# Fundamentos de Arquitetura de Computadores

Tiago Alves

Faculdade UnB Gama
Universidade de Brasília

## Objetivo

Apresentar a ideia de duas técnicas para aumento de desempenho de arquiteturas de processadores:

- **Pipeline**: uma técnica de projeto onde o hardware processa mais de uma instrução de cada vez sem esperar que uma instrução termine antes de começar a próxima.
- **Superescalar**: Uso eficiente de múltiplas unidades funcionais.

## Pipelining: Conceito Básico

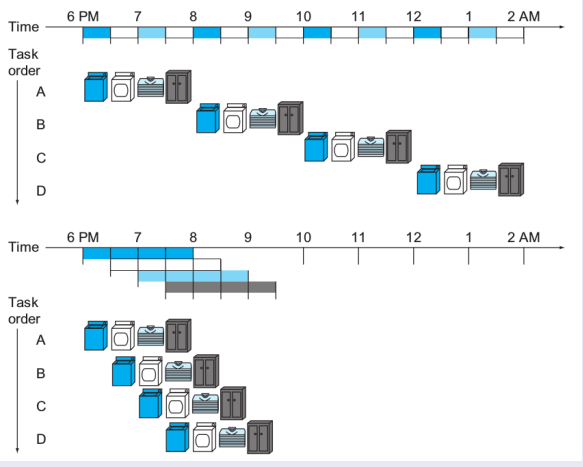
Lavar roupas!


Sequência:

1. Inserir na lava roupas uma carga de roupas sujas;
2. Quando a lava roupas concluir a lavagem, transferir a carga de roupas lavadas na secadora;
3. Quando a secadora concluir a secagem, colocar a carga de roupas secas na mesa e organizar as peças: dobrar e empilhar.
4. Quando a organização estiver concluída, guardar as roupas no armário.

## Pipelining: Conceito Básico

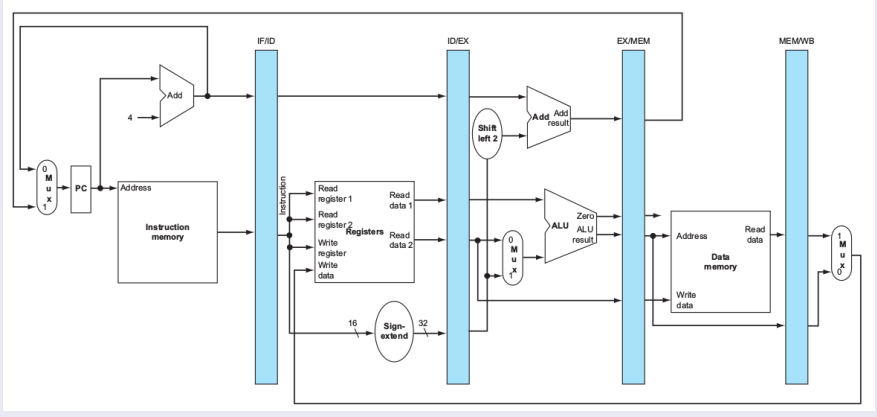
Lavar roupas!



## Metodologia do Módulo

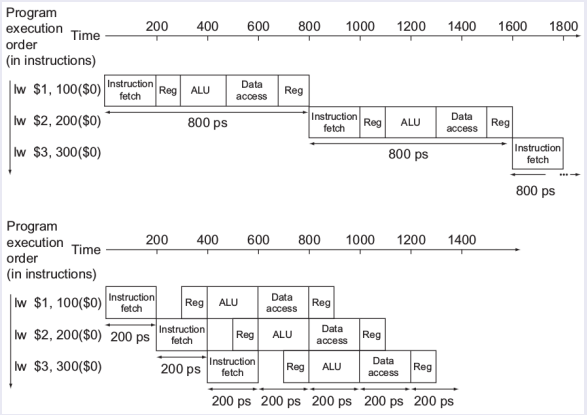
- A ideia é criar um Pipeline para o MIPS;
- Atenção especial para 8(oito) instruções: `lw, sw, add, sub, and, or, slt e beq`.
- Mesmas etapas das implementações Uniciclo e Multiciclo: 5 etapas.
  - Busca de Instruções (IF),
  - Decodificação da Instrução (ID),
  - Execução ou Cálculo de endereço (EX),
  - Acesso à memória de dados (MEM) e
  - Escrever o Resultado em um registrador (WB).

## Versão em Pipeline do Caminho de Dados



## Uniciclo vs. Pipeline

### Comparação de Desempenho




Estágios:

Acesso à Memória (Registradores): 100ps

Demais: 200ps

## Análise

Latência: 5 ciclos.

Vazão: 1 instrução/ciclo

Qual a aceleração ideal?

- Para estágios balanceados, condições ideais e grande número de instruções:

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

- Porque estágios balanceados? (lacunas)
- Por que condições ideais? (hazards)
- Porque grande número de instruções? (overhead final)

## Efeito da Exploração do Pipeline

Melhorar o desempenho aumentando a vazão das instruções, em vez de diminuir o tempo de execução de uma instrução individual.

## Pipeline no MIPS

O que facilita o pipeline:

- Todas instruções têm o mesmo comprimento.

  Obs.: IA-32 instruções de 1 a 17 bytes (mais desafiador!)

- Poucos formatos de instruções.

  Obs: Tipo-R, I , J operandos "quase" nas mesmas posições

- Operandos em memória só aparecem em `loads` e `stores`.

  Obs.: Pode-se usar a ULA para cálculo de endereço. O mesmo não vale para busca de operandos da ULA da memória (IA-32)

O que complica (hazards):

- Risco Estruturais
- Riscos de Controle
- Riscos de Dados

## Risco de Dados

Pipeline precisa ser interrompido por que uma etapa precisa esperar até que outra seja concluída.

- O que fazer se 1 pé de meia seu foi lavado junto com as roupas de seu colega?
- Exemplo:

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

Soluções:

- inserir 3 "bolhas". (dado $s0, só escrito no fim da etapa 5)
- execução fora de ordem (compilador e processador)

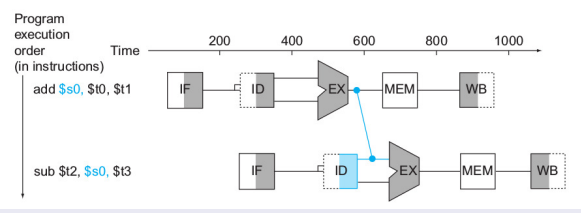
Solução: **forwarding** ou **bypassing**.

## Forwarding

Observação: não é necessário esperar que a instrução termine antes de tentar resolver o risco de dados.

- O que fazer se 1 pé de meia seu foi lavado junto com as roupas de seu colega?
- Exemplo:

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



## Forwarding

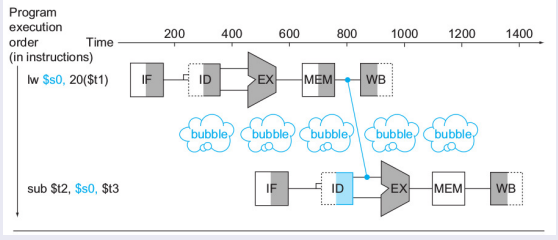
E se fosse um load?

- Pipeline Stall
  Ex.: C:

```
A=B+E;
C=B+F;
```

Assembly:

```
lw    $t1,0($t0)
lw    $t2,4($t0)
add $t3,$t1,$t2
sw    $t3,12($t0)
lw    $t4,8($t0)
add $t5,$t1,$t4
sw    $t5,16($t0)
```




- **Q**: Quantos riscos/hazards? Como melhorar? Quanto melhora?

## Forwarding

**A**: Nas duas instruções add.

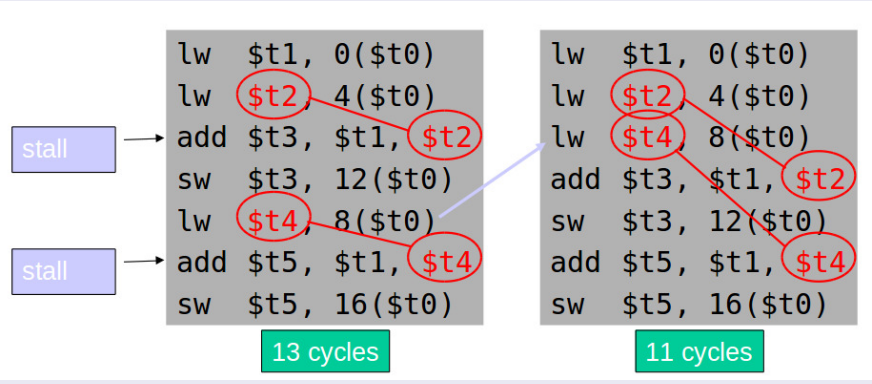
- **Forwarding**: passar o resultado à frente
- **Bypassing**: passar o resultado pelo banco de registradores para a unidade desejada.

Código melhorado:

```
lw $t1, 0($t0)
lw $t2, 4($t1)
lw $t4, 8($t1)
add $t3, $t1, $t2
sw $t3, 12($t0)
add $t5, $t1, $t4
sw $t5, 16($t0)
```

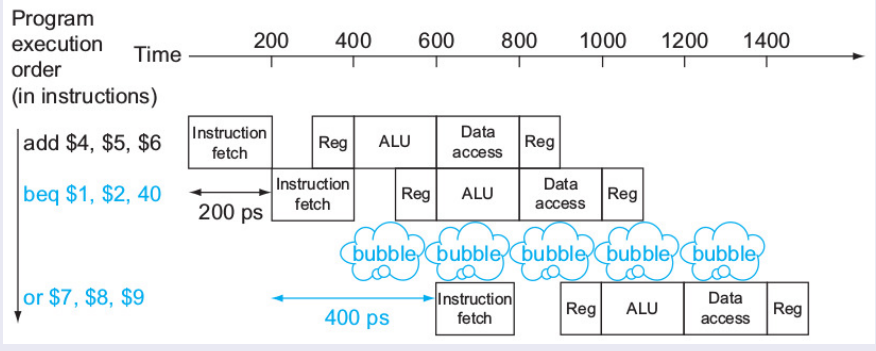
São 2 ciclos de relógio a menos!

## Forwarding


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
sw $t5, 16($t0)
13 cycles
lw $t1, 0($t0)
lw $t2, 4($t0)
lw $t4, 8($t0)
add $t3, $t1, $t2
sw $t3, 12($t0)
add $t5, $t1, $t4
sw $t5, 16($t0)
11 cycles

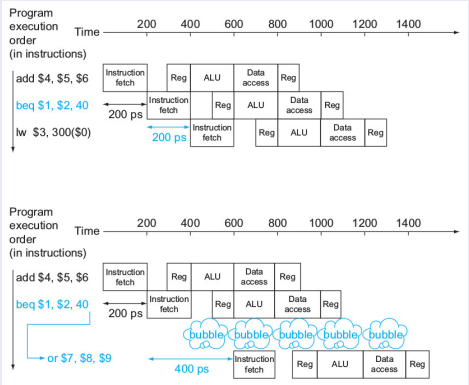
## Risco de Controle

Necessidade de tomar uma decisão com base nos resultados de uma instrução enquanto outras estão sendo executadas.



## Risco de Controle

Prevendo desvios: não tomado e tomado



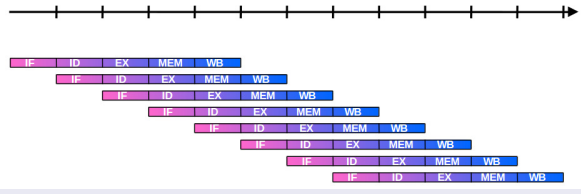

Formas Dinâmicas: Endereço é anterior? Prevê tomar o desvio (implementação de loops)

Armazenamento do histórico para decisão mais adequada.

Delayed Branch: desvio adiado (eficiente para pequenos desvios). Executa a próxima instrução (montador). Abordagem MIPS!

## Aceleração decorrente do Pipeline

**Q**: A velocidade máxima de uma máquina com pipeline é de uma instrução por ciclo de clock?



## Aceleração decorrente do Pipeline

Atrasos: reescalonamento de instruções.

```
lw $t0, 0($t1)
lw $t2, 4($t1)
sw $t2, 0($t1)
sw $t0, 4($t1)
```

Acrescentar um *branch delay slot*

- A próxima instrução após o desvio é sempre executada.
- Confiar no compilador para preencher o slot com alguma coisa útil.

**Superscalar**: iniciar mais de uma instrução no mesmo ciclo.

## Aceleração decorrente do Pipeline

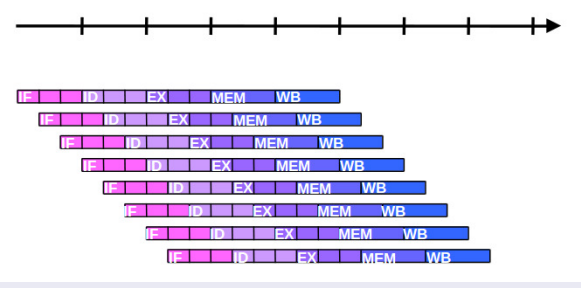
Duas técnicas para iniciar múltiplas instruções por ciclo de clock:

- arquiteturas superpipeline
- arquiteturas superescalares.

## Aceleração decorrente do Superpipeline

- A arquitetura Superpipeline subdivide cada estágio do pipeline em subestágios e multiplica o clock internamente.
- Cada (sub)estágio continua executando uma instrução por clock. Mas como o clock interno é multiplicado, o pipeline pode aceitar duas ou mais instruções para cada clock externo.

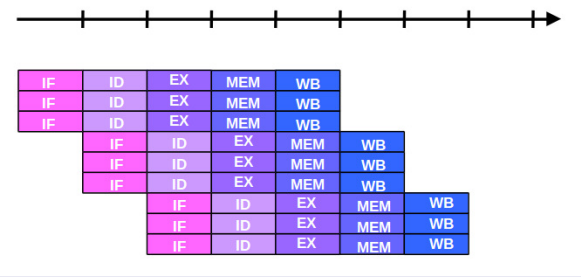
## Aceleração decorrente do Superpipeline



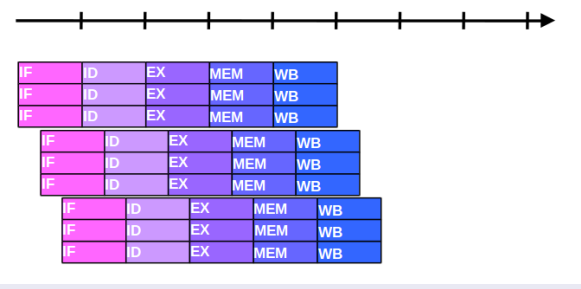
## Aceleração decorrente de Estrutura Superescalar

- A arquitetura superescalar contém múltiplas unidades de execução que são capazes de fazer a mesma coisa.
- Isto permite ao processador executar várias instruções similares concorrentemente, pelo roteamento (escalonamento) das instruções às unidades de execução disponíveis.

## Aceleração decorrente de Estrutura Superescalar



## Aceleração decorrente de Estrutura Superescalar e Superpipeline



## Execução dinâmica: escalonamento

O hardware executa o escalonamento

- O hardware tenta encontrar instruções para executar
- É possível a execução de instruções fora de ordem
- Execução especulativa e predição dinâmica de desvios.

Todos processadores modernos são bastante complicados

- Compaq/DEC Alpha 21264: 9 estágios de pipeline, 6 instruction issue
- Intel: Pentium III : 10 estágios de pipeline

  Pentium IV: 20 estágios de pipeline (nível de microinstruções)

  Pentium D: 31 estágios (Maior frequência, maior num. bolhas)

  Core2 : 14 estágios Core i7 : 24 estágios

  Athlon 64: 12 estágios
- PowerPC e Pentium: tabela de histórico de desvios
- O papel do compilador é muito importante na busca por desempenho.

## Pipeline com Escalonamento Dinâmico

Três unidades primárias de um pipeline com escalonamento dinâmico.