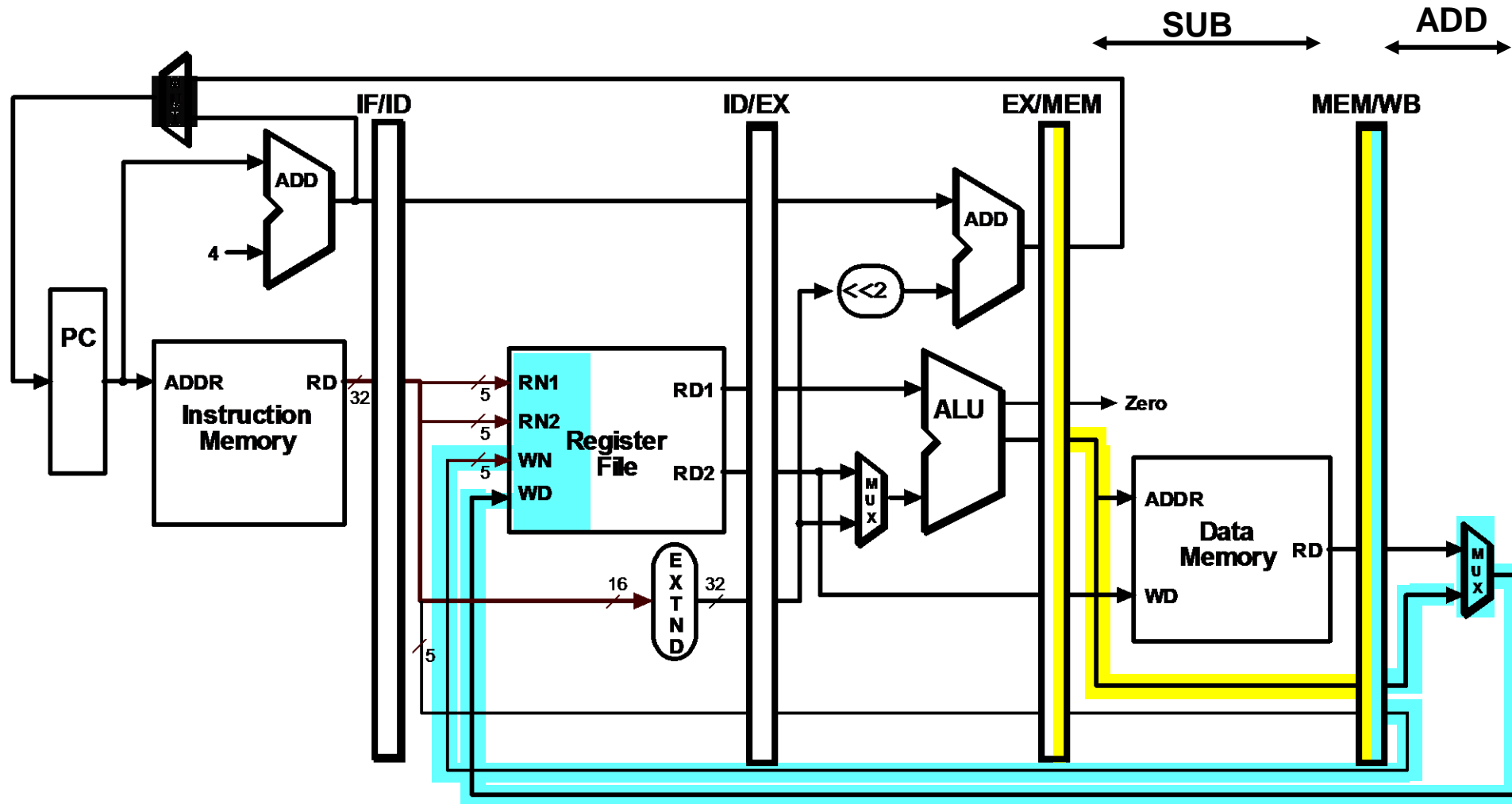
Single-Clock-Cycle Diagram: Clock Cycle 5



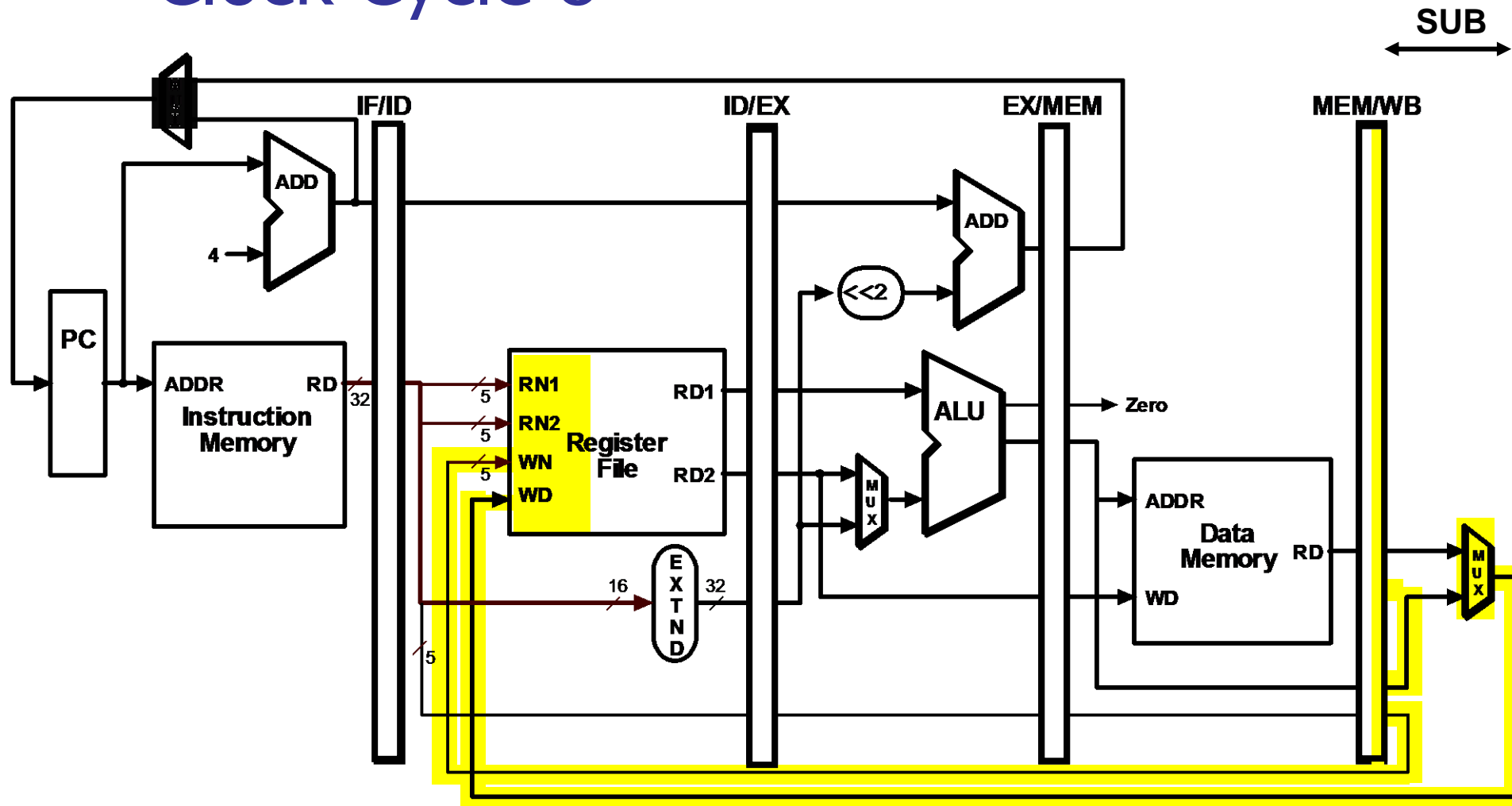
Single-Clock-Cycle Diagram: Clock Cycle 6



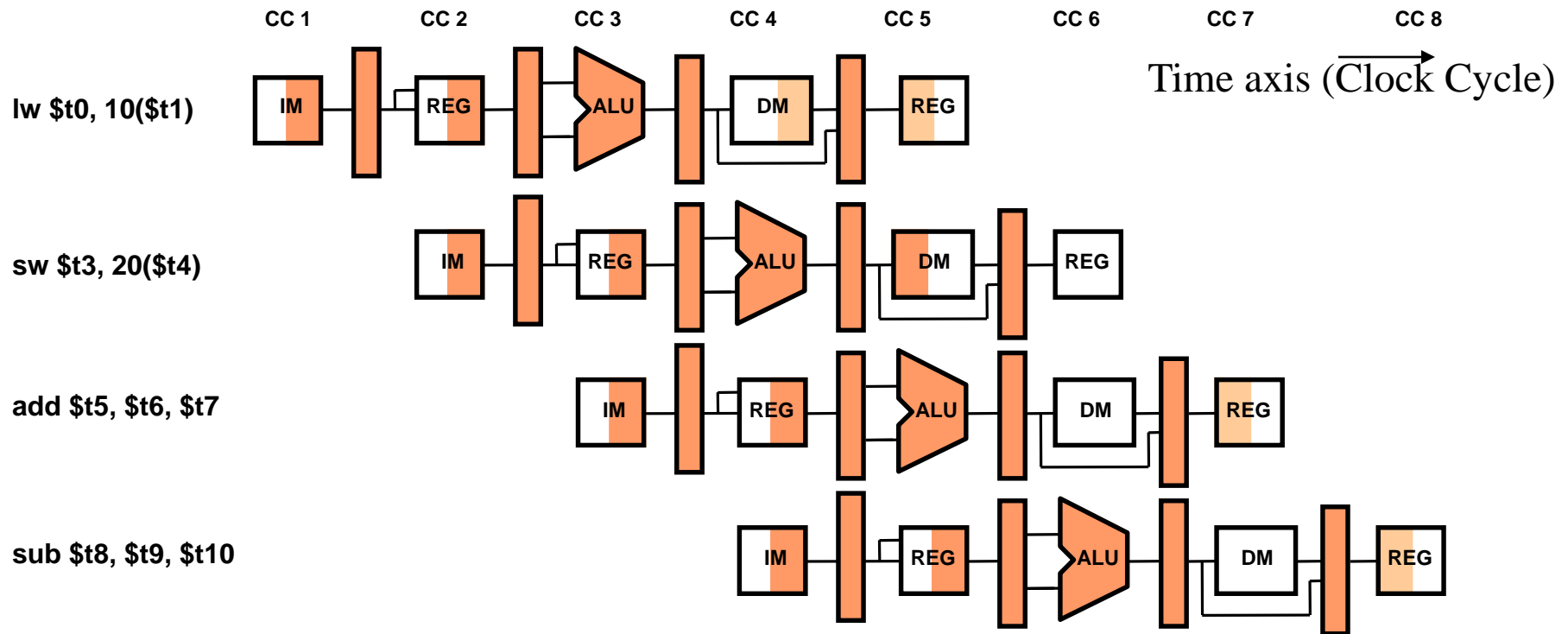
Single-Clock-Cycle Diagram: Clock Cycle 7



Single-Clock-Cycle Diagram: Clock Cycle 8



Alternative View – Multiple-Clock-Cycle Diagram



Not:

- Bir R-tipi komutunun yürütülmesi sürecinde; Multicycle ve pipeline gerçekleştirmeler arasında önemli fark:
 - R tipi bir komutta, registere write-back işlemi 5. pipeline kesimindedir (last write-back segmentindedir). Oysa multicycle gerçekleştirmede ise bu işlem 4. durumda gerçekleşir. Niyen?
 - Reg.file'a yazarken yapısal tehlikelerden dolayı olabilir.
- *Tekrar etmeye değer:* Boru hattı ve multicycle uygulamalar arasındaki temel fark, 5 durumu birbirinden ayırtırmak için pipeline register yerleştirilmesidir.
- İdeal Pipeline (Duraksama-gecikme yosa) için $CPI = 1$ 'dir. Niye?

Simple Example: Comparing Performance

- *Compare performance for single-cycle, multicycle, and pipelined datapaths using the gcc instruction mix*
 - assume 2 ns for memory access, 2 ns for ALU operation, 1 ns for register read or write
 - assume gcc instruction mix 23% loads, 13% stores, 19% branches, 2% jumps, 43% ALU
 - for pipelined execution assume
 - 50% of the loads are followed immediately by an instruction that uses the result of the load
 - 25% of branches are mispredicted
 - branch delay on misprediction is 1 clock cycle
 - jumps always incur 1 clock cycle delay so their average time is 2 clock cycles

Simple Example: Comparing Performance

- *Single-cycle* (p. 373): average instruction time 8 ns
- *Multicycle* (p. 397): average instruction time 8.04 ns
- *Pipelined*:
 - loads use 1 cc (clock cycle) when no load-use dependency and 2 cc when there is dependency – given 50% of loads are followed by dependency the average cc per load is 1.5
 - stores use 1 cc each
 - branches use 1 cc when predicted correctly and 2 cc when not – given 25% misprediction average cc per branch is 1.25
 - jumps use 2 cc each
 - ALU instructions use 1 cc each
 - therefore, average CPI is
$$1.5 \times 23\% + 1 \times 13\% + 1.25 \times 19\% + 2 \times 2\% + 1 \times 43\% = 1.18$$
 - therefore, average instruction time is $1.18 \times 2 = 2.36$ ns

Vektör İşlemleri

Birçok yazılım uygulaması (Görüntü işleme, insan genetiği, Sismik veri analizi v.b), büyük sayı dizileri üzerinde çok büyük sayıda aritmetik işlem yapmayı gerektirir. Bu sayılar çoğu zaman Floating Point sayı düzlemindeki tek boyutlu veya çok boyutlu diziler şeklinde toplanmışlardır. . Sıradan bilgisayarlar bu çalışmaları çok uzun sürede sonlandırabilir. Sayılan işlemlerin yapılabilmesi için iyi kaliteli (vektör ve paralel işleme özelliği olan) bilgisayarlara gerek vardır.

Vektör: Tek Boyutlu veri dizisidir.

Satır veya sütun vektörleri şeklinde ifade edilebilirler. $V = [V1 \ V2 \ V3 \dots Vn]$.

Elemanları V olan bir satır vektörü $V(I)$ şeklinde I indisi ile gösterilir ve bir hafızada dizi şeklinde saklanır.

Bir vektör olmayan işlemci ile vektör işlemci arasındaki farkı anlamak için aşağıdaki döngüyü göz önüne alalım.

```
DO 20 I = 1, 100
20 C(I) = B(I) + A(I)
```

- Conventional computer

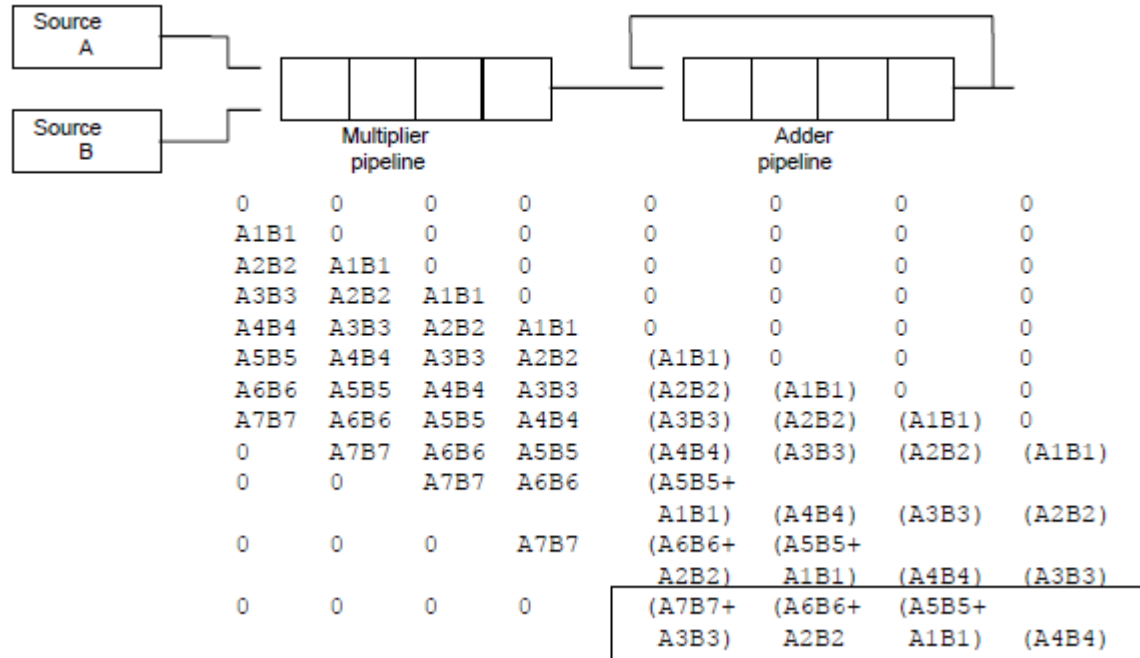
```
Initialize I = 0
20 Read A(I)
   Read B(I)
   Store C(I) = A(I) + B(I)
   Increment I = I + 1
   If I ≤ 100 goto 20
```

- Vector computer

```
C(1:100) = A(1:100) + B(1:100)
```


Operation code	Base address source 1	Base address source 2	Base address destination	Vector length
----------------	-----------------------	-----------------------	--------------------------	---------------

Vektör işlemci için komut formatı



İç çarpım hesabı için PIPELINE