

Sistemas Reconfiguráveis

Eng. de Computação

Profs. Francisco Garcia e Antônio Hamilton Magalhães

Aula 6

Outros tipos.

Exemplos de uso do código concorrente: Unidade lógica e aritmética (ALU)

VHDL

Tipos

Vimos anteriormente os tipos pré-definidos na linguagem VHDL padrão IEEE 1076:

- BIT / BIT_VECTOR, BOOLEAN, INTEGER, NATURAL, POSITIVE, REAL, TIME, CHARACTER

Vimos também os tipos definidos através do pacote IEEE 1164:

- STD_LOGIC / STD_LOGIC_VECTOR, STD_ULOGIC / STD_ULOGIC_VECTOR

Um tipo define um conjunto de valores e um conjunto de operações permitidas

Vamos ver agora outros tipos, definidos pelo usuário ou definidos através de outros pacotes

VHDL

Tipos definidos pelo usuário

Tipos definidos pelo usuário:

- Subconjunto de **INTEGER**
- Enumerados
- **ARRAY**

VHDL

Tipos definidos pelo usuário

Subconjunto de **INTEGER**

- Exemplo de declaração e uso:

```
TYPE notas : INTEGER RANGE 0 TO 100;  
SIGNAL h : notas;
```

.....

```
h <= 90;
```

Declaração de
tipo

VHDL

Tipos definidos pelo usuário

Enumerados

- Exemplo de declaração e uso:

```
TYPE cores IS (vermelho, verde, azul);  
SIGNAL c : cores;
```

.....

```
c <= azul;
```

Separado por
vírgula

VHDL

Tipos definidos pelo usuário

ARRAY

- Sintaxe:

`TYPE array_type_name IS ARRAY (integer_range) OF type_name;`

- Exemplo de declaração e uso:

```
TYPE meu_array IS ARRAY (0 TO 3) OF INTEGER;  
SIGNAL z : meu_array;
```

.....

```
z <= (123, 234, 345, 456);
```

Equivale a:

```
z(0) <= 123;
```

```
z(1) <= 234;
```

```
z(2) <= 345;
```

```
z(3) <= 456;
```

VHDL

Tipos definidos pelo usuário

ARRAY

- Exemplo de declaração e uso de ARRAY bidimensional:

```
TYPE meu_array2 IS ARRAY (0 TO 3, 0 TO 1) OF INTEGER;  
SIGNAL w : meu_array2;
```

```
.....
```

```
w(0,1) <= 123;
```

VHDL

Tipos definidos pelo usuário

Um **TYPE** pode ser declarado dentro de um **PACKAGE** (arquivo separado) e esse **PACKAGE** pode ser declarado como uma biblioteca

- Exemplo – declarando os tipos no arquivo **meus_tipos.vhd**:

```
PACKAGE meus_tipos IS
    TYPE vetor4x8 IS ARRAY (0 TO 3) OF BIT_VECTOR(7 DOWNT0 0);
    TYPE uint5 IS INTEGER RANGE 0 TO 31;
END meus_tipos;
```


VHDL

Tipos definidos pelo usuário

Um **TYPE** pode ser declarado dentro de um **PACKAGE** (arquivo separado) e esse **PACKAGE** pode ser declarado como uma biblioteca

- Exemplo – declarando o pacote e usando os tipos definidos em outra **ENTITY** (outro arquivo):

```
LIBRARY work;  
USE work.meus_tipos.all;  
  
ENTITY ex1 IS  
    PORT (  
        a : IN vetor4x8;  
        b : IN uint5;  
        .....  
    );  
END ENTITY;
```

VHDL

Tipos definidos pelo usuário

Um **TYPE** pode ser declarado como sendo **irrestrito**, ou seja, sem que a faixa seja definida na declaração. Nesse caso, ao se criar um objeto com esse **TYPE**, a faixa deve ser declarada.

- Exemplo:

```
TYPE meu_tipo IS ARRAY (NATURAL RANGE <>) OF BIT_VECTOR(7 DOWNT0 0);
```

```
SIGNAL aaa : meu_tipo (0 TO 3);
```

```
SIGNAL bbb : meu_tipo (15 DOWNT0 0);
```

Faixa a ser
posteriormente definida

VHDL - Op. aritméticas com STD_LOGIC_VECTOR



O tipo `STD_LOGIC_VECTOR` definido no pacote IEEE 1164 pode ser usado apenas em operações lógicas. Operações aritméticas não são permitidas com esse tipo. Usando um dos pacotes `std_logic_unsigned` ou `std_logic_signed` torna-se possível realizar operações aritméticas com esse tipo (além das operações lógicas)

Exemplo no slide seguinte

VHDL - Op. aritméticas com STD_LOGIC_VECTOR

- Exemplo:

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

ENTITY ex2 IS
    PORT (
        a,b : IN STD_LOGIC_VECTOR(7 DOWNT0 0);
        x,y : OUT STD_LOGIC_VECTOR(7 DOWNT0 0)
    );
END ENTITY;

ARCHITECTURE arch OF ex2 IS
BEGIN
    x <= a + b;
    y <= a AND b;
    ....
```

Pode-se usar `std_logic_unsigned` ou
`std_logic_signed`
Nunca os dois ao mesmo tempo.

VHDL – Pacote arith

Tipos SIGNED e UNSIGNED

Tipos definidos no pacote `arith` do IEEE:

- `SIGNED`
- `UNSIGNED`

Semelhantes a `SignedLogicVector`, porém permitem operações aritméticas

Obs.: É necessário declarar biblioteca:

```
LIBRARY ieee;  
USE ieee.std_logic_arith.all;
```

VHDL

Pacote numeric_std

O pacote arith, mencionado no slide anterior, cria dois tipos novos, SIGNED e UNSIGNED.

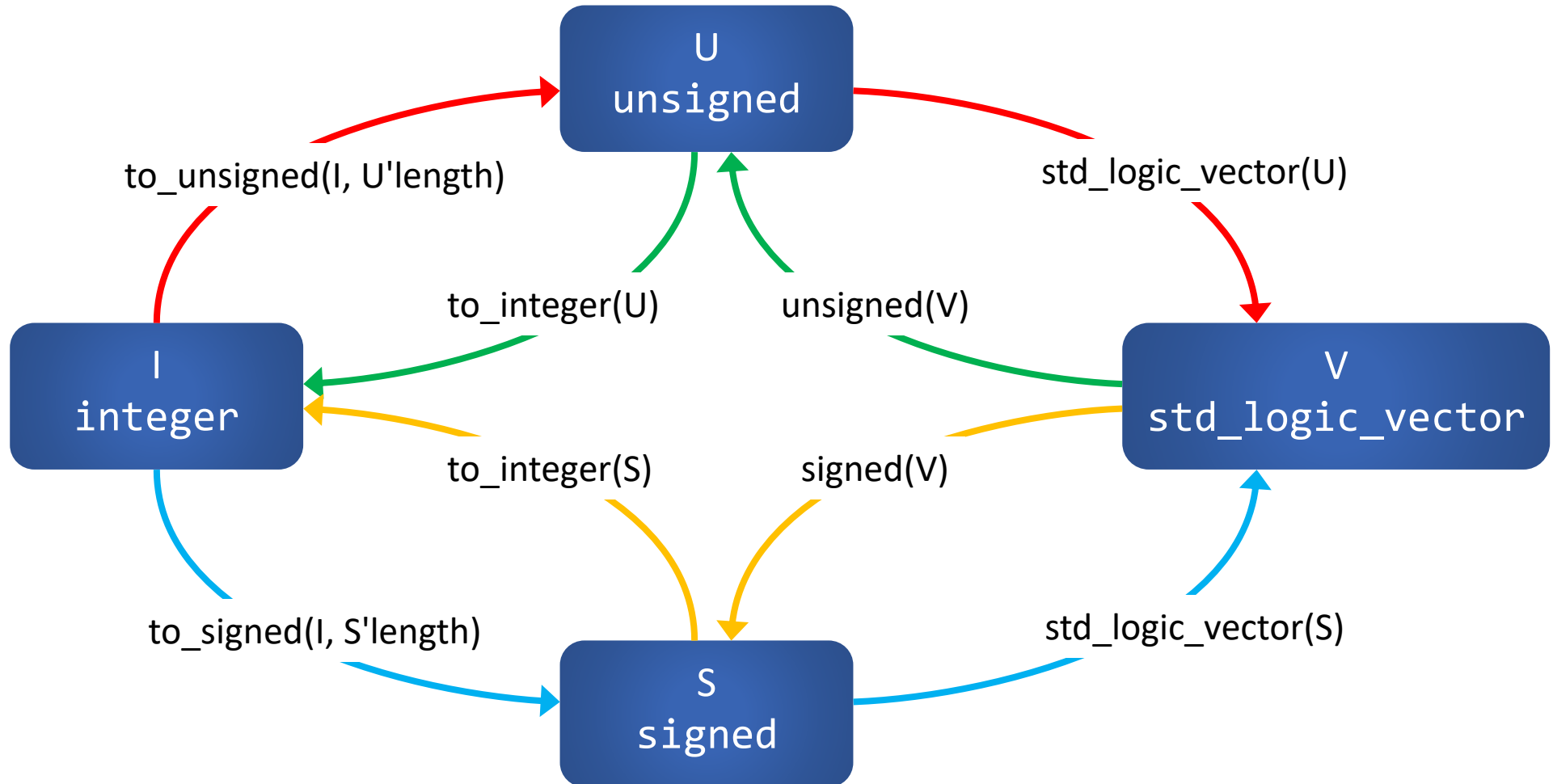
No entanto, existe algumas ambiguidades quando se usa esse pacote, no que diz respeito às operações envolvendo números com sinal e sem sinal.

Para sanar esses problemas, foi gerado um pacote chamado **numeric_std**, que deve ser preferido no lugar do pacote arith.

Além de definir os tipos SIGNED e UNSIGNED, o pacote numeric_std tem funções para conversão de tipos.

VHDL – pacote numeric_std

Conversão de tipos



VHDL – pacote numeric_std

Conversão de tipos

- Exemplo:

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

ENTITY .....
.....

ARCHITECTURE arch OF ex3 IS
    SIGNAL p, q : STD_LOGIC_VECTOR(7 DOWNT0 0);
    SIGNAL j : INTEGER RANGE 0 TO 255;
    SIGNAL k : INTEGER RANGE -128 TO 127;
BEGIN
    j <= TO_INTEGER(UNSIGNED(p));
    q <= STD_LOGIC_VECTOR(TO_SIGNED(k,8));
    ....
```

É necessário
declarar biblioteca

Conversão de vetor
para inteiro

Conversão de
inteiro para vetor

VHDL – pacote numeric_std

Conversão de tipos

- Exemplo – multiplexador de 8 entradas para 1 saída:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY mux_8x1 IS
    PORT (
        i: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        s: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        y: OUT STD_LOGIC
    );
END ENTITY;
ARCHITECTURE arch OF mux_8x1 IS
BEGIN
    WITH s SELECT
        y <= i(0) WHEN "000",
              i(1) WHEN "001",
              i(2) WHEN "010",
              i(3) WHEN "011",
              i(4) WHEN "100",
              i(5) WHEN "101",
              i(6) WHEN "110",
              i(7) WHEN "111";
END arch;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY mux_8x1 IS
    PORT (
        i: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        s: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        y: OUT STD_LOGIC
    );
END ENTITY;
ARCHITECTURE arch OF mux_8x1 IS
    SIGNAL s_int : INTEGER RANGE 0 TO 7;
BEGIN
    s_int <= TO_INTEGER(UNSIGNED(s));
    y <= i(s_int);
END arch;
```

Código concorrente



Nas últimas aulas, vimos alguns exemplos de circuitos combinacionais utilizando código concorrente de VHDL:


- Decodificador
- Codificador
- Multiplexador

Na aula de hoje vamos ver outros exemplos de circuitos combinacionais

- Unidade lógica e aritmética (ALU)

Unidade lógica e aritmética (*ALU*)

- Duas entradas de dados (operandos)
- A entrada de seleção de operação (N bits) indica qual operação a ser efetuada sobre os operandos
- Pode ter até 2^N operações diferentes
- Uma saída de resultado
- Pode ter saídas auxiliares que indicam condições especiais do resultado, por exemplo:
 - Zero
 - ***Carry***
 - *Overflow*
- Pode ter uma entrada auxiliar de *carry*, usada em operações aritméticas e de deslocamento



Muitas ALU têm
saída de *carry*

Adição binária

- Exemplos:

binário	decimal
$\begin{array}{r} 01011010 \\ + 00110111 \\ \hline 10010001 \end{array}$	$\begin{array}{r} 90 \\ + 55 \\ \hline 145 \end{array}$

binário	decimal
$\begin{array}{r} 11011100 \\ + 00110111 \\ \hline 00010011 \end{array}$	$\begin{array}{r} 220 \\ + 55 \\ \hline 19 \end{array}$

Houve um *carry*
(vai um) no bit
mais significativo

O resultado é maior que 255 e não
pode ser representado com 8 bits

Subtração binária

- A subtração é feita somando-se o minuendo com o complemento de dois do subtraendo:

binário		decimal
01011010	01011010	90
- 00110111	+ 11001001	55
	<hr/>	<hr/>
	00100011	35

Houve um *carry* (vai um) no bit mais significativo. O resultado está correto

Como o minuendo é maior que o subtraendo, o resultado é positivo

Subtração binária

- A subtração é feita somando-se o minuendo com o complemento de dois do subtraendo:

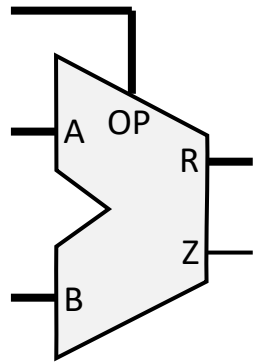
binário		decimal
00110111	00110111	55
- 01011010	+ 10100110	90
	<hr/>	<hr/>
	11011101	-35

Não houve um *carry* (vai um) no bit mais significativo. O resultado é negativo e está em complemento de dois

Como o minuendo é menor que o subtraendo, o resultado é negativo

Exemplo lu_1

Exemplo lu_1: unidade lógica simples, com operandos de 8 bits e com 4 operações



Resultado da
operação,
conforme tabela

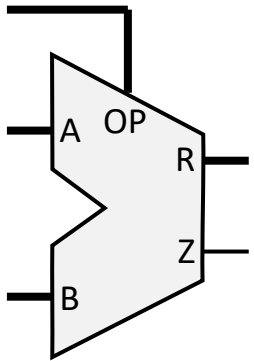
Z = '1' se o
resultado for zero

Obs.: Nesse exemplo,
A, B e R têm 8 bits

Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	A OR B
1	0	A XOR B
1	1	NOT A

Exemplo lu_1

Exemplo lu_1: unidade lógica simples, com operandos de 8 bits e com 4 operações



Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	A OR B
1	0	A XOR B
1	1	NOT A

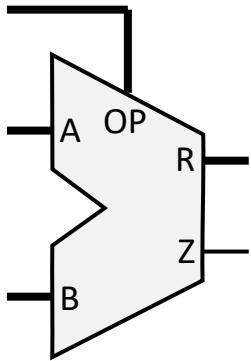
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY lu_1 IS
    PORT (
        a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        op: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        r: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        z: OUT STD_LOGIC
    );
END ENTITY;
```

Continua na próxima página

Exemplo lu_1

Exemplo lu_1: unidade lógica simples, com operandos de 8 bits e com 4 operações



Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	A OR B
1	0	A XOR B
1	1	NOT A

```
ARCHITECTURE arch OF lu_1 IS
    SIGNAL aux : STD_LOGIC_VECTOR(7 DOWNT0 0);
BEGIN

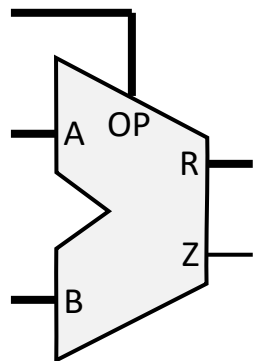
    WITH op SELECT
        aux <= a AND b WHEN "00",
              a OR b  WHEN "01",
              a XOR b WHEN "10",
              NOT a   WHEN "11";

    r <= aux;
    z <= '1' WHEN aux = "00000000" ELSE '0';

END arch;
```

Exemplo alu_2

Exemplo alu_2: unidade lógica e aritmética (ALU) simples, com operandos de 8 bits e com 4 operações



Resultado da operação lógica ou aritmética, conforme tabela

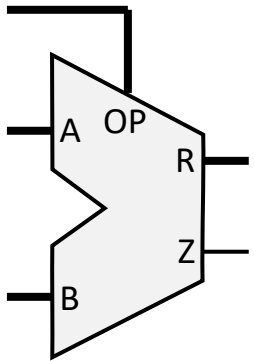
Z = '1' se o resultado for zero

Obs.: Nesse exemplo, A, B e R têm 8 bits

Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	NOT A
1	0	A + B
1	1	A - B

Exemplo alu_2

Exemplo alu_2: unidade lógica e aritmética (ALU) simples, com operandos de 8 bits e com 4 operações



Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	NOT A
1	0	A + B
1	1	A - B

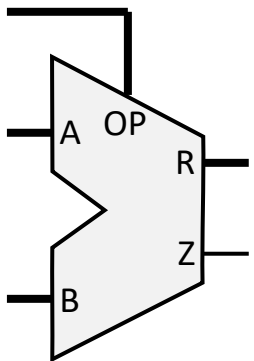
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY alu_2 IS
    PORT (
        a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        op: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        r: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        z: OUT STD_LOGIC
    );
END ENTITY;
```

Continua na próxima página

Exemplo alu_2

Exemplo alu_2: unidade lógica e aritmética (ALU) simples, com operandos de 8 bits e com 4 operações



Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	NOT A
1	0	A + B
1	1	A - B

```
ARCHITECTURE arch OF alu_2 IS
    SIGNAL aux : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN

    WITH op SELECT
        aux <= a AND b WHEN "00",
              NOT a  WHEN "01",
              a + b  WHEN "10",
              a - b  WHEN "11";

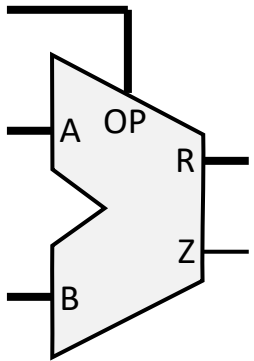
    r <= aux;
    z <= '1' WHEN aux = "00000000" ELSE '0';

END arch;
```

Errado, pois a e b são vetores e não admitem operações aritméticas

Exemplo alu_2

Exemplo alu_2: unidade lógica e aritmética (ALU) simples, com operandos de 8 bits e com 4 operações



O problema apontado na página anterior pode ser resolvido:

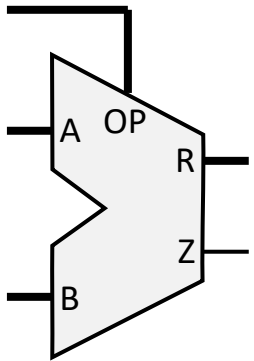
- Usando um dos pacotes [std_logic_unsigned](#) ou [std_logic_signed](#), que torna possível realizar operações aritméticas com o tipo [STD_LOGIC_VECTOR](#).
- Usando o pacote [numeric_std](#) e as funções de conversão de tipo.

Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	NOT A
1	0	A + B
1	1	A - B

Exemplo alu_2

Solução 1

Exemplo alu_2: unidade lógica e aritmética (ALU) simples, com operandos de 8 bits e com 4 operações



Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	NOT A
1	0	A + B
1	1	A - B

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

ENTITY alu_2 IS
    PORT (
        a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        op: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        r: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        z: OUT STD_LOGIC
    );
END ENTITY;
```

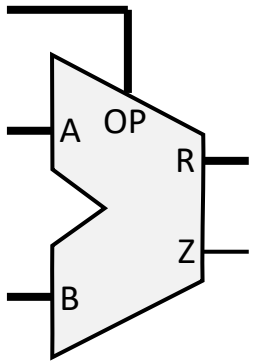
Pacote
acrescentado

Continua na próxima página

Exemplo alu_2

Solução 1

Exemplo alu_2: unidade lógica e aritmética (ALU) simples, com operandos de 8 bits e com 4 operações



Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	NOT A
1	0	A + B
1	1	A - B

```
ARCHITECTURE arch1 OF alu_2 IS
    SIGNAL aux : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
```

```
    WITH op SELECT
        aux <= a AND b WHEN "00",
               NOT a   WHEN "01",
               a + b   WHEN "10",
               a - b   WHEN "11";
```

```
    r <= aux;
    z <= '1' WHEN aux = "00000000" ELSE '0';
```

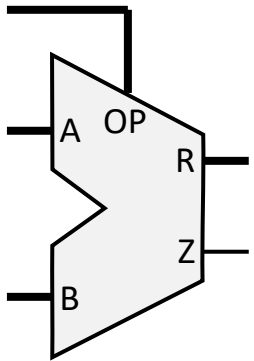
```
END arch1;
```

Operações possíveis com o pacote std_unsigned

Exemplo alu_2

Solução 2

Exemplo alu_2: unidade lógica e aritmética (ALU) simples, com operandos de 8 bits e com 4 operações



Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	NOT A
1	0	A + B
1	1	A - B

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

ENTITY alu_2 IS
    PORT (
        a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        op: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        r: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        z: OUT STD_LOGIC
    );
END ENTITY;
```

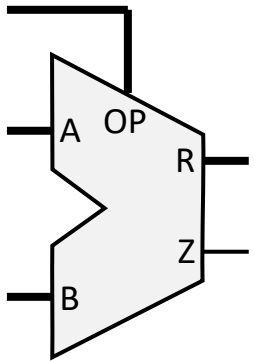
Pacote
acrescentado

Continua na próxima página

Exemplo alu_2

Solução 2

Exemplo alu_2: unidade lógica e aritmética (ALU) simples, com operandos de 8 bits e com 4 operações



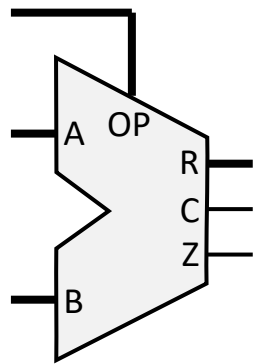
Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	NOT A
1	0	A + B
1	1	A - B

```
ARCHITECTURE arch2 OF alu_2 IS
    SIGNAL aux : STD_LOGIC_VECTOR(7 DOWNT0 0);
    SIGNAL aux_add : INTEGER RANGE 0 TO 255;
    SIGNAL aux_sub : INTEGER RANGE 0 TO 255;
BEGIN
    aux_add <= TO_INTEGER(UNSIGNED(a)) + TO_INTEGER(UNSIGNED(b));
    aux_sub <= TO_INTEGER(UNSIGNED(a)) - TO_INTEGER(UNSIGNED(b));
    WITH op SELECT
        aux <= a AND b WHEN "00",
              NOT a WHEN "01",
              STD_LOGIC_VECTOR(TO_UNSIGNED(aux_add , 8)) WHEN "10",
              STD_LOGIC_VECTOR(TO_UNSIGNED(aux_sub , 8)) WHEN "11";

    r <= aux;
    z <= '1' WHEN aux = "00000000" ELSE '0';
END arch2;
```

Exemplo alu_3

Exemplo alu_3: ALU simples, com operandos de 8 bits e 4 operações



Resultado da operação, conforme tabela

C = '1' se houver *carry* nas operações aritméticas. Caso contrário, C = '0'

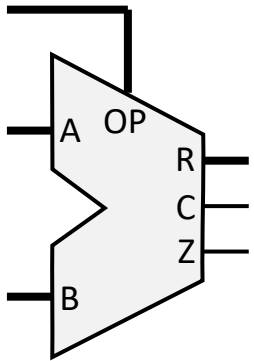
Z = '1' se o resultado for zero

Obs.: Nesse exemplo, A, B e R têm 8 bits

Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	NOT A
1	0	A + B
1	1	A - B

Exemplo alu_3

Exemplo alu_3: ALU simples, com operandos de 8 bits e 4 operações



Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	NOT A
1	0	A + B
1	1	A - B

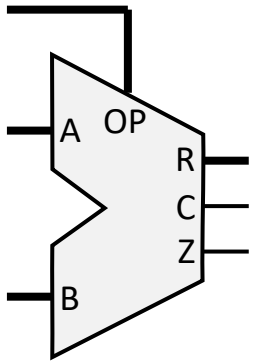
Conforme já apontado anteriormente, não é possível realizar operações aritméticas com o tipo `STD_LOGIC_VECTOR`, a não ser:

- Usando um dos pacotes `std_logic_unsigned` ou `std_logic_signed`, que torna possível realizar operações aritméticas com o tipo `STD_LOGIC_VECTOR`.
- Usando o pacote `numeric_std` e as funções de conversão de tipo.

Para a detecção do *carry*, é necessário que as operações sejam feitas com **9 bits**. O bit de *carry* será o **bit mais significativo**.

Exemplo alu_3

Exemplo alu_3: ALU simples, com operandos de 8 bits e 4 operações



Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	NOT A
1	0	A + B
1	1	A - B

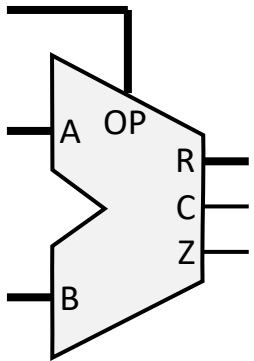
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY alu_3 IS
    PORT (
        a, b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        op: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        r: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        c, z: OUT STD_LOGIC
    );
END ENTITY;
```

Continua na próxima página

Exemplo alu_3

Exemplo alu_3: ALU simples, com operandos de 8 bits e 4 operações



Entradas		Operação
OP1	OP0	R
0	0	A AND B
0	1	NOT A
1	0	A + B
1	1	A - B

```
ARCHITECTURE arch OF alu_3 IS
    SIGNAL aux : STD_LOGIC_VECTOR(8 DOWNT0 0);
BEGIN
    WITH op SELECT
        aux <= '0' & (a AND b)           WHEN "00",
              '0' & NOT a                 WHEN "01",
              ('0' & a) + ('0' & b)      WHEN "10",
              ('0' & a) - ('0' & b)      WHEN OTHERS;

    r <= aux(7 DOWNT0 0);
    c <= aux(8);
    z <= '1' WHEN aux(7 DOWNT0 0) = "00000000" ELSE
        '0';

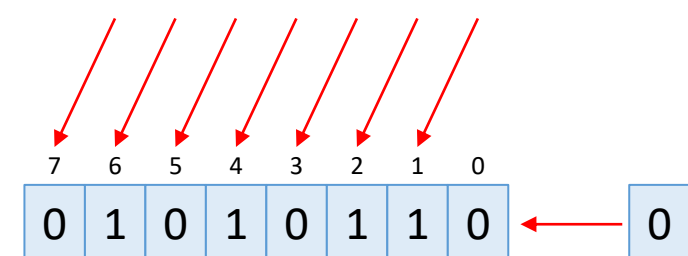
END arch;
```

Deslocamento (*shift*)

Muitas unidades lógicas e aritméticas têm operações de deslocamento (*shift*) e rotação (*rotate*), que são muito usadas em processadores.

A operação de deslocamento para a esquerda (*shift left*) equivale a uma multiplicação binária, enquanto o deslocamento para a direita (*shift right*) equivale a uma divisão.

Exemplo:

Número binário com 8 bits								Equivalente decimal
7	6	5	4	3	2	1	0	
0	0	1	0	1	0	1	1	43(d)
								
7	6	5	4	3	2	1	0	
0	1	0	1	0	1	1	0	86(d)

O número foi **deslocado** para a esquerda de 1 posição, sendo preenchido com '0' à direita. O último bit à esquerda foi perdido.

Deslocamento (*shift*)



Existem dois tipos de deslocamento: lógico e aritmético

- Deslocamento lógico
 - SLL = *Shift left logic*
 - SRL = *Shift right logic*

O número é deslocado para a esquerda ou direita, sendo preenchido com '0' do lado oposto ao deslocamento.

Exemplos:

- O número "00101110" deslocado de duas posições para a esquerda resulta em "10111000"
- O mesmo número deslocado de uma posição para a direita resulta em "00010111"

Deslocamento (*shift*)



Existem dois tipos de deslocamento: lógico e aritmético

- Deslocamento aritmético
 - SLA = *Shift left arithmetic*
 - SRA = *Shift right arithmetic*

← Equivale a divisão de um número com sinal (SIGNED)

O número é deslocado para a esquerda ou direita, sendo preenchido com a repetição do último bit do lado oposto ao deslocamento.

Exemplos:

- O número “10101101” deslocado de duas posições para a esquerda resulta em “101101**11**”
- O mesmo número deslocado de uma posição para a direita resulta em “**1**1010110”

Deslocamento (*shift*)



Obs.: Existem definições conflitantes no que diz respeito à operação de deslocamento aritmético para a esquerda:

- Em alguns casos, o bit mais a direita é repetido para preencher a posição deixada vazia pelo deslocamento.
- Em outros, a posição mais a direita é preenchida com '0', o que seria equivalente ao deslocamento lógico.

Deslocamento (*shift*)



Na linguagem VHDL existem os operadores **SLL**, **SRL**, **SLA** e **SRA**, que operam somente sobre o tipo **BIT**. No entanto, existem alguns erros nessas operações, e esses operadores não são mais usados.

No pacote **numeric_std** do IEEE, foram introduzidos os operadores **SHIFT_LEFT** e **SHIFT_RIGHT**, que operam sobre os tipos **SIGNED** ou **UNSIGNED**.

```
result <= SHIFT_LEFT(operand1, operand2);
```

```
result <= SHIFT_RIGHT(operand1, operand2);
```

O operando 2 é um inteiro que estabelece o número de posições deslocadas.

O resultado e o operando 1 devem ser do mesmo tipo. Se forem do tipo **UNSIGNED**, o deslocamento é do tipo lógico. Se forem do tipo **SIGNED**, o deslocamento é aritmético.

Deslocamento (*shift*)

Abaixo são mostrados exemplos de implementação da função de deslocamento para a direita sem usar nenhum operador especial e trabalhando com o tipo `STD_LOGIC_VECTOR`:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ex_sr IS
    PORT (
        a : IN STD_LOGIC_VECTOR(7 DOWNT0 0);
        x,y : OUT STD_LOGIC_VECTOR(7 DOWNT0 0)
    );
END ENTITY;
ARCHITECTURE arch OF ex_sr IS
BEGIN
    x <= '0' & a(7 downto 1); -- deslocamento lógico
    y <= a(7) & a(7 downto 1); -- deslocamento aritmético
END arch;
```

Equivale a:

```
x(7) <= '0';
X(6) <= a(7);
X(5) <= a(6);
X(4) <= a(5);
X(3) <= a(4);
X(2) <= a(3);
X(1) <= a(2);
X(0) <= a(1);
```

Exercício 1

Tendo como ponto de partida o último exemplo apresentado (alu_3), implementar e testar no módulo DE2 uma unidade lógica aritmética que funcione de acordo com a tabela abaixo:

Entradas			Operação
OP[2..0]			R
0	0	0	A AND B
0	0	1	A OR B
0	1	0	A XOR B
0	1	1	NOT A
1	0	0	SLL A
1	0	1	SRL A
1	1	0	A + B
1	1	1	A - B

} Deslocamentos
de 1 posição

A saída **z** deverá ser igual a '1' quando o resultado **r** for igual a "00000000".

A saída **c**:

- Nas operações lógicas, deverá ser igual a '0'.
- Nas operações aritméticas, deverá se comportar conforme descrito anteriormente (*carry out*)
- Nas operações de deslocamento, deverá receber o bit que seria perdido pelo deslocamento.

Configurar algumas chaves do módulo para as entradas e LEDs para as saídas



Fim

Até a próxima aula!