

# EA075

## Processadores Dedicados



Faculdade de Engenharia Elétrica e de Computação (FEEC)  
Universidade Estadual de Campinas (UNICAMP)

Prof. Rafael Ferrari

(Documento baseado nas notas de aula do Prof. Levy Boccato)

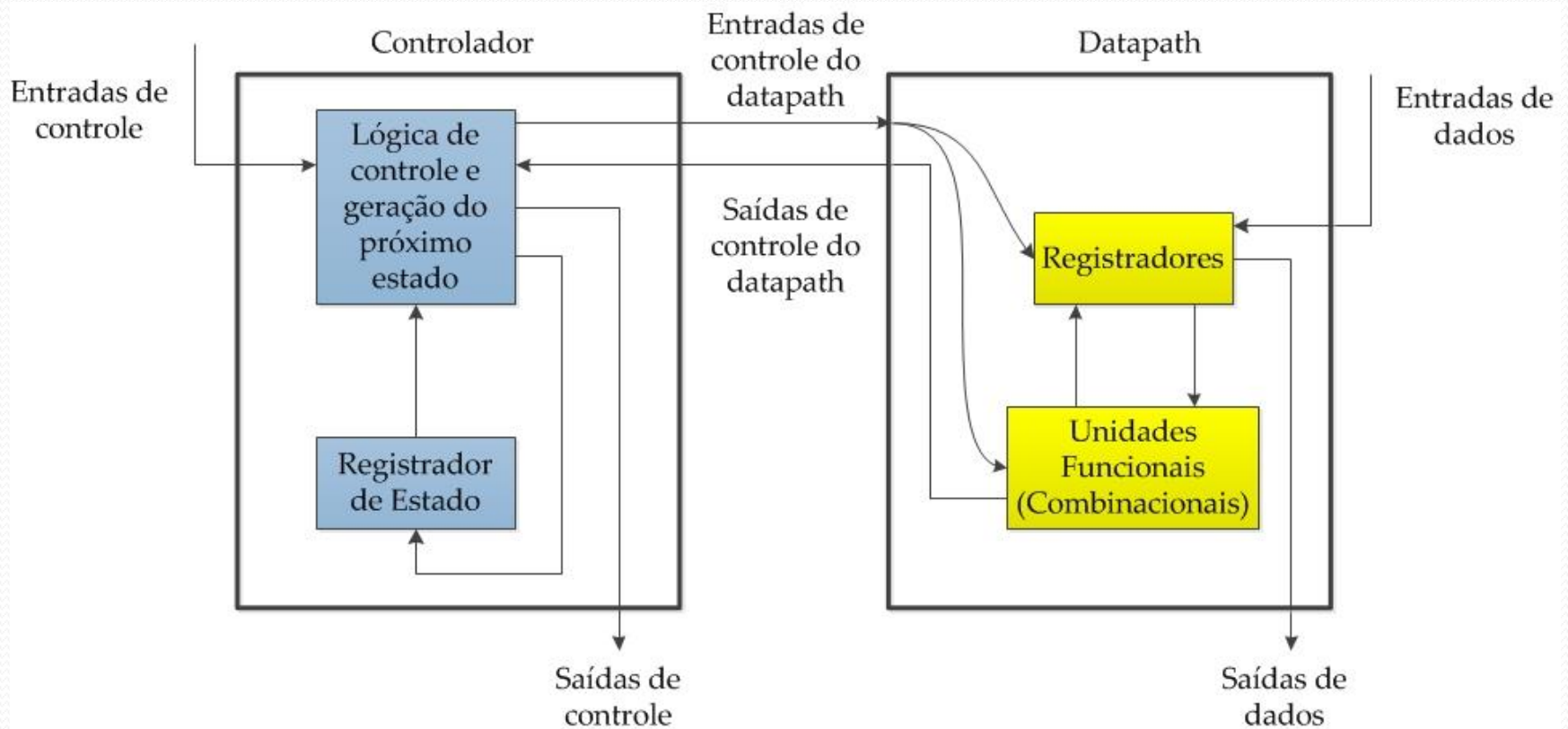
# Introdução

- **Processador:** circuito digital projetado para realizar tarefas de computação.
- **Dois elementos básicos:**
  - *Datapath*: responsável por armazenar e manipular dados.
  - **Controlador (Unidade de controle):** responsável por gerenciar a execução ordenada das operações.

# Introdução

- **Processador Dedicado:** é construído para realizar uma tarefa de computação específica.
- Algumas tarefas são tão comuns que é possível encontrar processadores dedicados padronizados que as implementam (e.g., temporizadores, ADCs, etc.)
- Outras, porém, são particulares para um determinado sistema embarcado.
- Nestes casos, a ideia de projetar o processador surge como uma opção interessante.

# Introdução

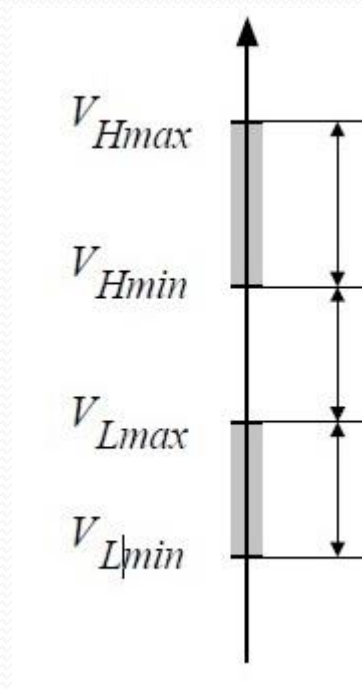


# Motivação

- A opção por efetivamente projetar o processador pode trazer alguns benefícios:
  - Melhor desempenho: mais rápidos
  - Menor ocupação de área no circuito integrado
  - Menor consumo de potência
- *Trade-offs*:
  - Alto custo de projeto do modelo completo do sistema (custo NRE).
  - Tempo para lançamento do dispositivo no mercado será maior.
  - Baixa flexibilidade.
  - Custo por unidade pode ser elevado para produção em baixa escala.

# Revisão – Circuitos Lógicos

- **Sistemas digitais:** são sistemas nos quais os sinais podem assumir um conjunto finito de valores discretos.



Nível alto ( $V_H$ )

Região proibida

Nível baixo ( $V_L$ )

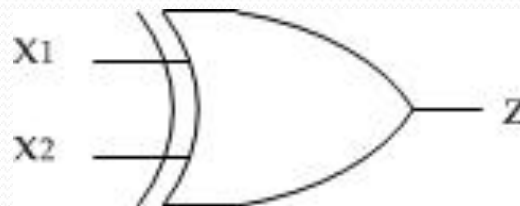
$$\begin{aligned} \text{TTL: } V_{Lmin} &= 0,0\text{V} \\ V_{Lmax} &= 0,8\text{V} \\ V_{Hmin} &= 2,0\text{V} \\ V_{Hmax} &= 5,0\text{V} \end{aligned}$$

# Revisão – Circuitos Lógicos

- **Sistemas combinacionais**

- Em qualquer instante, o nível lógico na saída do sistema corresponde a uma combinação dos níveis lógicos presentes nas entradas (i.e., depende apenas dos valores atuais das entradas).

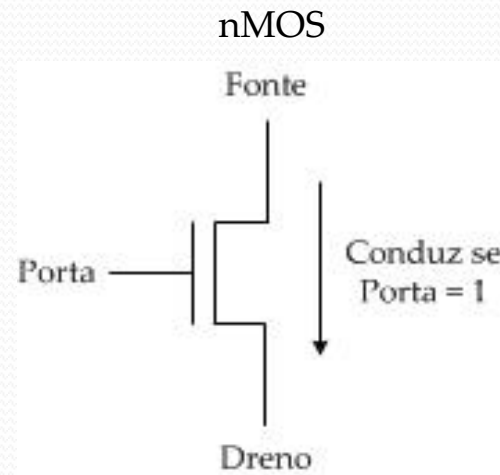
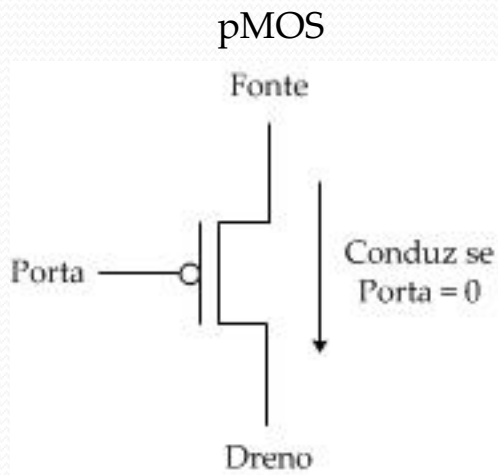
- **Função de saída:**  $z(t) = F(x(t))$



# Revisão – Circuitos Lógicos

- **Elemento básico:** transistor.

- Atua como uma chave liga/desliga.
- A tensão colocada na porta (gate) determina o fluxo de corrente entre a fonte e o dreno.
- CMOS (*complementary metal oxide semiconductor*):

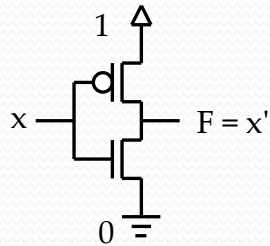


- É possível construir funções lógicas usando transistores pMOS e nMOS.



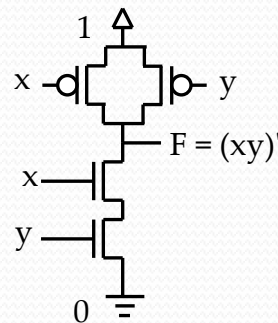
# Revisão – Circuitos Lógicos

- pMOS: conduzem bem tensões altas (nível 1).
- nMOS: conduzem bem tensões baixas (nível 0).
  - Por isso, transistores nMOS são usados na parte do circuito associada ao conjunto-zero de uma função lógica, enquanto transistores pMOS são utilizados no circuito referente ao conjunto-um.
- Exemplos: inversor (NOT), NAND e NOR



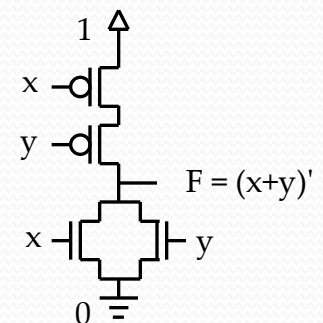
inversor

x	F
0	1
1	0



NAND

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

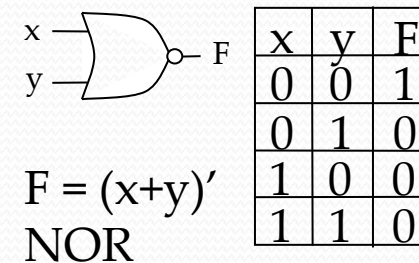
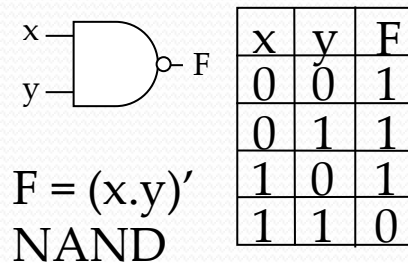
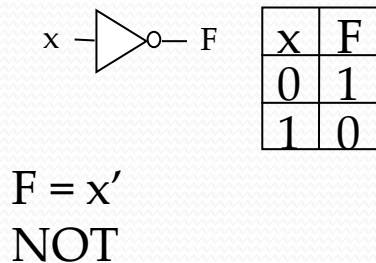
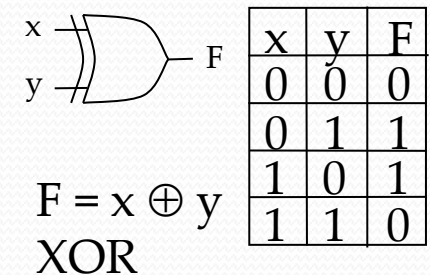
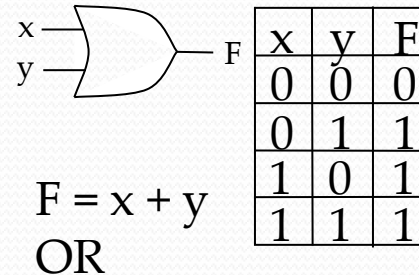
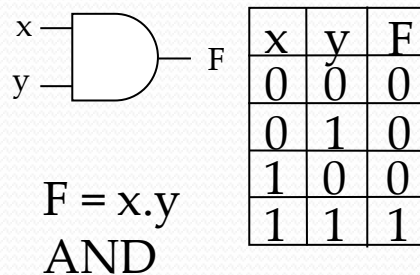


NOR

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

# Revisão – Circuitos Lógicos

- É preferível trabalhar no nível de abstração de portas lógicas.



# Projeto de Lógica Combinacional

- Exemplo

## 1. Descrição do problema

Como as variáveis de saída se comportam em função das variáveis de entrada.

## 2. Tabela verdade

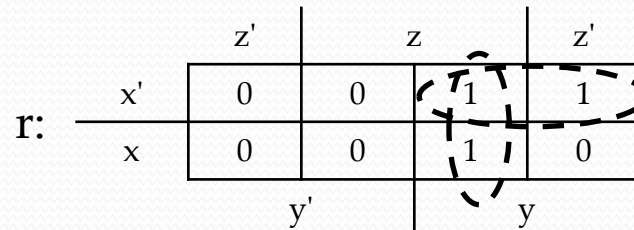
Entradas			Saídas	
x	y	z	r	s
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

## 3. Expressões booleanas

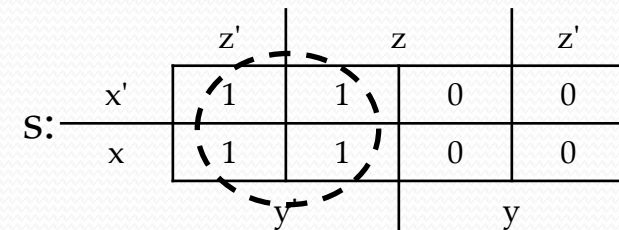
$$r = x'y'z' + x'yz + xyz$$

$$s = x'y'z' + x'y'z + xy'z' + xy'z$$

## 4. Minimização das equações de saídas

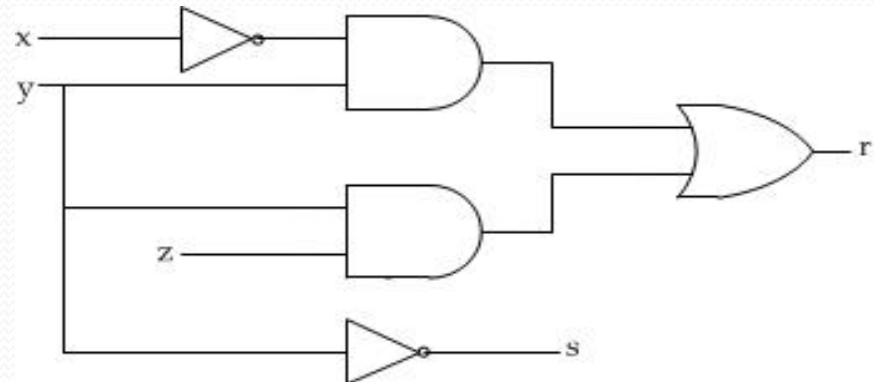


$$r = x'y + yz$$

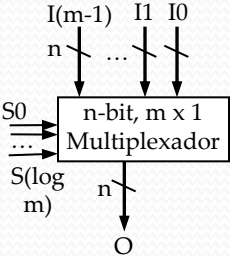
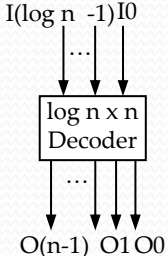
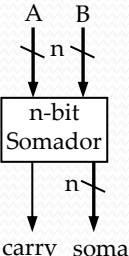
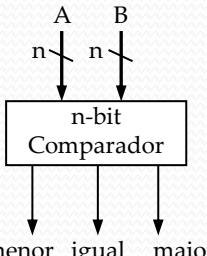
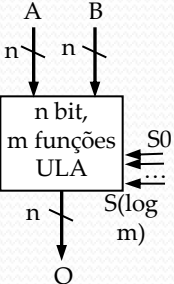


$$s = y'$$

## 5. Circuito final

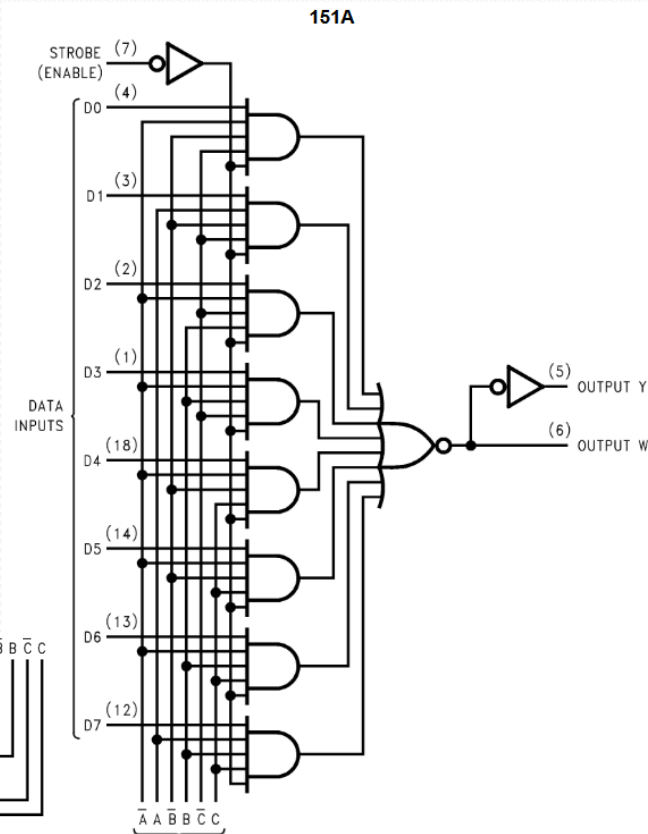


# Componentes Combinacionais

				
<p> <math>O =</math>  <math>I_0</math> se <math>S=0..00</math>  <math>I_1</math> se <math>S=0..01</math>                      ...  <math>I_{(m-1)}</math> se <math>S=1..11</math> </p>	<p> <math>O_0 = 1</math> se <math>I=0..00</math>  <math>O_1 = 1</math> se <math>I=0..01</math>                      ...  <math>O_{(n-1)} = 1</math> se <math>I=1..11</math> </p>	<p>                     soma = <math>A+B</math>                      (primeiros <math>n</math> bits)                       carry (vai um) =                      (<math>n+1</math>)'ésimo bit de <math>A+B</math> </p>	<p>                     menor = 1 se <math>A &lt; B</math>                      igual = 1 se <math>A = B</math>                      maior = 1 se <math>A &gt; B</math> </p>	<p> <math>O = A \text{ op } B</math>                      op determinado por <math>S</math>.                 </p>
	<p>                     Com entrada de habilitação (enable) e <math>\rightarrow</math> todos <math>O</math>'s são 0 se <math>e=0</math> </p>	<p>                     Com entrada de carry-in <math>C_i \rightarrow</math>                      soma = <math>A + B + C_i</math> </p>		<p>                     Pode possuir saídas de status, carry, zero, etc.                 </p>

# Multiplexadores

- Seleciona e conecta uma de suas  $n$  entradas à saída.
- A escolha é feita através de  $\log_2(n)$  entradas de seleção.
- Exemplo: 74151 – 1bit,  $8 \times 1$  multiplexer

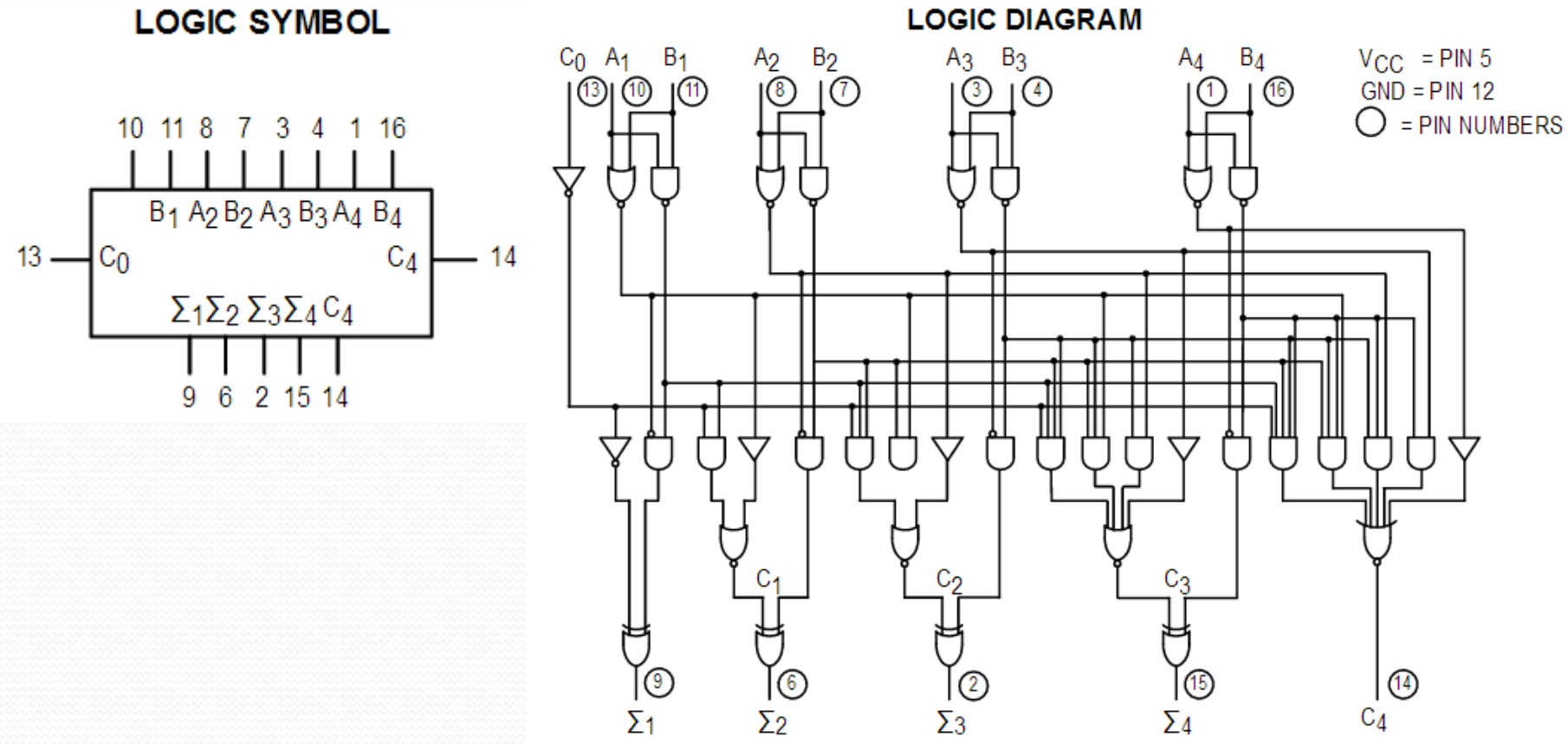


Inputs			Outputs		
Select			Strobe S	Y	W
C	B	A			
X	X	X	H	L	H
L	L	L	L	D0	$\overline{D0}$
L	L	H	L	D1	$\overline{D1}$
L	H	L	L	D2	$\overline{D2}$
L	H	H	L	D3	$\overline{D3}$
H	L	L	L	D4	$\overline{D4}$
H	L	H	L	D5	$\overline{D5}$
H	H	L	L	D6	$\overline{D6}$
H	H	H	L	D7	$\overline{D7}$



# Somadores

- Exemplo: 7483 – somador de 4 bits com *carry*

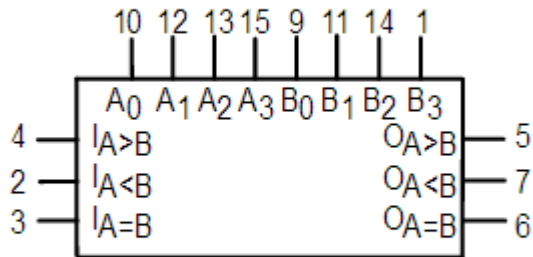


$$C_0 + (A_1+B_1)+2(A_2+B_2)+4(A_3+B_3)+8(A_4+B_4) = \Sigma_1+2\Sigma_2+4\Sigma_3+8\Sigma_4+16C_4$$

# Comparadores

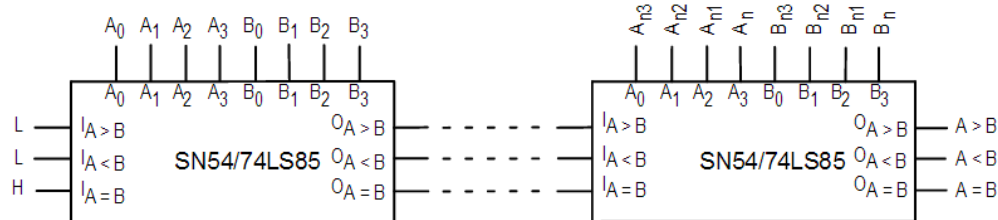
- Exemplo: 7485 – comparador de 4 bits

## LOGIC SYMBOL



$V_{CC}$  = PIN 16  
GND = PIN 8

COMPARING INPUTS				CASCADING INPUTS			OUTPUTS		
$A_3, B_3$	$A_2, B_2$	$A_1, B_1$	$A_0, B_0$	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$O_{A>B}$	$O_{A<B}$	$O_{A=B}$
$A_3 > B_3$	X	X	X	X	X	X	H	L	L
$A_3 < B_3$	X	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	L	L	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	H	L	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	X	X	H	L	L	H
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	H	L	L	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	L	L	H	H	L

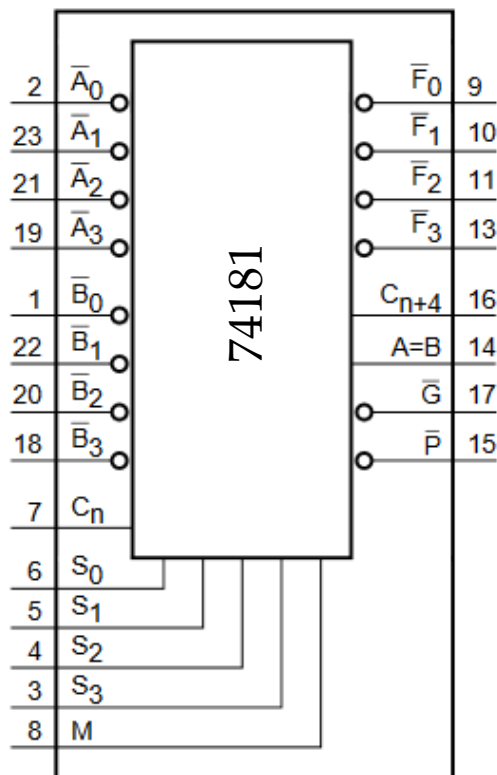


L = LOW LEVEL  
H = HIGH LEVEL



# Unidades lógico-aritméticas (ULA)

- Permite a execução de diversas operações lógicas e aritméticas em seus operandos de entrada.
- Exemplo: 74181 – 4-bit ULA



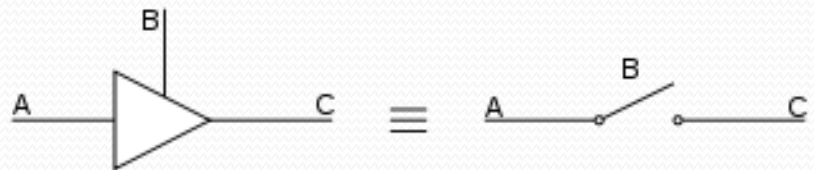
MODE SELECT INPUTS				ACTIVE HIGH INPUTS AND OUTPUTS	
S <sub>3</sub>	S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>	LOGIC (M=H)	ARITHMETIC <sup>(2)</sup> (M=L; C <sub>n</sub> =H)
L	L	L	L	$\bar{A}$	A
L	L	L	H	$\overline{A+B}$	A + B
L	L	H	L	$\overline{AB}$	A + $\bar{B}$
L	L	H	H	logical 0	minus 1
L	H	L	L	$\overline{AB}$	A plus $\overline{AB}$
L	H	L	H	$\bar{B}$	(A + B) plus $\overline{AB}$
L	H	H	L	A $\oplus$ B	A minus B minus 1
L	H	H	H	$\overline{AB}$	$\overline{AB}$ minus 1
H	L	L	L	$\overline{A+B}$	A plus AB
H	L	L	H	$\overline{A \oplus B}$	A plus B
H	L	H	L	B	(A + $\bar{B}$ ) plus AB
H	L	H	H	AB	AB minus 1
H	H	L	L	logical 1	A plus A <sup>(1)</sup>
H	H	L	H	A + $\bar{B}$	(A + B) plus A
H	H	H	L	A + B	(A + $\bar{B}$ ) plus A
H	H	H	H	A	A minus 1

MODE SELECT INPUTS				ACTIVE LOW INPUTS AND OUTPUTS	
S <sub>3</sub>	S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>	LOGIC (M=H)	ARITHMETIC <sup>(2)</sup> (M=L; C <sub>n</sub> =L)
L	L	L	L	$\bar{A}$	A minus 1
L	L	L	H	$\overline{AB}$	$\overline{AB}$ minus 1
L	L	H	L	$\overline{A+B}$	$\overline{A+B}$ minus 1
L	L	H	H	logical 1	minus 1
L	H	L	L	$\overline{A+B}$	A plus (A + $\bar{B}$ )
L	H	L	H	$\bar{B}$	$\overline{AB}$ plus (A + $\bar{B}$ )
L	H	H	L	$\overline{A \oplus B}$	A minus B minus 1
L	H	H	H	A + $\bar{B}$	A + $\bar{B}$
H	L	L	L	$\overline{AB}$	A plus (A + B)
H	L	L	H	A $\oplus$ B	A plus B
H	L	H	L	B	$\overline{AB}$ plus (A + B)
H	L	H	H	A + B	A + B
H	H	L	L	logical 0	A plus A <sup>(1)</sup>
H	H	L	H	$\overline{AB}$	$\overline{AB}$ plus A
H	H	H	L	AB	$\overline{AB}$ plus A
H	H	H	H	A	A

# Buffer tri-state

- Opera de modo equivalente a uma chave.
- Pode ser usado para viabilizar a conexão de várias saídas a uma mesma entrada.
- Pode ser usado para aumentar a corrente máxima fornecida por uma saída (driver).

B	A	C
0	X	Alta impedância
1	0	0
1	1	1



# Revisão – Circuitos Lógicos

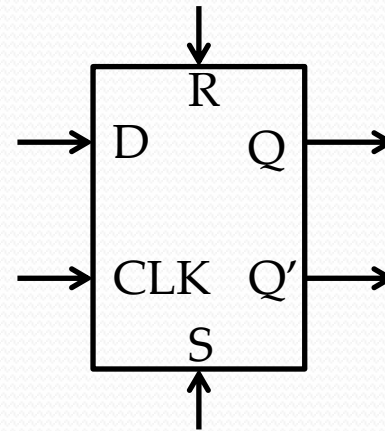
- **Sistemas sequenciais**

- Em qualquer instante de tempo  $t$ , a saída é uma função dos níveis lógicos das entradas atuais, bem como de seus valores em instantes anteriores.
- A dependência temporal do sistema é modelada por meio de uma máquina de estados finitos, de modo que o comportamento do sistema sequencial passa a ser descrito por meio das seguintes funções:
  - Função de transição de estado:  $s(t+1) = G(x(t),s(t))$
  - Função de saída:  $z(t) = H(s(t),x(t))$

# Flip Flop

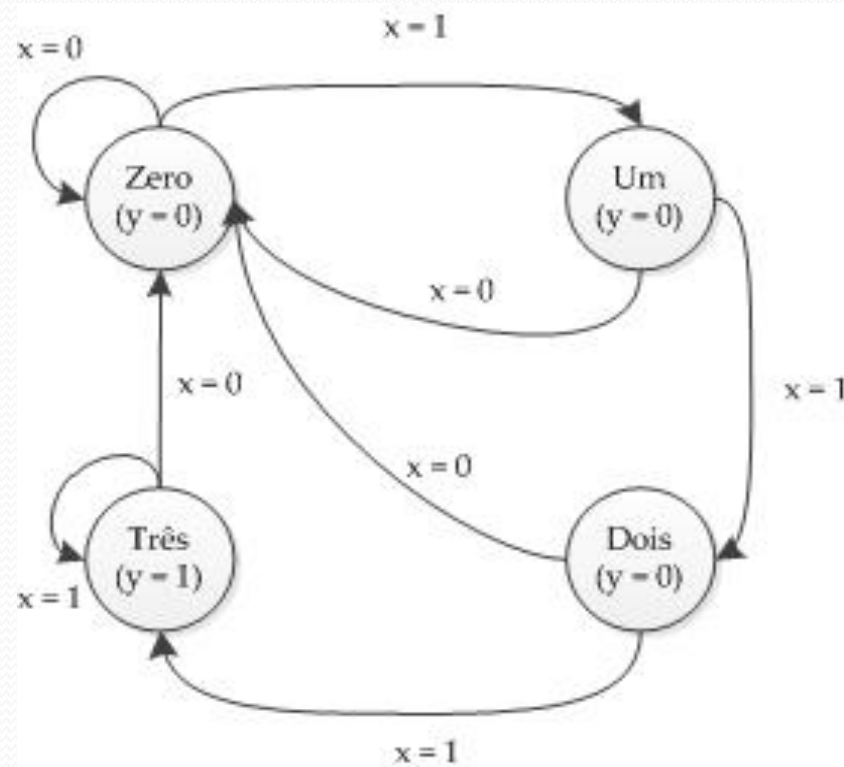
- Elemento de memória: circuito combinacional com **realimentação**.
- Flip Flop D:

R	S	CLK	D	Q
0	0	↑	0	0
0	0	↑	1	1
0	0	not rising	X	Qa
1	0	X	X	0
0	1	X	X	1



# Projeto de Lógica Sequencial

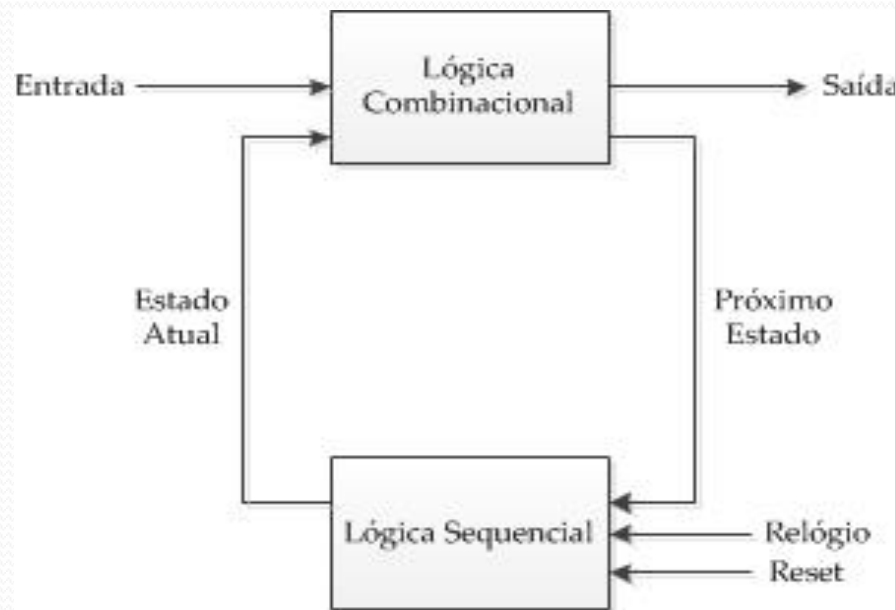
- Projeto via máquina de estados finitos
- Ex.: Detector de sequência '111'



# Projeto de Lógica Sequencial

- **Forma canônica**

- Flip-flops / Registrador de estado: armazena o estado atual.
- Circuito combinacional determina o próximo estado e as saídas.



# Projeto de Lógica Sequencial

- **Etapas do projeto**

- Descrição do problema (variáveis de entrada e saída);
  - Elaboração de um diagrama de estados que representa o sistema;
  - Definição de um modelo de implementação baseado na forma canônica;
  - Criação de uma tabela verdade que relaciona o estado atual e as entradas com as saídas e o próximo estado;
  - A partir desse ponto, o processo se resume ao projeto da lógica combinacional que produzirá a(s) saída(s) e o próximo estado.
- 
- Exemplo: projeto de um controlador de motor de passo

# Motor de passo

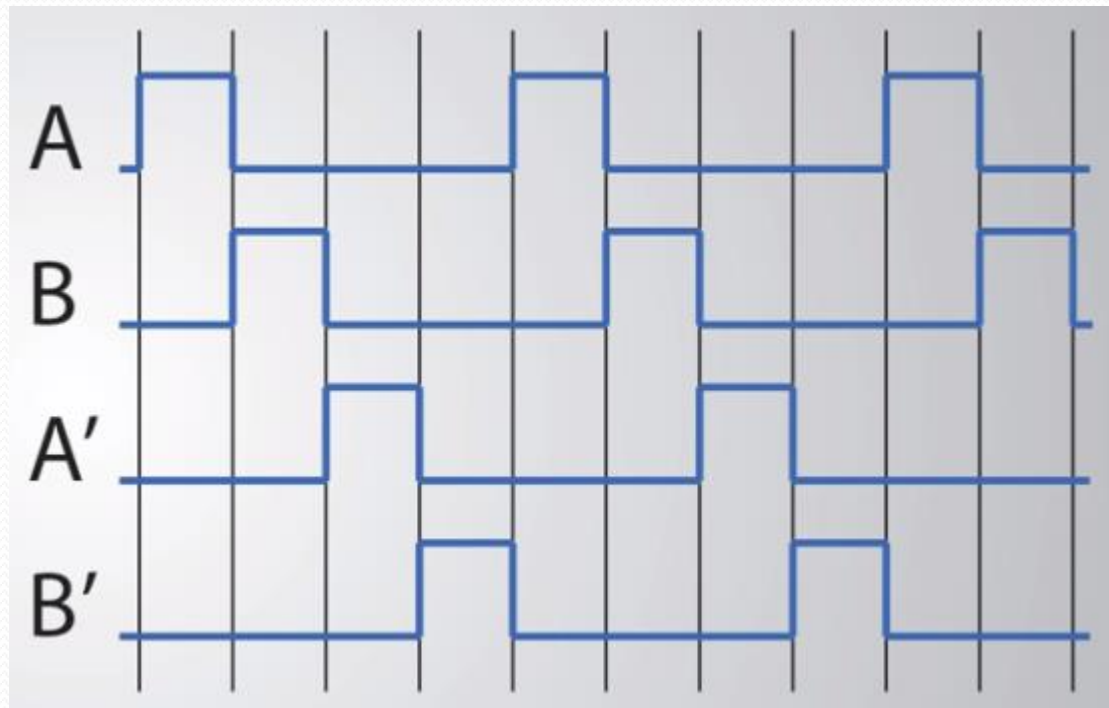
- É um motor de corrente contínua (CC) sem escovas que movimentam seu rotor em passos discretos.
- Pode ser precisamente posicionado em malha aberta, ou seja, não requer controle com realimentação.
- Muito usado em sistemas de posicionamento: leitores de CD/DVD, scanners, impressoras, plotters, máquinas CNC, impressoras 3D, etc.
- Vídeo que ilustra seu funcionamento:

<https://www.youtube.com/watch?v=TWMai3oirnM&spfreload=10>



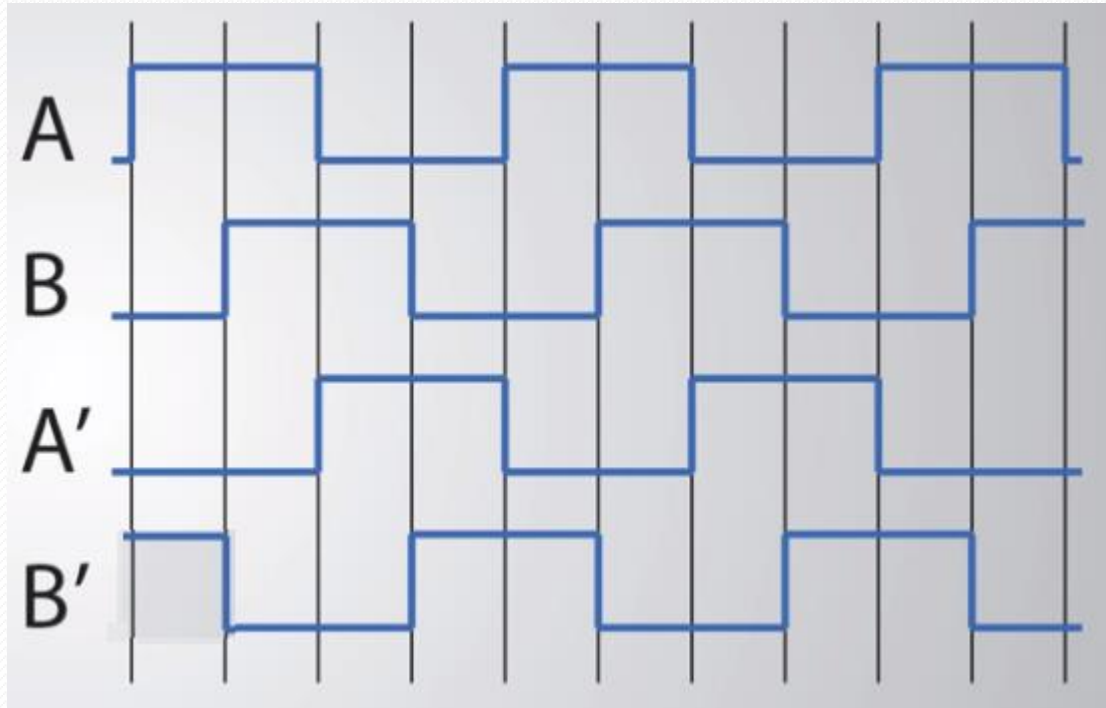
# Motor de passo

- Modos de acionamento:
  - Passo completo (*full step*) com uma única fase ativa por passo:



# Motor de passo

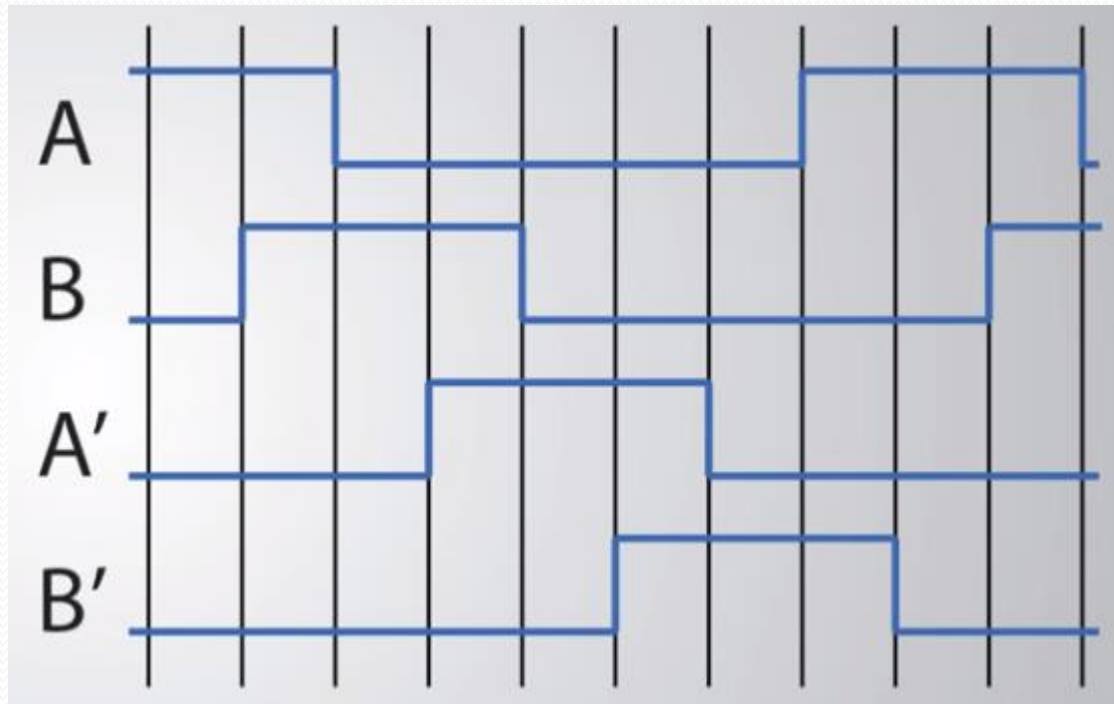
- Modos de acionamento:
  - Passo completo (*full step*) com duas fases ativas por passo:



- Maior torque.

# Motor de passo

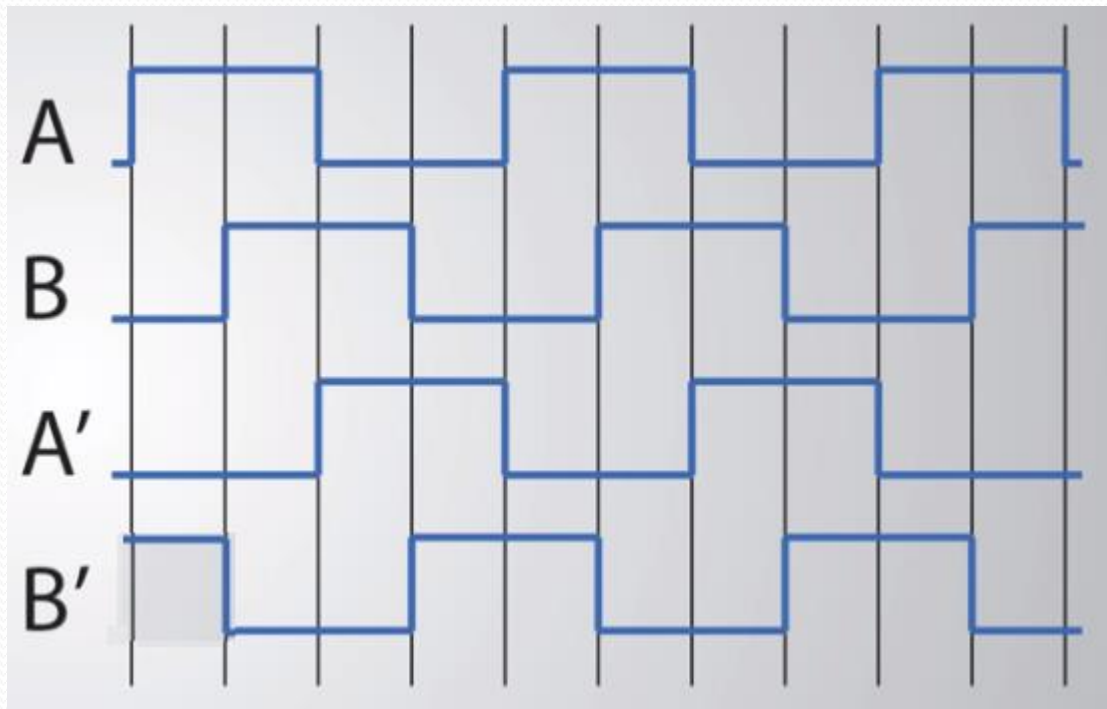
- Modos de operação acionamento:
  - Meio passo (*half step*) que combina o acionamento com uma e duas bobinas:



- Maior resolução.

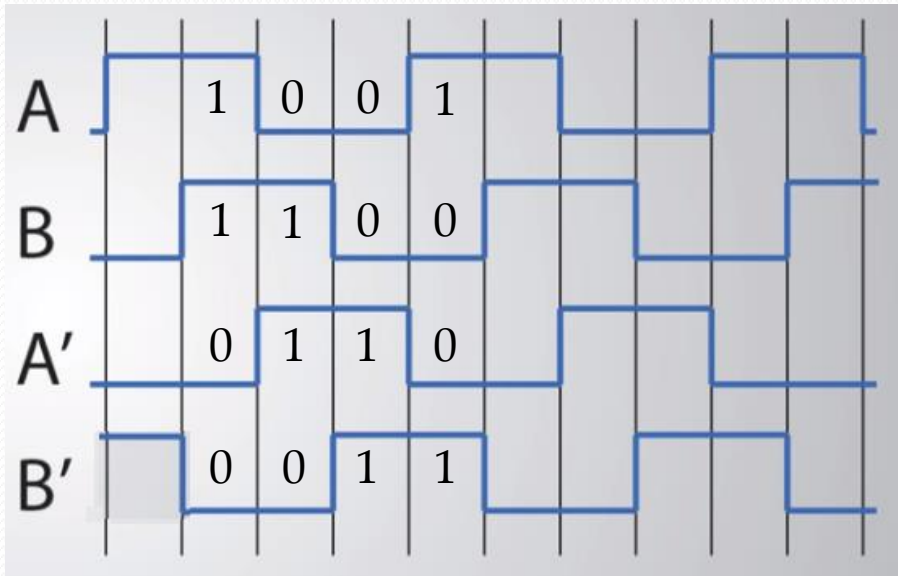
# Controlador de motor de passo

- Projete um controlador que gere os sinais necessários para o acionamento de um motor de passo no modo passo completo com duas fases ativas por passo, de acordo com o diagrama de tempo abaixo.

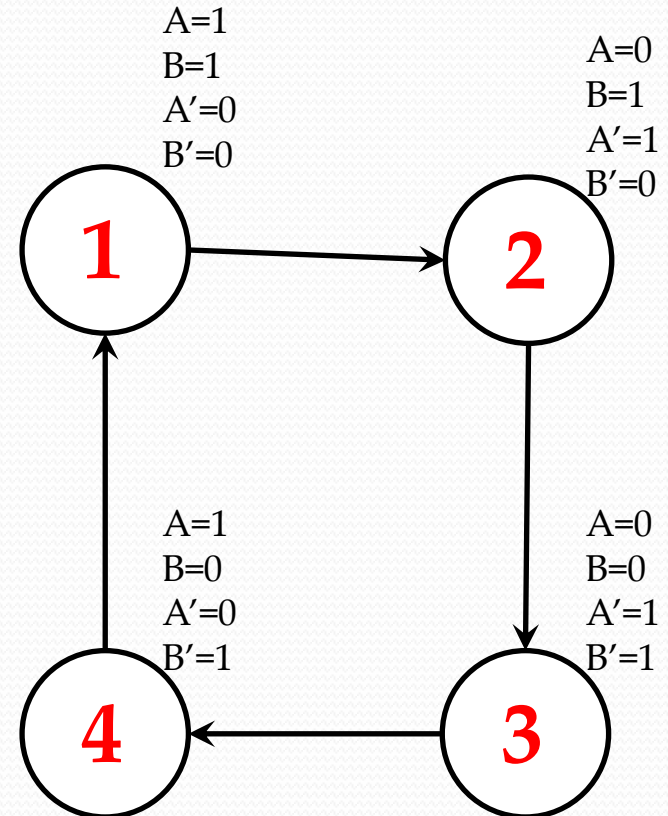


# Controlador de motor de passo

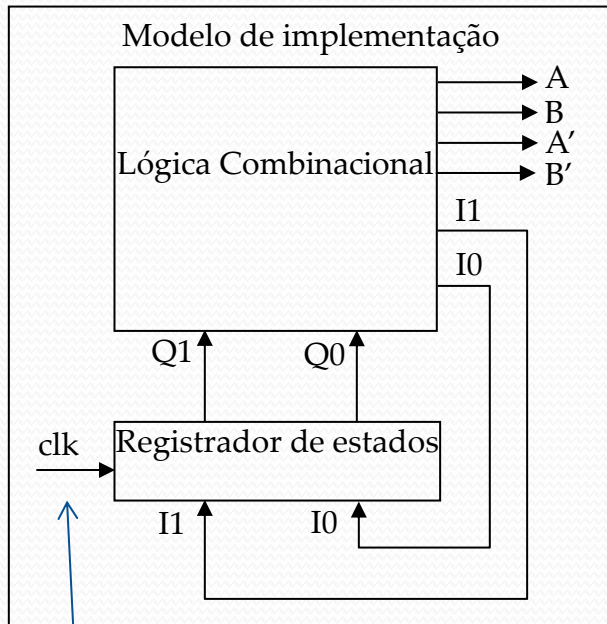
1 2 3 4 1 2 3 4 1 2



Estado	Q1	Q0
1	0	0
2	0	1
3	1	0
4	1	1



# Controlador de motor de passo



Q1	Q0	I1	I0	A	B	A'	B'
0	0	0	1	1	1	0	0
0	1	1	0	0	1	1	0
1	0	1	1	0	0	1	1
1	1	0	0	1	0	0	1

A cada borda de subida do relógio (clk) ocorre uma transição de estado que resulta em um passo do motor.

# Controlador de motor de passo

- Expressões simplificadas das saídas do circuito combinacional:

	Q0'	Q0
Q1'	0	1
Q1	1	0

$$I1 = Q1'Q0 + Q1Q0'$$

	Q0'	Q0
Q1'	1	1
Q1	0	0

$$B = Q1'$$

	Q0'	Q0
Q1'	1	0
Q1	1	0

$$I0 = Q0'$$

	Q0'	Q0
Q1'	0	1
Q1	1	0

$$A' = Q1'Q0 + Q1Q0' = I1$$

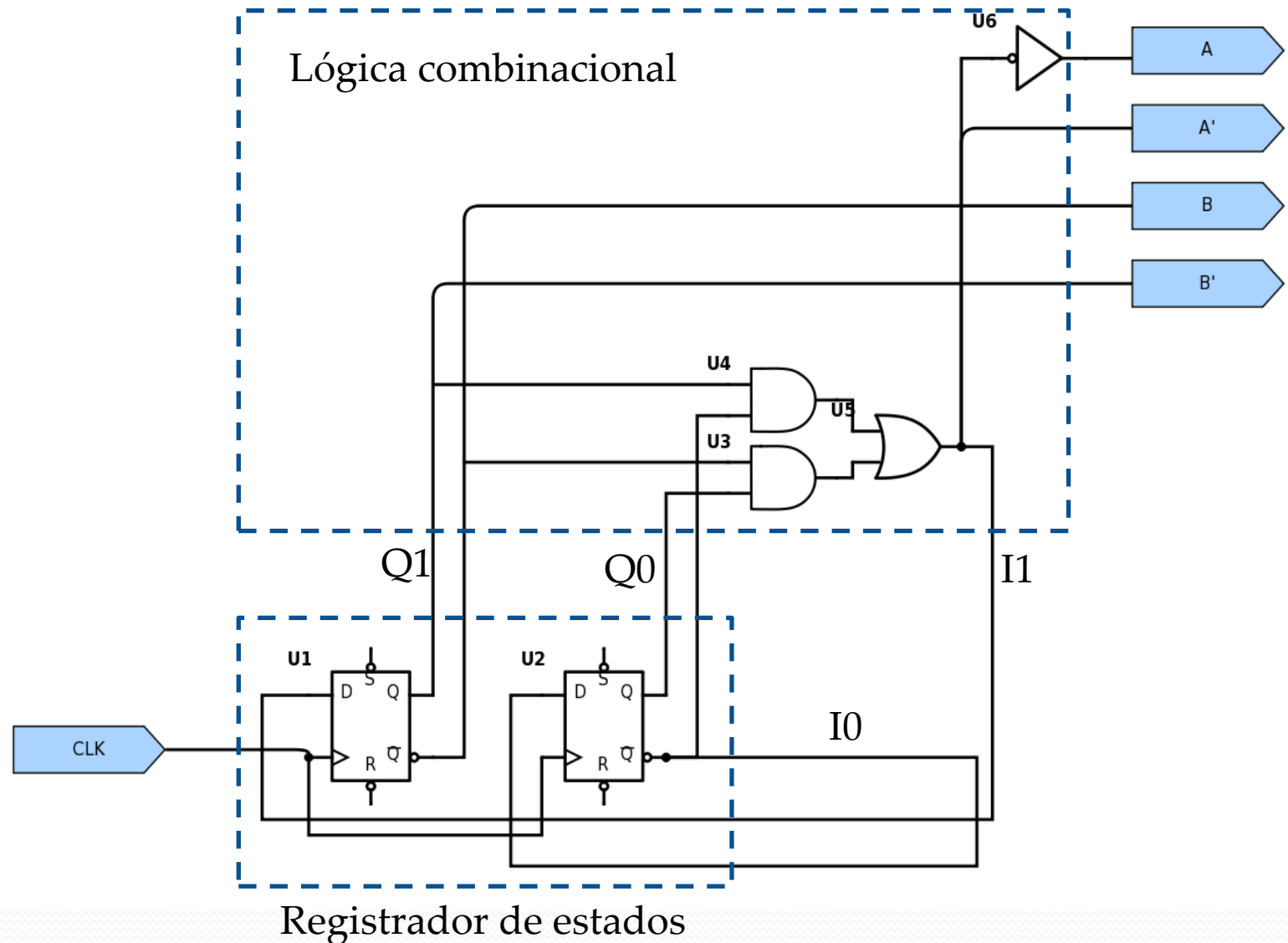
	Q0'	Q0
Q1'	1	0
Q1	0	1

$$A = Q1'Q0' + Q1Q0 = I1'$$

	Q0'	Q0
Q1'	0	0
Q1	1	1

$$B' = Q1$$

# Controlador de motor de passo





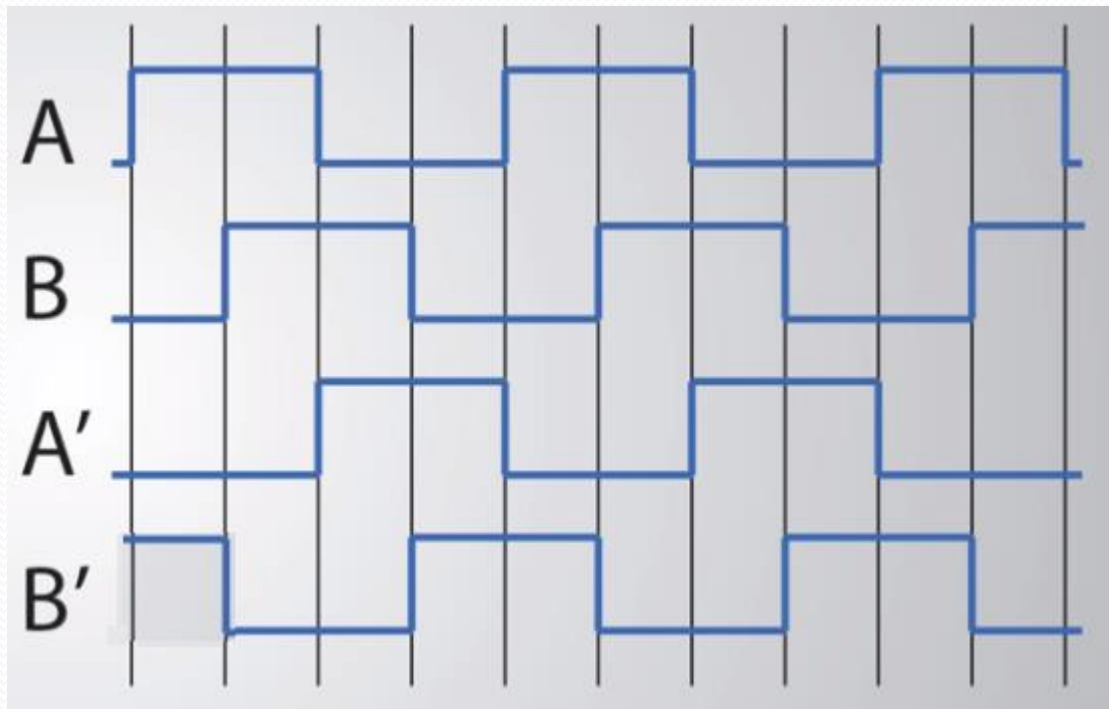
# Controlador de motor de passo

- Um circuito digital geralmente não fornece a corrente (tensão) necessária para acionamento de um motor.
- É necessário usar algum tipo de amplificador (driver) analógico.
- No caso de motores de passo bipolares, é necessário empregar pontes H.
- Exemplo: L293D – dupla ponte H de baixa potência (corrente máxima de 600mA)
- Livro interessante sobre o assunto (ebook disponível para download gratuito na rede da Unicamp):
  - S. Ball, *“Analog Interfacing to Embedded Microprocessor Systems”*, Elsevier, 2003.

<https://www.elsevier.com/books/analog-interfacing-to-embedded-microprocessor-systems/ball/978-0-7506-7723-3>

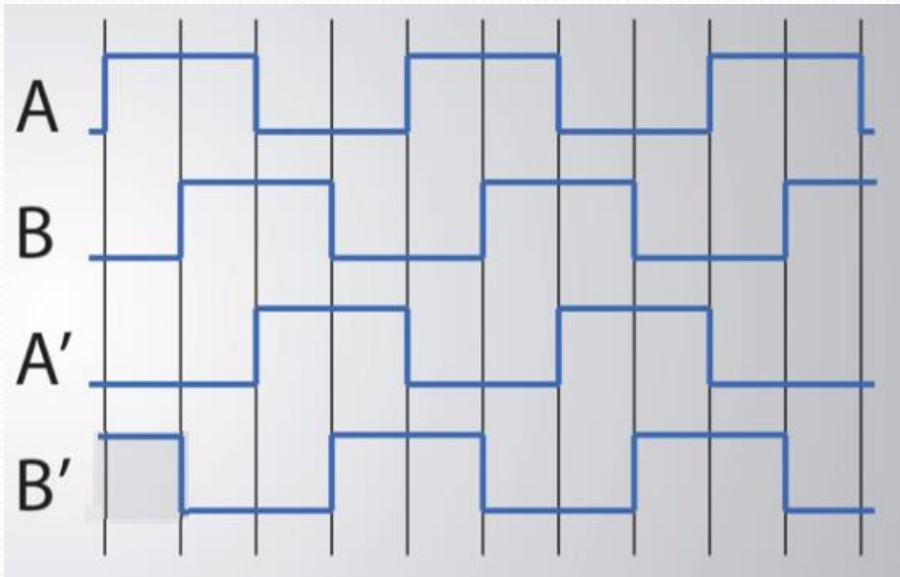
# Controlador de motor de passo

- Projete um controlador que gere os sinais necessários para o acionamento de um motor de passo no modo passo completo com duas fases ativas por passo, de acordo com o diagrama de tempo abaixo. Acrescente um sinal que permita o controle do sentido de rotação do motor.



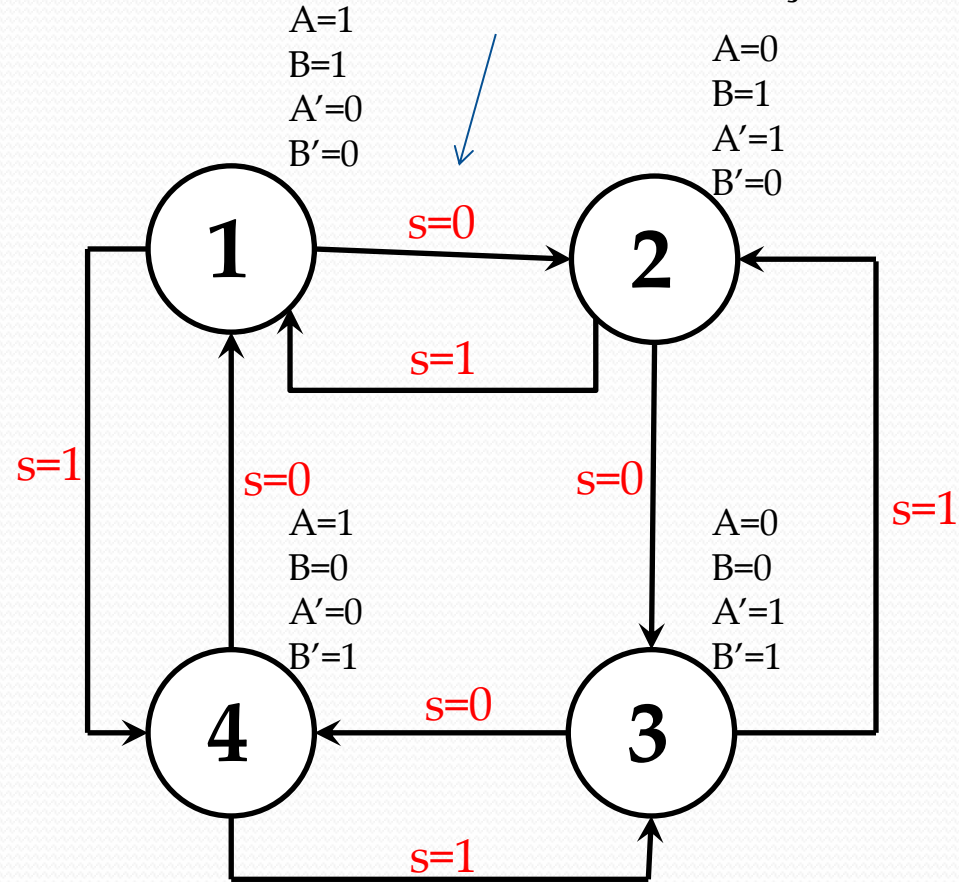
# Controlador de motor de passo

1 2 3 4 1 2 3 4 1 2

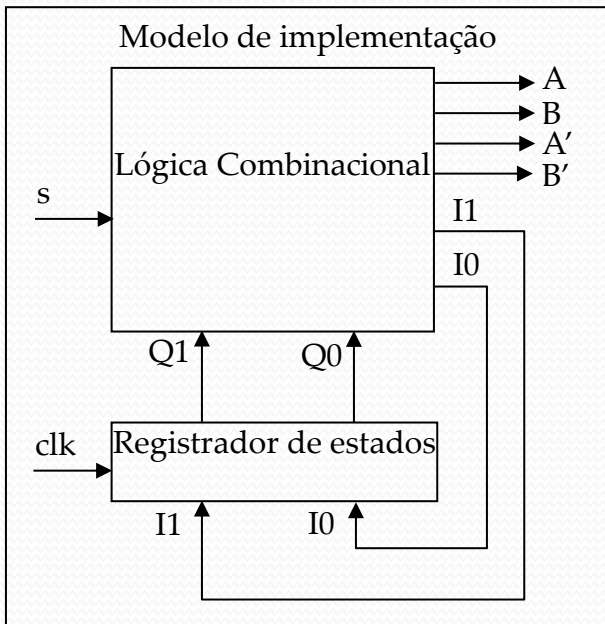


Estado	Q1	Q0
1	0	0
2	0	1
3	1	0
4	1	1

s: entrada que controla o sentido de rotação



# Controlador de motor de passo



s	Q1	Q0	I1	I0	A	B	A'	B'
0	0	0	0	1	1	1	0	0
0	0	1	1	0	0	1	1	0
0	1	0	1	1	0	0	1	1
0	1	1	0	0	1	0	0	1
1	0	0	1	1	1	1	0	0
1	0	1	0	0	0	1	1	0
1	1	0	0	1	0	0	1	1
1	1	1	1	0	1	0	0	1

# Controlador de motor de passo

- Expressões simplificadas das saídas do circuito combinacional:

$$I1 = s'Q1'Q0' + s'Q1Q0' + sQ1'Q0' + sQ1Q0$$

$$I0 = Q0'$$

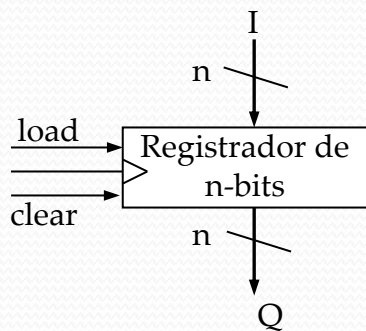
$$A = Q1'Q0' + Q1Q0$$

$$B = Q1'$$

$$A' = Q1'Q0 + Q1Q0'$$

$$B' = Q1$$

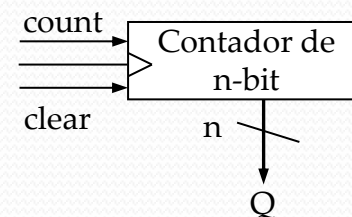
# Componentes Sequenciais



Q =  
0 if clear=1,  
I se load=1 e clock= $\uparrow$ ,  
Q(anterior) caso contrário.



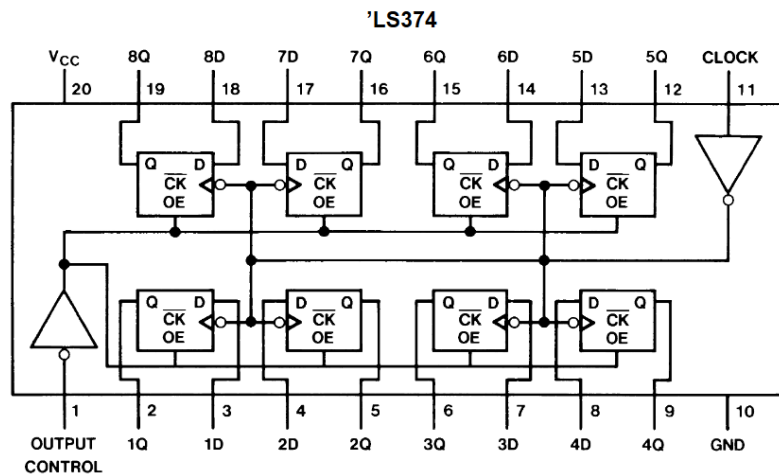
Q = lsb (bit menos significativo)  
- Conteúdo deslocado  
- I armazenado no msb (bit mais significativo)



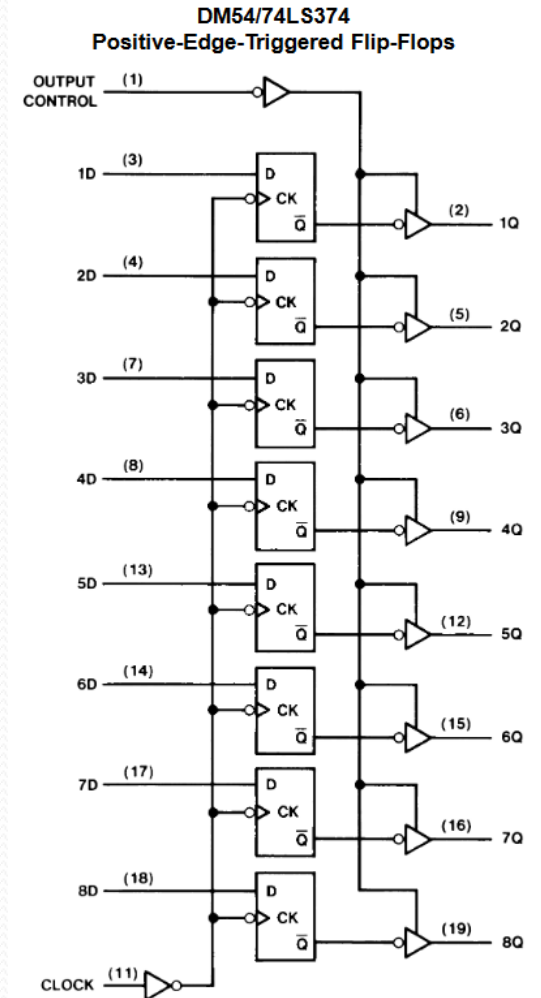
Q =  
0 se clear=1,  
Q(anterior)+1 se count=1 e clock= $\uparrow$ .

# Registadores

- Exemplo: 74374 – registrador de 8 bits

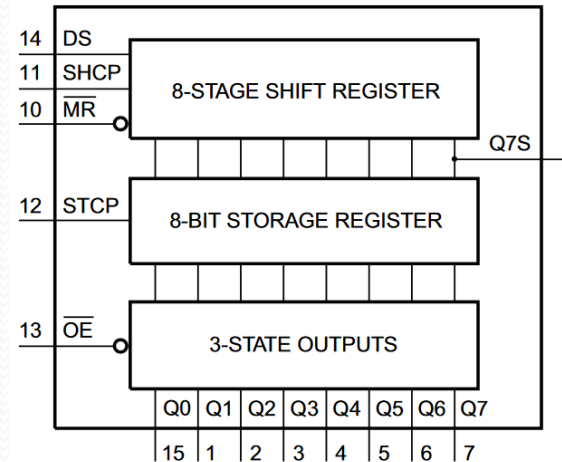
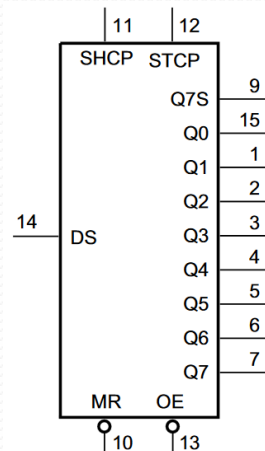


Output Control	Clock	D	Output
L	↑	H	H
L	↑	L	L
L	L	X	$Q_0$
H	X	X	Z



# Registadores de deslocamento

- Exemplo: 74595 – registrador de deslocamento de 8 bits

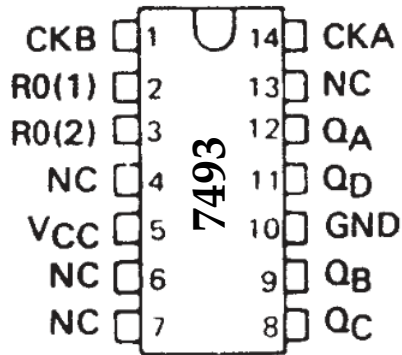


Control				Input	Output	Function	
SHCP	STCP	$\overline{OE}$	$\overline{MR}$	DS	Q7S	Qn	
X	X	L	L	X	L	NC	a LOW-level on $\overline{MR}$ only affects the shift registers
X	↑	L	L	X	L	L	empty shift register loaded into storage register
X	X	H	L	X	L	Z	shift register clear; parallel outputs in high-impedance OFF-state
↑	X	L	H	H	Q6S	NC	logic HIGH-level shifted into shift register stage 0. Contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6S) appears on the serial output (Q7S).
X	↑	L	H	X	NC	QnS	contents of shift register stages (internal QnS) are transferred to the storage register and parallel output stages
↑	↑	L	H	X	Q6S	QnS	contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages



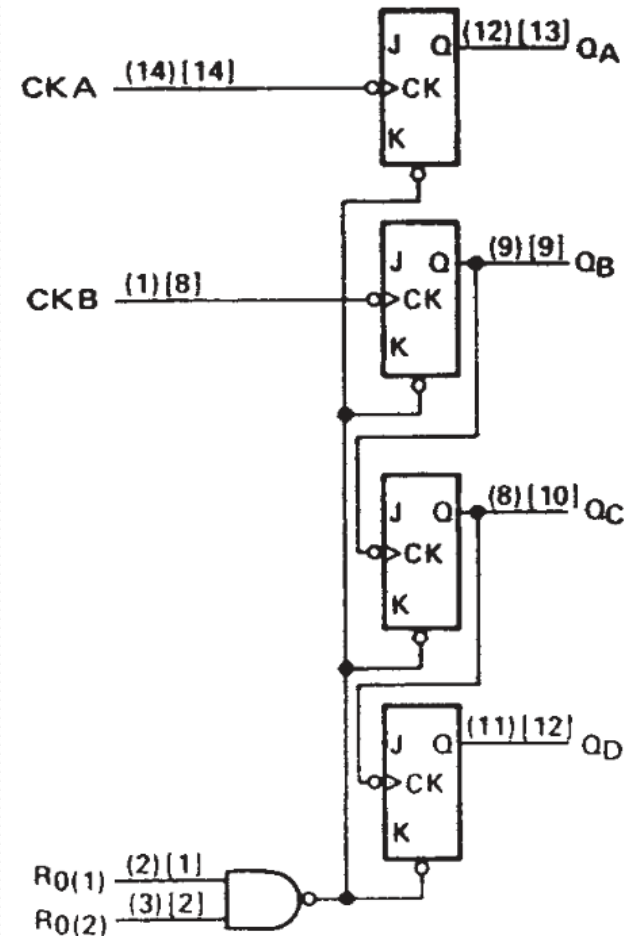
# Contadores

- Exemplo: 7493 – contador binário de 4 bits



RESET INPUTS		OUTPUT			
R <sub>0</sub> (1)	R <sub>0</sub> (2)	Q <sub>D</sub>	Q <sub>C</sub>	Q <sub>B</sub>	Q <sub>A</sub>
H	H	L	L	L	L
L	X	COUNT			
X	L	COUNT			

COUNT	OUTPUT			
	Q <sub>D</sub>	Q <sub>C</sub>	Q <sub>B</sub>	Q <sub>A</sub>
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H
10	H	L	H	L
11	H	L	H	H
12	H	H	L	L
13	H	H	L	H
14	H	H	H	L
15	H	H	H	H



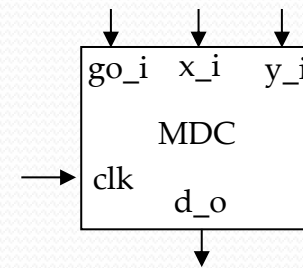
# Projeto de Processadores Dedicados

- **Estratégia de projeto:**

**1** Inicialmente, preparamos um programa sequencial que descreve a tarefa de computação que desejamos implementar.

- **Exemplo: Cálculo do máximo divisor comum**

```
0: int x, y;  
1: while (1) {  
2:   while (!go_i);  
3:   x = x_i;  
4:   y = y_i;  
5:   while (x != y) {  
6:     if (x < y)  
7:       y = y - x;  
8:     else  
9:       x = x - y;  
   }  
9:   d_o = x;  
}
```



# Projeto de Processadores Dedicados

2

Convertemos o programa em um diagrama de estados, no qual os estados e os arcos podem incluir expressões aritméticas, e tais expressões podem utilizar entradas e saídas externas bem como variáveis.

- Este diagrama é chamado de **máquina de estados finitos com datapath (FSMD)**.
- Para isto, usamos alguns modelos básicos de conversão de comandos típicos de linguagens de programação em um diagrama de transição de estados.

# Projeto de Processadores Dedicados

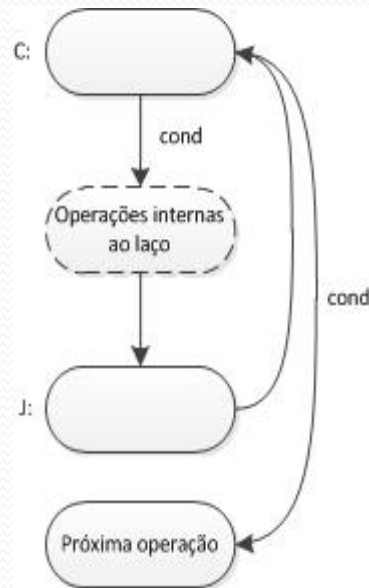
## Atribuição de valor a uma variável

A = operação(B)  
Próxima operação



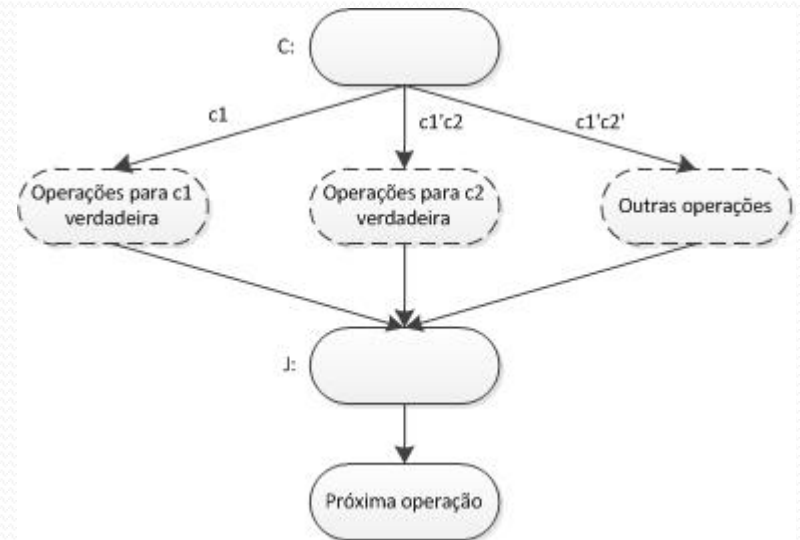
## Laço de repetição

Enquanto (cond) {  
Operações internas ao laço  
}  
Próxima operação



## Desvio condicional

Se (c1)  
Operações para c1 verdadeira  
Senão, se (c2)  
Operações para c2 verdadeira  
Senão,  
Outras operações  
Próxima operação



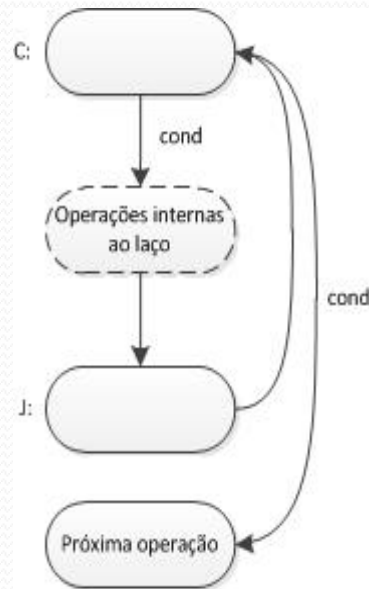
# Projeto de Processadores Dedicados

## Atribuição de valor a uma variável

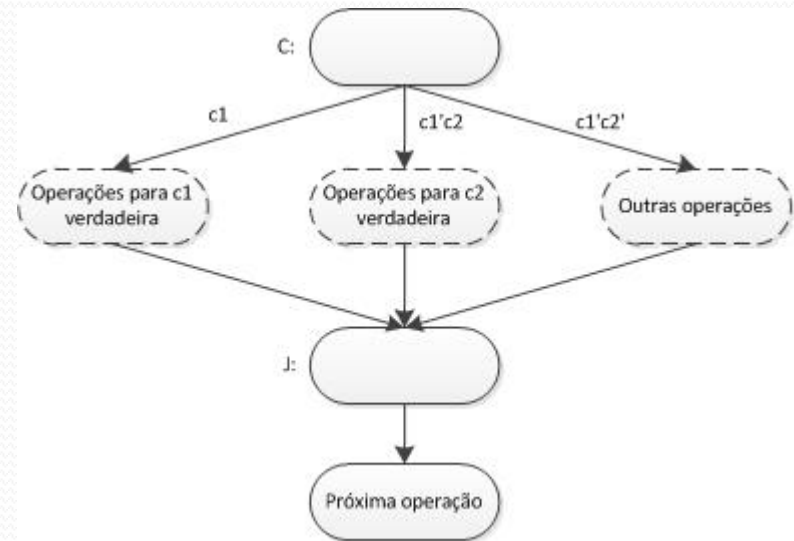


```
0: int x, y;  
1: while (1) {  
2:   while (!go_i);  
3:   x = x_i;  
4:   y = y_i;  
5:   while (x != y) {  
6:     if (x < y)  
7:       y = y - x;  
8:       else  
9:         x = x - y;  
10:  }  
11:  }  
12:  d_o = x;  
13: }
```

## Laço de repetição

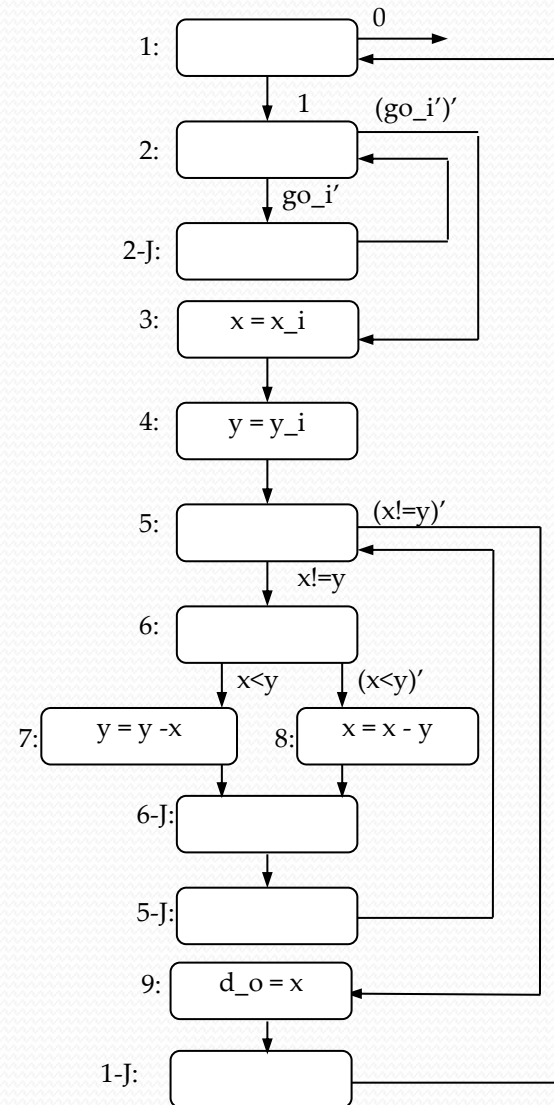


## Desvio condicional



# Projeto de Processadores Dedicados

```
0: int x, y;  
1: while (1) {  
2:   while (!go_i);  
3:   x = x_i;  
4:   y = y_i;  
5:   while (x != y) {  
6:     if (x < y)  
7:       y = y - x;  
8:     else  
9:       x = x - y;  
   }  
9:   d_o = x;  
}
```



# Projeto de Processadores Dedicados

3

Dividimos a funcionalidade em uma parte referente ao datapath e outra referente ao controlador.

- **Datapath:** interconexão de componentes combinacionais (que realizam as operações do processador) e sequenciais (registradores para armazenamento temporário de dados).
- **Controlador:** é descrito por uma máquina de estados finitos (FSM) pura, i.e., que contém apenas ações e condições booleanas.

# Construção do Datapath

- O datapath pode ser construído da seguinte maneira:
  1. Aloca-se um registrador para qualquer variável declarada.



# Construção do Datapath

Registrador: x

Registrador: y

```
0: int x, y;
1: while (1) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
10:  }
11:  d_o = x;
12: }
```

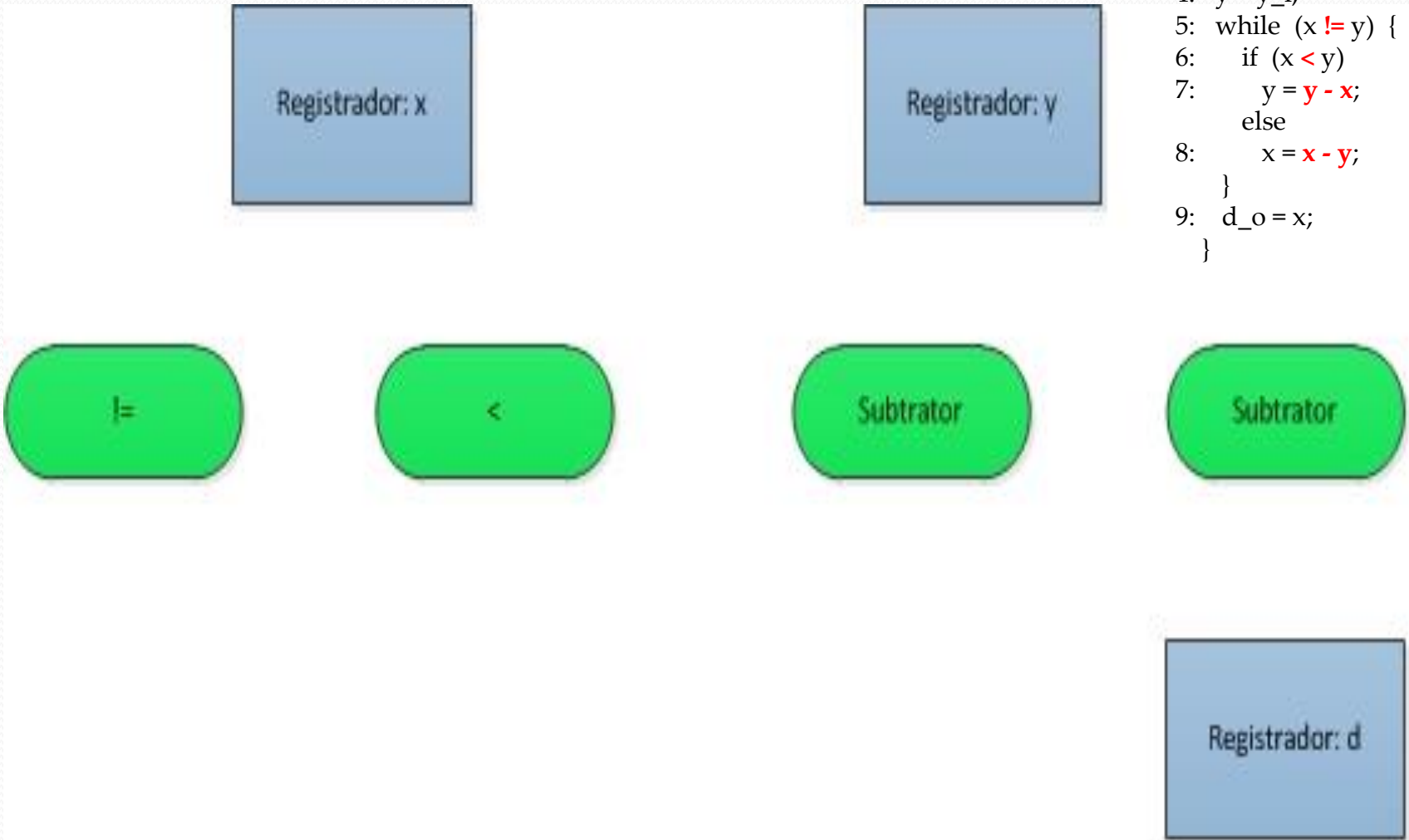
Registrador: d

# Construção do Datapath

- O datapath pode ser construído da seguinte maneira:
  2. Aloca-se uma unidade funcional para cada operação lógico-aritmética na FSMD.

# Construção do Datapath

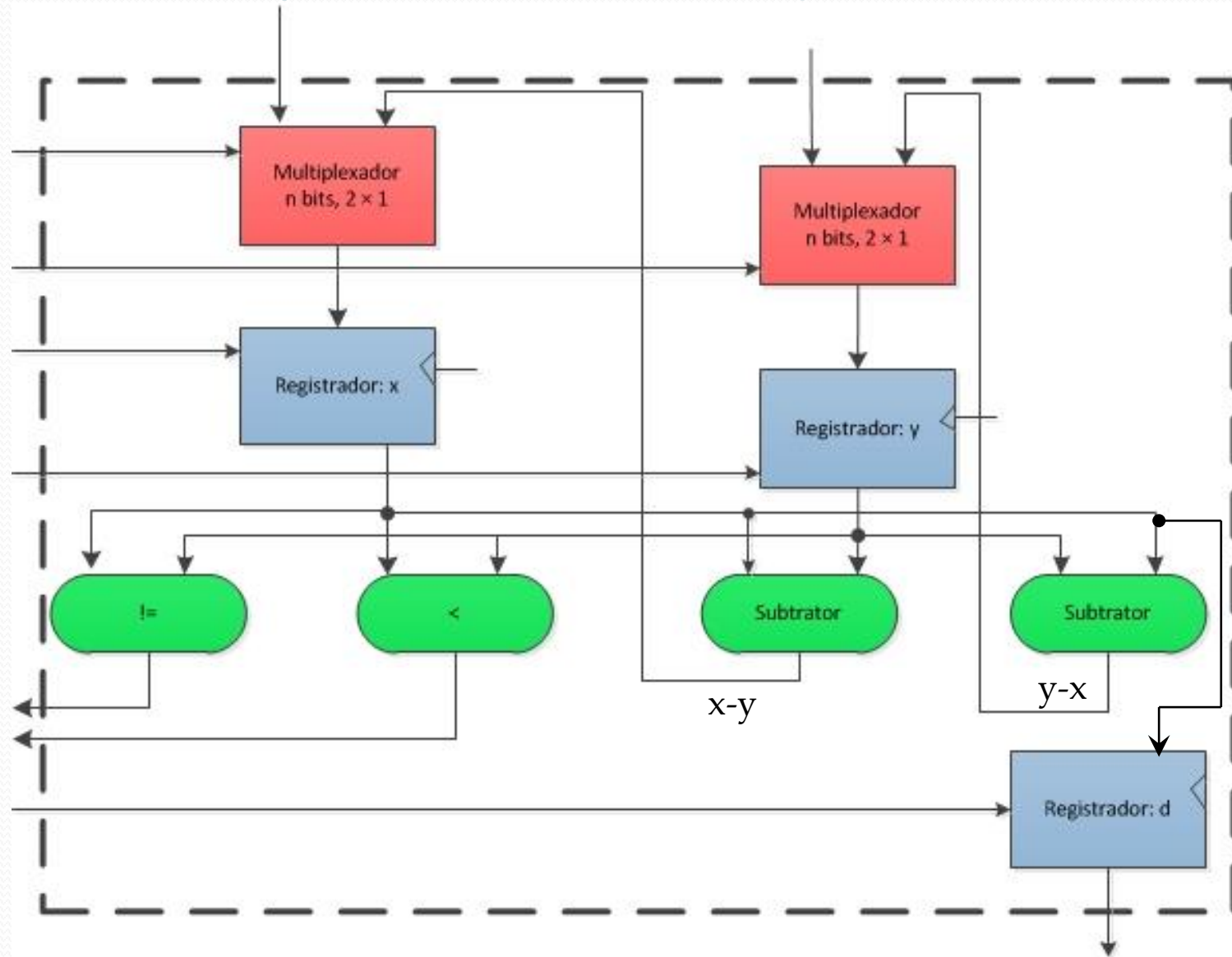
```
0: int x, y;  
1: while (1) {  
2:   while (!go_i);  
3:   x = x_i;  
4:   y = y_i;  
5:   while (x != y) {  
6:     if (x < y)  
7:       y = y - x;  
8:     else  
9:       x = x - y;  
   }  
9:   d_o = x;  
}
```



# Construção do Datapath

- O datapath pode ser construído da seguinte maneira:
  3. Conectam-se as portas lógicas, os registradores e as unidades funcionais.
    - Uso de multiplexadores para operação com diversas entradas

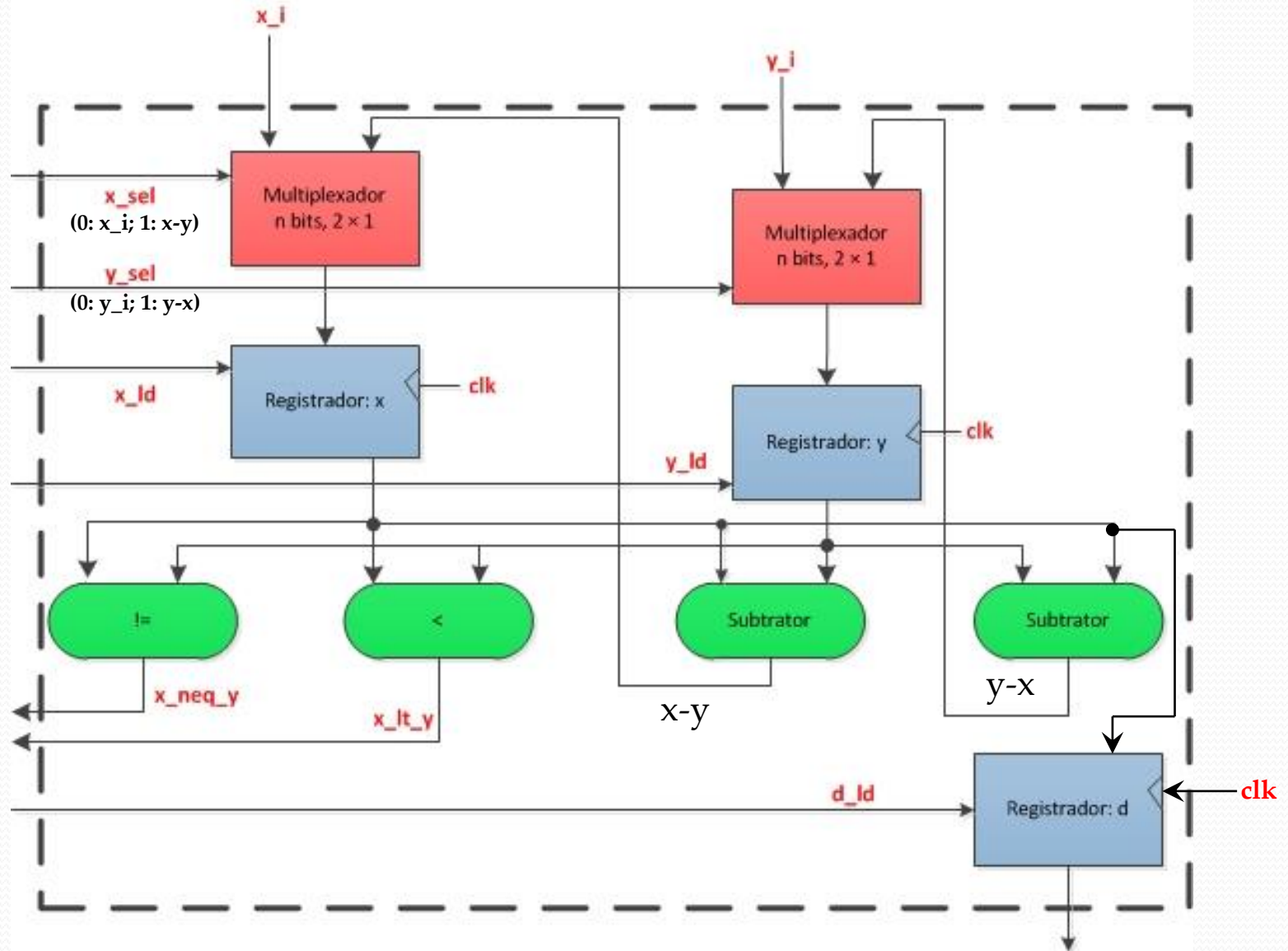
# Construção do Datapath



# Construção do Datapath

- O datapath pode ser construído da seguinte maneira:
  4. Cria-se um identificador único para cada sinal de controle dos componentes do datapath.

# Construção do Datapath



# Controlador

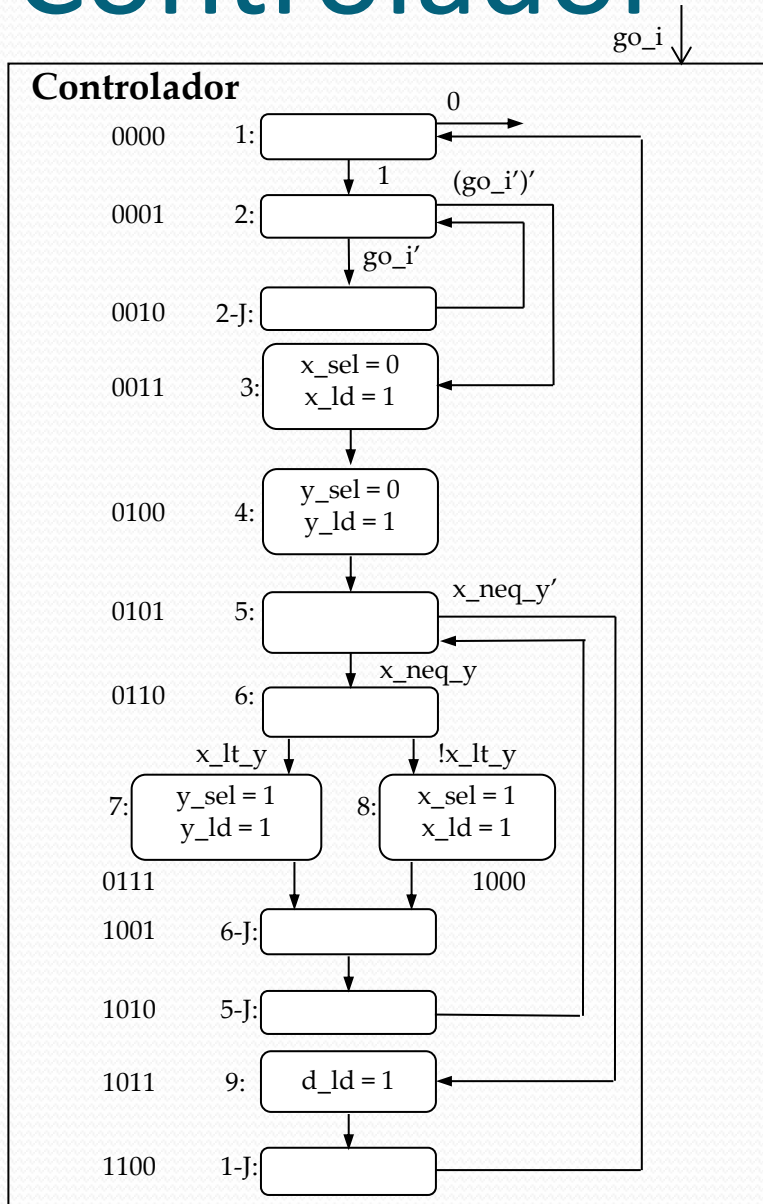
- De posse do datapath, podemos modificar a FSMD de modo a obter uma FSM que descreve o comportamento do controlador.
- A FSM possui os mesmos estados e transições da FSMD. Entretanto, as operações existentes são convertidas em versões booleanas, fazendo uso dos sinais de controle do datapath.



# Controlador

- Operações de escrita em variável:
  - Ajustar os sinais de seleção de multiplexadores e sinais de carregamento (*load*) em registradores.
- Operações lógicas em uma condição:
  - Usar os sinais de controle de saída da unidade funcional correspondente.

# Controlador



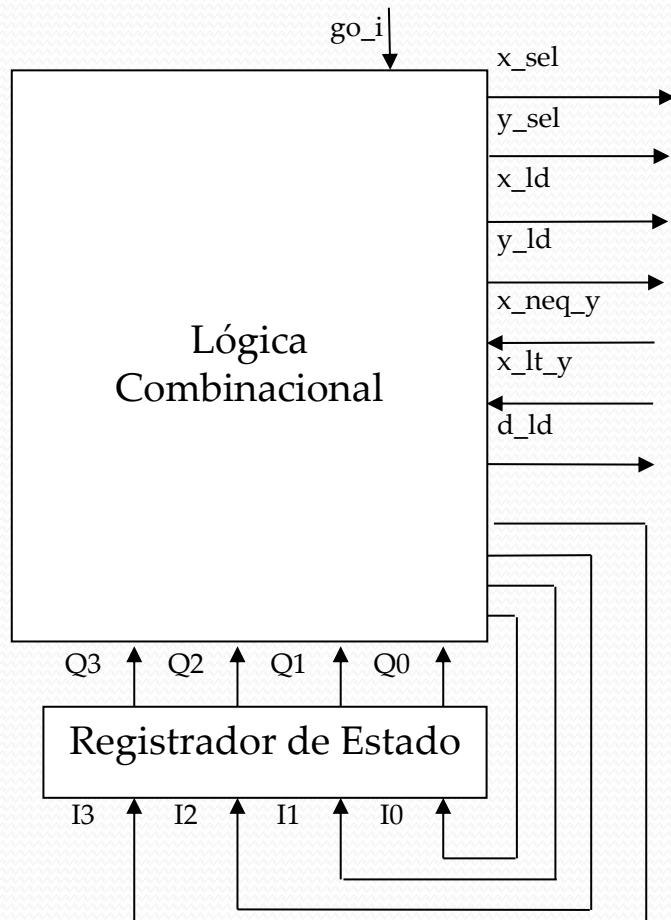
- **Cuidados:**

- As operações especificadas em um mesmo estado ocorrem em paralelo.

- Operações de escrita em uma variável não atualizam seu valor imediatamente – é preciso esperar pelo próximo ciclo de relógio.

# Controlador

- Modelo de implementação do controlador



- É possível completar o projeto do controlador implementando a FSM através da técnica de projeto de lógica sequencial descrita anteriormente.
- Nesta etapa, ferramentas computacionais de auxílio ao projeto (CAD) que automatizem o processo de desenvolvimento das lógicas combinacional e sequencial podem ser muito úteis. (Por exemplo, ferramentas de síntese).

# Controlador

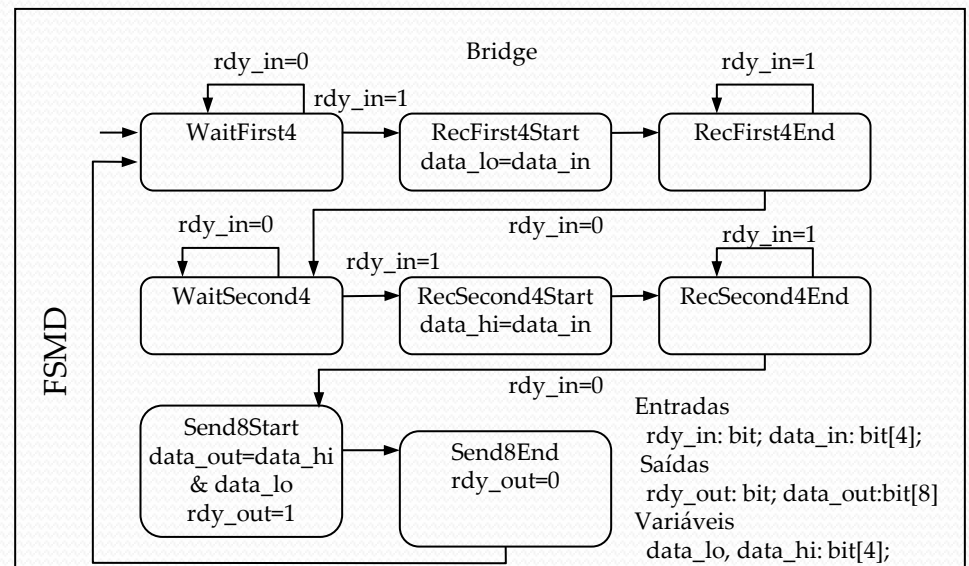
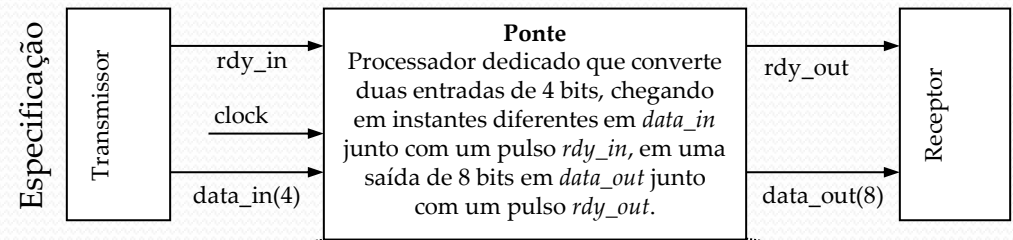
Inputs							Outputs								
Q3	Q2	Q1	Q0	x_neq_y	x_lt_y	go_i	I3	I2	I1	I0	x_sel	y_sel	x_ld	y_ld	d_ld
0	0	0	0	*	*	*	0	0	0	1	X	X	0	0	0
0	0	0	1	*	*	0	0	0	1	0	X	X	0	0	0
0	0	0	1	*	*	1	0	0	1	1	X	X	0	0	0
0	0	1	0	*	*	*	0	0	0	1	X	X	0	0	0
0	0	1	1	*	*	*	0	1	0	0	0	X	1	0	0
0	1	0	0	*	*	*	0	1	0	1	X	0	0	1	0
0	1	0	1	0	*	*	1	0	1	1	X	X	0	0	0
0	1	0	1	1	*	*	0	1	1	0	X	X	0	0	0
0	1	1	0	*	0	*	1	0	0	0	X	X	0	0	0
0	1	1	0	*	1	*	0	1	1	1	X	X	0	0	0
0	1	1	1	*	*	*	1	0	0	1	X	1	0	1	0
1	0	0	0	*	*	*	1	0	0	1	1	X	1	0	0
1	0	0	1	*	*	*	1	0	1	0	X	X	0	0	0
1	0	1	0	*	*	*	0	1	0	1	X	X	0	0	0
1	0	1	1	*	*	*	1	1	0	0	X	X	0	0	1
1	1	0	0	*	*	*	0	0	0	0	X	X	0	0	0
1	1	0	1	*	*	*	0	0	0	0	X	X	0	0	0
1	1	1	0	*	*	*	0	0	0	0	X	X	0	0	0
1	1	1	1	*	*	*	0	0	0	0	X	X	0	0	0

# Projeto de Processadores Dedicados – Nível de Registradores

- Em alguns casos, é preferível iniciar o projeto diretamente com a elaboração de uma FSMD:
  - Quando a temporização ou o comportamento ciclo a ciclo do sistema é crucial (e.g., circuitos de espera e/ou sincronização).

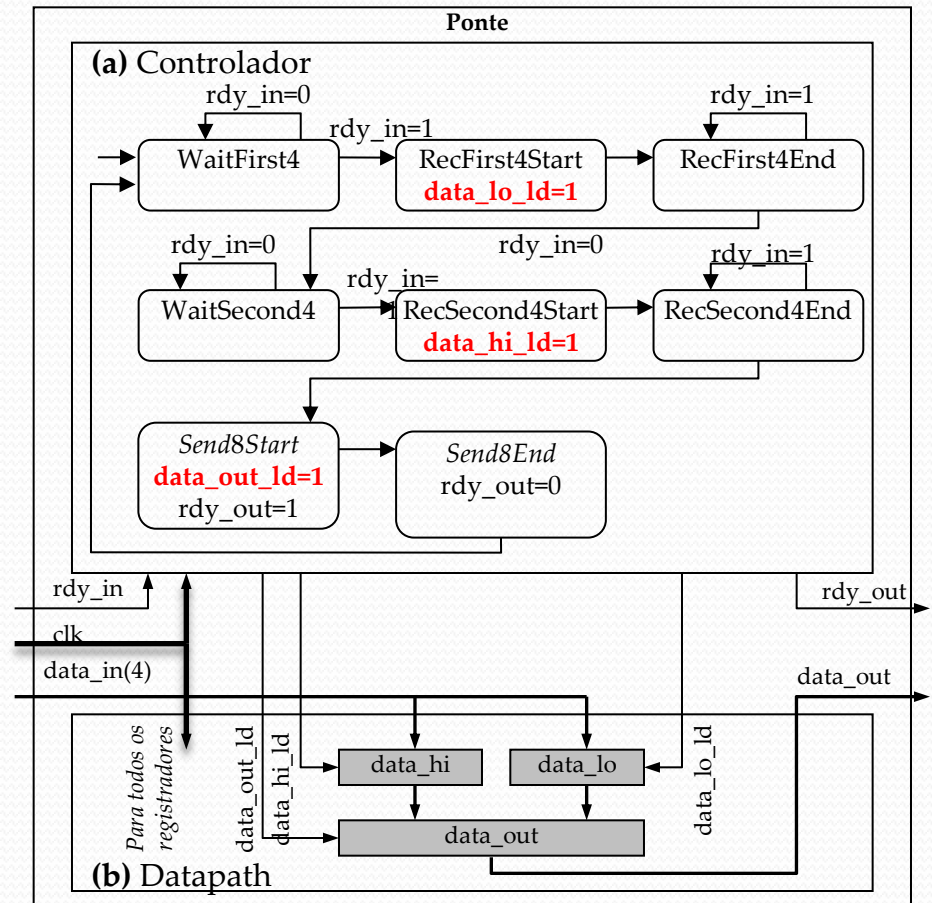
# Projeto de Processadores Dedicados – Nível de Registradores

- Exemplo: Ponte entre um barramento de 4 bits e um barramento de 8 bits.



# Projeto de Processadores Dedicados – Nível de Registradores

- Exemplo: Ponte entre um barramento de 4 bits e um barramento de 8 bits.



# Otimização de Processadores Dedicados

- A estratégia de projeto de processadores dedicados vista anteriormente oferece uma maneira sistemática para construir o datapath e para obter a FSM que descreve o controlador.
- No entanto, muitas oportunidades para simplificar os componentes do processador foram desconsideradas.

Exemplo:

- Estados que não executam operação alguma.
- Repetição desnecessária de unidades funcionais.



# Otimização de Processadores Dedicados

- Por que isso é importante?
  - No contexto de sistemas embarcados, otimizar os componentes do processador pode ser crucial para atingir uma implementação que atenda adequadamente aos requisitos desejados quanto às métricas de projeto, como velocidade de processamento, consumo de potência e tamanho.
  - Visa alcançar os melhores valores possíveis para as métricas de projeto.

# Otimização do Programa

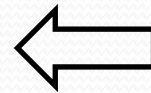
- Há várias partes associadas ao projeto de um processador dedicado em que surgem oportunidades para otimização:
  - Programa
  - FSMD
  - Datapath
  - FSM

# Otimização do Programa

- Análise do algoritmo em termos de sua complexidade computacional no tempo e no espaço.

- Exemplo:

```
0: int x, y, r;  
1: while (1) {  
2:   while (!go_i);  
3:   if (x_i >= y_i) {  
4:     x=x_i;  
5:     y=y_i;  
6:   }  
7:   else {  
8:     x=y_i;  
9:     y=x_i;  
10:  }  
11:  while (y != 0) {  
12:    r = x % y;  
13:    x = y;  
14:    y = r;  
15:  }  
16:  d_o = x;  
17: }
```

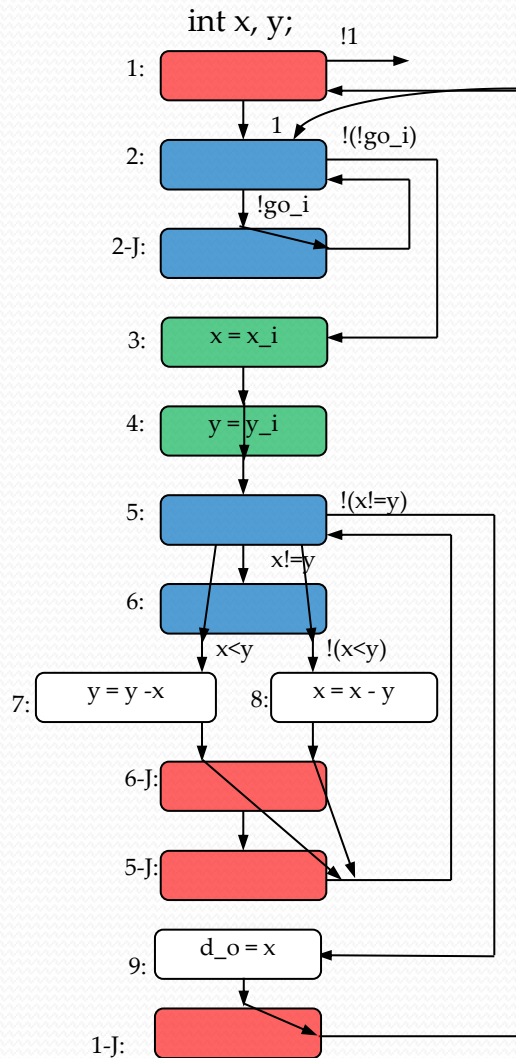


Fazendo uso de uma operação que computa de modo eficiente o resto de uma divisão, podemos reduzir o número de iterações em relação ao primeiro algoritmo.

# Otimização da FSMD

- Possíveis melhorias:
  - Fusão de estados
    - Estados com constantes especificando as transições condicionais podem ser eliminados pois a transição destes estados é sempre a mesma.
    - Estados adjacentes que realizam operações independentes podem ser reunidos em um único estado.
  - Separação de estados
    - Estados associados a operações complexas podem ser quebrados em estados menores para reduzir o tamanho do *hardware*.

# Otimização da FSMD



*Eliminação* do estado 1 – transições possuem valores constantes.

*Fusão* do estado 2 com o estado 2J – eliminação do loop entre eles pois não há operações no laço.

*Fusão* dos estados 3 e 4 – as operações são independentes entre si.

*Fusão* dos estados 5 e 6 – transições a partir do estado 6 podem ser realizadas no estado 5.

*Eliminação* dos estados 5J e 6J – as transições de cada estado podem ser realizadas a partir do estado 7 e 8, respectivamente.

*Eliminação* do estado 1-J – transições a partir do estado 1-J podem ser feitas diretamente a partir do estado 9.

**Atenção:** essas modificações alteram a temporização da FSMD!

# Otimização do Datapath

- Economia de unidades funcionais:
  - Podemos compartilhar o uso de unidades funcionais no *datapath*, em vez de destinar uma para cada operação lógico-aritmética realizada (desde que tais operações ocorram em estados distintos).
- Além disso, existem diferentes componentes no nível de transferência entre registradores que podem compor um *datapath*: as peculiaridades de cada dispositivo podem ser benéficas ou não dependendo do contexto e das métricas de projeto para aquele sistema embarcado.

# Otimização da FSM

- Entram em cena questões envolvendo:
  - A **codificação dos estados**, que exerce influência na complexidade do circuito combinacional que gera o próximo estado.
  - A **minimização do número de estados**, através da fusão de estados equivalentes.

# Resumo

- Processadores dedicados:
  - Possuem hardware especialmente selecionado para realizar a tarefa desejada.
  - Técnica sistemática de projeto.
  - Podem ser desenvolvidos a partir de um algoritmo (programa) ou de uma FSMD.
  - Otimização em todos os níveis pode levar a implementações que satisfaçam os requisitos de projeto.