



# ORGANIZAÇÃO E ARQUITETURA DE CÓMPUTADORES

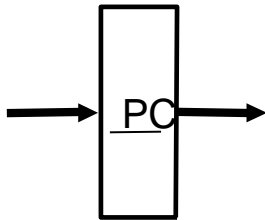
**Arquitetura RISC-V Monociclo**  
*Via de Dados e Unidade de Controle*

*Profª. Fabiana F F Peres*

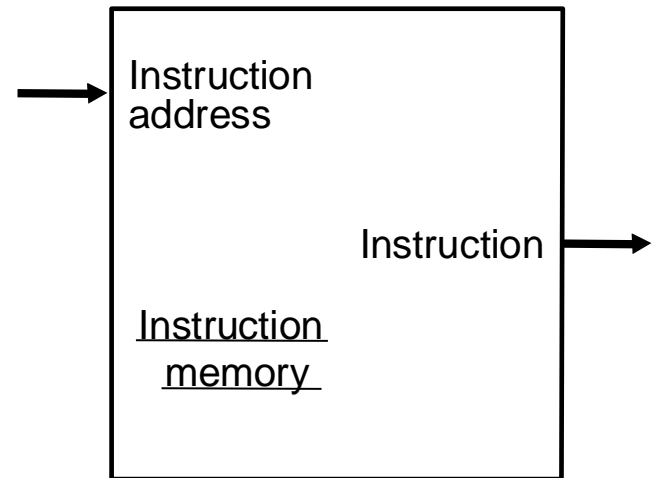
*Apoio: Camile Bordini*

# *Execução de Instruções*

- É dividido em vários passos:
  - **Busca a próxima instrução da memória;**



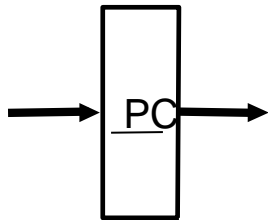
b. Program counter



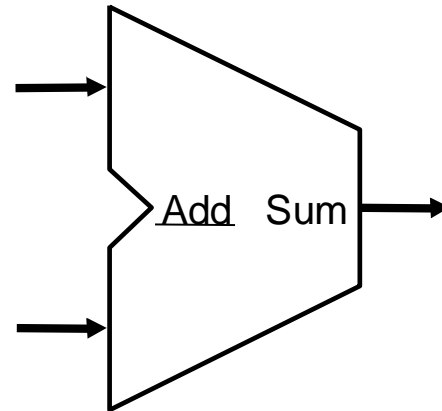
a. Instruction memory

# *Execução de Instruções*

- É dividido em vários passos:
  - Busca a próxima instrução da memória;
  - **Atualiza o contador de programa (Registrador) para que ele aponte para a próxima instrução;**



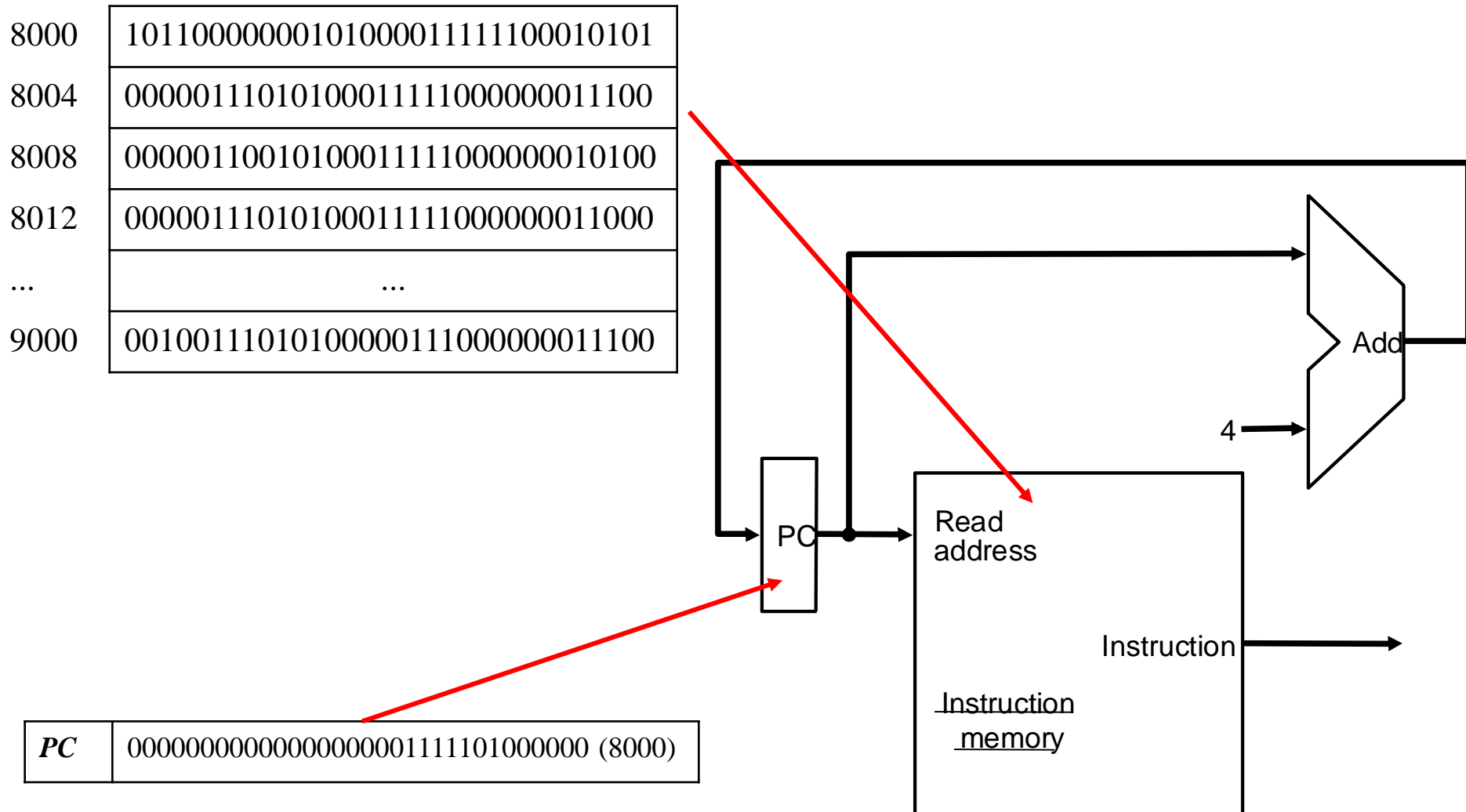
b. Program counter



c. Adder

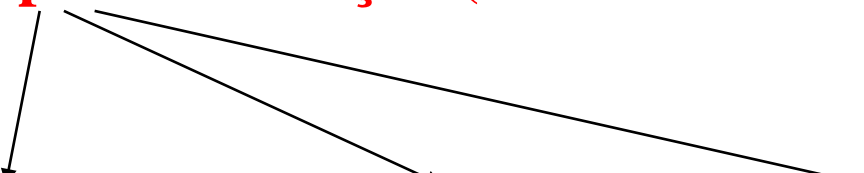
# *Via de Dados*

(Conexão entre os elementos)



# *Execução de Instruções*

- É dividido em vários passos:
  - Busca a próxima instrução da memória;
  - Atualiza o contador de programa (Registrador) para que ele aponte para a próxima instrução;
  - **Determina o tipo da instrução (Decodifica a instrução);**



<b>Tipo-R</b>	<i>funct7</i>	<i>rs2</i>	<i>rs1</i>	<i>funct3</i>	<i>rd</i>	<i>opcode</i>
<b>Tipo-I</b>	<i>imm[11:0]</i>		<i>rs1</i>	<i>funct3</i>	<i>rd</i>	<i>opcode</i>
<b>Tipo-S</b>	<i>imm[11:5]</i>	<i>rs2</i>	<i>rs1</i>	<i>funct3</i>	<i>imm[4:0]</i>	<i>opcode</i>
<b>Tipo-B</b>	<i>imm[12]/imm[10:5]</i>	<i>rs2</i>	<i>rs1</i>	<i>funct3</i>	<i>imm[4:1]/imm[11]</i>	<i>opcode</i>
<b>Tipo-U</b>	<i>imm[31-12]</i>				<i>rd</i>	<i>opcode</i>
<b>Tipo-J</b>	<i>imm[20]/imm[10:1]/imm[11]/imm[19:12]</i>				<i>rd</i>	<i>opcode</i>

# Conjunto de Instruções

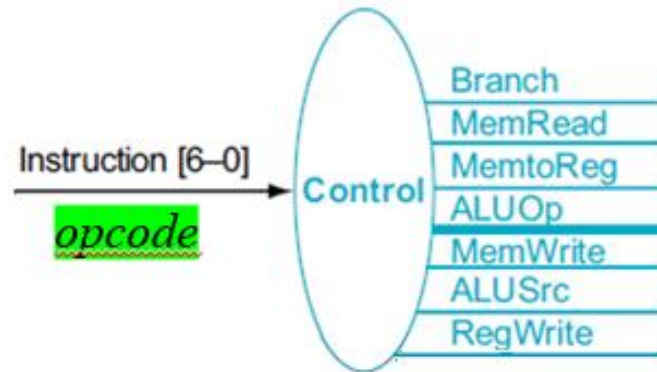
Classe de instruções	Instruções	Tipo
Instruções aritméticas	<b>add rd, rs1, rs2</b>	<b>R</b>
	<b>sub rd, rs1, rs2</b>	<b>R</b>
	<b>addi rd, rs1, imm</b>	<b>I</b>
Instruções de transferência de dados	<b>lw rd, imm(rs1)</b>	<b>I</b>
	<b>ld rd, imm(rs1)</b>	<b>I</b>
	<b>sw rs2, imm(rs1)</b>	<b>S</b>
	<b>sd rs2, imm(rs1)</b>	<b>S</b>
Instruções lógicas	<b>or rd, rs1, rs2</b>	<b>R</b>
	<b>and rd, rs1, rs2</b>	<b>R</b>
	<b>xor rd, rs1, rs2</b>	<b>R</b>
Instruções de deslocamentos	<b>sll rd, rs1, rs2</b>	<b>R</b>
	<b>srl rd, rs1, rs2</b>	<b>R</b>
	<b>slli rd,rs1, imm</b>	<b>I</b>
Instruções de desvios condicionais	<b>bne rs1, rs2, L</b>	<b>B</b>
	<b>beq rs1, rs2, L</b>	<b>B</b>
Instruções de desvios incondicionais	<b>jal rd, imm</b>	<b>J</b>
	<b>jalr rd, imm(rs1)</b>	<b>I</b>

Mas como a Unidade de  
Controle **decodifica a**  
**instrução?**

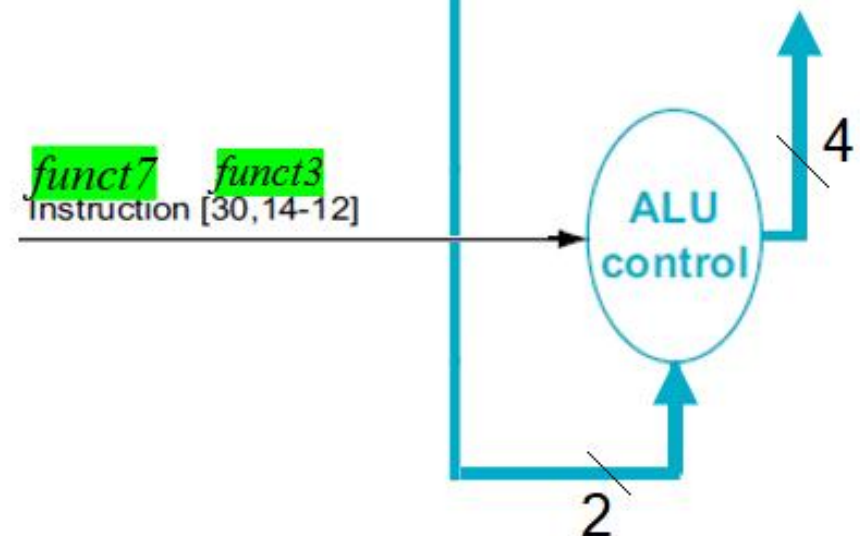
# Unidade de Controle



# Elementos Funcionais - Via de Dados



- **ALUOp**: 2 bits derivados do **opcode** da instrução e transferidos à **ALU control**, que junto com **funct3** e **funct7** fornece para a ALU 4 bits (operação a ser executada)



# Elementos Funcionais - Via de Dados

- Se **ALUOp=00**: é uma instrução *lw* ou *sw*, e indica para a **ALU control** para ser realizado uma **soma**
  - Atualizando a entrada da ALU para 4 bits: **0010**

Obs: **X** representa “*don't care*”, não importando se é 0 ou 1

Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
lw	00	load word	XXXXXXX	XXX	add	0010
sw	00	store word	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

# *Elementos Funcionais - Via de Dados*

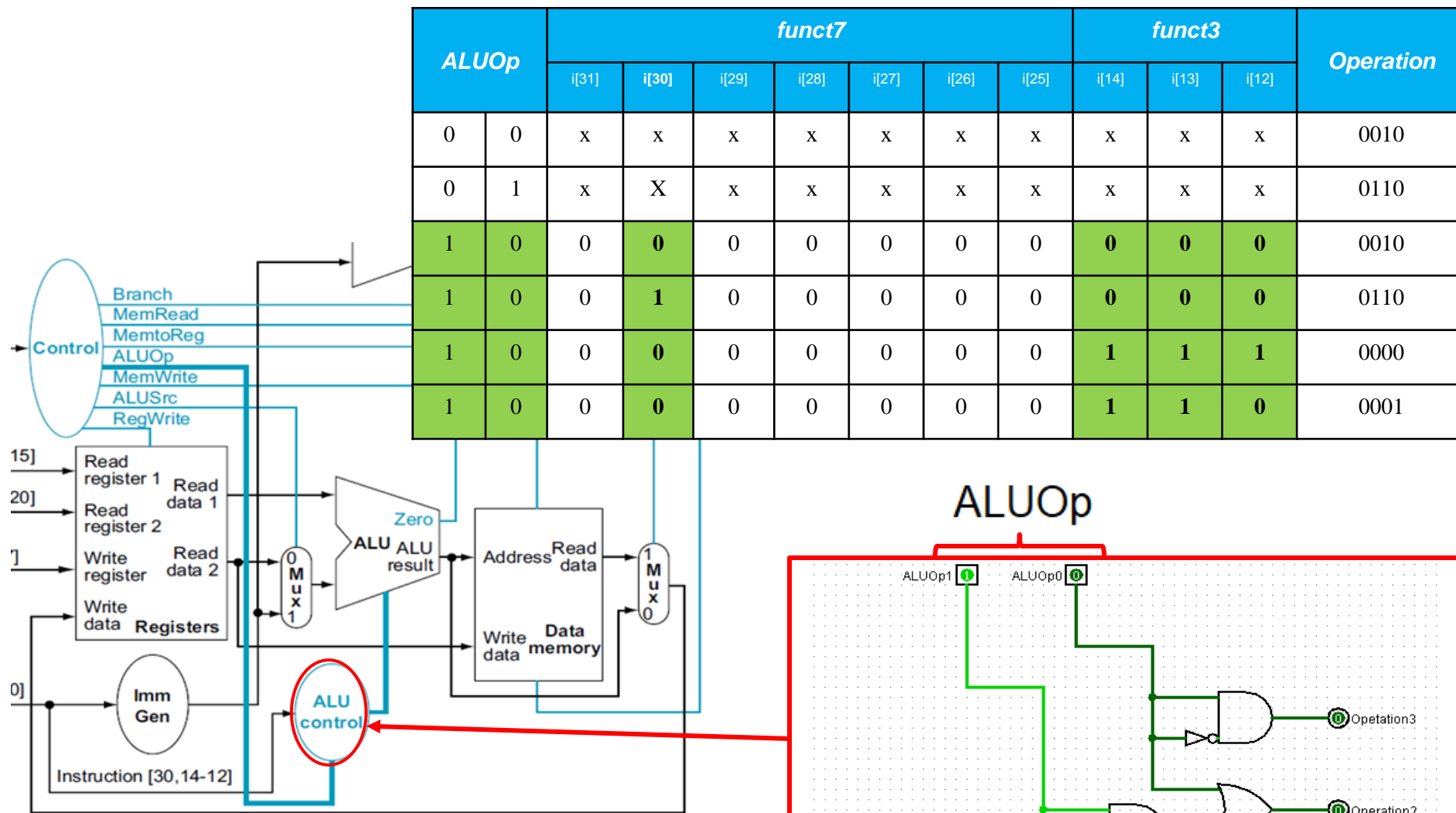
- Se **ALUOp=01**: é uma instrução *beq*, e indica para a **ALU control** para ser realizado uma **subtração**
  - Atualizando a entrada da ALU para 4 bits: **0110**

Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
lw	00	load word	XXXXXXX	XXX	add	0010
sw	00	store word	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

# Elementos Funcionais - Via de Dados

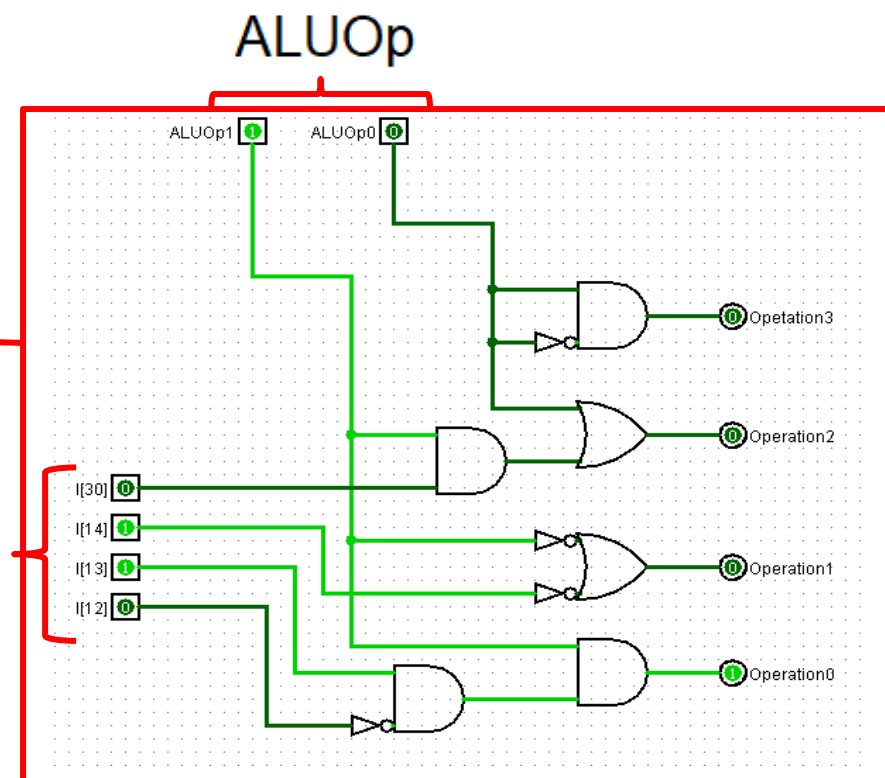
- Se **ALUOp=10**: é uma instrução do **Tipo R**, portanto, necessário verificar nos campos **funct3**, **funct7**
  - Atualizando a entrada da ALU para 4 bits, conforme for a operação desejada:
    - soma: **0010**
    - AND: **0000**
    - subtração: **0110**
    - OR: **0001**

Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
lw	00	load word	XXXXXXXX	XXX	add	0010
sw	00	store word	XXXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001



ALUOp		funct7							funct3			Operation
		i[31]	i[30]	i[29]	i[28]	i[27]	i[26]	i[25]	i[14]	i[13]	i[12]	
0	0	x	x	x	x	x	x	x	x	x	x	0010
0	1	x	X	x	x	x	x	x	x	x	x	0110
1	0	0	0	0	0	0	0	0	0	0	0	0010
1	0	0	1	0	0	0	0	0	0	0	0	0110
1	0	0	0	0	0	0	0	0	1	1	1	0000
1	0	0	0	0	0	0	0	0	1	1	0	0001

Instruction [30,14-12]



# Exercício

Quais os valores assumidos pelos sinais de controle *Branch*, *ALUSrc*, *MemToReg*, *ALUOp*, *RegWrite*, *MemRead*, *MemWrite*

a) em instruções Tipo-R

b) na instrução Load

c) na instrução Store

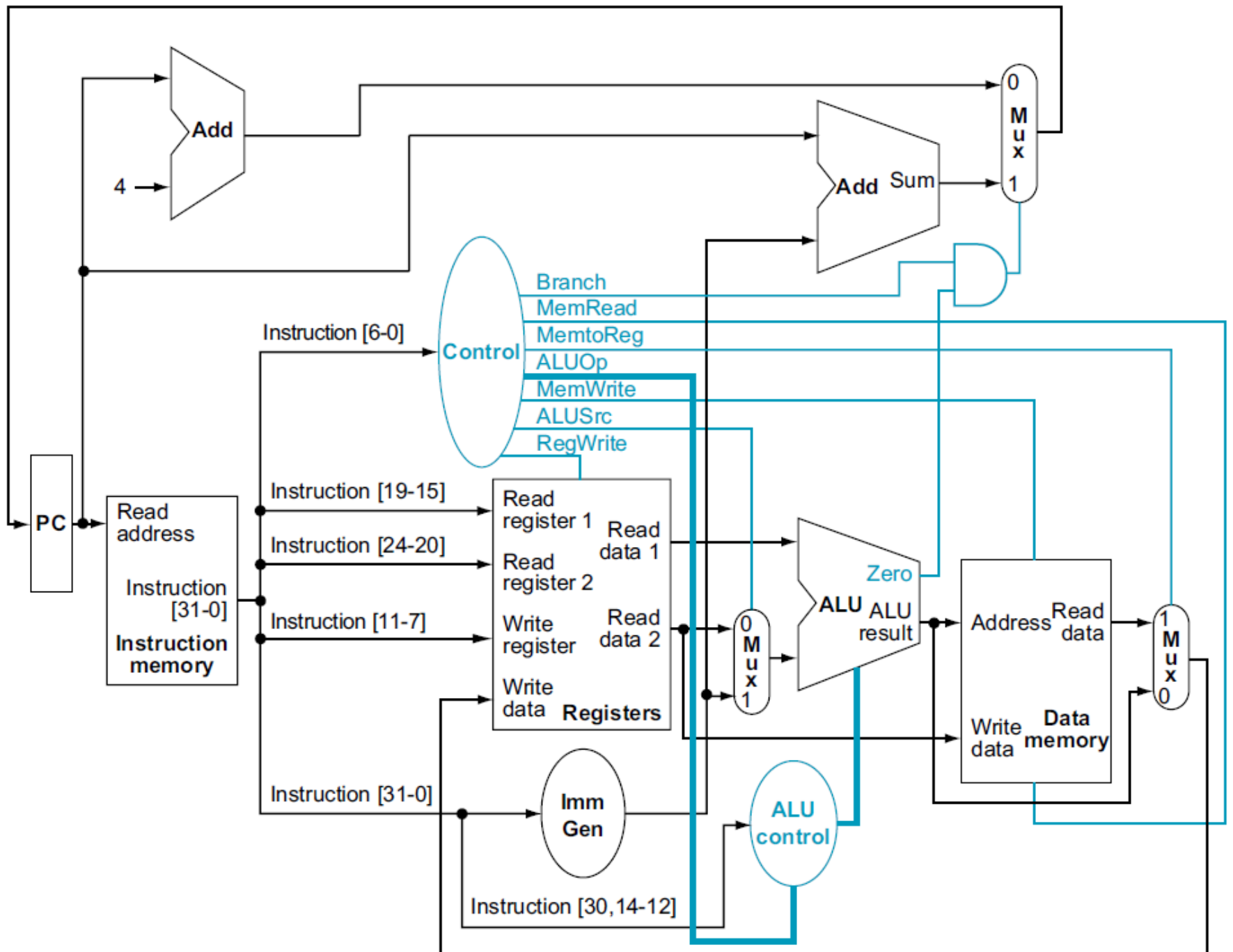
d) na instrução Beq

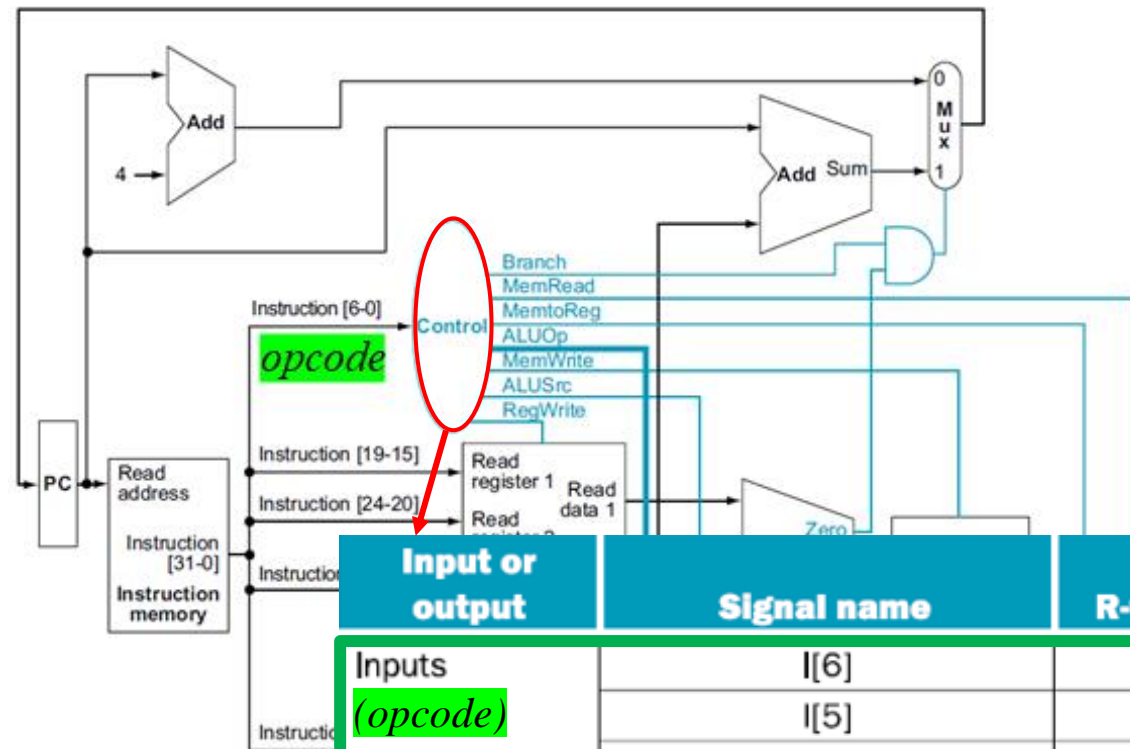
	<i>add/sub and/or</i>
Branch	
ALUSrc	
ALUOp	
MemRead	
MemWrite	
MemToReg	
RegWrite	

	<i>lw</i>
Branch	
ALUSrc	
ALUOp	
MemRead	
MemWrite	
MemToReg	
RegWrite	

	<i>sw</i>
Branch	
ALUSrc	
ALUOp	
MemRead	
MemWrite	
MemToReg	
RegWrite	

	<i>beq</i>
Branch	
ALUSrc	
ALUOp	
MemRead	
MemWrite	
MemToReg	
RegWrite	



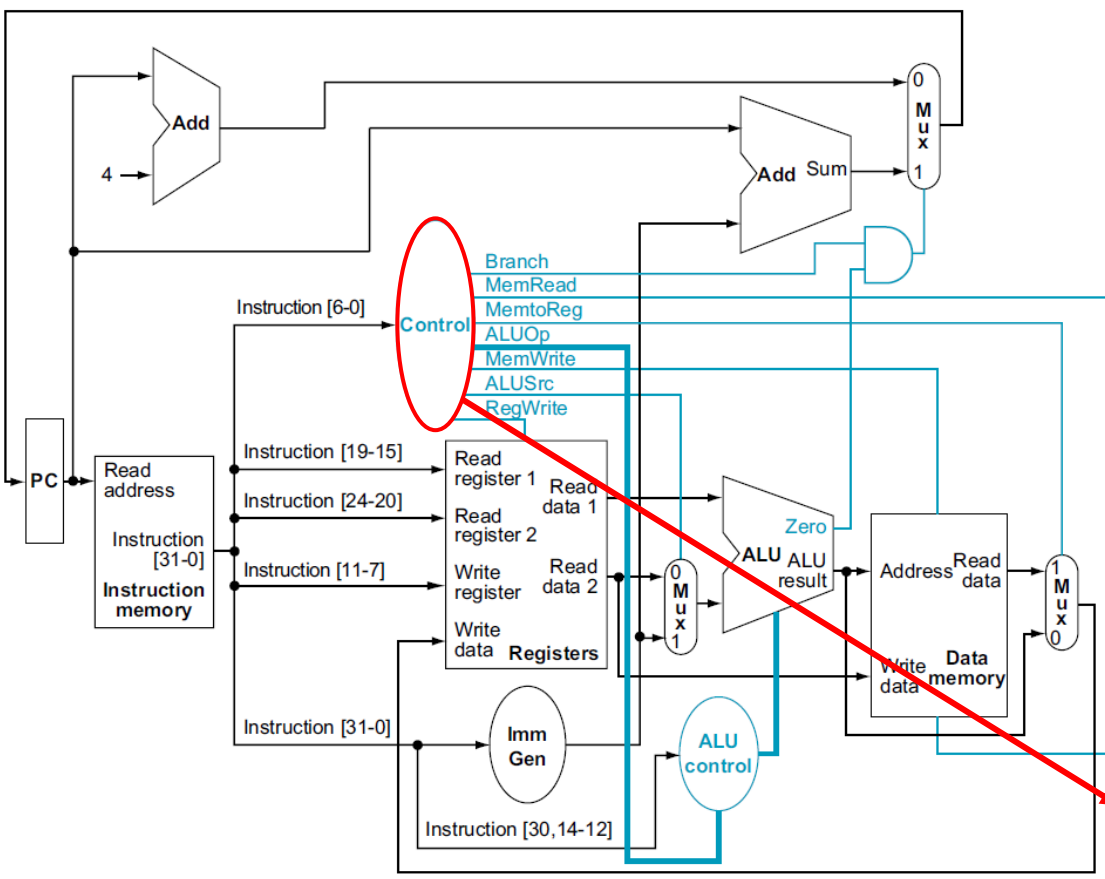


	<b>opcode</b>	
<i>add</i>	0110011	51
<i>sub</i>	0110011	51
<i>lw</i>	0000011	3
<i>sw</i>	0100011	35
<i>beq</i>	1100011	99

Input or output	Signal name	R-format	lw	sw	beq
Inputs (opcode)	I[6]	0	0	0	1
	I[5]	1	0	1	1
	I[4]	1	0	0	0
	I[3]	0	0	0	0
	I[2]	0	0	0	0
	I[1]	1	1	1	1
	I[0]	1	1	1	1
Outputs	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

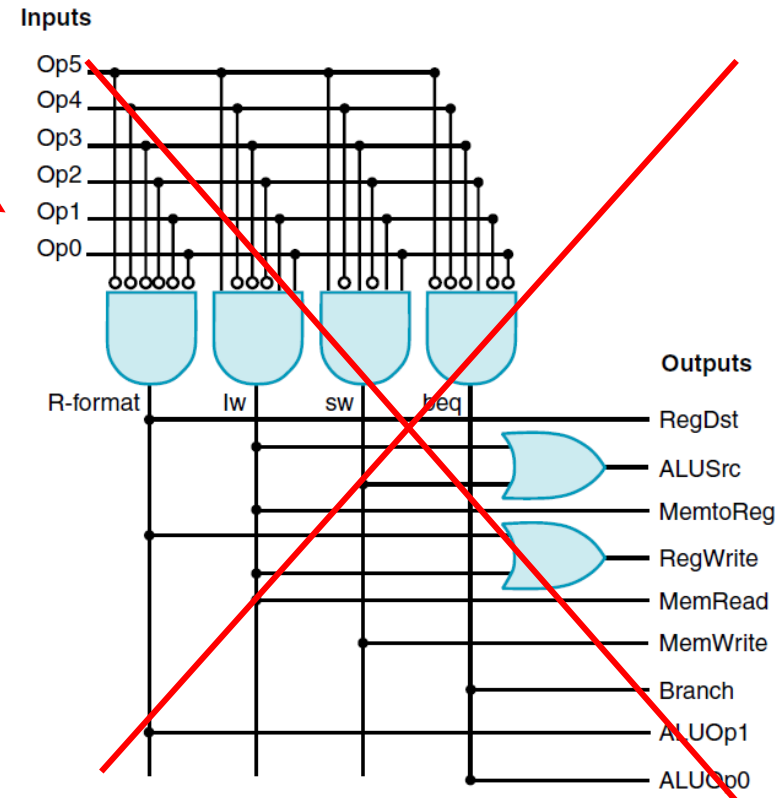


E como a Unidade de  
Controle **é implementada?**



- Implementação incorreta no livro (PDF)!

- Então como é implementada?
  - Com base nos resultados do exercício anterior, acompanhe a resolução no quadro



Input or output	Signal name	R-format	lw	sw	beq
Inputs (opcode)	I[6]	0	0	0	1
	I[5]	1	0	1	1
	I[4]	1	0	0	0
	I[3]	0	0	0	0
	I[2]	0	0	0	0
	I[1]	1	1	1	1
	I[0]	1	1	1	1
Outputs	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

Name (Bit position)	Fields					
	31:25	24:20	19:15	14:12	11:7	6:0
(a) R-type	funct7	rs2	rs1	funct3	rd	opcode
(b) I-type	immediate[11:0]		rs1	funct3	rd	opcode
(c) S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
(d) B-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode

Name (Field size)	Field						Comments
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
B-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format

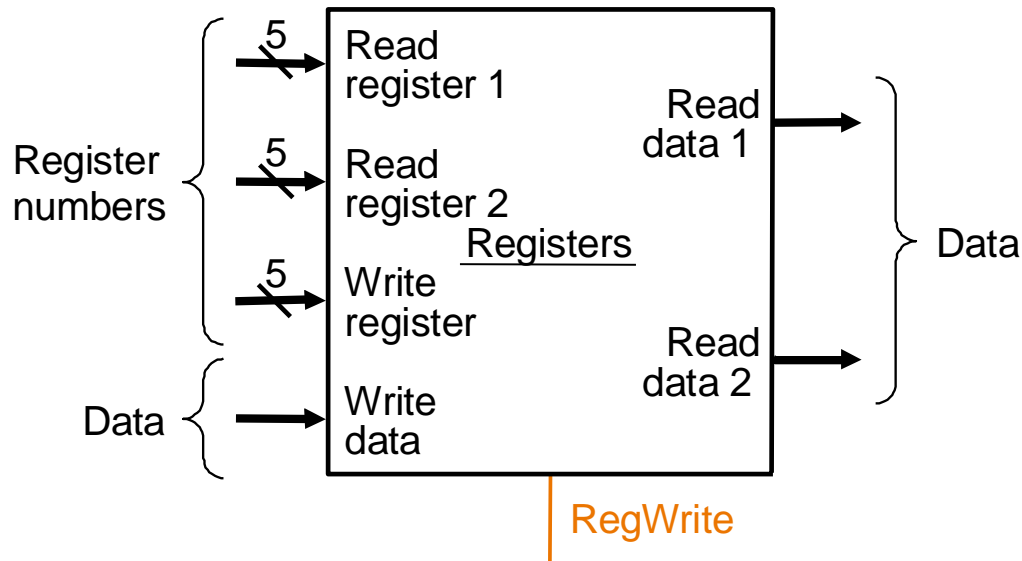
R-type Instructions	funct7	rs2	rs1	funct3	rd	opcode	Example
add (add)	0000000	00011	00010	000	00001	0110011	add x1, x2, x3
sub (sub)	0100000	00011	00010	000	00001	0110011	sub x1, x2, x3
I-type Instructions	immediate		rs1	funct3	rd	opcode	Example
addi (add immediate)	001111101000		00010	000	00001	0010011	addi x1, x2, 1000
lw (load word)	001111101000		00010	010	00001	0000011	lw x1, 1000 (x2)
S-type Instructions	immed-iate	rs2	rs1	funct3	immed-iate	opcode	Example
sw (store word)	0011111	00001	00010	010	01000	0100011	sw x1, 1000(x2)

Format	Instruction	Opcode	Funct3	Funct6/7
R-type	add	0110011	000	0000000
	sub	0110011	000	0100000
	sll	0110011	001	0000000
	xor	0110011	100	0000000
	srl	0110011	101	0000000
	sra	0110011	101	0000000
	or	0110011	110	0000000
	and	0110011	111	0000000
	lr.d	0110011	011	0001000
	sc.d	0110011	011	0001100
I-type	lb	0000011	000	n.a.
	lh	0000011	001	n.a.
	lw	0000011	010	n.a.
	lbu	0000011	100	n.a.
	lhu	0000011	101	n.a.
	addi	0010011	000	n.a.
	slli	0010011	001	0000000
	xori	0010011	100	n.a.
	srli	0010011	101	0000000
	srai	0010011	101	0100000
	ori	0010011	110	n.a.
	andi	0010011	111	n.a.
	jalr	1100111	000	n.a.
S-type	sb	0100011	000	n.a.
	sh	0100011	001	n.a.
	sw	0100011	010	n.a.
SB-type	beq	1100111	000	n.a.
	bne	1100111	001	n.a.
	blt	1100111	100	n.a.
	bge	1100111	101	n.a.
	bltu	1100111	110	n.a.
	bgeu	1100111	111	n.a.
U-type	lui	0110111	n.a.	n.a.
UJ-type	jal	1101111	n.a.	n.a.

Fig. 2.18

Voltando ao ciclo  
busca -decodificação -  
execução...

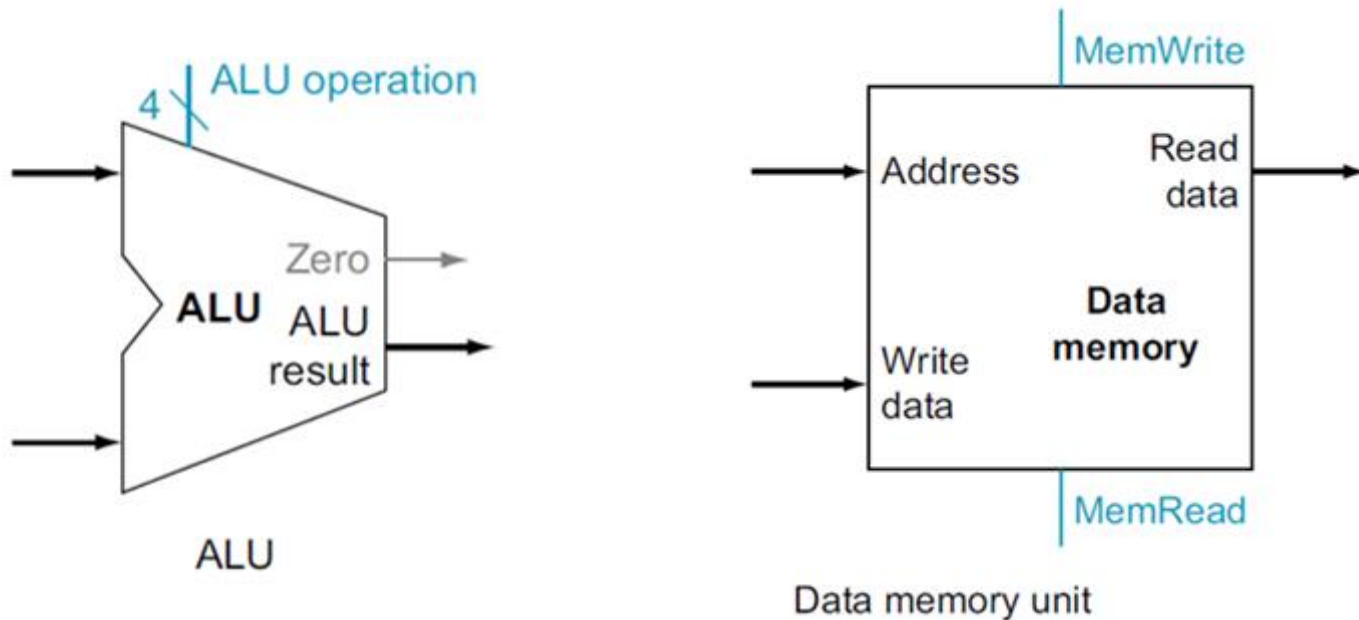
- É dividido em vários passos:
  - Busca a próxima instrução da memória;
  - Atualiza o contador de programa (Registrador) para que ele aponte para a próxima instrução;
  - Determina o tipo da instrução (Decodifica a instrução);
  - **Acessa os dados contidos em registradores**



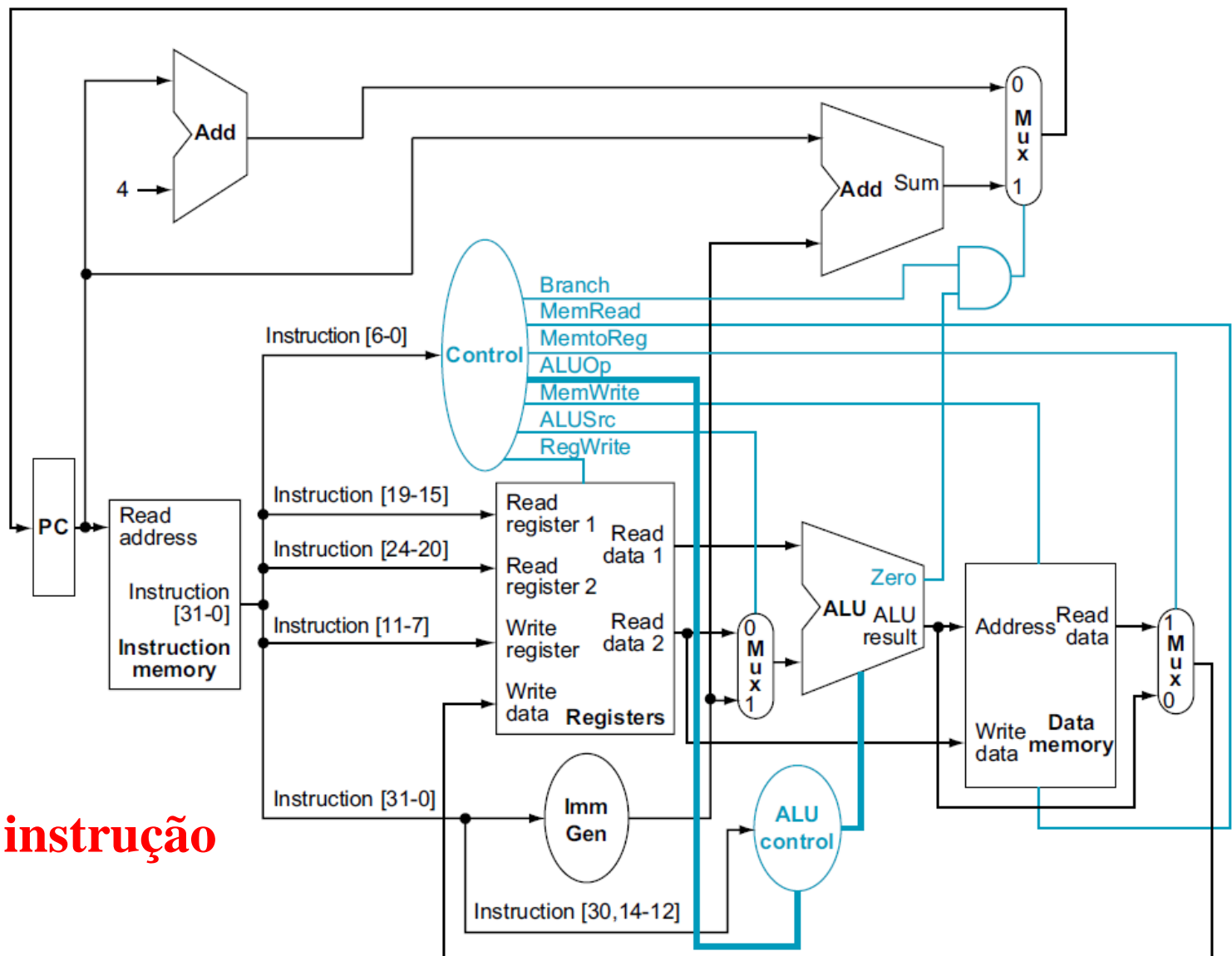
Instruções	Tipo
add rd, <b>rs1</b> , <b>rs2</b>	R
sub rd, <b>rs1</b> , <b>rs2</b>	R
lw rd, imm( <b>rs1</b> )	I
sw <b>rs2</b> , imm( <b>rs1</b> )	S
or rd, <b>rs1</b> , <b>rs2</b>	R
and rd, <b>rs1</b> , <b>rs2</b>	R
beq <b>rs1</b> , <b>rs2</b> , L	B

a. Registers

- É dividido em vários passos:
  - Busca a próxima instrução da memória;
  - Atualiza o contador de programa (Registrador) para que ele aponte para a próxima instrução;
  - Determina o tipo da instrução (Decodifica a instrução);
  - Acessa os dados contidos em registradores
  - **Executa a instrução**
    - **faz a operação na ULA;**
    - **acessa memória de dados, se necessário;**



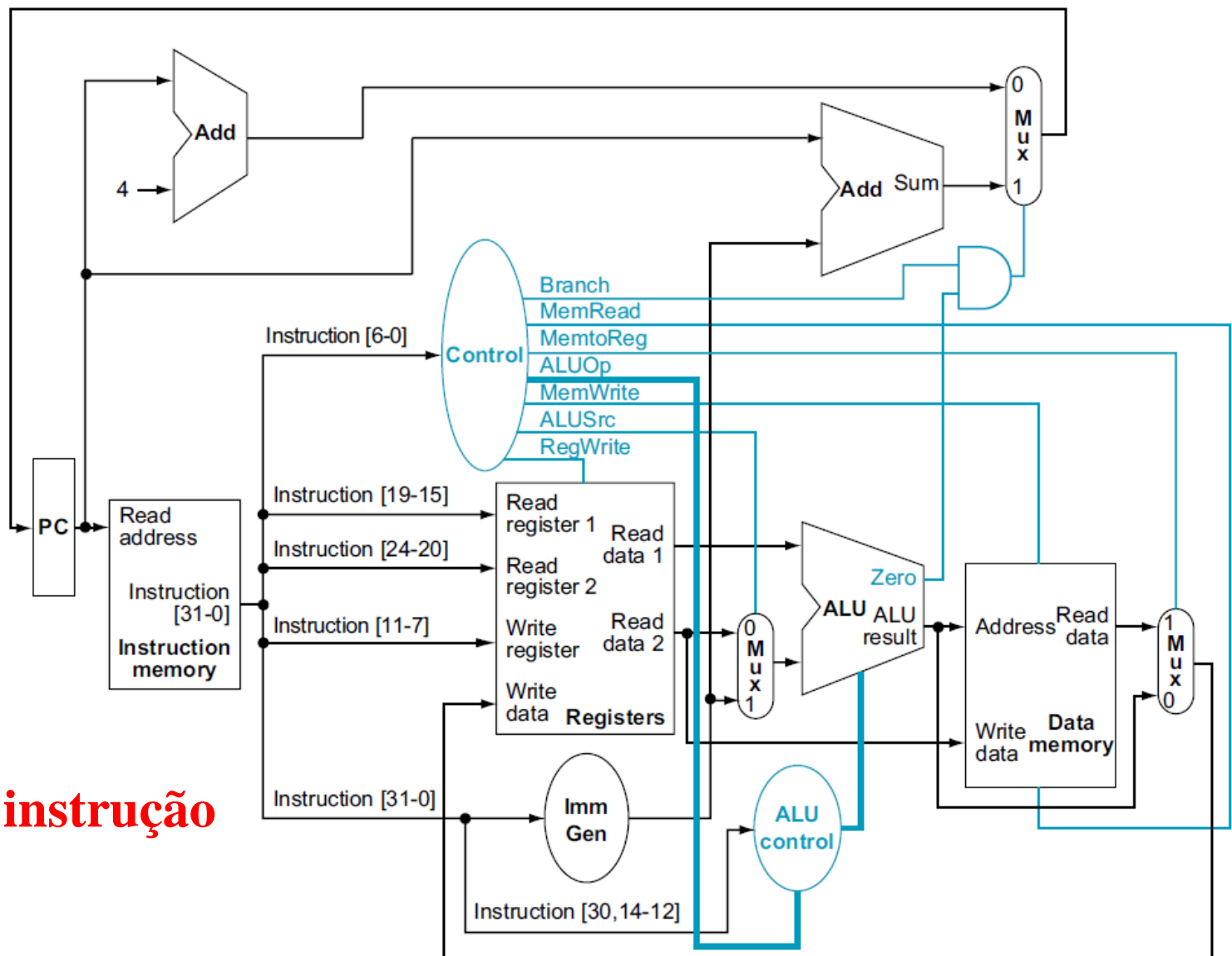




## - Executa a instrução

Se *add*, *sub*, *and* ou *or*:

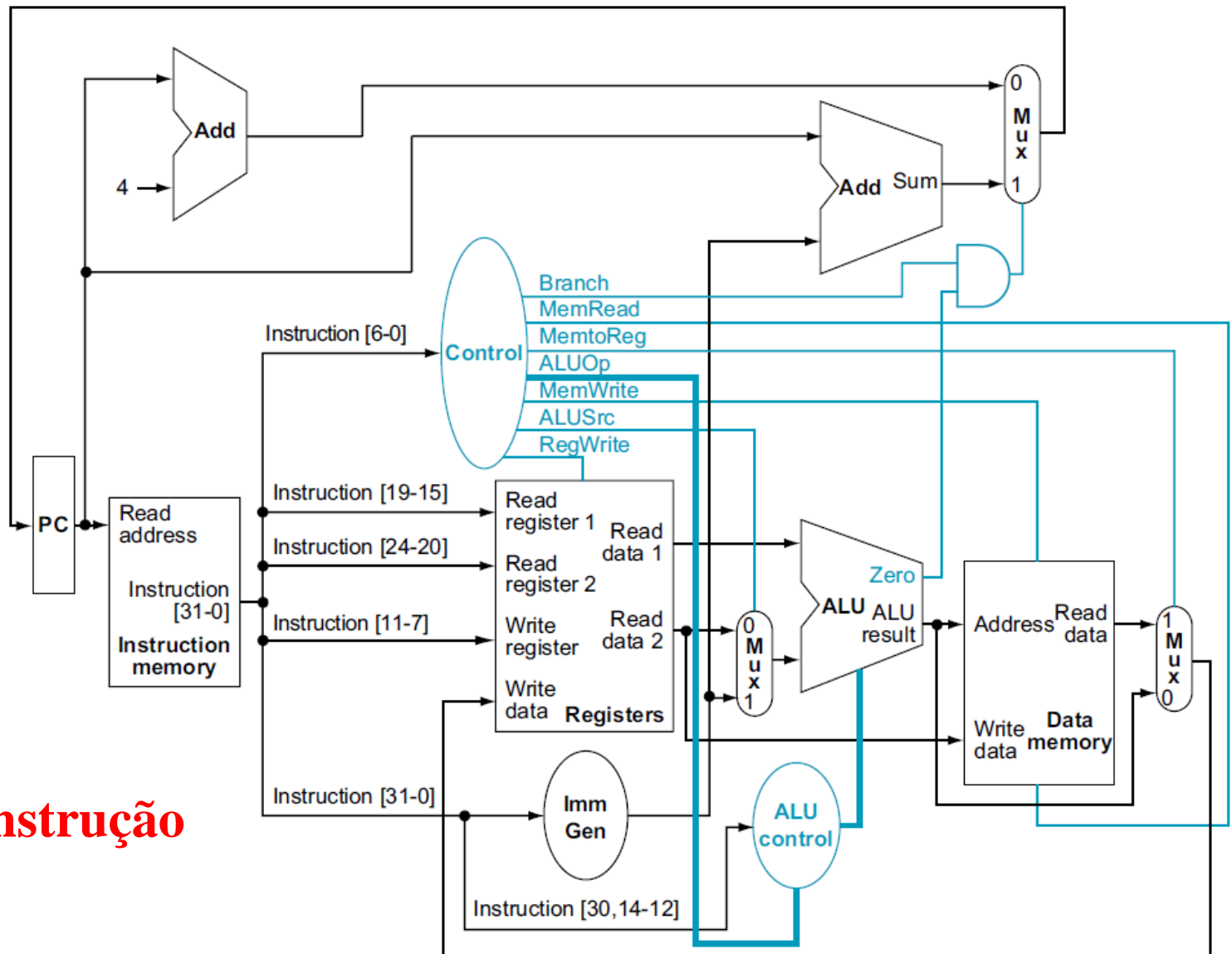
- **Passo 3:** Realiza uma operação na **ULA**, com o conteúdo dos **registradores lidos**;
- **Passo 4:** Escreve o resultado da operação em um **registrador** (rd).



- Executa a instrução

Se *lw*:

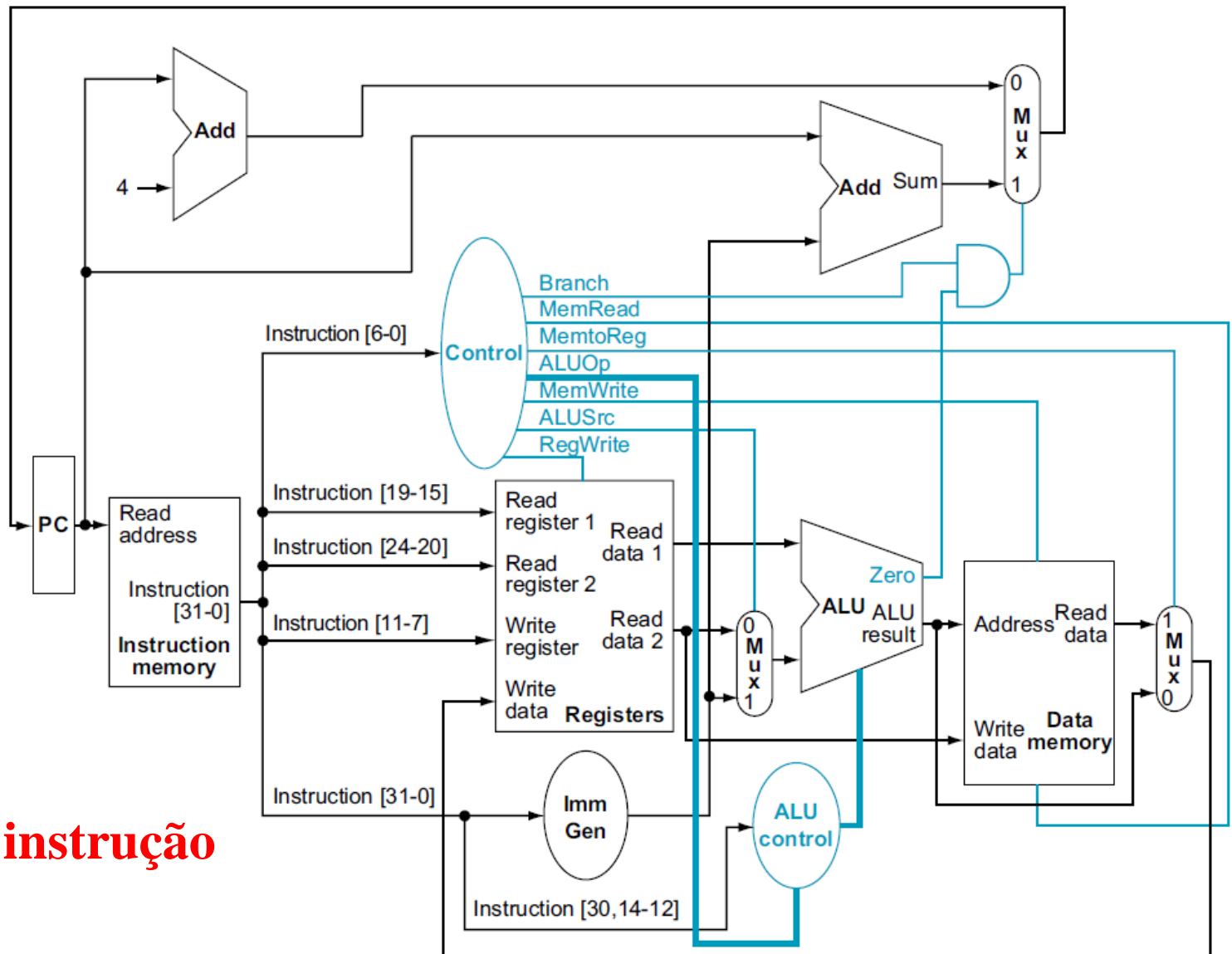
- **Passo 3:** Realiza a soma do **reg. de base** com o *imm* (calcula **end.** a ser acessado na memória);
- **Passo 4:** Acessa a **memória** para ler o valor contido no **endereço calculado**;
- **Passo 5:** Escreve o valor lido da memória dentro de um **registrador** (rd).



- Executa a instrução

Se *sw*:

- **Passo 3:** Realiza a soma do **reg. de base** com o *imm* (calcula **end.** a ser acessado na memória);
- **Passo 4:** Acessa a **memória** para escrever o valor do **registrador** `rs2` no **endereço** calculado;



## - Executa a instrução

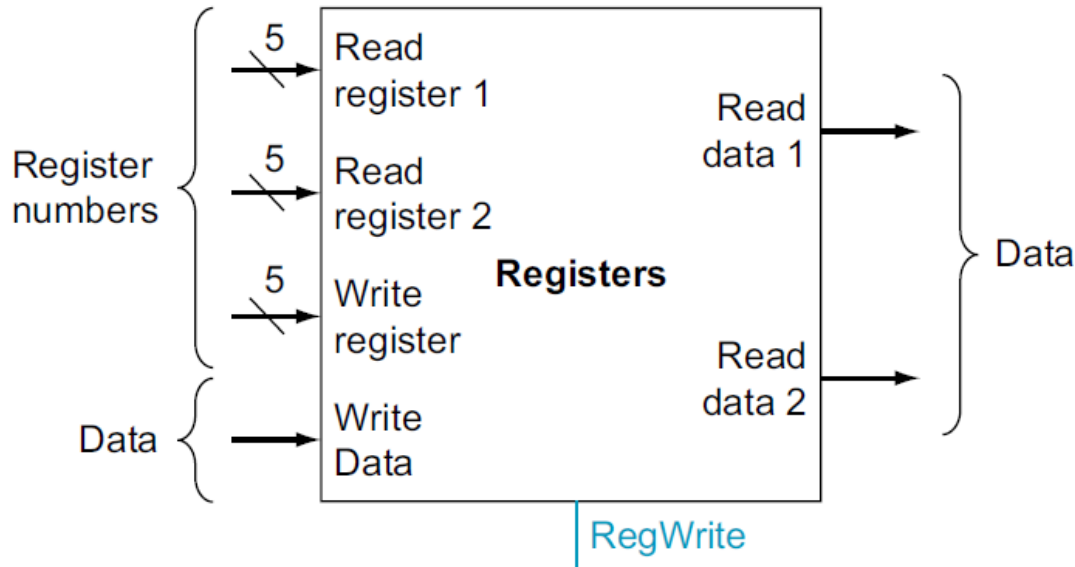
Se *beq*:

- **Passo 3:** Subtrai do valor de `rs1` o valor de `rs2` na **ULA**
- **Passo 4:** Se o bit *zero* for *true*, **atualiza PC** com  $PC + imm$

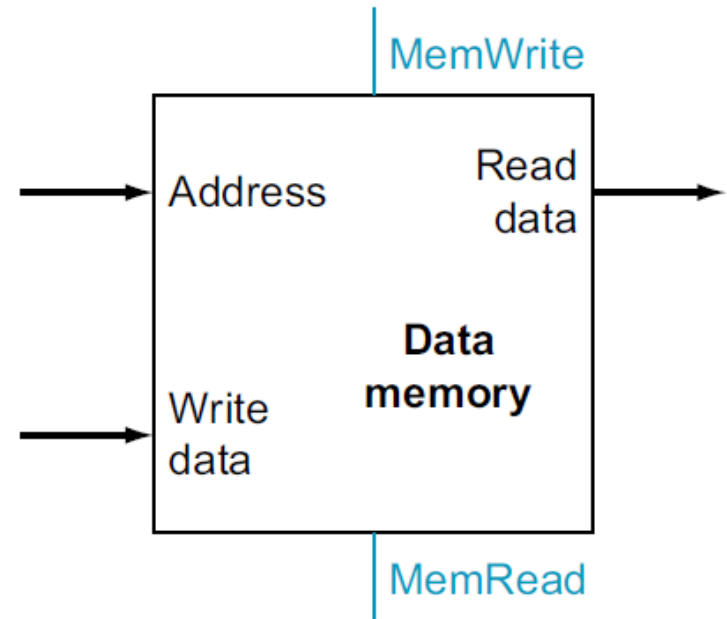
# *Execução de Instruções*

- É dividido em vários passos:
  - Busca a próxima instrução da memória;
  - Atualiza o contador de programa (Registrador) para que ele aponte para a próxima instrução;
  - Determina o tipo da instrução (Decodifica a instrução);
  - Acessa os dados contidos em registradores
  - Executa a instrução
    - faz a operação na ULA;
    - acessa memória de dados, se necessário;
  - **Armazena o resultado em locais apropriados**
  - Volta ao passo 1 (inicia a execução da próxima instrução)

# *Via de Dados*



a. Registers



a. Data memory unit

# *Execução de Instruções*

- É dividido em vários passos:
  - Busca a próxima instrução da memória;
  - Atualiza o contador de programa (Registrador) para que ele aponte para a próxima instrução;
  - Determina o tipo da instrução (Decodifica a instrução);
  - Acessa os dados contidos em registradores
  - Executa a instrução
    - faz a operação na ULA;
    - acessa memória de dados, se necessário;
  - Armazena o resultado em locais apropriados
  - **Volta ao passo 1 (inicia a execução da próxima instrução)**

# *Execução de Instruções - SPEC 2006*

Classes de instrução RISC-V: correspondência entre frequência de uso das instruções em linguagem de alto nível e de instruções RISC-V executadas nos benchmarks do SPEC CPU2006 de ponto flutuante e inteiro.

Instruction class	RISC-V examples	HLL correspondence	Frequency	
			Integer	Fl. Pt.
Arithmetic	add, sub, addi	Operations in assignment statements	16%	48%
Data transfer	lw, sw, lh, sh, lb, sb, lui	References to data structures in memory	35%	36%
Logical	and, or, xor, sll, srl, sra	Operations in assignment statements	12%	4%
Branch	beq, bne, blt, bge, bltu, bgeu	<i>If</i> statements; loops	34%	8%
Jump	jal, jalr	Procedure calls & returns; <i>switch</i> statements	2%	0%



# Referências

- PATTERSON, David A; HENNESSY, John L; Computer Organization and Design – The hardware/software interface RISC-V edition; Elsevier – Morgan Kaufmann/ Amsterdam.
- PATTERSON, David; Waterman, Andrew; The RISC-V reader: an open architecture atlas; First edition. Berkeley, California: Strawberry Canyon LLC, 2017.