Organização e Arquitetura de Computadores

Prof. Lúcio Renê Prade

Hazards

Há situações especiais em que a próxima instrução do programa não pode ser executada no ciclo seguinte. Estes eventos são chamados de ***hazards***.

Veremos três tipos:

A. **Estruturais**
B. **Dados**
C. **Controle**

Hazard Estrutural

Situação de conflito pelo uso (simultâneo) de um mesmo recurso de *hardware*.

Ocorre quando duas instruções precisam utilizar o mesmo componente de *hardware* – para fins distintos ou com dados diferentes – no mesmo ciclo de relógio.

A arquitetura MIPS foi pensada tendo em vista uma implementação em *pipeline*. Por isso, as escolhas feitas facilitam a tarefa dos projetistas em evitar os *hazards* estruturais.

Nesse exemplo a solução foi fzr uma camada de abstração entre a memória RAM e o processador

Exemplo: caso não houvesse duas memórias distintas (dados e instrução), teríamos um *hazard* estrutural no momento em que uma instrução estivesse no 4º estágio da *pipeline* (para efetuar um acesso à memória) e uma nova instrução estivesse para ser buscada.

UNISINOS
DESAFIE O AMANHÃ.

Hazard de Dados

Dado ainda n processado é necessário

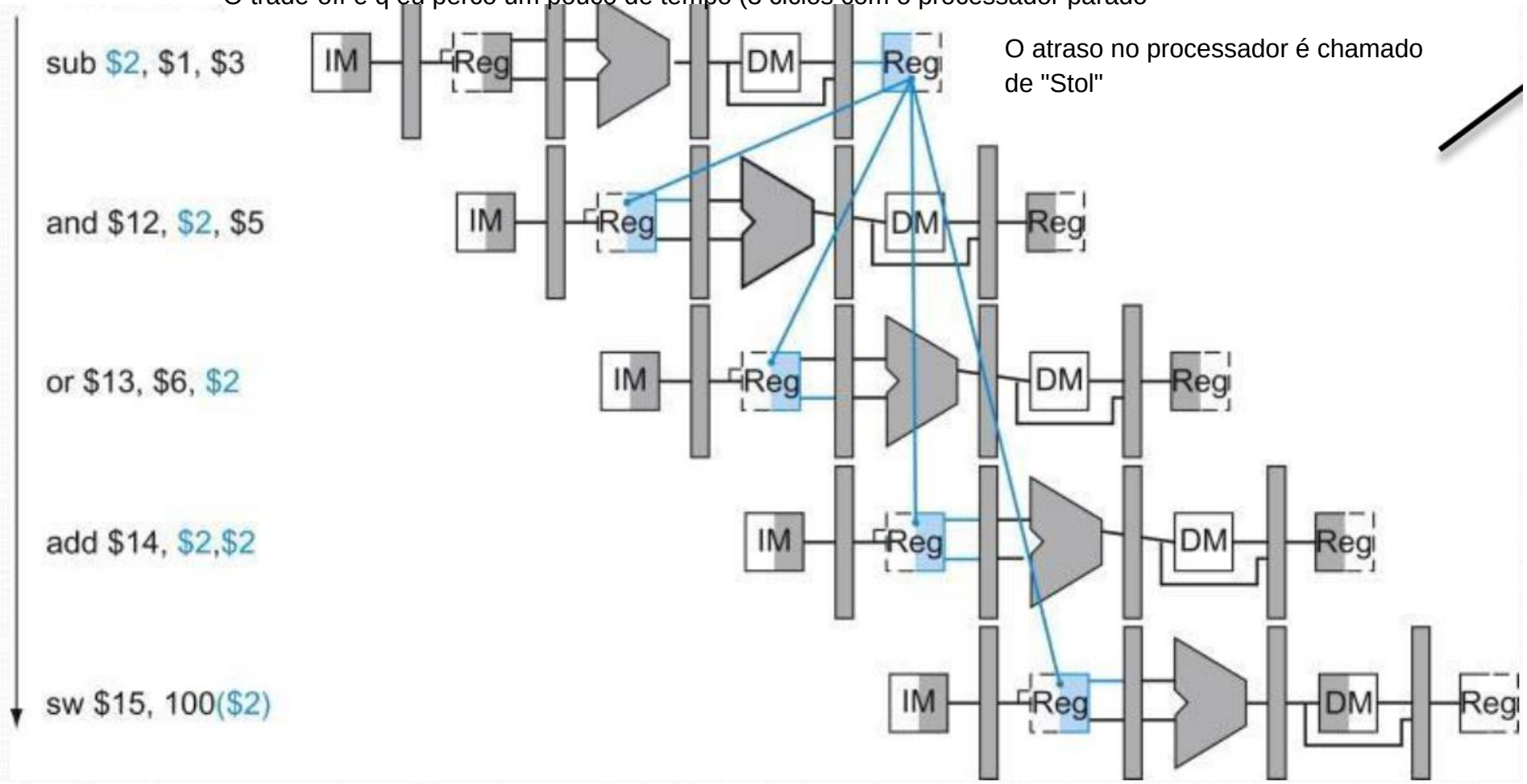
```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

O valor de $2 é atualizado pela operação *sub*. O programador certamente deseja que todas as instruções seguintes utilizem o novo valor contido em $2.

UNISINOS
DESAFIE O AMANHÃ.

Hazard de Dados

Resolução dando um atraso de leve nas instruções seguintes
O trade-off é q eu perco um pouco de tempo (3 ciclos com o processador parado



O atraso no processador é chamado de "Stol"

Não está aqui expresso, mas também é possível rearranjar a ordem das instruções para evitar perder tempo dentro do processador
-> quem faz isso normalmente é o compilador. A menos q seja assembly, aí é o programador mesmo

UNISINOS
DESAFIE O AMANHÃ.

Hazard de Dados

Quando o valor de **$2** está disponível?

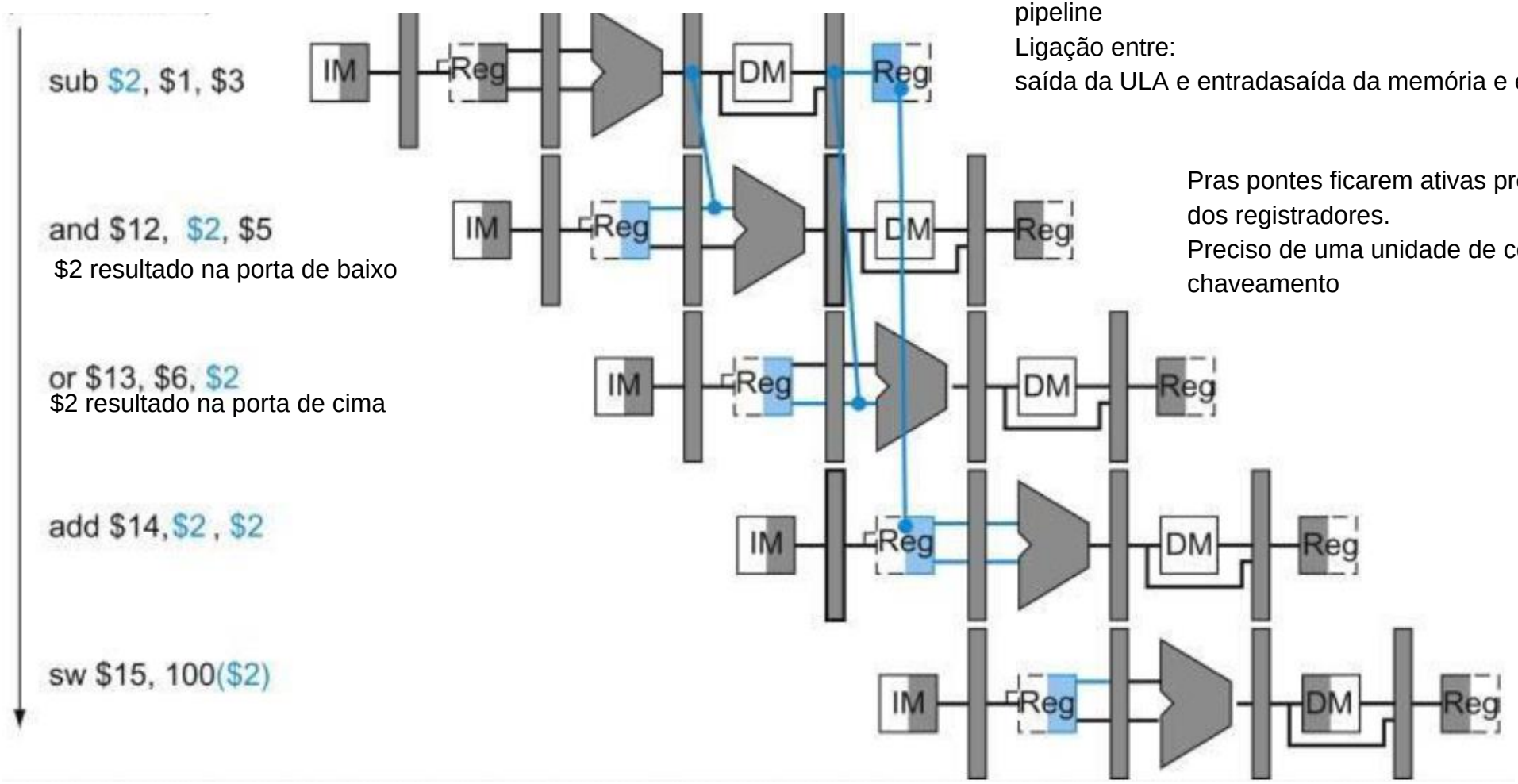
- No final do estágio EX (3º ciclo).

Quando o valor de **$2** é realmente necessário para as operações AND e OR?

- No início do estágio EX (4º e 5º ciclos, respectivamente).

Neste caso, é possível passar adiante (*forwarding*) o valor de **$2** assim que ele estiver pronto para quaisquer unidades que precisem dele antes de ele ser armazenado no arquivo de registradores.

Hazard de Dados

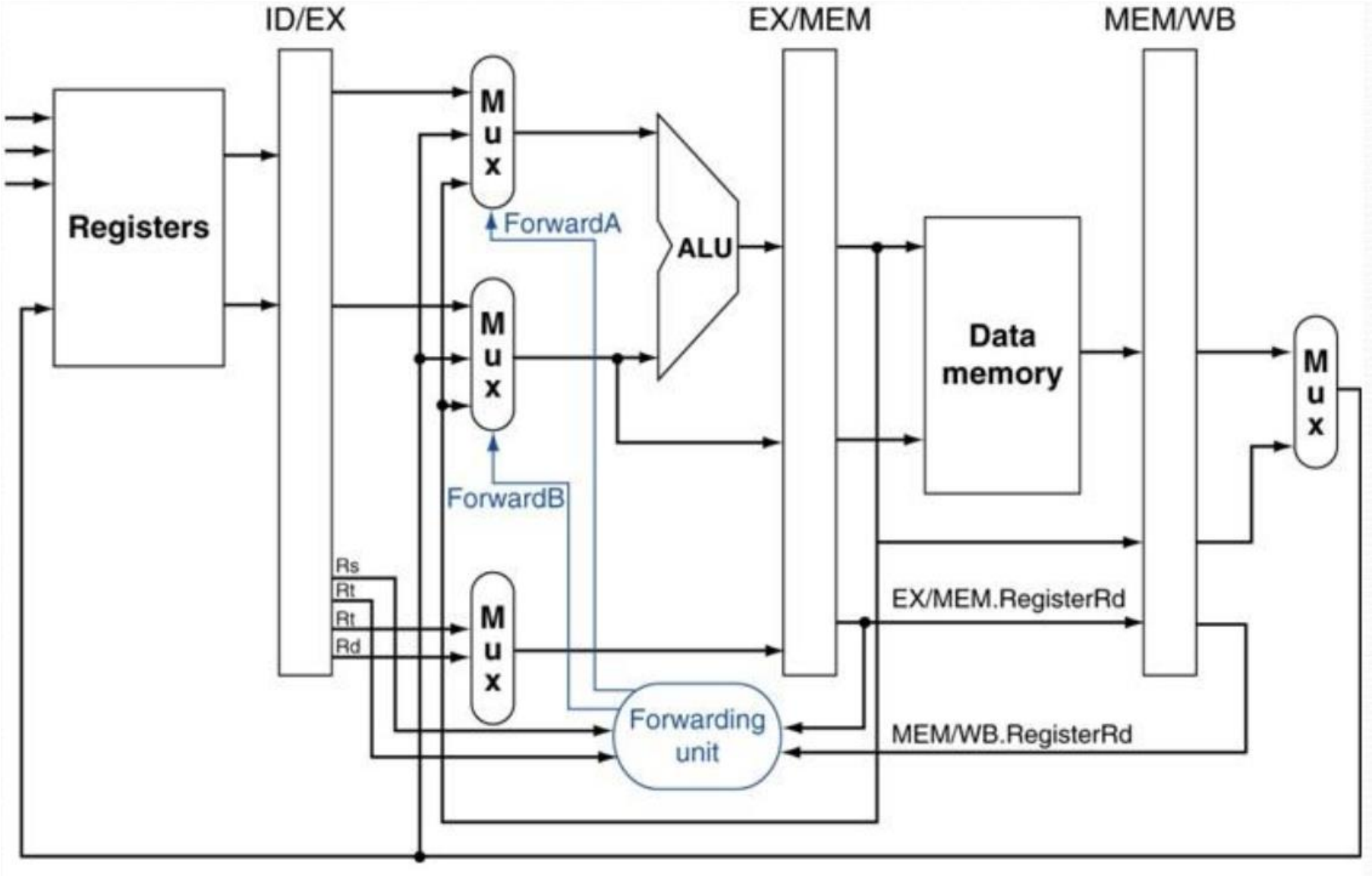


Não precisa reordenar ou atrasar, menos ainda degradar a velocidade da pipeline
Ligação entre:
saída da ULA e entradasaída da memória e entrada da ULA

Pras pontes ficarem ativas precisa comparar os endereços dos registradores.
Preciso de uma unidade de controle específica pra esse chaveamento

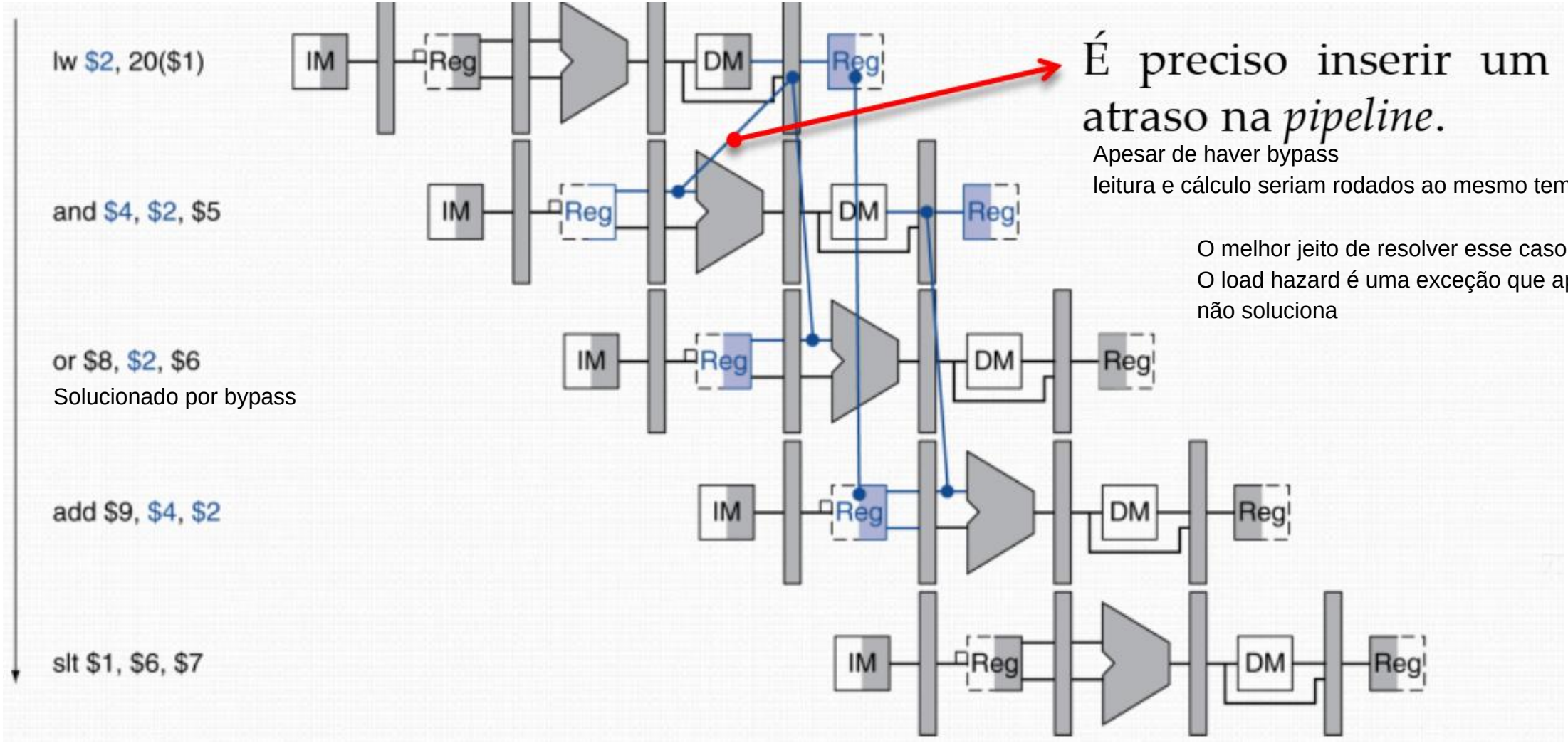
UNISINOS
DESAFIE O AMANHÃ.

Hazard de Dados



UNISINOS
DESAFIE
O AMANHÃ.

Hazard de Dados



Apesar de haver bypass

leitura e cálculo seriam rodados ao mesmo tempo

Solucionado por bypass

O melhor jeito de resolver esse caso é estratégia mista

O load hazard é uma exceção que apenas o bypass não soluciona

UNISINOS
DESAFIE O AMANHÃ.

Hazard de Dados

Stalls reduzem o desempenho da *pipeline*.

- Porém, são necessários para que se garanta o resultado correto.

Compiladores podem rearranjar o código a fim de evitar a ocorrência de *hazards* e *stalls*.

- Isto requer um conhecimento a respeito da estrutura da *pipeline*.

Hazard de Controle

Entenda controle por desvio condicional (IF) - jumps condicionais

Surge por causa da necessidade de tomar uma decisão baseada em resultados de uma instrução enquanto outras estão em execução.

BEQ (desvia-se igual) $1 $2 100
se o valor dos 2 registradores é igual vai pra PC+100
se for diferente é falso e faz PC+4 (sempre pula de 4 em 4 por causa dos Bytes)

Ou seja, está ligado a instruções do tipo *branch*.

Problema:

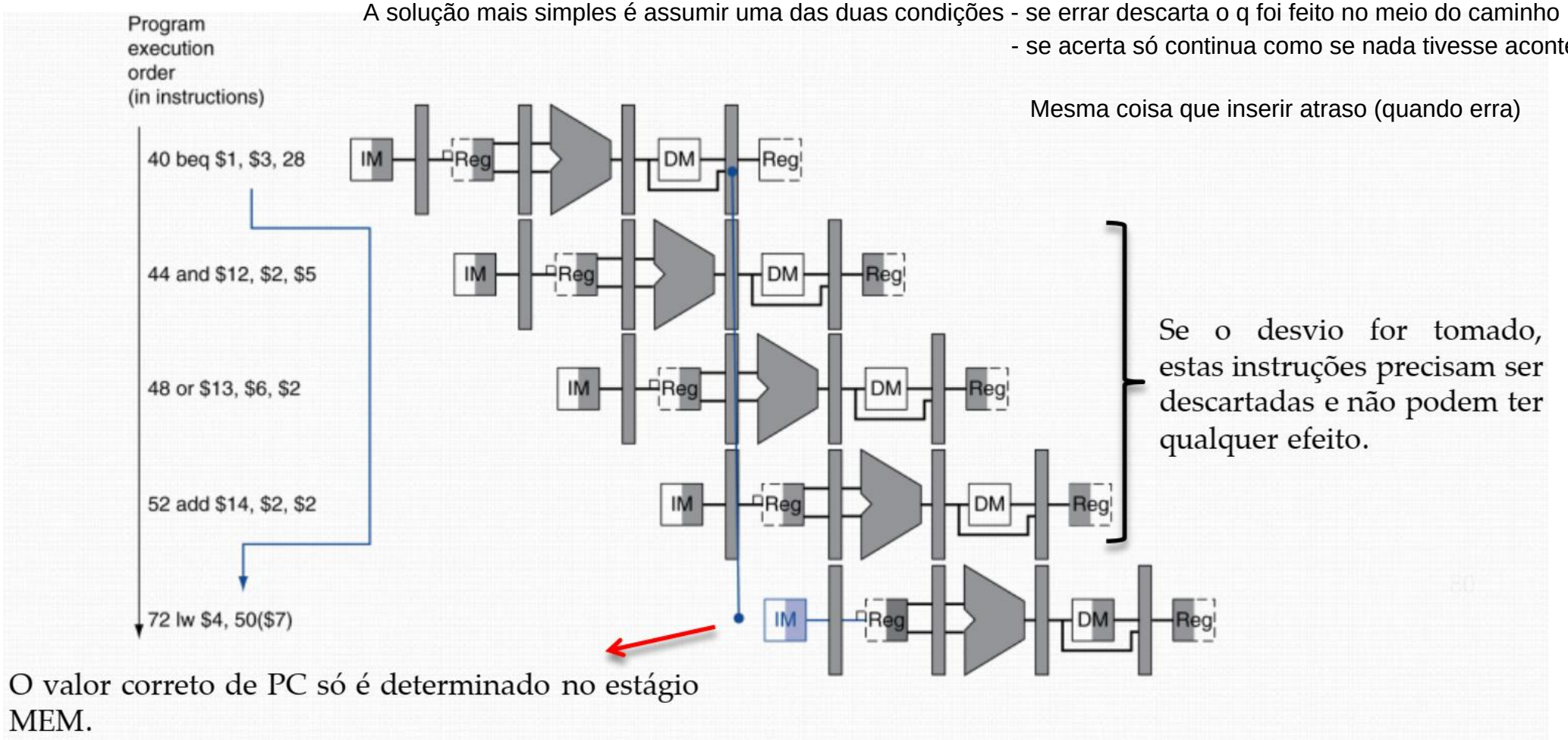
- A *pipeline* inicia a busca da instrução subsequente ao *branch* no próximo ciclo de relógio.
- Porém, não há como a *pipeline* saber qual é a instrução correta a ser buscada, uma vez que acabou de receber o próprio *branch* da memória.

UNISINOS
DESAFIE O AMANHÃ.

Hazard de Controle

Não sabe qual instrução é antes de terminar a instrução toda
A solução mais simples é assumir uma das duas condições - se errar descarta o q foi feito no meio do caminho
- se acerta só continua como se nada tivesse acontecido

Mesma coisa que inserir atraso (quando erra)



O valor correto de PC só é determinado no estágio MEM.

UNISINOS
DESAFIE O AMANHÃ.

Hazard de Controle

Como já comentamos anteriormente, uma solução simples seria sempre imaginar que o desvio não será tomado.

Se ele for tomado, as instruções sendo buscadas e decodificadas precisam ser descartadas.

Flush: limpar a *pipeline* – é preciso zerar os sinais de controle para todas as instruções (nos estágios IF, ID e EX) quando o *branch* atinge o estágio MEM.

Hazard de Controle

Previsão dinâmica:

- Tentativa de prever o comportamento de um desvio durante a execução do programa.

Essa instrução sendo normalmente associada a repetições, aumenta a taxa de acertos do sistema e faz sentido guardar histórico

Sistemas multiprocessos quebram as pernas desse tipo de implementação - por isso não são usados com multithreading

- **Possibilidade:** buffer de previsão (tabela de histórico de desvio)
  - Indexada pelos bits menos significativos do endereço da instrução de desvio.
  - Contém 1 bit que informa se o *branch* foi recentemente tomado ou não.
  - **Desvantagem:** mesmo se um desvio quase sempre for tomado, é possível cometer dois erros de previsão, em vez de apenas um.