

# Arquitetura e Organização de Computadores 1

Fonte: Adaptado de Digital Design and Computer Architecture RISC-V Edition Sarah L. Harris David Harris

Prof. Luciano de Oliveira Neris  
luciano@dc.ufscar.br

# Hazards

# Pipeline Hazards

- Em uma organização pipelined, a princípio uma nova instrução inicia a execução em cada ciclo.
  - Infelizmente, nem sempre isso é possível. Há situações em que a próxima instrução não pode ser executada no ciclo de clock seguinte. 😞
  - Essas situações são chamadas de pipeline hazards, ou simplesmente **hazards**.
    - *Obs: tradução de hazards → "riscos, perigos"*  
*tradução de hazards mais adequada → "conflitos"*
- Hazards podem reduzir significativamente o speedup alcançado
- Por isso, várias soluções engenhosas foram desenvolvidas para lidar com o problema

# Hazards

- Existem três tipos de hazards:
  - Hazards estruturais
    - Conflito de unidades funcionais
  - Hazards de controle
    - O que ocorre em desvios condicionais e incondicionais
  - Hazards de dados
    - Ocorre quando uma etapa precisa esperar até que outra seja executada.
    - É necessário uma intervenção para obter o resultado correto da operação

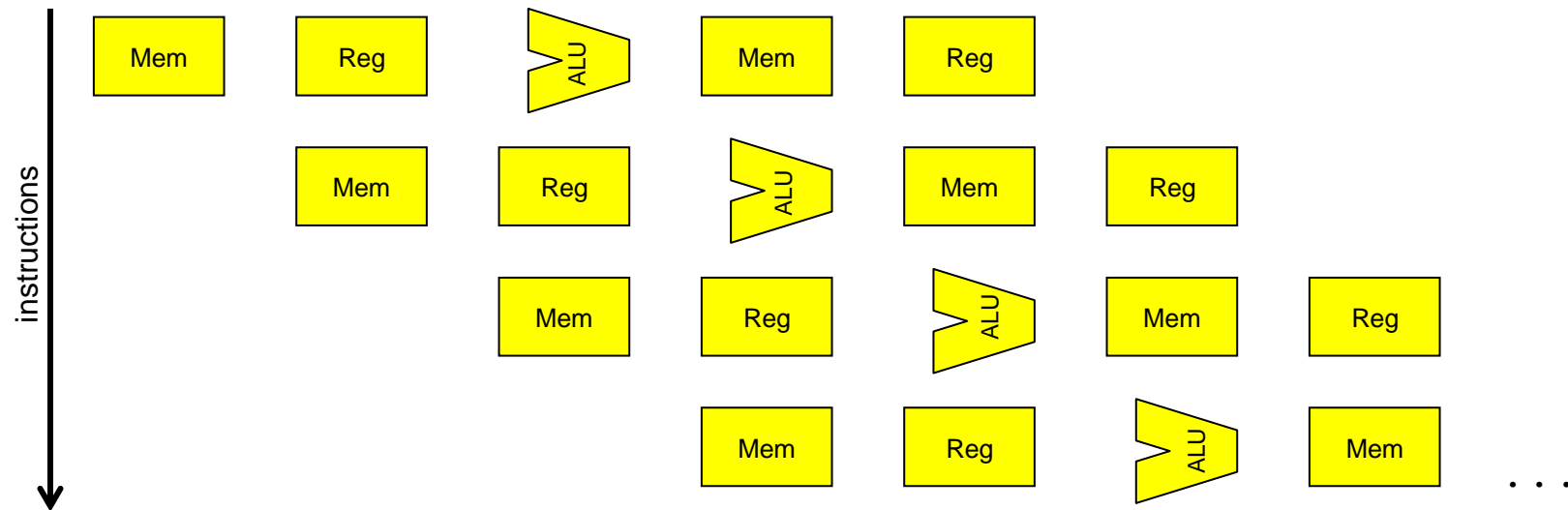
# Hazards Estruturais

# Hazards Estruturais

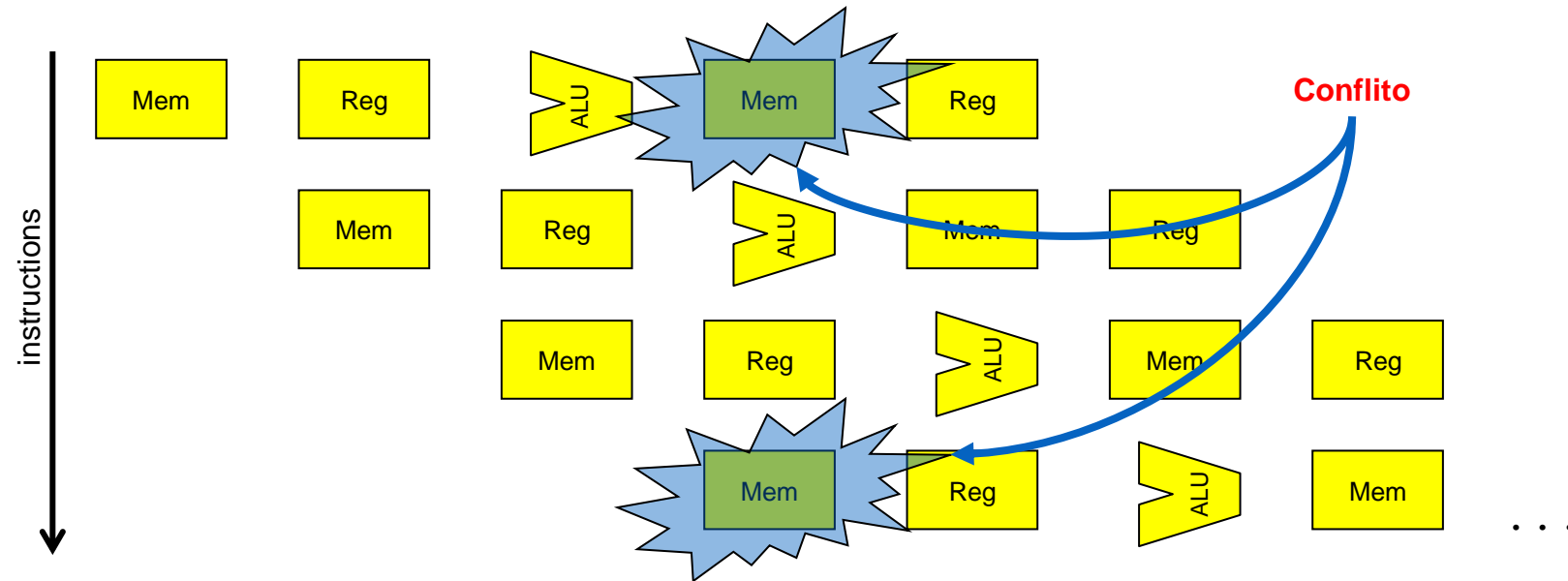
- Situação de conflito pelo **uso** (**simultâneo**) de um mesmo recurso de hardware.
- Ocorre quando duas instruções precisam **utilizar** o **mesmo** componente de **hardware** - para fins distintos ou com dados diferentes - no mesmo ciclo de relógio.

# Hazards Estruturais: Exemplo 1

- Exemplo: Suponha que nosso datapath **não** tenha memórias separadas p/ instruções e dados.
- Nesse caso, toda instrução que faz acesso à memória (Load/Store) irá entrar em conflito no estágio de busca de uma instrução, ou seja, um hazard estrutural.
- Datapaths como esse **não são considerados "fully pipelined"**



# Hazards Estruturais: Exemplo 1





# Hazards Estruturais: Exemplo 1

- Considere a seguinte sequencia de instruções:

LW R12, 100(R2)

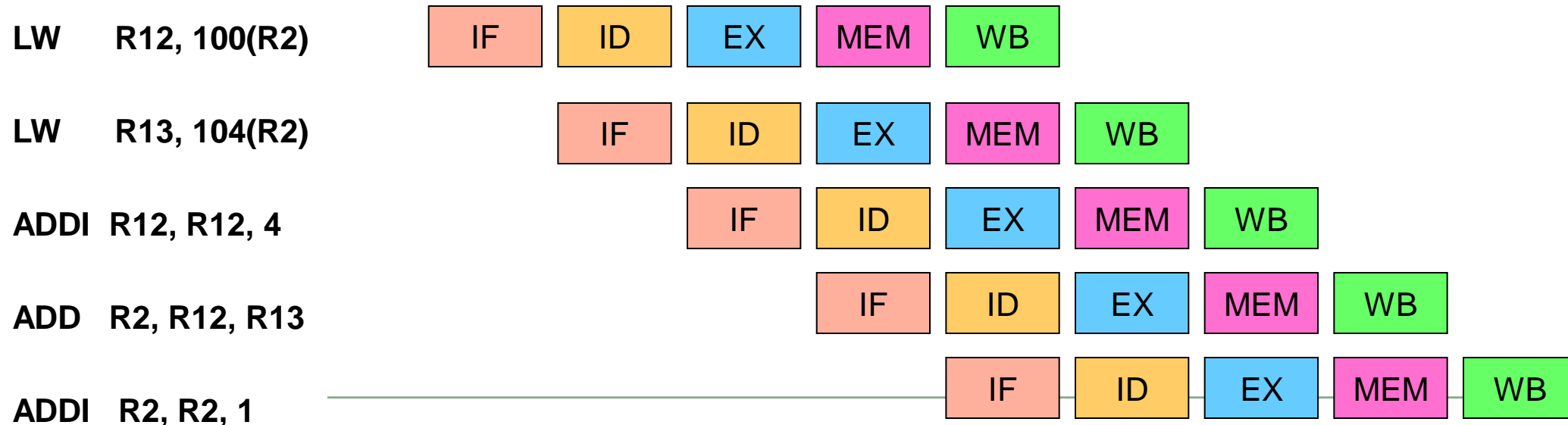
LW R13, 104(R2)

ADDI R12, R12, 4

ADD R2, R12, R13

ADDI R2, R2, 1

- Quais são os conflitos de hardware que ocorrerão ?



# Hazards Estruturais: Exemplo 1

- Aqui temos dois conflitos

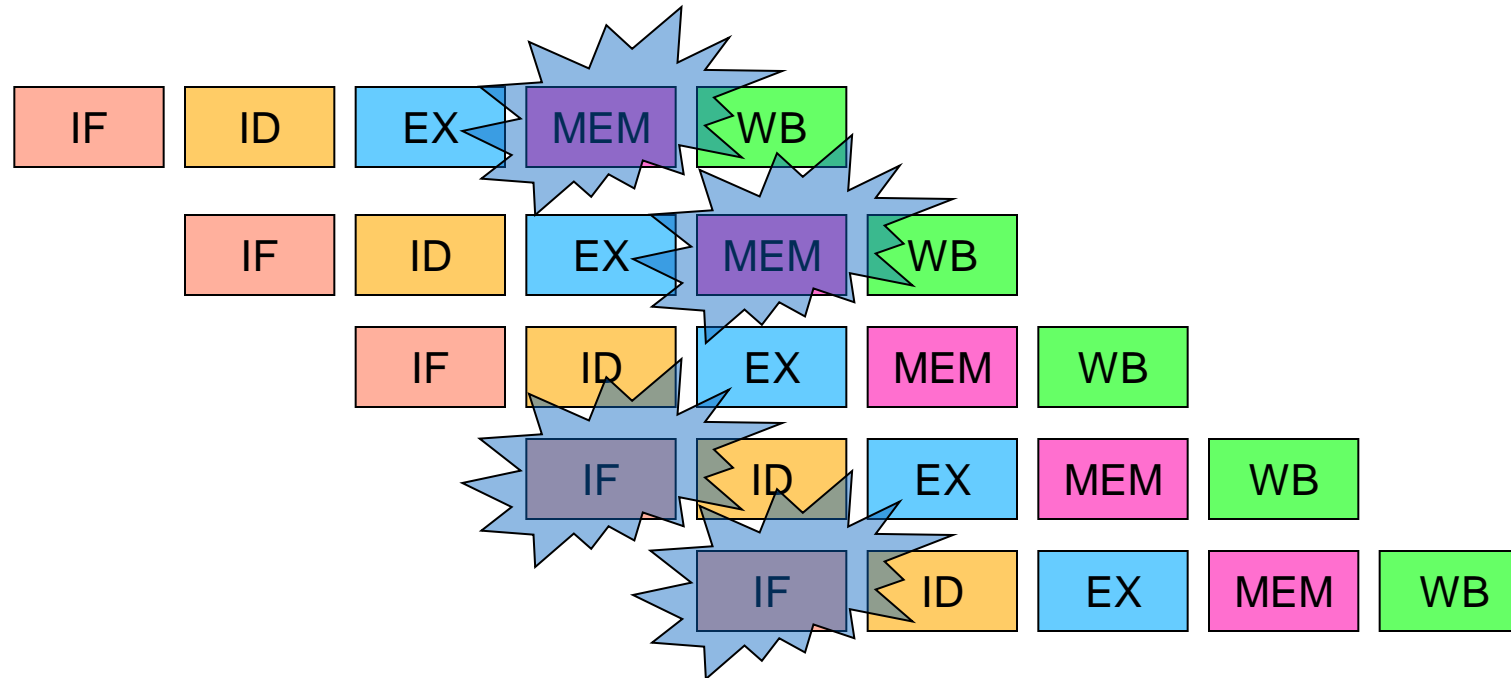
LW R12, 100(R2)

LW R13, 104(R2)

ADD R12, R12, 4

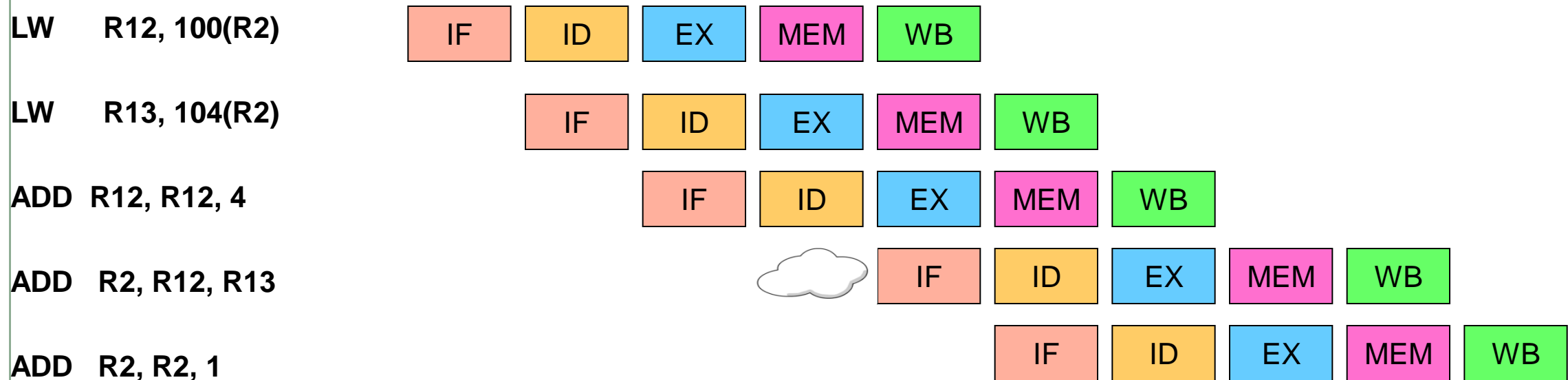
ADD R2, R12, R13

ADD R2, R2, 1



# Hazards Estruturais: Exemplo 1

- Solução p/ o primeiro conflito: inserir uma parada (stall)

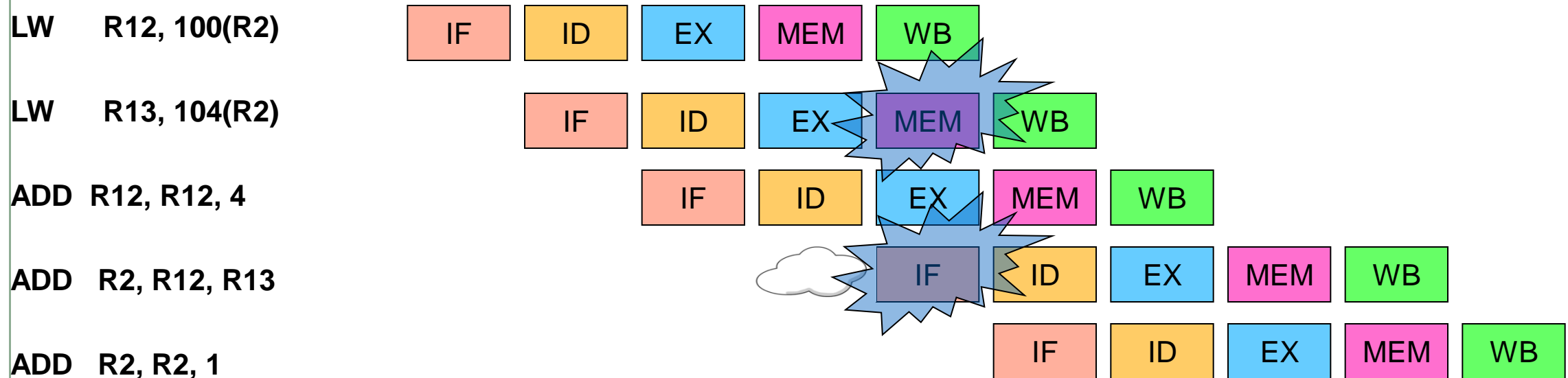


# Hazards Estruturais: Exemplo 1

- Stall significa **Parar** a execução de uma instrução em um estágio intermediário do pipeline, esperando um evento qualquer terminar (também chamado de “bolha”).
- Quando uma instrução para, **todas** as **instruções** emitidas após ela, ou seja, que estão atrás no pipeline, também **são paradas**
  - Obs: as instruções emitidas anteriormente, ou seja, que estão à frente no pipeline, continuam executando normalmente.
- **Nenhuma** nova instrução é buscada / emitida durante uma parada

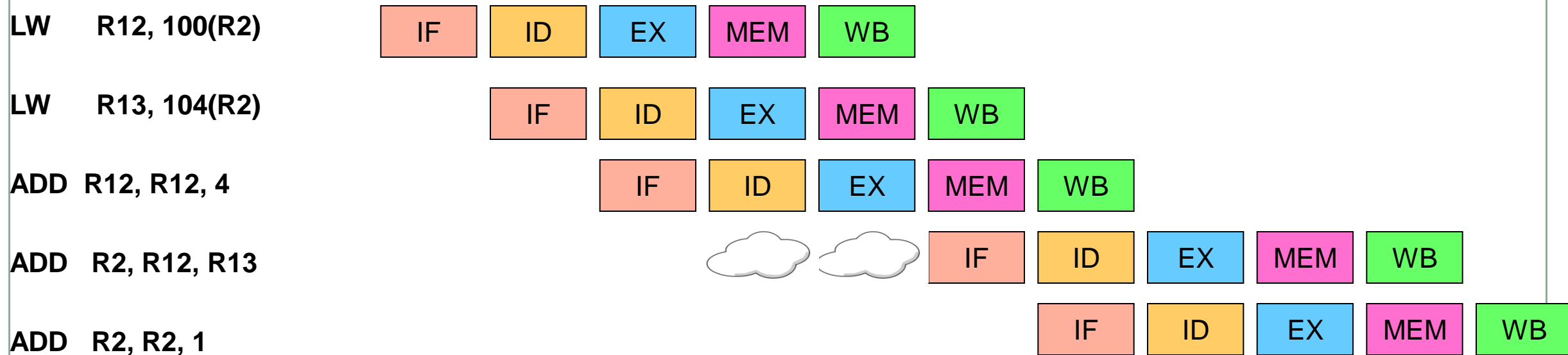
# Hazards Estruturais: Exemplo 1

- Mas ainda temos o outro conflito!



# Hazards Estruturais: Exemplo 1

- Solução p/ o segundo conflito: inserir outra parada (stall)



# Hazard Estrutural: Desempenho

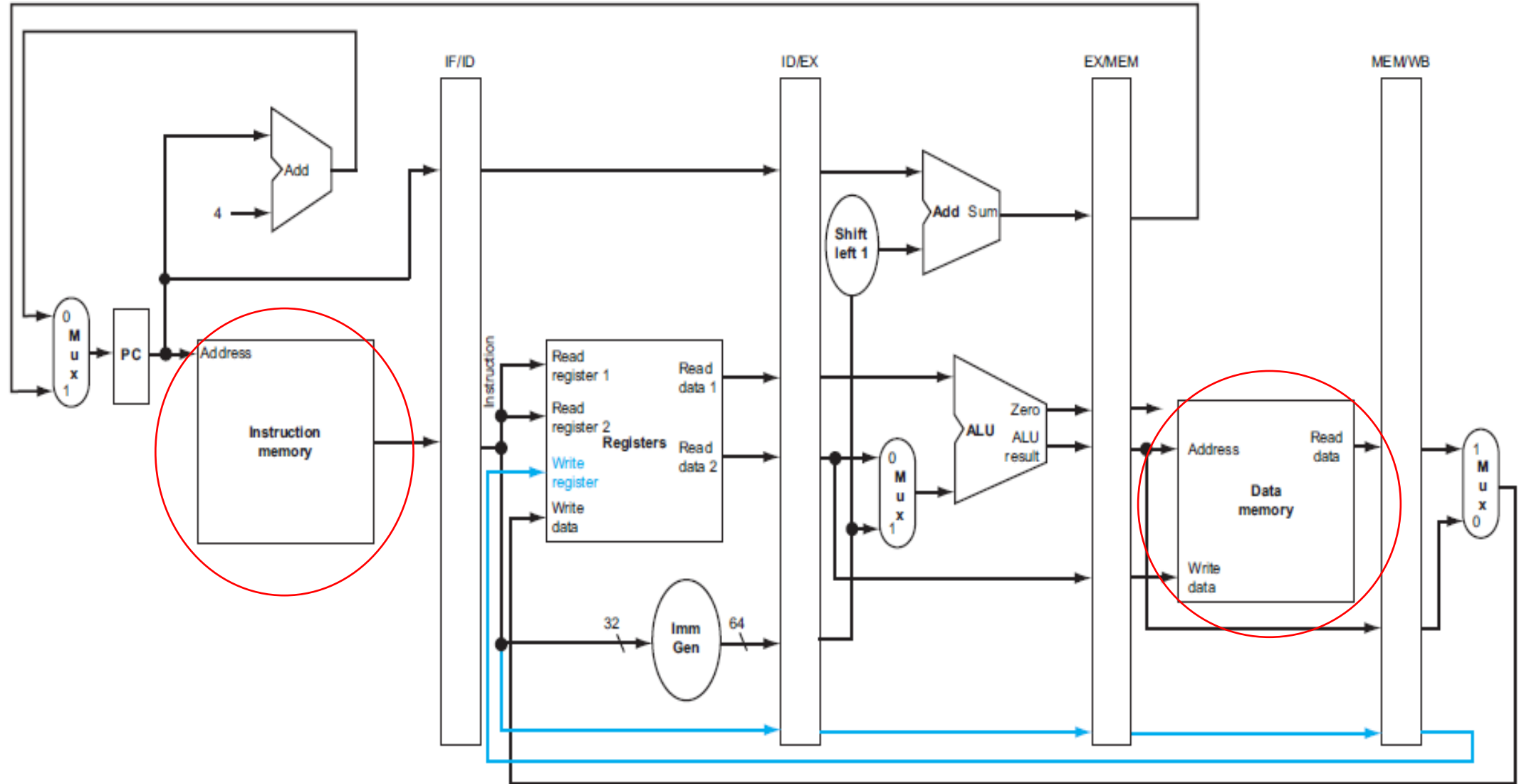
- Implementação da Inserção de Paradas
  - Durante a decodificação da instrução, se for um Load ou Store, insira uma parada no segundo estágio à frente
- Obviamente que isso causa impacto no desempenho do processador, e estes são bastante frequentes no caso de Loads e Stores!

# Resposta

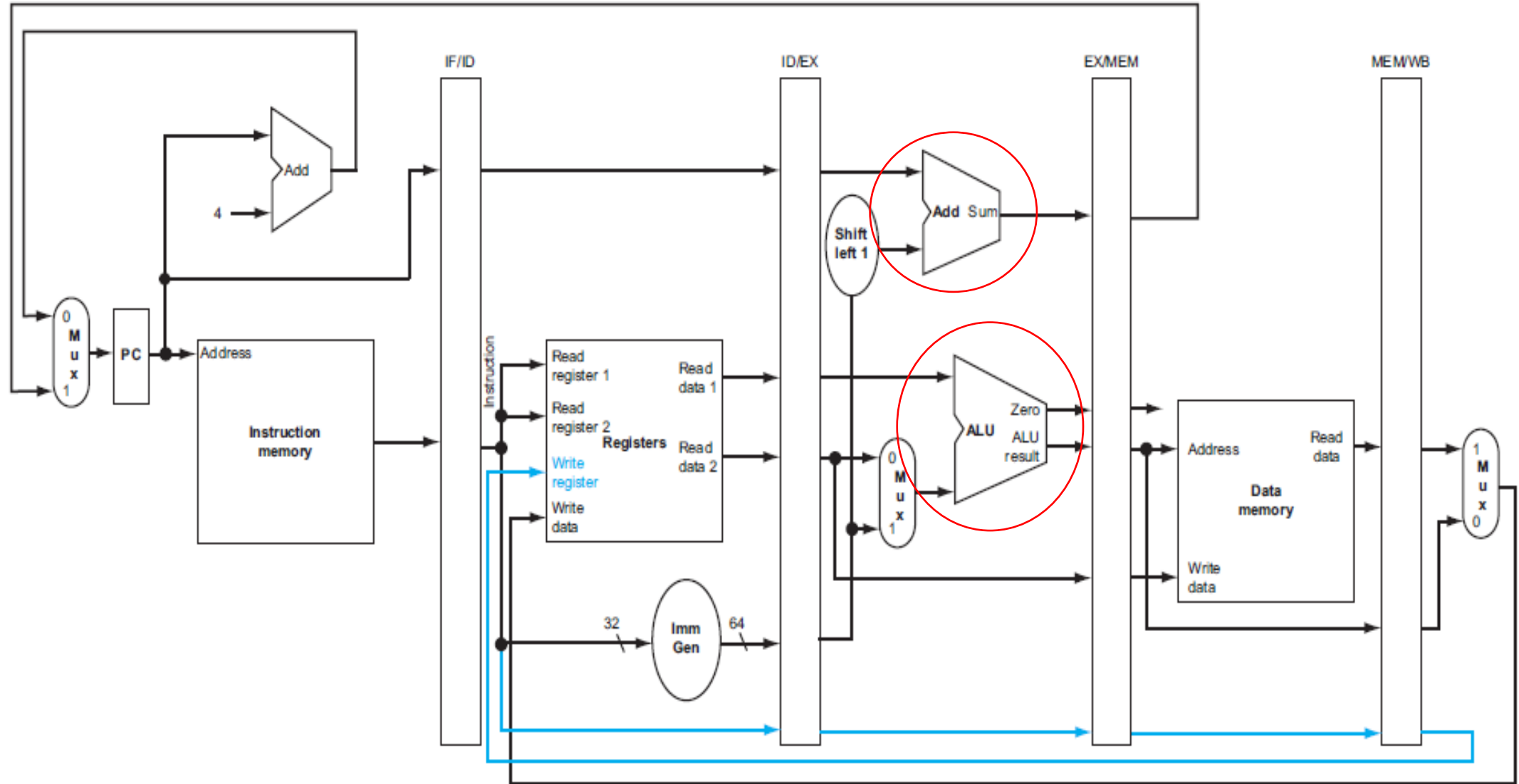
- Como o hardware estrutural do exemplo pode ser resolvido?
  - Resposta → Utilizando memórias separadas para dados e instruções
- Em processadores modernos isso é alcançado por meio de caches "on-chip" separados, para dados e instruções
  - Ex: Processadores Intel, AMD: I-Cache, D-Cache



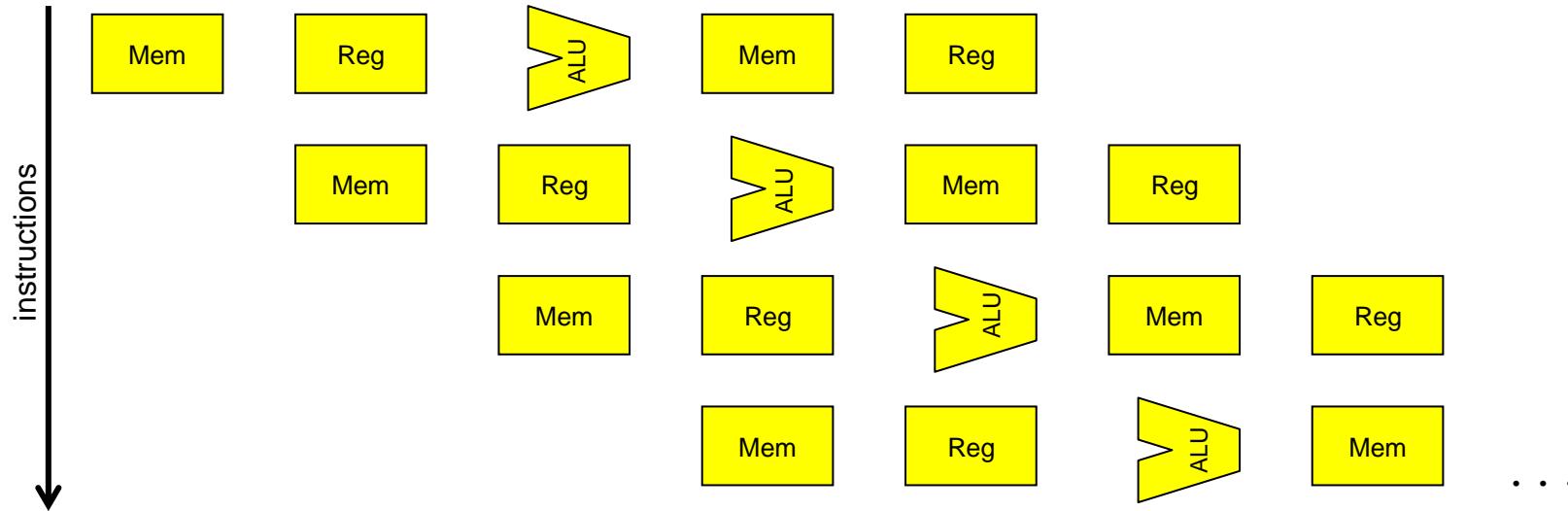
# Hazard Estrutural



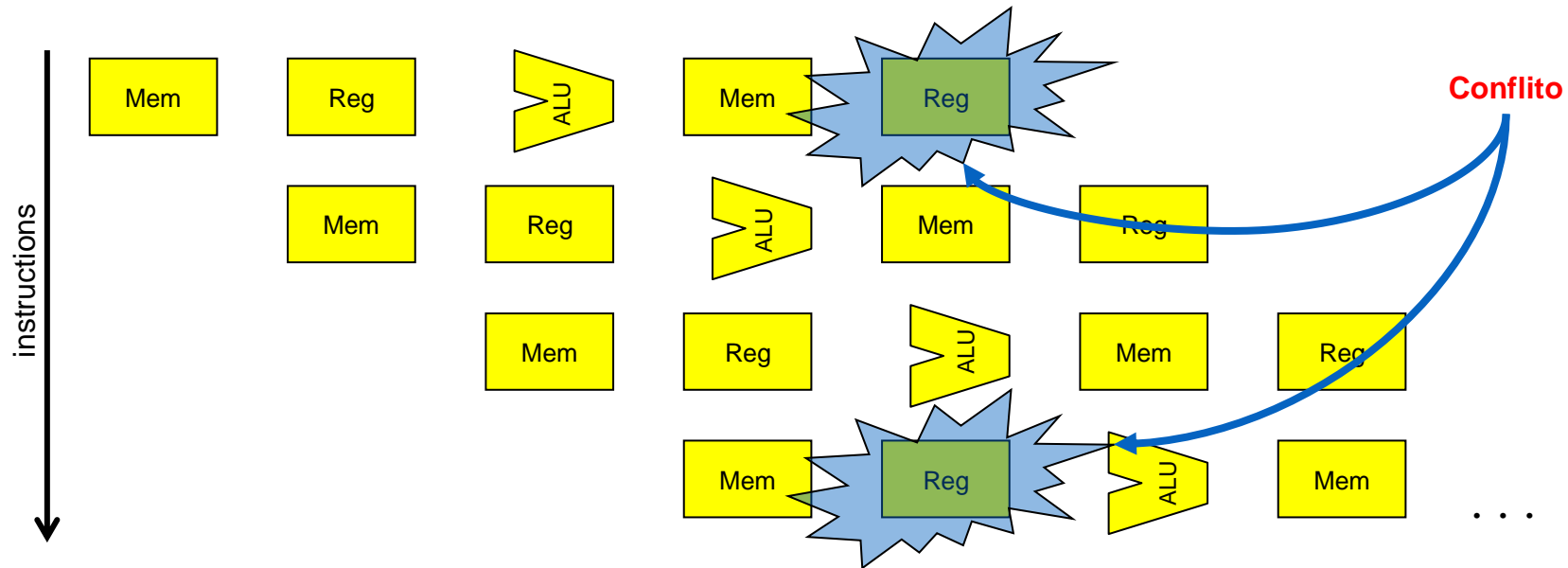
# Hazard Estrutural



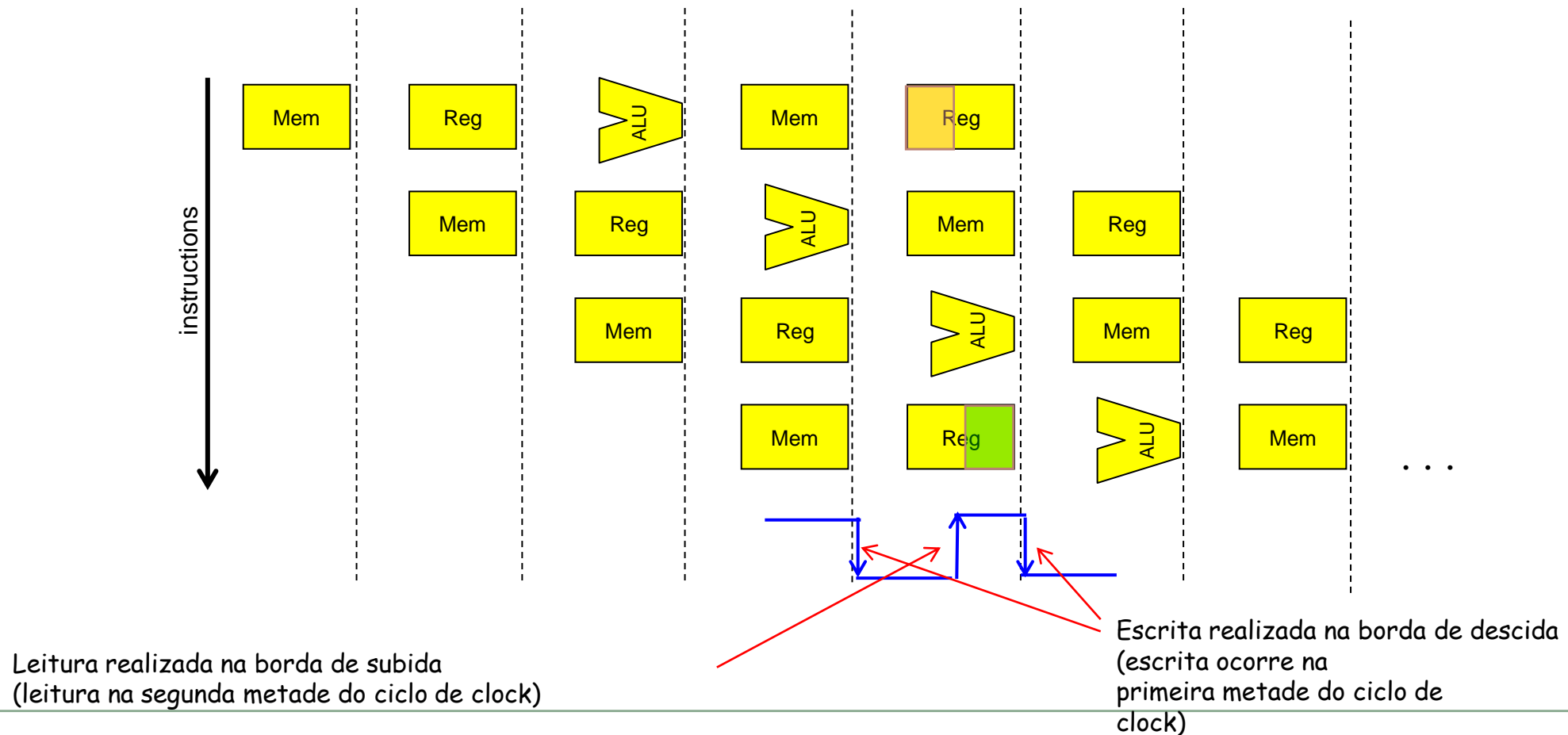
# Hazards Estruturais: Exemplo 2



# Hazards Estruturais: Exemplo 2



# Hazards Estruturais: Exemplo 2



# Hazards Estruturais

- A **arquitetura RISC-V** foi pensada tendo em vista uma implementação em pipeline. Por isso, as escolhas feitas **facilitam** a tarefa dos projetistas em evitar os hazards estruturais.
  - Caso **não** houvesse duas memórias distintas (dados e instrução), teríamos um hazard estrutural no momento em que uma instrução estivesse no 4º estágio do pipeline (para efetuar um acesso à memória) e uma nova instrução estivesse para ser buscada.

# Hazards de Dados

# Hazards de Dados

- Ocorrem quando uma instrução **depende** da **conclusão** de uma instrução prévia que ainda esteja no pipeline para realizar sua operação e/ou acessar um dado.
- Exemplo:  
    add s0, t0, t1  
    sub t2, s0, t3
  - A instrução add somente escreve seu resultado no final do 5º estágio do pipeline.
  - Logo, teríamos que desperdiçar três ciclos de relógio aguardando até que o resultado correto (s0) pudesse ser lido pela instrução sub.



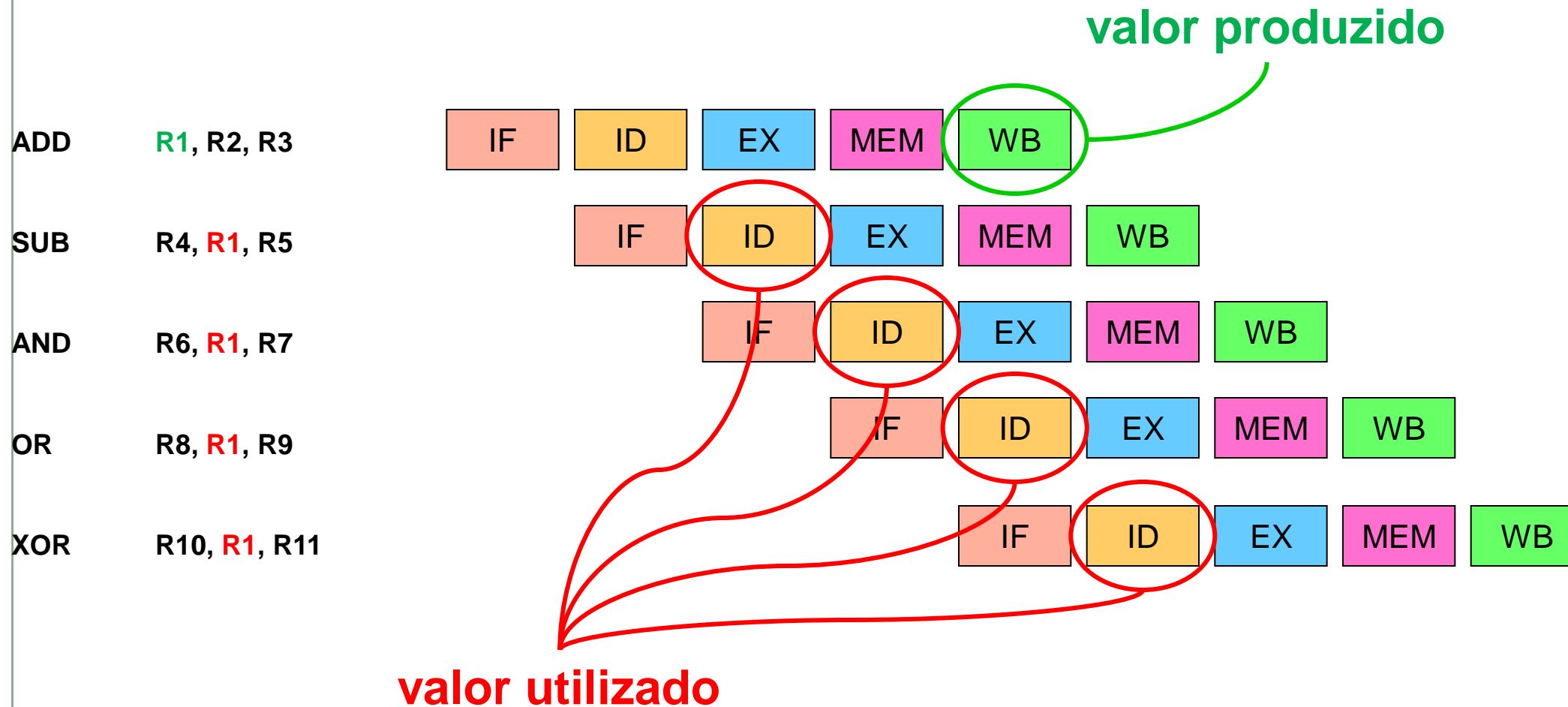
# Hazards de Dados

- Considere a seguinte sequencia de instruções:

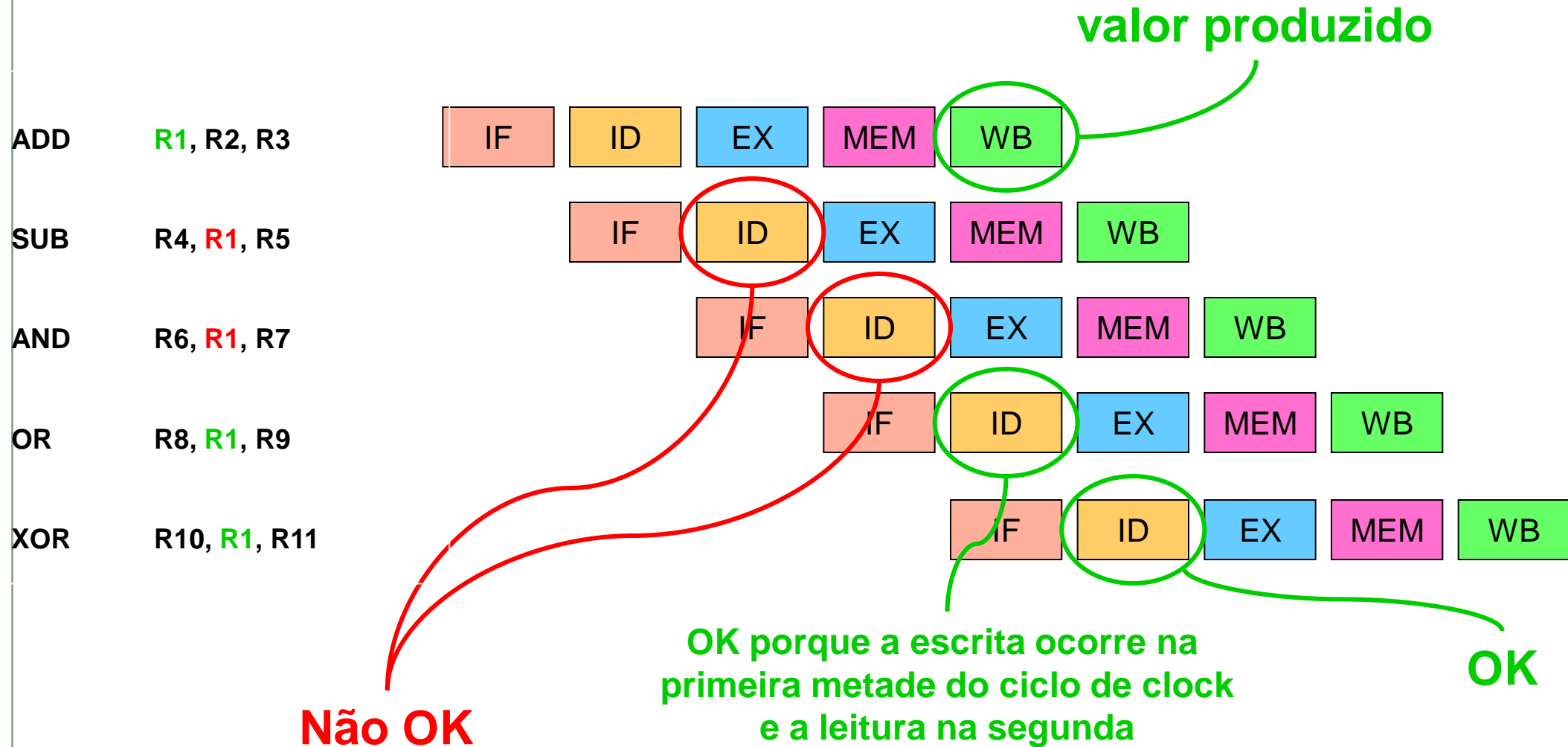
ADD	R1, R2, R3
SUB	R4, R1, R5
AND	R6, R1, R7
OR	R8, R1, R9
XOR	R10, R1, R11

- Todas as instruções utilizam o resultando produzido pela instrução ADD
- Sequencialmente isso é ok, mas pode causar problemas em nosso pipeline →

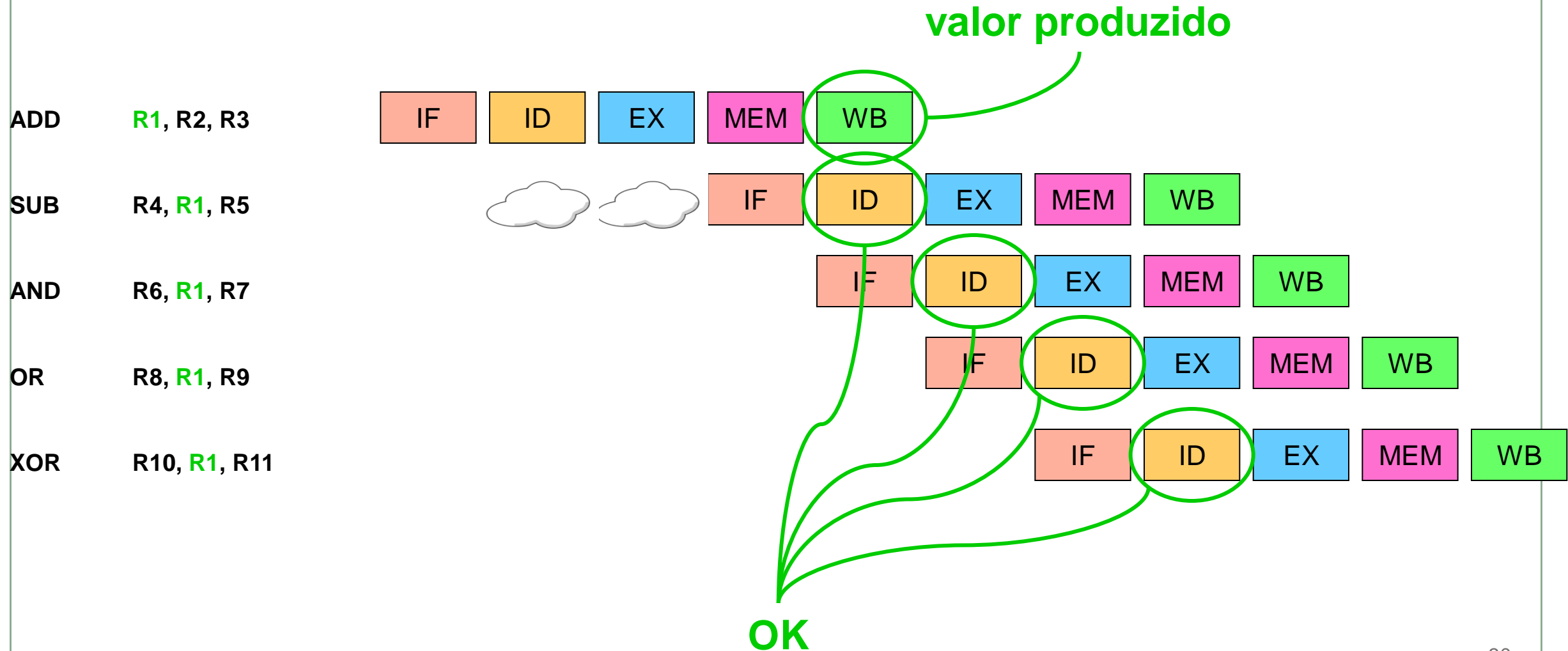
# Hazards de Dados: Exemplo



# Hazards de Dados



# Hazards de Dados



# Hazards de Dados

- Conforme visto anteriormente, inserir paradas (stalls) pode causar grandes perdas de desempenho
- Stalls → solução de **último caso**, deve ser evitado!
- No exemplo anterior, observa-se que:
  - instrução #1 calcula o valor necessário pelas demais em seu estágio **EX**

# Hazards de Dados

- Uma possível solução pode ser proposta a partir da observação de que não é necessário aguardar até que a instrução termine sua execução.
- No exemplo anterior, assim que a ALU cria o resultado da soma prevista pela instrução **add**, este valor poderia ser passado como entrada para a subtração no ciclo seguinte (**tornar esse valor disponível assim que ele é calculado, e não apenas no estágio WB**).
- A **inserção de hardware extra** para recuperar uma informação referente a uma instrução passada caracteriza a estratégia conhecida como **forwarding** ou **bypassing**.

# Hazards de Dados: Exemplo

- Pergunta:
  - Até chegar ao estágio WB, onde fica armazenado esse o valor produzido no estágio EX ?

# Hazards de Dados: Exemplo

- Pergunta:

- Até chegar ao estágio WB, onde fica armazenado esse o valor produzido no estágios EX ?

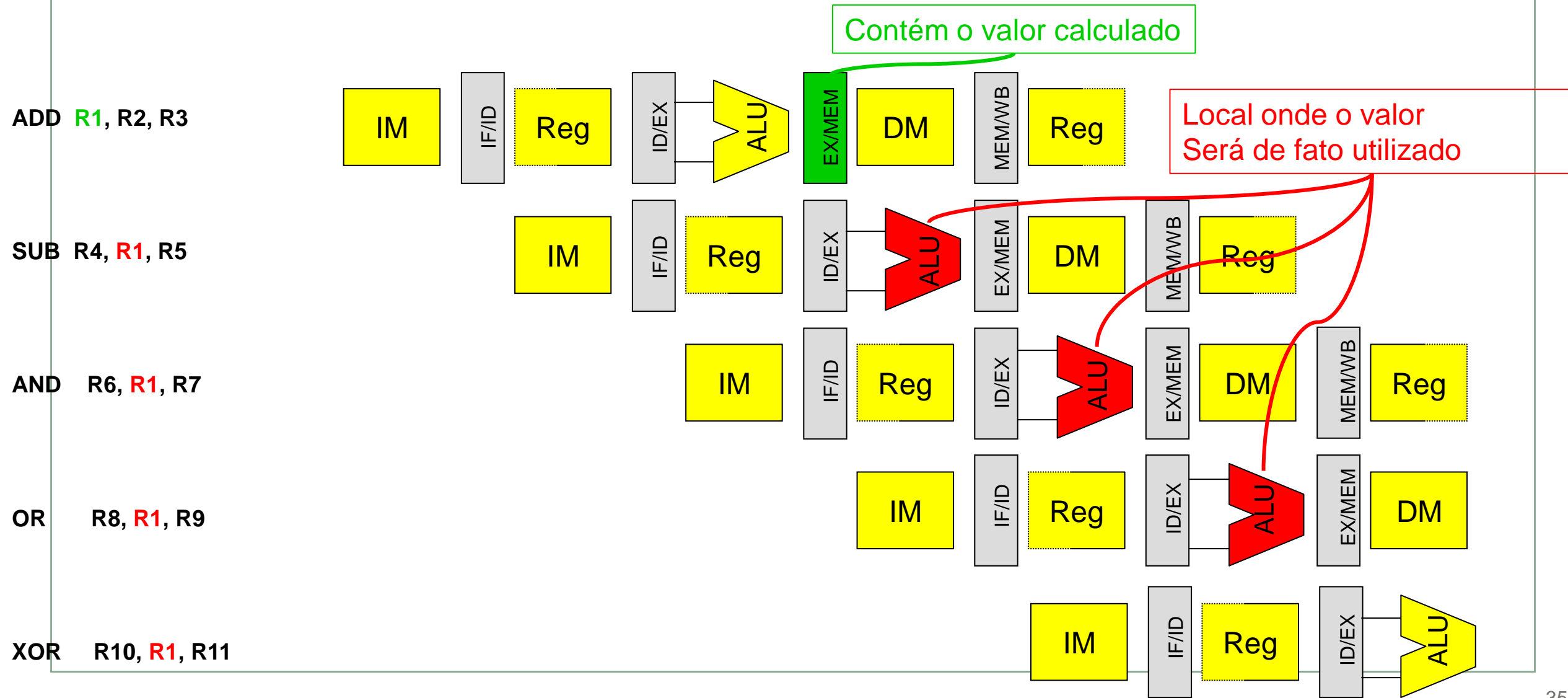
- Resposta → **Nos registradores de pipeline**

- Idéia:

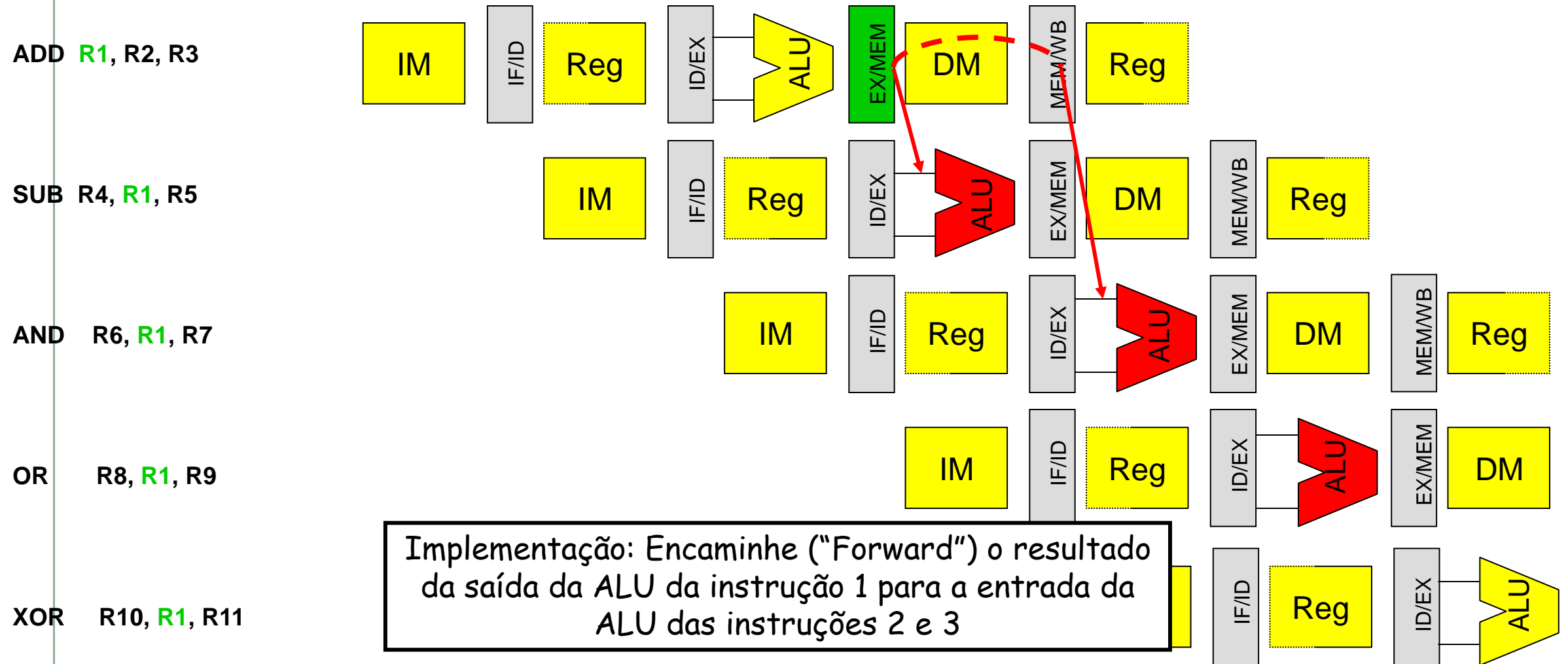
- Permitir que os componentes utilizando o valor produzido consigam acessá-lo diretamente dos registradores de pipeline.



# Hazards de Dados: Exemplo

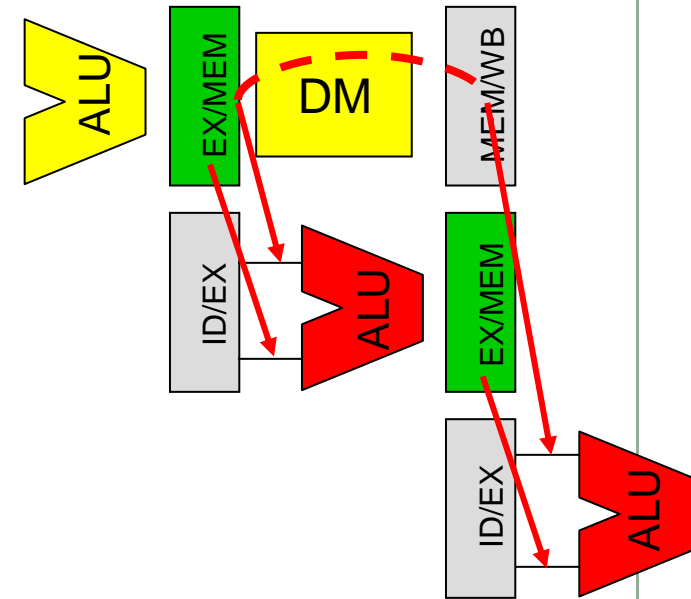


# Hazards de Dados: Exemplo

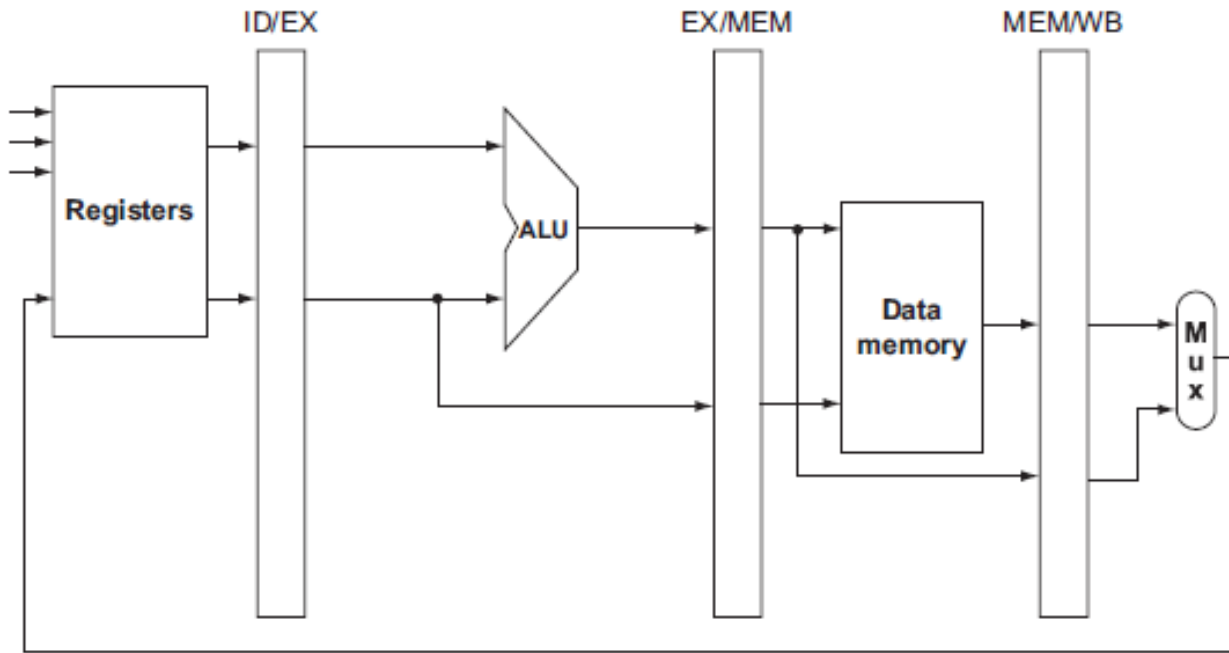


# Hazards de Dados: Exemplo

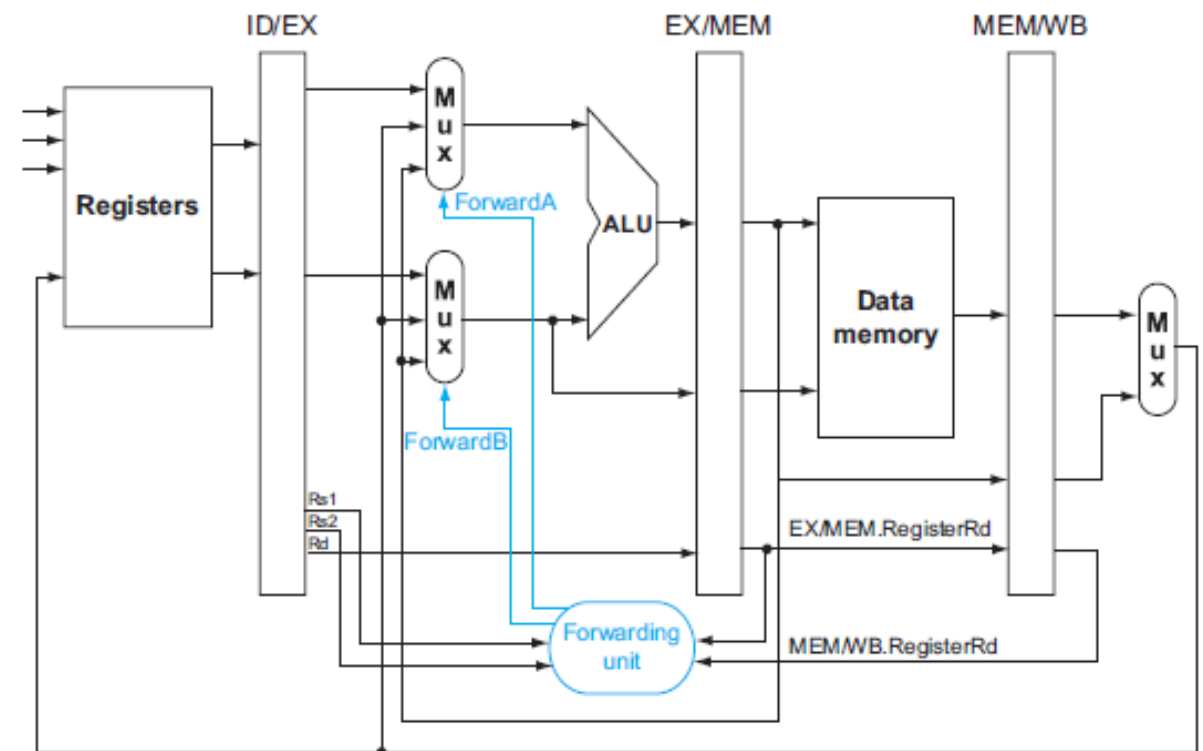
- Como o encaminhamento é implementado ?
  - Os **registradores** de pipeline EX/MEM (saída da ALU), e MEM/WB (saída da memória de dados) são sempre **interconectados** com a entrada da ALU.
  - **Circuitos** lógicos são **acrescentados** para determinar se o valor lido por uma instrução é aquele que se encontra no banco de registradores, ou que está chegando dessa interconexão.



# Unidade de Forwarding



a. No forwarding



b. With forwarding

# Unidade de Forwarding

- Implementação

## EX hazard

if (EX/MEM.RegWrite and (EX/MEM.RegisterRd  $\neq$  0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs1))

ForwardA = 10

if (EX/MEM.RegWrite and (EX/MEM.RegisterRd  $\neq$  0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs2))

ForwardB = 10

## MEM hazard

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd  $\neq$  0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs1))

ForwardA = 01

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd  $\neq$  0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs2))

ForwardB = 01

# Exercício

- Considere as seguintes instruções

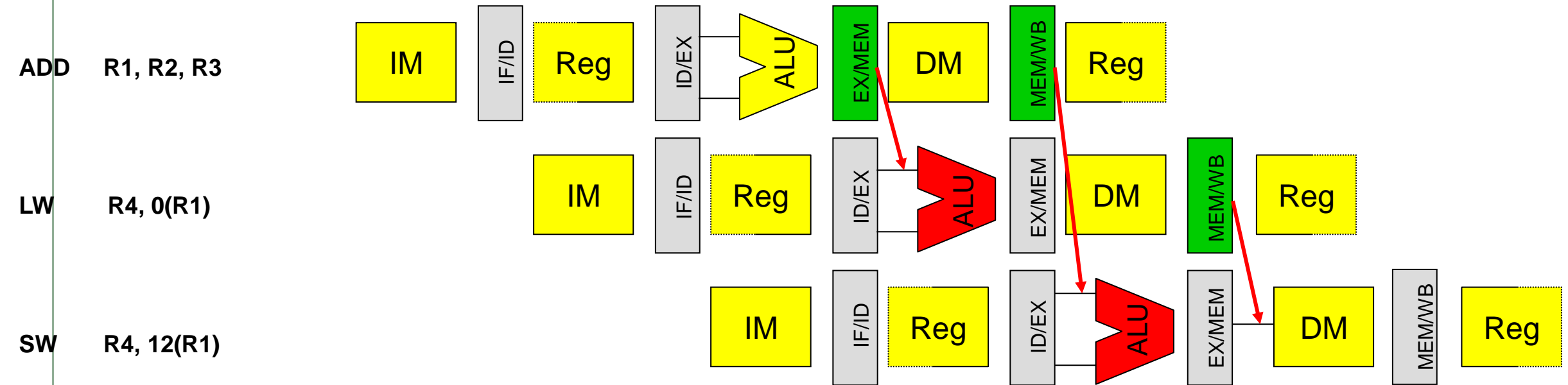
ADD        R1, R2, R3

LW         R4, 0(R1)

SW         R4, 12(R1)

- Desenhe o diagrama com os componentes do pipeline e a execução dessas instruções em sequencia.
- Indique os encaminhamentos necessários para que não sejam necessários stalls durante a execução.

# Resposta



# Forwarding Impossível

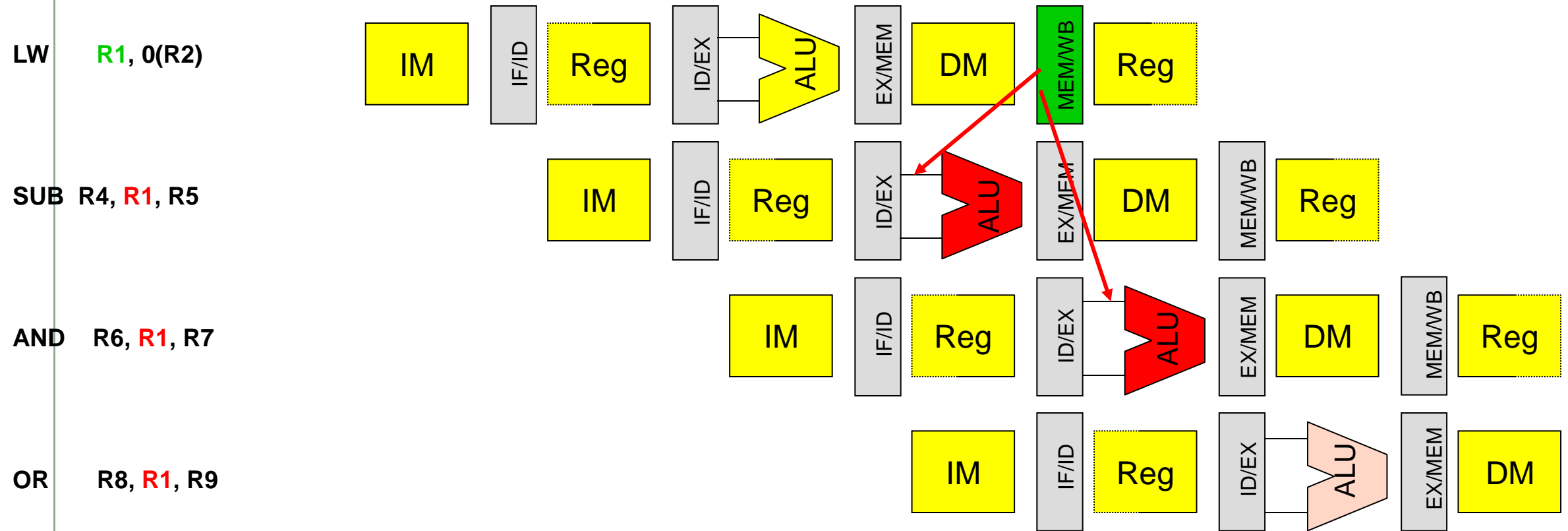
- Considere as seguintes instruções:

LW	R1, 0(R2)
SUB	R4, R1, R5
AND	R6, R1, R7
OR	R8, R1, R9

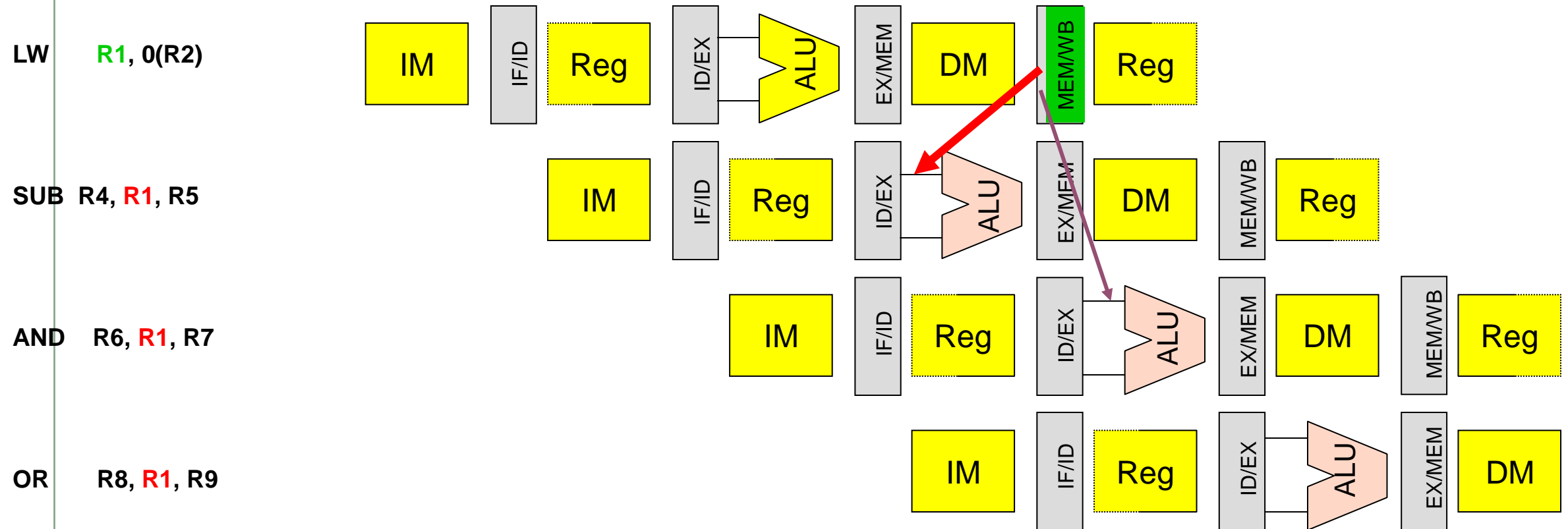
- Todas as instruções utilizam o valor lido da memória pela primeira instrução, no registrador R1



# Forwarding Impossível



# Forwarding Impossível

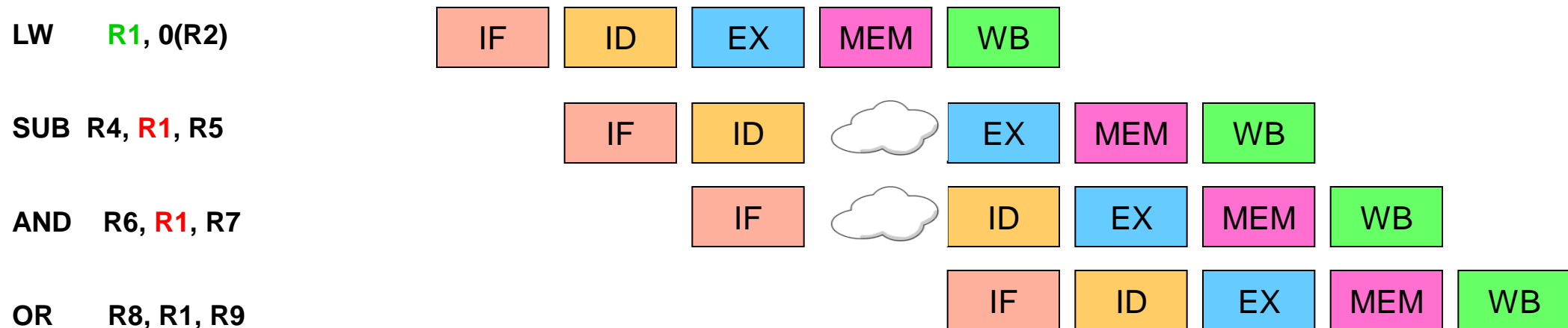


Forward para "Trás"!  
(impossível, o valor ainda não existe)  
Neste caso, é necessário inserir um stall

# Pipeline Interlock

- **Pipeline Interlock** é um esquema de hardware que determina se um ou mais stalls devem ser inseridos após a emissão de uma dada instrução.
- Isso é feito durante o estágio ID, levando-se em conta o número (Id) dos registradores que a instrução está utilizando.

*exemplo do slide anterior:*

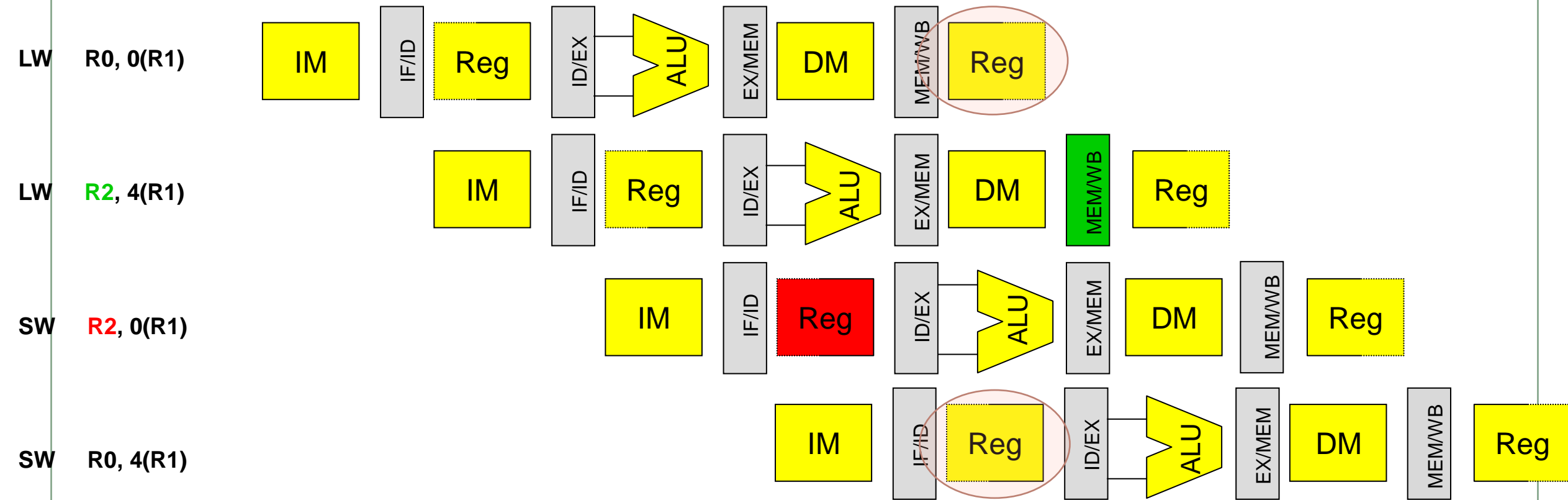


# Pipeline Interlock

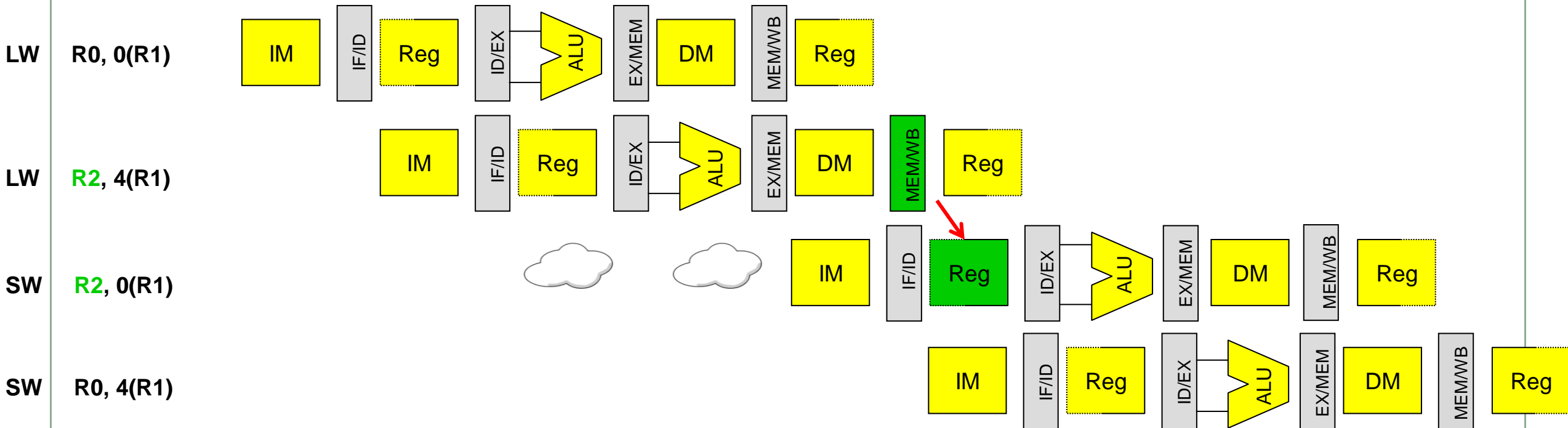
- Exemplo: encontre o conflito existente no código do procedimento swap, abaixo.

	# 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)

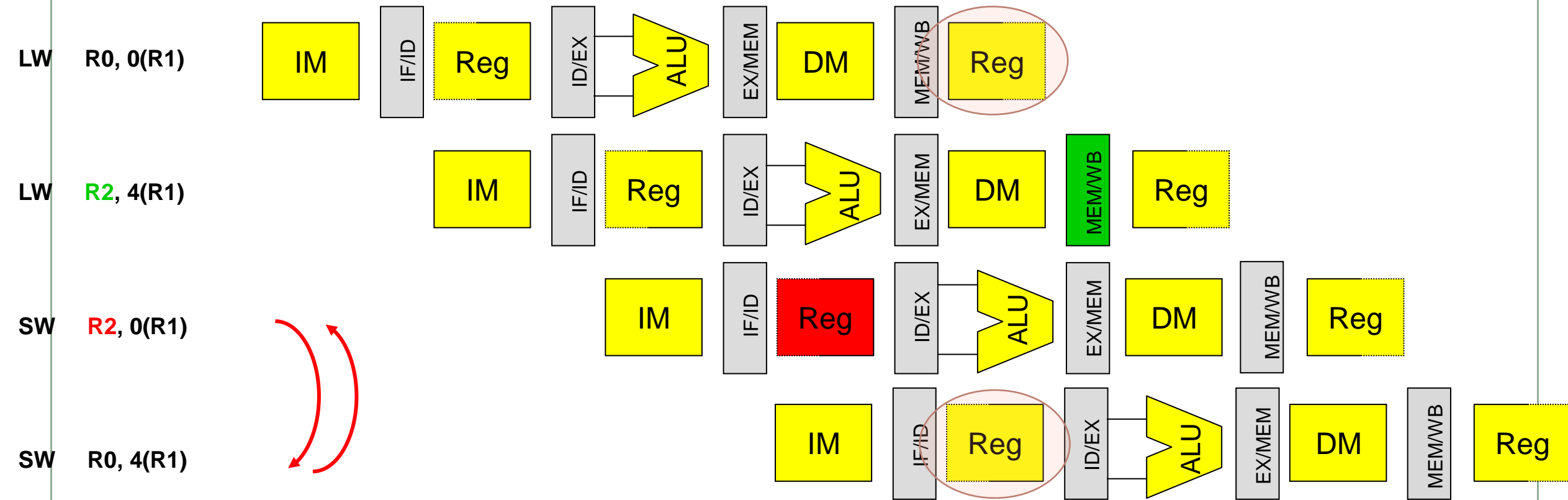
# Pipeline Interlock



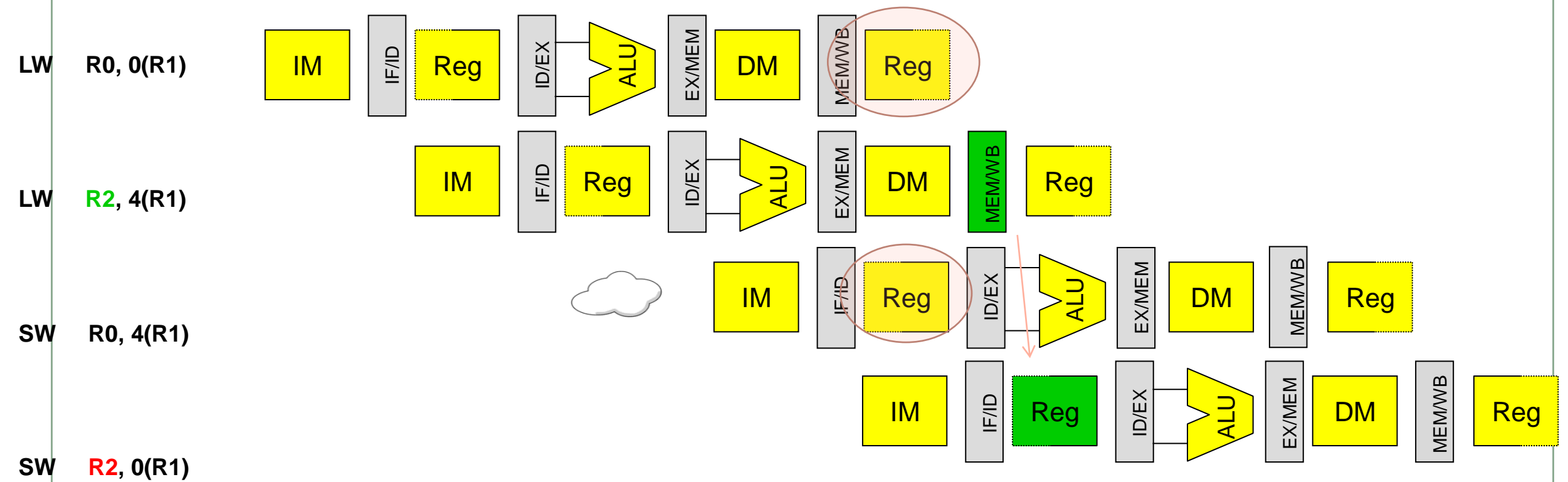
# Pipeline Interlock



# Pipeline Interlock



# Pipeline Interlock





# Hazards de dados e Stalls

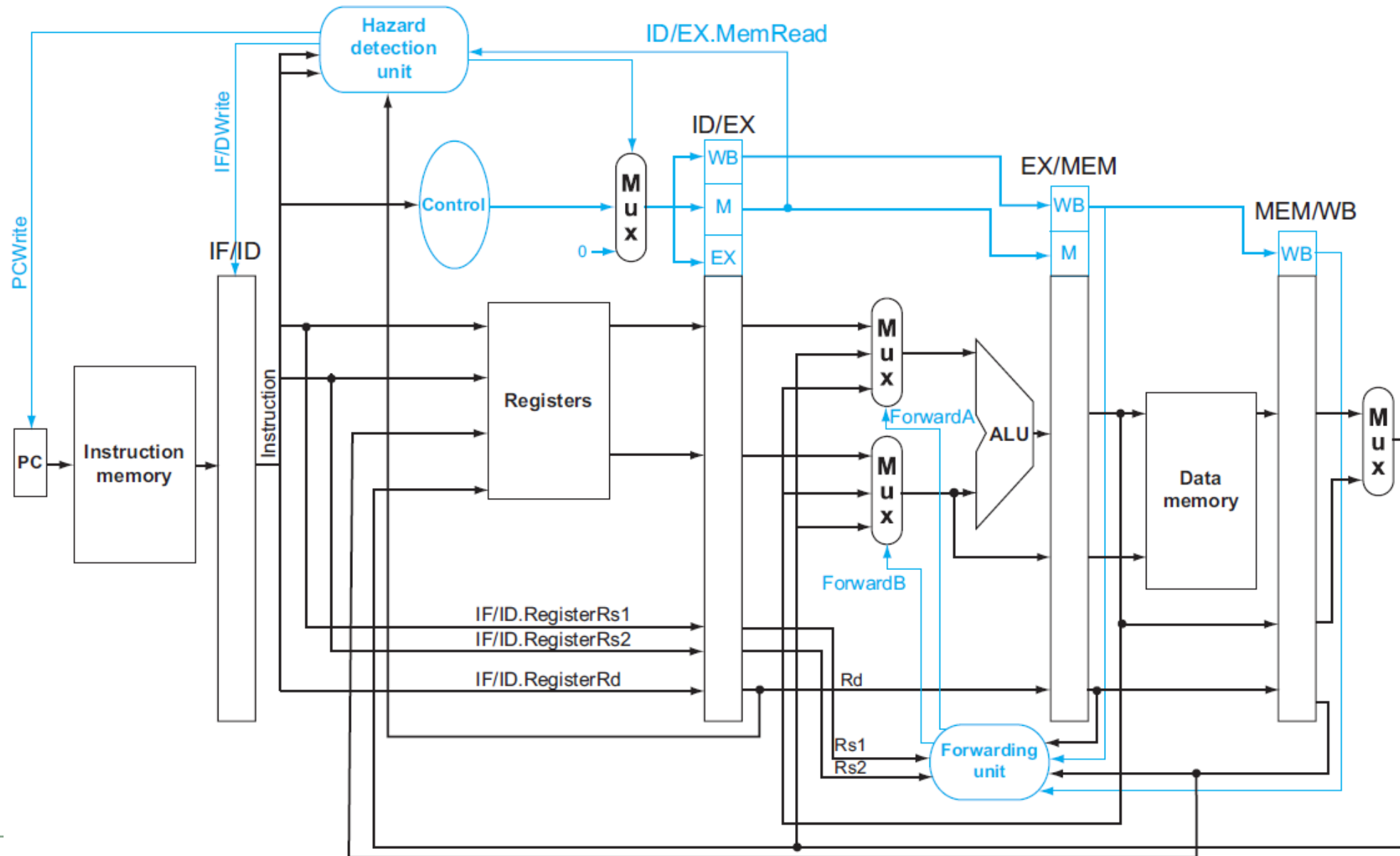
- Caso de stall inevitável por forwarding: após uma instrução load e acesso ao mesmo registrador
- Necessário um detector desse tipo de hazard no estágio ID
- No estágio ID da instrução posterior ao load é possível descobrir se há um load-use hazard. Ou, equivalentemente, quando a instrução load está no 3º estágio (EX).

```
if (ID/EX.MemRead and
    ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or
     (ID/EX.RegisterRd = IF/ID.RegisterRs2)))
    stall the pipeline
```

# Hazards de dados e Stalls

- Stall: implementação
  - Deve-se evitar que o conteúdo dos registradores PC e IF/ID seja modificado.
  - Para isto, basta forçar os sinais de controle contidos no registrador ID/EX para o valor 0.
  - Desta forma, os estágios EX, MEM e WB não terão qualquer efeito sobre a memória, o arquivo de registradores e o valor de PC.
  - Em essência, eles fazem um nop (no-operation).
  - Assim, a instrução em uso (i.e., a que foi lida após o load) será decodificada novamente e a instrução subsequente será buscada mais uma vez.
  - O atraso de 1 ciclo permite que o estágio MEM leia o dado referente à instrução load, o qual poderá ser passado via forwarding para a próxima instrução (que estará no estágio EX).

# Data Hazard and Stalls



# Hazards de Contrôle

# Hazards de Controle

- Surge por causa da necessidade de tomar uma decisão baseada em resultados de uma instrução enquanto outras estão em execução.
- Ou seja, está ligado a instruções do tipo **branch** que controlam o fluxo de execução de um programa:
  - If-then-else
  - Loops (for, while, repeat)
- Problema:
  - A busca da instrução subsequente ao branch é iniciada no próximo ciclo de relógio.
  - Porém, não há como o pipeline saber qual é a instrução correta a ser buscada, uma vez que acabou de receber o próprio branch da memória.

# Hazards de Controle

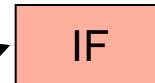
- Quando uma instrução de desvio é executada, ela pode ou não modificar o valor do PC p/ um valor diferente do default (PC+4)
  - Desvio tomado (taken): PC= endereço alvo (label) da instrução de desvio
  - Desvio não tomado (not taken): PC= PC+4
- Em instruções de desvio o PC é atualizado ao final do estágio MEM. Para evitar a inserção de muitos stalls, pode-se atualizar o PC no estágio ID:
  - IF: PC= PC+4
  - ID: Calcular o endereço alvo somando uma constante ao PC
  - ID: Executar o teste da instrução de desvio (ex:  $r1 == r2$ ), para decidir se o desvio será tomado ou não.
  - ID: Caso o desvio seja tomado, atualizar o valor do PC p/ PC= endereço alvo

# Ex.: Desvio Tomado

Taken branch



branch successor



**Instrução errada!!**

branch target



branch target + 1



branch target + 2

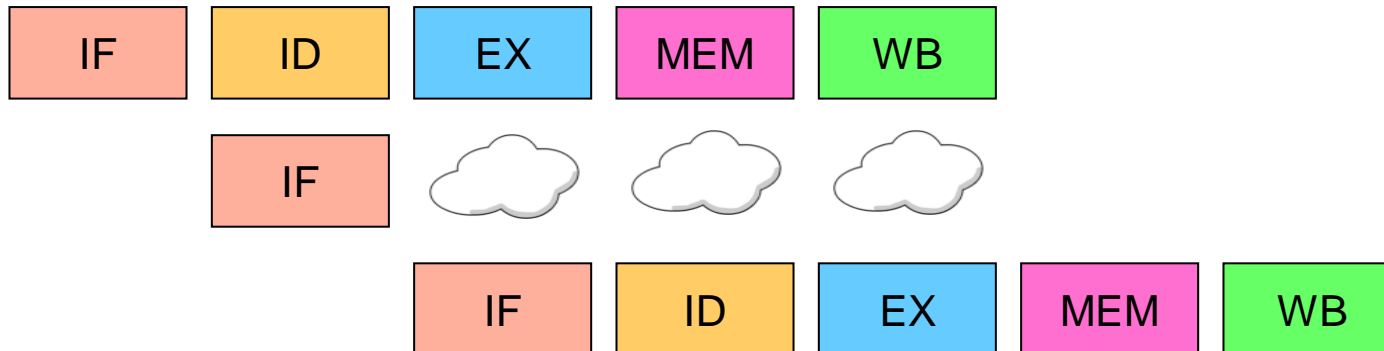


Buscou a instrução errada, logo uma parada (stall) deve ser inserida.

**Isso ocorrerá para todo desvio tomado!**

# Hazards de Controle: Soluções

- Inserir um stall após cada instrução branch
  - Problema: perda de desempenho desnecessário quando o desvio não é tomado

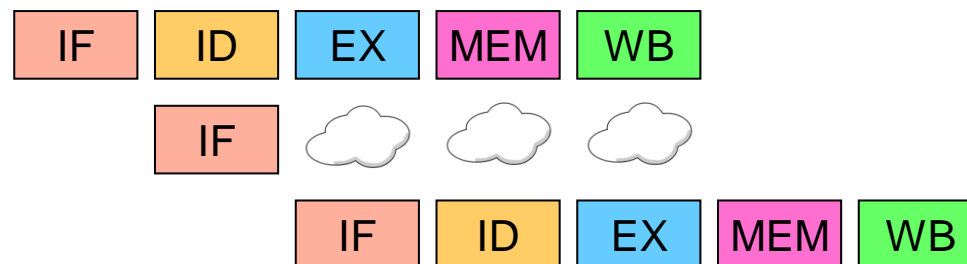




# Hazards de Controle: Soluções

- Prever qual é a próxima instrução: considerar que o desvio nunca será tomado.
- Quando o palpite estiver correto, não haverá atrasos na pipeline.
- Somente quando o desvio for tomado é que algum atraso será introduzido.

Desvio Tomado



Desvio não Tomado

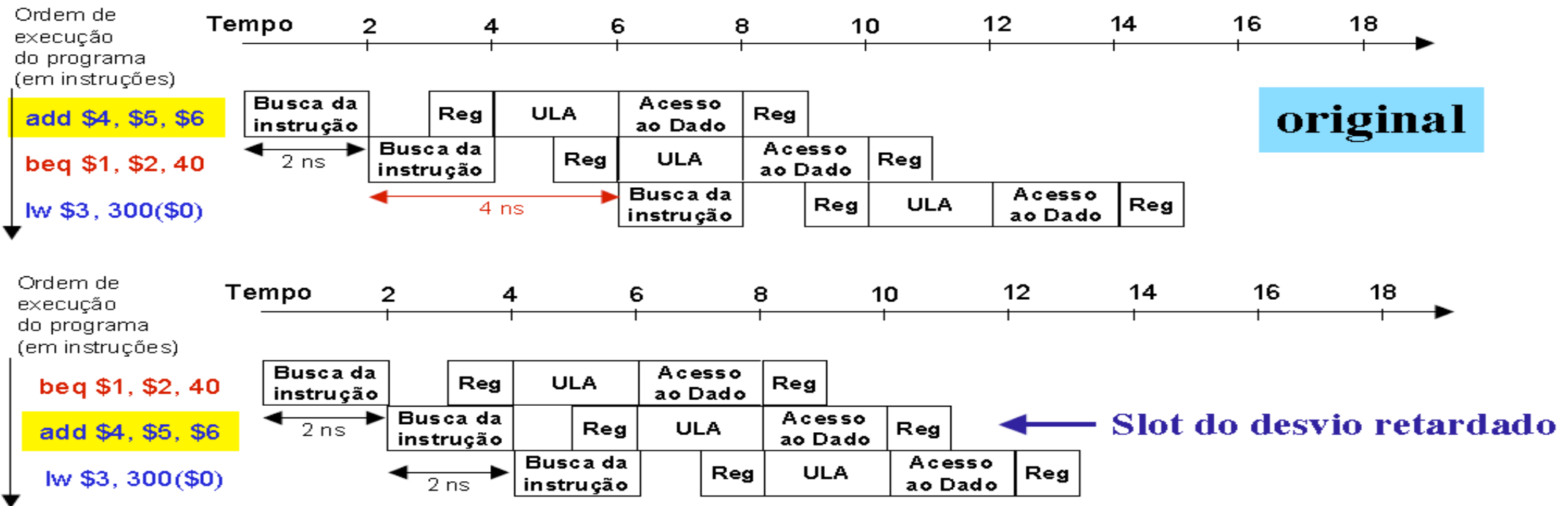


# Hazards de Controle: Soluções

- Desvio Atrasado (Delayed Branch): antecipa ou retarda a execução de uma instrução que seria executada independentemente do desvio
  - Adiciona tempo para a execução do estágio id do desvio
  - Mantém o pipeline preenchido

# Hazards de Controle: Soluções

- Desvio Atrasado (Delayed Branch)



# Hazards de Controle: Soluções

## □ Desvio Atrasado (Delayed Branch)

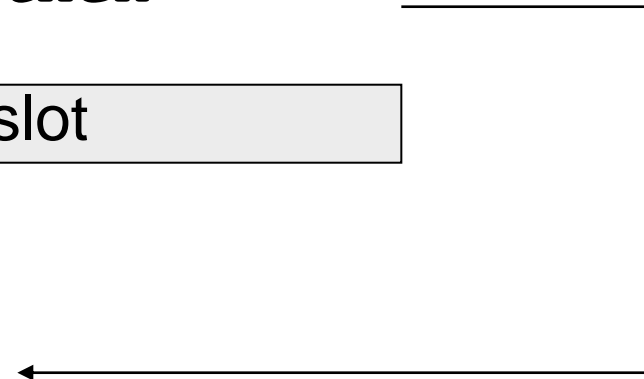
### □ Exemplo1:

**ADD     R1, R2, R3**

**if   R2 == 0   then**

delay slot

. . .

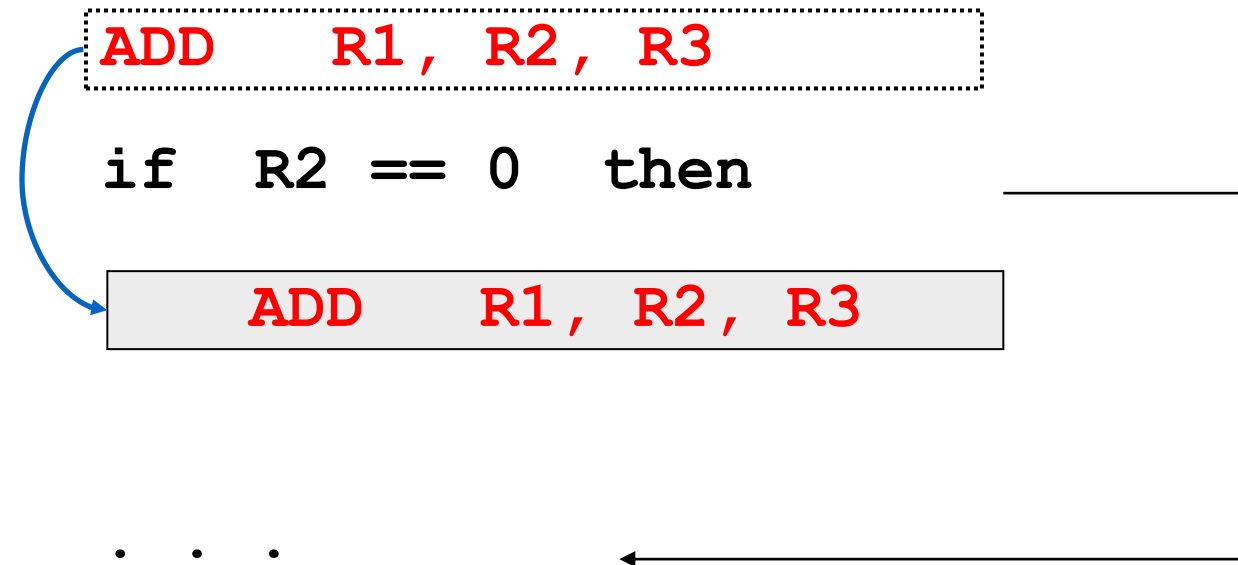


- A instrução *ADD* deve ser executada de qualquer modo, logo pode ser deslocada para após o desvio

# Hazards de Controle: Soluções

- Desvio Atrasado (Delayed Branch)

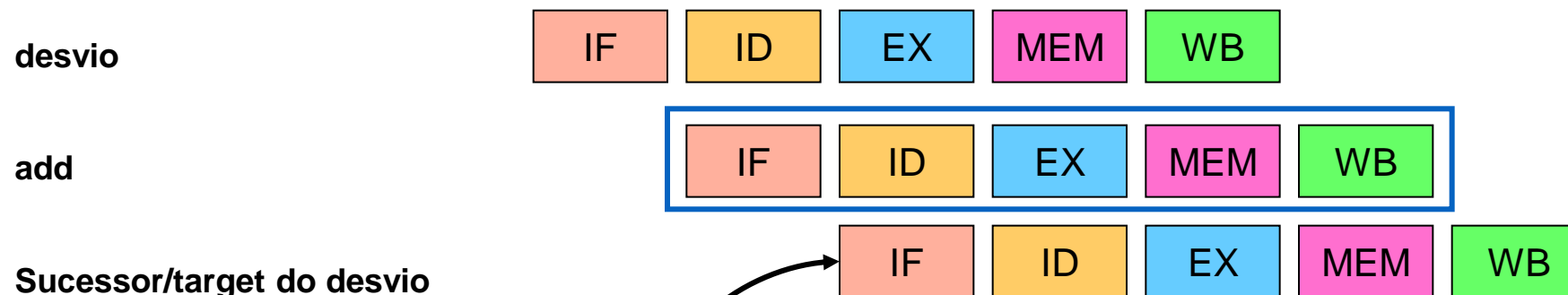
- Exemplo1:



# Hazards de Controle: Soluções

- Desvio Atrasado (Delayed Branch)

- Exemplo1:



Neste ponto, a instrução de desvio já foi executada, sabendo-se exatamente o novo valor do PC.

# Hazards de Controle: Soluções

- Desvio Atrasado (Delayed Branch)

- Exemplo2:

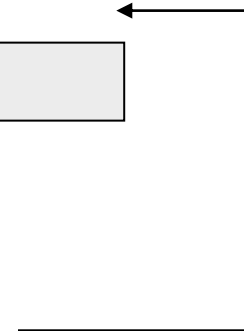
ADD R1, R2, R3

if R1 == 0 then

SUB R4, R5, R6

OR R7, R8, R9

SUB R4, R5, R6



# Resumo

- A técnica de **pipelining** proporciona altos **ganhos desempenho**
  - CPI, clock rate
- Porém, **hazards** podem **comprometer** o seu uso
- **Soluções** de **hardware** e **compilação** são conhecidas e podem resolver o problema em boa parte dos casos.
- Mesmo assim, algumas situações obrigam a inserção de um grande número de paradas:
  - Espera por leitura à memória, quando ocorre um **cache miss** (instruções ou dados).
  - Esta constitui-se a principal causa de perda de desempenho em processadores modernos.
  - O tratamento de **exceções** também é mais complexo e custoso em processadores c/ pipeline, pois o pipeline dever ser "esvaziado" antes de tratar a exceção, e reenchido depois.