ANEXO I

CÓDIGOS DESENVOLVIDOS NO PROJETO

AluControl (Controle da Unidade Lógica-Aritmética)

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;

ENTITY AluControl IS
    PORT (
        ALU_OP : IN STD_LOGIC_VECTOR(0 TO 2);
        FUNCT : IN STD_LOGIC_VECTOR(0 TO 5);
        ULA_CODE : OUT STD_LOGIC_VECTOR(0 TO 1)
    );
END AluControl;

ARCHITECTURE AC OF AluControl IS
BEGIN
    PROCESS (ALU_OP, FUNCT)
    BEGIN
        -- I-TYPE - LW and SW
        IF (ALU_OP = "000") THEN
            ULA_CODE <= "00";
        END IF;

        -- I-TYPE - BEQ
        IF (ALU_OP = "001") THEN
            ULA_CODE <= "01";
        END IF;

        -- R-TYPE
        IF (ALU_OP = "010") THEN
            IF (FUNCT = "100000") THEN
                ULA_CODE <= "00";
            END IF;

            IF (FUNCT = "100010") THEN
                ULA_CODE <= "01";
            END IF;

            IF (FUNCT = "100100") THEN
                ULA_CODE <= "10";
            END IF;
            IF (FUNCT = "100101") THEN
                ULA_CODE <= "11";
            END IF;
        END IF;

        -- I-TYPE ARITHMETHIC
        -- SOMA
        IF (ALU_OP = "011") THEN
            ULA_CODE <= "00";
        END IF;
        -- SUB
        IF (ALU_OP = "100") THEN
            ULA_CODE <= "01";
        END IF;
        -- AND
        IF (ALU_OP = "101") THEN
            ULA_CODE <= "10";
        END IF;
        -- OR
        IF (ALU_OP = "111") THEN
            ULA_CODE <= "11";
        END IF;
    END PROCESS;
END AC;
```

ControlUnit (Unidade de Controle)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;

ENTITY ControlUnit IS
    PORT (
        -- POSITION: (0: RegWrite, 1: MEMtoREG)
        WB : OUT STD_LOGIC_VECTOR(0 TO 1) := "00";
        -- POSITION: (0: BRANCH, 1: MEMRead, 2: MEMWrite)
        MEM : OUT STD_LOGIC_VECTOR(0 TO 2) := "000";
        -- POSITION: (0: REGDst, 1,2,3: ALUop, 4: ALUsrc) (X0010)
        EX : OUT STD_LOGIC_VECTOR(0 TO 4) := "00000";
        -- SIGNAL FOR ChOOSE PC INPUT
        SIGNAL_JUMP : OUT STD_LOGIC_VECTOR(0 TO 1) := "00";

        INSTRUCTION : IN STD_LOGIC_VECTOR(0 TO 31)
    );
END ControlUnit;

ARCHITECTURE UC OF ControlUnit IS
BEGIN
    PROCESS (INSTRUCTION)
    BEGIN
        CASE INSTRUCTION(0 TO 5) IS
                --ok
            WHEN "000001" => --TYPE R
                WB <= "11";
                MEM <= "0X0";
                EX <= "10100";
                SIGNAL_JUMP <= "00";
                --ok
            WHEN "000010" => --TYPE I ADD
                WB <= "11";
                MEM <= "0X0";
                EX <= "00111";
                SIGNAL_JUMP <= "00";
                --ok
            WHEN "000011" => --TYPE I SUB
                WB <= "11";
                MEM <= "0X0";
                EX <= "01001";
                SIGNAL_JUMP <= "00";
                --ok
            WHEN "000100" => --TYPE I AND
```

```vhdl
        WB <= "11";
        MEM <= "0X0";
        EX <= "01011";
        SIGNAL_JUMP <= "00";
        --ok
    WHEN "000101" => --TYPE I OR
        WB <= "11";
        MEM <= "0X0";
        EX <= "01111";
        SIGNAL_JUMP <= "00";
        --ok
    WHEN "000110" => --LW
        WB <= "10";
        MEM <= "010";
        EX <= "00001";
        SIGNAL_JUMP <= "00";
        --ok
    WHEN "000111" => --SW
        WB <= "0X";
        MEM <= "001";
        EX <= "00001";
        SIGNAL_JUMP <= "00";
        --ok
    WHEN "001000" => --Beq
        WB <= "0X";
        MEM <= "100";
        EX <= "X0010";
        SIGNAL_JUMP <= "00";
        --ok
    WHEN "001001" => --Jump
        WB <= "00";
        MEM <= "000";
        EX <= "XXXXX";
        SIGNAL_JUMP <= "01";
        --ok
    WHEN "001010" => --Jr
        WB <= "00";
        MEM <= "000";
        EX <= "XXXXX";
        SIGNAL_JUMP <= "10";

    WHEN "000000" => -- NOP
        WB <= "00";
        MEM <= "000";
        EX <= "XXXXX";
        SIGNAL_JUMP <= "00";
    WHEN OTHERS =>
        WB <= "00";
        MEM <= "0X0";
```

```vhdl
                EX <= "XXXXX";
                SIGNAL_JUMP <= "00";
        END CASE;
    END PROCESS;
END UC;
```

PCIncrement (Incremento do Contador de Programas)

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY PCincrement IS
    PORT (
        PC : IN STD_LOGIC_VECTOR (0 TO 31); -- Our program counter
        X : OUT STD_LOGIC_VECTOR(0 TO 31) := "000000000000000000000000000
00000"); -- Where our program counter + 4 will be stored and returned
END PCincrement;

ARCHITECTURE INC OF PCincrement IS
BEGIN
    X <= PC + "00000000000000000000000000000100"; -
- Add 4 to our PC and store in X
END;
```

ProgramCounter (Contador de Programas)

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY ProgramCounter IS
    PORT (
        CLOCK : IN STD_LOGIC;
        PC_INC : IN STD_LOGIC_VECTOR(0 TO 31);
        PC : OUT STD_LOGIC_VECTOR(0 TO 31) := "00000000000000000000000000
000000"
    );
END ProgramCounter;

ARCHITECTURE PC OF ProgramCounter IS
BEGIN
    PROCESS (CLOCK, PC_INC)
    BEGIN
        IF (CLOCK'EVENT AND CLOCK = '1') THEN
            PC <= PC_INC;
        END IF;
    END PROCESS;
END;
```

ShiftLeft (Deslocamento Lógico à Esquerda)

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

ENTITY ShiftLeft IS
    PORT (
        A : IN STD_LOGIC_VECTOR (0 TO 31); -
- Our data that will be shifted
        X : OUT STD_LOGIC_VECTOR(0 TO 31) -
- Where our shifted data will be stored
    );
END ShiftLeft;

ARCHITECTURE SL OF ShiftLeft IS
BEGIN
    X <= A(2 TO 31) & "00"; -
- Concats the first 30 bits of our data with "00"
END;
```

ShiftLeft2_26to28 (Deslocamento Lógico à Esquerda 26 p/ 28 bits)

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

ENTITY ShiftLeft2_26to28 IS
    PORT (
        A : IN STD_LOGIC_VECTOR (0 TO 25); -
- Our data that will be shifted
        X : OUT STD_LOGIC_VECTOR(0 TO 27) -
- Where our shifted data will be stored
    );
END ShiftLeft2_26to28;

ARCHITECTURE SL OF ShiftLeft2_26to28 IS
BEGIN
    X <= A(0 TO 25) & "00"; -
- Concats the first 30 bits of our data with "00"
END;
```

DataMemory (Memória de Dados)

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

ENTITY DataMemory IS
    PORT (
        ADDRESS : IN STD_LOGIC_VECTOR(0 TO 31);
        CLOCK : IN STD_LOGIC;
        MEM_WRITE : IN STD_LOGIC;
        WRITE_DATA : IN STD_LOGIC_VECTOR(0 TO 31);
        MEM_READ : IN STD_LOGIC;
        READ_DATA : OUT STD_LOGIC_VECTOR(0 TO 31));

END DataMemory;

ARCHITECTURE MEM OF DataMemory IS
    TYPE MEM_TYPE IS ARRAY(0 TO 400) OF STD_LOGIC_VECTOR(0 TO 7);
    SIGNAL MEMORY : MEM_TYPE;
BEGIN
    PROCESS (CLOCK)
    BEGIN
        IF (CLOCK'EVENT AND CLOCK = '1') THEN
            IF (MEM_WRITE = '1') THEN

                MEMORY(TO_INTEGER(UNSIGNED(ADDRESS))) <= WRITE_DATA(0 TO
7);
                MEMORY(TO_INTEGER(UNSIGNED(ADDRESS)) + 1) <= WRITE_DATA(8
 TO 15);
                MEMORY(TO_INTEGER(UNSIGNED(ADDRESS)) + 2) <= WRITE_DATA(1
6 TO 23);
                MEMORY(TO_INTEGER(UNSIGNED(ADDRESS)) + 3) <= WRITE_DATA(2
4 TO 31);

            END IF;

            IF (MEM_READ = '1') THEN

                READ_DATA <= MEMORY(TO_INTEGER(UNSIGNED(ADDRESS))) &
                    MEMORY(TO_INTEGER(UNSIGNED(ADDRESS)) + 1) &
                    MEMORY(TO_INTEGER(UNSIGNED(ADDRESS)) + 2) &
                    MEMORY(TO_INTEGER(UNSIGNED(ADDRESS)) + 3);

            ELSE
```

```
                READ_DATA <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";

            END IF;
        END IF;
    END PROCESS;
END;
```

InstructionMemory (Memória de Instrução)

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

ENTITY InstructionMemory IS

    PORT (
        ADDRESS : IN STD_LOGIC_VECTOR(0 TO 31);
        INSTRUCTION : OUT STD_LOGIC_VECTOR(0 TO 31) := "00000000000000000
000000000000000"
    );

END InstructionMemory;

ARCHITECTURE MEM OF InstructionMemory IS
    TYPE MEM_TYPE IS ARRAY(0 TO 400) OF STD_LOGIC_VECTOR(0 TO 7);
    SIGNAL MEMORY : MEM_TYPE;
BEGIN
    -- LOADED WITH:

    -- ADDI $2, $2, 8
    -- SW $2, 0($4)
    -- LW $3, 0($4)
    -- ADDI $3, $6, 0
    -- JR $3

    -- Expected result: Infinite 8 times table in $2 register

    -- MEMORY(000) <= "00001000";
    -- MEMORY(001) <= "01000010";
    -- MEMORY(002) <= "00000000";
    -- MEMORY(003) <= "00001000";
    --
    -- MEMORY(004) <= "00000000";
    -- MEMORY(005) <= "00000000";
    -- MEMORY(006) <= "00000000";
    -- MEMORY(007) <= "00000000";
    --
    -- MEMORY(008) <= "00000000";
    -- MEMORY(009) <= "00000000";
    -- MEMORY(010) <= "00000000";
    -- MEMORY(011) <= "00000000";
    --
    -- MEMORY(012) <= "00000000";
    -- MEMORY(013) <= "00000000";
    -- MEMORY(014) <= "00000000";
    -- MEMORY(015) <= "00000000";
    -- MEMORY(016) <= "00011100";

    -- MEMORY(017) <= "10000010";
    -- MEMORY(018) <= "00000000";
    -- MEMORY(019) <= "00000000";
    --
    -- MEMORY(020) <= "00000000";
    -- MEMORY(021) <= "00000000";
    -- MEMORY(022) <= "00000000";
    -- MEMORY(023) <= "00000000";
    --
    -- MEMORY(024) <= "00000000";
    -- MEMORY(025) <= "00000000";
    -- MEMORY(026) <= "00000000";
    -- MEMORY(027) <= "00000000";
    --
    -- MEMORY(028) <= "00000000";
    -- MEMORY(029) <= "00000000";
    -- MEMORY(030) <= "00000000";
    -- MEMORY(031) <= "00000000";
    -- MEMORY(032) <= "00011000";

    -- MEMORY(033) <= "10000011";
    -- MEMORY(034) <= "00000000";
    -- MEMORY(035) <= "00000000";
    --
    -- MEMORY(036) <= "00000000";
    -- MEMORY(037) <= "00000000";
    -- MEMORY(038) <= "00000000";
    -- MEMORY(039) <= "00000000";
    --
    -- MEMORY(040) <= "00000000";
    -- MEMORY(041) <= "00000000";
    -- MEMORY(042) <= "00000000";
    -- MEMORY(043) <= "00000000";
    --
    -- MEMORY(044) <= "00000000";
    -- MEMORY(045) <= "00000000";
    -- MEMORY(046) <= "00000000";
    -- MEMORY(047) <= "00000000";
    --
    -- MEMORY(048) <= "00000000";
    -- MEMORY(049) <= "00000000";
    -- MEMORY(050) <= "00000000";
    -- MEMORY(051) <= "00000000";
    --
```

```
    -- MEMORY(052) <= "00000000";
    -- MEMORY(053) <= "00000000";
    -- MEMORY(054) <= "00000000";
    -- MEMORY(055) <= "00000000";
    --
    -- MEMORY(056) <= "00000000";
    -- MEMORY(057) <= "00000000";
    -- MEMORY(058) <= "00000000";
    -- MEMORY(059) <= "00000000";

    -- MEMORY(060) <= "00001000";
    -- MEMORY(061) <= "11000011";
    -- MEMORY(062) <= "00000000";
    -- MEMORY(063) <= "00000000";
    --
    -- MEMORY(064) <= "00000000";
    -- MEMORY(065) <= "00000000";
    -- MEMORY(066) <= "00000000";
    -- MEMORY(067) <= "00000000";
    --
    -- MEMORY(068) <= "00000000";
    -- MEMORY(069) <= "00000000";
    -- MEMORY(070) <= "00000000";
    -- MEMORY(071) <= "00000000";
    --
    -- MEMORY(072) <= "00000000";
    -- MEMORY(073) <= "00000000";
    -- MEMORY(074) <= "00000000";
    -- MEMORY(075) <= "00000000";
    --
    -- MEMORY(076) <= "00000000";
    -- MEMORY(077) <= "00000000";
    -- MEMORY(078) <= "00000000";
    -- MEMORY(079) <= "00000000";

    -- MEMORY(080) <= "00101000";
    -- MEMORY(081) <= "01100000";
    -- MEMORY(082) <= "00000000";
    -- MEMORY(083) <= "00000000";

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

    -- LOADED WITH:

    -- ORI $1, $1, 5
    -- ADDI $3, $3, 7
    -- J NEXT_INSTR
    -- SUBI $4, $1, 1
    -- NEXT_INSTR:
    -- AND $3, $3, $1

    -- Expected result: 5 in register 3 and 0 in register 2

    MEMORY(000) <= "00010100";
    MEMORY(001) <= "00100001";
    MEMORY(002) <= "00000000";
    MEMORY(003) <= "00000101";

    MEMORY(004) <= "00000000";
    MEMORY(005) <= "00000000";
    MEMORY(006) <= "00000000";
    MEMORY(007) <= "00000000";

    MEMORY(008) <= "00000000";
    MEMORY(009) <= "00000000";
    MEMORY(010) <= "00000000";
    MEMORY(011) <= "00000000";

    MEMORY(012) <= "00000000";
    MEMORY(013) <= "00000000";
    MEMORY(014) <= "00000000";
    MEMORY(015) <= "00000000";

    MEMORY(016) <= "00000000";
    MEMORY(017) <= "00000000";
    MEMORY(018) <= "00000000";
    MEMORY(019) <= "00000000";

    MEMORY(020) <= "00001000";
    MEMORY(021) <= "01100011";
    MEMORY(022) <= "00000000";
    MEMORY(023) <= "00000111";

    MEMORY(024) <= "00000000";
    MEMORY(025) <= "00000000";
    MEMORY(026) <= "00000000";
    MEMORY(027) <= "00000000";

    MEMORY(028) <= "00000000";
    MEMORY(029) <= "00000000";
    MEMORY(030) <= "00000000";
    MEMORY(031) <= "00000000";

    MEMORY(032) <= "00000000";
    MEMORY(033) <= "00000000";
    MEMORY(034) <= "00000000";
    MEMORY(035) <= "00000000";
    MEMORY(036) <= "00100100";
    MEMORY(037) <= "00000000";
    MEMORY(038) <= "00000000";
    MEMORY(039) <= "00010011";

    MEMORY(040) <= "00000000";
    MEMORY(041) <= "00000000";
    MEMORY(042) <= "00000000";
    MEMORY(043) <= "00000000";

    MEMORY(044) <= "00000000";
    MEMORY(045) <= "00000000";
    MEMORY(046) <= "00000000";
    MEMORY(047) <= "00000000";

    MEMORY(048) <= "00000000";
    MEMORY(049) <= "00000000";
    MEMORY(050) <= "00000000";
    MEMORY(051) <= "00000000";

    MEMORY(052) <= "00000000";
    MEMORY(053) <= "00000000";
    MEMORY(054) <= "00000000";
    MEMORY(055) <= "00000000";

    MEMORY(056) <= "00001100";
    MEMORY(057) <= "00100010";
    MEMORY(058) <= "00000000";
    MEMORY(059) <= "00000001";

    MEMORY(060) <= "00000000";
    MEMORY(061) <= "00000000";
    MEMORY(062) <= "00000000";
    MEMORY(063) <= "00000000";

    MEMORY(064) <= "00000000";
    MEMORY(065) <= "00000000";
    MEMORY(066) <= "00000000";
    MEMORY(067) <= "00000000";

    MEMORY(068) <= "00000000";
    MEMORY(069) <= "00000000";
    MEMORY(070) <= "00000000";
    MEMORY(071) <= "00000000";

    MEMORY(072) <= "00000000";
    MEMORY(073) <= "00000000";
    MEMORY(074) <= "00000000";
```

```
    MEMORY(075) <= "00000000";

    MEMORY(076) <= "00000000";
    MEMORY(077) <= "00000000";
    MEMORY(078) <= "00000000";
    MEMORY(079) <= "00000000";

    MEMORY(080) <= "00000100";
    MEMORY(081) <= "01100001";
    MEMORY(082) <= "00011000";
    MEMORY(083) <= "00100100";

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

    -- LOADED WITH:

    --      ADDI $1, $1, 5
    -- ADD $2, $2, $1
    -- LOOP:
    --      ADDI $1, $1, 5
    --      ADDI $3, $3, 1
    --      BEQ $3, $2, J_SUB
    --      J LOOP
    -- J_SUB:
    --      SUB $1, $3, $1

    -- Expected result: 25 in $1 register

    -- MEMORY(000) <= "00001000";
    -- MEMORY(001) <= "00100001";
    -- MEMORY(002) <= "00000000";
    -- MEMORY(003) <= "00000101";
    --
    -- MEMORY(004) <= "00000000";
    -- MEMORY(005) <= "00000000";
    -- MEMORY(006) <= "00000000";
    -- MEMORY(007) <= "00000000";
    --
    -- MEMORY(008) <= "00000000";
    -- MEMORY(009) <= "00000000";
    -- MEMORY(010) <= "00000000";
    -- MEMORY(011) <= "00000000";
    --
    -- MEMORY(012) <= "00000000";
    -- MEMORY(013) <= "00000000";
    -- MEMORY(014) <= "00000000";
    -- MEMORY(015) <= "00000000";
    --
```

```
-- MEMORY(016) <= "00000100";
-- MEMORY(017) <= "01000001";
-- MEMORY(018) <= "00010000";
-- MEMORY(019) <= "00100000";
--
-- MEMORY(020) <= "00000000";
-- MEMORY(021) <= "00000000";
-- MEMORY(022) <= "00000000";
-- MEMORY(023) <= "00000000";
--
-- MEMORY(024) <= "00000000";
-- MEMORY(025) <= "00000000";
-- MEMORY(026) <= "00000000";
-- MEMORY(027) <= "00000000";
--
-- MEMORY(028) <= "00000000";
-- MEMORY(029) <= "00000000";
-- MEMORY(030) <= "00000000";
-- MEMORY(031) <= "00000000";
--
-- MEMORY(032) <= "00001000";
-- MEMORY(033) <= "00100001";
-- MEMORY(034) <= "00000000";
-- MEMORY(035) <= "00000101";
--
-- MEMORY(036) <= "00000000";
-- MEMORY(037) <= "00000000";
-- MEMORY(038) <= "00000000";
-- MEMORY(039) <= "00000000";
--
-- MEMORY(040) <= "00000000";
-- MEMORY(041) <= "00000000";
-- MEMORY(042) <= "00000000";
-- MEMORY(043) <= "00000000";
--
-- MEMORY(044) <= "00000000";
-- MEMORY(045) <= "00000000";
-- MEMORY(046) <= "00000000";
-- MEMORY(047) <= "00000000";
--
-- MEMORY(048) <= "00001000";
-- MEMORY(049) <= "01100011";
-- MEMORY(050) <= "00000000";
-- MEMORY(051) <= "00000001";
--
-- MEMORY(052) <= "00000000";
-- MEMORY(053) <= "00000000";
-- MEMORY(054) <= "00000000";
-- MEMORY(055) <= "00000000";
```

```
--
-- MEMORY(056) <= "00000000";
-- MEMORY(057) <= "00000000";
-- MEMORY(058) <= "00000000";
-- MEMORY(059) <= "00000000";
--
-- MEMORY(060) <= "00000000";
-- MEMORY(061) <= "00000000";
-- MEMORY(062) <= "00000000";
-- MEMORY(063) <= "00000000";
--
-- MEMORY(064) <= "00100000";
-- MEMORY(065) <= "01100010";
-- MEMORY(066) <= "00000000";
-- MEMORY(067) <= "00000110";
--
-- MEMORY(068) <= "00000000";
-- MEMORY(069) <= "00000000";
-- MEMORY(070) <= "00000000";
-- MEMORY(071) <= "00000000";
--
-- MEMORY(072) <= "00000000";
-- MEMORY(073) <= "00000000";
-- MEMORY(074) <= "00000000";
-- MEMORY(075) <= "00000000";
--
-- MEMORY(076) <= "00000000";
-- MEMORY(077) <= "00000000";
-- MEMORY(078) <= "00000000";
-- MEMORY(079) <= "00000000";
--
-- MEMORY(080) <= "00000000";
-- MEMORY(081) <= "00000000";
-- MEMORY(082) <= "00000000";
-- MEMORY(083) <= "00000000";
--
-- MEMORY(084) <= "00100100";
-- MEMORY(085) <= "00000000";
-- MEMORY(086) <= "00000000";
-- MEMORY(087) <= "00000111";
--
-- MEMORY(088) <= "00000000";
-- MEMORY(089) <= "00000000";
-- MEMORY(090) <= "00000000";
-- MEMORY(091) <= "00000000";
--
-- MEMORY(092) <= "00000000";
-- MEMORY(093) <= "00000000";
-- MEMORY(094) <= "00000000";
```

```
    -- MEMORY(095) <= "00000000";
    --
    -- MEMORY(096) <= "00000100";
    -- MEMORY(097) <= "00100010";
    -- MEMORY(098) <= "00001000";
    -- MEMORY(099) <= "00100010";

    PROCESS (ADDRESS)
    BEGIN
        INSTRUCTION <= MEMORY(TO_INTEGER(UNSIGNED(ADDRESS))) &
            MEMORY(TO_INTEGER(UNSIGNED(ADDRESS)) + 1) &
            MEMORY(TO_INTEGER(UNSIGNED(ADDRESS)) + 2) &
            MEMORY(TO_INTEGER(UNSIGNED(ADDRESS)) + 3);
    END PROCESS;
END;
```

Alu (Unidade Lógica-Aritmética)

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

ENTITY Alu IS
    PORT (
        A : IN STD_LOGIC_VECTOR(0 TO 31); -- REGISTER
        B : IN STD_LOGIC_VECTOR(0 TO 31); -- REGISTER
        ALU_CODE : IN STD_LOGIC_VECTOR(0 TO 1); -
- CODE OF THE ARITHMETIC OPERATION OPTION
        ALU_OUT : OUT STD_LOGIC_VECTOR(0 TO 31); -
- WHERE WE WILL STORE THE RESULT OF THE ARITHMETIC OPERATION
        ZERO : OUT STD_LOGIC);
END Alu;

ARCHITECTURE ALU OF Alu IS
    SIGNAL AUX : STD_LOGIC_VECTOR(0 TO 31);
BEGIN
    PROCESS (A, B, ALU_CODE)
    BEGIN
        -- SWITCH CASE TO DO THE RIGHT OPERATION BASED IN ALU_CODE
        CASE ALU_CODE IS
            WHEN "00" => AUX <= A + B;
            WHEN "01" => AUX <= A - B;
            WHEN "10" => AUX <= A AND B;
            WHEN "11" => AUX <= A OR B;
            WHEN OTHERS => AUX <= "00000000000000000000000000000000";
        END CASE;
        IF (AUX = "00000000000000000000000000000000") THEN
            ZERO <= '1';
        ELSE
            ZERO <= '0';
        END IF;
        ALU_OUT <= AUX;
    END PROCESS;

END ALU;
```

SignExtend (Extensor de Sinal)

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

ENTITY SignExtend IS
    PORT (
        A : IN STD_LOGIC_VECTOR (0 TO 15); -- Data that will be extended
        X : OUT STD_LOGIC_VECTOR(0 TO 31)); -
- Where the extended data will be stored
END SignExtend;

ARCHITECTURE SE OF SignExtend IS
BEGIN
    -
- Resize our data (16 bits) to the X lenght (32 bits) and stores it in X
    X <= STD_LOGIC_VECTOR(RESIZE(SIGNED(A), X'LENGTH));
END;
```

Mux_2to1_5b (Multiplexador 2 para 1)

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

ENTITY Mux_2to1_5b IS
    GENERIC (DATA_SIZE : INTEGER := 5); -
- Generic data size to ensure that we can receive any size data
    PORT (
        CONTROL : IN STD_LOGIC; -- Controller to select the desired data
        A : IN STD_LOGIC_VECTOR (0 TO DATA_SIZE - 1); -
- The first data option
        B : IN STD_LOGIC_VECTOR (0 TO DATA_SIZE - 1); -
- The second data option
        X : OUT STD_LOGIC_VECTOR (0 TO DATA_SIZE - 1)); -
- Will be the selected data
END Mux_2to1_5b;

ARCHITECTURE MUX OF Mux_2to1_5b IS
BEGIN
    -- If control equals 0 then A, else B
    X <= A WHEN (CONTROL = '0') ELSE B;
END MUX;
```

Mux_2to1_32b (Multiplexador 2 para 1)

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

ENTITY Mux_2to1_32b IS
    GENERIC (DATA_SIZE : INTEGER := 32); -
- Generic data size to ensure that we can receive any size data
    PORT (
        CONTROL : IN STD_LOGIC; -- Controller to select the desired data
        A : IN STD_LOGIC_VECTOR (0 TO DATA_SIZE - 1); -
- The first data option
        B : IN STD_LOGIC_VECTOR (0 TO DATA_SIZE - 1); -
- The second data option
        X : OUT STD_LOGIC_VECTOR (0 TO DATA_SIZE - 1)); -
- Will be the selected data
END Mux_2to1_32b;

ARCHITECTURE MUX OF Mux_2to1_32b IS
BEGIN
    -- If control equals 0 then A, else B
    X <= A WHEN (CONTROL = '0') ELSE B;
END;
```

Generic3to1Mux (Multiplexador 3 para 1)

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

ENTITY Generic3to1Mux IS
    GENERIC (DATA_SIZE : INTEGER := 32); -
- Generic data size to ensure that we can receive any size data
    PORT (
        JUMP_SIGNAL : IN STD_LOGIC_VECTOR (0 TO 1); -
- Controller to select the desired data
        A, B, C : IN STD_LOGIC_VECTOR (0 TO DATA_SIZE - 1); -
- The first,second and third data option
        X : OUT STD_LOGIC_VECTOR (0 TO DATA_SIZE - 1)); -
- Will be the selected data
END Generic3to1Mux;

ARCHITECTURE MUX OF Generic3to1Mux IS
BEGIN
    WITH JUMP_SIGNAL SELECT
        -- If control equals 0 then A, If equal 1 then B, else C
        X <= A WHEN "00",
        B WHEN "01",
        C WHEN OTHERS;
END MUX;
```

FileRegister (Registrador de Arquivos)

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

ENTITY FileRegister IS
    PORT (
        -- IN --
        REGWRITE : IN STD_LOGIC;
        CLOCK : IN STD_LOGIC;
        READ_REGISTER_1 : IN STD_LOGIC_VECTOR(0 TO 4);
        READ_REGISTER_2 : IN STD_LOGIC_VECTOR(0 TO 4);
        WRITE_REGISTER : IN STD_LOGIC_VECTOR(0 TO 4);
        WRITE_DATA : IN STD_LOGIC_VECTOR(0 TO 31);
        -- OUT --
        READ_DATA_1 : OUT STD_LOGIC_VECTOR(0 TO 31);
        READ_DATA_2 : OUT STD_LOGIC_VECTOR(0 TO 31);
        DEB_FILE_REG_1 : OUT STD_LOGIC_VECTOR(0 TO 31);
        DEB_FILE_REG_2 : OUT STD_LOGIC_VECTOR(0 TO 31);
        DEB_FILE_REG_3 : OUT STD_LOGIC_VECTOR(0 TO 31);
        DEB_FILE_REG_AUX : OUT STD_LOGIC);
END FileRegister;

ARCHITECTURE REGS OF FileRegister IS
    TYPE REGISTER_TYPE IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL REGISTERS : REGISTER_TYPE;
BEGIN
    DEB_FILE_REG_1 <= REGISTERS(1);
    DEB_FILE_REG_2 <= REGISTERS(2);
    DEB_FILE_REG_3 <= REGISTERS(3);
    PROCESS (CLOCK)
    BEGIN
        IF (CLOCK'EVENT AND CLOCK = '1' AND REGWRITE = '1' AND NOT (WRITE
_REGISTER = "00000")) THEN
            DEB_FILE_REG_AUX <= '1';
            REGISTERS(TO_INTEGER(UNSIGNED(WRITE_REGISTER))) <= WRITE_DATA
;
        END IF;
    END PROCESS;
    READ_DATA_1 <= REGISTERS(TO_INTEGER(UNSIGNED(READ_REGISTER_1)));
    READ_DATA_2 <= REGISTERS(TO_INTEGER(UNSIGNED(READ_REGISTER_2)));
END;
```

Reg_Pipe_IFID (Registrador de Pipeline - Estágio IF/ID)

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

-
- This component implements a pipeline register, in this case (is the) fi
rst register of the first stage
ENTITY Reg_Pipe_IFID IS
    PORT (
        -- IN --
        CLOCK : IN STD_LOGIC;
        IN_PC_MAIS_4 : IN STD_LOGIC_VECTOR(0 TO 31); -- PC Increment
        IN_INSTR_MEM : IN STD_LOGIC_VECTOR(0 TO 31); -- Instruction
        -- OUT --
        OUT_PC_MAIS_4 : OUT STD_LOGIC_VECTOR(0 TO 31) := "0000000000000000
0000000000000000"; -- Out PC Increment
        OUT_INSTR_MEM : OUT STD_LOGIC_VECTOR(0 TO 31) := "0000000000000000
0000000000000000"); -- Out instruction
END Reg_Pipe_IFID;

ARCHITECTURE REG_PIPE OF Reg_Pipe_IFID IS
BEGIN
    PROCESS (CLOCK)
    BEGIN
        IF (CLOCK'EVENT AND CLOCK = '1') THEN
            OUT_INSTR_MEM <= IN_INSTR_MEM;
            OUT_PC_MAIS_4 <= IN_PC_MAIS_4;
        END IF;
    END PROCESS;
END;
```

Reg_Pipe_IDEX (Registrador de Pipeline - Estágio ID/EX)

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

ENTITY Reg_Pipe_IDEX IS
    PORT (
        -- IN --
        CLOCK : IN STD_LOGIC;
        IDEX_IN_WB : IN STD_LOGIC_VECTOR(0 TO 1); -
- POSITION: (0 - RegWrite, 1 - MEMtoREG)
        IDEX_IN_MEM : IN STD_LOGIC_VECTOR(0 TO 2); -
- POSITION: (0 - BRANCH, 1 - MEMRead, 2- MEMWrite)
        IDEX_IN_EX : IN STD_LOGIC_VECTOR(0 TO 4); -
- POSITION: (0 - REGDst, 1,2 - ALUOp(2bits), 3 - ALUSrc)
        IDEX_IN_PC : IN STD_LOGIC_VECTOR(0 TO 31); -- PC+4
        IDEX_IN_READ1 : IN STD_LOGIC_VECTOR(0 TO 31); -- READ 1
        IDEX_IN_READ2 : IN STD_LOGIC_VECTOR(0 TO 31); -- READ 2
        IDEX_IN_IMED : IN STD_LOGIC_VECTOR(0 TO 31); -- IMMEDIATE
        IDEX_IN_RT : IN STD_LOGIC_VECTOR(0 TO 4); -- RT REGISTER ID
        IDEX_IN_RD : IN STD_LOGIC_VECTOR(0 TO 4); -- RD REGISTER ID
        -- OUT --
        IDEX_OUT_WB : OUT STD_LOGIC_VECTOR(0 TO 1) := "00";
        IDEX_OUT_MEM : OUT STD_LOGIC_VECTOR(0 TO 2) := "000";
        IDEX_OUT_EX : OUT STD_LOGIC_VECTOR(0 TO 4) := "00000";
        IDEX_OUT_PC : OUT STD_LOGIC_VECTOR(0 TO 31) := "00000000000000000
000000000000000";
        IDEX_OUT_READ1 : OUT STD_LOGIC_VECTOR(0 TO 31) := "00000000000000
000000000000000000";
        IDEX_OUT_READ2 : OUT STD_LOGIC_VECTOR(0 TO 31) := "00000000000000
000000000000000000";
        IDEX_OUT_IMED : OUT STD_LOGIC_VECTOR(0 TO 31) := "000000000000000
00000000000000000";
        IDEX_OUT_RT : OUT STD_LOGIC_VECTOR(0 TO 4) := "00000";
        IDEX_OUT_RD : OUT STD_LOGIC_VECTOR(0 TO 4) := "00000");
END Reg_Pipe_IDEX;

ARCHITECTURE REG_PIPE OF Reg_Pipe_IDEX IS

BEGIN
    PROCESS (CLOCK)
    BEGIN
        IF (CLOCK'EVENT AND CLOCK = '1') THEN
            IDEX_OUT_WB <= IDEX_IN_WB;
            IDEX_OUT_MEM <= IDEX_IN_MEM;
```

```
                IDEX_OUT_EX <= IDEX_IN_EX;
                IDEX_OUT_PC <= IDEX_IN_PC;
                IDEX_OUT_READ1 <= IDEX_IN_READ1;
                IDEX_OUT_READ2 <= IDEX_IN_READ2;
                IDEX_OUT_IMED <= IDEX_IN_IMED;
                IDEX_OUT_RT <= IDEX_IN_RT;
                IDEX_OUT_RD <= IDEX_IN_RD;
            END IF;
        END PROCESS;
    END;
```

Reg_Pipe_EXMEM (Registrador de Pipeline - Estágio EX/MEM)

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY Reg_Pipe_EXMEM IS
    PORT (
        -- IN --
        CLOCK : IN STD_LOGIC;
        EXMEM_IN_ZERO : IN STD_LOGIC; -
- ALU RESULTS EQUALS TO ZERO SIGNAL
        EXMEM_IN_WB : IN STD_LOGIC_VECTOR(0 TO 1); -
- POSITION: (0 - RegWrite, 1 - MEMtoREG)
        EXMEM_IN_MEM : IN STD_LOGIC_VECTOR(0 TO 2); -
- POSITION: (0 - BRANCH, 1- MEMRead, 2 - MEMWrite)
        EXMEM_IN_RESULT_ADDER : IN STD_LOGIC_VECTOR(0 TO 31); -
- ADDER RESULT
        EXMEM_IN_RESULT_ULA : IN STD_LOGIC_VECTOR(0 TO 31); -- ALU RESULT
        EXMEM_IN_READ2 : IN STD_LOGIC_VECTOR(0 TO 31); -- READ
        EXMEM_IN_REGDST : IN STD_LOGIC_VECTOR(0 TO 4); -
- EXIT MUX, RT OR RD
        -- OUT --
        EXMEM_OUT_ZERO : OUT STD_LOGIC;
        EXMEM_OUT_WB : OUT STD_LOGIC_VECTOR(0 TO 1);
        EXMEM_OUT_MEM : OUT STD_LOGIC_VECTOR(0 TO 2);
        EXMEM_OUT_RESULT_ADDER : OUT STD_LOGIC_VECTOR(0 TO 31);
        EXMEM_OUT_RESULT_ULA : OUT STD_LOGIC_VECTOR(0 TO 31);
        EXMEM_OUT_READ2 : OUT STD_LOGIC_VECTOR(0 TO 31);
        EXMEM_OUT_REGDST : OUT STD_LOGIC_VECTOR(0 TO 4));
END Reg_Pipe_EXMEM;

ARCHITECTURE REG_PIPE OF Reg_Pipe_EXMEM IS

BEGIN
    PROCESS (CLOCK)
    BEGIN
        IF (CLOCK'EVENT AND CLOCK = '1') THEN
            EXMEM_OUT_ZERO <= EXMEM_IN_ZERO;
            EXMEM_OUT_WB <= EXMEM_IN_WB;
            EXMEM_OUT_MEM <= EXMEM_IN_MEM;
            EXMEM_OUT_RESULT_ADDER <= EXMEM_IN_RESULT_ADDER;
            EXMEM_OUT_RESULT_ULA <= EXMEM_IN_RESULT_ULA;
            EXMEM_OUT_READ2 <= EXMEM_IN_READ2;
            EXMEM_OUT_REGDST <= EXMEM_IN_REGDST;
        END IF;
    END PROCESS;
END REG_PIPE;
```

Reg_Pipe_MEMWB (Registrador de Pipeline - Estágio MEM/WB)

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY Reg_Pipe_MEMWB IS
    PORT (
        -- IN --
        CLOCK : IN STD_LOGIC;
        MEMWB_IN_WB : IN STD_LOGIC_VECTOR(0 TO 1); -
- (0 - RegWrite, 1 - MemToReg)
        MEMWB_IN_RESULT_ULA : IN STD_LOGIC_VECTOR(0 TO 31); -- ALU RESULT
        MEMWB_IN_REGDST : IN STD_LOGIC_VECTOR(0 TO 4); -
- The ID of destination register
        MEMWB_IN_READ_DATA : IN STD_LOGIC_VECTOR(0 TO 31); -
- Read data from data memory
        -- OUT --
        MEMWB_OUT_WB : OUT STD_LOGIC_VECTOR(0 TO 1); -
- (0 - RegWrite, 1 - MemToReg)
        MEMWB_OUT_RESULT_ULA : OUT STD_LOGIC_VECTOR(0 TO 31); -
- ALU RESULT
        MEMWB_OUT_REGDST : OUT STD_LOGIC_VECTOR(0 TO 4); -
- The ID of destination register
        MEMWB_OUT_READ_DATA : OUT STD_LOGIC_VECTOR(0 TO 31)); -
- Read data from data memory
END Reg_Pipe_MEMWB;

ARCHITECTURE A OF Reg_Pipe_MEMWB IS
BEGIN
    PROCESS (CLOCK)
    BEGIN
        IF (CLOCK'EVENT AND CLOCK = '1') THEN
            MEMWB_OUT_WB <= MEMWB_IN_WB;
            MEMWB_OUT_RESULT_ULA <= MEMWB_IN_RESULT_ULA;
            MEMWB_OUT_REGDST <= MEMWB_IN_REGDST;
            MEMWB_OUT_READ_DATA <= MEMWB_IN_READ_DATA;
        END IF;
    END PROCESS;
END A;
```

Cpu (UNIDADE CENTRAL DE PROCESSAMENTO)

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY Cpu IS
    GENERIC (DATA_SIZE : INTEGER := 32); -
- Generic data size to map in components
    PORT (
        CLOCK : IN STD_LOGIC;
        INSTRUCTION_OUT_IFID : OUT STD_LOGIC_VECTOR(0 TO 31);
        DEB_REGS_PC : OUT STD_LOGIC_VECTOR(0 TO 31);
        DEB_CONTROL : OUT STD_LOGIC;
        DEB_ULA_IN_1 : OUT STD_LOGIC_VECTOR(0 TO 31);
        DEB_ULA_IN_2 : OUT STD_LOGIC_VECTOR(0 TO 31);
        DEB_OUT_ULA : OUT STD_LOGIC_VECTOR(0 TO 31);
        DEB_RegDst : OUT STD_LOGIC_VECTOR(0 TO 4);

        DEB_REG_ULA_IN_1 : OUT STD_LOGIC_VECTOR(0 TO 4);

        DEB_SINAL_MUX_MEMWB : OUT STD_LOGIC;
        DEB_SINAL_REG_WRITE : OUT STD_LOGIC;
        DEB_WRITE_REG : OUT STD_LOGIC_VECTOR(0 TO 4);
        DEB_WRITE_DATA : OUT STD_LOGIC_VECTOR(0 TO 31);

        DEB_FILE_REG_1 : OUT STD_LOGIC_VECTOR(0 TO 31);
        DEB_FILE_REG_2 : OUT STD_LOGIC_VECTOR(0 TO 31);
        DEB_FILE_REG_3 : OUT STD_LOGIC_VECTOR(0 TO 31);
        DEB_FILE_REG_AUX : OUT STD_LOGIC
    );
END Cpu;
ARCHITECTURE CPU OF Cpu IS

    -- ControlUnit - PORT MAP
    COMPONENT ControlUnit
        PORT (
            WB : OUT STD_LOGIC_VECTOR(0 TO 1);
            MEM : OUT STD_LOGIC_VECTOR(0 TO 2);
            EX : OUT STD_LOGIC_VECTOR(0 TO 4);
            SIGNAL_JUMP : OUT STD_LOGIC_VECTOR(0 TO 1);
            INSTRUCTION : IN STD_LOGIC_VECTOR(0 TO 31)
        );
    END COMPONENT;

    -- AluControl - PORT MAP
```

```vhdl
COMPONENT AluControl
    PORT (
        ALU_OP : IN STD_LOGIC_VECTOR(0 TO 2);
        FUNCT : IN STD_LOGIC_VECTOR(0 TO 5);
        ULA_CODE : OUT STD_LOGIC_VECTOR(0 TO 1)
    );
END COMPONENT;

-- ShiftLeft - PORT MAP
COMPONENT ShiftLeft
    PORT (
        A : IN STD_LOGIC_VECTOR (0 TO 31);
        X : OUT STD_LOGIC_VECTOR(0 TO 31)
    );
END COMPONENT;

-- ShiftLeft2_26to28 - PORT MAP
COMPONENT ShiftLeft2_26to28
    PORT (
        A : IN STD_LOGIC_VECTOR (0 TO 25);
        X : OUT STD_LOGIC_VECTOR(0 TO 27)
    );
END COMPONENT;

-- Signal Extend - PORT MAP
COMPONENT SignalExtend
    PORT (
        A : IN STD_LOGIC_VECTOR (0 TO 15);
        X : OUT STD_LOGIC_VECTOR(0 TO 31)
    );
END COMPONENT;

-- Mux_2to1_32b - PORT MAP
COMPONENT Mux_2to1_32b
    PORT (
        CONTROL : IN STD_LOGIC;
        A : IN STD_LOGIC_VECTOR (0 TO DATA_SIZE - 1);
        B : IN STD_LOGIC_VECTOR (0 TO DATA_SIZE - 1);
        X : OUT STD_LOGIC_VECTOR (0 TO DATA_SIZE - 1)
    );
END COMPONENT;

--  Generic3to1Mux - PORT MAP
COMPONENT Generic3to1Mux
    PORT (
        JUMP_SIGNAL : IN STD_LOGIC_VECTOR (0 TO 1);
        A, B, C : IN STD_LOGIC_VECTOR (0 TO DATA_SIZE - 1);

        X : OUT STD_LOGIC_VECTOR (0 TO DATA_SIZE - 1)
        );
    END COMPONENT;

    -- Alu - PORT MAP
    COMPONENT Alu
        PORT (
            A : IN STD_LOGIC_VECTOR(0 TO 31);
            B : IN STD_LOGIC_VECTOR(0 TO 31);
            ALU_CODE : IN STD_LOGIC_VECTOR(0 TO 1);
            ALU_OUT : OUT STD_LOGIC_VECTOR(0 TO 31);
            ZERO : OUT STD_LOGIC
        );
    END COMPONENT;

    -- Data Memory - PORT MAP
    COMPONENT DataMemory
        PORT (
            ADDRESS : IN STD_LOGIC_VECTOR(0 TO 31);
            CLOCK : IN STD_LOGIC;
            MEM_WRITE : IN STD_LOGIC;
            WRITE_DATA : IN STD_LOGIC_VECTOR(0 TO 31);
            MEM_READ : IN STD_LOGIC;
            READ_DATA : OUT STD_LOGIC_VECTOR(0 TO 31)
        );
    END COMPONENT;

    -- InstructionMemory - PORT MAP
    COMPONENT InstructionMemory
        PORT (
            ADDRESS : IN STD_LOGIC_VECTOR(0 TO 31);
            INSTRUCTION : OUT STD_LOGIC_VECTOR(0 TO 31) := "0000000000000
0000000000000000000"
        );
    END COMPONENT;

    -- PCIncrement - PORT MAP
    COMPONENT PCIncrement
        PORT (
            PC : IN STD_LOGIC_VECTOR (0 TO 31);
            X : OUT STD_LOGIC_VECTOR(0 TO 31)
        );
    END COMPONENT;

    -- Reg_Pipe_IFID - PORT MAP
    COMPONENT Reg_Pipe_IFID
        PORT (
            CLOCK : IN STD_LOGIC;
            IN_PC_MAIS_4 : IN STD_LOGIC_VECTOR(0 TO 31);
            IN_INSTR_MEM : IN STD_LOGIC_VECTOR(0 TO 31);
            OUT_PC_MAIS_4 : OUT STD_LOGIC_VECTOR(0 TO 31);
            OUT_INSTR_MEM : OUT STD_LOGIC_VECTOR(0 TO 31)
        );
    END COMPONENT;

    -- Mux_2to1_5b - PORT MAP
    COMPONENT Mux_2to1_5b
        PORT (
            CONTROL : IN STD_LOGIC; -
- Controller to select the desired data
            A : IN STD_LOGIC_VECTOR (0 TO 4); -- The first data option
            B : IN STD_LOGIC_VECTOR (0 TO 4); -- The second data option
            X : OUT STD_LOGIC_VECTOR (0 TO 4)
        );

    END COMPONENT;

    -- Reg_Pipe_IDEX - PORT MAP
    COMPONENT Reg_Pipe_IDEX
        PORT (
            CLOCK : IN STD_LOGIC;
            IDEX_IN_WB : IN STD_LOGIC_VECTOR(0 TO 1);
            IDEX_IN_MEM : IN STD_LOGIC_VECTOR(0 TO 2);
            IDEX_IN_EX : IN STD_LOGIC_VECTOR(0 TO 4);
            IDEX_IN_PC : IN STD_LOGIC_VECTOR(0 TO 31);
            IDEX_IN_READ1 : IN STD_LOGIC_VECTOR(0 TO 31);
            IDEX_IN_READ2 : IN STD_LOGIC_VECTOR(0 TO 31);
            IDEX_IN_IMED : IN STD_LOGIC_VECTOR(0 TO 31);
            IDEX_IN_RT : IN STD_LOGIC_VECTOR(0 TO 4);
            IDEX_IN_RD : IN STD_LOGIC_VECTOR(0 TO 4);

            IDEX_OUT_WB : OUT STD_LOGIC_VECTOR(0 TO 1);
            IDEX_OUT_MEM : OUT STD_LOGIC_VECTOR(0 TO 2);
            IDEX_OUT_EX : OUT STD_LOGIC_VECTOR(0 TO 4);
            IDEX_OUT_PC : OUT STD_LOGIC_VECTOR(0 TO 31);
            IDEX_OUT_READ1 : OUT STD_LOGIC_VECTOR(0 TO 31);
            IDEX_OUT_READ2 : OUT STD_LOGIC_VECTOR(0 TO 31);
            IDEX_OUT_IMED : OUT STD_LOGIC_VECTOR(0 TO 31);
            IDEX_OUT_RT : OUT STD_LOGIC_VECTOR(0 TO 4);
            IDEX_OUT_RD : OUT STD_LOGIC_VECTOR(0 TO 4)
        );
    END COMPONENT;

    -- Reg_Pipe_EXMEM - PORT MAP
    COMPONENT Reg_Pipe_EXMEM
        PORT (
            CLOCK : IN STD_LOGIC;
            EXMEM_IN_ZERO : IN STD_LOGIC;
```

```vhdl
            EXMEM_IN_WB : IN STD_LOGIC_VECTOR(0 TO 1);
            EXMEM_IN_MEM : IN STD_LOGIC_VECTOR(0 TO 2);
            EXMEM_IN_RESULT_ADDER : IN STD_LOGIC_VECTOR(0 TO 31);
            EXMEM_IN_RESULT_ULA : IN STD_LOGIC_VECTOR(0 TO 31);
            EXMEM_IN_READ2 : IN STD_LOGIC_VECTOR(0 TO 31);
            EXMEM_IN_REGDST : IN STD_LOGIC_VECTOR(0 TO 4);

            EXMEM_OUT_ZERO : OUT STD_LOGIC;
            EXMEM_OUT_WB : OUT STD_LOGIC_VECTOR(0 TO 1);
            EXMEM_OUT_MEM : OUT STD_LOGIC_VECTOR(0 TO 2);
            EXMEM_OUT_RESULT_ADDER : OUT STD_LOGIC_VECTOR(0 TO 31);
            EXMEM_OUT_RESULT_ULA : OUT STD_LOGIC_VECTOR(0 TO 31);
            EXMEM_OUT_READ2 : OUT STD_LOGIC_VECTOR(0 TO 31);
            EXMEM_OUT_REGDST : OUT STD_LOGIC_VECTOR(0 TO 4)
        );
    END COMPONENT;

    -- Reg_Pipe_MEMWB - PORT MAP
    COMPONENT Reg_Pipe_MEMWB
        PORT (
            CLOCK : IN STD_LOGIC;
            MEMWB_IN_WB : IN STD_LOGIC_VECTOR(0 TO 1);
            MEMWB_IN_RESULT_ULA : IN STD_LOGIC_VECTOR(0 TO 31);
            MEMWB_IN_REGDST : IN STD_LOGIC_VECTOR(0 TO 4);
            MEMWB_IN_READ_DATA : IN STD_LOGIC_VECTOR(0 TO 31);

            MEMWB_OUT_WB : OUT STD_LOGIC_VECTOR(0 TO 1);
            MEMWB_OUT_RESULT_ULA : OUT STD_LOGIC_VECTOR(0 TO 31);
            MEMWB_OUT_REGDST : OUT STD_LOGIC_VECTOR(0 TO 4);
            MEMWB_OUT_READ_DATA : OUT STD_LOGIC_VECTOR(0 TO 31)
        );
    END COMPONENT;

    -- FileRegister - PORT MAP
    COMPONENT FileRegister
        PORT (
            REGWRITE : IN STD_LOGIC;
            CLOCK : IN STD_LOGIC;
            READ_REGISTER_1 : IN STD_LOGIC_VECTOR(0 TO 4);
            READ_REGISTER_2 : IN STD_LOGIC_VECTOR(0 TO 4);
            WRITE_REGISTER : IN STD_LOGIC_VECTOR(0 TO 4);
            WRITE_DATA : IN STD_LOGIC_VECTOR(0 TO 31);

            READ_DATA_1 : OUT STD_LOGIC_VECTOR(0 TO 31);
            READ_DATA_2 : OUT STD_LOGIC_VECTOR(0 TO 31);

            DEB_FILE_REG_1 : OUT STD_LOGIC_VECTOR(0 TO 31);

            DEB_FILE_REG_2 : OUT STD_LOGIC_VECTOR(0 TO 31);
            DEB_FILE_REG_3 : OUT STD_LOGIC_VECTOR(0 TO 31);
            DEB_FILE_REG_AUX : OUT STD_LOGIC
        );
    END COMPONENT;

    -- FileRegister - PORT MAP
    COMPONENT ProgramCounter
        PORT (
            CLOCK : IN STD_LOGIC;
            PC_INC : IN STD_LOGIC_VECTOR(0 TO 31);
            PC : OUT STD_LOGIC_VECTOR(0 TO 31)
        );
    END COMPONENT;

    -- SignExtend - PORT MAP
    COMPONENT SignExtend
        PORT (
            A : IN STD_LOGIC_VECTOR (0 TO 15);
            X : OUT STD_LOGIC_VECTOR(0 TO 31)
        );
    END COMPONENT;

    --------------------------------------------------------------------
----------------------------------------
    --------------------------------------------------- SIGNALS ---------
----------------------------------------

    -- Used to do nothing normally
    SIGNAL UNUSED : STD_LOGIC;

    --******************** INSTRUCTION FETCH ********************--

    -- SIGNALS - TO CONTROL COMPONENT INTERACTION IN EXECUTION TIME
    SIGNAL SIG_PC_SRC : STD_LOGIC;
    SIGNAL SIG_JUMP : STD_LOGIC;

    -- WIRES - TO CONNECT PORT MAP BETWEEN COMPONENTS
    SIGNAL WIRE_OUT_PC_INC : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_OUT_PC : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_MUX_PC : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_MUX_JUMP : STD_LOGIC_VECTOR(0 TO 31); --
Used to connect mux
    SIGNAL WIRE_INST_MEM_IFID : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_MUX_PC_PC : STD_LOGIC_VECTOR(0 TO 31);

    SIGNAL WIRE_SHIFT_MUX_JUMP : STD_LOGIC_VECTOR(0 TO 27);
    SIGNAL WIRE_SHIFT_CONCAT : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_MUX_PC_MUX_JUMP : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_MUX_JUMP_PC : STD_LOGIC_VECTOR(0 TO 31);
    ---------------------------------------------------------------------
-----------------------------------------
    --------------------------------------------------- DECODE STAGE ----
----------------------------------------------

    --WIRES
    SIGNAL WIRE_PC_INC_IFID_TO_IDEX : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_OUT_IFID_INSTRUCTION : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_READ_DATA1_IDEX : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_READ_DATA2_IDEX : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_SIGNAL_EXTEND_IDEX : STD_LOGIC_VECTOR(0 TO 31);
    -- Control Unit
    SIGNAL WIRE_UC_IDEX_WB : STD_LOGIC_VECTOR(0 TO 1);
    SIGNAL WIRE_UC_IDEX_MEM : STD_LOGIC_VECTOR(0 TO 2);
    SIGNAL WIRE_UC_IDEX_EX : STD_LOGIC_VECTOR(0 TO 4);
    SIGNAL WIRE_UC_JUMP_SIGNAL : STD_LOGIC_VECTOR(0 TO 1) := "00";

    --
******************** INSTRUCTION EXECUTION STAGE  ********************--

    -- WIRES OF REG_PIPE
    SIGNAL WIRE_IDEX_WB_EXMEM_WB : STD_LOGIC_VECTOR(0 TO 1);
    SIGNAL WIRE_IDEX_MEM_EXMEM_MEM : STD_LOGIC_VECTOR(0 TO 2);
    SIGNAL WIRE_PC_INC_IDEX_TO_ADDER : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_IDEX_READ1_ALU_A : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_OUT_IDEX_READ2 : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_OUT_IDEX_IMED : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_IDEX_RT_MUX_REGDST : STD_LOGIC_VECTOR(0 TO 4);
    SIGNAL WIRE_IDEX_RD_MUX_REGDST : STD_LOGIC_VECTOR(0 TO 4);
    SIGNAL WIRE_OUT_IDEX_EX : STD_LOGIC_VECTOR(0 TO 4);
    -- WIRES OF THIS STAGE
    SIGNAL WIRE_SHIFT_LEFT_ADDER_B : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_MUX_ALU_B : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_MUX_REGDST_EXMEM : STD_LOGIC_VECTOR(0 TO 4);
    SIGNAL WIRE_ULA_CODE : STD_LOGIC_VECTOR(0 TO 1);
    SIGNAL WIRE_ALU_RES_EXMEM : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_ZERO_EXMEM : STD_LOGIC;
    SIGNAL WIRE_ADDER_RES_EXMEM : STD_LOGIC_VECTOR(0 TO 31);

    --******************** INSTRUCTION MEMORY STAGE ********************-
-

    -- WIRES OF REG_PIPE
    SIGNAL WIRE_OUT_EXMEM_ZERO : STD_LOGIC;
    SIGNAL WIRE_EXMEM_WB_MEMWB_WB : STD_LOGIC_VECTOR(0 TO 1);
    SIGNAL WIRE_OUT_EXMEM_MEM : STD_LOGIC_VECTOR(0 TO 2);
    SIGNAL WIRE_EXMEM_ADDER_RES_MUXPC : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_OUT_EXMEM_ALU_RES : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_EXMEM_READ2_WRITE_DATA : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_EXMEM_REGDST_MEM_WB : STD_LOGIC_VECTOR(0 TO 4);

    -- WIRES OF THIS STAGE

    SIGNAL PCSrc : STD_LOGIC := '0';
    SIGNAL WIRE_READ_DATA_MEMWB : STD_LOGIC_VECTOR(0 TO 31);

    --
******************** INSTRUCTION WRITE BACK STAGE ********************--

    -- WIRES OF REG_PIPE
    SIGNAL WIRE_OUT_MEMWB_WB : STD_LOGIC_VECTOR(0 TO 1);
    SIGNAL WIRE_MEMWB_ALU_RES : STD_LOGIC_VECTOR(0 TO 31);
    SIGNAL WIRE_MEMWB_REG_DST : STD_LOGIC_VECTOR(0 TO 4);
    SIGNAL WIRE_MEMWB_READ_DATA_MUX_WB : STD_LOGIC_VECTOR(0 TO 31);
    --
    SIGNAL WIRE_MUX_WB_WRITE_DATA : STD_LOGIC_VECTOR(0 TO 31);

BEGIN

    --
******************** COMPONENTS INSTRUCTION FETCH ********************--

    INST_MEM : InstructionMemory PORT MAP(WIRE_OUT_PC, WIRE_INST_MEM_IFID
);
    MUX_PC : Mux_2to1_32b PORT MAP(PCSrc, WIRE_OUT_PC_INC, WIRE_EXMEM_ADD
ER_RES_MUXPC, WIRE_MUX_PC_MUX_JUMP);
    PC_INC : PCIncrement PORT MAP(WIRE_OUT_PC, WIRE_OUT_PC_INC);
    MUX_JUMP : Generic3to1Mux PORT MAP(WIRE_UC_JUMP_SIGNAL, WIRE_MUX_PC_M
UX_JUMP, WIRE_SHIFT_CONCAT, WIRE_READ_DATA1_IDEX, WIRE_MUX_JUMP_PC);
    PC : ProgramCounter PORT MAP(CLOCK, WIRE_MUX_JUMP_PC, WIRE_OUT_PC);

    DEB_REGS_PC <= WIRE_OUT_PC_INC;
    --******************** REG_PIPELINE IF/ID ********************--

    IFID : Reg_Pipe_IFID PORT MAP(CLOCK, WIRE_OUT_PC_INC, WIRE_INST_MEM_I
FID, WIRE_PC_INC_IFID_TO_IDEX, WIRE_OUT_IFID_INSTRUCTION);

    INSTRUCTION_OUT_IFID <= WIRE_OUT_IFID_INSTRUCTION;

    --
******************** COMPONENTS INSTRUCTION DECODE STAGE ****************
****--

    WIRE_SHIFT_CONCAT <= WIRE_PC_INC_IFID_TO_IDEX(0 TO 3) & WIRE_SHIFT_MU
X_JUMP;

    SHIFT_IF : ShiftLeft2_26to28 PORT MAP(WIRE_OUT_IFID_INSTRUCTION(6 TO
31), WIRE_SHIFT_MUX_JUMP);
```

```
    DEB_REG_ULA_IN_1 <= WIRE_OUT_IFID_INSTRUCTION(6 TO 10);
    DEB_SINAL_REG_WRITE <= WIRE_OUT_MEMWB_WB(0);
    FILE_REG : FileRegister PORT MAP(
        WIRE_OUT_MEMWB_WB(0),
        CLOCK,
        WIRE_OUT_IFID_INSTRUCTION(6 TO 10),
        WIRE_OUT_IFID_INSTRUCTION(11 TO 15),
        WIRE_MEMWB_REG_DST,
        WIRE_MUX_WB_WRITE_DATA,
        WIRE_READ_DATA1_IDEX,
        WIRE_READ_DATA2_IDEX,

        DEB_FILE_REG_1,
        DEB_FILE_REG_2,
        DEB_FILE_REG_3,

        DEB_FILE_REG_AUX
    );
    SIGNAL_EXTEND : SignExtend PORT MAP(WIRE_OUT_IFID_INSTRUCTION(16 TO 3
1), WIRE_SIGNAL_EXTEND_IDEX);
    UC : ControlUnit PORT MAP(WIRE_UC_IDEX_WB, WIRE_UC_IDEX_MEM, WIRE_UC_
IDEX_EX, WIRE_UC_JUMP_SIGNAL, WIRE_OUT_IFID_INSTRUCTION);

    --******************** REG_PIPELINE ID/EX ********************--
    IDEX : Reg_Pipe_IDEX PORT MAP(
        -- IN --
        CLOCK, WIRE_UC_IDEX_WB, WIRE_UC_IDEX_MEM, WIRE_UC_IDEX_EX, WIRE_P
C_INC_IFID_TO_IDEX, WIRE_READ_DATA1_IDEX, WIRE_READ_DATA2_IDEX, WIRE_SIGN
AL_EXTEND_IDEX, WIRE_OUT_IFID_INSTRUCTION(11 TO 15), WIRE_OUT_IFID_INSTRU
CTION(16 TO 20),
        -- OUT --
        WIRE_IDEX_WB_EXMEM_WB, WIRE_IDEX_MEM_EXMEM_MEM, WIRE_OUT_IDEX_EX,
 WIRE_PC_INC_IDEX_TO_ADDER, WIRE_IDEX_READ1_ALU_A, WIRE_OUT_IDEX_READ2, W
IRE_OUT_IDEX_IMED, WIRE_IDEX_RT_MUX_REGDST, WIRE_IDEX_RD_MUX_REGDST
    );
    --
******************** COMPONENTS INSTRUCTION EXECUTION  ******************
**--

    SHIFT_LEFT : ShiftLeft PORT MAP(WIRE_OUT_IDEX_IMED, WIRE_SHIFT_LEFT_A
DDER_B);
    MUX_ALU_B : Mux_2to1_32b PORT MAP(WIRE_OUT_IDEX_EX(4), WIRE_OUT_IDEX_
READ2, WIRE_OUT_IDEX_IMED, WIRE_MUX_ALU_B);
    MUX_REGDST : Mux_2to1_5b PORT MAP(WIRE_OUT_IDEX_EX(0), WIRE_IDEX_RT_M
UX_REGDST, WIRE_IDEX_RD_MUX_REGDST, WIRE_MUX_REGDST_EXMEM);
    ALU_CONTROL : AluControl PORT MAP(WIRE_OUT_IDEX_EX(1 TO 3), WIRE_OUT_
IDEX_IMED(26 TO 31), WIRE_ULA_CODE);
```

```
    ADDER_SHIFT2_PC_INC : Alu PORT MAP(WIRE_PC_INC_IDEX_TO_ADDER, WIRE_SH
IFT_LEFT_ADDER_B, "00", WIRE_ADDER_RES_EXMEM, UNUSED);
    MAIN_ALU : Alu PORT MAP(WIRE_IDEX_READ1_ALU_A, WIRE_MUX_ALU_B, WIRE_U
LA_CODE, WIRE_ALU_RES_EXMEM, WIRE_ZERO_EXMEM);

    DEB_CONTROL <= WIRE_OUT_IDEX_EX(0);
    DEB_ULA_IN_1 <= WIRE_IDEX_READ1_ALU_A;
    DEB_ULA_IN_2 <= WIRE_MUX_ALU_B;
    DEB_OUT_ULA <= WIRE_ALU_RES_EXMEM;
    DEB_RegDst <= WIRE_MUX_REGDST_EXMEM;
    --******************** REG_PIPELINE EX/MEM ********************--
    EXMEM : Reg_Pipe_EXMEM PORT MAP(
        -- IN --
        CLOCK,
        WIRE_ZERO_EXMEM,
        WIRE_IDEX_WB_EXMEM_WB,
        WIRE_IDEX_MEM_EXMEM_MEM,
        WIRE_ADDER_RES_EXMEM,
        WIRE_ALU_RES_EXMEM,
        WIRE_OUT_IDEX_READ2,
        WIRE_MUX_REGDST_EXMEM,
        -- OUT --
        WIRE_OUT_EXMEM_ZERO,
        WIRE_EXMEM_WB_MEMWB_WB,
        WIRE_OUT_EXMEM_MEM,
        WIRE_EXMEM_ADDER_RES_MUXPC,
        WIRE_OUT_EXMEM_ALU_RES,
        WIRE_EXMEM_READ2_WRITE_DATA,
        WIRE_EXMEM_REGDST_MEM_WB
    );

    --
******************** COMPONENTS INSTRUCTION MEMORY  ********************-
-
    PCSrc <= WIRE_OUT_EXMEM_MEM(0) AND WIRE_OUT_EXMEM_ZERO;
    DATA_MEMORY : DataMemory PORT MAP(WIRE_OUT_EXMEM_ALU_RES, CLOCK, WIRE
_OUT_EXMEM_MEM(2), WIRE_EXMEM_READ2_WRITE_DATA, WIRE_OUT_EXMEM_MEM(1), WI
RE_READ_DATA_MEMWB);

    --******************** REG_PIPELINE MEM/WB ********************--

    MEMWB : Reg_Pipe_MEMWB PORT MAP(
        -- IN --
        CLOCK,
        WIRE_EXMEM_WB_MEMWB_WB,
        WIRE_OUT_EXMEM_ALU_RES,
        WIRE_EXMEM_REGDST_MEM_WB,
        WIRE_READ_DATA_MEMWB,
        -- OUT --
        WIRE_OUT_MEMWB_WB,
        WIRE_MEMWB_ALU_RES,
        WIRE_MEMWB_REG_DST,
        WIRE_MEMWB_READ_DATA_MUX_WB
    );

    --
******************** COMPONENTS INSTRUCTION WRITE BACK ******************
**--
    MUX_WB : Mux_2to1_32b PORT MAP(WIRE_OUT_MEMWB_WB(1), WIRE_MEMWB_READ_
DATA_MUX_WB, WIRE_MEMWB_ALU_RES, WIRE_MUX_WB_WRITE_DATA);
    DEB_SINAL_MUX_MEMWB <= WIRE_OUT_MEMWB_WB(1);

    DEB_WRITE_REG <= WIRE_MEMWB_REG_DST;
    DEB_WRITE_DATA <= WIRE_MUX_WB_WRITE_DATA;

    ---------------------------------------------------------------------
------------------------------------------
    ---------------------------------------------------- DEBUG ----------
------------------------------------------
END;
```