Arquitetura e Organização de Computadores II
Pipelines

Prof. Daniel S. Marcon
danielstefani@unisinos.br

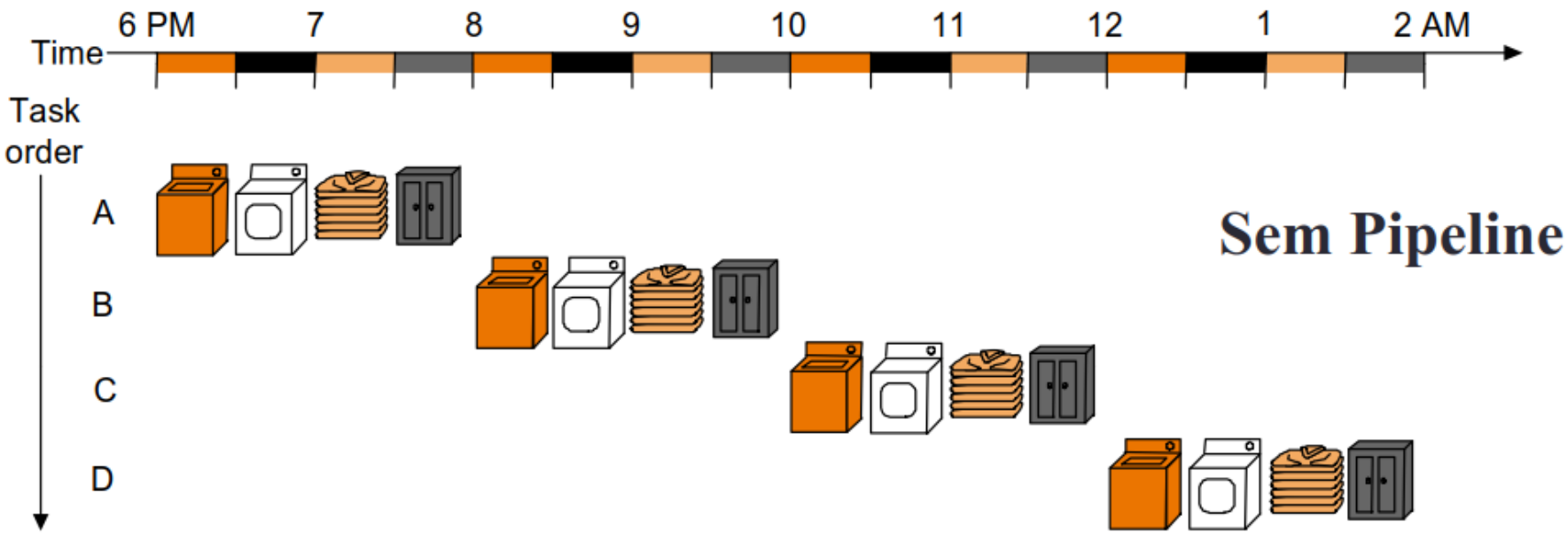
UNISINOS

Introdução

- Pipeline consiste na divisão de uma tarefa em n estágios
- N tarefas são executadas simultaneamente, uma em cada estágio
- Começa a funcionar o mais cedo possível, sem perder tempo
- **Objetivo**: aumentar o desempenho

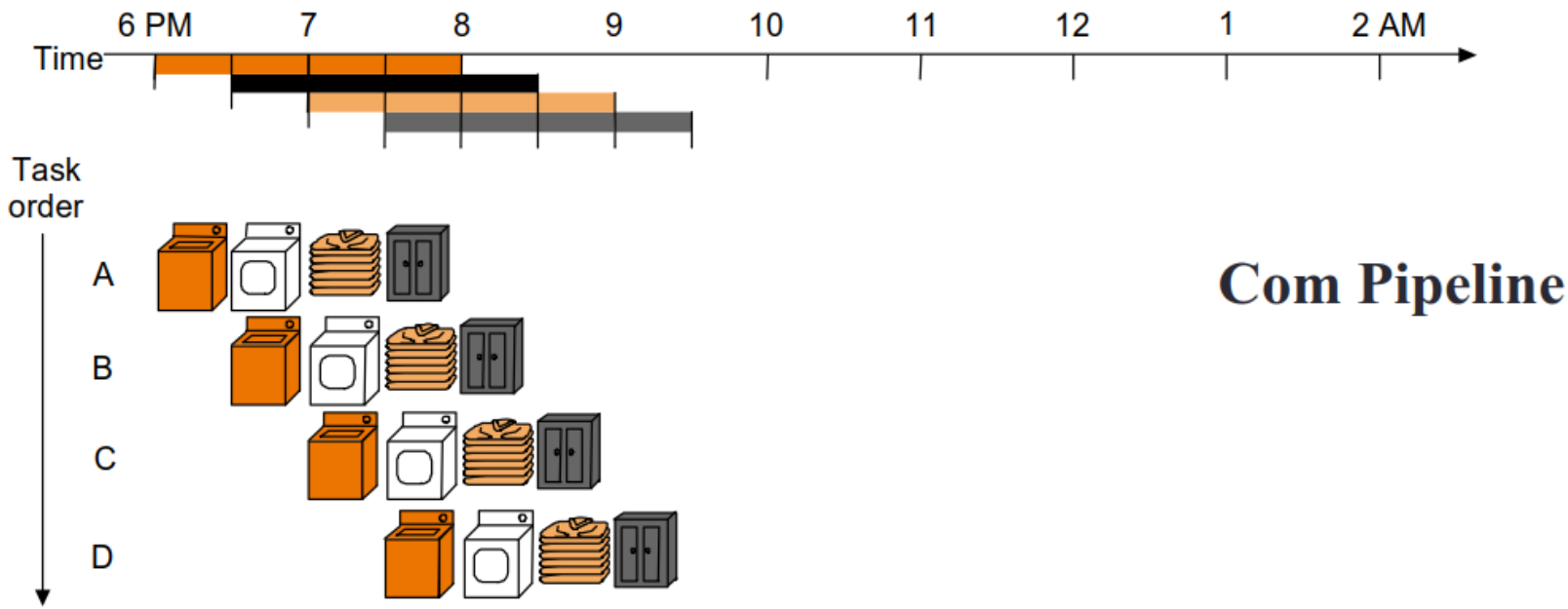
Analogia com uma Lavanderia

- Tarefa decomposta em lavar, secar, dobrar e guardar
- Duração de cada tarefa: 30 minutos



Analogia com uma Lavanderia (cont.)

- Estágios diferentes de cada tarefa usam diferentes recursos
  - Podem ser feitos ao mesmo tempo



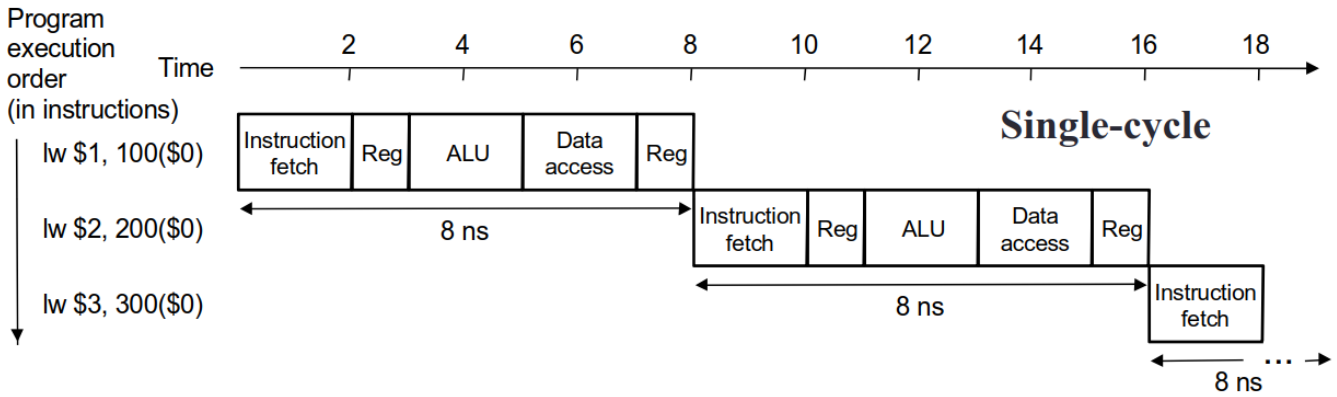
Com Pipeline

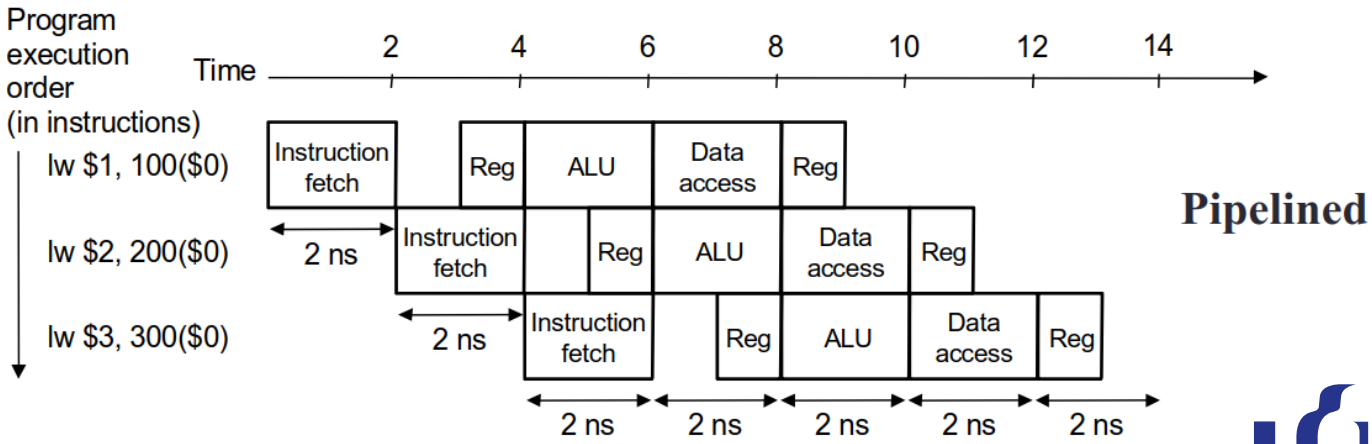
Pipeline no MIPS

1. Busca da instrução (instruction fetch - IF)
2. Decodificação da instrução e leitura dos registradores (instruction decode - ID)
3. Operações com a ULA (execution - EX)
4. Acesso à memória de dados (MEM)
5. Escrita em registradores (write back - WB)

Monociclo vs. Pipeline

- Assumindo 2ns para acessos à memória e operação na ULA, e 1ns para acesso ao banco de registradores.
  - Período do ciclo: 8 ns para o monociclo e 2ns para o pipeline





Atenção!

- Pipeline não reduz a latência de uma única tarefa, mas aumenta o throughput de toda a carga de trabalho
- O período do pipeline é limitado pelo estágio mais longo
  - Ganho em potencial = número de estágios
  - Períodos desbalanceados dos estágios reduzem o aumento de desempenho
- Há um tempo gasto para encher o pipeline e para esvaziá-lo
- Múltiplas instruções executam simultaneamente
  - Há sobreposição temporal de diversas fases das instruções

Desempenho

- Cada instrução leva o mesmo tempo (com ou sem pipeline)
  - Com pipeline, a **média** de tempo por instrução é dividida por n
- Assumindo que há s instruções e n estágios
- Primeira instrução: n ciclos do relógio
- $s - 1$ instruções seguintes: $s - 1$ ciclos do relógio
- Tempo total sem pipeline: n $\times$ s ciclos

Desempenho

- Cada instrução leva o mesmo tempo (com ou sem pipeline)
  - Com pipeline, a **média** de tempo por instrução é dividida por n
- Assumindo que há s instruções e n estágios
- Primeira instrução: n ciclos do relógio
- $s - 1$ instruções seguintes: $s - 1$ ciclos do relógio
- Tempo total sem pipeline: $n \times s$ ciclos
- Tempo total com pipeline: $n + (s - 1)$ ciclos

Desempenho

- Cada instrução leva o mesmo tempo (com ou sem pipeline)
  - Com pipeline, a **média** de tempo por instrução é dividida por n
- Assumindo que há s instruções e n estágios
- Primeira instrução: n ciclos do relógio
- $s - 1$ instruções seguintes: $s - 1$ ciclos do relógio
- Tempo total sem pipeline: n $\times$ s ciclos
- Tempo total com pipeline: n $+ (s - 1)$ ciclos
- Speedup: $\frac{n \times s}{n + (s-1)}$

Vantagens e Desvantagens

- Vantagens:

Vantagens e Desvantagens

- Vantagens:
  - Reduz o tempo médio de execução dos programas
    - Reduz o CPI médio
    - Reduz a duração do ciclo de relógio
  - Acelera o processamento sem mudar a forma de programação

Vantagens e Desvantagens

- Vantagens:
  - Reduz o tempo médio de execução dos programas
    - Reduz o CPI médio
    - Reduz a duração do ciclo de relógio
  - Acelera o processamento sem mudar a forma de programação
- Desvantagens:

Vantagens e Desvantagens

- Vantagens:
  - Reduz o tempo médio de execução dos programas
    - Reduz o CPI médio
    - Reduz a duração do ciclo de relógio
  - Acelera o processamento sem mudar a forma de programação
- Desvantagens:
  - Implementação mais complexa (acrescenta custos em tempo de projeto e hardware de controle)

Pipeline em Computadores

- Executar bilhões de intruções: throughput é o que interessa
- Qual o conjunto de instruções para fazer pipeline?
  - Tamanho de instruções variável ou todas as instruções de mesmo tamanho?
  - Operandos que acessam a memória em qualquer instrução ou acesso à memória somente durante load e store?
  - Registradores fonte em vários locais possíveis na instrução ou registradores sempre localizados no mesmo campo?

Exemplo do MIPS

- Todas instruções tem o mesmo comprimento
  - Busca e decodificação são similares para todas instruções
- Poucos formatos de instruções
  - Simplifica a decodificação, que pode ser feita em um estágio
- Operandos de memória aparecem apenas em loads/stores
  - Acessos à memória podem ser retardados para um estágio posterior
- Operandos são alinhados na memória
  - Uma transferência requer um estágio para acesso à memória

Exercício

- Como o desempenho seria afetado se a ULA tiver um atraso de 4ns ao invés de 2ns?

Conflitos do Pipeline

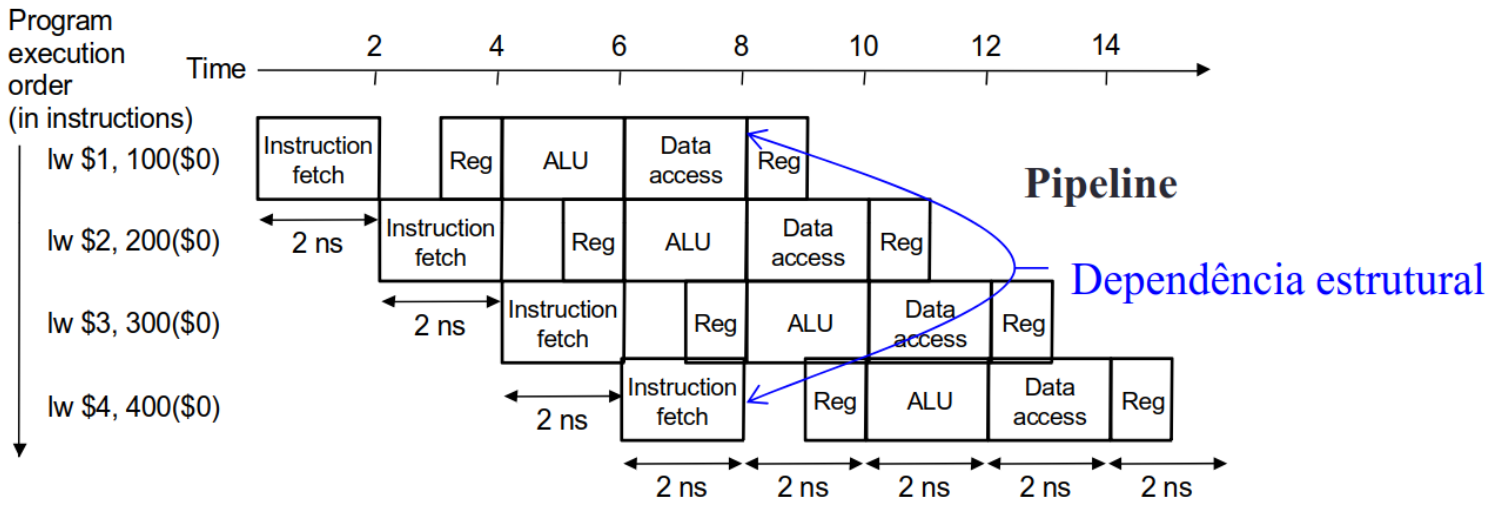
- Existem situações no pipeline em que a instrução seguinte não pode ser executada no próximo ciclo do relógio
  - Por quê?

Conflitos do Pipeline

- Existem situações no pipeline em que a instrução seguinte não pode ser executada no próximo ciclo do relógio
  - Por quê?
- Tipos de conflitos
  - Estruturais
  - De controle
  - De dados

Conflitos Estruturais

- Instruções diferentes, em estágios diferentes do pipeline, precisam utilizar o mesmo recurso de hardware
- Ou seja, o hardware não pode suportar uma determinada combinação de instruções no mesmo ciclo
- **Exemplo**: caso a memória de instruções e a memória de dados fosse a mesma no MIPS



Conflitos de Controle

- Necessidade de tomar uma decisão baseada nos resultados de uma instrução que ainda não foi concluída
  - A próxima instrução que irá ao pipeline depende da saída de uma instrução de desvio, que já está no pipeline
  - **Exemplo**: no MIPS, não sabemos qual é a decisão do desvio condicional até o quarto estágio

Conflitos de Controle

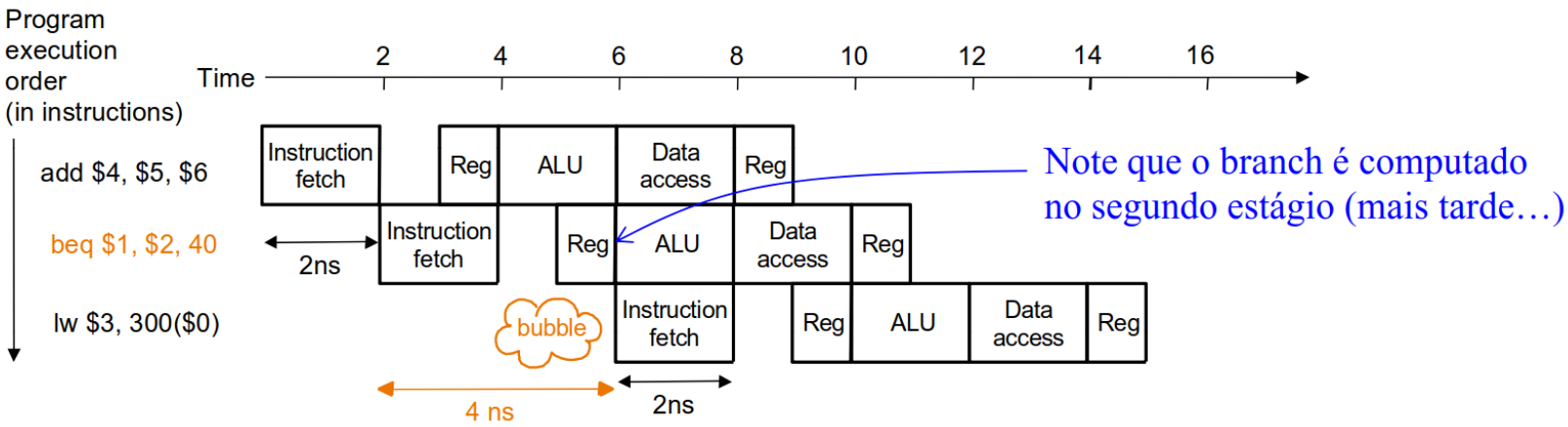
- Necessidade de tomar uma decisão baseada nos resultados de uma instrução que ainda não foi concluída
  - A próxima instrução que irá ao pipeline depende da saída de uma instrução de desvio, que já está no pipeline
  - **Exemplo**: no MIPS, não sabemos qual é a decisão do desvio condicional até o quarto estágio
- Possíveis soluções?

Conflitos de Controle

- Necessidade de tomar uma decisão baseada nos resultados de uma instrução que ainda não foi concluída
  - A próxima instrução que irá ao pipeline depende da saída de uma instrução de desvio, que já está no pipeline
  - **Exemplo**: no MIPS, não sabemos qual é a decisão do desvio condicional até o quarto estágio
- Possíveis soluções?
  - Parada
  - Predição estática
  - Predição dinâmica
  - Decisão retardada

Conflitos de Controle: parada

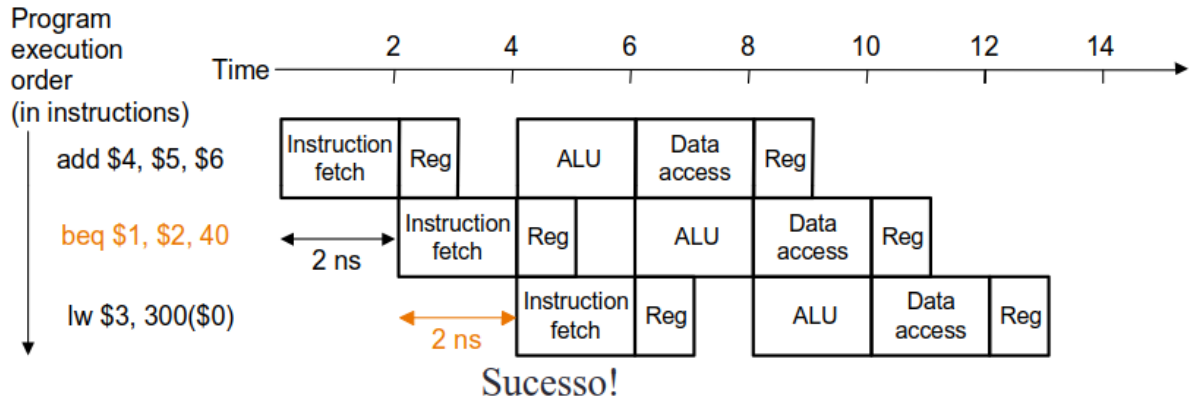
- Não executar uma instrução enquanto não tiver o resultado
- Inserir bolhas enquanto o resultado não for obtido



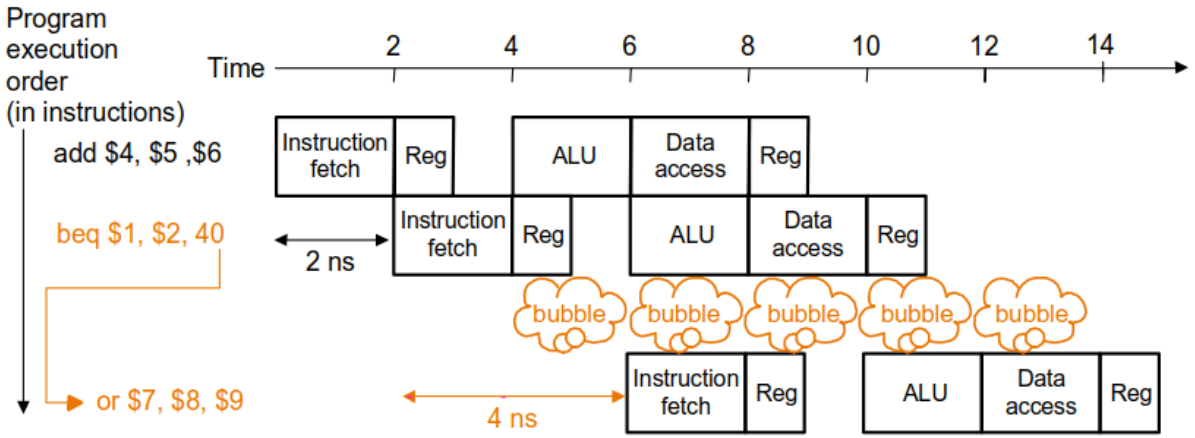
Pipeline stall

Conflitos de Controle: predição estática

- Prever o resultado de desvio (exemplo: não ocorre desvio)



Sucesso!



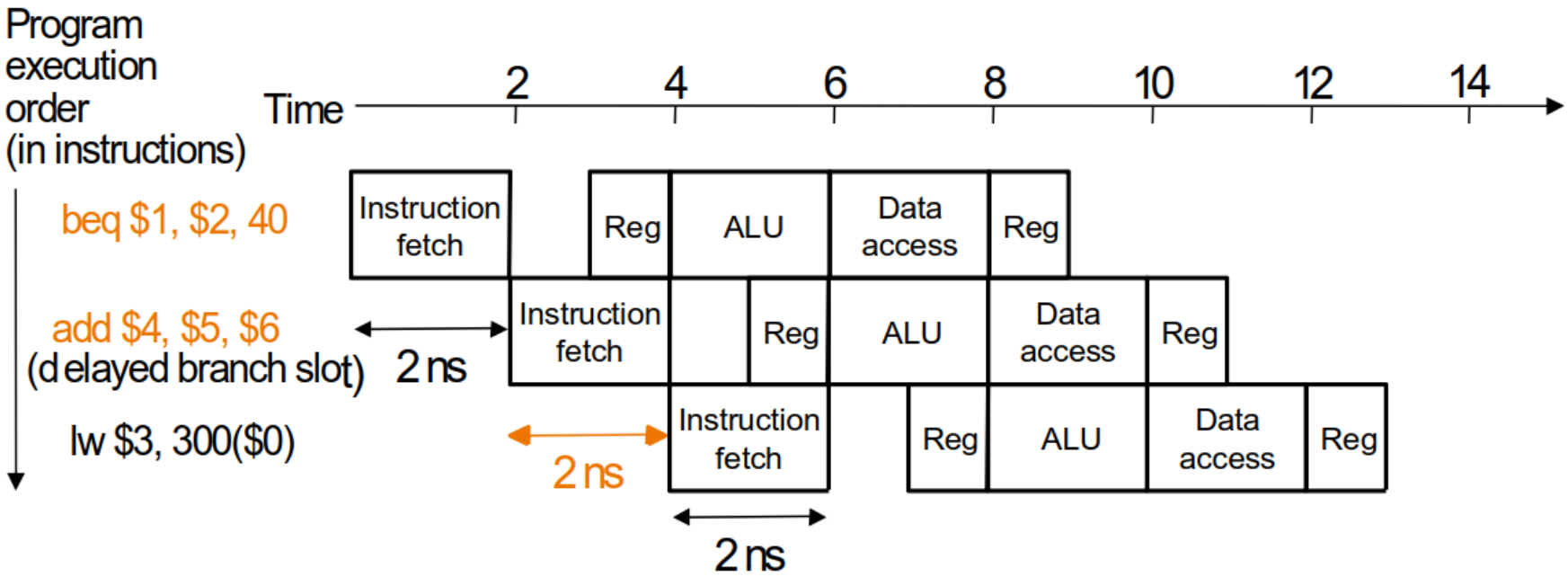
Falha: Desfazer (=flush) `lw`

Conflitos de Controle: predição dinâmica

- Hardware decide o que fazer com base no histórico
- Pode mudar o comportamento para um mesmo desvio com o decorrer da execução do programa
- Utiliza look-up table
- Apresenta até 85% a 90% de acerto

Conflitos de Controle: decisão retardada

- Insere uma instrução que não depende do desvio após a instrução de desvio

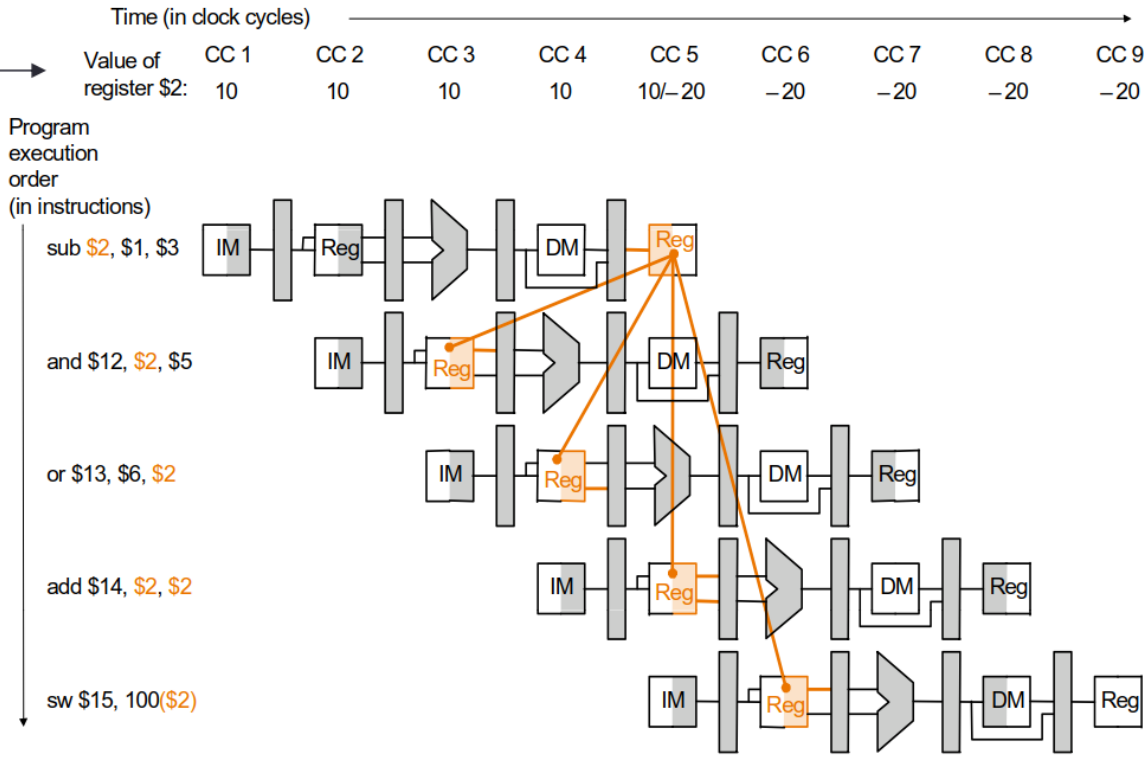


Conflitos de Dados

- Instruções consecutivas podem fazer acesso aos mesmos operandos
  - A execução de uma instrução depende do resultado de outra que ainda está no pipeline
- Tipos de dependências de dados:
  - Dependência verdadeira
  - Antidependência
  - Dependência de saída

Conflitos de Dados

$2 = 10 antes do sub;
$2 = -20 depois do sub




```
sub $2,   $1,   $3
and $12,  $2,   $5
or  $13,  $6,   $2
add $14,  $2,   $2
sw  $15,  100($2)
```

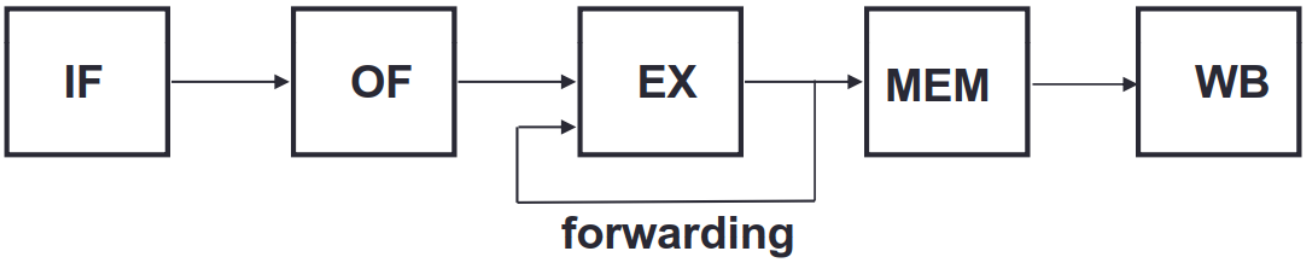

UNISINOS

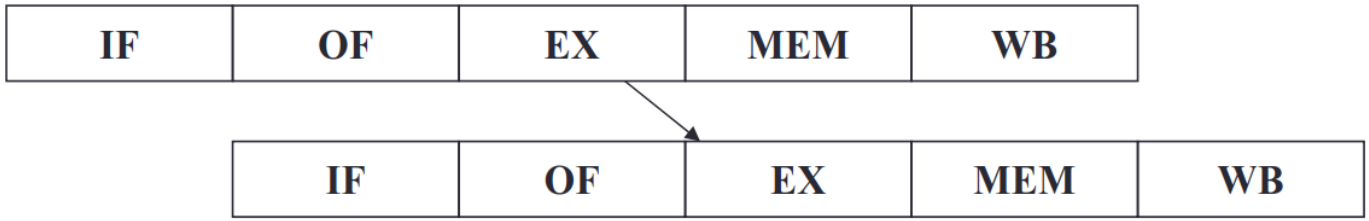
Dependências Verdadeiras

- Exemplo
  1. ADD **R3**, R2, R1 ; R3 = R2 + R1
  2. SUB R4, **R3**, 1 ; R4 = R3 – 1
- Instrução 2 depende de valor de R3 calculado pela instrução 1
- Instrução 1 precisa atualizar valor de R3 antes que instrução 2 busque os seus operandos
- Pipeline precisa ser parado durante certo número de ciclos

Solução 1: forwarding (adiantamento)

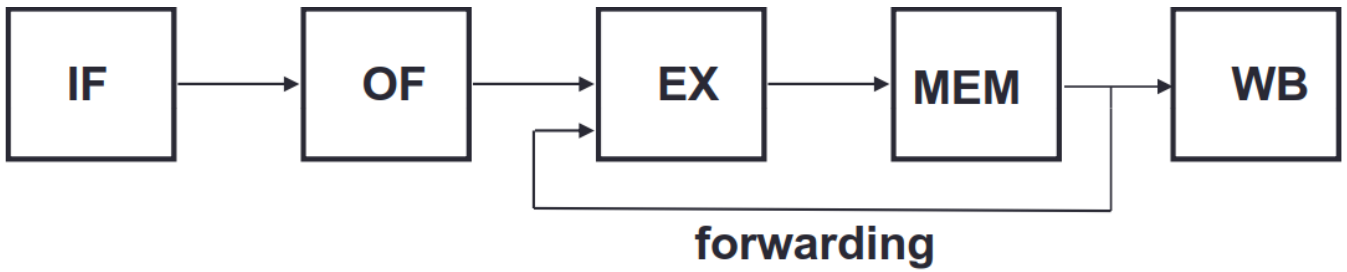
- O resultado pode ser disponibilizado para os demais estágios tão logo ele seja obtido
  - Inserção de sinais de adiantamento
- Caminho interno entre a saída da ULA e a entrada da ULA

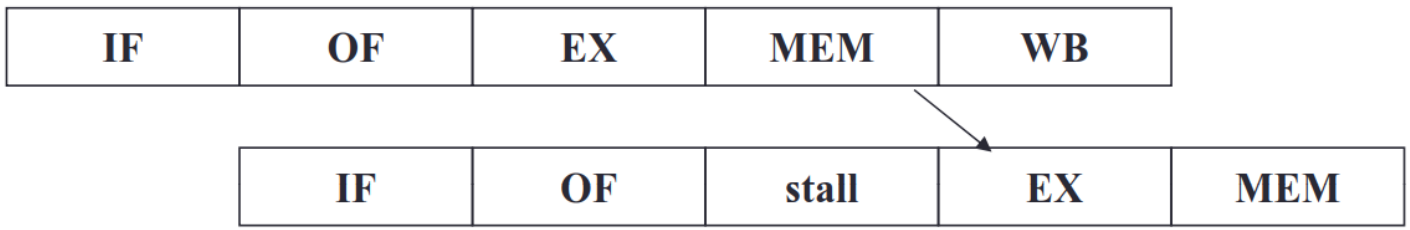




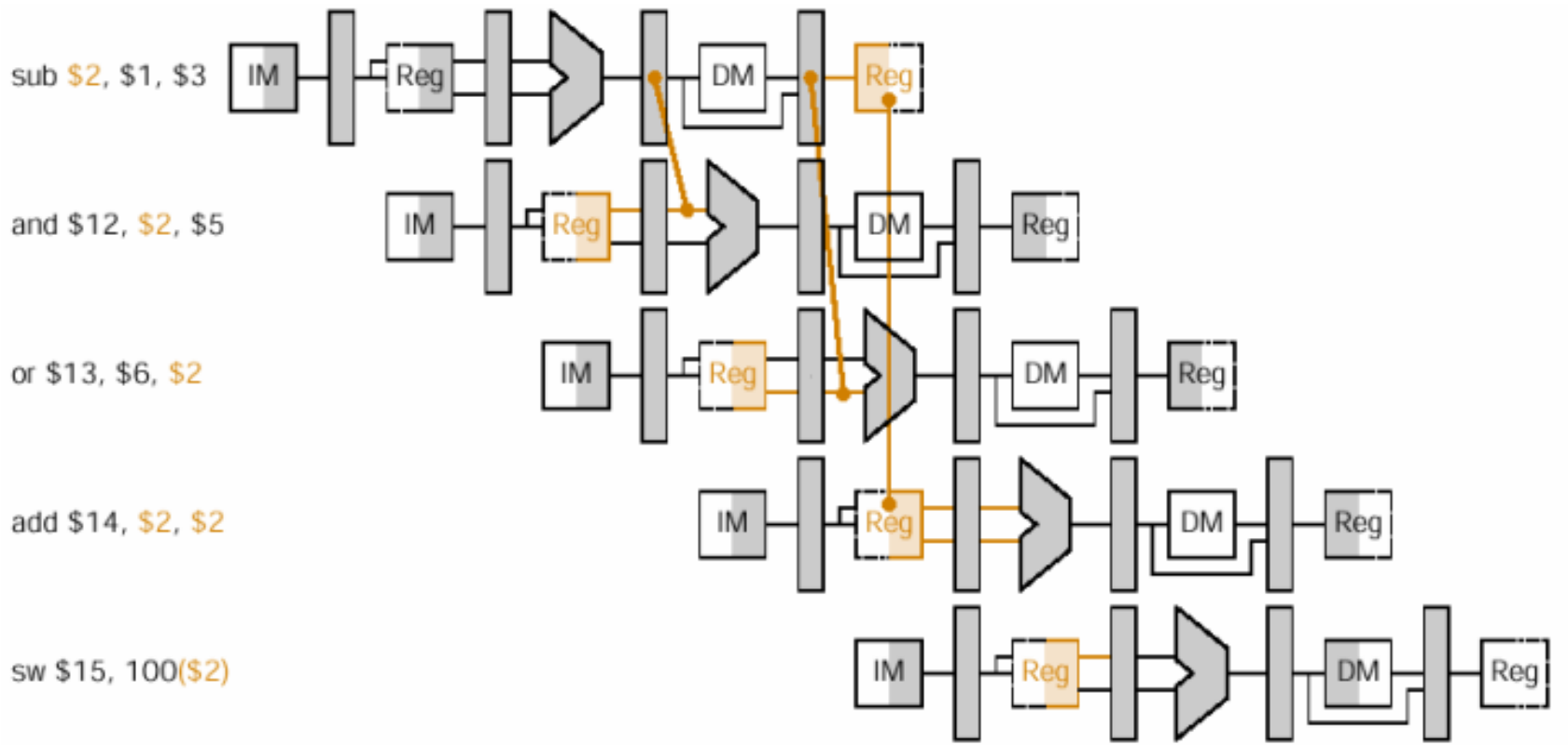
Solução 1: forwarding (adiantamento)

- Exemplo 2:
  1. LOAD **R3**, 100 (R0)
  2. ADD R1, R2, **R3**
- Forwarding: caminho interno dentro do pipeline entre a saída da memória de dados e a entrada da ULA





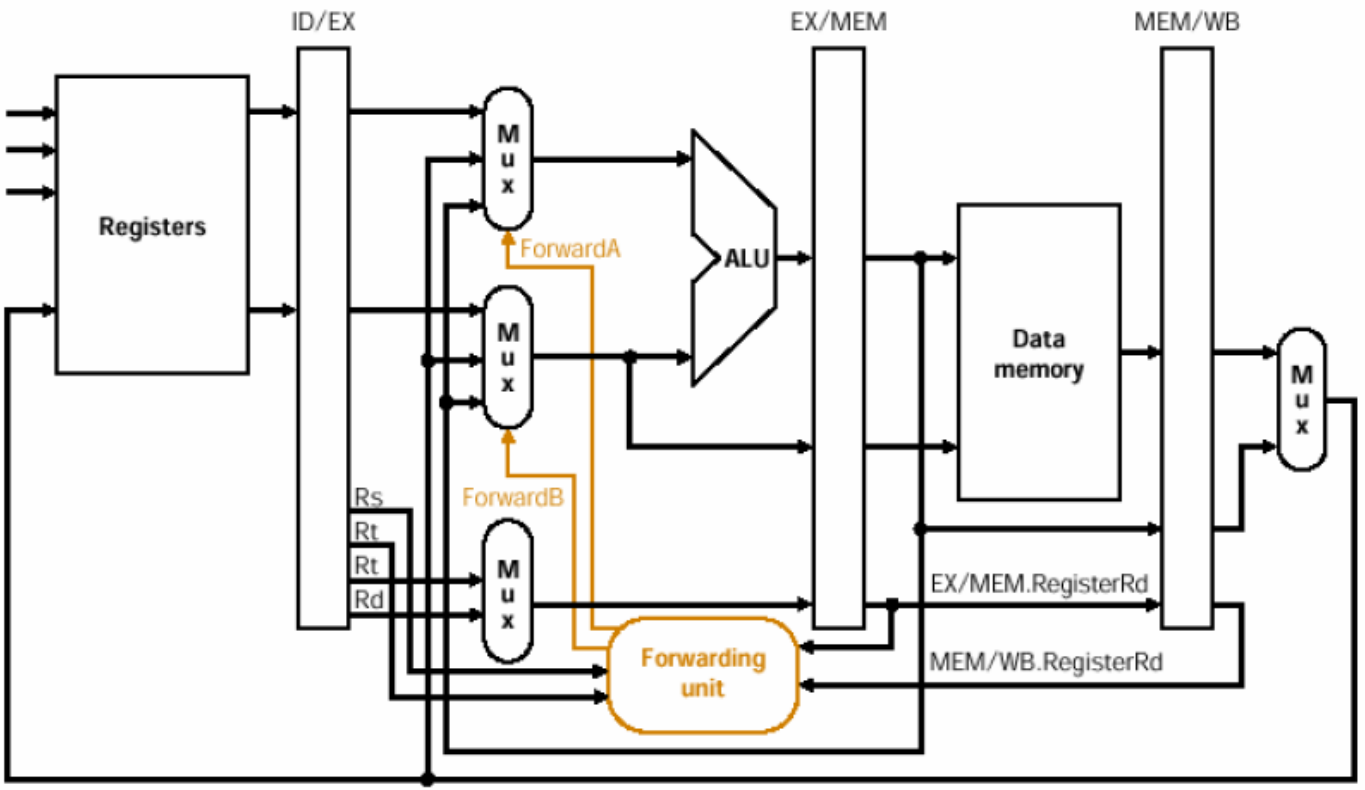
Solução 1: forwarding (adiantamento)



UNISINOS

Solução 1: forwarding (adiantamento)

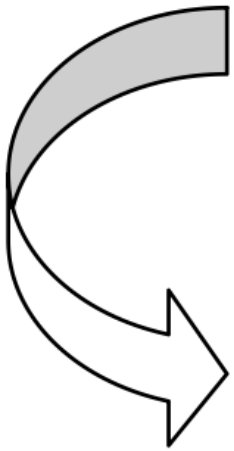
- **Implementação**: laços com multiplexadores que desviam o fluxo de dados de acordo com o controle



UNISINOS

Solução 2: reordenação do código

- Compilador busca escalonar as instruções de forma a resolver os conflitos e obter um código mais eficiente



```
lw $t0, 0($t1)      // $t1 = &v[k], $t0 = v[k]
lw $t2, 4($t1)      // $t2 = v[k+1]
sw $t2, 0($t1)      // v[k] = reg $t2
sw $t0, 4($t1)      // v[k+1] = reg $t0 (temp)
```

```
lw $t0, 0($t1)      // $t1 = &v[k], $t0 = v[k]
lw $t2, 4($t1)      // $t2 = v[k+1]
sw $t0, 4($t1)      // v[k+1] = reg $t0 (temp)
sw $t2, 0($t1)      // v[k] = reg $t2
```


UNISINOS

Antidependência

- Exemplo
  1. ADD R3, **R2**, R1 ; R3 = R2 + R1
  2. SUB **R2**, R4, 1 ; R2 = R4 – 1
- Instrução 1 utiliza operando em R2 que é escrito pela instrução 2
- instrução 2 não pode salvar resultado em R2 antes que a instrução 1 tenha lido seus opperandos
- Não é um problema em pipelines onde a ordem de execução das instruções é mantida
  - Escrita do resultado é sempre o último estágio
- Problema em processadores **superescalares**

Dependências de saídas

- Exemplo
  1. ADD **R3**, R2, R1 ; R3 = R2 + R1
  2. SUB R2, **R3**, 1 ; R2 = R3 – 1
  3. ADD **R3**, R2, R5 ; R3 = R2 + R5
- Instruções 1 e 3 escrevem no mesmo R3
- Instrução 1 tem que escrever seu resultado em R3 antes do que a instrução 3, senão o valor final de R3 ficará errado
- Também só é problema em processadores **superescalares**

Exercícios

1. O que é pipeline? Para que serve? Há alguma consequência negativa disso?
2. Por que podem existir conflitos no pipeline?
3. Quais são os tipos de conflitos existentes no pipeline? Como resolvê-los?

Exercícios

4 Monte, para o código a seguir, três pipelines diferentes:

a) considerando que não há unidades de adiantamento

b) buscando reordenar as instruções

c) utilizando unidades de adiantamento (e indicando onde são usadas)

Quantos ciclos de execução foram necessários em cada caso?

Calcule o speedup.

código
`lw  Rb, b`
`lw  Rc, c`
`add Ra, Rb, Rc`
`sw  Ra, a`
`lw  Re, e`
`lw  Rf, f`
`sub Rd, Re, Rf`
`sw  Rd, d`

Utilize 5 estágios para o pipeline. Os estágios são os seguintes: IF (busca da instrução), ID (leitura dos registradores), EX (execução na ULA), MEM (acesso à memória de dados) e WB (escrita em registradores).

Exercícios

5 Calcule o desempenho de um pipeline de 7 estágios para uma aplicação com 10 instruções e tempo de 3ns por estágio. Após, calcule o desempenho considerando que o estágio busca da instrução dure 1ns e o estágio de execução dure 4ns (sendo que o tempo dos demais estágios permance em 3ns).

Exercícios

6 Usando um desenho similar ao da figura inferior do slide 23, mostre os caminhos para adiantamento de dados para a execução das seguintes instruções:

```
add $2, $3, $4
add $4, $5, $6
add $5, $3, $4
```

Utilize 5 estágios para o pipeline. Os estágios são os seguintes: IF (busca da instrução), ID (leitura dos registradores), EX (execução na ULA), MEM (acesso à memória de dados) e WB (escrita em registradores).

Exercícios

7 Identifique todas as dependências de dados existentes no código a seguir. Quais dependências são conflitos que podem ser resolvidos com adiantamento? Faça dois pipelines: um considerando a existência e outro a não-existência de unidades de adiantamento.

```
add $2, $5, $4
add $4, $2, $5
sw  $5, 100($2)
add $3, $2, $4
```

Utilize 5 estágios para o pipeline. Os estágios são os seguintes: IF (busca da instrução), ID (leitura dos registradores), EX (execução na ULA), MEM (acesso à memória de dados) e WB (escrita em registradores).

Referências

HENNESSY, J.L. PATTERSON, D. A. Arquitetura de Computadores: uma abordagem quantitativa. Rio de Janeiro: Campus, 2003. 827p. ISBN: 85-352-110 85-35285-3.

UNISINOS