Noções de Desempenho

Anotações do material suplementar (apresentações PPT) ao Livro do Hennessy e Patterson e do material do Prof. Celso Alberto Saibel Santos (DI/CT).

Um pouco sobre Desempenho...

- Nesta aula veremos que o tempo de execução depende de 3 fatores chave:
  - Número de instruções
  - Tempo de ciclo de clock
  - Número de Clocks Por Instrução (CPI)
- Iremos focar e princípios e técnicas usados para implementar um processador (CPU) capaz de executar programas MIPS: por questões didáticas e de tempo, apenas um subconjunto das instruções será considerado no projeto.

Desempenho de Computadores

- Dois parâmetros tradicionais:

1. Tempo de Resposta (*Latency*)
   - — Quanto tempo leva minha tarefa para rodar?
   - — Quanto tempo leva a execução da minha tarefa?
   - — Quanto tempo devo esperar para uma consulta a uma base?
2. Vazão (*Throughput*)
   - — Quantas tarefas a máquina pode rodar por vez?
   - — Qual é a taxa de execução?
   - — Quanto trabalho é feito?

Tempos de Execução

- Tempo Total
  - Leva em conta “tudo” *(acesso a disco e memória, I/O , etc.)*
  - Um número útil, mas às vezes não tão bom para propósitos de comparação
- Tempo de CPU (CPU *time*)
  - Não conta tempo de I/O nem tempo gasto em outros programas
  - Pode ser dividido em tempo do sistema e tempo do usuário
- Tempo do Usuário:
  - O tempo gasto apenas na execução das instruções que estão no programa (código compilado)

Desempenho de CPU

- Para algum programa rodando na máquina X

  $$\text{Desempenho}_X = 1 \ / \ \text{Tempo-de-Execução}_X$$

Se "Máquina X é *N* vezes mais rápida que Maquina Y", então:

$$\text{Desempenho}_X/\text{Desempenho}_Y = \text{Tempo-de-Execução}_Y/\text{Tempo-de-Execução}_X = N$$

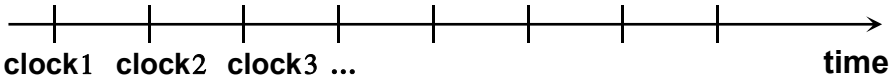
- **Problema** 1: Como comparar desempenho na mesma arquitetura?
- **Problema** 2: Como comparar desempenho em arquiteturas diferentes?

Ciclos de Clock

- Ao invés de observar o tempo de execução em segundos, *Hennessy e Patterson* preferem usar ciclos:

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}} = \frac{\text{Ninst}}{\text{program}} \times \frac{\text{cycle}}{\text{inst}} \times \frac{\text{seconds}}{\text{cycle}}$$

- "Pulsos" de Clock indicam quando iniciar/terminar atividades:



- Tempo de ciclo = Tempo entre pulsos = Duração em segundos/ciclo
- Taxa de *clock* (frequência) = ciclos/segundo (1 Hz = 1 ciclo/s)

Ex: Um computador com *clock* de 2 GHz tem um tempo de ciclo de $\frac{1}{2 \times 10^9} \times 10^9 = 0{,}5\,\eta\text{s}$

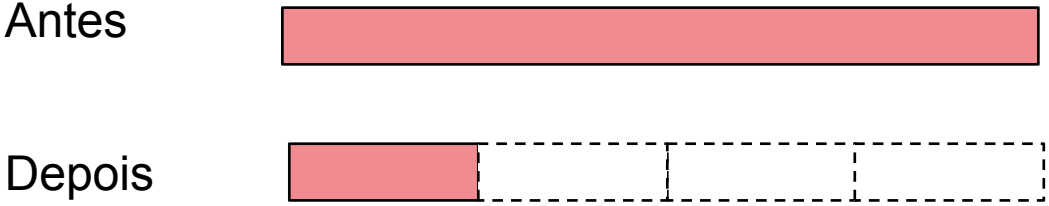
Como melhorar o desempenho ?

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- Assim, para melhorar o desempenho, mantendo-se "todo o resto" inalterado, pode-se pensar em reduzir:
  - O Número de ciclos de *clock* requeridos pelo programa; ou
  - Tempo do ciclo de *clock* (ou seja, aumentar a frequência de *clock*).

Speed-up

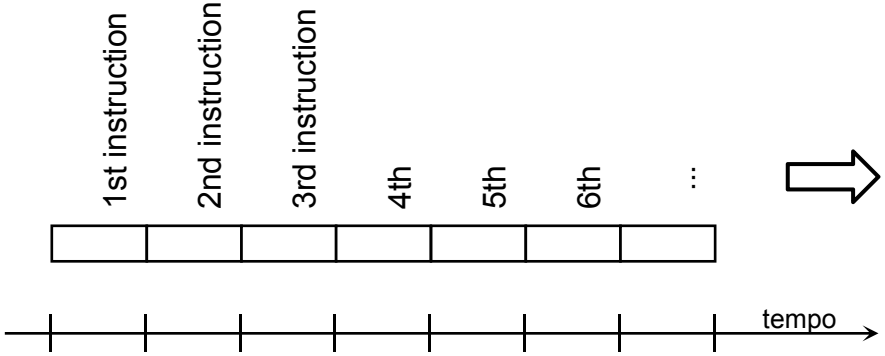
$$speedup = \frac{T_{exec\_antes}}{T_{exec\_depois}}$$ (**Ganho de velocidade** em % diretamente)



Speedup = 4
Novo programa roda 4x mais rápido que o antigo!

- Observe que o *speedup* pode ser calculado em função dos 3 fatores chave de desempenho: CPI, clock, número de instruções

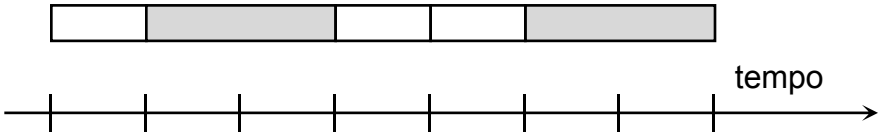
Quantos clocks preciso para executar um programa ?



No. de *ciclos* será sempre igual
No. de *instruções do programa ?*

- **A afirmação é INCORRETA!!!**
- Diferentes instruções de máquina gastam tempos diferentes em máquinas diferentes.

Número de ciclos para diferentes instruções



- Multiplicação gasta mais tempo que adição
- Operações de ponto flutuante são mais lentas que as de inteiros
- Acesso à memória gasta mais tempo que acesso a registradores

Importante: Mudanças no tempo de ciclo (taxa de *clock*) podem também alterar o número de ciclos exigidos para executar cada uma das classes de instruções anteriores...

Já que entendemos o que ciclos representam...

- Cada programa irá exigir, para sua execução, um certo número de Instruções (de máquina), Ciclos (clocks), Tempo (segundos)
- Este "certo número" está relacionado aos valores de:
  - Taxa de clock (ciclos/s) ou Tempo de ciclo (s/ciclo)
  - CPI (Ciclos Por Instrução – *cycles per instruction*)
- Máquina MIPS (Milhões de Instruções Por Segundo)

Desempenho

- Desempenho está intimamente ligado ao tempo de execução.
- Outras variáveis também podem ser usadas para medir desempenho:
  - No. de ciclos para executar um programa?
  - No. de instruções um programa?
  - No. de ciclos por segundo (frequência de *clock*)?
  - No. médio de ciclos por instrução (CPI)?
  - No. médio de instruções por segundo?

- **Erro comum**: pensar que APENAS UMA destas variáveis é indicativa de desempenho quando ela sozinha realmente NÃO É. Mais recentemente, outras preocupações apareceram (dissipação calor, consumo de energia, espaço/tamanho...)

Benchmarks

- Desempenho pode ser melhor estimado usando-se uma aplicação real
  - Usar programas com cargas (*workloads*) ou *classes de aplicações típicas* para a arquitetura (compiladores, editores, aplicações científicas/gráficas, jogos, navegadores, etc.)
- *Benchmarks* oferecem uma forma simples para padronizar e de uso livre para todos para comparar desempenhos
- SPEC (*System Performance Evaluation Cooperative*, de 1988)
  - 1ª geração SPEC CPU89 somente para CPUs
  - Fabricantes entraram num acordo para definir um conjunto de programas e entradas reais para avaliação a serem usados livremente por todos os interessados
  - Indicador valioso de desempenho (e tecnologia de compilação)

Benchmarks

- Hoje, várias *benchmarks*:

  https://www.spec.org/benchmarks.html

- Desde 1989, outros SPEC surgiram e sumiram!
  - SPECjvm98: Java; SPECweb99: servidores WWW; SPECmail2001: servidor de correio eletrônico. SPEC CPU2017

  https://www.spec.org/benchmarks.html

Como vimos…

- A execução de um programa obriga a CPU a buscar instruções a serem executadas na memória.
- Para isso, a CPU deve seguir os seguintes passos básicos do **ciclo de busca e execução**:
  1. Buscar instruções;
  2. Decodificar instruções;
  3. Buscar operandos/dados usados nas instruções;
  4. Executar instruções (processar dados/operandos);
  5. Escrever o resultado da execução.