Paralelismo a Nível de Instrução

É possível obter maior desempenho computacional com:

- tecnologias mais avançadas, tais como circuitos mais rápidos;
- melhor organização da CPU, tais como o uso de múltiplos registradores e memória cache;
- *pipeline* de instruções.

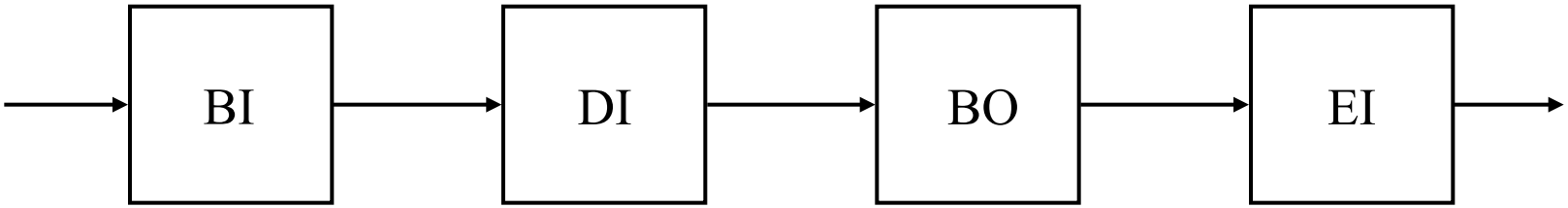
A ideia básica em um *pipeline* de instruções é a de novas entradas serem aceitas, antes que as entradas aceitas previamente tenham terminado.

Esse conceito assume que uma instrução tem vários estágios.

Sequência de eventos num ciclo de instrução:

```
início
      busca de instrução;
      decodificação da instrução;
      cálculo de endereço de operando;
      busca de operando;
      execução da instrução;
      cálculo de endereço do resultado;
      verificação de interrupção;
      se interrupção então tratamento de interrupção;
      cálculo de endereço da próxima instrução;
      volta para busca de instrução;
end.
```

Ao invés da execução sequencial do algoritmo, poderíamos associar cada etapa a um estágio do *pipeline*.



O primeiro estágio busca a instrução e a armazena em uma área de armazenamento temporário.

Quando o segundo estágio está livre, o primeiro passa para ele a instrução armazenada.

Enquanto o segundo estágio está executando essa instrução, o primeiro tira proveito de ciclos de memória que não são usados para buscar e armazenar a próxima instrução.

A busca antecipada de instrução (*instruction prefetch*) ou superposição de busca (*fetch overlap*) consiste em buscar a próxima instrução, enquanto a atual está sendo executada.

O aumento da taxa de execução de instruções, no exemplo anterior, pode não ser possível pois:

- tempo de execução é geralmente maior que o de busca (o estágio de busca pode ter que esperar antes que possa esvaziar a área de armazenamento temporário);
- instruções de desvio condicional fazem com que o endereço da próxima instrução seja desconhecido (o estágio de busca teria de esperar pelo endereço da próxima instrução e o estágio de execução teria que esperar enquanto a próxima instrução é buscada).

Para conseguir maior desempenho, o *pipeline* deve ter o maior número de estágios possível.

Exemplo: Considere um *pipeline* com 6 estágios de mesma duração:

- busca de instrução (BI);
- decodificação de instrução (DI);
- cálculo de operandos (CO);
- busca de operandos (BO);
- execução de instrução (EI);
- escrita de operando (EO).

Pipeline com seis estágios:

tempo	1	2	3	4	5	6	7	8	9	10	11	12	13	14
instrução														
1	BI	DI	CO	BO	EI	EO								
2		BI	DI	CO	BO	EI	EO							
3			BI	DI	CO	BO	EI	EO						
4				BI	DI	CO	BO	EI	EO					
5					BI	DI	CO	BO	EI	EO				
6						BI	DI	CO	BO	EI	EO			
7							BI	DI	CO	BO	EI	EO		
8								BI	DI	CO	BO	EI	EO	
9									BI	DI	CO	BO	EI	EO

Consideramos que cada instrução passa por todos os estágios do *pipeline* (o que nem sempre é necessário), simplificando o hardware.

Consideramos que todos os estágios podem ser executados em paralelo, não havendo conflito, por exemplo, nos acesso à memória (o dado pode estar no cache ou alguns estágios, que requerem acesso à memória, não estão sendo usados).

Se os seis estágios não têm duração igual, existe certa espera envolvida em vários estágios.

Uma instrução de desvio condicional pode invalidar diversas buscas de instrução. Da mesma forma, a ocorrência de interrupção.

Exemplo: Suponha que a instrução 3 seja um desvio condicional para a instrução 15.

tempo	1	2	3	4	5	6	7	8	9	10	11	12	13	14
instrução														
1	BI	DI	CO	BO	EI	EO								
2		BI	DI	CO	BO	EI	EO							
3			BI	DI	CO	BO	EI	EO						
4				BI	DI	CO	BO							
5					BI	DI	CO							
6						BI	DI							
7							BI							
15								BI	DI	CO	BO	EI	EO	
16									BI	DI	CO	BO	EI	EO

Dizer que uma máquina A é *n* vezes mais rápida que uma máquina B significa que:

$$\frac{tempo\ de\ execução_{\mathrm{B}}}{tempo\ de\ execução_{\mathrm{A}}} = n$$

Desempenho é definido como o inverso do tempo de execução:

$$n = \frac{tempo\ de\ execução_{\mathrm{B}}}{tempo\ de\ execução_{\mathrm{A}}} = \frac{\frac{1}{desempenho_{\mathrm{B}}}}{\frac{1}{desempenho_{\mathrm{A}}}} = \frac{desempenho_{\mathrm{A}}}{desempenho_{\mathrm{B}}}$$

A lei de Amdahl define o *speedup* (*S*), que consiste do ganho em desempenho que pode ser obtido ao melhorar determinada característica do computador:

$$S = \frac{\text{desempenho de toda a operação usando a melhoria}}{\text{desempenho de toda a operação sem usar a melhoria}}$$

$$S = \frac{\text{tempo de execução de toda a operação sem usar a melhoria}}{\text{tempo de execução de toda a operação usando a melhoria}}$$

O tempo de ciclo τ de um *pipeline* de instrução é o tempo requerido para avançar um conjunto de instruções um estágio.

O tempo de ciclo pode ser determinado da seguinte maneira:

$$\tau = \max[\tau_i] + d = \tau_m + d,\ 1 \leq i \leq k$$

onde:

τ_m = atraso máximo de estágio
k = número de estágios do *pipeline* de instrução
d = tempo necessário para propagar sinais e dados de um estágio para o próximo

Em geral, d é equivalente ao pulso de um relógio e $\tau_m >> d$.

Exemplo: Suponha que sejam processadas *n* instruções, sem que ocorra desvio.

$T_k = [k+(n-1)]\ \tau$, tempo total de execução

O *speedup* para a execução com o *pipeline* de instruções em relação à execução sem o uso do pipeline é:

$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{[k+(n-1)]\tau} = \frac{nk}{k+(n-1)}$$

Em função do número de instruções executadas sem desvio, no limite ($n \rightarrow \infty$), o fator de aceleração é igual a k.

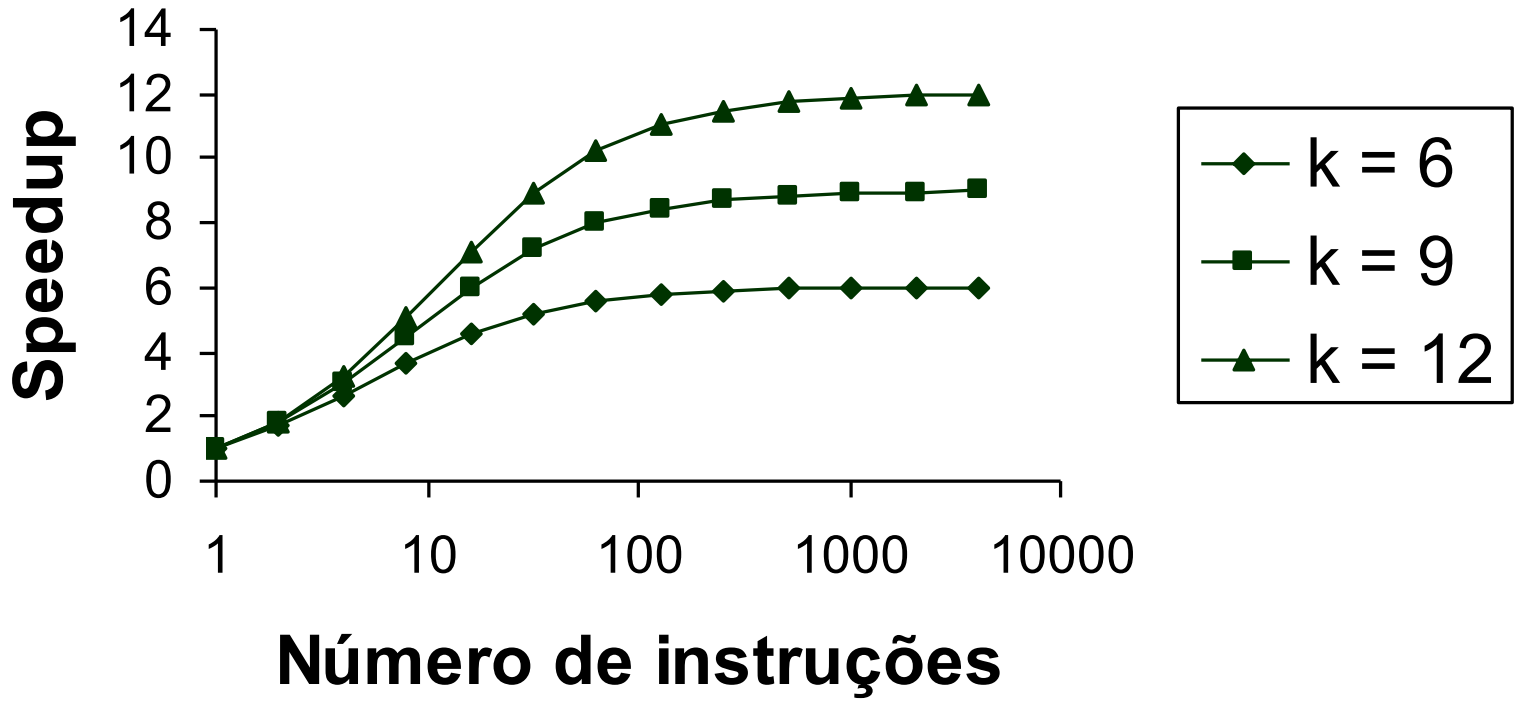
Em função do número de estágios, o fator de aceleração se aproxima do número de instruções que podem ser introduzidas no *pipeline* sem desvio.

Quanto maior o número de estágios do *pipeline*, maior o *speedup*. No entanto, o ganho diminui devido:

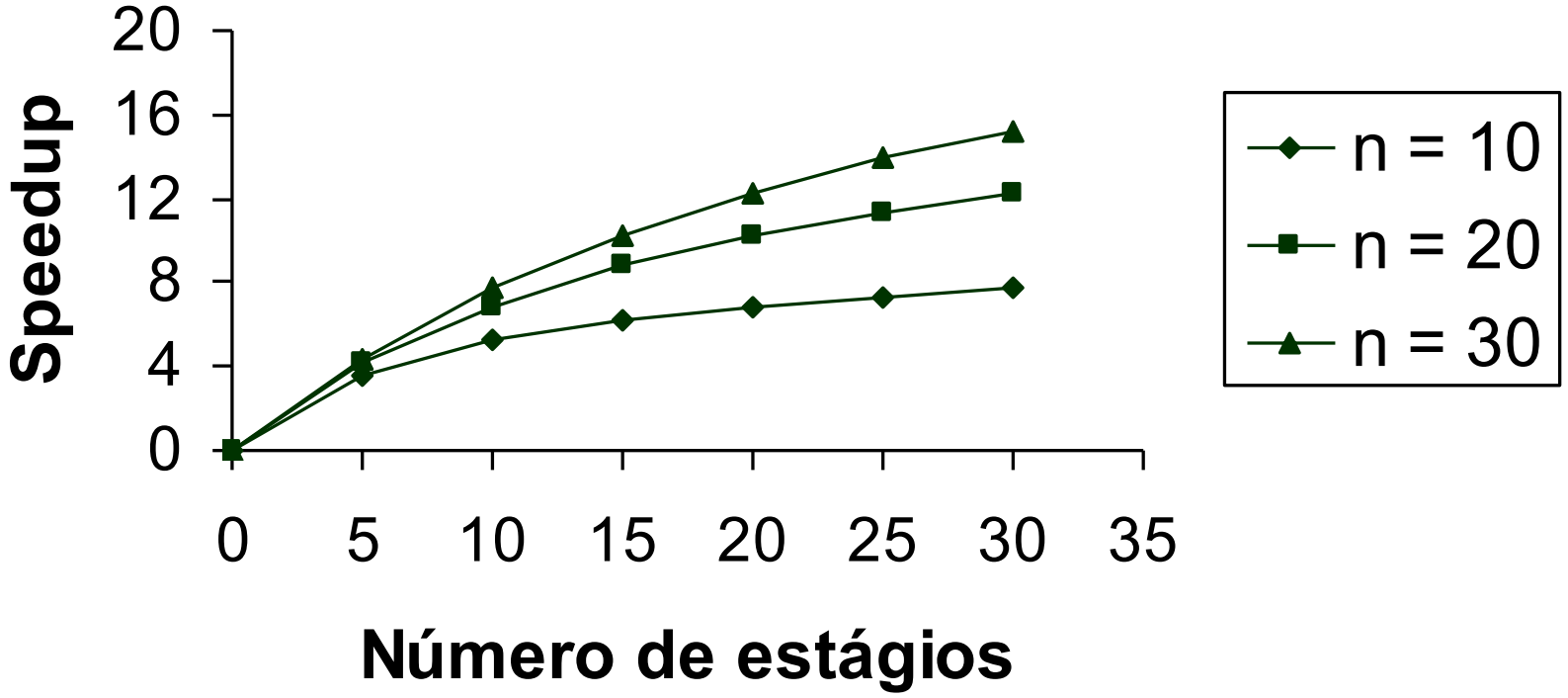
- ao aumento no custo da implementação;
- aos atrasos entre estágios;
- aos atrasos no processo de esvaziamento do *pipeline* quando ocorre instrução de desvio.

Um número de estágios entre 6 e 9 parece ser mais adequado.

Speedup para execução com pipeline de instruções em relação à execução sem pipeline



Speedup para execução com pipeline de instruções em relação à execução sem pipeline

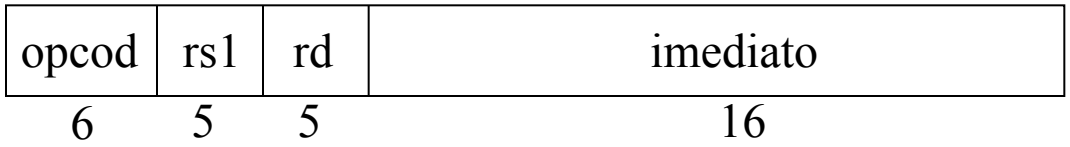


Considere a arquitetura do processador DLX, sem *pipeline*:

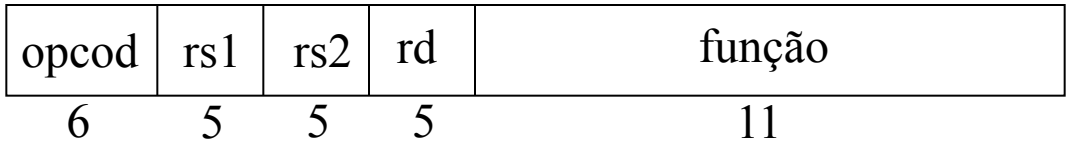
- 32 registradores de 32 bits (R0 a R31);
- 31 registradores de ponto flutuante (F0 a F30);
- endereçamento de dados é imediato ou deslocamento;
- endereçamento de byte, com endereço de 32 bits;
- instruções de carga e armazenamento;
- instruções aritméticas e lógicas;
- instruções de desvio.

Todas as instruções são de 32 bits, com 6 bits para código de operação e 16 bits para endereçamento por deslocamento, constantes imediatas e endereços de desvio relativos ao contador de programas (PC):

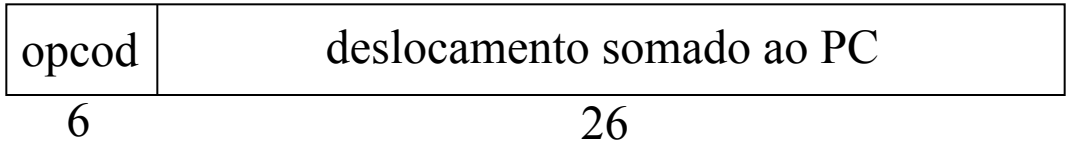
- instrução do tipo I:



- instrução do tipo R:



- instrução do tipo J:



Há quatro classes de instruções: cargas e armazenamentos, operações com a ALU, desvios e operações de ponto flutuante.

Todas as instruções levam, no máximo, cinco ciclos de clock para serem executadas:

1 – ciclo de busca de instrução (IF):

IR ← mem[PC];
NPC ← PC + 4

2 – ciclo de decodificação de instrução/busca de registrador (ID):

A ← regs[$IR_{6 .. 10}$];
B ← regs[$IR_{11 .. 15}$];
Imm ← ($IR_{16 .. 31}$);

3 – ciclo de execução/endereço efetivo (EX):

ALUoutput ← A + Imm; endereçamento de memória

ALUoutput ← A op B; operação entre registradores

ALUoutput←A op Imm; operação entre registrador e imediato

ALUoutput ← NPC + Imm; cálculo do endereço de desvio

Cond ← (A op 0); operação de comparação dependendo do código de operação (*i.e.*, ==)

4 – ciclo de acesso à memória/complemento de desvio (MEM):

LMD ← mem[ALUoutput] ou
mem[ALUoutput] ← B; endereçamento de memória

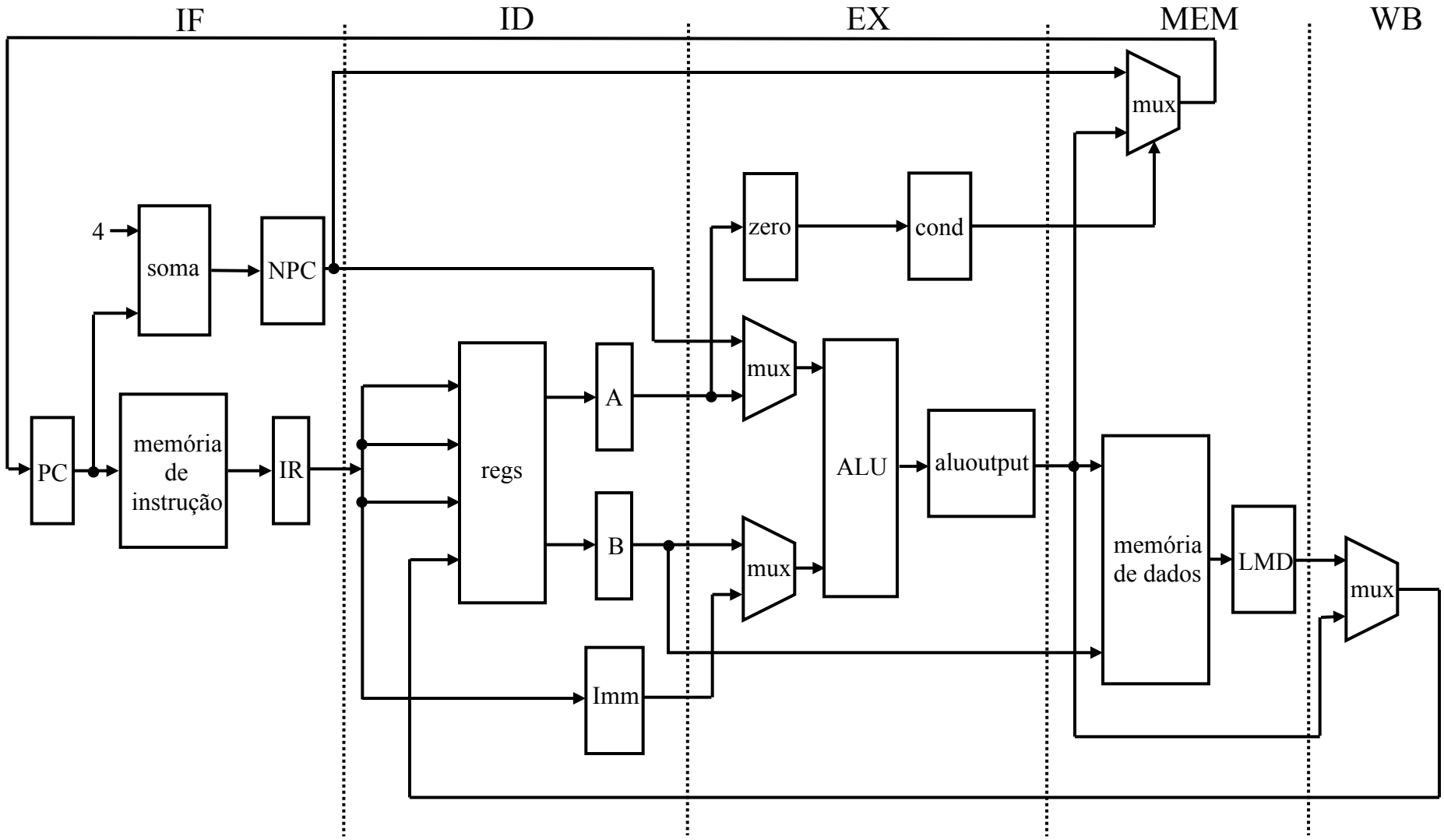
if (cond)
then PC ← ALUoutput
else PC ← NPC; desvio condicional

5 – ciclo de escrita (WB):

regs[$IR_{16 .. 20}$] ← ALUoutput;

regs[$IR_{11 .. 15}$] ← ALUoutput;

regs[$IR_{11 .. 15}$] ← LMD


IF
ID
EX
MEM
WB
4
soma
NPC
zero
cond
mux
PC
memória de instrução
IR
regs
A
B
Imm
mux
mux
ALU
aluoutput
memória de dados
LMD
mux

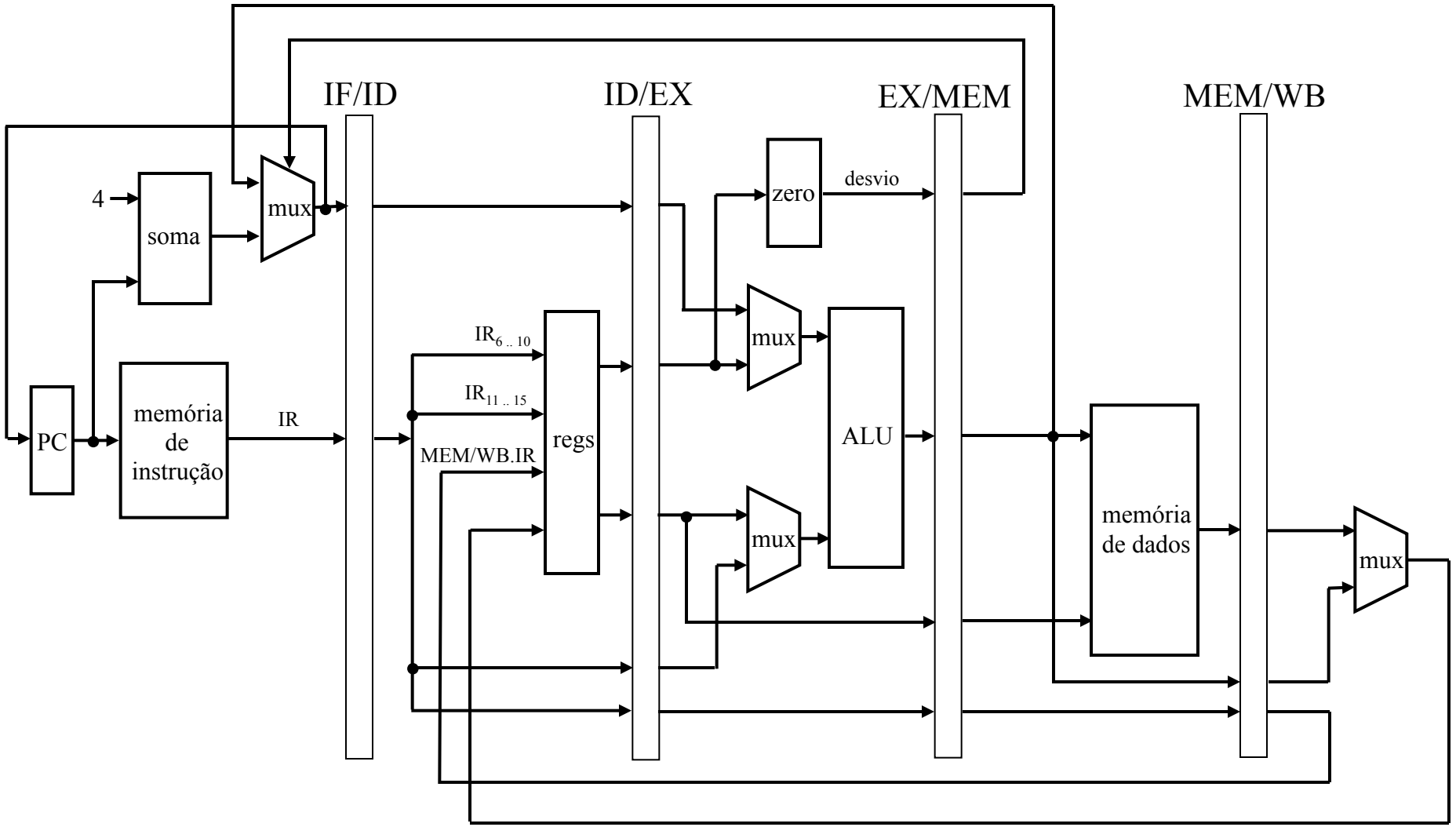
Ao término de cada ciclo de *clock*, cada valor computado durante aquele ciclo e requerido num ciclo mais tarde (quer seja para essa instrução ou a próxima) é escrito em um meio de armazenamento, que pode ser a memória, um registrador de propósito geral, o PC ou um registrador temporário (LMD, Imm, A, B, IR, NPC, ALUoutput ou Cond).

Esses registadores temporários armazenam valores entre ciclos de *clock* para uma instrução, enquanto os outros meios de armazenamento são elementos do estado da arquitetura e guardam valores entre instruções sucessivas.

Nessa arquitetura, instruções de desvio requerem quatro ciclos de *clock* e todas as outras requerem cinco ciclos de *clock*.

Pode-se implementar *pipeline* nessa arquitetura começando uma nova instrução a cada ciclo de *clock* e associando um estágio do *pipeline* a cada ciclo da arquitetura descrita.

	ciclos de clock								
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


IF/ID
ID/EX
EX/MEM
MEM/WB
4
soma
mux
PC
memória
de
instrução
IR
IR6 .. 10
IR11 .. 15
MEM/WB.IR
regs
zero
desvio
mux
mux
ALU
memória
de dados
mux

Os registradores do *pipeline* armazenam tanto dados quanto controle de um estágio do *pipeline* para o próximo. Qualquer valor necessário em um estágio adiante deve ser posto em um desses registradores e copiado de um registrador para outro, até não ser mais requerido.

Por exemplo, o campo de um operando usado em uma escrita ou numa operação da ALU é fornecido pelo registrador do estágio MEM/WB, ao invés do registrador do estágio IF/ID. Isto porque o estágio IF/ID está, no momento, associado a outra instrução que não aquela correspondente à operação no estágio MEM/WB.

Qualquer instrução está ativa em exatamente um estágio do *pipeline* de cada vez.

Estágio	Qualquer instrução		
IF	IF/ID.IR←mem[PC]; IF/ID.NPC,PC←(se EX/MEM.cond então (EX/MEM.NPC) senão (PC+4);		
ID	ID/EX.A←regs[IF/ID.IR$_{6\ ..\ 10}$]; ID/EX.B←regs[IF/ID.IR$_{11\ ..\ 15}$]; ID/EX.NPC←IF/ID.NPC; ID/EX.IR←IF/ID.IR; ID/EX.Imm←IR$_{16\ ..\ 31;}$		
	Instrução para ALU	Carga ou armazenamento	Desvio
EX	EX/MEM.IR←ID/EX.IR; EX/MEM.ALUoutput← ID.EX.A op ID/EX.B; ou EX/MEM.ALUoutput← ID/EX.A op ID/EX.Imm; EX/MEM.cond←0;	EX/MEM.IR←ID/EX.IR; EX/MEM/ALUoutput←ID/EX.Imm; EX/MEM.cond←0; EX/MEM.B←ID/EX.B;	EX/MEM.ALUoutput←ID/EX.NPC+ID/EX.Imm; EX/MEM.cond←(ID/EX.A op 0);
MEM	MEM/WB.IR←EX/MEM.IR; MEM/WB.ALUoutput← EX/MEM.ALUoutput;	MEM/WB.IR←EX/MEM.IR; MEM/WB.LMD← mem[EX/MEM.ALUoutput]; ou mem[EX/MEM.ALUoutput] ←EX/MEM.B;	
WB	Regs[MEM/WB.IR$_{16\ ..\ 20}$] ←MEM/WB.ALUoutput; ou Regs[MEM/WB.IR$_{11\ ..\ 15}$]← MEM/WB.ALUoutput	Regs[MEM/WB.IR$_{11\ ..\ 15}$]←MEM/WB.LMD;	

Se a instrução *i* for um desvio a ser tomado, então o PC será modificado ao final do estágio MEM, após o complemento do cálculo do endereço e comparação.

O método mais simples de tratar com desvios é parar o *pipeline*, assim que um desvio é detetado, até chegar ao estágio MEM, que vai determinar o próximo PC.

Nesse caso, a parada do *pipeline* só ocorre após o estágio ID, quando se identifica que a instrução é um desvio.

Um desvio causa uma parada de três ciclos no *pipeline*. A instrução depois do desvio é buscada, mas é ignorada.

	ciclos de clock									
instrução	1	2	3	4	5	6	7	8	9	10
i (desvio)	IF	ID	EX	MEM	WB					
i+1		IF	parada	parada	IF	ID	EX	MEM	WB	
i+2						IF	ID	EX	MEM	WB
i+3							IF	ID	EX	MEM
i+4								IF	ID	EX
i+5									IF	ID

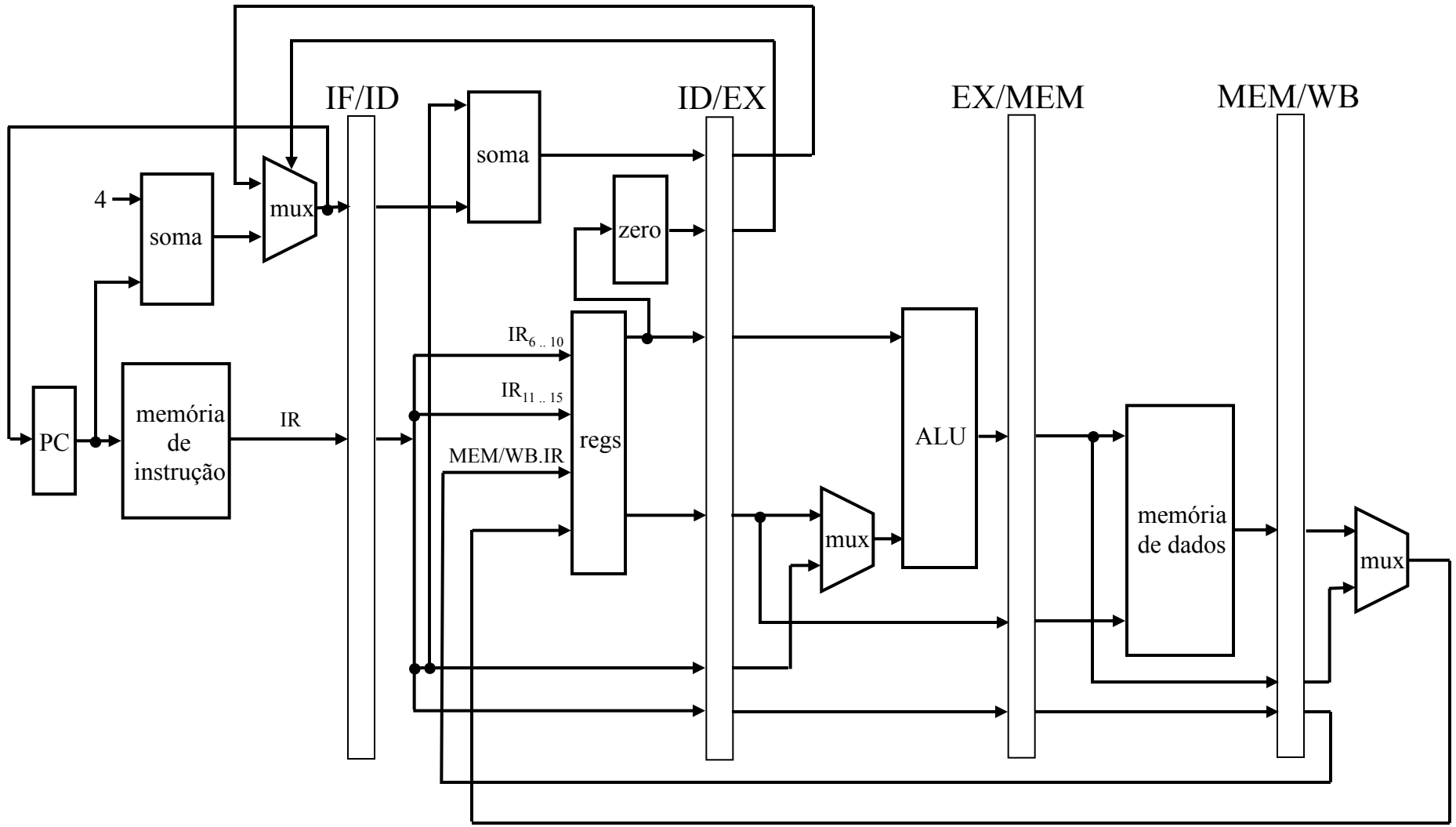
O número de ciclos de *clock,* numa parada por desvio, pode ser reduzido através de duas ações:

1 – identificar mais cedo se o desvio deve ser tomado ou não;

2 – computar mais cedo o endereço alvo de desvio.

Na arquitetura do DLX, é possível completar o teste da condição de desvio ao final do estágio ID.

Para tirar vantagem do teste da condição nesse estágio, os valores possíveis do PC já devem estar computados.


IF/ID
ID/EX
EX/MEM
MEM/WB
4
soma
mux
soma
zero
PC
memória
de
instrução
IR
IR6 .. 10
IR11 .. 15
MEM/WB.IR
regs
ALU
mux
memória
de dados
mux

Uma vez que o desvio é feito ao final do estágio ID, os estágio EX, MEM e WB não são utilizados durante um desvio.

Estágio	Instrução de desvio
IF	IF/ID.IR←mem[PC]; IF/ID.NPC,PC←(se EX/MEM.cond então (EX/MEM.NPC) senão (PC+4);
ID	ID/EX.A←regs[IF/ID.$IR_{6 .. 10}$]; ID/EX.B←regs[IF/ID.$IR_{11 .. 15}$]; ID/EX.NPC←IF/ID.NPC + $IR_{16 .. 31}$; ID/EX.IR←IF/ID.IR; ID/EX.cond←(regs[IF/ID.$IR_{6 .. 10}$] op 0]; ID/EX.Imm←$IR_{16 .. 31}$;
EX	
MEM	
WB	