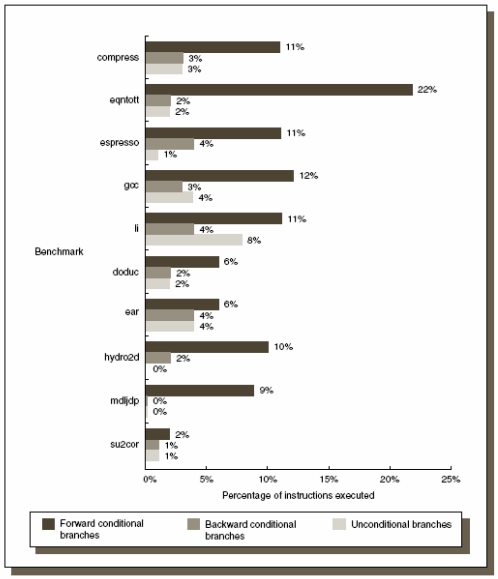
Aula 07: *Control Hazards*, Branches e Interrupções

Características de Branches



Características de Branches

- Programas inteiros
  - 13% das instrs. são branches condicionais para frente
  - 3% das instrs. são dos branches condicionais para trás
  - 4% das instrs. são branches incondicionais
- Programas FP´s
  - 7% das instrs. são branches condicionais para frente
  - 2% das instrs. são dos branches condicionais para trás
  - 1% das instrs. são branches incondicionais
- Dependências adicionais: precisamos saber quais branches são tomados ou não
  - 67% dos branches são tomados
  - 60% dos branches para frente são tomados (if-then-else)
  - 85% dos branches para trás são tomados (while)

Características de Branches

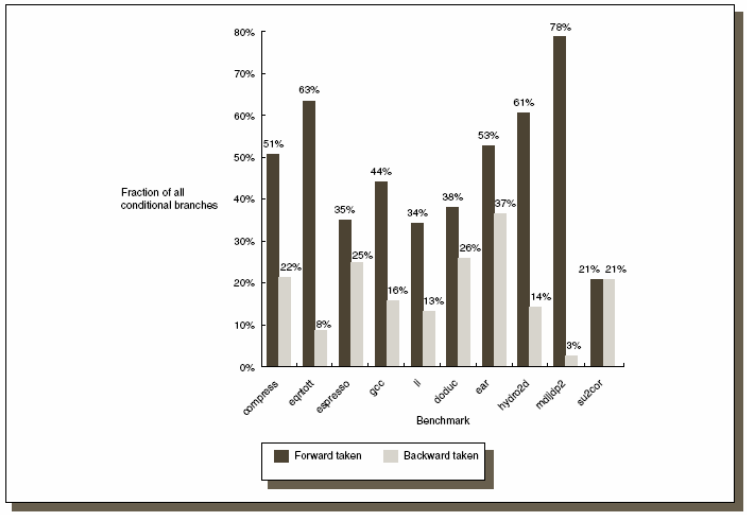
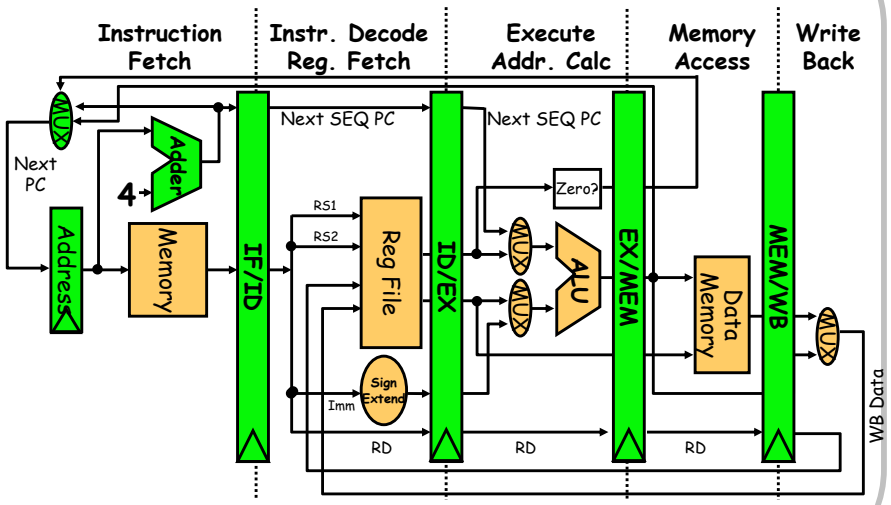


FIGURE 3.25 Together the forward and backward taken branches account for an average of 67% of all conditional branches.

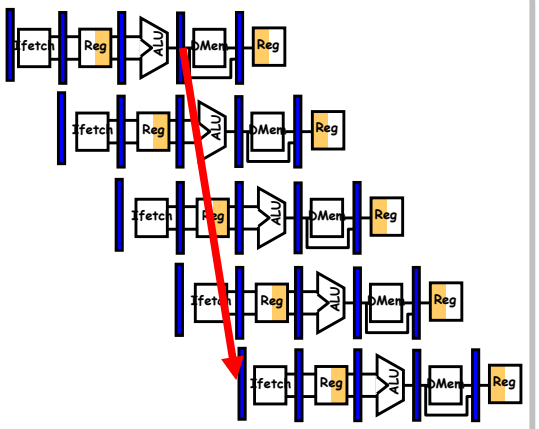
MIPS Datapath (Figure A.18, Page A-31)



- **Data-stationary control**
  - **local decode for each instruction phase / pipeline stage**

Control Hazard on Branches three-Stage Stall

```
12: beq r1,r3,36
16: and r2,r3,r5
20: or  r6,r1,r7
24: add r8,r1,r9
36: xor r10,r1,r11
```



What do you do with the 3 instructions in between?

How do you do it?

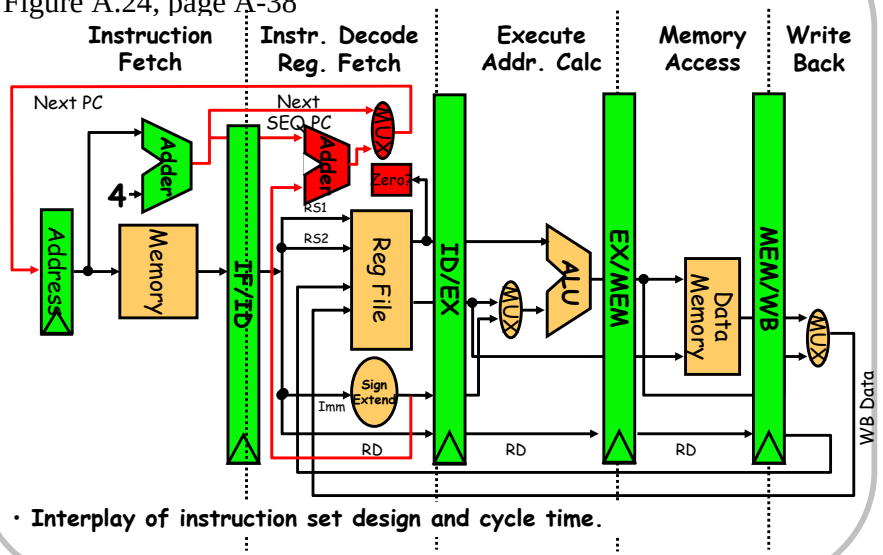
Where is the "commit"?

Branch Stall Impact

- If CPI = 1, 30% branch,
  Stall 3 cycles => new CPI = 1.9!
- Two part solution:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier
- MIPS branch tests if register = 0 or ≠ 0
- MIPS Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty for branch versus 3

Pipelined MIPS Datapath

Figure A.24, page A-38



- **Interplay of instruction set design and cycle time.**

Four Branch Hazard Alternatives

#1: Stall until branch direction is clear

#2: Predict Branch Not Taken

- Execute successor instructions in sequence
- "Squash" instructions in pipeline if branch actually taken
- Advantage of late pipeline state update
- 47% MIPS branches not taken on average
- PC+4 already calculated, so use it to get next instruction

#3: Predict Branch Taken

- 53% MIPS branches taken on average
- But haven't calculated branch target address in MIPS
  - MIPS still incurs 1-cycle branch penalty
  - Other machines: branch target known before outcome

Four Branch Hazard Alternatives

#4: Delayed Branch

- Define branch to take place AFTER a following instruction

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

Branch delay* de tamanho *n

- 1-slot delay allows proper decision and branch target address in 5-stage pipeline
- MIPS uses this
- Experience has shown it to be a bad ISA design decision

Previsão de Branches como não-tomados no DLX

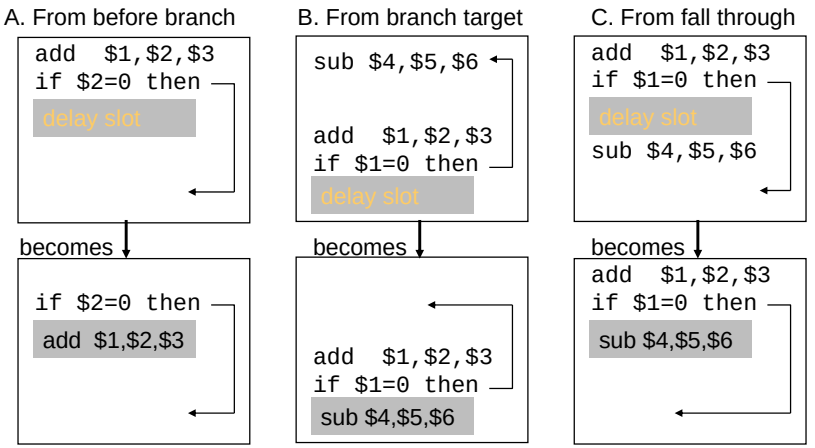
Not Taken

Instrução	1	2	3	4	5	6	7	8	9
branch	IF	ID	EX	MEM	WB				
i+1		IF	ID	EX	MEM	WB			
I+2			IF	ID	EX	MEM	WB		

Taken

Instrução	1	2	3	4	5	6	7	8	9
branch	IF	ID	EX	MEM	WB				
t		IF	IF	ID	EX	MEM	WB		
t+1				IF	ID	EX	MEM	WB	

Scheduling Branch Delay Slots (Fig A.14)



- A is the best choice, fills delay slot & reduces instruction count (IC)
- In B, the `sub` instruction may need to be copied, increasing IC
- In B and C, must be okay to execute `sub` when branch fails

Delayed Branch

- Compiler effectiveness for single branch delay slot:
  - Fills about 60% of branch delay slots
  - About 80% of instructions executed in branch delay slots useful in computation
  - About 50% (60% x 80%) of slots usefully filled
- Delayed Branch downside: As processors go to deeper pipelines and multiple issue, the branch delay grows and need more than one delay slot
  - Delayed branching has lost popularity compared to more expensive but more flexible dynamic approaches
  - Growth in available transistors has made dynamic approaches relatively cheaper

Delayed Branch: requisitos

Scheduling strategy	Requirements	Improves performance when?
(a) From before	Branch must not depend on the rescheduled instructions.	Always.
(b) From target	Must be OK to execute rescheduled instructions if branch is not taken. May need to duplicate instructions.	When branch is taken. May enlarge program if instructions are duplicated.
(c) From fall through	Must be OK to execute instructions if branch is taken.	When branch is not taken.

Evaluating Branch Alternatives

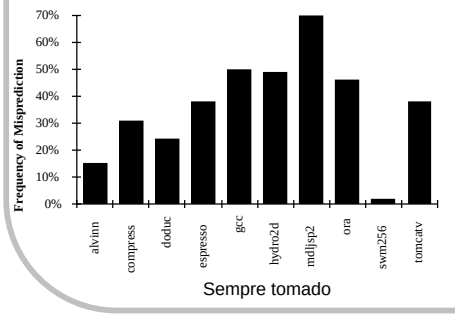
$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

Assume 4% unconditional branch, 6% conditional branch-untaken, 10% conditional branch-taken

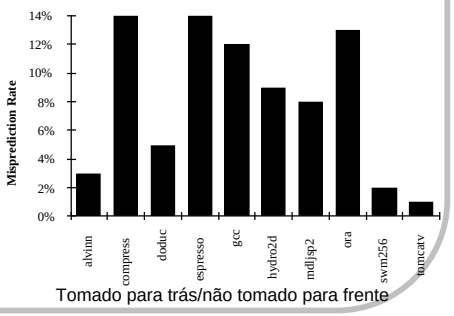
Scheduling scheme	*Branch penalty*	*CPI*	*speedup v. unpipelined*	*speedup v. stall*
Stall pipeline	3	1.60	3.1	1.0
Predict taken	1	1.20	4.2	1.33
Predict not taken	1	1.14	4.4	1.40
Delayed branch	0.5	1.10	4.5	1.45

Outras Técnicas de Previsão Estática

- Melhora estratégia de alocação de instruções no *delay slot*
- Duas estratégias
  - Prediz branches para trás tomados, branches para frente não tomados
  - Baseado em *profiles*: registre comportamento de branches a partir de execuções anteriores



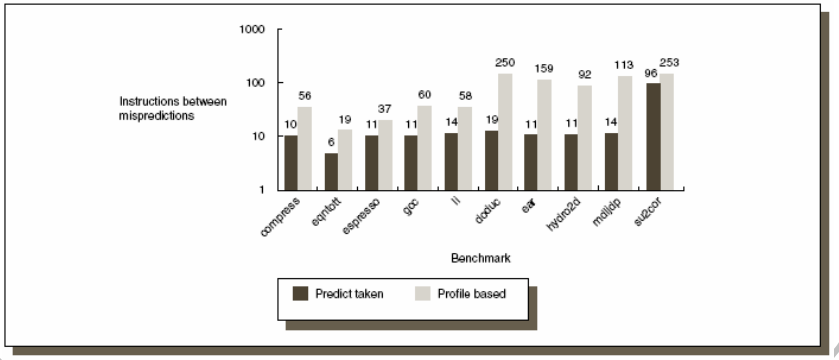
Sempre tomado



Tomado para trás/não tomado para frente

Avaliação de Técnicas Estáticas de Previsão

- Previsão incorreta ignora freqüência do *branch*
- "Instruções entre *branches* previstos incorretamente" é uma métrica melhor



Complicações no Pipeline

- O que vimos até agora foi a parte fácil de pipelines
- O que torna pipeline difícil: *interrupções*
  - Não sabemos quando o estado da instrução pode ser salvo

Problems with Pipelining

- Exception: An unusual event happens to an instruction during its execution
  - Examples: divide by zero, undefined opcode
- Interrupt: Hardware signal to switch processor to new instruction stream
  - Example: sound card interrupts when it needs more audio output samples (audio "click" happens if it is left waiting)
- Problem: Exception or interrupt must appear to happen between 2 instructions (I_i and I_{i+1})
  - The effect of all instructions up to and including I_i is totally complete
  - No effect of any instruction after I_i can take place
- Interrupt (exception) handler either aborts program or restarts at instruction I_{i+1}

Complicações no Pipeline

Exception event	IBM 360	VAX	Motorola 680x0	Intel 80x86
I/O device request	Input/output interruption	Device interrupt	Exception (Level 0...7 autovector)	Vectored interrupt
Invoking the operating system service from a user program	Supervisor call interruption	Exception (change mode supervisor trap)	Exception (unimplemented instruction)—on Macintosh	Interrupt (INT instruction)
Tracing instruction execution	Not applicable	Exception (trace fault)	Exception (trace)	Interrupt (single-step trap)
Breakpoint	Not applicable	Exception (breakpoint fault)	Exception (illegal instruction or breakpoint)	Interrupt (breakpoint trap)
Integer arithmetic overflow or underflow; FP trap	Program interruption (overflow or underflow exception)	Exception (integer overflow trap or floating underflow fault)	Exception (floating-point coprocessor errors)	Interrupt (overflow trap or math unit exception)
Page fault (not in main memory)	Not applicable (only in 370)	Exception (translation not valid fault)	Exception (memory-management unit errors)	Interrupt (page fault)

Complicações no Pipeline

Exception event	IBM 360	VAX	Motorola 680x0	Intel 80x86
Misaligned memory accesses	Program interrup-tion (specification exception)	Not applicable	Exception (address error)	Not applicable
Memory protection violations	Program interrup-tion (protection exception)	Exception (access control violation fault)	Exception (bus error)	Interrupt (protection exception)
Using undefined instructions	Program interrup-tion (operation exception)	Exception (opcode privileged/ reserved fault)	Exception (illegal instruction or break-point/unimplemented instruction)	Interrupt (invalid opcode)
Hardware malfunctions	Machine-check interruption	Exception (machine-check abort)	Exception (bus error)	Not applicable
Power failure	Machine-check interruption	Urgent interrupt	Not applicable	Nonmaskable interrupt

Complicações no Pipeline

- Interrupções: 5 instruções executando nos 5 estágios do pipeline
  - Como parar o pipeline?
  - Como recomeçar o pipeline?
  - Qual delas causou a interrupção?

Pipeline stage	Problem exceptions occurring
IF	Page fault on instruction fetch; misaligned memory access; memory-protection violation
ID	Undefined or illegal opcode
EX	Arithmetic exception
MEM	Page fault on data fetch; misaligned memory access; memory-protection violation
WB	None

Parando e Re-executando Instruções

- Dentro da instrução
  - Força instrução de *trap* no pipeline
  - Até o *trap* ser executado, desligue todas as escritas (salvamento de estado)
  - Rotina de SO primeiramente salva PC para reinicialização
- E se máquina tiver *delayed branches*?
  - Temos que salvar mais de um PC
  - Não poderemos recriar estado da máquina
- E se máquina tiver instruções que salvem o estado antes do final?
  - Veremos mais adiante

Complicações no Pipeline

Exception type	Synchronous vs. asynchronous	User request vs. coerced	User maskable vs. nonmaskable	Within vs. between instructions	Resume vs. terminate
I/O device request	Asynchronous	Coerced	Nonmaskable	Between	Resume
Invoke operating system	Synchronous	User request	Nonmaskable	Between	Resume
Tracing instruction execution	Synchronous	User request	User maskable	Between	Resume
Breakpoint	Synchronous	User request	User maskable	Between	Resume
Integer arithmetic overflow	Synchronous	Coerced	User maskable	Within	Resume
Floating-point arithmetic overflow or underflow	Synchronous	Coerced	User maskable	Within	Resume
Page fault	Synchronous	Coerced	Nonmaskable	Within	Resume
Misaligned memory accesses	Synchronous	Coerced	User maskable	Within	Resume
Memory-protection violations	Synchronous	Coerced	Nonmaskable	Within	Resume
Using undefined instructions	Synchronous	Coerced	Nonmaskable	Within	Terminate
Hardware malfunctions	Asynchronous	Coerced	Nonmaskable	Within	Terminate
Power failure	Asynchronous	Coerced	Nonmaskable	Within	Terminate

Complicações no Pipeline

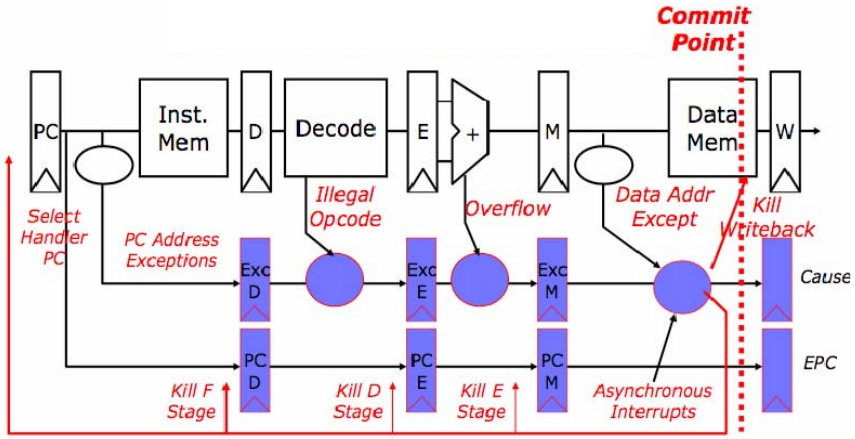
- Exceções simultâneas em mais de um estágio do pipeline
  - Load com *page fault* para busca de dados em MEM
  - Add com *page fault* para busca de instrução em IF
  - Falha de Add acontecerá antes da falha do Load
- Solução #1
  - Vetor de status de interrupção por instrução
  - Atrase verificação até último estágio, então não deixe estado ser alterado para executar exceção
- Solução #2
  - Interrompa tão logo quanto exceção ocorra
  - Recomece tudo que está incompleto

* Outra vantagem para deixar mudança de estado para o último do pipeline!

Complicações no Pipeline

- Modos de endereçamento complexos e instruções complexas
- Modos de endereçamento: autoincremento causa mudança de estado durante execução
  - E se uma interrupção ocorrer? Necessário restaurar estado
  - Adicionam hazards tipo WAR e WAW
- Instruções de movimentação de blocos de memória
  - Precisam lidar com *page faults* múltiplas
  - Executam por muitos ciclos: salvam estado intermediário (8086)
- *Condition Codes*

Precise Exceptions in Static Pipelines

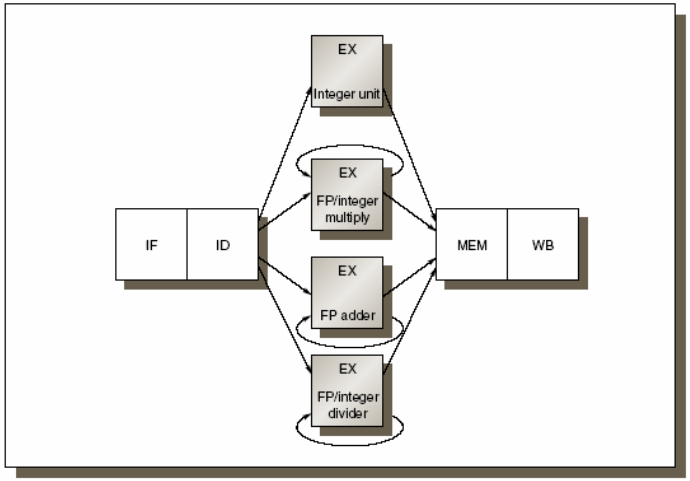


Key observation: architected state only changes in memory- and register-write stages.

Pipeline Multiciclos MIPS

1. The main integer unit that handles loads and stores, integer ALU operations, and branches.
2. FP and integer multiplier.
3. FP adder that handles FP add, subtract, and conversion.
4. FP and integer divider.

Pipeline Multiciclos MIPS



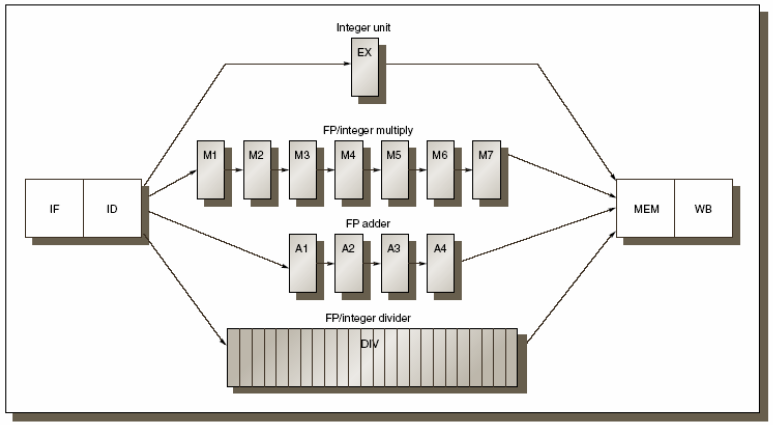
Pipeline Multiciclos MIPS

Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

FIGURE 3.43 Latencies and initiation intervals for functional units.

Complicações no Pipeline

- *Floating Point*: tempo de execução muito longo



Hazards e Forwarding em Pipelines com maiores latências

1. Because the divide unit is not fully pipelined, *structural hazards* can occur. These will need to be detected and issuing instructions will need to be stalled.
2. Because the instructions have varying running times, the number of register writes required in a cycle can be larger than 1.
3. WAW hazards are possible, since instructions no longer reach WB in order. Note that WAR hazards are not possible, since the register reads always occur in ID.
4. Instructions can complete in a different order than they were issued, causing problems with exceptions; we deal with this in the next subsection.
5. Because of longer latency of operations, stalls for RAW hazards will be more frequent.

Interrupções Precisas

- *Pipeline* pode ser interrompido de tal forma que instruções anteriores à exceção completem e instruções posteriores à exceção possam ser reinicializadas
  - Isso é possível? Não, senão não estaríamos vendo isso aqui
  - Ex:

```
DIVF F0,F2,F4
ADDF F10,F10,F8
SUBF F12,F12,F14
```

  - Alpha, Power-2 e R8000 possuem dois modos de operação:
    - c/ interrupções precisas: mais lento
    - s/ interrupções precisas: velocidade máxima do processador

Pipeline Stages (R4000)

Stage	Functional unit	Description
A	FP adder	Mantissa ADD stage
D	FP divider	Divide pipeline stage
E	FP multiplier	Exception test stage
M	FP multiplier	First stage of multiplier
N	FP multiplier	Second stage of multiplier
R	FP adder	Rounding stage
S	FP adder	Operand shift stage
U		Unpack FP numbers

FIGURE 3.55 The eight stages used in the R4000 floating-point pipelines.

Complicações no Pipeline

- Unidades podem ter *pipelines* internos para poder iniciar instruções sem ter que esperar instruções anteriores completarem

FP instruction	Latency	Initiation interval	Pipe stages
Add, subtract	4	3	U,S+A,A+R,R+S
Multiply	8	4	U,E+M,M,M,M,N,N+A,R
Divide	36	35	U,A,R,D^{27},D+A,D+R,D+A,D+R,A,R
Square root	112	111	U,E,$(A+R)^{108}$,A,R
Negate	2	1	U,S
Absolute value	2	1	U,S
FP compare	3	2	U,A,R

FIGURE 3.56 The latencies and initiation intervals for the FP operations both depend on the FP unit stages that a given operation must use. (MIPS R4000)

Complicações no Pipeline

- *Divide* e *Square Root* gastam de 10× a 30× mais tempo do que *Add*
  - O que acontece com interrupções?
  - Adicionam *hazards* de WAR e WAW já que instruções não ficam no *pipeline* pelo mesmo tempo

Resumo de Introdução ao Pipeline

- *Hazards* limitam performance
  - Estrutural: falta de recursos de hardware
  - Dados: necessita de *forwarding* e escalonamento de instruções
  - Controle: avaliação do PC mais cedo, *delayed branch*, previsão do *branch*
- Aumento da profundidade causa maior impacto com *hazards;*
- Interrupções, conjunto de instruções e FPs tornam a implementação do *pipeline* mais difícil;
- Compiladores reduzem penalidades de *hazards* de controle e dados;