



Pontifícia Universidade Católica do Rio Grande do Sul  
Instituto de Informática  
Organização de Computadores - GAPH

## Unidade 4\* - Aulas 1/2:

# “Pipelines”, “Hazards” e “Forwarding”

Profs. Fernando Gehm Moraes e Ney Laert Vilar Calazans

**6 de Junho de 2000**

\* Adaptado de Apresentações de Randy Katz, University of Berkeley

*Gaph*  
Grupo de Apoio ao Projeto de Hardware

# Sumário

---

## ♥ Pipeline

- Introdução
- Pipelines em Computadores
- Arquitetura DLX
- Organização DLX com Pipelines

## ♥ Hazards

- Hazards Estruturais
- Hazards de Dados
- Forwarding



## ♥ Pipeline

### – Introdução

#### » Bibliografia:

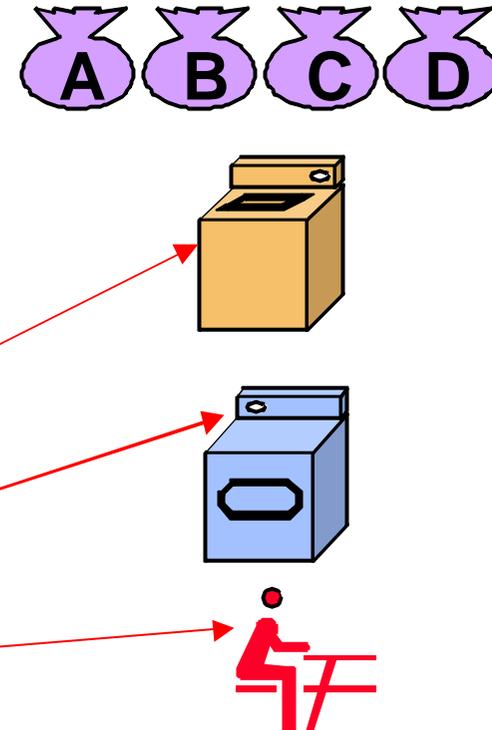
- Hennessy, J. L. & Paterson, D. A. *Computer Architecture: a quantitative approach*. Morgan Kaufmann, Segunda Edição, 1996.
  - Capítulos 3 (Pipelines com estudo de caso - DLX), 2 (Especificação da arquitetura DLX), 4 (Pipeline Avançado).



# Pipeline é Natural!

---

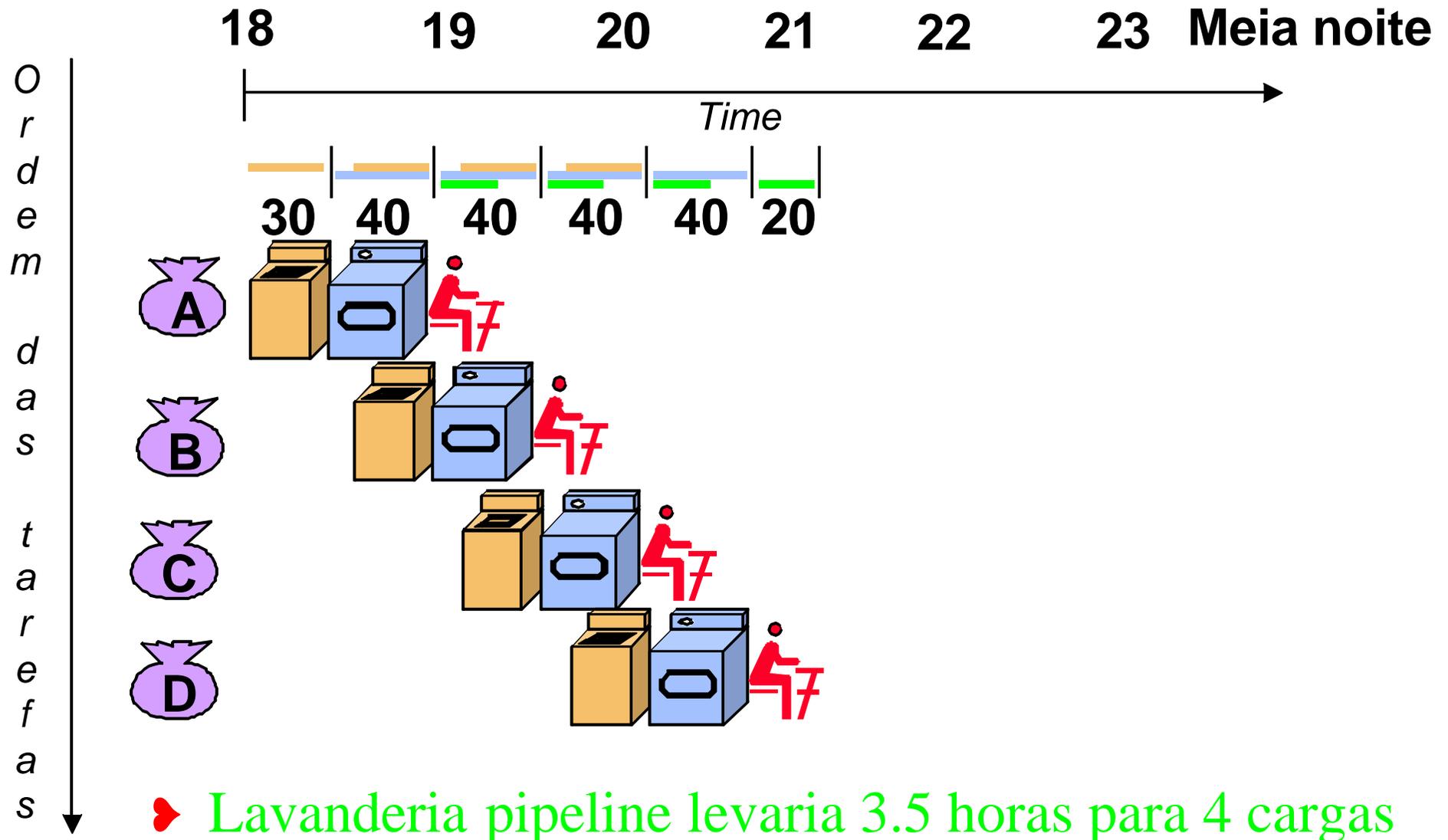
- Exemplo da Lavanderia:
- Ana, Bruno, Cristiane e Daniela têm cada um uma trouxa de roupas para lavar, secar e dobrar;
- Lavagem leva 30 minutos;
- Secagem leva 40 minutos;
- Dobragem leva 20 minutos.







# Lavanderia pipeline



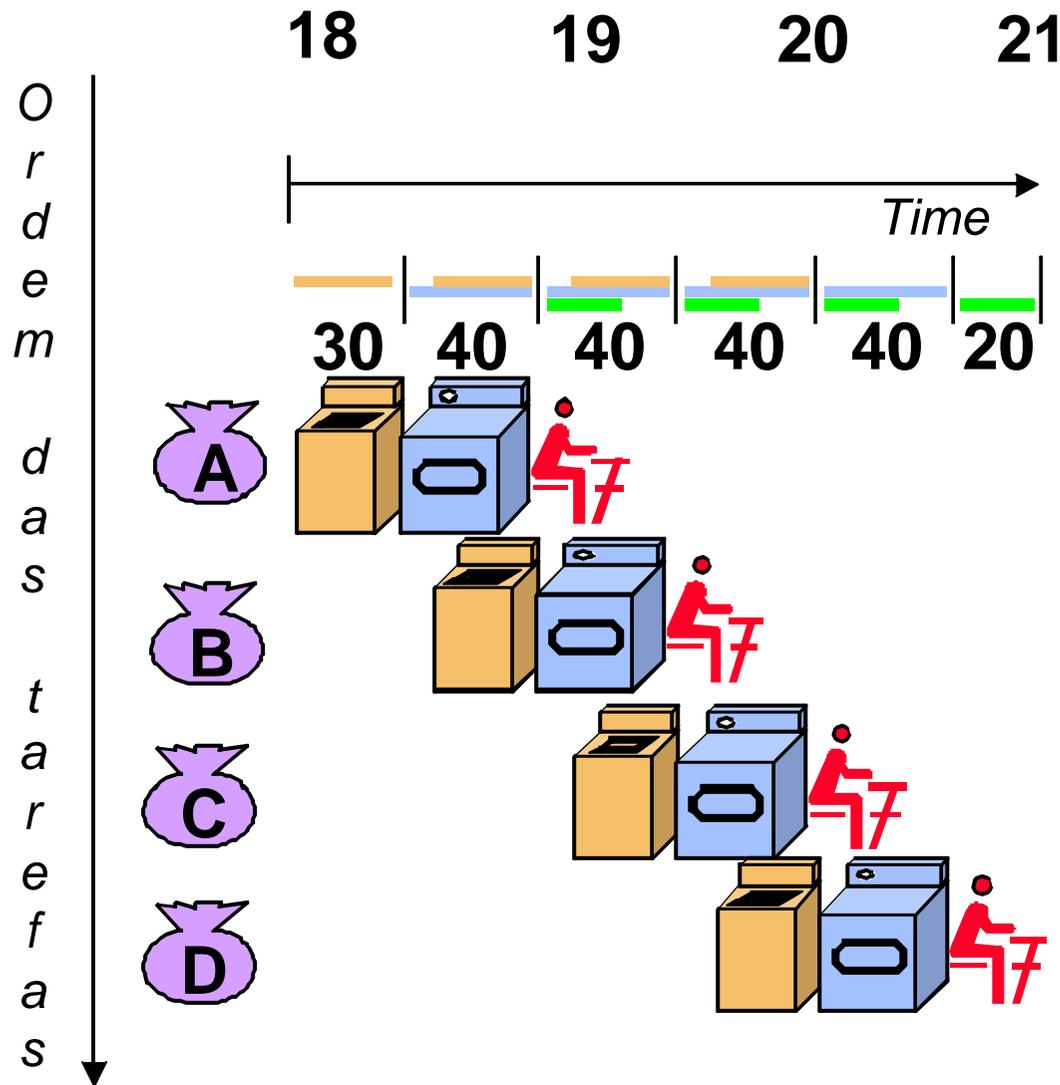
► Lavanderia pipeline levaria 3.5 horas para 4 cargas

# *Definições para Pipelines*

---

- ❖ **Pipeline** = em inglês, tubo, oleoduto - instruções entram numa ponta e são processadas na ordem de entrada;
- ❖ Tubo é dividido em **estágios** ou **segmentos**;
- ❖ Tempo que uma instrução fica no tubo = **latência** ;
- ❖ Número de instruções executadas na unidade de tempo = **desempenho** ou “**throughput**”.
- ❖ Tempo que uma instrução permanece em um estágio = **ciclo de máquina** - normalmente, igual a um ciclo de relógio (excepcionalmente dois);
- ❖ **Balanceamento** - medida da uniformidade do tempo gasto em cada estágio.

# Lições ensinadas por Pipelines



- ▶ Pipeline não reduz a **latência** de uma única tarefa, ajuda no **throughput** de todo o trabalho;
- ▶ A taxa de pipeline é limitada pelo estágio **mais lento**;
- ▶ Tarefas **múltiplas** operam de forma simultânea;
- ▶ Aceleração potencial (speedup) = **Número de estágios do pipe**;
- ▶ Comprimentos desbalanceados de estágios reduz speedup;
- ▶ Tempo para “**preencher**” o pipeline e tempo para “**drená-lo**” reduzem speedup.



# Sumário

---

## ➤ Pipeline

- ✓ Introdução

- Pipelines em Computadores



# Pipelines em Computadores

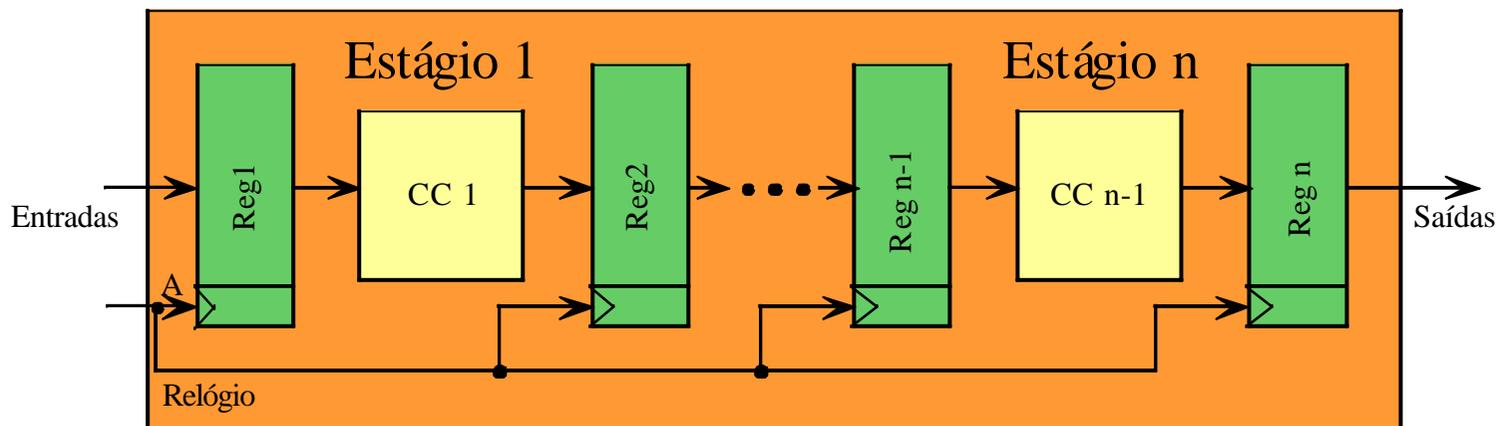
---

- ▶ Técnica de implementação - múltiplas instruções com execução superposta;
- ▶ Chave para criar processadores velozes, hoje;
- ▶ Similar a uma linha de montagem de automóveis:
  - **Linha de montagem**: vários estágios; cada estágio em paralelo com outros, sobre automóveis diferentes;
  - **Pipeline em computadores**: cada estágio completa parte de instrução; como antes, diferentes estágios sobre partes de diferentes instruções; Registradores separam estágios;
  - **Partes de uma instrução**: Busca, busca de operandos, execução.



# Organização Geral Pipeline

- ▶ Alternância de elementos de memória e blocos combinacionais:
  - **memória**: segura dados entre estágios, entre ciclos de relógio;
  - **CCs**: lógica combinacional, processam informação.



# *Pipelines em Computadores*

---

- ♥ Se **estágios** perfeitamente “balanceados”:
  - tempo para terminar de executar instruções com pipeline = tempo por instrução na máquina sem pipeline / número de estágios no pipeline;
- ♥ **Meta do projetista** - balancear estágios;



# *Pipelines - Vantagens e Inconvenientes*

---

## ♥ Vantagens

- reduz **tempo médio de execução** de programas;
- reduz o **CPI (clocks por instrução)** médio;
- reduz duração do **ciclo de clock**;
- acelera processamento sem mudar forma de programação.

## ♥ Inconvenientes

- estágios em geral **não** podem ser totalmente **balanceados**;
- implementação **complexa**, acrescenta **custos** (hardware, tempo);
- para ser implementado, conjunto de instruções deve ser simples.

## ♥ Conclusão

- Pipelines são difíceis de implementar, fáceis de usar.



# Sumário

---

## ➤ Pipeline

- ✓ Introdução
- ✓ Pipelines em Computadores
  - Arquitetura DLX



# *Arquitetura DLX-1*

---

- Microprocessador RISC de 32 bits, load-store
  - 32 registradores de 32 bits de propósito geral (GPRs) - R0-R31;
  - registradores de ponto flutuante (FPRs) visíveis como precisão simples, 32x32 (F0, F1, ..., F31) ou precisão dupla 16x64 (F0, F2, ..., F30);
  - R0 é constante, vale 0;
  - Alguns registradores especiais - Status, FPStatus.
  
- Modos de endereçamento
  - imediato com operando de 16 bits (em hardware);
  - base-deslocamento com endereço de 16 bits (em hardware);
  - a registrador (base deslocamento com deslocamento 0);
  - absoluto (direto) com operando de 16 bits (base-deslocamento R0 é base).

# *Arquitetura DLX-2*

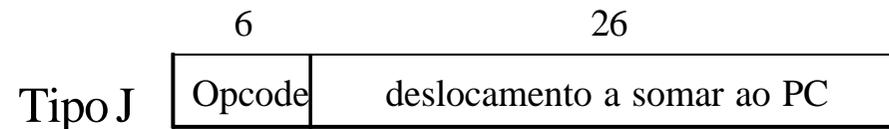
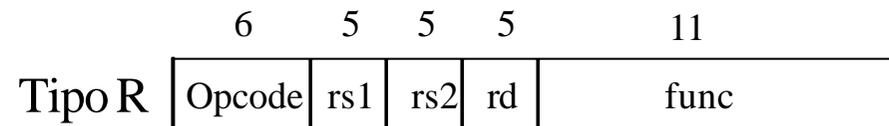
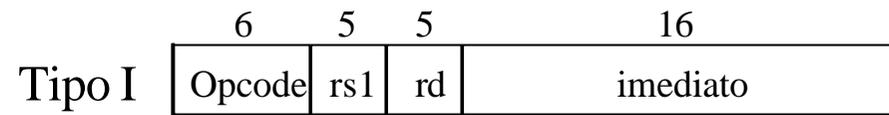
---

- Barramento de dados e endereços de 32 bits;
- Portanto, cada leitura da memória traz para dentro do processador 32 bits:
  - 4 bytes;
  - 2 meia-palavras;
  - 1 palavra;
- Memória endereçável a byte, modo Big Endian
  - Big Endian - dados de mais de um byte são guardados em posições de memória a partir do byte mais significativo (SPARC, PPC, etc.);
  - Little Endian - dados de mais de um byte são guardados em posições de memória a partir do byte menos significativo (Intel);
  - acesso a byte, meia-palavra (16 bits) ou palavra (32bits).

# Arquitetura DLX-3

## Formatos de Instrução

- Tipo I:
  - » Loads, stores, de bytes, meia-palavra e palavra, todos os imediatos, saltos condicionais (rs1 é registrador, rd não usado), salto incondicional a registrador;
- Tipo R:
  - » operações com a ULA e registradores, func diz a operação, operações com registradores especiais;
- Tipo J:
  - » salto incondicional, exceções e retornos de exceção.



# Sumário

---

## ➤ Pipeline

- ✓ Introdução
- ✓ Pipelines em Computadores
- ✓ Arquitetura DLX
- Organização DLX com Pipelines





# Ciclos de Máquina do DLX - 1 de 2

---

## ♥ 1- Ciclo de Busca de Instrução (IF):

- $IR \leftarrow Mem(PC); NPC \leftarrow PC+4;$

## ♥ 2 - Ciclo de decodificação de instrução/busca de registrador (ID)

- $A \leftarrow Regs(IR[6:10]); B \leftarrow Regs(IR[11:15]); Imm \leftarrow (IR[16])^{16}##IR[16:31];$
- A, B, Imm são regs temporários; operação sobre Imm é Extensão de sinal.

## ♥ 3 - Ciclo de execução e cálculo de endereço efetivo (EX)

- Referência à memória:  $ALUoutput \leftarrow A + Imm;$
- Instrução Reg-Reg/ALU:  $ALUoutput \leftarrow A \text{ op } B;$
- Instrução Reg-Imm/ALU:  $ALUoutput \leftarrow A \text{ op } Imm;$
- Desvioscondicionais:  $ALUoutput \leftarrow NPC + Imm; Cond \leftarrow (A \text{ op } 0);$ 
  - op no último tipo é um operador relacional, tal como  $<, >, ==,$  etc.





## *Ciclos de Máquina do DLX - 2 de 2*

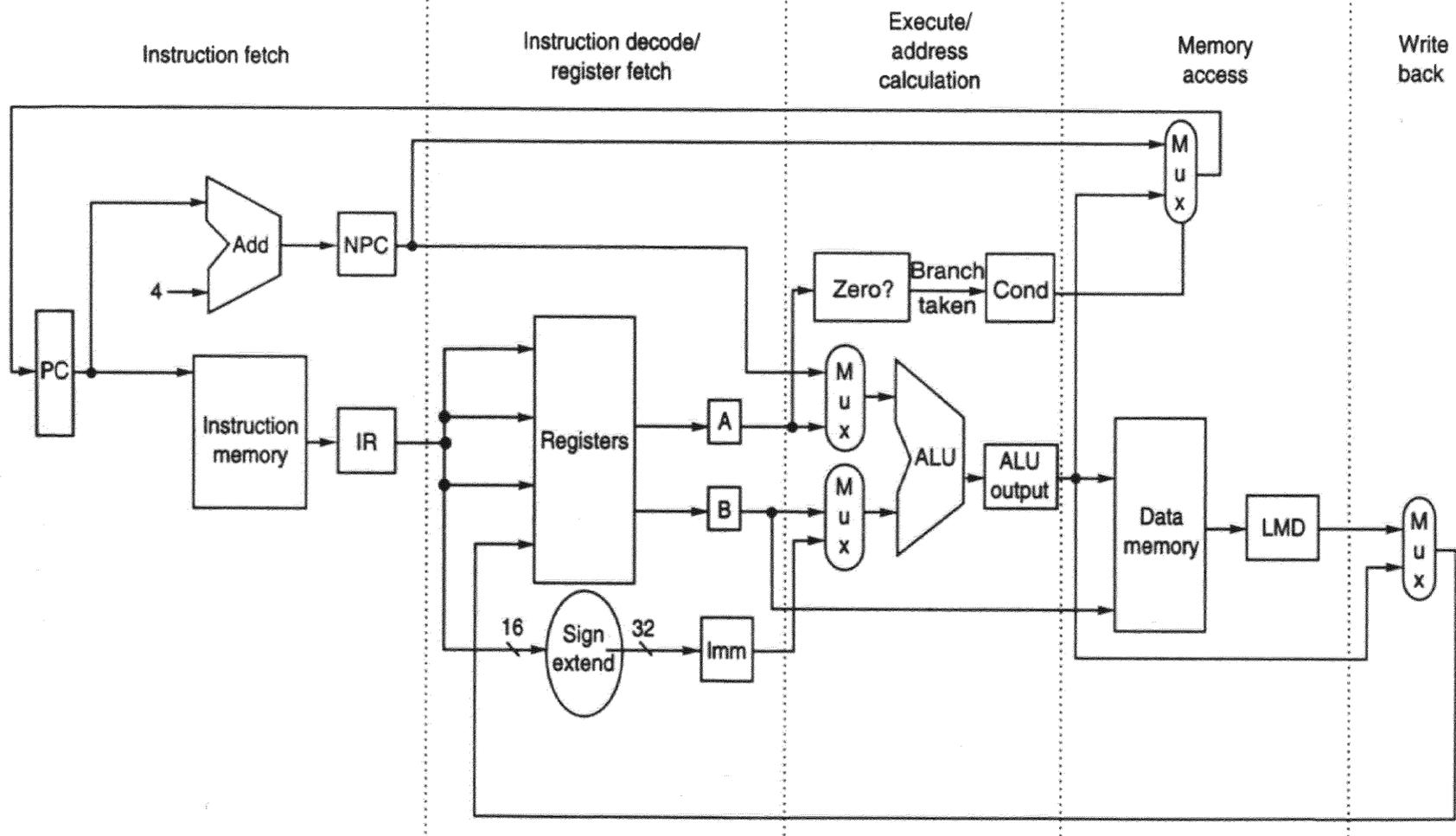
---

- ▶ 4 - Ciclo de acesso à memória/término de desvio condicional (MEM)
  - Referência à memória: `LMD <-- Mem[ALUoutput] OU Mem[ALUoutput] <-- B;`
  - DesvioCondicional: `if (cond) PC <-- ALUoutput else PC <-- NPC;`
- ▶ 5 - Ciclo de atualização ou **write-back** (WB)
  - Instrução Reg-Reg/ALU: `Regs(IR[16:20]) <-- ALUoutput;`
  - Instrução Reg-Imm/ALU: `Regs(IR[11:15]) <-- ALUoutput;`
  - Instrução Load: `Regs(IR[11:15]) <-- LMD;`
- ▶ Próxima página ilustra a implementação sem pipeline.



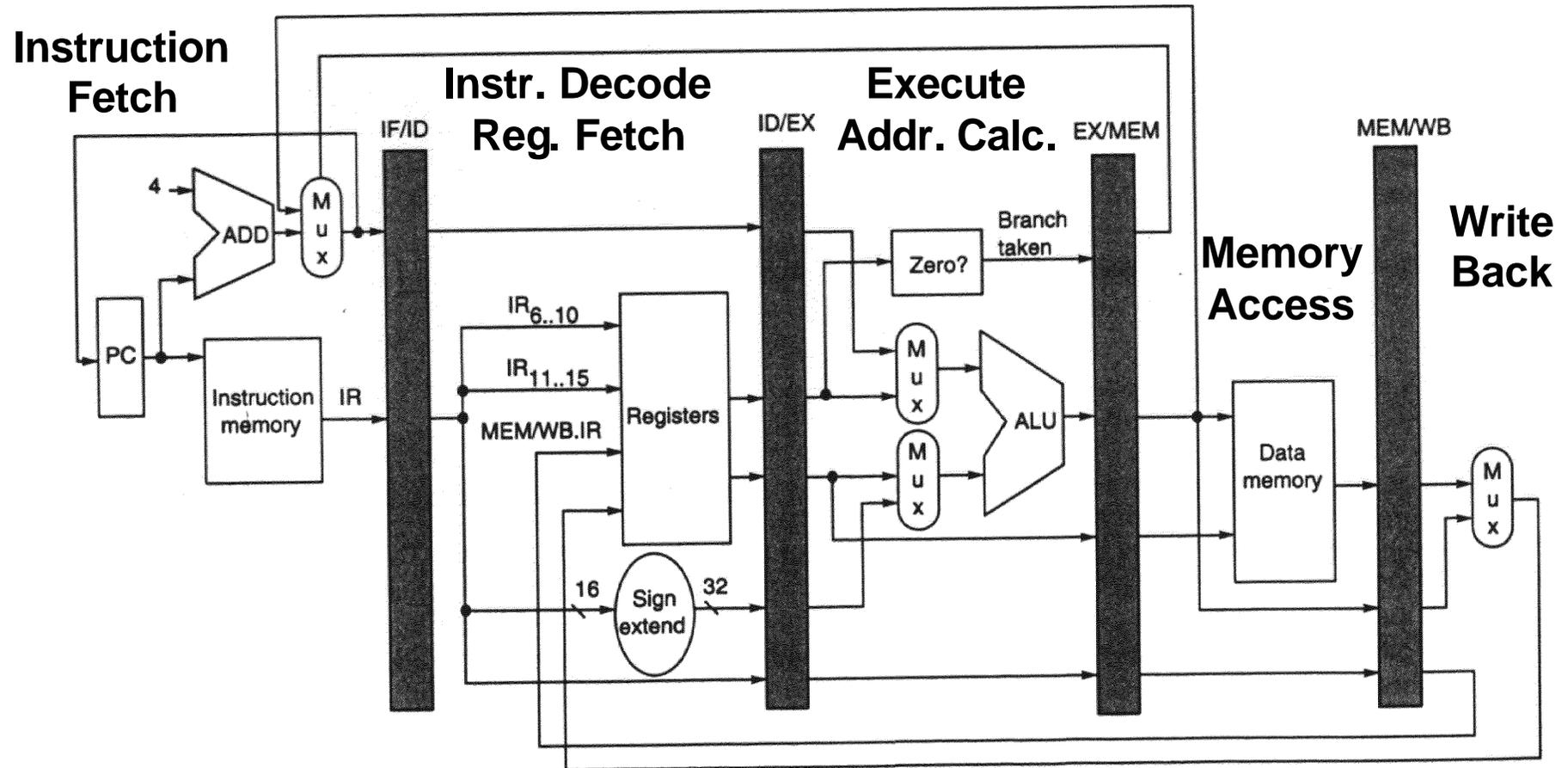


# Um Bloco de Dados p/ o DLX *Fig 3.1, Página 130*





# Bloco de Dados DLX com Pipeline *Fig 3.4, Página 134*



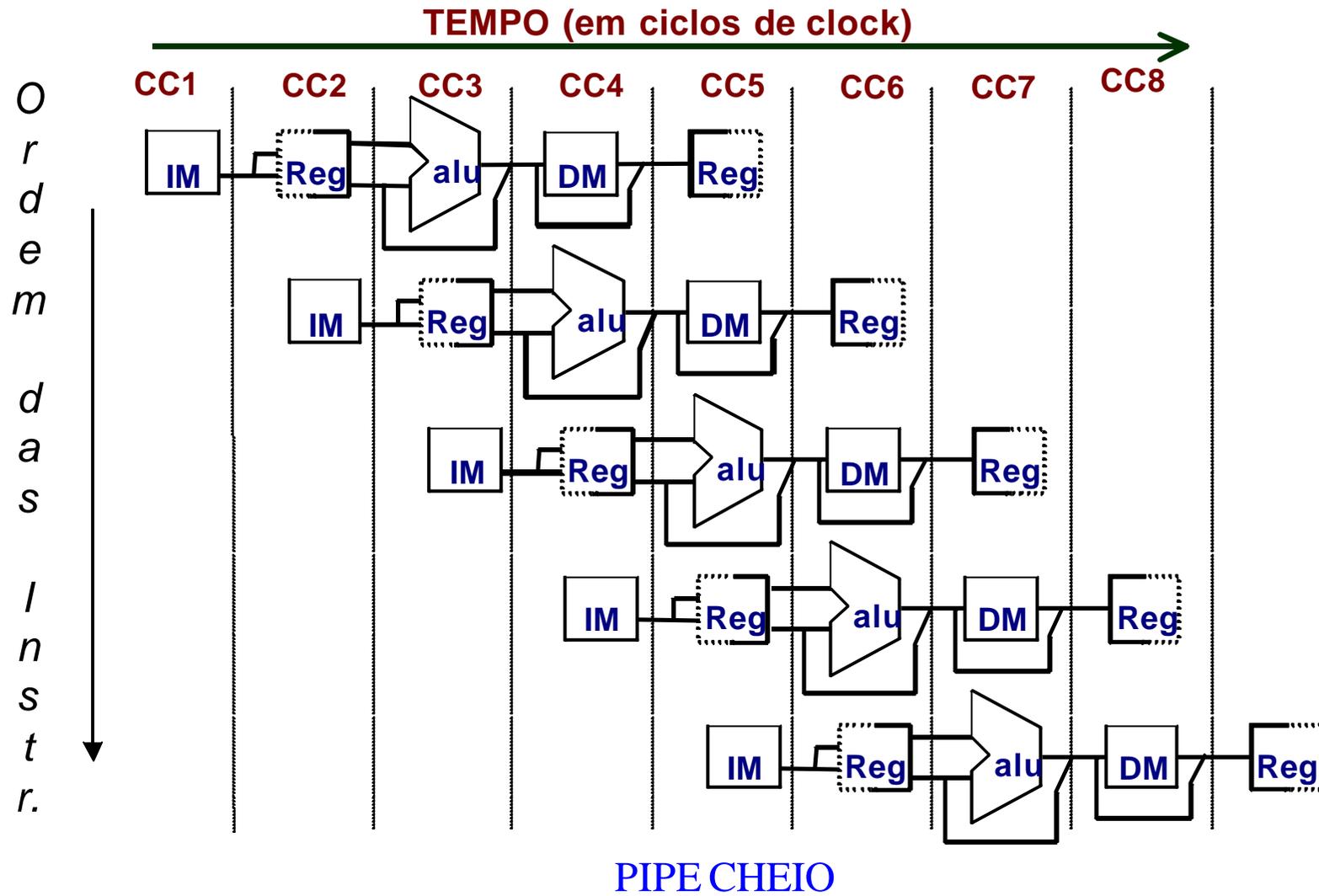
- **Controle de Dados Estacionário**

- decodificação local para cada fase da instrução ou estágio do pipeline





# Pipelines ao longo do Tempo Fig 3.3, Página 133





## *Pipeline em Computadores é Complicado!*

---

- ❖ Limitações de pipelines: **Hazards** (perigos) evitam que uma próxima instrução execute durante um determinado ciclo de clock:
  - **Hazard estrutural** : HW não pode dar suporte a uma determinada combinação de instruções;
  - **Hazard de dados**: Instrução depende do resultado de uma instrução anterior anda no pipeline;
  - **Hazard de controle**: Pipeline de saltos e outras instruções que mudam o PC.
- ❖ Solução comum é suspender (**stall**) o pipeline até que o hazard “**bolhas**” temporais no pipeline (tratado a seguir).



# Sumário

---

## ✓ Pipeline

- ✓ Introdução

- ✓ Pipelines em Computadores

- ✓ Arquitetura DLX

- ✓ Organização DLX com Pipelines

## ♥ Hazards

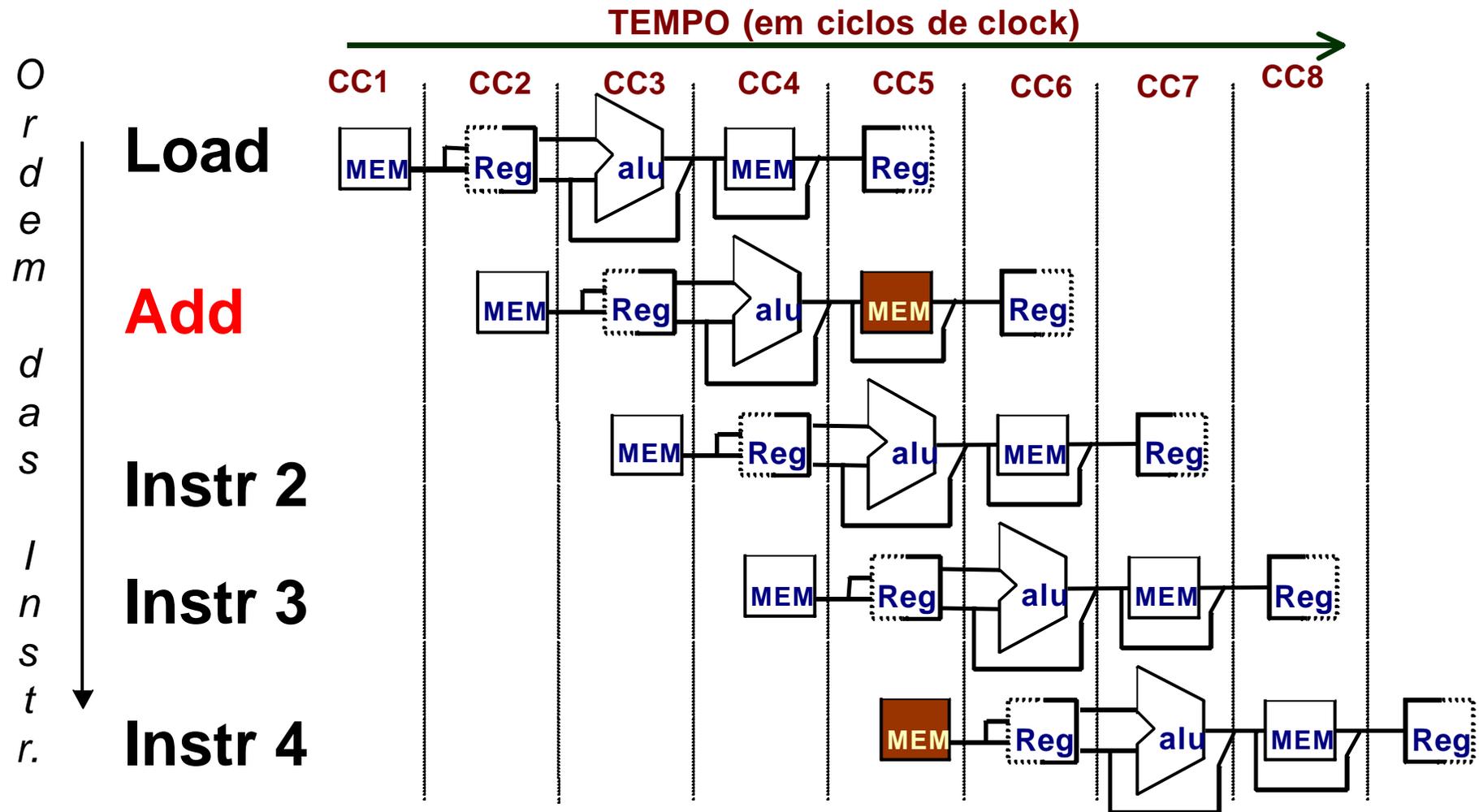
- Hazards Estruturais





# Hazard Estrutural / Memórias de uma porta

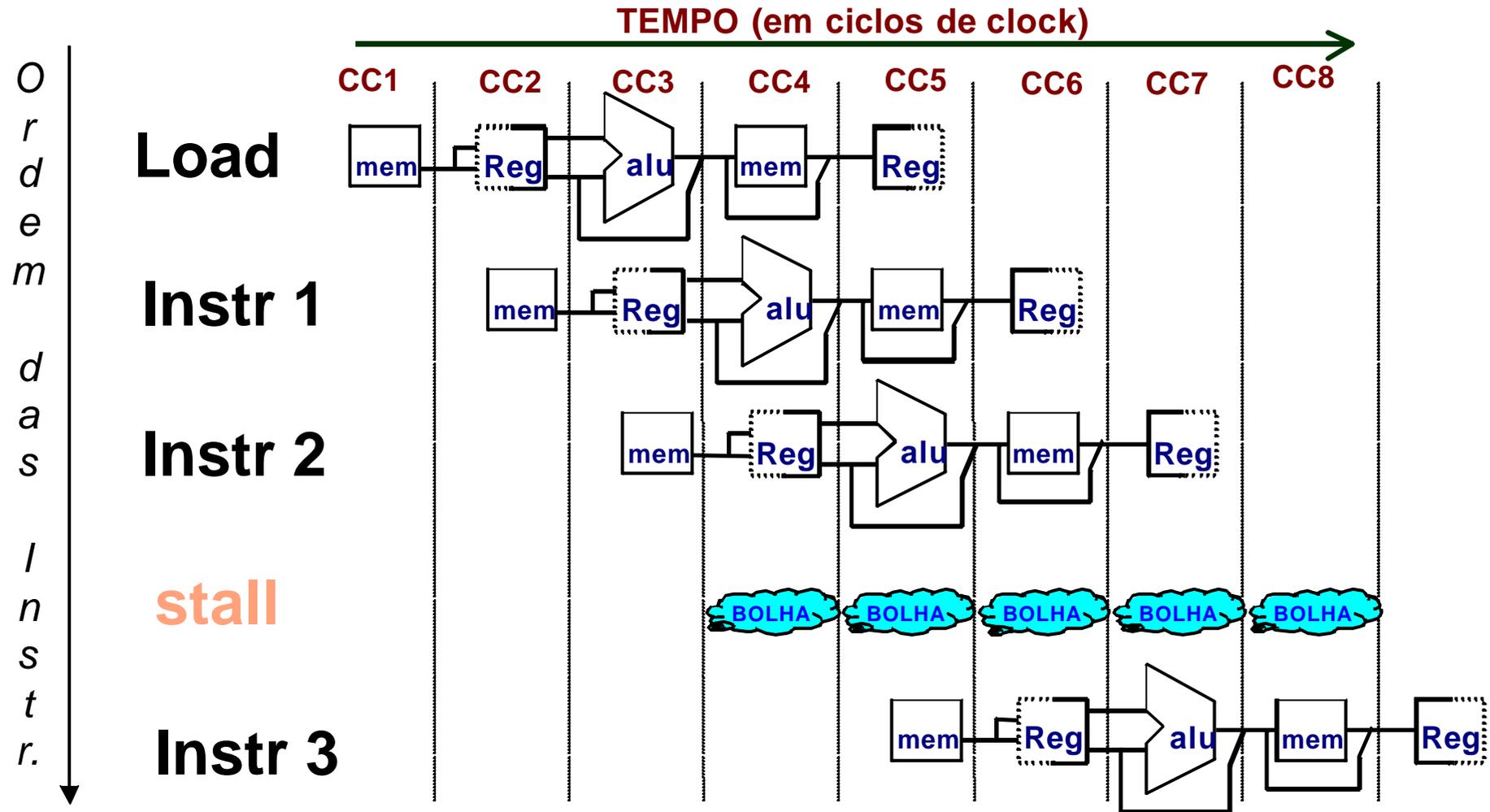
Fig 3.6, Página 142





# Hazard Estrutural / Memórias de uma porta

Fig 3.7, Página 143





## *Equação de Speed Up para Pipeline*

---

$$\begin{aligned}\text{Speedup from pipelining} &= \frac{\text{Ave Instr Time unpipelined}}{\text{Ave Instr Time pipelined}} \\ &= \frac{\text{CPI}_{\text{unpipelined}} \times \text{Clock Cycle}_{\text{unpipelined}}}{\text{CPI}_{\text{pipelined}} \times \text{Clock Cycle}_{\text{pipelined}}} \\ &= \frac{\text{CPI}_{\text{unpipelined}}}{\text{CPI}_{\text{pipelined}}} \times \frac{\text{Clock Cycle}_{\text{unpipelined}}}{\text{Clock Cycle}_{\text{pipelined}}}\end{aligned}$$

$$\text{Ideal CPI} = \text{CPI}_{\text{unpipelined}} / \text{Pipeline depth}$$

$$\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{CPI}_{\text{pipelined}}} \times \frac{\text{Clock Cycle}_{\text{unpipelined}}}{\text{Clock Cycle}_{\text{pipelined}}}$$





## *Equação de Speed Up para Pipeline*

---

$\text{CPI}_{\text{pipelined}} = \text{Ideal CPI} + \text{Pipeline stall clock cycles per instr}$

$$\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle}_{\text{unpipelined}}}{\text{Clock Cycle}_{\text{pipelined}}}$$

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle}_{\text{unpipelined}}}{\text{Clock Cycle}_{\text{pipelined}}}$$

## *Exemplo: duas portas vs. uma porta*

---

- Máquina A: Memória de duas portas;
- Máquina B: Memória de uma porta , mas com implementação pipeline possui um clock 1.05 vezes mais rápido (5%);
- CPI ideal = 1 para ambos;
- Loads são 40% das instruções executadas;

$$\begin{aligned}\text{SpeedUp}_A &= \text{Pipeline Depth} / (1 + 0) \times (\text{clock}_{\text{unpipe}} / \text{clock}_{\text{pipe}}) \\ &= \text{Pipeline Depth}\end{aligned}$$

$$\begin{aligned}\text{SpeedUp}_B &= \text{Pipeline Depth} / (1 + 0.4 \times 1) \\ &\quad \times (\text{clock}_{\text{unpipe}} / (\text{clock}_{\text{unpipe}} / 1.05)) \\ &= (\text{Pipeline Depth} / 1.4) \times 1.05 \\ &= 0.75 \times \text{Pipeline Depth}\end{aligned}$$

$$\begin{aligned}\text{SpeedUp}_A / \text{SpeedUp}_B &= \text{Pipeline Depth} / (0.75 \times \text{Pipeline Depth}) \\ &= 1.33\end{aligned}$$

- Máquina A é 1.33 vezes mais rápida (33%).

# Sumário

---

## ✓ Pipeline

- ✓ Introdução

- ✓ Pipelines em Computadores

- ✓ Arquitetura DLX

- ✓ Organização DLX com Pipelines

## ♥ Hazards

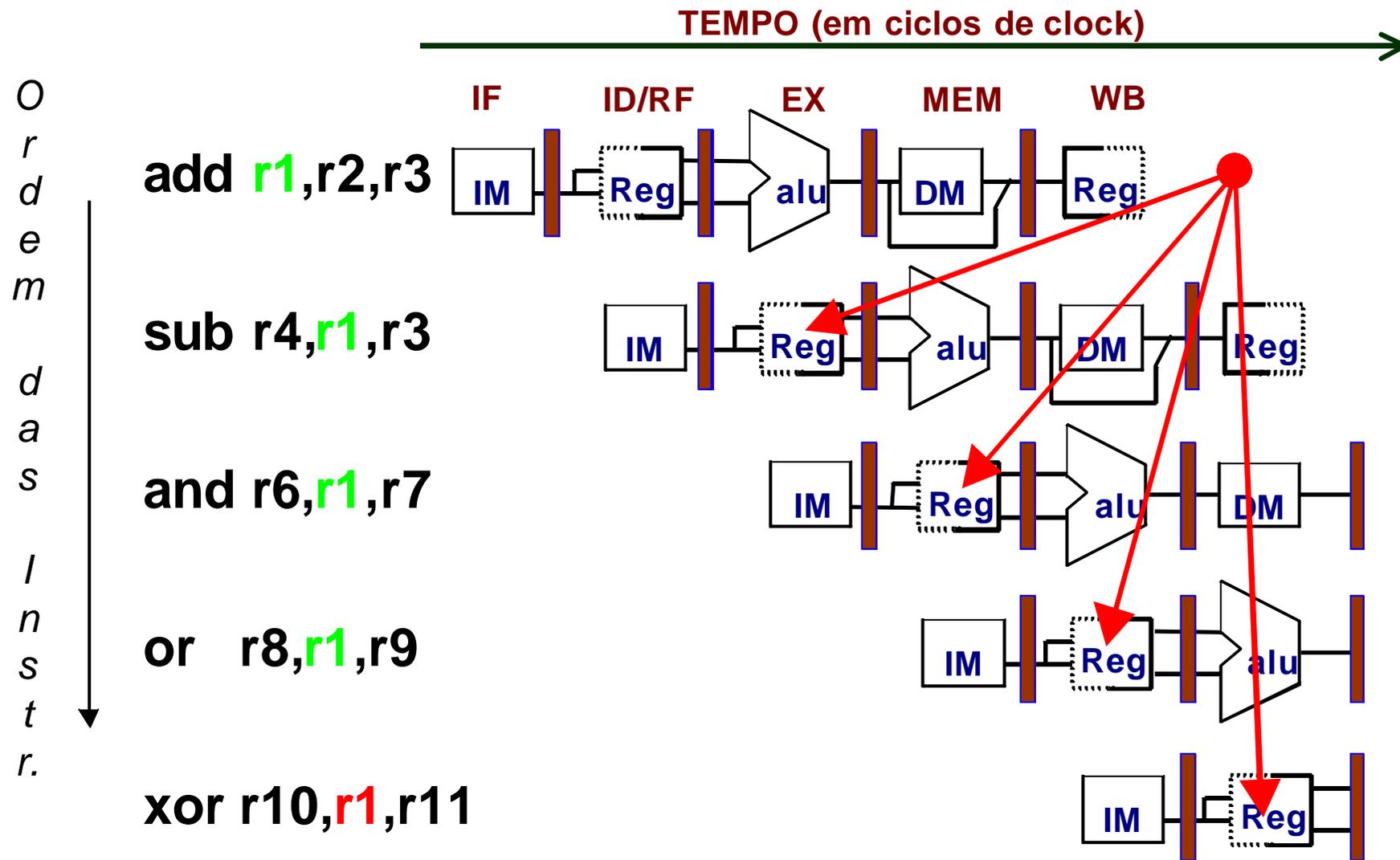
- ✓ Hazards Estruturais

- Hazards de Dados





# Hazard de Dados em R1 Fig 3.9, Página 147



# Três Hazards de Dados Genéricos

Instr<sub>I</sub> seguida pela Instr<sub>J</sub>

- ♥ **Leitura Após Escrita (RAW)**

Instr<sub>J</sub> tenta ler operando antes que a Instr<sub>I</sub> escreva ele;

# *Três Hazards de Dados Genéricos*

---

Instr<sub>I</sub> seguida pela Instr<sub>J</sub>

➤ **Escrita Após Leitura (WAR)**

Instr<sub>J</sub> tenta escrever operando antes que a Instr<sub>I</sub> leia ele

➤ Não pode acontecer no pipeline do DLX porque:

- Todas as instruções levam 5 estágios,
- Leituras são sempre no estágio 2, e
- Escritas são sempre no estágio 5

# Três Hazards de Dados Genéricos

Instr<sub>I</sub> seguida pela Instr<sub>J</sub>

## ▶ Escrita Após Escrita (WAW)

Instr<sub>J</sub> tenta escrever operando antes que a Instr<sub>I</sub> o escreva;

- Quando ocorre, dá resultados incorretos ( Instr<sub>I</sub> e não Instr<sub>J</sub>)

## ▶ Não pode acontecer no pipeline do DLX porque :

- Todas instruções ocupam 5 estágios, e
- Escritas são sempre no estágio 5

## ▶ Pipelines mais complicados podem apresentar hazards dos tipos **WAR** e **WAW**.

# Sumário

---

## ✓ Pipeline

- ✓ Introdução

- ✓ Pipelines em Computadores

- ✓ Arquitetura DLX

- ✓ Organização DLX com Pipelines

## ♥ Hazards

- ✓ Hazards Estruturais

- ✓ Hazards de Dados

- Forwarding





# Forwarding pode evitar Hazard de Dados!

Fig 3.10, Página 149

Tempo (em ciclos de clock)

Ordem das Instr.

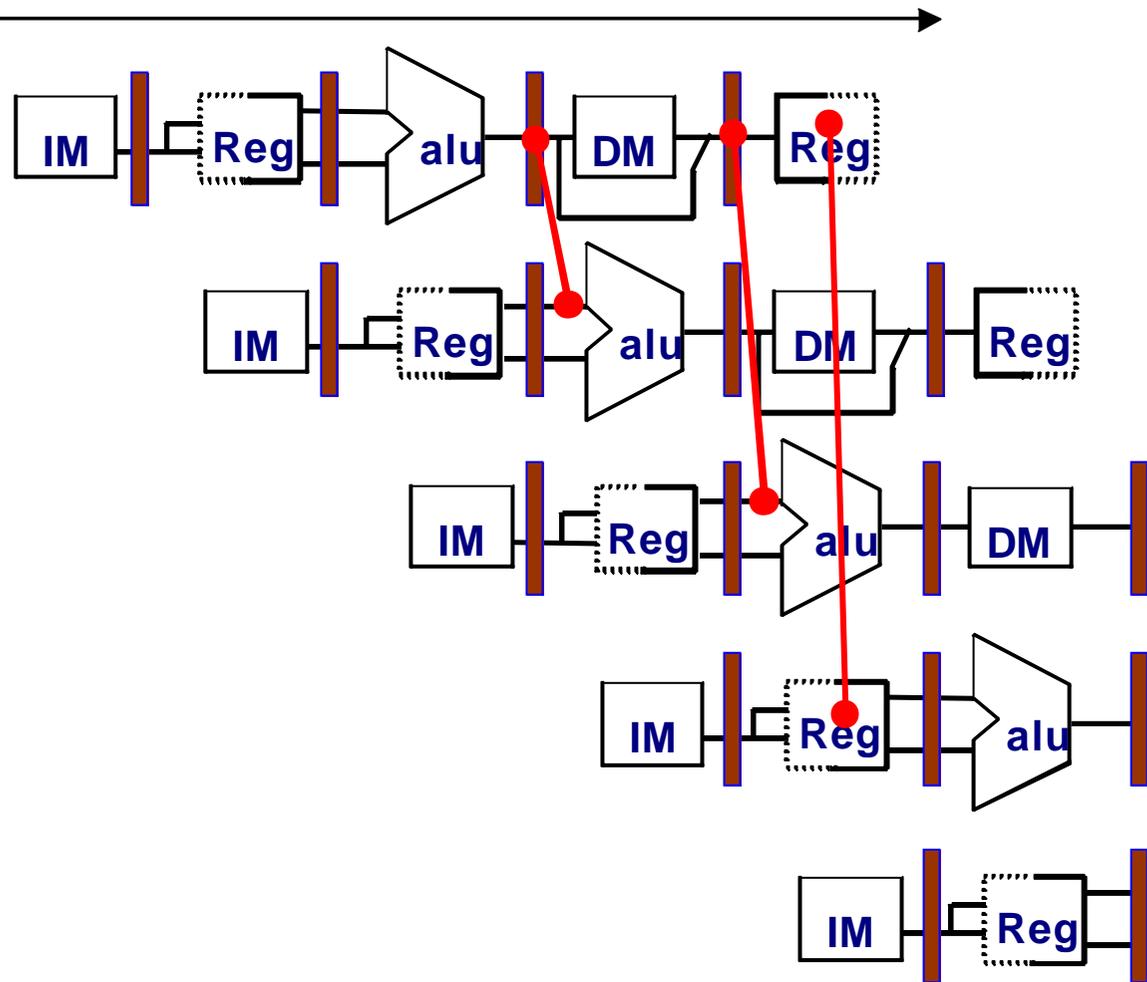
add r1,r2,r3

sub r4,r1,r3

and r6,r1,r7

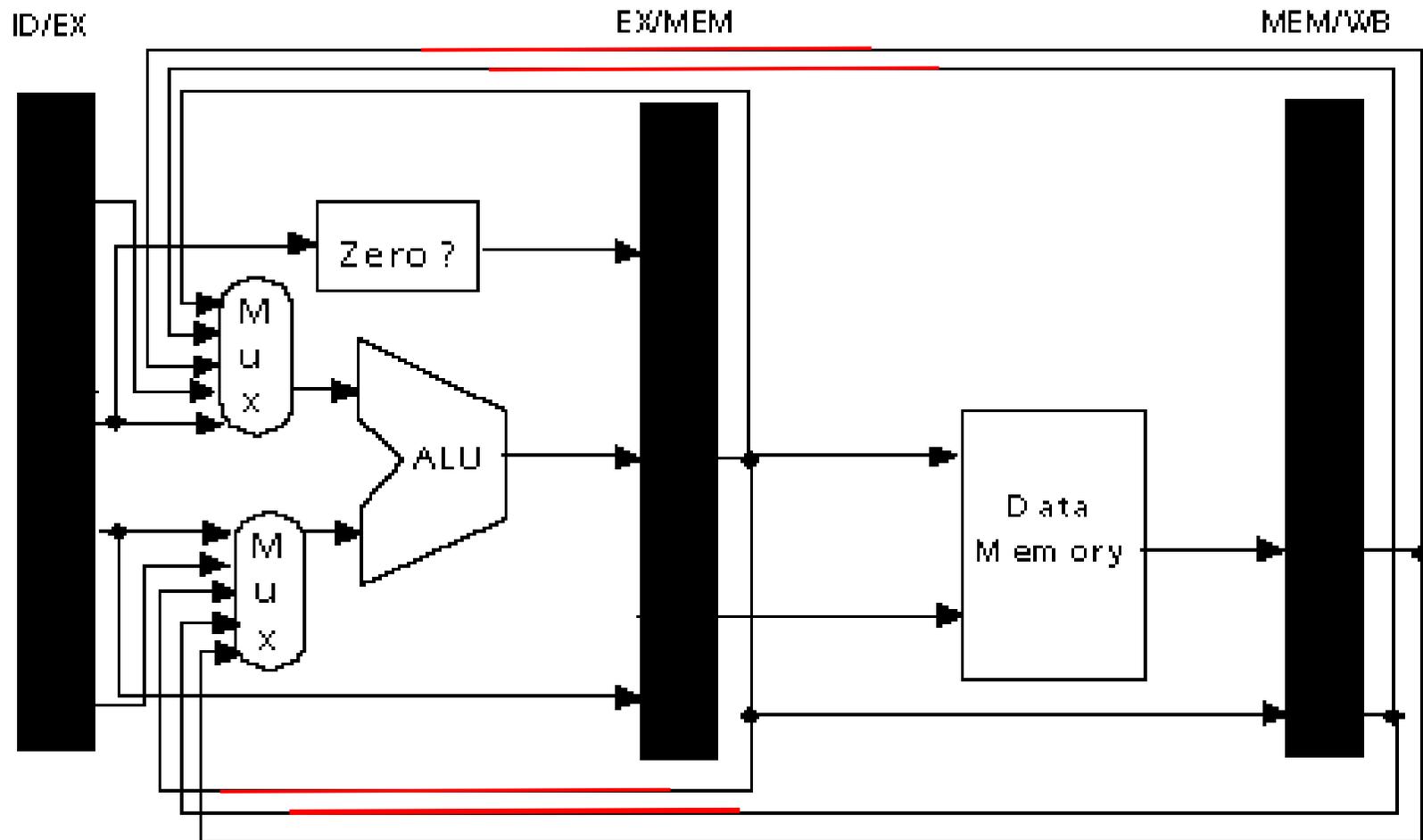
or r8,r1,r9

xor r10,r1,r11



# Mudança de HW para Forwarding

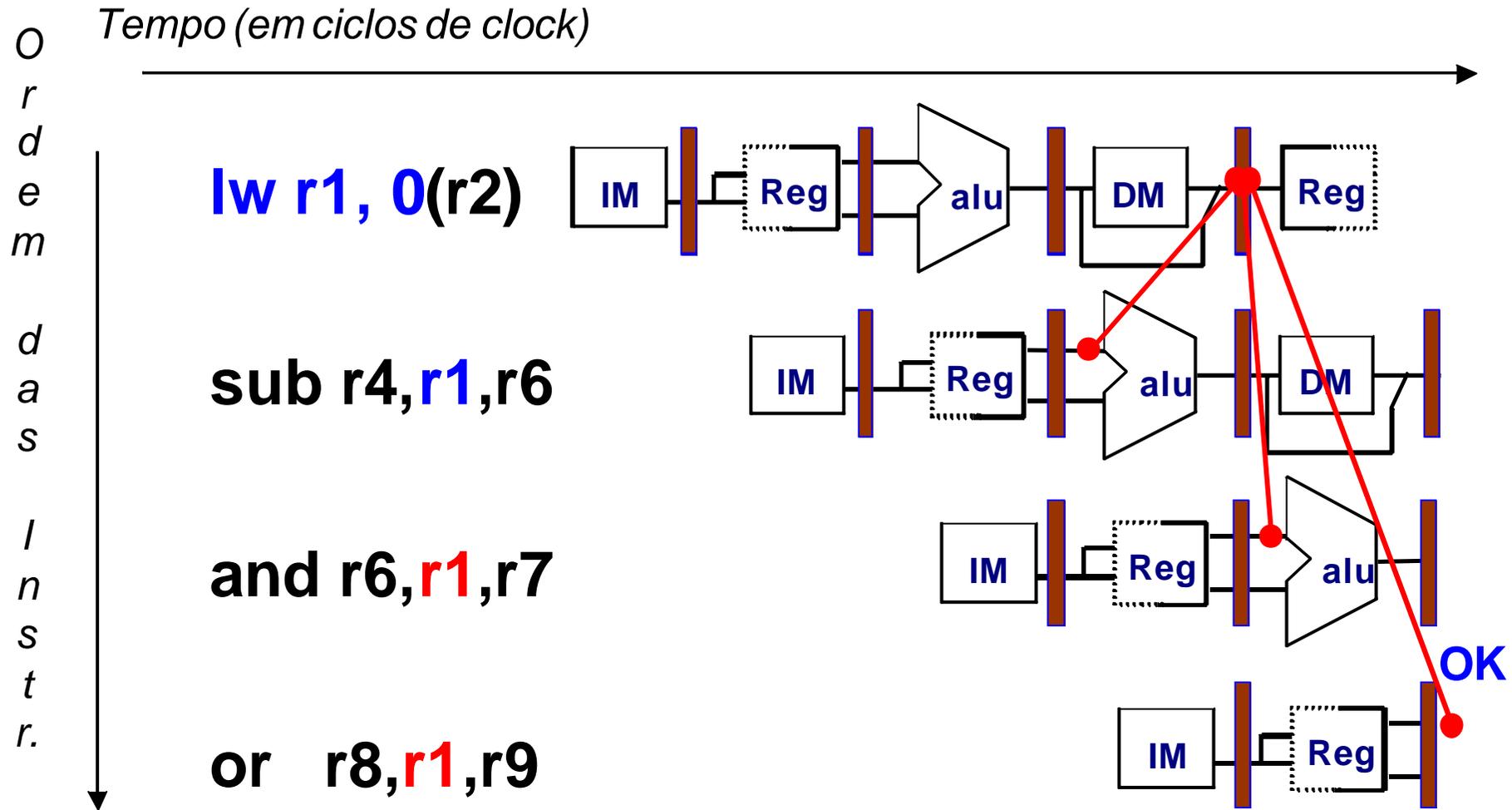
Fig 3.20, Página 161





# Hazard de Dados mesmo com Forwarding

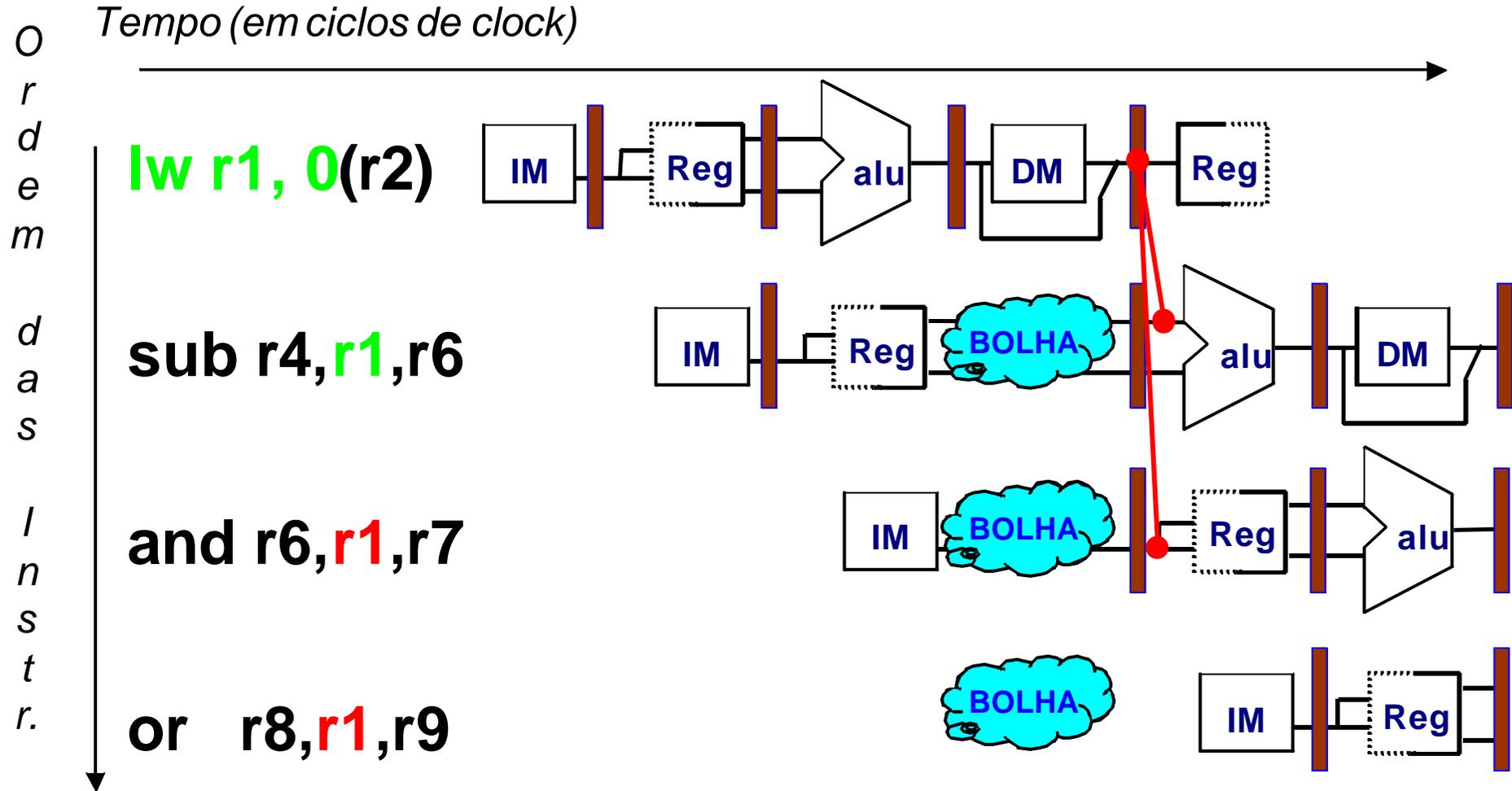
Fig 3.12, Página 153





# Hazard de Dados mesmo com Forwarding

Fig 3.13, Página 154





## *Escalonamento em Software para evitar Hazards de Loads*

---

**O compilador tenta produzir código eficiente para**

**a = b + c;**

**d = e - f;**

**assumindo a, b, c, d, e, e f em memória.**

**Código lento:**

```
LW    Rb,b
LW    Rc,c
ADD   Ra,Rb,Rc
SW    a,Ra
LW    Re,e
LW    Rf,f
SUB   Rd,Re,Rf
SW    d,Rd
```

**Código Rápido:**

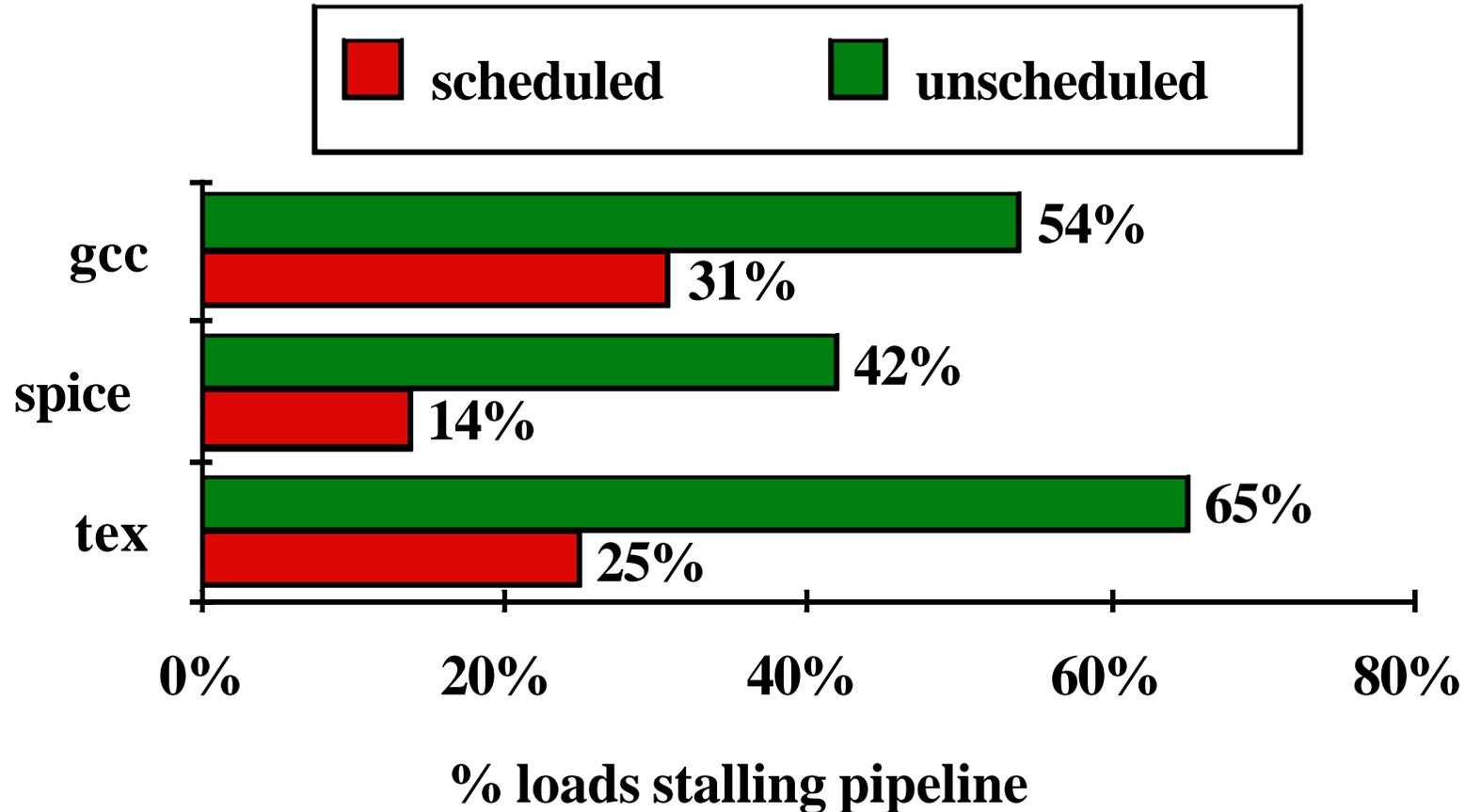
```
LW    Rb,b
LW    Rc,c
LW    Re,e
ADD   Ra,Rb,Rc
LW    Rf,f
SW    a,Ra
SUB   Rd,Re,Rf
SW    d,Rd
```







## *Compilador pode evitar Stalls de Loads*



# *Resumo de Pipelines*

---

- Superpõe tarefas, é fácil se tarefas são totalmente independentes;
- Speed Up  $\leq$  Profundidade do Pipeline; se CPI ideal é 1, então:

$$\text{Speedup} = \frac{\text{Pipeline Depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle Unpipelined}}{\text{Clock Cycle Pipelined}}$$

- Hazards limitam desempenho de pipelines:
  - Estrutural: precisa de mais recursos de HW;
  - Dados: precisa de forwarding e escalonamento por compilador;
  - Controle: discute-se em Arquitetura de Computadores.

**Por hoje é só! Até a próxima!**

---



Organização de Computadores