



# Aula 12

## Implementação MIPS

### MIPS – Uniciclo – Unidade Operativa



# Introdução

- O desempenho (tempo de execução) de uma máquina pode ser determinado por três fatores:
  - número de instruções executadas ( $I$ )
  - Número de ciclos por instrução (CPI)
  - período do *clock* ( $T$ )
- Para um determinado algoritmo, o compilador e a ISA determinam o número de instruções a serem executadas.
- A implementação do processador determina o período de *clock* e o CPI.



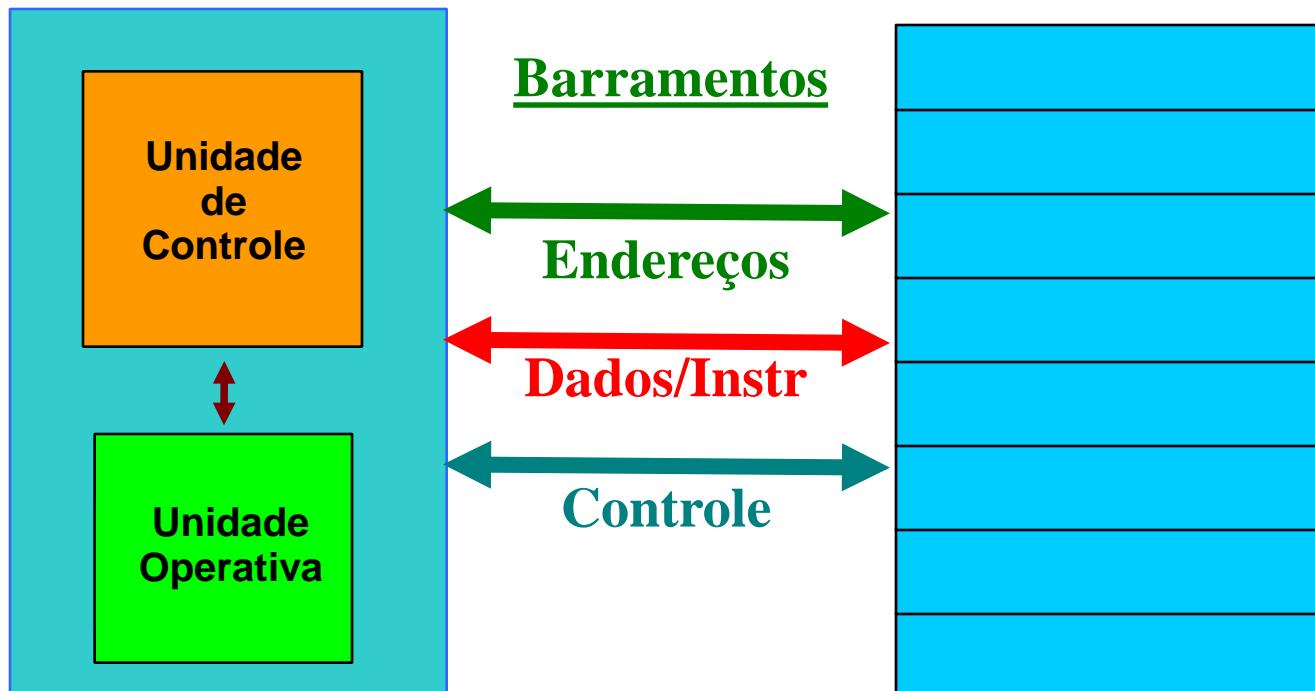
# Introdução

- Vamos fazer a implementação de apenas um pequeno subconjunto das instruções do MIPS
- O inter-relacionamento entre ISA e a implementação é exemplificado em três projetos alternativos:
  - Processador uniciclo
  - Processador multiciclo
  - Processador pipeline



# Arquitetura Von Neumann

## ■ Estrutura básica



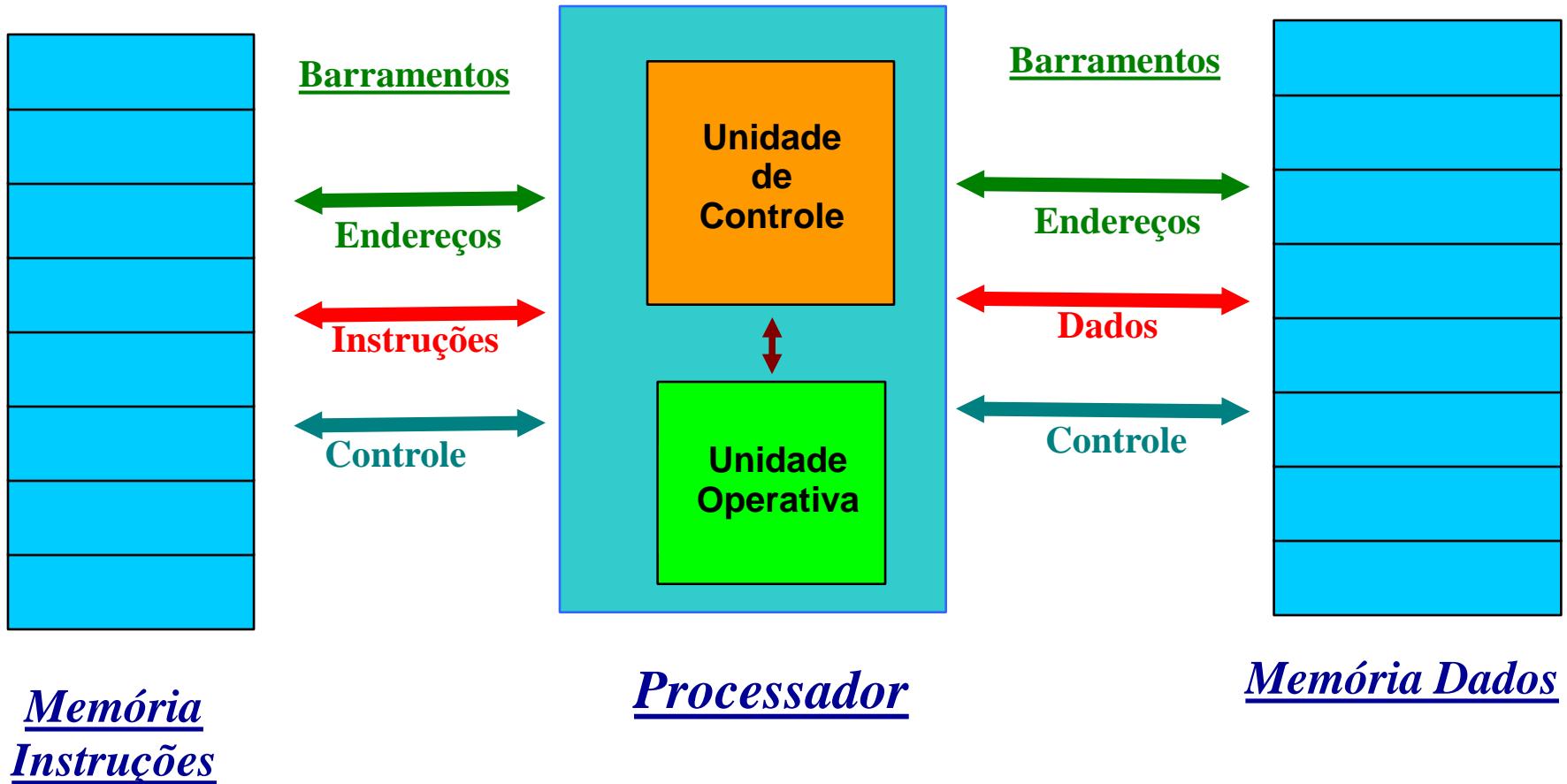
*Processador*

*Memória*



# Arquitetura Harvard

## ■ Estrutura básica





# Unidade Operativa

- Também chamada “Parte Operativa”, “Caminho de Dados” ou, em inglês, “*Datapath*”
- É construída a partir dos seguintes componentes:
  - elementos de armazenamento (registradores, flip-flops)
  - operadores lógico-aritméticos (somadores, ULA)
  - recursos de interconexão (barramentos, mux)



# Unidade Operativa MIPS

- Será projetada para implementar o seguinte subconjunto de instruções do MIPS:
  - Instruções de acesso à memória: lw e sw
  - Instruções lógicas e aritméticas: add, sub, and, or e slt
  - Instruções de fluxo de controle: beq e j

Conjunto incompleto, mas demonstra os princípios básicos de projeto.

Obs. No laboratório, além destas, estão implementadas várias outras

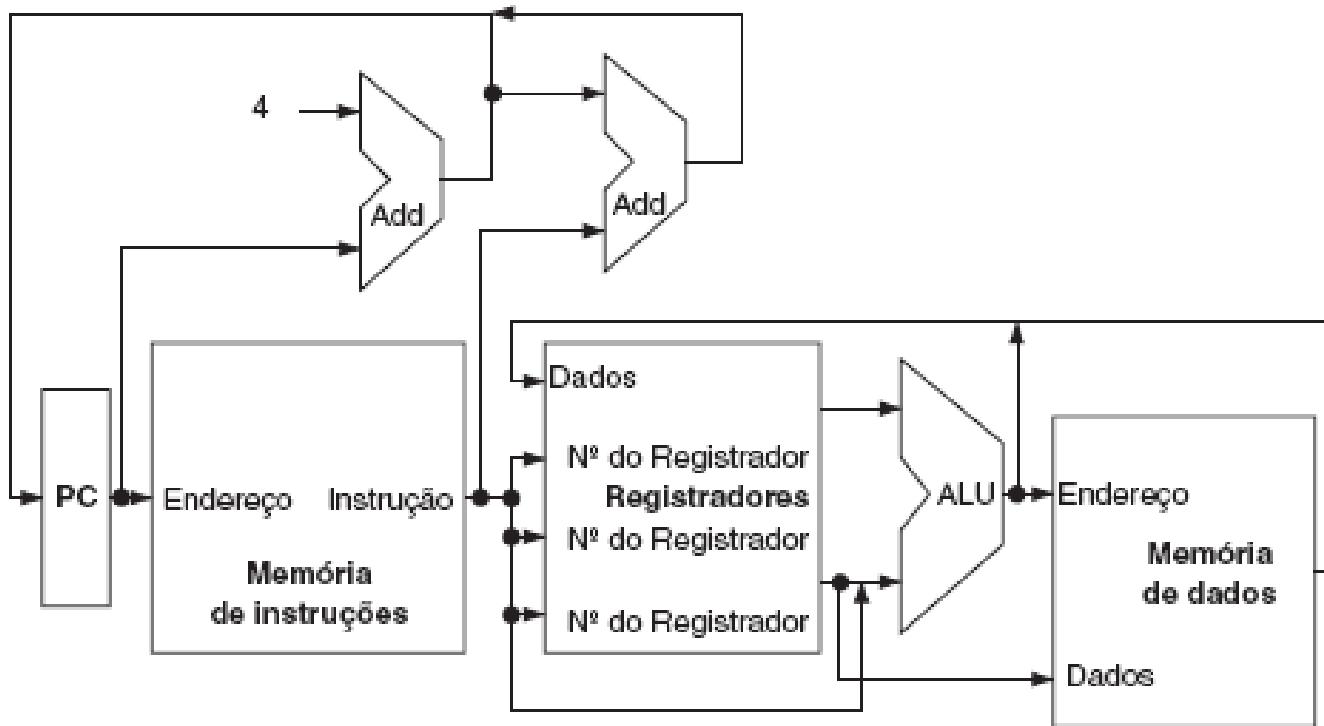


# Sinopse da Implementação

- Implementação genérica:
  - use o contador de programa (PC) para fornecer endereço da instrução
  - obtenha a instrução da memória
  - leia os registradores
  - use a instrução para decidir exatamente o que fazer
- Todas as instruções usam a ALU após lerem os registradores (exceto jump)
  - Por quê? Referência à memória? Aritmética? Fluxo de controle?

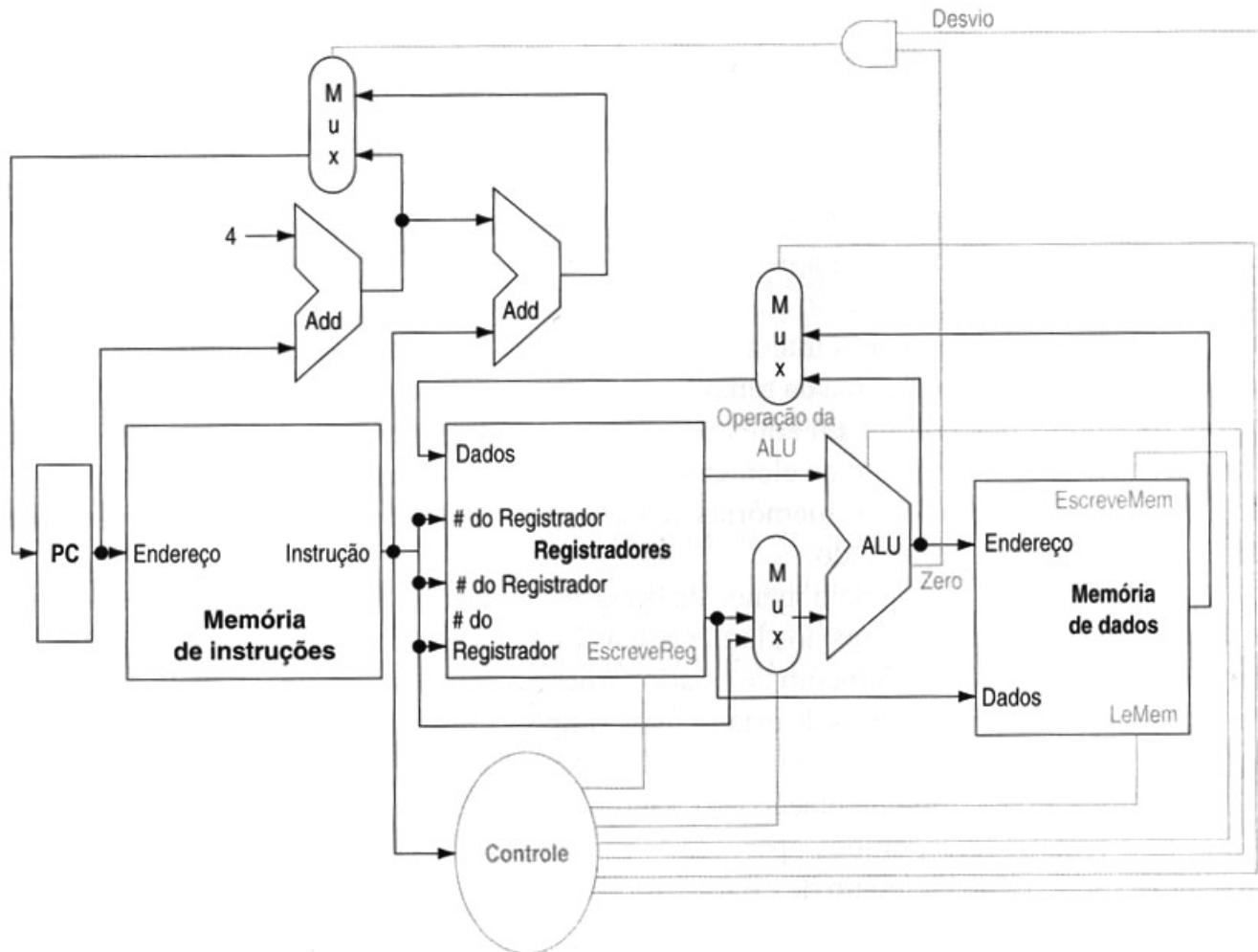


# Visão de alto nível da implementação



Obs.: - Em vários pontos temos dados vindo de duas origens diferentes  
- As unidades precisam ser controladas

# Implementação básica do subconjunto MIPS

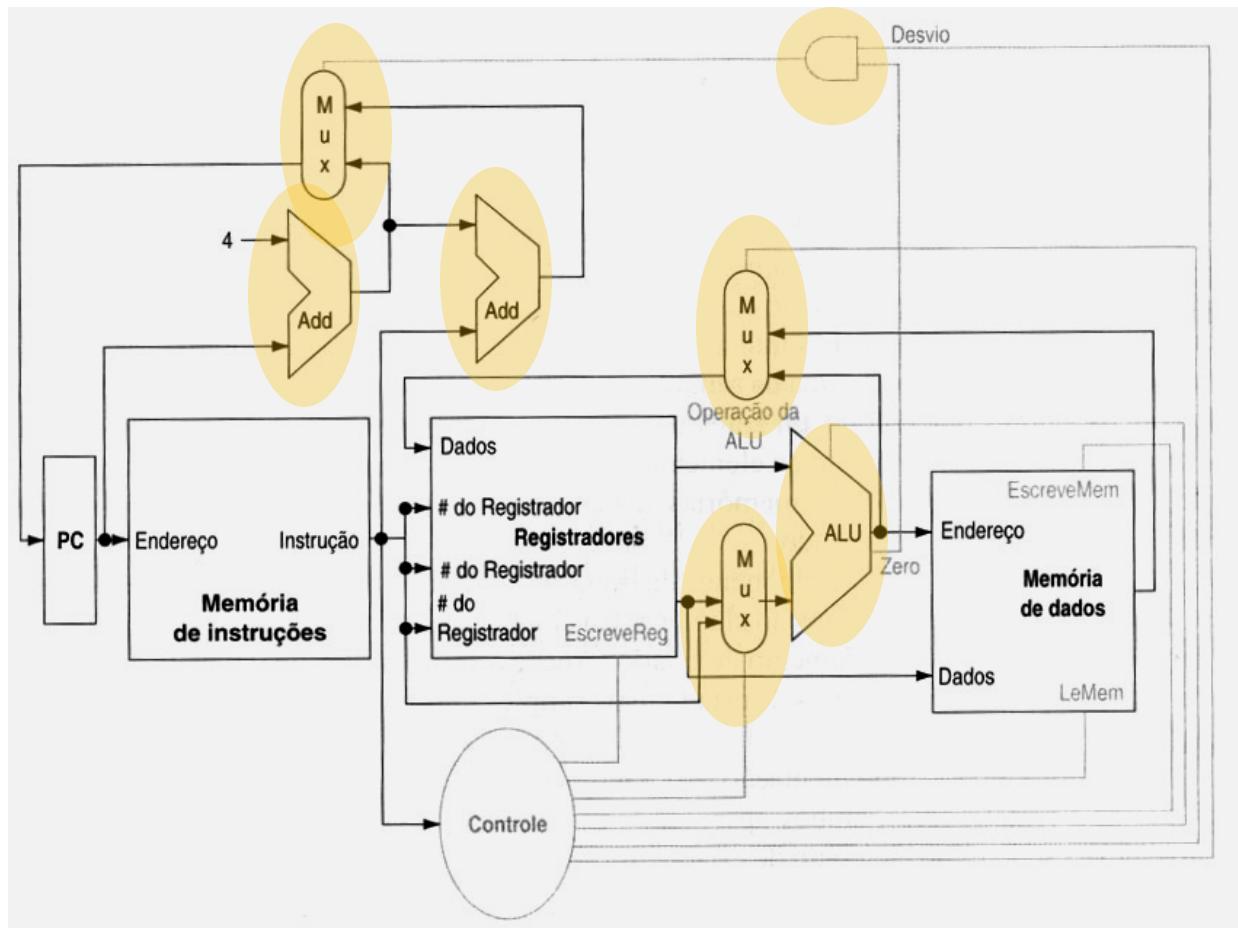


- Elementos combinacionais e sequenciais
- Unidade de controle
  - 1 único ciclo de clock
  - Vários ciclos de clock



# Componentes Combinacionais

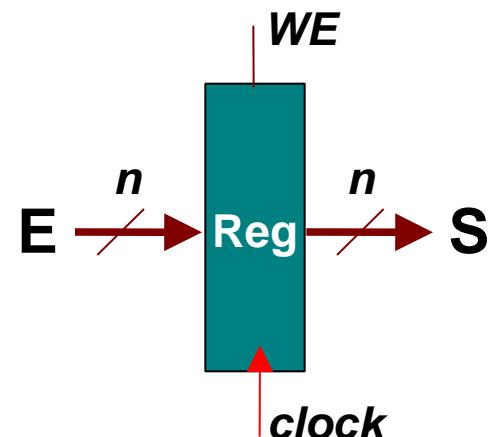
- Componentes combinacionais definem o valor de suas saídas apenas em função dos valores presentes nas suas entradas





# Componentes Sequenciais

- Componentes sequenciais têm um *estado*, que define seu valor em um dado instante de tempo
- Registrador:
  - Um conjunto de flip-flops tipo D
    - Com  $n$  bits de entrada e saída
    - entrada de habilitação de escrita
  - Habilitação de escrita (*Write Enable* - WE):
    - 0: o dado de saída não muda
    - 1: o dado de entrada será carregado (saída = entrada) na transição do *clock*





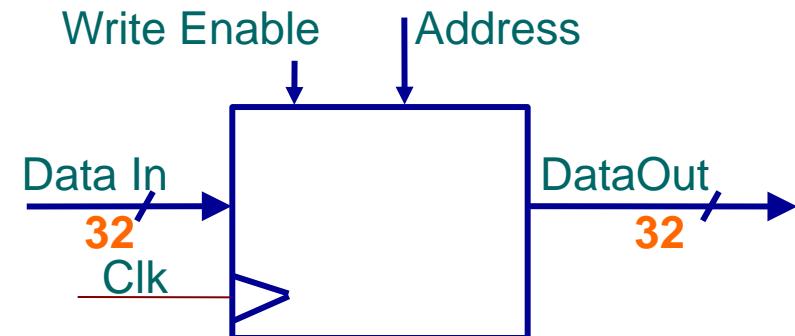
# Memória

## ■ Memória (*One/Two/Three... Ports*)

- Um barramento de entrada: *Data In*
- Um barramento de saída: *Data Out*

## ■ Uma palavra é selecionada por:

- Um endereço seleciona a palavra para ser colocada na saída (*Data out*)
- Write Enable = 1: Permite que a palavra selecionada seja escrita com a informação na entrada (*Data in*)



## ■ Entrada de Clock (*CLK*)

- Sincroniza os processos de acesso à memória

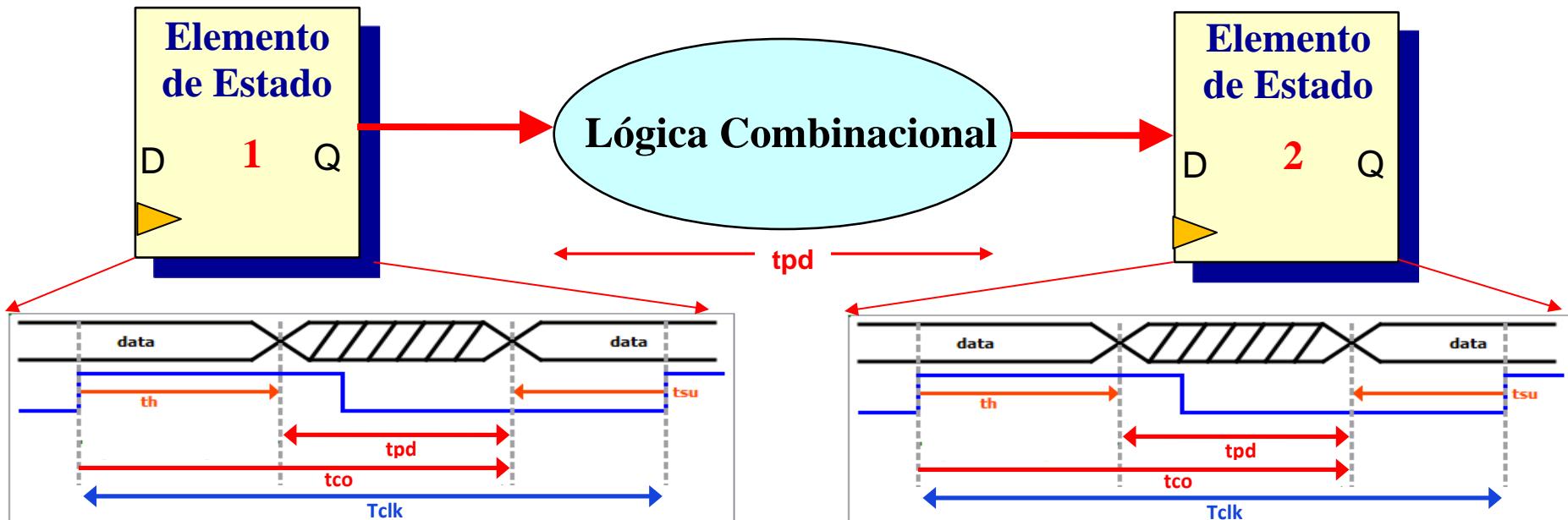


# Estratégia de Temporização

- Uma metodologia de temporização define quando os sinais podem ser lidos e quando eles podem ser escritos
- É necessário evitar situações de conflito, por exemplo, querer ler uma palavra e escrevê-la simultaneamente
- Será adotada uma metodologia de temporização sensível às transições do sinal do *clock*
- Nesta metodologia, qualquer valor armazenado nos **elementos de estado** só pode ser atualizado durante a **transição do sinal de relógio (*clock*)**



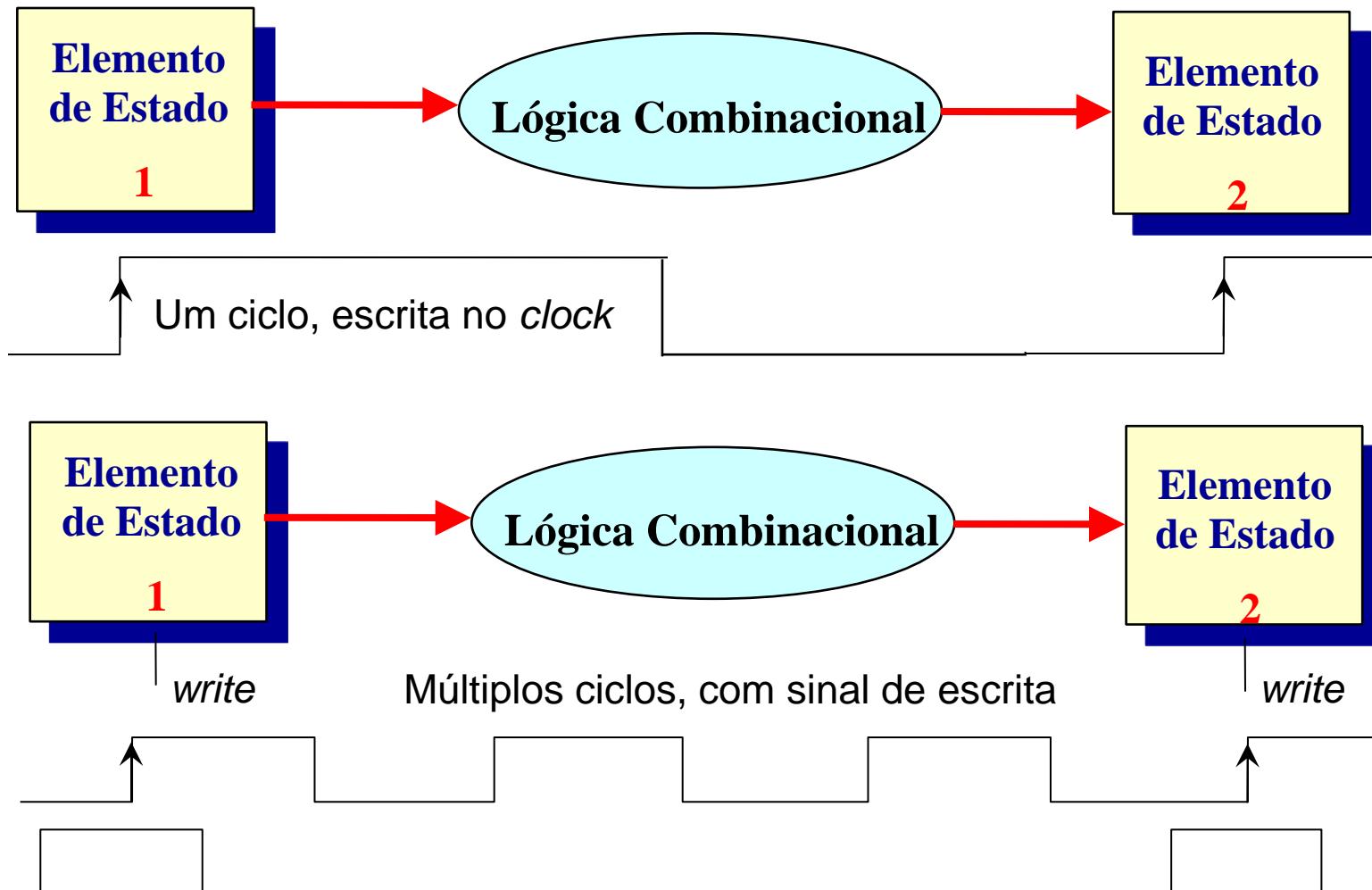
# Revisão: Temporização em circuitos digitais



- A escrita no elemento de estado 1 deve respeitar os tempos de:
  - tempo de pré-carga (*setup time*, tsu) do elemento 1
  - tempo de hold (*hold time*, th) do elemento 1
- O elemento de estado 2 só pode ser escrito depois que os dados em sua entrada estarem estáveis
  - atraso de propagação da saída dado o clock (*clock to output*, tco) do elemento 1
  - atraso da lógica combinacional (*propagation delay*, tpd= $\max(tp_{HL}, tp_{LH})$  )
  - tempo de pré-carga (*setup time*, tsu) do elemento de estado 2



# Estratégia de Temporização





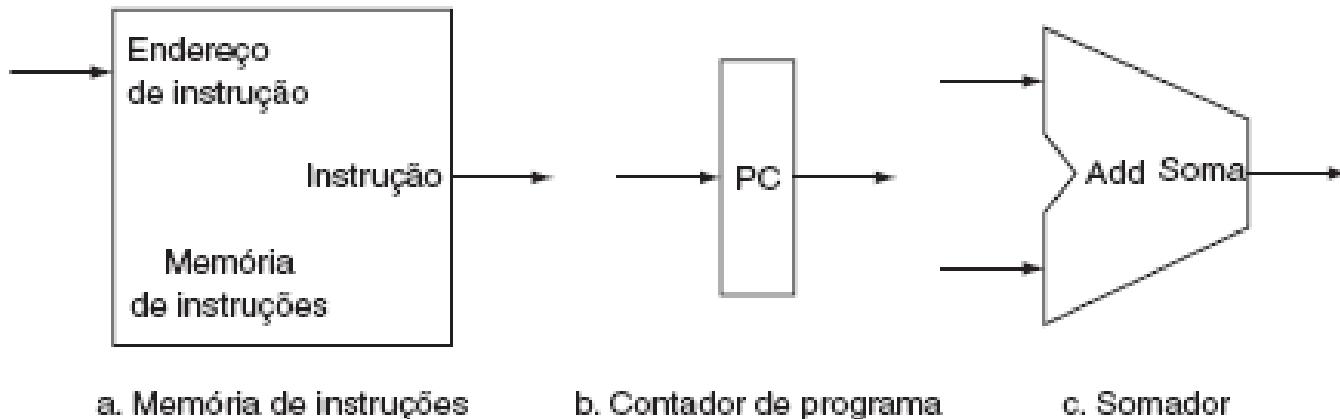
# Criando a Unidade Operativa

- Uma maneira de se começar o projeto de uma unidade operativa (***data path***) é examinar cada um dos componentes necessários para a execução de cada uma das classes de instruções do MIPS
- Elementos necessários:
  - um lugar para armazenar as instruções (***memória de instruções***)
  - Um registrador para armazenar o endereço de instrução (***Program Counter – PC***)
  - Um contador para incrementar o PC, para compor o endereço da próxima instrução (soma 4 para endereçar próxima instrução)



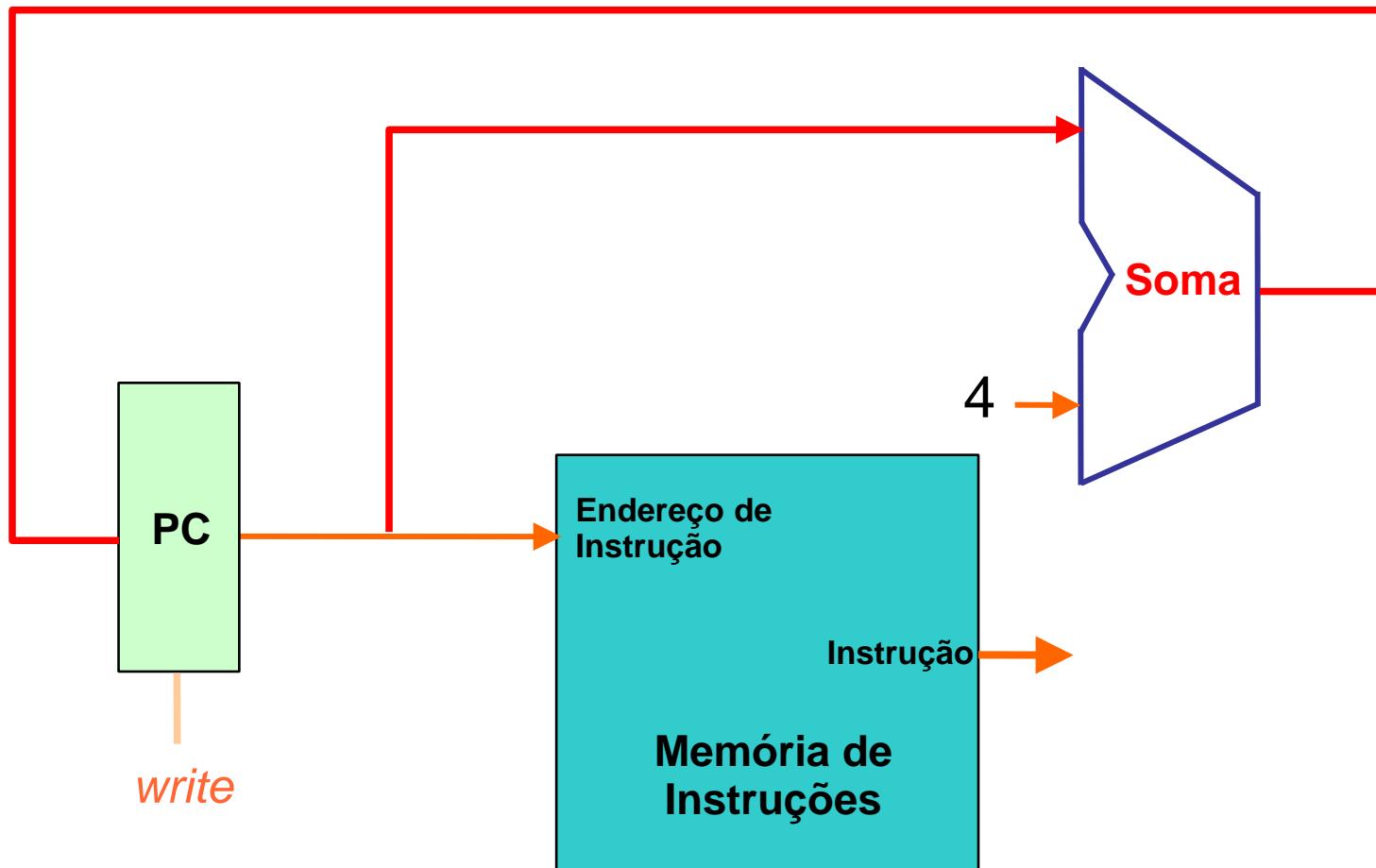
# Criando a Unidade Operativa ...

## ■ Elementos Básicos





# Busca de Instruções





# Instruções Lógico-Aritméticas

- Formato de uma instrução tipo R no MIPS:

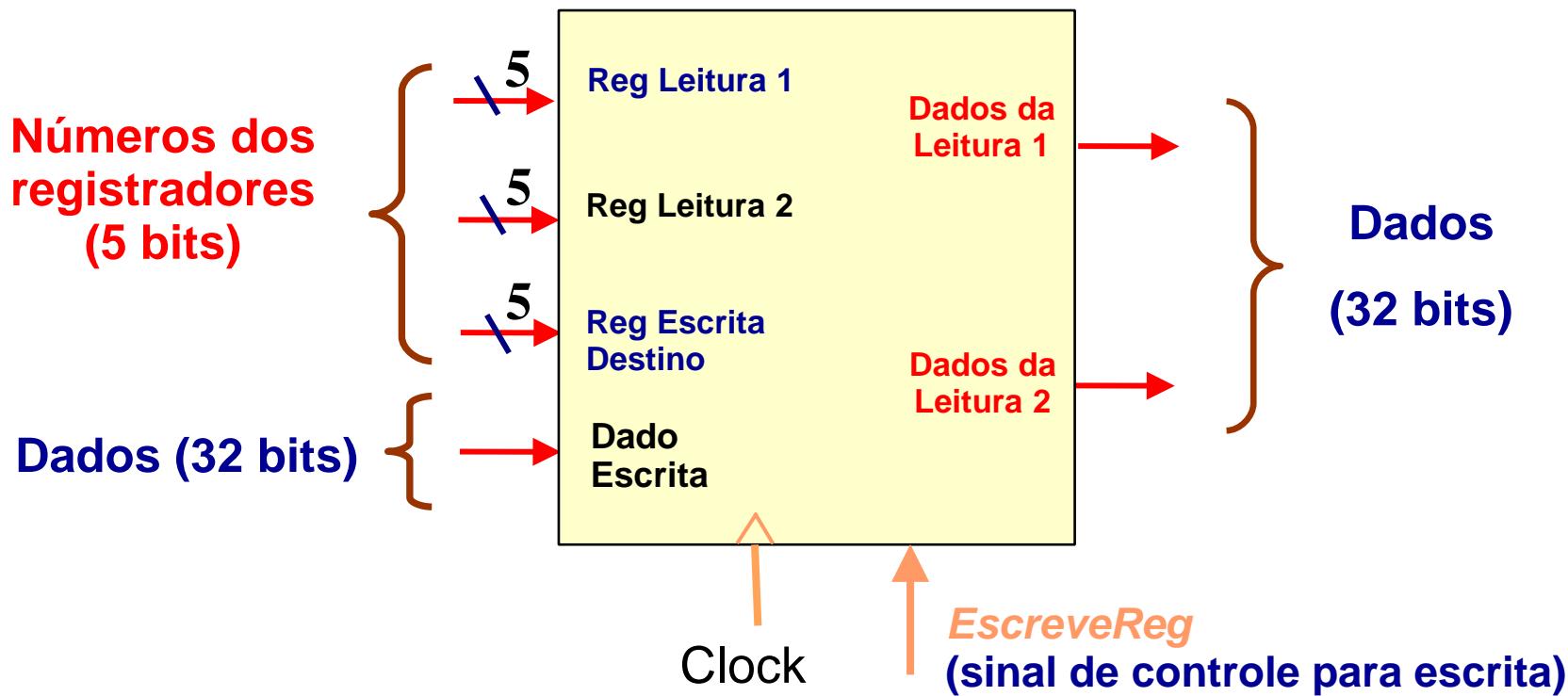
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- Semântica:
  - $\$rd \leftarrow op(\$rs, \$rt)$
- Estrutura de suporte: banco de registradores



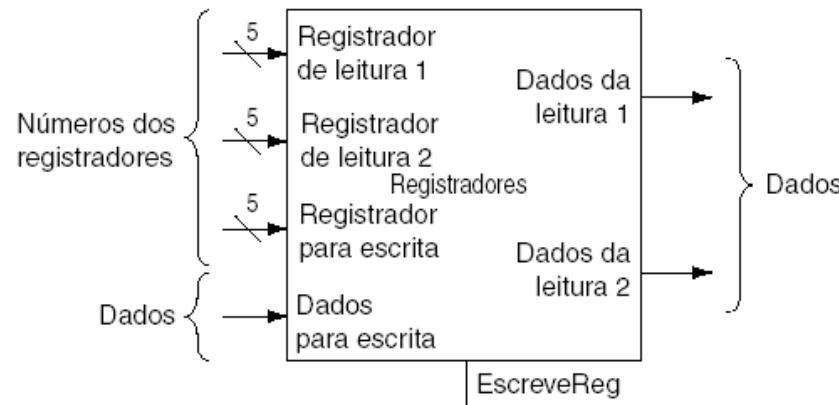
# Banco de Registradores

- Dupla porta: leitura de dois registradores ao mesmo tempo
- Sinal de controle para escrita - leitura não necessita controle





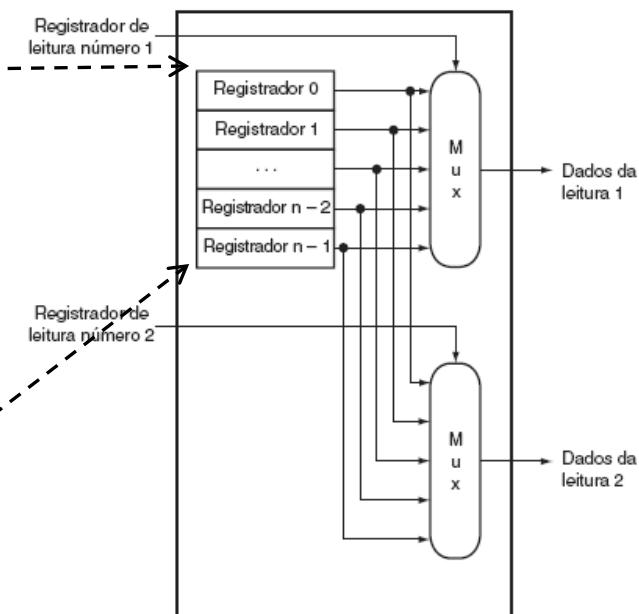
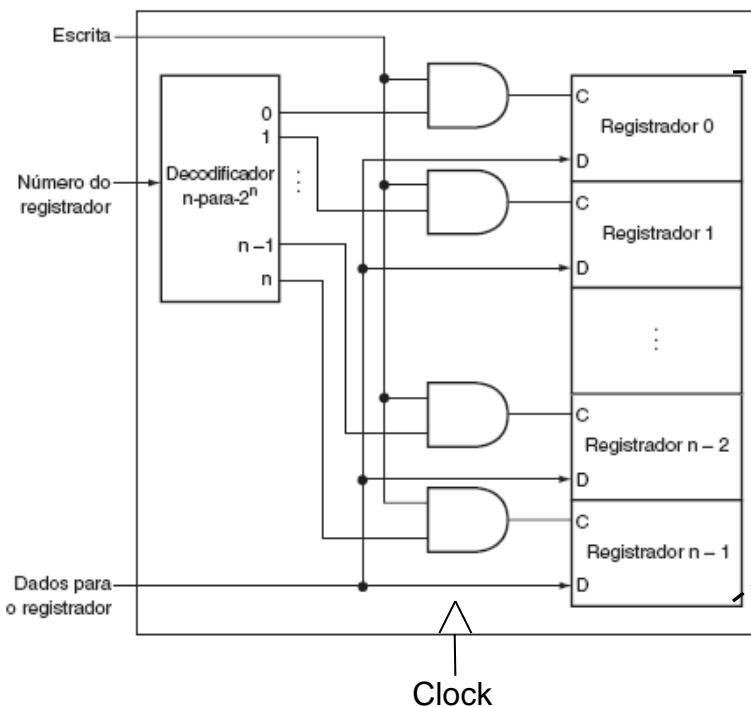
# Banco de Registradores



a. Registradores

Leitura:

Escrita:

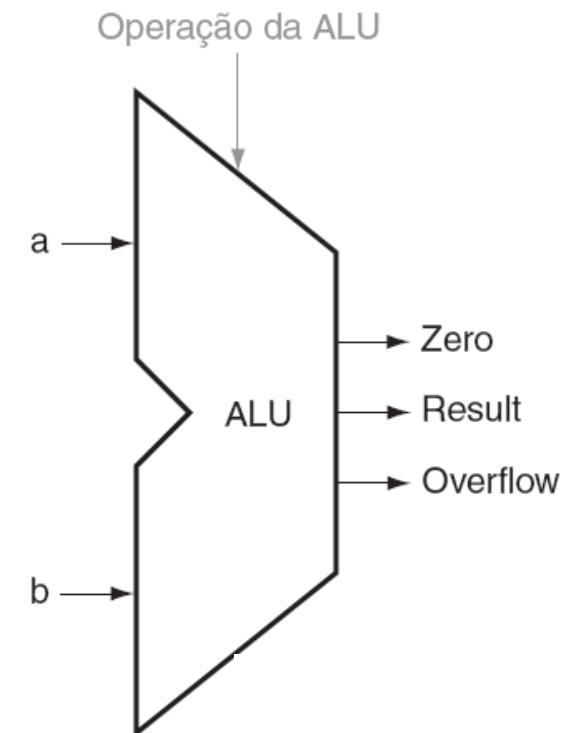




# Projeto da ULA

- A ULA foi desenvolvida no capítulo anterior
- 4 bits de controle indicam a operação a ser realizada

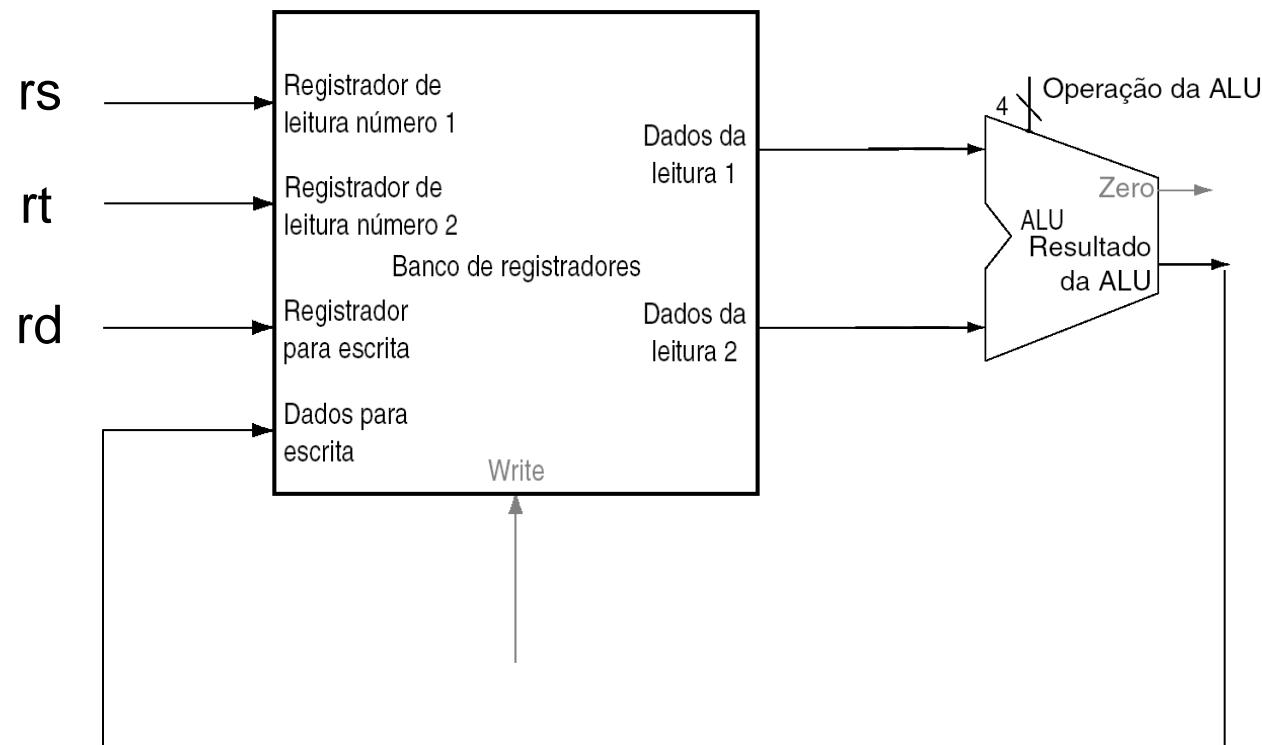
Linhas de controle da ALU	Função
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR





# Unidade Operativa: Instruções Tipo-R

<b>op</b>	<b>rs</b>	<b>rt</b>	<b>rd</b>	<b>shamt</b>	<b>funct</b>
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits



Ex.: add \$s1,\$s2,\$s3

0	18	19	17	0	32
---	----	----	----	---	----



# Unidade Operativa: Instruções Tipo-I

- No nosso caso simples: acesso a memória (lw e sw):

op	rs	rt	Imediato
6 bits	5 bits	5 bits	16 bits

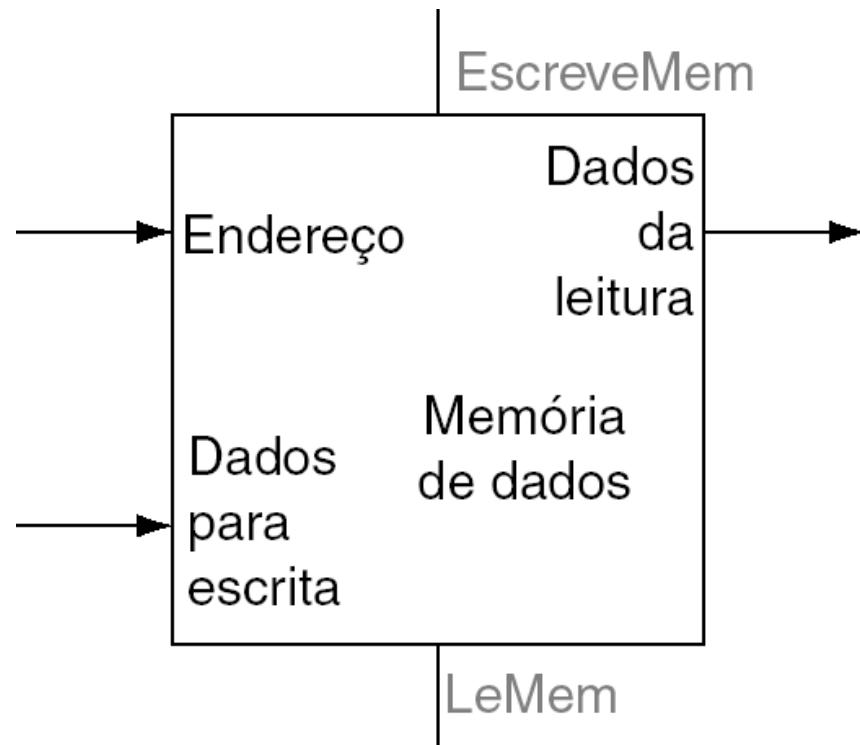
- Ex:
  - lw \$t0, 32(\$t1)
  - end = \$t1 + 32

35	9	8	32
----	---	---	----



# Memória

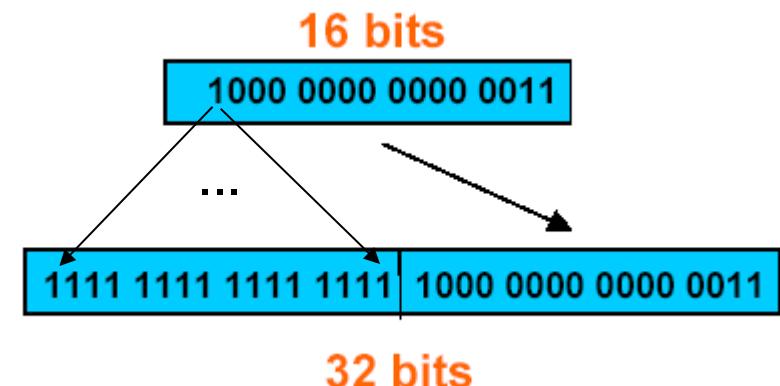
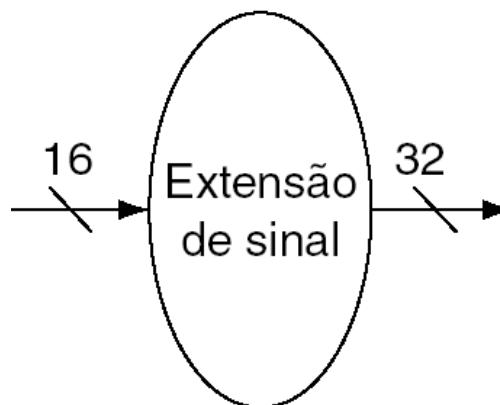
- Memória com um barramento de entrada independente do de saída
- Controle de escrita (*write*) e leitura (*read*)
- Barramento de endereços
- Um acesso de cada vez





# Extensão de Sinal do Imediato

- Imediato deve ser extendido de 16 para 32 bits, mantendo-se o sinal
  - se for negativo, 16 bits superiores = 1
  - se for positivo, 16 bits superiores = 0



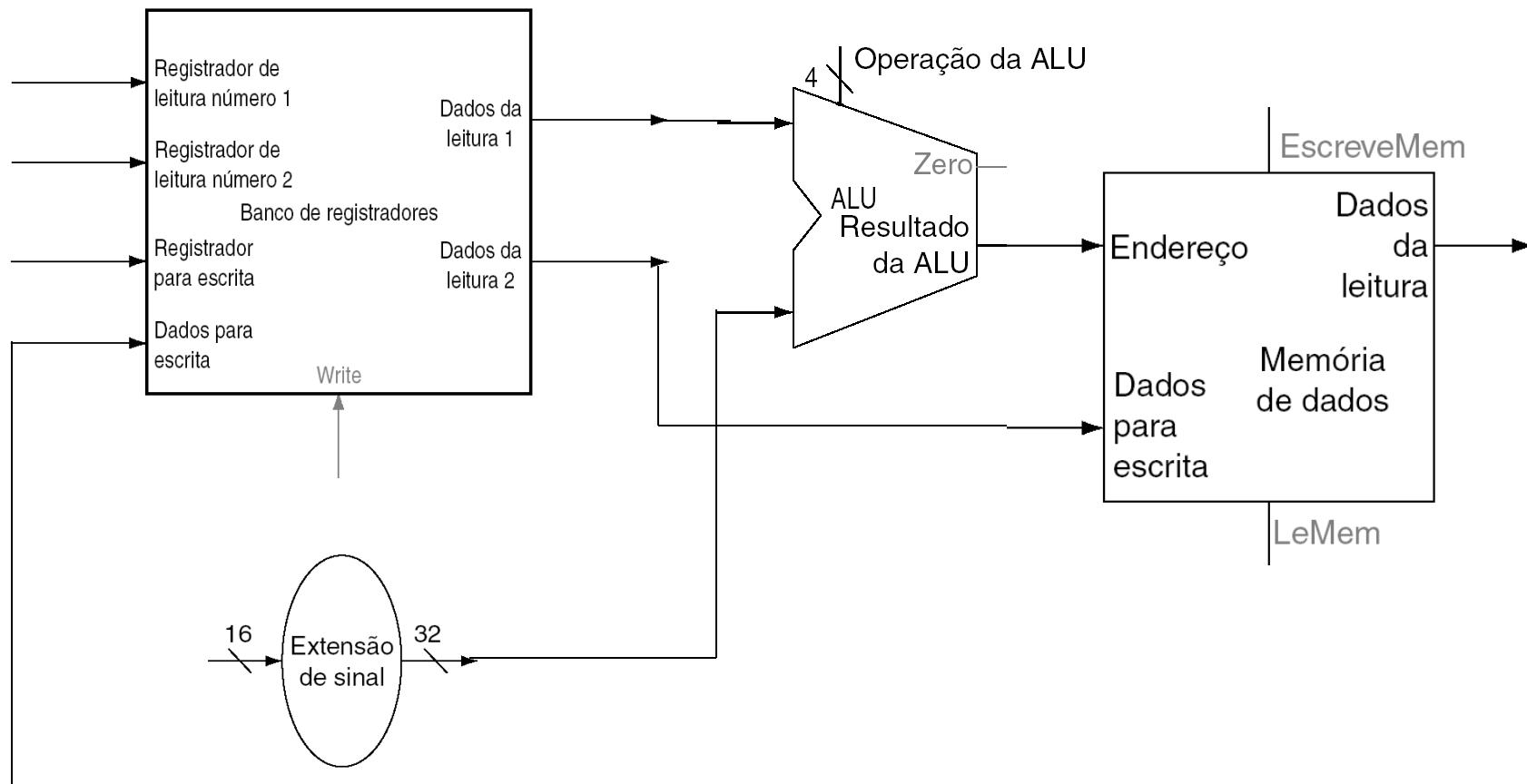


# Acesso a Memória

- **lw** lê um dado da memória e escreve em um registrador
  - conexão entre a saída da memória e a entrada do banco de registradores
- **sw** lê um dado de um registrador e escreve na memória
  - ligação entre saída do banco de registradores e entrada de dados da memória
- endereço calculado através da ULA
  - saída da ULA ligada ao barramento de endereços da memória



# Unidade Operativa: Instruções Tipo-I acesso memória



<b>op</b>	<b>rs</b>	<b>rt</b>	<b>Imediato</b>
<b>6 bits</b>	<b>5 bits</b>	<b>5 bits</b>	<b>16 bits</b>



# Instruções de Desvio

- **beq \$t1, \$t2, desloc**
  - compara dois registradores
  - soma *desloc* words ao endereço PC+4 se  $\$t1 == \$t2$ 
    - PC + 4 é o endereço da próxima instrução
  - no Assembly utiliza-se uma versão simplificada, com o rótulo do destino
    - `beq $t1, $t2, LABEL`
    - neste caso, o montador calcula o deslocamento a ser usado
  - *desloc* é um deslocamento de **palavras**, ou seja, cada unidade de *desloc* corresponde a 4 bytes

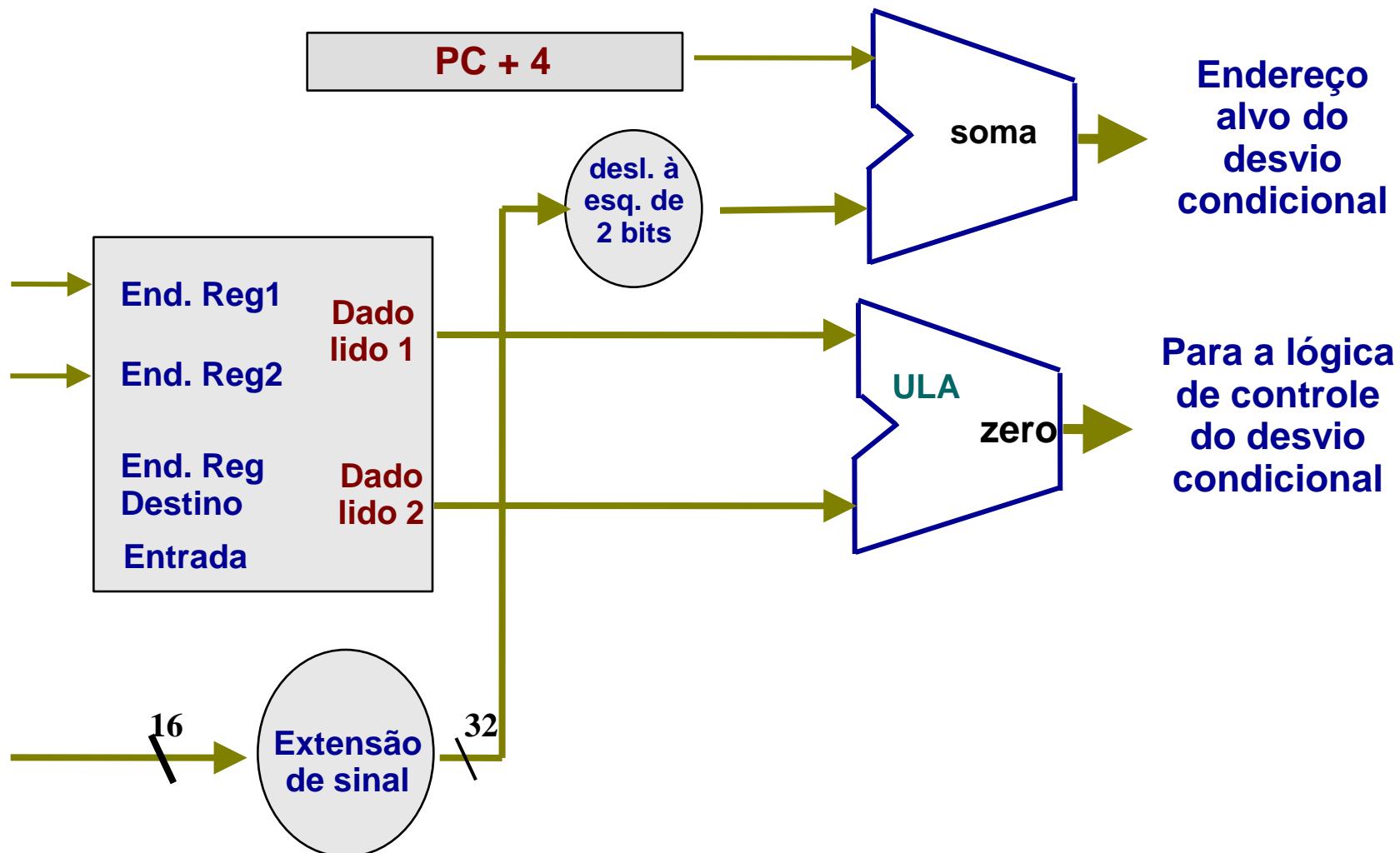


# Instruções de Desvio

- A comparação entre os registradores é feita subtraindo-os na ULA e verificando se o resultado é zero
- O PC deve ser carregado com  $PC + 4$  ou  $PC + 4 + desloc * 4$ , de acordo com o resultado do teste
- A multiplicação de *desloc* por 4 é feita deslocando-se 2 bits a esquerda o seu valor



# Círcuito Cálculo Endereço Desvio





# Observações

- Para realizar a comparação dos dois operandos precisamos usar o banco de registradores
- O cálculo do endereço de desvio foi incluído no circuito
- Na instrução não é preciso escrever no banco de registradores
- A comparação será feita pela ULA, subtraindo-se os registradores e utilizando a saída zero para verificar a igualdade



# Unidade Operativa: Quase-Final

