**Fundamentos de Sistemas Digitais**

# Lógica Sequencial

**Prof. Dr. Alexandre M. Amory**
**Prof. Dr Edson I. Moreno**

# Referências

- Sugiro estudarem nesta ordem de preferência:
- Floyd, Cap 7 até 7.4, 9, 10.
  - Não tem nada de VHDL
- D'AMORE, Roberto. VHDL: Descrição e Síntese de Circuitos Digitais. Rio de Janeiro: LTC, 2005. 259 p.
  - Cap 6 – Descrição de Circuitos Síncronos
- Digital Design and Computer Architecture
  - Sec 2.9, Cap 3, 4, 5
- Free Range VHDL
- Vahid, Cap 3 – Parte referente à flip-flop e registradores
- PEDRONI, Volnei A. Eletrônica Digital Moderna e VHDL. Elsevier Ltda. Editora, Rio de Janeiro, RJ: 2010. 619 p.
  - Teoria: Cap 11 e 12
  - VHDL: Cap 20 e 21

# ATENÇÃO

- Não se aprende alguma linguagem :
  - Escutando alguém explicar
  - Somente lendo livros

- Aprende praticando
  - Exercício EXTRA-CLASSE é FUNDAMENTAL !
  - Façam exercícios sugeridos e procurem outros nos livros indicados

# Foco deste PPT

- Ênfase na representação RTL e FSM em VHDL

Níveis de abstração típicos:

1o : máscara de layout (baixa abstração)

2o: diagrama de transistores

**3o: portas lógicas (sequenciais)**

**4o: blocos / funções lógicos**

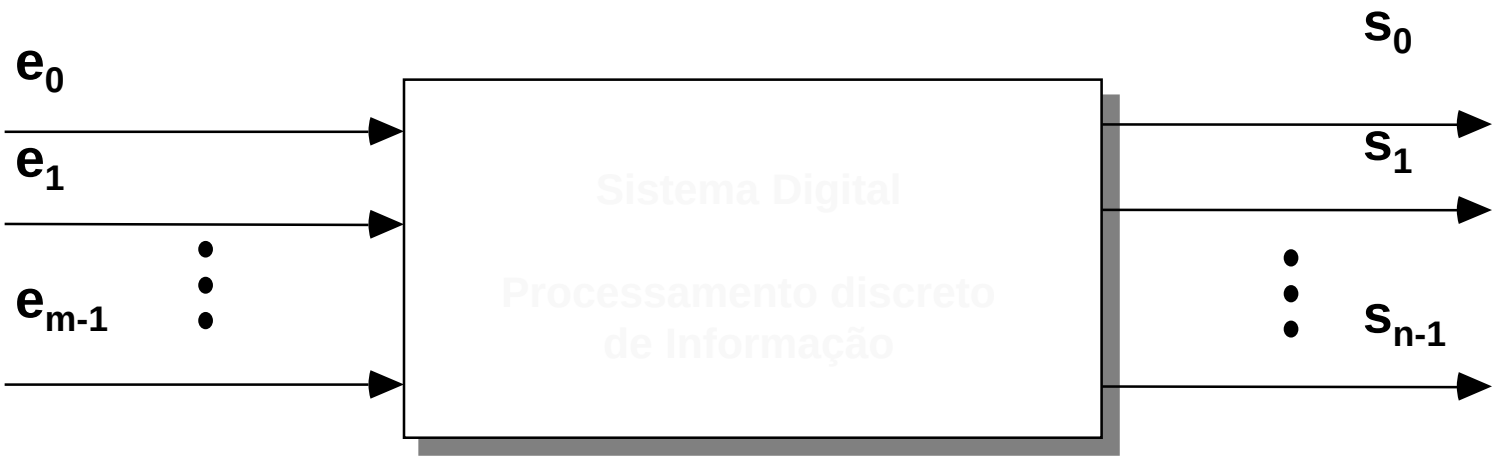
**5o: nível de registradores**

# Sistemas Digitais

## Definição funcional:

**Aparato** dotado de conjuntos finitos de **entradas** e **saídas** e capaz de processar informação representada sob forma **discreta**

## Representação estrutural:




## Subdivisão:

- **Circuitos combinacionais**
- **Circuitos seqüenciais**

# Sistemas Digitais Seqüenciais

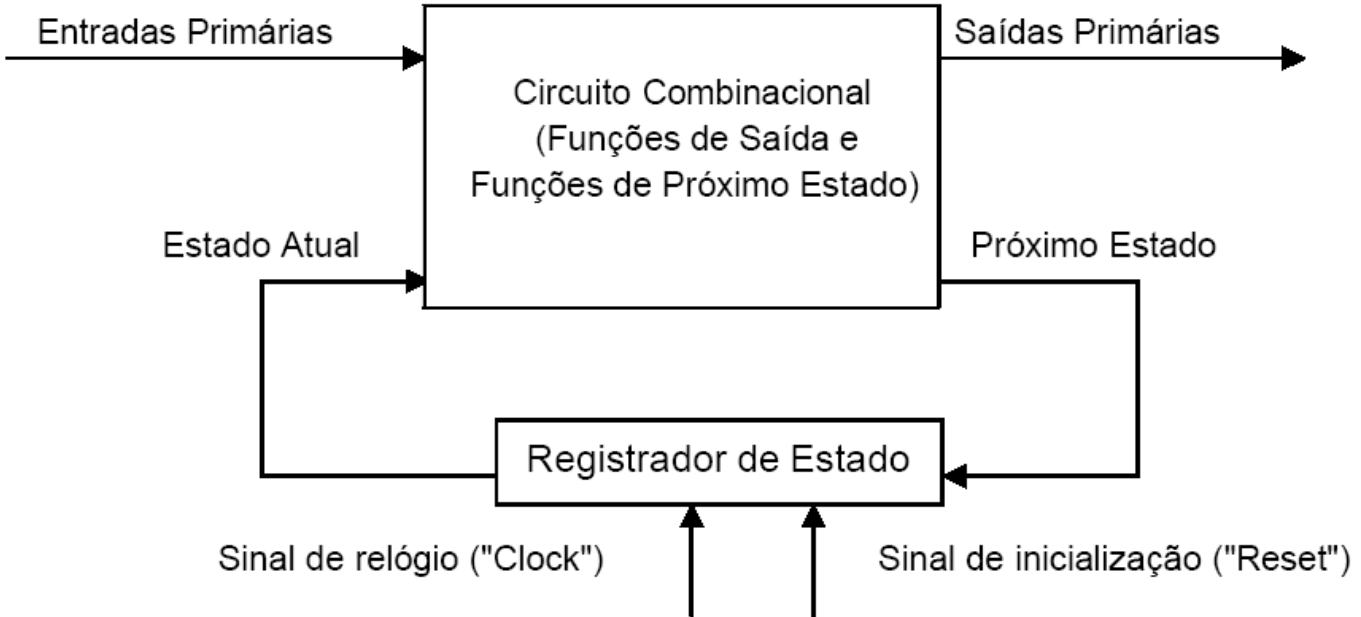
**Definição:**

Circuito, cujo comportamento de cada saída é descrito como função dos valores instantâneos das entradas e de seus valores passados

Intuitivamente esta definição leva ao conceito de memória, responsável por armazenar os valores passados

Memórias requerem sinais de controle para determinar os instantes de carga e os valores de inicialização

**Representação:**

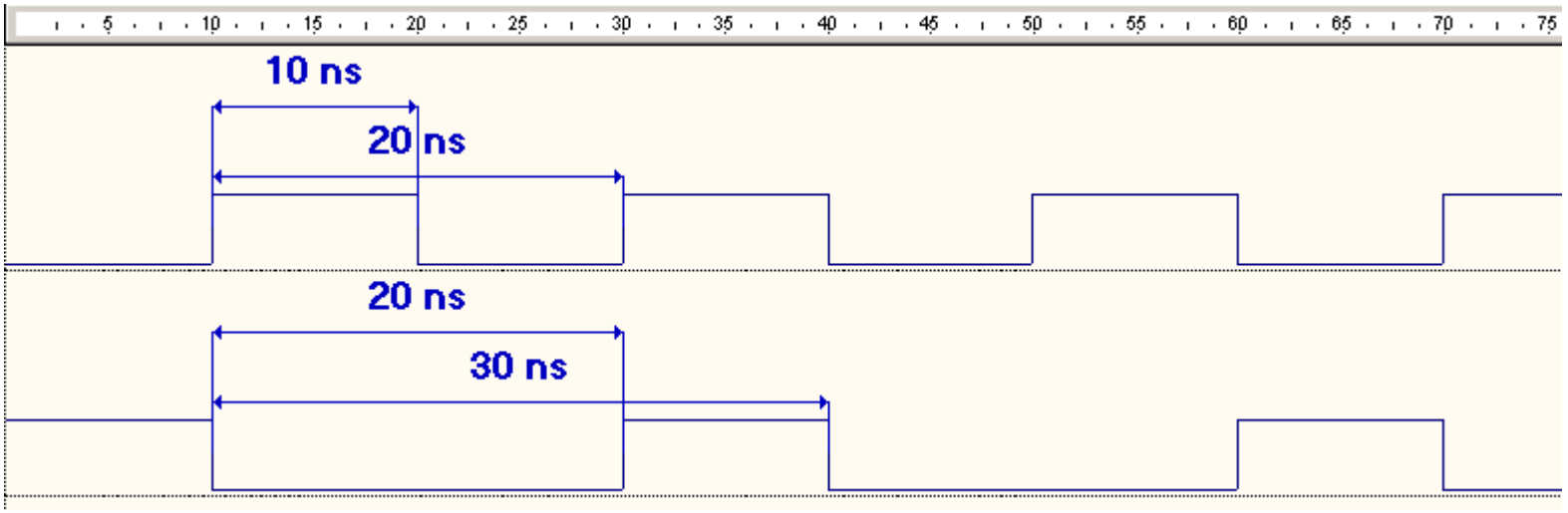


# Sistemas Digitais Seqüenciais

- **Circuito seqüencial pode ser representado por um circuito combinacional associado à uma memória**
- **A parte combinacional tem como entradas, as entradas externas (entradas atuais) e as entradas internas, provenientes da memória (entradas passadas), que fornecem o *estado* do circuito**
- **A parte combinacional tem como saídas, as saídas externas (saídas do circuito seqüencial) e as saídas internas (fornecem o cálculo do próximo estado) para o circuito**
- **A memória tem como entradas o valor do próximo estado que será armazenado e sinais de controle**
  - Os principais sinais de controle são o relógio (do inglês, clock) e sinais de inicialização, tais como set, reset, clear, e outros. Estes últimos usados para iniciar a operação do circuito seqüencial em um estado conhecido
- **Circuitos seqüenciais são normalmente divididos em síncronos e assíncronos**
  - Síncronos são aqueles cujas transições são dependentes do evento de um sinal único de sincronismo – normalmente chamado de relógio
  - Assíncronos são aqueles cujas transições não são coordenadas por um único sinal

# Relógio

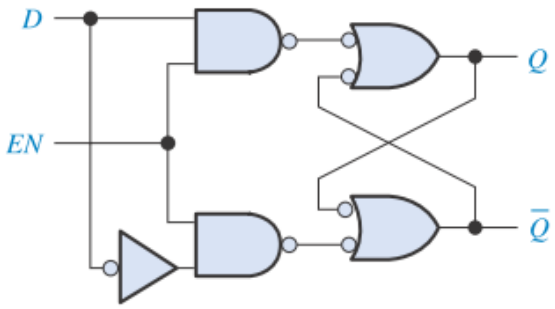
- **Relógio é uma onda periódica com freqüência, fase e amplitude. O objetivo deste sinal é determinar os instantes de tempo em que o circuito seqüencial deve avaliar as suas entradas**
- **Exemplo:**
  - Abaixo seguem dois relógios CK1 e CK2 com períodos 20ns e 30ns, respectivamente. Conseqüentemente com freqüências 50MHz e 33,33MHz



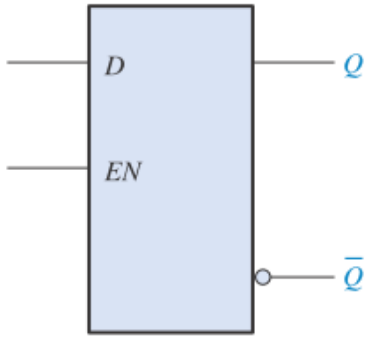

- **Exercícios:**
  1. Desenhar um relógio com as seguintes características: freqüência de 1GHz, com 25% do período em 1 e 75% em 0
  2. Fazer o VHDL que corresponde ao relógio desenhado

# LATCHES & FLIP-FLOPS

# Latch D c/ enable



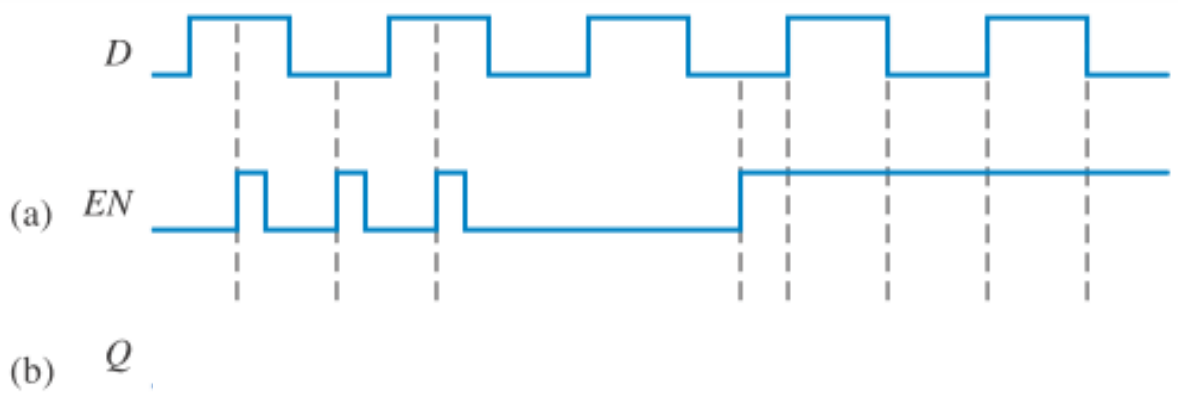

(a) Diagrama lógico




(b) Símbolo lógico

| Entradas | | Saídas | | |
|---|---|---|---|---|
| D | EN | Q | $\overline{Q}$ | Comentários |
| 0 | 1 | 0 | 1 | RESET |
| 1 | 1 | 1 | 0 | SET |
| X | 0 | $Q_0$ | $\overline{Q}_0$ | Repouso |

*Nota:* $Q_0$ é o nível da saída anterior antes que as condições de entrada fossem estabelecidas.



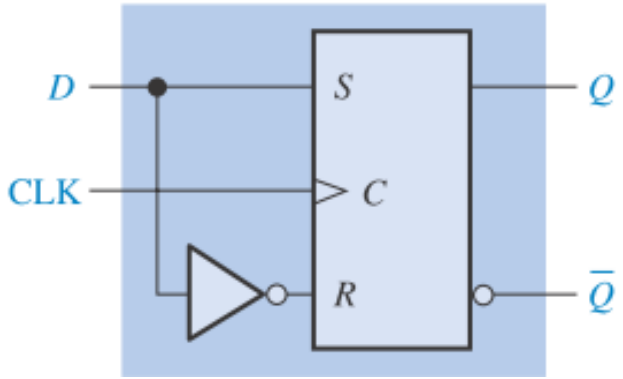

* Floyd. Cap 7

# Latch D em VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity latch is
  port(clk: in  STD_LOGIC;
       d:   in  STD_LOGIC_VECTOR(3 downto 0);
       q:   out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of latch is
begin
  process(clk, d) begin
    if clk = '1' then q <= d;
    end if;
  end process;
end;
```

* Digital Design and Computer Architecture. Cap 4

# Flip-Flop D Sensível à Borda




**▶ FIGURA 7–20**

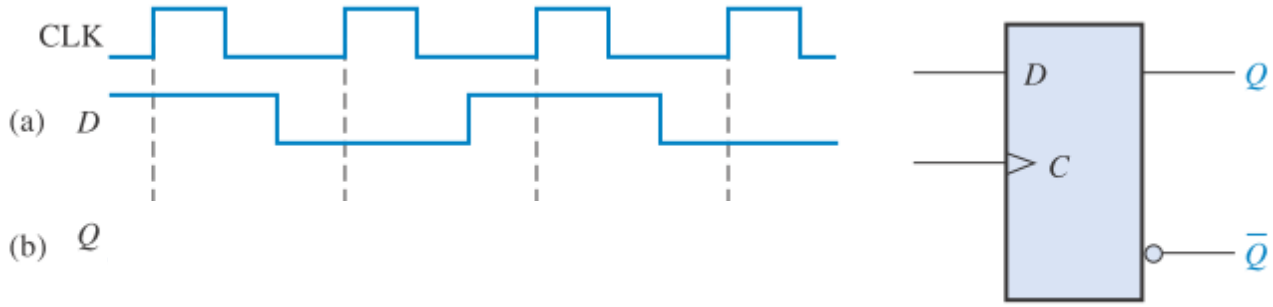
Um flip-flop D disparado por borda positiva construído a partir de um flip-flop S-R e um inversor.

| ENTRADAS | | SAÍDAS | | COMENTÁRIOS |
|---|---|---|---|---|
| D | CLK | Q | $\overline{Q}$ | |
| 1 | ↑ | 1 | 0 | SET (armazena um nível 1) |
| 0 | ↑ | 0 | 1 | RESET (armazena um nível 0) |

↑ = Transição do clock do nível BAIXO para o ALTO.

**◀ TABELA 7–3**

Tabela-verdade para um flip-flop D disparado por borda positiva




**▲ FIGURA 7–21**

# Timing – Latch vs FF D

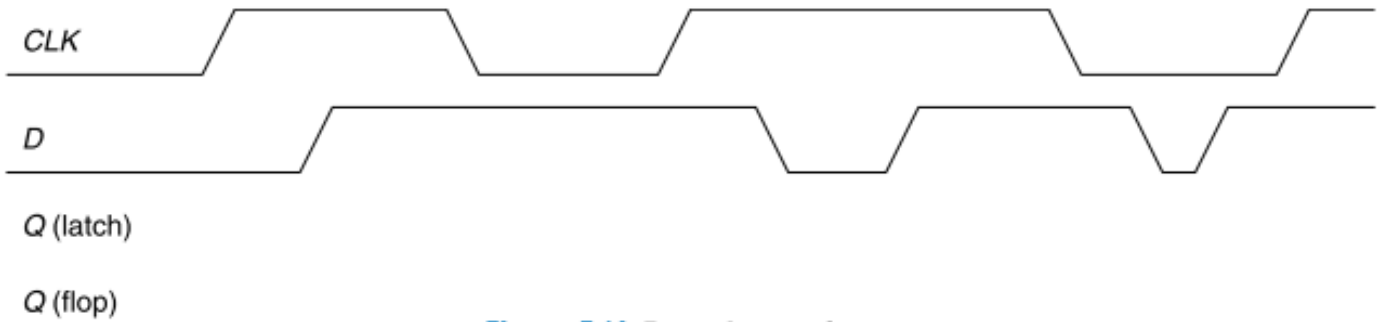



**Figure 3.14** Example waveforms

* Digital Design and Computer Architecture. Sec 2.9

# Timing – Latch vs FF D

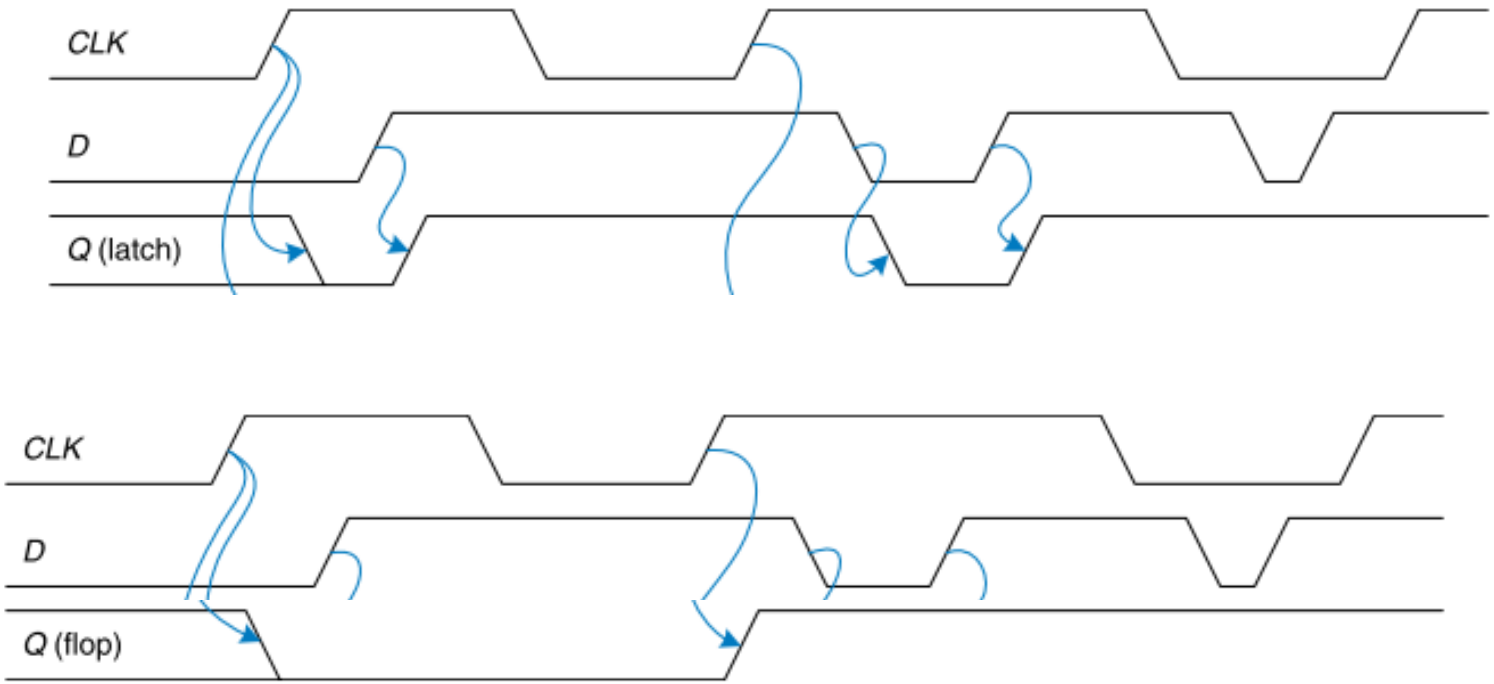



**Figure 3.15** Solution waveforms

* Digital Design and Computer Architecture. Sec 2.9

# FF D em VHDL

**Listing 6.2: Solution to Example 14.**

```
-------------------------------------
-- Model of a simple D Flip-Flop --
-------------------------------------
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity d_ff is
    port ( D, CLK : in  std_logic;
           Q :      out std_logic);
end d_ff;
-- architecture
architecture my_d_ff of d_ff is
begin
    dff:  process(CLK)
    begin
        if (rising_edge(CLK))       then
--or     if (CLK'event and CLK='1') then
            Q <= D;
        end if;
    end process dff;
end my_d_ff;
```

* mais exemplos no capítulo 12 do Free Range VHDL

# FF D com Reset

**Listing 6.4: Solution to Example 16.**

```
-------------------------------------
-- FET D Flip-flop model with active-
-------------------------------------
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity d_ff_r is
    port (   D,R :  in  std_logic;
             CLK :  in  std_logic;
               Q :  out std_logic);
end d_ff_r;
-- architecture
architecture my_d_ff_r of d_ff_r is
begin
   dff:  process (R,CLK)
   begin
      if (R = '1') then
         Q <= '0';
      elsif (falling_edge(CLK)) then
         Q <= D;
      end if;
   end process dff;
end my_d_ff_r;
```

# FF D com Load

## 12.3 8-Bit Register with Load Enable - Behavioral Model

```
-----------------------------------------------
--  Register: 8-bit Register with load enable.
--
--  Required signals:
-----------------------------------------------
--  CLK,LD: in  std_logic;
--  D_IN:   in  std_logic_vector(7 downto 0);
--  D_OUT:  out std_logic_vector(7 downto 0);
-----------------------------------------------
process (CLK)
begin
   if (rising_edge(CLK)) then
      if (LD = '1') then   -- positive logic for LD
         D_OUT <= D_IN;
      end if;
   end if;
end process;
--
```

# FF D com set/reset

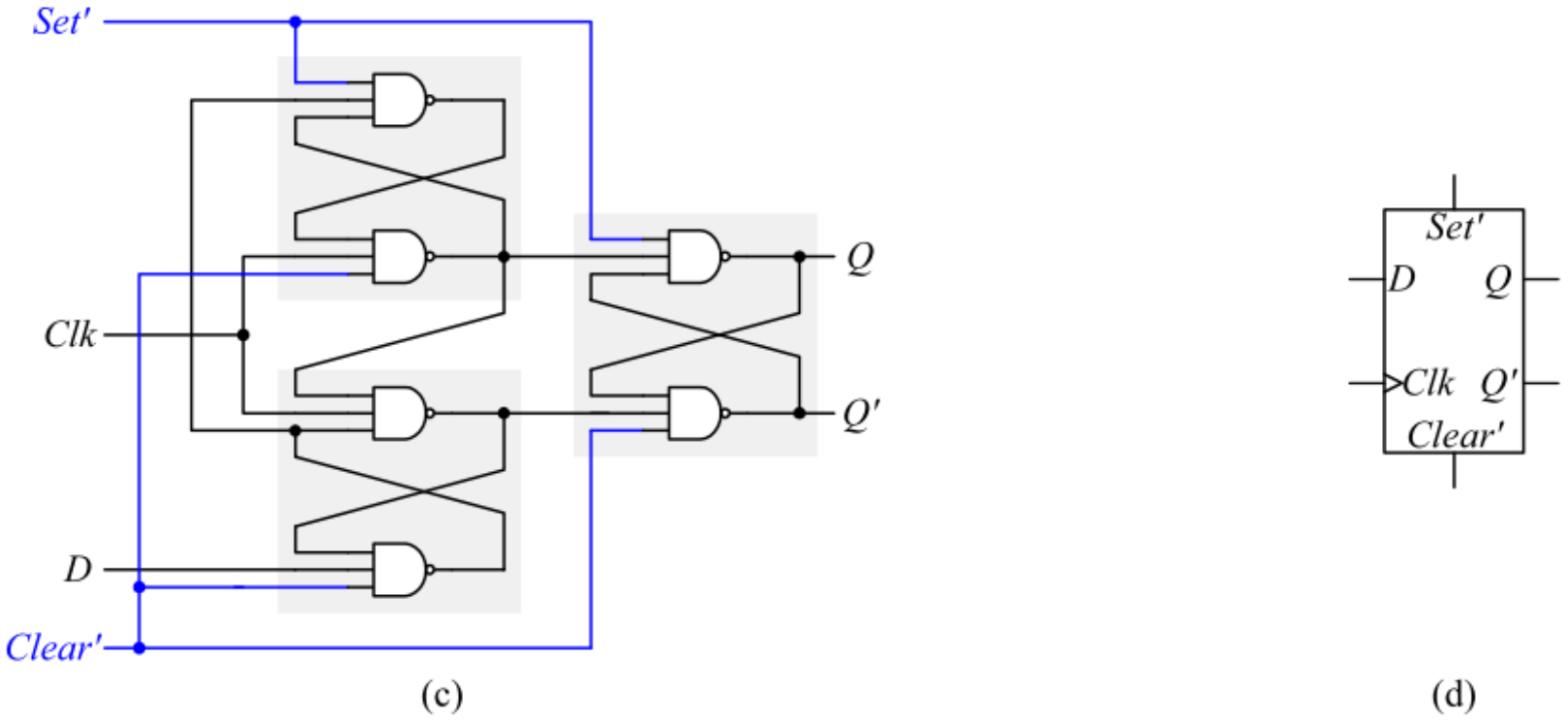
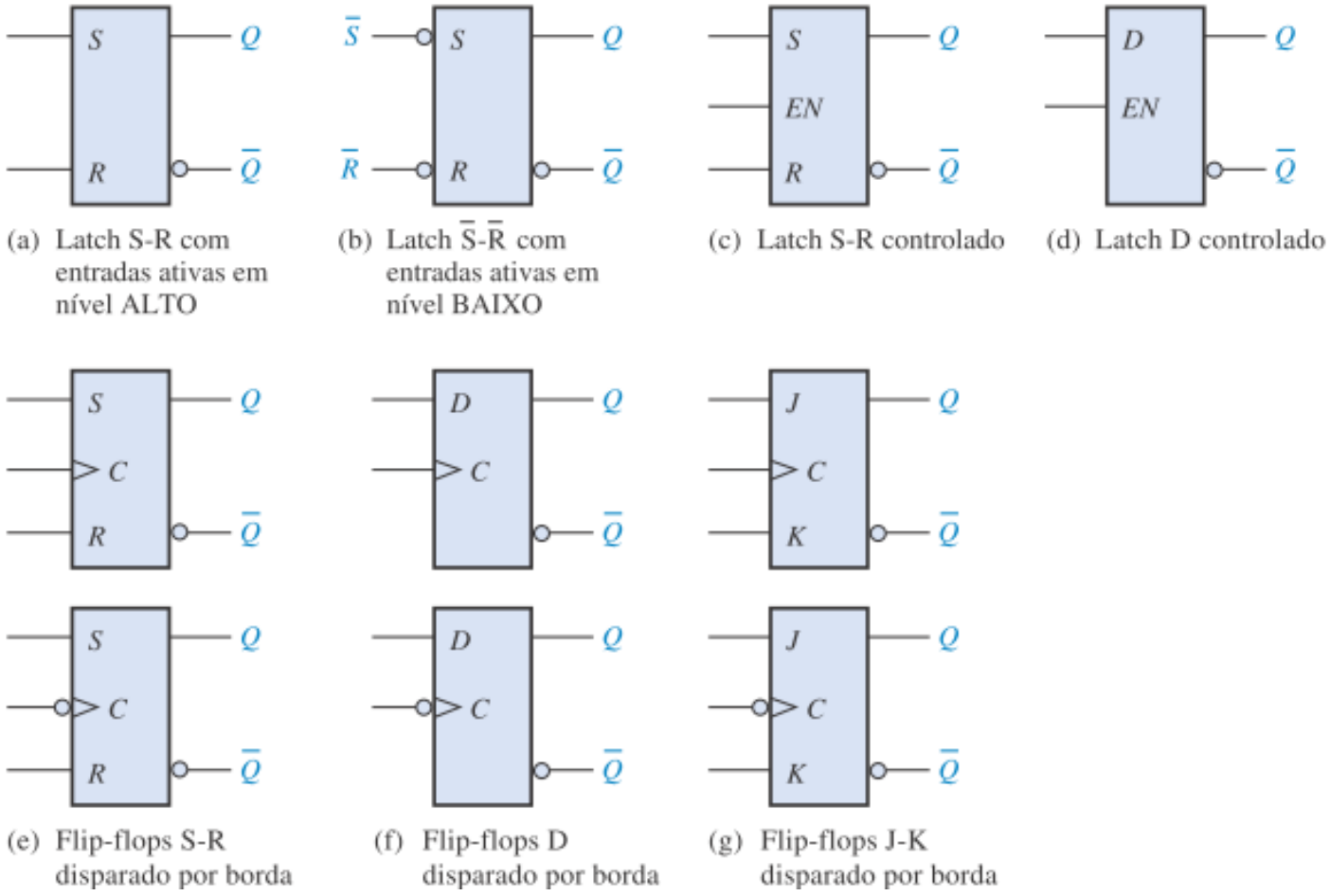



**Figure 6.14** Storage elements with asynchronous inputs: (a) D latch with active-low set and clear; (b) logic symbol for (a); (c) D edge-triggered flip-flop with active-low set and clear; (d) logic symbol for (c).

# Resumo de Latches e FFs




(a) Latch S-R com entradas ativas em nível ALTO

(b) Latch $\overline{S}$-$\overline{R}$ com entradas ativas em nível BAIXO

(c) Latch S-R controlado

(d) Latch D controlado

(e) Flip-flops S-R disparado por borda

(f) Flip-flops D disparado por borda

(g) Flip-flops J-K disparado por borda

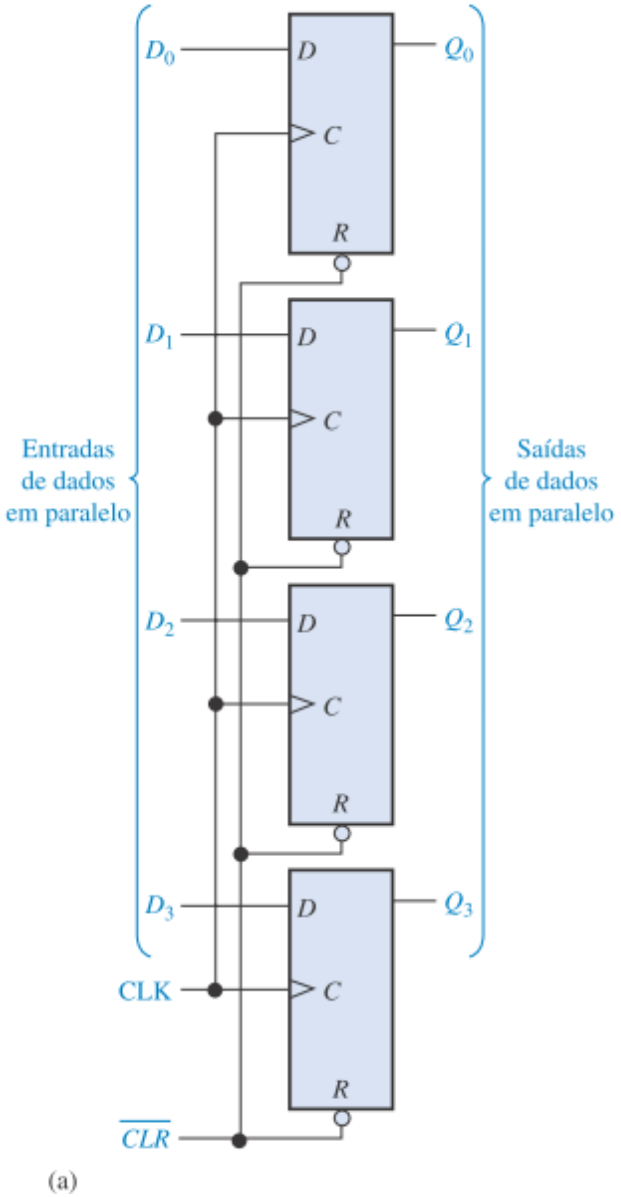
► FIGURA 7–67

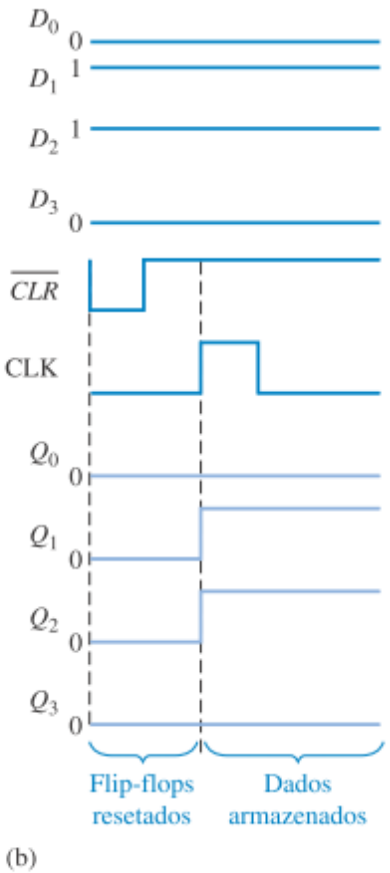
*Floyd

# Atividade da Próxima Aula

- Atividade p Próxima Aula
  - Entregar no ínicio da aula
  - Descrever em VHDL
    - **Flip-flop D com set, reset, load**

# Registradores



# Registrador em VHDL

**Listing 9.1: Solution to Example 23.**

```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity reg8 is
    Port ( REG_IN  :  in std_logic_vector(7 downto 0);
           LD,CLK  :  in std_logic;
           REG_OUT : out std_logic_vector(7 downto 0));
end reg8;
-- architecture
architecture reg8 of reg8 is
begin
   reg: process(CLK)
   begin
      if (rising_edge(CLK)) then
         if (LD = '1') then
            REG_OUT <= REG_IN;
         end if;
      end if;
   end process;
end reg8;
```

# Como determinar o período do relógio

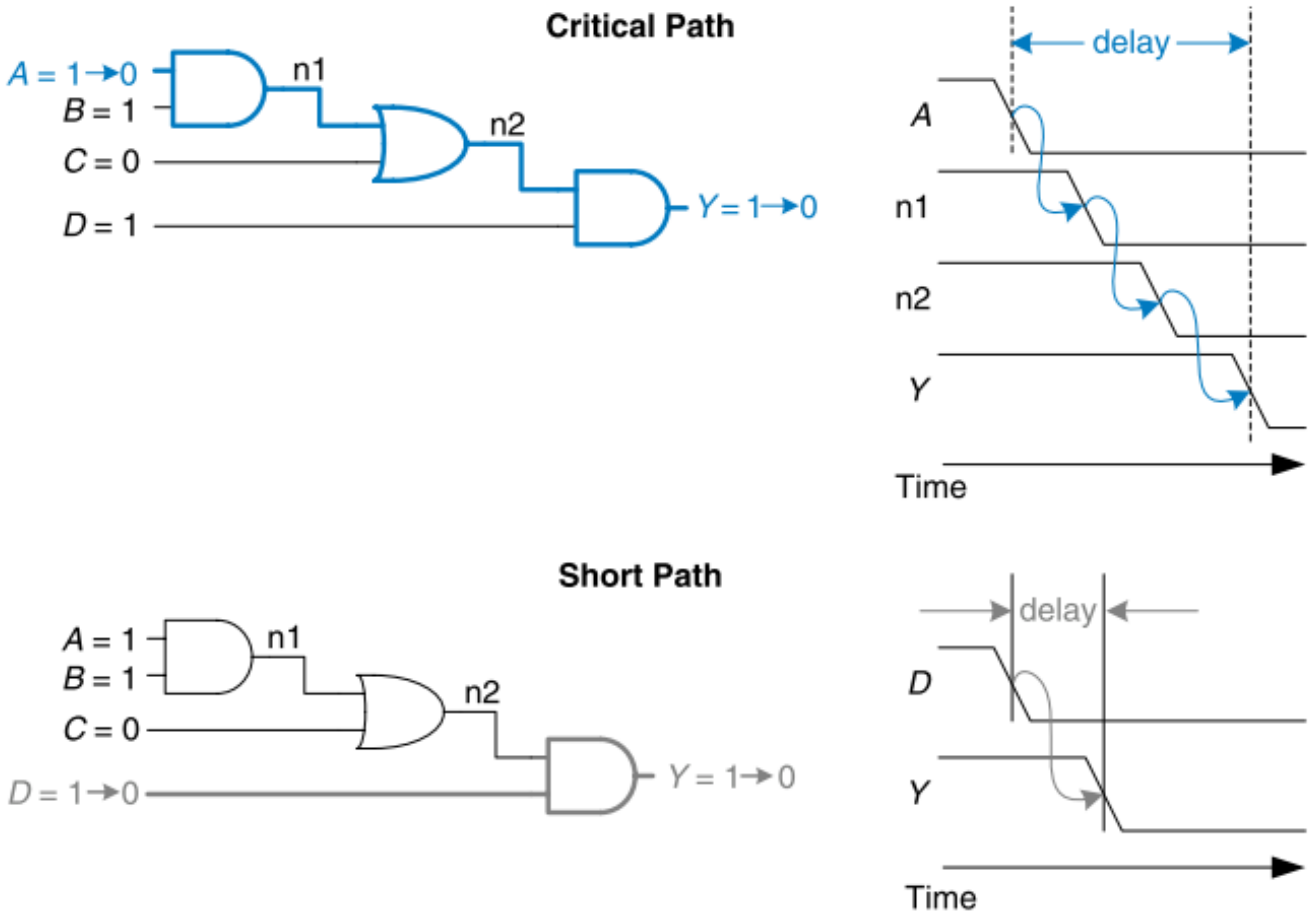



**Figure 2.69** Critical and short path waveforms

* Digital Design and Computer Architecture. Sec 2.9

**Exercise 2.32** Determine the propagation delay and contamination delay of the circuit in Figure 2.83. Use the gate delays given in Table 2.8.

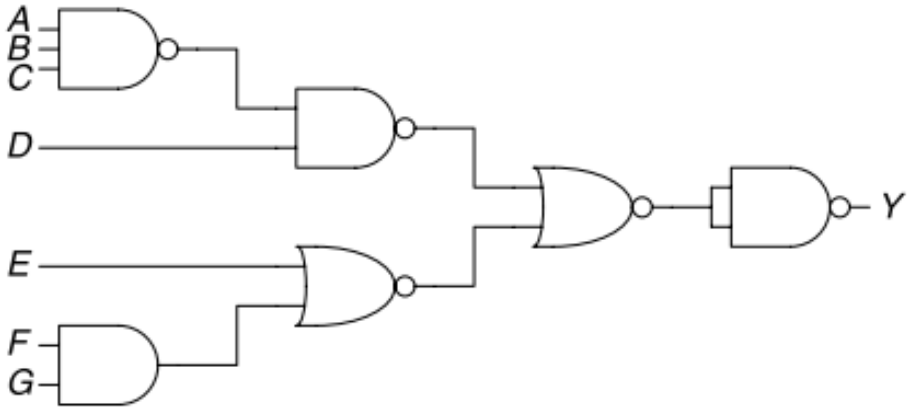



**Figure 2.83** Circuit schematic

**Table 2.8** Gate delays for Exercises 2.32–2.35

| Gate | $t_{pd}$ (ps) |
|---|---|
| NOT | 15 |
| 2-input NAND | 20 |
| 3-input NAND | 30 |
| 2-input NOR | 30 |
| 3-input NOR | 45 |
| 2-input AND | 30 |
| 3-input AND | 40 |
| 2-input OR | 40 |
| 3-input OR | 55 |
| 2-input XOR | 60 |

# Timing de Circuito Síncrono

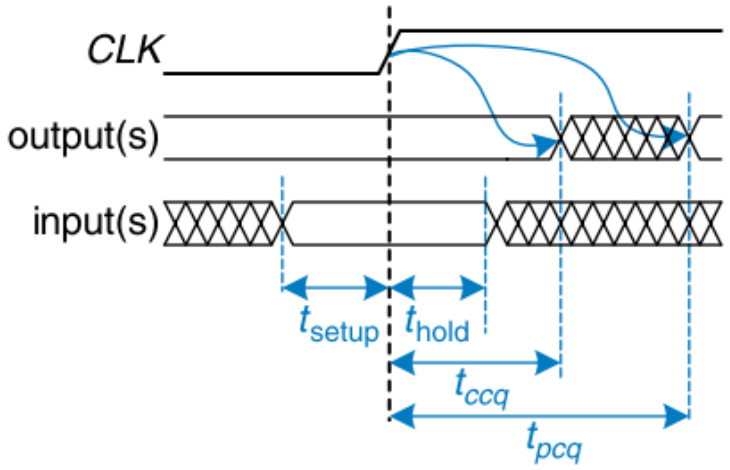
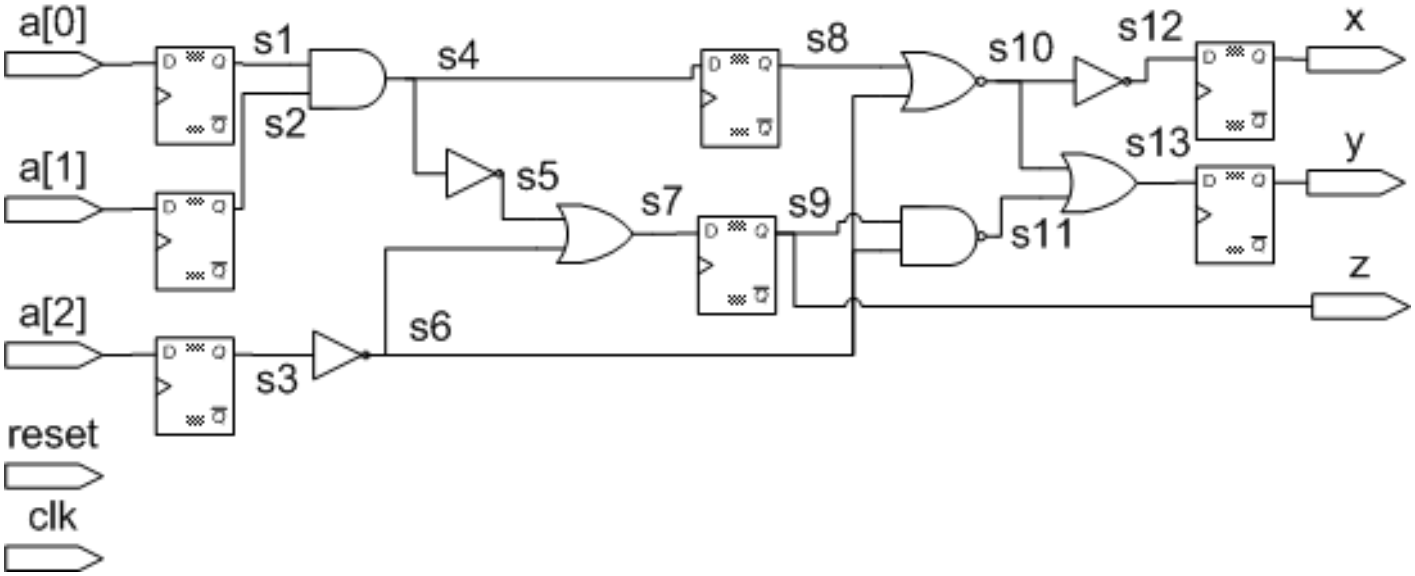



**Figure 3.35** **Timing specification for synchronous sequential circuit**

* Digital Design and Computer Architecture. Sec 3.5

# Extra-classe: Entrega na Próxima Aula

- Atrasos:
  - Inversores: 1ns
  - NAND e NOR: 3ns
  - AND e OR: 5ns
  - FF: 10ns

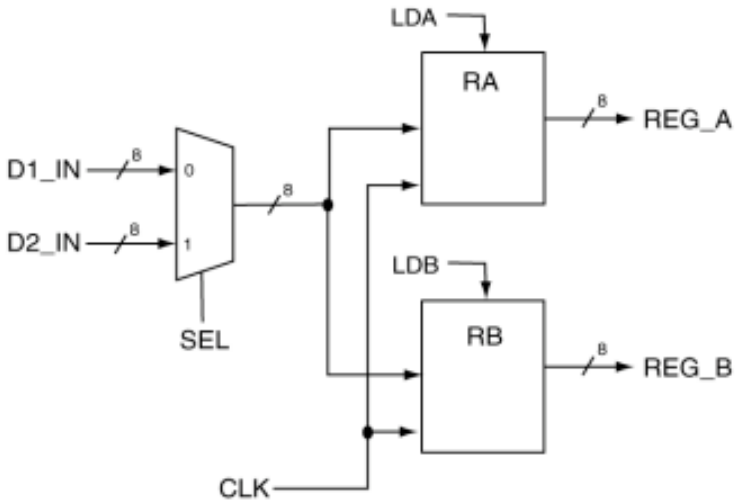



Implementar em VHDL e simular 2 cenários:
- período do clock MAIOR que caminho crítico
- período do clock MENOR que caminho crítico

# RTL

# Register Transfer Level (RTL)

**EXAMPLE 24.** Use VHDL behavioral modeling to design the circuit shown on the right. Consider both the loading signals to be active high. Consider the circuit to be synchronized to the rising edge of the clock signal.

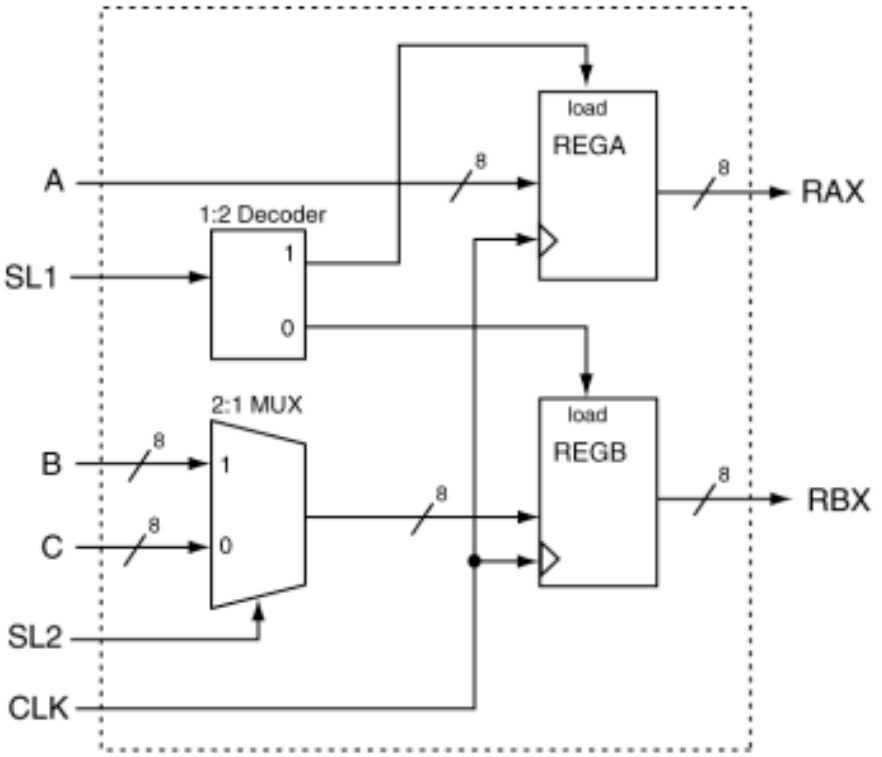



Solução: Listing 9.2, Free Range VHDL

# Atividades Extra-Classe

- Fazer pelo menos 2 exercícios da Seção 9.2 do livro Free Range VHDL

**EXERCISE 5.** Provide a VHDL model that can be used to implement the following circuit.



# BLOCOS SEQUENCIAIS

# Contador simples

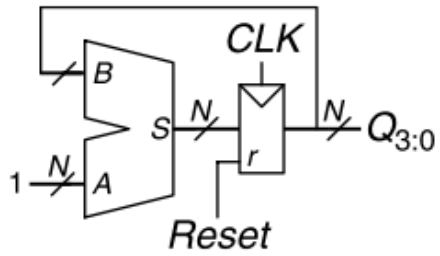



**Figure 5.31** ***N*-bit counter**

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity counter is
  generic (N: integer := 8);
  port (clk,
        reset: in STD_LOGIC;
        q: buffer STD_LOGIC_VECTOR(N-1 downto 0));
end;

architecture synth of counter is
begin
  process (clk, reset) begin
    if reset = '1' then
      q <= CONV_STD_LOGIC_VECTOR (0, N);
    elsif clk'event and clk = '1' then
      q <= q + 1;
    end if;
  end process;
end;
```

* Digital Design and Computer Architecture. Sec 5.4

# Registrador de Deslocamento

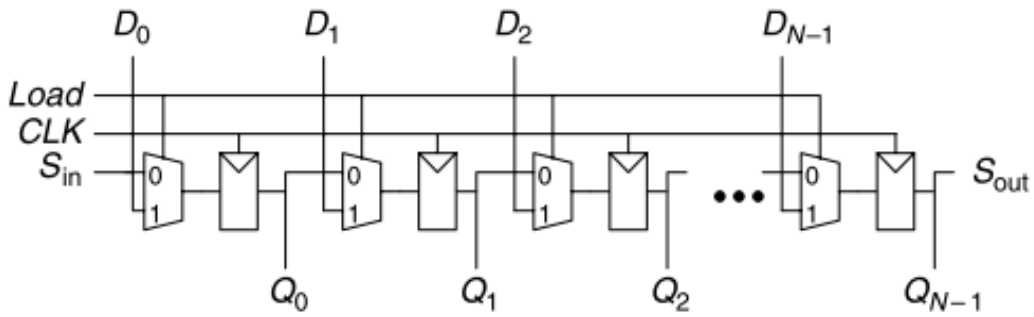
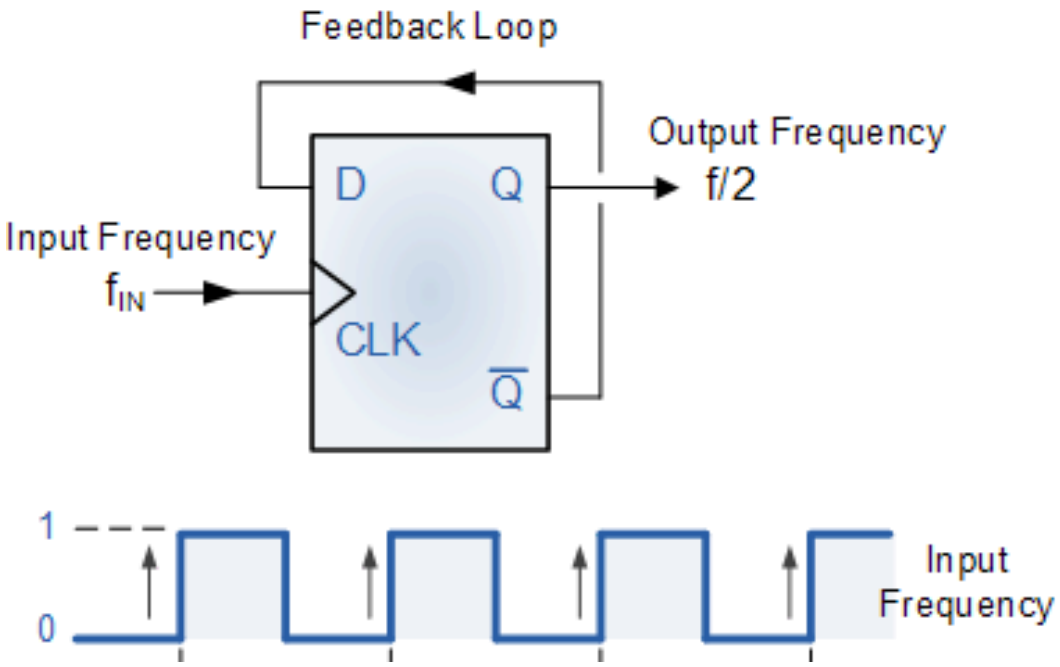



**Figure 5.35** Shift register with parallel load

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity shiftreg is
  generic (N: integer := 8);
  port(clk, reset,
       load: in    STD_LOGIC;
       sin:  in    STD_LOGIC;
       d:    in    STD_LOGIC_VECTOR(N-1 downto 0);
       q: buffer   STD_LOGIC_VECTOR(N-1 downto 0);
       sout: out   STD_LOGIC);
end;

architecture synth of shiftreg is
begin
  process (clk, reset) begin
    if reset = '1' then
      q <= CONV_STD_LOGIC_VECTOR (0, N);
    elsif clk'event and clk = '1' then
      if load = '1' then
        q <= d;
      else
        q <= q(N-2 downto 0) & sin;
      end if;
    end if;
  end process;
  sout <= q(N-1);
end;
```

*Free Range

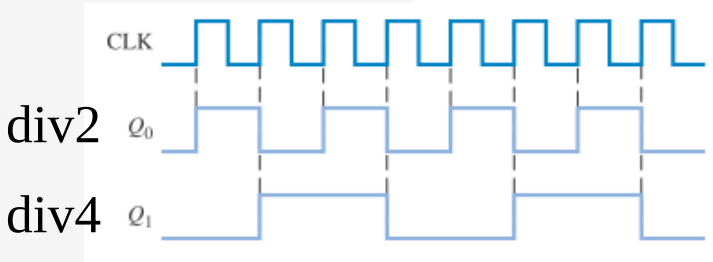
# Divisor de Clock



# Divisor de Clock

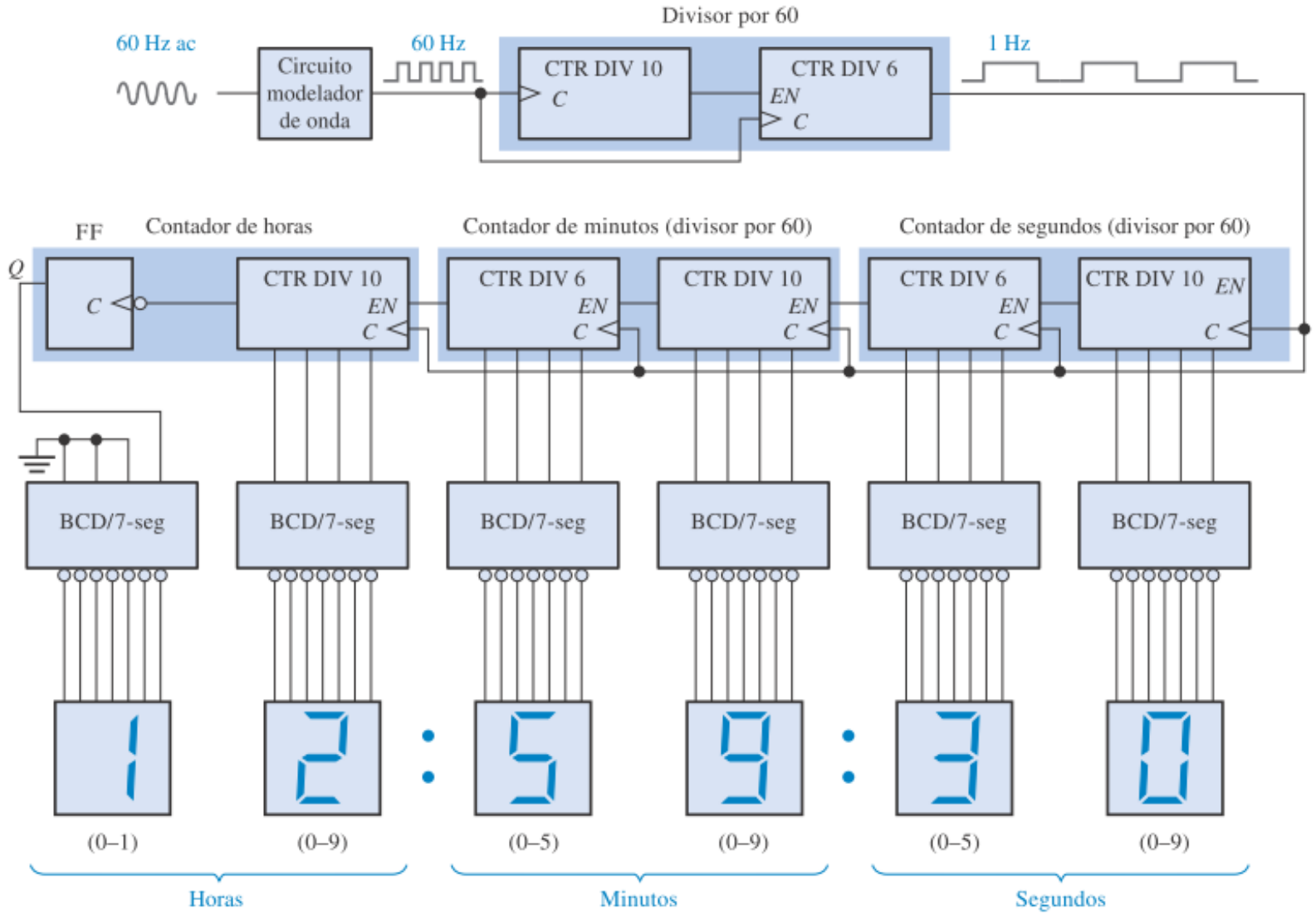
**Listing 10.3: Solution to Example 26.**

```
1
2  -- library declaration
3  library IEEE;
4  use IEEE.std_logic_1164.all;
5  use IEEE.numeric_std.all;
6
7  -- entity
8  entity clk_div is
9  Port (
10     clk    : in  std_logic;
11     div_en : in  std_logic;
12     sclk   : out std_logic);
13 end clk_div;
14
15 -- architecture
16 architecture my_clk_div of clk_div is
17     type my_count is range 0 to 100;      -- user-defined type
18     constant max_count : my_count := 31; -- user-defined constant
19     signal tmp_sclk : std_logic;          -- intermediate signal
20     begin
21         my_div: process (clk,div_en)
22         variable div_count : my_count := 0;
23         begin
24             if (div_en = '0') then
25                 div_count := 0;
26                 tmp_sclk <= '0';
27             elsif (rising_edge(clk)) then
28                     -- divider enabled
29                     if (div_count = max_count) then
30                         tmp_sclk <= not tmp_sclk;    -- toggle output
31                         div_count := 0;              -- reset count
32                     else
33                         div_count := div_count + 1; -- count
34                     end if;
35             end if;
36         end process my_div;
37         sclk <= tmp_sclk;                          -- final assignment
38 end my_clk_div;
```




*Free Range

# Aplicação de Divisores




▲ **FIGURA 8–51**

Diagrama lógico simplificado para um relógio digital de 12 horas. Os detalhes lógicos usando dispositivos específicos são mostrados na Figura 8–52 e 8–53.

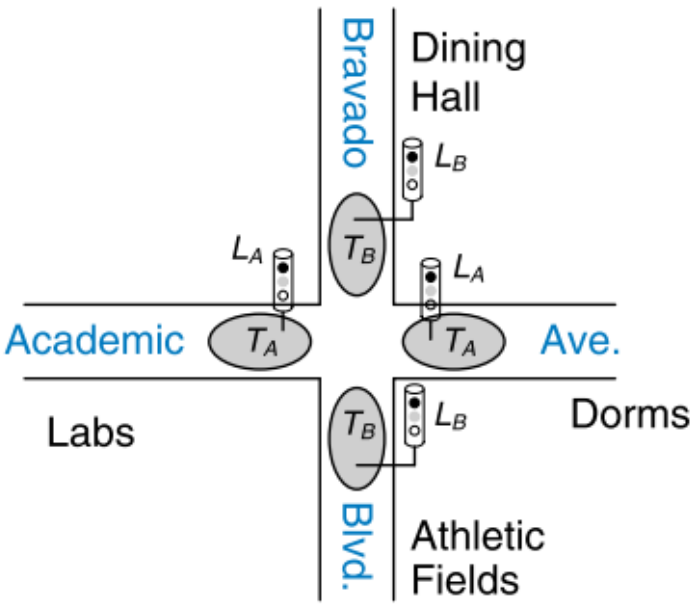
*Floyd

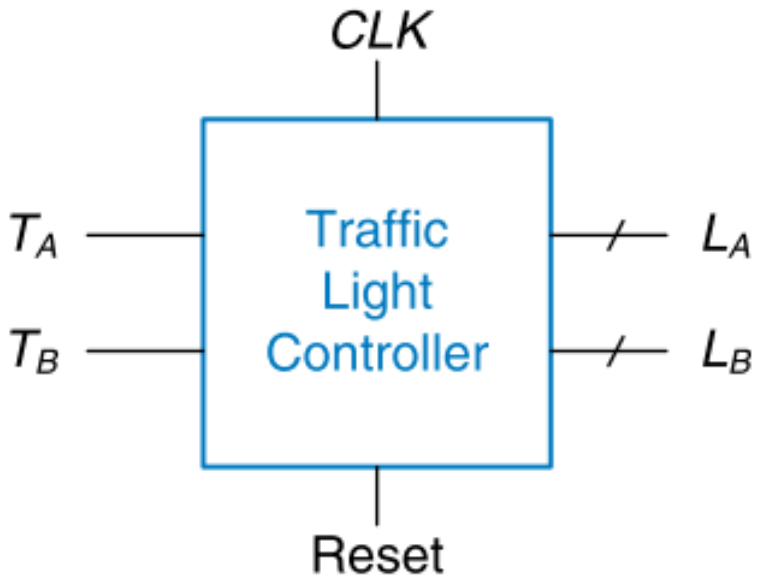
# Atividade da Próxima Aula

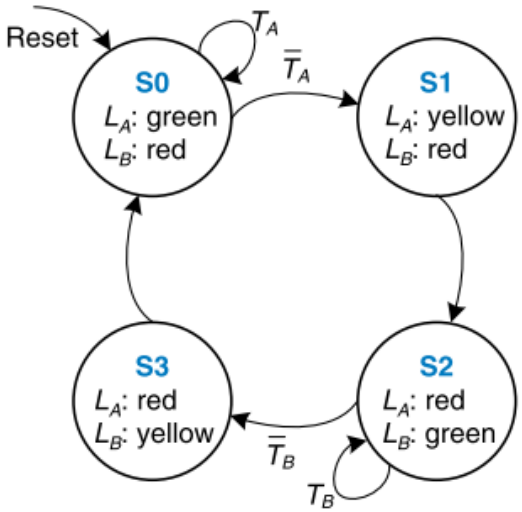
- Atividade p Próxima Aula
  - Baseado no exemplo anterior do contador usando flops e um somador '+1', desenvolva a seguinte extensão:
    - contador (flop + 1) que tenha opção de ‘reset‘ para zerar o contador e ‘enable’ para carga de valor inicial em ‘datain’. desta forma, a entidade deve ter os seguintes pinos:
    - clock (in 1bit)
    - reset (in 1bit)
    - enable (in 1bit)
    - datain (in 4 bits)
    - dataout (out 4 bits)
  - o pino 'datain' tem  o valor inicial, que deve ser  carregado no flop quando o 'enable' estiver habilitado.
  - o pino dataout tem o valor atual do contador.

# FSM – FINITE STATE MACHINE

# Problema → FSM → VHDL




VHDL

Solução completa em “Digital Design and Computer Architecture”. Sec 3.4.15

# Tipos de FSM

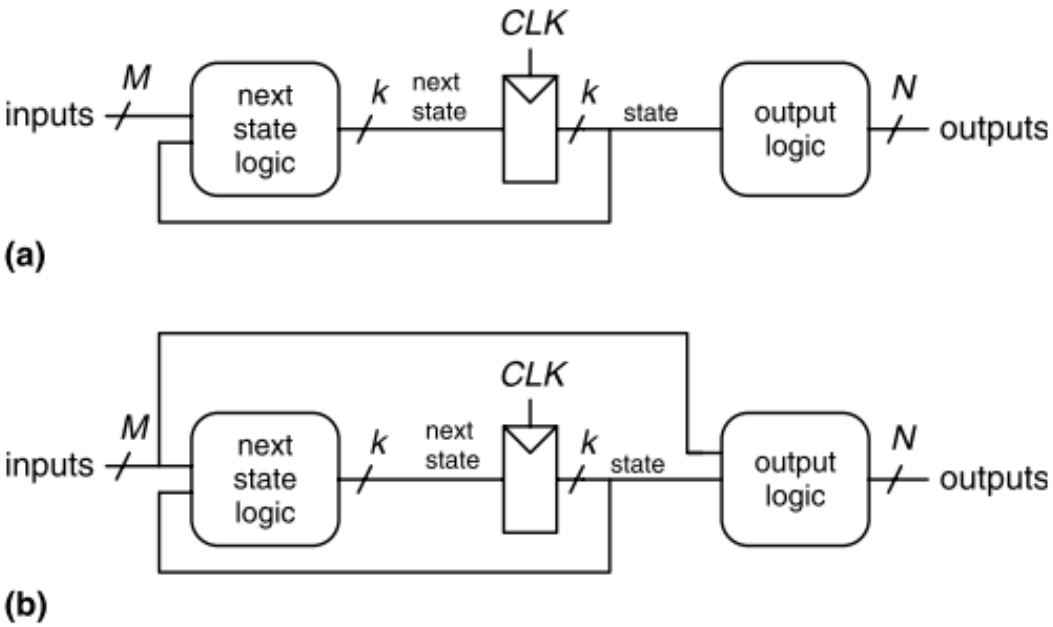



**Figure 3.22** **Finite state machines: (a) Moore machine, (b) Mealy machine**

- Moore: valor de saída depende somente do estado atual
- Mealy: valor de saída depende do estado atual e das entradas

# Exemplo em VHDL

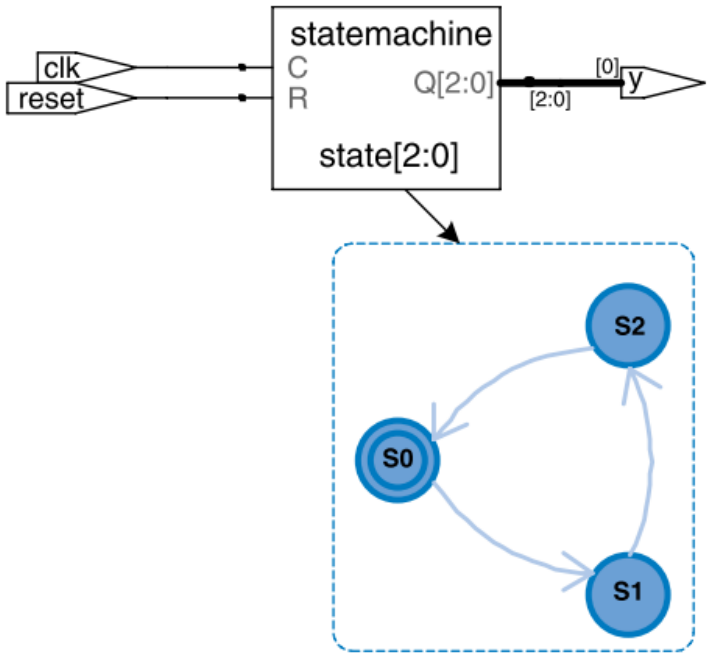
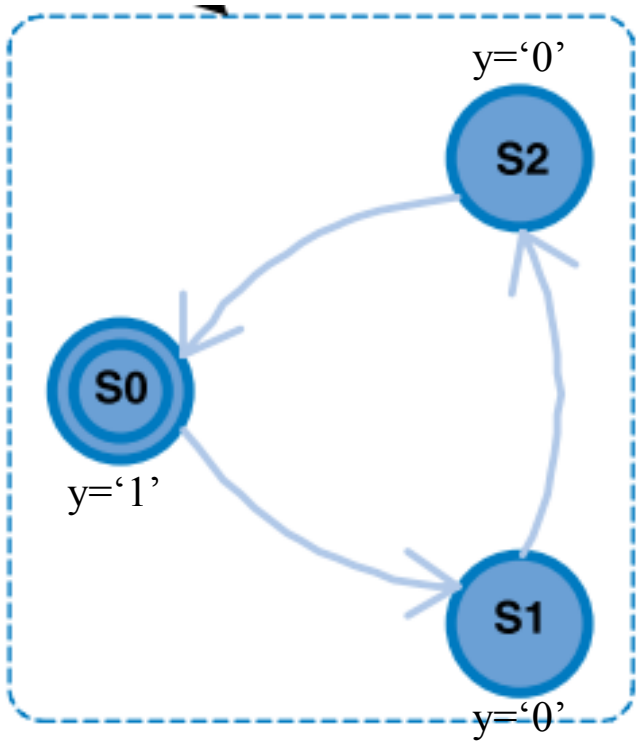
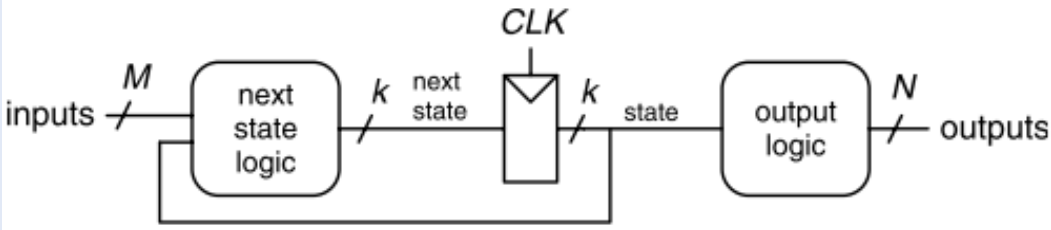


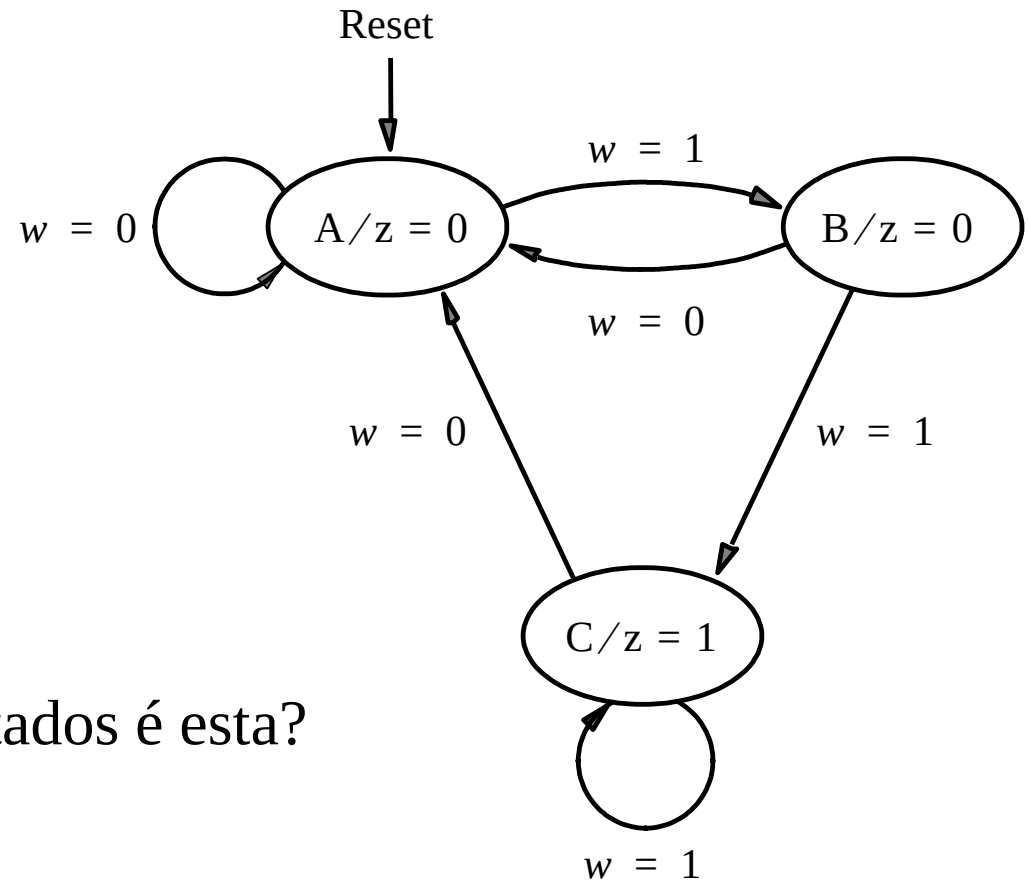

**Figure 4.27** `divideby3fsm` **synthesized circuit**

# FSM (Moore) em VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity divideby3FSM is
  port(clk, reset: in  STD_LOGIC;
       y:          out STD_LOGIC);
end;

architecture synth of divideby3FSM is
  type statetype is (S0, S1, S2);
  signal state, nextstate: statetype;
begin
  -- state register
  process (clk, reset) begin
    if reset = '1' then state <= S0;
    elsif clk'event and clk = '1' then
      state <= nextstate;
    end if;
  end process;

  -- next state logic
  nextstate <= S1 when state = S0 else
               S2 when state = S1 else
               S0;

  -- output logic
  y <= '1' when state = S0 else '0';
end;
```






Que tipo de máquina de estados é esta?
O que faz esta FSM ?

* Fundamentals of Digital Logic with VHDL Design

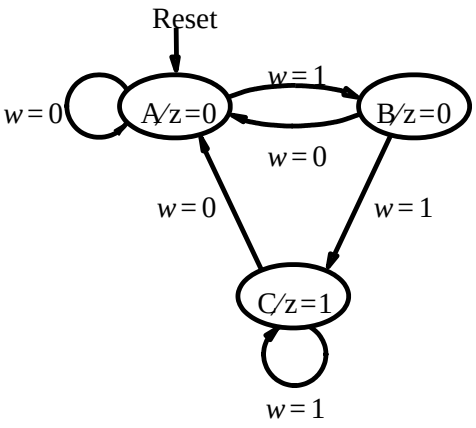
# FSM Moore

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY moore IS
        PORT (      Clock, Resetn, w    : IN        STD_LOGIC ;
                    z                   : OUT STD_LOGIC ) ;
END moore ;
```

Create a user-defined signal type.

y represents the state flip-flops

Process statement describes the FSM as a sequential circuit.




```
7   ARCHITECTURE Behavior OF moore IS
8       TYPE State_type IS (A, B, C) ;
9       SIGNAL y : State_type ;
10  BEGIN
11      PROCESS ( Resetn, Clock )
12      BEGIN
13          IF Resetn = '0' THEN
14              y <= A ;
15          ELSIF (Clock'EVENT AND Clock = '1') THEN
16              CASE y IS
17                  WHEN A =>
18                      IF w = '0' THEN
19                          y <= A ;
20                      ELSE
21                          y <= B ;
22                      END IF ;
23                  WHEN B =>
24                      IF w = '0' THEN
25                          y <= A ;
26                      ELSE
27                          y <= C ;
28                      END IF ;
29                  WHEN C =>
30                      IF w = '0' THEN
31                          y <= A ;
32                      ELSE
33                          y <= C ;
34                      END IF ;
35              END CASE ;
36          END IF ;
37          END PROCESS ;
38          z <= '1' WHEN y = C ELSE '0' ;
39  END Behavior ;
```

VHDL compiler automatically determines the correct number of state flip-flops required.

Sensitivity list

Reset is asynchronous

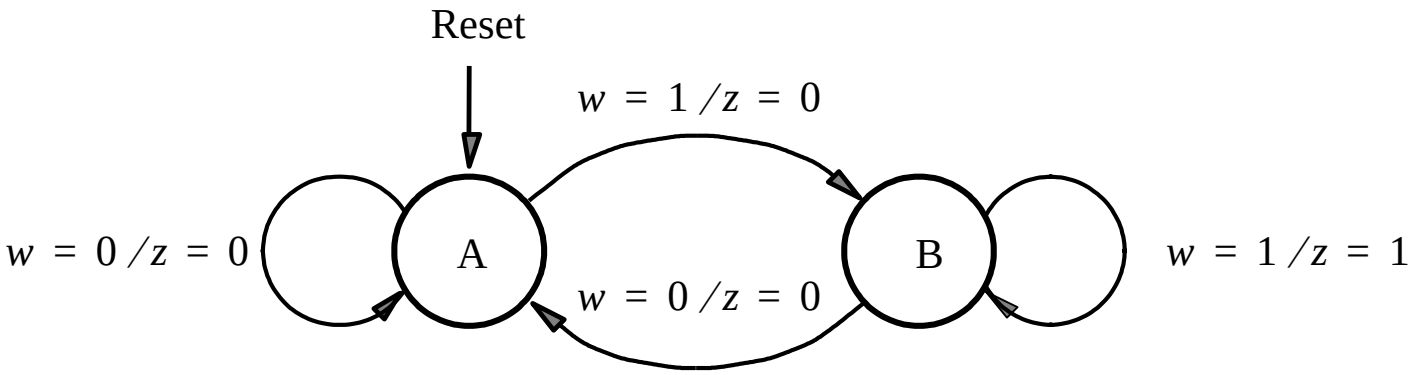
Positive edge-triggered

The Case statement describes the behavior of the FSM.

Each When clause represents one state in the FSM.

The When clauses correspond to the states in the state diagram.

Output specified.




Que tipo de máquina de estados é esta?
O que faz esta FSM ?

* Fundamentals of Digital Logic with VHDL Design

# FSM Mealy

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mealy IS
        PORT ( Clock, Resetn, w :   IN          STD_LOGIC ;
                        z       :   OUT         STD_LOGIC ) ;
END mealy ;
```

Create a user-defined signal type.

VHDL compiler automatically determines the correct number of state flip-flops required.

Sensitivity list

Reset is asynchronous

y represents the state flip-flops

Positive edge-triggered

Process statement describes the FSM as a sequential circuit.

This Case statement describes the behavior of the FSM.

This Case statement describes the behavior of the output.

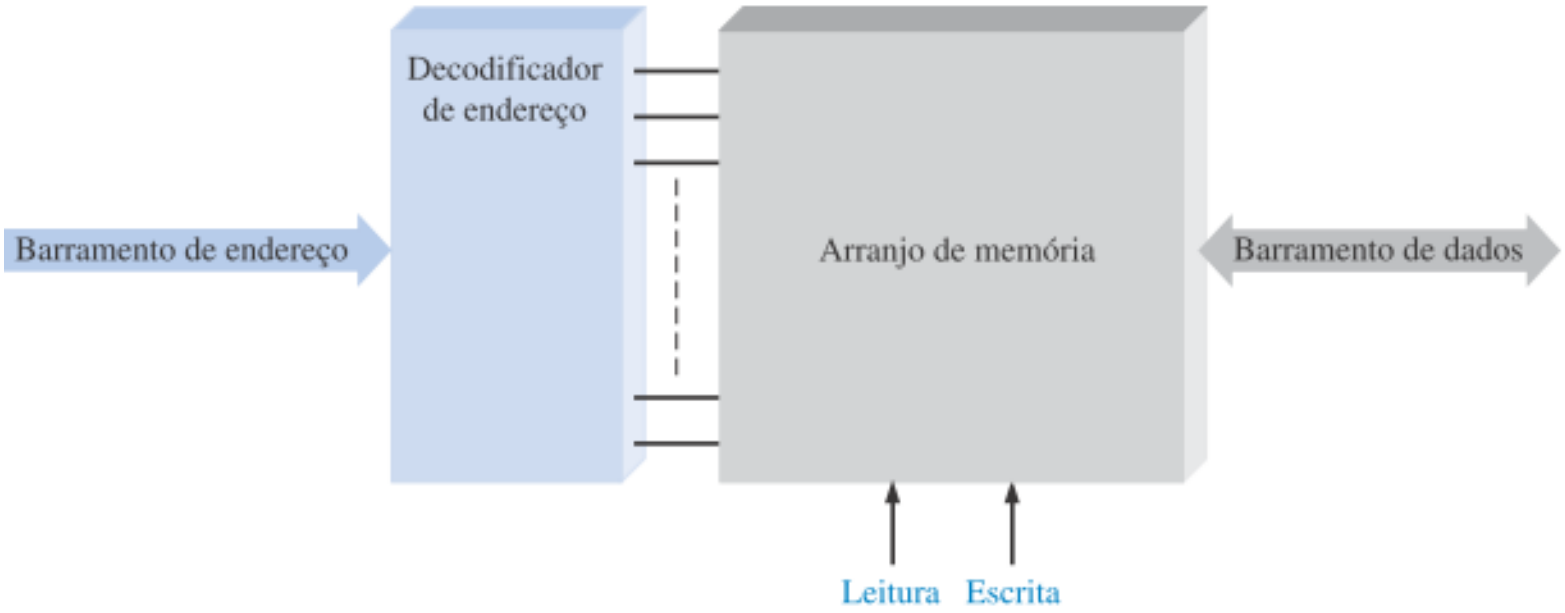
```
ARCHITECTURE Behavior OF mealy IS
    TYPE State_type IS (A, B) ;
    SIGNAL y : State_type ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
    IF Resetn = '0' THEN
            y <= A ;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
    CASE y IS
    WHEN A =>
            IF w = '0' THEN y <= A ;
            ELSE y <= B ;
            END IF ;
     WHEN B =>
            IF w = '0' THEN y <= A ;
            ELSE y <= B ;
            END IF ;
    END CASE ;
    END IF ;
    END PROCESS ;

    PROCESS ( y, w )
    BEGIN
    CASE y IS
            WHEN A =>
                    z <= '0' ;
            WHEN B =>
                    z <= w ;
    END CASE ;
    END PROCESS ;
END Behavior ;
```

# Atividade da Próxima Aula

- Atividade p Próxima Aula
  - Entregar no início da aula
  - Descrever em VHDL
    - Comportamento
      - Descrever FSM que, quando encontrar o padão “10101” na entrada *datain*, ativa a saída *found*.
    - Entidade
      - datain : in bit
      - Clk, rest:  in bit
      - found: out bit

# MEMÓRIAS

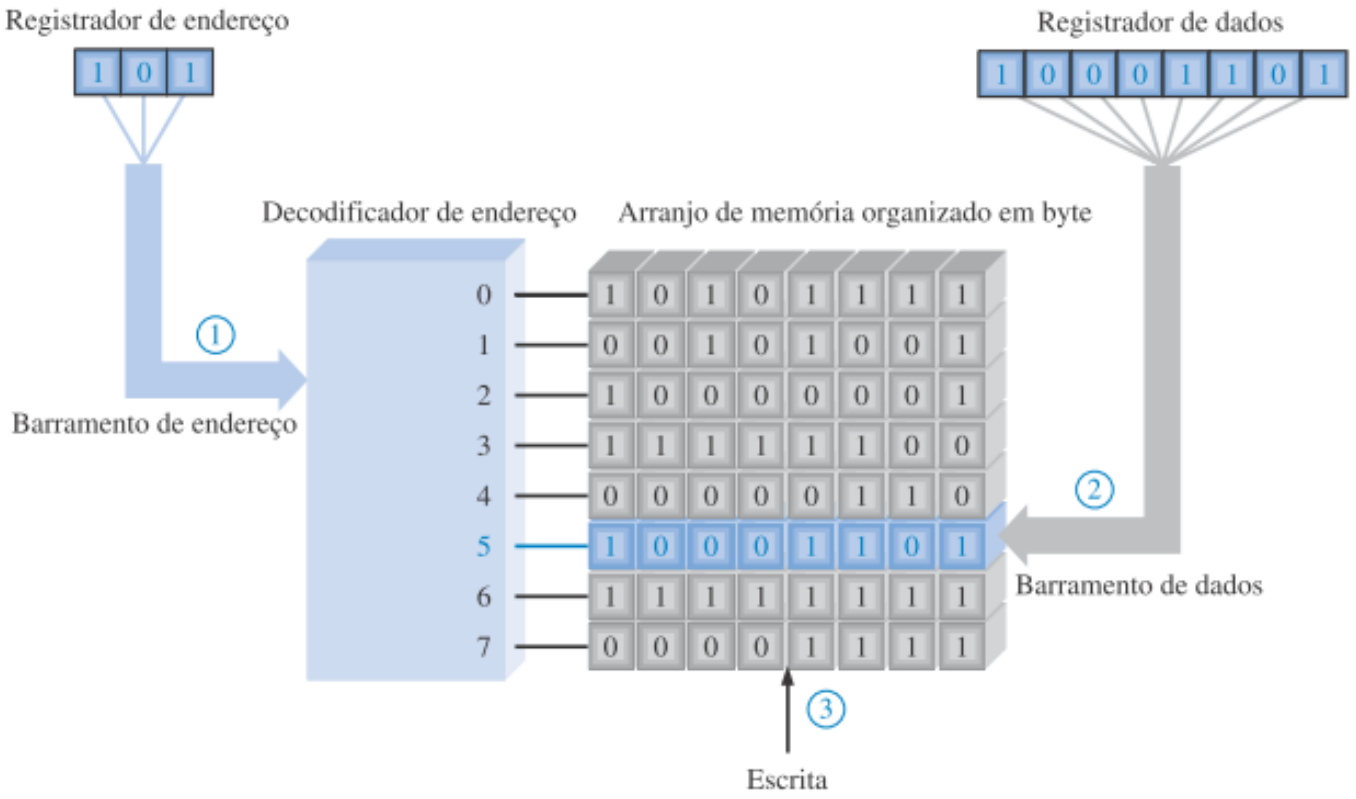
# Memória




(a) Arranjo de memória de 2 dimensões

*Floyd. Cap 10

# Escrita em Memória

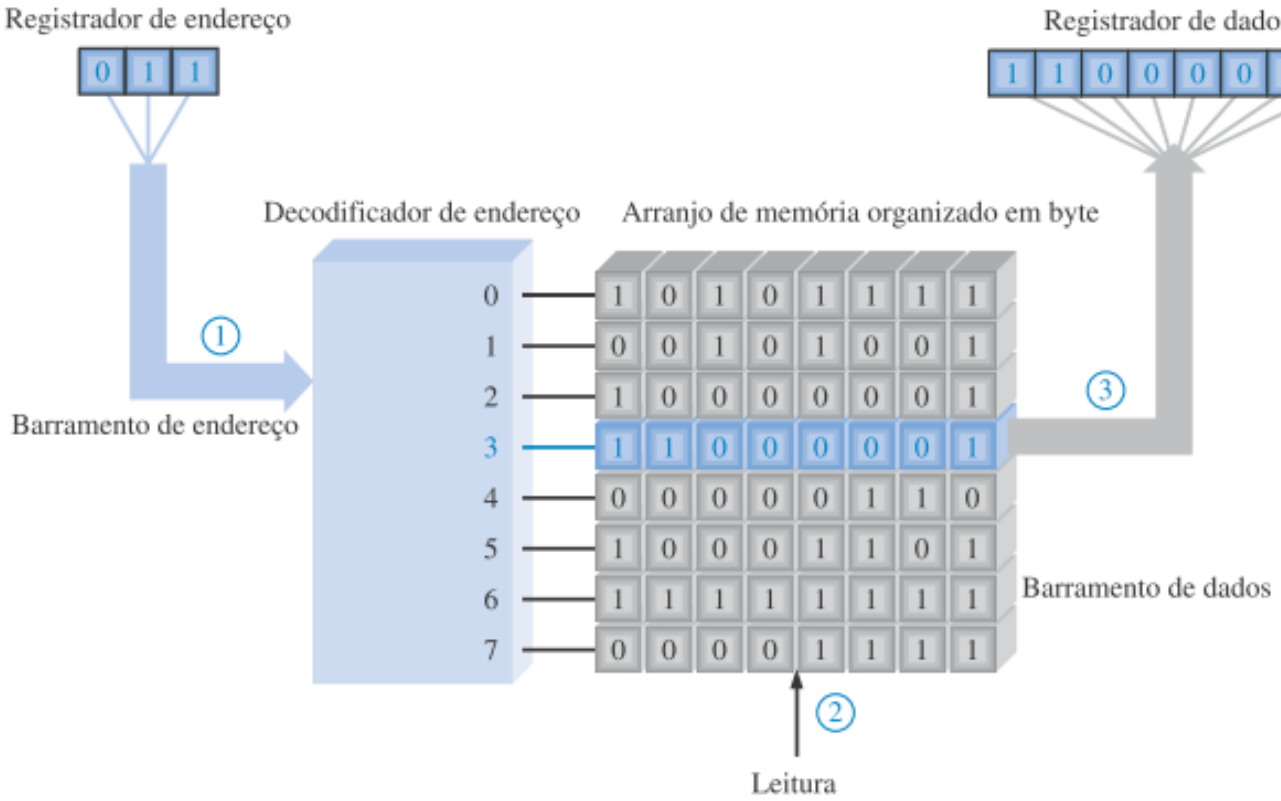



1. O código de endereço 101 é colocado no barramento de endereço e o endereço 5 é selecionado.
2. O byte de dados é colocado no barramento de dados.
3. O comando de escrita faz com que o byte de dados seja armazenado no endereço 5, substituindo o dado anterior.

**◀ FIGURA 10–5**

Ilustração da operação de escrita.

# Leitura em Memória

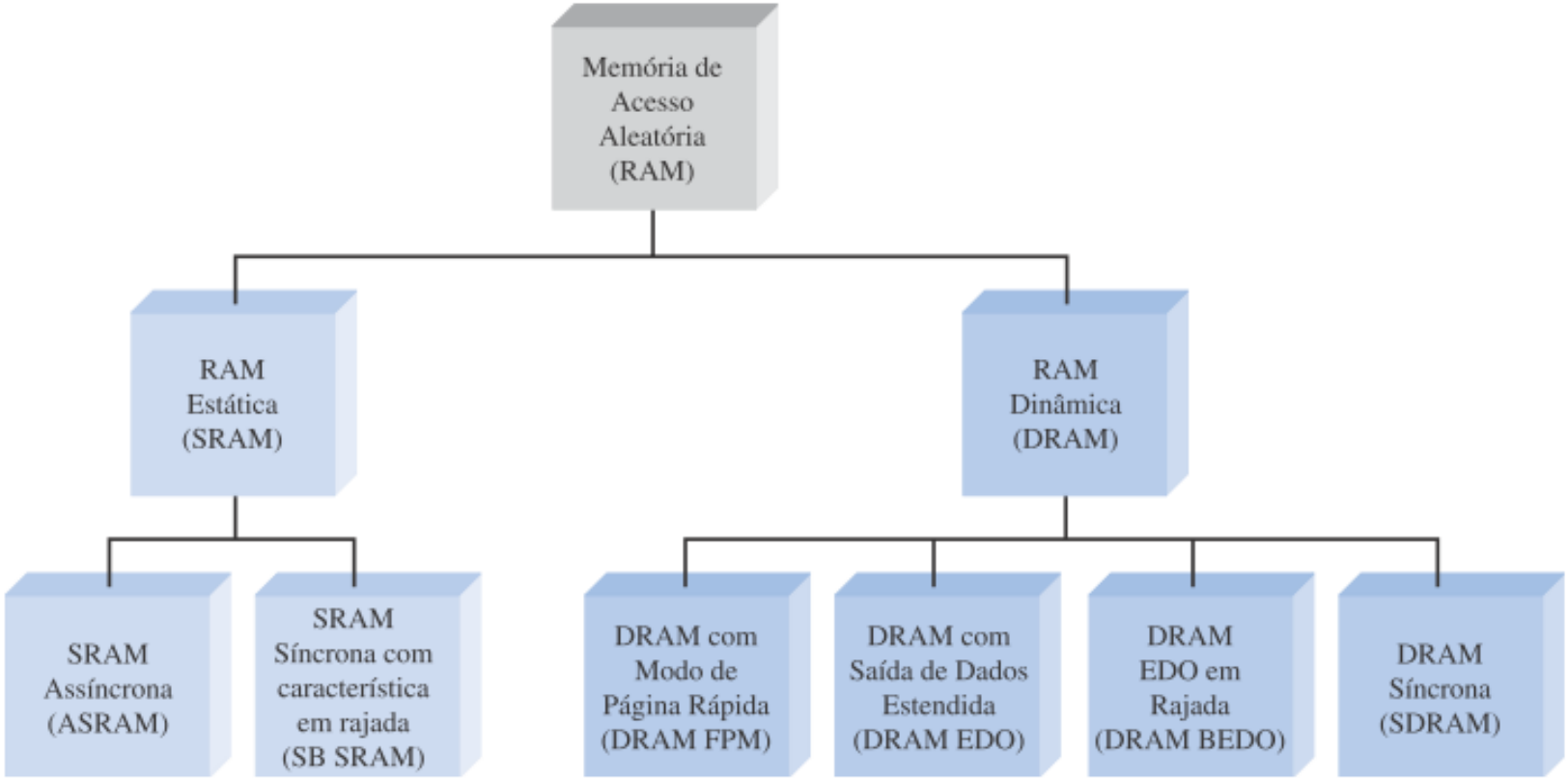



1. O código de endereço 011 é colocado no barramento de endereço e o endereço 3 é selecionado.
2. O comando de leitura é aplicado.
3. O conteúdo do endereço 3 é colocado no barramento de dados e deslocado no registrador de dados. O conteúdo do endereço 3 não é apagado pela operação de leitura.

◀ **FIGURA 10–6**

Ilustração da operação de leitura.

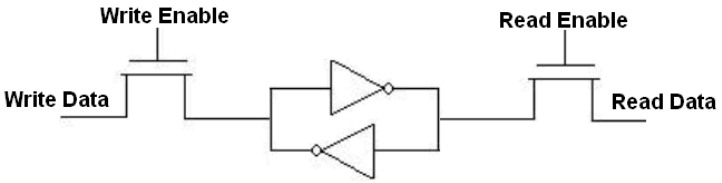
# Tipos de RAM

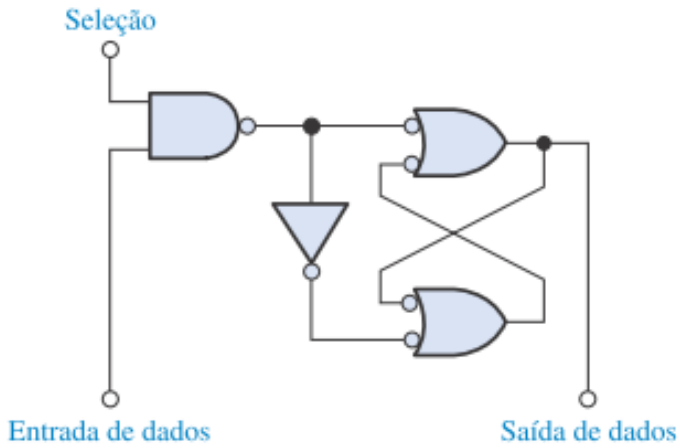


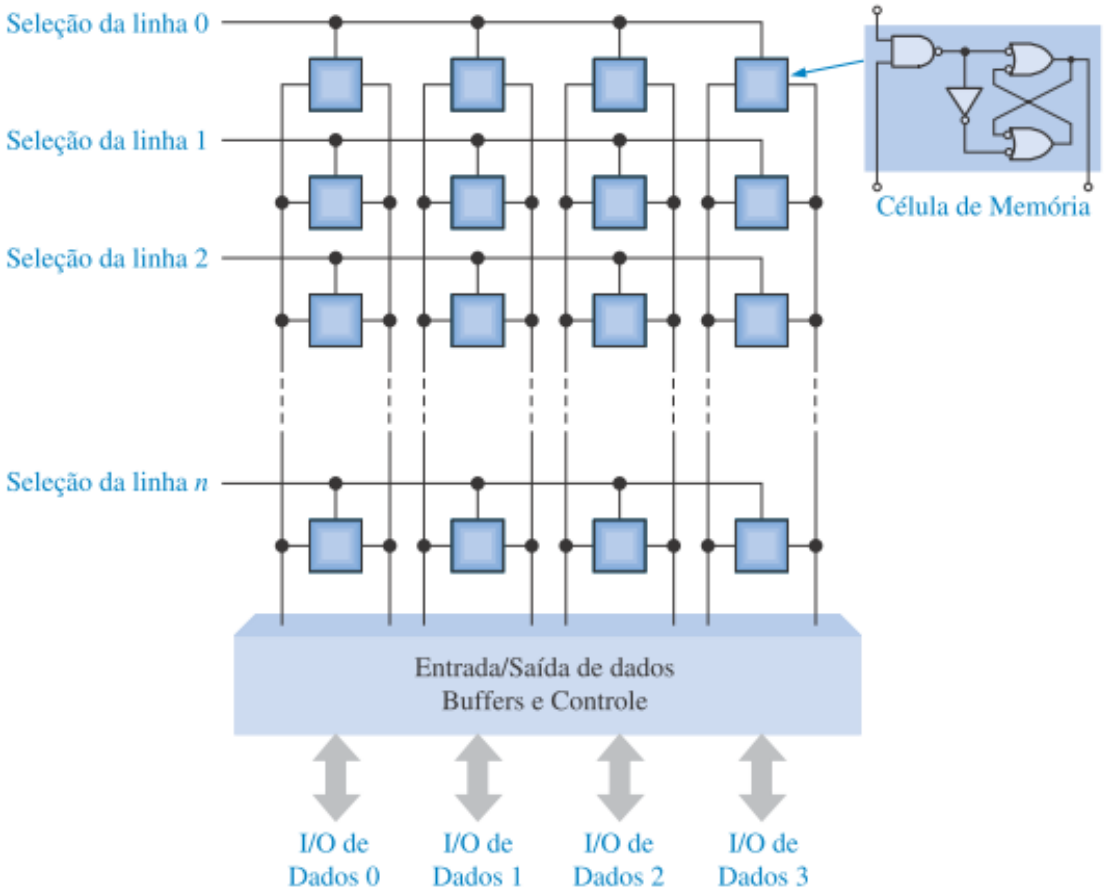

▲ **FIGURA 10–7**

A família RAM.

# SRAM



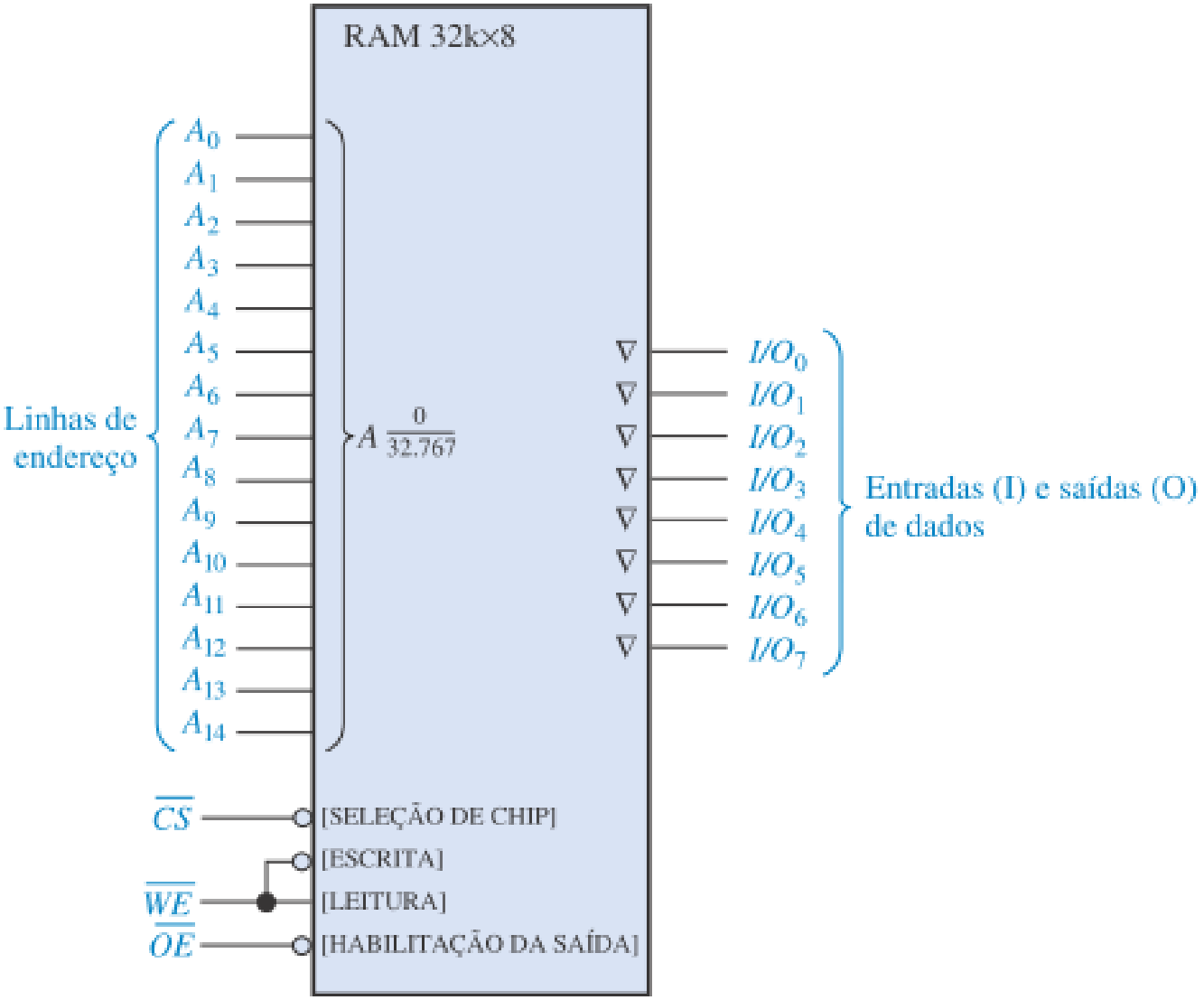




◀ **FIGURA 10–8**

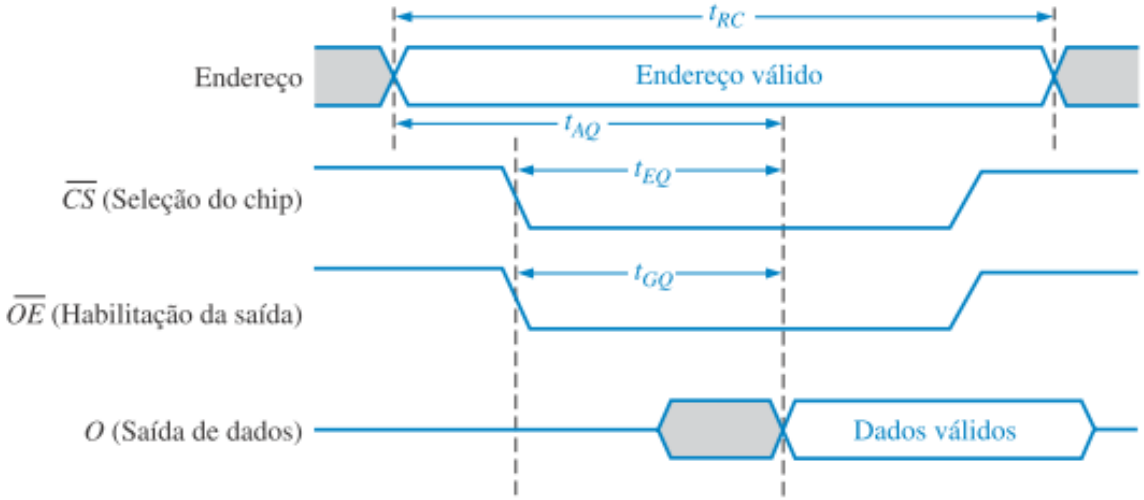
Uma típica célula de memória latch SRAM.

# Memória SRAM

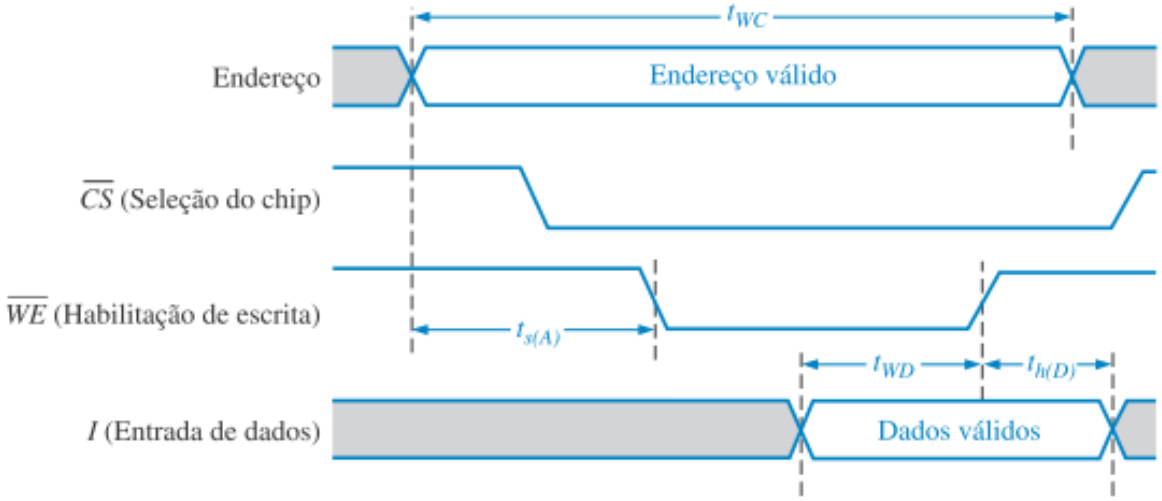



Quantos bits de endereço deve ter uma memória de 4Mx8 ?

# Ciclo de Leitura e Escrita




(a) Ciclo de leitura ($\overline{WE}$ em nível ALTO)




(b) Ciclo de escrita ($\overline{WE}$ em nível BAIXO)

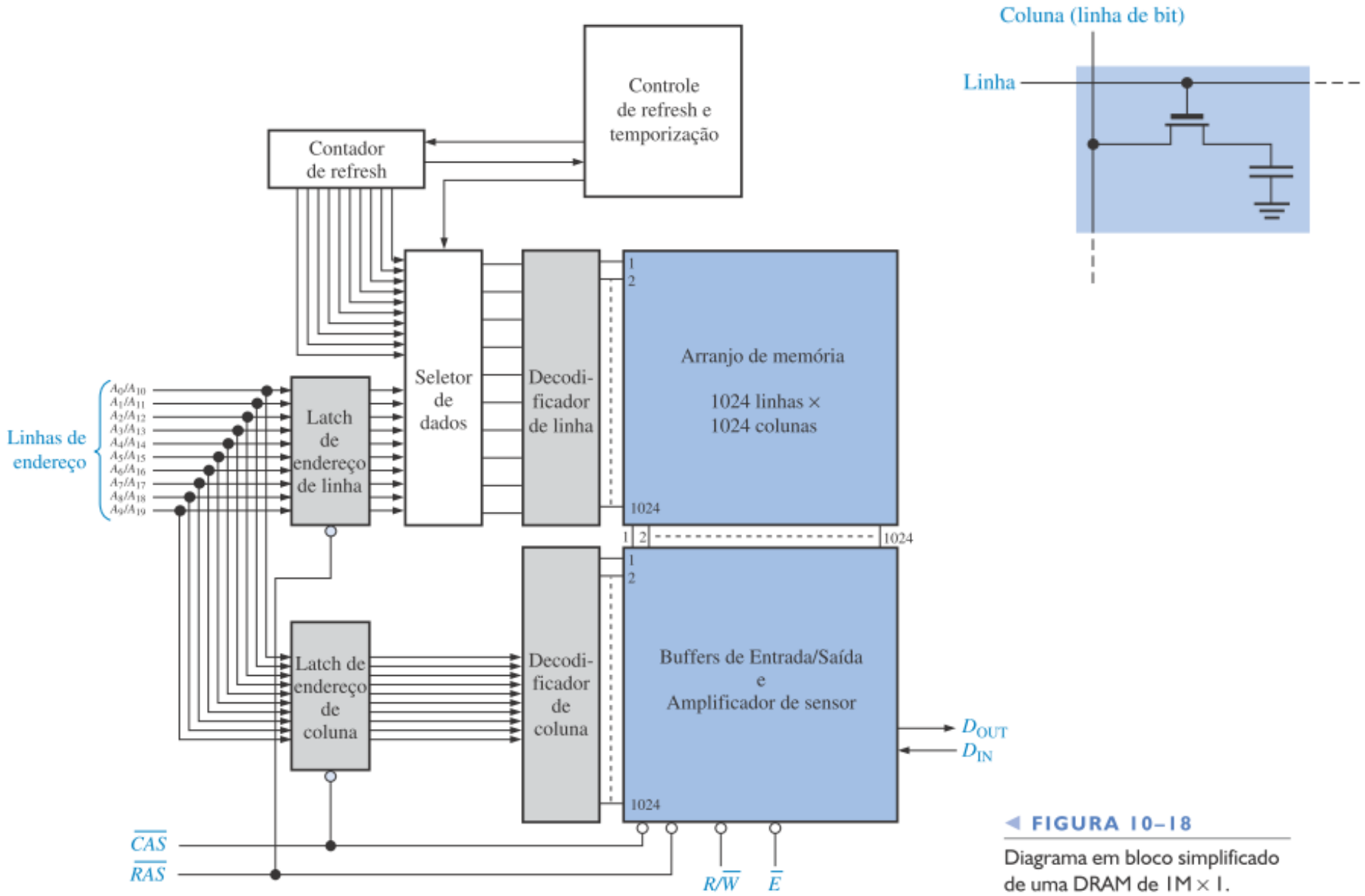
# SRAM em VHDL

**VHDL**

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ram_array is
  generic (N: integer := 6; M: integer := 32);
  port (clk,
        we:   in  STD_LOGIC;
        adr:  in  STD_LOGIC_VECTOR(N-1 downto 0);
        din:  in  STD_LOGIC_VECTOR(M-1 downto 0);
        dout: out STD_LOGIC_VECTOR(M-1 downto 0));
end;

architecture synth of ram_array is
  type mem_array is array((2**N-1) downto 0)
       of STD_LOGIC_VECTOR(M-1 downto 0);
  signal mem: mem_array;
begin
  process (clk) begin
    if clk' event and clk = '1' then
      if we = '1' then
        mem (CONV_INTEGER (adr)) <= din;
      end if;
    end if;
  end process;

  dout <= mem (CONV_INTEGER(adr));
end;
```

* Digital Design and Computer Architecture. Sec 5.5

# DRAM



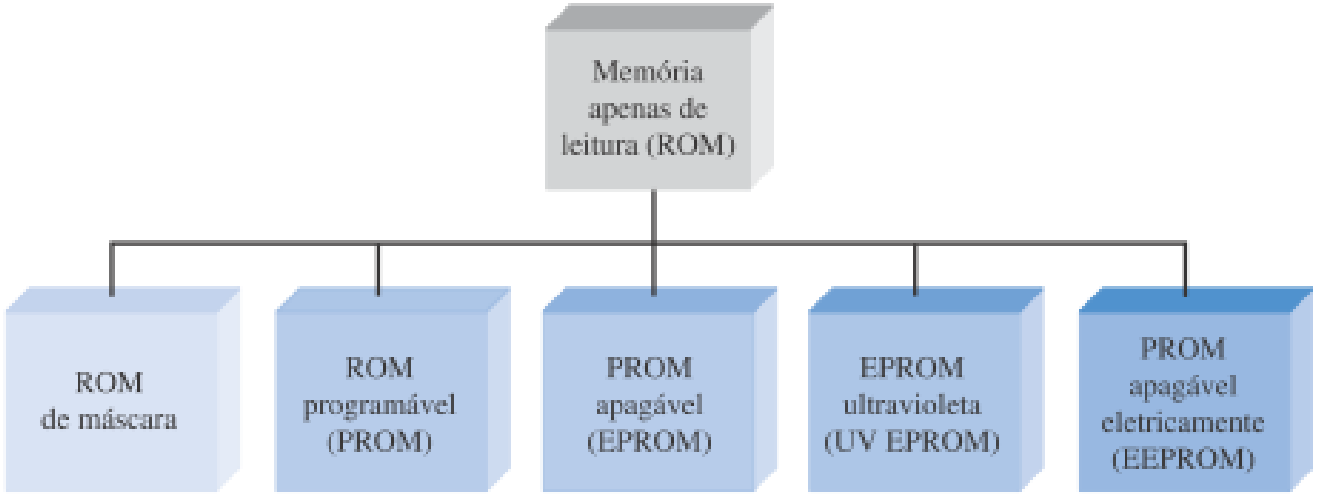




◀ **FIGURA 10–18**

Diagrama em bloco simplificado de uma DRAM de 1M × 1.

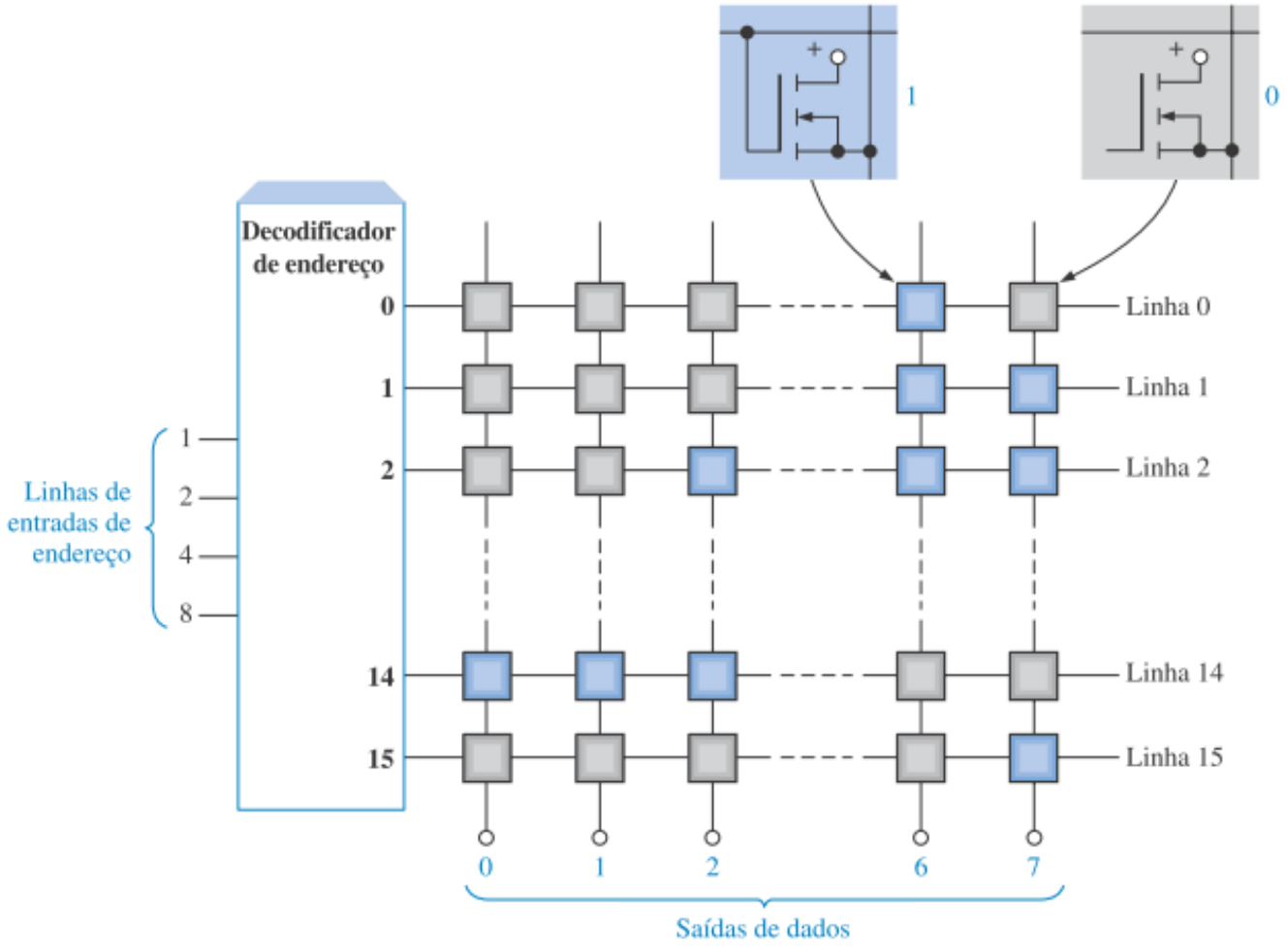
# Tipos de Memória ROM




◀ FIGURA 10–22

Família de ROMs.

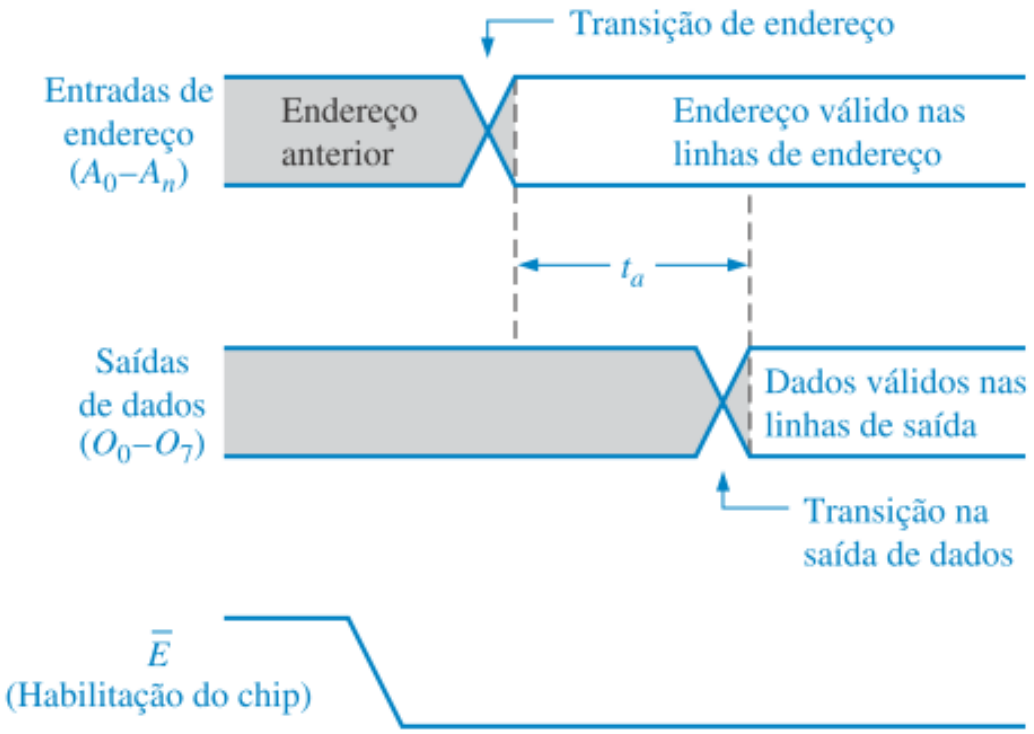
# ROM 16 x 8 bits




▲ **FIGURA 10–24**

Uma representação de um arranjo de uma ROM de 16 × 8 bits.

# Temporização da ROM



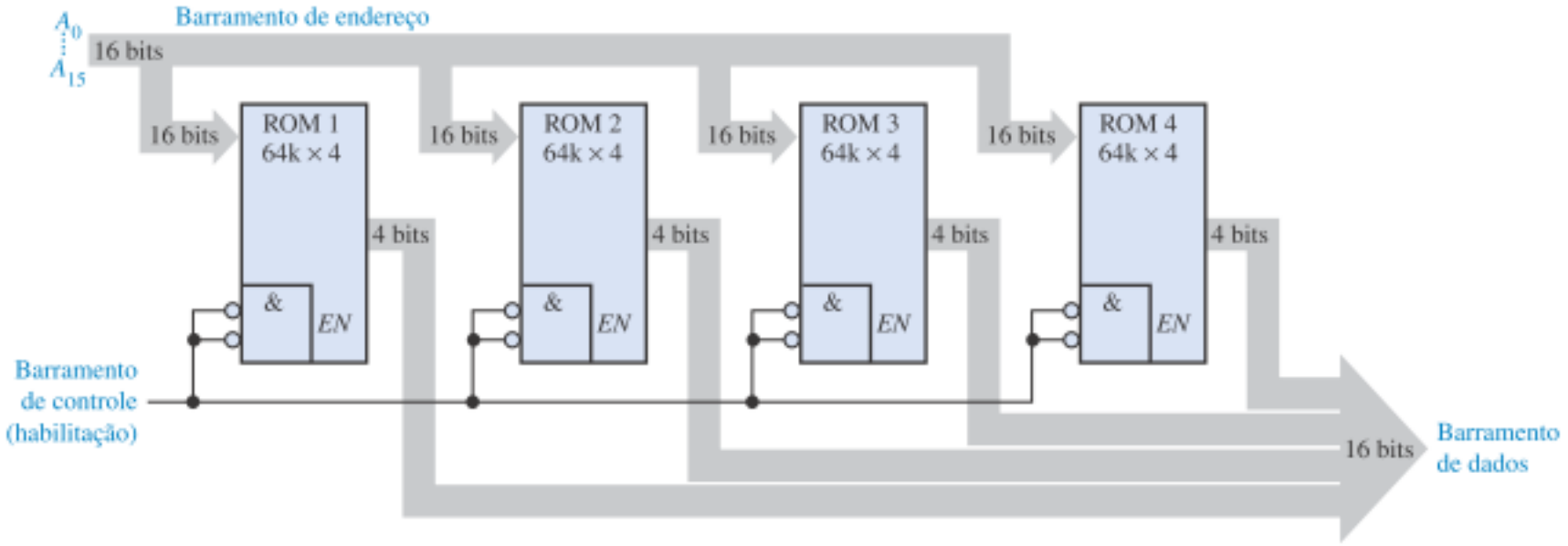

► **FIGURA 10–28**

Tempo de acesso ($t_a$) de uma ROM a partir da mudança de endereço para a saída de dados com a habilitação do chip já ativa.

# ROM em VHDL

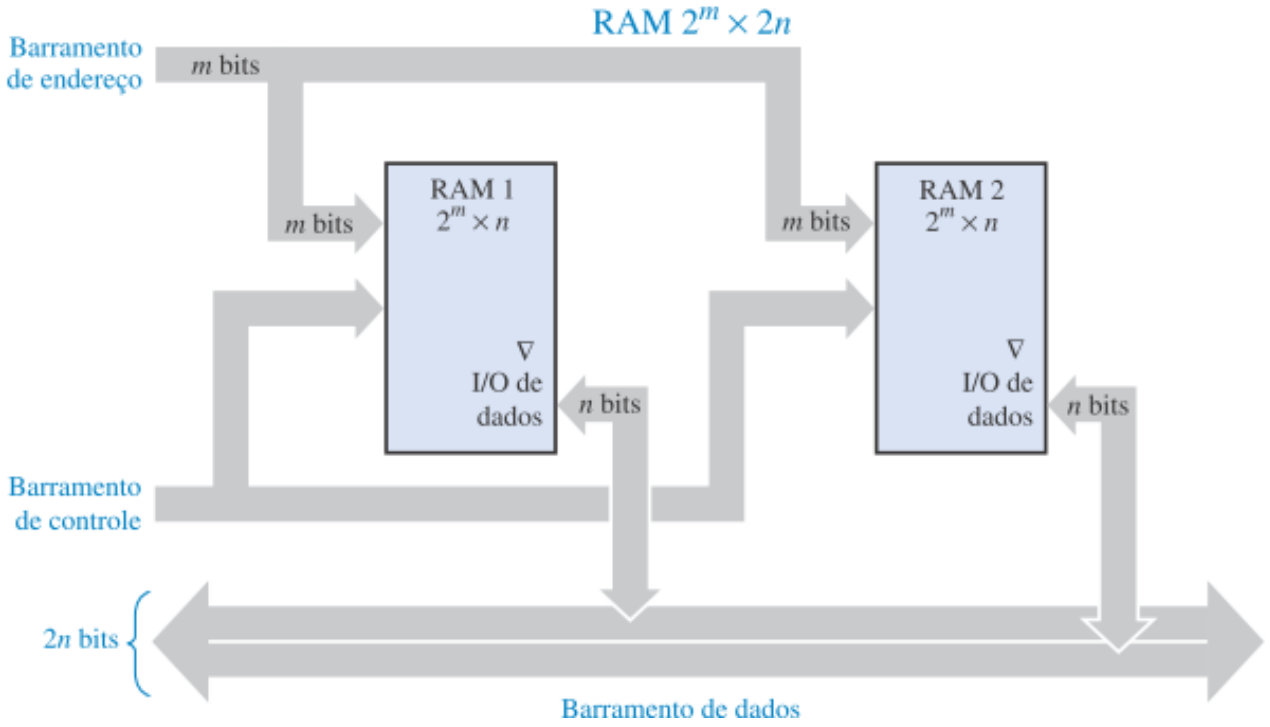
```
-- typical custom data type for a 20-byte ROM
type memory is array (0 to 19 of std_logic_vector(7 downto 0);
constant my_rom : memory := (
         1 => "11111111"
         2 => "11110111"
         5 => "11001111"
        12 => "10110101"
        18 => "10001101"
    others => "00000000");
```

# Combinação de ROMs




▲ FIGURA 10–41

# Combinação de RAMs




► **FIGURA 10–42**

Ilustração da expansão do tamanho da palavra com duas RAMs de $2^m \times n$ formando uma RAM de $2^m \times 2n$.

# Resumo dos tipos de memórias

**TABELA 10–2**

Comparações entre tipos de memórias

| TIPO DE MEMÓRIA | NÃO-VOLÁTIL | ALTA DENSIDADE | CÉLULA DE UM TRANSISTOR | ESCRITA NO PRÓPRIO SISTEMA |
|---|---|---|---|---|
| Flash | Sim | Sim | Sim | Sim |
| SRAM | Não | Não | Não | Sim |
| DRAM | Não | Sim | Sim | Sim |
| ROM | Sim | Sim | Sim | Não |
| EPROM | Sim | Sim | Sim | Não |
| EEPROM | Sim | Não | Não | Sim |

# PROCESSADOR GENÉRICO

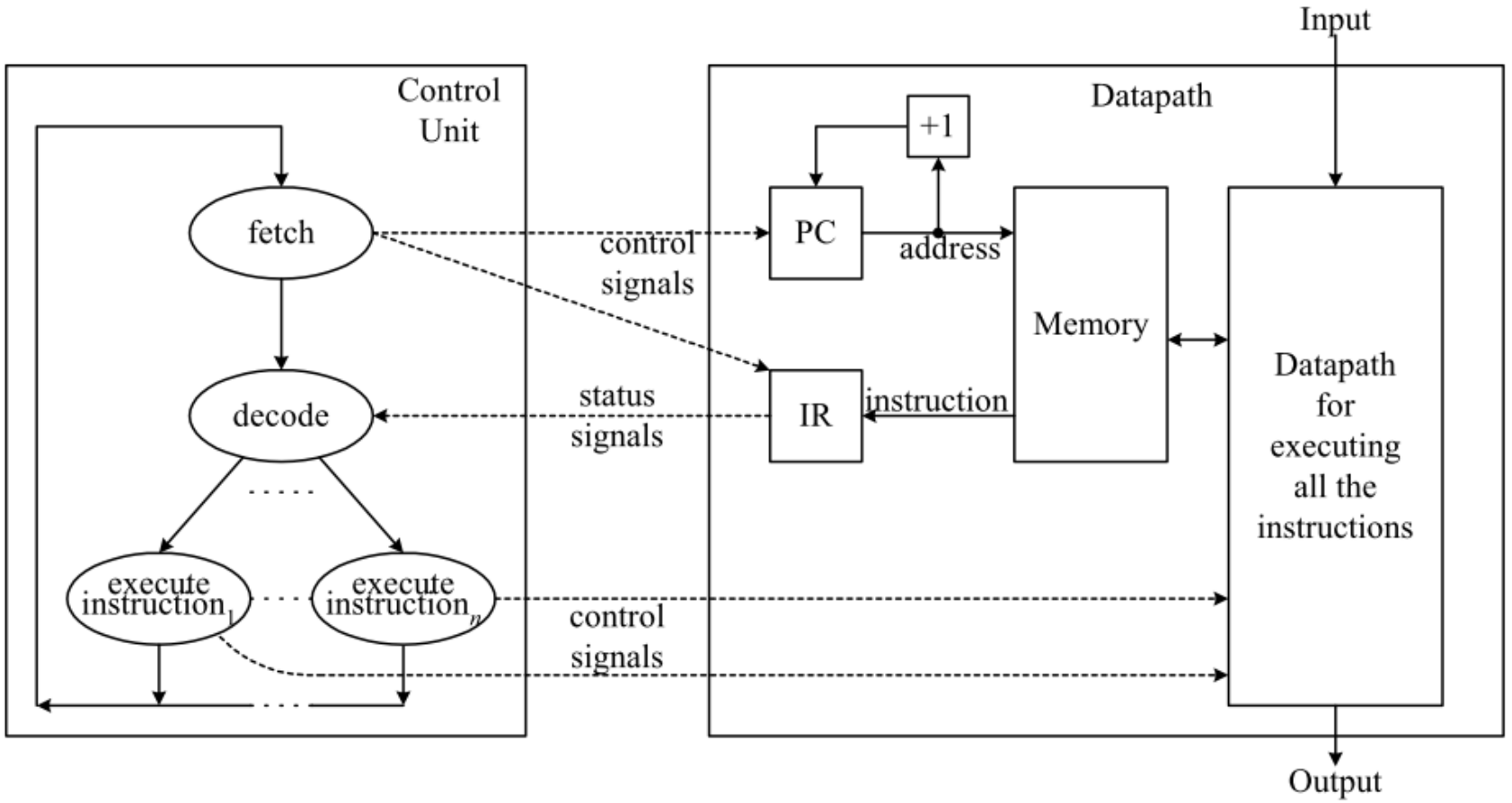



**Figure 12.1**. Overview of a general-purpose microprocessor.

Detalhes em OAC1 !!!

* Digital Logic and Microprocessor Design With VHDL by Enoch O. Hwang

# Atividade da Próxima Aula

- Atividade p Próxima Aula
  - Entregar no início da aula
  - Descrever em VHDL
    - uma memória RAM 1Kx16 usando memórias de 512x8