

MC722
Organização de Computadores
 Teoria e Prática

2012
 Prof. Paulo Cesar Centoducatte
ducatte@ic.unicamp.br
www.ic.unicamp.br/~ducatte

MC722
4.3

MC722
Arquitetura de Computadores
 Micro-Arquitetura Pipeline

"COD" - (Capítulo)





MC722
4.2

Micro-Arquitetura Pipeline
 Sumário

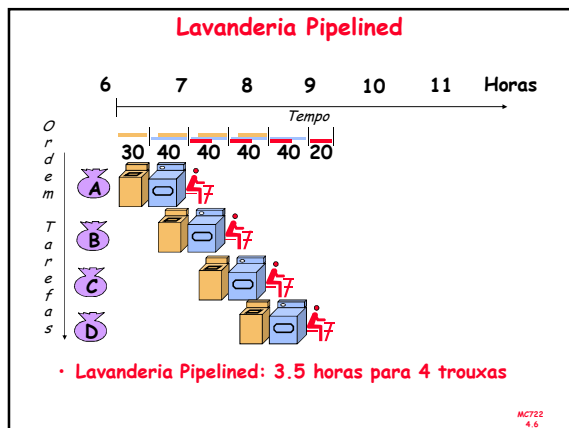
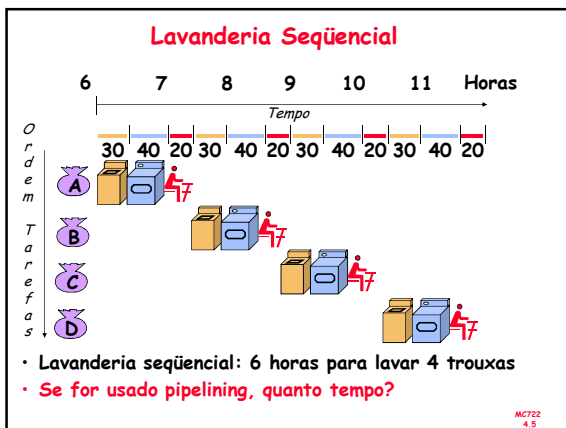
- **Introdução**
 - Execução Pipeline
- **MIPS Pipelining**
 - 5 estágios
 - MIPS Pipelined
 - Execução de Instruções no MIPS Pipeline
 - DataPath Single-Cycle e Pipelining
 - Controle do Pipeline
- **Limites de Pipelining**
 - Hazards
 - » Estrutura
 - » Dado
 - » Controle
- **Como Tratar os Hazards de Dados**

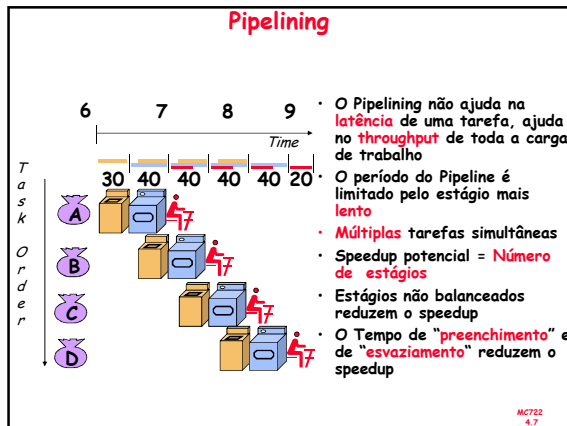
MC722
4.3

Pipelining - Conceito

- **Exemplo: Lavanderia**
- 4 trouxas para serem lavadas 
- Lavar: 30 minutos 
- Secar: 40 minutos 
- Passar: 20 minutos 

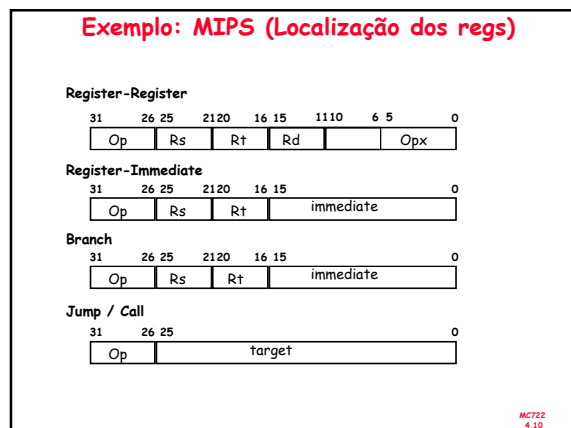
MC722
4.4



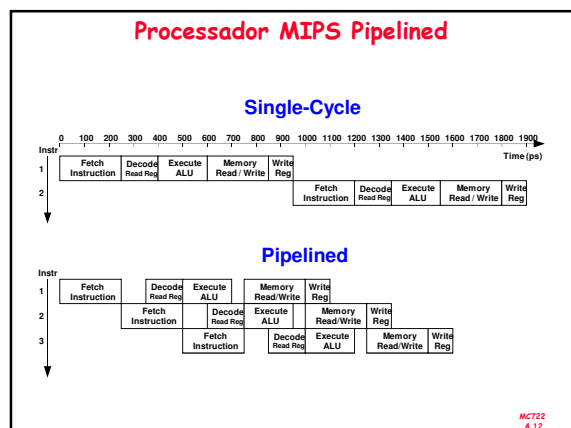


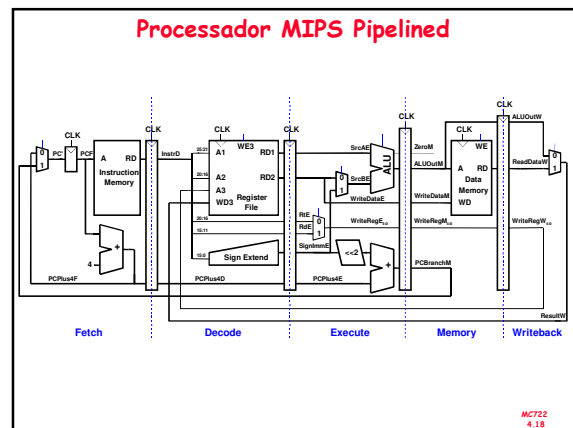
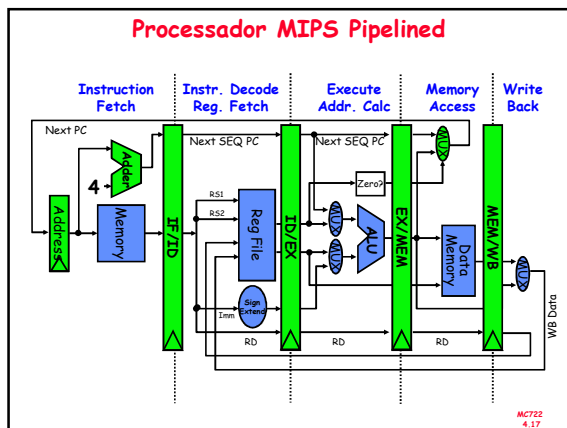
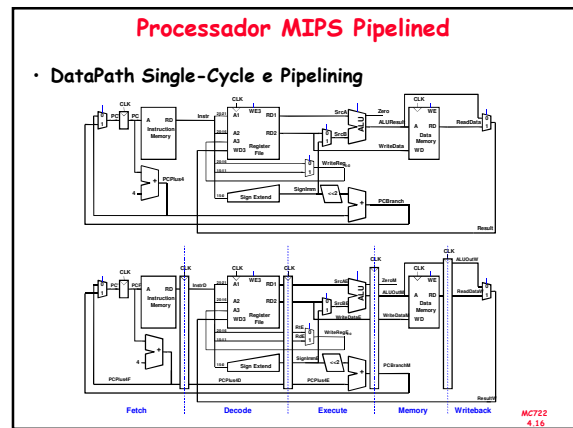
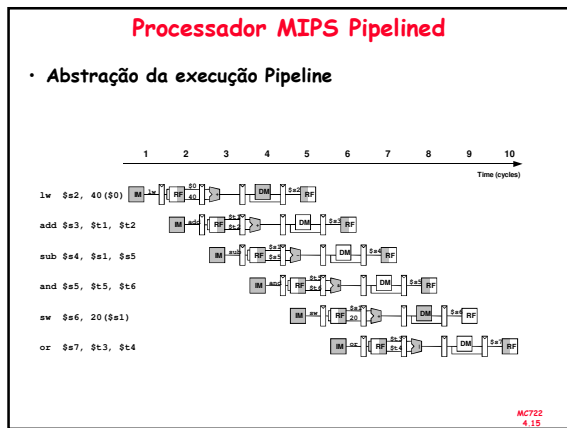
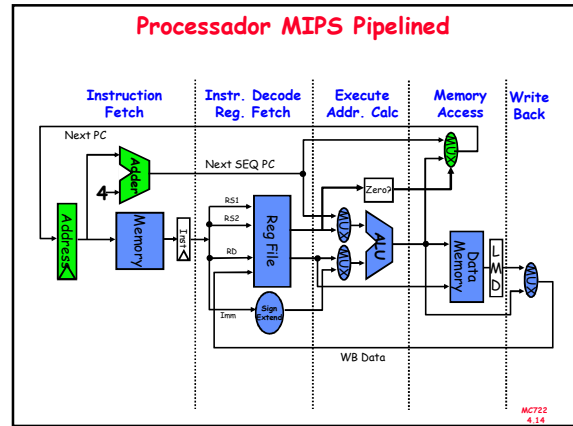
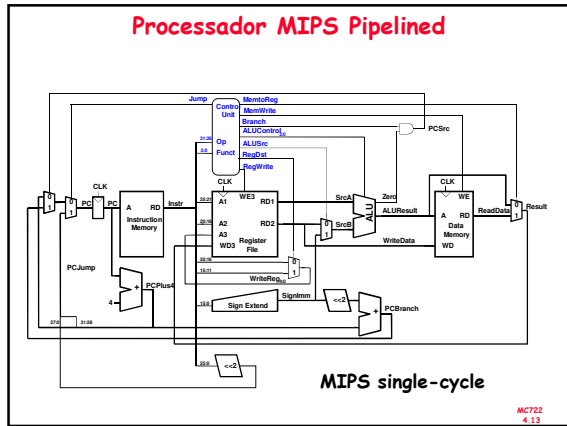
- ### CPU Pipeline
- Executam bilhões de instruções: **throughput**
 - Características desejáveis em um conjunto de instruções (ISA) para pipelining?
 - Instruções de tamanho variável vs. Todas instruções do mesmo tamanho?
 - Operandos em memória em qq operação vs. operandos em memória somente para **loads e stores**?
 - Formato das instruções irregular vs. formato regular das instruções (i.e. Operandos nos mesmos lugares)?
- MCT22 4.8

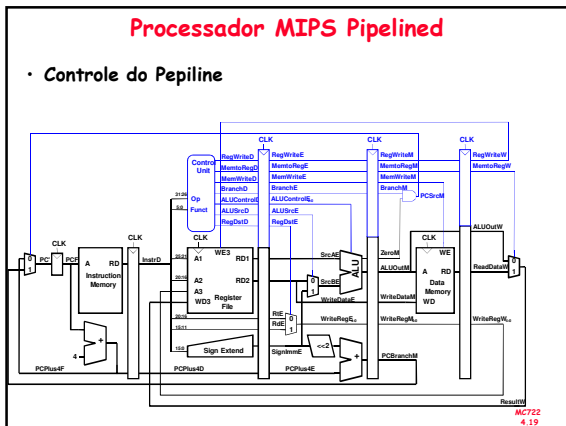
- ### Um RISC Típico
- Formato de instruções de 32-bit (3 formatos)
 - Acesso à memória somente via instruções **load/store**
 - 32 GPR de 32-bits (R0 contém zero)
 - Instruções aritméticas: 3-address, reg-reg, registradores no mesmo lugar
 - Modo de endereçamento simples para **load/store (base + displacement)**
 - 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
- MCT22 4.9



- ### Processador MIPS Pipelined
- Paralelismo Temporal
 - Divide o processador single-cycle em 5 estágios:
 - Fetch
 - Decode
 - Execute
 - Memory
 - Writeback
 - Adicionar registradores de pipeline entre os estágios
- MCT22 4.11



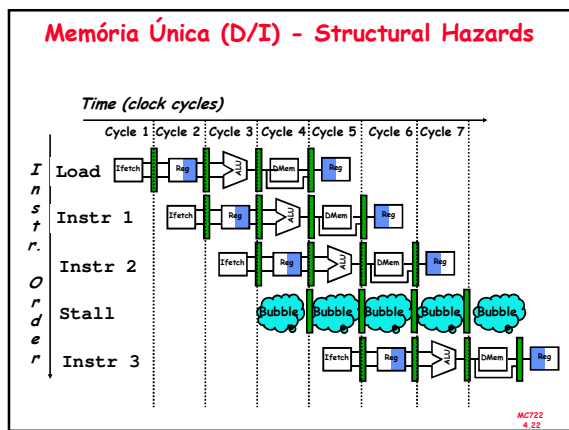
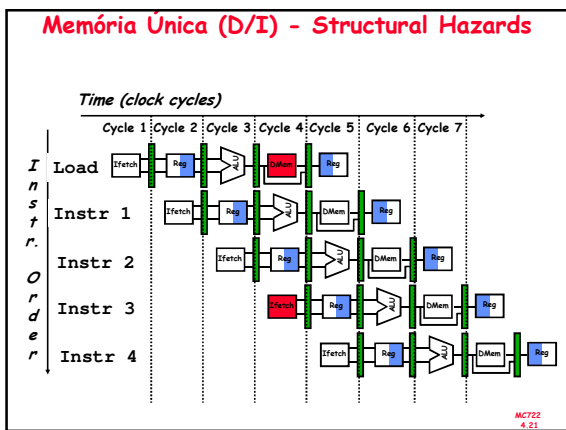




Limites de Pipelining

- **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**).

MCT22 4.20



Data Hazards

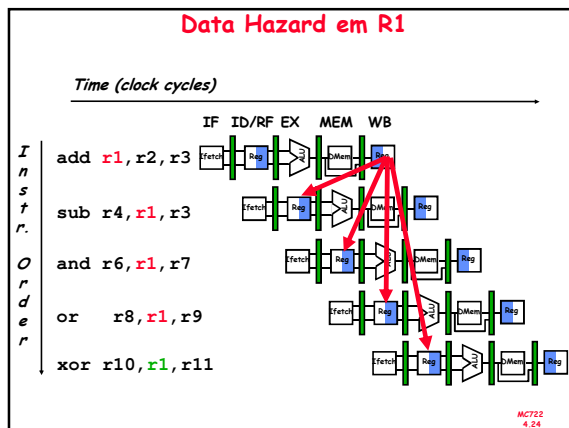
• **Read After Write (RAW)**

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).

MCT22 4.23



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

MCT22
4.25

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)

MCT22
4.25

Como Tratar os Data Hazards

- Inserir instruções nops no código
- Rearranjar o código em tempo de compilação
- Stall o processador em tempo de execução (run-time)
- Forward data

MCT22
4.27

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
and R12, R2, R5 ; resultado do sub disponível
or R13, R6, R2
add R14, R2, R2
sw 100 (R2), R15
    
```

MCT22
4.28

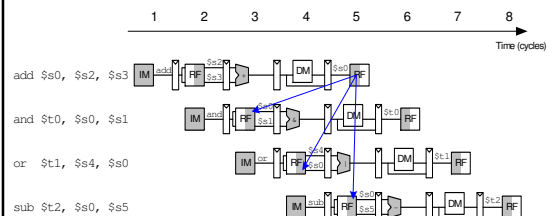
Data Hazard Control: Stalls

- 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

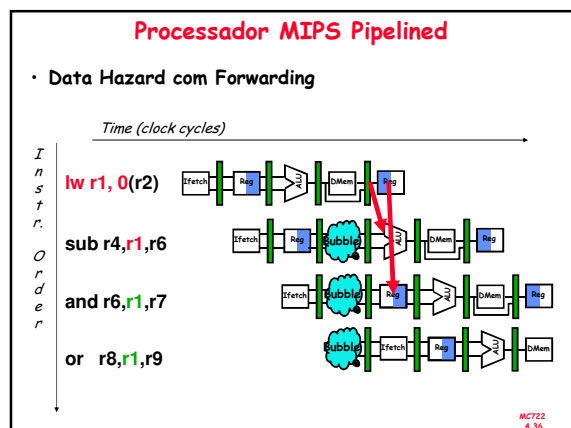
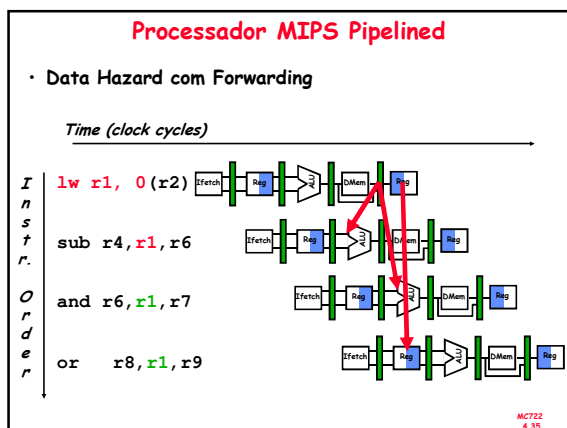
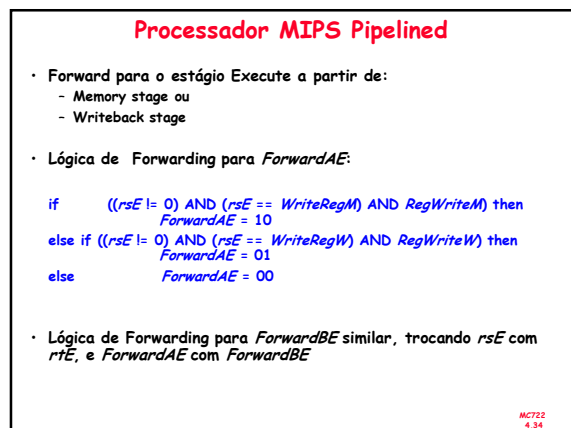
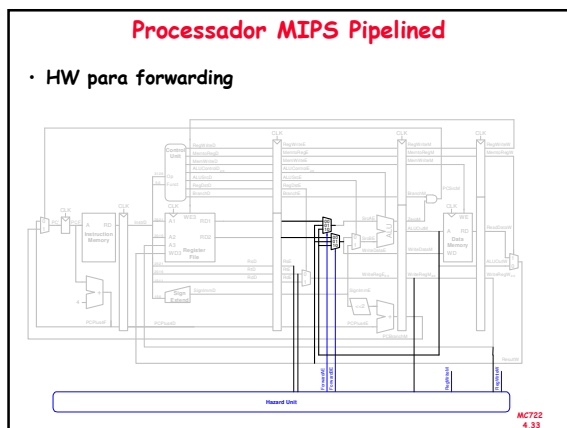
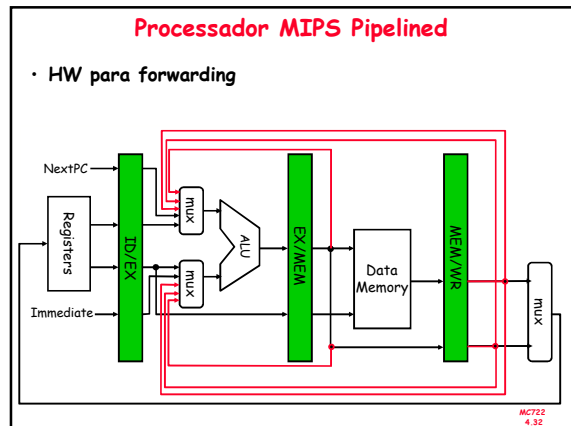
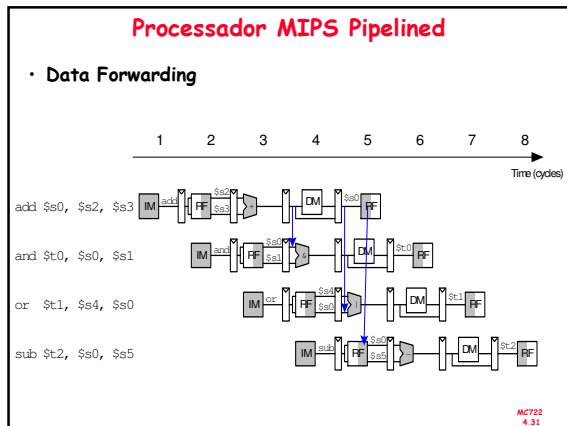
MCT22
4.29

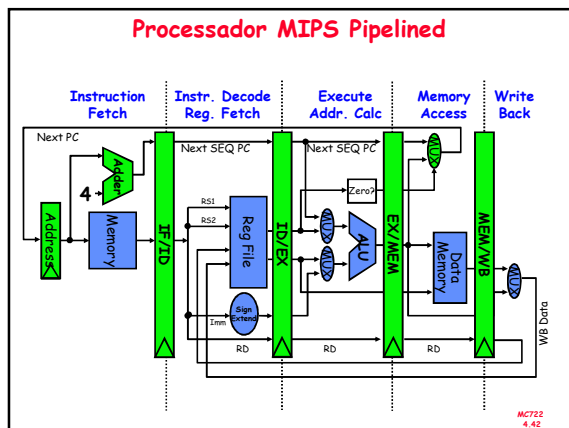
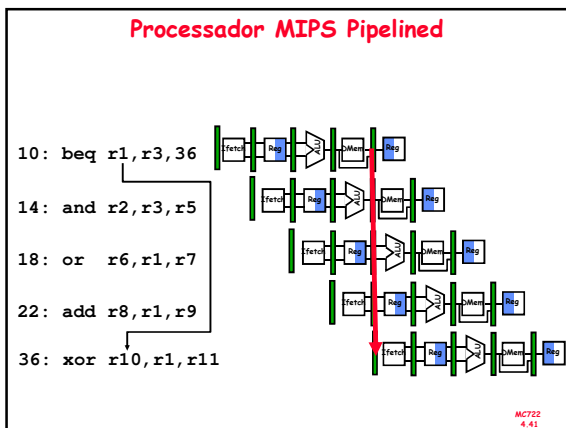
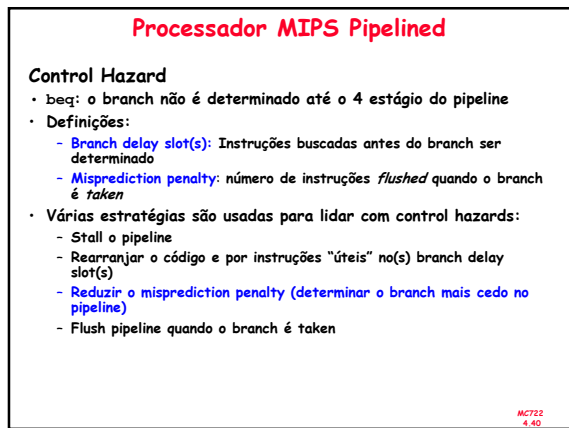
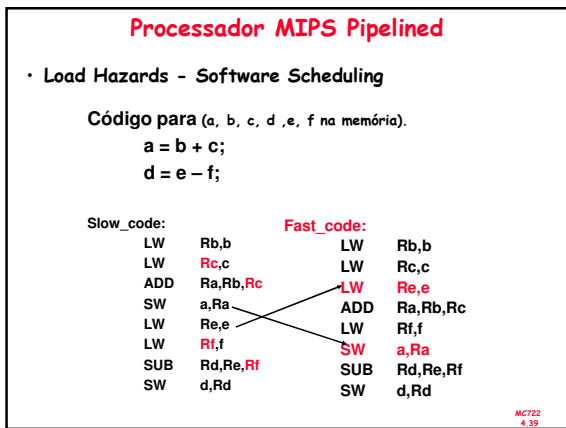
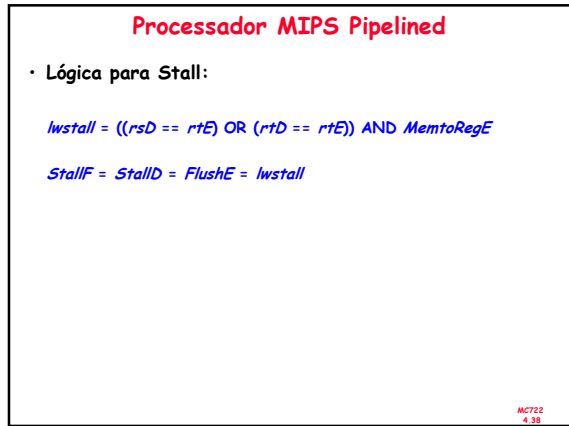
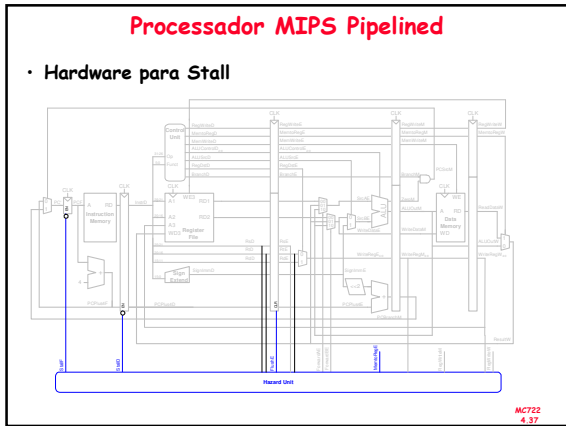
Processador MIPS Pipelined

- Data Forwarding



MCT22
4.30





Processador MIPS Pipelined

Exemplo: Impacto do Branch Stall no Desempenho

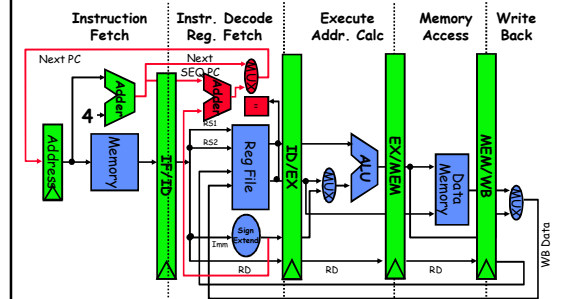
- Se CPI = 1, 30% branches, 3-cycle stall

$$\Rightarrow \text{CPI} = 1.9!$$

- 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 $\text{regs} = \text{ou} \neq$
- 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

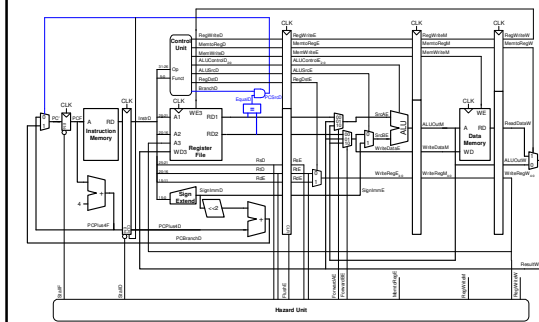
MCT22
4.43

Processador MIPS Pipelined



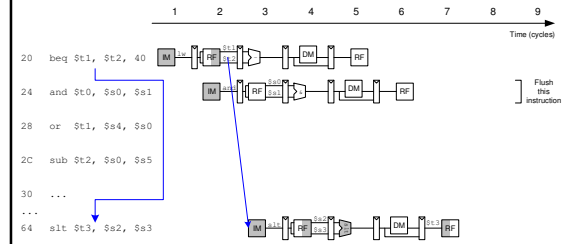
MCT22
4.44

Processador MIPS Pipelined



MCT22
4.45

Processador MIPS Pipelined



MCT22
4.46

Processador MIPS Pipelined

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 tomado
- 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

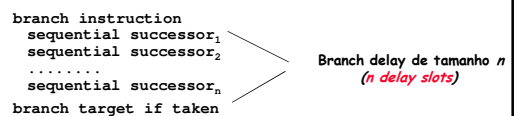
MCT22
4.47

Processador MIPS Pipelined

Alternativas para Branch Hazard

#4: Delayed Branch

- Define-se que o branch será tomado **APÓS** a uma dada quantidade de instruções



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

MCT22
4.48

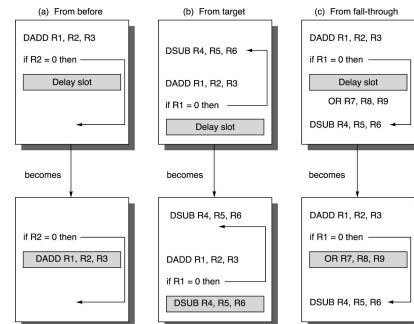
Processador MIPS Pipelined

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)

MCT22
4.49

Processador MIPS Pipelined



MCT22
4.50

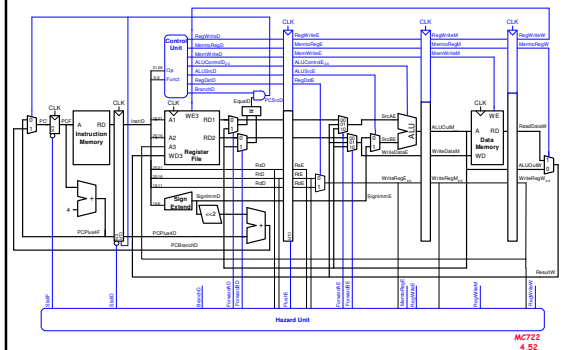
Processador MIPS Pipelined

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

MCT22
4.51

Processador MIPS Pipelined

Tratando data e control hazards



MCT22
4.52

Processador MIPS Pipelined

Control Forwarding e Stalling Hardware

- Lógica de Forwarding:
 - $ForwardAD = (rsD \neq 0) \text{ AND } (rsD == WriteRegM) \text{ AND } RegWriteM$
 - $ForwardBD = (rtD \neq 0) \text{ AND } (rtD == WriteRegM) \text{ AND } RegWriteM$
- Lógica de Stalling:
 - $branchstall = BranchD \text{ AND } RegWriteE \text{ AND } (WriteRegE == rsD \text{ OR } WriteRegE == rtD) \text{ OR } BranchD \text{ AND } MemtoRegM \text{ AND } (WriteRegM == rsD \text{ OR } WriteRegM == rtD)$
 - $StallF = StallD = FlushE = ldstall \text{ OR } branchstall$

MCT22
4.53

Processador MIPS Pipelined

Desempenho do MIPS Pipelined

- Ideal: $CPI = 1, IPC = 1$
- Porém devido aos stall (causados por loads e branches)
- SPECINT2000 benchmark:
 - 25% loads
 - 10% stores
 - 11% branches
 - 2% jumps
 - 52% R-type
- Suponha que:
 - 40% dos loads são usados pela próxima instrução
 - 25% dos branches são mispredicted

Qual o CPI médio?
 $Average\ CPI = (0.25)(1.4) + (0.1)(1) + (0.11)(1.25) + (0.02)(2) + (0.52)(1) = 1.15$

MCT22
4.54

Processador MIPS Pipelined

Desempenho do MIPS Pipelined

- Pipelined processor critical path:

$$T_c = \max \{$$

$$t_{pcq} + t_{mem} + t_{setup}$$

$$2(t_{Rfread} + t_{mux} + t_{eq} + t_{AND} + t_{mux} + t_{setup})$$

$$t_{pcq} + t_{mux} + t_{mux} + t_{ALU} + t_{setup}$$

$$t_{pcq} + t_{memwrite} + t_{setup}$$

$$2(t_{pcq} + t_{mux} + t_{Rfwrite}) \}$$

MCT22
4.55

Processador MIPS Pipelined

Exemplo de Desempenho

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{Rfread}	150
Register file setup	$t_{Rfsetup}$	20
Equality comparator	t_{eq}	40
AND gate	t_{AND}	15
Memory write	$t_{memwrite}$	220
Register file write	$t_{Rfwrite}$	100 ps

$$T_c = 2(t_{Rfread} + t_{mux} + t_{eq} + t_{AND} + t_{mux} + t_{setup})$$

$$= 2[150 + 25 + 40 + 15 + 25 + 20] \text{ ps} = 550 \text{ ps}$$

MCT22
4.56

Processador MIPS Pipelined

Exemplo de Desempenho

- Para um programa que executa 100 bilhões de instruções em um processador MIPS pipelined,

- CPI = 1.15
- $T_c = 550 \text{ ps}$

$$\text{Execution Time} = (\# \text{ instructions}) \times \text{CPI} \times T_c$$

$$= (100 \times 10^9)(1.15)(550 \times 10^{-12})$$

$$= 63 \text{ seconds}$$

MCT22
4.57

Processador MIPS Pipelined

Exemplo de Desempenho

Processor	Execution Time (seconds)	Speedup (single-cycle is baseline)
Single-cycle	95	1
Multicycle	133	0.71
Pipelined	63	1.51

MCT22
4.58