# Organização e Arquitetura de Computadores I

## Caminho de Dados

# Sumário

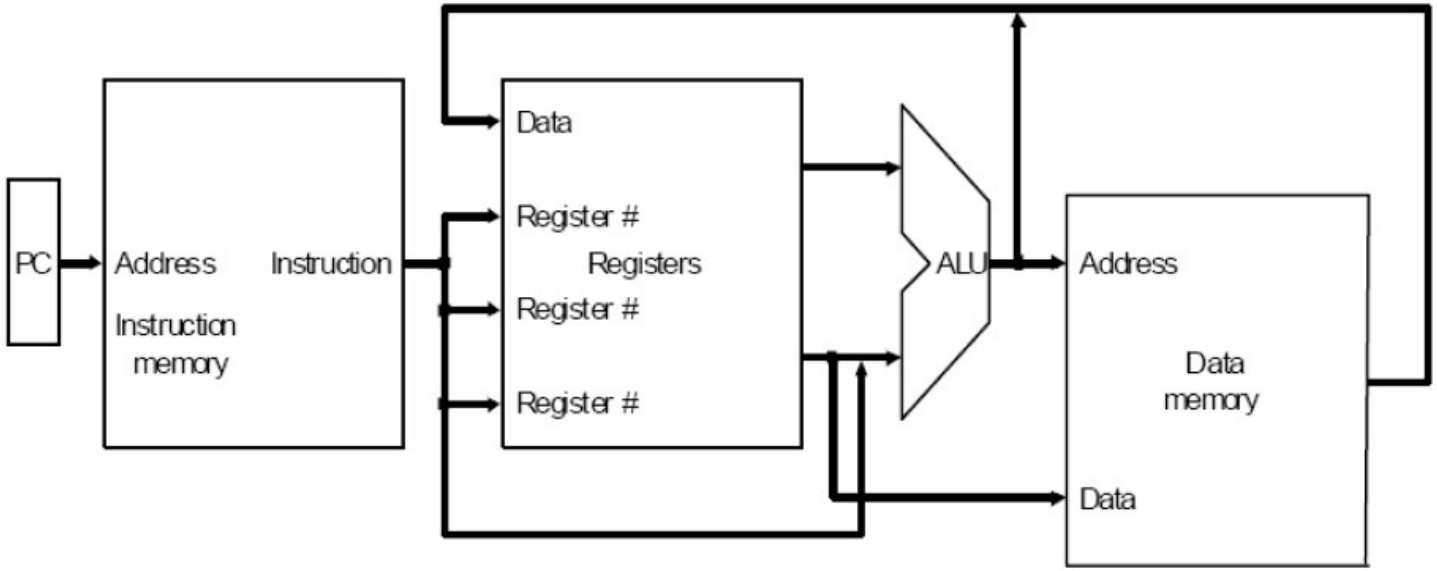
- Introdução
- Convenções Lógicas de Projeto
- Construindo um Caminho de Dados
- O Controle da ULA
- Projeto da Unidade de Controle Principal
- Operação do Caminho de Dados
  - Operação do Caminho de Dados – Tipo R
  - Operação do Caminho de Dados – Load
  - Operação do Caminho de Dados – beq
- Uma Implementação Multiciclo
- Etapas de Execução

# Introdução

- Até o momento, analisamos as instruções MIPS básicas:
  - Aritméticas (add, sub);
  - Lógicas(or, slt);
  - De Referência à memória (lw, sw);
  - De desvio(beq, j, jr).
- Para cada instrução, duas etapas são idênticas:
  - Enviar o contador de Programa (PC) à memória que contém o código e buscar a instrução dessa memória;
  - Ler um ou mais registradores, usando campos da instrução para selecionar os registradores a serem lidos .

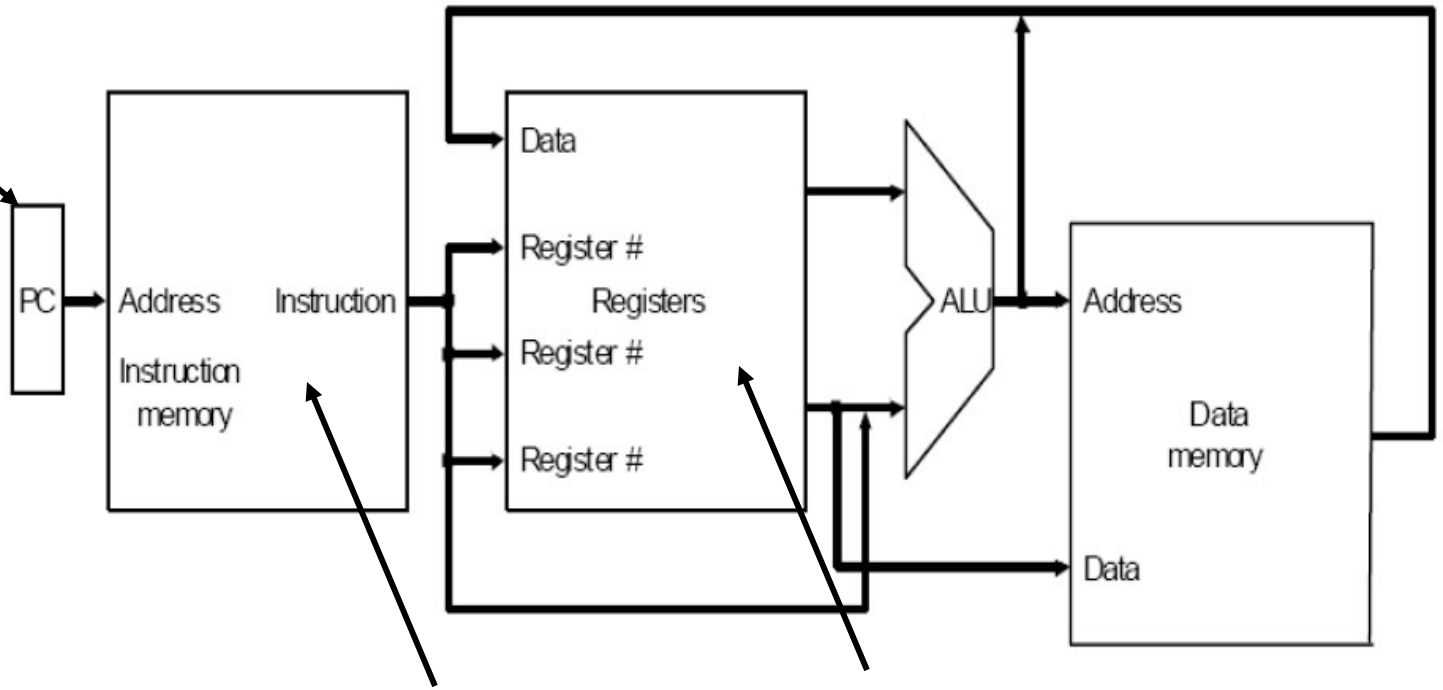
# Introdução

- Após essas duas etapas, as ações necessárias para completar a instrução dependem da classe de instrução:



# Introdução

Informa qual o endereço da próxima instrução a ser executada




Depois que a instrução é buscada, os registradores usados como operandos pela instrução são especificados com campos dessa instrução

# Introdução

- Registradores:
  - Podem ser operadores para calcular endereço (**lw e sw**);
  - Podem ser operadores para calcular um resultado aritmético (**lógica e aritmética**);
  - Podem ser operadores para calcular uma comparação (desvio).
- Se a instrução for uma **instrução lógica ou aritmética**, o resultado da ULA precisa ser escrito em um registrador.
- Se a operação for um ***load* ou *store***, o resultado da ULA é usado como um endereço para armazenar o valor de um registrador ou ler um valor da memória para um registrador. O resultado da ULA ou memória é escrito de volta no banco de registradores.
- Os **desvios** exigem o uso da saída da ULA para determinar o próximo endereço de instrução, que vem da ULA ou de um somador que incrementa PC atual em 4.

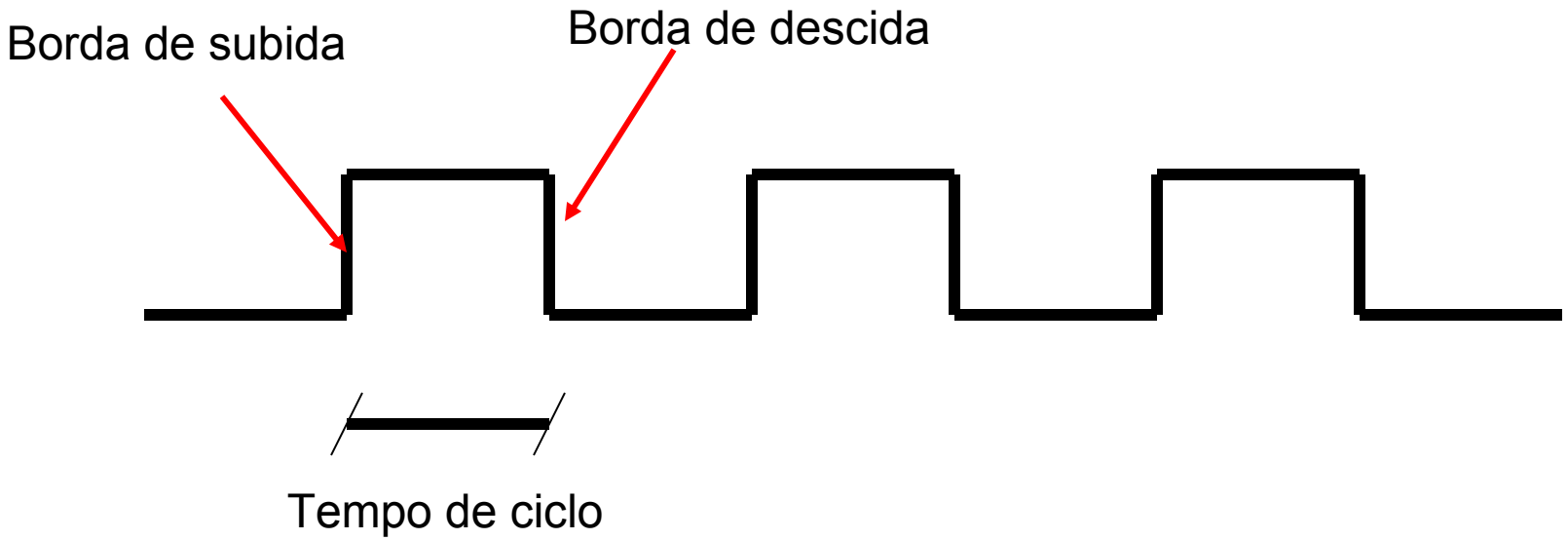
# Convenções Lógicas de Projeto

- As unidades funcionais na implementação MIPS consistem em dois tipos diferentes de elementos lógicos:
  - **Combinacionais**: saídas dependem apenas das entradas atuais (**Portas Lógicas**);
  - **De Estado**: que contém estado se tiver algum armazenamento interno (**Flip-Flop**).

# Convenções Lógicas de Projeto

- Uma **metodologia de *clocking*** define quando os sinais podem ser lidos e quando podem ser escritos. **É importante para especificar a sincronização das leituras e escritas, evitando que as mesmas ocorram simultaneamente, o que geraria um erro.**
- Uma **metodologia de sincronização acionada por transição**, significa que quaisquer valores armazenados em um elemento lógico sequencial são atualizados apenas em uma transição do *clock*.

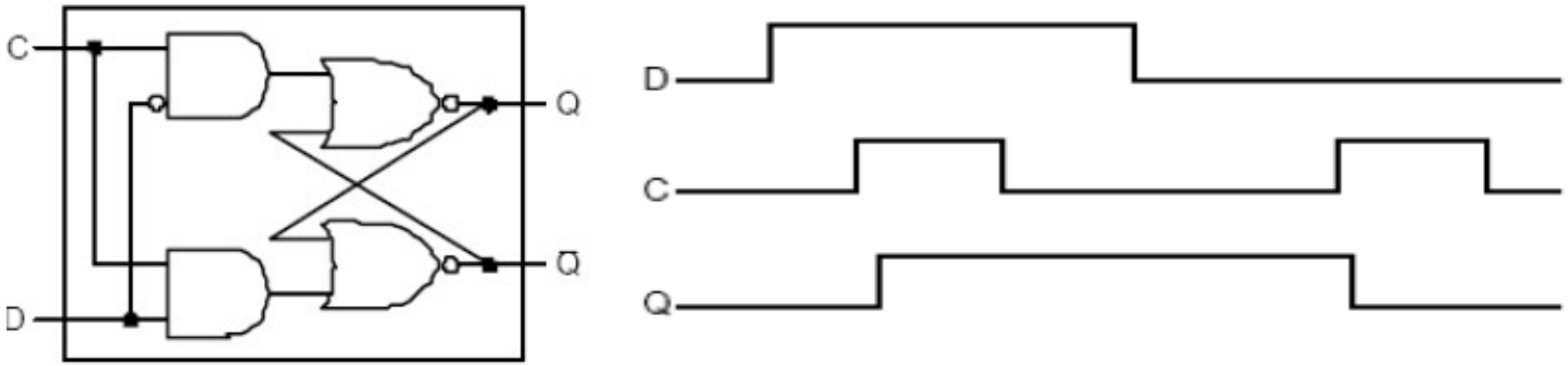
# Convenções Lógicas de Projeto



# Convenções Lógicas de Projeto

- Uma metodologia acionada por transição permite que um elemento de estado seja lido e escrito no mesmo ciclo de *clock* sem criar uma disputa que poderia levar a valores de dados indeterminados.

# Convenções Lógicas de Projeto

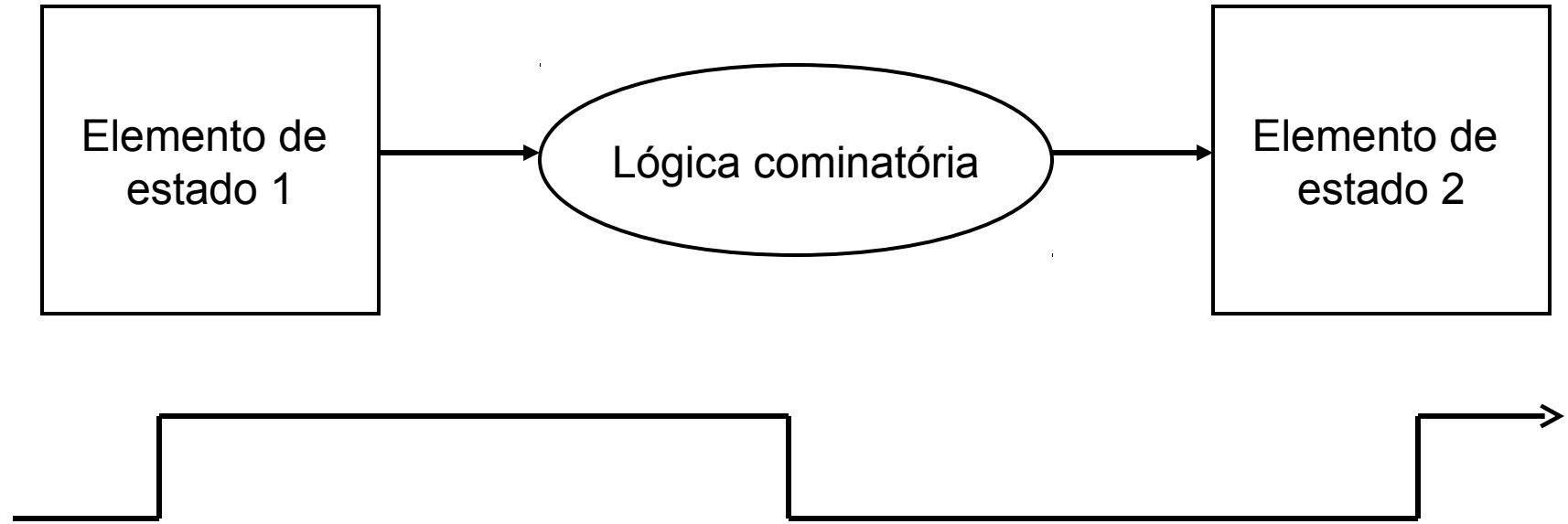



- O valor a ser armazenado (D);
- O sinal do relógio (C) indicando leitura ou escrita;
- O valor do estado interno (Q) e o seu complemento (Q).

# Convenções Lógicas de Projeto

- Como apenas os elementos de estado podem armazenar valores de dados, qualquer coleção de lógica combinatória precisa ter suas entradas vindo de um conjunto de elementos de estados e suas saídas escritas em um conjunto de elementos de estado. **As entradas são valores escritos em um ciclo de *clock* anterior, enquanto as saídas são valores que podem ser usados em um ciclo de *clock* seguinte.**

# Convenções Lógicas de Projeto

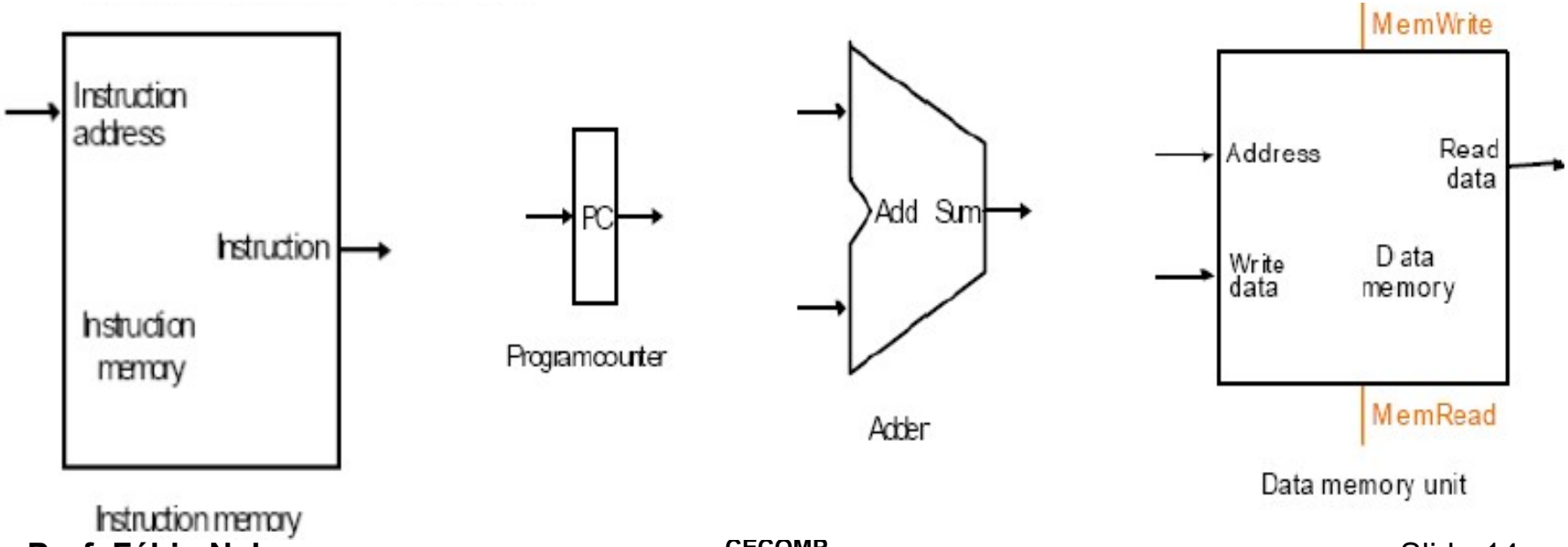





Ciclo de *clock*

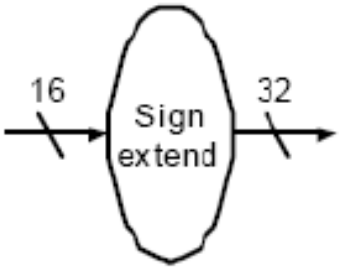
# Construindo um Caminho de Dados

- Principais componentes necessários para executar cada classe de instrução MIPS.
  **Elementos do caminho de dados:**

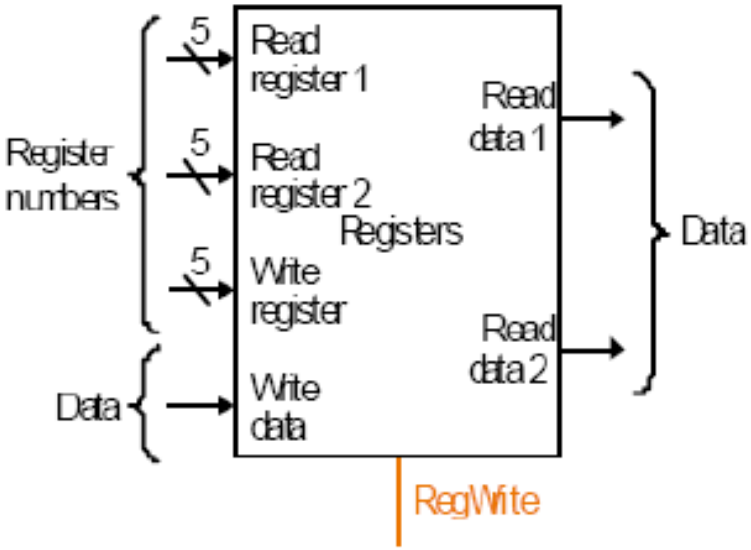


# Construindo um Caminho de Dados

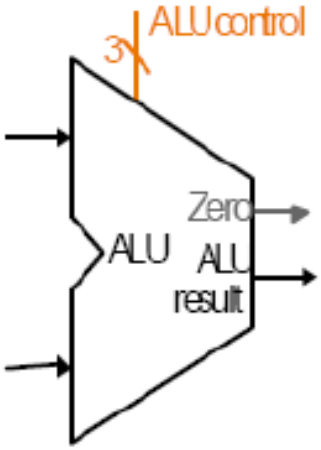
- **Elementos do caminho de dados:**




Sign-extension unit




Registers




ALU

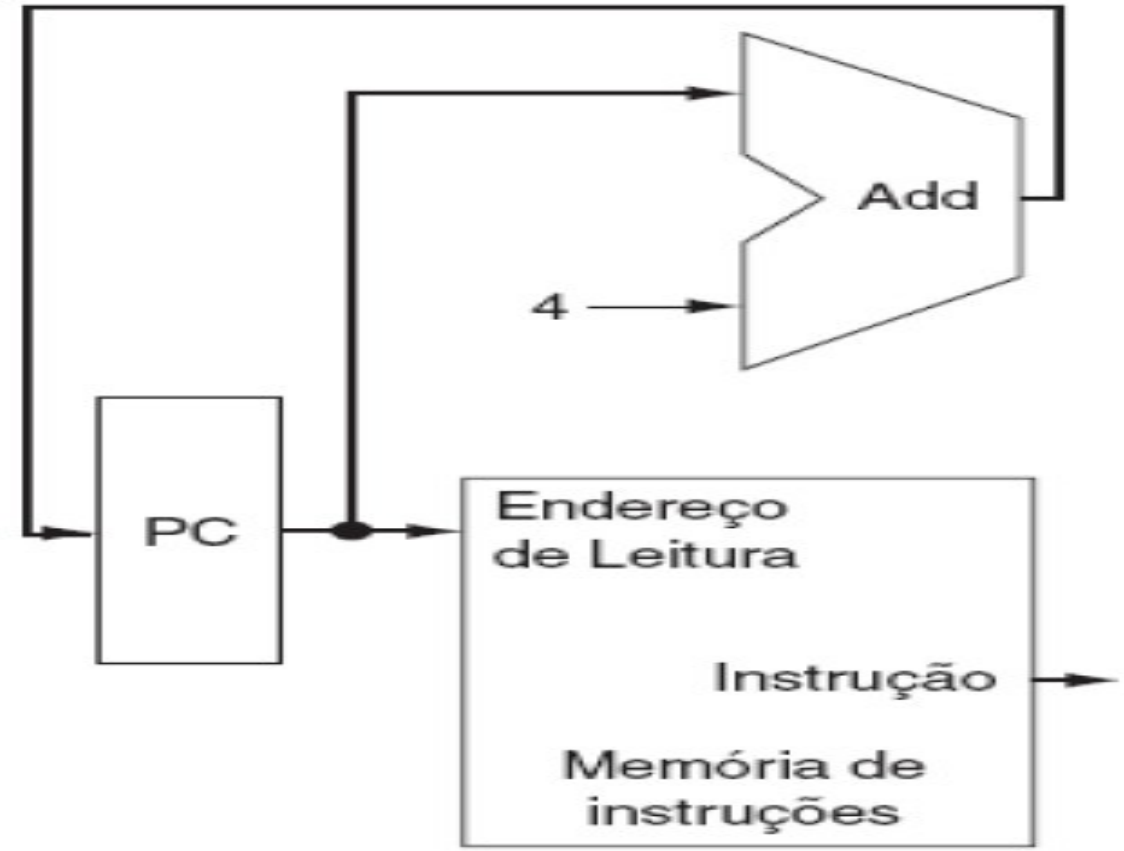
# Construindo um Caminho de Dados

- A **memória de Instruções** (*instruction memory*) armazena as instruções de um programa e fornece instruções dado um endereço.
- O **contador de programa** (*program* counter – PC) é usado para conter o endereço da instrução atual.
- O **somador** (*Adder*) é utilizado para incrementar o PC para o endereço da próxima instrução, é um somador combinatório.

# Construindo um Caminho de Dados

- Para executar qualquer instrução, precisamos começar buscando a instrução na memória de instruções. Para preparar para executar a próxima instrução, também temos de incrementar o contador de programa de modo que aponte para a próxima instrução.

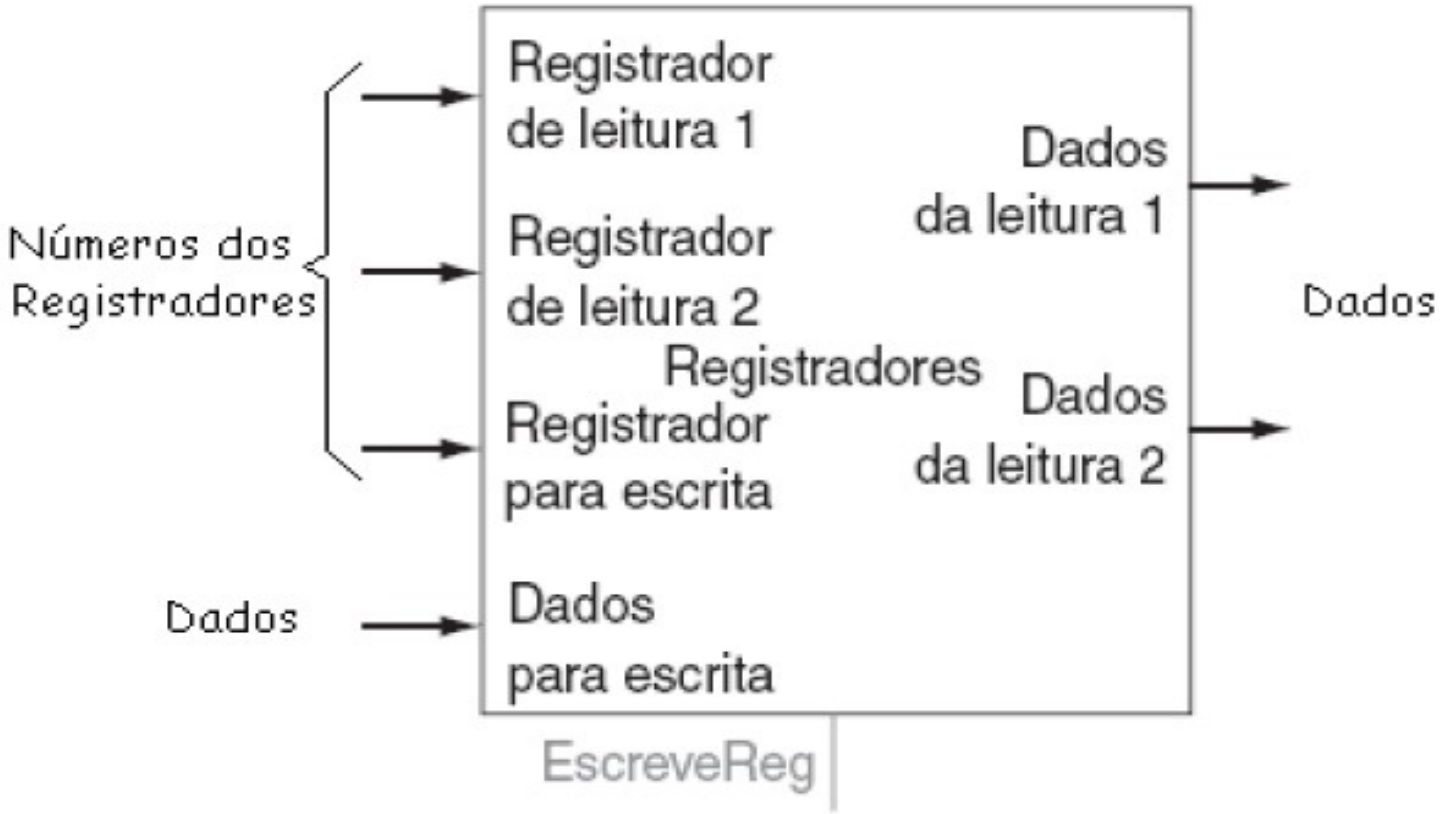
# Construindo um Caminho de Dados



# Construindo um Caminho de Dados

- **Banco de registradores:**
  - Armazena os registradores de uso geral de 32 bits do processador;
  - Os registradores são identificados por um número e podem ser lidos ou escritos utilizando-o;
  - Contém o estado dos registradores da máquina que poderá ser passado para a ULA.

# Construindo um Caminho de Dados



# Construindo um Caminho de Dados

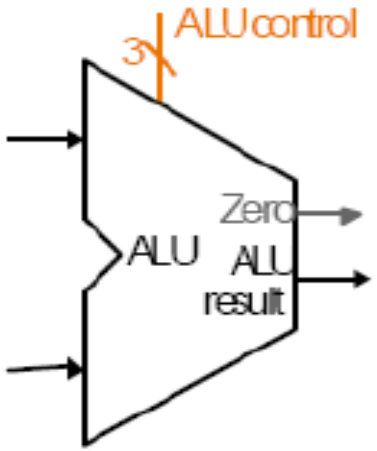
- Uma instrução do formato R lê dois registradores, realiza uma operação na ULA com o conteúdo dos registradores e escreve o resultado em um terceiro registrador.
- Devido às instruções de formato R terem três operandos de registrador, precisaremos ler duas palavras de dados do banco de registradores e escrever uma palavra de dados no banco de registradores para cada instrução.

# Construindo um Caminho de Dados

- Para ler cada palavra de dados precisamos de uma entrada no banco de registradores que especifique o número do registrador a ser lido e uma saída que conduzirá o valor lido dos registradores.
- Para escrever uma palavra de dados, precisaremos de duas entradas:
  - Uma especificando o **número do registrador** a ser escrito;
  - Uma com os **dados** a serem escritos no registrador.
- **As escritas, no entanto, são controladas pelo sinal de controle de escrita, que precisa ser ativo para que uma escrita ocorra na transição do *clock*.**

# Construindo um Caminho de Dados

- ULA:
  - Usa duas entradas de 32 bits e produz um resultado de 32 bits, bem como um sinal de 1 bit se o resultado for 0;
  - A ULA possui um entrada de 4 bits de controle (ALU *control*) para determinar a operação.




ALU

# Construindo um Caminho de Dados

- Unidade de extensão de sinal:
  - Utilizada para estender o sinal de um valor de 16 bits para 32 bits;
  - Útil para quando for ler ou escrever na memória utilizando instruções que somam um valor de deslocamento (*offset*) ao endereço base armazenado em um registrador (ex: lw e sw).