Introdução ao Processamento Paralelo

Prof. Rômulo Calado Pantaleão Camara

Carga Horária: 2h/60h

Introdução

- ✓ Crescente aumento de desempenho dos PCs (máquinas convencionais).
- ✓ Existem aplicações que requisitam de mais desempenho.
  - Exemplos:
    - simulações físicas;
    - matemática computacional;
    - previsão de tempo;
    - procura de petróleo entre outras.

Introdução

- ✓ Uso de PCs convencionais → Seria necessário várias semanas ou meses para executar tais aplicações.
  - Em alguns casos, haveria falta de memória.
- ✓ Exemplo:
  - Compro o melhor PC no mercado e utilizo um simulador para prever o tempo daqui a **três** dias.
  - O resultado pode ficar pronto daqui a **sete** dias.

Introdução

- ✓ Área da computação que trata destes problemas → **Processamento de alto desempenho**.
- ✓ Como resolver problemas semelhantes ao da simulação da previsão de tempo?
  - Simplificar o modelo → Resultados menos precisos.
  - **Arquiteturas paralelas ou especiais.**

Introdução

- ✓ **Arquiteturas paralelas** → Obtém melhor desempenho replicando o número de unidades ativas (normalmente processadores).
- ✓ **Problemas.**
  - O aumento do número de processadores torna o sistema computacional mais complexo.
    - **A programação torna-se complexa** → Particionar e alocar cada parte do programa às unidades ativas.

Processamento Paralelo

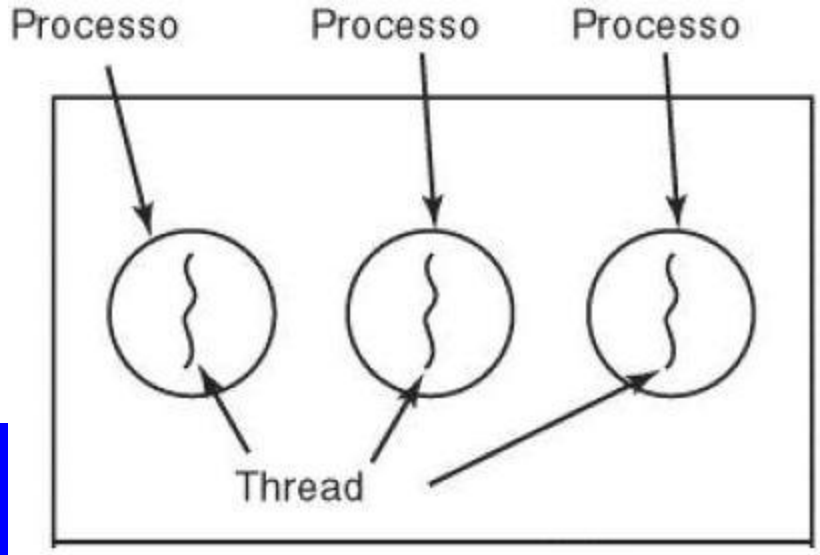
- ✓ O tipo de processamento que ocorre nestas máquinas → **Processamento Paralelo**.

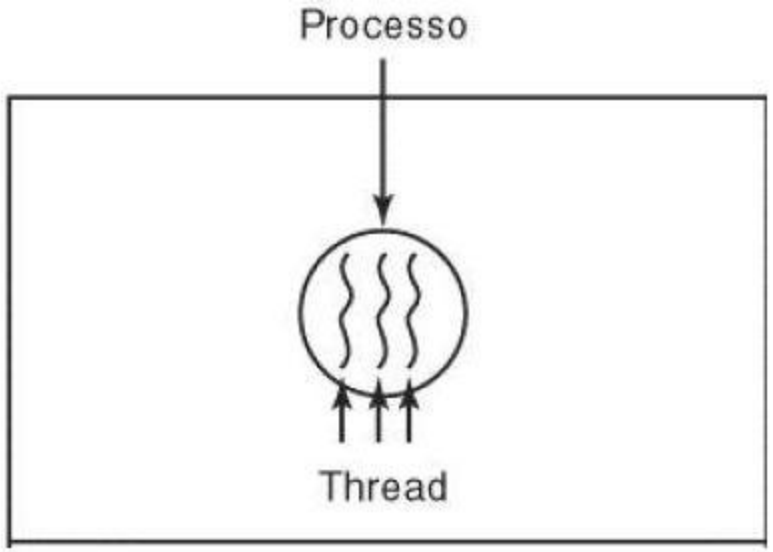
- ✓ **Definição:** várias unidades ativas colaboram na resolução de um mesmo problema com o objetivo de reduzir o tempo total de execução.

- ✓ Os programas devem ser preparados para serem executados em computadores paralelos.

Processamento Paralelo

- ✓ Exemplos de programas paralelos:
  - Uma aplicação escrita em *C* ou java com várias *threads*.
  - Uma aplicação escrita em *C* que foi dividida em vários processos que se comunicam.





Processamento Paralelo

- ✓ **Motivação para o uso de processamento paralelo:**
  - **Desempenho** → Reduzir o tempo de execução.
  - **Tolerância à falhas** → Vários processadores podem realizar um mesmo cálculo (Reduz a probabilidade de falhas.).
  - **Aproveitar recursos**.

Processamento Paralelo

- ✓ Como aumentar o desempenho de uma aplicação usando PP?
  - Utilizar mais unidades ativas?
    - Sim, mas existem problemas!
      - Dependência de dados.
      - Distribuição de dados.
      - Sincronização.
      - Áreas críticas.

Processamento Paralelo

- ✓ **Exemplo:** construção de um muro.
  - **Um pedreiro** faz um muro em **3 horas**.
  - **Dois pedreiros** fazem um muro em **2 horas**.
  - **Três pedreiros** fazem um muro em **1,5 horas**.
  - **Quatro pedreiros** fazem um muro em **2 horas**.
- ✓ Qual a conclusão deste exemplo?
  - A quantidade de trabalho a ser feita limita o número de unidades ativas que podem ser usadas de forma eficiente.

Processamento Paralelo

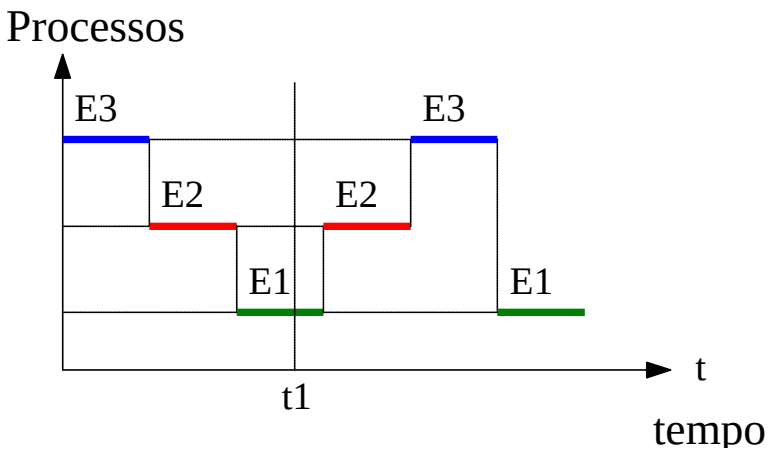
- ✓ Problemas do exemplo:
  - O muro só pode ser feito de baixo para cima (**dependência de dados**);
  - Os tijolos devem ser distribuídos entre os pedreiros (**distribuição de dados**);
  - Um pedreiro não pode levantar o muro do seu lado muito na frente dos outros pedreiros (**sincronização**);
  - Se existir um único carrinho, este será disputado pelos pedreiros – Fila (**áreas críticas**).

Processamento Paralelo

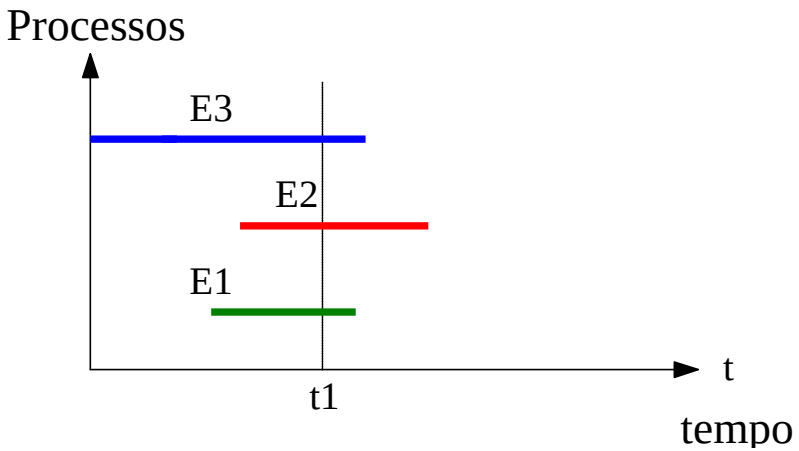
- ✓ Toda aplicação tem um número ideal de unidades ativas para obtenção do melhor desempenho.
- ✓ O ideal é que cada unidade ativa esteja 100% do tempo executando uma tarefa da aplicação.

Conceitos Básicos

- ✓ Paralelismo.



Paralelismo virtual



Paralelismo físico

Conceitos Básicos

- ✓ **Granulação (Nível de paralelismo).**
  - Indica o tamanho das unidades de trabalho submetidas aos processadores.
- ✓ Existem três níveis:
  - **Fina** → Unidades de trabalho pequenas;
  - **Média** → Unidades de trabalho médias;
  - **Grossa** → Unidades de trabalho grandes.

Conceitos Básicos

- ✓ ***Speedup*** → Indica o aumento de desempenho.

$$SpeedUp = \frac{T_1}{T_p}$$

T_1 é o tempo de execução em um processador.

T_p é o tempo de execução em p processadores.

Conceitos Básicos

- ✓ **Eficiência.**

$$Eficiência = \frac{SpeedUp}{p}$$

  - No caso ideal (*SpeedUp = p*), a eficiência seria máxima (Eficiência = 1).

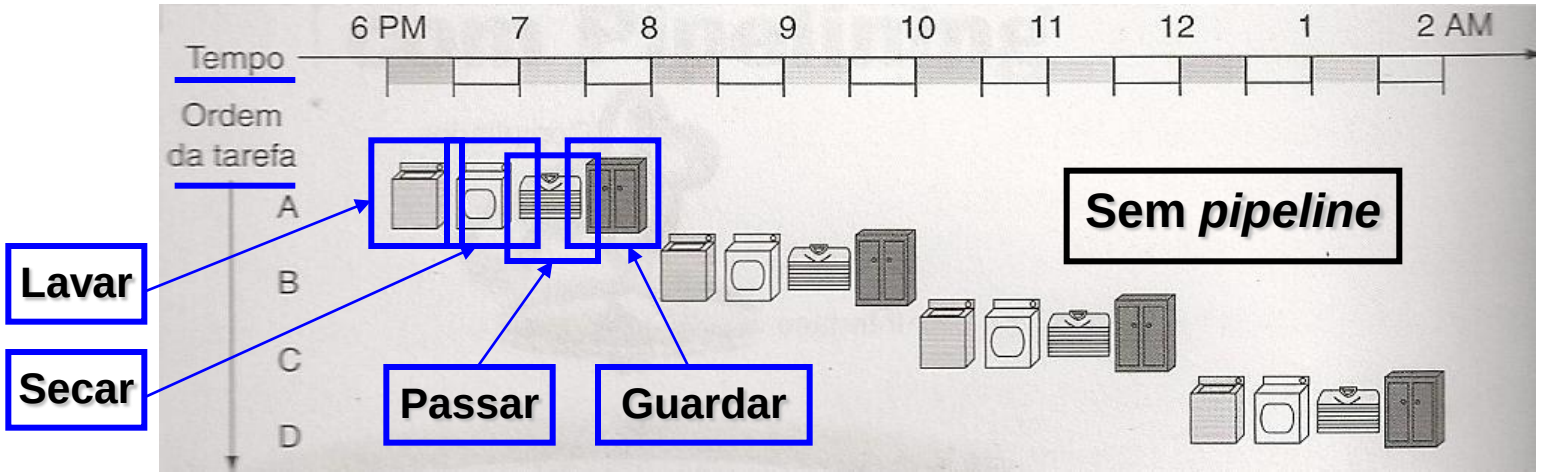
Pipeline

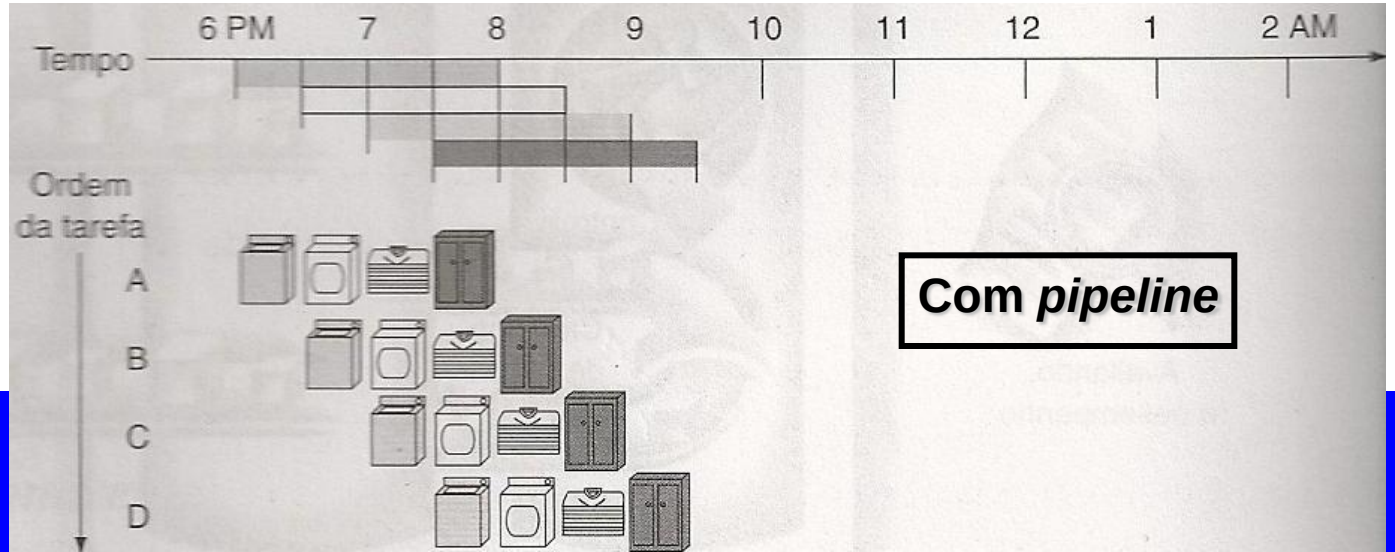
- ✓ **Definição** → Técnica utilizada em processadores que permite executar os diferentes estágios de várias instruções ao mesmo tempo.

- ✓ **Objetivo** → Melhorar o desempenho dos processadores – Torná-los mais rápidos.

Pipeline

- ✓ **Exemplo:** lavar roupas.





Pipeline

- ✓ Cada etapa de execução → **Estágio em *pipelining*.**
  - Quantos estágios no exemplo lavar roupas?
  - O tempo de execução de uma tarefa completa diminui com relação ao caso sem *pipeline*?

"Se todos os estágios tiverem o mesmo tempo de execução e **houver trabalho suficiente**, o ganho usando *pipelining* é igual ao número de estágios".

Pipeline

- ✓ Qual o ganho no nosso exemplo?

$$Ganho = \frac{T_{S/Pipeline}}{T_{C/Pipeline}} = \frac{8,0hs}{3,5hs} = 2,28$$

  - Usamos apenas quatro trouxas de roupas.

$Ganho \rightarrow N\acute{u}mero\ de\ est\acute{a}gios$
$Se\ N\acute{u}mero\ de\ tarefas \square\ N\acute{u}mero\ de\ est\acute{a}gios$

- ✓ **Tarefa:** determinem o ganho para 20 tarefas.

Pipeline

- ✓ Os princípios do exemplo anterior se aplicam na execução de instruções em processadores.
- ✓ **Exemplo:** Arquitetura MIPS (***M**icroprocessor without interlocked **p**ipeline **s**tages*).
- ✓ As instruções MIPS exigem cinco etapas:

1. **Buscar instrução da memória;**
2. **Ler registradores enquanto a instrução é decodificada.**
3. **Executar a operação ou calcular um endereço.**
4. **Acessar um operando na memória de dados.**
5. **Escrever o resultado em um registrador.**

Pipeline

- ✓ **Tarefa:** pesquisem na internet sobre a arquitetura MIPS e estudem o arquivo MIPS_instruções.pdf.
- ✓ **Simulador de *pipeline* MIPS – WebSimple-MIPS.**
  - http://201.17.130.17/matheus/simple/?page=manual

Desempenho: ciclo único x *pipeline*

- ✓ Comparar o tempo médio entre as instruções MIPS de uma **implementação em ciclo único e com *pipeline***.

Desempenho: ciclo único x *pipeline*

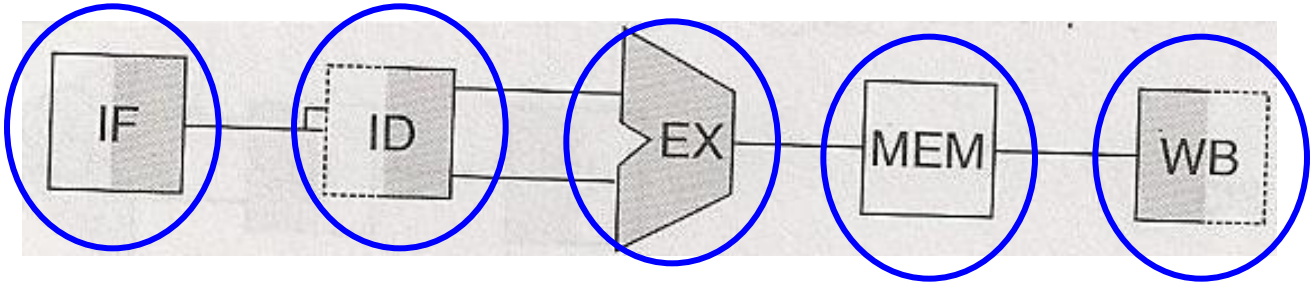
- ✓ **Modelo de ciclo único.**
  - Instruções MIPS exigem cinco estágios.
  - Todas as instruções levam **1 ciclo de *clock***.
    - Ciclo de *clock* = Tempo entre instruções = cte.
  - O **ciclo de *clock*** deve ser igual ao **tempo total** da instrução mais lenta.

Desempenho: ciclo único x *pipeline*

- ✓ **Modelo com *pipeline*.**
  - Semelhante ao modelo de ciclo único.
  - O **ciclo de *clock*** deve ser igual ao **tempo de duração do estágio mais lento**.

Desempenho: ciclo único x *pipeline*

- ✓ Os cinco estágios das instruções MIPS:



Representação dos estágios de uma instrução MIPS.

1. Buscar instrução da memória;
2. Ler registradores enquanto a instrução é decodificada.
3. Executar a operação ou calcular um endereço.
4. Acessar um operando na memória de dados.
5. Escrever o resultado em um registrador.

Desempenho: ciclo único x *pipeline*

Classe de instrução	Busca de instruções	Leitura de registradores	Operação da ALU	Acesso a dados	Escrita de registradores	Tempo total
Load word(lw)	200ps	100ps	200ps	200ps	100ps	800ps
Store word (sw)	200ps	100ps	200ps	200ps		700ps
Formato R (add, sub, and, or, slt)	200ps	100ps	200ps		100ps	600ps
Branch (beq)	200ps	100ps	200ps			500ps

Tempo gasto em cada estágio e tempo total de algumas instruções MIPS.

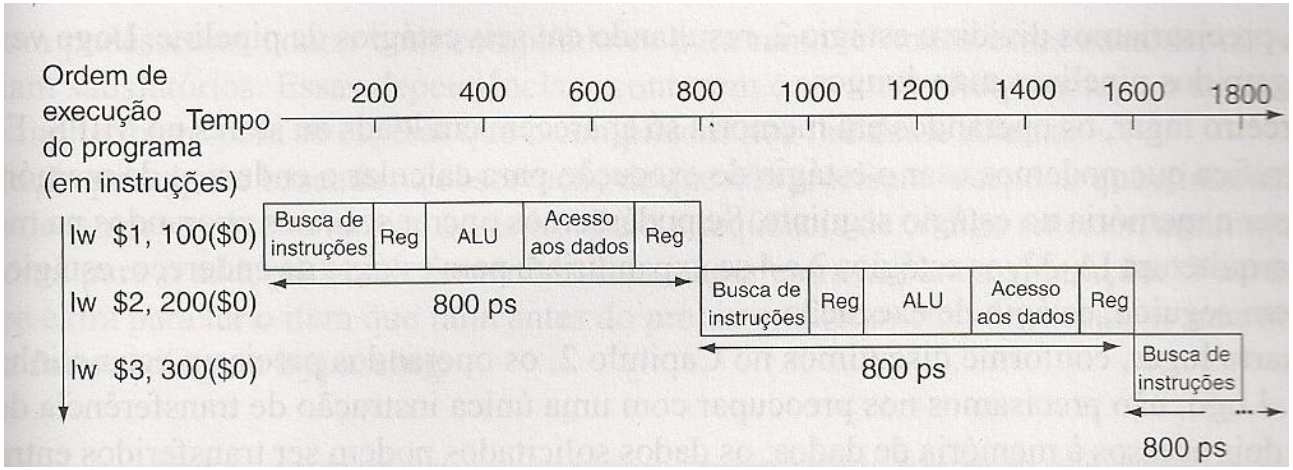
✓ Qual o **tamanho do ciclo de *clock*** para um projeto de ciclo único e para um projeto com *pipeline*?

- Ciclo único → 800 ps (instrução lw).

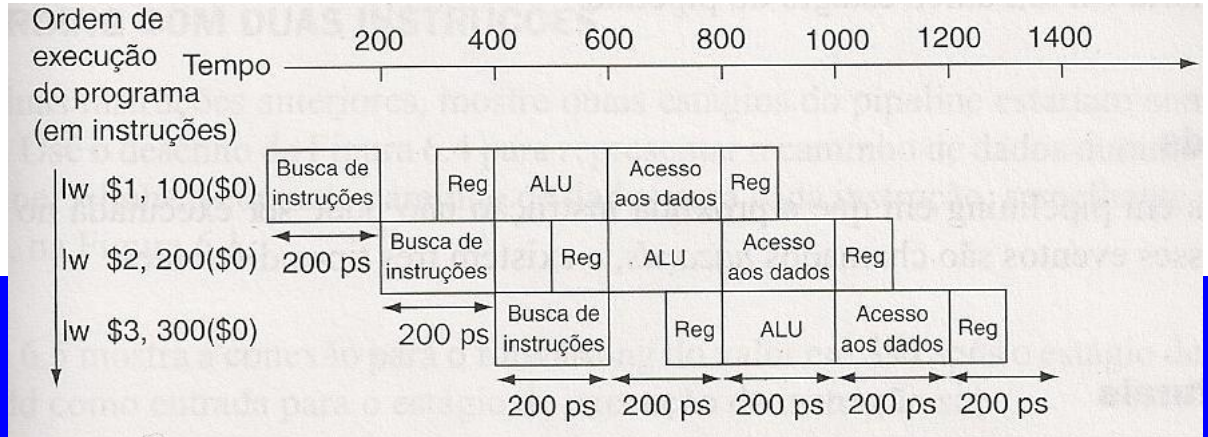
- *Pipeline* → 200 ps (estágio mais lento).

Desempenho: ciclo único x *pipeline*

- ✓ Execução do ciclo único – Três instruções lw.



- ✓ Execução com *pipeline* – Três instruções lw.



Desempenho: ciclo único x *pipeline*

$$Tempo\ entre\ instruções_{com\ pipeline} = \frac{Tempo\ entre\ instruções_{sem\ pipeline}}{Número\ de\ estágios}$$

- ✓ **Exemplo:** lavar roupas.

$$Tempo\ entre\ instruções_{com\ pipeline} = \frac{2,0}{4,0} = 0,5h$$

- ✓ **Exemplo:** três instruções lw.

$$Tempo\ entre\ instruções_{com\ pipeline} = \frac{800}{5} = \underline{160 \neq 200} \quad ???$$

Desempenho: ciclo único x *pipeline*

- ✓ No exemplo das instruções **lw**, os tempos dos estágios são desbalanceados, ao contrário do exemplo lavar roupas.
  - Tempos dos estágios iguais.

$$Ganho_{\max} = \frac{Tempo\ entre\ instruções_{sem\ pipeline}}{Tempo\ entre\ instruções_{com\ pipeline}}$$

$$Ganho_{exec\_total} = \frac{Tempo\ execução\ total_{sem\ pipeline}}{Tempo\ execução\ total_{com\ pipeline}}$$

$$Se\ Número\ de\ instruções = Número\ de\ estágios$$

$$Ganho_{exec\_total} \rightarrow Número\ de\ estágios$$

Desempenho: ciclo único x *pipeline*

- ✓ Tempos de estágios diferentes.

$$Ganho_{M\,ax} = \frac{Tempo\ entre\ instruções_{sem\ pipeline}}{Tempo\ entre\ instruções_{com\ pipeline}} < Número\ de\ estágios$$

$$Ganho_{Exec\_total} = \frac{Tempo\ execução\ total_{sem\ pipeline}}{Tempo\ execução\ total_{com\ pipeline}}$$

$$Se\ Número\ de\ instruções \neq Número\ de\ estágios$$

$$Ganho_{Exec\_total} \rightarrow Ganho_{Max} < Número\ de\ estágios$$

Exercício

- ✓ Para um projeto de ciclo único e para um projeto com *pipeline*, determine para a sequência de instruções a seguir.
  - O tamanho do ciclo de *clock*.
  - Desenhe os estágios de cada projeto.
  - O ganho máximo.
  - O ganho no tempo de execução total.

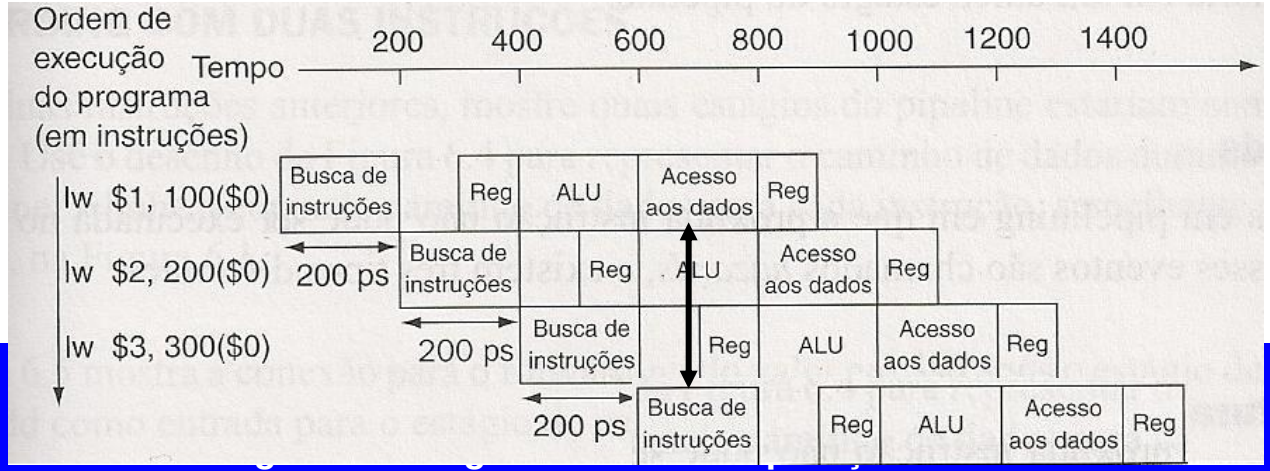
```
add $s0, $t0, $t1
sub $s1, $t2, $t3
sw  $s2, $s3
```

Pipeline Hazards (Riscos)

- ✓ **Ocorrência** → Quando a próxima instrução não pode ser executada no ciclo de *clock* seguinte.
- ✓ Existem três tipos de *pipeline hazards*:
  - Estruturais;
  - Dados;
  - Controle.

Pipeline Hazards

- ✓ ***Hazards* estruturais.**
  - O *hardware* não pode admitir a combinação de instruções que queremos executar no mesmo ciclo de *clock*.
  - **Exemplo** → Considere que temos uma única memória. Se dois estágios de instruções diferentes precisarem acessar à memória, um *hazard* estrutural ocorrerá.

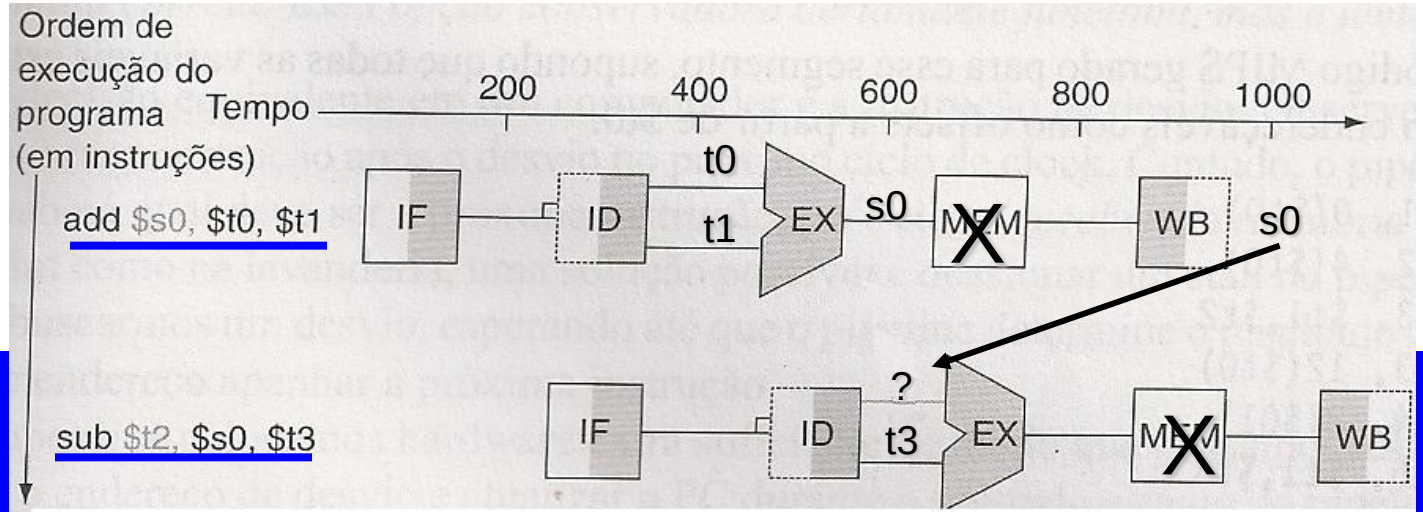


Pipeline Hazards

✓ ***Hazards* de dados.**

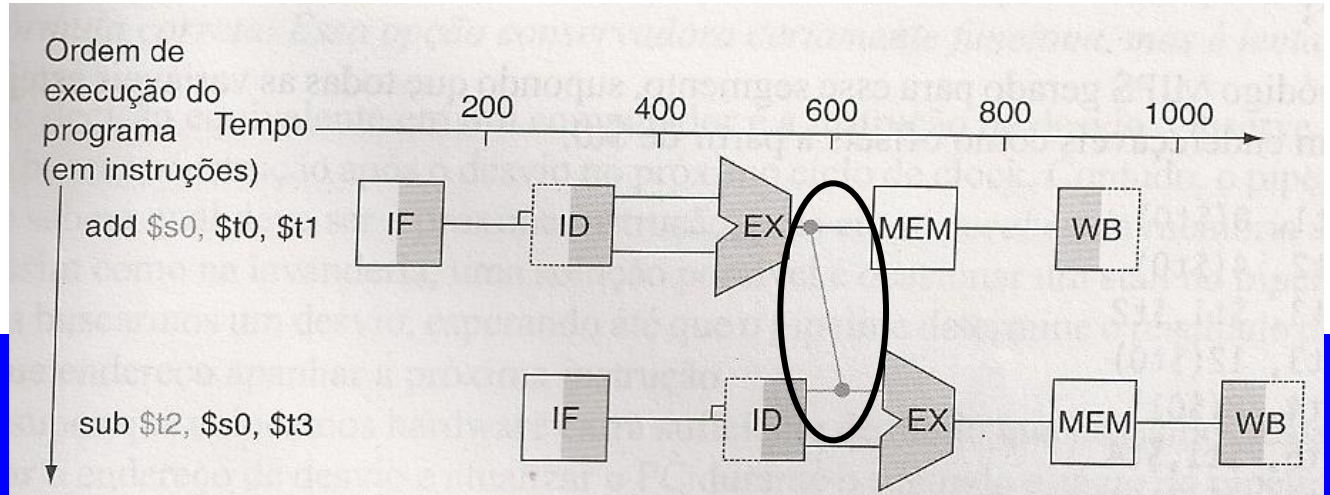
- Ocorrem quando o *pipeline* precisa ser interrompido porque um estágio precisa esperar até que outro seja concluído.

- **Exemplo** → Quando uma instrução depende de uma anterior que ainda está no *pipeline*.



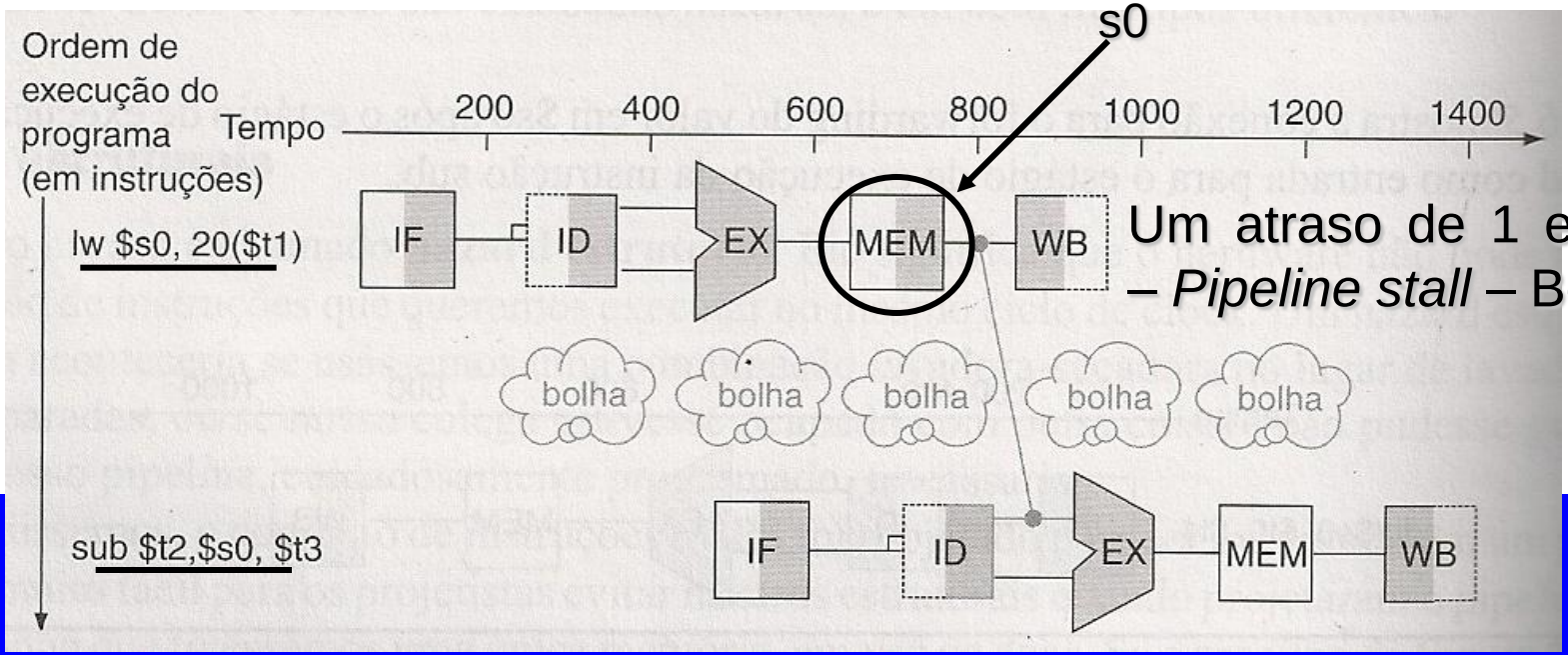
Pipeline Hazards

- ✓ ***Hazards* de dados.**
  - **Solução:**
    - Assim que a ULA gerar o resultado da soma, podemos fornecê-lo como uma entrada para a subtração.
    - Acrescentar *hardware* → ***Forwarding*** ou ***bypassing***.



Pipeline Hazards

- ✓ ***Hazards* de dados.**
  - *Forwarding* funciona bem, mas não resolve todas as situações.
  - **Exemplo** → Suponha no exemplo anterior uma instrução lw ao invés da instrução add.



Um atraso de 1 estágio – *Pipeline stall* – Bolha.

Pipeline Hazards

- ✓ ***Hazards* de controle.**

  - Utilizado para tomar decisões com base nos resultados de uma instrução enquanto outras estão sendo executadas.

  - **Exemplo** → Lavar roupas muito sujas.

    - A configuração da lavadora é suficiente para limpar a roupa?

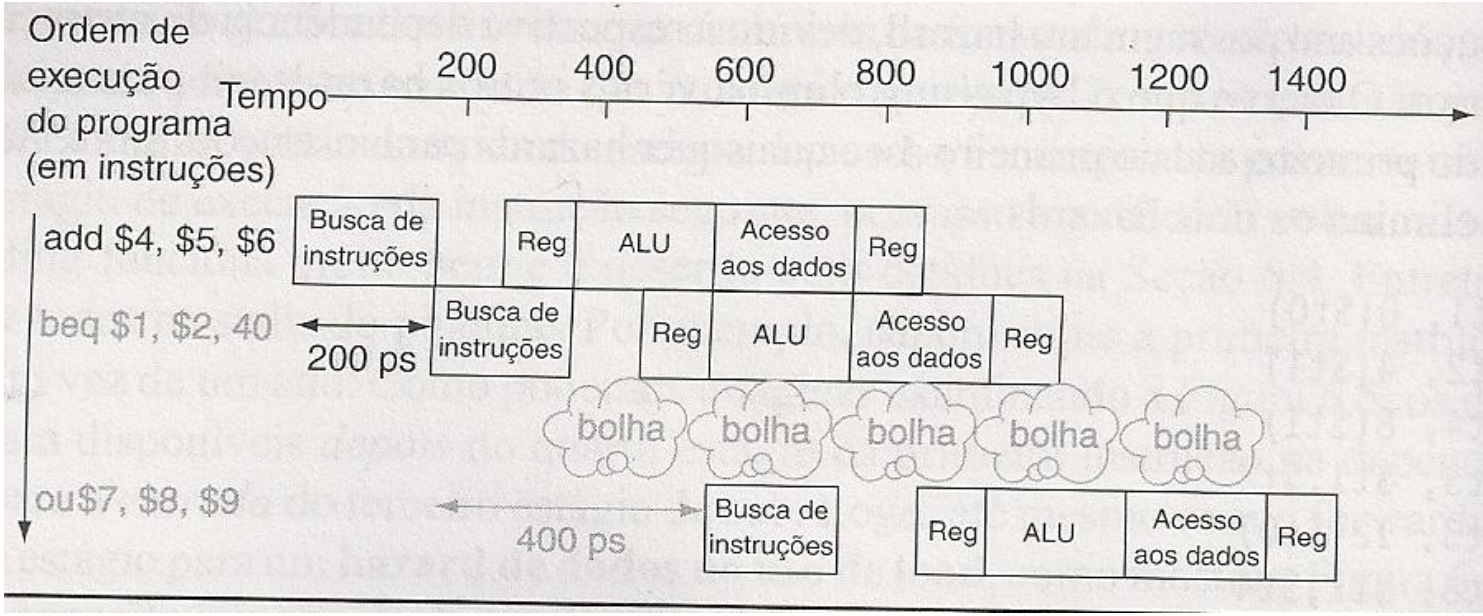
    - Esperar até o segundo estágio e verificar se é necessário mudar a configuração.

Pipeline Hazards

- ✓ ***Hazards* de controle.**
  - **Solução →** *Stall* (bolha) - Operar sequencialmente até que o primeiro lote esteja seco e depois repetir até obter a configuração correta.
    - Funciona, mas é uma solução lenta.
  - **Solução equivalente em um PC →** Instrução de desvio.
    - Após o desvio, é necessário buscar a instrução no ciclo de *clock* seguinte – Qual instrução?
    - **Solução →** Gerar uma bolha após buscar a instrução de desvio, até descobrir a próxima instrução.

Pipeline Hazards

- ✓ *Hazards* de controle.



Pipeline Hazards

- ✓ ***Hazards* de controle.**

  - **Outra solução** → Prever – Se estiver certo da configuração necessária para lavar as roupas, basta continuar lavando as roupas seguintes.
    - Sem desvios - Sem atraso.
    - Caso esteja errado, terá que refazer.

Pipeline Hazards

- ✓ ***Hazards* de controle.**
  - Os PCs utilizam técnicas de previsão.
  - Algumas técnicas:
    - Os desvios nunca ocorrerão → Estando certo, o *pipeline* seguirá. Caso contrário, sofre uma bolha.
    - Considerar alguns desvios previstos como tomados e outros como não tomados.
    - Manter histórico de desvios tomados e não tomados, e depois utilizá-los na previsão.