UFRRJ
UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

# Arquitetura de Comptuadores II

## O Processador:
## Caminho de Dados e Controle

Marcelo Zamith
zamith.marcelo@gmail.com

# Introdução

- O desempenho de um computador é determinado por três fatores principais:
  - Número de instruções executadas
  - Período do *clock*
  - Número de ciclos gastos por instrução

# Introdução

- O compilador e o conjunto de instruções determinam o número de instruções necessário para a execução de um determinado programa

- O período do *clock* e o número de ciclos do *clock* por instrução são determinados pela implementação do processador

# Introdução

- Este capítulo descreve como construir o caminho de dados e a unidade de controle para duas implementações diferentes do conjunto de instruções do processador MIPS

# Introdução

- Um processador é organizado em duas unidades:
  - Seção de Processamento (caminho de dados)
  - Seção de Controle

# Introdução

- Conjunto de instruções do processador MIPS:
  - Acesso à memória: lw e sw
  - Lógica e aritmética: add, sub, and, or e slt
  - Desvio: beq e j

# Introdução

- Este subconjunto de instruções é considerado representativo dos princípios fundamentais de um projeto de um caminho de dados e de uma unidade de controle
  - Qualquer conjunto de instruções possui pelo menos estas três classes de instruções: as de acesso à memória, as lógicas e aritméticas, e as de transferência de controle

# Visão Geral da Implementação

- Parte do que é necessário ser realizado para implementar as instruções que tratam de números inteiros, de acesso à memória e de desvio não depende da classe de instruções que está sendo implementada

- Para qualquer instrução é necessário primeiro buscá-la na memória

# Visão Geral da Implementação

- Depois de buscar a instrução na memória, pode ser necessário usar os campos da instrução para selecionar os registradores a serem lidos

- Após estes passos, as ações realizadas para completar a execução dependem da classe de instruções

# Visão Geral da Implementação

- Mesmo entre classes de instruções diferentes existem algumas semelhanças

- Todas as instruções, exceto jump, usam a ALU após a leitura dos registradores
  - Acesso à memória efetuam o cálculo do endereço
  - Lógica e aritmética executam a operação
  - Desvios condicionais efetuam comparações

# Visão Geral da Implementação

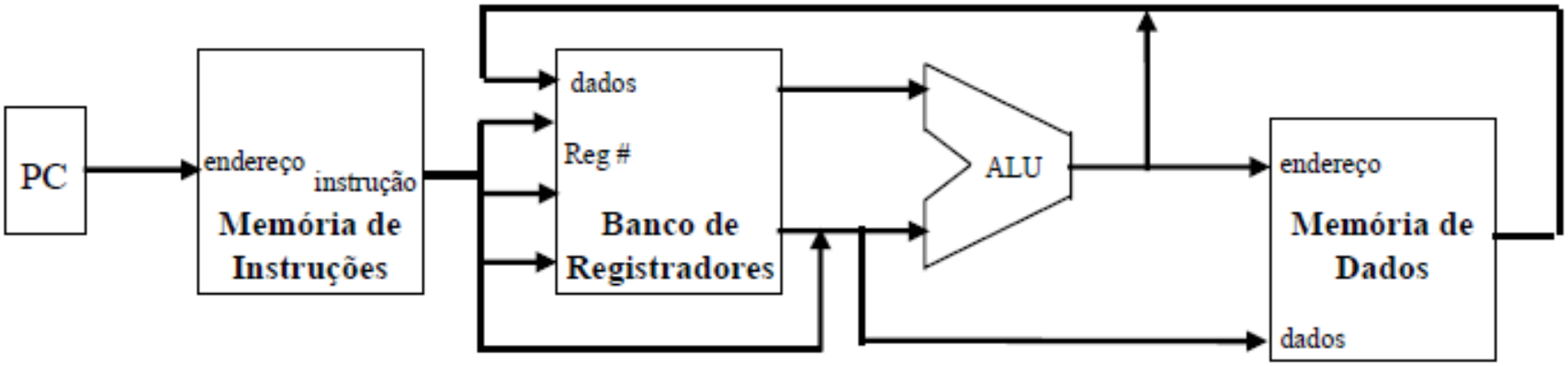
- A simplicidade e a regularidade do conjunto de instruções simplifica a implementação
  - Torna semelhante a execução de muitas das classes de instruções

# Visão Geral da Implementação

- Após usar a ALU, as ações necessárias para completar a execução das instruções de cada classe são diferentes
  - Referência a memória precisa realizar o acesso à memória
  - Lógica e aritmética precisa armazenar o resultado produzido pela ALU num registrador
  - Desvio pode ser necessário modificar o endereço da próxima instrução a ser buscada da memória

# Visão Geral da Implementação

- Visão abstrata de uma implementação




- Visão refinada
  - Novas unidades funcionais
  - Conexões entre as unidades
  - Unidade de controle

# Convenções da Lógica e *Clock*

- No projeto de uma máquina é necessário decidir como a implementação lógica da máquina deve operar e como será o esquema do *clock* usado

- Sinal de controle está ativo quando ele está no nível alto (1 lógico)

# Convenções da Lógica e *Clock*

- As unidades funcionais implementadas no processador MIPS são construídas a partir de componentes lógicos combinacionais e sequenciais

- Metodologia de temporização é **sensível às transições** do sinal de *clock*
  - Transições positivas (nível baixo para alto)

# Convenções da Lógica e *Clock*

- No processador MIPS a maioria dos elementos combinacionais e de estado têm entradas/saídas de 32 bits. A maioria dos dados e todas as instruções têm tamanho de 32 bits

# Convenções da Lógica e *Clock*

- Duas implementações diferentes do subconjunto de instruções
  - Projeto Monociclo: único ciclo de *clock* que é grande o suficiente para atender todas as instruções consideradas
  - Projeto Multiciclo: cada instrução possui um tempo de execução que varia de acordo com a quantidade de ciclos de *clock* necessários a cada uma delas

# Construção do Caminho de Dados

- Examinar os componentes utilizados na execução de cada uma das classes de instruções

- Mostrar os sinais de controle associados aos componentes

# Construção do Caminho de Dados

- A construção do caminho de dados terá como base os seguintes passos de execução de uma instrução: **busca**, **execução** e **resultado**

- O passo de **decodificação** de uma instrução pertence à seção de controle de um processador

# Construção do Caminho de Dados

- O primeiro passo de uma instrução é o passo de **busca**, que é comum a qualquer instrução

- Componentes:
  - Memória de Instruções (MI)
  - *Program Counter* (PC)
  - Somador (SO)

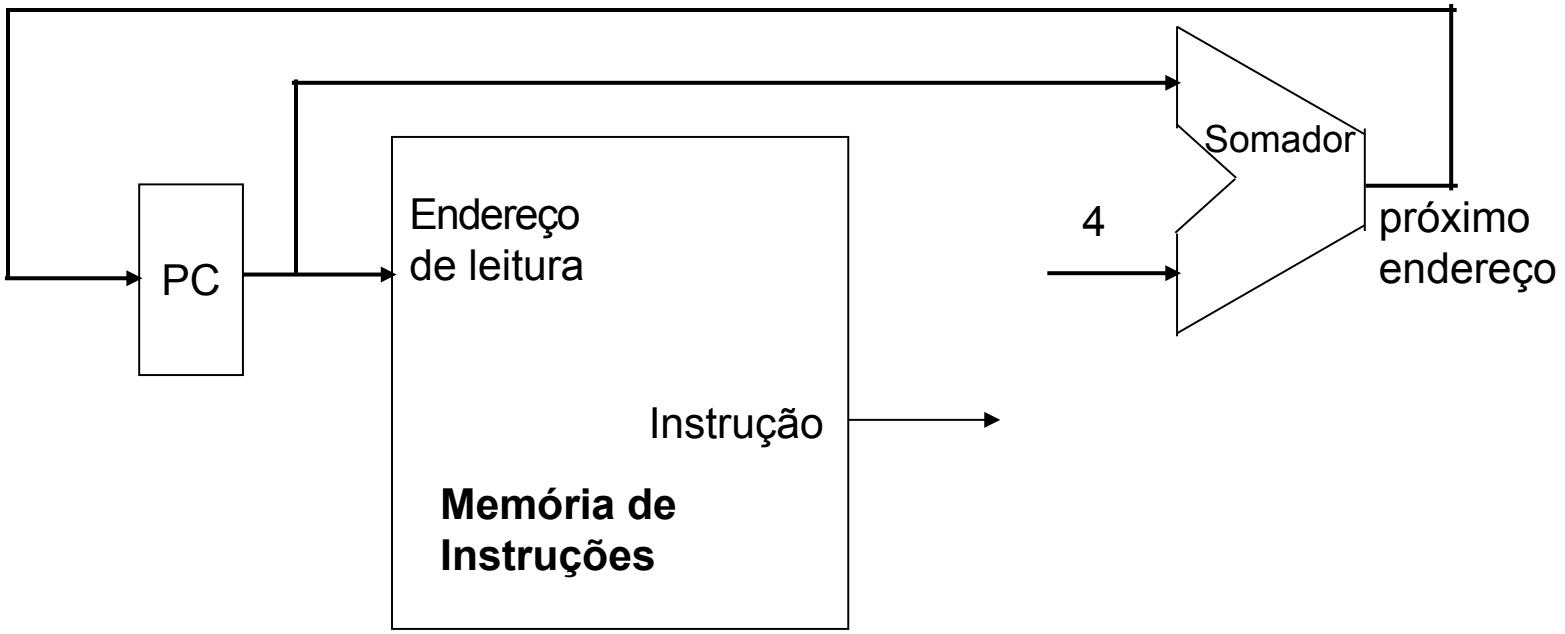
# Construção do Caminho de Dados

- O PC é um registrador de 32 bits que é atualizado ao final de cada ciclo de *clock*

- A tarefa do somador é incrementar o valor do PC. Pode ser visto como uma ALU que só realiza operação de soma
  - A próxima instrução está armazenada 4 bytes depois do valor corrente do PC

# Construção do Caminho de Dados

- Busca



# Instruções Lógicas e Aritméticas

- Execução de Instruções Lógicas e Aritméticas
  - Ler dois registradores fontes
  - Realizar uma operação na ALU com o conteúdo dos registradores fontes
  - Escrever o resultado em um registrador destino

- Esta classe de instruções inclui operações como **add**, **sub**, **slt**, **and**, **or**

# Instruções Lógicas e Aritméticas

- Componentes
  - Banco de Registradores (BR)
  - ALU

- A escrita em um registrador precisa ser indicada explicitamente com o sinal de controle de escrita (**EscReg**)
  - O número do registrador, o valor a ser escrito e o sinal de controle de escrita precisam estar válidos nas transições ativas do *clock*

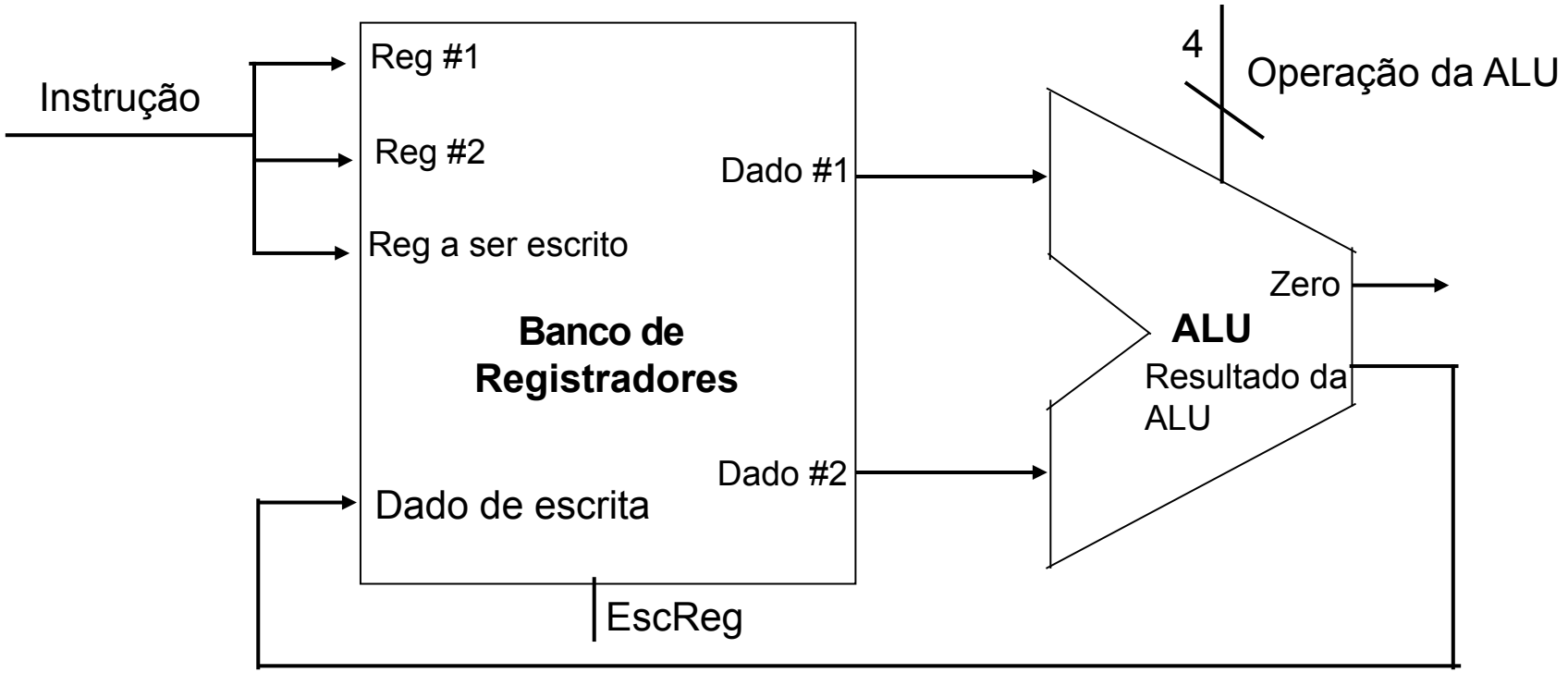
# Instruções Lógicas e Aritméticas

- A operação realizada pela ALU é controlada por um sinal que contém 4 bits

- A saída da ALU para detecção do valor zero é usada para a implementação de desvios condicionais

# Instruções Lógicas e Aritméticas

- Execução de instruções lógicas e aritméticas



# Instruções de Acesso à Memória

- Instruções de *load word* e *store word*
  - **lw $t1, deslocamento($t2)**
  - **sw $t1, deslocamento($t2)**
  - Calcula o endereço de memória somando o registrador base ($t2) ao número de 16 bits com sinal
  - Para lw: o valor lido da memória de dados é escrito no registrador $t1
  - Para sw: o valor a ser armazenado na memória de dados é lido do registrador $t1

# Instruções de Acesso à Memória

- Componentes
  - Banco de Registradores (BR)
  - ALU
  - Memória de Dados (MD)
  - Extensão de Sinal (Ext)

- A memória de dados possui dois sinais para controlar a leitura (**LerMem**) e a escrita de dados (**EscMem**)

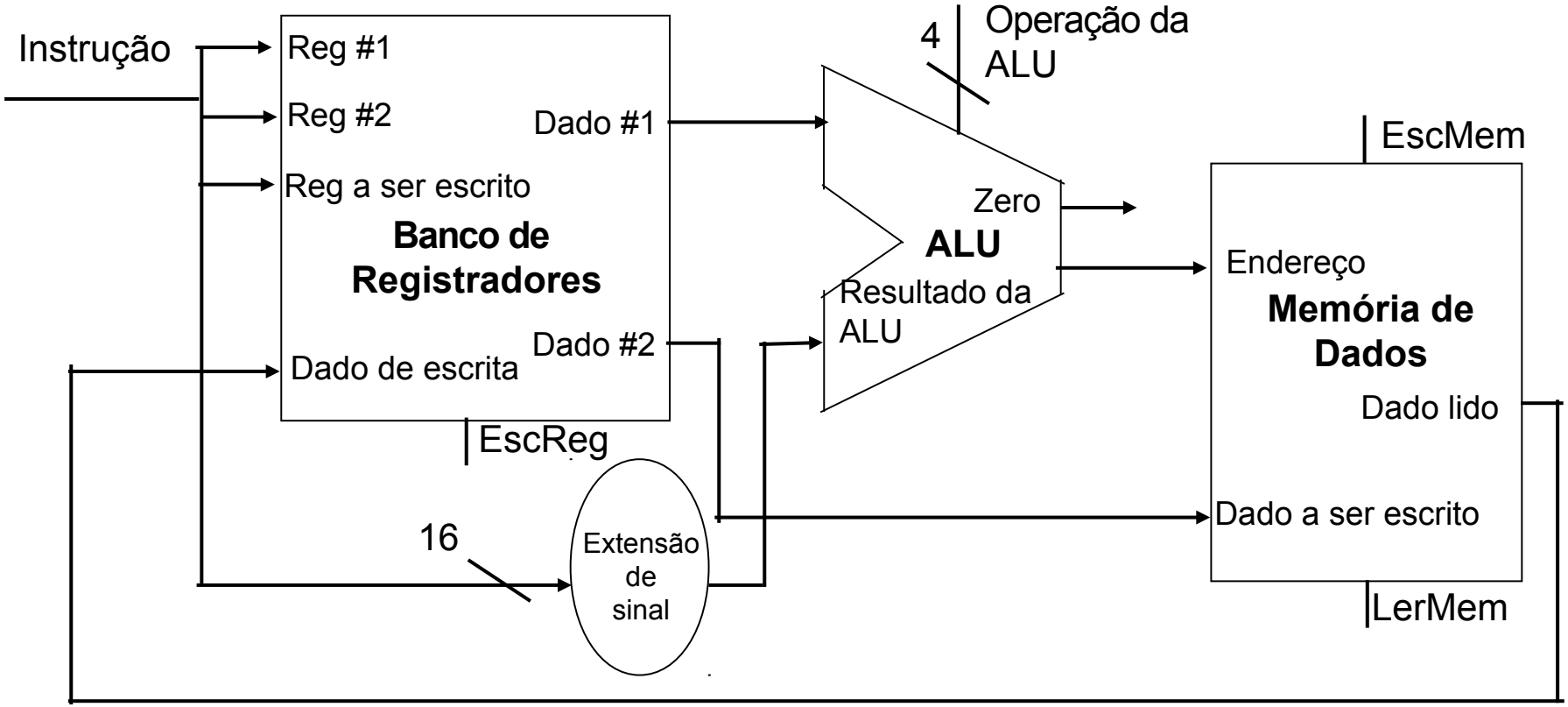
# Instruções de Acesso à Memória

- A unidade de extensão de sinal tem uma entrada de 16 bits que, após a extensão do sinal, transforma-se num resultado de 32 bits que é disponibilizado na sua saída

- O valor especificado no campo de deslocamento após ter seu sinal estendido torna-se a segunda entrada da ALU

# Instruções de Acesso à Memória

- Execução de instruções de *load* e *store*



# Instruções de Desvio

- Instruções de desvio condicional
  - Por exemplo: **beq $t1, $t2, label**
  - Dois registradores cujos conteúdos são comparados
  - Um deslocamento de 16 bits usado no cálculo do endereço alvo do desvio

# Instruções de Desvio

- Instruções de desvio condicional
  - A arquitetura do conjunto de instruções estabelece que a base para o cálculo do endereço alvo do desvio é igual ao valor do PC atualizado (PC + 4 no passo de busca)
  - A arquitetura também define que o campo de deslocamento deve ser deslocado de 2 bits à esquerda. Isto significa um deslocamento relativo à palavra do processador

# Instruções de Desvio

- Instruções de desvio condicional
  - Quando a condição é verdadeira o endereço alvo do desvio calculado deve ser armazenado no PC
  - Caso contrário, o valor do PC, incrementado no passo de busca, não deve ser substituído

# Instruções de Desvio

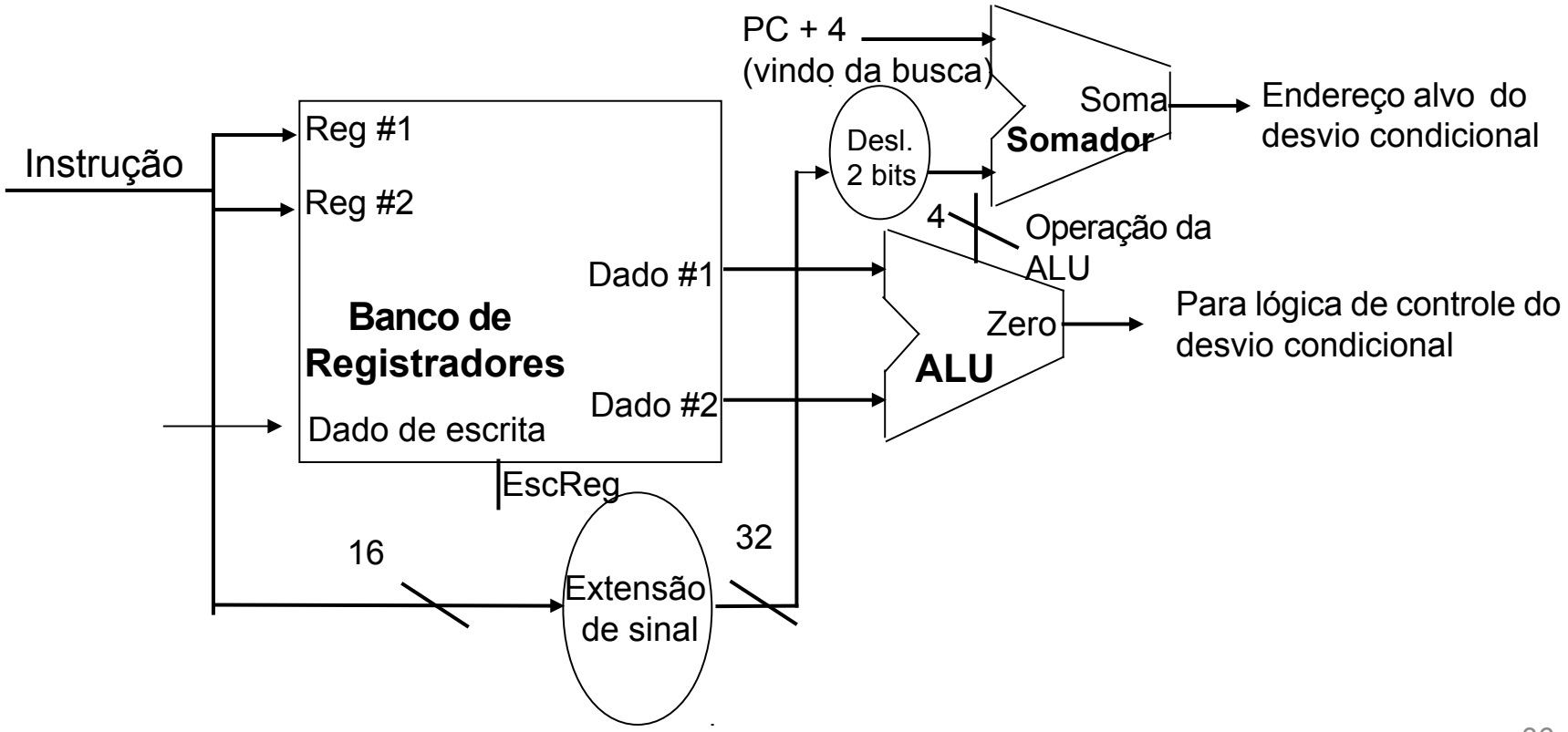
- Componentes das Instruções de Desvio Condicional
  - Banco de Registradores (BR)
  - ALU
  - Extensão de Sinal (Ext)
  - Somador (SO)
  - Deslocamento de 2 bits (D2)

# Instruções de Desvio

- Instruções de Desvio Condicional
  - A ALU possui um sinal de saída indicando se o resultado calculado é ou não igual a zero
  - Um sinal de controle pode ser enviado para a ALU indicando a realização de uma subtração, por exemplo, para a condição de desvio da instrução **beq**. Se o sinal **zero** da ALU estiver ativo os dois valores são iguais

# Instruções de Desvio

- Execução de Instruções de Desvio Condicional

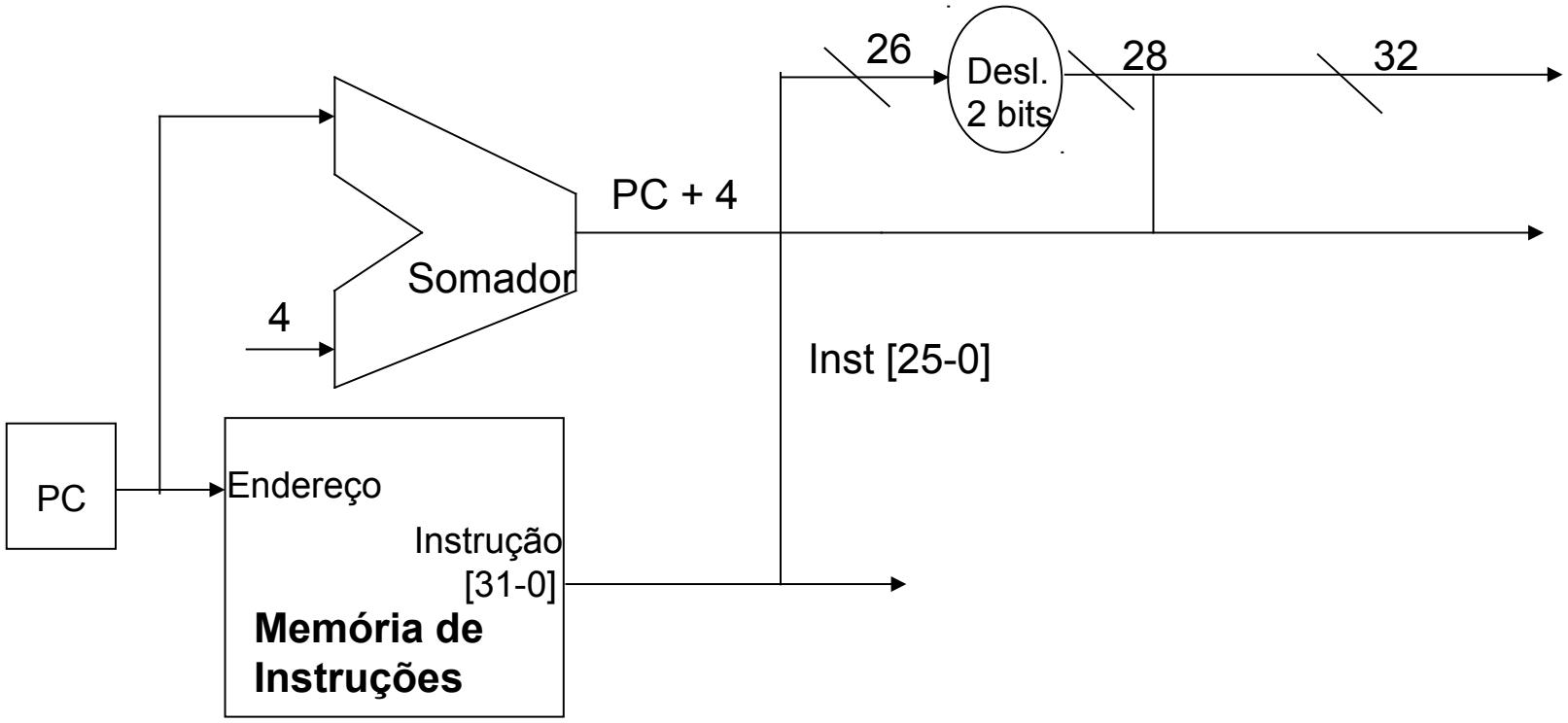


# Instruções de Desvio

- Instruções de desvio incondicional
  - Por exemplo: **j label**
  - Endereço alvo do desvio
    - PC atual (PC + 4), 4 bits mais significativos
    - 28 bits menos significativos
      (26 bits da instrução + 2 bits de deslocamento)
  - Componentes
    - Deslocamento de 2 bits (D2)

# Instruções de Desvio

- Execução de Instruções de Desvio Incondicional



# Um Esquema Simples para Implementação

- Construir o caminho de dados completo a partir dos caminhos de dados abordados anteriormente para cada uma das classes de instruções, adicionando linhas de controle se necessário

# Um Esquema Simples para Implementação

- O caminho de dados mais simples se propõe a executar todas as instruções dentro de um único período de *clock*
  - Nenhum dos seus recursos pode ser usado mais de uma vez por instrução. Se for necessário, o recurso deve ser replicado
    - Memória de instruções e de dados
  - **Projeto Monociclo**

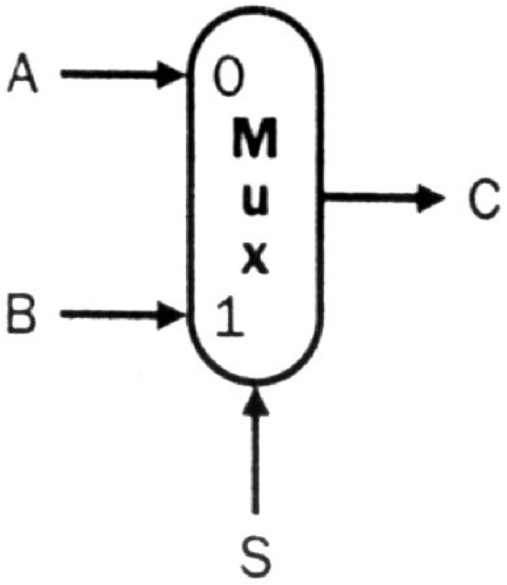
# Um Esquema Simples para Implementação

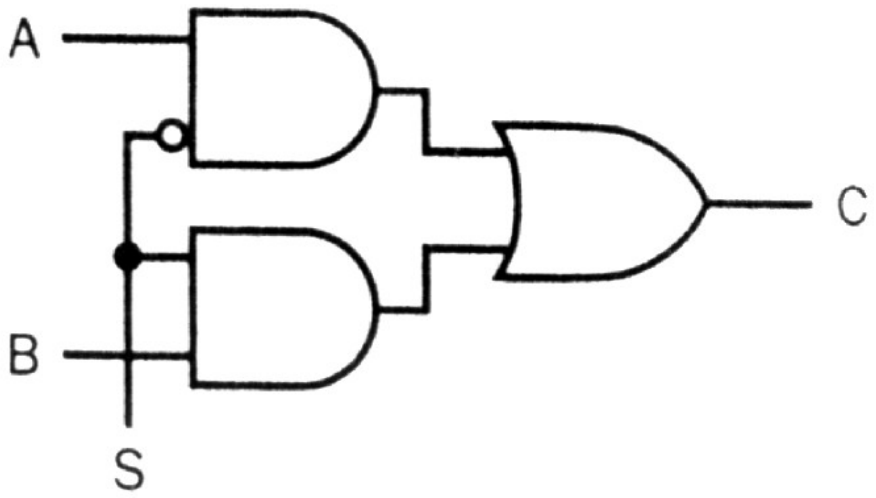
- Elementos podem ser compartilhados em razão dos diferentes fluxos de execução das instruções

- Para isso, deve-se permitir múltiplas conexões para a entrada de um certo elemento, tendo sinais para controlar a seleção entre as entradas
  - Multiplexador (seletor de dados)

# Um Esquema Simples para Implementação

- Multiplexadores 2 x 1



# Exercício

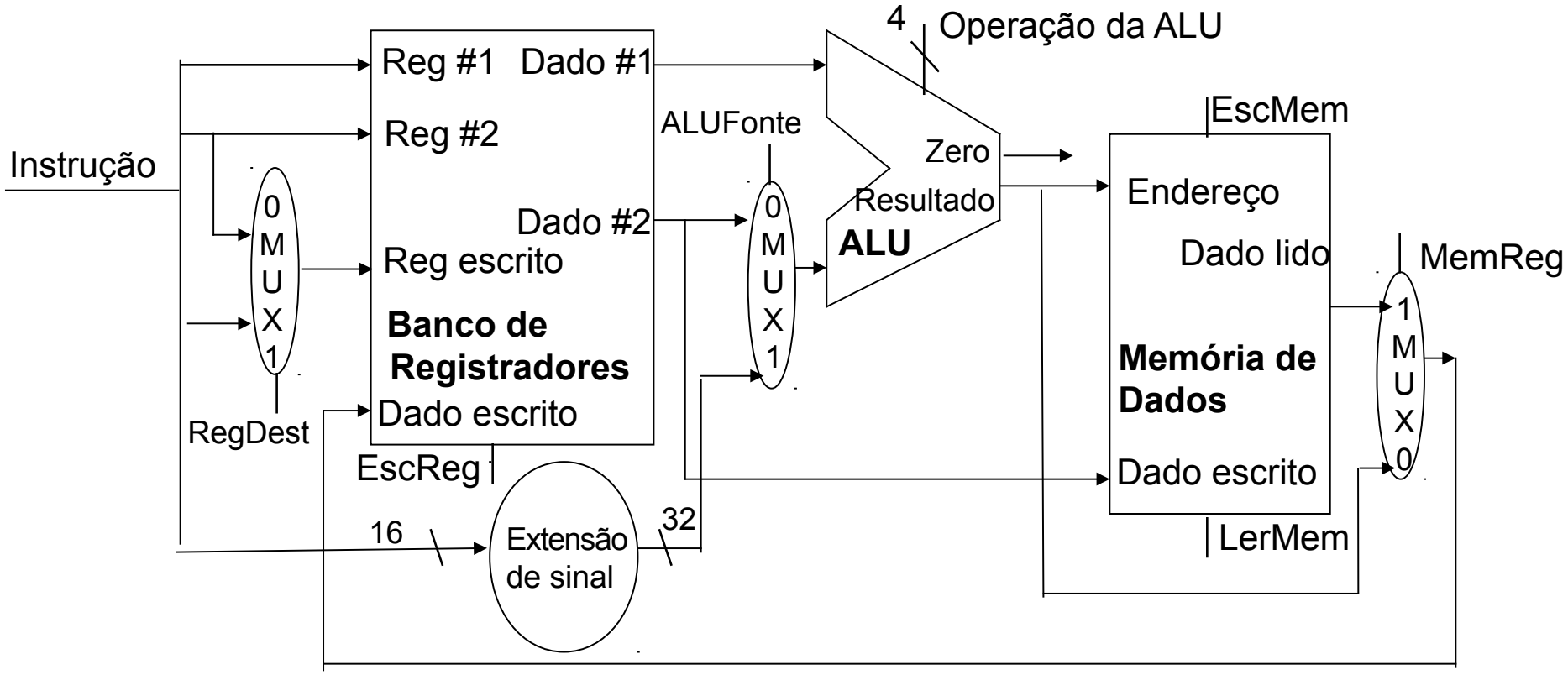
- O caminho de dados para as instruções aritméticas e lógicas e para as instruções de acesso à memória são muito similares. As principais diferenças são:
  - A segunda entrada da ALU corresponde ao conteúdo de um registrador ou corresponde à extensão do sinal do deslocamento
  - O valor armazenado no registrador de destino vem da ALU ou da memória de dados

# Exercício

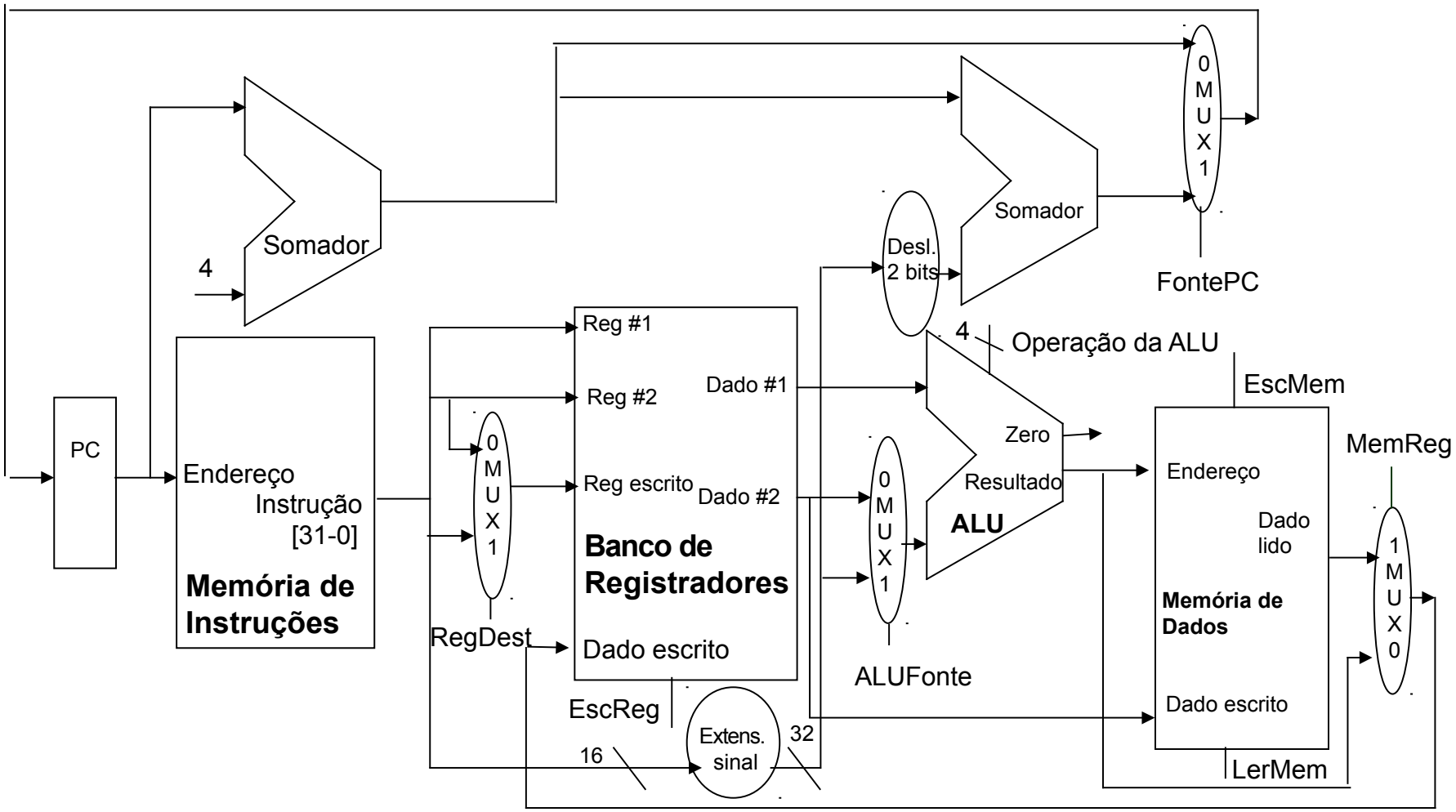
- Mostre como combinar o caminho de dados do passo de execução das instruções lógicas e aritméticas e de acesso à memória. Utilize multiplexadores e não duplique os elementos comuns. Ignore o controle dos multiplexadores

# Solução do Exercício

- Caminho de dados combinando as instruções de acesso à memória e de lógica e aritmética



# Instruções de acesso à memória, lógica e aritmética e desvio condicional



# Unidade de Controle

- Controla o caminho de dados
- A partir dos sinais de entrada gera
  - Sinais de escrita para todos os elementos de estado
  - Sinais seletores para todos os multiplexadores
  - Sinais para o controle das operações da ALU

# Controle da ALU

- O controle da ALU possui 4 entradas
  - Apenas 5 das 16 combinações de entrada possíveis são usadas neste subconjunto de instruções

| Entrada da ALU | Função |
|---|---|
| 0000 | AND |
| 0001 | OR |
| 0010 | soma |
| 0110 | subtração |
| 0111 | *set less than* |

# Controle da ALU

- As instruções *load word* e *store word* usam a ALU para calcular o endereço de memória pela operação de soma
- As instruções do tipo R precisam da ALU para realizar uma das cinco ações (AND, OR, soma, subtração ou *set on less than*) dependendo do valor do campo **funct**
- A instrução *branch equal* precisa da ALU para realizar uma operação de subtração

# Controle da ALU

- Pode-se gerar os 4 bits de controle da ALU usando uma unidade de controle pequena
- Esta unidade recebe como entrada o campo **funct** e um campo de controle de 2 bits, denominado **ALUOp**
- Campo de controle ALUOp

| | |
|---|---|
| 00 | Soma |
| 01 | Subtração |
| 10 | Campo funct |

# Controle da ALU

- Bits de controle da ALU

| Código de operação da instrução | ALUOp | Operação da Instrução | Campo de Função | Operação desejada da ALU | Operação da ALU |
|---|---|---|---|---|---|
| LW | 00 | load word | XXXXXX | soma | 0010 |
| SW | 00 | store word | XXXXXX | soma | 0010 |
| Branch equal | 01 | branch equal | XXXXXX | subtração | 0110 |
| Tipo R | 10 | add | 100000 | soma | 0010 |
| Tipo R | 10 | subtract | 100010 | subtração | 0110 |
| Tipo R | 10 | AND | 100100 | and | 0000 |
| Tipo R | 10 | OR | 100101 | or | 0001 |
| Tipo R | 10 | set on less than | 101010 | set on less than | 0111 |

# Controle da ALU

- ALUOp (2 bits) + campo **funct** (6 bits)
  - Tabela verdade possui $2^8 = 256$ entradas
- Combinações relevantes ao controle da ALU

| ALUOp | | funct | | | | | | Operação |
|---|---|---|---|---|---|---|---|---|
| bit 1 | bit 0 | F5 | F4 | F3 | F2 | F1 | F0 | da ALU |
| 0 | 0 | x | x | x | x | x | x | 0010 |
| x | 1 | x | x | x | x | x | x | 0110 |
| 1 | x | x | x | 0 | 0 | 0 | 0 | 0010 |
| 1 | x | x | x | 0 | 0 | 1 | 0 | 0110 |
| 1 | x | x | x | 0 | 1 | 0 | 0 | 0000 |
| 1 | x | x | x | 0 | 1 | 0 | 1 | 0001 |
| 1 | x | x | x | 1 | 0 | 1 | 0 | 0111 |

# Unidade de Controle

- Controle do caminho de dados
  - Dependendo do formato das instruções, a UC é responsável por gerar os valores para cada um dos sinais (linhas de controle)

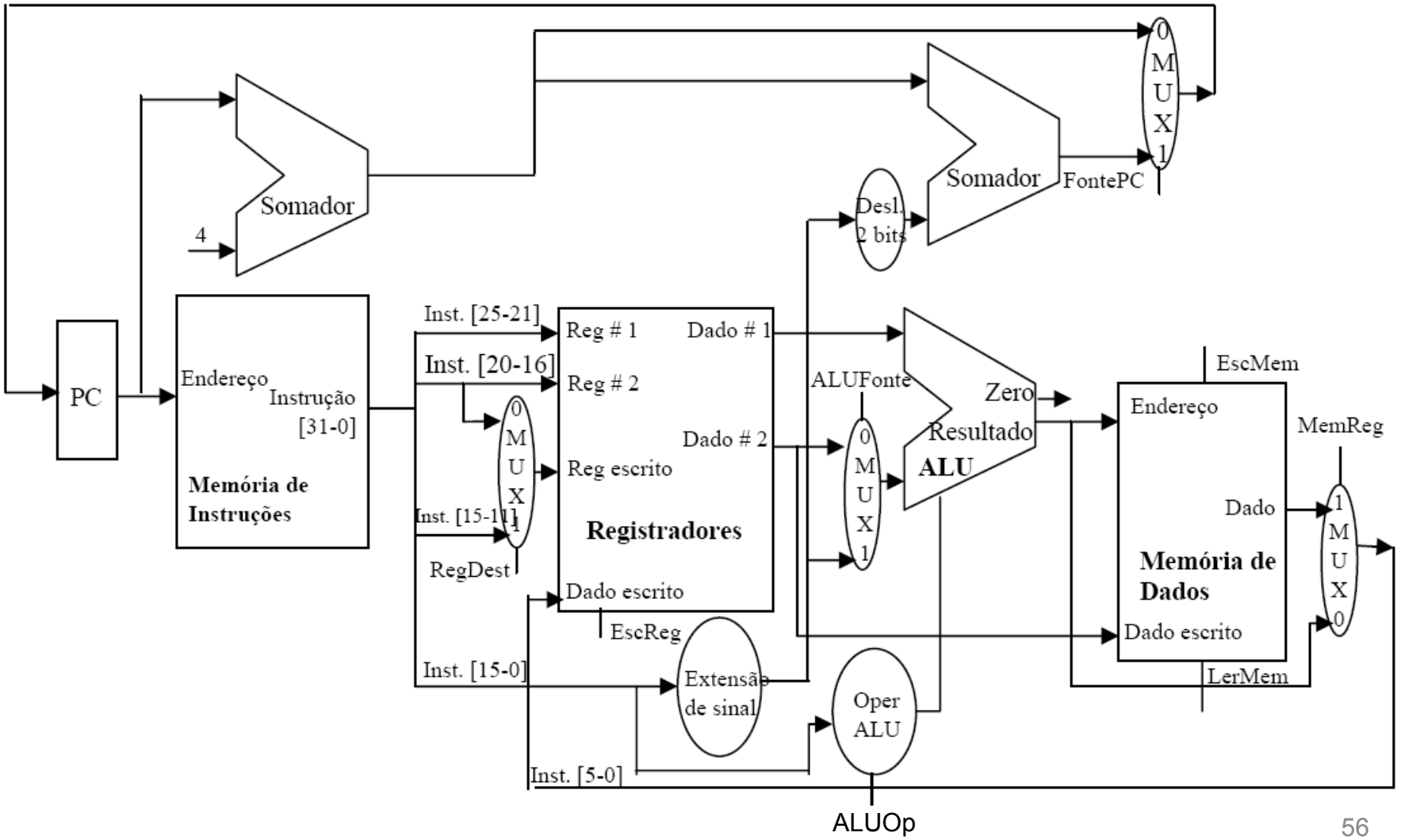
# Formato das Instruções

- O campo op (**opcode**) está sempre contido nos bits 31-26
- Os dois registradores a serem lidos (**rs** e **rt**) estão nas posições 25-21 e 20-16
  - Instruções do tipo R, *branch equal* e *store*
- Registrador base para as instruções *load*e *store* está nas posições de bits 25-21 (**rs**)

# Formato das Instruções

- O deslocamento de 16 bits está sempre nas posições 15-0
  - Instruções *branch equal*, *load* e *store*
- Registrador de destino
  - Para *load* nos bits 20-16 (rt)
  - Para tipo R nos bits 15-11 (rd)
  - Necessário multiplexador para selecionar o campo da instrução a ser usado para indicar o número do registrador a ser escrito

# Caminho de Dados



# Unidade de Controle

- Existem sete linhas de controle de 1 bit cada mais os dois sinais de controle ALUOp

- A unidade de controle pode gerar os valores para todas as linhas de controle com base no campo **opcode**, exceto para a linha de controle **FontePC**

# Unidade de Controle

- Linha de controle **FontePC**
  - O valor dessa linha deve ser 1 se a instrução for **beq** e a saída Zero da ALU for verdadeira
  - Para gerar este sinal de controle é necessário colocar um sinal vindo da unidade de controle, denominado **DvC**, na entrada de uma porta AND, junto com o sinal Zero, presente na saída da ALU

# Sinais de Controle

- RegDest
  - Inativo: O número do registrador destino onde será escrito o resultado da operação (Reg Escrito) vem do campo **rt** (bits 20-16)
  - Ativo: O número do registrador destino onde será escrito o resultado da operação (Reg Escrito) vem do campo **rd** (bits 15-11)

# Sinais de Controle

- EscReg
  - Inativo: Nenhum efeito
  - Ativo: O registrador na entrada **Reg Escrito**é escrito com o valor presente na entrada **Dado Escrito**

# Sinais de Controle

- ALUFonte
  - Inativo: O segundo operando da ALU vem do segundo registrador do banco de registradores (Dado # 2)
  - Ativo: O segundo operando da ALU é o resultado da extensão de sinal de 16 bits menos significativos da instrução

# Sinais de Controle

- FontePC
  - Inativo: O PC é substituído pelo valor presente na saída do somador que calcula PC + 4
  - Ativo: O PC é substituído pelo valor presente na saída do somador que calcula o endereço alvo do desvio condicional

# Sinais de Controle

- LerMem
  - Inativo: Nenhum efeito
  - Ativo: O conteúdo da memória designado pela entrada de endereço é colocado na saída **Dado**

# Sinais de Controle

- EscMem
  - Inativo: Nenhum efeito
  - Ativo: O conteúdo da memória designado pela entrada de endereço é substituído pelo valor presente na entrada **Dado Escrito**

# Sinais de Controle

- MemReg
  - Inativo: O valor na entrada do registrador de escrita vem da ALU
  - Ativo: O valor na entrada do registrador de escrita vem da memória de dados

# Unidade de Controle

- Tabela verdade para a definição dos sinais de saída da UC

| Instrução | RegDest | ALUFonte | MemReg | EscReg | LerMem | Escmem | DvC | ALUOp | |
|---|---|---|---|---|---|---|---|---|---|
| Tipo R | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| lw | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| sw | x | 1 | x | 0 | 0 | 1 | 0 | 0 | 0 |
| beq | x | 0 | x | 0 | 0 | 0 | 1 | 0 | 1 |

# Os quatro passos na execução de uma instrução do Tipo R

- Passos de execução
  - Busca da instrução na memória de instruções e incremento do PC
  - Dois registradores são lidos do banco de registradores. A unidade de controle coloca valores nas linhas de controle
  - ALU opera sobre os dados lidos do banco de registradores, usando o código da função (bits 5-0 do campo **funct**) para gerar a função da ALU
  - O resultado da ALU é escrito no banco de registradores usando-se os bits 15-11 da instrução para selecionar o registrador destino

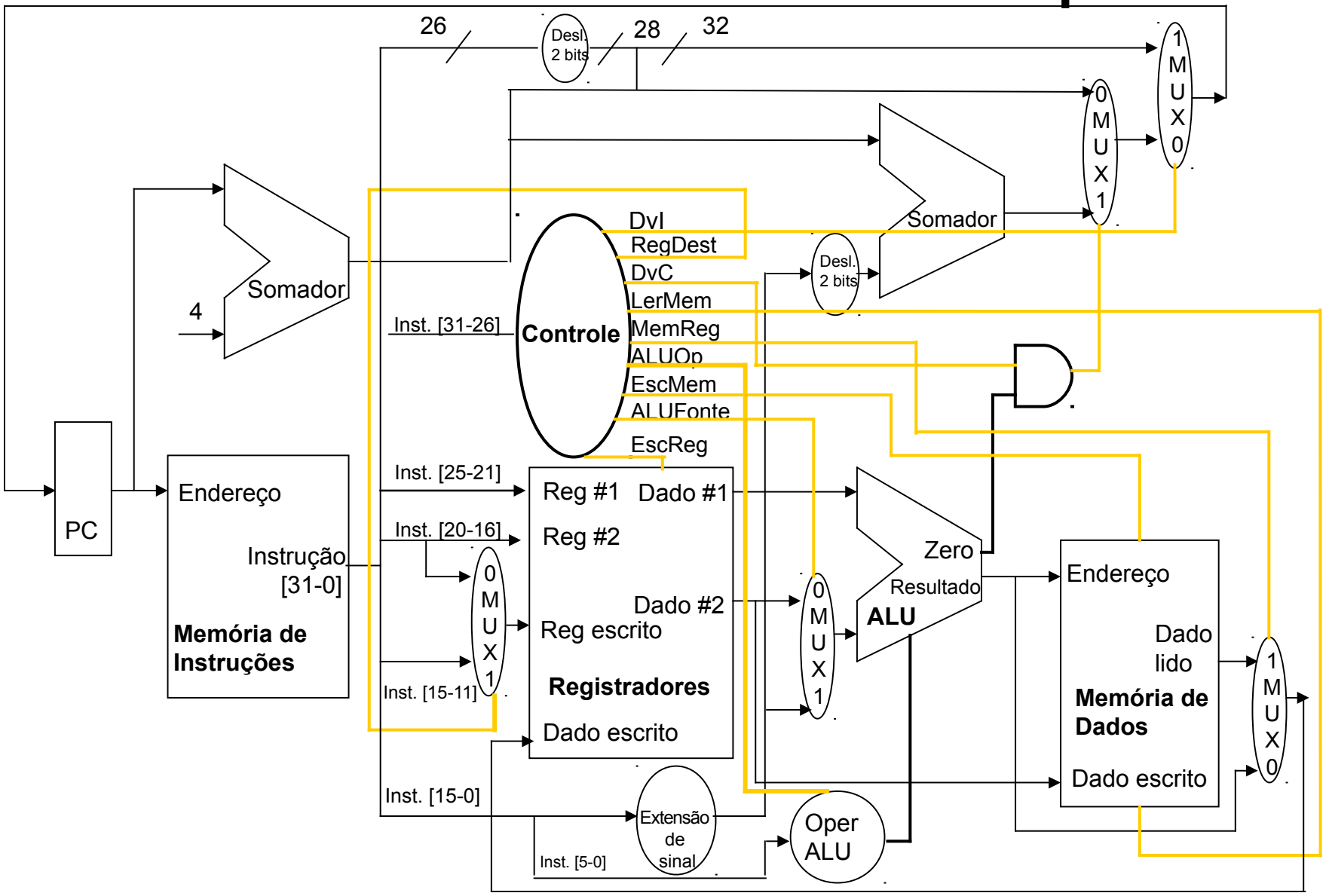
# Os cinco passos na execução de uma instrução *load word*

- Instrução **lw $t1, deslocamento($t2)**
  - Busca da instrução na memória de instruções e incremento do PC
  - Leitura do conteúdo de um registrador ($t2) do banco de registradores
  - Cálculo da soma do valor lido do banco de registradores com o resultado da extensão do sinal de 16 bits menos significativos da instrução (deslocamento)
  - O resultado da soma é usado para endereçar a memória de dados
  - O dado vindo da unidade de memória é escrito no banco de registradores; o número do registrador destino é dado pelos bits 20-16 da instrução ($t1)

# Os quatro passos na execução de uma instrução *branch equal*

- Instrução **beq $t1, $t2, deslocamento**
  - Busca da instrução na memória de instruções e incremento do PC
  - Leitura do conteúdo dos registradores $t1 e $t2
  - Realização de uma subtração pela ALU sobre os dois valores lidos do bando de registradores. O valor de PC+4 é somado ao resultado da extensão do sinal dos 16 bits menos significativos da instrução (deslocamento) deslocado de dois bits à esquerda. O resultado dessa soma é o endereço alvo do desvio
  - O sinal Zero da ALU é usado para decidir se o PC deve ser atualizado com o valor de PC+4 ou com o valor do endereço alvo do desvio condicional

# Caminho de Dados Completo



# Projeto Monociclo

- O projeto monociclo abordado possui um funcionamento correto e todas as instruções são executadas dentro de um único ciclo de *clock*.

- O problema com este projeto é a sua ineficiência

# Projeto Monociclo

- O tamanho do ciclo deve ser grande o suficiente para suportar o maior atraso relativo aos componentes utilizados por uma determinada instrução
  - Em geral, as instruções de acesso à memória possuem o maior tempo de execução

# Projeto Monociclo

- Tempo de execução de instruções X unidades funcionais utilizadas

| Classe | Unidades funcionais utilizadas | | | | | Tempo |
|---|---|---|---|---|---|---|
| Tipo R | Busca (2) | Acesso a reg (1) | ALU (2) | Acesso a reg (1) | | 6 ns |
| Load word | Busca (2) | Acesso a reg (1) | ALU (2) | Acesso a mem (2) | Acesso a reg (1) | 8 ns |
| Store word | Busca (2) | Acesso a reg (1) | ALU (2) | Acesso a mem (2) | | 7 ns |
| Branch | Busca (2) | Acesso a reg (1) | ALU (2) | | | 5 ns |
| Jump | Busca (2) | | | | | 2 ns |

# Projeto Monociclo

- O impacto da filosofia monociclo no desempenho pode ser desastroso
  - Depende da porcentagem de cada classe de instruções na execução de um programa

- As máquinas atuais não são monociclo

# Projeto Monociclo

- O projeto monociclo pressupõe que o ciclo de *clock* deve ser igual ao do pior caso
- Na implementação monociclo cada unidade funcional só pode ser utilizada uma vez a cada ciclo de *clock*
  - Duplicação de algumas unidades funcionais
  - Aumento no custo de implementação

# Projeto Multiciclo

- Para solucionar os problemas relacionados ao desempenho e ao custo do hardware, pode-se usar técnicas de implementação que tenham um ciclo de *clock* menor

- Estes ciclos menores podem ser obtidos a partir dos retardos das unidades funcionais básicas

# Projeto Multiciclo

- Vários ciclos de *clock* são necessários para a execução de cada instrução

- Este esquema alternativo de implementação é denominado **multiciclo**

# Implementação Multiciclo

- A execução de cada instrução pode ser dividida em uma série de etapas que correspondem às operações das unidades funcionais envolvidas
- Estas etapas podem ser usadas para criar uma implementação **multiciclo**
- Neste tipo de implementação, cada etapa de execução gasta um ciclo de *clock*

# Implementação Multiciclo

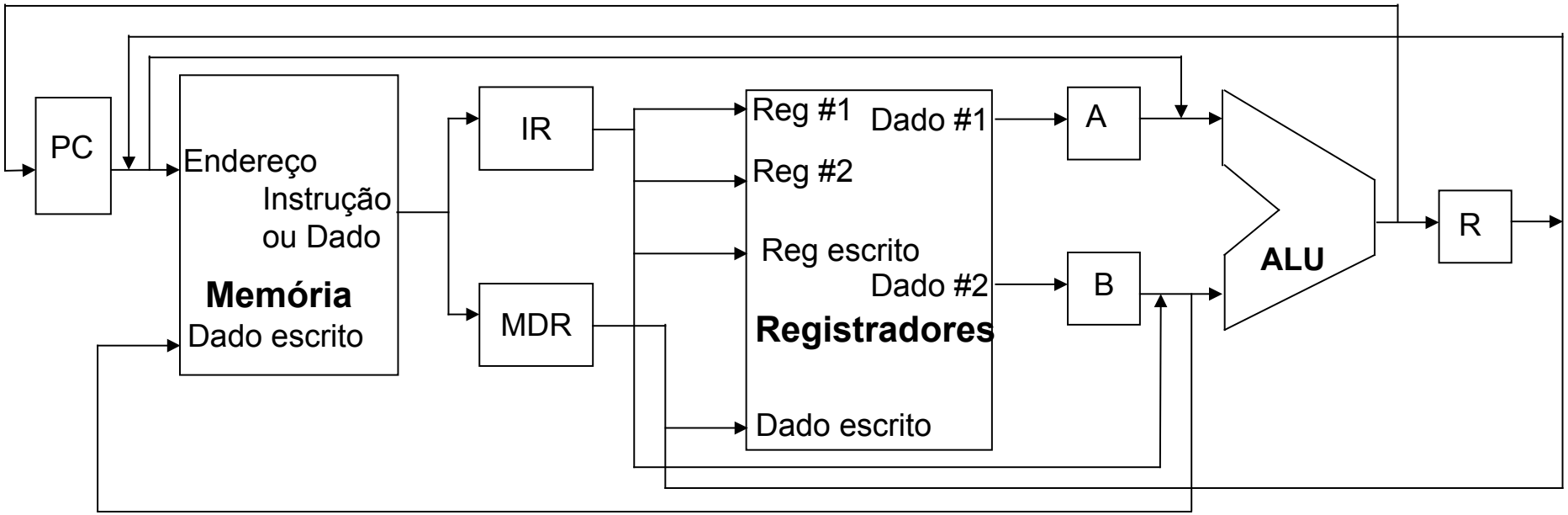
- A implementação multiciclo permite que uma unidade funcional seja utilizada mais de uma vez por instrução. Este fato ocorre porque a unidade funcional está sendo usada em diferentes ciclos de *clock*
  - Reduz a quantidade de hardware necessário à implementação por causa do compartilhamento

# Implementação Multiciclo

- As principais vantagens da implementação multiciclo são:
  - Executar instruções em quantidades diferentes de ciclos de *clock*
  - Compartilhamento de unidades funcionais dentro da execução de uma única instrução

# Caminho de Dados Multiciclo

- Visão abstrata do caminho de dados



# Monociclo X Multiciclo

- Principais diferenças:
  - Uso de uma única memória tanto para instruções quanto para dados
  - Existência de uma única ALU
  - Colocação de um ou mais registradores depois de cada unidade funcional para armazenar temporariamente o resultado calculado até que seja utilizado em um ciclo de *clock* subsequente

# Monociclo X Multiciclo

- No final de um ciclo de *clock,* todos os dados que precisam ser usados em ciclos subsequentes devem ser armazenados em um elemento de estado
- Os dados a serem usados em **outras instruções** devem ser armazenados em um dos elementos de estado visíveis ao programador:
  - Banco de registradores, PC ou memória

# Monociclo X Multiciclo

- Os dados usados pela **mesma instrução** em um ciclo posterior precisam ser armazenados nos registradores adicionais
- A posição dos registradores adicionais é determinada por dois fatores:
  - Quais as unidades funcionais cujo uso ficará restrito a um único ciclo de *clock*
  - Quais dados serão usados em ciclos de *clock* posteriores na execução da instrução

# Projeto Multiciclo

- Nesse projeto multiciclo, considera-se que o ciclo de *clock* pode acomodar no máximo uma das seguintes operações:
  - Um acesso à memória
  - Um acesso ao banco de registradores (duas leituras ou uma escrita)
  - Uma operação da ALU
- Qualquer dado produzido por estas unidades funcionais deve ser armazenado em registradores (IR, MDR, A, B, e R)

# Projeto Multiciclo

- O IR necessita de um sinal de controle de escrita para armazenar a instrução durante todo o tempo de sua execução
- Os demais registradores adicionais não precisam de nenhum sinal de controle porque guardam dados somente entre dois ciclos de *clock* adjacentes

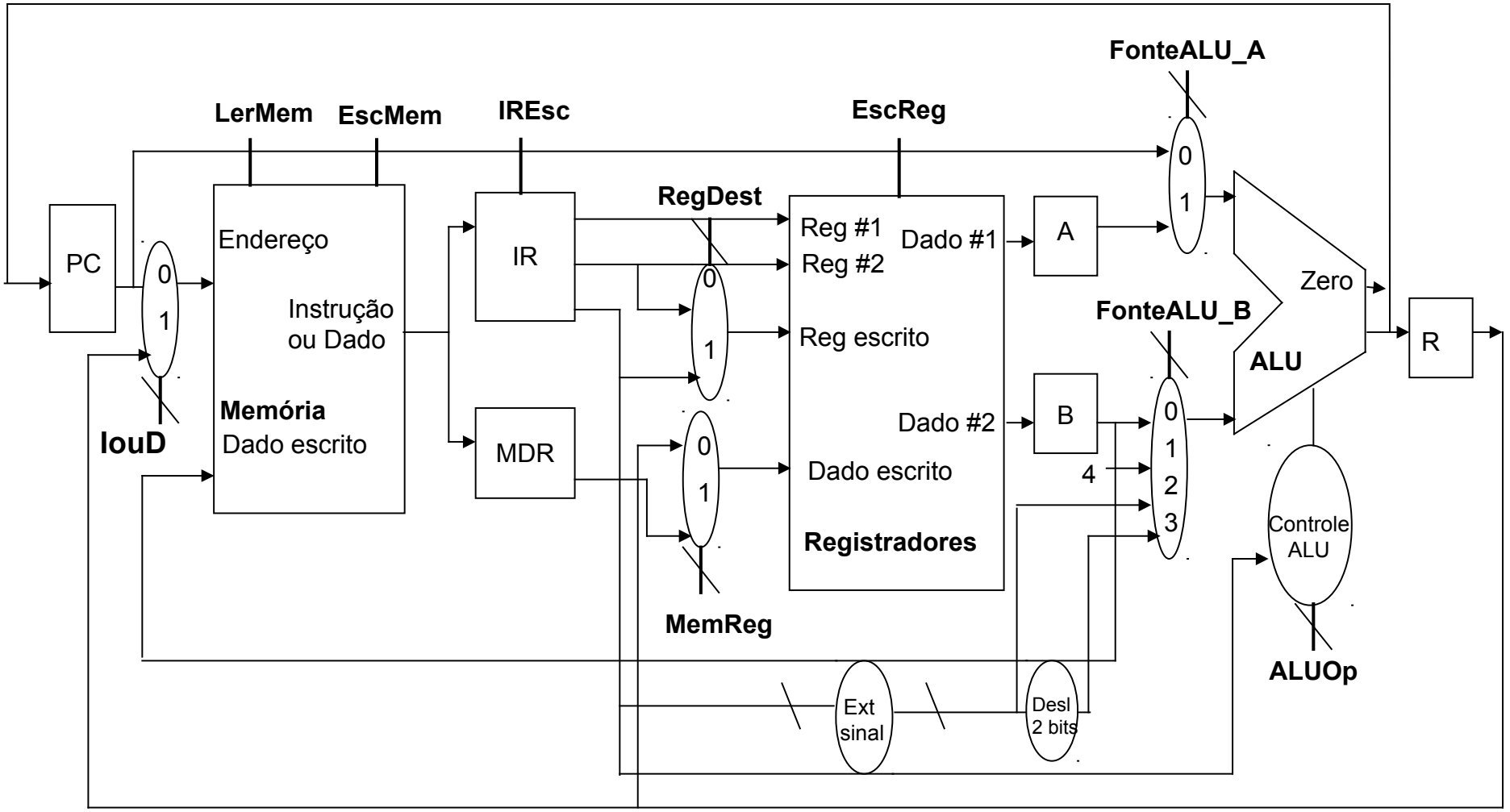
# Projeto Multiciclo

- Compartilhamento das unidades funcionais
  - Adicionar novos multiplexadores ou expandir os já existentes
- Memória precisa de um multiplexador (2x1) para selecionar uma das duas fontes de endereços de memória
  - PC (instrução)
  - Registrador (dado)

# Projeto Multiciclo

- ALU precisa de dois multiplexadores
  - Primeira entrada (2x1)
    - PC
    - Registrador A
  - Segunda entrada (4x1)
    - Registrador B
    - Constante 4
    - Valor resultante da extensão de sinal
    - Valor para o cálculo do endereço alvo de desvio condicional

# Caminho de Dados Multiciclo



# Projeto Multiciclo

- Considerando que os multiplexadores e os registradores causam pouco impacto se comparados com a memória e somadores, estas modificações contribuem para uma redução significativa tanto no tamanho quanto no custo do hardware

# Projeto Multiciclo

- Suporte para desvios condicionais e incondicionais são necessários 3 fontes para o PC:
  - ALU produz PC + 4 no passo de busca
  - Registrador R armazena o endereço alvo de desvio condicional calculado pela ALU
  - Os 26 últimos bits do IR, deslocados de 2 bits à direita e concatenados com os 4 bits superiores do PC + 4, formam o endereço alvo de desvio incondicional

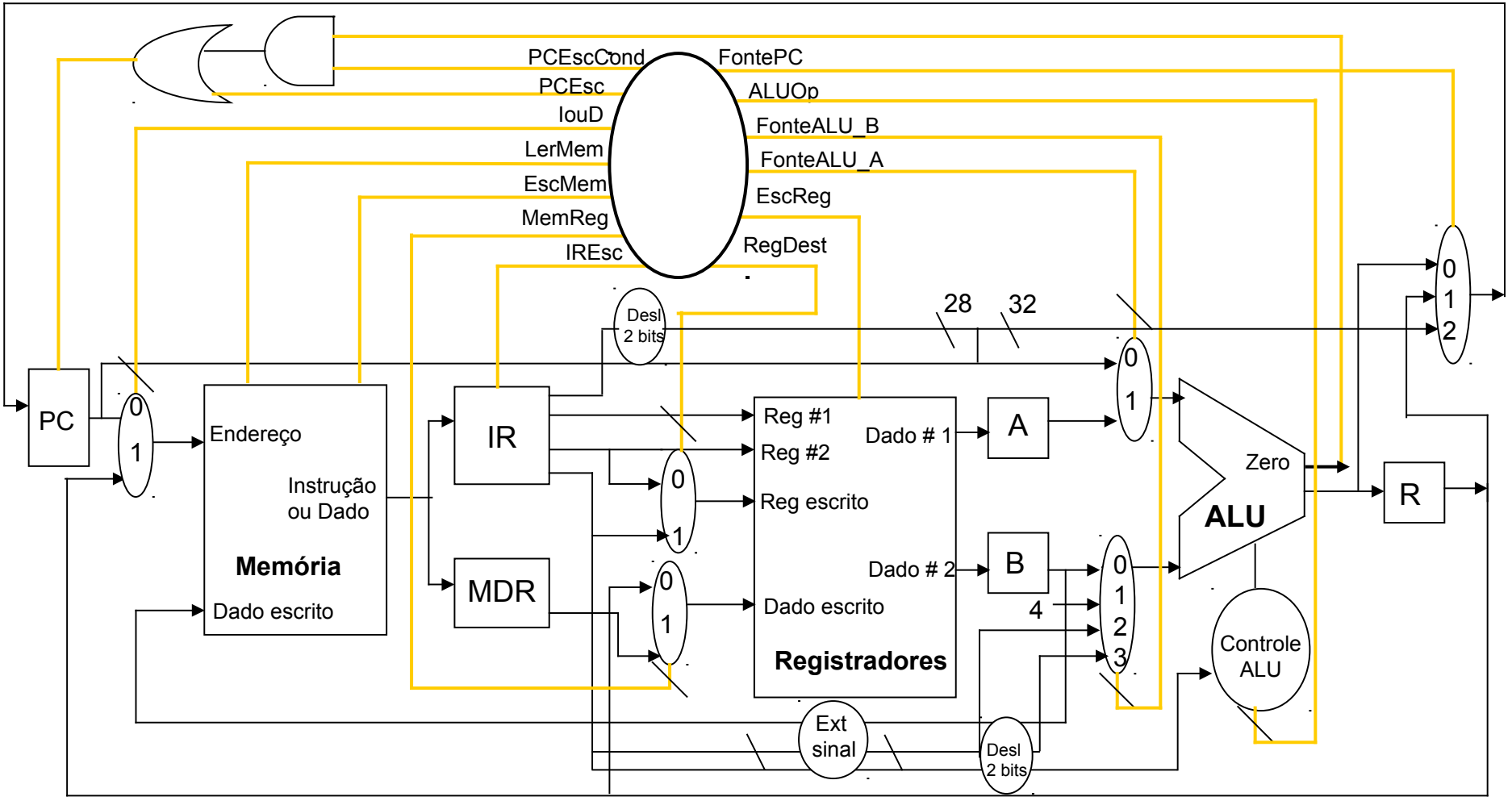
# Projeto Multiciclo

- Escrita no PC:
  - Durante um incremento normal
  - Desvios incondicionais
  - Se instrução de desvio condicional e se a condição for verdadeira
- Controle de escrita possui 2 sinais:
  - PCEsc
  - PCEscCond

# Projeto Multiciclo

- Portas lógicas para a obtenção do sinal de escrita combinando 3 sinais:
  - PCEsc, PCEscCond e Zero (saída da ALU)
- Porta AND
  - Entrada (Zero . PCEscCond)
  - Saída para porta OR (Zero . PCEscCond)
- Porta OR
  - Entrada (saída AND + PCEsc)
  - Saída controle de escrita do PC

# Caminho de Dados Multiciclo Completo



# Projeto Multiciclo

- Sinais de controle de 1 bit

| Sinal | Inativo | Ativo |
|---|---|---|
| RegDest | seleciona rt | seleciona rd |
| EscReg | - | Reg geral carregado com dado escrito |
| FonteALU_A | operando é o PC | operando é o A |
| LerMem | - | Conteúdo da memória é colocado na saída |
| EscMem | - | carrega dado na memória |
| MemReg | dado vem de R | dado vem do MDR |
| IouD | PC é usado para fornecer endereço | R é usado para fornecer endereço |
| IREsc | - | Saída da memória escrita no IR |
| PCEsc | - | PC é atualizado |
| PCEscCond | - | PC é atualizado se saída Zero estiver ativa |

# Projeto Multiciclo

- Sinais de controle de 2 bits

| Sinal | Combinações | Significado |
|---|---|---|
| ALUOp | 00 | soma |
| | 01 | subtração |
| | 10 | depende do campo funct |
| FonteALU_B | 00 | operando é o B |
| | 01 | operando é a constante 4 |
| | 10 | operando vem da extensão de sinal IR |
| | 11 | operando vem da extensão de sinal IR + desl. 2 bits |
| FontePC | 00 | PC + 4 enviado ao PC |
| | 01 | endereço alvo de desvio produzido pela ALU (DvC) |
| | 10 | endereço alvo de desvio enviado ao PC (DvI) |

# Divisão da Execução de Instruções em Ciclos de *Clock*

- A análise do que acontece em cada ciclo de *clock* numa execução multiciclo define os sinais de controle e seus valores
  - Devem ser assumidos ao longo da execução

- A idéia do projeto multiciclo é melhorar o desempenho permitindo que cada instrução execute em diferentes quantidades de ciclos de *clock*

# Divisão da Execução de Instruções em Ciclos de *Clock*

- O tamanho do ciclo de *clock* se reduz a uma operação da ALU, ou um acesso à memória ou ao banco de registradores

- O *clock* terá o tamanho do ciclo igual ao componente de maior tempo de retardo

# Divisão da Execução de Instruções em Ciclos de *Clock*

- Leitura ou escrita em registradores independentes são parte de um mesmo ciclo de *clock*
- Leitura ou escrita no banco de registradores gasta um ciclo de *clock* extra
  - Controles adicionais
- Cada instrução precisa de 3 a 5 passos para ser executada

# Passo 1 – Busca

IR = Memória[PC];

PC = PC + 4;

- Sinais envolvidos

| | | | |
|---|---|---|---|
| LerMem | 1 | FonteALU_B | 01 |
| IREsc | 1 | ALUOp | 00 |
| IouD | 0 | PCEsc | 1 |
| FonteALU_A | 0 | FontePC | 00 |

# Passo 2 – Decodificação

- O **opcode** está sendo decodificado neste estágio, portanto não se sabe ainda qual a instrução está no IR

- Realizar ações com antecedência, que não prejudicam a execução da instrução

  A = Reg [IR[25-21]];

  B = Reg [IR[20-16]];

  R = PC + extensão de sinal (IR[15-0] << 2)

# Passo 2 – Decodificação

- Sinais envolvidos

  FonteALU_A 0 (PC)

  FonteALU_B 11 (extensão de sinal deslocada)

  ALUOp 00 (soma)

- Após este ciclo de *clock* a ação a ser realizada depende do conteúdo da instrução

# Passo 3 – Execução

- A operação do caminho de dados é determinada pela classe da instrução
- A ALU opera sobre os dados preparados no passo anterior

# Passo 3 – Execução

- Acesso à memória

  R = A + extensão de sinal (IR[15-0]);

- Sinais envolvidos

  FonteALU_A 1 (seleciona A)

  FonteALU_B 10 (seleciona extensão de sinal)

  ALUOp 00 (soma)

# Passo 3 – Execução

- Instrução aritmética ou lógica (tipo R)

  R = A op B;

- Sinais envolvidos

  FonteALU_A  1 (seleciona A)

  FonteALU_B  00 (seleciona B)

  ALUOp  10 (campo funct para a operação)

# Passo 3 – Execução

- Desvio Condicional

  Se (A == B)  PC = R;

- Sinais envolvidos

| | |
|---|---|
| FonteALU_A | 1 (seleciona A) |
| FonteALU_B | 00 (seleciona B) |
| ALUOp | 01 (subtração) |
| PCEscCond | 1 |
| FontePC | 01 (PC =  R) |

# Passo 3 – Execução

- Desvio Incondicional

  PC = PC[31-28] | (IR[25-0] << 2);

- Sinais envolvidos

  PCEsc 1

  FontePC 10 (PC = PC[31-28] | (IR[25-0]<<2))

# Passo 4 – Final de Execução

- Acesso à memória

| **lw** | **sw** |
| --- | --- |
| MDR = Memória[R]; | Memória[R] = B |

- Sinais envolvidos

| | | | |
| --- | --- | --- | --- |
| LerMem | 1 | EscMem | 1 |
| IouD | 1 | IouD | 1 |

# Passo 4 – Final de Execução

- Instrução aritmética ou lógica (tipo R)

  Reg[IR[15-11]] = R;

- Sinais envolvidos

  EscReg 1

  RegDest 1

  MemReg 0

# Passo 5 – Final da Leitura da Memória

- Leitura da Memória (lw)

  Reg[IR[20-16]] = MDR;

- Sinais envolvidos

  EscReg 1

  RegDest 0

  MemReg 1

# Definindo o Controle

- O controle para o caminho de dados multiciclo precisa especificar os sinais a serem definidos em qualquer passo e o próximo passo na sequência

- Para especificar o controle multiciclo será usado uma **máquina de estados finitos**

# Máquina de Estados Finitos

- Uma máquina de estados finitos consiste:
  - Conjunto de estados
  - Diretrizes de como mudar de estado
    - **Função de próximo estado**: mapeia o estado atual e as entradas para um novo estado
- Cada estado especifica um conjunto de saídas ativadas quando a máquina está neste estado

# Máquina de Estados Finitos

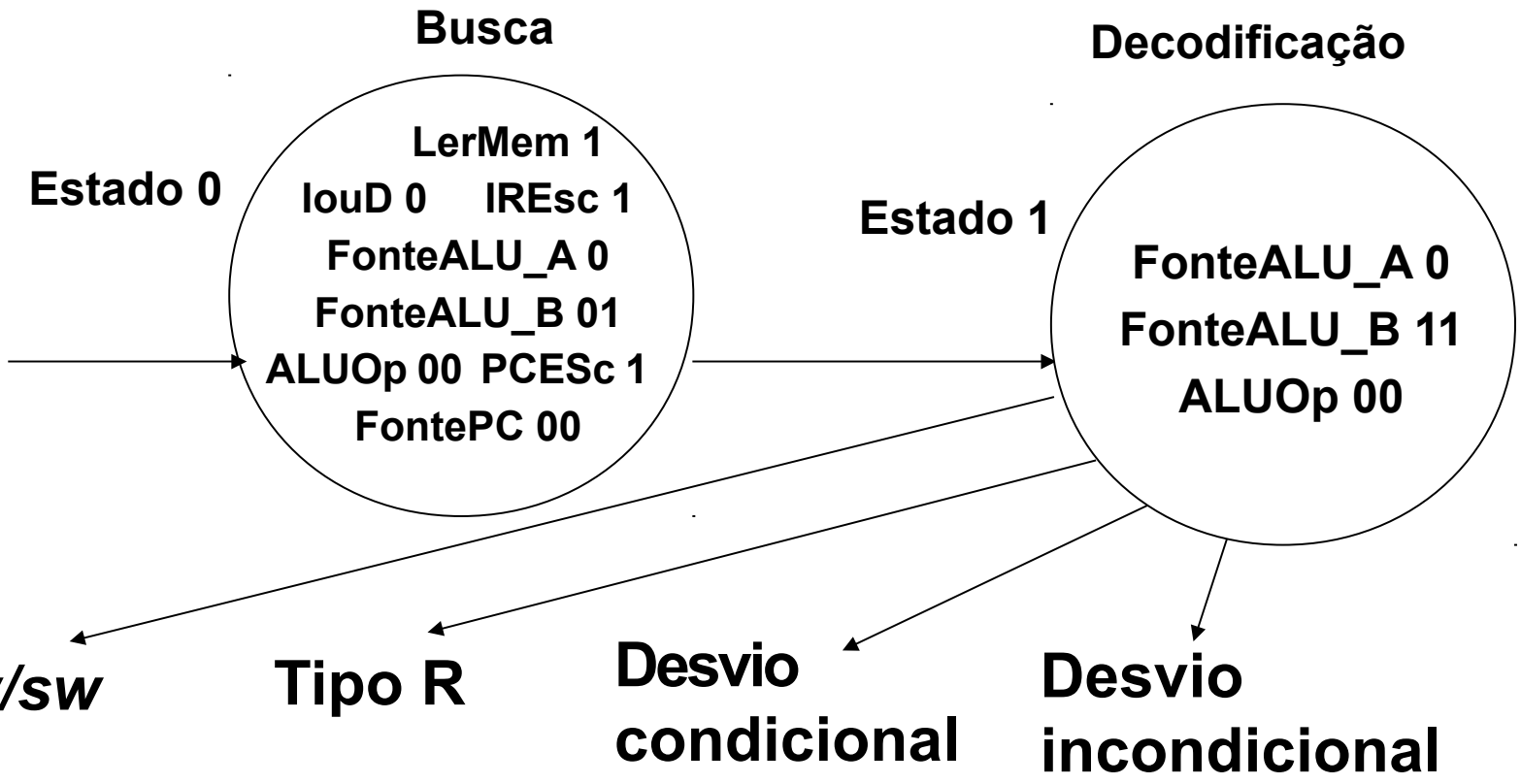
- Em geral, considera-se que todos os sinais não ativados explicitamente estão inativos
- Para controles de multiplexadores, a falta de uma definição específica indica que não nos importa a definição do multiplexador

# Máquina de Estados Finitos

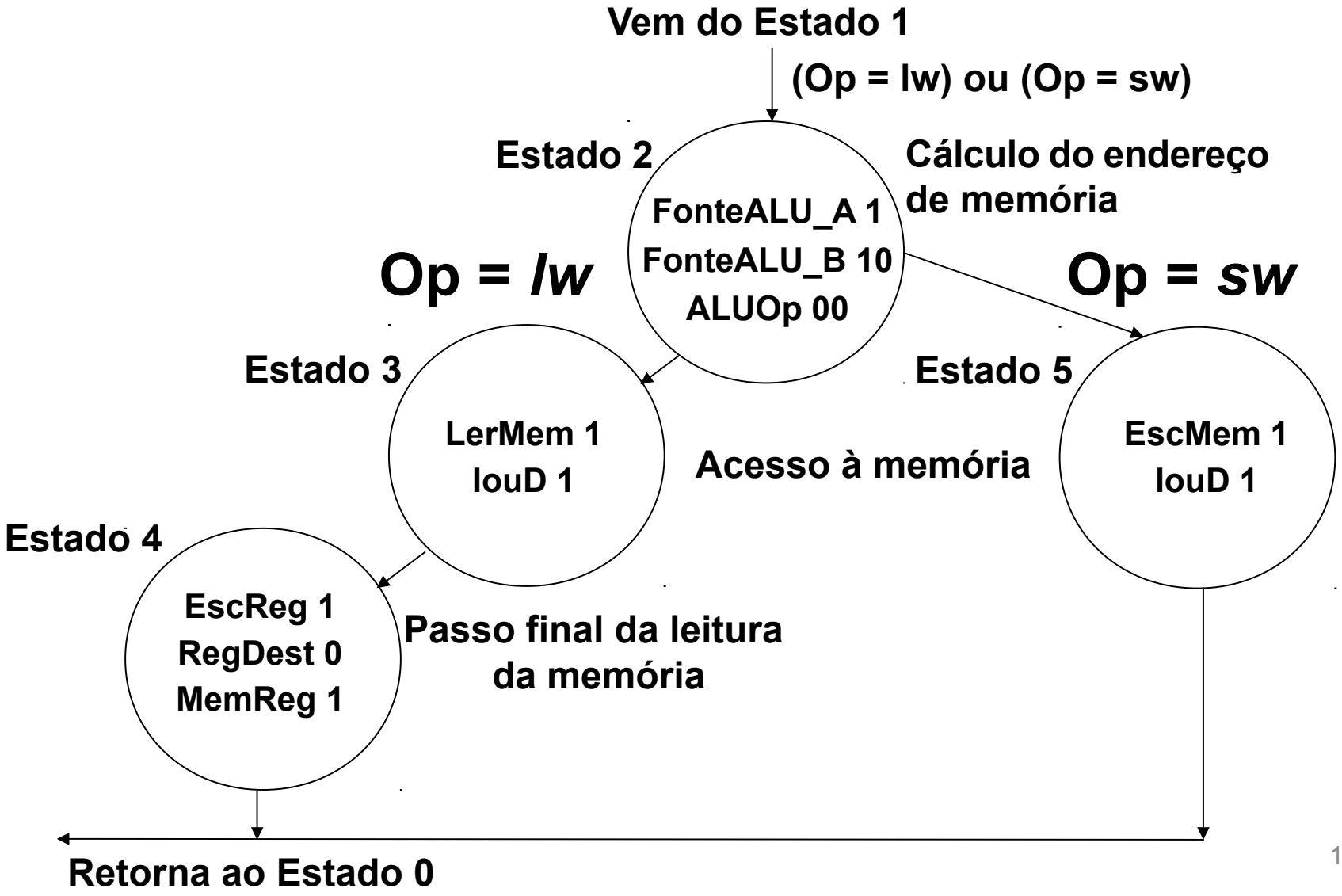
- Cada estado corresponde a um dos passos de execução
- Cada estado na máquina de estados finitos usará um ciclo de *clock*
- Os dois primeiros estados são iguais para todas as instruções

# Busca e Decodificação

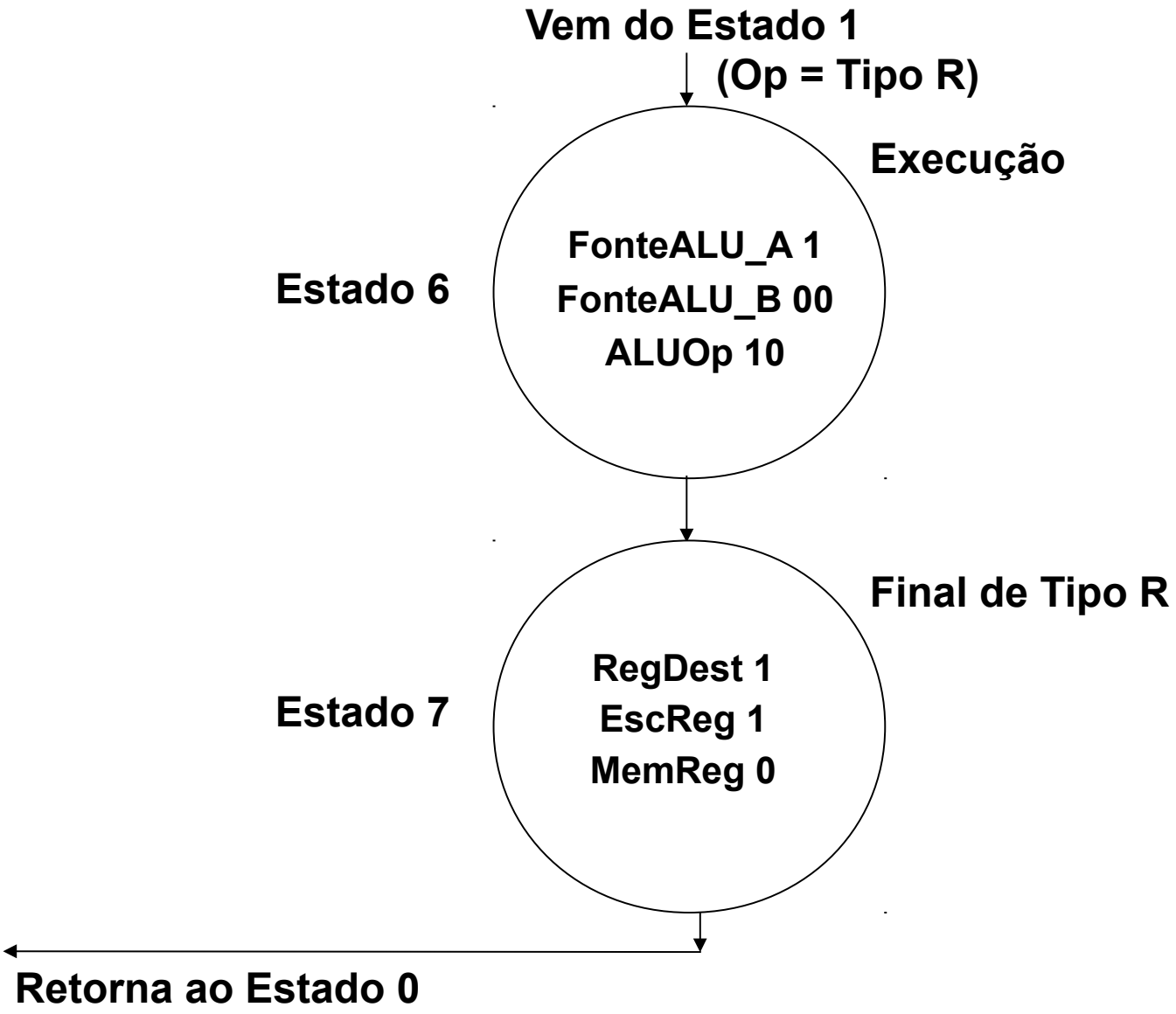
- Máquina de estados finitos



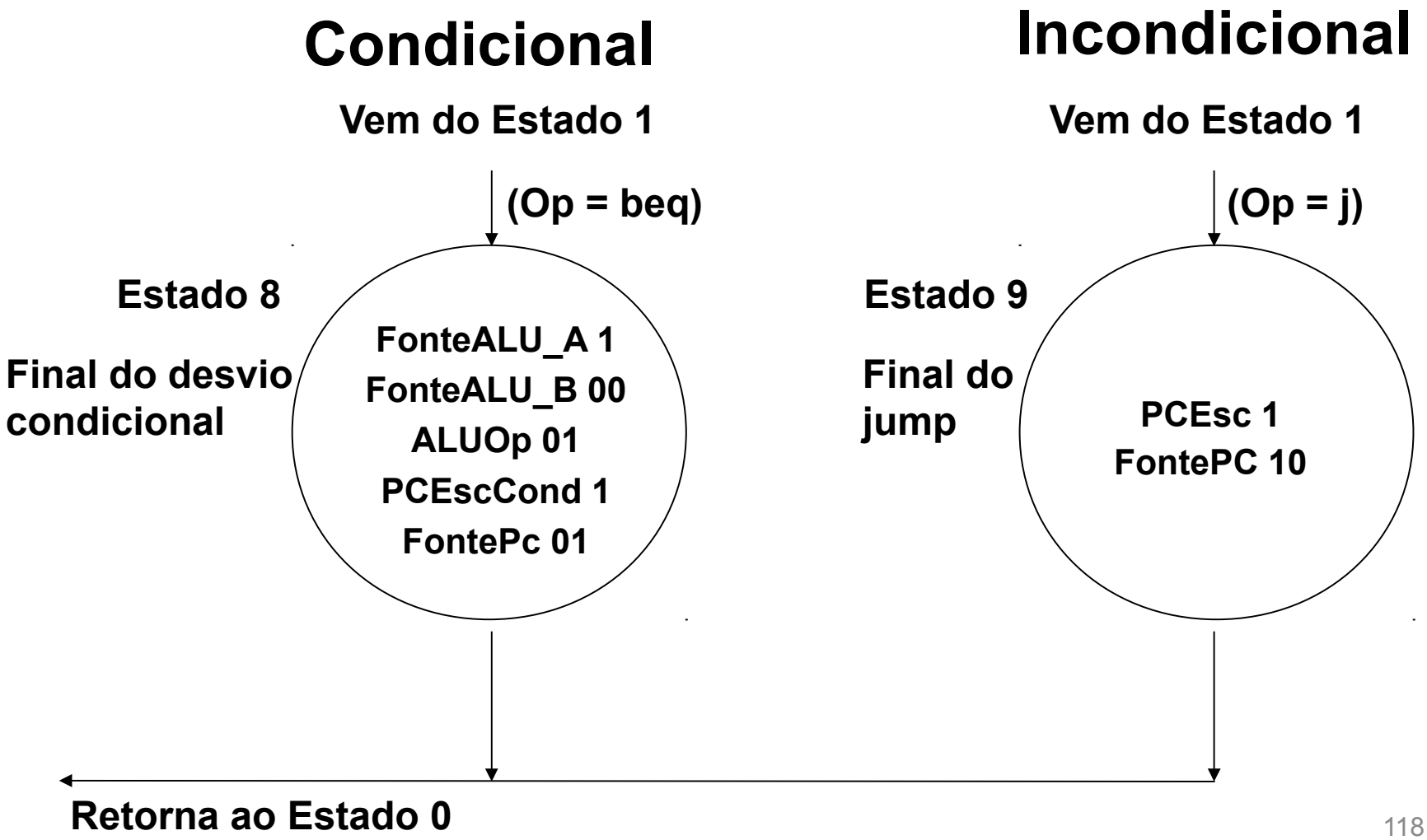
# Acesso à Memória



# Tipo R


Vem do Estado 1
(Op = Tipo R)
Execução
Estado 6
FonteALU_A 1
FonteALU_B 00
ALUOp 10
Final de Tipo R
Estado 7
RegDest 1
EscReg 1
MemReg 0
Retorna ao Estado 0

# Desvios


Condicional
Vem do Estado 1
(Op = beq)
Estado 8
Final do desvio condicional
FonteALU_A 1
FonteALU_B 00
ALUOp 01
PCEscCond 1
FontePc 01
Incondicional
Vem do Estado 1
(Op = j)
Estado 9
Final do jump
PCEsc 1
FontePC 10
Retorna ao Estado 0

# Bibliografia

[1] Maria Clicia S de Castro. “Capítulo 3: O Processador: Caminho de Dados e Controle.”, Apostila disponível em http://www.ime.uerj.br/professores/Mariaclicia/Oc2/oc2.htm

[2] D. A. Patterson e J. L. Hennessy, “Organização e projeto de computadores: a interface hardware/software”, Ed. Campus, 2005.

**Parte do material cedido pela Profa. Clicia (UERJ)**