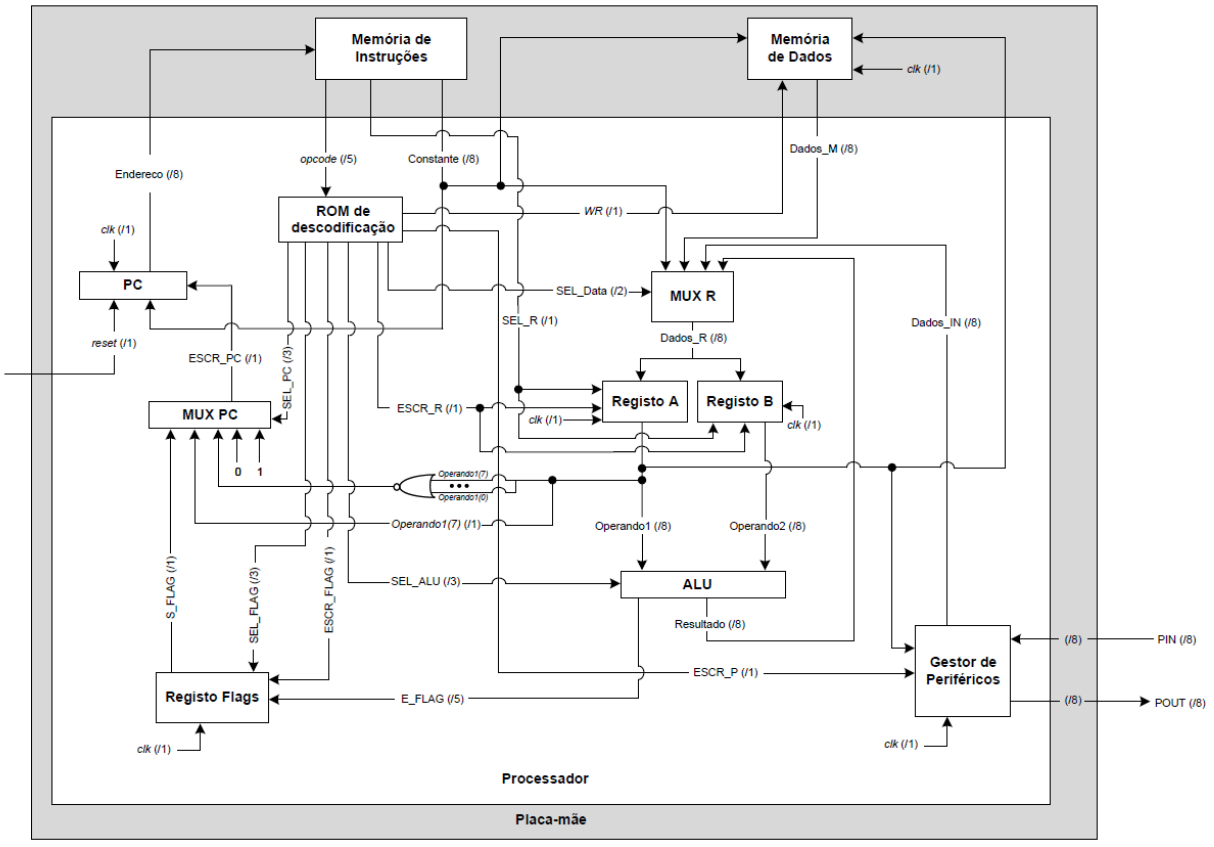
UNIVERSIDADE da MADEIRA

Faculdade de Ciências Exatas e da Engenharia

Licenciatura em Engenharia Informática

Arquitetura de Computadores



DOCENTES:

Dionísio Barros

Sofia Inácio

Pedro Camacho

Dino Vasconcelos

DISCENTES:

Bjorn André Costa Foss nº2048319

Joana Andrade Azevedo nº2076220

Conteúdo

1. Introdução

No ambiento da cadeira Arquitetura de Computadores, foi proposto a realização de um projeto que consiste no desenvolvimento e implementação de um processador. A ferramenta utilizada para desenvolver o projeto é o ISE da Xillinx.

A placa-mãe é composta pela memória de instruções, a memória de dados e o processador, este que é constituído por diversos módulos.

Este processador é capaz de realizar operações lógicas, tais como AND, OR, NOR e XOR, operações aritméticas, soma e subtração, e operações de comparação. A memória de instrução armazena as instruções do programa a ser executado. A memória de dados guarda os dados presentes no sinal de entrada.

2. Objetivos

Este projeto tem como objetivo aprender a criar um processador, que utiliza um conjunto de instruções apresentadas na tabela do Anexo A. Essas instruções serão posteriormente implementadas em VHDL e produzirão saídas diferentes com base nas entradas inseridas.

Além disso, o projeto visa alcançar outros objetivos: desenvolver habilidades de programação em VHDL e consolidar os conhecimentos adquiridos sobre o funcionamento de um PEPE-8 durante as aulas.

3. Desenvolvimento

A placa-mãe é composta por 3 elementos fundamentais:

- Memória de instruções
- Memória de dados
- Processador

3.1 Processador

3.1.1 CONTADOR DE PROGRAMA (PC)

O contador de programa indica qual é a próxima instrução a ser executada pelo processador. A cada ciclo do clk, ele envia um sinal de endereço de 8 bits para a memória de instruções. Se a entrada "ESCR_PC" for 0, o PC seguirá a sequência normal de execução. Se for 1, o processador salta para o endereço de instrução indicado pela entrada "Constante" de 8 bits. A entrada "Reset" permite reiniciar a execução do programa do início.

3.1.2 MUX PC

O multiplexer do PC tem como entrada o SEL_PC, de 3 bits que indica qual dos valores de entrada deve sair na saída e como saída o ESCR_PC, de um bit. O MUX_PC indica se o contador de programa deve efetuar um salto ou incrementar o contador.

O MUX_PC têm 5 saídas possíveis: SEL_PC =>000 - S_FLAG; SEL_PC =>001 – Operação NOR; SEL_PC =>010 – Operando (7); SEL_PC =>011 – ‘0’; SEL_PC =>100 – ‘1’.

3.1.3 REGISTO A E B

Os registos A e B, ambos tem como entrada o sinal ESCR_R, 1 bit, o sinal SEL_R, 1 bit, Dados_R, 8 bits e o sinal clk, 1 bit. O registo A tem como saída o Operando 1 e o registo B tem como saída o Operando 2. Quando o SEL_R está a ‘0’, escolhe o registo A a ser escrito e quando está a ‘1’ é registo B. Ambos os registos continuam a fazer leituras, além da escrita.

3.1.4 MUX R

O multiplexer de registos possui como entradas o Resultado, os Dados_IN, os Dados_M e a Constante, ambos com 8 bits e o SEL_Data, de 2 bits. A sua saída é os Dados_R, de 8 bits. O sinal SEL_Data seleciona qual dos 4 sinais de entrada de 8 bits referidos será encaminhado para o sinal de saída.

3.1.5 ALU

A unidade aritmética e lógica é responsável pelas operações aritméticas(soma e subtração), operações lógicas(AND, OR, NOR e XOR) e operações de comparação(maior, maior ou igual, igual, maior, maior ou igual) de um processador.

O Operando 1 e Operando 2, ambos de 8 bits, e o sinal SEL_ALU, 3 bits, são sinais de entrada. O Resultado, 8 bits, e o E_FLAG, de 5 bits, são as saídas do ALU. A Saída E_FLAG será atualizada apenas quando é executada uma comparação, Quando o SEL_ALU = 110 o bit do E_FLAG representa uma comparação: E_FLAG(0) => menor; E_FLAG(1) => menor ou igual; E_FLAG(2) => igual; E_FLAG(3) => maior; E_FLAG(4) => maior ou igual;

3.1.6 REGISTO DE FLAGS

Guarda o valor do sinal de entrada do E_FLAG, 5 bits, quando o sinal de relógio (clk) estiver na transição ascendente e quando o sinal de entrada ESCR_FLAG, de 1 bit, estiver a ‘1’. Os registos de Flags pode ser: SEL_FLAG => 000 – bit 0 do registo (<); SEL_FLAG => 001 – bit 1 do registo (<=); SEL_FLAG => 010 – bit 2 do registo (=); SEL_FLAG => 011 – bit 3 do registo (>); SEL_FLAG => 100 – bit 4 do registo (>=);

3.1.7 GESTOR DE PERIFÉRICOS

Permite a comunicação entre o processador e o exterior. O gestor de periféricos permite a utilizador inserir dados para depois serem realizadas as operações com os mesmos. Tem como sinais de entrada, o ESCR_P, 1 bit, o PIN, 8 bits, o clk e o Operando 1, 8 bits. E como saída tem os Dados_IN e o POUT, ambos com 8 bits.

Se o ESCR_P está a ‘0’ então escreve o PIN nos Dados_IN. Se o clk estiver na transição ascendente e se ESCR_P a ‘1’, então o Operando 1 escreve no POUT.

3.1.8 ROM DE DESCODIFICAÇÃO

O ROM tem com entrada o sinal opcode, de 5 bits, e como saída os seguintes sinais: SEL_PC, 3 bits, SEL_FLAG, 3 bits, ESCR_FLAG, 1 bit, SEL_ALU, 3 bits, ESCR_R, 1 bit, SEL_DATA, 2 bits, ESCR_P, 1 bit, e WR, 1 bit.

O ROM é responsável por atribuir valores aos sinais de controlo.

3.2 Memória de Instruções

A memória de instruções é o local onde as instruções do programa a ser executadas estão armazenadas. Tem uma dimensão de 14 bits, esta endereçada pelo sinal de Endereco, 8 bits, e a sua saída o sinal opcode, de 5 bits, o SEL_R, 1 bit e o sinal de constante de 8 bits.

3.3 Memória de Dados

A memória de Dados, ou RAM, permite guardar os dados do sinal de entrada Operando 1, quando o WR estiver a '1' e clk na transição ascende na posição de memória indicada pelo sinal de entrada Constante, ou seja, quando o opcode for '00100'. Quando o WR estiver a '0' é realizada apenas uma leitura à posição de memoria indicada pelo sinal Constante, sendo esse valor atribuído ao sinal de saída Dados_M, 8 bits.

4. Discussão de resultado

O processador, de acordo com o PIN, reset, o clock e as instruções da memória de instruções, vai realizar várias operações que resultarão em diferentes valores de POUT. As instruções utilizadas neste programa são listadas no anexo A e foram projetadas para produzir os seguintes resultados:

PIN <-13

Se o valor de PIN for menor que -13, o valor de POUT são positivos. No caso do PIN = -15 o valor de POUT=1. Tal pode ser confirmado com a imagem 6 do anexo A em 7.2.

PIN <0 e >= -13

Quando o PIN é menor que 0 o valor do POUT é sempre negativo. Quando colocamos PIN = -1 o POUT =-13. Tal pode ser confirmado com a imagem 5 do anexo A em 7.2.

PIN >=0 e <=10

Quando o valor do PIN é maior ou igual a 0 e menor ou igual a 10, o valor do POUT é sempre -1. Podemos confirmar este valor com as imagens 2, 3 e 4 do anexo A em 7.2.

PIN >10

Se o valor de PIN for maior que 10, o valor de POUT será o triplo do PIN (POUT= 3*PIN). No caso do PIN = 15 o valor de POUT=45(15*3). Tal pode ser confirmado com a imagem 1 do anexo A em 7.2.

5. Conclusão

Podemos concluir que os objetivos requisitados foram alcançados, pelo que podemos afirmar que a implementação do processador foi um sucesso, pois realiza as instruções pedidas.

A realização deste projeto foi muito útil, pois adquiriu-se mais conhecimento sobre o funcionamento de um processador e como implementá-lo no software de desenvolvimento ISE da Xilinx usando a linguagem VHDL.

6. Bibliografia

[1] J. Delgado e C. Ribeiro, Arquitetura de Computadores, FCA, 2014.

7. Anexo A

7.1 Tabela das instruções de teste

Endereço	Instrução - assembly	Instrução – código máquina		
		opcode	Sel_R	Constante
(0) 00000000	LD RB, 10	000101	1	00001010
(1) 00000001	LDP RA	00000	0	XXXXXXXX
(2) 00000010	JN RA, 7	10010	X	00000111
(3) 00000011	CMP RA, RB	01011	X	XXXXXXXX
(4) 00000100	JG 14	01111	X	00001110
(5) 00000101	LD RA, -1	00010	0	11111111
(6) 00000110	JMP 29	10011	X	00011101
(7) 00000111	LD RB, -1	00010	1	11111111
(8) 00001000	XOR RA, RB	01010	0	XXXXXXXX
(9) 00001001	LD RB, 1	00010	1	00000001
(10) 00001010	ADD RA, RB	00101	0	XXXXXXXX
(11) 00001011	LD RB, 14	00010	1	00001110
(12) 00001100	SUB RA, RB	00110	0	XXXXXXXX
(13) 00001101	JMP 29	10011	X	00011101
(14) 00001110	ST [10], RA	00100	X	00001010
(15) 00001111	LD RA, 0	00010	0	0000000
(16) 00010000	ST [11], RA	00100	X	00001011
(17) 00010001	LD RA, 3	00010	0	00000011
(18) 00010010	ST [5], RA	00100	X	00000101
(19) 00010011	LD RA, [11]	00011	0	00001011
(20) 00010100	LD RB, [10]	00011	1	00001010
(21) 00010101	ADD RA, RB	00101	0	XXXXXXXX
(22) 00010110	ST [11], RA	00100	X	00001011
(23) 00010111	LD RA, [5]	00011	0	00000101
(24) 00011000	LD RB, 1	00010	1	00000001
(25) 00011001	SUB RA, RB	00110	0	XXXXXXXX
(26) 00011010	JZ RA, 28	10001	X	00011100
(27) 00011011	JMP 18	10011	X	00010010
(28) 00011100	LD RA, [11]	00011	0	00001011
(29) 00011101	STP RA	00001	X	XXXXXXXX
(30) 00011110	JMP 30	10011	X	00011110

7.2 Simulação

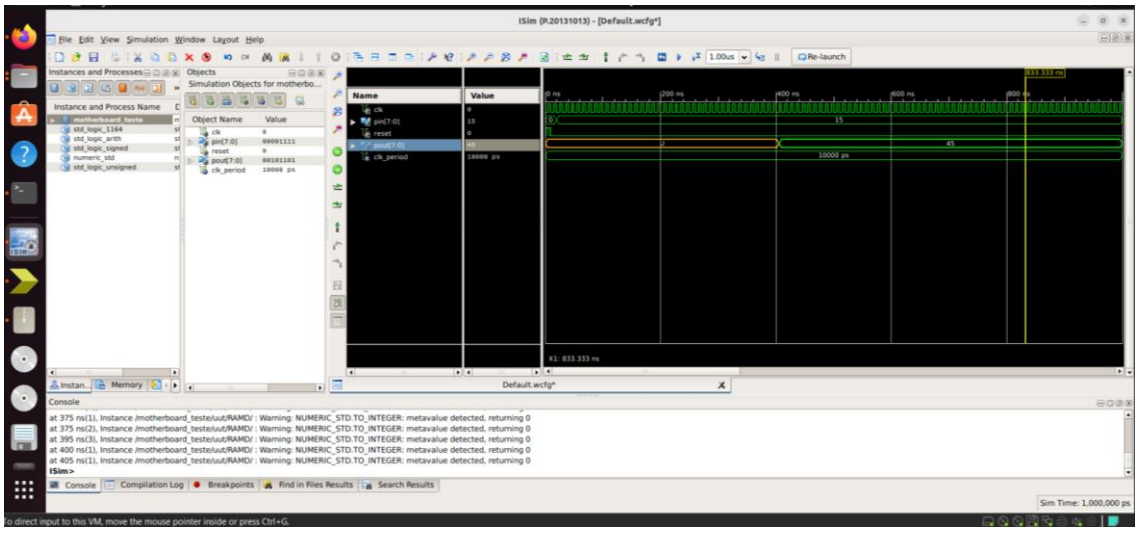


IMAGEM 1 - PIN = 15

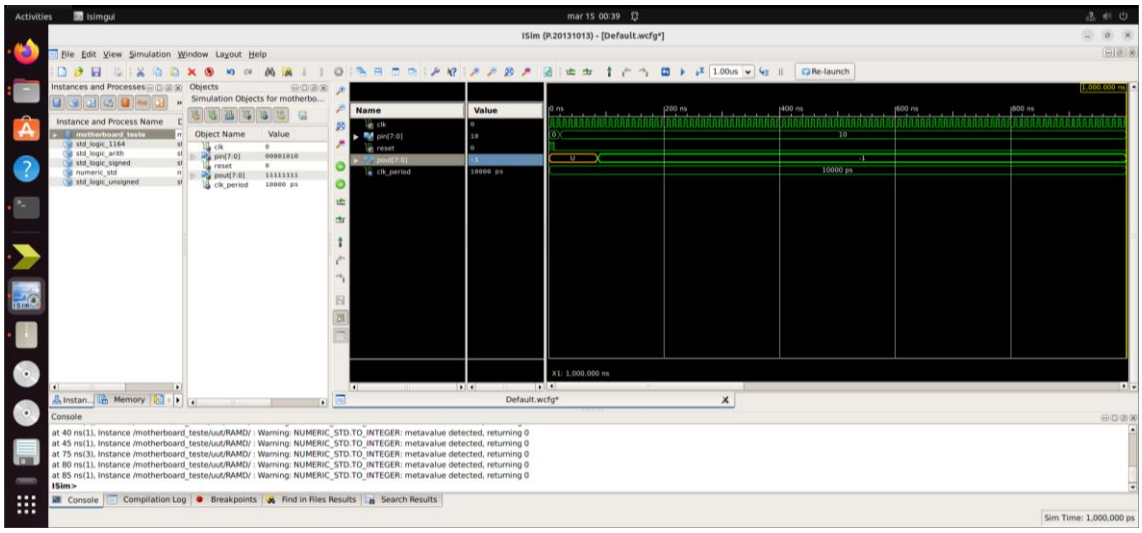


IMAGEM 2 - PIN= 10

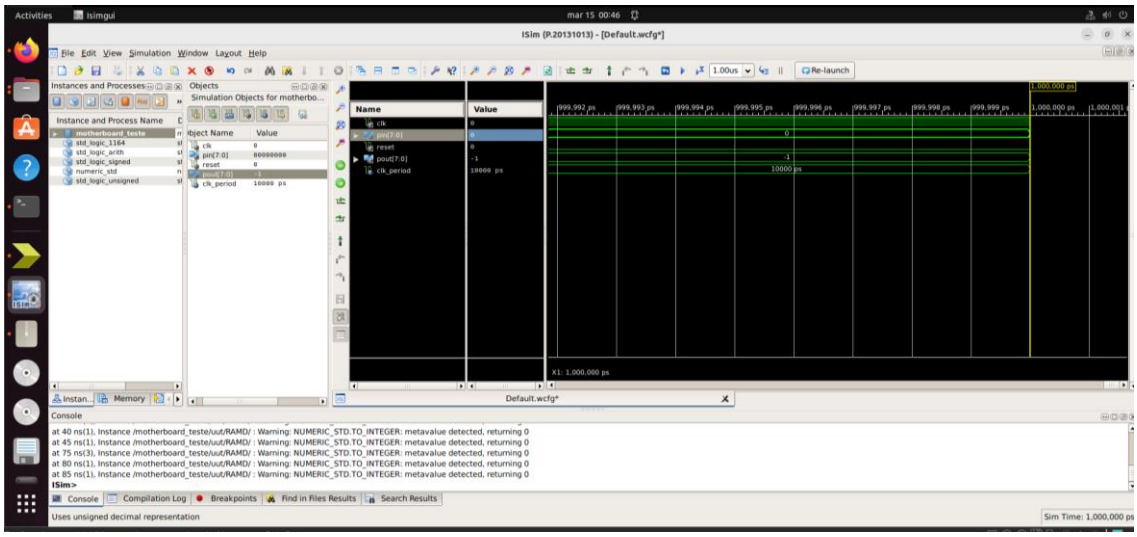


IMAGEM 3 - PIN=0

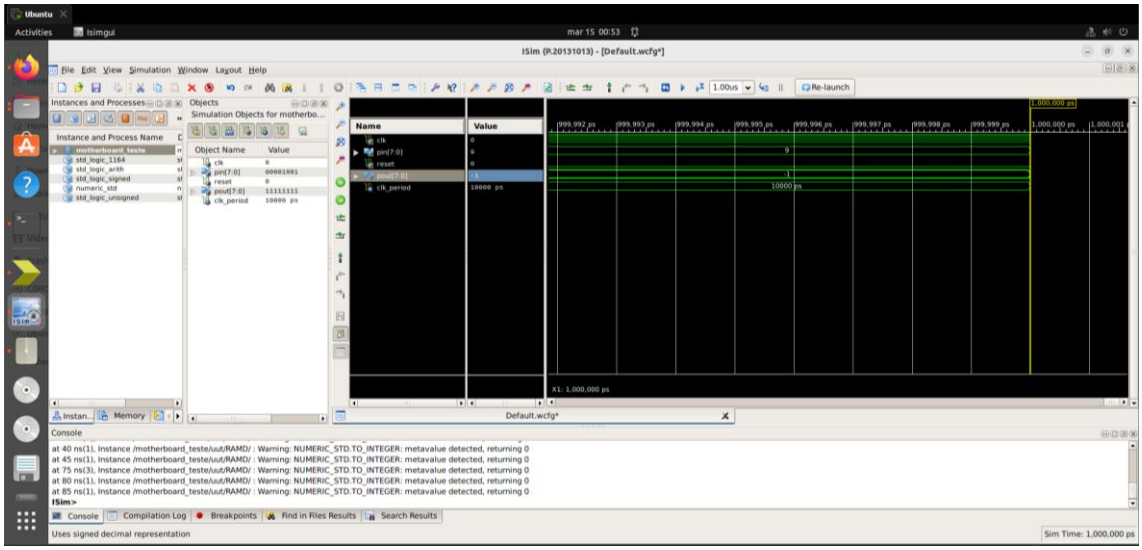


IMAGEM 4 - PIN= 9

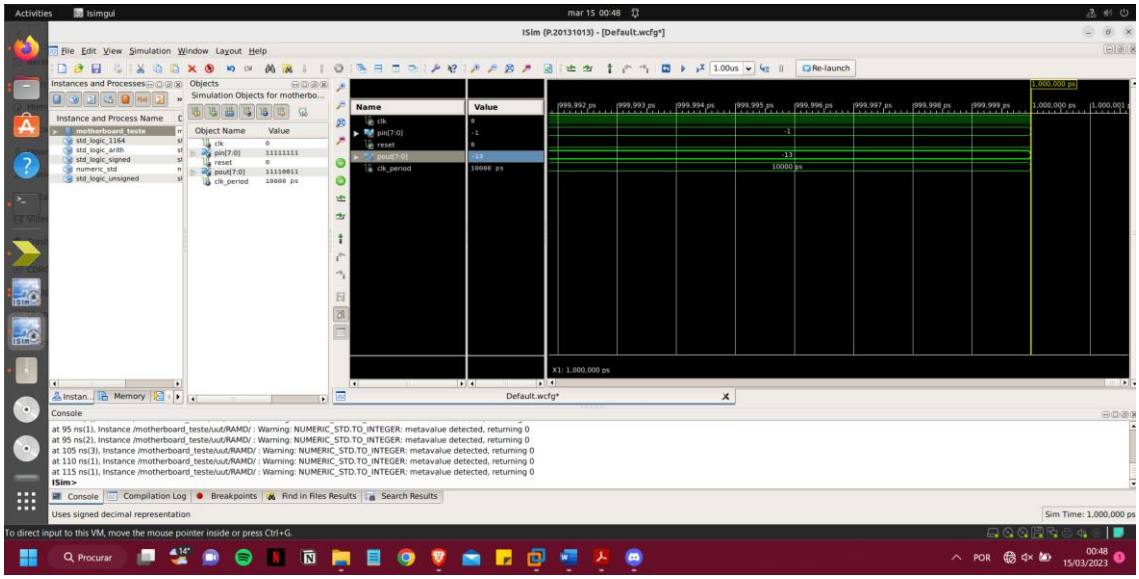


IMAGEM 5 - PIN=-1

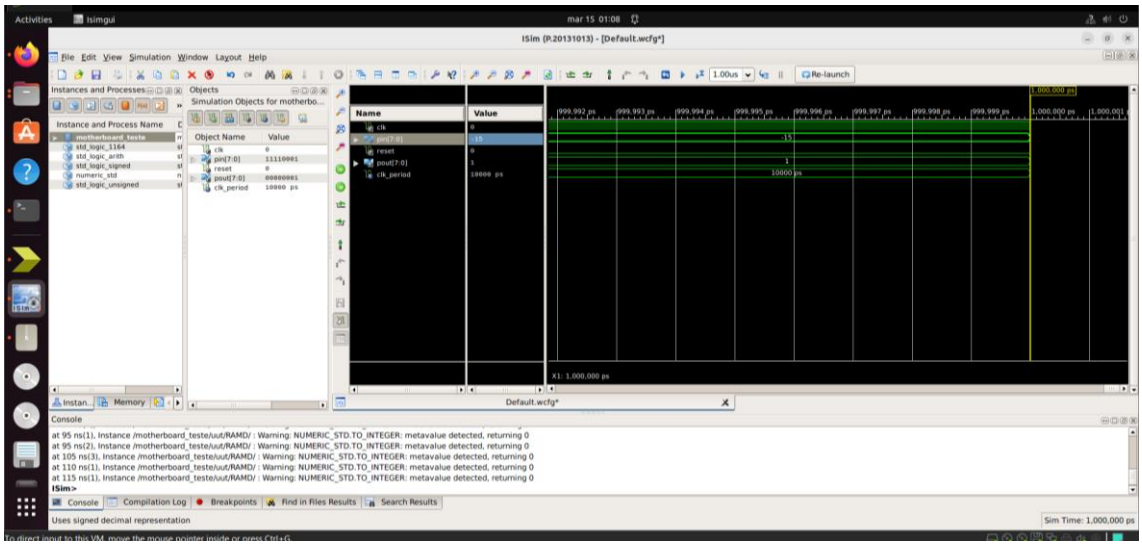


IMAGEM 6-PIN=-15

8. Anexo B

Motherboard

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;    -- IMPORTAR BIBLIOTECAS --

ENTITY Motherboard IS
    Port ( clk : in  STD_LOGIC;
           PIN : in  STD_LOGIC_VECTOR (7 downto 0);

           reset : in  STD_LOGIC;
           POUT : out  STD_LOGIC_VECTOR (7 downto 0));
END Motherboard;

ARCHITECTURE Struct OF Motherboard IS

COMPONENT Processor IS
    Port ( clk : in  STD_LOGIC;
           PIN : in  STD_LOGIC_VECTOR (7 downto 0);
           reset : in  STD_LOGIC;
           opcode : IN  STD_LOGIC_VECTOR (4 downto 0);
           SEL_R : in  STD_LOGIC;
           Constante : IN  STD_LOGIC_VECTOR (7 downto 0);
           Dados_M : in  STD_LOGIC_VECTOR (7 downto 0);
           Endereco : OUT  STD_LOGIC_VECTOR (7 downto 0);
           WR : out  STD_LOGIC;

              Operandol : INOUT  STD_LOGIC_VECTOR (7 downto 0);

              ------------------------------------------------

           POUT : out  STD_LOGIC_VECTOR (7 downto 0));
END COMPONENT;

COMPONENT InstructionsMemory IS
    Port ( Endereco : in  STD_LOGIC_VECTOR (7 downto 0);
              opcode : out  STD_LOGIC_VECTOR (4 downto 0);
              SEL_R : out  STD_LOGIC;
              Constante : out  STD_LOGIC_VECTOR (7 downto 0));
END COMPONENT;

COMPONENT DataMemory IS
    Port ( Operandol : in  STD_LOGIC_VECTOR (7 downto 0);
           WR : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           Constante : in  STD_LOGIC_VECTOR (7 downto 0);
           Dados_M : out  STD_LOGIC_VECTOR (7 downto 0));
END COMPONENT;

SIGNAL opcode : STD_LOGIC_VECTOR (4 DOWNTO 0);
SIGNAL SEL_R : STD_LOGIC;
SIGNAL Constante : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL Dados_M : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL Endereco : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL WR : STD_LOGIC;
SIGNAL Operandol : STD_LOGIC_VECTOR (7 DOWNTO 0);

BEGIN

Processador :   Processor              PORT MAP (clk, PIN, reset, opcode, SEL_R, Constante, Dados_M, Endereco, WR, Operandol, POUT);
IMEM :          InstructionsMemory  PORT MAP (Endereco, opcode, SEL_R, Constante);
RAMD :          DataMemory             PORT MAP (Operandol, WR, clk, Constante, Dados_M);

END Struct;
```

Processador

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

ENTITY Processor IS
    Port ( clk : in  STD_LOGIC;
           PIN : in  STD_LOGIC_VECTOR (7 downto 0);
           reset : in  STD_LOGIC;
           opcode : IN STD_LOGIC_VECTOR (4 downto 0);
           SEL_R : in  STD_LOGIC;
           Constante : IN  STD_LOGIC_VECTOR (7 downto 0);
           Dados_M : in  STD_LOGIC_VECTOR (7 downto 0);
           Endereco : OUT  STD_LOGIC_VECTOR (7 downto 0);
           WR : out  STD_LOGIC;

              Operando1 : INOUT STD_LOGIC_VECTOR (7 downto 0);

              ------------------------------------------------

           POUT : out  STD_LOGIC_VECTOR (7 downto 0));
END Processor;

ARCHITECTURE Struct OF Processor IS

component PeripheralsManager is
    Port ( ESCR_P : in  STD_LOGIC;
              clk : in  STD_LOGIC;
           PIN : in  STD_LOGIC_VECTOR (7 downto 0);
              Operando1 : in STD_LOGIC_VECTOR (7 downto 0);
              Dados_IN : out STD_LOGIC_VECTOR (7 downto 0);
           POUT : out  STD_LOGIC_VECTOR (7 downto 0));
end component;

component MultiplexerRegistos is
    Port ( Constante : in  STD_LOGIC_VECTOR (7 downto 0);
           Dados_M : in  STD_LOGIC_VECTOR (7 downto 0);
           Dados_IN : in  STD_LOGIC_VECTOR (7 downto 0);
           Resultado : in  STD_LOGIC_VECTOR (7 downto 0);
           SEL_Data : in  STD_LOGIC_VECTOR (1 downto 0);
           Dados_R : out  STD_LOGIC_VECTOR (7 downto 0));
end component;

component RegistoA is
    Port ( ESCR_R : in  STD_LOGIC;
           Dados_R : in  STD_LOGIC_VECTOR (7 downto 0);
           SEL_R : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           Operando1 : out  STD_LOGIC_VECTOR (7 downto 0));
end component;

component RegistoB is
    Port ( ESCR_R : in  STD_LOGIC;
           Dados_R : in  STD_LOGIC_VECTOR (7 downto 0);
           SEL_R : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           Operando2 : out  STD_LOGIC_VECTOR (7 downto 0));
end component;

component ALogicUnit is
    Port ( Operando1 : in  STD_LOGIC_VECTOR (7 downto 0);
           Operando2 : in  STD_LOGIC_VECTOR (7 downto 0);
           SEL_ALU : in  STD_LOGIC_VECTOR (2 downto 0);      -- 3-Bits in each operation
           E_FLAG : out  STD_LOGIC_VECTOR (4 downto 0);
           Resultado : out  STD_LOGIC_VECTOR (7 downto 0));
end component;
```

```
84
85      component RegistoFlags is
86          Port ( E_FLAG : in  STD_LOGIC_VECTOR (4 downto 0);
87                 clk : in  STD_LOGIC;
88                 ESCR_FLAG : in  STD_LOGIC;
89                 SEL_FLAG : in  STD_LOGIC_VECTOR (2 downto 0);
90                 S_FLAG : out  STD_LOGIC);
91      end component;
92
93      component ProgramCounter is
94          Port ( Constante : in  STD_LOGIC_VECTOR (7 downto 0);
95                 ESCR_PC : in  STD_LOGIC;
96                 clk : in  STD_LOGIC;
97                 reset : in  STD_LOGIC;
98                 Endereco : out  STD_LOGIC_VECTOR (7 downto 0));
99      end component;
100
101     component MUXProgramCounter is
102         Port ( SEL_PC : in  STD_LOGIC_VECTOR (2 downto 0);
103                S_FLAG : in  STD_LOGIC;
104                  Output_NOR : IN STD_LOGIC;
105                  Operando1 : in STD_LOGIC_VECTOR (7 downto 0);
106                ESCR_PC : out  STD_LOGIC);
107     end component;
108
109     COMPONENT Porta_NOR IS
110         PORT (  Operando1: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
111                     Output_NOR : out STD_LOGIC);
112     END COMPONENT;
113
114     component ROMDecoder is
115         Port ( opcode : in  STD_LOGIC_VECTOR (4 downto 0);
116                SEL_PC : out  STD_LOGIC_VECTOR (2 downto 0);
117                SEL_FLAG : out  STD_LOGIC_VECTOR (2 downto 0);
118                ESCR_FLAG : out  STD_LOGIC;
119                SEL_ALU : out  STD_LOGIC_VECTOR (2 downto 0);
120                ESCR_R : out  STD_LOGIC;
121                SEL_Data : out  STD_LOGIC_VECTOR (1 downto 0);
122                ESCR_P : out  STD_LOGIC;
123                WR : out  STD_LOGIC);
124     end component;
125
126     SIGNAL ESCR_P : STD_LOGIC;
127     SIGNAL Dados_IN : STD_LOGIC_VECTOR (7 DOWNTO 0);
128     SIGNAL Resultado : STD_LOGIC_VECTOR (7 DOWNTO 0);
129     SIGNAL SEL_Data : STD_LOGIC_VECTOR (1 DOWNTO 0);
130     SIGNAL Dados_R : STD_LOGIC_VECTOR (7 DOWNTO 0);
131     SIGNAL ESCR_R : STD_LOGIC;
132     SIGNAL Operando2 : STD_LOGIC_VECTOR (7 DOWNTO 0);
133     SIGNAL E_FLAG : STD_LOGIC_VECTOR (4 DOWNTO 0);
134     SIGNAL SEL_ALU : STD_LOGIC_VECTOR (2 DOWNTO 0);
135     SIGNAL ESCR_PC : STD_LOGIC;
136     SIGNAL SEL_FLAG : STD_LOGIC_VECTOR (2 DOWNTO 0);
137     SIGNAL S_FLAG : STD_LOGIC;
138     SIGNAL Output_NOR : STD_LOGIC;
139     SIGNAL SEL_PC : STD_LOGIC_VECTOR (2 DOWNTO 0);
140     SIGNAL ESCR_FLAG : STD_LOGIC;
141
```

```
141
142     BEGIN
143
144     -- ASSEMBLING EACH COMPONENT --
145
146     Peripherals : PeripheralsManager    PORT MAP (ESCR_P, clk, PIN, Operando1, Dados_IN, POUT);
147     MuxRegistos : MultiplexerRegistos   PORT MAP (Constante, Dados_M, Dados_IN, Resultado, SEL_Data, Dados_R);
148     REGA            : RegistoA                  PORT MAP (ESCR_R, Dados_R, SEL_R, clk, Operando1);
149     REGB            : RegistoB                  PORT MAP (ESCR_R, Dados_R, SEL_R, clk, Operando2);
150     LogicUnit   : ALogicUnit                PORT MAP (Operando1, Operando2, SEL_ALU, E_FLAG, Resultado);
151     RegFlags    : RegistoFlags          PORT MAP (E_FLAG, clk, ESCR_FLAG, SEL_FLAG, S_FLAG);
152     PCounter    : ProgramCounter            PORT MAP (Constante, ESCR_PC, clk, reset, Endereco);
153     MUXPC       : MUXProgramCounter         PORT MAP (SEL_PC, S_FLAG, Output_NOR, Operando1, ESCR_PC);
154     NOR_GATE        : Porta_NOR                 PORT MAP (Operando1, Output_NOR);
155     ROMD            : ROMDecoder                PORT MAP (opcode, SEL_PC, SEL_FLAG, ESCR_FLAG, SEL_ALU, ESCR_R, SEL_Data, ESCR_P, WR);
156
157     -- #
158
159     END Struct;
160
161
```

Gestor de Periféricos

```
1   ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  entity PeripheralsManager is
24      Port ( ESCR_P : in  STD_LOGIC;
25             PIN : in  STD_LOGIC_VECTOR (7 downto 0);
26             clk : in  STD_LOGIC;
27               Operando1 : in STD_LOGIC_VECTOR (7 downto 0);
28               Dados_IN : out STD_LOGIC_VECTOR (7 downto 0);
29             POUT : out  STD_LOGIC_VECTOR (7 downto 0));
30  end PeripheralsManager;
31
32  architecture Behavioral of PeripheralsManager is
33
34  begin
35      process(ESCR_P, PIN, clk, Operando1)
36      begin
37
38          if ESCR_P = '0' then
39              Dados_IN <= PIN;
40          end if;
41
42          if rising_edge (clk) then
43              if ESCR_P = '1' then
44                  POUT <= Operando1;
45              end if;
46          end if;
47
48      end process;
49
50  end Behavioral;
51
52
```

Multiplexer Registos

```
1  ----------------------------------------------------------------------------
20 LIBRARY IEEE;
21 USE IEEE.STD_LOGIC_1164.ALL;
22
23 ENTITY MultiplexerRegistos IS
24     Port ( Constante : in  STD_LOGIC_VECTOR (7 downto 0);
25            Dados_M : in  STD_LOGIC_VECTOR (7 downto 0);
26            Dados_IN : in  STD_LOGIC_VECTOR (7 downto 0);
27            Resultado : in  STD_LOGIC_VECTOR (7 downto 0);
28            SEL_Data : in  STD_LOGIC_VECTOR (1 downto 0);
29            Dados_R : out  STD_LOGIC_VECTOR (7 downto 0));
30 END MultiplexerRegistos;
31
32 ARCHITECTURE Behavioral OF MultiplexerRegistos IS
33
34 BEGIN
35     PROCESS(Constante, Dados_M, Dados_IN, Resultado, SEL_Data)      -- Inicia o Processo do MUX R
36     BEGIN
37
38         CASE SEL_Data IS
39
40             WHEN    "00"    =>  Dados_R <=  Resultado;
41             WHEN    "01"    =>  Dados_R <=  Dados_IN;
42             WHEN    "10"    =>  Dados_R <=  Dados_M;
43             WHEN    "11"    =>  Dados_R <=  Constante;
44
45             WHEN OTHERS =>  Dados_R <=  "XXXXXXXX";
46
47         END CASE;
48
49     END PROCESS;
50
51 END Behavioral;
52
53
```

Registo A

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  entity RegistoA is
24      Port ( ESCR_R : in  STD_LOGIC;
25             Dados_R : in  STD_LOGIC_VECTOR (7 downto 0);
26             SEL_R : in  STD_LOGIC;
27             clk : in  STD_LOGIC;
28             Operando1 : out  STD_LOGIC_VECTOR (7 downto 0));
29  end RegistoA;
30
31  architecture Behavioral of RegistoA is
32
33  BEGIN
34      PROCESS(ESCR_R, Dados_R, SEL_R, clk)
35      Variable registo1 : STD_LOGIC_VECTOR (7 downto 0);
36      BEGIN
37          IF ESCR_R = '0' THEN
38           --Operando1 <= registo1;
39          ELSE
40              IF RISING_EDGE (clk) THEN
41                  IF SEL_R = '0' THEN
42                      registo1 := Dados_R;
43                  END IF;
44              END IF;
45          END IF;
46      Operando1 <= registo1;
47      END PROCESS;
48
49  END Behavioral;
50
51
```

Registo B

```
1   ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  ENTITY RegistoB IS
24      Port ( ESCR_R : in  STD_LOGIC;
25             Dados_R : in  STD_LOGIC_VECTOR (7 downto 0);
26             SEL_R : in  STD_LOGIC;
27             clk : in  STD_LOGIC;
28             Operando2 : out  STD_LOGIC_VECTOR (7 downto 0));
29  END RegistoB;
30
31  ARCHITECTURE Behavioral OF RegistoB IS
32
33  BEGIN
34
35      PROCESS(ESCR_R, Dados_R, SEL_R, clk)
36      Variable registo2 : STD_LOGIC_VECTOR (7 downto 0);
37      BEGIN
38          IF ESCR_R = '0' THEN
39           --Operando2 <= registo2;
40          ELSE
41              IF RISING_EDGE (clk) THEN
42                  IF SEL_R = '1' THEN
43                      registo2 := Dados_R;
44                  END IF;
45              END IF;
46          END IF;
47      Operando2 <= registo2;
48      END PROCESS;
49
50
51  END Behavioral;
52
53
```

ALU

```
1  ----------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_SIGNED.ALL;
23
24 entity ALogicUnit is
25     Port ( Operando1 : in  STD_LOGIC_VECTOR (7 downto 0);
26            Operando2 : in  STD_LOGIC_VECTOR (7 downto 0);
27            SEL_ALU : in  STD_LOGIC_VECTOR (2 downto 0);     -- 3-Bits in each operation
28            E_FLAG : out  STD_LOGIC_VECTOR (4 downto 0);
29            Resultado : out  STD_LOGIC_VECTOR (7 downto 0));
30 end ALogicUnit;
31
32 architecture Behavioral of ALogicUnit is
33
34 begin
35     process(Operando1,Operando2,SEL_ALU)
36         variable vetor: STD_LOGIC_VECTOR (7 downto 0);
37         begin
38         case SEL_ALU is -- sinal de selecção da ALU que determina os casos de todas as operações executadas na ALU
39
40             when "000"  =>  vetor := Operando1  +       Operando2;
41             when "001"  => vetor := Operando1   -       Operando2;
42             when "010"  => vetor := Operando1   AND     Operando2;
43             when "011"  => vetor := Operando1   OR  Operando2;
44             when "100"  => vetor := Operando1   NOR     Operando2;
45             when "101"  => vetor := Operando1   XOR     Operando2;
46
47             WHEN "110"  => IF (Operando1 < Operando2)
48                                 then
49                                     E_FLAG(0)   <=  '1';
50                                 else
51                                     E_FLAG(0)   <=  '0';
52                            END IF;
53
54                            IF (Operando1 <= Operando2)
55                                 then
56                                     E_FLAG(1)   <=  '1';
57                                 else
58                                     E_FLAG(1)   <=  '0';
59                            END IF;
60
61                            IF (Operando1 = Operando2)
62                                 then
63                                     E_FLAG(2)   <=  '1';
64                                 else
65                                     E_FLAG(2)   <=  '0';
66                            END IF;
67
68                            IF (Operando1 > Operando2)
69                                 then
70                                     E_FLAG(3)   <=  '1';
71                                 else
72                                     E_FLAG(3)   <=  '0';
73                            END IF;
74
75                            IF (Operando1 <= Operando2)
76                                 then
77                                     E_FLAG(4)   <=  '1';
78                                 else
79                                     E_FLAG(4)   <=  '0';
80                            END IF;
81             when others =>  vetor   :=  (others =>  '0');       -- other cases
82         end case;
83         Resultado <= vetor; -- Atribuição da variável à saída Resultado
84     end process;
85 end Behavioral;
```

Registo de Flags

```
1   ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  entity RegistoFlags is
24      Port ( E_FLAG : in  STD_LOGIC_VECTOR (4 downto 0);
25             clk : in  STD_LOGIC;
26             ESCR_FLAG : in  STD_LOGIC;
27             SEL_FLAG : in  STD_LOGIC_VECTOR (2 downto 0);
28             S_FLAG : out  STD_LOGIC);
29  end RegistoFlags;
30
31  architecture Behavioral of RegistoFlags is
32
33  begin
34      process(E_FLAG, clk, ESCR_FLAG, SEL_FLAG)
35      Variable VAR_E_FLAG : STD_LOGIC_VECTOR(4 DOWNTO 0);
36      BEGIN
37
38      IF RISING_EDGE(CLK) THEN            -- Transição Ascendente do Ciclo de Relógio
39          IF ESCR_FLAG = '1' THEN         -- Se a Flag de Escrita estiver ativa
40
41              VAR_E_FLAG := E_FLAG;       -- Atribuímos uma variável à FLAG
42
43          END IF;
44
45      END IF;
46
47          CASE SEL_FLAG IS                    -- FLAG de Seleção que determina cada caso
48
49              WHEN "000"  => S_FLAG <= VAR_E_FLAG(0);     -- Operação (< )
50              WHEN "001"  => S_FLAG <= VAR_E_FLAG(1);     -- Operação (<=)
51              WHEN "010"  => S_FLAG <= VAR_E_FLAG(2);     -- Operação (= )
52              WHEN "011"  => S_FLAG <= VAR_E_FLAG(3);     -- Operação (> )
53              WHEN "100"  => S_FLAG <= VAR_E_FLAG(4);     -- Operação (>=)
54              WHEN OTHERS => S_FLAG <= 'X';                       -- Restantes Casos ..
55
56          END CASE;
57
58      END PROCESS;
59
60  END Behavioral;
61
```

PC

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ProgramCounter is
    Port ( Constante : in  STD_LOGIC_VECTOR (7 downto 0);
           ESCR_PC : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           Endereco : out  STD_LOGIC_VECTOR (7 downto 0));
end ProgramCounter;

architecture Behavioral of ProgramCounter is

begin
    PROCESS(Constante, ESCR_PC, clk, reset)
    variable conta : STD_LOGIC_VECTOR (7 downto 0);
    begin

    if rising_edge (clk) then
        if reset = '0' then
            if ESCR_PC = '0' then
                conta := conta + 1;
            else
                conta := Constante;
            end if;
        else
            conta := "00000000";
        end if;
    end if;

    Endereco <= conta;
    end process;

end Behavioral;
```

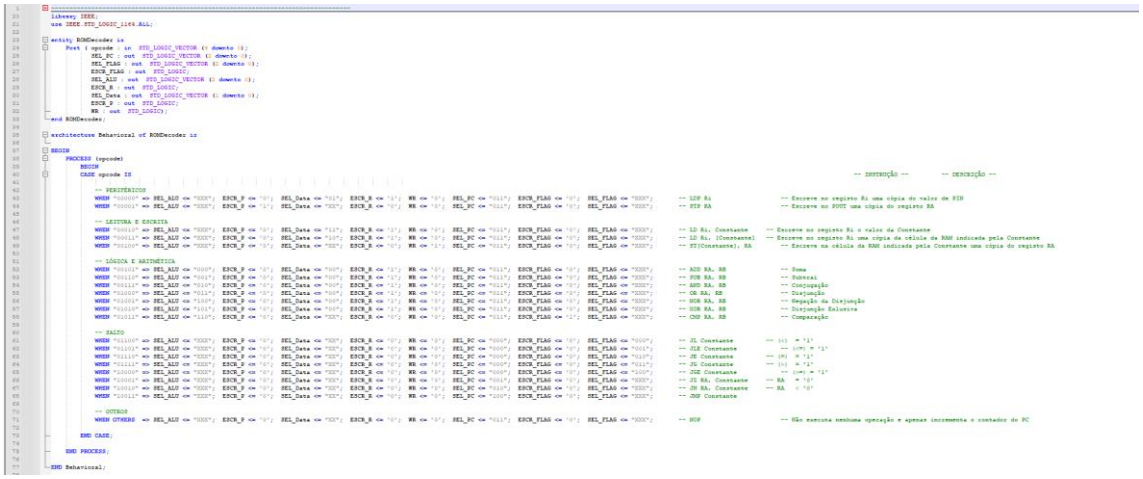
Multiplexer PC

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUXProgramCounter is
    Port ( SEL_PC : in  STD_LOGIC_VECTOR (2 downto 0);
           S_FLAG : in  STD_LOGIC;
           Output_NOR : IN STD_LOGIC;
           Operando1 : in STD_LOGIC_VECTOR (7 downto 0);
           ESCR_PC : out  STD_LOGIC);
end MUXProgramCounter;

architecture Behavioral of MUXProgramCounter is

begin
    process(SEL_PC, S_FLAG, Operando1)
    begin

        case SEL_PC is      -- determina os casos através do sinal de seleção do PC

            when "000"  =>  ESCR_PC <=  S_FLAG;
        --  when "001"  =>  ESCR_PC <=  NOT(Operando1(7) OR Operando1(6) OR Operando1(5) OR Operando1(4) OR Operando1(3) OR Operando1(2) OR Operando1(1) OR Operando1(0));
            WHEN "001"  =>  ESCR_PC <= Output_NOR;
            when "010"  =>  ESCR_PC <=  Operando1(7);
            when "011"  =>  ESCR_PC <=  '0';
            when "100"  =>  ESCR_PC <=  '1';

            when others =>  ESCR_PC <=  'X';

        end case;

    end process;

end Behavioral;
```

Porta NOR

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    17:26:09 03/07/2023
-- Design Name:
-- Module Name:    Porta_NOR - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Porta_NOR is
    PORT (  Operando1: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
            Output_NOR : out STD_LOGIC);
end Porta_NOR;

architecture Behavioral of Porta_NOR is

begin

    PROCESS (Operando1)
    BEGIN

        Output_NOR <= NOT(Operando1(7) OR Operando1(6) OR Operando1(5) OR Operando1(4) OR Operando1(3) OR Operando1(2) OR Operando1(1) OR Operando1(0));

    END PROCESS;

end Behavioral;
```

ROM



Memória de Instruções

```
1   --------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  entity InstructionsMemory is
24      Port (  Endereco : in  STD_LOGIC_VECTOR (7 downto 0);
25              opcode : out  STD_LOGIC_VECTOR (4 downto 0);
26              SEL_R : out  STD_LOGIC;
27              Constante : out  STD_LOGIC_VECTOR (7 downto 0));
28  end InstructionsMemory;
29
30  ARCHITECTURE Behavioral of InstructionsMemory is
31
32  begin
33
34      PROCESS (Endereco)
35      --BEGIN
36
37          TYPE Memory IS ARRAY (0 TO 255) OF STD_LOGIC_VECTOR (13 DOWNTO 0);          -- DIMENSÃO de 14-Bits :
38          Variable Address_Memory : Memory;                                                   -- opcode       - 5-Bits
39          BEGIN                                                                                                   -- SEL_R            - 1-Bit
40                                                                                                                  -- Constante    - 8-Bits
41          CASE Endereco IS        -- Todos os casos dos endereços em 8-Bits Binary
42
43  -- ENDEREÇO -- INSTRUÇÃO                                                                -- DESCRIÇÃO de cada Instrução
44
45  --  1           -- LD RB, 10
46                  WHEN "00000000"     =>  opcode      <=  "00010";                -- LDP Ri, Constante
47                                          SEL_R       <=  '1';                        -- RB -> SEL_R = '1'
48                                          Constante   <=  "00001010";         -- 10 -> 1010
49
50  --  2           -- LDP RA
51                  WHEN "00000001"     =>  opcode      <=  "00000";                -- LDP Ri
52                                          SEL_R       <=  '0';                        -- RA -> SEL_R = '0'
53                                          Constante   <=  "XXXXXXXX";         -- Não existe Constante
54
55  --  3           -- JN RA, 7
56                  WHEN "00000010"     =>  opcode      <=  "10010";                -- JN RA, Constante
57                                          SEL_R       <=  '0';                        -- RA -> SEL_R = '0'
58                                          Constante   <=  "00000111";         -- 7 -> 111
59
60  --  4           -- CMP RA, RB
61                  WHEN "00000011" =>  opcode      <=  "01011";                -- CMP RA, RB
62                                          SEL_R       <=  'X';                        -- RA e RB na mesma instrução -> SEL_R = 'X'
63                                          Constante   <=  "XXXXXXXX";         -- Não existe Constante
64
65  -- 5            -- JG 14
66                  WHEN "00000100" =>      opcode      <=  "01111";                -- JG Constante
67                                          SEL_R       <=  'X';                        -- Não existe Registos nesta instrução
68                                          Constante   <=  "00001110";         -- 14 -> 1110
69
70  -- 6            -- LD RA, -1
71                  WHEN "00000101" =>  opcode      <=  "00010";                -- LD Ri, Constante
72                                          SEL_R       <=      '0';                    -- RA -> SEL_R = '0'
73                                          Constante   <=      "11111111";         -- -1 -> 1111 1111
74
75  -- 7            -- JMP 29
76                  WHEN "00000110" =>      opcode      <=      "10011";            -- JMP Constante
77                                          SEL_R       <=      'X';                    -- Não existe Registos nesta instrução
78                                          Constante   <=      "00011101";         -- 29 -> 11101
79
80  -- 8            -- LD RB, -1
81                  WHEN "00000111" =>      opcode      <=      "00010";            -- LD Ri, Constante
82                                          SEL_R       <=      '1';                    -- RB -> SEL_R = '1'
83                                          Constante   <=      "11111111";         -- -1 -> 1111 1111
84
```

```
-- 9           -- XOR RA, RB
            WHEN "00001000" =>      opcode      <=      "01010";            -- XOR RA, RB
                                    SEL_R       <=      '0';                    -- Não existe Registos nesta instrução
                                    Constante   <=      "XXXXXXXX";         -- Não existe Constante

-- 10          -- LD RB, 1
            WHEN "00001001" =>      opcode      <=      "00010";            -- LD Ri, Constante
                                    SEL_R       <=      '1';                    -- RB -> SEL_R = '1'
                                    Constante   <=      "00000001";         -- 1 -> 1

-- 11       -- ADD RA, RB
            WHEN "00001010" =>      opcode      <=      "00101";            -- ADD RA, RB
                                    SEL_R       <=      '0';                    -- RA e RB na mesma instrução -> SEL_R = 'X'
                                    Constante   <=      "XXXXXXXX";         -- Não existe Constante

-- 12          -- LD RB, 14
            WHEN "00001011" =>      opcode      <=      "00010";            -- LD Ri, Constante
                                    SEL_R       <=      '1';                    -- RB -> SEL_R = '1'
                                    Constante   <=      "00001110";         -- 14 -> 1110

-- 13          -- SUB RA, RB
            WHEN "00001100" =>      opcode      <=      "00110";            -- SUB RA, RB
                                    SEL_R       <=      '0';                    -- RA e RB na mesma instrução -> SEL_R = 'X'
                                    Constante   <=      "XXXXXXXX";         -- Não existe Constante

-- 14          -- JMP 29
            WHEN "00001101" =>      opcode      <=      "10011";            -- JMP Constante
                                    SEL_R       <=      'X';                    -- Não existe Registos nesta instrução
                                    Constante   <=      "00011101";         -- 29 -> 11101

-- 15          -- ST[10], RA
            WHEN "00001110" =>      opcode      <=      "00100";            -- ST[Constante], RA
                                    SEL_R       <=      '0';                    -- RA -> SEL_R = '0'
                                    Constante   <=      "00001010";         -- 10 -> 1010

-- 16          -- LD RA, 0
            WHEN "00001111" =>      opcode      <=      "00010";            -- LD Ri, Constante
                                    SEL_R       <=      '0';                    -- RA -> SEL_R = '0'
                                    Constante   <=      "00000000";         -- 0 -> 0

-- 17          -- ST[11], RA
            WHEN "00010000" =>      opcode      <=      "00100";            -- ST[Constante], RA
                                    SEL_R       <=      '0';                    -- RA -> SEL_R = '0'
                                    Constante   <=      "00001011";         -- 11 -> 1011

-- 18          -- LD RA, 3
            WHEN "00010001" =>      opcode      <=      "00010";            -- LD Ri, Constante
                                    SEL_R       <=      '0';                    -- RA -> SEL_R = '0'
                                    Constante   <=      "00000011";         -- 3 -> 11

-- 19          -- ST[5], RA
            WHEN "00010010" =>      opcode      <=      "00100";            -- ST[Constante], RA
                                    SEL_R       <=      '0';                    -- RA -> SEL_R = '0'
                                    Constante   <=      "00000101";         -- 5 -> 101

-- 20          -- LD RA, [11]
            WHEN "00010011" =>      opcode      <=      "00011";            -- LD Ri, [Constante]
                                    SEL_R       <=      '0';                    -- RA -> SEL_R = '0'
                                    Constante   <=      "00001011";         -- 11 -> 1011

-- 21          -- LD RB, [10]
            WHEN "00010100" =>      opcode      <=      "00011";            -- LD Ri, [Constante]
                                    SEL_R       <=      '1';                    -- RB -> SEL_R = '1'
                                    Constante   <=      "00001010";         -- 10 -> 1010
```

```
                                        Constante   <=      "00001010";         -- 10 -> 1010

    -- 22           -- ADD RA, RB
                    WHEN "00010101"     =>  opcode      <=      "00101";            -- ADD RA, RB
                                            SEL_R       <=      '0';                    -- RA e RB na mesma instrução -> SEL_R = 'X'
                                            Constante   <=      "XXXXXXXX";         -- Não existe Constante

    -- 23           -- ST[11], RA
                    WHEN "00010110" =>      opcode      <=      "00100";            -- ST[Constante], RA
                                            SEL_R       <=      '0';                    -- RA -> SEL_R = '0'
                                            Constante   <=      "00001011";         -- 11 -> 1011

    -- 24           -- LD RA, [5]
                    WHEN "00010111" =>      opcode      <=      "00011";            -- LD Ri, [Constante]
                                            SEL_R       <=      '0';                    -- RA -> SEL_R = '0'
                                            Constante   <=      "00000101";         -- 5 -> 101

    -- 25           -- LD RB, 1
                    WHEN "00011000" =>      opcode      <=      "00010";            -- LD Ri, Constante
                                            SEL_R       <=      '1';                    -- RB -> SEL_R = '1'
                                            Constante   <=      "00000001";         -- 1 -> 1

    -- 26           -- SUB RA, RB
                    WHEN "00011001" =>      opcode      <=      "00110";            -- SUB RA, RB
                                            SEL_R       <=      '0';                    -- RA e RB na mesma instrução -> SEL_R = 'X'
                                            Constante   <=      "XXXXXXXX";         -- Não existe Constante

    -- 27           -- JZ RA, 28
                    WHEN "00011010" =>      opcode      <=      "10001";            -- JZ RA, Constante
                                            SEL_R       <=      '0';                    -- RA -> SEL_R = '0'
                                            Constante   <=      "00011100";         -- 28 -> 11100

    -- 28           -- JMP 18
                    WHEN "00011011" =>      opcode      <=      "10011";            -- JMP Constante
                                            SEL_R       <=      'X';                    -- Não existe Registos nesta instrução
                                            Constante   <=      "00010010";         -- 18 -> 10010

    -- 29           -- LD RA, [11]
                    WHEN "00011100" =>      opcode      <=      "00011";            -- LD Ri, [Constante]
                                            SEL_R       <=      '0';                    -- RA -> SEL_R = '0'
                                            Constante   <=  "00001011";         -- 11 -> 1011

    -- 30       -- STP RA
                    WHEN "00011101" =>      opcode      <=      "00001";            -- STP RA
                                            SEL_R       <=      '0';                    -- RA -> SEL_R = '0'
                                            Constante   <=      "XXXXXXXX";         -- Não existe Constante

    -- 31           -- JMP 30
                    WHEN "00011110" =>      opcode      <=      "10011";            -- JMP Constante
                                            SEL_R       <=      'X';                    -- Não existe Registos nesta instrução
                                            Constante   <=      "00011110";         -- 30 -> 11110

    -- NOP      -- OUTROS CASOS                                                     -- NO Operation
                    WHEN OTHERS         =>      opcode      <=      "XXXXX";
                                            SEL_R       <=      'X';
                                            Constante   <=      "XXXXXXXX";

        END CASE;

    END PROCESS;

end Behavioral;
```

Memória de Dados

```
1   ----------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  USE IEEE.NUMERIC_STD.ALL;       -- To declare TO_INTEGER
24
25  ENTITY DataMemory IS
26      Port ( Operando1 : in  STD_LOGIC_VECTOR (7 downto 0);
27             WR : in  STD_LOGIC;
28             clk : in  STD_LOGIC;
29             Constante : in  STD_LOGIC_VECTOR (7 downto 0);
30             Dados_M : out  STD_LOGIC_VECTOR (7 downto 0));
31  END DataMemory;
32
33  ARCHITECTURE Behavioral OF DataMemory IS
34
35  BEGIN
36      PROCESS(Operando1, WR, clk, Constante)
37
38      -- Cria um array de 256 bits
39      TYPE Memory IS ARRAY (0 TO 255) OF STD_LOGIC_VECTOR (7 DOWNTO 0);
40      Variable AMemory : Memory;
41      BEGIN
42
43          IF WR = '0' THEN
44              Dados_M <=  AMemory(TO_INTEGER(unsigned(Constante)));
45          ELSE
46              IF RISING_EDGE(clk) THEN
47                  AMemory(TO_INTEGER(unsigned(Constante))) := Operando1;
48              END IF;
49          END IF;
50
51      END PROCESS;
52
53  END Behavioral;
54
55
```