

Infraestrutura de Hardware

Revisão Pipeline, Superescalar e Multicores

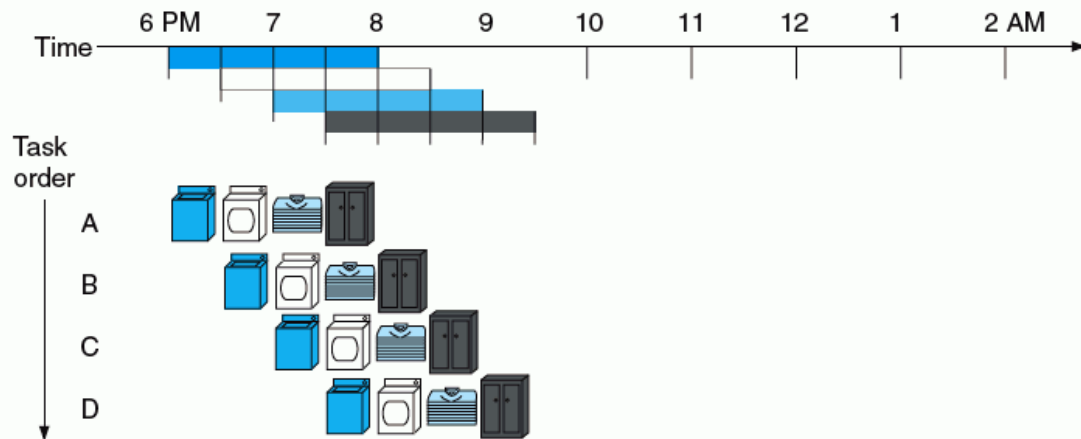
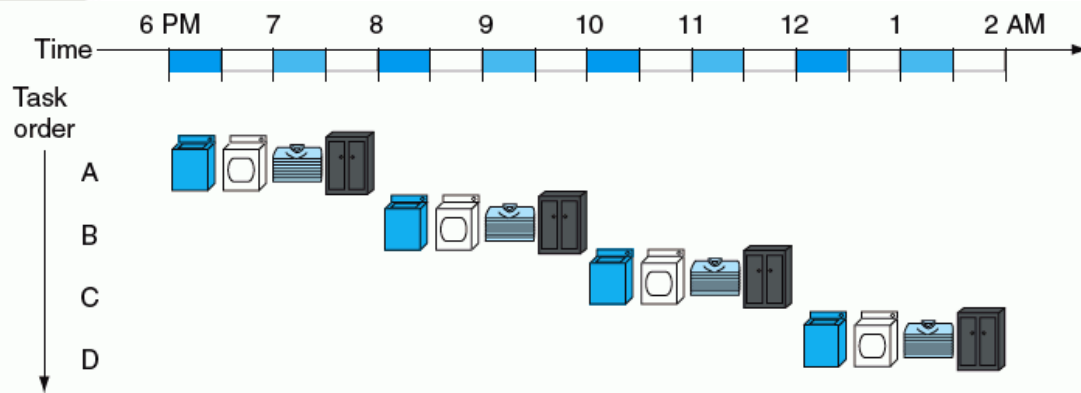


UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

Pipeline

- Pipeline é uma técnica que visa aumentar o nível de paralelismo de execução de instruções
 - ILP (Instruction-**L**evel **P**aralellism)
- Permite que várias instruções sejam processadas simultaneamente com cada parte do HW atuando numa instrução distinta
 - Maximiza uso do HW
 - Instruções quebradas em estágios
 - Sobreposição temporal
- Visa aumentar desempenho
 - Latência de instruções é a mesma ou maior
 - Throughput aumenta
- Tempo de execução de instrução é o mesmo ou maior, **MAS** tempo de execução de programa é menor

Lavanderia: Sequencial x Pipeline



- Para 4 lavagens de roupa:

Ganho de desempenho em $8/3,5 = 2,3x$

- Tempo de execução de uma tarefa é o mesmo

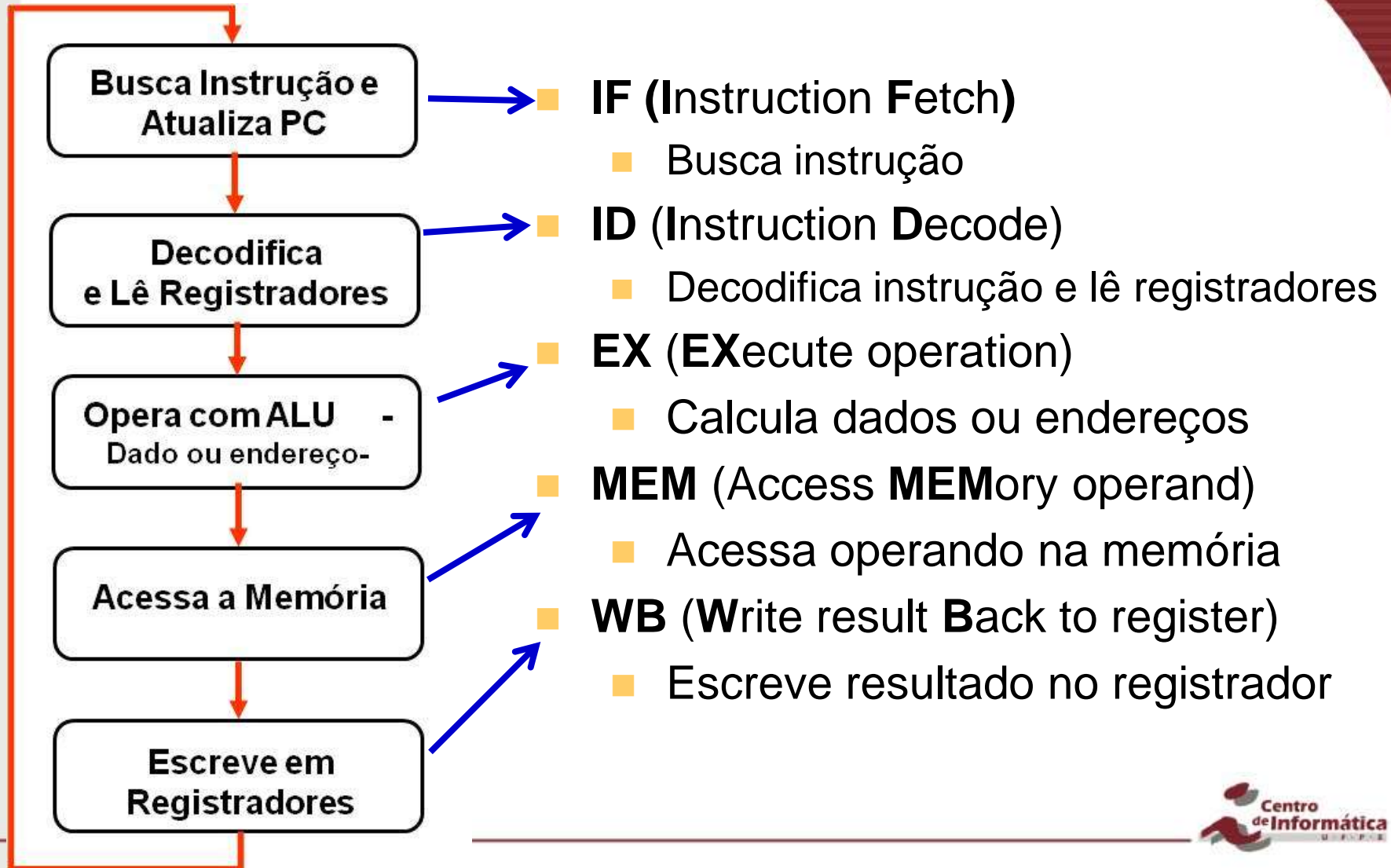
- Pipeline melhora o throughput

Tempo de execução de um conjunto de tarefas

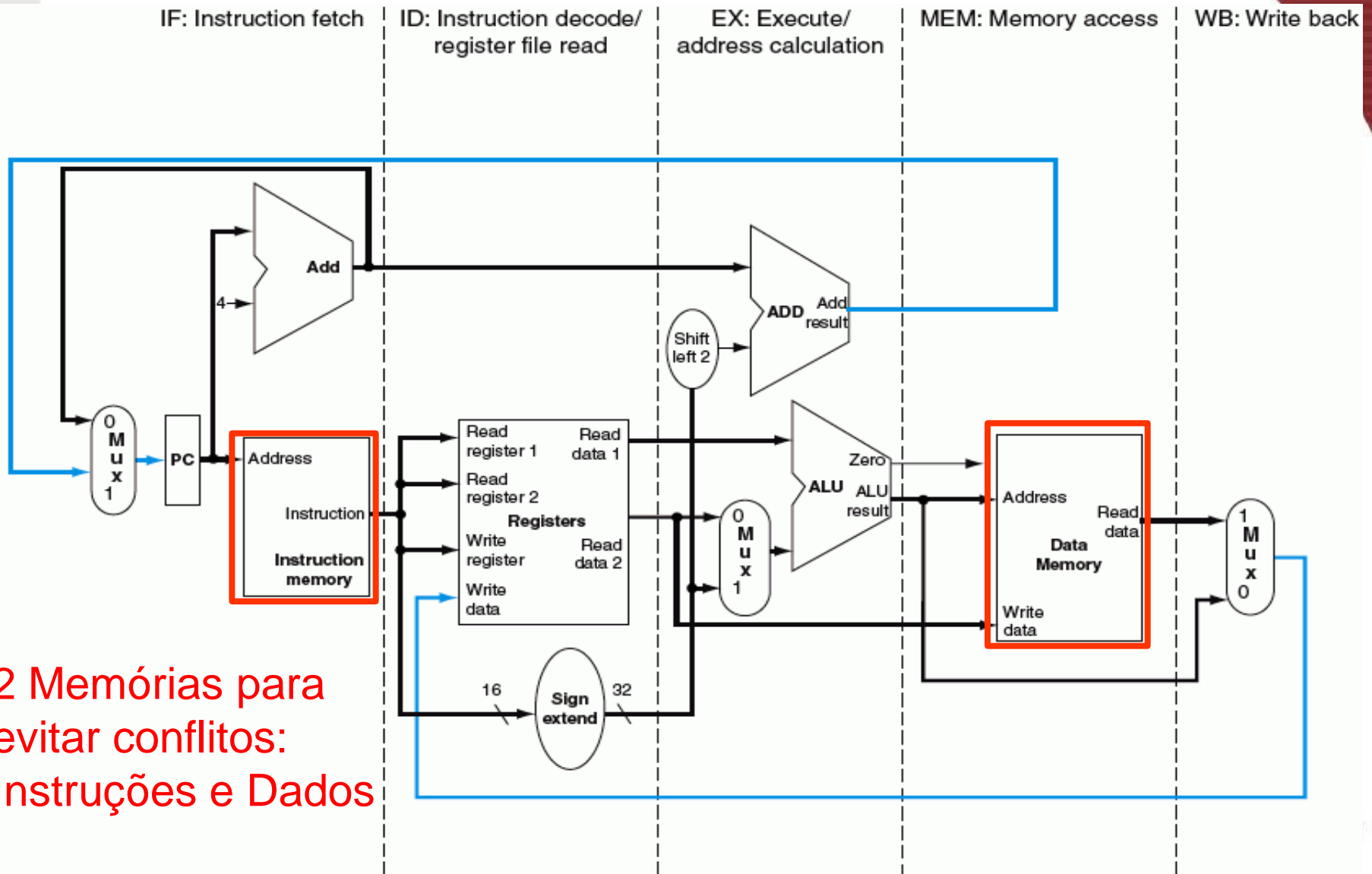
Mais sobre Pipeline...

- Melhora no throughput → melhora no desempenho
- Aumento do número de estágios do pipeline → Aumento de desempenho
 - Mais execuções paralelas
- Throughput é limitado pelo estágio mais lento do pipeline
 - Estágios devem ter a mesma duração
- Pode ocorrer dependências entre diferentes instâncias de execução, gerando espera
 - Reduz o desempenho

Estágios de uma Implementação Pipeline de um Processador



Implementação Pipeline do Datapath



Sentido dos Dados e Instruções em um Pipeline

IF: Instruction fetch

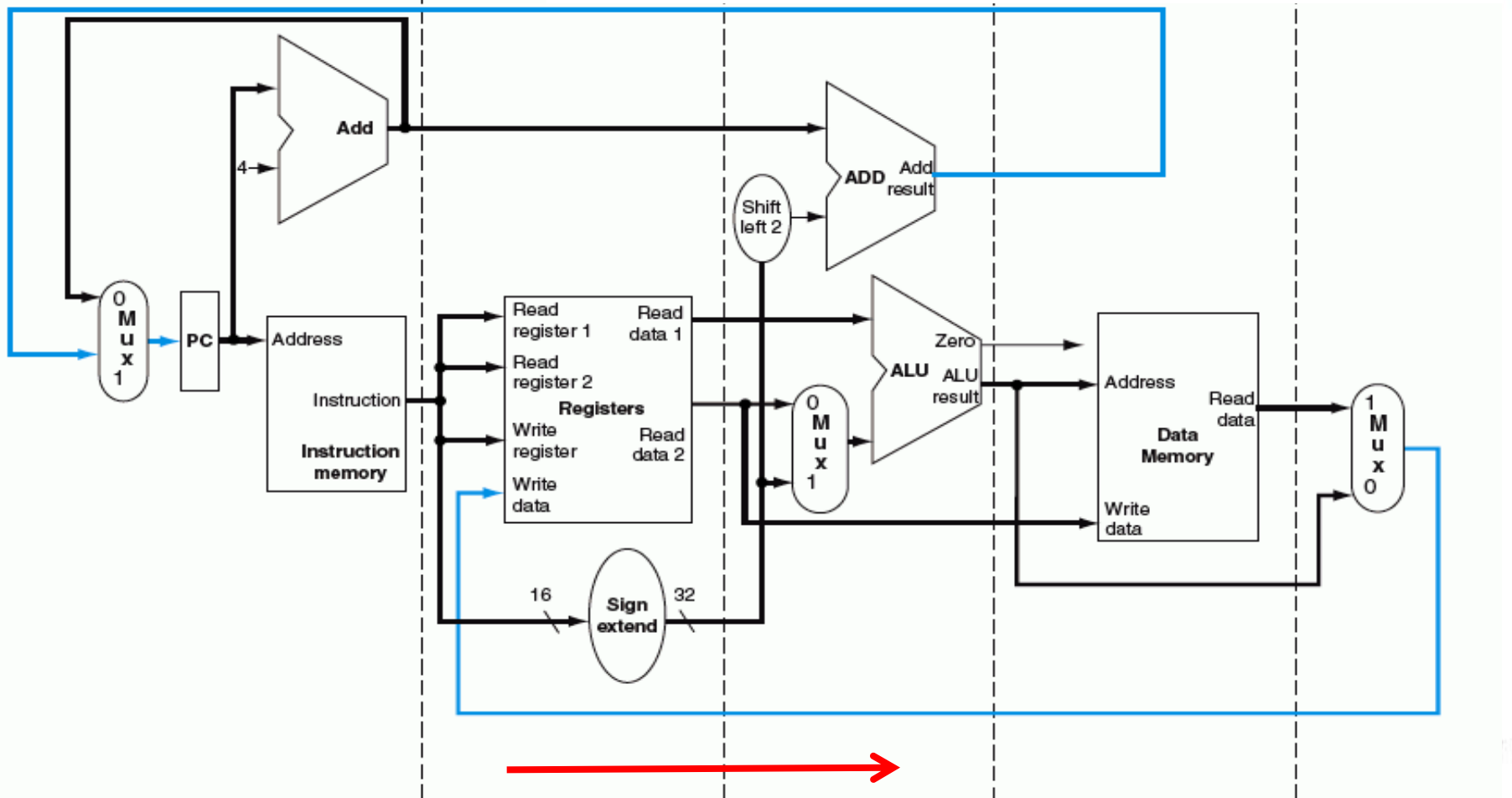
ID: Instruction decode/
register file read

EX: Execute/
address calculation

MEM: Memory access

WB: Write back

Dados e instruções se movem da esquerda para a direita, exceto as linhas azuis marcadas



Registradores do Pipeline

- São necessários registradores entre estágios
Para armazenar informação produzido no ciclo anterior

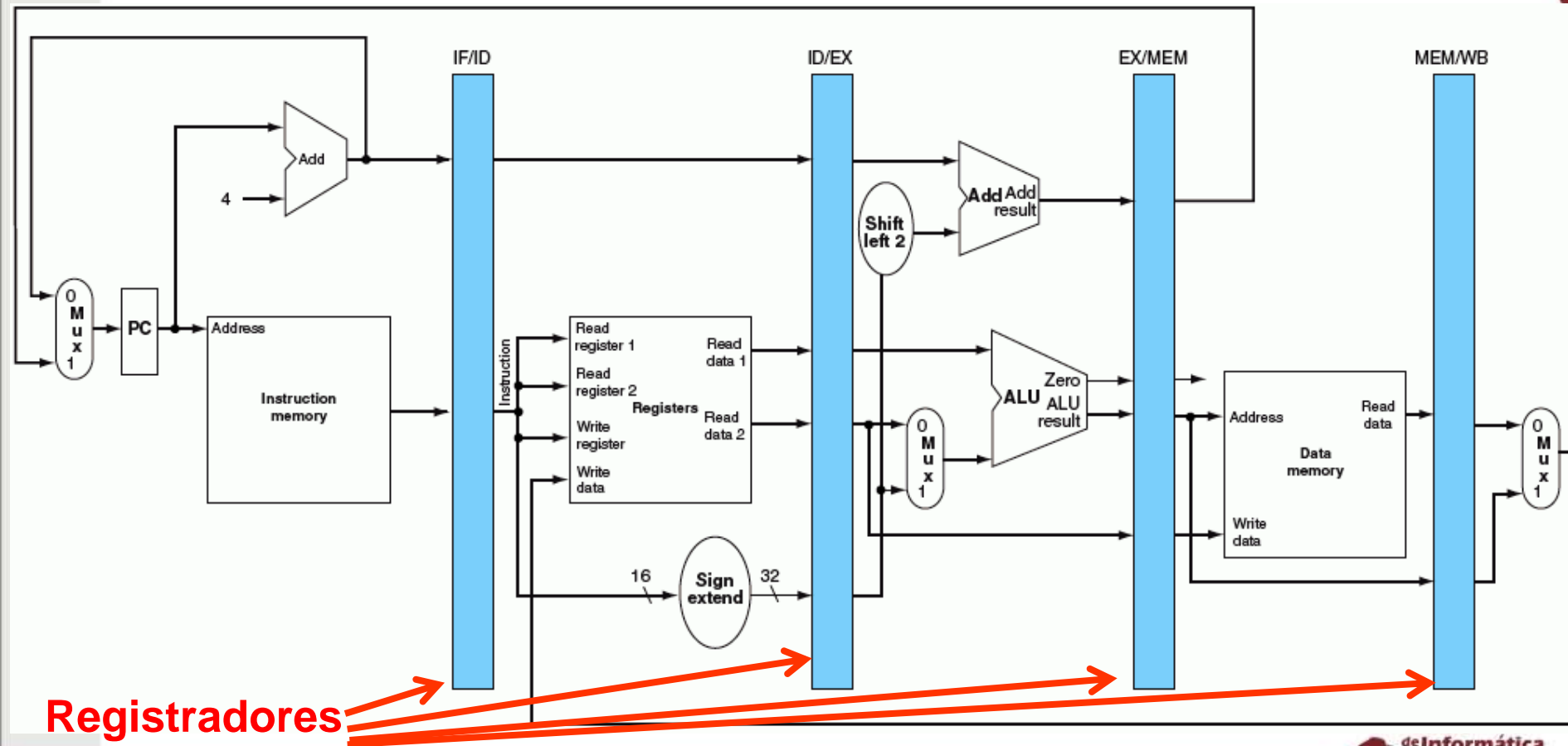
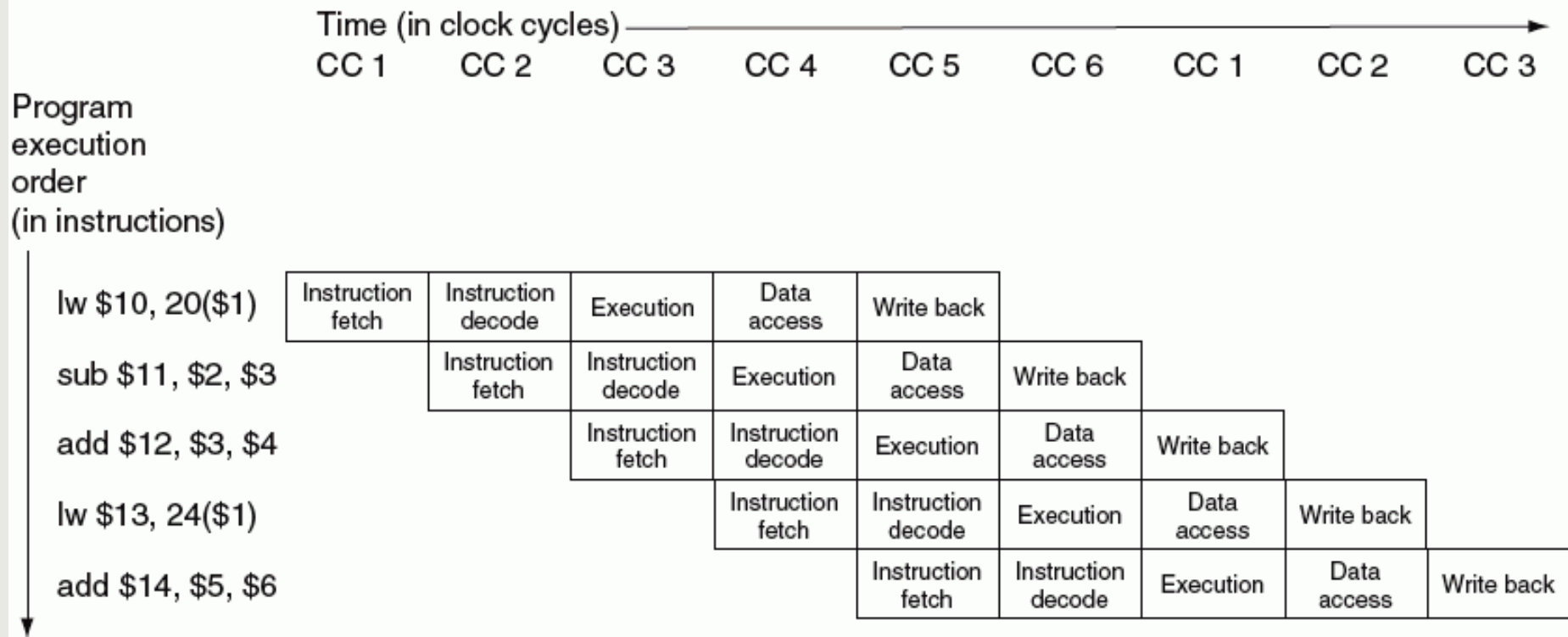
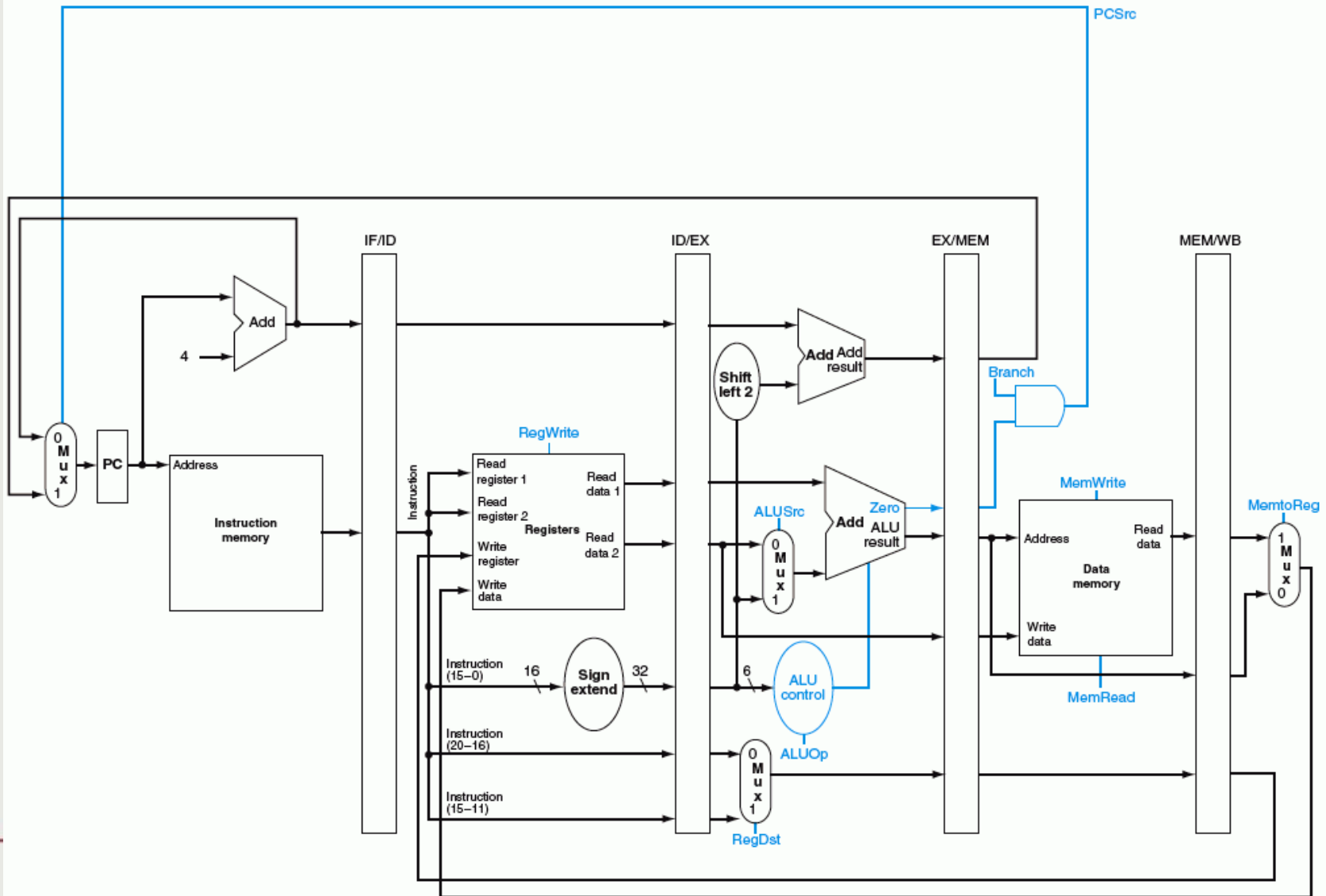


Diagrama de Múltiplos Ciclos de Clock

- Versão tradicional identifica cada estágio pelo nome



Implementando Pipeline: Sinais de Controle

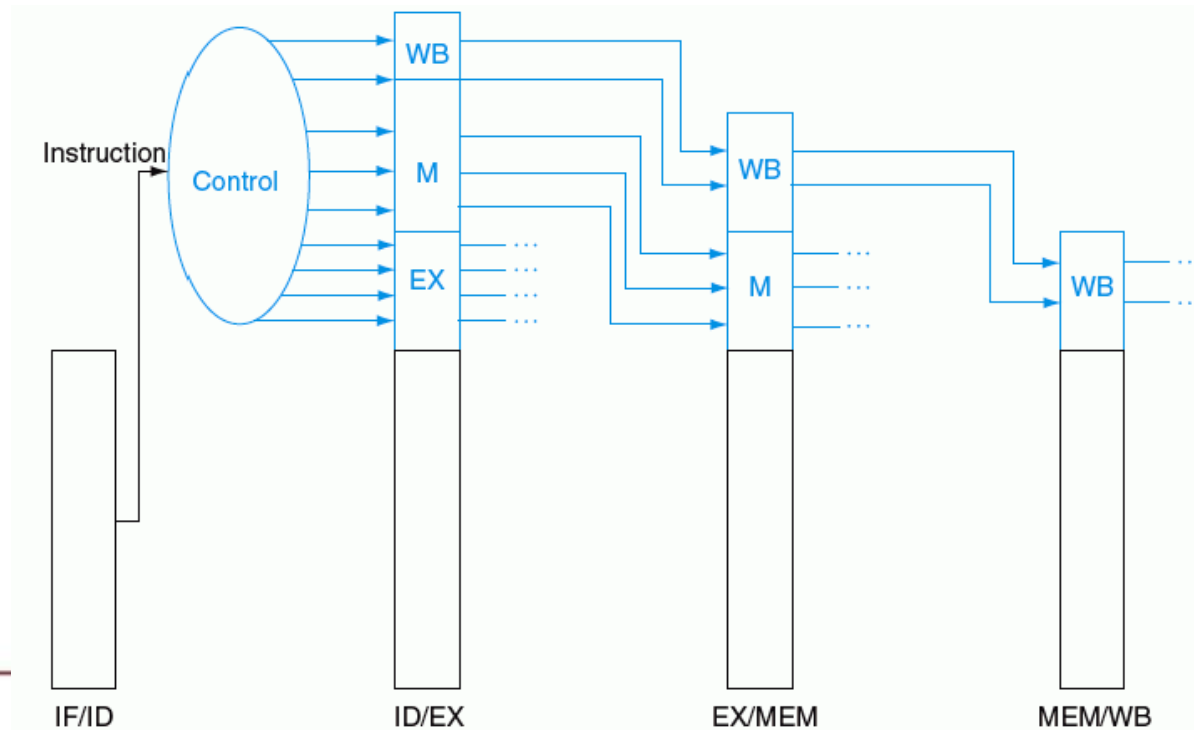


Como Controlar Cada Estágio?

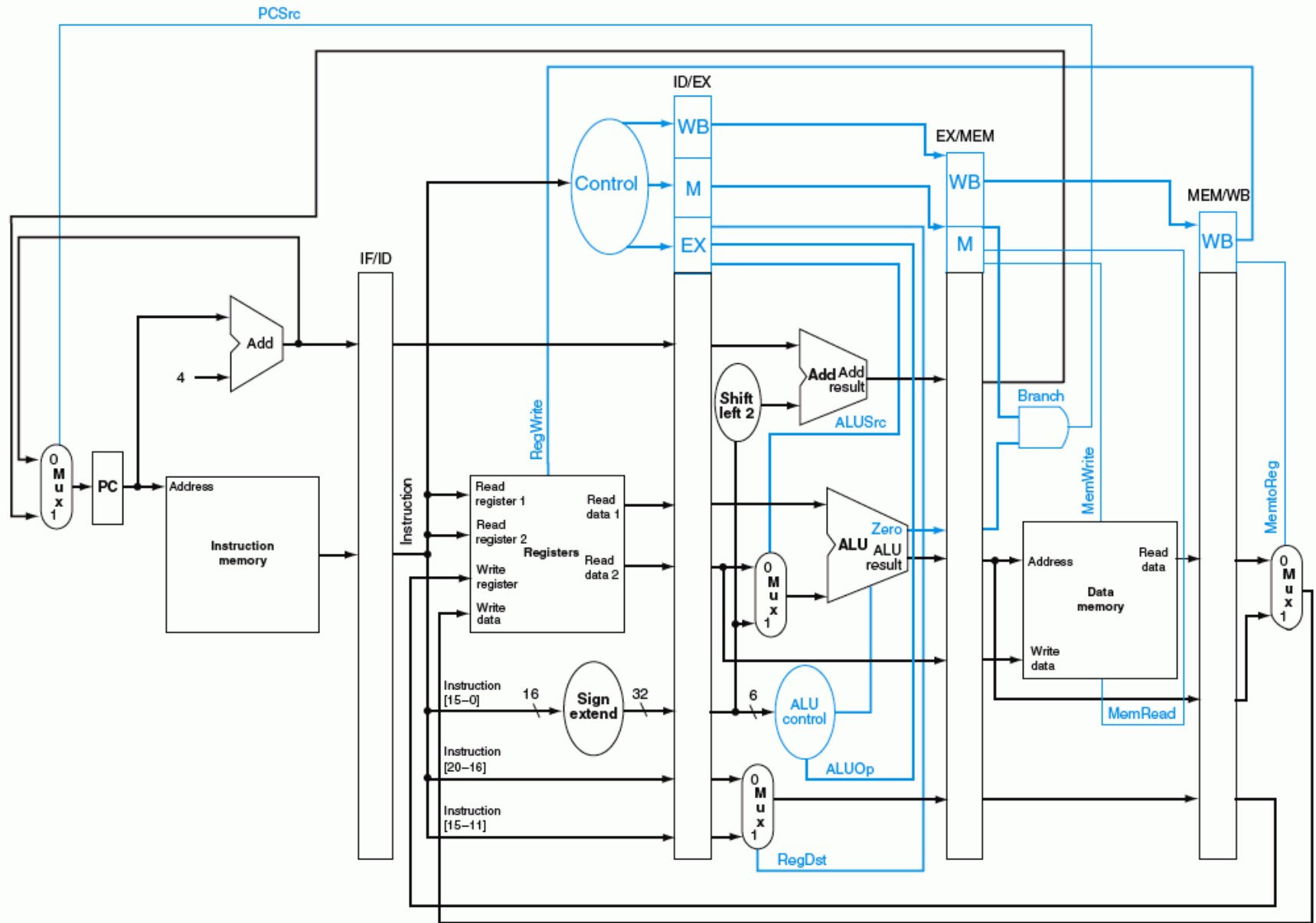
- **Sinais de controle são repassados aos registradores do pipeline**

Durante o estágio de decodificação, sinais de controle para o resto dos estágios podem ser gerados e armazenados

A cada ciclo do clock, o registrador corrente passa os sinais para o registrador do próximo estágio



Processador Pipeline Completo



Conflitos

- Situações que evitam que uma nova instrução seja iniciada no próximo ciclo
- Tipos:

Estruturais

- Recurso necessário para execução de uma instrução está ocupado

Dados

- Dependência de dados entre instruções

Controle

- Decisão da próxima instrução a ser executada depende de uma instrução anterior

Conflitos Estruturais

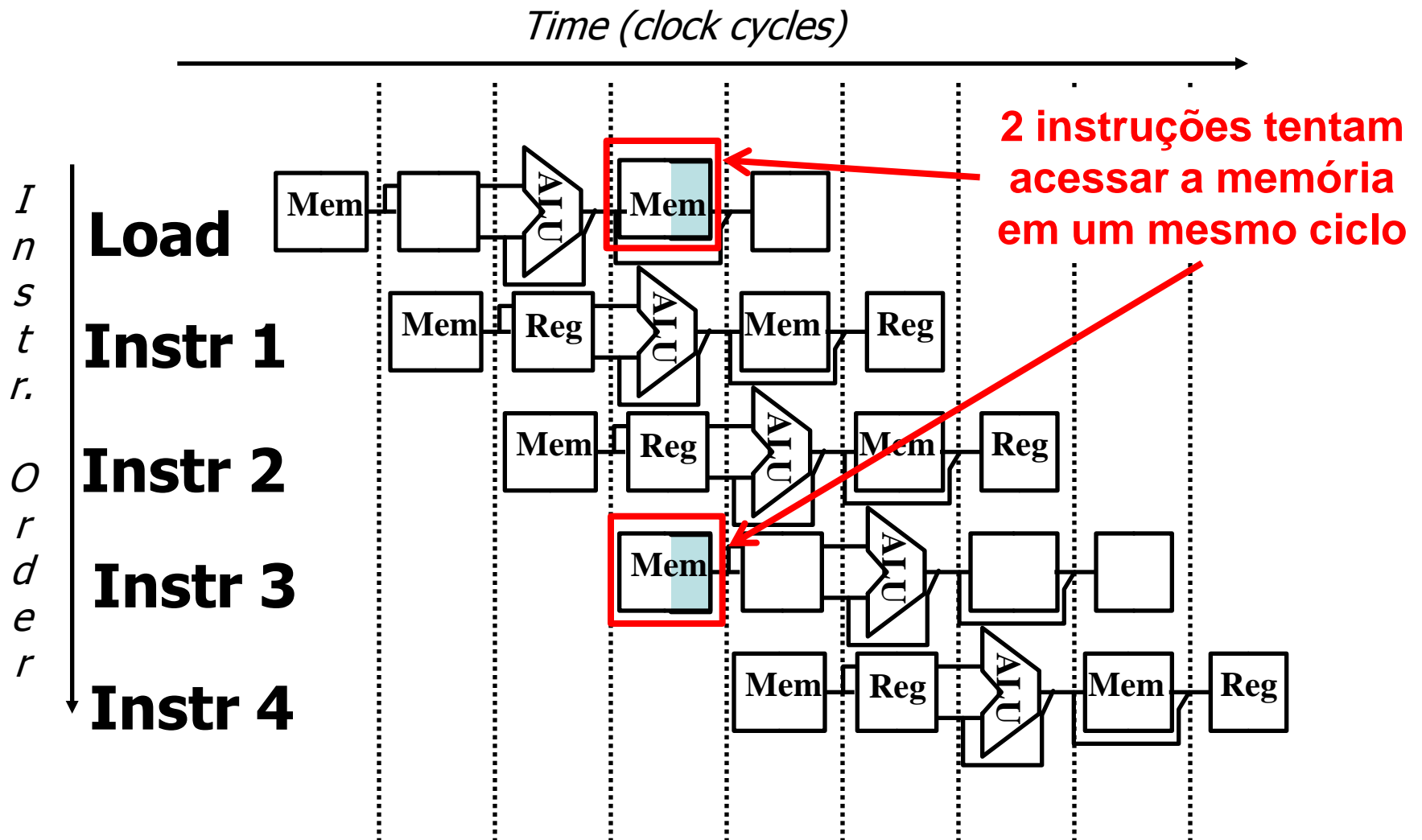
- Conflito pelo uso de um recurso
- Hardware não permite que determinadas combinações de instruções sejam executadas em um pipeline
- Utilização de uma só memória para dados e instruções é um exemplo

Load/store requer acesso a dados

Estágio de busca de instrução teria que esperar o load/store terminar o acesso a memória para começar

**Solução comum:
Replicar recursos**

Exemplo de Conflito Estrutural: Memória Única



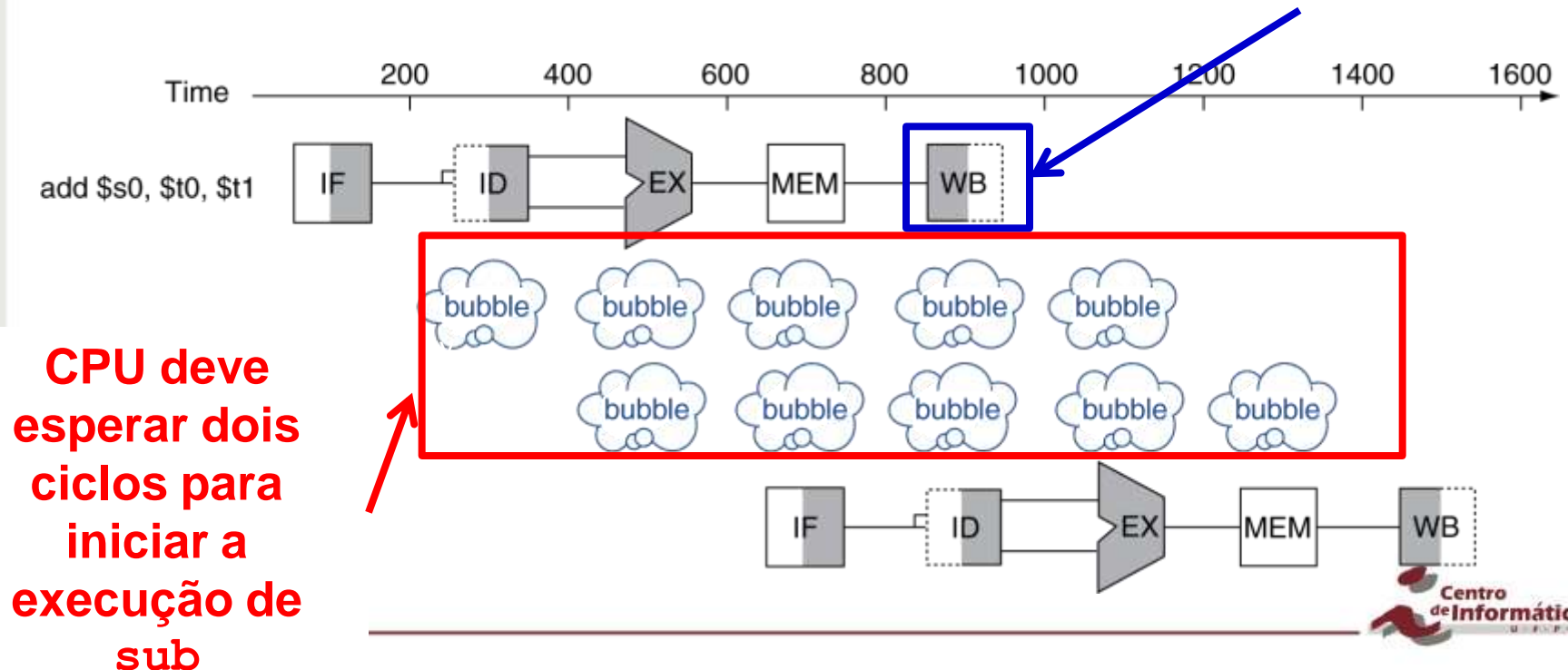
Conflito de Dados

- Uma instrução para ser executada depende de um dado gerado por uma instrução anterior

add **\$s0**, \$t0, \$t1

sub \$t2, **\$s0**, \$t3

**Resultado da soma
só será escrito em
\$s0 neste ciclo**



Resolvendo Conflitos de Dados

- Soluções em software (compilador/montador)
 - Inserção de NOPs
 - Re-arrumação de código
- Soluções em hardware
 - Método de Curto-circuito (Forwarding)
 - Inserção de retardos (stalls)

Inserção de NOPs no Código

- **Compilador/Montador deve identificar conflitos de dados e evitá-los inserindo NOPs no código**

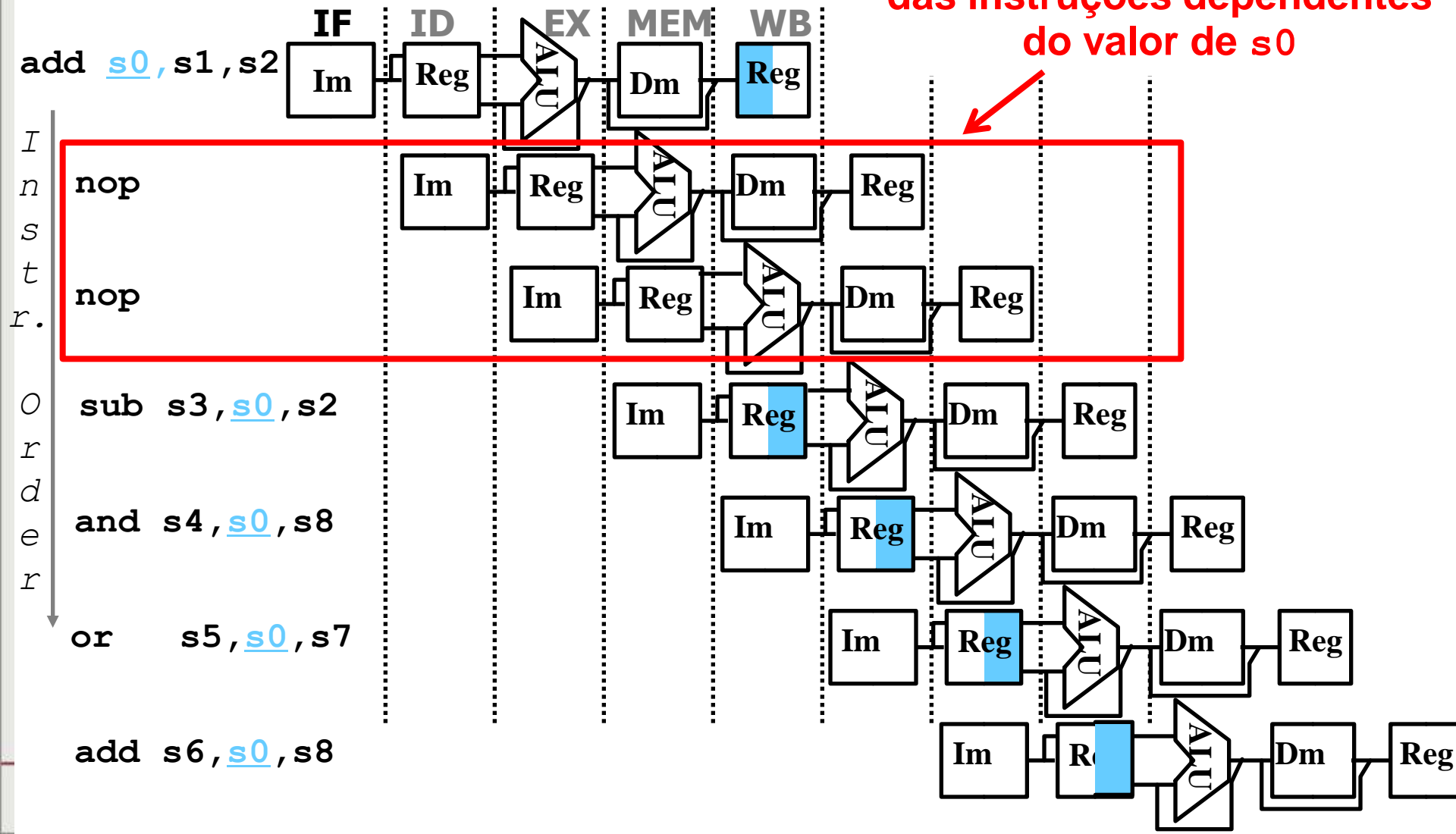
Conflitos de dados

```
add $s0, $s1, $s2
sub $s3, $s0, $s2
and $s4, $s0, $s8
or  $s5, $s0, $s7
add $s6, $s0, $s8
```



```
add $s0, $s1, $s2
nop
nop
sub $s3, $s0, $s2
and $s4, $s0, $s8
or  $s5, $s0, $s7
add $s6, $s0, $s8
```

NOPs retardam a execuções das instruções dependentes do valor de s0



Re-arrumação do Código

- **Compilador/Montador deve identificar conflitos de dados e evitá-los re-arrumando o código**

Executa instruções que não tem dependência de dados e que a ordem de execução não altera a corretude do programa

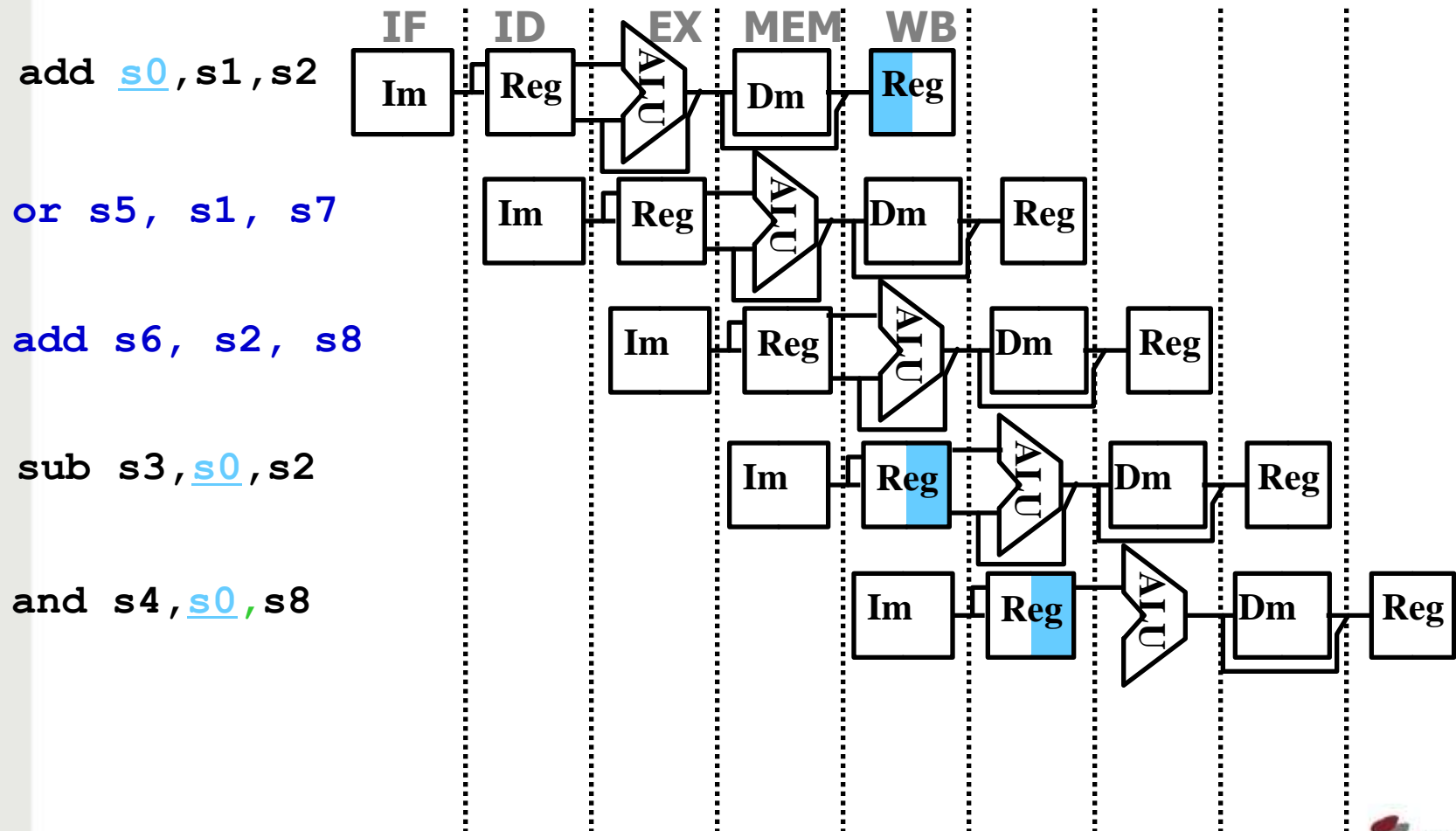
Conflitos de dados

```
add $s0, $s1, $s2
sub $s3, $s0, $s2
and $s4, $s0, $s8
or  $s5, $s1, $s7
add $s6, $s2, $s8
```



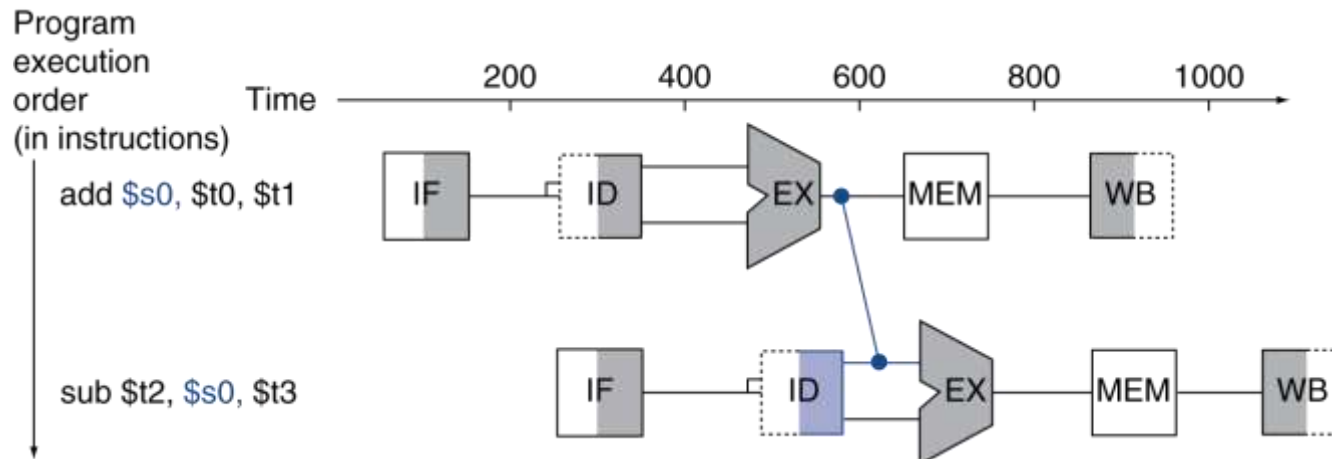
```
add $s0, $s1, $s2
or  $s5, $s1, $s7
add $s6, $s2, $s8
sub $s3, $s0, $s2
and $s4, $s0, $s8
```

Re-arrumação do Código

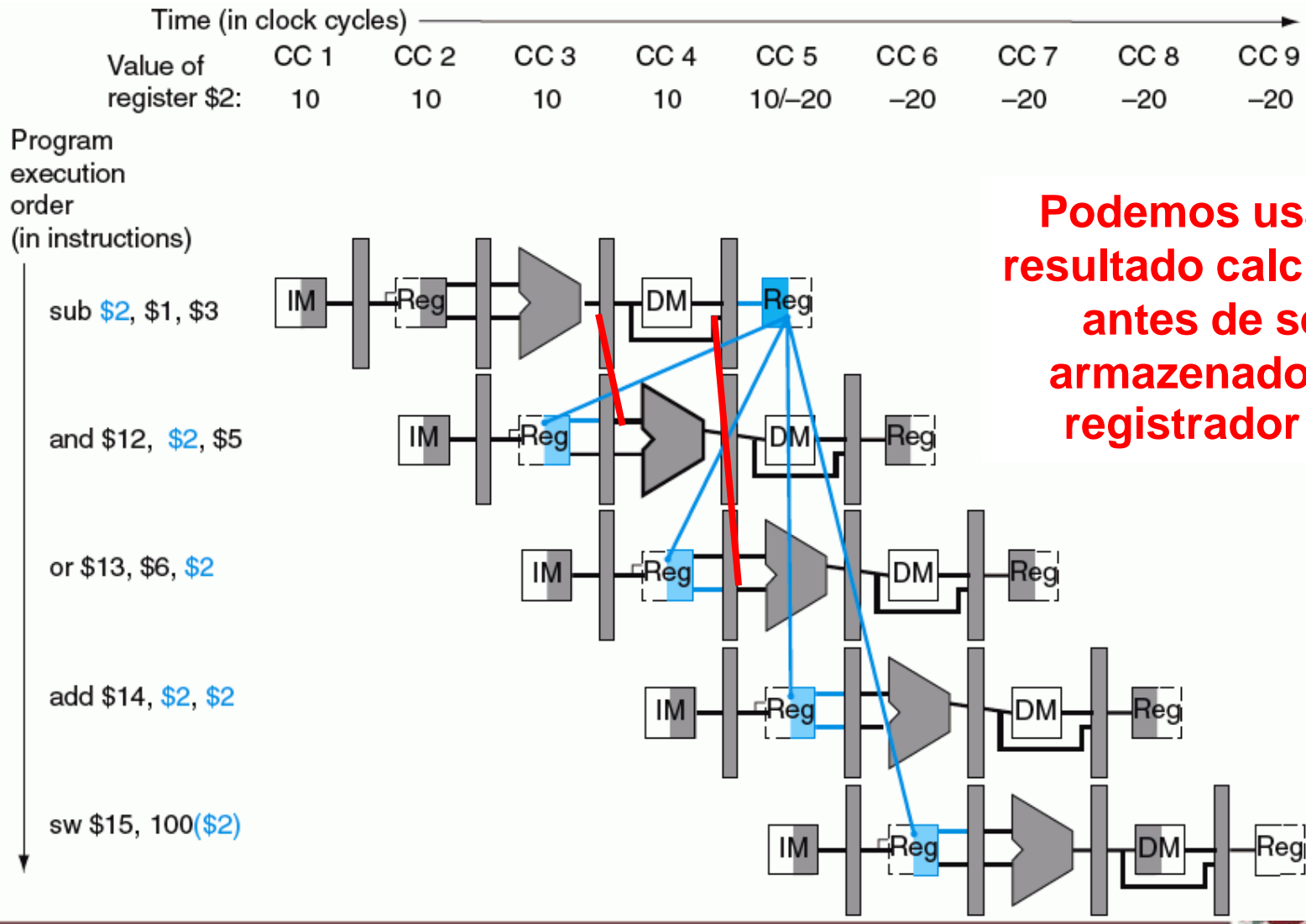


Método do Curto-Circuito (Forwarding ou Bypassing)

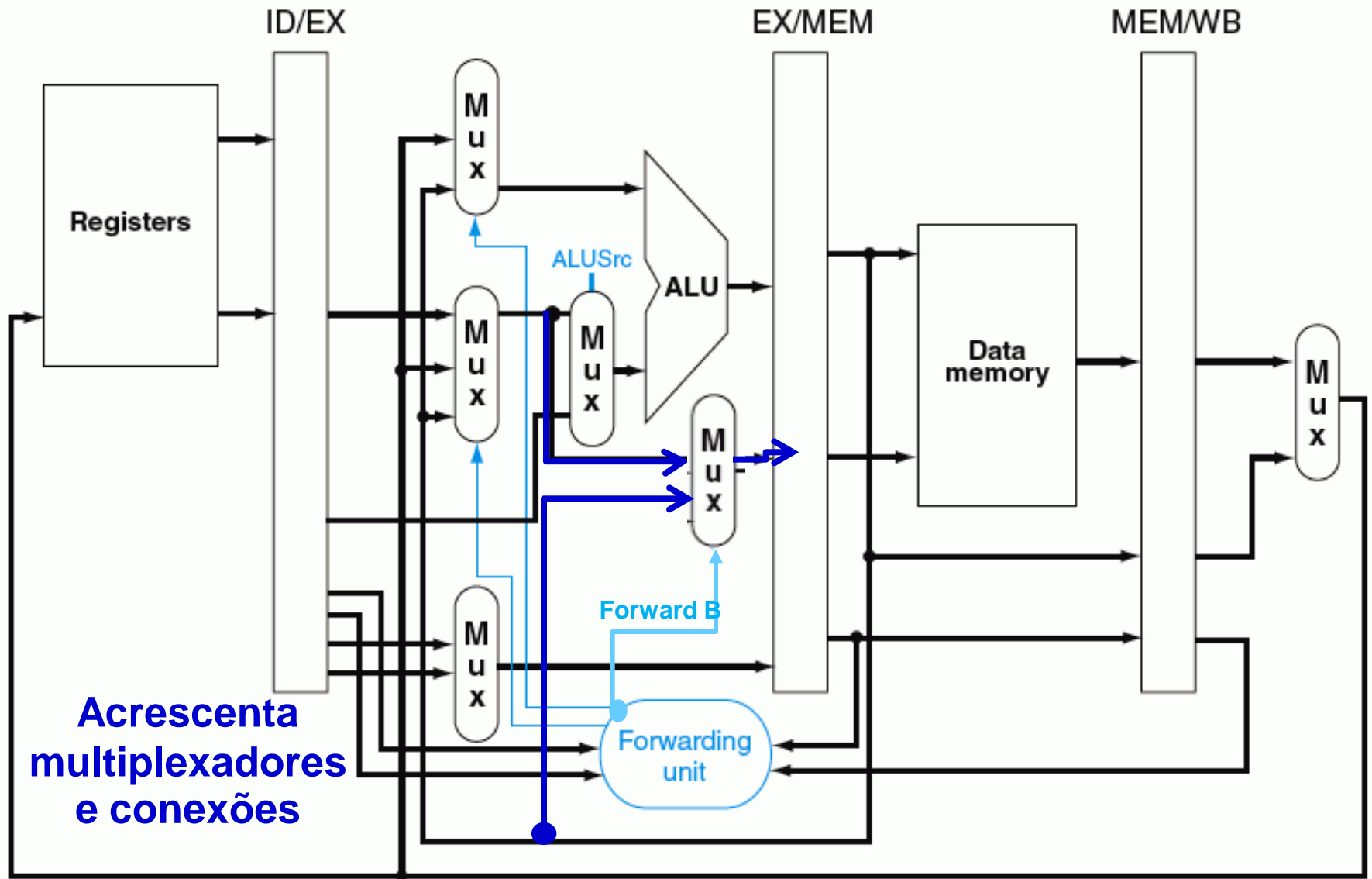
- **Usa o resultado desejado assim que é computado**
Não espera ser armazenado no registrador
Requer conexões extras na unidade de processamento



Dependências e Forwarding



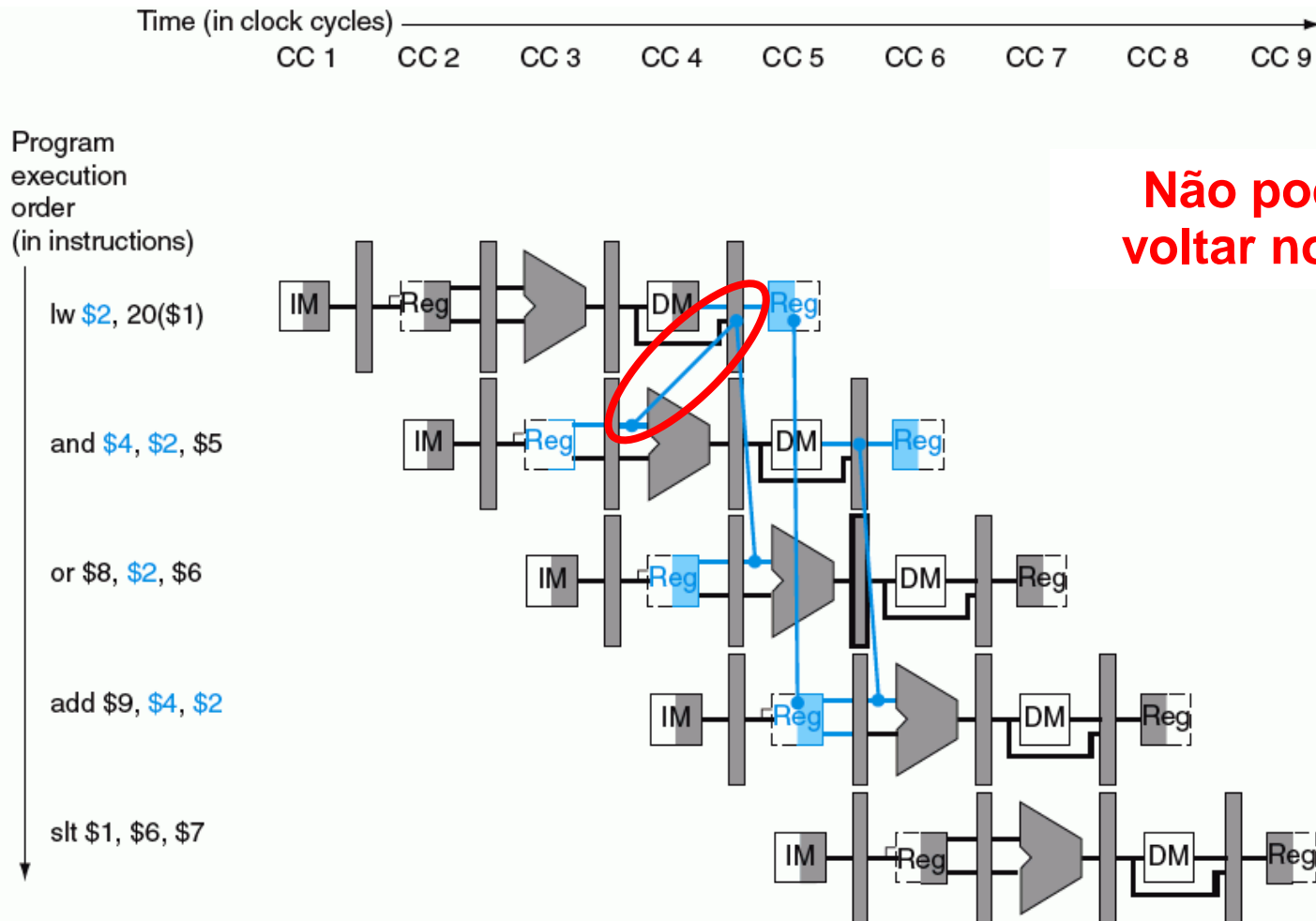
Datapath Simplificado Com Forwarding (Considerando store (sw) e Imediato)



Conflito de Dados Pelo Uso do Load

- Nem sempre se pode utilizar forwarding

Se o valor ainda não tiver sido computado quando necessário

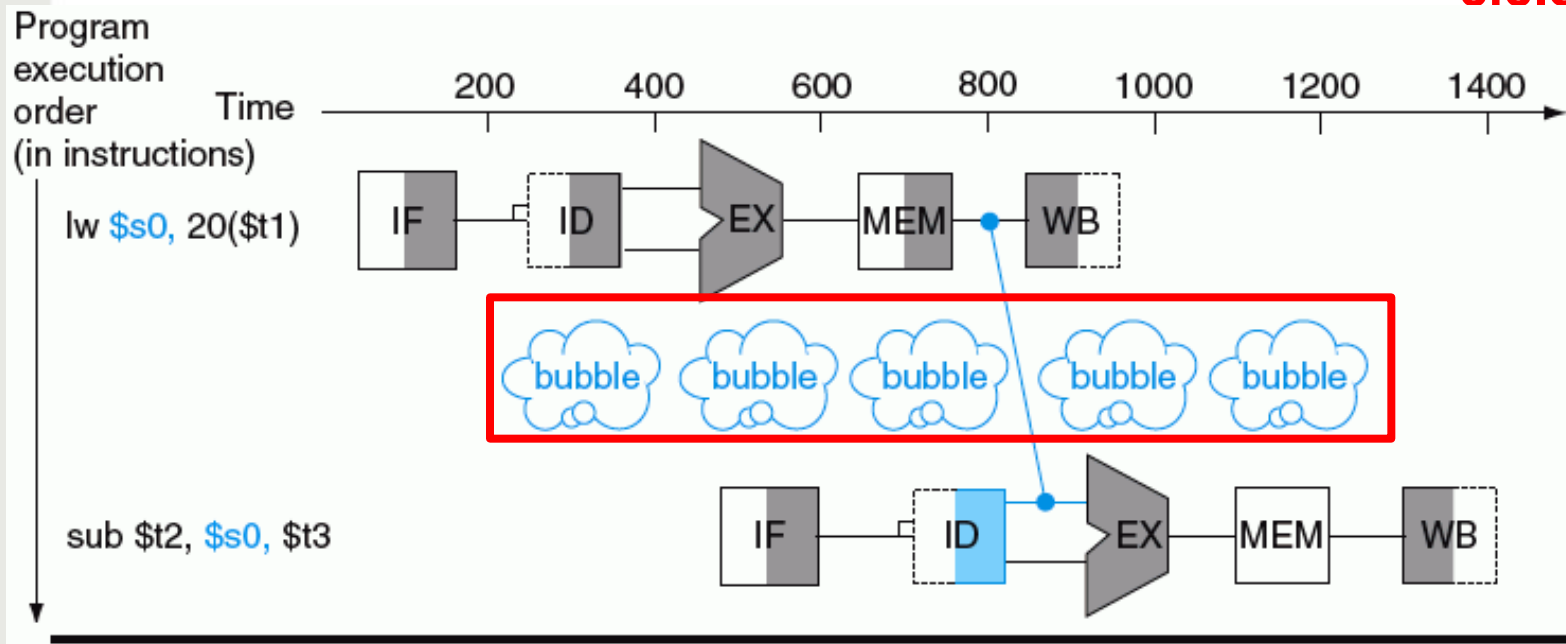


Não podemos voltar no tempo

Inserção de Retardos

- Quando não podemos utilizar forwarding para resolver conflitos, inserimos retardos

**Retarda a execução
da próxima
instrução em um
ciclo**



Como Detectar Este Tipo de Conflito de Dado?

- **Verificar se instrução depende do load no estágio ID**
- **Números dos registradores do operandos da ALU são dados por:**
IF/ID.RegisterRs, IF/ID.RegisterRt
- **Conflito acontece quando**
ID/EX.MemRead and
(ID/EX.RegisterRt = IF/ID.RegisterRs) or
(ID/EX.RegisterRt = IF/ID.RegisterRt)
- **Se detectado, insira um retardo**

Como Inserir Retardos em um Pipeline?

- **Forçar sinais de controle no registrador ID/EX para terem valor 0**

EX, MEM and WB

Instrução que depende do load se torna um nop

- **Não permitir a atualização do PC e do registrador IF/ID**

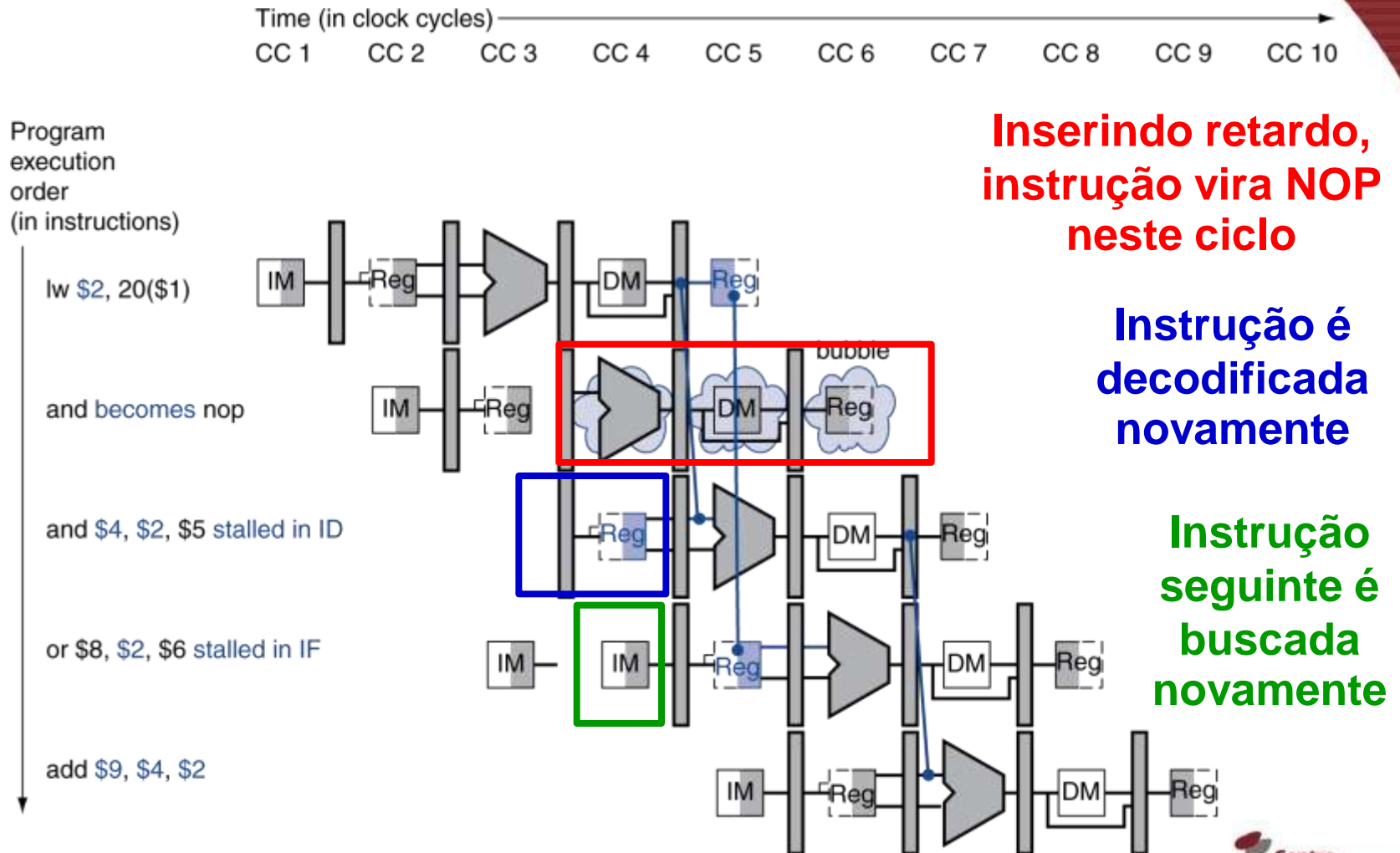
Instrução é decodificada de novo

Instrução seguinte é buscada novamente

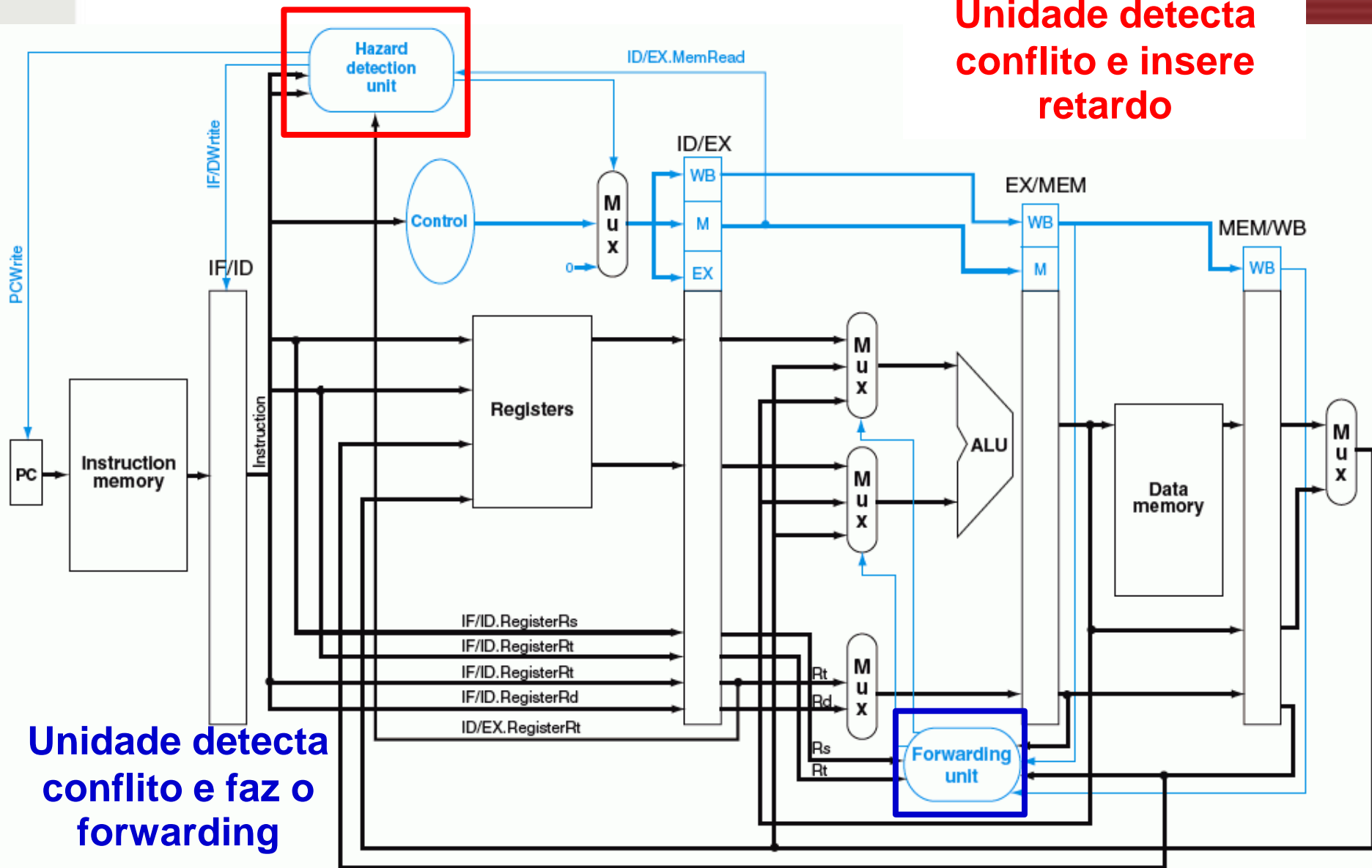
Retardo de 1 ciclo permite que MEM leia dado do load

- Depois se pode utilizar um forward do estágio EX

Inserindo um Retardo no Pipeline



Datapath com Unidades de Retardo e Forwarding



Unidade detecta conflito e insere retardo

Unidade detecta conflito e faz o forwarding

Exercício

(POSCOMP 2005 - 21) Considere uma CPU usando uma estrutura pipeline com 5 estágios (IF, ID, EX, MEM, WB) e com memórias de dados e de instruções separadas, sem mecanismo de data forwarding, escrita no banco de registradores na borda de subida do relógio e leitura na borda de descida do relógio e o conjunto de instruções a seguir:

I1: lw \$2, 100(\$5)

I2: add \$1, \$2, \$3

I3: sub \$3, \$2, \$1

I4: sw \$2, 50(\$1)

I5: add \$2, \$3, \$3

I6: sub \$2, \$2, \$4

- Quantos ciclos de relógio são gastos para a execução deste código?

Resposta do Exercício

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
I1	IF	ID	EX	ME	WB												
I2		IF	ID	X	X	EX	ME	WB									
I3			IF	X	X	ID	X	X	EX	ME	WB						
I4						IF	X	X	ID	EX	ME	WB					
I5									IF	ID	X	EX	ME	WB			
I6										IF	X	ID	X	X	EX	ME	WB
I7																	

I1: lw \$2, 100(\$5)

I2: add \$1, \$2, \$3

I3: sub \$3, \$2, \$1

I4: sw \$2, 50(\$1)

I5: add \$2, \$3, \$3

I6: sub \$2, \$2, \$4

17 ciclos

Resposta do Exercício (Com Forward)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
I1	IF	ID	EX	ME	WB												
I2		IF	ID	X	EX	ME	WB										
I3			IF	X	ID	EX	ME	WB									
I4					IF	ID	EX	ME	WB								
I5						IF	ID	EX	ME	WB							
I6							IF	ID	EX	ME	WB						
I7																	

I1: lw \$2, 100(\$5)

I2: add \$1, \$2, \$3

I3: sub \$3, \$2, \$1

I4: sw \$2, 50(\$1)

I5: add \$2, \$3, \$3

I6: sub \$2, \$2, \$4

11 ciclos

Conflitos de Controle

- **Causados por alteração de fluxo de controle**

- Desvios, chamadas e retorno de subrotinas

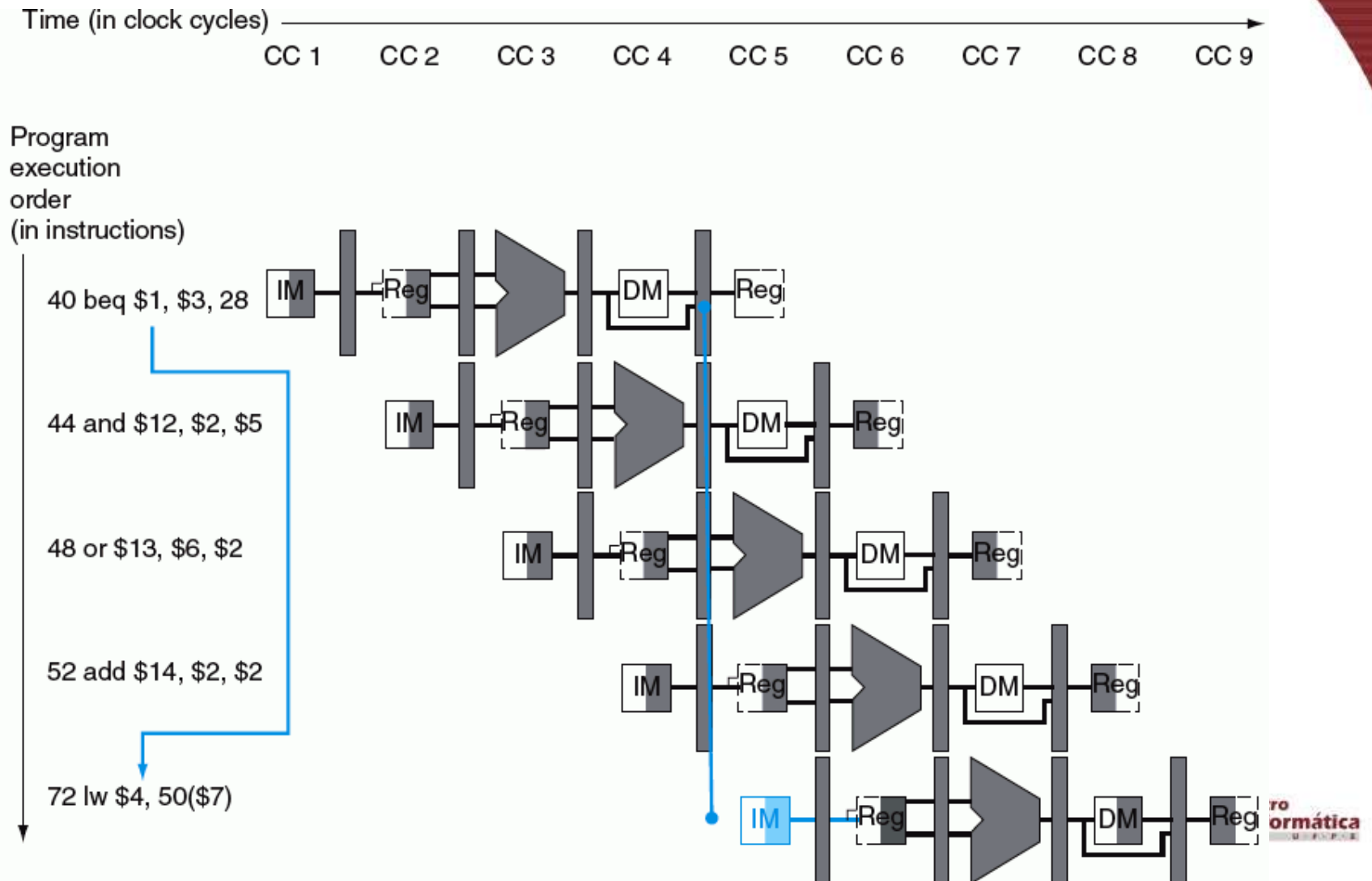
- Busca de nova instrução depende do resultado da instrução anterior

- Pipeline nem sempre pode buscar a instrução correta

- Pois instrução que altera fluxo de controle ainda está no estágio de ID

- **Como resolver conflito minimizando perda de desempenho?**

Exemplo de Conflito de Controle



Resolvendo Conflitos de Controle

- Soluções em software (compilador/montador)

 - Re-arrumação de código

 - Desvio com efeito retardado

- Soluções em hardware

 - Congelamento do pipeline

 - Execução especulativa

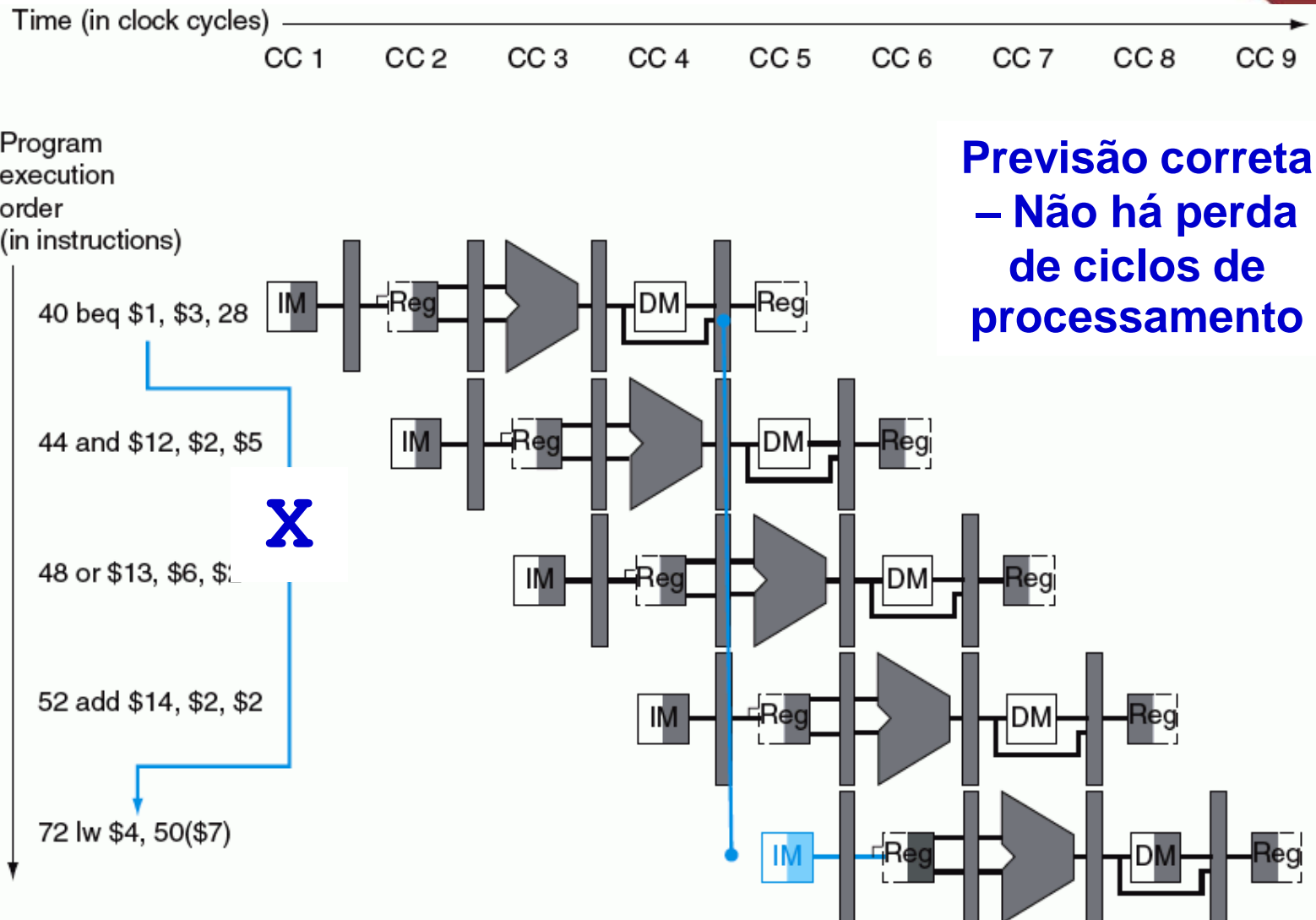
 - Estática e Dinâmica

 - Aceleração de avaliação de desvio

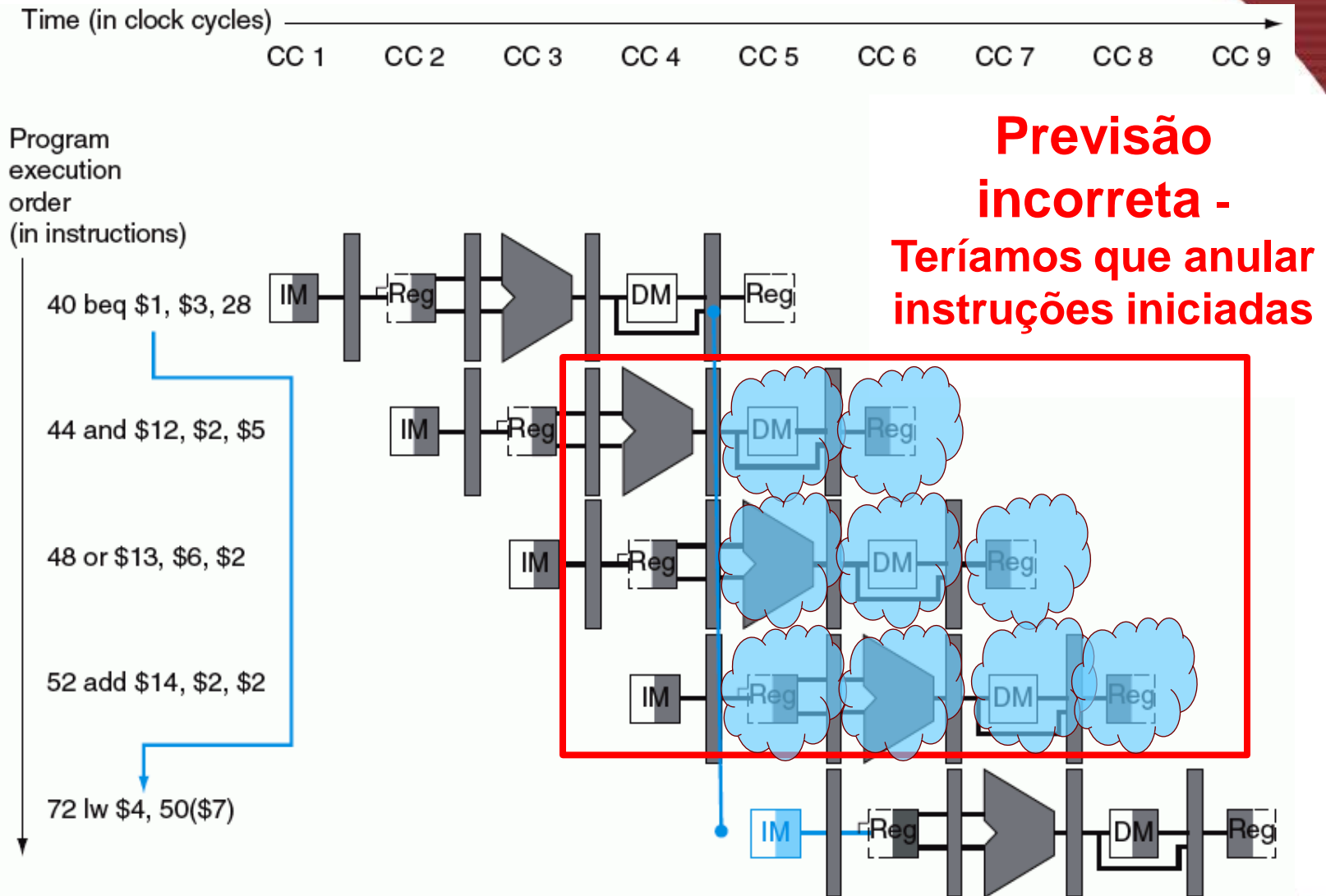
Execução Especulativa

- Congelamento em pipelines mais longos provocam uma perda de desempenho inaceitável
- **Um método para tratar conflitos de controle é especular qual será o resultado da instrução de desvio (Execução especulativa)**
 - Só congela se a previsão for errada
- Previsão pode ser:
 - Estática
 - Nunca haverá desvio, ou sempre haverá desvio
 - Dinâmica
 - De acordo com comportamento do código

Previsão Estática – Nunca Haverá Desvio



Previsão Estática – Nunca Haverá Desvio



Previsão Estática – Nunca Haverá Desvio

- Previsão errada obriga processador a anular **(flush)** instruções executadas erroneamente

Zera os sinais de controle relativos às instruções executadas erroneamente

Tem-se que fazer isto para instruções que estão nos estágios IF, ID e EX

- Branch só é corretamente avaliado depois de EX

Exercício

Considere uma CPU usando uma estrutura pipeline com 5 estágios (IF, ID, EX, MEM, WB) e com memórias de dados e de instruções separadas, com mecanismo de data forwarding, com previsão estática de que o desvio não se confirmará e o conjunto de instruções a seguir:

I1: addi \$2, \$zero, 1

I2: bne \$2,\$zero, I7

I3: sub \$3, \$2, \$1

I4: sw \$2, 50(\$1)

I5: add \$2, \$3, \$3

I6: sub \$2, \$2, \$4

I7: lw \$4, 0 (\$1)

- Quantos ciclos de relógio são gastos para a execução deste código?

Resposta do Exercício

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
I1	IF	ID	EX	ME	WB												
I2		IF	ID	EX	ME	WB											
I3			IF	ID	EX	F	F										
I4				IF	ID	F	F	F									
I5					IF	F	F	F	F								
I7						IF	ID	EX	ME	WB							
I8																	

I1: addi \$2, \$zero, 1
I2: bne \$2,\$zero, I7
I3: sub \$3, \$2, \$1
I4: sw \$2, 50(\$1)
I5: add \$2, \$3, \$3
I6: sub \$2, \$2, \$4
I7: lw \$4, 0 (\$1)

10 ciclos

Previsão Dinâmica

■ Hardware mede comportamento dos desvios

Registra a história recente de todos os desvios em uma tabela

Assume que o comportamento futuro dos desvios continuará o mesmo

Quando errado, anula instruções executadas erroneamente (coloca os valores de controle para zero), busca as instruções corretas e atualiza a história do comportamento do desvio

Acelerando Avaliação do Desvio

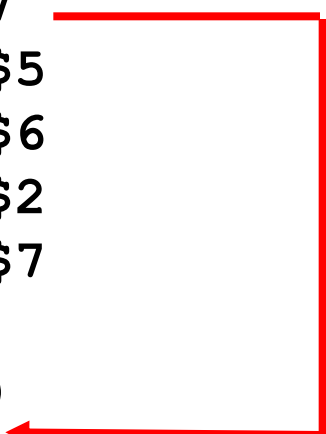
- Uma forma de acelerar a avaliação do desvio é colocar hardware que calcula resultado do desvio no estágio ID

Somador para endereço destino

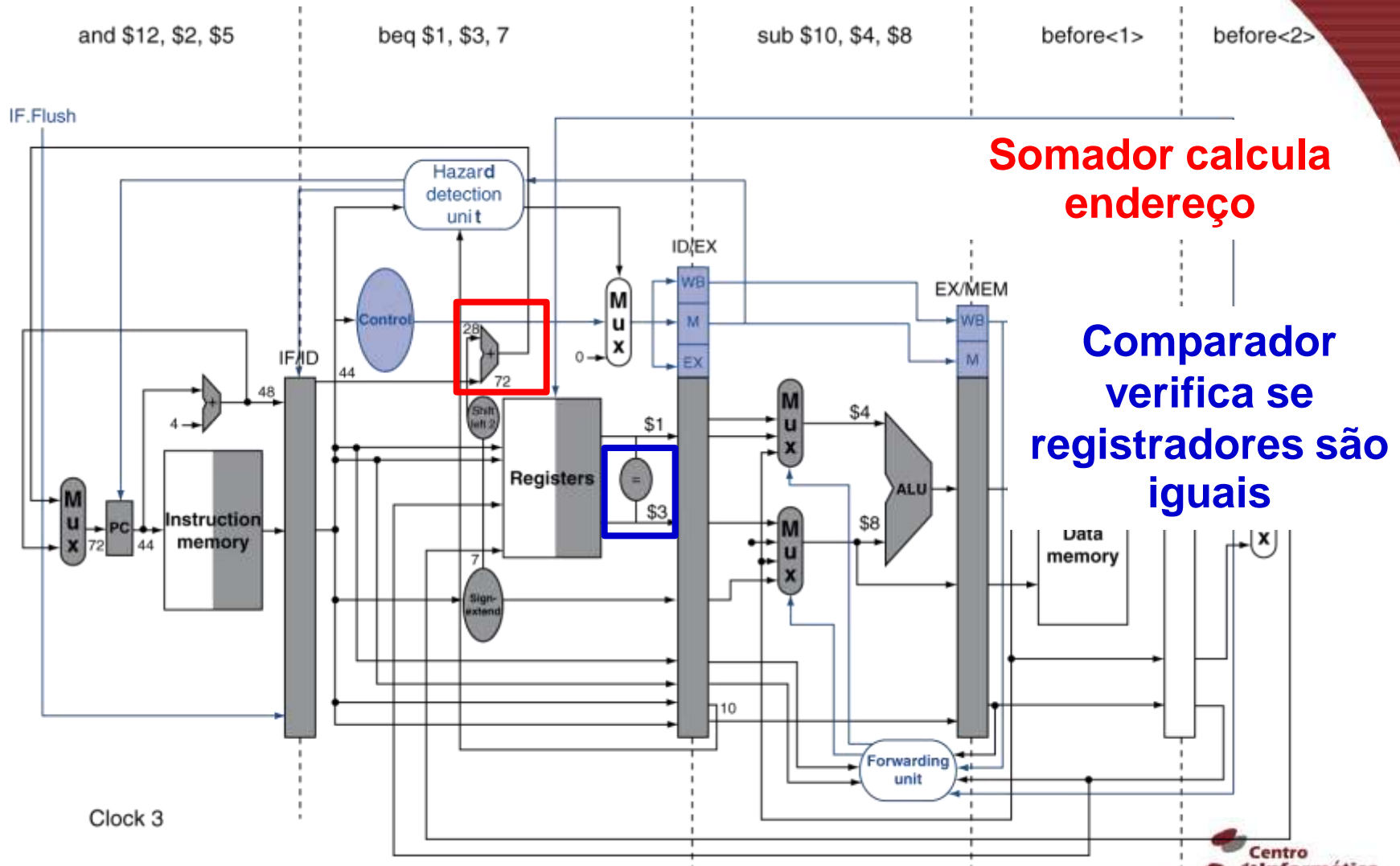
Comparador de registradores

- **Exemplo: desvio confirmado**

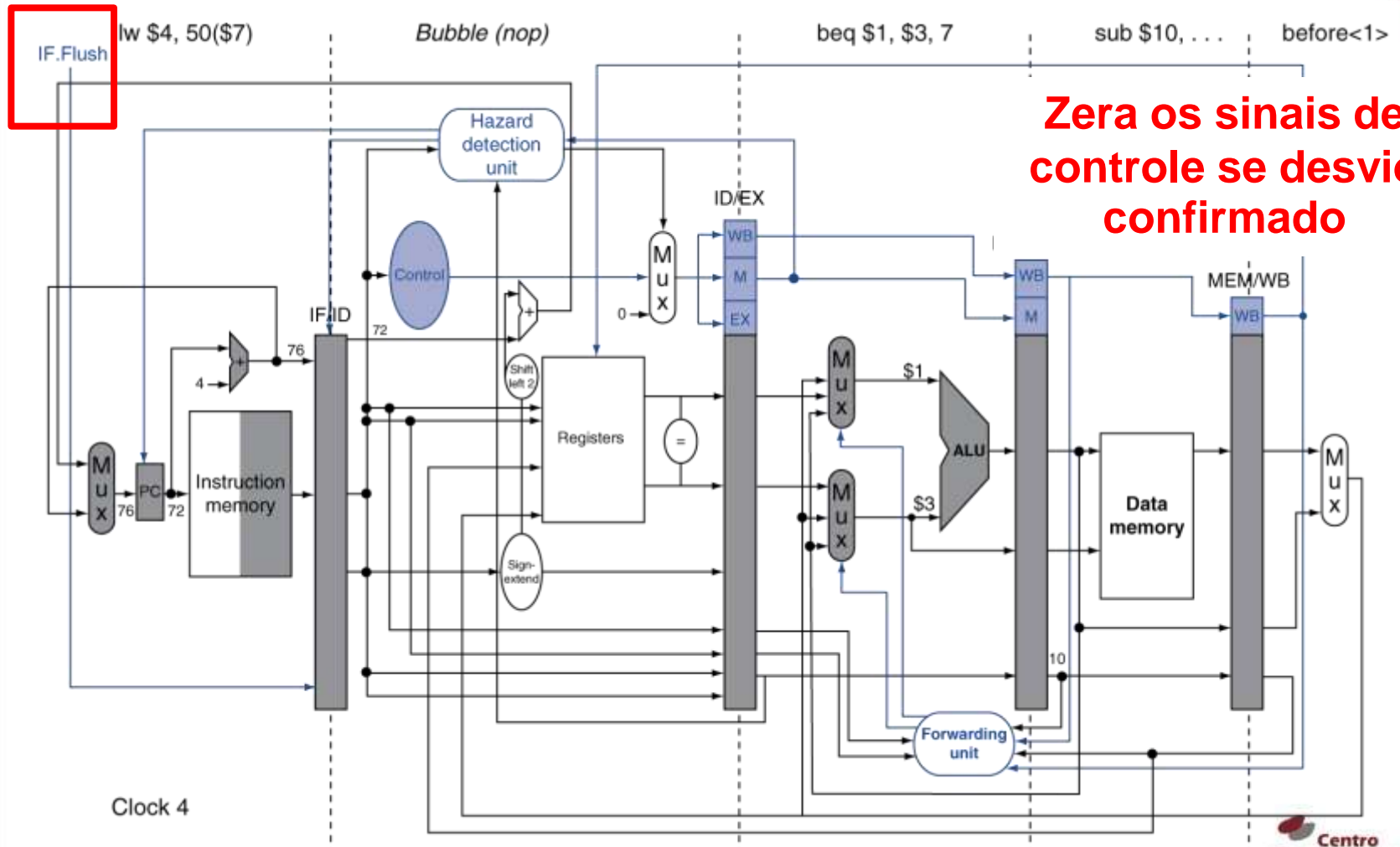
```
36:  sub  $10, $4, $8
40:  beq  $1,  $3, 7
44:  and  $12, $2, $5
48:  or   $13, $2, $6
52:  add  $14, $4, $2
56:  slt  $15, $6, $7
    ...
72:  lw   $4, 50($7)
```



Exemplo: Desvio Confirmado



Exemplo: Desvio Confirmado



Zera os sinais de controle se desvio confirmado

Técnica de SW para Resolver Conflitos de Controle

■ Desvio com efeito retardado (**Delayed branch**)

Consiste em re-arrumar o código de modo que enquanto a avaliação do desvio está sendo executada, uma nova instrução possa ser executada

Assembler re-arruma código

Arquitetura do MIPS dá suporte a execução de uma instrução enquanto o branch é avaliado

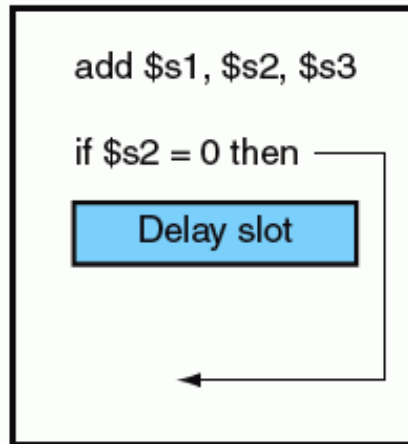
- Quantidade de instruções executadas durante a avaliação do branch é conhecida **delay slot**

Idealmente, instrução executada é independente do desvio

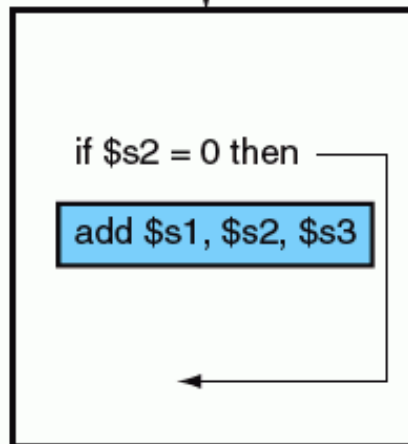
Desvio com Efeito Retardado

**Situação ideal:
instrução
independente**

a. From before



Becomes

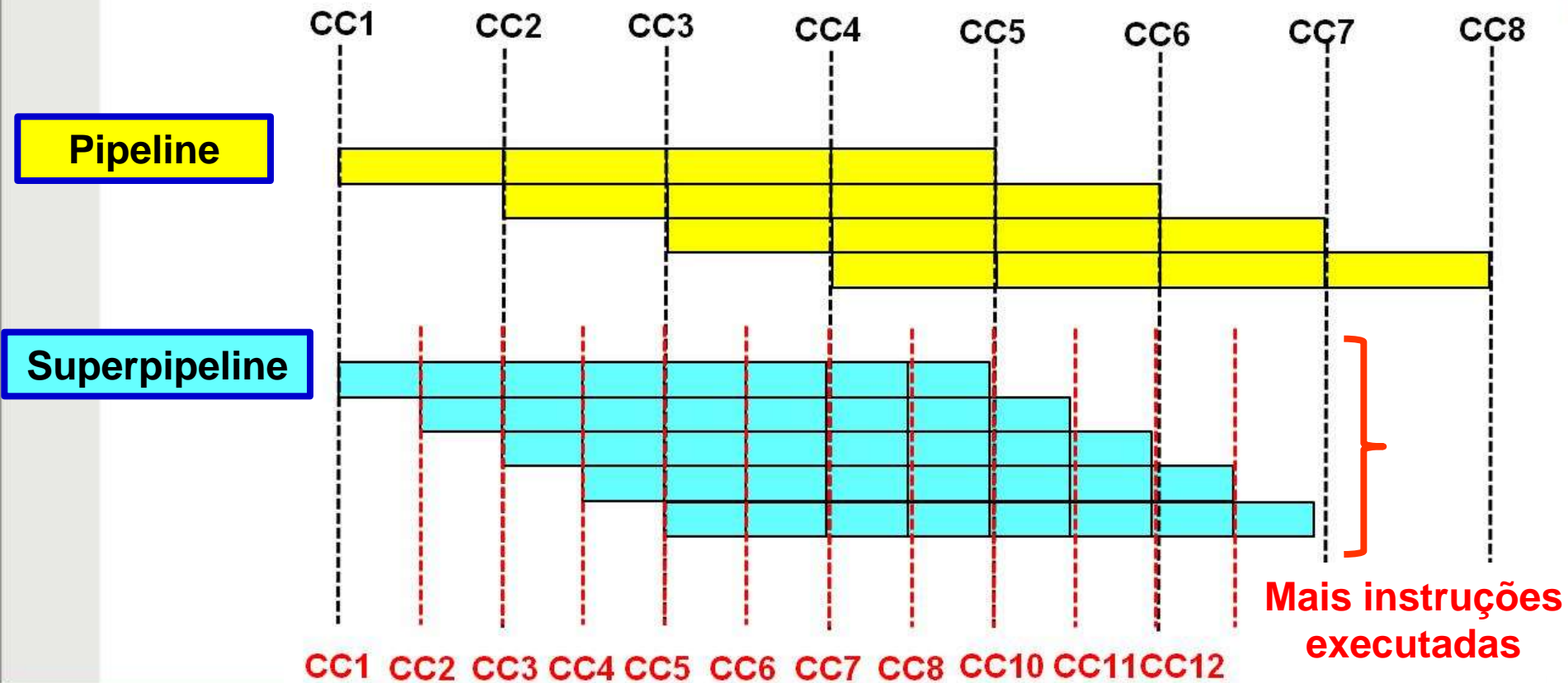


Como Melhorar Desempenho de Pipeline?

- Aumentando o número de estágios
 - Estágios de menor duração
 - Frequência do clock maior
 - Superpipeline**
- Aumentando a quantidade de instruções que executam em paralelo
 - Paralelismo real
 - Replicação de recursos de HW
 - Superescalar e VLIW**

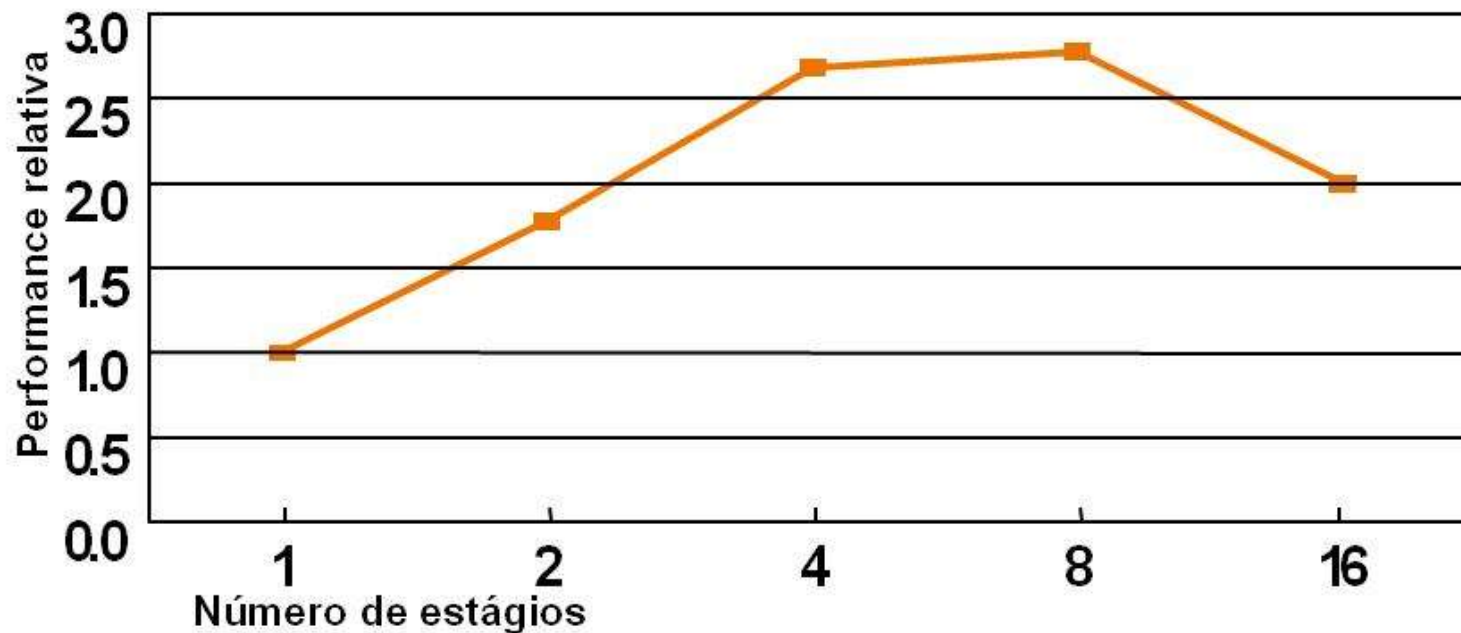
Pipeline x Superpipeline

- Superpipeline: Maior throughput → Melhor desempenho
Maior números de estágios, frequência maior de clock
Mais instruções podem ser processadas simultaneamente



Desempenho Relativo ao Número de Estágios

- Aumentar profundidade do pipeline nem sempre vai melhorar desempenho



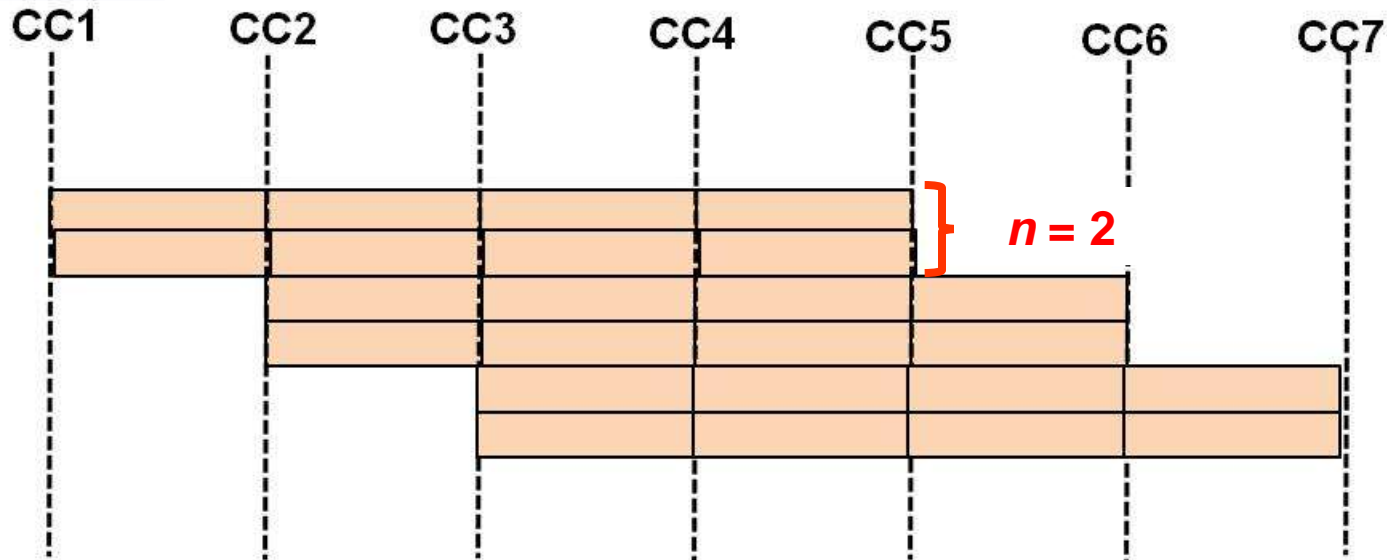
Superescalar

- Processador com n pipelines de instrução replicados
 n dá o grau do pipeline superescalar
- Instruções diferentes podem iniciar a execução ao mesmo tempo
- Requer replicação de recursos de HW
- Aplicável a arquiteturas RISC e CISC
 - RISC : melhor uso efetivo
 - CISC : implementação mais difícil

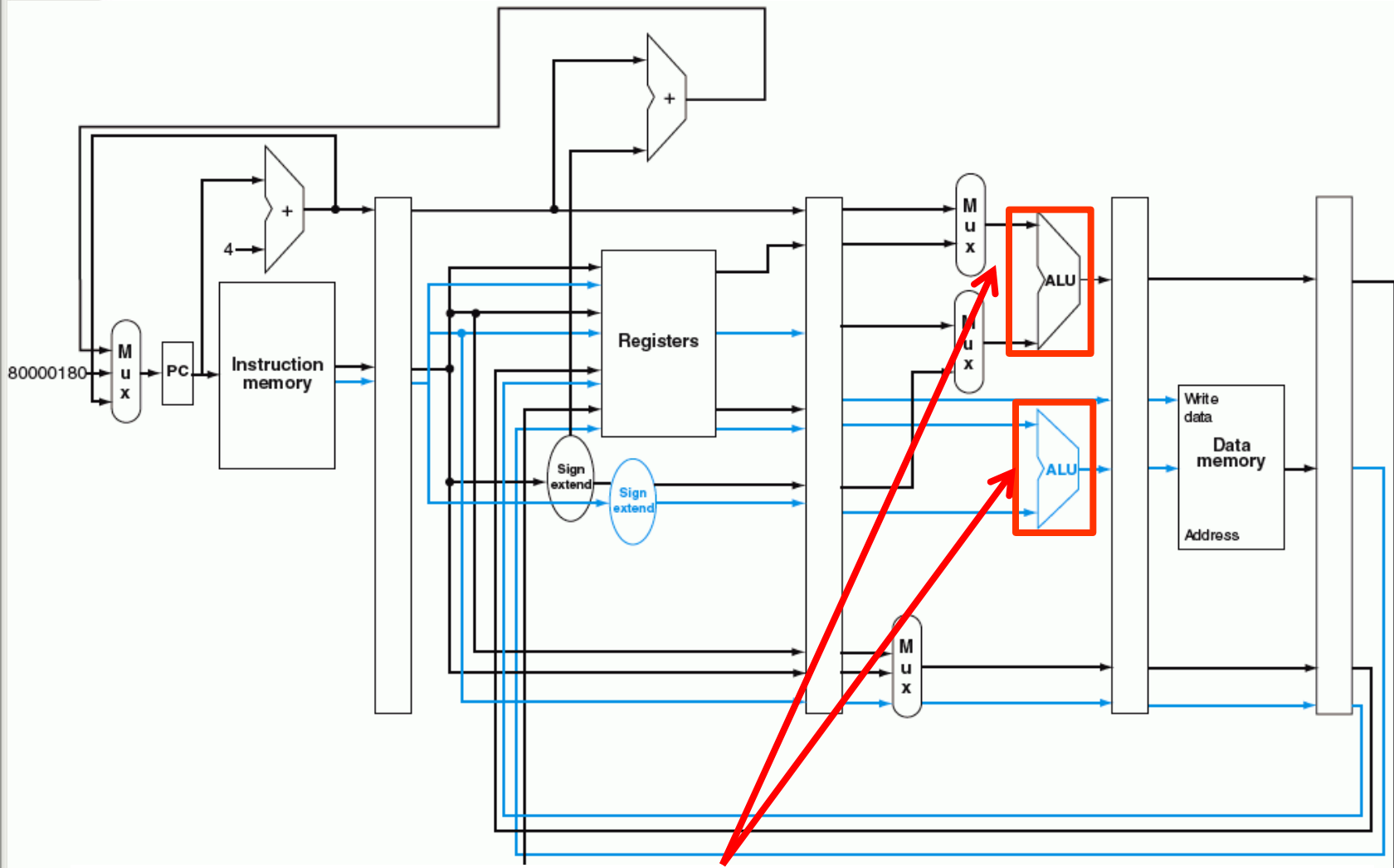
Idéia Geral de Processadores Superescalares

- Grupos de instruções podem ser executados ao mesmo tempo
Número n de instruções por grupo define o grau do pipeline superescalar

Superescalar

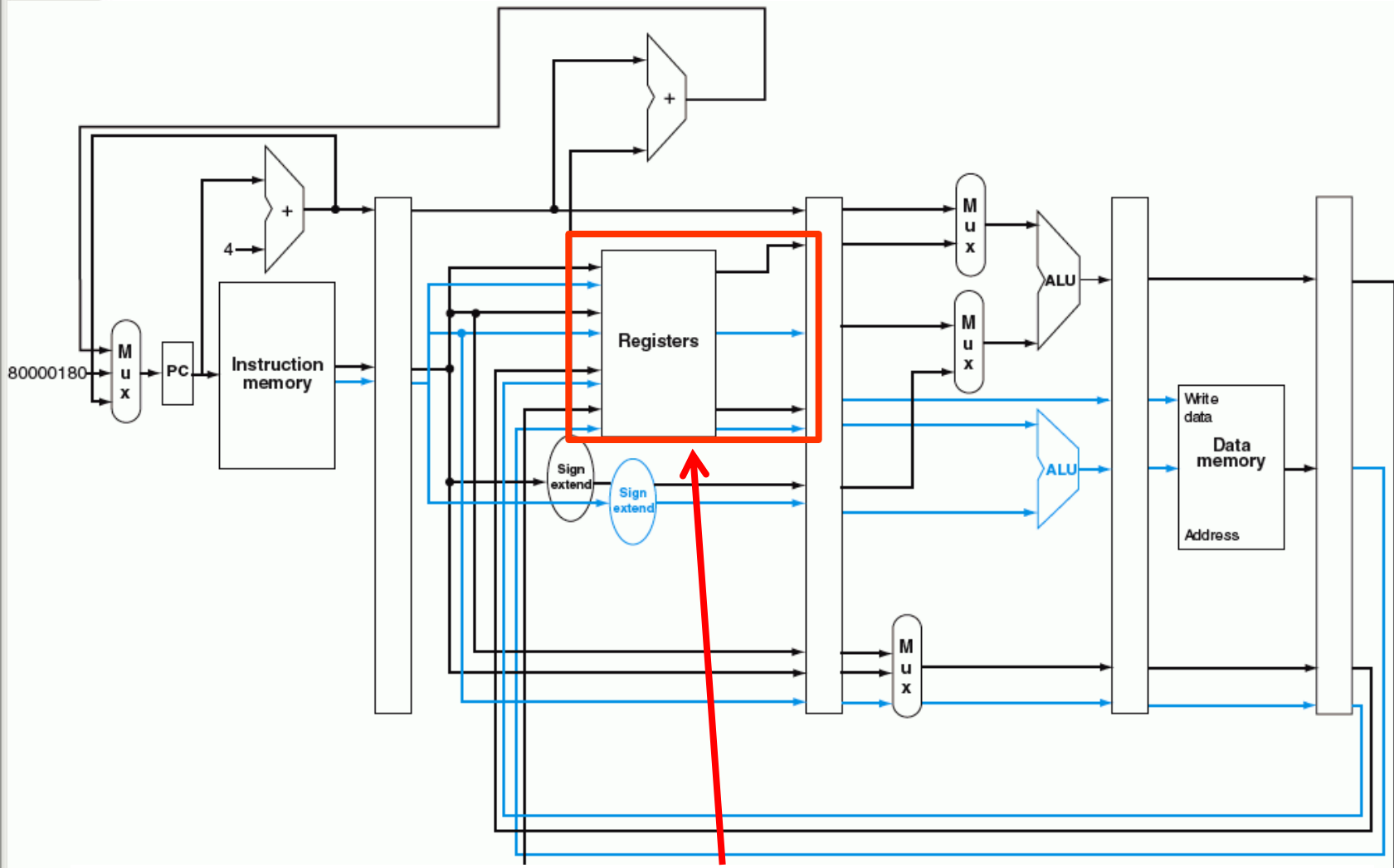


Processador Superescalar com ISA do MIPS



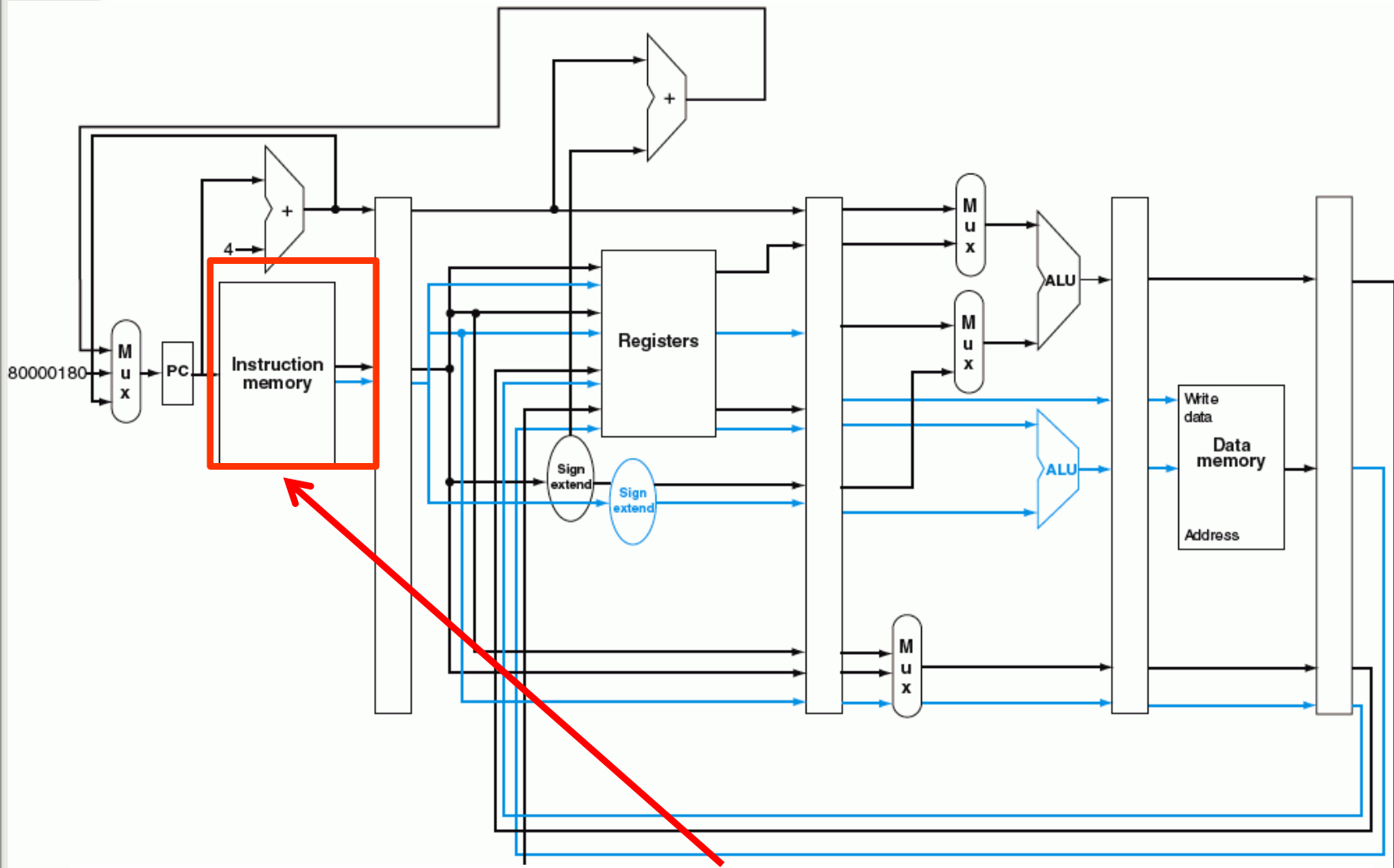
Replicação de ALU – Capacidade para realizar operações aritméticas/branches e de acesso a memória simultaneamente

MIPS Superescalar – Banco de Registradores



Banco de registradores capazes de fazer 4 leituras e 2 escritas

MIPS Superescalar – Memória de Instrução



Memória de Instrução permite leitura de 64 bits

Operando com MIPS Superescalar

- Leitura da instrução de memória deve ser de 64 bits
Execução das instruções aos pares
Primeiros 32 bits, instrução aritmética/branch, e os outros 32 bits, instrução load/store

Instruction type	Pipe stages							
ALU or branch instruction	IF	ID	EX	MEM	WB			
Load or store instruction	IF	ID	EX	MEM	WB			
ALU or branch instruction		IF	ID	EX	MEM	WB		
Load or store instruction		IF	ID	EX	MEM	WB		
ALU or branch instruction			IF	ID	EX	MEM	WB	
Load or store instruction			IF	ID	EX	MEM	WB	
ALU or branch instruction				IF	ID	EX	MEM	WB
Load or store instruction				IF	ID	EX	MEM	WB

Otimizando o Desempenho do Programa

```
Loop: lw $t0,0($s1)
      addu $t0,$t0,$s2
      sw $t0,0($s1)
      addi $s1,$s1,-4
      bne $s1,$zero,Loop
```

Loop Unrolling



```
Loop: addi $s1,$s1,-16
      lw $t0,0($s1)
      lw $t1,12($s1)
      addu $t0,$t0,$s2
      lw $t2,8($s1)
      addu $t1,$t1,$s2
      lw $t3,4($s1)
      addu $t2,$t2,$s2
      sw $t0,16($s1)
      addu $t3,$t3,$s2
      sw $t1,12($s1)
      sw $t2,8($s1)
      sw $t3,4($s1)
      bne $s1,$zero,Loop
```

Carregando 4
elementos por
iteração

Desempenho do Programa Otimizado

	ALU ou Branch	Load ou Store	Clock
Loop	<code>addi \$s1, \$s1, -16</code>	<code>lw \$t0, 0(\$s1)</code>	1
	<code>nop</code>	<code>lw \$t1, 12(\$s1)</code>	2
	<code>addu \$t0, \$t0, \$s2</code>	<code>lw \$t2, 8(\$s1)</code>	3
	<code>addu \$t1, \$t1, \$s2</code>	<code>lw \$t3, 4(\$s1)</code>	4
	<code>addu \$t2, \$t2, \$s2</code>	<code>sw \$t0, 16(\$s1)</code>	5
	<code>addu \$t3, \$t3, \$s2</code>	<code>sw \$t1, 12(\$s1)</code>	6
	<code>nop</code>	<code>sw \$t2, 8(\$s1)</code>	7
	<code>Bne \$s1, \$zero, Loop</code>	<code>sw \$t3, 4(\$s1)</code>	8

■ 8 ciclos para 4 iterações ou 2 ciclos por iteração

■ 14 instruções

CPI = 8/14 ou 0,57

Escolhendo as Instruções que são Executadas em Paralelo

- Escolha de quais instruções serão executadas em paralelo pode ser feita por hardware ou software
- Hardware
 - Lógica especial deve ser inserida no processador
 - Decisão em tempo de execução (escolha dinâmica)
- Software
 - Compilador
 - Rearruma código e agrupa instruções
 - Decisão em tempo de compilação (escolha estática)

Escolha pelo HW

- O termo **Superescalar** é mais associado a processadores que utilizam o hardware para fazer esta escolha
- CPU decide se 0, 1, 2, ... instruções serão executadas a cada ciclo
 - Escalonamento de instruções
 - Evitando conflitos
- Evita a necessidade de escalonamento de instruções por parte do compilador
 - Embora o compilador possa ajudar
 - Semântica do código é preservada pela CPU

Escolha de Instruções pelo SW (Processadores VLIW)

- O termo **VLIW (Very Long Instruction Word)** é associado a processadores parecidos com superescalares mas que dependem do software(compilador) para fazer esta escolha
- O compilador descobre as instruções que podem ser executadas em paralelo e as agrupa formando uma longa instrução **(Very Long Instruction Word)** que será despachada para a máquina
 - Escalonamento de instruções
 - Evitando conflitos

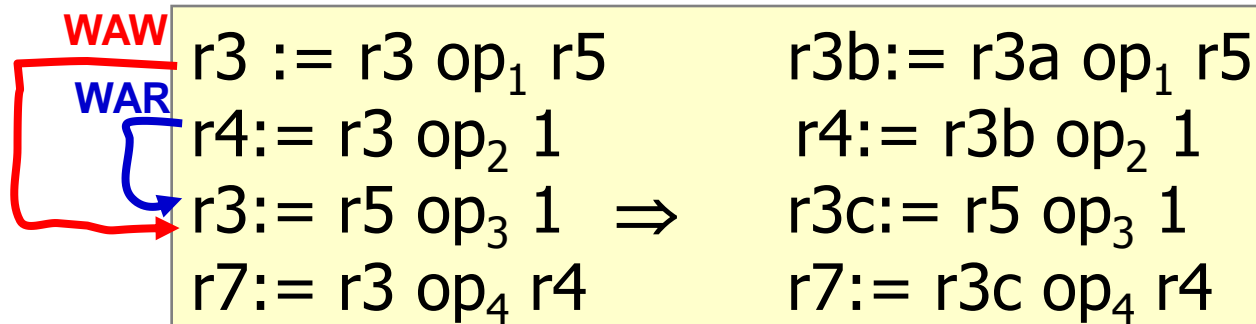
Revendo Dependências de Dados

$r3 := r0 \text{ op}_1 r5$	(i1)
$r4 := r3 \text{ op}_2 1$	(i2)
$r3 := r5 \text{ op}_3 1$	(i3)
$r7 := r3 \text{ op}_4 r4$	(i4)

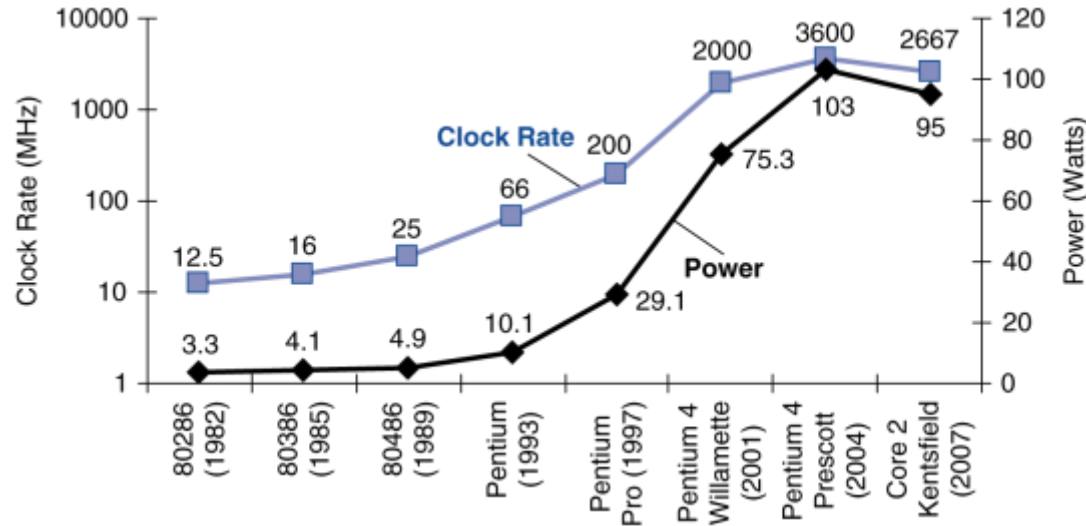
- Dependência Verdadeira (*Read-After-Write – RAW*)
i2 e i1, i4 e i3, i4 e i2
- Anti-dependência (*Write-After-Read - WAR*)
i3 não pode terminar antes de i2 iniciar
- Dependência de Saída (*Write-After-Write – WAW*)
i3 não pode terminar antes de i1

Resolvendo Dependências WAR e WAW

- Mesmo não causando nenhum problema para o pipeline, estas dependências podem “iludir” o compilador ou HW, devendo portanto ser eliminadas
- Solução: **Renomeação de registradores**



Obstáculo para Desempenho: Potência



- Tecnologia CMOS de circuitos integrados

$$\text{Potencia} = \text{Capacitancia} \times \text{Voltagem}^2 \times \text{Frequencia}$$

× 30

Últimos 25 anos

5V → 1V

× 1000

Reduzindo Potência

- Quanto maior a frequência, maior a potência dissipada
 - Sistema de esfriamento do processador é impraticável para frequências muito altas
 - Ruim para dispositivos móveis que dependem de bateria
- **Power wall**
 - Não se consegue reduzir ainda mais a voltagem
 - Sistema de esfriamento não consegue eliminar tanto calor

Como aumentar então desempenho de uma CPU?

Multiprocessadores

- **Múltiplos processadores menores, mas eficientes trabalhando em conjunto**

- **Melhoria no desempenho:**

Paralelismo ao nível de processo

- Processos independentes rodando simultaneamente

Paralelismo ao nível de processamento de programa

- Único programa rodando em múltiplos processadores

- **Outros Benefícios:**

Escalabilidade, Disponibilidade, Eficiência no Consumo de Energia

Programação Paralela

- Desenvolver software para executar em HW paralelo
- Necessidade de melhoria significativa de desempenho
Senão é melhor utilizar processador com único core rápido,
pois é mais fácil de escrever o código
- Dificuldades
 - Particionamento de tarefas (Balanceamento de carga)
 - Coordenação
 - Overhead de comunicação

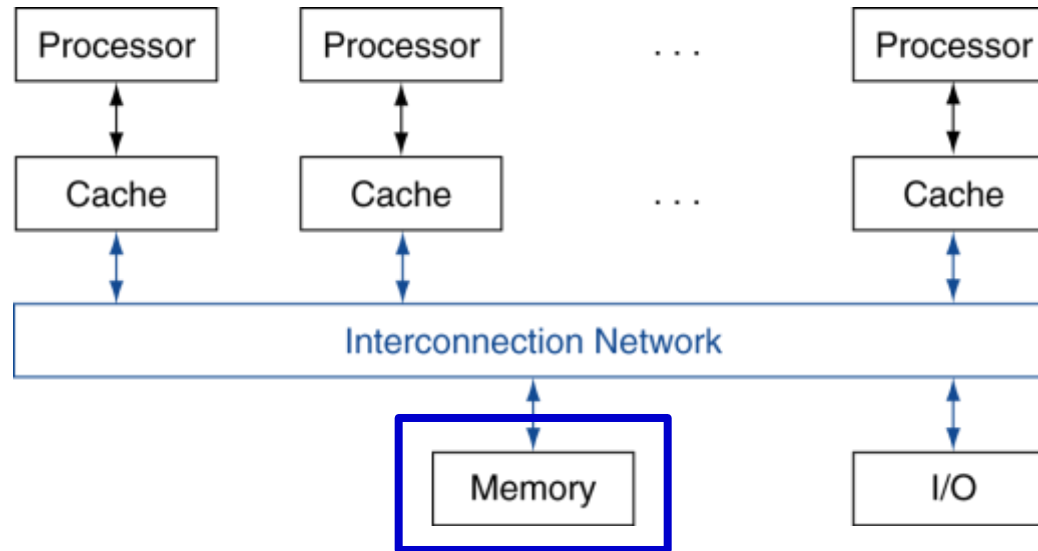
Multicores – Perguntas Importantes

Pergunta 1: Como os diferentes processadores (cores) compartilham dados ?

Pergunta 2: Como é feita a comunicação entre os diferentes processadores ?

Memória Compartilhada

■ SMP: shared memory multiprocessor

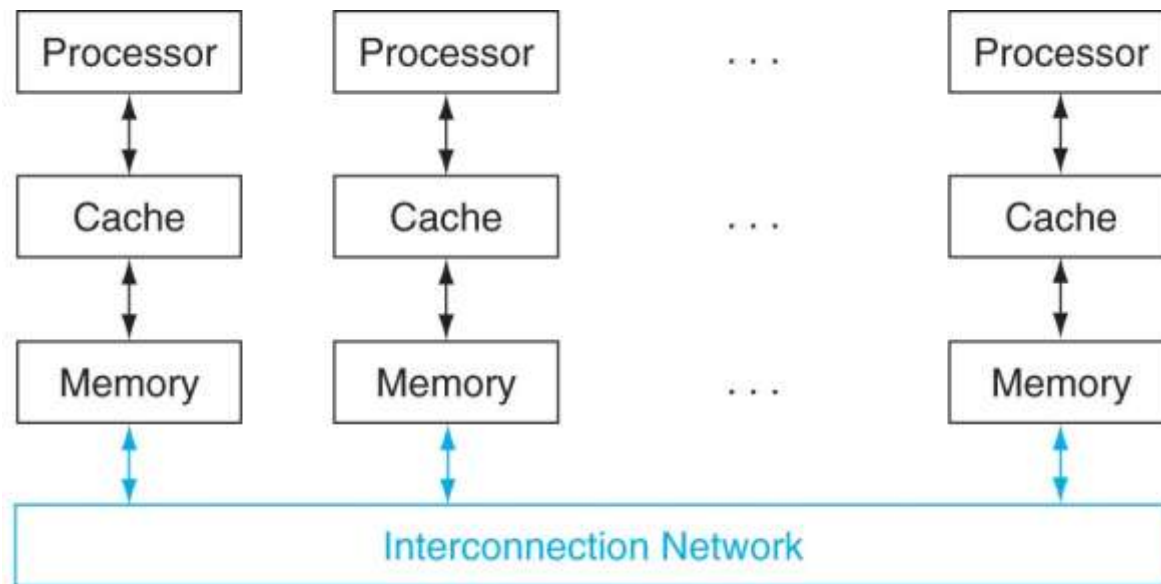


Resposta 1: Mesmo espaço de memória é compartilhado pelos diferentes processadores

Resposta 2: Comunicação é feita através de variáveis compartilhadas

Passagem de Mensagens

■ Message Passing



Resposta 1: Processadores compartilham dados enviando explicitamente os dados (mensagem)

Resposta 2: Comunicação é feita através de primitivas de comunicação (*send* e *receive*)

Comunicação com SMP

- Variáveis compartilhadas contêm os dados que são comunicados entre um processador e outro
- Processadores acessam variáveis via loads/stores
- Acesso a estas variáveis deve ser controlado (sincronizado)

Uso de **locks (semáforos)**

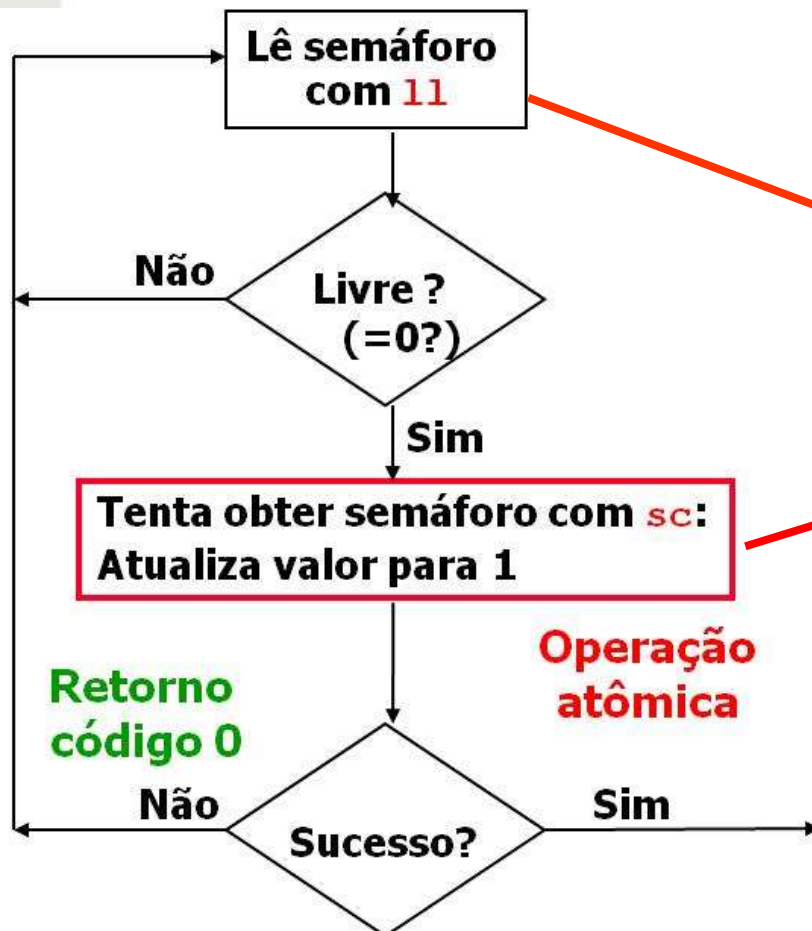
Apenas um processador pode adquirir o lock em um determinado instante de tempo

Suporte do MIPS para Sincronização

- ISA do MIPS possui duas instruções que dão suporte ao acesso sincronizado de uma posição de memória
 - ll** (**l**oad **l**inked)
 - sc** (**s**tore **c**onditional)
- O par de instruções deve ser utilizado para garantir leitura e escrita atômica da memória
 - Garantia de obtenção do semáforo para uma variável compartilhada

Uso do **ll** e **sc**

- **ll** lê uma posição de memória, **sc** escreve nesta mesma posição de memória se ela não tiver sido modificada depois do **ll**



```
try: addi $t0,$zero,1
      ll $t1, 0($s1)
      bne $t1,$zero,try
      sc $t0,0($s1)
      beq $t0,$zero,try
      ... #região crítica
      ...
```

Hardware Multi-Threading

- Abordagem multi-thread

Aumentar utilização de recursos de hardware permitindo que múltiplas **threads** possam executar virtualmente de forma simultânea em único processador

Compartilhamento de unidades funcionais

- Objetivos

Melhor utilização de recursos (cache e unidades de execução são compartilhadas entre threads)

Ganho de desempenho (em média 20%)

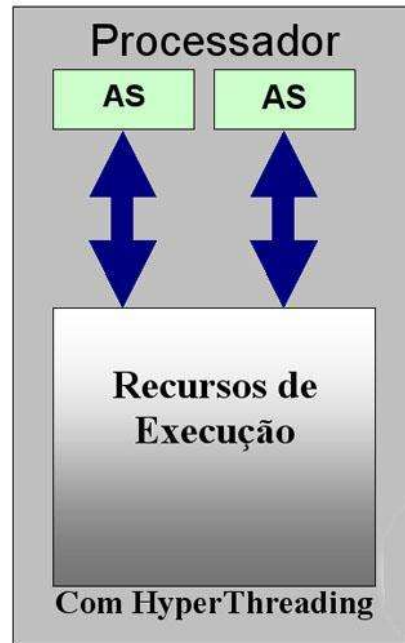
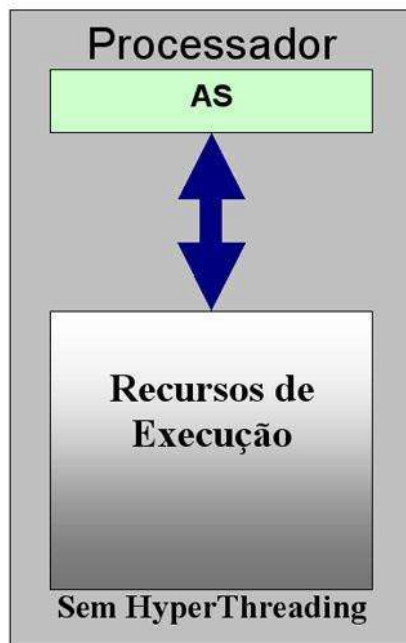
Baixo consumo de área (< 5% da área do chip)

Hardware Multi-Threading

■ Componentes adicionais:

Lógica de controle e duplicação de unidades referente ao contexto do thread em execução (pilha, banco de registradores, buffers de instruções e de dados etc.)

- Permitir concorrência na execução dos threads
- Suporte a troca de contexto eficiente



AS – Architectural State

Tipos de Multi-Threading

■ **Fine-grain** (granularidade fina)

Processador troca de thread a cada instrução

Usa escalonamento Round-robin

- Pula threads paradas (stalled)

Processador deve trocar threads a cada ciclo de clock

■ **Coarse-grain** (granularidade grossa)

Processador troca de threads somente em paradas (stalls) longas

- Exemplo: quando dados não estão na cache

Simultaneous Multi-Threading (SMT)

- **SMT** é uma variação de multi-threading para processadores superescalares com escalonamento dinâmico de threads e instruções

Escalonamento de instruções de threads diferentes

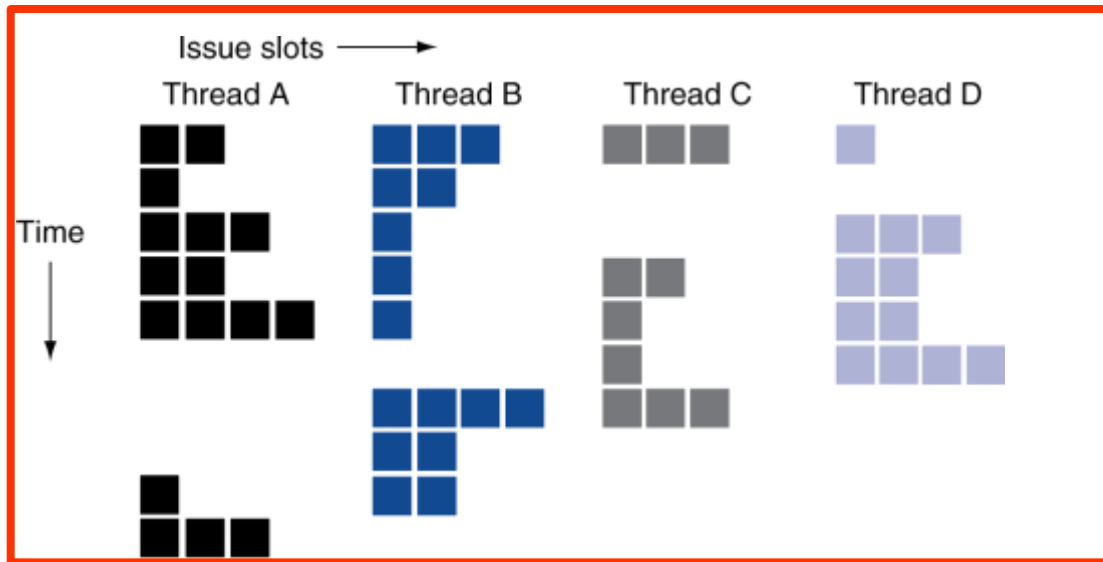
Instruções de threads diferentes podem executar paralelamente desde que haja unidades funcionais livres

Explora paralelismo ao nível de instrução (Instruction **L**evel **P**arallelism - **ILP**)

Explora paralelismo ao nível de threads (**T**hread **L**evel **P**arallelism – **TLP**)

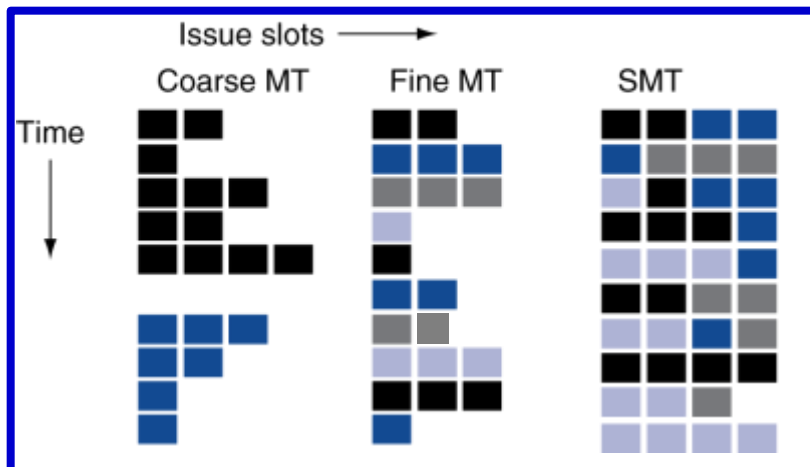
- Hyper-Threading é um caso de de SMT proposto pela Intel
Disponível em processadores Xeon, Pentium 4- HT, Atom

Exemplo de Multi-Threading



■ instrução

4 threads executando isoladamente em um processador superescalar de grau 4 sem multi-threading



4 threads executando conjuntamente em um processador superescalar de grau 4 com multi-threading utilizando diferentes estratégias de multi-threading

Graphics Processing Units (GPUs)

- Indústria de jogos eletrônicos impulsionou o desenvolvimento de dispositivos de alto desempenho para processamento gráfico
- GPUs são aceleradores especializados em processamento gráfico que trabalham em conjunto com a CPU

Livra a CPU de processamento custoso proveniente de aplicações gráficas

GPU dedica todos os seus recursos ao processamento gráfico

Combinação CPU-GPU –
multiprocessamento **heterogêneo**



Características da Arquitetura de GPUs

- Orientado ao processamento paralelo de dados
Composto por vários processadores(cores) paralelos
- GPUs utilizam multi-threading intensivamente
- Compensa o acesso lento a memória com troca intensa de threads
Menos dependência em caches multi-níveis
- Memória utilizada é mais larga e possui largura de banda maior,
Porém são menores que memória para CPU

Exercícios

2. **(POSCOMP 2008 – 32 Modificada)** Analise as seguintes afirmativas e indique quais são falsas.

- I. Uma arquitetura multithreading executa simultaneamente o código de diversos fluxos de instruções (threads).
- II. Em uma arquitetura VLIW, o controle da execução das várias instruções por ciclo de máquina é feito pelo compilador.
- III. Uma arquitetura superescalar depende de uma boa taxa de acerto do mecanismo de predição de desvio para obter um bom desempenho.
- IV. Um processador dual-core tem eficiência equivalente em termos de consumo de energia do que dois processadores *single-core* de mesma tecnologia.

Falsa