



**DEEC**

DEPARTAMENTO DE ENGENHARIA  
ELETROTÉCNICA E DE COMPUTADORES

TÉCNICO LISBOA

# ***Arquitectura de Computadores***

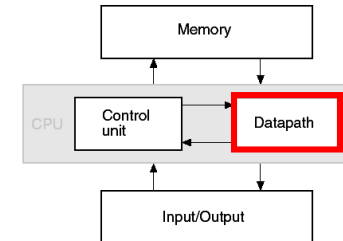
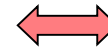
## ***MEEC (2016/17 – 2º Sem.)***

### ***Unidade de Processamento***

**Prof. Nuno Horta**

# PLANEAMENTO

- ❑ *Introdução*
- ❑ ***Unidade de Processamento***
- ❑ *Unidade de Controlo*
- ❑ *Arquitectura do Conjunto de Instruções*
- ❑ *Unidade Central de Processamento (CPU)*
- ❑ *Unidade de Entrada/Saída (I/O)*
- ❑ *Unidade de Memória*
- ❑ *Perspectiva Evolutiva das Arquitecturas de Computadores*



# SUMÁRIO

## Unidade de Processamento

### Introdução

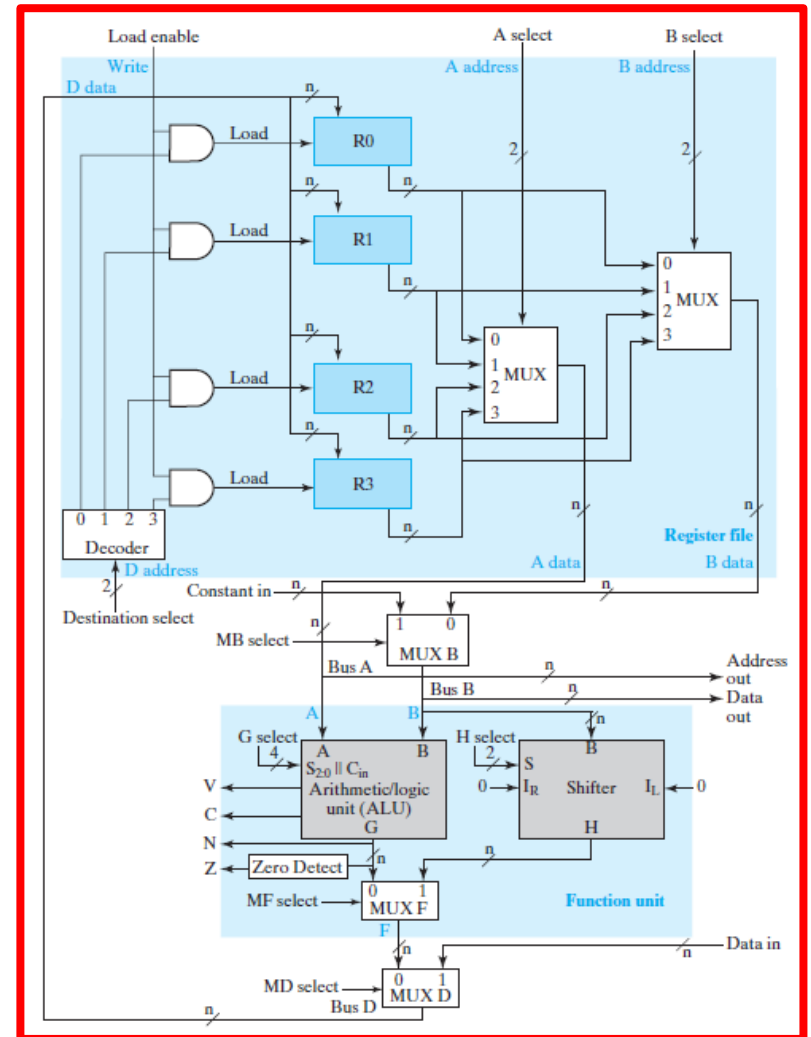
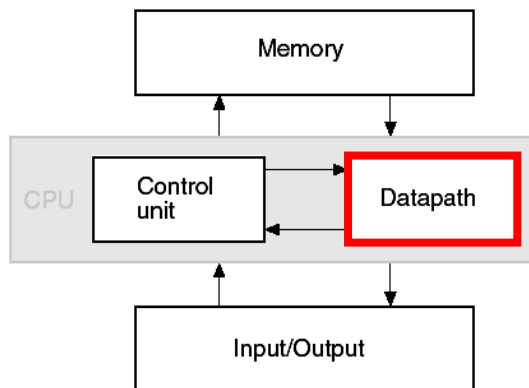
### Unidade de Armazenamento

### Unidade Funcional

### Palavra de Controlo

### Temporizações

### Projeto em VIVADO®

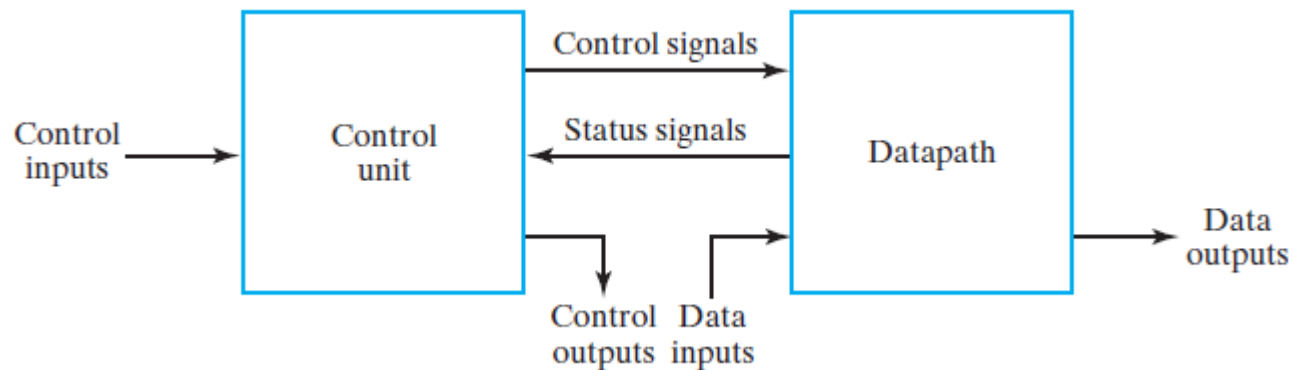


# UNIDADE DE PROCESSAMENTO

*Sistemas Digitais Complexos: Unidade de Processamento + Unidade de Controlo*

*Unidade de Processamento (Datapath): Módulo responsável pela **execução das operações** de processamento de dados.*

*Unidade de Controlo: Módulo responsável pelo **controlo da sequência de operações** a executar na Datapath para implementação de uma tarefa.*



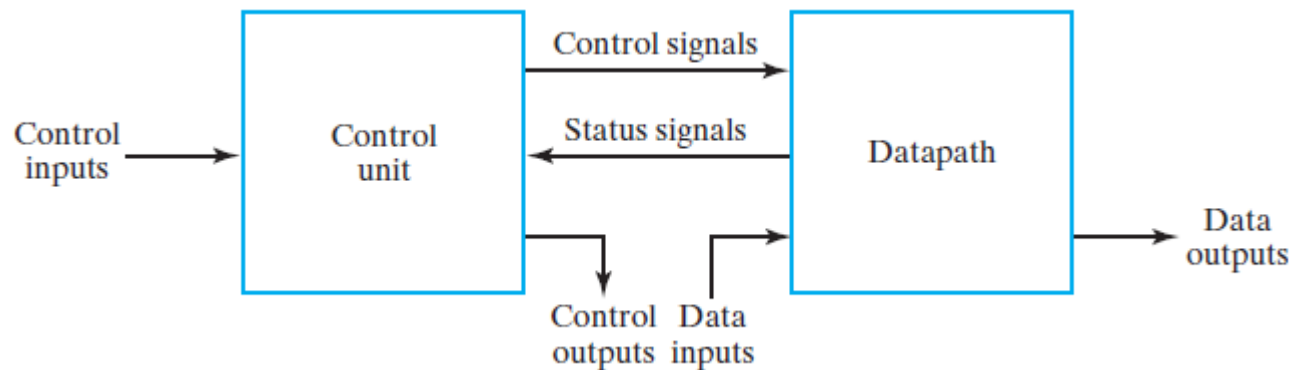
# UNIDADE DE PROCESSAMENTO

**Datapath:** Caracteriza-se pelo **conjunto de registos** e pelo **conjunto de operações** realizado sobre os dados armazenados nos registos.

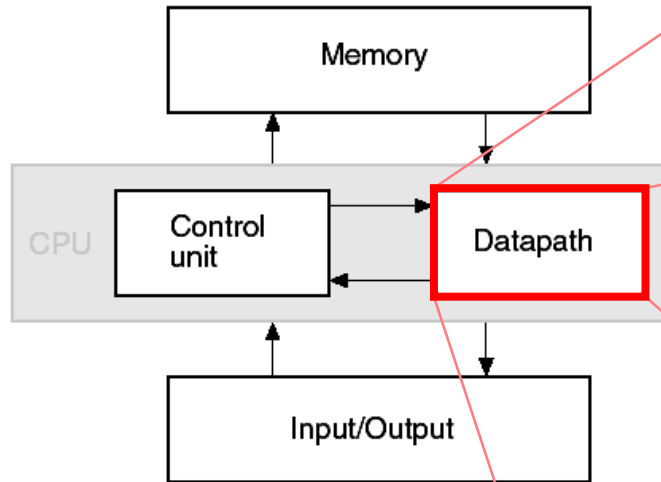
**Operações Elementares sobre Registos:** Shift (Deslocamento), Clear (Apagar), Load (Carregamento), Incrementar, Decrementar, Somar, Subtrair, etc.

**Microoperações (Aritméticas, Lógicas, Deslocamento):** Operações elementares aplicadas sobre os dados em registos.

**Unidade de Controlo:** Fornece os sinais que permitem sequenciar as microoperações de um modo definido, e.g., sequência do conjunto de operações para realizar uma multiplicação.

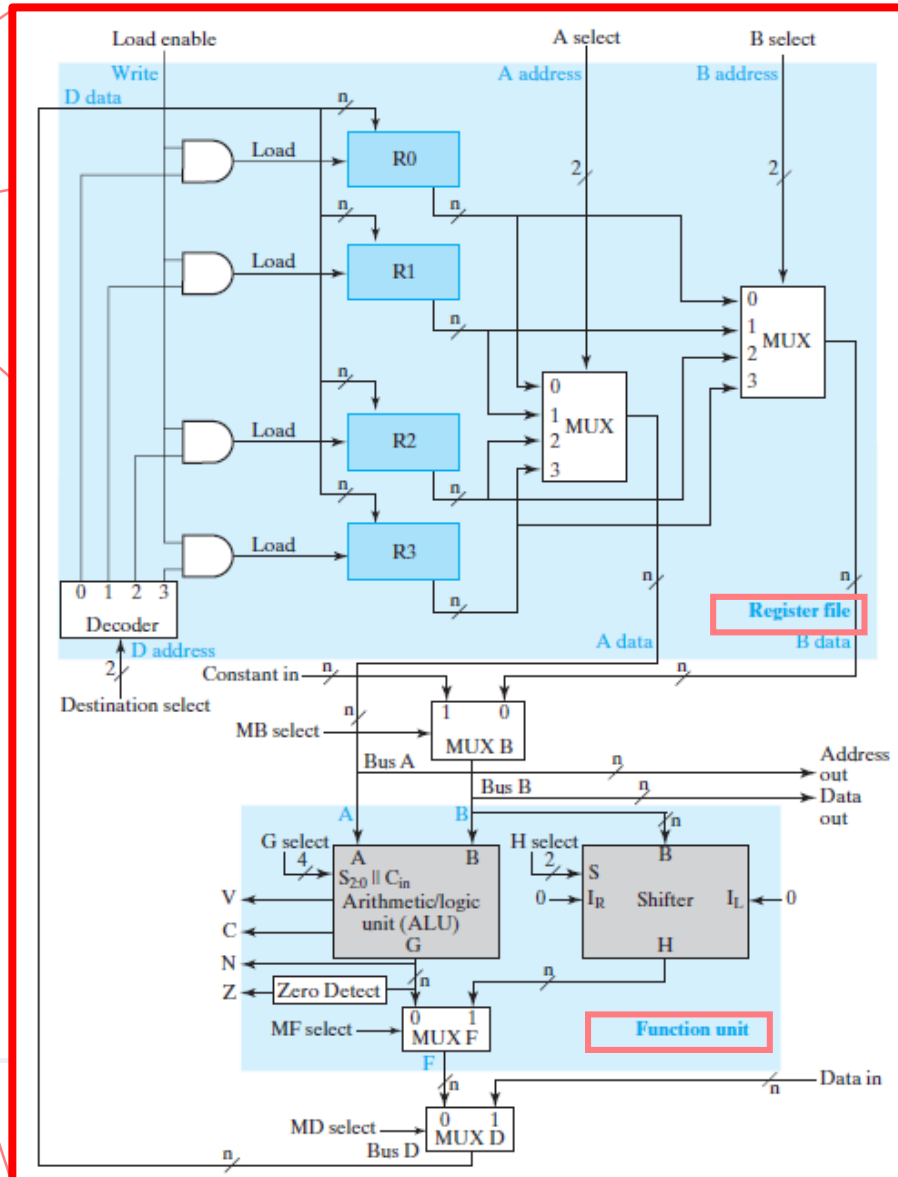


# UNIDADE DE PROCESSAMENTO



**Datapath:** Caracteriza-se pelo conjunto de registos e lógica de seleção (**Unidade de Armazenamento**)

e pela lógica para implementação de microoperações e lógica de seleção (**Unidade Funcional**) a realizar sobre os dados armazenados nos registos.



# UNIDADE DE PROCESSAMENTO

## Datapath (Exemplo)

Unidade de Armazenamento

Unidade Funcional

ALU: Unidade Lógica e Aritmética

Shifter: Unidade de Deslocamento

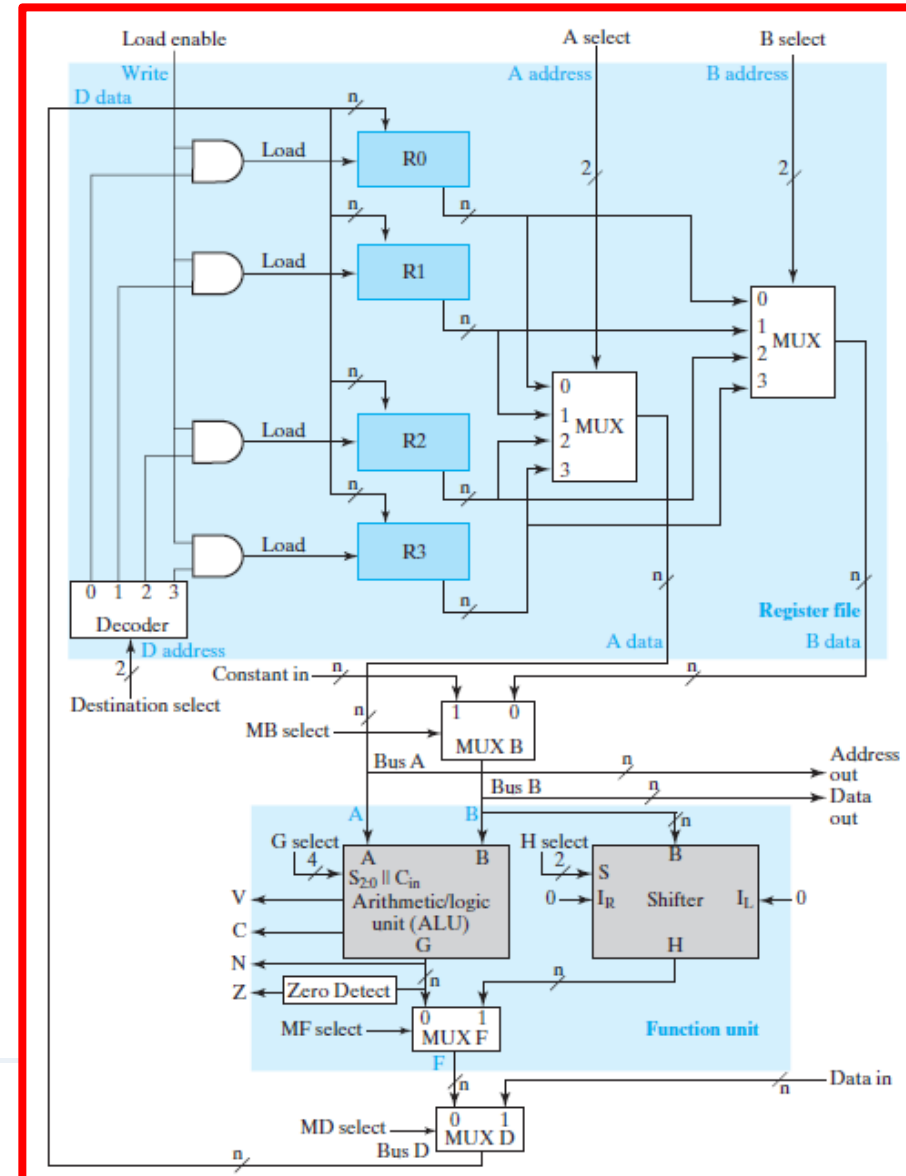
## (Exemplo de Operação)

$$R0 \leftarrow R1 + R2$$

## Entradas de Controlo (da U. de Controlo)

- Selecção de A
- Selecção de B
- Selecção de G
- Selecção de H
- Selecção de MB
- Selecção de MF
- Selecção de MD
- Selecção do Destino
- Carregamento do Resultado

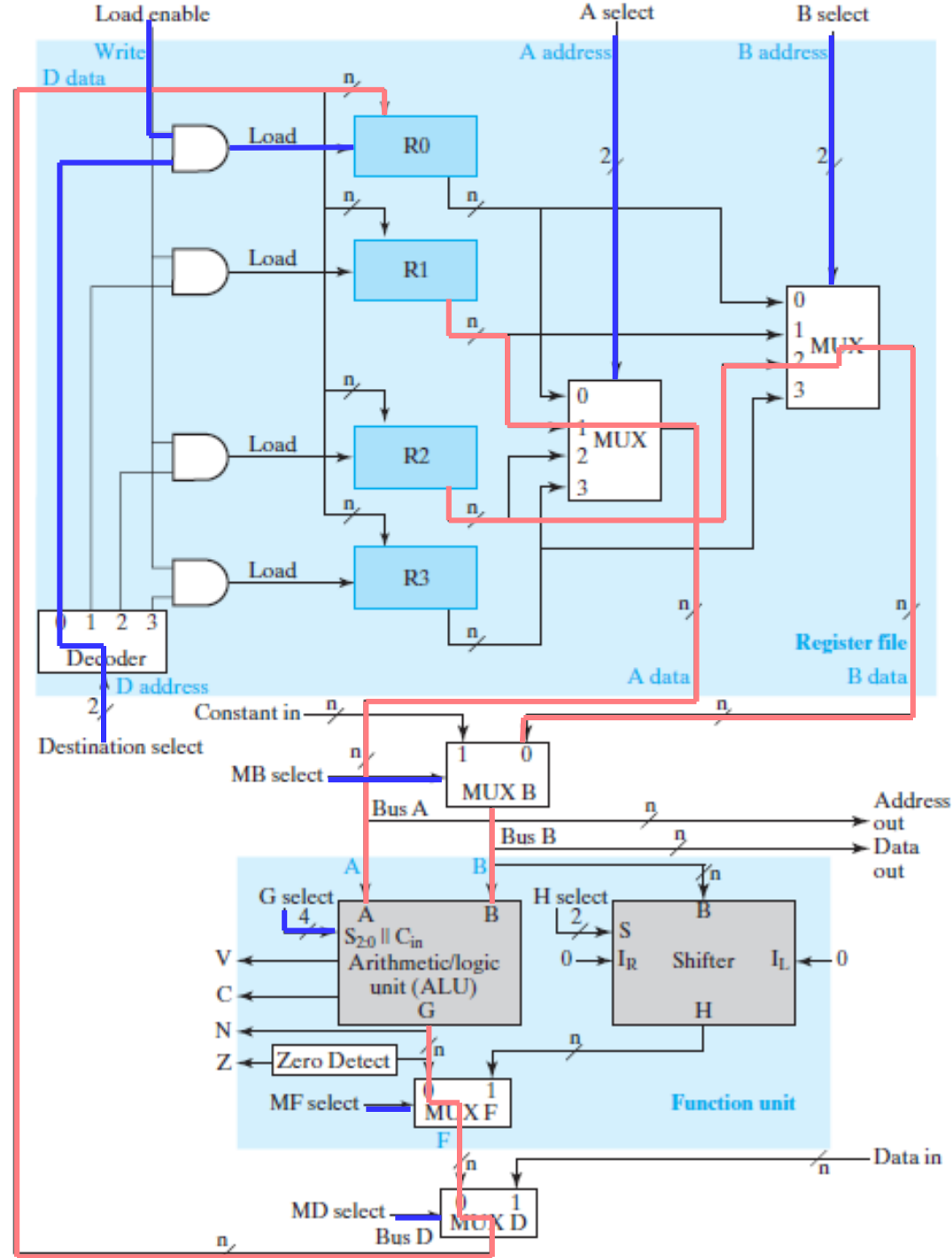
Operação realizada num 1 ciclo de relógio  
(Cálculo e Carregamento)



# UNIDADE DE PROCESSAMENTO

*Exemplo de Operação:*

$$R0 \leftarrow R1 + R2$$

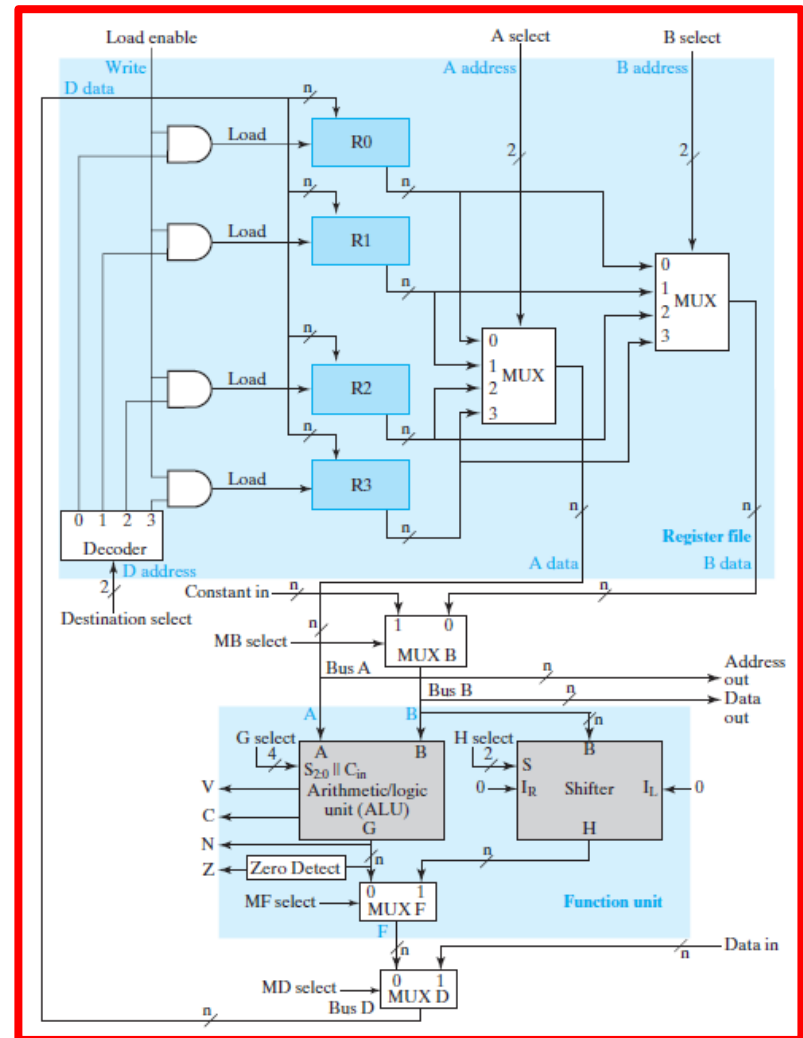
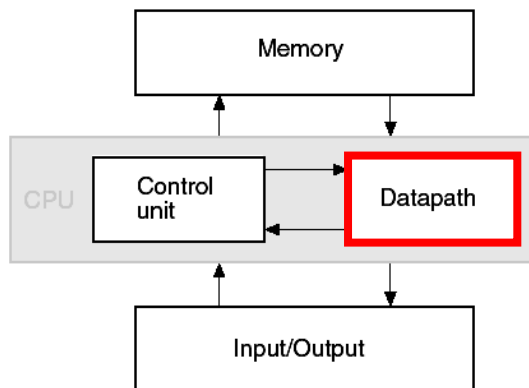




# SUMÁRIO

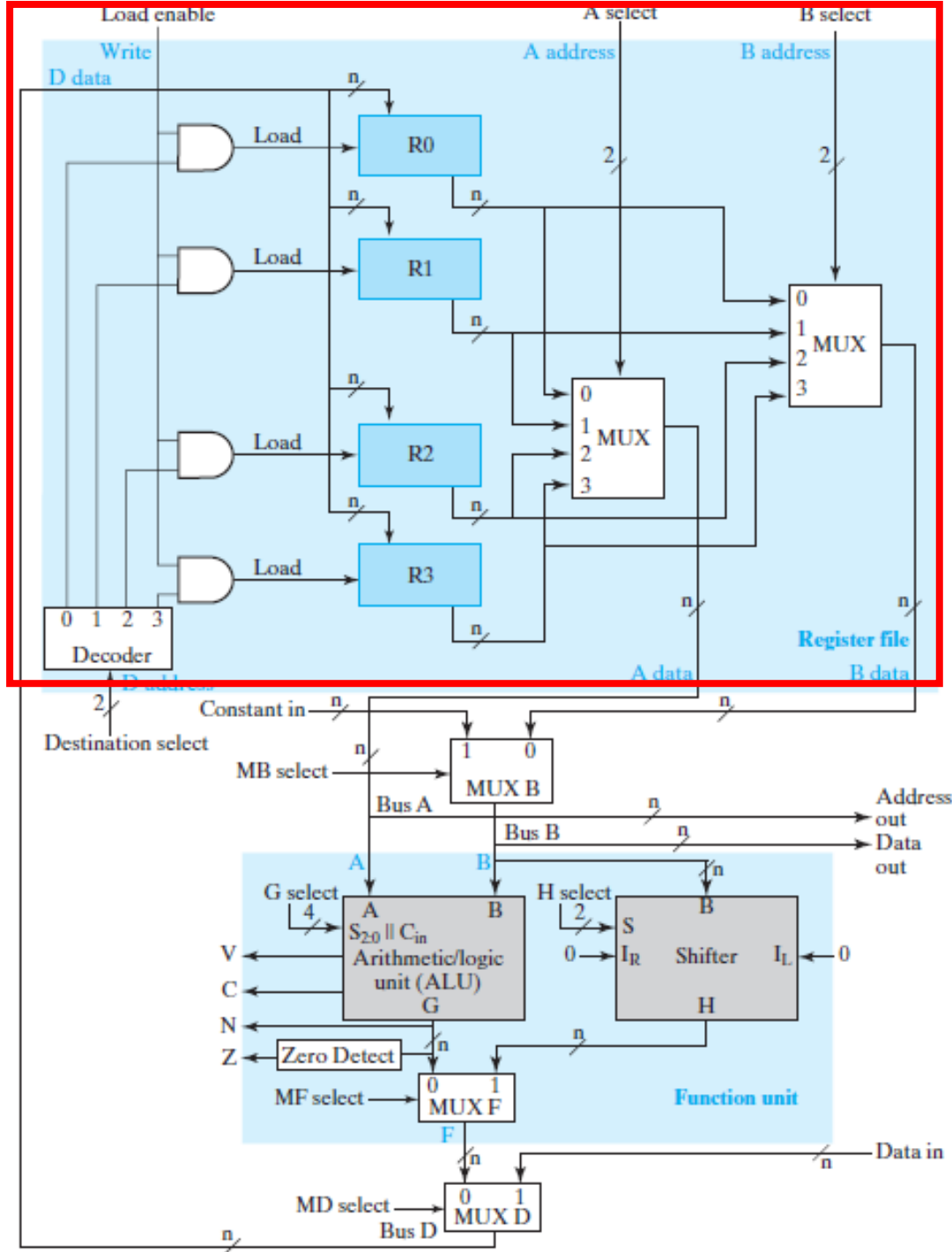
## Unidade de Processamento

- Introdução
- Unidade de Armazenamento**
- Unidade Funcional
- Palavra de Controlo
- Temporizações
- Projeto em VIVADO®



# UNIDADE DE PROCESSAMENTO

*Unidade  
de Armazenamento:*



# UNIDADE DE PROCESSAMENTO

## Unidade de Armazenamento

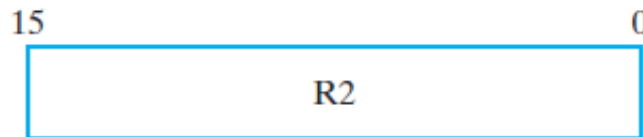
**Registo:** Elemento básico de memória que permite armazenar um conjunto de  $N$  bits (dimensão do registo).



(a) Register R



(b) Individual bits of 8-bit register



(c) Numbering of 16-bit register



(d) Two-part 16-bit register

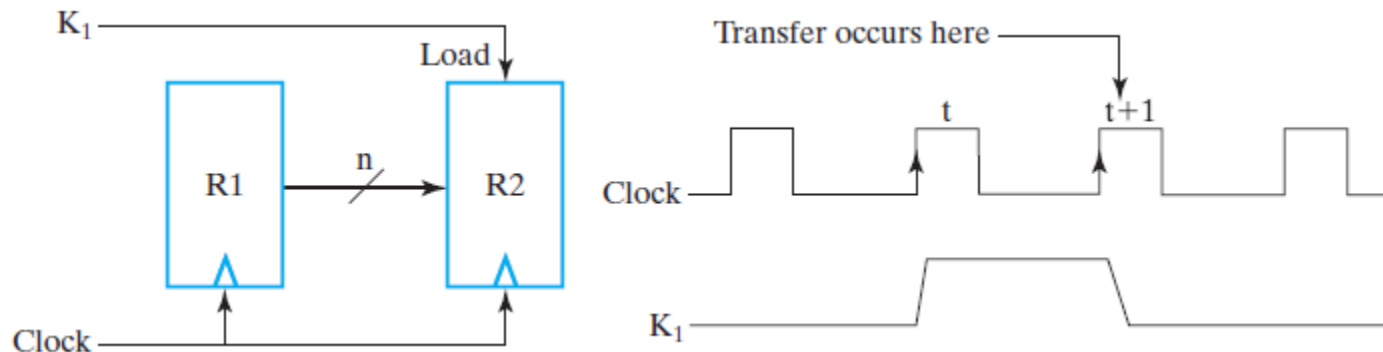
# UNIDADE DE PROCESSAMENTO

## *Unidade de Armazenamento*

*Operação de Transferência de Dados entre 2 Registos:*

if ( $K_1 = 1$ ) then ( $R2 \leftarrow R1$ )

$K_1: R2 \leftarrow R1$



# UNIDADE DE PROCESSAMENTO

## Unidade de Armazenamento

### *Simbologia: (Transferências de Registos em RTL e VHDL)*

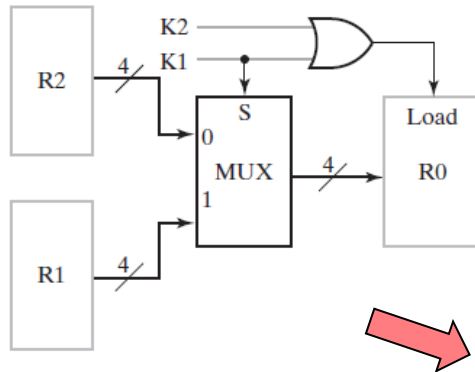
Symbol	Description	Examples
Letters (and numerals)	Denotes a register	$AR, R2, DR, IR$
Parentheses	Denotes a part of a register	$R2(1), R2(7:0), AR(L)$
Arrow	Denotes transfer of data	$R1 \leftarrow R2$
Comma	Separates simultaneous transfers	$R1 \leftarrow R2, R2 \leftarrow R1$
Square brackets	Specifies an address for memory	$DR \leftarrow M[AR]$

Operation	Text RTL	VHDL	Verilog
Combinational assignment	=	<= (concurrent)	assign = (nonblocking)
Register transfer	←	<= (concurrent)	<= (nonblocking)
Addition	+	+	+
Subtraction	−	−	−
Bitwise AND	∧	and	&
Bitwise OR	∨	or	
Bitwise XOR	⊕	xor	^
Bitwise NOT	− (overline)	not	~
Shift left (logical)	Sl	sll	<<
Shift right (logical)	Sr	srl	>>
Vectors/registers	A(3:0)	A(3 down to 0)	A[3:0]
Concatenation		&	{ , }

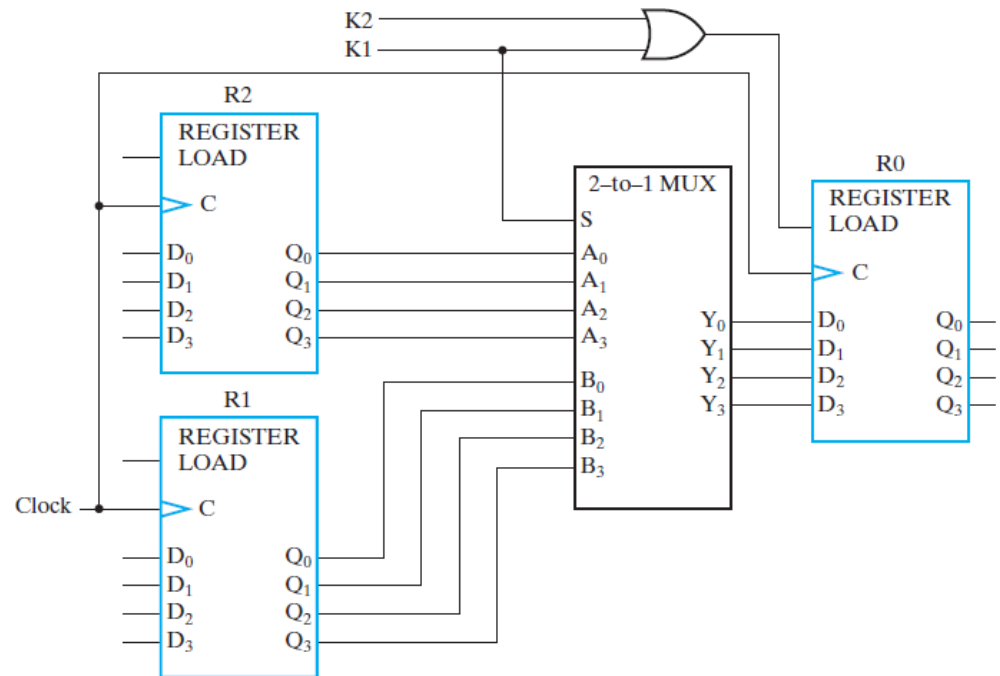
# UNIDADE DE PROCESSAMENTO

## Unidade de Armazenamento

### Operações de Transferência entre Registos: (Multiplexagem)



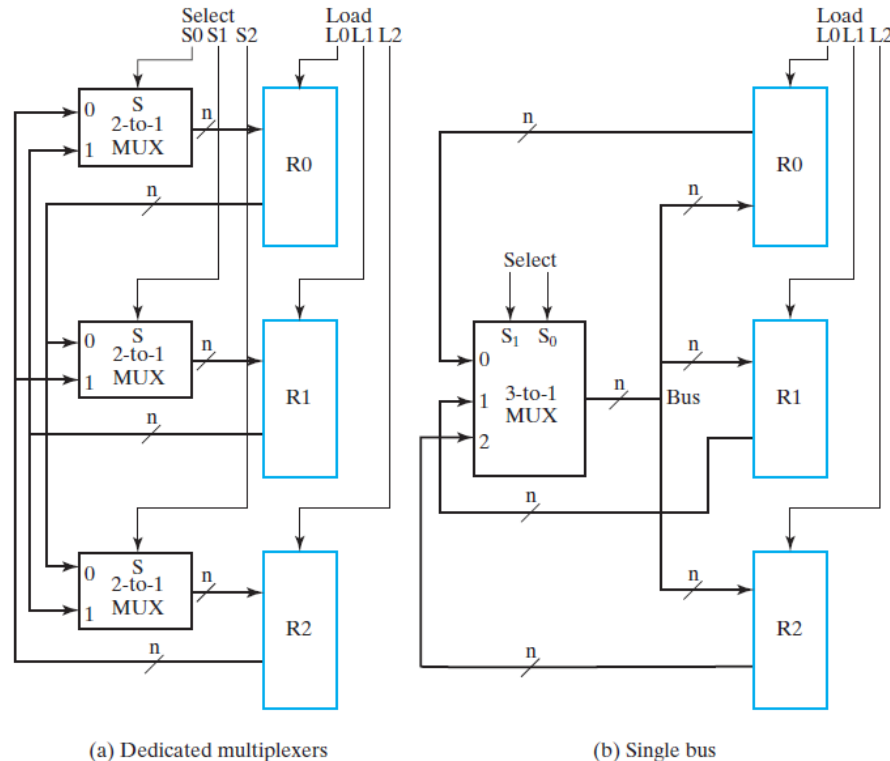
$K_1: R0 \leftarrow R1, \bar{K}_1 K_2: R0 \leftarrow R2$



# UNIDADE DE PROCESSAMENTO

## Unidade de Armazenamento

### Operações de Transferência entre Registos: (BUS)



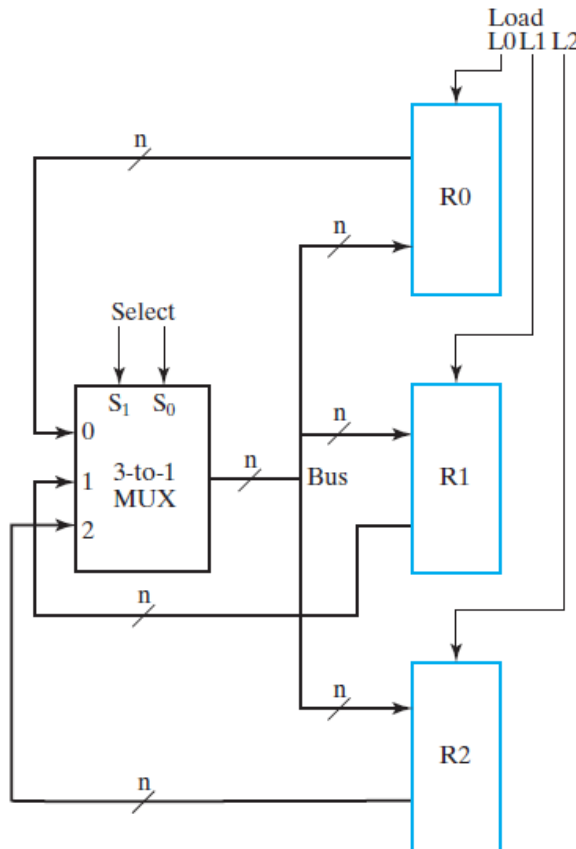
**Multiplexers Dedicados: 9 portas lógicas ( $3 \times (2\text{AND} + 1\text{OR})$ ), 6 linhas de entrada.**

**BUS: 4 portas lógicas ( $1 \times (3\text{AND} + 1\text{OR})$ ), 3 linhas de entrada.**

# UNIDADE DE PROCESSAMENTO

## Unidade de Armazenamento

### Operações de Transferência entre Registos: (BUS)



(b) Single bus

### Exemplo:

Register Transfer	Select		Load		
	S1	S0	L2	L1	L0
$R0 \leftarrow R2$	1	0	0	0	1
$R0 \leftarrow R1, R2 \leftarrow R1$	0	1	1	0	1
$R0 \leftarrow R1, R1 \leftarrow R0$	Impossible				

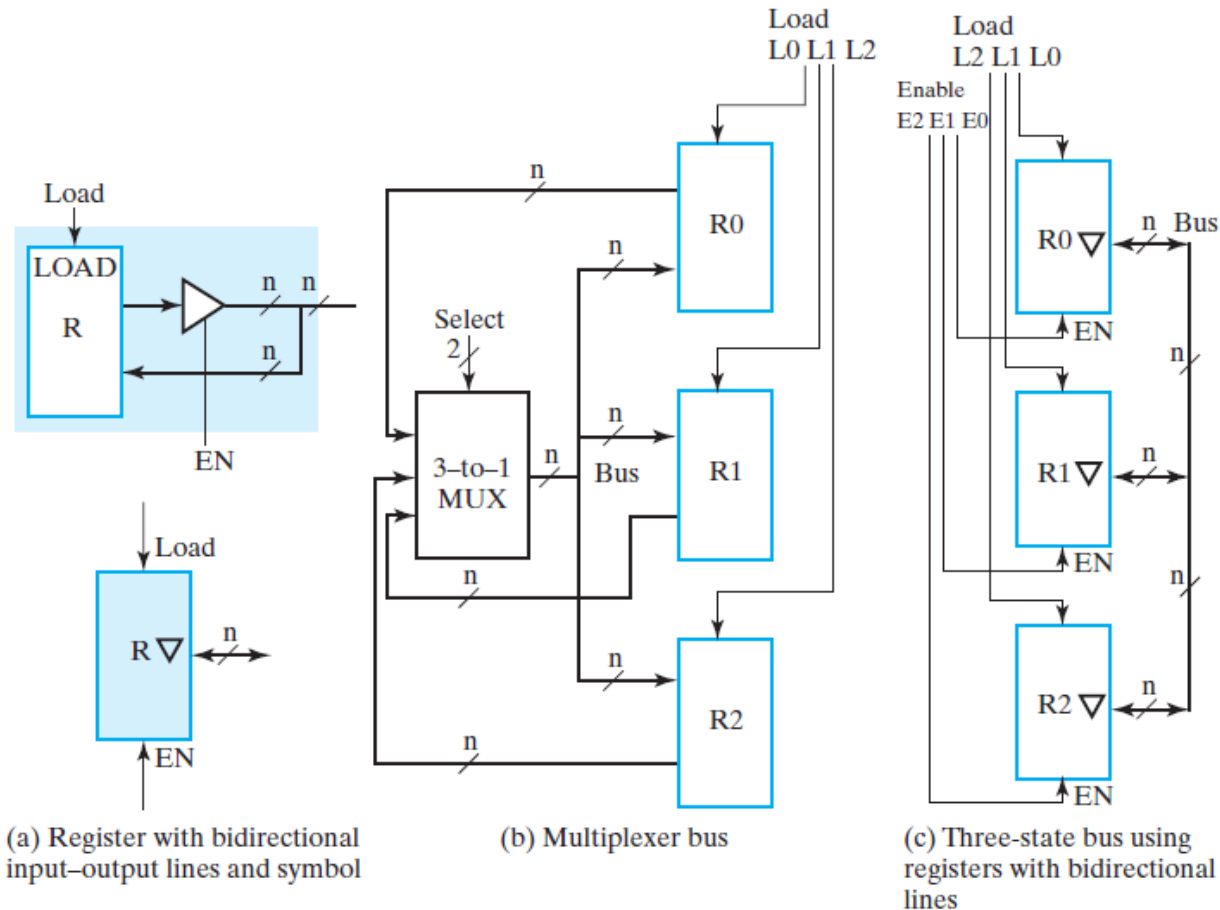
O número de registos fonte em transferências simultâneas condiciona o número mínimo de BUSES no sistema



# UNIDADE DE PROCESSAMENTO

## Unidade de Armazenamento

### Operações de Transferência entre Registos: (BUS Tri-State)



# UNIDADE DE PROCESSAMENTO

## *Unidade de Armazenamento*

*Exemplo de código VHDL para um registo de deslocamento de 4 bits.*

```
// 4-Bit Left Shift Register with Reset

library ieee;
use ieee.std_logic_1164.all;

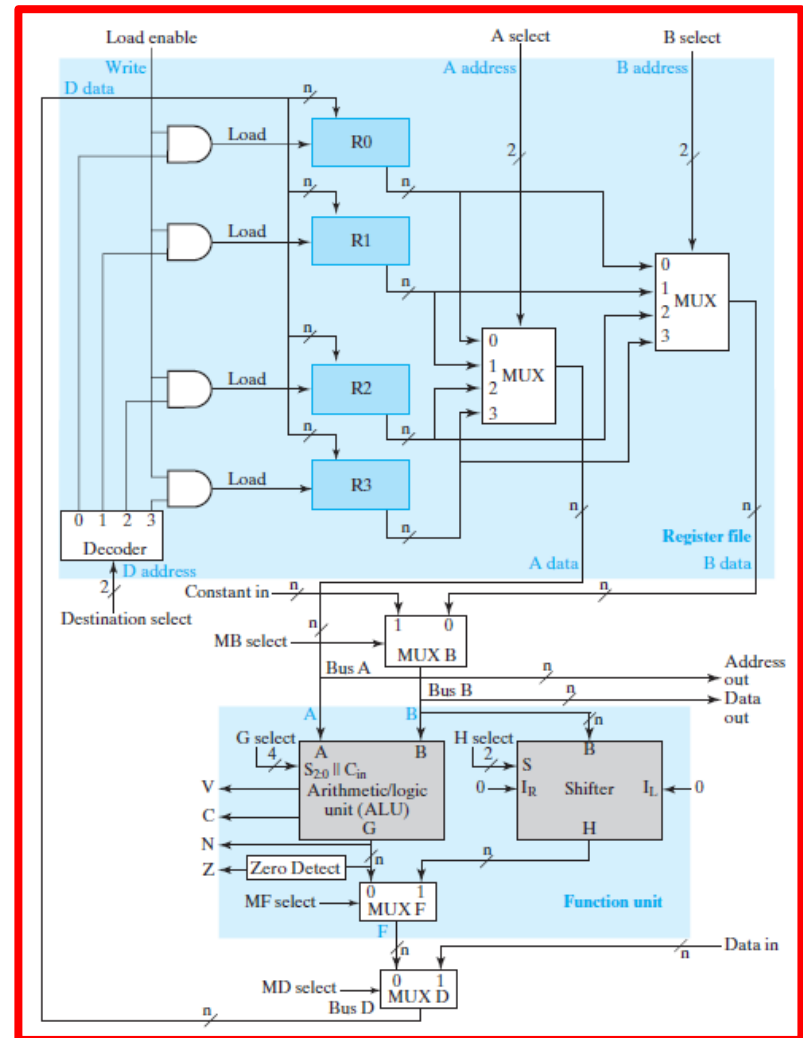
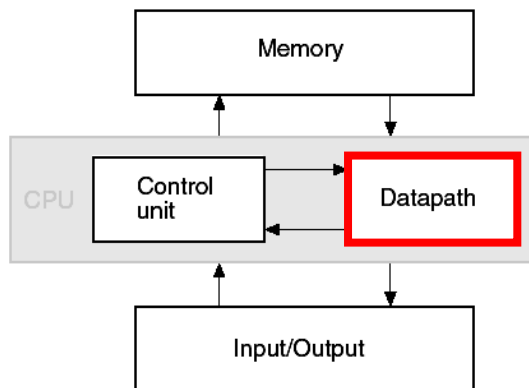
entity srg_4_r is
  port(CLK, RESET, SI : in std_logic;
        Q : out std_logic_vector(3 downto 0);
        SO : out std_logic);
end srg_4_r;

architecture behavioral of srg_4_r is
  signal shift : std_logic_vector(3 downto 0);
begin
  process (RESET, CLK)
  begin
    if (RESET = '1') then
      shift <= "0000";
    elsif (CLK'event and (CLK = '1')) then
      shift <= shift(2 downto 0) & SI;
    end if;
  end process;
  Q <= shift;
  SO <= shift(3);
end behavioral;
```

# SUMÁRIO

## Unidade de Processamento

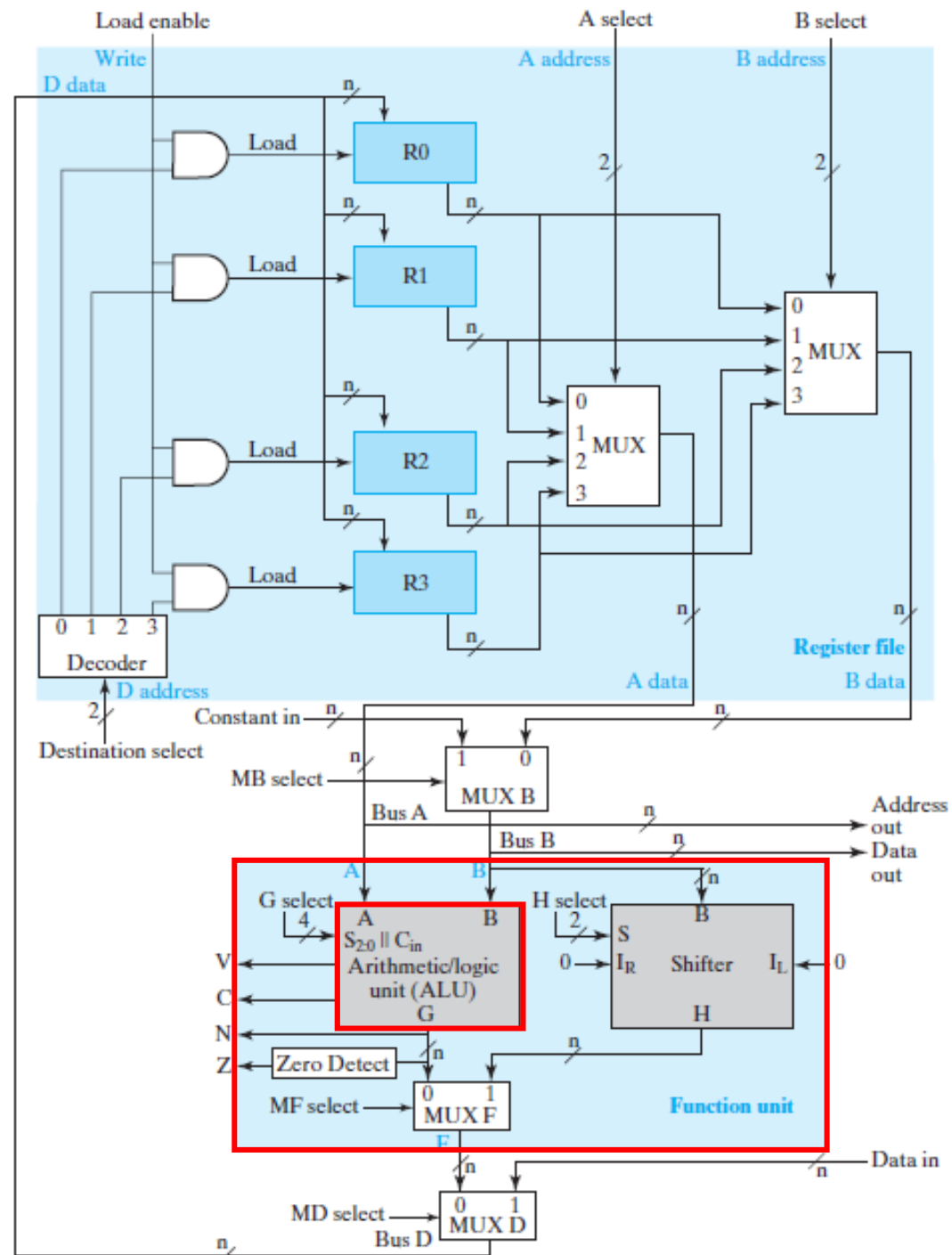
- Introdução
- Unidade de Armazenamento
- Unidade Funcional**
- Palavra de Controlo
- Temporizações
- Projeto em VIVADO®



# UNIDADE DE PROCESSAMENTO

*Unidade Funcional:*

*Unidade Aritmética e Lógica (ALU)*



# UNIDADE DE PROCESSAMENTO

## *Unidade Funcional – Unidade Aritmética e Lógica*

### *Microoperações: (Aritméticas)*

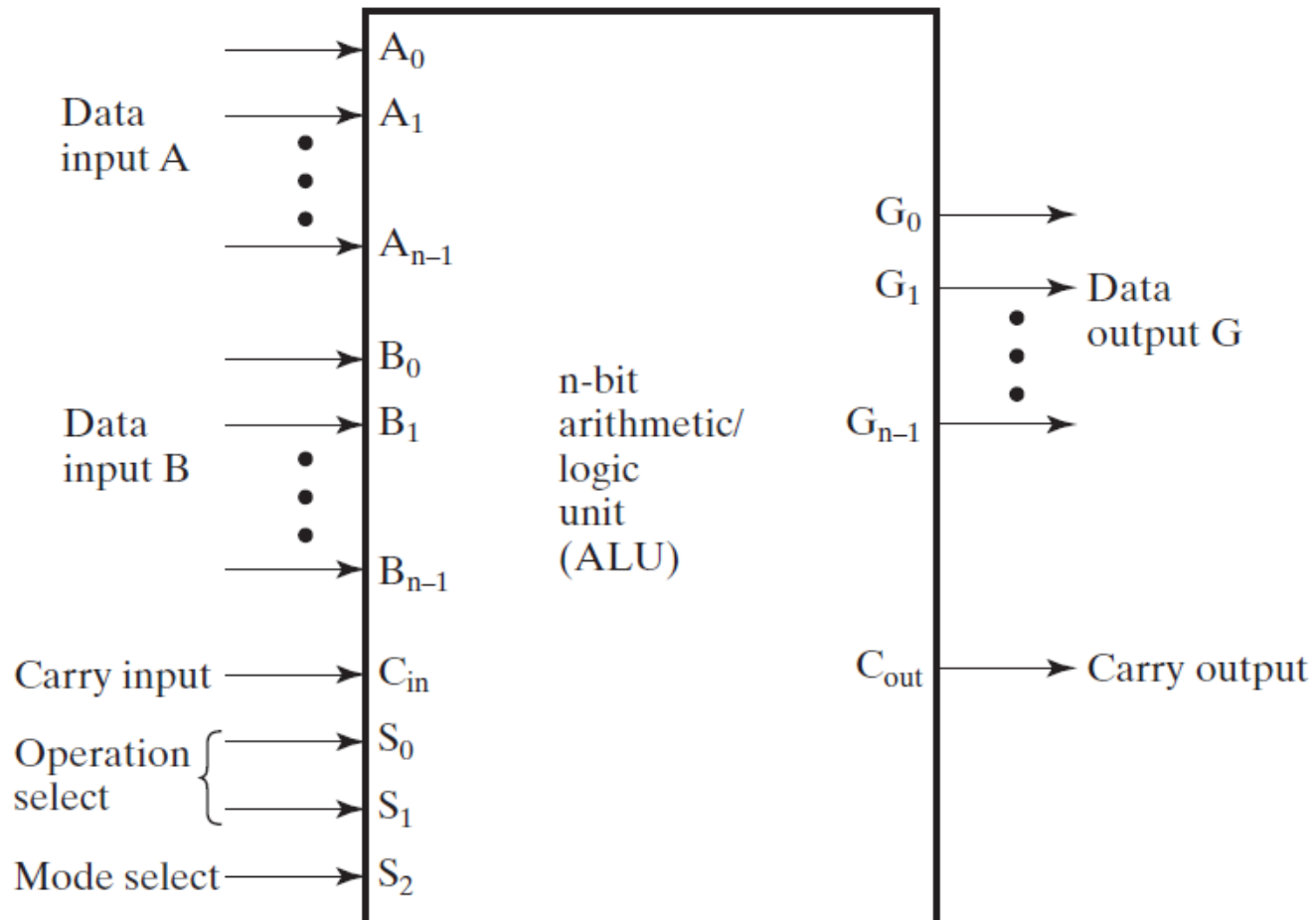
Symbolic Designation	Description
$R0 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R0$
$R2 \leftarrow \overline{R2}$	Complement of the contents of $R2$ (1s complement)
$R2 \leftarrow \overline{R2} + 1$	2s complement of the contents of $R2$
$R0 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus 2s complement of $R2$ transferred to $R0$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ (count up)
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ (count down)

### *Microoperações: (Lógicas)*

Symbolic Designation	Description
$R0 \leftarrow \overline{R1}$	Logical bitwise NOT (1s complement)
$R0 \leftarrow R1 \wedge R2$	Logical bitwise AND (clears bits)
$R0 \leftarrow R1 \vee R2$	Logical bitwise OR (sets bits)
$R0 \leftarrow R1 \oplus R2$	Logical bitwise XOR (complements bits)

# UNIDADE DE PROCESSAMENTO

## *Unidade Funcional – Unidade Aritmética e Lógica*

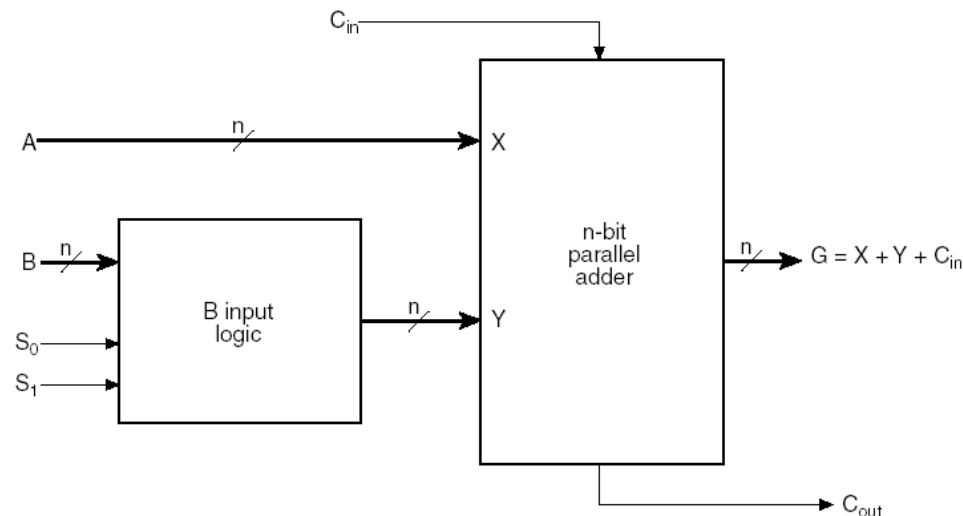


# UNIDADE DE PROCESSAMENTO

## Unidade Funcional – Unidade Aritmética e Lógica

### ALU: Circuito Aritmético (Exemplo)


Select		Input	$G = (A + Y + C_{in})$	
$S_1$	$S_0$	$Y$	$C_{in} = 0$	$C_{in} = 1$
0	0	all 0s	$G = A$ (transfer)	$G = A + 1$ (increment)
0	1	$B$	$G = A + B$ (add)	$G = A + B + 1$
1	0	$\overline{B}$	$G = A + \overline{B}$	$G = A + \overline{B} + 1$ (subtract)
1	1	all 1s	$G = A - 1$ (decrement)	$G = A$ (transfer)



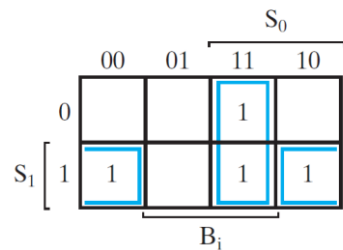
## UNIDADE DE PROCESSAMENTO

## Unidade Funcional – Unidade Aritmética e Lógica

## ALU: Circuito Aritmético (Exemplo)



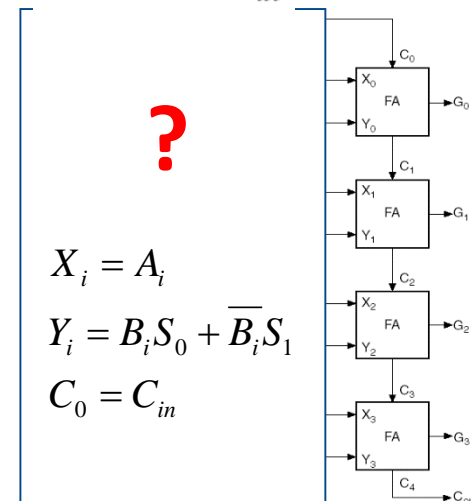
Inputs			Output
$S_1$	$S_0$	$B_i$	$Y_i$
0	0	0	0 $Y_i = 0$
0	0	1	0
0	1	0	0 $Y_i = B_i$
0	1	1	1
1	0	0	1 $Y_i = \overline{B_i}$
1	0	1	0
1	1	0	1 $Y_i = 1$
1	1	1	1



$$Y_i = \beta_0 + \beta_1 S_i + \epsilon_i$$

Select		Input	$G = (A + Y + C_{in})$	
$S_1$	$S_0$	$Y$	$C_{in} = 0$	$C_{in} = 1$
0	0	all 0s	$G = A$ (transfer)	$G = A + 1$ (increment)
0	1	$B$	$G = A + B$ (add)	$G = A + B + 1$
1	0	$\overline{B}$	$G = A + \overline{B}$	$G = A + \overline{B} + 1$ (subtract)
1	1	all 1s	$G = A - 1$ (decrement)	$G = A$ (transfer)

$$G = A + Y + C_{in}$$





# UNIDADE DE PROCESSAMENTO

## Unidade Funcional – Unidade Aritmética e Lógica

### ALU: Circuito Aritmético (Exemplo)

Soma

Subtração

Incremento

Decremento

Transferência

...

$$G = A + Y + C_{in}$$

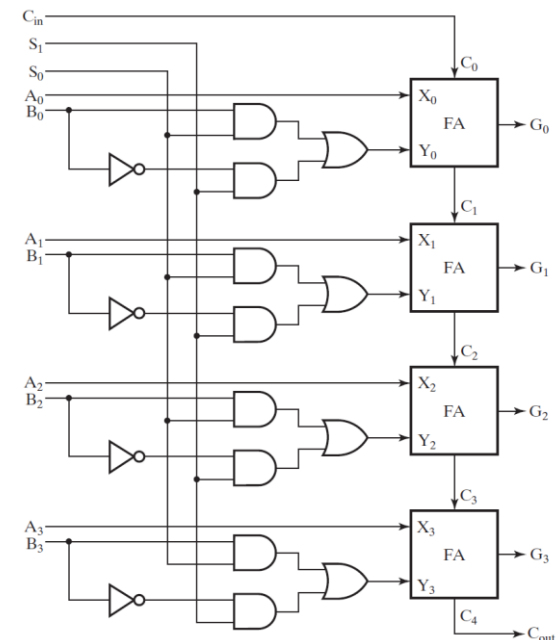
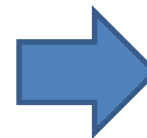
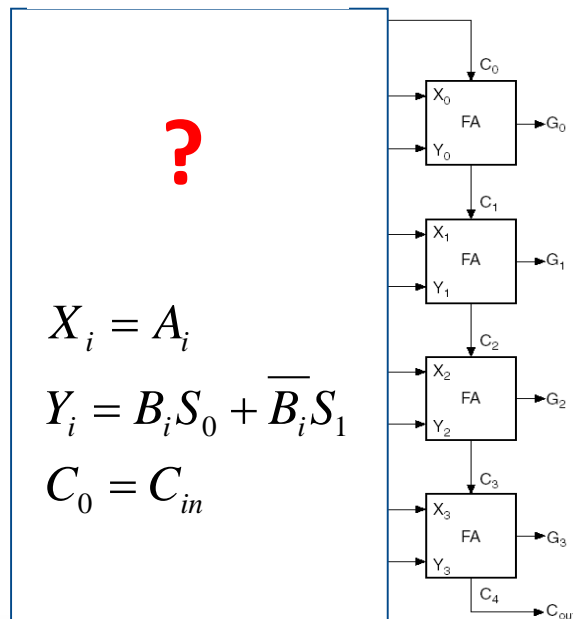
?

$$X_i = A_i$$

$$Y_i = B_i S_0 + \overline{B_i} S_1$$

$$C_0 = C_{in}$$

Select		Input	$G = (A + Y + C_{in})$	
$S_1$	$S_0$	$Y$	$C_{in} = 0$	$C_{in} = 1$
0	0	all 0s	$G = A$ (transfer)	$G = A + 1$ (increment)
0	1	$B$	$G = A + B$ (add)	$G = A + B + 1$
1	0	$\overline{B}$	$G = A + \overline{B}$	$G = A + \overline{B} + 1$ (subtract)
1	1	all 1s	$G = A - 1$ (decrement)	$G = A$ (transfer)

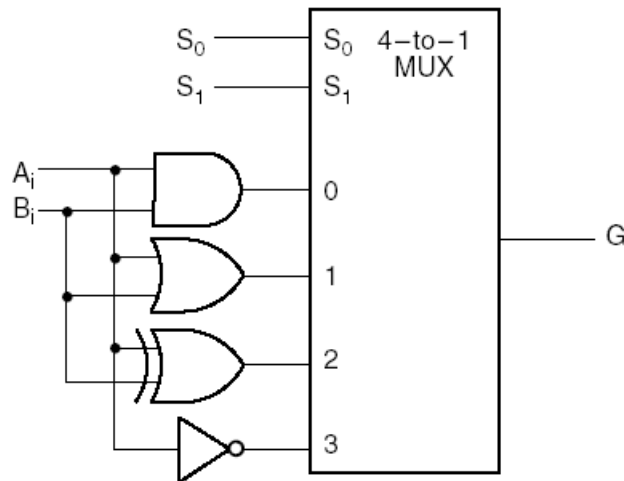


# UNIDADE DE PROCESSAMENTO

## Unidade Funcional – Unidade Aritmética e Lógica

ALU: Circuito Lógico (Exemplo)

$S_1$	$S_0$	Output	Operation
0	0	$G = A \wedge B$	AND
0	1	$G = A \vee B$	OR
1	0	$G = A \oplus B$	XOR
1	1	$G = \overline{A}$	NOT



?

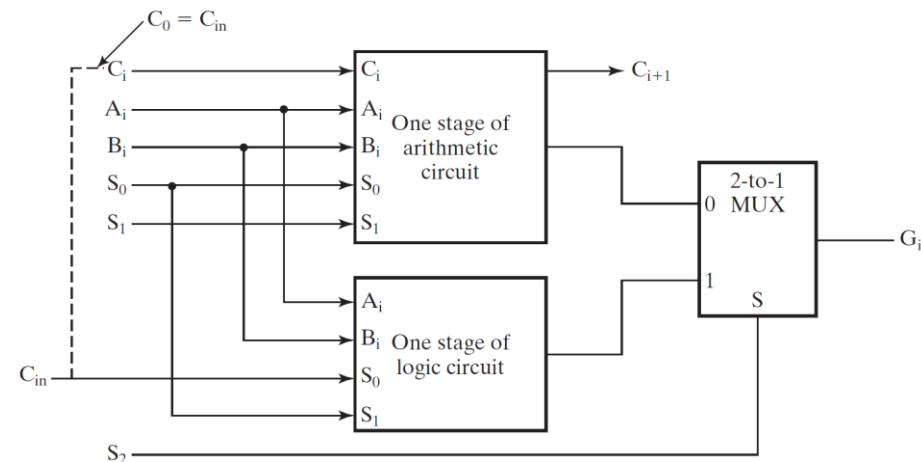
# UNIDADE DE PROCESSAMENTO

## Unidade Funcional – Unidade Aritmética e Lógica

ALU: Circuitos Aritmético

e Lógico (Exemplo)

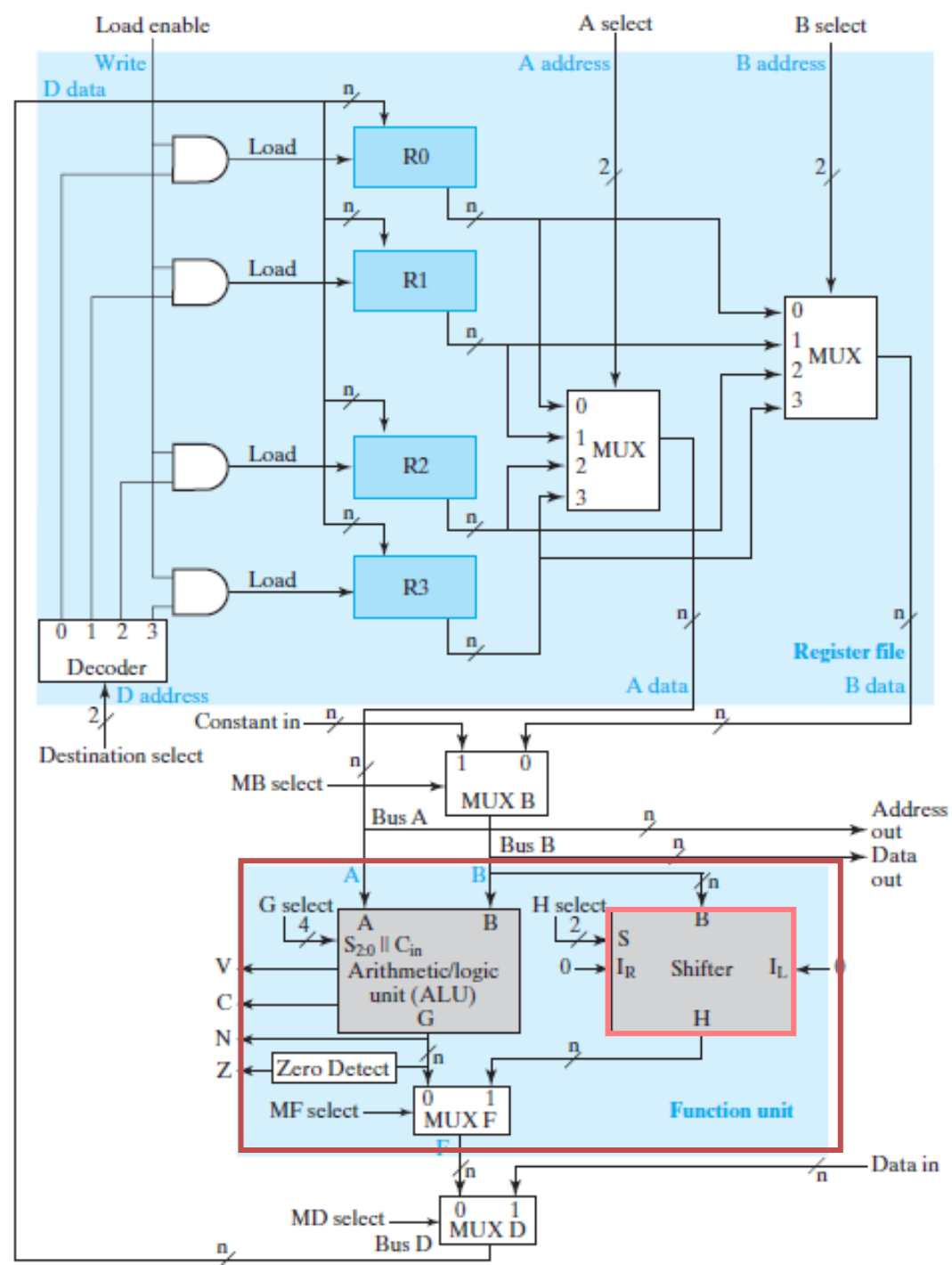
Operation Select				Operation	Function
$S_2$	$S_1$	$S_0$	$C_{in}$		
0	0	0	0	$G = A$	Transfer $A$
0	0	0	1	$G = A + 1$	Increment $A$
0	0	1	0	$G = A + B$	Addition
0	0	1	1	$G = A + B + 1$	Add with carry input of 1
0	1	0	0	$G = A + \bar{B}$	$A$ plus 1s complement of $B$
0	1	0	1	$G = A + \bar{B} + 1$	Subtraction
0	1	1	0	$G = A - 1$	Decrement $A$
0	1	1	1	$G = A$	Transfer $A$
1	X	0	0	$G = A \wedge B$	AND
1	X	0	1	$G = A \vee B$	OR
1	X	1	0	$G = A \oplus B$	XOR
1	X	1	1	$G = \bar{A}$	NOT (1s complement)



# UNIDADE DE PROCESSAMENTO

*Unidade Funcional:*

*Unidade de  
Deslocamento*



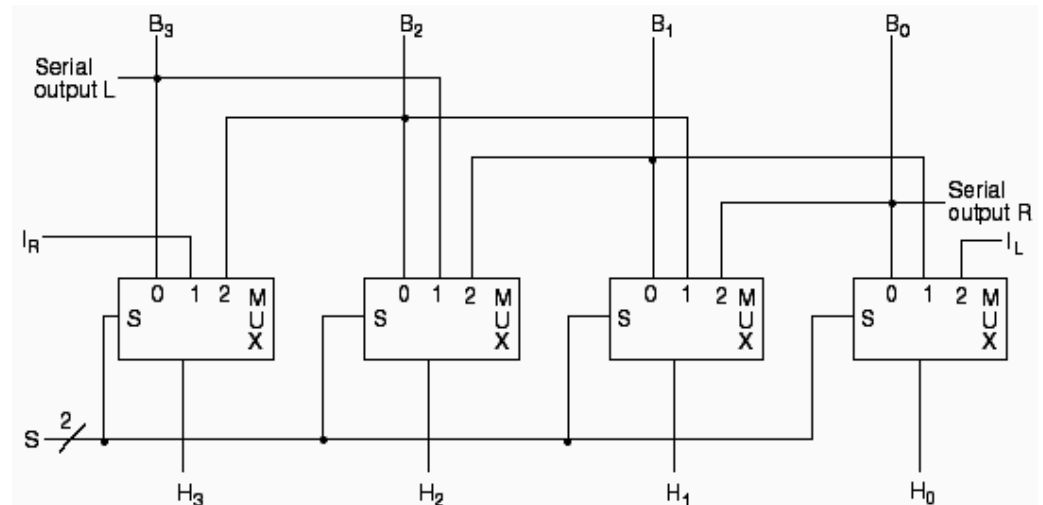
# UNIDADE DE PROCESSAMENTO

## Unidade Funcional – Unidade de Deslocamento

### Microoperações: (Deslocamento)

Type	Symbolic Designation	Eight-Bit Examples	
		Source $R2$	After Shift: Destination $R1$
Shift left	$R1 \leftarrow sl\ R2$	10011110	00111100
Shift right	$R1 \leftarrow sr\ R2$	11100101	01110010

### Circuito de Deslocamento (Exemplo1)

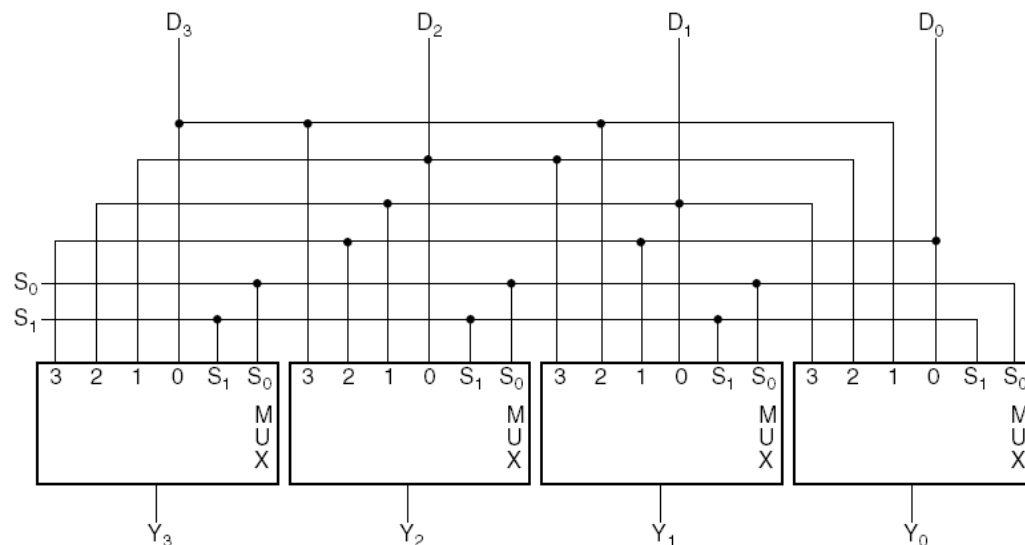


# UNIDADE DE PROCESSAMENTO

## Unidade Funcional – Unidade de Deslocamento

### Circuito de Deslocamento (Exemplo2)

Select		Output				Operation
S <sub>1</sub>	S <sub>0</sub>	Y <sub>3</sub>	Y <sub>2</sub>	Y <sub>1</sub>	Y <sub>0</sub>	
0	0	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	No rotation
0	1	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	D <sub>3</sub>	Rotate one position
1	0	D <sub>1</sub>	D <sub>0</sub>	D <sub>3</sub>	D <sub>2</sub>	Rotate two positions
1	1	D <sub>0</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	Rotate three positions



# UNIDADE DE PROCESSAMENTO

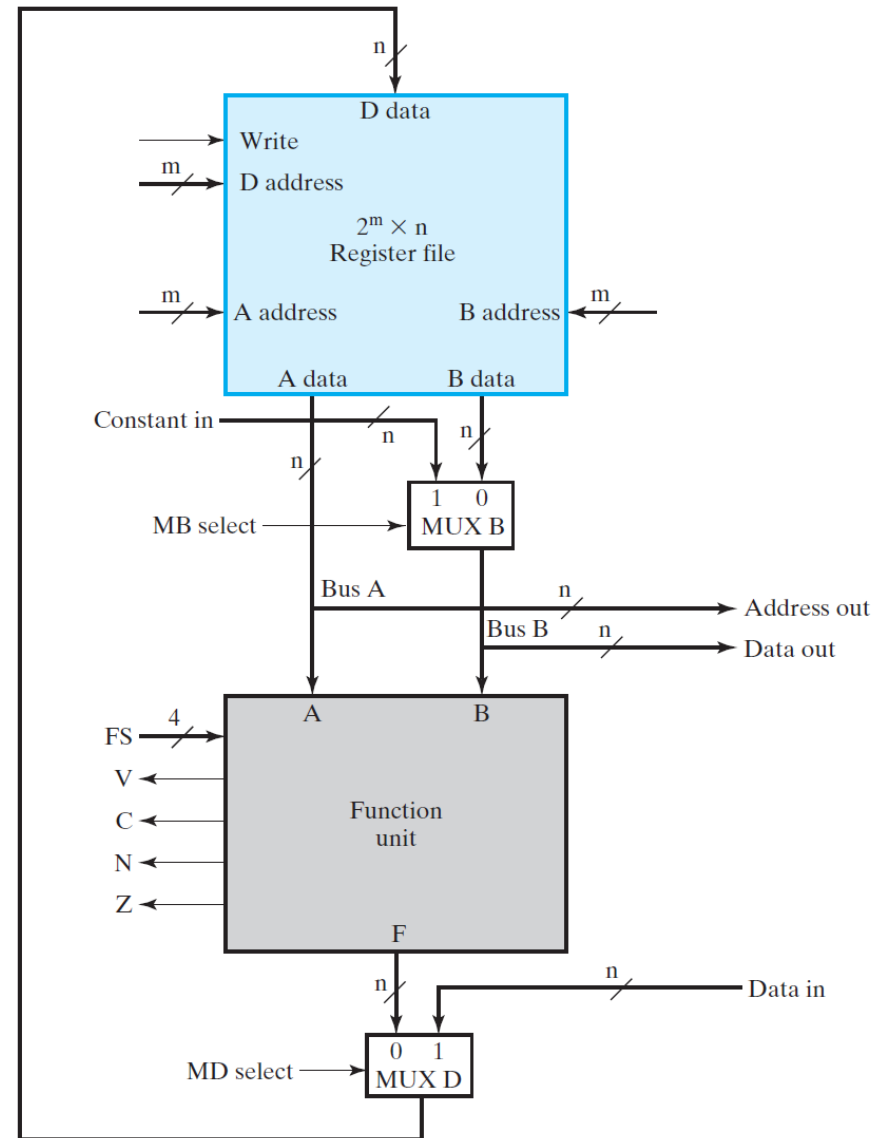
## Representação Hierárquica

### Diagrama de Blocos:

Unidade de Armazenamento

Unidade Funcional

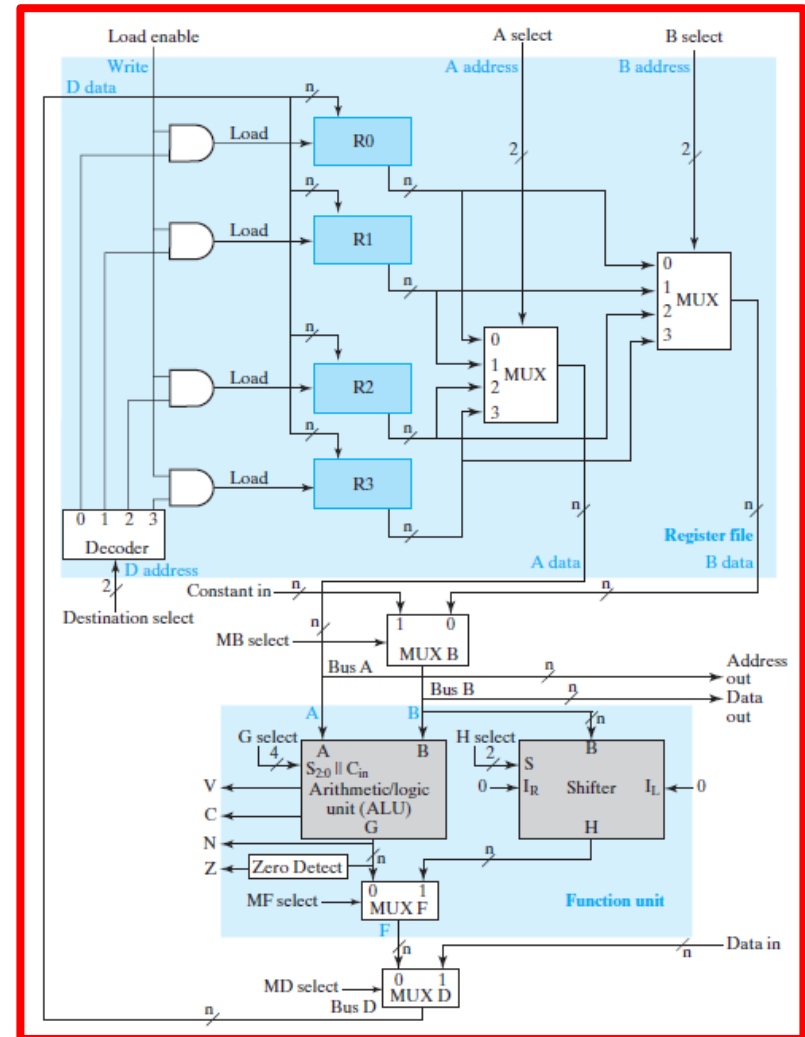
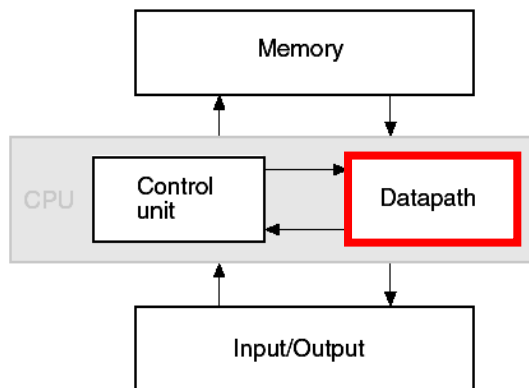
FS(3:0)	MF Select	G Select(3:0)	H Select(3:0)	Microoperation
0000	0	0000	XX	$F = A$
0001	0	0001	XX	$F = A + 1$
0010	0	0010	XX	$F = A + B$
0011	0	0011	XX	$F = A + B + 1$
0100	0	0100	XX	$F = A + \overline{B}$
0101	0	0101	XX	$F = A + \overline{B} + 1$
0110	0	0110	XX	$F = A - 1$
0111	0	0111	XX	$F = A$
1000	0	1X00	XX	$F = A \wedge B$
1001	0	1X01	XX	$F = A \vee B$
1010	0	1X10	XX	$F = A \oplus B$
1011	0	1X11	XX	$F = \overline{A}$
1100	1	XXXX	00	$F = B$
1101	1	XXXX	01	$F = sr B$
1110	1	XXXX	10	$F = sl B$



# SUMÁRIO

## Unidade de Processamento

- Introdução
- Unidade de Armazenamento
- Unidade Funcional
- Palavra de Controlo**
- Temporizações
- Projeto em VIVADO®

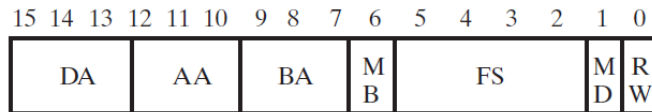




# UNIDADE DE PROCESSAMENTO

## Palavra de Controlo

Conjunto de bits correspondente às variáveis de controlo que permitem seleccionar as **microoperações**.



AA – Selecção do Registo A

BA – Selecção do Registo B

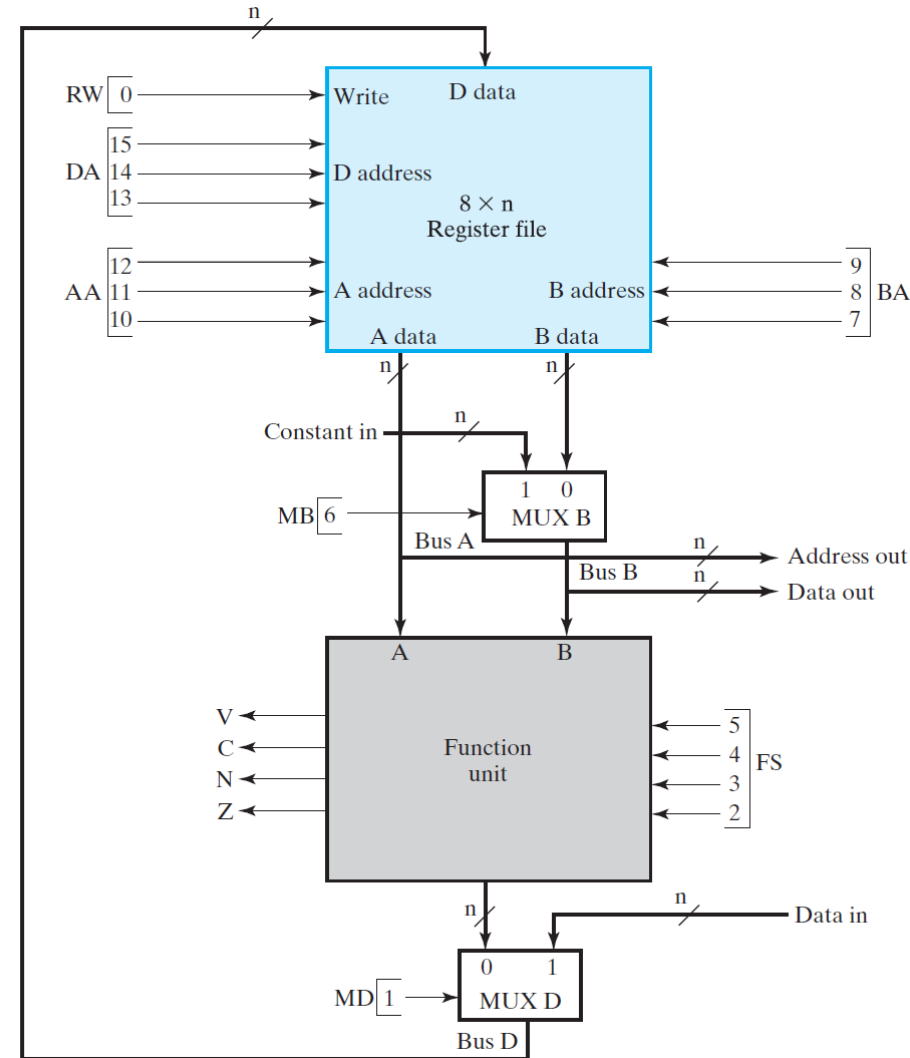
DA – Selecção do R. de Destino

MB – Selecção do Operando B

FS – Selecção da Função na U. Funcional

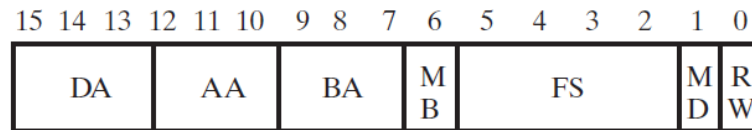
MD – Sel. dos Dados para R. de Destino

RW - Carregamento



# UNIDADE DE PROCESSAMENTO

## Codificação da Palavra de Controlo



DA, AA, BA		MB		FS		MD		RW	
Function	Code	Function	Code	Function	Code	Function	Code	Function	Code
R0	000	Register 0	0	$F = A$	0000	Function 0	0	No Write	0
R1	001	Constant 1	1	$F = A + 1$	0001	Data in	1	Write	1
R2	010			$F = A + B$	0010				
R3	011			$F = A + B + 1$	0011				
R4	100			$F = A + \overline{B}$	0100				
R5	101			$F = A + \overline{B} + 1$	0101				
R6	110			$F = A - 1$	0110				
R7	111			$F = A$	0111				
				$F = A \wedge B$	1000				
				$F = A \vee B$	1001				
				$F = A \oplus B$	1010				
				$F = \overline{A}$	1011				
				$F = B$	1100				
				$F = sr B$	1101				
				$F = sl B$	1110				

**Nota:** o número de palavras de controlo, com significado no presente caso, não é  $2^{16}=65536$ , mas apenas 61440, justifique!

# UNIDADE DE PROCESSAMENTO

## Palavra de Controlo (*Exemplo de Microoperações*)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DA			AA			BA		M B	FS			M D	R W		

$$R1 \leftarrow R2 + \overline{R3} + 1$$



Field:	DA	AA	BA	MB	FS	MD	RW
Symbolic:	$R1$	$R2$	$R3$	Register	$F = A + \overline{B} + 1$	Function	Write
Binary:	001	010	011	0	0101	0	1

# UNIDADE DE PROCESSAMENTO

## Palavra de Controlo (Exemplo de Microoperações)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DA			AA			BA		M B	FS			M D	R W		

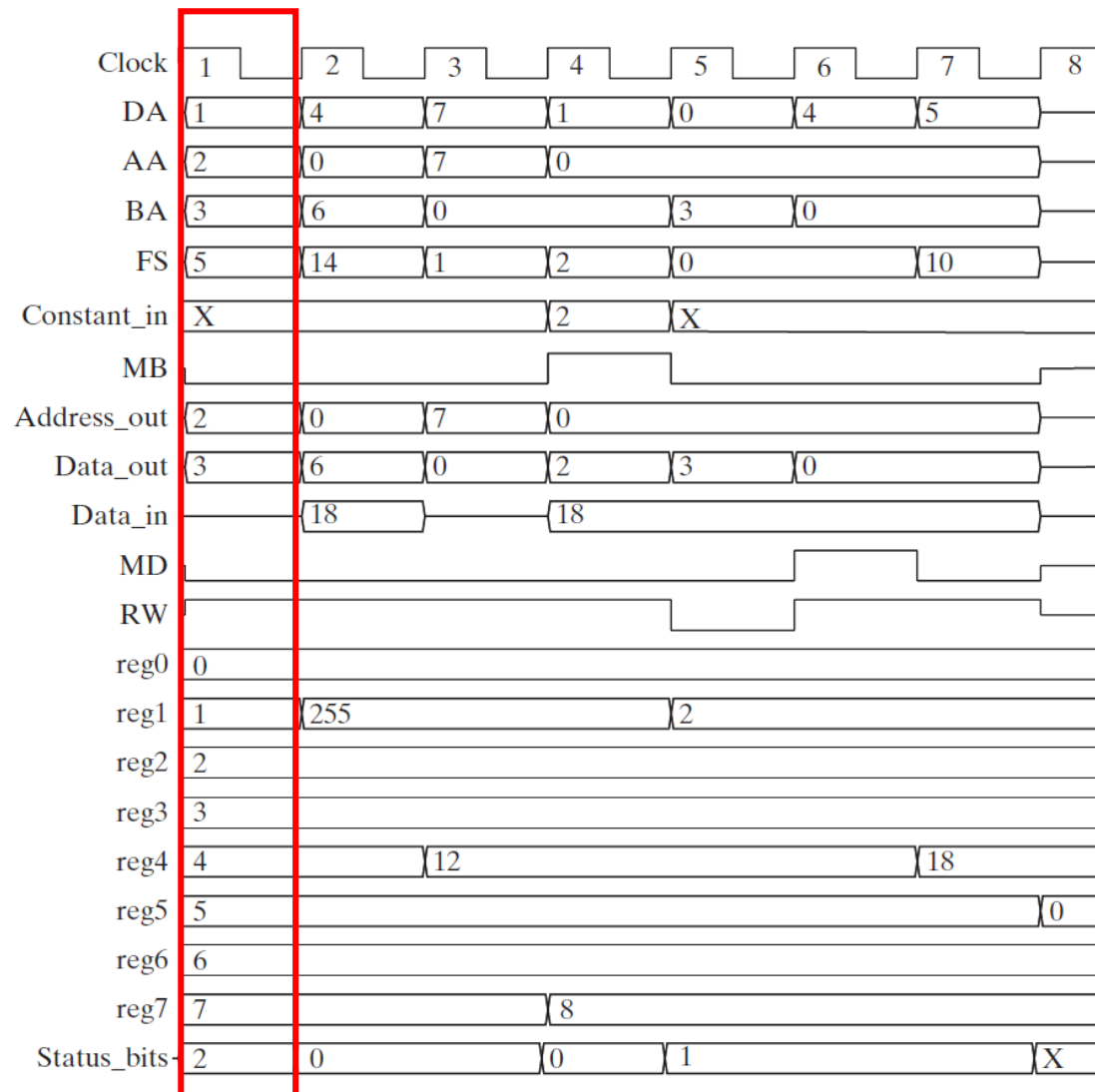
Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	R1	R2	R3	Register	$F = A + \overline{B} + I$	Function	Write
$R4 \leftarrow \text{sl } R6$	R4	—	R6	Register	$F = \text{sl } B$	Function	Write
$R7 \leftarrow R7 + 1$	R7	R7	—	—	$F = A + 1$	Function	Write
$R1 \leftarrow R0 + 2$	R1	R0	—	Constant	$F = A + B$	Function	Write
Data out $\leftarrow R3$	—	—	R3	Register	—	—	No Write
$R4 \leftarrow \text{Data in}$	R4	—	—	—	—	Data in	Write
$R5 \leftarrow 0$	R5	R0	R0	Register	$F = A \oplus B$	Function	Write

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	001	010	011	0	0101	0	1
$R4 \leftarrow \text{sl } R6$	100	XXX	110	0	1110	0	1
$R7 \leftarrow R7 + 1$	111	111	XXX	X	0001	0	1
$R1 \leftarrow R0 + 2$	001	000	XXX	1	0010	0	1
Data out $\leftarrow R3$	XXX	XXX	011	0	XXXX	X	0
$R4 \leftarrow \text{Data in}$	100	XXX	XXX	X	XXXX	1	1
$R5 \leftarrow 0$	101	000	000	0	1010	0	1

# UNIDADE DE PROCESSAMENTO

## Simulação da Execução de uma Sequência de Microoperações (Exemplo)

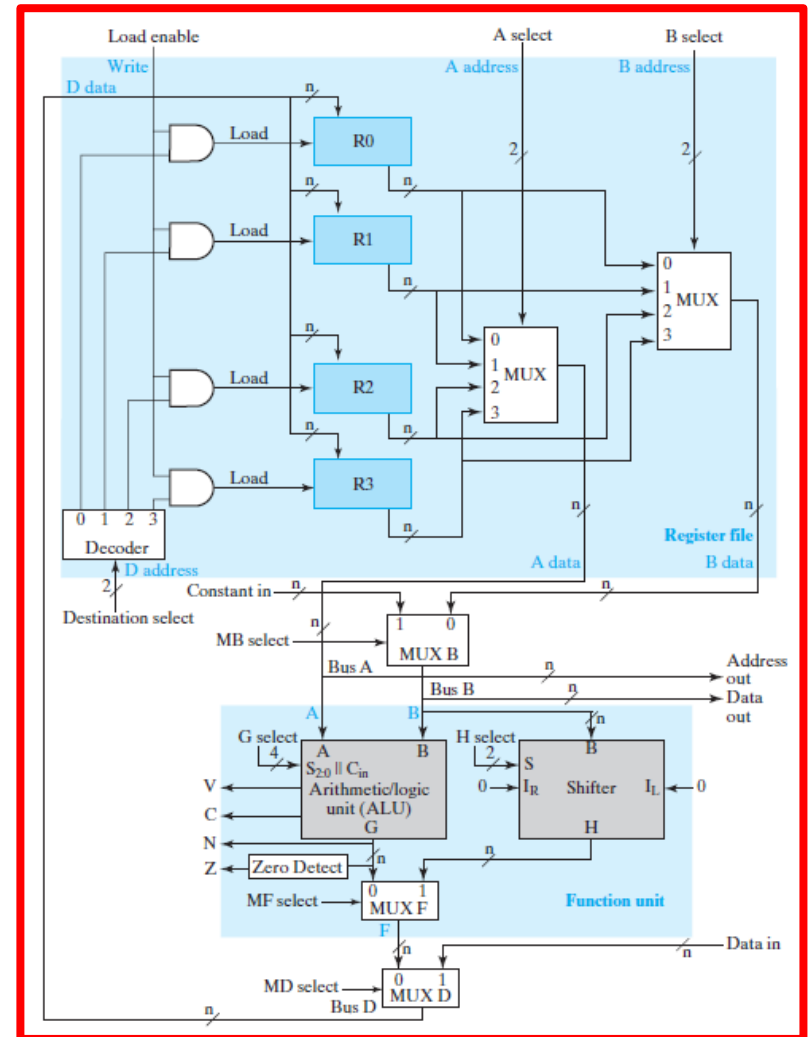
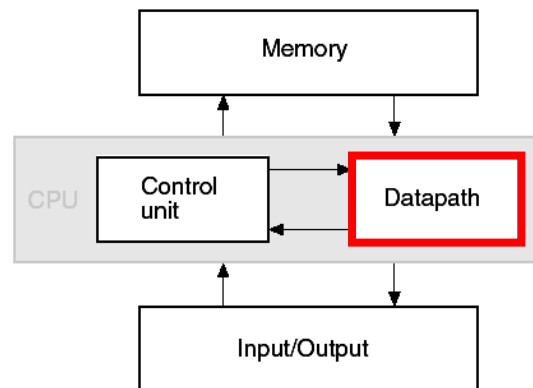
Micro-operation	DA	AA	BA	MB
$R1 \leftarrow R2 - R3$	R1	R2	R3	Register
$R4 \leftarrow sl\ R6$	R4	—	R6	Register
$R7 \leftarrow R7 + 1$	R7	R7	—	—
$R1 \leftarrow R0 + 2$	R1	R0	—	Constant
$Data\ out \leftarrow R3$	—	—	R3	Register
$R4 \leftarrow Data\ in$	R4	—	—	—
$R5 \leftarrow 0$	R5	R0	R0	Register



# SUMÁRIO

## Unidade de Processamento

- Introdução
- Unidade de Armazenamento
- Unidade Funcional
- Palavra de Controlo
- Temporizações**
- Projeto em VIVADO®
- Pipeline



# UNIDADE DE PROCESSAMENTO

## Temporizações

$$F_{\max} = 1/T_{\min}$$

$F_{\max}$  – Frequência máxima de funcionamento

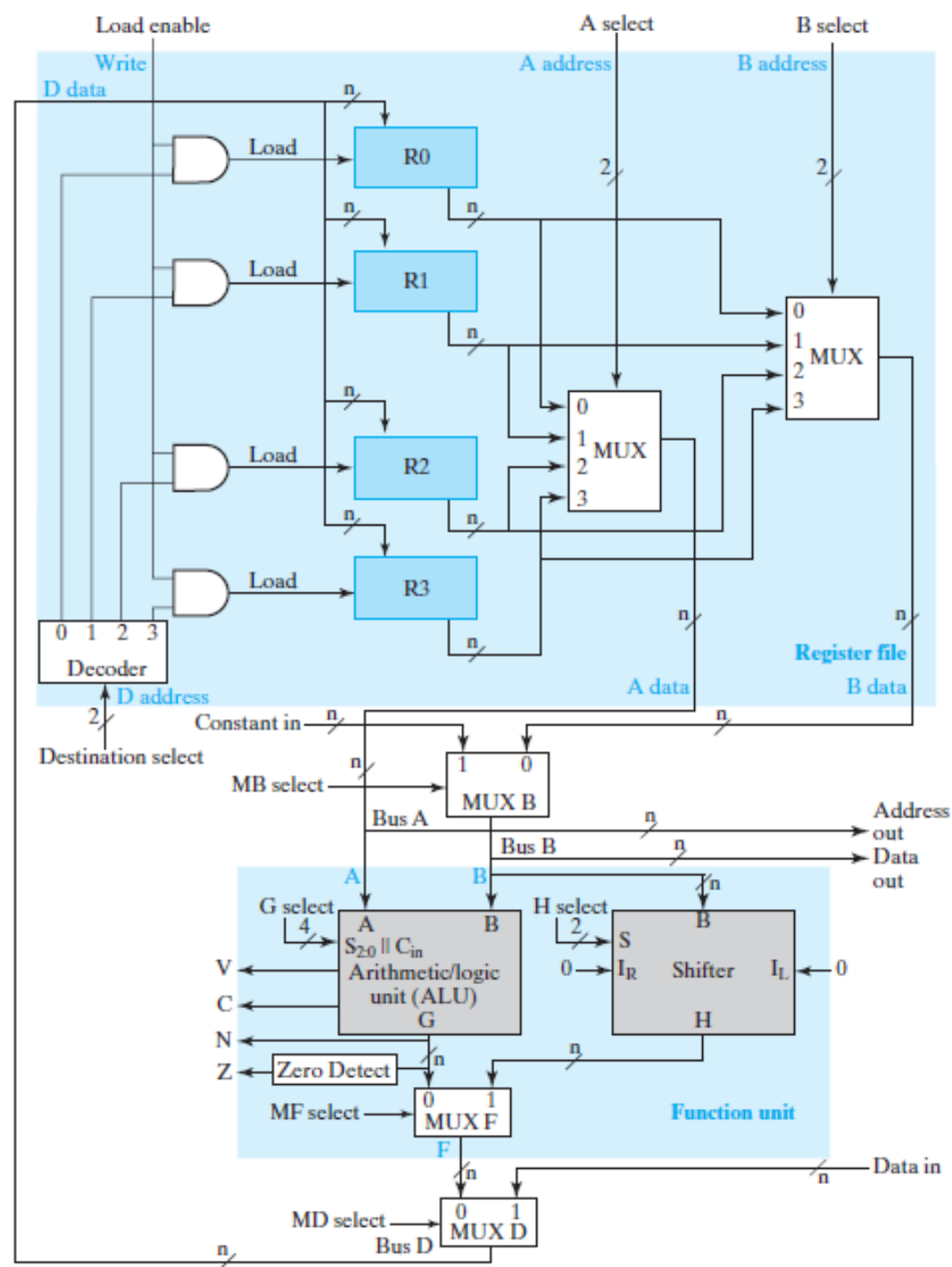
$T_{\min}$  – Período mínimo que respeita todas as restrições temporais impostas por cada componente

$$F_{\max} = 1/(T_{ua} + T_{uf} + T_{lc})$$

$T_{ua}$  – Restrição temporal imposta pela UA

$T_{uf}$  – Restrição temporal imposta pela UF

$T_{lc}$  – Restrição temporal impostas pela lógica combinatória adicional



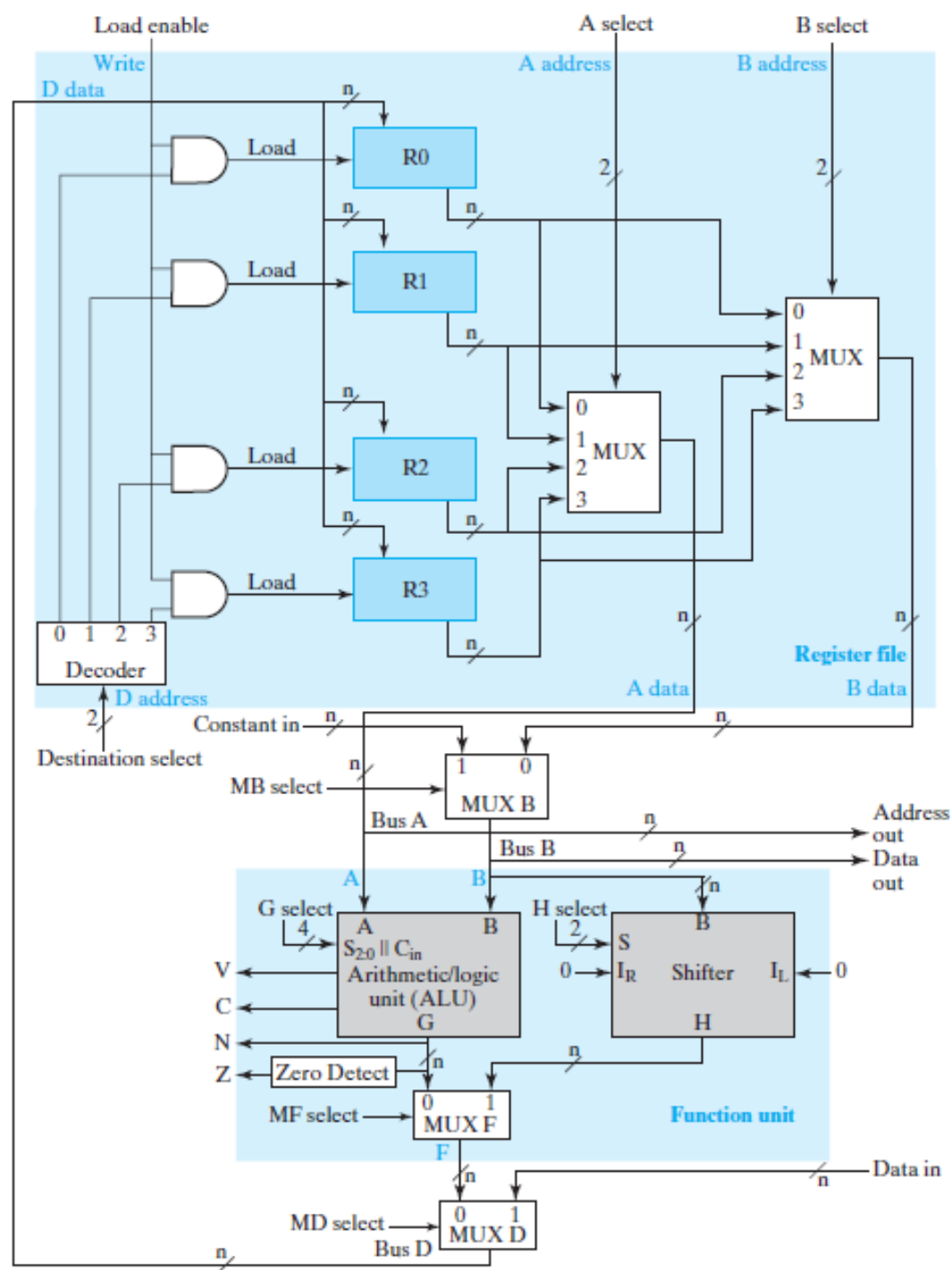
# UNIDADE DE PROCESSAMENTO

## Unidade de Armazenamento

### Temporizações

**Tua** – Restrição temporal imposta pela UA

Não considerando, para já, a geração dos sinais de controlo, a contribuição da UA para a definição do período do sinal de relógio corresponde, por um lado, ao caminho dos dados que inclui o tempo de propagação nos registos e o tempo de propagação nos MUXs e, por outro lado, ao caminho dos sinais de controlo correspondentes à geração do sinal de load. Estes 2 caminhos são concorrentes na determinação do período máximo devendo ser considerada a situação mais desfavorável.





# UNIDADE DE PROCESSAMENTO

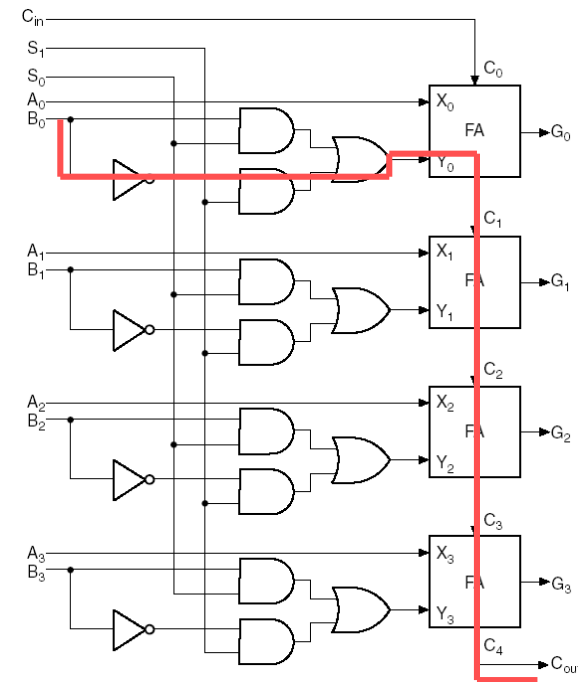
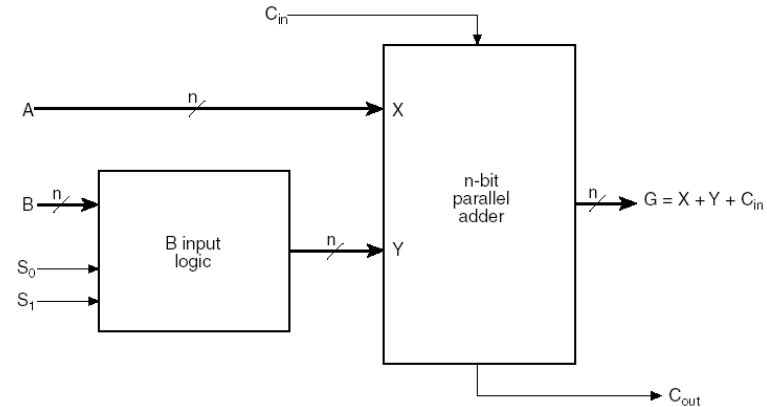
## Unidade Funcional (1) – Unidade Aritmética

### Temporizações

**Tuf** – Restrição temporal imposta pela UF

Neste caso há que considerar a situação mais desfavorável para a propagação do sinal ao nível dos vários circuitos combinatórios: Unidade Aritmética, Unidade Lógica e Unidade de Deslocamento.

Na Unidade Aritmética há que ter em conta a propagação do Carry ao longo do somador.



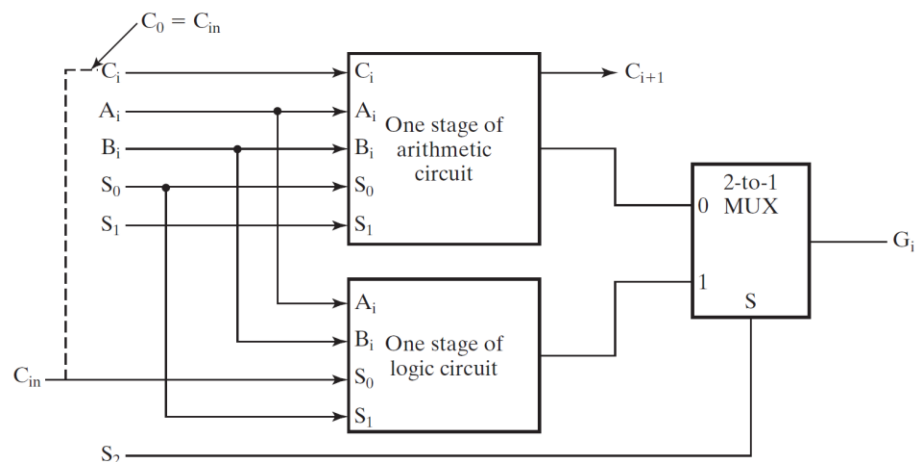
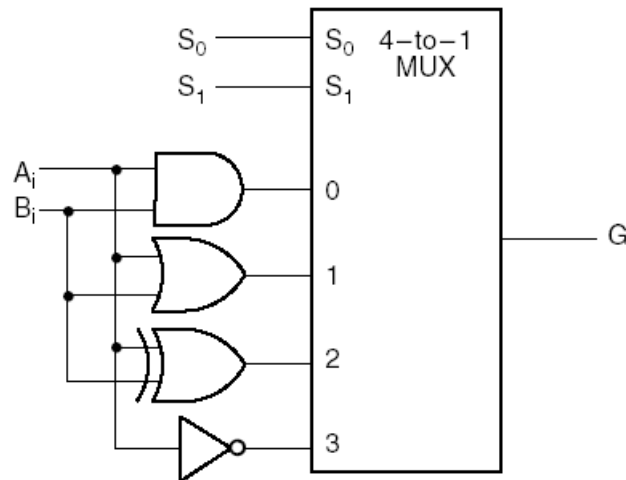
# UNIDADE DE PROCESSAMENTO

## Unidade Funcional (2) – Unidade Lógica

### Temporizações

Na Unidade Lógica há que ter em conta a diferente complexidade das várias portas lógicas envolvidas.

Considerando o bloco U. Aritmética e Lógica deve, neste caso, ser considerado o maior dos atrasos associados a estes blocos e adicionar o atraso associado ao MUX de interligação.



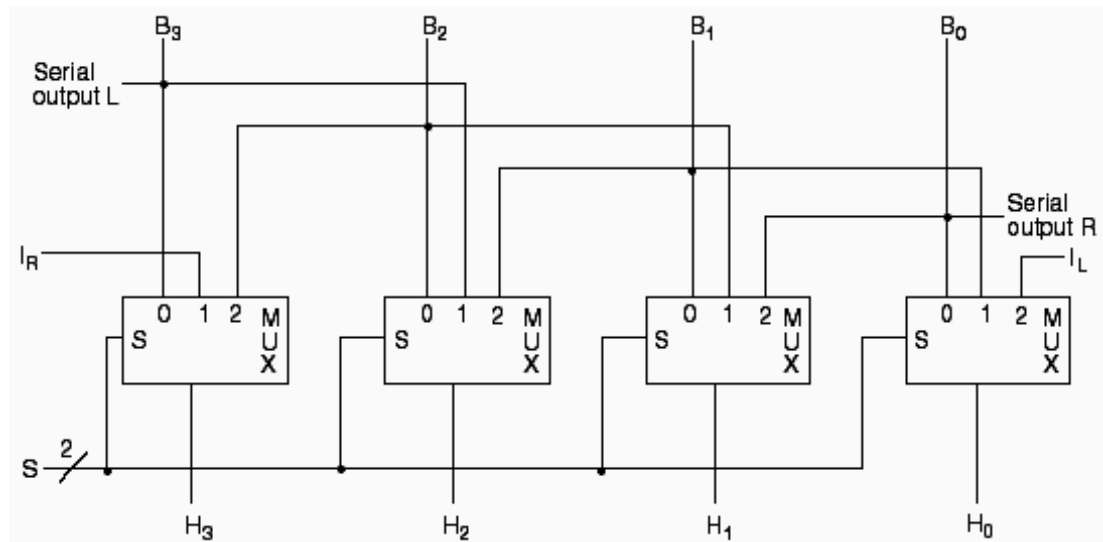
# UNIDADE DE PROCESSAMENTO

## *Unidade Funcional (2) – Unidade de Deslocamento*

### *Temporizações*

Na U. De Deslocamento há que considerar o tempo de propagação através dos MUXs (1 nível de MUXs)

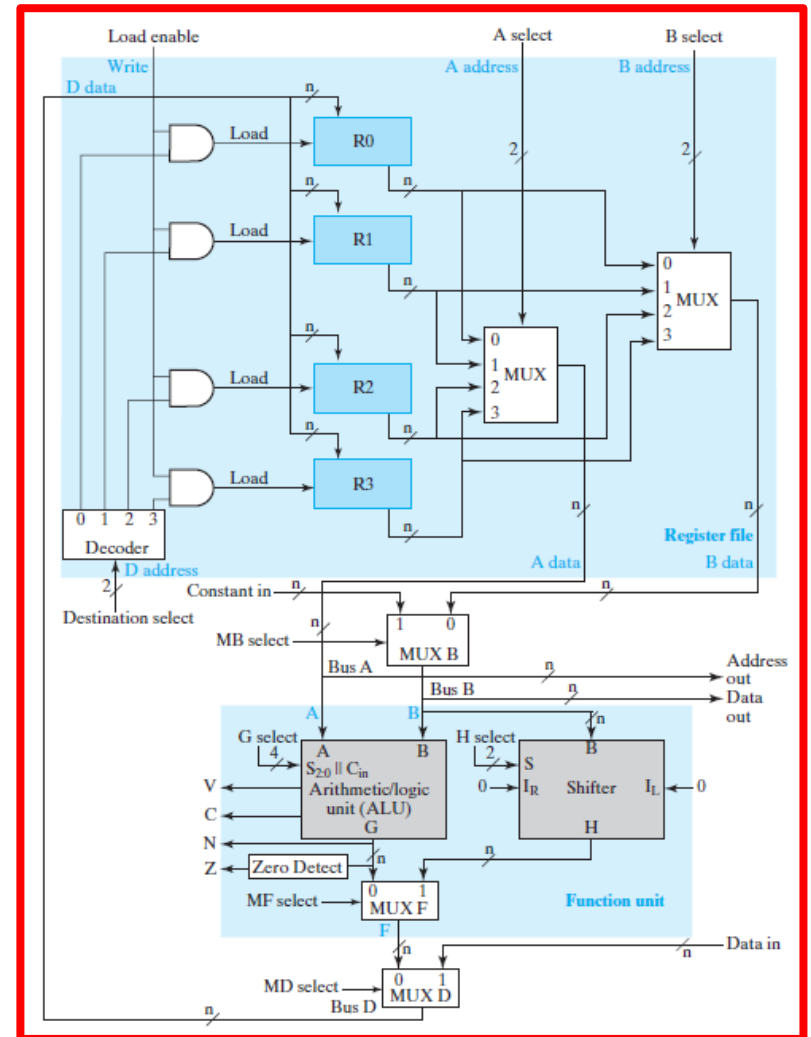
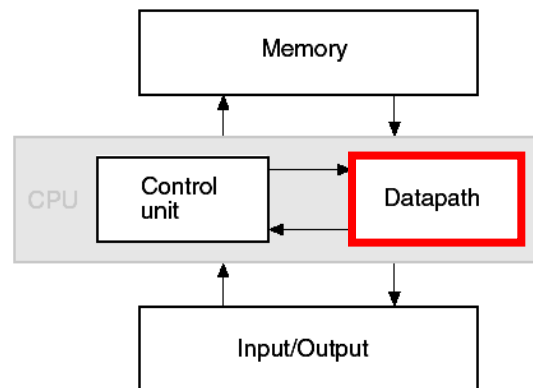
Em conjugação com Uarit. e Ulog deve ser considerado ainda o tempo de propagação associado ao MUX de interligação.



# SUMÁRIO

## Unidade de Processamento

- Introdução
- Unidade de Armazenamento
- Unidade Funcional
- Palavra de Controlo
- Temporizações
- Projeto em VIVADO®**



# UNIDADE DE PROCESSAMENTO

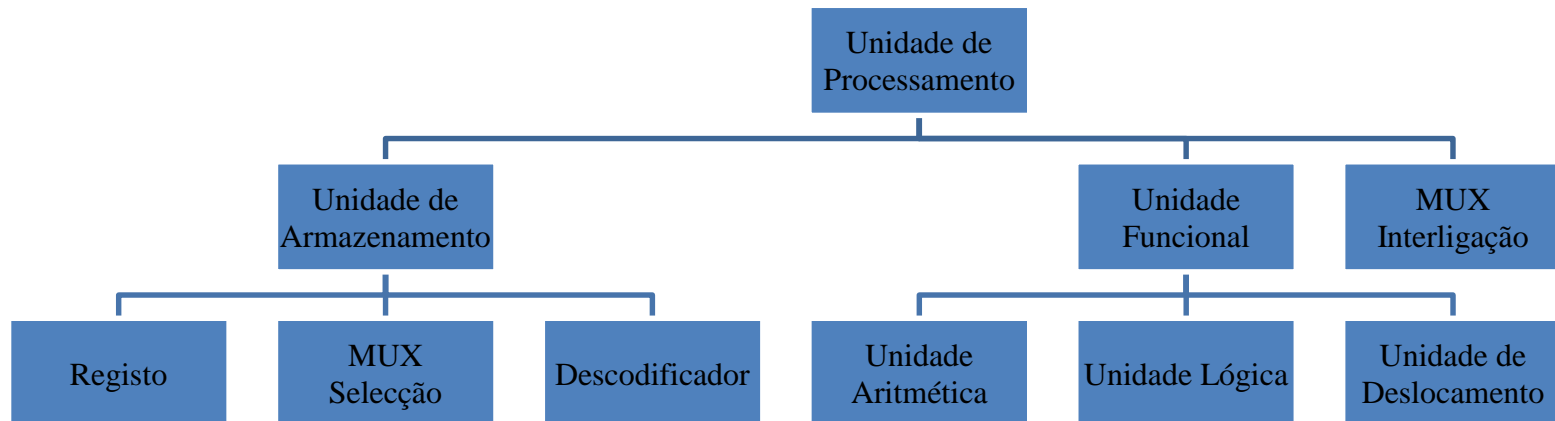
## *Projeto no VIVADO (Especificação e Hierarquia)*

UP de 16 bits

UA com Registos de 6 bits

UF composta por UArit, ULog, UDesl

MUXs de interligação



# UNIDADE DE PROCESSAMENTO

## Projeto no VIVADO (Abrir Projecto)

The screenshot displays the Vivado 2016.4 interface. The 'Quick Start' panel on the left has the 'Open Project' button circled in red. The 'Open Project' dialog box is open, showing the project files, with 'projeto.sim' circled in red. The 'Project Manager' window shows the project hierarchy, with 'DataPath - structural (DataPath.vhd) (3)' circled in red. The 'Project Summary' window shows project settings, and the 'Design Runs' table is visible at the bottom.

**Project Summary**

**Project Settings**

Project name: projeto  
Project location: C:/Users/NHorta/Desktop/ACOMP 20162017 aulas/projeto/projeto  
Product family: Artix-7  
Project part: xc7a35tfgq256-1  
Top module name: DataPath  
Target language: VHDL  
Simulator language: Mixed

**Synthesis**

Status: Not started  
Messages: No errors or warnings

**Implementation**

Status: Not started  
Messages: No errors or warnings

**Design Runs**

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	PCle %	Start
synth_1	constrs_1	Not started													
impl_1	constrs_1	Not started													

# UNIDADE DE PROCESSAMENTO

## Projeto no VIVADO (Analisar Descrições de Componentes)

The image displays the Vivado IDE interface with several windows open, illustrating the project structure and the implementation of a 16-bit register.

**Sources Window (Top Left):** Shows the project hierarchy. The file **DataPath - structural (DataPath.vhd) (3)** is circled in red.

**Sources Window (Middle Left):** Shows the hierarchy for the **RF1 - RegisterFile - structural (RegisterFile.vhd) (18)** component. The file **R1 - Register 16 - structural (Register16.vhd)** is circled in red.

**Project Manager - projeto (Bottom Left):** Shows the project hierarchy. The file **R1 - Register 16 - structural (Register16.vhd)** is circled in red.

**Source File Properties (Bottom Left):** Shows the properties for the selected file **Register16.vhd**. The **Enabled** checkbox is checked. The location is **C:/Users/NHorta/Desktop/ACOMP 20162017 aulas/proje**.

**Register16.vhd (Right):** Shows the VHDL code for the **Register16** component. The code is as follows:

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity Register16 is
5     port(
6         D: in std_logic_vector(15 downto 0);
7         Load: in std_logic;
8         CLK: in std_logic;
9         Q: out std_logic_vector(15 downto 0) := (others => '0')
10    );
11 end Register16;
12
13 architecture structural of Register16 is
14 begin
15     Q <= D when CLK'event and CLK='1' and Load='1';
16 end structural;
```

# UNIDADE DE PROCESSAMENTO

## Projeto no VIVADO (Simulação – Importar ficheiro)

The image displays the Xilinx Vivado IDE interface during a simulation setup project. The 'Add Sources' dialog is open, showing the 'Add or create simulation sources' option selected. The 'Add Source Files' window is also open, showing a list of files to be added, including '2ª SimulDataPath', '3ª SimulDataPath', '4ª SimulDataPath', '5ª SimulDataPath', and '6ª SimulDataPath'. The 'Behavioral Simulation' project structure is visible in the 'Sources' window, showing the hierarchy of components and files, including 'SimulDataPath - Behavioral (2ª SimulDataPath.vhd)' and 'SimulDataPath - Behavioral (SimulDataPath.vhd) (1)'.

**Add Sources**

This guides you through the process of adding and creating sources for your project

- ☐ Add or create constraints
- ☐ Add or create design sources
- ☒ Add or create simulation sources
- ☐ Add or create DSP sources
- ☐ Add existing block design sources
- ☐ Add existing IP

To continue, click Next

**Add Source Files**

Look in: 2.3

- 2ª SimulDataPath
- 3ª SimulDataPath
- 4ª SimulDataPath
- 5ª SimulDataPath
- 6ª SimulDataPath

File name: 2ª SimulDataPath.vhd

Files of type: Design Source Files (.vhd, .vhdl, .vhf, .vhdp, .vho, .v, .vf, .verilog, ...)

**Behavioral Simulation - Functional - sim\_1 - SimulDataPath**

Sources

Messages: 2 warnings

- FU1 - FunctionalUnit - structural (FunctionalUnit.vhd)
- DM1 - DataMemory - structural (DataMemory.vhd)
- Text (1)
- Constraints
- Simulation Sources (3)
  - sim\_1 (3)
    - Non-module Files (1)
    - SimulDataPath - Behavioral (2ª SimulDataPath.vhd) (3)
      - DP0 - DataPath - structural (DataPath.vhd) (3)
      - RF1 - RegisterFile - structural (RegisterFile.vhd)
      - FU1 - FunctionalUnit - structural (FunctionalUnit.vhd)
      - DM1 - DataMemory - structural (DataMemory.vhd)
      - SimulDataPath - Behavioral (SimulDataPath.vhd) (1)

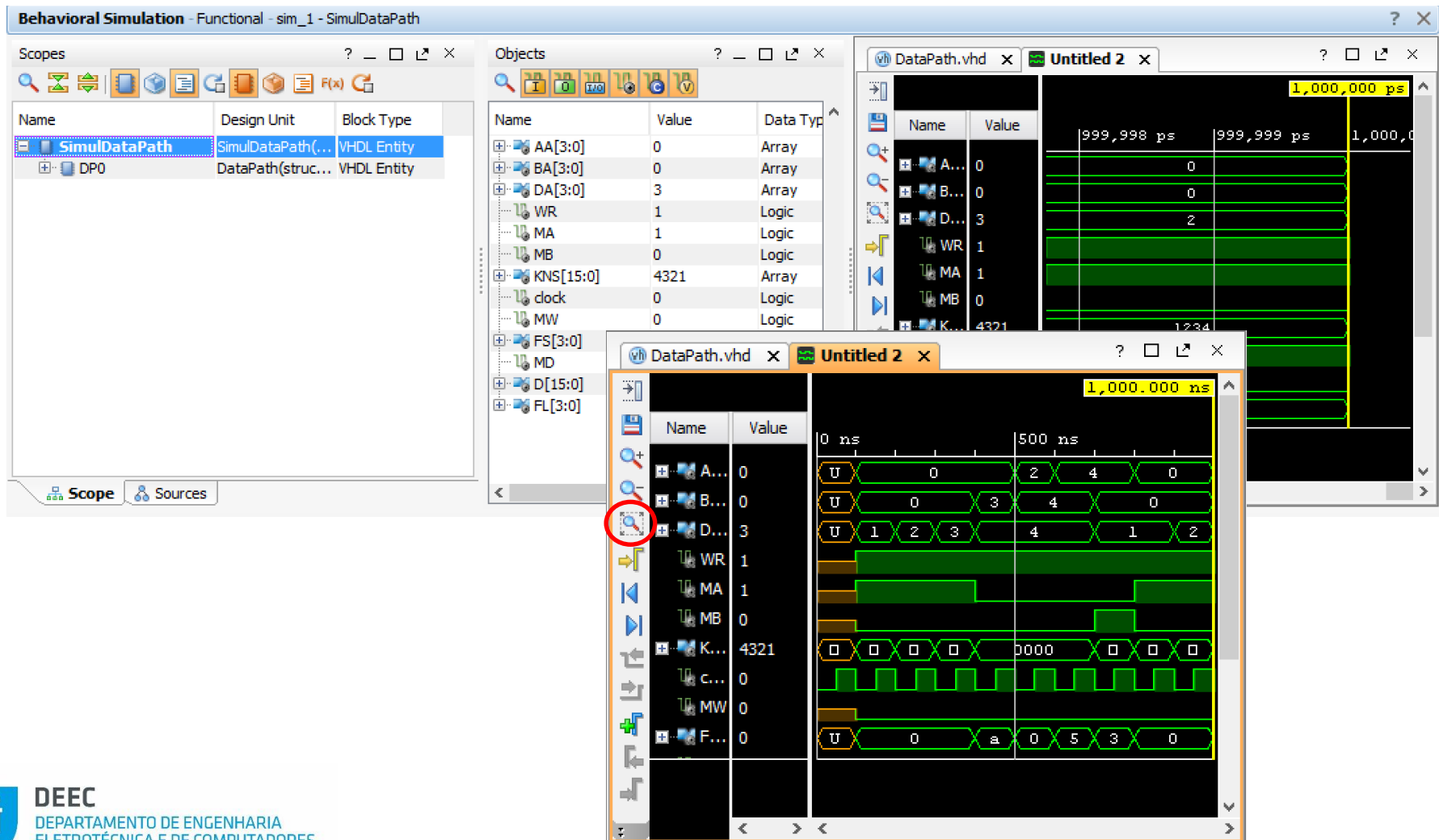
Hierarchy Libraries Compile Order

Scope Sources



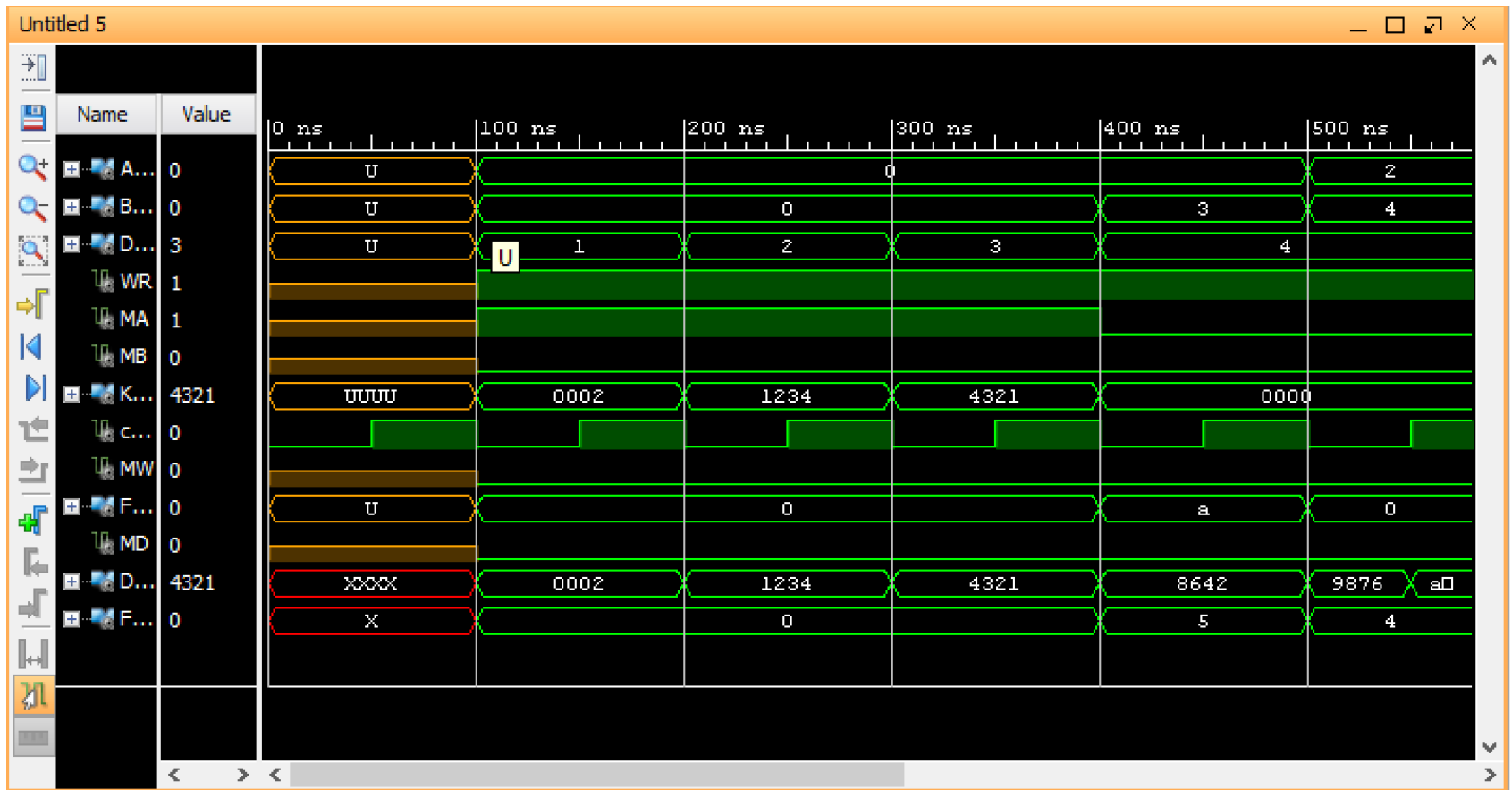
# UNIDADE DE PROCESSAMENTO

## Projeto no VIVADO (Simulação – executar)



## UNIDADE DE PROCESSAMENTO

### Projeto no VIVADO (Simulação – executar)



### • Bibliografia

[1] M. Morris Mano, Charles R. Kime, “Logic and Computer Design Fundamentals”, 5<sup>th</sup> Edition, Prentice-Hall International, 2016.

- Cap. 8: Computer Design Basics

[2] G. Arroz, J. Monteiro, A. Oliveira, “Arquitectura de Computadores: dos Sistemas Digitais aos Microprocessadores”, IST Press, 2009.

### • Outras Referências

[3] J. Hennessy, D. Patterson, “Computer Architecture – A Quantitative Approach”, Morgan Kaufmann, 2007.

[4] D. Patterson, J. Hennessy, “Computer Organization and Design”, Morgan Kaufmann, 2009.

**Nota:** Todas as imagens não referenciadas pertencem à referência principal [1] da disciplina.