UPPSALA
UNIVERSITET

Digital Electronics Design with VHDL
8bit Micro-processor Project Report

Jatin Anand

Abstract

A microprocessor is a programmable electronics chip that has computing and decision making capabilities similar to central processing unit of a computer. Any microprocessor based systems having limited number of resources are called microcomputers. Nowadays, microprocessor can be seen in almost all types of electronics devices like mobile phones, printers, washing machines etc. Microprocessors are also used in advanced applications like radars, satellites and flights. Due to the rapid advancements in electronic industry and large scale integration of devices results in a significant cost reduction and increase application of microprocessors and their derivatives.[5]

The project is to design an 8 bit microporcessor and implement it on a development board DE1-SoC. There are several components in this design that follow a set of instructions and perform a given program. The main components of the 8bit microprocessor are the Control unit, the memory, Arithematic Logic Unit (ALU). There are two data and two address registers. It also has an Instruction finder to find the given instruction and send it to the Instruction decoder. A program counter increments at every execution cycle. It has a Multiplexer to select addresses from different address registers.

The control unit is designed in such a way that it fetches for instructions from the memory in one clock cycle and executes those instructions in the next clock cycle. The fetched instruction is sent back to the control unit from the Instruction decoder to send outputs for execution. This makes the control unit the most important component of the microprocessor.

Contents

1 Introduction

An 8 bit computer in which the information is grouped in 8 bit data packages, is the base of all 16 bit, 32 bit or 64 bit computers. It is the amount of bits a processor can process in a system. 8 bit processors are commonly used in simple computational tasks in systems like home appliances and insductrial specific systems.
In this project, each unit of the microprocessor is programmed in VHDL which is compiled and simulated in Quartus. Next step is to download it on the FPGA (Intel- Cyclone V- DE1-SoC).
The processors has an instruction set stored in the memory to perform certain tasks.The working of the processor requires some functional units to perform its tasks. A global clock is used to synchronise the tasks in the system. There is a control unit which is the main unit of the system. The processor starts with a program counter starting from zero and incrementing every other clock cycle.
Control unit is designed as a state machine with two states, in this processor. FETCH and EXECUTE. The working of each states is explained in the next section. Control unit starts from enabling the instruction finder to fetch an instruction from the memory and tranfer it to the instruction decoder for decoding. Decoder sends a decoded message to the control unit for execution of the task.

By the end of this project, I was left with huge understanding of FPGA programming and implementation of gate level arrays on a device. A more detailed schematic is shown in Implementation section. Further, the program will be downloaded on the hardware and will be tested with buttons for reset input and the Hex display to display the register values.

The following **Figure 1** depicts connections of all the components used in this 8 bit microprocessor. All the components get an Enable input from the control unit which is connected to a global clock and a reset input. The components found in the figure are described in details in later sections.

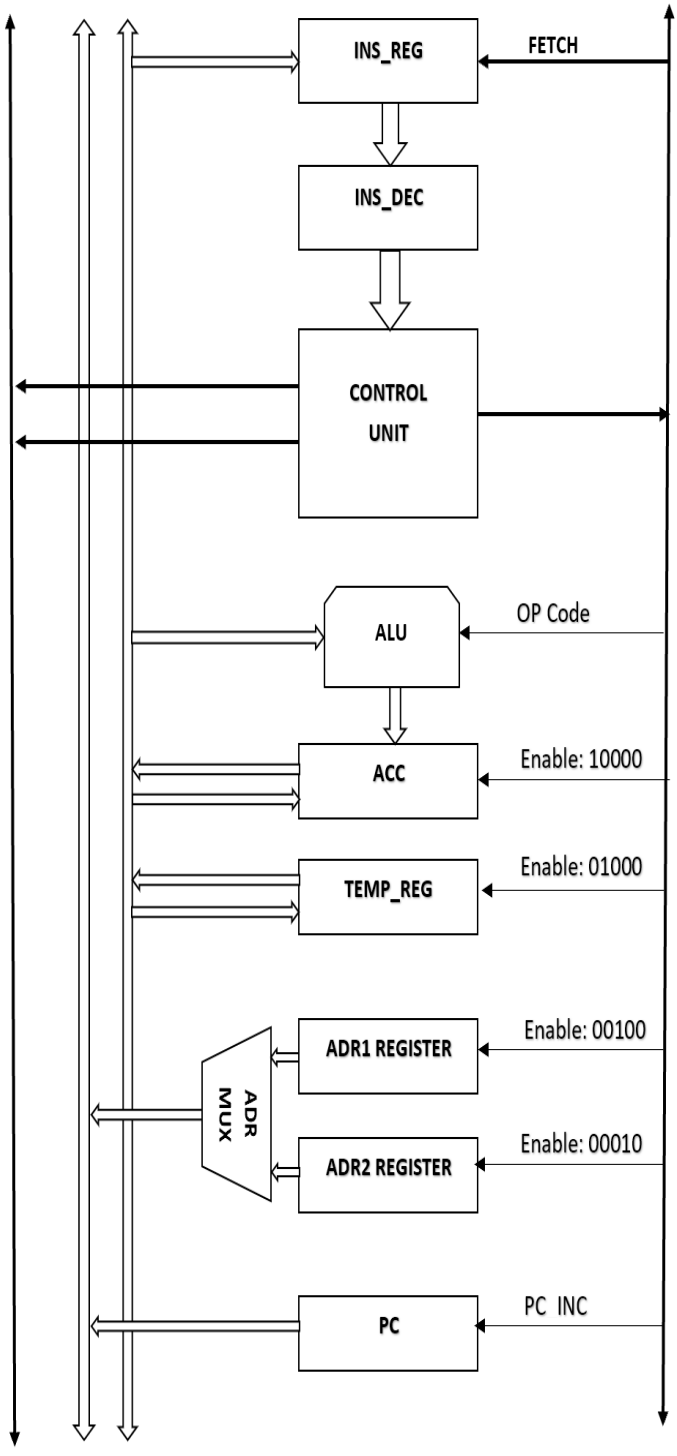

Figure 1: Block diagram of all units

2 Project Description

2.1 Tools and Software

Softwares like Quartus and ModelSim are used to compile and simulate the VHDL program code of the 8 bit micro-processor.
The project task is to design a system that can run a set of instructions and implement A=B+C program and if A>=0 then B=C. The program code should be downloaded to a DE1-SoC device with inputs from its buttons and outputs on a Hexadecimal display mounted on the device.

2.2 Design

The microprocessor is designed in such a way that control unit is, in someway, connected to inputs and outputs ports of all other components.
The design has other smaller components that are programmed to perform certain functions. Description of each component with their entity name in brackets, are mentioned below:

- Control Unit(control_unit): Its the main component of the microprocessor that drives all the processes and other components. Its designed as a state machine with two state for fetching the instruction and executing the instructions in each state. The control unit is responsible for sending and receiving all the information required to perform a task. It runs with a global clock that makes it switch through its states and process certain tasks in each state.
- Memory (RAM): Memory of the processor contains the instruction set to perform tasks and other memory spaces for reading and storing data.
- Arithematic Logical Unit (ALU): Its function is to perform arihtematic and logical operations on operands. It is programmed to perform add, subtract, AND,OR operations.
- Accumulator (ACC): This register is used to store operands and results of operations.
- Temporary register (TEMP): This register is used to store operands.
- Address register 1 (ADR1): This register is used to store an address (memory location) of operands.
- Address register 2 (ADR2): This register is used to store an address (memory location) of operands.
- Program Counter (PC): Its function is to store the address of the current instruction which starts from address zero and send the value to control unit for further processing.
- Instruction Finder (INS_REG): Its function is to receive the instruction from memory and send it to the Instruction Decoder for decoding the instruction.

- Instruction Decoder (INS_DEC): Its function is to decode the received instruction and inform the control unit for further execution of the task.
- Address Selector (ADR_MUX): This is mux that is used to receive values from address register 1 and address register 2 and send them to the control unit according to the enabe value.

3 Theory

FPGA as a design tool is very reliable and fast to implement. It is a great platform for testing designs as it is easily reprogrammable. Further these designs can be mass produced as ASIC (Application Specific Integrated Circuit) chips and released to the market. Thus FPGA enables rapid transition from lab designing to market implementation.[1]

The different types of computers are classified according to the amount of information they process, the type of operations they execute, or their architecture. The information is grouped into 8-bit data packages called bytes. A pair of bytes is called a word (16-bit), a pair of words is called a double word (32-bit), and four words are called a quad word (64-bit). This is how computers are classified according to the amount of information they can process (i.e. 8, 16, 32, 64-bit)[4].

All computers perform data transfer instructions in order to interchange data among the different memories and peripherals. There are integer arithmetic processors, floating point processors, digital signal processors, and application specific processors. Integer arithmetic processors are best suited for general-purpose applications. Thus, it is possible to find byte, word, dword, and qword sized general-purpose processors.
Beside the previous classifications, there are others based on the type of instructions computers execute and the number of instructions they are able to execute. Based on the instruction types, computers are classified into three kinds: the Reduced Instruction Set Computer (RISC), the Complex Instruction Set Computer (CISC), and the Specific instruction Set Computer (SISC). General purpose processors are in the RISC or CISC[4].

I have designed an 8 bit Micro-processor using just basic functional units programmed in VHDL and implemented it on Intel FPGA Board. I have used certain instructions to perform a program with an 8 bit RAM, 4 Registers, a Control Unit and an 8 bit ALU along with a counter and address Multiplexer. The design was implemented in Quartus software and simulated on Modelsim.

4 Implementation

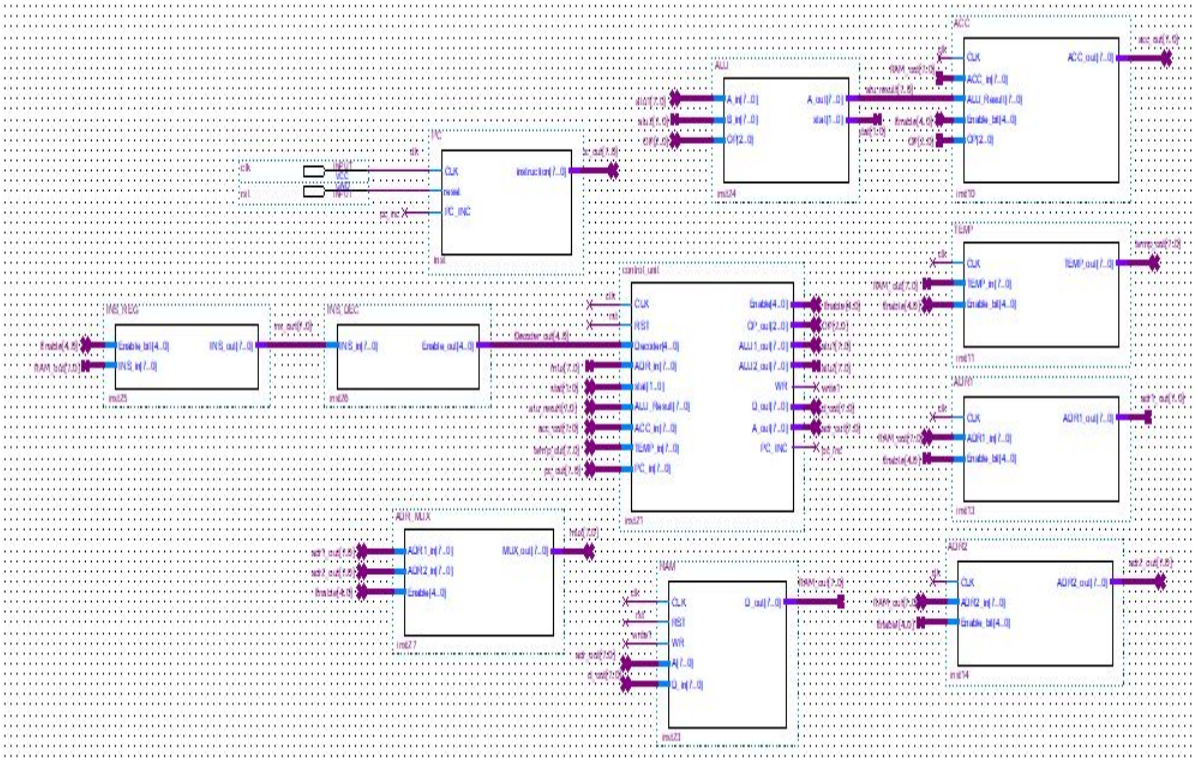

Figure 2: Schematic of all the units

A clearer picture of each component with signals names is described in the following subsections. **Figure 2** contains an overview of all the components with the control unit in the center, registers on the right and other components on the left of control unit.
These components are connected to the control unit which is connected to a global clock input. Each component symbol has some inputs and output ports that drives all the information required for the processor to perform a task.

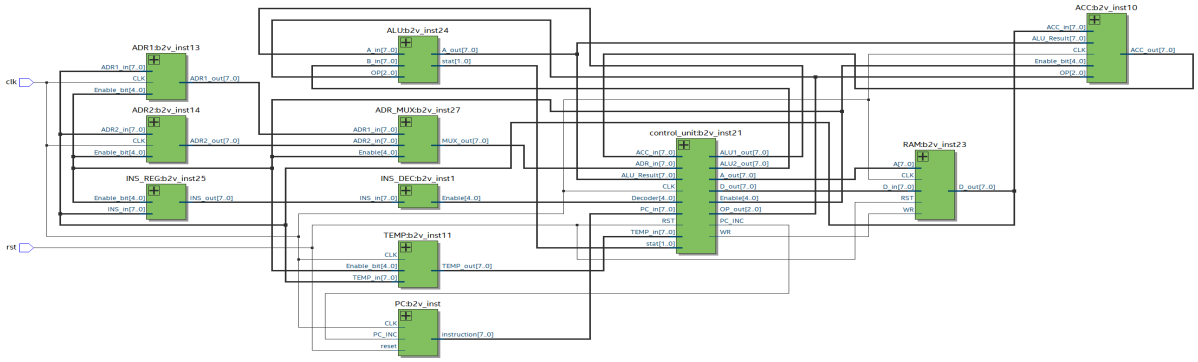

Figure 3: RTL view of all components

4.1 Control Unit

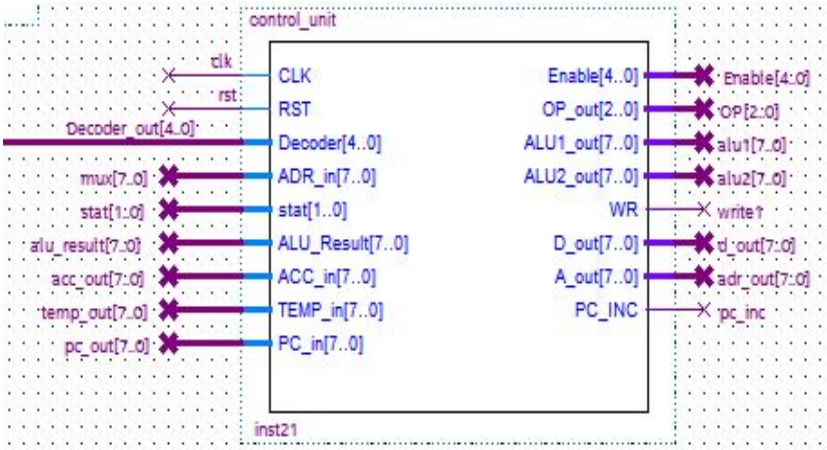

Figure 4: Control Unit Component

Figure 4 is the symbol for the control unit with nine input ports and eight output ports. As the control unit is very important for working of this processor, it is designed as a state machine which jumps between two states, one for fetching the instruction from the memory and the other for implementing the instruction in the micro-processor. Fetch state gets the instruction from the memory, and sends it to the control unit through instruction register and decoder. Execute state sets enables for all the components accroding to the given instruction.

Input Ports :

- CLK: Global clock input
- RST: Reset for control unit
- Decoder: A 5 bit input from the decoder that contains information for sending the required output signals. First four bits are register enables and the last bit is for storing data to memory.

0	1	2	3	4
ACC_en	TEMP_en	ADR1_en	ADR2_en	Store

Figure 5: Enable_in

- ADR_in: An 8 bit input from address MUX which selects input address from Address registers.
- stat: A 2 bit status input from ALU in which the first bit is an overflow status and second bit is a non-zero status. Overflow turns HIGH when there is an overflow and non-zero turns HIGH when the ALU_Result value is zero.

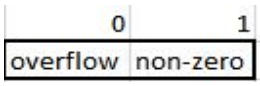

Figure 6: stat

- ALU_Result: 8 bit result from ALU
- ACC_in: Accumulator's last value is received by this port to process further.
- TEMP_in: Temp register's last value is received by this port to process further.
- PC_in: Inputs the signal from program counter to inform the instruction address to the memory.

Output Ports:

- Enable: A 5 bit enable output similar to input from Decoder that has enable bits for all the registers and for addition in ALU and storing data into the memory.
- OP_out: OP code output to ALU for selecting its operations.
- ALU1_out: One of ALU's inputs for an operation.
- ALU2_out: One of ALU's inputs for an operation.
- WR: Write enable sent to the memory.
- D_out: This output signal contains the input data for writing into the memory when WR is enabled.
- A_out: This output signal contains the address to a memory location.
- PC_INC: This signal is HIGH at every EXECUTE cycle which increments the program counter to get the next instruction from the memory.

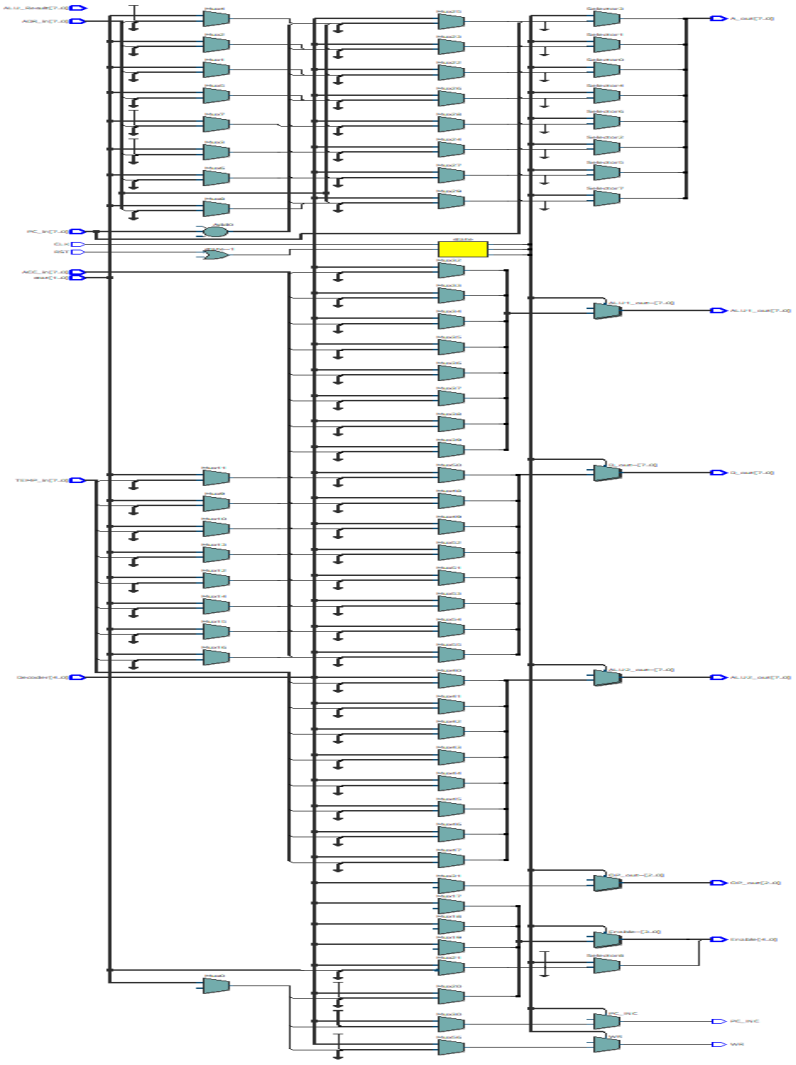

Figure 7: RTL view of control_unit

4.2 Memory

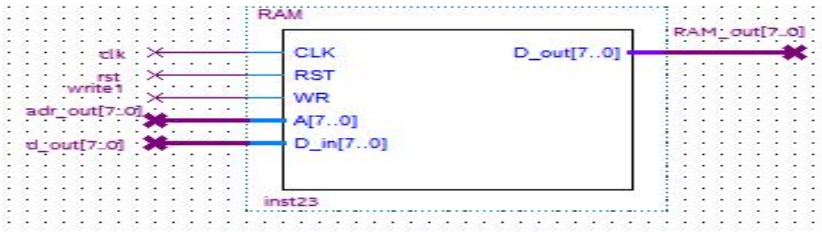

Figure 8: Memory Component

Figure 7 is the symbol for a memory with five input ports (CLK,RST, WR, A and D_in) and one 8 bit output port (D_out). When RST is LOW and WR

is HIGH at rising edge of the CLK, the memory stores the data from D_in at address A in the memory,synchronously. Data is always read when address A is inputed and sent to output port D_out.
Memory has a capacity of 256 locations of 8 bit each which are divided into two sections.
The first section is to store the intruction set that is to be read by the instruction register. The second section has memory spaces from address 100 onwards that is used to store additional data, in this processor.

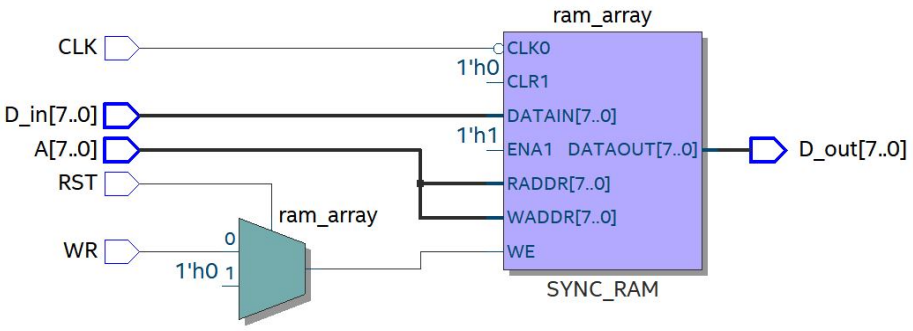

Figure 9: RTL view of RAM

4.3 Arithematic Logic Unit

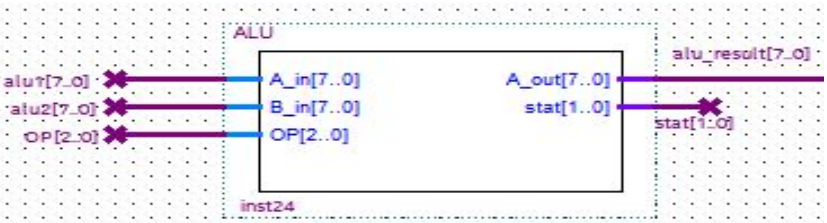

Figure 10: Arithematic Logic Unit Component

Figure 9 is the symbol for Arithematic Logic Unit with three input ports (A_in, B_in and OP) and two output ports (A_out and stat). According to OP signal, the ALU performs a specified operation within the inputs from other input ports. The result of those operations are sent through A_out. stat signal indicates overflow and if the ALU result is a non-zero value.
OP code is a 3 bit input that enables ALU to perform four operations of addition, subtraction, AND and OR, depending on OP code.

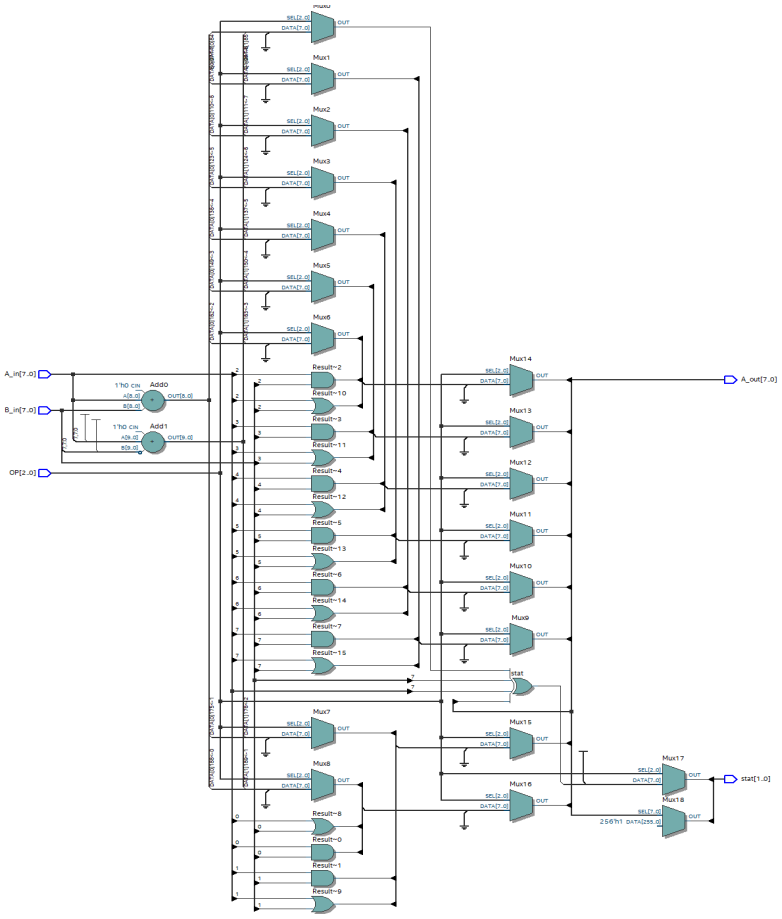

Figure 11: RTL view of ALU

‘

4.4 Program Counter

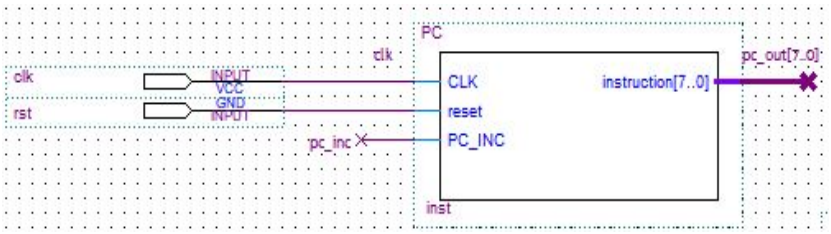

Figure 12: PC component

Figure 11 is the symbol for program counter with three input ports (CLK, reset and PC_INC) and one 8 bit output port (instruction). When PC_INC is enabled, it increments the counter by 1 at risign edge of CLK.

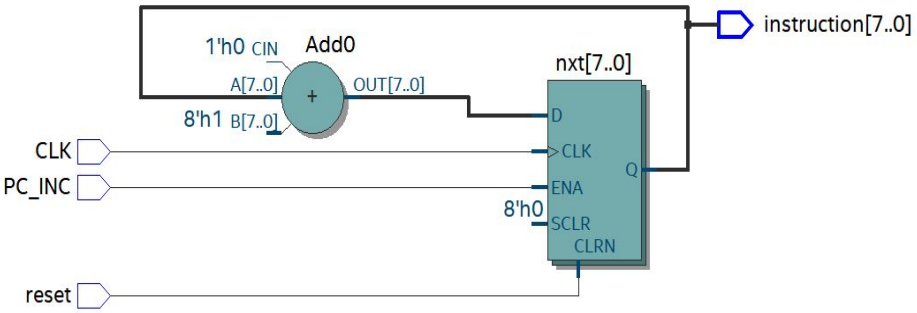

Figure 13: RTL view of PC

4.5 Instruction Finder

Figure 14: Instruction finder Component

Figure 13 is the symbol for instruction finder with two input ports (Enable_bit and INS_in) and one 8 bit output port (INS_out). When the Enable_bit activates instruction finder, INS_in takes in the instruction from memory and outputs it to the decoder.

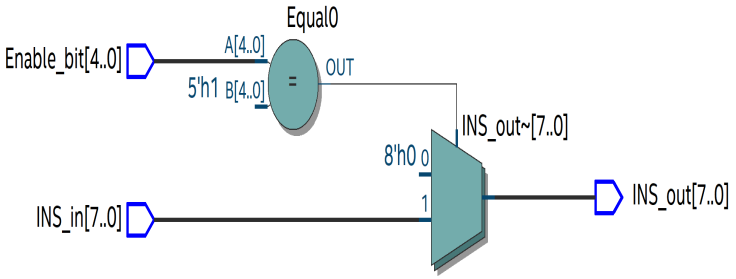

Figure 15: RTL view of INS_REG

4.6 Instruction Decoder

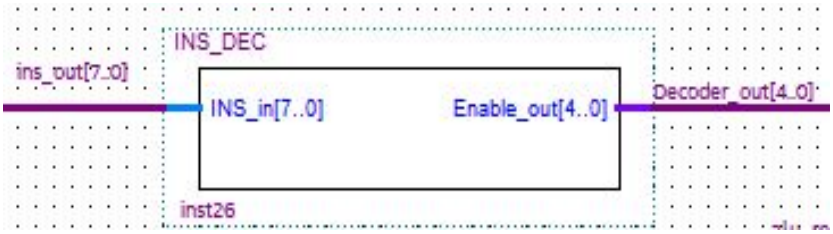

Figure 16: Instruction decoder Component

Figure 15 is the symbol for instruction decoder with one input port (INS_in) and one output port (Enable_out). INS_in is the instruction data that gets decoded in this register and outputs a 5 bit Enable_out that contains enable information for all the other componets. This will be used by control unit to activate the required register according to the instruction given.

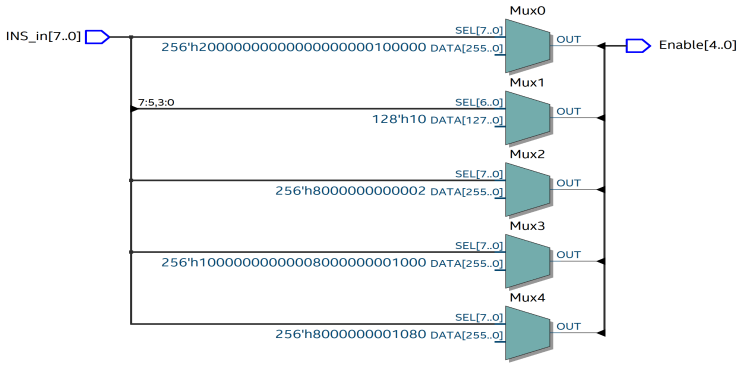

Figure 17: RTL view of INS_DEC

4.7 Accumulator

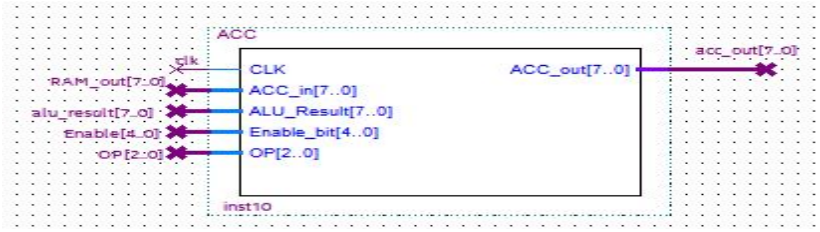

Figure 18: Accumulator Component

Figure 17 is the symbol for Accumulator register with five input ports (CLK, 8 bit ACC_in, 8 bit ALU_Result, Enable_bit and OP) and one 8 bit output port (ACC_out). At every rising edge of CLK, Enable_bit activates the accumulator and depending on OP input and Enable_bit, one of the two 8 bit inputs are received and stored in the accumulator.

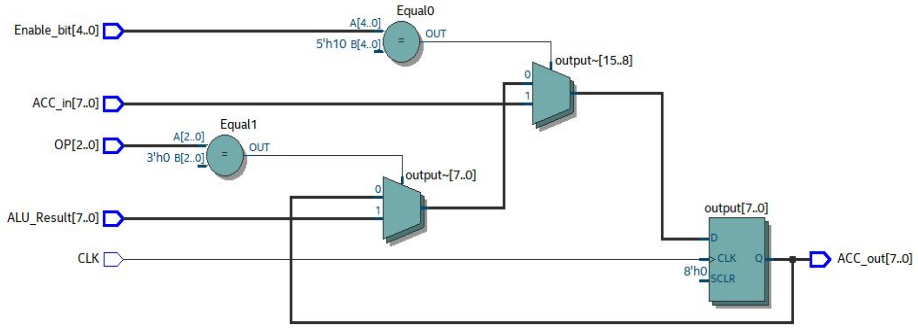

Figure 19: RTL view of ACC

4.8 Temporary Register

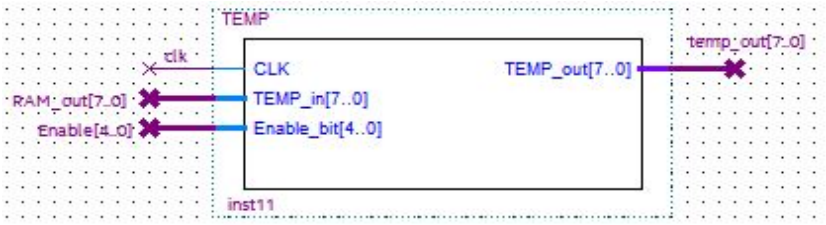

Figure 20: Temporary Register Component

Figure 19 is the symbol for a Temporary register with three input ports (CLK, TEMP_in and Enable_bit) and one 8 bit output port (TEMP_out). At every rising edge of CLK, when Enable_bit activates the register, TEMP_in is stored in the register and outputs at TEMP_out.

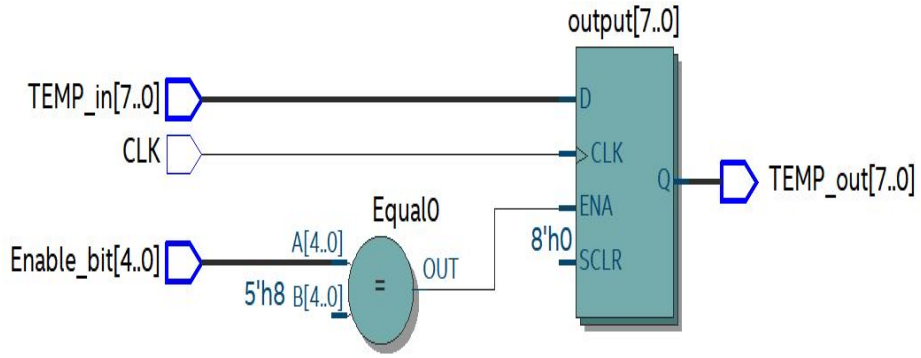

Figure 21: RTL view of TEMP

4.9 Address Register 1

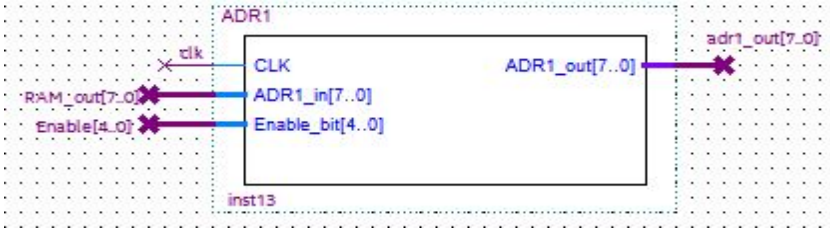

Figure 22: Address Register 1 Component

Figure 21 is the symbol for a Address register with three input ports (CLK, ADR1_in and Enable_bit) and one 8 bit output port (ADR1_out). At every rising edge of CLK, when Enable_bit activates the register. ADR1_in is stored in the register and outputs at ADR1_out.

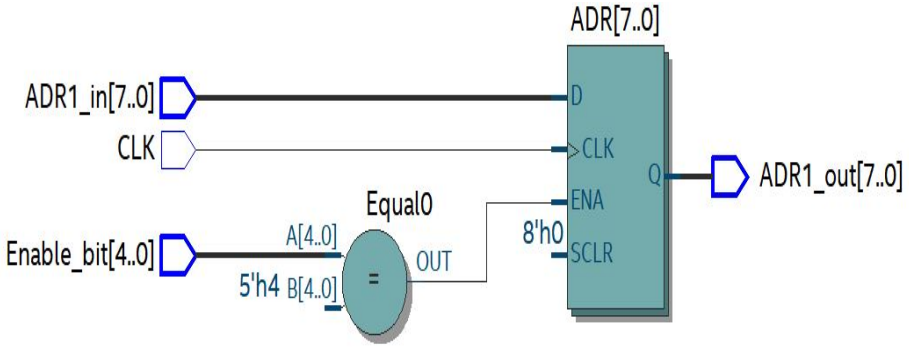

Figure 23: RTL view of ADR1

4.10 Address Register 2

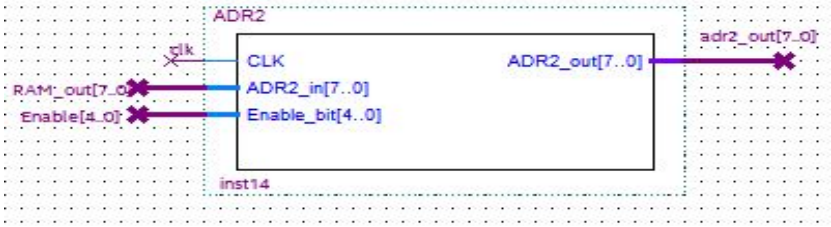

Figure 24: Address Register 2 Component

Figure 23 is the symbol for a Address register with three input ports (CLK, ADR2_in and Enable_bit) and one 8 bit output port (ADR2_out). At every rising edge of CLK, when Enable_bit activates the register. ADR2_in is stored in the register and outputs at ADR2_out.

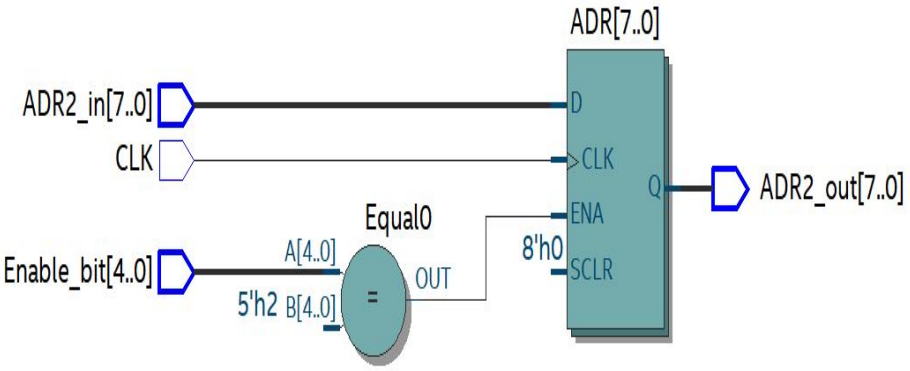

Figure 25: RTL view of ADR2

4.11 Address MUX

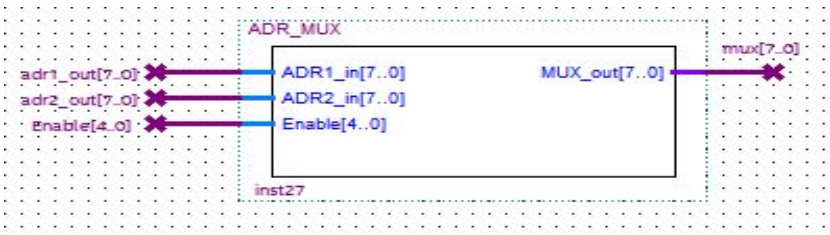

Figure 26: Address MUX Component

Figure 25 is the symbol for a Address MUX with three input ports (ADR1_in, ADR2_in and Enable) and one 8 bit output port (MUX_out). MUX_en selects between the input ports data and outputs at MUX_out.

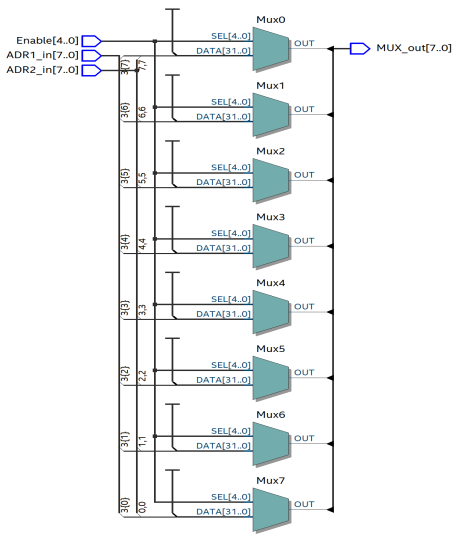

Figure 27: RTL view of ADR_MUX

5 Tests and Results

All the components were connected as shown in **Figure 2** with certain signals used to transfer data within the components.

This design was simulated in Modelsim and signals were observed. The instruction set that is saved in the memory does the following:

- LOAD Address Register 1

  This instruction is to load the value from memory of address PC+1 and store in the Address register 1.

- LOAD Address Register 2

  This instruction is to load the value from memory of address PC+1 and store in the Address register 2.

- LOAD Accumulator

  This instruction is to load the value from memory and store in the Accumulator. The address of memory location is taken from Addres register 1.

- LOAD Temporary register

  This instruction is to load the value from memory and store in the Temporary register. The address of memory location is taken from Addres register 2.

- ADD Accumulator and Temporary register

  This instruction is to add the data in Accumulator and Temporary register and store the result in Accumulator. Addition takes place with OP code ”000”.

- Store the result from Accumulator to the memory

  This instruction stores the data of Accumulator in the memory. The memory location to store the value is taken from Address register 1.

- Checks if ALU result is positive and takes the data from memory location of Address Register 2 and stores it in memory location of Address Register 1.

  This instruction checks if the result from ALU is positive. Ie. if *stat* is disabled then data in temporary register gets stored in memory location of address register 1.

- LOAD Temporary register with the new data from memory location of address register 1.

This instruction is to load data from memory location of address register 1 and store in Temporary register.

- LOAD Accumulator again and perform ADD operation again and load result in ACC. The next four instructions are repeated and new data is loaded in accumulator then addition is performed with the new result and it is loaded in accumulator again.

These instructions are performed by the processor with program counter pointing at them in the memory at alternate clock cycles (In FETCH state)

5.1 Simulations

The requirement for this project is to implement a program that adds two values from the registers and stores it in a memory location. This is simulated using the instructions mentioned in the previous section.

A figure showing the simulation is given below.

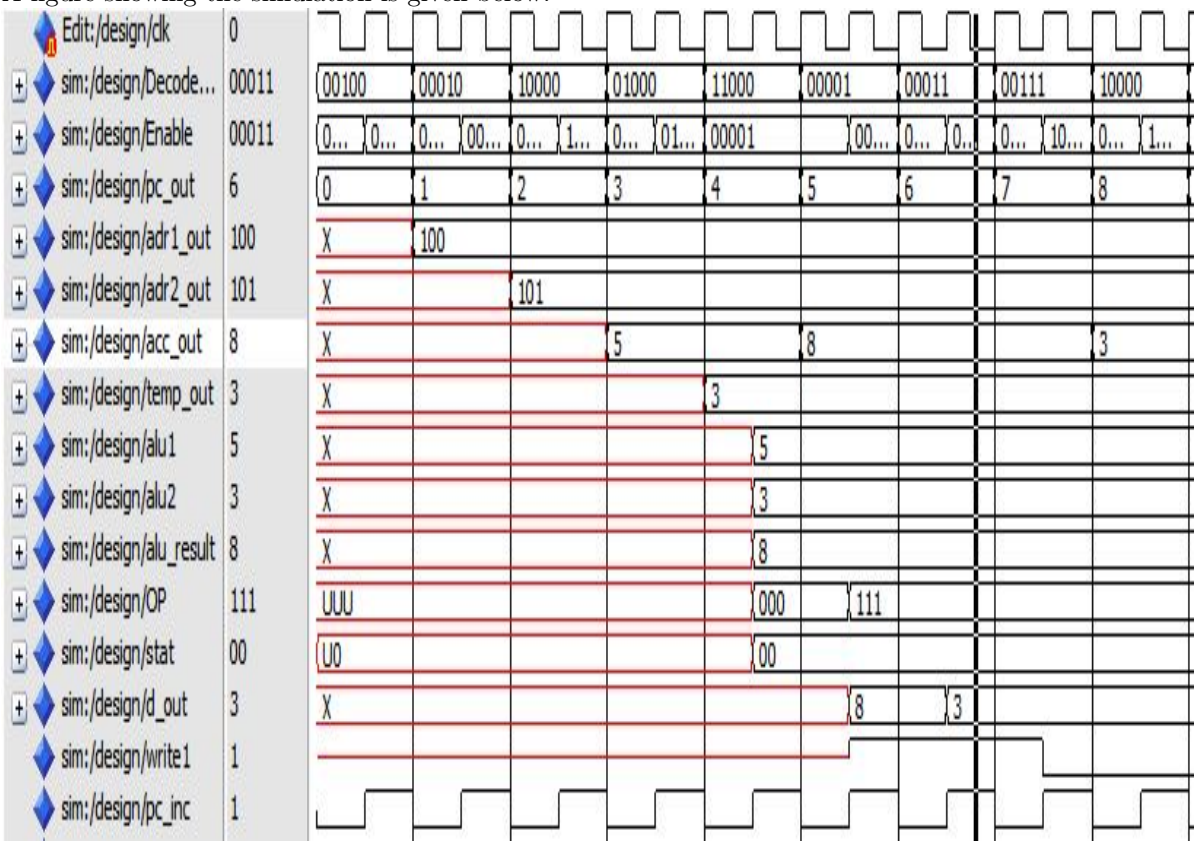

Figure 28: Simulation of all the units

The above figure shows the signals that are being used to perform the instructions.

In the first FETCH cycle, instruction is loaded from the memory to the control unit. In the EXECUTE cycle, the instuction is executed to load the address from memory to adr1_out in Address register 1.
Similary, next FETCH cycles loads data to adr2_out. Further instructions store data in acc_out and then to temp_out.
When OP is "000". alu1 and alu2 sends 5 and 3 to the ALU for addition and the result is stored in acc_out. stat is input to the control unit. After this instruction, write1 is enabled and data from accumulator is stored in memory location of adr1_out in the memory.
These instructions also check if ALU result is positive by checking the *stat* value which has the overflow flag then make B=C ie.

$$MEM(ADR1) <= MEM(ADR2)$$

Other signals like stat (overflow and non zero flag), d_out(input to memory for writing), pc_inc for incrementing the program counter, RAM_out(output from memory) and ins_out showing the instruction fetched, are used to perform the instruction tasks.

The following steps show what happens in the simulation above.

Data(acc_out): 5 from Address(adr1_out): 100 stored in Accumulator
Data(temp_out): 3 from Address(adr2_out): 101 stored in Temporary register
Addition
ALU_result: 8 and stat:00 which means the result is positive.
Address: 100 gets New Data: 3 from Address: 101

Here, you can notice that the new data in accumulator(acc_out) gets stored from address 101. The cycle of adding and storing and checking if result is postive, repeats again.

The next simulation is with the a negative value in accumulator to activate the overflow loop. When ALU_Result is negative, the instruction register sends control to an address with no intstruction stored in it.

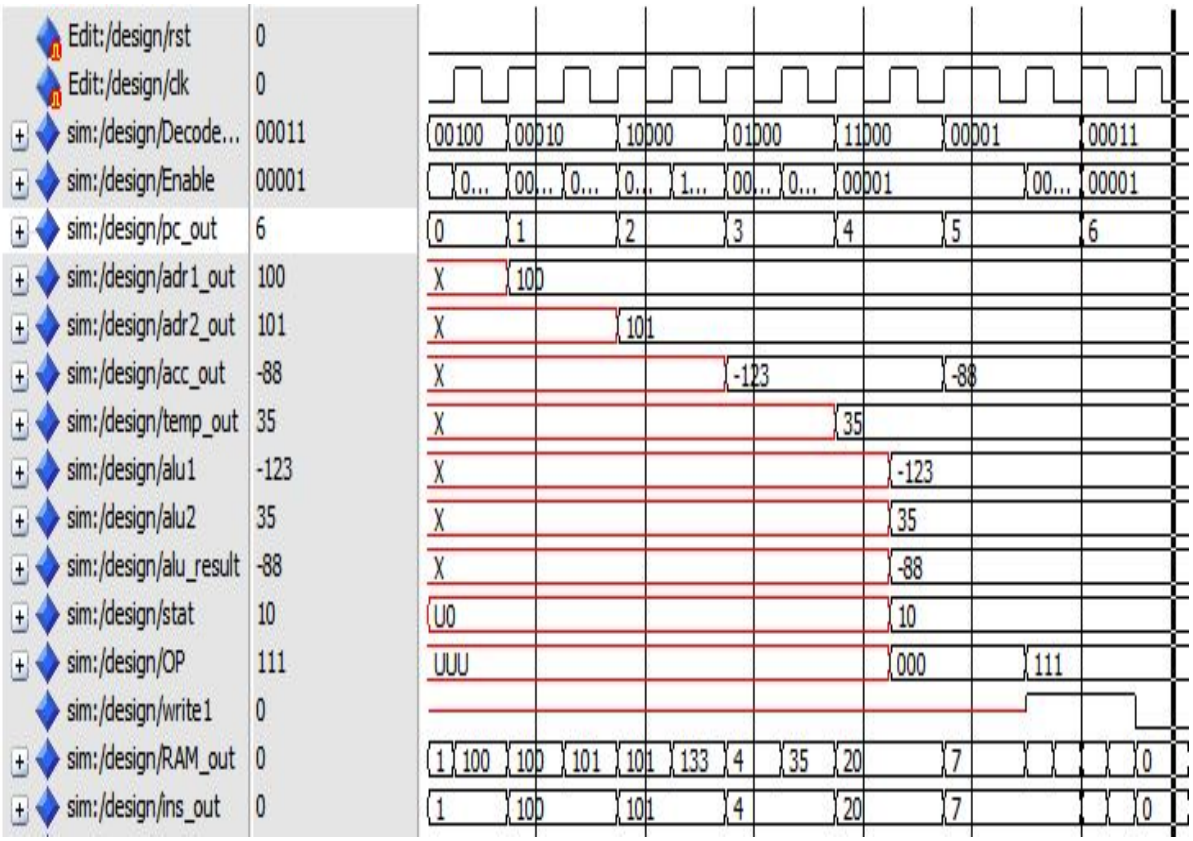

Figure 29: Simulation with negative data

6 Conclusions and evaluation

The project was designed with several components and the instructions were performed successfully. This 8 bit microprocessor will then be downloaded to a DE1-SoC device and tested using input buttons and a Hexadecimal display available on the device to show data from accumulator and other registers.

By evaluating my design, I came across different output when I had designed the registers with clock and without clock. In testing phase, simulations were observed with and without clock inputs to some components. There is one global clock driving the control unit and other registers. The simulation presented in the previous section has a clock assigned in data and address registers, program counter and the control unit.

7 Appendix A. Source Code

Listing 1: Control unit

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity control_unit is
6
7      PORT(
8          CLK          :   in      std_logic;
9          RST          :   in  std_logic;
10         Decoder      :   in      std_logic_vector(4 downto 0);-- from decoder
11         ADR_in       :   in      std_logic_vector(7 downto 0);--from ADR_MUX
12         stat             :   in  std_logic_vector(1 downto 0);--from ALU
               status
13         ALU_Result   :   in      std_logic_vector(7 downto 0);-- from
               ALU_Result
14         ACC_in       :   in      std_logic_vector(7 downto 0);--from ACC
               output
15         TEMP_in      :   in      std_logic_vector(7 downto 0);--from TEMP
               output
16         PC_in            :   in      std_logic_vector(7 downto 0);--from PC
               output
17
18         Enable       :   out std_logic_vector(4 downto 0);-- enables for all
               registers
19         OP_out       :   out std_logic_vector(2 downto 0);--OP code
20         ALU1_out         :   out std_logic_vector(7 downto 0);--ALU input A
21         ALU2_out         :   out std_logic_vector(7 downto 0);-- ALU input B
22         WR               :   out std_logic;
23         D_out            :   out std_logic_vector(7 downto 0);-- To RAM data
               in for writing
24         A_out            :   out std_logic_vector(7 downto 0);-- to RAM
               Address
25         PC_INC       :   out std_logic -- for incrementing PC
26         );
27 end control_unit;
28
29 ARCHITECTURE logic OF control_unit is
30
31     type state_type is (idle,FETCH,EXECUTE);    -- states
32     signal state : state_type;
33
34 begin
35 first : process(CLK,RST,state)
36     begin
37         if RST='0' then
38                 state<=idle;
39         elsif rising_edge(CLK) then
40             case state is
41                 when idle => state<= FETCH;
42                 when FETCH   => state <= EXECUTE;
43                 when EXECUTE => state <= FETCH;
44                 when others => state <= idle;
45             end case;
46         end if;
47 end process;
48
49 second : process(state,Decoder,stat,ADR_in,PC_in,TEMP_in,ACC_in)
50 begin
51         Enable <="00000";
52         OP_out <="111";
53         ALU1_out     <="00000000";
54         ALU2_out     <="00000000";
55         WR  <= '0';
56         D_out    <="00000000";
57         A_out    <="00000000";
58         PC_INC <='0';
59     case state is
60         when idle =>
61             PC_INC <= '0';
```

```
62          when FETCH =>
63              A_out<=PC_in;  -- address to RAM
64              Enable<="00001";    -- enables INS_REG
65              PC_INC<='0';
66          when EXECUTE =>
67              case Decoder is
68                  when "00100" =>
69                      Enable<="00100";    -- enables ADR1 and disable INS_REG
70                      A_out <= PC_in + '1';   --address from the next
                            instruction in memory
71                      PC_INC<='1';
72                  when "00010" =>
73                      Enable<="00010";    -- Enables ADR2
74                      A_out <= PC_in + '1';
75                      PC_INC<='1';
76                  when "10000" =>
77                      Enable<="10000";    -- enables ACC
78                      A_out <= ADR_in;
79                      PC_INC<='1';
80                  when "01000" =>
81                      Enable<="01000";    -- enables TEMP_REG
82                      A_out <= ADR_in;
83                      PC_INC<='1';
84                  when "11000" =>
85                      OP_out<="000";  -- addition in ALU
86                      ALU1_out<=ACC_in;
87                      ALU2_out<=TEMP_in;
88                      PC_INC<='1';
89                  when "00001" =>
90                      OP_out<="111";
91                      Enable<="00000";    -- all registers disabled
92                      A_out<=ADR_in;
93                      D_out<=ACC_in;
94                      WR<='1';    -- value from accumulator is written in the
                            memory
95                      PC_INC<='1';
96                  when "00011" =>
97                      WR<='0';
98                      case stat is
99                          when "00" =>    --checks if ALU result is positive
100                             Enable<="00011";
101                             A_out<=ADR_in;      -- write
102                             D_out<=TEMP_in; --writes TEMP to address 100(B=C)
103                             WR<='1';
104                             PC_INC<='1';
105                         when "10" =>    --checks if ALU result is negative
106                             A_out<="00110010"; -- address 50 that has no
                                    instruction
107                             Enable<="00001";    --INS_REG JNEG
108                         when others => null;
109                     end case;
110                 when "00111" =>
111                     WR<='0';
112                     Enable<="10000";        --loads ACC
113                     A_out<=ADR_in;      -- loads ACC with new value.
114                     PC_INC<='1';
115                 when others =>
116                     PC_INC<='1';    -- increment program counter
117                 end case;
118         when others => null;
119     end case;
120 end process;
121 end logic;
```

Listing 2: Memory

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity RAM is
6
7      PORT(
```

```
8         CLK      :   in       std_logic;
9         RST      :   in       std_logic;
10        WR       :   in       std_logic;
11        A        :   in       std_logic_vector(7 downto 0):=(others => '0');
12        D_in     :   in       std_logic_vector(7 downto 0);
13        D_out    :   out std_logic_vector(7 downto 0)
14        );
15 end RAM;
16
17 ARCHITECTURE ram OF RAM is
18     type memory is array(0 to 255) of std_logic_vector(7 downto 0);
19     signal  ram_array:  memory := (
20 --instuction set from address 0 to 5
21             0 => "00000001", --load adr1 =100       VAL=1
22             1 => "01100100", --load adr2 =101       VAL=100
23             2 => "01100101", --load acc             VAL=101
24             3 => "00000100", --load temp reg            VAL=4
25             4 => "00010100", --ADD                      VAL=20
26             5 => "00000111", --store acc to memory  VAL=7
27             6 => "00001100", --check if A>=0 then B=C VAL=12
28             7 => "00110011", --new value of B to ACC    VAL=51
29             8 => "01100101", --load acc             VAL=101
30             9 => "00000100", --load temp reg            VAL=4
31             10 => "00010100", --ADD                     VAL=20
32             11 => "00000111", --store acc to memory VAL=7
33
34             100 => "10000101", --storing value -123 in address 100      VAL
                   =-123
35             101 => "00100011", --storing value 35 in adress 101         VAL
                   =35
36
37             --100 => "00000101", --storing value 5 in address 100       VAL=5
38             --101 => "00000011", --storing value 3 in adress 101
                   VAL=3
39             others => "00000000"
40             );
41
42 begin
43 process(RST,WR,A,D_in,CLK)
44         begin
45             if rising_edge(CLK) then
46                 if RST='0' then
47                     if WR='1' then
48                         ram_array(conv_integer(A)) <= D_in;
49                     end if;
50                 end if;
51             end if;
52         end process;
53 D_out <= ram_array(conv_integer(A));
54 end ram;
```

Listing 3: Arithematic Logic Unit

```
1  Library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.numeric_std.all;
4
5  entity ALU is
6     Port (
7      A_in                :   in   std_logic_vector(7 downto 0);-- 8 bits input
8     B_in              :   in   std_logic_vector(7 downto 0);-- 8 bit input
9      OP                  :   in   STD_LOGIC_VECTOR(2 downto 0);-- 3 bits input
           for selecting function
10     A_out               :   out STD_LOGIC_VECTOR(7 downto 0);-- 8 bits output
11     stat                :   out std_logic_vector(1 downto 0)--status bit
12     );
13 end ALU;
14
15 architecture unit of ALU is
16
17 signal Result    : std_logic_vector (7 downto 0);
18 signal tmp       : std_logic_vector(8 downto 0);
19 begin
```

```
20 process(A_in,B_in,OP,Result,tmp)
21     variable flag : std_logic;
22  begin
23     stat<="11";
24     Result<="00000000";
25     tmp<="000000000";
26   case(OP) is
27
28   when "000" =>                    -- Addition
29                       tmp <= std_logic_vector(signed(A_in(7) & A_in) + signed(
                             B_in(7) & B_in));
30                       Result <= tmp(7 downto 0);
31                       flag := tmp(8);
32                       stat(1) <= Result(7) xor A_in(7) xor B_in(7) xor flag;--
                             overflow bit
33
34   when "001" =>                   -- Subtraction
35                       tmp <= std_logic_vector(signed(A_in(7) & A_in) - signed(
                             B_in(7) & B_in));
36                       Result <= tmp(7 downto 0);
37                       flag := tmp(8);
38                       stat(1) <= Result(7) xor A_in(7) xor B_in(7) xor flag;
39
40   when "010" => Result <= A_in and B_in;  -- Logical AND
41
42   when "011" => Result <= A_in or B_in;  -- Logical OR
43
44   when others => null;
45
46 end case;
47
48 case (Result) is
49     when "00000000" => stat(0) <= '1'; --if result is zero
50     when others      => stat(0) <= '0'; -- if result is not zero
51 end case;
52
53 A_out<=Result;
54
55 end process;
56
57 end unit;
```

Listing 4: Program counter

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use IEEE.Std_logic_Unsigned.all;
5
6  entity PC is
7    port(
8      CLK         :   in std_logic;
9      reset       :   in std_logic;
10     PC_INC      :   in std_logic;--to increment program counter
11     instruction :   out std_logic_vector(7 downto 0)-- next instruction
12   );
13 end PC;
14
15
16 architecture logic of PC is
17
18 signal nxt : std_logic_vector(7 downto 0):="00000000";
19
20 begin
21 process(CLK,reset,PC_INC,nxt)
22   begin
23     if reset='1' then
24         nxt <= "00000000";
25     elsif rising_edge(CLK) then
26         if PC_INC='1' then
27             nxt <= nxt +'1';
28         end if;
29     end if;
```

```
30      end process;
31  instruction<=nxt;
32  end logic;
```

Listing 5: Instruction Finder

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4
5   entity INS_REG is
6       Port(
7           Enable_bit  :   in  std_logic_vector(4 downto 0);   -- from CU
8           INS_in      :   in std_logic_vector(7 downto 0); --from RAM
9           INS_out     :   out std_logic_vector(7 downto 0)    -- to INS_DEC
10          );
11
12  end INS_REG;
13
14  architecture logic of ins_reg is
15  begin
16  process(Enable_bit,INS_in)
17      begin
18          INS_out<="00000000";
19              if Enable_bit="00001" then
20                  INS_out<=INS_in;
21              end if;
22      end process;
23
24  end logic;
```

Listing 6: Instruction Decoder

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4
5   entity INS_DEC is
6       Port(
7           INS_in      :   in std_logic_vector(7 downto 0);
8           Enable  :   out std_logic_vector(4 downto 0)
9           );
10
11  end INS_DEC;
12
13  architecture logic of ins_dec is
14
15  signal Enable_out : std_logic_vector(4 downto 0);
16
17  begin
18  process(INS_in)
19      begin
20          Enable_out<="00000";
21              case (INS_in) is
22                  when "00000001" =>  Enable_out  <=  "00100"; --ADR1
23
24                  when "01100100" =>  Enable_out  <=  "00010";    --ADR2
25
26                  when "01100101" =>  Enable_out  <=  "10000";    --ACC
27
28                  when "00000100" =>  Enable_out  <=  "01000";    --TEMP
29
30                  when "00010100" =>  Enable_out  <=  "11000";    -- ADD
31
32                  when "00000111" =>  Enable_out  <=  "00001";    --STORE IN
                        ACC
33
34                  when "00001100" =>  Enable_out <= "00011"; --A>=0 then B=C
35
36                  when "00110011" =>  Enable_out <= "00111"; --loads ACC with
                        new value
37
```

```
38                 when others => Enable_out <= "00000";
39
40             end case;
41     end process;
42     Enable <= Enable_out;
43 end logic;
```

Listing 7: Accumulator

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity ACC is
6      Port(
7          CLK          :    in std_logic;
8          ACC_in       :    in std_logic_vector(7 downto 0);
9          ALU_Result   :    in std_logic_vector(7 downto 0);
10         ACC_out      :    out std_logic_vector(7 downto 0);
11         Enable_bit   :    in std_logic_vector(4 downto 0);--to enable ACC
12         OP           :    in  std_logic_vector(2 downto 0)
13         );
14
15 end ACC;
16
17 architecture logic of ACC is
18
19 signal output : std_logic_vector(7 downto 0);
20
21 begin
22 process(ACC_in,ALU_Result,Enable_bit,OP,CLK,output)
23     begin
24         if rising_edge(CLK) then
25             if Enable_bit = "10000" then     --enables Accumulator
26                 output <= ACC_in;
27             elsif OP = "000" then
28                 output<= ALU_Result;
29             end if;
30         end if;
31 end process;
32     ACC_out<=output;
33 end logic;
```

Listing 8: Temporary Register

```
1
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity TEMP is
7      Port(
8          CLK          :    in std_logic;
9          TEMP_in      :    in std_logic_vector(7 downto 0);--from RAM
10         TEMP_out     :    out std_logic_vector(7 downto 0);
11         Enable_bit   :    in std_logic_vector(4 downto 0)--enables TEMP_REG
12         );
13
14 end TEMP;
15
16 architecture logic of TEMP is
17
18
19 signal output : std_logic_vector(7 downto 0);
20
21
22 begin
23 process(CLK,TEMP_in,Enable_bit,output)
24     begin
25         if rising_edge(CLK) then
26             if Enable_bit="01000"then    --enables temporary register
27                 output <= TEMP_in;
```

```
28              end if;
29          end if;
30      end process;
31      TEMP_out<=output;
32  end logic;
```

Listing 9: Address Register 1

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4
5   entity ADR1 is
6       Port(
7           CLK          :   in std_logic;
8           ADR1_in      :   in std_logic_vector(7 downto 0);--from RAM
9           ADR1_out     :   out std_logic_vector(7 downto 0);   -- to ADR_MUX
10          Enable_bit   :   in  std_logic_vector(4 downto 0)--enables address
                register 1
11          );
12  end ADR1;
13
14  architecture logic of ADR1 is
15
16
17  signal ADR  :   std_logic_vector(7 downto 0);
18
19
20  begin
21  process(CLK,ADR1_in,Enable_bit,ADR)
22      begin
23          if rising_edge(CLK) then
24              if Enable_bit = "00100" then    --enables address 1 register
25                  ADR <= ADR1_in;
26              end if;
27          end if;
28  end process;
29  ADR1_out<=ADR;
30  end logic;
```

Listing 10: Address Register 2

```
1
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   use ieee.std_logic_unsigned.all;
5
6   entity ADR2 is
7       Port(
8           CLK          :   in std_logic;
9           ADR2_in      :   in std_logic_vector(7 downto 0);--from RAM
10          ADR2_out         :   out std_logic_vector(7 downto 0);   -- to ADR_MUX
11          Enable_bit   :   in std_logic_vector(4 downto 0)--enables address
                register 2
12          );
13
14  end ADR2;
15
16  architecture logic of ADR2 is
17
18  signal ADR  :   std_logic_vector(7 downto 0);
19
20  begin
21  process(CLK,ADR2_in,Enable_bit,ADR)
22      begin
23          if rising_edge(CLK) then
24              if Enable_bit = "00010" then    --enables address 2 register
25                  ADR <= ADR2_in;
26              end if;
27          end if;
28      end process;
29  ADR2_out<=ADR;
```

```
30 end logic;
```

Listing 11: Address Mux

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity ADR_MUX is
6      Port (
7          ADR1_in      :   in      std_logic_vector(7 downto 0);--from ADR1
               register
8          ADR2_in      :   in  std_logic_vector(7 downto 0);--from ADR2 register
9          Enable       :   in  std_logic_vector(4 downto 0);
10         MUX_out      :   out std_logic_vector(7 downto 0)
11         );
12
13 end ADR_MUX;
14
15 architecture logic of ADR_MUX is
16
17 signal ADR  :   std_logic_vector(7 downto 0);
18
19 begin
20 process(Enable,ADR1_in,ADR2_in,ADR)
21     begin
22         ADR<="11111111";
23         case Enable is
24             when "10000" => ADR <= ADR1_in;
25             when "00000" =>     ADR <= ADR1_in;
26             when "00011" => ADR <= ADR1_in;
27             when "01000" =>     ADR <= ADR2_in;
28             when "00010" =>     ADR <= ADR2_in;
29             when others =>   null;
30         end case;
31 end process;
32 MUX_out<=ADR;
33 end logic;
```

References

[1] *International Journal of Electrical Electronics and Computer Science Engineering. Custom 8 Bit Microprocessor Designing and Implementation on FPGA Board Gautam R. Gare, Kenneth Peterand Amaresh L. R.*

[2] *8 Bit Microprocessor Using VHDL. Pallavi Deshmane, Maithili Lad, Pooja Mhetre, Sharan Kumar: Electronics department*

[3] *Modeling Registers and Counters.*
Website: https://www.Xilinx.com/university

[4] *Design of a General Purpose 8-bit RISC Processor for Computer Architecture Learning. Antonio Hernández Zavala, Oscar Camacho Nieto, Jorge A. Huerta Ruelas, Arodí R. Carvallo Domínguez*

[5] *Microcontroller and microprocessor theory and applications.*