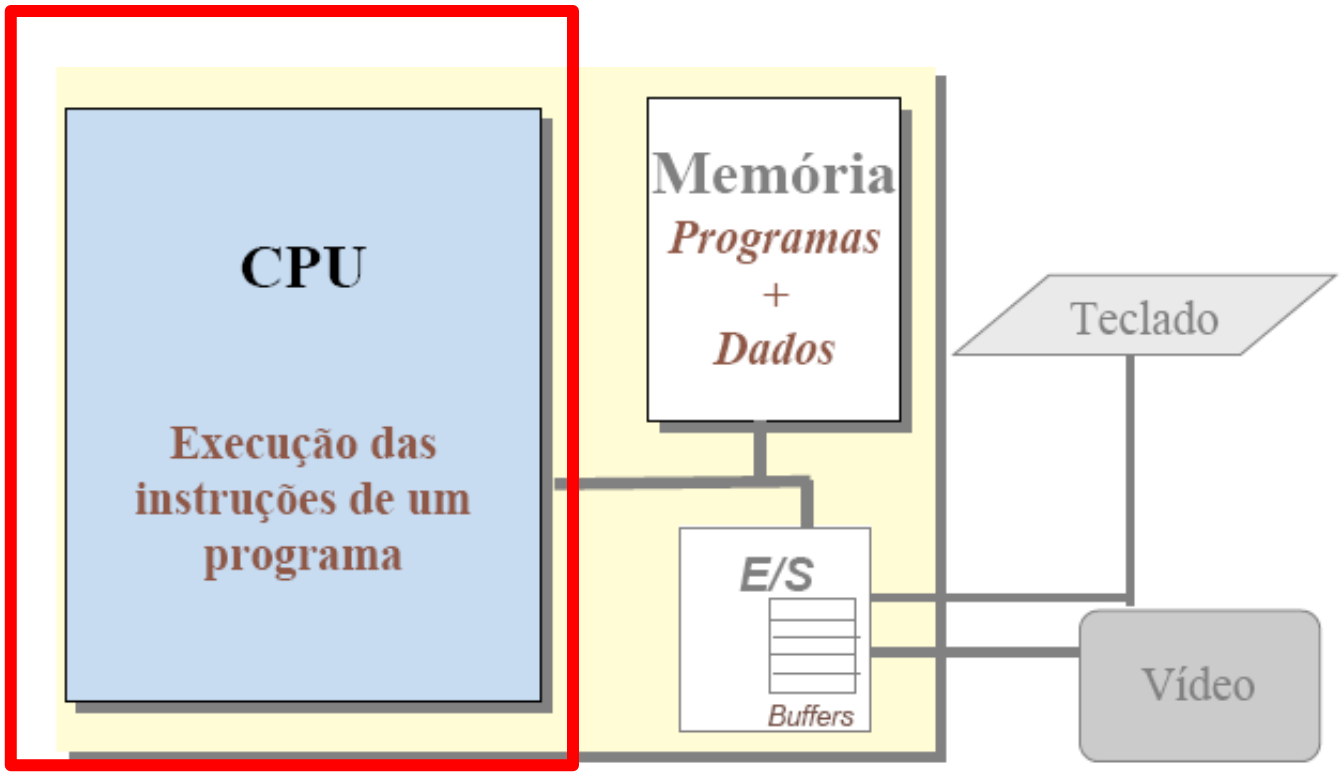
ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Arquitetura RISC-V

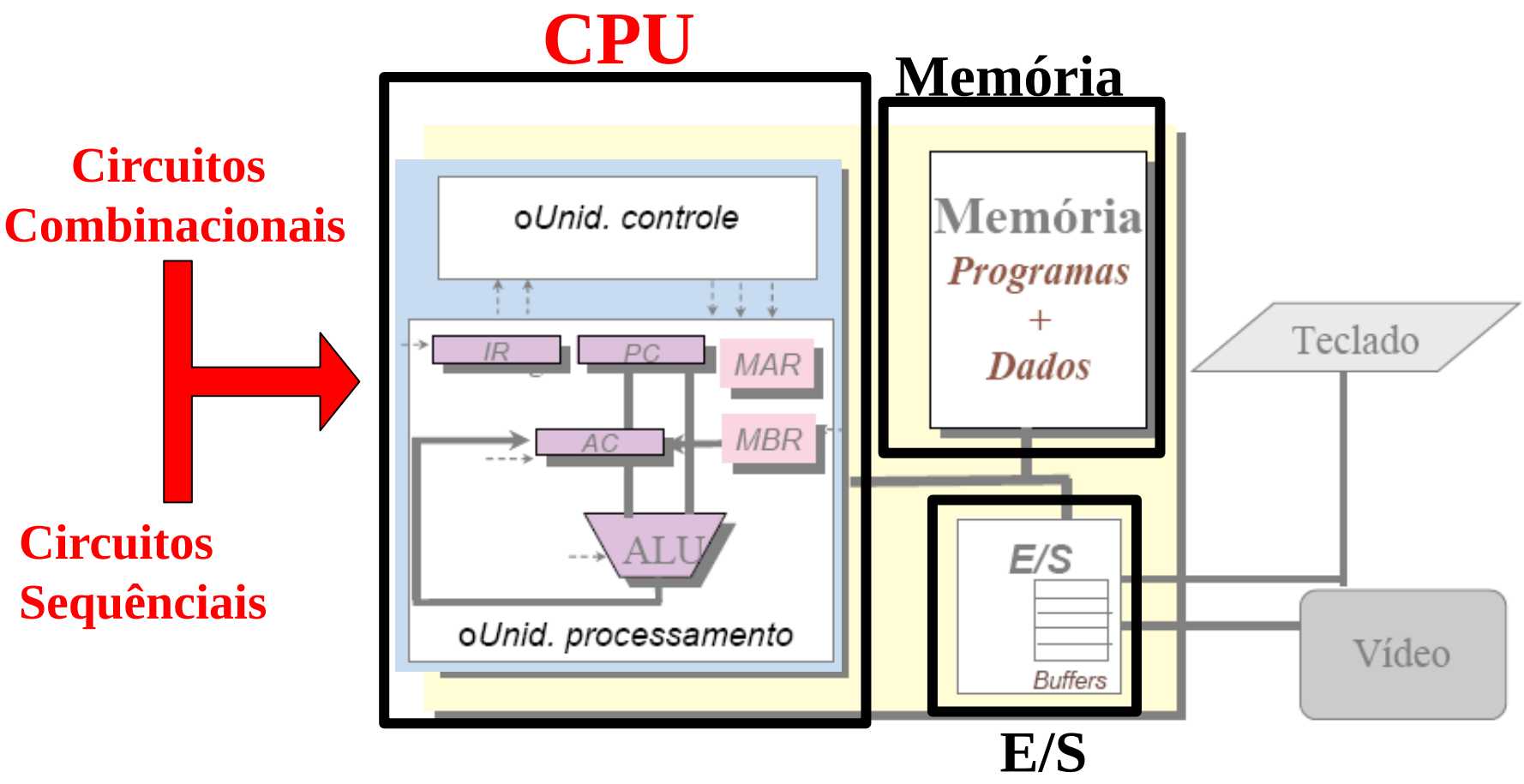
Profª. Fabiana F F Peres

Apoio: Camile Bordini

Composição Básica de um Computador



Composição Básica de um Computador



Composição Básica de um Computador

- Blocos lógicos são categorizados de dois tipos:

1. **Combinacionais**
   - Blocos sem memória
   - A saída depende apenas da entrada atual
2. **Sequenciais**
   - Blocos com memória
   - Saídas podem depender tanto das entradas quanto do valor armazenado na memória, que é chamado de estado do bloco lógico

Circuito Combinacional vs Circuito Sequêncial

CIRCUITOS COMBINACIONAIS

1. Codificador/decodificador
2. Comparadores
3. Geradores de paridade
4. Multiplexador/Demux
5. PLAs
6. Memórias ROM
7. Somador / Subtrator
8. ULA
9. Multiplicadores / Divisores

CIRCUITOS SEQUENCIAIS

1. Latches
2. Flip-Flop
3. Registradores
4. Contadores
5. Máquina de Estados
6. Geradores de clock
7. Memória RAM
8. Sequenciadores

Circuito Combinacional vs Circuito Sequêncial

CIRCUITOS COMBINACIONAIS

1. Codificador/decodificador
2. Comparadores
3. Geradores de paridade
4. Multiplexador/Demux
5. PLAs
6. Memórias ROM
7. Somador / Subtrator
8. ULA
9. Multiplicadores / Divisores

CIRCUITOS SEQUENCIAIS

1. Latches
2. Flip-Flop
3. Registradores
4. Contadores
5. Máquina de Estados
6. Geradores de clock
7. Memória RAM
8. Sequenciadores

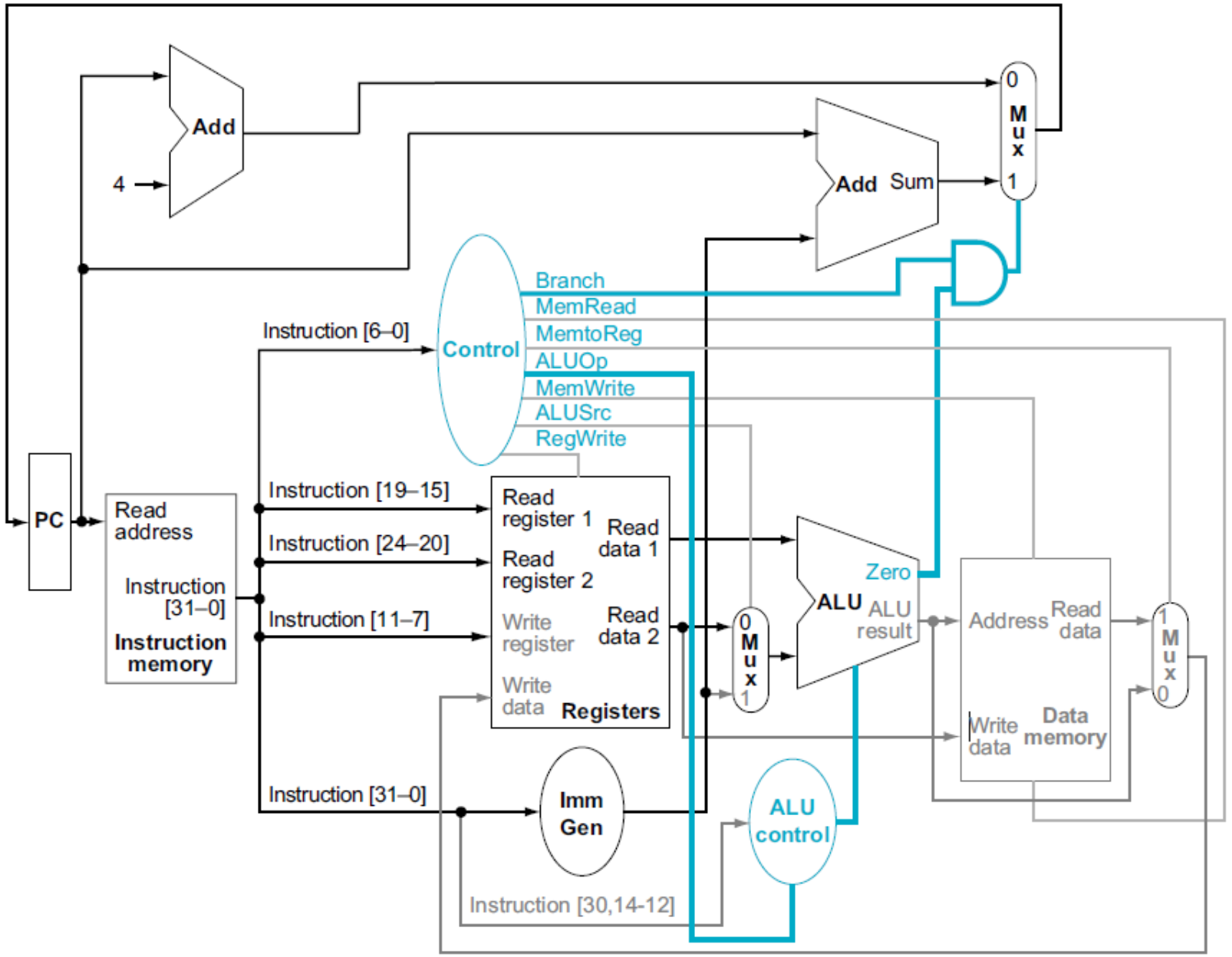
Arquitetura

- Circuitos combinacionais e sequenciais estão presentes em diversas **arquiteturas (=ISA)**
  - **RISC-V**
  - **MIPS**
  - **Intel**
  - **...**
- Nesta disciplina, focaremos nos circuitos para a arquitetura **RISC-V**

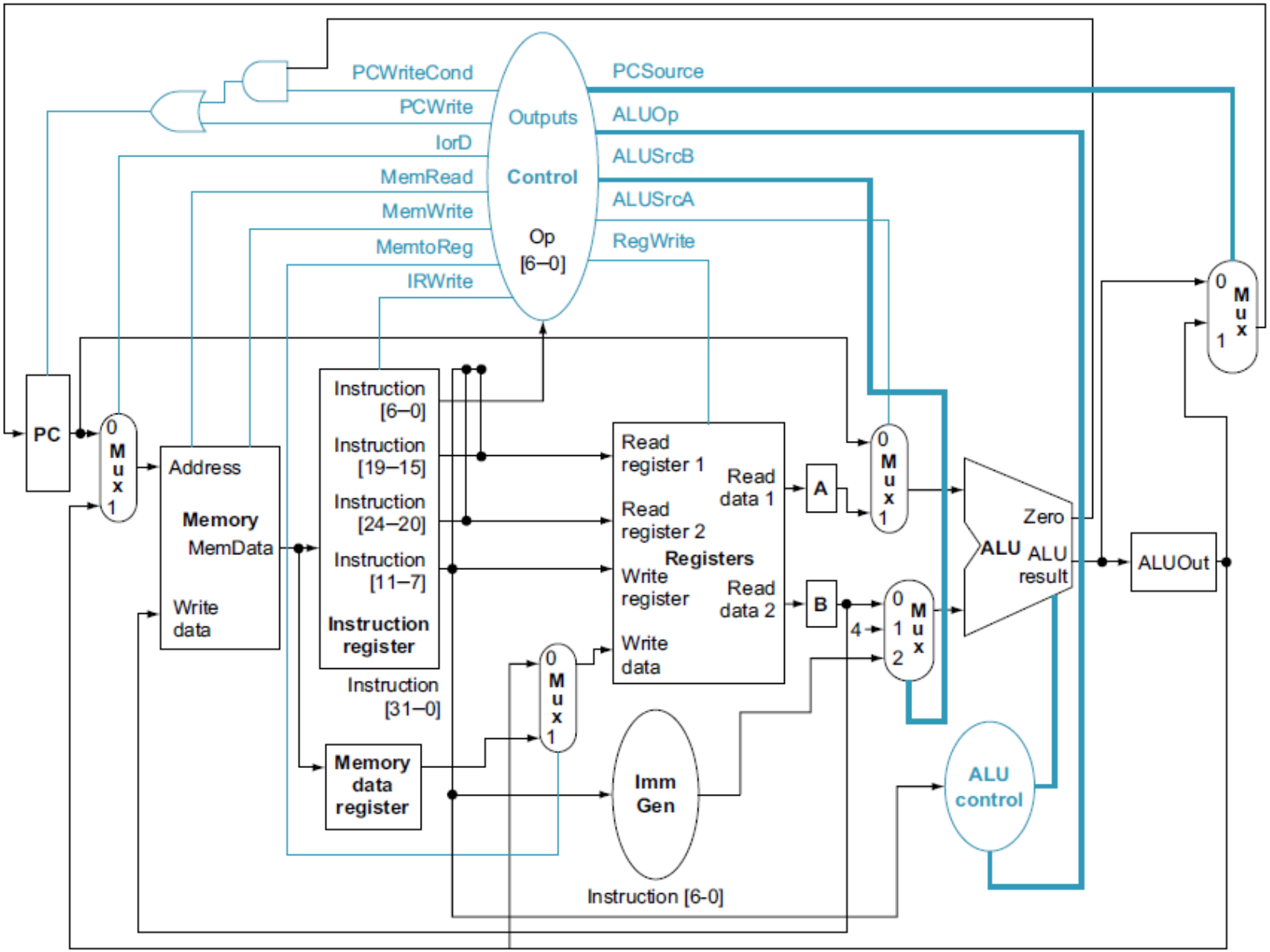
Organização

- Além disso, a arquitetura **RISC-V** pode ter diferentes **implementações**:
  1. **Monociclo**
  2. **Multiciclo**
  3. **Pipeline**

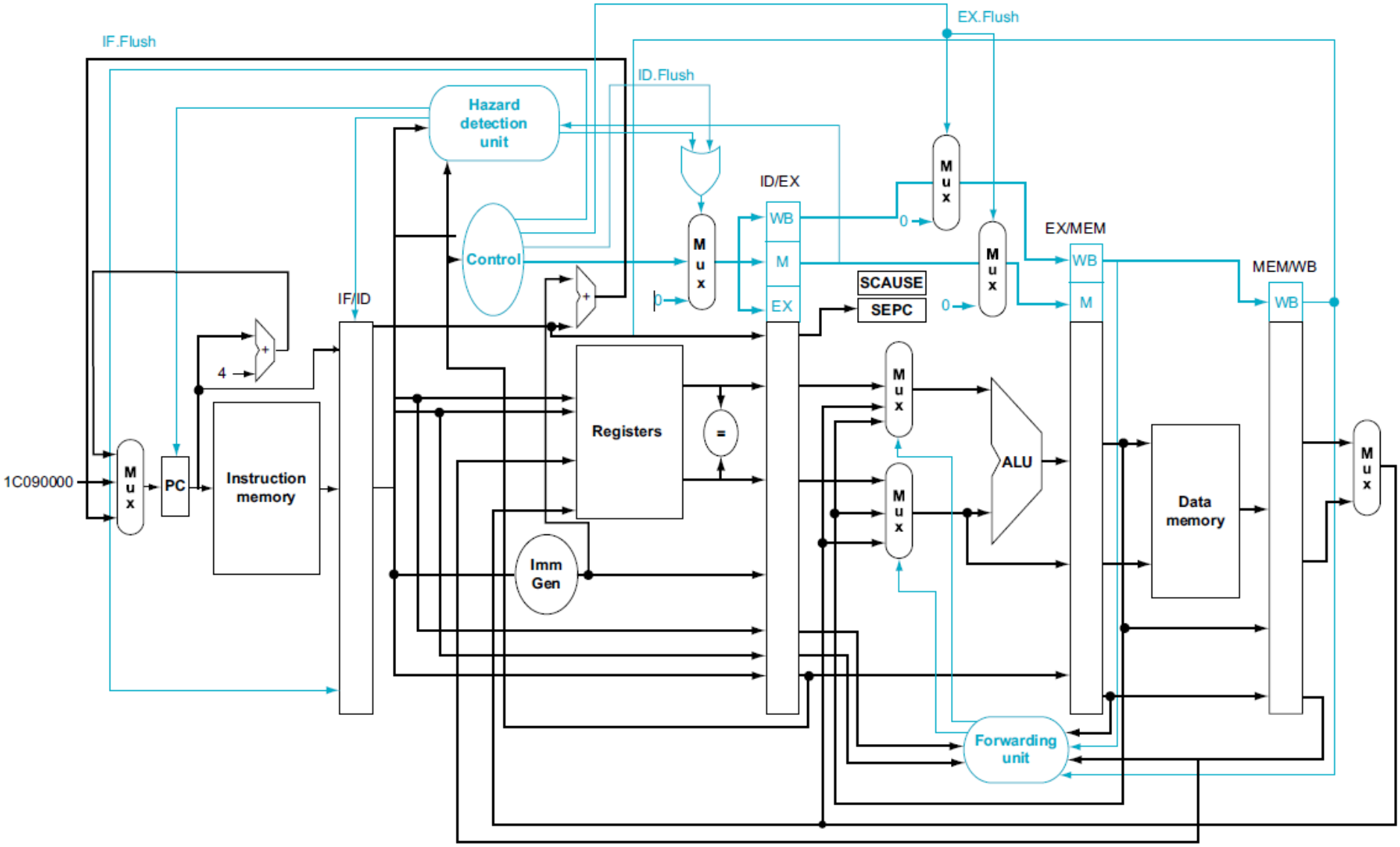
Monociclo


Add
4
Add
Sum
0
M
u
x
1
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [6–0]
PC
Read
address
Instruction
[31–0]
Instruction
memory
Instruction [19–15]
Instruction [24–20]
Instruction [11–7]
Instruction [31–0]
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
Registers
Zero
ALU
ALU
result
Address
Read
data
Write
data
Data
memory
Imm
Gen
ALU
control
Instruction [30,14-12]

Multiciclo


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[6–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
PC
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction
[6–0]
Instruction
[19–15]
Instruction
[24–20]
Instruction
[11–7]
Instruction
register
Instruction
[31–0]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
ALU
Zero
ALU
result
ALUOut
ALU
control
Imm
Gen
Instruction [6-0]

Pipeline

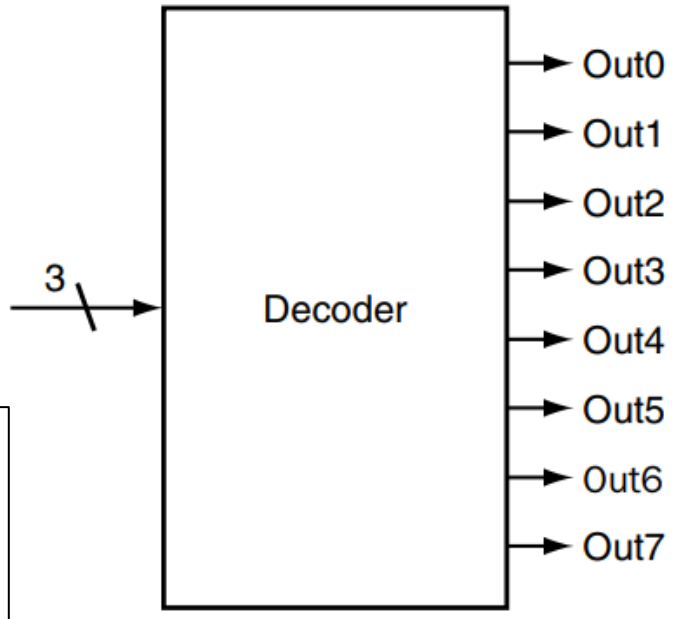

IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
Control
IF/ID
ID/EX
EX/MEM
MEM/WB
WB
M
EX
SCAUSE
SEPC
Mux
0
4
1C090000
PC
Instruction memory
Registers
=
Imm Gen
ALU
Data memory
Forwarding unit

Decodificador

Decodificador

- Bloco lógico utilizado para a construção de componentes maiores
- Tipo comum:
  - **Entrada de n bits** e **2^n saídas**
  - Apenas uma saída é ativada para cada combinação de entrada
  - O decodificador traduz a entrada de **n bits** em um sinal que corresponde ao **valor binário da entrada de n bits**
  - As saídas são, portanto, numeradas: **Out0, Out1,…, Out2^n –1**

- Ex: "*decodificador 3-a-8*": **3 bits de entrada** e 2^3 **saídas**



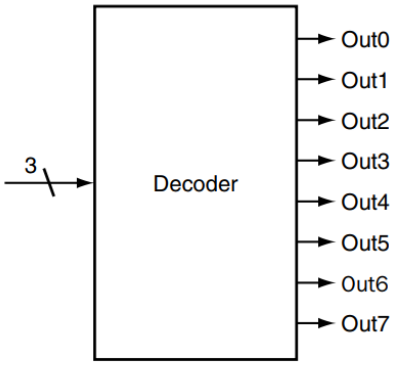
a. A 3-bit decoder

Obs: Se o valor da entrada for **i**, então **Outi** será *verdadeiro* e todas as outras saídas serão falsas

Inputs			Outputs							
12	11	10	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

b. The truth table for a 3-bit decoder

Decodificador 3x8



a. A 3-bit decoder

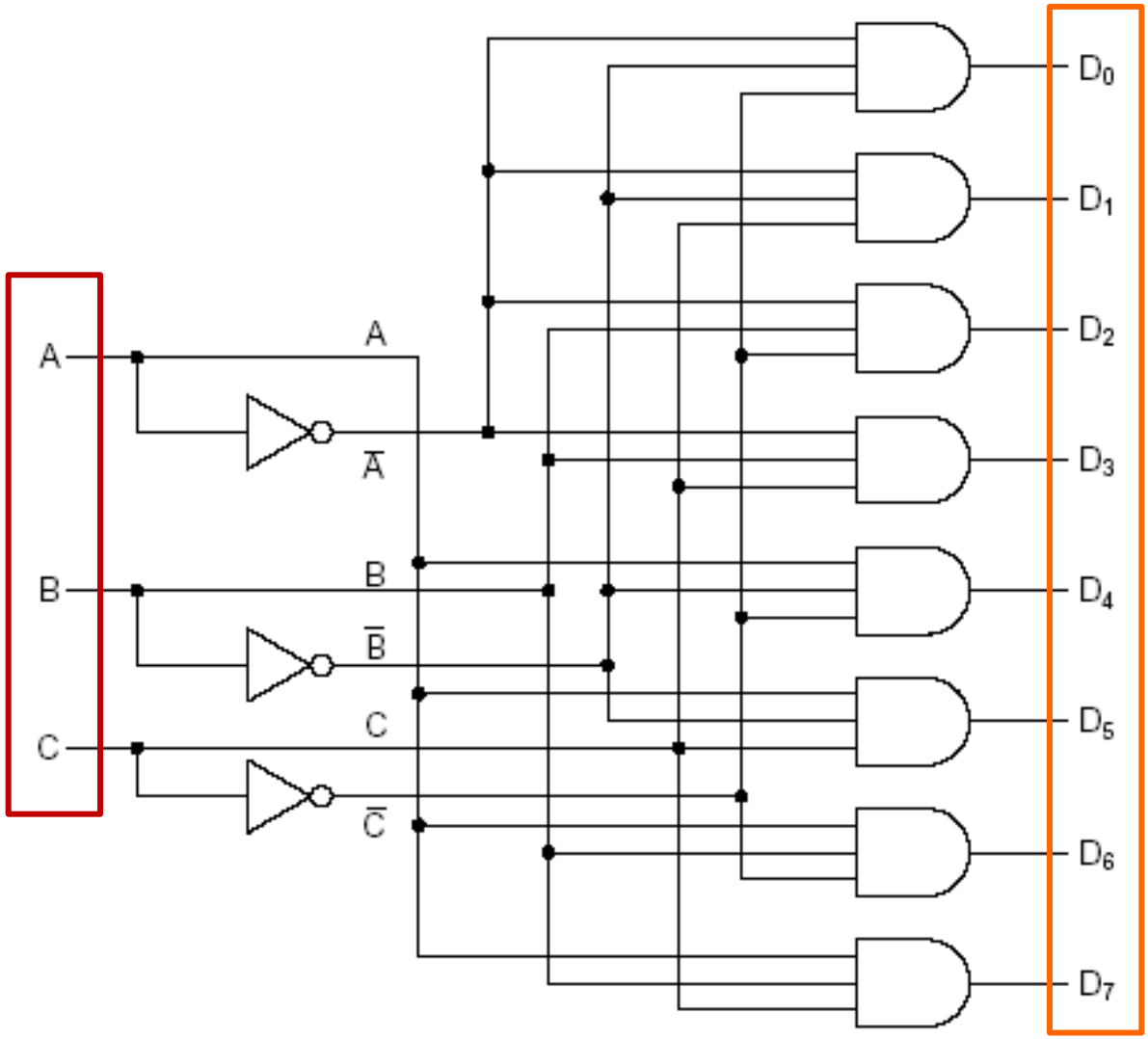


Figure 3-13. A 3-to-8 decoder circuit.

Decodificador

- Quantas saídas para um decodificador com **5 entradas**?
- **2^5 = 32 saídas**

- Alguma semelhança com a arquitetura **RISC-V**?

Decodificador

- Quantas saídas para um decodificador com **5 entradas**?
- **2^5 = 32 saídas**

- Alguma semelhança com a arquitetura **RISC-V**?
  - **RISC-V** possui um banco de **32 registradores**!
  - Portanto, um decodificador vai servir para endereçar cada um dos 32 registradores

Multiplexador

Multiplexador (Mux)

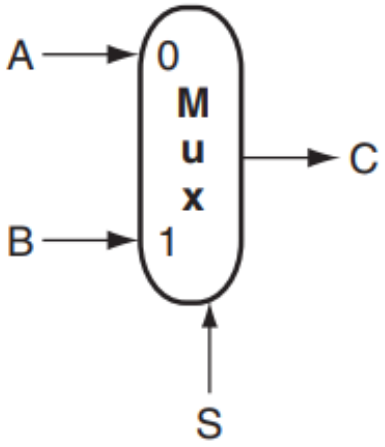
- Finalidade: selecionar uma de suas entradas e conectá-la à sua única saída

- Utiliza para isso um **seletor**, também chamado **valor de controle**: O sinal de controle que é usado para selecionar um dos valores de entradas de um multiplexador como sendo sua saída.

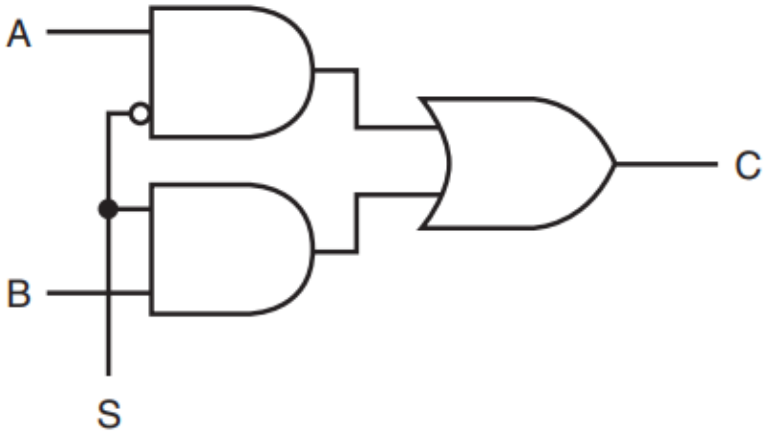
- Pode ter 2 ou mais **entradas**, mas apenas 1 **saída**

Multiplexador (Mux)

- Ex: multiplexador de 3 entradas: **2 valores de dados** (A e B) e um **seletor** (S)



S	C
0	A
1	B



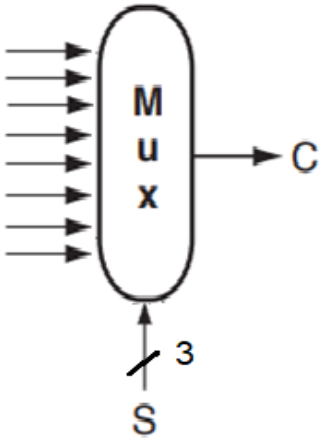
$$C = (A \cdot \overline{S}) + (B \cdot S)$$

Obs: com **n entradas de dados**: necessário $\lceil \log_2 n \rceil$ **seletores**

Multiplexador (Mux)

- Um **multiplexador** é basicamente:
  1. Um **decodificador** que gera **n sinais**, cada um indicando um **valor de entrada diferente**
  2. Um arranjo de **n portas AND**, cada uma combinando **uma das entradas** com um **sinal do decodificador**
  3. Uma **única porta OR** maior que incorpora as saídas das portas **AND**

Multiplexador com 8 entradas



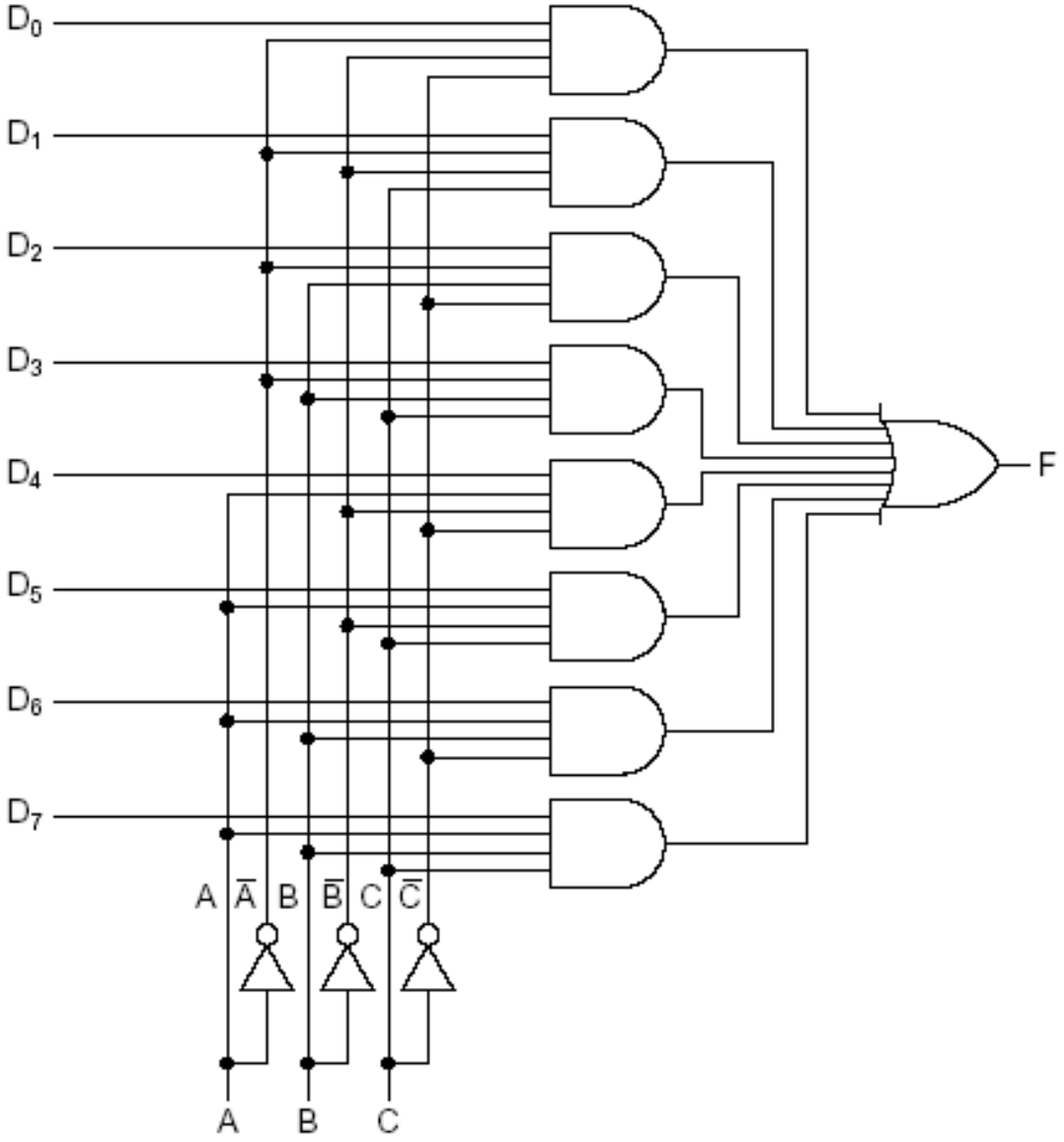
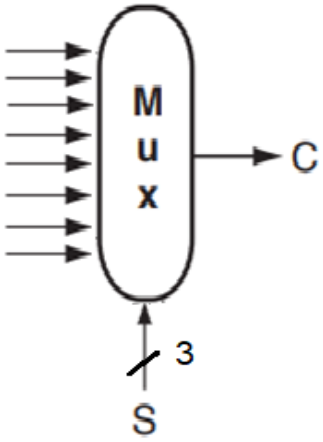


Figure 3-11. An eight-input multiplexer circuit.

Multiplexador com 8 entradas



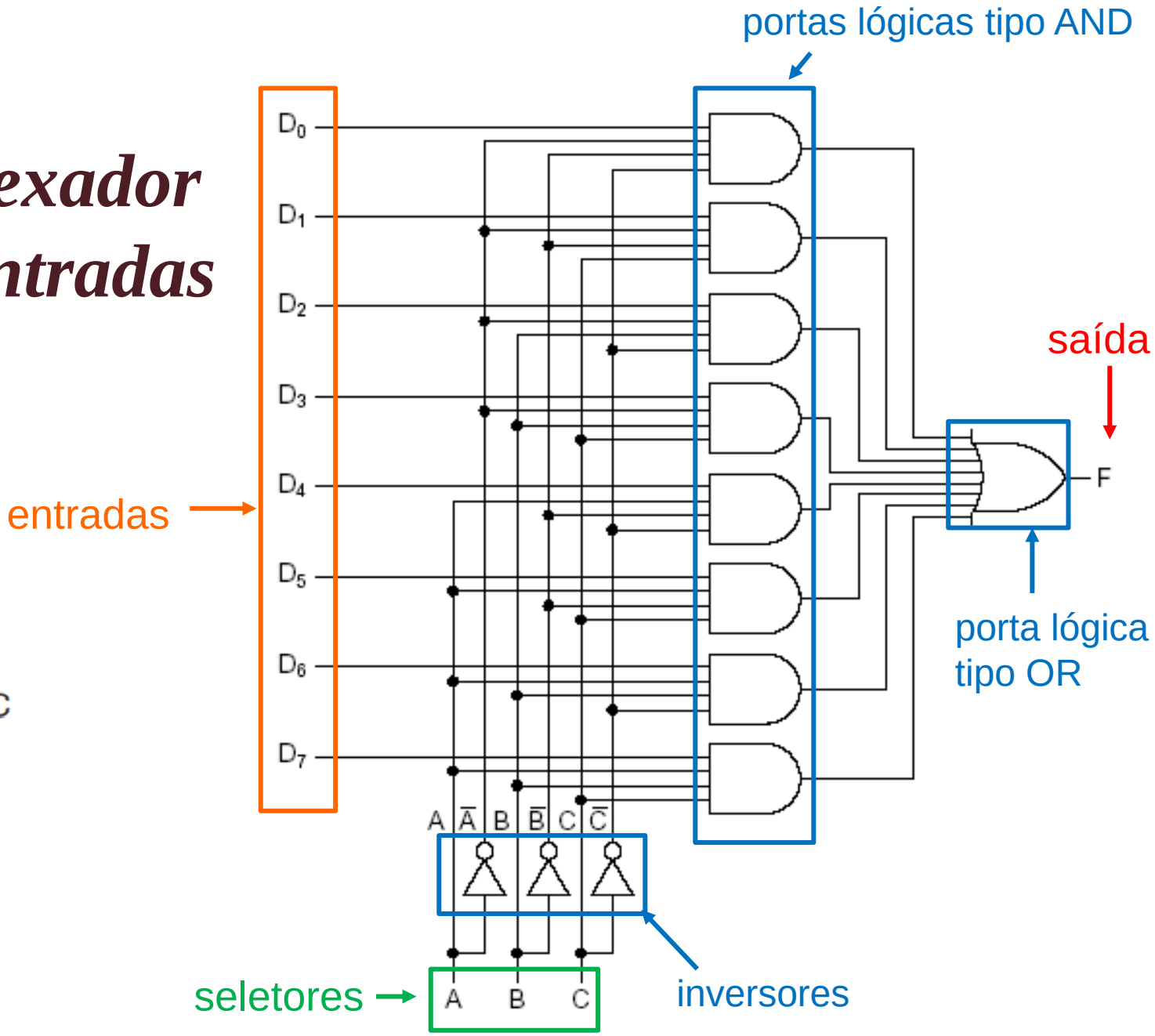


Figure 3-11. An eight-input multiplexer circuit.

PLA

Lógica de 2 níveis

- Representação de 2 níveis:
  - Forma canônica de uma **função lógica**: cada **entrada** é uma *variável afirmada* ou *negada* (*complemento*)
  - Existe apenas 2 níveis de portas lógicas: uma sendo **AND** e outra **OR**, com um possível **INVERSOR** na saída final
  - Pode ser através da ***soma de produtos*** ou do ***produto de somas***.

Lógica de 2 níveis

- Ex1: $E = (A \cdot B \cdot \overline{C}) + (A \cdot C \cdot \overline{B}) + (B \cdot C \cdot \overline{A})$
  - Equação está na forma de **soma de produtos**
  - 2 níveis de lógica e inversores apenas em variáveis individuais

- A equação abaixo está na forma de **produtos de somas**:

$$E = \overline{(\overline{A} + \overline{B} + C) \cdot (\overline{A} + \overline{C} + B) \cdot (\overline{B} + C + A)}$$

- Não vamos usar!

Lógica de 2 níveis

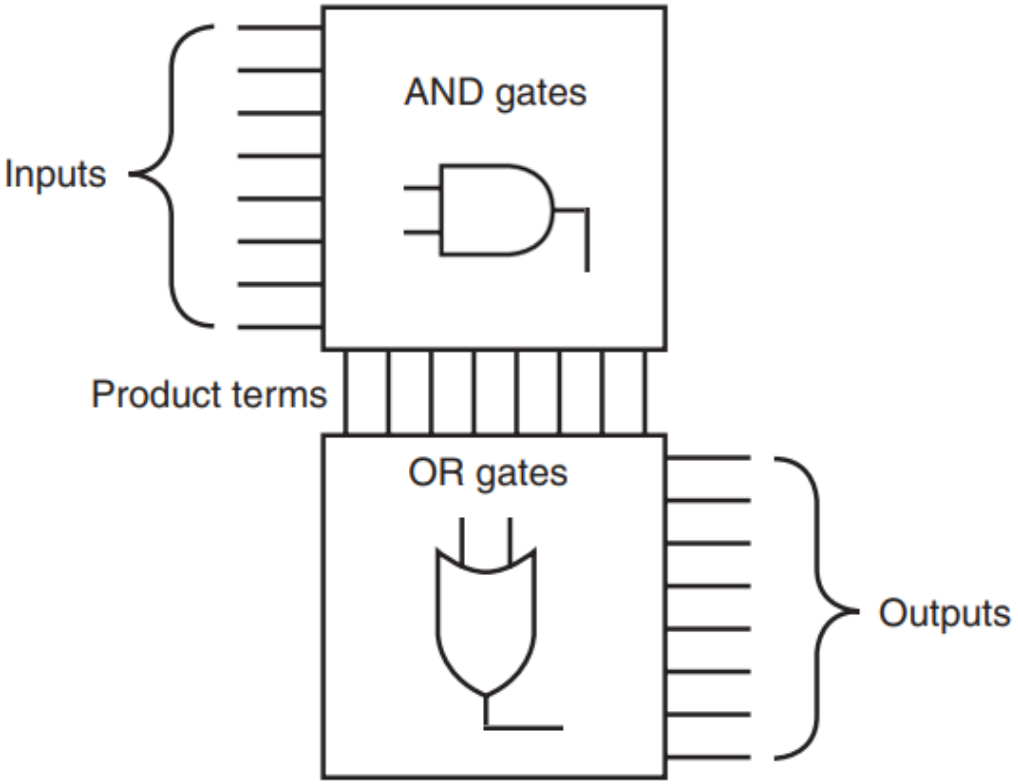
- É possível observar que qualquer função lógica pode ser representada como uma **soma de produtos** construindo tal representação a partir da tabela verdade.

A	B	C	S	
0	0	0	0	
0	0	1	1	$\Rightarrow \overline{A} \cdot \overline{B} \cdot C$
0	1	0	1	$\Rightarrow \overline{A} \cdot B \cdot \overline{C}$
0	1	1	0	
1	0	0	1	$\Rightarrow A \cdot \overline{B} \cdot \overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	$\Rightarrow A \cdot B \cdot C$

$$D = (\overline{A} \cdot \overline{B} \cdot C) + (\overline{A} \cdot B \cdot \overline{C}) + (A \cdot \overline{B} \cdot \overline{C}) + (A \cdot B \cdot C)$$

PLA

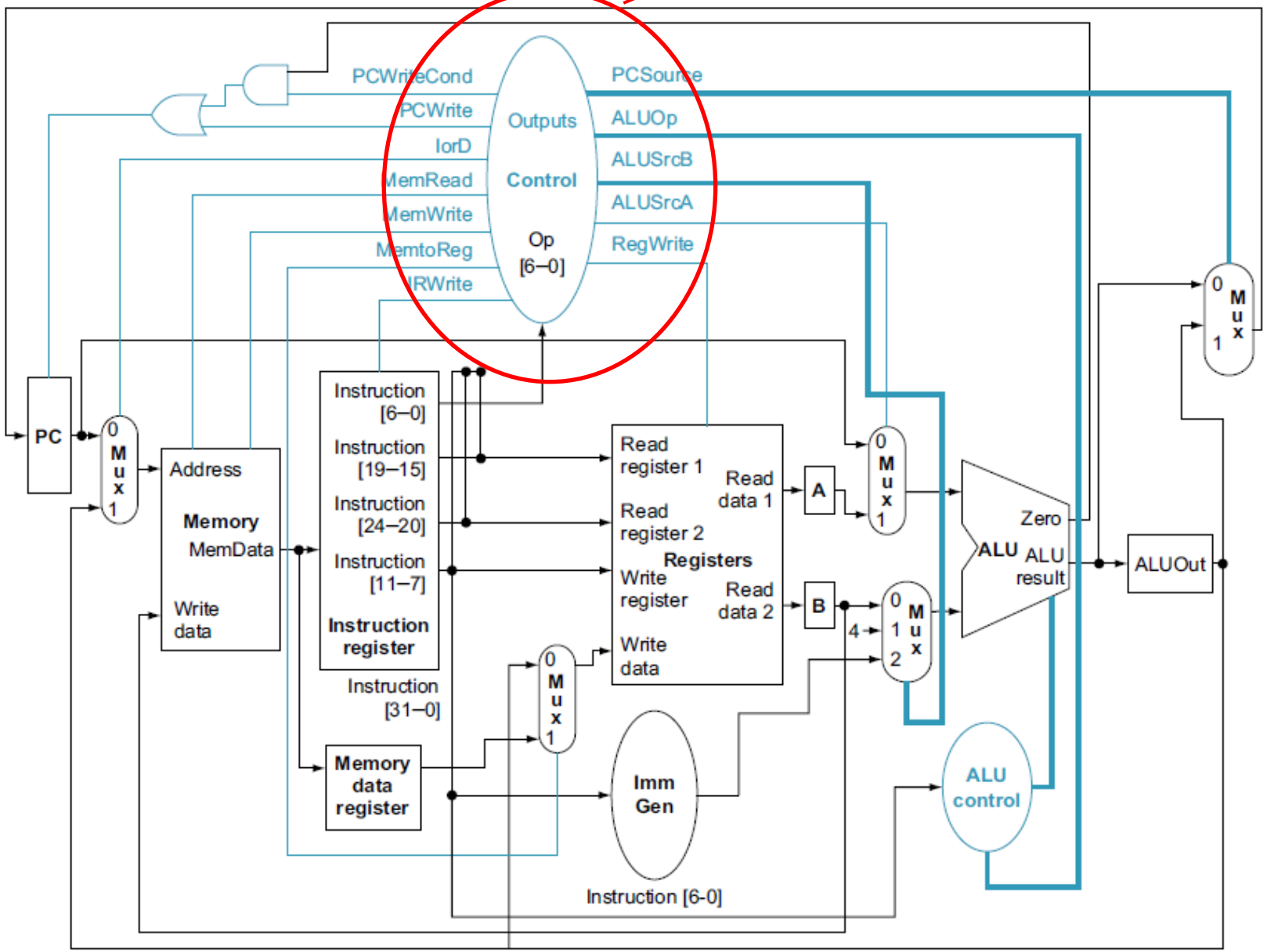
- A representação da soma dos produtos corresponde a uma implementação de uma **matriz lógica programável** (**PLA**)



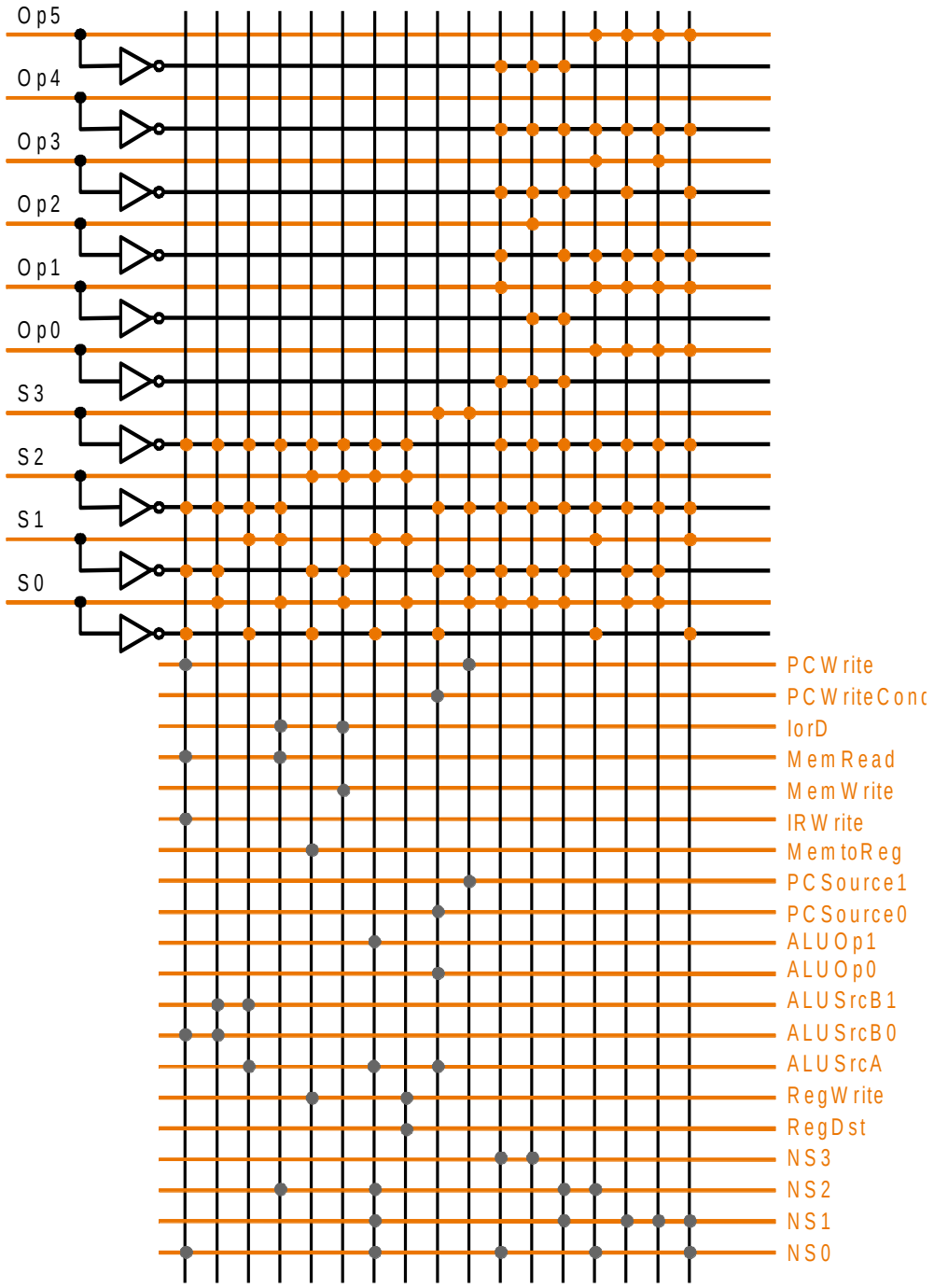
- Um arranjo de AND's seguido por um arranjo de OR's
- Uma porta AND pode ter qualquer número de entradas, e corresponde a um termo do produto

Multiciclo

Um PLA está presente na Unidade de Controle



Unidade de Controle da MIPS



Unidade Lógico Aritmética

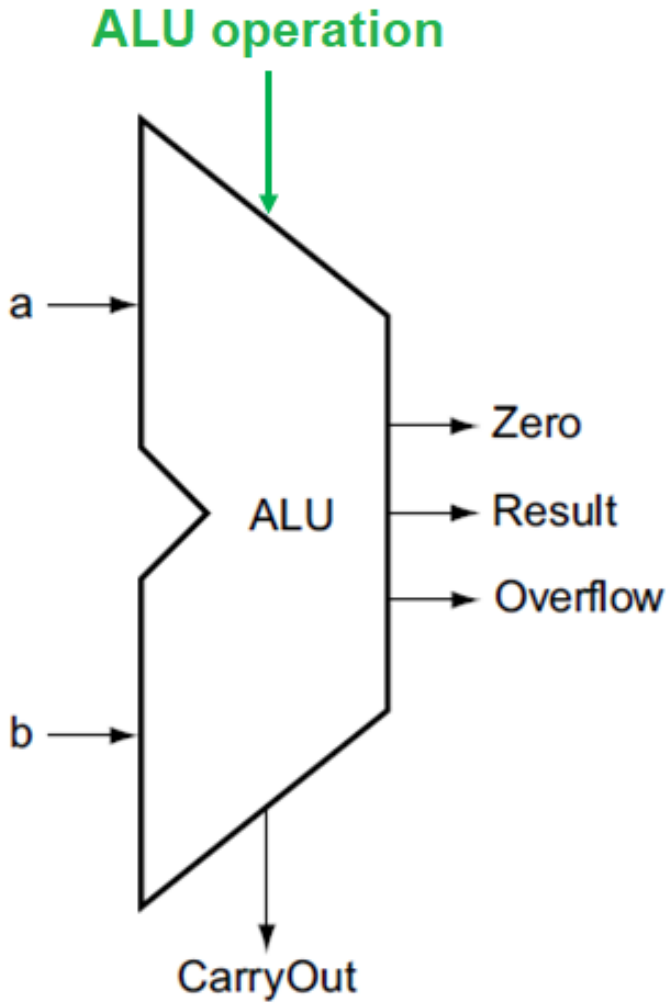
Instruções executadas pela ULA

Instruções	Operação
`add rd, rs1, rs2`	`rd = rs1 + rs2`
`sub rd, rs1, rs2`	`rd = rs1 - rs2`
`addi rd, rs1, imm`	`rd = rs1 + imm`
`lw rd, imm(rs1)`	`rd = M[imm + rs1]`
`sw rs2, imm(rs1)`	`M[imm + rs1] = rs2`
`ld rd, imm(rs1)`	`rd = M[imm + rs1]`
`sd rs2, imm(rs1)`	`M[imm + rs1] = rs2`
`or rd, rs1, rs2`	`rd = rs1 | rs2`
`and rd, rs1, rs2`	`rd = rs1 & rs2`
`xor rd, rs1, rs2`	`rd = rs1 xor imm`
`bne rs1, rs2, L`	`if (rs1 != rs2)go to (PC+constante)`
`beq rs1, rs2, L`	`if (rs1 == rs2)go to (PC+constante)`
`jal rd, imm` `jalr rd, imm(rs1)`	`jal rd, imm` `jalr rd, imm(rs1)`
`sll rd, rs1, rs2` `srl rd, rs1, rs2`	`sll rd, rs1, rs2` `srl rd, rs1, rs2`

Operações:

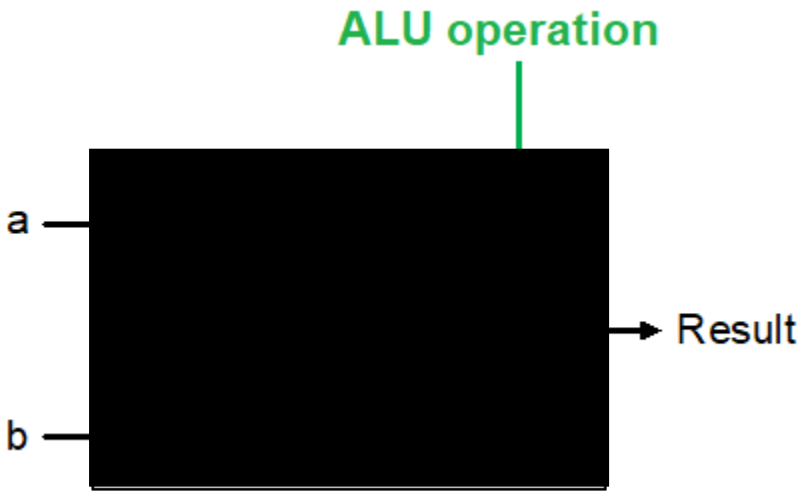
- `and`
- `or`
- `soma`
- `subtração`
- `...`

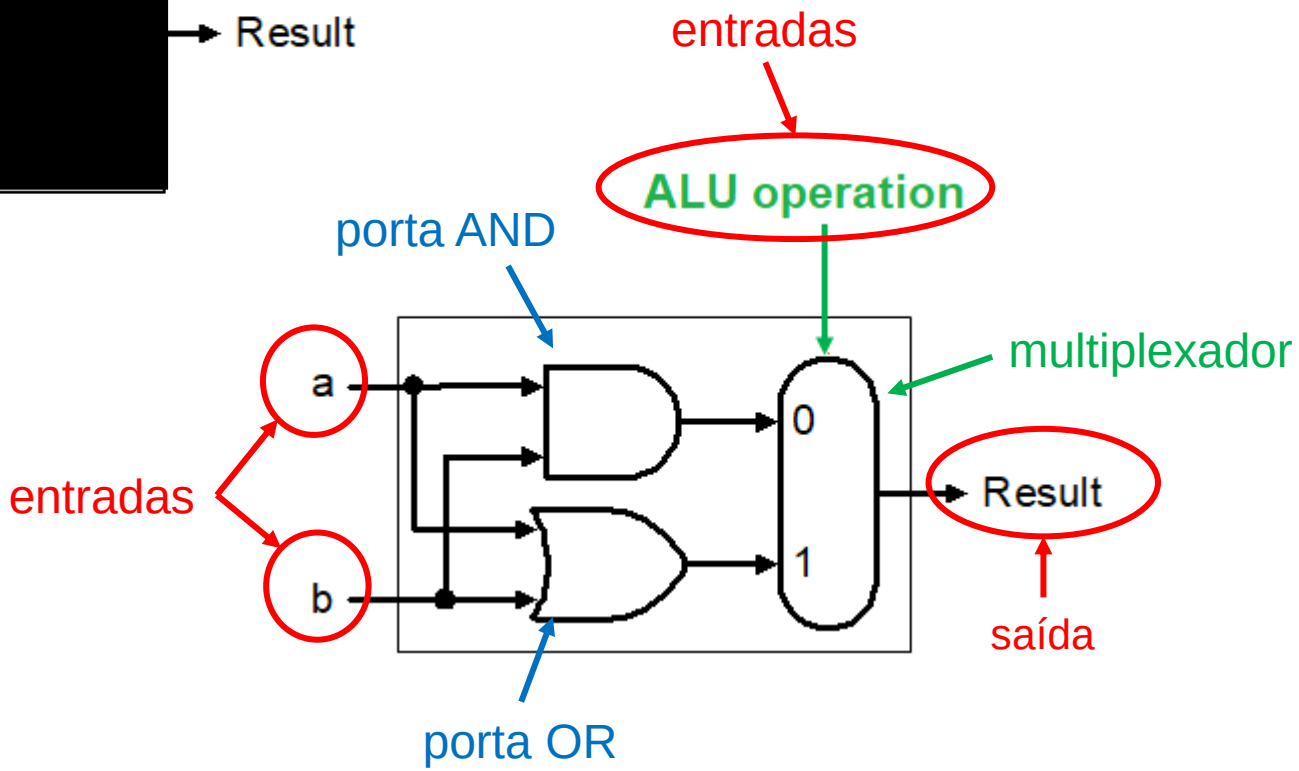
ULA (1bit)



- **a, b**: operandos
- **ALU operation**: qual operação será feita
- **Result**: resultado da operação
- **Zero**: bit em 1, se o resultado da operação resultou em zero
- **Overflow**: bit em 1, se gerou overflow

ULA - and e or (1 bit)

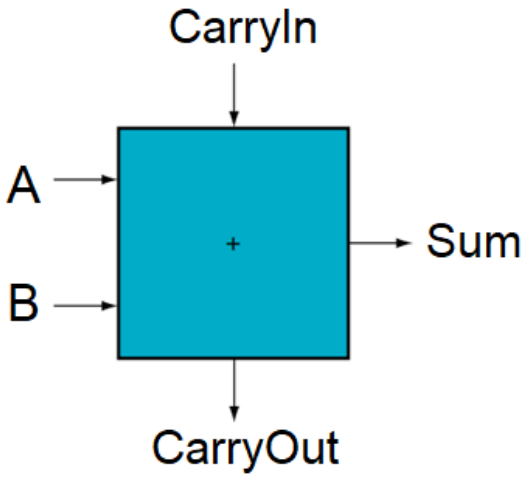




ULA - soma (1 bit)

- Com a operação de ***soma***
- ***Full Adder*** ou ***(3,2)-adder***
- É possível especificar as saídas baseado nas entradas através da tabela-verdade

$$\begin{array}{r} C_{out}\ C_{in} \\ + \quad A \\ \quad\ B \\ \hline \quad\ S \end{array}$$



S	C	B	S	C_{out}
0	A	0	0	0
1	B	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

ULA - soma (1 bit)

- Simplificação do circuito por Mapa de Karnaugh
- Para a saída C_{out}

C_{in}	A	B	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

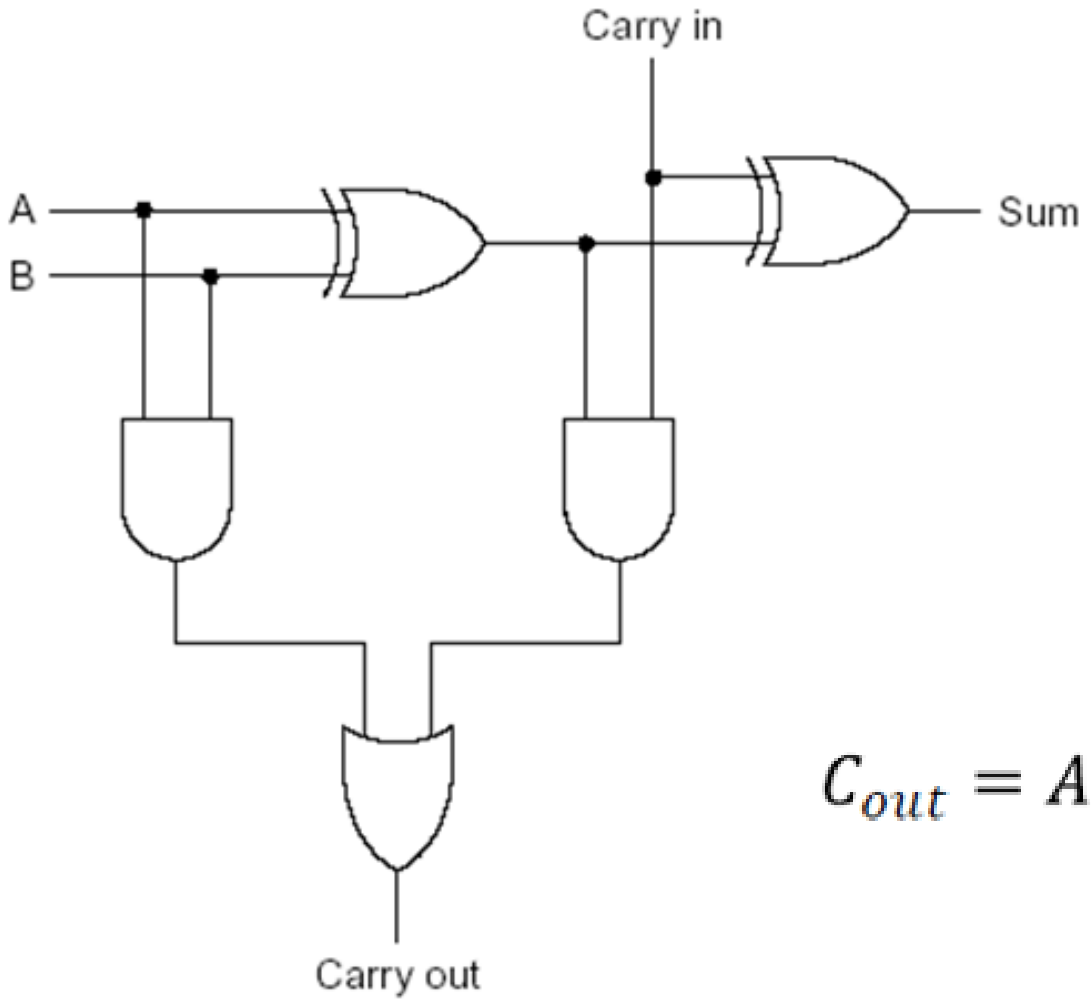
$C_{out} = ??$

		$\overline{C_{in}}$	C_{in}
	AB \ C	0	1
$\bar{A}\bar{B}$	00	0	1
$\bar{A}B$	01	2	1 3
AB	11	1 6	1 7
$A\bar{B}$	10	4	1 5

$$AB + AC_{in} + BC_{in}$$

$$C_{out} = AB + C_{in}(A + B)$$

ULA - soma (1 bit)



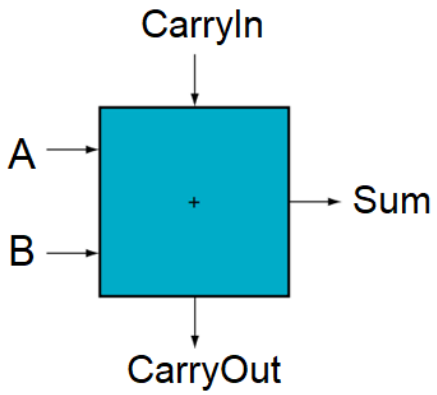
$$C_{out} = AB + C_{in}(A + B)$$

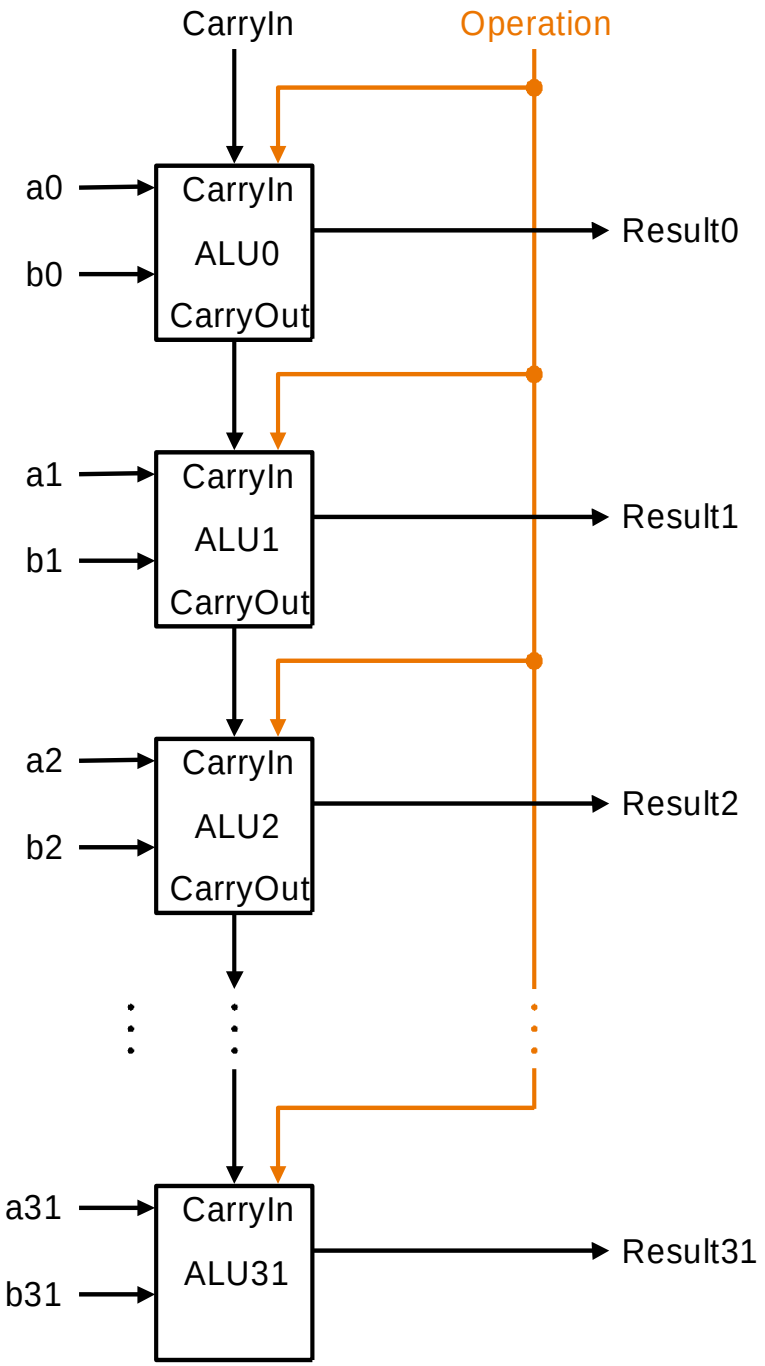
ULA - soma (1 bit)

- No entanto, sabemos que os registradores RISC-V têm 32 bits de largura
- Portanto, precisamos de uma ULA de 32 bits
- Conectaremos 32 ULAs de 1 bit para criar a ULA desejada.

Implementação da ULA (operação de soma com 32 bits)

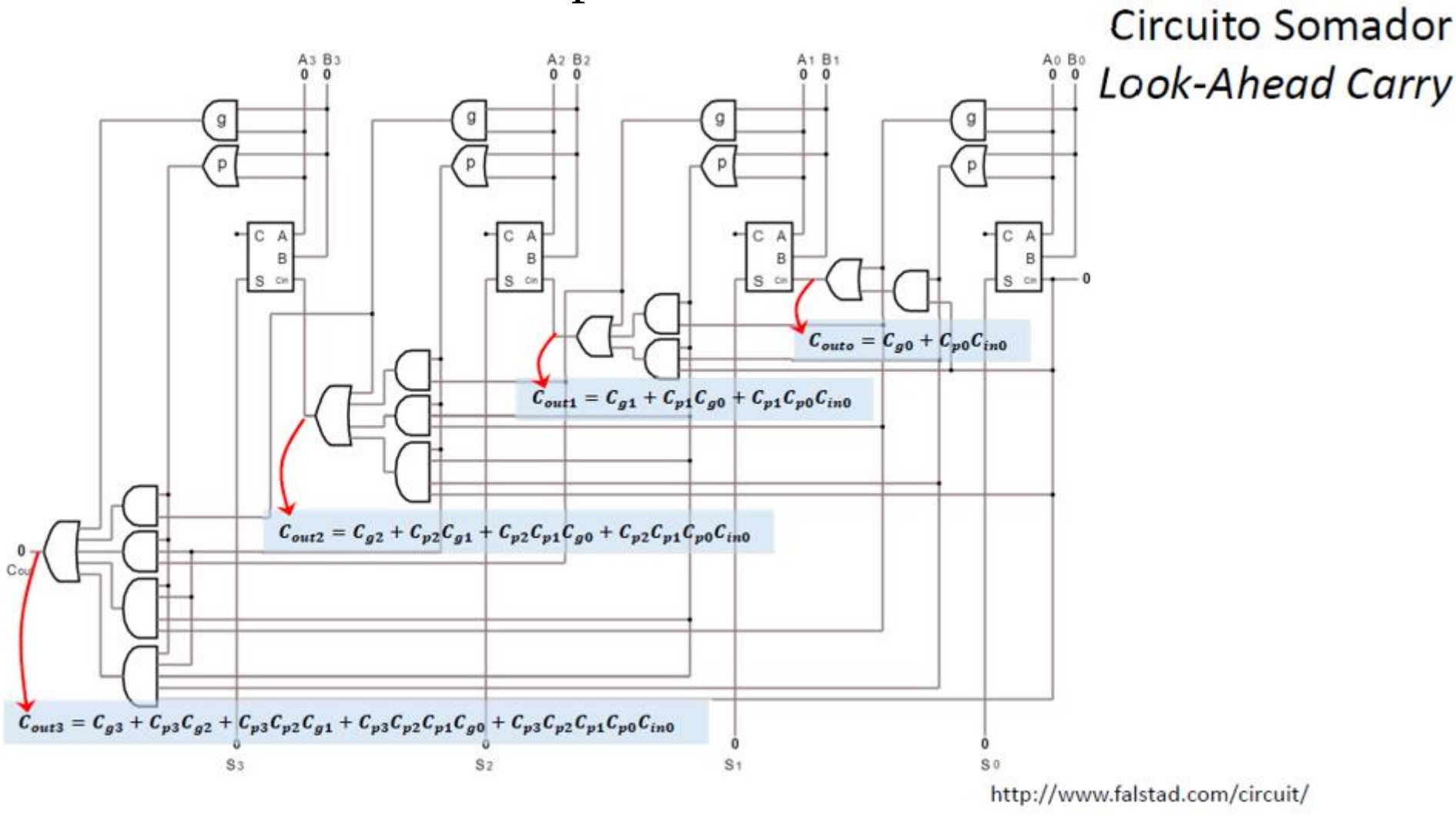
- Soma dos bits de maneira sequencial





ULA (soma otimizada com 3 bits

- Soma dos bits em paralelo



ULA (soma, subtração, AND e OR)

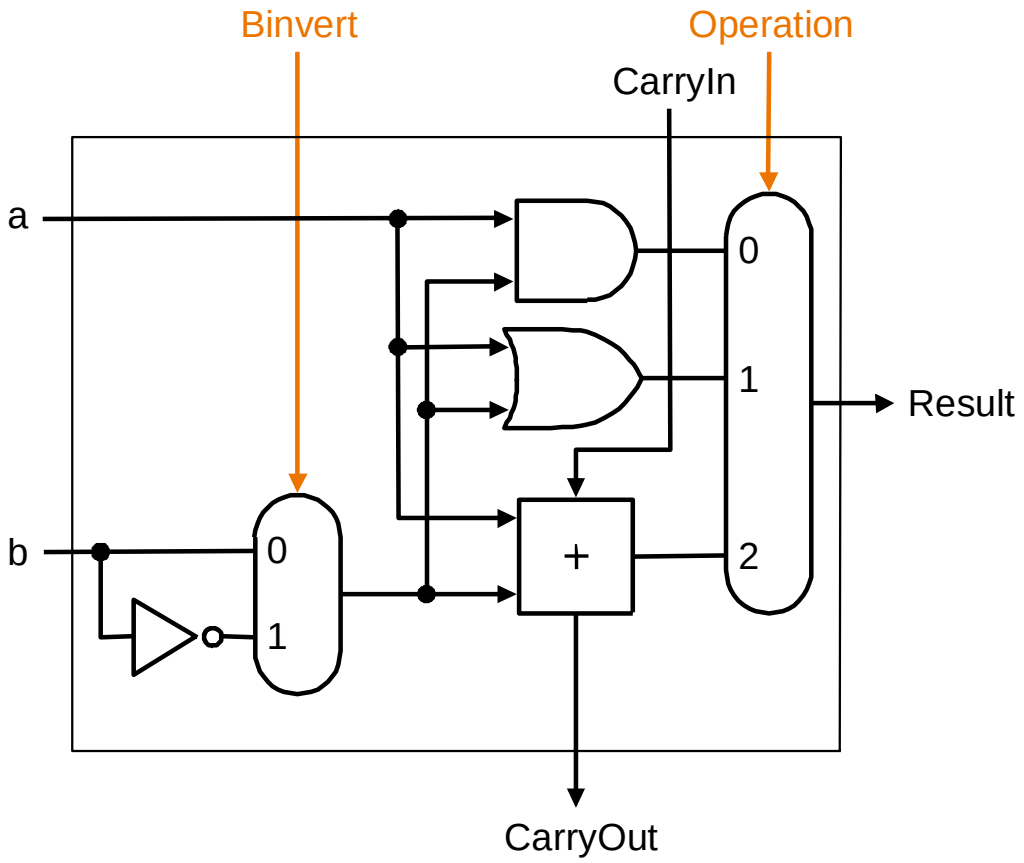
- Subtração:

$$a - b = a + (-b) = a + \bar{b} + 1$$

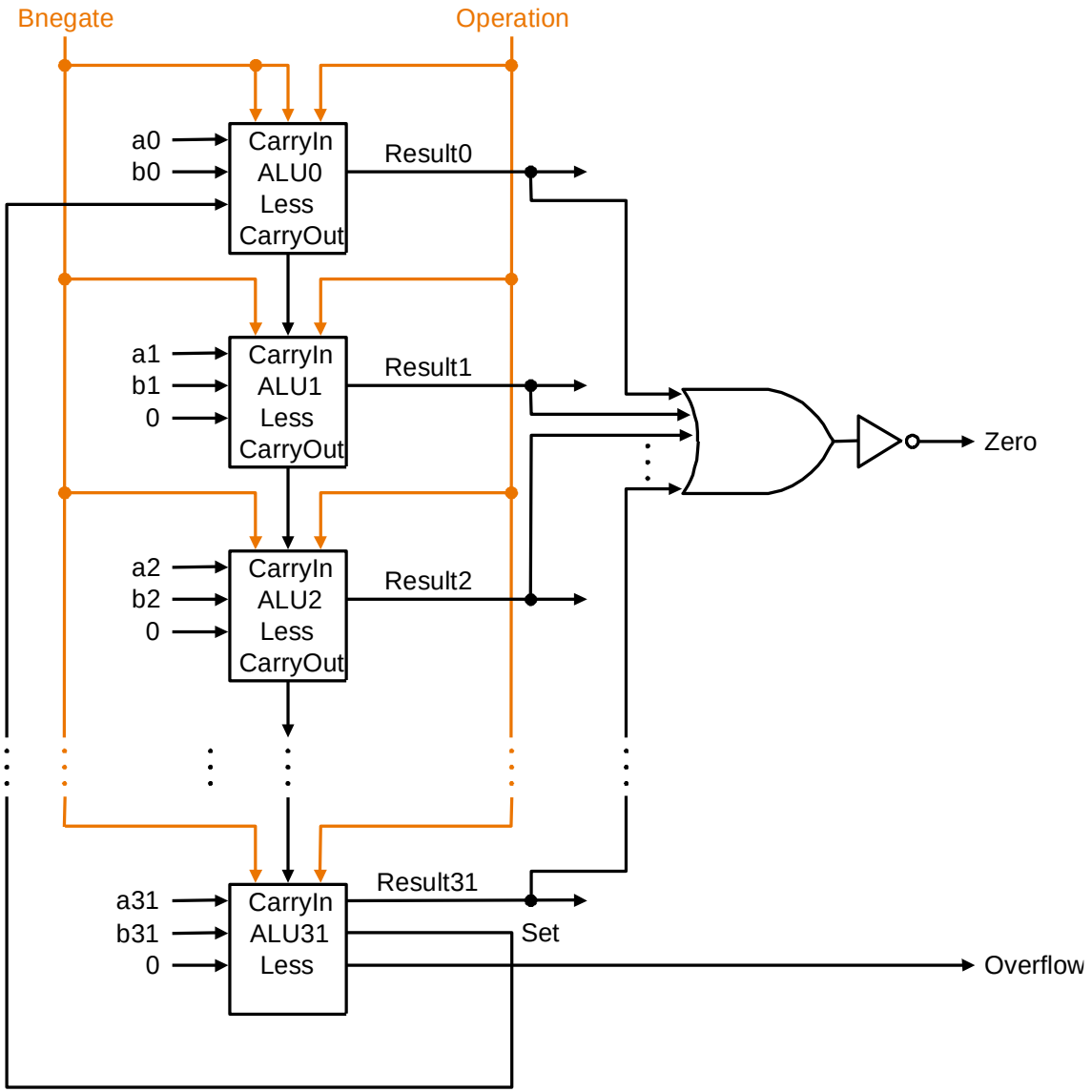
ULA (soma, subtração, AND e OR)

$$a - b = a + (-b) = a + \bar{b} + 1$$

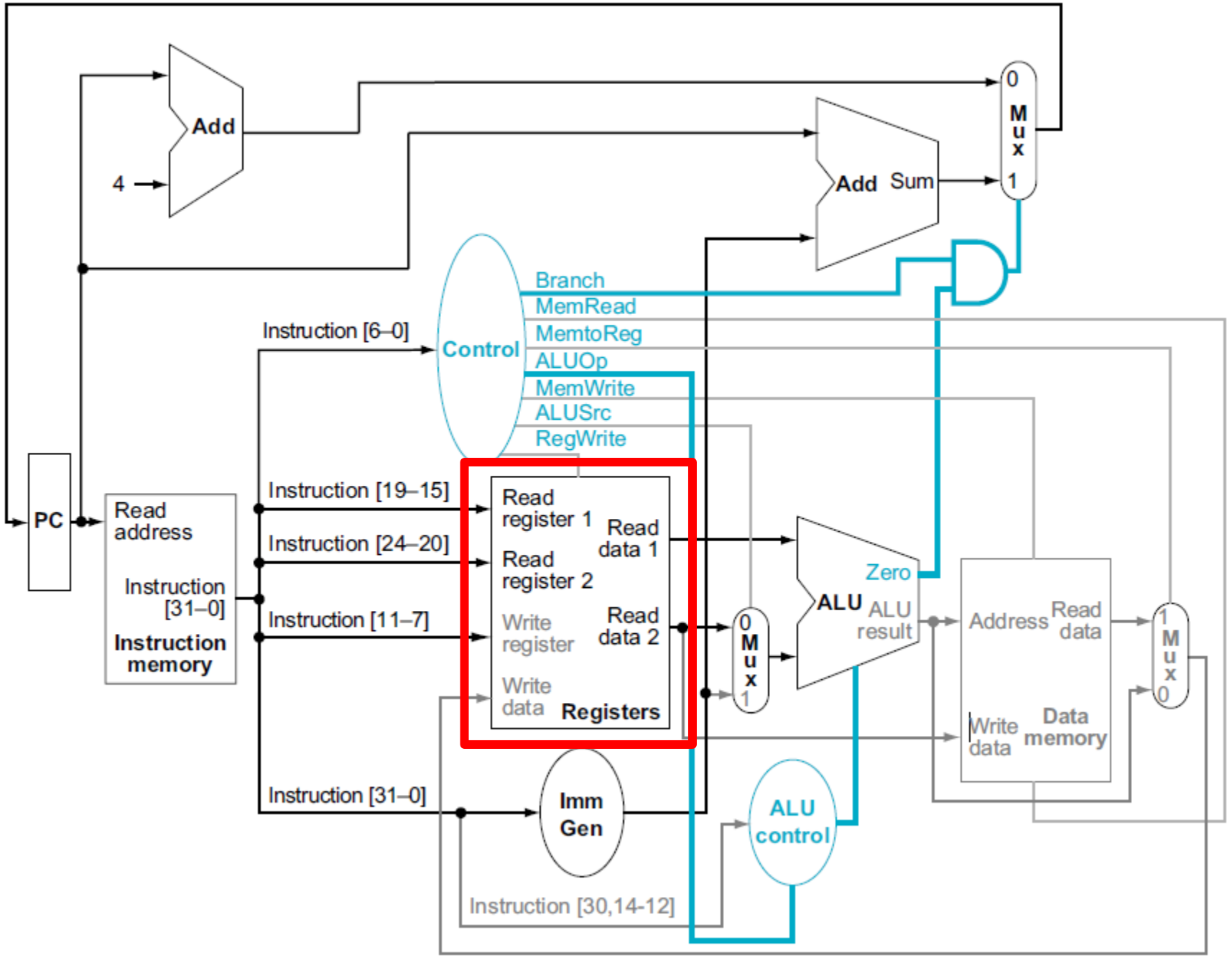
- **Binvert**: para indicar que **b** deve ser invertido
- **CarryIn** vale **1**, nestes casos, para ser somado ao b
- Operation seleciona o resultado da soma para ser transmitido



Implementação da ULA



Banco de Registradores

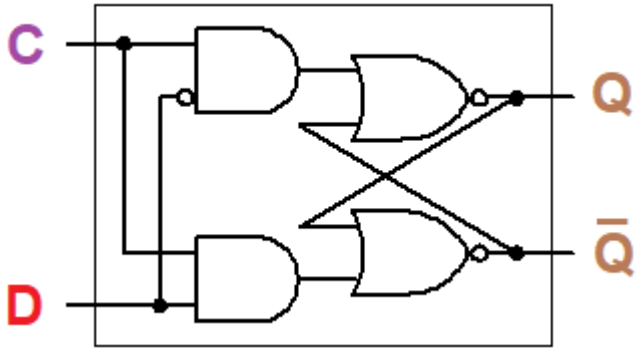

Add
4
Add
Sum
0
M
u
x
1
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [6–0]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [19–15]
Instruction [24–20]
Instruction [11–7]
Instruction [31–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Imm Gen
ALU
Zero
ALU result
ALU control
Address
Read data
Write data
Data memory
Instruction [30,14-12]

Implementação do conjunto de Registradores

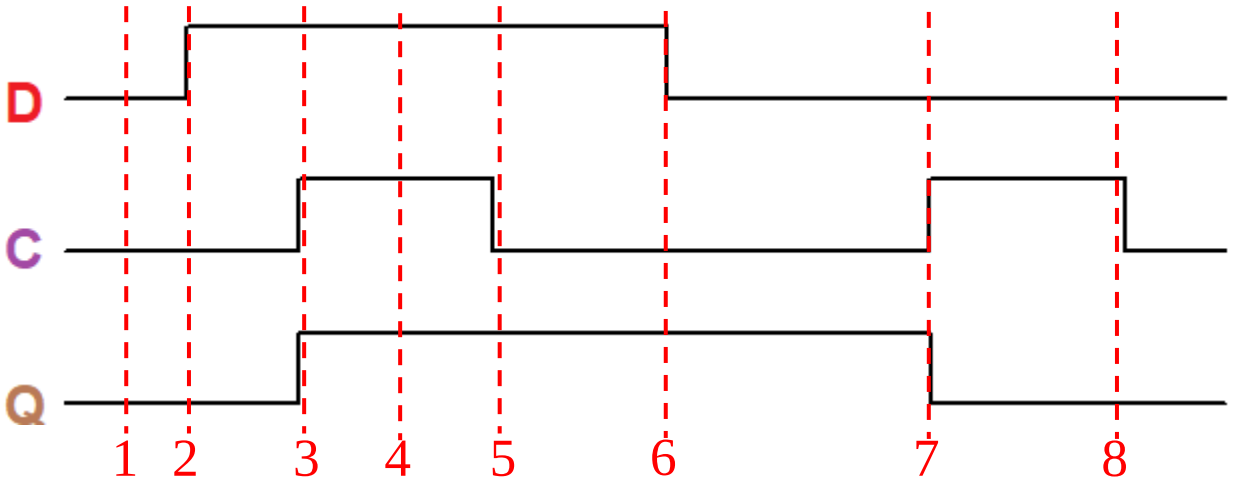
Latches:

- Para armazenar 1 bit
- Sensível ao nível do sinal

de controle

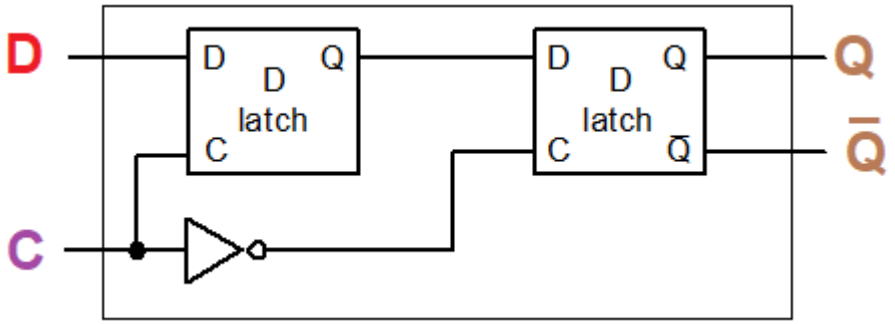


– O estado (**Q**) muda sempre que a entrada mudar (**D**) e o sinal de clock (**C**) estiver ativo

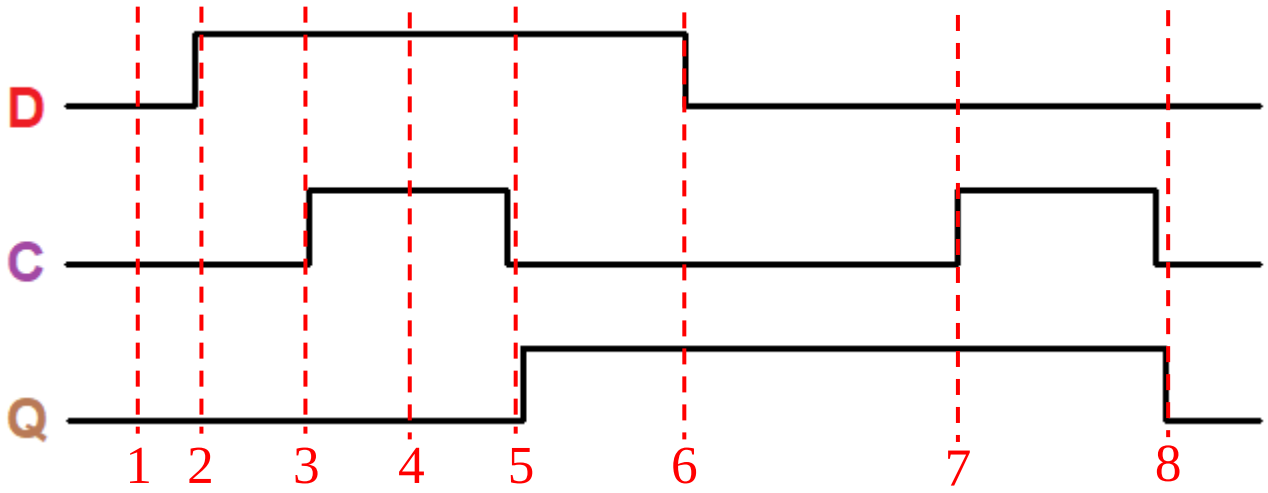


Implementação do conjunto de Registradores

- **Flip-flops**
- Para armazenar 1 bit



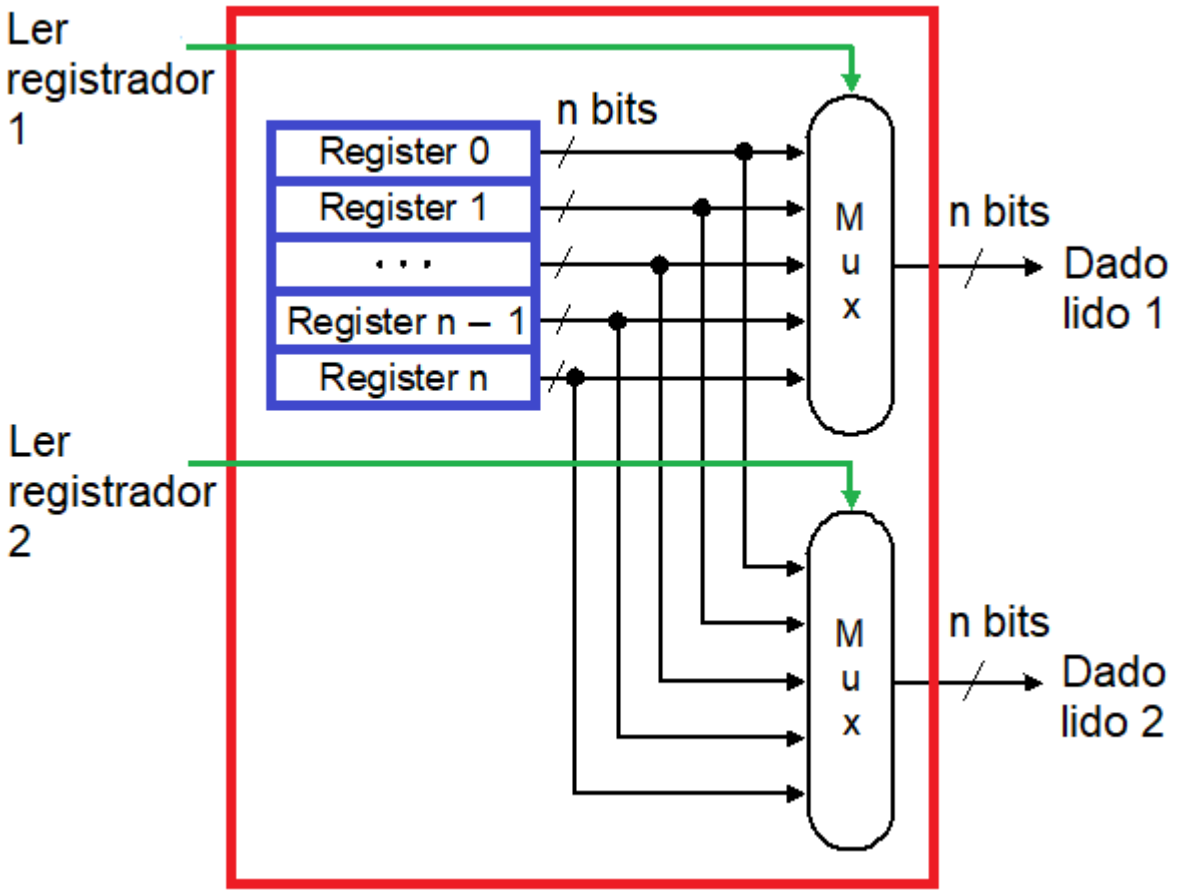
  - o estado (**Q**) muda somente na transição do clock (**C**)…
    - …de 0 para 1,
    - …ou de 1 para 0 (exemplo abaixo)



Implementação do conjunto de Registradores (leitura)

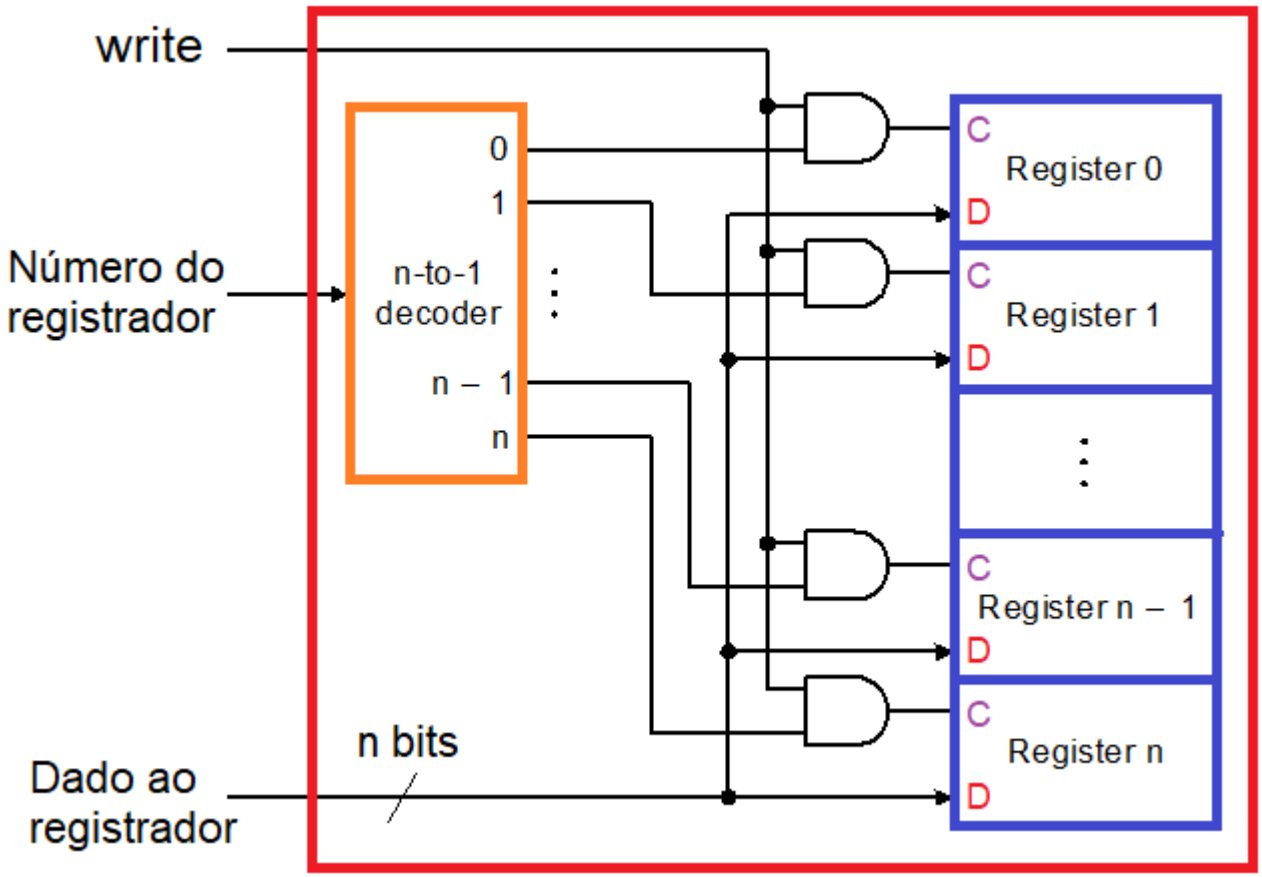
Ex: deseja-se ler os registradores de números 1 e 2

- **Seletores** dos Mux's selecionam os dois registradores
- Saídas de **n bits** com as informações dos registradores



Implementação do conjunto de Registradores (escrita)

- Decodificador na entrada seleciona o registrador que vai ser escrito
- Sinal ***write*** ativo
- ***Register data*** com a informação a ser gravada (**n bits**)



Implementação do conjunto de Registradores 4x3

- Questão: qual seria a “malha” do RISC-V? Quantos flip-flops?
- 32x32

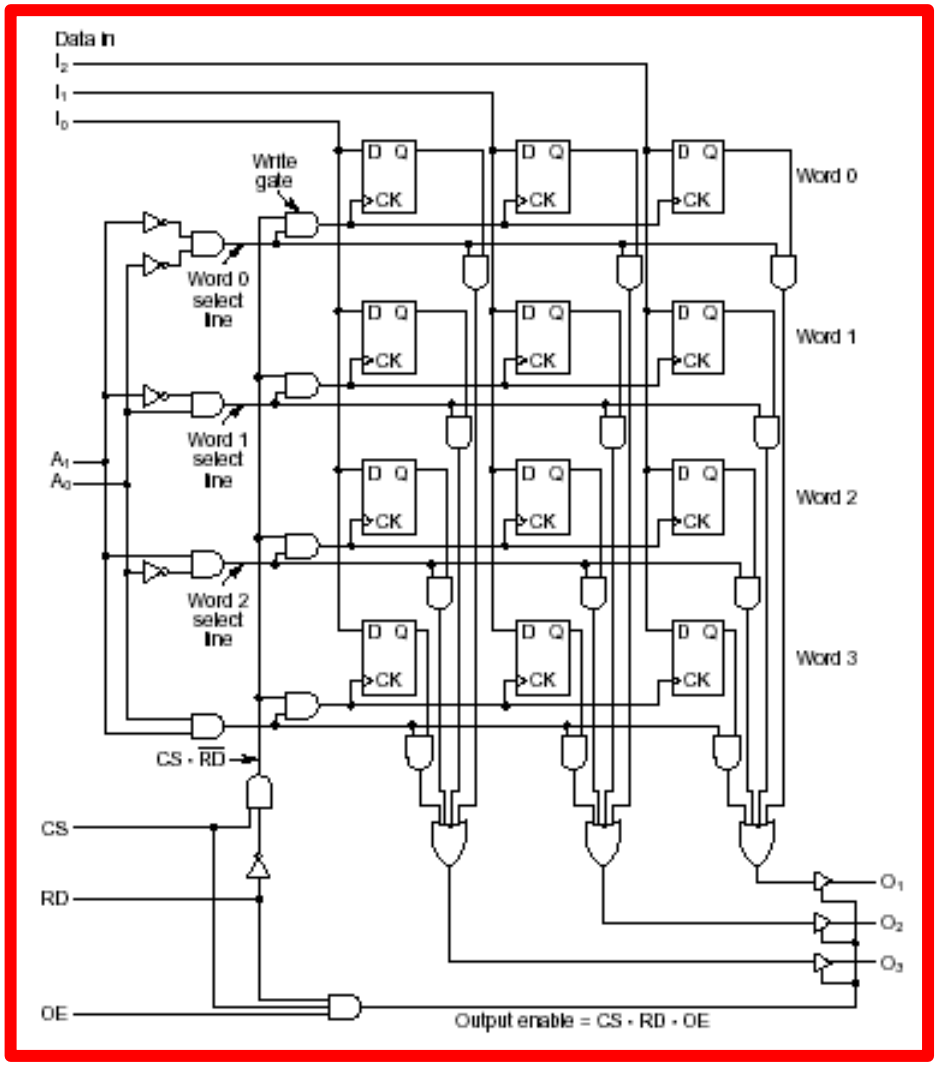
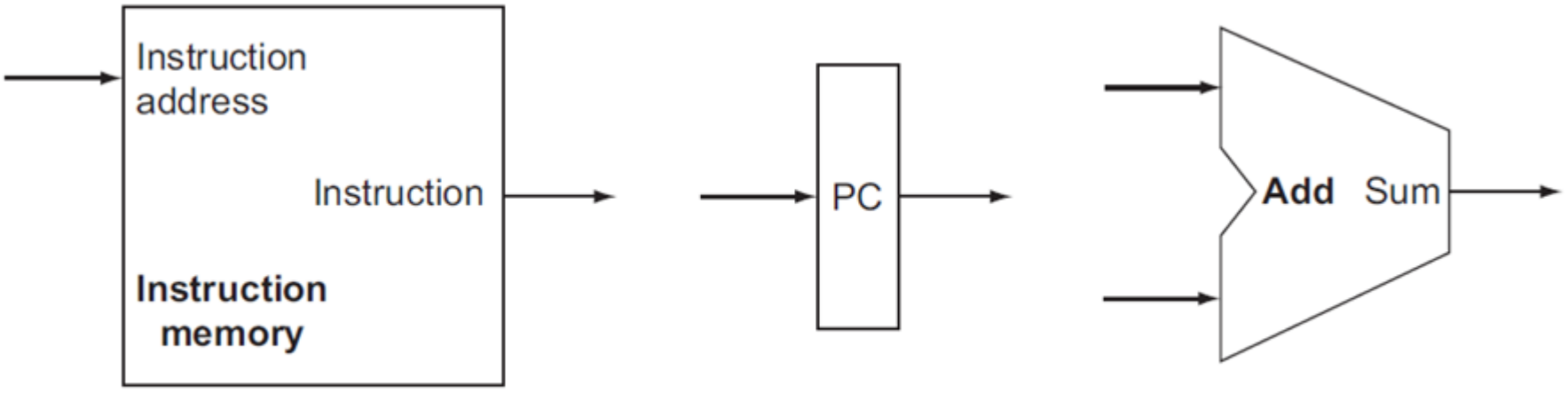


Figure 3-29. Logic diagram for a 4 × 3 memory. Each row is one of the four 3-bit words. A read or write operation always reads or writes a complete word.

Elementos necessários para a implementação do RISC V

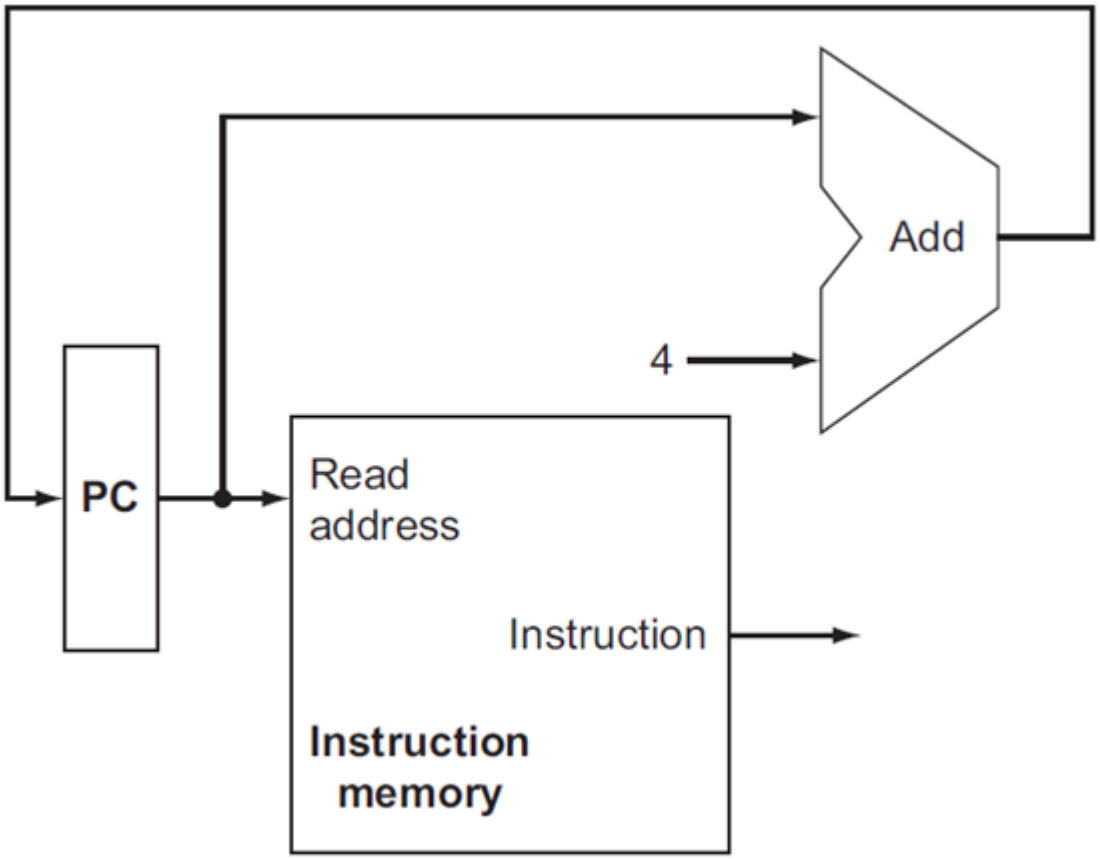
- Dois elementos de estado são necessários para armazenar e acessar instruções: **memória (RAM) das instruções** e o **PC**; e um **somador** é necessário para computar o endereço da próxima instrução



a. Instruction memory b. Program counter c. Adder

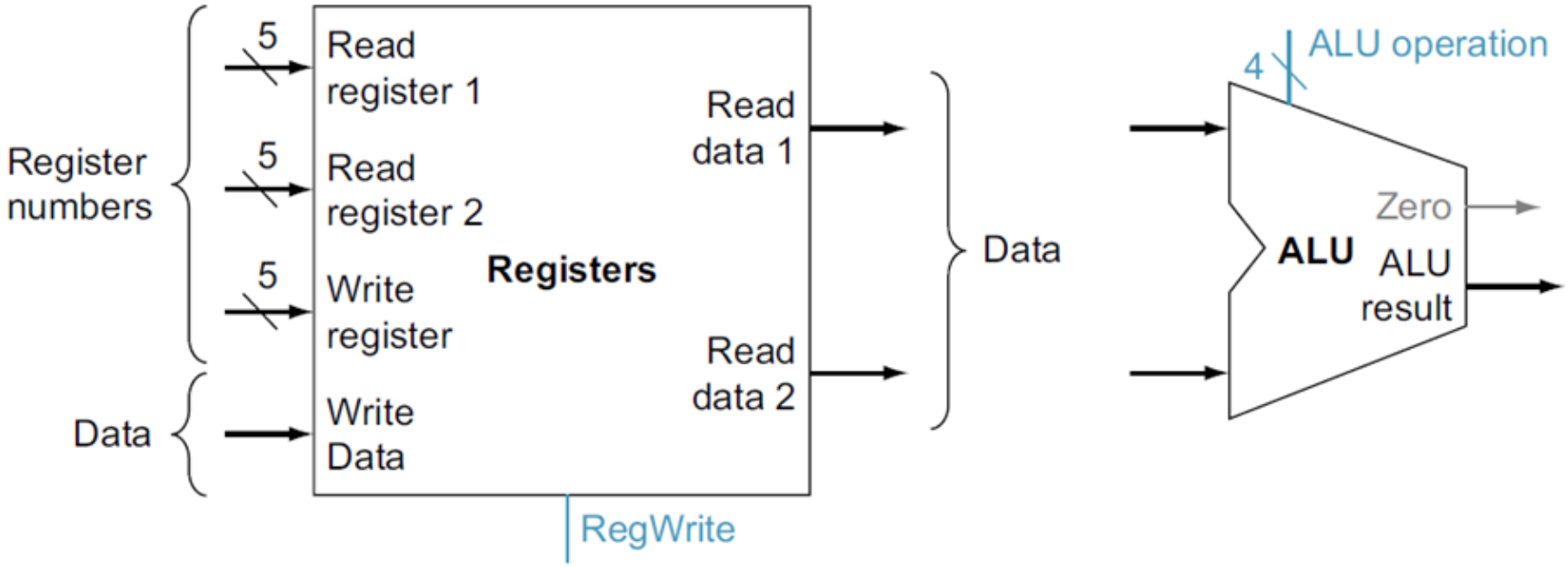
Elementos necessários para a implementação do RISC V

- Parte do caminho de dados que busca instruções e incrementa o PC



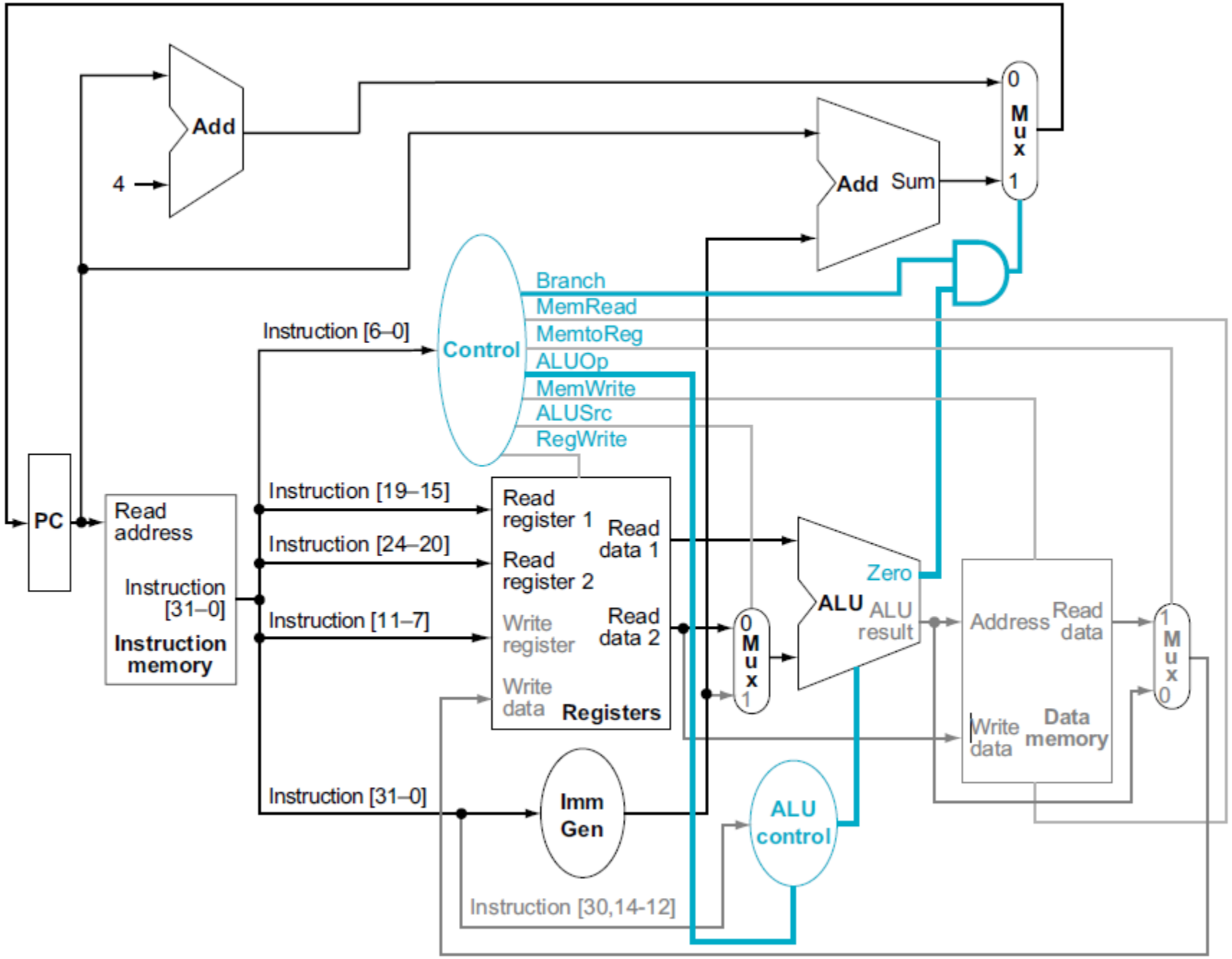
Elementos necessários para a implementação do RISC V

- Dois elementos necessários para implementar operações do Tipo-R são: o **conjunto de registradores** e a **ALU**



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**)
  - Armazena o resultado em locais apropriados
  - Volta ao passo 1 (inicia a execução da próxima instrução)


Add
4
Add
Sum
0
M
u
x
1
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [6–0]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [19–15]
Instruction [24–20]
Instruction [11–7]
Instruction [31–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Imm Gen
ALU
Zero
ALU result
ALU control
Address
Read data
Write data
Data memory
Instruction [30,14-12]

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.

Referências

- Slides do projeto de Ensino de Sistemas Digitais; Prof. Jorge Habib Hanna el Khouri; 2020.