10-bit ISA Implementation

ECE 3570 Lab 3A

Junliang Fei, Adam Dubs,
and Connelly Bale

March 12, 2018

ISA OVERVIEW

There is **NO CHANGE** in the design result of lab1. The ISA architecture of lab1 is showed as below.

BASIC INSTRUCTION FORMATS

R-TYPE (opcode = 00 or 01)

opcode		function		rs			rt or immediate		
bit9	bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

I-TYPE (opcode = 10)

opcode		immediate							
bit9	bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

J-TYPE (opcode = 11)

opcode		function		address					
bit9	bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

INSTRUCTION SET

No.	NAME	MNEMONIC	FORMAT	Example	RTL Example	OPCODE/ FUNCT
1	Add	add	R	add $s0, $s1	reg_t0 = reg_s0 + reg_s1;	00/00
2	Sub	sub	R	sub $s0, $s1	reg_t0 = reg_s0 - reg_s1;	00/01
3	Compare	cmp	R	cmp $s0, $s1	if(reg_s0 =reg_s1), reg_t0 = 1;	00/10
4	Less Than	slt	R	slt $s0, $s1	if(reg_s0 <reg_s1), reg_t0 = 1;	00/11
5	Move	mov	R	mov $s0, $t0	reg_s0 = reg_t0;	01/00
6	Load Word	load	R	load 4($sp)	mem_rd_addr = reg_sp+4; reg_t1 = mem_rd_data;	01/01
7	Store Word	str	R	str 0($sp)	mem_wr_addr = reg_sp+4;	01/10
8	Return	ret	R	ret	instr_addr = reg_ra;	01/11
9	load immediate	ldi	I	ldi 17	reg_t1 = 17	10
10	Jump	j	J	j LOOP	instr_addr = instr_addr + jump_addr;	11/00
11	Jump True	je	J	je LOOP	if(reg_t0 = 1) instr_addr = instr_addr + jump_addr;	11/01
12	Jump and link	jal	J	jal LOOP	reg_ra = instr_addr +1; instr_addr = instr_addr + jump_addr;	11/10
13	Halt	halt	J	halt	No operation	11/11

REGISTER LIST

No.	Name	Use	Bit	Notes
1	$s0	Saved Temporary 0	000	
2	$s1	Saved Temporary 1	001	
3	$s2	Saved Temporary 2	010	
4	$s3	Saved Temporary 3	011	
5	$t0	Temporary 0	100	$t0 is used for saving the return value of add, sub, slt and cmp
6	$t1	Temporary 1	101	$t1 is used for saving the return value of load, str and ldi
7	$ra	Return Address	110	
8	$sp	Stack Pointer	111	

Memory Addresses from 0-1023.

Control Unit

The control unit was designed to act as both the control center and the decode stage.

Table 1 - Control Interface List

No	Name	Bit Width	IN/OUT	Function
1	instr	10	input	instruction input
2	done	1	output	signals the completion of a program
3	writeval_op	2	output	controls the input for reg2 in the alu
4	imm_val	10	output	the immediate value to be stored in $t1
5	fetch_op	2	output	the instruction control for the fetch unit
6	alu_op	2	output	the instruction control for the alu unit
7	wr_en	1	output	the write enabler
8	jmp_addr	10	output	the jump offset
9	read_reg1	3	output	register 1's identification number
10	read_reg2	3	output	register 2's identification number
11	wr_reg	3	output	the register to be written to
12	ldst_en	1	output	controls the data input for the wr_reg within the register unit

Design

The control unit receives only the instruction bits. Subsequently, it parses up the instruction and sends the corresponding data to each other unit. Likewise, it interprets the opcode and produces controlling bits to ensure accurate calculations are made. The longest delay was found to be 5.507 ns.

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	∞	3	2	8	instr[7]	fetch_op[1]	5.507	3.148	2.359	∞	input port clock		
Path 2	∞	3	2	8	instr[7]	wr_en	5.423	3.009	2.414	∞	input port clock		
Path 3	∞	3	2	8	instr[7]	writeval_op[1]	5.402	3.136	2.266	∞	input port clock		
Path 4	∞	2	1	8	instr[5]	jmp_addr[5]	5.345	2.956	2.389	∞	input port clock		
Path 5	∞	3	2	8	instr[7]	alu_op[1]	5.277	3.037	2.240	∞	input port clock		
Path 6	∞	2	1	8	instr[5]	jmp_addr[6]	5.267	2.956	2.312	∞	input port clock		
Path 7	∞	3	2	13	instr[8]	alu_op[0]	5.262	3.034	2.227	∞	input port clock		
Path 8	∞	3	2	13	instr[8]	fetch_op[0]	5.247	3.136	2.111	∞	input port clock		
Path 9	∞	2	1	8	instr[5]	jmp_addr[7]	5.191	2.959	2.232	∞	input port clock		
Path 10	∞	3	2	8	instr[7]	done	5.172	3.137	2.036	∞	input port clock		

Figure 1 - Longest Pathway Times

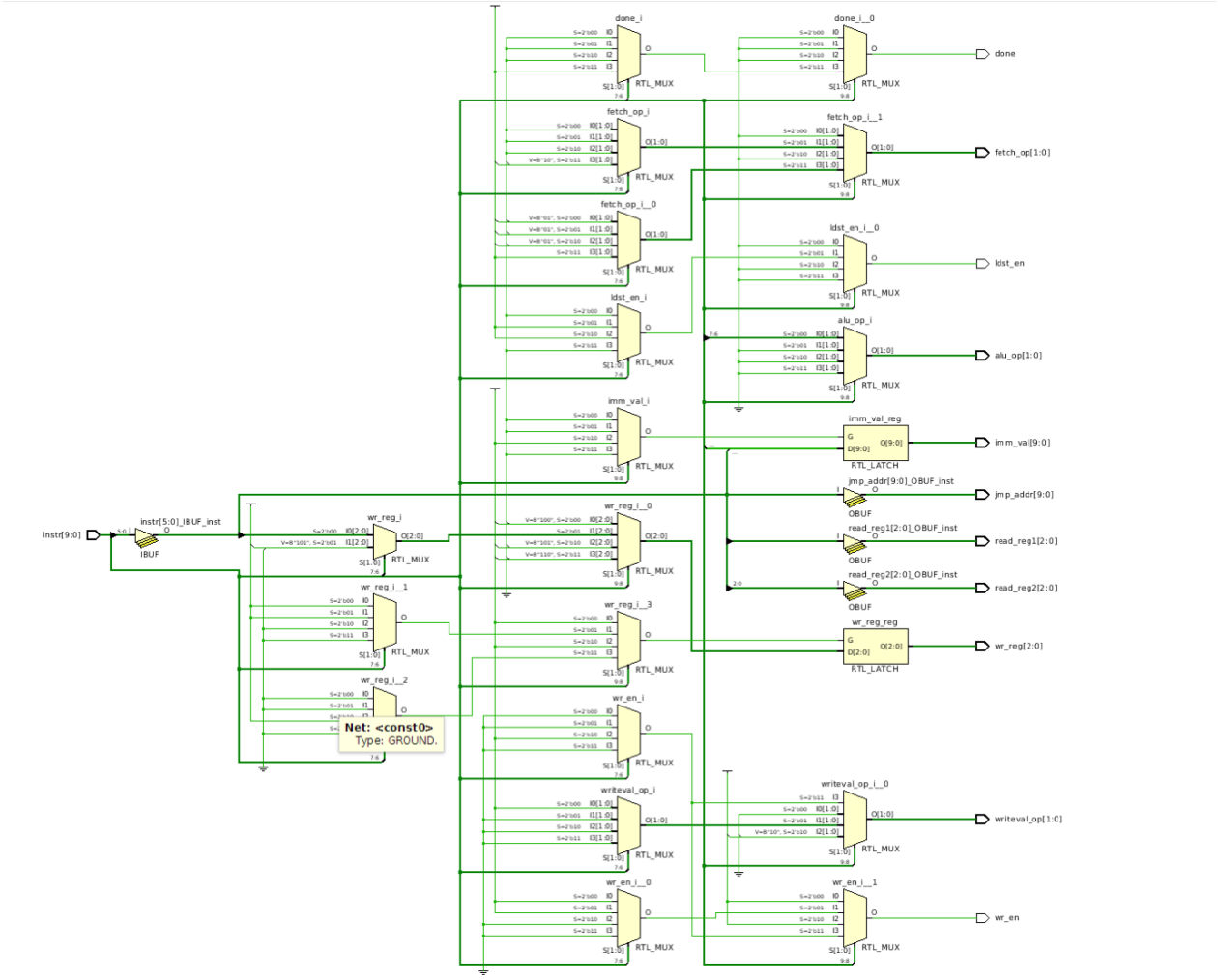


Figure 2 - Control Unit Schematic

Modifications to Old Design

I. Fetch Unit

a. Modifications

Register $t0 was added as an input to the fetch since it is needed to realize the je instruction as described in the ISA. Likewise, the program counter was made into an output of this unit so that it could be stored in $ra whenever the jal instruction is called. Finally, there is only one register realized in this design vs. two registers utilized in the old one.

b. Schematic

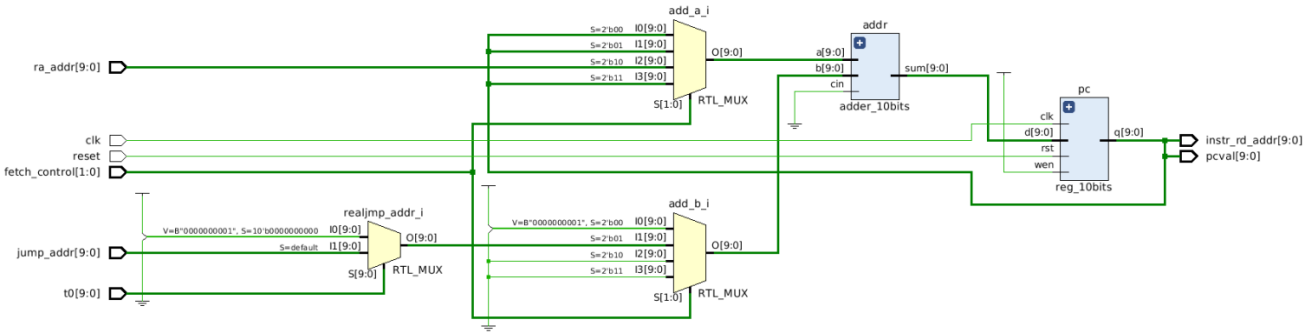


Figure 3 - Fetch Unit Schematic

c. Timing

Table 2 - Fetch Unit Pathway Timing

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	∞	7	6	10	t0[8]	pc/reg8/q_reg/D	4.606	0.963	3.642	∞	input port clock		
Path 2	∞	7	6	10	t0[8]	pc/reg6/q_reg/D	4.550	0.963	3.587	∞	input port clock		
Path 3	∞	7	6	10	t0[8]	pc/reg9/q_reg/D	4.526	0.963	3.562	∞	input port clock		
Path 4	∞	7	6	10	t0[8]	pc/reg7/q_reg/D	4.482	0.963	3.519	∞	input port clock		
Path 5	∞	6	5	10	t0[8]	pc/reg5/q_reg/D	4.268	0.920	3.348	∞	input port clock		
Path 6	∞	2	1	5	pc/reg3/q_reg/C	pcval[3]	4.138	2.438	1.699	∞			
Path 7	∞	2	1	4	pc/reg4/q_reg/C	pcval[4]	4.086	2.433	1.654	∞			
Path 8	∞	2	1	4	pc/reg8/q_reg/C	pcval[8]	4.070	2.464	1.606	∞			
Path 9	∞	2	1	4	pc/reg7/q_reg/C	pcval[7]	4.067	2.468	1.599	∞			
Path 10	∞	6	5	10	t0[8]	pc/reg4/q_reg/D	4.059	0.920	3.139	∞	input port clock		

The longest pathway in the older design was 5.688ns whereas the longest now is only 4.606ns.

II. ALU

a. *Modifications*

The program counter, the immediate value, and the write back operation were all added as input to the ALU. The write back operation is the to determine if the immediate value, the program counter, or the ALU results will be sent back to the register file. This allows for the load immediate and jump and link instructions to be realized.

b. *Schematic*

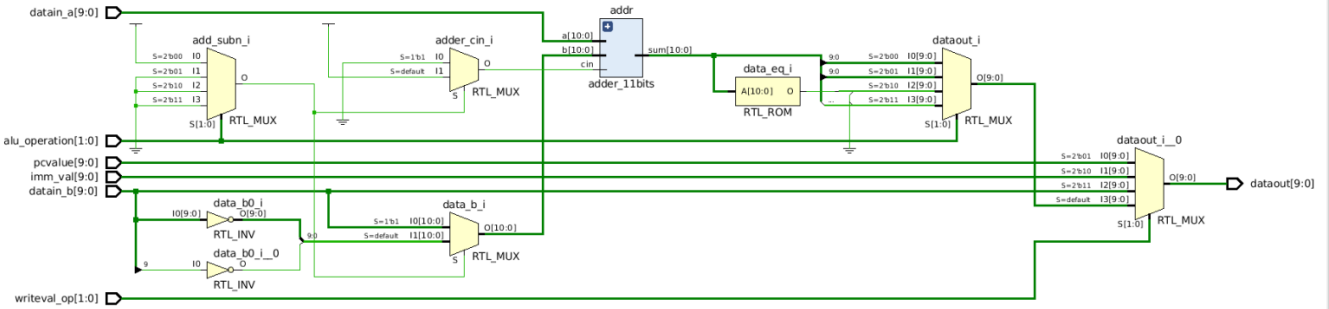


Figure 4 - ALU Schematic

c. *Timing*

Table 3 - ALU Pathway Timing

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	∞	8	7	21	alu_operation[1]	dataout[0]	8.644	3.183	5.461	∞	input port clock		
Path 2	∞	8	7	21	alu_operation[1]	dataout[9]	8.545	3.228	5.316	∞	input port clock		
Path 3	∞	8	7	21	alu_operation[1]	dataout[8]	8.521	3.215	5.306	∞	input port clock		
Path 4	∞	6	5	21	alu_operation[1]	dataout[4]	8.215	3.208	5.007	∞	input port clock		
Path 5	∞	6	5	21	alu_operation[1]	dataout[5]	8.074	3.241	4.833	∞	input port clock		
Path 6	∞	7	6	21	alu_operation[1]	dataout[6]	7.887	3.193	4.695	∞	input port clock		
Path 7	∞	7	6	21	alu_operation[1]	dataout[7]	7.820	3.172	4.648	∞	input port clock		
Path 8	∞	5	4	21	alu_operation[1]	dataout[3]	7.430	3.071	4.359	∞	input port clock		
Path 9	∞	5	4	21	alu_operation[1]	dataout[2]	7.412	3.110	4.301	∞	input port clock		
Path 10	∞	5	4	21	alu_operation[1]	dataout[1]	6.905	3.110	3.796	∞	input port clock		

The longest pathway's time on the old design was 7.842ns. Even though this design has a longer pathway, it is better because it can complete the its function within the CPU.

III. Register Unit

a. *Modifications*

First, the eight MUXs used to determine the enable output were eliminated from this design as requested. Second, both $t0 and $ra are outputs of the register file now to permit the realizations of the je and the ret instructions. Finally, the ldst_en was added to control the output of reg2_out. This allows for the addressing mode described in the ISA for load and store instructions.

b. *Schematic*

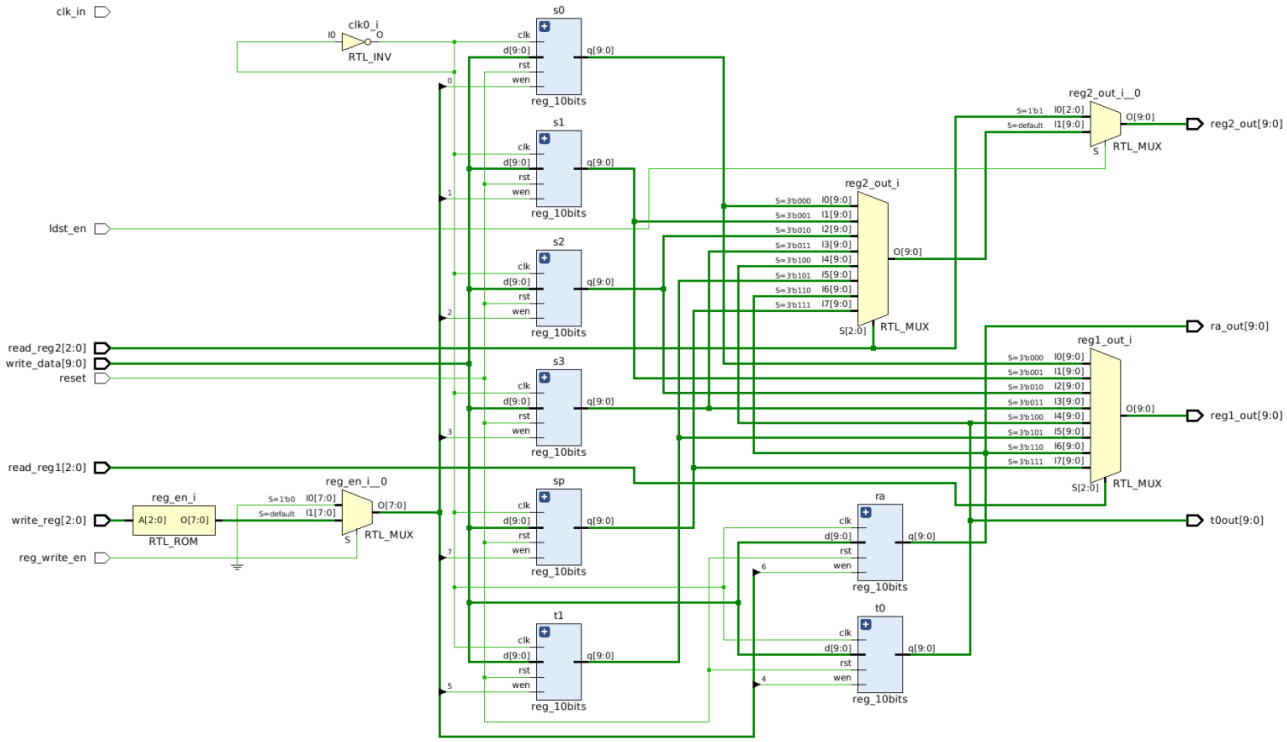


Table 4 - Register File Schematic

c. *Timing*

Table 5 - Register File Timing

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	∞	4	2	20	read_reg1[1]	reg1_out[3]	6.778	3.154	3.624	∞	input port clock		
Path 2	∞	4	2	20	read_reg1[1]	reg1_out[7]	6.637	3.151	3.487	∞	input port clock		
Path 3	∞	4	2	20	read_reg1[1]	reg1_out[8]	6.567	3.140	3.427	∞	input port clock		
Path 4	∞	4	2	20	read_reg1[1]	reg1_out[4]	6.534	3.130	3.405	∞	input port clock		
Path 5	∞	4	2	20	read_reg1[1]	reg1_out[1]	6.530	3.115	3.415	∞	input port clock		
Path 6	∞	4	2	20	read_reg1[1]	reg1_out[0]	6.510	3.118	3.392	∞	input port clock		
Path 7	∞	4	2	20	read_reg1[0]	reg1_out[6]	6.499	3.131	3.368	∞	input port clock		
Path 8	∞	4	2	20	read_reg1[0]	reg1_out[9]	6.493	3.135	3.358	∞	input port clock		
Path 9	∞	4	2	20	read_reg1[1]	reg1_out[5]	6.451	3.134	3.317	∞	input port clock		
Path 10	∞	4	3	21	read_reg2[1]	reg2_out[8]	6.138	3.052	3.086	∞	input port clock		

The old time was 5.958ns whereas this unit is now 6.778ns.

Pieced Together

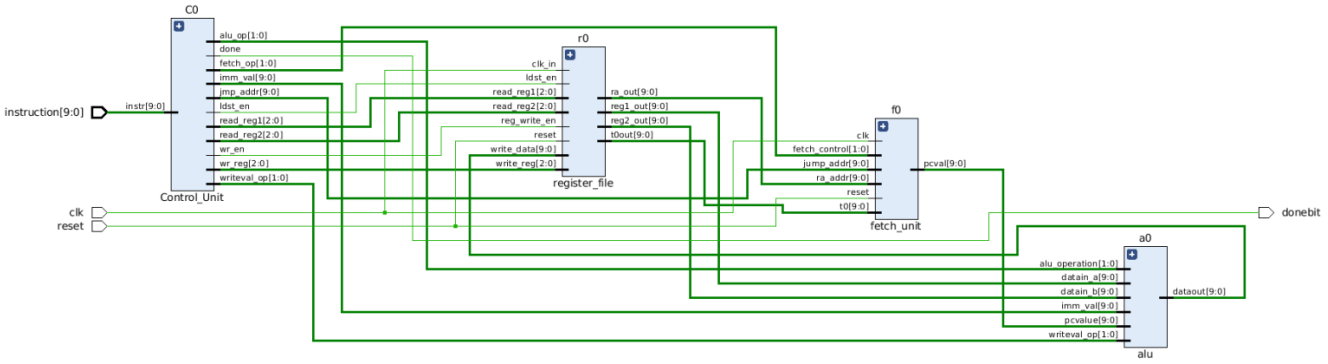


Figure 5 - CPU

I. Design

The only inputs to the above CPU is the instructions, clock, and reset. Once a program is complete, when the halt instruction is called, the program stops moving the program counter and outputs one to the donebit.

a. Timing

Table 6 - CPU Timing

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	∞	3	2	1	instruction[9]	donebit	4.866	2.935	1.931	∞	input port clock		

The total delay was 4.866 ns. Therefore, during testing, we had a clock period of 5ns with the clock alternating on and off every 2.5 ns. In order for the system to work correctly, the register file needed to have double the frequency of the clock input.

b. Testing

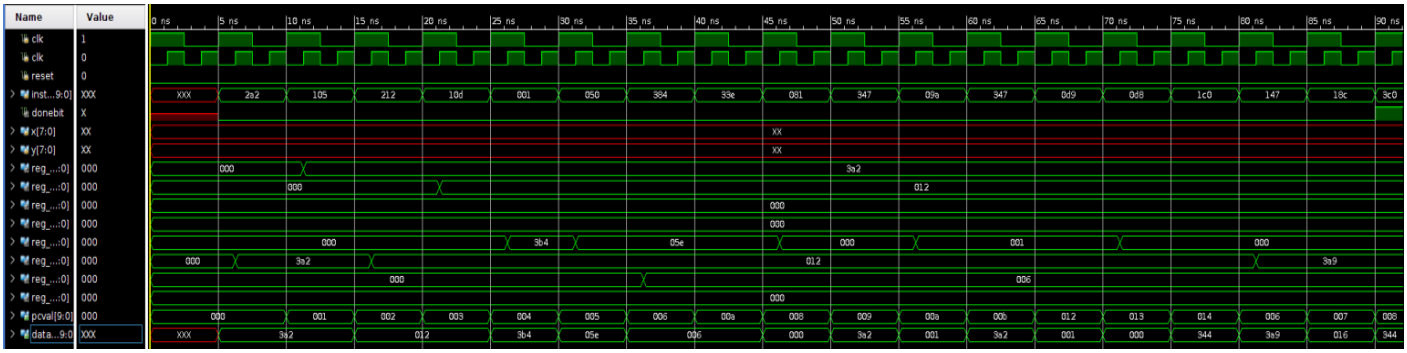


Figure 6 - Testing Results

All instructions behaved according to their purpose.

```
#5 instruction = {2'b10, 8'ha2};    //ldi example
#5 instruction = 10'b0100000101;    //mov example
#5 instruction = {2'b10, 8'h12};    //ldi example
#5 instruction = 10'b0100001101;    //move example
#5 instruction = 10'b0000000001;    //add example
#5 instruction = 10'b0001010000;    //sub example
#5 instruction = 10'b1110000100;    //jal example
#5 instruction = 10'b1100111110;    //j example
#5 instruction = 10'b0010000001;    //cmp
#5 instruction = 10'b1101000111;    //je with $t0 = false
#5 instruction = 10'b0010011010;    //cmp
#5 instruction = 10'b1101000111;    //je with $t0 = true
#5 instruction = 10'b0011011001;    //slt examples
#5 instruction = 10'b0011011000;
#5 instruction = 10'b0111000000;    //ret example
#5 instruction = 10'b0101000111;    //load example
#5 instruction = 10'b0110001100;    //str example
#5 instruction = 10'b1111000000;    //halt
```

Figure 7 - Instruction Input

F = X * Y – 4

I. Simulation Code

```
`timescale 1ns / 1ps
module sim_cpu;

        reg clk, reset;
        reg [9:0] instruction;
        wire donebit;

        reg [7:0] x,y;
        reg [1:0] case0;

        cpu nomem_cpu(clk, instruction, reset, donebit);

        always #2.5 begin clk = ~clk; end

        initial begin
        #0 begin
          clk = 1;
          reset = 0;
          case0 = 0;
          x = 8'bVariable;
          y = 8'bVariable;
        end

                                                                    //LINE
        #5  instruction = {2'b10, x};              //ldi x          0
        #5  instruction = 10'b0100000101;          //mov $s0, $t1   1
        #5  instruction = {2'b10, y};              //ldi y          2
        #5  instruction = 10'b0100001101;          //mov $s1, $t1   3
        #5  instruction = 10'b1000000000;          //ldi 0          4
//check if x = 0
        #5  instruction = 10'b0010000101;          //cmp $s0, $t1   5
        #5  instruction = {4'b1101,6'd24};         //je  end        6
        if(x!=0) begin
//check if y = 0
        #5  instruction = 10'b0010001101;          //cmp $s1, $t1   7
        #5  instruction = {4'b1101,6'd22};         //je  end        8

        if(y!=0) begin
//check if x > 0
        #5  instruction = 10'b0011101000;          //slt $t1, $s0   9
        #5  instruction = {4'b1101,6'd12};         //je  positiveX  10
```

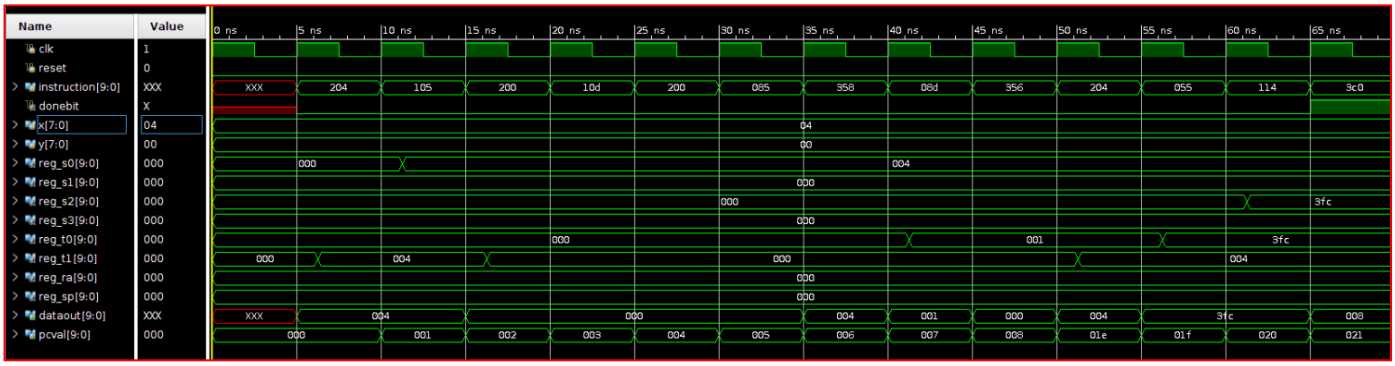
```
if (x[7]==1) begin
        x=~x+1;
        #5  instruction = 10'b0001101000;       //sub $t1, $s0      11
        #5  instruction = 10'b0100000100;       //mov $so, $t0      12
//check if y > 0
        #5  instruction = 10'b0011101001;       //slt $t1, $s1      13
        #5  instruction = {4'b1101, 6'd5};      //je  positiveY1    14
if (y[7]==1) begin
        #5  instruction = 10'b0001101001;       //sub $t1, $s1      15
        #5  instruction = 10'b0100001100;       //mov $s1, $t0      16
        #5  instruction = {4'b1110,6'd19};      //jal mul           17
case0 = 0;
end
//positiveY1:
        #5  instruction = {4'b1110,6'd17};      //jal mul           19
case0 = 1;
end
//positiveX:
        #5  instruction = 10'b0011101001;       //slt $t1, $s1      23
        #5  instruction = {4'b1101,6'd77};      //je  positiveY2    24

if(y[7]==1) begin
        #5  instruction = 10'b0001101001;       //sub $t1, $s1      25
        #5  instruction = 10'b0100001100;       //mov $s1, $t0      26
        #5  instruction = {4'b1110,6'd9};       //jal mul           27
case0 = 2;
end
//positiveY2:
        #5  instruction = {4'b1110,6'd5};       //jal mul           31



//mul
        #5  instruction = 10'b1000000000;       //ldi 0             36
        #5  instruction = 10'b0011101000;       //slt $t1, $s0      37
        #5  instruction = {4'b1101,6'd2};       //je loop           38

while(x>0) begin
//loop
        #5  instruction = 10'b0000010001;       //add $s2, $s1      40
        #5  instruction = 10'b0100010100;       //mov $s2, $t0      41
        #5  instruction = 10'b1000000001;       //ldi 1             42
        #5  instruction = 10'b0001000101;       //sub $s0, $t1      43
        #5  instruction = 10'b0100000100;       //mov $s0, $t0      44
        #5  instruction = 10'b1101110111;       //j   mul           45
//mul
```

```
                #5  instruction = 10'b1000000000;       //ldi 0             36
                #5  instruction = 10'b0011101000;       //slt $t1, $s0      37
                #5  instruction = {4'b1101,6'd2};       //je loop           38
        x=x-1;
                end

                #5  instruction = 10'b0111000000;       //ret               39
    case(case0)
          0: begin #5  instruction = {4'b1100,6'd14};   //j   end           18
          1: begin
                #5  instruction = 10'b0001101010;       //sub $t1, $s2      20
                #5  instruction = 10'b0100010100;       //mov $s2, $t1      21
                #5  instruction = {4'b1100,6'd10};      //j   end           22
          end
       2: begin
                #5  instruction = 10'b0001101010;       //sub $t1, $s2      28
                #5  instruction = 10'b0100010100;       //mov $s2, $t1      29
                #5  instruction = {4'b1100,6'd2};       //j   end           30
        end
    endcase

    end end

    //end
                #5  instruction = 10'b1000000100;       //ldi 4             32
                #5  instruction = 10'b0001010101;       //sub $s2, $t1      33
                #5  instruction = 10'b0100010100;       //mov $s2, $t0      34
                #5  instruction = 10'b1111000000;       //halt              35
    end

endmodule
```

Note: The line numbers on the far right indicate their ordering in memory. They are out of order for the sake of the simulation.

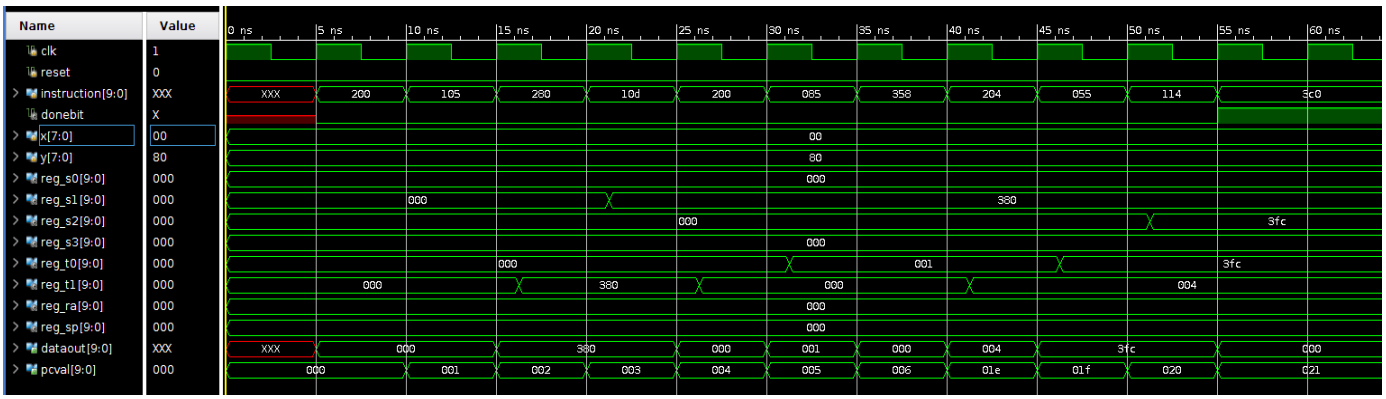
II. Simulation Results (F stored in $s2)

a. Y = 0



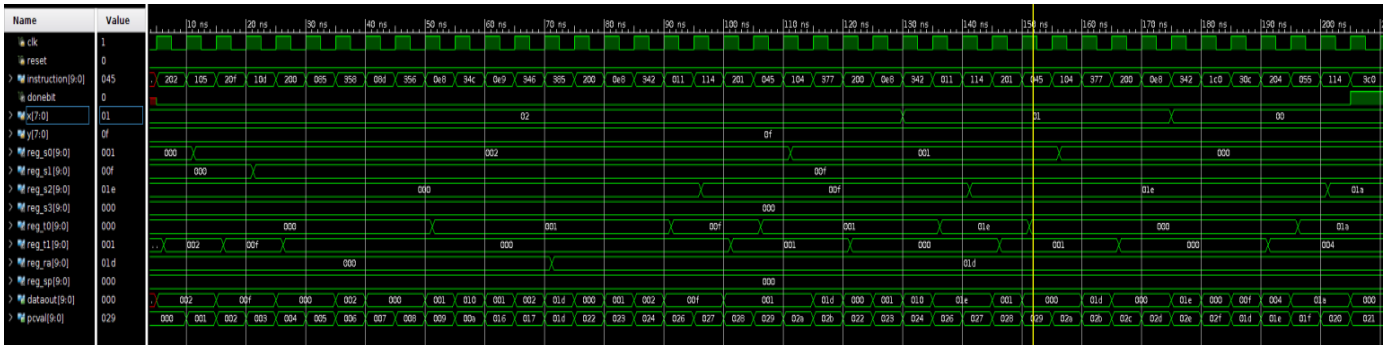
$s2 @ donebit = 1 is 0x3fc or -4

b. X = 0



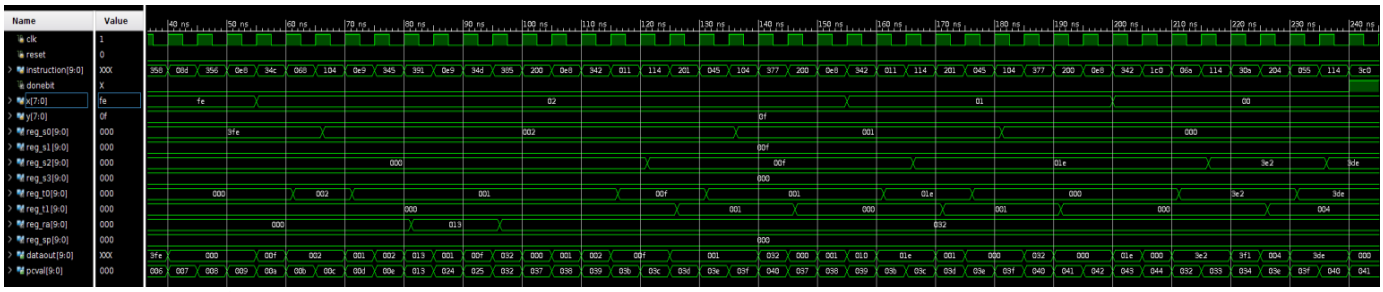
$s2 @ donebit = 1 is 0x3fc or -4

c. X, Y > 0



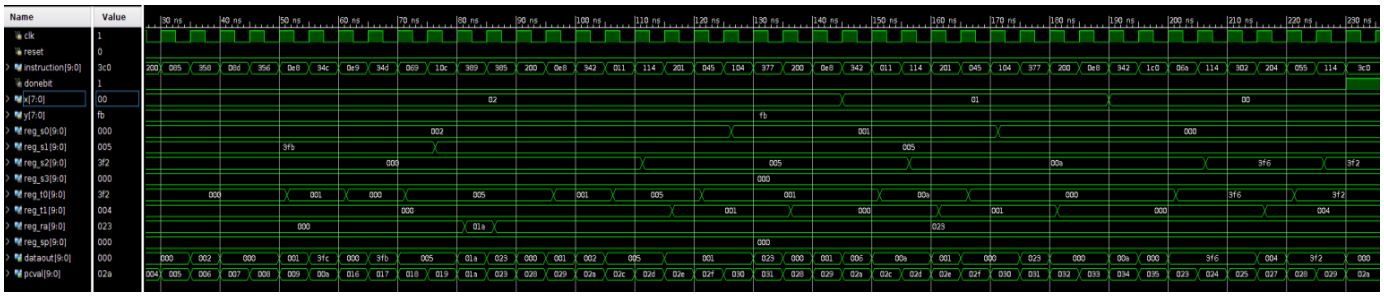
$s2 @ donebit = 0h01a = 26. @t=0: X = 2, Y = 15. 2*15-4 = 26

d. X < 0, Y > 0



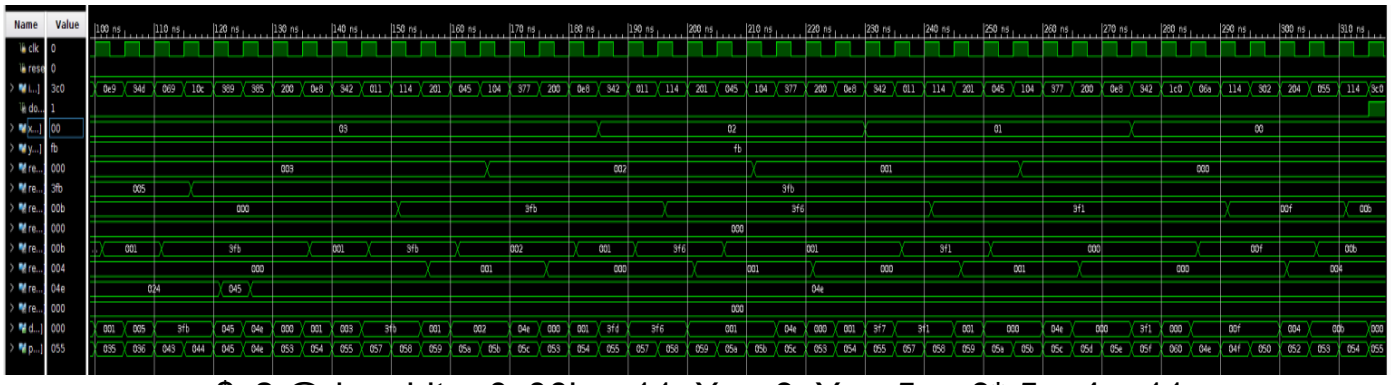
$s2 @ donebit = 1 is 0x3de = -34. X = -2, Y = 15 .:-2*15-4 = -34

e. X > 0, Y < 0



$s2 @ donebit = 0x3f2 = -14. X = 2, Y = -5 .: 2*-5 – 4 = -14

f. X < 0, Y < 0



$s2 @donebit = 0x00b = 11. X = -3, Y = -5 .: -3*-5 – 4 = 11