UNIVERSIDADE da MADEIRA

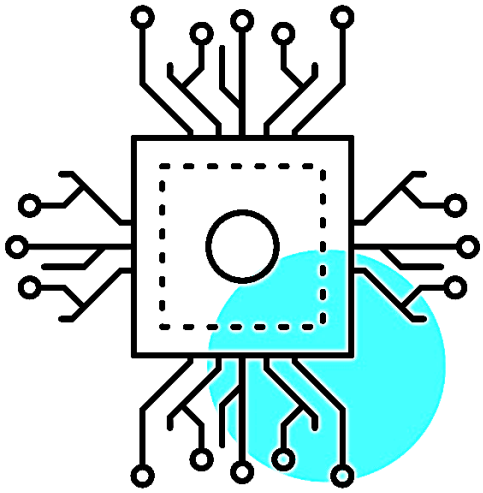
Faculdade de Ciências Exatas e da Engenharia

2019/2020

Arquitetura de Computadores

Licenciatura em Engenharia Informática

1º Projeto – Processador Básico



Trabalho realizado por:

Diego Briceño (nº 2043818)

Rúben Rodrigues (nº 2046018)

Funchal, 14 de março de 2020

Índice

1. Introdução

Este relatório apresentará os objetivos relacionados ao primeiro trabalho prático da unidade curricular de Arquitetura de Computadores assim como o seu desenvolvimento, discussão de resultados e a conclusão a que os alunos chegaram no fim.

Os processadores são as unidades centrais dos sistemas computacionais. Se comparássemos um sistema computacional a uma pessoa, o processador seria o "cérebro" pois é ele que executa as instruções de máquina, que são qualquer tarefa que o processador possa executar utilizando uma série de cálculos e decisões.

2. Objetivos

O objetivo deste trabalho é realizar um processador básico, com um conjunto mínimo de instruções, em linguagem de descrição de hardware (VHDL). Para este fim, utilizou-se o programa ISE da Xilinx com a simulação sendo efetuada no ISim e o teste em FPGA (Spartan 3E e Artix 7).

3. Desenvolvimento

O processador desenvolvido é constituído por vários módulos que, quando conectados com a memória de dados e a memória de instruções, formam a placa mãe. Cada módulo foi implementado em separado, como será descrito de seguida, de modo a facilitar a implementação, sendo só necessário no fim ligar os diferentes módulos.

3.1. Placa-mãe

3.1.1. Memória de Dados (RAM)

A RAM (Random-Access Memory) é um tipo de memória que permite a escrita/leitura de dados, individual e aleatória (como o nome indica), através do seu endereço. É uma memória volátil, ou seja, os seus dados perdem-se quando a memória perde a alimentação elétrica.

No caso desta placa-mãe, a memória de dados guarda os dados presentes no sinal de entrada ***Operando1,*** de 8 bits, quando o sinal ***WR*** está a '1' na transição ascendente do sinal de relógio (***clk***), no endereço indicado pelo sinal de entrada ***Constante***, de 8 bits. Quando o sinal ***WR*** está a '0' é feita a leitura dos dados, na posição de memória indicada por ***Constante*** e o valor lido é atribuído ao sinal de saída ***Dados_M***, de 8 bits.

3.1.2. Memória de Instruções

É neste módulo que ficam armazenas as instruções do programa a ser executado. Apresenta uma dimensão de 14 bits, onde o endereço da instrução é determinado pelo sinal ***Endereço***, de 8 bits, e à saída é disponibilizado o ***opcode***, de 5 bits, o sinal ***SEL_R***, de 1 bit, e o sinal ***Constante***, de 8 bits.

3.1.3.Processador

3.1.3.1. Periférico de Entrada

É neste módulo que é feita a comunicação do processador com o exterior, permitindo ao utilizador inserir dados para posteriormente serem realizadas operações com os mesmos. Alguns exemplos deste tipo de periféricos são o teclado e o rato.

Este módulo é controlado pelo sinal $\overline{\boldsymbol{ESCR\_P}}$, de 1 bit, que quando está a '1' é feita uma leitura dos dados de entrada, ***PIN***, de 8 bits, colocando-os na saída do periférico, ***Dados_IN***, de 8 bits.

3.1.3.2. Periférico de Saída

Este módulo permite que o utilizador veja os dados e informações processados pelo computador. Alguns exemplos deste tipo de periféricos são o monitor, a impressora e colunas de som. Este periférico é controlado pelo sinal ***ESCR_P***, de 1 bit, que quando está a '1', na transição ascendente do relógio (clk), escreve no sinal de saída, ***POUT***, de 8 bits, o valor do sinal à entrada do módulo, ***Operando1***, também de 8 bits.

3.1.3.3. Multiplexer dos Registos (Mux R)

Este módulo é responsável por encaminhar um dos quatro sinais disponíveis, de 8 bits, à sua entrada (***Resultado, Dados_IN, Dados_M e Constante***) para apresentar na sua saída, ***Dados_R***, de 8 bits. O sinal a encaminhar depende do valor do sinal de entrada ***SEL_Data***, de 2 bits.

SEL_DATA	DADOS_R
00	Resultado
01	Dados_IN
10	Dados_M
11	Constante

Tabela 1 Sinal de saída do Mux R em função do sinal SEL_DATA.

3.1.3.4. Registos A e B

A escrita nos registos A e B é controlada pelo sinal ***ESCR_R***, de 1 bit. Quando o sinal está a '1' o valor presente no sinal de entrada ***Dados_R***, de 8 bits, é guardado no registo especificado pelo sinal ***SEL_R***, de 1 bit, na transição ascendente do sinal de relógio (clk). Estes registos estão continuamente a efetuar leituras. As saídas ***Operando1*** e ***Operando2***, ambas de 8 bits, apresentam os valores guardados nos registos A e B, respetivamente.

SEL_R	Registo a ser escrito
0	Registo A
1	Registo B

Tabela 2 Sinal de seleção de escrita nos registos A e B

3.1.3.5. Unidade Aritmética e Lógica (ALU)

Este módulo permite realizar operações aritméticas e lógicas, tal como o nome indica. No caso desta placa-mãe, a unidade aritmética e lógica do processador, é capaz de realizar as operações soma, subtração, AND, OR e XOR, com os sinais de entrada ***Operando1*** e ***Operando2***, ambos de 8 bits, que representam números inteiros com sinal. Os sinais de saída da ALU são determinados pelo sinal de seleção ***SEL_ALU***, de 3 bits, A saída ***Resultado***, de 8 bits, será atualizada no caso de cada operação e, a saída ***COMP_RES***, de 5 bits, será atualizada apenas quando é realizada uma comparação, cada um dos seus bits indicando o resultado de uma das cinco comparações apresentadas no módulo *Comparação*.

SEL_ALU	Operação
000	Operando1 + Operando2
001	Operando1 – Operando2
010	Operando1 **AND** Operando2
011	Operando1 **OR** Operando2
100	Operando1 **XOR** Operando3
101	Operando1 > Operando2 Operando1 >= Operando2 Operando1 = Operando2 Operando1 <= Operando2 Operando1 < Operando2

Tabela 3 Operações da ALU

3.1.3.6. Comparação

O funcionamento deste módulo é semelhante ao módulo dos registos. Guarda o sinal de entrada, que neste caso é o sinal ***COMP_RES***, sinal de 5 bits, quando o sinal ***COMP_FLAG*** está a '1' e o sinal de relógio encontra-se na transição ascendente. Este módulo está constantemente a efetuar leituras, mas apenas um dos 5 bits do sinal guardado é encaminhado para a saída, ***S_FLAG***, de 1 bit. O sinal de seleção ***SEL_COMP***, de 3 bits, determina qual o bit guardado que está disponível na saída, do modo apresentado na tabela abaixo.

SEL_COMP	S_FLAG
000	COMP_RES(0) (>)
001	COMP_RES(1) (>=)
010	COMP_RES(2) (=)
011	COMP_RES(3) (<=)
100	COMP_RES(4) (<)

Tabela 4 Sinal de saída do multiplexer de comparação em função do sinal de seleção SEL_COMP

3.1.3.7. Contador de programa (PC)

O contador de programa indica qual é a posição atual da sequência de execução de um programa. Na transição ascendente do relógio, a saída ***Endereço***, de 8 bits, é enviada à Memória de Instruções. A sequência de execução será incrementada de um em um quando a entrada ***ESCR_PC***, de 1 bit, estiver a '0', caso contrário, a saída do contador corresponderá ao valor da entrada ***Constante***, de 8 bits, e neste caso ocorrerá um salto para o endereço de instrução indicado por este sinal. A entrada ***Reset***, de 1 bit, permite voltar ao início do programa quando ativa.

3.1.3.8. *Multiplexer* do *Program Counter* (Mux_PC)

Este *multiplexer* indica ao contador de programa se é para realizar um salto ou simplesmente incrementar o contador, através do sinal de saída ***ESCR_PC***, de 1 bit, como já foi visto no módulo do Contador de programa.

O sinal de seleção deste *multiplexer* é o sinal ***SEL_PC***, de 3 bits, que indica qual dos valores de entrada deve passar para a saída, como indicado na tabela seguinte.

SEL_PC	ESCR_PC
000	'0'
001	'1'
010	S_FLAG
011	Operando1(7)
100	NOT (Operando1(7) OR Operando1(6) OR Operando1(5) OR Operando1(4) OR Operando1(3) OR Operando1(2) OR Operando1(1) OR Operando1(0))

Tabela 5 Valor de saída do MUX_PC em função do sinal de seleção SEL_PC

3.1.3.9. ROM de descodificação (ROM)

Esta ROM é responsável por fornecer aos restantes módulos os seus sinais de controlo. Esta recebe o sinal ***opcode***, de 5 bits, da memória de instruções e coloca na sua saída os valores correspondentes aos seguintes sinais de controlo: ***SEL_PC***, ***SEL_COMP*** e ***SEL_ALU***, de 3 bits, ***SEL_Data***, de 2 bits, e os sinais ***COMP_FLAG***, ***ESCR_R***, ***ESCR_P*** e ***WR***, de 1 bit. Na tabela 6 presente no enunciado encontra-se a relação entre o sinal *opcode* e os sinais de controlo, onde cada instrução está também indicada em linguagem *assembly*. De considerar que o *Ri* corresponde ao registo indicado pelo sinal ***SEL_R***, de 1 bit.

No código é utilizado um *case* para implementar a tabela referida anteriormente.

4. Discussão de Resultados

Para saber o procedimento que o processador efetuará quando são executadas as instruções mostradas no teste do enunciado, cuja tabela encontra-se no Anexo A, foi preciso traduzir as instruções que se encontravam em linguagem *assembly* para código máquina de modo a programar a memória de instruções.

O teste referido está feito de maneira a que o processador siga as seguintes operações:

- Quando o PIN a ser introduzido representa um valor negativo, o programa realiza a operação:

  POUT = - PIN

  - ❖ Esta situação encontra-se representada na simulação presente no Anexo A, ponto 7.3.

- Quando o PIN a ser introduzido representa um valor maior ou igual a 40, o programa realiza a operação:

  POUT = PIN – 20

  - ❖ Esta situação encontra-se representada na simulação presente no Anexo A, ponto 7.4.

- Caso contrário, o PIN é um valor positivo menor que 40, o processador realiza a operação:

  POUT = 3 * PIN

  - ❖ Esta situação encontra-se representada na simulação presente no Anexo A, ponto 7.5.

O programa encontra-se esquematizado através de um fluxograma no ponto 7.2 do Anexo A.

5. Conclusão

Para concluir, este projeto permitiu aos alunos ganhar uma melhor compreensão sobre o funcionamento interno do computador, em particular o processador. As simulações no ISim e os testes nas FPGA resultaram como era esperado, confirmando-se assim o sucesso durante a elaboração deste primeiro projeto.

6. Bibliografia

J. Delgado e C. Ribeiro, Arquitectura de Computadores, FCA - Editora de Informática, 2010.

7. Anexo A

7.1. Tabela de Instruções de teste

	Endereço	Instrução (Assembly)	Instrução (código máquina)		
			Opcode	SEL_R	Constante
0	00000000	LD RA, 3	00010	0	00000011
1	00000001	ST [0], RA	00100	0	00000000
2	00000010	LD RA, 20	00010	0	00010100
3	00000011	ST [1], RA	00100	0	00000001
4	00000100	LDP RA	00000	0	XXXXXXXX
5	00000101	JN 22	10001	X	00010110
6	00000110	LD RB, 40	00010	1	00101000
7	00000111	CMP RA, RB	01010	X	XXXXXXXX
8	00001000	JGE 27	01100	X	00011011
9	00001001	ST [2], RA	00100	0	00000010
10	00001010	ST [3], RA	00100	0	00000011
11	00001011	LD RA, [0]	00011	0	00000000
12	00001100	LD RB, 1	00010	1	00000001
13	00001101	SUB RA, RB	00110	0	XXXXXXXX
14	00001110	JZ 20	10010	X	00010100
15	00001111	ST [0], RA	00100	0	00000000
16	00010000	LD RA, [3]	00011	0	00000011
17	00010001	LD RB, [2]	00011	1	00000010
18	00010010	ADD RA, RB	00101	0	XXXXXXXX
19	00010011	JMP 10	10000	X	00001010
20	00010100	LD RA [3]	00011	0	00000011
21	00010101	JMP 29	10000	X	00011101
22	00010110	LD RB, -1	00010	1	11111111
23	00010111	XOR RA, RB	01001	0	XXXXXXXX
24	00011000	LD RB, 1	00010	1	00000001
25	00011001	ADD RA, RB	00101	0	XXXXXXXX
26	00011010	JMP 29	10000	X	00011101
27	00011011	LD RB, [1]	00011	1	00000001
28	00011100	SUB RA, RB	00110	0	XXXXXXXX
29	00011101	STP RA	00001	X	XXXXXXXX
30	00011110	JMP 30	10000	X	00011110

Tabela 6 Instruções de teste do projeto

7.2. Fluxograma

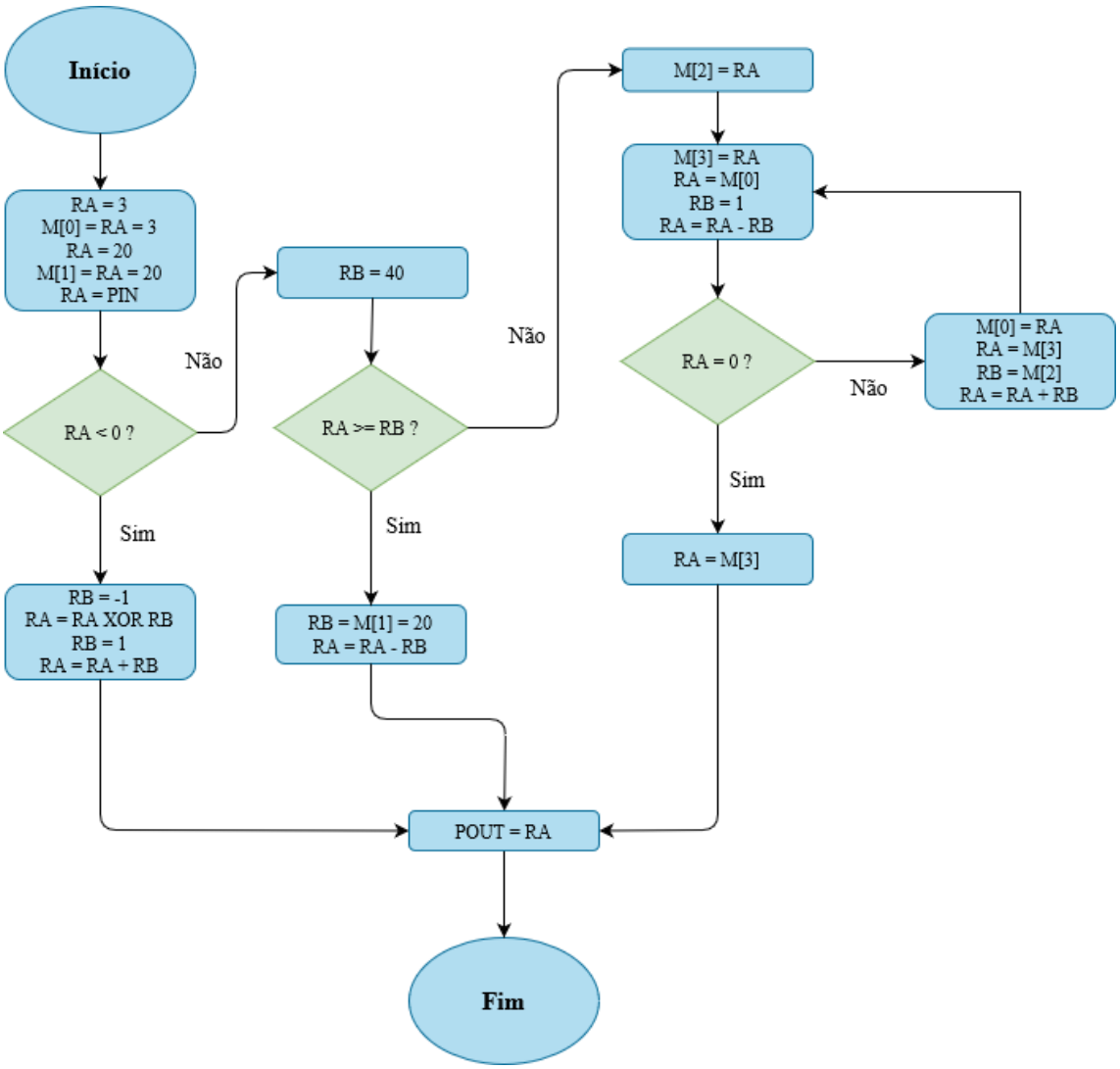


Figura 1 Fluxograma do projeto

7.3. Simulação com o PIN < 0

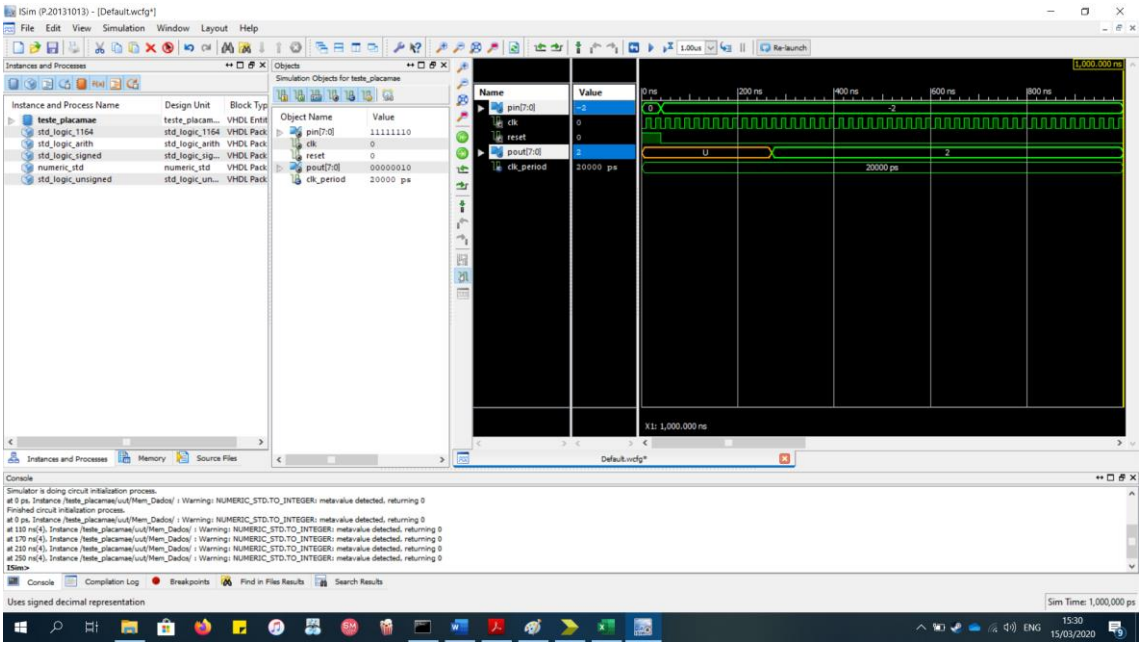


Ilustração 1 Simulação com o valor de PIN a -2

7.4. Simulação com o PIN >= 40

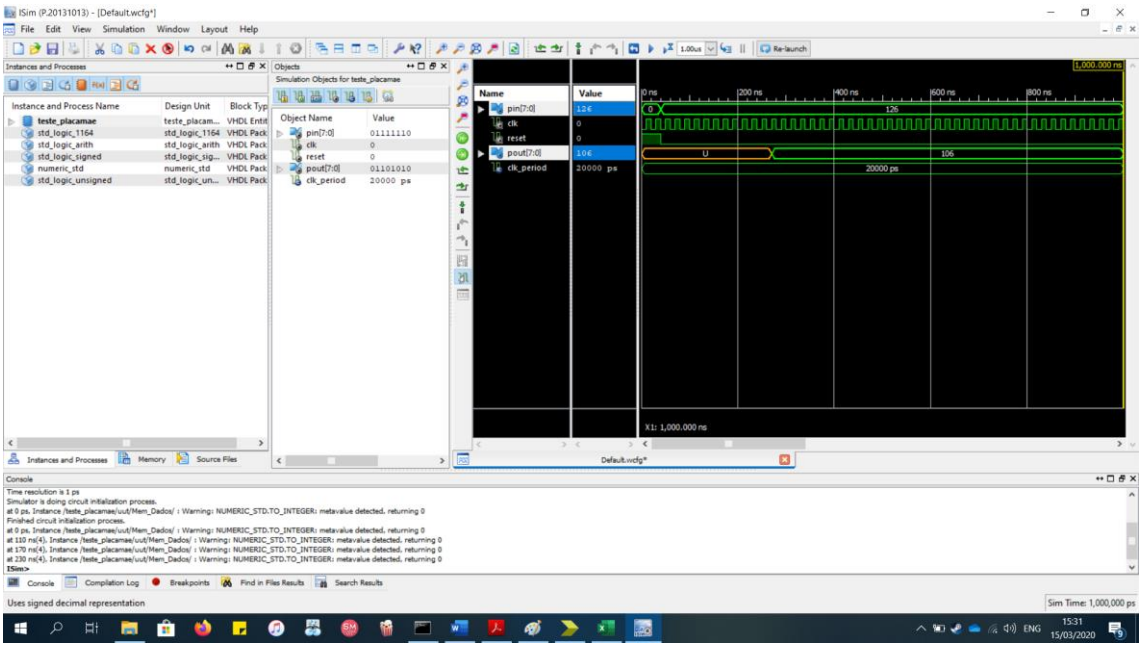


Ilustração 2 Simulação com o valor de PIN a 126

7.5. Simulação com o 0<=PIN<40

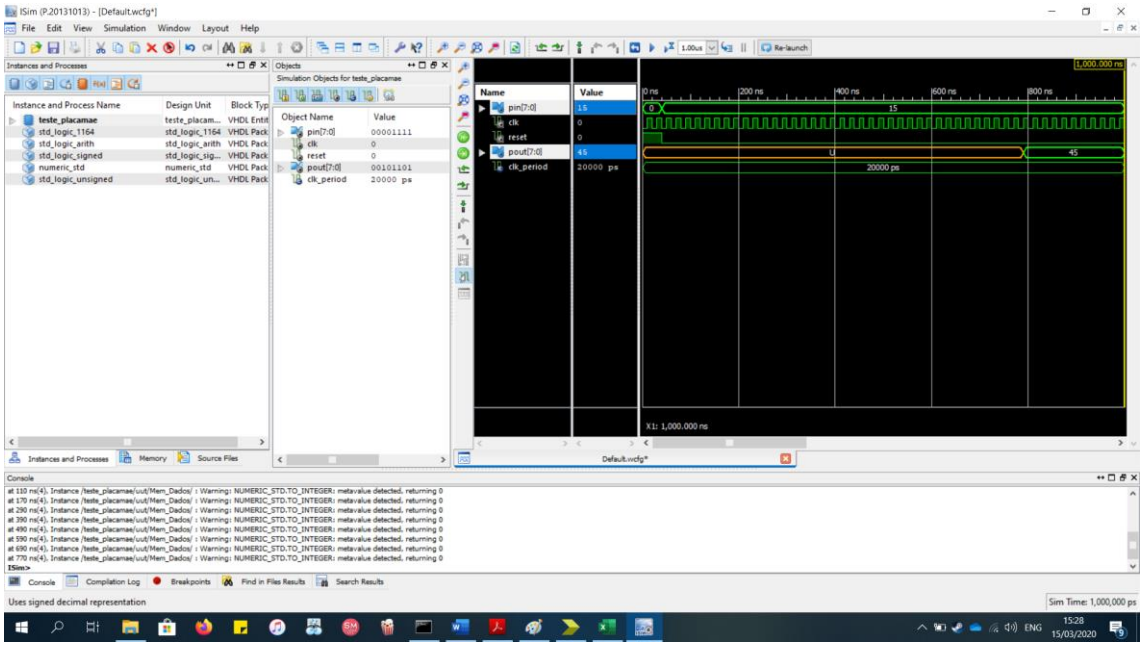


Ilustração 3 Simulação com o valor de PIN a 15

8. Anexo B

8.1. Placa-Mãe

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity PlacaMae is
   Port ( Pin : in  STD_LOGIC_VECTOR (7 downto 0);
          Clk : in  STD_LOGIC;
          Reset : in  STD_LOGIC;
          Pout : out  STD_LOGIC_VECTOR (7 downto 0));
end PlacaMae;

architecture Struct of PlacaMae is
Component Memoria_Instrucoes 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 Processador is
   Port ( Pin,Dados_M, Const : in  STD_LOGIC_VECTOR (7 downto 0);
          Clk, Reset, Sel_R : in  STD_LOGIC;
          Opcode : in  STD_LOGIC_VECTOR (4 downto 0);
          Pout, Endereco, Op1,Const_Out, ResulALU, Op2 : out  STD_LOGIC_VECTOR
(7 downto 0);
            ResComp : out  STD_LOGIC_VECTOR (4 downto 0);
            WR : out STD_LOGIC);
end Component;

Component RAM is
   Port ( Operando1, Address : in  STD_LOGIC_VECTOR (7 downto 0);
          WR, Clock : in  STD_LOGIC;
          Dados_M : out  STD_LOGIC_VECTOR (7 downto 0));
end Component;

signal Select_Reg,WriteRead: STD_LOGIC;
signal Const,Const_Out,Instrucao, Dados_Mem, Operando1, ResulALU, Op2:
STD_LOGIC_VECTOR (7 downto 0);
signal OPCode, ResComp : STD_LOGIC_VECTOR (4 downto 0);
begin
Proc: Processador Port Map(Pin, Dados_Mem, Const, Clk, Reset, Select_Reg, OPCode,
Pout, Instrucao, Operando1, Const_Out, ResulALU, Op2, ResComp, WriteRead);
Mem_Instrucs: Memoria_Instrucoes Port Map(Instrucao,OPCode,Select_Reg,Const);
Mem_Dados: RAM Port Map(Operando1,Const,WriteRead,Clk,Dados_Mem);

end Struct;
```

8.1.1. Processador

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

entity PlacaMae is
   Port ( Pin : in  STD_LOGIC_VECTOR (7 downto 0);
          Clk : in  STD_LOGIC;
          Reset : in  STD_LOGIC;
          Pout : out  STD_LOGIC_VECTOR (7 downto 0));
end PlacaMae;

architecture Struct of PlacaMae is
Component Memoria_Instrucoes 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 Processador is
   Port ( Pin,Dados_M, Const : in  STD_LOGIC_VECTOR (7 downto 0);
          Clk, Reset, Sel_R : in  STD_LOGIC;
          Opcode : in  STD_LOGIC_VECTOR (4 downto 0);
          Pout, Endereco, Op1,Const_Out, ResulALU, Op2 : out  STD_LOGIC_VECTOR
(7 downto 0);
                              ResComp : out  STD_LOGIC_VECTOR (4 downto 0);
                              WR : out STD_LOGIC);
end Component;

Component RAM is
   Port ( Operando1, Address : in  STD_LOGIC_VECTOR (7 downto 0);
          WR, Clock : in  STD_LOGIC;
          Dados_M : out  STD_LOGIC_VECTOR (7 downto 0));
end Component;

signal Select_Reg,WriteRead: STD_LOGIC;
signal Const,Const_Out,Instrucao, Dados_Mem, Operando1, ResulALU, Op2:
STD_LOGIC_VECTOR (7 downto 0);
signal OPCode, ResComp : STD_LOGIC_VECTOR (4 downto 0);
begin
Proc: Processador Port Map(Pin, Dados_Mem, Const, Clk, Reset, Select_Reg, OPCode,
Pout, Instrucao, Operando1, Const_Out, ResulALU, Op2, ResComp, WriteRead);
Mem_Instrucs: Memoria_Instrucoes Port Map(Instrucao,OPCode,Select_Reg,Const);
Mem_Dados: RAM Port Map(Operando1,Const,WriteRead,Clk,Dados_Mem);

end Struct;
```

8.1.1.1. Periférico de Entrada

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

entity Periferico_Entrada is
   Port ( ESCR_P : in  STD_LOGIC;
          PIN : in  STD_LOGIC_VECTOR (7 downto 0);
          Dados_In : out  STD_LOGIC_VECTOR (7 downto 0));
end Periferico_Entrada;

architecture Behavioral of Periferico_Entrada is
begin
          process (ESCR_P, PIN)
                    begin
                              if (ESCR_P = '0') then
                                        Dados_In <= PIN;
                              end if;
          end process;
end Behavioral;
```

8.1.1.2. Periférico de Saída

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

entity Periferico_Saida is
   Port ( Escr_P : in  STD_LOGIC;
          CLK : in  STD_LOGIC;
          Operando1 : in  STD_LOGIC_VECTOR (7 downto 0);
          POut : out  STD_LOGIC_VECTOR (7 downto 0));
end Periferico_Saida;

architecture Behavioral of Periferico_Saida is
signal temp : STD_LOGIC_VECTOR (7 downto 0);
begin
process (CLK)
begin
          if rising_edge (CLK) then
                    if (Escr_P = '1') then
                              temp <= Operando1;
                    end if;
          end if;
end process;
POut <= temp;
end Behavioral;
```

8.1.1.3. Multiplexer dos Registos (Mux R)

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

entity Mux_Registos is
   Port ( Resultado : in  STD_LOGIC_VECTOR (7 downto 0);
          Dados_IN : in  STD_LOGIC_VECTOR (7 downto 0);
          Dados_M : in  STD_LOGIC_VECTOR (7 downto 0);
          Constante : in  STD_LOGIC_VECTOR (7 downto 0);
          SEL_Data : in  STD_LOGIC_VECTOR (1 downto 0);
          Dados_Reg : out  STD_LOGIC_VECTOR (7 downto 0));
end Mux_Registos;

architecture Behavioral of Mux_Registos is
signal temp :  STD_LOGIC_VECTOR (7 downto 0);
begin
process (SEL_Data, Resultado, Dados_IN, Dados_M, Constante)
begin
          case SEL_Data is
          when "00" => temp <= Resultado;
          when "01" => temp <= Dados_IN;
          when "10" => temp <= Dados_M;
          when "11" => temp <= Constante;
          when others => temp <= (others => 'X');
          end case;
end process;
Dados_Reg <= temp;
end Behavioral;
```

8.1.1.4. Registos A e B

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

entity RegistosAeB 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);
          Operando2 : out  STD_LOGIC_VECTOR (7 downto 0));
end RegistosAeB;

architecture Behavioral of RegistosAeB is
begin
          process(ESCR_R, Dados_R, SEL_R,clk)
```

```
        begin
            if rising_edge(clk) then
                if (ESCR_R = '1') then
                    if (SEL_R = '0') then
                        Operando1 <= Dados_R;
                    else
                        Operando2 <= Dados_R;
                    end if;
                end if;
            end if;
    end process;
end Behavioral;
```

8.1.1.5. Unidade Aritmética e Lógica (ALU)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity ALU 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);
          Resultado : out  STD_LOGIC_VECTOR (7 downto 0);
          Comp_Res : out  STD_LOGIC_VECTOR (4 downto 0));
end ALU;

architecture Behavioral of ALU is
begin
    process(Operando1,Operando2,Sel_ALU)
        begin
            case Sel_ALU is
                when "000" => Resultado <= Operando1 + Operando2;
                when "001" => Resultado <= Operando1 - Operando2;
                when "010" => Resultado <= Operando1 and Operando2;
                when "011" => Resultado <= Operando1 or Operando2;
                when "100" => Resultado <= Operando1 xor Operando2;
                when "101" =>
                                    if (Operando1 > Operando2)
                                        then
                                            Comp_Res <=
(0 => '1', 1=> '1', others => '0');
                                    end if;

                                    if (Operando1 = Operando2)
                                        then
```

```
                                                    Comp_Res <=
(1=> '1',2 => '1',3 => '1', others => '0');
                                                end if;

                                                if (Operando1 < Operando2)
                                                        then
                                                    Comp_Res <=
(4 => '1',3 => '1', others => '0');
                                                end if;
                                when others => Comp_Res <= (others => 'X'); Resultado
<= (others => 'X');
                        end case;
        end process;
end Behavioral;
```

8.1.1.6. Comparação

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

entity Comparacao is
   Port ( Comp_Res : in  STD_LOGIC_VECTOR (4 downto 0);
          Comp_Flag : in  STD_LOGIC;
          Clk : in  STD_LOGIC;
          Sel_Comp : in  STD_LOGIC_VECTOR (2 downto 0);
          S_Flag : out  STD_LOGIC);
end Comparacao;

architecture Behavioral of Comparacao is
begin
process (Clk, Sel_Comp, Comp_Flag, Comp_Res)
variable mem : STD_LOGIC_VECTOR (4 downto 0);
begin
case Sel_Comp is
          when "000" => S_Flag <= mem(0);
          when "001" => S_Flag <= mem(1);
          when "010" => S_Flag <= mem(2);
          when "011" => S_Flag <= mem(3);
          when "100" => S_Flag <= mem(4);
          when others => S_Flag <= 'X';
end case;
if rising_edge (Clk)  then
                    if Comp_Flag = '1' then
                              mem := Comp_Res;
                    end if;
end if;
end process;
end Behavioral;
```

8.1.1.7. Contador de programa (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,Clock,Reset : in  STD_LOGIC;
          Endereco : out  STD_LOGIC_VECTOR (7 downto 0));
end ProgramCounter;

architecture Behavioral of ProgramCounter is
Signal contagem :  STD_LOGIC_VECTOR (7 downto 0);
begin
process (Clock)
begin
if rising_edge(Clock) then
          if (Reset = '1') then
                    contagem <= (others=>'0');
          else
                    if (ESCR_PC ='1') then
                              contagem<=Constante;
                    else
                              contagem<=contagem+"00000001";
                    end if;
          end if;
end if;
end Process;
Endereco<=contagem;
end Behavioral;
```

8.1.1.8. Multiplexer do Program Counter (Mux_PC)

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

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

architecture Behavioral of Mux_PC is
begin
          process (S_FLAG, Operando1,SEL_PC)
                    begin
```

```
                        case SEL_PC is
                                when "000" => ESCR_PC <= '0';
                                when "001" => ESCR_PC <= '1';
                                when "010" => ESCR_PC <= S_FLAG;
                                when "011" => ESCR_PC <= Operando1(7);
                                when "100" => 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 others => ESCR_PC <= 'X';
                        end case;
        end process;
end Behavioral;
```

8.1.1.9. ROM de descodificação (ROM)

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

entity ROM_Descodificacao is
   Port ( Opcode : in  STD_LOGIC_VECTOR (4 downto 0);
          Sel_ALU : out  STD_LOGIC_VECTOR (2 downto 0);
          Escr_Perif_Saida : out  STD_LOGIC;
          Sel_Data : out  STD_LOGIC_VECTOR (1 downto 0);
          Escr_Registo : out  STD_LOGIC;
          WR : out  STD_LOGIC;
          Sel_PC : out  STD_LOGIC_VECTOR (2 downto 0);
          Comp_Flag : out  STD_LOGIC;
          Sel_Comp : out  STD_LOGIC_VECTOR (2 downto 0));
end ROM_Descodificacao;

architecture Behavioral of ROM_Descodificacao is

begin

process (Opcode)

begin

case Opcode is
        -- LDP Ri
        when "00000" => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '0'; Sel_Data <=
"01";
        Escr_Registo <= '1'; WR <= '0'; Sel_PC <= "000"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        -- STP RA
        when "00001" => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '1'; Sel_Data <=
"XX";
```

```
        Escr_Registo <= '0'; WR <= '0'; Sel_PC <= "000"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        -- LD Ri, constante
        when "00010" => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '0'; Sel_Data <=
"11";
        Escr_Registo <= '1'; WR <= '0'; Sel_PC <= "000"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        -- LD Ri, [constante]
        when "00011" => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '0'; Sel_Data <=
"10";
        Escr_Registo <= '1'; WR <= '0'; Sel_PC <= "000"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        -- ST [constante], RA
        when "00100" => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '0'; Sel_Data <=
"XX";
        Escr_Registo <= '0'; WR <= '1'; Sel_PC <= "000"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        -- ADD RA,RB
        when "00101" => Sel_ALU <= "000"; Escr_Perif_Saida <= '0'; Sel_Data <= "00";
        Escr_Registo <= '1'; WR <= '0'; Sel_PC <= "000"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        -- SUB RA,RB
        when "00110" => Sel_ALU <= "001"; Escr_Perif_Saida <= '0'; Sel_Data <= "00";
        Escr_Registo <= '1'; WR <= '0'; Sel_PC <= "000"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        -- AND RA,RB
        when "00111" => Sel_ALU <= "010"; Escr_Perif_Saida <= '0'; Sel_Data <= "00";
        Escr_Registo <= '1'; WR <= '0'; Sel_PC <= "000"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        -- OR RA,RB
        when "01000" => Sel_ALU <= "011"; Escr_Perif_Saida <= '0'; Sel_Data <= "00";
        Escr_Registo <= '1'; WR <= '0'; Sel_PC <= "000"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        -- XOR RA,RB
        when "01001" => Sel_ALU <= "100"; Escr_Perif_Saida <= '0'; Sel_Data <= "00";
        Escr_Registo <= '1'; WR <= '0'; Sel_PC <= "000"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        -- CMP RA,RB
        when "01010" => Sel_ALU <= "101"; Escr_Perif_Saida <= '0'; Sel_Data <=
"XX";
        Escr_Registo <= '0'; WR <= '0'; Sel_PC <= "000"; Comp_Flag <='1'; Sel_Comp
<= "XXX";
        -- JG constante
        when "01011" => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '0'; Sel_Data <=
"XX";
        Escr_Registo <= '0'; WR <= '0'; Sel_PC <= "010"; Comp_Flag <='0'; Sel_Comp
<= "000";
```

```
        -- JGE constante
        when "01100" => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '0'; Sel_Data <=
"XX";
        Escr_Registo <= '0'; WR <= '0'; Sel_PC <= "010"; Comp_Flag <='0'; Sel_Comp
<= "001";
        -- JE constante
        when "01101" => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '0'; Sel_Data <=
"XX";
        Escr_Registo <= '0'; WR <= '0'; Sel_PC <= "010"; Comp_Flag <='0'; Sel_Comp
<= "010";
        -- JLE constante
        when "01110" => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '0'; Sel_Data <=
"XX";
        Escr_Registo <= '0'; WR <= '0'; Sel_PC <= "010"; Comp_Flag <='0'; Sel_Comp
<= "011";
        -- JL constante
        when "01111" => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '0'; Sel_Data <=
"XX";
        Escr_Registo <= '0'; WR <= '0'; Sel_PC <= "010"; Comp_Flag <='0'; Sel_Comp
<= "100";
        -- JMP constante
        when "10000" => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '0'; Sel_Data <=
"XX";
        Escr_Registo <= '0'; WR <= '0'; Sel_PC <= "001"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        -- JN RA, constante
        when "10001" => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '0'; Sel_Data <=
"XX";
        Escr_Registo <= '0'; WR <= '0'; Sel_PC <= "011"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        -- JZ RA, constante
        when "10010" => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '0'; Sel_Data <=
"XX";
        Escr_Registo <= '0'; WR <= '0'; Sel_PC <= "100"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        -- NOP
        when others => Sel_ALU <= "XXX"; Escr_Perif_Saida <= '0'; Sel_Data <=
"XX";
        Escr_Registo <= '0';   WR <= '0'; Sel_PC <= "000"; Comp_Flag <='0'; Sel_Comp
<= "XXX";
        end case;
end process;
end Behavioral;
```

8.1.2. Memória de Instruções

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

entity Memoria_Instrucoes 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 Memoria_Instrucoes;

architecture Behavioral of Memoria_Instrucoes is
begin
process (Endereco)
begin
case Endereco is
        when "00000000" => Opcode <= "00010"; Sel_R <= '0'; Constante <=
"00000011";
        when "00000001" => Opcode <= "00100"; Sel_R <= '0'; Constante <=
"00000000";
        when "00000010" => Opcode <= "00010"; Sel_R <= '0'; Constante <=
"00010100";
        when "00000011" => Opcode <= "00100"; Sel_R <= '0'; Constante <=
"00000001";
        when "00000100" => Opcode <= "00000"; Sel_R <= '0'; Constante <=
"XXXXXXXX";

        when "00000101" => Opcode <= "10001"; Sel_R <= 'X'; Constante <=
"00010110";
        when "00000110" => Opcode <= "00010"; Sel_R <= '1'; Constante <=
"00101000";
        when "00000111" => Opcode <= "01010"; Sel_R <= 'X'; Constante <=
"XXXXXXXX";
        when "00001000" => Opcode <= "01100"; Sel_R <= 'X'; Constante <=
"00011011";
        when "00001001" => Opcode <= "00100"; Sel_R <= '0'; Constante <=
"00000010";

        when "00001010" => Opcode <= "00100"; Sel_R <= '0'; Constante <=
"00000011";
        when "00001011" => Opcode <= "00011"; Sel_R <= '0'; Constante <=
"00000000";
        when "00001100" => Opcode <= "00010"; Sel_R <= '1'; Constante <=
"00000001";
        when "00001101" => Opcode <= "00110"; Sel_R <= '0'; Constante <=
"XXXXXXXX";
```

```
        when "00001110" => Opcode <= "10010"; Sel_R <= 'X'; Constante <=
"00010100";

        when "00001111" => Opcode <= "00100"; Sel_R <= '0'; Constante <=
"00000000";
        when "00010000" => Opcode <= "00011"; Sel_R <= '0'; Constante <=
"00000011";
        when "00010001" => Opcode <= "00011"; Sel_R <= '1'; Constante <=
"00000010";
        when "00010010" => Opcode <= "00101"; Sel_R <= '0'; Constante <=
"XXXXXXXX";
        when "00010011" => Opcode <= "10000"; Sel_R <= 'X'; Constante <=
"00001010";

        when "00010100" => Opcode <= "00011"; Sel_R <= '0'; Constante <=
"00000011";
        when "00010101" => Opcode <= "10000"; Sel_R <= 'X'; Constante <=
"00011101";
        when "00010110" => Opcode <= "00010"; Sel_R <= '1'; Constante <=
"11111111";
        when "00010111" => Opcode <= "01001"; Sel_R <= '0'; Constante <=
"XXXXXXXX";
        when "00011000" => Opcode <= "00010"; Sel_R <= '1'; Constante <=
"00000001";


        when "00011001" => Opcode <= "00101"; Sel_R <= '0'; Constante <=
"XXXXXXXX";
        when "00011010" => Opcode <= "10000"; Sel_R <= 'X'; Constante <=
"00011101";
        when "00011011" => Opcode <= "00011"; Sel_R <= '1'; Constante <=
"00000001";
        when "00011100" => Opcode <= "00110"; Sel_R <= '0'; Constante <=
"XXXXXXXX";
        when "00011101" => Opcode <= "00001"; Sel_R <= '0'; Constante <=
"XXXXXXXX";
        when "00011110" => Opcode <= "10000"; Sel_R <= 'X'; Constante <=
"00011110";

        when others => Opcode <= "XXXXX"; Sel_R <= 'X'; Constante <=
"XXXXXXXX";

end case;
end process;
end Behavioral;
```

8.1.3. Memória de Dados (RAM)

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

entity RAM is
   Port ( Operando1, Address : in  STD_LOGIC_VECTOR (7 downto 0);
          WR, Clock : in  STD_LOGIC;
          Dados_M : out  STD_LOGIC_VECTOR (7 downto 0));
end RAM;

architecture Behavioral of RAM is
Type mem is array(256 downto 0) of STD_LOGIC_VECTOR(7 Downto 0);
Signal Memoria : mem := (others=>(others=>'0'));
begin
process(Clock)
begin
if rising_edge(Clock) then
          if WR = '1' then
                    Memoria(to_integer(Unsigned(Address)))<=Operando1;
          end if;
end if;
end process;
Dados_M<=Memoria(to_integer(Unsigned(Address)));
end Behavioral;
```