<u>EEE 333 Lab2 – Assignment – Fall 2015</u>

Name (First, Last): *Tiago, Guido Piai*

ASU ID: *1209830918*

Lab Teammates names (First,Last) : *Igor, Viana de Pinho Tavares*

Victor, de Almeida Piccoli Ferreira

Abstract

In this lab, the group created three components (a Multiplexor, a Registrator and an ALU) used together to create a simple process that executes sixteen different operations.

Four test bench were created representing each one of the components and the complete logic, but only the main test bench was demonstrated to the T.A. All the test bench were added to the Lab Report.

Differently of the previous Lab, the main logic was not updated to the FPGA.

1. Introduction

VHDL is programming language that supports structural and behavioral modeling. Hence, it is possible to program in VHDL in different ways, even combining both models to simplify the function.

The behavioral modeling describes a specified function in an abstract way , which execute actions similar to high-level convectional languages. The types of actions that can be performed include evaluating expressions, assigning values to variables, conditional execution, repeated execution and subprogram calls.

The structural modeling is an alternative way to designate the implementation of an entity using only subcomponents. Sometimes it is necessary to call a component with a specified behavior in different situations. Therefore, it is easier just call a same component many times with the requested behavior than repeat lines of codes many times to execute the same function.

Combinational Logic is a sort of circuit where the outputs are only defined by the actual values of inputs. The circuit does not keep internal states or values produced in previous process.

The opposite of the Combinational Logic is the Sequential Logic, in which the circuit maintain an internal state. The produced outputs are a result of the actual inputs and the previous outputs generated in the past states. Sequential circuits can be design as synchronous circuits, which uses a rising or falling edge of a clock, or a level of an enable signal, to control advance of state or storage of data. An asynchronous circuit is a kind of circuit without an enable or clock.

According the logic created for the Lab 2, the output values produced in a present state have direct relation with values produced in previous states. Therefore, the circuit confirms a sequential behavior.

2. *Design explanation*

A microprocessor is a very important tool in the electrical world, it is capable of execution multiple operation. A basic sample processor could be designed in VHDL without much difficult. For this laboratory was required to develop and test a processor using structural approach, based on the following schematic:

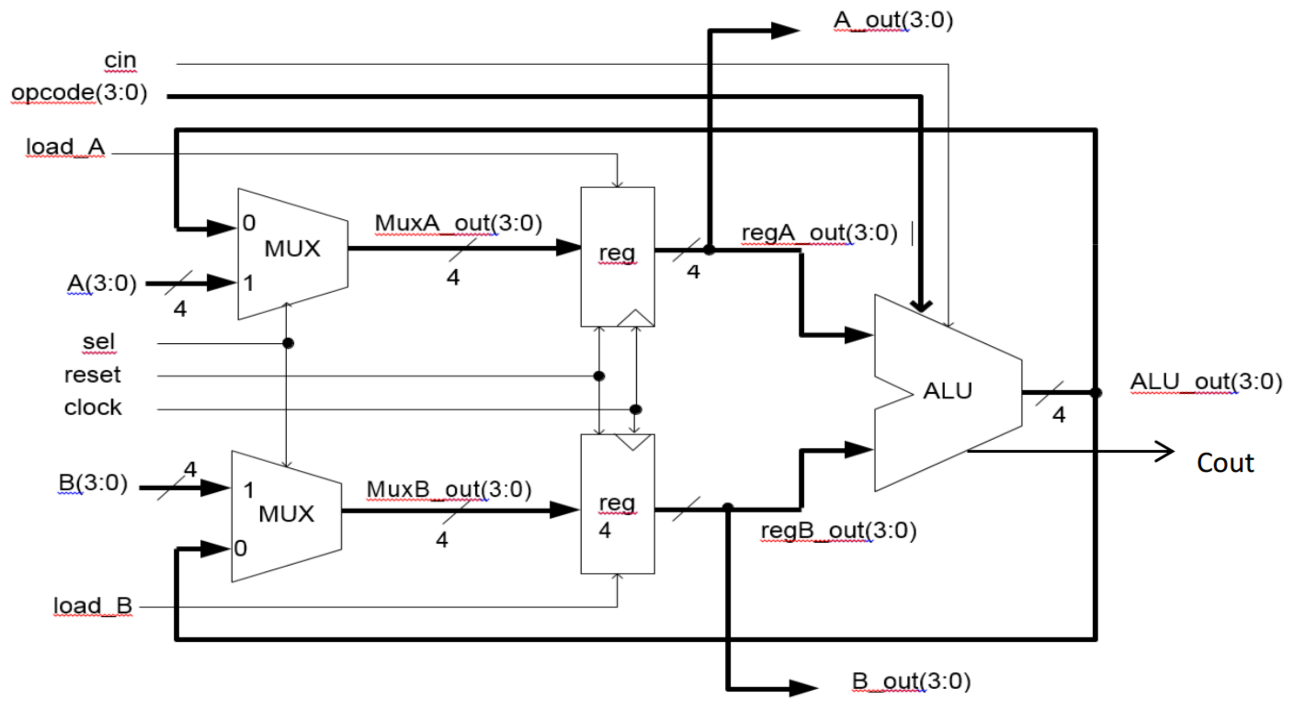


Figure 1: Sample Processor Schematic

For this processor were used three types of components: Multiplexer (2), Register (2), Arithmetic Logic Unit (ALU).

2.1 Multiplexer

The multiplexer has the function of selecting a signal. For this particular case the multiplexers have two inputs with four bits each and the output will be only one of this two inputs, depending on the value of the selector bit. If the selector bit is ‘0’ the first input will be on the output, if the selector bit is ‘1’ the second input will be on the output.

2.2 Register

The main function of register is to temporarily store a value that will be used for some operation. A register is usually designed using flip-flops D in parallel. The number of flip-flops that will be used depends on the number of bits that are being stored. For this case there are four bits, so four flip-flops D are used.

To store a value in the register is needed to enable the flip-flops, so the input becomes the output. The operation to this is called load. Another important thing that is important to be in a register is the reset, once it leaves an option to the system reset if it fails and the values in the registers will be known.

2.3 Arithmetic Logic Unit

The ALU works based in a control input, which indicates the type of operation to be executed. The ALU use the inputs and make the selected operation with then, presenting the result in the output. For this case the ALU has a control input with four bits, allowing sixteen different operations. The operations are listed in the following table:

Table 1 – ALU Operations

Opcode	Operation
0000	NOT A
0001	NOT B
0010	A AND B
0011	A OR B
0100	A NAND B
0101	A NOR B
0110	A XOR B
0111	A XNOR B
1000	Transfer A
1001	Increment A
1010	Decrement A
1011	Transfer B
1100	Increment B
1101	Decrement B
1110	A + B
1111	A - B

In the table are described all the operation that will be implemented. The first eight operations are logic and the last eight are arithmetic.

2.4 Sample Processor

Following the schematic in the Figure 1, is possible to see that it has two data inputs A and B, which are four bits, the load_A and load_B inputs that enable the respective register to store the values. The selector is responsible to select which signal will put into the register, the result in the ALU or the data input. The opcode input choose the operation that the ALU will execute. The output for this processor are the outputs of the two registers and the Cout that can result of the ALU operations.

3. *Implementation*

The implementation of this processor consists in the description of each component and then a main file to join the designs.

3.1 Multiplexer

For the multiplexer were implemented two inputs with four bits each, A and B, and a one bit input ‘sel’ for selection of the signal that will be on the four bits output ‘O’. For the code was used a WHEN/ELSE approach. So when the selector is ‘0’ the output value will be A and when the selector is ‘1’ the output value will be B.

Since the input ‘sel’ is an STD_LOGIC data type is important to describe a situation if ‘sel’ receive values that are not ‘1’ or ‘0’, that is why in this design, after the second WHEN/ELSE statement was written “0000”, so the output of the mux will be “0000” when this situation happens.

3.2 Register

The register was implemented considering a four bits data input, which will be passed to the output when the enable input bit is ‘1’ and happens a rising edge in the clock. There is also a reset input to put “0000” in the register output.

A process was described with four parameters in the sensitivity list: reset (rst), clock (clk), enable (en) and data input (Data_in). An if was inserted to check if the rst is one if it happens to be the output will be “0000”, else the code will check for a clock alteration and if it was also a rising edge. If the rising edge happens the register will check if the enable is ‘1’, if it is the output will be updated with the input value, otherwise the output will remain the same that was previous.

3.3 Arithmetic Logic Unit

This ALU consists in the sixteen operations shown in the Table 1. A and B are the data inputs with four bits each. The opcode tells which operation the ALU will do. The output will be the result of the operation in a four bits STD_LOGIC_VECTOR plus a one bit STD_LOGIC to represent the carry out when possible.

The code implementation of the ALU consisted in checking the opcode value and redirecting to the proper operation. The programming resource used was the CASE. So when the opcode has determined value the correct operation is executed using the data inputs A and B.

The eight first operation are logic. The “0000” and “0001” operations make the one’s complement of A and B inputs, respectively. The “0010” and “0011” makes an AND and OR operation between the A and B inputs, respectively. And for the other operations “0100”, “0101”, “0110” and “0111” are NAND, NOR, XOR and XNOR between A and B, respectively.

All this logic operations are send directly to the ALU four bits output and Cout will always be ‘0’;

The last eight operations are arithmetic. The “1000” and “1011” operations transfer the A and B to the output, respectively, being the Cout ‘0’. The “1001” and “1100” operations increments the A and B, respectively, in this case Cout is used and the method applied will be described below. The “1010” and the “1101” operations decrements A and B, respectively, no carry out is applied for this operations. The operation “1110” add unsigned A and unsigned B, in this case the carry out is applied. The last operation “1111” execute a subtraction between signed A and signed B, and the carry out is not used.

To calculate the carry out a five bits STD_LOGIC_VECTOR was declared to store the result of the operations that use the carry out. In this operation the most significant bit is the Cout output and the other four least significant are the ALU output (ALU_out).

3.4 Sample Processor

For the processor all the components previously described were put together and connect with the proper signals. The processor contains two multiplexer, two registers and one ALU.

In the first multiplexer was inserted in the ALU output and the A data input. In the second the same ALU output was inserted and the B data input. The selectors of the multiplexers were connected into the same ‘sel’ input.

The data inputs of the registers are the outputs of the multiplexers. To load the data input in a register a load input was necessary to be declared, so for each of the register there is a load input connected to the enable input of the register component. Another input necessary to load the register is the clock, once a rising edge happens the output receives the data input if the load is active. The last input of the register is the reset, responsible for putting “0000” in the register output. The resets are connected together.

In the ALU the two data inputs are the data that come from the registers outputs, and the opcode is an input of the processor entity. Given the proper inputs the result obtained is send to the four bit ALU output that is connected with the two multiplexers as said before. The other output is the carry out (Cout) which is set when carry out happens.

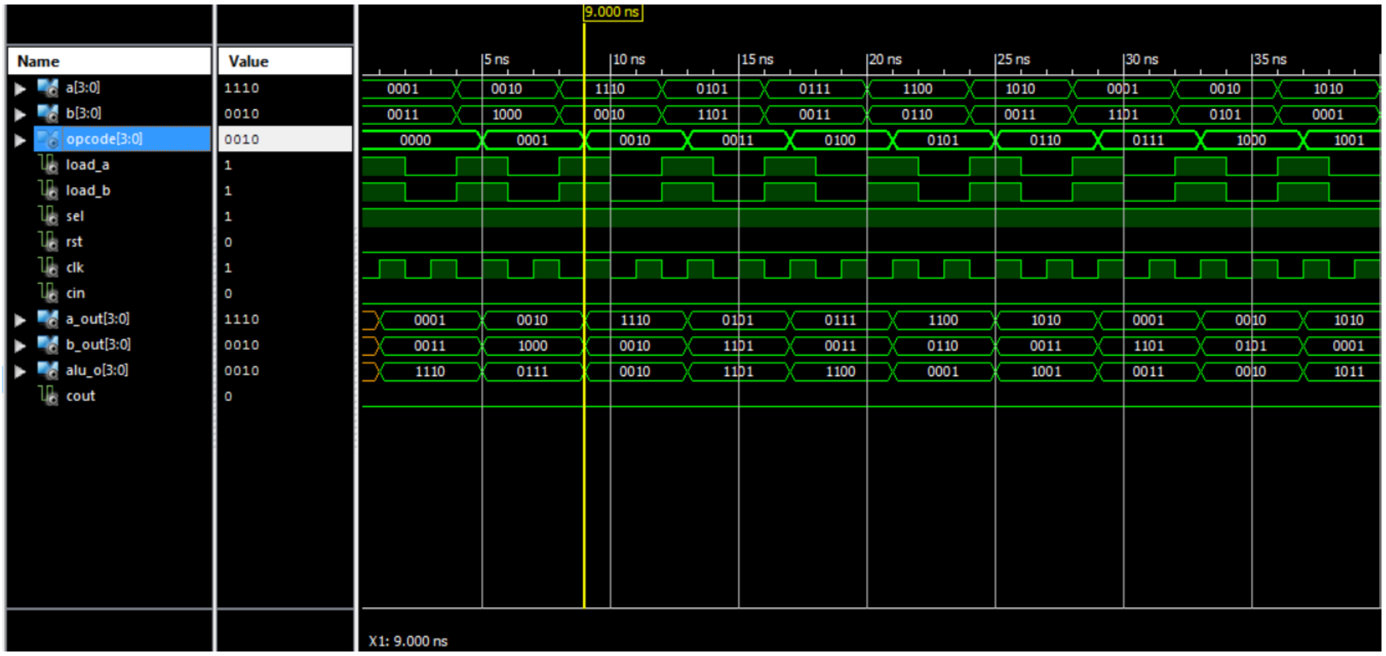
The data outputs of the processor are the outputs of the registers, being these A_out and B_out. The third output is the Cout. Five internal signals were necessary to the implementation: two for the multiplexers data outputs (one for each), two for the registers data outputs (one for each) and one for the ALU output.

To make an easier test bench an ALU output were inserted so the values will be more visible and less clock cycles were used.

4. Results

A test bench is a simulation used to confirm the behavior of a design developed. In this case four test benches were generated. Three test benches were used to simulate each component of the processor and one test bench was used to simulate the processor.

4.1 The processor



Name	Value
a[3:0]	1110
b[3:0]	0010
opcode[3:0]	0010
load_a	1
load_b	1
sel	1
rst	0
clk	1
cin	0
a_out[3:0]	1110
b_out[3:0]	0010
alu_o[3:0]	0010
cout	0

Figure 2: Sample Processor Waveform Simulation – part 1

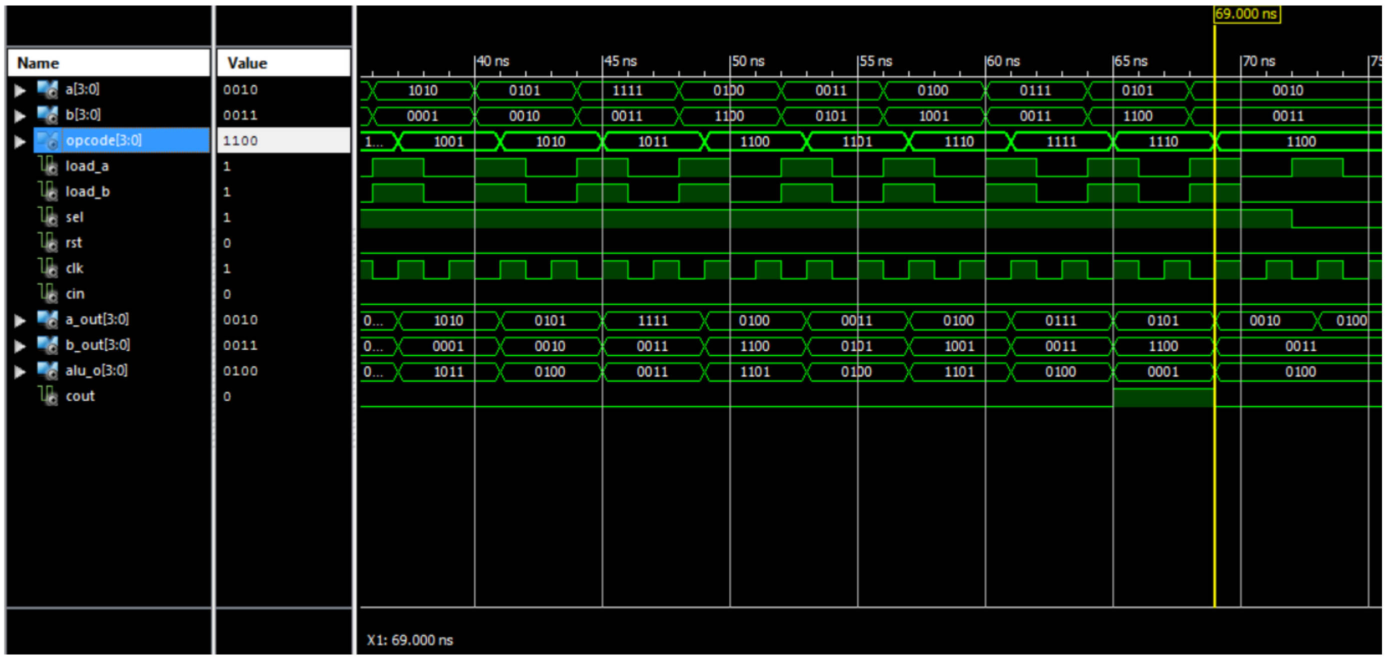


Figure 3: Sample Processor Waveform Simulation – part 2

Table 2: Sample Processor Simulated Data

A	B	Operation	ALU_out	Cout	opcode
0001	0010	y<= not a	1110	0	0000
0010	1000	y<= not b	0111	0	0001
1110	0010	y<= a and b	0010	0	0010
0101	1101	y<= a or b	1101	0	0011
0111	0011	y<= a nand b	1100	0	0100
1100	0110	y<= a nor b	0001	0	0101
1010	0011	y<= a xor b	1001	0	0110
0001	1101	y<= a xnor b	0011	0	0111
0010	0101	y<= a	0010	0	1000
1010	0001	y<= a + 1	1011	0	1001
0101	0010	y<= a - 1	0100	0	1010
1111	0011	y<= b	0011	0	1011
0100	1100	y<= b + 1	1101	0	1100
0011	0101	y<= b - 1	0100	0	1101
0100	1001	y<= a + b	1101	0	1110
0111	0011	y<= a - b	0100	0	1111
0111	0011	y<= a + b	0001	1	1110

The processor is regulated by a clock that is set to 1 ns and the operations are set to change every 4 ns at the test bench. The type of operation is regulated by the opcode and the ALU_out receives the results of the operations. The components used were:

4.2 The 2 to 1 multiplexer

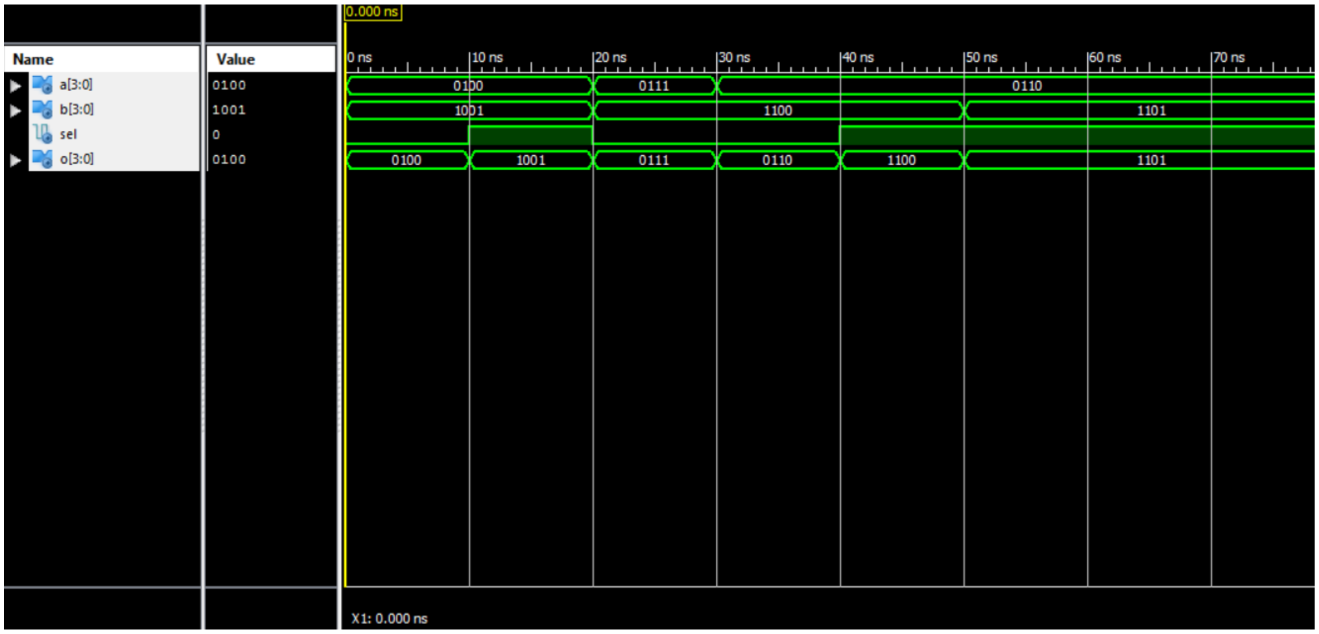


Figure 4: Multiplexer Waveform Simulation

The multiplexer selects between ALU_out or A, or ALU_out or B. It is controlled by the sel input.

4.3 The 4 bit register

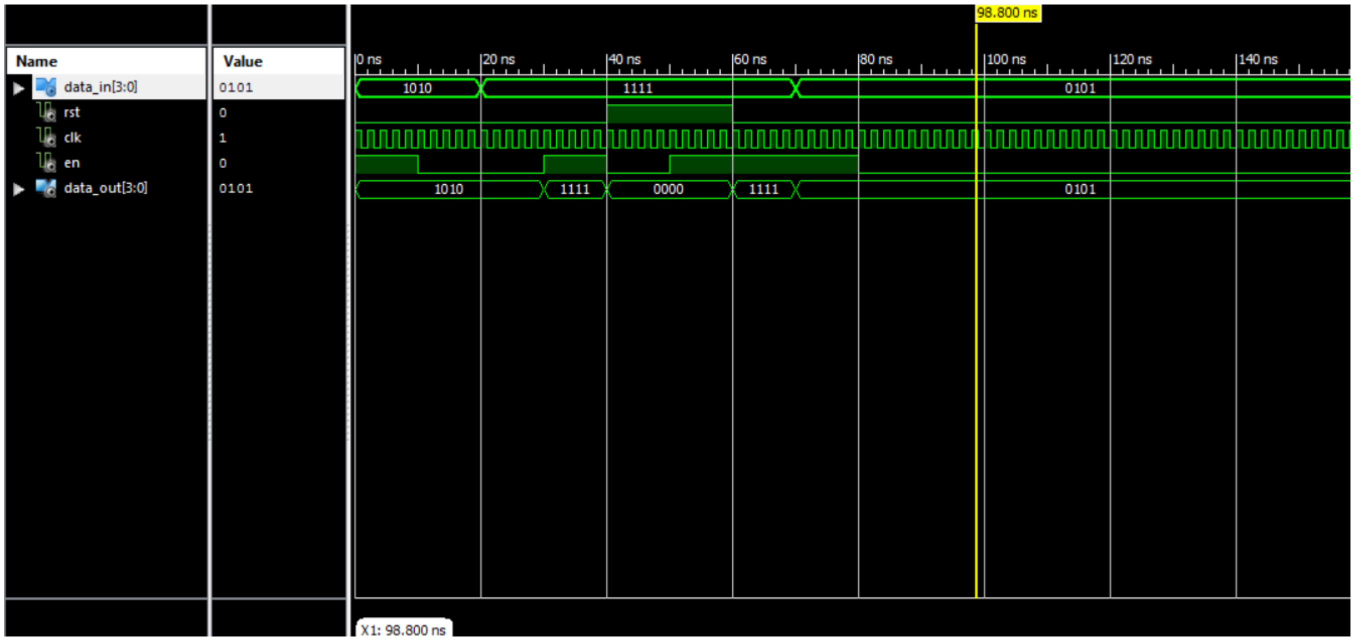


Figure 5: Register Waveform Simulation

The register holds a value of the input on the output and changes it based on the clock. It either holds the value of A or ALU_out, or B or ALU_out. The rst input sets the output to zero and the enable input either holds the value of the output or lets it change based on the clock.

4.4 The ALU

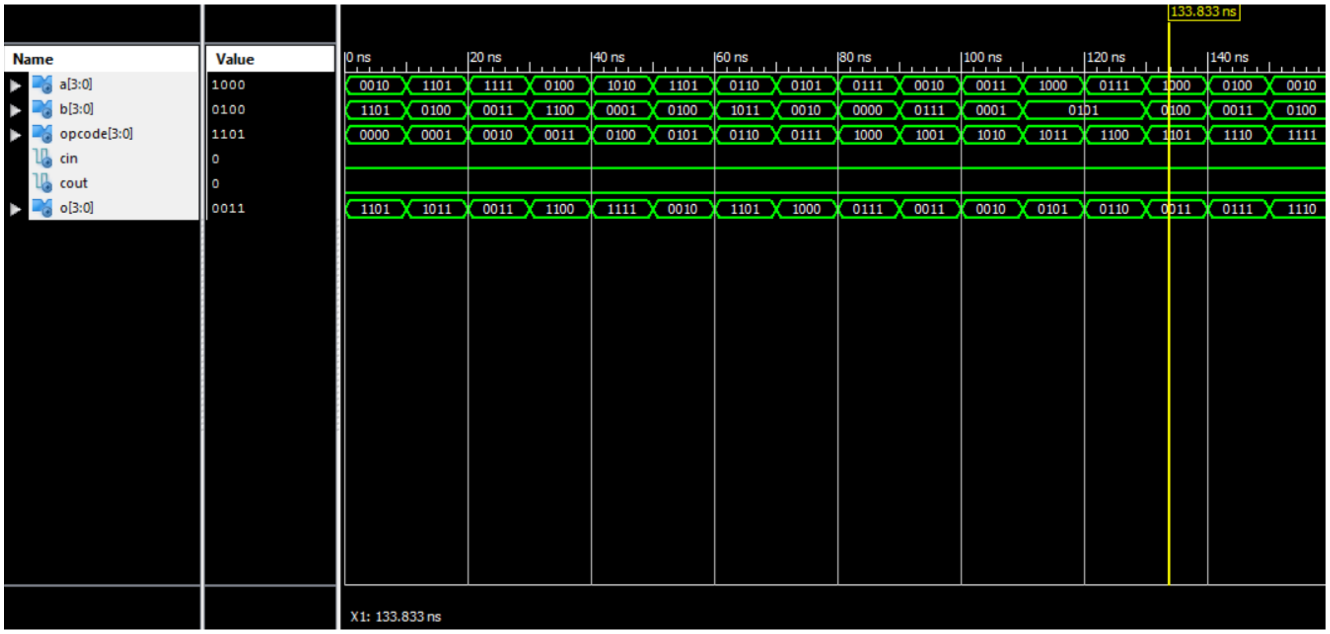


Figure 6: ALU Waveform Simulation

Table 3: ALU Simulated Data

A	B	Operation	ALU_out	Cout	opcode
0010	1101	y<= not a		0	0000
1101	0100	y<= not b		0	0001
1111	0011	y<= a and b		0	0010
0100	1100	y<= a or b		0	0011
1010	0001	y<= a nand b		0	0100
1101	0100	y<= a nor b		0	0101
0110	1011	y<= a xor b		0	0110
0101	0010	y<= a xnor b		0	0111
0111	0000	y<= a		0	1000
0010	0111	y<= a + 1		0	1001
0011	0001	y<= a - 1		0	1010
1000	0101	y<= b		0	1011
0111	0101	y<= b + 1		0	1100
1000	0100	y<= b - 1		0	1101
0100	0011	y<= a + b		0	1110
0010	0100	y<= a - b		0	1111

The ALU does the operations and it is controlled by the opcode. The carry out is only used in addition and subtraction operations. The output of the operations is the ALU_out signal. The cin is not used in this lab.

5. Conclusion

In this lab, we set out to implement a VHDL of a simple process with sixteen main operations. We proved that the VHDL has the possibility to keep output values in internal statements and use them in future process. .

The most difficult part of the design was designing a correct test bench that changed the values in the correct pulse of clock and executed them in an expected way. Since is the first time that we worked with pulses of clock, it is easy to understand way controlling correctly the pulses was the hardest part.

Appendix

CODES

```vhdl
-- File: mux2to1.vhd
-- Description: Multiplexer four bits two input to one output.

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux2to1 IS
PORT (A, B: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        sel: IN STD_LOGIC;
        O: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END mux2to1;

ARCHITECTURE func OF mux2to1 IS
BEGIN
O <=    A WHEN sel = '0' ELSE
        B WHEN sel = '1' ELSE
        "0000";
END func;



-- File: reg_4bits.vhd
-- Description: Four bits register

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY reg_4bits IS
PORT(Data_in: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        rst, clk, en: IN STD_LOGIC;
        Data_out: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END reg_4bits;

ARCHITECTURE func OF reg_4bits IS
BEGIN

PROCESS(rst, clk, en, Data_in)
BEGIN
IF (rst = '1') THEN
    Data_out <= "0000";
ELSIF (clk'EVENT AND clk = '1') THEN
    IF (en = '1') THEN
        Data_out <= Data_in;
    END IF;
END IF;
END PROCESS;
END func;


-- File: ALU_4bits.vhd
-- Description: ALU with eight arithmetic operations and eight logic operations. The operations are realized with two four bits inputs

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY ALU_4bits IS
PORT(A, B: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        opcode: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        cin: IN STD_LOGIC;
        Cout: OUT STD_LOGIC;
        O: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END ALU_4bits;

ARCHITECTURE func OF ALU_4bits IS
BEGIN

PROCESS(A, B, opcode, cin)
VARIABLE carry: STD_LOGIC_VECTOR(4 DOWNTO 0);
BEGIN
CASE opcode IS
    WHEN "0000" =>
    -- Operation: NOT A
        O <= NOT A;
        Cout <= '0';
    WHEN "0001" =>
    -- Operation: NOT B
        O <= NOT B;
        Cout <= '0';
    WHEN "0010" =>
    -- Operation: A AND B
        O <= A AND B;
        Cout <= '0';
    WHEN "0011" =>
    -- Operation: A OR B
        O <= A OR B;
        Cout <= '0';
    WHEN "0100" =>
    -- Operation: A NAND B
        O <= A NAND B;
        Cout <= '0';
    WHEN "0101" =>
    -- Operation: A NOR B
        O <= A NOR B;
        Cout <= '0';
    WHEN "0110" =>
    -- Operation: A XOR B
        O <= A XOR B;
        Cout <= '0';
    WHEN "0111" =>
    -- Operation: A XNOR B
        O <= A XNOR B;
        Cout <= '0';
    WHEN "1000" =>
    -- Operation: Transfer A
        O <= A;
        Cout <= '0';
    WHEN "1001" =>
    -- Operation: Increments A
        carry := '0'&A + '1';
        O <= carry(3 DOWNTO 0);
        Cout <= carry(4);
    WHEN "1010" =>
    -- Operation: Decrements A
        carry := '0'&A - '1';
        O <= carry(3 DOWNTO 0);
        Cout <= carry(4);
    WHEN "1011" =>
    -- Operation: Transfer B
        O <= B;
        Cout <= '0';
    WHEN "1100" =>
    -- Operation: Increments B
        carry := '0'&B + '1';
        O <= carry(3 DOWNTO 0);
        Cout <= carry(4);
    WHEN "1101" =>
    -- Operation: Decrements B
        carry := '0'&B - '1';
        O <= carry(3 DOWNTO 0);
        Cout <= carry(4);
    WHEN "1110" =>
    -- Operation: A + B
        carry := ('0'&A) + ('0'&B);
        O <= carry(3 DOWNTO 0);
        Cout <= carry(4);
    WHEN "1111" =>
    -- Operation: A - B
        O <= A + (NOT B + '1');
        Cout <= '0';
    WHEN OTHERS =>
        O <= "0000";
        Cout <= '0';
    END CASE;
END PROCESS;
END func;



-- File: processor.vhd
-- Description: Combination of the components described previosly to create a sample processor

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY processor_s IS
PORT(A, B: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        opcode: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        load_A, load_B, sel, rst, clk, cin: IN STD_LOGIC;
        A_out, B_out, ALU_o: OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        Cout: OUT STD_LOGIC);
END processor_s;

ARCHITECTURE func OF processor_s IS

-- Multiplexer
COMPONENT mux2to1
PORT(A, B: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        sel: IN STD_LOGIC;
        O: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END COMPONENT;

-- Register
COMPONENT reg_4bits
PORT(Data_in: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        rst, clk, en: IN STD_LOGIC;
        Data_out: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END COMPONENT;

-- Arithmetic Logic Unit
COMPONENT ALU_4bits
PORT(A, B: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        opcode: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        cin: IN STD_LOGIC;
        Cout: OUT STD_LOGIC;
        O: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END COMPONENT;

SIGNAL MuxA_out, MuxB_out: STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL regA_out, regB_out: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL ALU_out: STD_LOGIC_VECTOR(3 DOWNTO 0);
```

```
BEGIN

A_out <= regA_out;  -- A output
B_out <= regB_out;  -- B output
ALU_o <= ALU_out;   -- used to simplify the understanding of the test bench

MuxA: mux2to1 PORT MAP (ALU_out, A, sel, MuxA_out);
MuxB: mux2to1 PORT MAP (ALU_out, B, sel, MuxB_out);
RegA: reg_4bits PORT MAP (MuxA_out, rst, clk, load_A, regA_out);
RegB: reg_4bits PORT MAP (MuxB_out, rst, clk, load_B, regB_out);
ALU:  ALU_4bits PORT MAP (regA_out, regB_out, opcode, cin, Cout, ALU_out);

END func;
```

```
-- File: mux2to1_tb.vhd
-- Description: Multiplexer Test Bench

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY mux2to1_tb IS
END mux2to1_tb;

ARCHITECTURE behavior OF mux2to1_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT mux2to1
    PORT(
         A : IN  std_logic_vector(3 downto 0);
         B : IN  std_logic_vector(3 downto 0);
         sel : IN  std_logic;
         O : OUT  std_logic_vector(3 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal A : std_logic_vector(3 downto 0) := (others => '0');
   signal B : std_logic_vector(3 downto 0) := (others => '0');
   signal sel : std_logic := '0';

    --Outputs
   signal O : std_logic_vector(3 downto 0);


BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: mux2to1 PORT MAP (
          A => A,
          B => B,
          sel => sel,
          O => O
        );

PROCESS
BEGIN
A <= x"4";
B <= x"9";

sel <= '0';

WAIT FOR 10 NS;

sel <= '1';

WAIT FOR 10 NS;

A <= x"7";
B <= x"C";

sel <= '0';

WAIT FOR 10 NS;

A <= x"6";

WAIT FOR 10 NS;

sel <= '1';

WAIT FOR 10 NS;

B <= x"D";

WAIT;
END PROCESS;

END;



-- File: reg_4bits_tb.vhd
-- Description: Register Test Bench.


LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY reg_4bits_tb IS
END reg_4bits_tb;

ARCHITECTURE behavior OF reg_4bits_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT reg_4bits
    PORT(
         Data_in : IN  std_logic_vector(3 downto 0);
         rst : IN  std_logic;
```

```vhdl
         clk : IN  std_logic;
         en : IN  std_logic;
         Data_out : OUT  std_logic_vector(3 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal Data_in : std_logic_vector(3 downto 0) := (others => '0');
   signal rst : std_logic := '0';
   signal clk : std_logic := '0';
   signal en : std_logic := '0';

    --Outputs
   signal Data_out : std_logic_vector(3 downto 0);

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: reg_4bits PORT MAP (
          Data_in => Data_in,
          rst => rst,
          clk => clk,
          en => en,
          Data_out => Data_out
        );

HUE: PROCESS
BEGIN
clk <= NOT clk;
WAIT FOR 1 NS;
END PROCESS;

data: PROCESS
BEGIN
Data_in <= x"A";
en <= '1';

WAIT FOR 10 NS;

en <= '0';

WAIT FOR 10 NS;

Data_in <= x"F";

WAIT FOR 10 NS;

en <= '1';

WAIT FOR 10 NS;

en <= '0';
rst <= '1';

WAIT FOR 10 NS;

en <= '1';

WAIT FOR 10 NS;

rst <= '0';

WAIT FOR 10 NS;

Data_in <= x"5";

WAIT FOR 10 NS;

en <= '0';

WAIT;

END PROCESS;

END;


-- File: ALU_4bits_tb.vhd
-- Description: ALU Test Bench.

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ALU_4bits_tb IS
END ALU_4bits_tb;

ARCHITECTURE behavior OF ALU_4bits_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT ALU_4bits
    PORT(
         A : IN  std_logic_vector(3 downto 0);
         B : IN  std_logic_vector(3 downto 0);
         opcode : IN  std_logic_vector(3 downto 0);
         cin : IN  std_logic;
         Cout : OUT  std_logic;
         O : OUT  std_logic_vector(3 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal A : std_logic_vector(3 downto 0) := (others => '0');
   signal B : std_logic_vector(3 downto 0) := (others => '0');
   signal opcode : std_logic_vector(3 downto 0) := (others => '0');
   signal cin : std_logic := '0';

    --Outputs
   signal Cout : std_logic;
   signal O : std_logic_vector(3 downto 0);

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: ALU_4bits PORT MAP (
          A => A,
          B => B,
          opcode => opcode,
          cin => cin,
          Cout => Cout,
          O => O
        );

    PROCESS
    BEGIN
        A <= X"2";
        B <= X"D";

        --COMMAND 0
        opcode <= X"0";
        -- Operation: NOT A

        WAIT FOR 10 NS;

        A <= X"D";
        B <= X"4";

        --COMMAND 1
        opcode <= X"1";
        -- Operation: NOT B

        WAIT FOR 10 NS;
        A <= X"F";
        B <= X"3";

        --COMMAND 2
        opcode <= X"2";
        -- Operation: A AND B

        WAIT FOR 10 NS;
        A <= X"4";
        B <= X"C";

        --COMMAND 3
        opcode <= X"3";
        -- Operation: A OR B

        WAIT FOR 10 NS;
        A <= X"A";
        B <= X"1";

        --COMMAND 4
        opcode <= X"4";
        -- Operation: A NAND B

        WAIT FOR 10 NS;
        A <= X"D";
        B <= X"4";

        --COMMAND 5
        opcode <= X"5";
        -- Operation: A NOR B

        WAIT FOR 10 NS;
        A <= X"6";
        B <= X"B";

        --COMMAND 6
        opcode <= X"6";
        -- Operation: A XOR B

        WAIT FOR 10 NS;
        A <= X"5";
        B <= X"2";

        --COMMAND 7
        opcode <= X"7";
        -- Operation: A XNOR B

        WAIT FOR 10 NS;
        A <= X"7";
        B <= X"0";
```

```vhdl
        --COMMAND 8
        opcode <= X"8";
        -- Operation: Transfer A

        WAIT FOR 10 NS;
        A <= X"2";
        B <= X"7";

        --COMMAND 9
        opcode <= X"9";
        -- Operation: Increment A

        WAIT FOR 10 NS;
        A <= X"3";
        B <= X"1";

        --COMMAND 10
        opcode <= X"A";
        -- Operation: Decrement A

        WAIT FOR 10 NS;
        A <= X"8";
        B <= X"5";

        --COMMAND 11
        opcode <= X"B";
        -- Operation: Transfer B

        WAIT FOR 10 NS;
        A <= X"7";
        B <= X"5";

        --COMMAND 12
        opcode <= X"C";
        -- Operation: Increment B
        WAIT FOR 10 NS;
        A <= X"8";
        B <= X"4";

        --COMMAND 13
        opcode <= X"D";
        -- Operation: Decrement B
        WAIT FOR 10 NS;
        A <= X"4";
        B <= X"3";

        --COMMAND 14
        opcode <= X"E";
        -- Operation: A + B

        WAIT FOR 10 NS;
        A <= X"2";
        B <= X"4";

        --COMMAND 15
        opcode <= X"F";
        -- Operation: A - B

        WAIT;

    END PROCESS;

END;




-- File: processor_s_tb.vhd
-- Description: Test Bench of the Processor.

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY processor_s_tb IS
END processor_s_tb;

ARCHITECTURE behavior OF processor_s_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT processor_s
    PORT(
         A : IN  std_logic_vector(3 downto 0);
         B : IN  std_logic_vector(3 downto 0);
         opcode : IN  std_logic_vector(3 downto 0);
         load_A : IN  std_logic;
         load_B : IN  std_logic;
         sel : IN  std_logic;
         rst : IN  std_logic;
         clk : IN  std_logic;
         cin : IN  std_logic;
         A_out : OUT  std_logic_vector(3 downto 0);
         B_out : OUT  std_logic_vector(3 downto 0);
         ALU_o : OUT  std_logic_vector(3 downto 0);
         Cout : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal A : std_logic_vector(3 downto 0) := (others => '0');
   signal B : std_logic_vector(3 downto 0) := (others => '0');
   signal opcode : std_logic_vector(3 downto 0) := (others => '0');
   signal load_A : std_logic := '0';
   signal load_B : std_logic := '0';
   signal sel : std_logic := '0';
   signal rst : std_logic := '0';
   signal clk : std_logic := '1';
   signal cin : std_logic := '0';

    --Outputs
   signal A_out : std_logic_vector(3 downto 0);
   signal B_out : std_logic_vector(3 downto 0);
   signal ALU_o : std_logic_vector(3 downto 0);
   signal Cout : std_logic;

BEGIN

   uut: processor_s PORT MAP (
          A => A,
          B => B,
          opcode => opcode,
          load_A => load_A,
          load_B => load_B,
          sel => sel,
          rst => rst,
          clk => clk,
          cin => cin,
          A_out => A_out,
          B_out => B_out,
          ALU_o => ALU_o,
          Cout => Cout
        );

pclk: PROCESS
BEGIN
    clk <= NOT clk;
    WAIT FOR 1 NS;
END PROCESS;

data:PROCESS
BEGIN

-- COMMAND 0
-- OPERATION: NOT A

    A <= x"1";
    B <= x"3";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;

    opcode <= "0000";

    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;


-- COMMAND 1
-- OPERATION: NOT B


    A <= x"2";
    B <= x"8";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;

    opcode <= "0001";

    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;


-- COMMAND 2
-- OPERATION: A AND B
```

```
    A <= x"E";
    B <= x"2";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;

    opcode <= "0010";

    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;

-- COMMAND 3
-- OPERATION: A OR B


    A <= x"5";
    B <= x"D";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;

    opcode <= "0011";

    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;

-- COMMAND 4
-- OPERATION: A NAND B


    A <= x"7";
    B <= x"3";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;

    opcode <= "0100";

    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;

-- COMMAND 5
-- OPERATION: A NOR B


    A <= x"C";
    B <= x"6";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;
    opcode <= "0101";
    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;

-- COMMAND 6
-- OPERATION: A XOR B


    A <= x"A";
    B <= x"3";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;
    opcode <= "0110";
    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;

-- COMMAND 7
-- OPERATION: A XNOR B


    A <= x"1";
    B <= x"D";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;
        opcode <= "0111";
    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;

-- COMMAND 8
-- OPERATION: TRANSFER A


    A <= x"2";
    B <= x"5";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;
        opcode <= "1000";
    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;

-- COMMAND 9
-- OPERATION: INCREMENT A


    A <= x"A";
    B <= x"1";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;
    opcode <= "1001";
    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;

-- COMMAND 10
-- OPERATION: DECREMENT A


    A <= x"5";
    B <= x"2";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;
    opcode <= "1010";
    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;

-- COMMAND 11
-- OPERATION: TRANSFER B
```

```vhdl
    A <= x"F";
    B <= x"3";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;
    opcode <= "1011";
    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;

-- COMMAND 12
-- OPERATION: INCREMENT B


    A <= x"4";
    B <= x"C";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;

    opcode <= "1100";

    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;


-- COMMAND 13
-- OPERATION: DECREMENT B


    A <= x"3";
    B <= x"5";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;

    opcode <= "1101";

    WAIT FOR 1 NS;


    load_A <= '0';
    load_B <= '0';


    WAIT FOR 2 NS;

-- COMMAND 14
-- OPERATION: A + B


    A <= x"4";
    B <= x"9";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;

    opcode <= "1110";

    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;

-- COMMAND 15
-- OPERATION: A - B


    A <= x"7";
    B <= x"3";
    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;

    opcode <= "1111";

    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;


-- CHECKING FOR CARRY OUT IN THE SUM
-- COMMAND 14
-- OPERATION: A + B


    A <= x"5";
    B <= x"C";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;

    opcode <= "1110";

    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;

-- CHECKING FOR ALU_out TRANSFER
-- COMMAND 12
-- OPERATION: INCREMENT B

    A <= x"2";
    B <= x"3";

    sel <= '1';

    load_A <= '1';
    load_B <= '1';

    WAIT FOR 1 NS;

    opcode <= "1100";

    WAIT FOR 1 NS;

    load_A <= '0';
    load_B <= '0';

    WAIT FOR 2 NS;

    sel <= '0';

    load_A <= '1';

    WAIT FOR 2 NS;

    load_A <= '0';

WAIT;

END PROCESS;

END;
```