

MC542

Organização de Computadores Teoria e Prática

2007

Prof. Paulo Cesar Centoducatte

ducatte@ic.unicamp.br

www.ic.unicamp.br/~ducatte

MC542

Arquitetura de Computadores

Micro-Arquitetura Pipeline Exceções

"DDCA" - (Capítulo 7)

"COD" - (Capítulo #)

Sumário

- **Exceções**
- **Micro-Arquiteturas Avançadas**
 - Deep Pipelining
 - Branch Prediction
 - Superscalar Processors
 - Out of Order Processors
 - Register Renaming
 - SIMD
 - Multithreading
 - Multiprocessors

Exceções

Revisão:

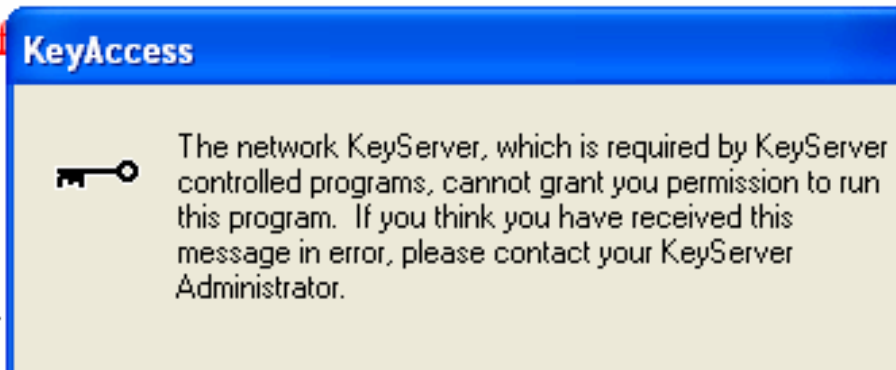
- Chamada de procedimento não “prevista” para tratamento de uma exceção
- Causado por:
 - Hardware, também chamado de *interrupção* (keyboard, ...)
 - Software, também chamado de *traps* (instrução indefinida, ...)
- Quando uma exceção ocorre, o processador:
 - Registra a causa da exceção (Cause register)
 - Salta para a rotina de tratamento da exceção no endereço de instrução 0x80000180
 - Retorna ao programa (EPC register)

Exemplo de Exceções

sequential circuits.¶

Can we design a spiff

Figure 2.11 shows a
inputs, A and B, and on
box indicates that it is
this case, the function is



Visio.exe - Application Error



The exception unknown software exception (0xc06d007e) occurred in the application at location 0x7c81eb33.

OK

Harris a
r — 26 A
C H A P

words, we say the output Y is a function of the two inputs A and B where
the function performed is A OR B.¶

The *implementation* of the combinational circuit is independent of its
functionality. Figure 2.1 and Figure 2.2 show two possible implementa-

Registradores de Exceção

- Não faz parte do register file.
 - Cause
 - » Registra a causa da exceção
 - » **Coprocessor 0 register 13**
 - **EPC (Exception PC)**
 - » Registra o PC onde ocorreu a exceção
 - » **Coprocessor 0 register 14**
- Move from Coprocessor 0
 - `mfc0 $t0, EPC`
 - Move o conteúdo de **EPC** para `$t0`

Causa de Exceções

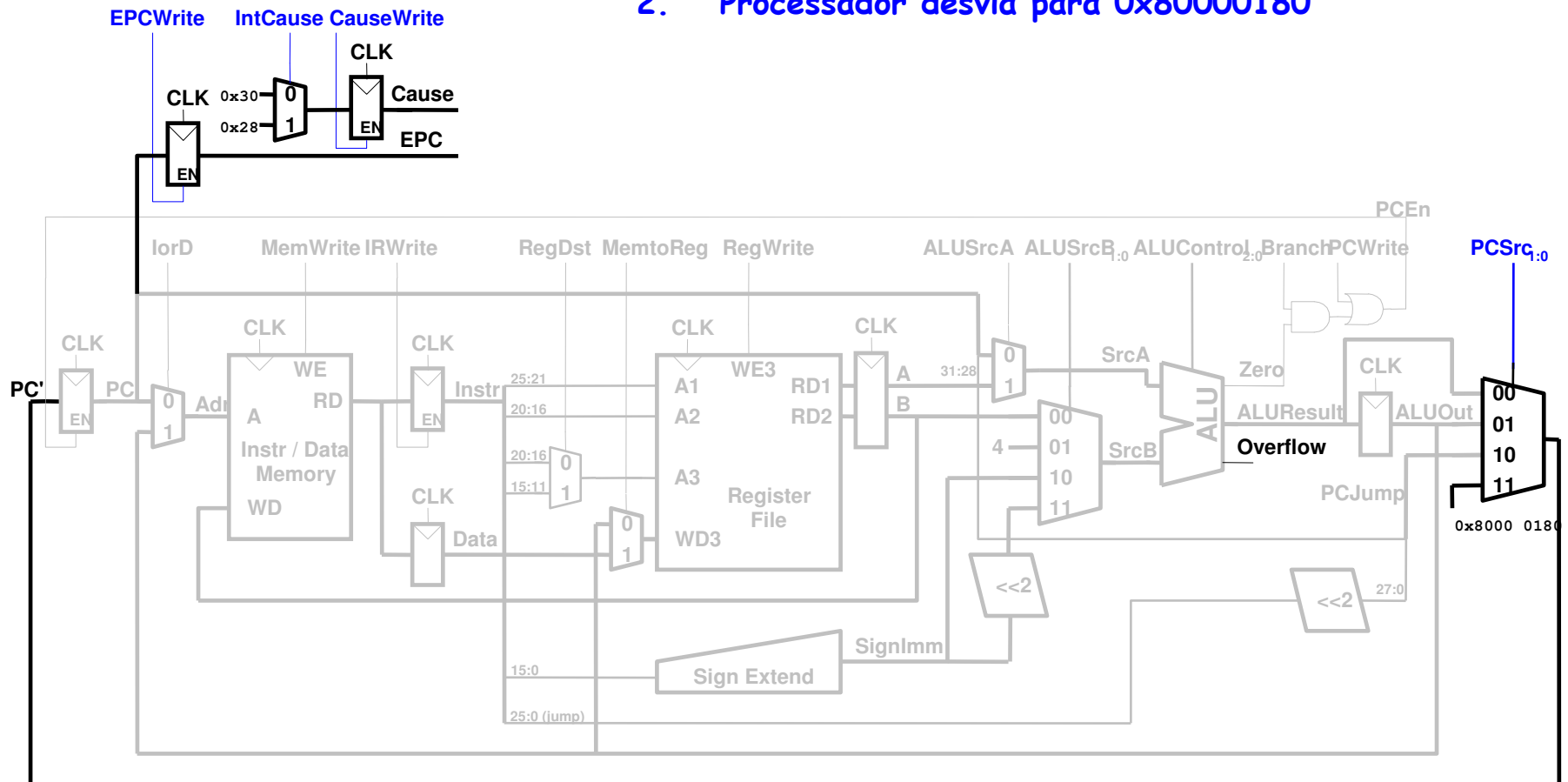
Exception	Cause
Hardware Interrupt	0x00000000
System Call	0x00000020
Breakpoint / Divide by 0	0x00000024
Undefined Instruction	0x00000028
Arithmetic Overflow	0x00000030

Modificar o MIPS multiciclos para tratar as duas últimas exceções.

1. O Processador salva a causa e o PC em Cause e EPC
2. Processador desvia para o **exception handler** (0x80000180)
3. Exception handler:
 - Salva os registradores na pilha
 - Lê o registrador Cause
`mfc0 Cause, $t0`
 - Trata a exceção
 - Restaura os registradores
 - Retorna ao programa
`mfc0 EPC, $k0`
`jr $k0`

Exceções: passos 1 e 2

1. O Processador salva a causa e o PC em Cause e EPC
2. Processador desvia para 0x8000180



Exceções: passo 3 (mfc0)

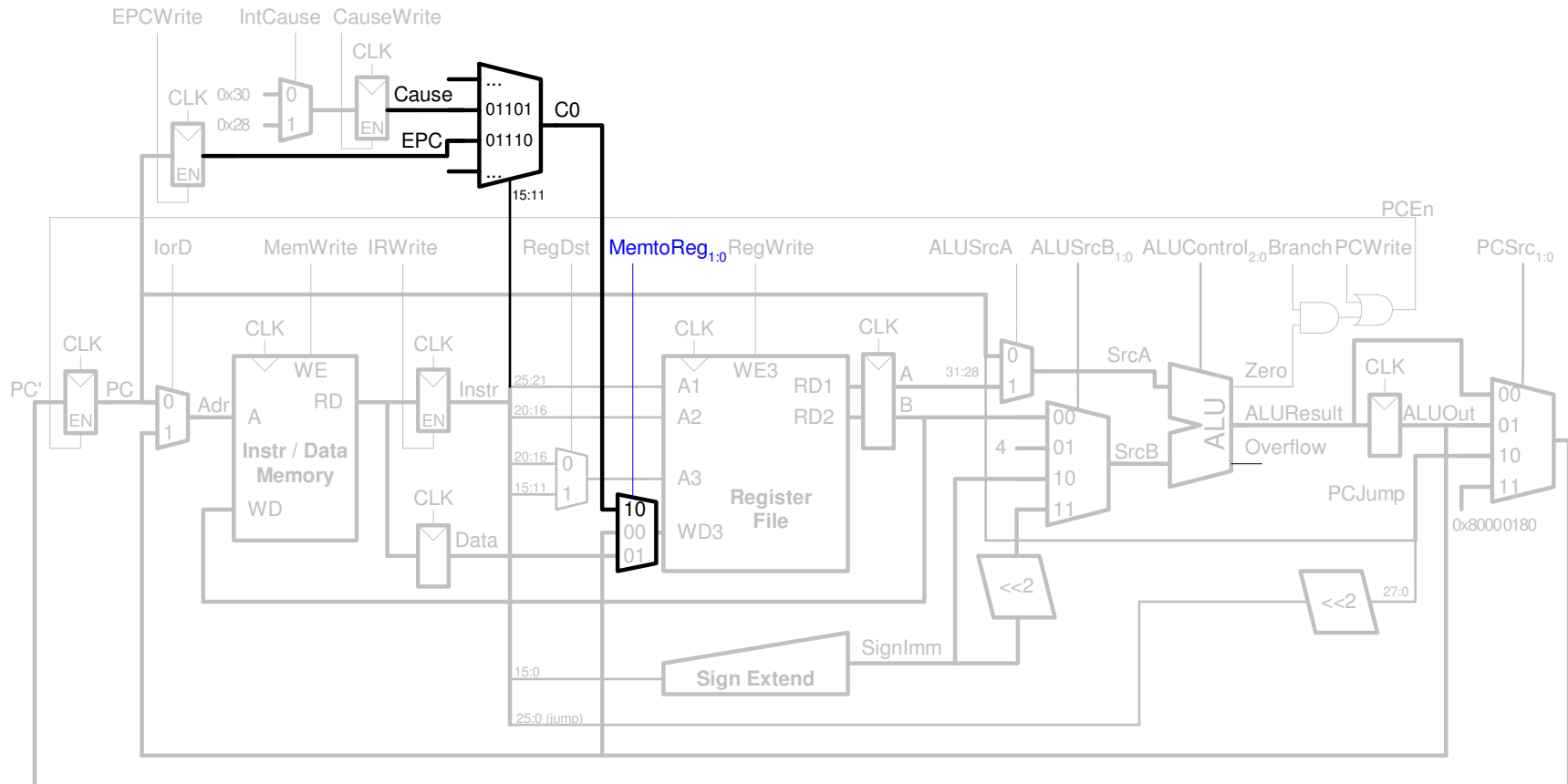
Exception handler

- `mfc0 $t0, Cause`

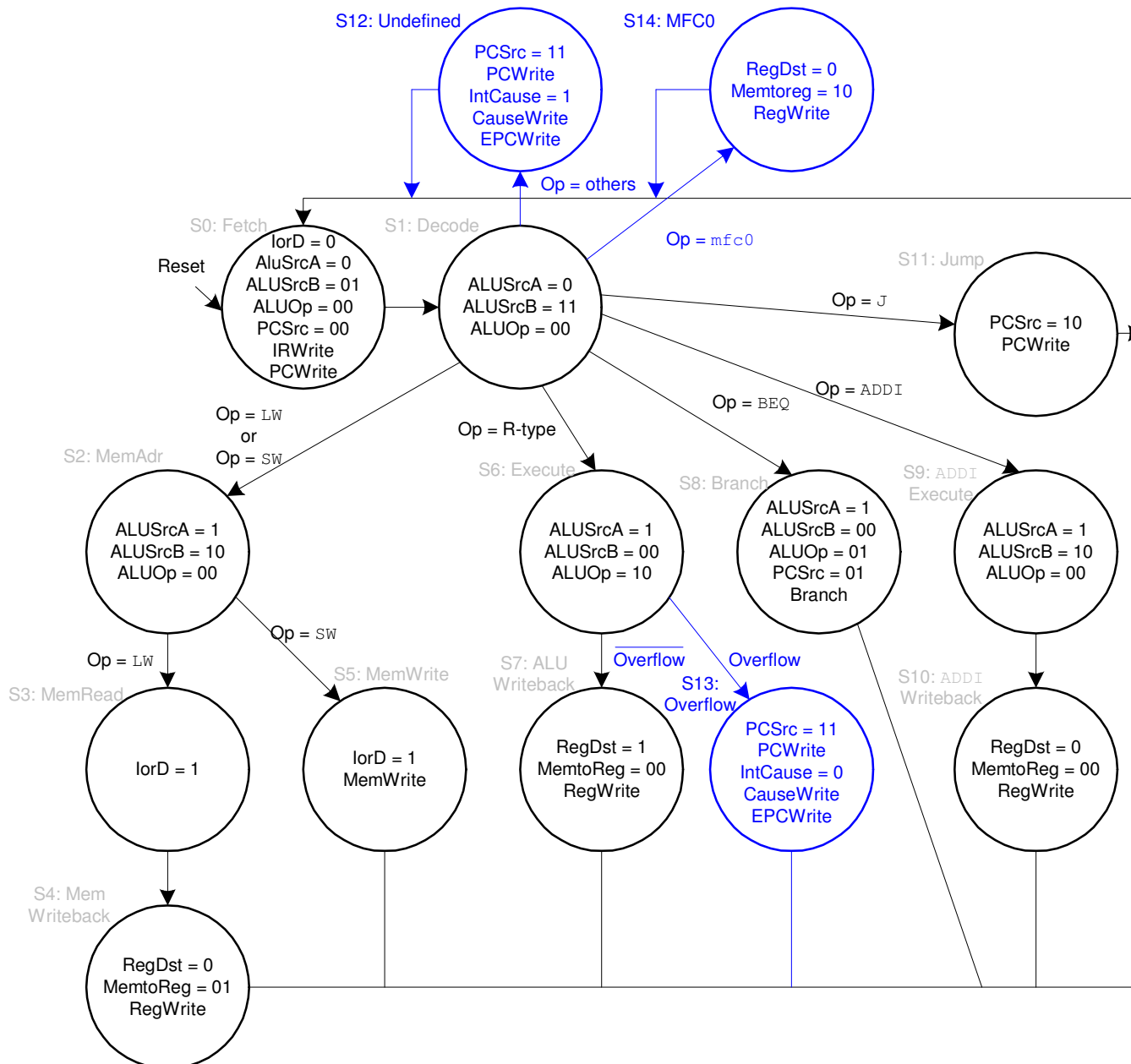
mfc0

<code>op</code>	<code>00000</code>	<code>\$t0 (8)</code>	<code>Cause (13)</code>	<code>000000000000</code>
31:26	25:21	20:16	15:11	10:0

Exceções: passo 3 (mfc0)



Exceções: FSM de Controle



Micro-Arquiteturas Avançadas

- Deep Pipelining
- Branch Prediction
- Superscalar Processors
- Out of Order Processors
- Register Renaming
- SIMD
- Multithreading
- Multiprocessors

Deep Pipelining

- Tipicamente 10 a 20 estágios
- O Número de estágios é limitado por:
 - Pipeline hazards
 - Sequencing overhead
 - Cost

Branch Prediction

- Processador pipelined Ideal: $CPI = 1$
- Branch misprediction aumenta o CPI
- Static branch prediction:
 - Avalia a direção do branch (forward ou backward)
 - se backward: **predict taken**
 - Caso contrário: **predict not taken**
- Dynamic branch prediction:
 - Mantém histórico dos últimos (centenas) branches em um **branch target buffer** (**Branch History Table**) que mantém:
 - » Destino do Branch
 - » E se o branch foi **taken**

Branch Prediction: Exemplo

```
add  $s1, $0, $0      # sum = 0
add  $s0, $0, $0      # i   = 0
addi $t0, $0, 10      # $t0 = 10
for:
    beq $t0, $t0, done # if i == 10, branch
    add $s1, $s1, $s0  # sum = sum + i
    addi $s0, $s0, 1   # increment i
    j    for
    .....
```

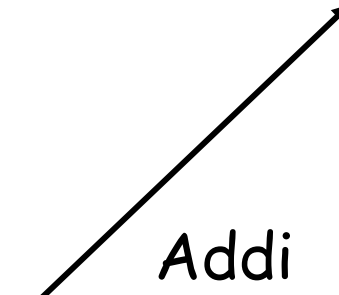
done:

1-Bit Branch Predictor

- Desempenho = $f(\text{precisão}, \text{custo do misprediction})$
- Branch History Table: Bits menos significativos do PC usados como índice de uma tabela de valores de 1 bit
 - Informa se o branch foi tomado ou não na última vez
 - Não há comparação do endereço (menos HW, mas pode não ser o branch correto)

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	1	0	0	1	1	1	0	0	0	0	0	1	0	1

Branch History Table

`0xaaa00028` 

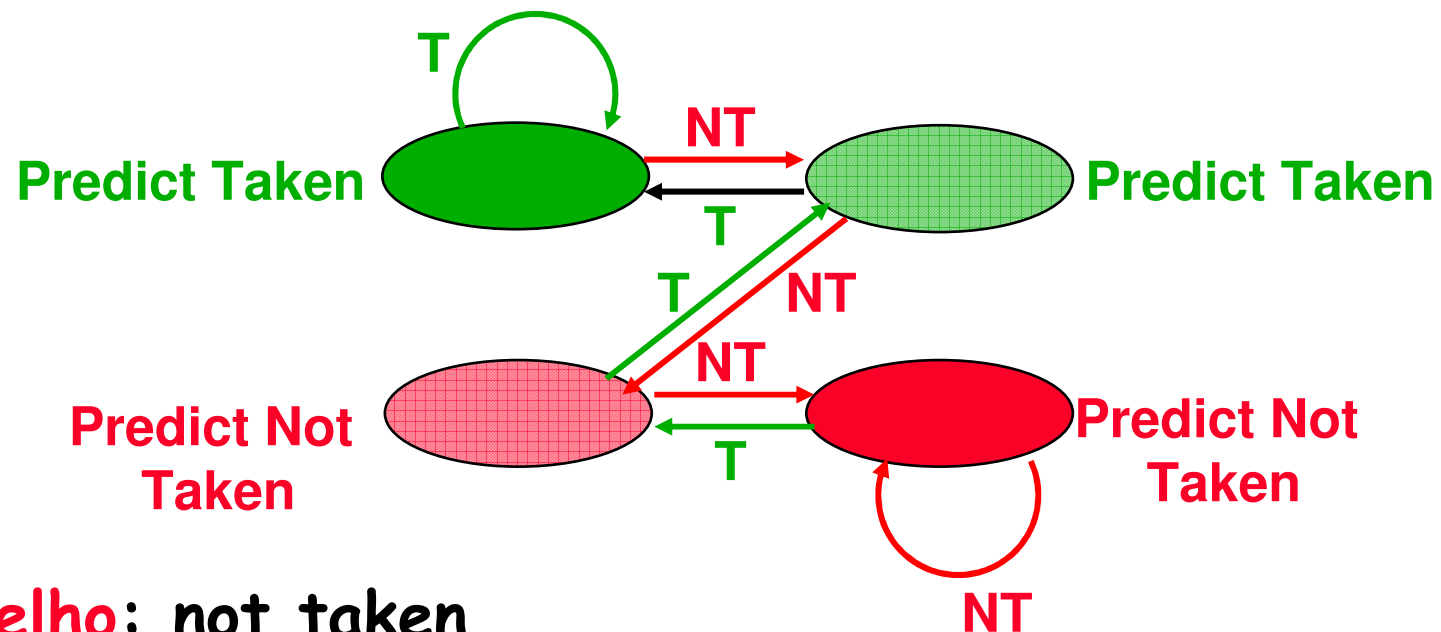
Addi	\$t0, \$s0, 10
Beq	\$t0, \$t0, 0xffff00002
Add	\$s1, \$s1, \$s0

1-Bit Branch Prediction

- Quando descobre que errou, atualiza a entrada correta, elimina as instruções erradas do pipeline e recomeça o fetch de **0xfff00002**
- Problema: em um loop, 1-bit BHT irá causar **2 mispredictions** (em média nos loops - na entrada e na saída):
 - No fim do loop quando ele termina
 - Na entrada do loop quando ele preve **exit** no lugar de looping
 - Em um loop com 10 iterações
 - » somente 80% de precisão
 - » mesmo que os **Taken** sejam 90% do tempo

2-Bit Branch Predictor

- Solução: esquema com 2-bit onde só há troca na previsão se houver duas **misprediction**:



- **Vermelho:** not taken
- **Verde:** taken
- *Adicionado uma Histerese (inércia)* para tomar a decisão

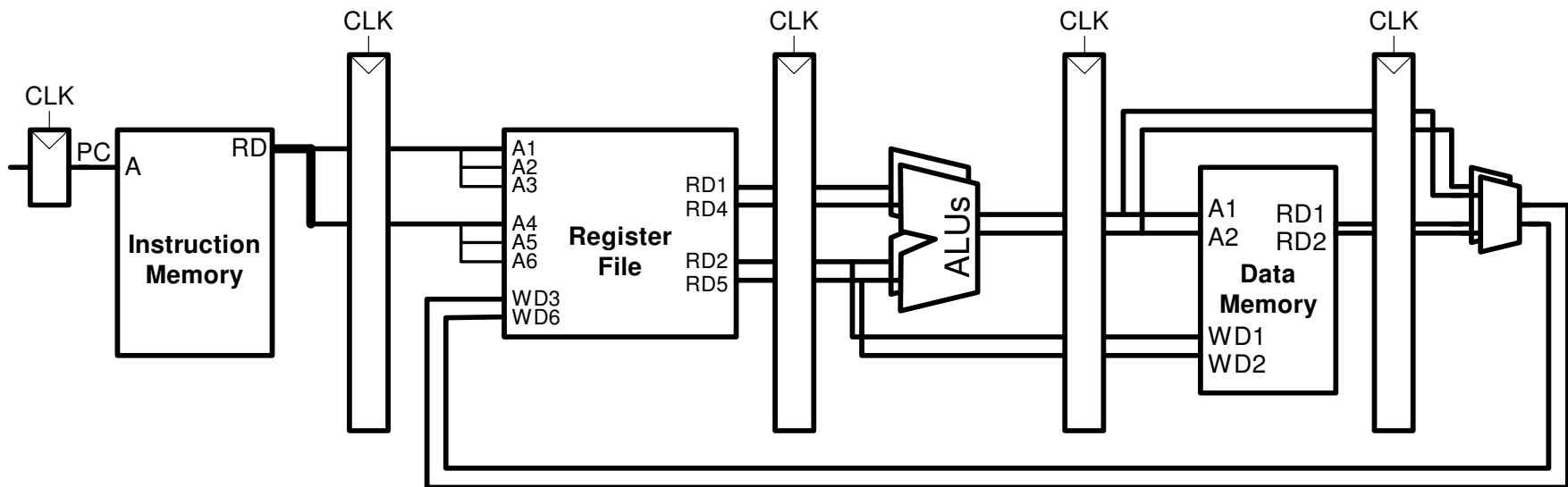
Branch Predictor

Vários outros esquemas:

- Correlating Branches
- Execução Predicada
- Tournament Predictors
- Branch Target Buffer (BTB)
- Return Addresses stack

Superscalar

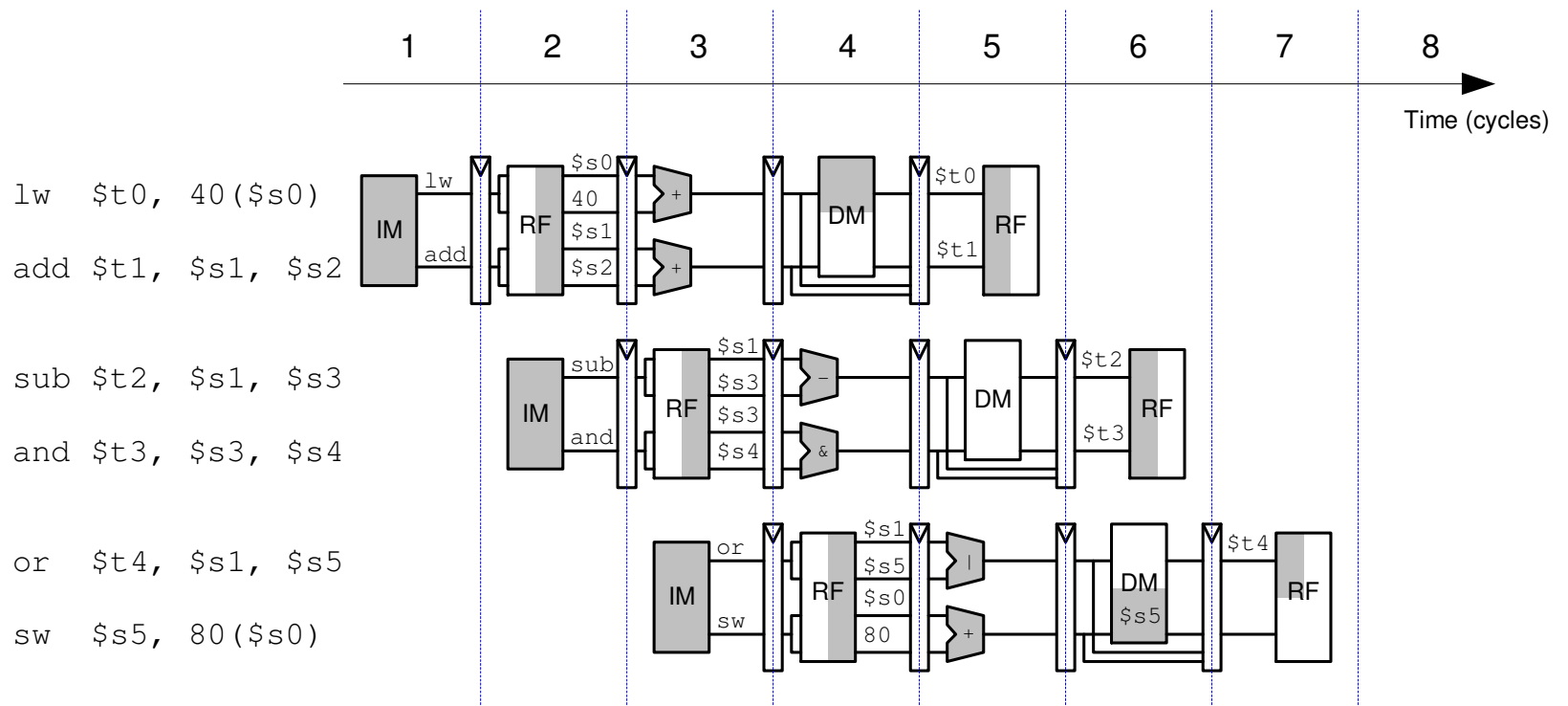
- Múltiplas cópias do datapath executando múltiplas instruções
- Dependências dificultam o despacho (Issue) de múltiplas instruções por vez



Superscalar: Exemplo

```
lw $t0, 40($s0)
add $t1, $s1, $s2
sub $t2, $s1, $s3
and $t3, $s3, $s4
or $t4, $s1, $s5
sw $s5, 80($s0)
```

IPC Ideal: 2
IPC: 2

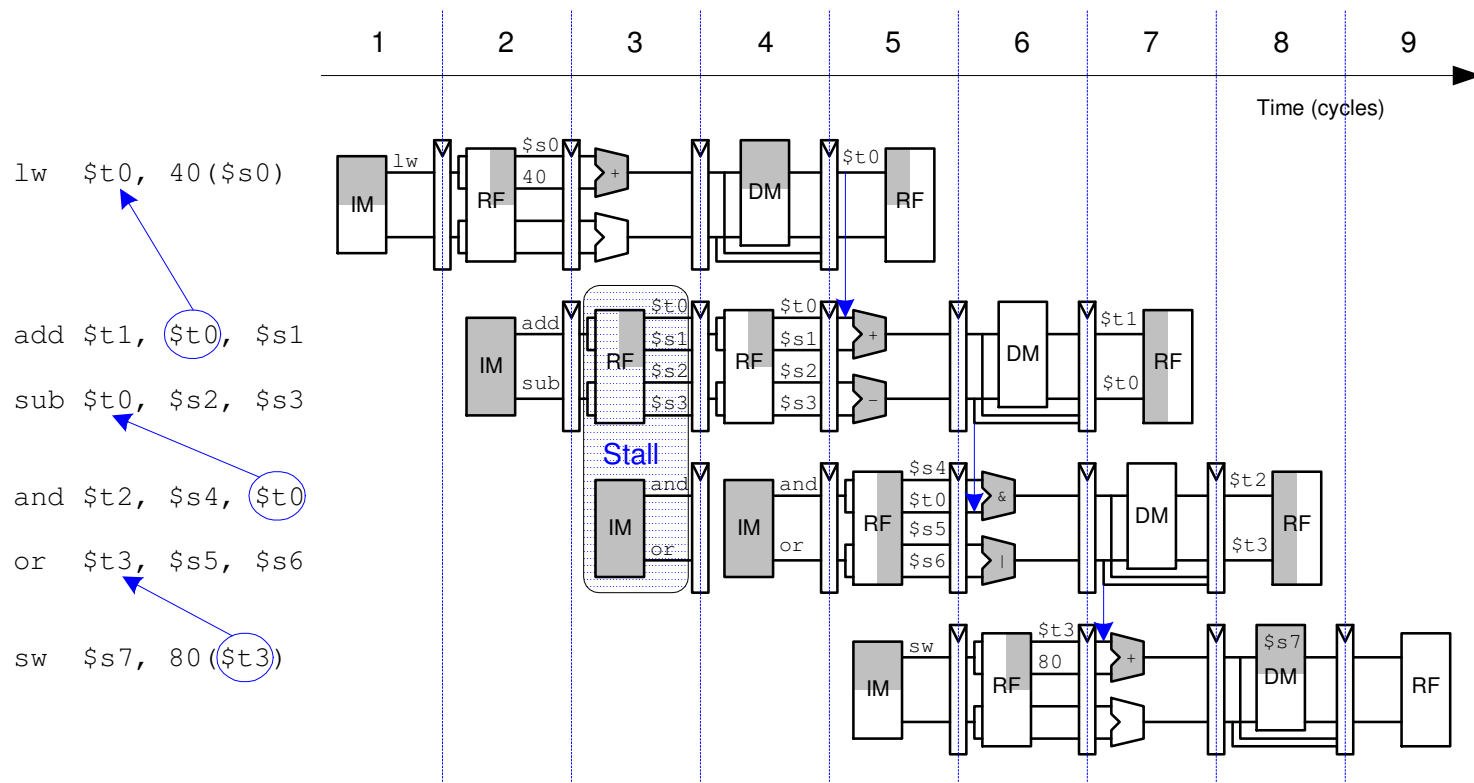


Superscalar Exemplo com Dependências

```
lw $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or $t3, $s5, $s6
sw $s7, 80($t3)
```

IPC Ideal: 2

IPC: $6/5 = 1.17$



Processador Out of Order

- Avaliar múltiplas instruções para despachar o máximo possível por vez
- Despachar instruções out of order se não tem dependências
- Dependências:
 - **RAW** (read after write): one instruction writes, and later instruction reads a register
 - **WAR** (write after read): one instruction reads, and a later instruction writes a register (also called an *antidependence*)
 - **WAW** (write after write): one instruction writes, and a later instruction writes a register (also called an *output dependence*)
- Instruction level parallelism: número de instruções que podem ser despachadas simultaneamente (na prática < 3)

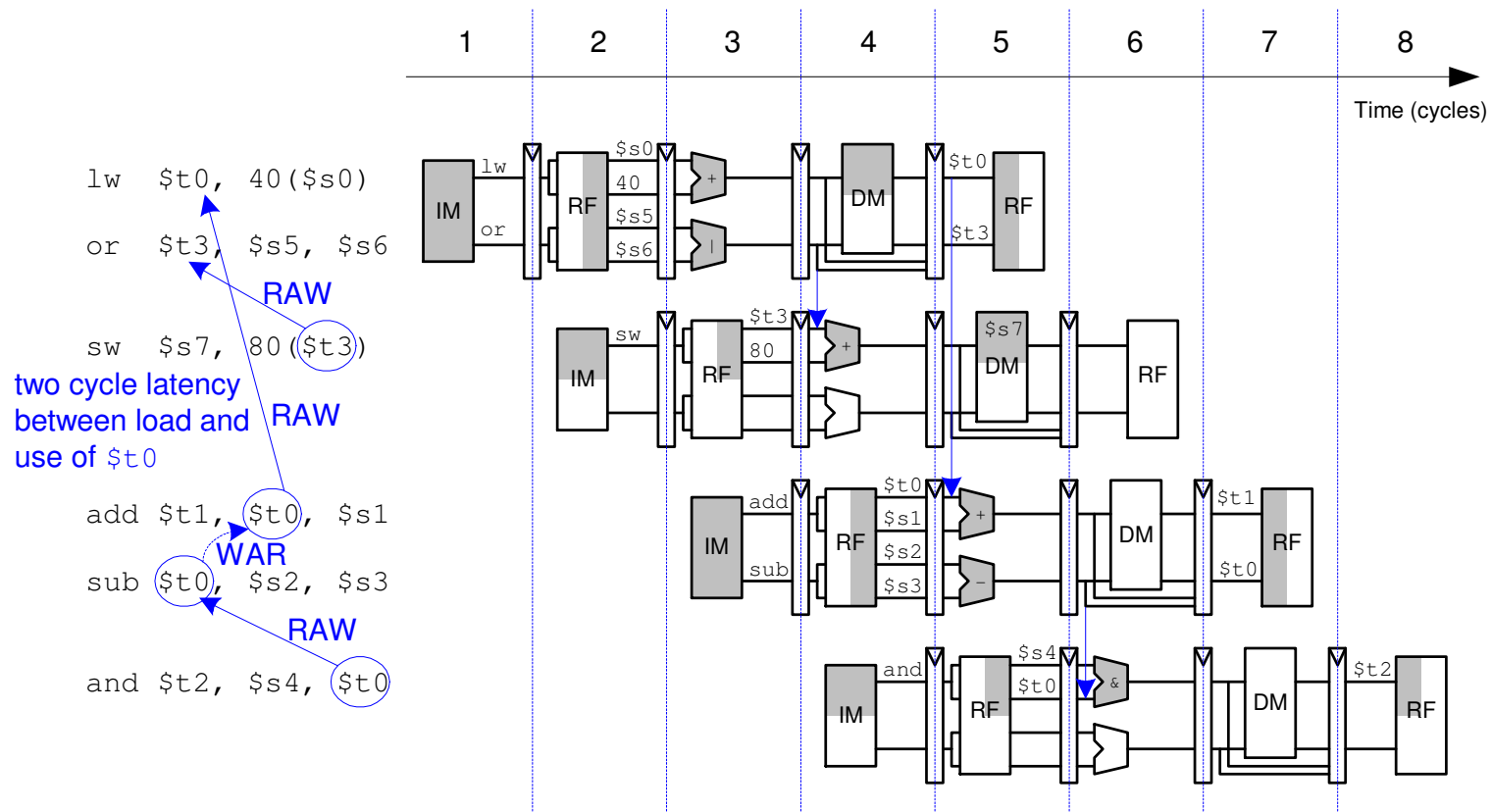
Processador Out of Order

- **Instruction level parallelism:** número de instruções que podem ser despachadas simultaneamente (na prática < 3)
- **Scoreboard:** tabela que mantém:
 - Instruções esperando para serem despachadas
 - Unidades funcionais disponíveis
 - Dependências
- **Tomasulo:**
 - Instruções esperando para serem despachadas
 - Unidades funcionais disponíveis
 - Dependências
 - Register Rename

Processor Out of Order: Exemplo

```
lw $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or $t3, $s5, $s6
sw $s7, 80($t3)
```

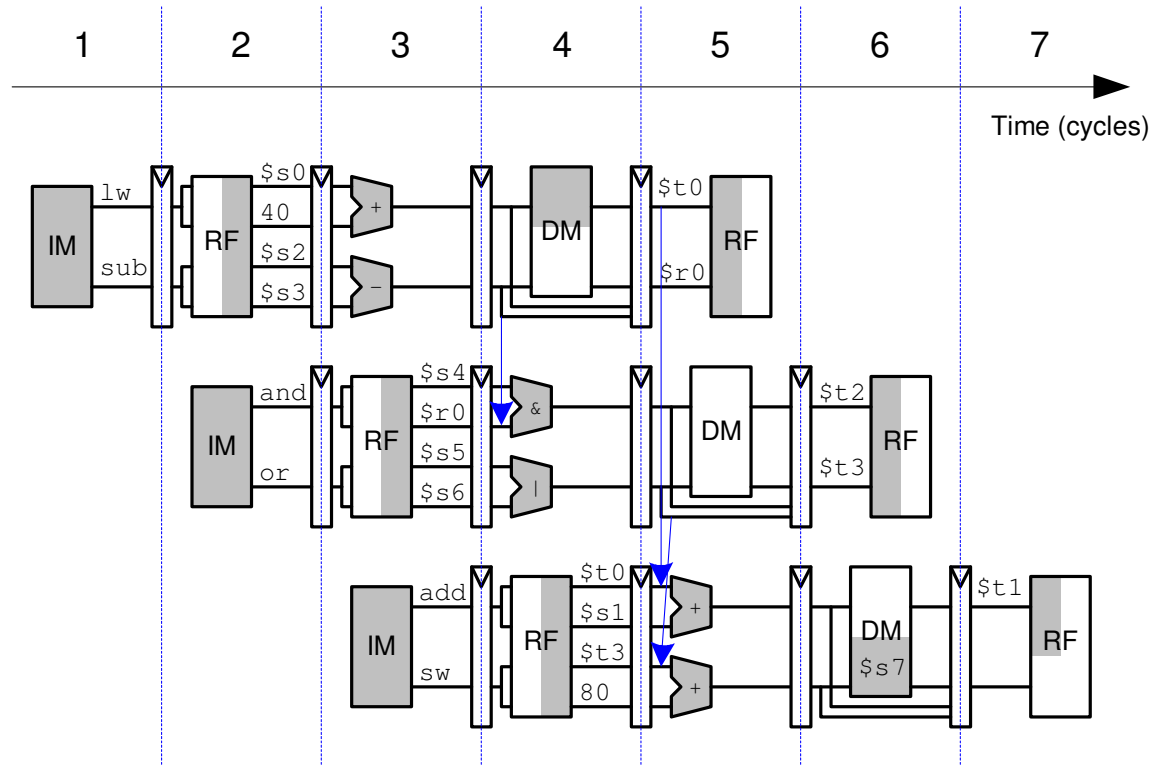
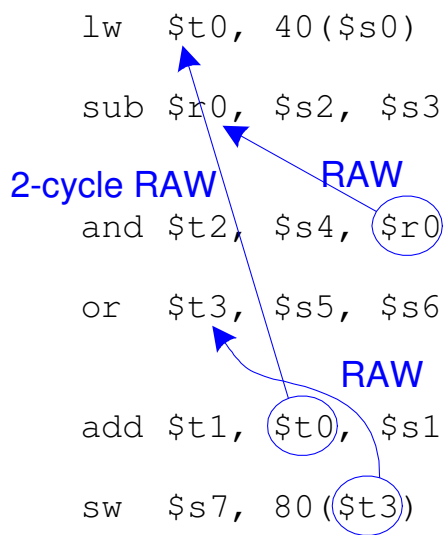
IPC Ideal: 2
IPC: 6/4 = 1.5



Register Renaming

```
lw $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or $t3, $s5, $s6
sw $s7, 80($t3)
```

IPC Ideal: 2
 IPC: $6/3 = 2$

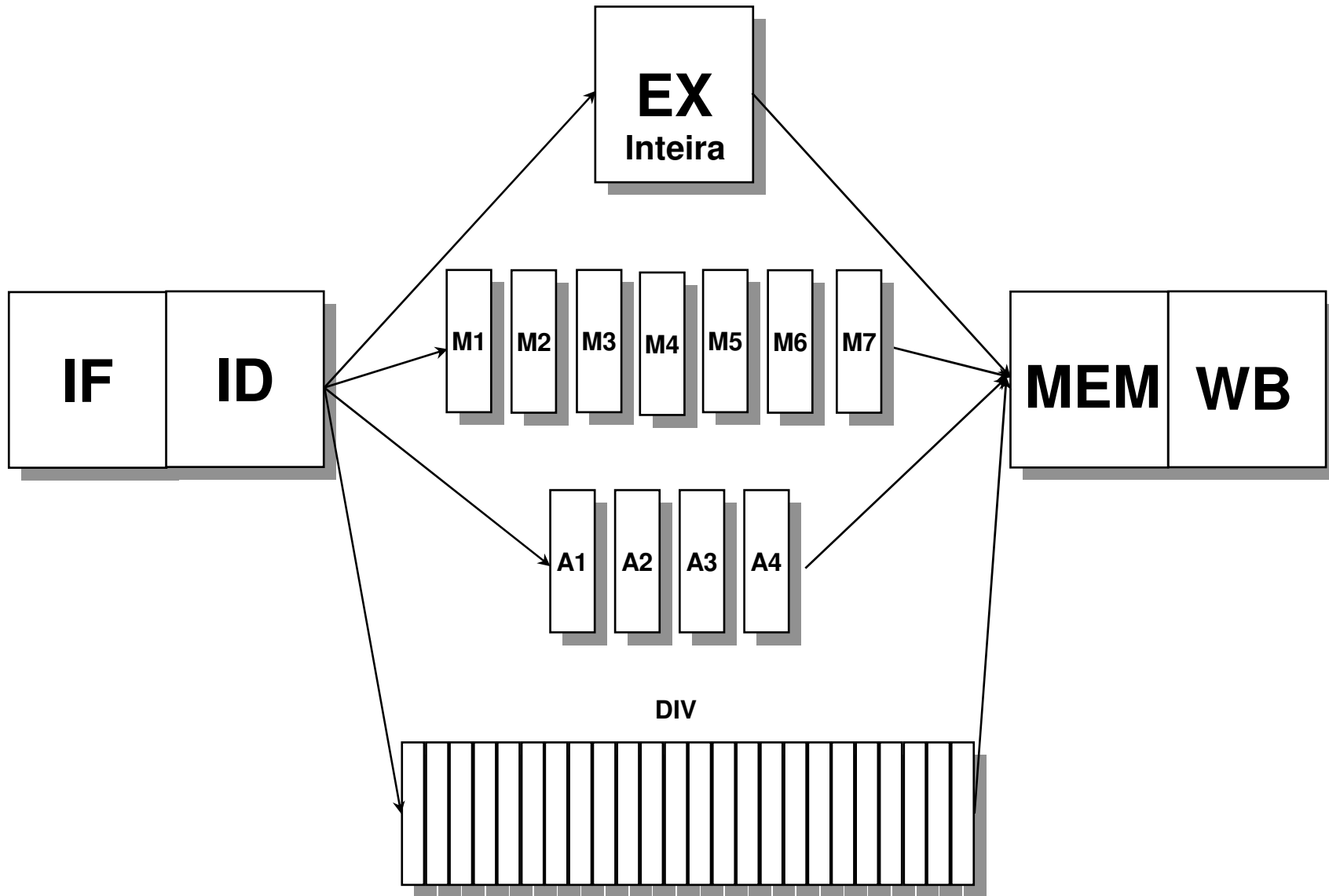


Algoritmo de Tomasulo

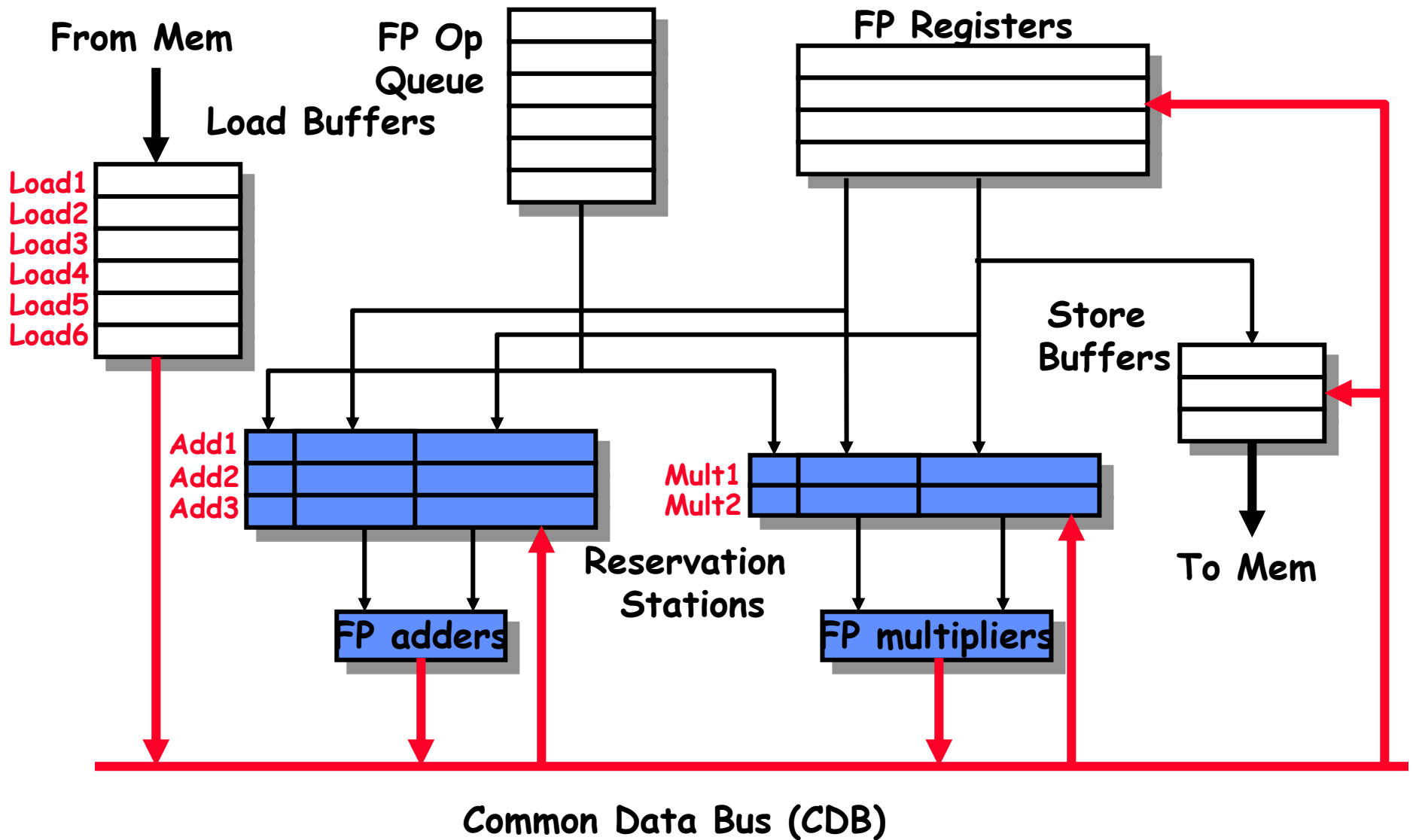
Exemplo

- Foco: Unidades de ponto-flutuante e load-store
- Cada estágio pode ter um número arbitrário de ciclos
- Múltiplas unidades funcionais
- Diferentes instruções possuem tempos diferentes no estágio EX
- Unidades disponíveis: **load-store; mult e adder**

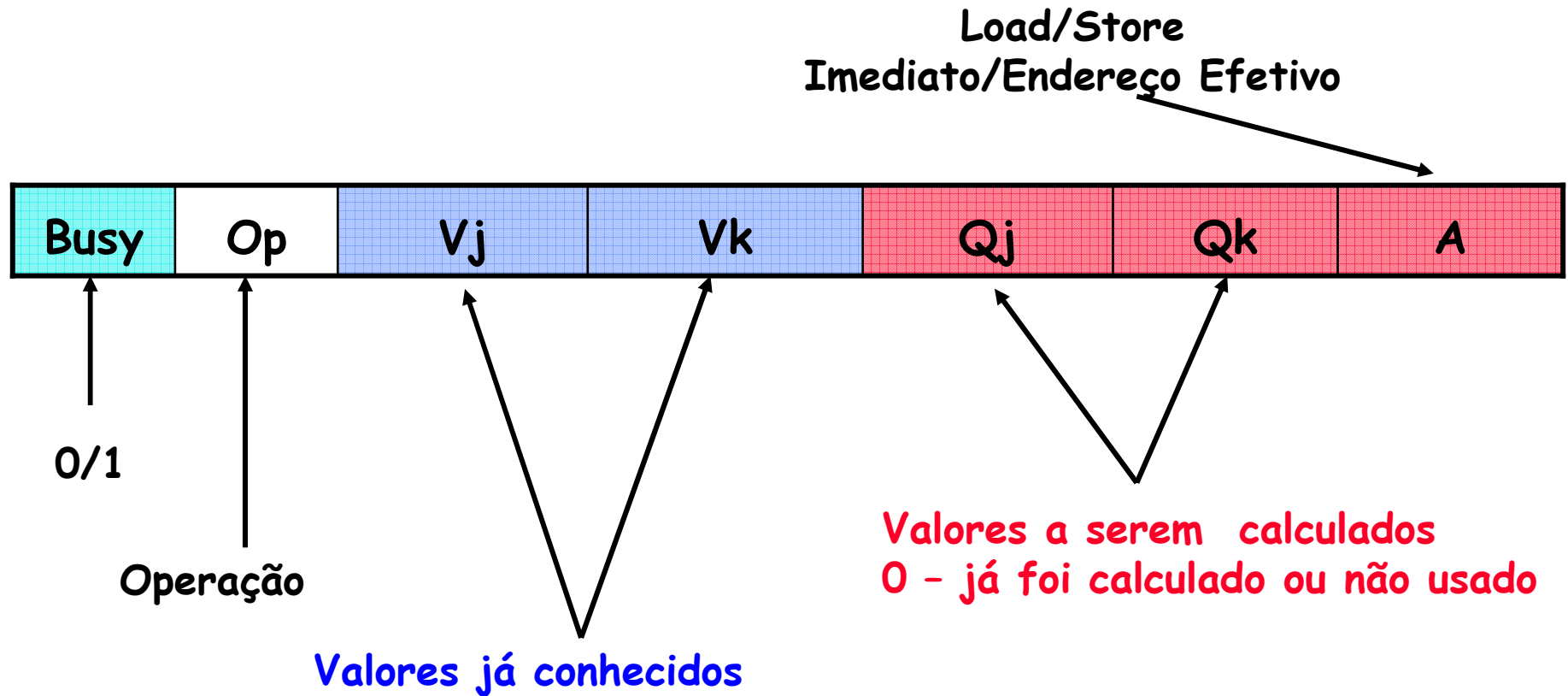
Scheduling Dinâmico Implementação - MIPS



Estrutura Básica de uma Implementação do Algoritmo de Tomasulo (para o MIPS)



Reservation Station



OBS.: Register File
Qi = No. RS

OBS.: Terminologia do Sreboard
Do CDC (ver apêndice A)

Reservation Station

Op: Operação a ser executada na unidade (e.g., + or -)

Vj, Vk: **Valores** dos operandos Fontes

- Store buffers tem campos V, resultados devem ser armazenados

Qj, Qk: Reservation Stations produzirá os operandos correspondentes (valores a serem escritos)

- $Q_j, Q_k = 0 \Rightarrow$ ready
- Store buffers tem somente Q_i para RS producing result

Busy: Indica que a Reservation Station e sua FU estão ocupadas

A: Mantém informação sobre o end. de memória calculado para load ou store

Register result status (campo Q_i no register file) — Indica para cada registrador a unidade funcional (reservation station) que irá escreve-lo. Em branco se não há instruções pendentes que escreve no registrador.

3 estágios do algoritmo de Tomasulo

1. Issue— pega a instrução na “FP Op Queue”

Se a **reservation station** está livre (não há hazard estrutural),
issues instr & envia operandos (**renames registers**)

2. Execute —executa a operação sobre os operandos (EX)

Se os dois operandos estão prontos executa a operação:

Se não, monitora o **Common Data Bus** (espera pelo cálculo do

operando, essa espera resolve RAW)

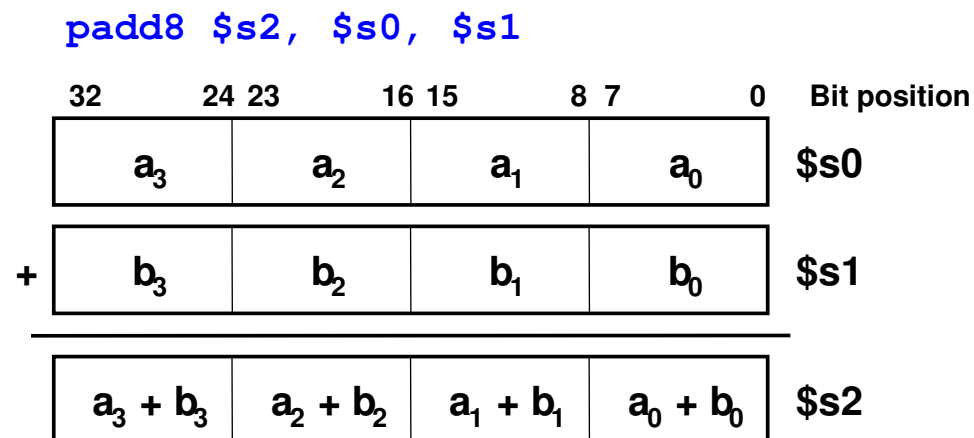
(quando um operando está pronto -> **reservation table**)

3 estágios do algoritmo de Tomasulo

- data bus normal: dado + destino (“go to” bus)
- Common data bus: dado + source (“come from” bus)
 - 64 bits de dados + 4 bits para endereço da Functional Unit
 - Escreve se há casamento com a Functional Unit (produz resultado)
 - broadcast

SIMD

- Single Instruction Multiple Data (SIMD)
 - Uma única instrução aplicada a múltiplos (pedaços de) dados
 - Aplicação Comum: computação gráfica
 - Executa operações aritméticas curtas (também denominadas de *packed arithmetic*)
- Exemplo, quatro **add** de elementos de 8-bit
- ALU deve ser modificada para eliminar os carries entre os valores de 8-bit



Técnicas Avançadas

- **Multithreading**
 - Wordprocessor: thread para typing, spell checking, printing

- **Multiprocessors**
 - Múltiplos processadores (cores) em um único chip

Multithreading: Algumas Definições

- **Processo:** programa executando em um computador
- Múltiplos processos podem estar em execução ao mesmo tempo: navegando na Web, ouvindo musica, escrevendo um artigo etc
- **Thread:** parte de um programa
- Cada processo possui múltiplas threads: em processador de texto tem threads para typing, spell checking, printing ...
- Em um **computador convencional:**
 - Uma thread está em execução por vez
 - Quando uma thread para (por exemplo, devido a um page fault):
 - » O estado da thread é guardado (registradores,)
 - » O estado da thread em espera é carregado no processador e inicia-se sua execução
 - » Chamado de **context switching**
 - Para o usuário parece que todas as threads executam simultaneamente (existem outras condições que provocam mudança da thread em execução: acesso a disco, time-out, ...)

Multithreading

- Múltiplas cópias de status da arquitetura (uma por thread)
- Múltiplas threads **ativas** por vez:
 - Quando uma thread para, outra inicia sua execução imediatamente (não é necessário armazenar e restaurar o status)
 - Se uma thread não tem todas as unidades de execução necessárias, outra thread pode ser executada
- Não aumenta o ILP de uma única thread, porém aumenta o throughput

Multiprocessors

- Multiple processors (cores) com alguma forma de comunicação entre eles
- Tipos de multiprocessamento:
 - **Symmetric multiprocessing (SMT)**: múltiplos cores com memória compartilhada
 - **Asymmetric multiprocessing**: cores separados para diferentes tarefas (por exemplo, DSP e CPU em um telefone celular)
 - **Clusters**: cada core possui seu próprio sistema de memória

Outras Fontes para Leitura

- Patterson & Hennessy's: *Computer Architecture: A Quantitative Approach* 3ª e 4ª Edições
- Conferências:
 - www.cs.wisc.edu/~arch/www/
 - ISCA (International Symposium on Computer Architecture)
 - HPCA (International Symposium on High Performance Computer Architecture)