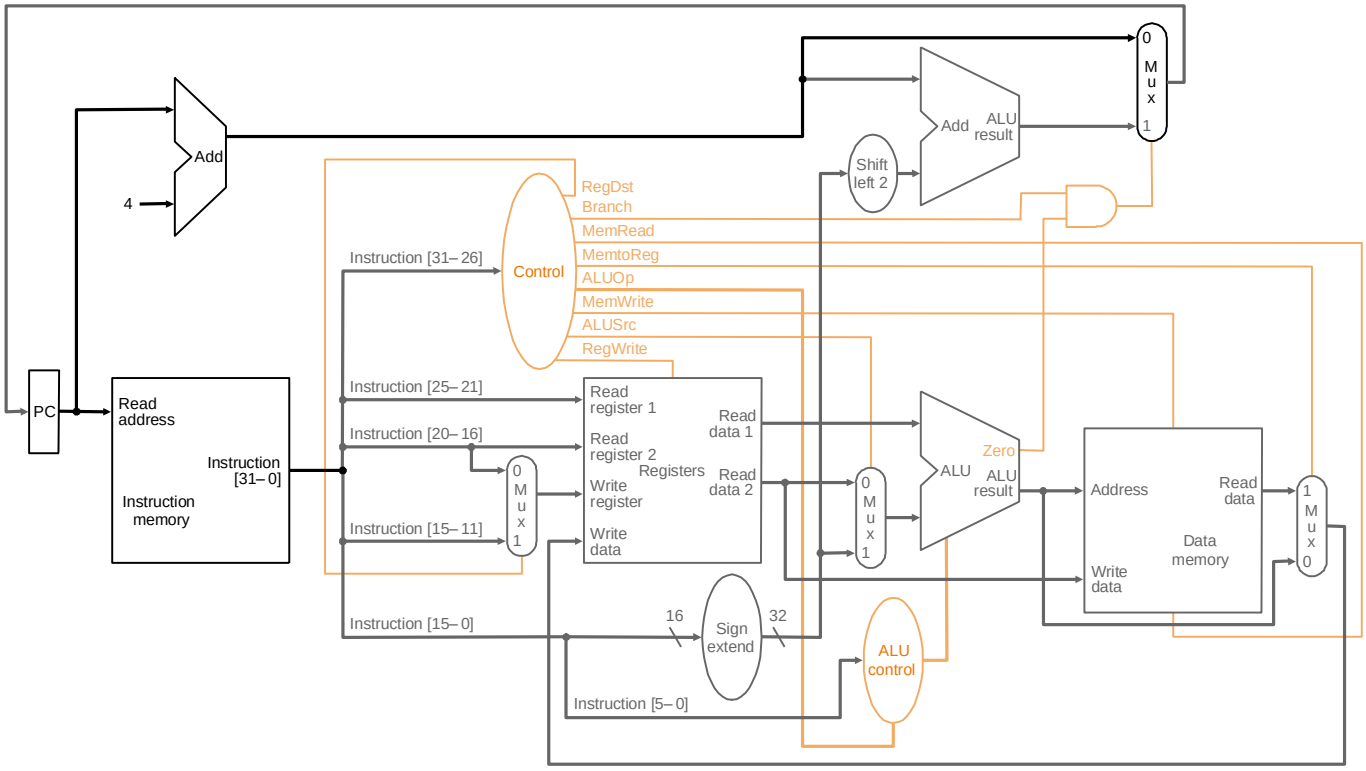
Arquitetura de Computadores III

Parte 3

Pipeline Escalar

O Processador: Caminho de Dados e Controle

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

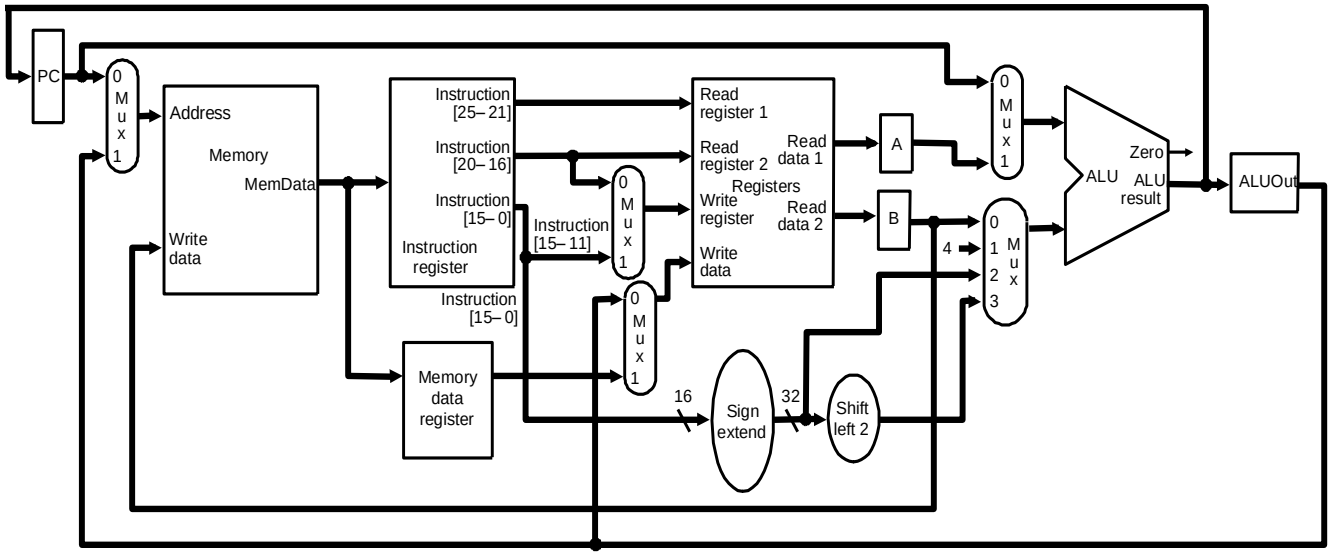
Implementação de um único ciclo

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						

- 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*: 8x24% + 7x12% + 6x44% + 5x18% + 2x2% = 6,3 ns
- Ganho de desempenho: Tempo execução *clock* fixo / 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 ns
- A mesma relação pode ser feita para operações em ponto flutuante e o ganho favorável ao *clock* variável pode ser ainda maior.

Abordagem multiciclo

- Quebre a instrução em passos, cada passo leva um ciclo
  - balanceie a quantidade de trabalho a ser feita
  - restrinja cada ciclo a usar apenas uma unidade funcional maior
- No final de cada ciclo
  - armazene os valores para uso em ciclos posteriores (a coisa mais fácil a ser feita)
  - introduza registradores “internos” adicionais



Cinco passos para execução

- 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]) << 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 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;

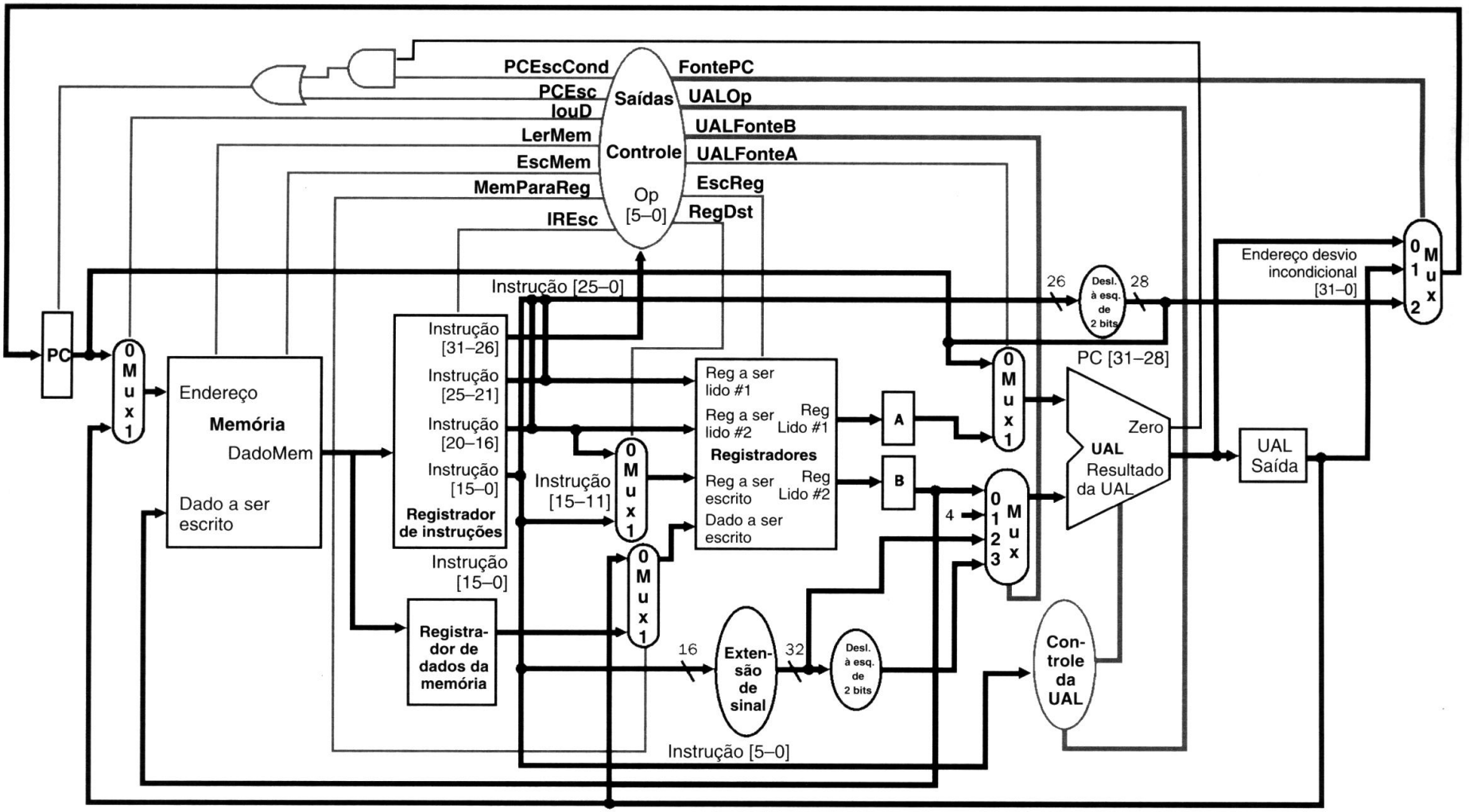
Instruções levam de 3 a 5 ciclos.

Sumário

Nome do passo	Ação nas instruções de tipo R	Ação nas instruções de referência à memória	Ação na instrução de desvio condicional	Ação na instrução de desvio incondicional
Busca da instrução	IR = Memória[PC] PC = PC + 4			
Decodificação da instrução/busca dos valores dos registradores	A = Reg [IR[25–21]] B = Reg [IR[20–16]] UALSaída = PC + extensão de sinal (IR[15–0]) << 2)			
Execução, cálculo do endereço, término de uma instrução de branch/jump	ALUOut = A op B	UALSaída = A + extensão de sinal (IR[15–0])	se (A == B) então PC = UALSaída	PC = PC [31–28] II (IR[25–0]<<2)
Término de uma instrução de referência à memória ou de tipo R	Reg [IR[15–11]] = UALSaída	Load: MDR = memória[UALSaída] ou Store: Memória [ALUOut] = B		
Término de leitura da memória		Load: Reg[IR[20–16]] = MDR		

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR = Memory[PC] PC = PC + 4			
Instruction decode/register fetch	A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A ==B) then PC = ALUOut	PC = PC [31-28] II (IR[25-0]<<2)
Memory access or R-type completion	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
Memory read completion		Load: Reg[IR[20-16]] = MDR		

Abordagem multiciclo



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

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?**

Pipelines

MIPS

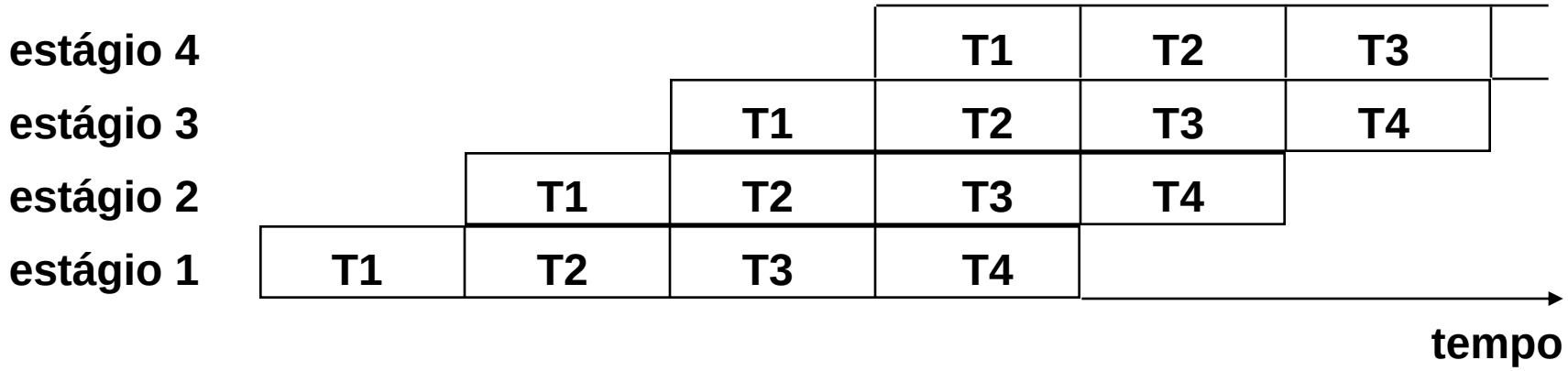
Pipelines

primeira parte

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

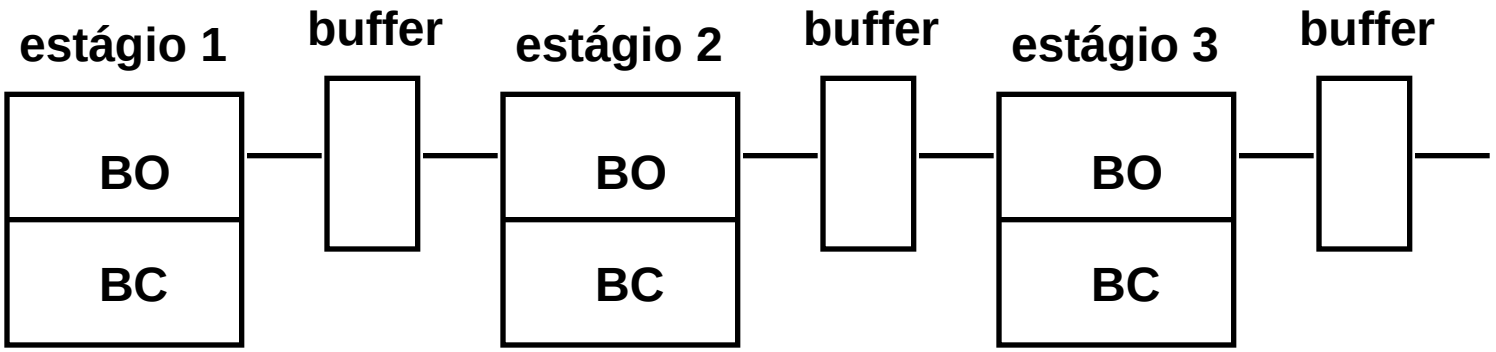
Introdução

- 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 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	**x 10^6**	**–**	**0.842**	**x 10^5**

0.157	**x 10^6**	**–**	**0.0842**	**x 10^6**

0.0628	**x 10^6**

0.628	**x 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

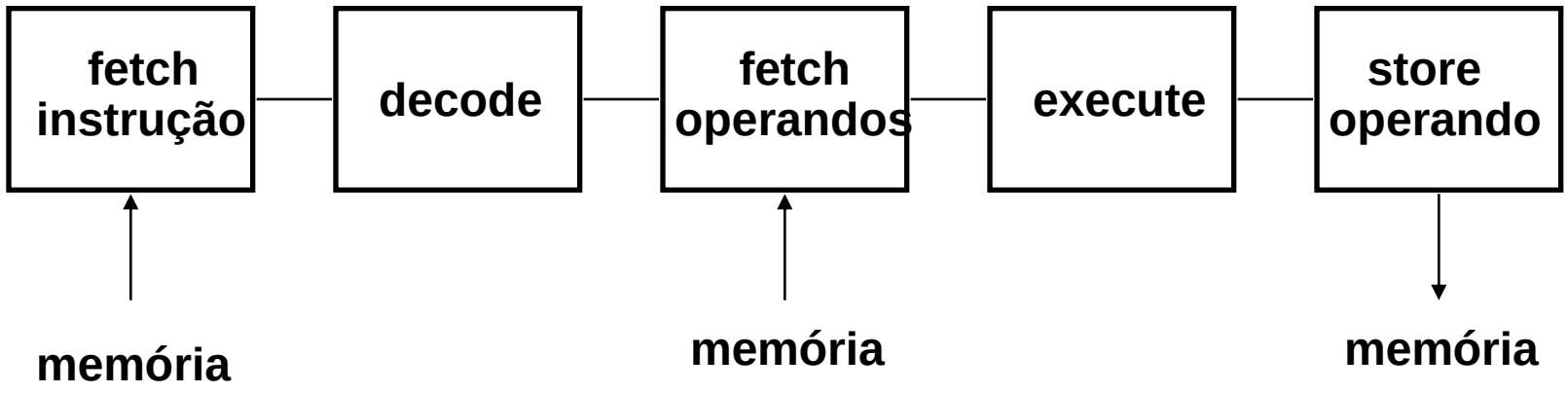
$$\textbf{Speed–up teórico} = \frac{s\,N}{N + (s - 1)} \qquad \textbf{limite} = N$$

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

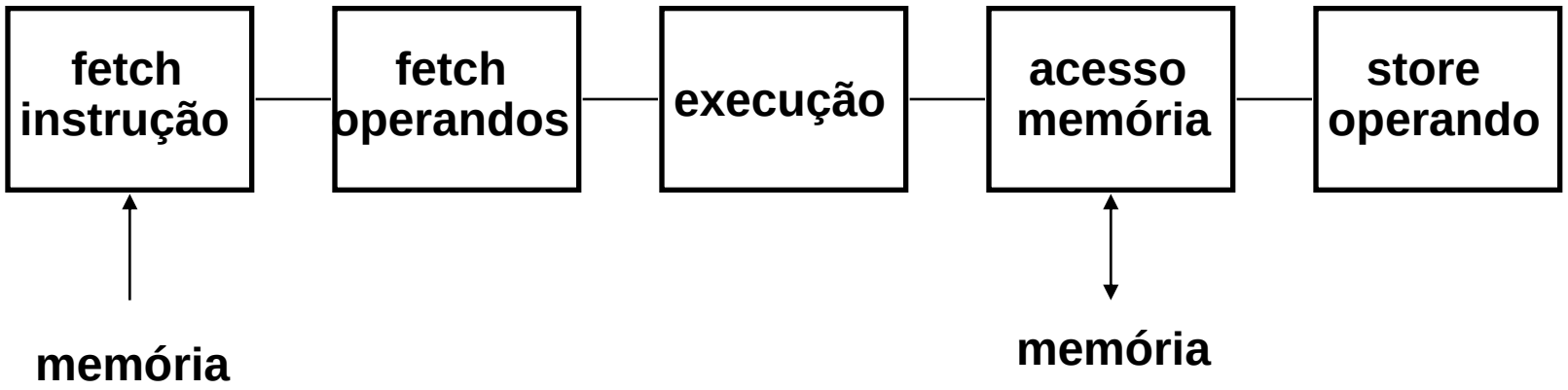
Conflitos de memória

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



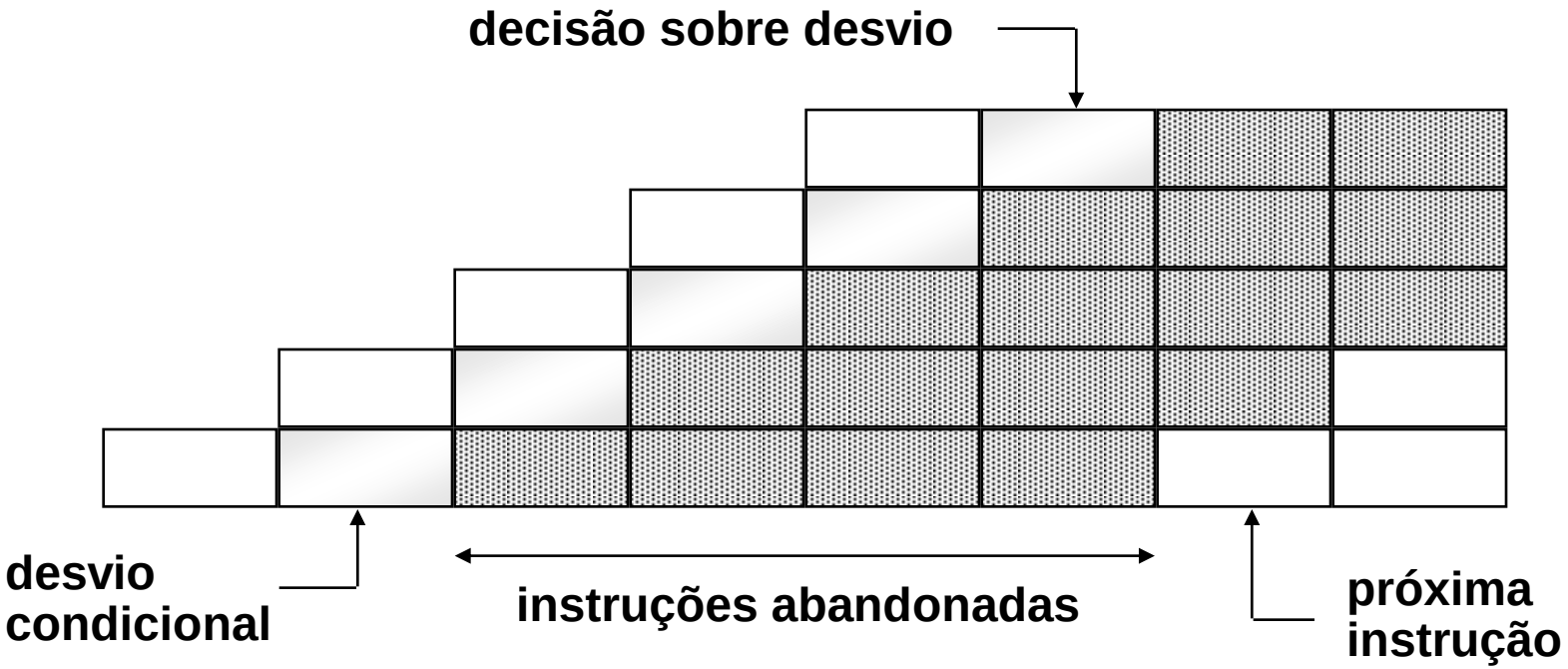
Processadores RISC

- 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?

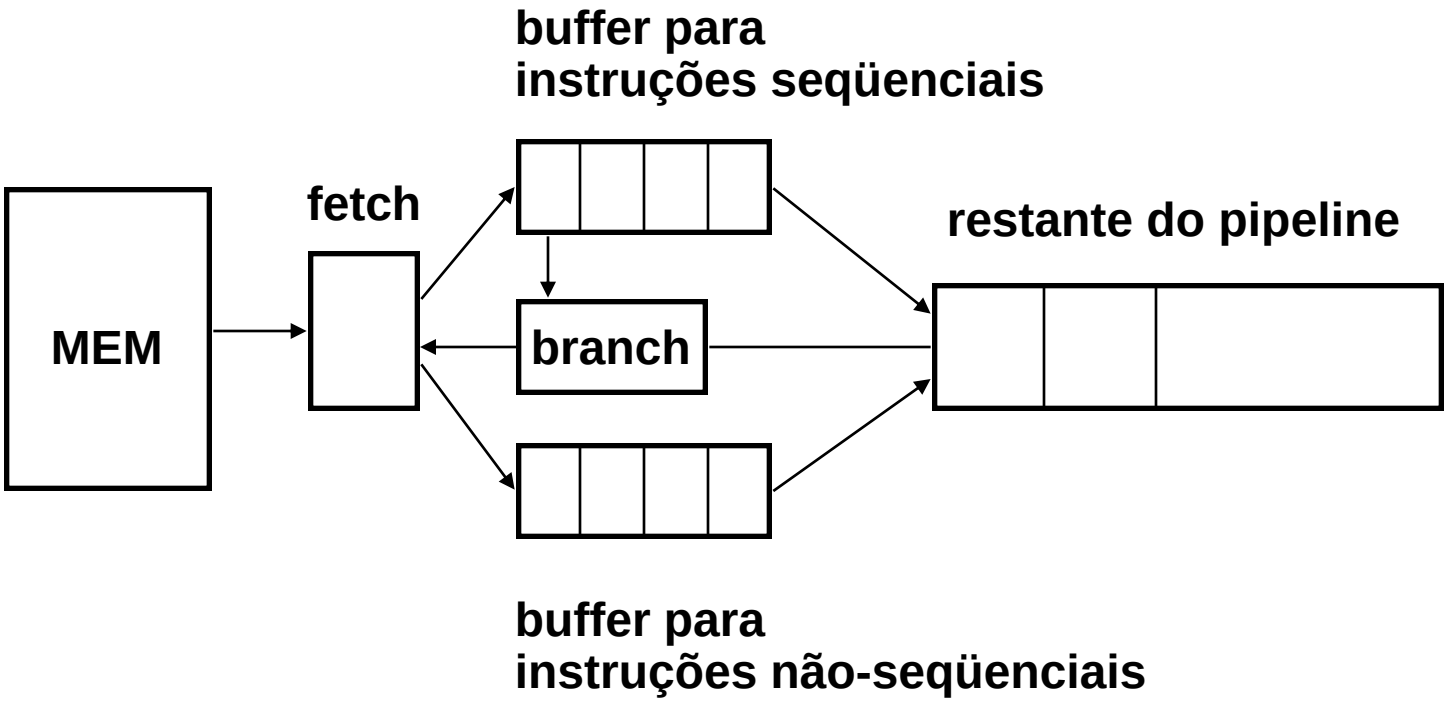
Técnicas de tratamento dos desvios condicionais

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”

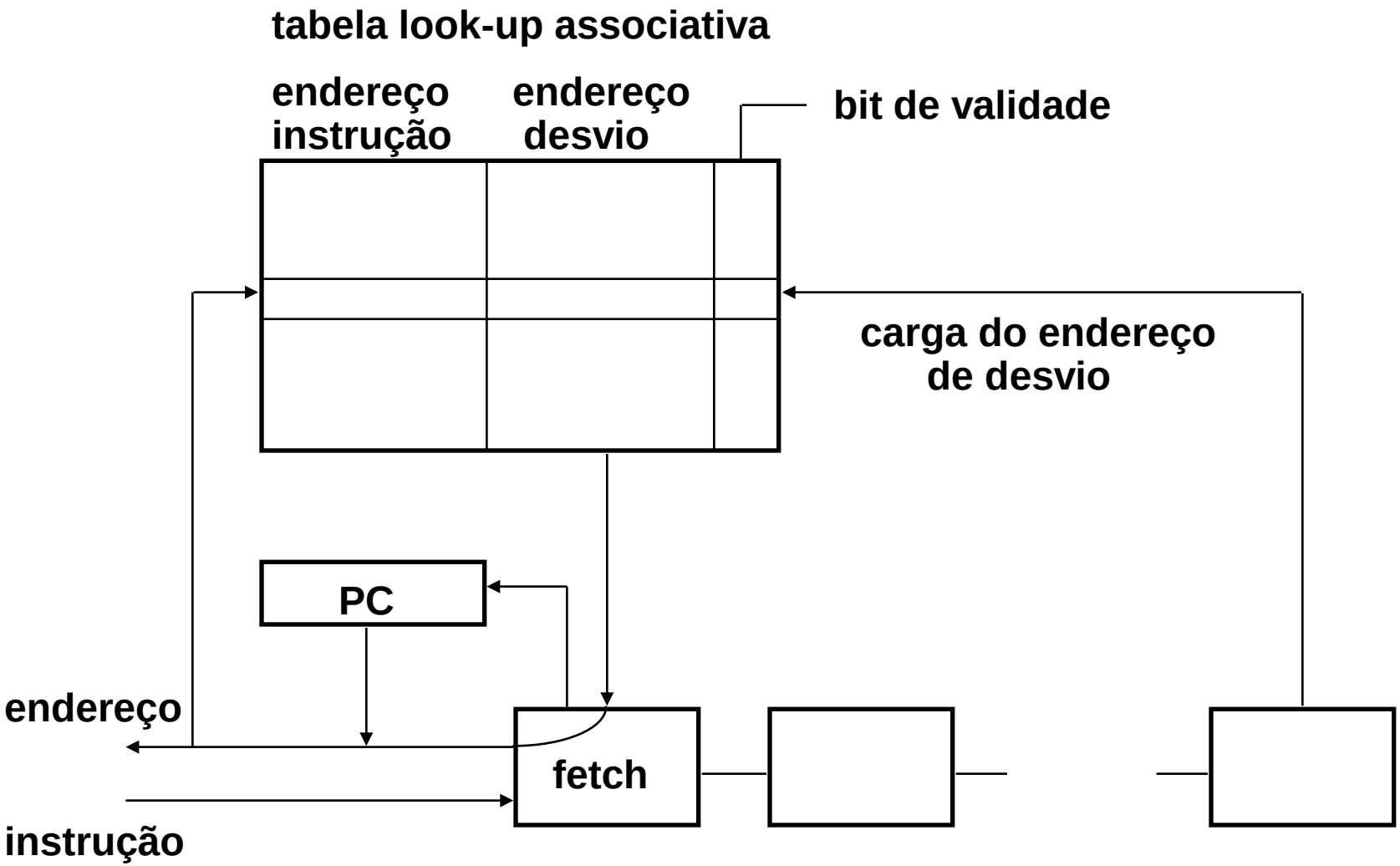
Buffers paralelos de instruções



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

Pipelines

Segunda parte

1. Introdução
2. Dependências verdadeiras
3. Dependências falsas
4. Pipeline interlock
5. Forwarding

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

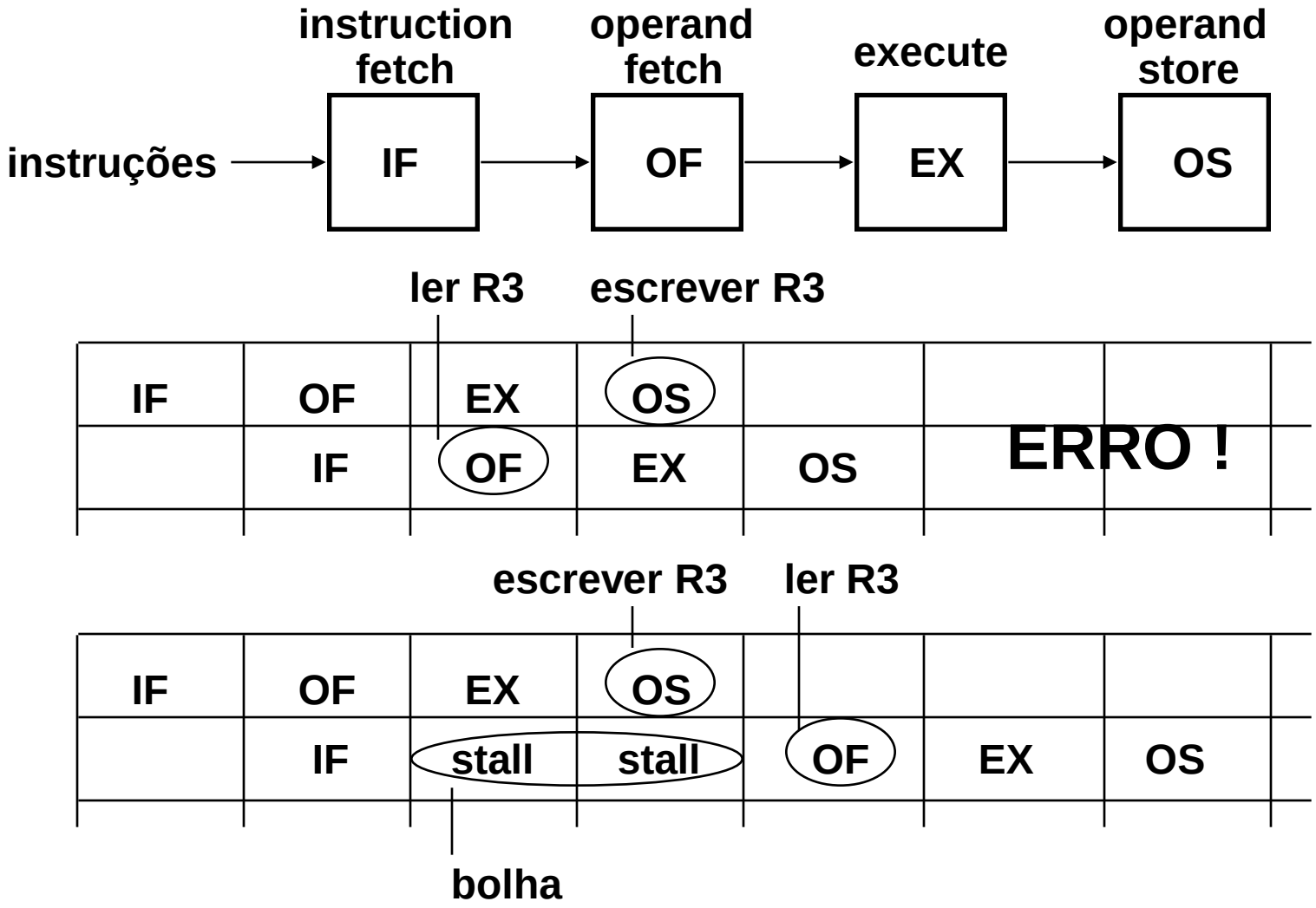
Dependências verdadeiras

- 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

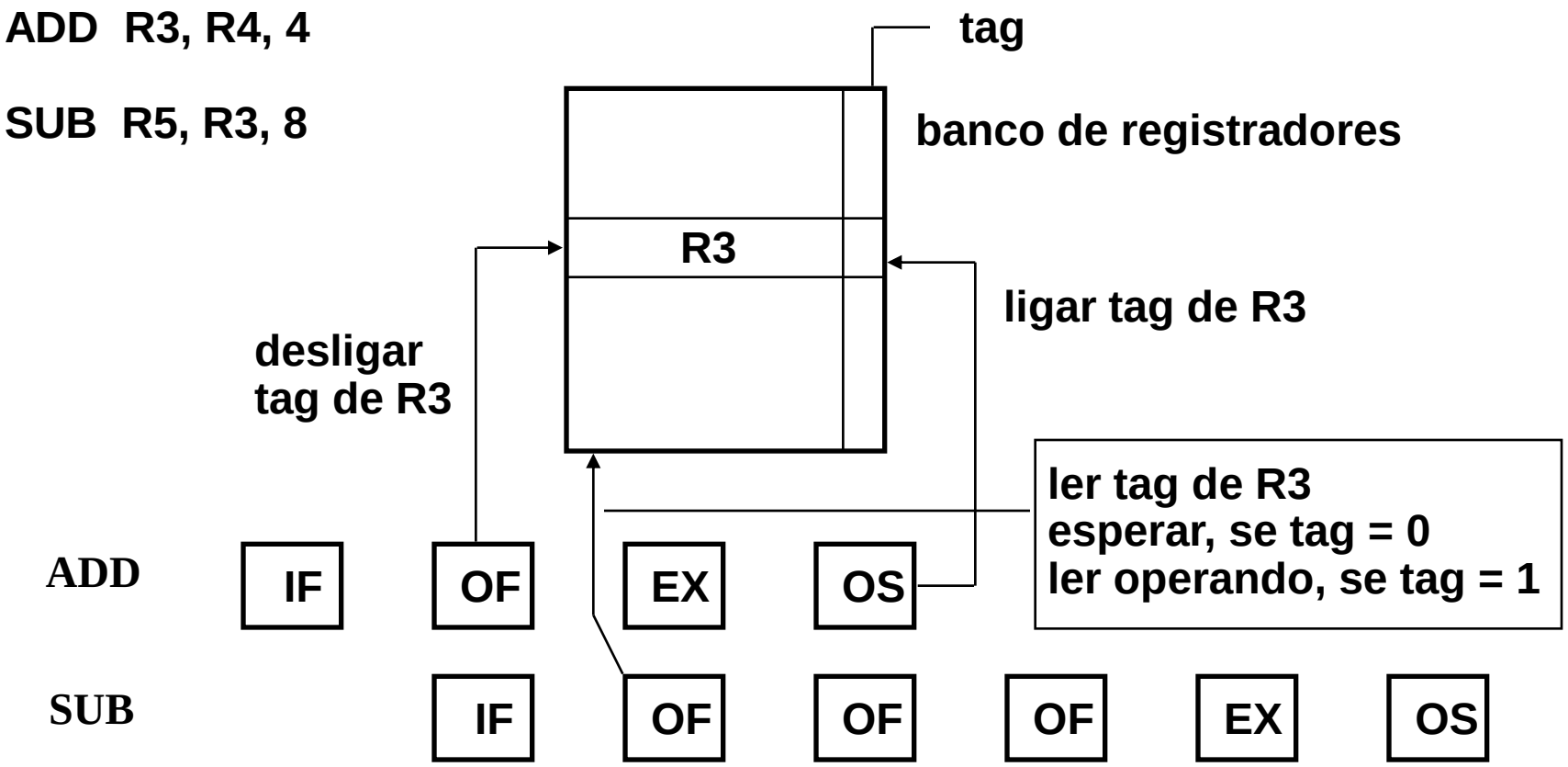
- instruções 1 e 3 escrevem no mesmo operando em R3
- instrução 1 tem que escrever seu resultado em R3 antes do que a instrução 3, senão valor final de R3 ficará errado
- *write-after-write hazard*
- também só é problema em processadores superescalares

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



Forwarding

- exemplo

  ADD R3, R2, R0

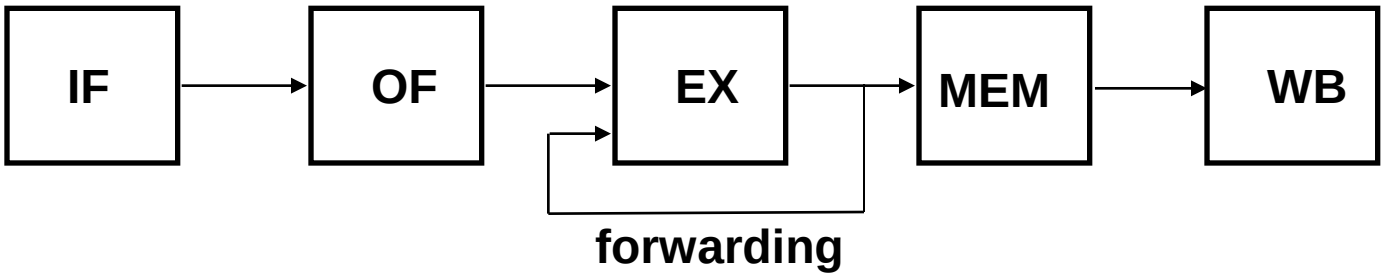
  SUB R4, R3, 8

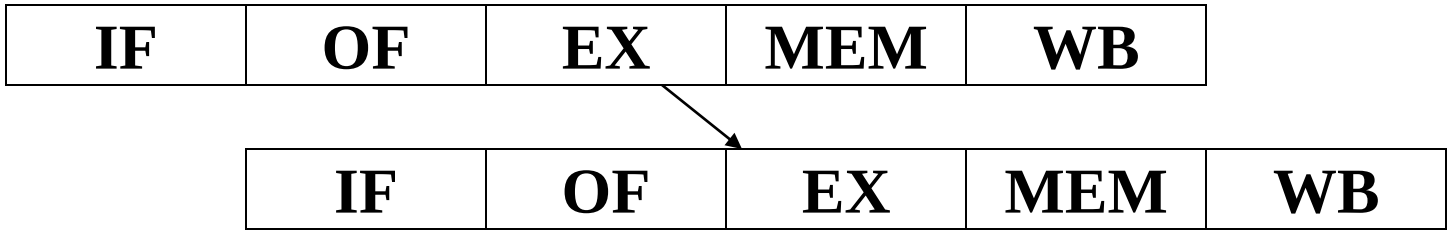
- instrução SUB precisa do valor de R3, calculado pela instrução ADD
  - valor é escrito em R3 por ADD no último estágio WB (write-back)
  - valor é necessário em SUB no terceiro estágio
  - instrução SUB ficará presa por 2 ciclos no 2º estágio do pipeline

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





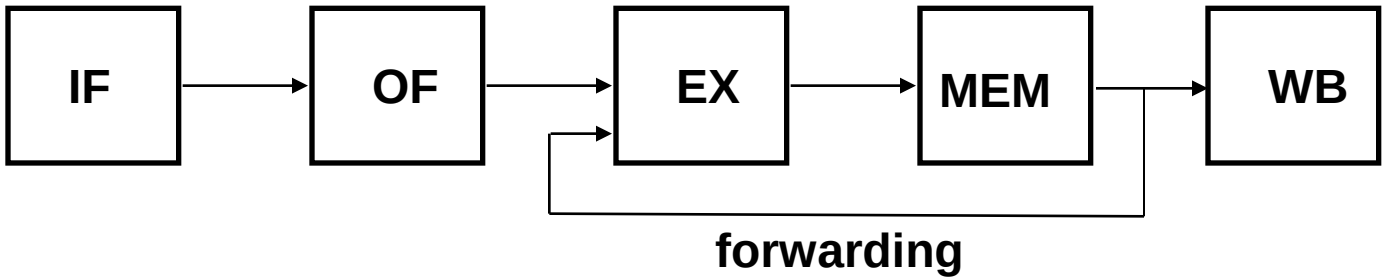
Forwarding

- exemplo 2

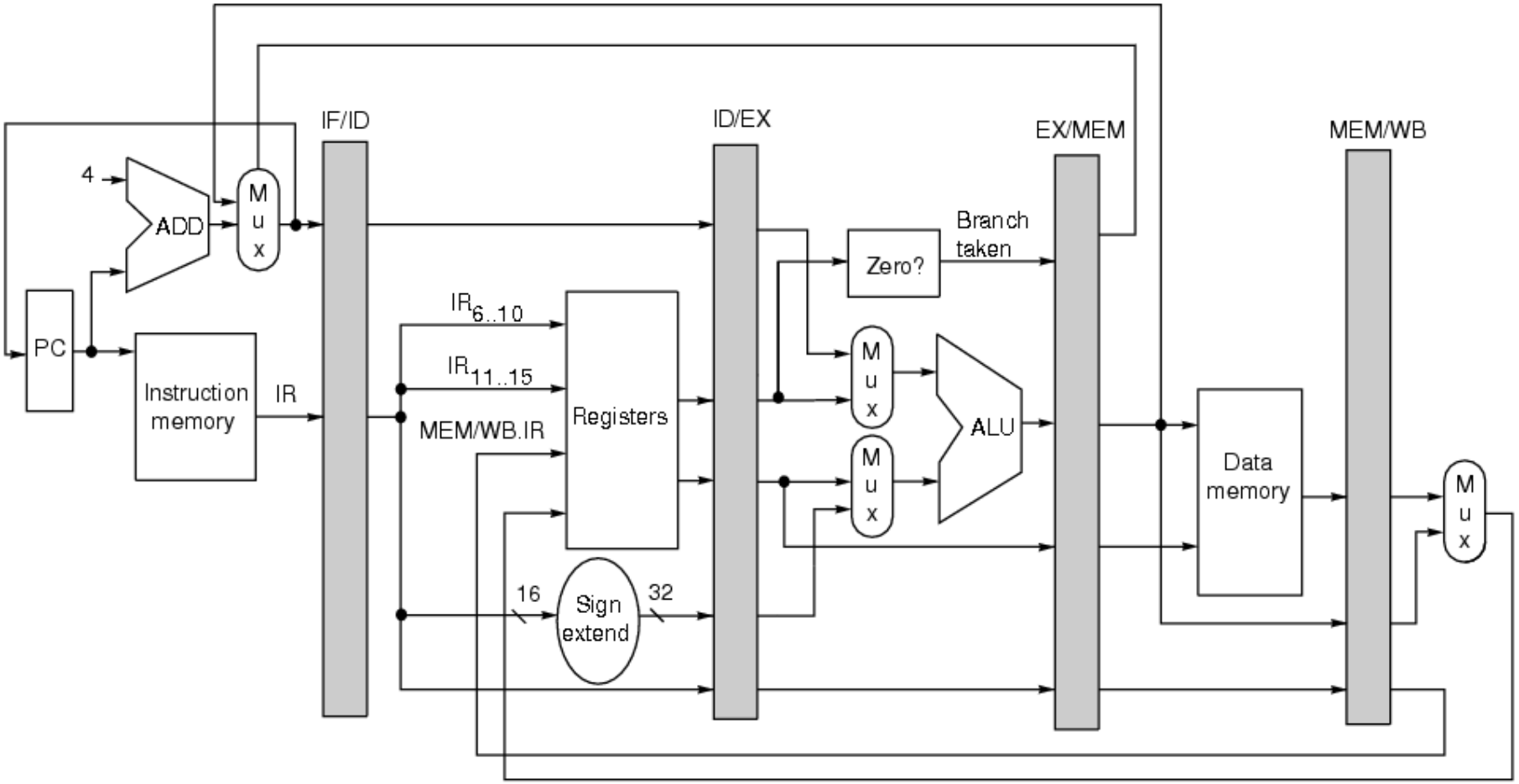
  LOAD R3, 100 (R0)

  ADD R1, R2, R3

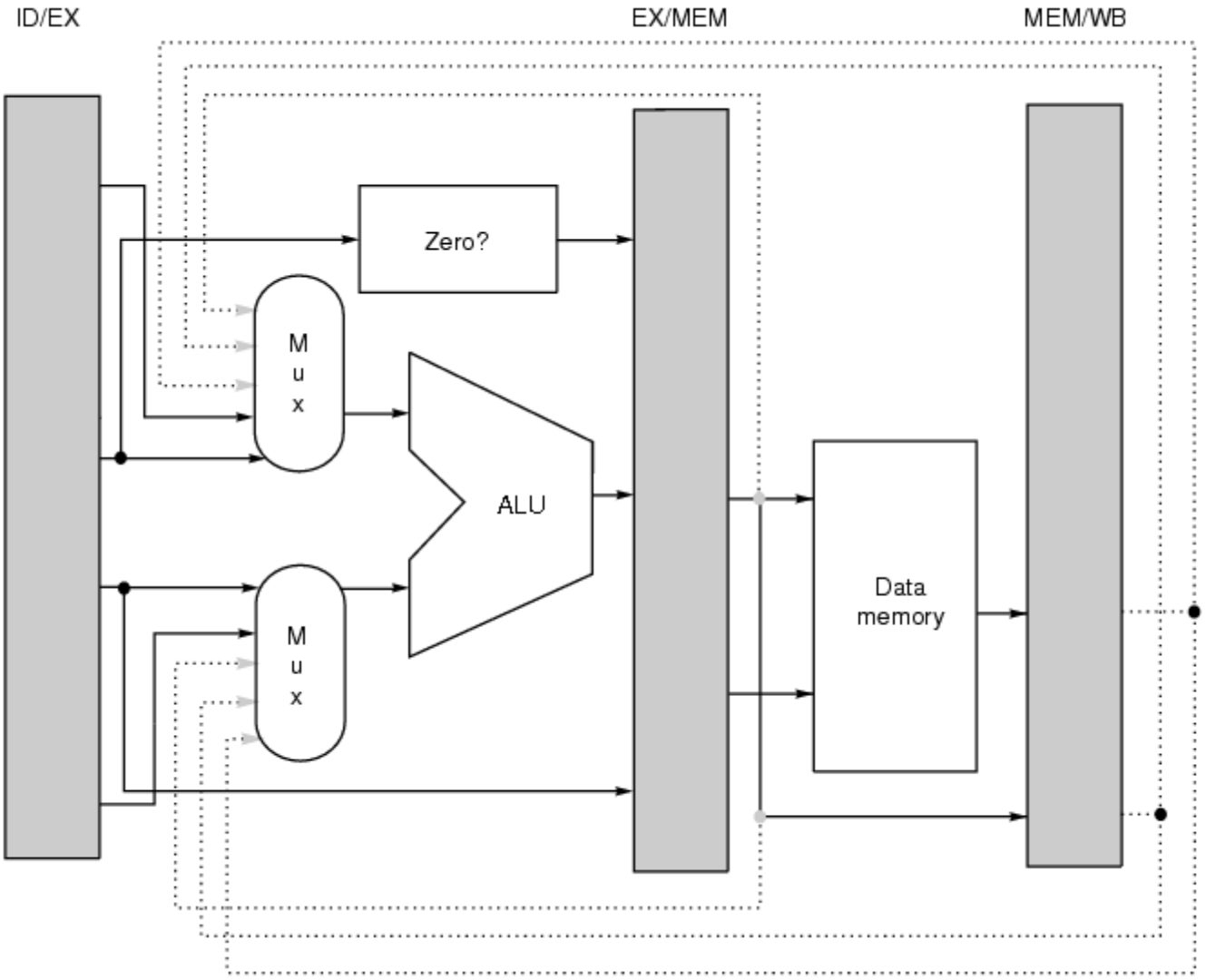
- forwarding: caminho interno dentro do pipeline entre a saída da memória de dados e a entrada da ALU



Como fazer o adiantamento de dados?



Forwarding



Pipelines

Terceira parte

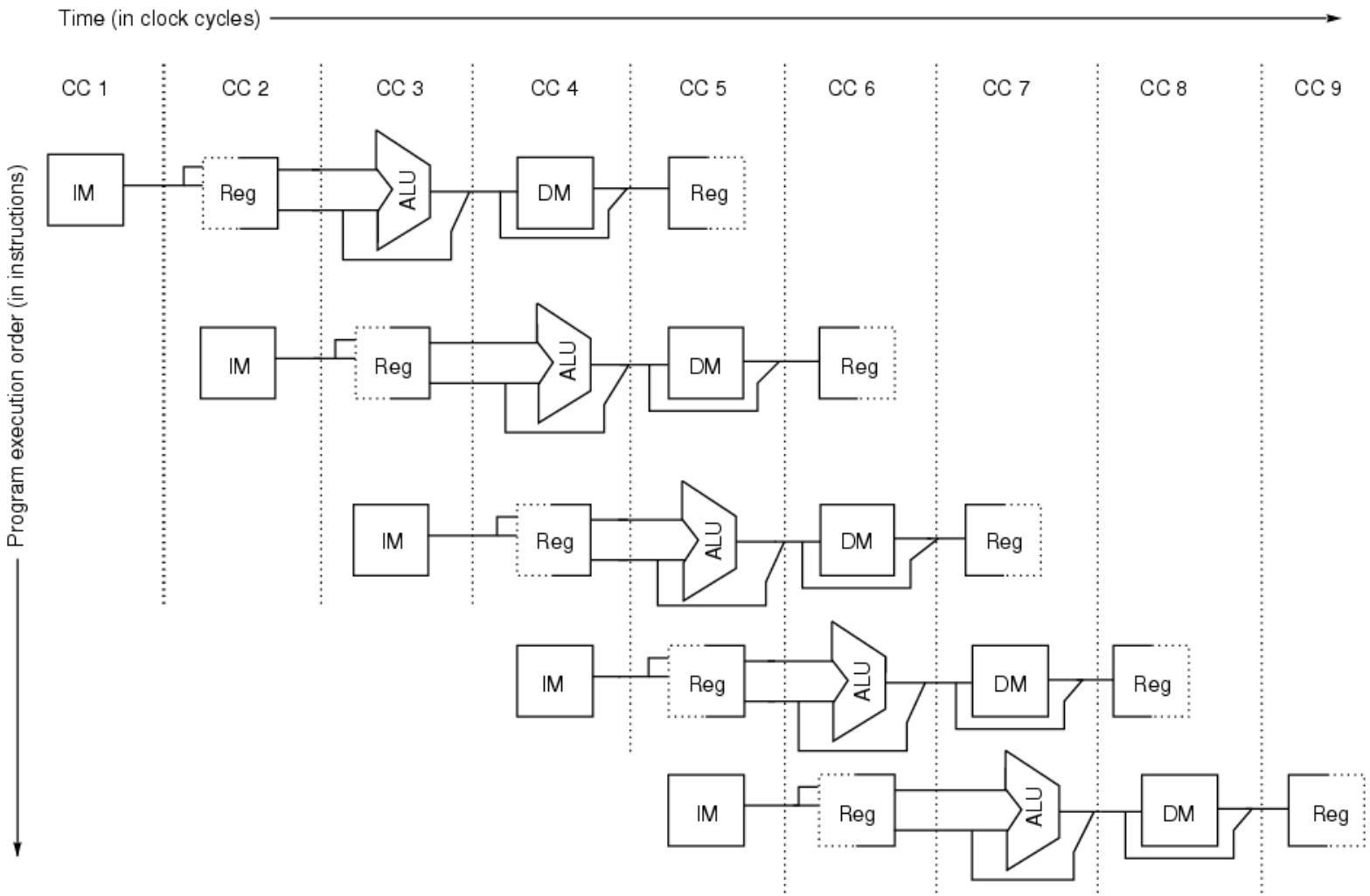
1. Pipeline do DLX (MIPS)
2. Dependências no Pipeline
3. Forwarding
4. Exemplos
5. Instruções de Desvio no Pipeline

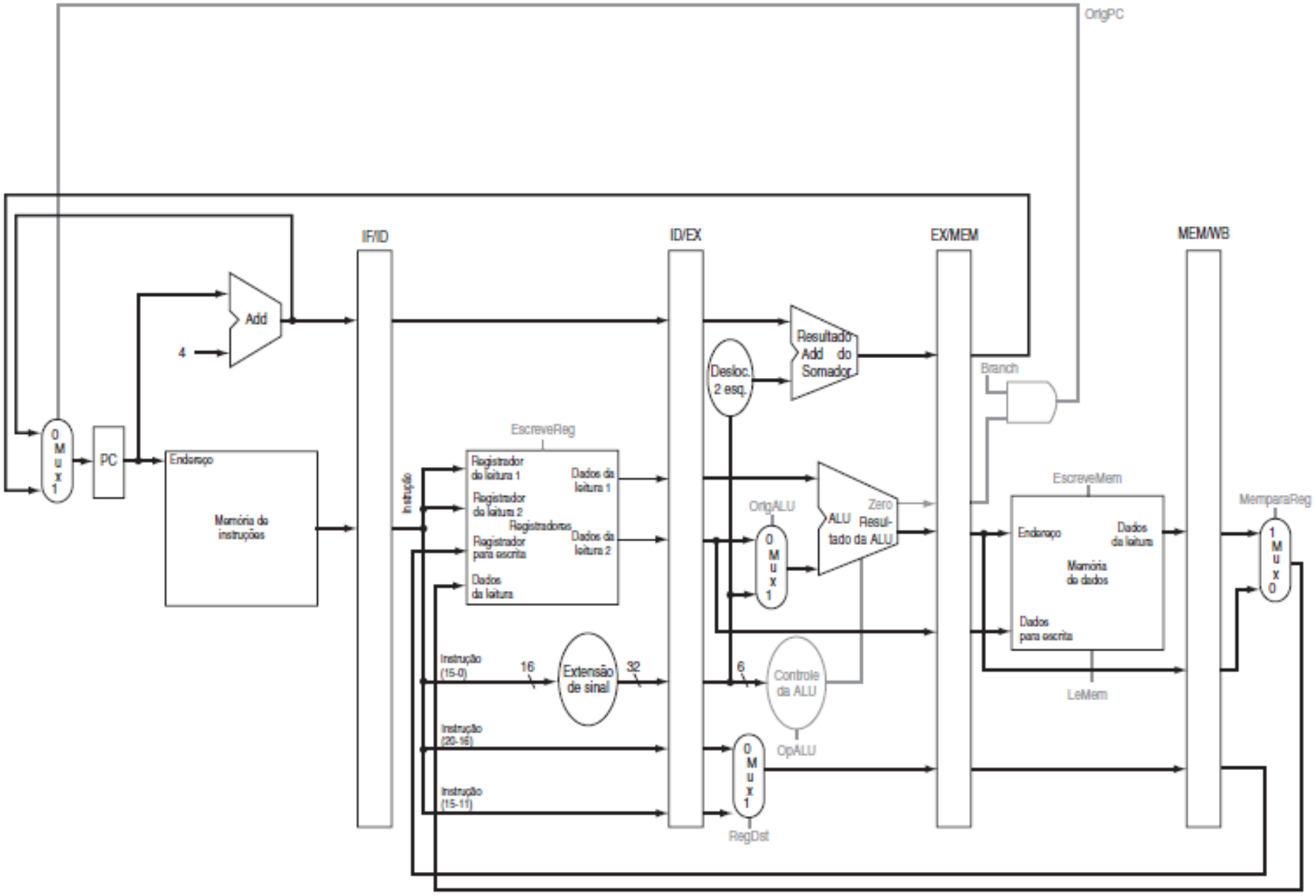
Pipeline do DLX

Instrução	Clock								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
i	IF	ID	EX	MEM	WB				
i + 1		IF	ID	EX	MEM	WB			
i + 2			IF	ID	EX	MEM	WB		
i + 3				IF	ID	EX	MEM	WB	
i + 4					IF	ID	EX	MEM	WB

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

Pipeline do DLX

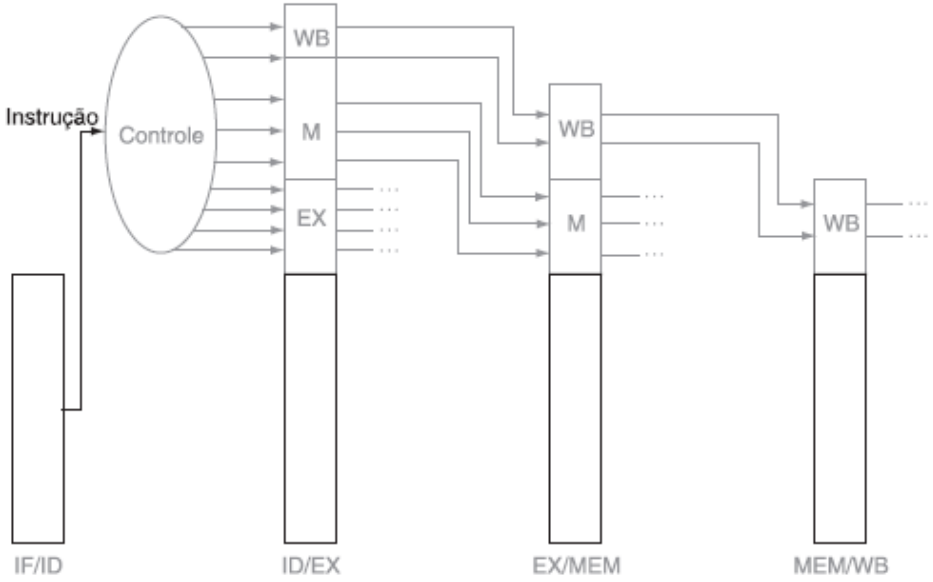


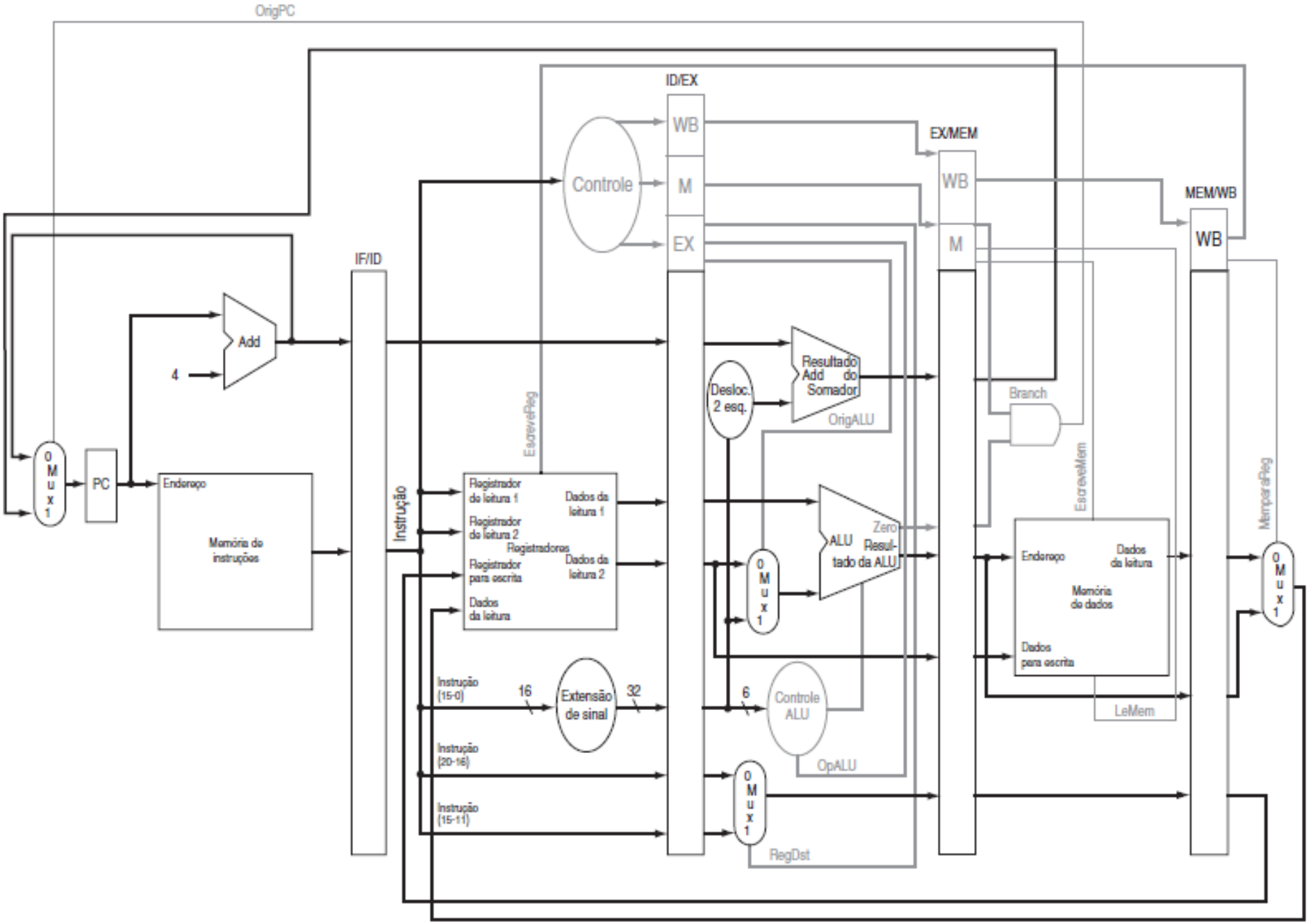

OrigPC
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Endereço
Memória de instruções
Instrução
EscreveReg
Registrador de leitura 1
Registrador de leitura 2
Registradores
Registrador para escrita
Dados de leitura
Dados de leitura 1
Dados de leitura 2
Desloc. 2 esq.
Resultado Add do Somador
Branch
OrigALU
ALU
Zero
Resul-tado da ALU
EscreveMem
Endereço
Dados de leitura
Memória de dados
Dados para escrita
LeMem
MemparaReg
Instrução (15-0)
16
Extensão de sinal
32
6
Controle da ALU
OpALU
Instrução (20-16)
Instrução (15-11)
RegDst

Controle do pipeline

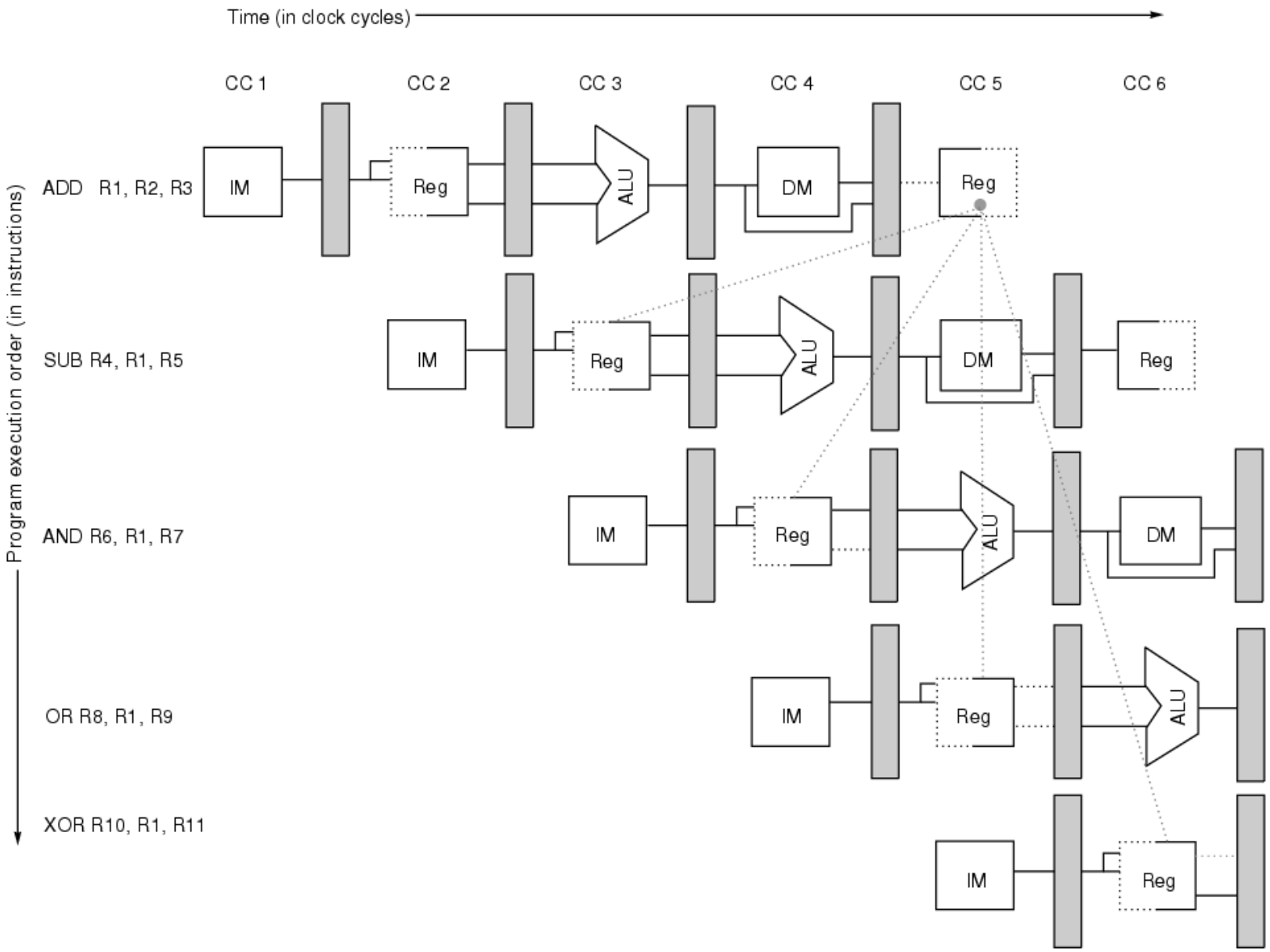
- Transfira os sinais de componente exatamente como os dados

	Linhas de controle do estágio de cálculo de endereço/execução				Linhas de controle do estágio de acesso à memória			Linhas de controle do estágio de escrita do resultado	
Instrução	RegDst	OpALU1	OpALU0	OrigALU	Branch	LeMem	Escreve Mem	Escreve Reg	Mem para Reg
Formato R	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

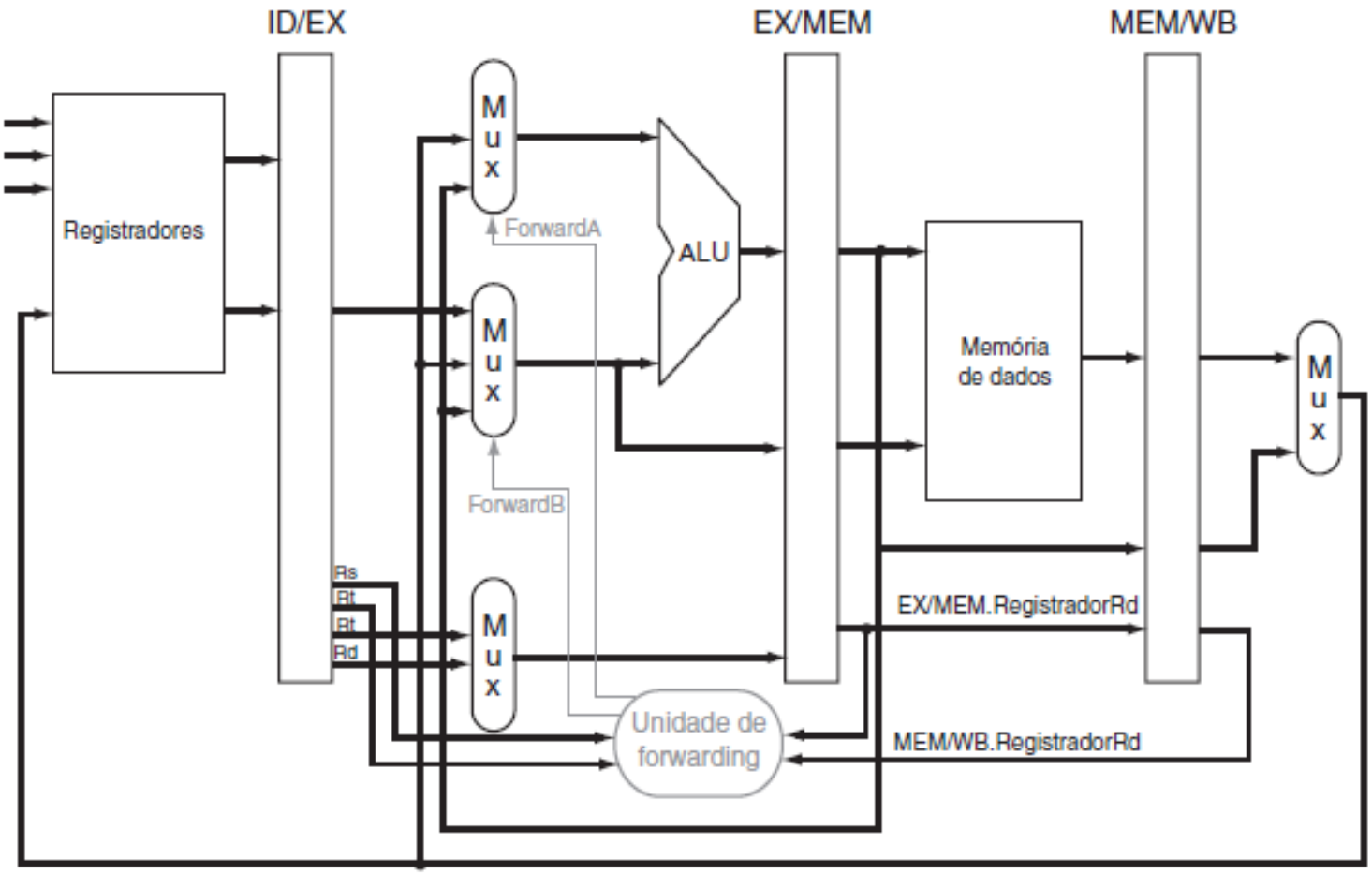



OrigPC
ID/EX
EX/MEM
MEM/WB
WB
M
EX
Controle
IF/ID
Add
4
0
M
u
x
1
PC
Endereço
Memória de instruções
Instrução
EscreveReg
Registrador da leitura 1
Registrador da leitura 2
Registradores
Registrador para escrita
Dados da leitura
Dados da leitura 1
Dados da leitura 2
Instrução (15-0)
16
Extensão de sinal
32
Instrução (20-16)
Instrução (15-11)
Desloc. 2 esq.
Resultado Add do Somador
OrigALU
ALU
Zero
Resul-tado da ALU
6
Controle ALU
OpALU
RegDst
Branch
EscreveMem
LeMem
Endereço
Dados da leitura
Memória de dados
Dados para escrita
MemparaReg

Dependências no Pipeline

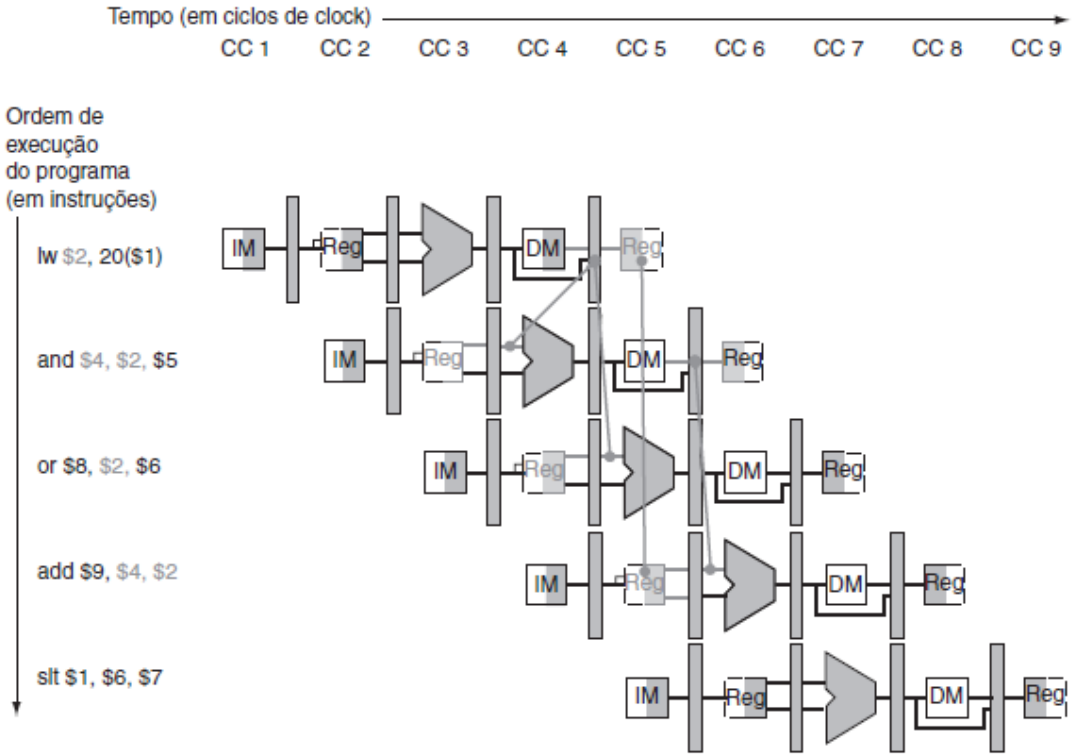


Forwarding


ID/EX
EX/MEM
MEM/WB
Registradores
M u x
ForwardA
ALU
Memória de dados
ForwardB
Rs
Rt
Rt
Rd
EX/MEM.RegistradorRd
Unidade de forwarding
MEM/WB.RegistradorRd

Forwarding nem sempre é possível

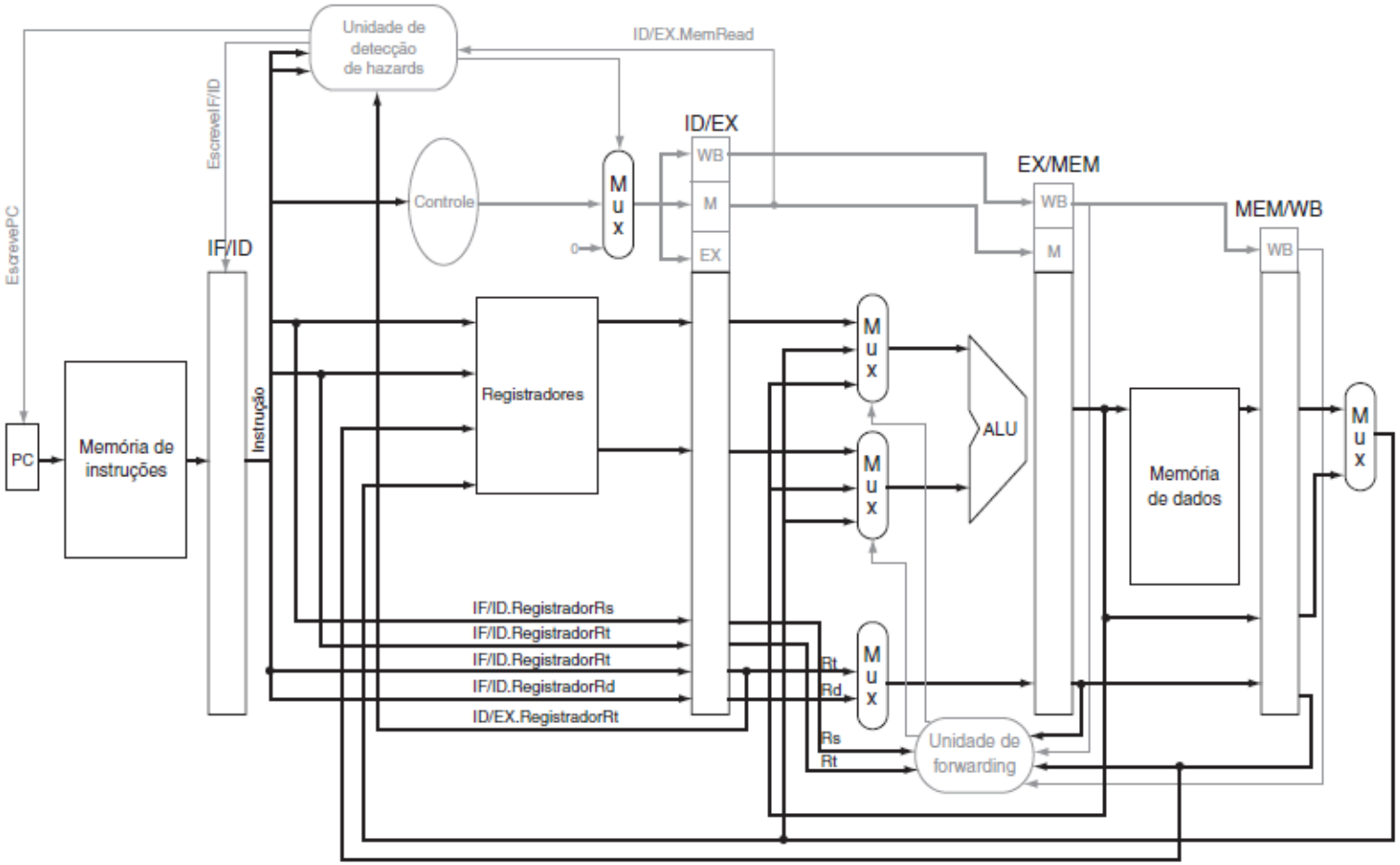
- **Load word ainda pode causar um hazard:**
  - **uma instrução tenta ler um registrador seguindo uma instrução load que escreve no mesmo registrador.**



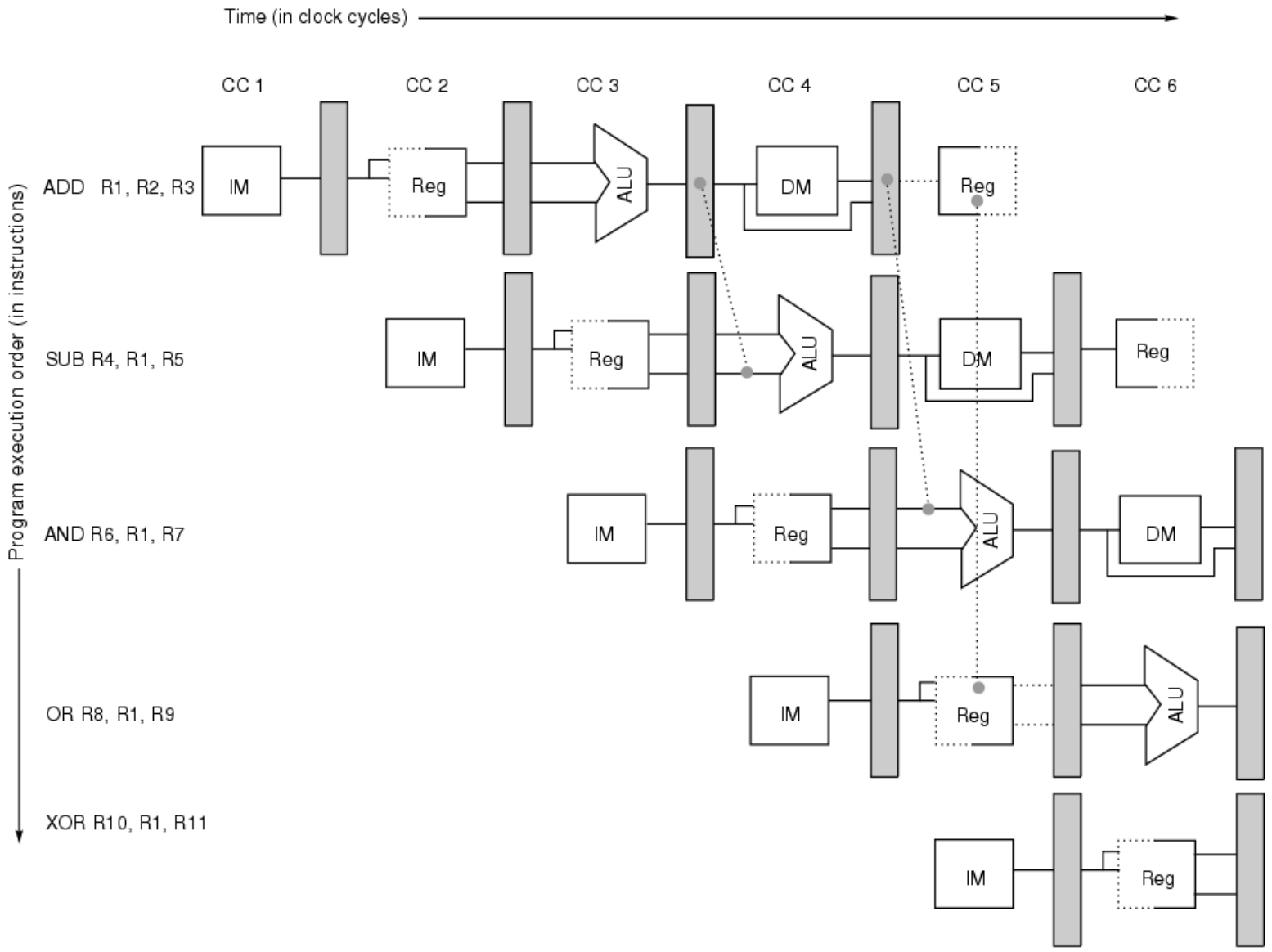
Portanto, precisamos de uma unidade de detecção de hazard para "deter" (stall) a instrução load

Unidade de detecção de hazard

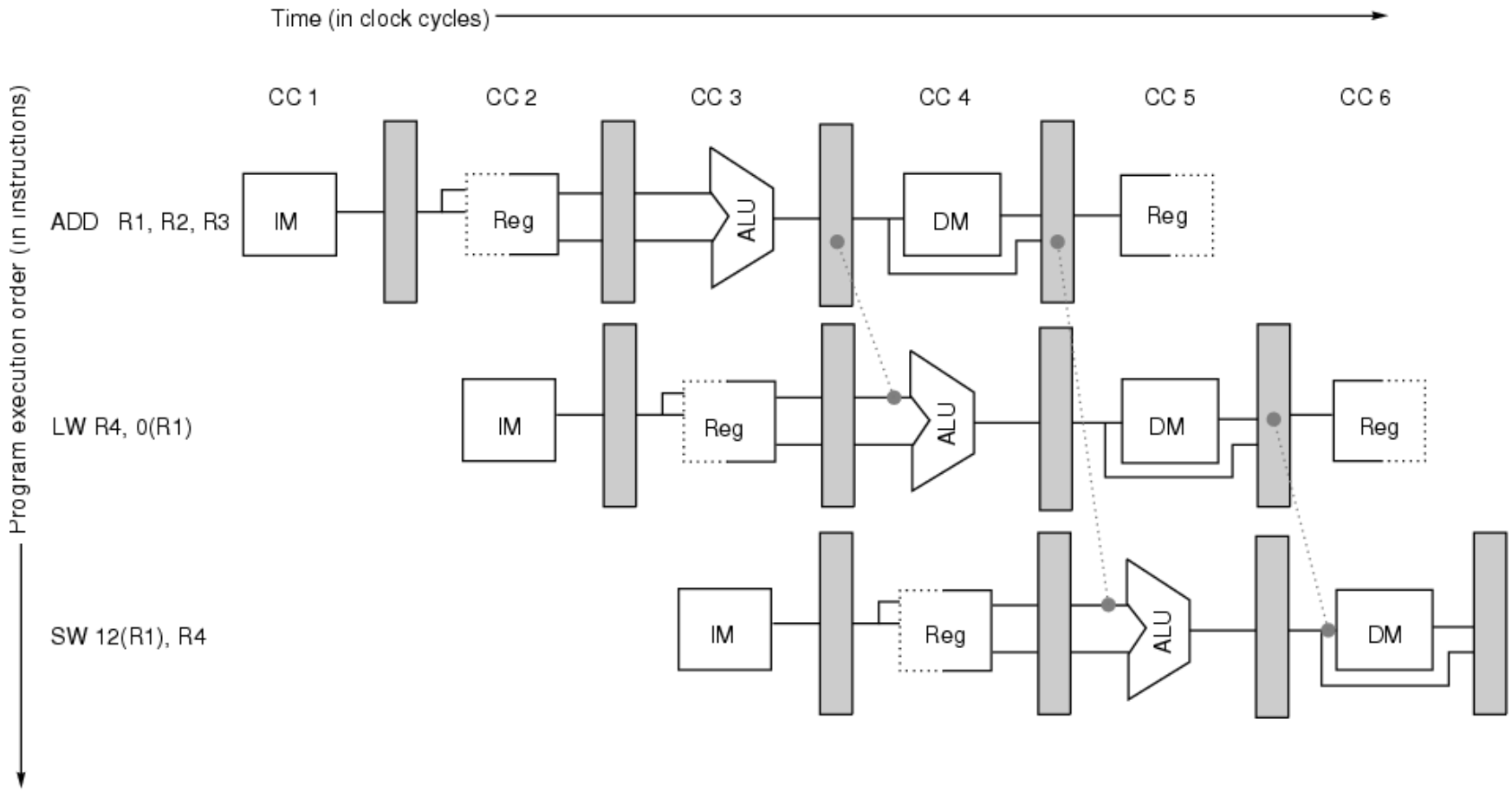
- **Ocasione um stall deixando que uma instrução que não escreverá nada prossiga**



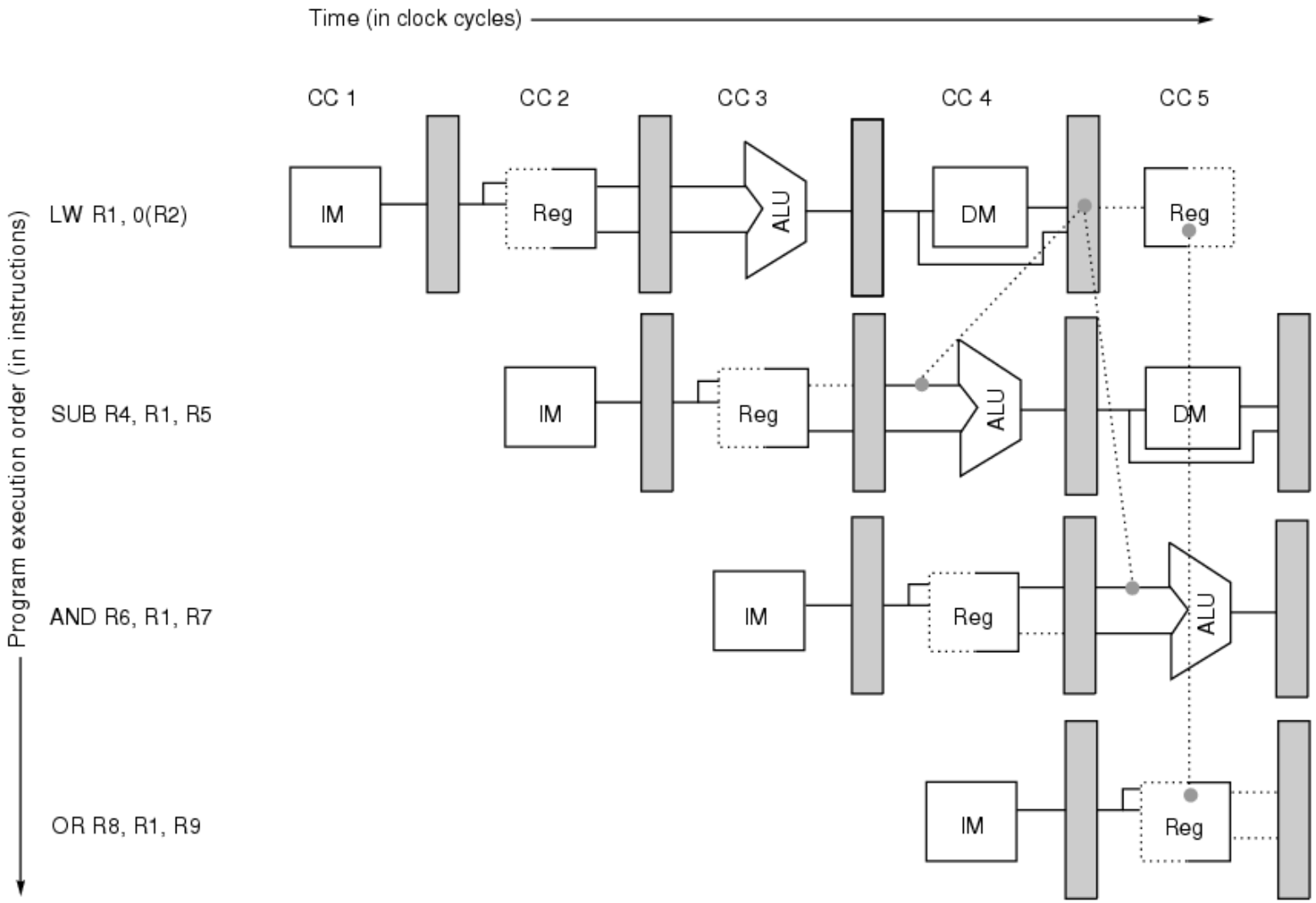
Forwarding


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
Program execution order (in instructions)
ADD R1, R2, R3
SUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
XOR R10, R1, R11
IM
Reg
ALU
DM
Reg

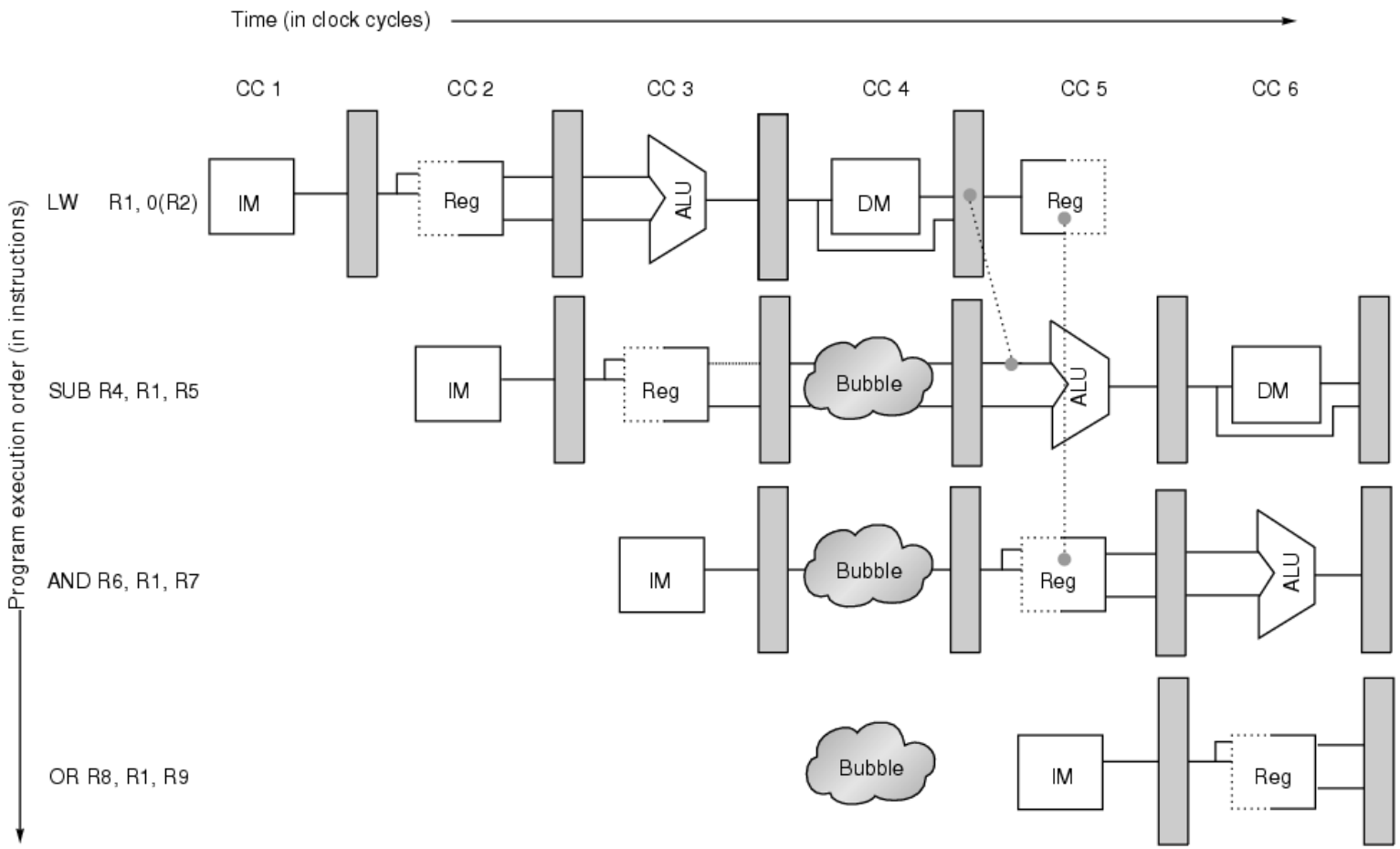
Forwarding


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
Program execution order (in instructions)
ADD R1, R2, R3
LW R4, 0(R1)
SW 12(R1), R4
IM
Reg
ALU
DM
Reg

Forwarding


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
Program execution order (in instructions)
LW R1, 0(R2)
SUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
IM
Reg
ALU
DM
Reg

Forwarding


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
Program execution order (in instructions)
LW R1, 0(R2)
SUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
IM
Reg
ALU
DM
Bubble

Instruções de Desvio no Pipeline

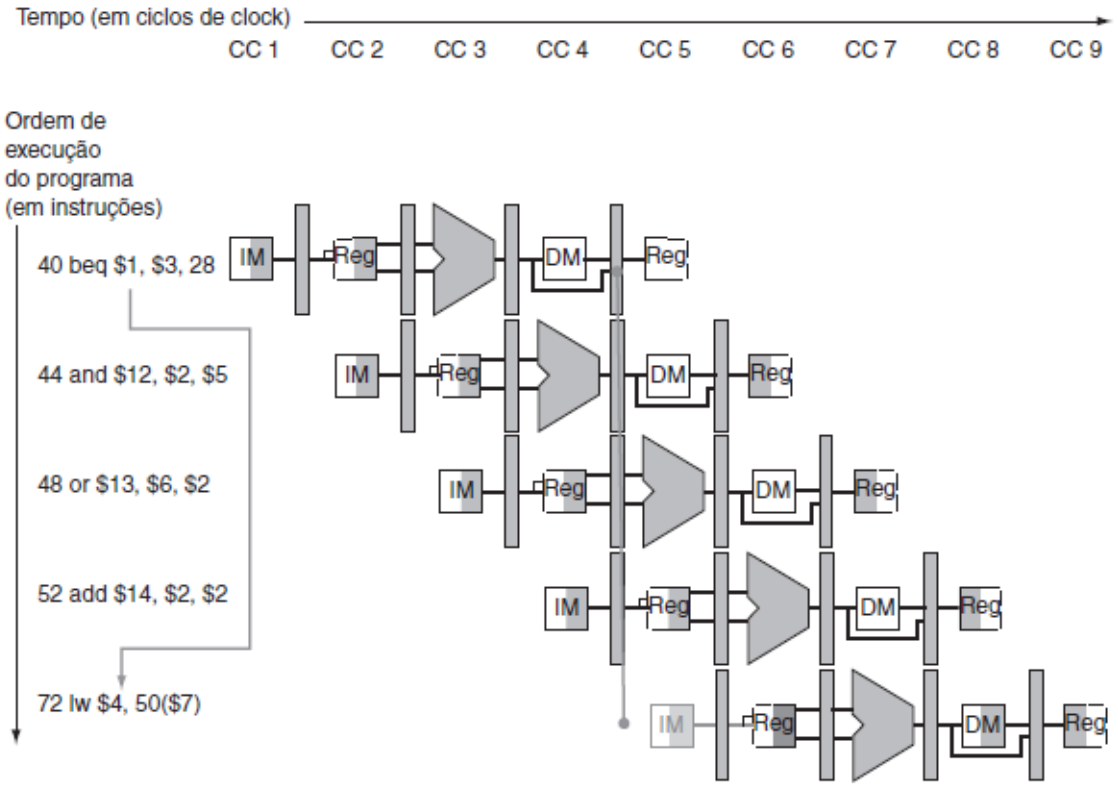
Desvio	IF	ID	EX	MEM	WB				
IS		IF	*stall*	*stall*	*IF*	ID	EX	MEM	WB
IS + 1						IF	ID	EX	MEM	WB
IS + 2							IF	ID	EX	MEM
IS + 3								IF	ID	EX
IS + 4									IF	ID
IS + 5										IF

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).**

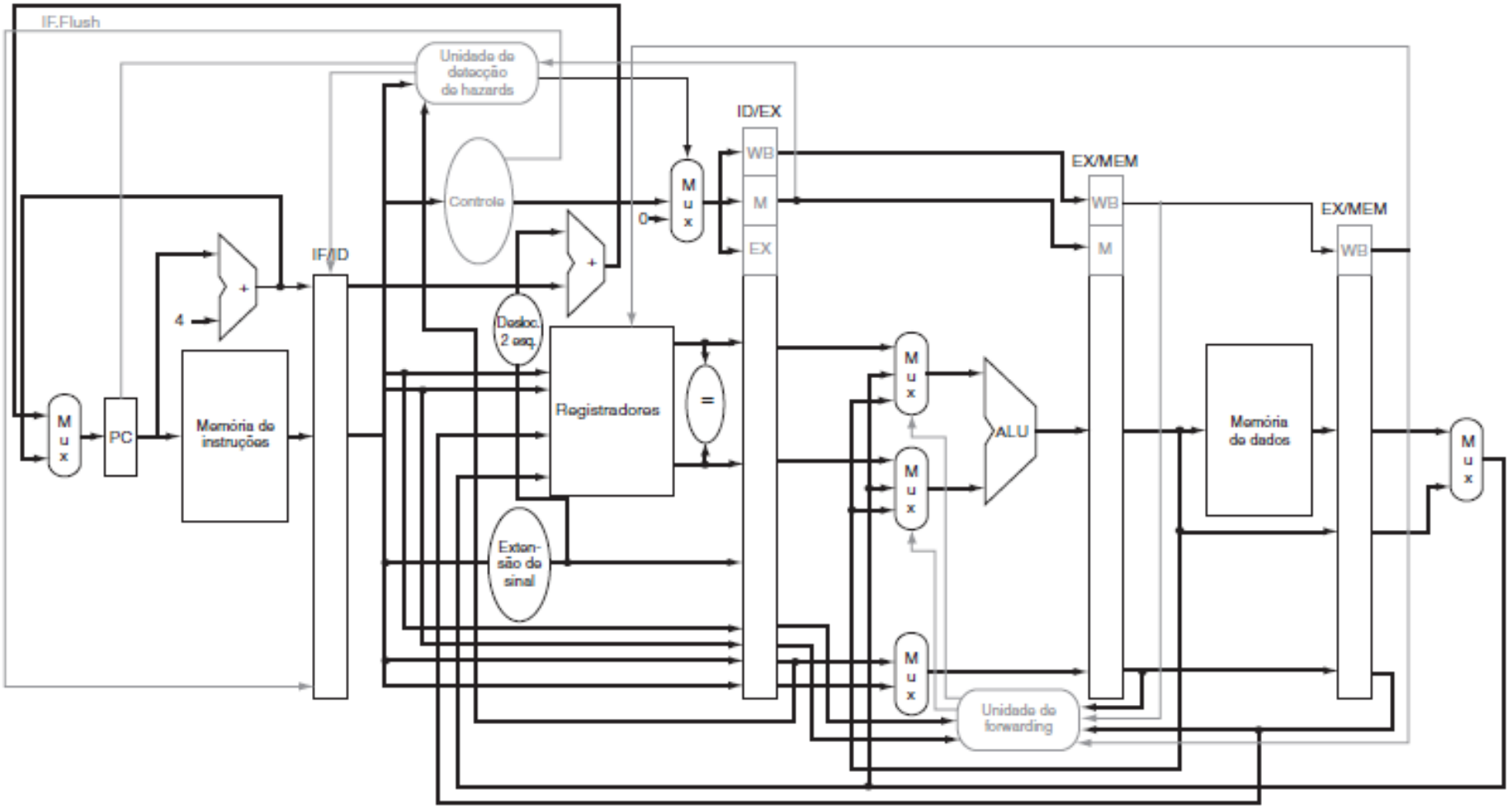
Hazards de desvio

- **Quando decidimos desviar, outras instruções estão no pipeline!**

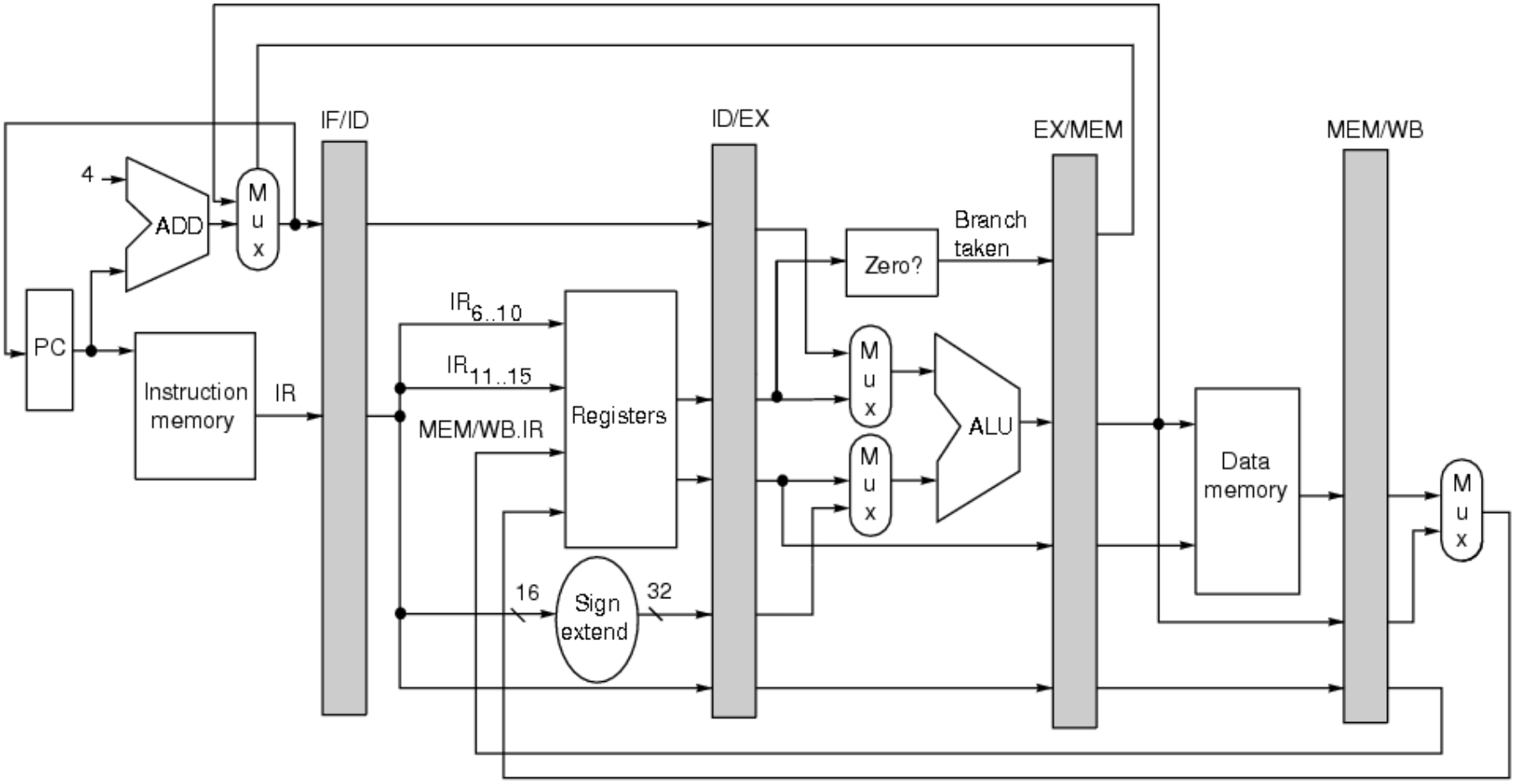


- **Estamos prevendo "desvio não tomado"**
  - **precisamos acrescentar hardware para descartar instruções se estivermos errados**

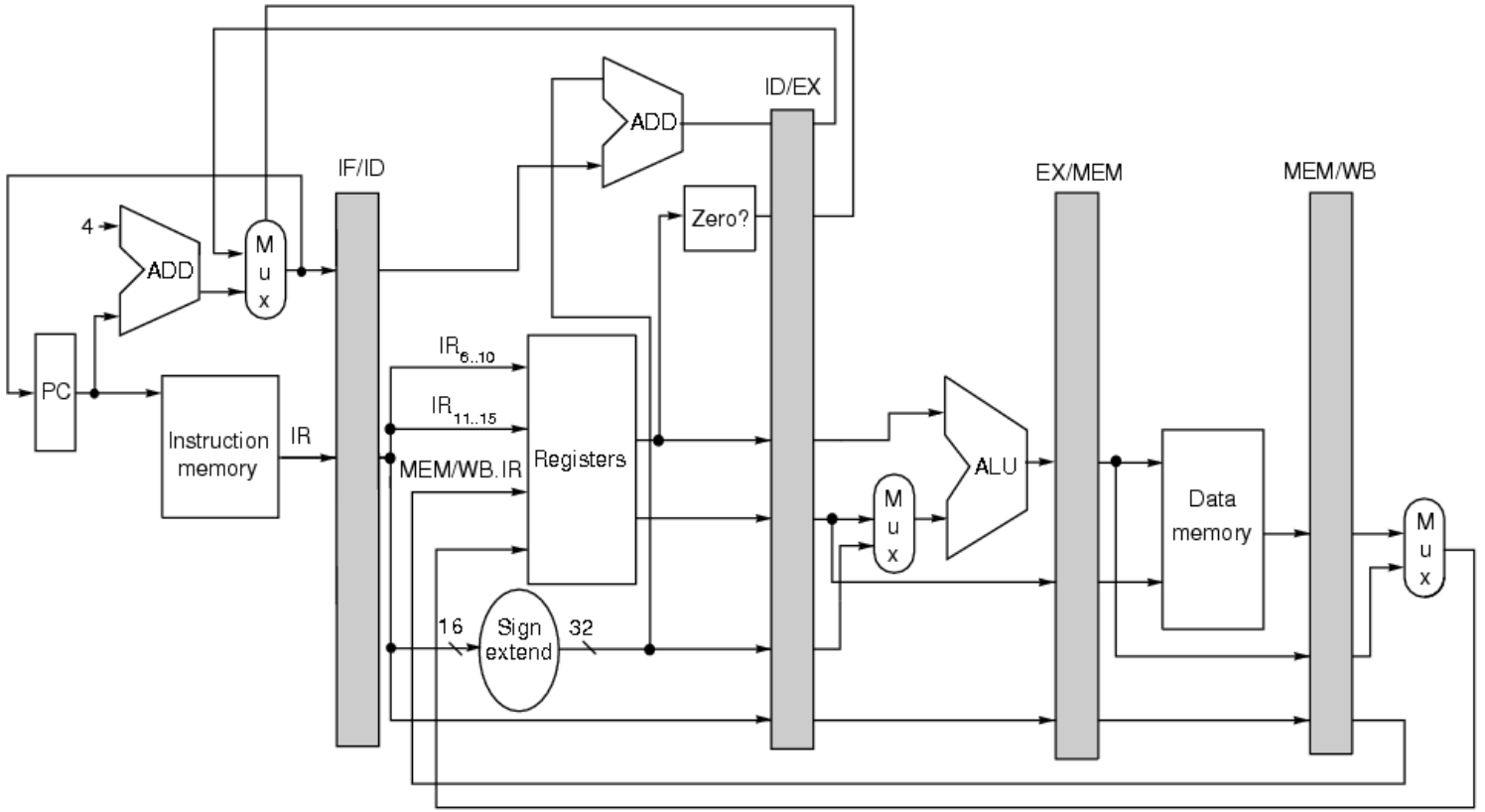
Descartando instruções



Instruções de Desvio no Pipeline



Instruções de Desvio no Pipeline



- Os atrasos devido às instruções de desvio podem ser reduzidos através da antecipação do cálculo do endereço alvo do desvio e do teste “Zero?” para a etapa de decodificação do pipeline (ID).