CSE590- Computer Architecture

PROJECT 1 : REPORT

DESIGN OF AN 8-BIT NON-PIPELINED PROCESSOR

Group Members:

Swati Sajee Kumar : (50289560)

Venugopal Shah : (50291126)

ABSTRACT:

Designed an 8-bit microprocessor using Verilog HDL by using Structural Verilog modelling. The individual components were designed using behavioral modelling.

INPUT GIVEN:

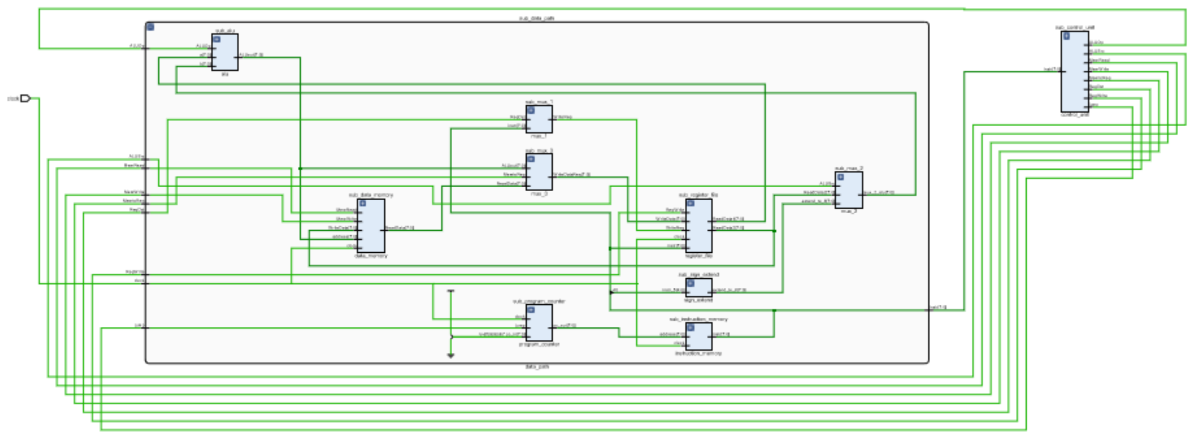
Table 1: The Instruction and Instruction format used

Instruction	**Instruction Type**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
load	I	0	0	0	1	0	0	0	0
store	I	0	0	1	1	0	0	0	1
add	R	0	1	0	1	0	0	1	0
addi	I	0	1	1	1	0	0	1	1
sub	R	1	0	0	0	1	1	0	0
jump	J	1	0	1	1	0	1	0	1

Table 2: Control Unit Signals for given instructions

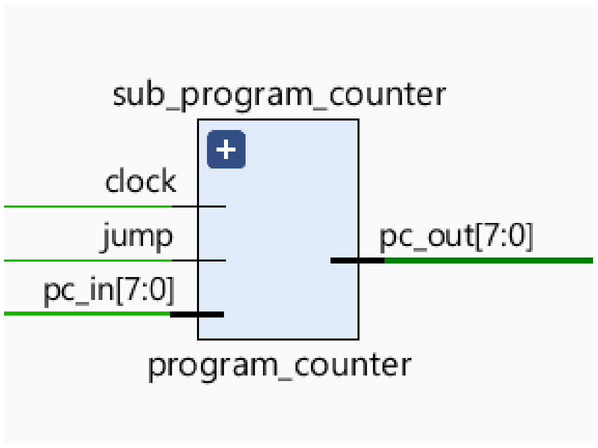
Instruction	**RegDst**	**MemtoReg**	**ALUOp**	**jump**	**MemRead**	**MemWrite**	**ALUSrc**	**RegWrite**
load	0	1	0	0	1	0	1	1
store	0	0	0	0	0	1	1	0
add	1	0	0	0	0	0	0	1
addi	0	0	0	0	0	0	1	1
sub	1	0	1	0	0	0	0	1
jump	0	0	0	1	0	0	0	0

Block Diagram- Overall View:



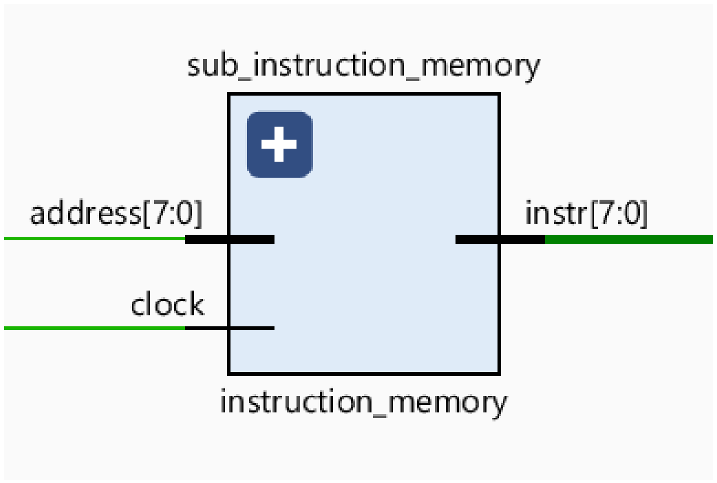
Block Diagrams of Individual Components:

1. **Program Counter**



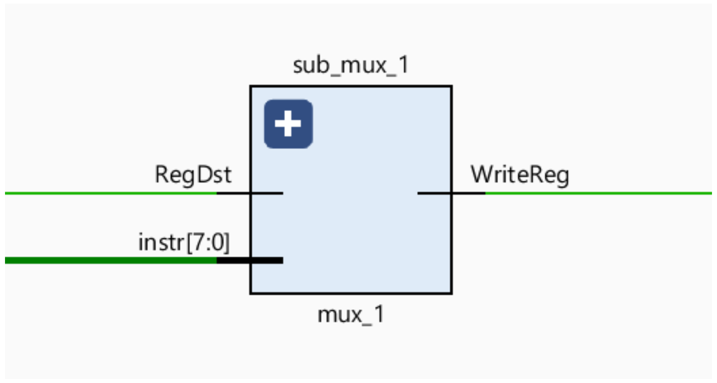
Inputs : clock,jump,pc_in[7:0]
Output:pc_out[7:0]
Working: In default program Counter increments the instruction address by 1. Except when "jump" is set high , the program counter jumps to the offset value following the opcode. Which is instr [4:0].

2. Instruction Memory



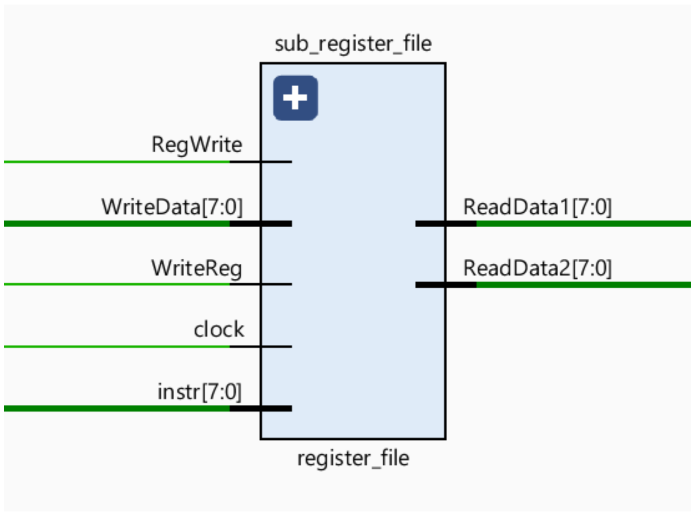
Inputs : clock, address[7:0]
Output: instr[7:0]
Working: Instruction memory gets the input address of the instruction from program counter.It breaks the code as R type, I type or J type. However here we are just passing the whole instr[7:0] to next module.

3. Mux 1



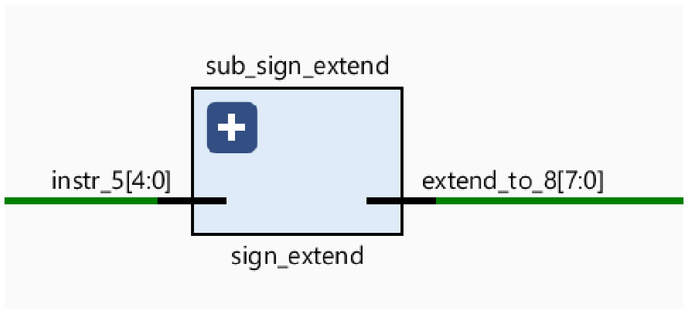
Inputs : RegDst,instr[7:0]
Output: WriteReg
Working: Since we just have two register R0,R1. This step can be skipped as well. We get the register destination from the instruction's 4th bit that is instr[4]. We input instruction[7:0] then extract instr[3] and instr[4] giving condition that is RegDst is 0 to extract instr[3] else if RegDst is 1 to extract instr[4] (this is used for R type instruction format). Also the RegDst is a control signal obtained from control unit.

4. Register File



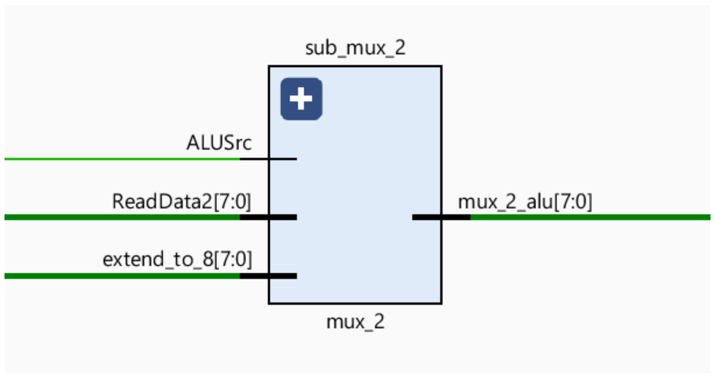
Inputs : RegWrite , WriteData[7:0], WriteReg, clock, instr[7:0]
Output: ReadData1[7:0], ReadData2[7:0]
Working: We have two registers R0 and R1 storing 8 bits of data. We input the instruction[7:0] and then store the instr[3] as ReadReg1 and instr[4] as ReadReg2. When RegWrite is set to 1, we use it for "store" command, where data needs to be written to the memory.

5. Sign Extend



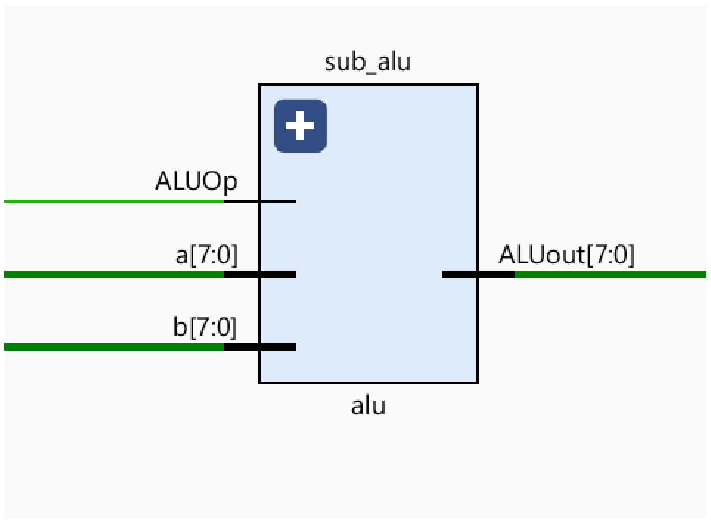
Inputs : instr[4:0]
Output: extend_to_8 [7:0]
Working: This module is used for *load, store and addi* instructions. Since they have I type instruction format. The last 5 bits of the instruction format is extended to 8 bit by adding three zeros in front of it.

6. Mux 2



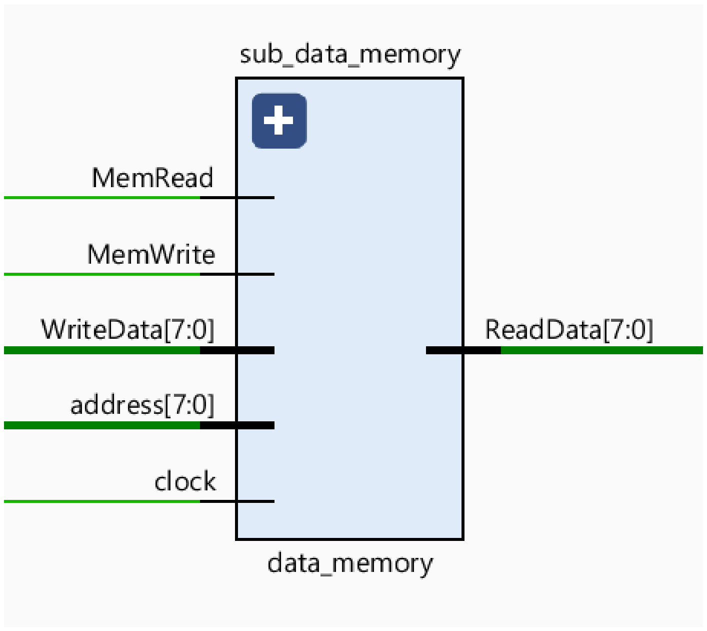
Inputs : RegData2[7:0], extend_to_8[7:0], ALUSrc
Output: mux_2_alu[7:0]
Working: This module is used as a deciding factor wheter it is an addi or add instruction. This can be decided on the basis of ALUSrc. If ALUSrc is set to 1, mux takes extend_to_8 as input,else it takes ReadData2[7:0] given from Register file as input to ALU.

7. ALU



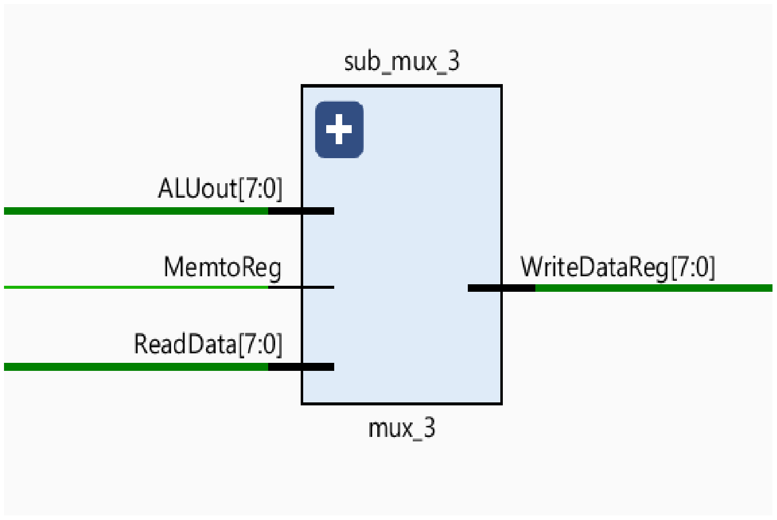
Inputs : a[7:0],b[7:0],ALUOp
Output: ALUout[7:0]
Working: The input a is array we obtained from ReadData1 from register file, the input b is the array we obtained from mux-2 . ALUOp decides whether we need to use an “add” operand or “subtract”. (Note: For jump, load, store, add, addi we need to add the two input values).

8. Data Memory



Inputs : MemRead, MemWrite, WriteData[7:0],address[7:0],clock
Output: ReadData[7:0]
Working: Since we just have two register R0,R1. This step can be skipped as well. We get the register destination from the instruction's 4th bit that is instr[4]. We input instruction[7:0] then extract instr[3] and instr[4] giving condition that is RegDst is 0 to extract instr[3] else if RegDst is 1 to extract instr[4] (this is used for R type instruction format). Also the RegDst is a control signal obtained from control unit.

9. Mux 3



Inputs : ALUout,ReadData[7:0],MemtoReg
Output: WriteDataReg[7:0]

Working: This mux is used to write data value to memory in register file. It is controlled using control signal “MemtoReg” , if MemtoReg is set as zero , mux selects ALUout as input, else it selects ReadData[7:0] (from data memory) as input.

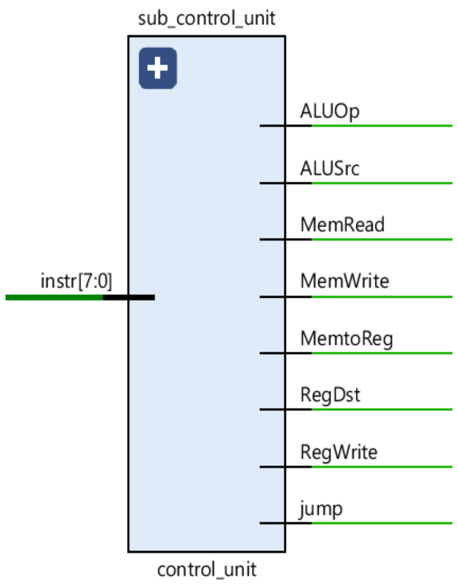
10. Data Path

Inputs : ALUOp, ALUSrc, MemRead,MemWrite,MemtoReg,RegDst, RegWrite,jump

Output: instr[7:0]

Working: Data path is not a physical module, but it connects all the above modules. That is it looks after the data flow of the program. The inputs of data path are the outputs of control unit, more about control unit is explained below.

11. Control Unit



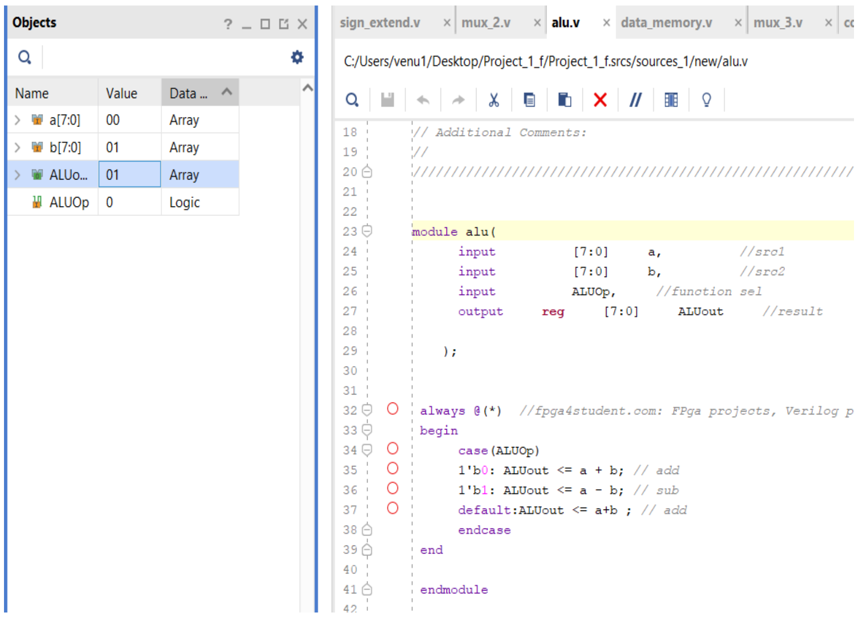
Inputs :instr[7:0]

Output : ALUOp, ALUSrc, MemRead,MemWrite,MemtoReg,RegDst, RegWrite,jump

Working: Control unit is used as a select signal for all the modules mentioned above. The individual working of the signals has been explained in above modules. The output of data path which is instruction is given as input to control unit. And the outputs from control unit is wired to input of modules by using another module – top level module, where we instantiate both data path and control unit.

SIMULATION RESULTS:

1. *add*



Objects

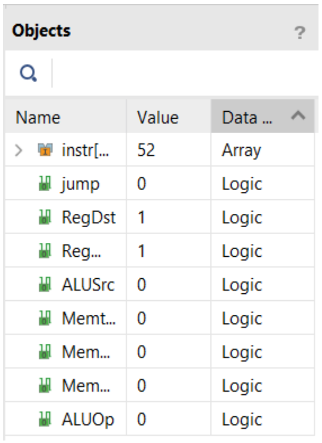
Name	Value	Data ...
a[7:0]	00	Array
b[7:0]	01	Array
ALUo...	01	Array
ALUOp	0	Logic

C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/new/alu.v

```
// Additional Comments:
//
//////////////////////////////////////////////////////


module alu(
      input          [7:0]     a,          //src1
      input          [7:0]     b,          //src2
      input          ALUOp,     //function sel
      output     reg      [7:0]     ALUout     //result

   );


 always @(*)  //fpga4student.com: FPga projects, Verilog p
 begin
      case(ALUOp)
      1'b0: ALUout <= a + b; // add
      1'b1: ALUout <= a - b; // sub
      default:ALUout <= a+b ; // add
      endcase
 end

 endmodule
```

Fig1.1 : Output for ALU

- The add instruction is **8'b01010010**
- Reg0 is storing value 0 (instruction[3])
- Reg1 is storing value 1 (instruction[4])
- The output is ALUout = 8'b00000001

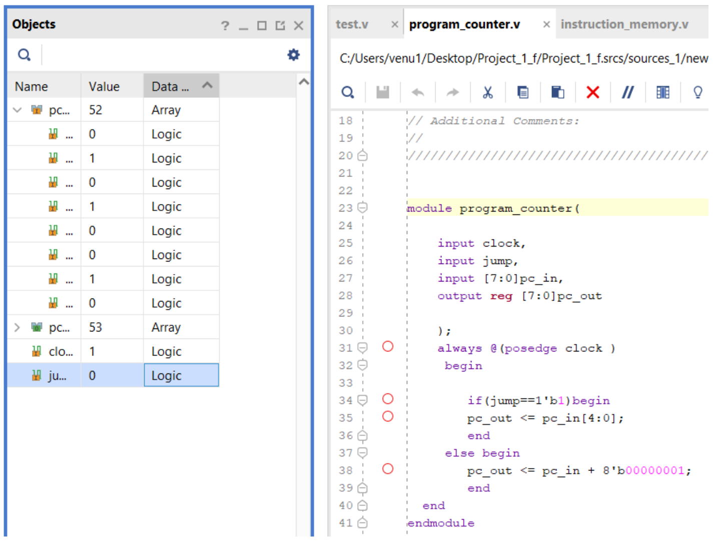


Objects

Name	Value	Data ...
instr[...	52	Array
jump	0	Logic
RegDst	1	Logic
Reg...	1	Logic
ALUSrc	0	Logic
Memt...	0	Logic
Mem...	0	Logic
Mem...	0	Logic
ALUOp	0	Logic

Fig1.2 Control Unit Output

- Only RegDst and RegRead are set at 1
- When ALUOp is 0 we do add , else subtract



Objects

Name	Value	Data ...
pc...	52	Array
...	0	Logic
...	1	Logic
...	0	Logic
...	1	Logic
...	0	Logic
...	0	Logic
...	1	Logic
...	0	Logic
pc...	53	Array
clo...	1	Logic
ju...	0	Logic

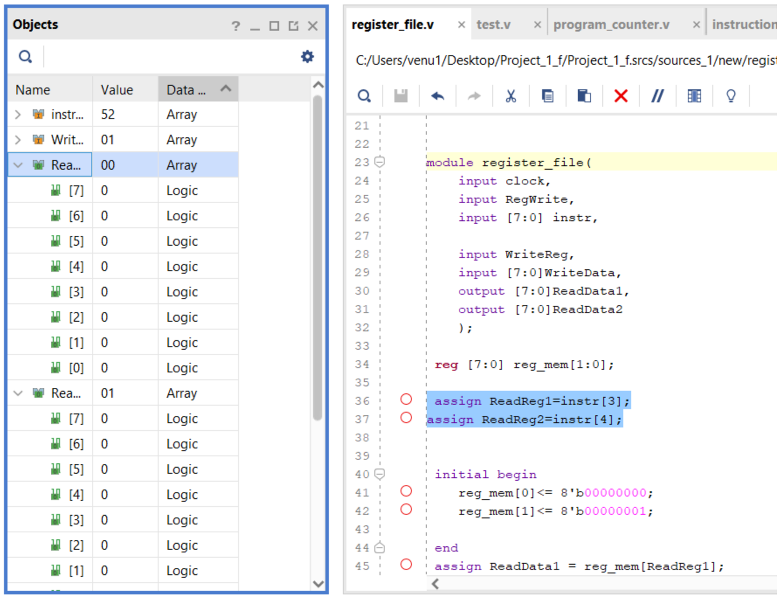
test.v | program_counter.v | instruction_memory.v

C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/new

```
// Additional Comments:
//
////////////////////////////////////////


module program_counter(

    input clock,
    input jump,
    input [7:0]pc_in,
    output reg [7:0]pc_out

    );
    always @(posedge clock )
     begin

        if(jump==1'b1)begin
        pc_out <= pc_in[4:0];
        end
     else begin
        pc_out <= pc_in + 8'b00000001;
        end
  end
endmodule
```

Fig 1.3: Program Counter

- Program Counter increments the address by 1, after completion of instruction
- Here it goes from 52 to 53. (Since input instruction in decimal is 52)



Objects

Name	Value	Data ...
instr...	52	Array
Writ...	01	Array
Rea...	00	Array
[7]	0	Logic
[6]	0	Logic
[5]	0	Logic
[4]	0	Logic
[3]	0	Logic
[2]	0	Logic
[1]	0	Logic
[0]	0	Logic
Rea...	01	Array
[7]	0	Logic
[6]	0	Logic
[5]	0	Logic
[4]	0	Logic
[3]	0	Logic
[2]	0	Logic
[1]	0	Logic

register_file.v | test.v | program_counter.v | instruction

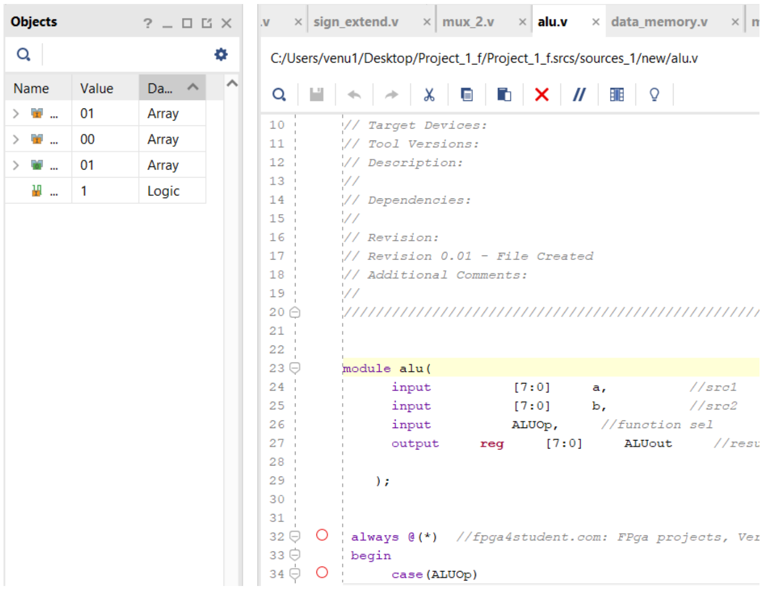
C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/new/regis

```


module register_file(
    input clock,
    input RegWrite,
    input [7:0] instr,

    input WriteReg,
    input [7:0]WriteData,
    output [7:0]ReadData1,
    output [7:0]ReadData2
    );

 reg [7:0] reg_mem[1:0];

 assign ReadReg1=instr[3];
assign ReadReg2=instr[4];


 initial begin
    reg_mem[0]<= 8'b00000000;
    reg_mem[1]<= 8'b00000001;

 end
 assign ReadData1 = reg_mem[ReadReg1];
```

Fig. 1.4 Register File

- Register file stores the instruction [3] data and instruction [4] data, which is 0 and 1 respectively (On the basis of the Instruction [7:0])

2. ***sub***



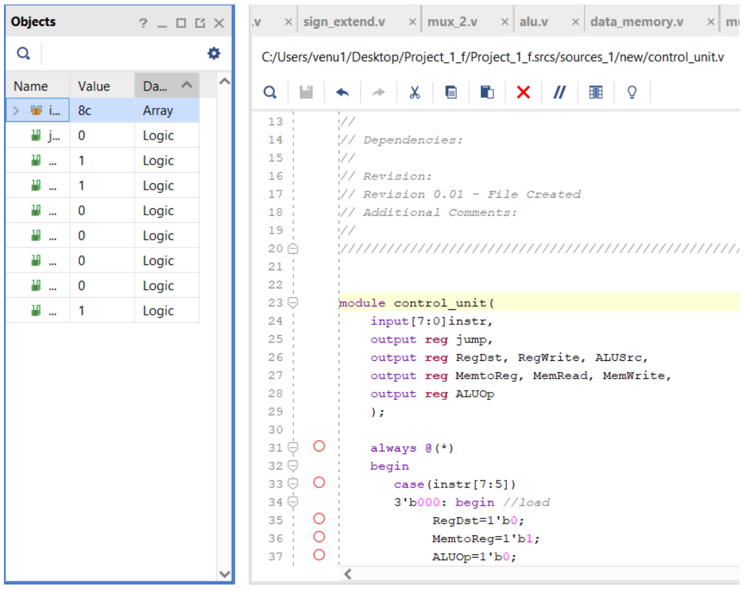
Name	Value	Da...
...	01	Array
...	00	Array
...	01	Array
...	1	Logic

C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/new/alu.v

```
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////


module alu(
      input          [7:0]     a,          //src1
      input          [7:0]     b,          //src2
      input          ALUOp,      //function sel
      output    reg      [7:0]     ALUout     //resu

    );


 always @(*)  //fpga4student.com: FPga projects, Ver
 begin
      case(ALUOp)
```

Fig. 2.1: ALU output for subtraction

- The Instruction for sub is **8'b10001100**
- Reg0 is storing value 0 (instruction[3])
- Reg1 is storing value 1 (instruction[4])
- The output is ALUout = 8'b00000001



Name	Value	Da...
i...	8c	Array
j...	0	Logic
...	1	Logic
...	1	Logic
...	0	Logic
...	0	Logic
...	0	Logic
...	0	Logic
...	1	Logic

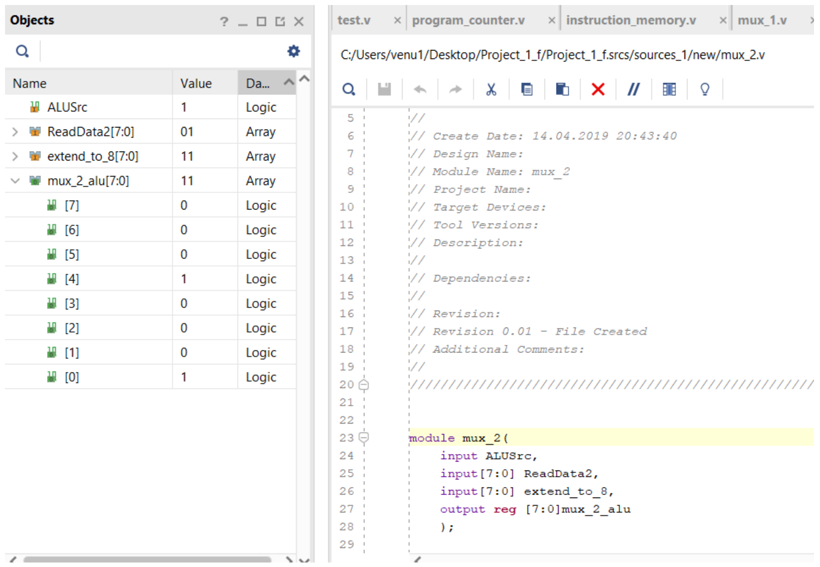
C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/new/control_unit.v

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////


module control_unit(
    input[7:0]instr,
    output reg jump,
    output reg RegDst, RegWrite, ALUSrc,
    output reg MemtoReg, MemRead, MemWrite,
    output reg ALUOp
    );

    always @(*)
    begin
       case(instr[7:5])
       3'b000: begin //load
            RegDst=1'b0;
            MemtoReg=1'b1;
            ALUOp=1'b0;
```

Fig. 2.2 Control Unit Signals

- The ALUOp (last) is set to 1 in case of subtraction only.

3. *store*



Objects

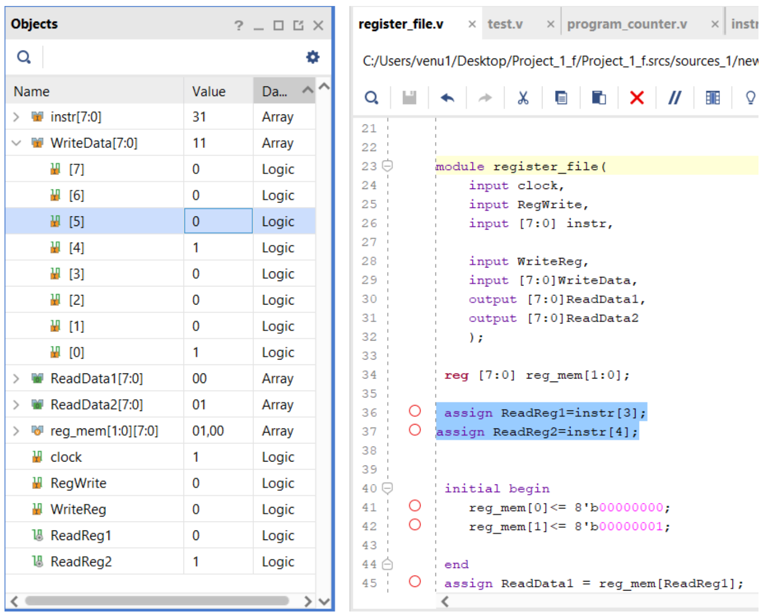
Name	Value	Da...
ALUSrc	1	Logic
ReadData2[7:0]	01	Array
extend_to_8[7:0]	11	Array
mux_2_alu[7:0]	11	Array
[7]	0	Logic
[6]	0	Logic
[5]	0	Logic
[4]	1	Logic
[3]	0	Logic
[2]	0	Logic
[1]	0	Logic
[0]	1	Logic

C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/new/mux_2.v

```
//
// Create Date: 14.04.2019 20:43:40
// Design Name:
// Module Name: mux_2
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////


module mux_2(
    input ALUSrc,
    input[7:0] ReadData2,
    input[7:0] extend_to_8,
    output reg [7:0]mux_2_alu
    );

```

Fig. 3.1 : Input and outputs for mux_2

- *The instruction for store is 8'b00110001*
- *The ALUSrc is set 1*
- *Also it passes through the signed extend module and stores the value 8'b00010001*
- *The output of the mux is same as signed extended output*



Objects

Name	Value	Da...
instr[7:0]	31	Array
WriteData[7:0]	11	Array
[7]	0	Logic
[6]	0	Logic
[5]	0	Logic
[4]	1	Logic
[3]	0	Logic
[2]	0	Logic
[1]	0	Logic
[0]	1	Logic
ReadData1[7:0]	00	Array
ReadData2[7:0]	01	Array
reg_mem[1:0][7:0]	01,00	Array
clock	1	Logic
RegWrite	0	Logic
WriteReg	0	Logic
ReadReg1	0	Logic
ReadReg2	1	Logic

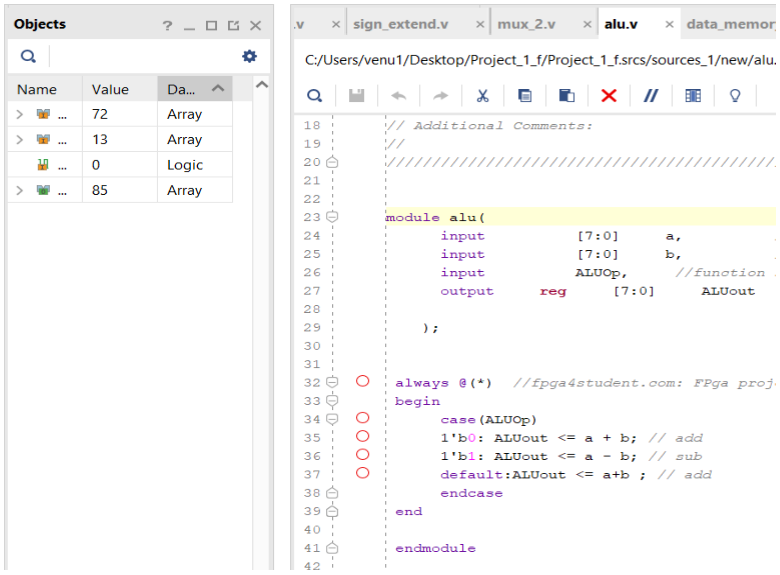
C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/new

```


module register_file(
    input clock,
    input RegWrite,
    input [7:0] instr,

    input WriteReg,
    input [7:0]WriteData,
    output [7:0]ReadData1,
    output [7:0]ReadData2
    );

 reg [7:0] reg_mem[1:0];

 assign ReadReg1=instr[3];
assign ReadReg2=instr[4];


 initial begin
    reg_mem[0]<= 8'b00000000;
    reg_mem[1]<= 8'b00000001;

 end
 assign ReadData1 = reg_mem[ReadReg1];
```

Fig. 3.2 Input and Outputs of register file

- Notice the value stored in WriteData (register) is 8'b00010001

4. *addi*



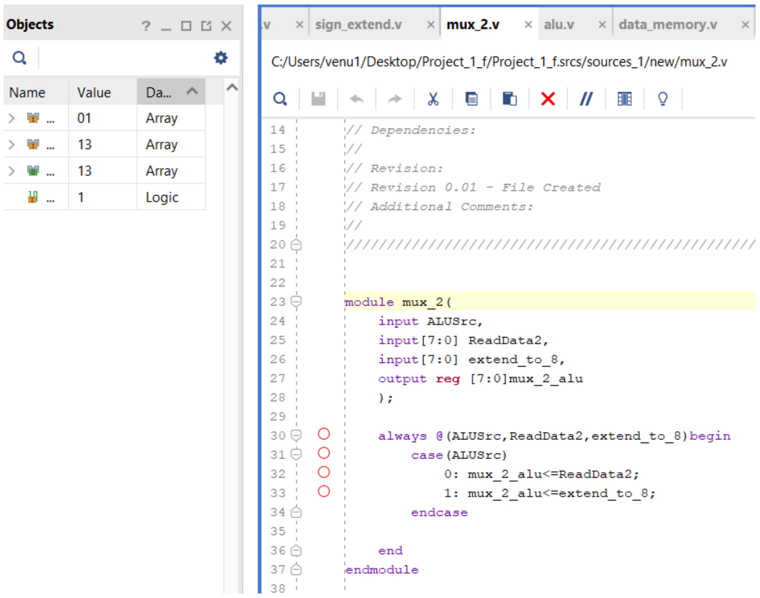
Name	Value	Da...
...	72	Array
...	13	Array
...	0	Logic
...	85	Array

C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/new/alu.

```verilog
// Additional Comments:
//
////////////////////////////////////////////


module alu(
        input           [7:0]     a,
        input           [7:0]     b,
        input          ALUOp,      //function
        output     reg      [7:0]      ALUout

    );


 always @(*)  //fpga4student.com: FPga proj
 begin
        case(ALUOp)
        1'b0: ALUout <= a + b; // add
        1'b1: ALUout <= a - b; // sub
        default:ALUout <= a+b ; // add
        endcase
 end

 endmodule
```

Fig. 4.1 ALU output

- The instruction given for addi is **8'01110011**
- "a" has the value from the register
- "b" has the value from mux, which points to immediate value
- The summation of both (a and b) is shown as decimal number 85



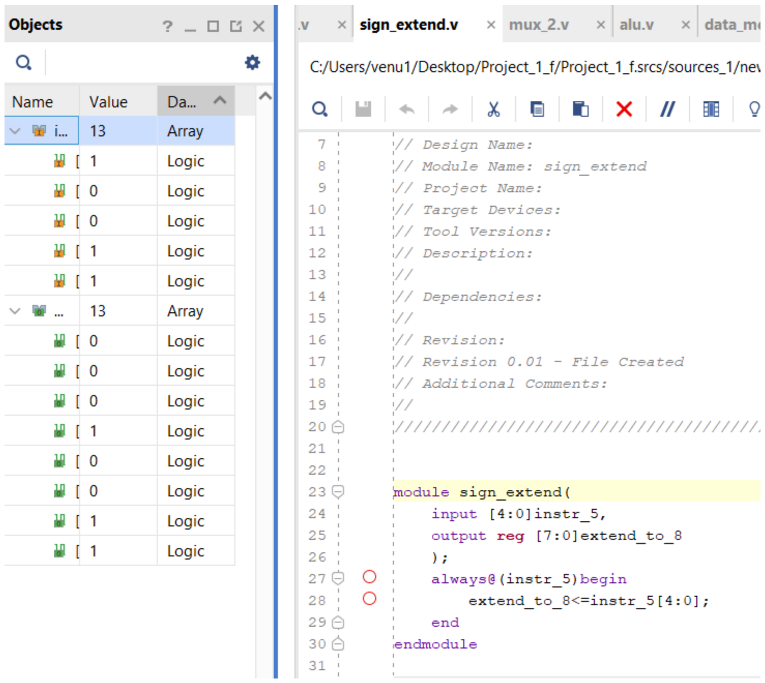
Name	Value	Da...
...	01	Array
...	13	Array
...	13	Array
...	1	Logic

C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/new/mux_2.v

```verilog
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////


module mux_2(
    input ALUSrc,
    input[7:0] ReadData2,
    input[7:0] extend_to_8,
    output reg [7:0]mux_2_alu
    );

    always @(ALUSrc,ReadData2,extend_to_8)begin
        case(ALUSrc)
            0: mux_2_alu<=ReadData2;
            1: mux_2_alu<=extend_to_8;
        endcase

    end
endmodule
```

Fig. 4.2 Mux-2

- When ALUSrc (Control signal) is set high, mux selects sign extended value as input for ALU



Objects

Name	Value	Da...
i...	13	Array
[	1	Logic
[	0	Logic
[	0	Logic
[	1	Logic
[	1	Logic
...	13	Array
[	0	Logic
[	0	Logic
[	0	Logic
[	1	Logic
[	0	Logic
[	0	Logic
[	1	Logic
[	1	Logic

.v | sign_extend.v | mux_2.v | alu.v | data_m

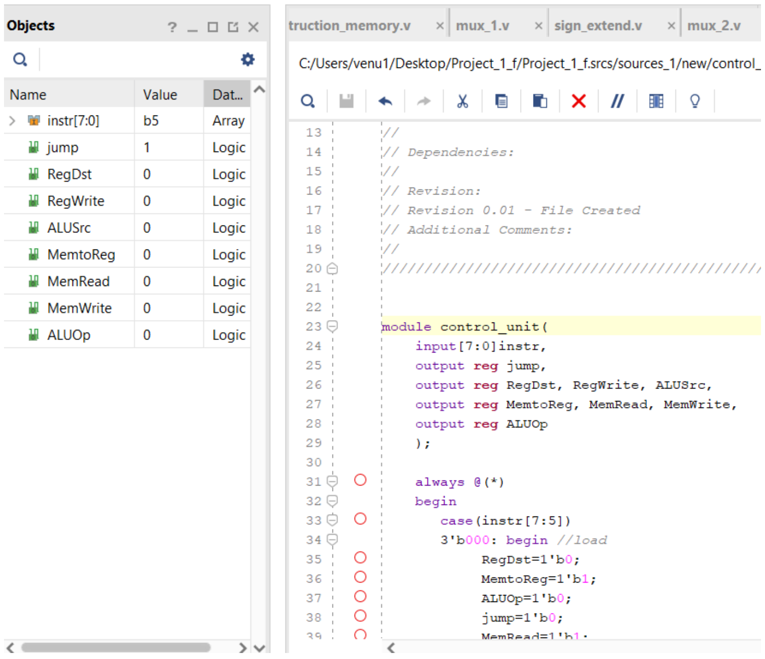
C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/nev

```
// Design Name:
// Module Name: sign_extend
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////


module sign_extend(
    input [4:0]instr_5,
    output reg [7:0]extend_to_8
    );
    always@(instr_5)begin
        extend_to_8<=instr_5[4:0];
    end
endmodule
```

Fig. 4.3 Sign Extend for addi

- We take the first five bits of the instruction [4:0] as input to sign extend and add three zeros in front to make it a eight bit instruction.

5. ***jump***



Objects

Name	Value	Dat...
instr[7:0]	b5	Array
jump	1	Logic
RegDst	0	Logic
RegWrite	0	Logic
ALUSrc	0	Logic
MemtoReg	0	Logic
MemRead	0	Logic
MemWrite	0	Logic
ALUOp	0	Logic

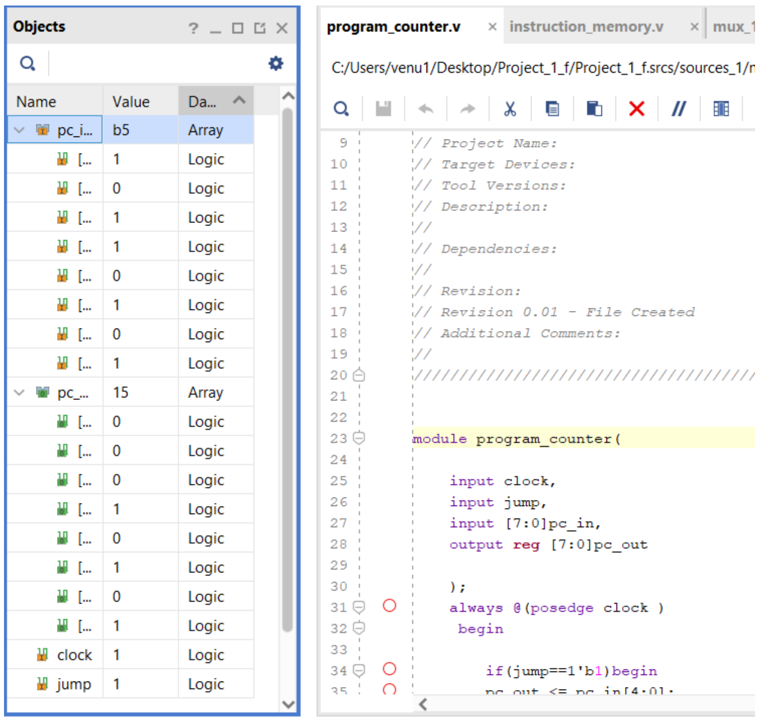
truction_memory.v | mux_1.v | sign_extend.v | mux_2.v

C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/new/control_

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////


module control_unit(
    input[7:0]instr,
    output reg jump,
    output reg RegDst, RegWrite, ALUSrc,
    output reg MemtoReg, MemRead, MemWrite,
    output reg ALUOp
    );

    always @(*)
    begin
        case(instr[7:5])
        3'b000: begin //load
            RegDst=1'b0;
            MemtoReg=1'b1;
            ALUOp=1'b0;
            jump=1'b0;
            MemRead=1'b1;
```

Fig. 5.1: Control Unit signals

- The instruction for jump is **8'b10110101**
- The "jump " control line is set high



Name	Value	Da...
pc_i...	b5	Array
[...	1	Logic
[...	0	Logic
[...	1	Logic
[...	1	Logic
[...	0	Logic
[...	1	Logic
[...	0	Logic
[...	1	Logic
pc_...	15	Array
[...	0	Logic
[...	0	Logic
[...	0	Logic
[...	1	Logic
[...	0	Logic
[...	1	Logic
[...	0	Logic
[...	1	Logic
clock	1	Logic
jump	1	Logic

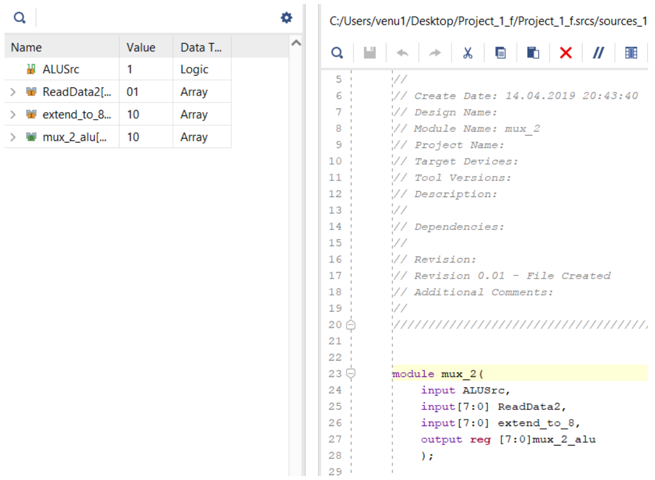
C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/r

```
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////


module program_counter(

    input clock,
    input jump,
    input [7:0]pc_in,
    output reg [7:0]pc_out

    );
    always @(posedge clock )
     begin

        if(jump==1'b1)begin
        pc_out <= pc_in[4:0];
```

Fig. 5.2: Program Counter for jump

- The Program counter now points to the last five bits of instruction [4:0] which is [0001010111].

6. load



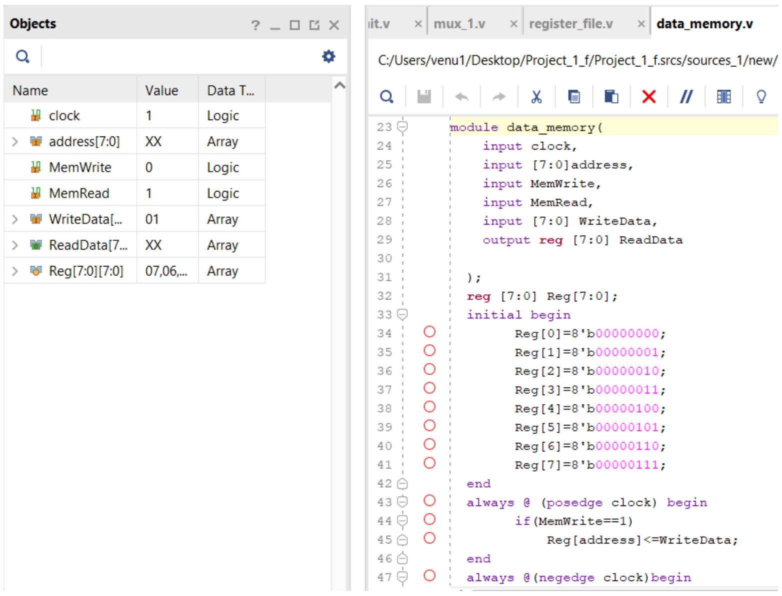
Name	Value	Data T...
ALUSrc	1	Logic
ReadData2[...	01	Array
extend_to_8...	10	Array
mux_2_alu[...	10	Array

C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1

```
//
// Create Date: 14.04.2019 20:43:40
// Design Name:
// Module Name: mux_2
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////


module mux_2(
    input ALUSrc,
    input[7:0] ReadData2,
    input[7:0] extend_to_8,
    output reg [7:0]mux_2_alu
    );
```

Fig . 6.1: Mux-2 Output and inputs

- Instruction for load used is 8’b00010000
- ALUSrc is set as high by control unit , since its an immediate transfer of value



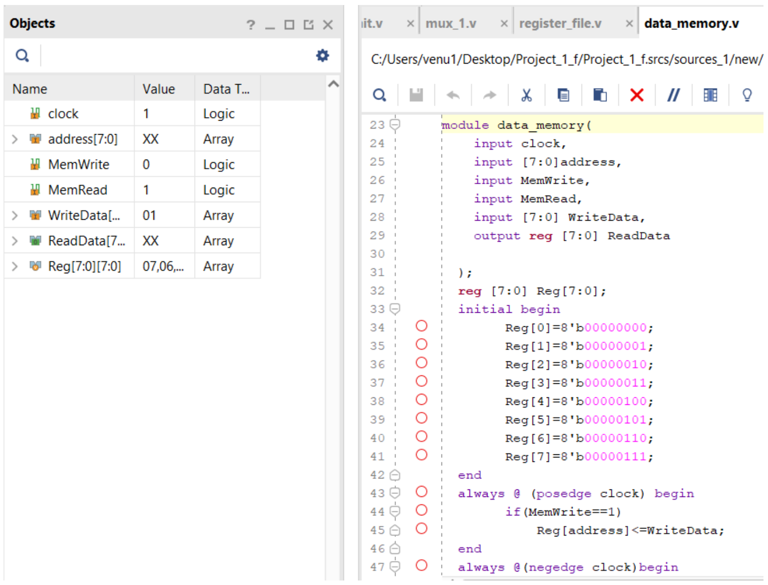
C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/new/

Name	Value	Data T...
clock	1	Logic
address[7:0]	XX	Array
MemWrite	0	Logic
MemRead	1	Logic
WriteData[...	01	Array
ReadData[7...	XX	Array
Reg[7:0][7:0]	07,06,...	Array

```
module data_memory(
    input clock,
    input [7:0]address,
    input MemWrite,
    input MemRead,
    input [7:0] WriteData,
    output reg [7:0] ReadData

  );
  reg [7:0] Reg[7:0];
  initial begin
        Reg[0]=8'b00000000;
        Reg[1]=8'b00000001;
        Reg[2]=8'b00000010;
        Reg[3]=8'b00000011;
        Reg[4]=8'b00000100;
        Reg[5]=8'b00000101;
        Reg[6]=8'b00000110;
        Reg[7]=8'b00000111;
  end
  always @ (posedge clock) begin
        if(MemWrite==1)
            Reg[address]<=WriteData;
  end
  always @(negedge clock)begin
```

Fig 6.2 : data memory output and inputs

- The register value to which we need to load the data is given as don't care
- The data to be written is given via WriteData , which is output of extended signed module.



C:/Users/venu1/Desktop/Project_1_f/Project_1_f.srcs/sources_1/new/

Name	Value	Data T...
clock	1	Logic
address[7:0]	XX	Array
MemWrite	0	Logic
MemRead	1	Logic
WriteData[...	01	Array
ReadData[7...	XX	Array
Reg[7:0][7:0]	07,06,...	Array

```
module data_memory(
    input clock,
    input [7:0]address,
    input MemWrite,
    input MemRead,
    input [7:0] WriteData,
    output reg [7:0] ReadData

  );
  reg [7:0] Reg[7:0];
  initial begin
        Reg[0]=8'b00000000;
        Reg[1]=8'b00000001;
        Reg[2]=8'b00000010;
        Reg[3]=8'b00000011;
        Reg[4]=8'b00000100;
        Reg[5]=8'b00000101;
        Reg[6]=8'b00000110;
        Reg[7]=8'b00000111;
  end
  always @ (posedge clock) begin
        if(MemWrite==1)
            Reg[address]<=WriteData;
  end
  always @(negedge clock)begin
```

Fig 6.3 : Register file output and inputs

- The Memory written to register is don't care, which is partially incorrect though.

CONTRIBUTIONS:

1. Venugopal Shah : Studied the detail flow of data and usage of control signals in 8 bit microprocessor. Also worked on the logic for the microprocessor. Specific modules worked were : instruction_memory, program_counter and contributed in other modules as well.
2. Swati Sajee Kumar : Studied the detail flow of data and control flow in 8-bits microprocessor. Worked on the logic as well as various modules such as register file, mux, data memory, datapath, control unit .

   Together we developed the logic and performed the simulation to obtain the outputs enclosed in the report above.