Infraestrutura de Hardware

Implementação Pipeline de um Processador Simples

Perguntas que Devem ser Respondidas ao Final do Curso

- Como um programa escrito em uma linguagem de alto nível é entendido e executado pelo HW?
- Qual é a interface entre SW e HW e como o SW instrui o HW a executar o que foi planejado?
- O que determina o desempenho de um programa e como ele pode ser melhorado?
- **Que técnicas um projetista de HW pode utilizar para melhorar o desempenho?**

Diferentes Métricas de Desempenho

- **Tempo de Execução (Latência)**

  Tempo que leva para completar uma tarefa

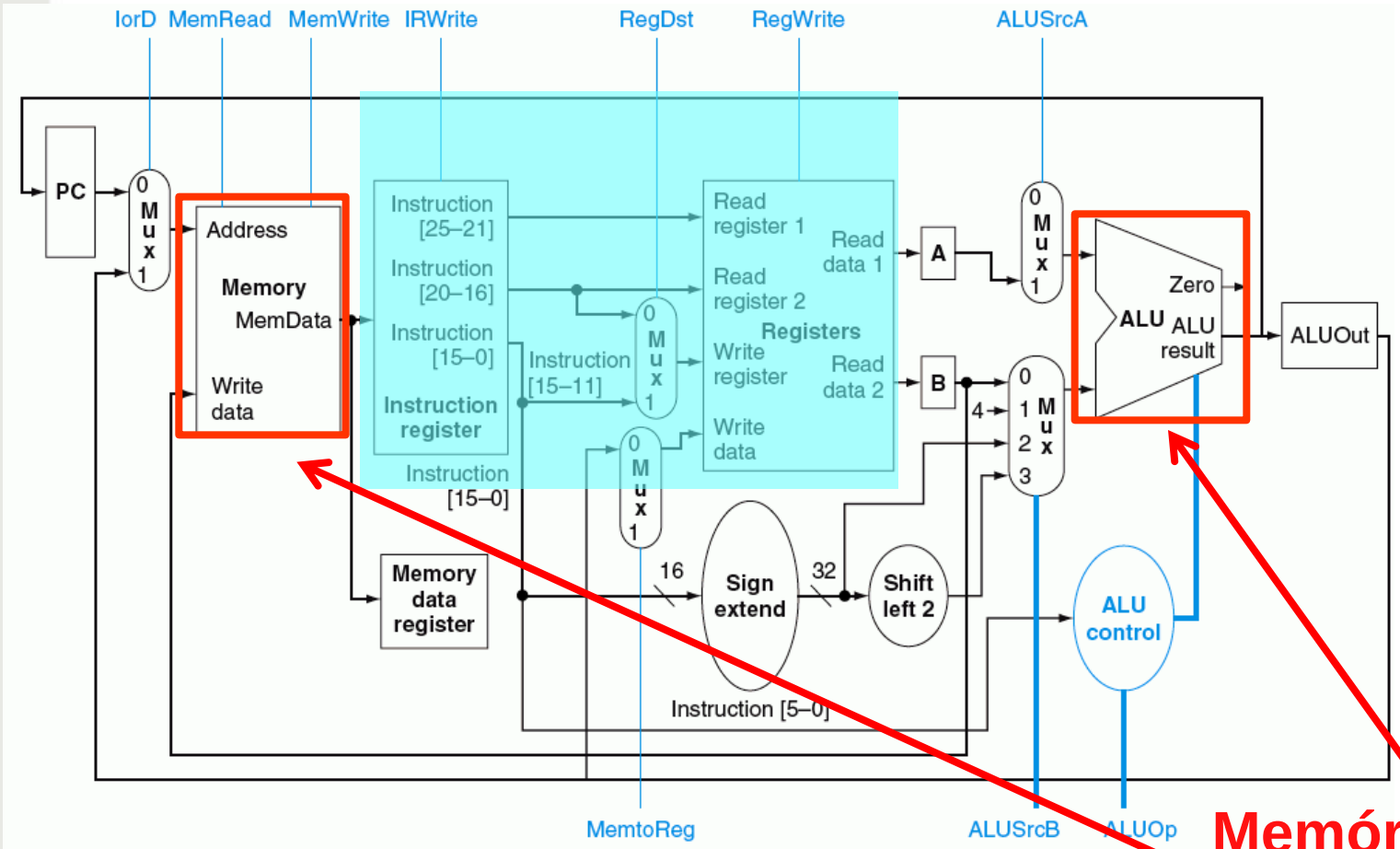
- **Throughput**

  Trabalho total feito por unidade de tempo

  - Exemplos: transações ou instruções por segundo ou hora

- **Uma métrica pode afetar a outra**

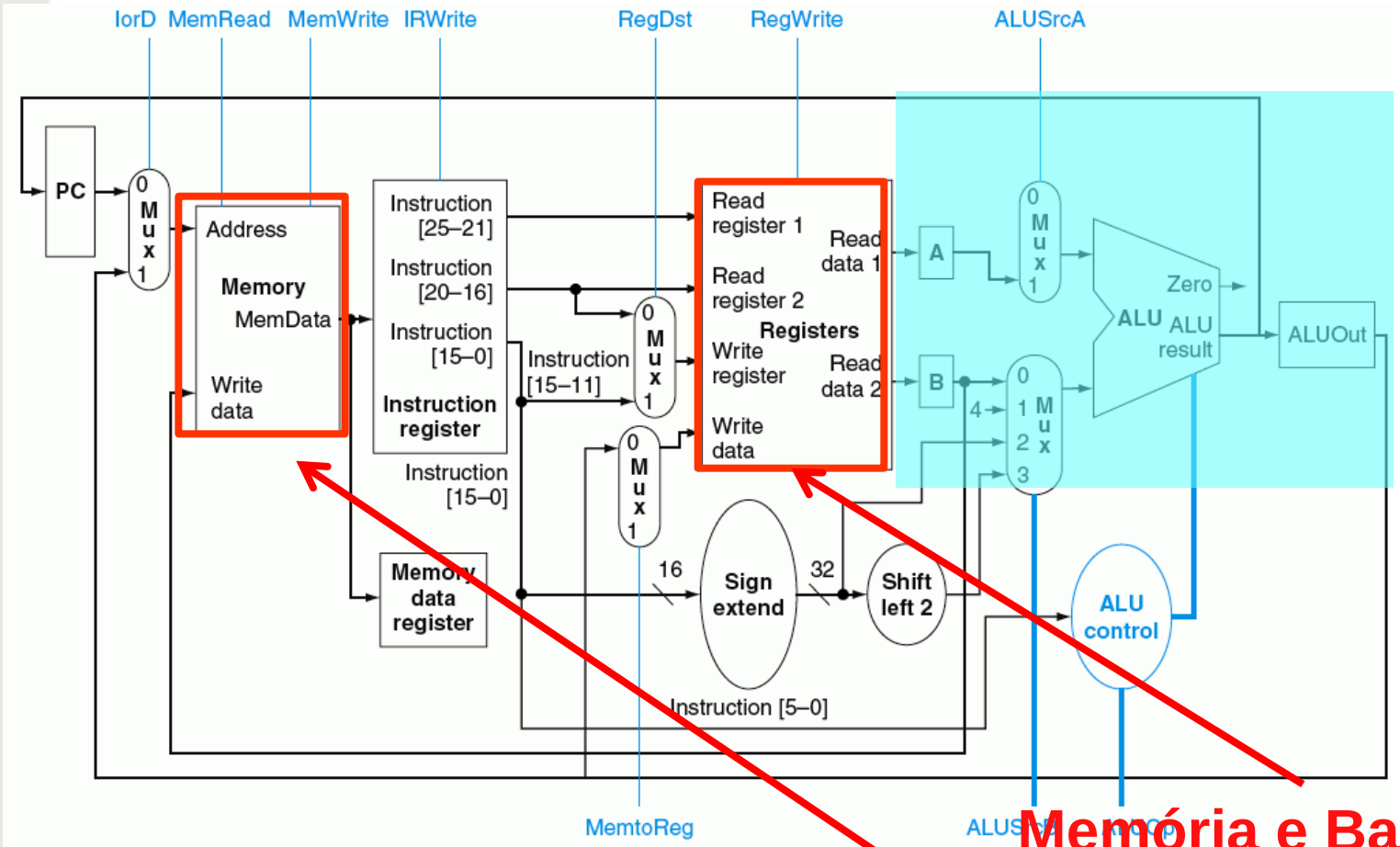
Execução Serial de Uma Instrução ADD (Implementação Multiciclo)



Etapa de decodificação e leitura de registradores

Memória e ALU não utilizados

Execução Serial de Uma Instrução ADD (Implementação Multiciclo)



Etapa de operação da ALU

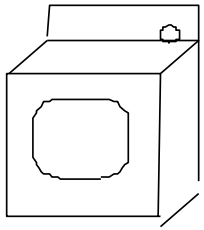
Memória e Banco de Registradores não utilizados

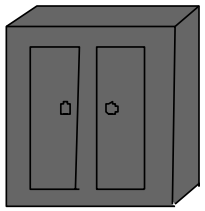
Melhorando Uso de HW com Pipeline

- Em uma implementação serial de instruções, partes do hardware ficam ociosas durante a execução de uma instrução
- **Pipeline** é uma técnica de implementação que visa maximizar a utilização dos diferentes recursos de HW
- Pipeline permite que várias instruções sejam processadas simultaneamente com cada parte do HW atuando numa instrução distinta

Analogia de um Pipeline: Lavanderia

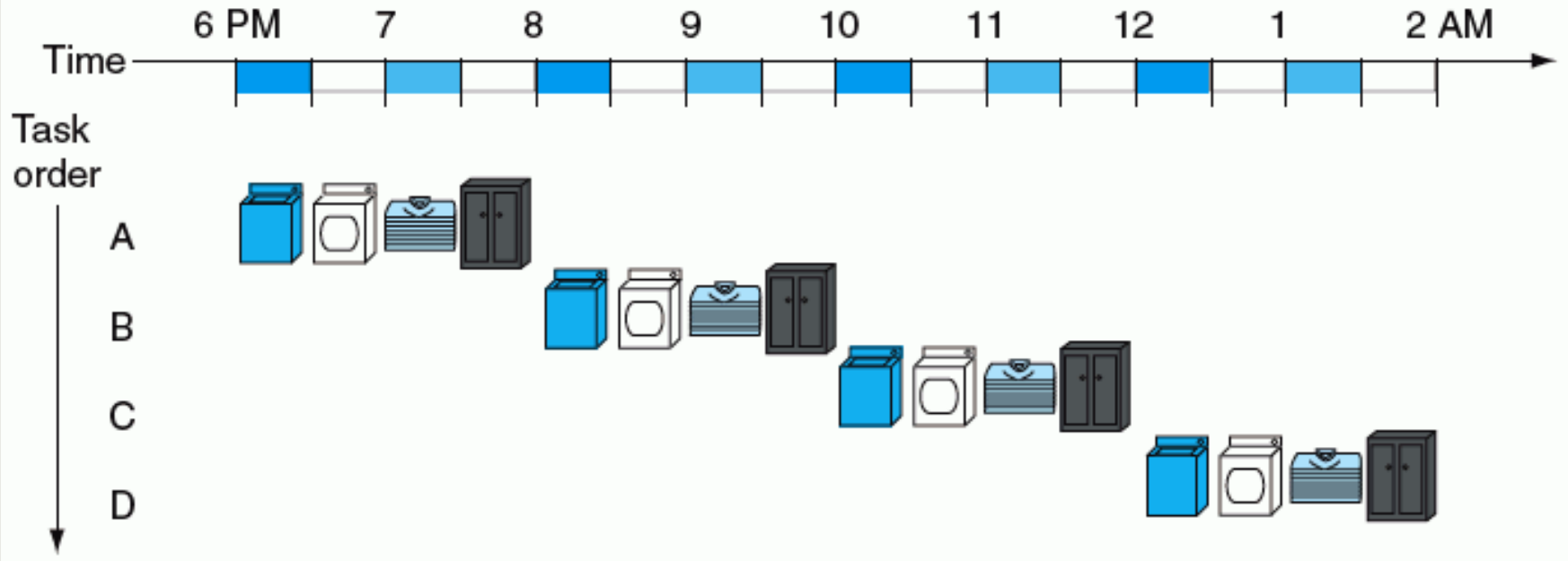
- Podemos dividir a tarefa de lavar roupa em 4 etapas:
  - Lavar: 30 min
  - Secar: 30 min
  - Dobrar: 30 min
  - Guardar : 30 min





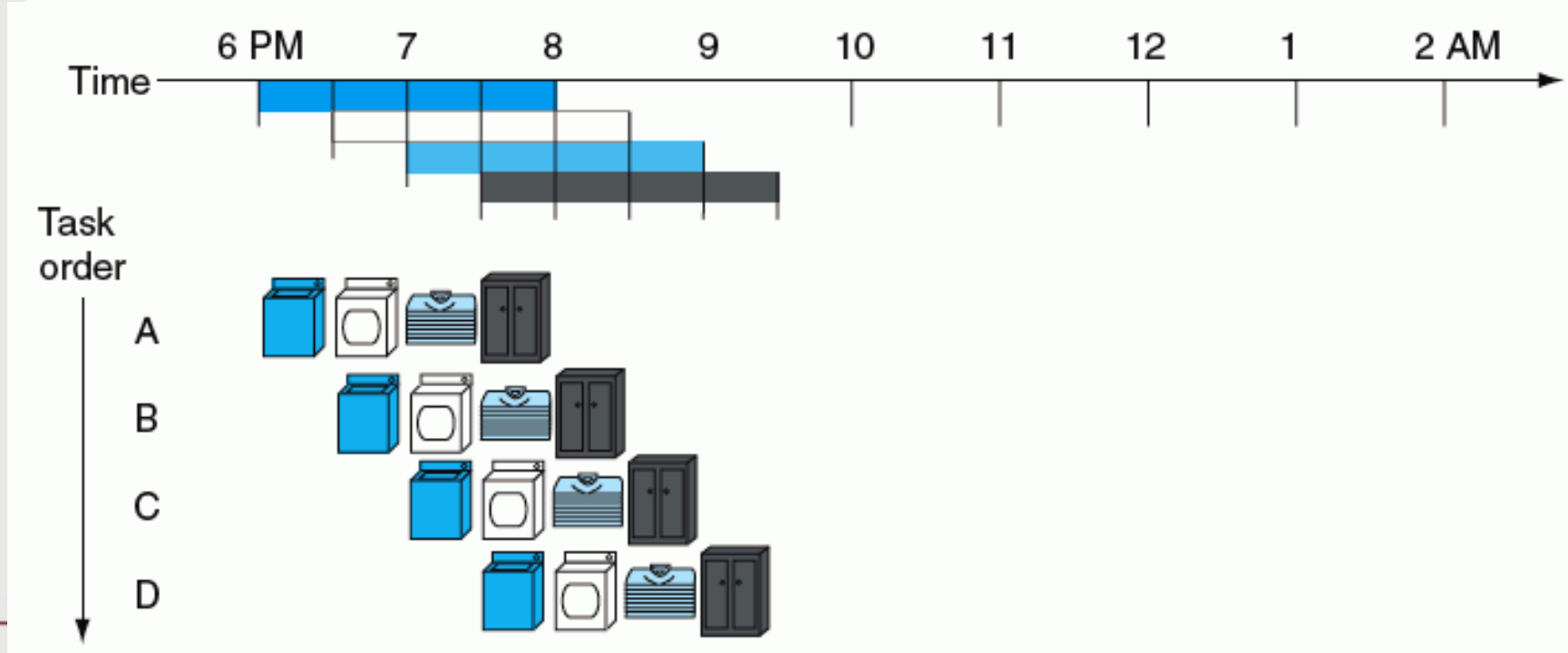
Lavanderia: Execução Sequencial de Tarefas

- Para 4 lavagens de roupa:
  8 horas

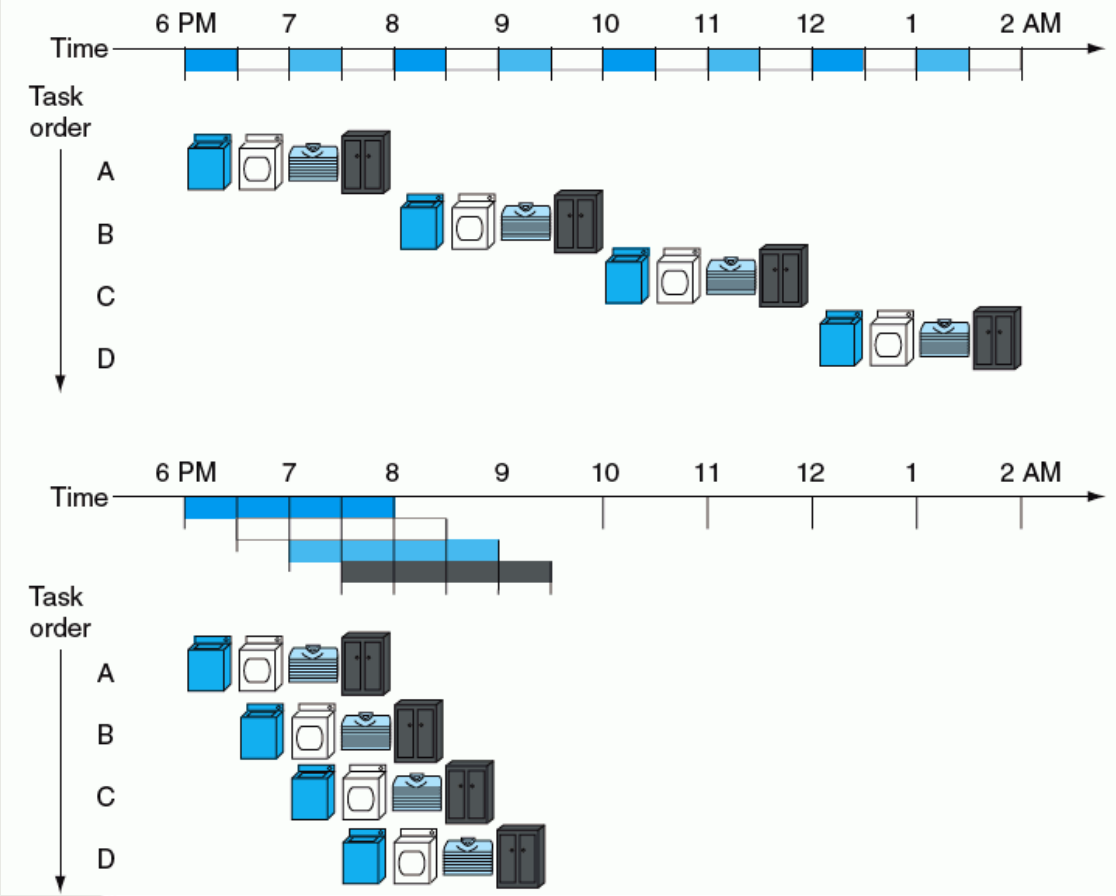


Lavanderia: Execução Pipeline de Tarefas

- Para 4 lavagens de roupa:
  - 3,5 horas
- Diferentes lavagens podem ser executadas simultaneamente, desde que não utilizem o mesmo recurso em um mesmo período de tempo



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 simultâneas
- 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 Processamento de Uma Instrução

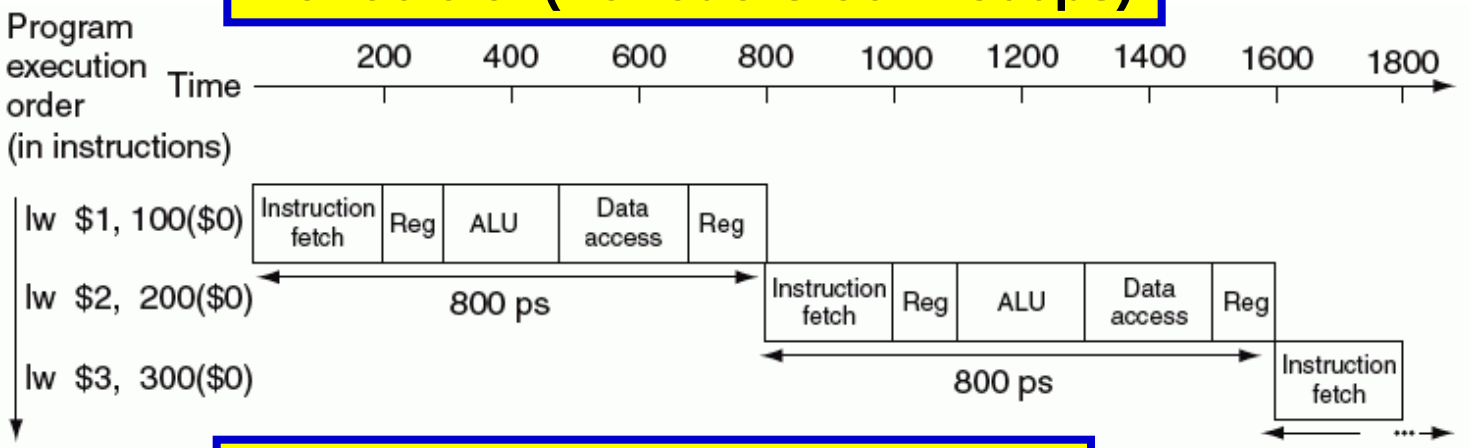
Tempo de Execução de Cada Estágio

- Assuma que o tempo para os diferentes estágios são:
  - 100ps para leitura/escrita de registrador
  - 200ps para os outros estágios

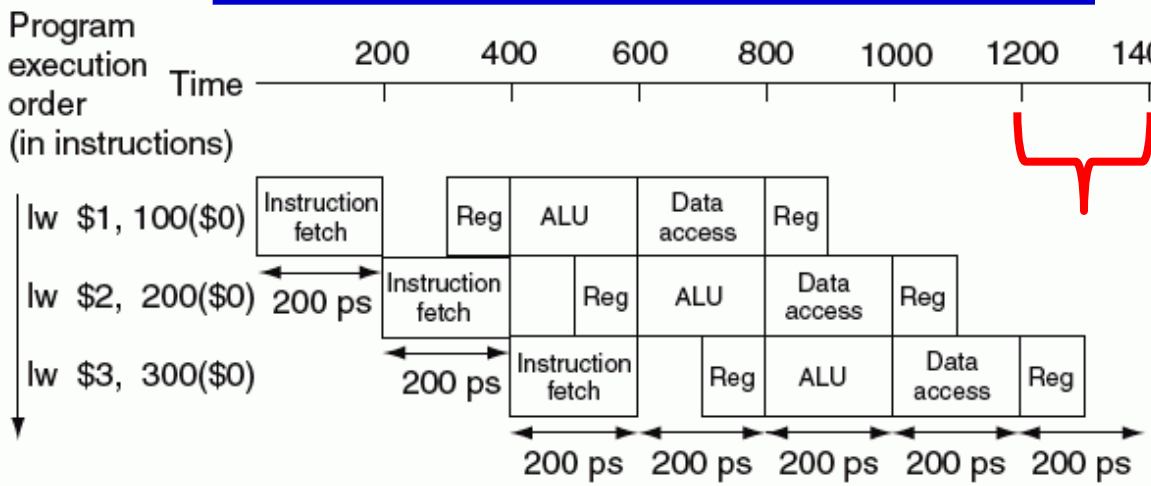
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

Desempenho : Monociclo x Pipeline

Monociclo: (Período Clock= 800ps)



Pipeline: (Período Clock= 200ps)

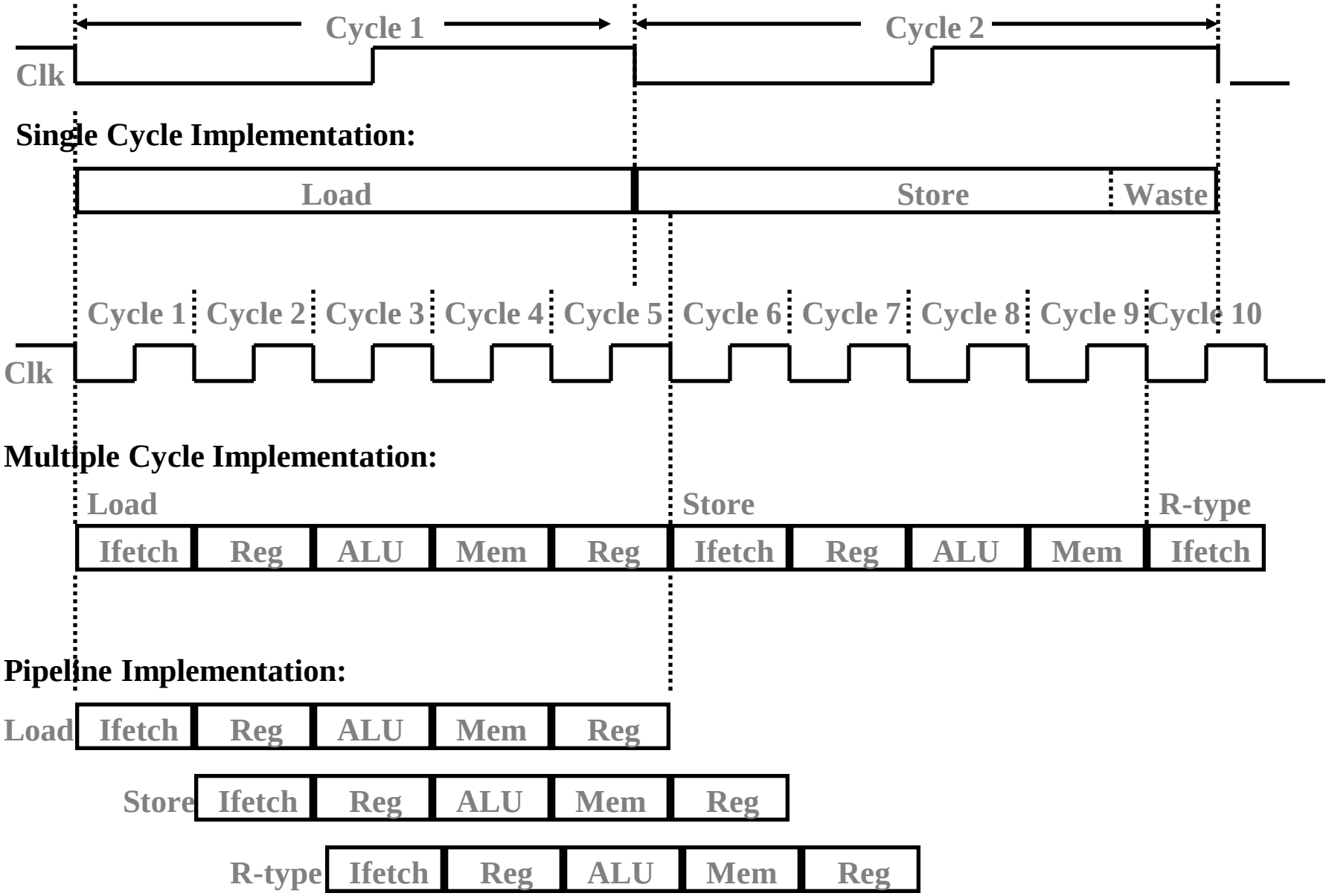


Cada estágio deve levar um ciclo: tempo do estágio mais lento define período

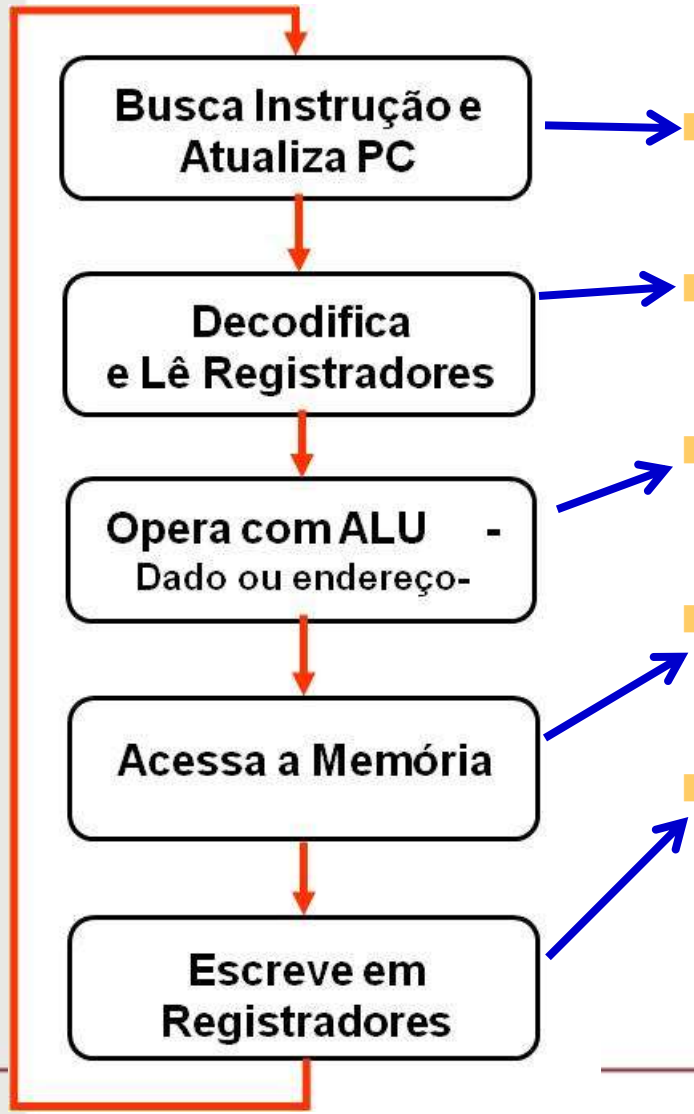
Melhora de Desempenho do Pipeline

- **Se todos os estágios balanceados, isto é, levam o mesmo tempo, e condições ideais**
  - $\text{Tempo entre instruções}_{\text{pipelined}} = \dfrac{\text{Tempo entre instruções}_{\text{nonpipelined}}}{\text{Número de estágios}}$
- Se não balanceado, aumento do desempenho é menor
- Latência (tempo de execução de cada instrução) não diminui com pipeline

Mono, multi-ciclo e Pipeline

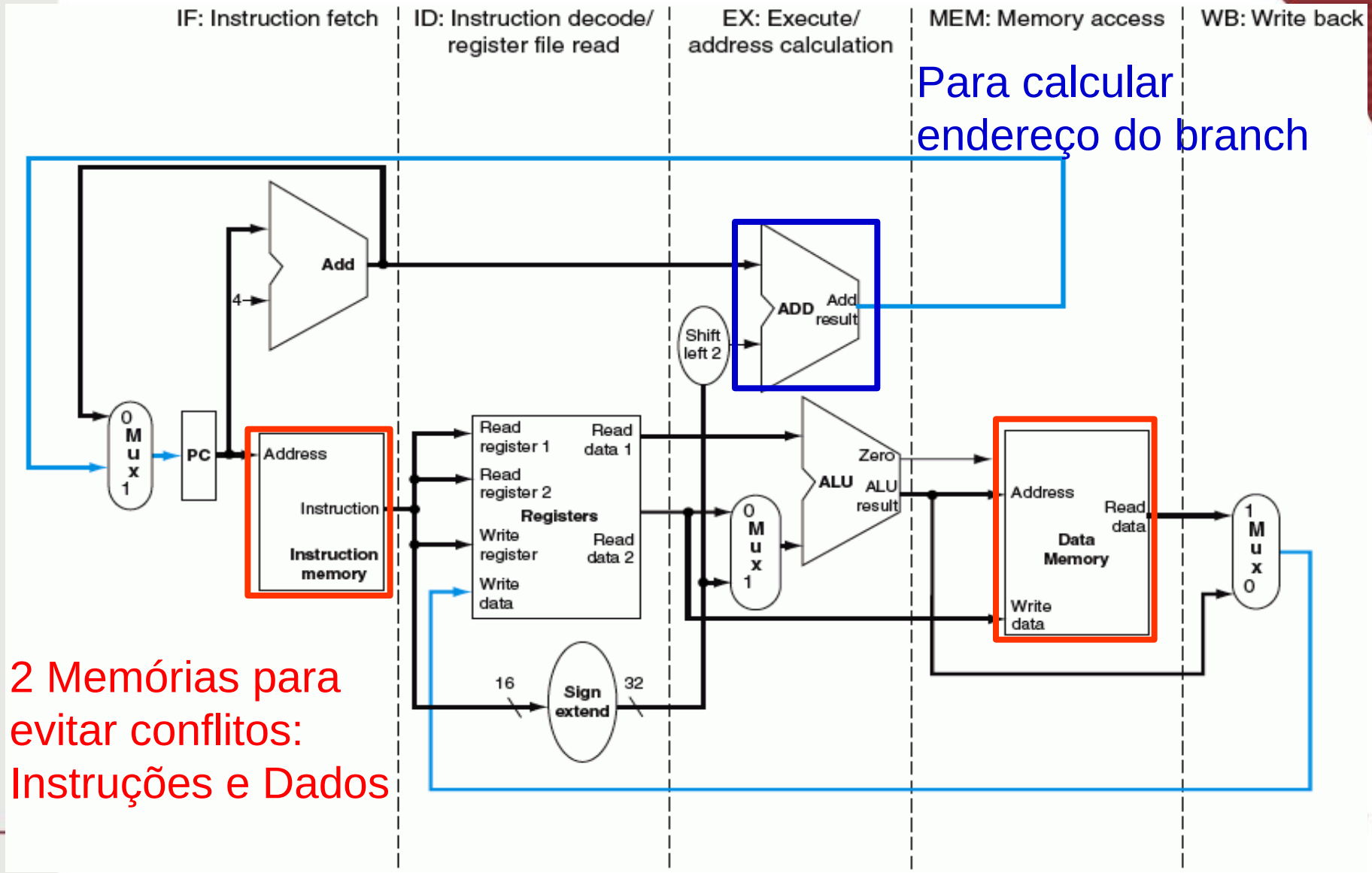


Estágios de uma Implementação Pipeline de um Processador

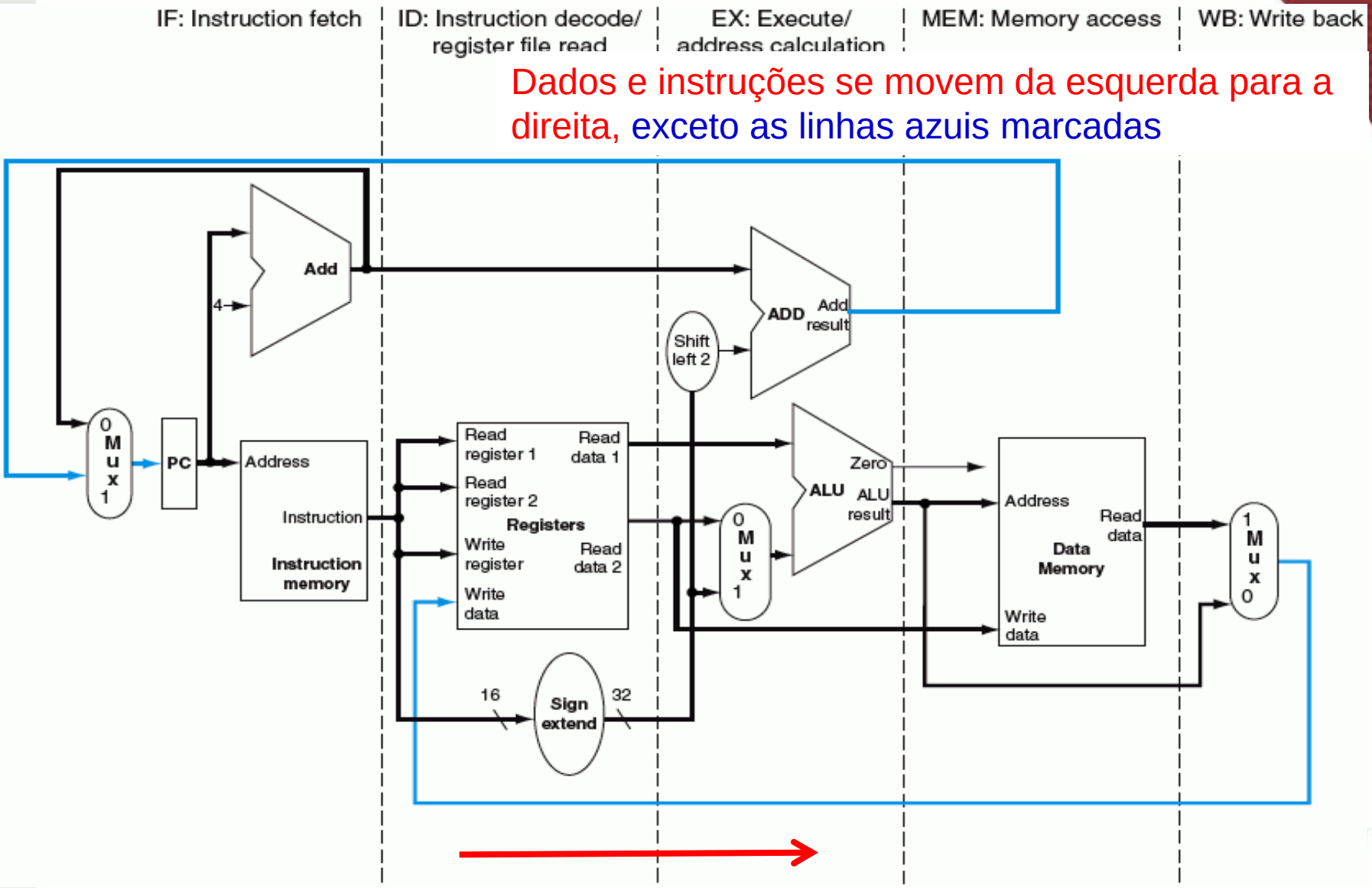


- **IF (I**nstruction **F**etch**)**
  - Busca instrução
- **ID** (**I**nstruction **D**ecode)
  - Decodifica instrução e lê registradores
- **EX** (**EX**ecute operation)
  - Calcula dados ou endereços
- **MEM** (Access **MEM**ory operand)
  - Acessa operando na memória
- **WB** (**W**rite result **B**ack to register)
  - Escreve resultado no registrador

Implementação Pipeline do Datapath



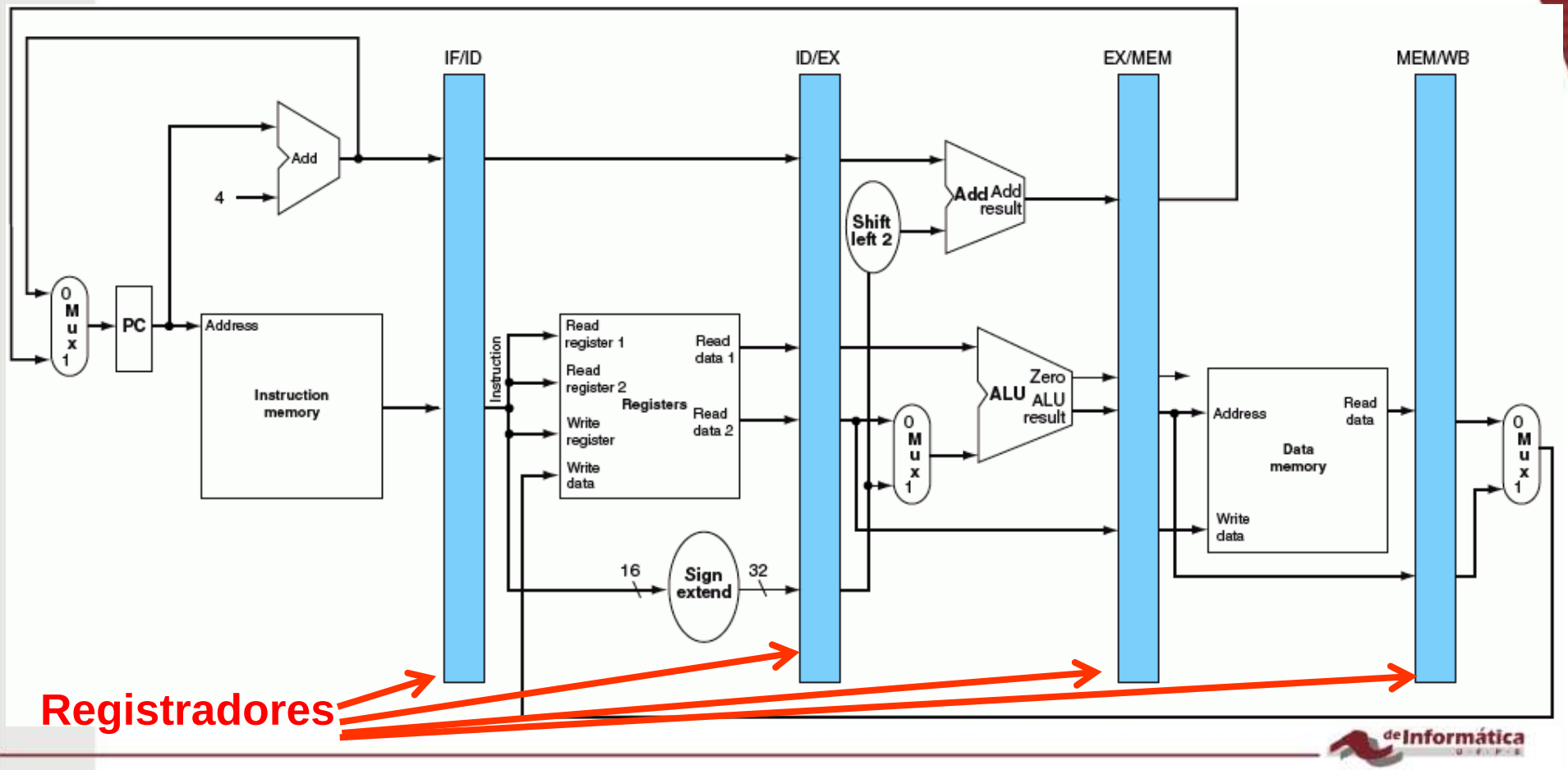
Sentido dos Dados e Instruções em um Pipeline



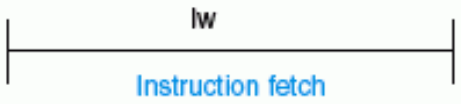
Registradores do Pipeline

- **São necessários registradores entre estágios**

  Para armazenar informação produzido no ciclo anterior



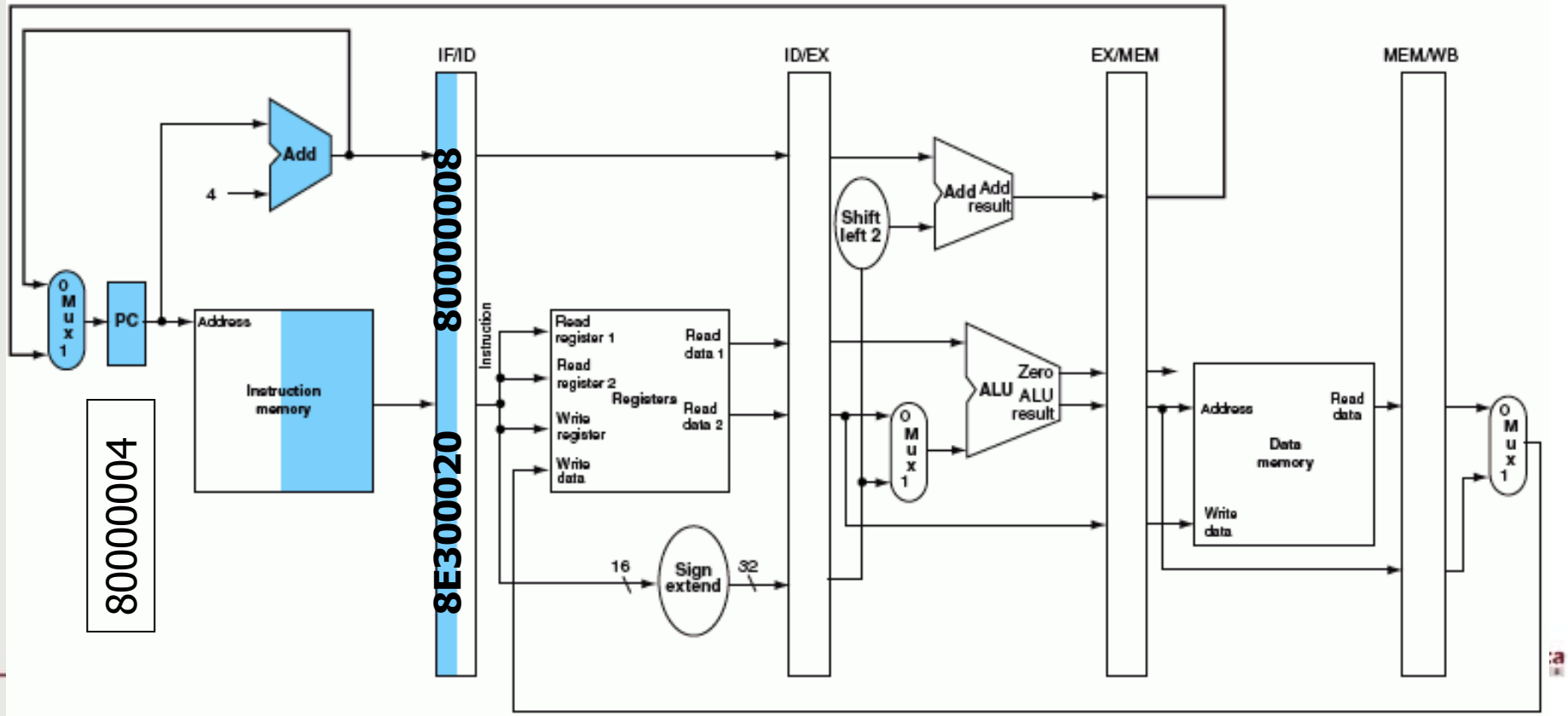
Operação de um Pipeline: lw (Busca Instrução)



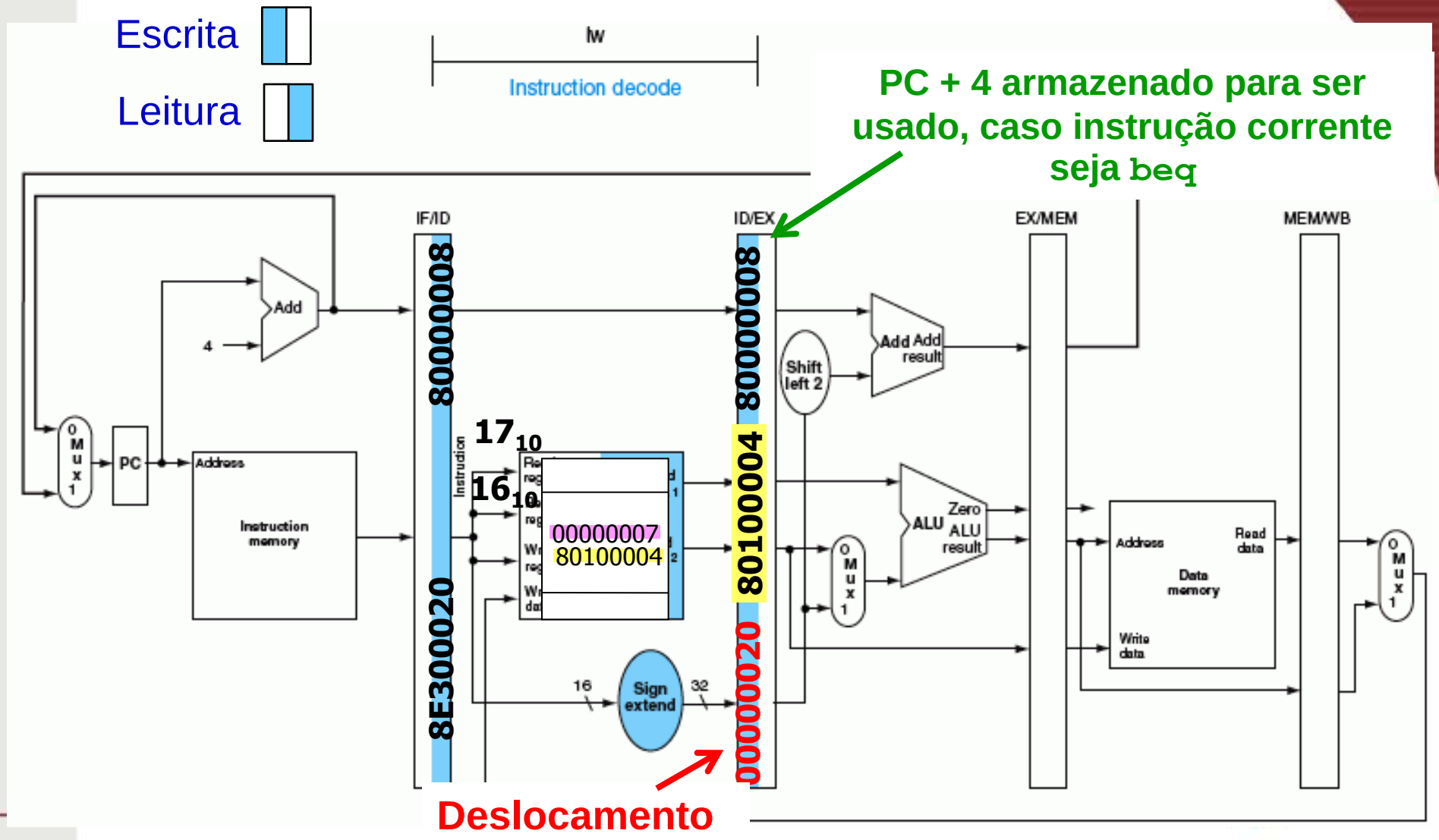

```
lw $s0,32($s1)
```

➡ `0x8E300020`

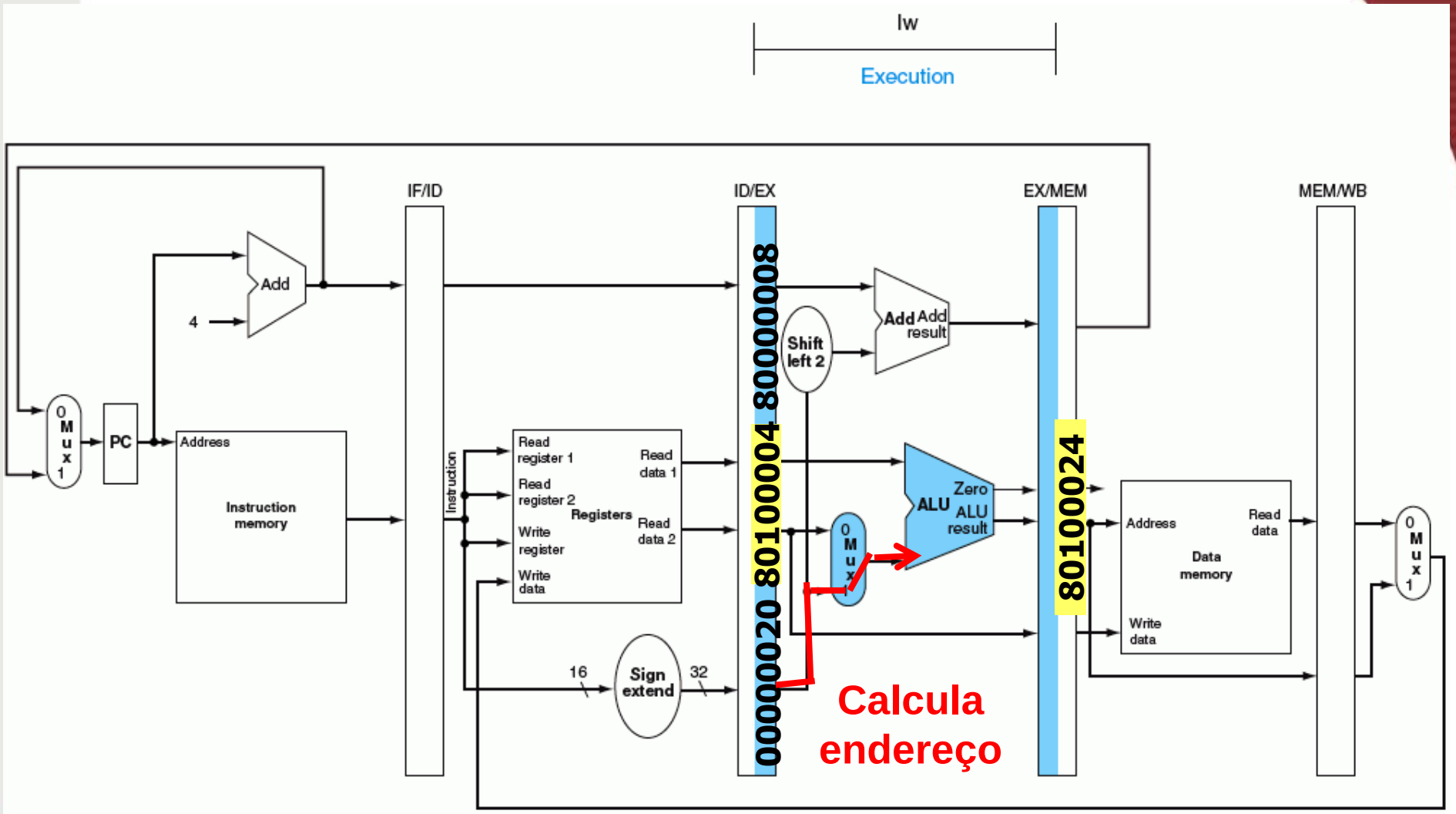
Escrita 



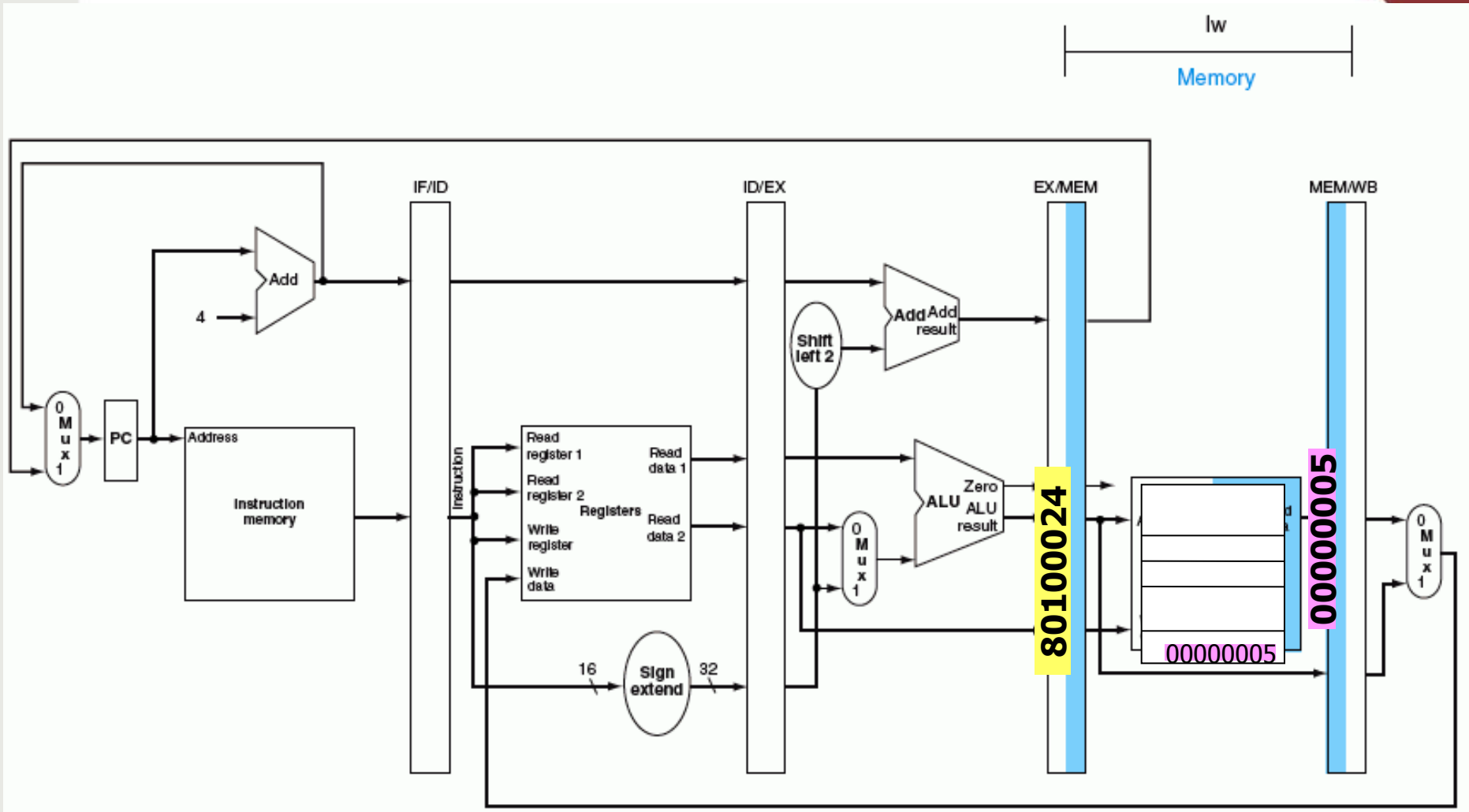
Operação de um Pipeline: lw (Decodificação)



Operação de um Pipeline: lw (Execução)

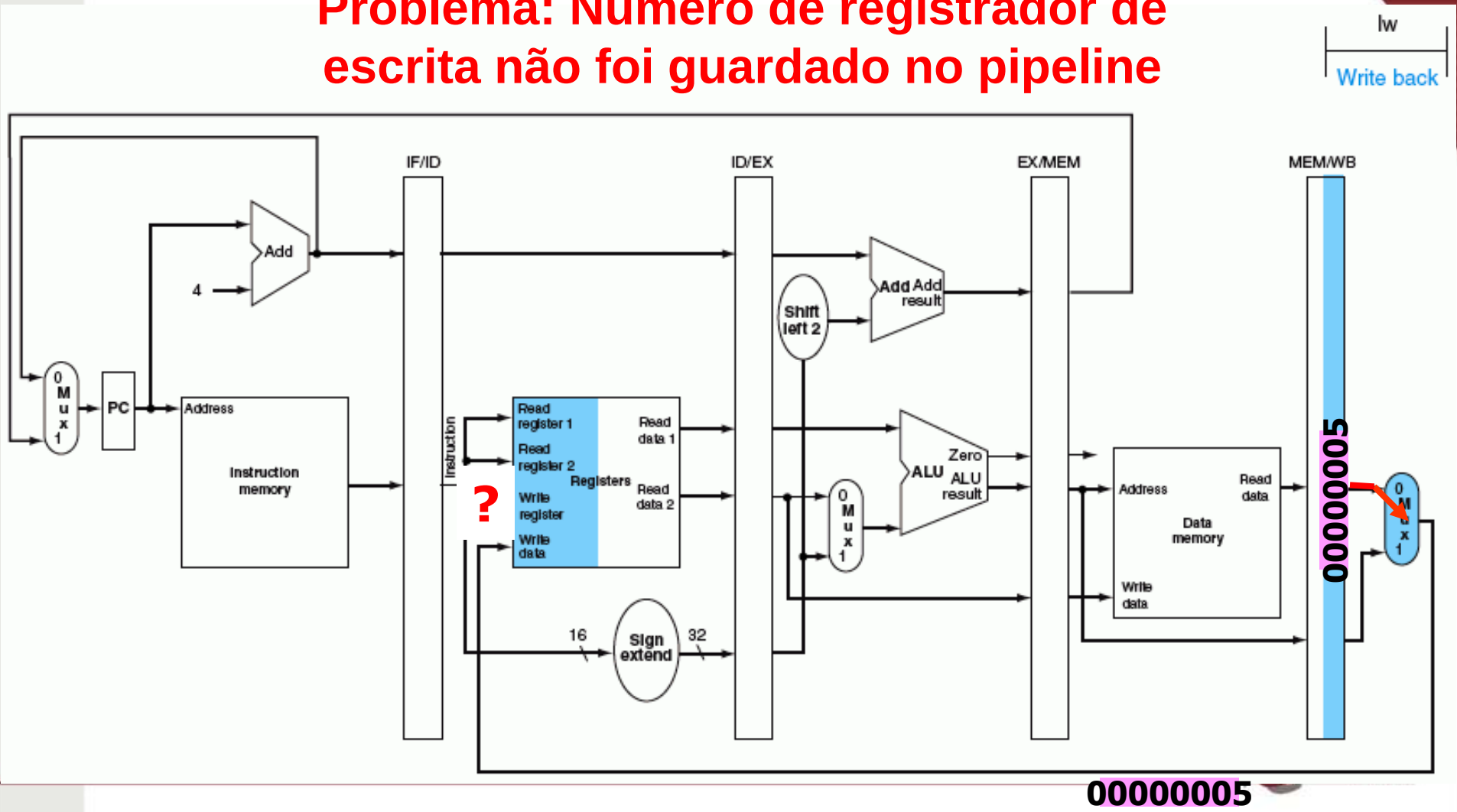


Operação de um Pipeline: lw (Acesso a Memória)



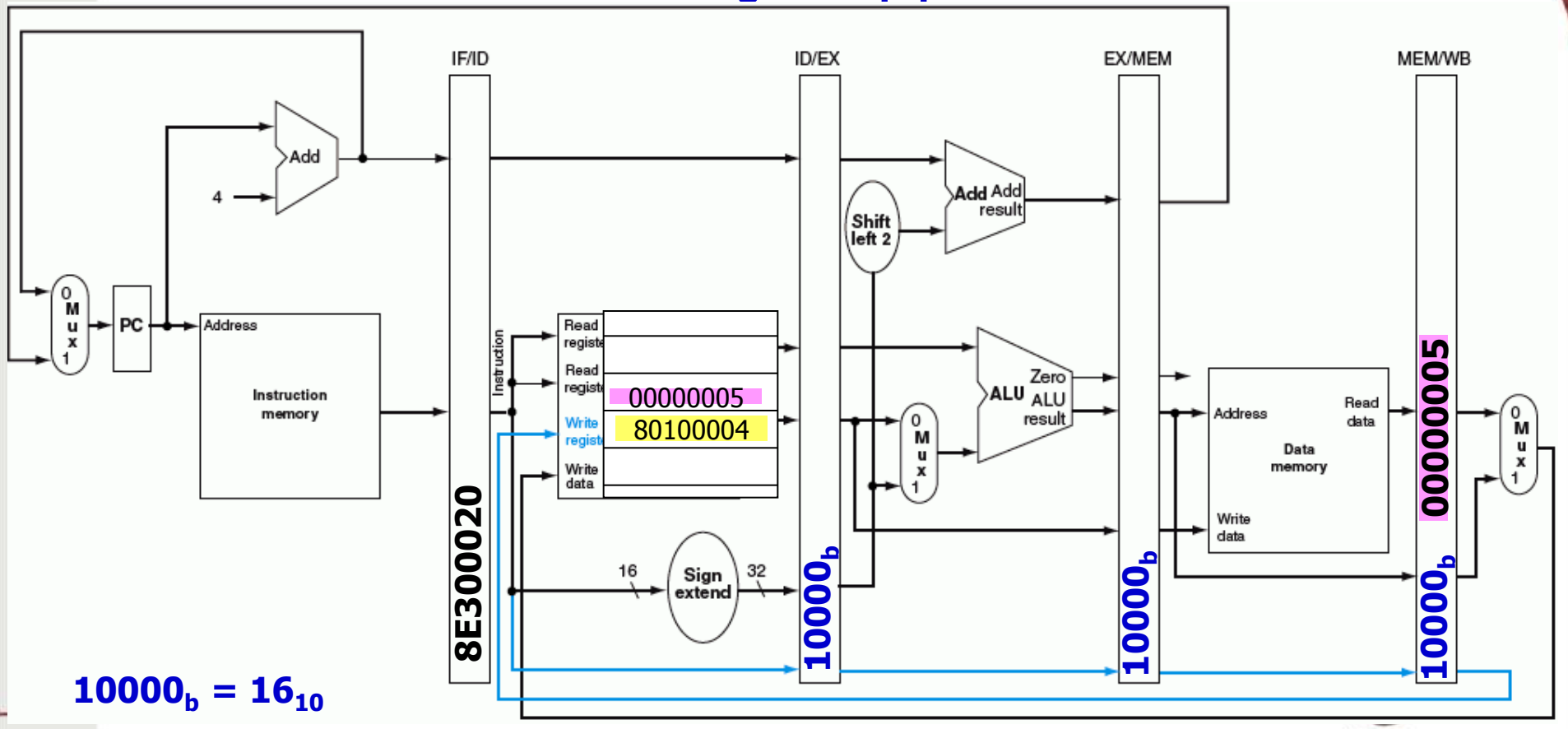
Operação de um Pipeline: lw (Escrita em Registrador)

Problema: Número de registrador de escrita não foi guardado no pipeline



Operação de um Pipeline: lw (Escrita em Registrador)

Corrigindo o problema: Número de registrador de escrita é armazenado nos registradores do pipeline em cada estágio do pipeline



$10000_b = 16_{10}$

Representação Gráfica de Pipelines

- Fluxo de processamento em um pipeline pode ser difícil de entender
- É comum representar pipelines usando 2 tipos de diagramas

  Diagrama de múltiplos ciclos de clock
  - Mostra a operação do pipeline ao longo do tempo

  Diagrama de um único ciclo de clock
  - Mostra uso do pipeline em um único ciclo
  - Destaca recursos utilizados por cada instrução no pipeline em um determinado ciclo

Diagrama de Múltiplos Ciclos de Clock

- **Versão não convencional que mostra a utilização de recursos em cada estágio**

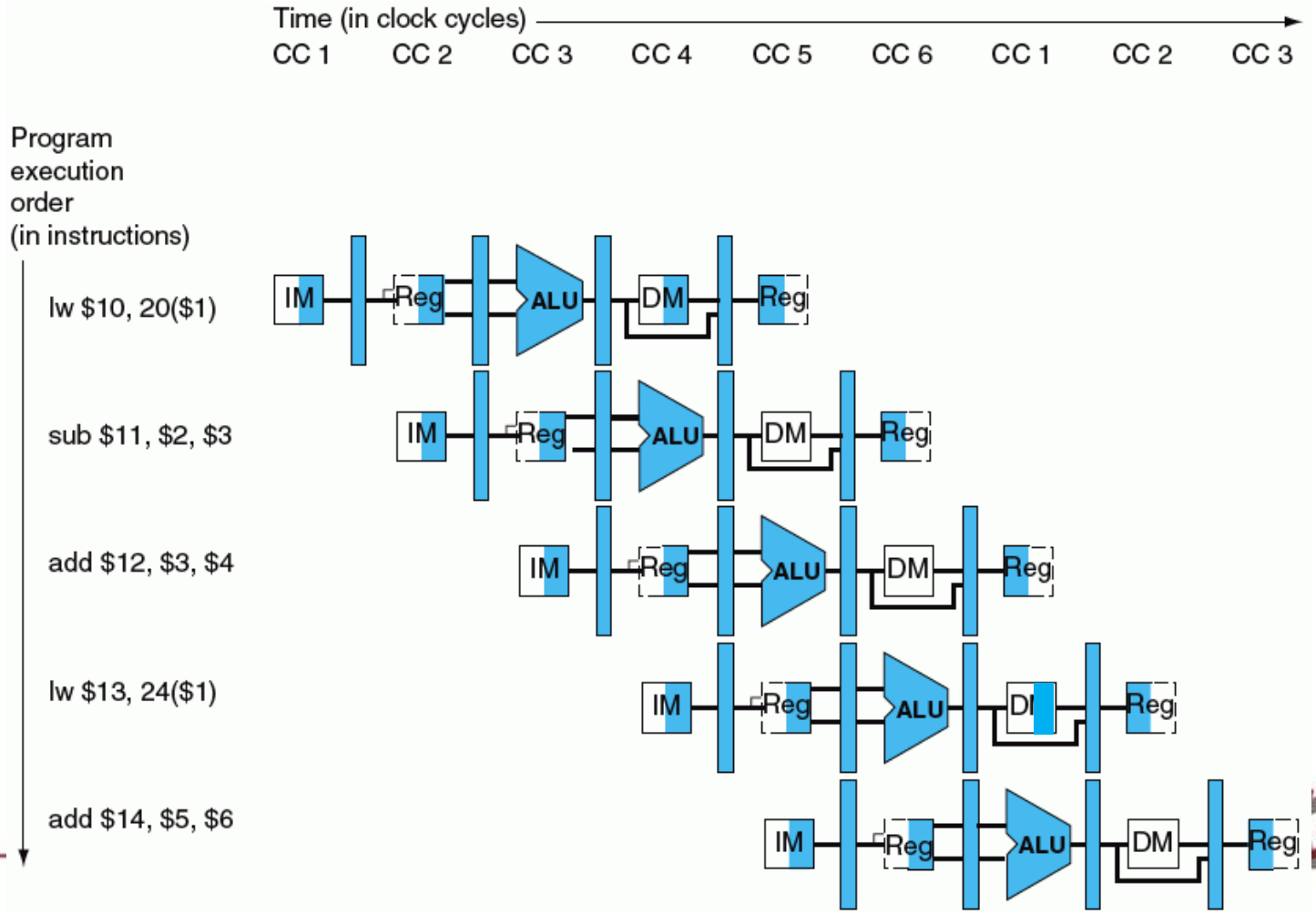


Diagrama de Múltiplos Ciclos de Clock

- **Versão tradicional identifica cada estágio pelo nome**

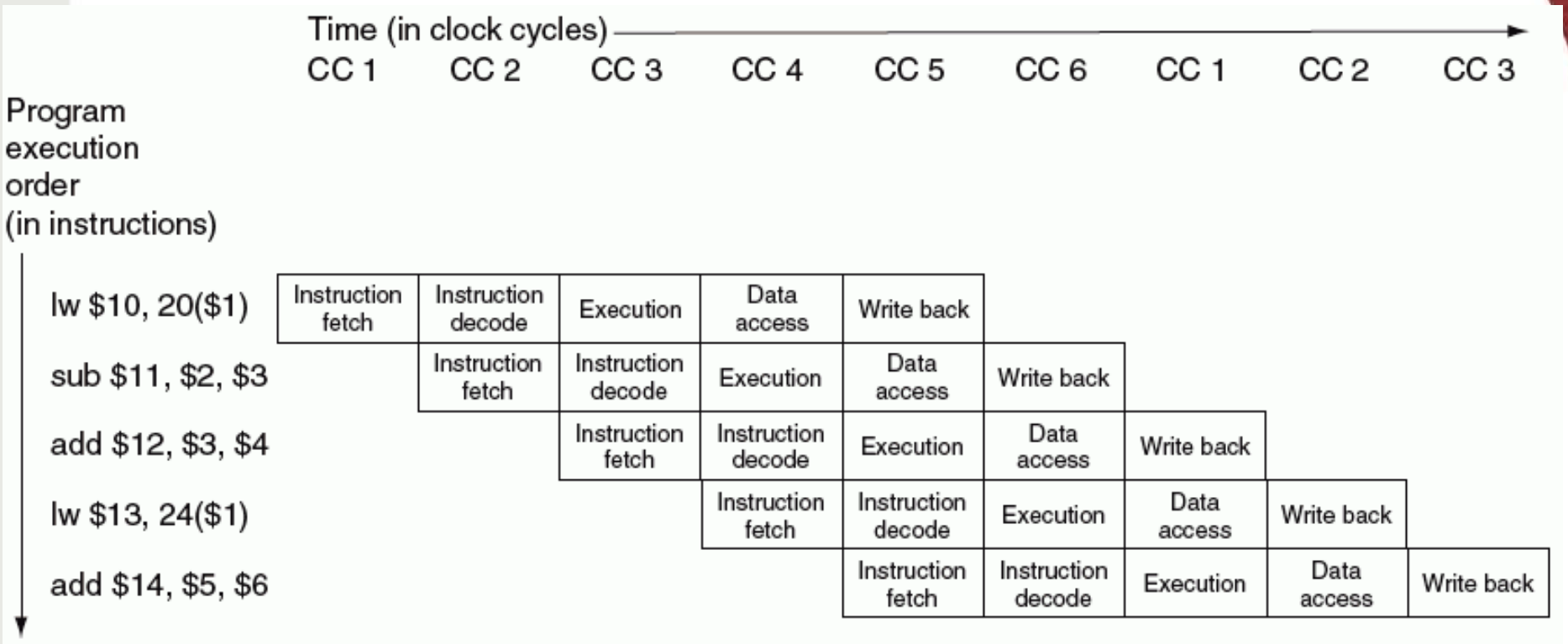
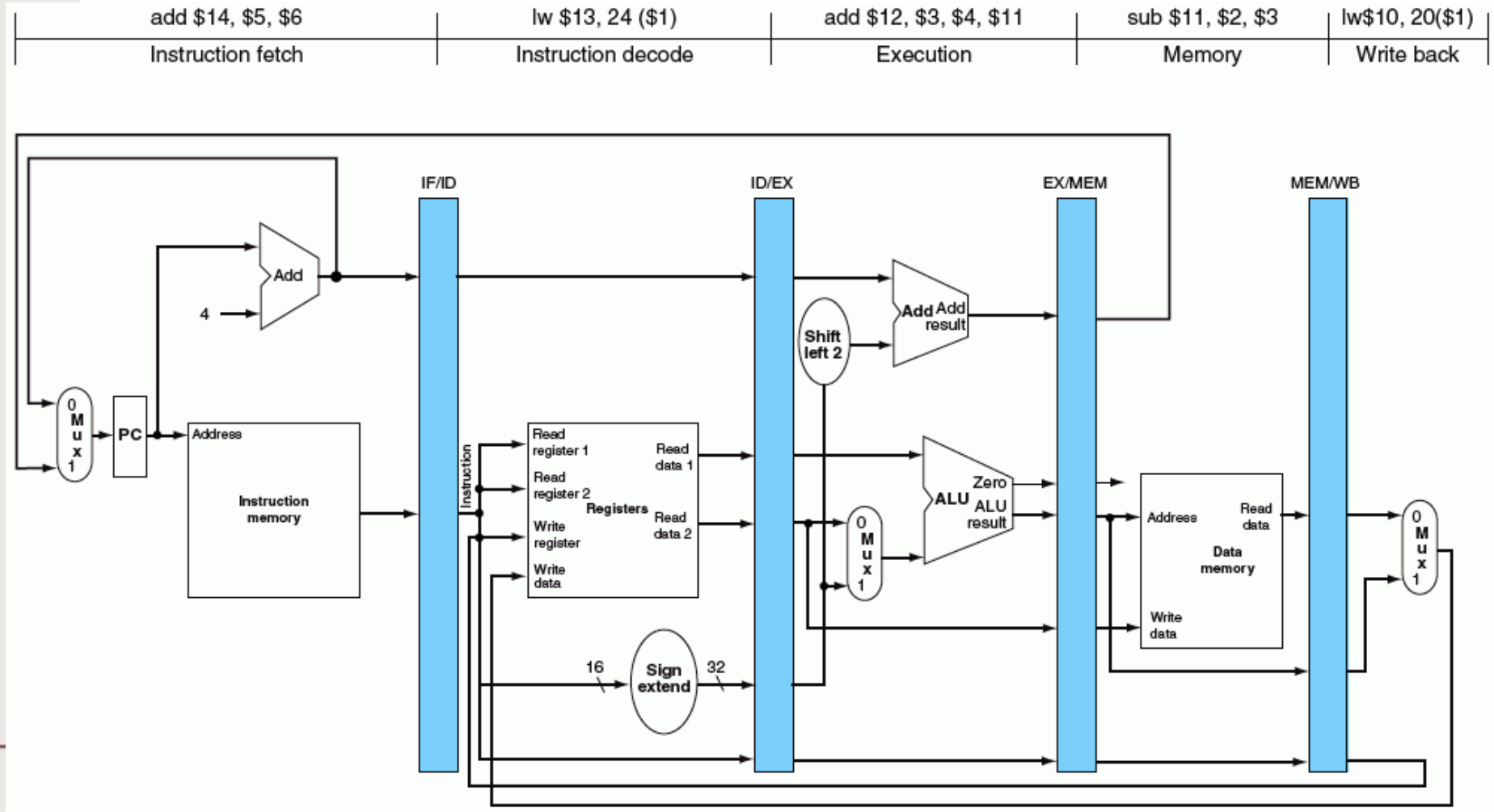


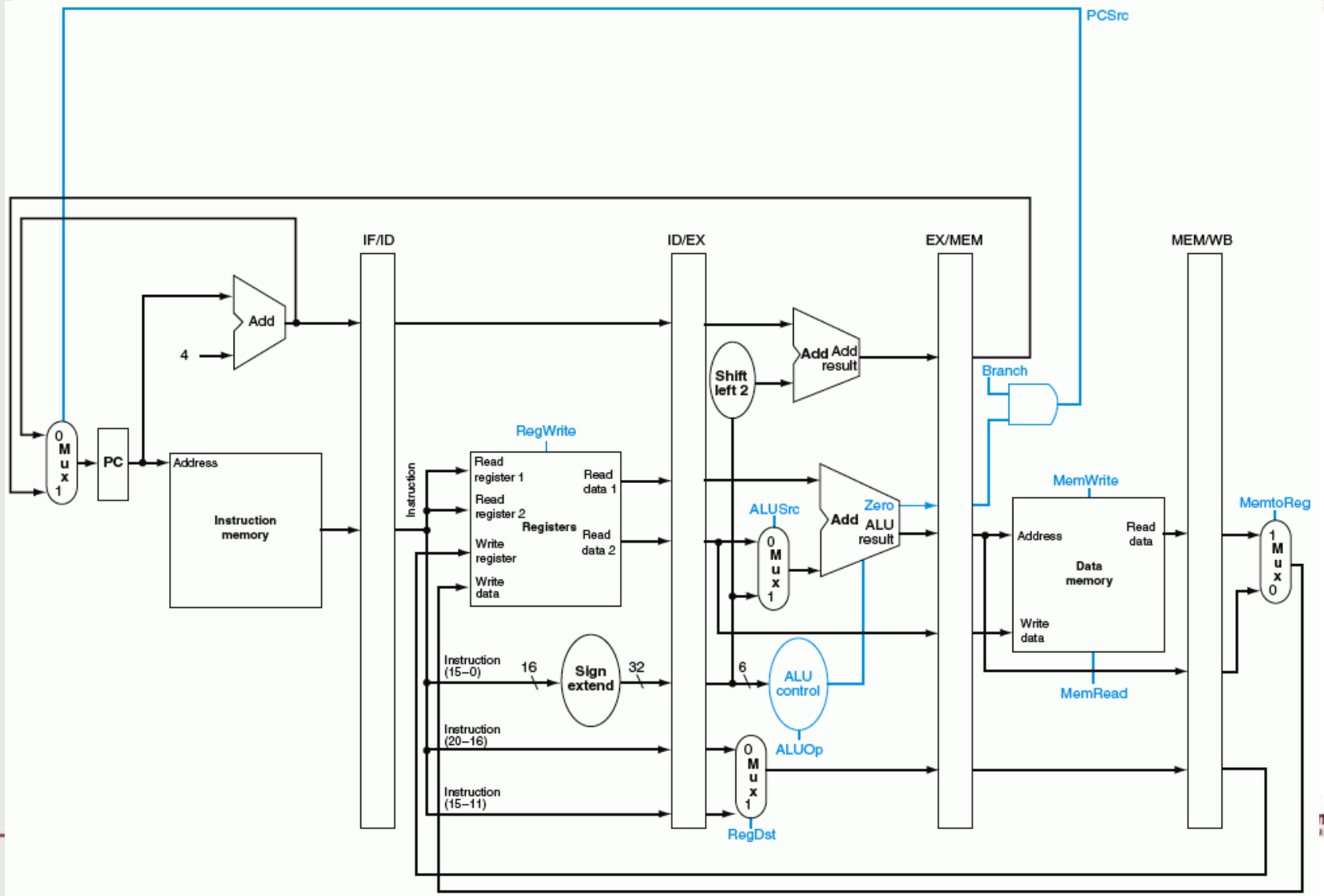
Diagrama de Único Ciclo de Clock

- **Mostra o estado do pipeline em um dado ciclo**

  Identifica recursos utilizados por cada instrução no dado ciclo



Implementando Pipeline: Sinais de Controle



Implementando o Controle do Pipeline

- Sinais de controle derivados da instrução
  - Como na implementação monociclo

- Boa parte dos sinais de controle da implementação monociclo serão aproveitados
  - ALU
  - Desvios
  - Multiplexadores para escolha do fonte do dado do registrador destino, etc

Reagrupando Sinais de Controle da Implementação Monociclo

- Sinais de controle são essencialmente os mesmos, porém precisam ser repassados aos estágios juntamente com a instrução

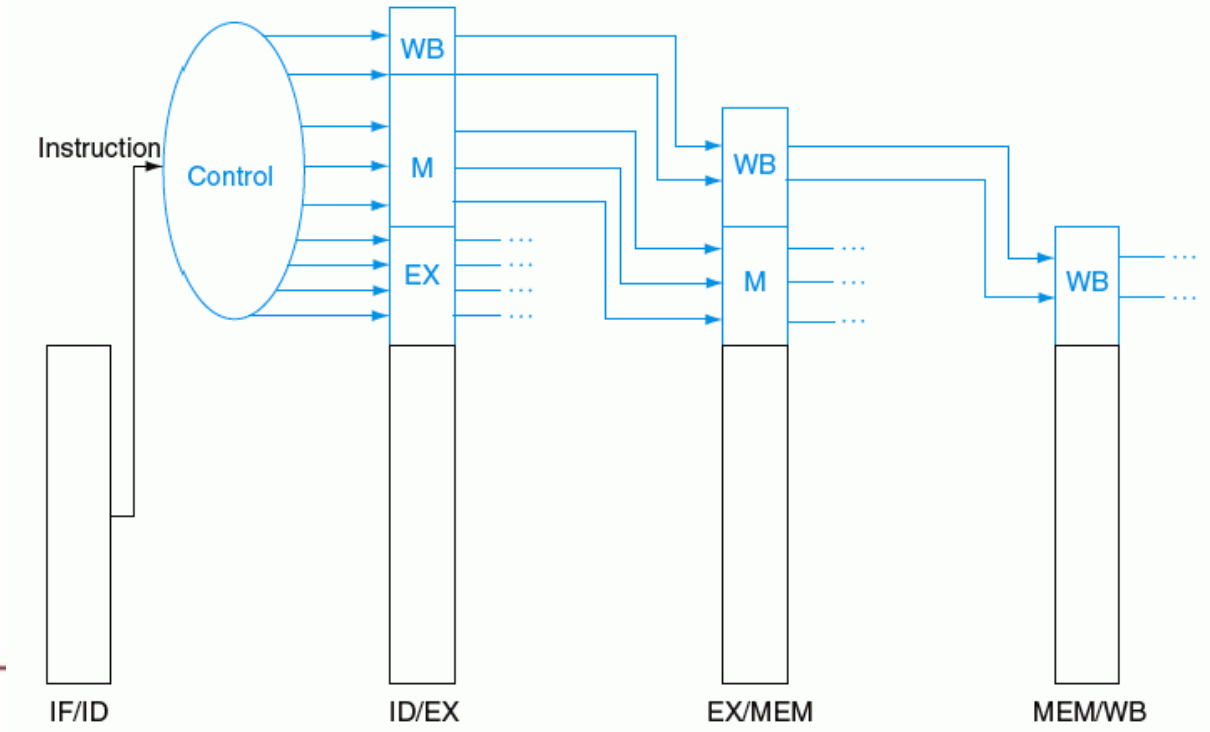
Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg Write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Como Controlar Cada Estágio?

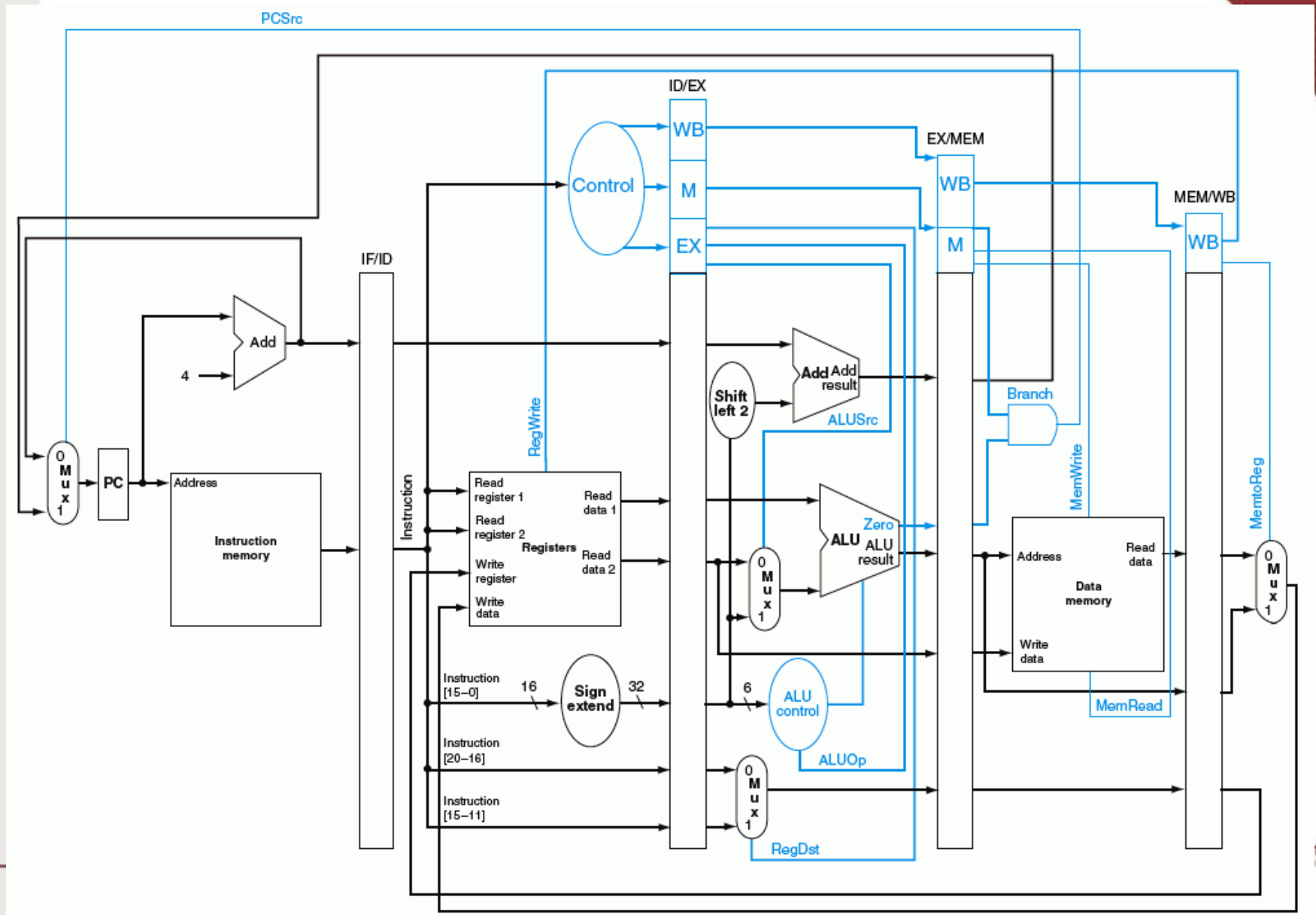
- **Sinais de controle são repassados aos registradores do pipeline**

  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



Pipeline e Projeto de ISA

- **ISA do MIPS projetada para pipeline**
  - Todas as instruções são de 32 bits
    - Mais fácil de buscar e decodificar em um ciclo
    - Contra-exemplo Intel x86: tem instruções variando de 1 a 17 bytes
  - Poucas instruções e instruções regulares
    - Permite decodificação e leitura de registradores em um estágio
  - Endereçamento do Load/store
    - Permite cálculo de endereço no terceiro estágio e acesso no quarto estágio
- **Projeto de ISA afeta a complexidade de implementação do pipeline**

Algumas Análises sobre Pipeline ...

- **Pode aumentar o custo de hardware:**

  Duplicação de hardware para evitar que instruções diferentes utilizem o mesmo recurso em um ciclo

  - Ex: Memória

  Maior quantidade de registradores para guardar estado de processamento da instrução

- **Hardware pode ser utilizado mais eficientemente**

  Evita que recursos fiquem ociosos

Algumas Análises sobre Pipeline ...

- **Melhora de desempenho por utilizar pipeline:**
  - estágios com a mesma duração
  - instruções independentes de resultados calculados em instrução anterior
  - execução sequencial das instruções

... mas as características acima quase sempre não são satisfeitas:

- ***dependência de dados***
- ***instruções de desvio***

Conflitos

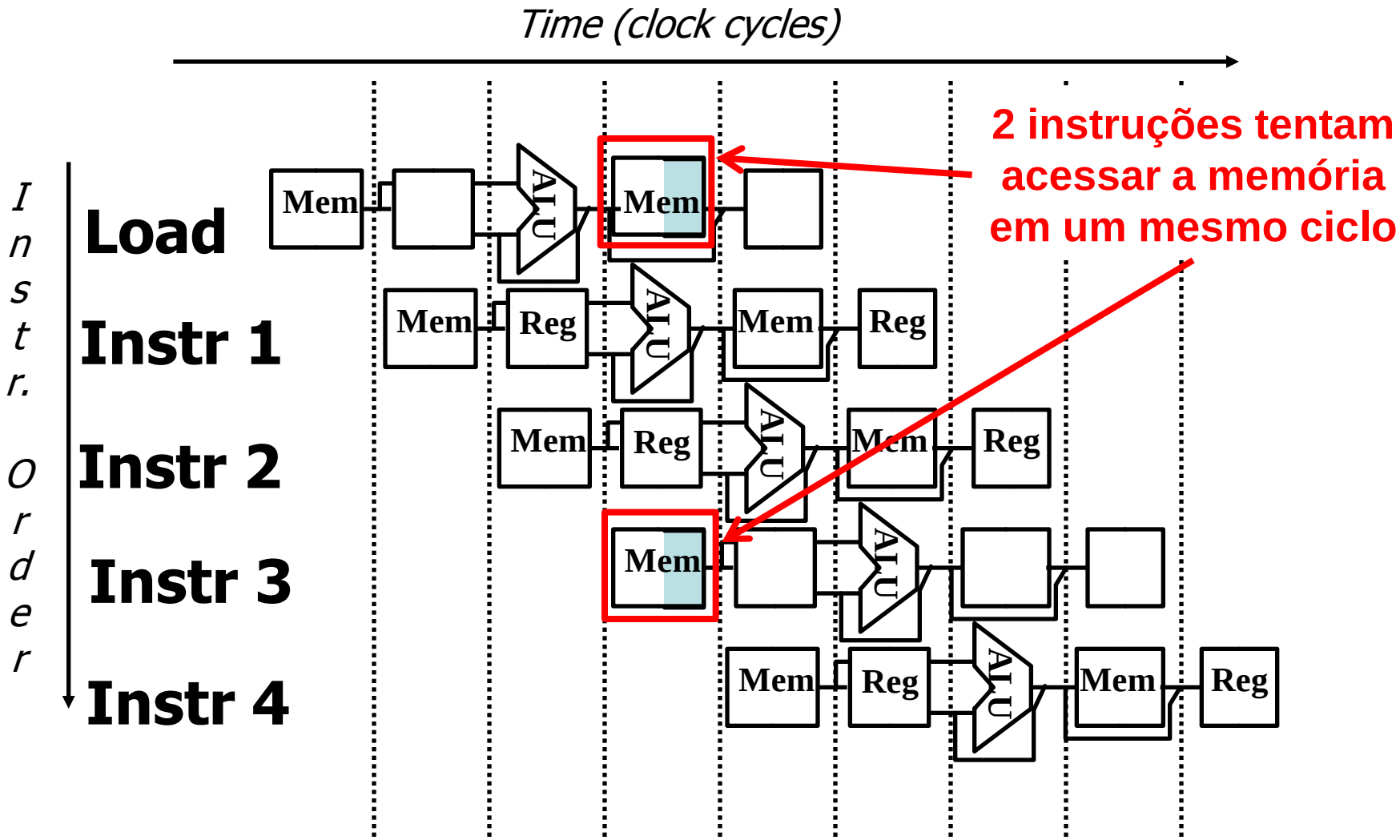
- Situações que evitam que uma nova instrução seja iniciada no próximo ciclo
- Tipos:
  - Estruturais
    - Recurso necessário para execução de uma instrução está ocupado
  - Dados
    - Dependência de dados entre instruções
  - Controle
    - Decisão da próxima instrução a ser executada depende de uma 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

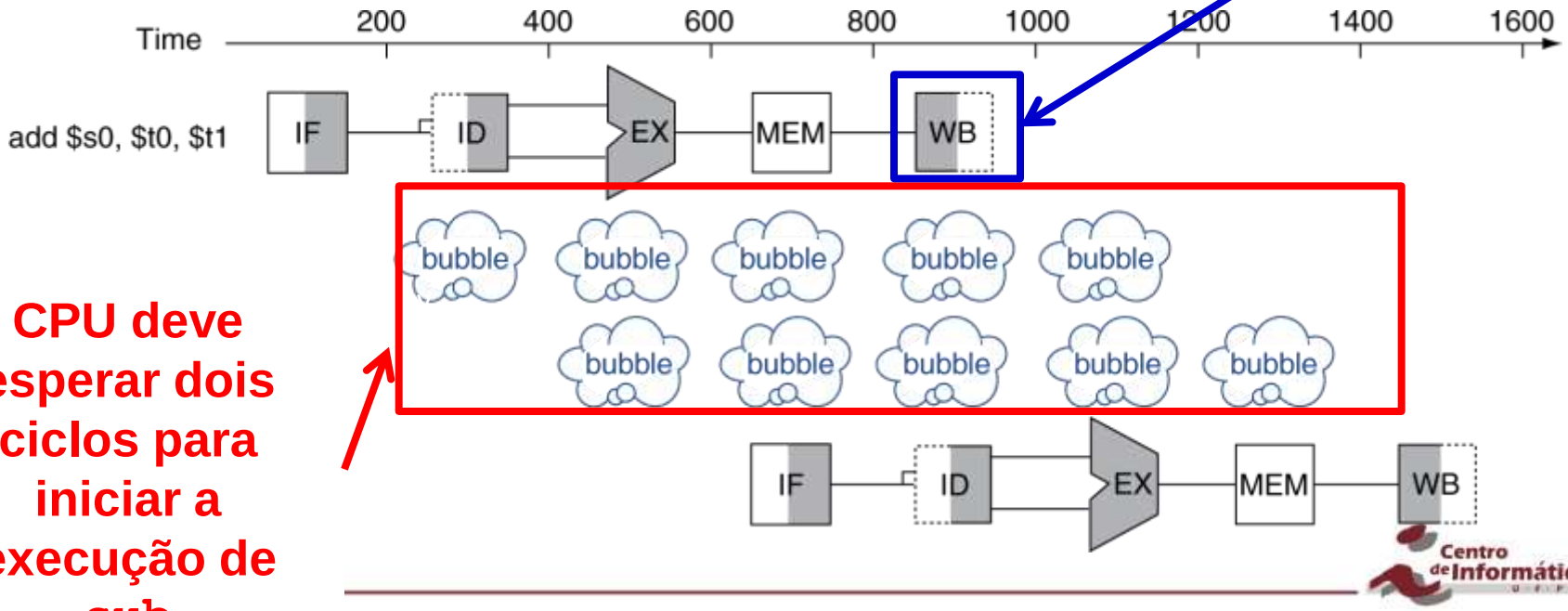


Conflito de Dados

- 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



CPU deve esperar dois ciclos para iniciar a execução de `sub`

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

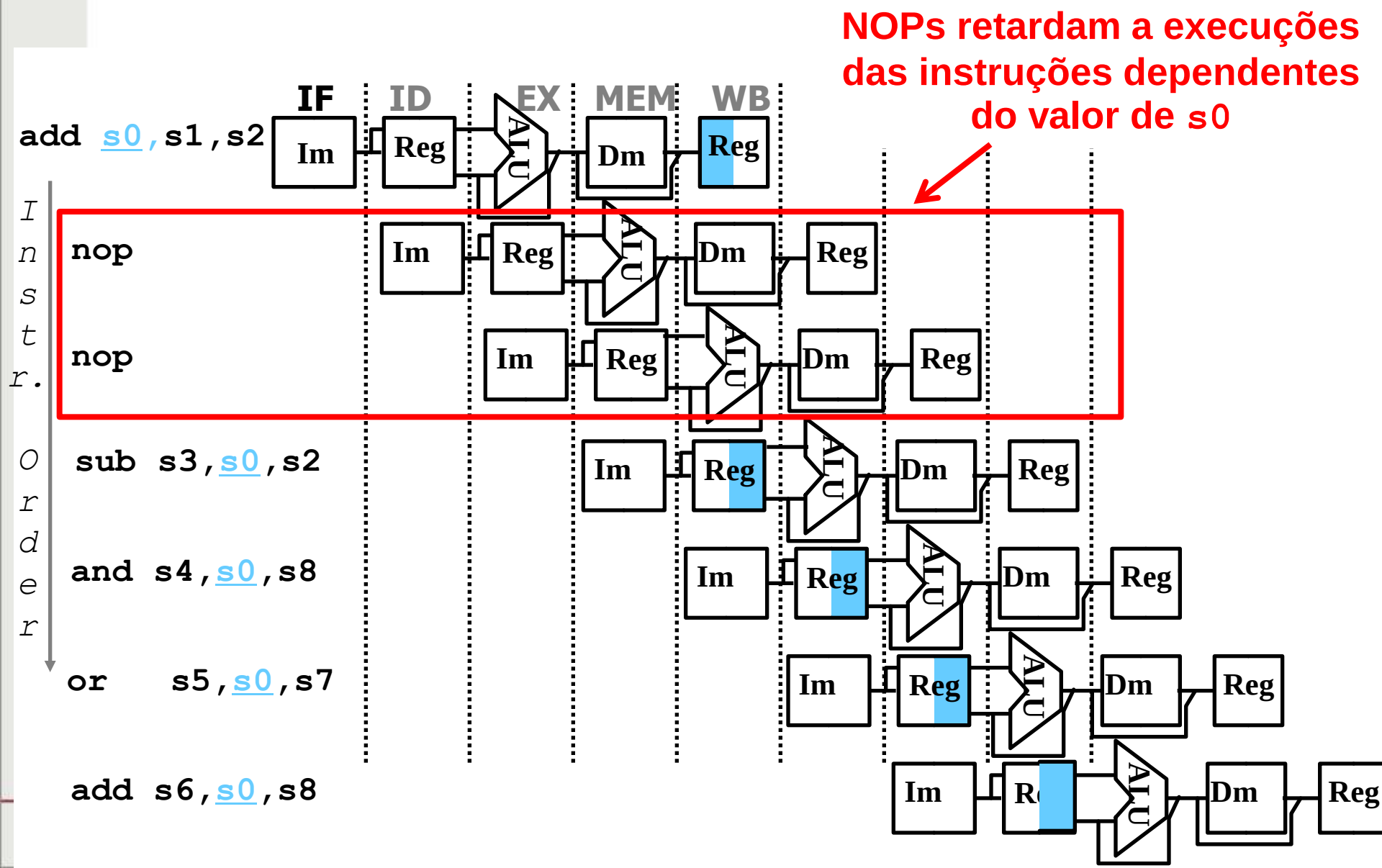
- Compilador/Montador deve identificar conflitos de dados e evitá-los inserindo NOPs no código

Conflitos de dados

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

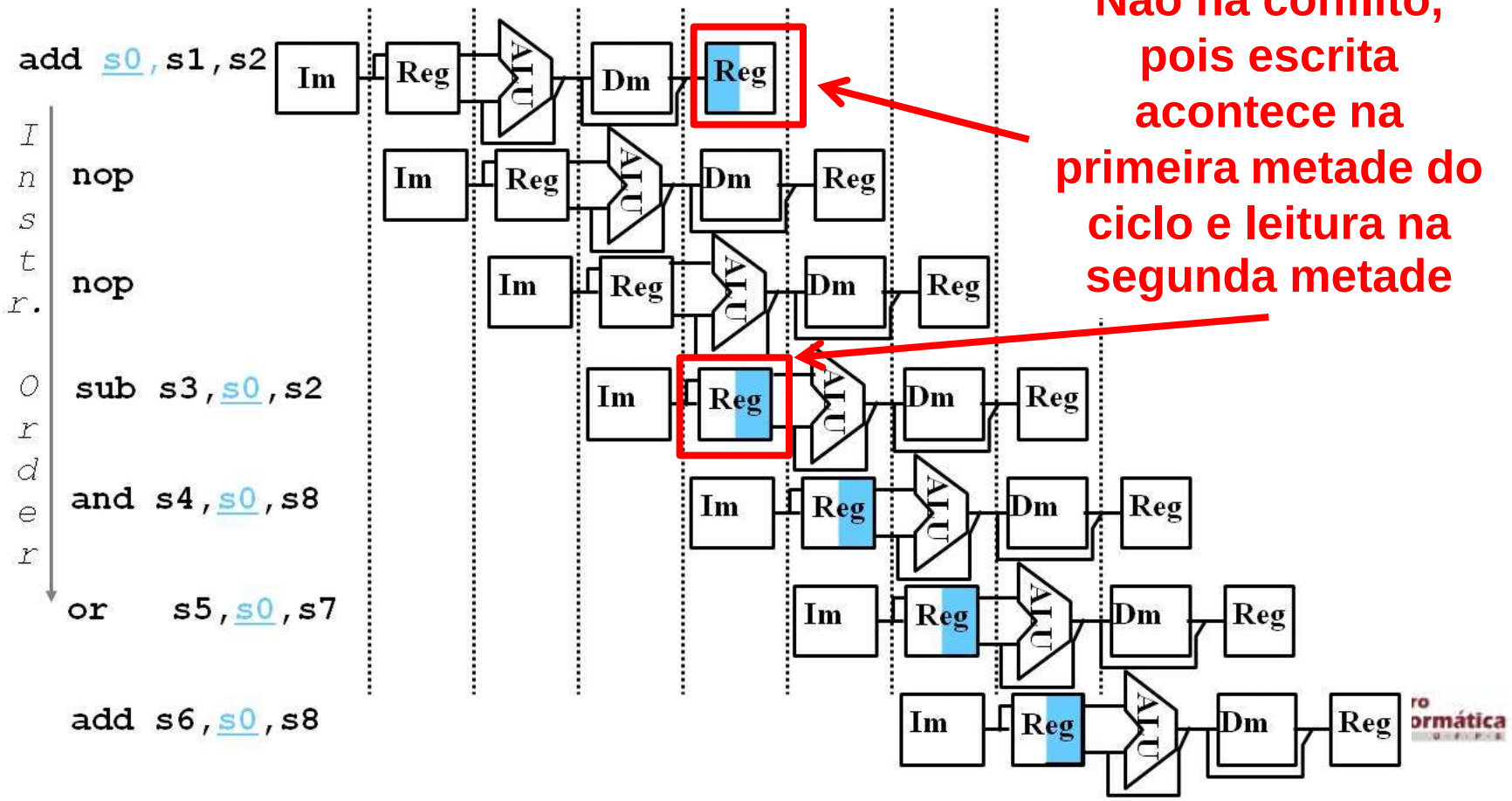
```
add $s0,$s1,$s2
nop
nop
sub $s3,$s0,$s2
and $s4,$s0,$s8
or  $s5,$s0,$s7
add $s6,$s0,$s8
```

Inserção de NOPs



Escrita e Leitura de Banco de Registradores no Mesmo Ciclo

- Banco de registradores permite a leitura de dois registradores e a escrita de um registrador simultaneamente



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

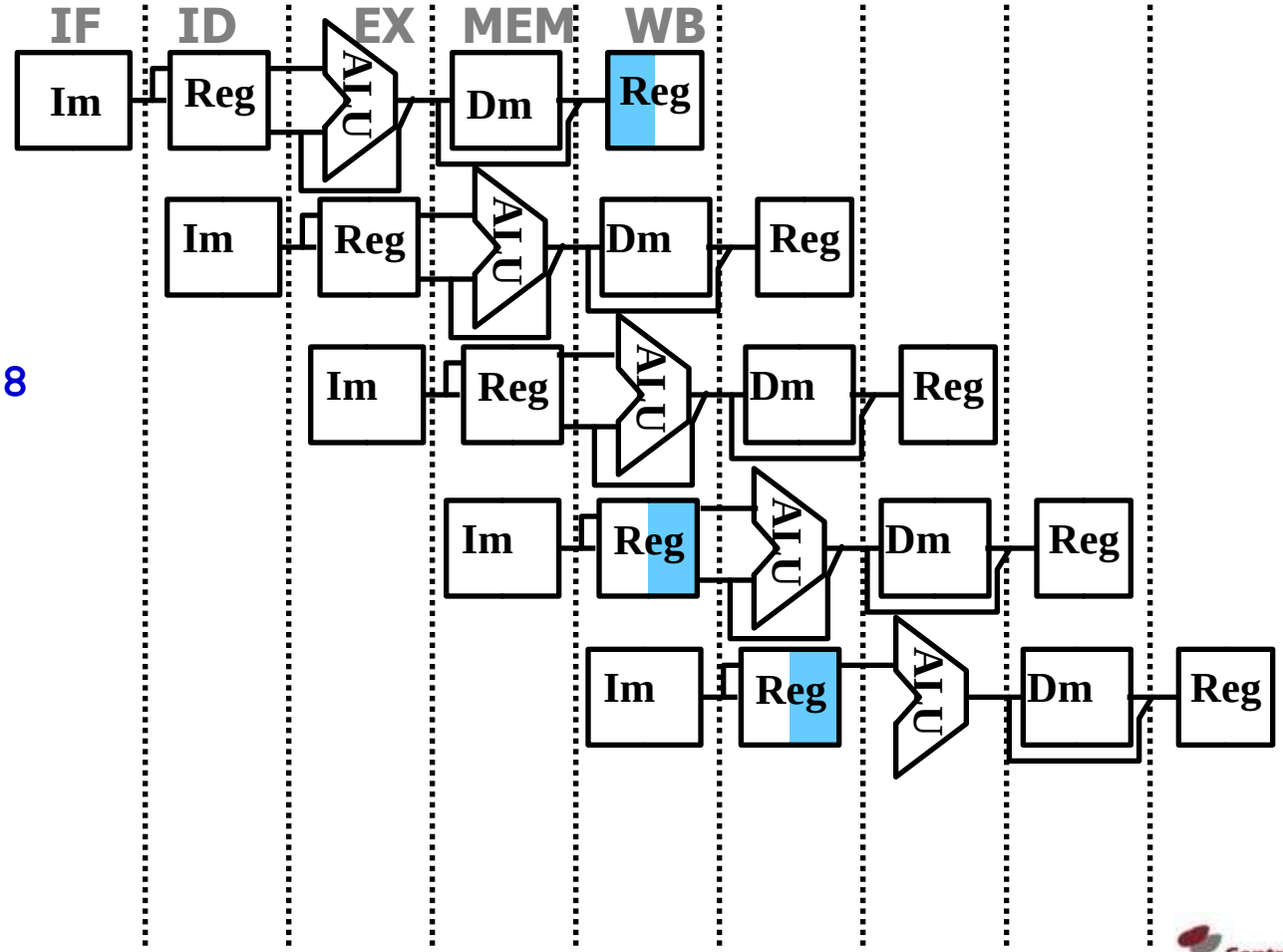
```
add $s0,$s1,$s2
sub $s3,$s0,$s2
and $s4,$s0,$s8
or  $s5,$s1,$s7
add $s6,$s2,$s8
```

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

Re-arrumação do Código

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

Instr. Order

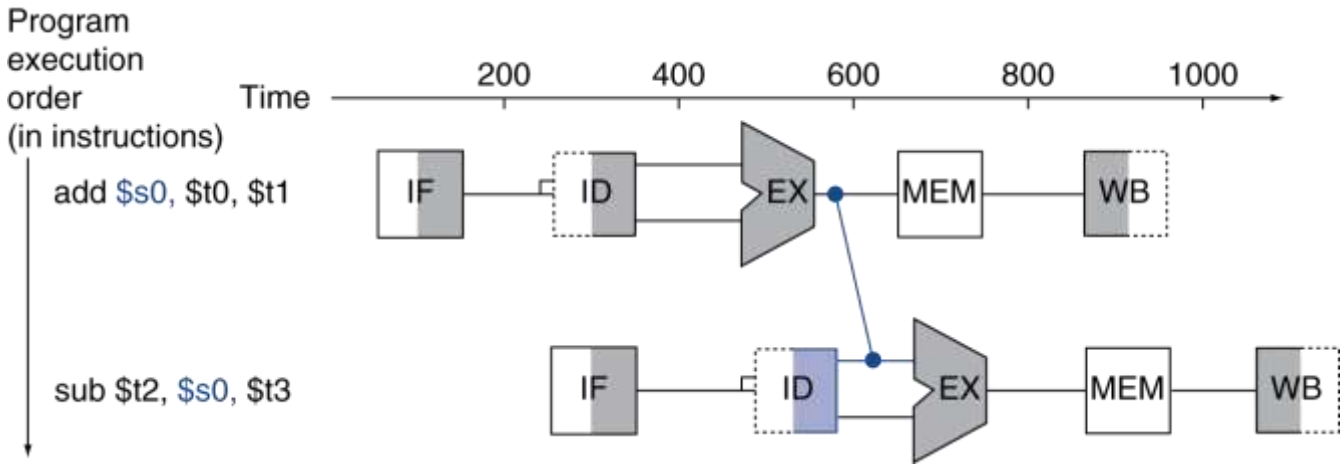


Análise de Soluções em SW para Conflitos de Dados

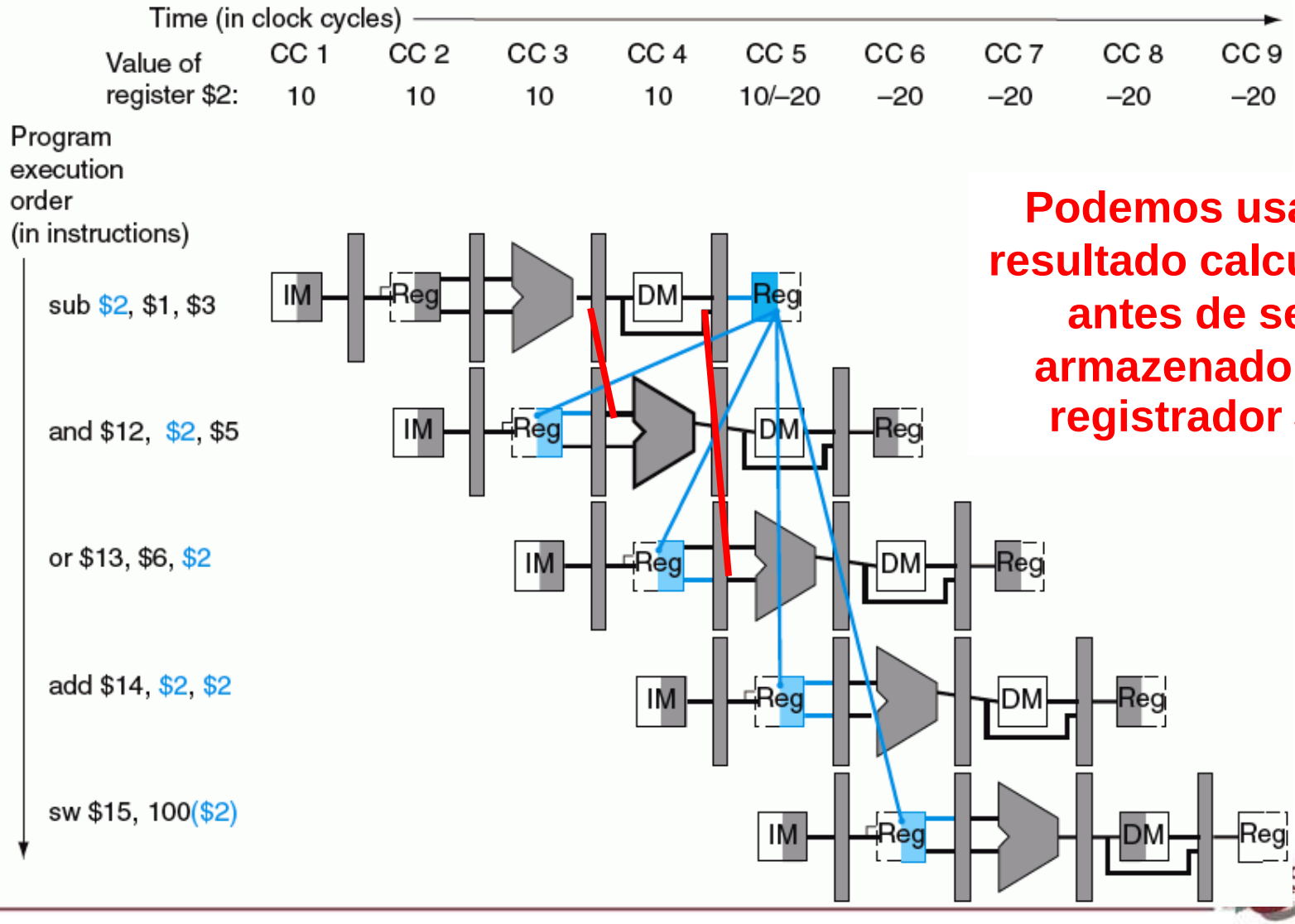
- Requerem compilador/montador “inteligente” que detecte conflitos de dados
  - Modifica o código para evitar conflitos
  - Mantem funcionamento correto do programa
  - Implementação do Compilador/Montador requer bom conhecimento do pipeline
- Inserção de NOPs
  - Insere retardos no programa
  - Degrada desempenho do sistema
- Re-arrumação de código
  - Não compromete desempenho
  - Mais complexidade na implementação do compilador/montador

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



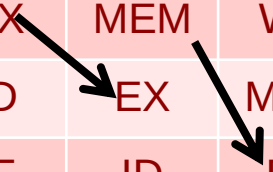
Dependências e Forwarding



Podemos usar o resultado calculado antes de ser armazenado no registrador $2

Usando Forwarding (Representação Tradicional)

	1	2	3	4	5	6	7	8	9
I1	IF	ID	EX	MEM	WB				
I2		IF	ID	EX	MEM	WB			
I3			IF	ID	EX	MEM	WB		
I4				IF	ID	EX	MEM	WB	



I1: sub $2, $1,$3
I2: and $12, $2, $5
I3: or $13, $6, $2
I4: and $14, $2, $2

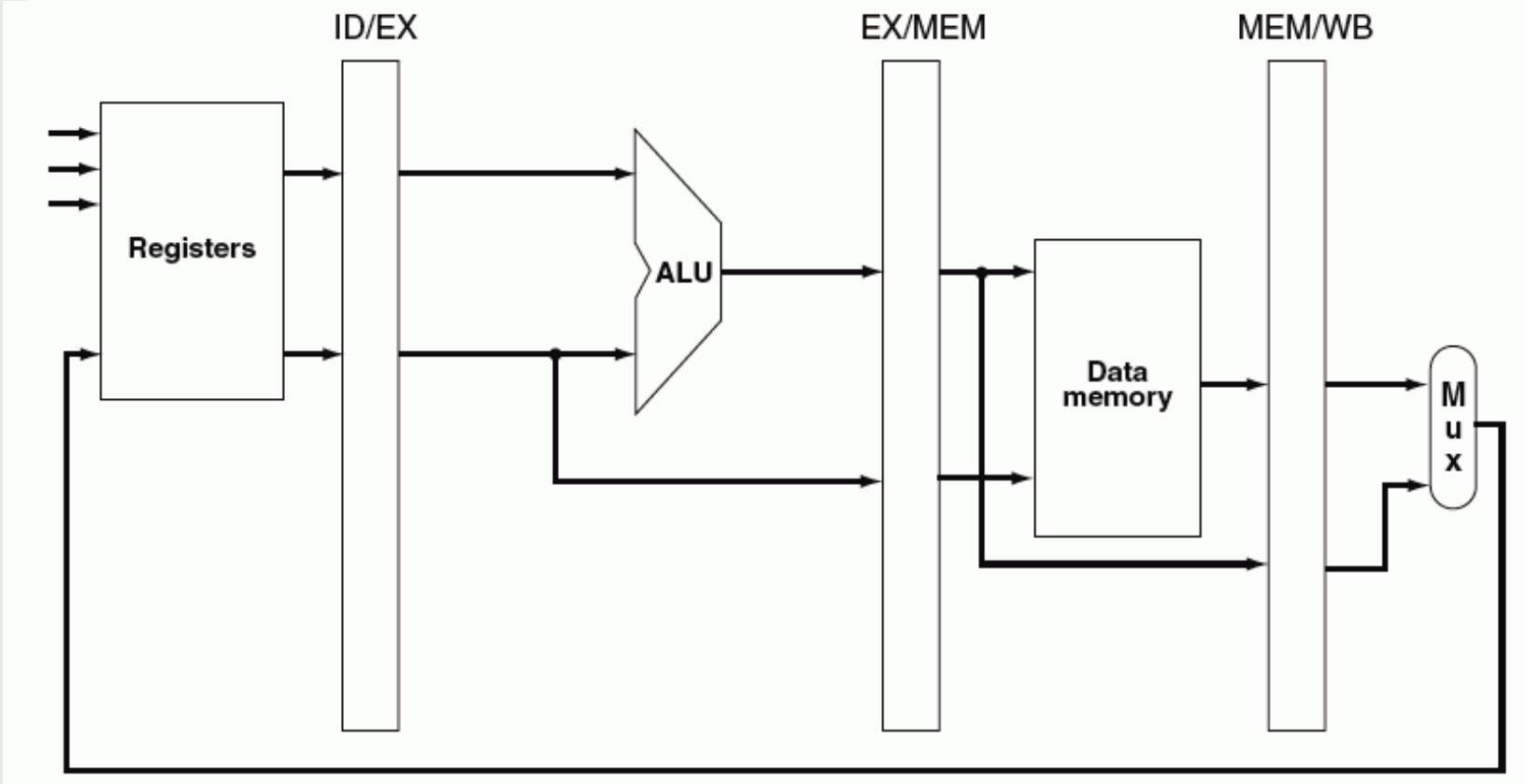
Como Implementar Forwarding?

- Idéia é acrescentar HW com uma lógica capaz de detectar conflitos de dados e controlar unidade de processamento para realizar o forwarding

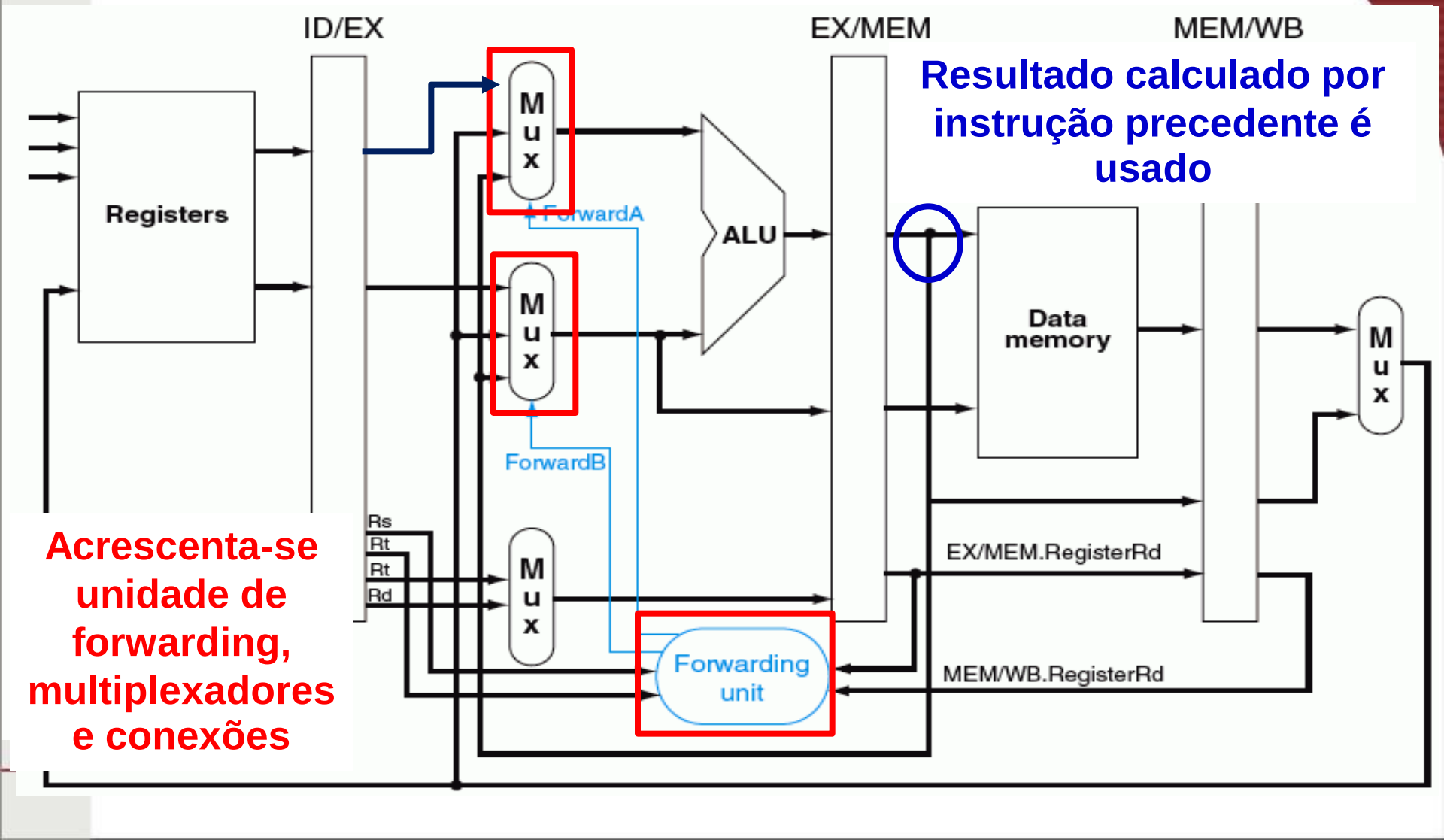
- Deve-se acrescentar mais conexões para permitir que resultados possam ser utilizados antes de escritos no banco de registradores

- Possivelmente, acrescenta-se multiplexadores para que outros estágios possam selecionar a fonte do operando
  - Banco de registradores ou resultado gerado por outra instrução anterior no pipeline

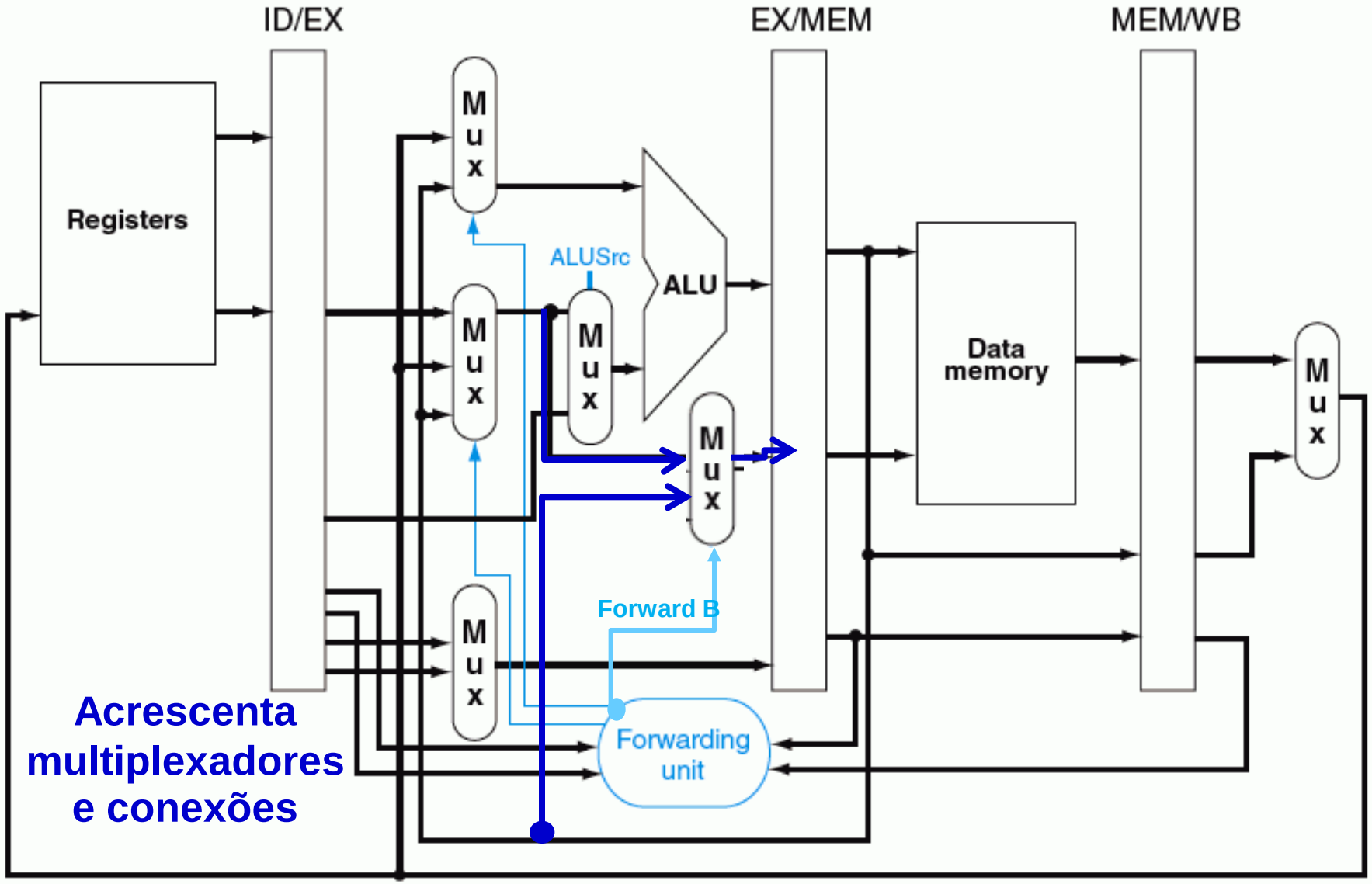
Datapath Simplificado Sem Forwarding



Datapath Simplificado Com Forwarding (Sem Considerar store (sw) e Imediato)



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



Como Detectar Necessidade de Forward?

- **Passar números de registradores ao longo do pipeline**

  Exemplo:. ID/EX.RegisterRs = número do registrador Rs armazenado no registradorID/EX do pipeline

- **Números dos registradores que são operandos da ALU no estágio EX são dados por**

  ID/EX.RegisterRs, ID/EX.RegisterRt

- **Conflitos ocorrem quando**

  1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
  1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

  Fwd do registrador EX/MEM

  2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

  Fwd do registrador MEM/WB

Como Detectar Necessidade de Forward?

- **Mas só vai dar um forward se a instrução for escrever em um registrador**
  - EX/MEM.RegWrite, MEM/WB.RegWrite
- **E somente se o Rd para a instrução não for $zero**

  EX/MEM.RegisterRd ≠ 0,
  MEM/WB.RegisterRd ≠ 0

Condições de Forwarding

- **Forwarding do resultado do estágio EX**

  if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
  ForwardA = 10

  if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
  ForwardB = 10

- **Forwarding do resultado do estágio MEM**

  if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
  and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
  ForwardA = 01

  if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
  and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
  ForwardB = 01

Conflito Duplo de Dados

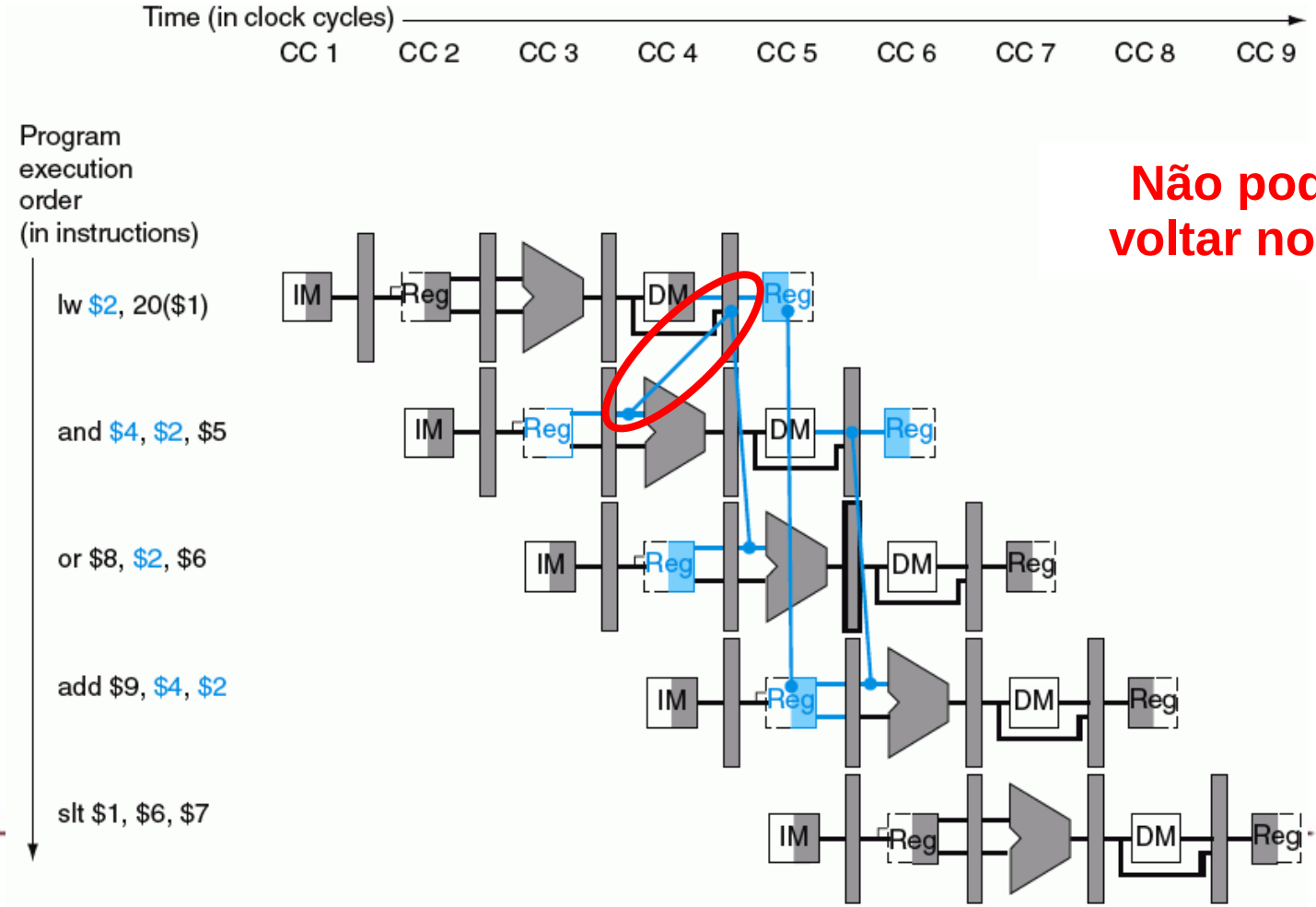
- Considere a sequência:

```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```

- O terceiro add deve pegar resultado do segundo add, não do primeiro
  - Resultado do primeiro add está em MEM/WB
  - Resultado do segundo add está em EX/MEM
- **Deve-se reescrever condição de forward do estágio MEM**
  - Somente se condição de forward de EX for falsa

Conflito de Dados Pelo Uso do Load

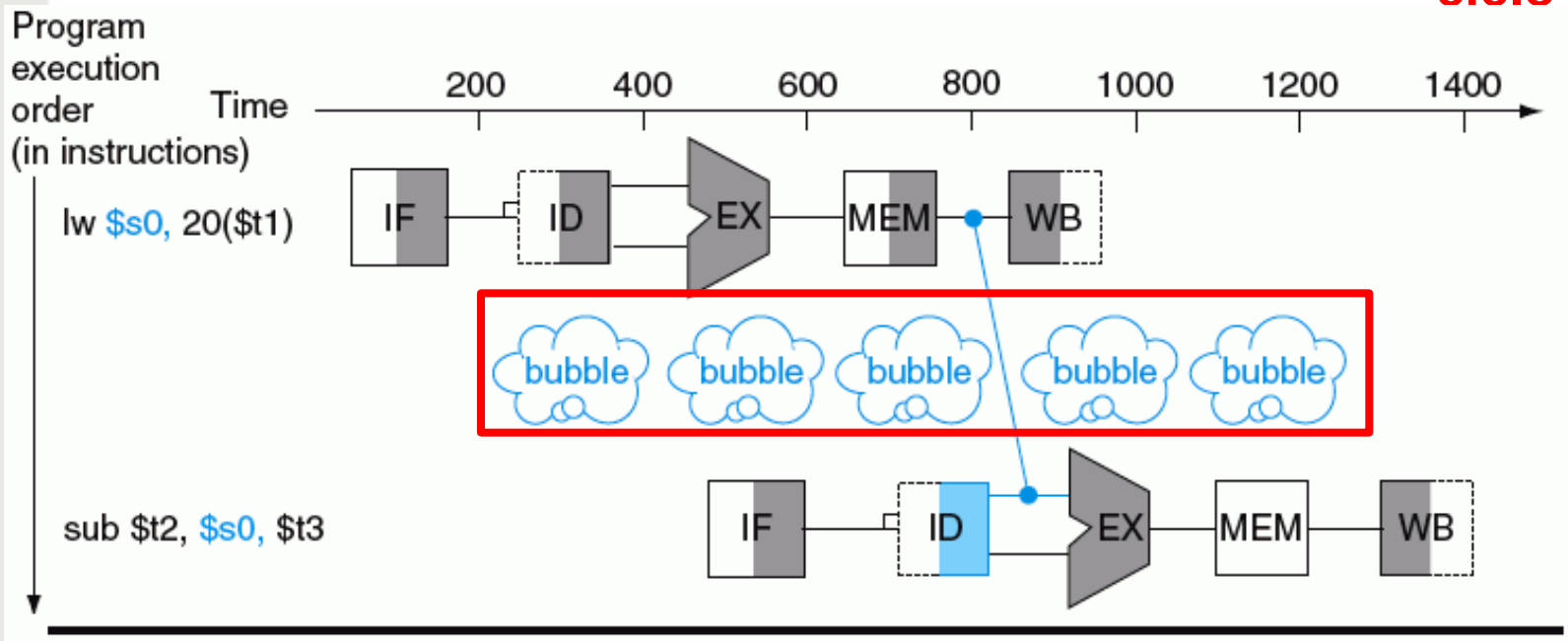
- Nem sempre se pode utilizar forwarding
  - Se o valor ainda não tiver sido computado quando necessário



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**

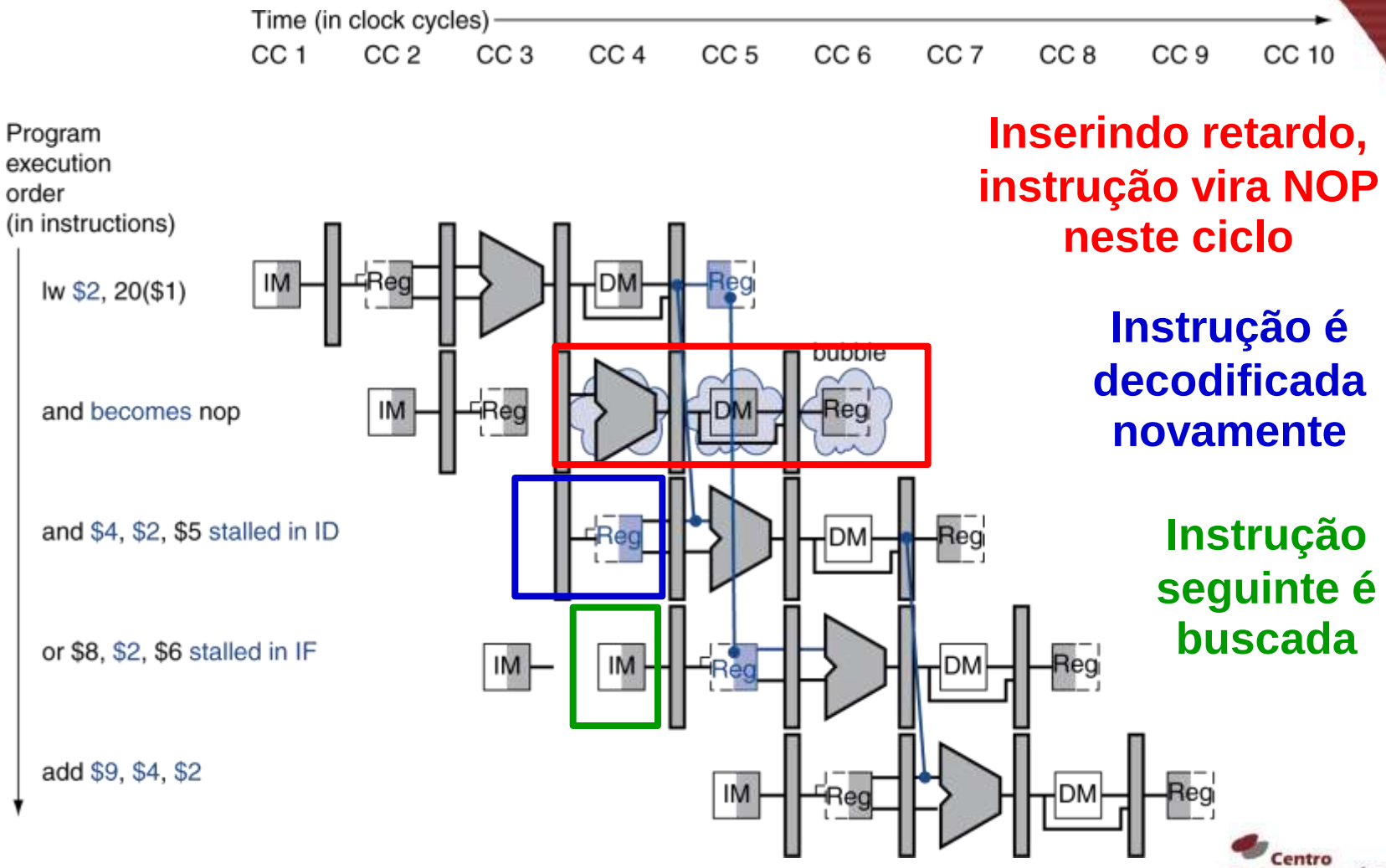
  ID/EX.MemRead and
  ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
  (ID/EX.RegisterRt = IF/ID.RegisterRt))

- **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 MEM

Inserindo um Retardo no Pipeline



Inserindo retardo, instrução vira NOP neste ciclo

Instrução é decodificada novamente

Instrução seguinte é buscada

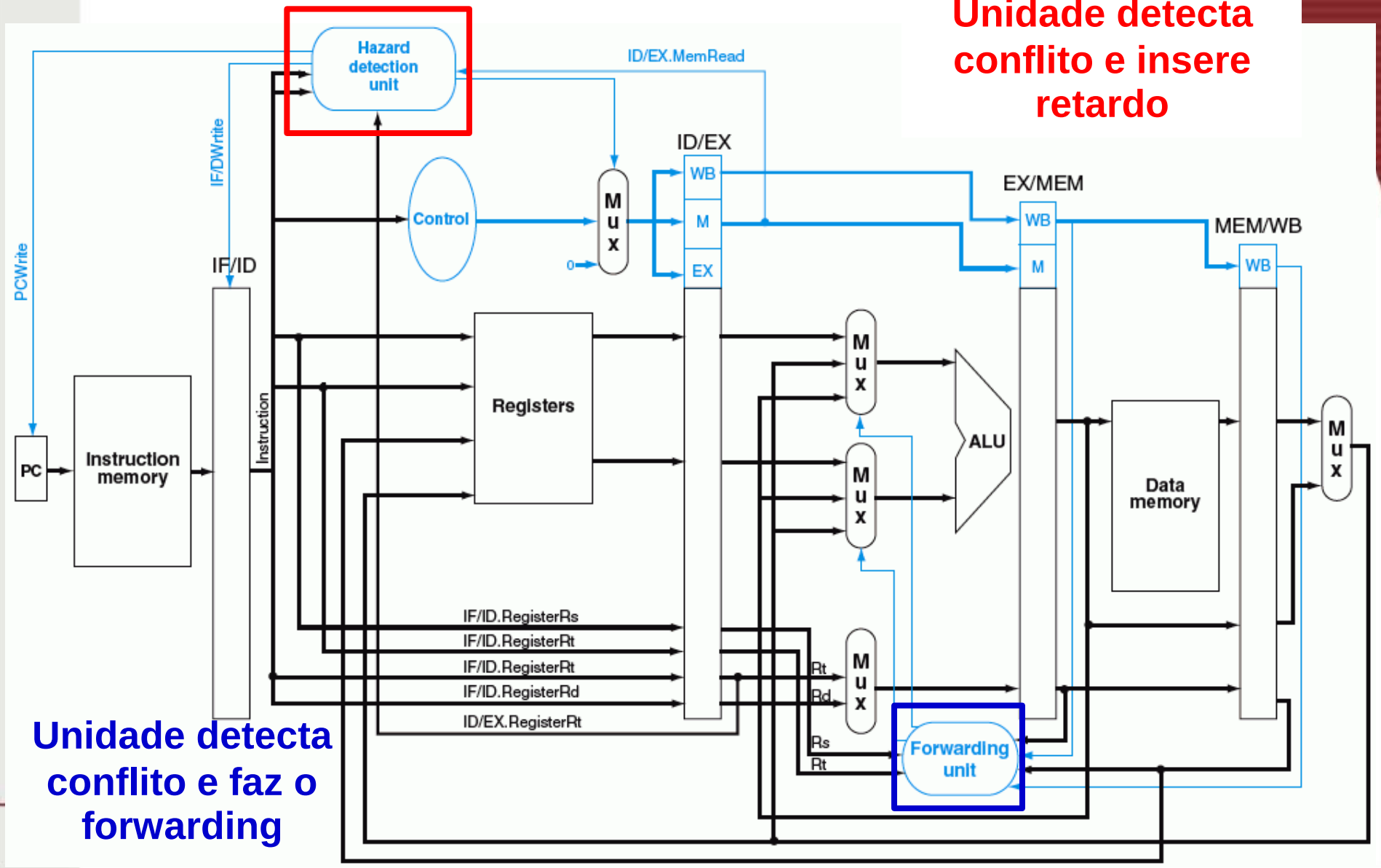
Inserindo um Retardo no Pipeline (Representação Tradicional)

	1	2	3	4	5	6	7	8	9
I1	IF	ID	EX	MEM	WB				
I2		IF	ID	X	EX	MEM	WB		
I3			IF	X	ID	EX	MEM	WB	
I4					IF	ID	EX	MEM	WB

I1: lw $2, 20($1)
I2: and $4, $2, $5
I3: or $8, $2, $6
I4: add $9, $4, $2

Datapath com Unidades de Retardo e Forwarding

Unidade detecta conflito e insere retardo



Unidade detecta conflito e faz o forwarding

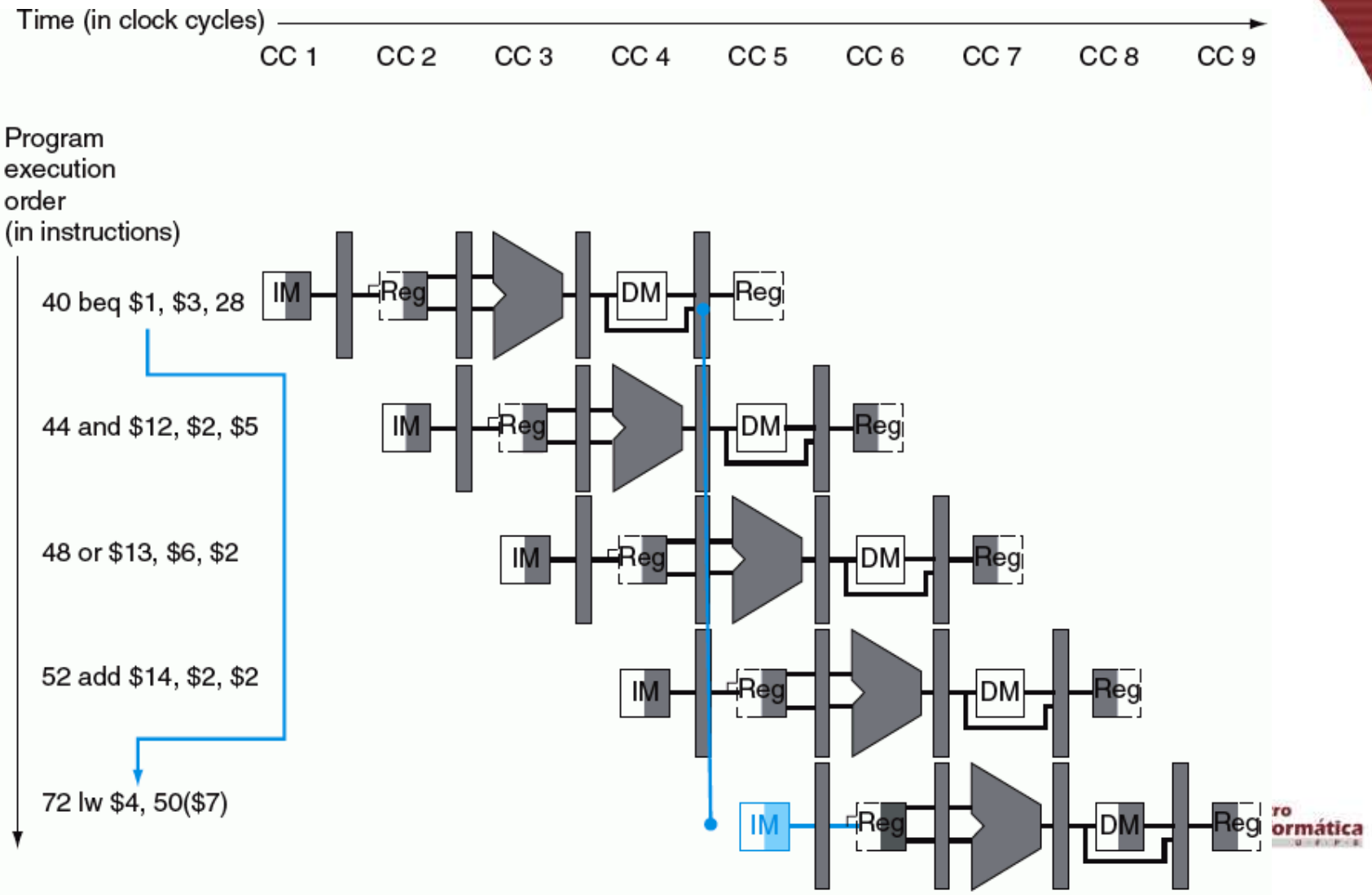
Análise de Soluções em HW para Conflitos de Dados

- Curto-circuito (Forwarding)
  - Não degrada desempenho
  - Requer custo um pouco maior de hardware
- Inserção de Retardos
  - Similar a solução de SW de inserção de NOPs
  - Degrada desempenho do sistema
- Pode-se utilizar os dois métodos em conjunto
  - Deve-se usar inserção de retardos, somente quando não é possível utilizar forwarding

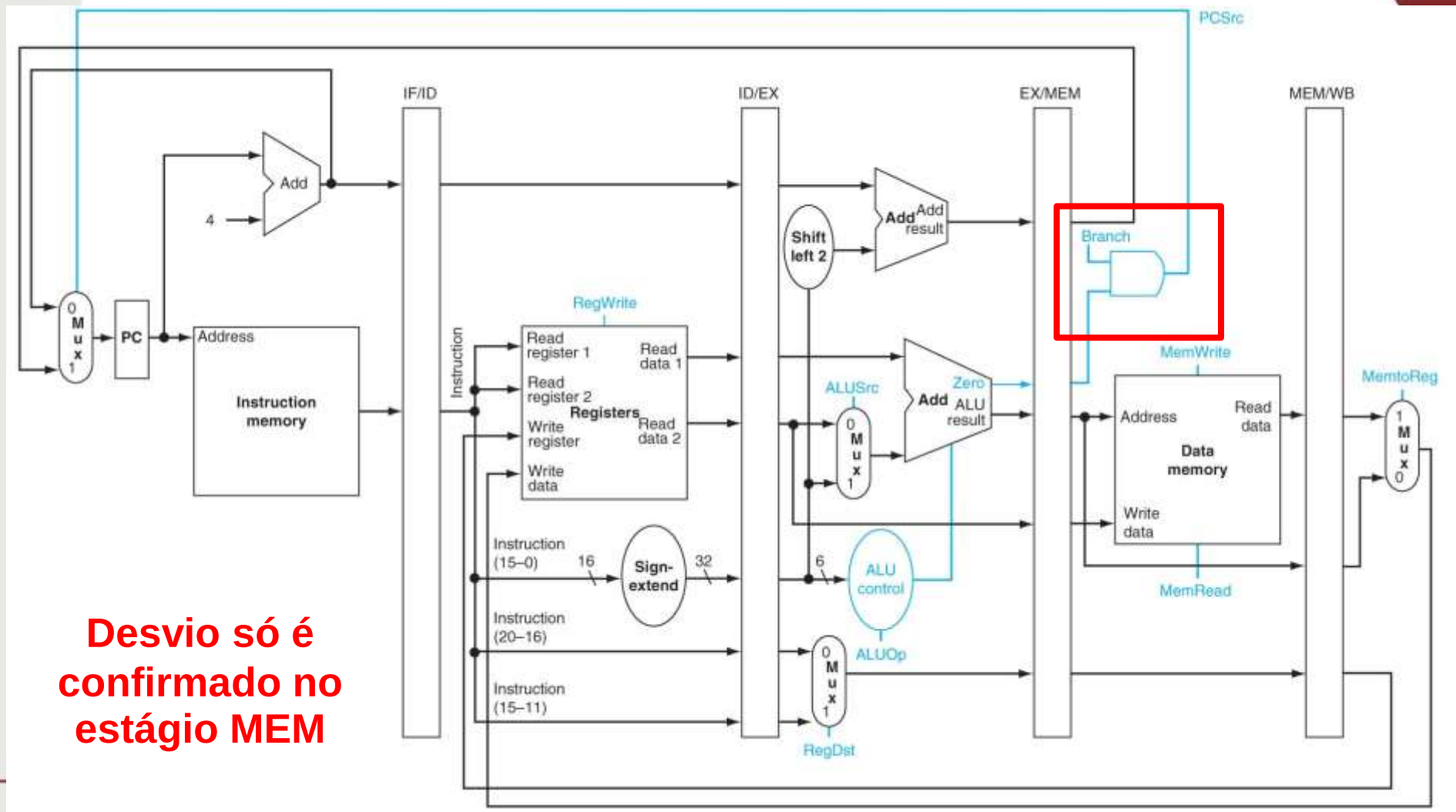
Conflitos de Controle

- **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



Penalidade de um Branch no Pipeline



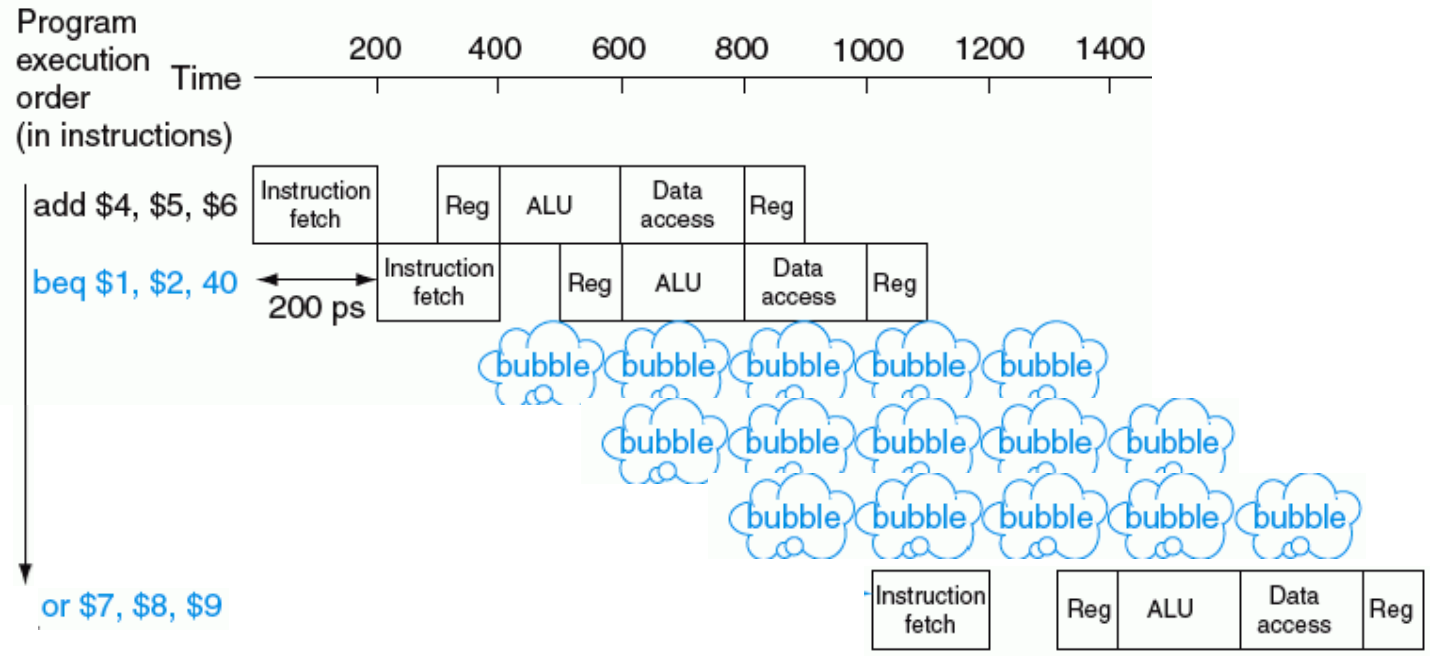
Desvio só é confirmado no estágio MEM

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

Congelamento do Pipeline

- Inserção de retardos até que se saiba se o desvio ocorrerá
- Só depois é que se faz a busca de nova instrução



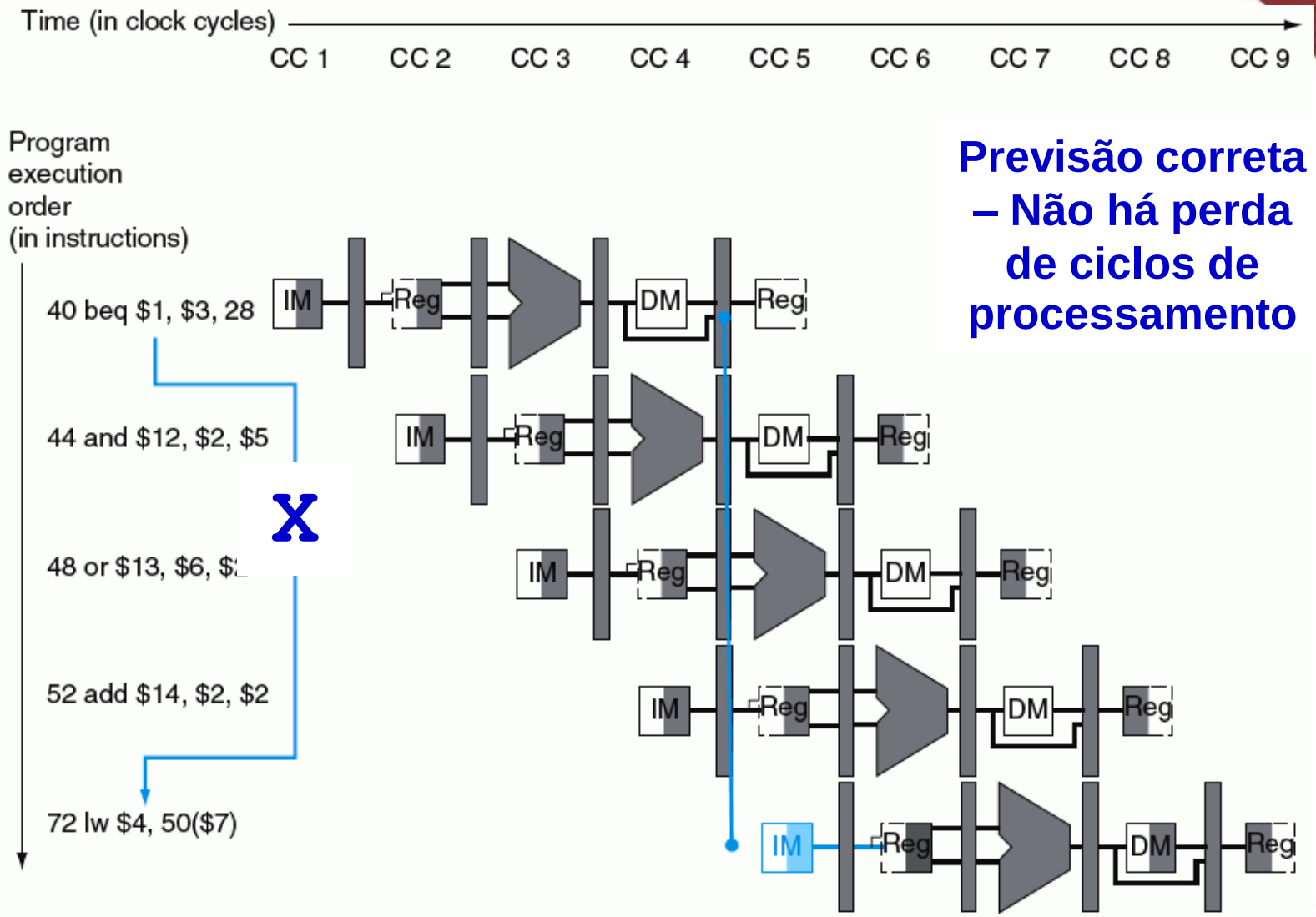
Execução Especulativa

- 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

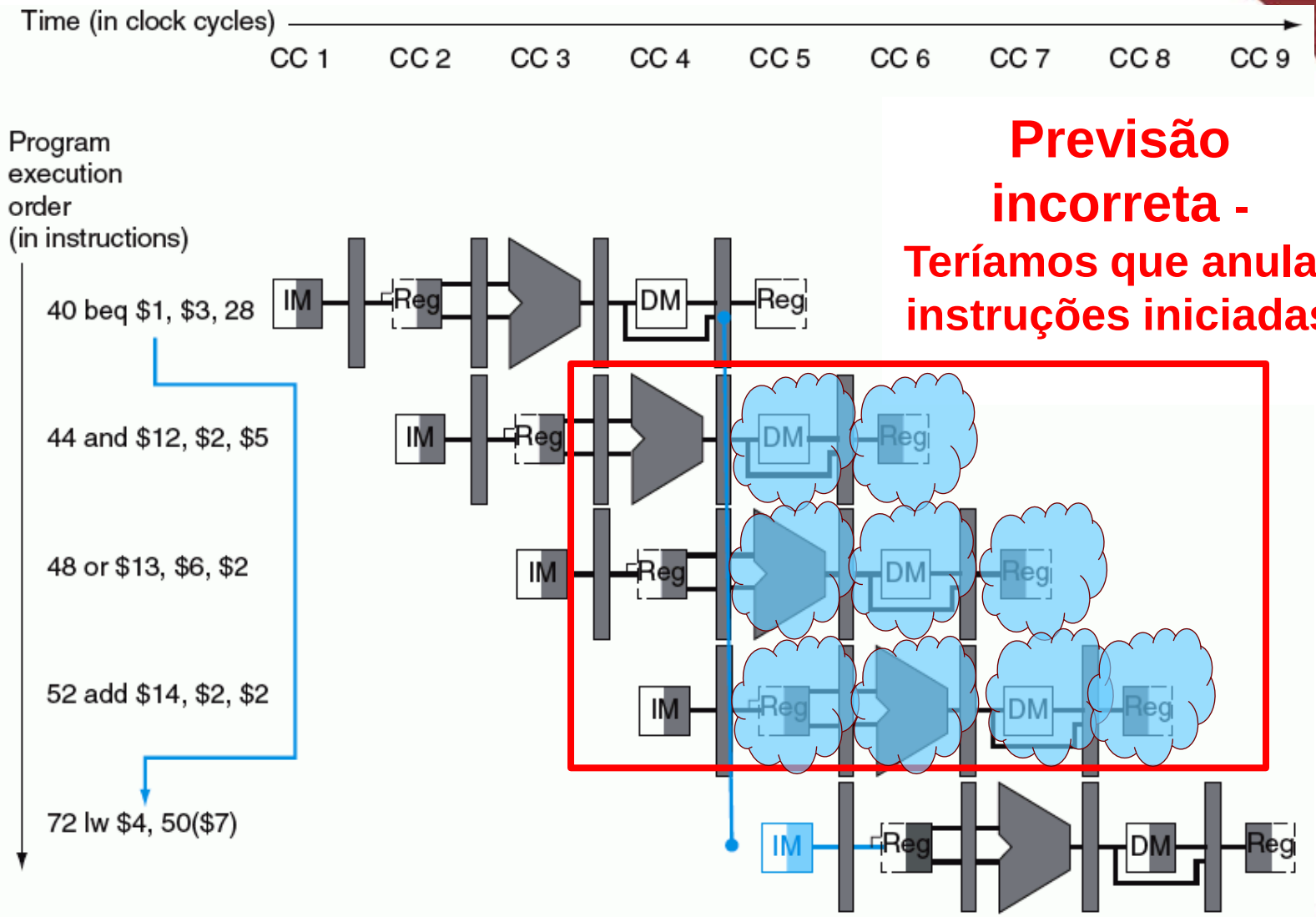
- Previsão pode ser:
  - Estática
    - Nunca haverá desvio, ou sempre haverá desvio
  - Dinâmica
    - De acordo com comportamento do código

Previsão Estática – Nunca Haverá Desvio



Previsão correta – Não há perda de ciclos de processamento

Previsão Estática – Nunca Haverá Desvio



Previsão Estática – Nunca Haverá Desvio

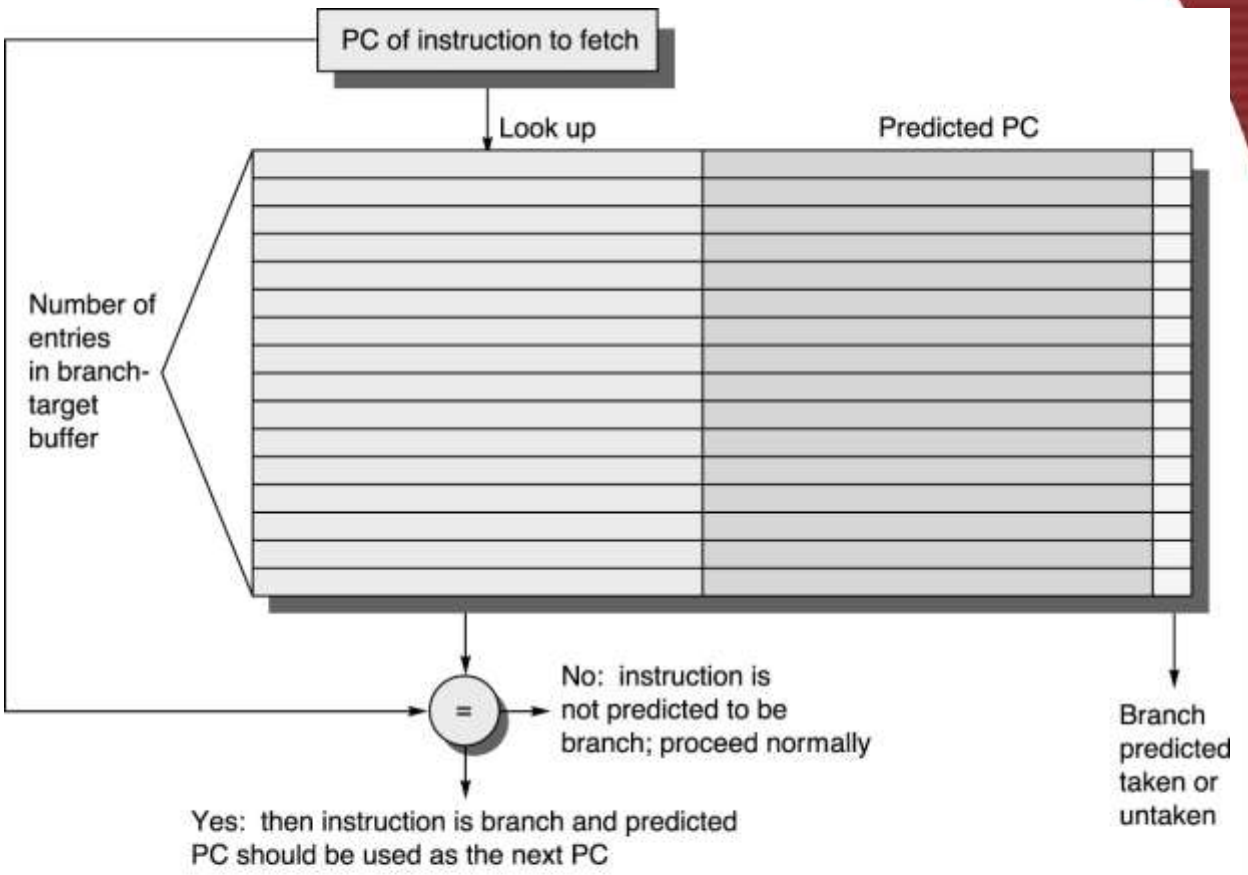
- Previsão errada obriga processador a anular **(flush)** instruções executadas erroneamente
  - Zera os sinais de controle relativos às instruções executadas erroneamente
  - Tem-se que fazer isto para instruções que estão nos estágios IF, ID e EX
    - Branch só é corretamente avaliado no estágio MEM

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

Exemplo de Uso de Previsão Dinâmica

- **Branch Target Buffer (BTB)**
  - Profiling
  - Pentium e PowerPC



Acelerando Avaliação do Desvio

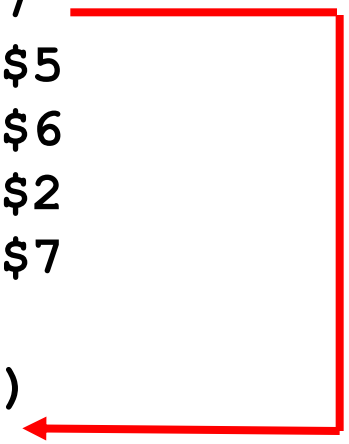
- Uma forma de acelerar a avaliação do desvio é colocar hardware que calcula resultado do desvio no estágio ID

  Somador para endereço destino

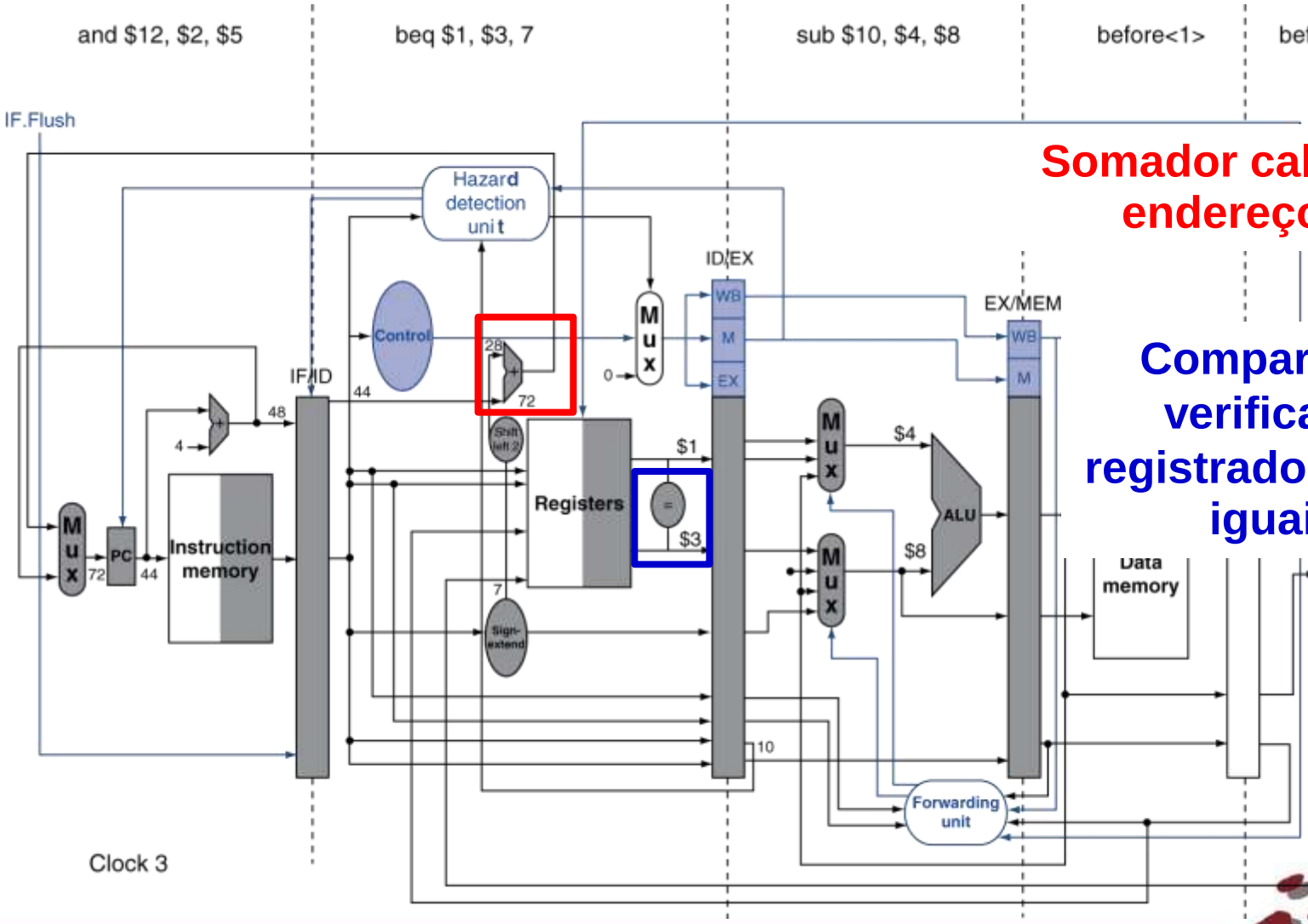
  Comparador de registradores

- **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)
```



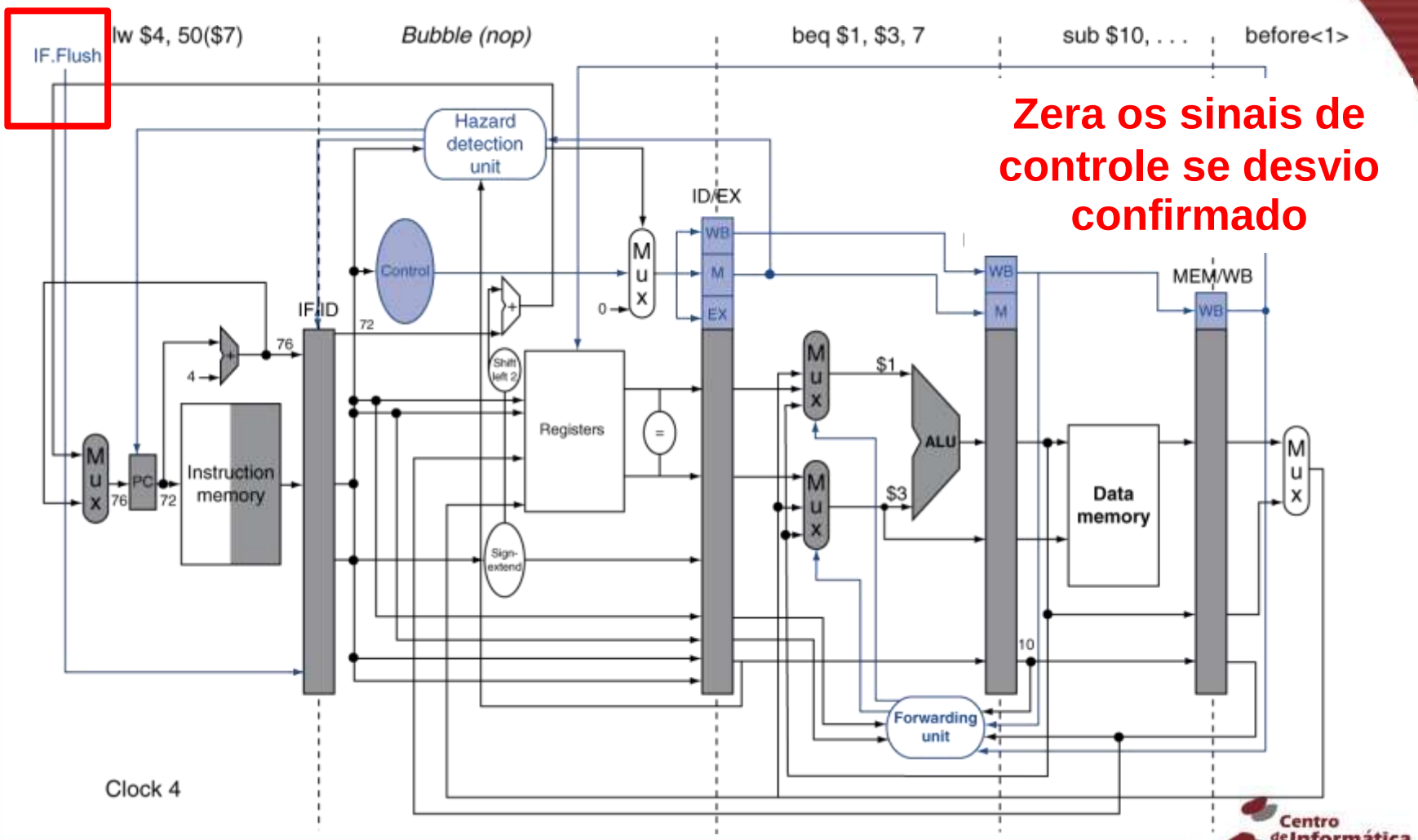
Exemplo: Desvio Confirmado



Somador calcula endereço

Comparador verifica se registradores são iguais

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

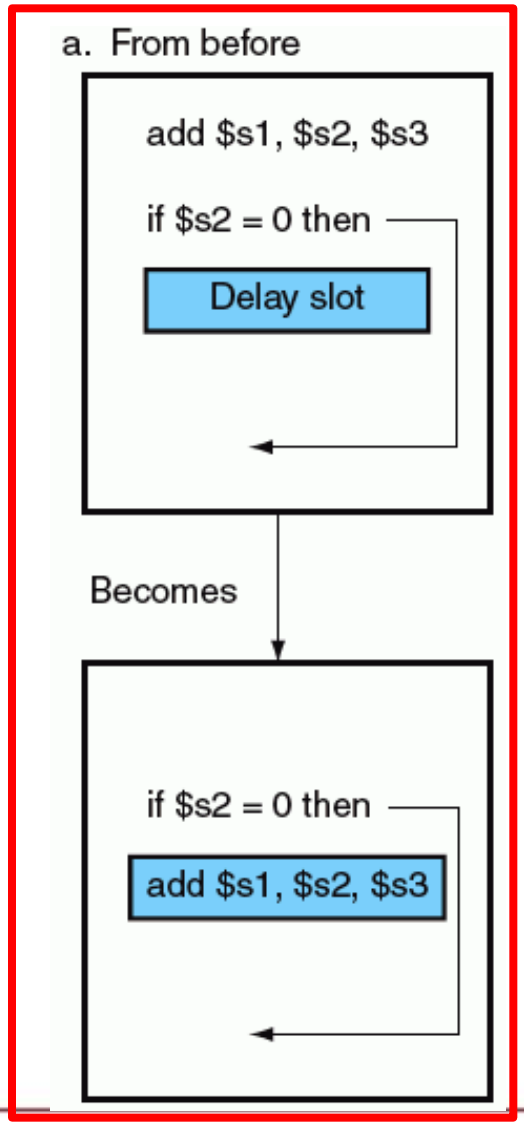
  Assembler re-arruma código

  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**
  - Requer mais hardware
  - instrução no slot é executada independente do resultado do desvio

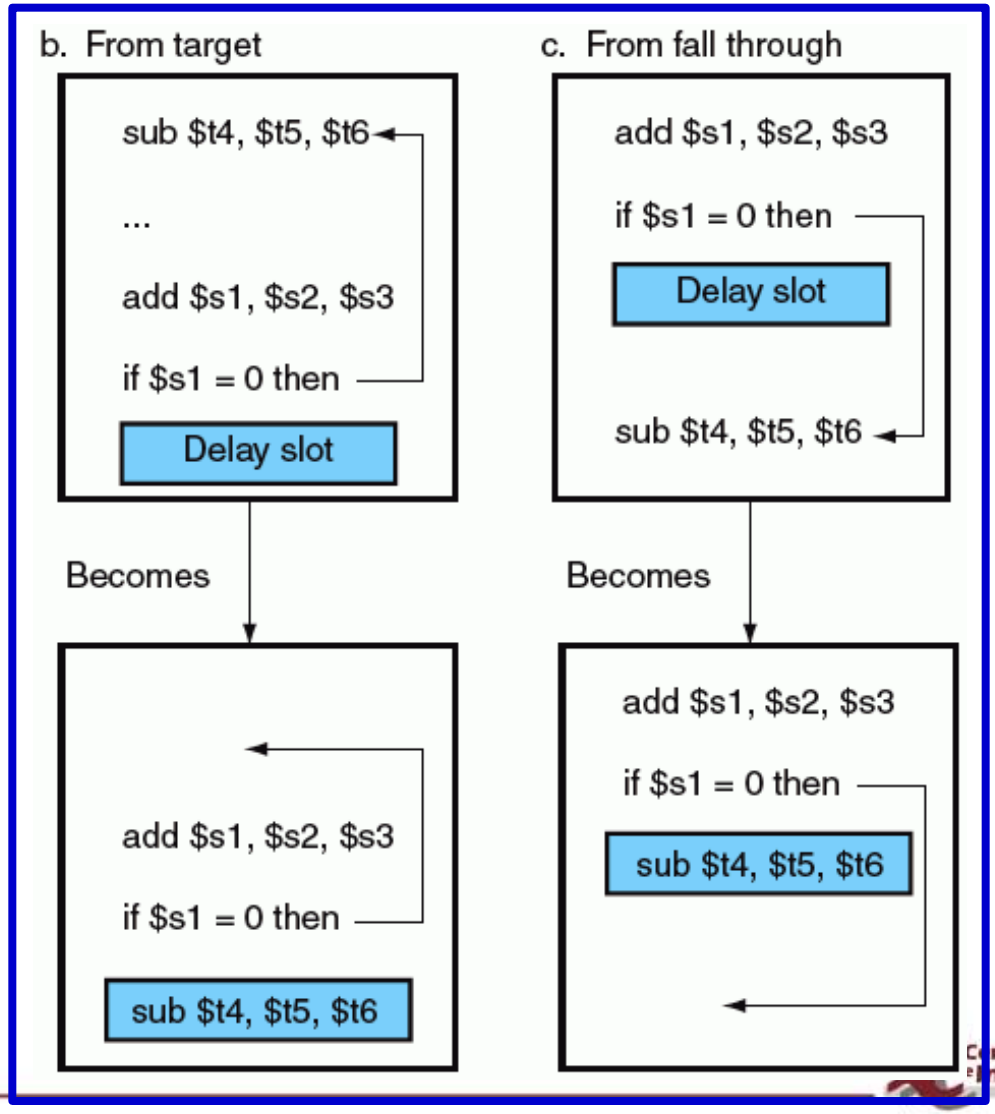
Desvio com Efeito Retardado

Situação ideal: instrução independente



Desvio com Efeito Retardado

Previsão do resultado, se errado só anula instrução fora do delay slot



Análise de Soluções para Conflitos de Controle

- Desvio com efeito retardado (SW)
  - Requer compilador/montador inteligente para identificar instruções independentes
  - Requer análise estática para prever comportamento de desvios
- Congelamento do pipeline (HW)
  - Simples
  - Degrada desempenho

Análise de Soluções para Conflitos de Controle

- Execução especulativa
  - Pode reduzir penalidades de desempenho decorridos de conflitos
  - Estática
    - Simplicidade de implementação
    - Não se baseia no comportamento do código, tendo resultados que variam de aplicação para aplicação
  - Dinâmica
    - Baseia-se no comportamento do código → maior probabilidade de acerto
    - Complexidade de implementação e maior custo de HW
- Aceleração de avaliação de desvios
  - Ajuda a diminuir penalidades de desempenho
  - Custo maior de HW, devido a replicação de componentes