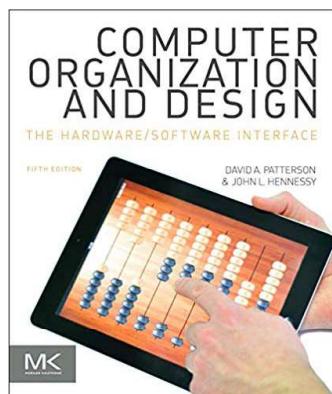


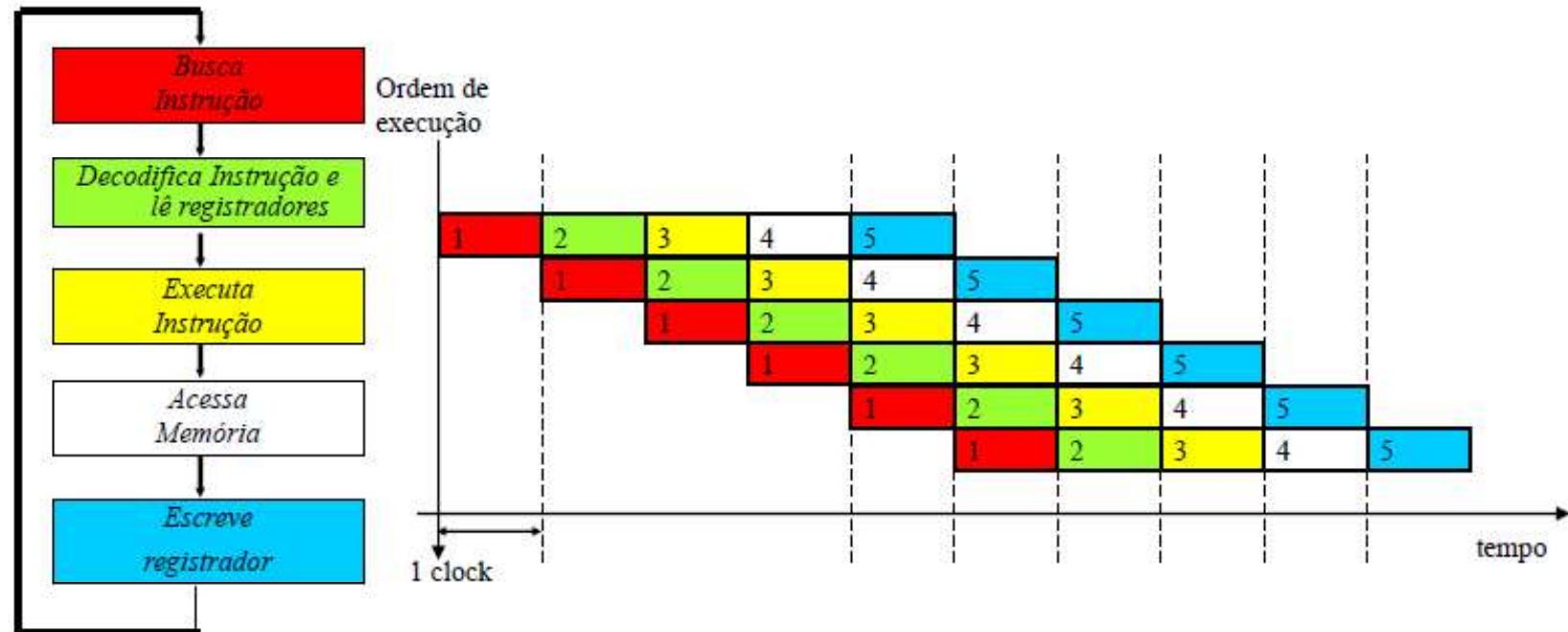
Aula 09 - Confitos de Pipeline (Pipeline Hazards)

Prof. Dr. Clodoaldo A. de Moraes Lima

Material baseado no livro “Patterson, David A., Hennessy, J. L. - Computer Organization And Design: The Hardware/Software Interface”

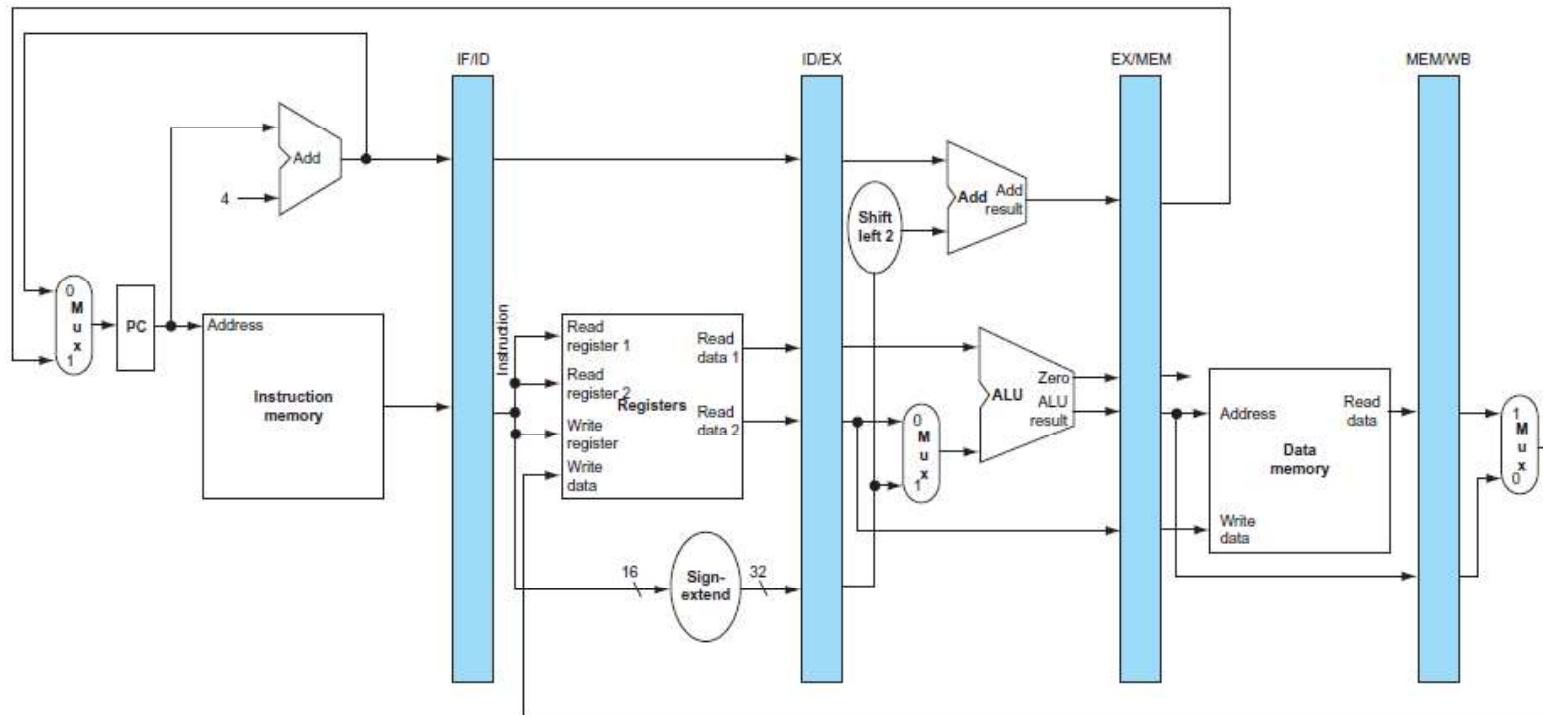


Implementação um pipeline



Praticamente todos os processadores modernos usam pipeline para ganhar desempenho com a execução sobrepostas de partes de instruções.

Implementação um pipeline



Pensando em ciclos de clock

- Resumo dos ciclos para executar instruções numa CPU não pipeline:
 - Branch - 3 clocks
 - Store - 4 clocks
 - Outras - 5 clocks
- EX: Assumindo que as instruções de desvio correspondem a 12% de todas as instruções e que stores a 10%, qual o CPI médio da CPU?

$$\text{CPI médio} = 0.12 * 3 + 0.10 * 0.10 + 0.78 * 5 = 4.66$$

Alguns Problemas à vista...

- ① A memória é acessada 2x durante cada ciclo de clock.
 - Separar memória de dados e de instruções
 - Importante: o período do clock é o mesmo para a CPU com pipeline e sem pipeline... Com isso, a memória deve trabalhar 5x mais rápido.
- ② Registradores acessados 2x por ciclo
 - Para tentar evitar o conflito por recurso, realizar a escrita na 1a metade do ciclo e a leitura na 2a metade do ciclo de clock.
 - Escreve-se primeiro porque o resultado de uma escrita (dado) pode ser usado por outra instrução com execução mais adiantada no pipeline
- ③ Interação entre o PC e o pipeline ao fim da fase IF/BI. Se a decodificação identifica uma instrução de desvio que modifica o PC, como isso afeta o pipeline?
 - O uso de registradores específicos para o pipeline fornecem à CPU uma memória temporária que permite armazenar estados intermediários da execução das instruções no pipeline.

Conflitos de Pipeline (Pipeline “Hazards”)

- O ganho de desempenho ocorre porque é possível se iniciar a execução de uma nova instrução a cada ciclo de clock. Porém, em um cenário realista, isso nem sempre é possível...
- Os chamados conflitos de pipeline (“pipeline hazards”) atrasam a execução da próxima instrução por alguns ciclos de clock, até que haja tempo para que sejam solucionados...
 - Usaremos hazard = conflito (hazard: azar, risco, acaso, ...)
- Três tipos:
 - 1 Estruturais
 - 2 De Controle
 - 3 De Dados

Conflitos Estruturais (Structural Hazards)

O hardware (do datapath) não tem recursos suficientes para atender às solicitações da execução sobreposta das instruções no pipeline em um dado ciclo de clock

O conjunto de instruções do MIPS foi projetado para executar em pipeline (é muito fácil evitar esses conflitos no design)

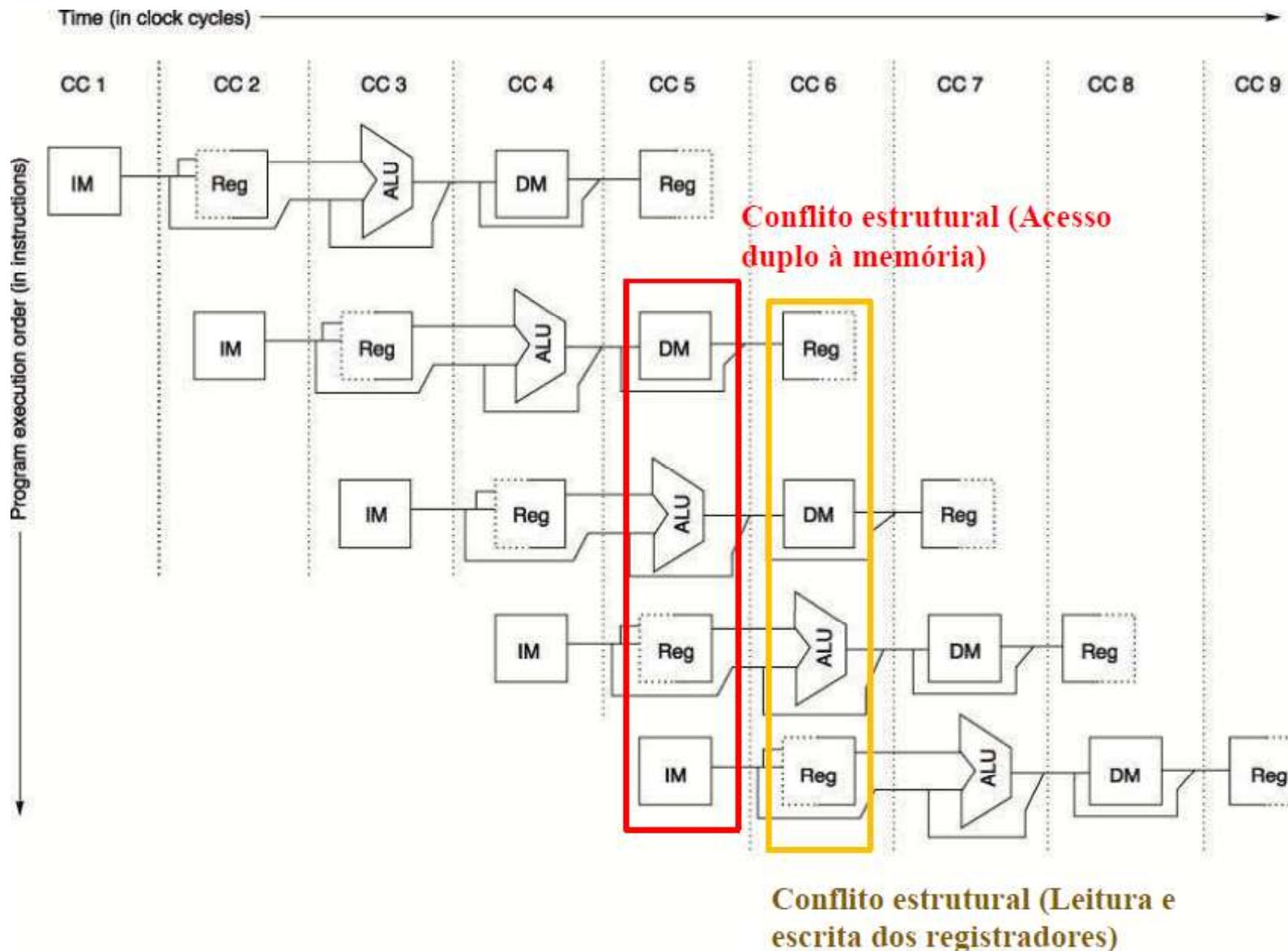
Uma possível solução é a parada do pipeline (“**bolha**”), isto é **interromper a progressão das instruções** pelo pipeline.

Conflitos Estruturais (Structural Hazards)

Se na CPU multiciclo vista anteriormente fosse mantida apenas uma memória única para dados e instruções, ocorreria um conflito estrutural quando uma instrução estiver sendo buscada ao mesmo tempo que um dado é escrito na memória (estágios BI e MEM)

Outro problema aconteceria se considerarmos que o banco de registradores só puder ser ou escrito ou lido num mesmo ciclo (estágios DI e ER)

Conflitos Estruturais (Structural Hazards)



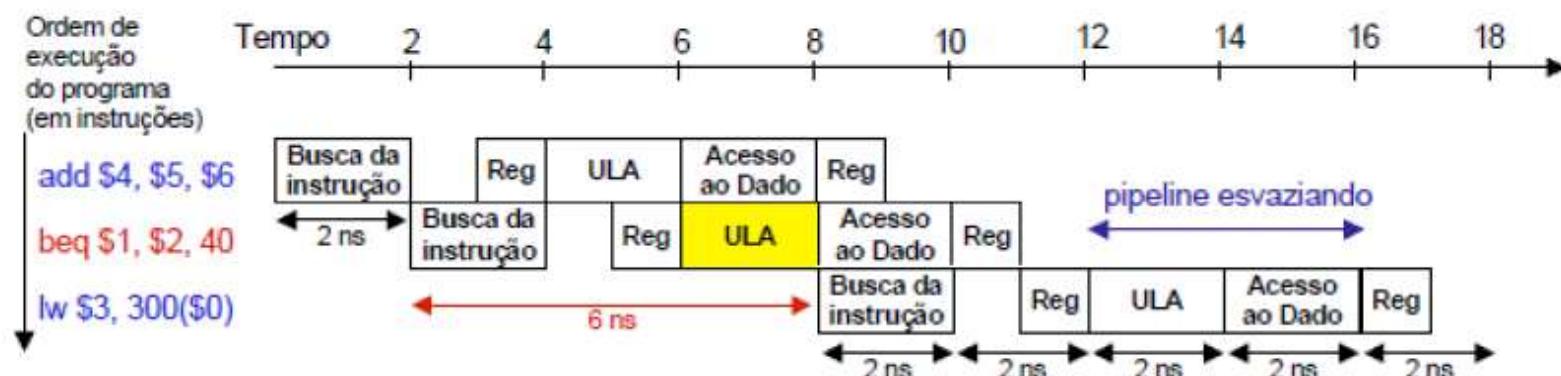
Conflitos de Controle (Control Hazards)

Interação PC – pipeline: conflitos se originam da necessidade de se tomar uma decisão para o valor de PC baseada nos resultados de uma instrução, a qual ainda não foi concluída.

Muito comuns em instruções de salto condicional (beq, no caso do MIPs reduzido)

Parada do pipeline num conflito de controle

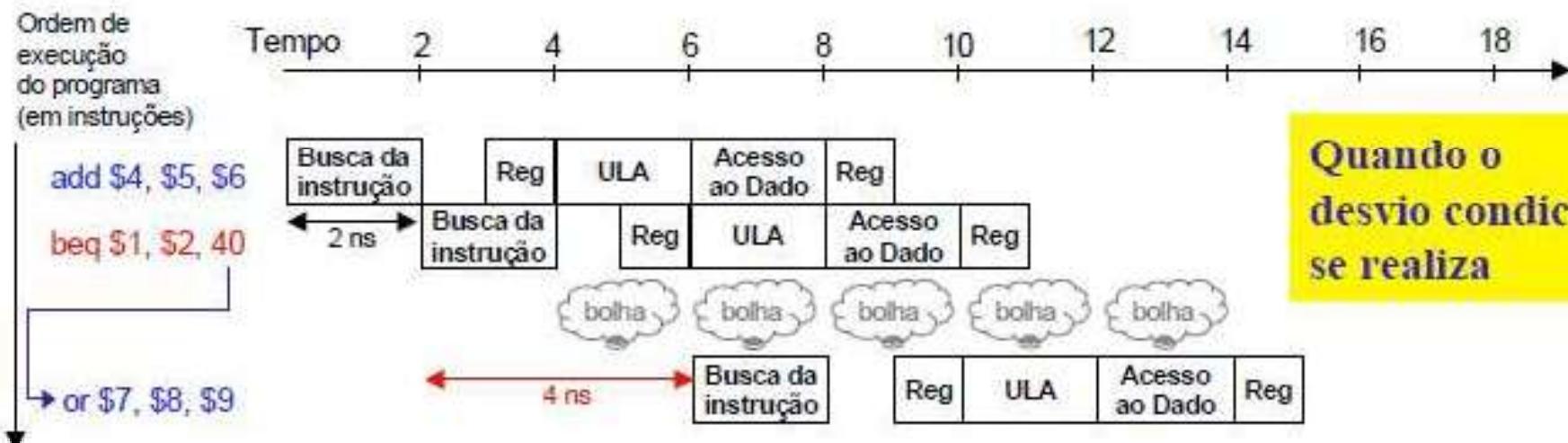
Assumir que um hardware extra foi adicionado para permitir testar registradores, calcular o endereço de desvio condicional e atualizar o PC, tudo isso no segundo estágio (decodificação). Isso é uma solução comum para minimizar o efeito desse tipo de conflito.



Outra solução para conflito de controle

- Predição estática: esquema muito usado em CPUs reais
- Simples: assumir que os desvios condicionais sempre irão falhar (teste gera FALSO)
 - Se acertar, o pipeline prossegue com velocidade máxima
 - Se errar, será necessário atrasar o pipeline
- Mais sofisticado: assumir que parte dos desvios ocorrem e outros não ocorrem
 - Desvios para endereços anteriores supostamente são feitos (última instrução de um loop)
 - Desvios para endereços posteriores supostamente falham (instruções tipo if-then-else)
 - Considero que as primeiras instruções de um programa estão nos endereços mais baixos, isto é, que o programa vai dos endereços mais baixos para os mais altos.

Pipeline com predição de desvio



Mais uma solução para conflito de controle

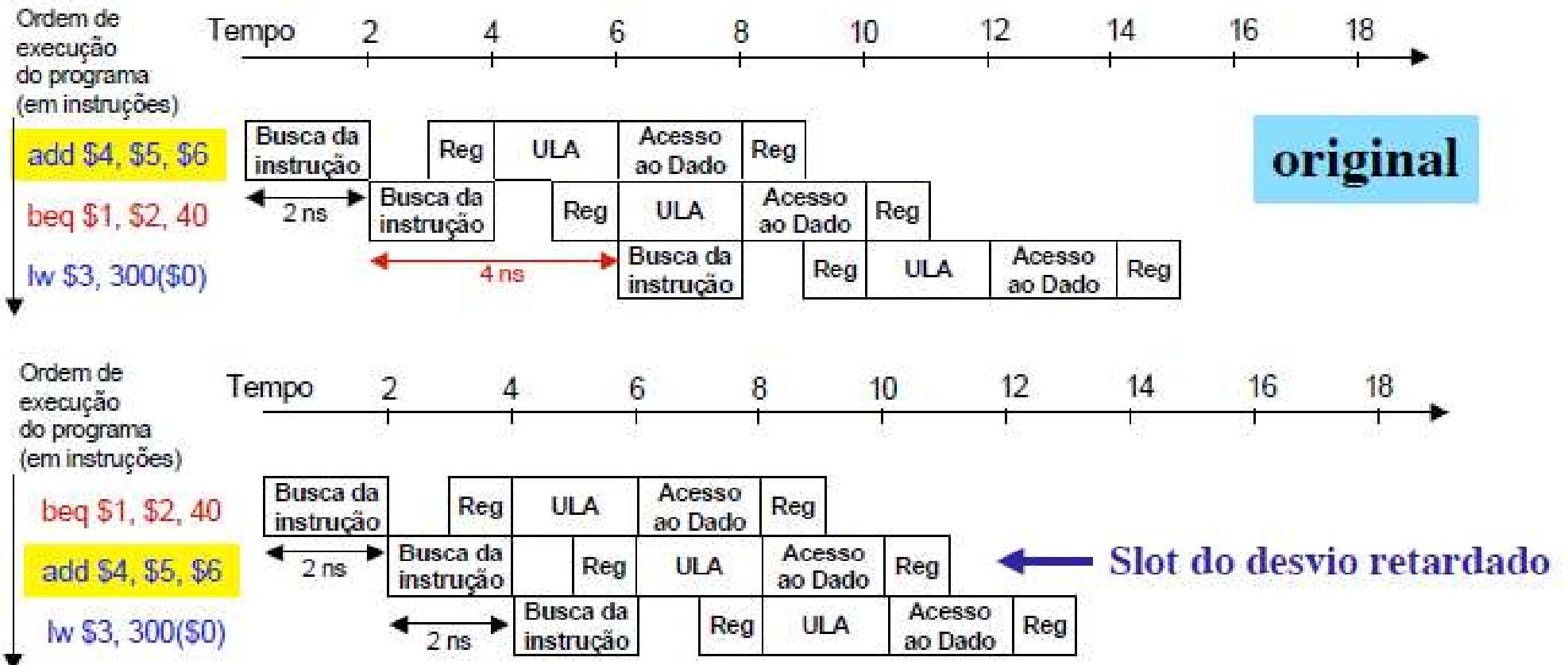
- Predição dinâmica: realizada em função do comportamento anterior (histórico) de cada desvio
- Podem mudar a predição para um mesmo ponto de desvio com o decorrer da execução do programa
- Um esquema comum é manter uma história para cada desvio realizado ou não-realizado
- Preditores dinâmicos são implementados em hardware e apresentam até 90% de acerto

Qualquer que seja o esquema de predição, quando o palpite (previsão) estiver errado(a), o controle do pipeline precisa assegurar que as instruções seguintes ao desvio não terão influência na execução das instruções futuras.

Mais uma solução para conflito de controle

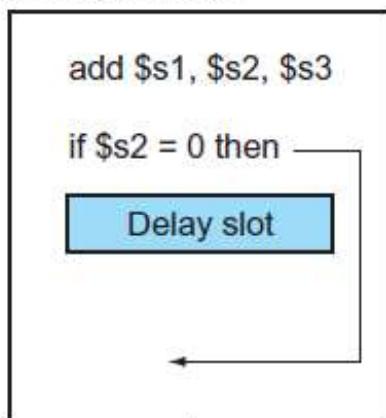
- O esquema de retardar a decisão (delayed branch) consiste em encaixar uma instrução “útil” para ser executada logo após a instrução de desvio, de modo a manter o pipeline preenchido, evitando a parada
- Isto é um trabalho para o Compilador!
 - Lembrem-se que o compilador tem papel fundamental para o desempenho de máquinas RISC

Solução para conflito de controle: delayed branch

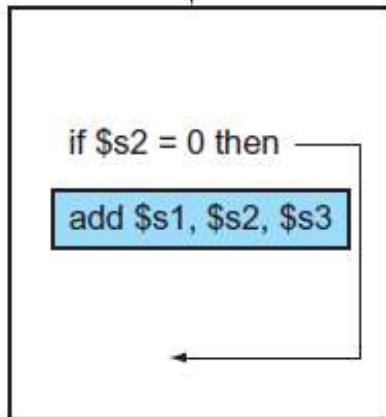


Soluções similares ao delayed branch

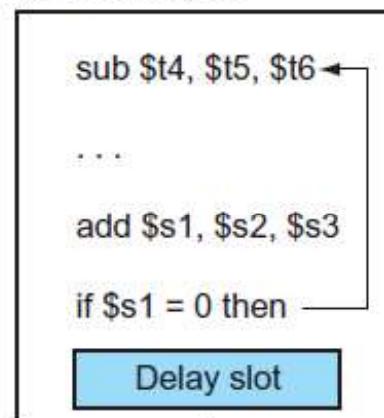
a. From before



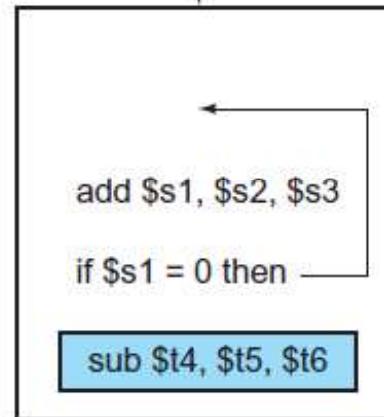
Becomes



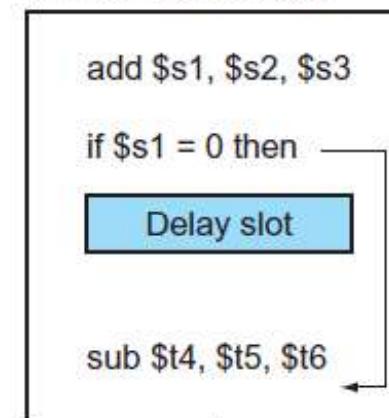
b. From target



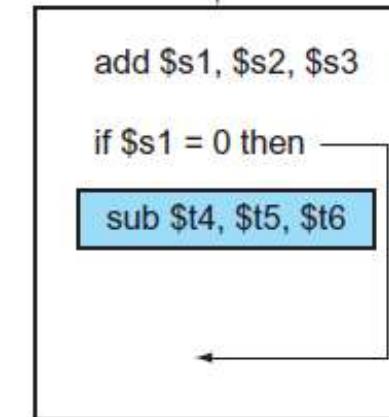
Becomes



c. From fall-through

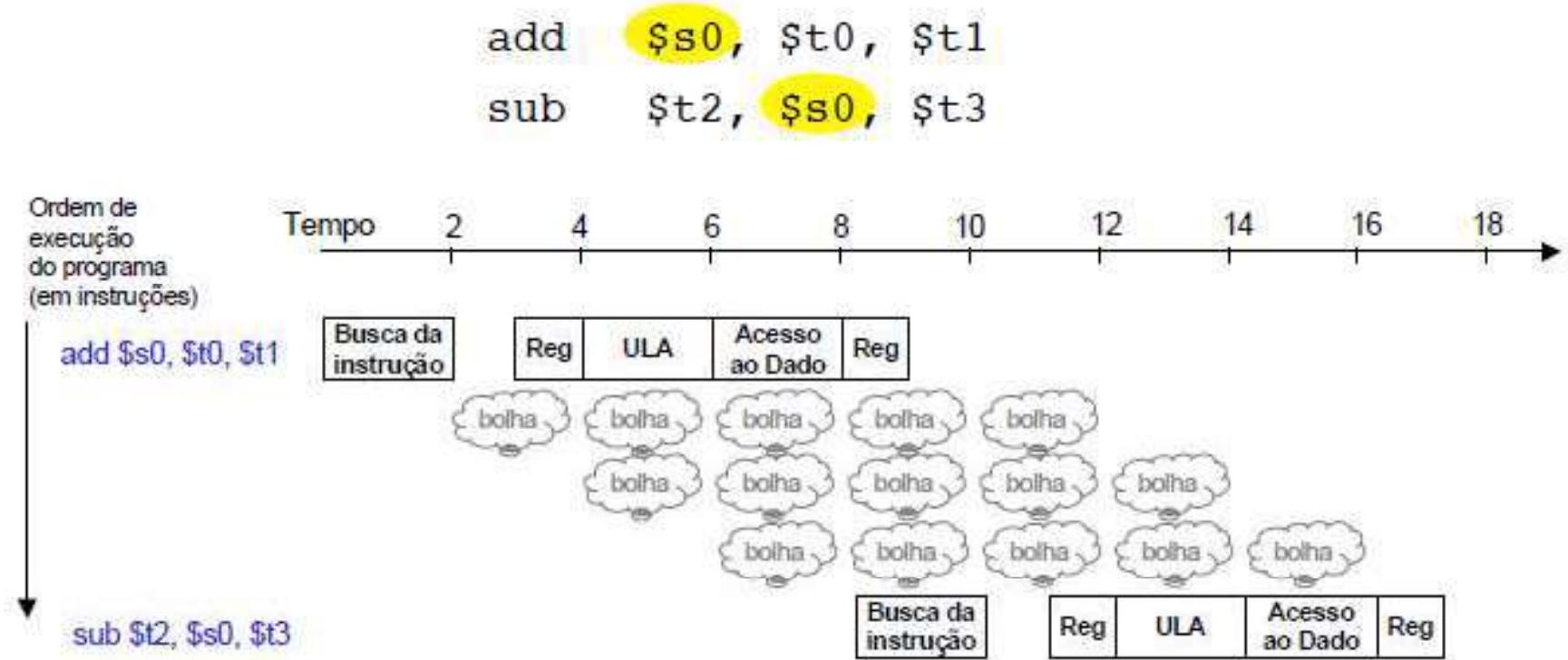


Becomes



Conflitos de Dados (Data Hazards)

Ocorre quando uma instrução depende de dados resultantes da execução de outra que ainda está no pipeline



Uma solução para conflito de dados

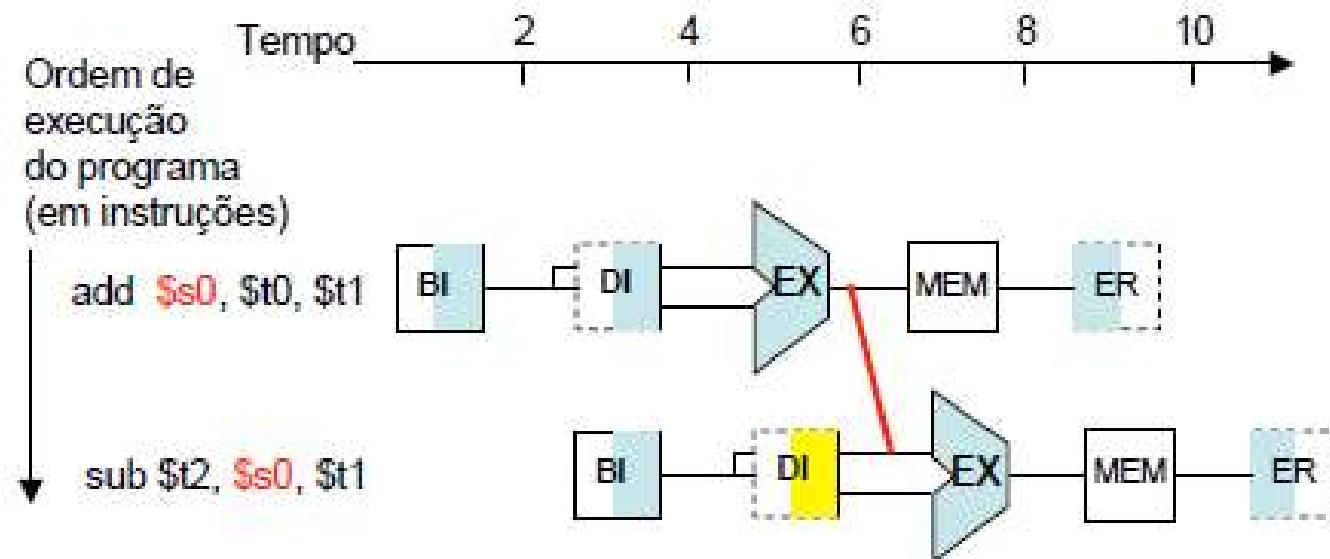
- Passar para o compilador a solução deste tipo de conflito é inviável, pois ele é muito frequente
- Adiantamento (forwarding ou bypass): tão logo a ULA calcule o resultado da operação, ele pode ser disponibilizado como operando para as instruções que vem logo a seguir.
 - Necessidade de um hardware de roteamento para detectar a dependência entre dados de instruções subsequentes.
- Supondo a sequência de execução anterior:

```
add    $s0, $t0, $t1
sub    $t2, $s0, $t3
```

Solução do conflito por adiantamento

Solução:

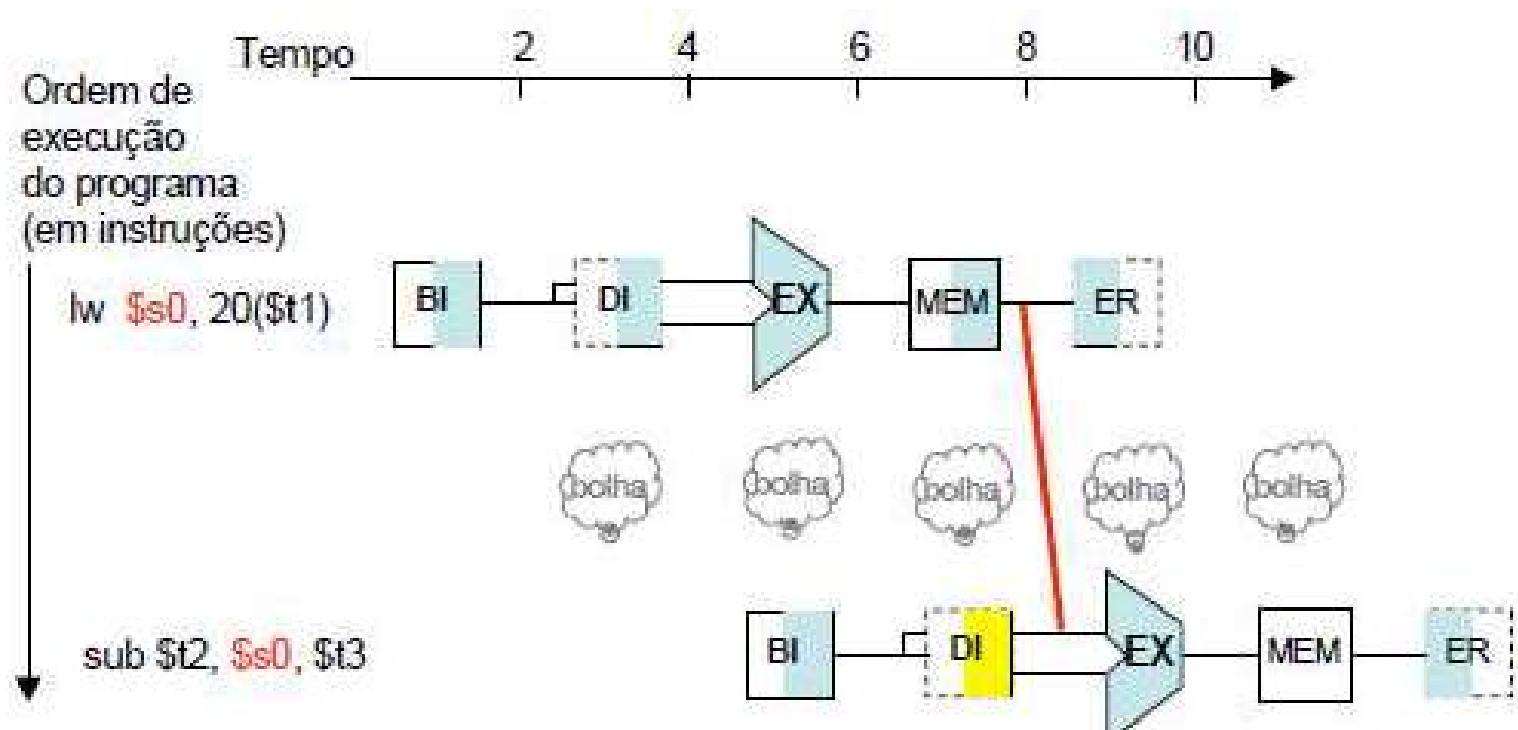
conexão entre a saída da ULA e a entrada da própria ULA



Não haverá mais bolha no pipeline!

Outro exemplo de adiantamento

O adiantamento para uma instrução `lw` seguida de uma instrução que usa o valor que acaba de ser buscado na memória:



Sempre haverá uma bolha no pipeline nesses casos! Inevitável!

Solução por reordenação de código

- O compilador reordena as instruções de forma a evitar os possíveis conflitos de dados.
- Ex: o código do procedimento swap abaixo exibe um conflito.

```
# reg $t1 possui o endereço de v[k]
lw $t0, 0($t1) # reg $t0 (temp) = v[k]
lw $t2, 4($t1) # reg $t2 = v[k+1]
sw $t2, 0($t1) # v[k] = reg $t2
sw $t0, 4($t1) # v[k+1] = reg $t0 (temp)
```

Solução por reordenação de código

Solução

Trocar a 2a e a 3a linhas de posição: código sem conflito

```
# reg $t1 possui o endereço de v[k]
lw $t0, 0($t1) # reg $t0 (temp) = v[k]
lw $t2, 4($t1) # reg $t2 = v[k+1]
sw $t0, 4($t1) # v[k+1] = reg $t0 (temp)
sw $t2, 0($t1) # v[k] = reg $t2
```

Conflitos provocam a parada do pipeline

Com os conflitos, algumas expressões envolvendo ganhos mais realistas de pipeline em termos de CPI poderiam ser calculados da seguinte forma:

$$\text{Speedup} = \frac{\text{CPI unpipelined}}{\text{CPI pipelined}}$$

$$\text{Speedup} = \frac{\text{No estágios}}{(1 + \text{paradas por instrução})}$$

$$\text{Speedup} = \frac{\text{Tmédio por Inst unpipelined}}{\text{Tmédio por Inst pipelined}}$$

OBS: Assume-se que o período de clock é o mesmo para as implementação com ou sem pipeline.

Conflitos provocam a parada do pipeline

Pode-se pensar no desempenho em termos de um ciclo de clock mais rápido para a CPU pipelined da seguinte forma:

$$\text{Speedup} = \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} \times \frac{\text{Tempo Ciclo Clock unpipelined}}{\text{Tempo Ciclo Clock pipelined}}$$

$$\text{Tempo Ciclo Clock pipelined} = \frac{\text{Tempo Ciclo Clock unpipelined}}{\text{No estágios}}$$

$$\text{Speedup} = \frac{1}{1 + \text{Paradas do pipeline por Instrução}} \times \text{No estágios}$$

Speedup × conflitos

- A solução básica para conflitos é a inserção de uma parada ou bolha no pipeline. O efeito é o retardamento da execução da próxima instrução de 1 ciclo a cada bolha inserida.
 - Obviamente, isso aumenta o CPI, piorando o desempenho!

Ex: Considere a comparação entre 2 CPUs. Uma delas tem conflito estrutural que ocorre 40bolha, mas com isso, trabalha com uma taxa de clock 1.05x mais alta que a CPU sem conflito. Quanto a CPU com conflito é mais rápida do que a sem conflito?

Speedup × conflitos

$$\text{Speedup} = \frac{\text{CPI sem conflito}}{\text{CPI com conflito}} \times \frac{\text{Tempo Ciclo Clock sem conflito}}{\text{Tempo Ciclo Clock com conflito}}$$

$$\text{Speedup} = \frac{1}{1 + 0.4 * 1} \times \frac{1}{1/1.05}$$

$$\text{Speedup} = 0.75$$

Note que:

- ① Apesar da taxa de clock do processador com conflito ser um pouco maior, o speedup é um pouco menor que 1 (perde desempenho, mas não muito...) e
- ② Não cuidar do conflito poderia ser uma solução adotada por conta de custos envolvendo as alterações na via de dados, aliada à baixa probabilidade de ocorrência deste conflito

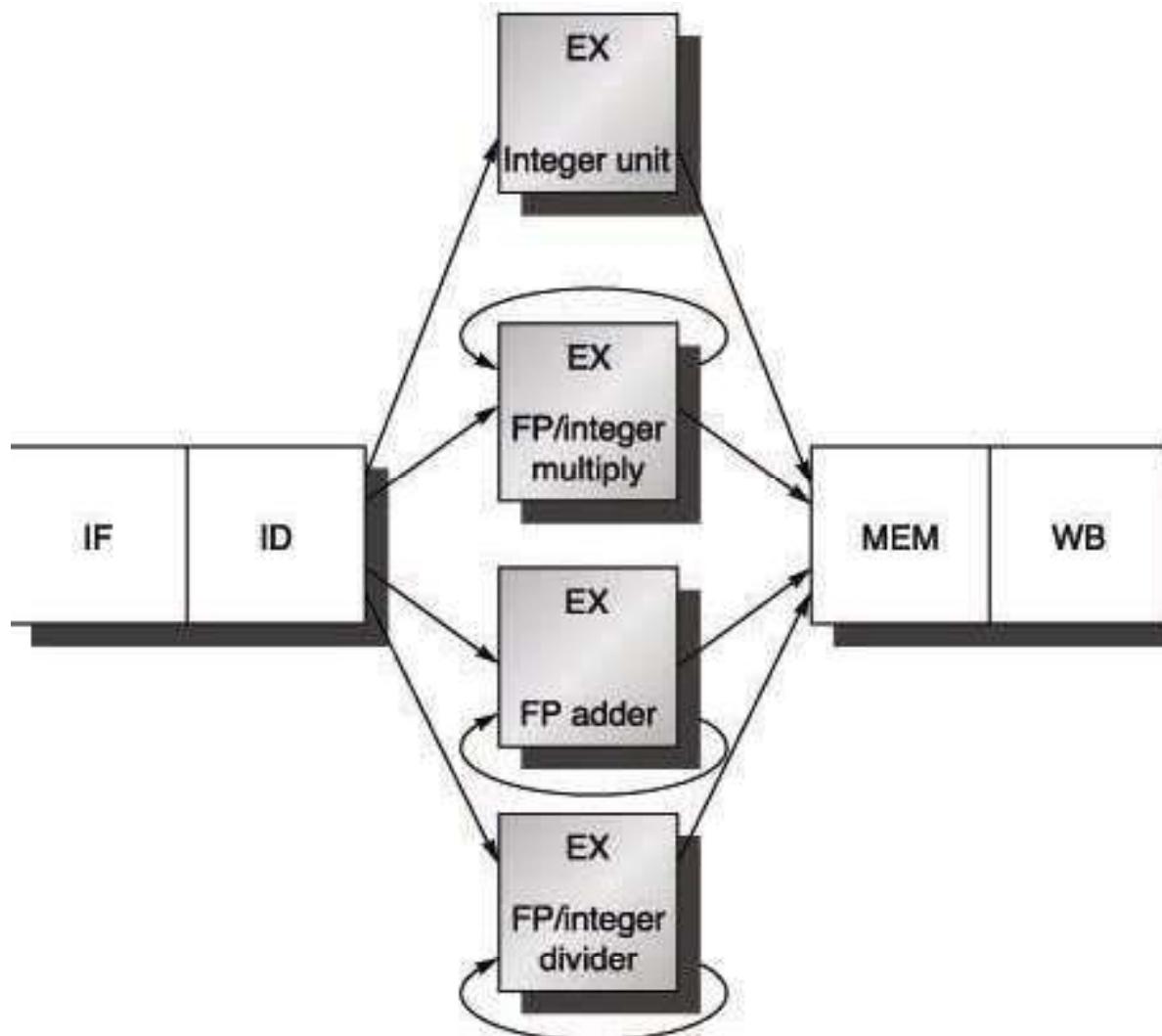
Pipeline não resolve sozinho o desempenho

- Precisaremos de buffers entre estágios do pipeline
 - Armazenar os estados intermediários da execução de cada um das instruções que está momentaneamente no pipeline
 - Detecção de dependências
- Além dos conflitos, ainda existem outros problemas a serem resolvidos...
 - Instruções com tempos diferentes por estágio (adição x multiplicação, ponto flutuante x inteiros)
 - Múltiplos pipelines em arquiteturas superescalares
 - Paralelismo em nível de instrução x compiladores paralelizantes
 - Predição de desvio por hardware

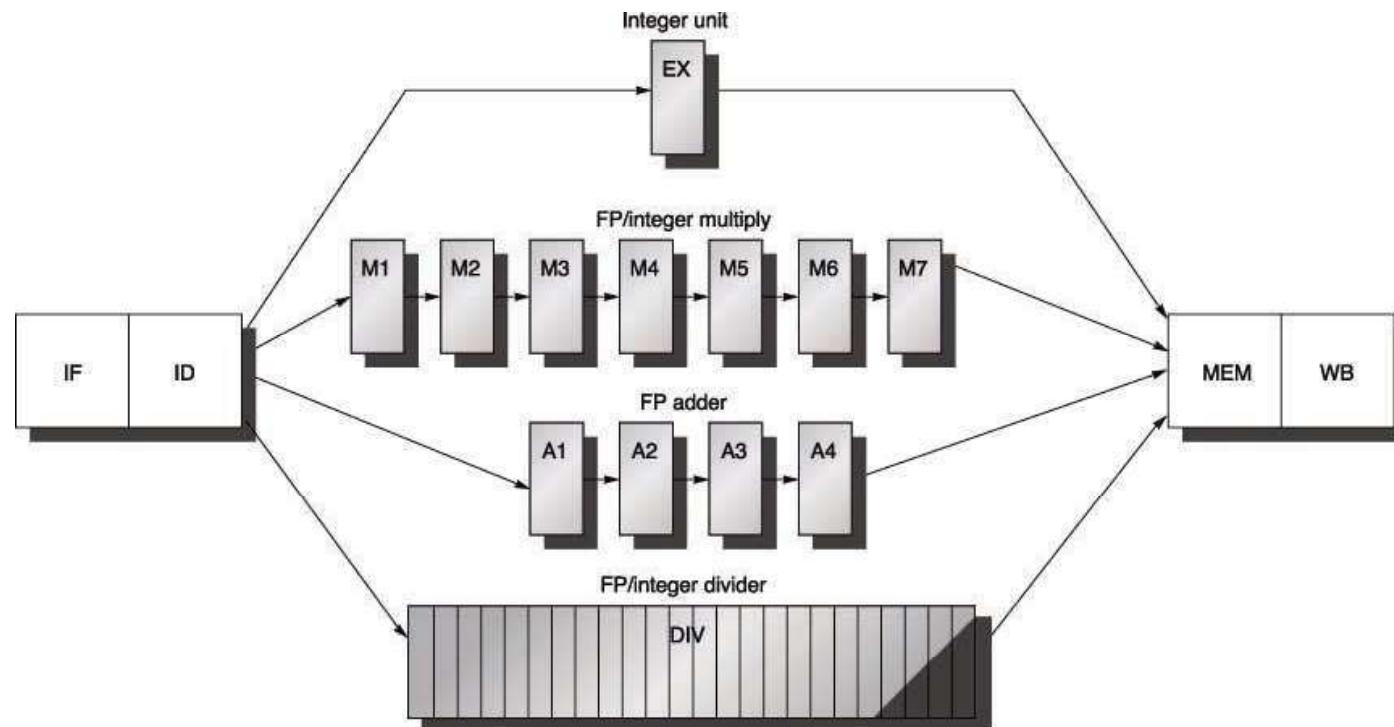
Operações com múltiplos clocks

- Algumas aplicações não serão completadas em apenas 1 clock.
Exemplos:
 - Multiplicação Floating Point
 - Divisão Floating Point
 - Soma Floating Point
- Pode se supor que existe um hardware específico para executar esse tipo de instrução.
- Em geral, Multiplicação e soma FP são totalmente pipelined, mas a divisão é não pipelined.

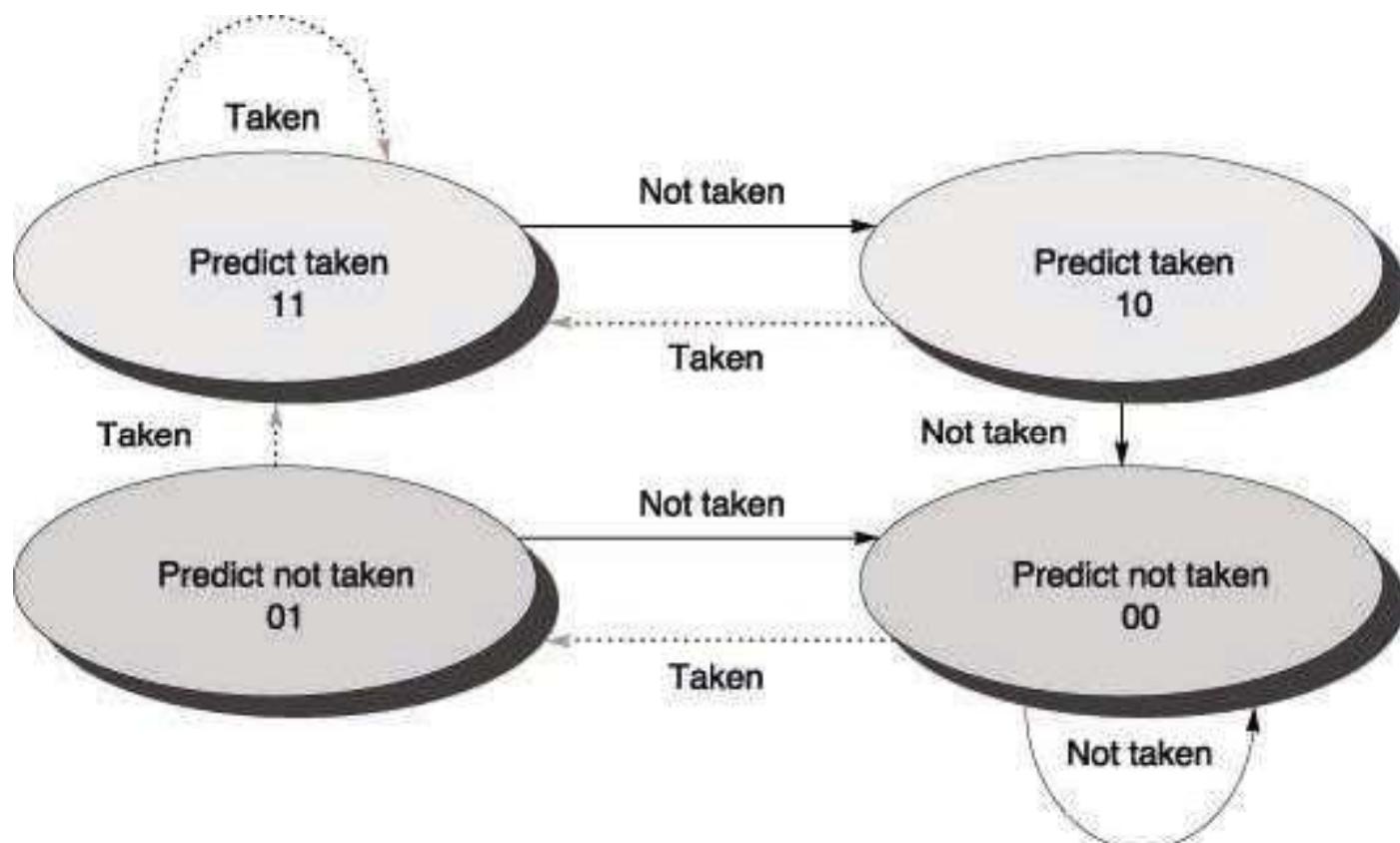
Múltiplos estágios de execução na via de dados



Estágios de execução (EX) pipelined



Previsão de desvio via hardware



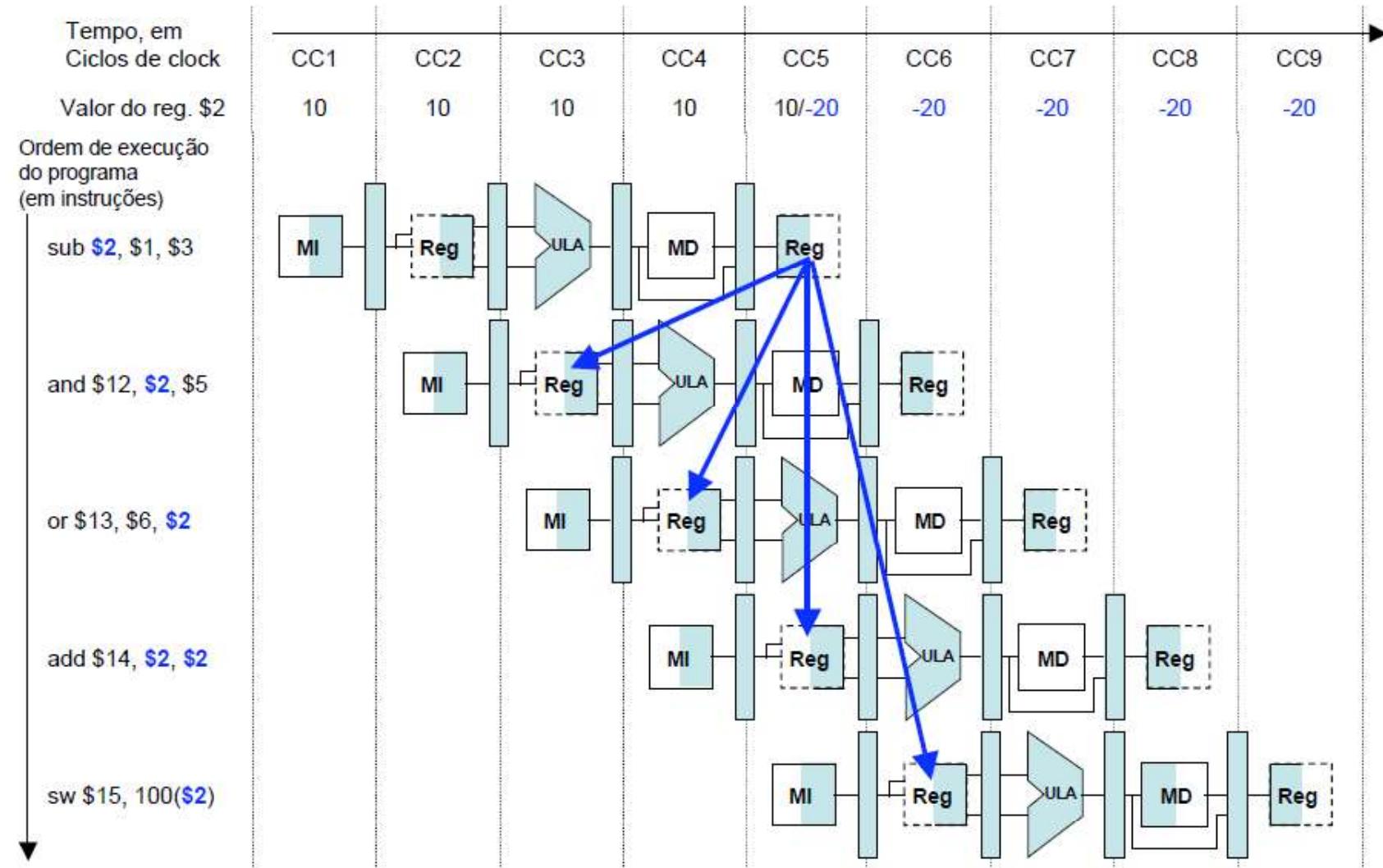
Conflitos por Dados

Seja o seguinte trecho de código, escrito para o MIPS

```
sub $2, $1, $3      # registrador $2 é escrito pela instrução sub  
and $12, $2, $5    # primeiro operando ($2) depende de sub  
or   $13, $6, $2    # segundo operando ($2) depende de sub  
add  $14, $2, $2    # primeiro e segundo operandos ($2) dependem de sub  
sw   $15, 100($2)  # base ($2) depende de sub
```

Para estudar as consequências destas dependências quando da execução em pipeline, usar um diagrama de pipeline de múltiplos ciclos.

Conflitos por Dados



Conflitos por Dados

Uma solução seria o compilador evitar sequências de instruções que gerassem conflitos por dados

```
sub $2, $1, $3      # registrador $2 é escrito pela instrução sub
nop                # na falta de instruções que sejam independentes, o
nop                # compilador inseriria instruções “nop”
and $12, $2, $5    # primeiro operando ($2) depende de sub
or   $13, $6, $2    # segundo operando ($2) depende de sub
add  $14, $2, $2    # primeiro e segundo operandos ($2) dependem de sub
sw   $15, 100($2)  # base ($2) depende de sub
```

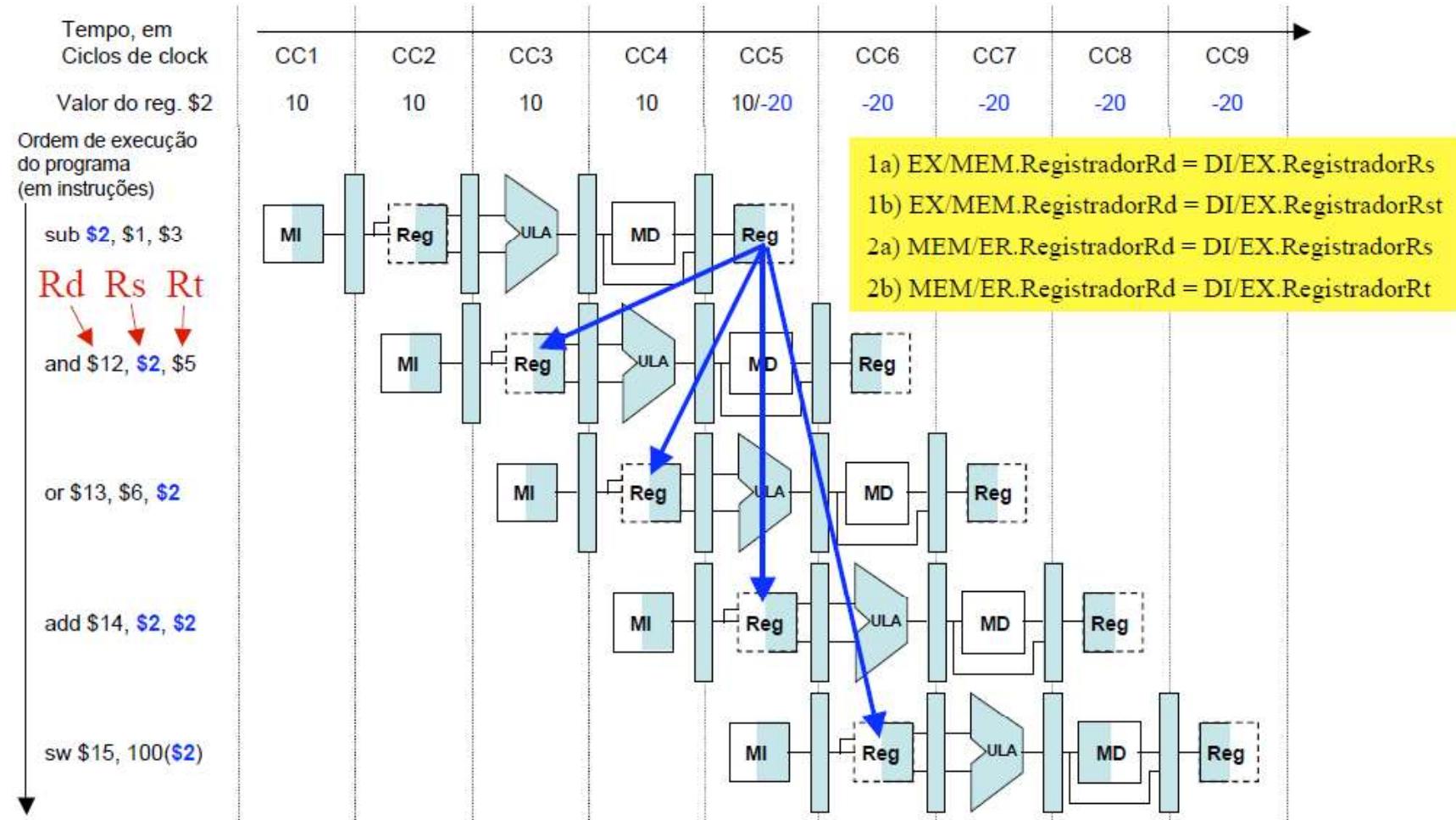
Problemas

- Conflitos por dados são muito frequentes
- A inserção de instruções nop causa perda de desempenho

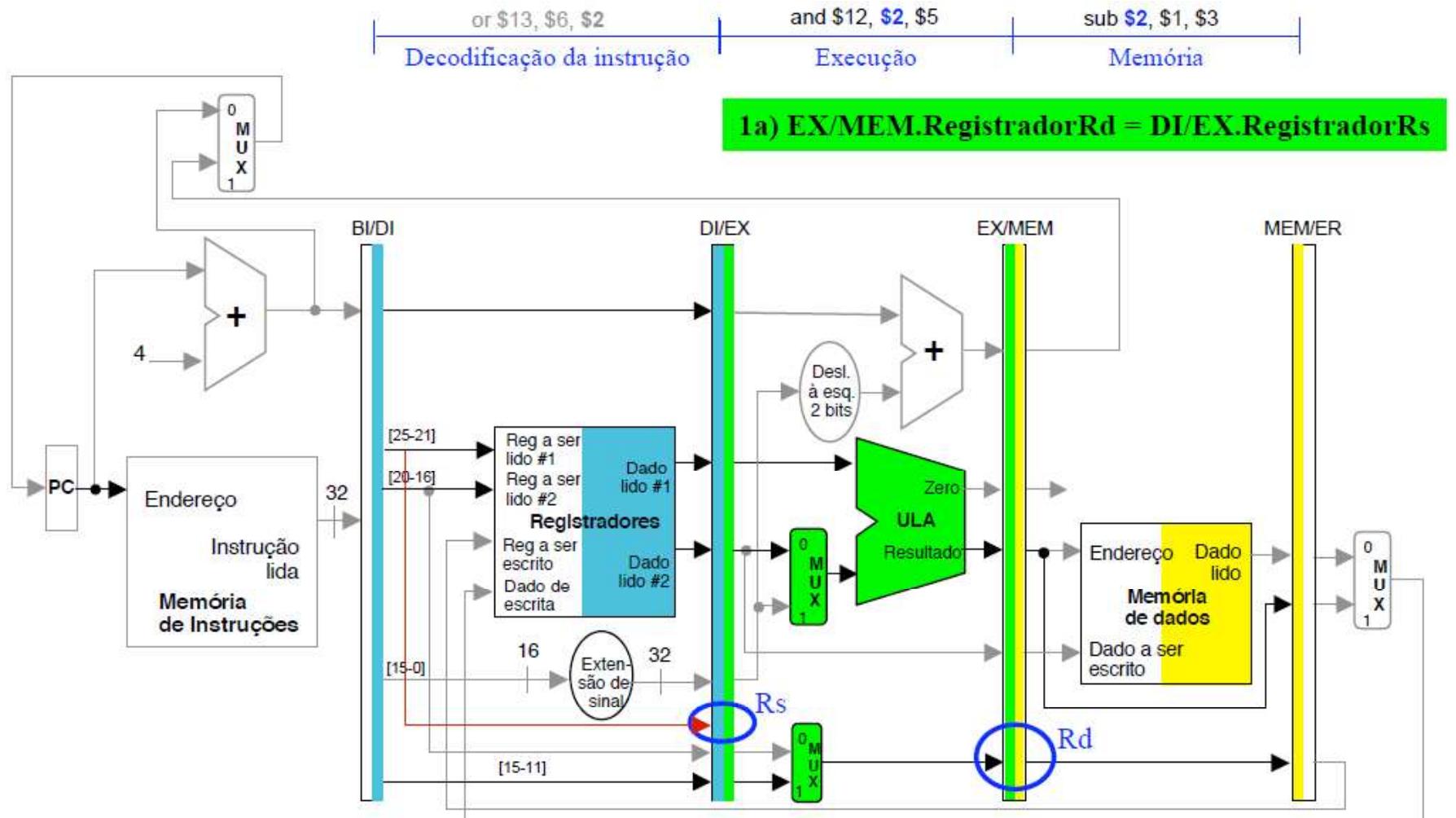
Outra solução

- Detectar o conflito
 - Adiantar o resultado da ULA (ou da memória de dados)
-
- Testes para detecção de conflito (uso dos registradores de pipeline):
 - 1a) EX/MEM.RegistradorRd = DI/EX.RegistradorRs
 - 1b) EX/MEM.RegistradorRd = DI/EX.RegistradorRt
 - 2a) MEM/ER.RegistradorRd = DI/EX.RegistradorRs
 - 2b) MEM/ER.RegistradorRd = DI/EX.RegistradorRt

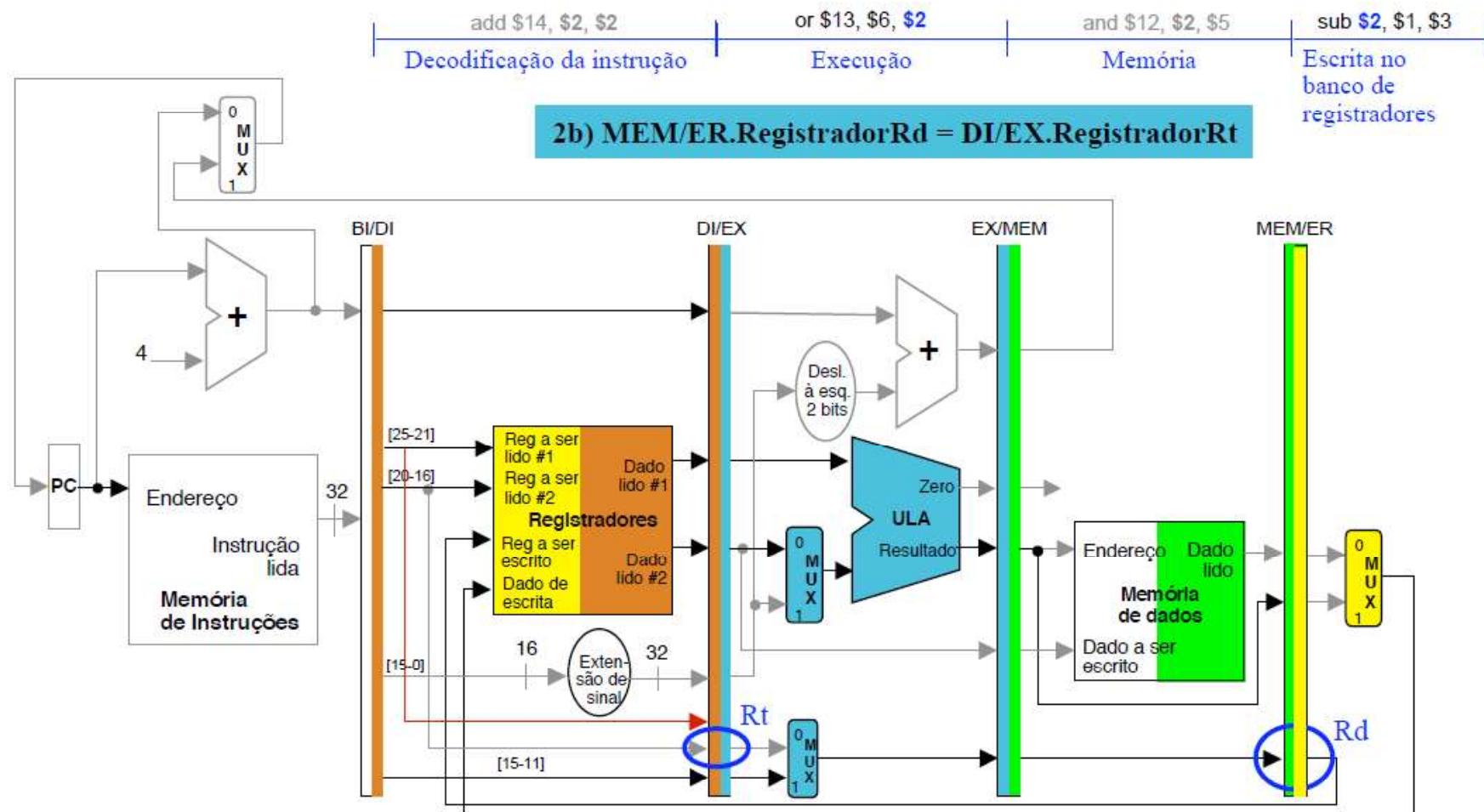
Conflitos por Dados



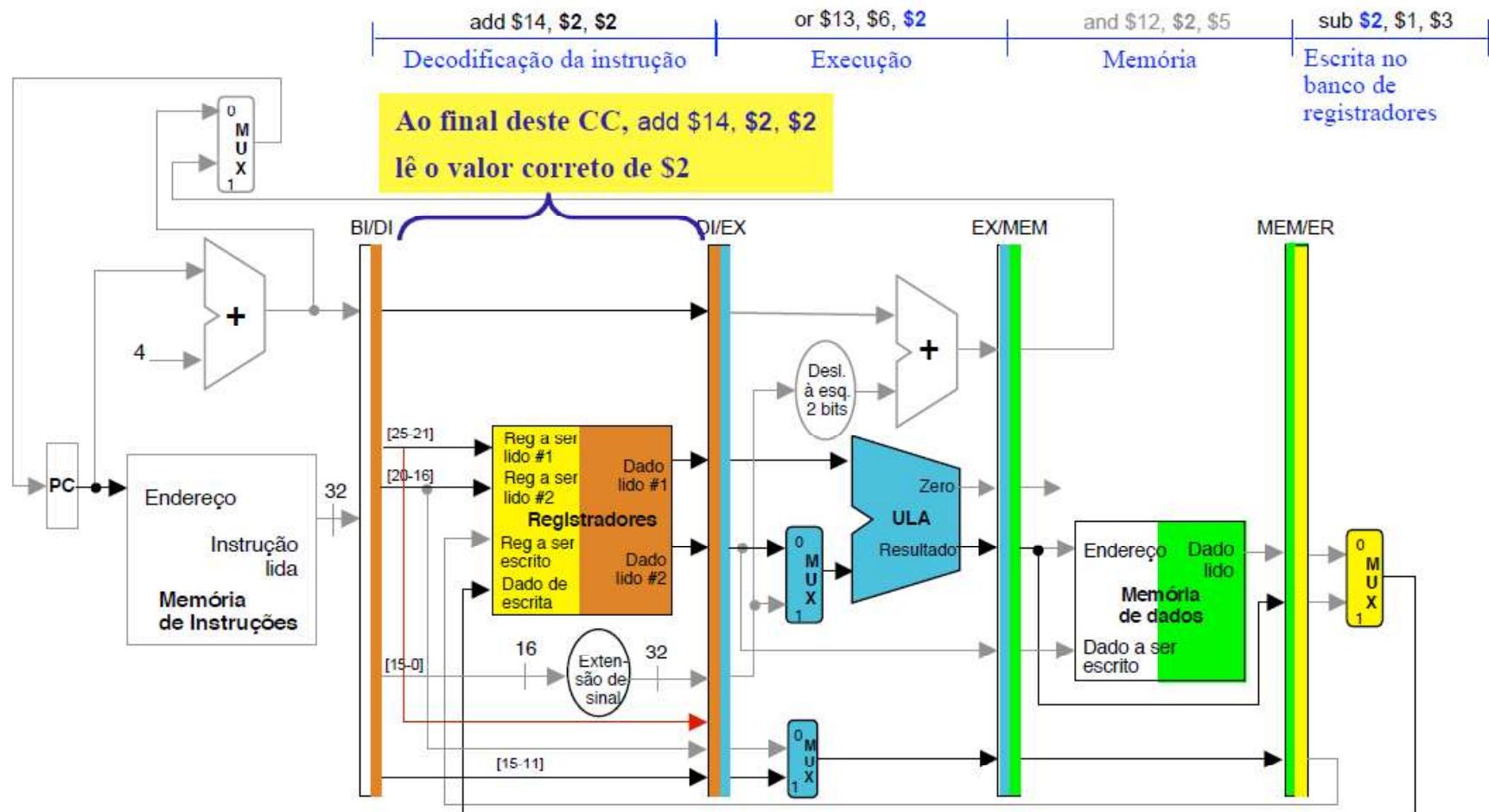
Conflitos por Dados



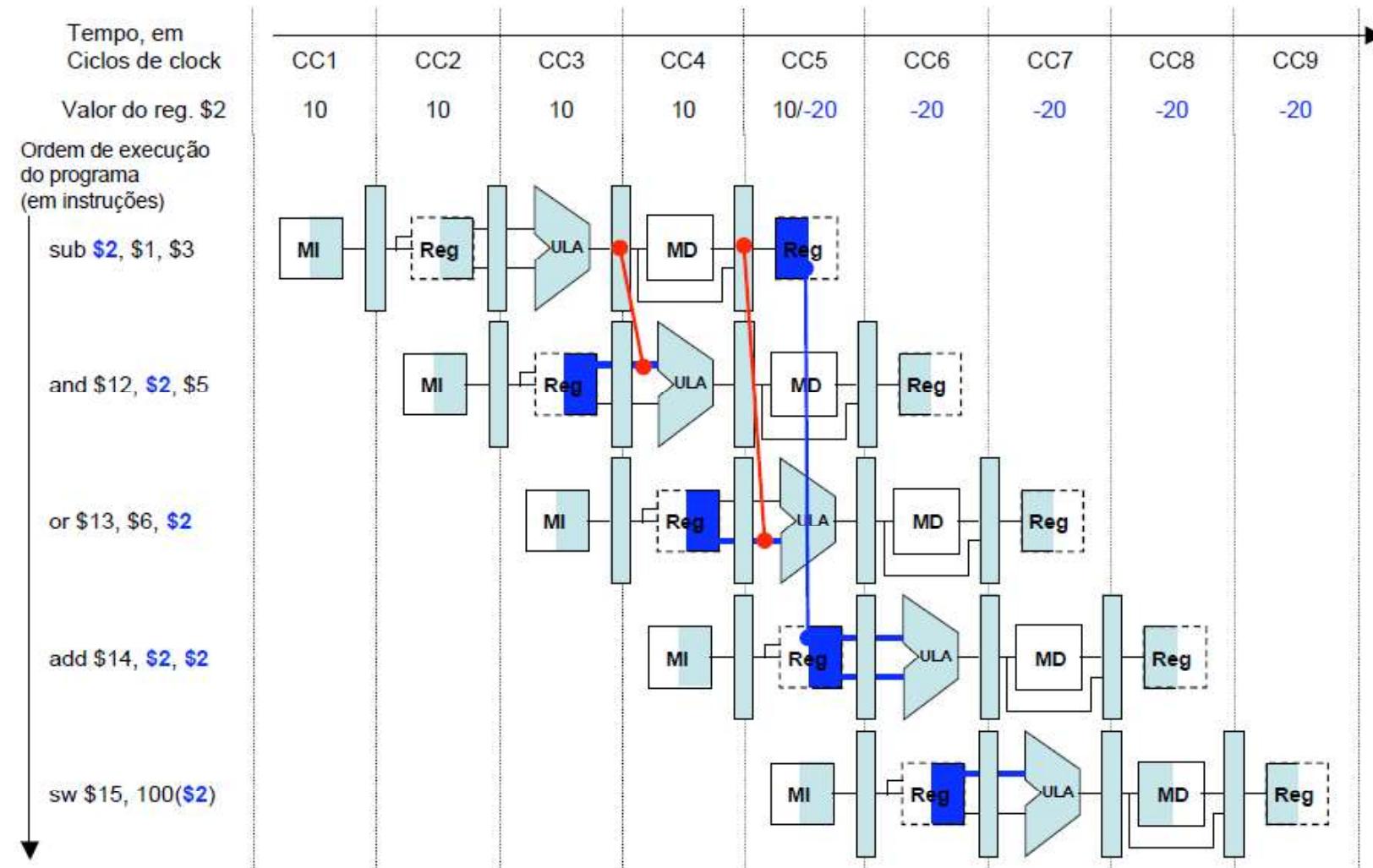
Conflitos por Dados



Conflitos por Dados



Conflitos por Dados



1. Conflitos no Estágio EX

se ($\text{EX/MEM.EscReg} = 1$

e ($\text{EX/MEM.RegistradorRd} \neq 0$)

e ($\text{EX/MEM.RegistradorRd} = \text{DI/EX.RegistradorRs}$)) Adianta.A = 10

se ($\text{EX/MEM.EscReg} = 1$

e ($\text{EX/MEM.RegistradorRd} \neq 0$)

e ($\text{EX/MEM.RegistradorRd} = \text{DI/EX.RegistradorRt}$)) Adianta.B = 10

2. Conflitos no Estágio MEM

se (EX/MEM.EscReg = 1

e (EX/MEM.RegistradorRd \neq 0)

e (MEM/ER.RegistradorRd = DI/EX.RegistradorRs)) Adianta.A = 01

se (EX/MEM.EscReg = 1

e (EX/MEM.RegistradorRd \neq 0)

e (MEM/ER.RegistradorRd = DI/EX.RegistradorRt)) Adianta.B = 01

Complicação:

- Conflito entre o resultado no estágio ER e o resultado no estágio MEM e o operando-fonte da instrução no estágio da ULA

add \$1, \$1, \$2

add \$1, \$1, \$3

add \$1, \$1, \$4

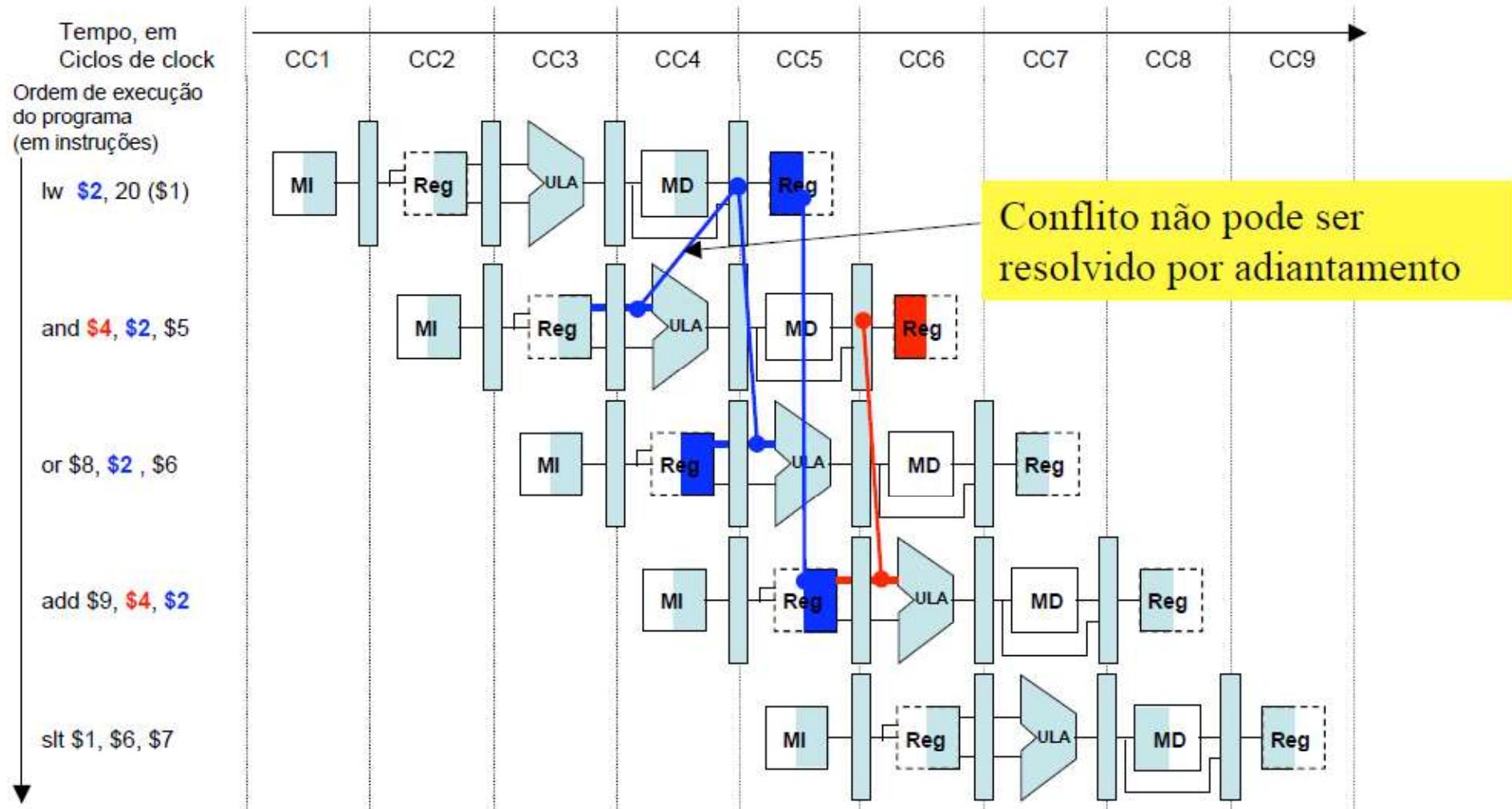
...

Conflitos por Dados e Paradas

- Nem sempre o adiantamento irá resolver um conflito por dados.
- Exemplo:

```
lw    $2, 20($1)      # registrador $2 é escrito  
and   $4, $2, $5      # primeiro operando ($2) depende de lw; registrador $4 é escrito  
or    $8, $2, $6      # primeiro operando ($2) depende de lw  
add   $9, $4, $2      # primeiro operando ($4) depende de and; segundo operando ($2)  
                  depende de lw  
slt   $1, $6, $7      # nenhuma dependencia
```

Conflitos por Dados e Paradas



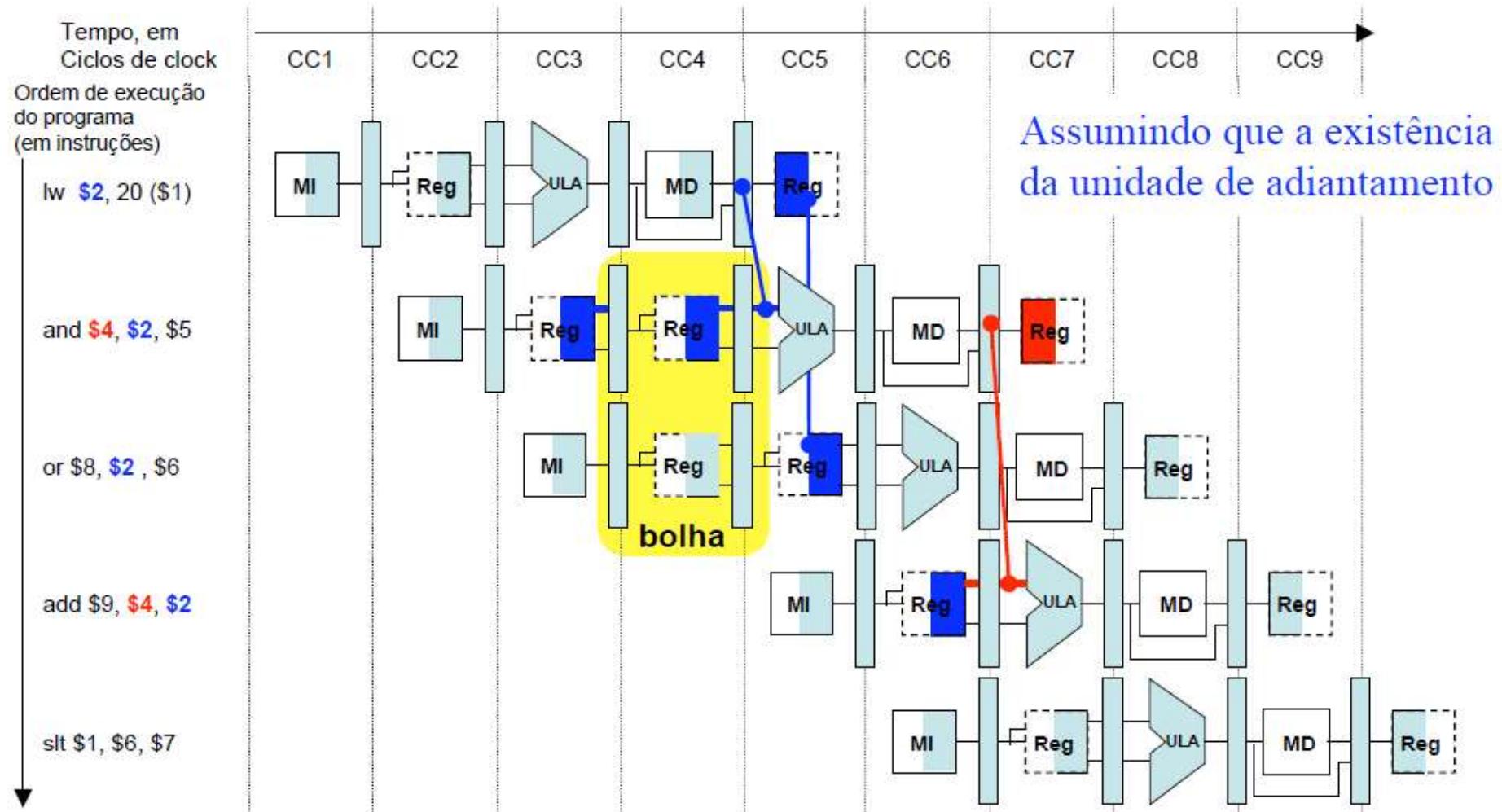
Unidade de Detecção de Conflito

- Faz o pipeline parar quando houver uma instrução load word, seguida de uma instrução que leia o registrador onde esta instrução de load word escreveu
- Vai operar durante o estágio DI, inserindo uma parada entre a instrução load word e o uso de seu resultado
- Condição a ser verificada:

load word é a única instrução que lê dados da memória

Se $(DI/EX.LerMem = 1) \text{ E } ((DI/EX.RegistradorRt} = BI/DI.RegistradorRs) \text{ OU } (DI/EX.RegistradorRt} = BI/DI.RegistradorRt))$
Então pára o pipeline por um ciclo de relógio

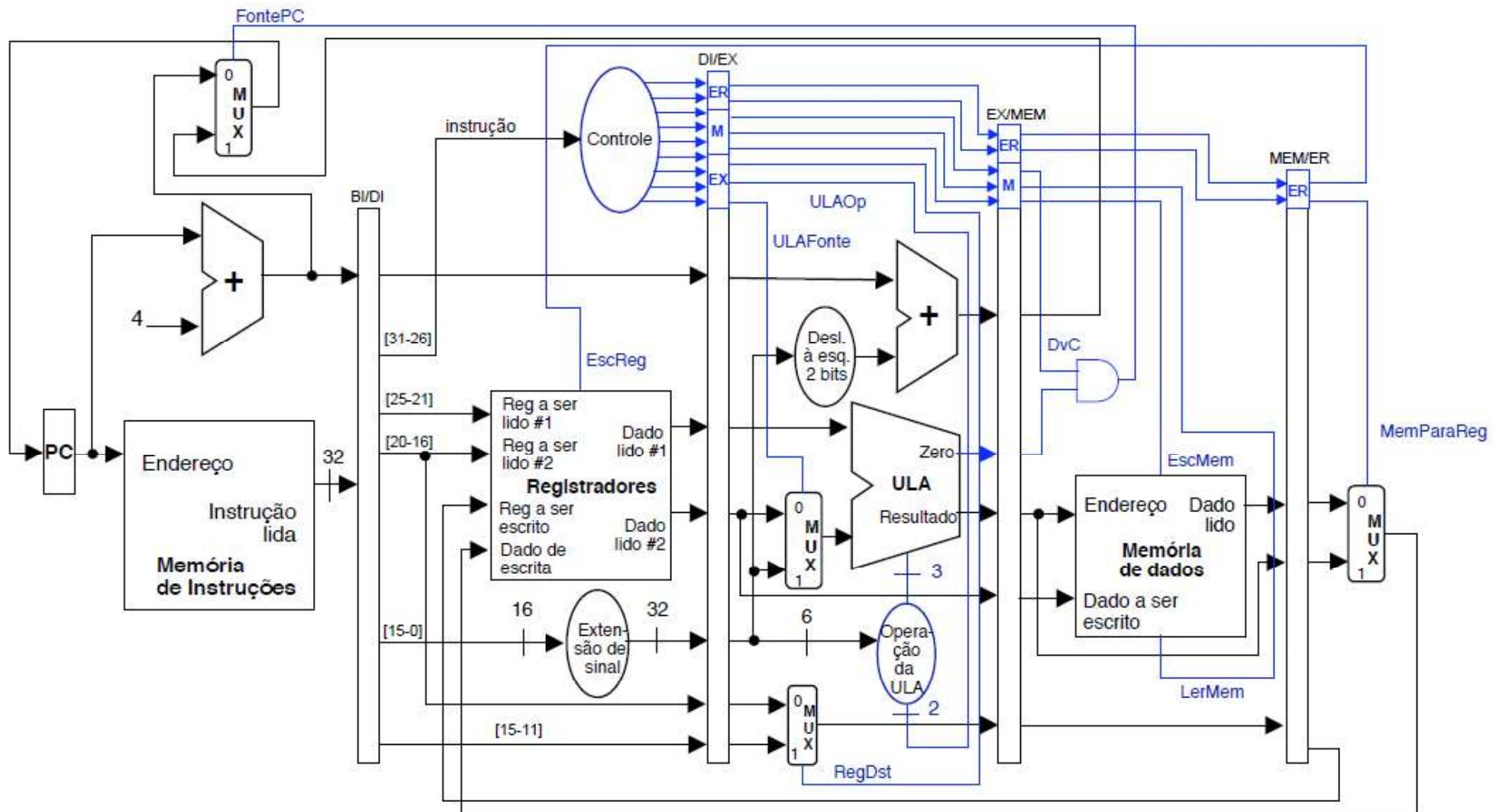
Conflitos por Dados e Paradas



Conflitos por Dados e Paradas

- Trancando o Prosseguimento das Instruções Posteriores a uma Instrução “load word”
 - Se a instrução que está no estágio DI estiver parada, então o estágio BI também precisa parar
 - Para impedir o avanço de instruções pelo pipeline, basta evitar que tanto o PC quanto o registrador BI/DI sejam escritos
 - As condições do item anterior fazem com que, no ciclo de relógio seguinte:
 - A instrução que está no BI seja lida novamente Os registradores lidos no estágio DI serão lidos novamente

Conflitos de Controle



Propagando uma Bolha pelo Pipeline

- Como a instrução load word prossegue pelo pipeline, cria-se uma “bolha” de execução, a qual deve também prosseguir pelo pipeline
- Uma “bolha” deve executar em cada estágio a mesma coisa que uma instrução NOP executa
- NOP:
 - Todos os sinais de controle em 0 (zero) para os estágios EX, MEM e ER.
 - Estes valores de sinais de controle são passados adiante a cada ciclo de relógio, produzindo o efeito desejado (nenhum registrador ou memória é escrito)

Conflitos de Controle (ou Conflitos de Desvios Condicionais)

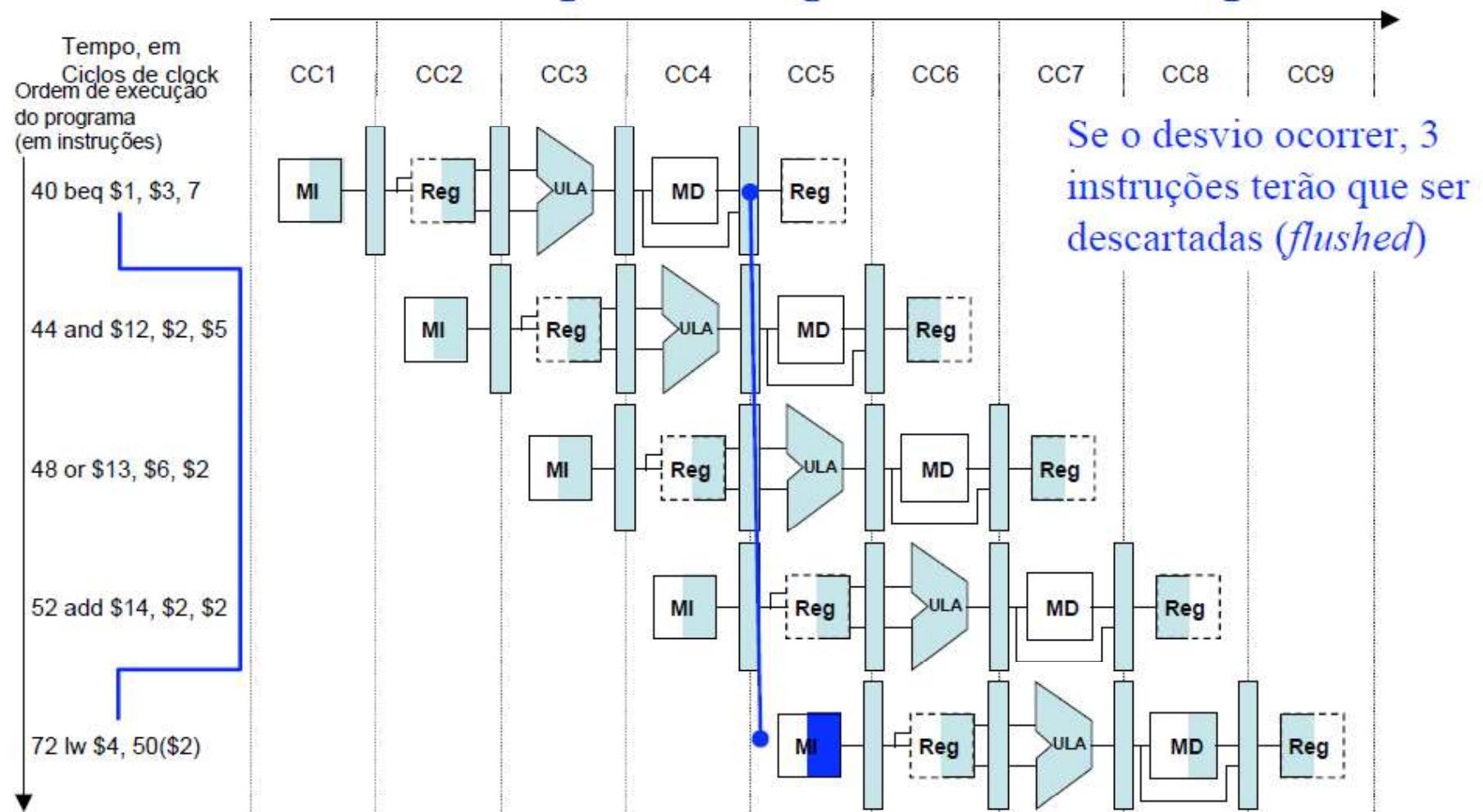
Desvio Condicional em Pipeline.

Exemplo.

```
36    sub    $10, $4, $8
40   beq    $1, $3, 7      # desvio relativo ao PC para  $40 + 4 + 7*4 = 72$ 
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)
```

Conflitos de Controle

Considerando o Bloco Operativo Pipeline Visto Até Aqui...



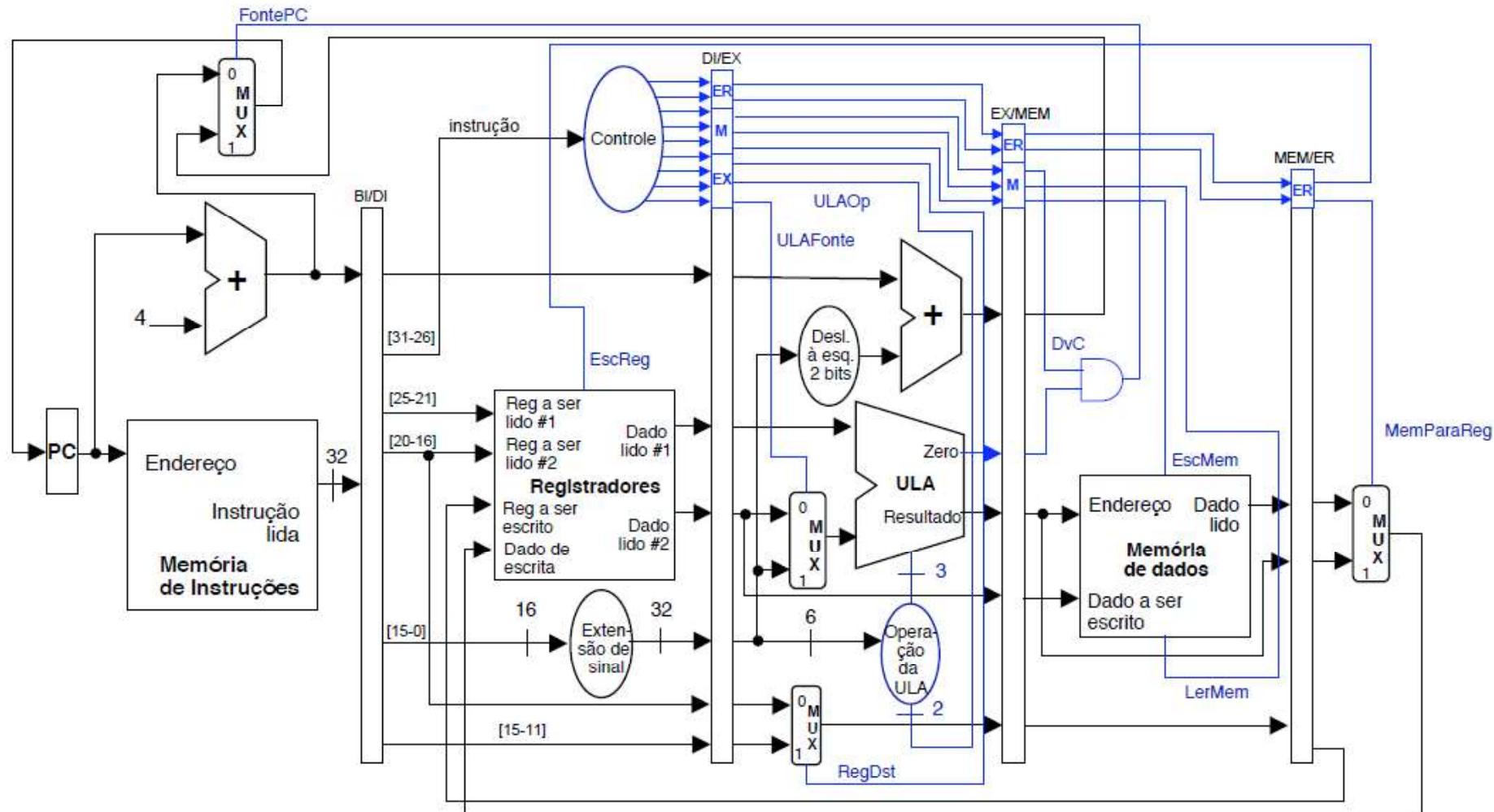
Conflitos de Controle

- Conforme já visto anteriormente, a parada no avanço das instruções não é uma solução viável para o desvio condicional
- Uma alternativa comum é considerar que os desvios condicionais não será tomado, considerando a seqüência normal de execução das instruções
- Caso o desvio se realize, será necessário descartar as instruções que estiverem sendo buscadas e executadas
- E a execução deve continuar a partir da instrução armazenada no endereço-alvo do desvio condicional...

Conflitos de Controle

- Para descartar instruções, basta mudar para 0 os valores originais dos sinais de controle e
- Também mudar as instruções que estiverem em BI, DI e EX, quando a instrução de desvio condicional chegar ao estágio MEM

Conflitos de Controle



Implementação um pipeline

