Componentes de blocos operativos

Prof. Ilan Sousa Correa

Universidade Federal do Pará (UFPA)

Instituto de Tecnologia (ITEC)

Faculdade de Eng. da Computação e Telecomunicações (FCT)

Introduçao

Nas aulas anteriores foram introduzidos alguns componentes básicos de sistemas digitais

- Portas lógicas, multiplexadores, decodificadores, registradores, e controladores

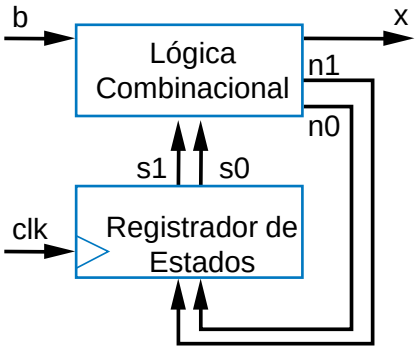
Controladores ou máquinas de estados são adequados para tratar entradas e saídas de controle

- Entrada de controle: geralmente 1 ou alguns bits representando o estado do ambiente
    - Pressionamento de botão, obstáculo detectado por um sensor

Como tratar dados mais complexos?

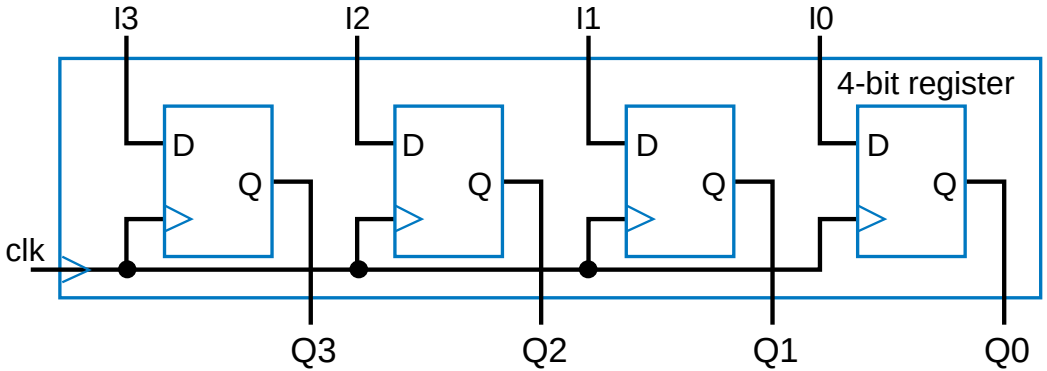
- Nesta aula, introduziremos o chamado “caminho de dados” ou bloco operativo
- Caminho de dados: associação de componentes básicos “em série” para tratar dados

Registradores



Já estudados antes

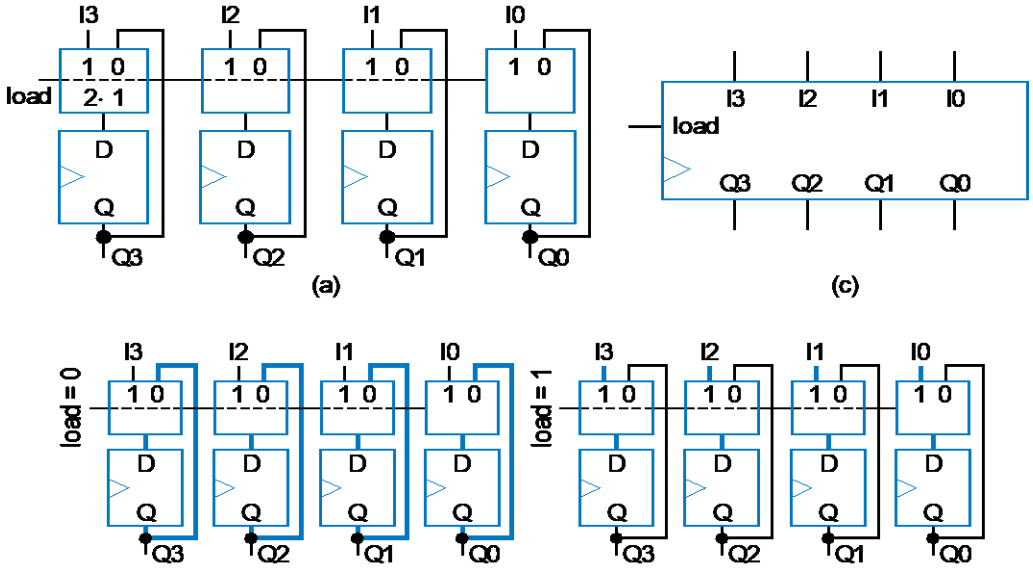
- Construídos a partir de flip-flops compartilhando o mesmo sinal de clock
- Usados nas aulas anteriores para a construção de memória que guardam o estado atual em uma máquina de estados
- Em sua construção básica, sempre salva a entrada e disponibiliza na saída



Registrador com carga paralela

Registrador + multiplexador para permitir salvar em ciclos específicos

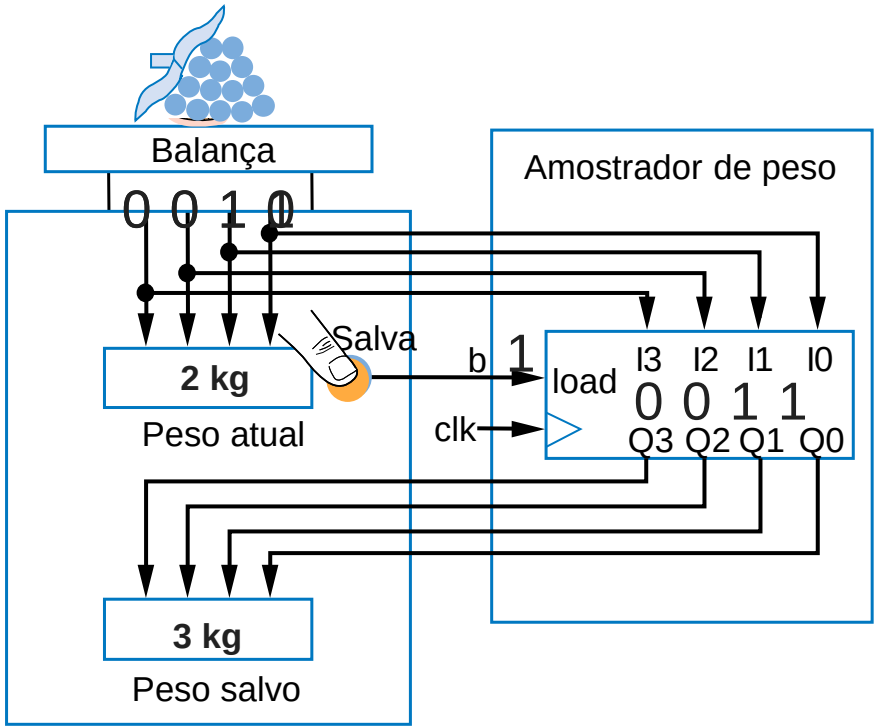
- Salva novo valor do barramento i, ou salva valor anterior Q através de mux e um sinal “load”



Registrador com carga paralela

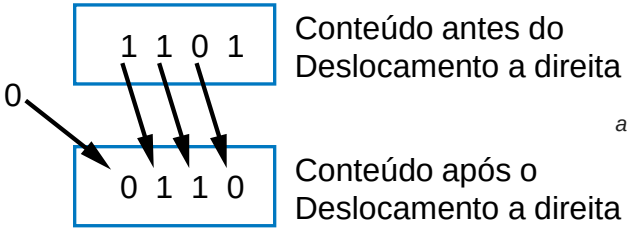
Exemplo de uso: amostrador de peso

- Recursos: mostrar o peso e salvar o peso
- Registrador armazena um peso anterior
- Dois displays

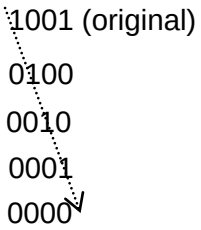


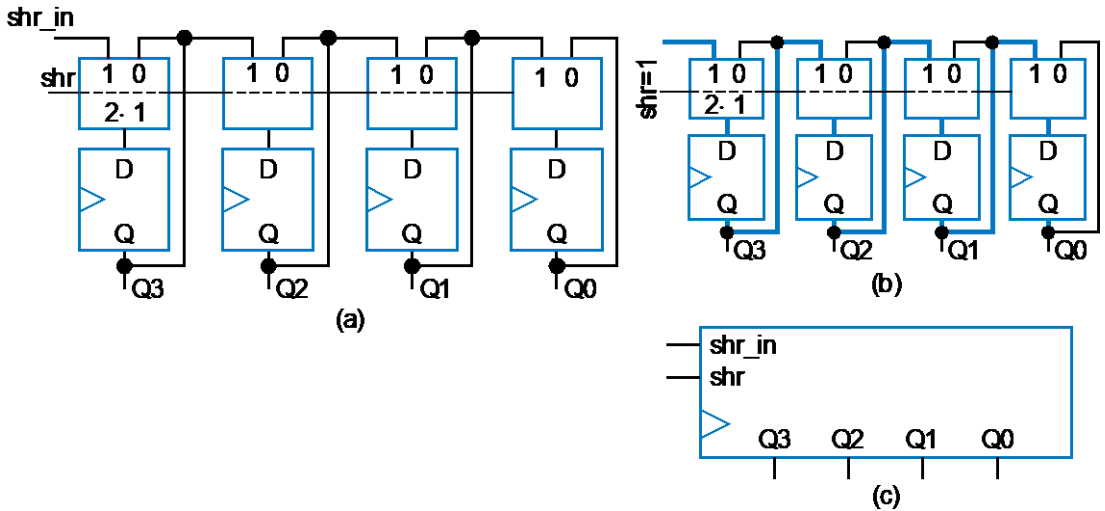
a

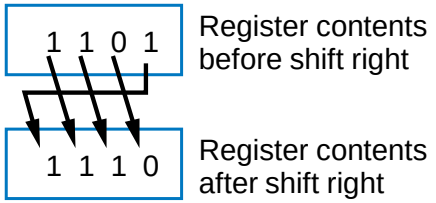
Registrador de deslocamento

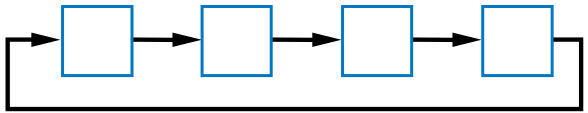


Registrador que pode manter o valor atual ou realizar um deslocamento





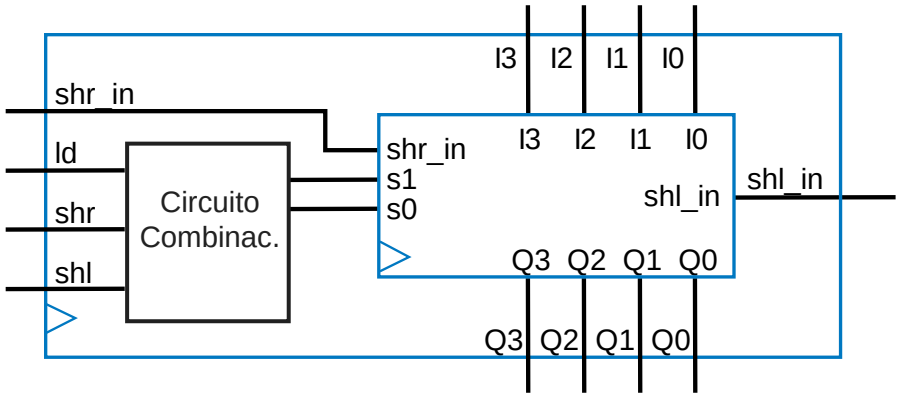




Registrador multi-funções

É possível combinar várias funções

Entradas			Saídas		
ld	shr	shl	s1	s0	Operação
0	0	0	0	0	Mantém valor
0	0	1	1	1	Deslocamento esquerda
0	1	0	1	0	Deslocamento direita
0	1	1	1	0	Deslocamento direita
1	0	0	0	1	Carga
1	0	1	0	1	Carga
1	1	0	0	1	Carga
1	1	1	0	1	Carga

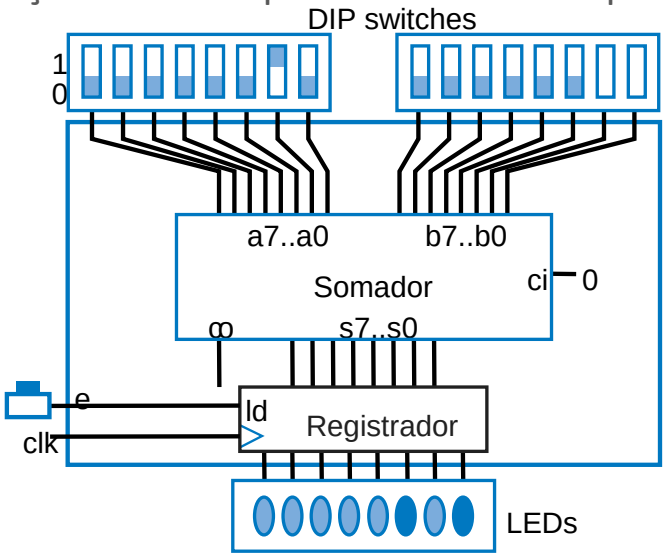


Somadores

Existem várias formas de implementar somadores (meio somador, somador complete, carry-ripple, etc)

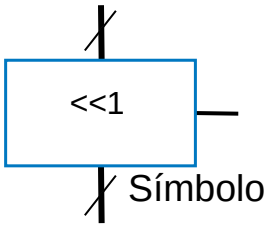
- Consideraremos a otimização realizada pela ferramenta de processamento de HDL, ou seja, para nós será “+”

Exemplo de uso

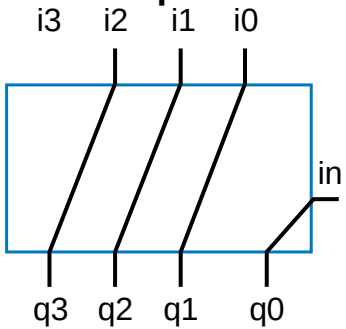


Deslocadores

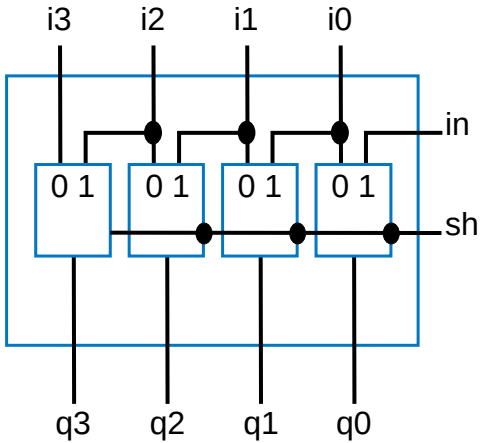
Circuito lógico que permite deslocar uma sequência de bits para esquerda ou para direita



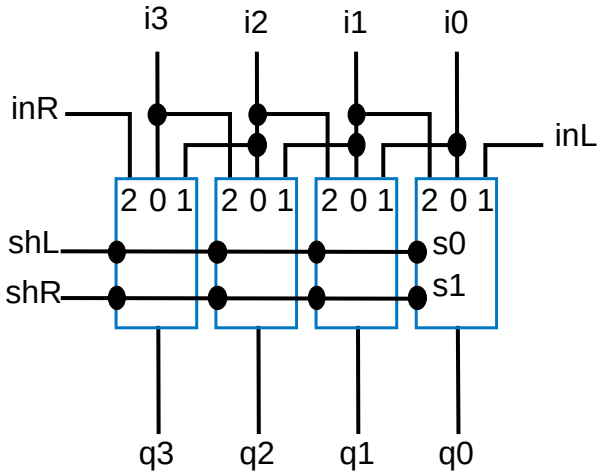
Somente esquerda



Esquerda ou Nenhum deslocamento



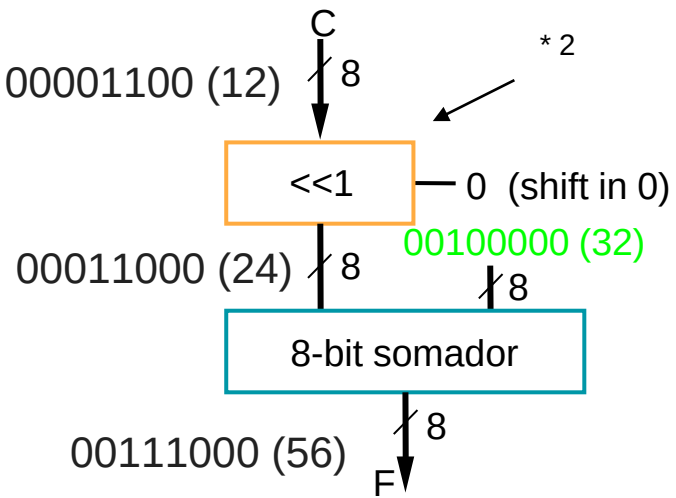
Esquerda, direita ou Nenhum deslocamento



Exemplo de caminho de dados com deslocador e somador

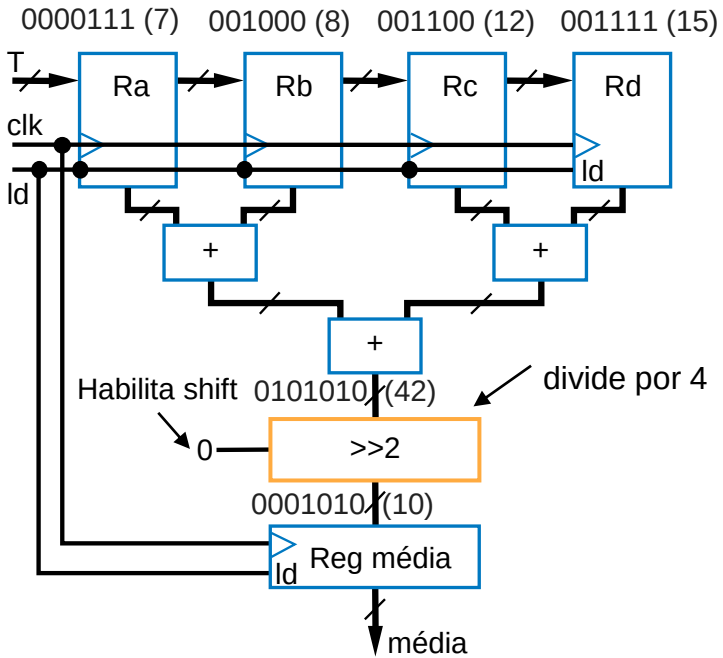
Conversão de temperatura de Celsius para Fahrenheit (8 bits)

- F = C * 9/5 + 32
- Aproximação: F = C*2 + 32
- Multiplicação por 2 implementada com deslocamento



Exemplo de caminho de dados com deslocador e somador

Média de quatro valores de temperatura (ou qualquer outra medição)

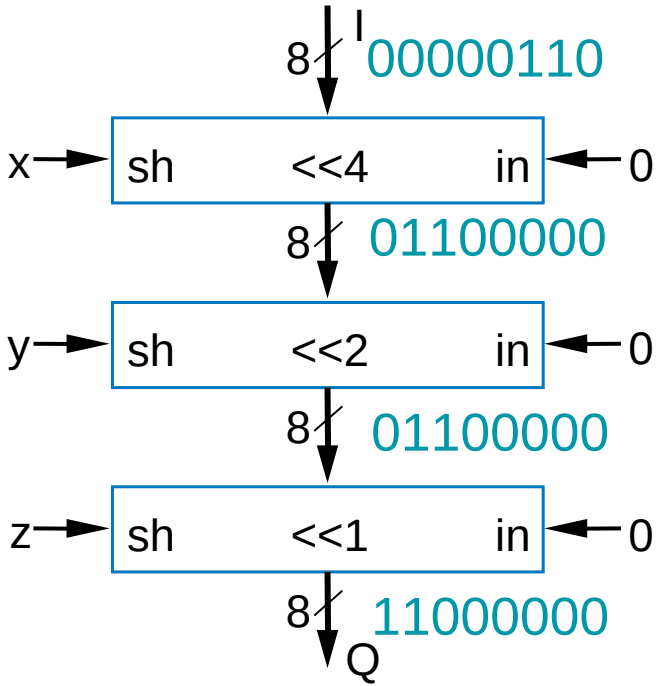


Deslocador configurável (barrel shifter)

Deslocador que pode ser configurável em 0,1,..., 7

Implementação mais eficiente

- Associação de deslocadores

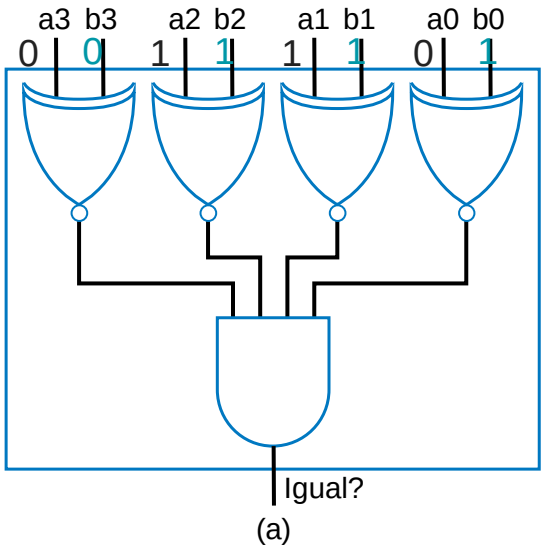


Comparadores

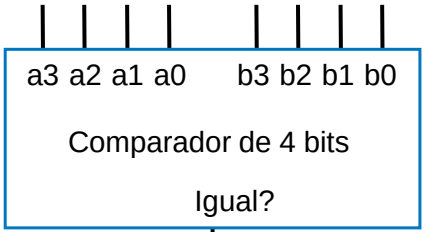
Comparador de igualdade de valores

- Saída é 1 se os valores são igual e 0 se diferentes

0110 = 0111 ?



(a)



(b)

Comparadores de magnitude

Indicar se A>B ou A=B (4 bits)

- Algoritmo: Compara bit a bit a partir do bit mais significativo. Logo, A3 e B3, A2 e B2, ..., para quando a comparação der diferente, o operando que tiver bit 1 na parada da comparação será o maior, se não parar até o último dos bits dos operandos, então, são iguais.

A=1011 B=1001

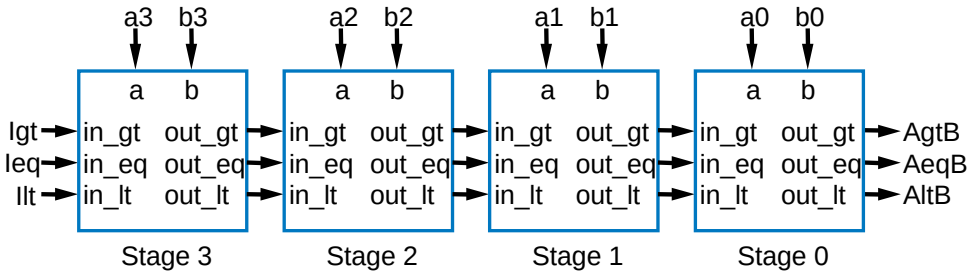
1011	**1**001	Iguais
1**0**11	1**0**01	Iguais
10**1**1	10**0**1	**Diferentes**

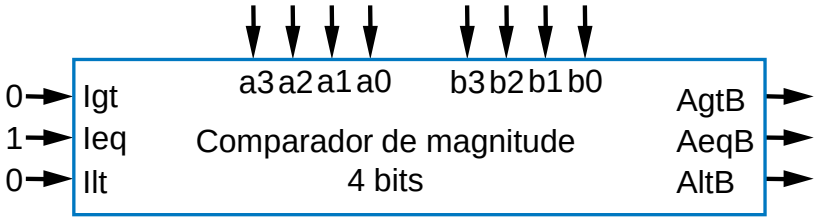
Logo, A > B

Comparadores de magnitude

Indicar se A>B ou A=B (4 bits)

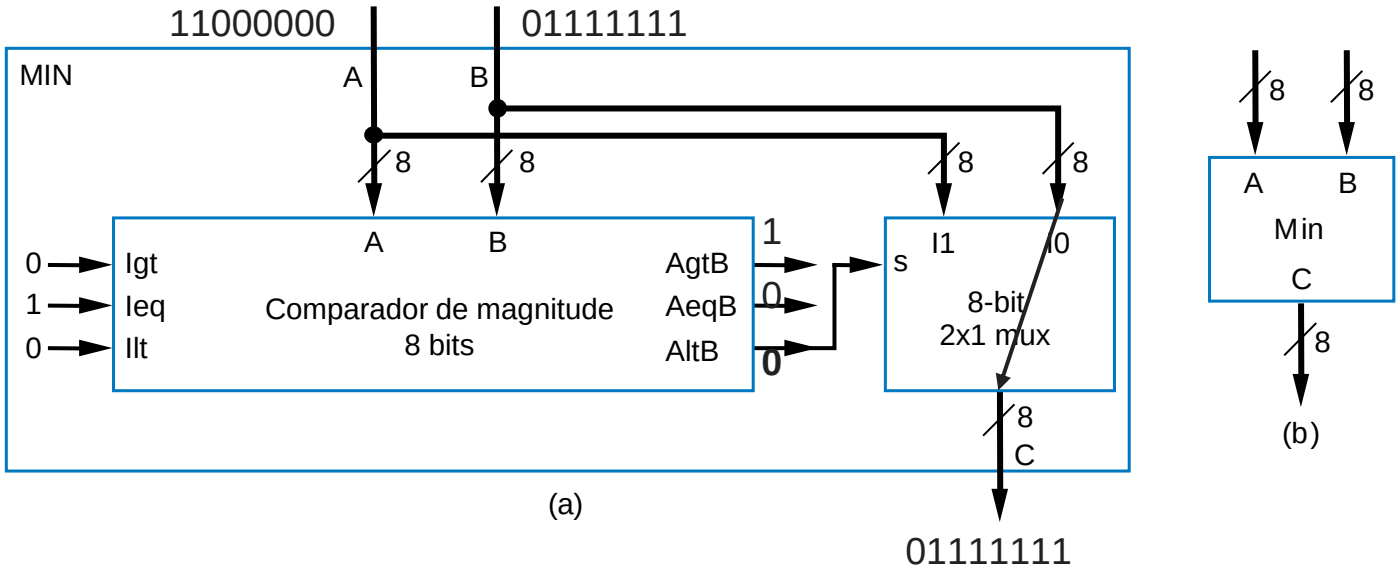
- Algoritmo: Compara bit a bit a partir do bit mais significativo. Logo, A3 e B3, A2 e B2, ..., para quando a comparação der diferente, o operando que tiver bit 1 na parada da comparação será o maior, se não parar até o último dos bits dos operandos, então, são iguais.





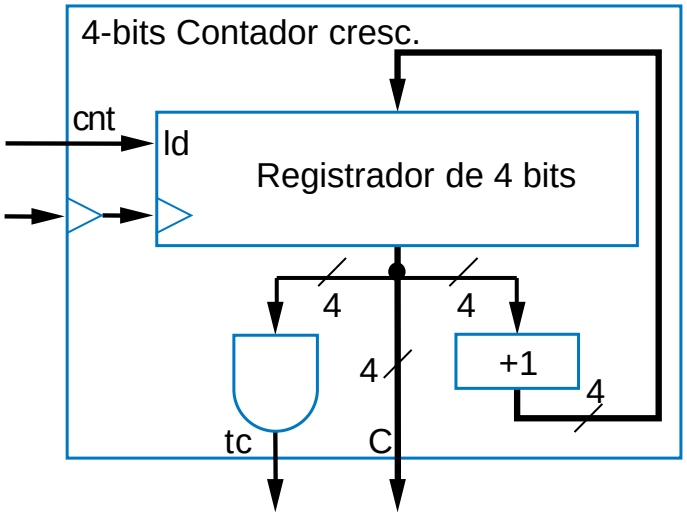
Comparadores de magnitude

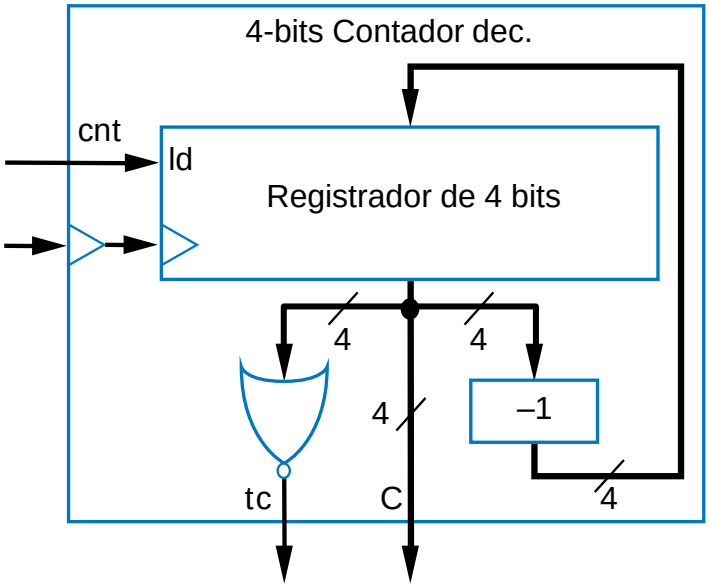
Exemplo de uso de comparadores de magnitude: direcionar à saída o menor entre dois números



Contadores

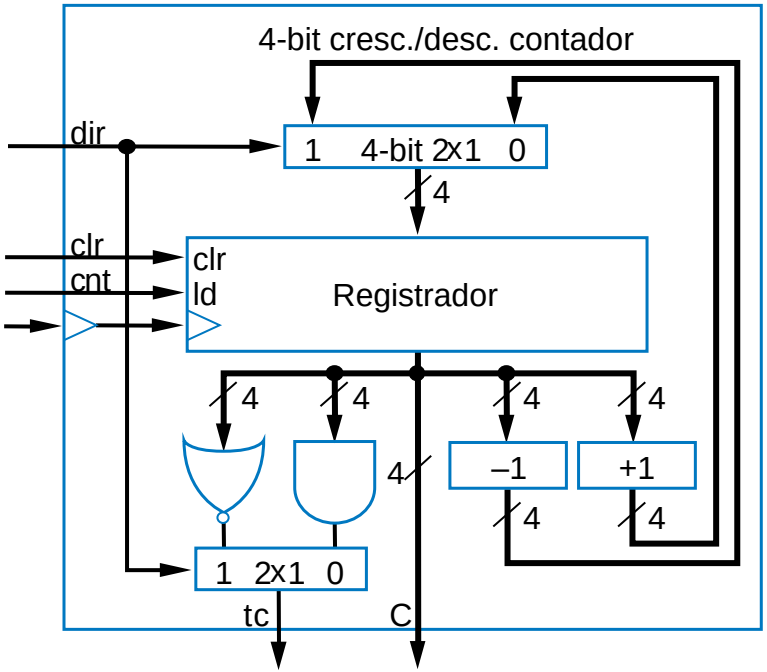
Incremento ou decremento de 1 a cada ciclo de clock

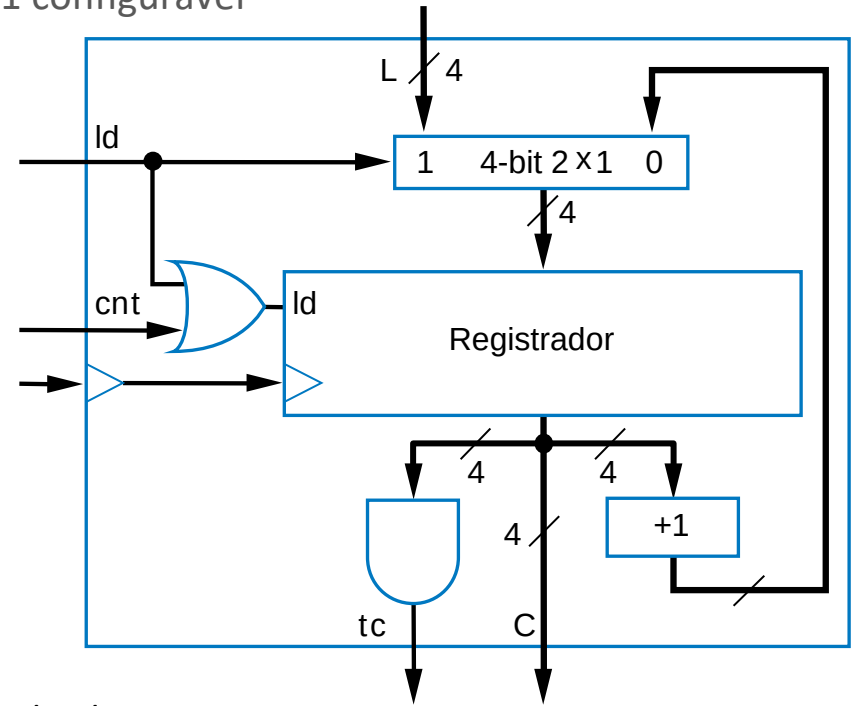




Contadores

Contador com incremento ou decremento de 1 configurável

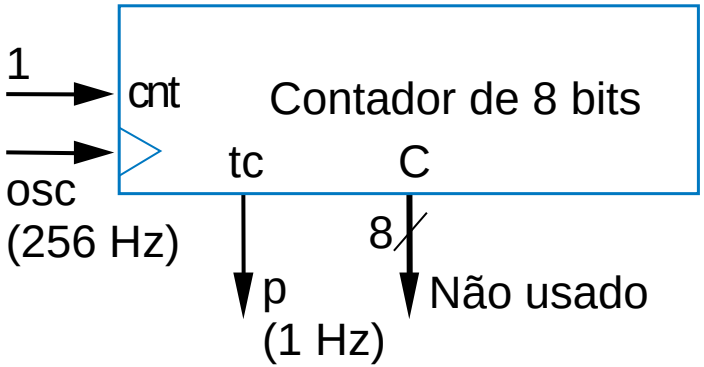




Contadores

Exemplo de aplicação: divisor de clock com contador

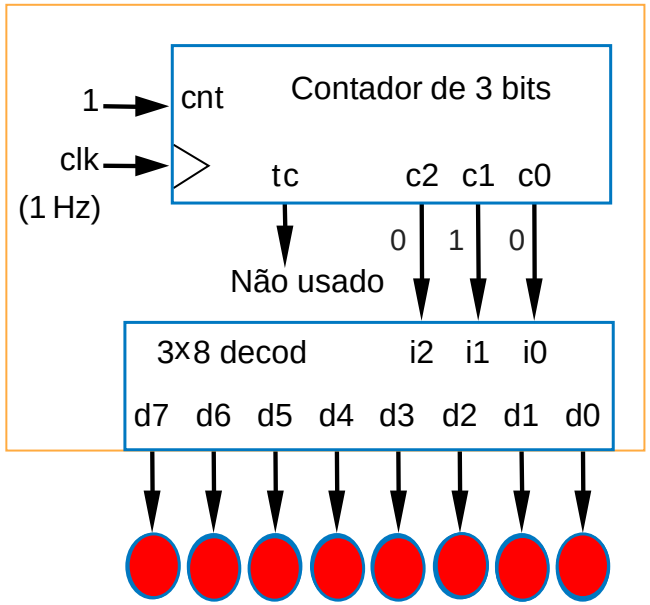
- Suponha que o clock do nosso sistema digital é de 256 Hz, e precisamos de um pulso de 1 Hz
- Utilização de um contador de 8 bits
- Contagem de 0 a 255 (ou 255 a 0) e gera um pulso em tc a cada segundo



Contadores

Exemplo de aplicação: painel de LEDs

- Deve-se iluminar um painel com 8 LEDs da direita para esquerda um de cada vez
- Contador de 3 bits



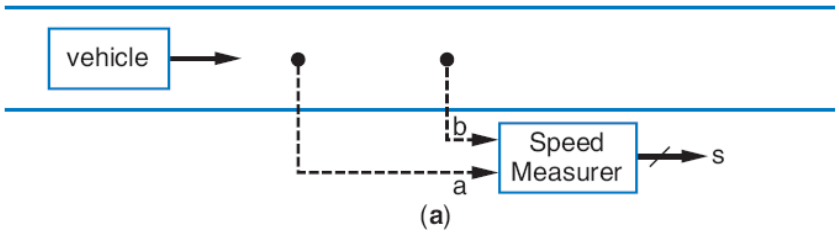
Temporizadores

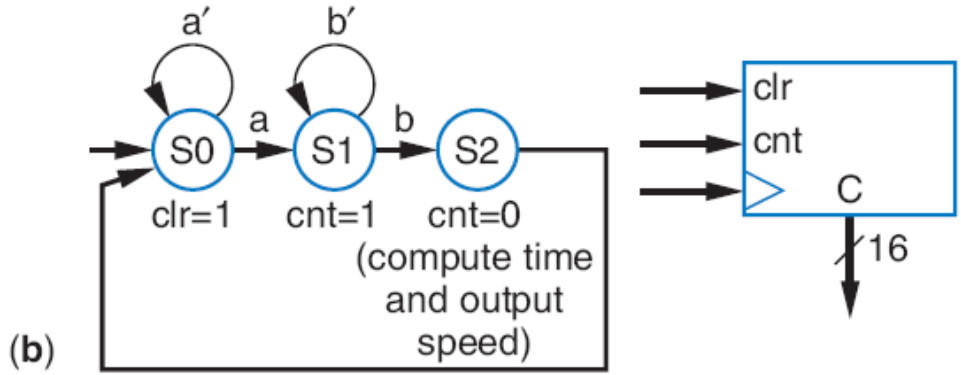
Um tipo de contador que pode ser usado para contar tempo

- Sabendo a frequência do clock, é possível medir a passagem do tempo

Exemplo: Medição de velocidade usando dois sensores de presença

- Sensor “a” inicia a contagem e o sensor “b” para a contagem





Unidades Lógicas Aritméticas (ULA)

Juntando todos os blocos básicos vistos anteriormente

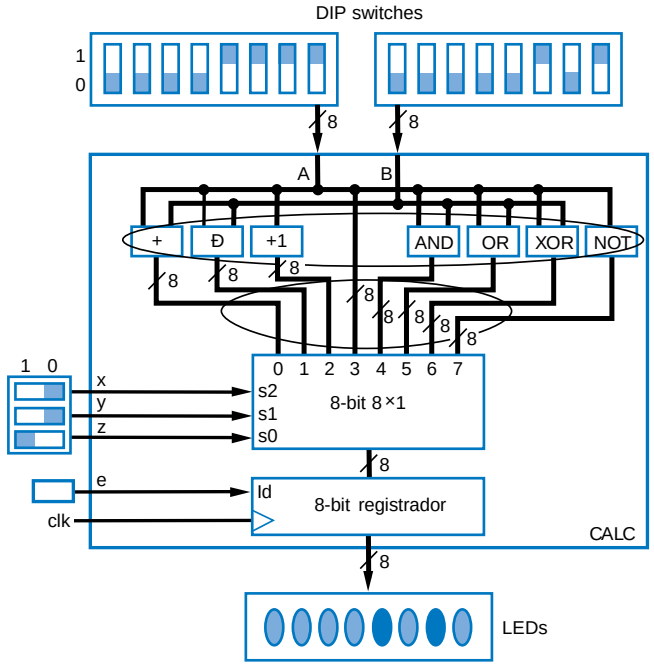
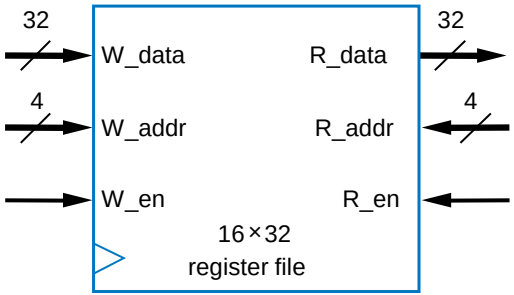


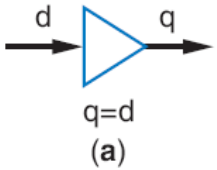
TABLE 4.2 Desired calculator operations

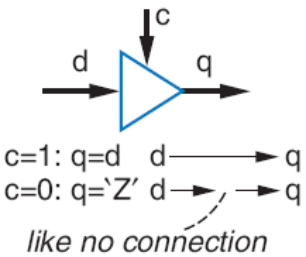
Inputs			Operation	Sample output if A=00001111, B=00000101
x	y	z		
0	0	0	S = A + B	S=00010100
0	0	1	S = A - B	S=00001010
0	1	0	S = A + 1	S=00010000
0	1	1	S = A	S=00001111
1	0	0	S = A AND B (bitwise AND)	S=00000101
1	0	1	S = A OR B (bitwise OR)	S=00001111
1	1	0	S = A XOR B (bitwise XOR)	S=00001010
1	1	1	S = NOT A (bitwise complement)	S=11110000

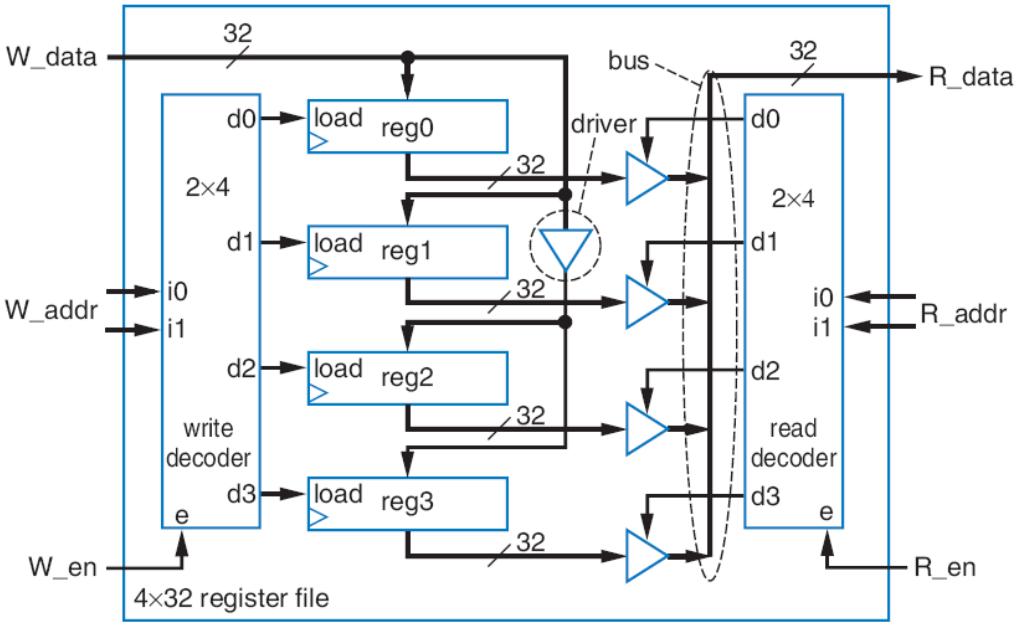
Banco de registradores

Componente que provê acesso a M registradores de N bits → Banco de registradores MxN



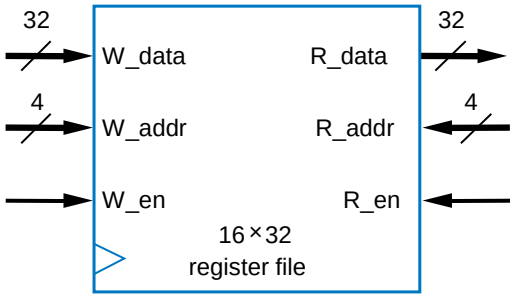


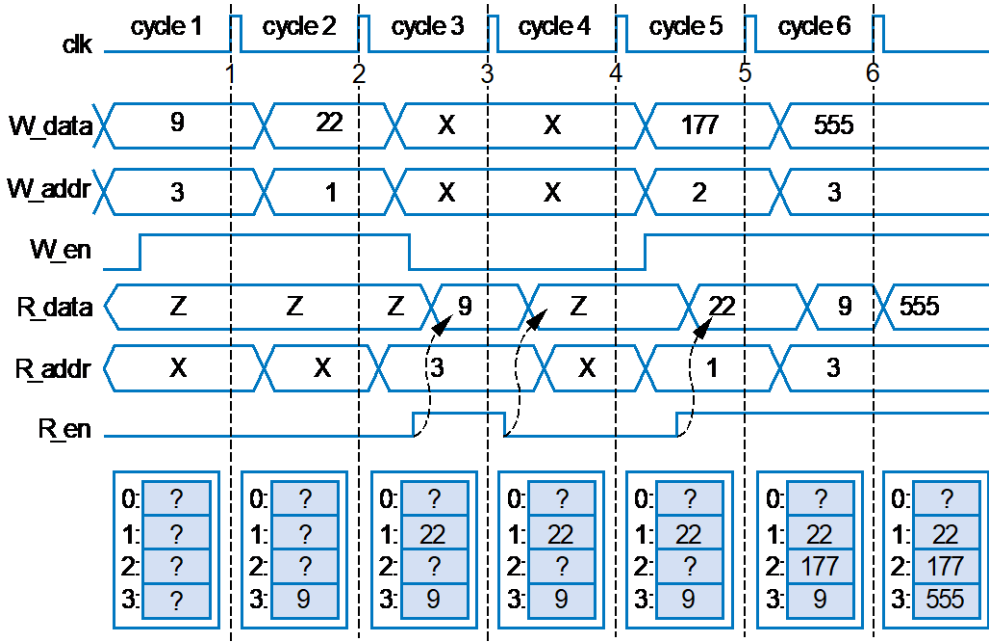




Banco de registradores

Componente que provê acesso a M registradores de N bits → Banco de registradores MxN





Resumo

Componentes comumente utilizados no nível RTL (não limitado aos vistos na aula)

- Registradores
- Deslocadores
- Somadores
- Comparadores
- Contadores
- Multiplicadores
- Subtratores
- Unidades lógicas aritméticos
- Banco de registradores

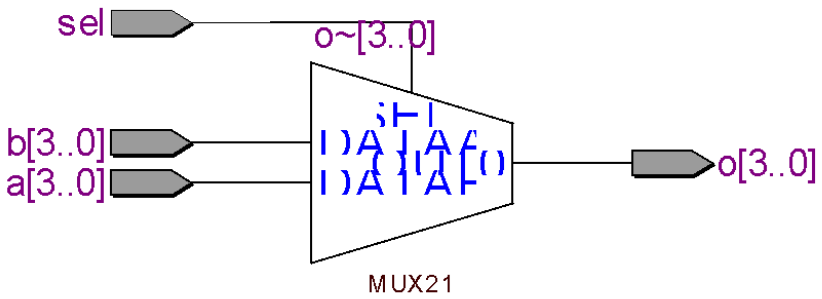
Vamos combinar todos esses blocos e máquinas de estados para construir sistemas mais complexos

Componentes de blocos operativos

Construção de caminhos de dados (datapaths) em VHDL

Componentes de blocos operativo

Inferência do Quartus II



Resumo de alguns blocos digitais básicos estudados

- Multiplexador ⇒ Circuito combinacional

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4  entity mux is
5     generic(
6        nbits : integer := 4
7     );
8     port(
9        a, b : in std_logic_vector(nbits-1 downto 0);
10       sel : in std_logic;
11       o : out std_logic_vector(nbits-1 downto 0)
12    );
13 end mux;
```

```
15 architecture comportamento of mux is
16    -- Somente utilizado na forma 4
17    signal sel_array : std_logic_vector(nbits-1 downto 0);
18 begin
23    -- Forma 1
24    process(a,b,sel)
25    begin
26       if sel = '1' then
27          o <= a;
28       else
29          o <= b;
30       end if;
31    end process;
32
33    -- Forma 2
34    -- o <= a when sel='1' else b;
35
36    --Forma 3
37 -- with sel select
38 --    o <= a when '1',
39 --         b when others;
```

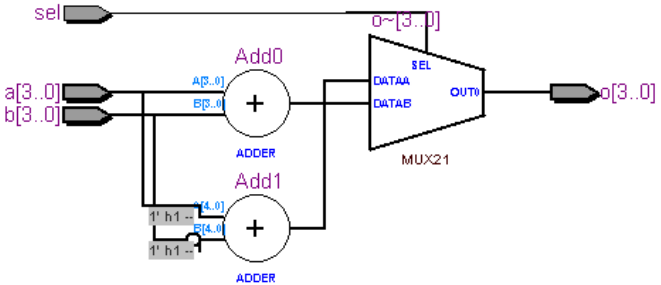
Várias implementações diferentes são interpretadas como o mesmo bloco digital
Arquivo disponível

Componentes de blocos operativos

Resumo de alguns blocos digitais básicos estudados

- Somador ou subtrator ⇒ Circuito combinacional

Inferência do Quartus II




```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.NUMERIC_STD.all;
4
5  entity soma_sub_unsig is
6     generic(
7        nbits : integer := 4
8     );
9     port(
10       a, b : in std_logic_vector(nbits-1 downto 0);
11       sel : in std_logic;
12       o : out std_logic_vector(nbits-1 downto 0)
13    );
14 end soma_sub_unsig;
15
```

```
16 architecture comportamento of soma_sub_unsig is
17 begin
18
19    process(a,b,sel)
20    begin
21       if sel='1' then
22          o <= std_logic_vector(unsigned(a)+unsigned(b));
23       else
24          o <= std_logic_vector(unsigned(a)-unsigned(b));
25       end if;
26    end process;
27
28 end architecture;
```

Várias implementações diferentes são interpretadas como o mesmo bloco digital
Arquivo disponível

Somador e subtrator podem ser implementados separadamente e então conectados

Componentes de blocos operativos

Resumo de alguns blocos digitais básicos estudados

- Somador ou subtrator ⇒ Circuito combinacional **estrutural**

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

entity soma_unsig is
   generic(
      nbits : integer := 4
   );
   port(
      a, b : in std_logic_vector(nbits-1 downto 0);
      o : out std_logic_vector(nbits-1 downto 0)
   );
end soma_unsig;

architecture comportamento of soma_unsig is
begin
   o <= std_logic_vector(unsigned(a)+unsigned(b));
end architecture;
```

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

entity sub_unsig is
   generic(
      nbits : integer := 4
   );
   port(
      a, b : in std_logic_vector(nbits-1 downto 0);
      o : out std_logic_vector(nbits-1 downto 0)
   );
end sub_unsig;

architecture comportamento of sub_unsig is
begin
   o <= std_logic_vector(unsigned(a)-unsigned(b));
end architecture;
```

Componentes de blocos operativos

Resumo de alguns blocos digitais básicos estudados

- Somador ou subtrator ⇒ Circuito combinacional **estrutu**

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.NUMERIC_STD.all;
4
5  entity soma_sub_unsig_estrutural is
6     generic(
7        nbits : integer := 4
8     );
9     port(
10       a, b : in std_logic_vector(nbits-1 downto 0);
11       sel : in std_logic;
12       o : out std_logic_vector(nbits-1 downto 0)
13    );
14 end soma_sub_unsig_estrutural;
```

Declaração do componentes

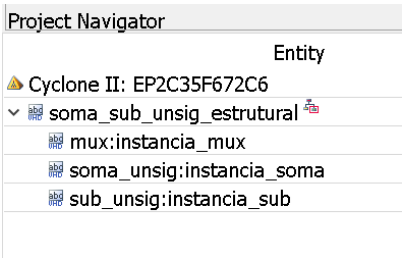
```
16 architecture estrutura of soma_sub_unsig_estrutural is
17    component sub_unsig
18    generic(
19       nbits : integer := 4
20    );
21    port(
22       a, b : in std_logic_vector(nbits-1 downto 0);
23       o : out std_logic_vector(nbits-1 downto 0)
24    );
25    end component;
26
27    component soma_unsig
28    generic(
29       nbits : integer := 4
30    );
31    port(
32       a, b : in std_logic_vector(nbits-1 downto 0);
33       o : out std_logic_vector(nbits-1 downto 0)
34    );
35    end component;
36
37    component mux
38    generic(
39       nbits : integer := 4
40    );
41    port(
42       a, b : in std_logic_vector(nbits-1 downto 0);
43       sel : in std_logic;
44       o : out std_logic_vector(nbits-1 downto 0)
45    );
46    end component;
```

Componentes de blocos operativos

Resumo de alguns blocos digitais básicos estudados

- Somador ou subtrator ⇒ Circuito combinacional **estrut**

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.NUMERIC_STD.all;
4
5  entity soma_sub_unsig_estrutural is
6     generic(
7        nbits : integer := 4
8     );
9     port(
10       a, b : in std_logic_vector(nbits-1 downto 0);
11       sel : in std_logic;
12       o : out std_logic_vector(nbits-1 downto 0)
13    );
14 end soma_sub_unsig_estrutural;
```



Instanciação e conexão dos componentes

```
48    signal soma, sub : std_logic_vector(nbits-1 downto 0);
49 begin
50    instancia_sub : sub_unsig
51    generic map(
52       nbits => nbits
53    )
54    port map(
55       a => a,
56       b => b,
57       o => sub
58    );
59
60    instancia_soma : soma_unsig
61    generic map(
62       nbits => nbits
63    )
64    port map(
65       a => a,
66       b => b,
67       o => soma
68    );
69
70    instancia_mux : mux
71    generic map(
72       nbits => nbits
73    )
74    port map(
75       a => soma,
76       b => sub,
77       sel => sel,
78       o => o
79    );
80 end architecture;
```

Componentes de blocos operativos

Resumo de alguns blocos digitais básicos estudados

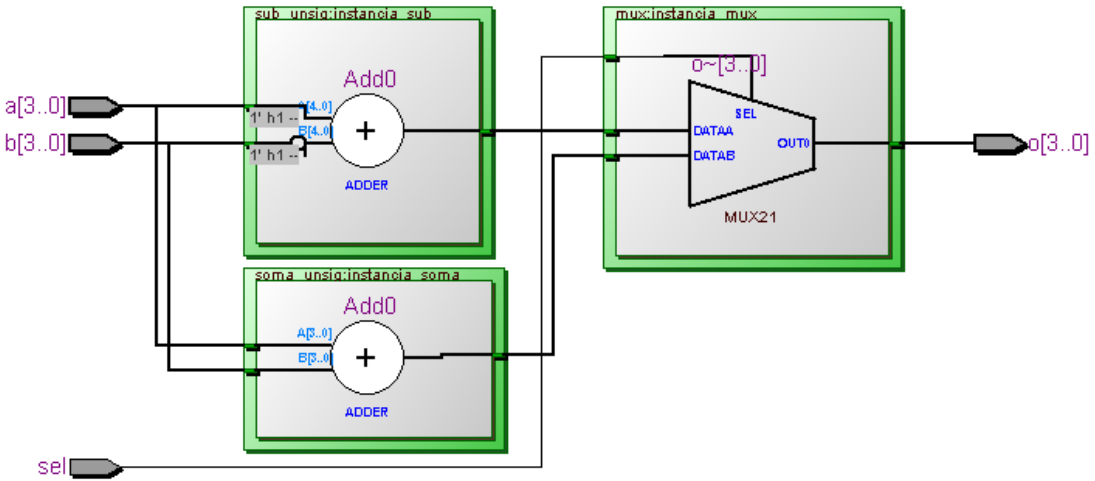
- Somador ou subtrator ⇒ Circuito combinacional **estrut**

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.a
3  use IEEE.NUMERIC_STD.all;
4
5  entity soma_sub_unsig_est
6     generic(
7        nbits : integer :=
8     );
9     port(
10       a, b : in std_logic
11       sel : in std_logic;
12       o : out std_logic_v
13    );
14 end soma_sub_unsig_estrut
```

Instanciação e conexão dos componentes

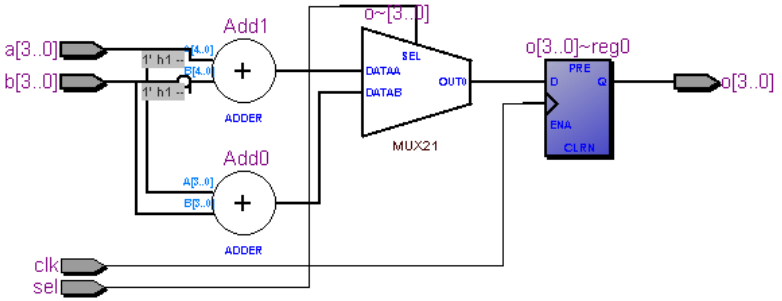
```
48    signal soma, sub : std_logic_vector(nbits-1 downto 0);
49 begin
50    instancia_sub : sub_unsig
51    generic map(
52       nbits => nbits
53    )
54    port map(
55       a => a,
56       b => b,
```

Inferência do Quartus II




```
80 end architecture;
```

Componentes de blocos operativos



Resumo de alguns blocos digitais básicos estudados

- Somador ou subtrator síncrono ⇒ Circuito combinacional e sequencial

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.NUMERIC_STD.all;
4
5  entity soma_sub_sinc is
6     generic(
7        nbits : integer := 4
8     );
9     port(
10       clk : in std_logic;
11       a, b : in std_logic_vector(nbits-1 downto 0);
12       sel : in std_logic;
13       o : out std_logic_vector(nbits-1 downto 0)
14    );
15 end soma_sub_sinc;
```

```
17 architecture comportamento of soma_sub_sinc is
18    signal soma, sub, mux : std_logic_vector(nbits-1 downto 0);
19 begin
20    -- Forma 1
21 -- process(clk)
22 -- begin
23 --    if rising_edge(clk) then
24 --       if sel='1' then
25 --          o <= std_logic_vector(unsigned(a)+unsigned(b));
26 --       else
27 --          o <= std_logic_vector(unsigned(a)-unsigned(b));
28 --       end if;
29 --    end if;
30 -- end process;
31
32    -- Forma 2
33    soma <= std_logic_vector(unsigned(a)+unsigned(b));
34    sub <= std_logic_vector(unsigned(a)-unsigned(b));
35    mux <= soma when sel = '1' else sub;
36
37    process(clk)
38    begin
39       if rising_edge(clk) then
40          o <= mux;
41       end if;
42    end process;
```

Várias implementações diferentes são interpretadas como o mesmo bloco digital
Arquivo disponível

Componentes de blocos operativos

Construção de caminhos de dados (datapaths) em VHDL

Componentes de blocos operativos

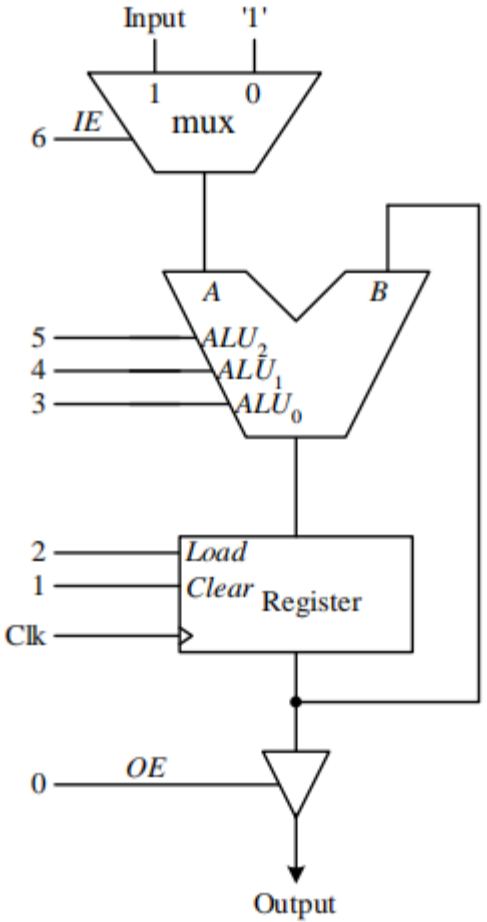
Construção de caminhos de dados (datapaths) em VHDL

- Definição: uma sequência de processamentos que podem ser executados em um ou mais dados
- Implementação básica em VHDL:
    - Encadeamento de vários processos (operações lógicas e aritméticas)
    - Possui certa configurabilidade

Componentes de blocos operativos

Construção de caminhos de dados (datapaths) em VHDL

- Definição: uma sequência de processamentos que podem ser executados em um ou mais dados
- Implementação básica em VHDL:
    - Encadeamento de vários processos (operações lógicas e aritméticas)
    - Possui certa configurabilidade

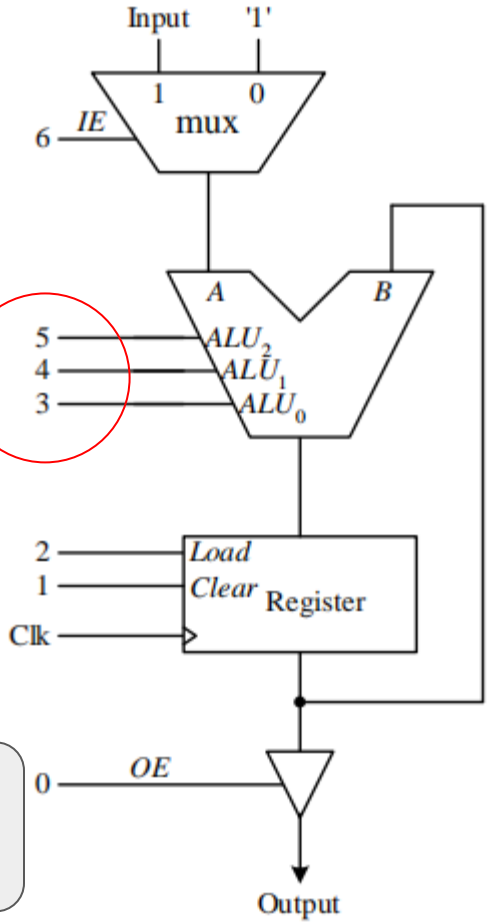


Componentes de blocos operativos

Construção de caminhos de dados (datapaths) em VHDL

- Definição: uma sequência de processamentos que podem ser executados em um ou mais dados
- Implementação básica em VHDL:
  - Encadeamento de vários processos (operações lógicas e aritméticas)
  - Possui certa configurabilidade

ALU_2	ALU_1	ALU_0	Operation
0	0	0	Pass through A
0	0	1	A AND B
0	1	0	A OR B
0	1	1	NOT A
1	0	0	$A + B$
1	0	1	$A - B$
1	1	0	$A + 1$
1	1	1	$A - 1$



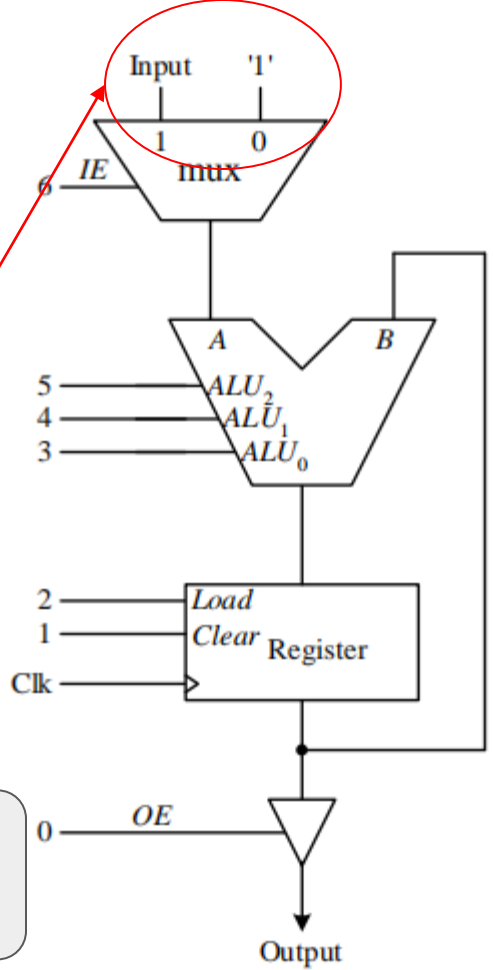
Várias operações podem ser executadas de acordo com as combinação dos bits de controle

Componentes de blocos operativos

Construção de caminhos de dados (datapaths) em VHDL

- Definição: uma sequência de processamentos que podem ser executados em um ou mais dados
- Implementação básica em VHDL:
    - Encadeamento de vários processos (operações lógicas e aritméticas)
    - Possui certa configurabilidade

ALU_2	ALU_1	ALU_0	Operation
0	0	0	Pass through A
0	0	1	A AND B
0	1	0	A OR B
0	1	1	NOT A
1	0	0	$A + B$
1	0	1	$A - B$
1	1	0	$A + 1$
1	1	1	$A - 1$



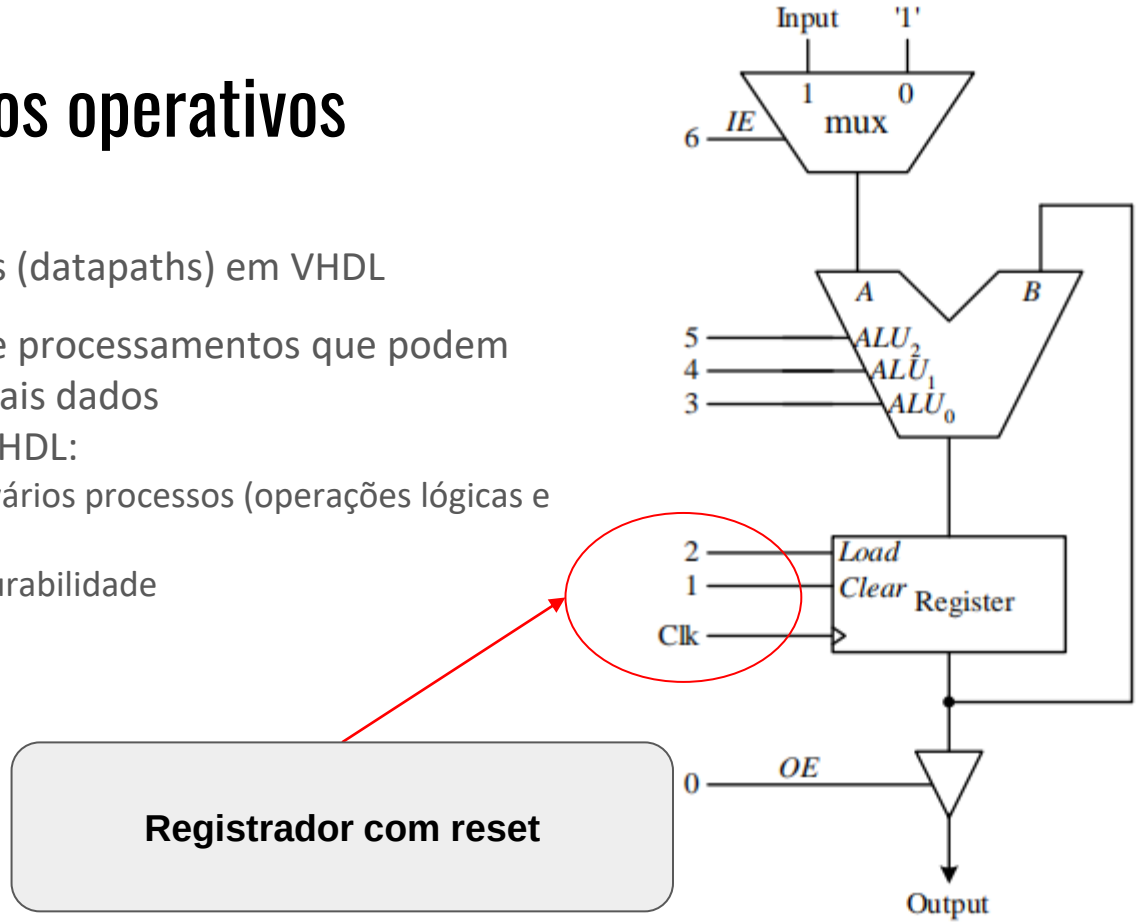
Um dado de entrada ou uma constante

Componentes de blocos operativos

Construção de caminhos de dados (datapaths) em VHDL

- Definição: uma sequência de processamentos que podem ser executados em um ou mais dados
- Implementação básica em VHDL:
    - Encadeamento de vários processos (operações lógicas e aritméticas)
    - Possui certa configurabilidade

ALU_2	ALU_1	ALU_0	Operation
0	0	0	Pass through A
0	0	1	A AND B
0	1	0	A OR B
0	1	1	NOT A
1	0	0	$A + B$
1	0	1	$A - B$
1	1	0	$A + 1$
1	1	1	$A - 1$

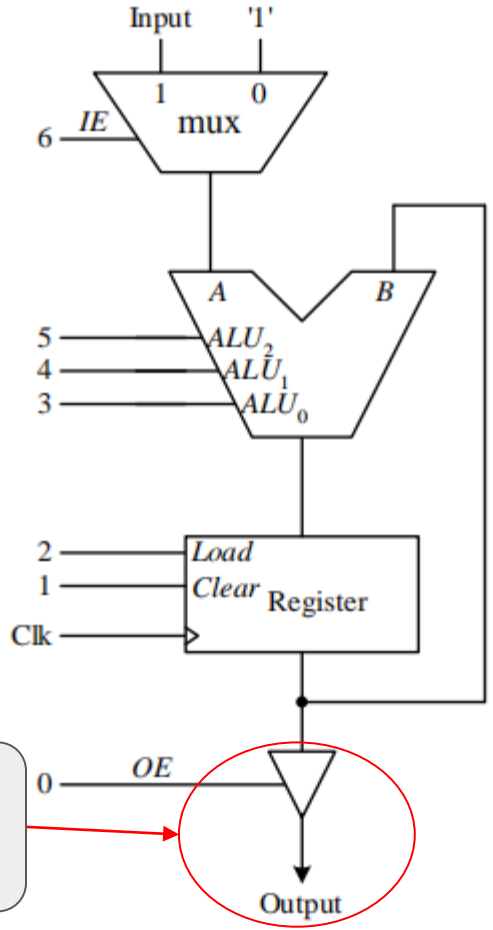


Componentes de blocos operativos

Construção de caminhos de dados (datapaths) em VHDL

- Definição: uma sequência de processamentos que podem ser executados em um ou mais dados
- Implementação básica em VHDL:
    - Encadeamento de vários processos (operações lógicas e aritméticas)
    - Possui certa configurabilidade

ALU_2	ALU_1	ALU_0	Operation
0	0	0	Pass through A
0	0	1	A AND B
0	1	0	A OR B
0	1	1	NOT A
1	0	0	$A + B$
1	0	1	$A - B$
1	1	0	$A + 1$
1	1	1	$A - 1$



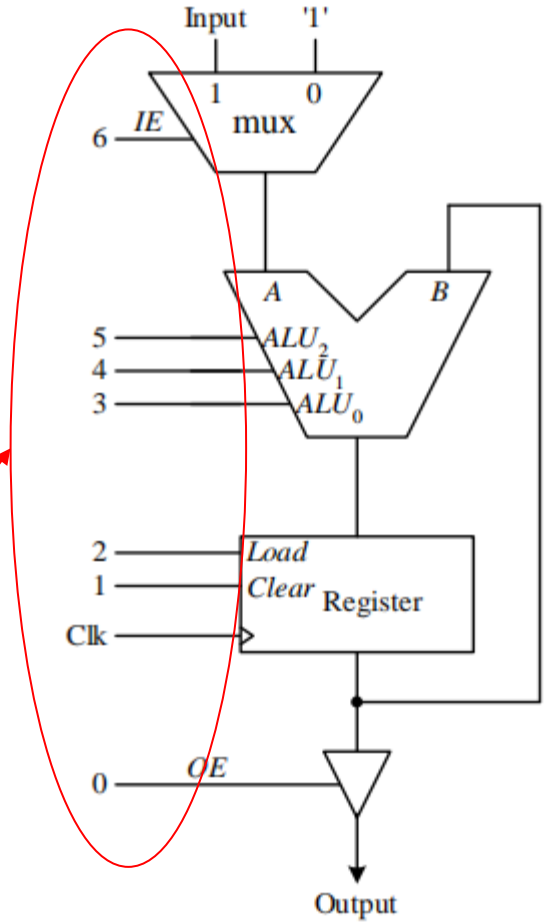
Interface com a saída

Componentes de blocos operativos

Construção de caminhos de dados (datapaths) em VHDL

- Definição: uma sequência de processamentos que podem ser executados em um ou mais dados
- Implementação básica em VHDL:
  - Encadeamento de vários processos (operações lógicas e aritméticas)
  - Possui certa configurabilidade

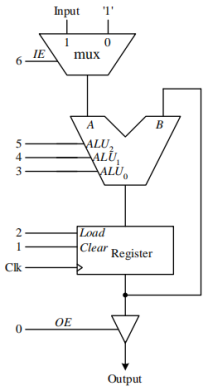
Sinais de controle fornecidos por uma entidade externa



Componentes de blocos operativos

Construção de caminhos de dados (datapaths) em VHDL

- Definição: uma sequência de processamentos que podem ser executados em um ou mais dados
- Implementação básica em VHDL:
  - Encadeamento de vários processos (operações lógicas e aritméticas)




```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity datapath is
   generic(
      nbits : integer := 4
   );
   port(
      clk : in std_logic;
      ie : in std_logic;
      alu : in std_logic_vector(2 downto 0);
      reg_load : in std_logic;
      reg_clear : in std_logic;
      input : in std_logic_vector(nbits-1 downto 0);
      output : out std_logic_vector(nbits-1 downto 0)
   );
end datapath;
```

```vhdl
begin
   -- Multiplexador
   mux_output <= input when ie='1' else UM;

   -- ALU
   process(mux_output, reg_output, alu)
   begin
      case alu is
         when "000" =>
            alu_output <= mux_output;
         when "001" =>
            alu_output <= mux_output and reg_output;
         when "010" =>
            alu_output <= mux_output or reg_output;
         when "011" =>
            alu_output <= not mux_output;
         when "100" =>
            alu_output <=
            std_logic_vector(signed(mux_output) + signed(reg_output));
         when "101" =>
            alu_output <=
            std_logic_vector(signed(mux_output) - signed(reg_output));
         when "110" =>
            alu_output <=
            std_logic_vector(signed(mux_output) + to_signed(1, nbits));
         when "111" =>
            alu_output <=
            std_logic_vector(signed(mux_output) - to_signed(1, nbits));
         when others =>
            alu_output <= (others => '0');
      end case;
   end process;
```

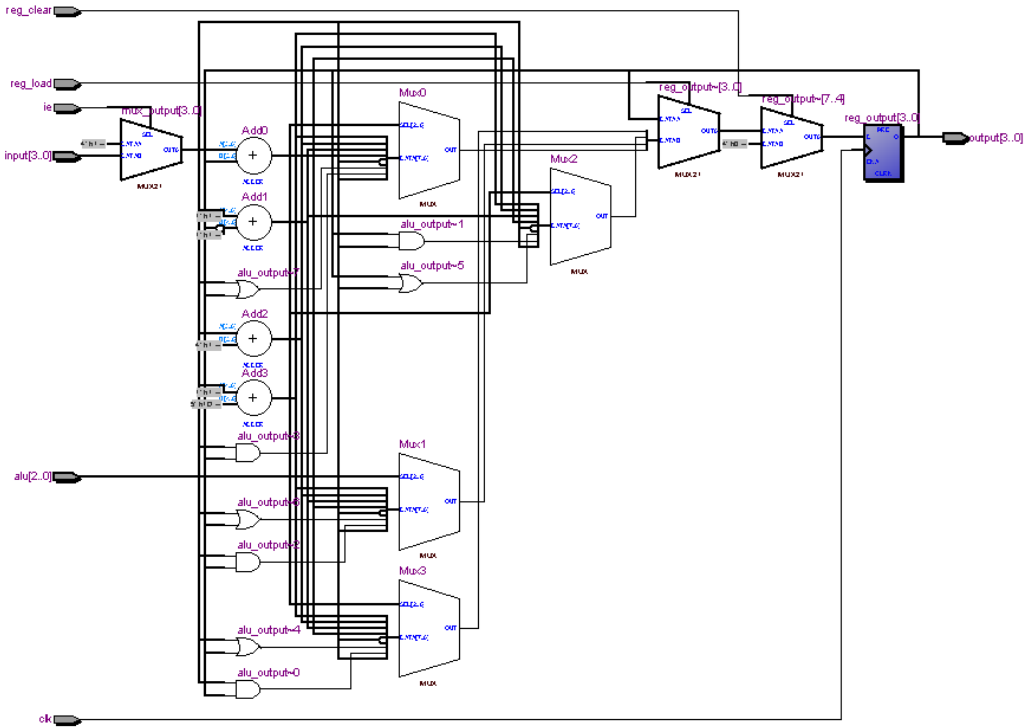
```vhdl
   -- Registrador
   process(clk)
   begin
      if rising_edge(clk) then
         if reg_clear = '1' then
            reg_output <= (others => '0');
         elsif reg_load = '1' then
            reg_output <= alu_output;
         end if;
      end if;
   end process;

   -- Atribuição da saída
   output <= reg_output;

end comportamento;
```

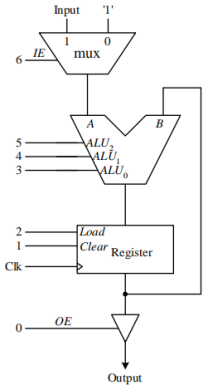
Componentes de blocos operativos

Construção de camir

- Definição: uma
  ser executados
- Implementaçã
  - Enca
    aritn

Inferência do Quartus II






```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity datapath is
6    generic(
7        nbits : integer := 4
8    );
9    port(
10       clk : in std_logic;
11       ie : in std_logic;
12       alu : in std_logic_vector(2 downto 0);
13       reg_load : in std_logic;
14       reg_clear : in std_logic;
15       input : in std_logic_vector(nbits-1 downto 0);
16       output : out std_logic_vector(nbits-1 downto 0)
17   );
```

```
ito of datapath is
gic_vector(nbits-1 downto 0)
r(to_signed(1, nbits));

td_logic_vector(nbits-1 downto 0);

en ie='1' else UM;

g_output, alu)

= mux_output;
= mux_output and reg_output;
= mux_output or reg_output;
= not mux_output;
=
ctor(signed(mux_output) + signed(reg_output));
=
ctor(signed(mux_output) - signed(reg_output));
=
ctor(signed(mux_output) + to_signed(1, nbits));
=
ctor(signed(mux_output) - to_signed(1, nbits));
= (others => '0');

e(clk) then
ar = '1' then
put <= (others => '0');
load = '1' then
put <= alu_output;
```

```
1  l
2  u
3  u
4
5  e
6
7
8
9
10
11
12
13
14
15
16
17
18 end datapath;
```

```
73    output <= reg_output;
74
75 end comportamento;
```

Componentes de blocos operativos

Construção de caminhos de dados (datapaths) em VHDL

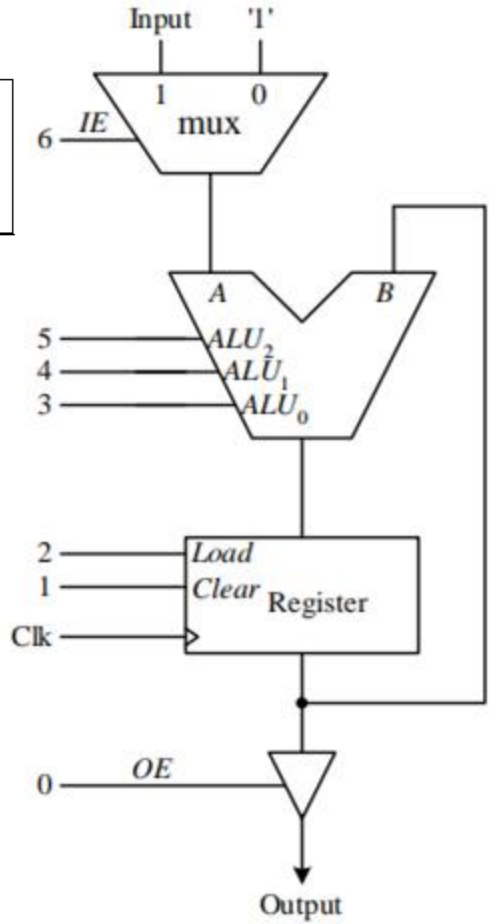
- Implementação de algoritmos com datapath

Componentes de blocos op

```
1          i = 0
2          while (i < 10){
3             i = i + 1
4             output i
5             }
```

Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath



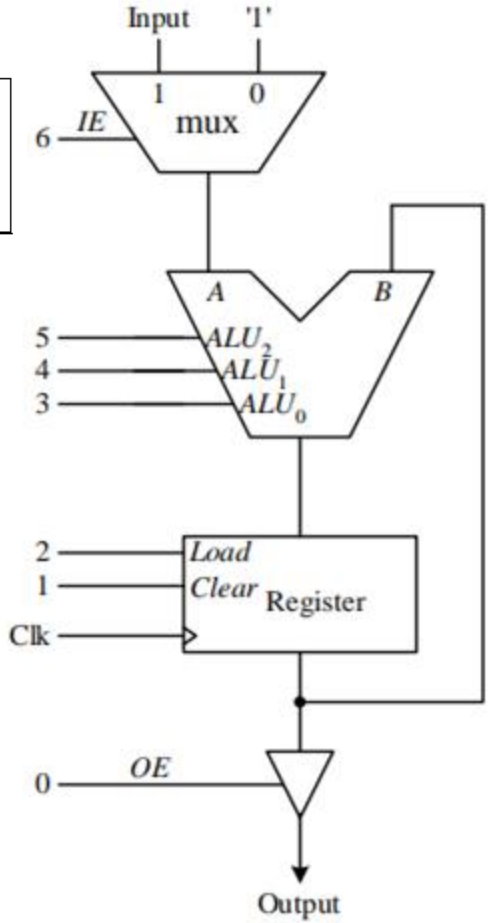
Componentes de blocos op

```
1          i = 0
2          while (i < 10){
3             i = i + 1
4             output i
5             }
```

Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath

Control Word	Instruction	IE	$ALU_2\ ALU_1\ ALU_0$	Load	Clear	OE
		6	5-3	2	1	0
1	$i = 0$	×	×××	0	1	0
2	$i = i + 1$	0	100 (add)	1	0	0
3	output i	×	×××	0	0	1



Componentes de blocos op

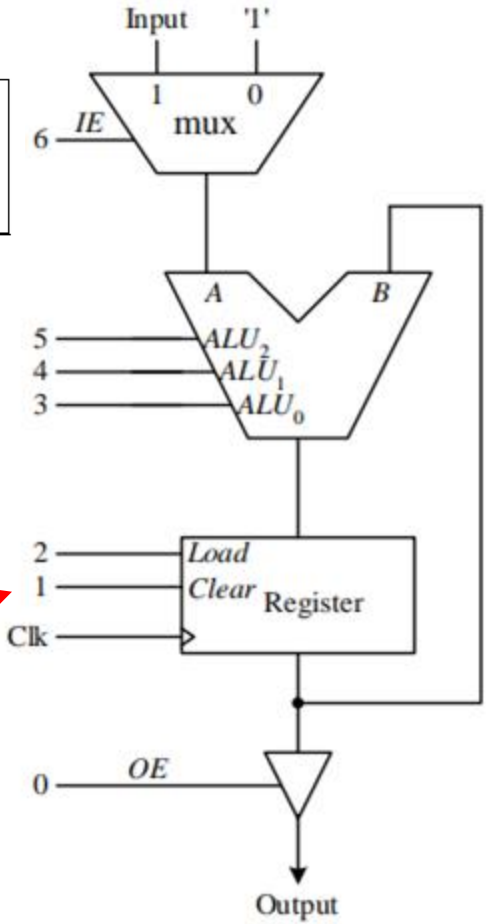
```
1        i = 0
2        while (i < 10){
3            i = i + 1
4            output i
5            }
```

Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath

Limpar o conteúdo do registrador
i = 0

Control Word	Instruction	IE	$ALU_2\ ALU_1\ ALU_0$	Load	Clear	OE
		6	5-3	2	1	0
1	$i = 0$	×	×××	0	1	0
2	$i = i + 1$	0	100 (add)	1	0	0
3	output i	×	×××	0	0	1



Componentes de blocos op

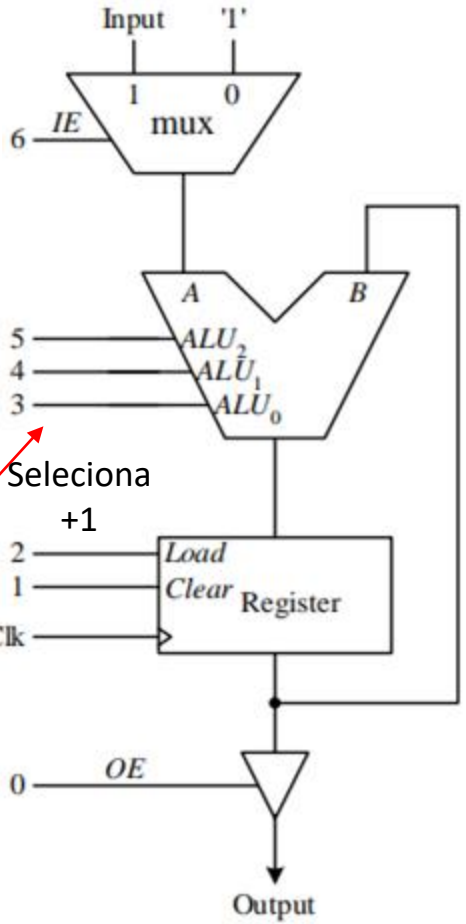
```
1          i = 0
2          while (i < 10){
3            i = i + 1
4            output i
5            }
```

Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath

Incremento de 1
i = i+1

Control Word	Instruction	IE	$ALU_2\ ALU_1\ ALU_0$	Load	Clear	OE
		6	5-3	2	1	0
1	$i = 0$	×	×××	0	1	0
2	$i = i + 1$	0	100 (add)	1	0	0
3	output i	×	×××	0	0	1



Componentes de blocos op

```
1          i = 0
2          while (i < 10){
3              i = i + 1
4              output i
5              }
```

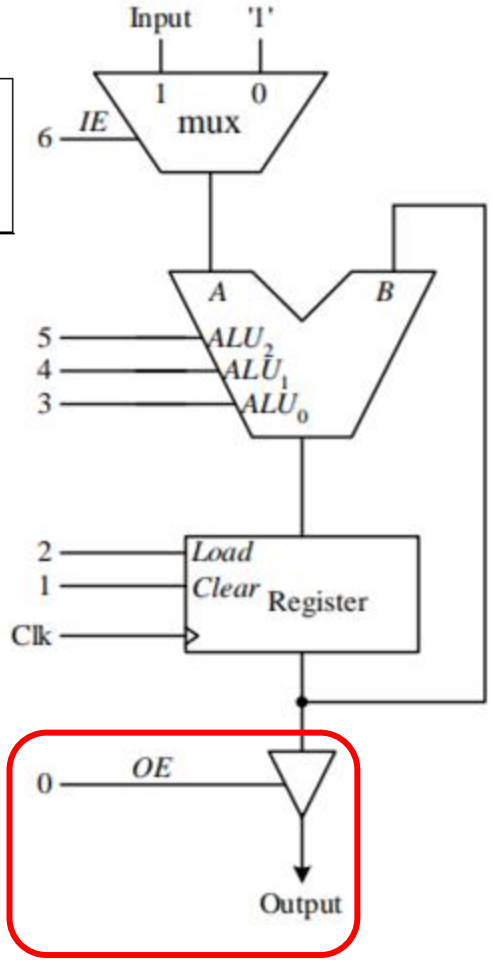
Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath

A saída deve ser monitorada por uma entidade externa para finalizar o processamento

Control Word	Instruction	IE	$ALU_2\ ALU_1\ ALU_0$	Load	Clear	OE
		6	5-3	2	1	0
1	$i = 0$	×	×××	0	1	0
2	$i = i + 1$	0	100 (add)	1	0	0
3	output i	×	×××	0	0	1

OE ⇒ output enable
Não será abordado na aula ⇒ podemos assumir que não está presente no circuito

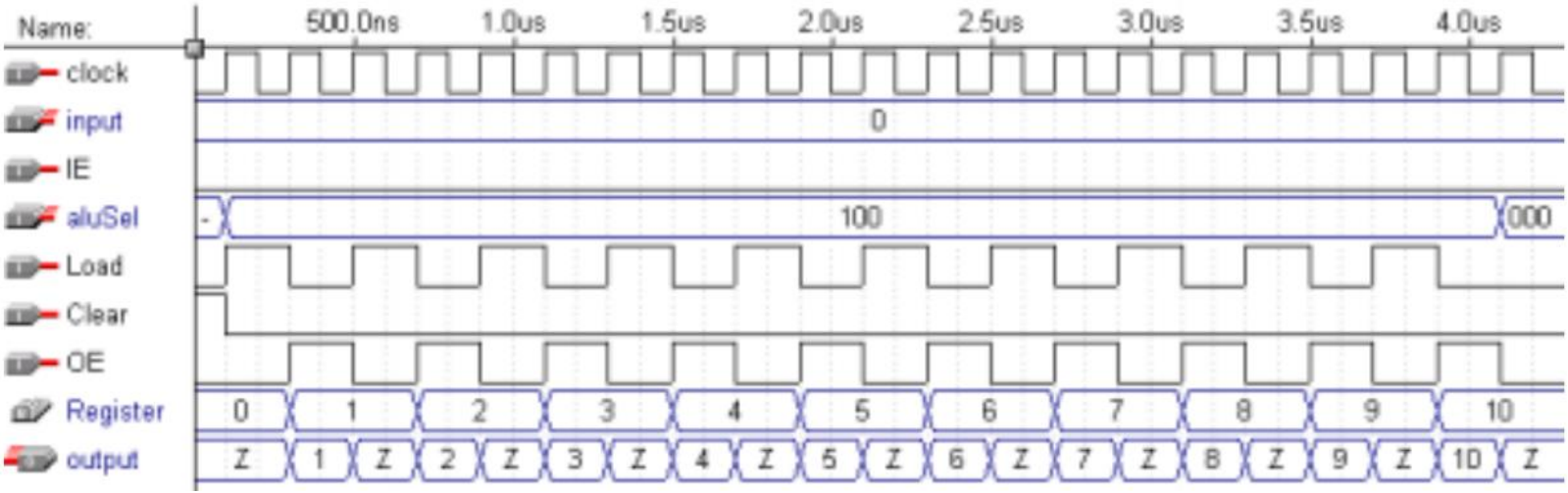


Componentes de blocos operativos

```
1          i = 0
2          while (i < 10){
3             i = i + 1
4             output i
5             }
```

Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath

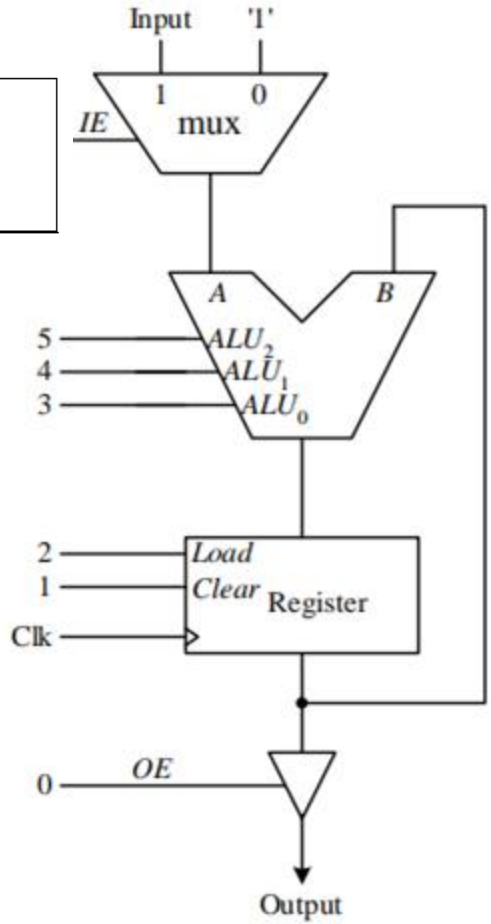


Componentes de blocos ope

```
1          i = 0
2          while (i < 10){
3             i = i + 1
4             output i
5             }
```

Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath
- Algoritmo mais complexo



i=0

While (i<10){

i = i+input

i = i+1

i = i AND 1

}

Componentes de blocos ope

```
1          i = 0
2          while (i < 10){
3             i = i + 1
4             output i
5             }
```

Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath
- Algoritmo mais complexo
    - **Passo 1:** Inicialização (1 ciclo de clock)

i=0

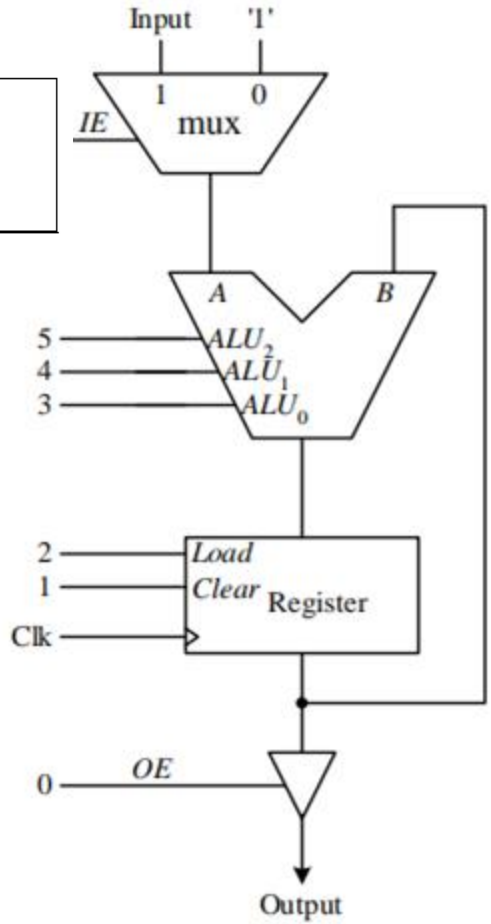
While (i<10){

i = i+input

i = i+1

i = i AND 1

}



Componentes de blocos ope

```
1          i = 0
2          while (i < 10){
3             i = i + 1
4             output i
5             }
```

Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath
- Algoritmo mais complexo
    - **Passo 2:** primeiro soma (1 ciclo de clock)

i=0

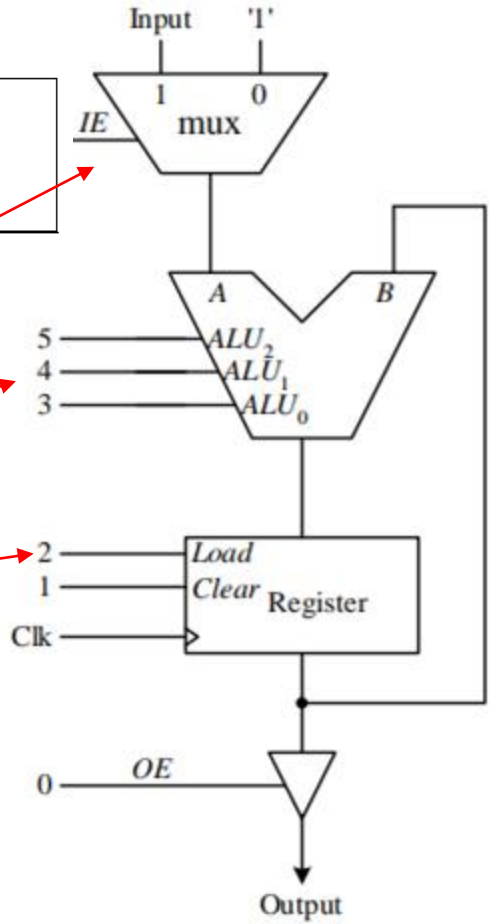
While (i<10){

i = i+input

i = i+1

i = i AND 1

}



Componentes de blocos ope

```
1          i = 0
2          while (i < 10){
3             i = i + 1
4             output i
5             }
```

Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath
- Algoritmo mais complexo
    - **Passo 3:** incremento de 1 (1 ciclo de clock)

i=0

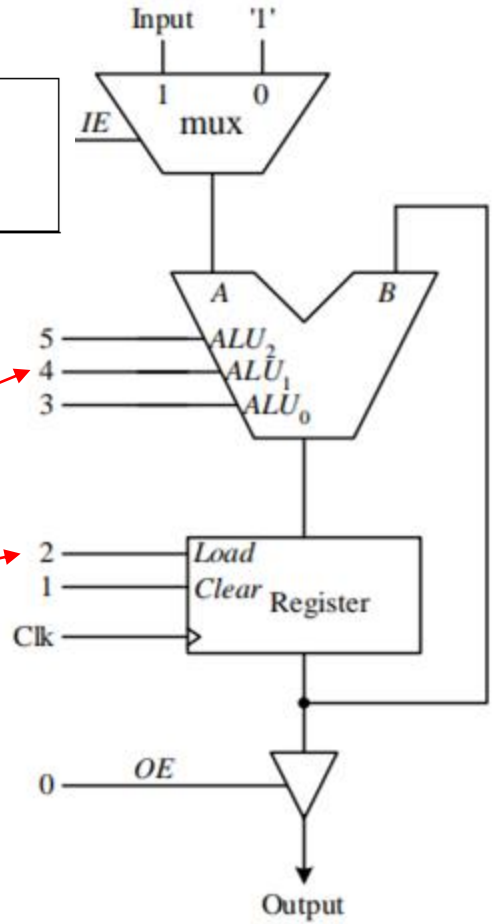
While (i<10){

i = i+input

i = i+1

i = i AND 1

}



Componentes de blocos ope

```
1          i = 0
2          while (i < 10){
3             i = i + 1
4             output i
5             }
```

Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath
- Algoritmo mais complexo
    - **Passo 4:** operação lógica (1 ciclo de clock)

i=0

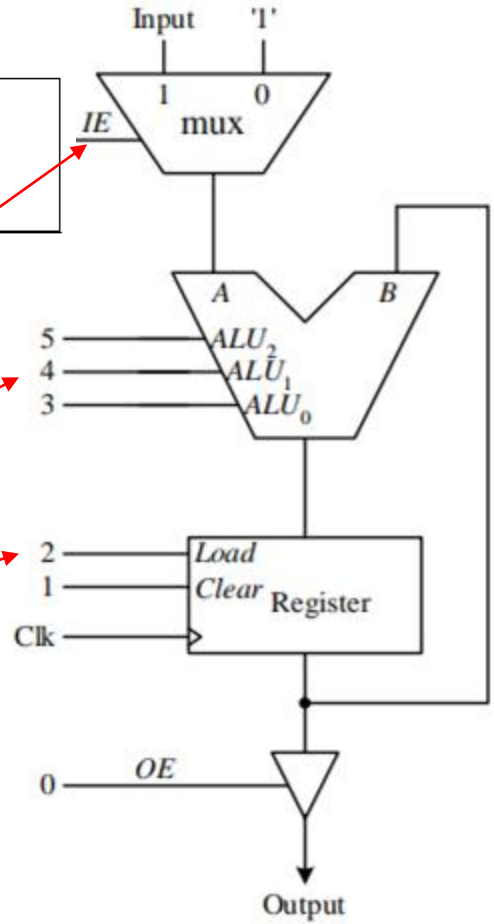
While (i<10){

i = i+input

i = i+1

i = i AND 1

}



Componentes de blocos ope

```
1          i = 0
2          while (i < 10){
3             i = i + 1
4             output i
5             }
```

Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath
- Algoritmo mais complexo

i=0

While (i<10){

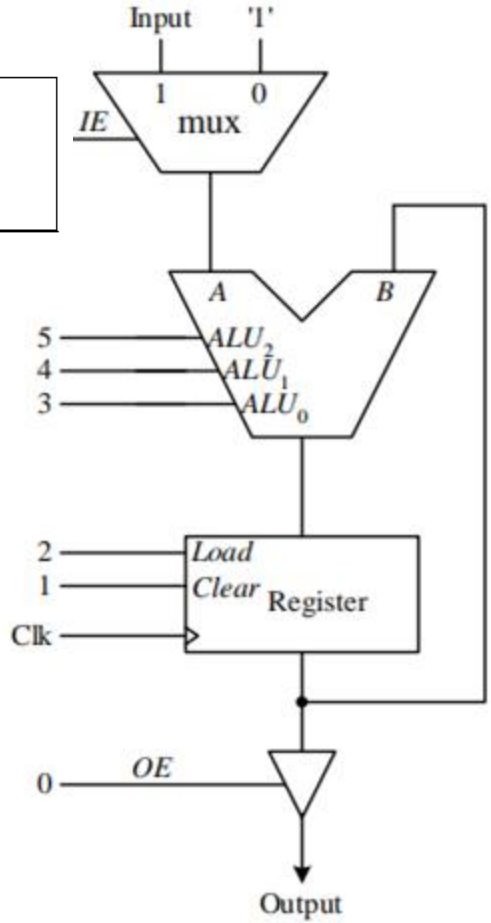
i = i+input

i = i+1

i = i AND 1

Cada uma deve ser feita em um ciclo de clock

}



Componentes de blocos ope

```
1          i = 0
2          while (i < 10){
3             i = i + 1
4             output i
5             }
```

Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath
- Algoritmo mais complexo

i=0

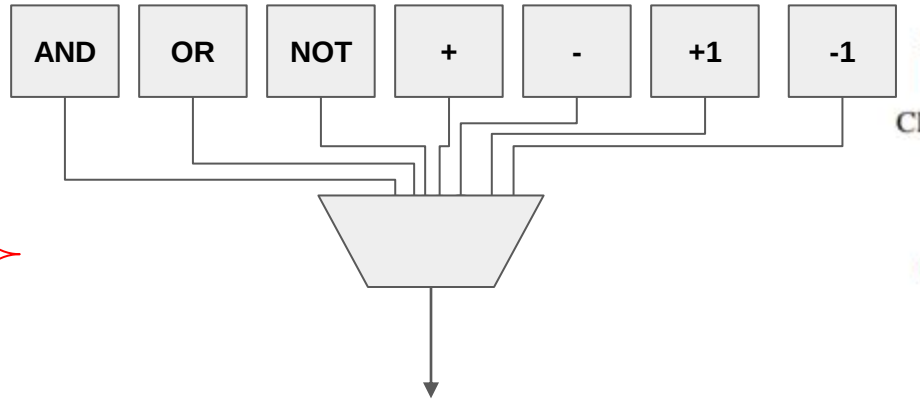
While (i<10){

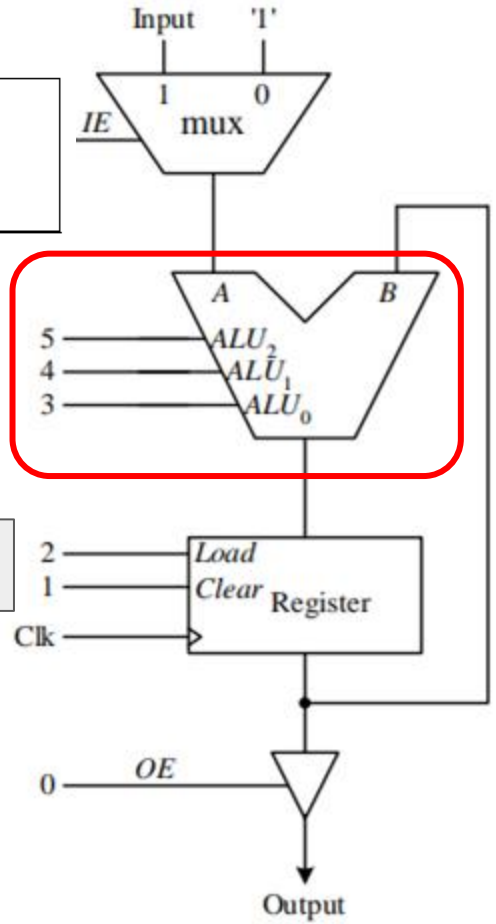
i = i+input

i = i+1

i = i AND 1

}





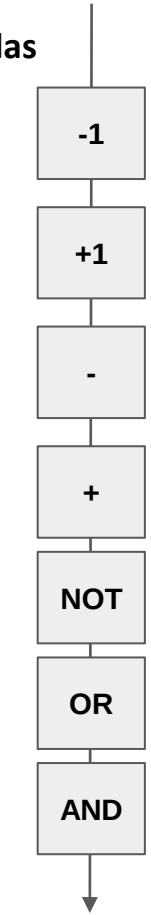
Componentes de blocos operativos

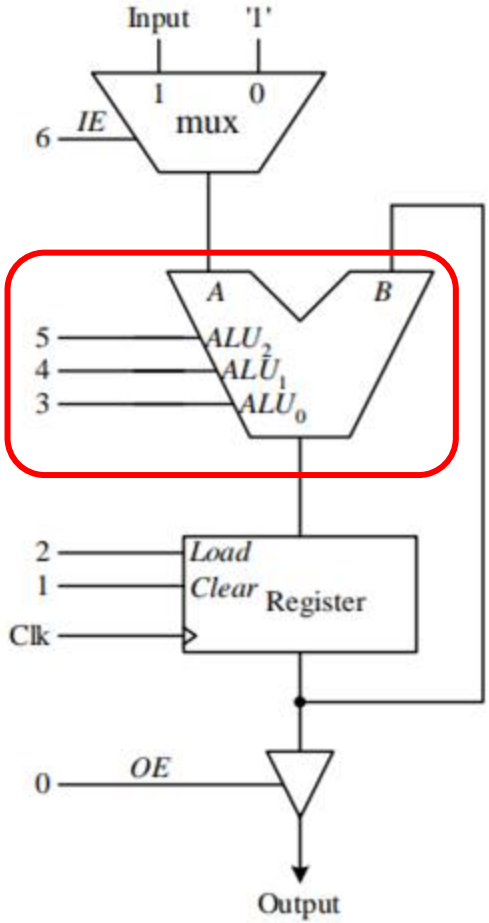
Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath
- Algoritmo mais complexo
    - Forma alternativa para a ALU ser implementada Para maior desempenho

```
i=0

While  (i<10){

        i = i+input

        i = i+1

        i = i AND 1

}
```

Mais bits em ALU para habilitar cada uma das operações

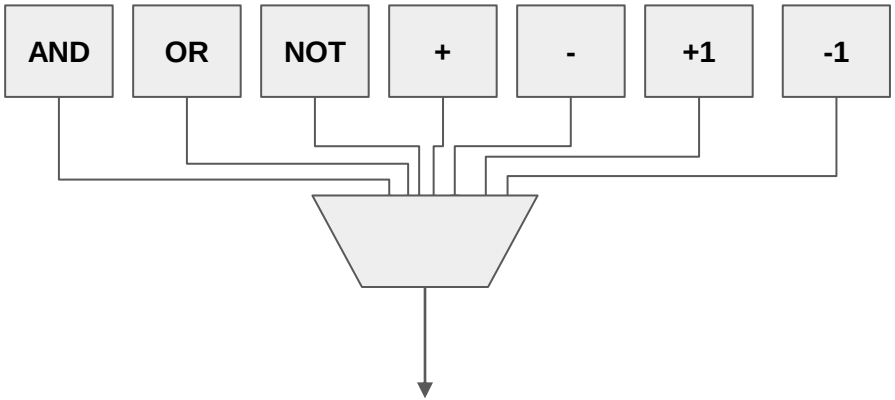


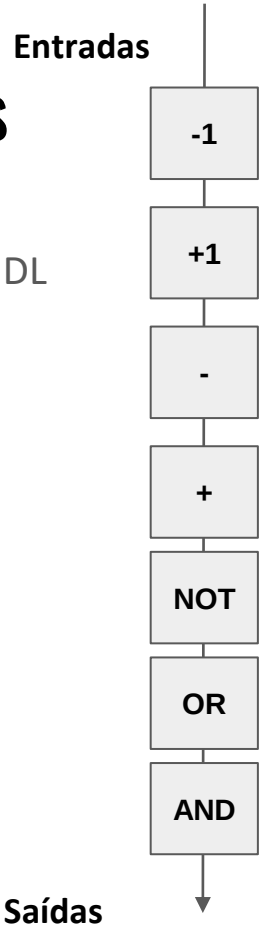


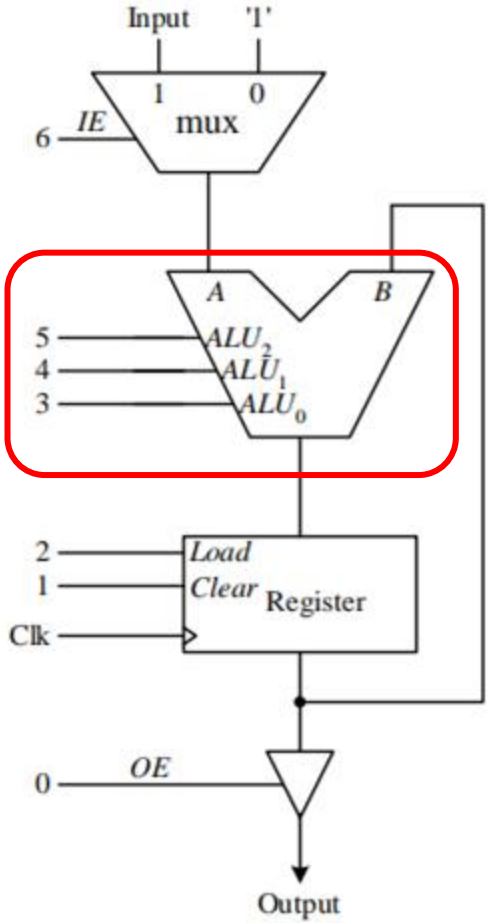
Componentes de blocos operativos

Construção de caminhos de dados (datapaths) em VHDL

- Implementação de algoritmos com datapath
- Algoritmo mais complexo
    - Duas possíveis implementações



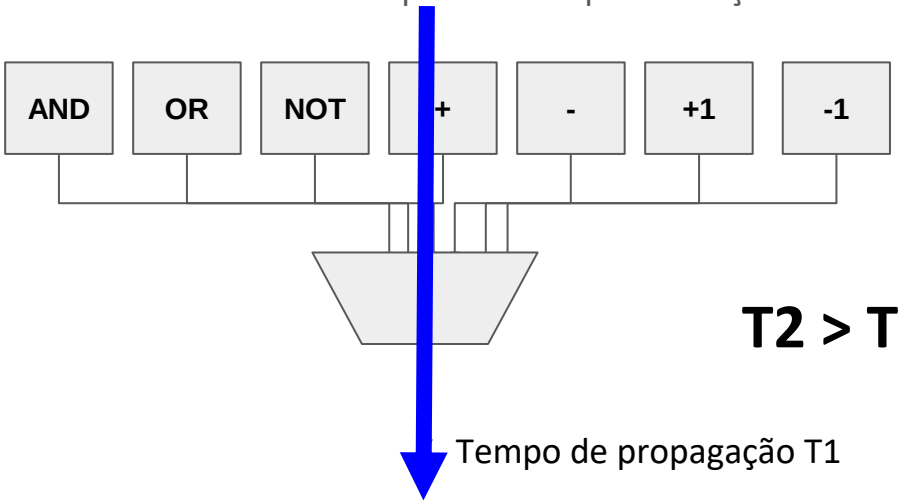




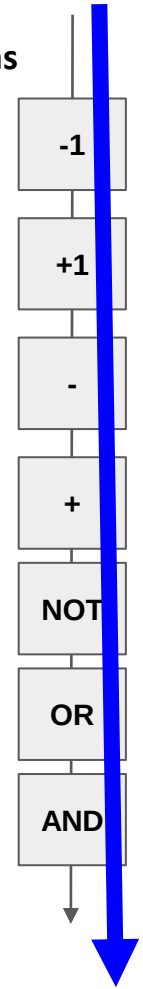
Componentes de blocos operativos

Construção de caminhos de dados (datapaths) em VHDL

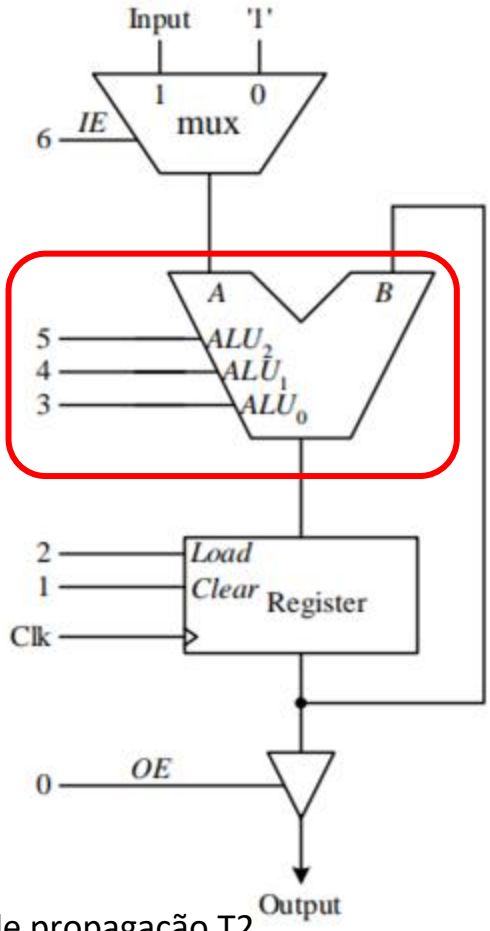
- Implementação de algoritmos com datapath
- Algoritmo mais complexo
  - Duas possíveis implementações



T2 > T1



Tempo de propagação T2



Bibliografia

- Sistemas digitais: Projeto, Otimização e HDLs, Frank Vahid, Ed. Bookman, 1ª Ed., 2008
- Tocci, R. J., Widmer, N. S. Sistemas digitais. 7. ed. Rio de Janeiro: LTC, 1998.