# MO401

## Arquitetura de Computadores I

2006/2007

Prof. Paulo Cesar Centoducatte
ducatte@ic.unicamp.br
www.ic.unicamp.br/~ducatte



## MO401

- Livro Texto: "Computer Architecture: A Quantitative Approach" - 4rd edition
  Hennessy & Patterson
- Revisão: "Computer Organization and Design the hardware / software interface"
  Patterson & Hennessy
  - Revisão em sala: Pipelining, Desempenho, Hierarquia de Memórias (cache)



## Sumário

- Revisão
- Fundamentos
- Conjunto de Instruções
- Paralelismo no nível de instruções (ILP)
  - Exploração dinâmica (Superescalar)
- ILP por software (VLIW)
- Hierarquia de Memórias
- Multiprocessadores
- Sistemas de Armazenamento
- Redes e Clusters



## MO401

## Arquitetura de Computadores I

## Revisão:

## Pipeline, Desempenho e Hierarquia de Memórias (Caches)

"Computer Organization and Design the hardware / software interface"

"Computer Architecture: A Quantitative Approach" - (Apêndice A)



## Revisão: Pipeline



## Pipelining - Conceito

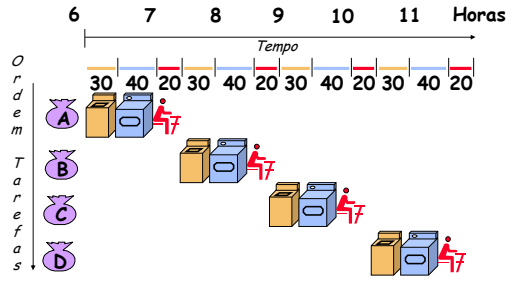
- Exemplo: Lavanderia
- 4 trouxas para serem lavadas


  - Lavar: 30 minutos

  - Secar: 40 minutos

  - Passar: 20 minutos

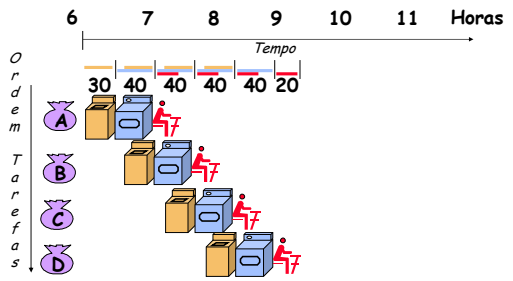
## Lavanderia Seqüencial




- Lavanderia seqüencial: 6 horas para lavar 4 trouxas
- Se for usado pipelining, quanto tempo?



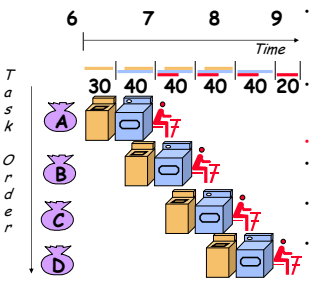
## Lavanderia Pipelined




- Lavanderia Pipelined: 3.5 horas para 4 trouxas



## Pipelining




- O Pipelining não ajuda na latência de uma tarefa, ajuda no throughput de toda a carga de trabalho
- O período do Pipeline é limitado pelo estágio mais lento
- Múltiplas tarefas simultâneas
- Speedup potencial = Número de estágios
- Estágios não balanceados reduzem o speedup
- O Tempo de "preenchimento" e de "esvaziamento" reduzem o speedup



## CPU Pipelines

- Executam bilhões de instruções: *throughput*
- Características desejáveis em um conjunto de instruções (ISA) para pipelining?
  - Instruções de tamanho variável vs. Todas instruções do mesmo tamanho?
  - Operandos em memória em qq operações vs. operandos em memória somente para loads e stores?
  - Formato das instruções irregular vs. formato regular das instruções (ié. Operandos nos mesmos lugares)?



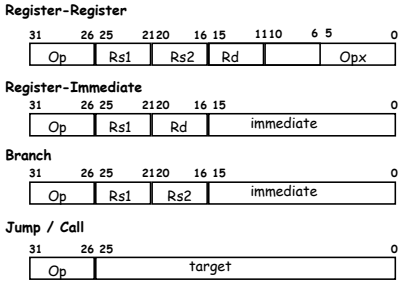
## Um RISC Típico

- Formato de instruções de 32-bit (3 formatos)
- Acesso à memória somente via instruções load/store
- 32 GPR de 32-bits (R0 contém zero)
- Instruções aritméticas: 3-address, reg-reg, registradores no mesmo lugar
- Modo de endereçamento simples para load/store (base + displacement)
  - Sem indireção
- Condições simples para o branch
- Delayed branch

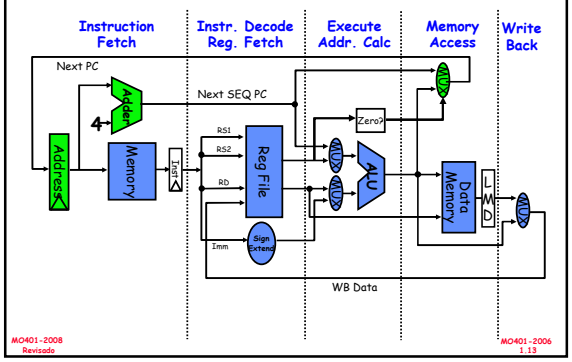
SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC, CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3



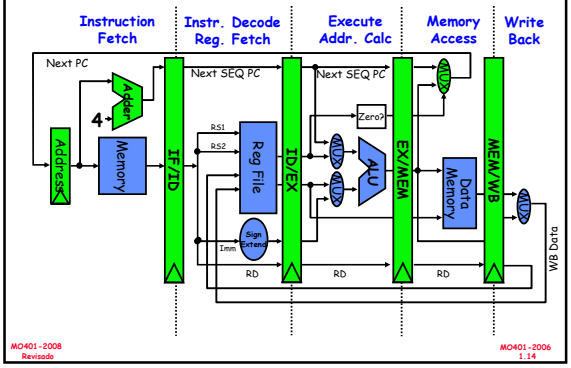
## Exemplo: MIPS (Localização dos regs)



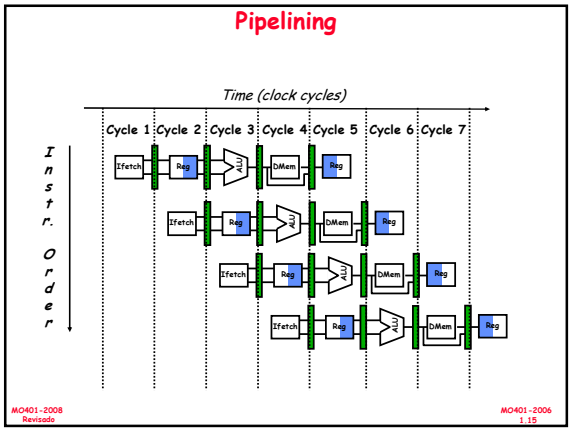
## MIPS Datapath - 5 estágios




## MIPS Datapath - 5 estágios




## Pipelining

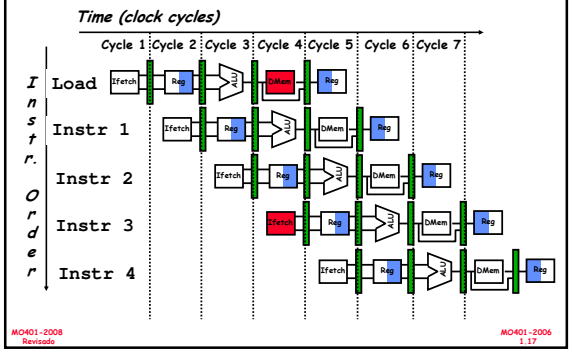



## Limites de Pipelining

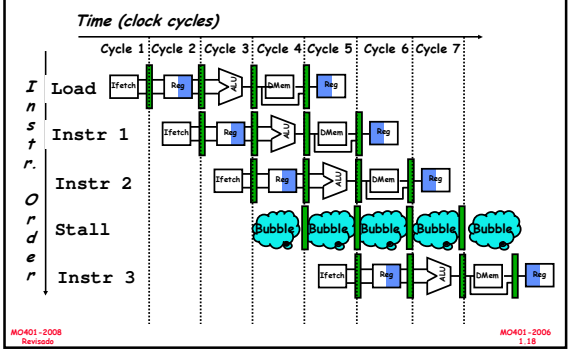
- **Hazards:** impedem que a próxima instrução seja executada no ciclo de clock "previsto" para ela
  - Structural hazards: O HW não suporta uma dada combinação de instruções
  - Data hazards: Uma Instrução depende do resultado da instrução anterior que ainda está no pipeline
  - Control hazards: Causado pelo delay entre o fetching de uma instrução e a decisão sobre a mudança do fluxo de execução (branches e jumps).



## Memória Única (D/I) - Structural Hazards




## Memória Única (D/I) - Structural Hazards



## Data Hazards

- Read After Write (RAW)

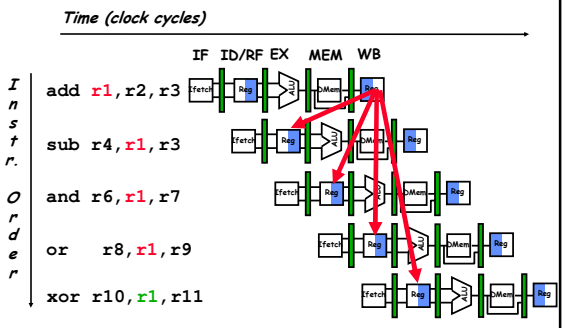
  $Instr_J$ lê o operando antes da $Instr_I$ escreve-lo

```
I: add r1,r2,r3
J: sub r4,r1,r3
```

  - Causada por uma "Dependência" (nomenclatura de compiladores).



## Data Hazard em R1






## Data Hazards

- Write After Read (WAR)
  $Instr_J$ escreve o operando ***antes*** que a $Instr_I$ o leia

```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Chamada "anti-dependência" (nomenclatura de compiladores). Devido ao reuso do nome "r1".
- Não ocorre no pipeline do MIPS:
  - Todas instruções usam 5 estágios, e
  - Leituras são no estágio 2, e
  - Escritas são no estágio 5



## Data Hazards

- Write After Write (WAW)
  $Instr_J$ escreve o operando ***antes*** que a $Instr_I$ o escreva.

```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Chamada "dependência de saída" (nomenclatura de compiladores). Devido ao reuso do nome "r1".
- Não ocorre no pipeline do MIPS:
  - Todas Instruções são de 5 estágios, e
  - Escritas são sempre no 5 estágio
  - (WAR e WAW ocorrem em pipelines mais sofisticados)



## Data Hazards – Solução por SW

- Compilador reconhece o data hazard e troca a ordem das instruções (quando possível)
- Compilador reconhece o data hazard e adiciona nops

Exemplo:

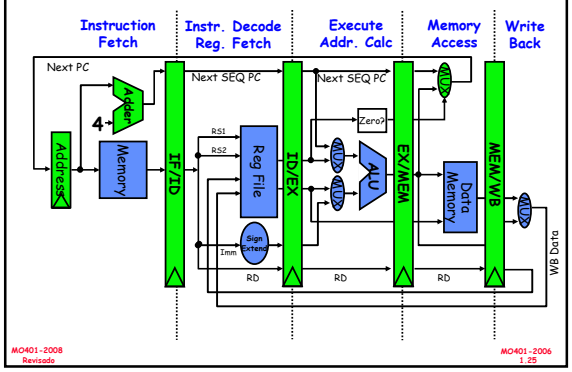
```
sub  R2, R1, R3    ; reg R2 escrito por sub
nop                ; no operation
nop
nop
and  R12, R2, R5   ; resultado do sub disponível
or   R13, R6, R2
add  R14, R2, R2
sw   100 (R2), R15
```



## Data Hazard Control: Stalls

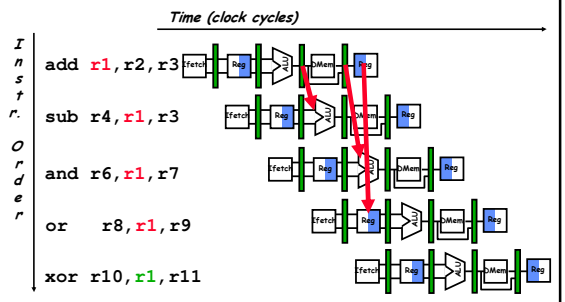
- Hazard ocorre quando a instr. Lê (no estágio ID) um reg que será escrito, por uma instr. anterior (no estágio EX, MEM, WB)
- Solução: Detectar o hazard e parar a instrução no pipeline até o hazard ser resolvido
- Detectar o hazard pela comparação do campo read no IF/ID pipeline register com o campo write dos outros pipeline registers (ID/EX, EX/MEM, MEM/WB)
- Adicionar bubble no pipeline
  - Preservar o PC e o IF/ID pipeline register

## MIPS Datapath - 5 estágios



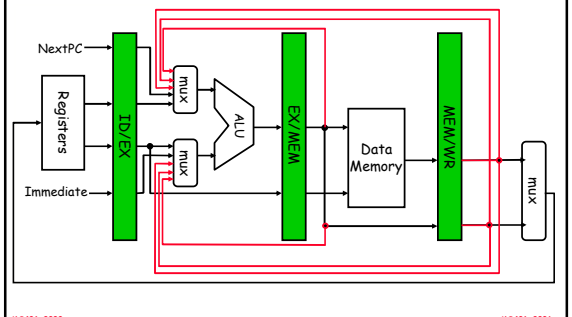

MO401-2008 Revisado — MO401-2006 1.25

## Redução do Data Hazard - Forwarding



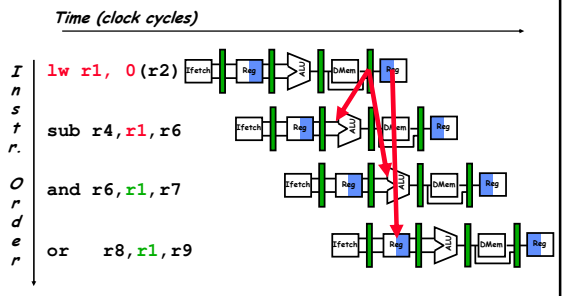

MO401-2008 Revisado — MO401-2006 1.26

## HW para Forwarding



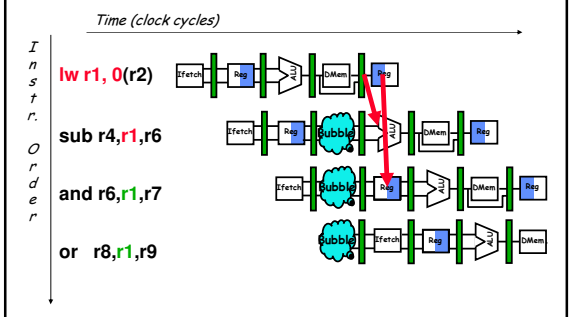

MO401-2008 Revisado — MO401-2006 1.27

## Data Hazard com Forwarding




MO401-2008 Revisado — MO401-2006 1.28

## Data Hazard com Forwarding




MO401-2008 Revisado — MO401-2006 1.29

## Load Hazards - Software Scheduling

**Código para** (a, b, c, d ,e, f na memória).

a = b + c;

d = e – f;

**Slow code:**

| | |
|---|---|
| LW | Rb,b |
| LW | Rc,c |
| ADD | Ra,Rb,Rc |
| SW | a,Ra |
| LW | Re,e |
| LW | Rf,f |
| SUB | Rd,Re,Rf |
| SW | d,Rd |

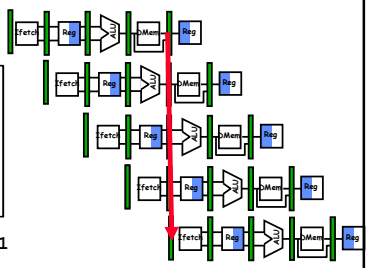
**Fast code:**

| | |
|---|---|
| LW | Rb,b |
| LW | Rc,c |
| LW | Re,e |
| ADD | Ra,Rb,Rc |
| LW | Rf,f |
| SW | a,Ra |
| SUB | Rd,Re,Rf |
| SW | d,Rd |

MO401-2008 Revisado — MO401-2006 1.30

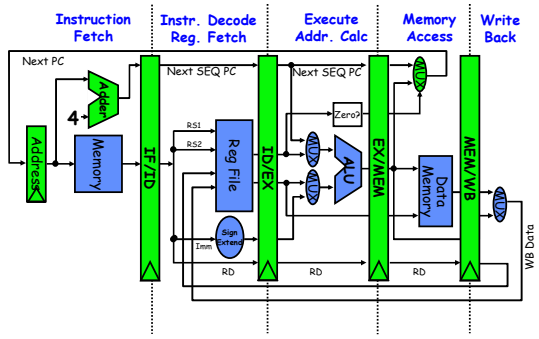
## Control Hazard - Branches (3 estágios de Stall)

```
10: beq r1,r3,36
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```






## MIPS Datapath - 5 estágios






## Exemplo: Impacto do Branch Stall

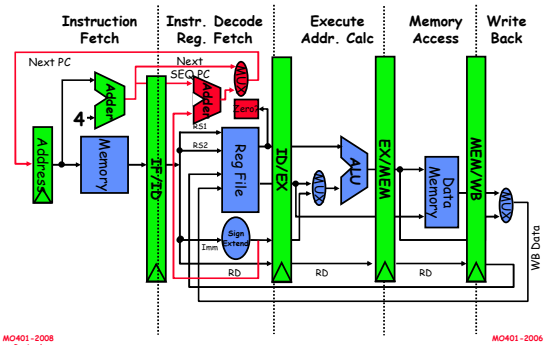
- Se CPI = 1, 30% branches, 3-cycle stall

  ⇒ CPI = 1.9!

- Solução para minimizar os efeitos:
  - Determinar branch taken ou não o mais cedo, e
  - Calcular o endereço alvo do branch logo
- MIPS branch: testa se reg = 0 ou ≠ 0
- Solução MIPS:
  - Zero test no estágio ID/RF
  - Adder para calcular o novo PC no estágio ID/RF
  - 1 clock cycle penalty por branch versus 3



## Pipelined MIPS Datapath






## Alternativas para Branch Hazard

**#1: Stall até a decisão se o branch será tomado ou não**

**#2: Predict Branch Not Taken**
- Executar a próxima instrução
- "Invalidar" as instruções no pipeline se branch é tamado
- Vantagem: retarda a atualização do pipeline
- 47% dos branches no MIPS não são tomados, em média
- PC+4 já está computado, use-o para pegar a próxima instrução

**#3: Predict Branch Taken**
- 53% dos branches do MIPS são tomados, em média
- "branch target address" no MIPS ainda não foi calculado
  - » 1 cycle branch penalty
  - » Em outras máquinas esse penalty pode não ocorrer



## Alternativas para Branch Hazard

**#4: Delayed Branch**
- Define-se que o branch será tamado APÓS a uma dada quantidade de instruções

```
branch instruction
  sequential successor1
  sequential successor2
  ........
  sequential successorn
branch target if taken
```

Branch delay de tamanho *n* (*n slots delay*)

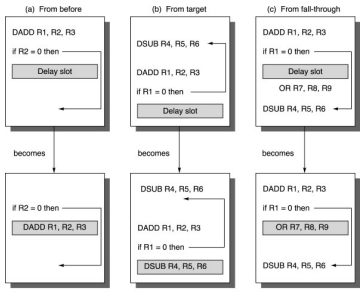
- 1 slot delay permite a decisão e o calculo do "branch target address" no pipeline de 5 estágios
- MIPS usa está solução

## Delayed Branch

- Qual instrução usar para preencher o branch delay slot?
  - Antes do branch
  - Do target address ( avaliada somente se branch taken)
  - Após ao branch (somente avaliada se branch not taken)



## Delayed Branch







## Delayed Branch

- Compilador: single branch delay slot:
  - Preenche +/- 60% dos branch delay slots
  - +/- 80% das instruções executadas no branch delay slots são utéis à computação
  - +/- 50% (60% x 80%) dos slots preenchidos são utéis



## Revisão: Desempenho



## Qual o mais rápido?

| Plane | DC to Paris | Speed | Passengers | Throughput (pmph) |
|---|---|---|---|---|
| Boeing 747 | 6.5 hours | 610 mph | 470 | 286,700 |
| BAD/Sud Concorde | 3 hours | 1350 mph | 132 | 178,200 |

- Time to run the task (ExTime)
  - Execution time, response time, latency
- Tasks per day, hour, week, sec, ns … (Desempenho)
  - Throughput, bandwidth



## Definições

- Desempenho (performance) é em unidades por segundo
  - Quanto maior melhor

$$\text{performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

" X é n vezes mais rápido que Y" significa que:

$$n = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = \frac{\text{Execution\_time}(Y)}{\text{Execution\_time}(X)}$$

## Aspectos sobre Desempenho de CPU (CPU Law)

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

| | Inst Count | CPI | Clock Rate |
|---|---|---|---|
| Program | X | | |
| Compiler | X | (X) | |
| Inst. Set. | X | X | |
| Organization | | X | X |
| Technology | | | X |



## Instruções por Ciclo (Throughput)

"Cycles per Instruction" médio

CPI = (CPU Time * Clock Rate) / Instruction Count
= Cycles / Instruction Count

$$\text{CPU time} = \text{Cycle Time} \times \sum_{j=1}^{n} \text{CPI}_j \times \text{I}_j$$

"Freqüência das Instruções"

$$\text{CPI} = \sum_{j=1}^{n} \text{CPI}_j \times \text{F}_j \qquad \text{where } \text{F}_j = \frac{\text{I}_j}{\text{Instruction Count}}$$



## Exemplo: Calculando CPI

Base Machine (Reg / Reg)

| Op | Freq | Cycles | CPI(i) | (% Time) |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | (33%) |
| Load | 20% | 2 | .4 | (27%) |
| Store | 10% | 2 | .2 | (13%) |
| Branch | 20% | 2 | .4 | (27%) |
| | | | 1.5 | |

Mix típico de instruções em programas



## Exemplo: Impacto de Branch Stall

- Assuma: CPI = 1.0 (ignorando branches stall por 3 ciclos)
- Se 30% são branch, Stall 3 ciclos

| Op | Freq | Cycles | CPI(i) | (% Time) |
|---|---|---|---|---|
| Other | 70% | 1 | .7 | (37%) |
| Branch | 30% | 4 | 1.2 | (63%) |

- => novo CPI = 1.9, ou aproximadamente 2 vezes mais lento



## Exemplo 2: SpeedUp para Pipelining

$$\text{CPI}_{\text{pipelined}} = \text{Ideal CPI} + \text{Average Stall cycles per Inst}$$

$$\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

For simple RISC pipeline, CPI = 1:

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$



## Exemplo 3: Branch Alternativas

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

| *Scheduling scheme* | *Branch penalty* | *CPI* | *speedup v. stall* |
|---|---|---|---|
| Stall pipeline | 3 | 1.42 | 1.0 |
| Predict taken | 1 | 1.14 | 1.26 |
| Predict not taken | 1 | 1.09 | 1.29 |
| Delayed branch | 0.5 | 1.07 | 1.31 |

## Exemplo 4: Dual-port vs. Single-port

- Máquina A: Dual ported memory ("Harvard Architecture")
- Máquina B: Single ported memory, porém seu pipelined é 1.05 vezes mais rápido (clock rate)
- CPI Ideal = 1 para ambas
- Loads: 40% das instruções executadas

$SpeedUp_A = Pipeline\ Depth/(1 + 0) \times (clock_{unpipe}/clock_{pipe})$
$= Pipeline\ Depth$
$SpeedUp_B = Pipeline\ Depth/(1 + 0.4 \times 1) \times (clock_{unpipe}/(clock_{unpipe} / 1.05)$
$= (Pipeline\ Depth/1.4) \times 1.05$
$= 0.75 \times Pipeline\ Depth$
$SpeedUp_A / SpeedUp_B = Pipeline\ Depth/(0.75 \times Pipeline\ Depth) = 1.33$

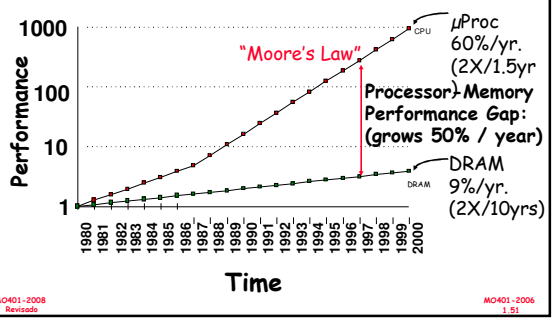
- Máquina A é 1.33 mais rápida que a B



## Revisão: Hierarquia de Memórias

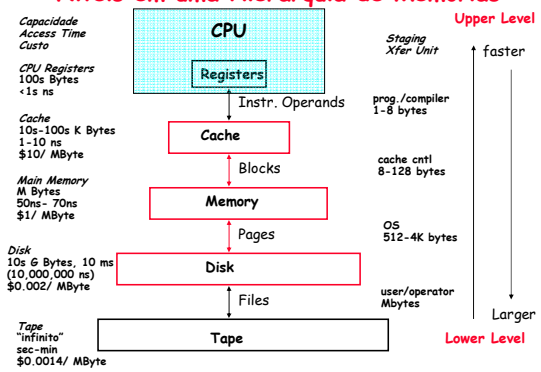


## Processor-DRAM Memory Gap (latency)






## Níveis em uma Hierarquia de Memórias





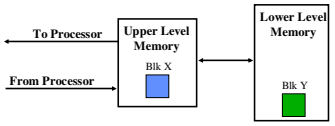

## Princípio da Localidadde

- Princípio da Localidade:
  - Programas acessam relativamente uma pequena porção do espaço de endereçamento em um dado instante de tempo.
- Dois tipos de Localidade:
  - Localidadde Temporal (Localidade no Tempo): Se um item é referenciado, ele tende a ser referenciado outra vez em um curto espaço de tempo (loops)
  - Localidade Espacial (Localidade no Espaço): Se um item é referenciado, itens próximos também tendem a serem referenciados em um curto espaço de tempo (acesso a array)

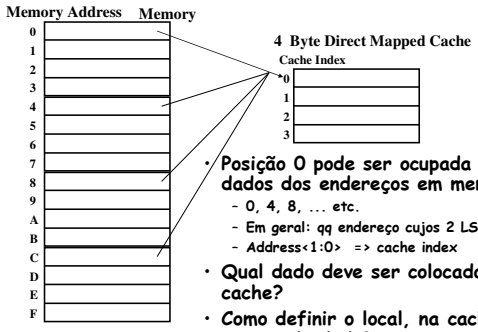


## Hierarquia de Memórias: Terminologia

- Hit: o dado está no upper level (examplo: Block X)
  - Hit Rate: taxa de hit no upper level no acesso à memória
  - Hit Time: Tempo para o acesso no upper level, consiste em: RAM access time + Time to determine hit/miss
- Miss: o dado precisa ser buscado em um bloco no lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Tempo para colocar um bloco no upper level + Tempo para disponibilizar o dado para o processador
- Hit Time << Miss Penalty (500 instruções no 21264!)



## Cache: Direct Mapped

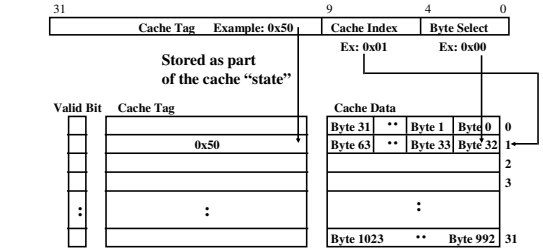



- Posição 0 pode ser ocupada por dados dos endereços em memória:
  - 0, 4, 8, ... etc.
  - Em geral: qq endereço cujos 2 LSBs são 0s
  - Address<1:0> => cache index
- Qual dado deve ser colocado na cache?
- Como definir o local, na cache, para cada dado?



## 1 KB Direct Mapped Cache, 32B blocks

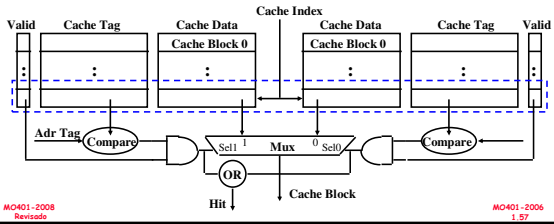
- Para uma cache de 2 ** N byte:
  - Os (32 - N) bits de mais alta ordem são "Cache Tag"
  - Os M bits de mais baixa ordem são "Byte Select" (Block Size = 2 ** M)






## Two-way Set Associative Cache

- N-way set associative: N entradas para cada Cache Index
  - N direct mapped caches opera em paralelo (N típico: 2 a 4)
- Examplo: Two-way set associative cache
  - Cache Index: seleciona um "set" na cache
  - As duas tags no set são comparadas em paralelo
  - O Dado é selecionado basedo no resultado da comparação das tag

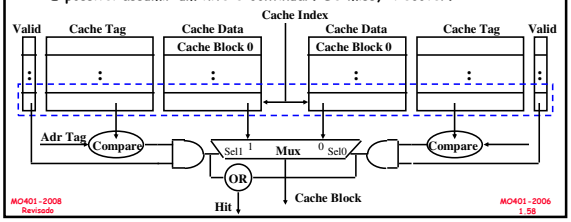





## Desvantagem de Set Associative Cache

- N-way Set Associative Cache v. Direct Mapped Cache:
  - N comparadores vs. 1
  - MUX extra, atrasa o acesso ao dado
  - Dado disponível APÓS Hit/Miss
- Direct mapped cache: Cache Block disponível ANTES do Hit/Miss:

  É possível assumir um hite e continuar. Se miss, Recover.





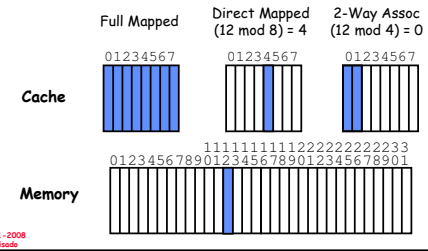

## Hierarquia de Memória: 4 Questões

- Q1: Em que lugar colocar um bloco no upper level?
  *(Block placement)*
- Q2: Como localizar o bloco se ele está no upper level?
  *(Block identification)*
- Q3: Qual bloco deve ser trocado em um miss?
  *(Block replacement)*
- Q4: O que ocorre em um write?
  *(Write strategy)*



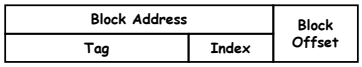
## Q1: Em que lugar colocar um bloco no upper level?

- Colocar o Bloco 12 em uma cache de 8 blocos :
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets



## Q2: Como localizar o bloco se ele está no upper level?

- Tag em cada bloco
  - Não é necessário testar o index ou block offset
- O aumento da associatividade reduz o index e aumenta a tag






## Q3: Qual bloco deve ser trocado em um miss?

- Fácil para Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)
  - FIFO

| Assoc: | 2-way | | 4-way | | 8-way | |
|---|---|---|---|---|---|---|
| Size | LRU | Ran | LRU | Ran | LRU | Ran |
| 16 KB | 5.2% | 5.7% | 4.7% | 5.3% | 4.4% | 5.0% |
| 64 KB | 1.9% | 2.0% | 1.5% | 1.7% | 1.4% | 1.5% |
| 256 KB | 1.15% | 1.17% | 1.13% | 1.13% | 1.12% | 1.12% |

Taxa de misses na cache: blocos de 16 bytes na arquitetura Vax; acessos de usuários e do sistema operacional

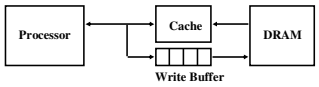


## Q4: O que ocorre em um write?

- *Write through* — A informação é escrita tanto no bloco da cache quanto no bloco do lower-level memory.
- *Write back* — A informação é escrita somente no bloco da cache. O bloco da cache modificado é escrito na memória principal somente quando ele é trocaddo.
  - block clean or dirty?
- Prós e Contras?
  - WT: read misses não pode resultar em writes
  - WB: não há repetição de writes na mesma posição
- WT, em geral, é combinado com write buffers, assim não há espera pelo lower level memory



## Write Buffer para Write Through

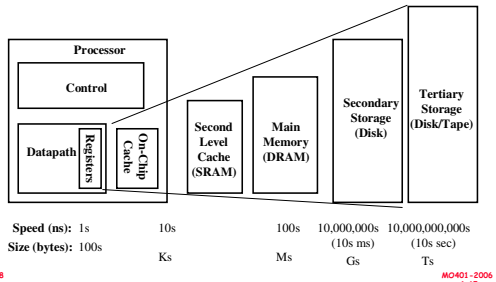



- Um Write Buffer colocado entre a Cache e a Memory
  - Processador: escreve o dado na cache e no write buffer
  - Memory controller: escreve o conteúdo do buffer na memória
- Write buffer é uma FIFO:
  - Número típico de entradas: 4
  - Trabalha bem se: freqüência de escrita (w.r.t. time) << 1 / DRAM write cycle
- Memory system é um pesadelo para o projetista :
  - freqüência de escrita (w.r.t. time) -> 1 / DRAM write cycle
  - Saturação do Write buffer



## Hierarquia de Memória Moderna

- Tirando vantagens do príncipio da localidade:
  - Provê ao usuário o máximo de memória disponibilizada pela tecnologia mais barata.
  - Provê acesso na velocidade oferecida pela tecnologia mais rápida.






## Resumo #1/3: Pipelining & Desempenho

- Sobreposição de tarefas; fácil se as tarefas são independentes
- Speed Up ≤ Pipeline Depth; Se CPI ideal for 1, então:

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

- Hazards limita o desempenho nos computadores:
  - Estrutural: é necessário mais recursos de HW
  - Dados (RAW,WAR,WAW): forwarding, compiler scheduling
  - Controle: delayed branch, prediction
- Tempo é a medida de desempenho: latência ou throughput
- CPI Law:

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

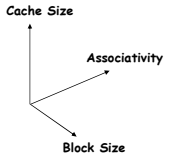
## Resumo #2/3: Caches

- Princípio da Localidade:
  - Programas acessam relativamente uma pequena porção do espaço de endereçamento em um dado instante de tempo.
    - » Localidadde Temporal :Localidade no Tempo
    - » Localidade Espacial : Localidade no Espaço
- Cache Misses: 3 categorias
  - Compulsory Misses
  - Capacity Misses
  - Conflict Misses
- Políticas de Escrita:
  - Write Through: (write buffer)
  - Write Back



## Resumo #3/3: Cache Design

- Várias Dimensões interagindo
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back




- Solução ótima é um compromiso
  - Depende da característica dos acessos
    - » workload
    - » I-cache, D-cache, TLB
  - Depende da razão tecnologia / custo