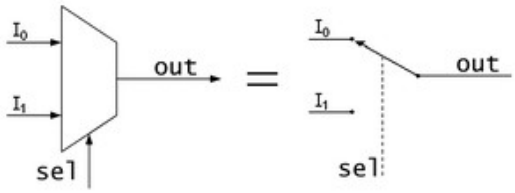
µProcessador 2 Multiplexação, Barramentos, Números: a ULA

Multiplexação

Um mux é um seletor: ele escolhe uma das entradas possíveis para fornecer à saída.



(Figura: wikipedia)

Se quisermos um mux com 4 entradas ao invés de 2, a descrição fica:

```
entity mux4x1 is
   port( sel0,sel1                : in std_logic;
         entr0,entr1,entr2,entr3 : in std_logic;
         saida                   : out std_logic
   );
end entity;
```

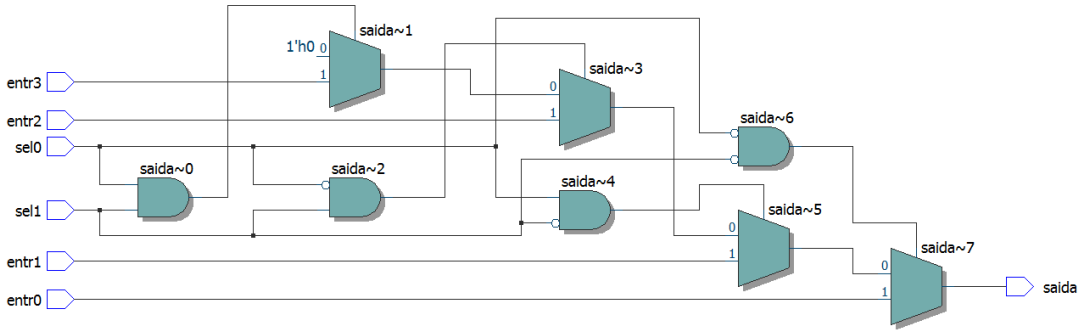
Observe a versão reduzida da tabela verdade do mux 4x1 acima.

sel1	sel0	saida
0	0	entr0
0	1	entr1
1	0	entr2
1	1	entr3

Em VHDL há diversas maneiras de fazer multiplexação. Vamos ficar com a seguinte:

```
architecture a_mux4x1 of mux4x1 is
begin
   saida <=    entr0    when sel1='0' and sel0='0' else
               entr1    when sel1='0' and sel0='1' else
               entr2    when sel1='1' and sel0='0' else
               entr3    when sel1='1' and sel0='1' else
               '0';     -- esse '0' é pra quando "der pau" em sel1 ou sel0
end architecture;
```

Lembre-se: *isso não é programação, isso é um circuito!* Abaixo vemos o RTL resultante deste trecho.

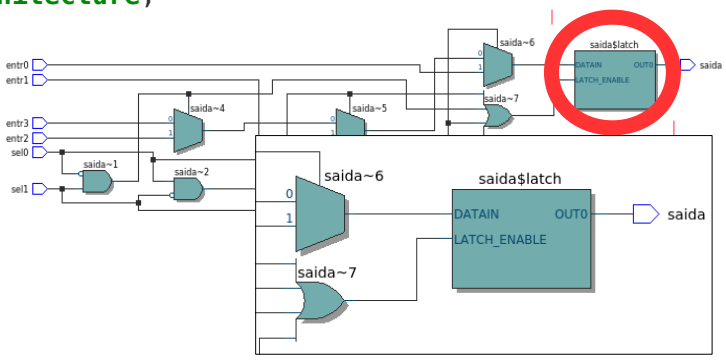


Mas pra que serve aquele “else '0';” da última linha? Não dá pra retirar?

Na verdade não... Lembre-se que o IEEE 1164 tem *nove* estados lógicos possíveis e *você deve cobrir todos os casos possíveis.* Se levássemos em conta apenas o 'U' (não inicializado) e o 'X' (desconhecido) além de 0 e 1, teremos *quinze* casos, se eu contei direito: 00, 01, 10, 11, 0U, U0, 1U, U1, 0X, X0, 1X, X1, XX, UX e XU. O “else zero” cobre os outros treze deste exemplo, e mais ainda[1].

Por uma esquisitice do VHDL, quando os casos não são totalmente cobertos, ele coloca um registro para os valores, ou seja, vai inserir um *latch* no meio do nosso circuito...(!) Veja abaixo o VHDL, caso você esqueça do “else '0';”, e o circuito resultante em RTL.

```
architecture a_mux4x1 of mux4x1 is
begin
   saida <=    entr0    when sel1='0' and sel0='0' else
               entr1    when sel1='0' and sel0='1' else
               entr2    when sel1='1' and sel0='0' else
               entr3    when sel1='1' and sel0='1'; -- OPA! CADÊ O ELSE ZERO?
end architecture;
```



Ou seja:

ATENÇÃO: Numa estrutura when-else sempre termine com else '0';

Tá avisado.

▶	Construa um multiplexador 2x1 só que com um pino de *enable*, ou seja, se este pino estiver desativado (em '0'), a saída do mux estará sempre em '0'; se ele estiver ativado (em '1'), o pino de seleção escolhe uma das duas entradas para repassar à saída. Faça um *testbench* adequado.

Barramentos

Um barramento (*bus*) é apenas um feixe de sinais, um conjunto de fios. Aqui nesta disciplina vamos usá-los sempre como números inteiros, para facilitar.

Para isso, sempre inclua a biblioteca adicional padrão “numeric_std”[2] a partir de agora:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;     -- para usarmos UNSIGNED
```

Ao definir um barramento, usamos a *keyword* “unsigned” (como em *unsigned integer,* inteiro não sinalizado, ou seja, não-negativo) e obrigatoriamente indicamos a largura do barramento em bits.

1 É possível simplesmente omitir a condição da última linha, como abaixo:

```
            entr2    when sel1='1' and sel0='0' else
            entr3;
```

A desvantagem é que se houver algum problema, veremos 'entr3' na saída; se ao invés disso observarmos a saída fixa em zero, é mais fácil localizar o *bug*.

2 Há também a opção de usar a biblioteca *não padrão* “std_logic_arith”; mas **não use essa** pois *dá conflito* e é proprietária.

Por exemplo, vamos fazer um multiplexador 4x1 de largura de 8 bits:

```
entity mux8bits is
        port(   entr0 : in unsigned(7 downto 0);
                entr1 : in unsigned(7 downto 0);
                entr2 : in unsigned(7 downto 0);
                entr3 : in unsigned(7 downto 0);
                sel   : in unsigned(1 downto 0); -- bits de seleção num só bus
                saida : out unsigned(7 downto 0) -- lembre: sem ';' aqui
end entity;
```

O tipo "unsigned(7 downto 0)" significa que o sinal tem 8 bits (8 pinos) e deve ser tratado como um inteiro não sinalizado, sem números negativos.[3]

A implementação é quase idêntica ao mux de 1 bit:

```
architecture a_mux8bits of mux8bits is
begin
   saida <=    entr0 when sel="00" else -- observe as aspas duplas!
               entr1 when sel="01" else
               entr2 when sel="10" else
               entr3 when sel="11" else
               "00000000";              -- saida tem 8 bits, portanto 8 zeros
end architecture;
```

Números / Constantes

As constantes devem ser obrigatoriamente representadas por um vetor de bits[4], representado por aspas duplas. Note que a extensão (*length*) deve bater! Veja exemplos para um sinal "saida" de 8 bits:

```
   saida <= "00000010"; -- certo: cte. de 8 bits atribuída a sinal de 8 bits
   saida <= "010";      -- pau: mismatch de largura
   saida <= 65;         -- pau: mismatch de tipo unsigned <= inteiro;
```

O erro reportado na linha "saida <= 65;" é:

```
porta_e.vhdl:18:13:  can't  match  integer  literal  with  type
array type "unsigned"
ghdl: compilation error
```

pois o VHDL não sabe quantos bits a constante 65 possui[5]. Para resolver de forma simples, use o número equivalente em binário, "01000001", com a largura esperada por "saida".

Será fundamental recortarmos e combinarmos dados. Para isso, usamos o operador de concatenação "&" e o recurso de *slicing,* que obtém um trecho de bits.

Por exemplo, suponha dois barramentos de 8 bits:

```
signal x,y: unsigned(7 downto 0);
```

Podemos combinar os bits dos sinais da forma que quisermos. Como ilustração simples, vamos isolar alguns trechos arbitrários:

3 A *keyword* "downto" indica que a representação é *bit-level little endian,* ou seja, o bit 0 é o LSB (*least significant bit,* tem valor numérico 2^0). No nosso exemplo, o bit 7 é o MSB (*most significant bit*) e tem valor 2^7.

4 Números inteiros, como 5 ou 122 por exemplo, são constantes sem especificação de largura em bits e portanto não podem ser usados diretamente com o tipo "unsigned."

5 VHDL é uma linguagem *fortemente tipada,* o que na prática significa que ele requer conversões explícitas de tipos por parte do programador. Isso ajuda o código a chegar com menos erros na etapa de simulação.

(a) O bit mais significativo (MSB) do sinal x é dado por x(7)
(b) Os três bits menos significativos (LSB) de x são obtidos com x(2 downto 0)
(c) Para obter de b5 a b3, basta fazer x(5 downto 3)

Agora se quisermos por bobeira montar um número de 8 bits que começa com '0' e é seguido por (a), (b) e (c) nesta ordem, concatenamos tudo isso com o operador "&":

```
y <= '0' & x(7) & x(2 downto 0) & x(5 downto 3);
```

Portanto se x vale "01101010", y será "00010101". Se x for "11001100", teremos "01100001" em y.

Operações Aritméticas

Quando usamos o tipo "unsigned"e a biblioteca "numeric_std", dá pra fazer as paradinhas diretão[6]. Por exemplo, vamos fazer um circuito com duas entradas que calcula tanto a soma quanto a subtração deles simultaneamente:

```
entity soma_e_subtrai is
   port (   x,y       :  in unsigned(7 downto 0);
            soma,subt :  out unsigned(7 downto 0)
   );
end entity;

architecture a_soma_e_subtrai of soma_e_subtrai is
begin
   soma <= x+y;
   subt <= x-y;
end architecture;
```

Muito mole.

Se quisermos saber se o x é maior do que o y e saber se o x é negativo ou não, incluímos na definição da entidade:

```
   maior,x_negativo : out std_logic;
```

E na arquitetura basta colocar as linhas:

```
   maior <= '1' when x>y else
            '0' when x<=y else
            '0';
   x_negativo <= x(7);
```

Peralá, *negativo?* Sim, dá pra usar números sinalizados de 8 bits pra maioria das coisas (os circuitos são idênticos, pois realizam cálculos em módulo 2^8). Infelizmente, isso não funciona para a divisão. A comparação acima também *só aceita números positivos*.

▶	Construa um *testbench* para o circuito "soma_e_subtrai" e o simule. Isso é importante para você observar corretamente a sintaxe de constantes e compreender a arquitetura acima. **Teste direito:** faça casos relevantes tentando cobrir todas as operações. Por exemplo, se você testou 3+5, não há necessidade de testar 13+15, mas talvez seja interessante testar 100+100 e 200+200... Talvez não precise. Não esqueça de testar negativos, como 18+(-3).

6 Isso não ocorre sem essa biblioteca, ok? Sem ela, precisa de uns malabarismos.

Tarefa para Entregar: ULA

Nesta primeira entrega quero apenas o arquivo VHDL e o *testbench*.

De acordo com o plano de aulas, o valor de cada laboratório destes é de **0,4 na média** em cada entrega. A cada 48h de atraso, 10% da nota deste item de cronograma vai desaparecendo até chegar em 20%. *Não coma bola!* A validação do processador, ao final do semestre, valerá 2,1 na média, portanto o conjunto de provas valerá 5,5 na média final.

Equipes de um ou dois alunos. Aproximadamente a cada dois laboratórios, as equipes deverão apresentar, rápida e pessoalmente, o funcionamento para o professor. Verifique o cronograma.

Especificação da ULA:

- duas entradas de dados de 16 bits;
- uma saída de resultado de 16 bits;
- (opcional) uma ou mais saídas de sinalização de um bit (resultado zero, indicação de "maior" em comparação);
- entradas para seleção de operações.

No mínimo 4 operações, incluindo:

- soma;
- subtração;
- uma maneira de comparar dois números (pode ser operador "maior ou igual", ou então verificar o sinal de um número, ou outra maneira);
- *não implemente divisão,* pois a implementação do compilador VHDL pode dar uns erros com divisão por zero (são contornáveis, mas melhor evitar).

- Um *testbench* que cobre todas as operações tem que ser entregue; inclua nele um conjunto de testes razoável, que cobre todos os casos de interesse.
- Será utilizada no projeto do processador, portanto escolha operações sensatas.
- *Por favor* dê nomes claros aos sinais (por ex., *op*, *selec, selec_op* ou s*elecionaoperacao* e não "slo" ou "s")

Resumindo: entram dois dados de 16 bits, escolhe-se qual das 4 operações deve ser executada pela ULA e o resultado desta operação surge na saída de 16 bits. Entregue .vhd e _tb.vhd.

Ou seja, tem que fazer as operações e botar um mux na saída. É assim que se faz.