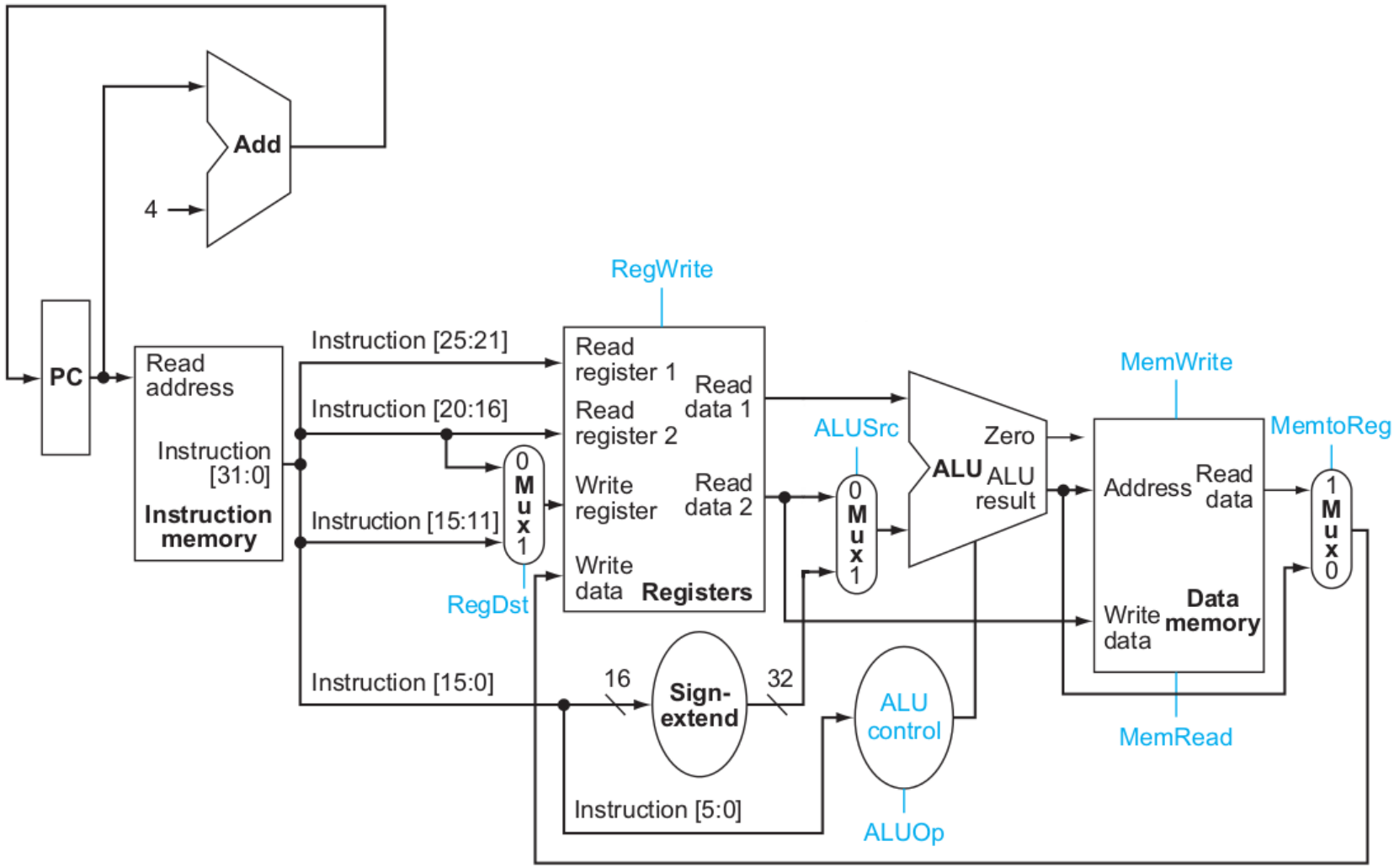
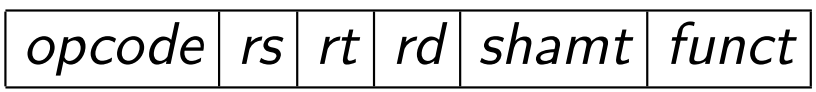
Instruções aritméticas e lógicas e de acesso à memória

Caminho de dados (inclui `sw`)



Comparação de add e lw

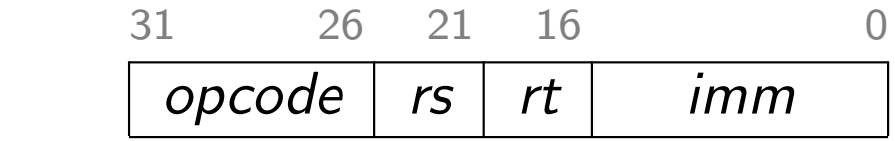
`add rd, rs, rt`	`lw rt, imm(rs)`
Leitura da instrução	
Identificação da instrução	
Leitura dos registos (`rs` e `rt`)	Leitura do registo (`rs`)
Soma (de `rs` e `rt`)	Soma (de `rs` e `imm`)
	Acesso à memória
Escrita do registo (`rd`)	Escrita do registo (`rt`)



Instrução beq

`beq rs, rt, imm`

É uma instrução tipo-I



bits	6	5	5	16

Execução

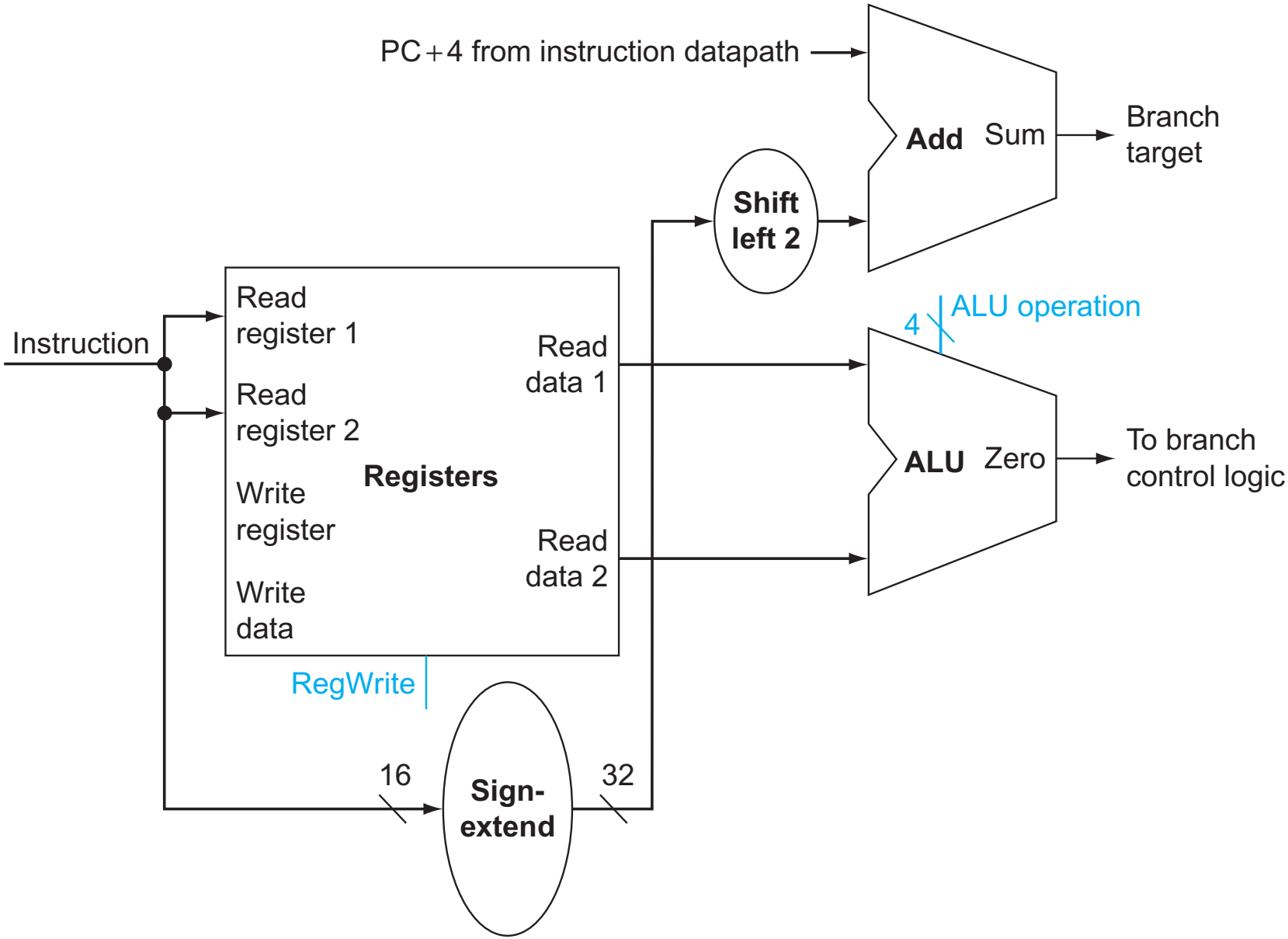
1. Leitura da instrução

   Identificação da instrução
2. Leitura do conteúdo dos registos `rs` e `rt`
3. Comparação dos valores lidos
4. Escolha do endereço da próxima instrução a executar:

$$\mathrm{PC} + 4 \quad \text{ou} \quad 4 \times \texttt{imm} + \mathrm{PC} + 4$$

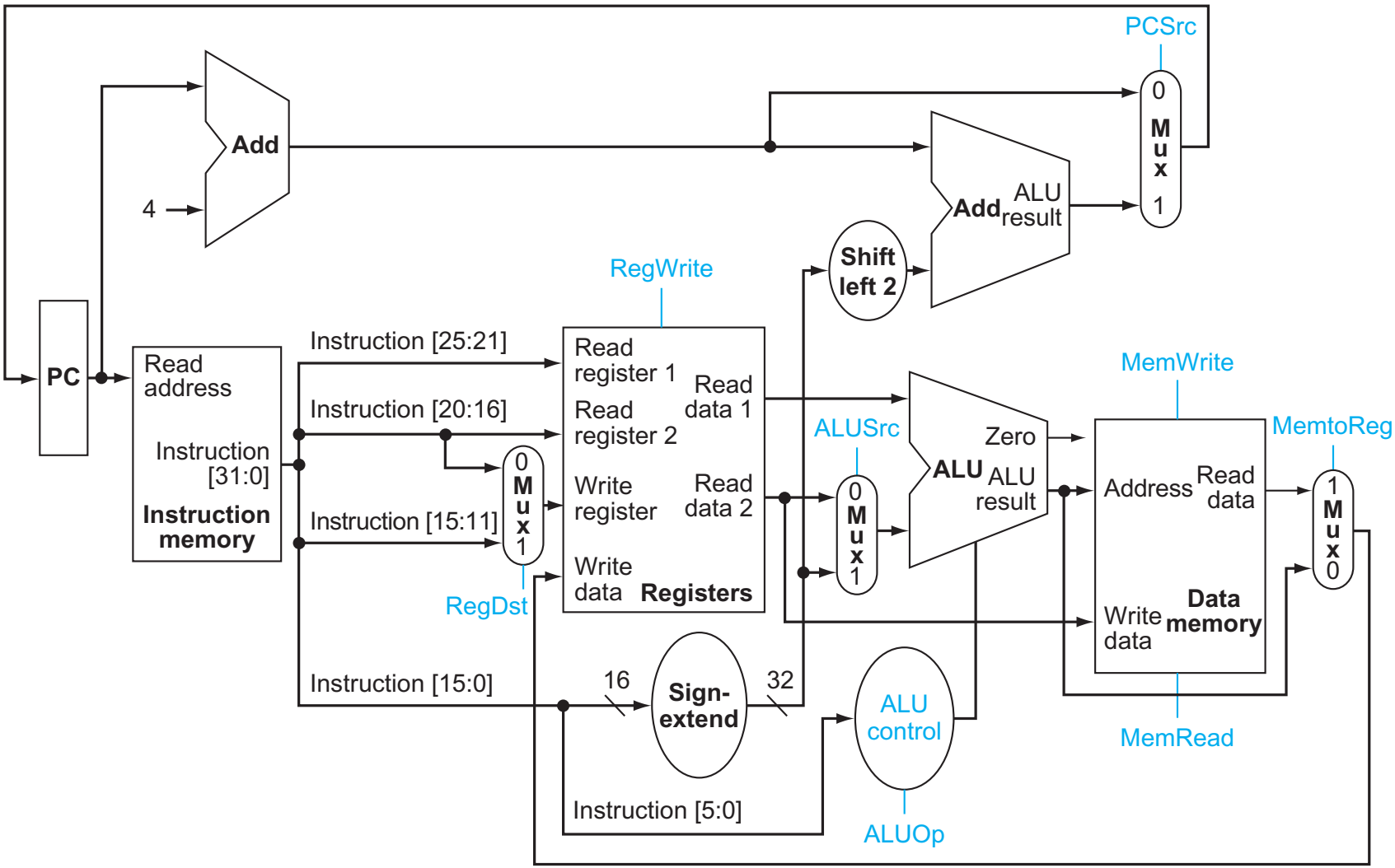
Valores calculados em paralelo

Caminho de dados parcial para beq

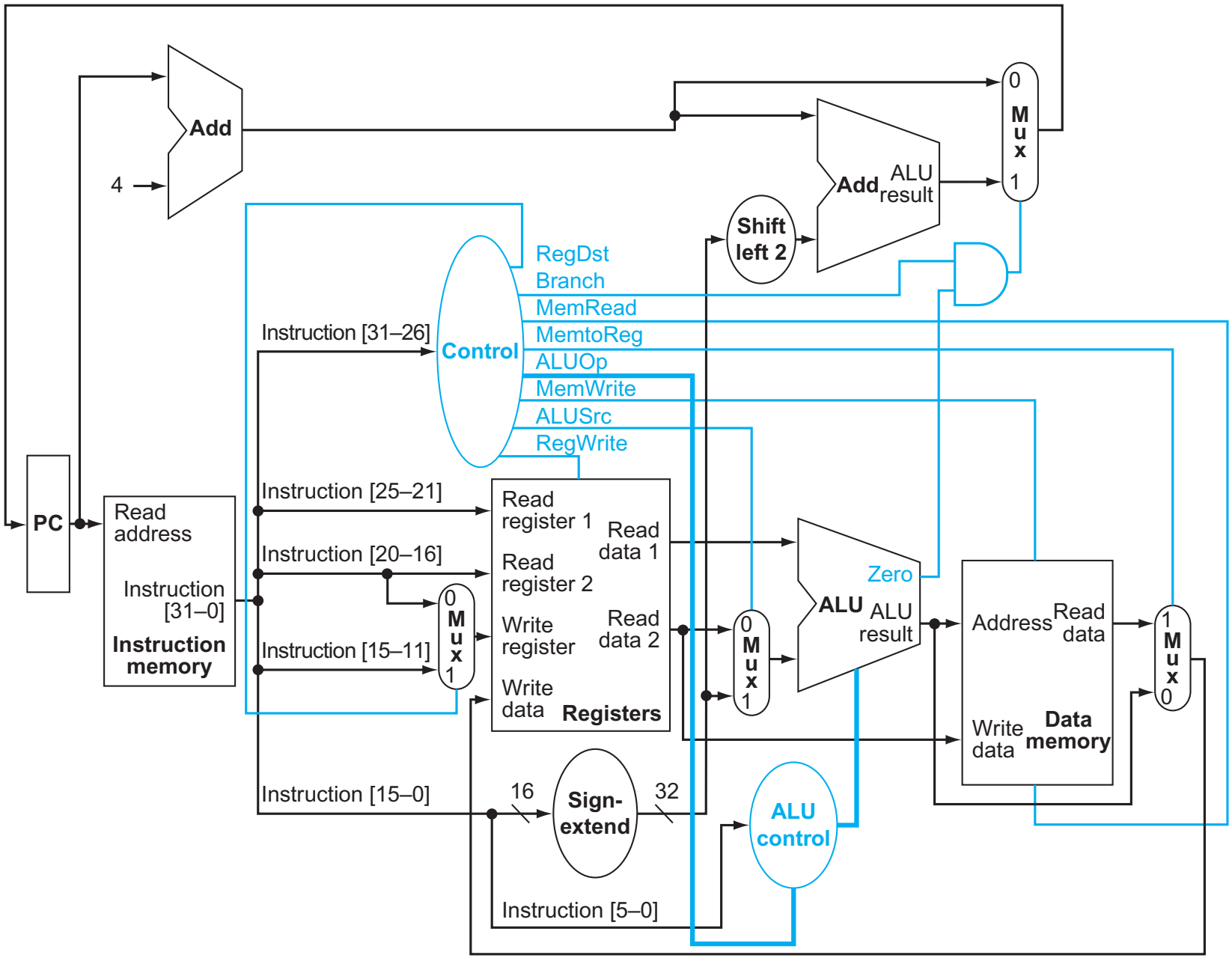


Caminho de dados completo

Para as instruções `add`, `sub`, `and`, `or`, `slt`, `lw`, `sw` e `beq`



Caminho de dados e controlo



Controlo da operação da ALU

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
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

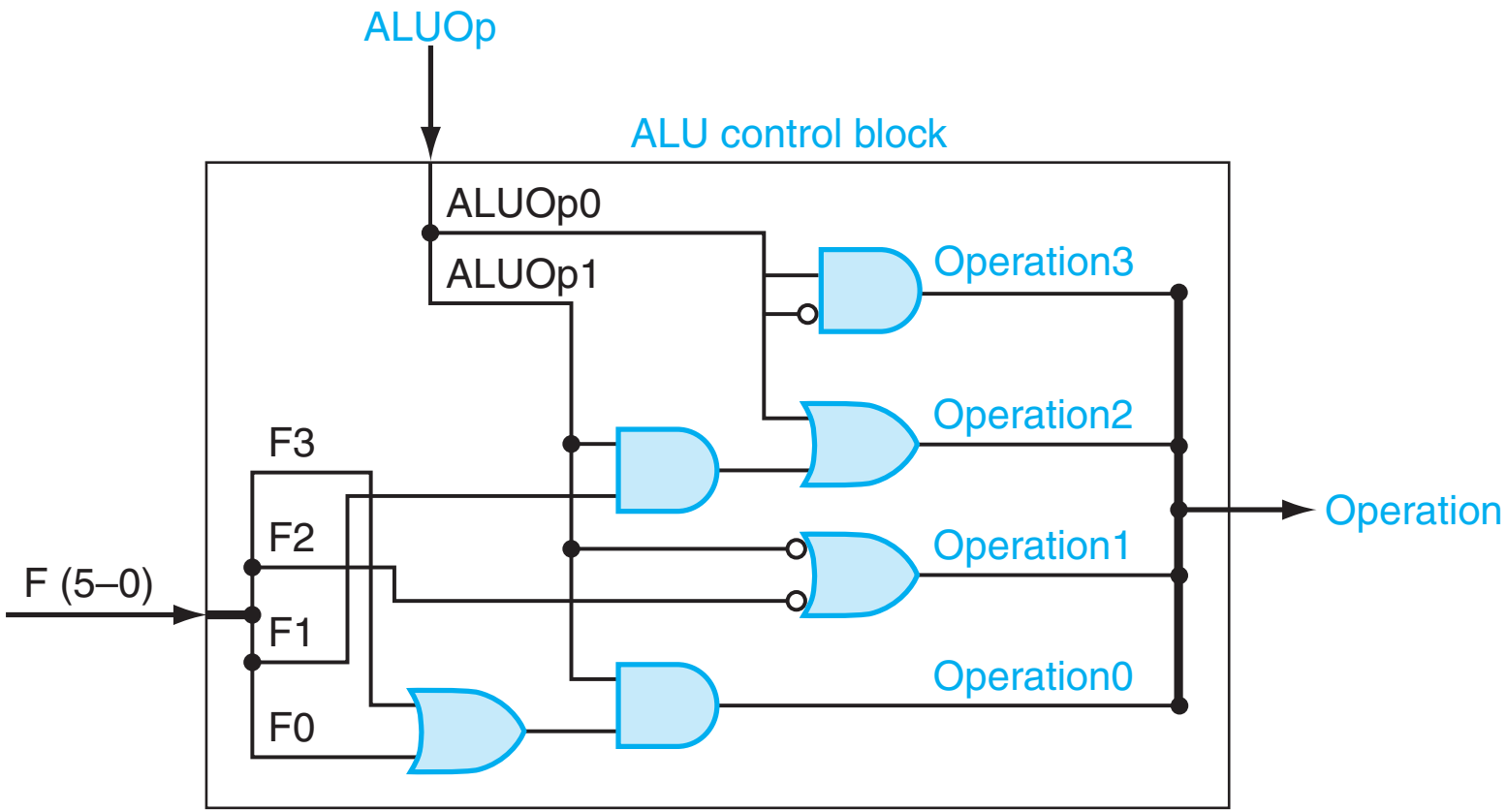
Lógica de controlo da ALU

$$\text{Operation}_3 = 0$$

$$\text{Operation}_2 = \overline{\text{ALUOp}_0} \text{ OR } \text{ALUOp}_1 \text{ AND } F_1$$

$$\text{Operation}_1 = \overline{\text{ALUOp}_1} \text{ OR } \overline{F_2}$$

$$\text{Operation}_0 = \text{ALUOp}_1 \text{ AND } (F_3 \text{ OR } F_0)$$



Sinais de controlo (1)

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

$$\text{PCSrc} = \text{Branch AND Zero}$$

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	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

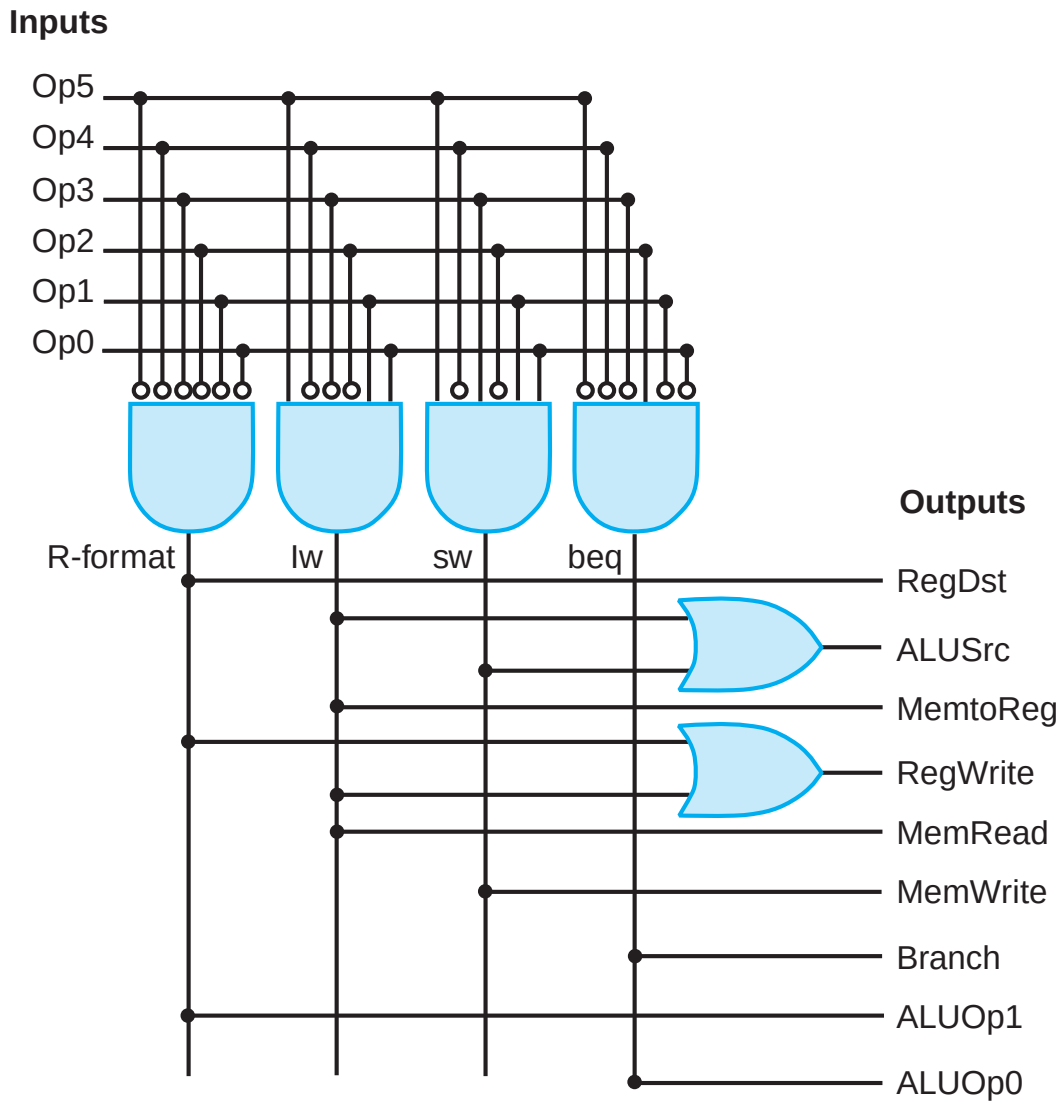
Sinais de controlo (2)

Valor dos sinais em função do *opcode*

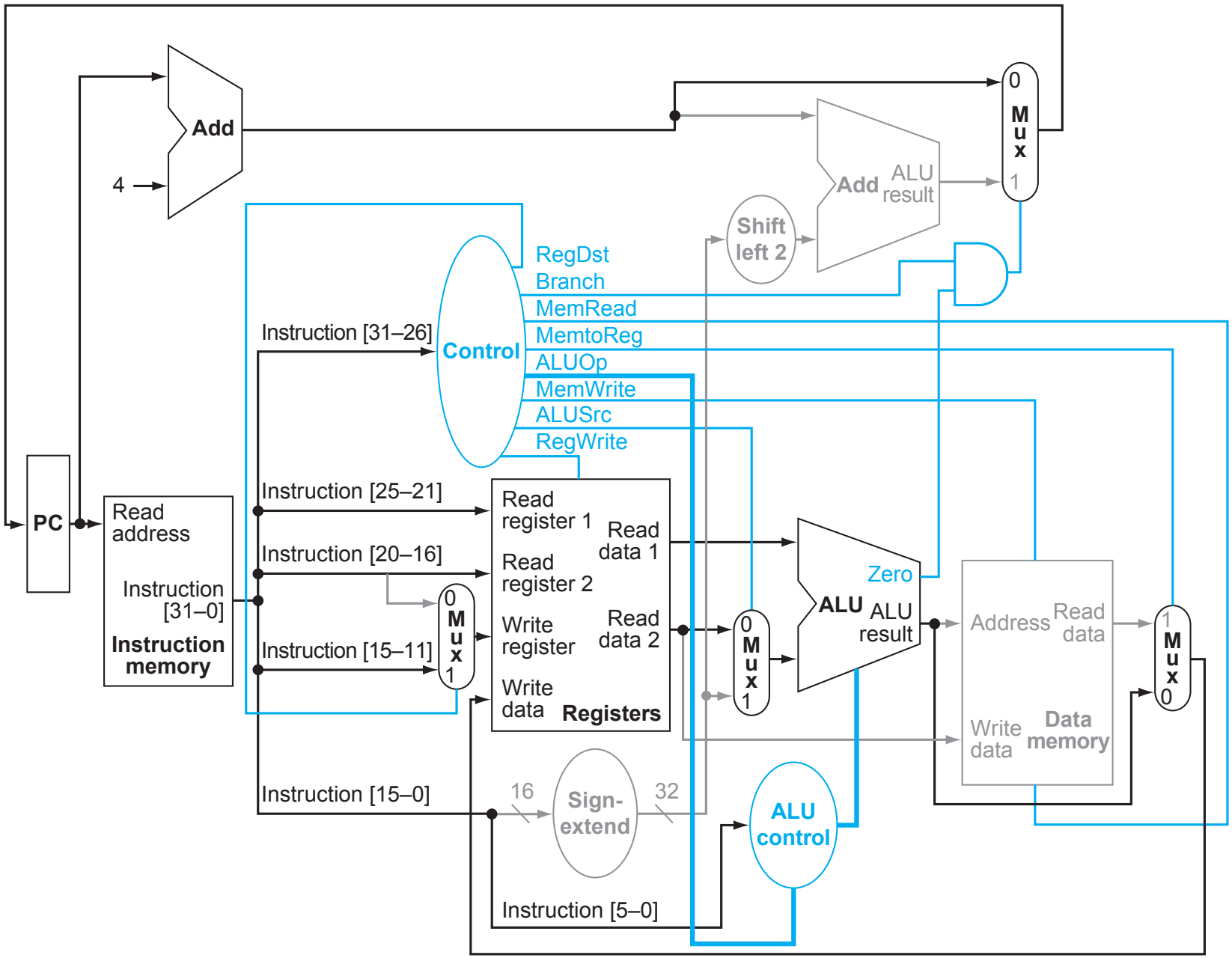
Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Lógica de controlo

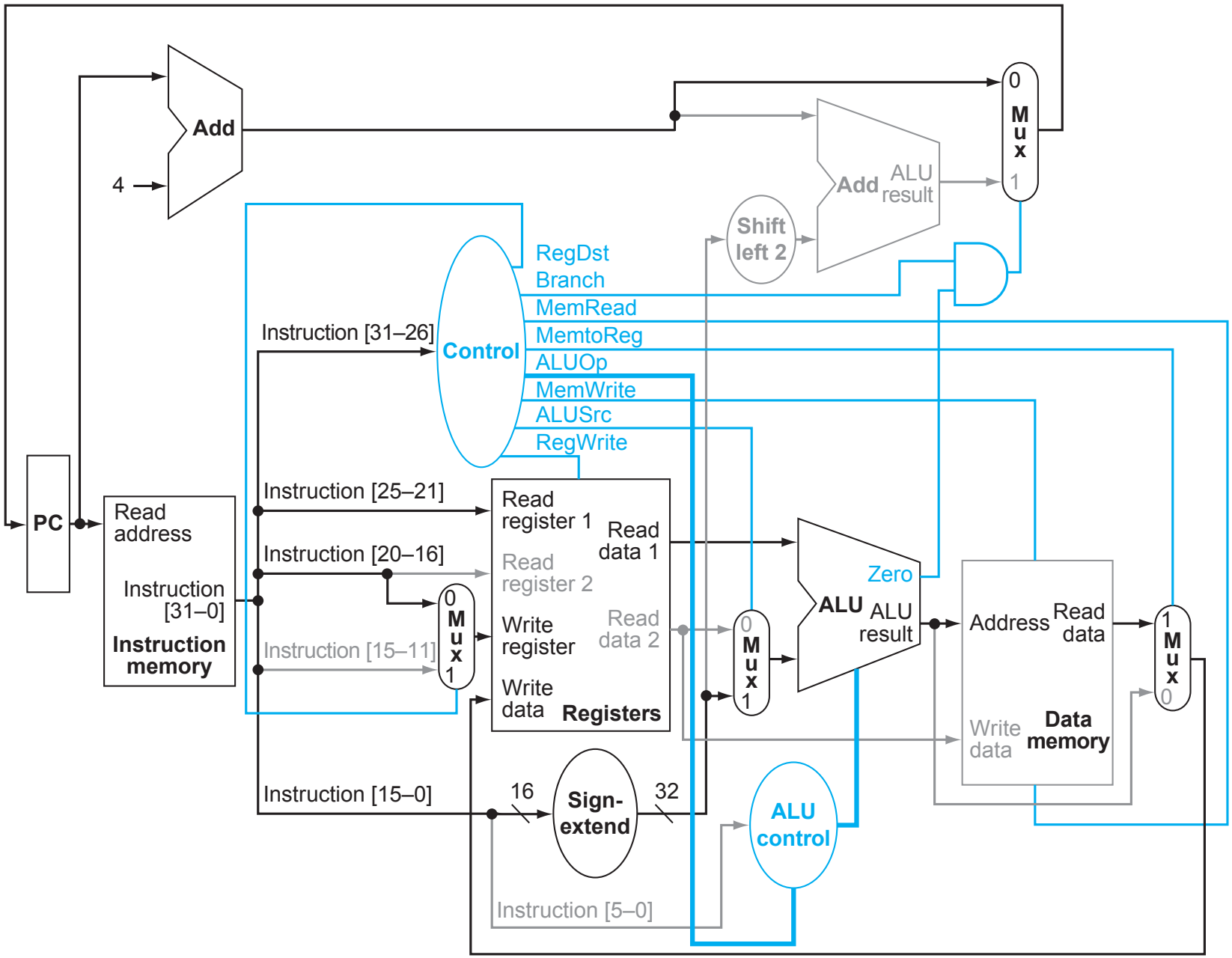
Descodificador



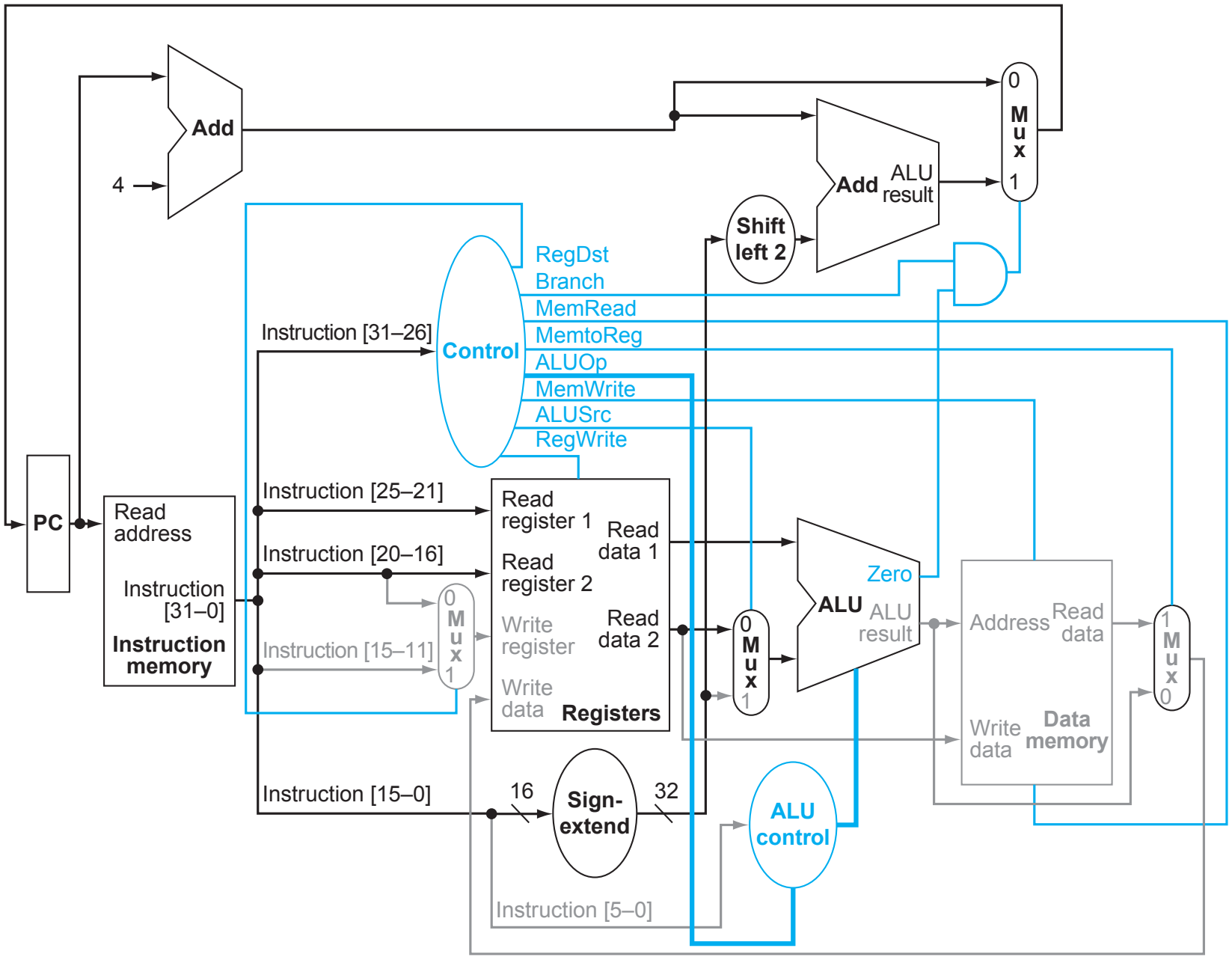
Partes activas para instruções tipo-R



Partes activas para `lw`



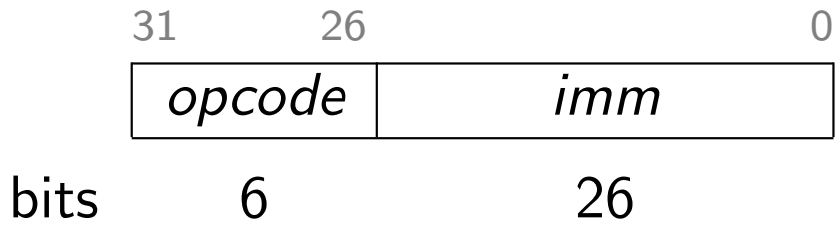
Partes activas para beq



Instrução `j` (`jump`)

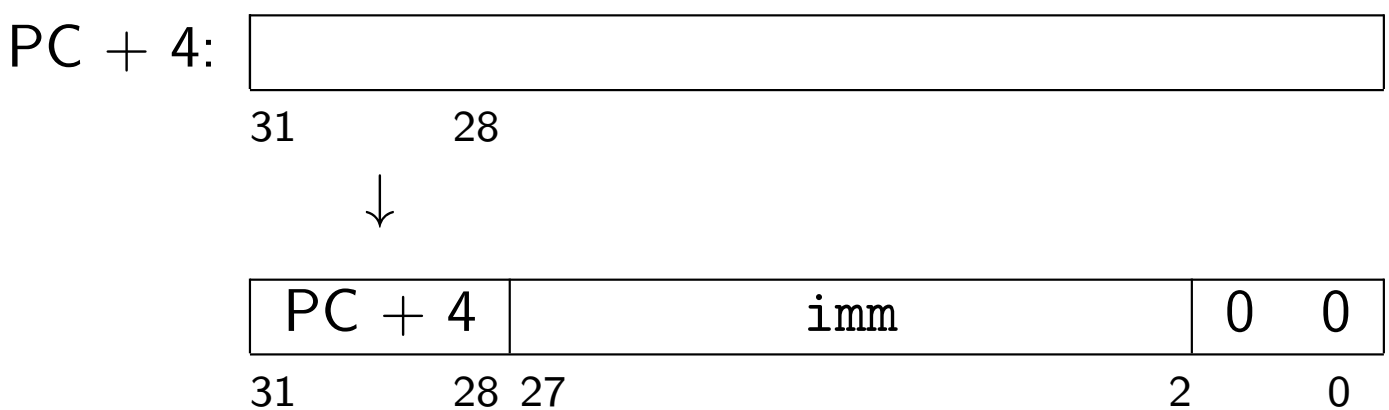
```
j imm
```

É uma instrução tipo-J

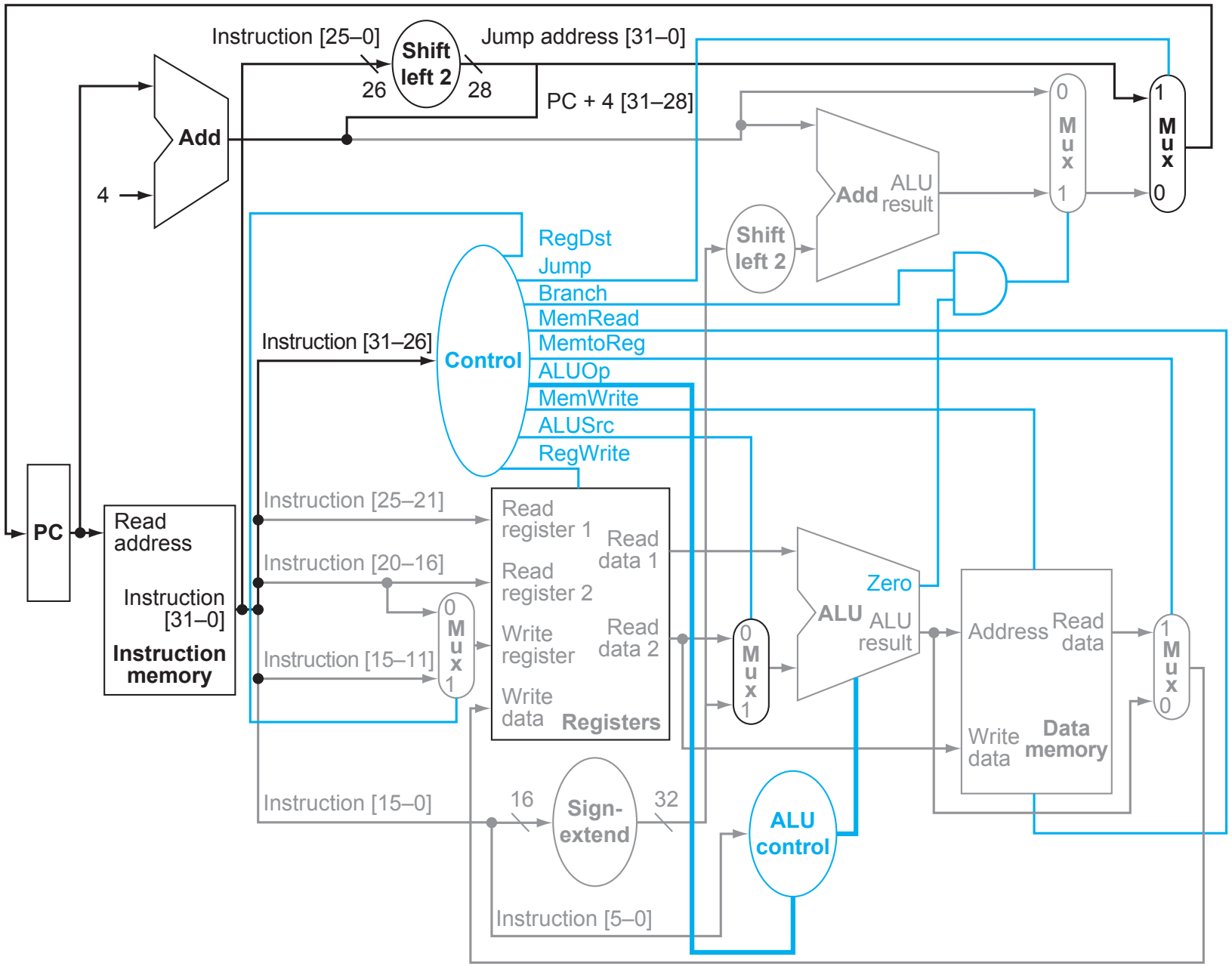


Execução

1. Leitura da instrução
   Identificação da instrução
2. Cálculo do endereço da próxima instrução a executar:



Caminho de dados e controlo com `jump`



Caminho de dados completo

Observações

Trata-se de uma implementação monociclo

Todas as instruções executam num único ciclo de relógio

E o *delay slot*?

- Nesta implementação não faz sentido falar de *delay slot*

  No fim do ciclo em que uma instrução de salto é executada, é conhecido o endereço da próxima instrução a ser executada, seja o destino do salto ou PC + 4

Duração de um ciclo (1)

Latência de um circuito

Tempo desde que os valores estão presentes nas entradas do circuito até às suas saídas estarem estabilizadas

Exemplos

Memória(s)	200 ps
ALU	200 ps
Banco de registos	100 ps

Caminho crítico de uma instrução

Caminho (no processador) cuja duração, se aumentar, provoca o aumento da duração da instrução

Compreende a sequência de operações, efectuadas durante a execução da instrução, cuja duração é mais longa

Fases da execução de add, lw e beq

`add rd, rs, rt`	`lw rt, imm(rs)`	`beq rs, rt, imm`
Leitura da instrução		
Identificação da instrução		
Leitura de `rs` e `rt`	Leitura de `rs`	Leitura de `rs` e `rt`
Soma de `rs` e `rt`	Soma de `rs` e `imm`	Comparação de `rs` e `rt`
	Acesso à memória	
Escrita de `rd`	Escrita de `rt`	

(O cálculo do novo valor do PC é efectuado em paralelo)

Duração de um ciclo (2)

Implementação monociclo

Todas as instruções levam o mesmo tempo: 1 ciclo

Duração do ciclo de relógio tem de acomodar a instrução cujo caminho crítico tem maior duração

Tempos por instrução

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add`, `sub`, `AND`, `OR`, `slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps

A instrução com a duração mais longa é `lw`

$$T \geq 800\,\text{ps} \quad \equiv \quad f \leq 1.25\,\text{GHz}$$