Arquitetura de Computadores

2020/2021

Processador em VHDL

1º Projeto Prático

Membros do grupo:

- Hugo Rocha nº 2046019;
- Sérgio Oliveira nº 2046719;

Índice

1- Introdução

Este projeto foi-nos proposto no âmbito da unidade curricular Arquitetura de Computadores. Este consiste em desenvolver e implementar um processador, no programa ISE da Xilinx, podendo assim usufruir da linguagem de descrição de hardware, conhecido por VHDL.

Assim sendo a motherboard é constituída por 3 componentes, sendo estes o processador, a memória de instruções e a memória de dados.

O processador conseguirá realizar operações aritméticas, operações lógicas e operações de comparação.

2- Objetivos

Este projeto continha alguns objetivos, sendo o principal implementar e aprender o processador PEPE-8, abordados nas aulas, que tem um conjunto de instruções a serem executadas.

Além disso, também foi importante consolidar o conhecimento adquirido nas vídeo-aulas sobre o PEPE-8 em conjunto com a linguagem de programação VHDL.

3- Desenvolvimento

Dentro deste tópico irão ser abordados os componentes que constituem a motherboard, nomeadamente a memória de dados, memória de instruções e vários componentes que são implementados para fazer um processador.

3.1- Gestor de periféricos

Permite que o utilizador insira dados antes de qualquer operação seja realizada, de modo no final das operações seja possível observar um resultado das operações realizadas pelo processador.

Tem como entradas *PIN*(8 bits), *clk*(1 bit), *ESCR_P*(1 bit*), Operando1*(8 bits) e como saídas *POUT*(8 bits) e *Dados_IN*(8 bits).

Assim este funciona da seguinte maneira, consoante o valor do sinal de controlo *ESCR_P.*

- ***ESCR_P = 1* na transição ascendente ->** escreve em *POUT*(saída) o valor de *Operando1.*
- ***ESCR_P = 0 ->*** realiza uma leitura a *PIN*, colocando este valor na saída *Dados_IN.*

3.2- Multiplexer dos Registos (Mux_R)

Tem como entradas *Dados_IN* (8 bits), *Constante* (8 bits), *Dados_M* (8bits), *Sel_D (2 bits), Resultado* (8bits) e como saída *Dados_R* (8 bits).

Através do *Sel_D,* encaminha à saída *Dados_R um dos 4 sinais de entrada.*

3.3- Banco de Registos

O banco de registos está encarregue em guardar os dados usados nas operações do processador. Distingue-se da memória de dados pois está situado dentro do próprio processador, sendo este também mais rápido, mas tendo em contrapartida uma menor capacidade.

No início de cada instrução, os dados a usar são primeiramente movidos para dentro do banco de registos. Após estes dados estarem situados no banco de registos, e terem sido realizadas as várias instruções, os dados necessários são movidos de volta para a memória de dados

3.4- Unidade Lógica e Artimérica

Tem como entradas operando1 (8 bits), operando2 (8 bits), SEL_ALU(4 bits) e tem como saída Resultado (8 bits).

Este módulo tem responsabilidade de permitir ao utilizador de inserir dados e permitir observar o resultado dos programas executados na variável de saída Resultado.

3.5- Registo de Flags

Tem como entradas ESCR_F (1 bit), SEL_F (3 bits), R_FLAG (3 bits), bit_maior_peso(1 bit), a saída do módulo IsZero, clk(1 bit) e como saída S_Flag(1 bit).

Consoante o ESCR_F e SEL_F estes indicam à saída se é para efetuar leituras continuamente que saõ mandadas para S_Flag.

3.6- Contador de programa (PC)

O contador de programa indica qual é a posição atual da sequência de execução de um programa. Assim, este envia, na transição ascendente de cada ciclo de relógio, a sua saída Endereco, de 8 bits, à Memória de Instruções. A sequência de execução será incrementada de um em um caso a entrada ESCR_PC, de 1 bit, esteja a ‘0’, caso contrário (ESCR_PC a ‘1’), a saída do PC corresponderá ao valor da entrada Constante, de 8 bits, e ocorrerá um salto para o endereço de instrução indicado por esse valor. A entrada Reset, de 1 bit, quando ativa, permite voltar ao início do programa.

3.7- Multiplexer do Program Counter (MUX_PC)

Tem como entradas os valores “0”, “1”, S_Flag (1 bit), Sel_PC (2 bits) e como saída ESCR_PC (1 bit).

A principal função deste módulo é indicar ao Program Counter se deve realizar um “jump” ou apenas incrementar o contador através do sinal ESCR_PC.

3.8- ROM de descodificação (ROM)

Apresenta como entrada o *opcode* (5 bits) e *reg* (6 bits) e como saídas *SEL_ALU* (5 bits), *ESCR_P* (1 bit), *SEL_D* (2 bit), *ESCR_R* (1 bit), *WR* (1 bit), *SEL_PC* (2 bits), *ESCR_F* (1 bit), *SEL_F* (3 bits).

Este módulo consoante o *opcode* atribui nas suas saídas certos valores.
É caracterizada por ser uma memória não volátil.

3.9- Memória de Instruções

Neste módulo ficam armazenadas as instruções do programa.

As saídas deste módulo *opcode* (5 bits), *reg* (6 bit), *Constante* (8 BITS), estão dependentes do valor de sinal de entrada *Endereco* (8 bits).

Para além do programa, criado pelos docentes desta unidade curricular e encontrado na tabela, foi também implementado um programa que visa a testar exaustivamente a várias instruções presentes no processador. Ao longo das várias instruções tivemos o cuidado de guardar uma foto do estado atual da máquina, e comparar este estado com o novo estado após implementar a nova instrução, de forma a garantir o correto funcionamento de todas as estas instruções.

Este nosso programa encontra-se na memória de instruções, mas foi comentado de forma a deixar apenas as instruções presentes na tabela, permitindo um "*debug*" mais fácil por parte do docente.

3.10- Memória de Dados (RAM)

Tem como sinais de entrada *Operando1* (8 bits), Constante (8 bits), *WR* (1 bit), *clk* (1 bit) e como saída *Dados*_M (8 bits).

Consoante o valor de WR este módulo atribui um dos valores de entrada na saída.

3.11- Is_zero

Este módulo não se encontra representado no diagrama de blocos do processador.

Após uma interpretação cuidadosa do diagrama de blocos do processador, chegámos à conclusão de que o registo de flags recebe um input dependente do valor do operando1, input este que é 1, se o operando1 for "00000000", e é 0 em qualquer outro caso.

Para implementar este comportamento no processador, decidimos criar um módulo adicional, e implementá-lo a nível comportamental. Tivemos o importante cuidado de garantir que o esquema gerado pela implementação deste módulo coincidisse com o esquema presente no diagrama, e fosse implementado fora do registo de flags. Tais valiosos cuidados garantiram que o processador mantivesse não só o comportamento, mas também a arquitetura desejada.

4- Discussão de resultados

Notas adicionais:

- No início do projeto, foi tomada a decisão de manter os nomes das variáveis e dos módulos, de forma a facilitar o trabalho dos professores na leitura do projeto.
- No entanto, devido à maior facilidade que temos a ler e escrever código em inglês, os comentários presentes no projeto não se encontram em português. Esta escolha foi feita também porque pretendemos alterar a arquitetura do PEPE-8 num futuro projeto não relacionado com a universidade, e tal decisão tem em mente vir a poupar tempo na futura conversão da língua utilizada. Foi uma decisão deliberada, e acreditamos não haver problema por parte dos docentes.

5- Conclusão

Concluindo, podemos afirmar que os objetivos requisitados foram alcançados. Deste modo, podemos também concluir que o processador foi implementado com sucesso pois realiza as instruções pretendidas.

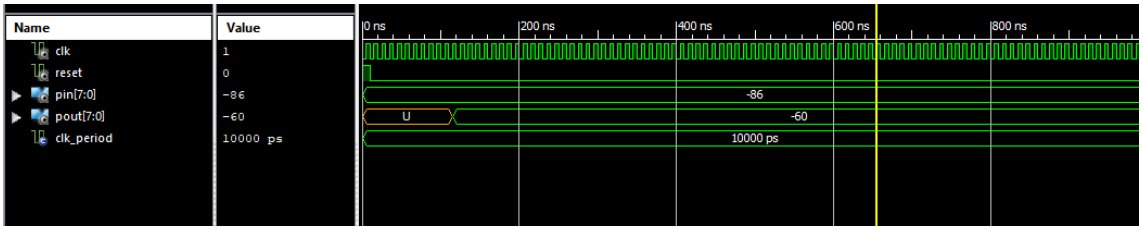
6- Bibliografia

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

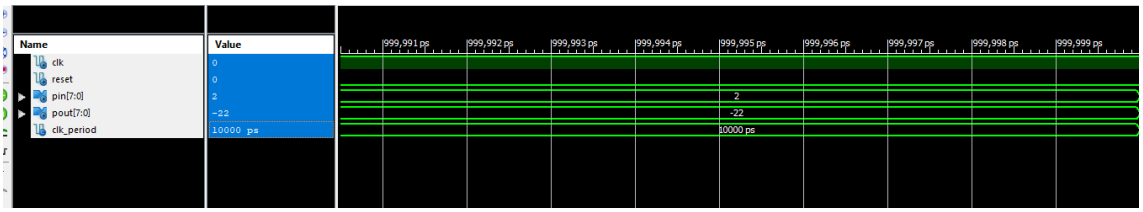
7- Anexo A

Endereço	Instrução (assembly)	Instrução (código máquina)
00000000	LD R0, 14	00010 000XXX 00001110
00000001	LD R1, 26	00010 001XXX 00011010
00000010	LD R2, -1	00010 010XXX 11111111
00000011	LD R3, 1	00010 011XXX 00000001
00000100	ST [0], R0	00100 000XXX 00000000
00000101	ST [36], R1	00100 001XXX 00100100
00000110	LDP R0	00000 000XXX XXXXXXXX
00000111	JN R0, 22	10100 000XXX 00010110
00001000	JZ R0, 15	10101 000XXX 00001111
00001001	LD R1, 29	00010 001XXX 00011101
00001010	CMP R0, R1	01111 000001 XXXXXXXX
00001011	JL 15	10010 XXXXXX 00001111
00001100	XOR R0, R2	01011 000010 XXXXXXXX
00001101	ADD R0, R3	00101 000011 XXXXXXXX
00001110	JMP 24	10011 XXXXXX 00011000
00001111	LD R4, 2	00010 100XXX 00000010
00010000	Shift Left (R0, R4)	01101 000100 XXXXXXXX
00010001	NAND R0, R2	01000 000010 XXXXXXXX
00010010	ADD R0, R3	00101 000011 XXXXXXXX
00010011	LD R5, [0]	00011 101XXX 00000000
00010100	SUB R0, R5	00110 000101 XXXXXXXX
00010101	JMP 24	10011 XXXXXX 00011000
00010110	LD R6, [36]	00011 110XXX 00100100
00010111	ADD R0, R6	00101 000110 XXXXXXXX
00011000	STP R0	00001 000XXX XXXXXXXX
00011001	JMP 25	10011 XXXXXX 00011001

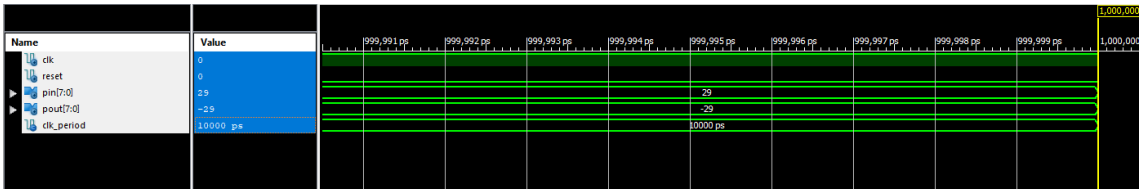
1.1-Tabela de instruções



1.2- Print sinais de entrada e saída PIN<0



1.3- Print sinais de entrada e saída $0 \leq PIN < 29$



1.4- Print sinais de entrada e saída $PIN \geq 29$

8- Anexo B

Motherboard Struct----------

-- Company:

-- Engineer:

--

-- Create Date: 16:37:25 04/07/2021

-- Design Name:

-- Module Name: motherboard - struct

-- Project Name:

-- Target Devices:

-- Tool versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY motherboard IS

  PORT (
    clk : IN STD_LOGIC;
    reset : IN STD_LOGIC;
    PIN : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    POUT : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );

END motherboard;

ARCHITECTURE struct OF motherboard IS

  COMPONENT Memoria_de_Instrucoes IS

    PORT (
      endereco : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      opcode : OUT STD_LOGIC_VECTOR (4 DOWNTO 0);
      reg : OUT STD_LOGIC_VECTOR (5 DOWNTO 0);
```

```
      constante : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
   );

END COMPONENT;

COMPONENT Memoria_de_Dados IS

   PORT (
      clk : IN STD_LOGIC;
      constante : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      operando1 : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      dados_M : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      WR : IN STD_LOGIC);

END COMPONENT;

COMPONENT Processor IS

   PORT (
      clk : IN STD_LOGIC;
      reset : IN STD_LOGIC;
      opcode : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
      reg : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
      constante : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      dados_M : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      PIN : IN STD_LOGIC_VECTOR (7 DOWNTO 0);

      endereco : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      WR : OUT STD_LOGIC;
      operando1 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      POUT : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
```

```
    );

  END COMPONENT;

  -- Memoria de instruÃ§Ãµes signals
  SIGNAL signal_endereco : STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL signal_opcode : STD_LOGIC_VECTOR(4 DOWNTO 0);
  SIGNAL signal_reg : STD_LOGIC_VECTOR(5 DOWNTO 0);
  SIGNAL signal_constante : STD_LOGIC_VECTOR(7 DOWNTO 0);

  -- Memoria de dados signals
  SIGNAL signal_operando1 : STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL signal_WR : STD_LOGIC;
  SIGNAL signal_dados_M : STD_LOGIC_VECTOR(7 DOWNTO 0);

BEGIN

  motherboard_Memoria_de_Instrucoes : Memoria_de_Instrucoes PORT
MAP(signal_endereco, signal_opcode, signal_reg, signal_constante);
  motherboard_Memoria_de_Dados : Memoria_de_Dados PORT MAP(clk, signal_constante,
signal_operando1, signal_dados_M, signal_WR);
  motherboard_processor : processor PORT MAP(clk, reset, signal_opcode, signal_reg,
signal_constante, signal_dados_M, PIN, signal_endereco, signal_WR, signal_operando1,
POUT);

END struct;
```

Memória de instruções

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    20:43:29 04/03/2021
```

```
-- Design Name:
-- Module Name:    Memoria_de_Instrucoes - 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;
USE IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY Memoria_de_Instrucoes IS

   PORT (
      endereco : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
```

```
    opcode : OUT STD_LOGIC_VECTOR (4 DOWNTO 0);
    reg : OUT STD_LOGIC_VECTOR (5 DOWNTO 0);
    constante : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );

END Memoria_de_Instrucoes;

ARCHITECTURE Behavioral OF Memoria_de_Instrucoes IS

BEGIN

  memoria_instrucoes : PROCESS (endereco)

    TYPE ram_type IS ARRAY (0 TO 255) OF STD_LOGIC_VECTOR(18 DOWNTO 0); -- Instruction
has 19 bits, 5 opcode, 3 R1, 3 R2, 8 constante
    VARIABLE ram : ram_type;

  BEGIN

    CASE (endereco) IS

      WHEN "00000000" => opcode <= "00010";
        reg <= "000XXX";
        constante <= "00001110";--LD R0, 14

      WHEN "00000001" => opcode <= "00010";
        reg <= "001XXX";
        constante <= "00011010"; --LD R1, 26

      WHEN "00000010" => opcode <= "00010";
        reg <= "010XXX";
```

```
   constante <= "11111111"; --LD R2, -1

WHEN "00000011" => opcode <= "00010";
   reg <= "011XXX";
   constante <= "00000001"; --LD R3, 1

WHEN "00000100" => opcode <= "00100";
   reg <= "000XXX";
   constante <= "00000000"; --ST [0], R0

WHEN "00000101" => opcode <= "00100";
   reg <= "001XXX";
   constante <= "00100100"; --ST [36], R1

WHEN "00000110" => opcode <= "00000";
   reg <= "000XXX";
   constante <= "XXXXXXXX"; --LDP R0

WHEN "00000111" => opcode <= "10100";
   reg <= "000XXX";
   constante <= "00010110"; --JN R0, 22

WHEN "00001000" => opcode <= "10101";
   reg <= "000XXX";
   constante <= "00001111"; --JZ R0, 15

WHEN "00001001" => opcode <= "00010";
   reg <= "001XXX";
   constante <= "00011101"; --LD R1, 29

WHEN "00001010" => opcode <= "01111";
```

```
    reg <= "000001";
    constante <= "XXXXXXXX"; --CMP R0, R1

WHEN "00001011" => opcode <= "10010";
    reg <= "XXXXXX";
    constante <= "00001111"; --JL 15

WHEN "00001100" => opcode <= "01011";
    reg <= "000010";
    constante <= "XXXXXXXX"; --XOR R0, R2

WHEN "00001101" => opcode <= "00101";
    reg <= "000011";
    constante <= "XXXXXXXX"; --ADD R0, R3

WHEN "00001110" => opcode <= "10011";
    reg <= "XXXXXX";
    constante <= "00011000"; --JMP 24

WHEN "00001111" => opcode <= "00010";
    reg <= "100XXX";
    constante <= "00000010"; --LD R4, 2

WHEN "00010000" => opcode <= "01101";
    reg <= "000100";
    constante <= "XXXXXXXX"; --shift_left(R0, R4)

WHEN "00010001" => opcode <= "01000";
    reg <= "000010";
    constante <= "XXXXXXXX"; --NAND R0, R2
```

```
WHEN "00010010" => opcode <= "00101";
   reg <= "000011";
   constante <= "XXXXXXXX"; --ADD R0, R3

WHEN "00010011" => opcode <= "00011";
   reg <= "101XXX";
   constante <= "00000000"; --LD R5, [0]

WHEN "00010100" => opcode <= "00110";
   reg <= "000101";
   constante <= "XXXXXXXX"; --SUB R0, R5

WHEN "00010101" => opcode <= "10011";
   reg <= "XXXXXX";
   constante <= "00011000"; --JMP 24

WHEN "00010110" => opcode <= "00011";
   reg <= "110XXX";
   constante <= "00100100"; -- LD R6, [36]

WHEN "00010111" => opcode <= "00101";
   reg <= "000110";
   constante <= "XXXXXXXX"; --ADD R0, R6

WHEN "00011000" => opcode <= "00001";
   reg <= "000XXX";
   constante <= "XXXXXXXX"; -- STP R0

WHEN "00011001" => opcode <= "10011";
   reg <= "XXXXXX";
   constante <= "00011001"; --JMP 25
```

```
   WHEN OTHERS => opcode <= "XXXXX";
      reg <= "XXXXXX";
      constante <= "XXXXXXXX"; --When the address that was given was none of the others

END CASE;

-- CASE(endereco) IS
--    -- NOP
--    WHEN "00000000" => opcode <= "11111";
--    reg <= "000XXX";
--    constante <= "XXXXXXXX";

--    -- ____________________________________________________________________________-

--    -- Loading constante to Ri

--    -- LD R0, -1
--    WHEN "00000001" => opcode <= "00010";
--    reg <= "000XXX";
--    constante <= "11111111";

--    -- LD R1, 1
--    WHEN "00000010" => opcode <= "00010";
--    reg <= "001XXX";
--    constante <= "00000001";

--    -- LD R2, 2
--    WHEN "00000011" => opcode <= "00010";
--    reg <= "010XXX";
--    constante <= "00000010";
```

```
--    -- LD R3, 3
--    WHEN "00000100" => opcode <= "00010";
--    reg <= "011XXX";
--    constante <= "00000011";

--    -- LD R4, 10110010
--    WHEN "00000101" => opcode <= "00010";
--    reg <= "100XXX";
--    constante <= "10110010";

--    -- ____________________________________________________________________-

--    -- Set Memoria_de_Dados[constante] to Ri

--    -- ST [0], R0
--    WHEN "00000110" => opcode <= "00100";
--    reg <= "000XXX";
--    constante <= "00000000";

--    -- ST [1], R1
--    WHEN "00000111" => opcode <= "00100";
--    reg <= "001XXX";
--    constante <= "00000001";

--    -- ST [2], R2
--    WHEN "00001000" => opcode <= "00100";
--    reg <= "010XXX";
--    constante <= "00000010";

--    -- ST [3], R3
```

```
--    WHEN "00001001" => opCode <= "00100";
--    reg <= "011XXX";
--    constante <= "00000011";

--    -- ____________________________________________________________________-

--    -- Loading peripheral input to Ri

--    -- LDP R6
--    WHEN "00001010" => opCode <= "00000";
--    reg <= "110XXX";
--    constante <= "XXXXXXXX";

--    -- LDP R7
--    WHEN "00001011" => opCode <= "00000";
--    reg <= "111XXX";
--    constante <= "XXXXXXXX";

--    -- ____________________________________________________________________-

--    -- Loading to peripheral output the value stored in Ri

--    -- STP R2
--    WHEN "00001100" => opCode <= "00001";
--    reg <= "010XXX";
--    constante <= "XXXXXXXX";

--    -- ____________________________________________________________________-

--    -- Loading Memoria_de_Dados[constante] to Ri
```

```
--    -- LD R5, [constante]
--    WHEN "00001101" => opCode <= "00011";
--    reg <= "101XXX";
--    constante <= "00000011";

--    -- ____________________________________________________________________-

--    -- Adding Ri to Rj, and saving the sum to Ri

--    -- ADD R2, R3
--    WHEN "00001110" => opCode <= "00101";
--    reg <= "010011";
--    constante <= "XXXXXXXX";

--    -- ____________________________________________________________________-

--    -- Subtract Rj to Ri, and save the subtraction to Ri

--    -- SUB R2, R0
--    WHEN "00001111" => opCode <= "00110";
--    reg <= "010000";
--    constante <= "XXXXXXXX";

--    -- SUB R3, R1
--    WHEN "00010000" => opCode <= "00110";
--    reg <= "011001";
--    constante <= "XXXXXXXX";

--    -- ____________________________________________________________________-

--    -- Ri AND Rj bit by bit, save the AND operation to Ri
```

```
--    -- R4 AND R7
--    WHEN "00010001" => opCode <= "00111";
--    reg <= "100111";
--    constante <= "XXXXXXXX";

--    -- ____________________________________________________________________-

--    -- Ri NAND Rj bit by bit, save the NAND operation to Ri

--    -- R4 NAND R7
--    WHEN "00010010" => opCode <= "01000";
--    reg <= "100111";
--    constante <= "XXXXXXXX";

--    -- ____________________________________________________________________-

--    -- Ri OR Rj bit by bit, save the OR operation to Ri

--    -- R4 OR R7
--    WHEN "00010011" => opCode <= "01001";
--    reg <= "100111";
--    constante <= "XXXXXXXX";

--    -- ____________________________________________________________________-

--    -- Ri NOR Rj bit by bit, save the NOR operation to Ri

--    -- R4 NOR R7
--    WHEN "00010100" => opCode <= "01010";
--    reg <= "100111";
```

```
--    constante <= "XXXXXXXX";

--    -- ____________________________________________________________________-

--    -- Ri XOR Rj bit by bit, save the XOR operation to Ri

--    -- R4 XOR R7
--    WHEN "00010101" => opCode <= "01011";
--    reg <= "100111";
--    constante <= "XXXXXXXX";

--    -- ____________________________________________________________________-

--    -- Ri NXOR Rj bit by bit, save the NXOR operation to Ri

--    -- R4 NXOR R7
--    WHEN "00010110" => opCode <= "01100";
--    reg <= "100111";
--    constante <= "XXXXXXXX";

--    -- ____________________________________________________________________-

--    -- Shift Ri left, Rj number of times. Save the shift to Ri

--    -- shift_left(R6, R3)
--    WHEN "00010111" => opCode <= "01101";
--    reg <= "110011";
--    constante <= "XXXXXXXX";

--    -- ____________________________________________________________________-
```

```
--  -- Shift Ri right, Rj number of times. Save the shift to Ri

--  -- shift_right(R6, R3)
--  WHEN "00011000" => opCode <= "01110";
--  reg <= "110011";
--  constante <= "XXXXXXXX";

--  -- ____________________________________________________________________-

--  -- Compare Ri to Rj. Save the result in the Registo_de_Flags

--  -- Registry Bank at the time of comparison
--  -- 0: 11111111
--  -- 1: 00000001
--  -- 2: 00000110
--  -- 3: 00000010
--  -- 4: 11111111
--  -- 5: 00000011
--  -- 6: 11101010
--  -- 7: 10101010

--  -- If greater comparisons

--  -- CMP(R1, R2)
--  WHEN "00011001" => opCode <= "01111";
--  reg <= "001010";
--  constante <= "XXXXXXXX";

--  -- JG constante. Jump to constante if Ri was greater than Rj at the time of comparison.
--  WHEN "00011010" => opCode <= "10000";
--  reg <= "XXXXXX";
```

```
--     constante <= "11111111";

--     --CMP(R2, R1)
--     WHEN "00011011" => opCode <= "01111";
--     reg <= "010001";
--     constante <= "XXXXXXXX";

--     -- JG constante. Jump to constante if Ri was greater than Rj at the time of comparison.
--     WHEN "00011100" => opCode <= "10000";
--     reg <= "XXXXXX";
--     constante <= "11111111";

--     -- If not equal to comparisons

--     -- CMP(R1, R2)
--     WHEN "00011101" => opCode <= "01111";
--     reg <= "001010";
--     constante <= "XXXXXXXX";

--     -- JNE constante. Jump to constante if Ri was not equal to Rj at the time of
comparison.
--     WHEN "00011110" => opCode <= "10000";
--     reg <= "XXXXXX";
--     constante <= "11111111";

--     -- CMP(R0, R4)
--     WHEN "00011111" => opCode <= "01111";
--     reg <= "000100";
--     constante <= "XXXXXXXX";

--     -- JNE constante. Jump to constante if Ri was not equal to Rj at the time of
comparison.
```

```
    --    WHEN "00100000" => opCode <= "10000";
    --    reg <= "XXXXXX";
    --    constante <= "11111111";

    --    -- _____-

    --    -- Debug instructions used to test if the comparation and respective jumps ocurred
correctly

    --    -- LD R7, 00000000
    --    WHEN "11111111" => opcode <= "00010";
    --    reg <= "111XXX";
    --    constante <= "00000000";
    --    WHEN OTHERS => opcode <= "XXXXX";
    --    reg <= "XXXXXX";
    --    constante <= "XXXXXXXX";
    -- END CASE;

  END PROCESS memoria_instrucoes;

END Behavioral;
```

Memória de dados

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    11:45:03 04/01/2021
-- Design Name:
-- Module Name:    Memoria_de_Dados - 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;
USE IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY Memoria_de_Dados IS

   PORT (
      clk : IN STD_LOGIC;
      constante : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      operando1 : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      WR : IN STD_LOGIC;
      dados_M : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
```

```
  );

END Memoria_de_Dados;

ARCHITECTURE Behavioral OF Memoria_de_Dados IS

BEGIN

  memoria_de_dados : PROCESS (clk, constante, operando1, WR)

    TYPE ram_type IS ARRAY (0 TO 255) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
    VARIABLE ram : ram_type;

  BEGIN

    IF rising_edge(clk) THEN

      IF (WR = '1') THEN

        ram(to_integer(unsigned(constante))) := operando1;

      END IF;

    END IF;

    dados_M <= ram(to_integer(unsigned(constante)));

  END PROCESS memoria_de_dados;

END Behavioral;
```

Processador Struct

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    20:20:06 04/05/2021
-- Design Name:
-- Module Name:    Processor - 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;
USE IEEE.STD_LOGIC_SIGNED.ALL;



-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
```

```
--use UNISIM.VComponents.all;

ENTITY Processor IS

  PORT (
    clk : IN STD_LOGIC;
    reset : IN STD_LOGIC;
    opcode : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
    reg : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
    constante : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    dados_M : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    PIN : IN STD_LOGIC_VECTOR (7 DOWNTO 0);

    endereco : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
    WR : OUT STD_LOGIC;
    operando1 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
    POUT : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );

END Processor;

ARCHITECTURE struct OF Processor IS

  -- Process counter
  COMPONENT PC IS

    PORT (
      clk : IN STD_LOGIC;
      reset : IN STD_LOGIC;
      constante : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      ESCR_PC : IN STD_LOGIC;
```

```
      endereco : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
   );

END COMPONENT;

-- ROM de descodificaÃ§Ã£o
COMPONENT Rom_de_Descodificacao IS

   PORT (
      opcode : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
      reg : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
      WR : OUT STD_LOGIC;
      escr_P : OUT STD_LOGIC;
      sel_D : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
      sel_R2 : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
      sel_R1 : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
      escr_R : OUT STD_LOGIC;
      sel_ALU : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
      escr_F : OUT STD_LOGIC;
      sel_F : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
      sel_PC : OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
   );

END COMPONENT;

-- MUX R
COMPONENT MUX_R IS

   PORT (
      sel_D : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
      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);
      dados_R : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));

END COMPONENT;

-- Banco de registos
COMPONENT banco_de_registos IS

   PORT (
      clk : IN STD_LOGIC;
      sel_R2 : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
      sel_R1 : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
      escr_R : IN STD_LOGIC;
      dados_R : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      operando1 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      operando2 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
   );

END COMPONENT;

-- MUX PC
COMPONENT MUX_PC IS

   PORT (
      S_FLAG : IN STD_LOGIC;
      SEL_PC : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
      ESCR_PC : OUT STD_LOGIC
   );
```

```
END COMPONENT;

-- Registo de flags
COMPONENT Registo_de_Flags IS

   PORT (
      clk : IN STD_LOGIC;
      is_zero : IN STD_LOGIC;
      R_FLAG : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
      ESCR_F : IN STD_LOGIC;
      bit_maior_peso : IN STD_LOGIC;
      SEL_FLAG : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
      S_FLAG : OUT STD_LOGIC
   );

END COMPONENT;

-- ALU
COMPONENT ALU IS

   PORT (
      SEL_ALU : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
      operando1 : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      operando2 : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      resultado : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      R_FLAG : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
   );

END COMPONENT;

-- Gestor de perifÃ©ricos
```

```
COMPONENT Gestor_de_Perifericos IS

   PORT (
      clk : IN STD_LOGIC;
      ESCR_P : IN STD_LOGIC;
      PIN : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      operando1 : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      POUT : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      dados_IN : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
   );

END COMPONENT;

-- Is zero
COMPONENT is_zero IS

   PORT (
      operando1 : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      is_zero : OUT STD_LOGIC
   );

END COMPONENT;

-- Process Counter signals
SIGNAL signal_ESCR_PC : STD_LOGIC;

-- Rom de descodificaÃ§Ã£o signals
SIGNAL signal_SEL_PC : STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL signal_SEL_F : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL signal_ESCR_F : STD_LOGIC;
SIGNAL signal_SEL_ALU : STD_LOGIC_VECTOR(3 DOWNTO 0);
```

```
    SIGNAL signal_ESCR_R : STD_LOGIC;
    SIGNAL signal_SEL_R1 : STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL signal_SEL_R2 : STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL signal_SEL_D : STD_LOGIC_VECTOR(1 DOWNTO 0);
    SIGNAL signal_ESCR_P : STD_LOGIC;

    -- Mux R signals
    SIGNAL signal_dados_IN : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL signal_resultado : STD_LOGIC_VECTOR(7 DOWNTO 0);
        SIGNAL signal_dados_R: STD_LOGIC_VECTOR(7 DOWNTO 0);

    -- Banco de registos signals
    SIGNAL signal_operando1 : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL signal_operando2 : STD_LOGIC_VECTOR(7 DOWNTO 0);

    -- Registo de flags signals
    SIGNAL signal_r_flag : STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL signal_bit_maior_peso : STD_LOGIC;
        SIGNAL signal_s_flag : STD_LOGIC;

    -- Is zero signals
    SIGNAL signal_is_zero : STD_LOGIC;


BEGIN

    signal_bit_maior_peso <= signal_operando1(7);
    operando1 <= signal_operando1;

    processor_PC : PC PORT MAP(clk, reset, constante, signal_ESCR_PC, endereco);
```

```
   processor_Rom_de_Descodificacao : Rom_de_Descodificacao PORT MAP(opcode, reg, WR, signal_escr_P, signal_sel_D, signal_sel_R2, signal_sel_R1, signal_ESCR_R, signal_sel_ALU, signal_escr_F, signal_sel_F, signal_SEL_PC);

   processor_MUX_R : MUX_R PORT MAP(signal_sel_D, constante, dados_M, signal_dados_IN, signal_resultado, signal_dados_R);

   processor_banco_de_registos : banco_de_registos PORT MAP(clk, signal_sel_R2, signal_sel_R1, signal_escr_R, signal_dados_R, signal_operando1, signal_operando2);

   processor_MUX_PC : MUX_PC PORT MAP(signal_S_FLAG, signal_SEL_PC, signal_ESCR_PC);

   processor_Registo_de_Flags : Registo_de_Flags PORT MAP(clk, signal_is_zero, signal_R_FLAG, signal_ESCR_F, signal_bit_maior_peso, signal_SEL_F, signal_S_FLAG);

   processor_ALU : ALU PORT MAP(signal_SEL_ALU, signal_operando1, signal_operando2, signal_resultado, signal_R_FLAG);

   processor_Gestor_de_Perifericos : Gestor_de_Perifericos PORT MAP(clk, signal_ESCR_P, PIN, signal_operando1, POUT, signal_dados_IN);

   processor_is_zero : is_zero PORT MAP(signal_operando1, signal_is_zero);


END struct;

Program Counter

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

-- Company:

-- Engineer:

--

-- Create Date:    15:05:14 03/27/2021

-- Design Name:

-- Module Name:    PC - 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;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY PC IS

  PORT (
    clk : IN STD_LOGIC;
    reset : IN STD_LOGIC;
    constante : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    ESCR_PC : IN STD_LOGIC;
    endereco : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );

END PC;

ARCHITECTURE Behavioral OF PC IS

BEGIN
```

```
PC : PROCESS (clk, reset, constante, ESCR_PC)

   VARIABLE counter : STD_LOGIC_VECTOR(7 DOWNTO 0);

BEGIN

   IF rising_edge(clk) THEN

      IF (reset = '0') THEN

         IF (ESCR_PC = '1') THEN

            counter := constante;

         ELSIF (ESCR_PC = '0') THEN

            counter := counter + 1;

         END IF;

      ELSIF (reset = '1') THEN

         counter := "00000000";

      END IF;

      endereco <= counter;

   END IF;
```

```
   END PROCESS PC;

END Behavioral;
```

Rom descodificação

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    14:56:50 04/01/2021
-- Design Name:
-- Module Name:    Rom_de_descodificacao - 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;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
```

```
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY Rom_de_Descodificacao IS

  PORT (
    opcode : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
    reg : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
    WR : OUT STD_LOGIC;
    escr_P : OUT STD_LOGIC;
    sel_D : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    sel_R2 : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    sel_R1 : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    escr_R : OUT STD_LOGIC;
    sel_ALU : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
    escr_F : OUT STD_LOGIC;
    sel_F : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    sel_PC : OUT STD_LOGIC_VECTOR (1 DOWNTO 0));

END Rom_de_Descodificacao;

ARCHITECTURE Behavioral OF Rom_de_descodificacao IS

BEGIN

    rom_descodificacao: process(opcode, reg)

    begin

      case( opcode ) is
```

```
        -- Peripherals

        when "00000" => sel_ALU <= "XXXX"; escr_P <= '0'; sel_D <= "10"; escr_R <= '1'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- LDP Ri

        when "00001" => sel_ALU <= "XXXX"; escr_P <= '1'; sel_D <= "XX"; escr_R <= '0'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- STP Ri


        -- Reading and writing

        when "00010" => sel_ALU <= "XXXX"; escr_P <= '0'; sel_D <= "00"; escr_R <= '1'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- LD Ri, constante

        when "00011" => sel_ALU <= "XXXX"; escr_P <= '0'; sel_D <= "01"; escr_R <= '1'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- LD Ri, [constante]

        when "00100" => sel_ALU <= "XXXX"; escr_P <= '0'; sel_D <= "XX"; escr_R <= '0'; WR
<= '1'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- ST [constante], Ri


        -- ALU operations

        when "00101" => sel_ALU <= "0000"; escr_P <= '0'; sel_D <= "11"; escr_R <= '1'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- Add Ri, Rj

        when "00110" => sel_ALU <= "0001"; escr_P <= '0'; sel_D <= "11"; escr_R <= '1'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- Sub Ri, Rj

        when "00111" => sel_ALU <= "0010"; escr_P <= '0'; sel_D <= "11"; escr_R <= '1'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- And Ri, Rj

        when "01000" => sel_ALU <= "0011"; escr_P <= '0'; sel_D <= "11"; escr_R <= '1'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- Nand Ri, Rj

        when "01001" => sel_ALU <= "0100"; escr_P <= '0'; sel_D <= "11"; escr_R <= '1'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- Or Ri, Rj

        when "01010" => sel_ALU <= "0101"; escr_P <= '0'; sel_D <= "11"; escr_R <= '1'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- Nor Ri, Rj

        when "01011" => sel_ALU <= "0110"; escr_P <= '0'; sel_D <= "11"; escr_R <= '1'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- Xor Ri, Rj

        when "01100" => sel_ALU <= "0111"; escr_P <= '0'; sel_D <= "11"; escr_R <= '1'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- Nxor Ri, Rj

        when "01101" => sel_ALU <= "1000"; escr_P <= '0'; sel_D <= "11"; escr_R <= '1'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- Shift_left(Ri, Rj)

        when "01110" => sel_ALU <= "1001"; escr_P <= '0'; sel_D <= "11"; escr_R <= '1'; WR
<= '0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- Shift_right(Ri, Rj)
```

```
            when "01111" => sel_ALU <= "1010"; escr_P <= '0'; sel_D <= "XX"; escr_R <= '0'; WR
<= '0'; sel_PC <= "10"; escr_F <= '1'; sel_F <= "XXX"; -- CMP Ri, Rj


            -- Jump

            when "10000" => sel_ALU <= "XXXX"; escr_P <= '0'; sel_D <= "XX"; escr_R <= '0'; WR
<= '0'; sel_PC <= "00"; escr_F <= '0'; sel_F <= "011"; -- JG constante

            when "10001" => sel_ALU <= "XXXX"; escr_P <= '0'; sel_D <= "XX"; escr_R <= '0'; WR
<= '0'; sel_PC <= "00"; escr_F <= '0'; sel_F <= "010"; -- JNE constante

            when "10010" => sel_ALU <= "XXXX"; escr_P <= '0'; sel_D <= "XX"; escr_R <= '0'; WR
<= '0'; sel_PC <= "00"; escr_F <= '0'; sel_F <= "100"; -- JL constante

            when "10011" => sel_ALU <= "XXXX"; escr_P <= '0'; sel_D <= "XX"; escr_R <= '0'; WR
<= '0'; sel_PC <= "01"; escr_F <= '0'; sel_F <= "XXX"; -- JMP constante

            when "10100" => sel_ALU <= "XXXX"; escr_P <= '0'; sel_D <= "XX"; escr_R <= '0'; WR
<= '0'; sel_PC <= "00"; escr_F <= '1'; sel_F <= "001"; -- JN Ri, constante

            when "10101" => sel_ALU <= "XXXX"; escr_P <= '0'; sel_D <= "XX"; escr_R <= '0'; WR
<= '0'; sel_PC <= "00"; escr_F <= '1'; sel_F <= "000"; -- JZ Ri, constante


            -- Other

            when others => sel_ALU <= "XXXX"; escr_P <= '0'; sel_D <= "XX"; escr_R <= '0'; WR <=
'0'; sel_PC <= "10"; escr_F <= '0'; sel_F <= "XXX"; -- NOP


        end case ;


        -- Registries

        SEL_R1(2) <= reg(5);

        SEL_R1(1) <= reg(4);

        SEL_R1(0) <= reg(3);


        SEL_R2(2) <= reg(2);

        SEL_R2(1) <= reg(1);

        SEL_R2(0) <= reg(0);



    end process rom_descodificacao;
```

```
END Behavioral;
```

Mux R

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    13:45:05 03/29/2021
-- Design Name:
-- Module Name:    MUX_R - 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;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
```

```
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY MUX_R IS

	PORT (
		SEL_D : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
		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);
		dados_R : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));

END MUX_R;

ARCHITECTURE Behavioral OF MUX_R IS

BEGIN

	MUX_R : PROCESS (SEL_D, constante, dados_M, dados_IN, resultado)

	BEGIN

		CASE (SEL_D) IS

			WHEN "00" => dados_R <= constante;
			WHEN "01" => dados_R <= dados_M;
			WHEN "10" => dados_R <= dados_IN;
			WHEN "11" => dados_R <= resultado;
			WHEN OTHERS => dados_R <= "00000000";
```

```
                END CASE;

        END PROCESS MUX_R;

END Behavioral;
```

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    14:27:24 03/30/2021
-- Design Name:
-- Module Name:    banco_de_registos - 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;
USE IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY banco_de_registos IS

  PORT (

    clk : IN STD_LOGIC;
    sel_R2 : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
    sel_R1 : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
    escr_R : IN STD_LOGIC;
    dados_R : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    operando1 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
    operando2 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)

  );

END banco_de_registos;

ARCHITECTURE Behavioral OF banco_de_registos IS

BEGIN

  banco_de_registos : PROCESS (clk, sel_R1, sel_R2, escr_R, dados_R)

    TYPE array_registos_type IS ARRAY(0 TO 7) OF STD_LOGIC_VECTOR (7 DOWNTO 0);
    VARIABLE array_registos : array_registos_type;
```

```
  BEGIN

    IF rising_edge(clk) THEN

      IF (escr_R = '1') THEN

        array_registos(to_integer(unsigned(SEL_R1))) := dados_R;

      END IF;

    END IF;

    operando1 <= array_registos(to_integer(unsigned(SEL_R1)));
    operando2 <= array_registos(to_integer(unsigned(SEL_R2)));

  END PROCESS banco_de_registos;

END Behavioral;
```

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    12:22:16 03/29/2021
-- Design Name:
-- Module Name:    MUX_PC - 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;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY MUX_PC IS

  PORT (
    S_FLAG : IN STD_LOGIC;
    SEL_PC : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
    ESCR_PC : OUT STD_LOGIC
  );

END MUX_PC;

ARCHITECTURE Behavioral OF MUX_PC IS

BEGIN
```

```
  MUX_PC : PROCESS (S_FLAG, SEL_PC)

  BEGIN

    CASE(SEL_PC) IS

      WHEN "00" => ESCR_PC <= S_FLAG;
      WHEN "01" => ESCR_PC <= '1';
      WHEN "10" => ESCR_PC <= '0';
      WHEN OTHERS => ESCR_PC <= '0';

    END CASE;

  END PROCESS MUX_PC;

END Behavioral;
```

Registo Flags

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    13:25:25 03/29/2021
-- Design Name:
-- Module Name:    Registo_de_Flags - 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;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY Registo_de_Flags IS

   PORT (
      clk : IN STD_LOGIC;
      is_zero : IN STD_LOGIC;
      R_FLAG : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
      ESCR_F : IN STD_LOGIC;
      bit_maior_peso : IN STD_LOGIC;
      SEL_FLAG : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
      S_FLAG : OUT STD_LOGIC
   );

END Registo_de_Flags;
```

```
ARCHITECTURE Behavioral OF Registo_de_Flags IS

BEGIN

  registo_de_flags : PROCESS (clk, is_zero, R_FLAG, ESCR_F, bit_maior_peso, SEL_FLAG)

    VARIABLE temp_is_zero : STD_LOGIC;
    VARIABLE temp_R_FLAG : STD_LOGIC_VECTOR (2 DOWNTO 0);
    VARIABLE temp_ESCR_F : STD_LOGIC;
    VARIABLE temp_bit_maior_peso : STD_LOGIC;
    VARIABLE temp_SEL_FLAG : STD_LOGIC_VECTOR (2 DOWNTO 0);

  BEGIN

    IF falling_edge(clk) THEN

      IF (ESCR_F = '1') THEN

        temp_is_zero := is_zero;
        temp_R_FLAG := R_FLAG;
        temp_ESCR_F := ESCR_F;
        temp_bit_maior_peso := bit_maior_peso;
        temp_SEL_FLAG := SEL_FLAG;

      END IF;

    END IF;


    CASE(SEL_FLAG) IS
```

```
        WHEN "000" => S_FLAG <= temp_is_zero;
        WHEN "001" => S_FLAG <= temp_bit_maior_peso;
        WHEN "010" => S_FLAG <= temp_R_FLAG(2);
        WHEN "011" => S_FLAG <= temp_R_FLAG(1);
        WHEN "100" => S_FLAG <= temp_R_FLAG(0);
        WHEN OTHERS => S_FLAG <= '0';

    END CASE;


  END PROCESS registo_de_flags;

END Behavioral;
```

ALU

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    10:57:12 03/27/2021
-- Design Name:
-- Module Name:    ALU - 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;
USE IEEE.STD_LOGIC_SIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY ALU IS

   PORT (
      SEL_ALU : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
      operando1 : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      operando2 : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      resultado : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      R_FLAG : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
   );

END ALU;

ARCHITECTURE Behavioral OF ALU IS
```

```
BEGIN

  ALU : PROCESS (SEL_ALU, operando1, operando2)

    VARIABLE vector : STD_LOGIC_VECTOR(7 DOWNTO 0);

  BEGIN

    CASE(SEL_ALU) IS

      -- A + B
      WHEN "0000" => vector := (operando1 + operando2);

      -- A - B
      WHEN "0001" => vector := (operando1 - operando2);

      -- A and B
      WHEN "0010" => vector := (operando1 AND operando2);

      -- A nand B
      WHEN "0011" => vector := (operando1 NAND operando2);

      -- A or B
      WHEN "0100" => vector := (operando1 OR operando2);

      -- A nor B
      WHEN "0101" => vector := (operando1 NOR operando2);

      -- A xor B
      WHEN "0110" => vector := (operando1 XOR operando2);
```

```
        -- A xnor B
        WHEN "0111" => vector := (operando1 XNOR operando2);

        -- shift_left(A, B)
        WHEN "1000" => vector := STD_LOGIC_VECTOR(shift_left(signed(operando1),
to_integer(unsigned(operando2))));

        -- shift_right(A, B)
        WHEN "1001" => vector := STD_LOGIC_VECTOR(shift_right(signed(operando1),
to_integer(unsigned(operando2))));

        -- < R_FLAG = 0; > R_FLAG = 1; /= R_FLAG = 2
        WHEN "1010" =>

        IF (operando1 < operando2) THEN
           R_FLAG(0) <= '1';
        ELSE
           R_FLAG(0) <= '0';
        END IF;

        IF (operando1 > operando2) THEN
           R_FLAG(1) <= '1';
        ELSE
           R_FLAG(1) <= '0';
        END IF;

        IF (operando1 /= operando2) THEN
           R_FLAG(2) <= '1';
        ELSE
           R_FLAG(2) <= '0';
        END IF;
```

```
        WHEN OTHERS => vector := "00000000";

    END CASE;

    resultado <= vector;

  END PROCESS ALU;

END Behavioral;
```

Gestor periféricos

```
    ----------------------------------------------------------------------------------
    -- Company:
    -- Engineer:
    --
    -- Create Date:    12:37:20 03/30/2021
    -- Design Name:
    -- Module Name:    Gestor_de_Perifericos - 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;
```

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY Gestor_de_Perifericos IS

   PORT (
      clk : IN STD_LOGIC;
      ESCR_P : IN STD_LOGIC;
      PIN : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      operando1 : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      POUT : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      dados_IN : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
   );

END Gestor_de_Perifericos;

ARCHITECTURE Behavioral OF Gestor_de_Perifericos IS

BEGIN

   gestor_de_perifericos : PROCESS (clk, ESCR_P, PIN, operando1)

   BEGIN
```

```
    IF rising_edge(clk) THEN

      IF (ESCR_P = '1') THEN

        POUT <= operando1;

      END IF;

    END IF;

    IF (ESCR_P = '0') THEN

      dados_IN <= PIN;

    END IF;

  END PROCESS gestor_de_perifericos;

END Behavioral;
```

Is_zero

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    20:56:42 04/05/2021
-- Design Name:
-- Module Name:    is_zero - 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;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY is_zero IS

   PORT (
      operando1 : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      is_zero : OUT STD_LOGIC
   );

END is_zero;

ARCHITECTURE Behavioral OF is_zero IS
```

```
BEGIN

   zero : PROCESS (operando1)

   BEGIN

      IF (operando1 = "00000000") THEN
         is_zero <= '1';

      ELSE
         is_zero <= '0';

      END IF;

   END PROCESS zero;

END Behavioral;
```