



# Circuitos Digitais II - 6882

André Barbosa Verona  
Nardênio Almeida Martins

Universidade Estadual de Maringá  
Departamento de Informática

Bacharelado em Ciência da Computação

# Aula de Hoje

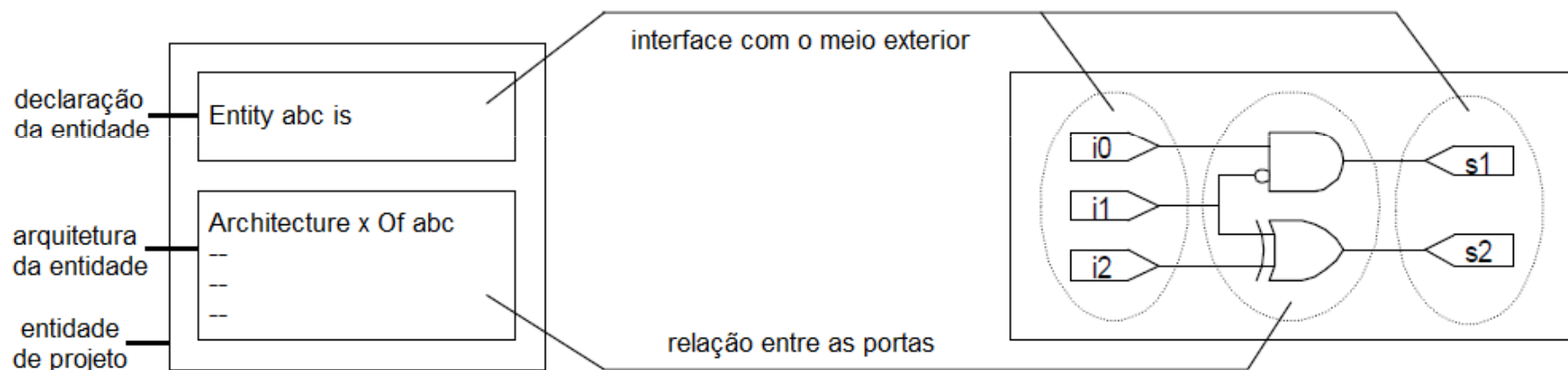
- **Revisão da aula anterior**
- **Tipos de Dados**

# Revisão

- **Características de Projeto**

# Características de Projeto

## Estrutura ou entidade de projeto



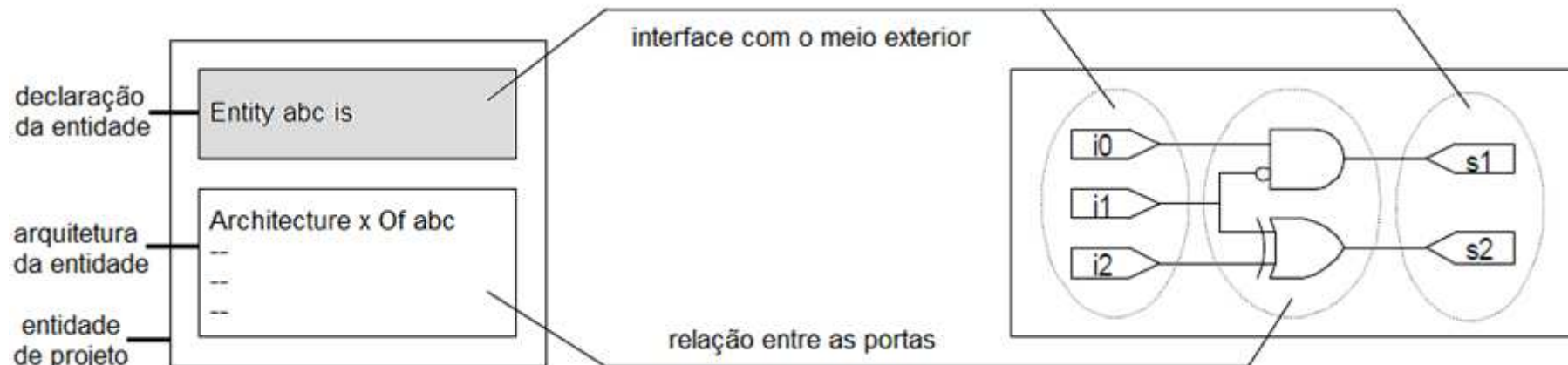
# Características de Projeto

## Estrutura ou entidade de projeto

- Pode representar desde uma simples porta lógica a um sistema completo
- Composta de duas partes:
  - Declaração da entidade → define portas de entrada e saída da descrição (equivalente ao símbolo de um bloco em captura esquemática)
  - Arquitetura da entidade → descreve as relações entre as portas (equivalente ao esquema contido no bloco em captura esquemática)

# Características de Projeto

## Declaração da entidade



```
ENTITY entidade_abc IS
```

```
    PORT (x0, x1      : IN      tipo_a;    -- entradas
          y0, y1      : OUT      tipo_b;    -- saídas
          y2          : BUFFER   tipo_c;    -- saída
          z0, z1      : INOUT    tipo_d);   -- entrada / saída
```

```
END entidade_abc;
```

# Características de Projeto

## Declaração da entidade

### Sintaxe:

```
ENTITY entidade_abc IS
    GENERIC (n      : INTEGER := 5);
    PORT      (x0, x1 : IN      tipo_a;      -- entradas
               y0, y1 : OUT     tipo_b;      -- saidas
               y2 : BUFFER   tipo_c;      -- saida
               z0, z1 : INOUT   tipo_d;      -- entrada/saida
END entidade_abc;
```

# Características de Projeto

## Declaração da entidade

**ENTITY:** inicia a declaração

**PORT:** define modo e tipo das portas

modo **IN** : entrada

modo **OUT** : saída

modo **BUFFER** : saída - pode ser referenciada  
internamente

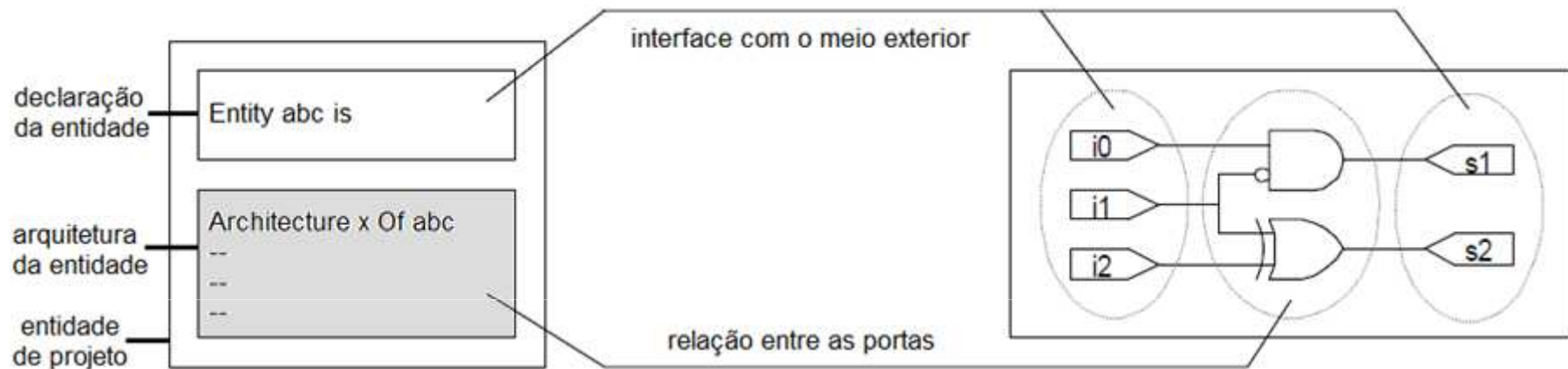
modo **INOUT** : bidirecional

**END:** termina a declaração



# Características de Projeto

## Arquitetura da entidade



```
ARCHITECTURE estilo_abc OF entidade_abc IS
    -- declaracoes de sinais e constantes
    -- declaracoes de componentes referenciados
    -- descricao de sub-programas locais
    -- definicao de novos tipos de dados locais
    --
BEGIN
    --
    -- declaracoes concorrentes
    --
END estilo_abc;
```

# Características de Projeto

## Arquitetura da entidade

**Sintaxe:** **ARCHITECTURE** nome\_identificador **OF** entidade\_abc **IS**

--

-- regio de declaracoes:

-- declaracoes de sinais e constantes

-- declaracoes de componentes referenciados

-- declaracao e corpo de subprogramas

-- definicao de novos tipos de dados locais

--

**BEGIN**

--

-- comandos concorrentes

--

**END;**

# Características de Projeto

## Arquitetura da entidade

**ARCHITECTURE:** inicia a declaração

- Linhas que seguem podem conter:
  - declaração de sinais e constantes
  - declaração de componentes referenciados
  - descrição de subprogramas locais
  - definição de novos tipos

**BEGIN:** inicia a descrição

**END:** termina a descrição

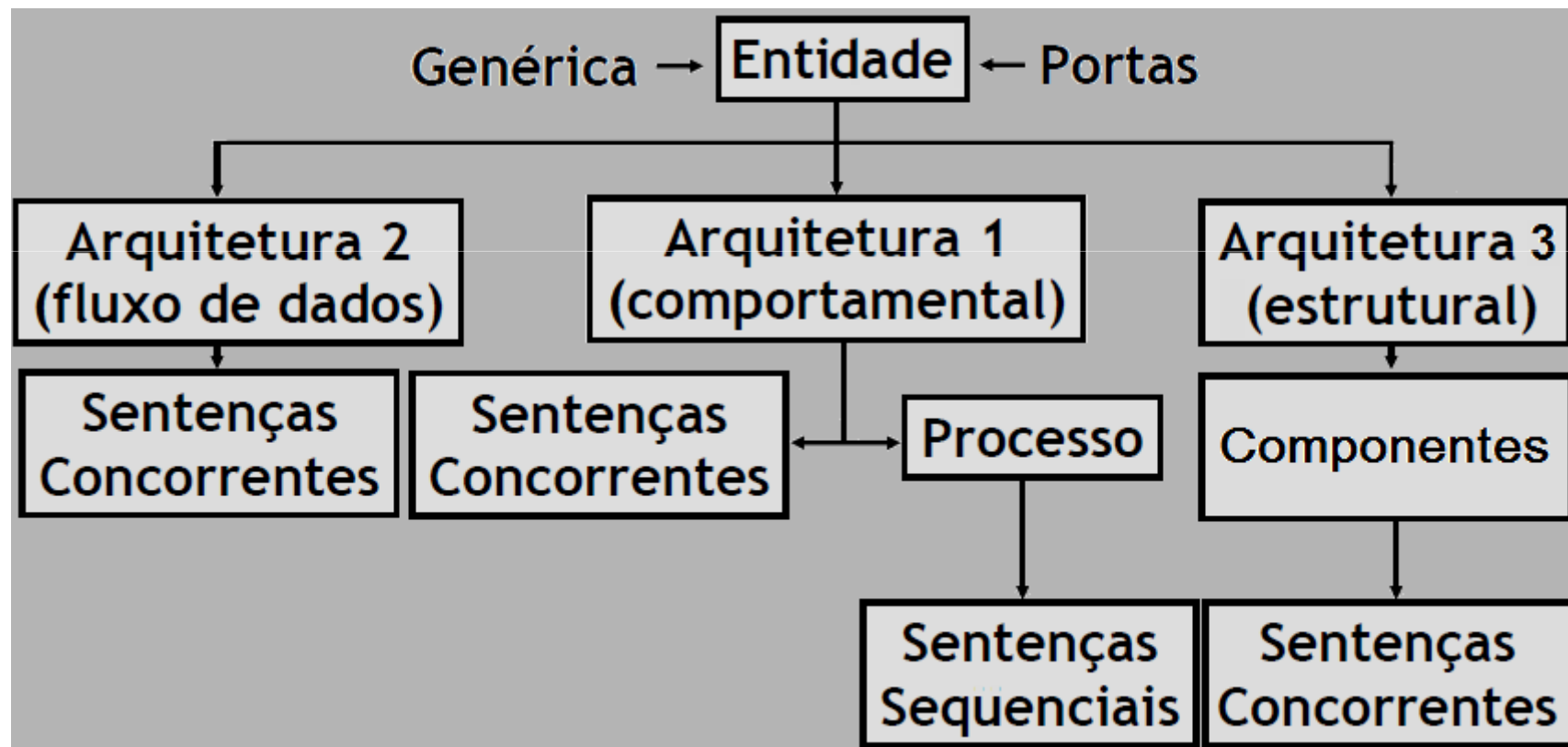
# Características de Projeto

## Estrutura básica de um código em VHDL

<b>LIBRARY IEEE;</b> <b>USE IEEE.STD_LOGIC_1164.all;</b> <b>USE IEEE.STD_LOGIC_UNSIGNED.all;</b>	LIBRARY (PACOTES)
<b>ENTITY</b> exemplo <b>IS</b> <b>PORT</b> ( <descrição dos pinos de I/O> ); <b>END</b> exemplo;	ENTITY (PINOS DE I/O)
<b>ARCHITECTURE</b> teste <b>OF</b> exemplo <b>IS</b> <b>BEGIN</b> ... <b>END</b> teste;	ARCHITECTURE (ARQUITETURA)

# Características de Projeto

## Estrutura básica de um código em VHDL



# Características de Projeto

## Descrição ou modelagem comportamental

- Declaração de um processo

```
. . .  
ARCHITECTURE nome_da_arquitetura OF nome_da_entidade IS  
BEGIN  
    PROCESS (lista_de_sensibilidade)  
    BEGIN  
        . . .  
        comandos;  
        . . .  
    END PROCESS;  
END nome_da_arquitetura;
```

# Características de Projeto

## Descrição ou modelagem estrutural usando componente

- Declaração e instanciação de um componente

. . . .

**ARCHITECTURE** nome\_da\_arquitetura **OF** nome\_da\_entidade **IS**

**COMPONENT** nome\_do\_componente -- declaracao do componente

**PORT** (lista\_de\_porta\_de\_interface);

**END** nome\_do\_componente;

**SIGNAL** lista\_de\_sinal : tipo\_de\_dado;

**BEGIN**

nome\_da\_instanciacao : nome\_componente **PORT MAP**

(lista\_de\_associacao\_de\_porta); -- instanciacao do componente

**END** nome\_da\_arquitetura;

- Nota: A declaração do componente é similar à declaração de entidade.

# Aula de Hoje

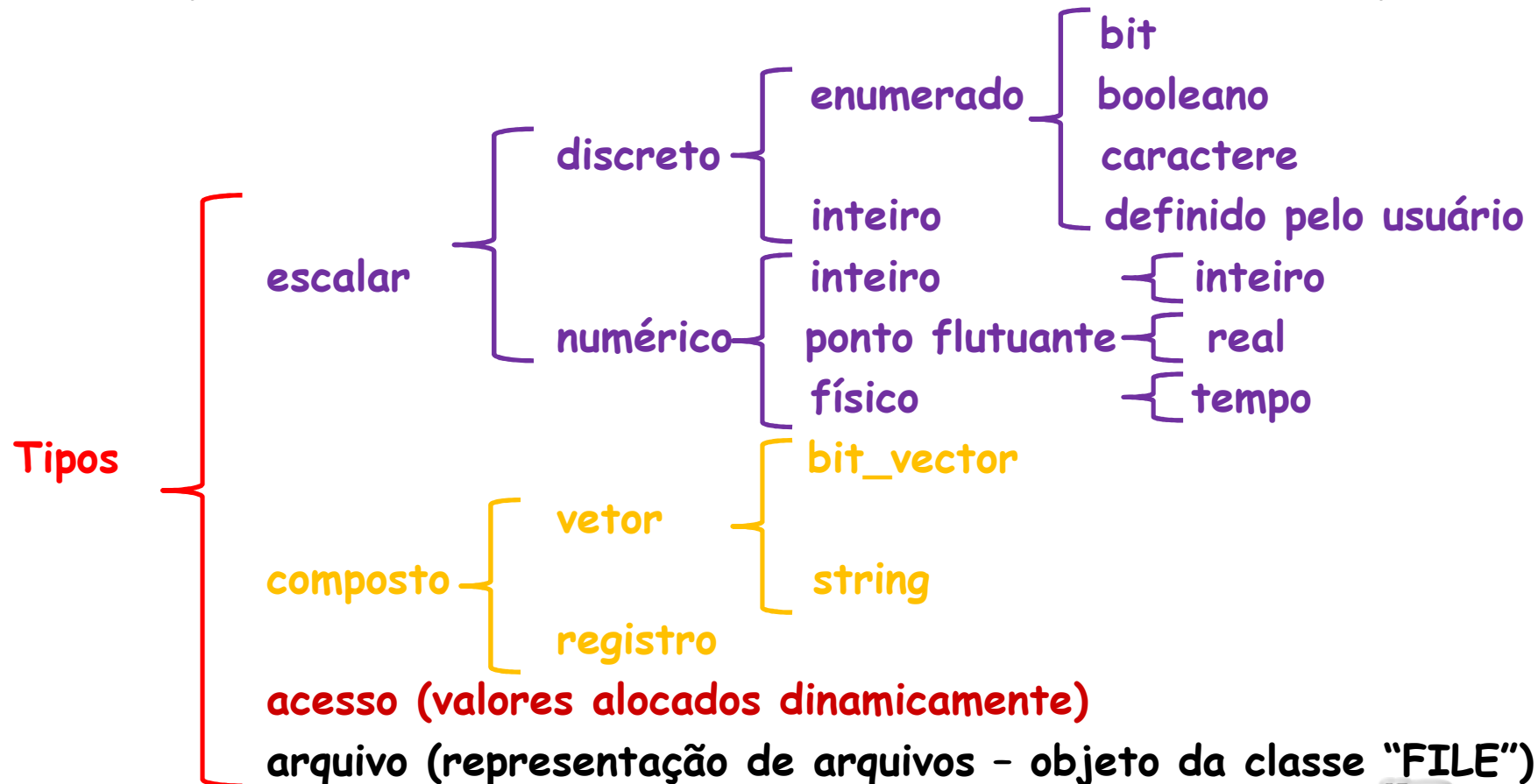
- Tipos de Dados



# VHDL - Tipos de Dados

## Tipos em VHDL

- Representação do conjunto de valores que um objeto pode armazenar e do conjunto de operações que podem ser realizadas com esse objeto.



# VHDL - Tipos de Dados

## Tipos e subtipos\* escalares predefinidos no pacote padrão

TIPO	VALOR	EXEMPLO
BIT	um, zero	'1', '0'
BOOLEAN	falso, verdadeiro	true, false
CHARACTER	caracteres	'a',..., 'z', 'A',..., 'Z', '0',..., '9', '?', '('
INTEGER	$-2^{31}-1 \leq x \leq 2^{31}-1$	123, 8#173#, 16#7B#, 2#11_11_011#
NATURAL*	$0 \leq x \leq 2^{31}-1$	123, 8#173#, 16#7B#, 2#11_11_011#
POSITIVE*	$1 \leq x \leq 2^{31}-1$	123, 8#173#, 16#7B#, 2#11_11_011#
REAL	$-1.0E+38 \leq x \leq 1.0E+38$	1.23, 1.23E+2, 16#7B#E+1
TIME	fs, ps=10 <sup>3</sup> fs, ns=10 <sup>3</sup> ps, us=10 <sup>3</sup> ns, ms=10 <sup>3</sup> us, sec=10 <sup>3</sup> ms, min=60sec, hr=60min	1us, 100ps, 1fs

- Nota: Objetos declarados com tipos diferentes não permitem a transferência de valores entre eles, a não ser que se faça uma operação de conversão.

# VHDL - Tipos de Dados

## BIT

TIPO	VALOR	EXEMPLO
BIT	um, zero	'1', '0'

- Implementação do tipo BIT predefinido em VHDL

```
TYPE bit IS ('0', '1'); -- declaração do tipo
```

- O uso de aspas simples ao redor dos valores é devido à linguagem definir o tipo BIT com a enumeração dos caracteres '0' e '1'.
- Utilizado em operações binárias

# VHDL - Tipos de Dados

## BOOLEAN (Booleano)

- Representação pela enumeração dos valores true e false.

### TIPO

### VALOR

### EXEMPLO

BOOLEAN

falso, verdadeiro

true, false

- Implementação do tipo BOOLEAN predefinido em VHDL

```
TYPE boolean IS (true, false); -- declaração do tipo
```

- De forma padrão, um sinal tipo booleano sempre se inicia com valor false. Caso necessite alterar esse valor, deve-se explicitar na declaração do sinal ou da variável.
- Sinais, constantes e variáveis do tipo booleano podem ser utilizados em operações com operadores lógicos e relacionais, onde o resultado será sempre do tipo booleano.

# VHDL - Tipos de Dados

## CHARACTER (Caractere)

- Tipo de dado enumerado que permite a associação de um dígito alfanumérico a constantes, variáveis e sinais.
- Constituição: 128 caracteres do padrão ASCII; 96 caracteres do padrão Latin 1; 33 caracteres de controle; 221 caracteres comuns.
- Os caracteres comuns são imprimíveis e devem sempre estar entre aspas simples para serem reconhecidos pela linguagem.

TIPO	VALOR	EXEMPLO
CHARACTER	caracteres	'a',..., 'z', 'A',..., 'Z', '0',..., '9', '?', '('

- Implementação do tipo CHARACTER predefinido em VHDL

```
TYPE character IS ('a', 'b', 'c', 'd'); -- declaração do tipo
```

# VHDL - Tipos de Dados

## Tipos definidos pelo usuário

- Criação de novos tipos definidos pelo usuário pode facilitar a leitura de um código, atribuindo nomes específicos para determinadas condições.
- Declaração de um tipo, que podem ser arrays ou conjuntos enumerados, é dada por meio da palavra reservada **TYPE**, seguida do nome escolhido para designar novo tipo e a sua especificação.
- Exemplos de declaração de tipos definidos pelo usuário:

```
TYPE estado IS (parado, inicio, caso_1, caso_2, caso_3);
```

```
TYPE Impares IS (1, 3, 5, 7, 9);
```

```
TYPE Vogais_Minusculas IS ('a', 'e', 'i', 'o', 'u');
```

```
TYPE BitVector4 IS ARRAY (3 DOWNTO 0) OF BIT;
```

```
TYPE Matriz IS ARRAY (1 TO 10, 1 TO 10) OF INTEGER;
```

# VHDL - Tipos de Dados

## Tipos definidos pelo usuário

- Pela utilização da palavra reservada **SUBTYPE** permite a redução da faixa de dados de um tipo predefinido por meio da criação de um subtipo.

➤ Exemplos:

```
SUBTYPE Maiusculas IS CHARACTER RANGE 'A' TO 'Z';  
SUBTYPE NatMenor20 IS INTEGER RANGE 0 TO 19;  
SUBTYPE Byte IS STD_LOGIC_VECTOR (7 DOWNTO 0);
```

- Nota 01: A palavra reservada **RANGE** especifica o tratamento de uma faixa reduzida de dados, indicando o seu alcance.
- Nota 02: Byte não é uma palavra reservada.

# VHDL - Tipos de Dados

## Tipos definidos pelo usuário

- Criação de registros (**RECORD**) cujas estruturas possuem vários elementos com tipos de dados que podem ser diferentes.
- Definição de registros:

```
TYPE codes IS ("add", "sub", "div", "lda", "sta", "beq", "bne", "jump");  
TYPE Instrucao2op IS  
    RECORD  
        op_code : codes;  
        operando1, operando2 : INTEGER;  
    END RECORD;
```



# VHDL - Tipos de Dados

## Tipos definidos pelo usuário

- Definição de sinais de registro:

```
SIGNAL instruction : Instrucao2op;  
SIGNAL soma: codes := "add"
```

- Atribuição aos campos de um registro individualmente:

```
instruction.operando1 <= A;  
instruction.operando2 <= B;  
instruction.op_code <= soma;
```

- Atribuição de valores a todos os campos de um registro:

```
instruction <= (A, B, soma);
```

# VHDL - Tipos de Dados

## INTEGER (Inteiro)

- Representação de números inteiros pertencentes à faixa de valores de -2.147.483.647 até 2.147.483.647, utilizando qualquer base entre 2 e 16.

TIPO	VALOR	EXEMPLO
INTEGER	$-2^{31}-1 \leq x \leq 2^{31}-1$	123, 8#173#, 16#7B#, 2#11_11_011#

- Implementação do tipo INTEGER em VHDL

```
TYPE dia_da_semana IS RANGE 1 TO 31;           -- declaração do tipo
```

```
TYPE valor_contagem IS RANGE 10 DOWNT0 0;      -- declaração do tipo
```

- Exemplo: Valor 205

```
Int0 <= 205;           -- associação de valores na base decimal
Int1 <= 2#1100_1101#;  -- associação de valores em binário
Int2 <= 8#315#;        -- associação do valor 205 na base octal
Int3 <= 16#CD#;         -- associação de valores em hexadecimal
```

- Nota: Associação de valores sempre deve seguir o formato **base#valor#** caso seja utilizada na base diferente da decimal.

# VHDL - Tipos de Dados

## INTEGER (Inteiro)

- Representação de uma variável, sinal ou constante do tipo INTEGER em hardware necessita de 32 bits.
- Caso não seja necessária a utilização de todos esses bits, é possível otimizar a síntese de um projeto que utiliza tal tipo de dado, reduzindo assim a sua faixa de valores.
- A linguagem possibilita a definição de subtipos e um subtipo delimita uma faixa de valores dentro da gama de valores do tipo.
- Um subtipo é um subconjunto dos valores de um tipo, logo, é possível a troca de valores entre objetos assim declarados.

# VHDL - Tipos de Dados

## INTEGER (Inteiro)

- Uso de dois subtipos do tipo INTEGER: NATURAL e POSITIVE.

SUBTIPO	VALOR	EXEMPLO
NATURAL	$0 \leq x \leq 2^{31}-1$	123, 8#173#, 16#7B#, 2#11_11_011#
POSITIVE	$1 \leq x \leq 2^{31}-1$	123, 8#173#, 16#7B#, 2#11_11_011#

- Restrição da faixa de valores de um sinal do tipo inteiro em VHDL

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

```
SIGNAL contador1 : natural RANGE 0 TO 15;  
SIGNAL contador2 : positive RANGE 15 DOWNT0 1;
```

# VHDL - Tipos de Dados

## REAL

- Representação de valores de números de ponto flutuante com faixa iniciando em  $-1.0E+38$  e chegando até  $1.0E+38$ .
- Normalmente, este tipo não é suportado pelas ferramentas de síntese, uma vez que são necessárias estruturas de hardware muito complexas para tratar de suas operações.

TIPO	VALOR	EXEMPLO
REAL	$-1.0E+38 \leq x \leq 1.0E+38$	1.23, 1.23E+2, 16#7B#E+1

- Implementação do tipo REAL em VHDL

```
TYPE valor_leitura IS RANGE -5.0 TO 5.0;           -- declaração do tipo
```

```
TYPE porcentagem IS RANGE 100.0 DOWNT0 0.0; -- declaração do tipo
```

# VHDL - Tipos de Dados

## REAL

- Associação de valores em sinais do tipo REAL:

```
var <= 1;      -- associação inválida  
var <= 1.0;    -- associação válida  
var <= 1.0E+6; -- associação válida
```

- Nota: Distinção entre um valor de um tipo REAL e um valor de um tipo INTEGER dá-se pela presença do ponto decimal.

# VHDL - Tipos de Dados

## Tipos físicos

- Representação de grandezas ou quantidades físicas, tais como tempo, massa, velocidade, comprimento, corrente ou tensão.
- Esses tipos não são sintetizáveis, sendo ignorados pelas ferramentas durante a síntese. No caso do tipo TIME, os intervalos de tempo são ignorados.

TIPO	VALOR	EXEMPLO
TIME	fs, ps=10 <sup>3</sup> fs, ns=10 <sup>3</sup> ps, us=10 <sup>3</sup> ns, ms=10 <sup>3</sup> us, sec=10 <sup>3</sup> ms, min=60sec, hr=60min	1us, 100ps, 1fs

- Atribuição de valores a sinais do tipo TIME:

```
Tempo <= 10 ns;  
Tempo <= 100 fs;
```

- O TIME (tempo) é o único tipo físico definido na biblioteca padrão da VHDL.

# VHDL - Tipos de Dados

## Tipos físicos

- Declaração do tipo TIME:

```
TYPE time IS RANGE IMPLEMENTATION_DEFINED
```

**UNITS**

fs;

ps = 1000 fs;

ns = 1000 ps;

us = 1000 ns;

ms = 1000 us;

sec = 1000 ms;

min = 60 sec;

hr = 60 min;

**END UNITS;**

- Variáveis e sinais do tipo TIME são muito úteis na representação de atrasos de propagação, simulação de frequências de clock e análise do tempo de simulação.



# VHDL - Tipos de Dados

## Tipos físicos

- A atribuição das respectivas unidades físicas ocorre com o uso da palavra reservada **UNITS**, sendo estabelecidas uma unidade base e sucessivas unidades múltiplas desta.
- A definição do tipo físico é dependente da implementação e o alcance pode variar de acordo com o simulador utilizado.
- É garantida, porém, uma faixa mínima entre -2.147.483.647 e 2.147.483.647.

# VHDL - Tipos de Dados

## Tipos físicos

- Exemplo de declaração do tipo físico RESISTENCIA:

```
TYPE resistencia IS RANGE 0 TO 1E9
    UNITS
        ohm;
        Kohm = 1000 ohm;
        Mohm = 1000 Kohm;
        Gohm = 1000 Mohm;
    END UNITS;
```

# VHDL - Tipos de Dados

## Tipos Compostos - Vetor

- Dois tipos definidos ARRAY (vetor)
  - BIT\_VECTOR contendo elementos do tipo BIT;
  - STRING contendo elementos do tipo CHARACTER.
- Tipos ARRAY predefinidos no pacote padrão

TIPO	VALOR	EXEMPLO
BIT_VECTOR	Array de bits	"1010", B"10_10", O"12", X"C"
STRING	Array de caracteres	"texto", " "incluindo_aspas" "

# VHDL - Tipos de Dados

## BIT\_VECTOR (Vetor de bits)

- Uma única variável ou um único sinal deste tipo permite a manipulação de um conjunto de bits (vetor de bits) facilitando assim a modelagem de circuitos mais complexos.

### TIPO

### VALOR

### EXEMPLO

BIT\_VECTOR

Array de bits

"1010", B"10\_10", O"12", X"C"

- Implementação do tipo BIT\_VECTOR, com limites em aberto, em VHDL:

```
TYPE BIT_VECTOR IS ARRAY (INTEGER RANGE <>) OF bit; -- declaração do tipo
```

- Por ser este tipo de dado um array com limites em aberto (<>), o usuário deve especificar o tamanho dos vetores, indicando a faixa de dados que eles alcançam.

# VHDL - Tipos de Dados

## BIT\_VECTOR (Vetor de bits)

- Declaração de entidade utilizando o tipo BIT\_VECTOR com o bit mais significativo à esquerda:

```
ENTITY contador_assinc IS
    PORT (clk : IN BIT;
          saida : OUT BIT_VECTOR (2 DOWNTO 0);
          clr: OUT BIT);
END contador_assinc;
```

- Declaração do tipo BIT\_VECTOR com o bit mais significativo à direita:

```
saida : OUT BIT_VECTOR (0 TO 2);
```

# VHDL - Tipos de Dados

## BIT\_VECTOR (Vetor de bits)

- Atribuição de valores:

- Cada bit do vetor separadamente

- ✓ Exemplo: 'saida' = "110"

```
saida(2) <= '1';  
saida(1) <= '1';  
saida(0) <= '0';
```

- Associação de valores ao objeto de uma só vez

- ✓ Exemplo: 'saida' = "010"

```
saida <= "010";  
saida <= ('0','1','0');
```

# VHDL - Tipos de Dados

## BIT\_VECTOR (Vetor de bits)

- Atribuição de valores:

- Troca de valores de apenas alguns bits

- ✓ Exemplo: 'saida' = "010"

```
saida <= (2 => '1', 0 => '1'); -- 'saida' = "111"
```

- Utilização da palavra reservada OTHERS que associa determinado valor a vários dados

- ✓ Exemplo: 'saida' = "101"

```
saida <= (1 => '0', OTHERS => '1'); -- 'saida' = "101"
```

# VHDL - Tipos de Dados

## BIT\_VECTOR (Vetor de bits)

- Inicialização de valores

- Utilização do caractere underscore (\_) entre seus valores, auxiliando o usuário na visualização do conjunto de dados

```
SIGNAL a: BIT_VECTOR (0 TO 7) := "1011_0011";
```

```
SIGNAL b: BIT_VECTOR (7 DOWNTO 0) := "1011_0011";
```

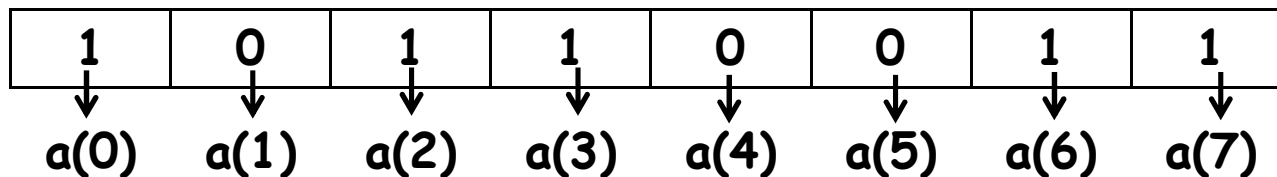


# VHDL - Tipos de Dados

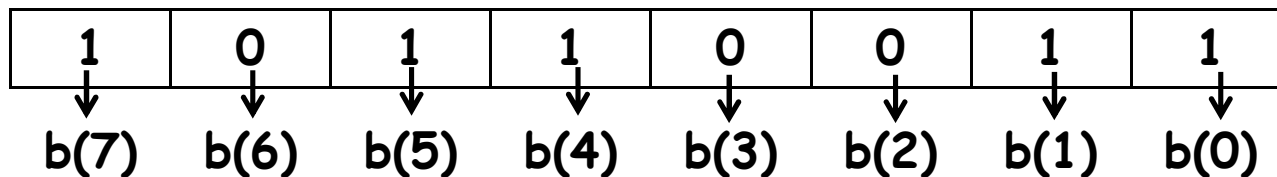
## BIT\_VECTOR - Declarações de vetores com definição de limites

- Na declaração de um objeto, a especificação do número de elementos contidos no vetor é dada pelas palavras reservadas **DOWNTO** e **TO**.

➤ **SIGNAL a: BIT\_VECTOR (0 TO 7) := "10110011";**



➤ **SIGNAL b: BIT\_VECTOR (7 DOWNTO 0) := "10110011";**



# VHDL - Tipos de Dados

## BIT\_VECTOR - Nomes indexados, partes de vetores e agregado

- Um vetor unidimensional pode ter os seus elementos referenciados por meio de:
  - nomes indexados → a(4), b(4), b(3)
  - partes de vetores → parcela crescente ou decrescente do vetor utilizando as palavras reservadas **DOWNTO** e **TO**.
- NOTA: Não existe a necessidade de inicializar um elemento do tipo BIT\_VECTOR a partir do índice 0 (zero).

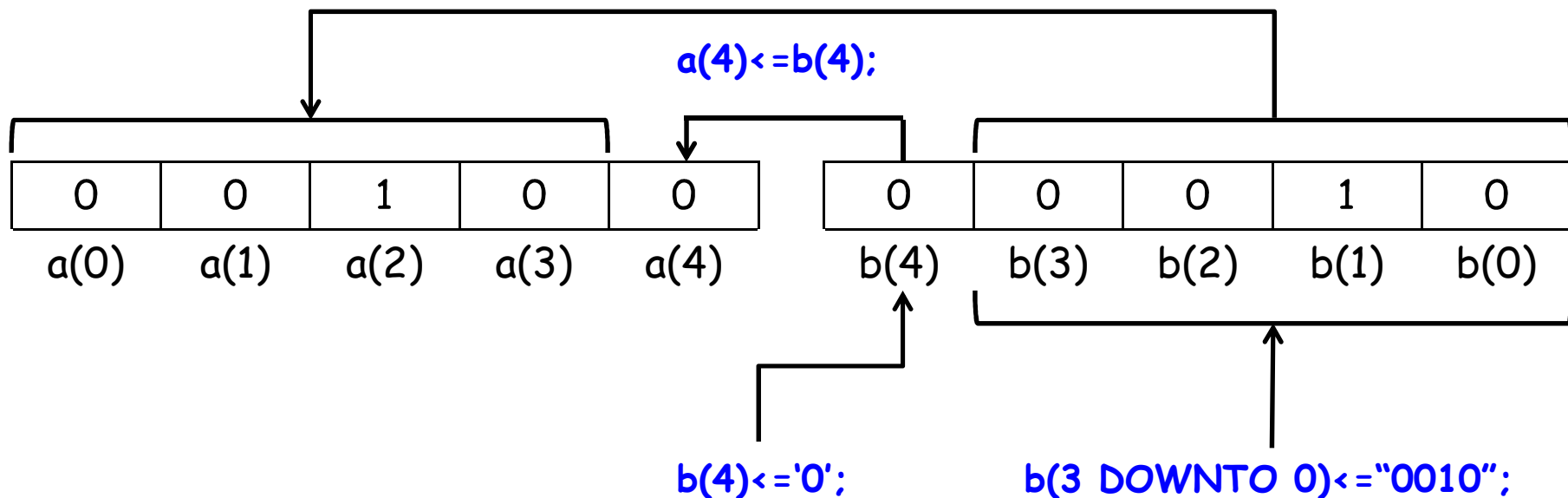
# VHDL - Tipos de Dados

## BIT\_VECTOR - Atribuições de valores com vetores unidimensionais

- Exemplo:

- **SIGNAL a, c : BIT\_VECTOR (0 TO 4);**
- **SIGNAL b, d : BIT\_VECTOR (4 DOWNTO 0);**

**a(0 TO 3) <= b(3 DOWNTO 0);**



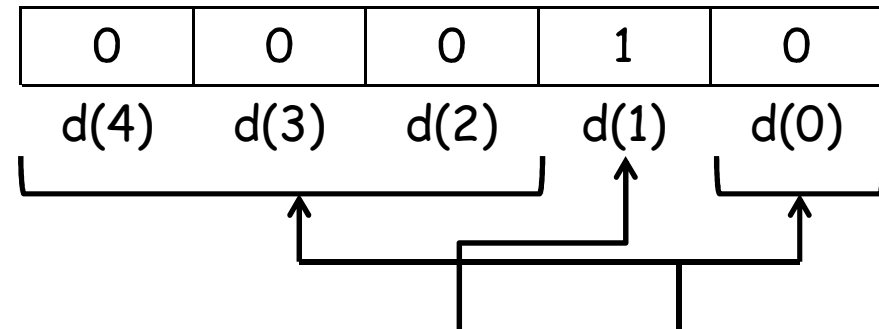
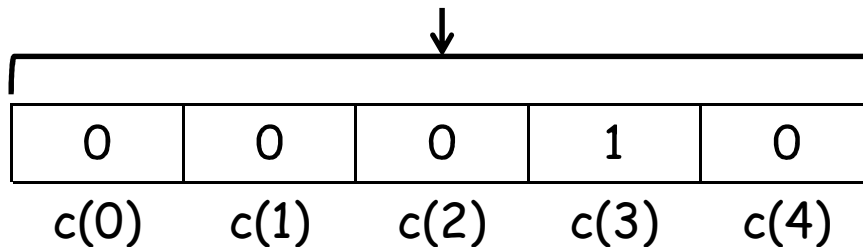
# VHDL - Tipos de Dados

## BIT\_VECTOR - Atribuições de valores com vetores unidimensionais (Cont.)

- Exemplo:

- **SIGNAL a, c : BIT\_VECTOR (0 TO 4);**
- **SIGNAL b, d : BIT\_VECTOR (4 DOWNTO 0);**

**c<=('0','0','0','1','0');**



**d<=(1=>'1', Others=>'0');**

- A palavra reservada **OTHERS** pode ser empregada para identificar todos os elementos não-especificados e deve ser sempre a última associação na lista de associações.

# VHDL - Tipos de Dados

## BIT\_VECTOR - Atribuições de valores em vetores: índice, parte e agregado

- Atribuição de valor a um objeto do tipo vetor, a expressão que contém o valor pode ser um valor, um nome indexado, uma parte de um vetor ou um agregado.

- Exemplo:

a(4)	<=	b(4);	-- 1 elemento<=1 elemento
a(0 TO 3)	<=	b(3 DOWNT0 0);	-- parte do vetor<=parte do vetor
b(4)	<=	'0';	-- 1 elemento<=valor
b(3 DOWNT0 0)	<=	"0010";	-- parte do vetor<=valor
c	<=	('0','0','0','1','0');	-- valor 00010 agregado notação posicional
d	<=	(1=>'1', OTHERS=>'0');	-- valor 00010 agregado associação por nomes

# VHDL - Tipos de Dados

## STRING (Vetor de caracteres)

- Representação de sequências de caracteres, em que os valores em constantes, variáveis e sinais devem ser apresentados entre aspas duplas.

TIPO	VALOR	EXEMPLO
------	-------	---------

STRING	Array de caracteres	"texto", "abc", "001", " "incluindo_aspas" "
--------	---------------------	--

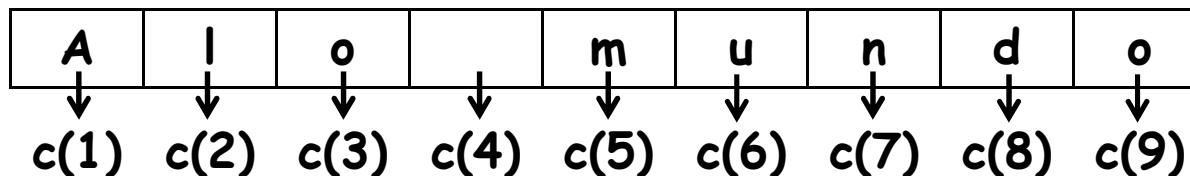
- Nota: A utilização do operador de concatenação (&) permite a união de diferentes vetores de caracteres (STRINGS).

# VHDL - Tipos de Dados

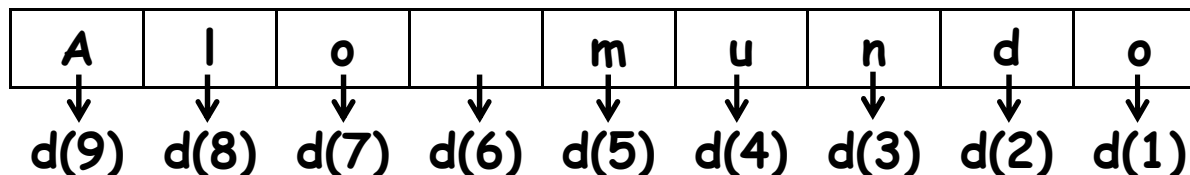
## STRING - Declarações de vetores com definição de limites

- Na declaração de um objeto, a especificação do número de elementos contidos no vetor é dada pelas palavras reservadas **DOWNTO** e **TO**.

➤ **CONSTANT** c: **STRING** (1 **TO** 9) := "Alo mundo";



➤ **VARIABLE** d: **STRING** (9 **DOWNTO** 1) := "Alo mundo";



- NOTA: Não existe a necessidade de inicializar um elemento do tipo **STRING** a partir do índice 0 (zero).

# VHDL - Tipos de Dados

## STD\_ULOGIC (Standard Unresolved Logic)

- Tipo criado pela Norma IEEE 1164 para representar outros valores diferentes de '0' e '1', como na simulação de alta impedância ou quando o sinal é desconhecido.
- Tipo não resolvido podendo ser utilizado somente no caso de uma única fonte.

### TIPO

### VALOR

### EXEMPLO

STD\_ULOGIC      um, zero e outros valores      '1', '0', 'X', 'Z', 'W', 'L', 'H', 'U', '\_'

- Declaração da biblioteca IEEE e do pacote IEEE 1164 no projeto.

```
LIBRARY IEEE;  
USE IEEE.STD_LOGIC_1164.ALL;
```

- Implementação do tipo STD\_ULOGIC predefinido em VHDL

```
TYPE std_ulogic IS ('1', '0', 'X', 'Z', 'W', 'L', 'H', 'U', '_'); -- declaração do tipo
```



# VHDL - Tipos de Dados

## STD\_ULOGIC (Standard Unresolved Logic)

- Possíveis valores para o tipo STD\_ULOGIC

### VALOR

### DESCRIÇÃO

'U'	Objeto ainda não foi inicializado ou inicialização desconhecida
'X'	Desconhecido forte ou impossível determinar o valor/resultado
'0'	0 forte ou estado lógico 0
'1'	1 forte ou estado lógico 1
'Z'	Alta Impedância
'W'	Desconhecido fraco ou impossível determinar se o valor deve ser 0 ou 1
'L'	0 fraco ou impossível determinar o valor mas provavelmente deve ser 0
'H'	1 fraco ou impossível determinar o valor mas provavelmente deve ser 1
'-'	Não importa ou não interessa o valor

# VHDL - Tipos de Dados

## STD\_ULOGIC (Standard Unresolved Logic)

- Exemplos:
  - Um buffer tri-state além de retornar os valores '0' e '1', pode retornar um valor de alta impedância ('Z') na saída.
  - A definição de nível lógico '1' (1 forte) como uma tensão de 5 V. Um '1' fraco ('H') poderia ser uma tensão alta, mas não perfeita, como um valor de 4 V.
- Nota: Tipo STD\_ULOGIC pode utilizar todas as operações lógicas definidas para o tipo BIT.

# VHDL - Tipos de Dados

## STD\_LOGIC (Standard Logic)

- Tipo resolvido STD\_LOGIC (subtipo do tipo STD\_ULOGIC) definido pela Norma IEEE 1164 contém uma função de resolução que permite que um sinal receba valores de múltiplas fontes. Esta operação não é permitida para os tipos BIT e STD\_ULOGIC.
- Exemplo 01 do uso do tipo STD\_LOGIC:

```
ARCHITECTURE Logica OF Example IS
```

```
SIGNAL s1 : STD_LOGIC;
```

```
BEGIN
```

```
s1 <= '0';
```

```
:
```

```
:
```

```
s1 <= '1';
```

```
END Logica;
```

# VHDL - Tipos de Dados

## STD\_LOGIC (Standard Logic)

- Exemplo 01 do uso do tipo STD\_LOGIC: (continuação)

```
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF
std_ulogic;
CONSTANT resolution_table : stdlogic_table := (
  --|U    X    0    1    Z    W    L    H    -    |    |
  ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
  ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
  ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
  ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
  ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
  ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
  ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
  ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
  ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ) -- | - |
);
```

# VHDL - Tipos de Dados

## STD\_LOGIC (Standard Logic)

- Exemplo 01 do uso do tipo STD\_LOGIC: (continuação)
  - Um estado lógico é determinado por duas condições: nível lógico e força
  - Cada valor possui um grau de força e um nível lógico.
  - 'U' é o valor de grau de força mais elevado
  - '-' é o valor de grau de força mais fraco
  - '0' e '1' possuem o mesmo grau de força
  - 'L' e 'H' possuem o mesmo grau de força → 'L' - nível lógico 0 e 'H' - nível lógico 1
  - Conflito de 'U' com qualquer outro valor sempre resultará no valor 'U'
  - Conflito de valores iguais, o resultado será o próprio valor
  - Conflito entre os valores 'X', '0' ou '1', o resultado será o valor 'X'
  - Conflito entre os valores 'W', 'L' ou 'H', o resultado será o valor 'W'

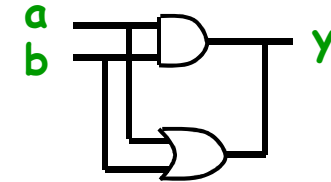
# VHDL - Tipos de Dados

## STD\_LOGIC (Standard Logic)

- Exemplo 02 do uso do tipo STD\_LOGIC:

VHDL não permite um sinal ter mais de uma fonte, a menos que tenha uma função de resolução associada.

```
ARCHITECTURE ComErro OF proj1 IS
BEGIN
    y <= a AND b;
    y <= a OR b;
END ComErro;
```



a	b	a and b	a or b	y
0	0	0	0	0
0	1	0	1	X
1	0	0	1	X
1	1	1	1	1

```
FUNCTION calcula_valor (a, b : X01) RETURN X01 IS
BEGIN
    IF a /= b THEN RETURN ('X'); -- se há conflito, sai 'X'
    ELSE RETURN (a);             -- ou return(b)
END;
```

# VHDL - Tipos de Dados

## STD\_LOGIC (Standard Logic)

- Nota 1: De maneira análoga ao tipo BIT, existe no STD\_ULOGIC uma representação vetorial de valores, o tipo composto STD\_ULOGIC\_VECTOR que substitui o tipo BIT\_VECTOR.
- Nota 2: O tipo composto STD\_ULOGIC\_VECTOR pode utilizar todas as operações lógicas definidas para o tipo BIT\_VECTOR.
- Nota 3: Objetos declarados com tipos STD\_ULOGIC e STD\_LOGIC permitem a transferência de valores entre eles.
- Nota 4: Objetos declarados com tipos STD\_ULOGIC\_VECTOR e STD\_LOGIC\_VECTOR não permitem a transferência de valores entre eles, a não ser que se faça uma operação de conversão.

# Resumo da Aula de Hoje

## Tópicos mais importantes:

- Tipos de Dados



# Próxima da Aula

- **Classes de Objetos**
  - **Constantes**
  - **Variáveis**
  - **Sinais**