**Universidade Federal do Rio Grande do Sul**
**Instituto de Informática**

# Organização de Computadores

# Aula 10

# Pipelines
## Parte 1

# Pipelines
## Primeira parte

1. Introdução

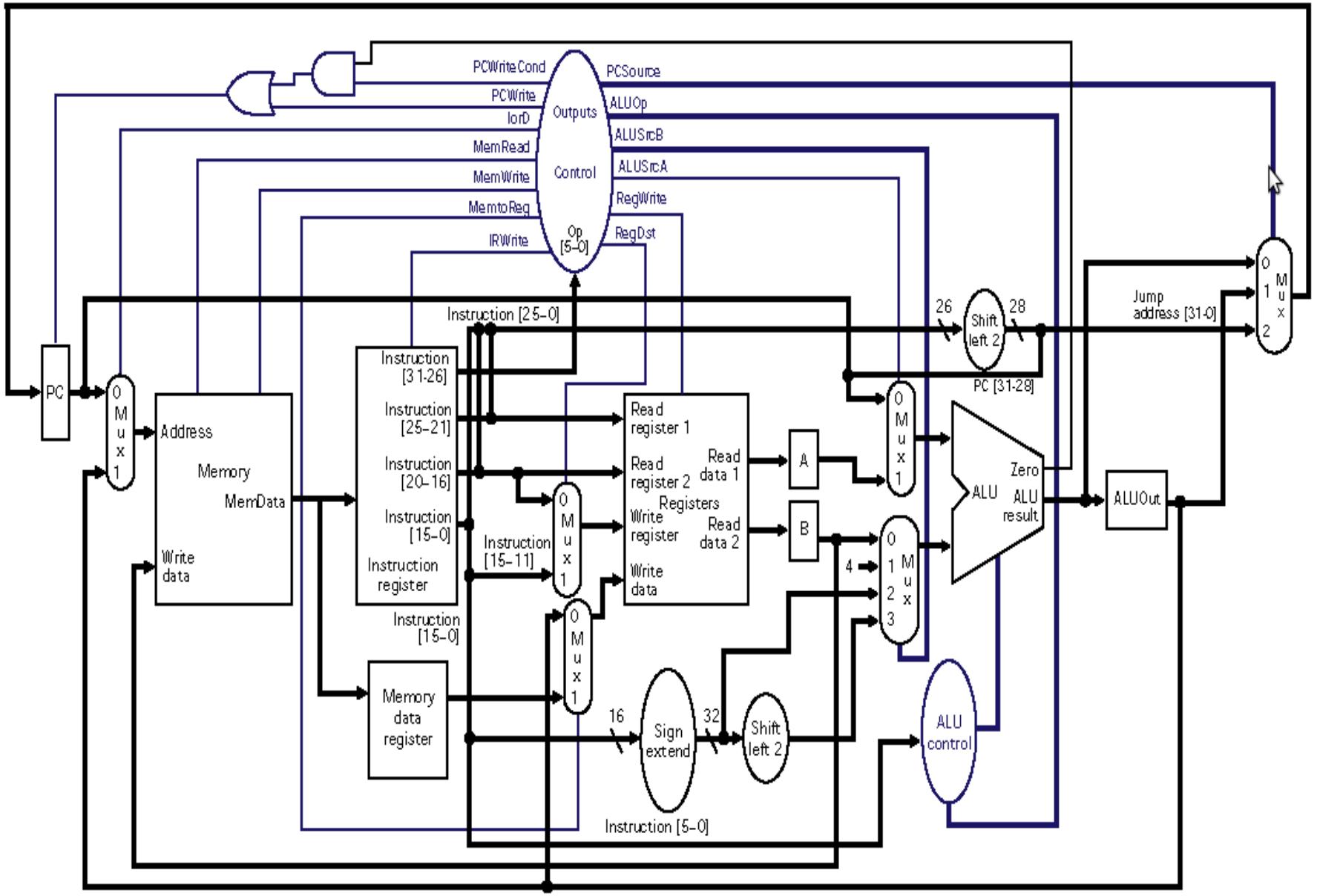
2. Pipelines aritméticos

3. Pipelines de instruções

4. Desempenho

5. Conflitos de memória

6. Dependências em desvios


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op [5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
PC
0 M u x 1
Address
Memory
MemData
Write data
Instruction [31-26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–0]
Instruction register
Instruction [15–0]
Instruction [15–11]
Instruction [25–0]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
4
ALU
Zero
ALU result
ALUOut
26
Shift left 2
28
PC [31-28]
Jump address [31-0]
0 1 2 M u x
0 1 2 3 M u x
Memory data register
16
Sign extend
32
Shift left 2
ALU control
Instruction [5–0]

# Introdução

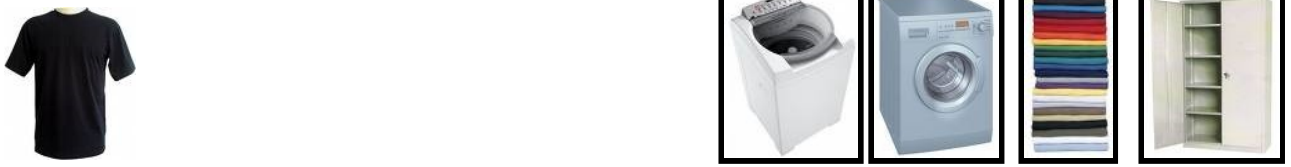
- **Estudo de Caso: Como ensinar a nova empregada a lavar 3 tipos de roupas diferentes ?**

- **Os tipos de roupa são:**
  - **Brancas**
  - **Pretas**
  - **Coloridas**

- **Vamos dividir a lavagem em etapas de 1 hora cada:**
  - **Lavar**
  - **Secar**
  - **Dobrar**
  - **Guardar**

# Introdução

Empregada multiciclo



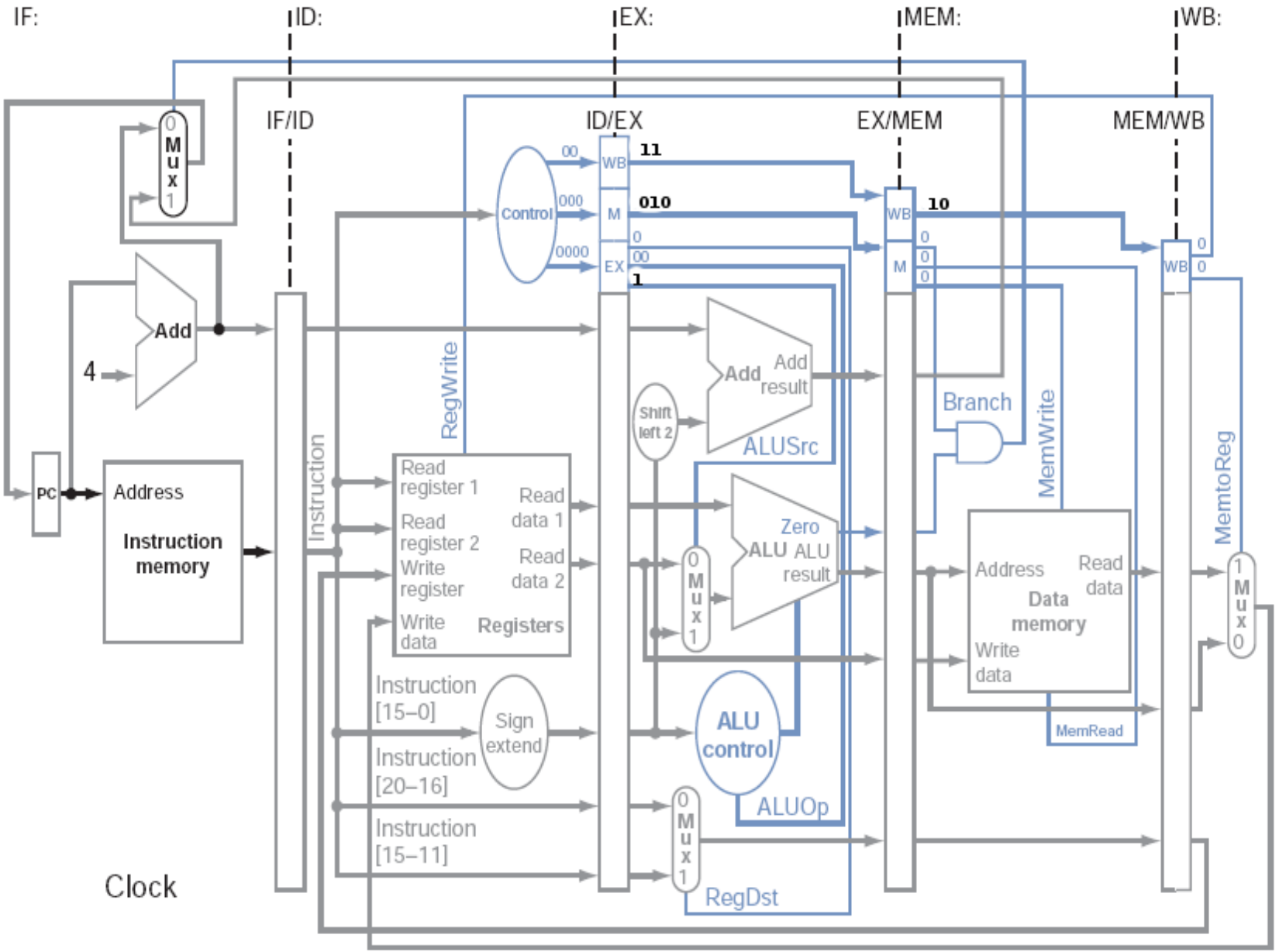
0h 12h

Tempo

# Introdução


Agora com empregada Pipelined de 4 estágios

6


IF:
ID:
EX:
MEM:
WB:
IF/ID
ID/EX
EX/MEM
MEM/WB
M u x
0
1
Add
4
PC
Address
Instruction memory
Instruction
RegWrite
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Control
00
000
0000
WB
M
EX
11
010
1
10
Shift left 2
Add
Add result
ALUSrc
Zero
ALU
ALU result
Branch
MemWrite
MemtoReg
Address
Data memory
Read data
Write data
MemRead
Instruction [15–0]
Instruction [20–16]
Instruction [15–11]
Sign extend
ALU control
ALUOp
RegDst
Clock

# Introdução

- **Lembrando que nossas instruções são executadas no nosso processador em 5 passos:**
  - **Busca da instrução na memória,**
  - **Leitura do registradores e decodificação,**
  - **Execução de uma operação ou cálculo de endereço,**
  - **Acesso a um operando na memória,**
  - **Escrita do resultado em um registrador.**

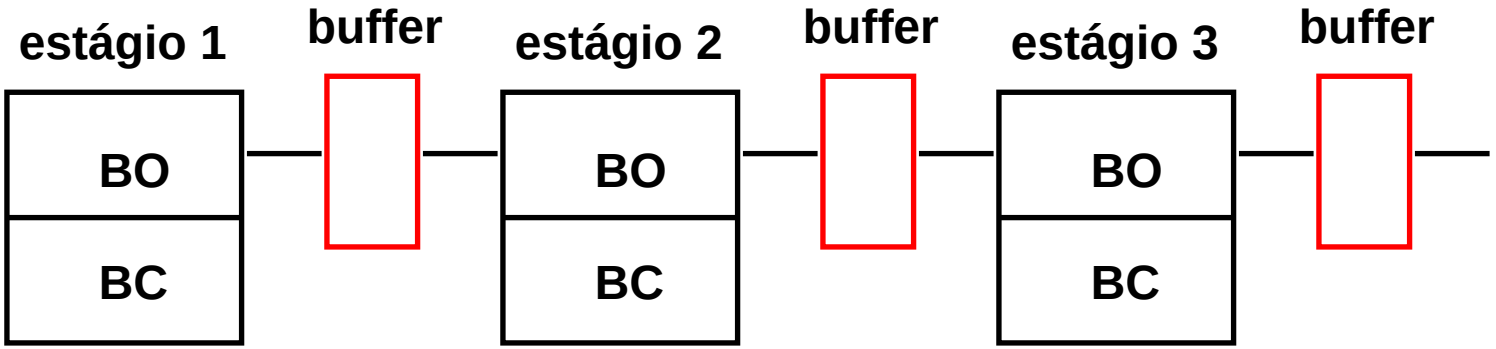
# Introdução Pipeline

- **Repare que o tempo para as ações de mesmo tipo são iguais para as diversas instruções**

| Tipo Instrução | Busca Instr | Ler Reg Dec | Oper UAL | Acesso Dado | Escrita Reg | TEMPO TOTAL |
|---|---|---|---|---|---|---|
| Load | 2ns | 1ns | 2ns | 2ns | 1ns | 8ns |
| Store | 2ns | 1ns | 2ns | 2ns | - | 7ns |
| R-type | 2ns | 1ns | 2ns | - | 1ns | 6ns |
| Branch | 2ns | 1ns | 2ns | - | - | 5ns |

# Introdução

- **Bloco operacional e bloco de controle independentes para cada estágio**
- **Necessidade de buffers entre os estágios**
- **Estágios devem sofre o atraso extra dos buffers**



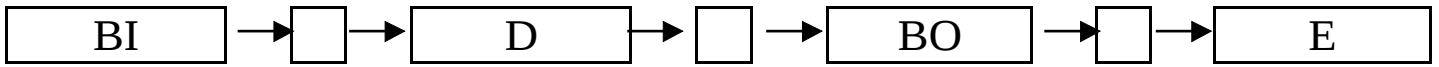

- **Temos 2 tipos principais de pipeline em processadores:**
  - **Instruções**
  - **Operações Aritméticas**

# PIPELINE DE INSTRUÇÕES

# 3. Pipeline de Instruções

- **Hoje todos microprocessadores possuem pipeline de instrução.**

- **O pipeline básico é composto pelos seguintes estágios:**
  **Busca de instrução**
  **Decodificação**
  **Busca de operando**
  **Execução**



# Pipelines de Instruções

- **Evolução no número de estágios:**
  - **2 Estágios**
    - **Fetch + decodificação | execução**
  - **3 Estágios**
    - **Fetch | decodificação + busca de operandos | execução**
  - **4 Estágios**
    - **Fetch | decodificação + busca de operandos | execução | store**
  - **5 Estágios**
    - **Fetch | decodificação + cálculo end. op. | busca de operandos | execução | store**
  - **6 Estágios**
    - **Fetch | decodificação | cálculo end. op. | busca de operandos | execução | store**
- **Estágio só de decodificação é usual em processadores CISC**

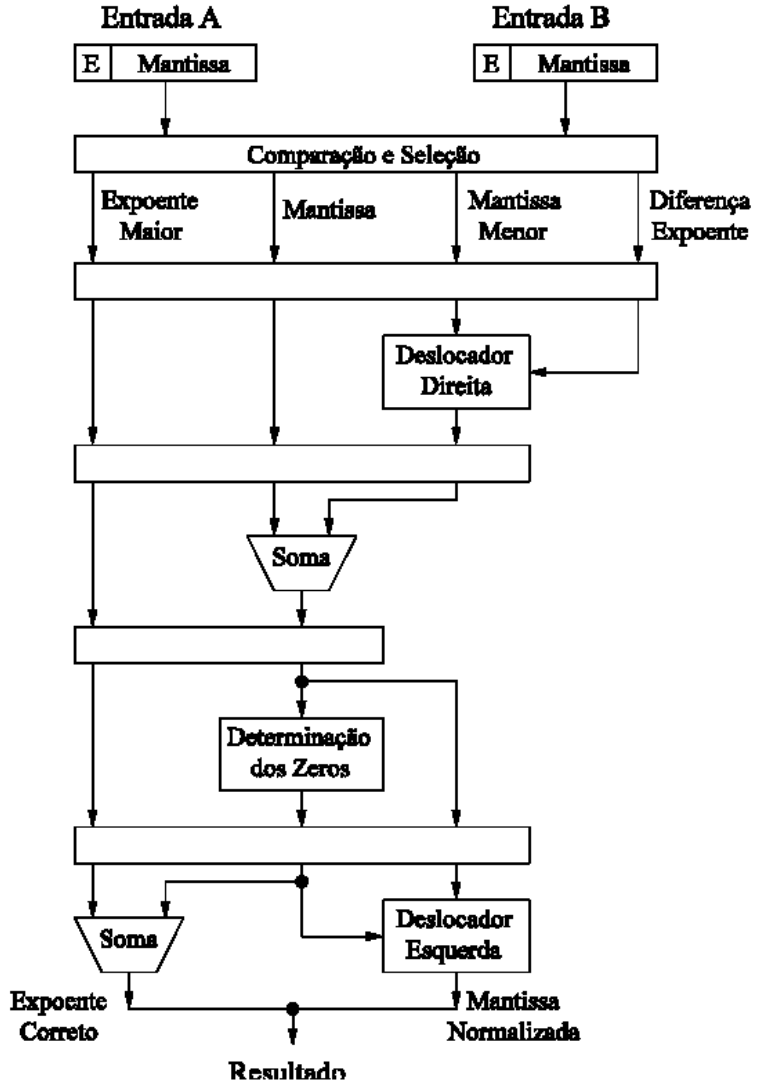
# PIPELINE ARITMÉTICO

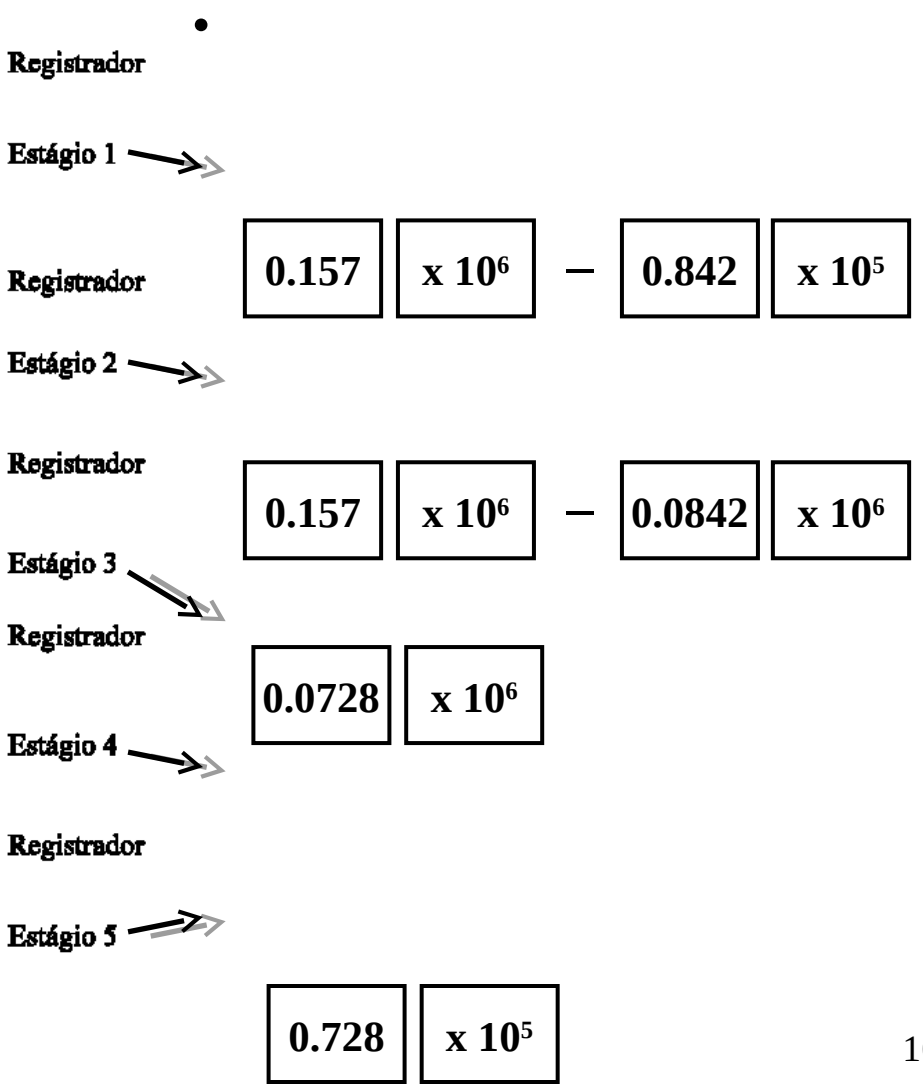
# 2. Pipeline Aritmético

- **O pipeline aritmético é empregado para acelerar as funções lógico e aritméticas das ULAs.**
- **É a divisão das operações aritméticas em suboperações.**
- **Foi a base para o surgimento dos Supercomputadores.**
- **Todos microprocessadores modernos possuem pipeline aritmético.**

- **Próximo slide apresenta um Somador de Ponto Flutuante com 5 estágios**
- **Os cinco estágios:**
  - **comparação dos operandos A e B,**
  - **ajuste da mantissa,**
  - **soma dos operandos A e B**
  - **verificação dos zeros da soma**
  - **ajuste do expoente final**
- **Resulta no final um expoente e uma mantissa**

# Pipeline Aritmético

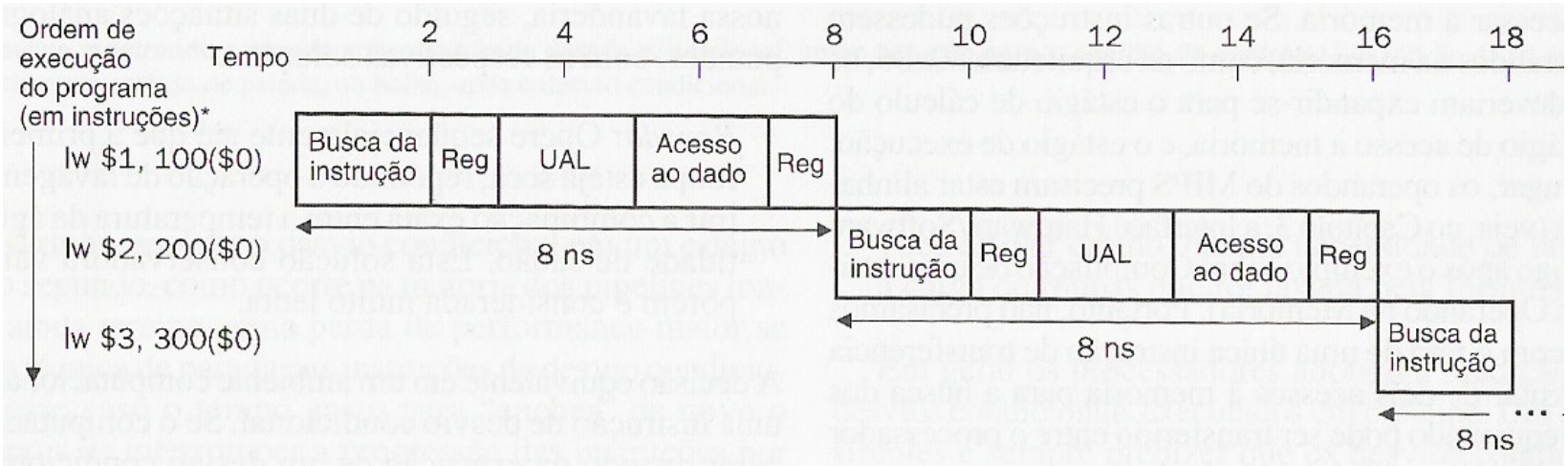


# PIPELINE NO MIPS

# Pipeline da nossa Arquitetura MIPS

- **Quantos ciclos são necessários para executar as 3 instruções ?**



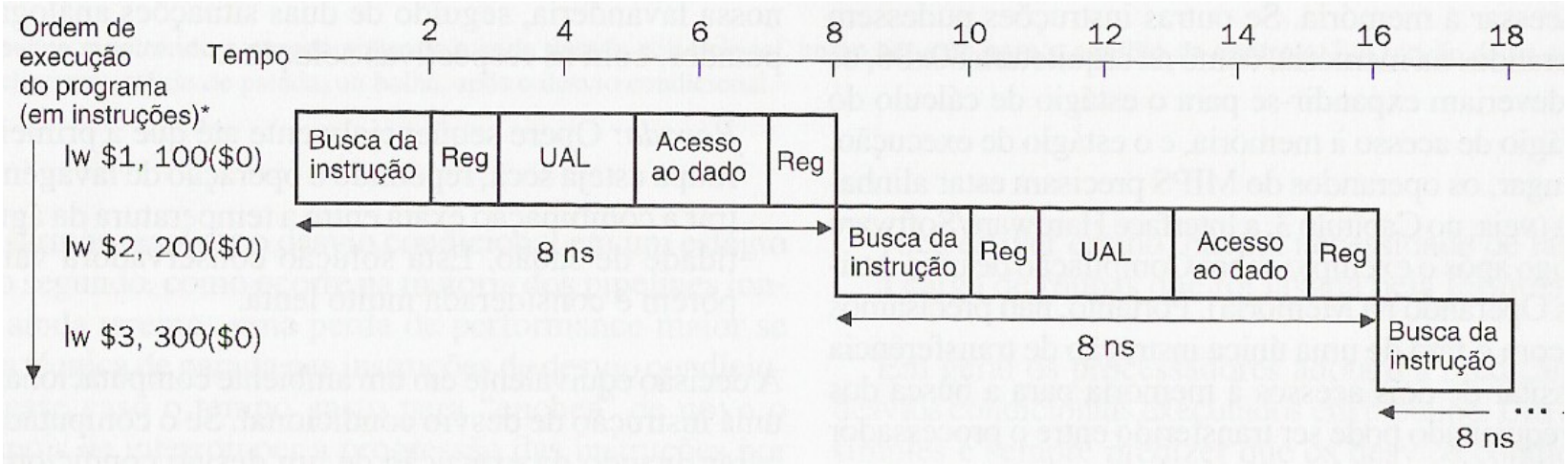
# Pipeline da nossa Arquitetura MIPS

- **Lembrando os tempos da nossa arquitetura tradicional.**

| Tipo Instrução | Busca Instr | Ler Reg Dec | Oper UAL | Acesso Dado | Escrita Reg | TEMPO TOTAL |
|---|---|---|---|---|---|---|
| Load | 2ns | 1ns | 2ns | 2ns | 1ns | 8ns |
| Store | 2ns | 1ns | 2ns | 2ns | | 7ns |
| R | 2ns | 1ns | 2ns | | 1ns | 6ns |
| Branch | 2ns | 1ns | 2ns | | | 5ns |

# Pipeline da nossa Arquitetura MIPS

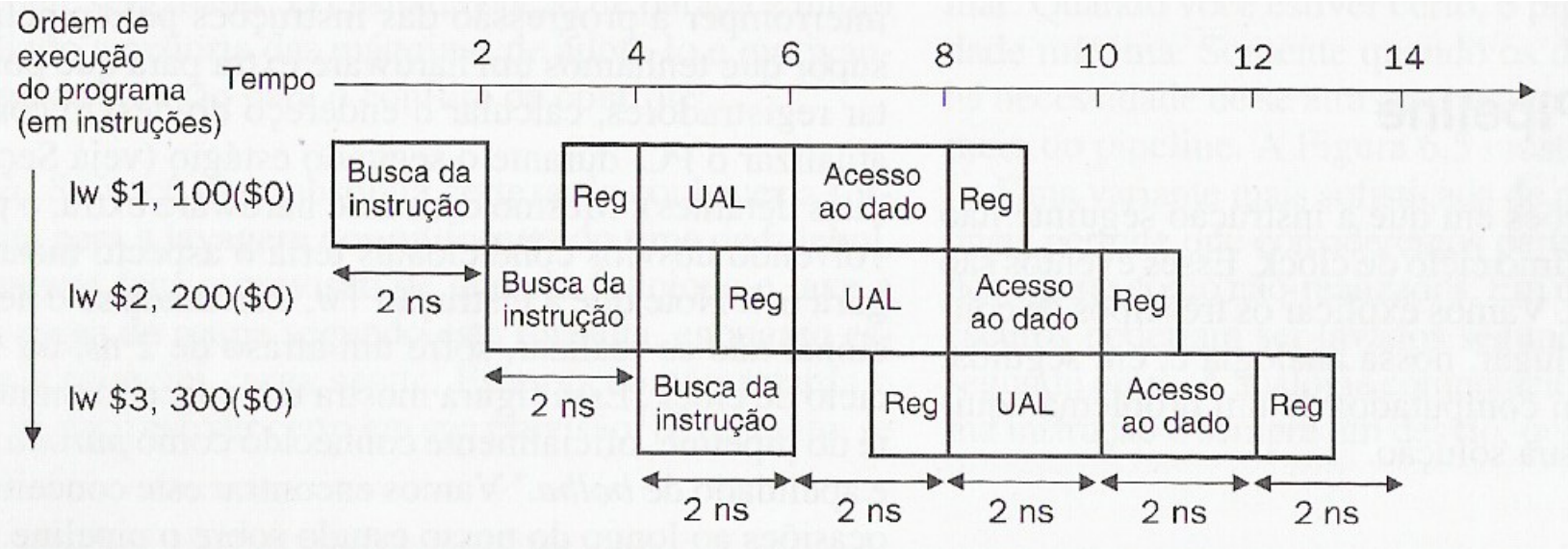
- **Para executar as 3 instruções, sem uso de Pipeline**
  - **MIPS multi-ciclos, 3 instruções lw tempo igual a 24ns**




  - **Quanto tempo levaria no caso MIPS pipelined ?**

# Pipeline da nossa Arquitetura MIPS

- **Com Pipeline**
  - **Clock deve ser da operação mais lenta, no caso 2ns**
  - **Total após 3 instruções  14ns**




- **Em condições ótimas o ganho de um pipeline é igual ao número de estágios deste.**

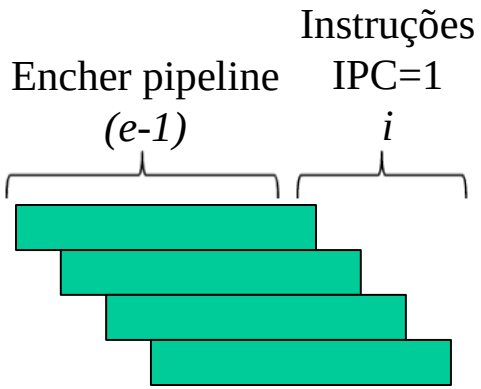
# DESEMPENHO DO PIPELINE

# Desempenho do Pipeline

- **O tempo do relógio é dado pelo tempo do estágio mais lento**

$$T_{ciclo} = \text{Máx}[T_{i\text{-}estágio}] + T_{reg}$$

- **Tempo total sem pipeline = *i* * *e***
  - ***i*** **Instruções**
  - ***e*** **Estágios**




- **Tempo total com pipeline ideal = *i* + *( e – 1 )***
  - **Encher o pipeline de *e* estágios: *e - 1* ciclos de relógio**
  - ***i*** **Instruções executadas: *i* ciclos de relógio**

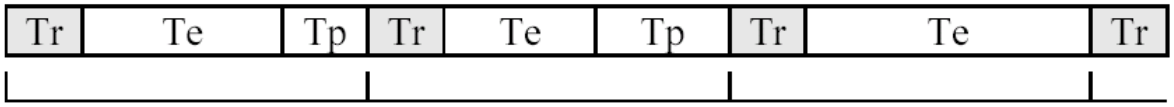
$$\text{Speed–up teórico} = \frac{i\ e}{i + (e - 1)} \qquad \text{limite} = e$$

# PROBLEMAS NO PIPELINE

# Problemas na implementação do Pipeline

- **Como dividir todas as diferentes instruções em um mesmo conjunto de operações dos estágios?**
- **Como obter estágios com tempos de execução similares?**

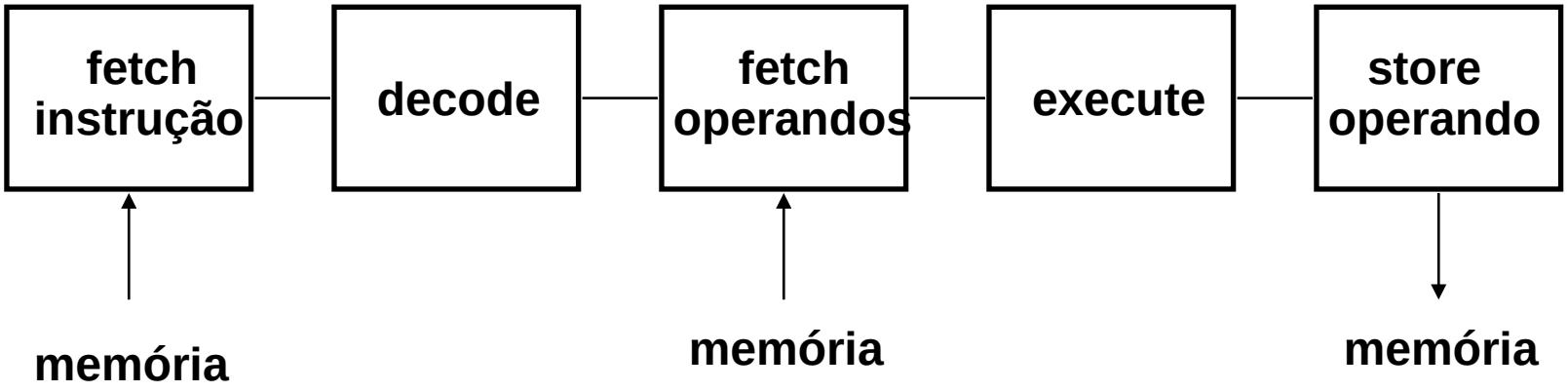
Tr - Tempo do registrador
Te - Tempo de execução
Tp - Tempo de perda




- **Conflitos de memória**
  - **Acessos simultâneos à memória por 2 ou mais estágios**
- **Instruções de desvio**
  - **Instrução seguinte não está no endereço seguinte ao da instrução anterior**
- **Dependências de dados (próxima aula)**
  - **Instruções dependem de resultados de instruções anteriores ou ainda não completadas**

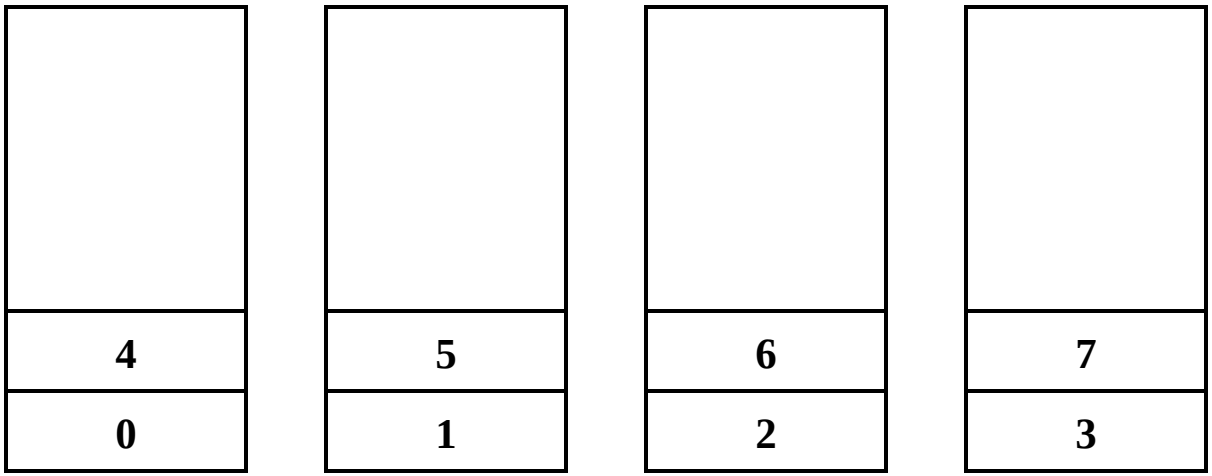
# 5. Conflitos de memória

- **Problema: acessos simultâneos à memória por 2 ou mais estágios**




- **Soluções**
  - **Caches separadas de dados e instruções**
  - **Memória entrelaçada**

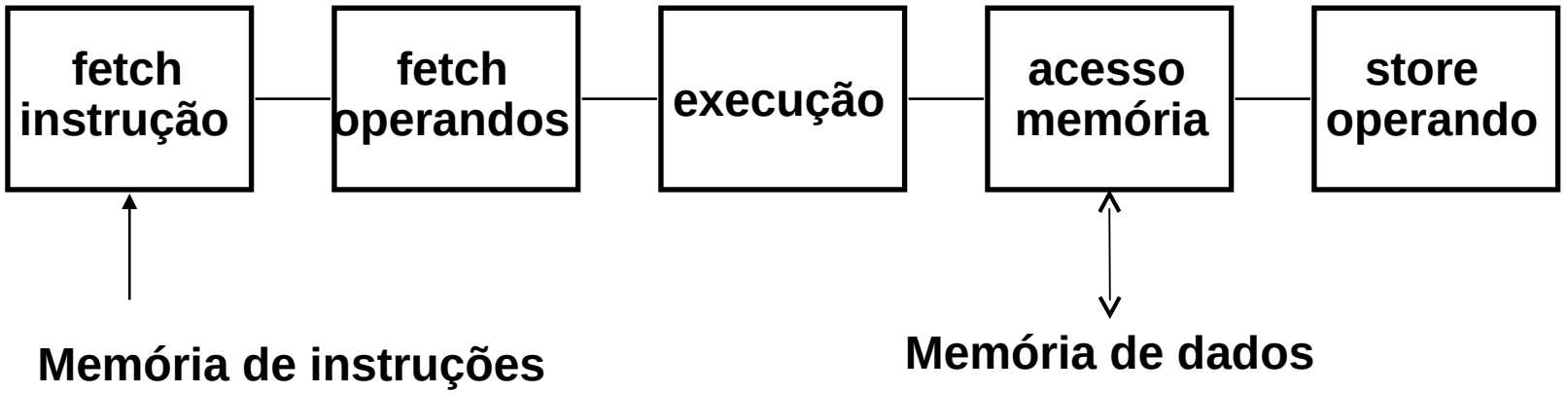
# Memória entrelaçada



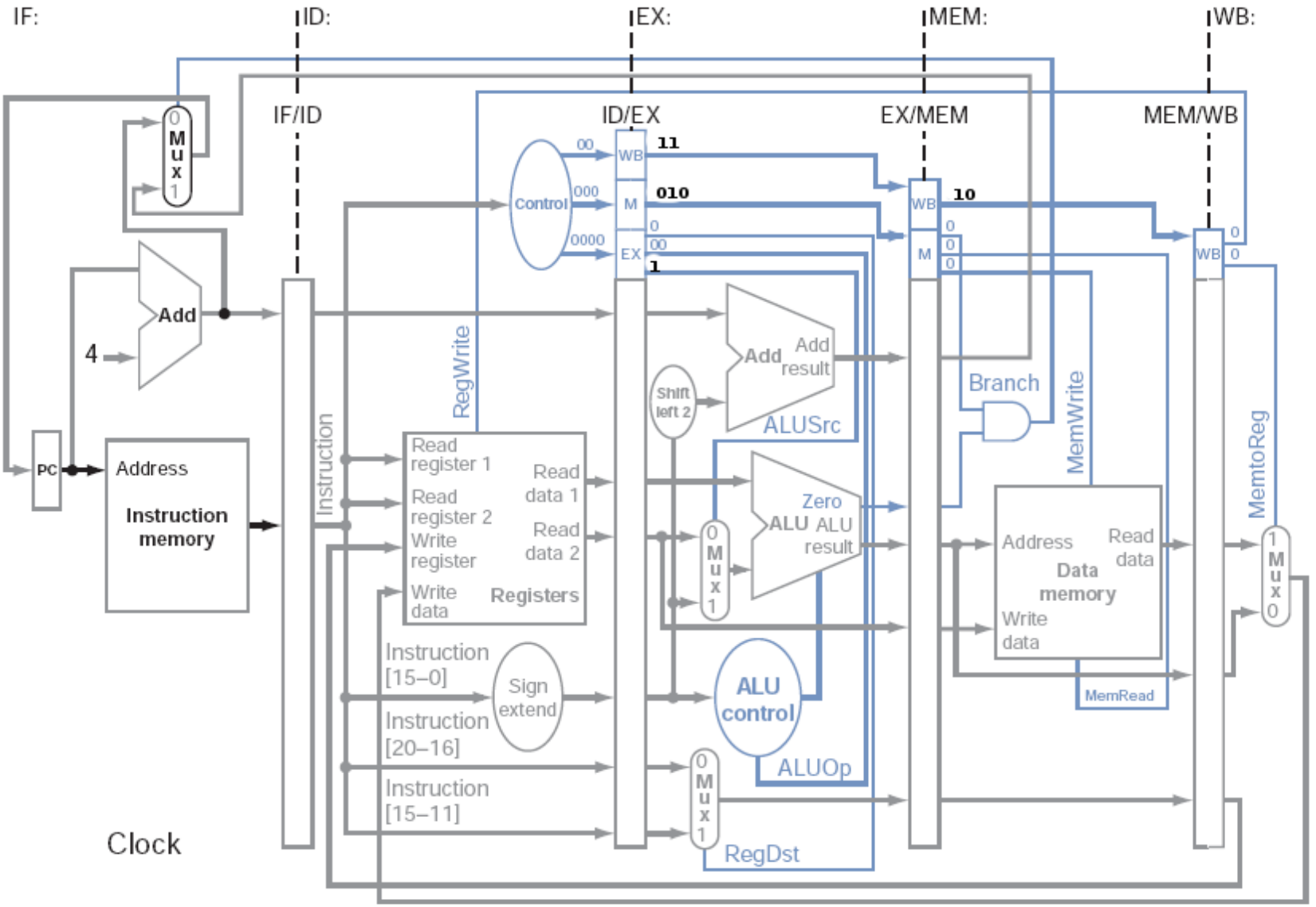

- Bancos de memória separados
- Cada um com seu registrador de dados
- Acessos simultâneos são possíveis a bancos diferentes
- Barramentos entre memória e processador devem operar a velocidade mais alta

# Processadores RISC

- **Memória é acessada apenas por instruções LOAD e STORE**
- **Apenas um estágio do pipeline faz acesso a operandos de memória**
  - **Apenas 1 instrução pode estar executando acesso a dados a cada instante**
- **Se caches de dados e instruções são separadas, não há nenhum conflito de acesso à memória**

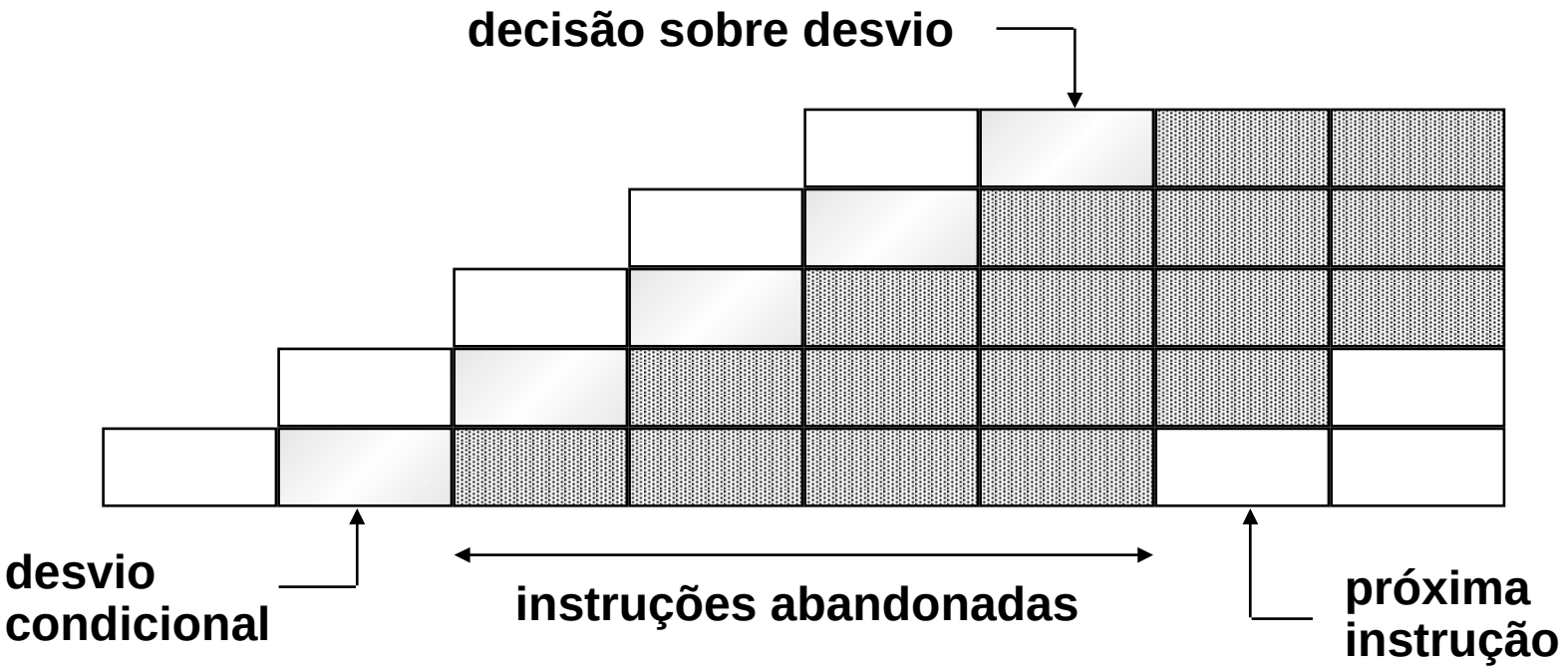


# 6. Dependências em desvios



# 6. Dependências em desvios

- **Efeito de desvios condicionais**
  - **Se ocorre um desvio, o pipeline precisa ser esvaziado**
  - **Não se sabe se desvio ocorrerá ou não até o momento de sua execução**



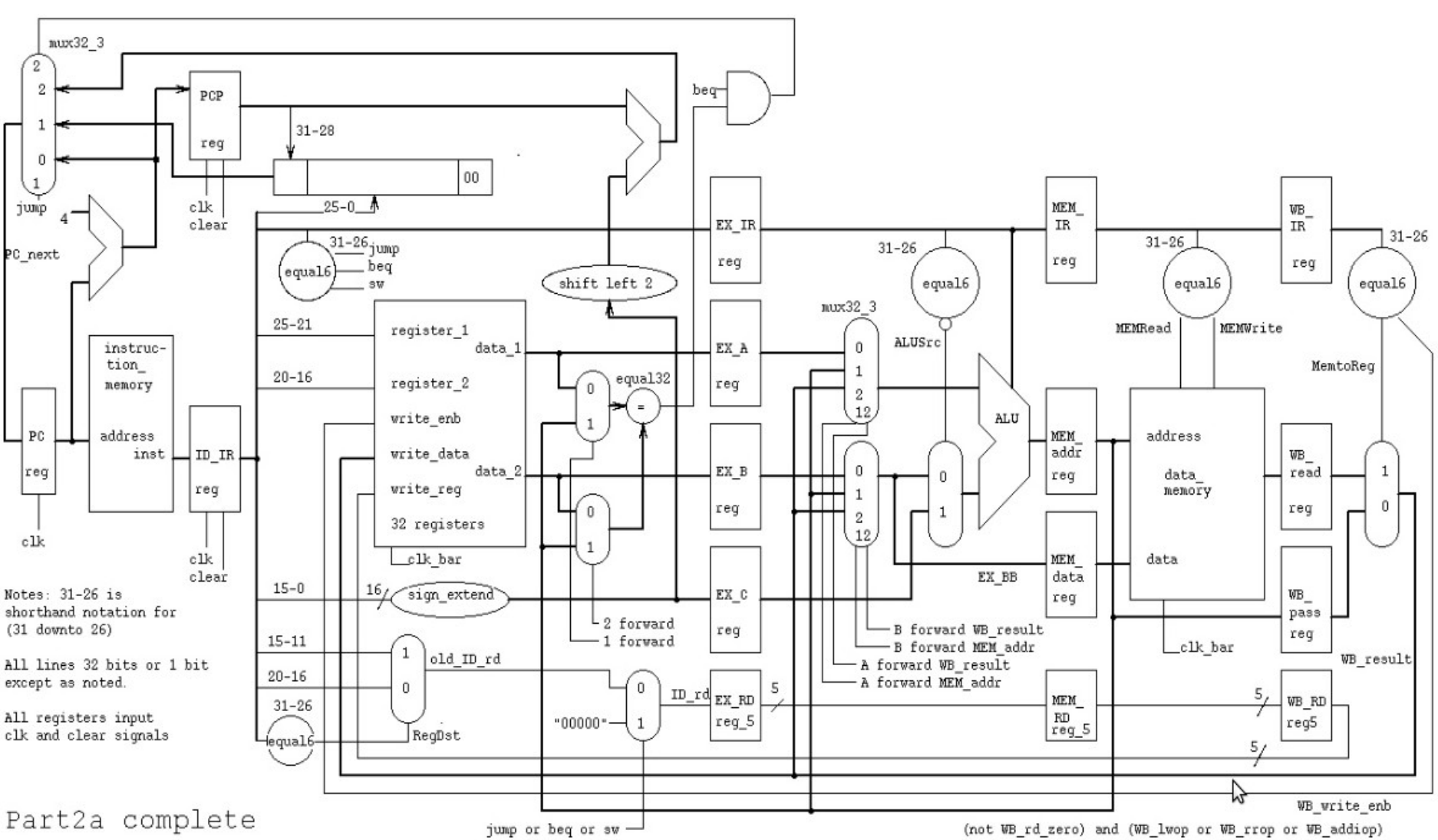
# Dependências em desvios

- **Instruções descartadas não podem ter afetado conteúdo de registradores e memórias**
  - **Isto geralmente é automático, porque escrita de valores é sempre feita no último estágio do pipeline**

- **Deve-se procurar antecipar a decisão sobre o desvio para o estágio mais cedo possível**

- **Desvios incondicionais**
  - **Sabe-se que é um desvio desde a decodificação da instrução ( segundo estágio do pipeline )**
  - **É possível evitar abandono de número maior de instruções**
  - **Problema: em que estágio é feito o cálculo do endereço efetivo do desvio?**

# 6. Dependências em desvios



# Técnicas de tratamento dos desvios condicionais

**1.Executar os dois caminhos do desvio**

**-Buffers paralelos de instruções**

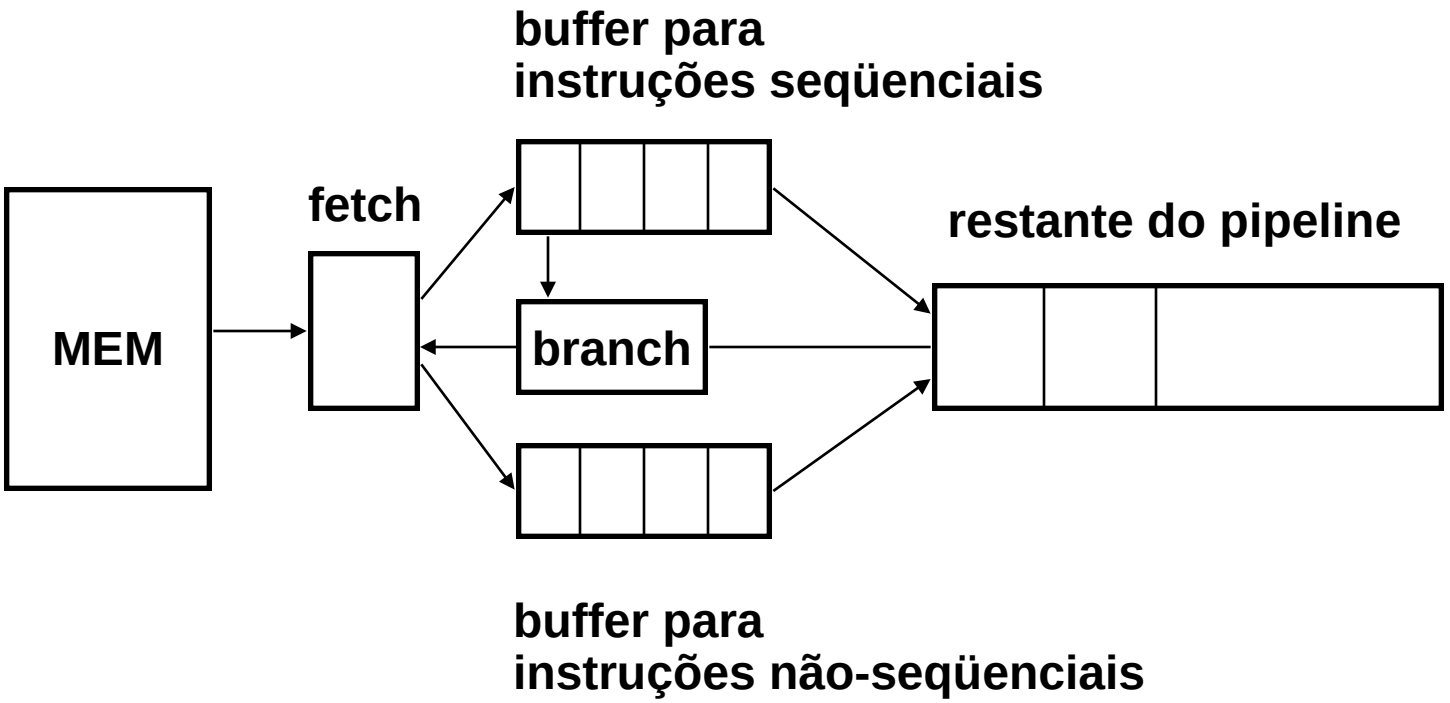
**2.Evitar o problema**

**-“delayed branch”**

**3.Prever o sentido do desvio**

- **Predição estática**
- **Predição dinâmica**

# 6.1 Buffers paralelos de instruções



# 6.2 *Delayed branch*

- **Compilador reorganiza as instruções do programa**
  - **Desloca instruções que não afetam o resultado para depois do desvio. É uma forma de adiantar a execução do desvio diminuindo o número de ciclos perdidos quando do descarte**
  - **Deve levar em conta as dependências entre instruções e manter a semântica do programa**
- **Funciona desde que instruções sejam independentes**
- **Desvio não ocorre imediatamente, e sim apenas após uma ou mais instruções seguintes**

# 6.2 *Delayed branch*

- **Caso mais simples: pipeline com 2 estágios – fetch + execute**
  - **desvio é feito depois da instrução seguinte**
  - **instrução seguinte não pode ser necessária para decisão sobre ocorrência do desvio**
  - **compilador reorganiza código**
  - **tipicamente, em 70% dos casos encontra-se instrução para colocar após o desvio**

- **Pipeline com N estágios**
  - **desvio é feito depois de N – 1 instruções**

# 6.3.1 Predição estática

- **Supor sempre mesma direção para o desvio**
  - **desvio sempre ocorre**
  - **desvio nunca ocorre**

- **Compilador define direção mais provável**
  - **instrução de desvio contém bit de predição, ligado / desligado pelo compilador**
  - **início de laço ( ou desvio para frente ): desvio improvável**
  - **final de laço ( ou desvio para trás ): desvio provável**

- **Até 85 % de acerto é possível**

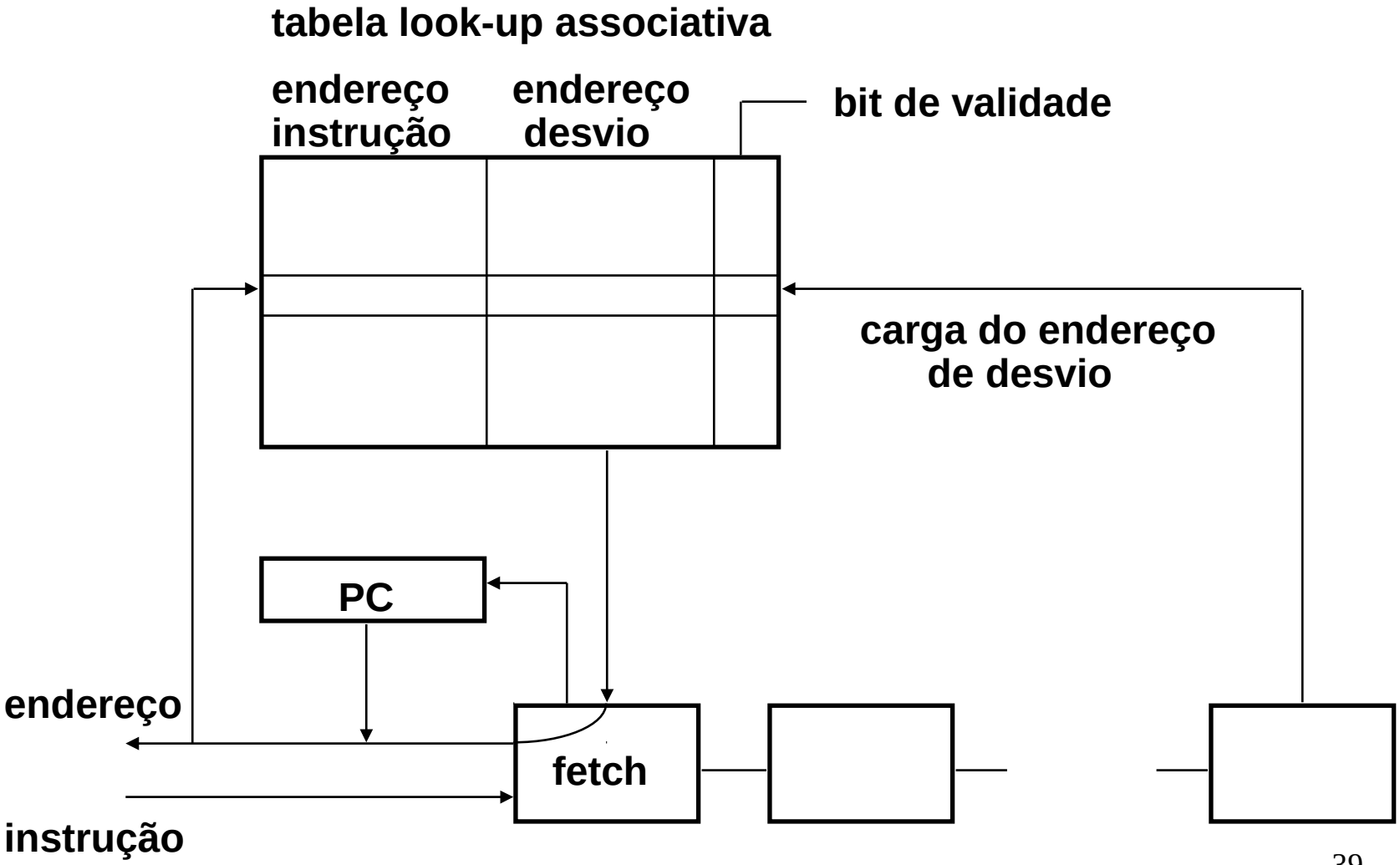
# 6.3.2 Predição dinâmica

- **Tabela look-up associativa armazena triplas**
  - **Endereços das instruções de desvio condicional mais recentemente executadas**
  - **Endereços de destino destes desvios**
  - **Bit de validade, indicando se desvio foi tomado na última execução**

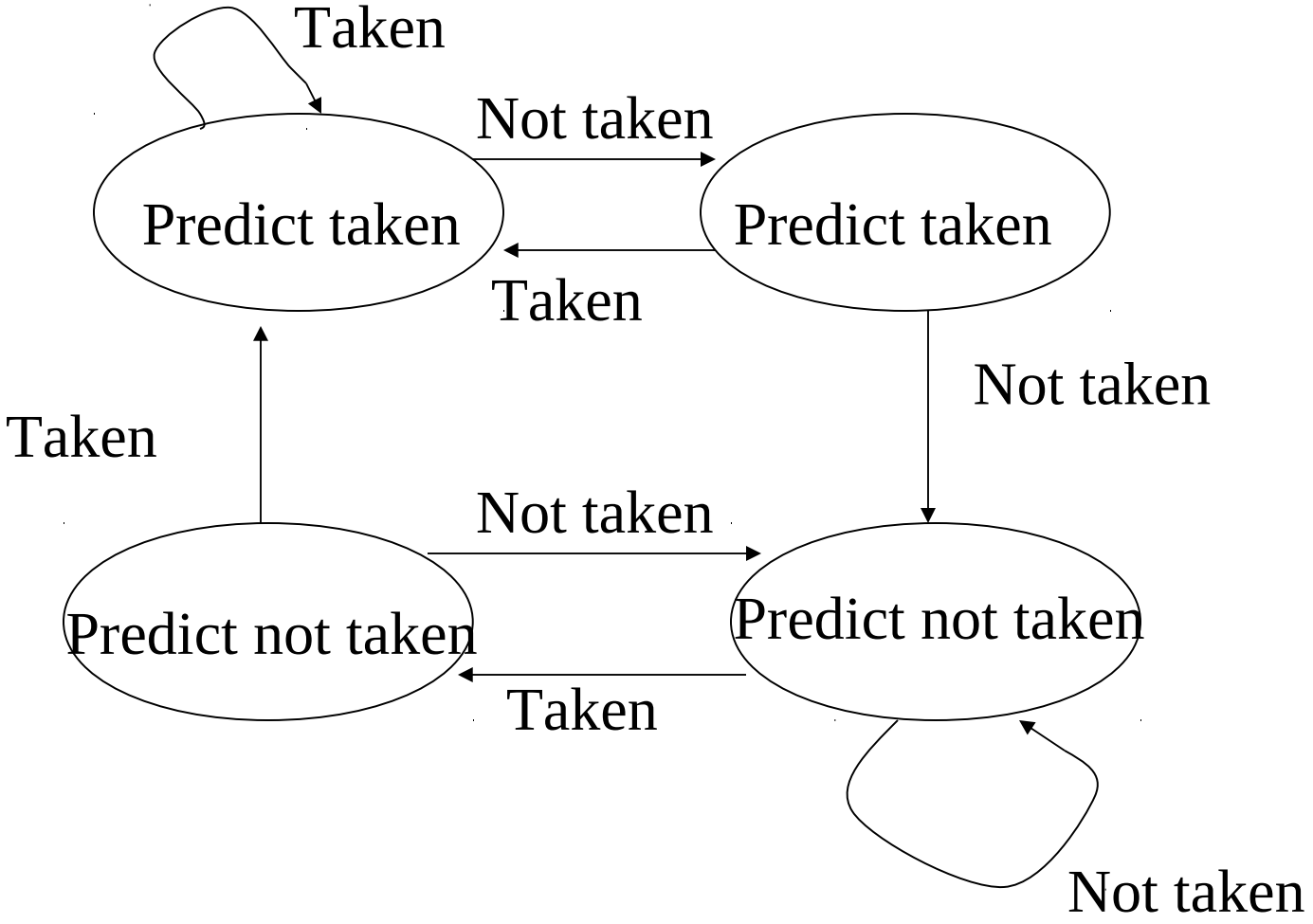
- **Quando instrução de desvio condicional é buscada na memória**
  - **Faz-se a comparação associativa na tabela, à procura do endereço desta instrução**
  - **Se o endereço é encontrado e bit de validade está ligado, o endereço de desvio armazenado na tabela é usado**
  - **No final da execução da instrução, endereço efetivo de destino do desvio e bit de validade são atualizados na tabela**

- **A tabela pode utilizar diversos mapeamentos e algoritmos de substituição**

# Predição dinâmica



# Esquema de predição de 2 bits



# Predição dinâmica com contador

- **Variação: BHT – *Branch History Table***
  - **Contador associado a cada posição da tabela**
  - **A cada vez que uma instrução de desvio contida na tabela é executada ...**
    - **Contador é incrementado se desvio ocorre**
    - **Contador é decrementado se desvio não ocorre**
  - **Valor do contador é utilizado para a predição**

# END