Aula 11 – Declaração de tipos, Vetores, Matrizes e Memórias

Tópicos da aula

- **Tipos**
- **Vetores e Matrizes**
- **Memória RAM**
- **Memória ROM**

Declaração de Tipo

```
TYPE temperatura IS (baixa, media, alta);
TYPE cores IS  ('R', 'G', 'B');
```

Declaração de tipo enumerado.

Declaração de Subtipo

```
TYPE    integer  IS RANGE -2147483648 TO 2147483647;
SUBTYPE natural  IS integer RANGE 0 TO integer'HIGH;
SUBTYPE positive IS integer RANGE 1 TO integer'HIGH;
```

Subtipos declarados no pacote padrão.

Vetores unidimensionais

```
Type vetor_ax  Is Array (0 To 7) Of Character;
Type vetor_ay  Is Array (Integer Range 0 To 7) Of Character;

Signal   ax : vetor_ax;
Signal   ay : vetor_ay;
Constant az : vetor_ay := (0 => 'e', 1 To 3 => 't', Others => 'a');

ax(0 To 7) <= "etttaaaa";
ay <= (0 => 'e', 1 To 3 => 't', Others => 'a');
```

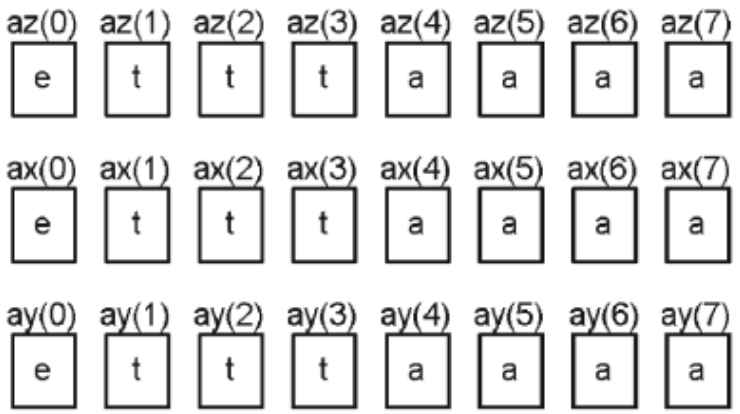
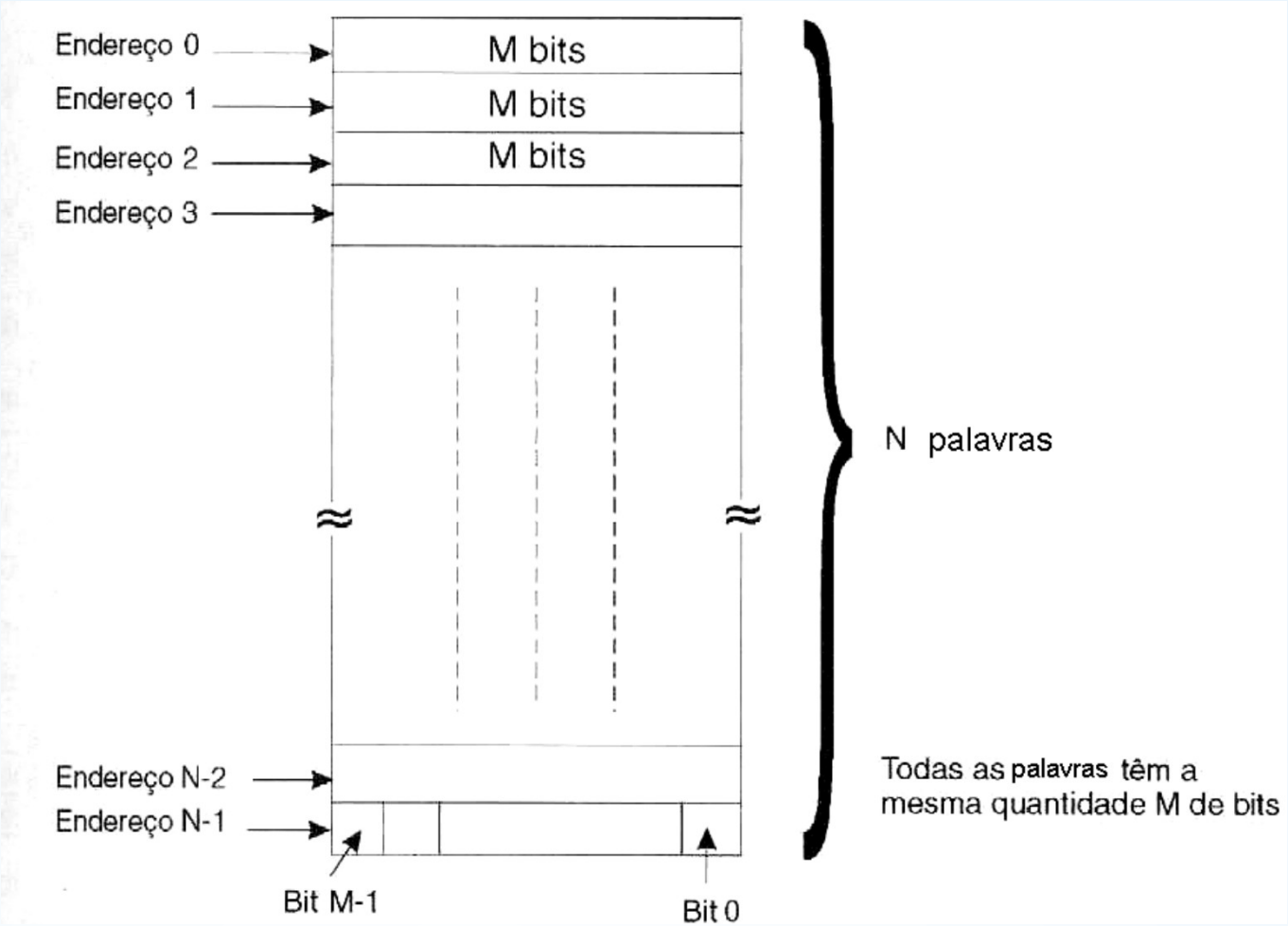


Ilustração de vetores unidimensionais contendo elementos tipo " CHARACTER ".

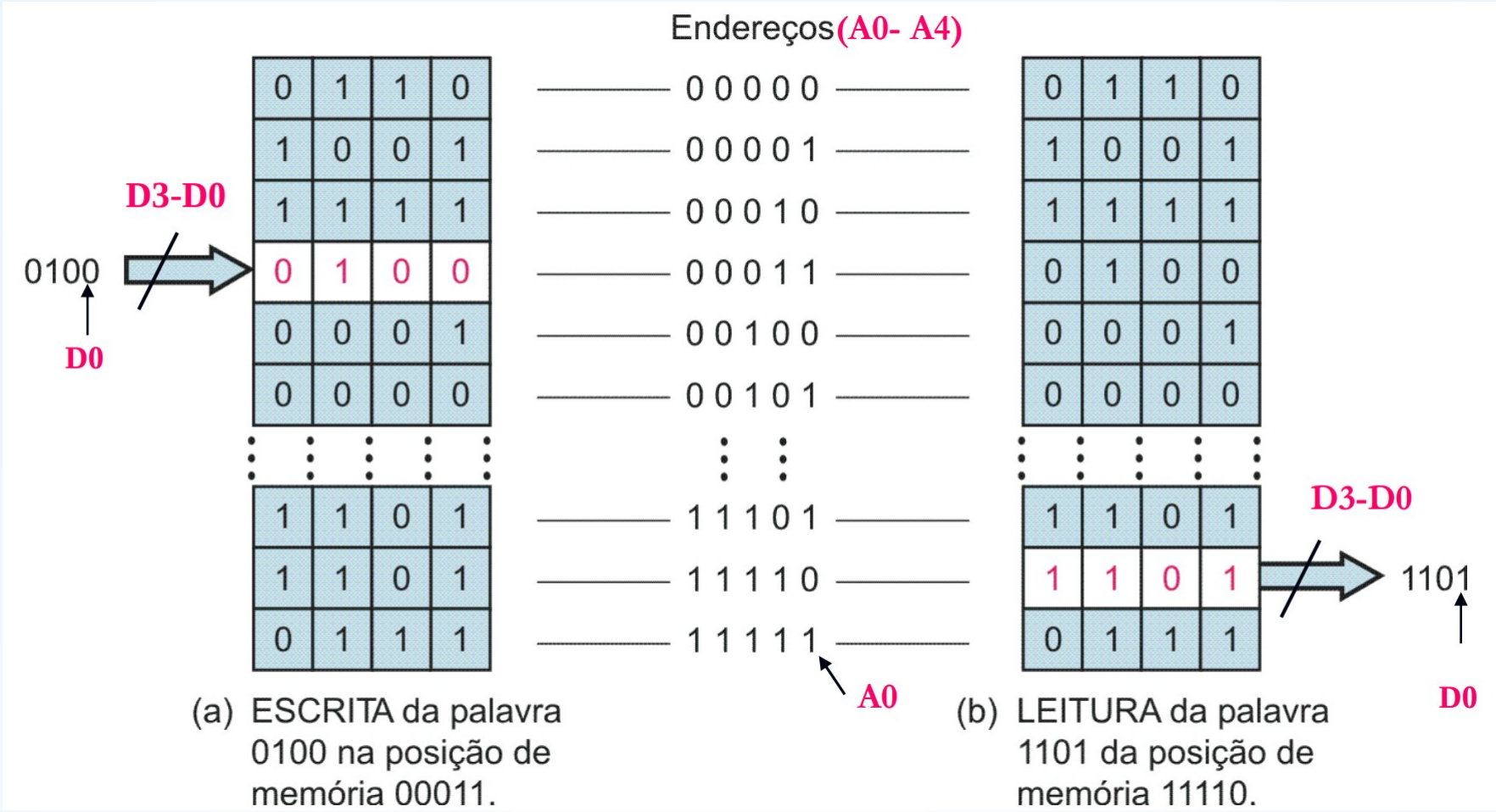
Vetores multidimensionais

```
 1  ENTITY teste_e1 IS
 2  END teste_e1;
 3
 4  ARCHITECTURE teste OF teste_e1 IS
 5    TYPE vetor_2d  IS ARRAY (0 TO 7) OF BIT_VECTOR(3 DOWNTO 0);
 6    TYPE vetor_3d  IS ARRAY (0 TO 2) OF vetor_2d;
 7
 8    SIGNAL   s_2d, t_2d: vetor_2d;
 9    CONSTANT c_2d: vetor_2d := (0 TO 2 =>('0','0','0','0'), OTHERS => ('1','0','1','1'));
10    SIGNAL   s_3d, t_3d: vetor_3d;
11
12  BEGIN
13    s_2d(7)(2) <= c_2d(7)(1);                       -- 1 elemento
14    s_2d(3) <= "1000";                              -- 1 indice
15    s_2d(4 TO 6) <= ("1010", OTHERS => "1011");     -- faixa de indices
16    s_2d(0 TO 2) <= c_2d(3 TO 5);                   -- faixa de indices
17    t_2d <= c_2d;                                   -- vetor completo
18
19    s_3d(2)(7)(3) <= c_2d(7)(1);                    -- 1 elemento
20    s_3d(0)(1)(2 DOWNTO 0) <= c_2d(3)(3 DOWNTO 1);  -- faixa
21    s_3d(1)(2 TO 3) <= c_2d(5 TO 6);                -- faixa
22    t_3d(2) <= c_2d;                                -- faixa
23    t_3d(0 TO 1) <= c_2d & c_2d;                    -- faixa
24  END teste;
```

Memórias – *N x M*



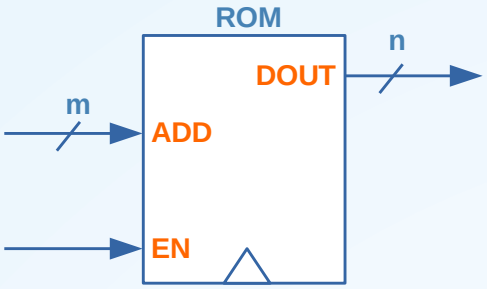
Ex.: memória de 32 *x* 4

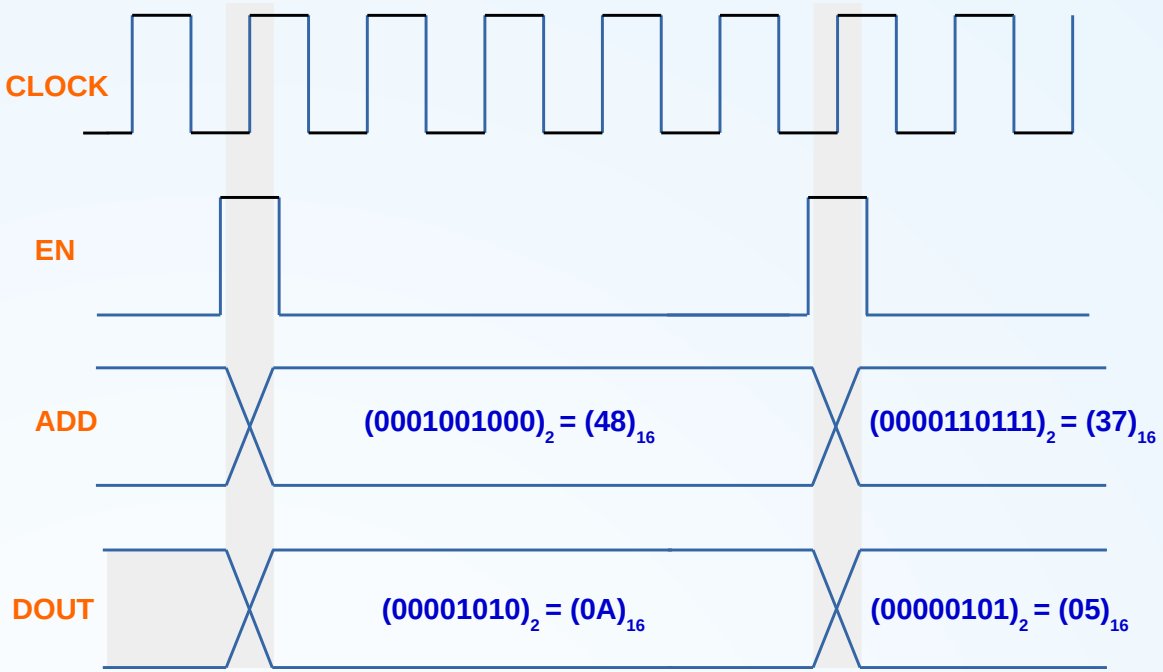


(a) ESCRITA da palavra 0100 na posição de memória 00011.

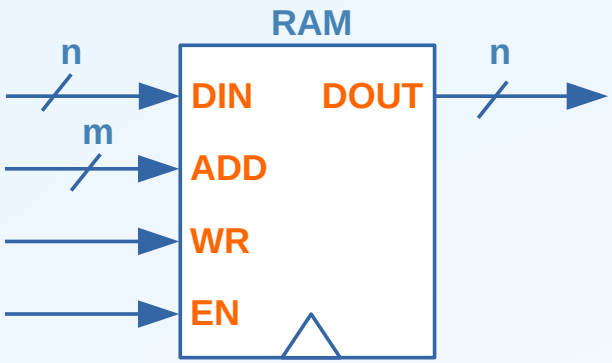
(b) LEITURA da palavra 1101 da posição de memória 11110.

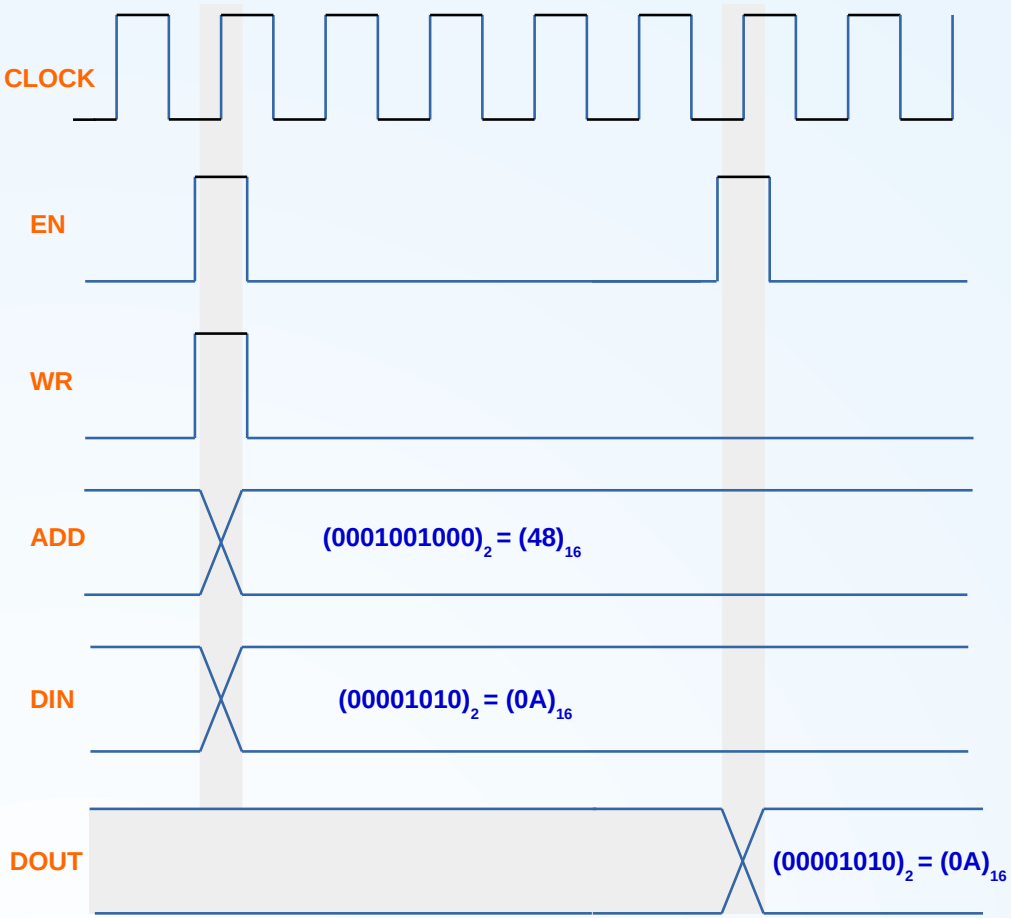
ROM (*Read Only Memory*)





RAM (*Random Access Memory*)





```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_arith.ALL;
4  use IEEE.STD_LOGIC_unsigned.ALL;
5
6  entity MEMORIA is
7      Generic(
8              p_DATA_WIDTH    : INTEGER := 16;         -- Número de bits dos dados.
9              p_ADD_WIDTH     : INTEGER := 6          -- Número de bits dos endereços.
10     );
11     Port (
12             i_CLK     : in  STD_LOGIC;
13             i_DATA    : in  STD_LOGIC_VECTOR ((p_DATA_WIDTH-1) downto 0);
14             i_WE      : in  STD_LOGIC;
15             i_ADDR    : in  STD_LOGIC_VECTOR ((p_ADD_WIDTH-1) downto 0);
16             i_ADDW    : in  STD_LOGIC_VECTOR ((p_ADD_WIDTH-1) downto 0);
17             o_DATA    : out STD_LOGIC_VECTOR ((p_DATA_WIDTH-1) downto 0)
18     );
19 end MEMORIA;
20
21 architecture Behavioral of MEMORIA is
22
23     type MEM_TYPE is array(i_ADDR'range) of std_logic_vector(i_DATA'range);
24     signal w_MEMORIA_RAM : MEM_TYPE;
25
26 begin
27
28         -- Process de escritura
29         process(i_CLK)  begin
30             if rising_edge(i_CLK)  then
31                 if (i_WE = '1')  then
32                     w_MEMORIA_RAM(conv_integer(i_ADDW)) <= i_DATA;
33                 end if;
34
35                 o_DATA <= w_MEMORIA_RAM(conv_integer(i_ADDR));
36             end if;
37         end process;
38
39 end Behavioral;
```

FIM AULA 11