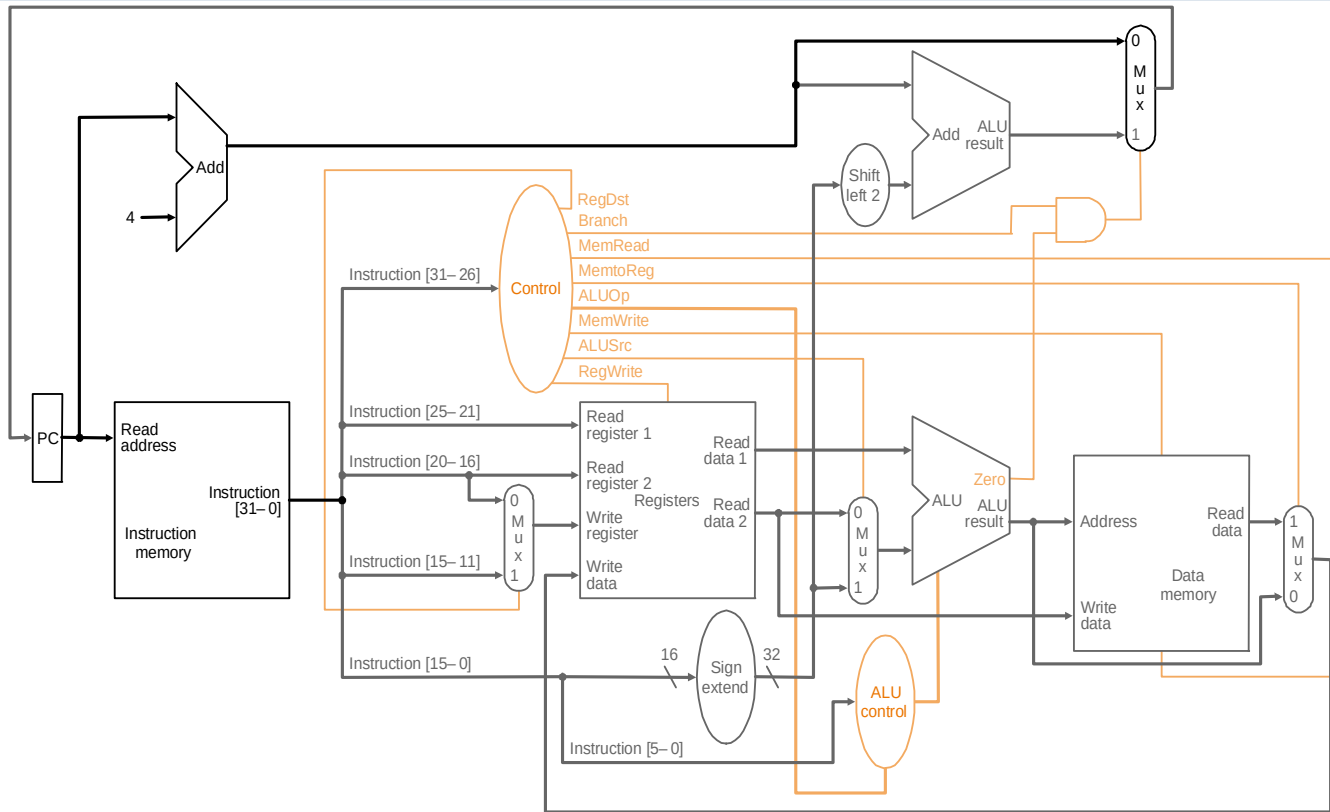


# Arquitetura de Computadores III

Pipeline Escalar de Instruções

# O Processador: Caminho de Dados e Controle

# Controle

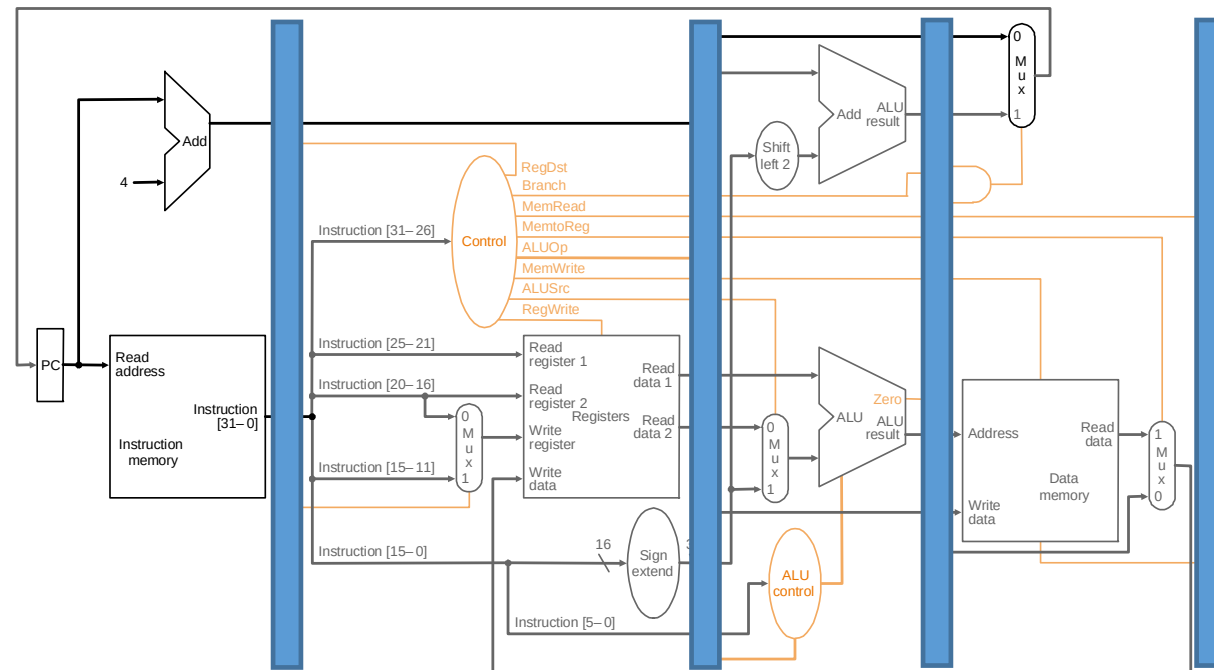


PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

9/15/22

Instruction	RegDst	ALUSrc	Memto- Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

# Controle

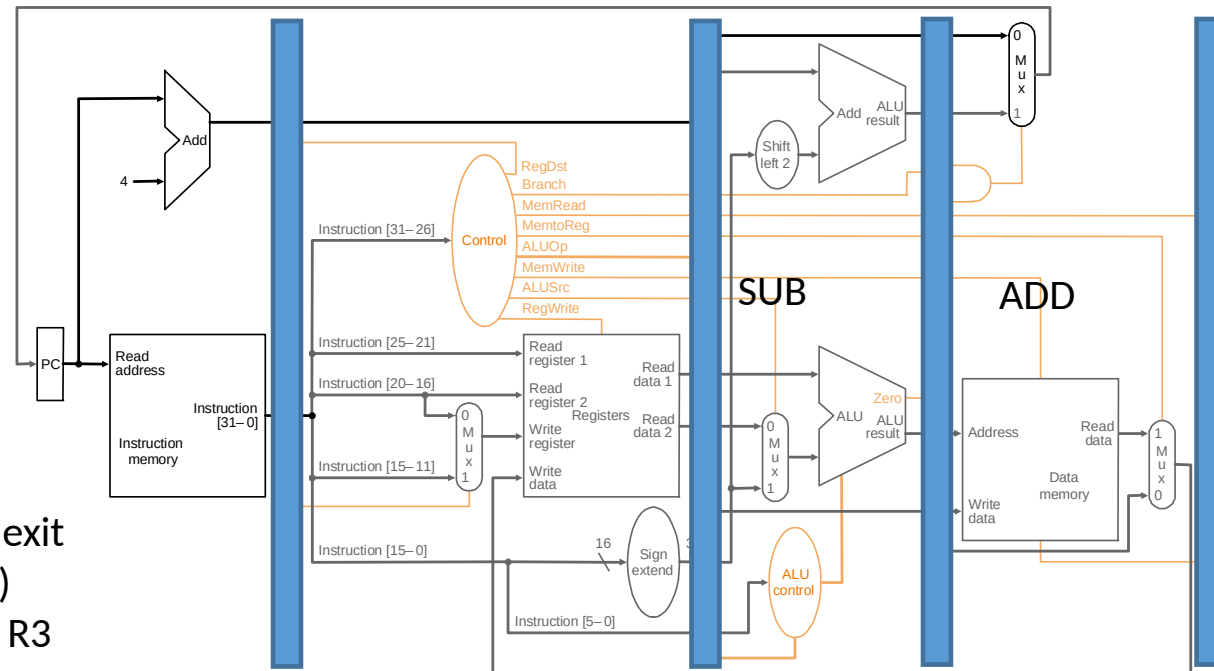


Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

# Controle

BEQ R1, R2, exit  
 LW R1, 0(R2)  
 ADD R1, R2, R3  
 SUB R4, R5, R1



Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

PATTERSON, David A.  
 Organização e projeto  
 de computadores a  
 interface  
 hardware/software

# Implementação de um único ciclo

Classe	Memória de Instruções	Leitura Registrador	Operação UAL	Memória de Dados	Escrita no registrador	Total
Formato R	2	1	2 (16)	0	1	6 ns
Load Word	2	1	2	2	1	8 ns
Store Word	2	1	2	2		7 ns
Branch	2	1	2			5 ns
Jump	2					2 ns
Formato R sem e com ponto flutuante						

PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

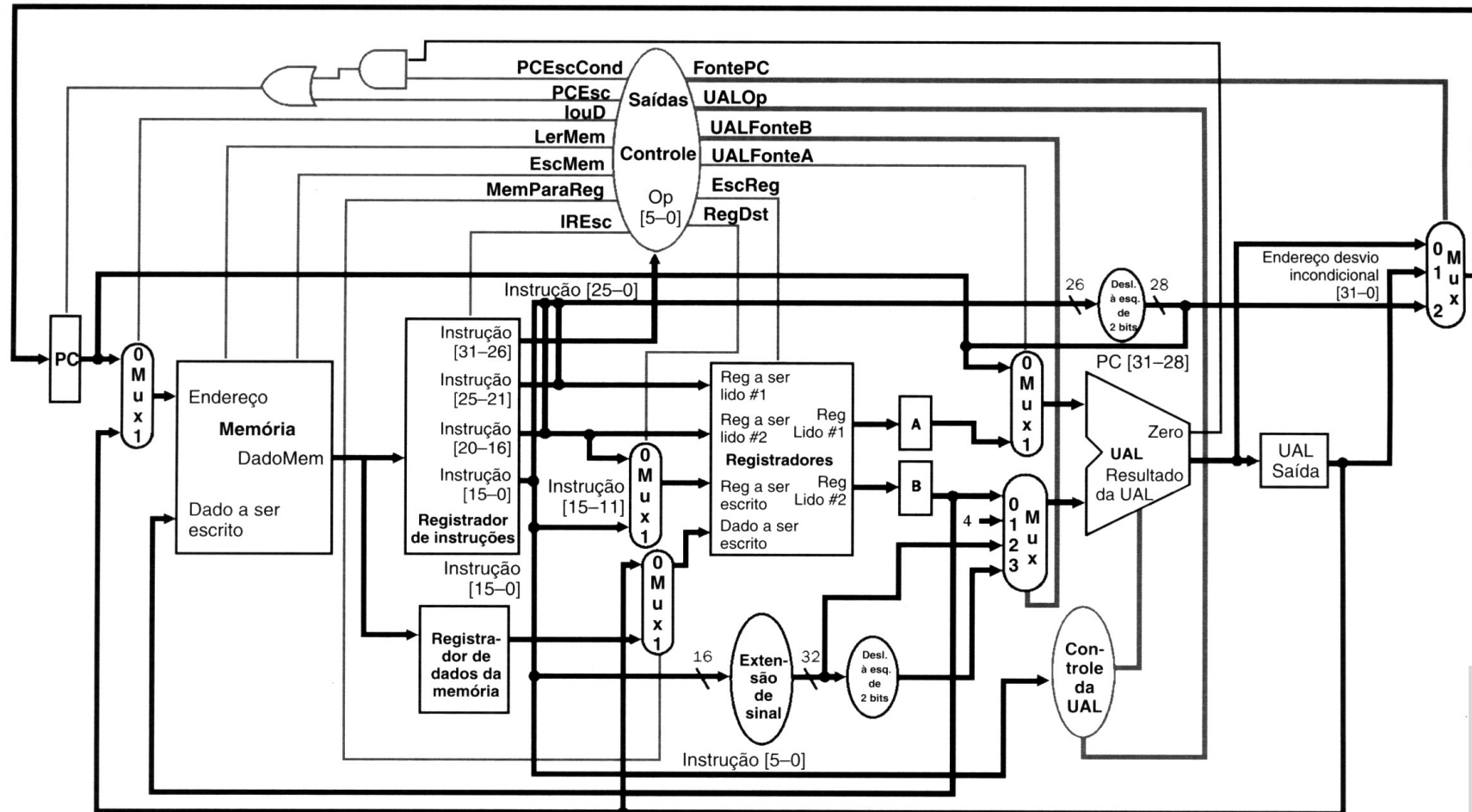
- O ciclo de clock é definido em função da duração da instrução mais longa = 8 ns.
- Uma máquina com ciclo de clock variável: 2ns a 8ns.
  - Ciclo de clock:  $8 \times 24\% + 7 \times 12\% + 6 \times 44\% + 5 \times 18\% + 2 \times 2\% = 6,3 \text{ ns}$
- Ganho de desempenho:  $\text{Tempo execução clock fixo} / \text{Tempo de execução clock variável} = 8 / 6,3 = 1,27$
- No caso de operações em ponto flutuante podemos ter para a multiplicação:  $2 + 1 + 16 + 1 = 20 \text{ ns}$
- A mesma relação pode ser feita para operações em ponto flutuante e o ganho favorável ao clock variável pode ser ainda maior.

# Cinco passos para execução multiciclo

- Busca de instrução
  - Pode ser descrito de forma sucinta usando a “Linguagem Transferência-Registrador” - RTL "Register-Transfer Language “
  - $IR = Memory[PC]$ ;
  - $PC = PC + 4$ ;
- Decodifica instrução e Busca Registrador
  - Leia os registradores rs e rt para o caso de precisarmos deles
  - Compute o endereço de desvio no caso da instrução ser um desvio
  - RTL:
    - $A = Reg[IR[25-21]]$ ;
    - $B = Reg[IR[20-16]]$ ;
    - $ALUOut = PC + (sign-extend(IR[15-0]) \ll 2)$ ;
  - Nós não ativamos linhas de controle baseados em tipo de instrução.
- Execução, Cálculo de Endereço de Memória, ou Conclusão de Desvio
  - A ULA está desempenhando uma das 3 funções, baseada no tipo de instrução
  - Referência à memória:  $ALUOut = A + sign-extend(IR[15-0])$ ;
  - Tipo-R:  $ALUOut = A \text{ op } B$ ;
  - Desvio: if (A==B)  $PC = ALUOut$ ;
- Acesso à Memória ou Conclusão de instruções tipo-R
  - Carrega ou armazena na memória:  $MDR = Memory[ALUOut]$ ; ou  $Memory[ALUOut] = B$ ;
  - Finaliza instruções Tipo-R:  $Reg[IR[15-11]] = ALUOut$ ;
- Passo de “Write-back”
  - $Reg[IR[20-16]] = MDR$ ;
- Instruções levam de 3 a 5 ciclos.

PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

# Abordagem multiciclo



PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software



# Exemplo de sinais de controle de uma instrução para abordagem multiciclo

- Busca da instrução:
  - LerMem = 1, IREsc = 1, IouD = 0, UALFonteA = 0, UALFonteB = 01, UALOp = 00, PCEsc = 1
- Decodificação e busca do registrador:
  - UALFonteA = 0, UALFonteB = 11, UALOp = 00
- Execução, cálculo do endereço de memória ou efetivação do desvio condicional:
  - Instruções de referência à memória: UALFonteA = 1, UALFonteB = 10, UALOp = 00
  - Instruções do tipo R: UALFonteA = 1, UALFonteB = 00, UALOp = 10
  - Desvio condicional: UALFonteA = 1, UALFonteB = 00, UALOp = 01, PCEscCond = 1, FontePC = 01
  - Desvio incondicional: FontePC = 11, PCEsc = 1
- Escrita em memória ou registrador:
  - Instruções de referência à memória: LerMem = 1 ou EscMem = 1, se lw, IouD = 1
  - Instruções do tipo R: RegDst = 1, EscReg = 1, MemParaReg = 0
- Leitura de memória e escrita em registrador:
  - MemParaReg = 1, EscReg = 1, RegDst = 0

PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

# Questão simples

- Quantos ciclos leva para executar esse código?

lw \$t2, 0(\$t3)

lw \$t3, 4(\$t3)

beq \$t2, \$t3, Label #assuma não

add \$t5, \$t2, \$t3

sw \$t5, 8(\$t3)

Label: ...

- O que está acontecendo durante o oitavo ciclo de execução?
- Em que ciclo acontece realmente a adição de \$t2 e \$t3?

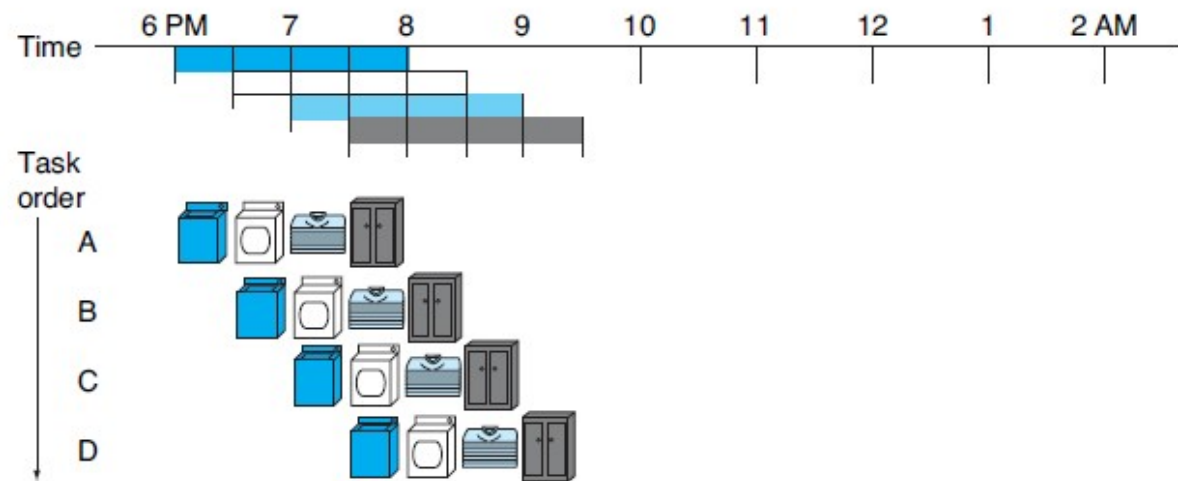
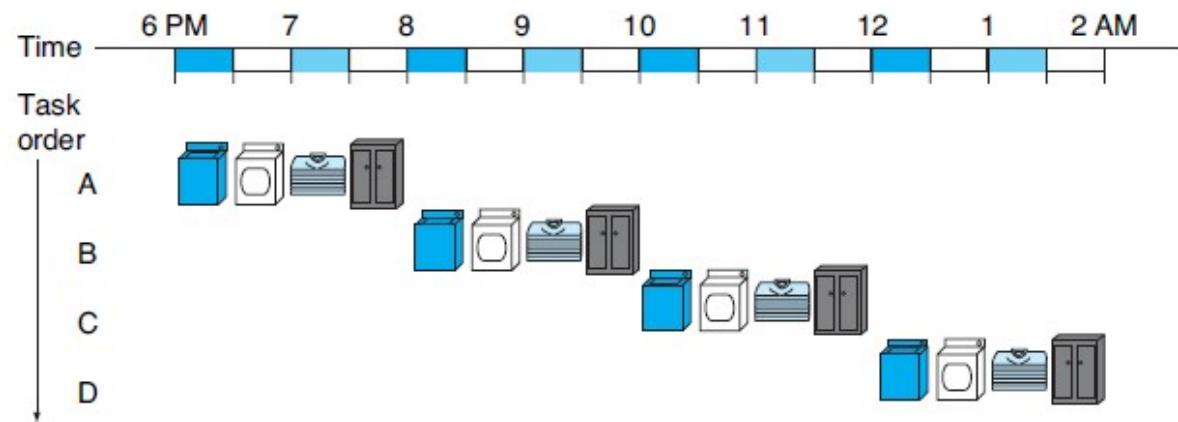
# Pipeline

MIPS

# Pipeline -primeira parte

- 1. Introdução
- 2. Pipelines aritméticos
- 3. Pipelines de instruções
- 4. Desempenho
- 5. Conflitos de memória
- 6. Dependências em desvios

# Introdução



PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

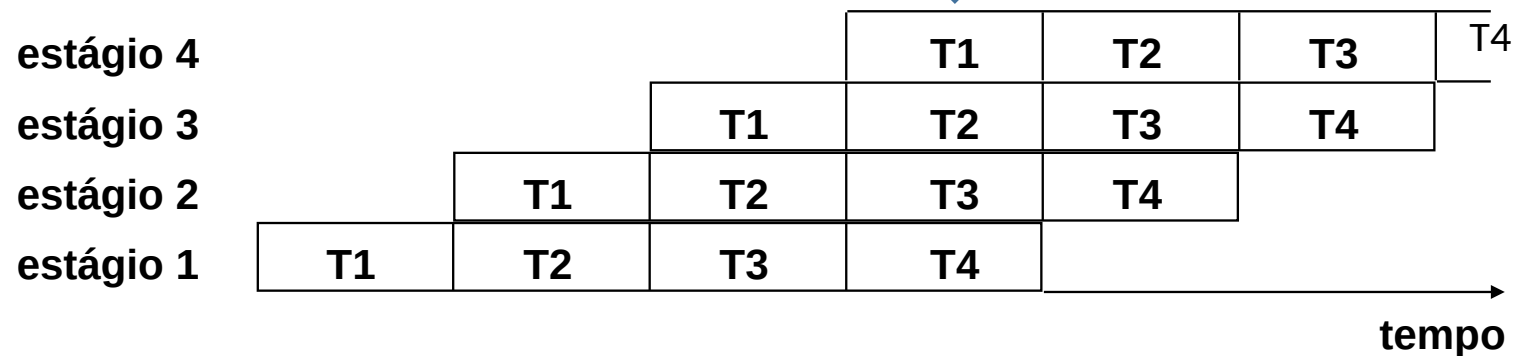
# Introdução

¬ Objetivo: aumento de desempenho

¬ divisão de uma tarefa em N estágios

¬ N tarefas executadas em paralelo, uma em cada estágio

¬ Diagrama espaço - tempo



# Introdução

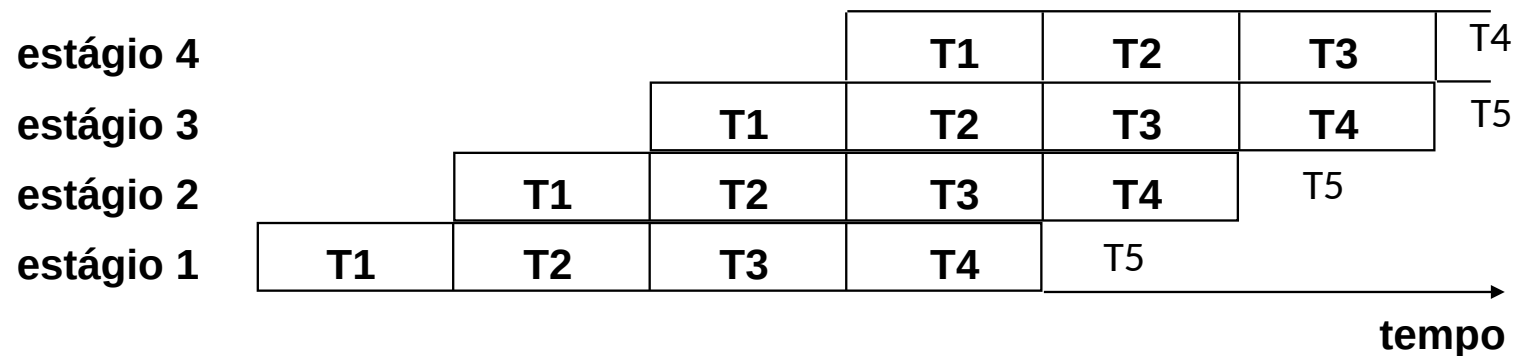
ADD R1, R2, R3  
SUB R4, R5, R1

¬ Objetivo: aumento de desempenho

¬ divisão de uma tarefa em N estágios

¬ N tarefas executadas em paralelo, uma em cada estágio

¬ Diagrama espaço - tempo



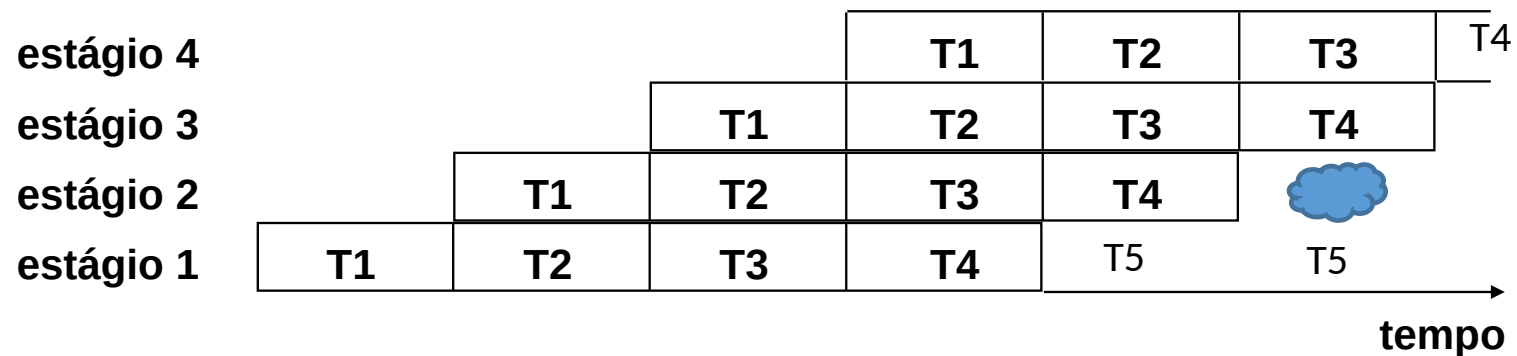
# Introdução

¬ Objetivo: aumento de desempenho

¬ divisão de uma tarefa em N estágios

¬ N tarefas executadas em paralelo, uma em cada estágio

¬ Diagrama espaço - tempo





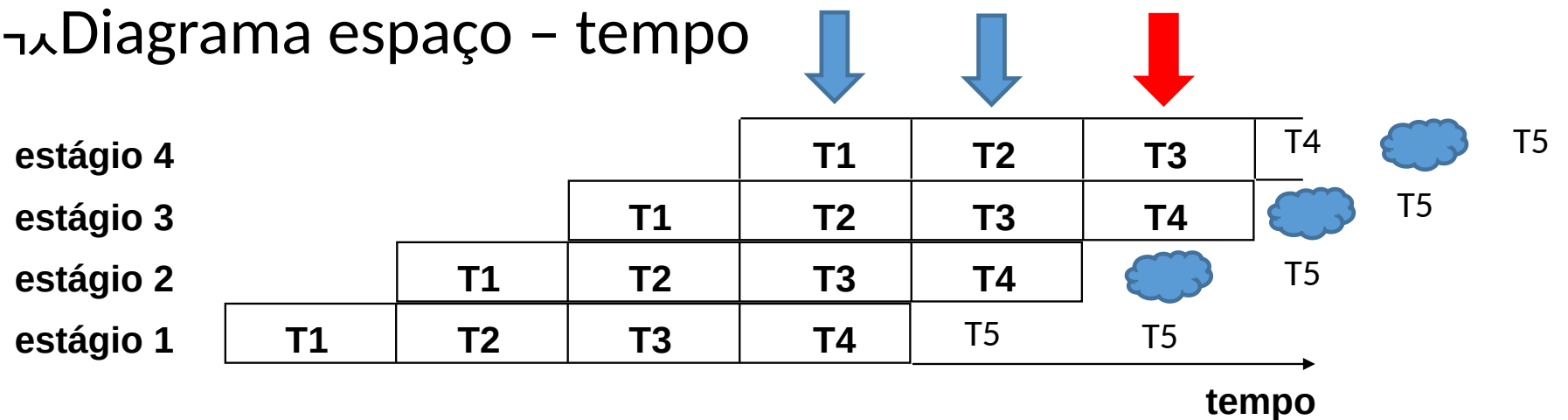
# Introdução

¬ Objetivo: aumento de desempenho

↪ divisão de uma tarefa em N estágios

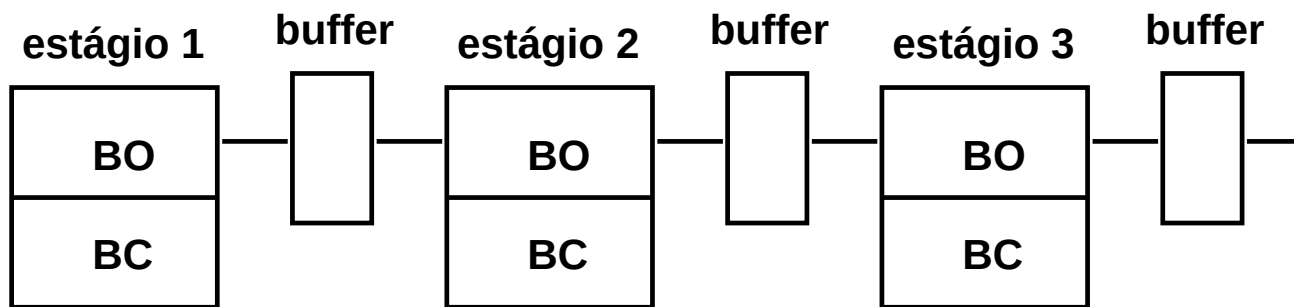
- N tarefas executadas em paralelo, uma em cada estágio

# Diagrama espaço - tempo



# Introdução

- bloco operacional e bloco de controle independentes para cada estágio
- necessidade de buffers entre os estágios



- pipelines de instruções
- pipelines aritméticos

# Pipelines Aritméticos

- exemplo: soma em ponto flutuante executada em 4 estágios
  - 1. comparar expoentes
  - 2. acertar expoentes dos operandos
  - 3. somar
  - 4. normalizar resultado
- exemplo

0.157	$\times 10^6$	-	0.842	$\times 10^5$
0.157	$\times 10^6$	-	0.0842	$\times 10^6$
0.0628	$\times 10^6$			
0.628	$\times 10^5$			

# Pipelines de Instruções

- 2 estágios
  - fetch / decodificação, execução
- 3 estágios
  - fetch, decodificação / busca de operandos, execução
- 4 estágios
  - fetch, decodificação / busca de operandos, execução, store
- 5 estágios
  - fetch, decodificação / cálculo de endereço de operandos, busca de operandos, execução, store
- 6 estágios
  - fetch, decodificação, cálculo de endereço de operandos, busca de operandos, execução, store
  - estágio só para decodificação é bom em processadores CISC

# Desempenho

- existe um tempo inicial até que o pipeline “encha”
- cada tarefa leva o mesmo tempo, com ou sem pipeline
- média de tempo por tarefa é no entanto dividida por N

**s** tarefas

**N** estágios

**primeira tarefa: N ciclos de relógio**

**s – 1 tarefas seguintes: s – 1 ciclos de relógio**

**Tempo total com pipeline = N + ( s – 1 )**

**Tempo total sem pipeline = s N**

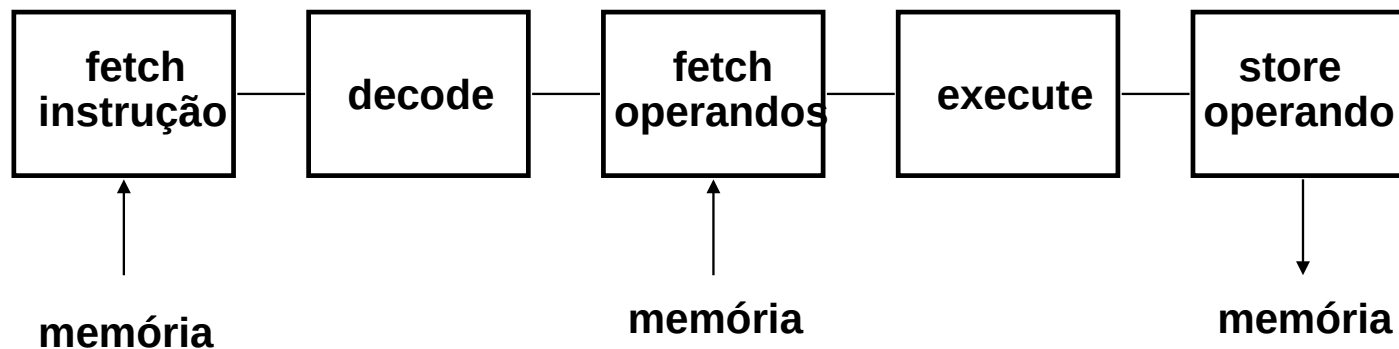
$$\text{Speed-up teórico} = \frac{s N}{N + (s - 1)}$$

# Problemas no desempenho

- como dividir todas as instruções num mesmo conjunto de estágios?
- como obter estágios com tempos de execução similares?
- conflitos de memória
  - acessos simultâneos à memória por 2 ou mais estágios
- dependências de dados
  - instruções dependem de resultados de instruções anteriores, ainda não completadas
- instruções de desvio
  - instrução seguinte não está no endereço seguinte ao da instrução anterior

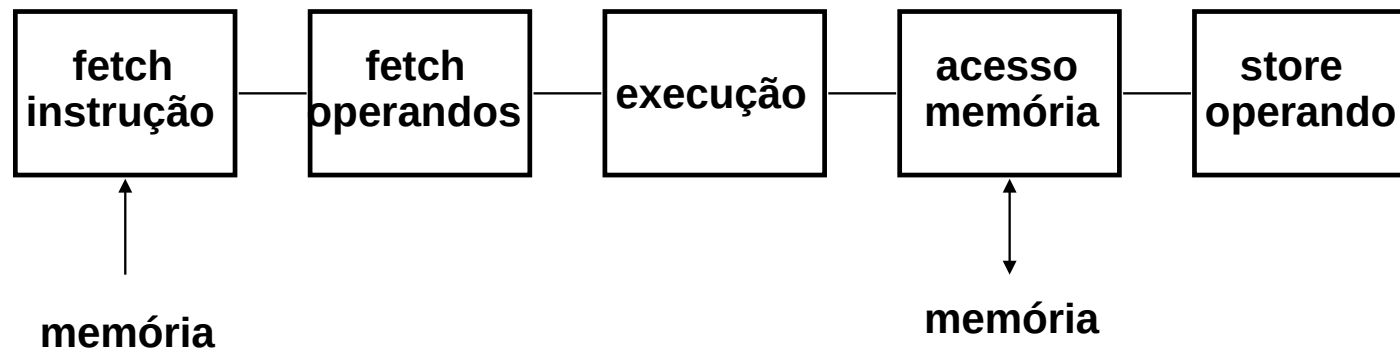
# Conflitos de memória

- problema: acessos simultâneos à memória por 2 ou mais estágios



# Processadores RISC

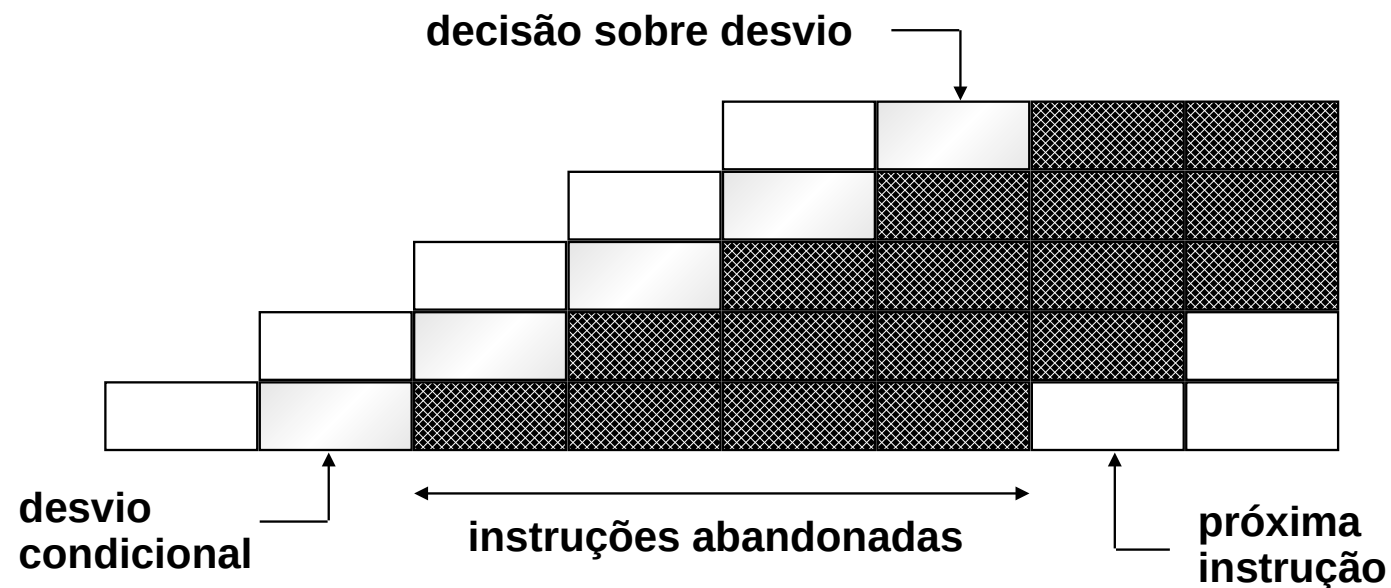
- memória é acessada apenas por instruções LOAD e STORE
- apenas um estágio do pipeline faz acesso a operandos de memória
  - apenas 1 instrução pode estar executando acesso a dados a cada instante
- se caches de dados e instruções são separadas, não há nenhum conflito de acesso à memória





# Dependências em desvios

- efeito de desvios condicionais
  - se o desvio ocorre, pipeline precisa ser esvaziado
  - não se sabe se desvio ocorrerá ou não até o momento de sua execução



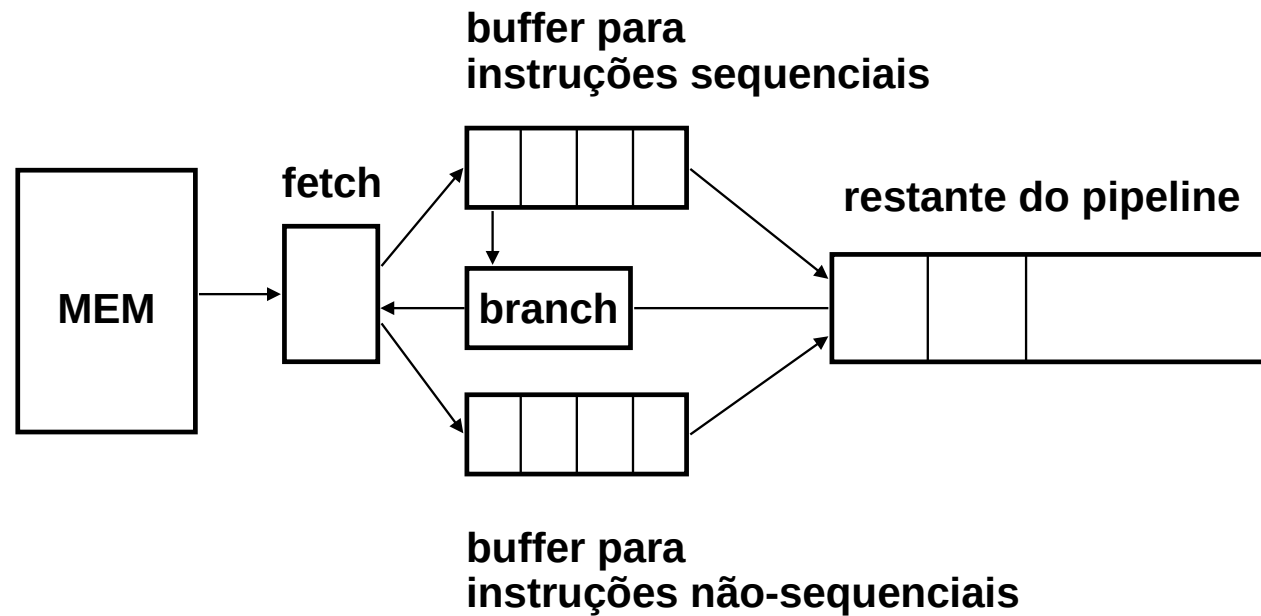
# Dependências de desvios

- instruções abandonadas não podem ter afetado conteúdo de registradores e memórias
  - isto é usualmente automático, porque escrita de valores é sempre feita no último estágio do pipeline
- deve-se procurar antecipar a decisão sobre o desvio para o estágio mais cedo possível
- desvios incondicionais
  - sabe-se que é um desvio desde a decodificação da instrução (segundo estágio do pipeline )
  - é possível evitar abandono de número maior de instruções
  - problema: em que estágio é feito o cálculo do endereço efetivo do desvio?

# Técnicas de tratamento dos desvios condicionais

- 1. Executar os dois caminhos do desvio
  - buffers paralelos de instruções
- 2. Prever o sentido do desvio
  - predição estática
  - predição dinâmica
- 3. Eliminar o problema
  - “delayed branch”

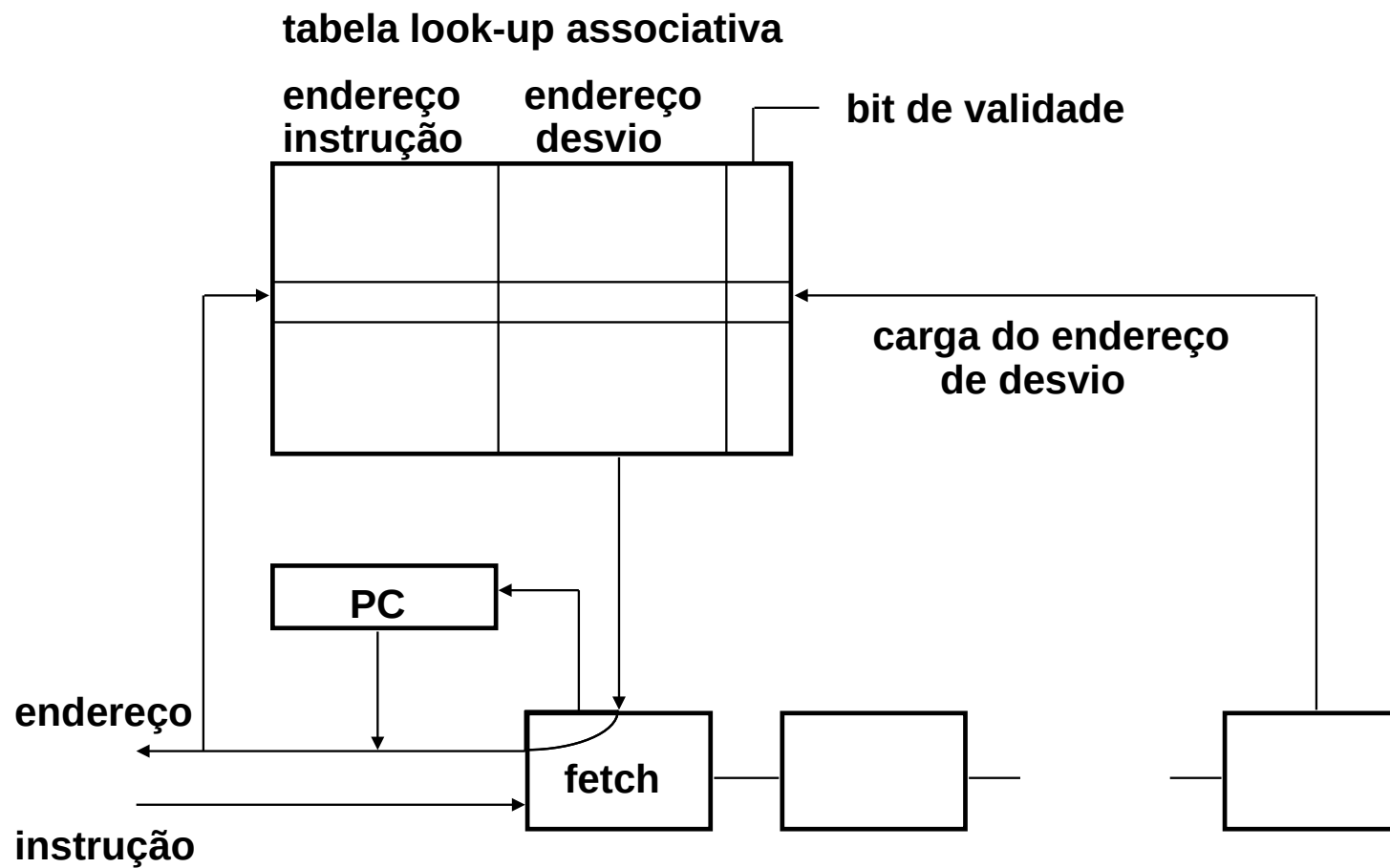
# Buffers paralelos de instruções



# Predição estática

- supor sempre mesma direção para o desvio
  - desvio sempre ocorre
  - desvio nunca ocorre
- compilador define direção mais provável
  - instrução de desvio contém bit de predição, ligado / desligado pelo compilador
  - início de laço ( ou desvio para frente ): desvio improvável
  - final de laço ( ou desvio para trás ): desvio provável
- até 85 % de acerto é possível

# Predição dinâmica



# Predição dinâmica

- tabela look-up associativa armazena triplas
  - endereços das instruções de desvio condicional mais recentemente executadas
  - endereços de destino destes desvios
  - bit de validade, indicando se desvio foi tomado na última execução
- quando instrução de desvio condicional é buscada na memória
  - é feita comparação associativa na tabela, à procura do endereço desta instrução
  - se endereço é encontrado e bit de validade está ligado, o endereço de desvio armazenado na tabela é usado
  - ao final da execução da instrução, endereço efetivo de destino do desvio e bit de validade são atualizados na tabela
- tabela pode utilizar diversos mapeamentos e algoritmos de substituição

# Predição dinâmica

- variação: branch history table
  - contador associado a cada posição da tabela
  - a cada vez que uma instrução de desvio contida na tabela é executada ...
    - contador é incrementado se desvio ocorre
    - contador é decrementado se desvio não ocorre
  - valor do contador é utilizado para a predição



# Delayed Branch

- desvio não ocorre imediatamente, e sim apenas após uma ou mais instruções seguintes
- caso mais simples: pipeline com 2 estágios – fetch + execute
  - desvio é feito depois da instrução seguinte
  - instrução seguinte não pode ser necessária para decisão sobre ocorrência do desvio
  - compilador reorganiza código
  - tipicamente, em 70% dos casos encontra-se instrução para colocar após o desvio
- pipeline com N estágios
  - desvio é feito depois de  $N - 1$  instruções

# Pipeline - segunda parte

- 1. Introdução
- 2. Dependências verdadeiras
- 3. Dependências falsas
- 4. Pipeline interlock
- 5. Forwarding (adiantamento de dados)

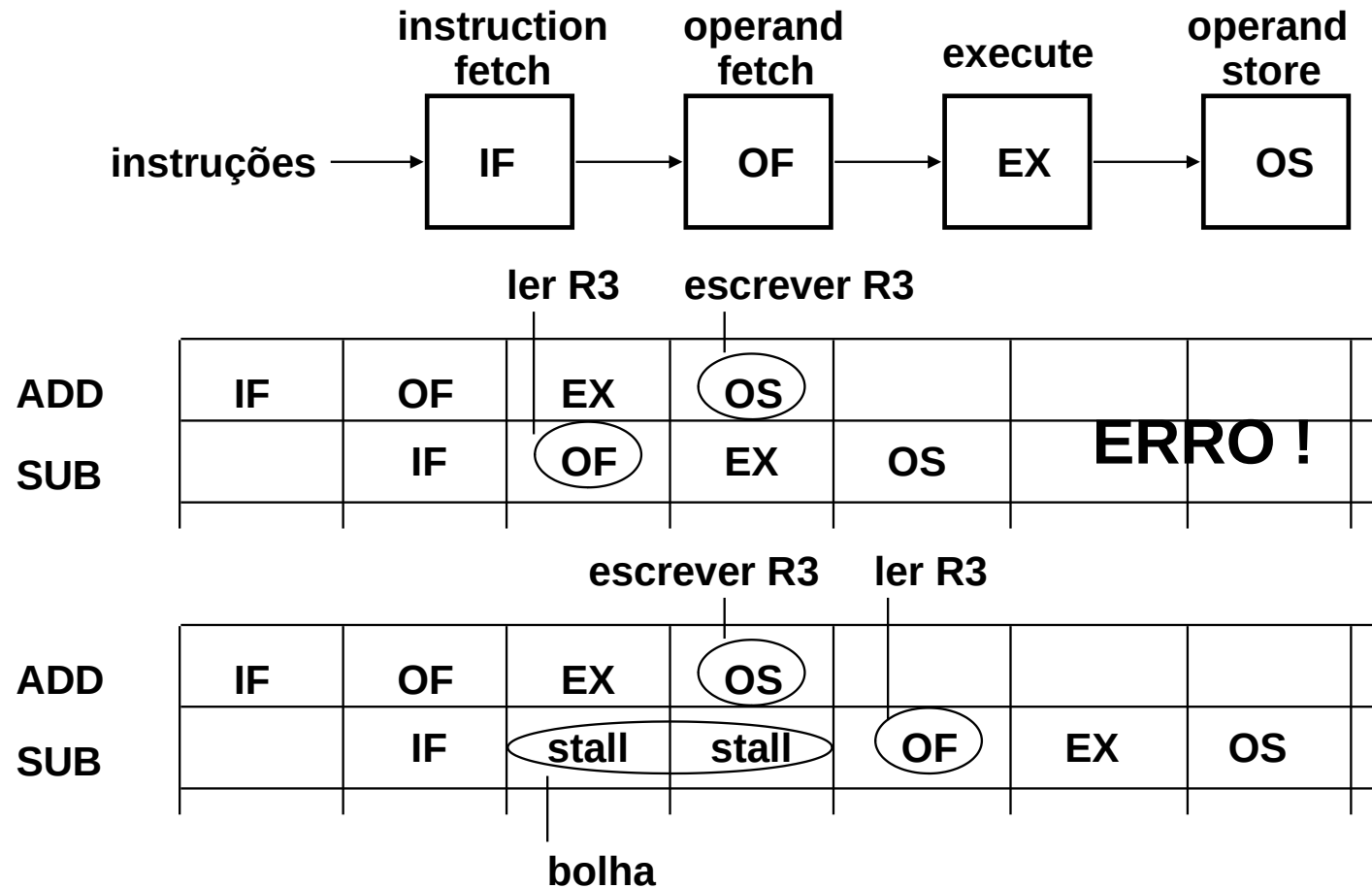
# Introdução

- problema: instruções consecutivas podem fazer acesso aos mesmos operandos
  - execução da instrução seguinte pode depender de operando calculado pela instrução anterior
- caso particular: instrução precisa de resultado anterior (p.ex. registrador) para cálculo de endereço efetivo de operando
- tipos de dependências de dados
  - dependência verdadeira
  - antidependência
  - dependência de saída

# Dependências verdadeiras

- exemplo
  - 1. ADD R3, R2, R1 ;  $R3 = R2 + R1$
  - 2. SUB R4, R3, 1 ;  $R4 = R3 - 1$
- instrução 2 depende de valor de R3 calculado pela instrução 1
- instrução 1 precisa atualizar valor de R3 antes que instrução 2 busque os seus operandos
- read-after-write hazard
- pipeline precisa ser parado durante certo número de ciclos

# Dependências verdadeiras



# Dependências falsas

- Antidependência
- exemplo
  - 1. ADD R3, R2, R1 ;  $R3 = R2 + R1$
  - 2. SUB R2, R4, 1 ;  $R2 = R4 - 1$
- instrução 1 utiliza operando em R2 que é escrito pela instrução 2
- instrução 2 não pode salvar resultado em R2 antes que instrução 1 tenha lido seus operandos
- write-after-read hazard
- não é um problema em pipelines onde a ordem de execução das instruções é mantida
  - escrita do resultado é sempre o último estágio
- problema em processadores superescalares

# Dependências falsas

- Dependência de saída
- exemplo
  - 1. ADD R3, R2, R1 ;  $R3 = R2 + R1$
  - 2. SUB R2, R3, 1 ;  $R2 = R3 - 1$
  - 3. ADD R3, R2, R5 ;  $R3 = R2 + R5$
- instruções 1 e 3 escrevem no mesmo operando em R3
- instrução 1 tem que escrever seu resultado em R3 antes do que a instrução 3, senão valor final de R3 ficará errado
- write-after-write hazard
- também só é problema em processadores superescalares

# Pipeline interlock

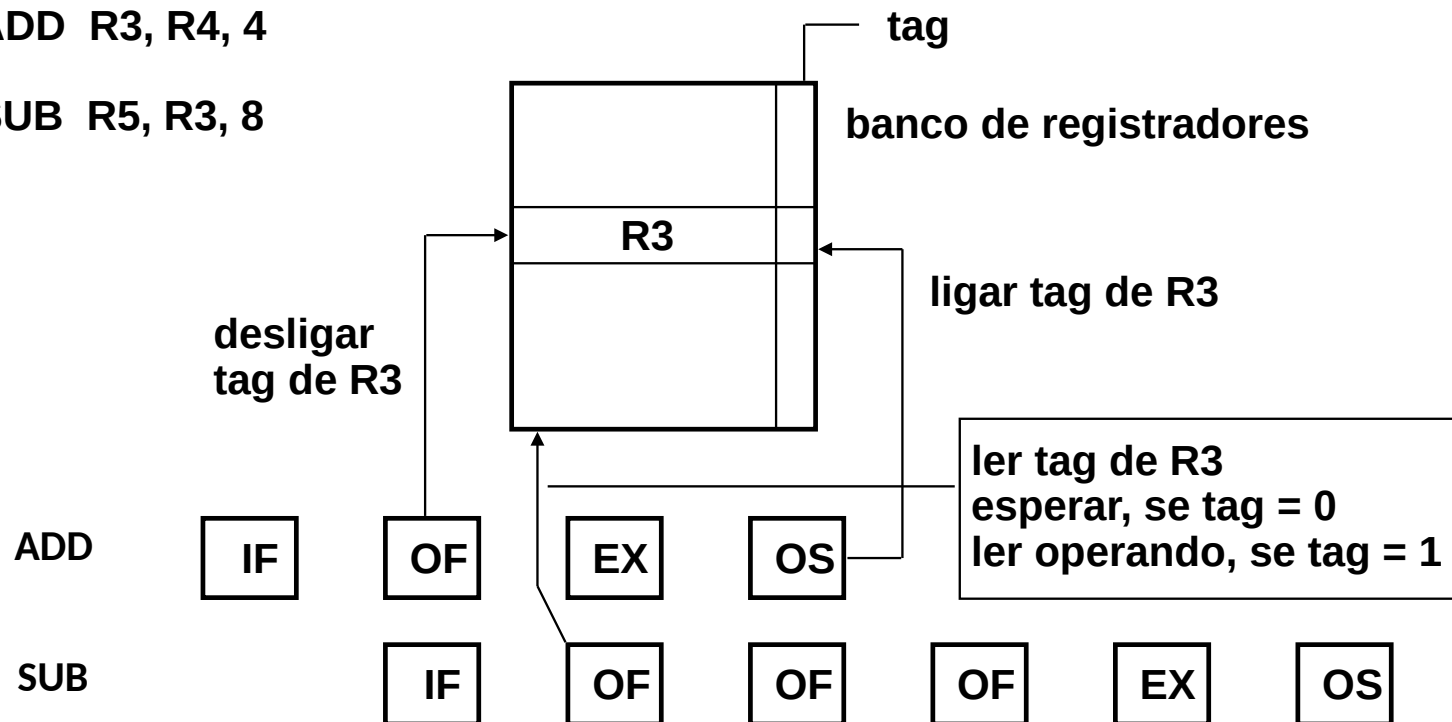
- método para manter seqüência correta de leituras e escritas em registradores
- tag de 1 bit é associado a cada registrador
  - tag = 0 indica valor não válido, = 1 indica valor válido
- se instrução que é buscada e decodificada escreve num registrador, o tag do mesmo é zerado
- tag é ligado quando instrução escreve o valor no registrador
- outras instruções que sejam executadas enquanto tag está zerado devem esperar tag = 1 para ler valor deste registrador



# Pipeline interlock

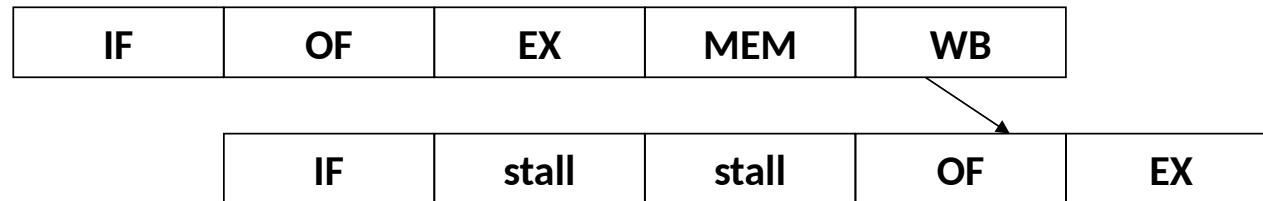
ADD R3, R4, 4

SUB R5, R3, 8



# Forwarding

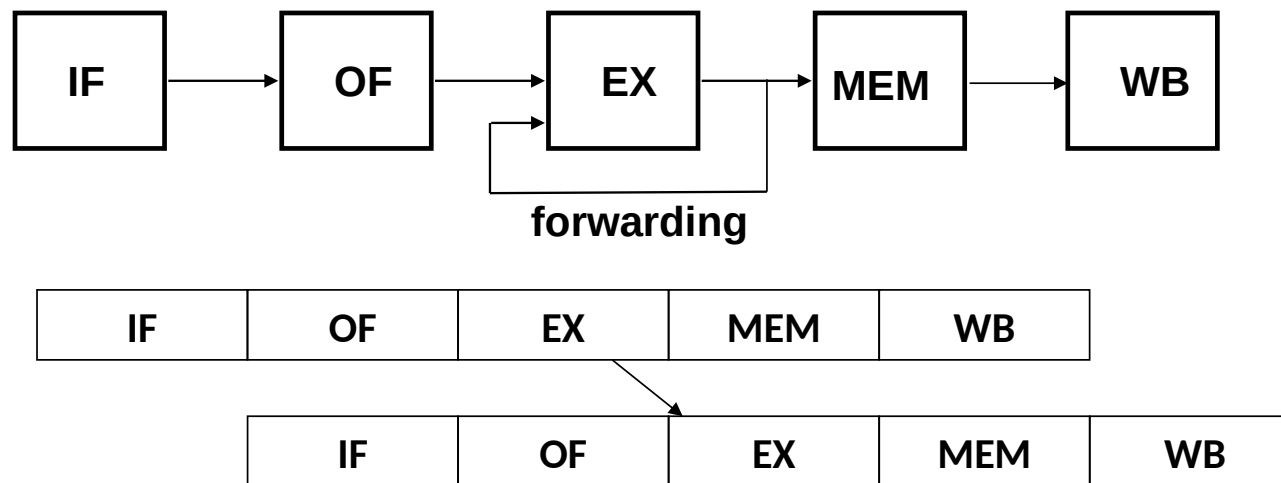
- exemplo
  - ADD R3, R2, R0
  - SUB R4, R3, 8
- instrução SUB precisa do valor de R3, calculado pela instrução ADD
  - valor é escrito em R3 por ADD no último estágio WB (write-back)
  - valor é necessário em SUB no terceiro estágio
  - instrução SUB ficará presa por 2 ciclos no 2º estágio do pipeline



*supõe-se escrita no banco de registradores na primeira metade do ciclo e leitura na segunda metade*

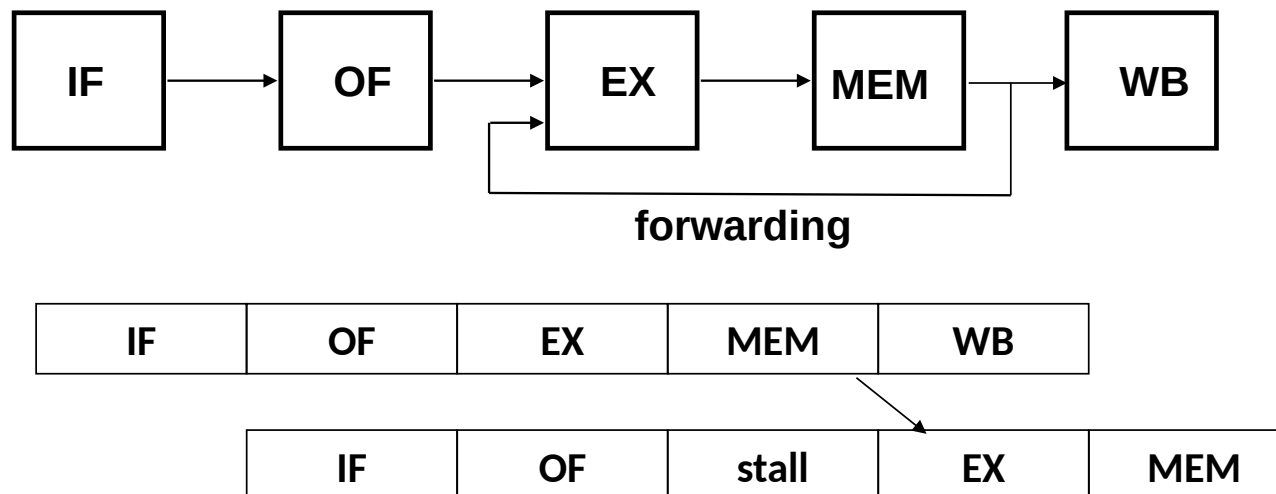
# Forwarding

- caminho interno dentro do pipeline entre a saída da ALU e a entrada da ALU
  - evita stall do pipeline

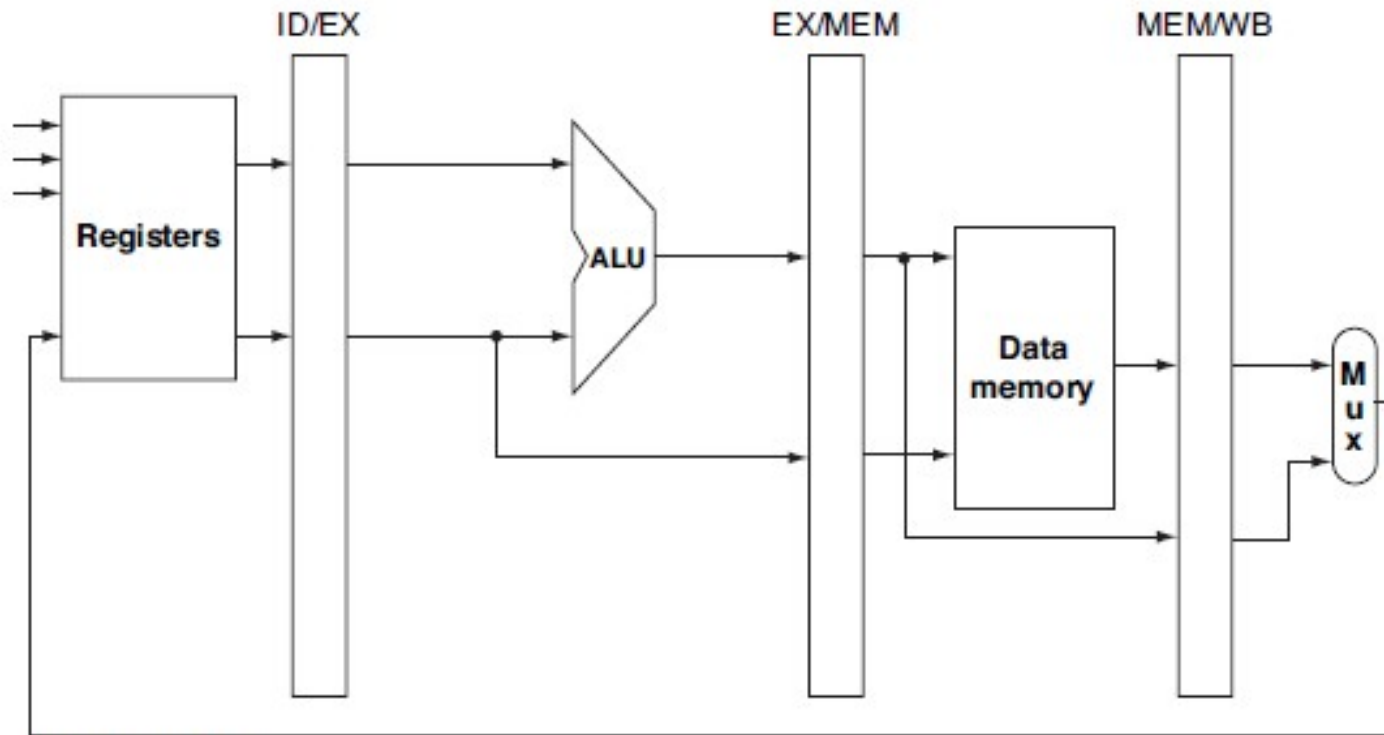


# Forwarding

- exemplo 2
  - LOAD R3, 100 (R0)
  - ADD R1, R2, R3
- forwarding: caminho interno dentro do pipeline entre a saída da memória de dados e a entrada da ALU

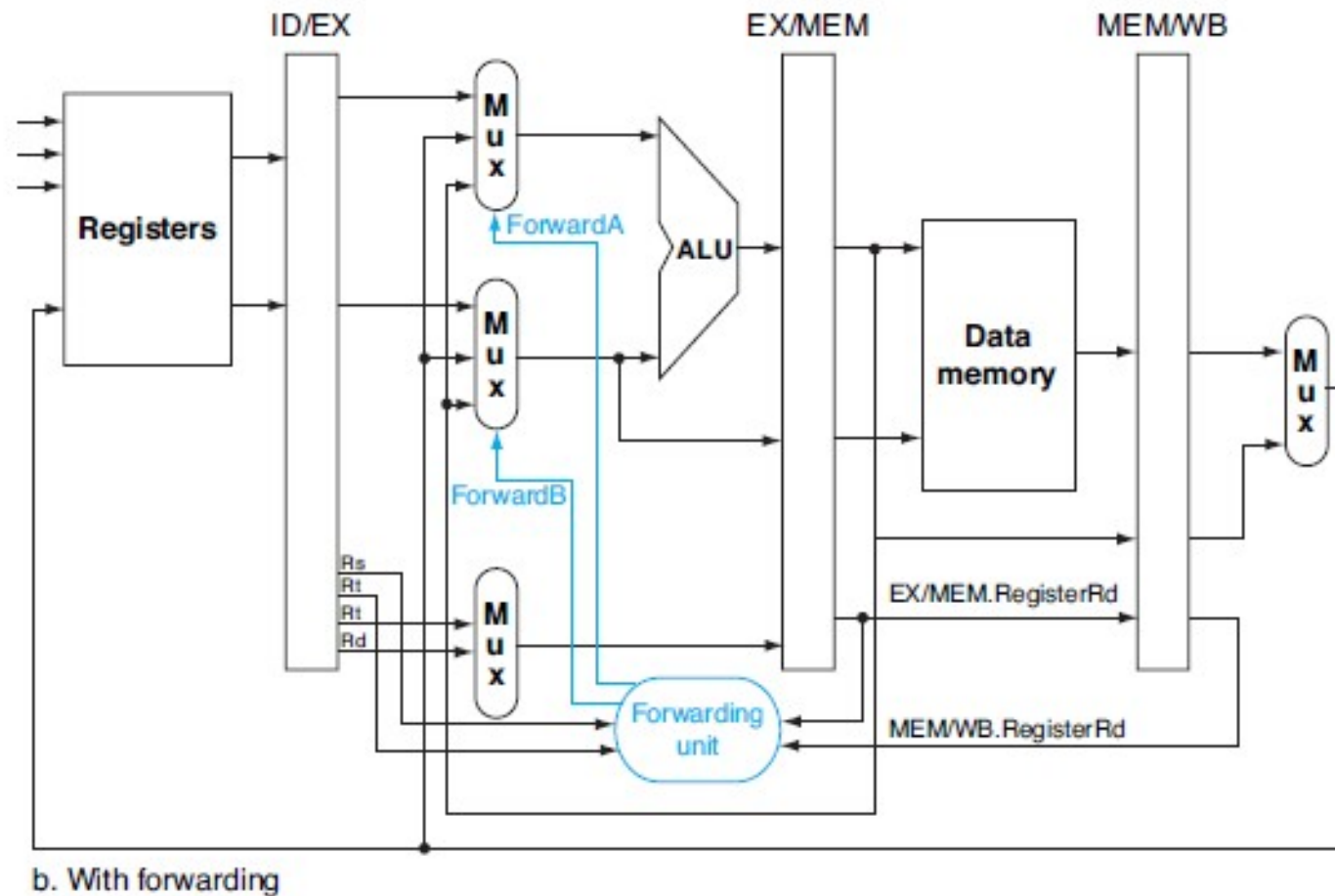


# Como fazer o adiantamento de dados?



a. No forwarding

# Como fazer o adiamento de dados?



b. With forwarding

PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

# Pipeline -terceira parte

- 1. Pipeline do MIPS
- 2. Dependências no Pipeline
- 3. Forwarding
- 4. Exemplos
- 5. Instruções de Desvio no Pipeline

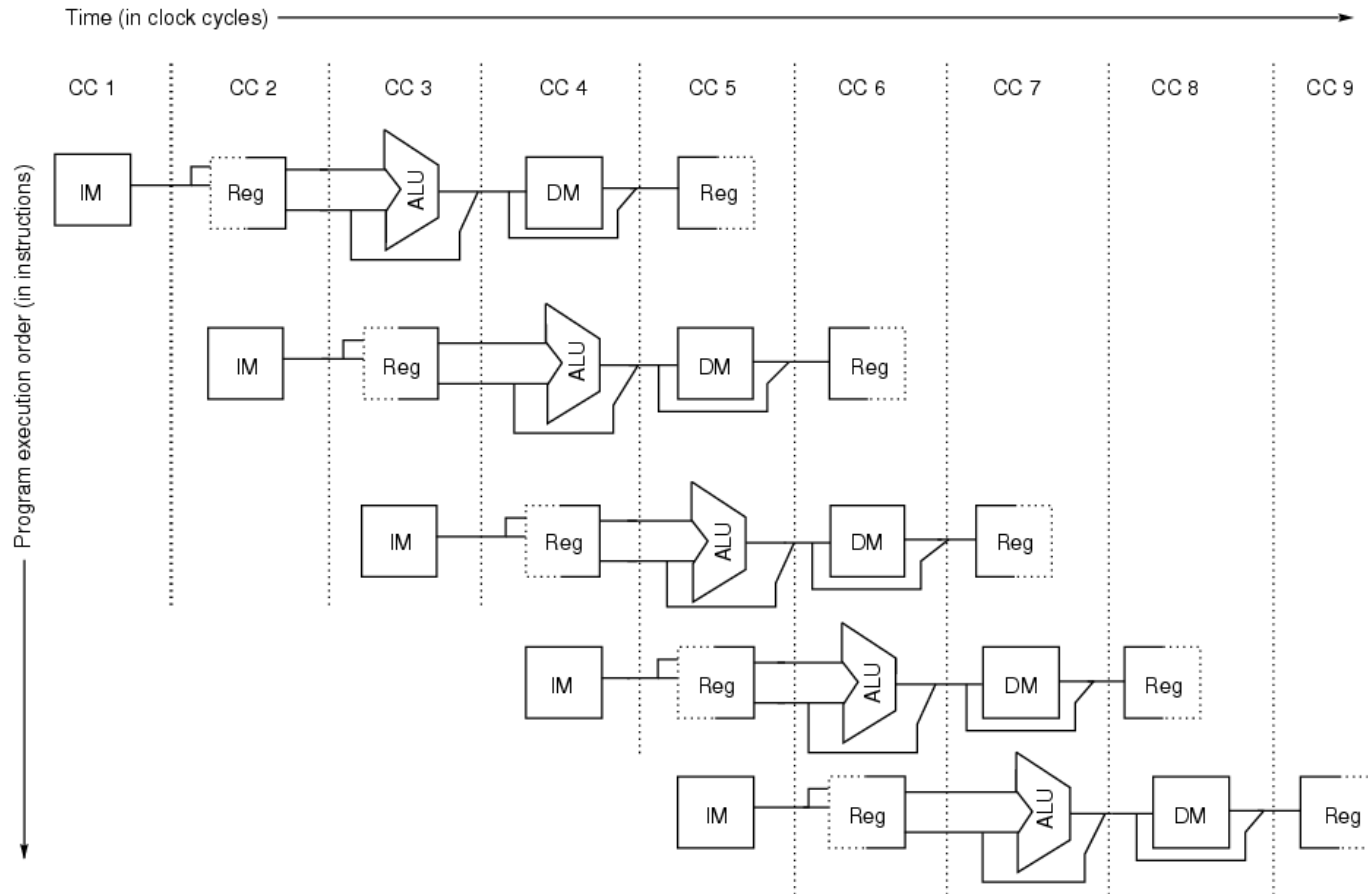
# Pipeline do MIPS

- IF – Instruction Fetch
- ID – Instruction Decode
- EX – Execution
- MEM – Memory Access
- WB – Write Back

Instrução	Clock								
	1	2	3	4	5	6	7	8	9
<b>i</b>	IF	ID	EX	MEM	WB				
<b>i + 1</b>		IF	ID	EX	MEM	WB			
<b>i + 2</b>			IF	ID	EX	MEM	WB		
<b>i + 3</b>				IF	ID	EX	MEM	WB	
<b>i + 4</b>					IF	ID	EX	MEM	WB

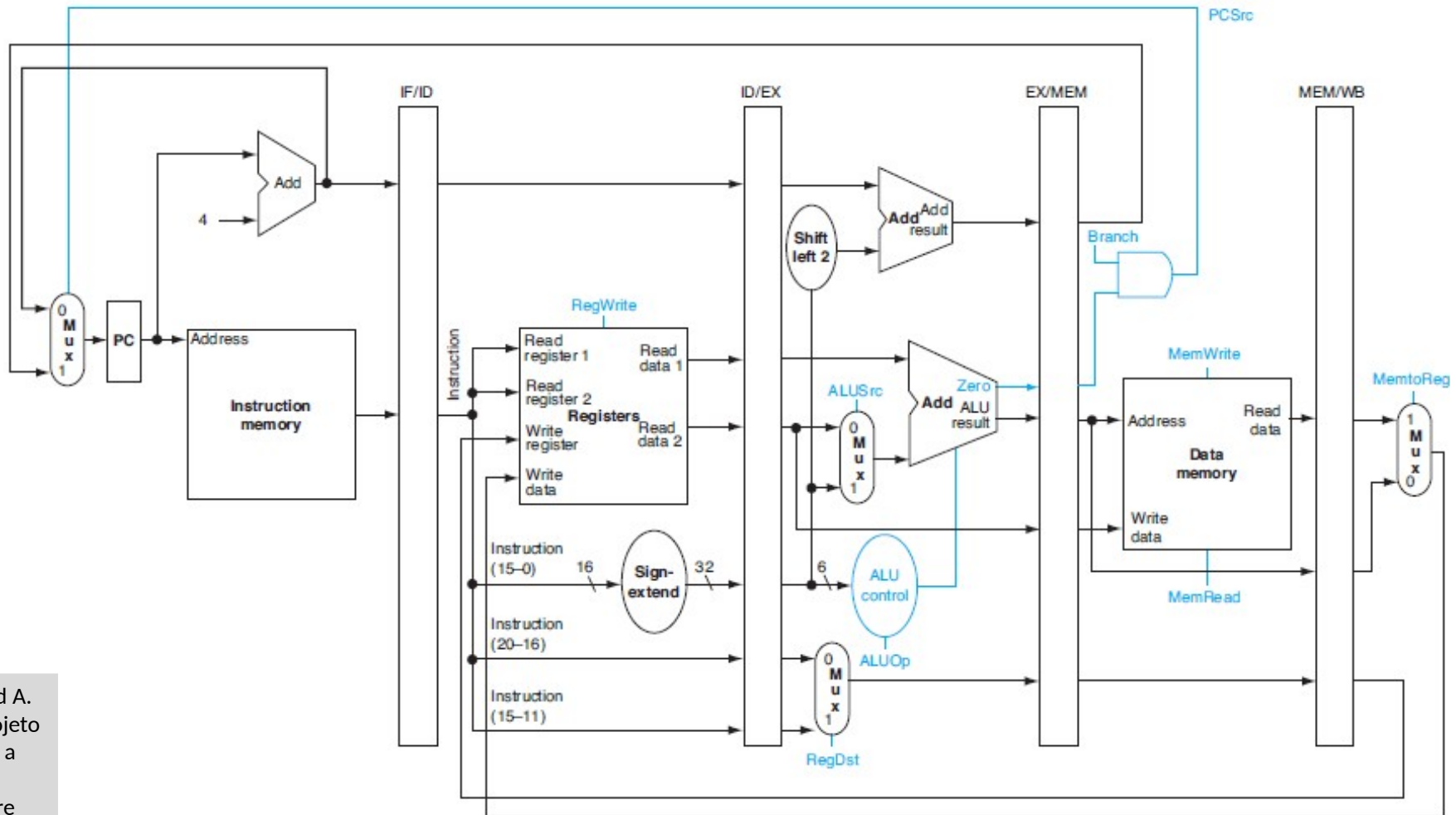


# Pipeline do MIPS



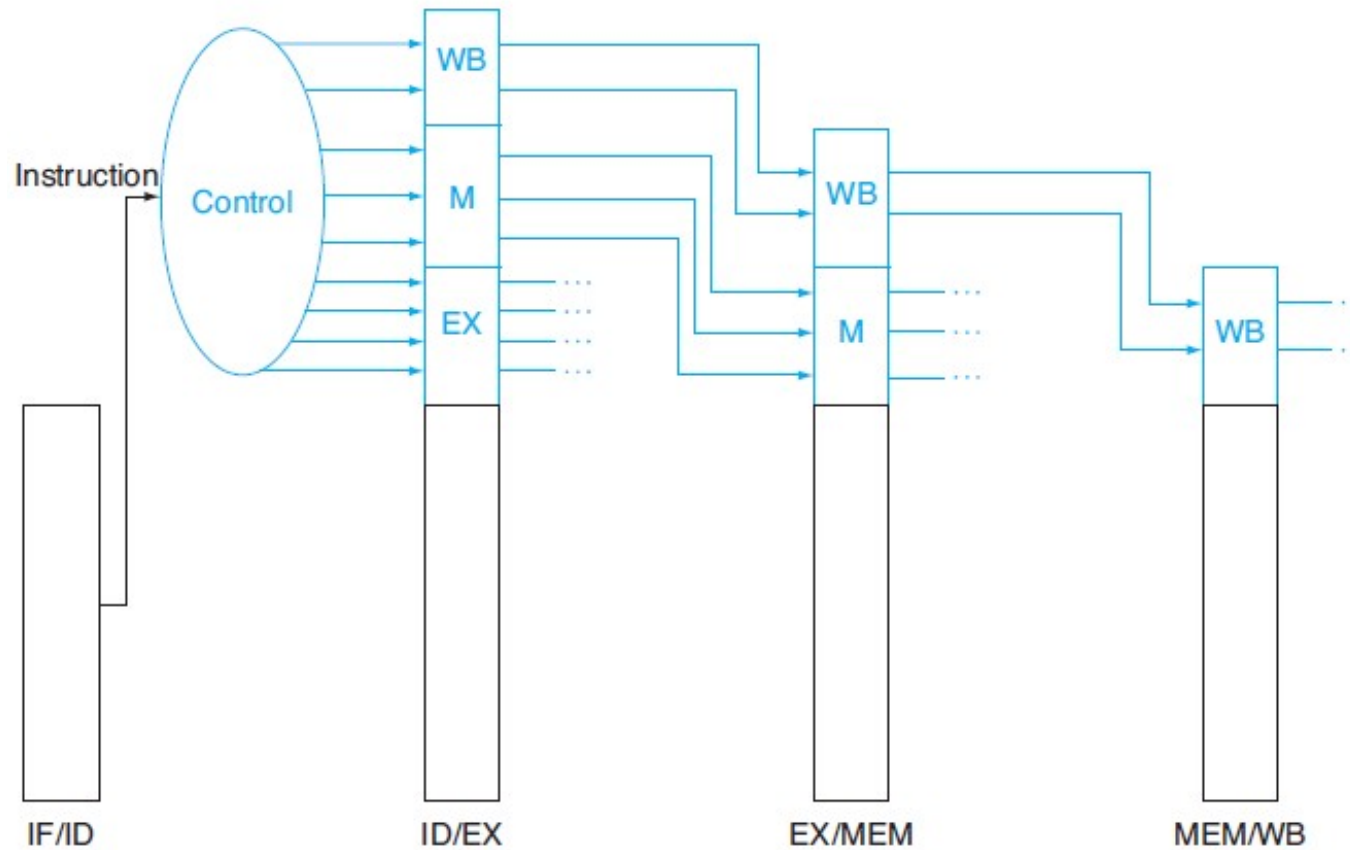
PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

# Pipeline do MIPS



PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

# Pipeline do MIPS



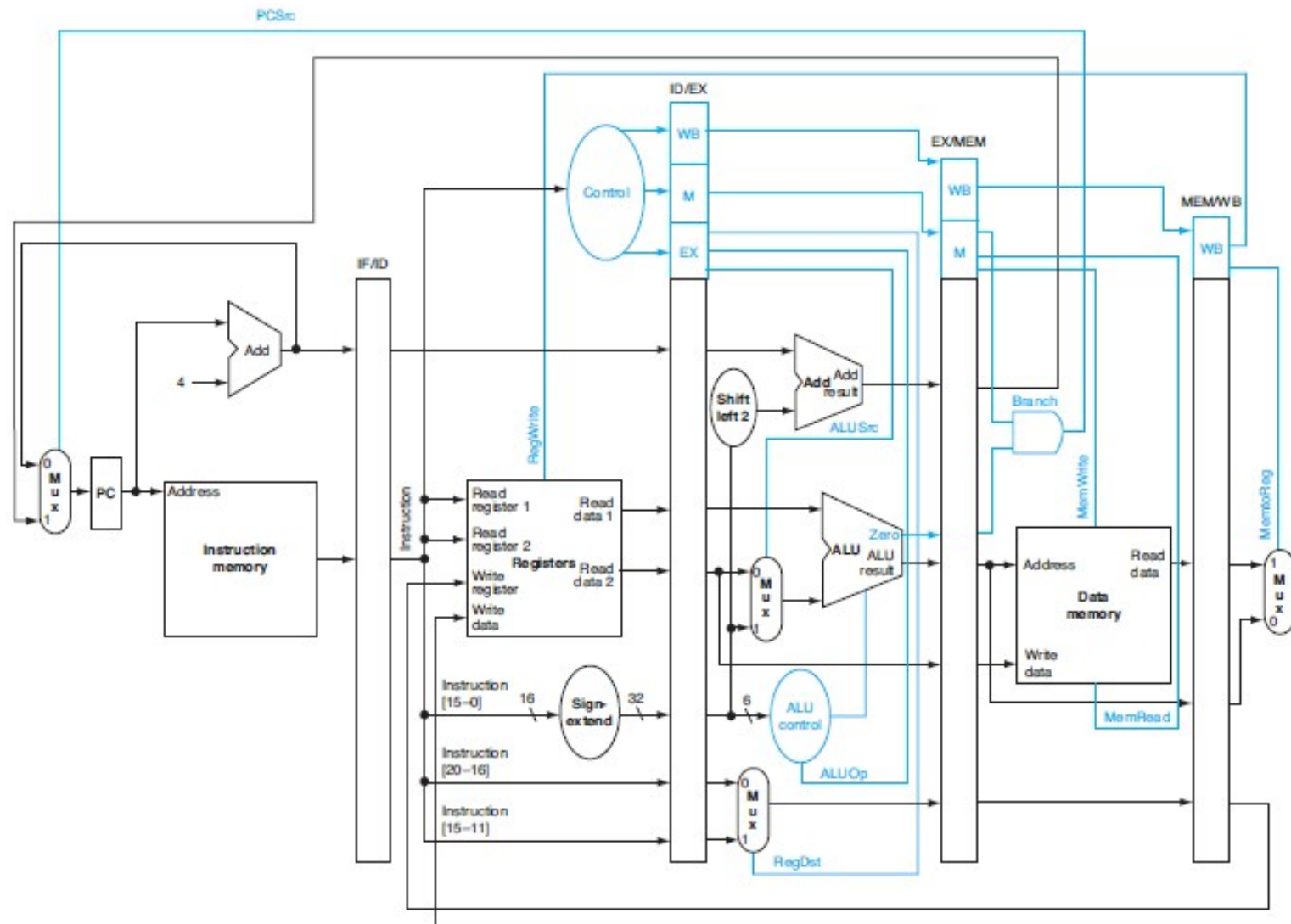
PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

9/15/22

Arquitetura de Computadores III - Engenharia e Ciência da Computação - PUC Minas

51

# Pipeline do MIPS

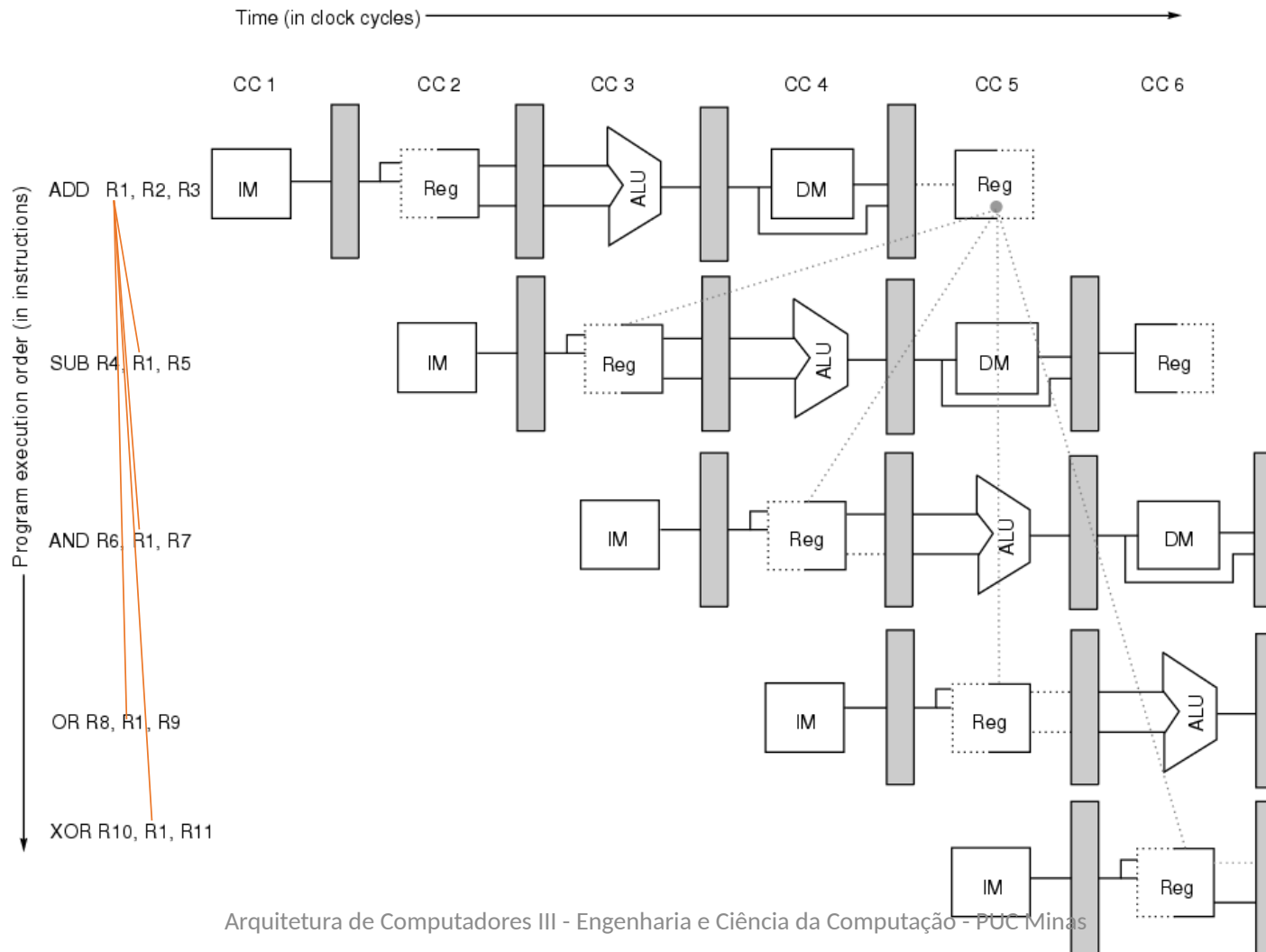


PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

9/15/22

52

# Dependências no Pipeline



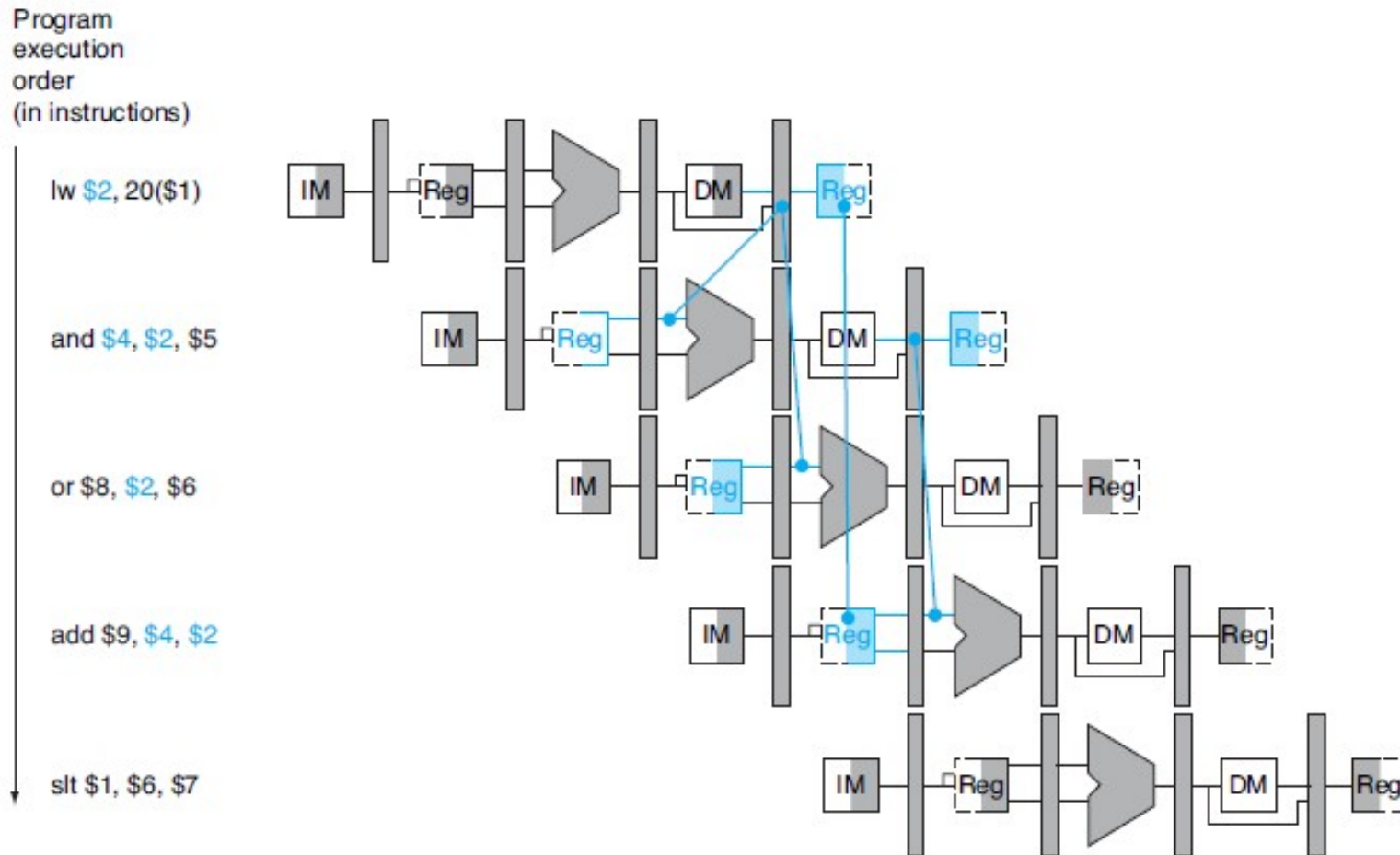
PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

9/15/22

Arquitetura de Computadores III - Engenharia e Ciência da Computação - PUC Minas

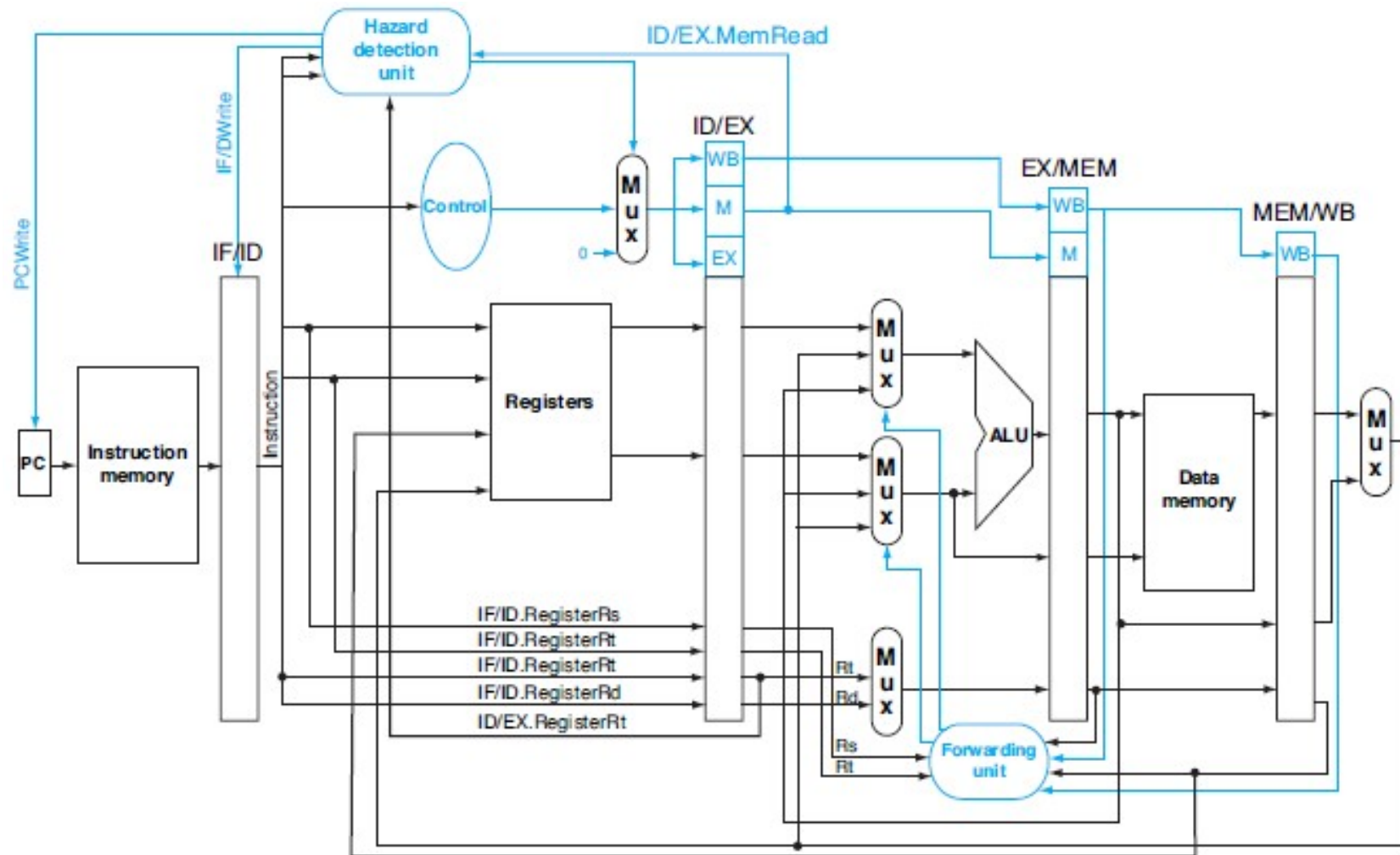
53

# Forwarding nem sempre elimina todas as bolhas



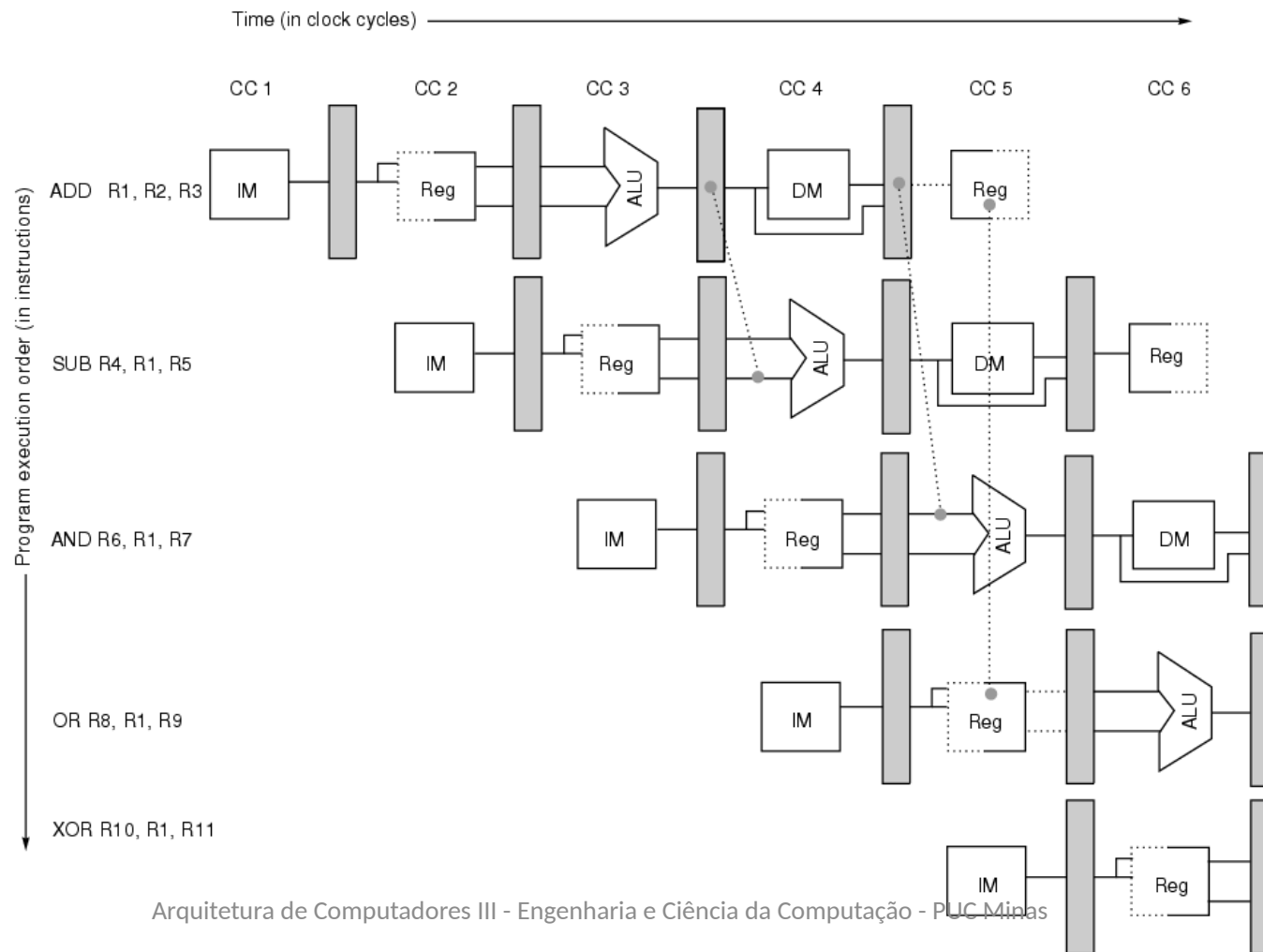
PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

# Como gerar bolha?



PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

# Forwarding



PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

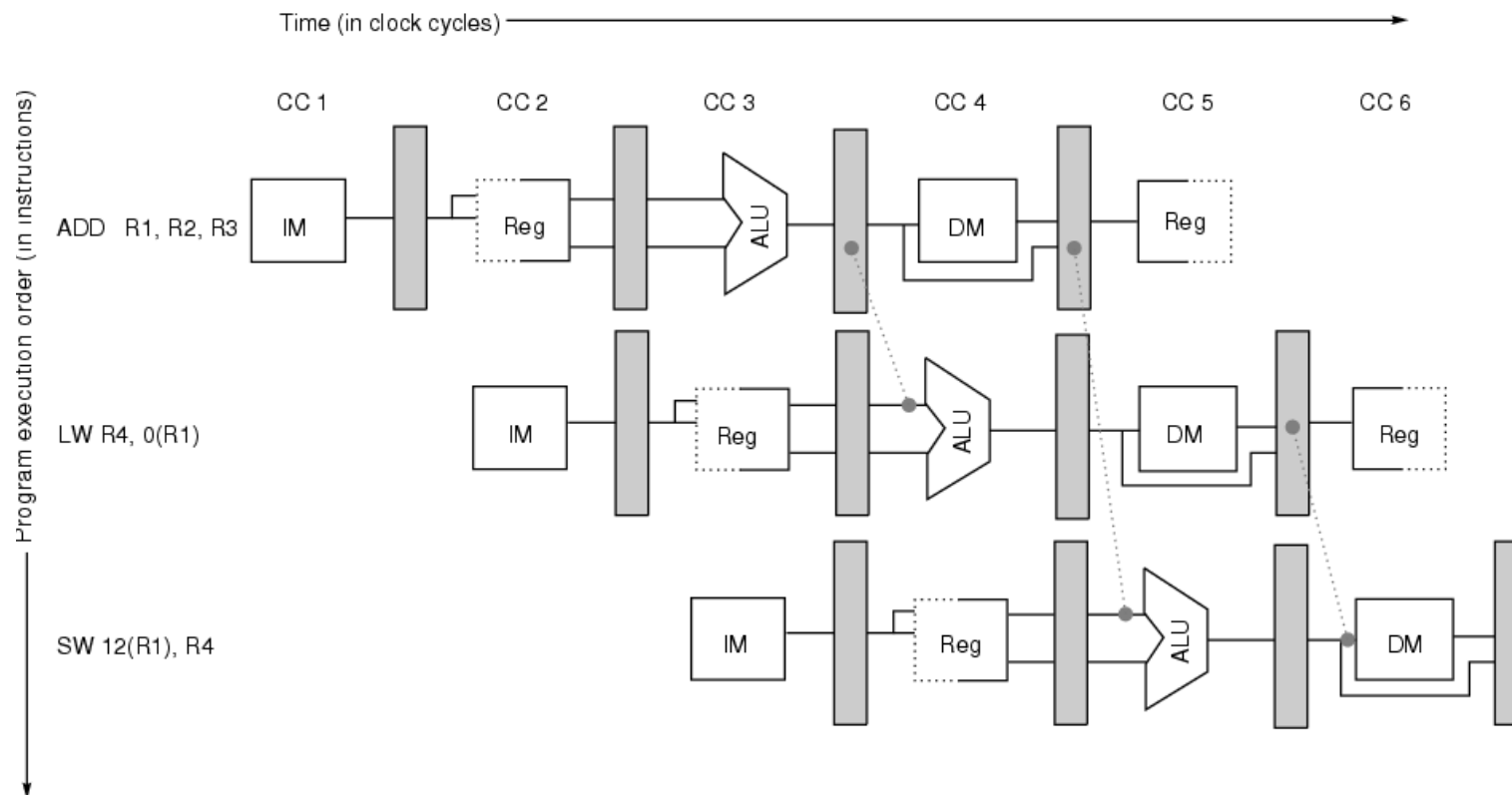
9/15/22

Arquitetura de Computadores III - Engenharia e Ciência da Computação - PUC Minas

56

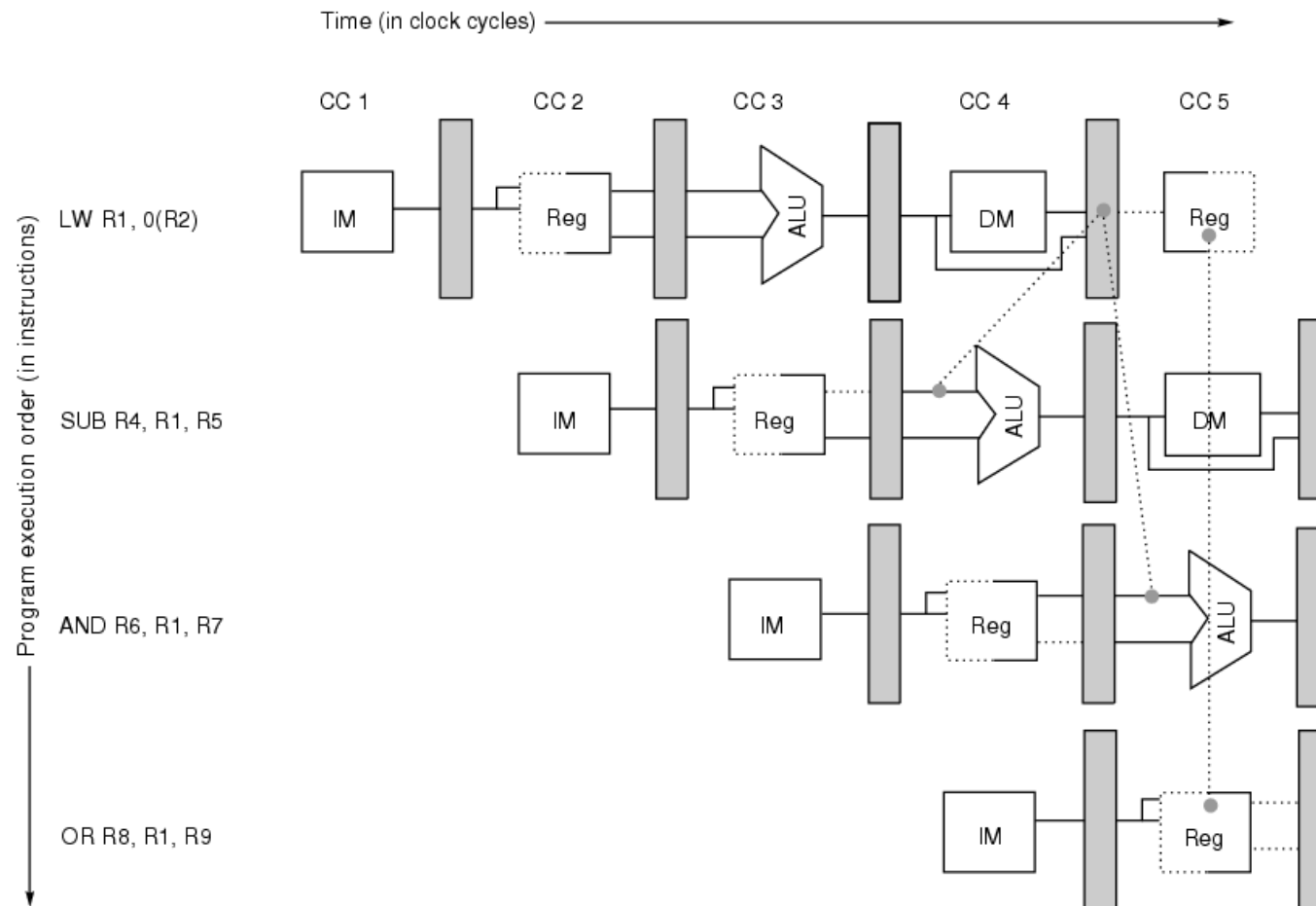


# Forwarding



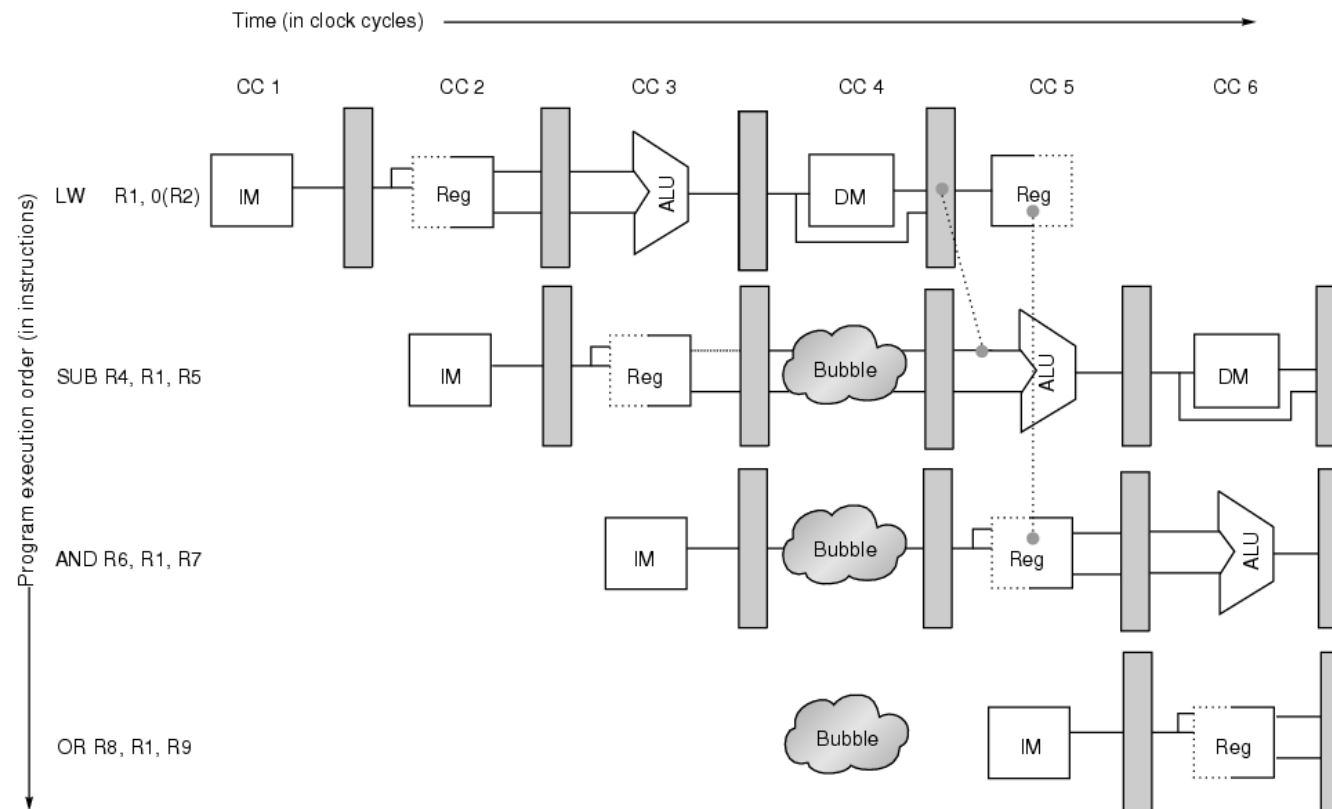
PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

# Forwarding



PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

# Forwarding



PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

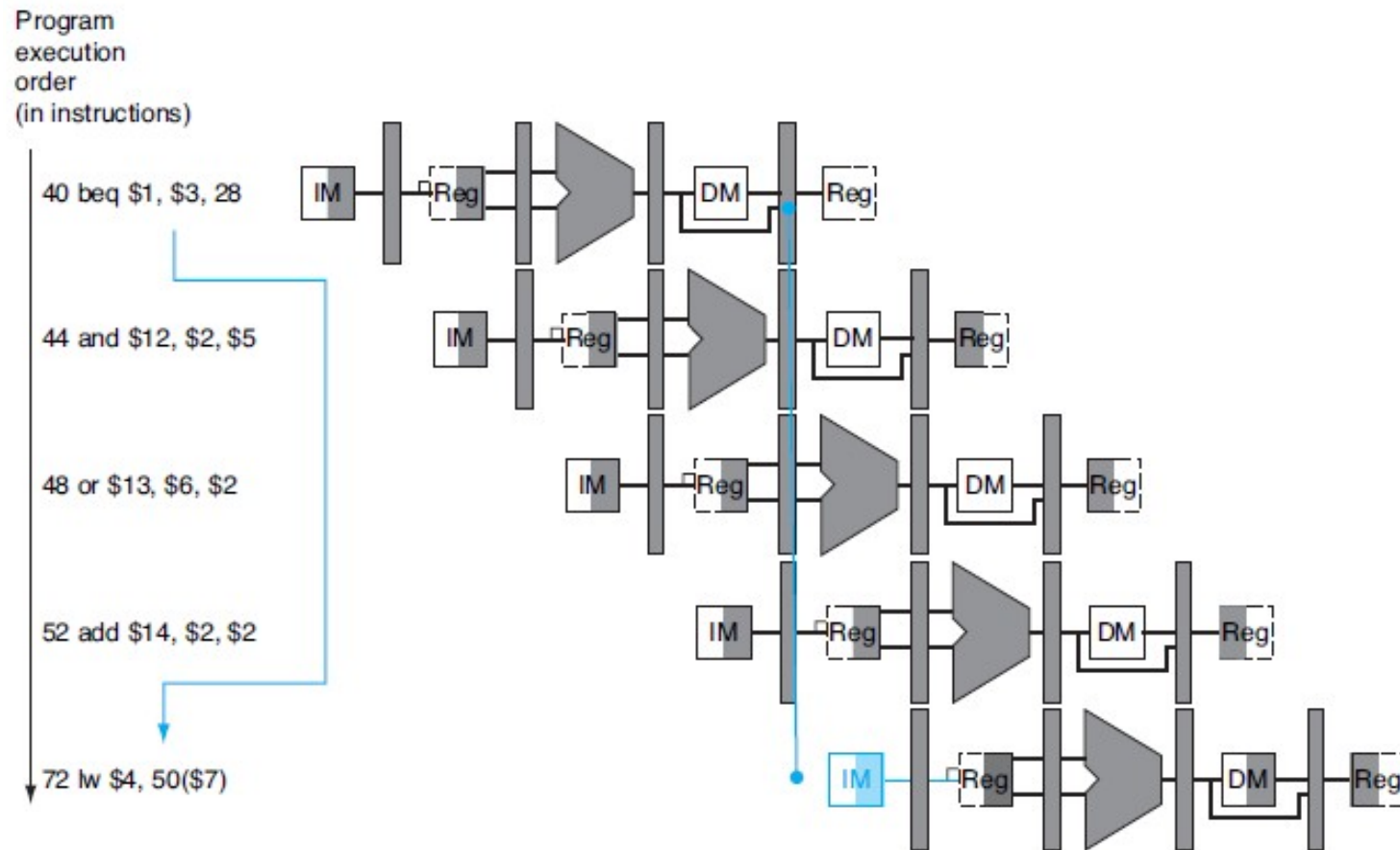
# Instruções de Desvio no Pipeline

Desvio	IF	ID	EX	MEM	WB					
IS		IF	<i>stall</i>	<i>stall</i>	IF	ID	EX	MEM	WB	
IS + 1						IF	ID	EX	MEM	WB
IS + 2							IF	ID	EX	MEM
IS + 3								IF	ID	EX
IS + 4									IF	ID
IS + 5										IF

IS – Instrução Sucessora.

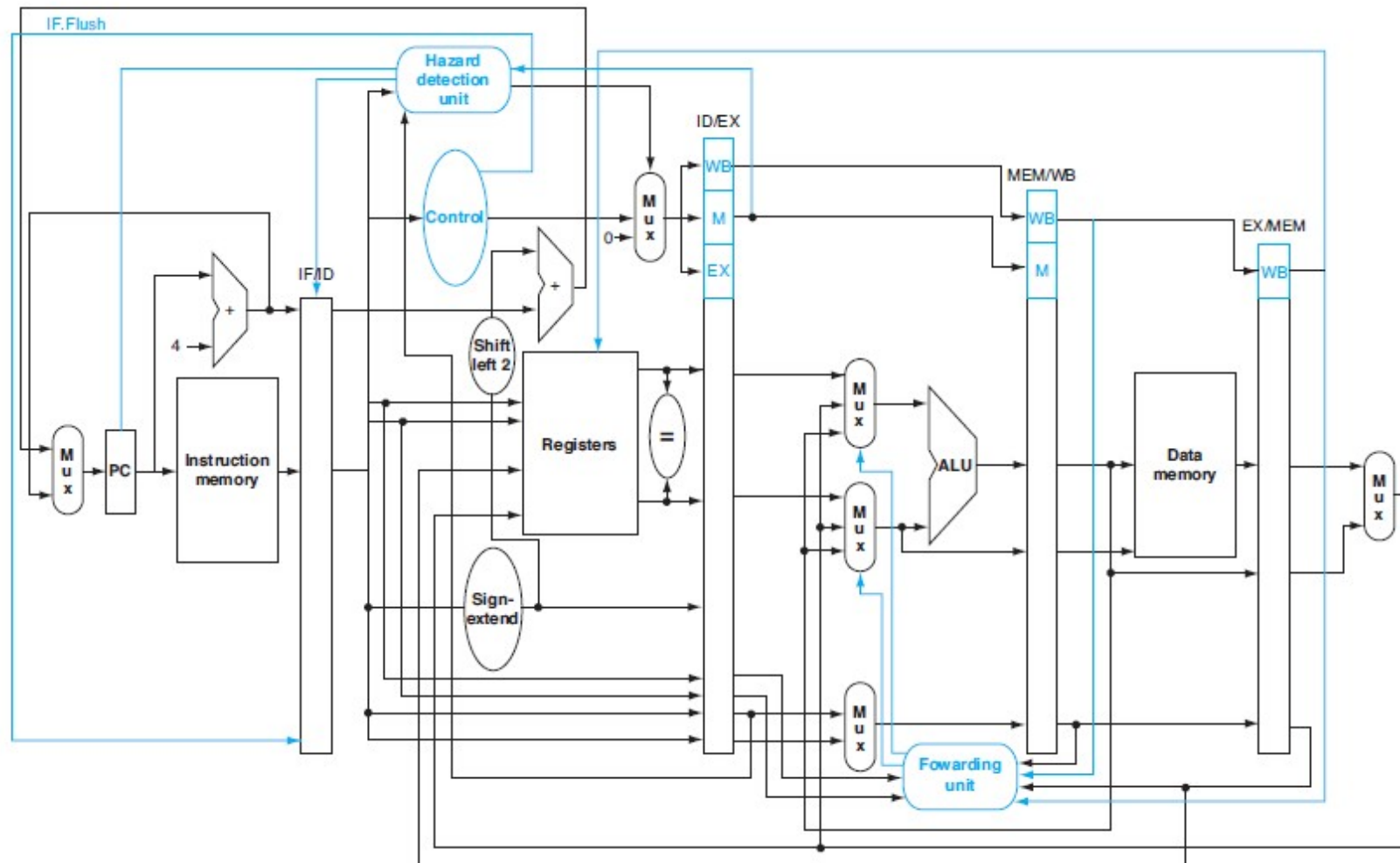
- ⌞ Uma instrução de desvio causa um atraso de três ciclos no pipeline:
  - ⌞ Um ciclo por repetir o estágio de busca (IF);
  - ⌞ Dois ciclos ociosos (stall).

# Desvio e descarte de instruções



PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

# Descartando instruções



PATTERSON, David A.  
Organização e projeto  
de computadores a  
interface  
hardware/software

# Exercício 1

- Explicar o que acontece na execução das instruções abaixo em ordem de entrada no pipeline do processador. Calcule quantidade de ciclos e CPI.

BNE = tomada de decisão no 3º estágio. Há adiantamento de dados. Escrita e leitura no mesmo ciclo.

Loop:   add \$t1, \$s3, \$s3

          add \$t1, \$t1, \$t1

          add \$t1, \$t1, \$s6

          lw \$t0, 0(\$t1)

          bne \$t0, \$s5, Exit # considere existência do desvio

          add \$s3, \$s3, \$s4

          j Loop # jump - instrução de desvio incondicional

Exit:    sub \$s1, \$s5, \$s4

Ciclos   Ciclos bolha  
5

# Exercício 1

- Explicar o que acontece na execução das instruções abaixo em ordem de entrada no pipeline do processador. Calcule quantidade de ciclos e CPI.

BNE = tomada de decisão no 3º estágio. Há adiantamento de dados. Escrita e leitura no mesmo ciclo.

Loop:	add \$t1, \$s3, \$s3		
	add \$t1, \$t1, \$t1	Ciclos	Ciclos bolha
	add \$t1, \$t1, \$s6	5	0
	lw \$t0, 0(\$t1)	1	0
	bne \$t0, \$s5, Exit # considere existência do desvio	1	0
	add \$s3, \$s3, \$s4	1	0
	j Loop # jump - instrução de desvio incondicional	1	1
Exit:	sub \$s1, \$s5, \$s4	0	1
		1	0

Ciclos: 13 , Qtd-Inst.: 6, CPI:  $13/6 = 2,17$



# Exercício 2

- Explicar o que acontece na execução das instruções abaixo em ordem de entrada no pipeline do processador. Calcule quantidade de ciclos e CPI. :

BEQ= tomada de decisão no 2º estágio. Não há adiantamento de dados. Escrita e leitura no mesmo ciclo.

Ciclos Ciclos bolha

```
Loop:  sub $s2, $s3, $s4
        lw $s0, 4($s2)
        lw $s1, 0($s3)
        slt $t0, $s0, $s1
        beq $t0, $zero, Loop # considere não há desvio
        sub $t0, $t0, $t0
        beq $t0, $zero, Loop # considere que há desvio
```

# Exercício 2

- Explicar o que acontece na execução das instruções abaixo em ordem de entrada no pipeline do processador. Calcule quantidade de ciclos e CPI. :

BEQ= tomada de decisão no 2º estágio. Não há adiantamento de dados. Escrita e leitura no mesmo ciclo.

	Ciclos	Ciclos bolha
Loop: sub \$s2, \$s3, \$s4	5	0
lw \$s0, 4(\$s2)	1	2
lw \$s1, 0(\$s3)	1	0
slt \$t0, \$s0, \$s1	1	2
beq \$t0, \$zero, Loop # considere <b>não</b> há desvio	1	0
sub \$t0, \$t0, \$t0	1	2
beq \$t0, \$zero, Loop # considere que há desvio		

Ciclos: 19, Qtd-Inst: 7 , CPI:  $19/7 = 2,71$

# Exercício da lista

- 1. lw \$t0, 0(\$t0)
- 2. lw \$t1, 0(\$t0)
- 3. beq \$t0, \$t1, Exit # desvio
- 4. sw \$t1, 0(\$t0)
- 5. lw \$t3, 0(\$t1)
- 6. Exit: sub \$t2, \$t3, \$s4
- 7. slt \$s5, \$t2, \$t3
- 8. sw \$t5, 0(\$t0)
- 9. lw \$t2, 0(\$s2)
- 10. add \$s4, \$s2, \$s1

Ciclos      Ciclos bolha

5      0 / 0

6      1 / 2

1      2 / 2

0      1 / 1

0      0 / 0

7      0 / 0

1      0 / 2

1      0 / 0

1      0 / 0

1      0 / 0

= 2      19/8 = 2,37

- Ciclos: 16, Inst: 8, CPI = 16 / 8