Aula 07: *Control Hazards*, Branches e Interrupções

Datapath do DLX com *Pipeline*

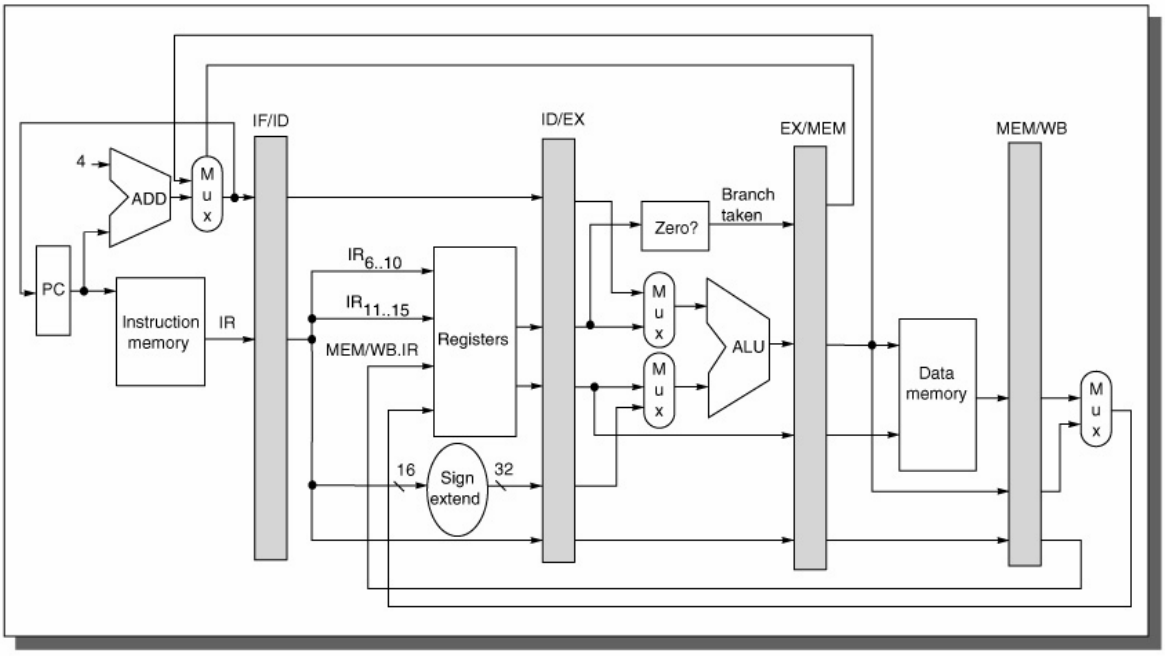
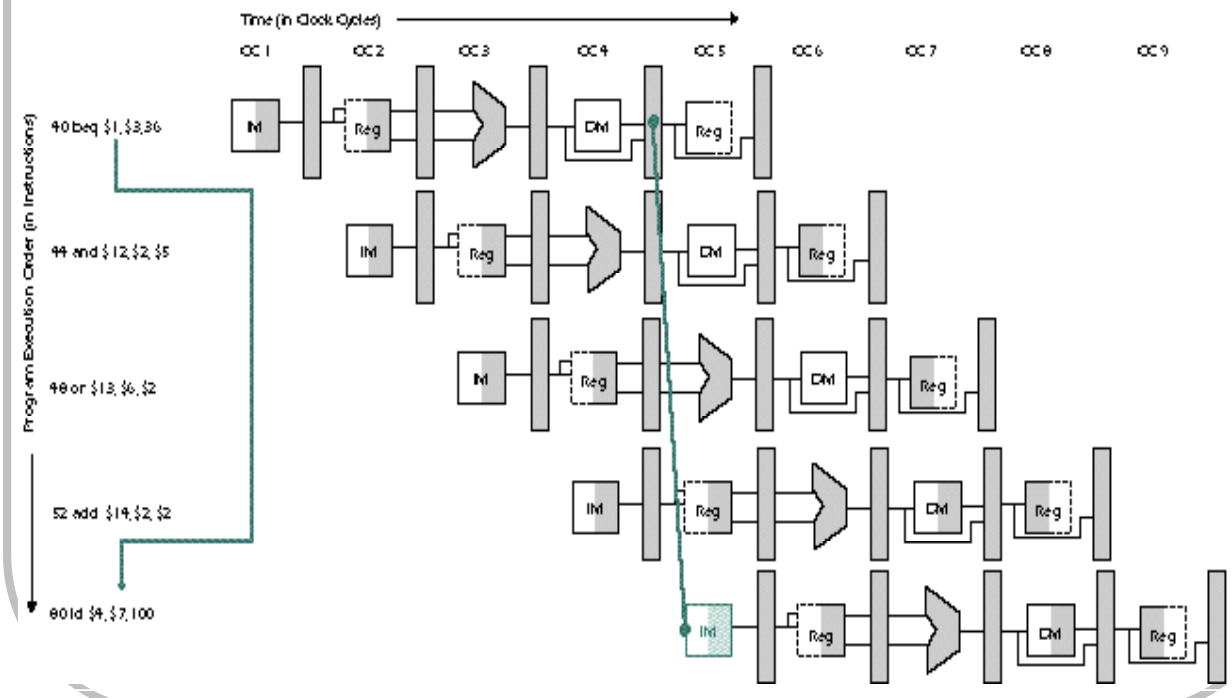


Figura 3.4

Representação Esquemática do *Pipeline*

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

Hazards de Controle *Stall* de 3 ciclos



Efeito de Branches Solução "Inocente"

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

Vamos parar o *pipeline* até endereço ser calculado:

- Simples de implementar

Impacto do *Stall* Devido a *Branches*

- Se CPI = 1, 30% de branches, Stall de 3 ciclos => novo CPI = 1.9!
  1/2 da performance é perdida !!!
- Solução em duas partes:
  - Determinar se branch é tomado ou não mais cedo, E
  - Calcular endereço de branch tomado mais cedo
- DLX testa para branch se registrador é 0 ou não
- Solução:
  - Mover teste de 0 para estágio ID/RF
  - Aloque um somador para calcular novo PC no estágio ID/RF
  - Reduz penalidade de 3 ciclos para 1!

Datapath do DLX com *Pipeline*

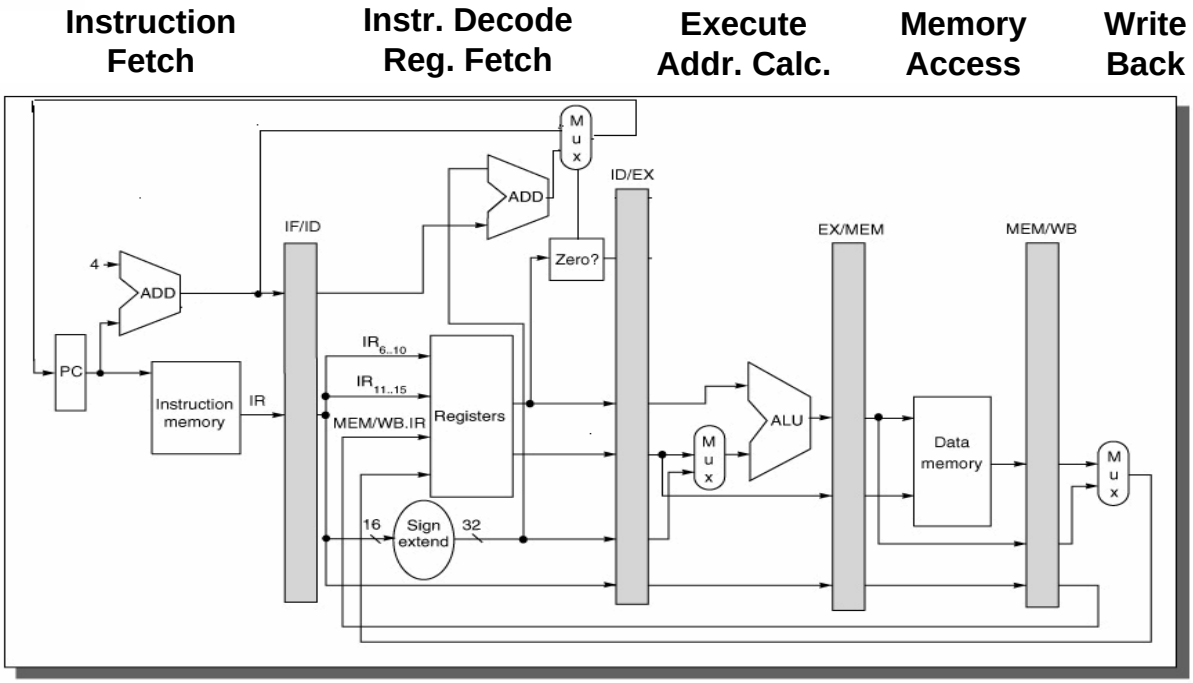


Figura 3.22 **ALGUMAS EDIÇÕES DO LIVRO TEXTO POSSUEM ERRO**

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)

Quatro Alternativas para Melhorar Performance de Branches

#1: Pára pipeline até sabermos se o branch será tomado ou não

#2: Previsão de branch como não tomado

- Continue executar instruções na sequência do pipeline
- "Mate" execução das instruções que estão no pipe se branch for tomado
- Vantagem se estado é alterado somente mais tarde (pode voltar atrás sem ter que desfazer nada)
- +- 1/3 dos branches no DLX não são tomados na média
- PC+4 já é calculado para a próxima instrução

#3: Previsão de branch como tomado

- +- 2/3 dos branches são tomados na média
- Mas no DLX o endereço demora de qualquer jeito 1 ciclo para ser calculado (não há ganho)
  - Outras máquinas: destino conhecido antes

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	

Quatro Alternativas para Melhorar Performance de Branches

#4: *Delayed Branch*

– Branch completa todas as instruções sucessoras

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

Branch delay* de tamanho *n

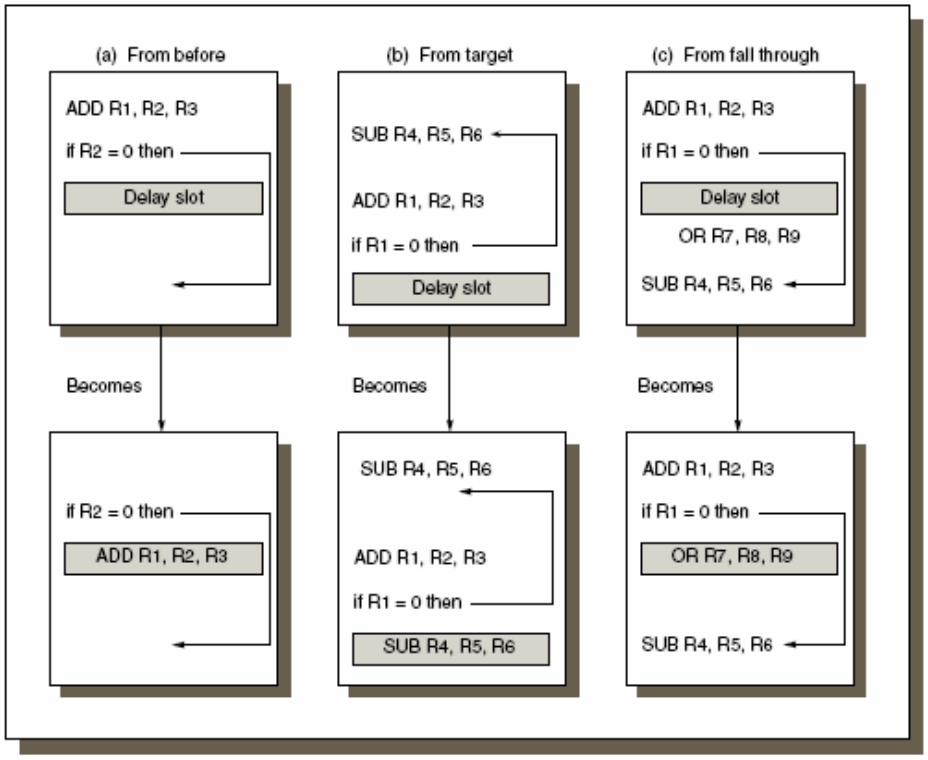
– Delay de 1 slot permite decisão para o endereço destino em um pipeline de 5 estágios

– DLX usa delay de 1 slot

Delayed Branch

- De onde vêm as instruções para encher o *branch delay slot*?
  - Instruções vindas anteriores ao branch
  - Vindas do destino: melhora desempenho só se o branch for tomado
  - Vindas do código sequencial: melhora desempenho só se o branch não for tomado
  - *Cancelling branches:* permite compilador preencher a instrução e decidir quando seqüência será cancelada
- Efetividade do compilador em preencher *branch delay slot* único:
  - Preenche aproximadamente 60% dos *branch delay slots*
  - Cerca de 80% das instruções executadas realizam código útil no programa
  - Cerca de 50% (60% x 80%) dos *slots* são preenchidos eficazmente

Delayed Branch



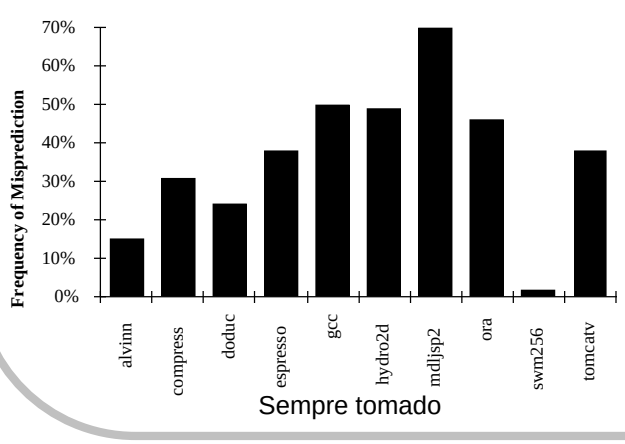
Avaliando Alternativas de Branch

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

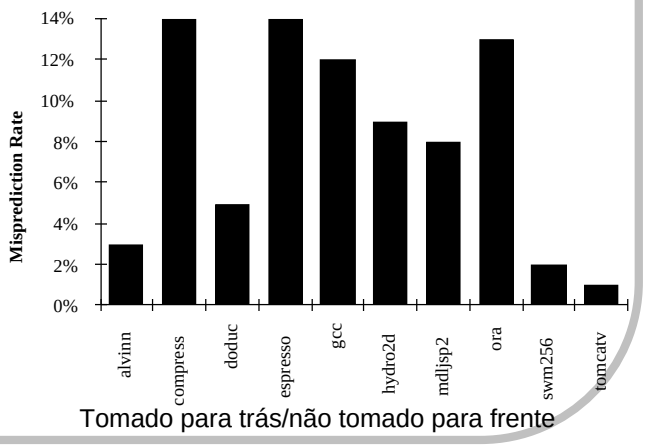
Escalonamento	*Penalidade*	*CPI*	*Speedup v. s/ pipeline*
Para pipe	3,0	1,42	3,5
Assume tomado	1,0	1,14	4,4
Assume não tomado	1,0	1,09	4,5
delayed branch	0,5	1,07	4,6

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 execução anterior



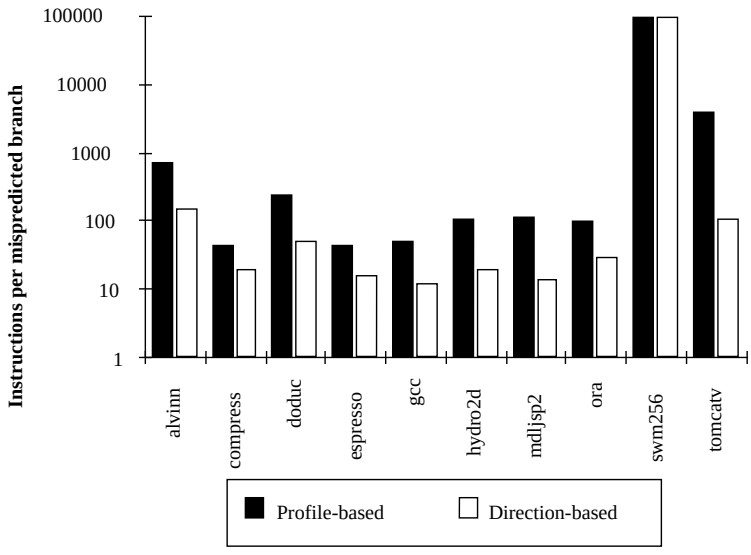
Sempre tomado



Tomado para trás/não tomado para frente

Avaliação de Técnicas Estáticas de Previsão

- Previsão errada ignora frequência do branch
- "Instruções entre branches previstos errôneamente" 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

Complicações no Pipeline

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

Estágio	*Causa da interrupção*
IF	*Page fault* na busca da instrução; acesso não alinhado; violação de proteção de memória
ID	Código de instrução inválido, protegido
EX	Erro de operação de ALU
MEM	*Page fault* na busca do dado; acesso não alinhado; violação de proteção de memória

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 interruption (specification exception)	Not applicable	Exception (address error)	Not applicable
Memory protection violations	Program interruption (protection exception)	Exception (access control violation fault)	Exception (bus error)	Interrupt (protection exception)
Using undefined instructions	Program interruption (operation exception)	Exception (opcode privileged/reserved fault)	Exception (illegal instruction or breakpoint/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

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

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?
  - Vamos ver mais adiante

Complicações no Pipeline

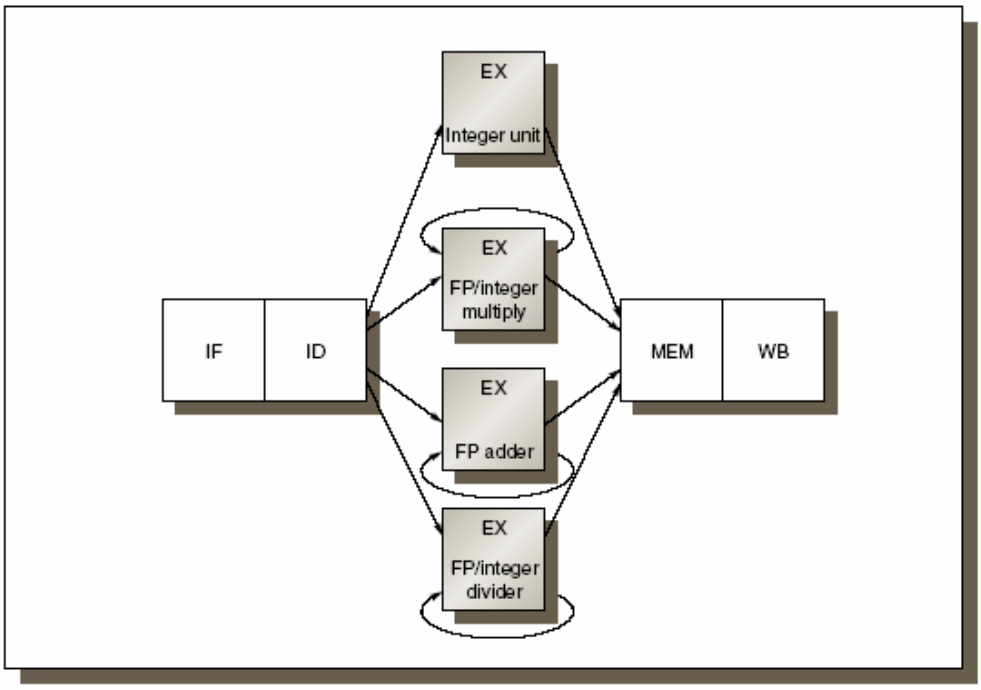
- Exceções simultâneas em mais de um estágio do pipeline
  - Load com *page fault* para busca de dados em MEM
  - Add com *page fault* para busca de instrução em IF
  - Falha de Add acontecerá antes da falha do Load
- Solução #1
  - Vetor de status de interrupção por instrução
  - Atrase verificação 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 por último no 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? Precisa restaurar estado
  - Adiciona hazards de WAR e WAW
- Instruções de movimentação de blocos de memória
  - Precisa lidar com *page faults* múltiplas
  - Executam por muitos ciclos: salvam estado intermediário (8086)
- *Condition Codes*

Pipeline Multiciclos (*DLX*)



Pipeline Multiciclos (*DLX*)

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.

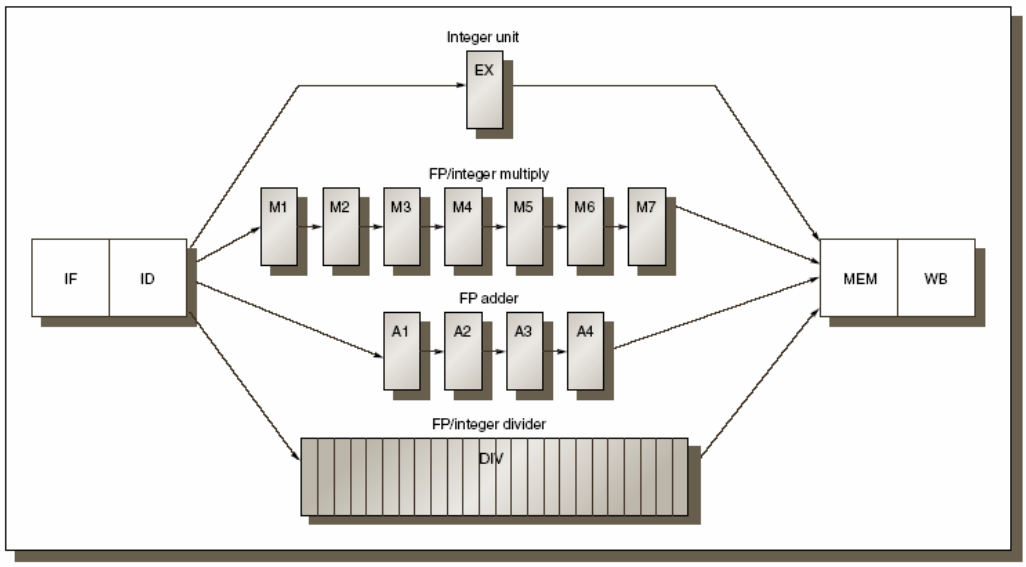
Pipeline Multiciclos (*DLX*)

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



Complicações no Pipeline

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

Instrução FP	*Latência*	*Initiation Rate*	*(MIPS R4000)*
Add, Subtract	4	3	
Multiply	8	4	
Divide	36	35	
Square root	112	111	
Negate	2	1	
Absolute value	2	1	
FP compare	3	2	

Complicações no Pipeline

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

Hazards e Forwarding em Pipelines com grandes 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 a exceção completam e instruções posteriores à exceção podem ser reinicializadas
  - Isto 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

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 fazem pipeline mais difícil
- Compiladores reduzem penalidades de hazards de controle e dados