Uma imagem com sentado, preto, computador

Descrição gerada automaticamente

**Faculdade de Ciências Exatas e da Engenharia**

**2019/2020**

**Arquitetura de Computadores**

Licenciatura em Engenharia Informática

**1º Projeto – Processador Básico**

Uma imagem com camisa

Descrição gerada automaticamente

**Trabalho realizado por:**

Diego Briceño (nº 2043818)

Rúben Rodrigues (nº 2046018)

Funchal, 14 de março de 2020

Índice

[1. Introdução 3](#_Toc35594537)

[2. Objetivos 3](#_Toc35594538)

[3. Desenvolvimento 3](#_Toc35594539)

[3.1. Placa-mãe 3](#_Toc35594540)

[3.1.1. Memória de Dados (RAM) 3](#_Toc35594541)

[3.1.2. Memória de Instruções 3](#_Toc35594542)

[3.1.3. Processador 4](#_Toc35594543)

[3.1.3.1. Periférico de Entrada 4](#_Toc35594544)

[3.1.3.2. Periférico de Saída 4](#_Toc35594545)

[3.1.3.3. Multiplexer dos Registos (Mux R) 4](#_Toc35594546)

[3.1.3.4. Registos A e B 4](#_Toc35594547)

[3.1.3.5. Unidade Aritmética e Lógica (ALU) 5](#_Toc35594548)

[3.1.3.6. Comparação 5](#_Toc35594549)

[3.1.3.7. Contador de programa (PC) 6](#_Toc35594550)

[3.1.3.8. *Multiplexer* do *Program Counter* (Mux\_PC) 6](#_Toc35594551)

[3.1.3.9. ROM de descodificação (ROM) 6](#_Toc35594552)

[4. Discussão de Resultados 7](#_Toc35594553)

[5. Conclusão 7](#_Toc35594554)

[6. Bibliografia 7](#_Toc35594555)

[7. Anexo A 8](#_Toc35594556)

[7.1. Tabela de Instruções de teste 8](#_Toc35594557)

[7.2. Fluxograma 9](#_Toc35594558)

[7.3. Simulação com o PIN < 0 10](#_Toc35594559)

[7.4. Simulação com o PIN >= 40 10](#_Toc35594560)

[7.5. Simulação com o 0<=PIN<40 11](#_Toc35594561)

[8. Anexo B 12](#_Toc35594562)

[8.1. Placa-Mãe 12](#_Toc35594563)

[8.1.1. Processador 13](#_Toc35594564)

[8.1.1.1. Periférico de Entrada 15](#_Toc35594565)

[8.1.1.2. Periférico de Saída 15](#_Toc35594566)

[8.1.1.3. Multiplexer dos Registos (Mux R) 17](#_Toc35594567)

[8.1.1.4. Registos A e B 17](#_Toc35594568)

[8.1.1.5. Unidade Aritmética e Lógica (ALU) 18](#_Toc35594569)

[8.1.1.6. Comparação 19](#_Toc35594570)

[8.1.1.7. Contador de programa (PC) 20](#_Toc35594571)

[8.1.1.8. Multiplexer do Program Counter (Mux\_PC) 20](#_Toc35594572)

[8.1.1.9. ROM de descodificação (ROM) 21](#_Toc35594573)

[8.1.2. Memória de Instruções 24](#_Toc35594574)

[8.1.3. Memória de Dados (RAM) 26](#_Toc35594575)

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.

1. **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).

1. **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.

* 1. **Placa-mãe**
     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.

* + 1. **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.

* + 1. **Processador**
       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 , 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.

* + - 1. **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.

* + - 1. **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.

* + - 1. **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

* + - 1. **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

* + - 1. **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

* + - 1. **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.

* + - 1. ***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

* + - 1. **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.

1. **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.

1. **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.

1. **Bibliografia**

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

1. **Anexo A**
   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

* 1. **Fluxograma**

**Uma imagem com iPod

Descrição gerada automaticamente**

Figura 1 Fluxograma do projeto

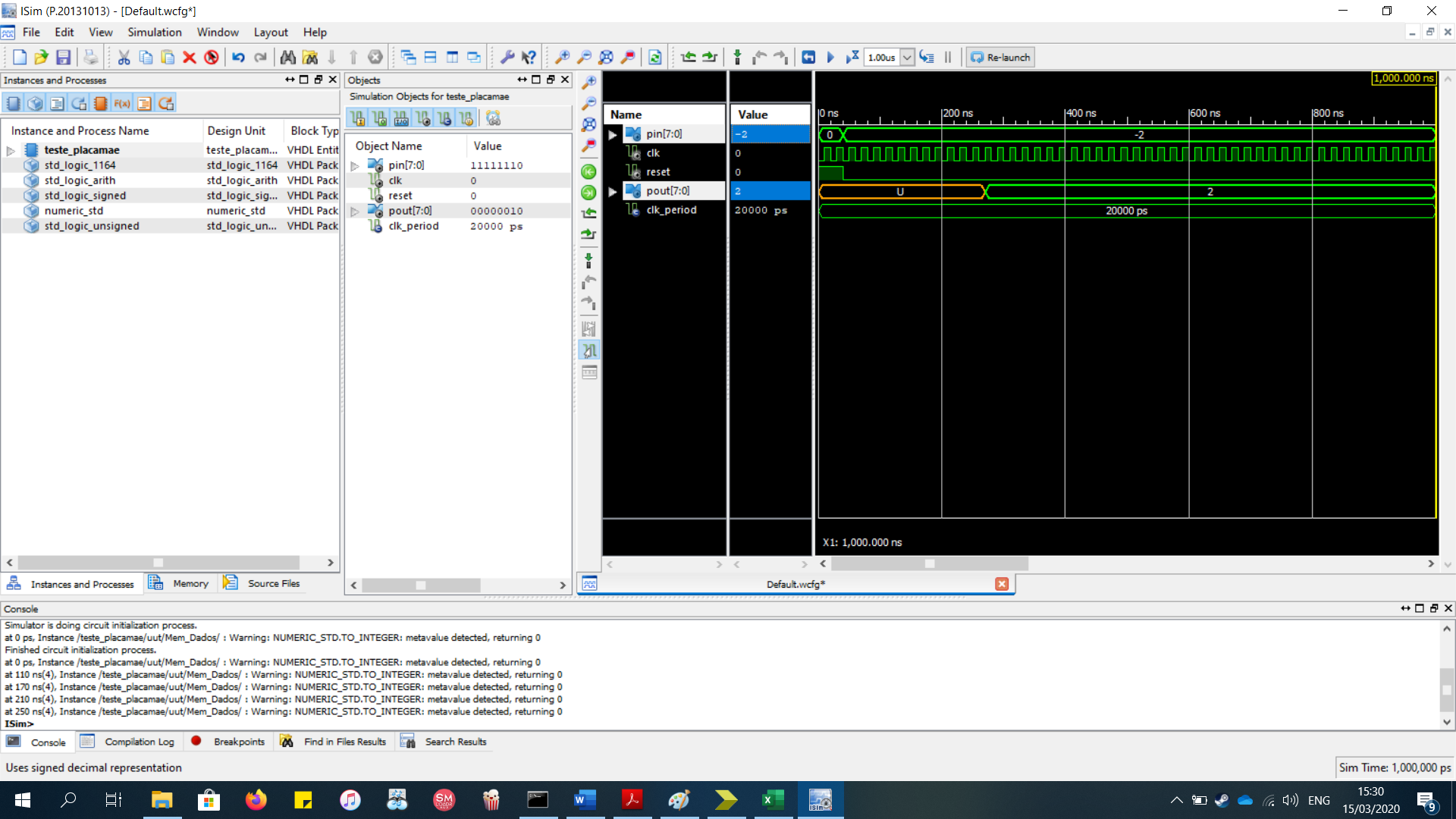
* 1. **Simulação com o PIN < 0**

Ilustração 1 Simulação com o valor de PIN a -2

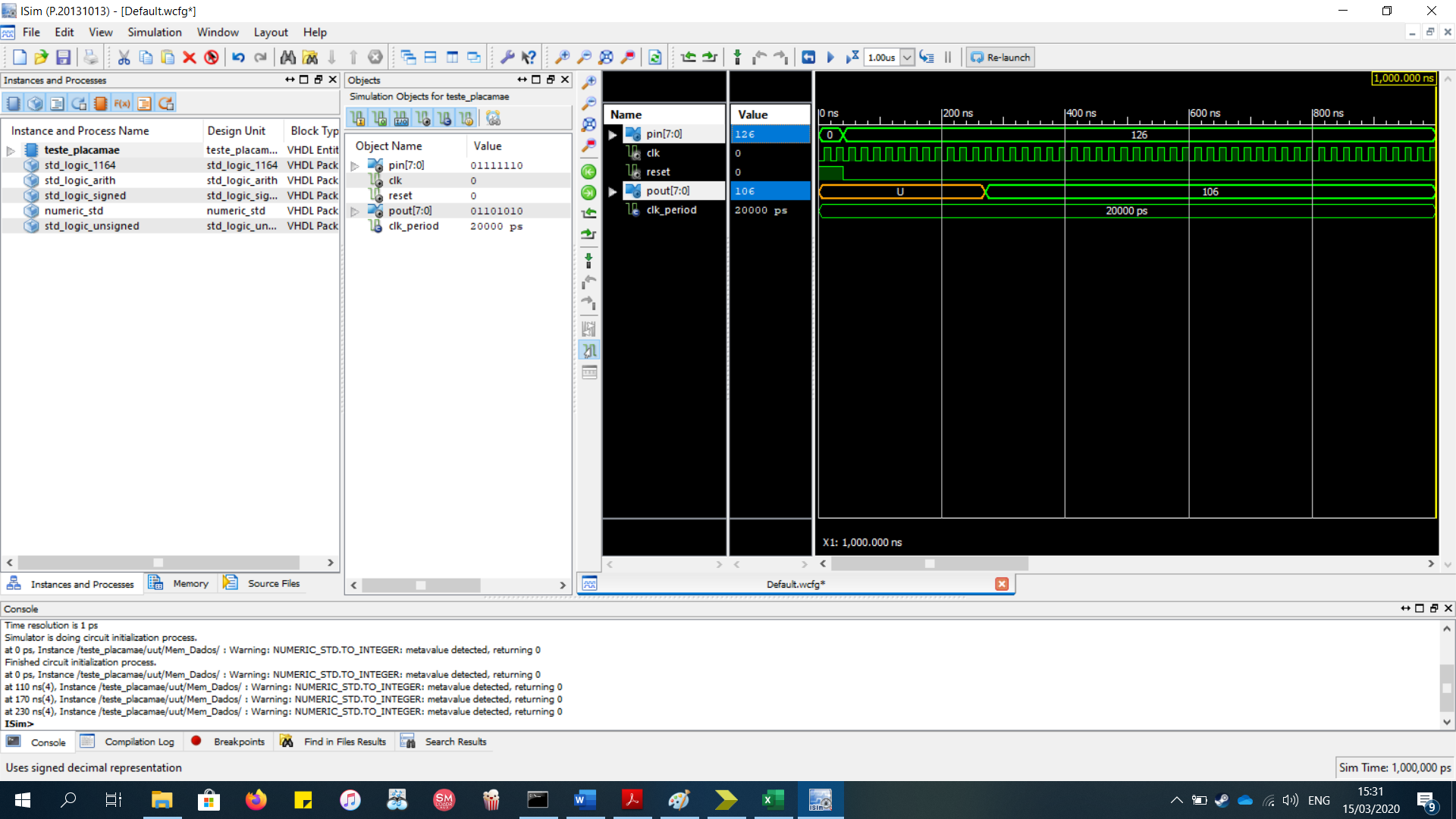
* 1. **Simulação com o PIN >= 40**

Ilustração 2 Simulação com o valor de PIN a 126

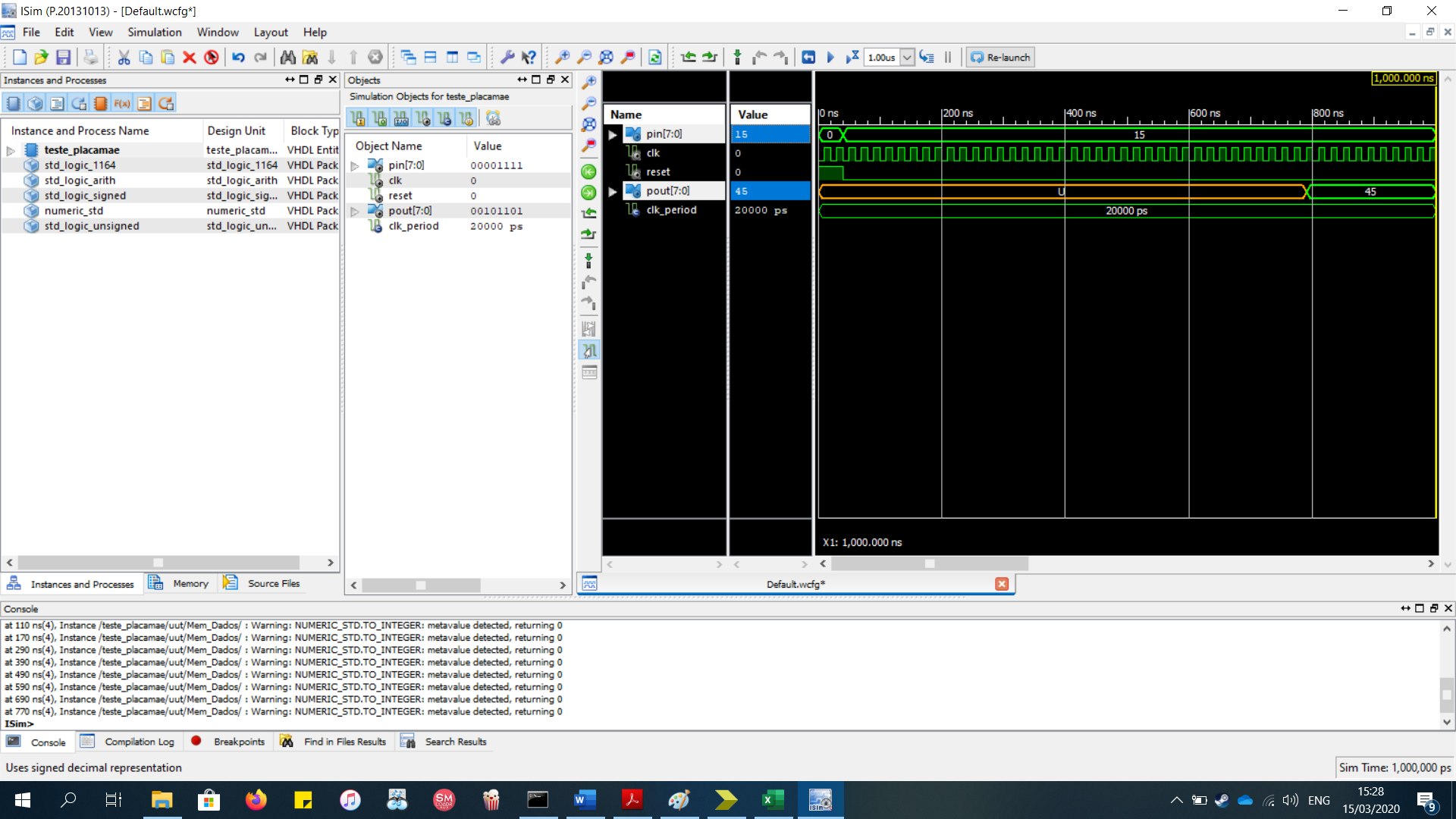
* 1. **Simulação com o 0<=PIN<40**

Ilustração 3 Simulação com o valor de PIN a 15

1. **Anexo B**
   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;

* + 1. **Processador**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity 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); --Algumas saidas sao so para propositos de teste

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 Processador;

architecture Struct of Processador is

Component 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 Component;

Component 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 Component;

Component 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 Component;

Component 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 Component;

Component 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 Component;

Component 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 Component;

Component 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 Component;

Component 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 Component;

Component 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 Component;

signal escrPerifs,escrPC, S\_Flag, Compa\_Flag,Select\_Reg,Escreve\_Reg: STD\_LOGIC;

signal DadosPIN, Ope1,Ope2, ResultadoALU,Dados\_REG: STD\_LOGIC\_VECTOR (7 downto 0);

signal Selec\_PC, Selec\_Comp, Selec\_ALU : STD\_LOGIC\_VECTOR (2 downto 0);

signal Sel\_Data : STD\_LOGIC\_VECTOR (1 downto 0);

signal Resultado\_Comparacao : STD\_LOGIC\_VECTOR (4 downto 0);

begin

PerifIn: Periferico\_Entrada Port Map(escrPerifs,Pin,DadosPIN);

PerifOut: Periferico\_Saida Port Map(escrPerifs,Clk,Ope1,Pout);

Multiplexer\_PC: Mux\_PC Port Map(S\_Flag,Ope1,Selec\_PC,escrPC);

PC: ProgramCounter Port Map(Const,escrPC,Clk, Reset, Endereco);

Comp: Comparacao Port Map(Resultado\_Comparacao,Compa\_Flag,Clk,Selec\_Comp,S\_Flag);

MUX\_Reg: Mux\_Registos Port Map(ResultadoALU,DadosPIN,Dados\_M,Const,Sel\_Data,Dados\_REG);

ArithLU: ALU Port Map(Ope1,Ope2,Selec\_ALU,ResultadoALU,Resultado\_Comparacao);

RegAeB: RegistosAeB Port Map(Escreve\_Reg,Dados\_REG,Sel\_R,Clk,Ope1,Ope2);

ROM\_Decode: ROM\_Descodificacao Port Map(Opcode,Selec\_ALU,escrPerifs,Sel\_Data,Escreve\_Reg,WR,Selec\_PC,Compa\_Flag,Selec\_Comp);

Op1 <= Ope1;

Const\_Out <= Const;

ResulALU <= ResultadoALU;

Op2 <= Ope2;

ResComp <= Resultado\_Comparacao;

end Struct;

* + - 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;

* + - 1. **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;

* + - 1. **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;

* + - 1. **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;

* + - 1. **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;

* + - 1. **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;

* + - 1. **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;

* + - 1. **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;

* + - 1. **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;

* + 1. **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;

* + 1. **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;