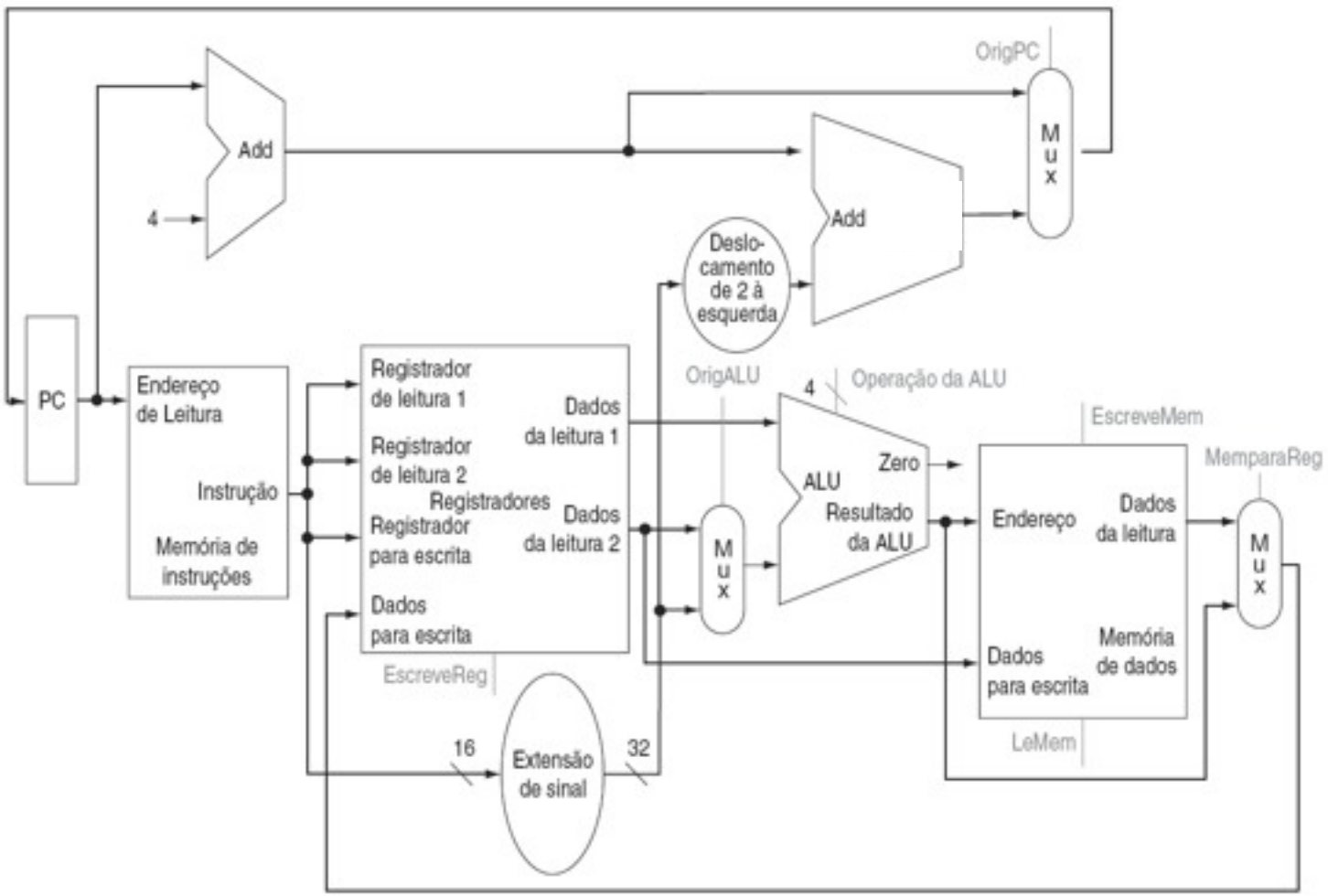
Universidade de Brasília
Departamento de Ciência da Computação

MIPS Uniciclo
Unidade de Controle

MIPS Uniciclo

- Foram desenvolvidas, na verdade, três tipos de unidades operativas:
  - uma para instruções no formato R (*add*, *sub*, *and, or, xor, slt*)
  - uma para instruções de acesso a memória *lw* e *sw* - formato I
  - uma para *instruções condicionais (beq)* – formato I
- Na via de dados mais simples deve-se propor executar as instruções em um único período do *clock.*
- **Isto quer dizer que nenhum dos recursos pode ser usado mais de uma vez por instrução.**

Unidade Operativa: Quase-Final



Controle do MIPS

- Cada operação requer a utilização adequada dos recursos do MIPS
- Ex: *add $t0, $s1, $s2*
  - é necessário que os endereços dos operandos $s1 e $s2 sejam enviados ao banco de registradores
  - Da mesma maneira, o endereço do registrador destino ($t0) deverá ser informando ao banco de registradores
  - Uma vez que o banco de registradores disponibilize os valores de $s1 e $s2, estes deverão ser encaminhados à ULA
  - Posteriormente, o resultado deverá ser escrito no banco de registradores (registrador $t0)

add $8, $17, $18 **Formato de instrução:**

000000	10001	10010	01000	00000	100000
op	rs	rt	rd	shamt	funct

Exemplo de projeto de controle hierárquico: **Controle da ULA**

- As operações da ULA são controladas por um código de 4 bits:

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

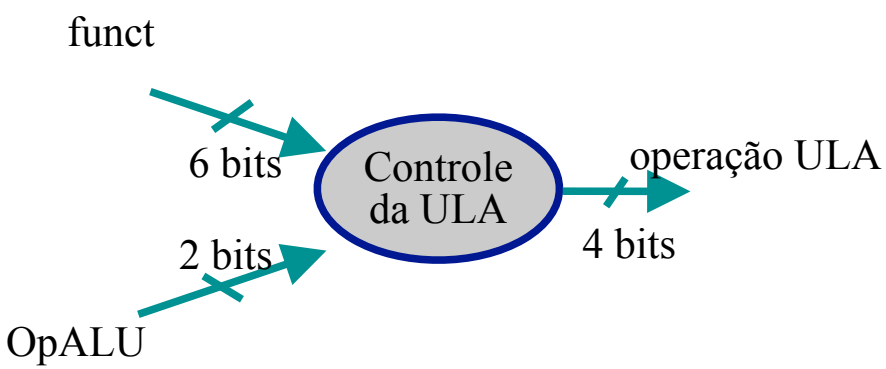
- As instruções **lw** e **sw** utilizam a operação de **soma** da ULA
- As instruções do **tipo R** utilizam uma das **5 operações**
- A instrução **beq** utiliza a **subtração** da ULA
- Nor não é usada nesta implementação.

Identificação da Operação

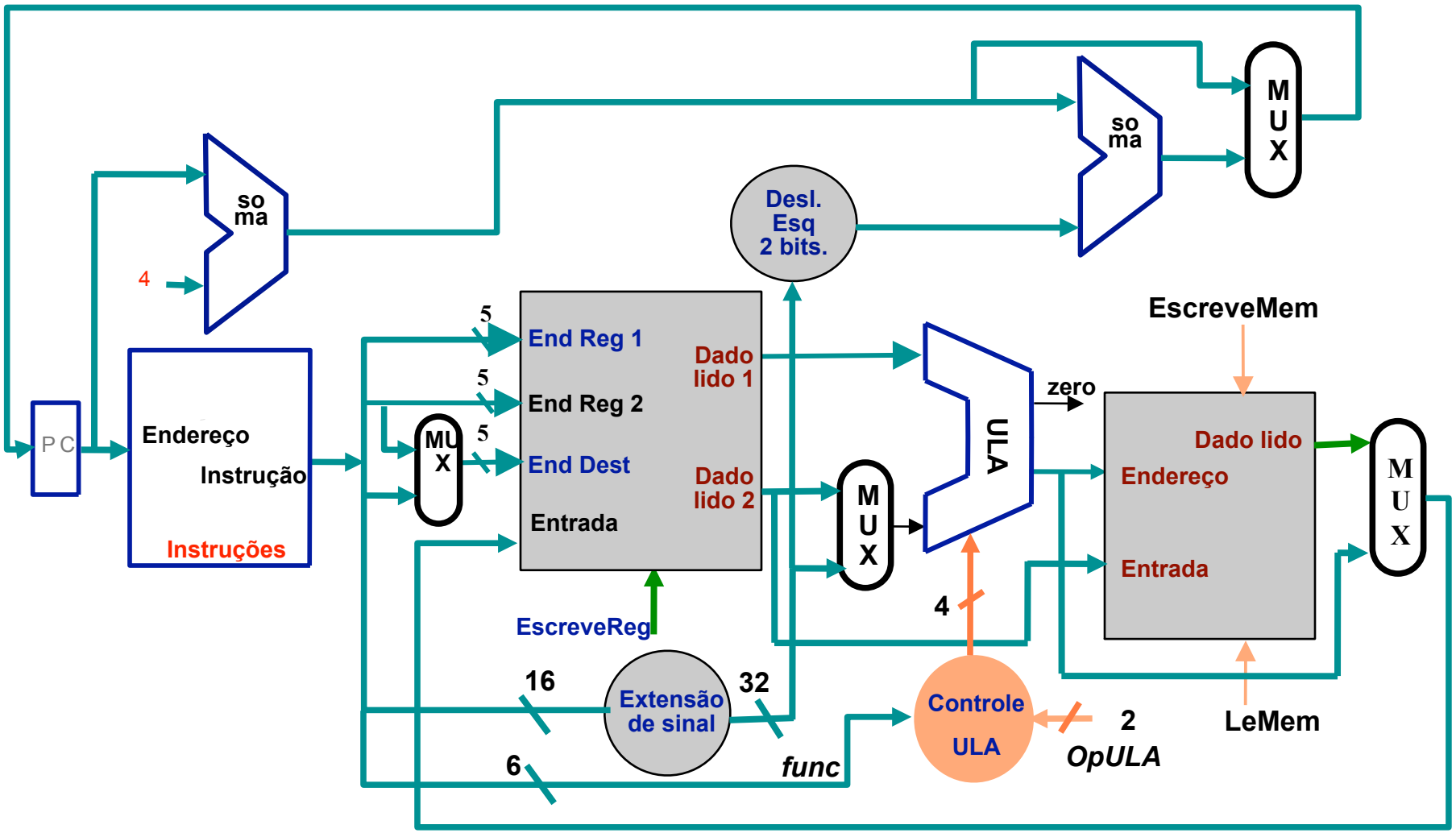
- O campo *funct*, de 6 bits (no formato R) indica que tipo de operação aritmética/lógica será executada:

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

- Vamos definir 2 bits para identificar cada uma das categorias de instruções (opALU):
  - 00 acesso à memória (lw, sw) [**add**]
  - 01 desvio (beq, bne) [**sub**]
  - 10 lógico-aritméticas ([**add, sub, and, or, slt**])



Lógica de Controle da ULA



Circuito de Controle

- Projeto do Circuito de controle da ULA
  - xxx indica irrelevâncias (*don't cares*)

Código de operação da instrução	OpALU	Operação da Instrução	Campo Função	Operação desejada da ULA	Entrada de controle da ULA
LW	00	load word	xxxxxx	soma	0010
SW	00	store word	xxxxxx	soma	0010
Beq	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 less than	101010	set less than	0111

Controle da ULA

Com base nas definições anteriores podemos montar a Tabela Verdade do circuito que gera os 4 bits de controle da ULA

OpALU		Campo Func.						Operação			
OpALU1	OpALU0	F5	F4	F3	F2	F1	F0	Op3	Op2	Op1	Op0
0	0	X	X	X	X	X	X	0	0	1	0
0	1	X	X	X	X	X	X	0	1	1	0
1	0	X	X	0	0	0	0	0	0	1	0
1	0	X	X	0	0	1	0	0	1	1	0
1	0	X	X	0	1	0	0	0	0	0	0
1	0	X	X	0	1	0	1	0	0	0	1
1	0	X	X	1	0	1	0	0	1	1	1

Equações Lógicas

- Uma versão simplificada do projeto lógico:

OpALU		Campo Func.						Operação			
OpALU1	OpALU0	F5	F4	F3	F2	F1	F0	Op3	Op2	Op1	Op0
0	0	X	X	X	X	X	X	0	0	1	0
0	1	X	X	X	X	X	X	0	1	1	0
1	0	X	X	0	0	0	0	0	0	1	0
1	0	X	X	0	0	1	0	0	1	1	0
1	0	X	X	0	1	0	0	0	0	0	0
1	0	X	X	0	1	0	1	0	0	0	1
1	0	X	X	1	0	1	0	0	1	1	1

Op3 = 0

Equações Lógicas

- Uma versão simplificada do projeto lógico:

OpALU		Campo Func.						Operação			
OpALU1	OpALU0	F5	F4	F3	F2	F1	F0	Op3	Op2	Op1	Op0
0	0	X	X	X	X	X	X	0	0	1	0
0	1	X	X	X	X	X	X	0	1	1	0
1	0	X	X	0	0	0	0	0	0	1	0
1	0	X	X	0	0	1	0	0	1	1	0
1	0	X	X	0	1	0	0	0	0	0	0
1	0	X	X	0	1	0	1	0	0	0	1
1	0	X	X	1	0	1	0	0	1	1	1

$$Op2 = opALU0 + opALU1 \cdot F1$$

Equações Lógicas

- Uma versão simplificada do projeto lógico:

OpALU		Campo Func.						Operação			
OpALU1	OpALU0	F5	F4	F3	F2	F1	F0	Op3	Op2	Op1	Op0
0	0	X	X	X	X	X	X	0	0	1	0
0	1	X	X	X	X	X	X	0	1	1	0
1	0	X	X	0	0	0	0	0	0	1	0
1	0	X	X	0	0	1	0	0	1	1	0
1	0	X	X	0	1	0	0	0	0	0	0
1	0	X	X	0	1	0	1	0	0	0	1
1	0	X	X	1	0	1	0	0	1	1	1

$$Op1 = \overline{opALU1} + \overline{F2}$$

Equações Lógicas

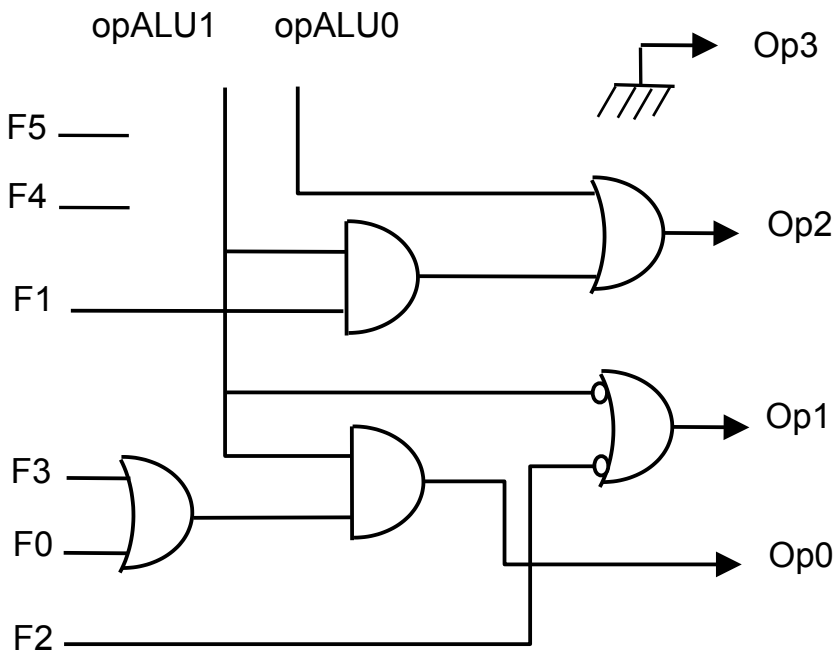
- Uma versão simplificada do projeto lógico:

OpALU		Campo Func.						Operação			
OpALU1	OpALU0	F5	F4	F3	F2	F1	F0	Op3	Op2	Op1	Op0
0	0	X	X	X	X	X	X	0	0	1	0
0	1	X	X	X	X	X	X	0	1	1	0
1	0	X	X	0	0	0	0	0	0	1	0
1	0	X	X	0	0	1	0	0	1	1	0
1	0	X	X	0	1	0	0	0	0	0	0
1	0	X	X	0	1	0	1	0	0	0	1
1	0	X	X	1	0	1	0	0	1	1	1

Op0 = opALU1(F3 + F0)

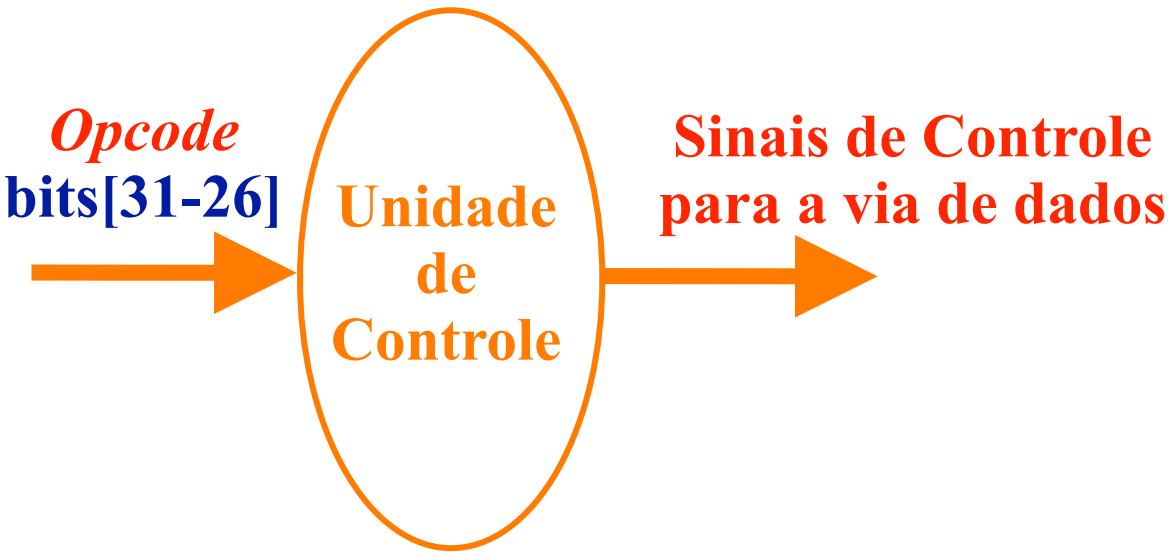
Equações Lógicas

- Uma versão simplificada do projeto lógico:
  - **Op3=0**
  - **Op2 = opALU0 + opALU1·F1**
  - **Op1 = ~~opALU1~~ + ~~F2~~**
  - **Op0 = opALU1(F3 + F0)**



Unidade de Controle Uniciclo

- A unidade de controle deve, a partir do código da instrução, fornecer os sinais que realizam as instruções na unidade operativa

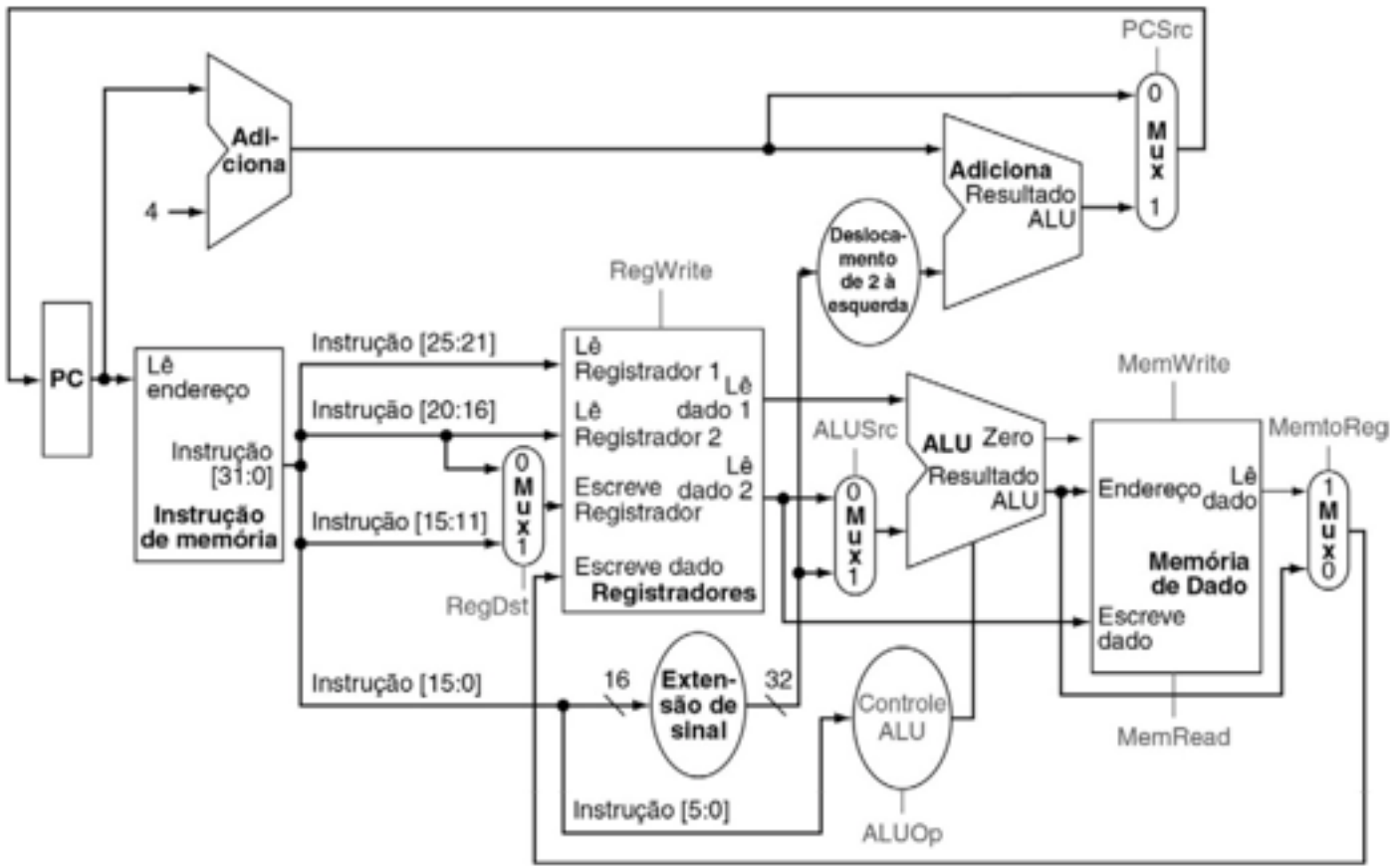


Entrada da Unidade de Controle

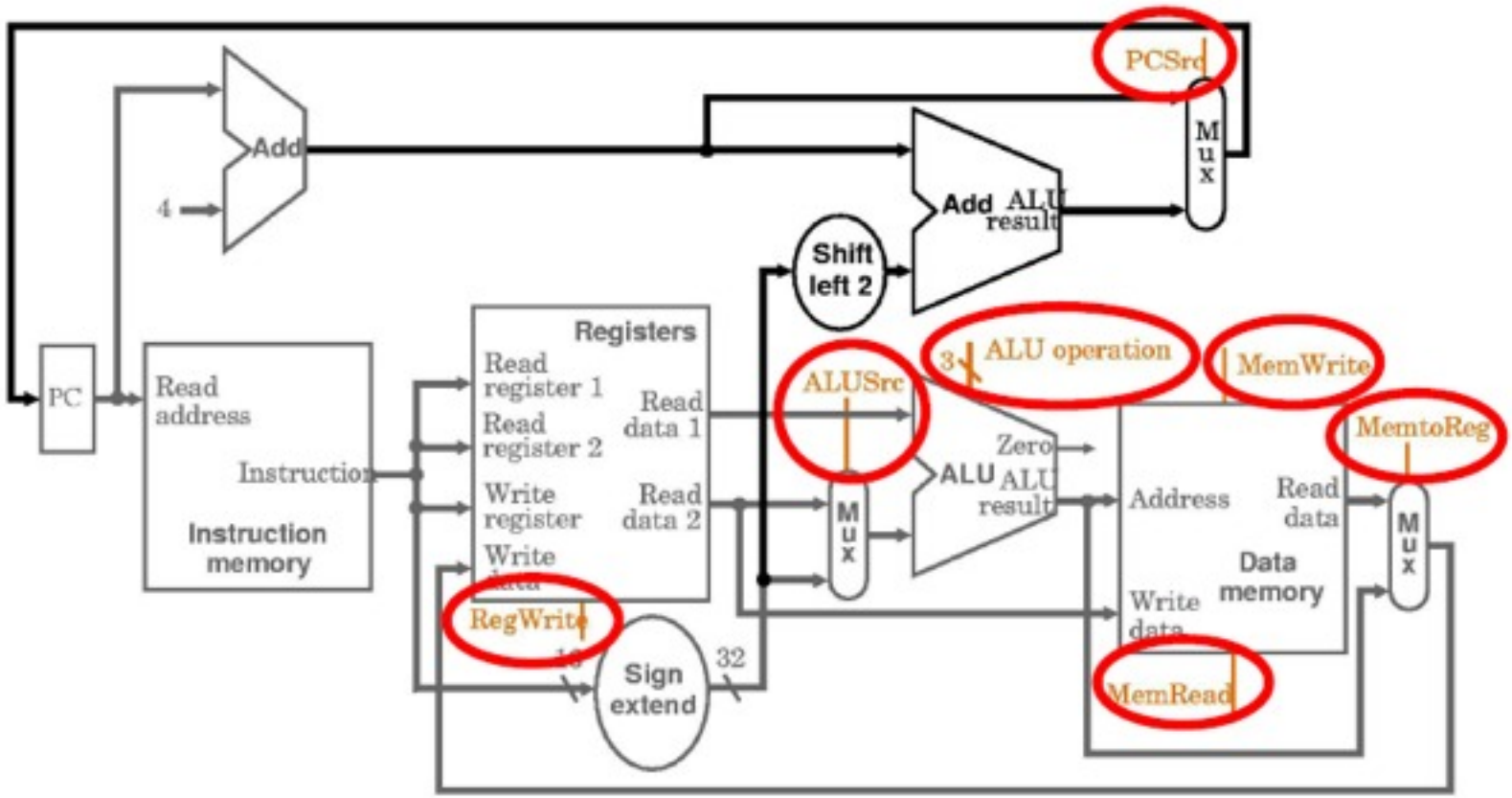
- **as informações necessárias a execução de uma instrução são retiradas da própria instrução**
  - O *opcode* (código de operação) sempre está nos bits [31-26]=Op[5:0]
  - Os 2 registradores a serem lidos (rs e rt) sempre estão nas posições [25-21] e [20-16] (tipo-R, beq e sw)
  - O registrador base (rs) para as instruções lw e sw sempre está especificado nas posições [25-21]
  - Os 16 bits de deslocamento para as instruções beq, lw e sw estão sempre nas posições [15-0]
  - O registrador destino está em uma das duas posições
    - [20-16] para lw (registrador rt)
    - [15-11] para instruções aritméticas/lógicas (registrador rd)

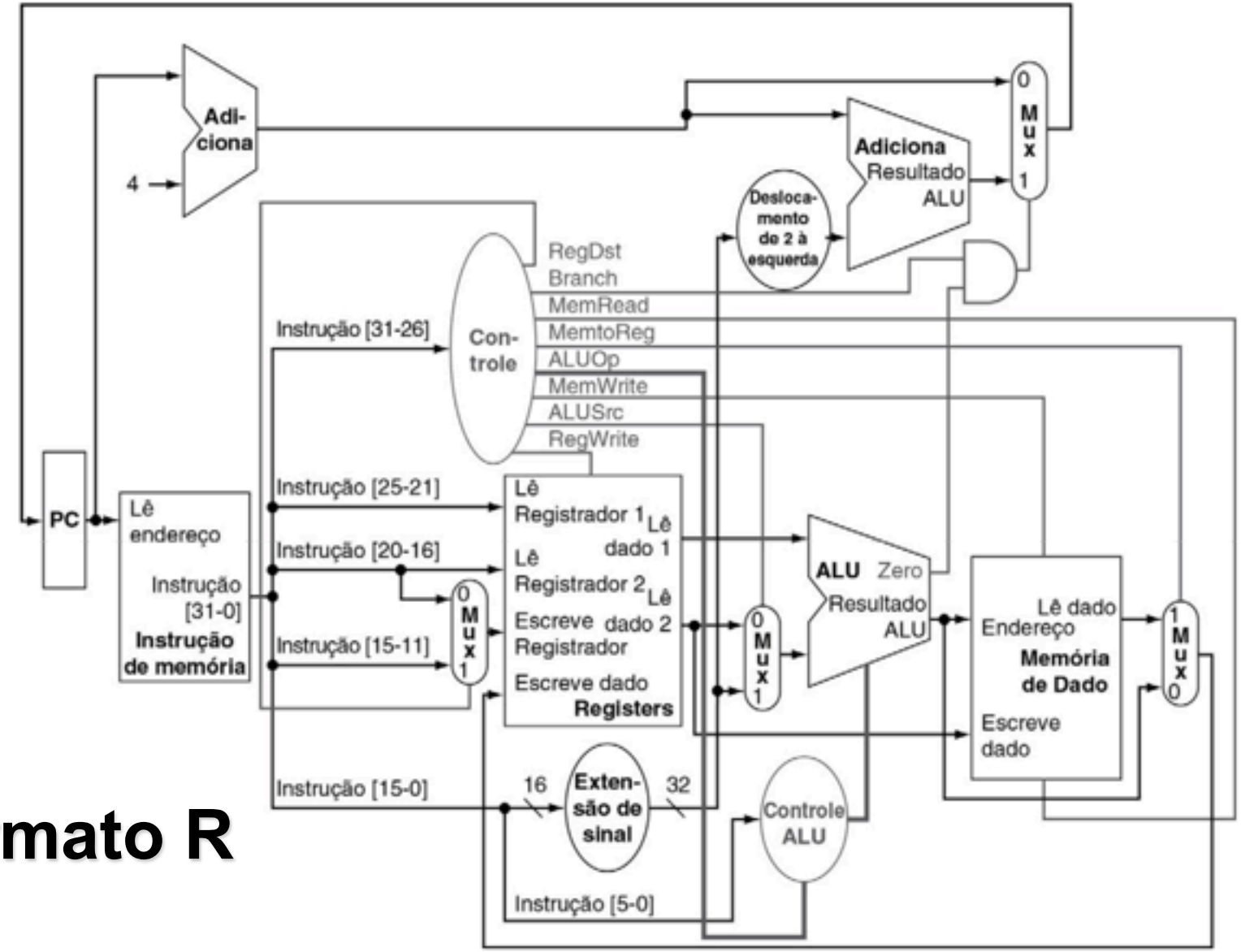
Sinais de Controle

- A unidade de controle deve prover:
  - sinais para os multiplexadores
  - sinais de leitura e escrita para as memórias
  - seleção da operação da ULA
  - controle do novo endereço a ser carregado no PC, para instruções de salto



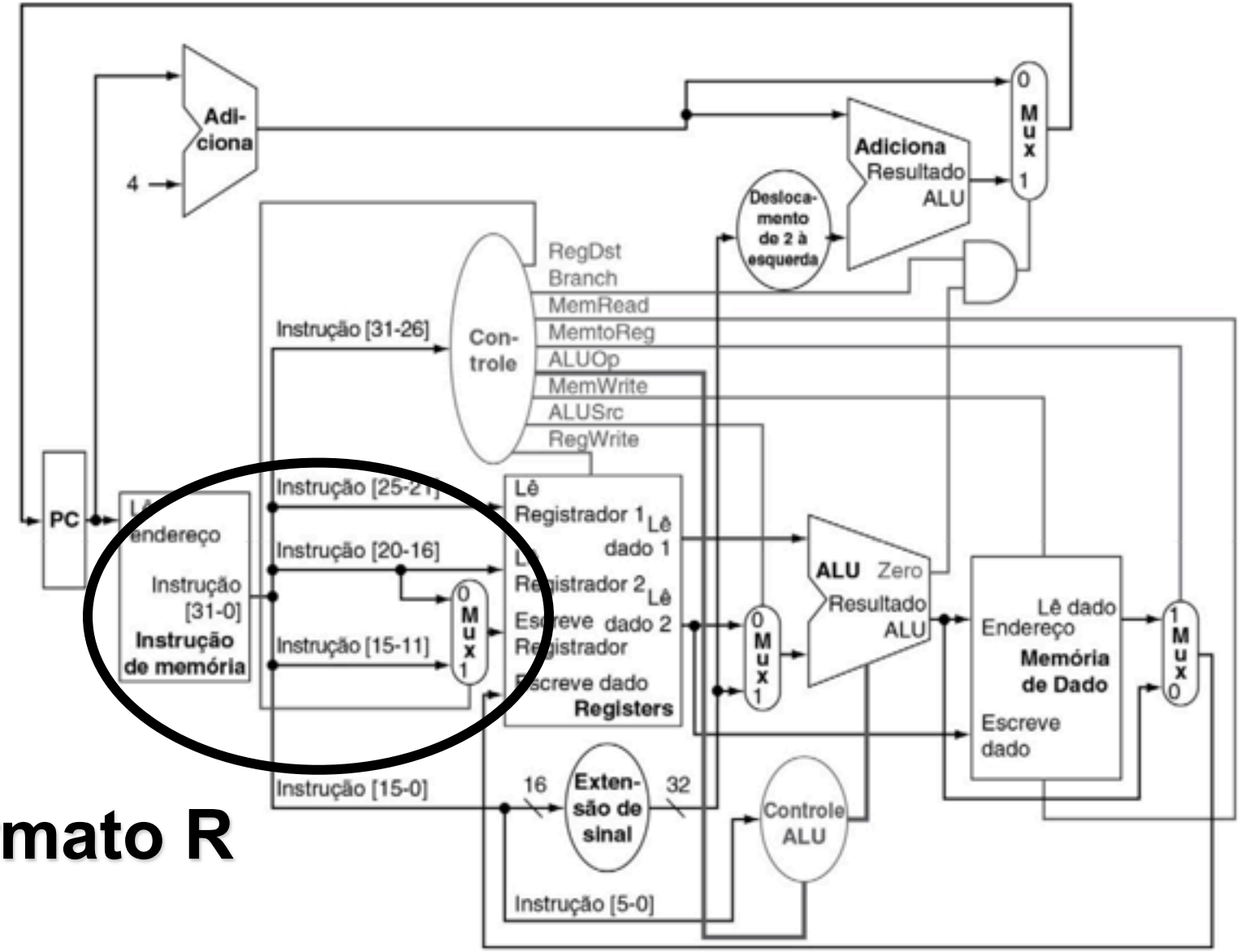
E o controle: sinais que precisamos gerar





formato R

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
formato R							

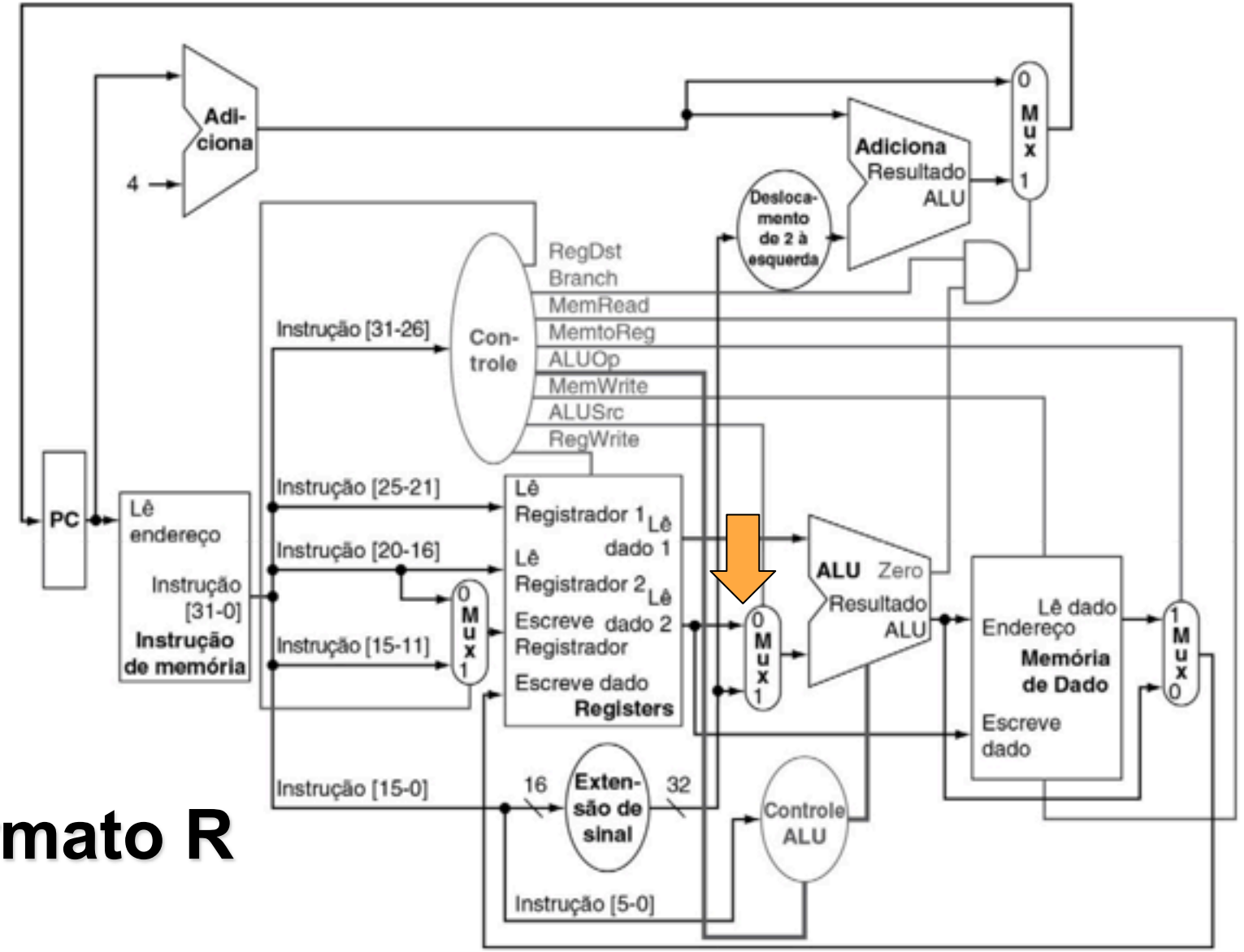


formato R

Instrução	RegDs
formato R	1

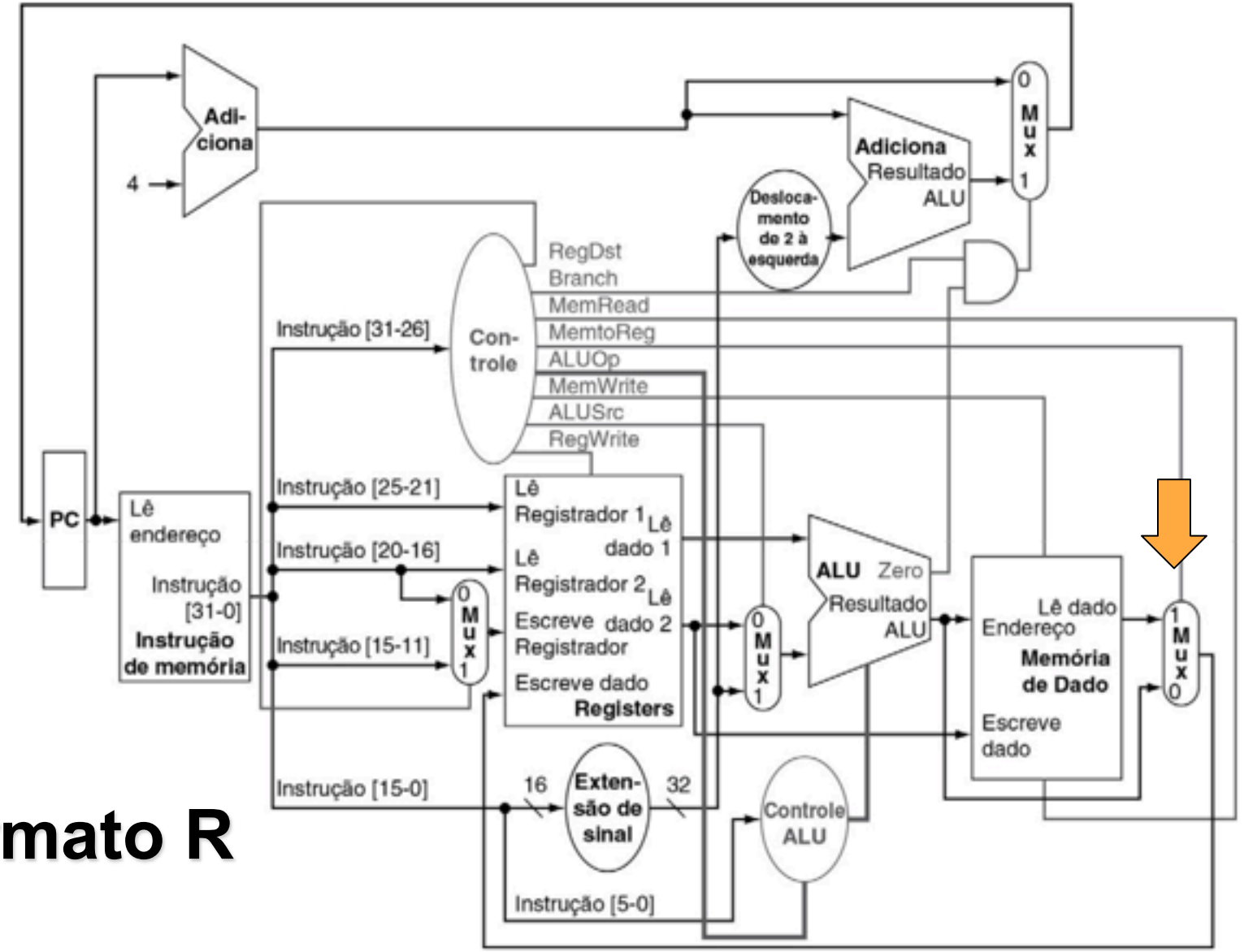
Campo	0	rs	rt	rd	shamt	funct
Posição de bit	31:26	25:21	20:16	15:11	10:6	5:0

a. Instrução do tipo R



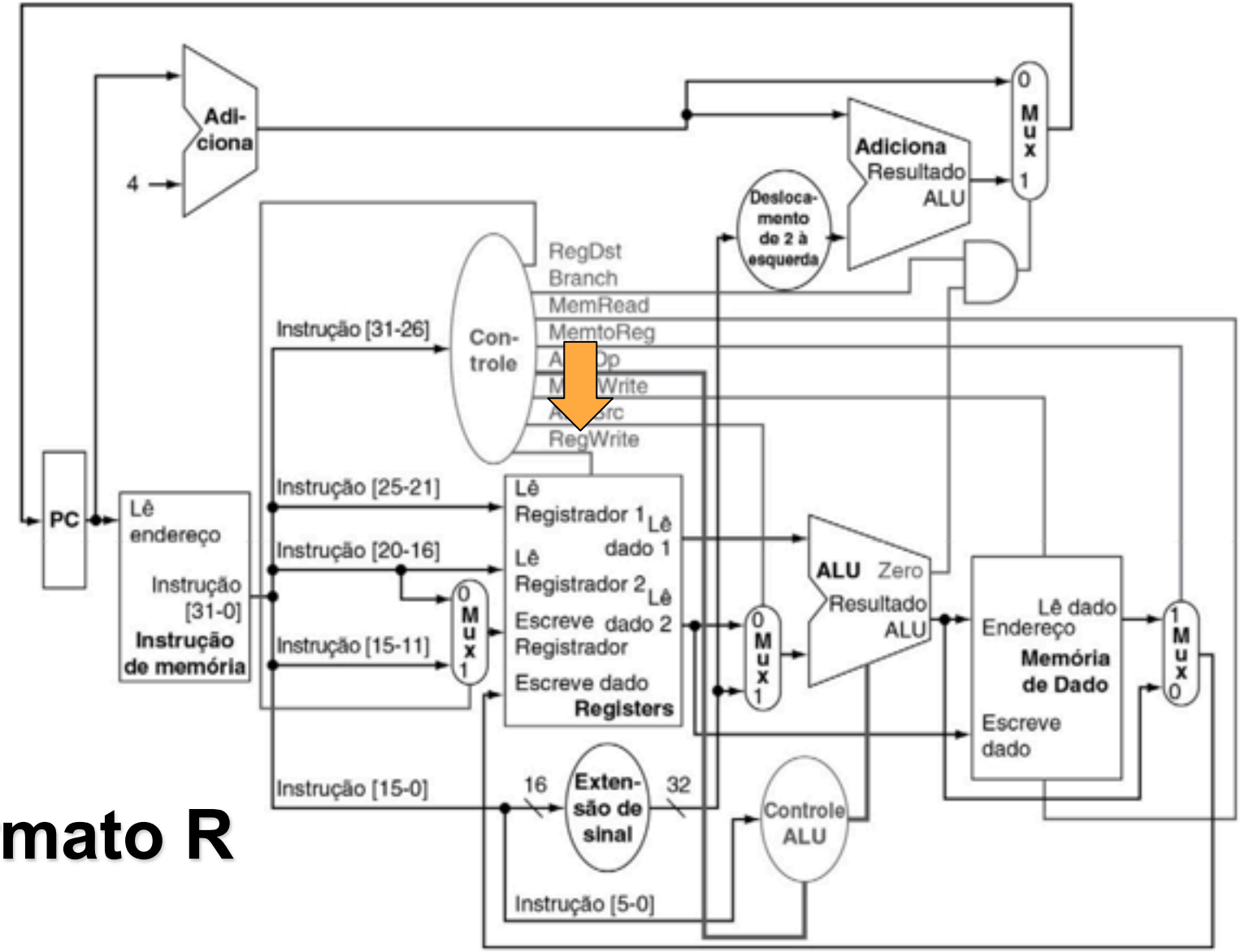
formato R

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
formato R	1	0					



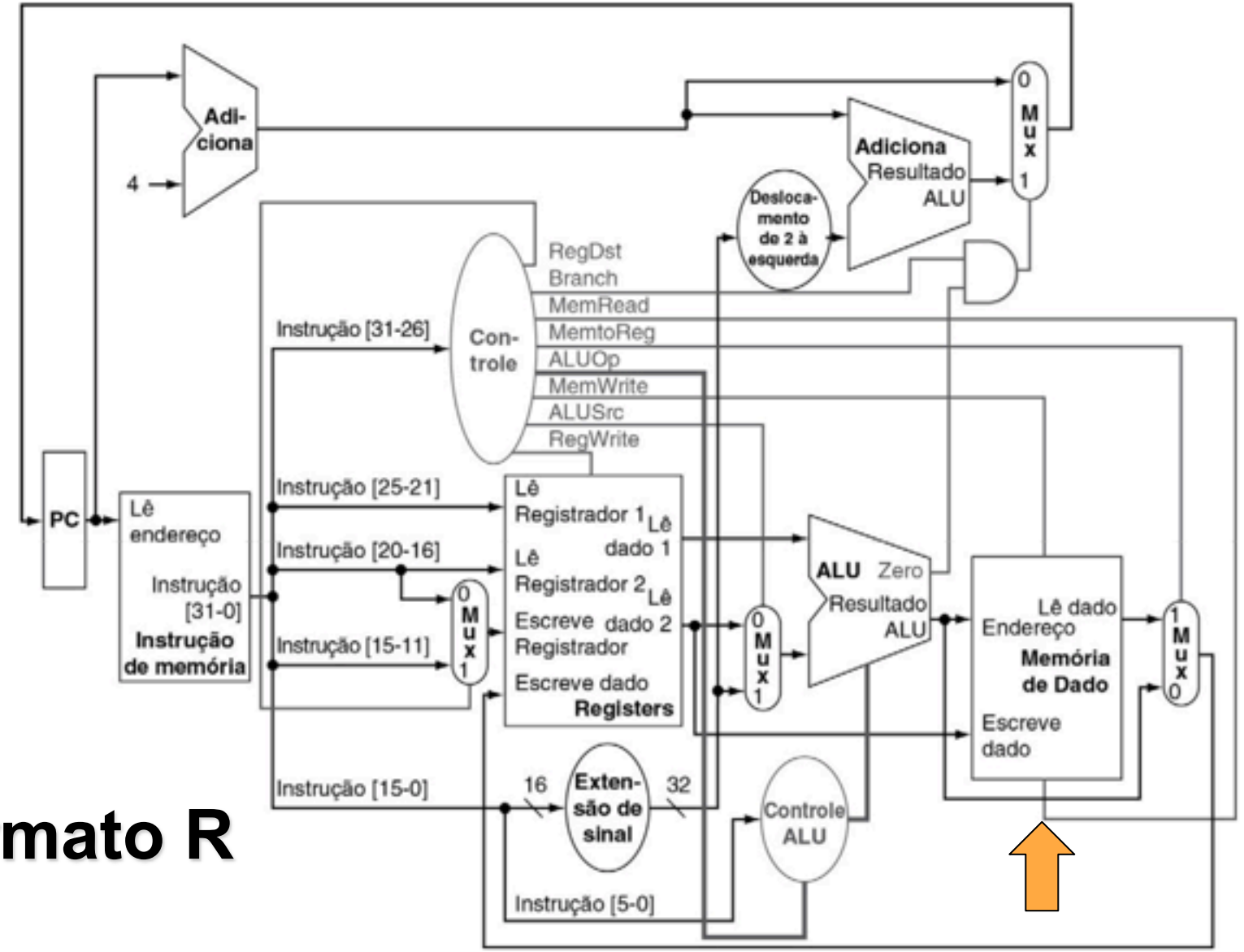
formato R

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
formato R	1	0	0				



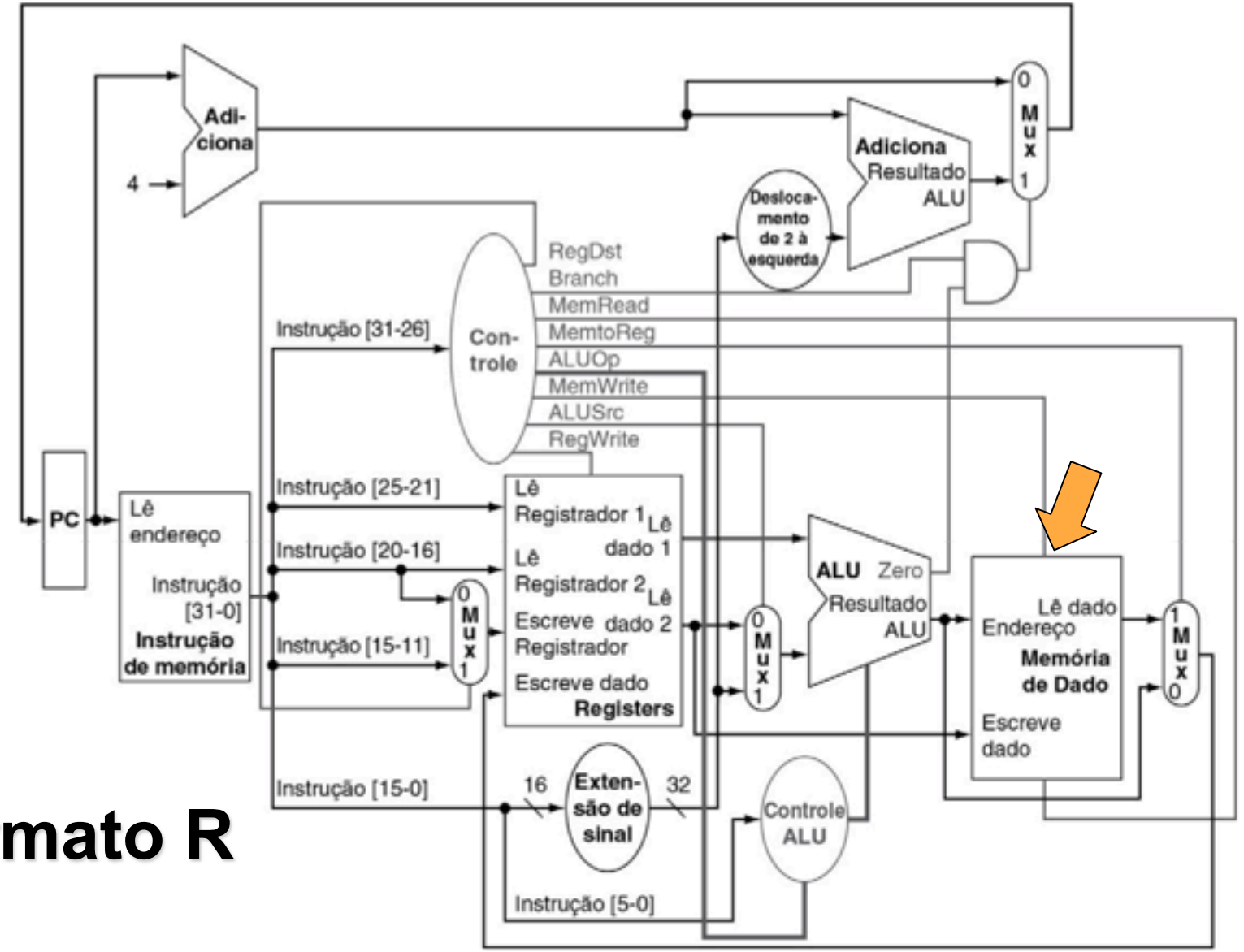
formato R

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
formato R	1	0	0	1			



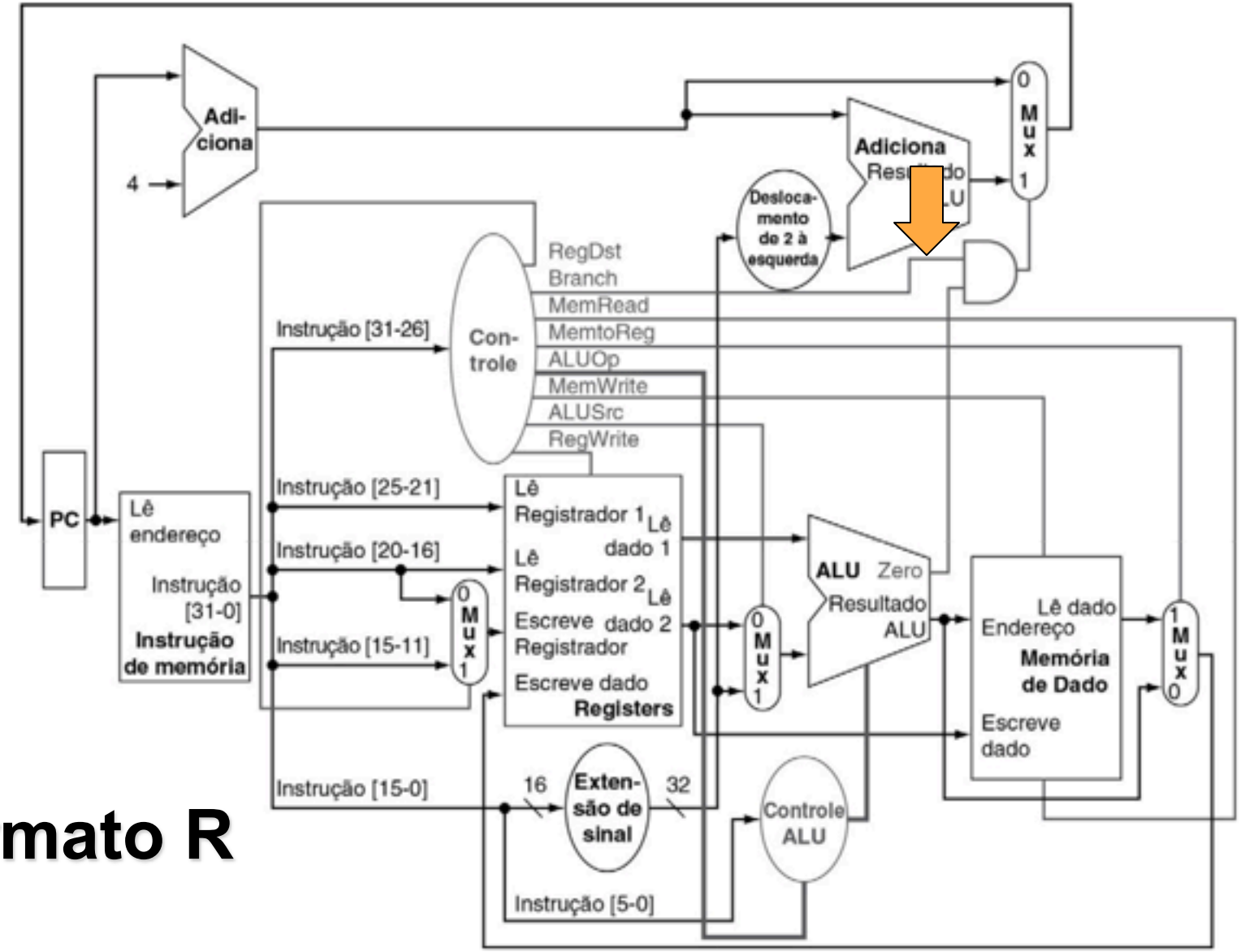
formato R

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
formato R	1	0	0	1	0		



formato R

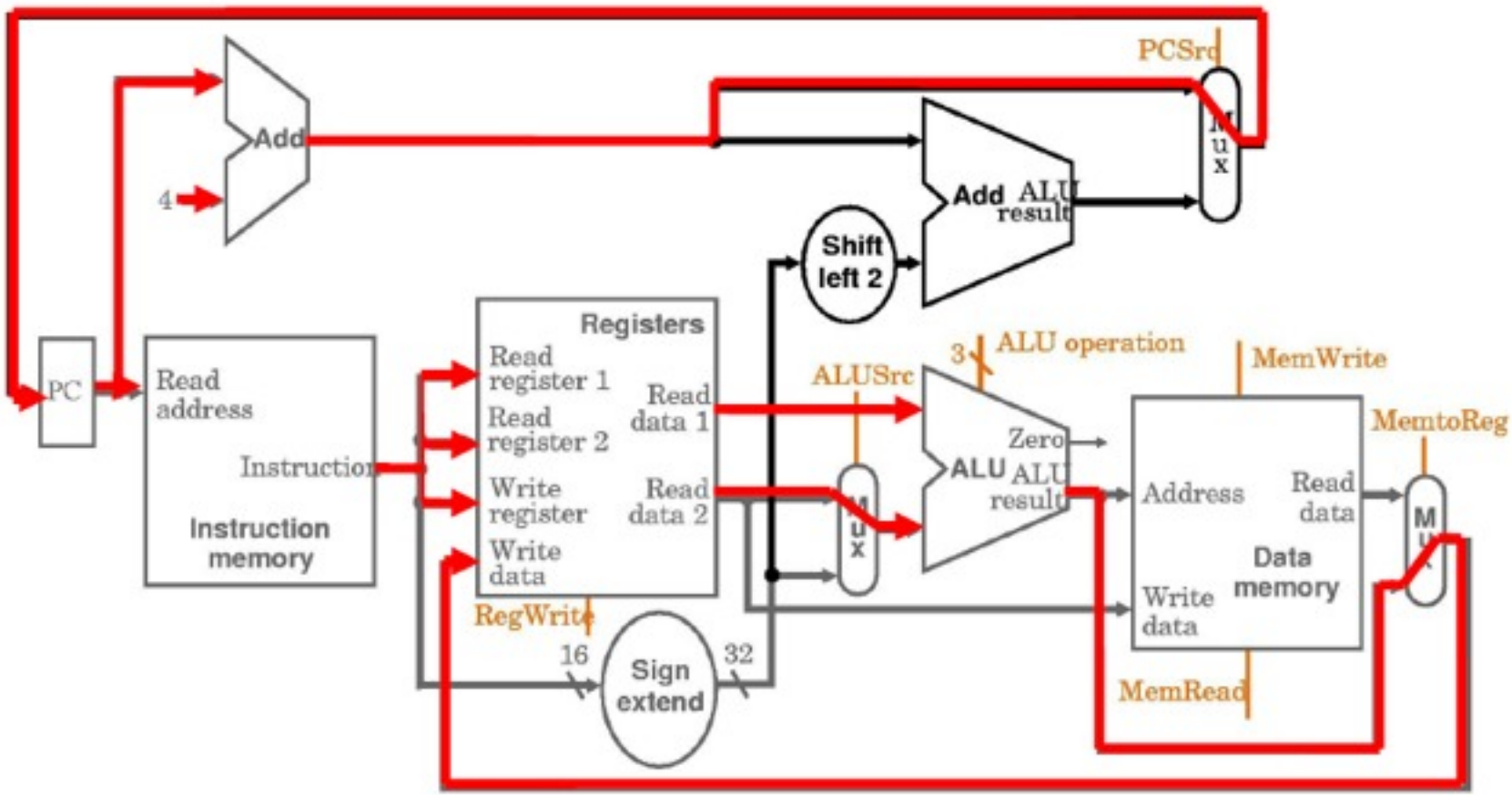
Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
formato R	1	0	0	1	0	0	



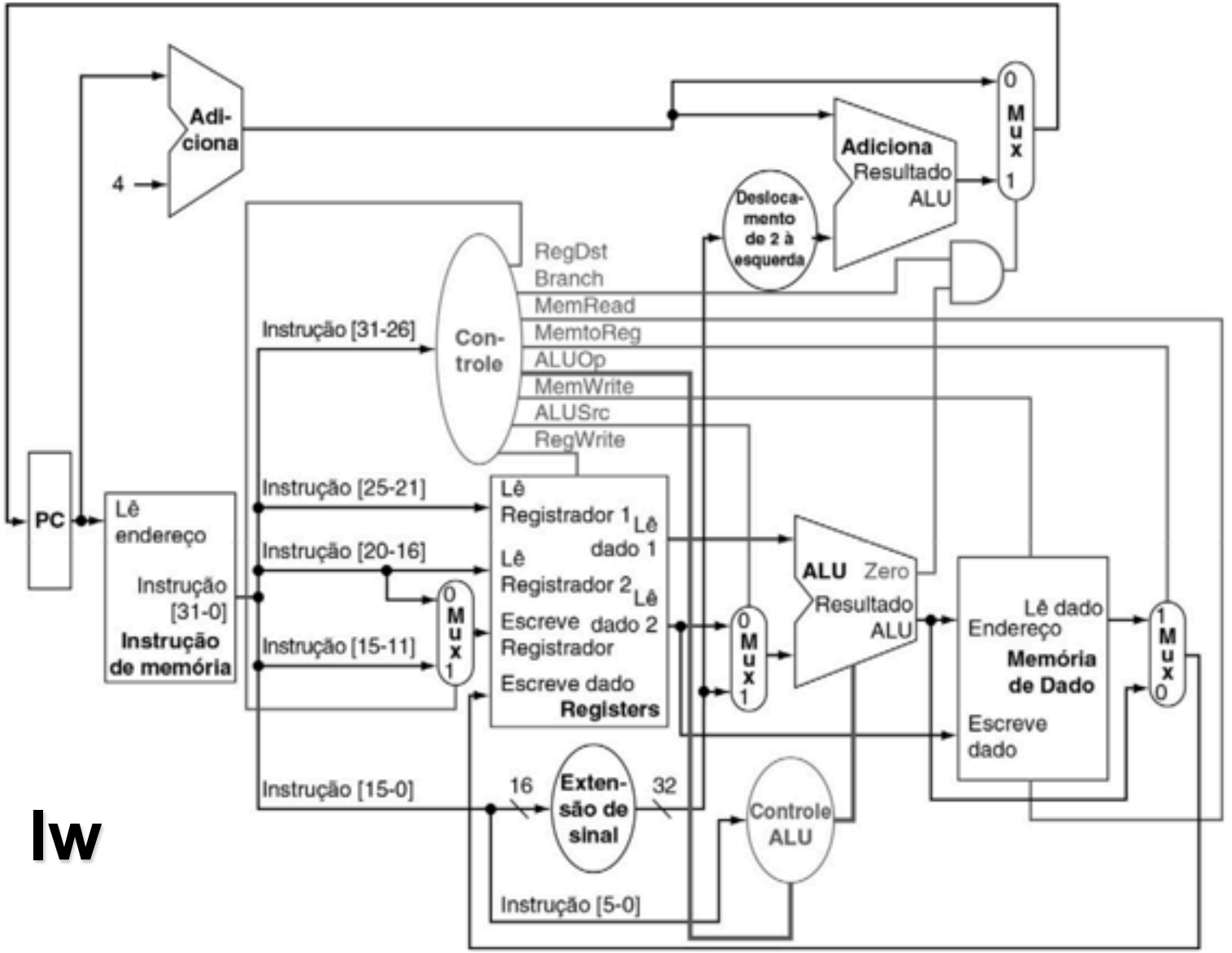
formato R

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
formato R	1	0	0	1	0	0	0

Caminho de dados para instruções tipo-R

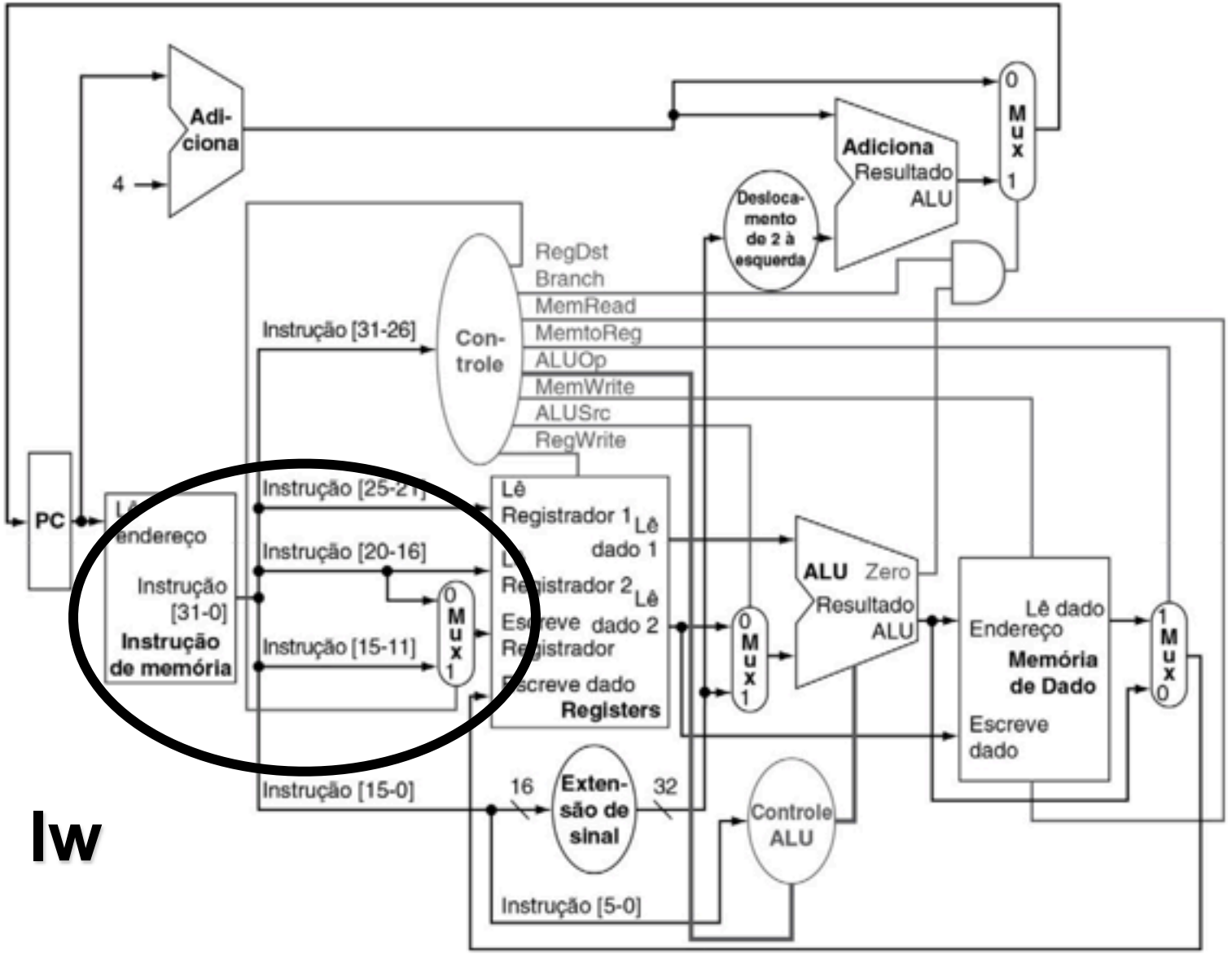


Example: **add $4, $18, $30**



lw

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
lw							

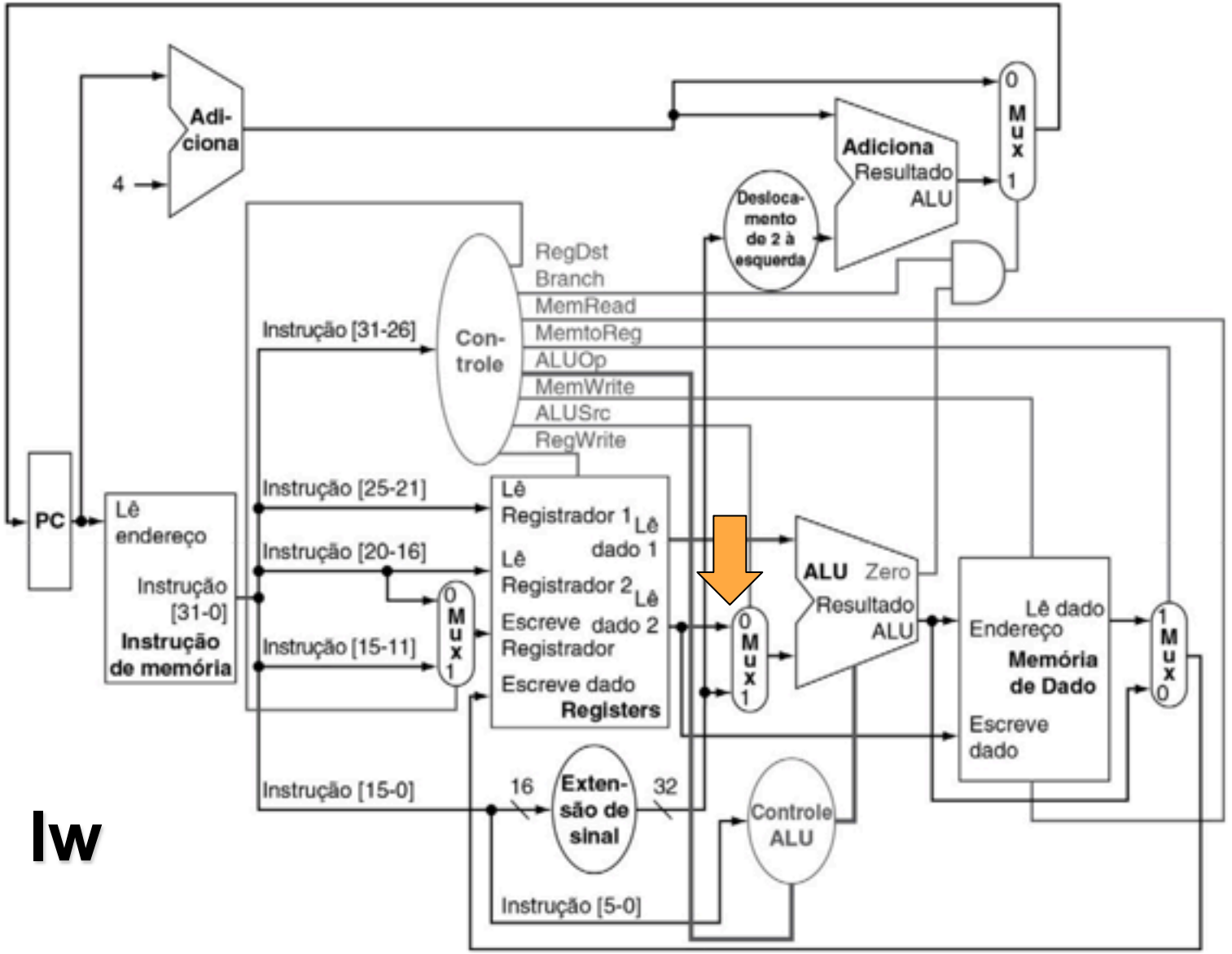


lw

Instrução	RegDs
lw	0

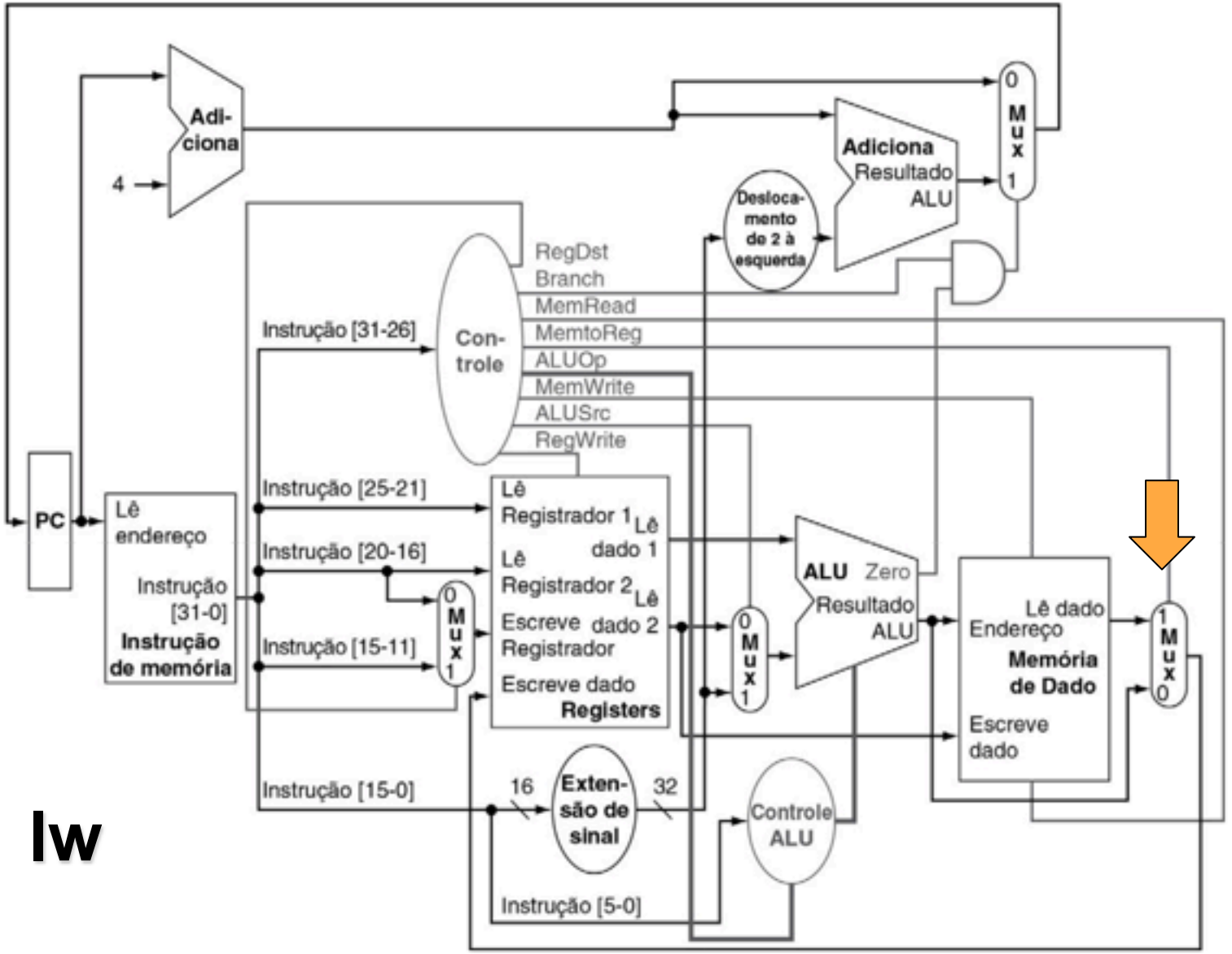
Campo	35 ou 43	rs	rt	address
Posição de bit	31:26	25:21	20:16	15:0

b. Carrega ou armazena instrução



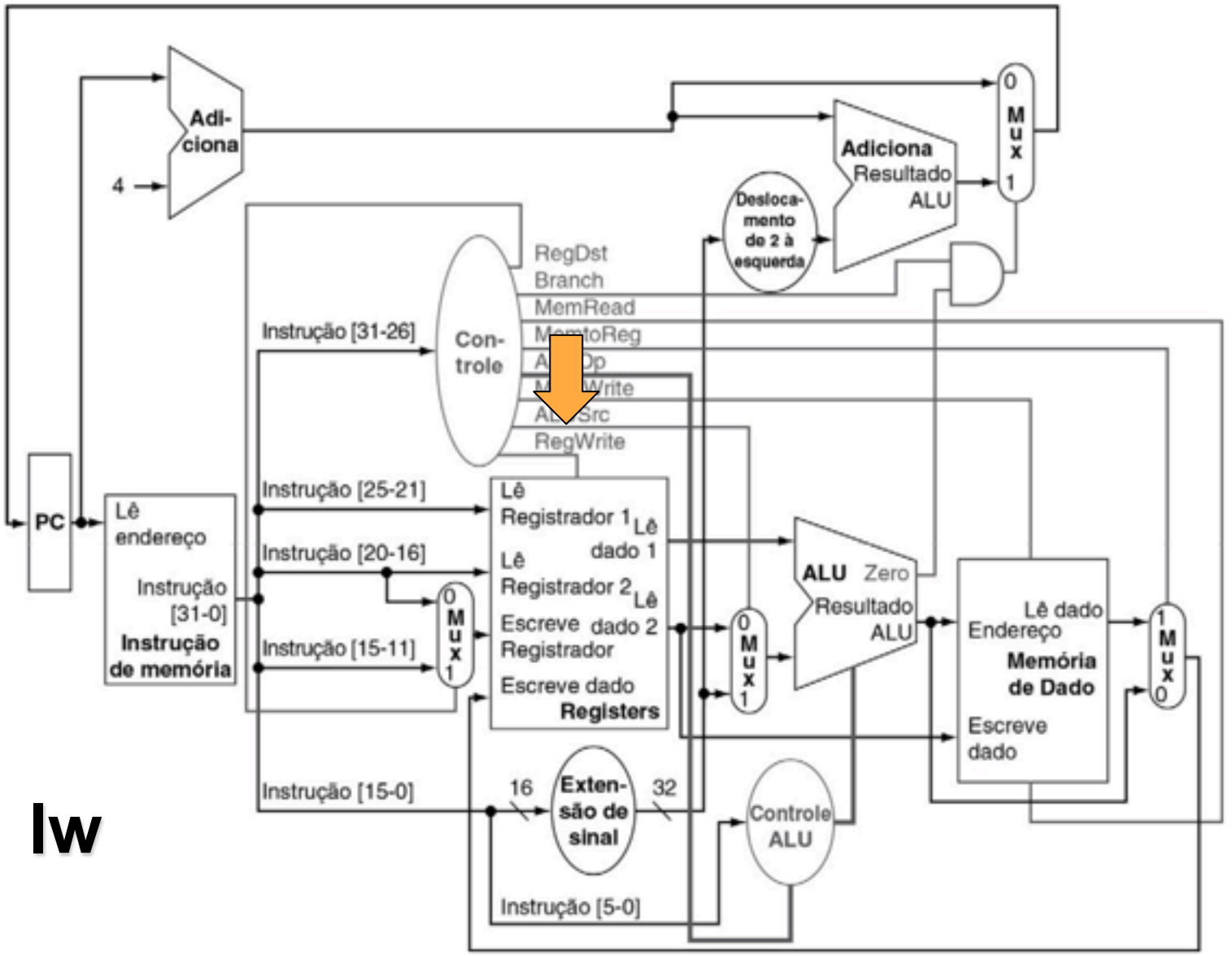
lw

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
lw	0	1					



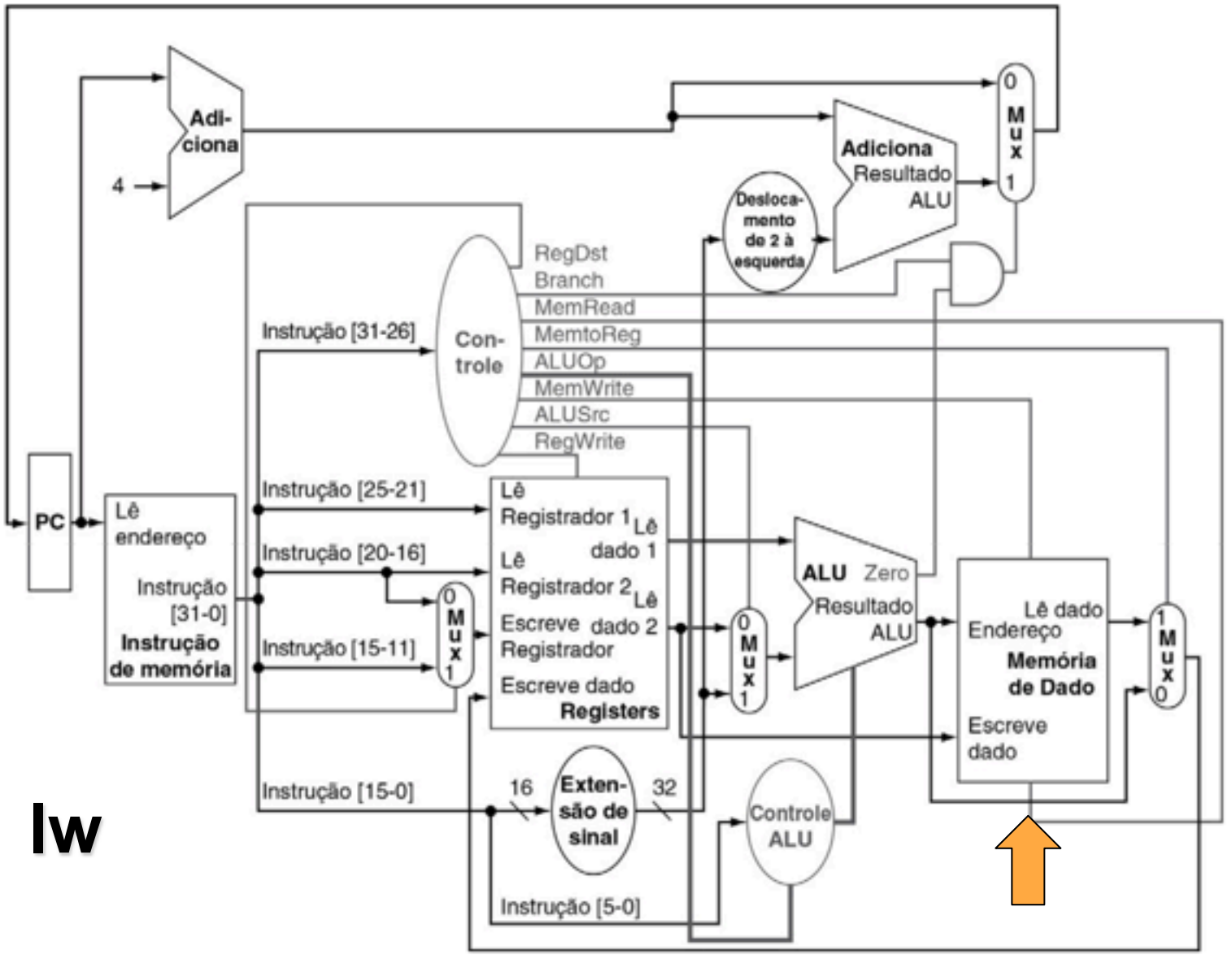
lw

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
lw	0	1	1				



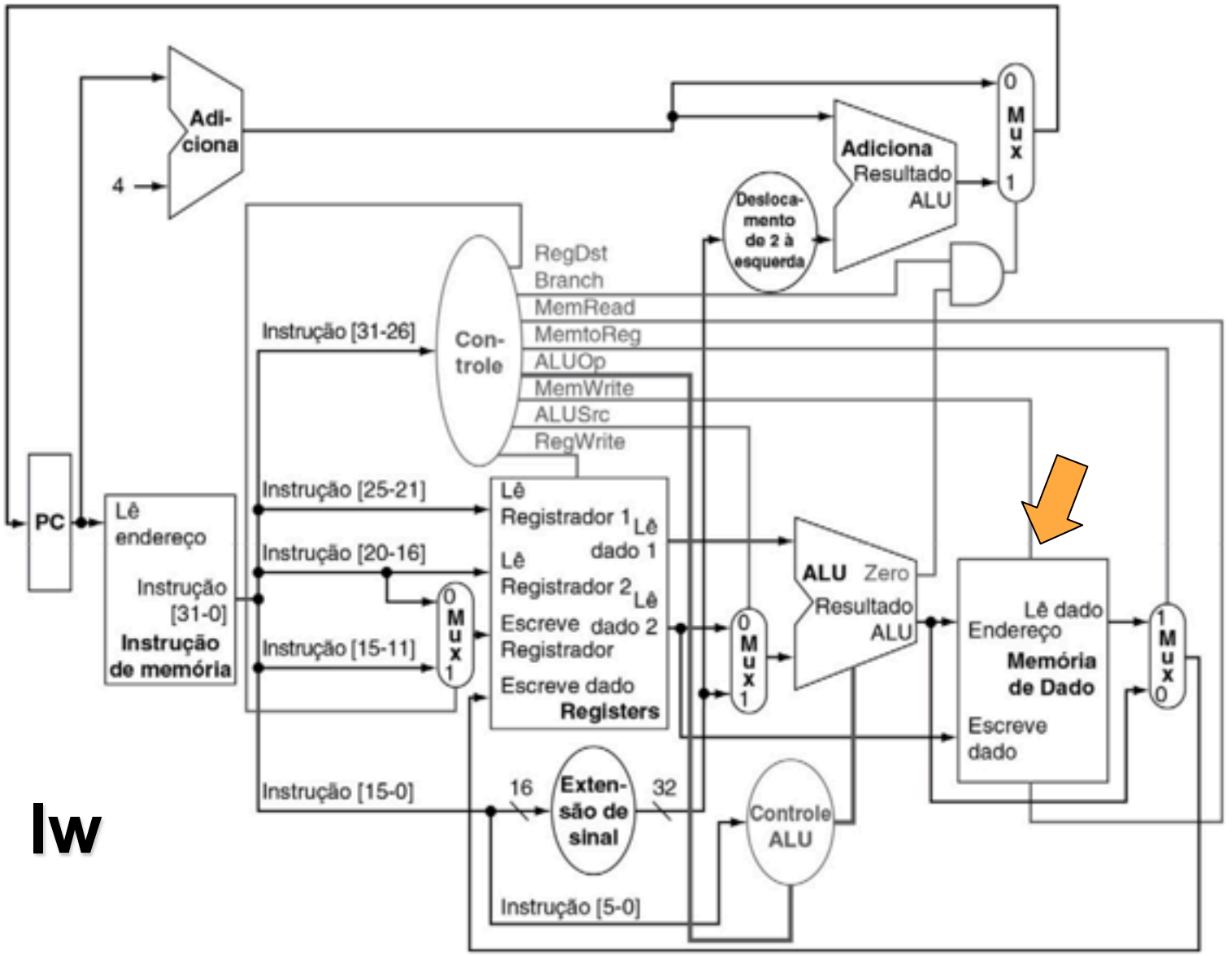
lw

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
lw	0	1	1	1			



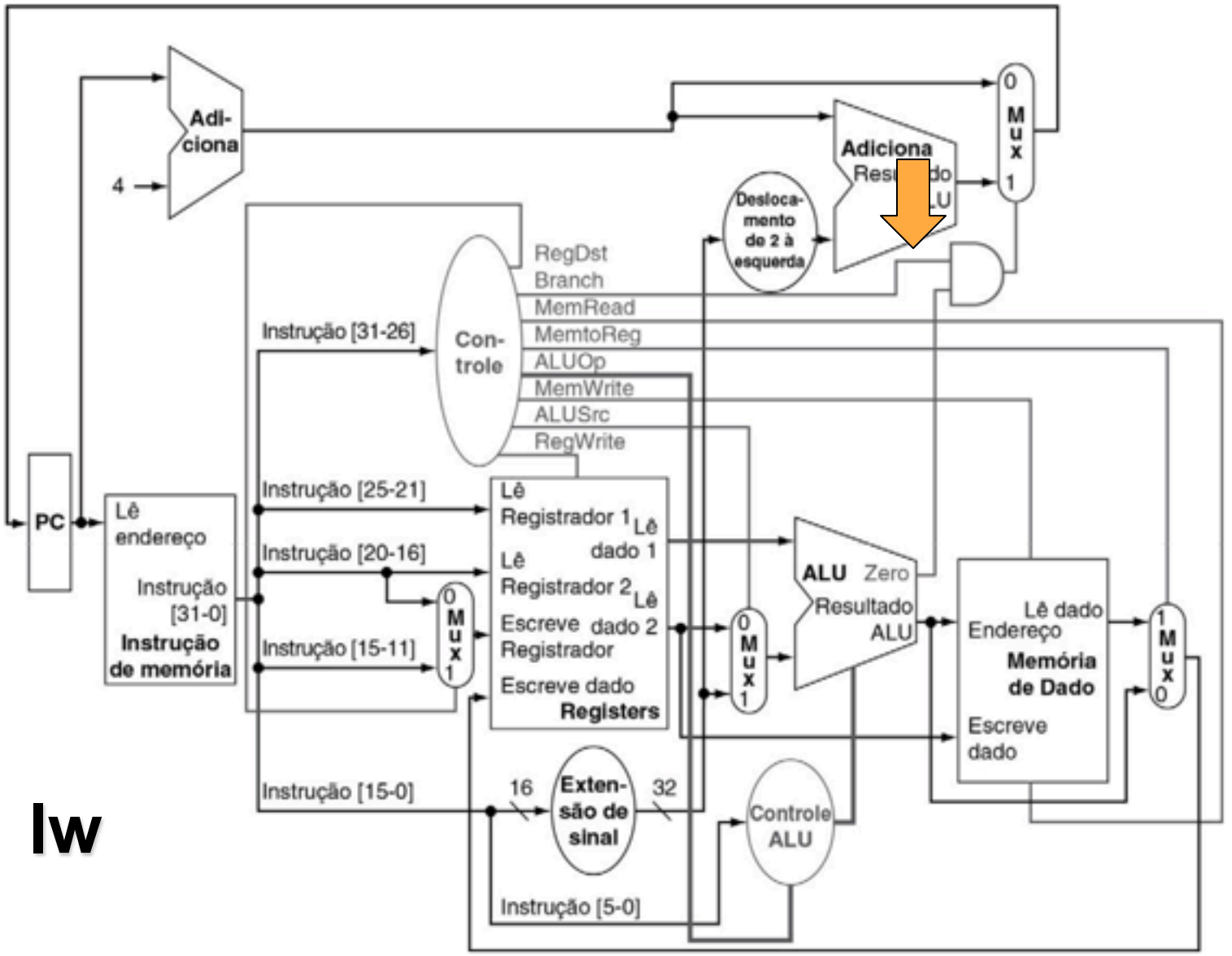
lw

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
lw	0	1	1	1	1		



lw

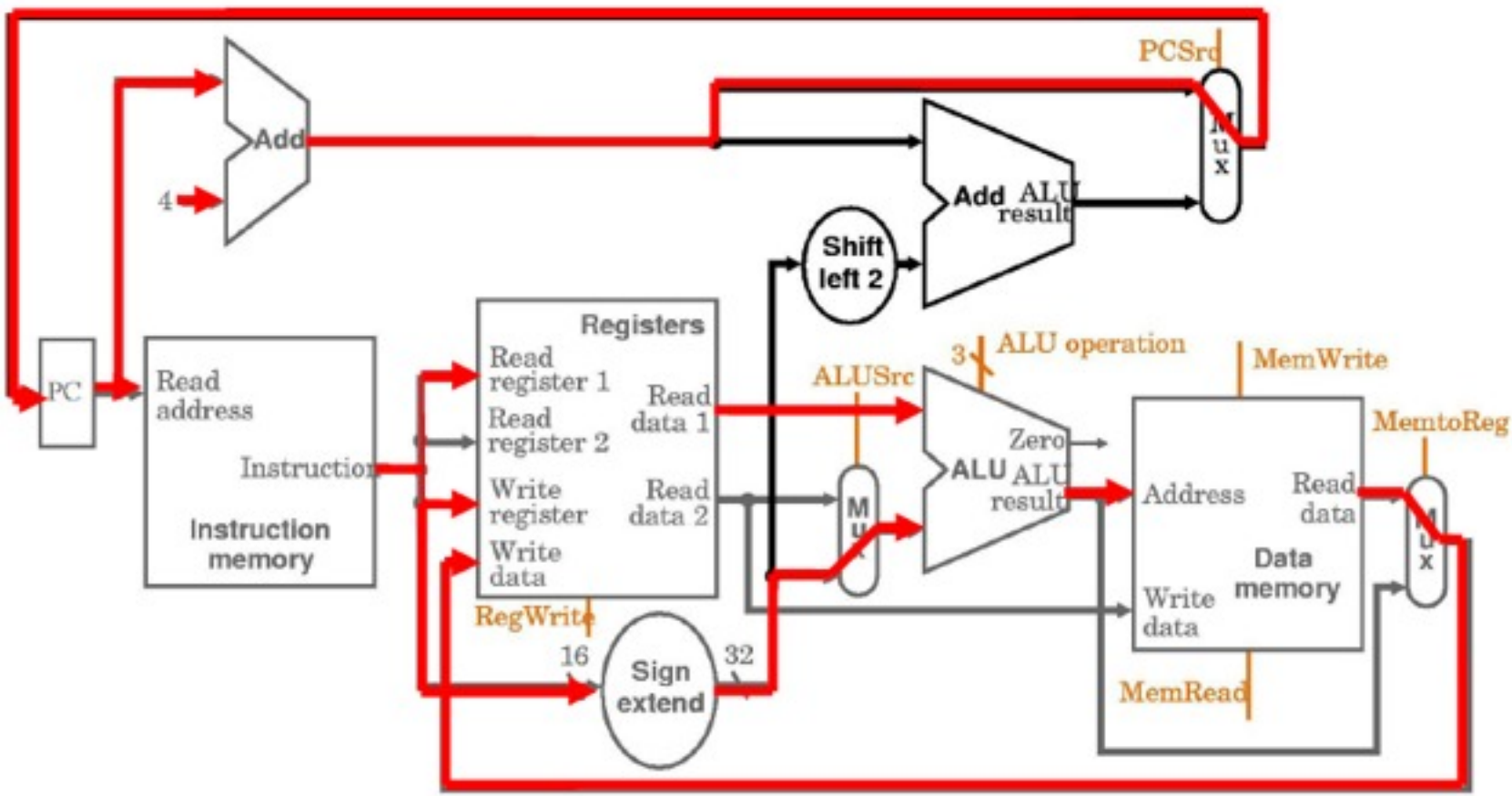
Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
lw	0	1	1	1	1	0	



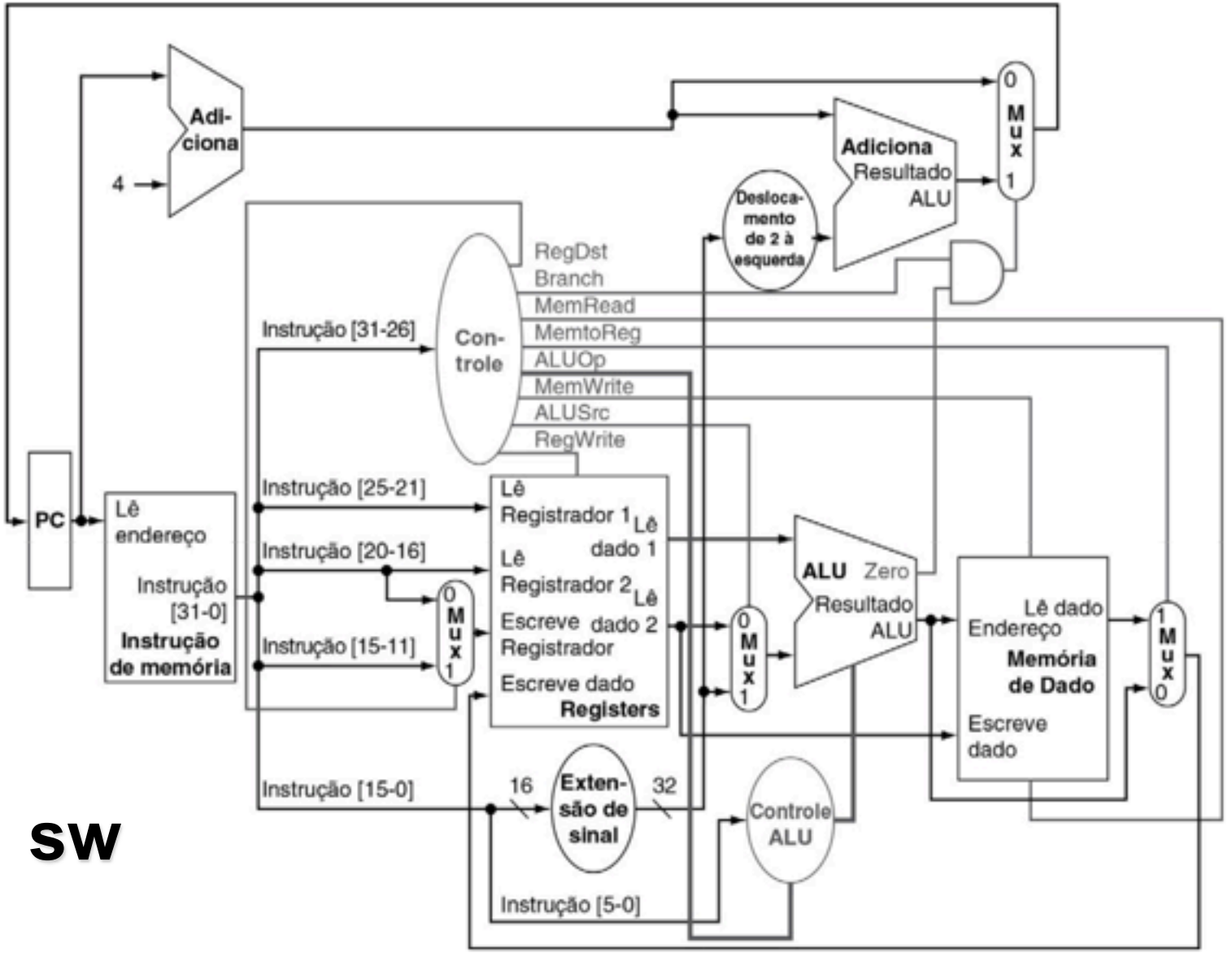
lw

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
lw	0	1	1	1	1	0	0

Caminho de dados para instrução load

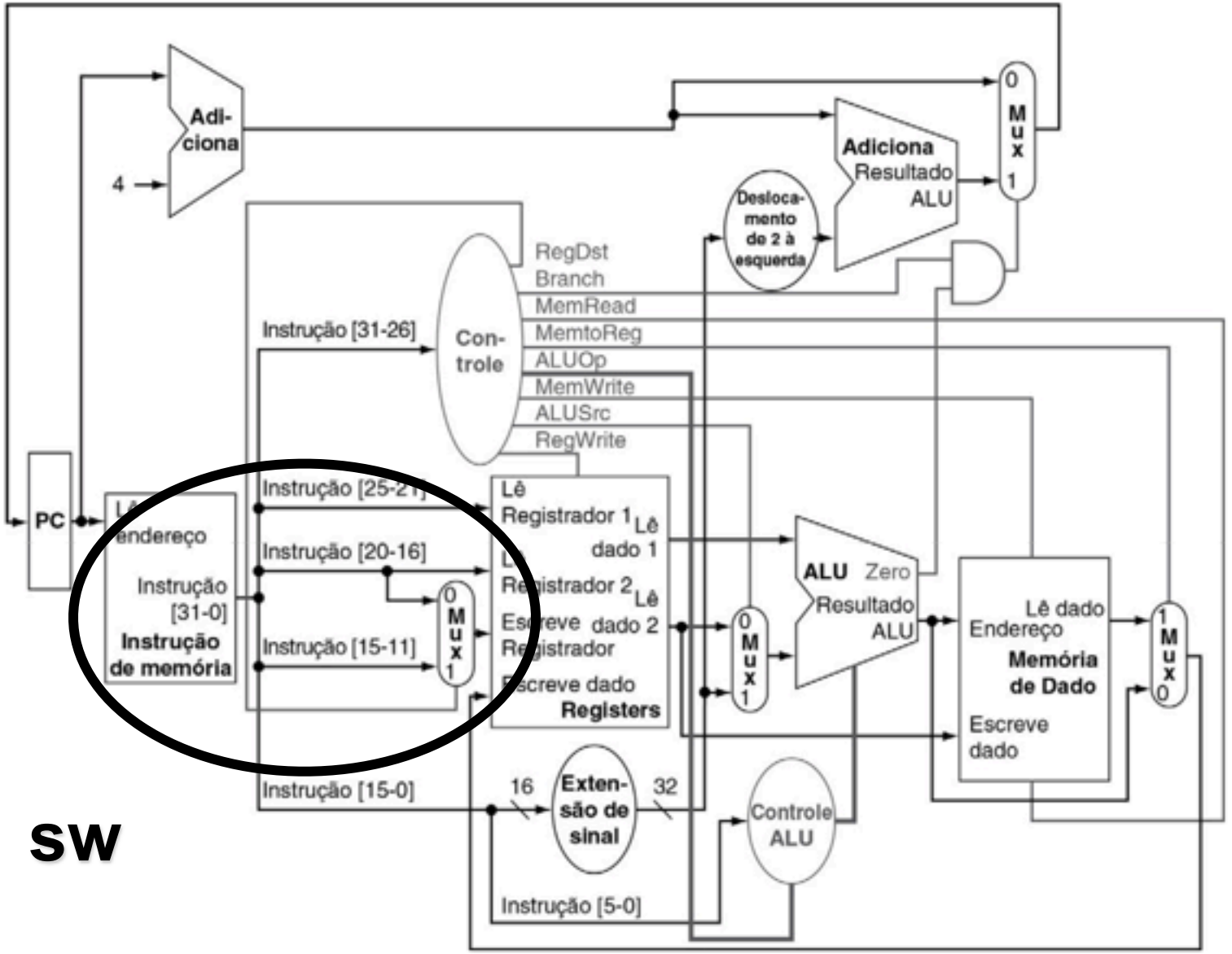


Example: lw S8, 112(S2)



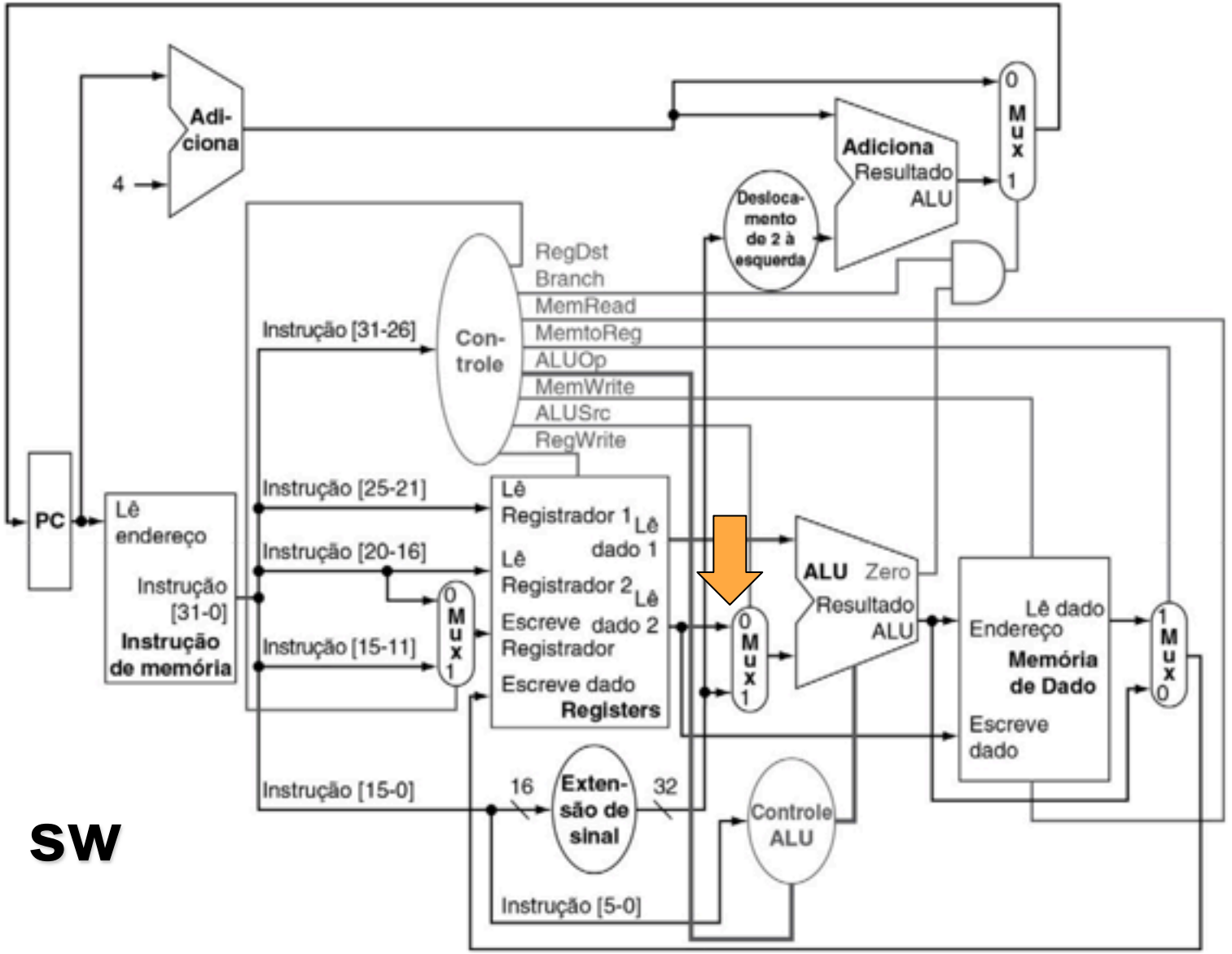
SW

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
SW							



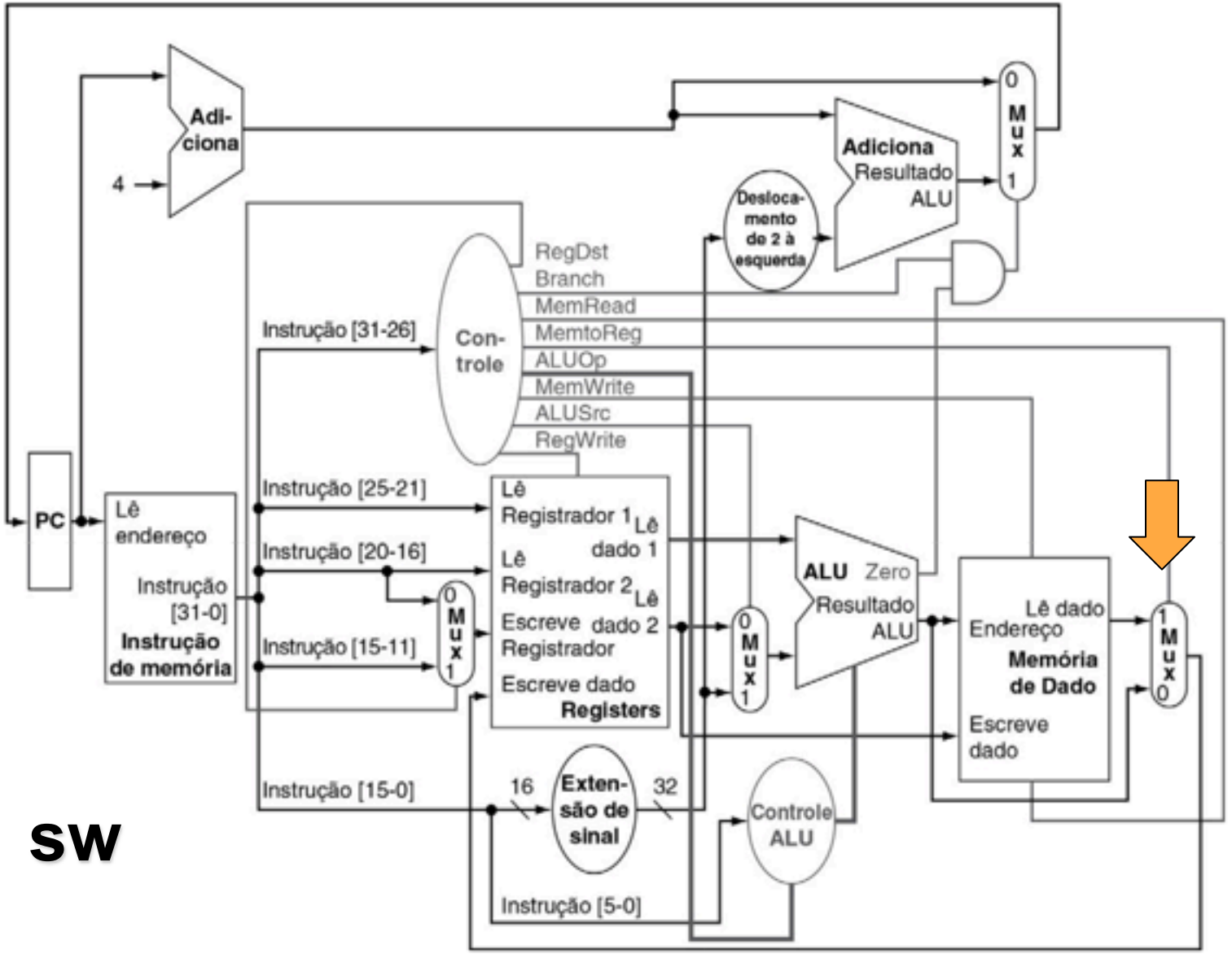
SW

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
SW	X						



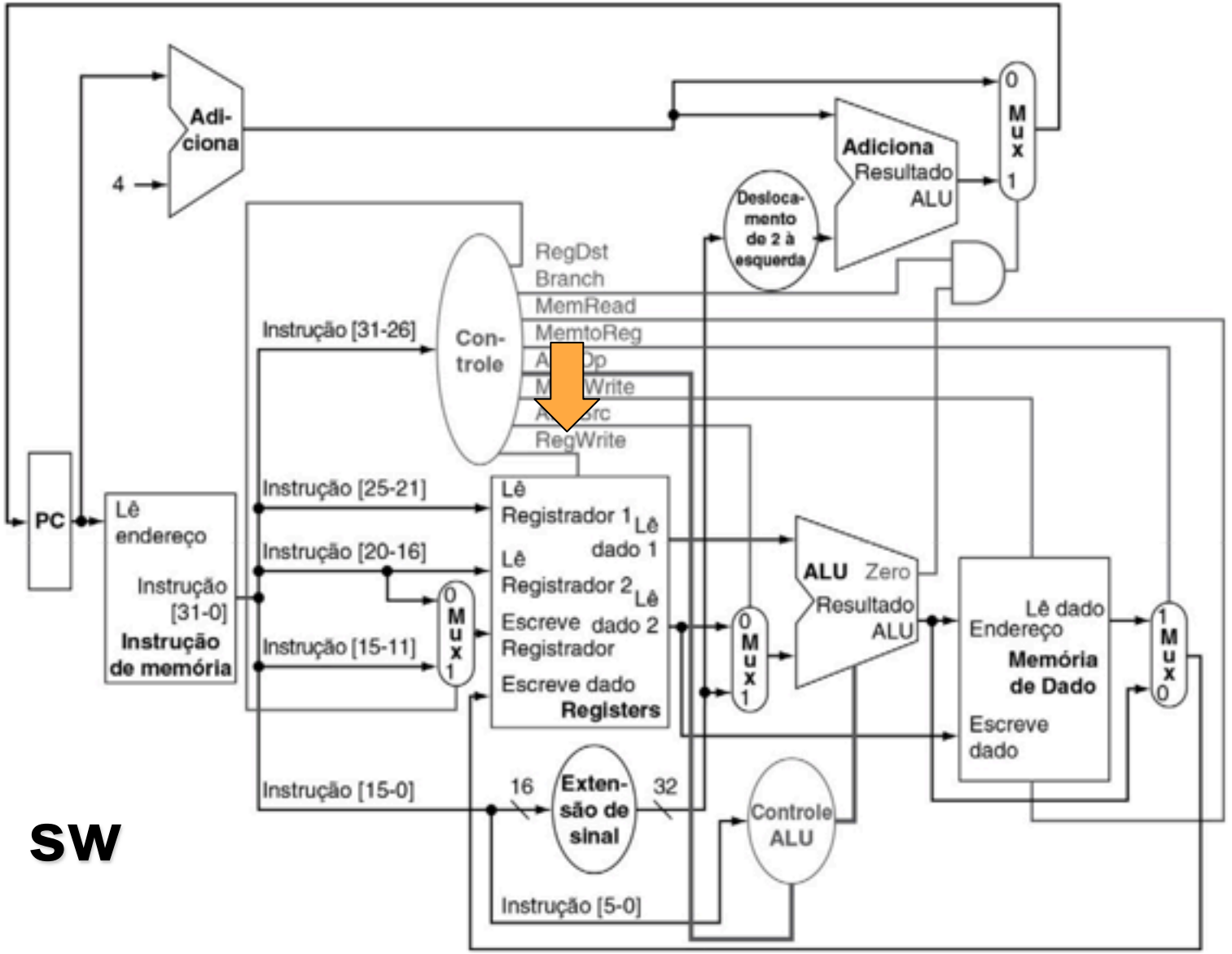
SW

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
sw	X	1					



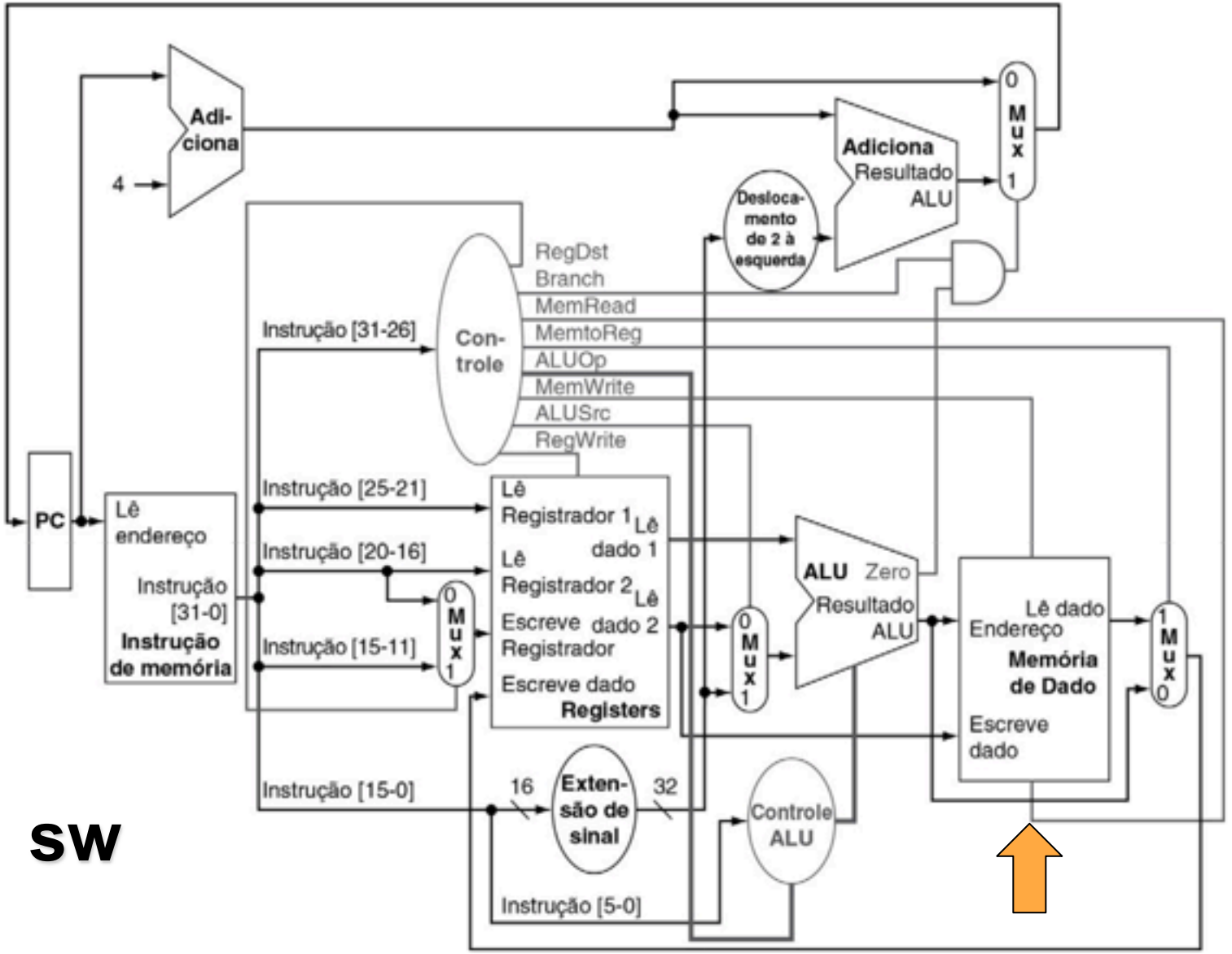
SW

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
sw	X	1	X				

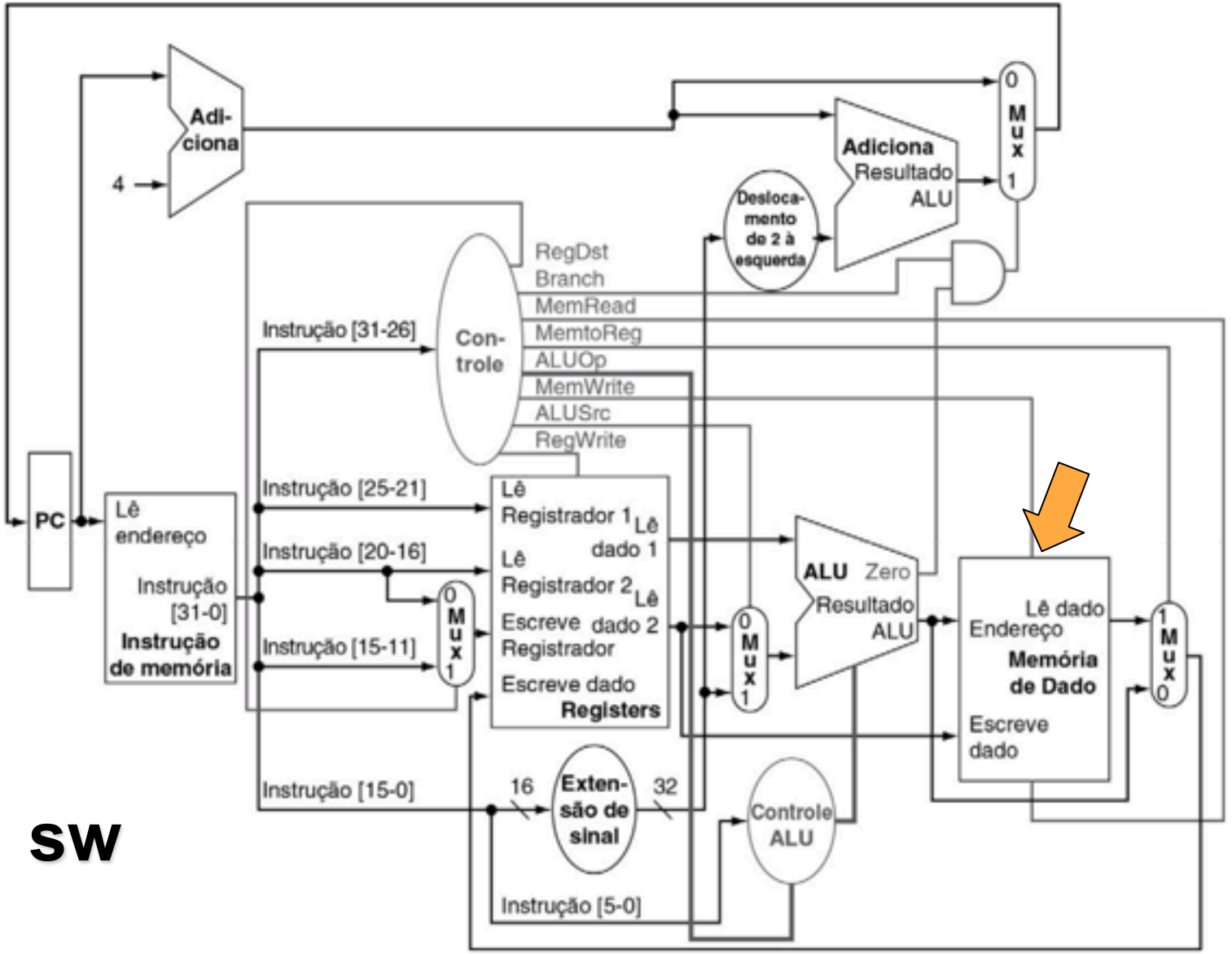


SW

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
sw	X	1	X	0			

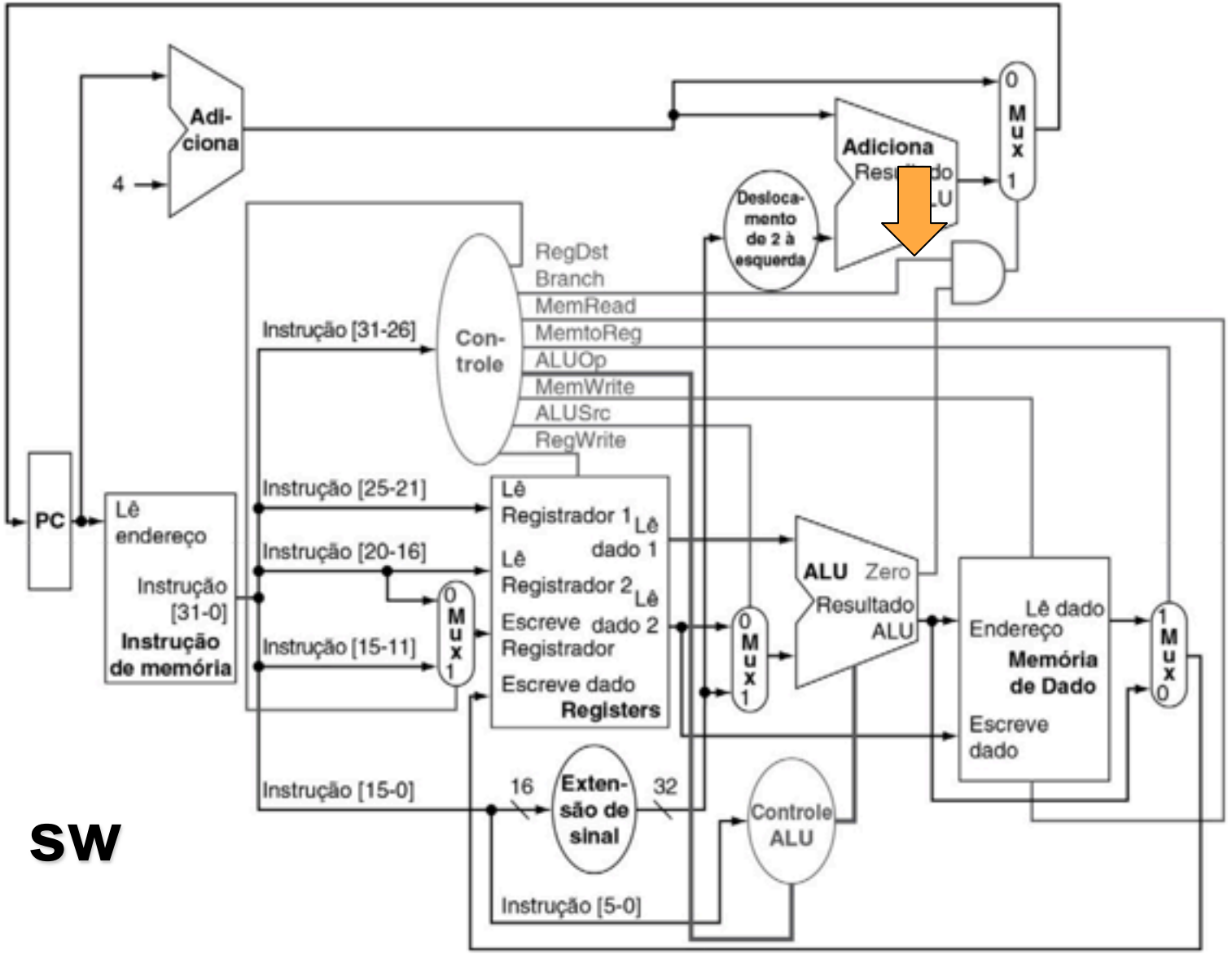


Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
sw	X	1	X	0	0		



SW

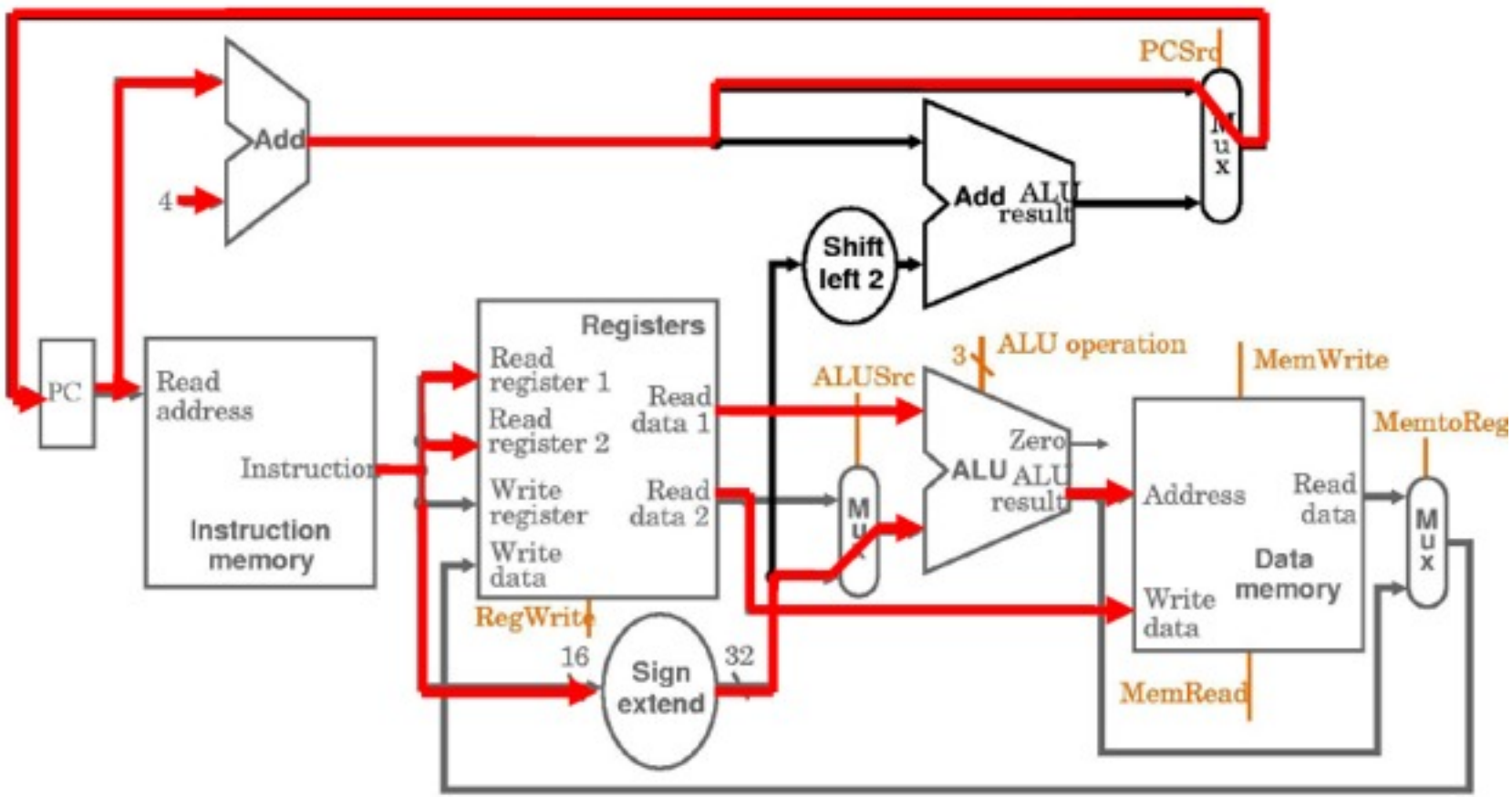
Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
sw	X	1	X	0	0	1	



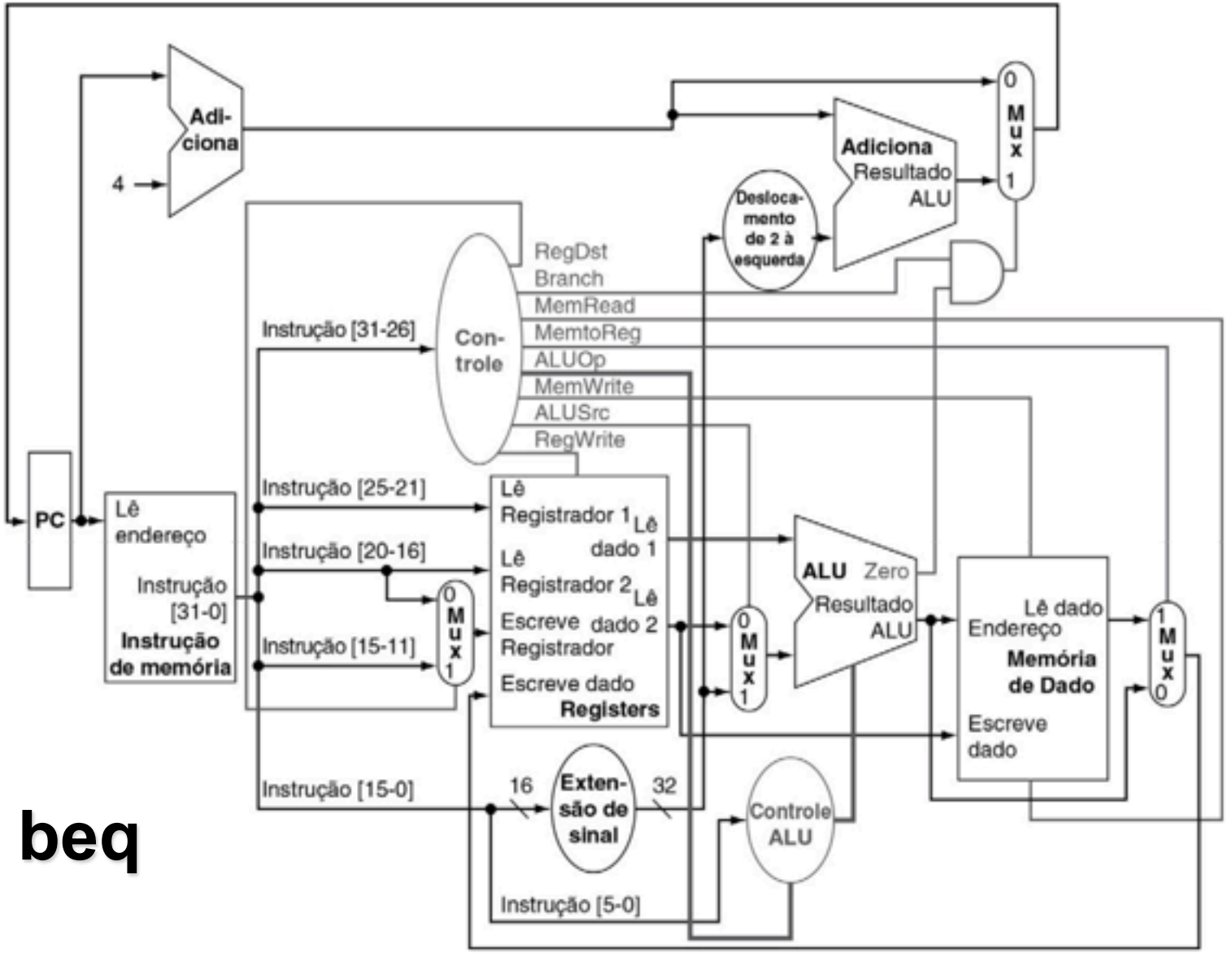
SW

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
sw	X	1	X	0	0	1	0

Caminho de dados para instrução store

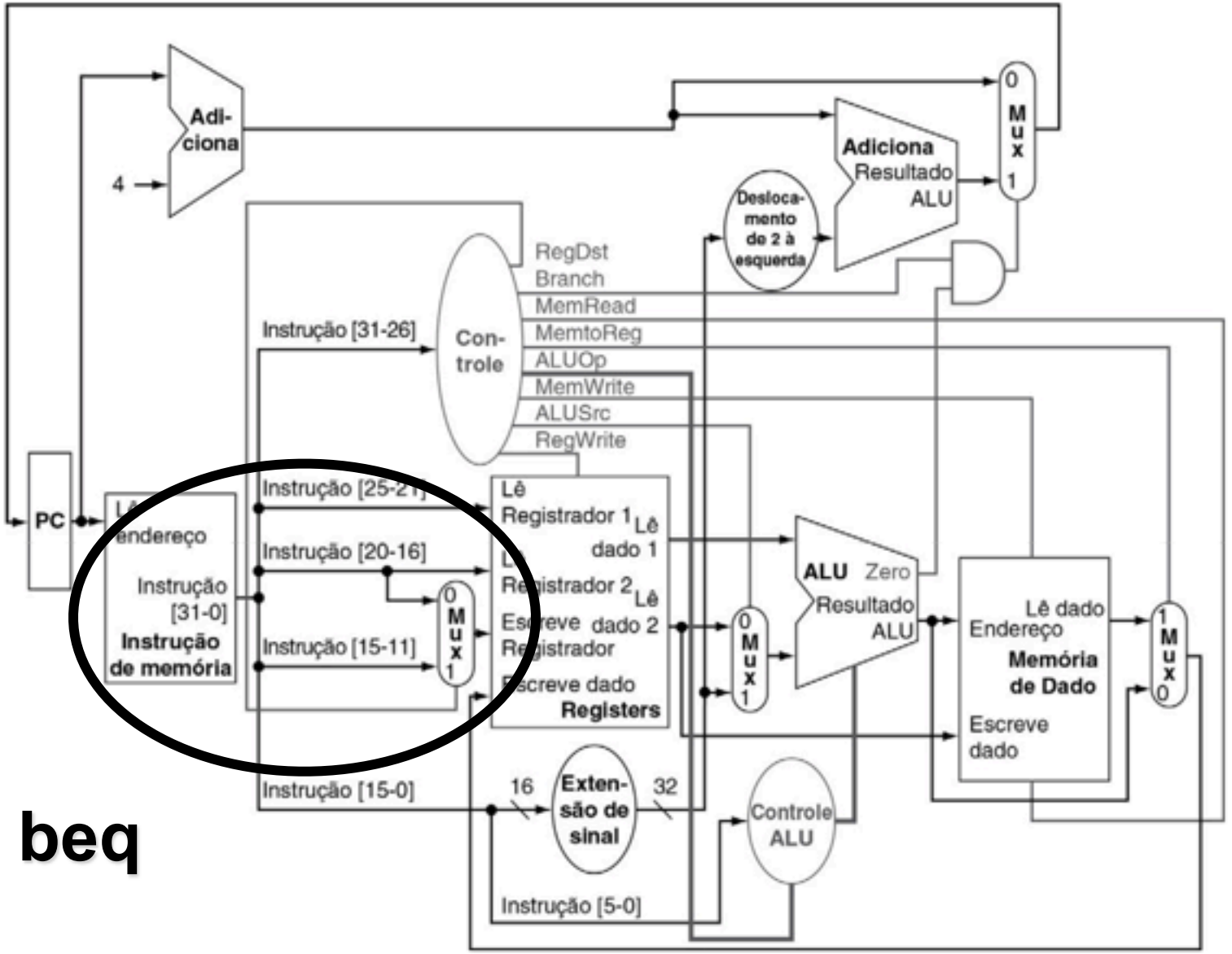


Example: **sw $10, 0($3)**



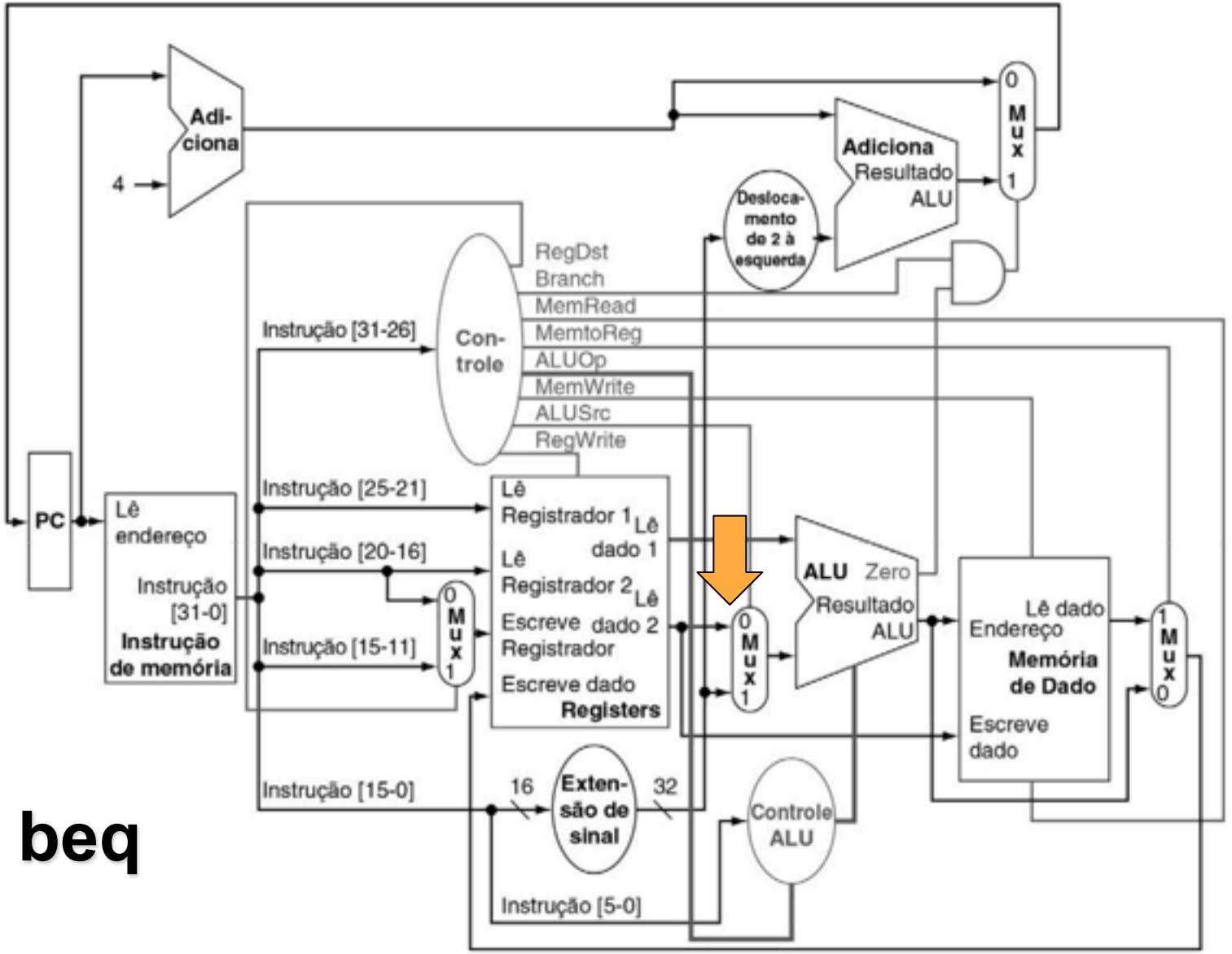
beq

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
beq							



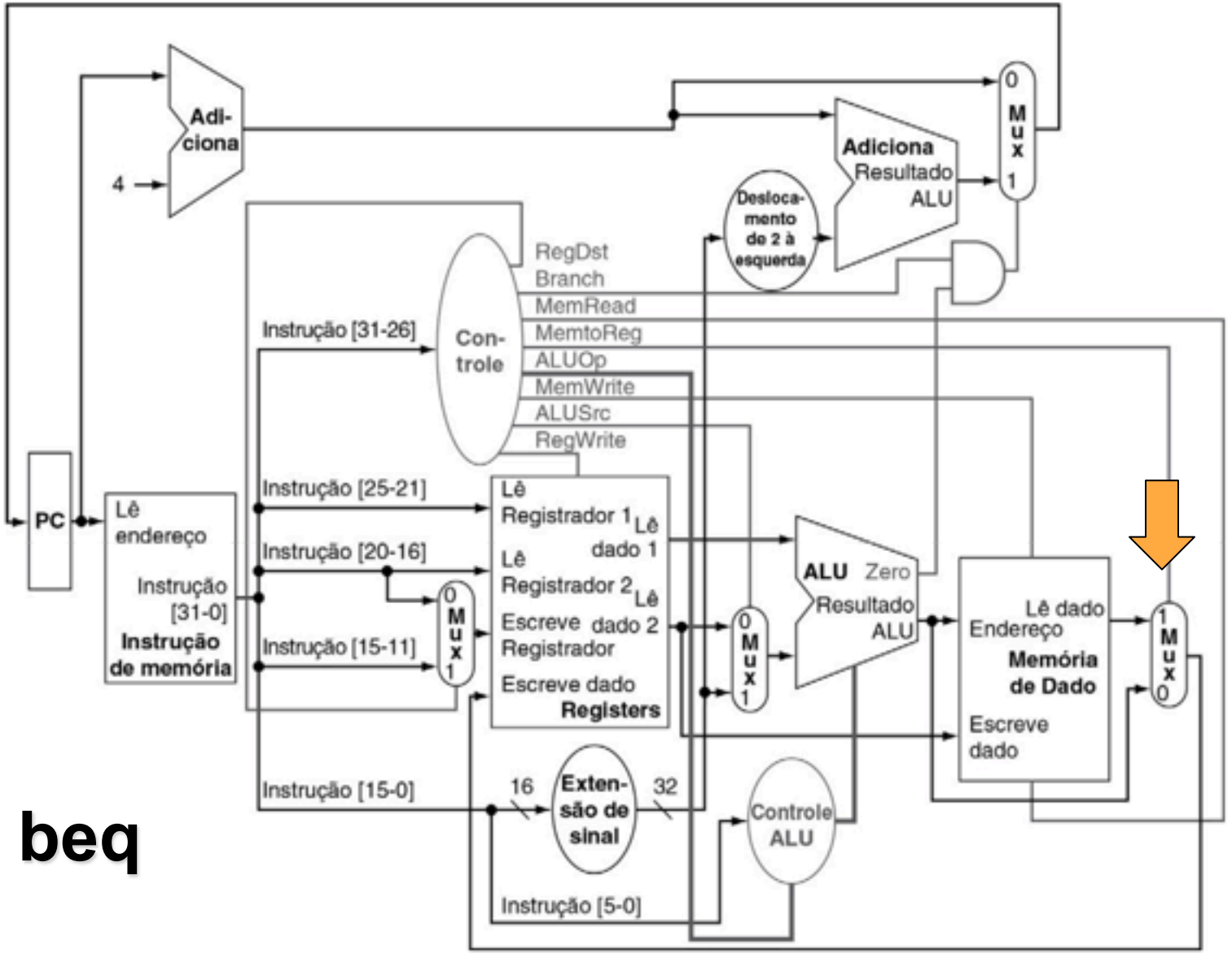
beq

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
beq	X						



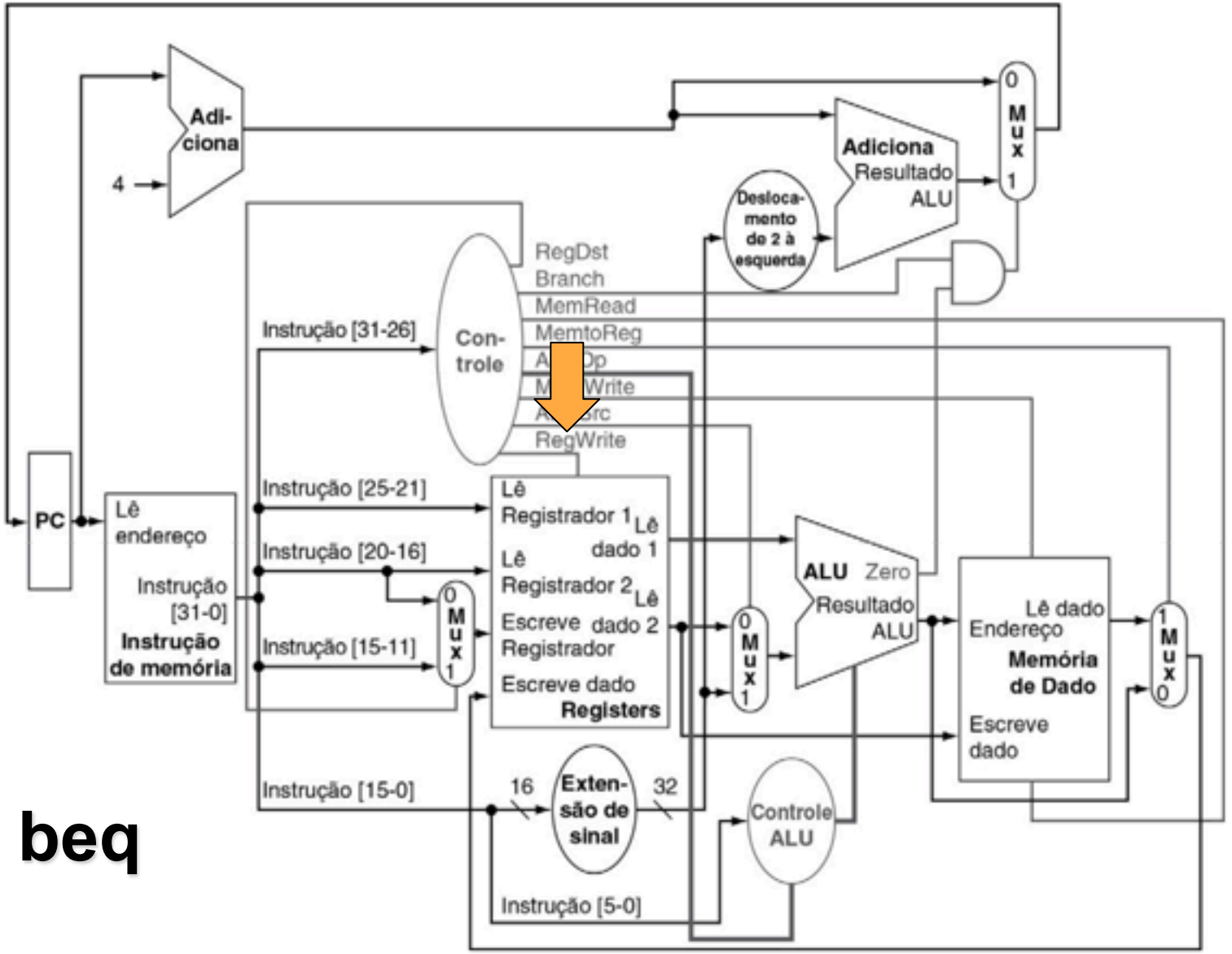
beq

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
beq	X	0					



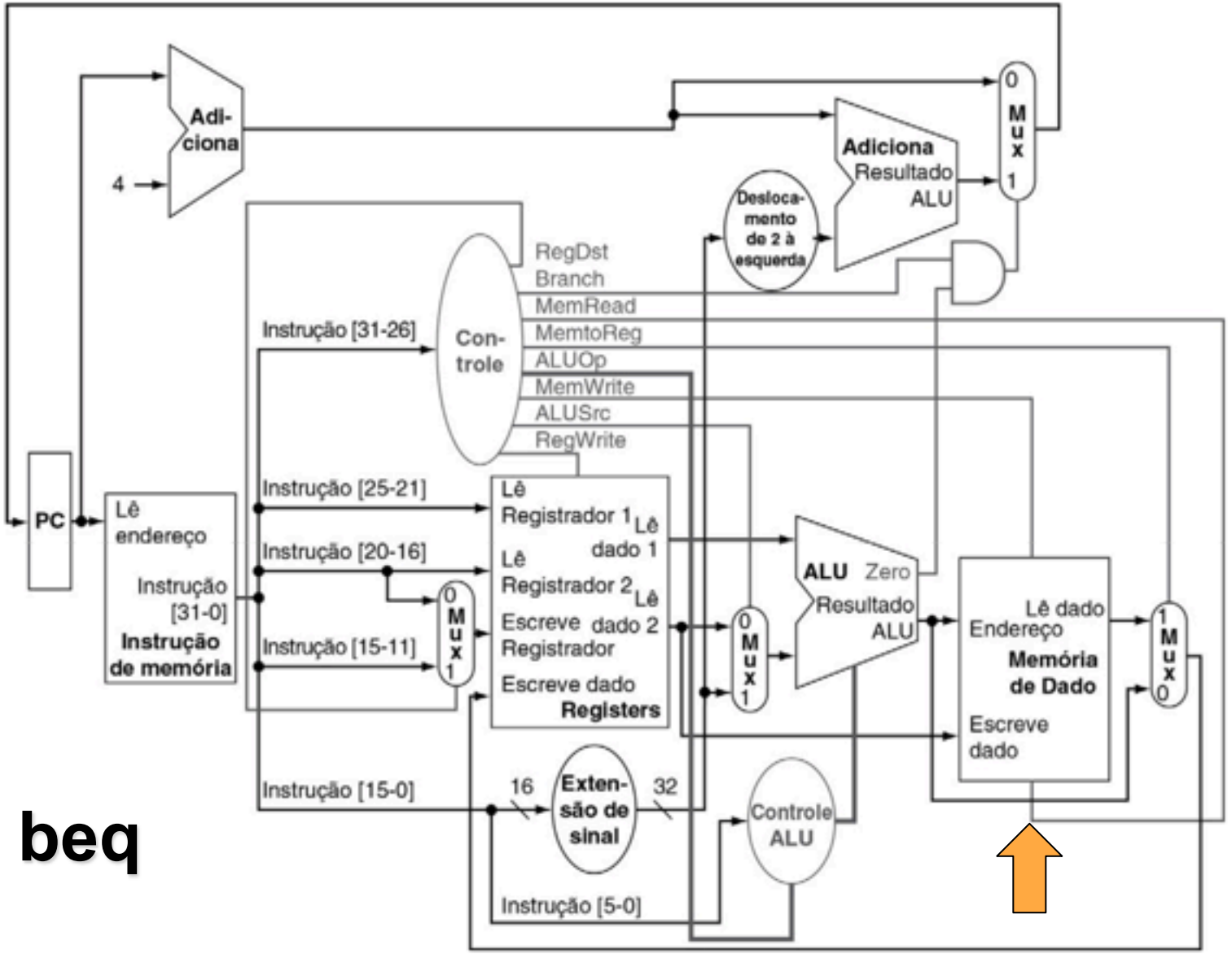
beq

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
beq	X	0	X				



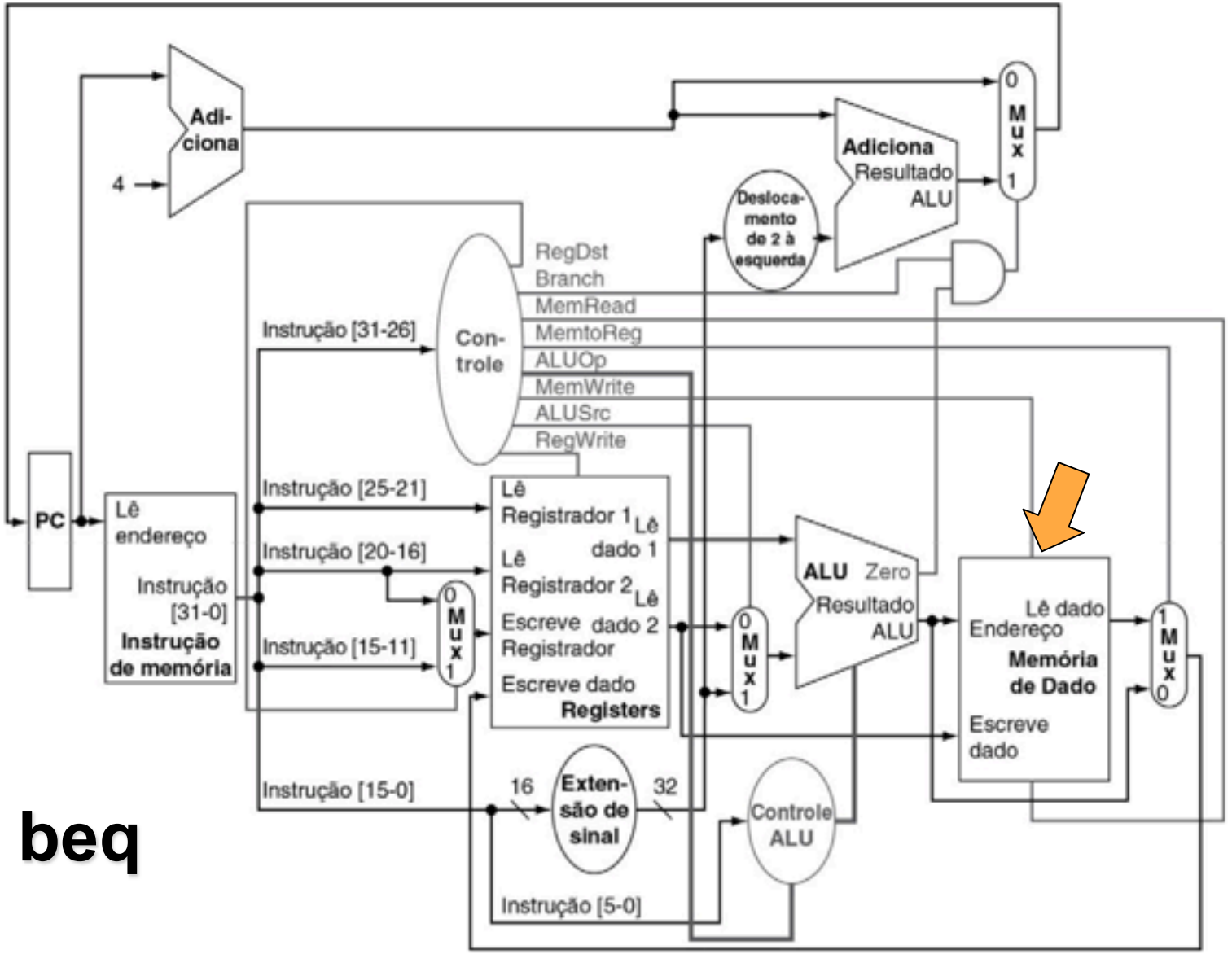
beq

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
beq	X	0	X	0			



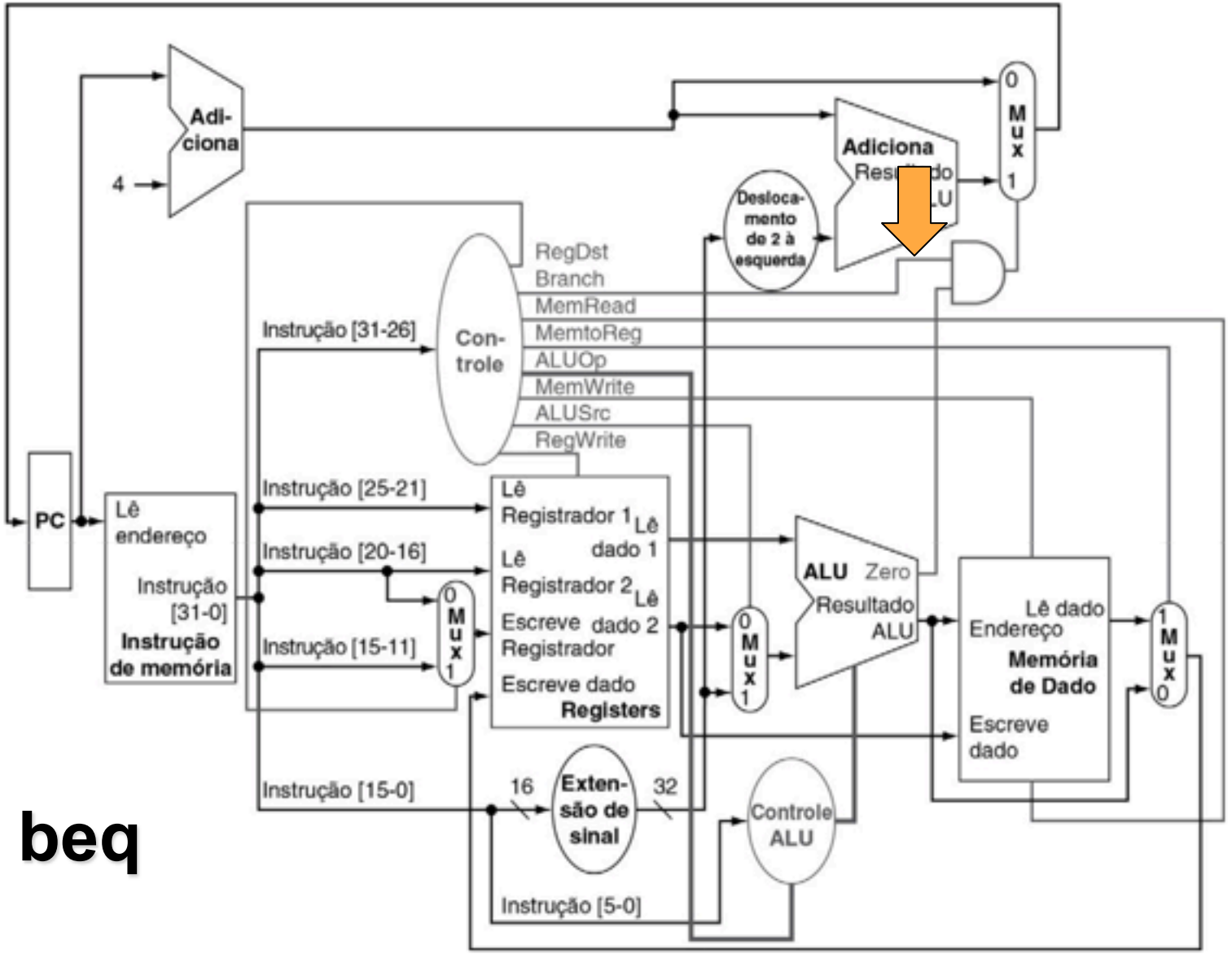
beq

Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
beq	X	0	X	0	0		



beq

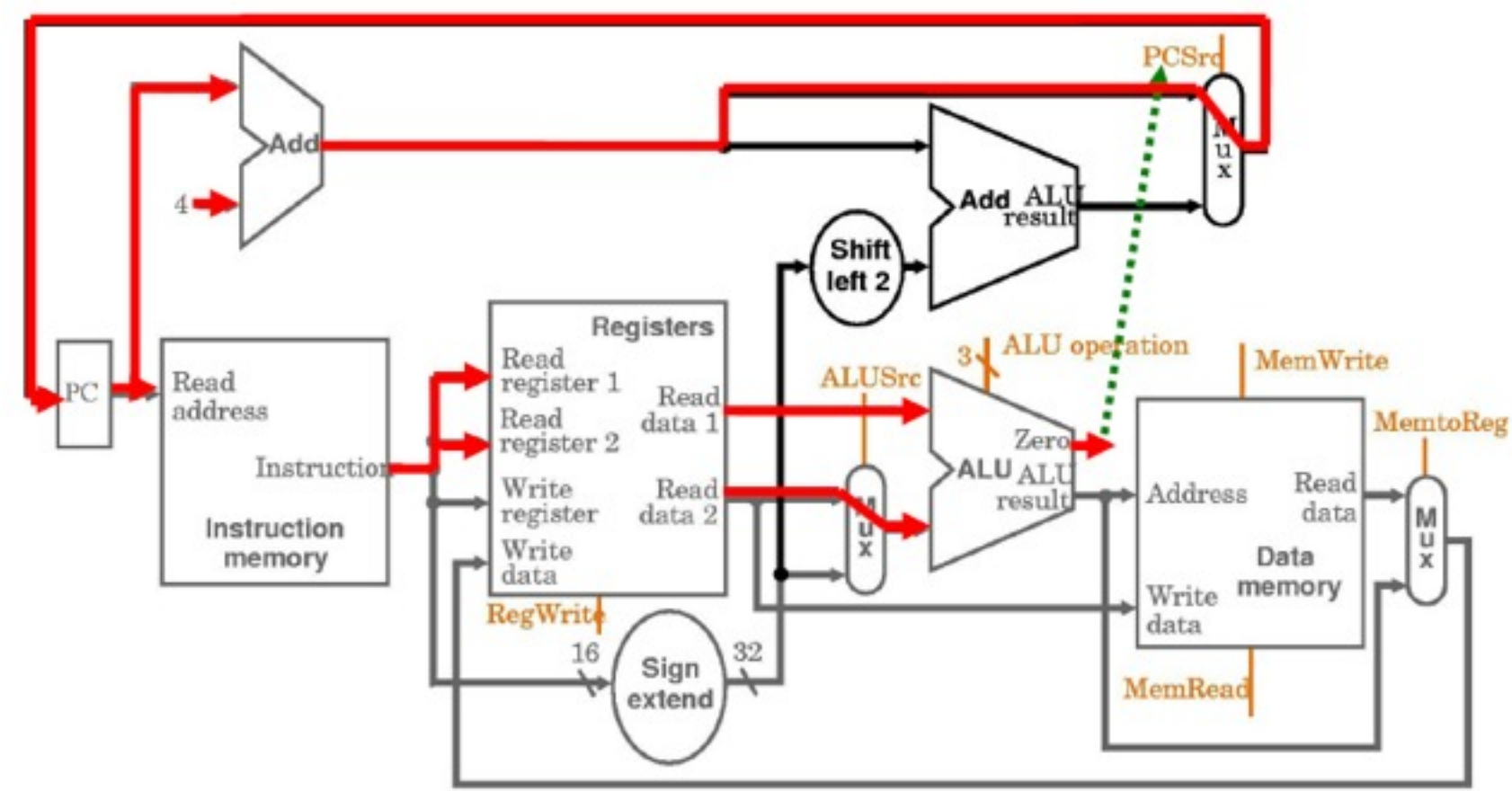
Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
beq	X	0	X	0	0	0	



beq

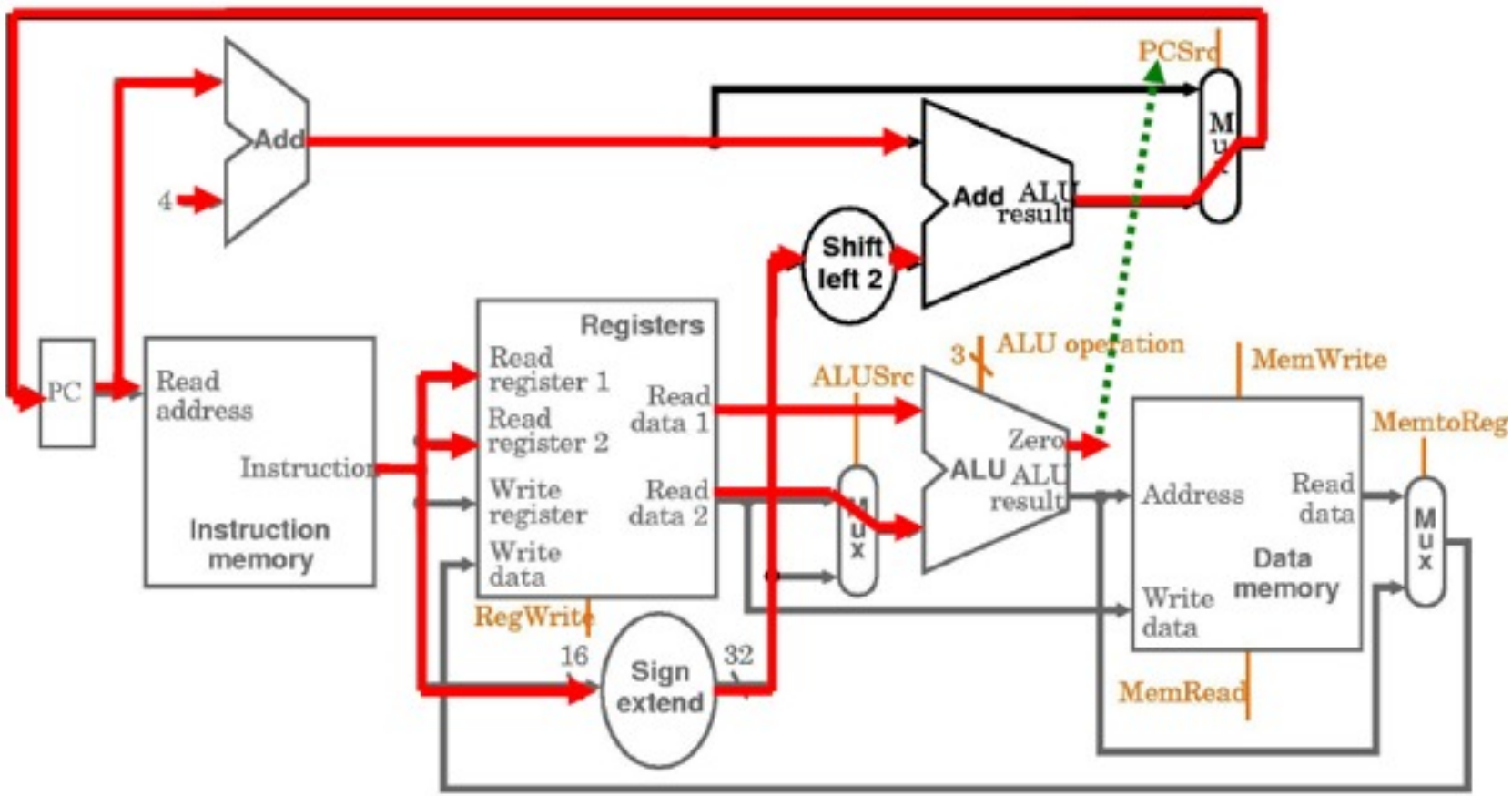
Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
beq	X	0	X	0	0	0	1

Caminho de dados para instrução beq (desvio não tomado)

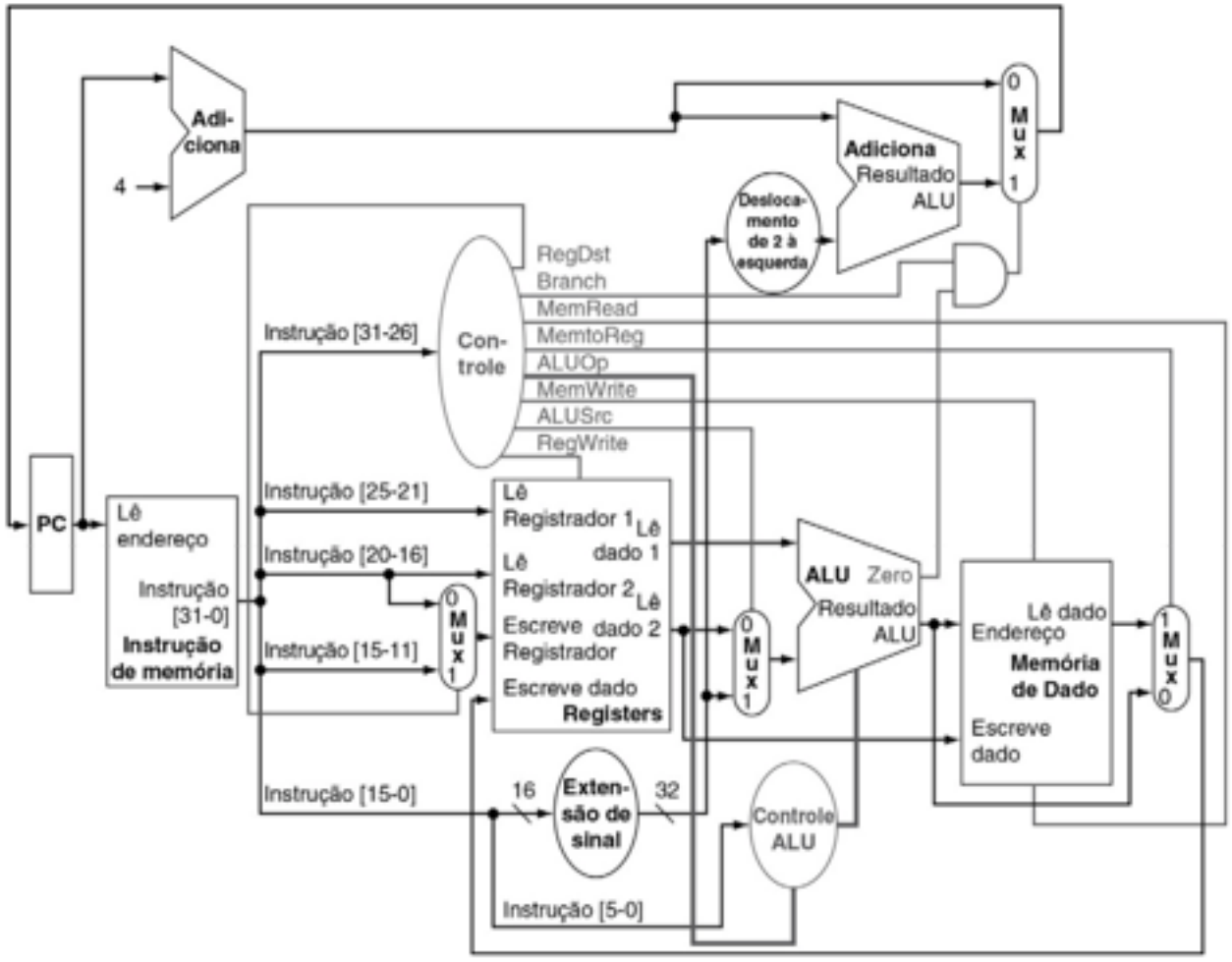


Example: **beq $28, $13, EXIT**

Caminho de dados para instrução beq (desvio tomado)



Example: **beq $28, $13, EXIT**



Instrução	RegDs	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch
formato R	1	0	0	1	0	0	0
lw	0	1	1	1	1	0	0
sw	X	1	X	0	0	1	0
beq	X	0	X	0	0	0	1

Instrução	RegDst	ALUSrc	MemtoReg	RegWrite	MemRea	MemWrite	Branch	opALU1	opALU0
formato R	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	1
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	0

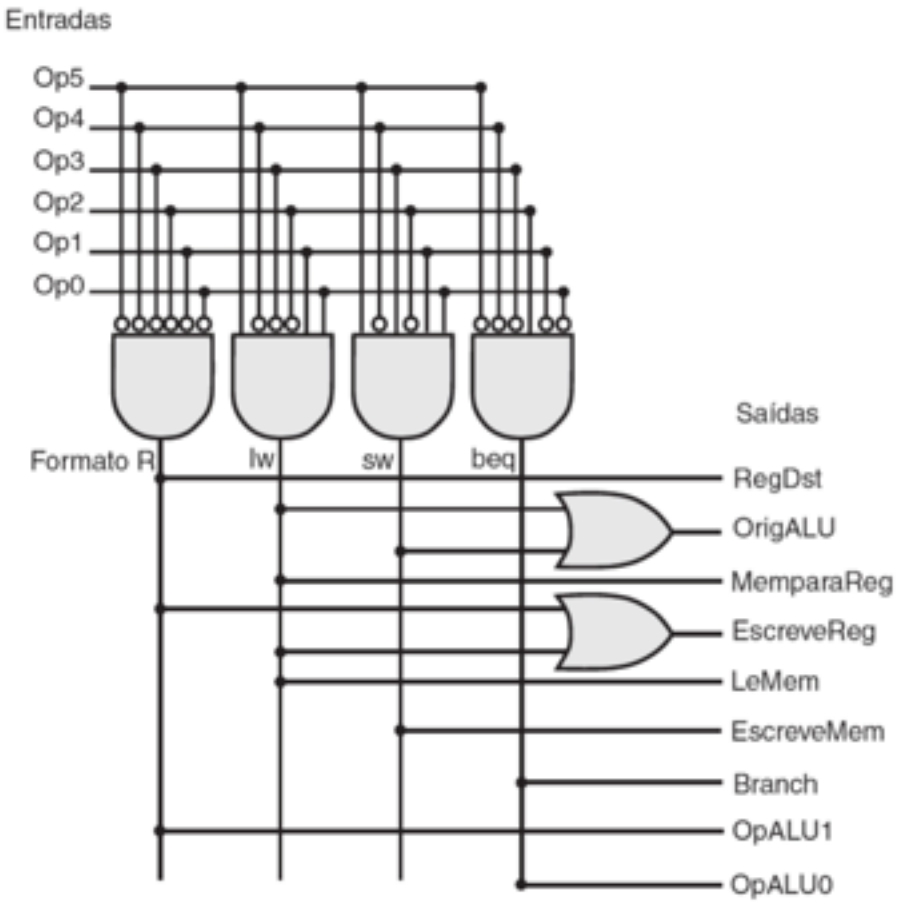
Código de operação da instrução	OpALU	Operação da Instrução	Campo Função	Operação desejada da ULA	Entrada de controle da ULA
LW	00	load word	xxxxxx	soma	0010
SW	00	store word	xxxxxx	soma	0010
Beq	01	branch equal	xxxxxx	subtração	0110
Tipo R	10	add	100000	soma	0010
Tipo R	10	subtract	100010	subtração	0110
Tipo R	10	and	100100	and	0000
Tipo R	10	or	100101	or	0001
Tipo R	10	set less than	101010	set less than	0111

Tabela Verdade do Controle

Controle	Nome do sinal	formato R	lw	sw	beq
Entradas	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Saídas	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemparaReg	0	1	X	X
	EscreveReg	1	1	0	0
	LeMem	0	1	0	0
	EscreveMem	0	0	1	0
	Branch	0	0	0	1
	OpALU1	1	0	0	0
	OpALU0	0	0	0	1

FIGURA C.2.4 A função de controle para a implementação simples de clock único é completamente satisfeita por essa tabela verdade.

Implementações



Exercício - Instrução JUMP

Estender a organização do MIPS para dar suporte a execução de JUMP, desvio incondicional

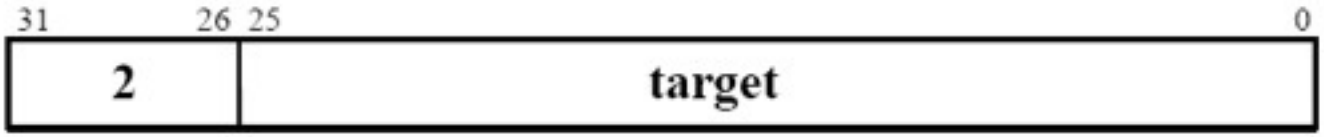
O endereço de desvio é obtido por:

{{PC+4[31 – 28]},{Instrução[25 – 0]}, 2'b00}

onde {},{} indica concatenação de bits

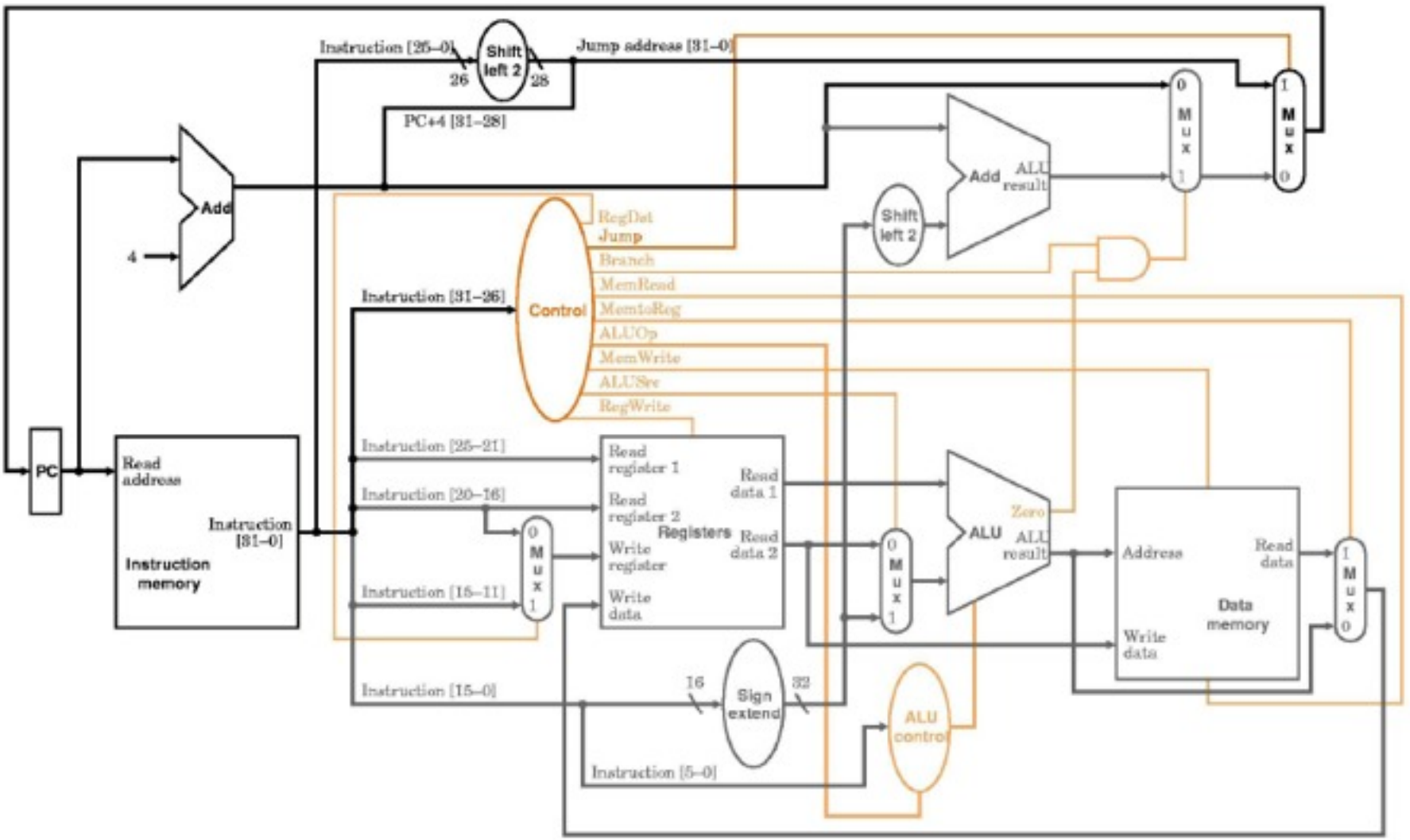
Exercício - Instrução JUMP

- Formato instruções tipo-J



- Próximo PC é formado shiftando 2 bits para esquerda os 26 bits do alvo e combinando-o com os 4 bits do PC+4
- Agora o próximo PC vai ser uma das seguintes opções:
  - PC + 4
  - Endereço alvo do beq
  - Endereço alvo do jump
- Precisamos de mais um MUX e mais um bit de controle

Exercício - Instrução JUMP

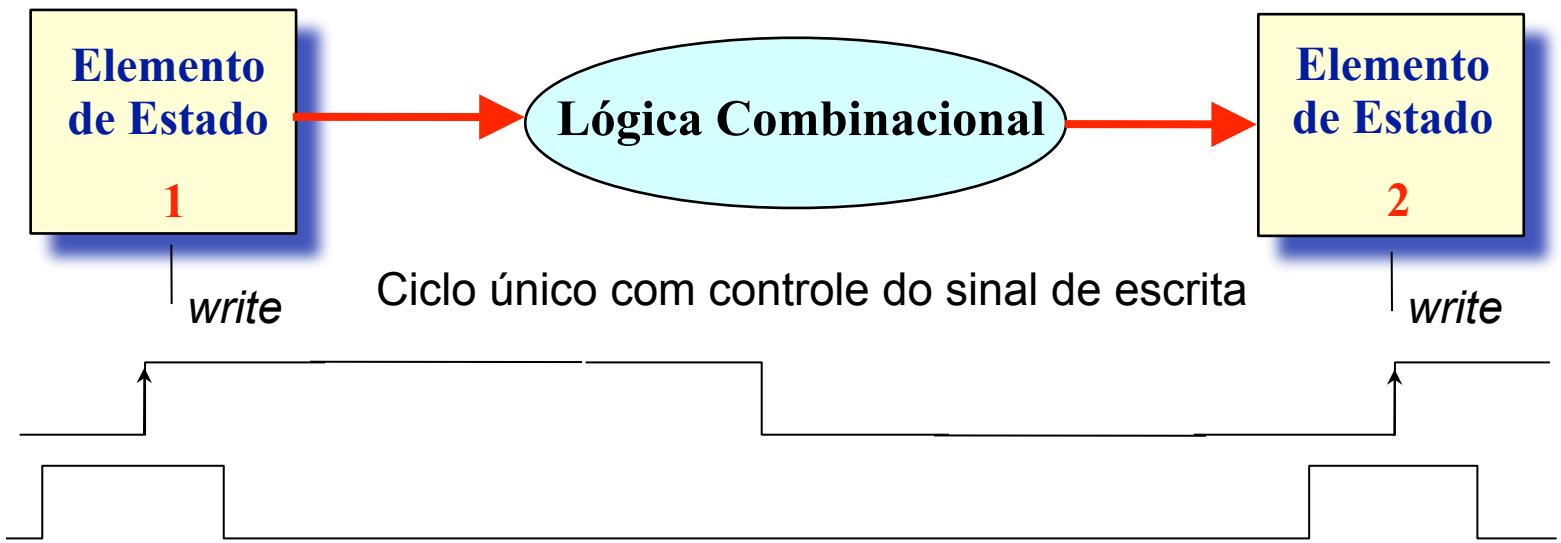


Exercício - Instrução JUMP

Instrução	RegDs	ALUSrc	MemtoReg	RegWrit	MemRead	MemWrite	Branch	opALU1	opALU	jump
formato R	1	0	0	1	0	0	0	1	0	0
lw	0	1	1	1	1	0	0	0	1	0
sw	X	1	X	0	0	1	0	0	0	0
beq	X	0	X	0	0	0	1	0	0	0
jump	X	X	X	X	0	0	0	X	X	1

Temporização

- Todas as tarefas executadas em 1 período do clock
- Elementos de estado alteram seu valor apenas na borda de subida do clock



Quais são os elementos de estado? Quando são escritos?

- PC não necessita controle de escrita (pq?)

Problemas com MIPS Uniciclo

- Período do relógio determinado pelo caminho mais longo
  - **instrução lw: (imagina se tivesse operações em Ponto Flutuante!)**
    - leitura da instrução
    - leitura do registrador de base, extensão de sinal
    - cálculo do endereço
    - leitura do dado da memória
    - escrita em registrador
- TODAS as instruções levam o mesmo tempo para executar: CPI=1

Análise de Desempenho Uniciclo

- Supondo os seguintes tempos de execução das unidades operativas:
  - Acesso a memória: 200ps
  - ULA e somadores: 100ps
  - Acesso ao banco de registradores (leitura ou escrita): 50ps
  - Multiplexadores, Unidade de Controle, acesso ao PC, Unidade de Extensão de Sinal e fios considerados sem atraso (!!!!)
- Qual das seguintes implementações seria mais rápida e quanto?

1) Implementação onde toda instrução é feita em 1 ciclo de clock de duração fixa.
2) Implementação onde toda instrução é feita em 1 ciclo de clock de duração somente o necessário (!!**exemplo ideal - didático**!!)

Para comparação consideremos um workload composto de:

25% loads, 10% stores, 45% ALU, 15% desvios condicionais, 5% jumps

- Lembrando que:

Tempo Execução = Contagem de Instruções x CPI x Período de clock

Classe de instrução	Unidades funcionais usadas pela classe de instrução				
tipo R	Busca de instrução	Acesso a registrador	ALU	Acesso a registrador	
Load word	Busca de instrução	Acesso a registrador	ALU	Acesso à memória	Acesso a registrador
Store word	Busca de instrução	Acesso a registrador	ALU	Acesso à memória	
Branch	Busca de instrução	Acesso a registrador	ALU		
Jump	Busca de instrução				

Usando esses caminhos críticos, podemos calcular o tamanho exigido para cada classe de instrução:

Classe de instrução	Memória de instrução	Leitura de registrador	Operação ALU	Memória de dados	Escrita de registrador	Total
Tipo R	200	50	100	0	50	400ps
Load word	200	50	100	200	50	600ps
Store word	200	50	100	200		550ps
Branch	200	50	100	0		350ps
Jump	200					200ps

Tempo médio:
Uniciclo fixo: 600ps
Uniciclo variável: 600x0.25+550x0.1+400x0.45+350x0.15+200x0.05=447.5ps

Fator de desempenho: n=600/447.5 = 1.34

- Quais unidades funcionais existentes seriam necessárias para implementar uma nova instrução
- LWI Rt, Rd(Rs) onde, Rt ← Mem[Reg[Rd] + Reg[Rs]]
- Seria necessário alguma unidade funcional adicional?
- Seria necessário algum sinal de controle adicional?