Pipeline

O *throughput* de um *pipeline* é determinado pela freqüência com que uma instrução sai do *pipeline*.

Todos os estágios devem estar prontos ao mesmo tempo para prosseguir.

O tempo requerido para mover uma instrução de um estágio a outro é denominado ciclo de máquina. Desta forma, o tempo de um ciclo de máquina corresponde ao tempo requerido pelo estágio mais demorado.

Se os estágios estiverem perfeitamente balanceados, o tempo gasto por instrução no pipeline é definido por:

$$tempoporinstruçãonopipeline = \frac{tempoporinstruçãosempipeline}{númerodeestágiosnopipeline}$$

Pipeline

Dessa forma, o *speedup* com *pipeline* corresponde ao número de estágios. No entanto, o *pipeline* não é perfeitamente balanceado e ainda envolve mais esforços.

Pipeline leva a uma redução no tempo médio de execução por instrução:

- diminuição do número de ciclos de *clock* por instrução (*cci*);
- diminuição do ciclo de *clock* (*cc*);
- combinação dos fatores acima.

Pipeline aumenta o *throughput* de instruções (número de instruções executadas por unidade de tempo), mas não reduz o tempo de execução de uma instrução.

Pipeline

O tempo de ciclo τ de um *pipeline* de instrução é o tempo requerido para avançar um conjunto de instruções de 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 $\tau_m >> d$.

Pipeline

Suponha que sejam processadas *n* instruções, sem que ocorra desvio. O tempo total de execução é dado por:

$$T_k = [k + (n-1)] \times \tau$$

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, o fator de aceleração é igual a *k* quando $n \rightarrow \infty$.

Pipeline

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

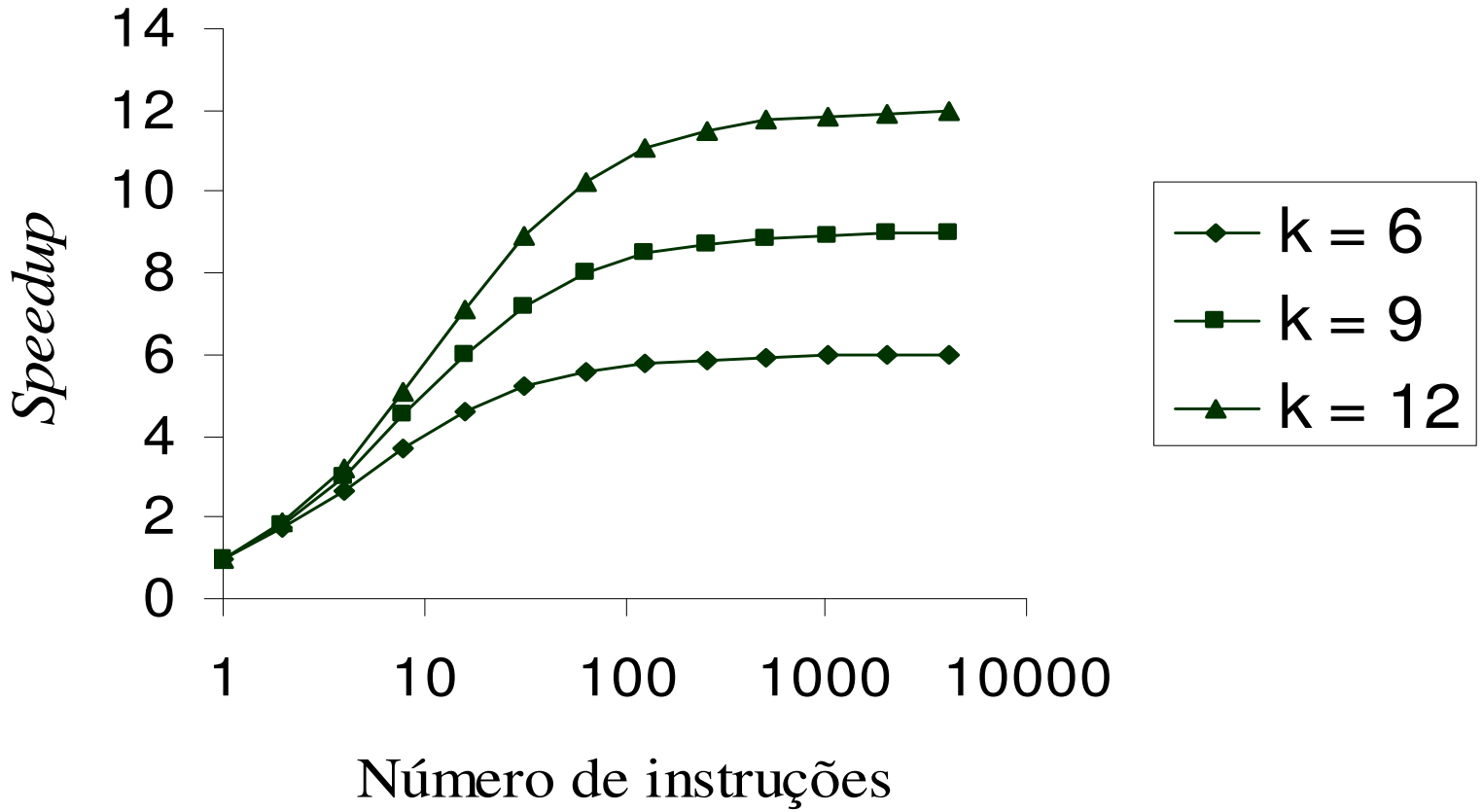
Quanto maior o número de estágios do *pipeline*, maior o *speedup*. No entanto, o ganho diminui devido:

- 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.

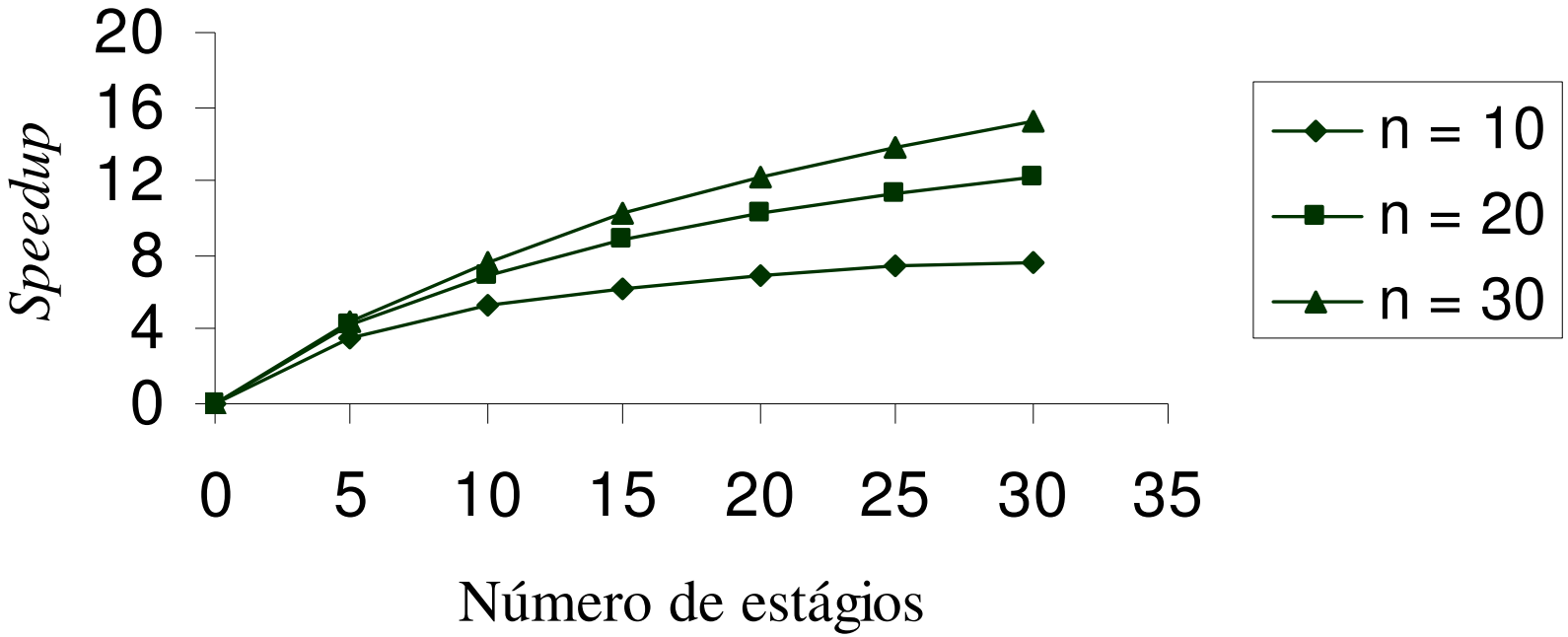
Pipeline

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



Pipeline

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



Pipeline

Múltiplos fluxos consiste em duplicar os estágios iniciais do *pipeline* para permitir a busca de ambas as instruções, usando dois fluxos de instruções.

Problemas:

- o uso de múltiplos *pipelines* introduz atrasos devidos à contenção de acesso a registradores e à memória;
- pode ocorrer a entrada de instruções de desvio adicionais no *pipeline*, antes que seja tomada a decisão sobre o desvio original.

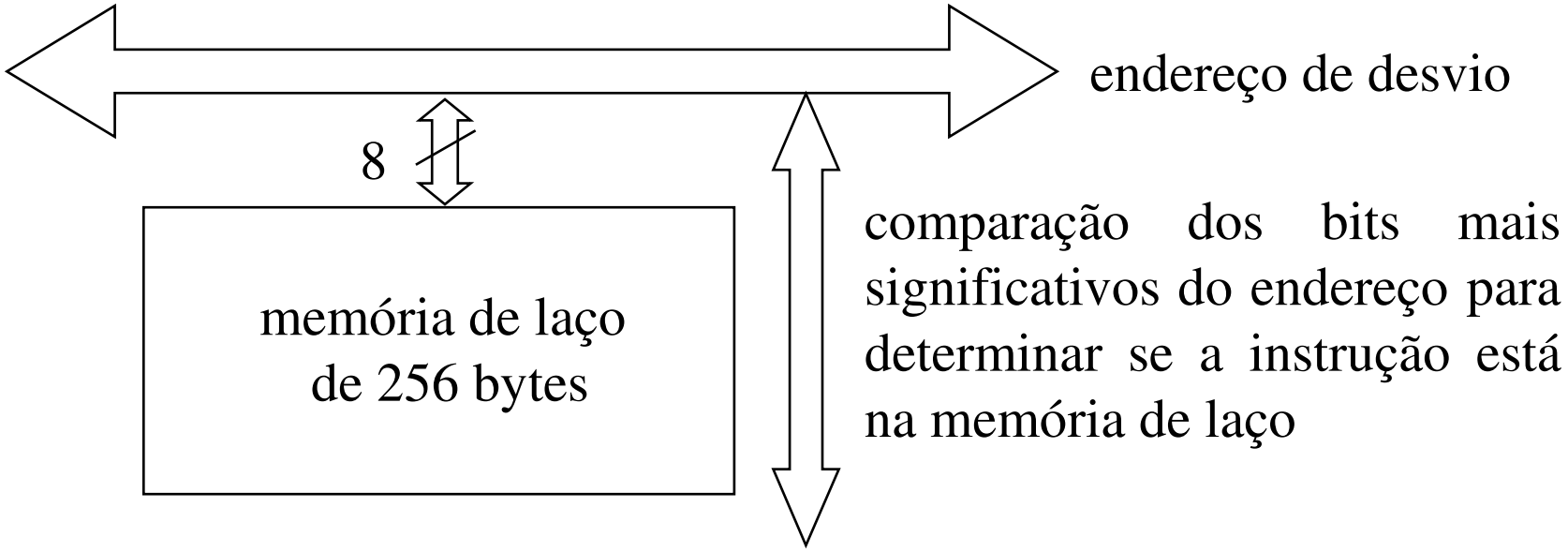
Pipeline

Busca antecipada da instrução-alvo do desvio consiste em buscar, antecipadamente, tanto a instrução-alvo do desvio quanto a instrução consecutiva ao desvio, no instante em que a instrução de desvio condicional é reconhecida. A instrução-alvo é armazenada em um registrador até que a instrução de desvio seja executada.

Memória de laço consiste em usar uma pequena memória de alta velocidade (memória de laço de repetição ou *loop buffer*), mantida pelo estágio de busca de instrução, para guardar as *n* instruções buscadas mais recentemente, em seqüência.

Pipeline

Considere uma memória de laço com 256 bytes e endereçamento de byte.



Pipeline

Vantagens:

- com o uso de busca antecipada, a memória de laço conterá certo número de instruções que estão à frente da instrução corrente;
- se ocorrer um desvio para alguma posição adiante do endereço da instrução de desvio, essa posição já estará na memória de laço (útil em instruções do tipo IF-THEN-ELSE);
- particularmente adequada para lidar com laços de repetição ou iterações (se a memória for grande o suficiente para conter as instruções de uma iteração, estas terão que ser buscadas da memória apenas uma vez, para a primeira iteração).

Pipeline

Previsão de desvio pode ser feita de várias formas:

- prever que o desvio nunca será tomado: abordagem simples e estática, isto é, não depende do histórico das instruções até o momento em que ocorre a instrução de desvio condicional; continua buscando instruções na seqüência em que ocorrem no programa.
- prever que o desvio sempre será tomado: abordagem simples e estática, isto é, não depende do histórico das instruções até o momento em que ocorre a instrução de desvio condicional; busca sempre as próximas instruções a partir do endereço-alvo do desvio.

Pipeline

- prever se o desvio será tomado ou não conforme o código de operação: abordagem simples e estática.
- prever o desvio com base em chaves de desvio tomado e de desvio não tomado: abordagem dinâmica, isto é, depende do histórico de execução.
- prever o desvio com base em uma tabela de histórico de desvios: abordagem dinâmica.

Se a busca da instrução consecutiva à instrução de desvio causar uma falta de página ou uma violação de proteção, o processador interromperá a busca antecipada da instrução até que tenha certeza de que essa instrução deve ser mesmo buscada.

Pipeline

Análises de comportamento de programas mostram que desvios condicionais são tomados em mais de 50% das vezes.

Se o custo da busca antecipada de instruções for o mesmo em qualquer caminho, o resultado obtido deverá ser melhor se a busca antecipada de instruções for sempre efetuada a partir do endereço-alvo do desvio.

Entretanto, em uma máquina que usa paginação, a busca antecipada de instruções, a partir do endereço de desvio, tem maior probabilidade de causar uma falta de página do que a busca de instruções consecutivas à instrução de desvio.

Pipeline

A previsão de desvio com base no código de operação da instrução de desvio pressupõe que para determinados códigos o desvio é sempre tomado e para outros não, havendo um aproveitamento de 75%.

Estratégias dinâmicas de previsão de desvio mantêm um histórico sobre as instruções de desvio condicional, i.e. um ou mais bits (chaves de desvio tomado ou de desvio não tomado) são associados a cada instrução de desvio condicional.

Utilizando-se somente um bit de histórico, pode-se registrar se a última execução da instrução resultou em desvio ou não.

Pipeline

Uma desvantagem neste caso ocorre quando o desvio é quase sempre tomado, tal como em instruções de desvio usadas para implementar laços de repetição. Sempre ocorrerão dois erros de previsão de desvio, cada vez que o laço de repetição for executado: uma vez na entrada e outra na saída.

Atraso de desvio consiste em reordenar as instruções, de modo que as instruções de desvio ocorram mais tarde.

Pipeline

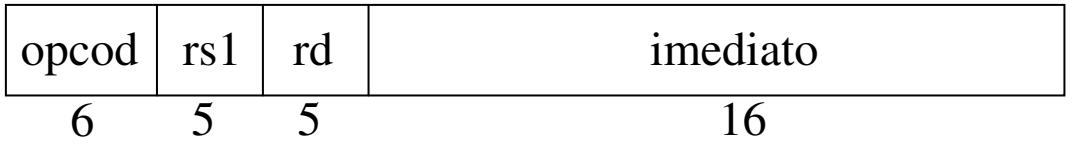
Considere a arquitetura do processador DLX, sem pipeline:

- 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.

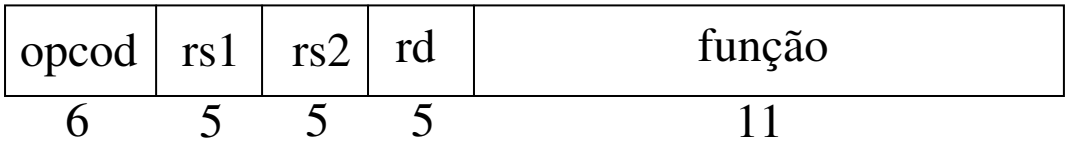
Pipeline

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):

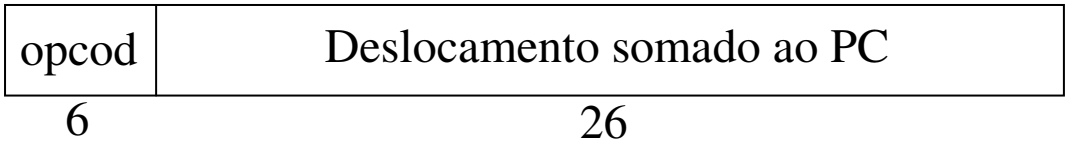
- instrução do tipo I:



- 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.

Pipeline

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

1 – ciclo de busca de instrução (IF):

$IR \leftarrow mem[PC];$
$NPC \leftarrow PC + 4$

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

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

Pipeline

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., ==)

Pipeline

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

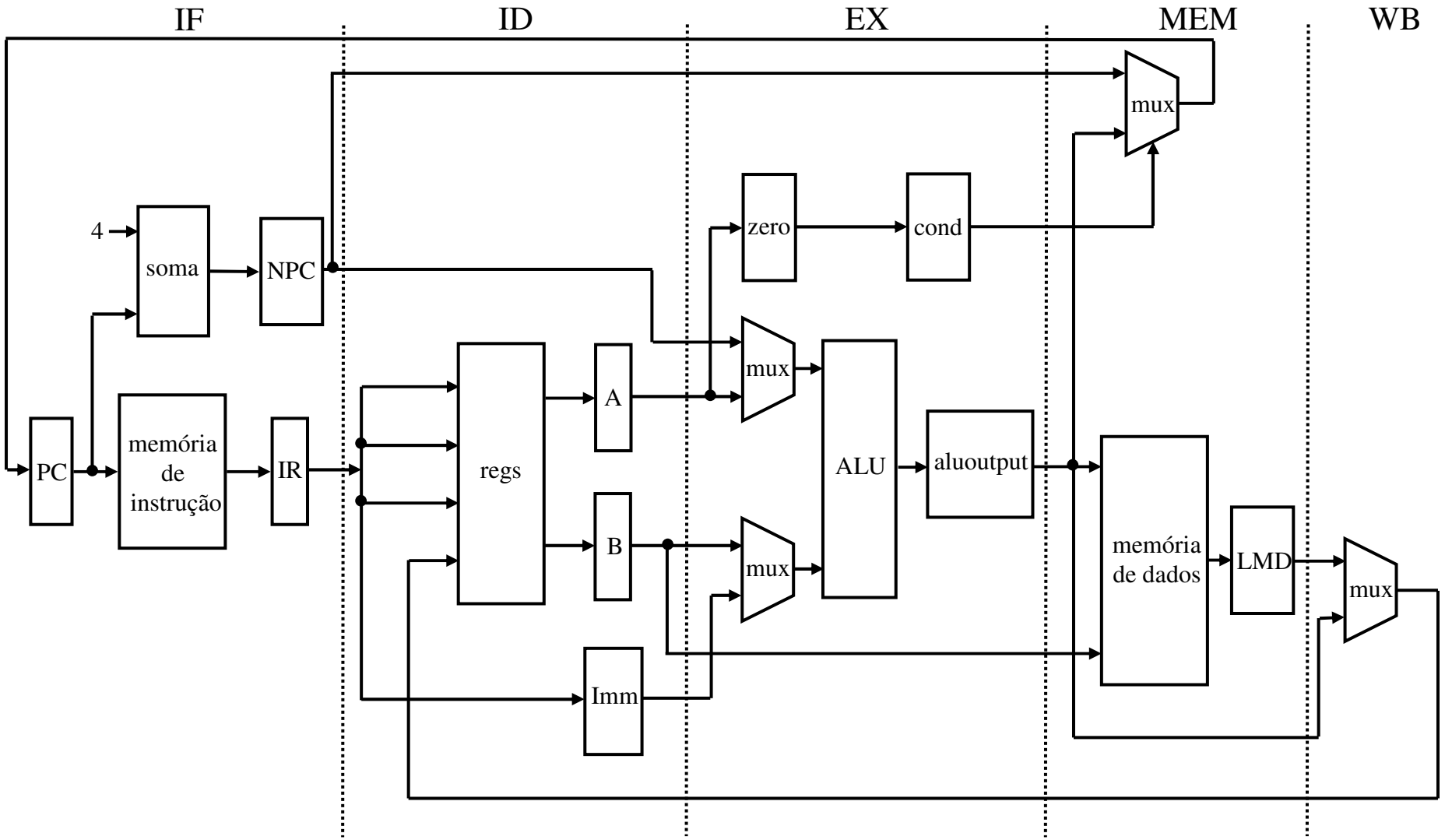
5 – ciclo de escrita (WB):

regs[$IR_{16 .. 20}$] ← ALUoutput;

regs[$IR_{11 .. 15}$] ← ALUoutput;

regs[$IR_{11 .. 15}$] ← LMD

Pipeline


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

Pipeline

Ao término de cada ciclo de *clock*, cada valor computado durante aquele ciclo e requerido num ciclo mais tarde (quer seja para esta 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.

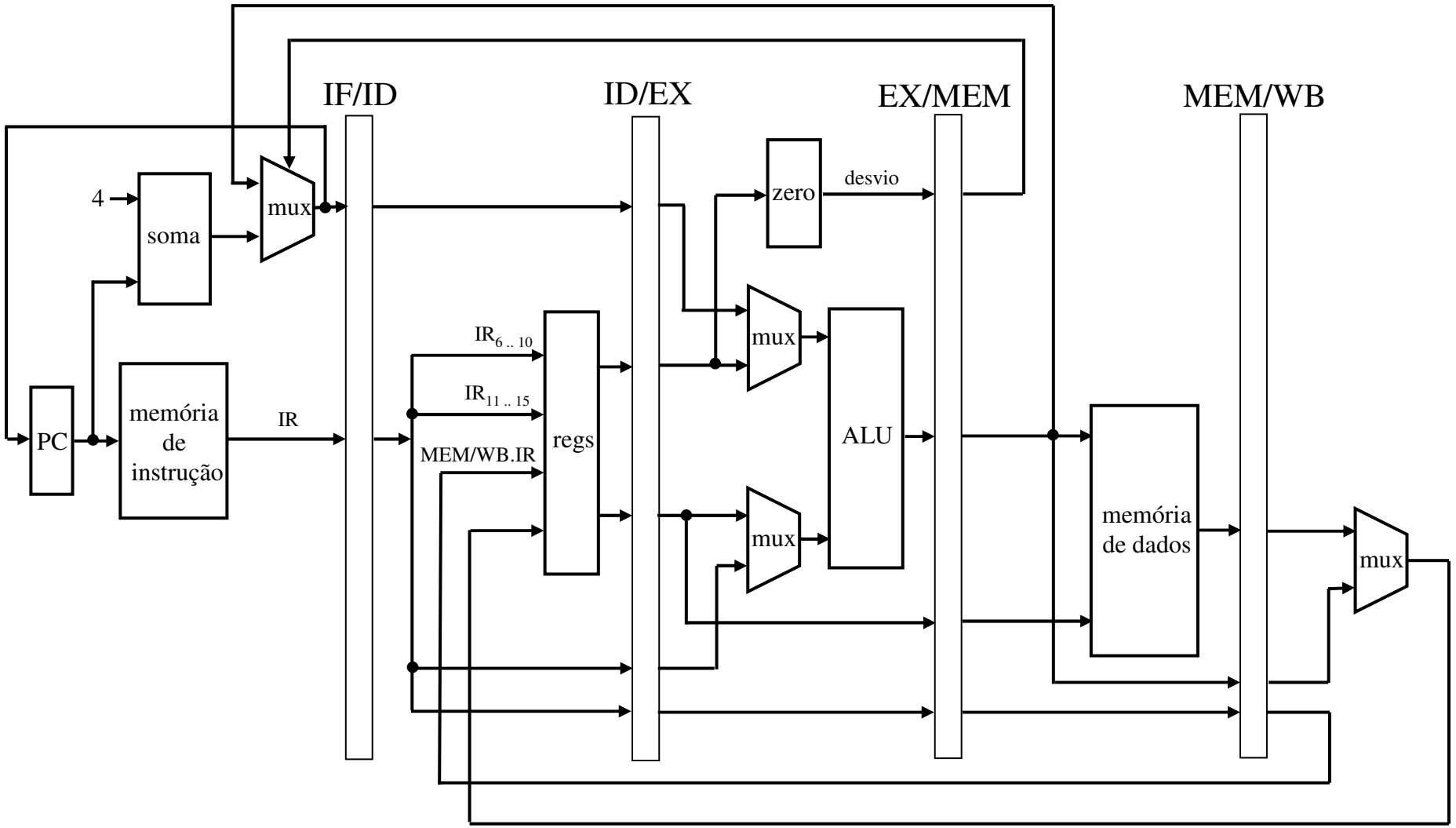
Nesta arquitetura, instruções de desvio requerem quatro ciclos de *clock* e todas as outras requerem cinco ciclos de clock.

Pipeline

Pode-se implementar pipeline nesta 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

Pipeline


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

Pipeline

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.

Pipeline

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;	

Pipeline

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.

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

Pipeline

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

Pipeline

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

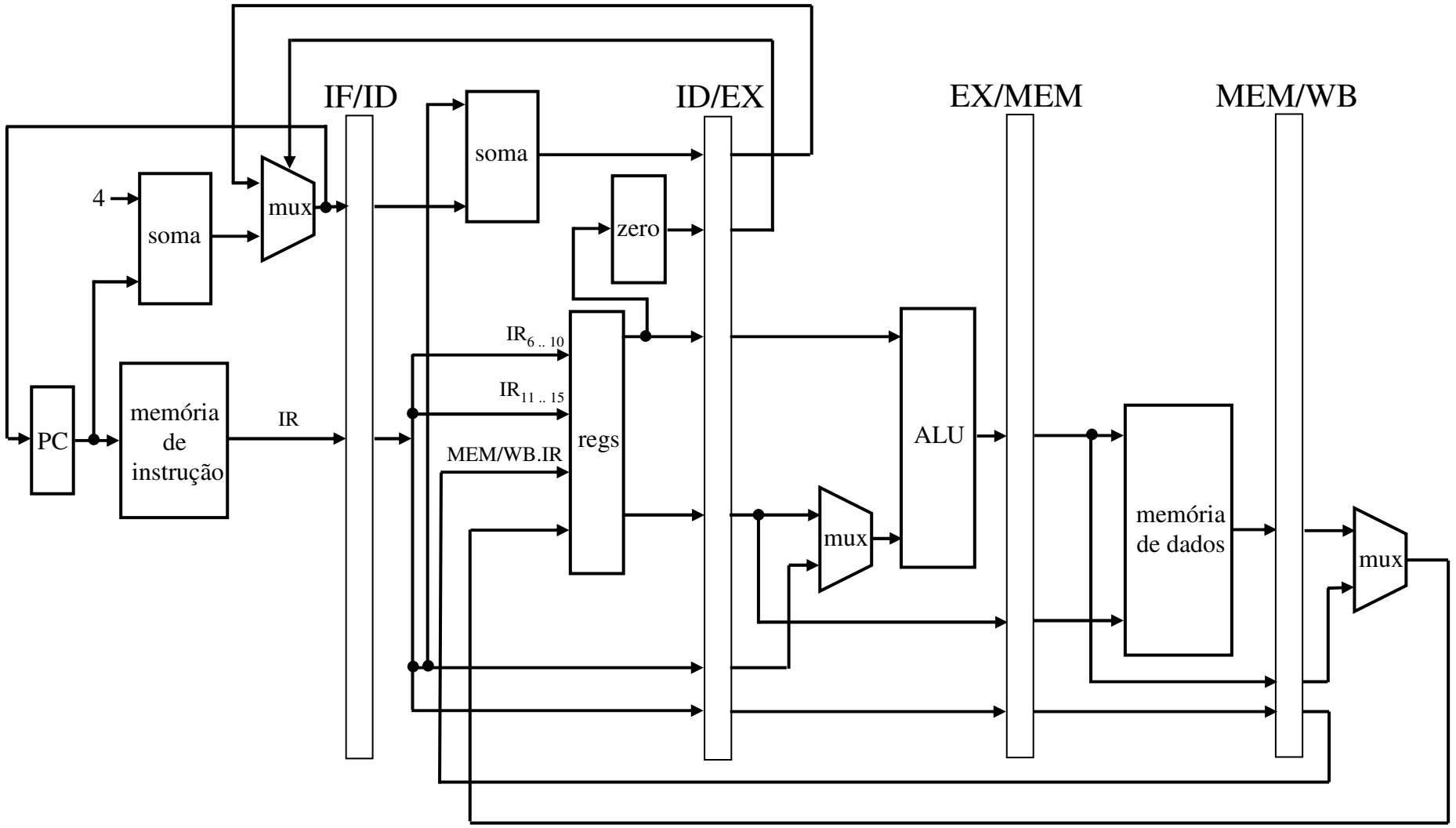
1 – identificar mais cedo se o desvio deve ser tomado ou não;

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.

Pipeline


IF/ID
ID/EX
EX/MEM
MEM/WB
soma
4
soma
mux
zero
PC
memória
de
instrução
IR
$IR_{6 .. 10}$
$IR_{11 .. 15}$
MEM/WB.IR
regs
ALU
mux
memória
de dados
mux

Pipeline

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	