University of Essex

Department of Computer Science and Electronic Engineering

Assignment 2: CPU Design
CE869: High Level Logic Design

Registration number: **2311849**

Total word Count: 1950

1 ABSTRACT

In this project, 16-bit Central Processing Unit (CPU) is designed and verified, focusing on general Datapath and control unit. Utilizing a modular coding approach, alongside generic and generate commands, the CPU design ensures flexibility and scalability. The top-down methodology employed in this project emphasizes thorough RTL analysis and simulation-based verification, ensuring functionality and correctness at each module level.

2 INTRODUCTION

This project presents the design and verification of a 16-bit CPU modified to address a specific task which is computing the sum of natural numbers less than a given nonzero integer N. The CPU is equipped with a general Datapath and control unit, designed to efficiently execute instructions derived from the problem statement. Notably, the input N is provided through 16 switches on the FPGA Basys 3 board, while the output, representing the computed sum, is displayed on a 7-segment display.

3 OVERVIEW

Control Processing Unit (CPU) is a simply a dedicated microprocessor that only executes software instructions [1]. Microprocessor is an integrated circuit that contains all the functions of a CPU of a computer. Basic block diagram Von Neumann model of a computer is given in Figure 1.

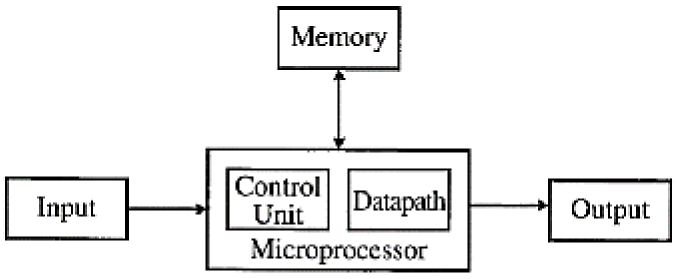


Figure 1: Components of Von Neumann model computer [2]

The 16-bit CPU design comprises of general data path and control path, which collectively handle all data operations within the microprocessor. The general data path is divided into two main components: The Arithmetic/Logic Instruction Datapath and the Control Flow Instruction Datapath.

Within the Arithmetic/Logic Instruction Datapath, there are various components including multiplexers for selecting inputs from instructions, switches and the output of Arithmetic Logic Unit (ALU). Register Files (RF) facilitates both writing too and reading from specific addresses, controlled by signals from the control unit. The ALU is responsible for executing arithmetic and logic operations on data stored in the register file or received from input lines, guided b control signals. The 'Z' status register holds 1-bit value crucial for conditional jumps, indicating whether the result of the previous operation was zero. Additionally, tri-state buffers enable updating the data after an "OUT" instruction.

On the other hand, the Control Flow Instruction Datapath manages the determination of the next program counter value, which serves as the 4-bit address of the instruction stored in Read-Only Memory (ROM). The Instruction Register (IR) loads the instruction at the current program counter and select its type. The driver multiplexer select pin plays a key role in deciding whether the new program address increment by 1 or is set by the instruction type.

The Control Unit crucially combines all operations within the data path and the microprocessor. Implemented using a Finite State Machine (FSM), the Control Unit cycles through various states, including states. During the fetch state, control signals such as PCload and IRload are set to retrieve the memory location specified by the Program Counter (PC) and transfer its content into the Instruction Register (IR). The decode state interprets bits from the IR to identity the current instruction to be executed, guiding subsequent operations. Finally, when in the current instruction state, appropriate control signals are asserted to the Arithmetic/Logic Instruction Datapath for the execution of the specified instructions.

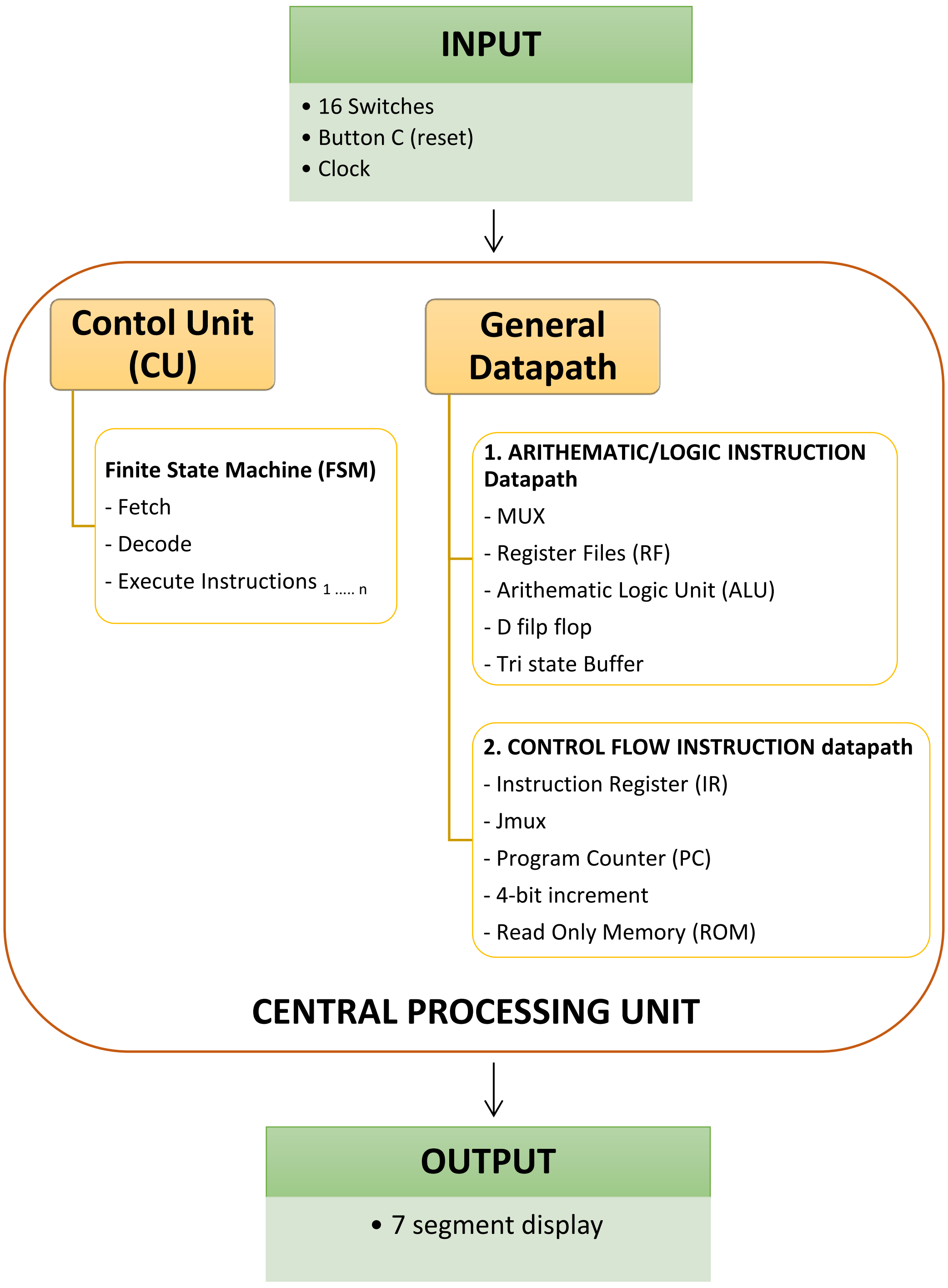


Figure 2: Block Diagram of Implemented CPU

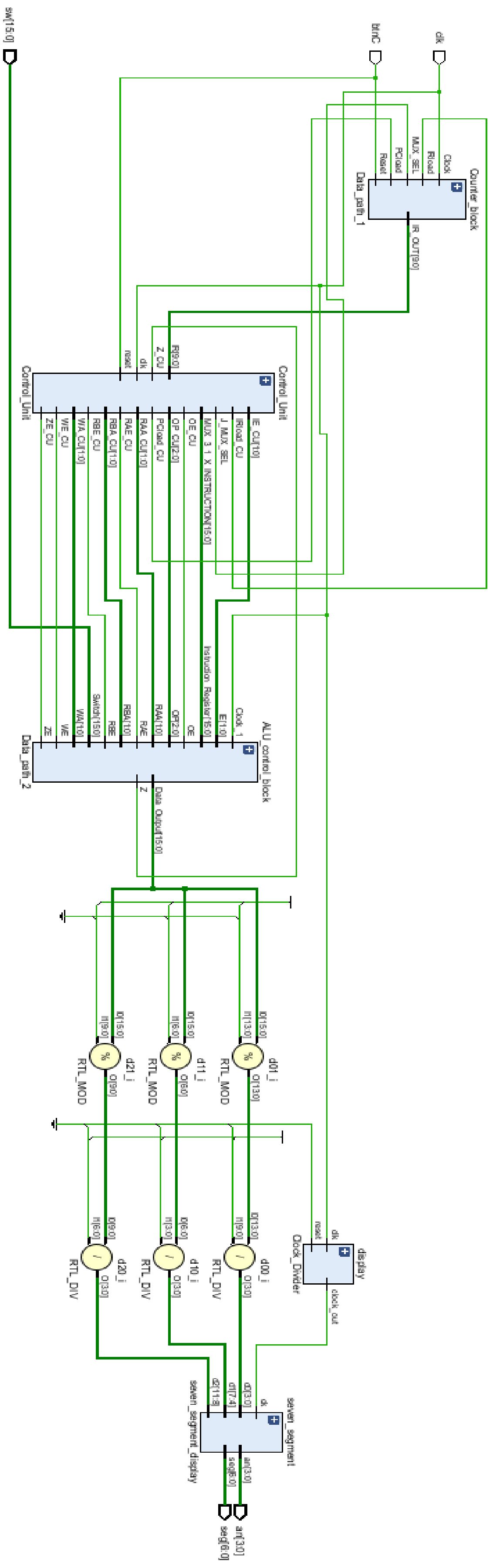


Figure 3: Schematic of Top-Level CPU

4 GENERAL DATAPATH

The Datapath is responsible for actual execution of all data operations performed by the microprocessor [2]. As mentioned above in Figure 2 consists of "Arithmetic/Logic instruction" data path and "Control flow Instruction" Datapath. Working flow and explanation of each data paths and their modules is explained in this section:

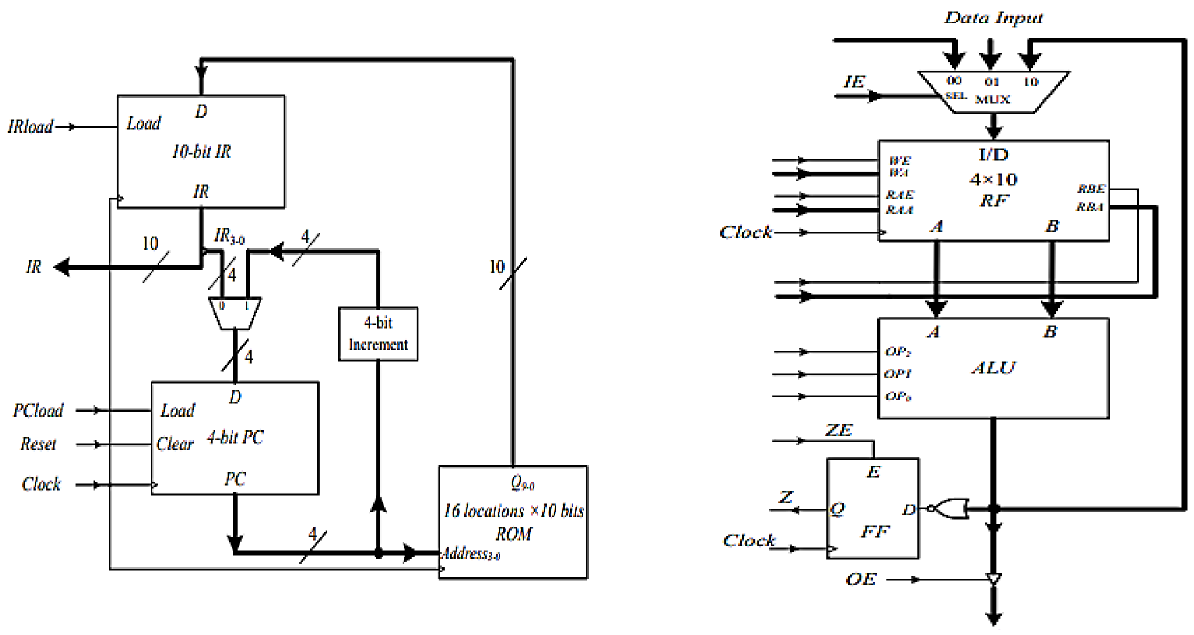


Figure 4: Datapath consisting of Control Flow Instruction and Arithmetic/Logic instruction data paths [3]

4.1 CONTROL FLOW INSTRUCTIONS DATA PATH

This Datapath is used for deciding the next program counter value. Program Counter (PC) should be increased by 1 following these instructions: MOV, IN, OUT, NOT, LT, INC, DEC, ADD. PC should be set to 4 least significant bits of instruction following JMP, JNZ (only if Z status flag is 0, otherwise PC=PC+1), JN (only if Z status flag is 1, otherwise PC=PC+1). For example: JMP – 0101000011 instruction should set PC to 0011 (meaning that the next executed instruction will come from ROM address 3) The decision is made by the following procedure [4]:

- Load instruction at current program counter (address), drive IRload pin.
- Check what instruction type it is (decode instruction), read IR.
- Drive multiplexer select pin to decide if the new program address is increased by 1 (select='1') or set by the instruction (select='0').
- Load newly selected program counter by driving PCload.

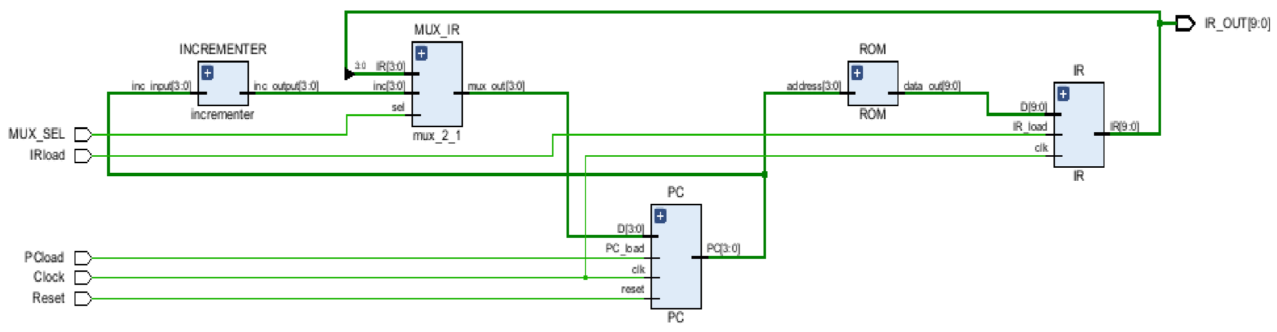


Figure 5: Top Level view of Control Flow Instruction Datapath

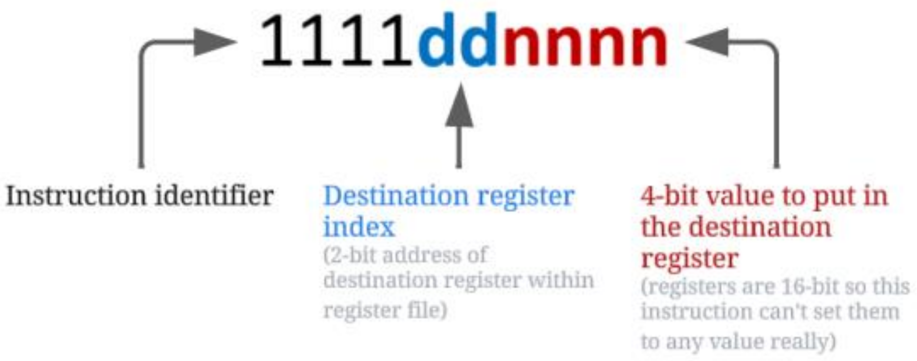
4.1.1 Read Only Memory (ROM)

ROM has 16 memory locations and of each having 10-bits. The input address is 4bit and output is 10 bits. Each instruction has different functionality as shown in Table 1. It is responsibility of the control unit within CPU to recognize what should be done when specific instruction is about to be executed and ensure that happens.

```
    "0010000011", -- IN Rdd //0// READ INPUT TO R3
   "1111000001", -- MOV Rdd, #nnnn //1// INIT R0 = 1
   "1111010000", -- MOV Rdd, #nnnn //2// INIT R1 = 0
   "1011010001", -- ADD Rdd, Rrr, Rqq //3// R1 = R0 + R1
   "1001000001", -- INC Rrr, #nnnn //4// R0 = R0 + 1
   "1000000011", -- LT Rrr, Rqq //5// IF R0<R3 THEN Z = 0 ELSE Z = 1
   "0110000011", -- JNZ aaaa //6// IF Z == 0 THEN GO ADDR 03 ELSE GO
NEXT ADDR
   "0011000001", -- OUT Rss //7// OUTPUT R1
   others => ("0000000000") -- HALT //8// OVER
```

Instruction	Encoding	Affects	Operation
HALT	0000000000		Halt
MOV R*dd*, R*ss*	000100ddss		R*dd* ← R*ss*
IN R*dd*	00100000dd		R*dd* ← Input
OUT R*ss*	00110000ss		Output ← R*ss*
NOT R*dd*, R*ss*	010000ddss	Z	R*dd* ← NOT R*ss*
JMP *aaaa*	010100aaaa		Jump to *aaaa*
JNZ *aaaa*	011000aaaa		Jump to *aaaa* if Z status flag is unset (i.e. is 0)
JN *aaaa*	011100aaaa		Jump to *aaaa* if Z status flag is set (i.e. is 1)
LT R*rr*, R*qq*	100000rrqq	Z	Set Z to 0 if R*rr* < R*qq*, to 1 otherwise
INC R*rr*, #*nnnn*	1001rrnnnn	Z	R*rr* ← R*rr* + *nnnn*
DEC R*rr*, #*nnnn*	1010rrnnnn	Z	R*rr* ← R*rr* - *nnnn*
ADD R*dd*, R*rr*, R*qq*	1011ddrrqq	Z	R*dd* ← R*rr* + R*qq*
SUB R*dd*, R*rr*, R*qq*	1100ddrrqq	Z	R*dd* ← R*rr* - R*qq*
AND R*dd*, R*rr*, R*qq*	1101ddrrqq	Z	R*dd* ← R*rr* AND R*qq*
OR R*dd*, R*rr*, R*qq*	1110ddrrqq	Z	R*dd* ← R*rr* OR R*qq*
MOV R*dd*, #*nnnnnnn*	1111ddnnnn		R*dd* ← *nnnn*

Table1: Table of Opcodes [3]



1111ddnnnn
Instruction identifier
Destination register index
(2-bit address of destination register within register file)
4-bit value to put in the destination register
(registers are 16-bit so this instruction can't set them to any value really)

Figure 6: Example format of "MOV Rdd, #nnnn" instruction [4]

4.2 ARITHMATIC/LOGIC INSTRUCTION DATAPATH

Arithmetic/Logic Instruction Datapath consists of multiplexer, Register Files, Arithmetic Logic Unit (ALU), Flipflops. Figure 7 shows the top-level view of Arithmetic/Logic Instruction Datapath.

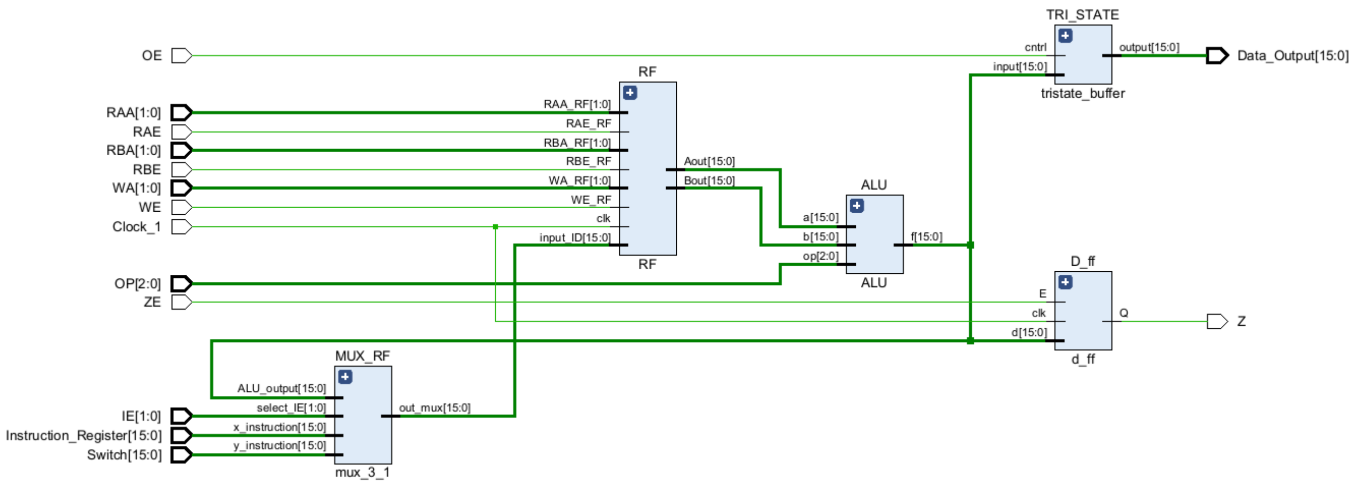


Figure 7: Top Level view of Arithmetic/Logic Instruction Datapath

4.2.1 Multiplexer

Multiplexer basically has 4 inputs. The output is depending on the select pin IE. Inputs are of three types: 1. Input from Instruction itself (this input is coming from the decode of the Control Unit). 2. Input from the switches of Basys3 FPGA board. 3. ALU output.

```
if select_IE = "00" then -- Input from Instruction
       out_mux <= x_instruction;
elsif select_IE = "01" then -- Input from Switches
       out_mux <= y_instruction;
```

```
     elsif select_IE = "10" then -- Input from ALU out
            out_mux <= z_instruction;
     elsif select_IE = "11" then -- Input from ALU out
            out_mux <= z_instruction;
```

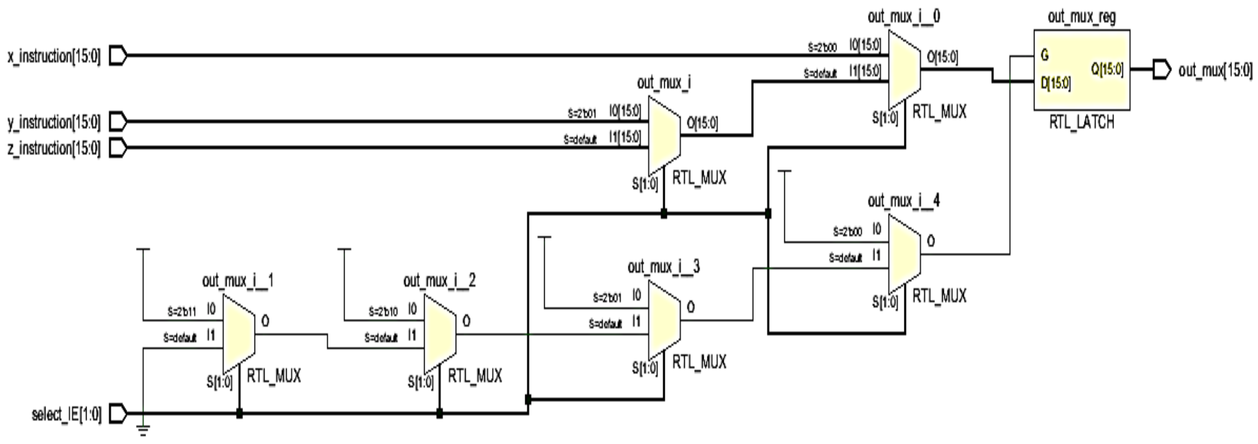


Figure 8: RTL Schematic of MUX

4.2.2 Dedicated Register Files

Register file is an array of registers in a CPU. It is a small memory within CPU with 4 locations, each storing 16-bits. Unlike Static Random Access Memory (SRAMs), it has dedicated read and write ports (ordinary SRAMs usually read and write through the same ports). It is usually used for the source operands and destination of the ALU and therefore it has one write port and two read ports. Since the ALU normally takes two input operands, the register file should be able to output two values from two different locations at the same time. Write is synchronous while read is often asynchronous: in a single clock, data can be read, transformed and written back [2].

Below code from [2] is used for Register file. The “write port” handles writing data to the register file based on the write enable signal and address. Read operations are performed based on the read enable signals for 2 ports (A and B), outputting data from the register file. When rising edge of clock, check if Write Enable is set to ‘1’, if yes → Write the input to the specified address. To read port A and B check if the respective enable is ‘1’. When RAE and/or RBE are low, the corresponding outputs will simply match the input “I” o the register file.

```
wait until rising_edge(clk);
        if WE_RF = '1' then
            reg_file (to_integer(unsigned(WA_RF))) <= input_ID;
        end  if;
   end process;
   -- READ PORT A
   Aout <= reg_file (to_integer(unsigned(RAA_RF))) when RAE_RF = '1'
   else (others => '0');
   -- READ PORT B
   Bout <= reg_file (to_integer(unsigned(RBA_RF))) when RBE_RF = '1'
   else (others => '0');
```

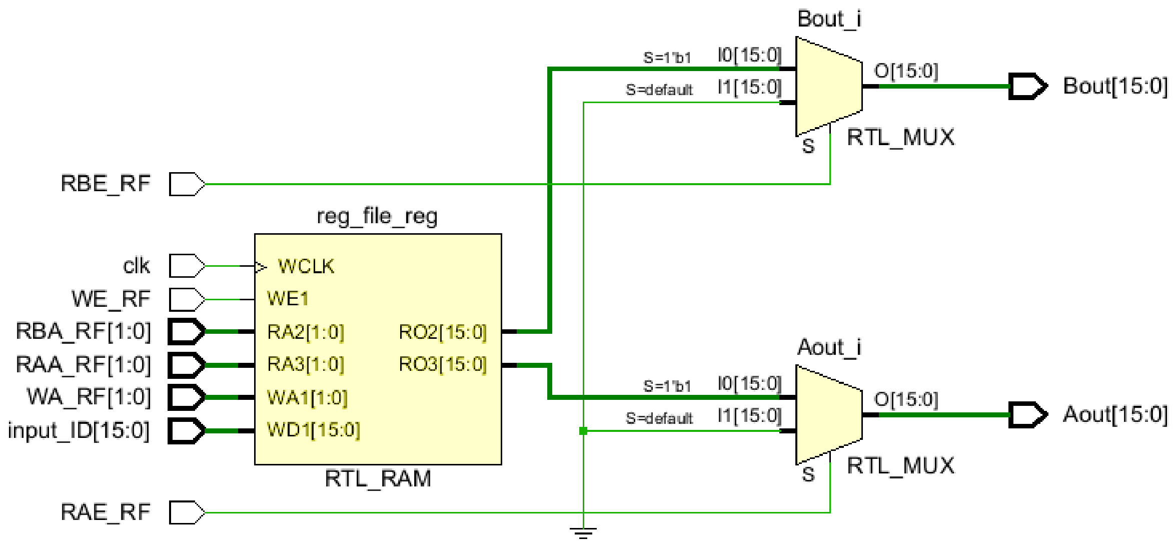


Figure 9: RTL Schematic of Register File (RF)

4.2.3 Arithmetic Logic Unit (ALU)

ALU is used to perform operations such as Pass, Logical AND, Logical OR, Logical NOT, Addition, Subtraction, Increment and Decrement based on the op codes as shown in Table 2. ALU consists of a four-bit adder, the logic extender (LE) which is used for manipulating all logic operations, the arithmetic extender (AE) which is used for manipulating all arithmetic operations. The carry extender (CE) which is used for modifying the primary carry-in signal, so that arithmetic operations are performed correctly.

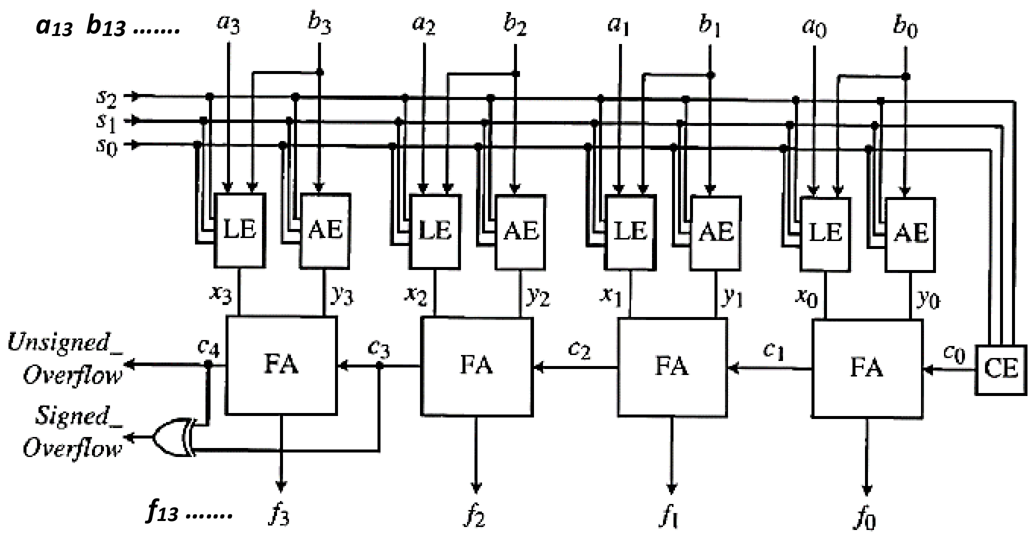


Figure 10: Block Diagram of ALU [4]

s_2	s_1	s_0	Operation	x_i (LE)	$y_i(AE)$	c_0(CE)
0	0	0	Pass	a_i	0	0
0	0	1	AND	a_i AND b_i	0	0
0	1	0	OR	a_i OR b_i	0	0
0	1	1	NOT	a'_i	0	0
1	0	0	Addition	a_i	b_i	0
1	0	1	Subtraction	a_i	b'_i	1
1	1	0	Increment	a_i	0	1
1	1	1	Decrement	a_i	1	0

Table 2: Truth Table of ALU [4]

Since the modules Logic Extender, Arithmetic Extender and ALU is 1 bit, but in this design, we are using 16bit inputs. To implement this “Generate command” is used in order to ease the coding.

```
g_GENERATE_FOR_LE_AE: for i in 0 to 15 generate
  arithematic_extender: entity work.arithematic_extender
      Port map (b_AE => b(i),
                y_AE => y(i),
                s_AE => op
               );
  logic_extender: entity work.logic_extender
      Port map (a_LE => a(i),
                b_LE => b(i),
                x_LE => x(i),
                    s_LE => op
                   );
   end generate g_GENERATE_FOR_LE_AE;
full_adder_0: entity work.full_adder
        Port map (x_FA => x(0), --LE output
                  y_FA => y(0), --AE output
                  carry_in_FA => CE_out,
                  sum_FA => f(0),
                  carry_out_FA => carry_wire(0)
                  );
   g_GENERATE_FOR_full_adder_1_15: for i in 1 to 15 generate
       full_adder_0: entity work.full_adder
        Port map (x_FA => x(i), --LE output
                  y_FA => y(i), --AE output
                  carry_in_FA => carry_wire(i-1),
                  sum_FA => f(0),
                  carry_out_FA => carry_wire(0)
                  );
  end generate g_GENERATE_FOR_full_adder_1_15;
```

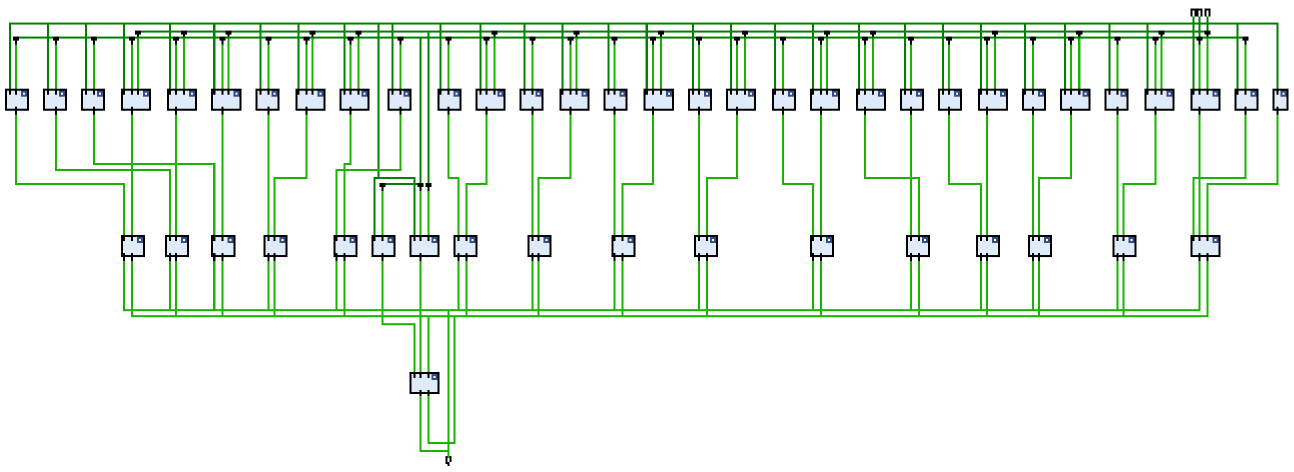


Figure 11: RTL Schematic of ALU

4.2.4 Data Flip-Flop

This flip-flop is to update the "Z" status flag. Output of the ALU is connected to NOR gate. If all 16-bit outputs are '0' output is '1' whereas if any bit is '1' the output will be '0'. Output of NOR gate (either '1' or '0') is connected to the D-ff. If enable is high and rising edge of clock, output is followed by the value of input.

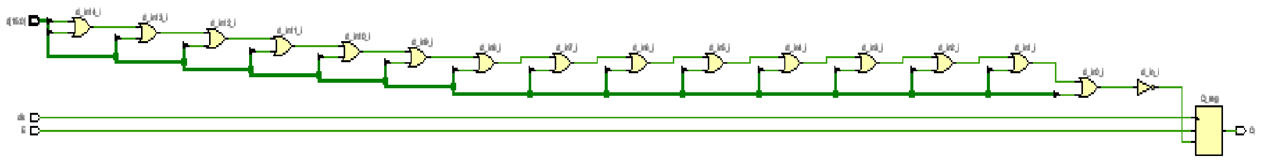


Figure 12: RTL Schematic of D-ff

Z status flag: It is a 1-bit value stored inside processor that allows to check if the result of previous operation/instruction (e.g. addition, subtraction) was equal to 0. This allows the processor to do conditional jumps (e.g., JNZ instruction will perform jump only if the result of previous Z-affecting instruction was not equal to zero) [4].

4.2.5 Tristate Buffer

It has three states 0, 1, and Z. The value Z represents a high-impedance state, which acts like a switch that is opened. It is used to connect several devices to the same bus. If two or more devices are connected directly to a bus without using tri-state buffer, signals will get corrupted on the bus.

```
if cntrl = '1' then
            output <= input;
        else
            output <= (others => 'Z') ;
        end if;
```

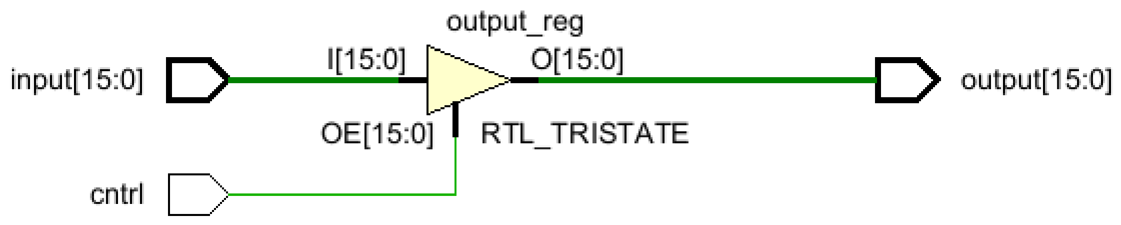


Figure 13: RTL Schematic of Tri-state buffer

5 CONTROL UNIT

When it's in the "fetch" state, it sets signals to grab the memory location pointed to by the Program Counter (PC) and puts that information into the Instruction Register (IR). In the "decode" state, it reads a set of instructions. It looks at the bits in the IR to figure out what needs to be done next. Finally, in “Instruction state”, it sends signals to the part of the microprocessor that does the actual calculations or operations, telling it what to do based on the instruction that was decoded earlier.

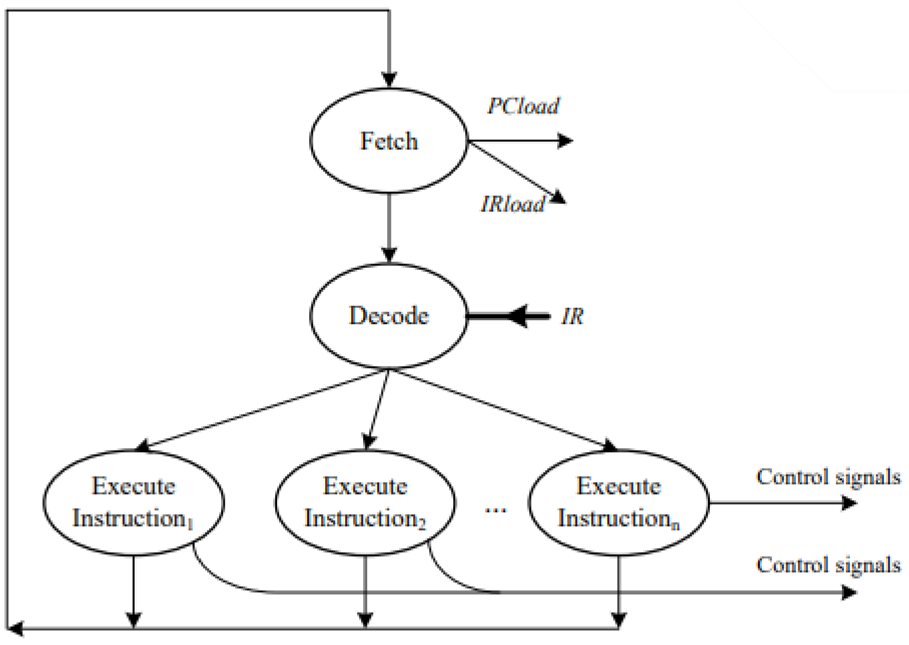


Figure 14: Control Unit FSM representation [3]

6 7 SEGMENT DISPLAY

7 segment is using the clock divider with a threshold value of 500,000 which is using generic method of code. The clock divider generates signal based on the system clock. This clock divided clock signal is then used for the clock for a seven-segment display.

```
display : entity work.Clock_Divider generic map (
            THRESHOLD => 500000
        )
        port map (.
            clk => clk,
            reset => '0',
            clock_out => clk_out_display
        );
```

The seven-segment display is responsible for driving the seven-segment display with the appropriate signals for each segment to display a numeric value. In this instantiation, various signals such as "an" (anodes), "seg" (segments), and "d0" through "d3" (display digits) are mapped to corresponding signals in the design.

Further, the value from output of Arithmetic/Logic Instruction Datapath is converted into individual digits. It does this by taking the modulus of the 10, 100, 1000 and 10000 to extract each digit from the rightmost to the leftmost. This process effectively decomposes the input numerical value into individual digits for display on the seven-segment display.

```
display_to_digits: process (display_signal)
        begin
            d0_signal <=
std_logic_vector(to_unsigned((to_integer(unsigned(display_signal))
mod 10), d0_signal'length));
            d1_signal <=
std_logic_vector(to_unsigned((to_integer(unsigned(display_signal))
mod 100)/10, d1_signal'length));
            d2_signal <=
std_logic_vector(to_unsigned((to_integer(unsigned(display_signal))
mod 1000)/100, d2_signal'length));
            d0_signal <=
std_logic_vector(to_unsigned((to_integer(unsigned(display_signal))
mod 10000)/1000, d3_signal'length));
         end process;
```

Overall, the CPU is implemented as shown in Figure 15.

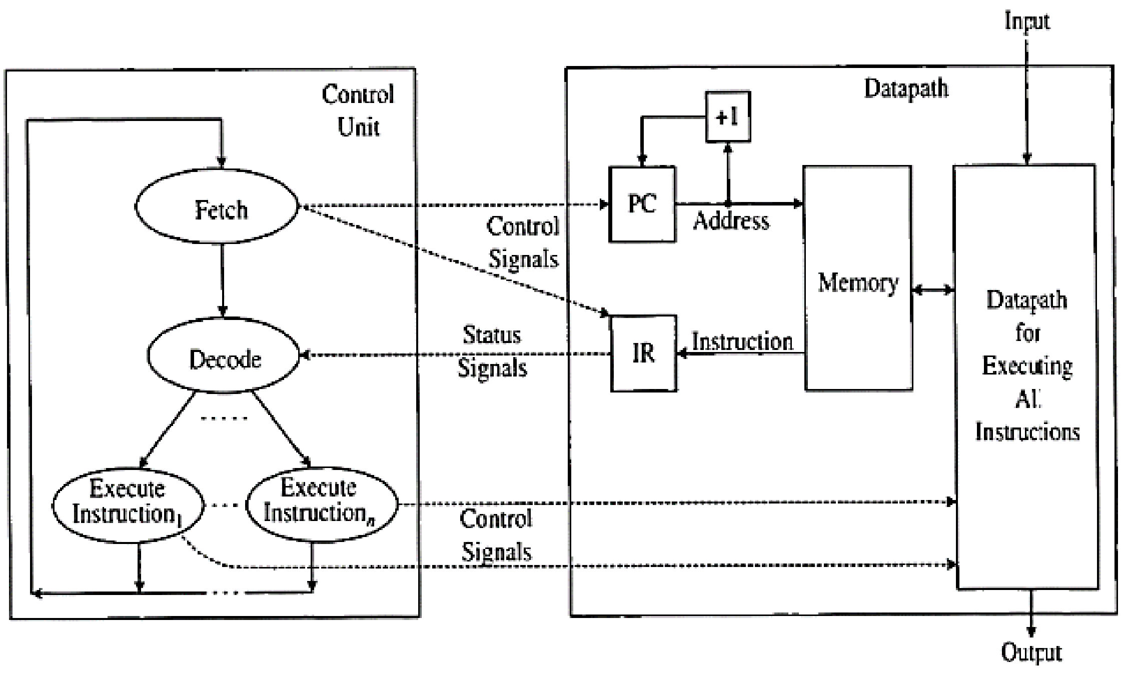


Figure 15: CPU

7 VERIFICATION

Verification included Register Transfer Level (RTL) analysis of each module to ensure functionality and correctness. Once each module was verified, top level modules Arithmetic/Logic Datapath, Control Flow Datapath, Control Unit and the high-level CPU was verified. To validate functionality, a testbench code simulated toggling the clock signal every 20 nanoseconds to drive CPU operations.

```
while TRUE loop
  clk <= '0';
  wait for 20ns;
  clk <= '1';
  wait for 20ns
```

8 CONCLUSION

In conclusion, the thorough verification process of our CPU design, ensuring each module functions flawlessly at the RTL level without any errors in the code. The analysis of the top-level RTL further strengthens confidence in the design's robustness and suitability for practical deployment.

Through this exercise, I gained knowledge on internal workings of a 16-bit CPU, operations of different Datapath, the significance of instructions and opcodes, and the efficient handling of read and write operations within register files. This understanding serves as a solid foundation for grasping higher-level designs, such as 32-bit and 64-bit CPUs, empowering me to delve into complex microprocessor architectures. Also, the basic components like adders, counters, and ALUs, when integrated, can function as a real processor, showcasing their role in real-world scenarios.

Looking ahead, I'm inspired to apply this knowledge on designing more advanced CPUs, including 32-bit and 64-bit variants. Additionally, I'm eager to explore and implement architectures like RISC-V and CISC. Adopting a Microcode Control Unit approach helps in scaling complex microprocessors, bug fixes and implement specialized microprocessors (different instruction sets) with same underlying hardware micro-architecture.

9 REFERENCES

[1] W. Yi, General Purpose Microprocessor Design, Essex: School of Computer Science and Electronic Engineering, University Of Essex (UK), 2024, p. 27.

[2] W. Yi, Microprocessor Design, Essex: School of Computer Science and Electronic Engineering, University Of ESSEX (UK), 2024.

[3] D. W. Yi, CE869 Assignment 2: CPU Design, Essex: CSEE - University of Essex, 2024.

[4] M. Borowski, Support Notes for Assignment 2 (CPU design), ESSEX: University of Essex, CE869 High Level Logic Design, 22nd February 2024.

APPENDICES

```vhdl
----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 20.03.2024 01:40:14
-- Design Name:
-- Module Name: CPU - 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 leaf cells in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity CPU is
    Port ( sw : in STD_LOGIC_VECTOR (15 downto 0);
           clk : in STD_LOGIC;
           btnC : in STD_LOGIC;
           an : out STD_LOGIC_VECTOR (3 downto 0);
           seg : out STD_LOGIC_VECTOR (6 downto 0)
           );
end CPU;

architecture Behavioral of CPU is

    signal IRload_signal : STD_LOGIC;
    signal IE_signal : STD_LOGIC_VECTOR (1 downto 0);
    signal WE_signal : STD_LOGIC;
    signal WA_signal : STD_LOGIC_VECTOR (1 downto 0);
    signal RAE_signal : STD_LOGIC;
    signal RAA_signal : STD_LOGIC_VECTOR (1 downto 0);
    signal RBE_signal : STD_LOGIC;
    signal RBA_signal : STD_LOGIC_VECTOR (1 downto 0);
    signal OP_signal : STD_LOGIC_VECTOR (2 downto 0);
    signal ZE_signal : STD_LOGIC;
    signal Z_signal : STD_LOGIC;
    signal OE_signal : STD_LOGIC;
    signal J_MUX_SEL_signal : STD_LOGIC;
    signal MUX_3_1_X_INSTRUCTION_signal: STD_LOGIC_VECTOR(15 downto 0);
    signal IR_signal : STD_LOGIC_VECTOR (9 downto 0);
    signal PCload_signal : STD_LOGIC;
    signal display_signal : STD_LOGIC_VECTOR(15 downto 0);
    signal clk_out_display: STD_LOGIC;
    signal d0_signal : STD_LOGIC_VECTOR (3 downto 0);
    signal d1_signal : STD_LOGIC_VECTOR (3 downto 0);
    signal d2_signal : STD_LOGIC_VECTOR (3 downto 0);
    signal d3_signal : STD_LOGIC_VECTOR (3 downto 0);

begin

   ALU_control_block: entity work.Data_path_2 port map (
           Instruction_register => MUX_3_1_X_INSTRUCTION_signal, ---check
           Switch => sw,
           IE => IE_signal,
           WE => WE_signal,
           WA => WA_signal,
           RAE => RAE_signal,
           RAA => RAA_signal,
           RBE => RBE_signal,
           RBA => RBA_signal,
           Clock_1 => clk,
           OP => OP_signal,
           ZE => ZE_signal,
           Z => Z_signal,
           OE => OE_signal,
           Data_Output => display_signal
    );

    Counter_block: entity work.Data_path_1 port map (
           IRload => IRload_signal,
           PCload => PCload_signal,
           Reset => btnC,
           Clock => clk,
           MUX_SEL => J_MUX_SEL_signal,
           IR_OUT => IR_signal
    );

    Control_Unit: entity work.Control_Unit port map (
           PCload_CU => PCload_signal,
           IRload_CU => IRload_signal,
           IE_CU => IE_signal,
           WE_CU => WE_signal,
           WA_CU => WA_signal,
           RAE_CU => RAE_signal,
           RAA_CU => RAA_signal,
           RBE_CU => RBE_signal,
           RBA_CU => RBA_signal,
           OP_CU => OP_signal,
           ZE_CU => ZE_signal,
           Z_CU => Z_signal,
           OE_CU => OE_signal,
           J_MUX_SEL => J_MUX_SEL_signal,
           MUX_3_1_X_INSTRUCTION => MUX_3_1_X_INSTRUCTION_signal,
           IR => IR_signal,
           clk => clk,
           reset => btnC
    );
```

```
        display : entity work.Clock_Divider generic map ( --This line 
instantiates an entity named "Clock_Divider" from the library work
            THRESHOLD => 500000 -- THRESHOLD generic parameter of the 
Clock_Divider entity will take the value 500000
        )
        port map ( --This part maps the ports of the instantiated entity 
Clock_Divider to other signals or ports in my design.
            clk => clk, -- clk is mapped to a signal named CLK100MHZ
            reset => '0', --reset is mapped to a constant value '0'
            clock_out => clk_out_display --clock_out is mapped to signal 
clk_out_display
        );

    seven_segment: entity work.seven_segment_display port map(
           ck => clk_out_display,
           an => an,
           seg => seg,
           d0  => d0_signal,
           d1  => d1_signal,
           d2  => d2_signal,
           d3  => d3_signal
    );

    display_to_digits: process (display_signal)
        begin
            d0_signal <= 
std_logic_vector(to_unsigned((to_integer(unsigned(display_signal)) mod 10), 
d0_signal'length));
            d1_signal <= 
std_logic_vector(to_unsigned((to_integer(unsigned(display_signal)) mod 
100)/10, d1_signal'length));
            d2_signal <= 
std_logic_vector(to_unsigned((to_integer(unsigned(display_signal)) mod 
1000)/100, d2_signal'length));
            d0_signal <= 
std_logic_vector(to_unsigned((to_integer(unsigned(display_signal)) mod 
10000)/1000, d3_signal'length));
         end process;
end Behavioral;
```

```
----------------------------------------------------------------------------
-------
-- Company: 
-- Engineer: 
-- 
-- Create Date: 13.03.2024 13:26:27
-- Design Name: 
-- Module Name: 4_bit_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 leaf cells in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity PC is
    Port (
         D : in STD_LOGIC_VECTOR (3 downto 0);
         PC_load : in STD_LOGIC;
         reset : in STD_LOGIC;
         clk : in STD_LOGIC;
         PC : out STD_LOGIC_VECTOR (3 downto 0));
end PC;

architecture Behavioral of PC is

begin

    process (clk)
        begin
        if reset = '1' then
            PC <= (others => '0');
        elsif PC_load = '1' then
            if rising_edge (clk) then
                PC <= D ;
            end if;
        end if;
    end process;

end Behavioral;


----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 21.02.2024 10:48:21
-- Design Name:
-- Module Name: ALU - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
```

```vhdl
-- 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 leaf cells in this code.
library UNISIM;
use UNISIM.VComponents.all;

ENTITY ALU IS
  PORT (
         a : IN std_logic_vector (15 DOWNTO 0);
         b : IN std_logic_vector (15 DOWNTO 0);
         op : IN std_logic_vector (2 DOWNTO 0);
         f : OUT std_logic_vector (15 DOWNTO 0)
       );
END ALU;

ARCHITECTURE BHV_ALU OF ALU IS

    signal x : std_logic_vector(15 downto 0); --LE output
    signal y : std_logic_vector(15 downto 0); --AEoutput
    signal CE_out: std_logic;
    signal carry_wire: std_logic_vector (16 DOWNTO 0);
    signal unsigned_overflow : std_logic;
    signal signed_overflow : std_logic;

begin

    carry_extender : entity work.carry_extender
        Port map (cout_CE => CE_out,
                  s_CE => op
                 );

  g_GENERATE_FOR_LE_AE: for i in 0 to 15 generate

    arithematic_extender: entity work.arithematic_extender
        Port map (b_AE => b(i),
                  y_AE => y(i), --AE output
                  s_AE => op
                 );

    logic_extender: entity work.logic_extender
        Port map (a_LE => a(i),
                  b_LE => b(i),
                  x_LE => x(i), --LE output
                  s_LE => op
                 );
   end generate g_GENERATE_FOR_LE_AE;


    full_adder_0: entity work.full_adder
        Port map (x_FA => x(0), --LE output
                  y_FA => y(0), --AE output
                  carry_in_FA => CE_out,
                  sum_FA => f(0),
                  carry_out_FA => carry_wire(0)
                  );

   g_GENERATE_FOR_full_adder_1_15: for i in 1 to 15 generate
       full_adder_0: entity work.full_adder
        Port map (x_FA => x(i), --LE output
                  y_FA => y(i), --AE output
                  carry_in_FA => carry_wire(i-1),
                  sum_FA => f(0),
                  carry_out_FA => carry_wire(0)
                  );

  end generate g_GENERATE_FOR_full_adder_1_15;


END BHV_ALU;


----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 06.03.2024 11:24:42
-- Design Name:
-- Module Name: arithematic_extender - 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 leaf cells in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity arithematic_extender is
    Port (
           s_AE : in STD_LOGIC_VECTOR (2 downto 0);
           b_AE : in STD_LOGIC;
           y_AE : out STD_LOGIC);
end arithematic_extender;
```

```
architecture Behavioral of arithematic_extender is

begin
    process
    begin
        case (s_AE) is
            when "000" =>  -- PASS
                y_AE <= '0';
            when "001" =>  -- AND
                y_AE <= '0';
            when "010" =>  -- OR
               y_AE <= '0';
            when "011" =>  -- NOT
                y_AE <= '0';
            when "100" =>  -- ADDITION
                y_AE <= b_AE;
            when "101" =>  -- SUBTRACTION
                y_AE <= b_AE;
            when "110" =>  -- INCREMENT
                y_AE <= '0';
            when "111" =>  -- DECREMENT
                y_AE <= '0';
            when others =>  --changed this be carefullllllllllll
                y_AE <= '0';
        end case;
    end process;


end Behavioral;


----------------------------------------------------------------------------
------
-- Company:
-- Engineer:
--
-- Create Date: 06.03.2024 11:42:00
-- Design Name:
-- Module Name: carry_extender - 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 leaf cells in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity carry_extender is
    Port (
           s_CE : in STD_LOGIC_VECTOR (2 downto 0);
           cout_CE : out STD_LOGIC
         );
end carry_extender;

architecture Behavioral of carry_extender is

begin

    process
    begin
        case (s_CE) is
            when "000" =>  -- PASS
                cout_CE <= '0';
            when "001" =>  -- AND
                cout_CE <= '0';
            when "010" =>  -- OR
               cout_CE <= '0';
            when "011" =>  -- NOT
                cout_CE <= '0';
            when "100" =>  -- ADDITION
                cout_CE <= '0';
            when "101" =>  -- SUBTRACTION
                cout_CE <= '1';
            when "110" =>  -- INCREMENT
                cout_CE <= '1';
            when "111" =>  -- DECREMENT
                cout_CE <= '0';
            when others =>  --changed this be carefulllllllllll
                cout_CE <= '0';
        end case;
    end process;
end Behavioral;


----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 13.03.2024 19:41:36
-- Design Name:
-- Module Name: Data_path - 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 leaf cells in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity Data_path_1 is
    Port ( IRload : in STD_LOGIC;
           PCload : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Clock : in STD_LOGIC;
           MUX_SEL: in STD_LOGIC;
           IR_OUT : out STD_LOGIC_VECTOR (9 downto 0));
end Data_path_1;

architecture Behavioral of Data_path_1 is

    signal ROM_out: std_logic_vector (9 downto 0);  -- output of ROM
    signal IR_9downto0: std_logic_vector (9 downto 0); --output of IR (last
3 bits)
    signal inc_out: std_logic_vector (3 downto 0); --output of INCREMENTER
    signal IR_mux_out: std_logic_vector (3 downto 0); --output of 2:1 MUX
    signal PC_out: std_logic_vector (3 downto 0); --output of PC

begin

    IR:  entity work.IR port map (
        D => ROM_out,
        IR_load => IRload,
        clk => Clock,
        IR => IR_9downto0
    );

    MUX_IR: entity work.mux_2_1 port map (
        IR => IR_9downto0(3 DOWNTO 0),
        inc => inc_out,
        sel => MUX_SEL,
        mux_out => IR_mux_out
    );

    PC: entity work.PC port map (
        D => IR_mux_out,
        PC_load => PCload,
        reset => Reset,
        clk => Clock,
        PC => PC_out
    );

    INCREMENTER: entity work.incrementer port map (
        inc_input => PC_out,
        inc_output => inc_out
    );

    ROM: entity work.ROM port map (
        address => PC_out,
        data_out => ROM_out
    );

   IR_OUT <= IR_9downto0;

 end Behavioral;


----------------------------------------------------------------------------
--------
-- Company:
-- Engineer:
--
-- Create Date: 13.03.2024 21:32:09
-- Design Name:
-- Module Name: Data_path_2 - 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 leaf cells in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity Data_path_2 is
    Port ( Instruction_Register : in STD_LOGIC_VECTOR (15 downto 0);
           Switch : in STD_LOGIC_VECTOR (15 downto 0);
           IE : in STD_LOGIC_VECTOR (1 downto 0);
           WE : in STD_LOGIC;
           WA : in STD_LOGIC_VECTOR (1 downto 0);
           RAE : in STD_LOGIC;
           RAA : in STD_LOGIC_VECTOR (1 downto 0);
           RBE : in STD_LOGIC;
           RBA : in STD_LOGIC_VECTOR (1 downto 0);
           Clock_1 : in STD_LOGIC;
           OP : in STD_LOGIC_VECTOR (2 downto 0);
           ZE : in STD_LOGIC;
           Z : out STD_LOGIC;
           OE : in STD_LOGIC;
           Data_Output : out STD_LOGIC_VECTOR (15 downto 0));
end Data_path_2;

architecture Behavioral of Data_path_2 is

    signal MUX_RF_OUT : std_logic_vector (15 downto 0); -- ouput of MUX_RF
    signal A_out_RF : std_logic_vector (15 downto 0); -- output of A_RF
    signal B_out_RF : std_logic_vector (15 downto 0); -- output of B_RF
    signal ALU_out : std_logic_vector (15 downto 0); -- output of ALU
    signal BUFF_OUT : std_logic_vector (15 downto 0); -- output of
TRI_STATE_BUFFER_OUTPUT

begin

    MUX_RF : entity work.mux_3_1 port map(
        x_instruction => Instruction_Register,
        y_instruction => Switch,
        ALU_output => ALU_out ,
        select_IE => IE,
        out_mux => MUX_RF_OUT
    );

    RF: entity work.RF port map (
        clk => Clock_1,
        WE_RF => WE,
        WA_RF => WA,
        RAE_RF => RAE,
        RAA_RF => RAA,
        RBE_RF => RBE,
        RBA_RF => RBA,
        input_ID => MUX_RF_OUT,
        Aout => A_out_RF,
        Bout => B_out_RF
    );

    ALU : entity work.ALU port map(
        a => A_out_RF,
        b => B_out_RF,
        op => OP,
        f => ALU_out
    );

    D_ff : entity work.d_ff port map (
        d => ALU_out,
        E => ZE,
        clk => Clock_1,
        Q => Z
    );

    TRI_STATE: entity work.tristate_buffer port map (
        input => ALU_out,
        cntrl => OE,
        output => BUFF_OUT
    );

    Data_Output <= BUFF_OUT;


    end Behavioral;
```

```vhdl
----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 06.03.2024 11:49:42
-- Design Name:
-- Module Name: full_adder - 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 leaf cells in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity full_adder is
    Port (
         x_FA : in STD_LOGIC;
         y_FA : in STD_LOGIC;
         carry_in_FA : in STD_LOGIC;
         sum_FA : out STD_LOGIC;
         carry_out_FA : out STD_LOGIC
        );
end full_adder;

architecture Behavioral of full_adder is

begin

    sum_FA <= x_FA xor y_FA xor carry_in_FA;
    carry_out_FA <= (x_FA and y_FA) or (carry_in_FA and (x_FA xor y_FA));

end Behavioral;


----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 13.03.2024 13:58:31
-- Design Name:
-- Module Name: incremeter - 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 leaf cells in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity incrementer is
    Port ( inc_input : in STD_LOGIC_VECTOR (3 downto 0);
           inc_output : out STD_LOGIC_VECTOR (3 downto 0));
end incrementer;

architecture Behavioral of incrementer is

    constant increment_value : std_logic_vector(3 downto 0) := "0001";
    constant c_in : std_logic := '0';
    signal cout_1 : std_logic_vector (3 downto 0);
    signal cout : std_logic ;

begin

      adder_increment: entity work.full_adder port map(
            x_FA => inc_input(0),
            y_FA => increment_value(0),
            carry_in_FA => c_in,
            sum_FA => inc_output(0),
            carry_out_FA => cout_1(0)
          );

   g_GENERATE_FOR: for i in 1 to 3 generate
            adder_increment_other : entity work.full_adder port map(
            x_FA => inc_input(i),
            y_FA => increment_value(i),
            carry_in_FA => cout_1(i-1),
            sum_FA => inc_output(i),
            carry_out_FA => cout_1(i)
          );
  end generate g_GENERATE_FOR;
  cout <= cout_1(3);

end Behavioral;


----------------------------------------------------------------------------
-------
-- Company: 
-- Engineer: 
-- 
-- Create Date: 13.03.2024 18:28:12
-- Design Name: 
-- Module Name: IR - 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 leaf cells in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity IR is
    Port ( D : in STD_LOGIC_VECTOR (9 downto 0);
           IR_load : in STD_LOGIC;
           clk : in STD_LOGIC;
           IR : out STD_LOGIC_VECTOR (9 downto 0));
end IR;

architecture Behavioral of IR is

begin

    process
    begin
        if rising_edge (clk) then
            if IR_load = '1' then
               IR <= D;
            end if;
        end if;
    
    end process;
end Behavioral;
```

```
----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 06.03.2024 11:37:39
-- Design Name:
-- Module Name: logic_extender - 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 leaf cells in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity logic_extender is
    Port ( s_LE : in STD_LOGIC_VECTOR (2 downto 0);
           a_LE : in STD_LOGIC;
           b_LE : in STD_LOGIC;
           x_LE : out STD_LOGIC
          );
end logic_extender;

architecture Behavioral of logic_extender is

begin

    process
    begin
        case (s_LE) is
            when "000" =>  -- PASS
                x_LE <= a_LE;
            when "001" =>  -- AND
                x_LE <= a_LE and b_LE;
            when "010" =>  -- OR
               x_LE <= a_LE or b_LE;
            when "011" =>  -- NOT
                x_LE <= not a_LE;
            when "100" =>  -- ADDITION
                x_LE <= a_LE;
            when "101" =>  -- SUBTRACTION
```

```
                x_LE <= a_LE;
            when "110" =>  -- INCREMENT
                x_LE <= a_LE;
            when "111" =>  -- DECREMENT
                x_LE <= a_LE;
            when others =>  --changed this be carefullllllllllll
                x_LE  <= '0';
        end case;
    end process;


end Behavioral;


----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 13.03.2024 14:59:21
-- Design Name:
-- Module Name: mux_2-1 - 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 leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux_2_1 is
    Port ( IR : in STD_LOGIC_VECTOR (3 downto 0);
           inc : in STD_LOGIC_VECTOR (3 downto 0);
           sel : in STD_LOGIC ;
           mux_out : out STD_LOGIC_VECTOR (3 downto 0)
         );
end mux_2_1;

architecture Behavioral of mux_2_1 is

begin
```

```
    process
    begin
        if sel = '0' then
            mux_out <= IR;
        else
            mux_out <= inc;
        end if;
    end process;

end Behavioral;


----------------------------------------------------------------------------
-------
-- Company:
-- Engineer: HARSHINI
--
-- Create Date: 13.03.2024 16:17:11
-- Design Name:
-- Module Name: mux_3_1 - Behavioral
-- Project Name: Assignment 2 - CPU DESIGN
-- Target Devices:
-- Tool Versions:
-- Description: This is a program for 4:1 mux but its mentioned as 3:1
because of its last 2 inputs
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
-------


library IEEE; -- 'IEEE'for standard logic operations and UNISIM for
components provided by Xilinx for simulation purposes.
use IEEE.STD_LOGIC_1164.ALL; --defines the basic data types for
representing digital signals (std_logic, std_logic_vector) and provides
operators for working with these types.

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL; -- provides numeric types such as signed and
unsigned, as well as functions and operators for arithmetic operations on
these types.

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
library UNISIM; --allows to use simulation models during simulation. These
models are essential for accurately simulating the behavior of design,
especially if it includes Xilinx-specific primitives
use UNISIM.VComponents.all; --library available for use in design without
needing to explicitly specify each component. This simplifies your code and
makes it easier to include Xilinx-specific primitives in your design.

entity mux_3_1 is --" mux_3_1" entity declares x_instruction,
y_instruction, z_instruction, select_IE, out_mux
    Port ( x_instruction : in STD_LOGIC_VECTOR (15 downto 0); --from
instruction
```

```
           y_instruction : in STD_LOGIC_VECTOR (15 downto 0); --from switch
           ALU_output : in STD_LOGIC_VECTOR (15 downto 0); --from ALU
output
           select_IE : in std_logic_vector (1 downto 0); -- select pin
           out_mux : out STD_LOGIC_VECTOR (15 downto 0)); -- output
end mux_3_1;

architecture Behavioral of mux_3_1 is -- " Behavioral architecture" defines
the behavior of the mux_3_1 entity.

begin

    process (select_IE) --sensitive to the changes in select pin
    begin
        if select_IE = "00" then -- if select pin is "00", intput is from
instruction
            out_mux <= x_instruction; -- represents input from the
instruction itself
        elsif select_IE = "01" then -- if select pin is "01", intput is
from switches
            out_mux <= y_instruction; -- output is the value of switches
        elsif select_IE = "10" then -- if select pin is "10", intput is
from ALU output
            out_mux <= ALU_output; -- output is the value of ALU output
        elsif select_IE = "11" then -- if select pin is "10", intput is
from ALU output
            out_mux <= ALU_output; -- output is the value of ALU output

        end if;
    end process;  -- end process
end Behavioral; -- terminate architecture


----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 05.02.2024 20:16:16
-- Design Name:
-- Module Name: one_digit - 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;
library UNISIM;
use UNISIM.VComponents.all;

entity one_digit is
    Port ( digit : in STD_LOGIC_VECTOR (3 downto 0);
           seg : out STD_LOGIC_VECTOR (6 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0));
end one_digit;

architecture Behavioral of one_digit is


begin

    an <= "1110";
    with digit select

           seg <=  "1000000" when "0000", --0
                   "1111001" when "0001", --1
                   "0100100" when "0010", --2
                   "0110000" when "0011", --3
                   "0011001" when "0100", --4
                   "0010010" when "0101", --5
                   "0000010" when "0110", --6
                   "1111000" when "0111", --7
                   "0000000" when "1000", --8
                   "0010000" when "1001", --9
                   "1000000" when others; --0 when other buttons are
pressed

end Behavioral;


----------------------------------------------------------------------------
-------
-- Company:
-- Engineer: HARSHINI
--
-- Create Date: 13.03.2024 17:07:12
-- Design Name:
-- Module Name: RF - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description: Program for register files
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
-------


library IEEE; -- 'IEEE'for standard logic operations and UNISIM for
components provided by Xilinx for simulation purposes.
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 leaf cells in this code.
```

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

entity RF is
    Port ( clk : in STD_LOGIC;
           WE_RF : in STD_LOGIC;
           WA_RF : in STD_LOGIC_VECTOR (1 downto 0);
           RAE_RF : in STD_LOGIC;
           RAA_RF : in STD_LOGIC_VECTOR (1 downto 0);
           RBE_RF : in STD_LOGIC;
           RBA_RF : in STD_LOGIC_VECTOR (1 downto 0);
           input_ID : in STD_LOGIC_VECTOR (15 downto 0);
           Aout : out STD_LOGIC_VECTOR (15 downto 0);
           Bout : out STD_LOGIC_VECTOR (15 downto 0));
end RF;

architecture Behavioral of RF is

    subtype reg is std_logic_vector (15 downto 0);
    type regArray is array (0 to 3) of reg;
    signal reg_file: regArray;

begin

    WritePort: process
    begin

        wait until rising_edge(clk);
        if WE_RF = '1' then
            reg_file (to_integer(unsigned(WA_RF))) <= input_ID;
        end  if;
   end process;

   -- READ PORT A
   Aout <= reg_file (to_integer(unsigned(RAA_RF))) when RAE_RF = '1'
   else (others => '0');

   -- READ PORT b
   Bout <= reg_file (to_integer(unsigned(RBA_RF))) when RBE_RF = '1'
   else (others => '0');

end Behavioral;


----------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date: 13.03.2024 12:34:28
-- Design Name:
-- Module Name: Read_Only_Memory - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
```

```vhdl
-- 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 leaf cells in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity ROM is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data_out : out STD_LOGIC_VECTOR (9 downto 0));
end ROM;

architecture Behavioral of ROM is

    type program is array (0 to 15) of std_logic_vector(9 downto 0); --16
locations
    signal first_code : program := (
        "0010000011", -- IN Rdd //0// READ INPUT TO R3
        "1111000001", -- MOV Rdd, #nnnn //1// INIT R0 = 1
        "1111010000", -- MOV Rdd, #nnnn //2// INIT R1 = 0
        "1011010001", -- ADD Rdd, Rrr, Rqq //3// R1 = R0 + R1
        "1001000001", -- INC Rrr, #nnnn //4// R0 = R0 + 1
        "1000000011", -- LT Rrr, Rqq //5// IF R0 < R3 THEN Z = 0 ELSE Z = 1
        "0110000011", -- JNZ aaaa //6// IF Z == 0  THEN GO ADDR 03 ELSE GO
NEXT ADDR
        "0011000001", -- OUT Rss //7// OUTPUT R1
        others => ("0000000000") -- HALT //8// OVER
       );


        signal current_program : program := first_code; ---assign address

begin

    data_out <= current_program (to_integer(unsigned(address)));  --adress
is converted to unsigned int., and coverted to regular integer

end Behavioral;
```