Caminho de Dados e Controle

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

Sumário

- Uma Implementação Multiciclo;
- Etapas de Execução
- Bibliografia.

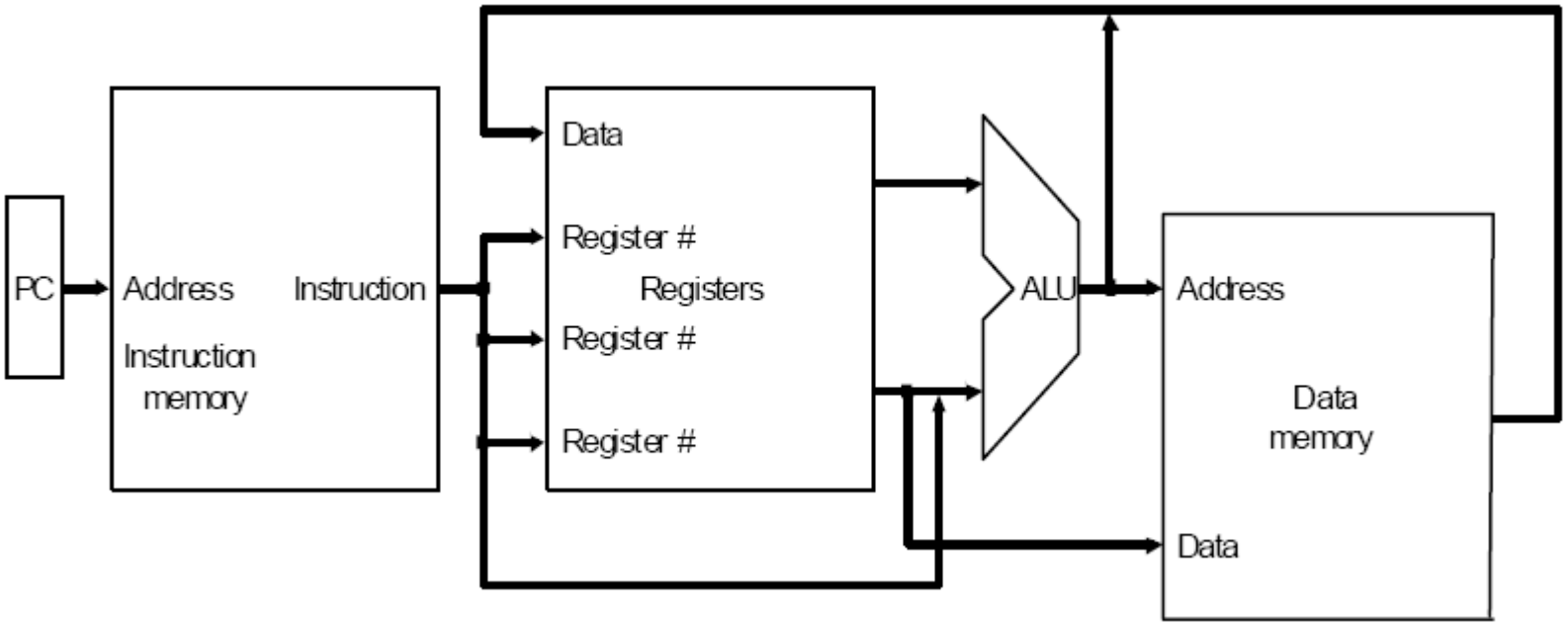
Introdução

- 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.

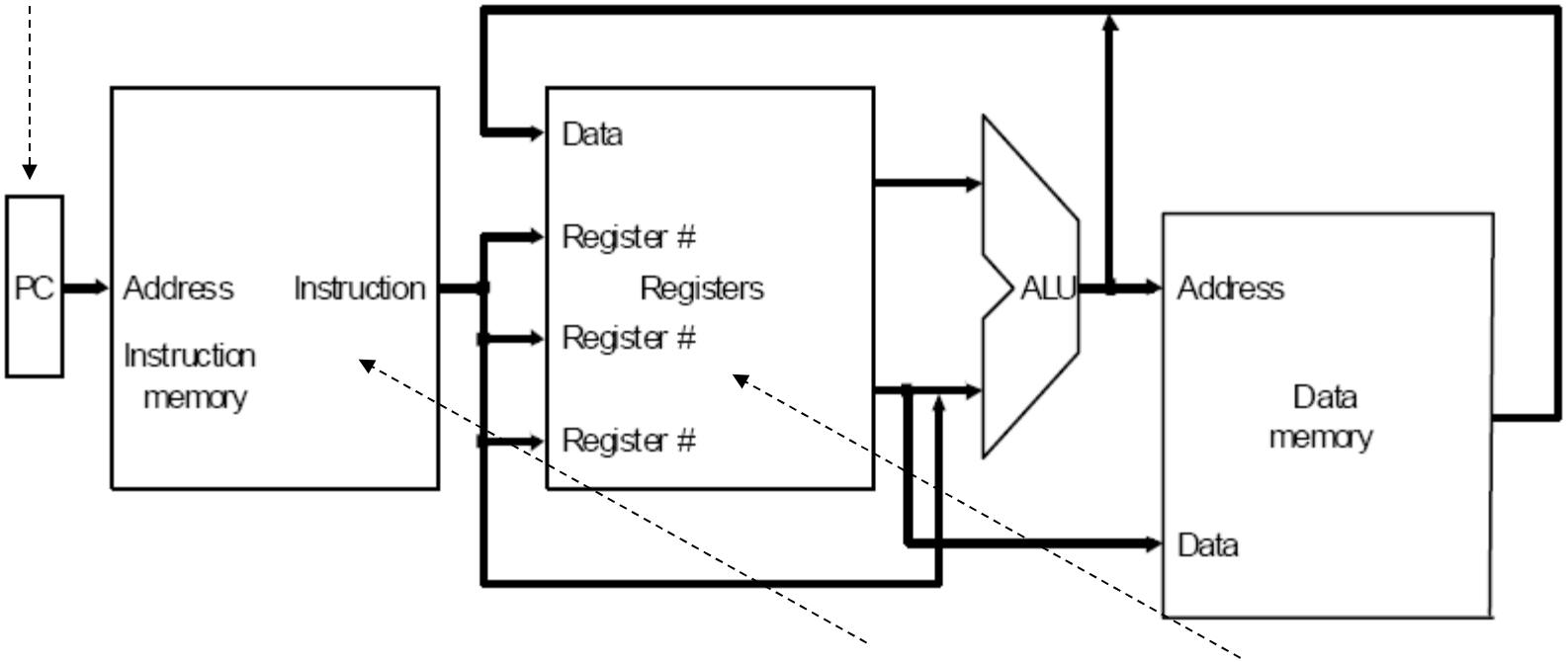
Introdução

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



Introdução

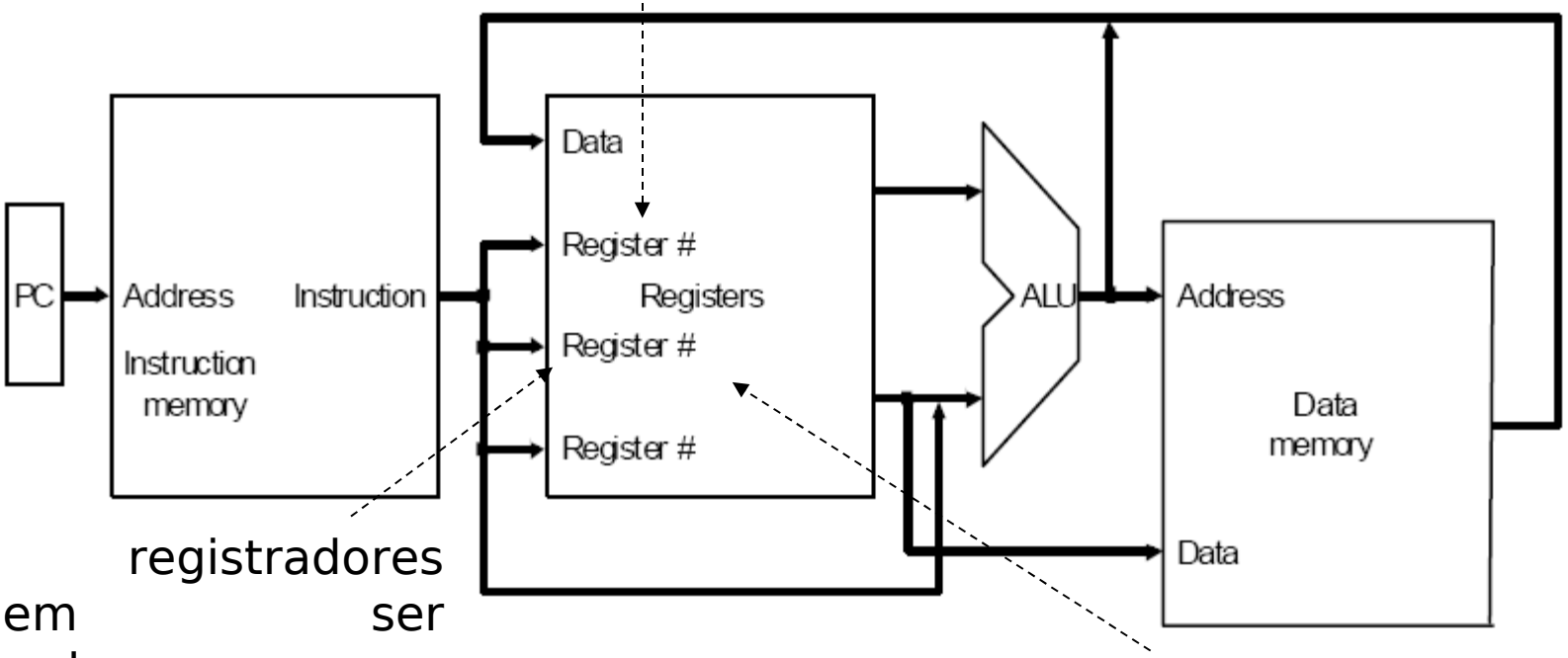
Usado para fornecer o endereço de instrução para a memória de instruções



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

Introdução

Os registradores, por sua vez, podem ser operadores para calcular endereço (lw e sw)



Os registradores podem ser operadores para calcular um resultado aritmético (lógica e aritmética)

Os registradores podem ser operadores para calcular uma comparação (desvio)

Introdução

- Considerações:
  - 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 desvio 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;

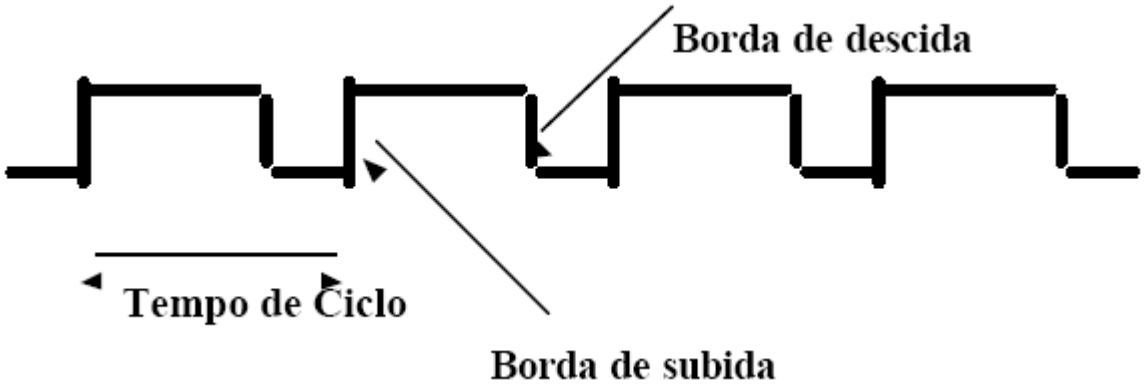
Convenções Lógicas de Projeto

- Antes de discutir o projeto de uma máquina MIPS, vejamos a implementação lógica dessa máquina e como ela será sincronizada(clock);

- 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);

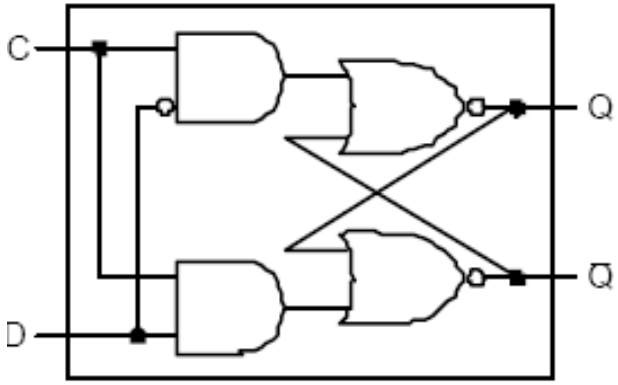
Convenções Lógicas de Projeto

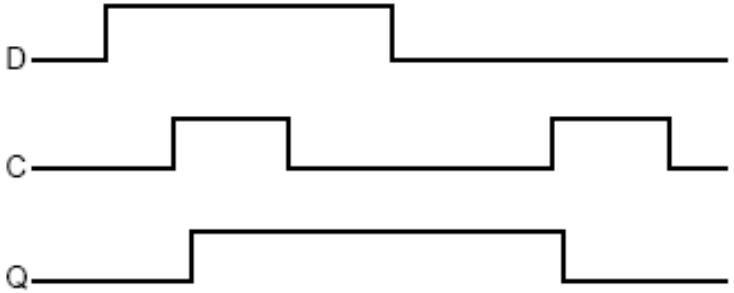
- Em relação ao clock, deve-se definir quando os sinais podem ser lidos e quando podem ser escritos;
  - Evita leituras e escritas simultâneas;



Convenções Lógicas de Projeto

- Ao usarmos uma “trava” os dados serão coordenados pelo clock, vejamos o exemplo com o flip-flop a seguir:





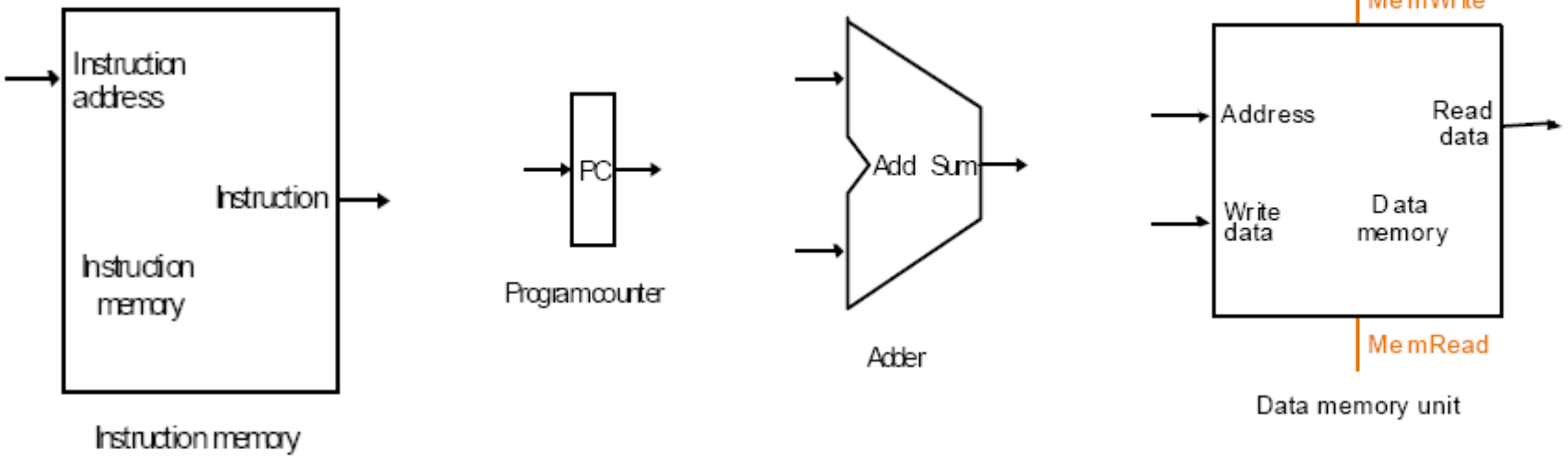
  - O valor a ser armazenado (D)
  - O sinal do relógio (C) indicando leitura ou escrita
  - O valor do estado interno (Q) eo seu complemento ($\overline{Q}$)

Convenções Lógicas de Projeto

- Como apenas os elementos de estado podem armazenar valores de dados:
  - qualquer coleção de lógica combinatória precisa ter duas entradas vindo de um conjunto de elementos de estados e
  - suas saídas escritas em um conjunto de elementos de estado;

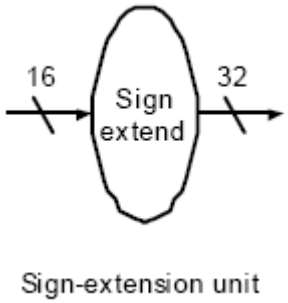
Construindo um Caminho de Dados

- Para iniciar um projeto de caminho de dados examinemos os principais componentes necessários para executar cada classe de instrução MIPS:

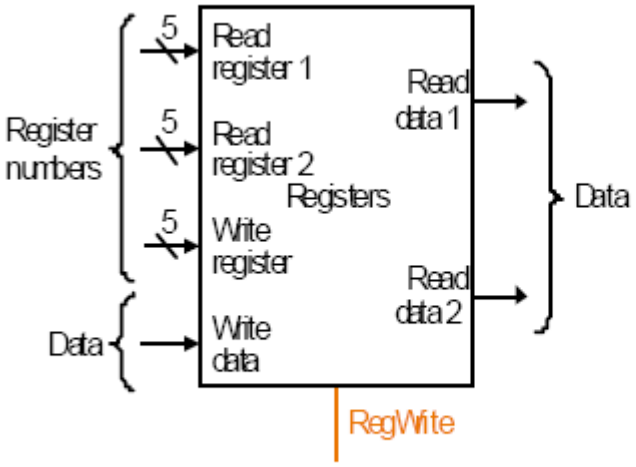


Construindo um Caminho de Dados

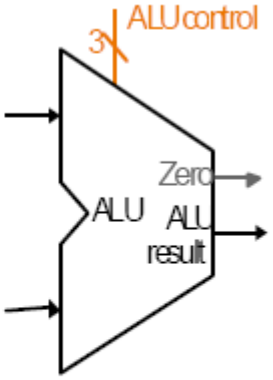
- Para iniciar um projeto de caminho de dados examinemos os principais componentes necessários para executar cada classe de instrução MIPS:



Sign-extension unit



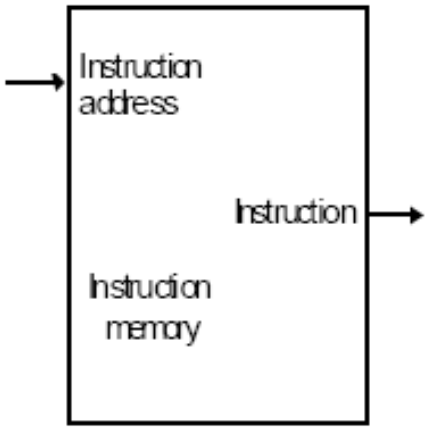
Registers



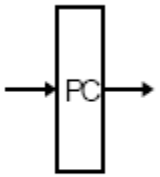
ALU

Construindo um Caminho de Dados

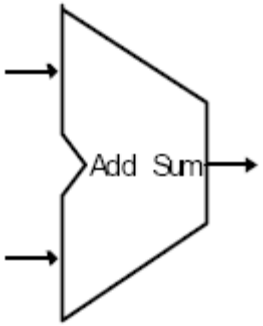
- Memória de Instruções:
  - Armazena as instruções de um programa e fornece instruções dado um endereço;
- Contador de Programa(PC):
  - Usado para conter o endereço da instrução atual;
- Somador:
  - Incrementa o PC para o endereço da próxima instrução;



Instruction memory



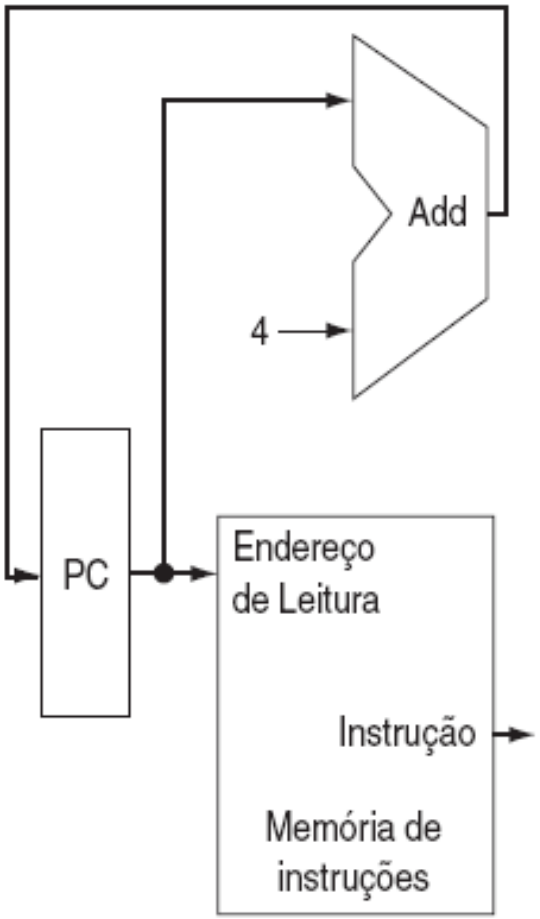
Programcounter



Adder

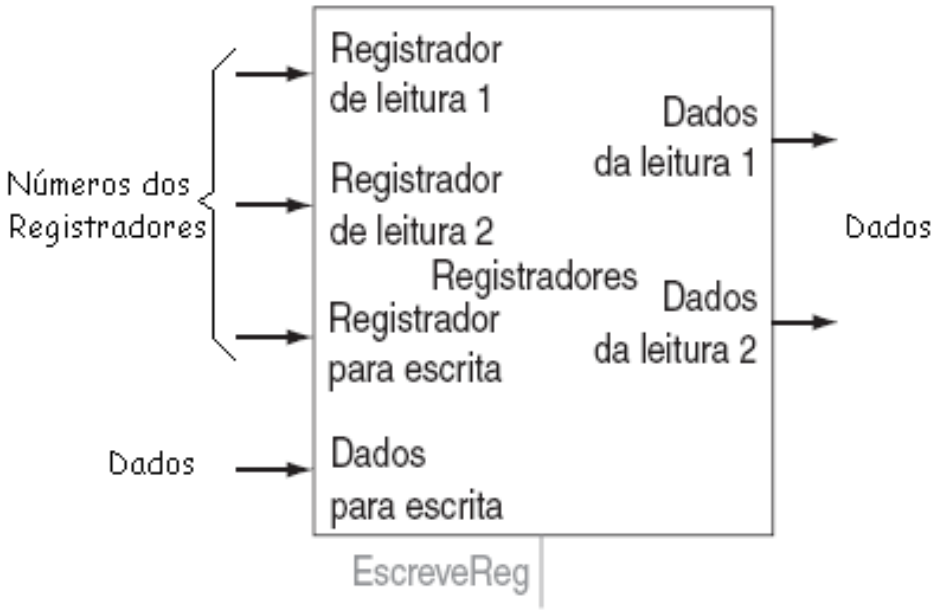
Construindo um Caminho de Dados

- Considerações:
  - Para executar qualquer instrução, precisamos começar buscando a instrução na memória de instruções;
  - Para preparar para executar a próxima instrução, também temos de incrementar a o contador de programa (4 bytes);



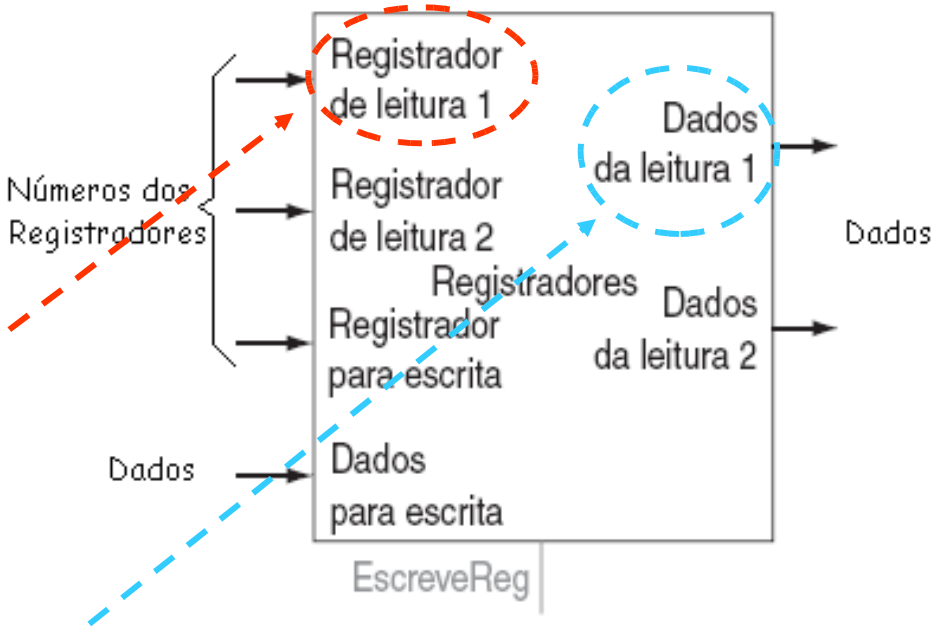
Construindo um Caminho de Dados

- Banco de Registradores:
  - Elemento de estado que consiste em um grupo de registradores que podem ser lidos e escritos fornecendo um número de registro a ser acessado;
  - O banco de registradores contém o estado dos registradores da máquina, que será passado para a ULA ;



Construindo um Caminho de Dados

- Considerando uma instrução do tipo R:
  - Precisamos ler duas palavras de dados do banco de registradores e escrever uma palavra de dados no banco de registradores;
  - 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;

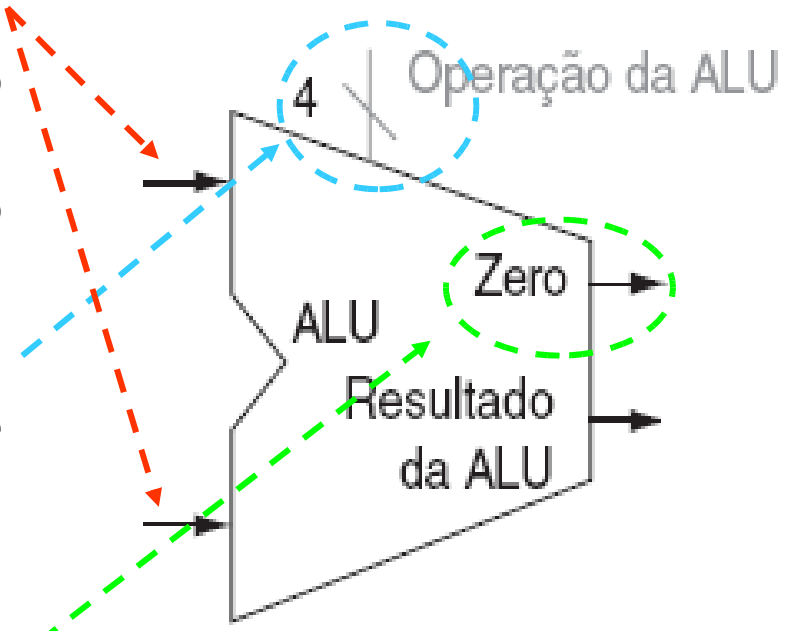


Construindo um Caminho de Dados

- Considerando uma instrução do tipo R:
  - Para escrever uma palavra de dados, precisamos de duas entradas:
  - Uma para especificar o número do registrador a ser escrito
  - e uma para fornecer os dados a serem escritos no registrador;
  - As escritas, no entanto, são controladas pelo sinal de controle de escrita, que precisa ser ativo na transmissão do clock;

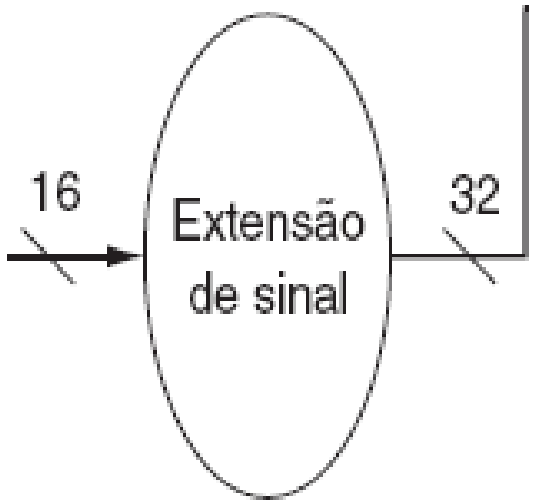
Construindo um Caminho de Dados

- 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 sinal de controle de quatro bits;
  - e possui ainda uma saída de detecção de Zero para implementar desvios;



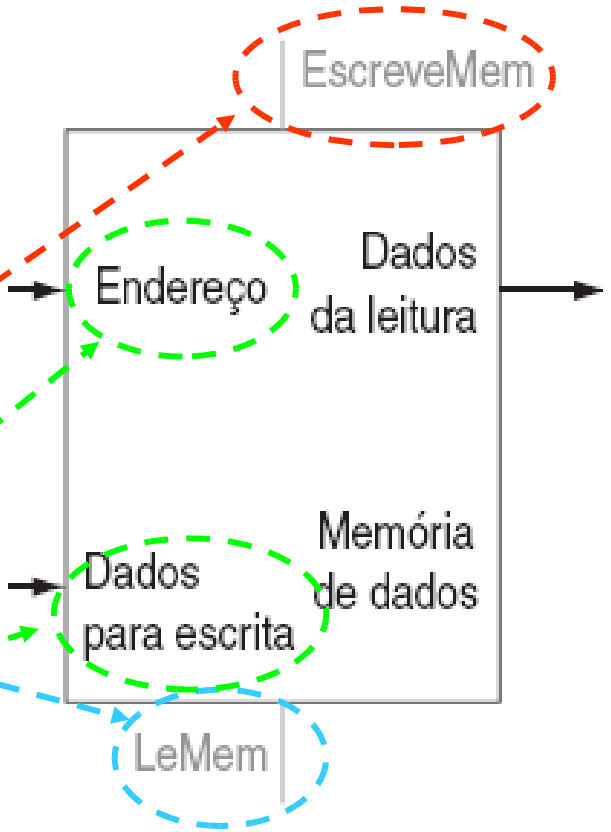
Construindo um Caminho de Dados

- Extensão de Sinal:
  - Considerando uma instrução do load word e store word:
  - São instruções que necessitam calcular um endereço de memória somado ao registrador base;
  - Como o campo reservado para offset (deslocamento) nessas instruções são de 16 bits precisaremos de uma unidade para estender o sinal para 32 bits;



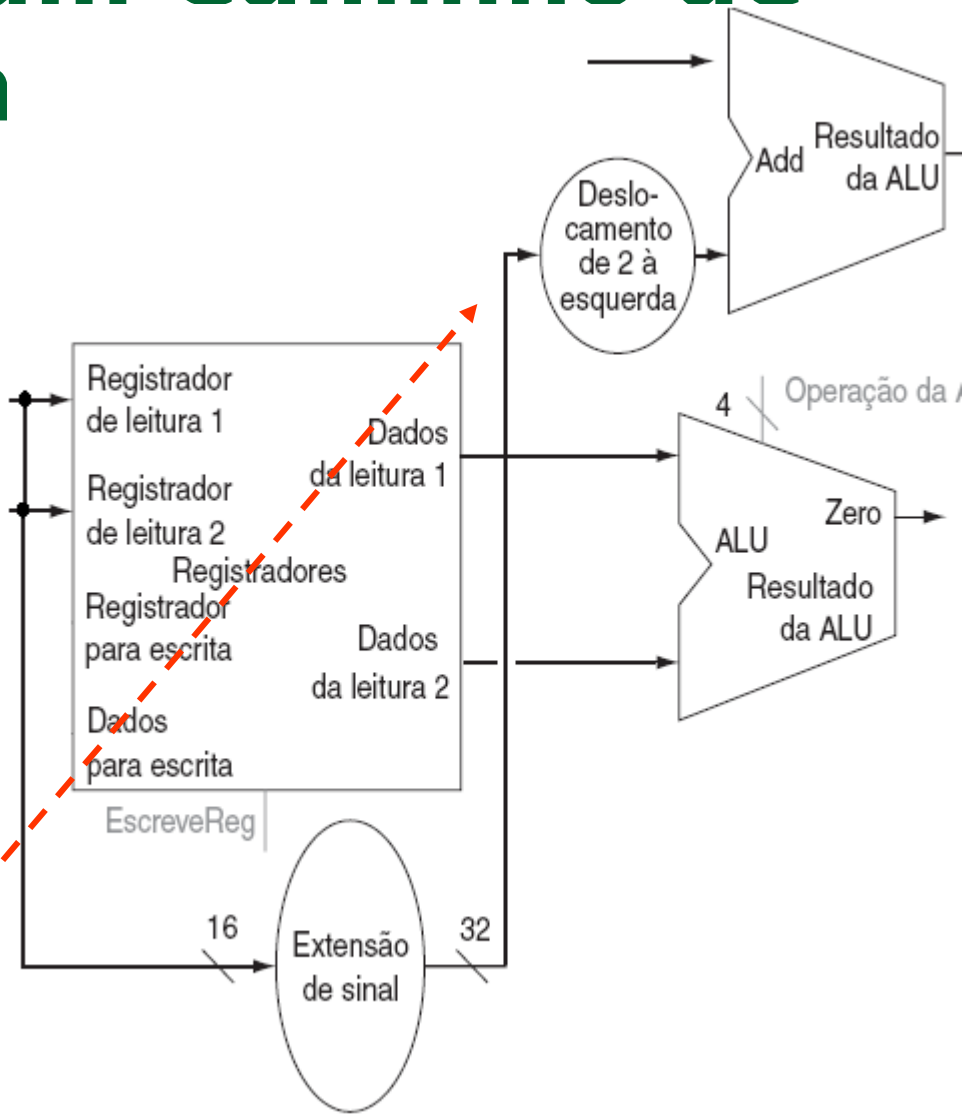
Construindo um Caminho de Dados

- Memória de Dados:
  - Instruções load e store necessitam ainda de uma unidade de memória da qual ler ou escreve dados;
  - A memória de dados precisa ser escrita com instrução store, portanto, ela tem sinais de controle de leitura(load) e escrita;
  - Uma entrada de endereço e uma entrada para dados a serem escritos na memória;



Construindo um Caminho de Da

- Endereço de destino do desvio:
  - Uma operação beq possui três operando (beq $t1, $t2, off_set);
  - Como implementar essa solução?
  - Solução: somando o campo off_set estendido ao sinal da instrução com o PC;



Construindo um Caminho de Dados

- Existem dois detalhes na definição de instruções de desvio:
  - O conjunto de instruções especifica que a base para o cálculo de desvio é o endereço da instrução seguinte ao desvio
  - A arquitetura também diz que o campo offset(deslocamento) é deslocado 2 bits para a esquerda de modo que seja um offset de uma word; esse deslocamento aumenta a faixa efetiva do campo offset por um fator de quatro vezes;

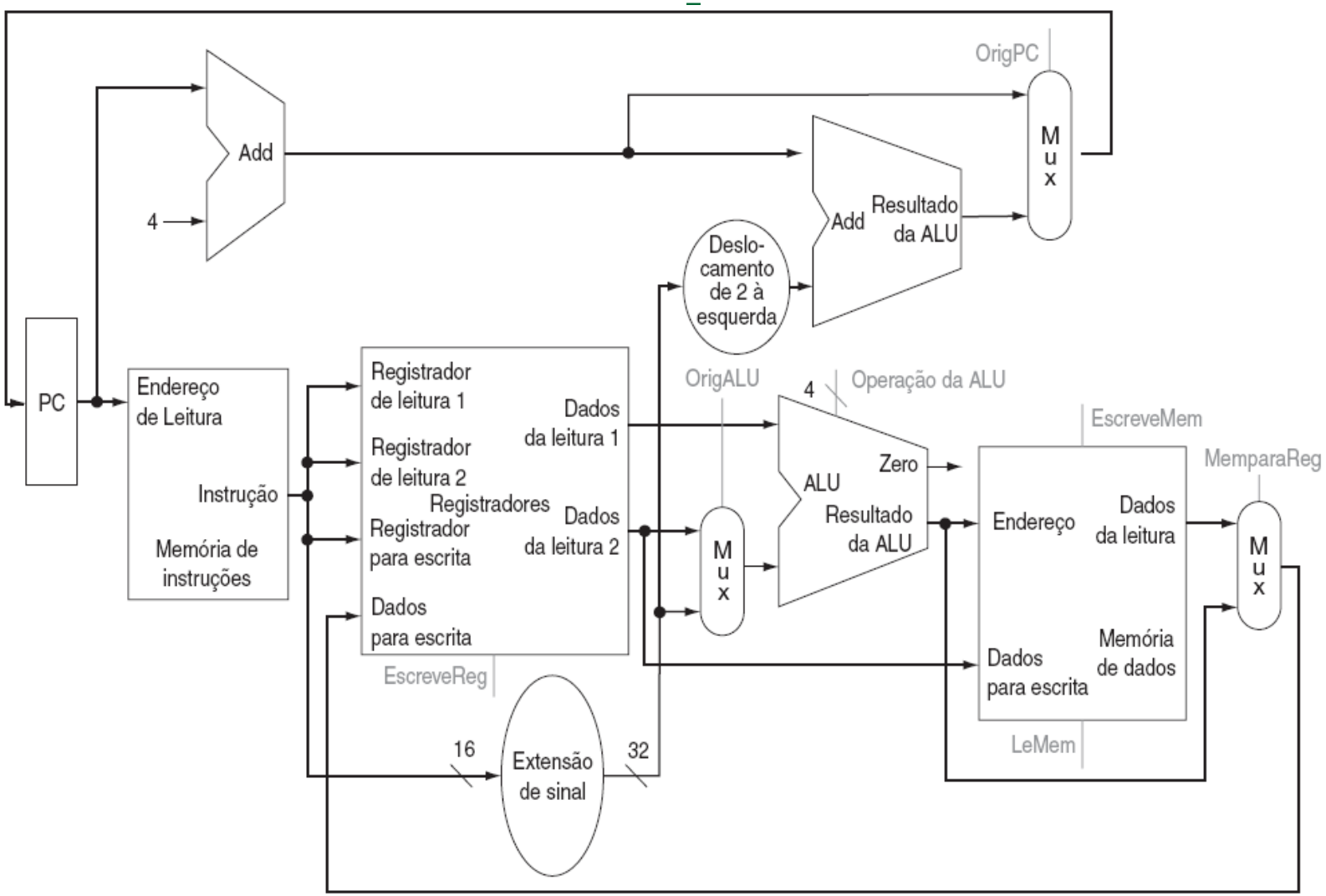
Construindo um Caminho de Dados

- Considerações do endereço de destino do desvio:
  - Além de calcular o endereço de destino do desvio, também precisamos saber se a próxima instrução é a instrução que seque seqüencialmente ou a instrução no endereço de destino do desvio;

```
beq $t1, $t2, off_set
```

  - Quando a condição é verdadeira, o endereço de destino do desvio se torno o novo PC
  - Quando a condição é falsa, o PC incrementado deve substituir o PC atual;

Construindo um Caminho de



O Controle da ULA

- Como vimos anteriormente, a ULA possui quatro entradas de controle.

- Esses bits não foram codificados, portanto apenas 6 das 16 combinações são usadas neste subconjunto:

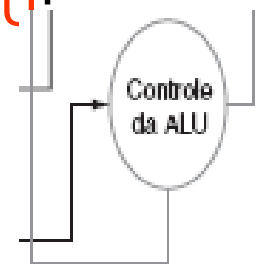
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

O Controle da ULA

- Considerações:
  - NOR é necessária para outras partes do conjunto de instruções MIPS;
  - Para lw e sw usamos a ULA para calcular o endereço de memória por adição.
  - Para instruções do tipo R, a ULA precisa realizar uma das cinco ações (AND, OR, add, subtract ou set on less than) dependendo do campo funct (função) de 6 bits;
  - Para branch equal, a ULA precisa realizar uma subtração;

O Controle da ULA

- Podemos gerar a entrada do controle da ULA de 4 bits usando uma pequena unidade de controle:
  - Essa unidade de controle tem como entradas o campo funct da instrução e o campo control de 2 bits que chamados de OpALU:



- OpALU indica se a operação a ser realizada deve ser:
  - add(00) para loads e stores,
  - subtract (01) para beq ou
  - determinar pela operação de codificada no campo funct(10);

O Controle da ULA

- Vejamos o quadro completo da relação OpALU de 2 bits e o código de função de 6 bits.

OpALU		Campo func.						
OpALU1	OpALU 0	F5	F4	F3	F2	F1	F0	Operação
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

  - A unidade de controle gera os bits OpALU, que, então são usados como entrada para o controle da ULA que gera os sinais reais para controlar a ULA;

Projeto da Unidade de Controle Principal

- Para continuar a construção o controle principal, é necessário considerar o formato das instruções:

R	op	rs	rt	rd	shamt	Funct
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
I	op	rs	rt	Imm		
	6 bits	5 bits	5 bits	16 bits		
J	op	Target				
	6 bits	26 bits				

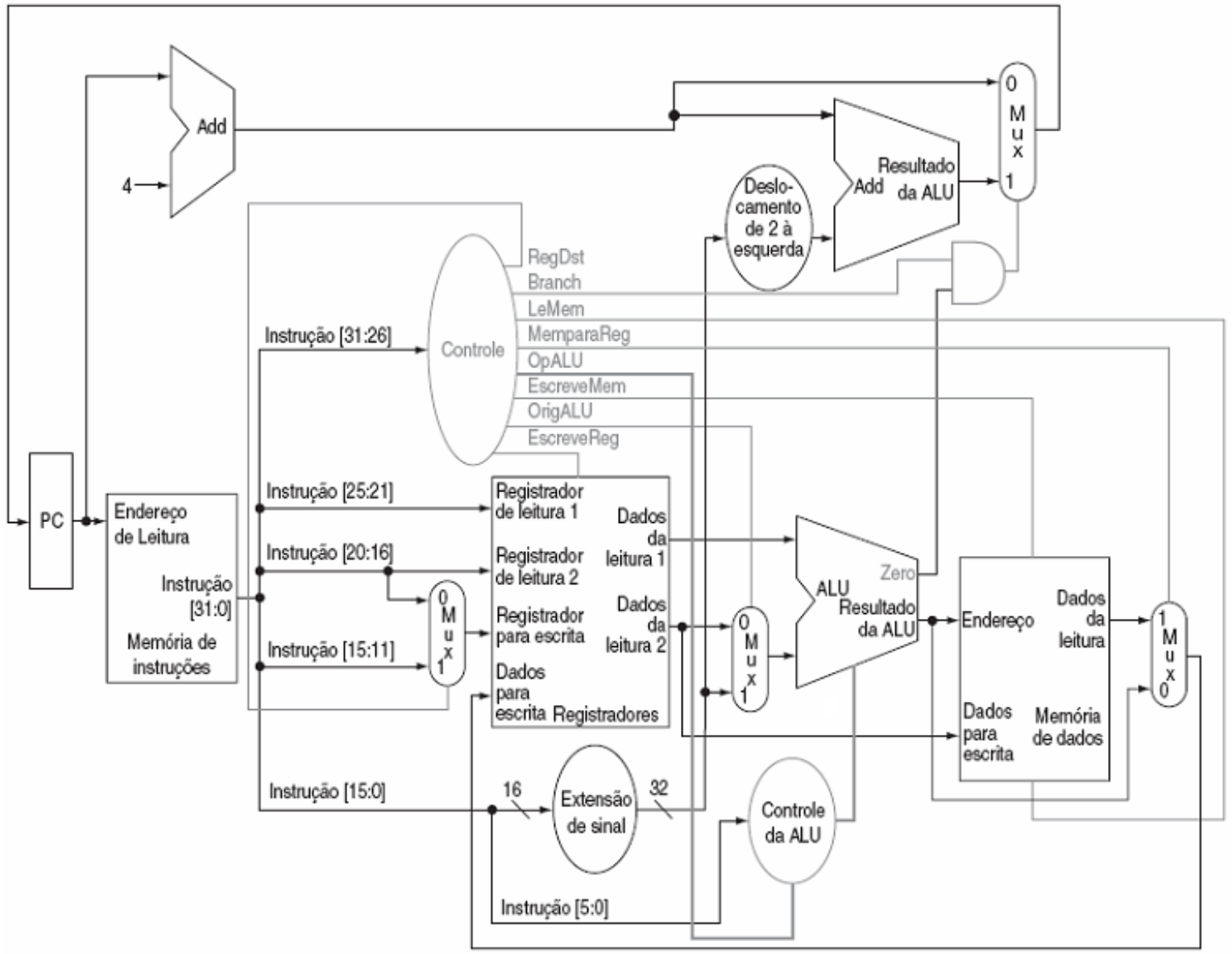
  - O campo op, está sempre contido nos bits 31:26. Iremos nos referir a esse campo como Op[31:26];

Projeto da Unidade de Controle Principal

- Continuação das considerações sobre o formato das instruções:
  - Os dois registradores a serem lidos nas instruções do tipo R, branch equal e store nas posições 25:21 e 20:16;
  - O registrador de base para a instrução de load e store está sempre nas posições de bit 25:21;
  - O offset de 16 bits para beq, load e store está sempre nas posições 15:0;

Projeto da Unidade de Controle Principal

- Continuação das considerações sobre o formato das instruções:
  - O registrador de destino está em um de dois lugares:
    - Para load: ele está nas posições 20:16 (rt)
    - Tipo R: ele está nas posições 15:11 (rd).
    - Portanto, precisamos incluir um multiplexador para selecionar que campo da instrução será usado para indicar o número de registrador a ser escrito;


Add
4
0
M
u
x
1
Resultado
Add
da ALU
Deslo-
camento
de 2 à
esquerda
RegDst
Branch
LeMem
MemparaReg
OpALU
EscreveMem
OrigALU
EscreveReg
Instrução [31:26]
Controle
PC
Endereço
de Leitura
Instrução
[31:0]
Memória de
instruções
Instrução [25:21]
Instrução [20:16]
Instrução [15:11]
Registrador
de leitura 1
Registrador
de leitura 2
Registrador
para escrita
Dados
para
escrita
Dados
da
leitura 1
Dados
da
leitura 2
Registradores
ALU
Zero
Resultado
da ALU
Endereço
Dados
da
leitura
Dados
para
escrita
Memória
de dados
Instrução [15:0]
16
Extensão
de sinal
32
Controle
da ALU
Instrução [5:0]

Operação do Caminho de Dados

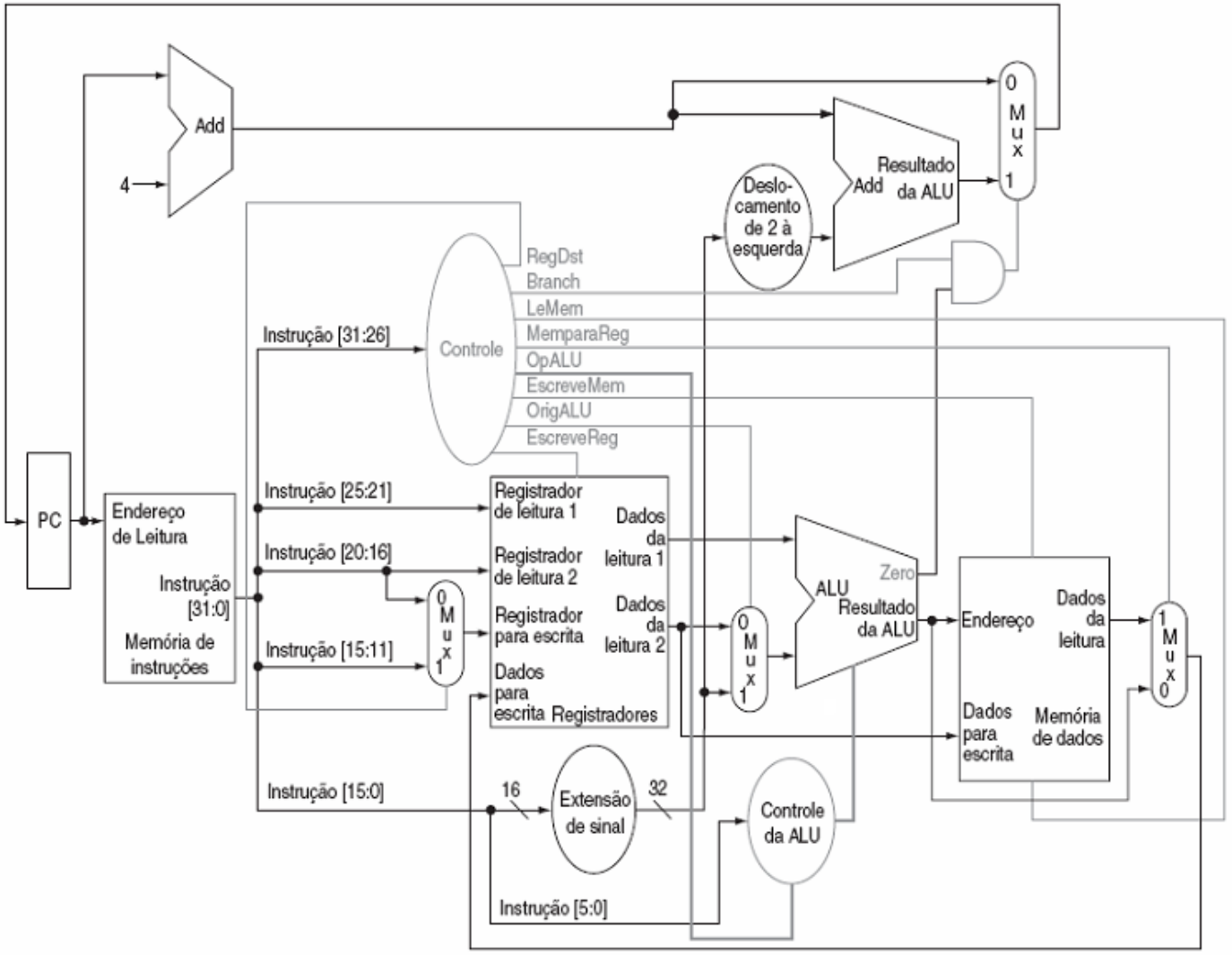
- Vejamos o fluxo das três classes de instrução (Tipo R, I e J) por meio do caminho de dados;
- É válido ressaltar que todas as operações ocorrem em 1 ciclo de clock.
- Dessa forma podemos pensar em quatro etapas para executar uma instrução do tipo R, vejamos:
- Tomemos a instrução a seguir:

add $t1, $t2, $t3

Operação do Caminho de Dados

- Operações ordenadas pelo fluxo da informação:
  - A instrução é buscada e o PC é incrementado;
  - Dois registradores. $t2 e $t3, são lidos do banco de registradores e a unidade de controle principal calcula a definição das linhas de controle também durante essa etapa;
  - A ULA opera nos dados lidos do banco de registradores, usando o código de função (bits 5:0) para gerar a função da ULA;
  - O resultado da ULA é escrito no banco de registradores usando os bits 15:11 da instrução para selecionar o registrador destino;

Operação do Caminho de Dados – Tipo R



Operação do Caminho de Dados

- Vejamos como ficaram as linhas de controle:

Instrução	RegDst	OrigALU	Mempara Reg	Escreve Reg	Le Mem	Escreve Mem	Branch	ALUOpl	ALUOp0
formato R	1	0	0	1	0	0	0	1	0

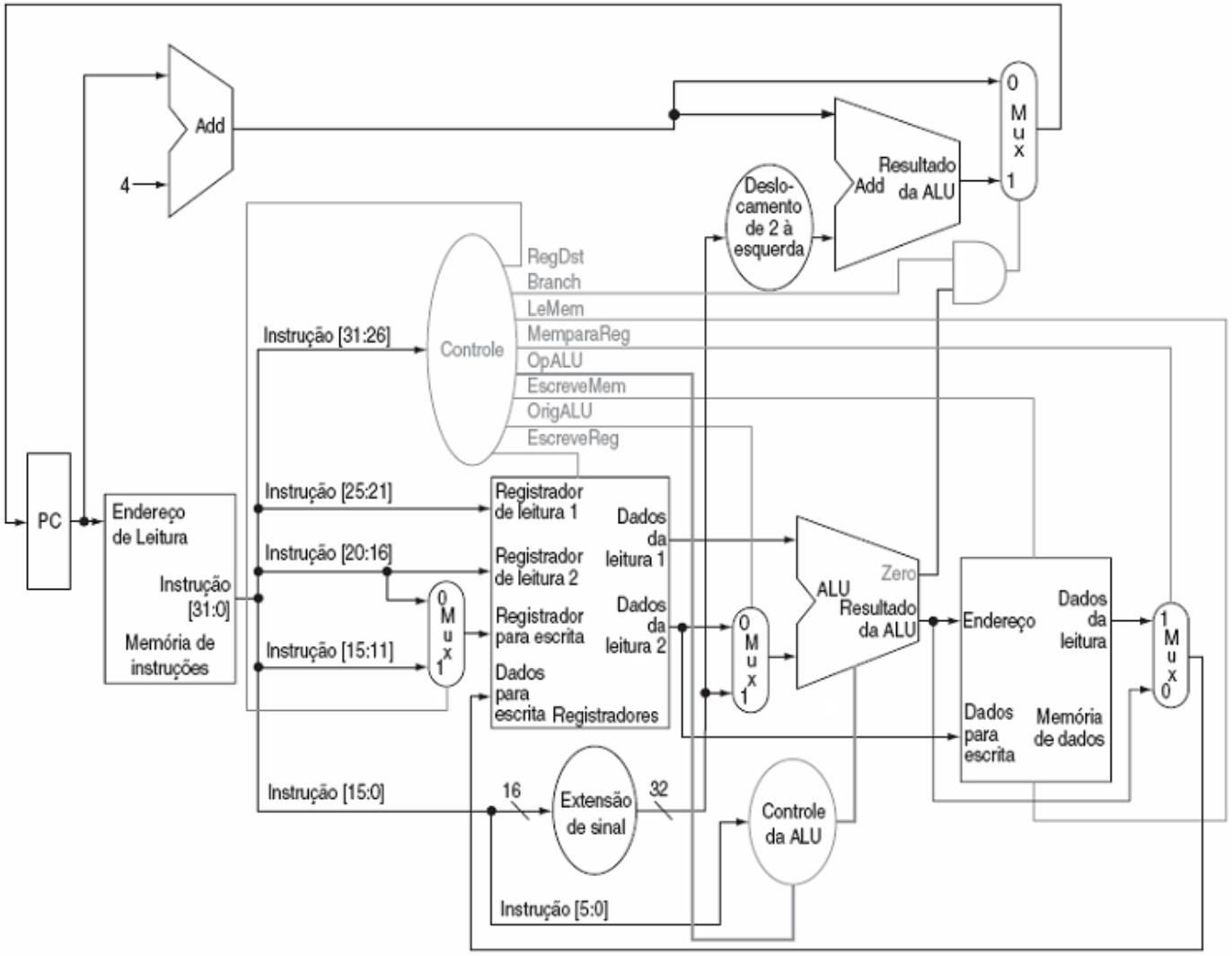
Operação do Caminho de Dados

- Tomemos a instrução a seguir:

lw $t1, offset($t2)

  - A instrução é buscada e o PC é incrementado;
  - Um valor do registrador ($t2) é lido do banco de registradores;
  - A ULA calcula a soma do valor lido do banco de registradores com os 16 bits menos significativos com sinal estendido da instrução offset;
  - A soma da ULA é usada como o endereço para a memória de dados;
  - Os dados da unidade de memória são escritos no banco de registradores, o registrador de destino é fornecido pelos bits 20:16 da instrução ($t1);

Operação do Caminho de Dados – Load



Operação do Caminho de Dados

- Vejamos como ficaram as linhas de controle:

Instrução	RegDst	OrigALU	Mempara Reg	Escreve Reg	Le Mem	Escreve Mem	Branch	ALUOpl	ALUOp0
formato R	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0

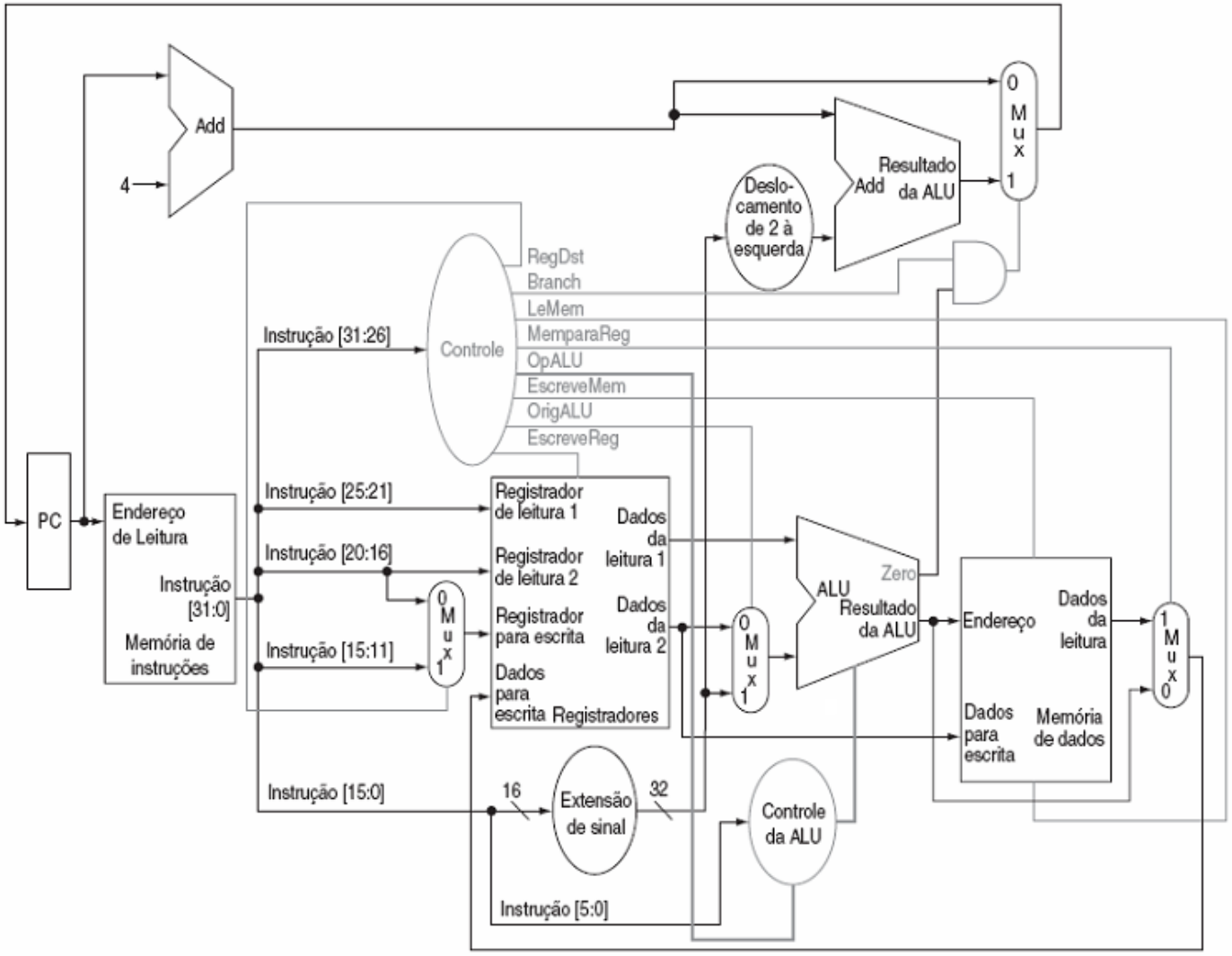
Operação do Caminho de Dados

- Tomemos a instrução a seguir:

beq $t1,$t2, offset

  - A instrução é buscada e o PC é incrementado;
  - Dois registradores, $t1 e $t2, são lidos do banco de registradores;
  - A ULA realiza uma subtração dos valores de dados lidos do banco de registradores. O valor de PC + 4 é somado aos bits menos significativos com sinal estendido da instrução offset deslocados de dois para a esquerda; O resultado é o endereço de destino do desvio
  - O resultado Zero da ULA é usado para decidir o resultado de que somador deve ser armazenado no PC;

Operação do Caminho de Dados – beq



Operação do Caminho de Dados

- Vejamos como ficaram as linhas de controle:

Instrução	RegDst	OrigALU	Mempara Reg	Escreve Reg	Le Mem	Escreve Mem	Branch	ALUOp1	ALUOp0
formato R	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	[illegible]								
beq	X	0	X	0	0	0	1	0	1

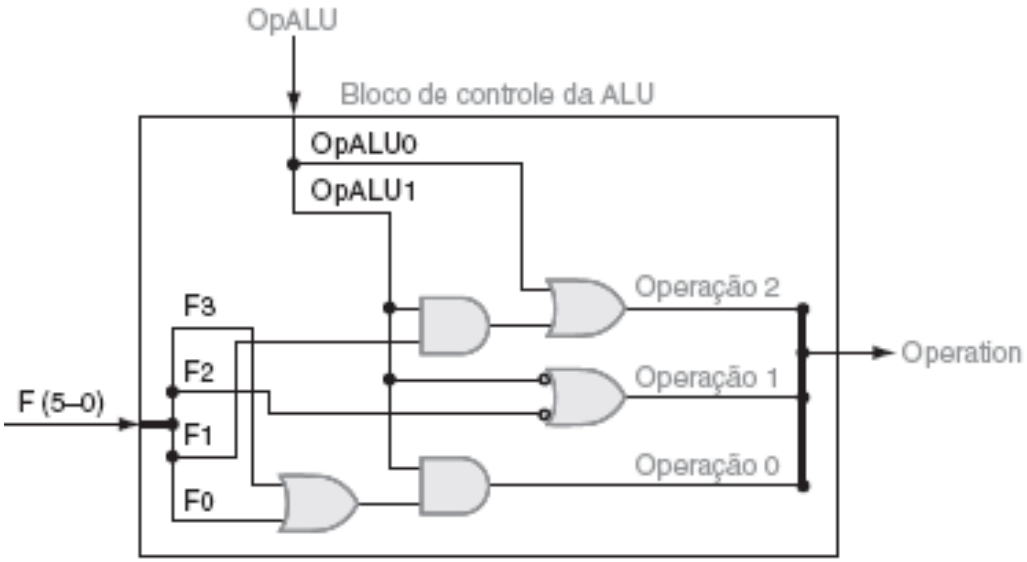
Exercício

- Qual seriam as linhas de controle para a instrução sw?

Instrução	RegDst	OrigALU	Mempara Reg	Escreve Reg	Le Mem	Escreve Mem	Branch	ALUOpl	ALUOp0
formato 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

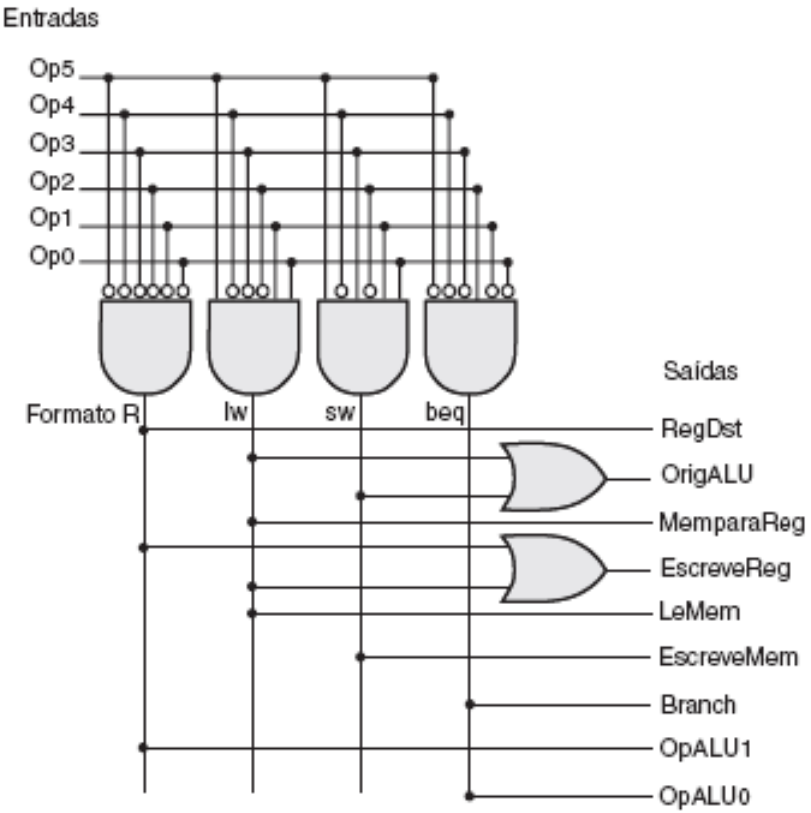
Operação do Caminho de Dados

- Vejamos detalhes da lógica combinacional do controle da ALU:



Operação do Caminho de Dados

- Vejamos detalhes da lógica combinacional do controle:



Uma Implementação Multiciclo

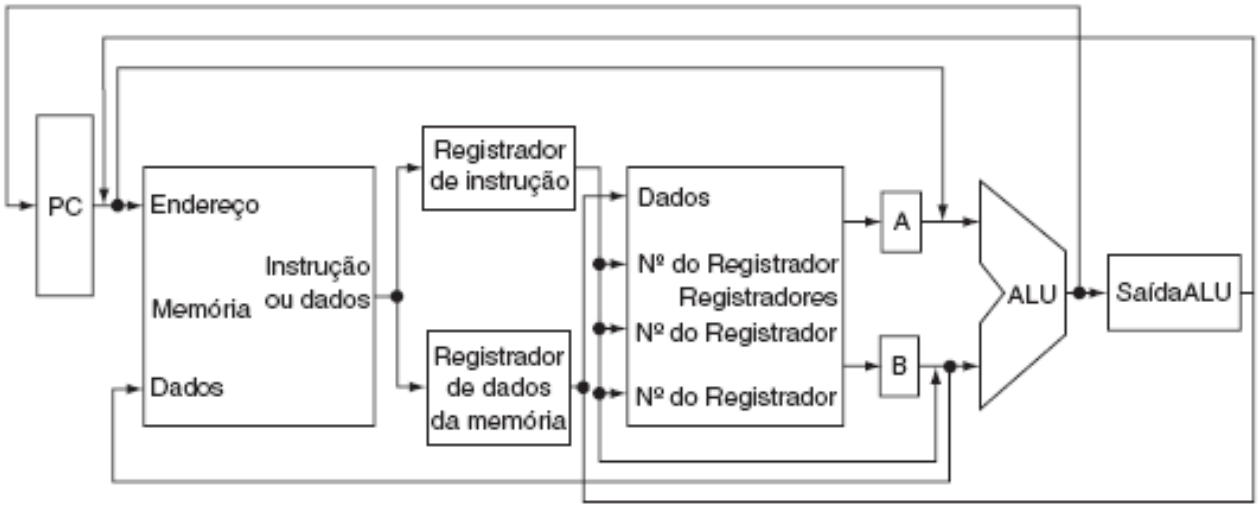
- Na implementação de ciclo único toda instrução opera em 1 clock de uma duração fixa;
- Nesse caso, consideramos que o ciclo de clock é igual ao atraso do pior caso para todas as instruções;
- Portanto, a implementação de ciclo único é ineficiente tanto em seu desempenho quanto em seu custo de hardware (unidades funcionais duplicadas);

Uma Implementação Multiciclo

- Como dividimos cada instrução em uma série de etapa correspondentes às operações das unidades funcionais, podemos usar essas etapas para definir ciclos;
- Em uma implementação multiciclo, cada etapa na execução levará 1 ciclo de clock;
- A implementação multiciclo permite que uma unidade funcional seja usada mais de uma vez por instrução;

Uma Implementação Multiciclo

- Vejamos a visão alto nível de um caminho de dados multiciclo:



Uma Implementação Multiciclo

- Considerações:
  - Uma única unidade de memória é usada para instruções e para dados;
  - Existe uma única ULA, em vez de uma ULA e dois somadores;
  - Um ou mais registradores são adicionados após cada unidade funcional para conter a saída dessa unidade até o valor a ser usado em um ciclo de clock subseqüente;
  - No final de um ciclo de clock, todos os dados usados nos ciclos de clock subseqüentes precisam ser armazenados em um elemento de estado visível ao programador: banco de registradores, o PC ou a memória;

Uma Implementação Multiciclo

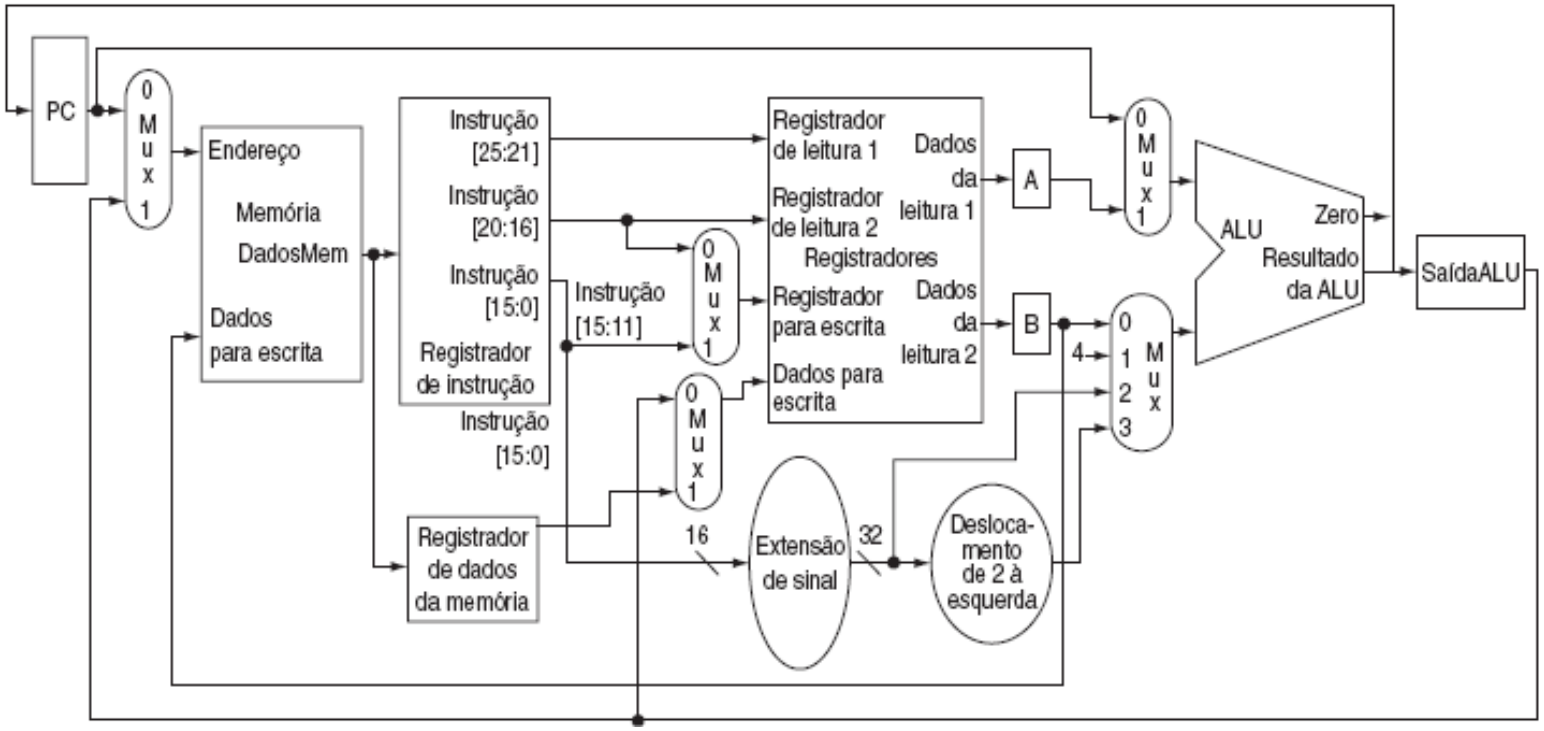
- Considerações:
  - Por outro lado, os dados usados pela mesma instrução em um ciclo de clock posterior precisam ser armazenados em um desses registradores adicionais;
  - O registrador IR e o registrador de dados da memória (MDR) são incluídos para salvar a saída da memória para uma leitura de instrução e uma leitura de dados respectivamente;
  - Os registradores A e B são usados para conter os valores dos registradores operandos lidos do banco de registradores;
  - O registrador SaídaALU contém a saída da ULA;

Uma Implementação Multiciclo

- Considerações:
  - Todos os registradores exceto o IR contém dados apenas entre um par de ciclos de clock adjacente, e portanto, não precisarão de um sinal de controle de escrita;
  - Como várias unidades funcionais são compartilhadas para diferentes finalidades, precisamos de ambos: incluir multiplexadores e expandir os multiplexadores existentes;
  - Vejamos a figura que mostra os detalhes do caminho de dados com os multiplexadores adicionais:

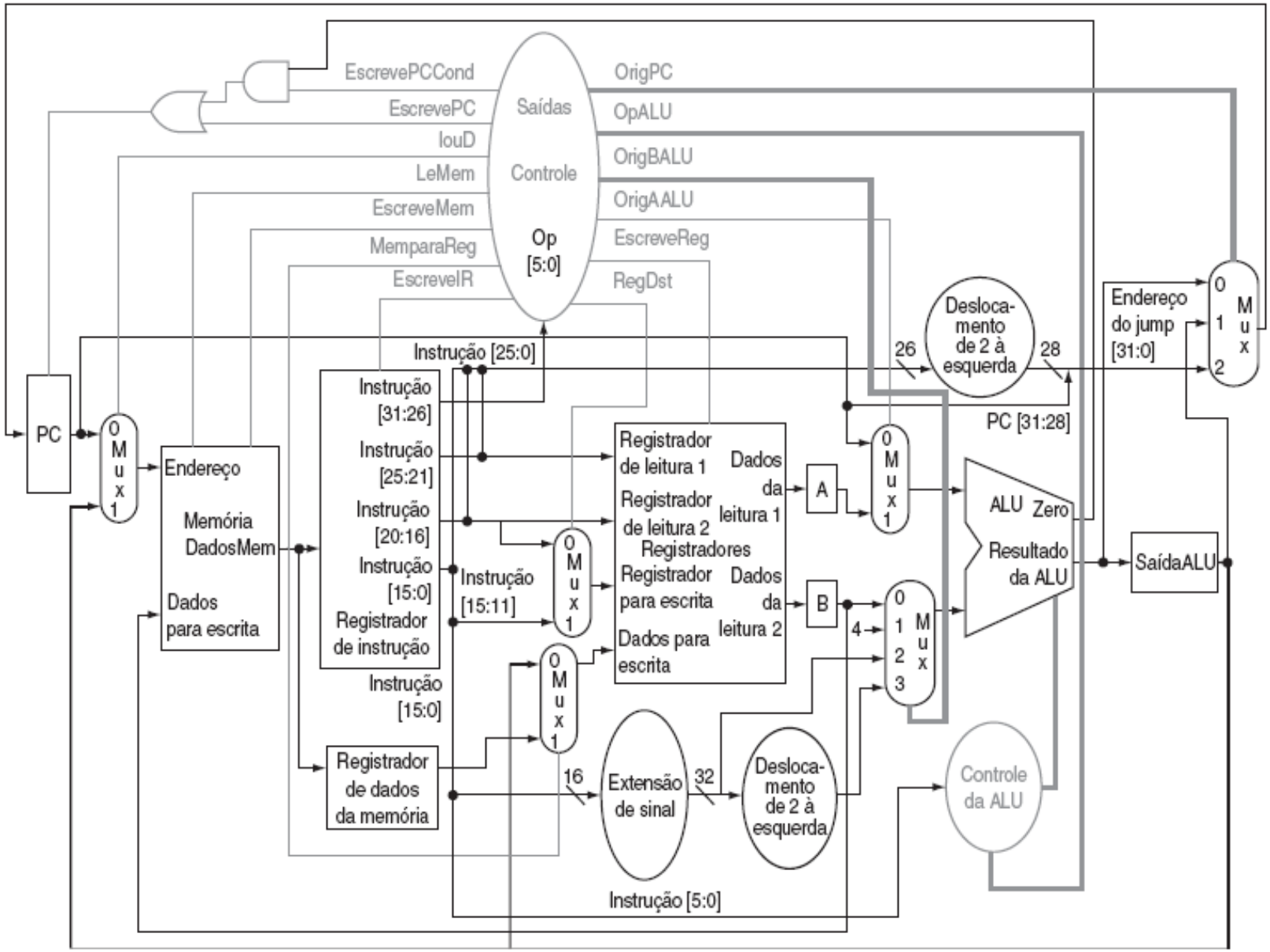
Uma Implementação Multiciclo

- Caminho de dados multiciclo para o MIPS manipular as instruções básicas



Uma Implementação Multiciclo

- O caminho de dados mostrado na figura anterior exigirá diferentes sinais de controle;
  - As unidades de estado visíveis aos programador (o PC, a memória e os registradores), bem como o IR, precisarão de sinais de controle de escrita;
  - A memória também precisará de um sinal de leitura;
  - Cada multiplexador de quatro entradas exige duas linhas de controle;
  - Vejamos o caminho de dados completo para a implementação multiciclo:


EscrevePCCond
EscrevePC
IouD
LeMem
EscreveMem
MemparaReg
EscreveIR
Saídas
Controle
Op
[5:0]
OrigPC
OpALU
OrigBALU
OrigAALU
EscreveReg
RegDst
Instrução [25:0]
26
Desloca-
mento
de 2 à
esquerda
28
Endereço
do jump
[31:0]
M
u
x
0
1
2
PC [31:28]
PC
0
M
u
x
1
Endereço
Memória
DadosMem
Dados
para escrita
Instrução
[31:26]
Instrução
[25:21]
Instrução
[20:16]
Instrução
[15:0]
Registrador
de instrução
Instrução
[15:11]
Instrução
[15:0]
Registrador
de dados
da memória
Registrador
de leitura 1
Registrador
de leitura 2
Registradores
Registrador
para escrita
Dados para
escrita
Dados
da
leitura 1
Dados
da
leitura 2
A
B
4
ALU
Zero
Resultado
da ALU
SaídaALU
0
1
2
3
16
Extensão
de sinal
32
Desloca-
mento
de 2 à
esquerda
Controle
da ALU
Instrução [5:0]

Etapas de Execução

- Dado o caminho de dados, vejamos o que deve acontecer em cada ciclo de clock da execução multiciclo:
  - Busca da Instrução;
  - Decodificação da instrução e busca dos registradores;
  - Execução, cálculo do endereço de memória ou conclusão do desvio;
  - Acesso à memória ou conclusão da instrução tipo R;
  - Etapa de escrita adiada (write-back)

Etapas de Execução

- Etapa 1: Busca da Instrução
  - Use o PC para obter a instrução e colocá-la no registrador de Instrução;
  - Incremente o PC em 4 e coloque o resultado novamente no PC;
  - Descrição na RTL (Register-Transfer Language), vejamos:

    IR <= Memory[PC];

    PC <= PC + 4;

Etapas de Execução

- Etapa 2: Decodificação da Instrução e Busca dos registradores
  - Leia os registradores rs e rt no caso de instruções que utilizem eles;
  - Calcule o endereço de desvio no caso da instrução ser um branch;
  - Vejamos em RTL:

    A <= Reg[IR[25:21]];

    B <= Reg[IR[20:16]];

    ALUOut <= PC + (sign-extend(IR[15:0]) < 2);

Etapas de Execução

- Etapa 3: Dependente da instrução
  - A ULA executará uma das três funções a seguir com base no tipo de instrução, vejamos em RTL:
  - Tipo R:

    ALUOut <= A op B;
  - Referência à memória:

    ALUOut <= A + sign-extend(IR[15:0]);
  - Branch:

    if (A == B) PC <= ALUOut;

Etapas de Execução

- Etapa 4: Tipo R ou acesso à memória
  - Loads e stores acessam a memória:

    MDR <= Memory[ALUOut];

    ou

    Memory[ALUOut] <= B;

  - Instruções do tipo R finalizam

    Reg[IR[15:11]] <= ALUOut;

Etapas de Execução

- Etapa 5: Conclusão da leitura da memória
  - Load:

Reg[IR[20:16]] <= MDR;

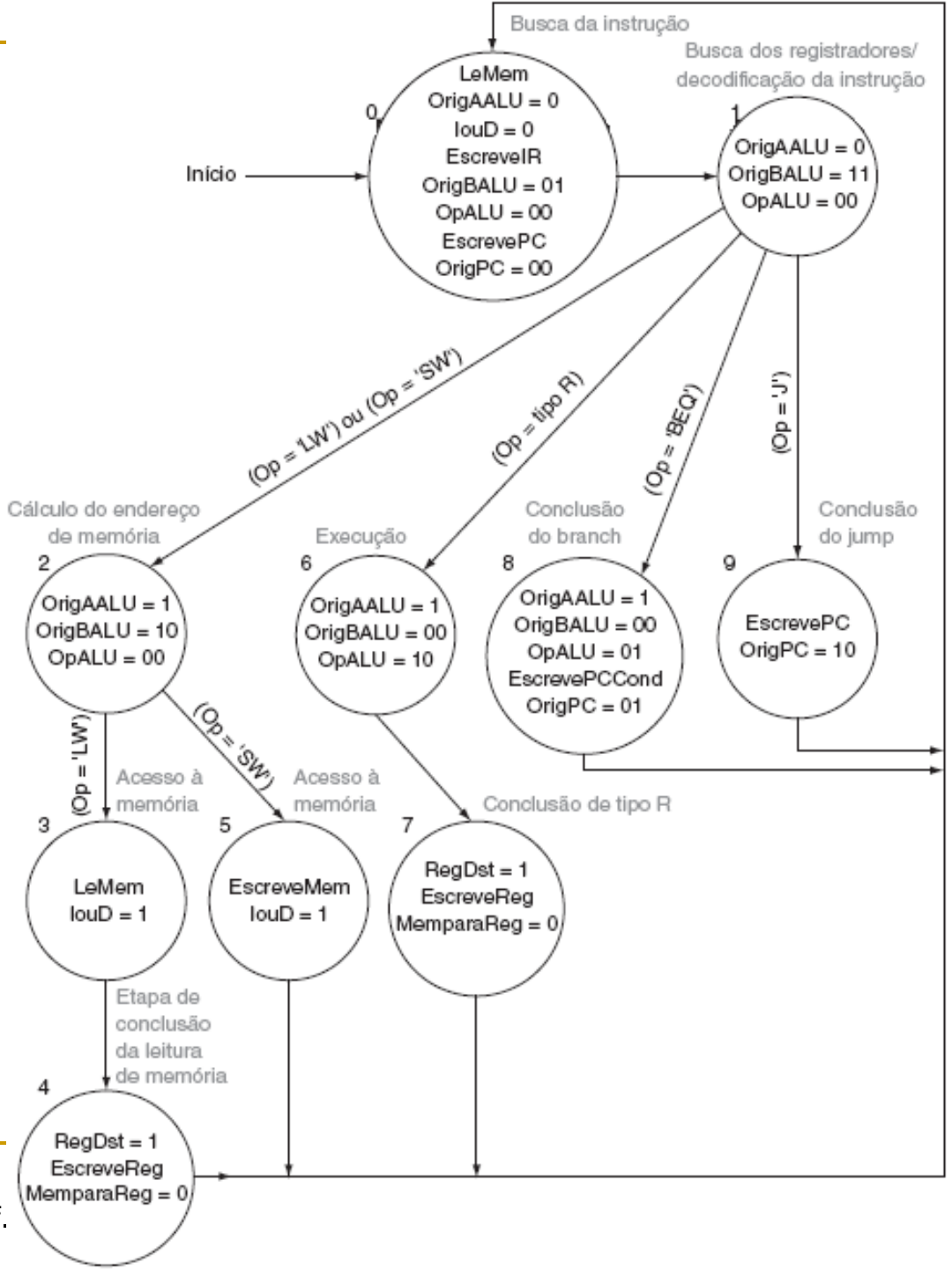
Etapas de Execução

- Vejamos um resumo:

Etapa	Ação para instruções tipo R	Ação para instruções de acesso à memória	Ação para desvios	Ação para jumps
Busca da instrução	IR <= Memória[PC] PC <= PC + 4			
Decodificação da instrução e busca dos registradores	A <= Reg[IR[25:21]] B <= Reg[IR[20:16]] SaídaALU <= PC + (estende-sinal (IR[15:0]) << 2)			
Execução, cálculo do endereço, conclusão do desvio/jump	SaídaALU <= A op B	SaídaALU <= A + estende-sinal (IR[15:0])	if (A == B) PC <= SaídaALU	PC <= {PC [31:28], (IR[25:0]),2'b00)}
Acesso à memória ou conclusão de instrução tipo R	Reg[IR[15:11]]<= SaídaALU	Load: MDR <= Memória[SaídaALU] ou Store: Memória [SaídaALU] <= B		
Conclusão da leitura da memória		Load: Reg[IR[20:16]] <= MDR		

Etapas de Execução

- Especificação gráfica:



Exercício

- Quantos ciclos serão necessários para executar o código a seguir:

```
lw $t2, 0($t3)
lw $t3, 4($t3)
beq $t2, $t3, Label #considere not
add $t5, $t2, $t3
sw $t5, 8($t3)
Label: ...
```

Bibliografia

- PATERSON, D. A. & HENNESSY, J. L. *Organização e Projeto de Computadores: a interface hardware/software,* Editora Campus. 3ª ed. RJ: 2005.