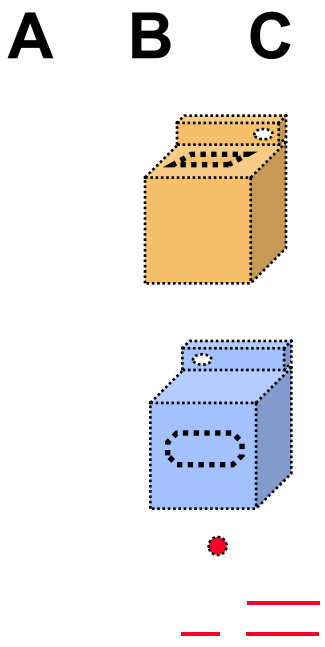
DCC007 – Organização de Computadores II

Aula 4 – Pipelining
Hazards: Estrutural e Dados

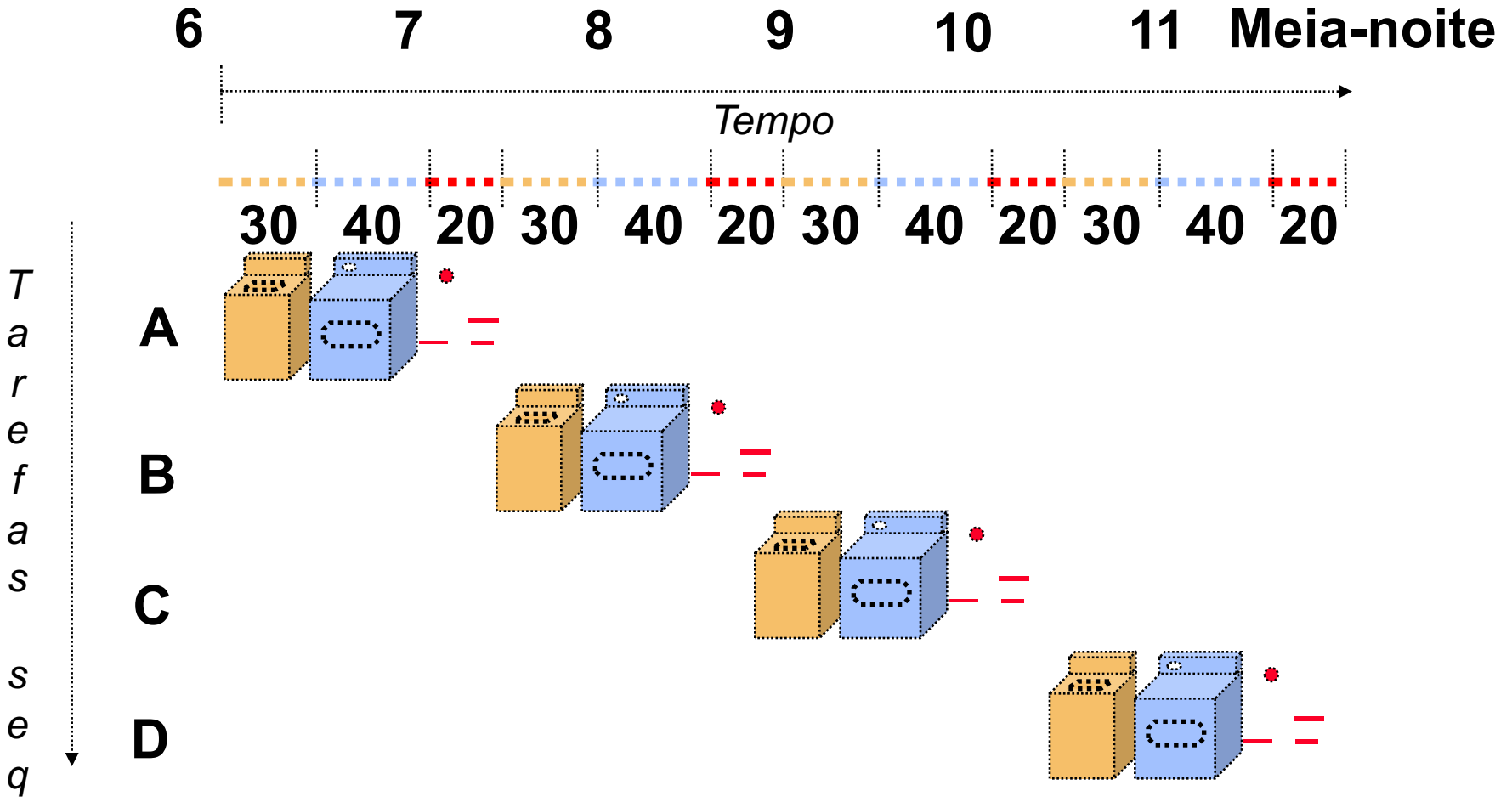
Prof. Omar Paranaiba Vilela Neto

Pipelining: É Natural!

- Lavanderia
- 4 pessoas A, B, C e D possuem 4 sacolas de roupa para lavar, secar e dobrar
- Lavar leva 30 minutos
- Secar leva 40 minutos
- Dobrar leva 20 minutos

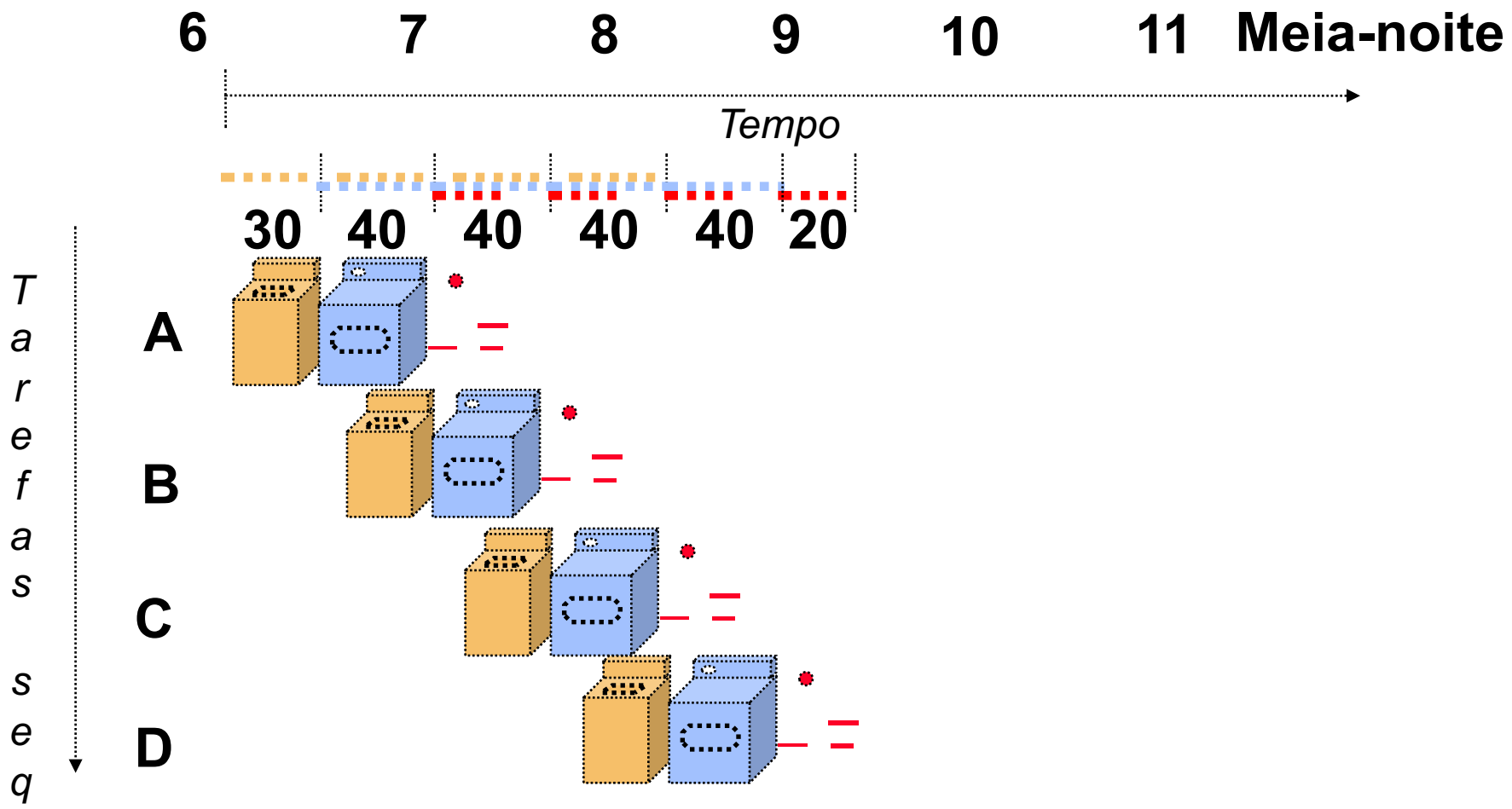


Lavanderia Sequencial



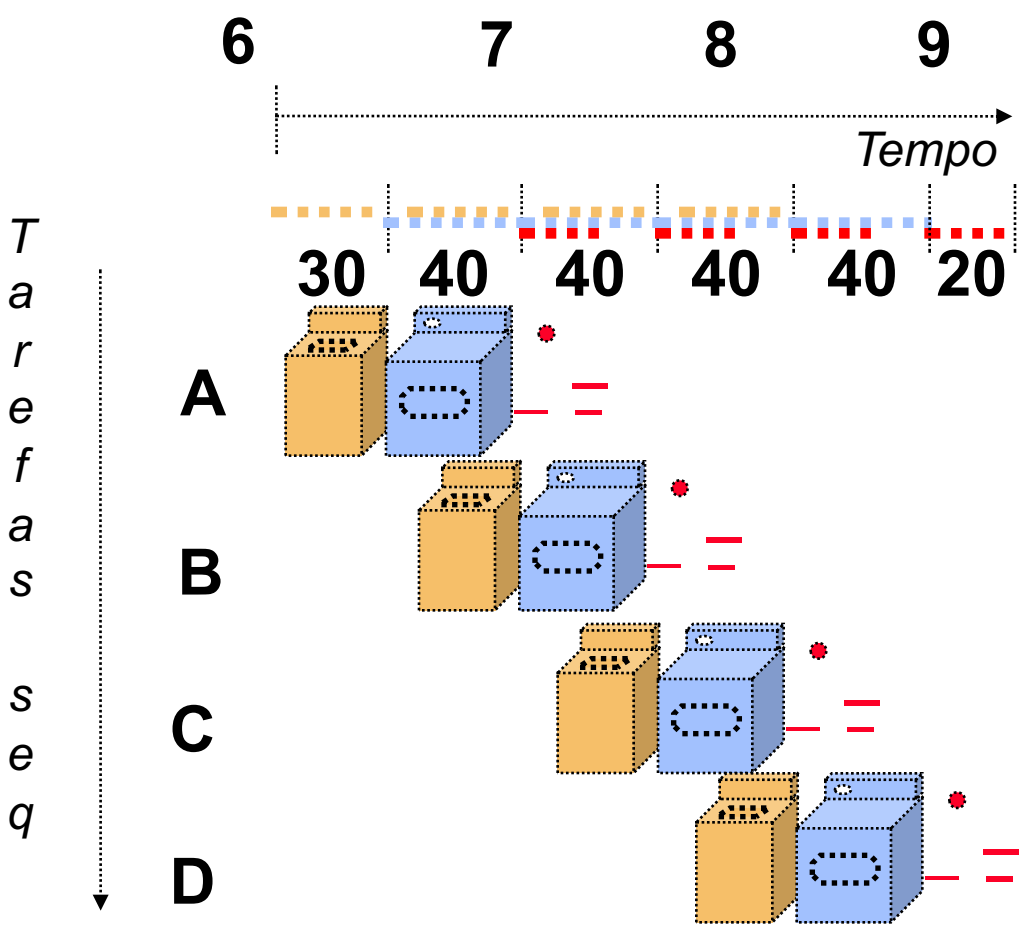
- Lavanderia sequencial leva 6 horas para terminar
- Se eles conhecessem computação, quanto tempo levaria?

Lavanderia com Pipelining



- Lavanderia com *pipelining* leva 3.5 horas !!!

Lições Aprendidas



- *Pipelining* não melhora a latência de uma única tarefa, mas melhora o throughput do trabalho todo
- Taxa de inserção de tarefas é **limitada** pela tarefa mais lenta
- Existem múltiplas tarefas sendo executadas em um dado instante
- SpeedUp potencial = número de estágios
- Tempo para encher o pipeline e tempo de dreno reduzem o speedup

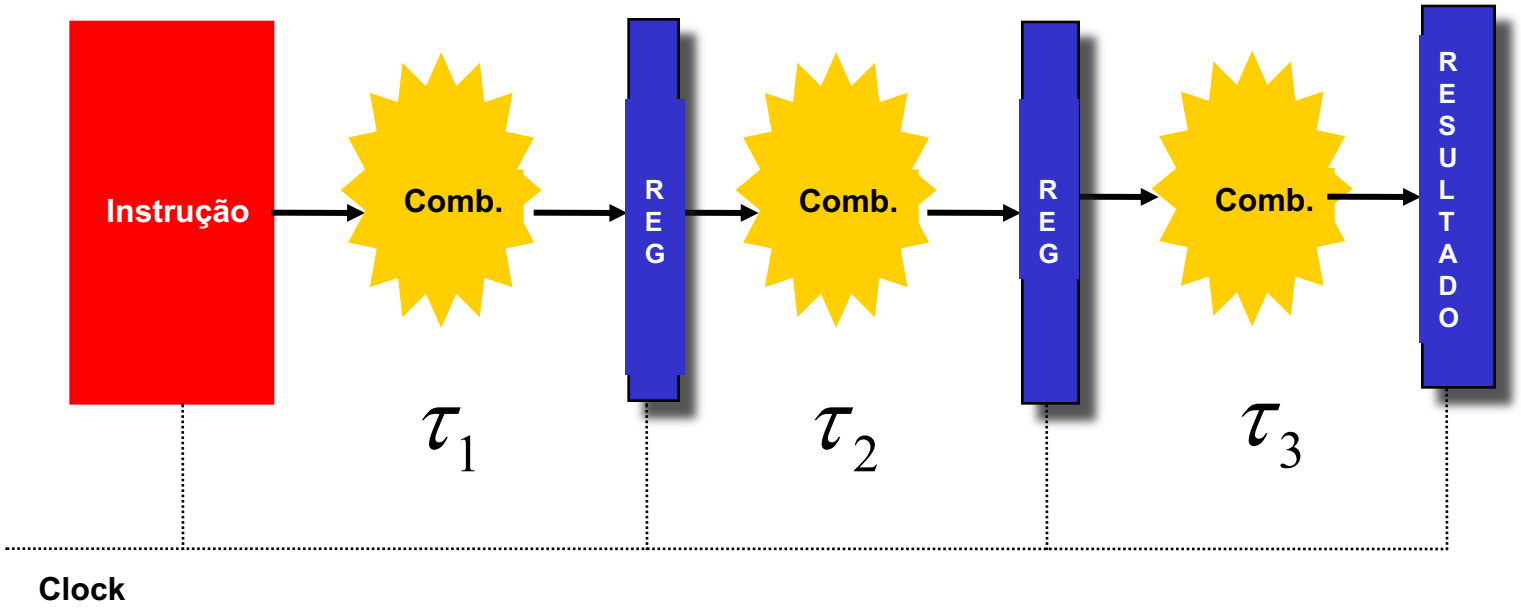
Pipelining

- Múltiplas instruções podem ser executadas a cada instante
- Leva em consideração o paralelismo existente entre instruções
- Chave para **implementação eficiente** em qualquer processador moderno

Pipelining

- Todos objetos passam por todos estágios;
- 2 estágios não compartilham recursos;
- O tempo de cada estágio é igual;
- O agendamento de um novo objeto no pipeline não é afetado pelas transações correntes no pipeline.
  - Problema: isto é verdade em uma linha de produção industrial;
  - Instruções dependem umas das outras - Hazards

Implementação de *Pipeline*

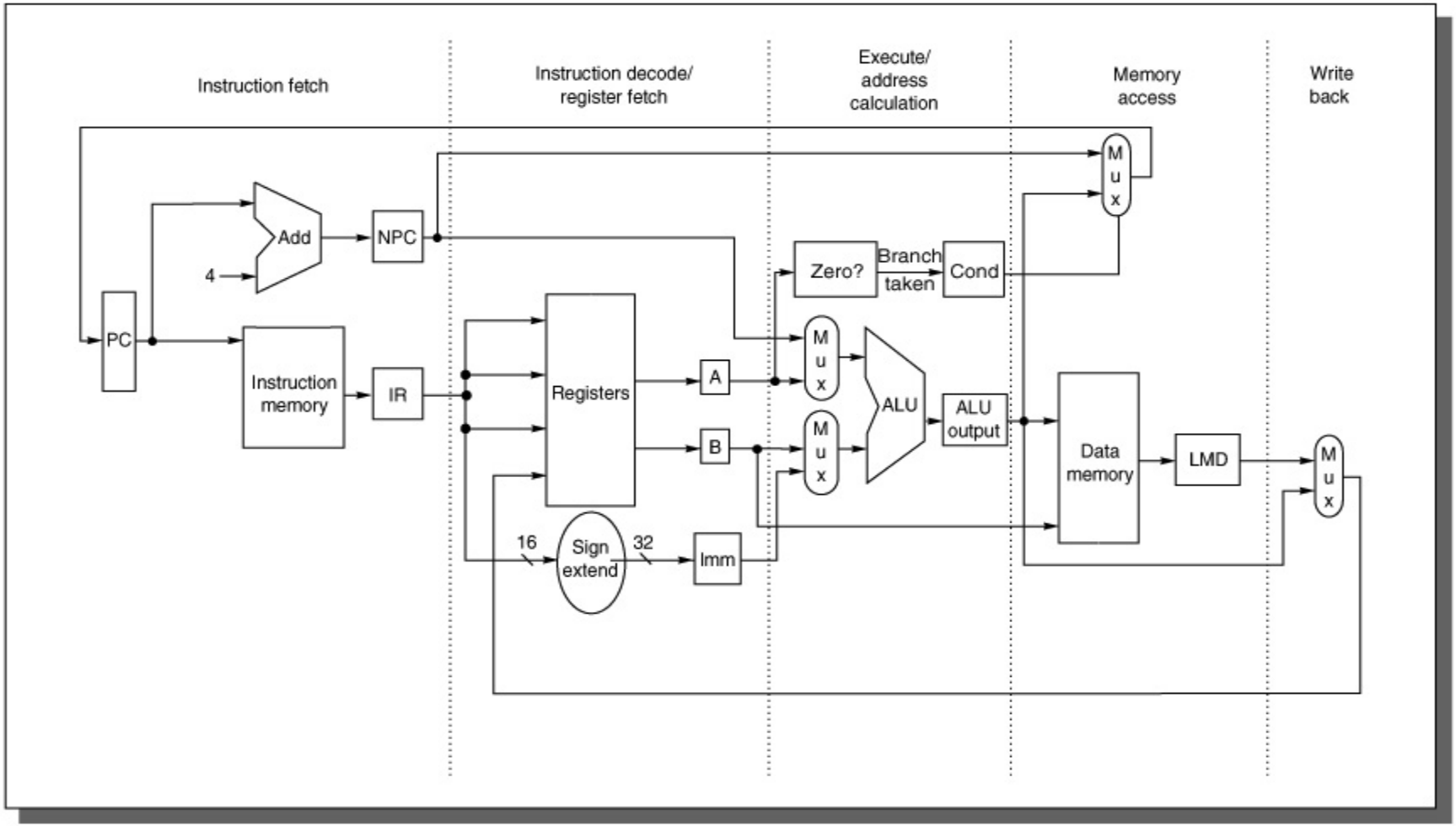


Tempo do Clock →

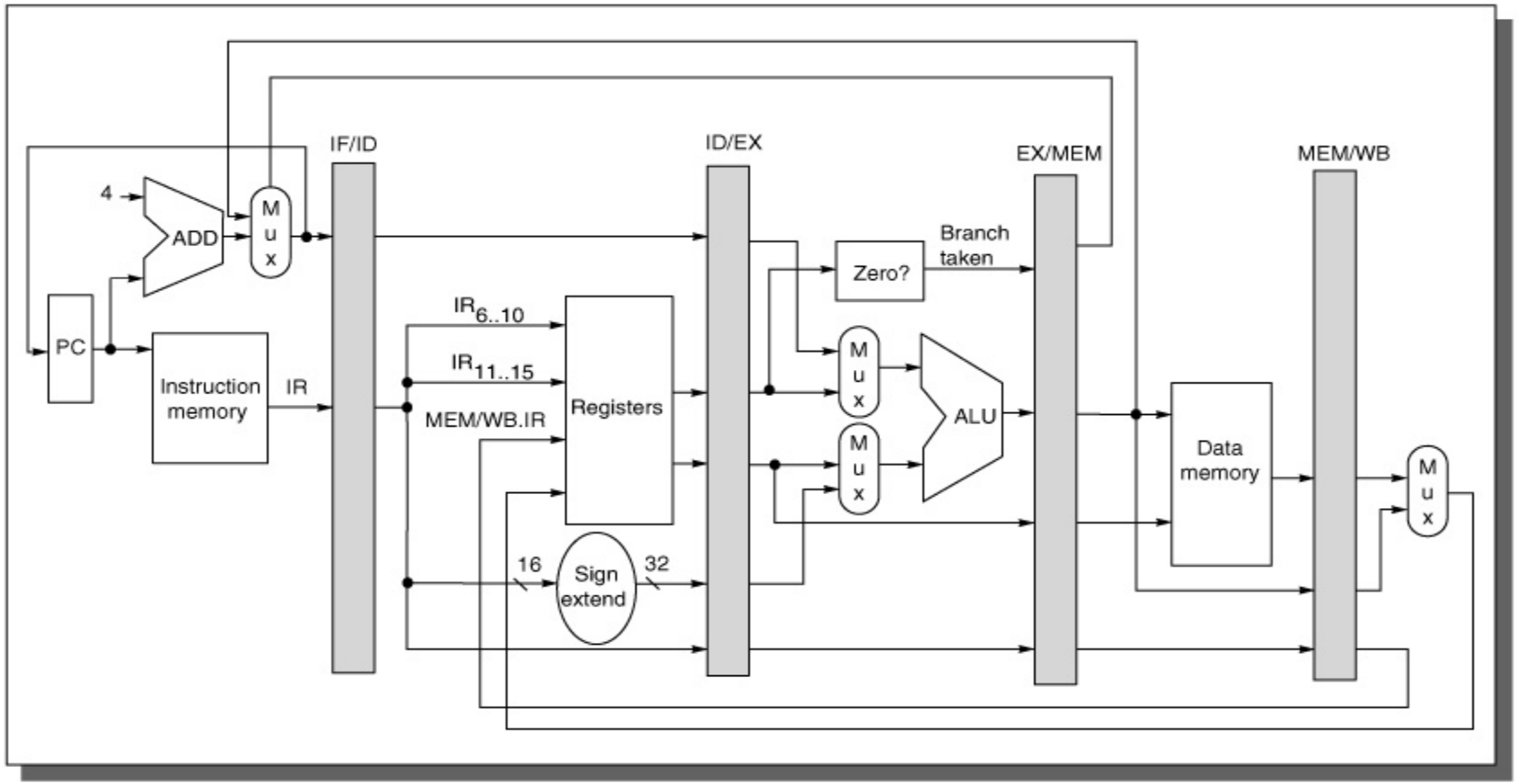
$$t_{s/pipe} = \tau_1 + \tau_2 + \tau_3$$

$$t_{c/pipe} = \max(\tau_1, \tau_2, \tau_3)$$

Implementação do MIPS
Sem Pipeline



Implementação do MIPS com Pipeline



Decodificação local para uma instrução em cada fase do pipeline

Desempenho do Processador

“Iron Law”

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

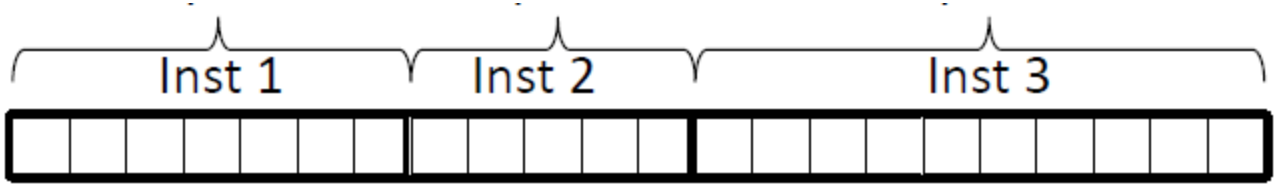
Desempenho do Processador

Máquina Ciclo Único

- 3 Instruções;
- 3 ciclos de clock;
- CPI = 1;
- Tempo de ciclo longo.

Desempenho do Processador

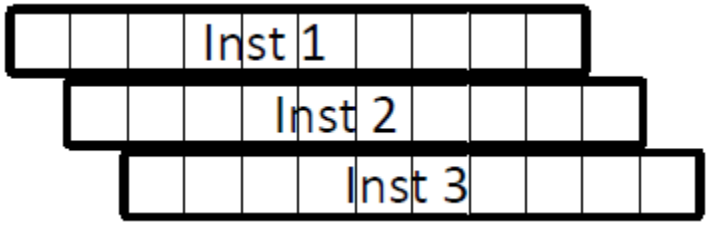
Máquina Multiciclo



- 3 Instruções;
- 22 ciclos de clock;
- CPI = 7,33;
- Tempo de ciclo curto.

Desempenho do Processador

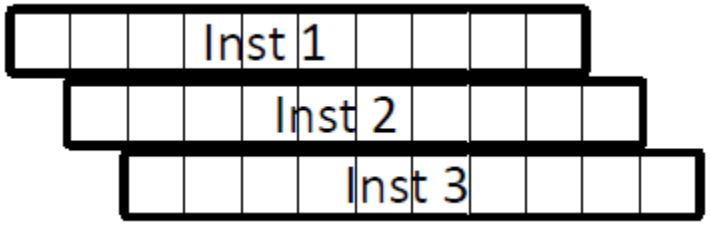
Máquina Pipelined



- 3 Instruções;
- 3 ciclos de clock (ideal);
- CPI = 1 (ideal);
- Tempo de ciclo curto.

Desempenho do Processador

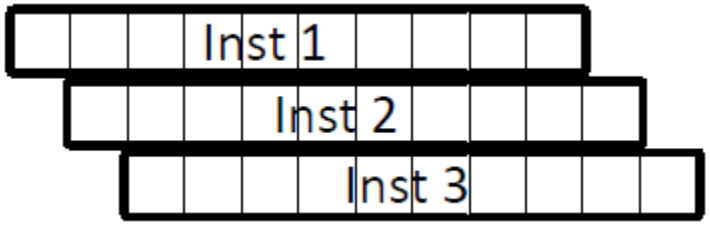
Máquina Pipelined



- 3 Instruções;
- 12 ciclos de clock;
- CPI = 4;
- Tempo de ciclo curto.

Desempenho do Processador

Máquina Pipelined



- N >>>> 3 Instruções;
- N ciclos de clock (ideal);
- CPI = 1 (ideal);
- Tempo de ciclo curto.

Considerações Tecnológicas

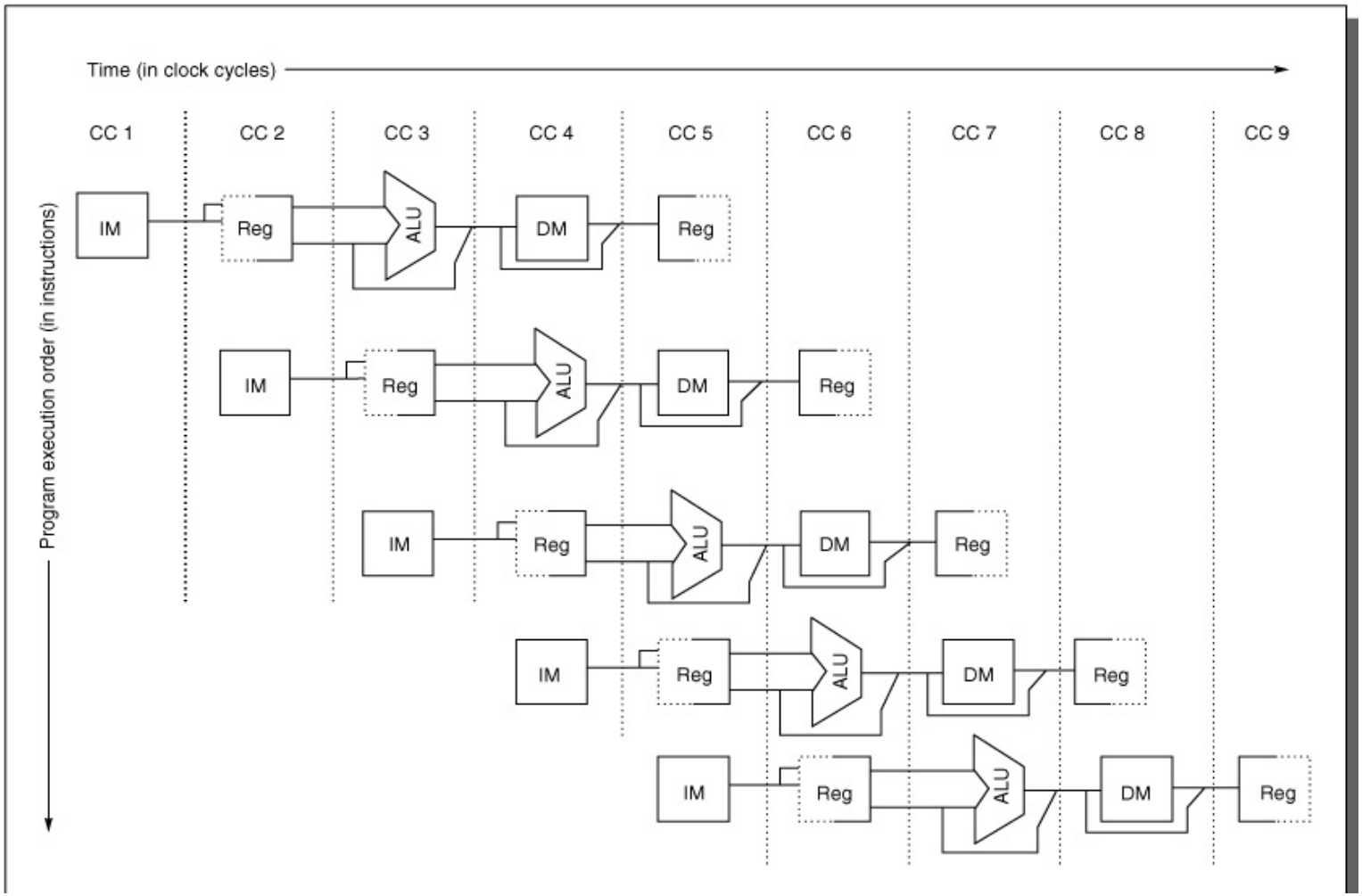
- Realização de algumas memórias cache muito rápidas;
- ALU rápida (principalmente para inteiros)
- Arquivos de registradores mais lentos

Logo, podemos considerar: $t_{IM} \approx t_{RF} \approx t_{ALU} \approx t_{DM} \approx t_{RW}$

Neste curso vamos iniciar no pipeline de 5 estágios.

Algumas implementações comerciais tem mais de **30 estágios de pipeline** para ADD inteiro.

Visualização do Pipeline



Representação Esquemática do *Pipeline*

Instrução	1	2	3	4	5	6	7	8	9
i	IF	ID	EX	MEM	WB				
i+1		IF	ID	EX	MEM	WB			
i+2			IF	ID	EX	MEM	WB		
i+3				IF	ID	EX	MEM	WB	
i+4					IF	ID	EX	MEM	WB

Entretanto, *pipeline* ainda não funciona

Por que Pipelines são Difíceis em Computadores

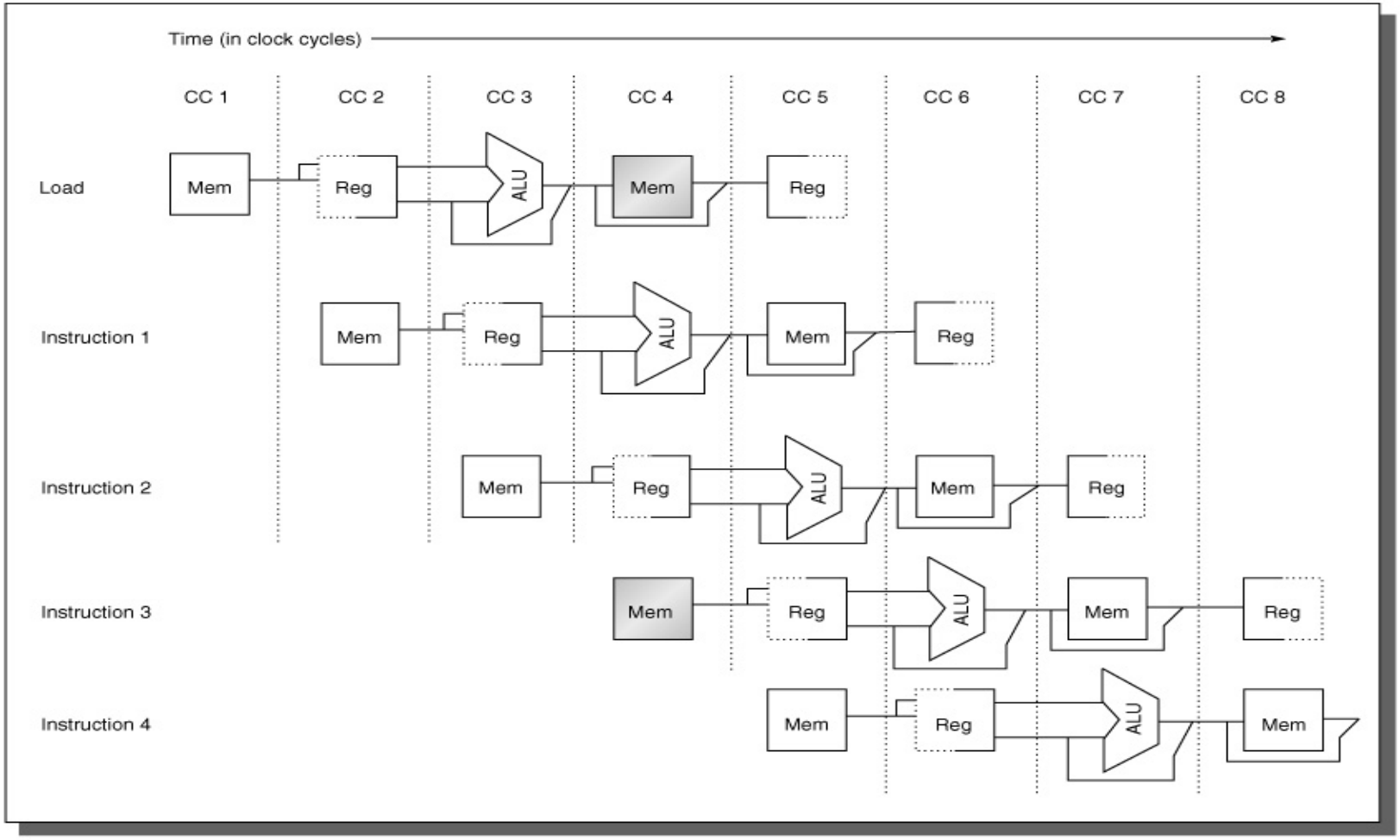
- Limites da arquitetura: Hazards não deixam próxima instrução executar no próximo ciclo
  - Structural hazards: conflito de hardware (**mais de uma instrução quer utilizar a mesma unidade funcional**)
  - Data hazards: instrução **depende de um resultado** de uma instrução **que ainda não completou**
  - Control hazards: Pipelining de **branches** e outras **insruções que modificam o PC**
- Solução comum:
  - Stall o pipeline (inserção de uma "bolha") até o hazard ser resolvido.

Hazard Estrutural

Hazard Estrutural

- Como resolver:
  - Planejamento: programador (compilador) **evita planejar** instruções que criarão hazard;
  - Stall: Hardware inclui **controle que para a máquina** até que a estrutura esteja disponível;
  - Duplicação: **Adiciona mais Hardware**, evitando o conflito de interesse nas estruturas.

Structural Hazard
Porto de Memória Único



Structural Hazard
Porto de Memória Único

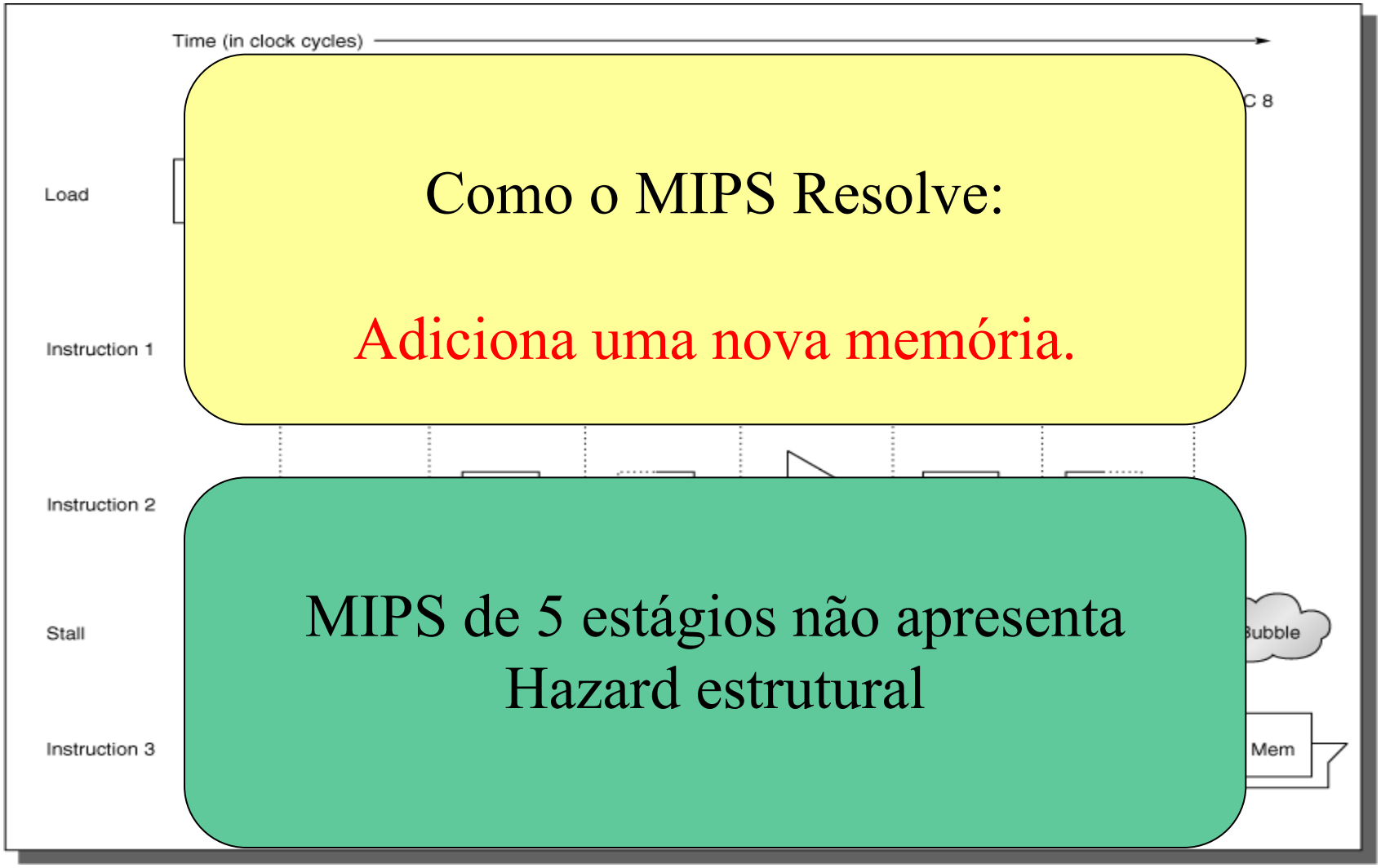


Diagrama Esquemático do *Pipeline*

Instrução	1	2	3	4	5	6	7	8	9
Load	IF	ID	EX	MEM	WB				
Instr. 1		IF	ID	EX	MEM	WB			
Instr. 2			IF	ID	EX	MEM	WB		
Stall				-	-	-	-	-	
Instr. 3					IF	ID	EX	MEM	WB

Representação alternativa (preferível)

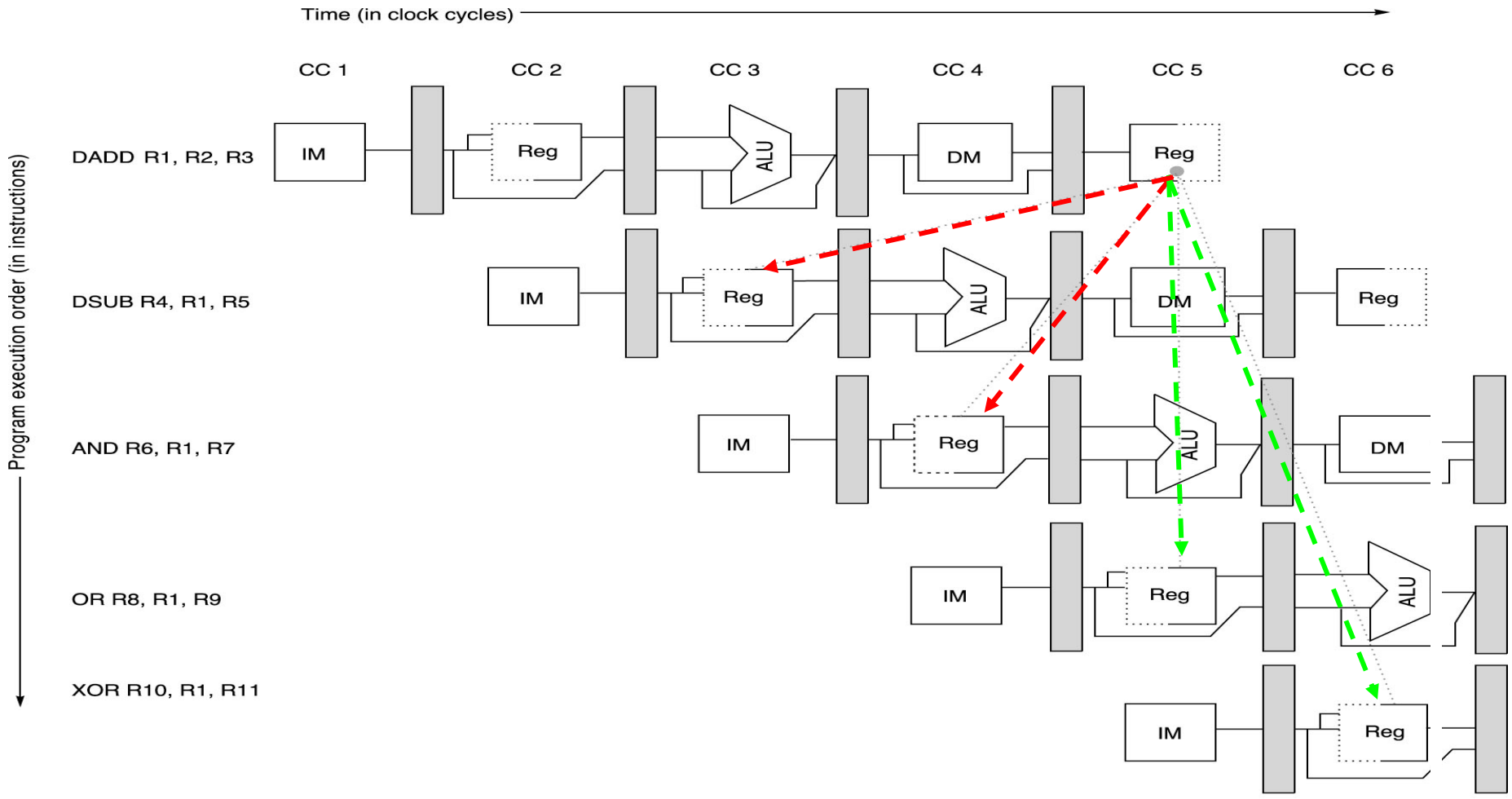
Instrução	1	2	3	4	5	6	7	8	9
Load	IF	ID	EX	MEM	WB				
Instr. 1		IF	ID	EX	MEM	WB			
Instr. 2			IF	ID	EX	MEM	WB		
Instr. 3				-	IF	ID	EX	MEM	WB

Hazard de Dados

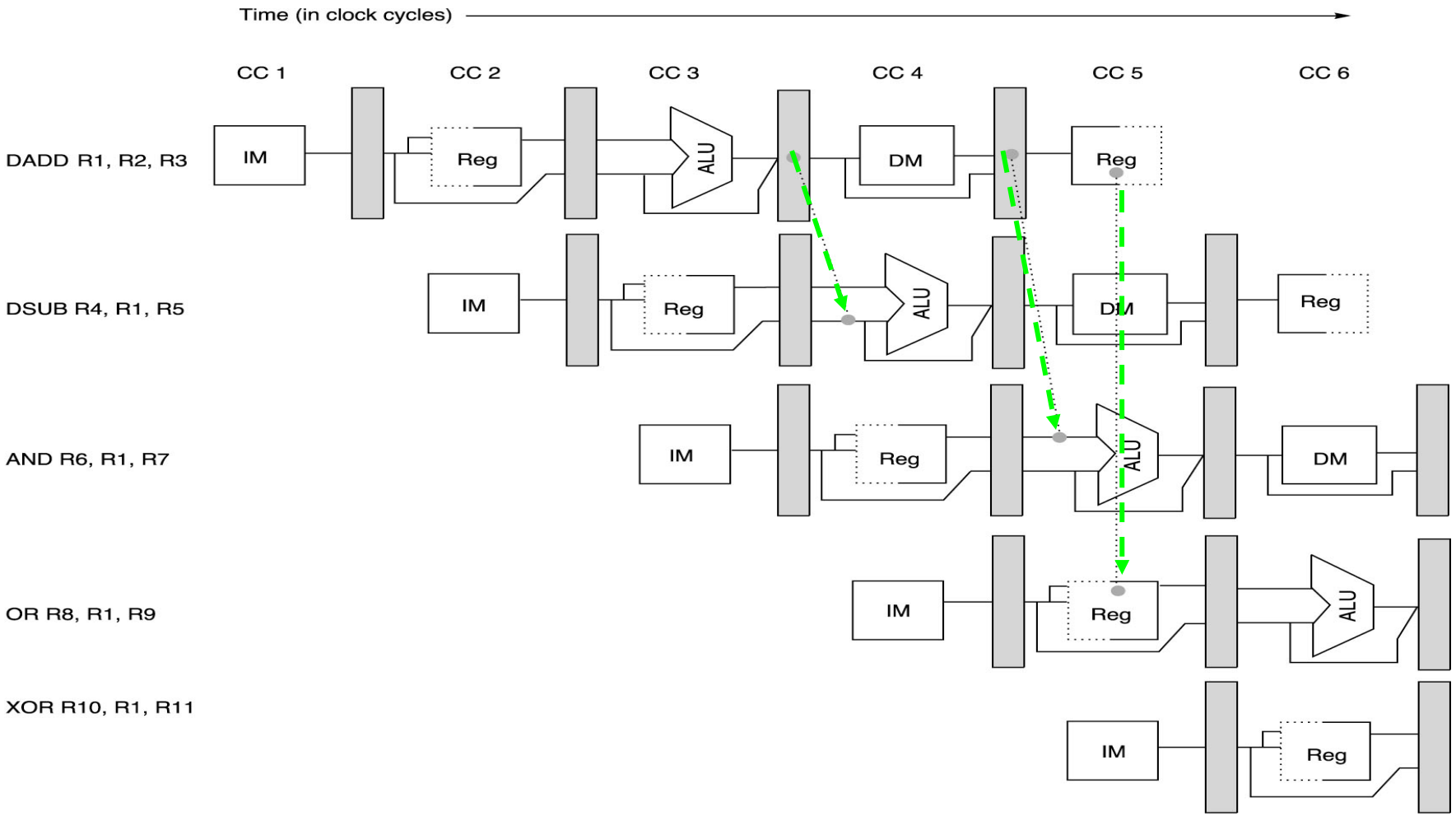
Hazard de Dados

- Como resolver:
  - Planejamento: programador (compilador) **evita planejar** instruções que criarão hazard;
  - Stall: Hardware inclui **controle que para a máquina** até que a instrução anterior disponibilize o dado;
  - Encaminhamento: **Hardware** permite que o dado desejado seja enviado para um estágio anterior.
  - Especulação: **Hardware considera que não há problema**. Se problema aparecer, mata instrução e começa outra vez.

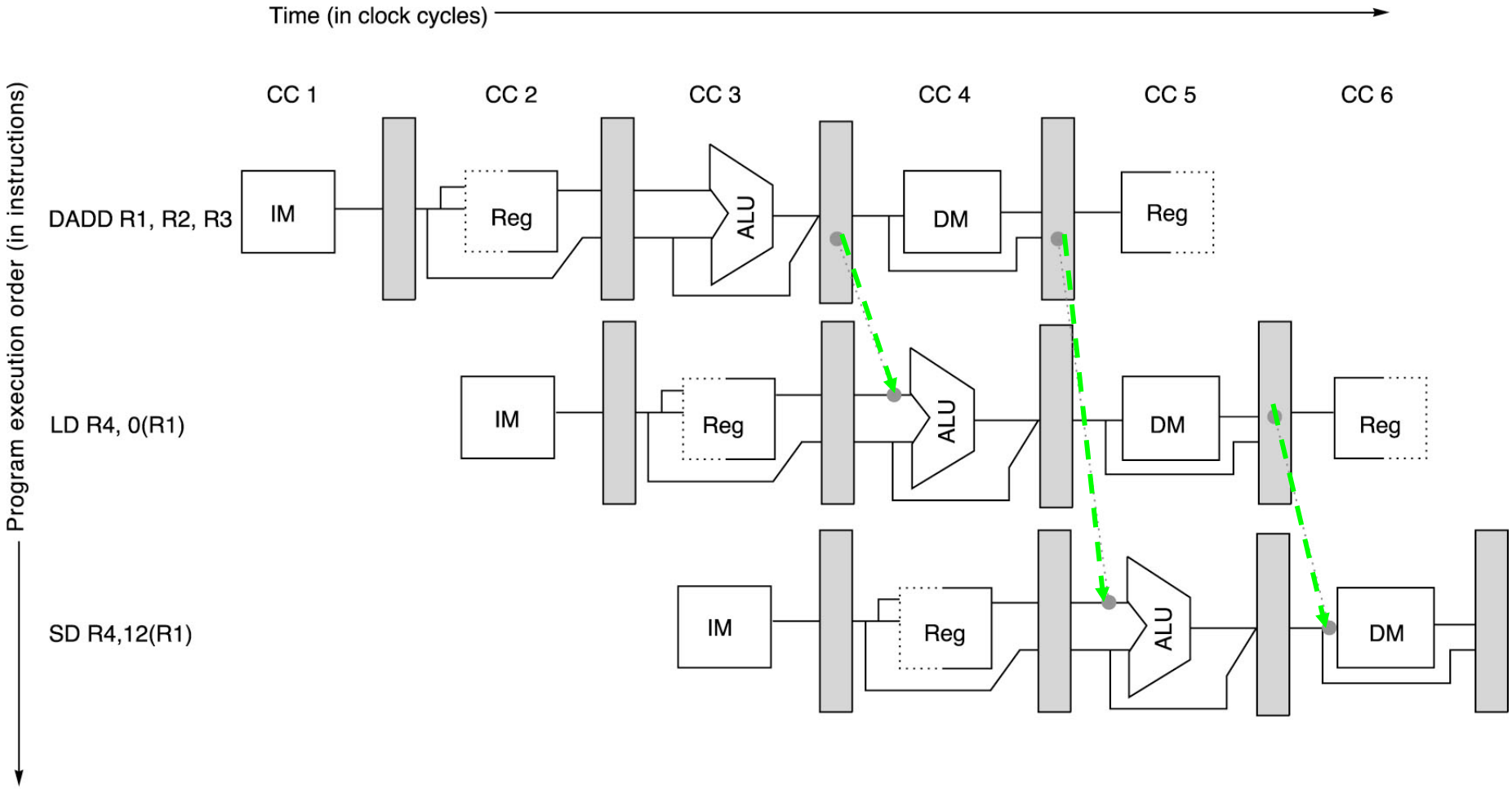
Data Hazard no MIPS



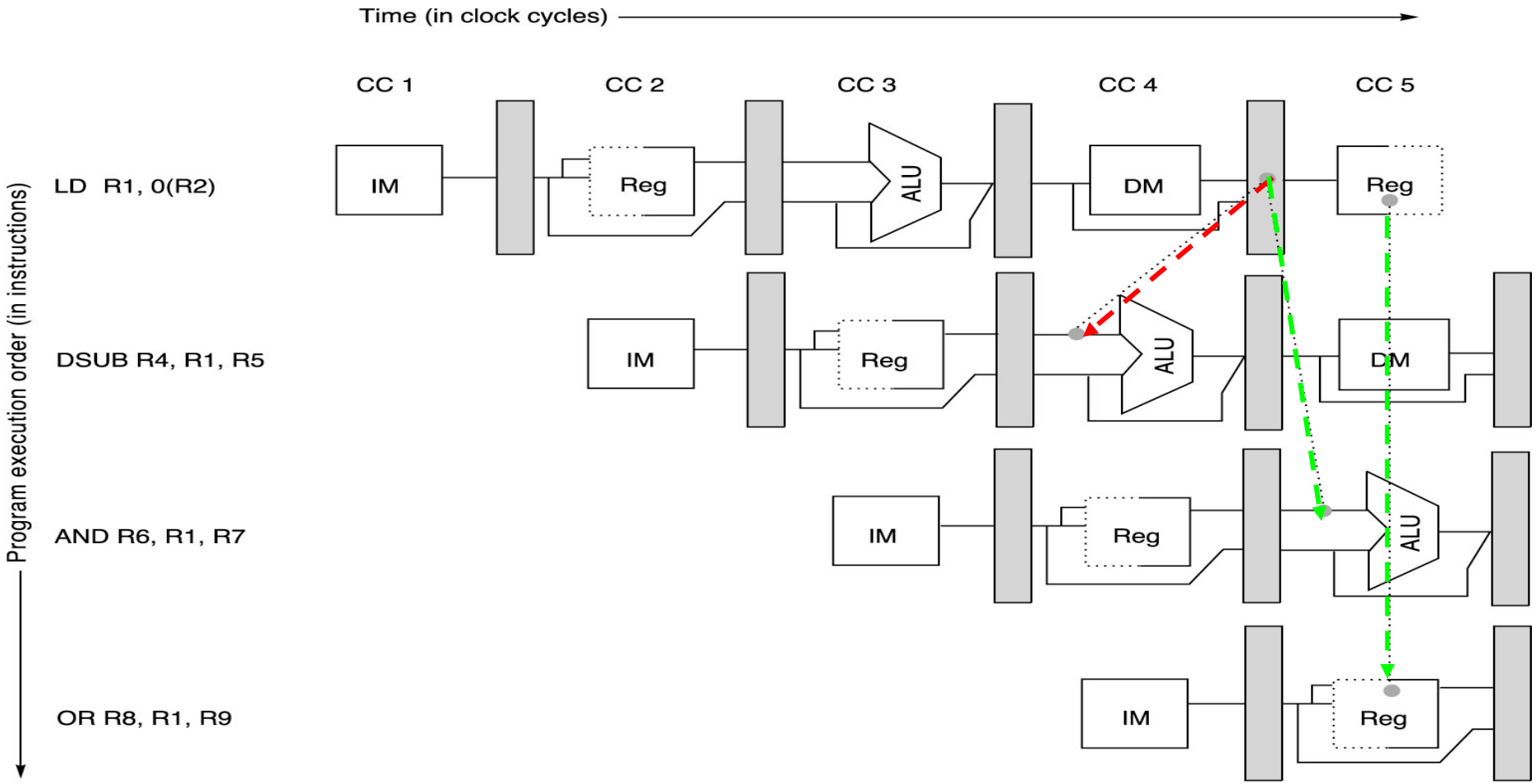
Forwarding para Evitar *Hazards* de Dados



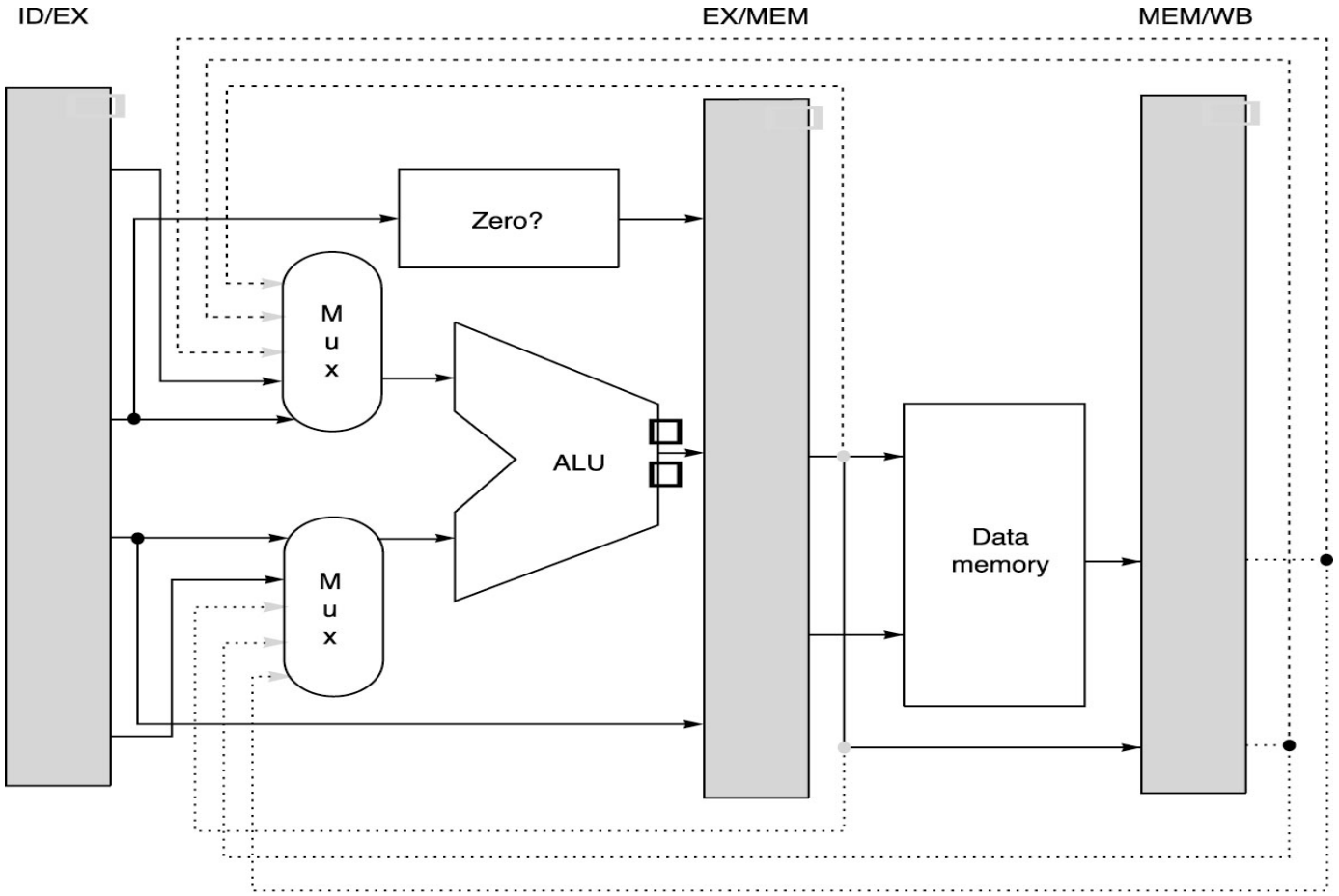
Forwarding para Evitar *Hazard* de Dados



Forwarding para Evitar *Hazard* de Dados – Load



Mudança do *Hardware* para Permitir *Forwarding*



Escalonamento de SW para Evitar *Load Hazards*

Produza o código mais rápido para

a = b + c;

d = e – f;

assumindo que a, b, c, d ,e, f estão na memória.

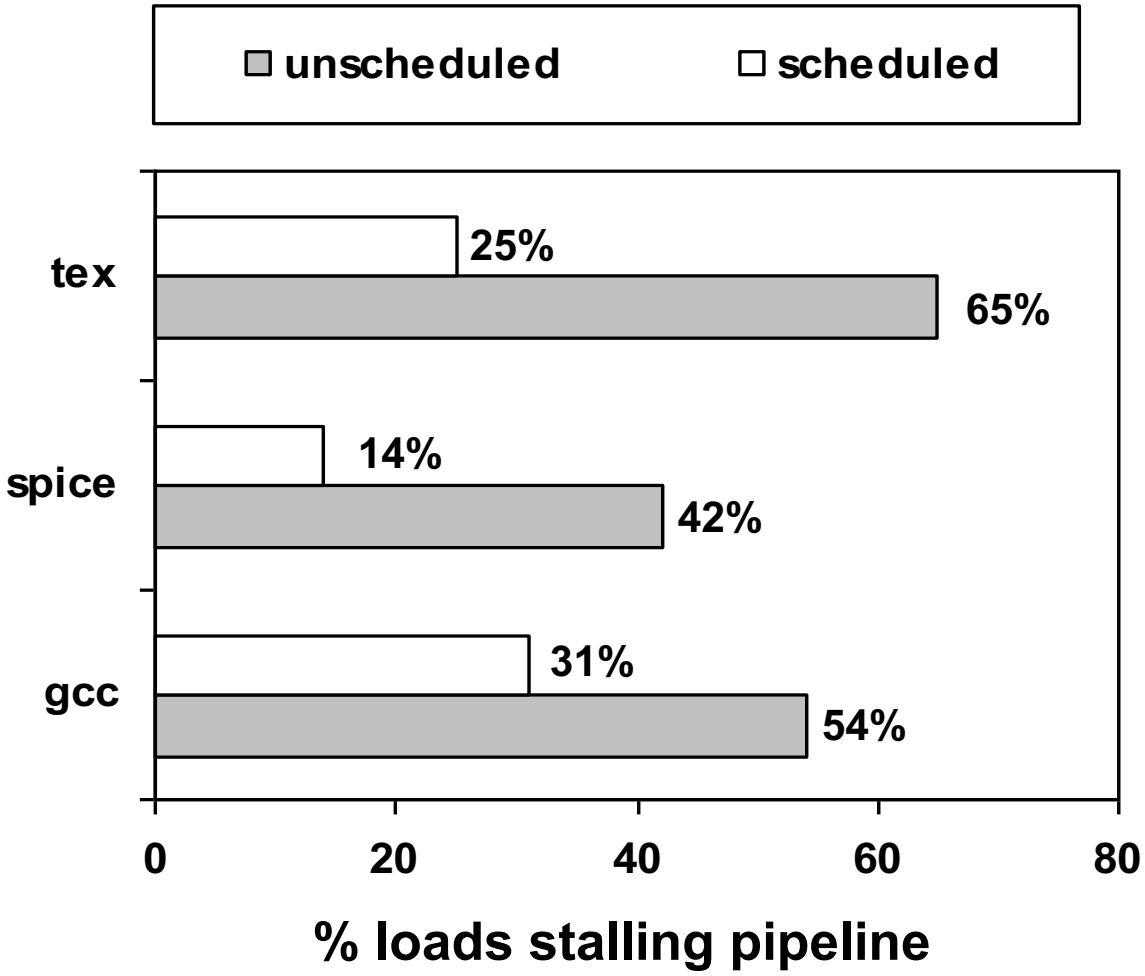
Código lento:

```
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
```

Sucesso dos Compiladores para Evitar *Load Stalls*



Tipos de *Data Hazards*

$Instr_I$ precede $Instr_J$

- *Read After Write* (RAW)
  $Instr_J$ tenta ler operando antes de $Instr_I$ escrever resultado

Tipos de *Data Hazards*

$Instr_I$ precede $Instr_J$

- *Write After Read* (WAR)
  $Instr_J$ tenta escrever resultado antes que $Instr_I$ leia operando

- Não ocorre no *pipeline* do MIPS porque:
  - Todas as instruções levam 5 ciclos,
  - Leitura de operandos ocorrem no estágio 2,
  - Escrita de resultados ocorre no estágio 5

Tipos de *Data Hazards*

$Instr_I$ precede $Instr_J$

- *Write After Write* (WAW)
  $Instr_J$ tenta escrever resultado antes que $Instr_I$ escreva
  - Deixa resultado errado
- Não ocorre no *pipeline* do MIPS porque :
  - Todas as instruções levam 5 ciclos,
  - Escrita de resultados ocorre no estágio 5
- WAR e WAW aparecerão em variações posteriores

Tipos de *Data Hazards*

$Instr_I$ precede $Instr_J$

- *E Read After Read* (RAR) ???
  $Instr_J$ tenta ler resultado antes que $Instr_I$ leia operando
  - NÃO CAUSA PROBLEMA ! ! !
- Só existem três tipos: RAW, WAW, WAR (pelo menos uma escrita tem que existir)

Resumo: Pipelining

- Hazards limitam o desempenho dos computadores:
  - Structural: é necessário alocar mais recursos de HW
  - Data: necessita de forwarding, e escalonamento de instruções pelo compilador
  - Control: a ser discutido