Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
IMD0041 – Introdução a Organização e Arquitetura de Computadores

Pipeline:
Execução, Dependências e Conflitos

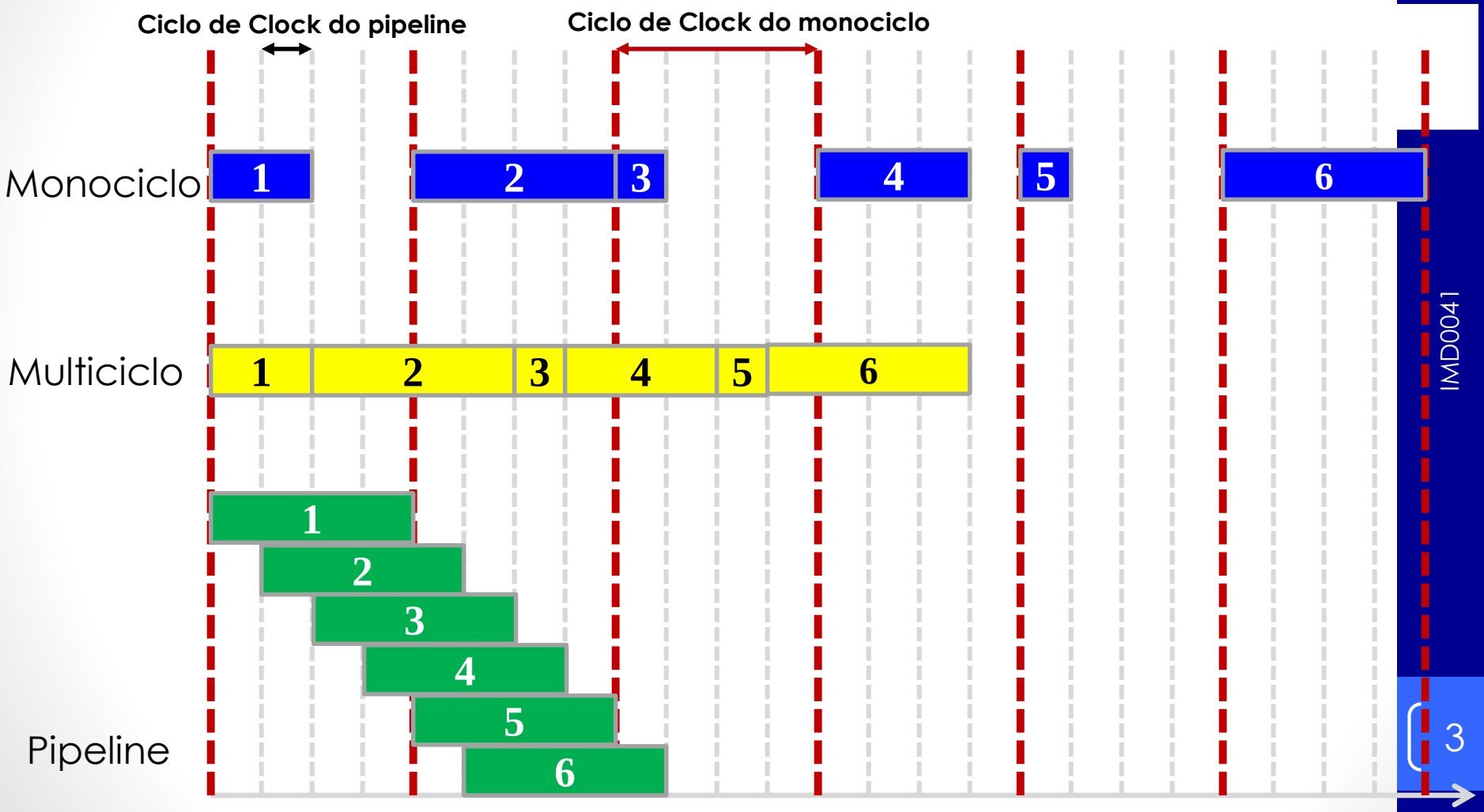
Profª Gustavo Girão

*Baseado no material do Professor Flávio Wagner - UFRGS

Plano de Aula

- Relembrar as três técnicas:
  - Monociclo
  - Multiciclo
  - Pipeline
- Exemplos de pipeline
- Dependências e Conflitos em pipelines

COMPARAÇÃO


Ciclo de Clock do pipeline
Ciclo de Clock do monociclo
Monociclo
1
2
3
4
5
6
Multiciclo
1
2
3
4
5
6
Pipeline
1
2
3
4
5
6

Pipeline de Instruções

- 2 estágios
  - (1) busca/ decodificação, (2) execução
- 3 estágios
  - (1) busca, (2) decodificação/busca de operandos, (3) execução
- 4 estágios
  - (1) busca, (2) decodificação/busca de operandos, (3) execução, (4) armazenamento do resultado
- 5 estágios
  - (1) busca, (2) decodificação/cálculo do endereço dos operandos, (3) busca de operandos, (4) execução, (5) armazenamento do resultado
- 6 estágios
  - (1) busca, (2) decodificação, (3) cálculo do endereço dos operandos, (4) busca de operandos, (5) execução, (6) armazenamento do resultado

Exemplos

- MIPS R4000 (superpipeline) – 8 estágios

http://www.ece.mtu.edu/faculty/rmkieckh/cla/4173/MIPS-R04K-uman3.pdf

1. Busca da instrução – metade 1
2. Busca da instrução – metade 2
3. Busca dos registradores
4. Execução
5. Busca de dados – metade 1
6. Busca de dados – metade 2
7. Verificação de tag *relacionado a hit e miss de cache – próx. aulas
8. Escrita do resultado

Exemplos

- Intel
  - i386 (1986): o primeiro a ter pipeline (3 estágios)
  - i486 (1989): aumentou para 5 estágios
  - Pentium (1993): dois pipelines (superscalar)
  - Pentium III: 11 estágios (superpipeline)
  - Pentium 4: 20 estágios (superpipeline)

Exemplos

Atenção: existe um limite na quantidade de estágios, pois as tarefas começam a ficar muito curtas e não dá para aplicar pipeline nelas. Além de aumento de custo de outros aspectos.

- Retomando o pipeline clássico de 5 estágios...

Processador RISC de cinco estágios

- Cinco estágios de execução
  - IF: busca da instrução (*instruction fetch*)
  - ID: decodificação (*instruction decode*)
  - EX: execução ou cálculo do endereço efetivo (*execution*)
  - MEM: acesso à memória (*memory*)
  - WB: escrita do resultado (*write-back*)
- Cada estágio leva um ciclo de *clock*
- Uma nova instrução é iniciada a cada ciclo

Processador RISC de cinco estágios

Instrução	Ciclo de clock								
	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

Projeto do RISC de cinco estágios (1)

- Para executar diferentes estágios em um mesmo ciclo de *clock*, precisamos garantir que os diferentes estágios não podem precisar de um mesmo componente

- Para isso:
  - Memória de dados é separada da memória de instruções

Problemas no desempenho

1. Dividir todas as instruções num mesmo conjunto de estágios

2. Obter estágios com tempos de execução similares

3. Conflitos de memória

   Acessos simultâneos à memória por 2 ou mais estágios

4. Dependência de dados

   Instruções dependem de resultados de instruções anteriores, ainda não completadas

5. Instruções de desvio

   Instrução seguinte não está no endereço seguinte ao da instrução anterior

DEPENDÊNCIAS E CONFLITOS

Dependências x Conflitos

- A sobreposição potencial de instruções é chamada paralelismo em nível de instruções

- Se duas instruções são paralelas, elas podem ser executadas em pipeline sem causar paradas (*stalls*), desde que o pipeline tenha recursos suficientes

Dependências x Conflitos

- Se duas instruções são dependentes, elas não são paralelas, e devem ser executados em sequência ou parcialmente sobrepostas
- As dependências são propriedades dos programas
- O fato de uma dependência resultar em detecção de conflito e o fato desse conflito causar uma parada são propriedades da organização do pipeline

DEPENDÊNCIA DE DADOS

Dependência de dados

- Ocorre quando uma instrução *i* produz um resultado que será usado pela instrução *j* (após i)

```
lw      R3, 0(R0)
add     R1, R2, R3
```

dependência em registrador

```
sw      R2, 0(R0)
lw      R3, 0(R0)
```

dependência em posição de memória

Precisa escrever e ler na mesma posição de memória

```
addi    R0, R1, 4
sw      R0, 4(R0)
lw      R3, 0(R0)
```

dependência em posição de memória

Usa registradores para fazer a soma e depois usa os mesmos registradores para escrever e ler na memória

Read-after-write (RAW)

- Corresponde a uma dependência de dados **verdadeira**
- Uma instrução j tenta ler um recurso antes que outra instrução precedente i o escreva, de modo que a instrução j usa o valor antigo

```
lw  R3, 0(R0)          IF  ID  EX  MEM  WB
add R1, R2, R3             IF  ID  EX   MEM  WB
```

CONFLITOS

Pipeline: Conflitos

- Conflitos (ou *hazards*) são situações que impedem que uma instrução planejada seja executada no ciclo de relógio previsto
- Para solucionar um conflito, pode ser necessário parar (*stall*) o pipeline
- Conflitos reduzem o desempenho real em relação ao ganho ideal obtido pelo pipeline

Pipeline: Conflitos

Estrutural (Resource ou Structural Hazards)	• Quando há conflito de duas ou mais instruções no uso de um recurso de hardware em um mesmo ciclo de clock
Dados (Data Hazards)	• Quando uma instrução depende do resultado da anterior
Controle (Control Hazards)	• Causado por instruções de saltos e desvios

Conflitos Estruturais

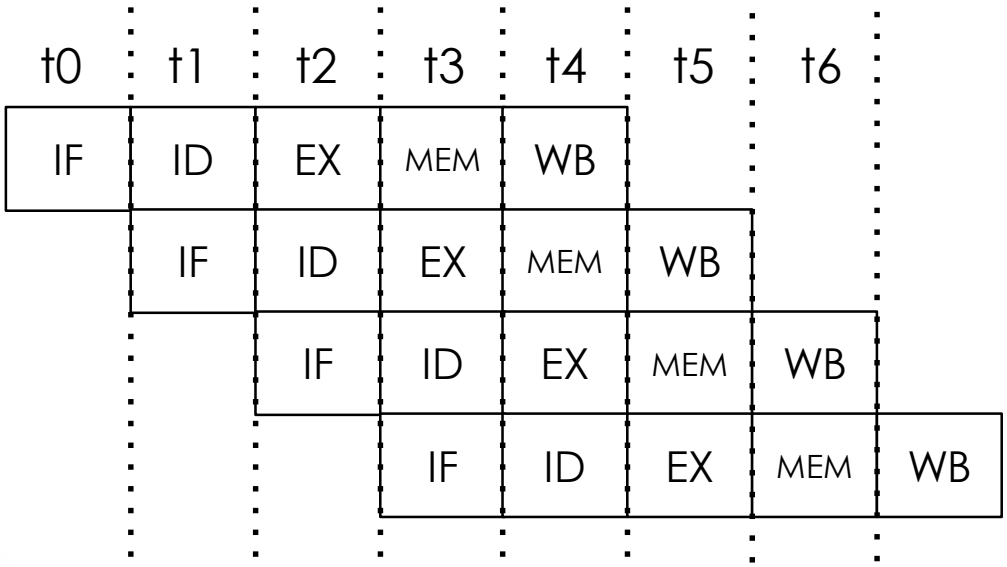
- Ocorrem quando duas (ou mais) instruções tentam utilizar simultaneamente um mesmo recurso de hardware

- Exemplo:
  - Sistema com acesso único para memória de dados e de instrução

Conflitos Estruturais - Exemplo

Somente uma memória para instruções e dados

```
1. lw    R0, 0(R1)
2. add   R2, R3, R4
3. sub   R7, R12, R4
4. addi  R10, R0, R1
```

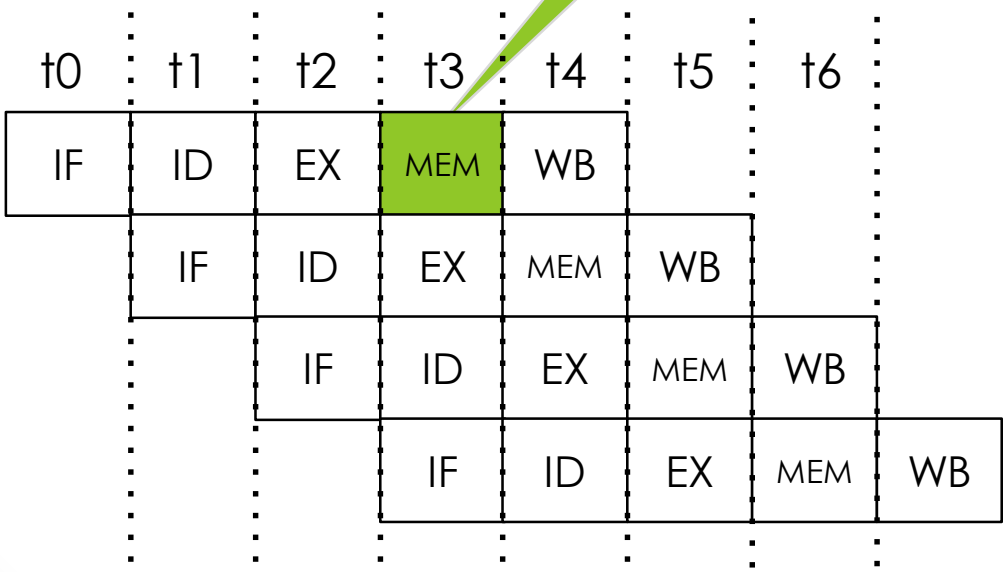


Conflitos Estruturais -

Fazendo uma leitura na memória

Somente uma memória para instruções e

```
1. lw    R0, 0(R1)
2. add   R2, R3, R4
3. sub   R7, R12, R4
4. addi  R10, R0, R1
```



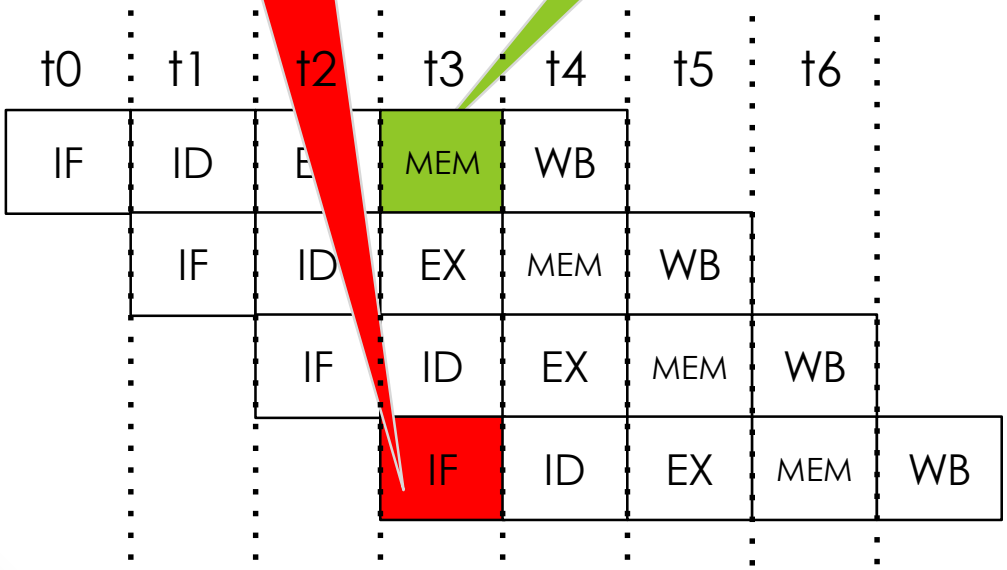
urais -

Fazendo uma leitura na memória

Fazendo uma leitura na memória

Somente u memória para instruções e s

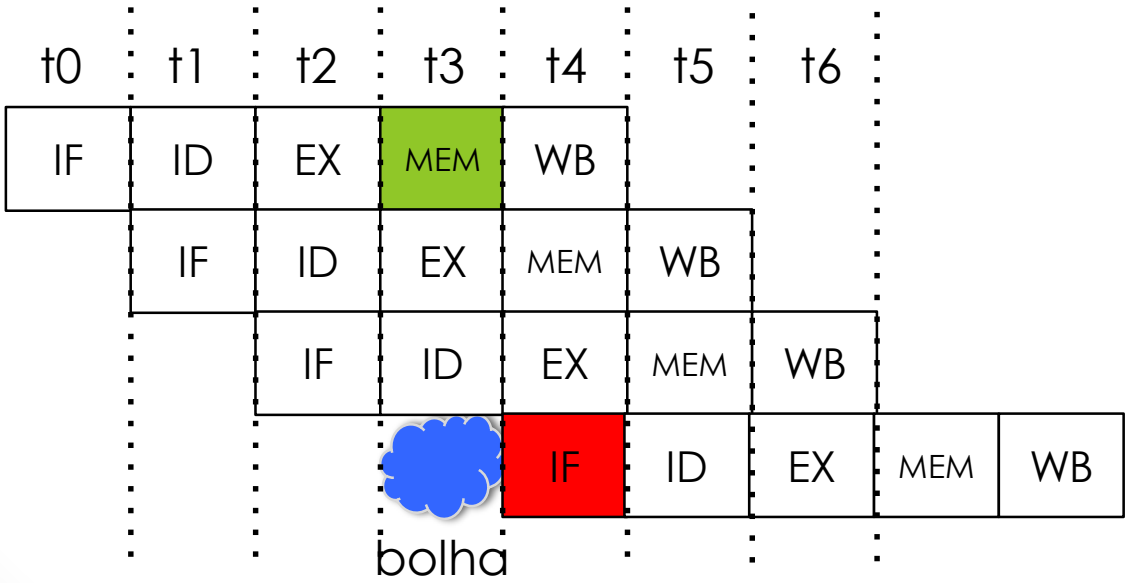
1. l R0, 0(R1)
2. a R2, R3, R4
3. su R7, R12, R4
4. add R10, R0, R1



Conflitos Estruturais - Exemplo

Somente uma memória para instruções e dados

```
1. lw    R0, 0(R1)
2. add   R2, R3, R4
3. sub   R7, R12, R4
4. addi  R10, R0, R1
```



Conflitos Estruturais - Exemplo

Somente uma memória para instruções e dados

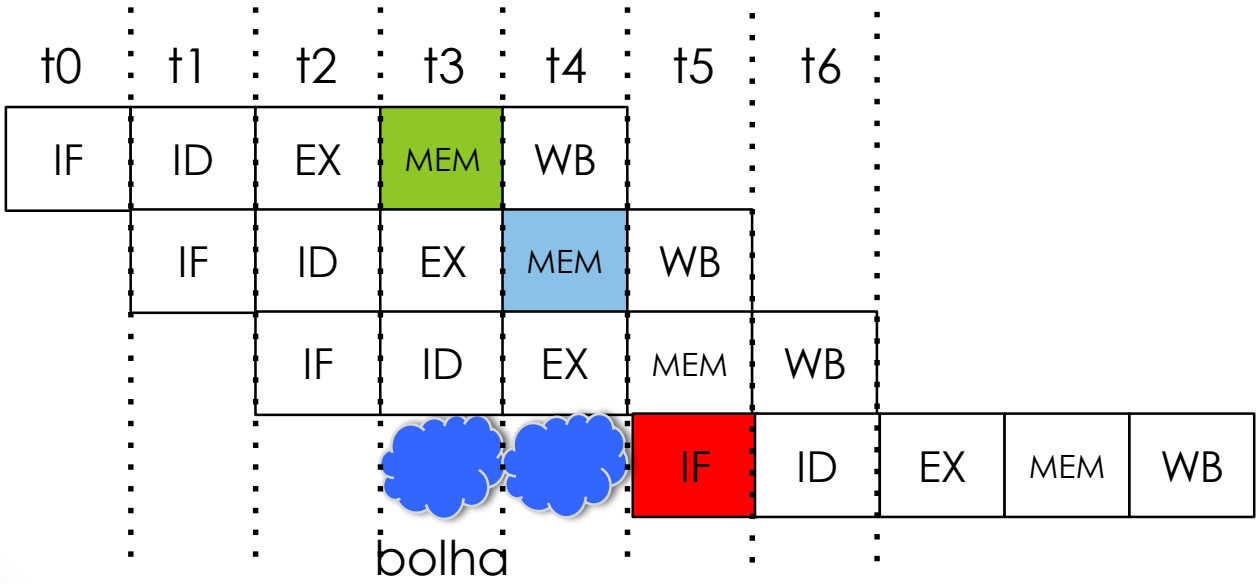
```
1. lw    R0, 0(R1)
2. lw    R2, 64(R5)
3. sub   R7, R12, R4
4. addi  R10, R0, R1
```

E NESSE EXEMPLO,
COMO FICA O PIPELINE?

Conflitos Estruturais - Exemplo

Somente uma memória para instruções e dados

```
1. lw    R0, 0(R1)
2. lw    R2, 64(R5)
3. sub   R7, R12, R4
4. addi  R10, R0, R1
```



Conflitos de dados

- Ocorrem quando uma instrução depende do resultado de uma anterior, e esta ainda não foi concluída
- Exemplo:

```
add   R1, R2, R3
sub   R4, R5, R1
```

Conflitos de dados

- Ocorrem quando uma instrução depende do resultado de uma anterior, e esta ainda não foi concluída

- Exemplo:

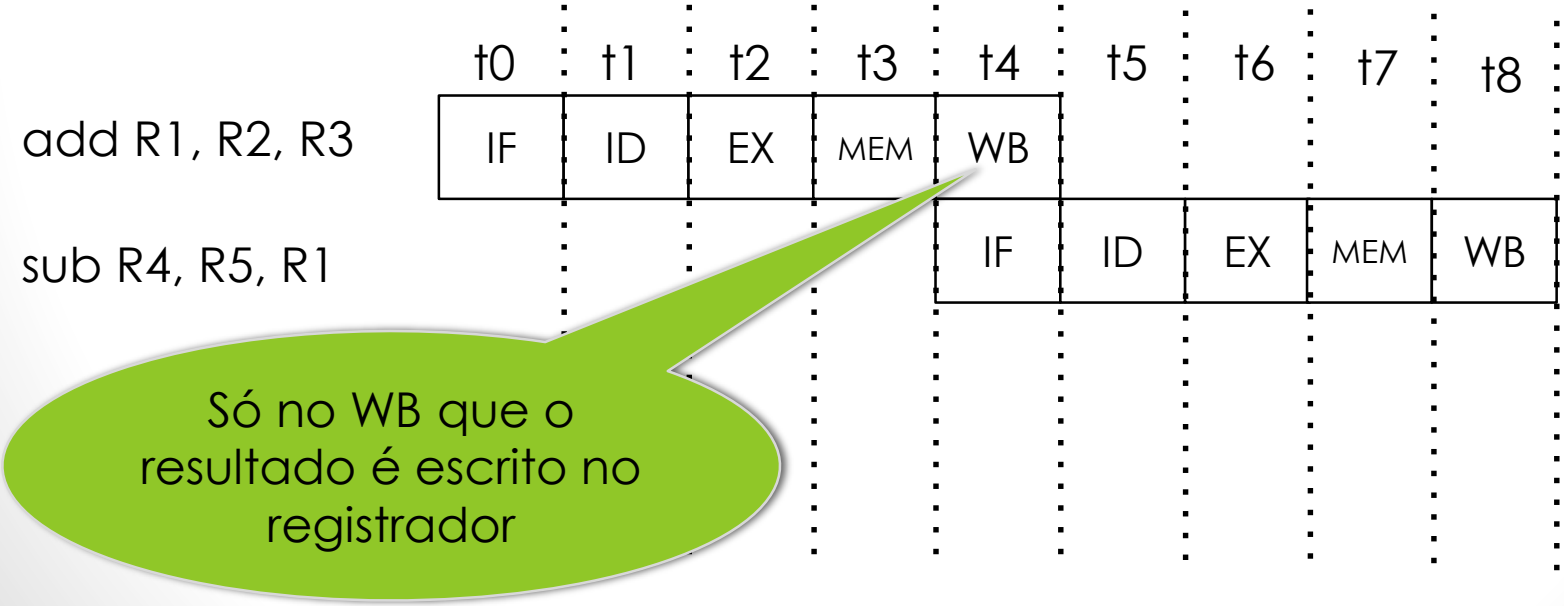
```
add   R1, R2, R3
sub   R4, R5, R1
```

EM QUAL TEMPO INICIA A SEGUNDA INSTRUÇÃO?

Conflitos de dados

- Ocorrem quando uma instrução depende do resultado de uma anterior, e esta ainda não foi concluída
- Exemplo:

```
add   R1, R2, R3
sub   R4, R5, R1
```



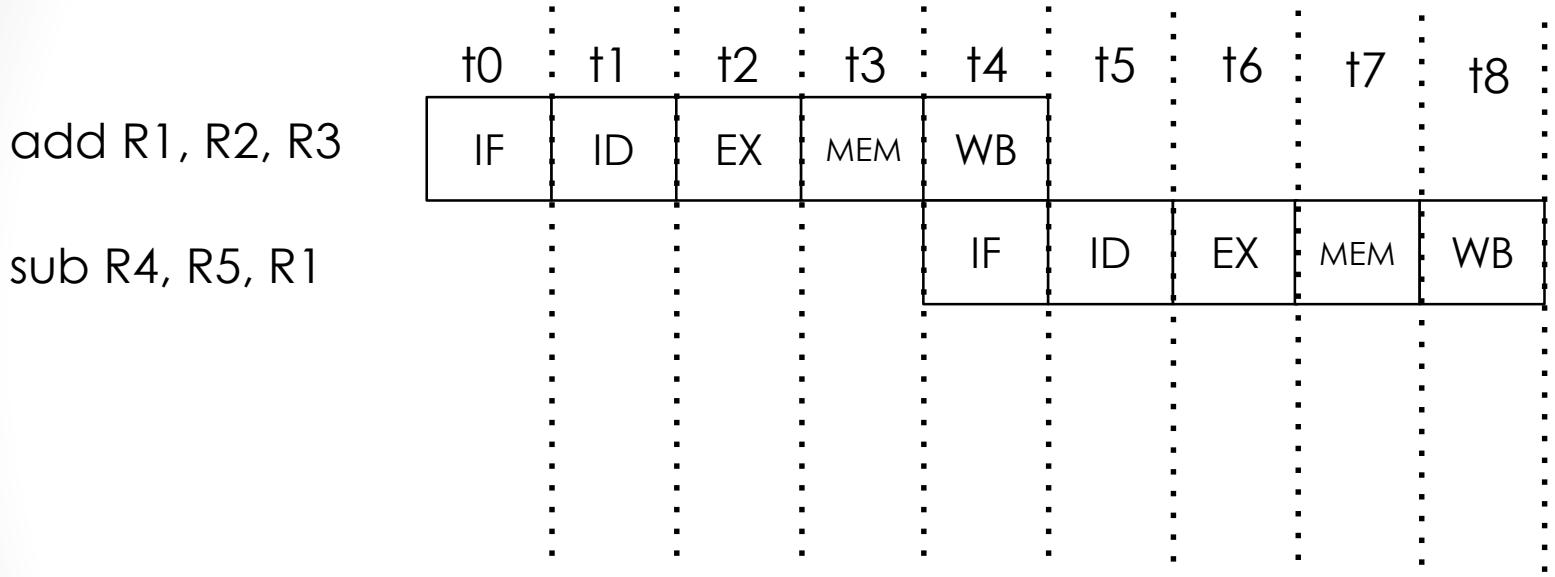
Conflitos de dados

POSSÍVEIS SOLUÇÕES?

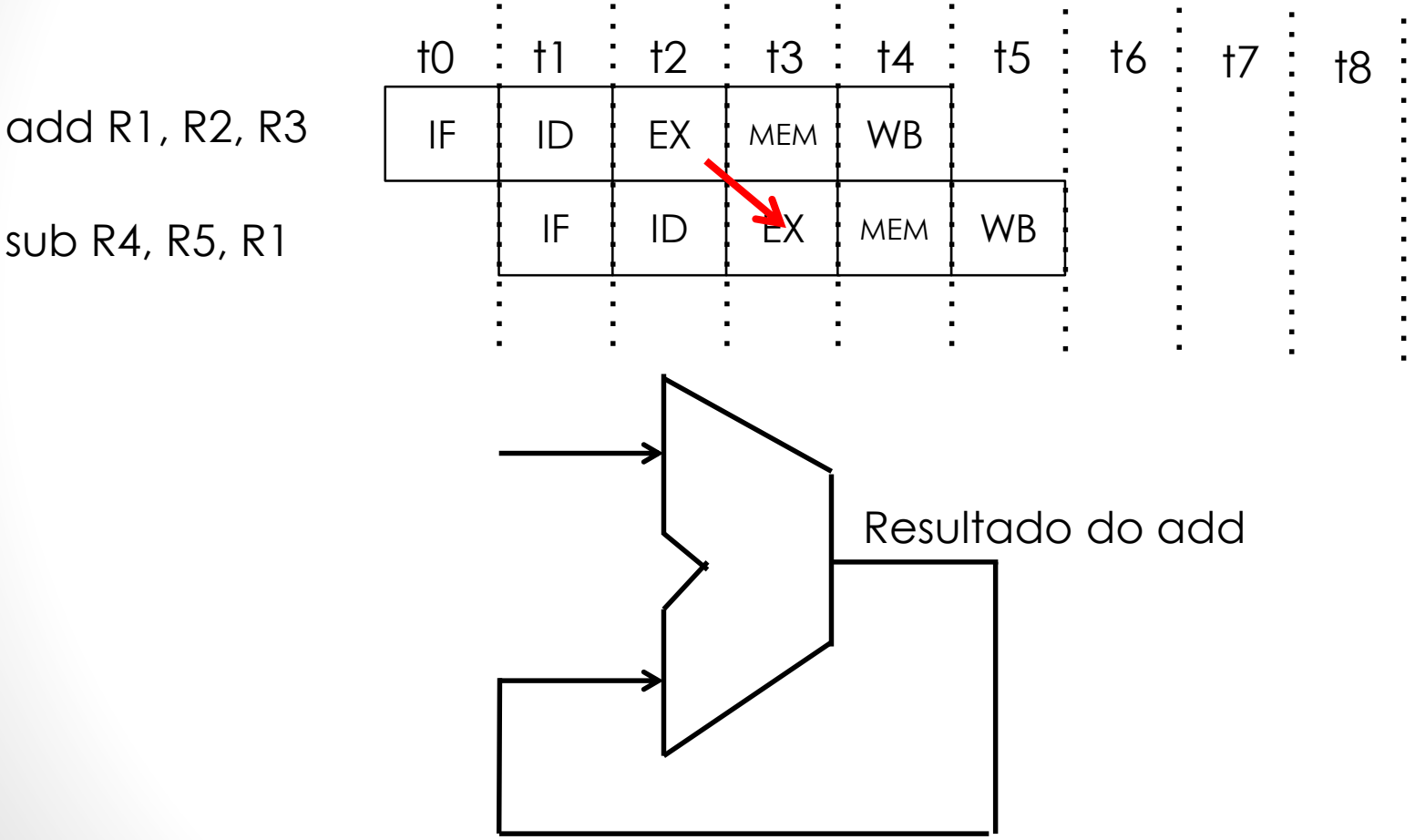
Conflitos de dados - Soluções

Stall	**1) Hardware para o pipeline ou 2) Compilador insere instruções NOP**
Forwarding (ou bypassing)	**Hardware redireciona resultados de um estágio para o outro**
Reordenamento	**Compilador ou programador alteram a ordem das instruções sem comprometer a correção do programa**

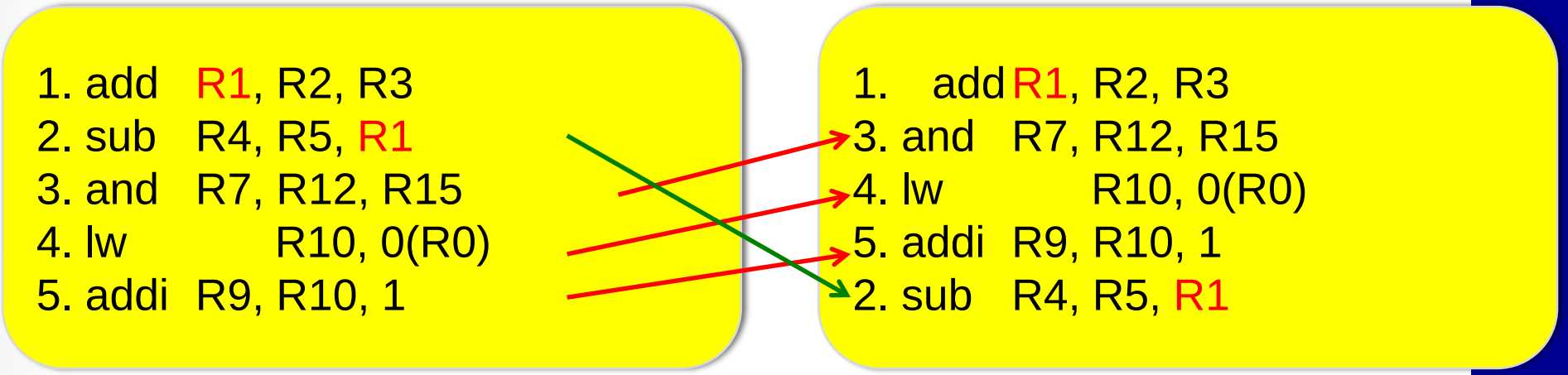
Stall


t0
t1
t2
t3
t4
t5
t6
t7
t8
add R1, R2, R3
IF
ID
EX
MEM
WB
sub R4, R5, R1
IF
ID
EX
MEM
WB

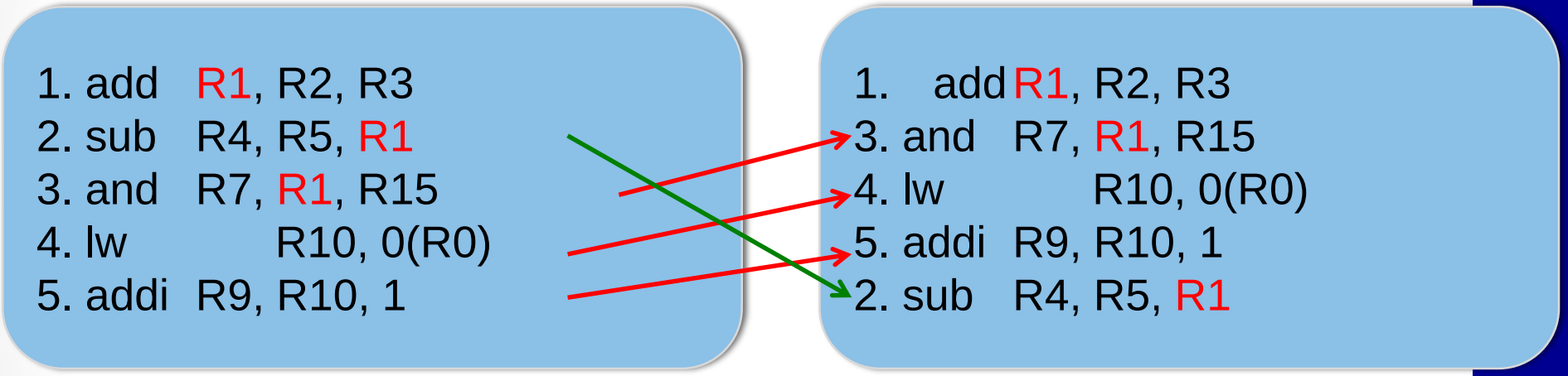
Redirecionamento



Reordenamento de código



Reordenamento de código



AS VEZES NÃO É POSSÍVEL REORDENAR

Considerando o pipeline de 5 estágios do MIPS, indique quantos ciclos são necessários para executar as instruções abaixo?

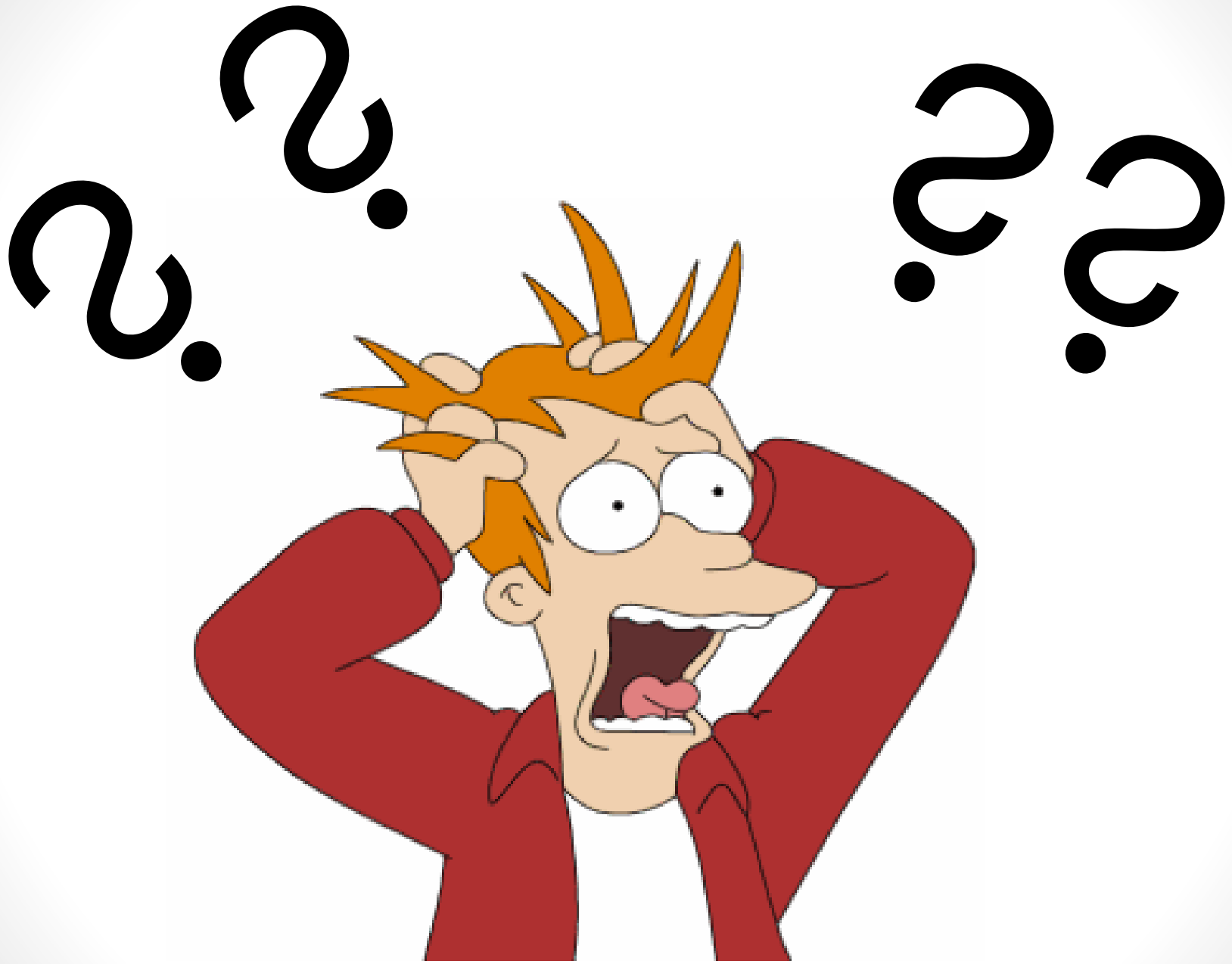
```
sub   $s0,   $s1,   $s2
addi  $s3,   $s0,   10
lw    $s4,   0($s0)
add   $s1,   $s4,   $s3
sub   $s2,   $s3,   $s1
sw    $s2,   10($s1)
and   $s9,         $s7,        $s7
addi  $s6,         $s7,        4
sw    $s3,   128($s1)
sw    $s9,   64($s1)
```

Usando REDIRECIONAMENTO, quantos ciclos para executar?

```
sub   $s0,   $s1,   $s2
addi  $s3,   $s0,   10
lw    $s4,   0($s0)
add   $s1,   $s4,   $s3
sub   $s2,   $s3,   $s1
sw    $s2,   10($s1)
and   $s9,         $s7,        $s7
addi  $s6,         $s7,        4
sw    $s3,   128($s1)
sw    $s9,   64($s1)
```

Usando REODERNAMENTO, quantos ciclos para executar?

```
sub   $s0,   $s1,   $s2
addi  $s3,   $s0,   10
lw    $s4,   0($s0)
add   $s1,   $s4,   $s3
sub   $s2,   $s3,   $s1
sw    $s2,   10($s1)
and   $s9,         $s7,         $s7
addi  $s6,         $s7,         4
sw    $s3,   128($s1)
sw    $s9,   64($s1)
```



Bibliografia

- Patterson e Hennessy

  Organização e projeto de computadores – A interface Hardware/Software

  CAPÍTULO 6 – Melhorando o desempenho com Pipelining

- William Stalling

  Arquitetura e Organização de Computadores

  CAPÍTULO 12 – Sessão 12.4

Próxima aula

- Pipeline: conflitos de controle