Arquitetura de Computadores

Arquitetura do processador MIPS

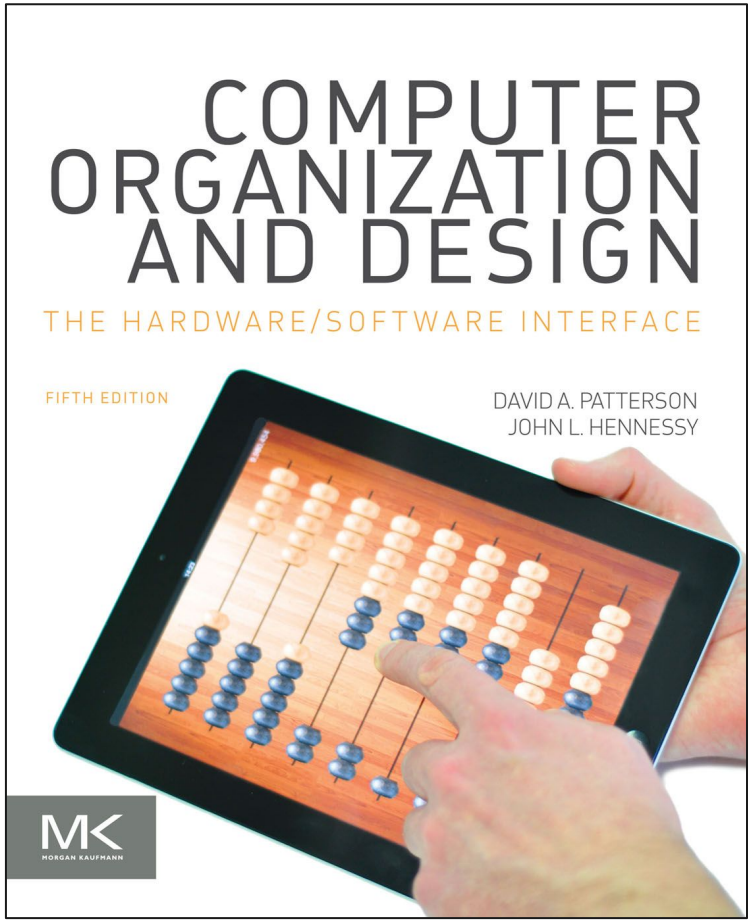
Uma implementação básica do MIPS

UNIVERSIDADE FEDERAL DE SERGIPE

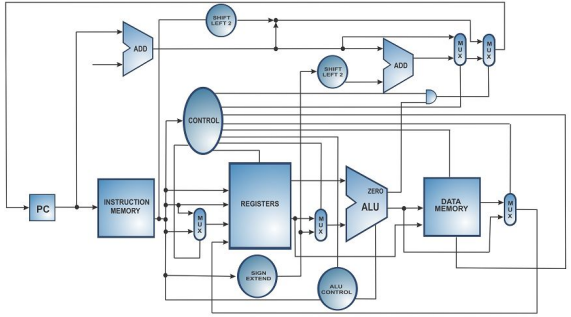
Professor
Calebe Conceição

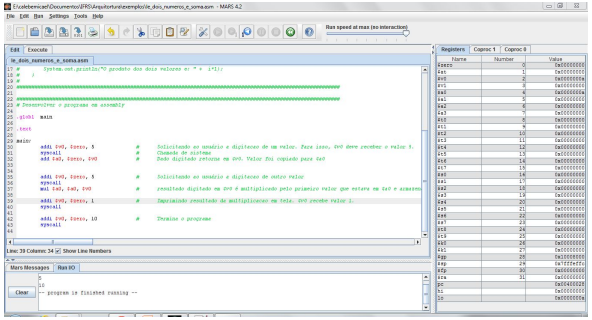
Aprofunde-se no livro texto!

Leia o capítulo 4 do Livro Computer Organization and Design, do Hennessy & Patterson.
(capítulo 5 na versão disponível na biblioteca)



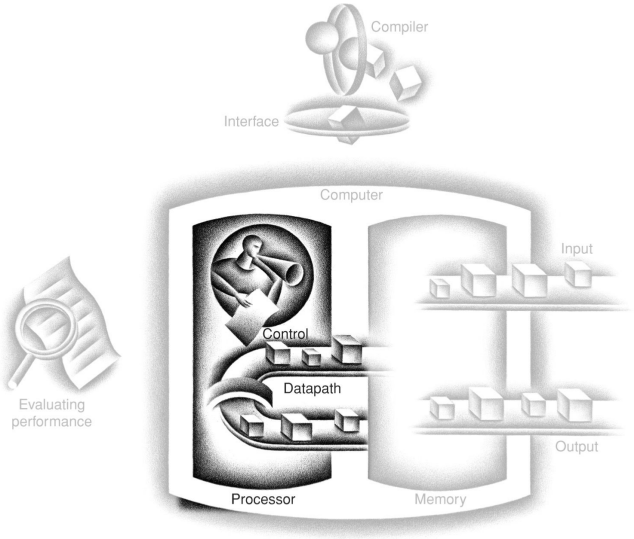
Na aula passada





- Vimos em mais detalhes o funcionamento de um processador.
  - Suas características gerais
  - Estudo inicial da Unidade Central de Processamento
  - Componentes da Via de Dados (*Datapath*)
  - Instruções e Via de Dados
  - Unidade de Controle
  - Arquiteturas RISC e CISC
- Conhecemos o assembly do MIPS e fizemos nossos primeiros programas
  - Conhecemos as instruções principais
  - Apresentamos e usamos o Simulador MARS
  - Visitamos os detalhes da estrutura das instruções.

Na aula de hoje



- Conheceremos a estrutura do processador MIPS
  - As instruções de referência à memória load word (lw) e store word (sw)
  - As instruções lógicas e aritméticas add, sub, AND, OR, and slt
  - As instruções de desvio branch equal (beq) e jump (j), que adicionaremos por último.
- Veremos sob um viés de implementação
  - Verilog, Verilog, Verilog!

Revisão Rápida

Tipos de Instrução

As instruções no MIPS são classificadas em 3 tipos, de acordo com o formato:

- Tipo R → 3 operandos (registradores)
- Tipo I → Um dos operandos vem junto com a instrução
- Tipo J → Instruções de desvio incondicional. Sem operandos.
- Coprocessador → Não serão abordados.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?
Exemplo: `add $a0, $t0, $t1`

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:

```
add $a0, $t0, $t1
add $4 , $8 , $9
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:

```
add $a0, $t0, $t1
add $4 , $8 , $9
```

Add é do tipo R, então:

Formato:

```
add $rd, $rs, $rt
    $4 , $8 , $9
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:
```
add $a0, $t0, $t1
add $4 , $8 , $9
```

Add é do tipo R, então:

Formato:
```
add $rd, $rs, $rt
    $4 , $8 , $9
```

Campos:

```
------.-----.-----.-----.-----.------
opcode. rs  . rt  .  rd .  sa .    fn
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:
```
add $a0, $t0, $t1
add $4 , $8 , $9
```

Add é do tipo R, então:

Formato:
```
add $rd, $rs, $rt
    $4 , $8 , $9
```

Campos:
```
______.01000.01001.00100._____.______
opcode. rs  . rt  .  rd . sa .    fn
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															

Common MIPS instructions.

Notes: *op*, *funct*, *rd*, *rs*, *rt*, *imm*, *address*, *shamt* refer to fields in the instruction format. The program counter PC is assumed to point to the next instruction (usually 4 + the address of the current instruction). M is the byte-addressed main memory.

Assembly instruction	Instr. format	*op* *op/funct*	Meaning	Comments
`add $rd, $rs, $rt`	R	0/32	`$rd = $rs + $rt`	Add contents of two registers
`sub $rd, $rs, $rt`	R	0/34	`$rd = $rs - $rt`	Subtract contents of two registers
`addi $rt, $rs, imm`	I	8	`$rt = $rs + imm`	Add signed constant

$t5	13
$t6	14
$t7	15

$fp	30
$ra	31
pc	
hi	
lo	

UNIVERSIDADE FEDERAL DE SERGIPE

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:
```
add $a0, $t0, $t1
add $4 , $8 , $9
```

Add é do tipo R, então:

Formato:
```
add $rd, $rs, $rt
    $4 , $8 , $9
```

Campos:

```
000000.01000.01001.00100.?????.100000
opcode. rs  . rt  .  rd . sa .    fn
```

shift amount (shamt)

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:
```
add $a0, $t0, $t1
add $4 , $8 , $9
```

Add é do tipo R, então:

Formato:
```
add $rd, $rs, $rt
    $4 , $8 , $9
```

Campos:

```
000000.01000.01001.00100.00000.100000
opcode. rs  . rt  .  rd . sa .    fn
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?
Exemplo: `sll $t2, $s0, 4`

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:

```
sll $t2, $s0, 4
sll $10, $16, 4
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:

```
sll $t2, $s0, 4
sll $10, $16, 4
```

sll é do tipo R, então:

Formato:

```
sll $rd, $rt, shamt
    $10, $16, 4
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:

```
sll $t2, $s0, 4
sll $10, $16, 4
```

sll é do tipo R, então:

Formato:

```
sll $rd, $rt, shamt
    $10, $16, 4
```

Campos:

```
 ------.-----.-----.-----.-----.------
 opcode. rs  . rt  .  rd .  sa .    fn
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:

```
sll $t2, $s0, 4
sll $10, $16, 4
```

sll é do tipo R, então:

Formato:

```
sll $rd, $rt, shamt
    $10, $16, 4
```

Campos:

```
 ______._____.10000.01010.00100._______
 opcode. rs   . rt   .  rd .  sa .    fn
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															

Common MIPS instructions.

Notes: *op*, *funct*, *rd*, *rs*, *rt*, *imm*, *address*, *shamt* refer to fields in the instruction format. The program counter PC is assumed to point to the next instruction (usually 4 + the address of the current instruction). M is the byte-addressed main memory.

Assembly instruction	Instr. format	*op* *op/funct*	Meaning	Comments
`ori $rt, $rs, imm`	I	13	`$rt = $rs | imm`	Logical OR, unsigned constant
`sll $rd, $rt, shamt`	R	0/0	`$rd = $rt << shamt`	Shift left logical (shift in zeros)
`srl $rd, $rt, shamt`	R	0/2	`$rd = $rt >> shamt`	Shift right logical (shift in zeros)

$t6	14
$t7	15

$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:
```
add $a0, $t0, $t1
add $4 , $8 , $9
```

Add é do tipo R, então:

Formato:
```
add $rd, $rs, $rt
    $4 , $8 , $9
```

Campos:
```
000000._____.10000.01010.00100.000000
opcode. rs   . rt   .  rd .  sa .    fn
```

aqui rs é inútil

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:
```
add $a0, $t0, $t1
add $4 , $8 , $9
```

Add é do tipo R, então:

Formato:
```
add $rd, $rs, $rt
    $4 , $8 , $9
```

Campos:
```
000000.00000.10000.01010.00100.000000
opcode. rs   . rt   .  rd .  sa .    fn
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?
Exemplo: `addi $t2, $s0, 9`

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:

```
addi $t2, $s0, 9
addi $10, $16, 9
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:

```
addi $t2, $s0, 9
addi $10, $16, 9
```

addi é do tipo I, então:

Formato:

```
addi $rs, $rt, IMM
     $10, $16, 9
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:

```
addi $t2, $s0, 9
addi $10, $16, 9
```

addi é do tipo I, então:

Formato:

```
addi $rs, $rt, IMM
     $10, $16, 9
```

Campos:

```
------.-----.-----.----------------
opcode. rs   . rt   .              IMM
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:
```
addi $t2, $s0, 9
addi $10, $16, 9
```

addi é do tipo I, então:

Formato:
```
addi $rs, $rt, IMM
     $10, $16, 9
```

Campos:
```
______.01010.10000.0000000000001001
opcode. rs   . rt   .              IMM
```

dita a operação

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															

Common MIPS instructions.

Notes: *op*, *funct*, *rd*, *rs*, *rt*, *imm*, *address*, *shamt* refer to fields in the instruction format. The program counter PC is assumed to point to the next instruction (usually 4 + the address of the current instruction). M is the byte-addressed main memory.

Assembly instruction	Instr. format	*op* *op/funct*	Meaning	Comments
`add $rd, $rs, $rt`	R	0/32	`$rd = $rs + $rt`	Add contents of two registers
`sub $rd, $rs, $rt`	R	0/34	`$rd = $rs - $rt`	Subtract contents of two registers
`addi $rt, $rs, imm`	I	8	`$rt = $rs + imm`	Add signed constant

$t5	13
$t6	14
$t7	15

$fp	30
$ra	31
pc	
hi	
lo	

Tipos de Instrução

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Como traduzir o código assembly para binário?

Exemplo:

```
addi $t2, $s0, 9
addi $10, $16, 9
```

addi é do tipo I, então:

Formato:

```
addi $rs, $rt, IMM
     $10, $16, 9
```

Campos:

```
001000.01010.10000.0000000000001001
opcode. rs   . rt   .              IMM
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Verificando o aprendizado.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

1) Traduza o código assembly a seguir para binário.

```
addiu $t0, $zero, 7
```

Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Verificando o aprendizado.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

1) **Traduza o código assembly a seguir para binário.**

```
addiu $t0, $zero, 7
```

2) **Traduza o código assembly a seguir para binário.**

```
sub $a0, $s0, $s1
```

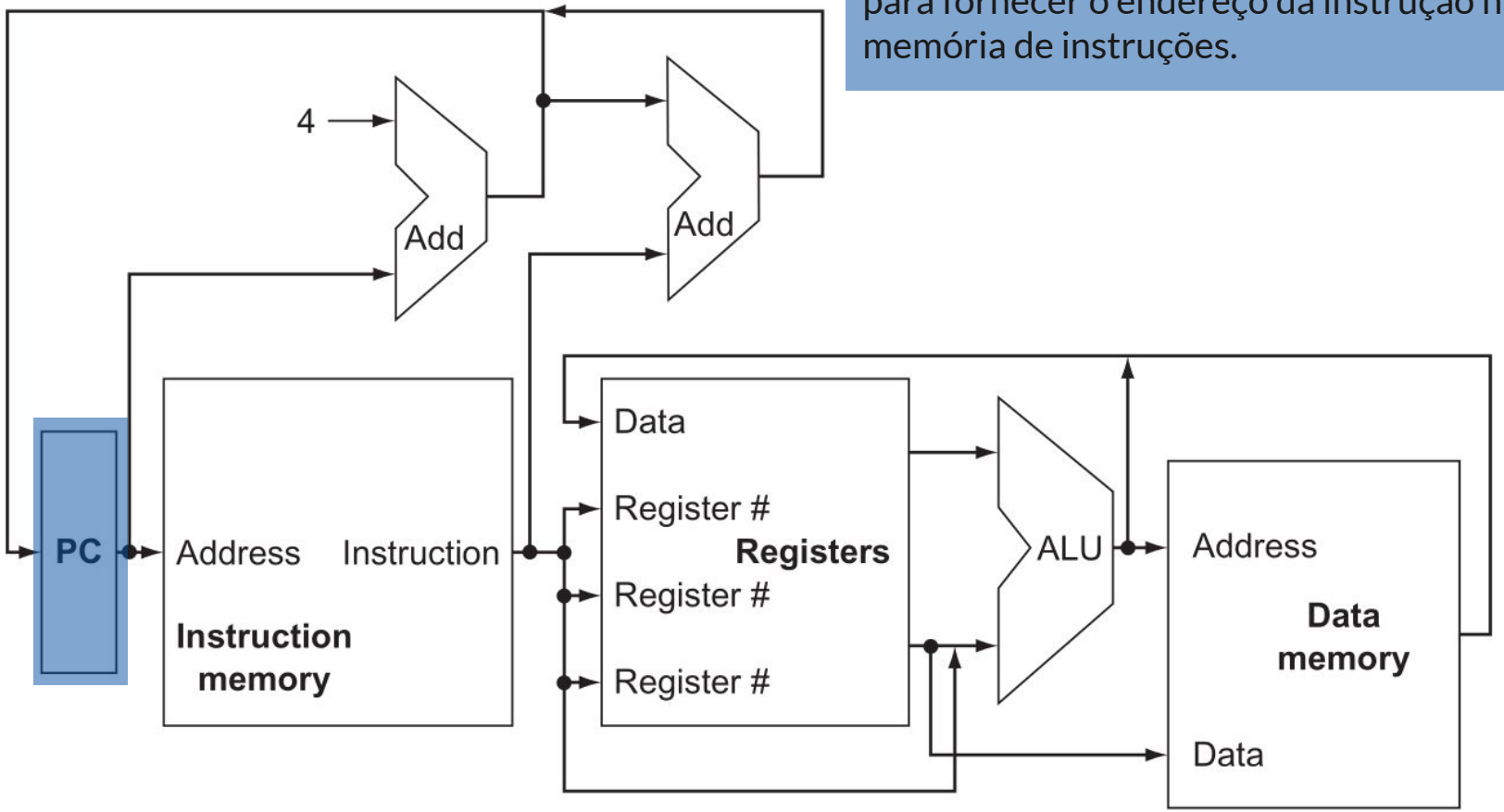
Name	Number
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15

$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31
pc	
hi	
lo	

Arquitetura do MIPS

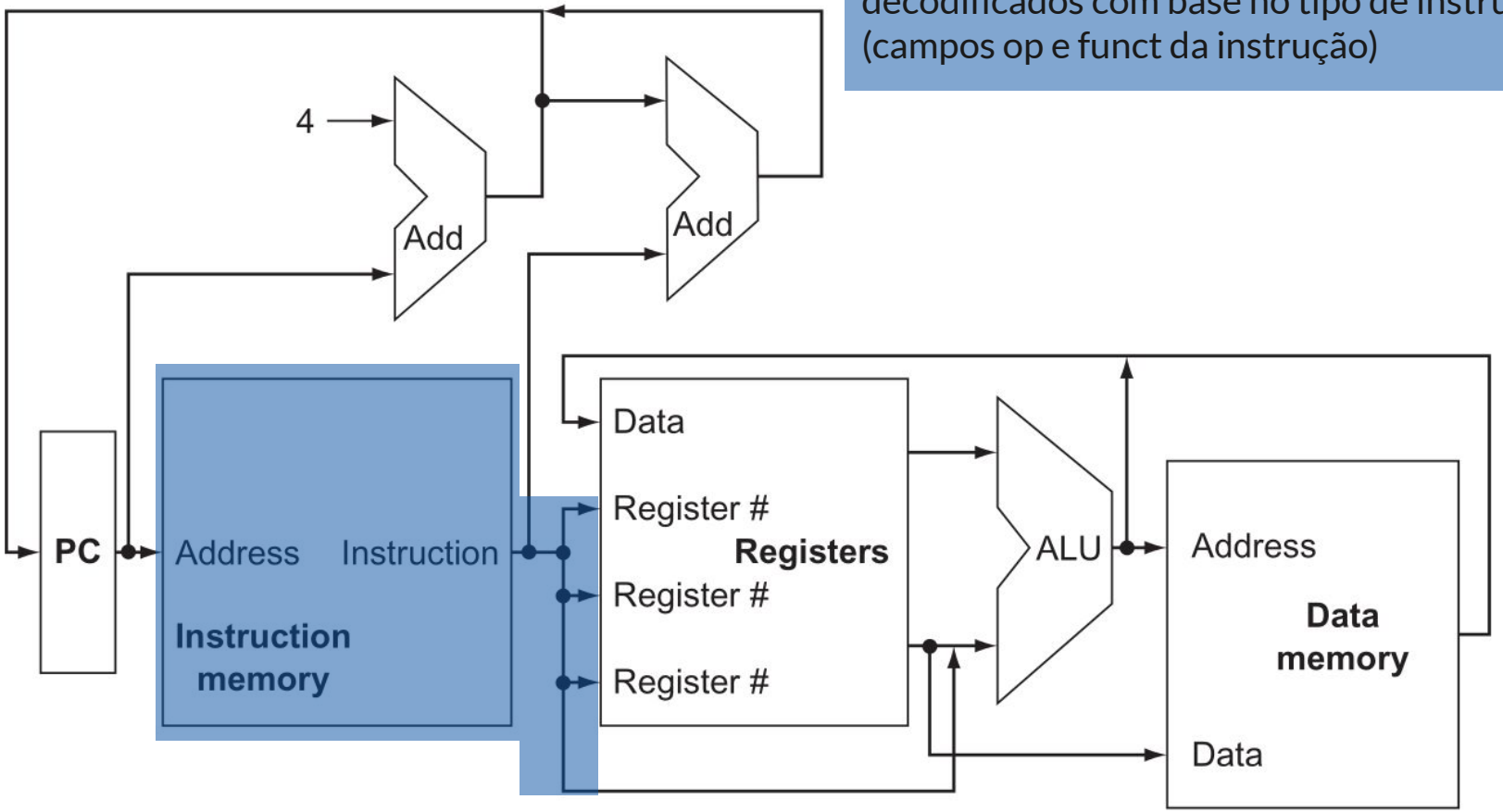
MIPS de ciclo único

Todas as instruções começam usando o PC para fornecer o endereço da instrução na memória de instruções.



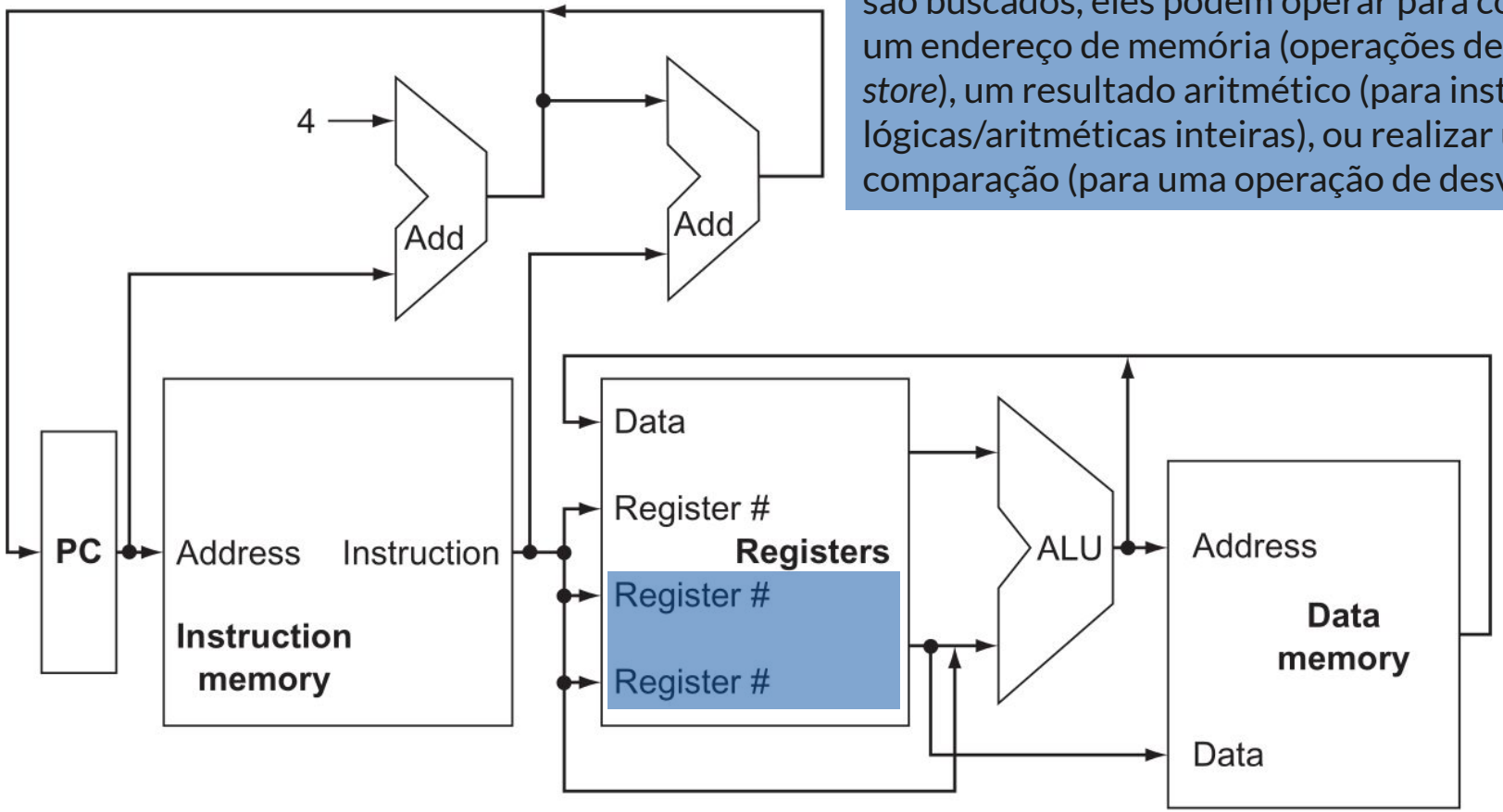
MIPS de ciclo único

Os endereços dos registradores são então decodificados com base no tipo de instrução (campos op e funct da instrução)



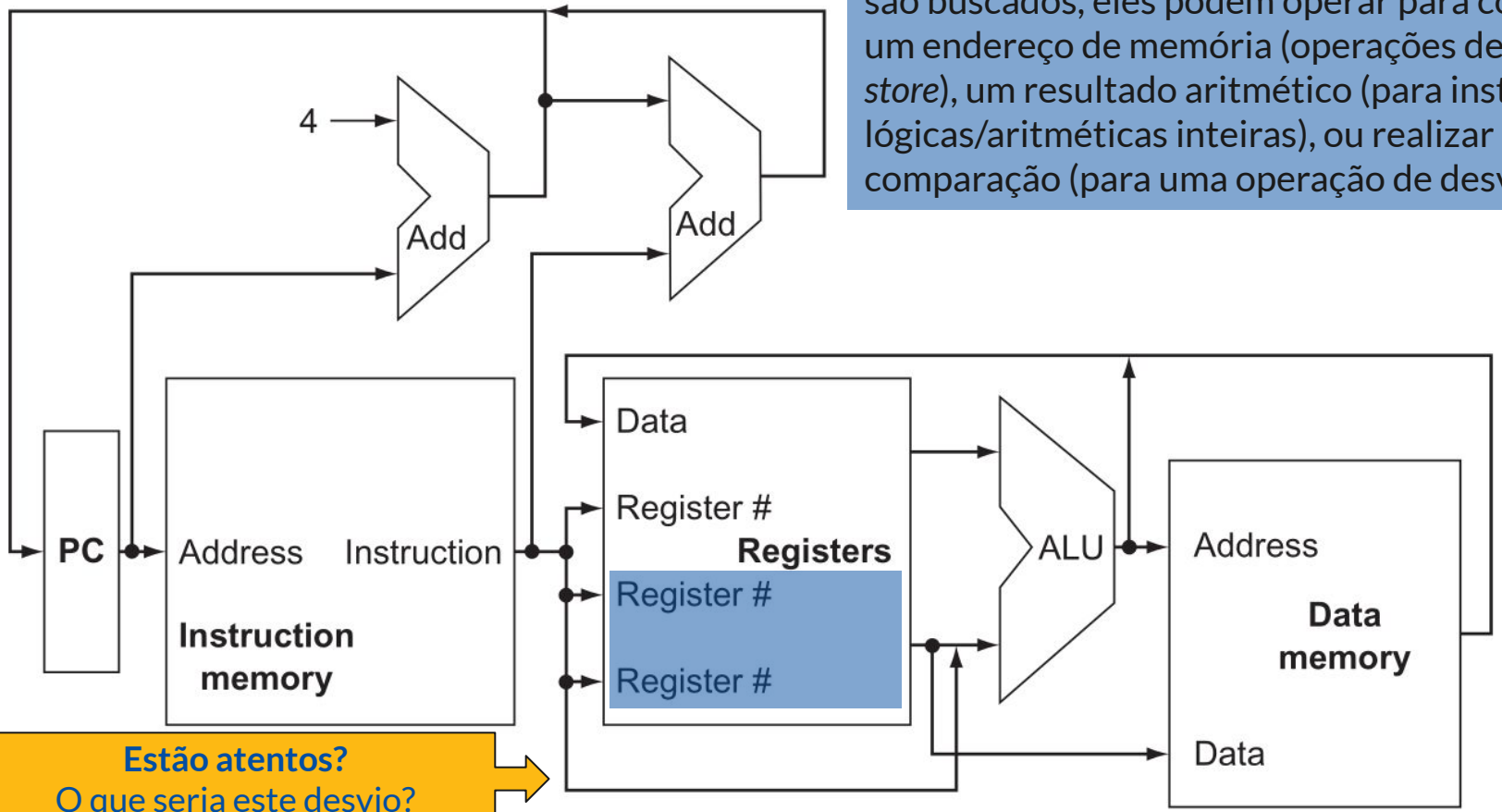
MIPS de ciclo único

Uma vez que os operandos (registradores) são buscados, eles podem operar para computar um endereço de memória (operações de *load* e *store*), um resultado aritmético (para instruções lógicas/aritméticas inteiras), ou realizar uma comparação (para uma operação de desvio).



MIPS de ciclo único

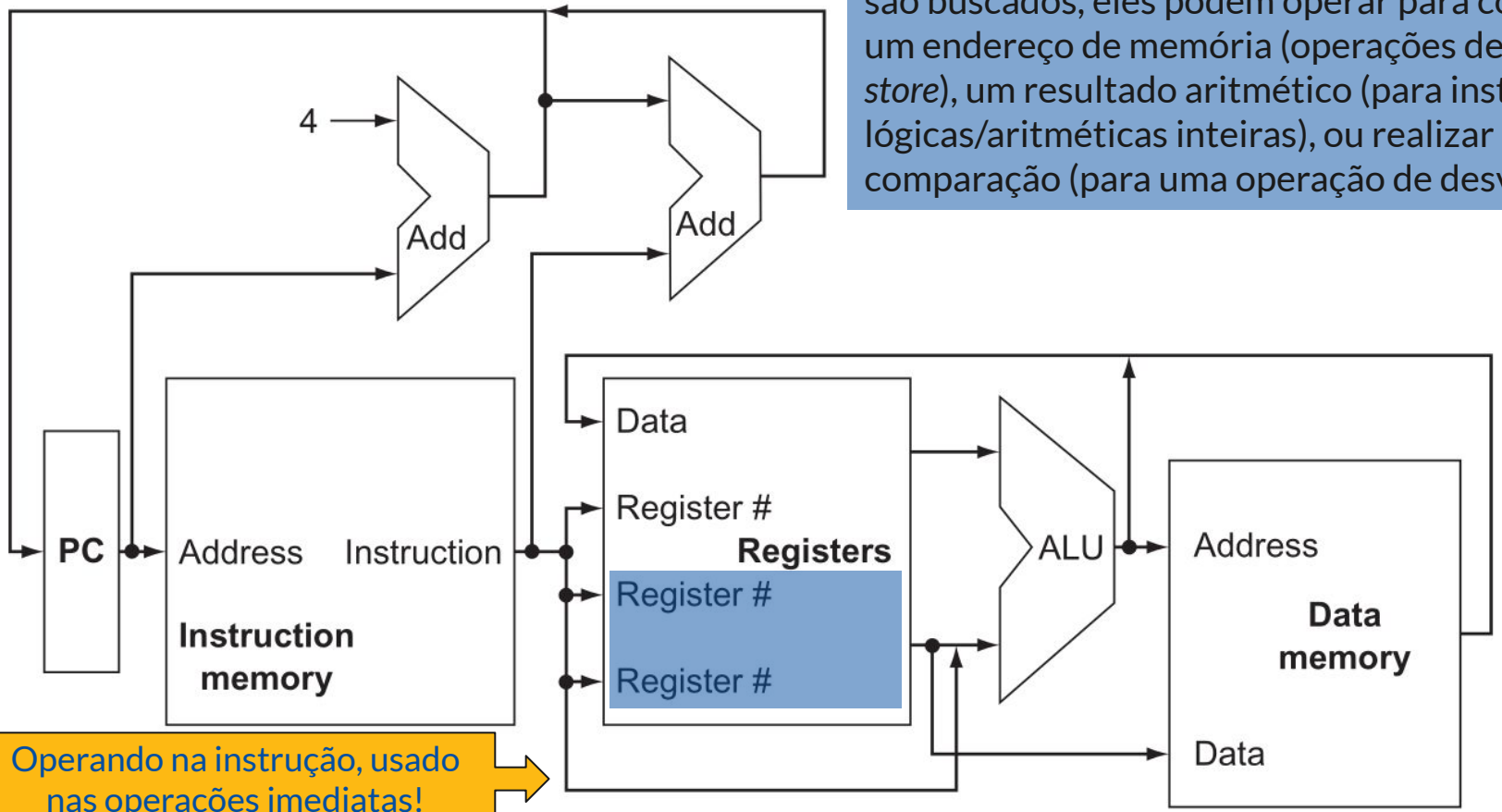
Uma vez que os operandos (registradores) são buscados, eles podem operar para computar um endereço de memória (operações de *load* e *store*), um resultado aritmético (para instruções lógicas/aritméticas inteiras), ou realizar uma comparação (para uma operação de desvio).



Estão atentos?
O que seria este desvio?

MIPS de ciclo único

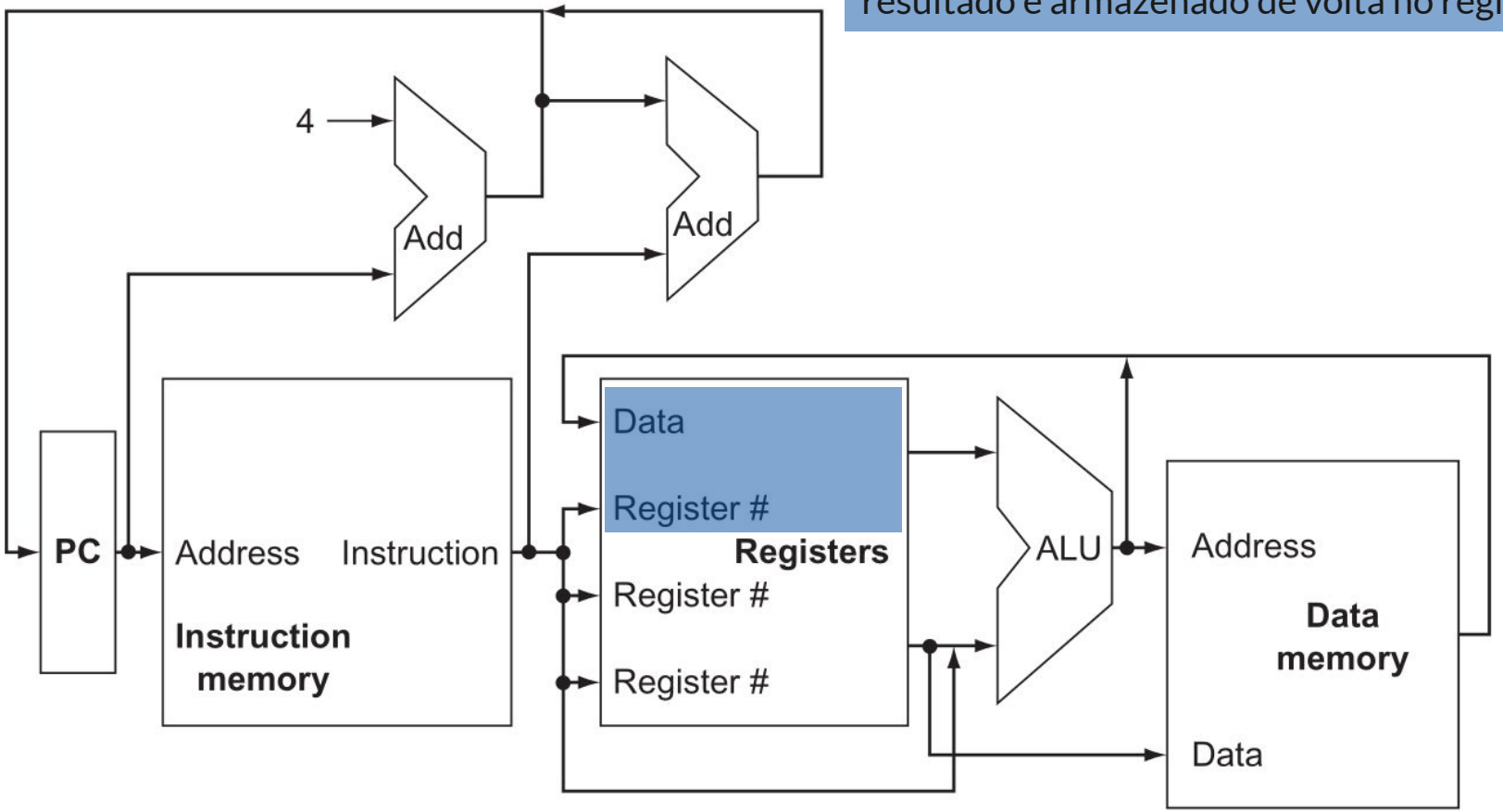
Uma vez que os operandos (registradores) são buscados, eles podem operar para computar um endereço de memória (operações de *load* e *store*), um resultado aritmético (para instruções lógicas/aritméticas inteiras), ou realizar uma comparação (para uma operação de desvio).



Operando na instrução, usado nas operações imediatas!

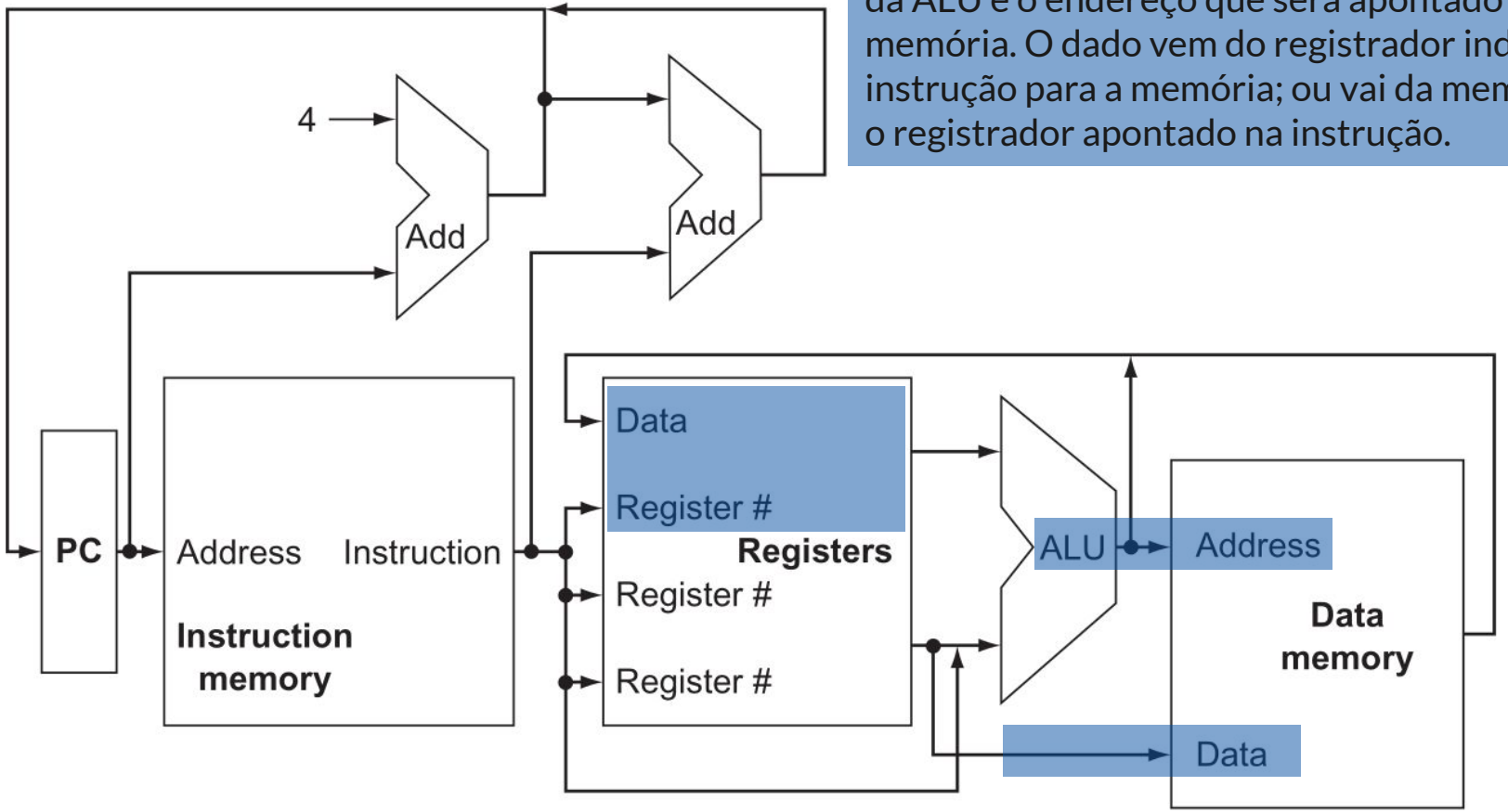
MIPS de ciclo único

Se a instrução é do tipo lógica-aritmética, o resultado é armazenado de volta no registrador



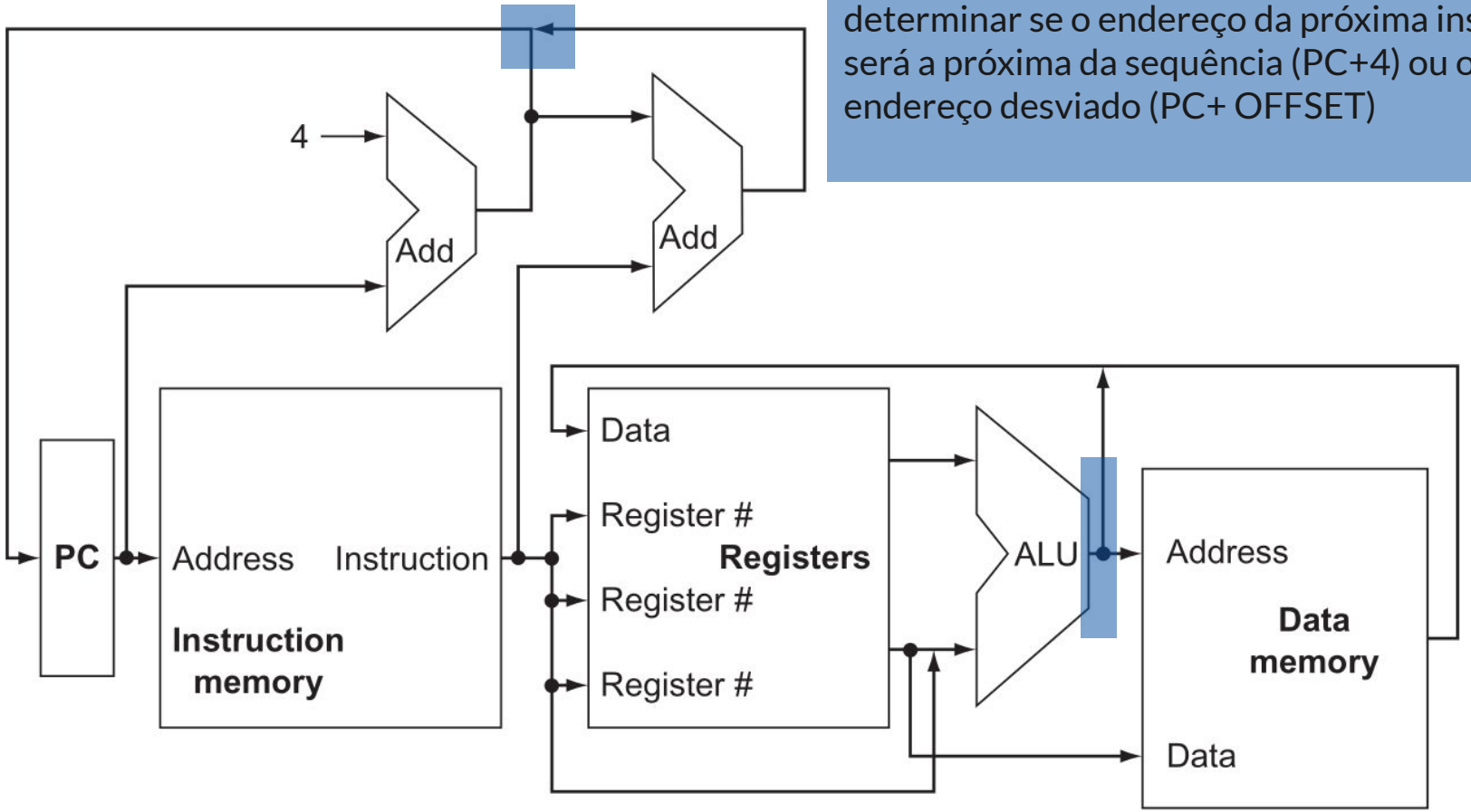
MIPS de ciclo único

Se a instrução é do load ou store, o resultado da ALU é o endereço que será apontado na memória. O dado vem do registrador indicado na instrução para a memória; ou vai da memória para o registrador apontado na instrução.

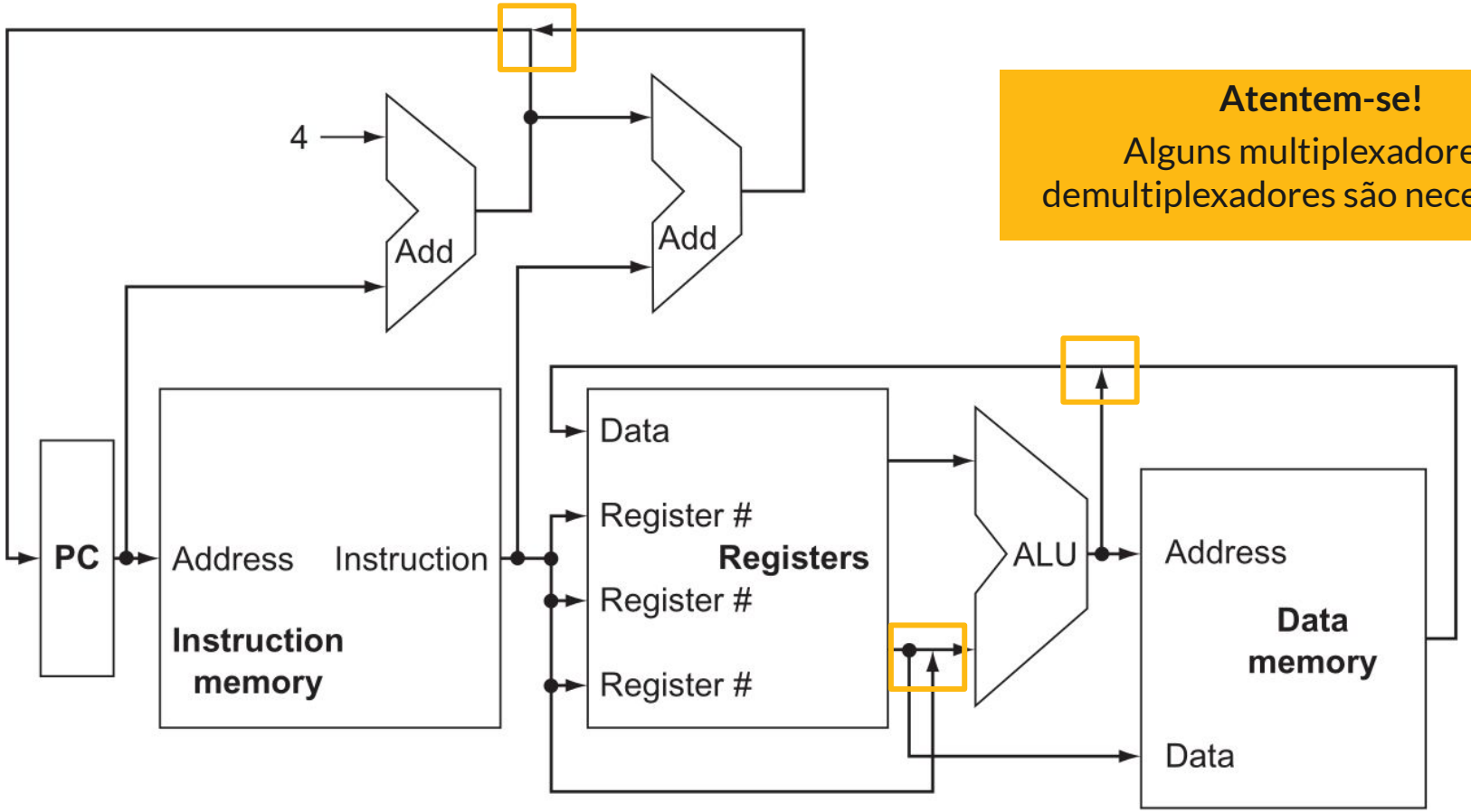


MIPS de ciclo único

Desvios precisam da saída da ALU para determinar se o endereço da próxima instrução será a próxima da sequência (PC+4) ou o endereço desviado (PC+ OFFSET)



MIPS de ciclo único



Atentem-se!

Alguns multiplexadores e demultiplexadores são necessários!

MIPS de ciclo único

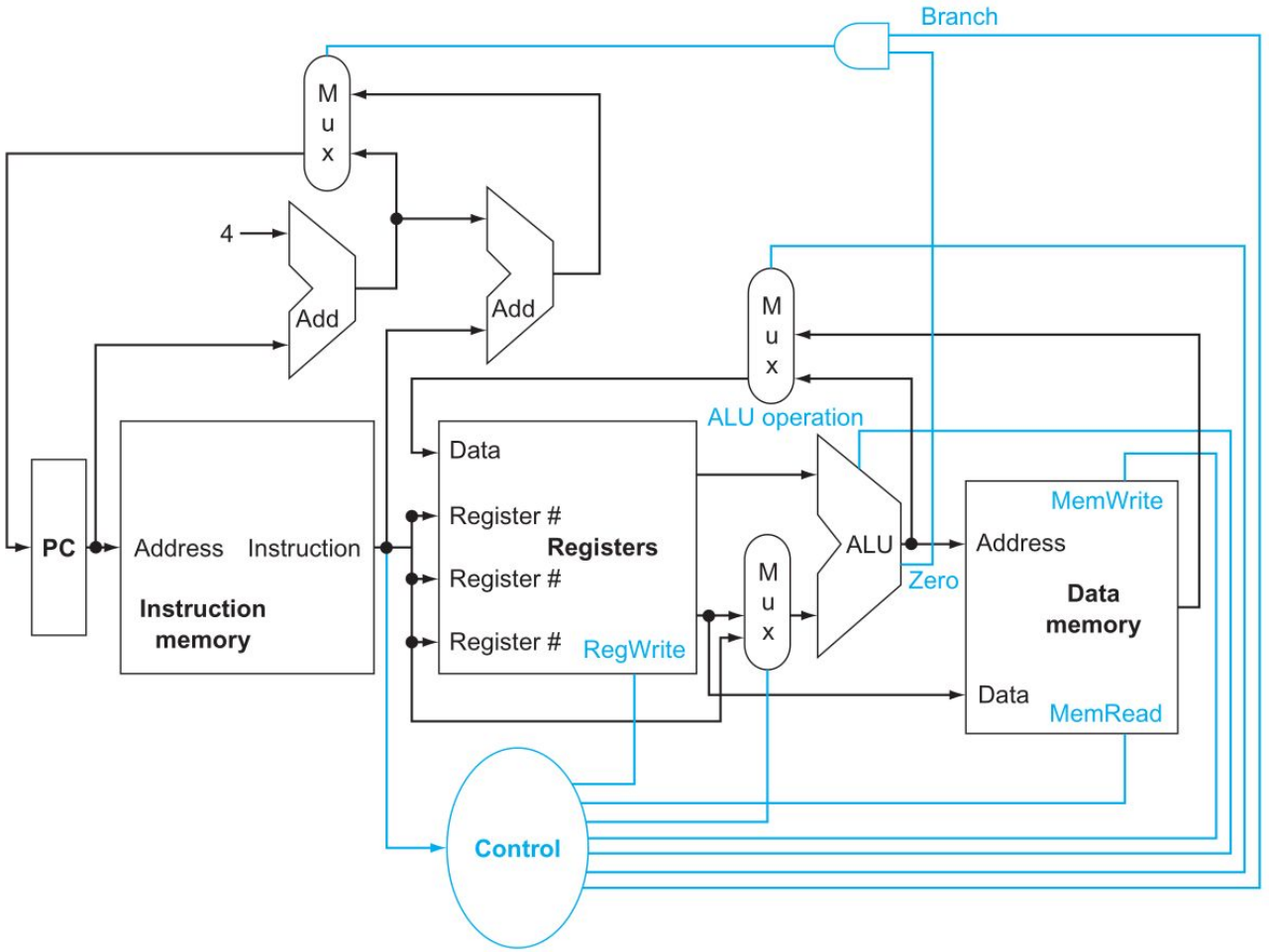
Adicionamos elementos de controle para:

Controlar a próxima instrução usando um indicador de saída 0 da ALU combinado com o sinal de branch do controle;

Habilitar a leitura e escrita da memória de dados e dos registradores;

Controlar a operação da ALU;

Escolher a origem do dado do registrador.



MIPS de ciclo único

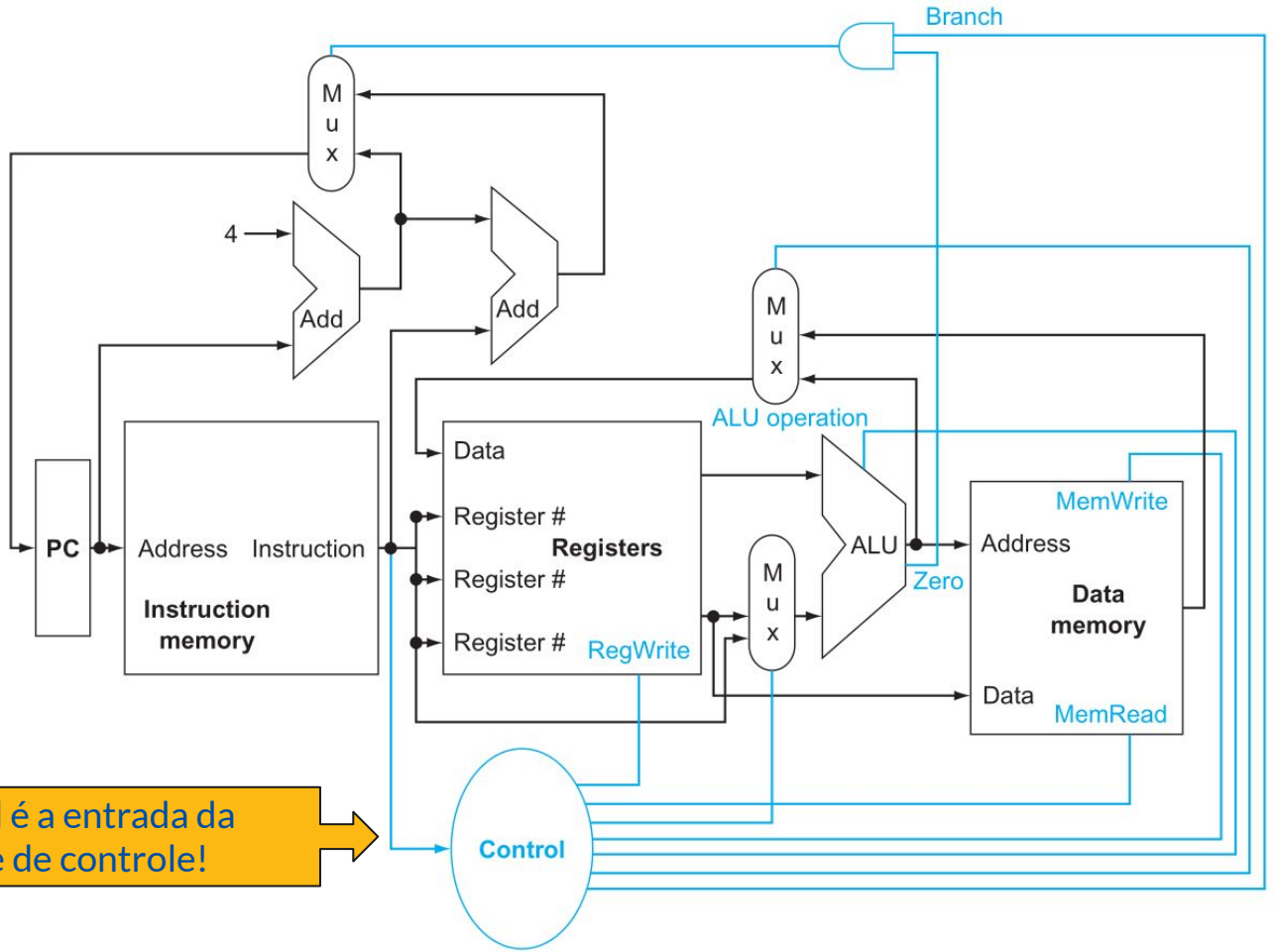
Adicionamos elementos de controle para:

Controlar a próxima instrução usando um indicador de saída 0 da ALU combinado com o sinal de branch do controle;

Habilitar a leitura e escrita da memória de dados e dos registradores;

Controlar a operação da ALU;

Escolher a origem do dado do registrador.



Olha qual é a entrada da unidade de controle!

Falar é fácil, mostre-me o código!

Calma! Antes, precisamos dar sequência ao conteúdo na próxima aula. Não perca. ;-)

Considerações gerais sobre a estrutura

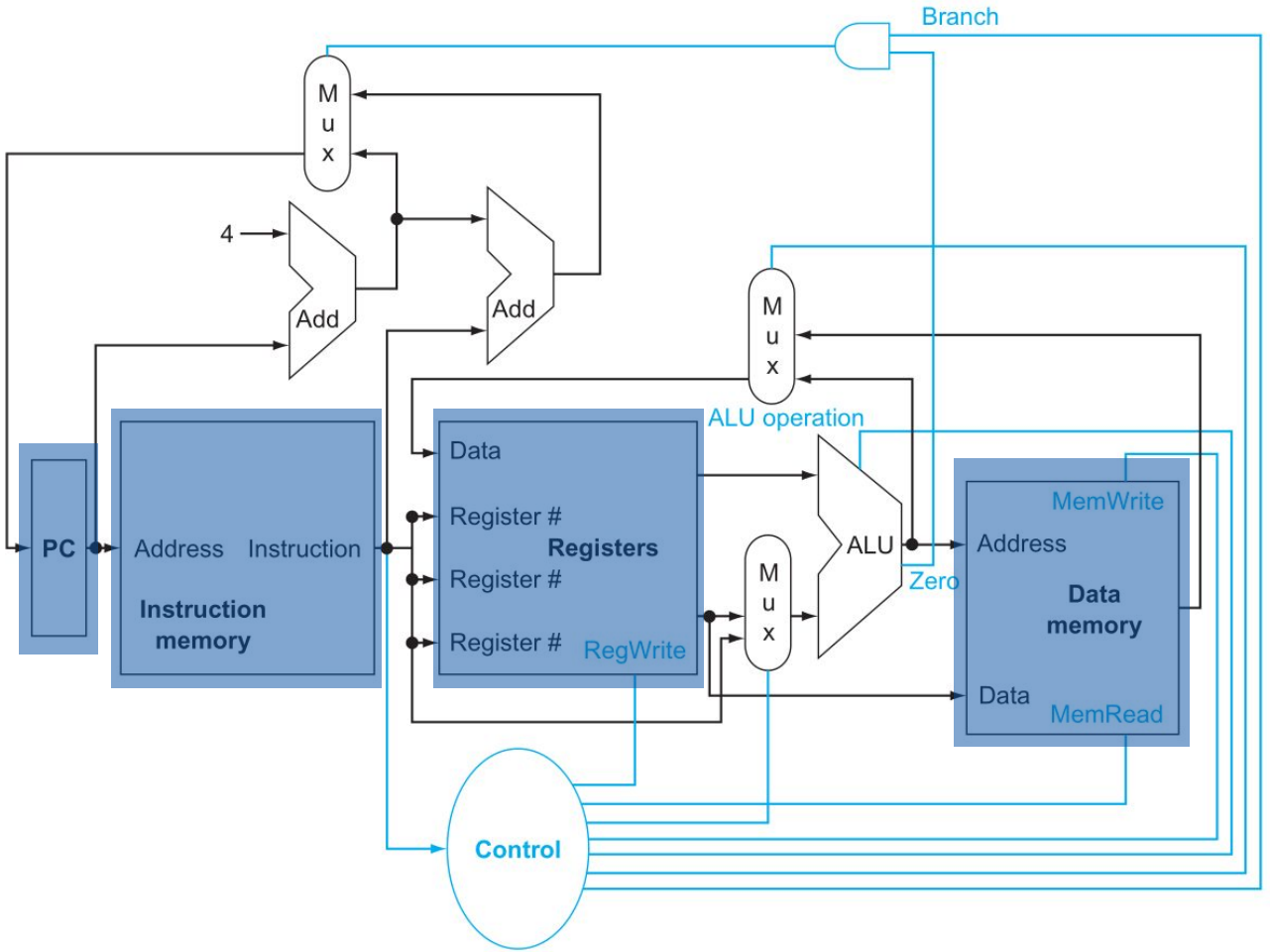
Algumas considerações

Elementos combinacionais e elementos de estado

O processador é composto por elementos combinacionais: ALU, multiplexadores, somadores...

E elementos de estado (sequenciais): registradores, memória de instrução e de dados.

Lembram da diferença?
Combinacional vs. Sequencial



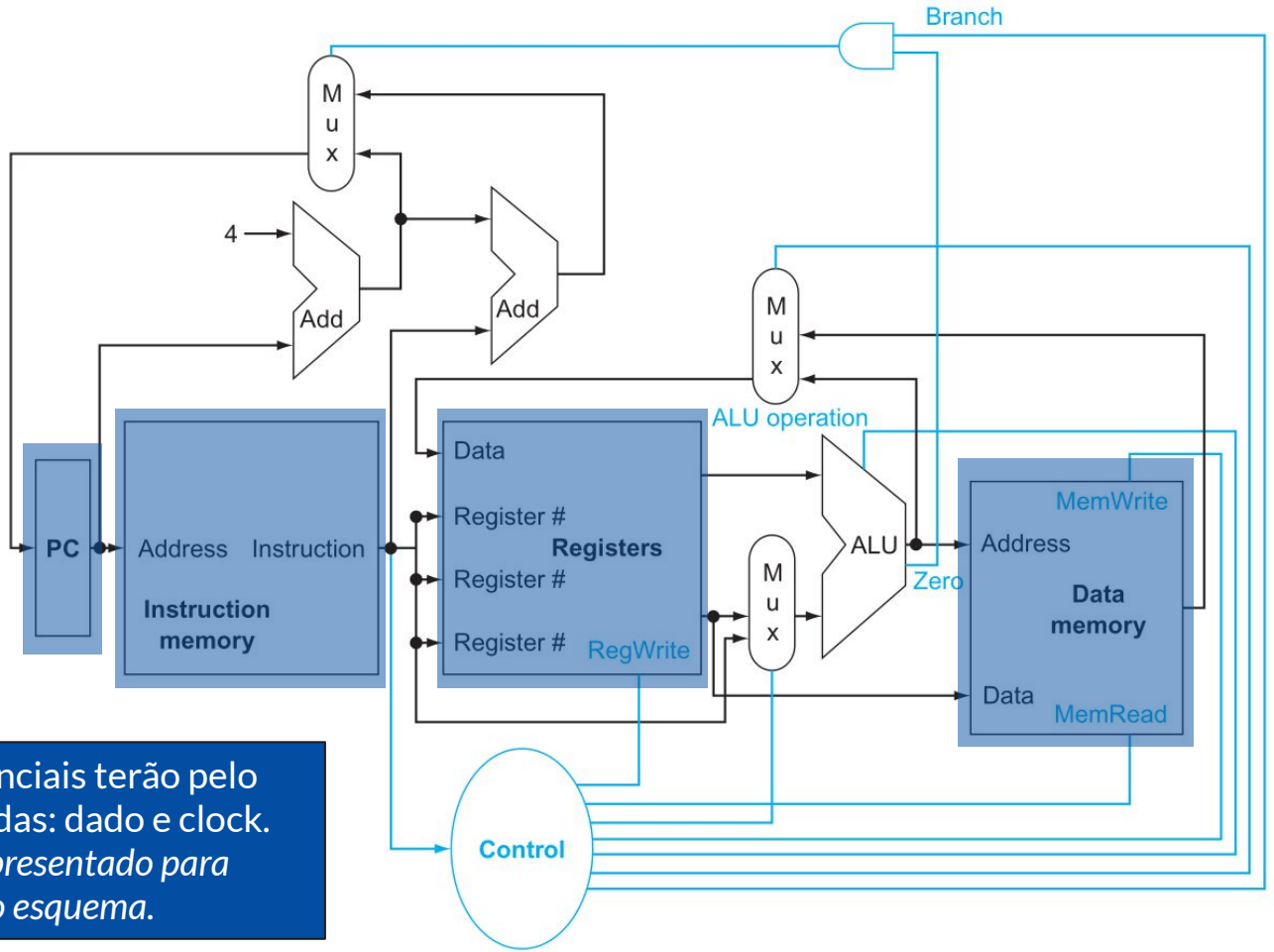
Algumas considerações

Elementos combinacionais e elementos de estado

O processador é composto por elementos combinacionais: ALU, multiplexadores, somadores...

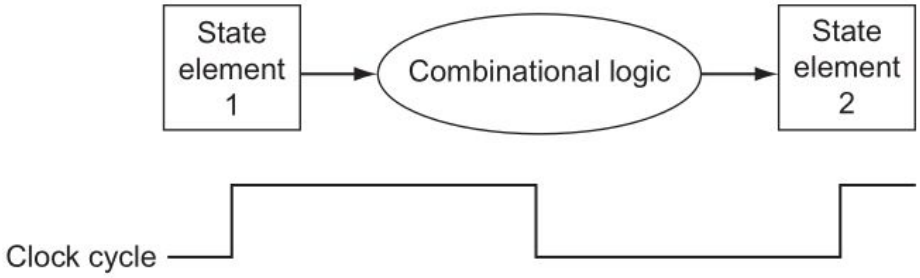
E elementos de estado (sequenciais): registradores, memória de instrução e de dados.

Lembram da diferença?
Combinacional vs. Sequencial



Elementos sequenciais terão pelo menos duas entradas: dado e clock. *O clock não é representado para simplificar o esquema.*

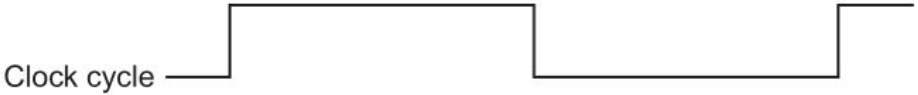
Algumas considerações

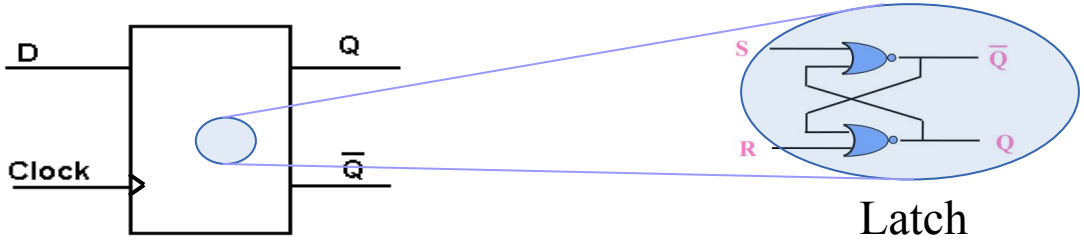


Elementos sequenciais terão pelo menos duas entradas: dado e clock.
Ex.: Flip-Flop tipo D

Algumas considerações

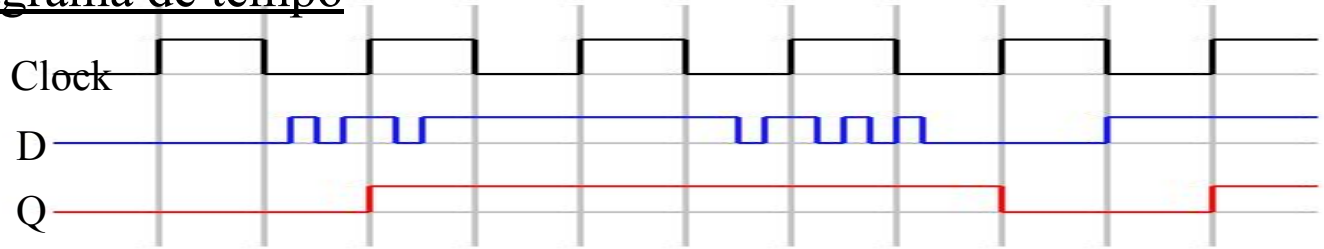
Elementos sequenciais terão pelo menos duas entradas: dado e clock.
Ex.: Flip-Flop tipo D sensível à borda

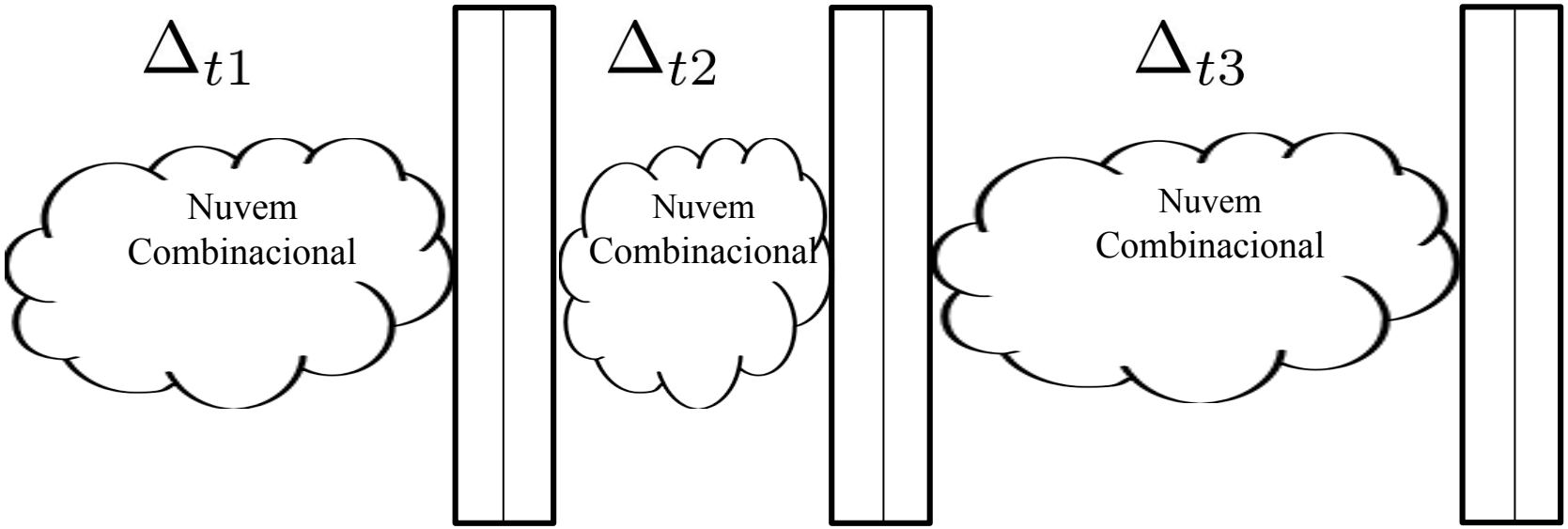




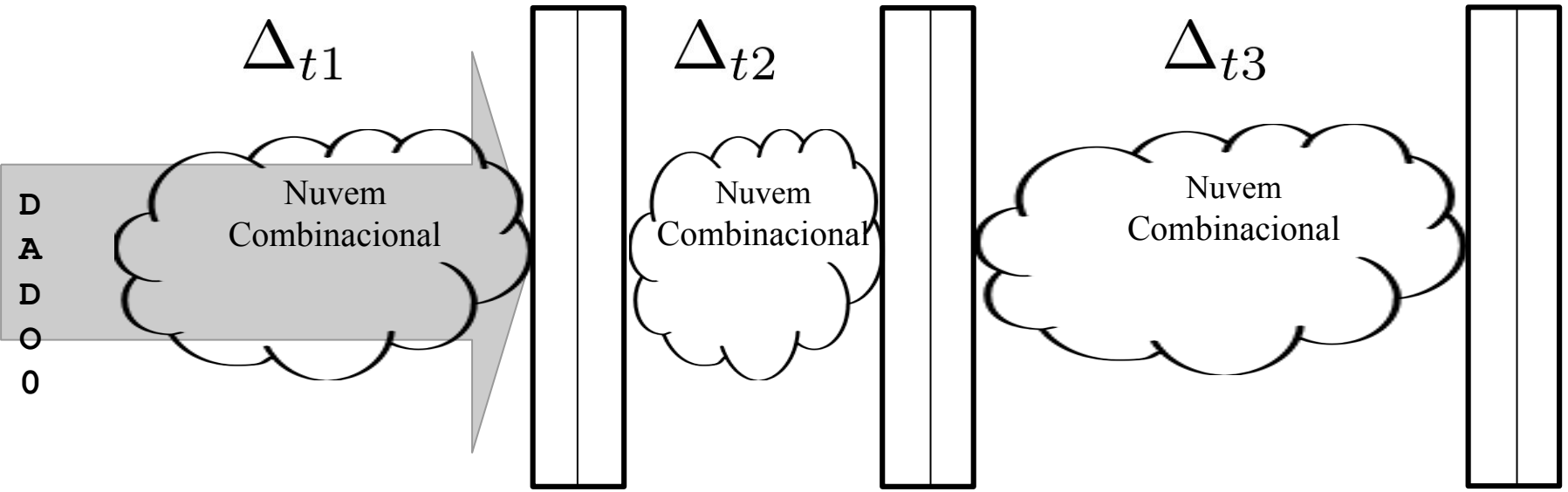
Latch

Diagrama de tempo





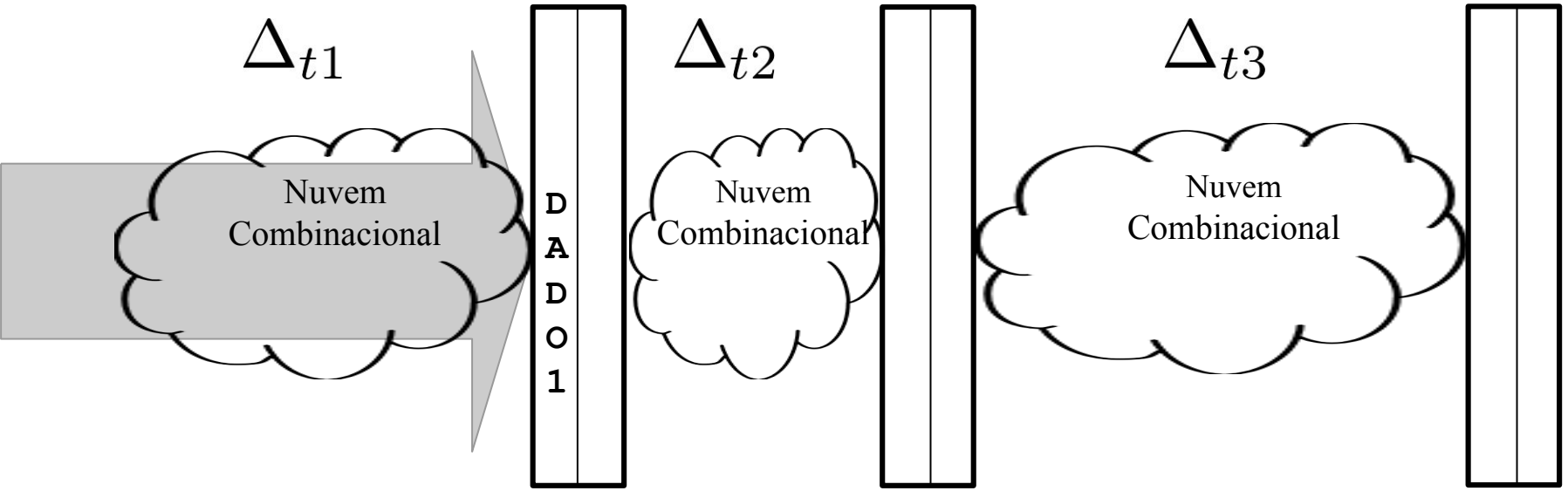
Barreira Temporal (registradores) **Barreira Temporal (registradores)** **Barreira Temporal (registradores)**



Barreira Temporal (registradores)

Barreira Temporal (registradores)

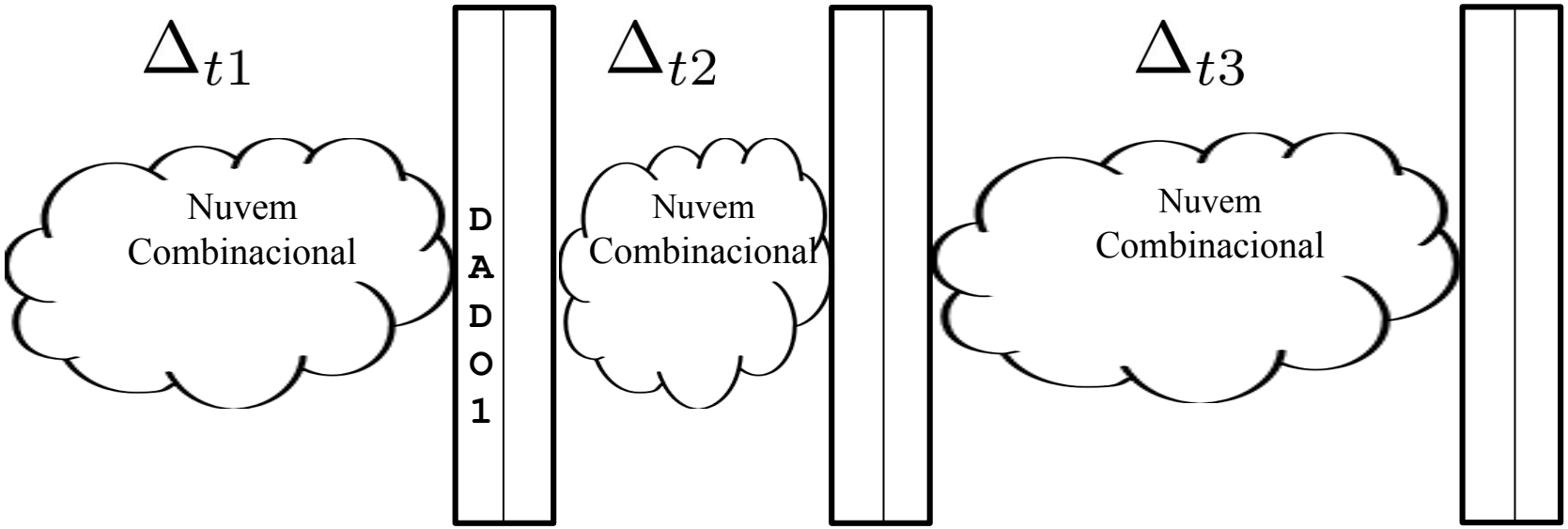
Barreira Temporal (registradores)



Barreira Temporal (registradores)

Barreira Temporal (registradores)

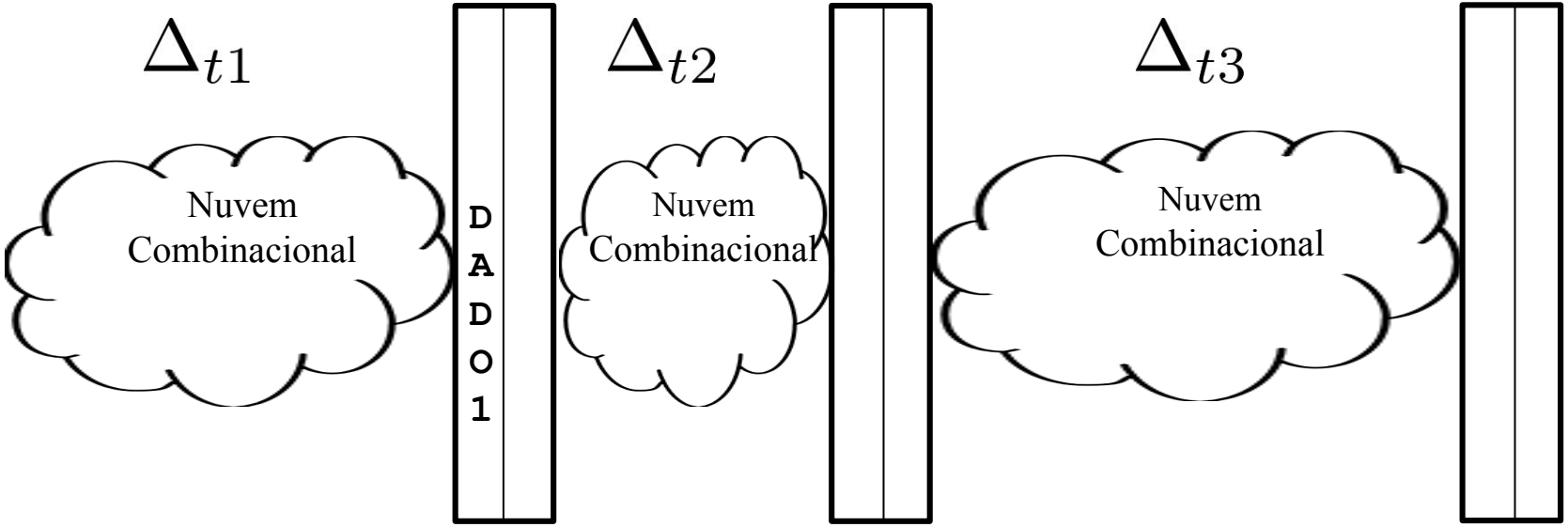
Barreira Temporal (registradores)

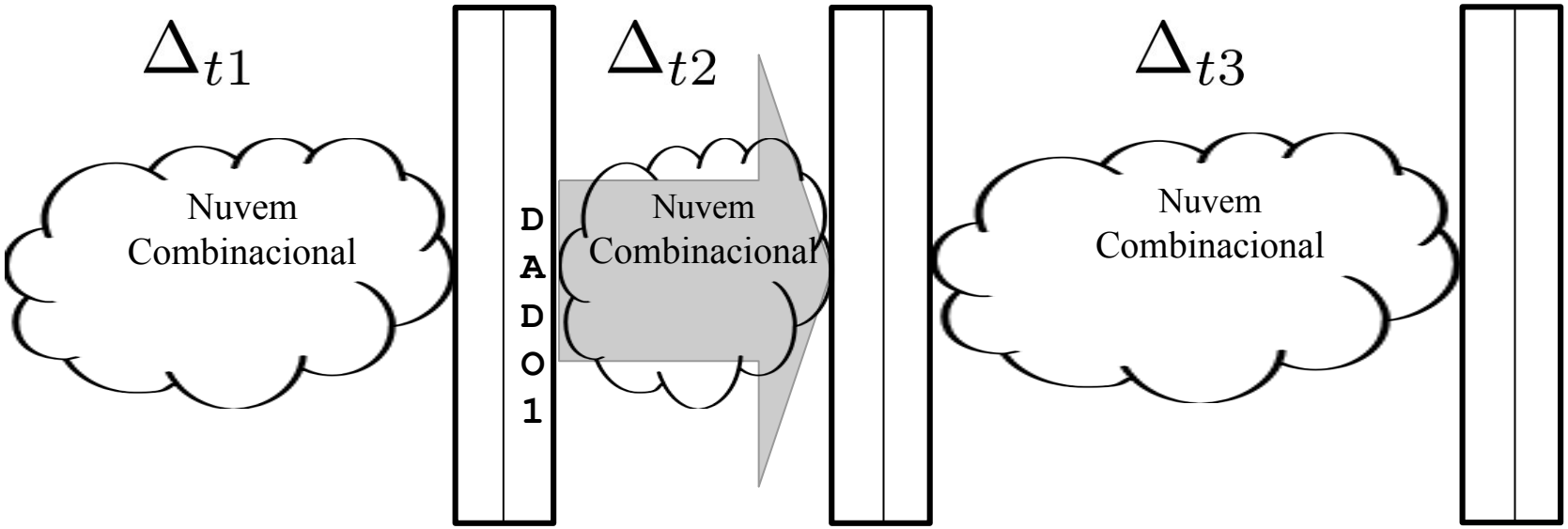


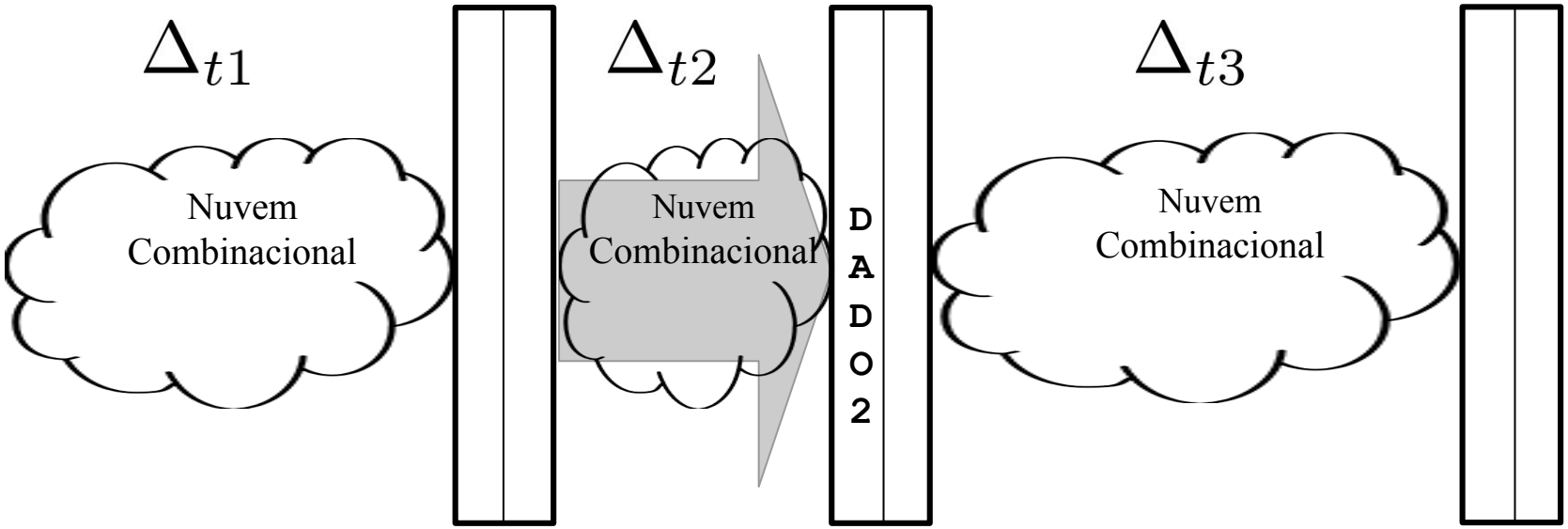
Barreira Temporal (registradores)

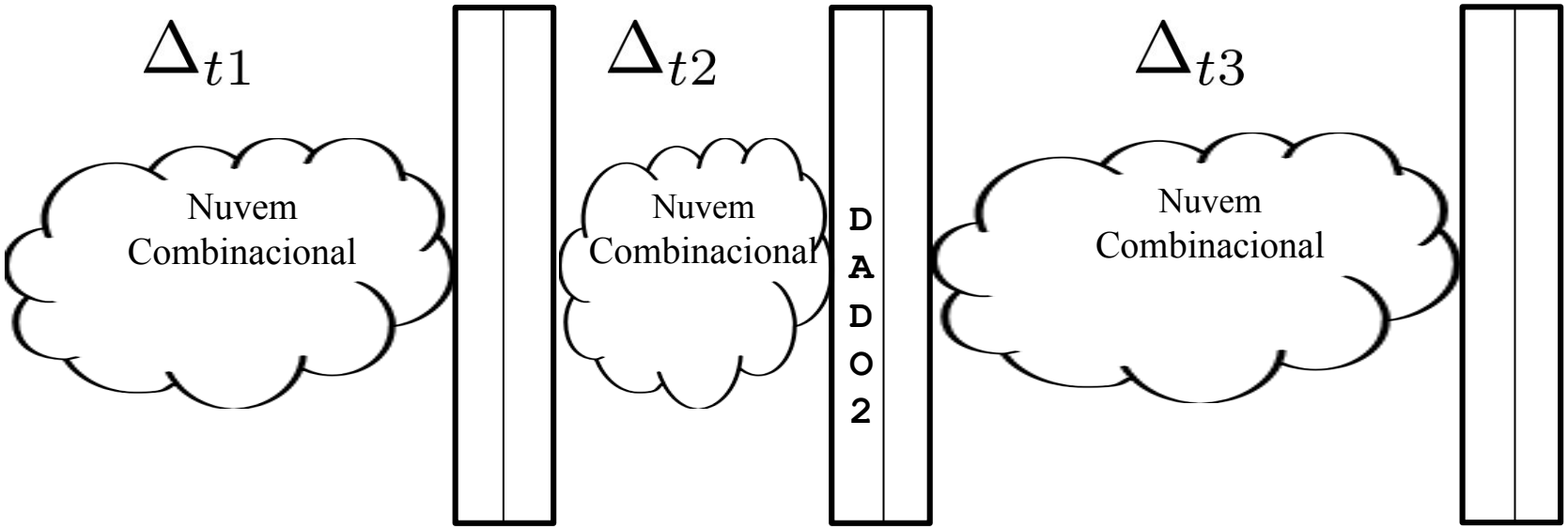
Barreira Temporal (registradores)

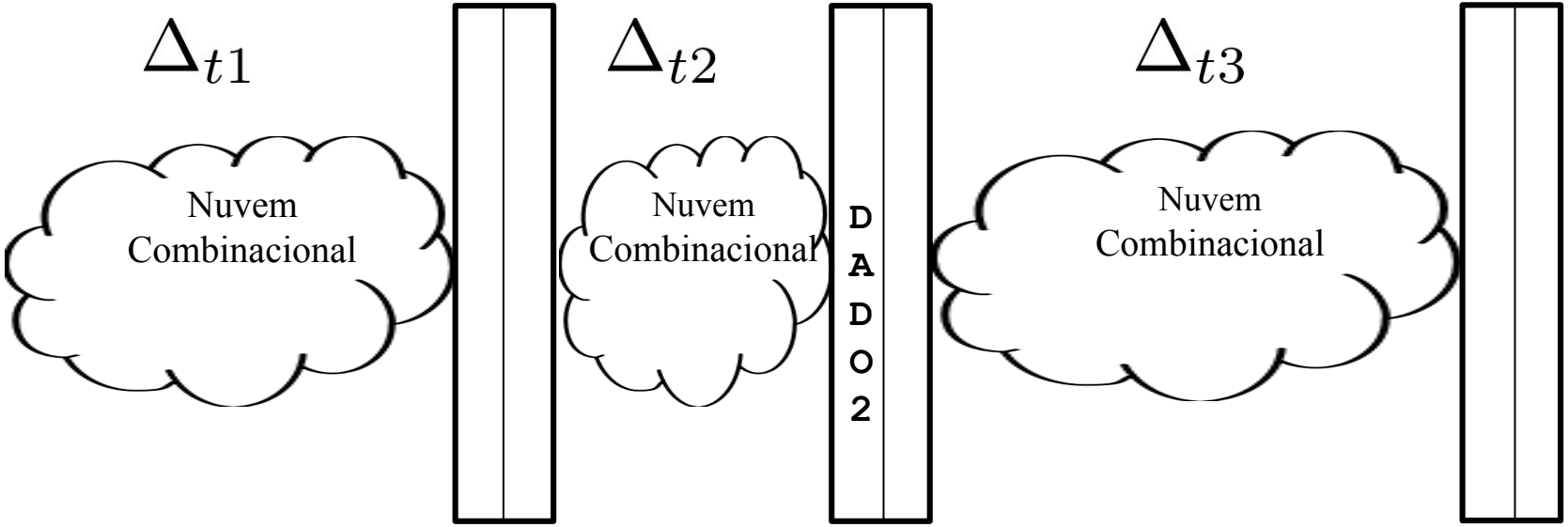
Barreira Temporal (registradores)

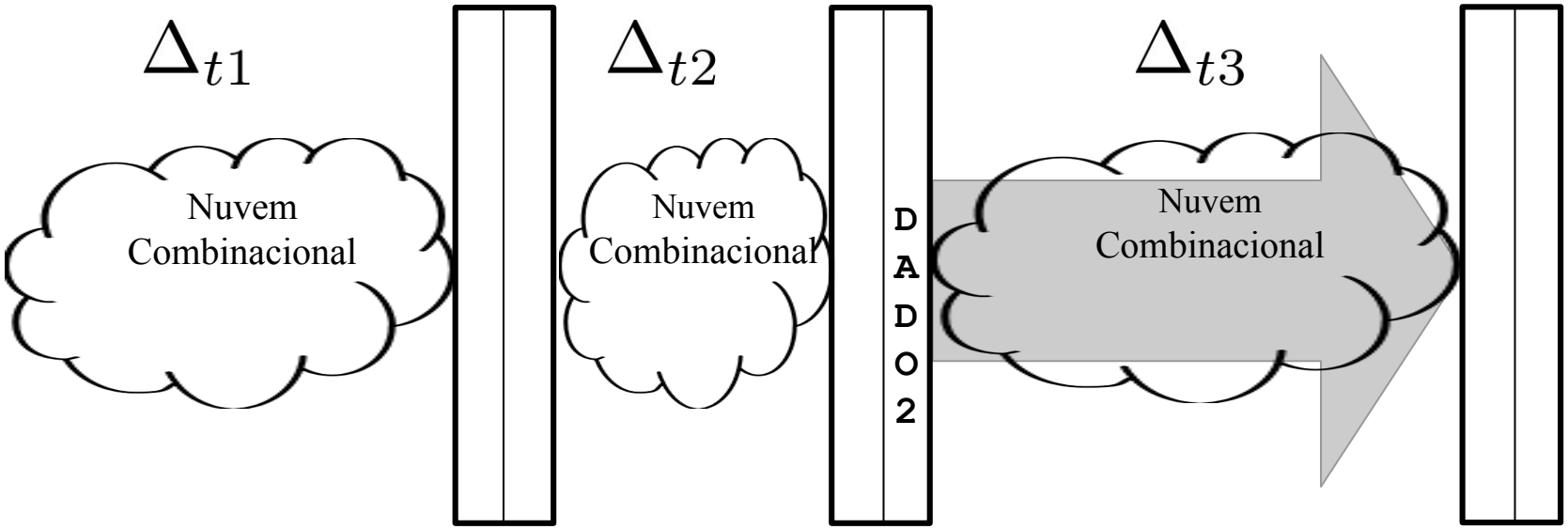

Δ_{t1}
Δ_{t2}
Δ_{t3}
Nuvem Combinacional
DADO1
Nuvem Combinacional
Nuvem Combinacional
Barreira Temporal (registradores)
Barreira Temporal (registradores)
Barreira Temporal (registradores)
Sobe o Clock


Δ_{t1}
Δ_{t2}
Δ_{t3}
Nuvem Combinacional
DADO1
Nuvem Combinacional
Nuvem Combinacional
Barreira Temporal (registradores)
Barreira Temporal (registradores)
Barreira Temporal (registradores)

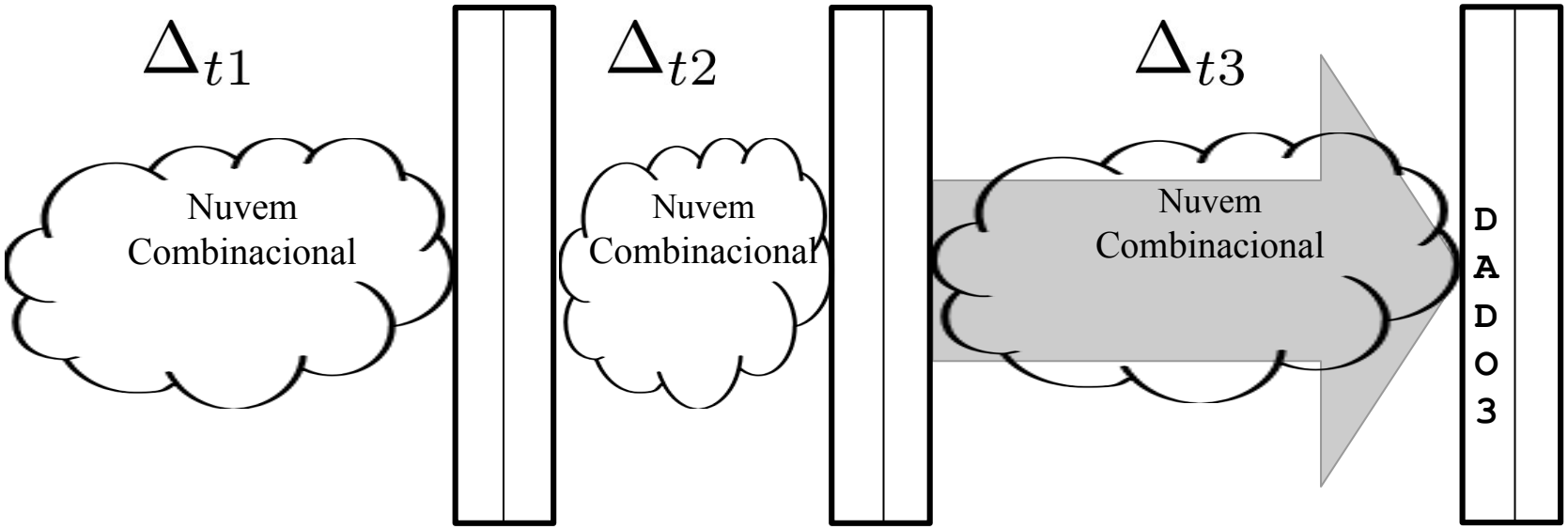

Δ_{t1}
Δ_{t2}
Δ_{t3}
Nuvem Combinacional
Nuvem Combinacional
Nuvem Combinacional
DADO2
Barreira Temporal (registradores)
Barreira Temporal (registradores)
Barreira Temporal (registradores)


Δ_{t1}
Δ_{t2}
Δ_{t3}
Nuvem Combinacional
Nuvem Combinacional
Nuvem Combinacional
DADO2
Barreira Temporal (registradores)
Barreira Temporal (registradores)
Barreira Temporal (registradores)

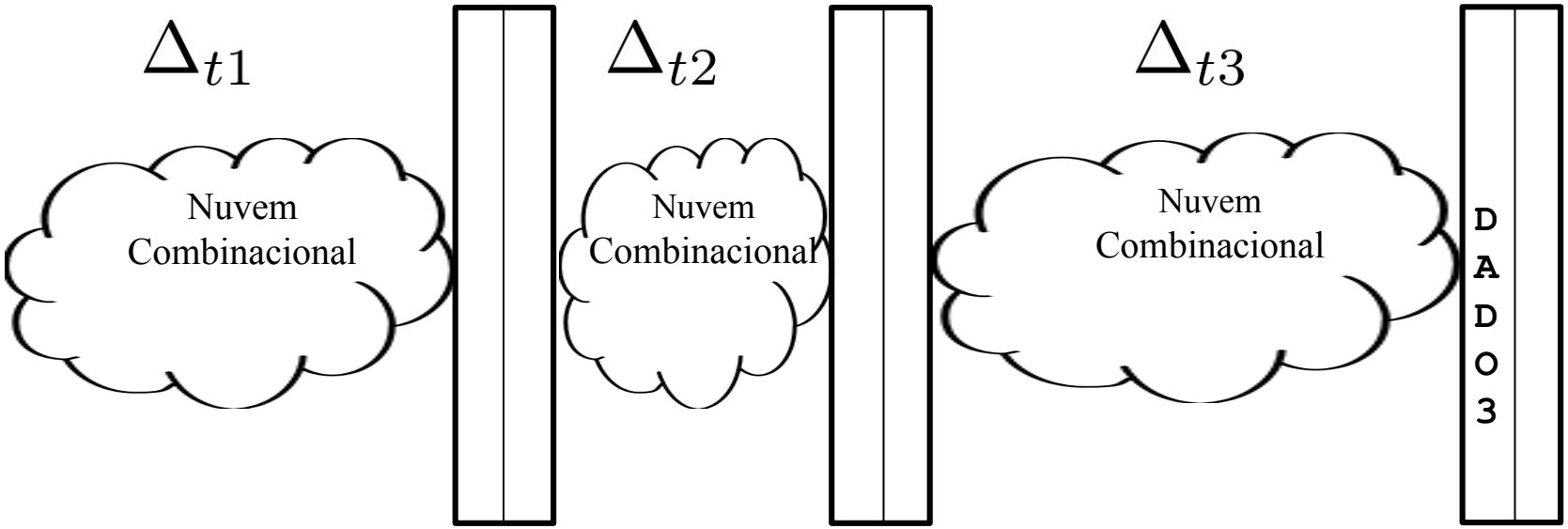

Δ_{t1}
Δ_{t2}
Δ_{t3}
Nuvem Combinacional
Nuvem Combinacional
Nuvem Combinacional
DADO2
Barreira Temporal (registradores)
Barreira Temporal (registradores)
Barreira Temporal (registradores)
Sobe o Clock



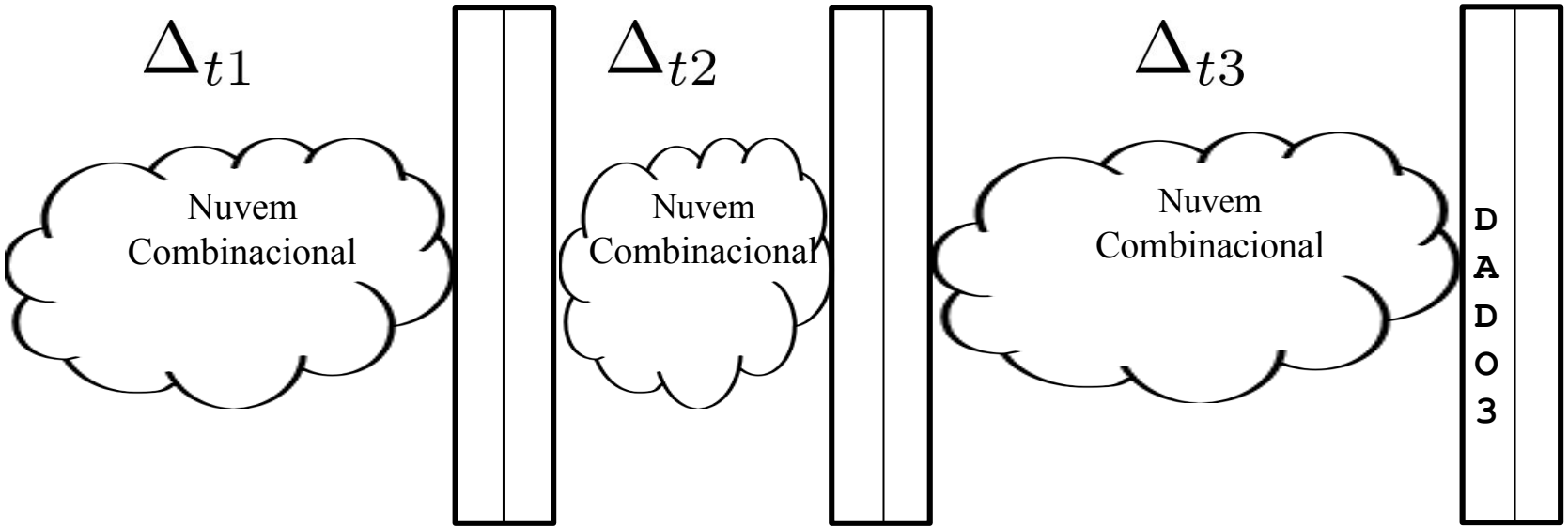
Barreira Temporal (registradores) **Barreira Temporal (registradores)** **Barreira Temporal (registradores)**



Barreira Temporal (registradores) **Barreira Temporal (registradores)** **Barreira Temporal (registradores)**



Barreira Temporal (registradores) **Barreira Temporal (registradores)** **Barreira Temporal (registradores)**

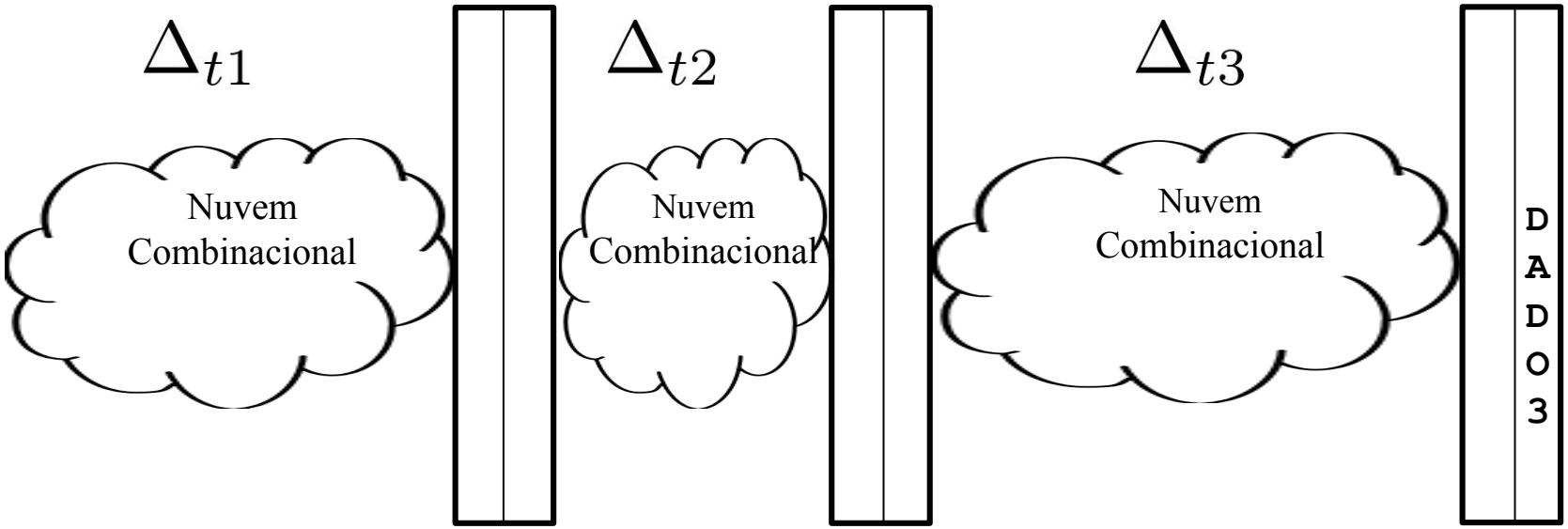


Barreira Temporal (registradores)

Barreira Temporal (registradores)

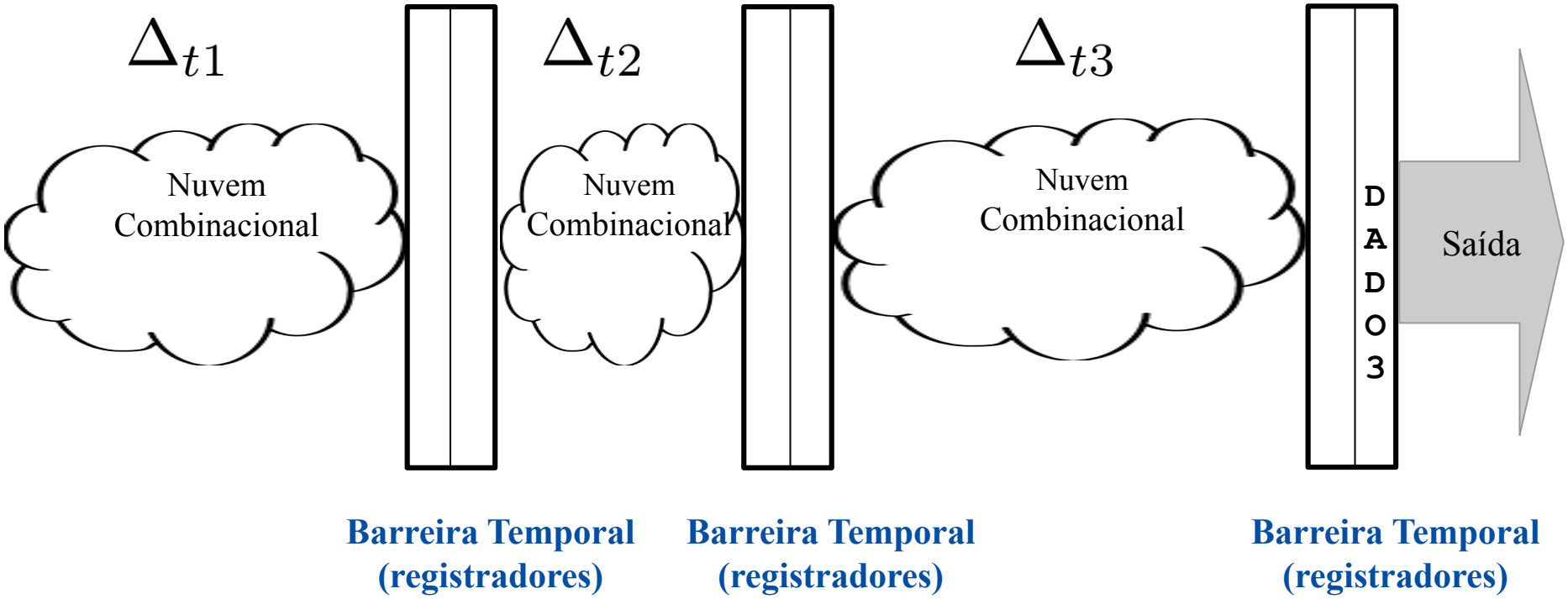
Barreira Temporal (registradores)

Sobe o Clock



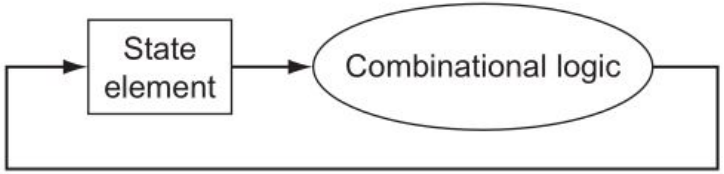
Barreira Temporal (registradores) **Barreira Temporal (registradores)** **Barreira Temporal (registradores)**

Em um sistema digital...

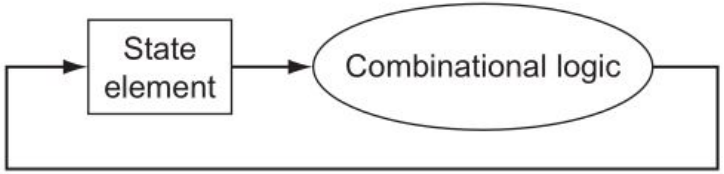


Nota: O sinal de clock é o mesmo para todo o sistema.

Em um processador MIPS de ciclo único

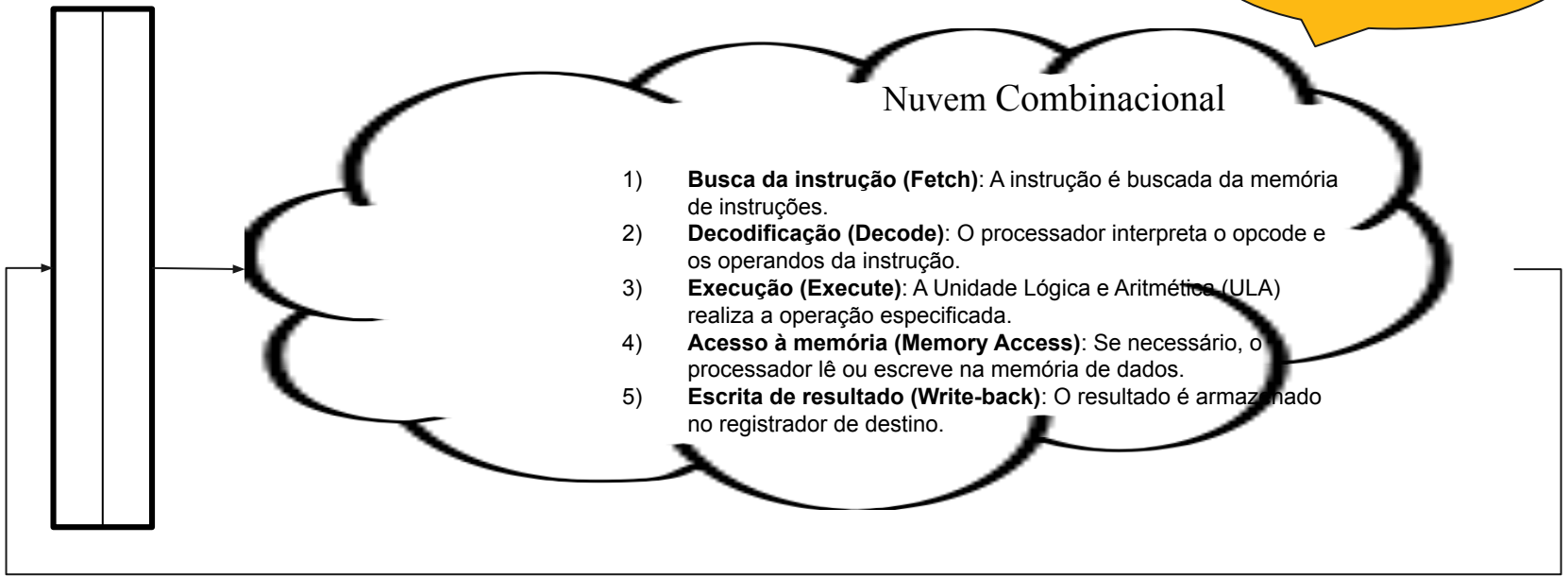


Em um processador MIPS de ciclo único



Barreira Temporal (registradores)

Em um só ciclo de relógio



Se as atividades são dependentes, como posso ter só um clock?

Se as atividades são dependentes, como posso ter só um clock?

Posso fazer a leitura independente do sinal de clock.

Como fica o código verilog?

```
// MEMORIA DE INSTRUCAO
module instr_mem(
    input [5:0] addr,
    output [31:0] instr
    );

    reg [31:0] RAM[63:0];
    // inicializacao com instrucoes
    assign instr=RAM[addr];
endmodule
```

```
// MEMORIA DE DADOS
module dmem(
    input clk,memwrite,
    input [31:0] addr,writedata,
    output [31:0] readdata
    );
    reg [31:0] RAM[63:0];//32x64 RAM
    // le dado da memoria
    assign readdata=RAM[addr[31:2]];
    // escreve dado na memoria
    always@(posedge clk)
    if(memwrite)
        RAM[addr[31:2]]<=writedata;
endmodule
```

Construindo o Datapath

Enfim, diagramas + código verilog.

Referências

PATTERSON, D. A., HENNESSY, J. L. Computer Organization and Design: The Hardware/Software Interface, 5th Edition, 2014

Hora-Trabalho de Hoje

Leia o capítulo 4 do livro PATTERSON & HENNESSY. Computer Organization and Design: The Hardware/Software Interface.

Seções 4.3.

Dúvidas?

Na próxima aula...

Continuaremos a detalhar o processador MIPS;

Não falte! 😉

Obrigado pela atenção

UNIVERSIDADE
FEDERAL DE
SERGIPE