Arquitetura e Organização de Computadores

Cap 4. O Processador

Parte 6 - Pipeline

Prof. Dr. João Fabrício Filho

Universidade Tecnológica Federal do Paraná
Campus Campo Mourão
2023

O que é uma linha de montagem?

Problemas de desempenho no *datapath* monociclo

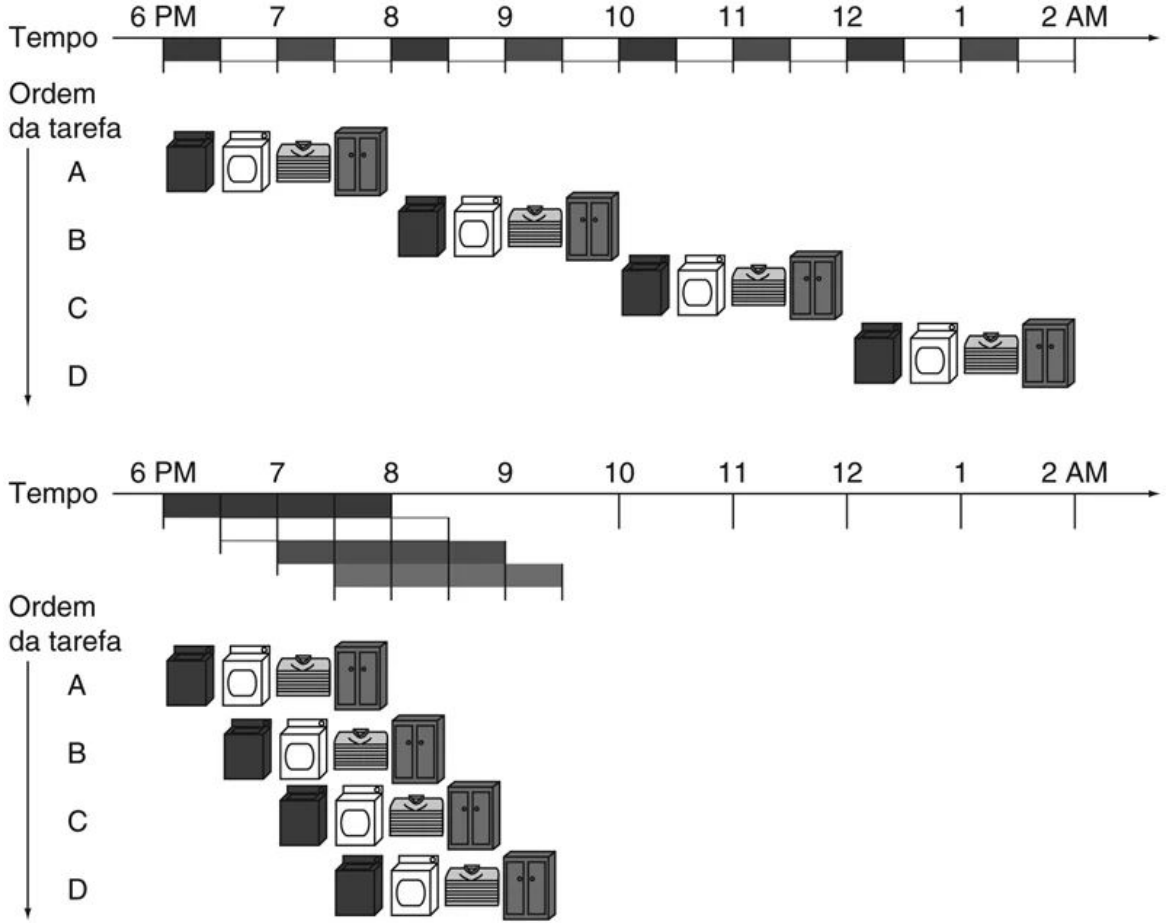
O caminho mais longo determina o período de *clock*

- Caminho crítico: instrução load
- Memória de instruções -> Banco de Registradores -> ULA -> Memória de dados -> Banco de Registradores

Não é possível variar o período para outras instruções

Viola o princípio de design: caso comum rápido

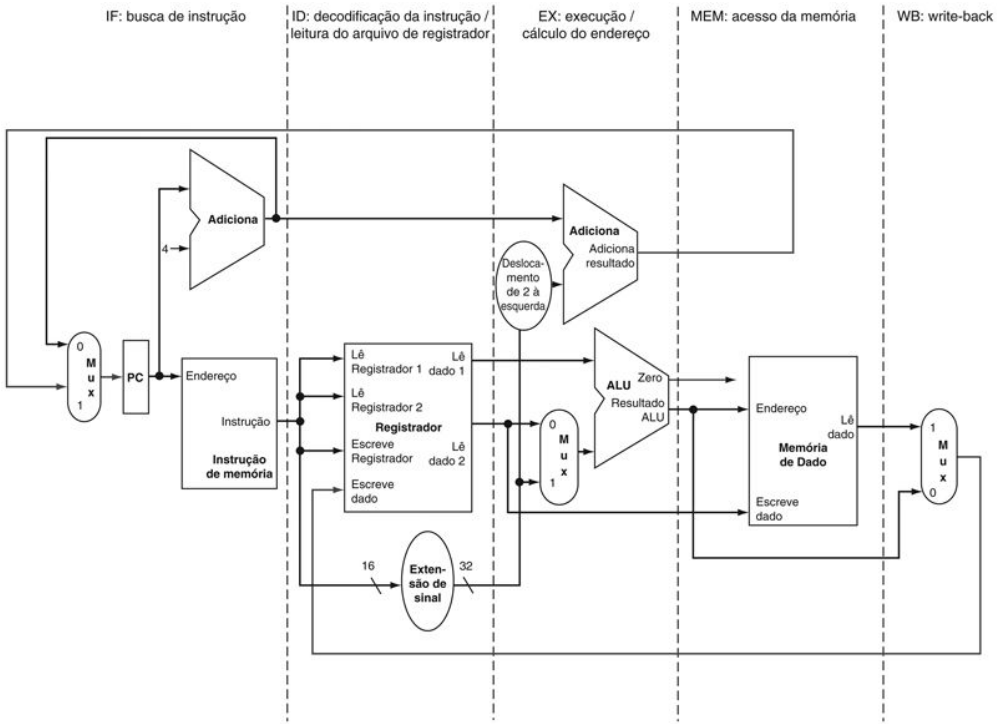
Analogia de Pipelining



Quatro tarefas
Speedup = 8 / 3,5 = 2,3x

Pipeline do MIPS

1. **IF**: Instruction Fetch
   - Busca instrução da memória
2. **ID**: Instruction Decode
   - Lê os registradores enquanto a instrução é decodificada
3. **EX**: Execute
   - Executa a operação ou calcula o endereço
4. **MEM**: Memory
   - Acessa um operando na memória de dados
5. **WB**: Write result Back
   - Escreve o resultado em um registrador



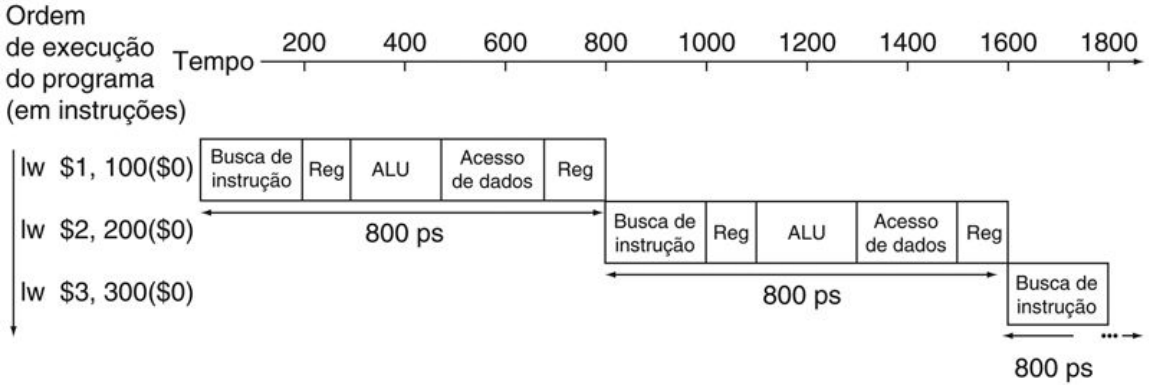
Desempenho do pipeline

- Suponha o tempo de estágios:
  - 100ps para leitura ou gravação de registrador
  - 200ps para outros estágios

- Compare o caminho de dados em pipeline com o caminho de dados monociclo

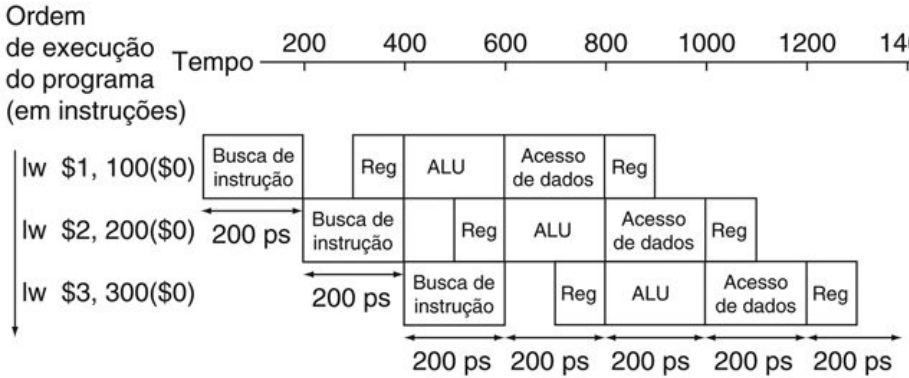
Classe de instrução	Busca de instrução	Leitura do registrador	Operação ALU	Acesso de dados	Escrita do registrador	Tempo Total
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, AND, OR, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

Caminho mais longo | Instrução lw



Monociclo
2400ps

Período de clock
800ps



Pipeline
1400ps

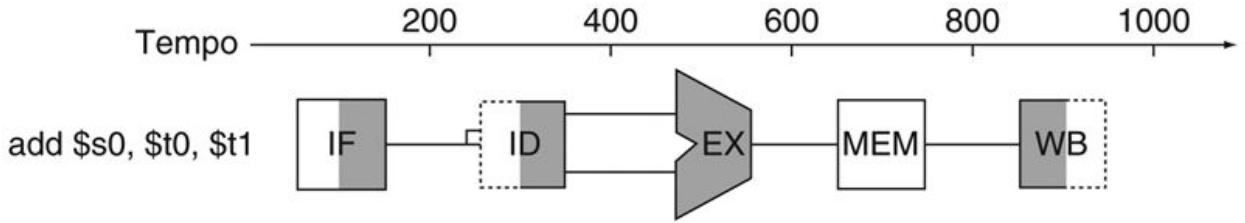
Período de clock
200ps

Speedup do pipeline

- Se os estágios estiverem equilibrados
  - Todos levam o mesmo tempo

$$\text{Tempo entre instruções}_{\text{com pipeline}} = \frac{\text{Tempo entre instruções}_{\text{sem pipeline}}}{\text{Número de estágios do pipeline}}$$

- Sem equilíbrio, o speedup é menor
- Speedup devido ao aumento do *throughput*
  - Latência para cada instrução isoladamente não diminui

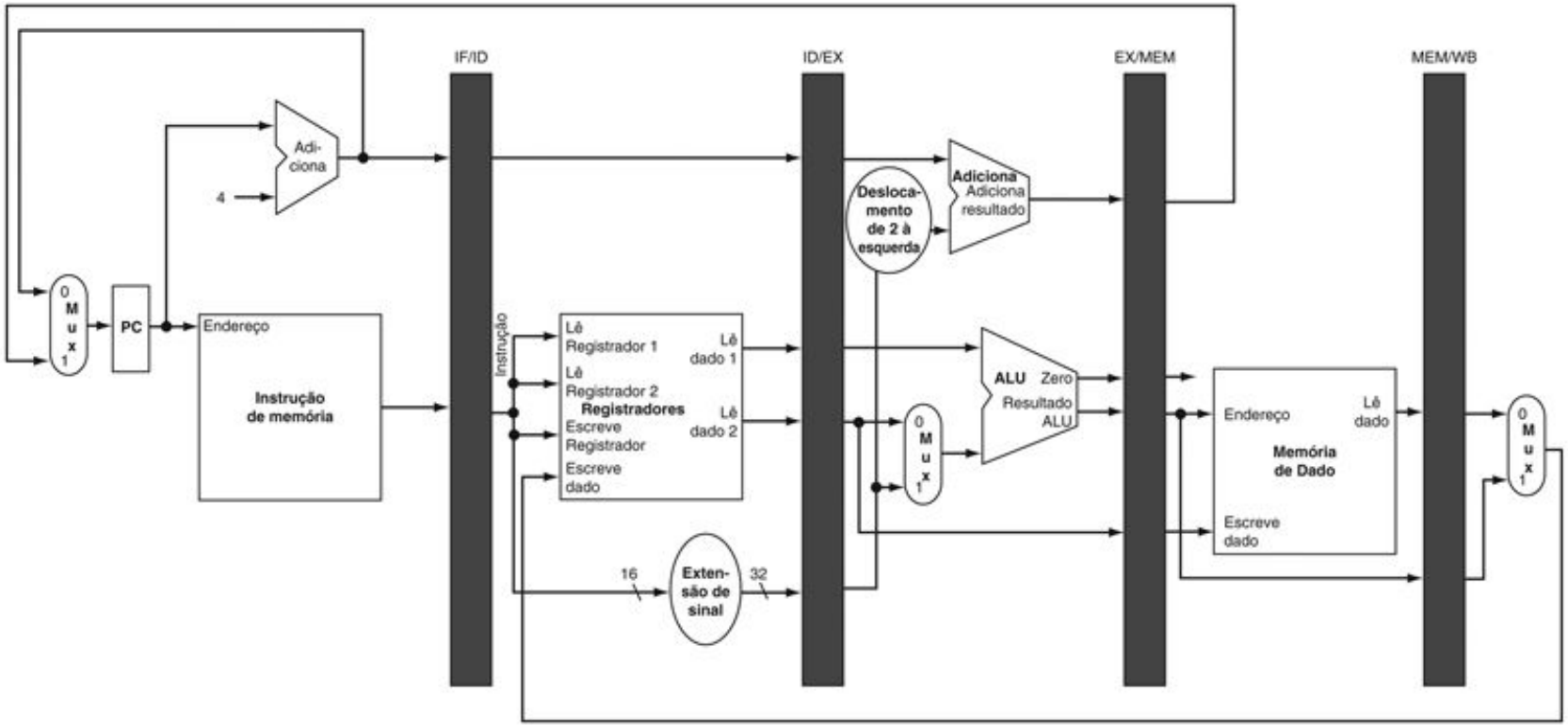


Pipelining e projeto de ISA

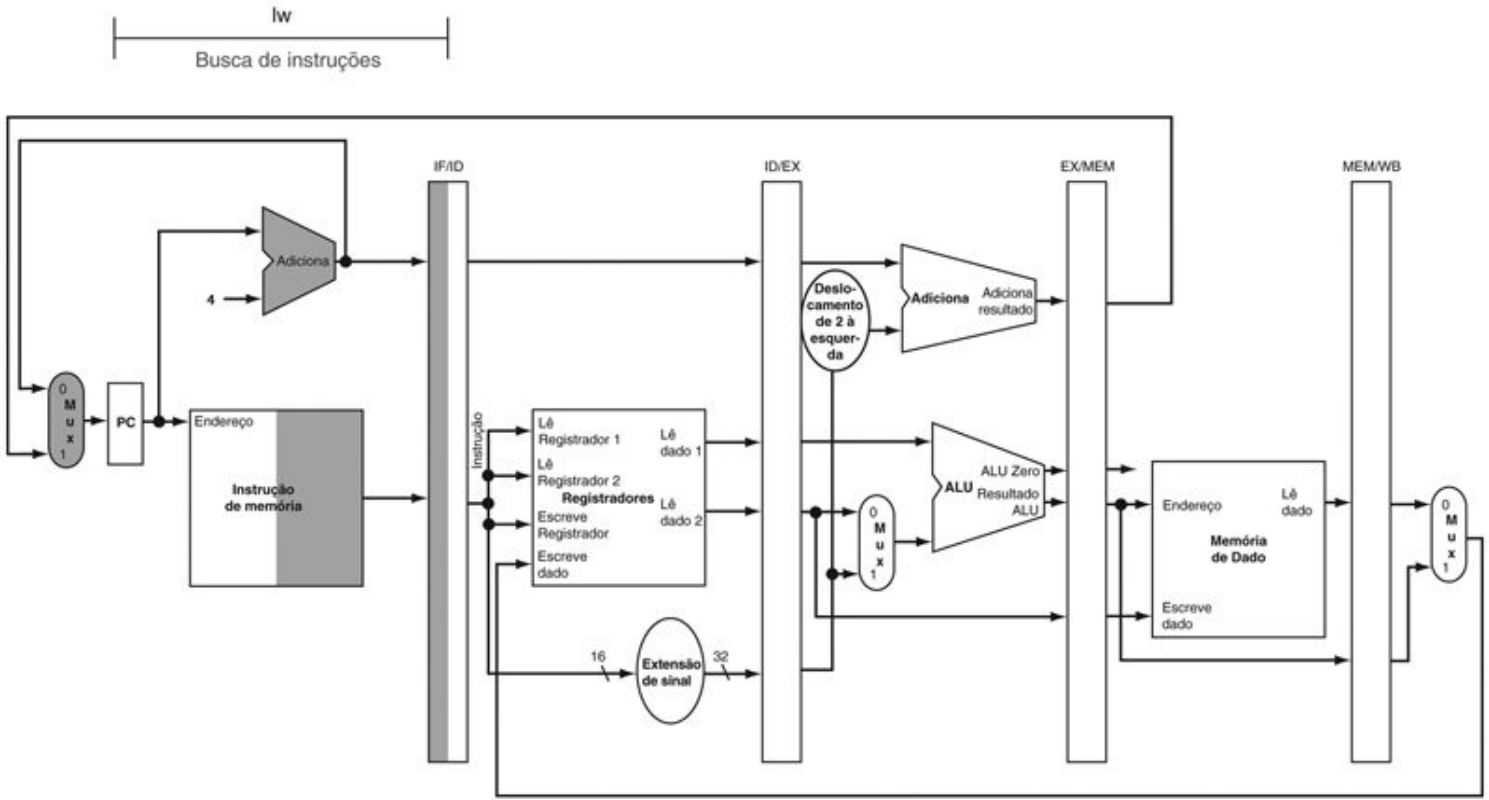
ISA do MIPS foi projetado para pipelining

- Todas as instruções possuem 32 bits
  - Mais fácil de buscar e decodificar em um ciclo
  - x86: 1 a 17 bytes
- Poucos e regulares formatos de instruções
  - Decodifica e lê registradores em um passo
- Endereçamento load/store
  - Calcula o endereço no estágio 3, acessa a memória no estágio 4
- Alinhamento de operandos de memória
  - O acesso leva apenas um ciclo

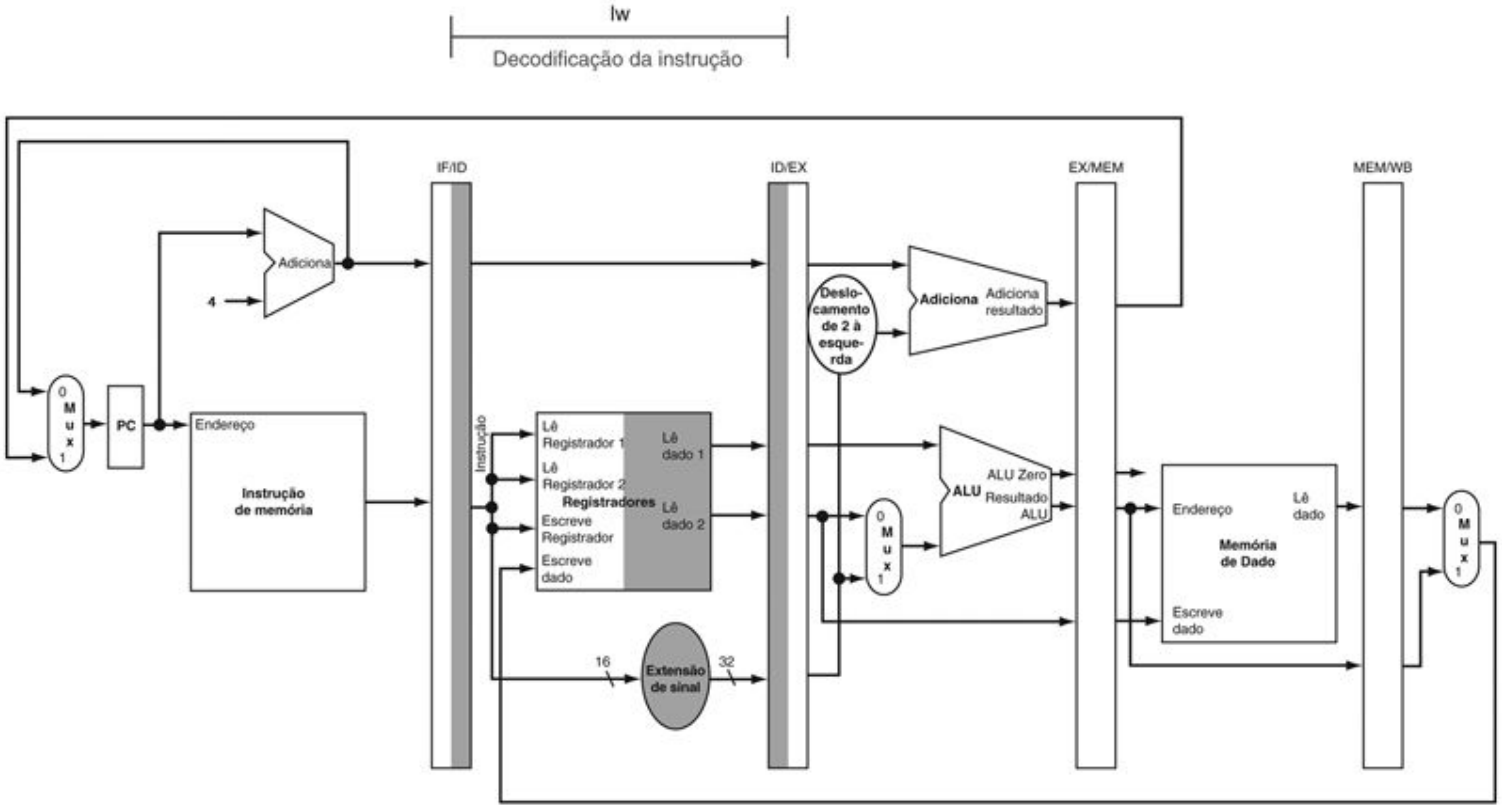
Registradores entre estágios



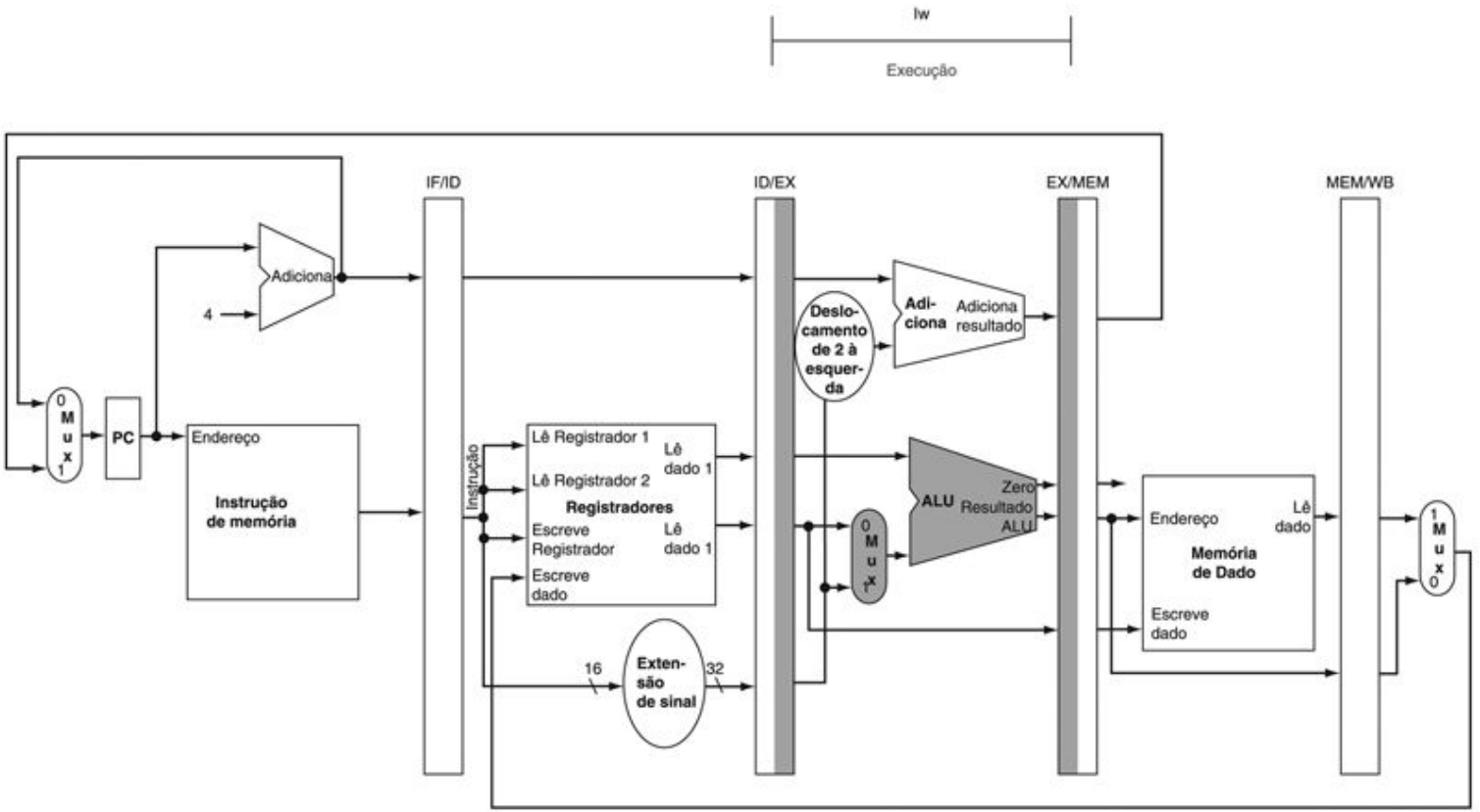
Caminho mais longo: instrução lw



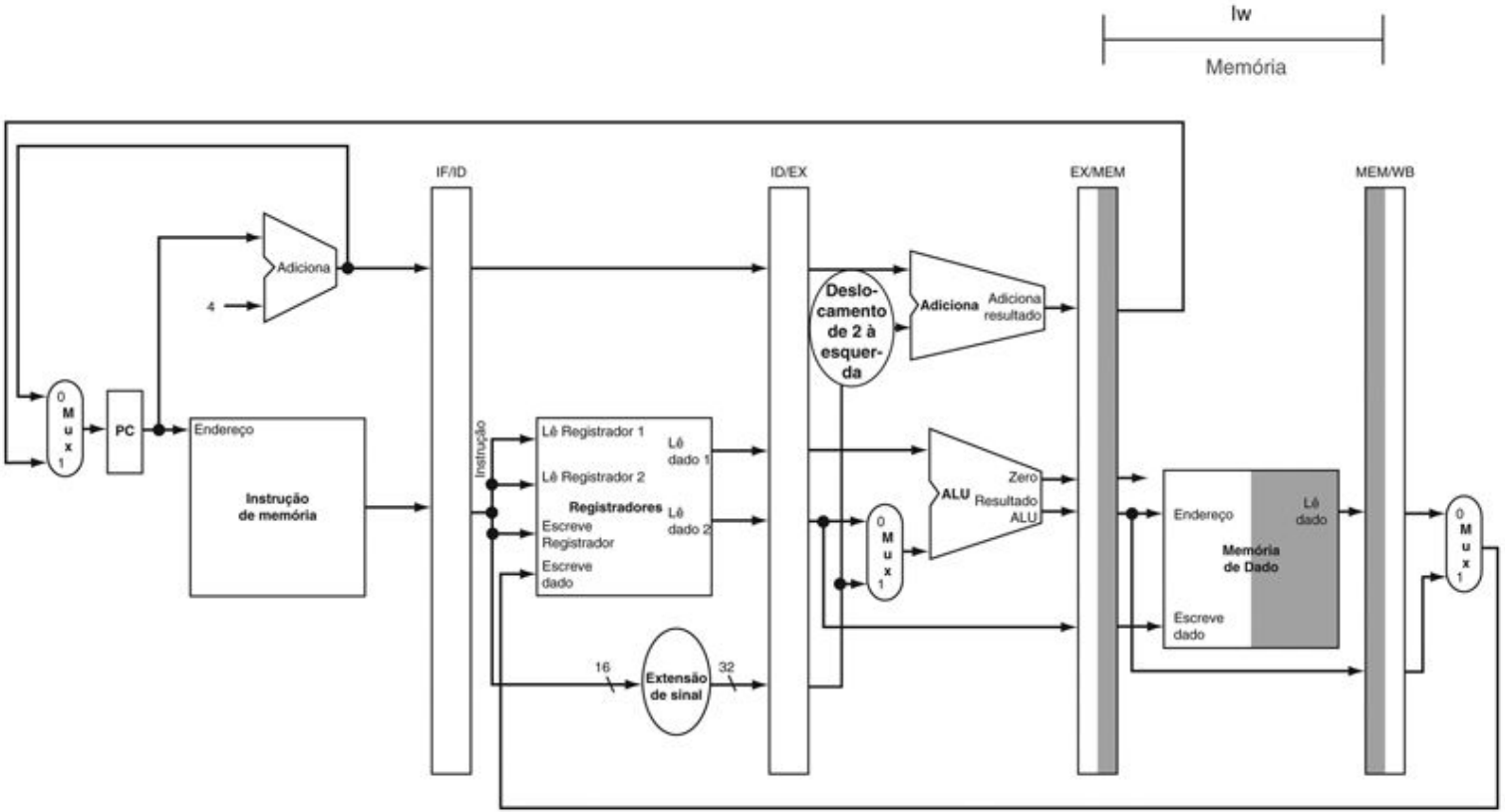
Caminho mais longo: instrução lw



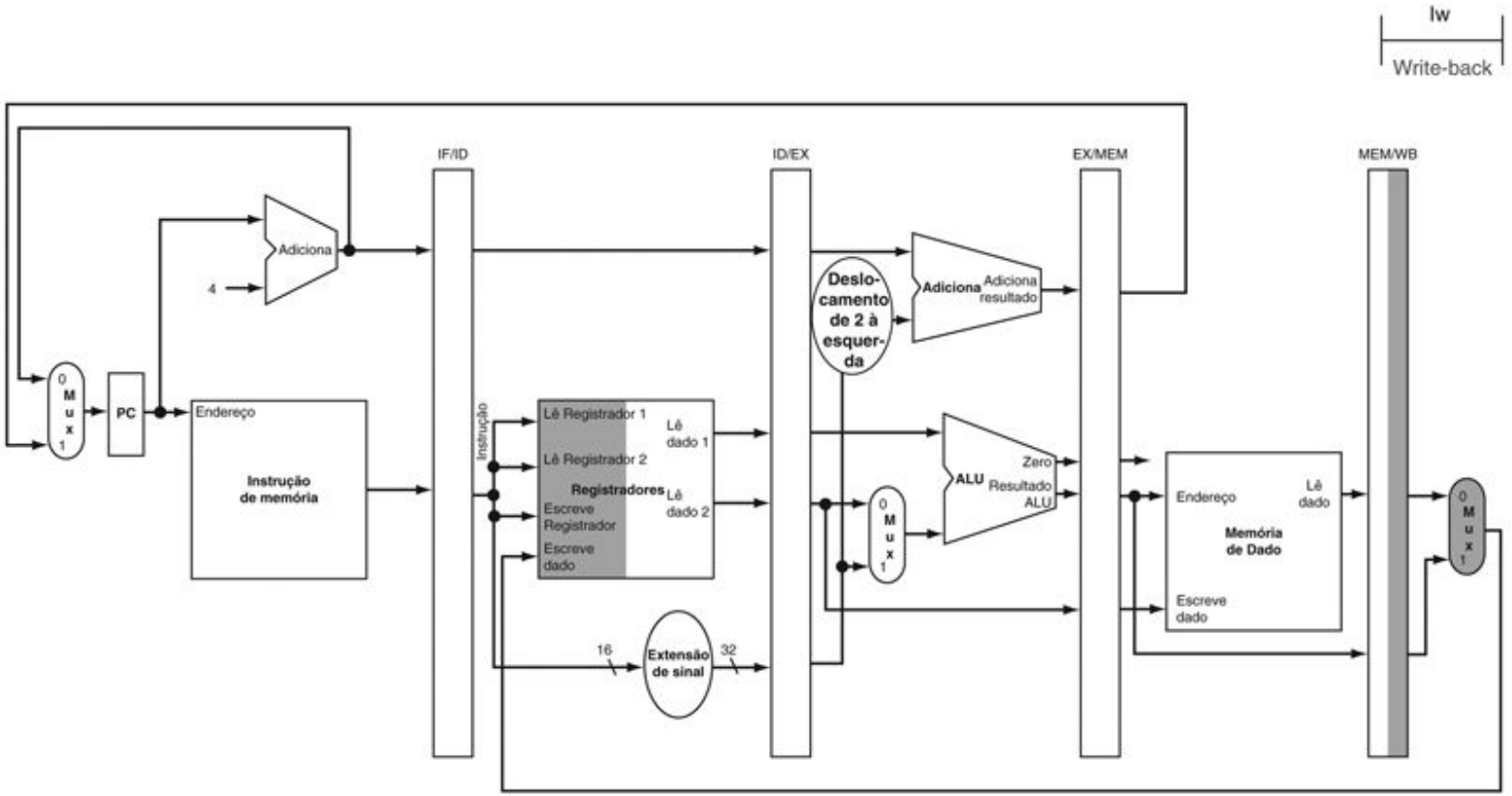
Caminho mais longo: instrução lw



Caminho mais longo: instrução lw

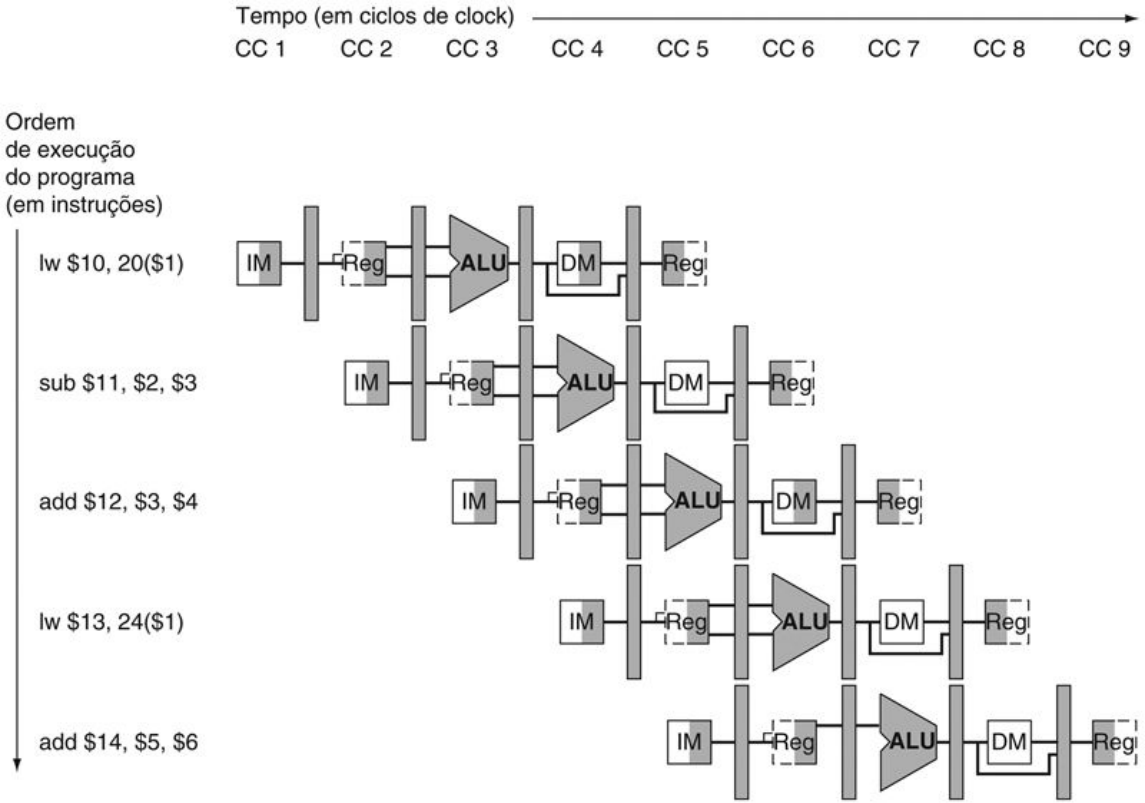


Caminho mais longo: instrução lw



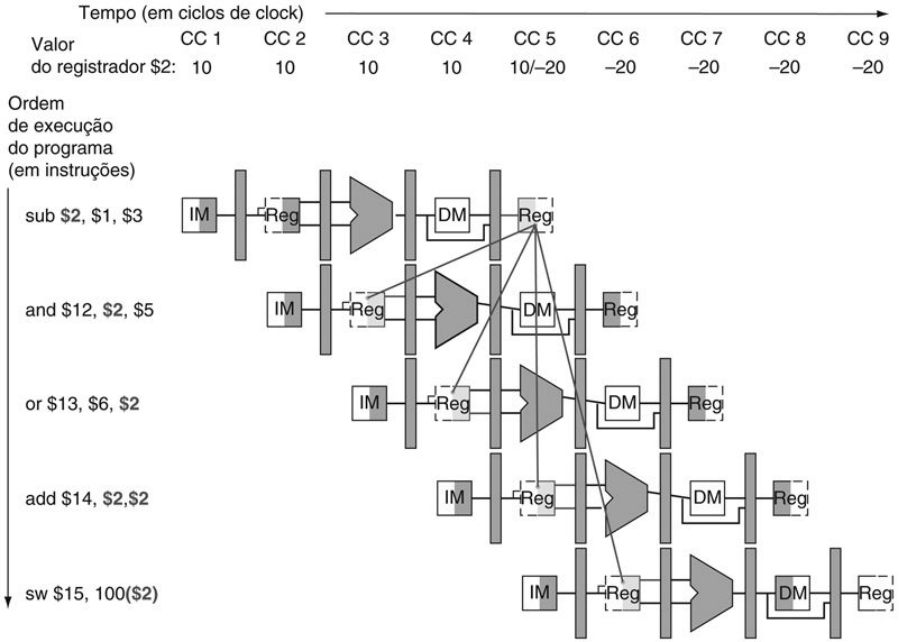
Representação gráfica multiciclo de pipeline

```
lw      $10, 20($1)
sub     $11, $2, $3
add     $12, $3, $4
lw      $13, 24($1)
add     $14, $5, $6
```



Dependência de resultados

```
sub    $2, $1,$3        # Registrador $2 escrito por sub
and    $12,$2,$5        # 1º operando ($2) depende de sub
or     $13,$6,$2        # 2º operando ($2) depende de sub
add    $14,$2,$2        # 1º e 2º operandos ($2) dependem de sub
sw     $15,100($2)      # Base ($2) depende de sub
```



Hazards

Situações que impedem o início da próxima instrução no próximo ciclo

- Hazards estruturais
  - Um recurso necessário está ocupado
- Hazards de dados
  - Precisa aguardar instruções anteriores gerarem dados para escrita ou leitura
- Hazards de controle
  - Decisão de controle depende de instrução anterior

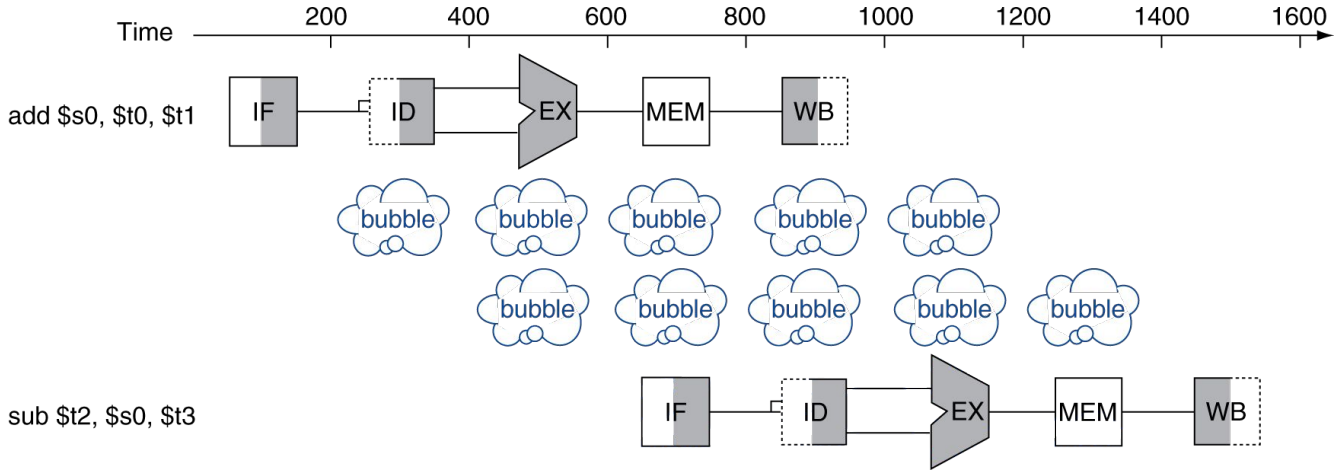
Hazard estrutural

- Conflito no uso de recursos
- Exemplo: pipeline do MIPS com uma única unidade de memória
  - Load/store requerem acesso à dados
  - A busca de instruções teria que esperar
- Caminhos de dados em pipeline requerem memórias separadas para instruções e dados
  - Ou separação de cache de instruções e dados

Hazard de dados

Quando uma instrução depende do resultado de uma instrução anterior

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```



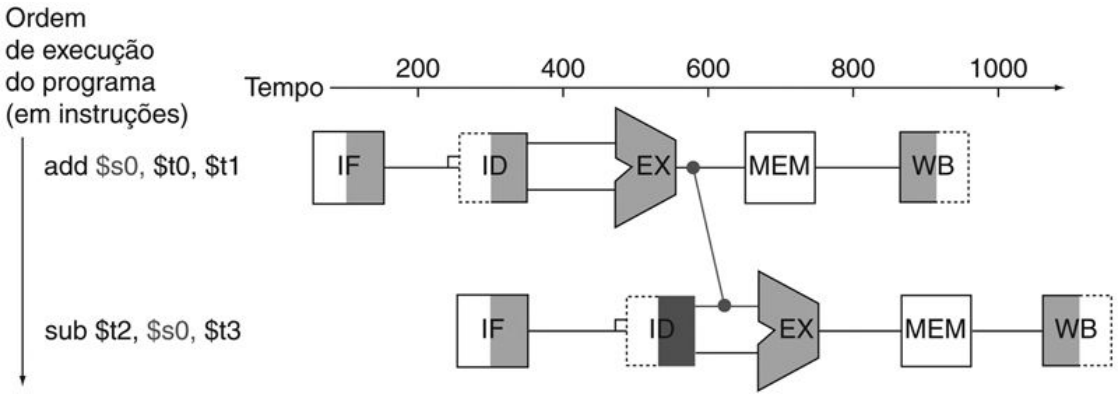
Forwarding

“What do you mean, why’s it got to be built? It’s a bypass. You’ve got to build bypasses.”
The Hitchhiker’s guide to the Galaxy

Encaminha o resultado calculado antes do final da instrução

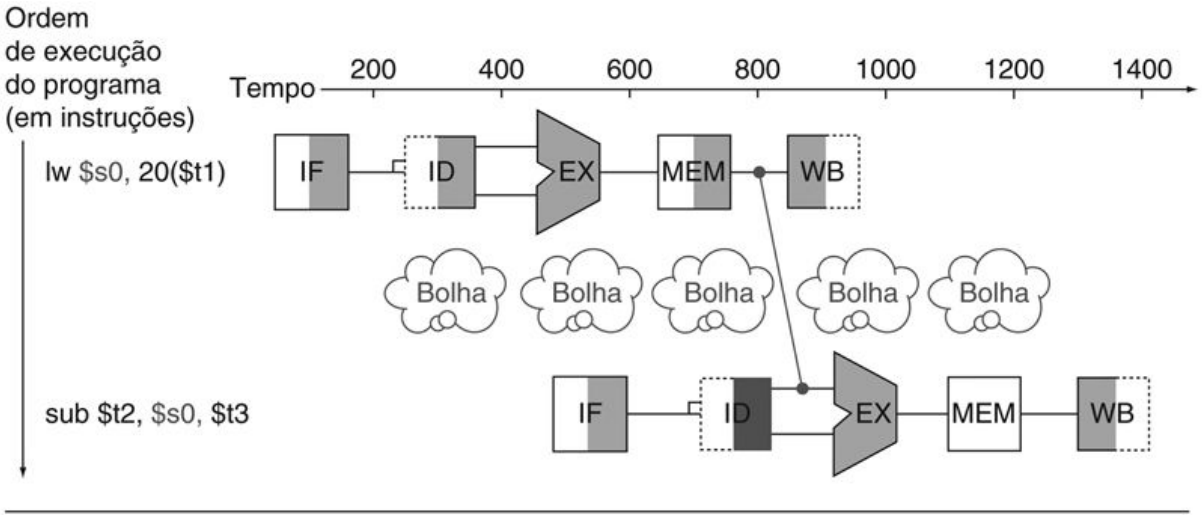
- Requer conexões extras no datapath

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```



Hazard de dados em load

- Nem sempre é possível evitar conflitos por forwarding
  - Se o valor não foi computado ainda, não é possível voltar no tempo
  - *Stall* atrasa a execução da próxima instrução



Reordenação de código para evitar *stalls*

Código C

```
a = b + e;
c = b + f;
```

```
lw  $t1, 0($t0)
lw  $t2, 4($t0)
add $t3, $t1, $t2
sw  $t3, 12($t0)
lw  $t4, 8($t0)
add $t5, $t1, $t4
```

Reordenação de código para evitar *stalls*

Código C

```
a = b + e;
c = b + f;
```

```
lw  $t1, 0($t0)
lw  $t2, 4($t0)
add $t3, $t1, $t2
sw  $t3, 12($t0)
lw  $t4, 8($t0)
add $t5, $t1, $t4
```

stall

stall

13 ciclos

Reordenação de código para evitar *stalls*

Código C

```
a = b + e;
c = b + f;
```

stall

stall

```
lw  $t1, 0($t0)
lw  $t2, 4($t0)
add $t3, $t1, $t2
sw  $t3, 12($t0)
lw  $t4, 8($t0)
add $t5, $t1, $t4
```

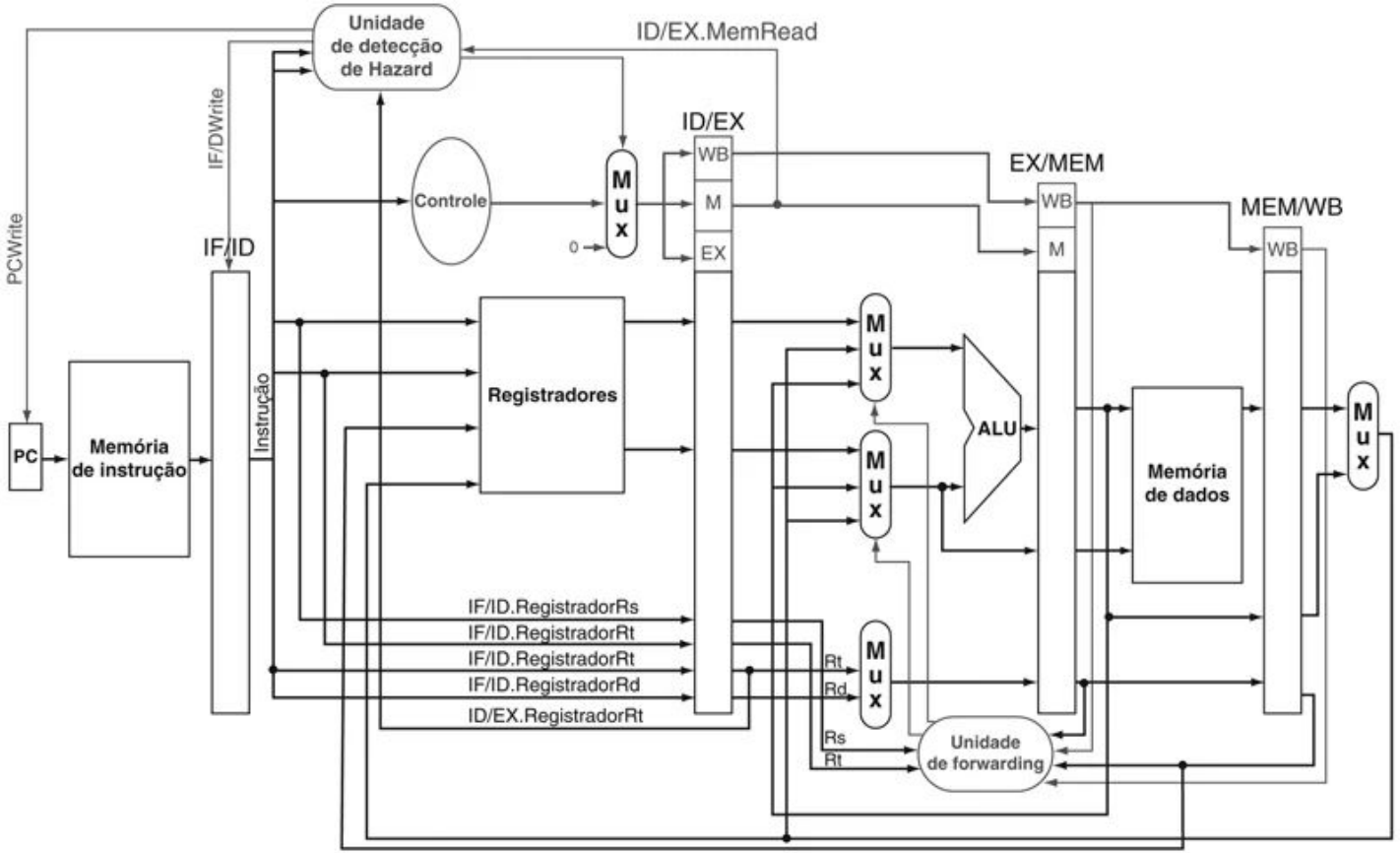
13 ciclos

reordenação

```
lw  $t1, 0($t0)
lw  $t2, 4($t0)
lw  $t4, 8($t0)
add $t3, $t1, $t2
sw  $t3, 12($t0)
add $t5, $t1, $t4
```

11 ciclos

Detecção de hazard e forwardings na arquitetura



Hazard de controle

O desvio determina o fluxo de controle

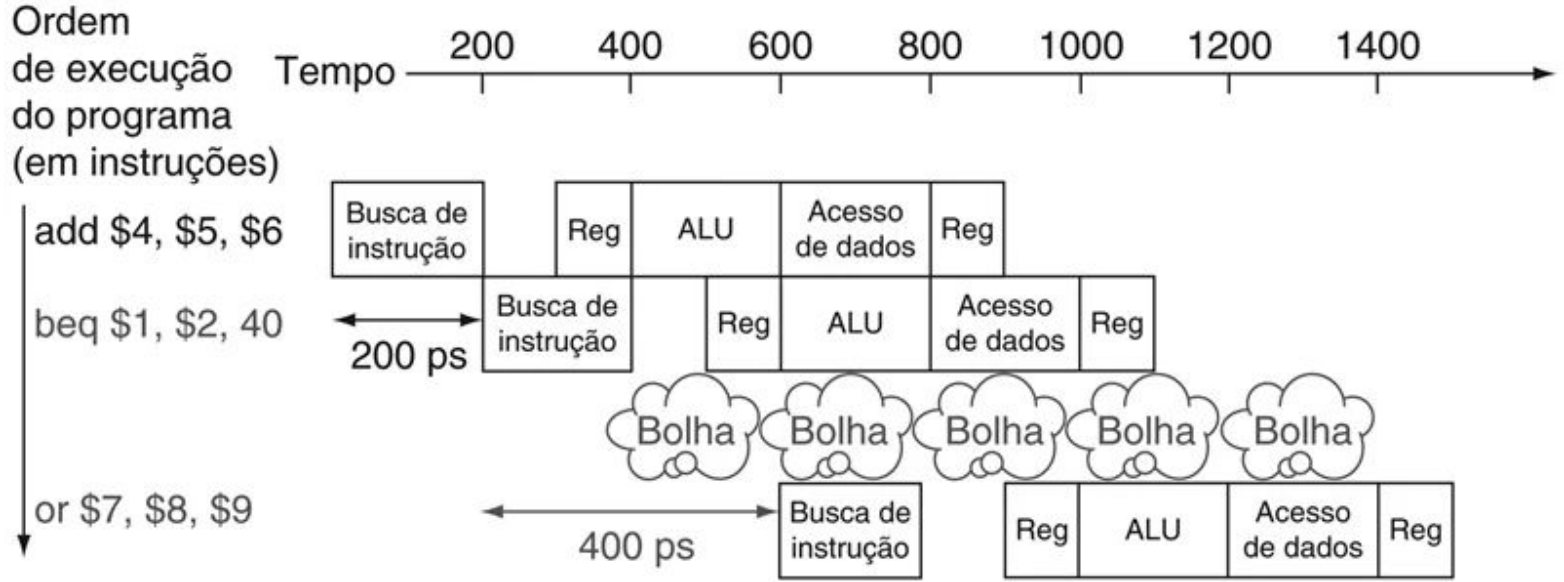
- A busca da próxima instrução depende do resultado do desvio
- O pipeline nem sempre pode buscar as instruções corretas
  - ainda trabalhando no estágio ID do desvio

No pipeline do MIPS

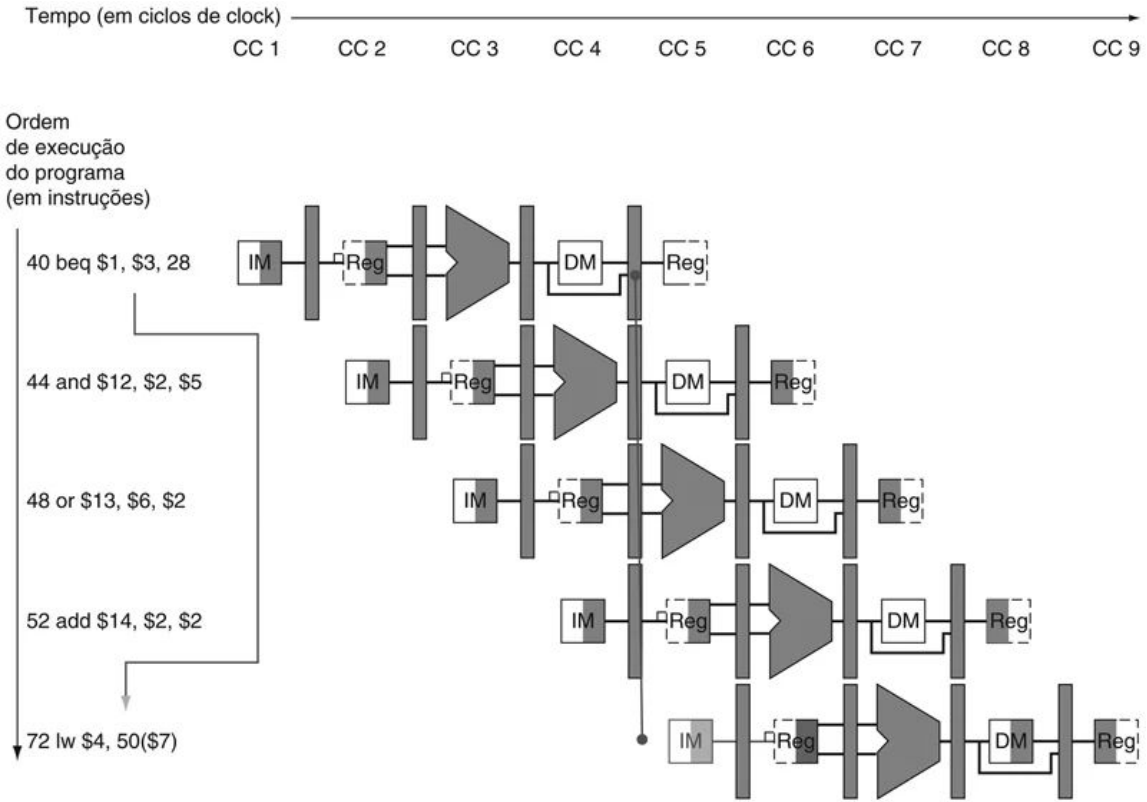
- Precisa comparar registradores e calcular o endereço destino no início do pipeline
- Hardware adicional para fazer no estágio ID

Stall de desvio

Aguarda até que o resultado do desvio seja determinado antes de buscar a próxima instrução



Impacto do pipeline em instruções branch



Previsão de desvio

Pipelines mais longos não conseguem determinar o resultado do desvio cedo

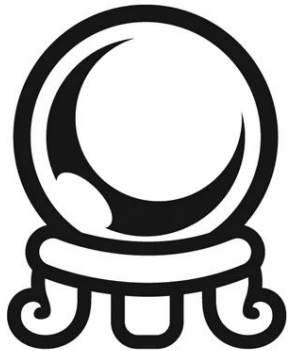
- A penalidade de parada se torna inaceitável

Previsão do resultado do desvio

- Parar apenas se a previsão estiver errada

No pipeline do MIPS

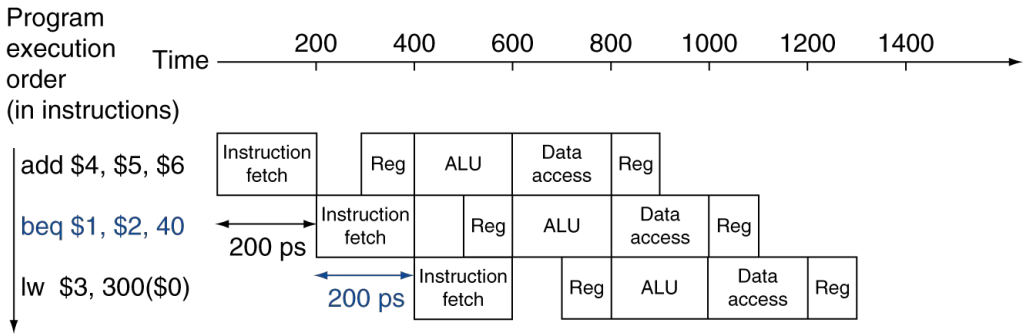
- Pode haver desvio não tomado
- Buscar instrução depois do desvio, sem atrasos



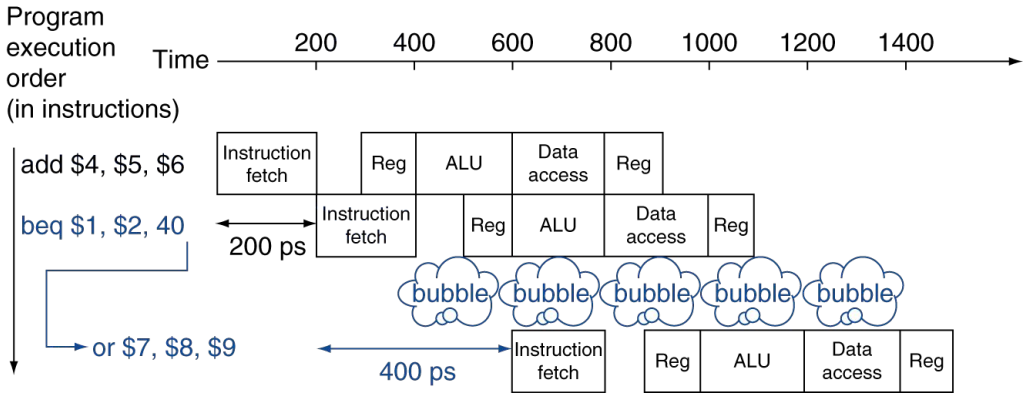
PREDIÇÃO

Impacto da previsão

Previsão correta



Previsão incorreta

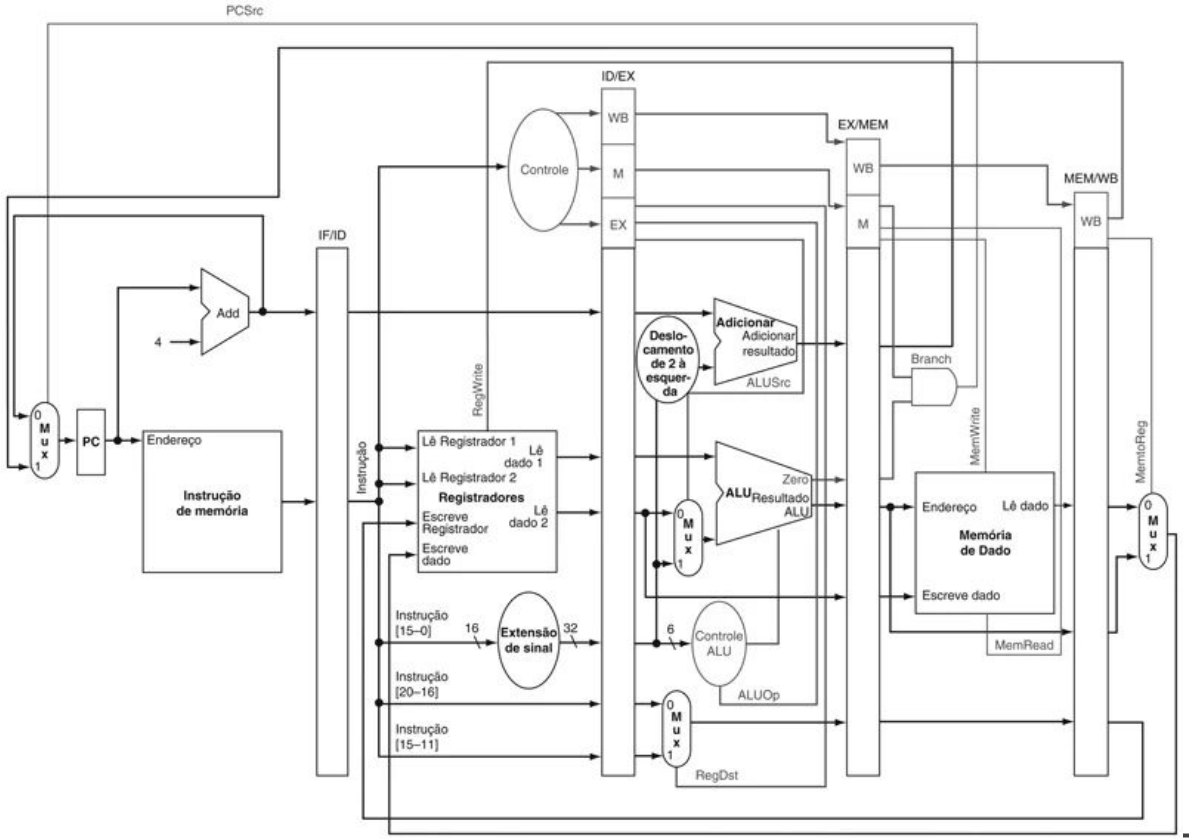


Previsão de desvio mais realista

- Previsão de desvios estática
  - Com base no comportamento típico do desvio
  - Exemplo: desvio de loop e if
    - Prever desvios para trás como tomados
    - Prever os desvios para frente como não tomados
- Previsão de desvios dinâmica
  - Hardware mede o comportamento real do branch
    - e.g., grava o histórico de cada branch
  - Assume o comportamento futuro continuará
    - Quando errado, acontece stall em um re-fetch e o histórico é atualizado

Controle com pipeline

Sinais de controle derivam da instrução, assim como na implementação monociclo

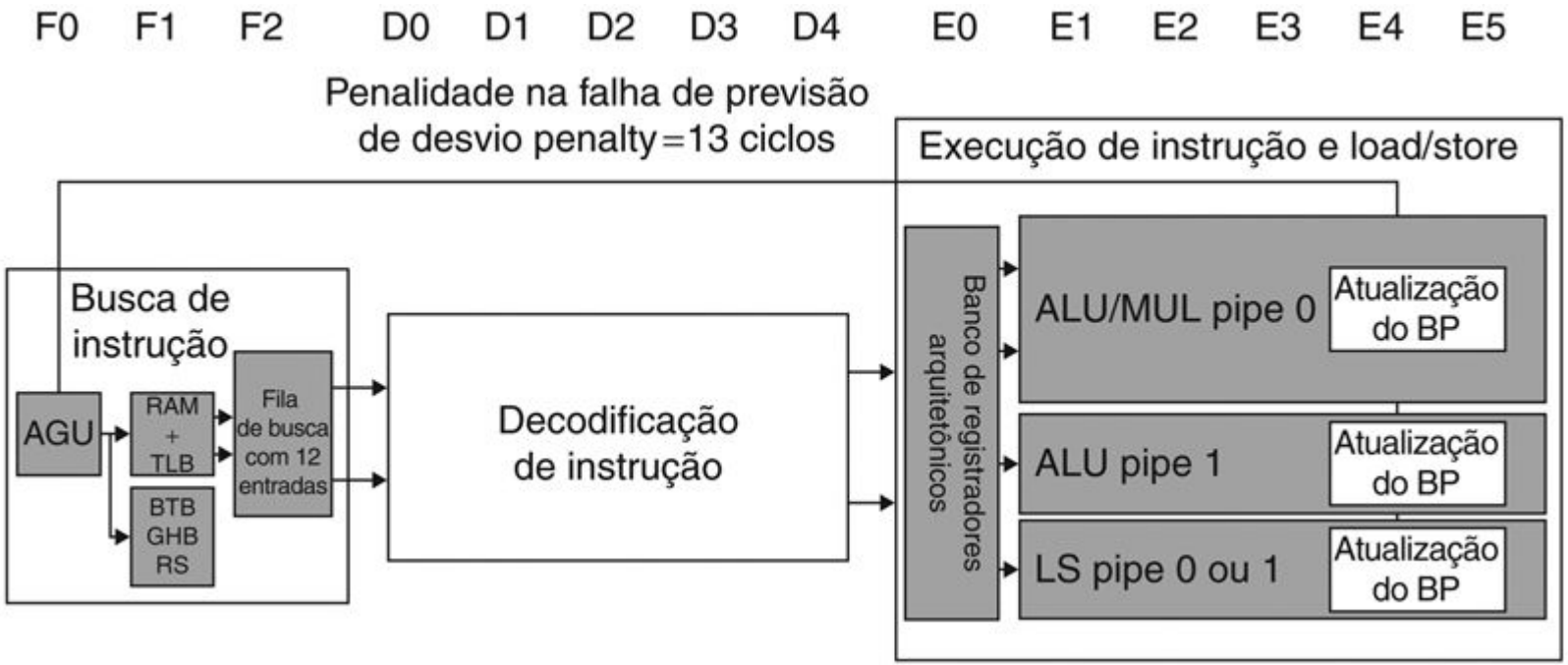


Paralelismo a nível de instrução (ILP)

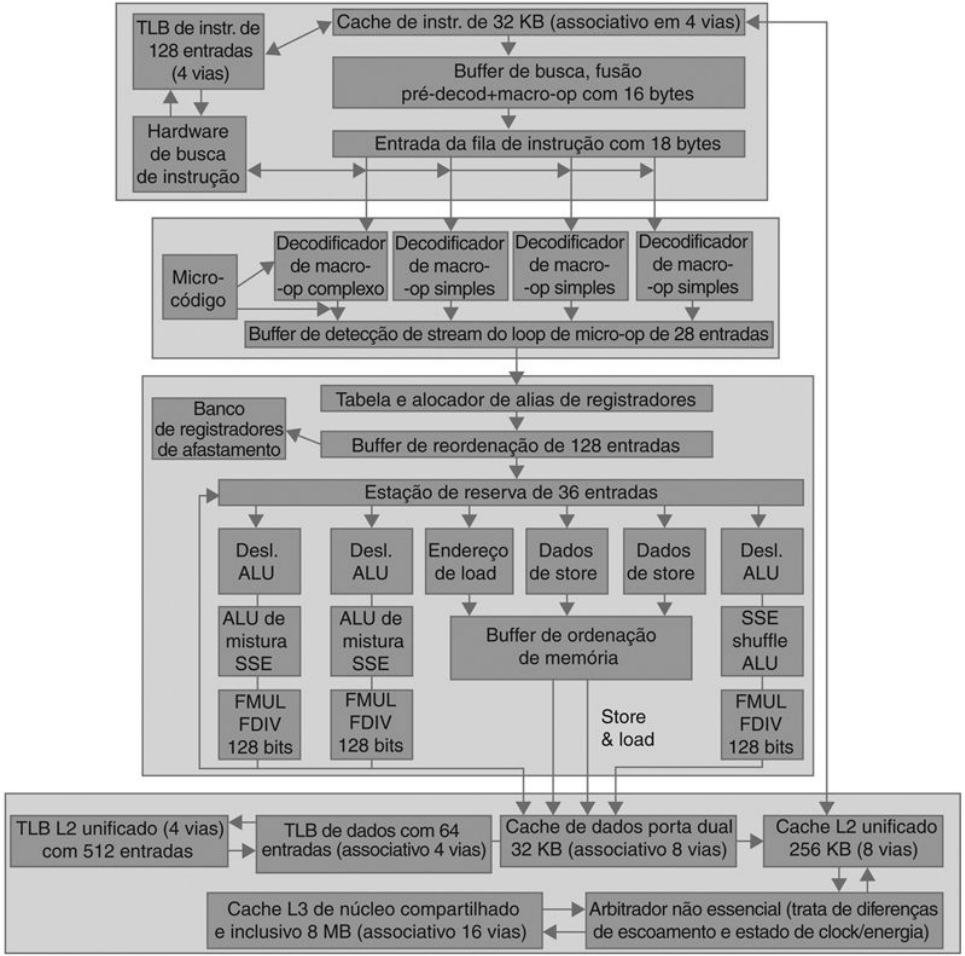
- Pipeline: executa múltiplas instruções em paralelo
- Aumentar ILP:
  - Pipeline mais profundo (deep pipe)
    - Menos trabalho por estágio -> período de clock mais curto
  - Múltiplos despachos
    - Replicar estágios de pipeline -> múltiplos pipelines
    - Iniciar várias instruções por ciclos de clock
      - CPI < 1, podemos utilizar instruções por ciclo (IPC)

ARM Cortex A8

Pipeline de 14 estágios



Intel Core i7 920



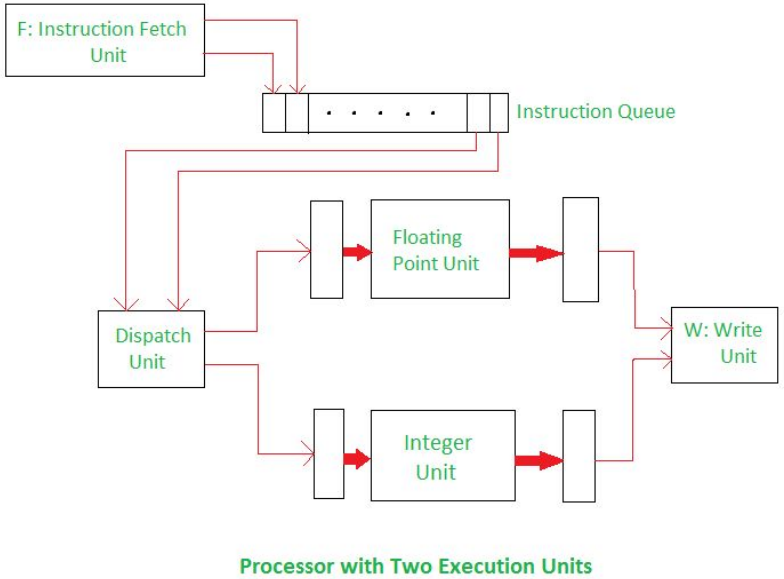
Otimização de pipeline

Loop unrolling

- Replica o corpo do loop para expor mais instruções ao pipeline
  - Reduz o overhead de controle do loop
- Usa diferentes registradores por replicação
- Evita dependências que atrapalhem o paralelismo
  - Store seguido de load do mesmo registrador

Processadores superescalares

- Várias cópias do datapath
- Múltiplas unidades de processamento
- Várias instruções abrangendo o pipeline
- Objetivo: Alcançar CPI<1
- Desempenho em detrimento da eficiência



Processor with Two Execution Units

Arquiteturas superescalares

- Hazards são evitados por seleção e ordenamento de instruções
- Compilador pode intercalar instruções que usam unidades diferentes
  - Várias técnicas de compilação podem tirar vantagem da arquitetura superescalar
- Problemas
  - Escalonamento
  - Penalidade de hazards

Resumo de Pipeline

The BIG Picture

Pipelining aumenta o desempenho aumentando o *throughput* das instruções

- Instruções executadas em paralelo
- Cada instrução com a mesma latência

Sujeito a hazards

- Estrutural, dados, controle

O design do ISA afeta a complexidade da implementação do pipeline

Stalls e desempenho

The BIG Picture

Stalls reduzem o desempenho

- Mas são necessários para ter resultados corretos

O compilador pode reordenar o código para evitar hazards e stalls

- Requer conhecimento da estrutura do pipeline

Falácias

Pipeline é fácil

- A ideia básica é fácil
- O diabo está nos detalhes
  - e.g., detectar hazard de dados

Pipeline é independente de tecnologia

- Então por que não fizemos pipeline sempre?
- Mais transistores deixam técnicas sofisticadas factíveis

Armadilhas

Design ruim de ISA pode deixar pipeline mais difícil e complexo

- e.g., conjuntos complexos (VAX, IA-32)
  - Overhead significativo para fazer pipeline
  - Microoperações IA-32
- e.g, modos de endereçamento complexos
  - Efeitos colaterais de atualização em registradores
  - Indireção em memória
- e.g., Desvios atrasados
  - Pipelines avançados possuem grandes slots de atrasos

Considerações finais

- ISA influencia no controle do pipeline
- Datapath e controle influenciam no design do ISA
- Pipelining melhora o throughput de instruções por meio do paralelismo
  - Mais instruções completadas por segundo
  - A latência para cada instrução não é reduzida
- Hazards: estrutural, dados, controle
- Dependências limitam o paralelismo

Capítulo 4 - O processador

Exercícios sugeridos:

Seção “verifique você mesmo”: 4.1, 4.2, 4.3, 4.4 4.5, 4.6, 4.7, 4.8

Exercícios do final do capítulo: 4.5, 4.7,4.8, 4.9, 4.10

Referências

Materiais disponibilizados pelos professores Lucas Wanner, Paulo Gonçalves, Ricardo Pannain e Rogério Ap. Gonçalves.

Patterson, David A. Hennessy, John L. Organização e Projeto de Computadores. Disponível em: Minha Biblioteca, (5a. edição). Grupo GEN, 2017.

Slides do livro PATTERSON, David A. e HENNESSY, John L. *Computer Organization and Design Risc-V Edition: The Hardware Software Interface*. Estados Unidos, Ed. Morgan Kauffman, 2020.