Arquitetura de computadores e sistemas operacionais

Segmentação

Gabriel V C Candido
gabriel.candido@ifpr.edu.br

Instituto Federal do Paraná - Pinhais

INSTITUTO FEDERAL
Paraná
Campus Pinhais

Sumário

Sumário

Processador de ciclo longo

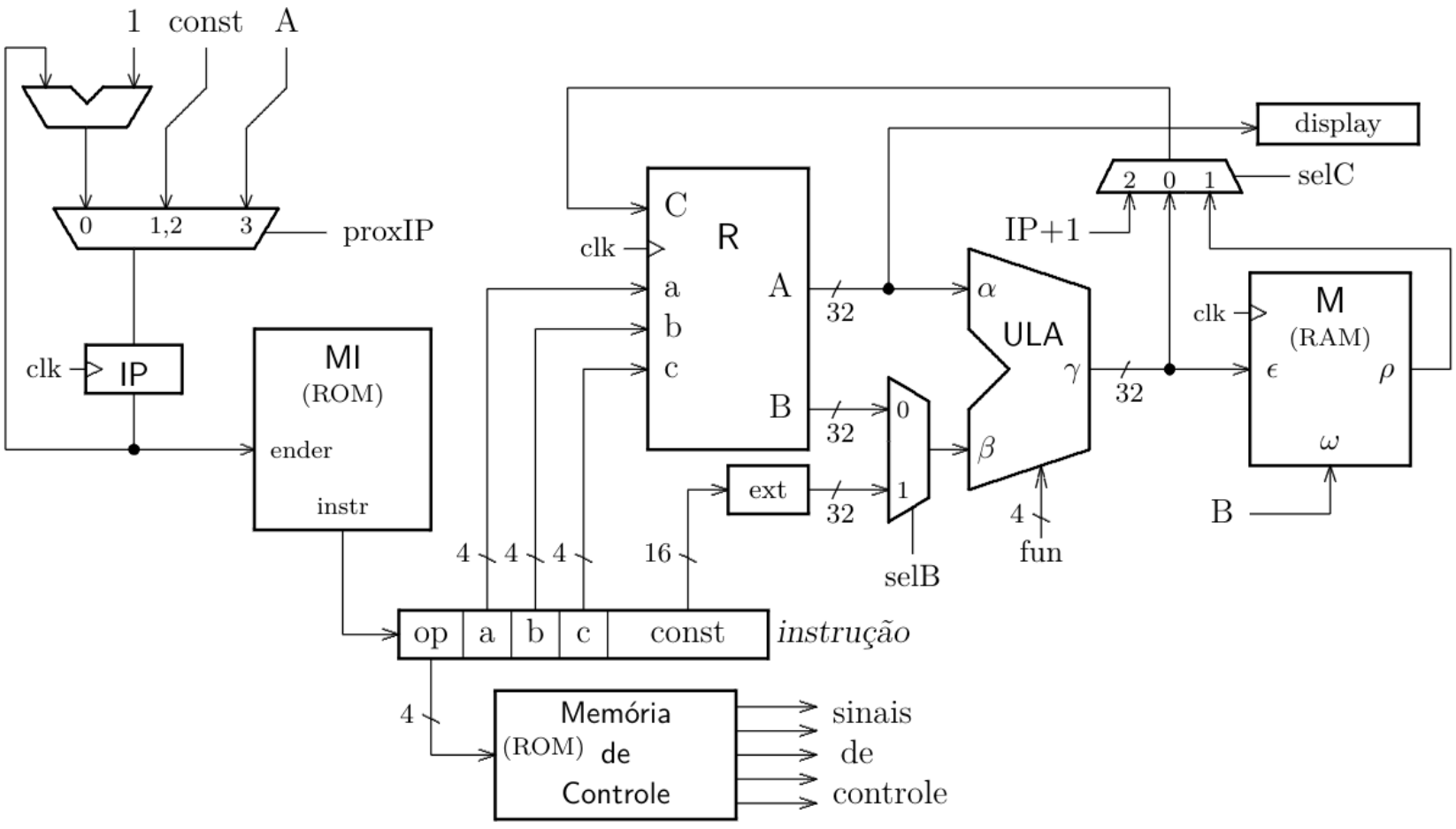

Figura: Processador de ciclo longo. Fonte: RH

INSTITUTO FEDERAL
Paraná
Campus Pinhais

Processador de ciclo longo

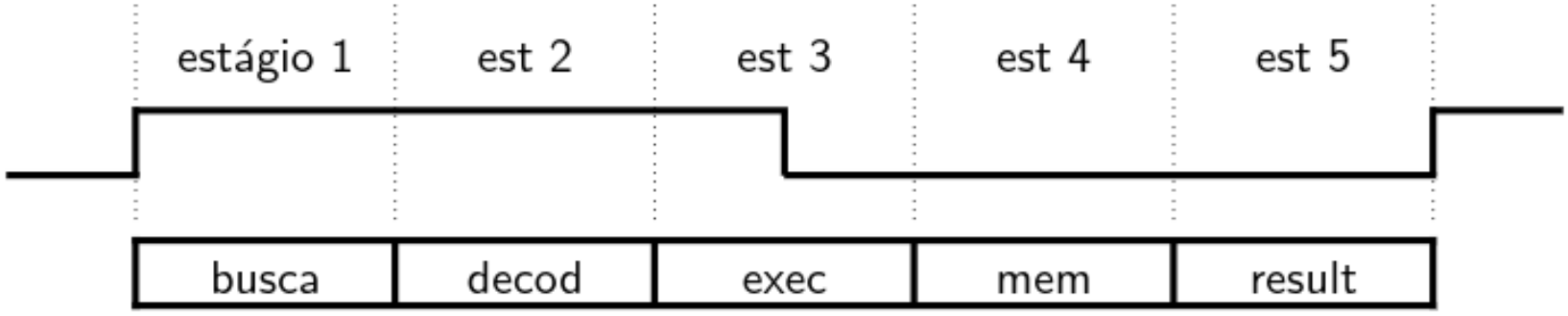

Figura: Instrução *load* em cinco estágios. Fonte: RH

Analogia com lavanderias

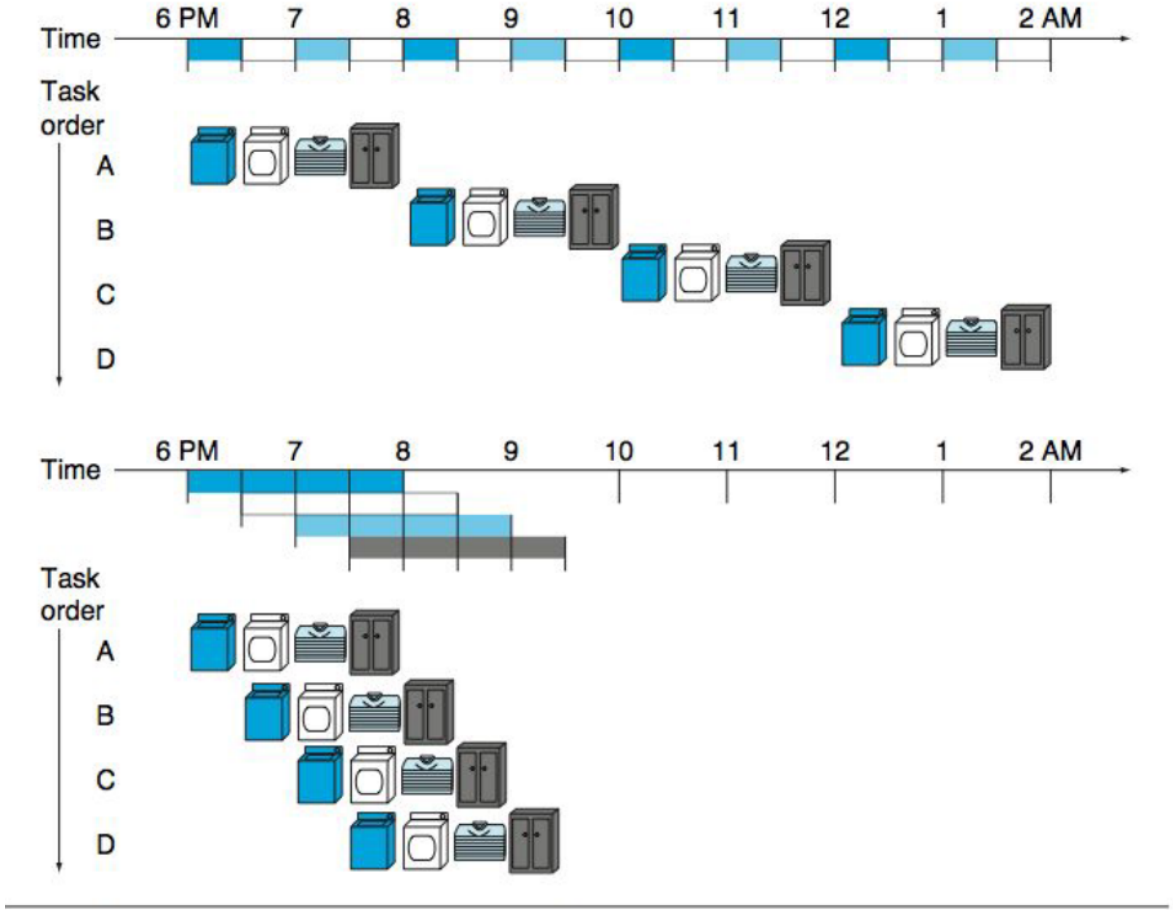

Figura: Analogia de lavanderias para *pipeline*. Fonte: PH 4.5

Segmentação

A ideia é dividir o trabalho em uma linha de produção para aumentar a eficiência

Segmentação

A ideia é dividir o trabalho em uma linha de produção para aumentar a eficiência

O trabalho deve ser dividido em tamanhos parecidos (de tempo)

Ciclo longo *vs* segmentação

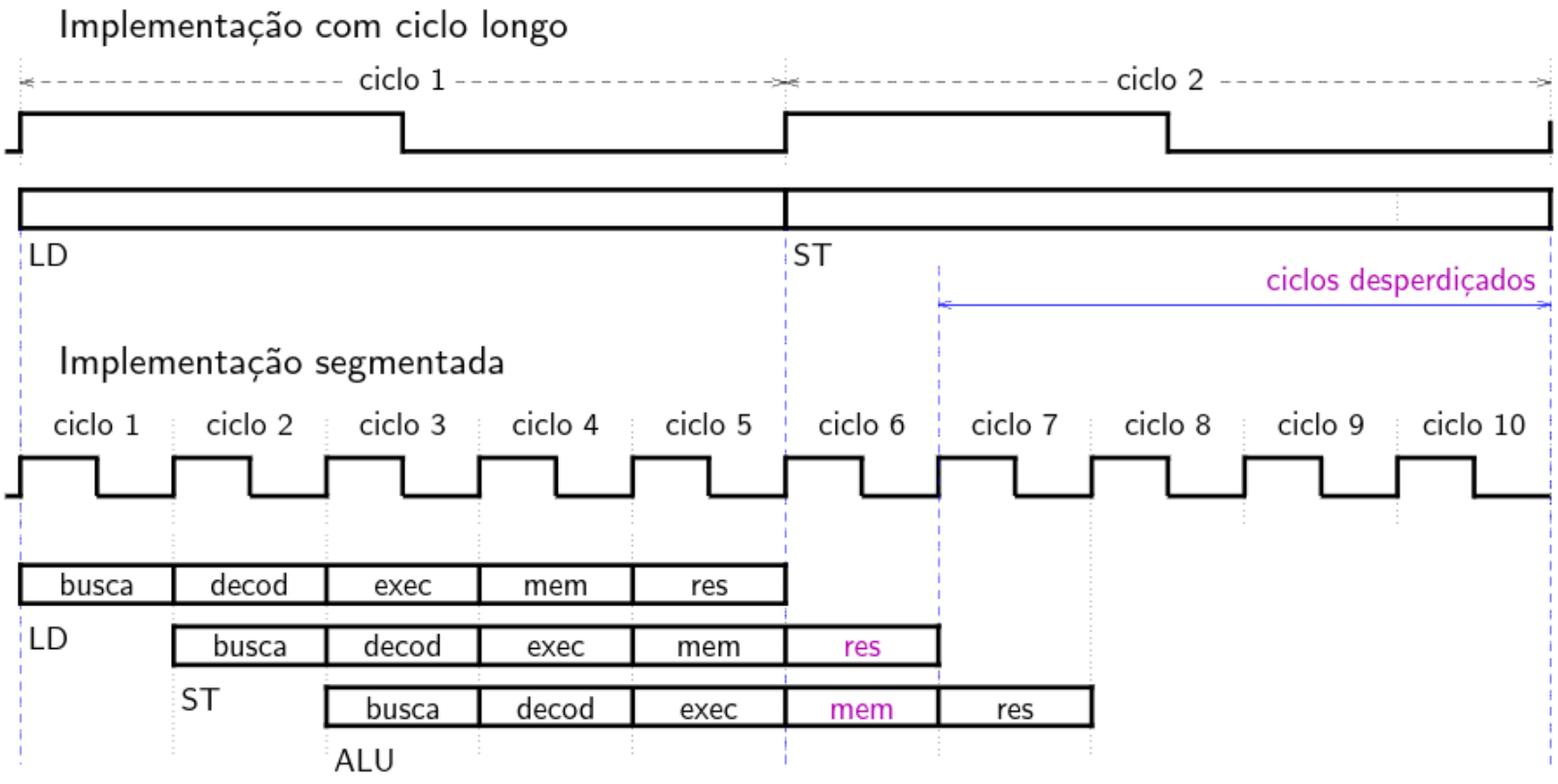

Figura: Ciclo longo *vs* segmentação. Fonte: RH

Sumário

Ciclo longo *vs* segmentação

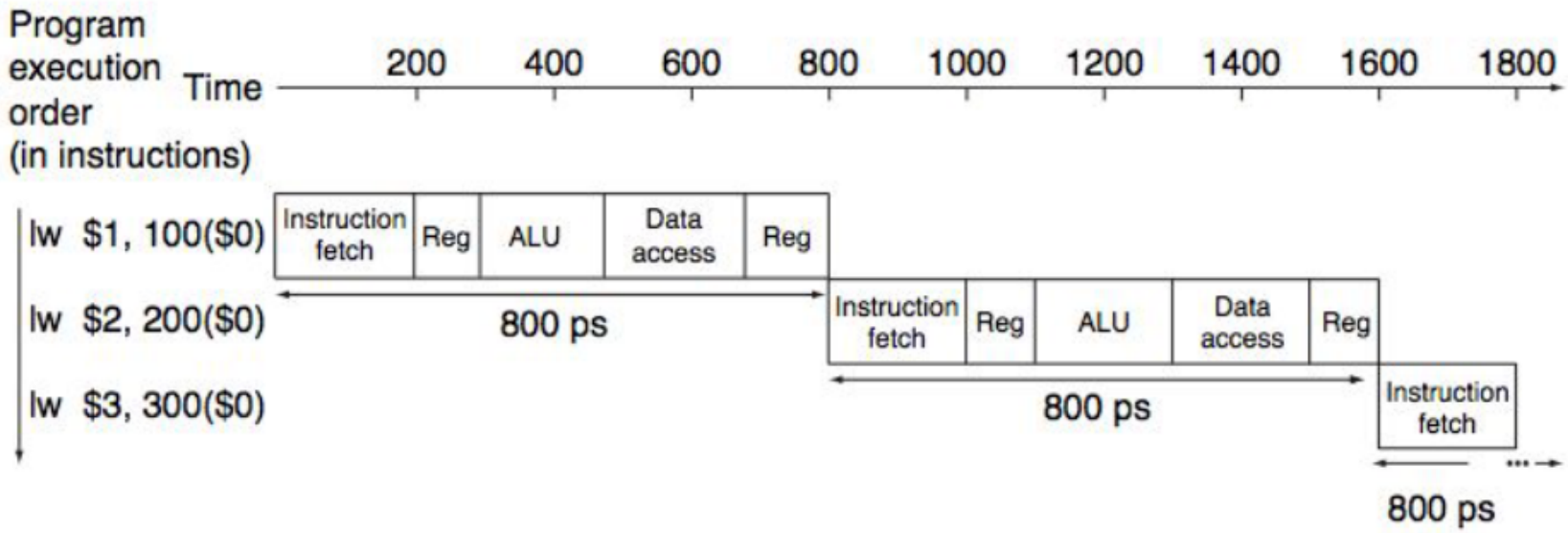

Figura: Execução de instruções no ciclo longo. Fonte: PH 4.5

$T_c = 2400ps$

Ciclo longo *vs* segmentação

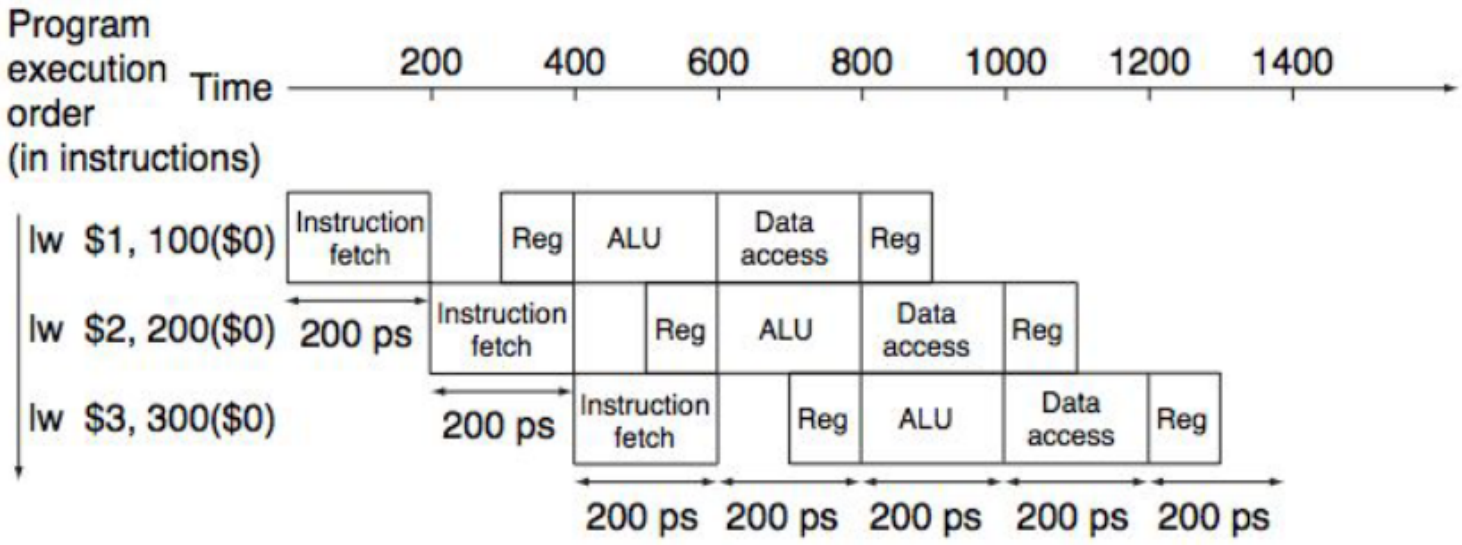

Figura: Execução de instruções com segmentação. Fonte: PH 4.5

$$T_c = 1400ps$$

Ciclo longo *vs* segmentação: desempenho

Vamos adicionar 1.000.000 instruções

- Ciclo longo: 800*ps*
- Segmentado: 200*ps* **depois do *pipeline* cheio!**

Ciclo longo *vs* segmentação: desempenho

Vamos adicionar 1.000.000 instruções

- Ciclo longo: 800*ps*
- Segmentado: 200*ps* **depois do *pipeline* cheio!**
- Ciclo longo: $1.000.000 \times 800 + 2.400$
- Segmentado: $1.000.000 \times 200 + 1.400$

Ciclo longo *vs* segmentação: desempenho

Vamos adicionar 1.000.000 instruções

- Ciclo longo: $800ps$
- Segmentado: $200ps$ **depois do *pipeline* cheio!**
- Ciclo longo: $1.000.000 \times 800 + 2.400$
- Segmentado: $1.000.000 \times 200 + 1.400$

$$\frac{800.002.400ps}{200.001.400ps} \approx \frac{800ps}{200ps} \approx 4$$

Ciclo longo *vs* segmentação: desempenho

Pipeline, ou segmentação, aumenta a **vazão**, mas não diminui o tempo de uma instrução individual!

Ciclo longo *vs* segmentação: desempenho

Pipeline, ou segmentação, aumenta a **vazão**, mas não diminui o tempo de uma instrução individual!

Como um programa executa bilhões de instruções, vazão é a métrica que queremos melhorar!

Ciclo longo *vs* segmentação: vazão

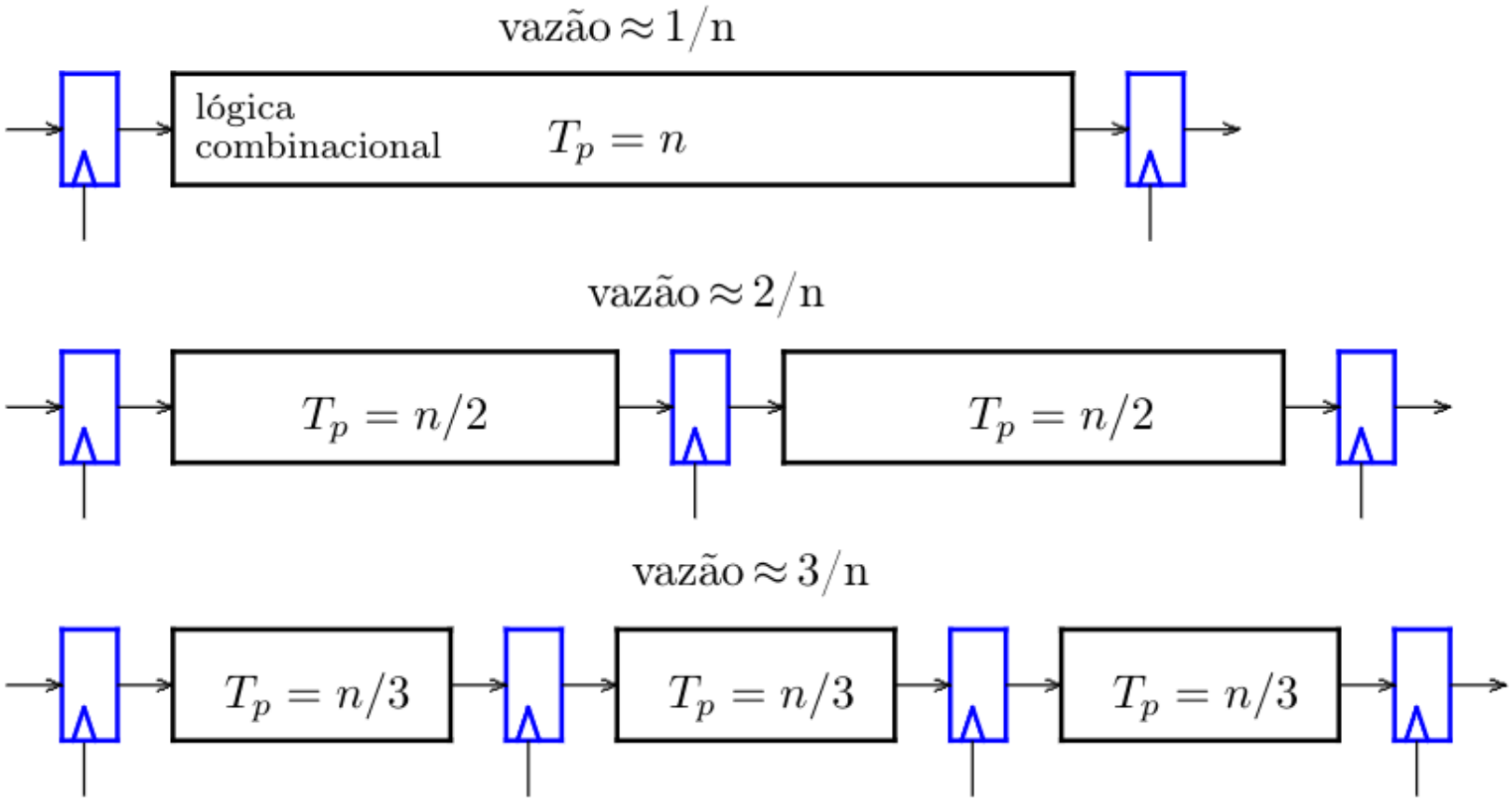

Figura: Circuito combinacional segmentado. Fonte: RH

Sumário

Processador segmentado

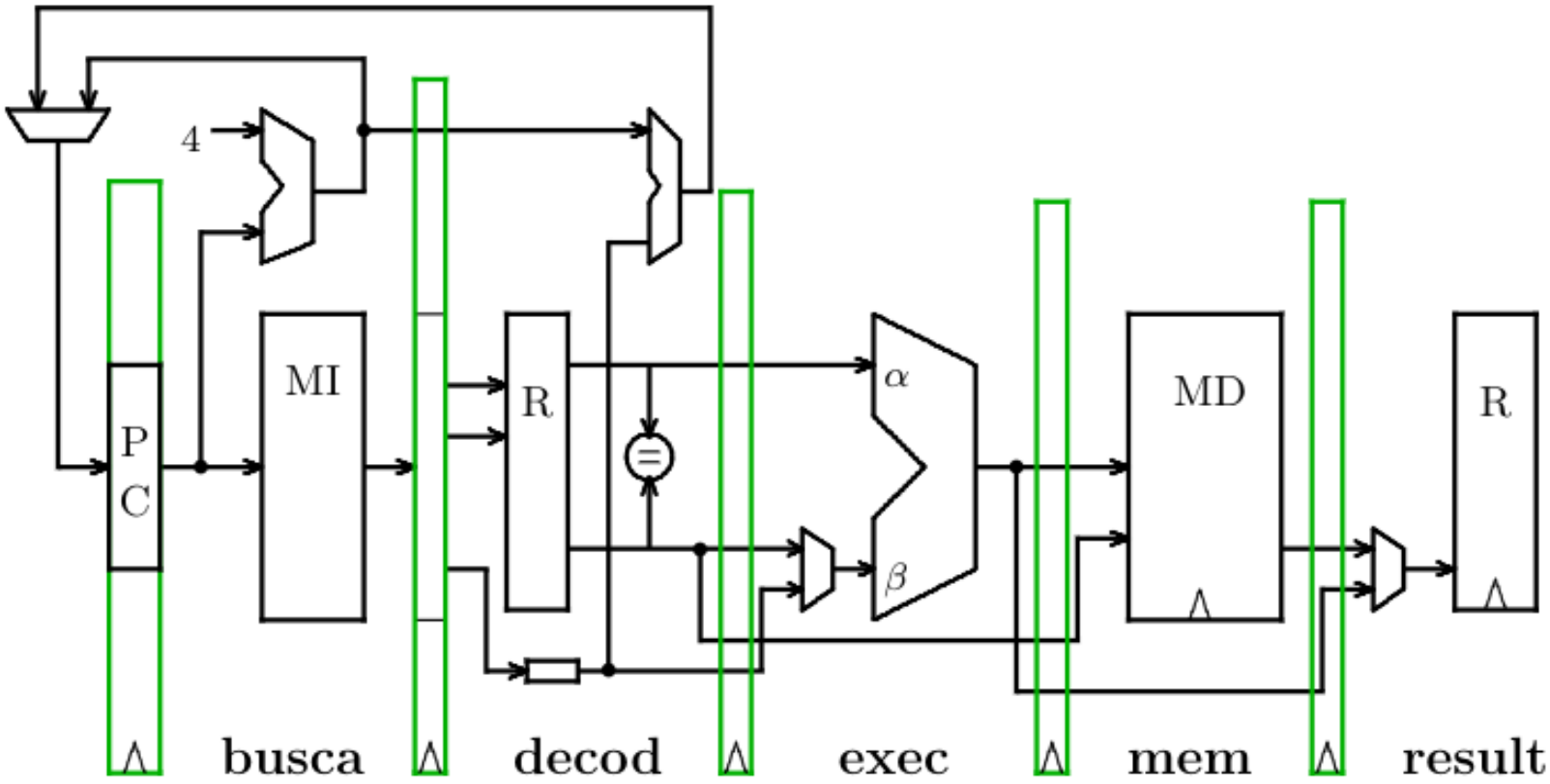

Figura: Processador segmentado em 5 estágios. Fonte: RH

Processador segmentado

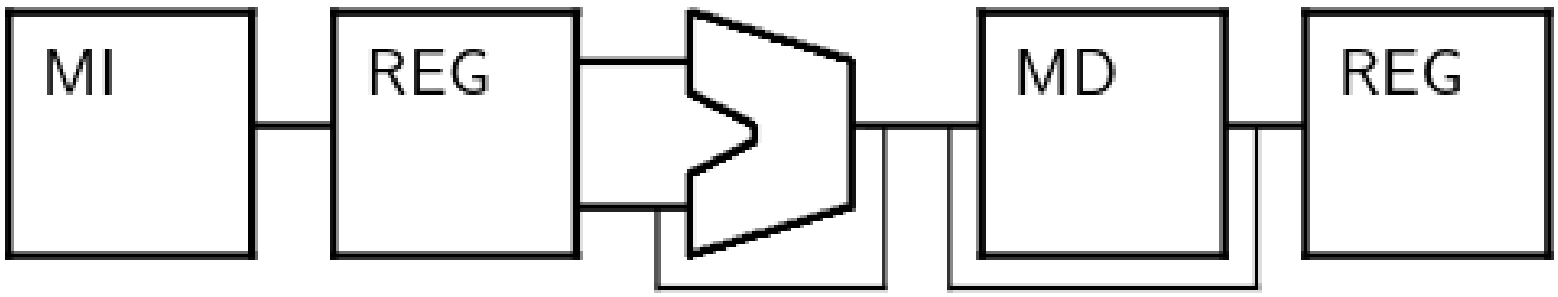

Figura: Representação do processador segmentado. Fonte: RH

Segmentação

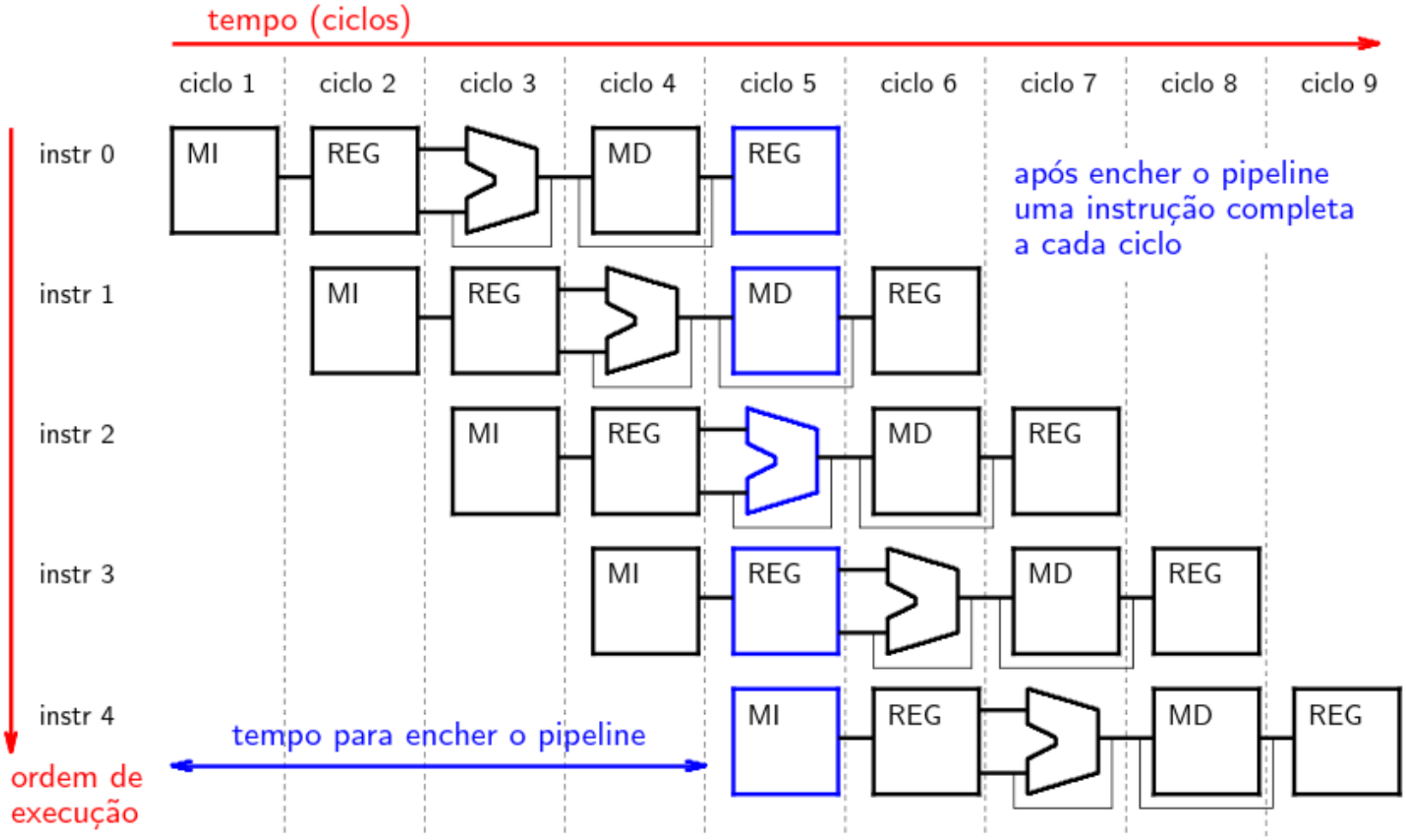

Figura: Representação da execução no *pipeline*. Fonte: RH

Segmentação

O que facilita a segmentação:

- instruções de mesmo tamanho
- poucos formatos de instrução
- poucas instruções de acesso à memória
- operandos alinhados na memória
- instrução escreve um resultado no final

Segmentação

O que dificulta a segmentação:

- riscos estruturais: uso do mesmo recurso
- riscos de controle: o que acontece nos desvios?
- riscos de dados: dependência entre instruções

Segmentação

O que dificulta a segmentação:

- riscos estruturais: uso do mesmo recurso
- riscos de controle: o que acontece nos desvios?
- riscos de dados: dependência entre instruções

Riscos sempre podem ser resolvidos com espera (*stall*)!

Segmentação

O que dificulta a segmentação:

- riscos estruturais: uso do mesmo recurso
- riscos de controle: o que acontece nos desvios?
- riscos de dados: dependência entre instruções

Riscos sempre podem ser resolvidos com espera (*stall*)! Controle deve detectar o risco e fazer algo para resolver

Sumário

Riscos estruturais

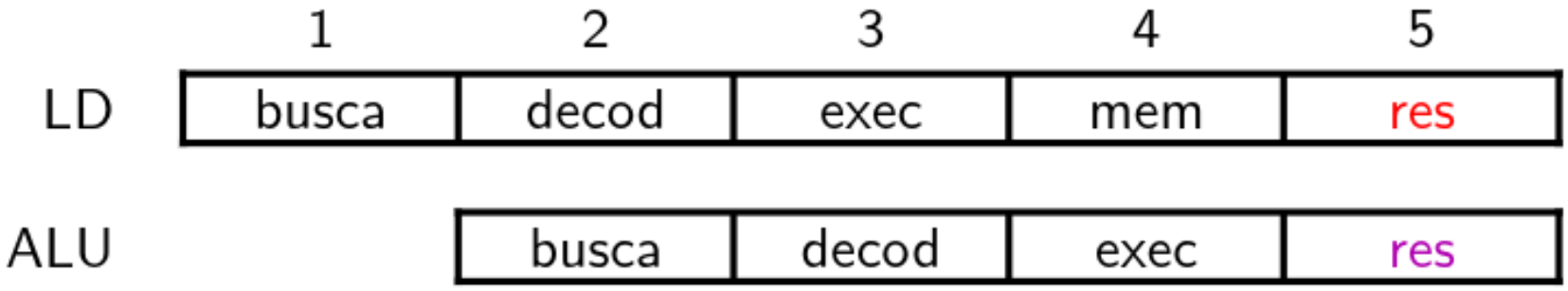

Figura: Estágios do *load* e de operações da ULA. Fonte: RH

Riscos estruturais: solução 1

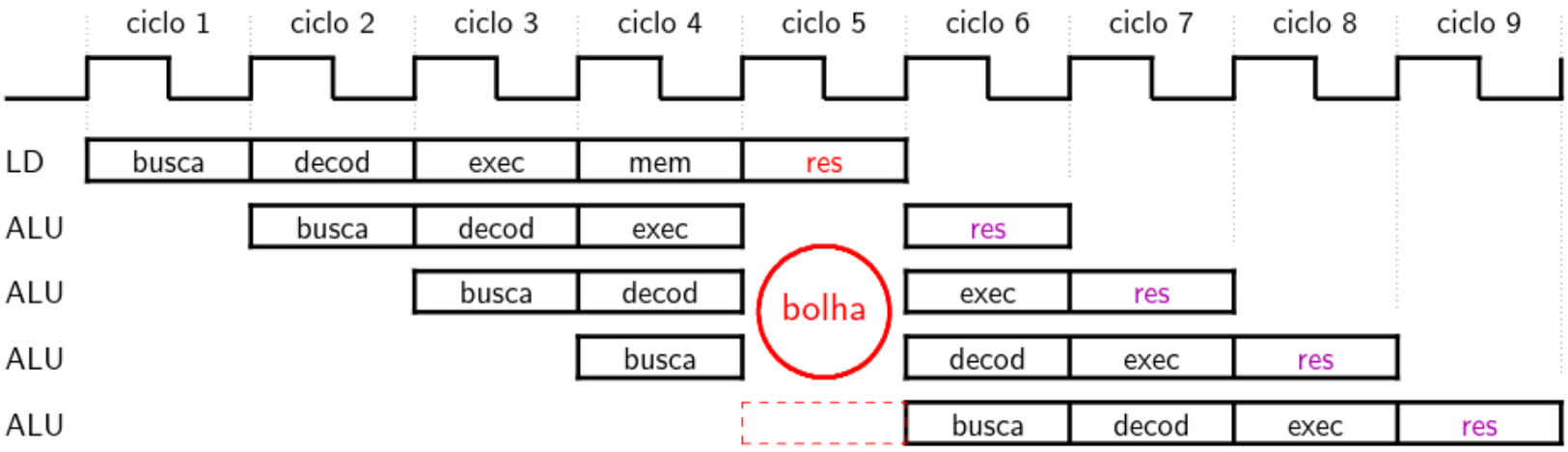

Figura: Inserção de bolha para resolver riscos estruturais.
Fonte: RH

Bolhas podem complicar o controle e, nesse exemplo, faz com que não busquemos uma instrução no ciclo 5; reduzem o paralelismo!

Riscos estruturais: solução 2

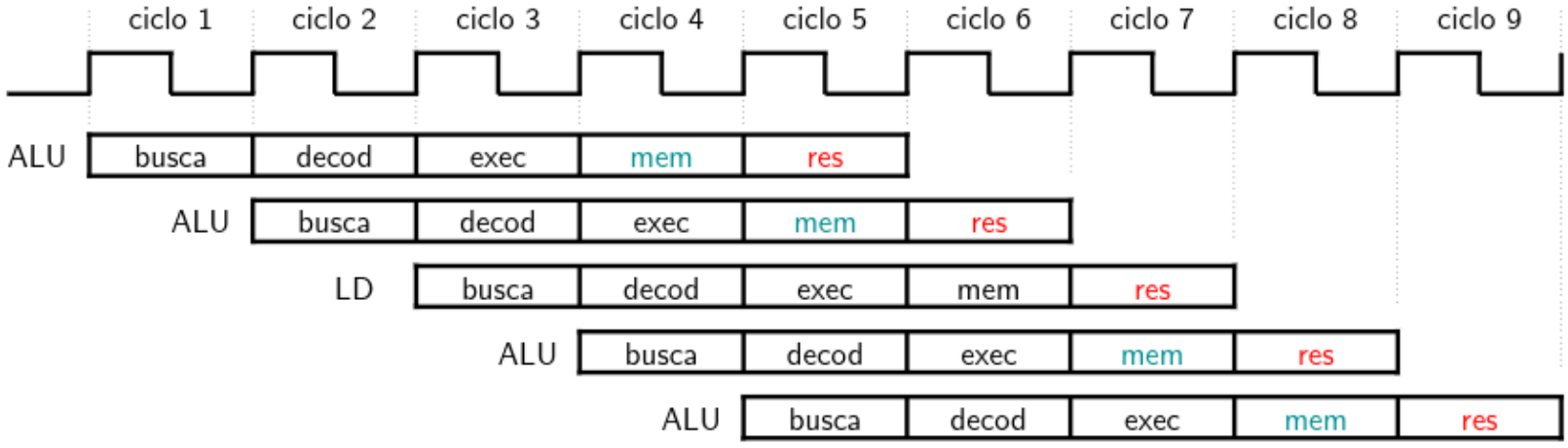

Figura: Atraso para resolver riscos estruturais. Fonte: RH

Atrasar a escrita: ULA também acessa a escrita dos registradores no quinto estágio

Riscos estruturais: solução 2

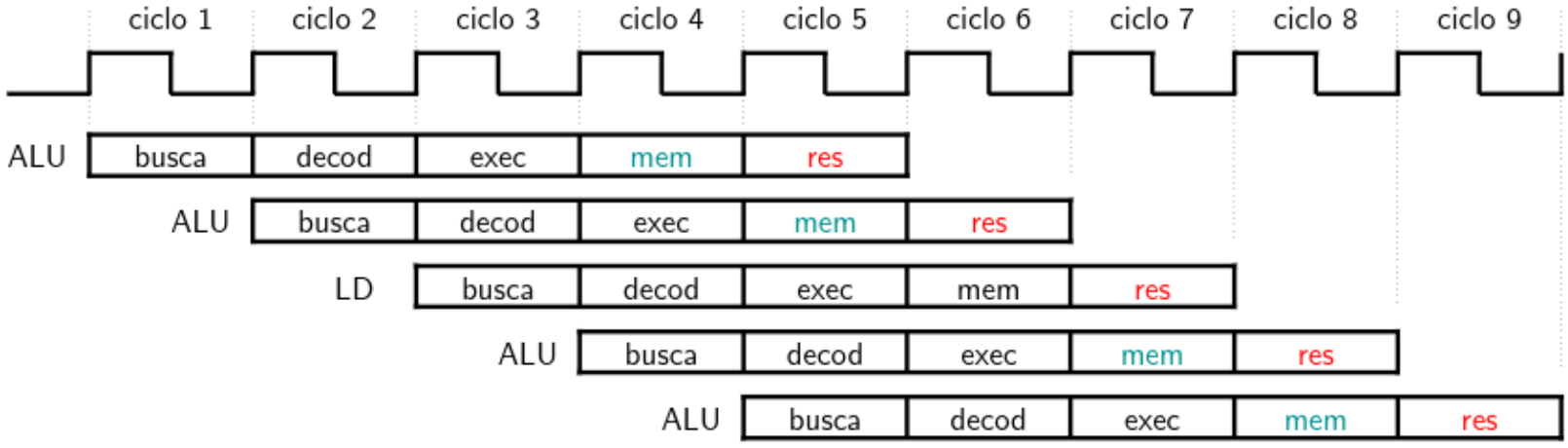

Figura: Atraso para resolver riscos estruturais. Fonte: RH

Atrasar a escrita: ULA também acessa a escrita dos registradores no quinto estágio

Estágio de memória pode ser projetado para não fazer nada nesse caso

INSTITUTO FEDERAL
Paraná
Campus Pinhais

Riscos estruturais: outro exemplo

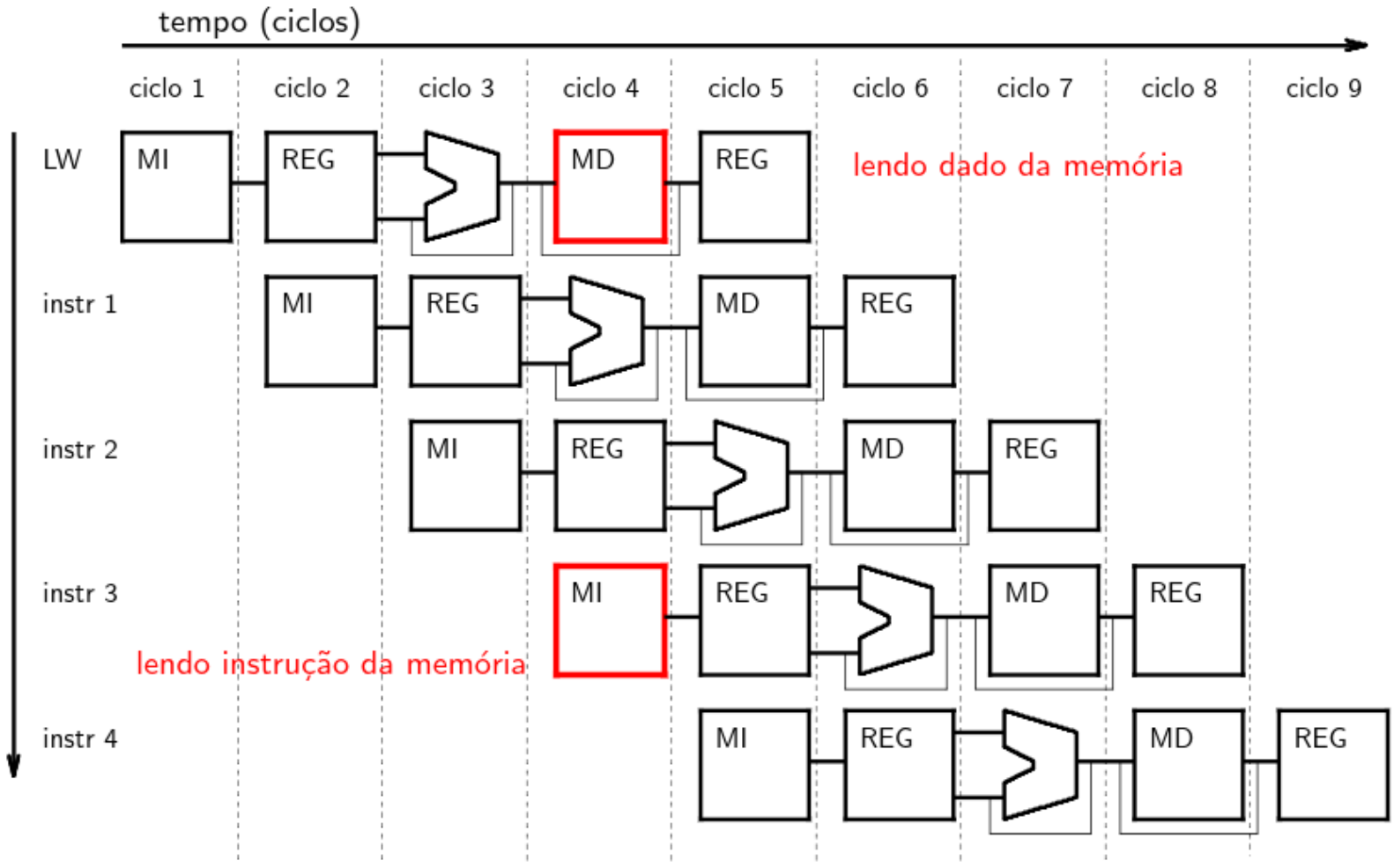

Figura: Risco estrutural no acesso à memória única. Fonte: RH

Sumário

Riscos de dados

```
add r1, r6, r7
sub r3, r2, r1
```

Riscos de dados

```
add r1, r6, r7
sub r3, r2, r1
```

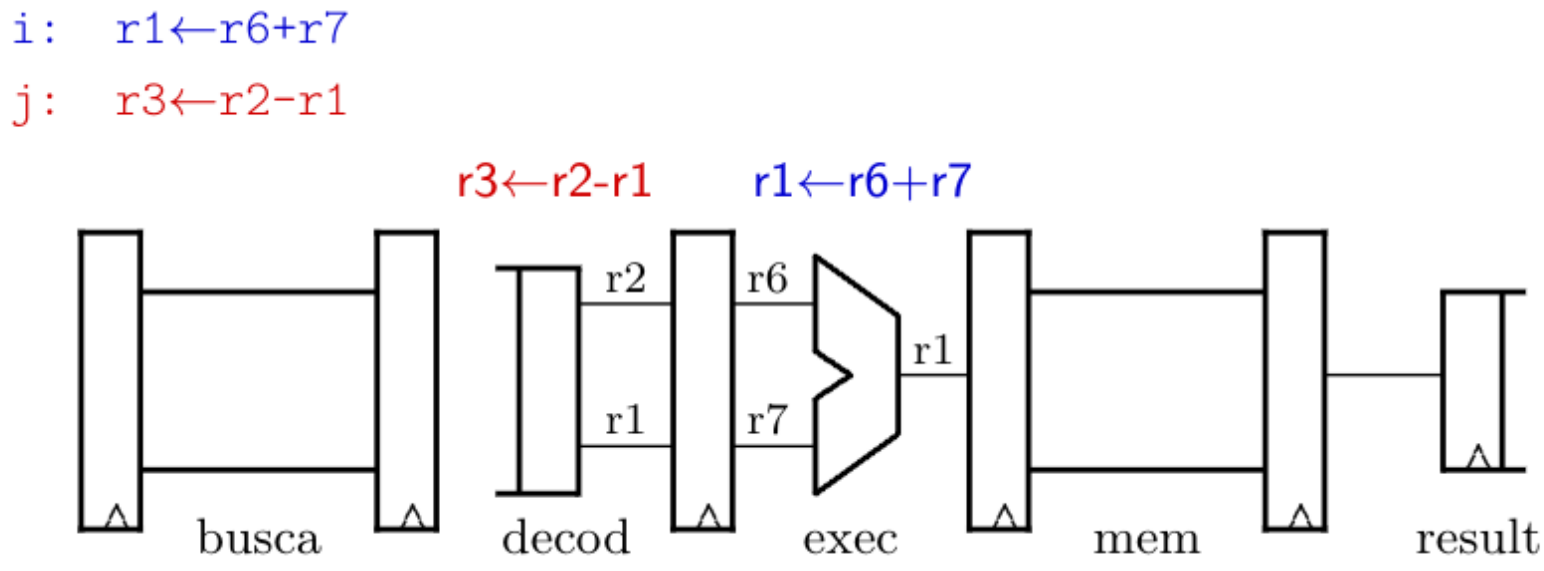

Figura: Dependência de dados. Fonte: RH

Riscos de dados: solução com nop

```
i:  r1←r6+r7
j:  nop
k:  nop
l:  r3←r2-r1
```

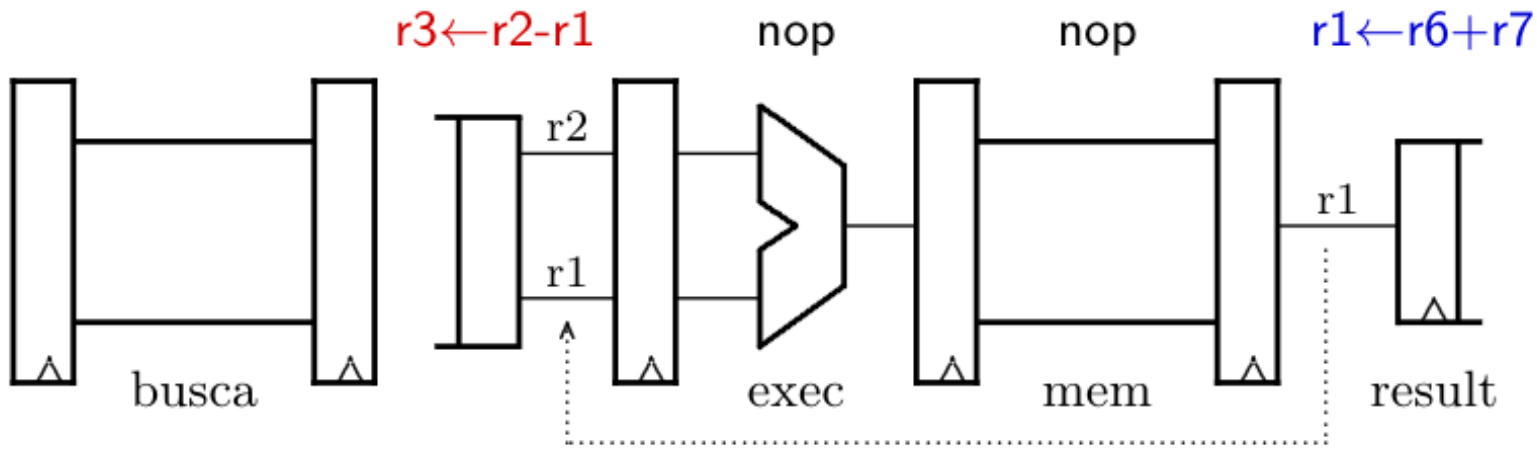

Figura: Dependência de dados resolvida com dois nop. Fonte: RH

Riscos de dados: solução com nop

```
# riscos por resolver
add  r5,  r6,  r7
sub  r8,  r9,  r5 # r5
addi r2,  r8,  9  # r8
xor  r10, r8,  r5 # r8,r5
add  r5,  r10, r2 # r10,r2
```

```
# riscos resolvidos
add  r5,  r6,  r7
nop
nop
sub  r8,  r9,  r5
nop
nop
addi r2,  r8,  9
xor  r10, r8,  r5
nop
add  r5,  r1 , r2
```

Figura: Dependência de dados e a vazão. Fonte: RH

Diminui a vazão em $\approx 50\%$

INSTITUTO FEDERAL
Paraná
Campus Pinhais

Riscos de dados: solução com adiantamentos

```
i:  r2←r8+r9
j:  r1←r6+r7
k:  r3←r2-r1
```

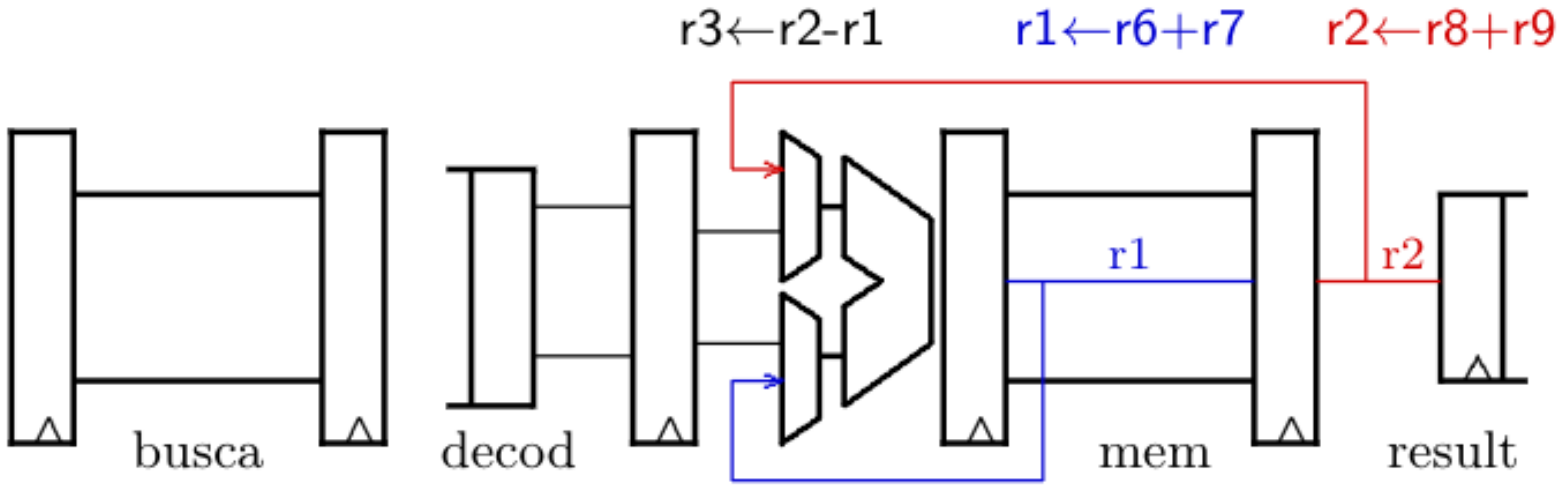

Figura: Dependência de dados resolvida com adiantamentos.
Fonte: RH

Riscos de dados: *loads*

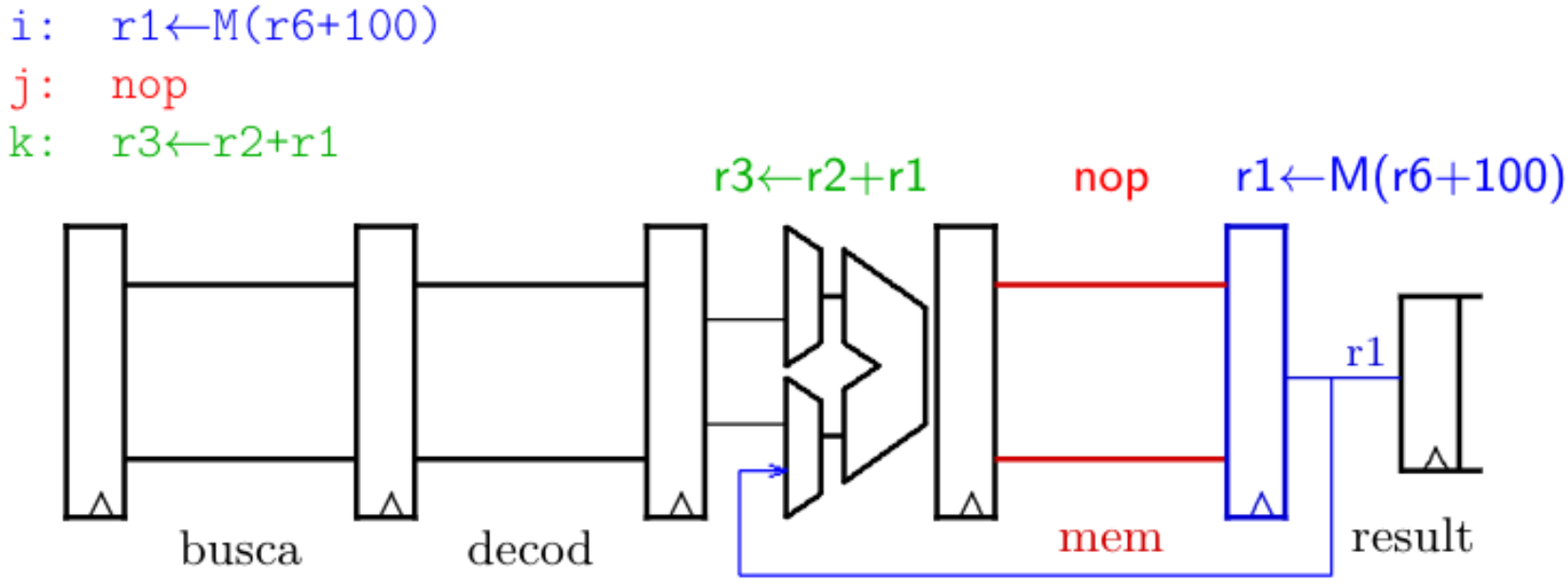

Figura: Dependência de dados nas operações de *load*. Fonte: RH

Riscos de dados: reordenar instruções

```
for: lw   r5, 0(r3)
     add  r4, r4, r5
     addi r3, r3, 4
     addi r1, r1, 4
     bne  r1, r2. for
fim:
```

```
for: lw   r5, 0(r3)
     nop
     add  r4, r4, r5
     addi r3, r3, 4
     addi r1, r1, 4
     nop
     nop
     bne  r1, r2. for
fim:
```

Figura: Dependência de dados resolvidas com `nop`. Fonte: RH

Podemos (programadores e compiladores) reordenar instruções para evitar os *delay-slots*

INSTITUTO FEDERAL
Paraná
Campus Pinhais

Sumário

Riscos de controle

Causados por desvios: já existem instruções (depois do desvio) no *pipeline*, antes do processador saber se vai desviar ou não
Resolvidos com bolhas (*stall*).

Riscos de controle

Causados por desvios: já existem instruções (depois do desvio) no *pipeline*, antes do processador saber se vai desviar ou não
Resolvidos com bolhas (*stall*).

Podem ser amenizados com *branch prediction*

Sumário

Predição de desvios

Prevê algo. Se errar, precisa anular as instruções que foram previstas

Predição de desvios

Prevê algo. Se errar, precisa anular as instruções que foram previstas

Predição estática: assumir que um desvio sempre será tomado ou que ele nunca será tomado.

Predição de desvios

Prevê algo. Se errar, precisa anular as instruções que foram previstas

Predição estática: assumir que um desvio sempre será tomado ou que ele nunca será tomado.

Em alguns projetos de processadores simples, a instrução no *branch delay-slot* é sempre executada: o compilador é **obrigado** a preencher com uma instrução útil ou então acrescentar um `nop` ao código

Predição de desvios

Predição dinâmica: formas mais elaboradas que olham para a execução atual

Predição de desvios

Predição dinâmica: formas mais elaboradas que olham para a execução atual

Exemplo: verificar se o desvio foi já foi tomado antes: olhar para o endereço da instrução. Laços repetem o mesmo desvio várias vezes.

Predição de desvios

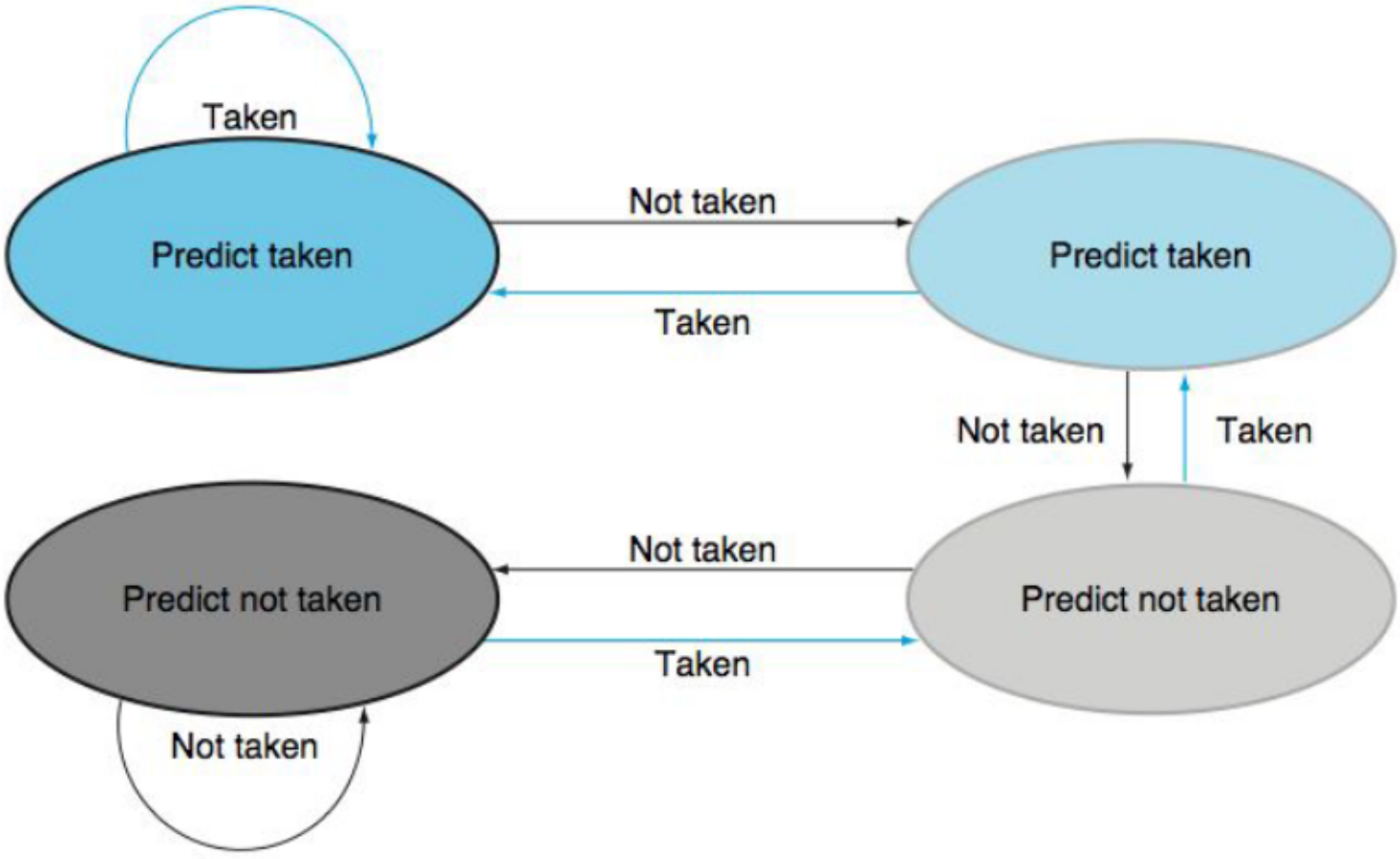

Figura: Esquema de predição usando 2 bits. Fonte: PH 4.8