Organização e Arquitetura de Computadores I

Caminho de Dados

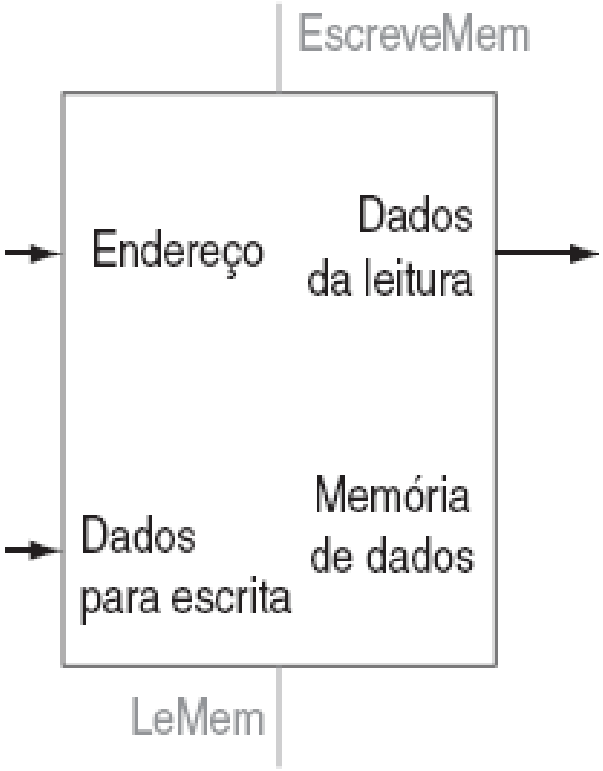
Sumário

- 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

Construindo um Caminho de Dados

- Memória de Dados:
  - Uma unidade de memória de dados é um elemento de estado com entradas para os endereços e os dados de escrita, e uma única saída para o resultado da leitura.
  - Possui controles de leitura e escrita separados, embora apenas um deles pode ser ativado em qualquer *clock* específico.

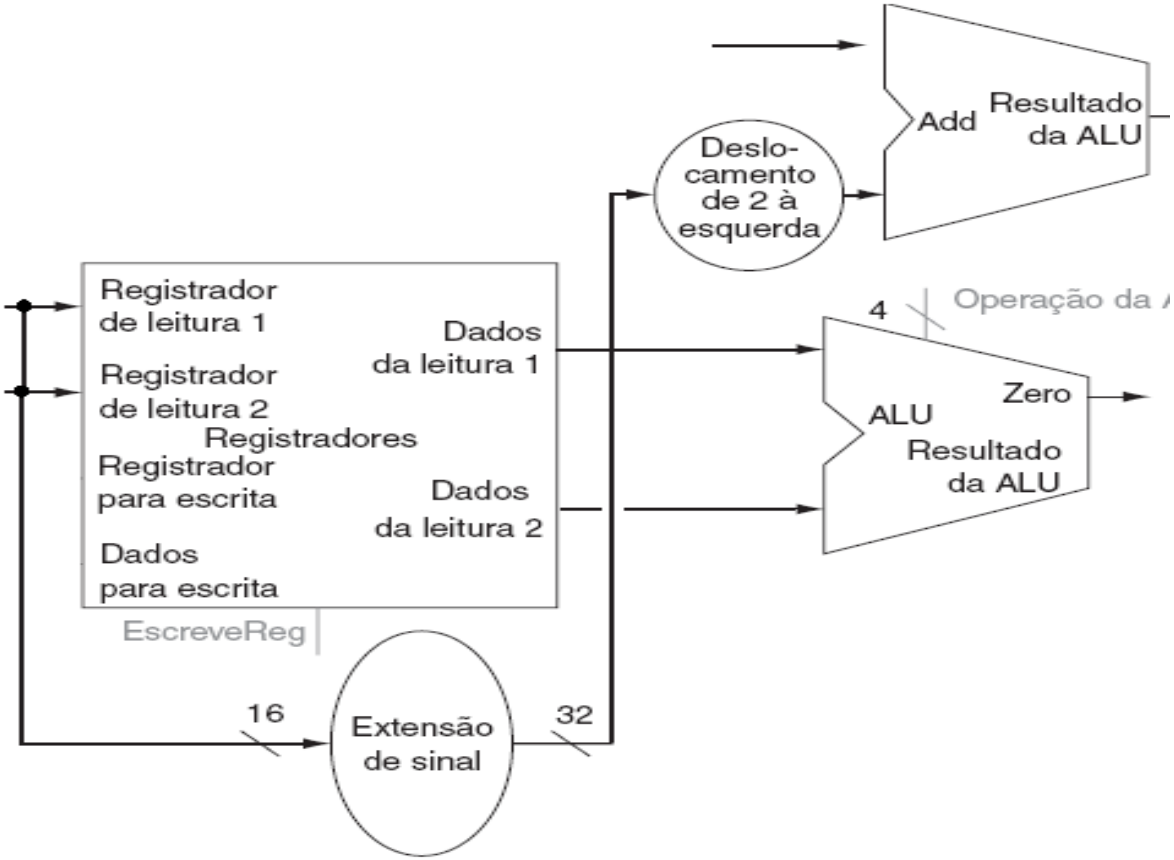
Construindo um Caminho de Dados



Construindo um Caminho de Dados

- Endereço de destino do desvio:
  - Endereço especificado em um desvio, que se torna o novo contador de programa (PC) se o desvio for tomado;
  - O destino do desvio é dado pela soma do campo *offset* da instrução e o endereço da instrução seguinte ao desvio;
  - Forma de instrução de desvio “beq $t1, $t2, *offset*”.

Construindo um Caminho de Dados



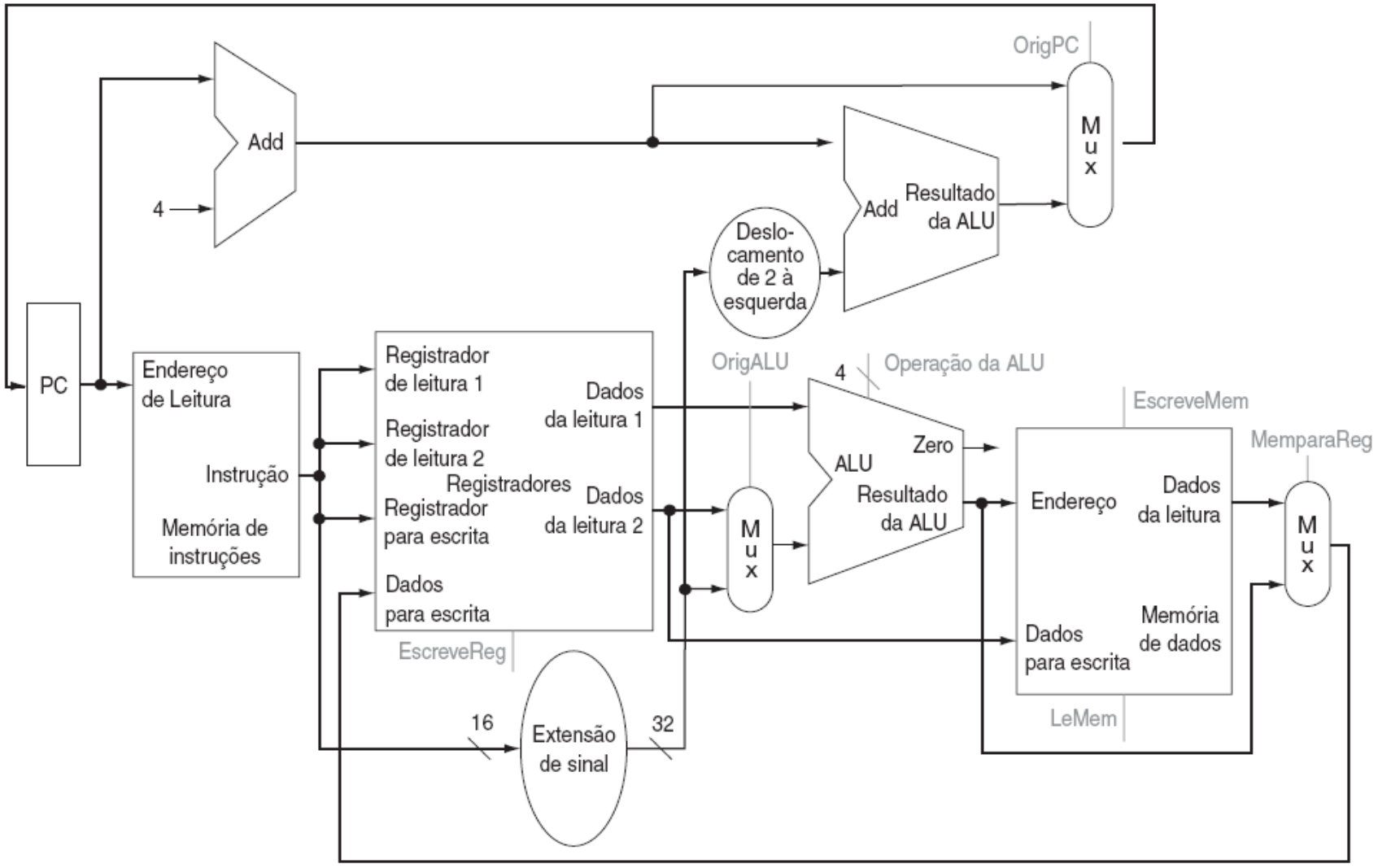
Construindo um Caminho de Dados

- 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

- Além de calcular o endereço de destino do desvio, também precisamos determinar se a próxima instrução é a instrução que segue sequencialmente ou a instrução no endereço de destino do desvio.
- Quando a condição é verdadeira, o endereço de destino do desvio se torna o novo PC, dizemos que o **desvio é tomado**.
- Quando a condição é falsa, o PC incrementado deve substituir o PC atual, dizemos que o **desvio é não tomado**.

Organização e Arquitetura de Computadores I



O Controle da ULA

- A ULA possui quatro entradas de controle.
- Esses bits não foram codificados, portanto, apenas 6 das 16 combinações possíveis são usadas neste subconjunto:

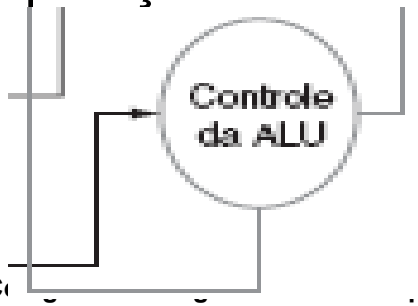
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

O Controle da ULA

- 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.
- A função NOR é necessária para outras partes do conjunto de instruções MIPS.
- Para as instruções lw ou sw, a ULA é usada para calcular o endereço de memória por adição.
- Para instruções *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 que tenha como entradas o campo funct da instrução e um campo *control* de 2 bits, que chamamos OpALU.
- OpALU indica se a operação a ser realizada deve ser:
  - add (00) para loads e stores;
  - subtract (01) para beq;
  - Determinada pela operação codificada no campo funct (10);



O Controle da ULA

- É utilizado uma técnica comum, a unidade de controle principal gera os bits de OpALU, que então, são usados como entrada para o controle da ULA que gera os sinais reais para controlar a ULA. Essa técnica visa diminuir a complexidade da unidade de controle central e aumentar a sua velocidade utilizando várias unidades de controle menores.

O Controle da ULA

- Tabela verdade para os três bits de controle da ULA (OpALU):

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

Projeto da Unidade de Controle Principal

- Para começar o projeto, precisamos identificar os campos de uma instrução e as linhas de controle necessárias para o caminho de dados. Para entender como conectar os campos de uma instrução:
  - Instruções do tipo R;
  - Instruções de desvio;
  - Instruções de acesso à memória.

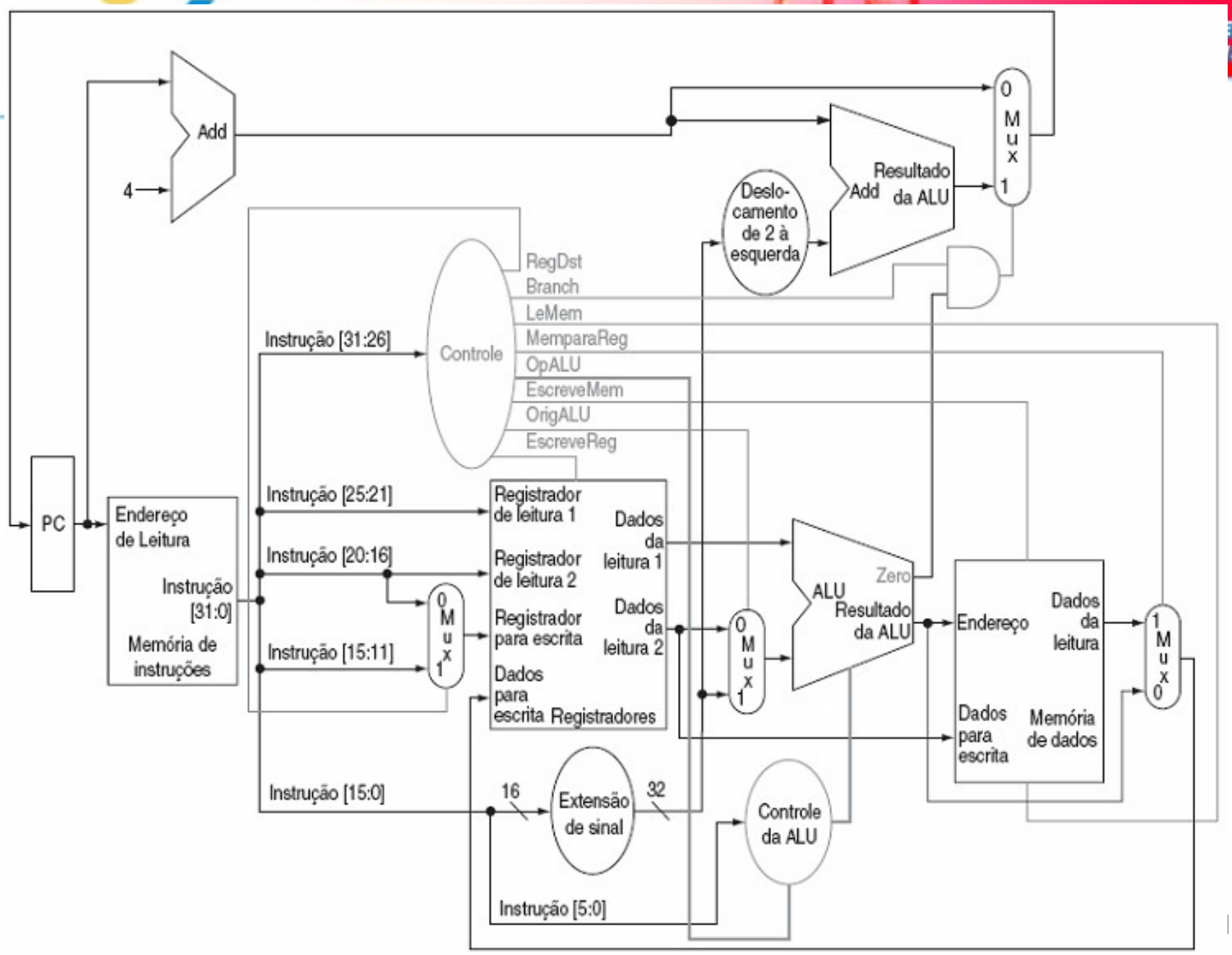
Projeto da Unidade de Controle Principal

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 (**opcode**), está sempre contido nos bits [31:26]. Iremos nos referir a esse campo como Op[5:0].

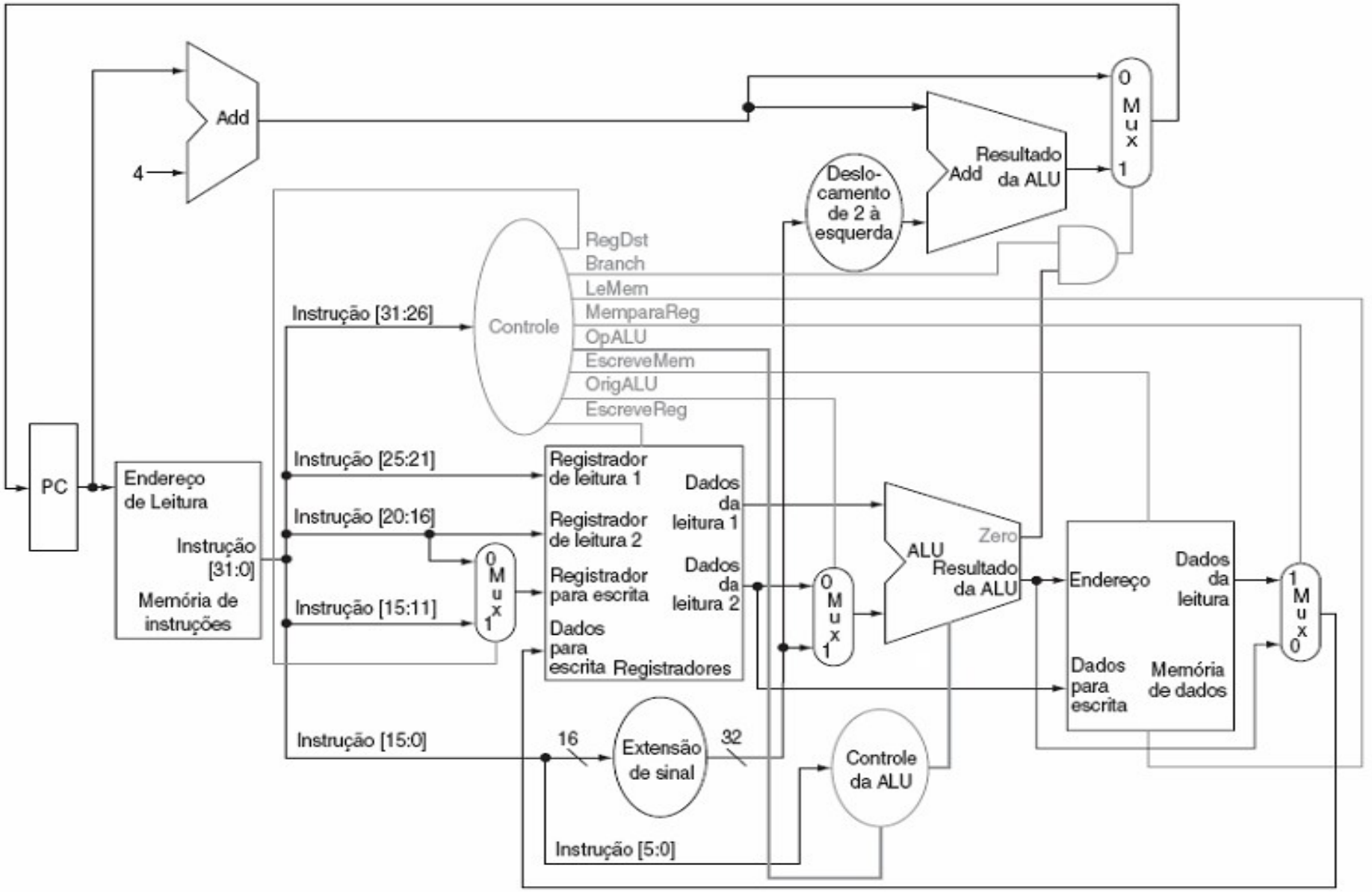
Projeto da Unidade de Controle Principal

- Os dois registradores a serem lidos sempre são especificados pelos campos rs e rt, nas posições [25:21] e [20:16], nas instruções do tipo R, *branch equal* ou *store*.
- O registrador de base para a instrução de load ou store está sempre nas posições de bit [25:21].
- O deslocamento (16 bits) para beq, *load* e *store* está sempre nas posições [15:0].
- O registrador de destino está em um dos dois lugares. Para load ele está nas posições [20:16] (rt). Para instruções do 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
Registradores
Dados
da
leitura 1
Dados
da
leitura 2
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

- Antes de mais nada, é necessário entender como cada instrução usa o caminho de dados.
- Todas as operações ocorrem em 1 ciclo de *clock*.
- Podemos pensar em quatro etapas para executar uma instrução do tipo R (add $t1, $t2, $t3); essas etapas são ordenadas pelo fluxo de informação:
  - A instrução é buscada e o PC é incrementado;
  - Dois registradores, $t2 e $t3, são lidos do banco de registradores e a unidade de controle principal calcula a definição das linhas de controle também durante essa etapa;
  - A ULA opera nos dados lidos do banco de registradores, usando o código de função (bits 5:0, campo funct) para gerar a função da ULA;
  - O resultado da ULA é escrito no banco de registradores usando os bits 15:11 da instrução para selecionar o registrador de destino ($t1).


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
Controle
Instrução [31:26]
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
Registradores
Dados
da
leitura 1
Dados
da
leitura 2
Zero
ALU
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

Linhas de Controle

OpALU

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

Operação do Caminho de Dados

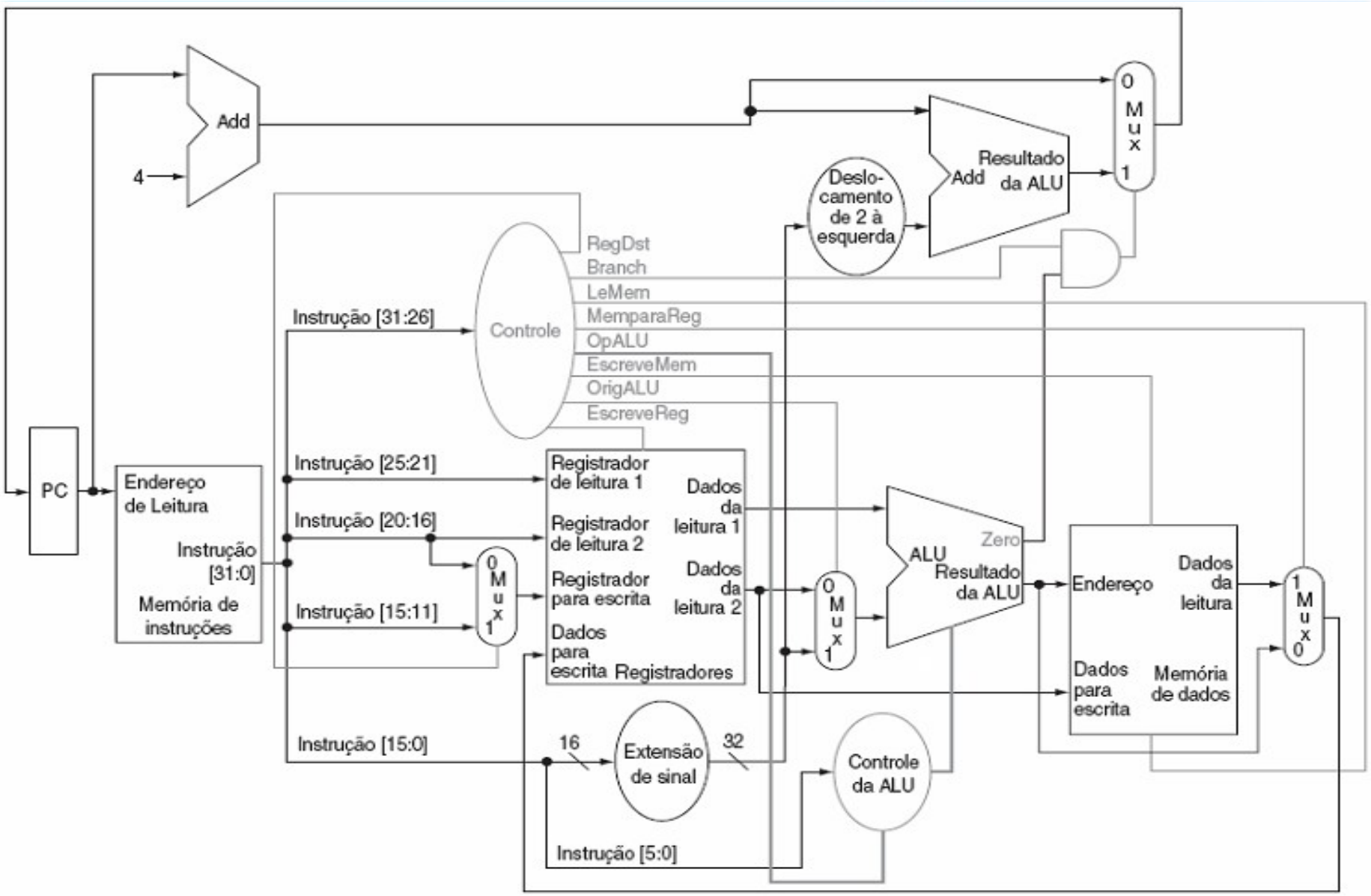
- Linhas de controle para uma instrução do tipo R:
  - Os campos registradores de origem são rs e rt e o campo registrador de destino é rd, isso especifica como os sinais OrigALU e RegDst são definidos;
  - Esse tipo de instrução escreve em um registrador (EscreveReg=1), mas não escreve ou lê a memória de dados (LeMem=0, EscreveMem=0);
  - Sendo *Branch*=0, o PC=PC+4;

Operação do Caminho de Dados

- Como exemplo vejamos a instrução abaixo:

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).


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
Controle
Instrução [31:26]
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
Registradores
Dados
da
leitura 1
Dados
da
leitura 2
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

- Linhas de Controle

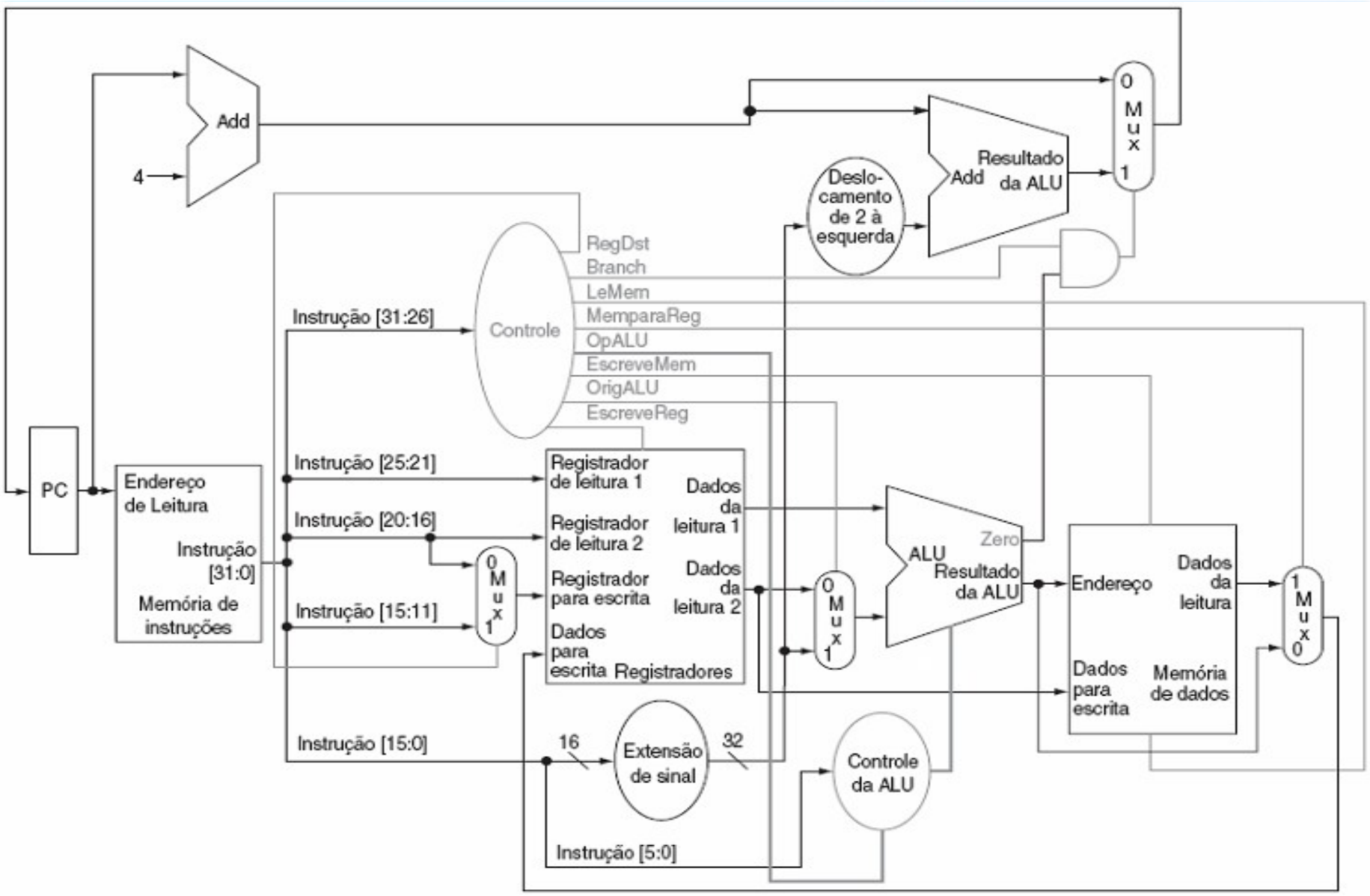
Instrução	RegDst	OrigALU	Mempara Reg	Escreve Reg	Le Mem	Escreve Mem	Branch	ALUOp1	ALUOp0
formato R	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0

Operação do Caminho de Dados

- Como exemplo vejamos a instrução abaixo:

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.


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
Controle
Instrução [31:26]
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
Registradores
Dados
da
leitura 1
Dados
da
leitura 2
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]

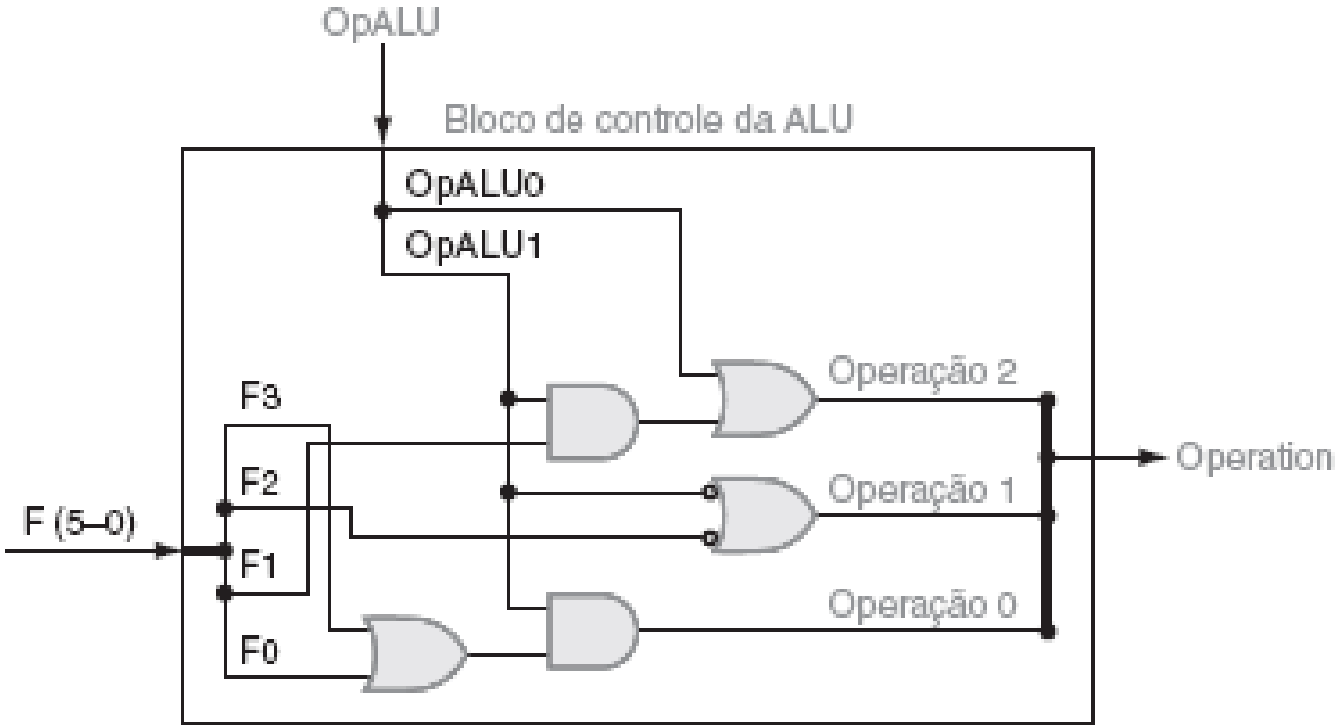
Operação do Caminho de Dados

- 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	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

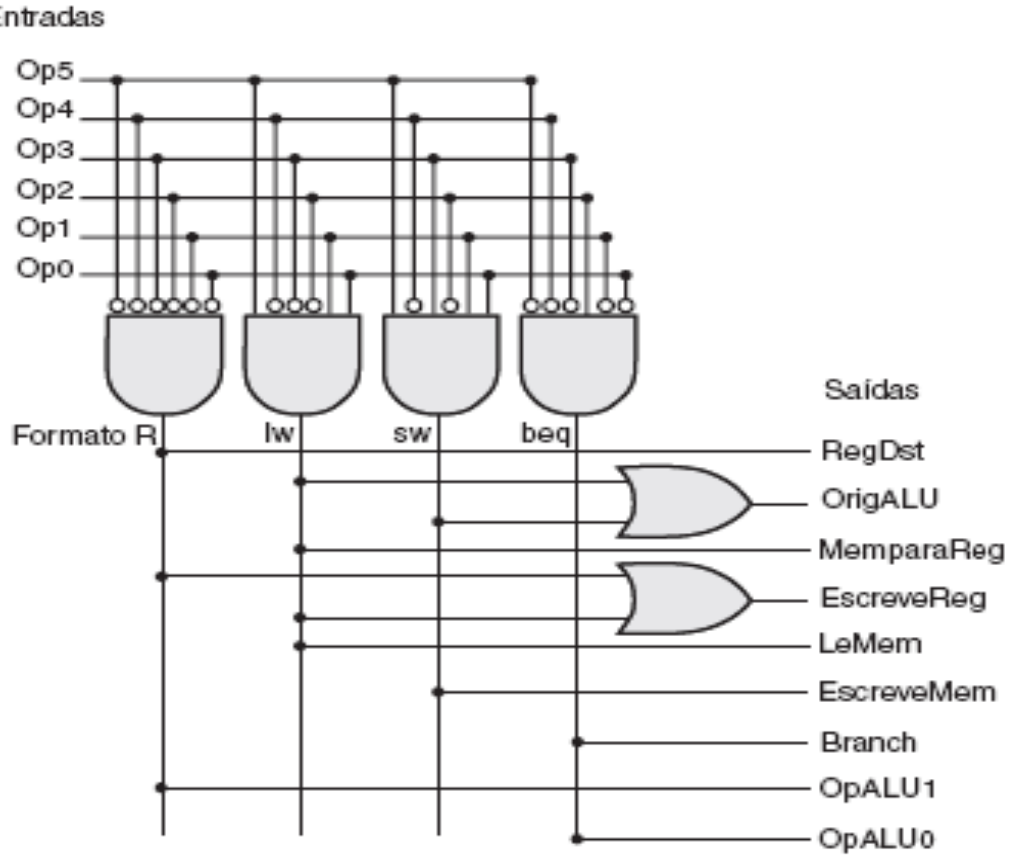
Operação do Caminho de Dados

- Detalhes da lógica combinacional do controle da ALU:



Operação do Caminho de Dados

- Detalhe ontrole:



Por que Multiciclo?

- Na implementação de ciclo único toda instrução opera em 1 *clock* de uma duração fixa.
- O ciclo de *clock* precisa ter a mesma duração para cada instrução. **O ciclo de *clock* é determinado pelo caminho mais longo da máquina.**
- O desempenho geral de uma implementação de ciclo único provavelmente não será muito bom, já que várias das classes de instrução poderiam ficar em um ciclo de *clock* mais curto.

Uma Implementação Multiciclo

- Se dividirmos cada instrução em uma série de etapas correspondentes às operações das unidades funcionais necessárias, podemos usar essas etapas para criar uma implementação multiciclo.
- Em uma implementação multiciclo, cada etapa da execução levará 1 ciclo de *clock*.
- A implementação multiciclo permite que uma unidade funcional seja usada mais de uma vez por instrução, desde que seja usada em diferentes ciclos de *clock*.

Uma Implementação Multiciclo

- Visão de alto nível de um caminho multiciclos: