PES UNIVERSITY, BENGALURU

FINAL SEMESTER ASSESSMENT (FSA) – B.TECH. (CSE) – III SEM

SESSION: AUGUST – DECEMBER, 2018

UE17CS206 – DIGITAL DESIGN & COMPUTER ORGANIZATION LABORATORY

Project Report

on

" Implement shift – left , shift – right and SLT operations for the existing alu & register "

Submitted by

VINAYAKA M HEGDE	PES1201701600
P SUDHAMSHU RAO	PES1201701634

Name of the Examiners	**Signature with Date**
1._____	_____
2._____	_____
3._____	_____
4._____	_____

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

PES UNIVERSITY

(ESTABLISHED UNDER KARNATAKA ACT NO. 16 OF 2013)

100 FEET RING ROAD, BENGALURU – 560 085, KARNATAKA, INDIA

PROBLEM DESCRIPTION

Our Code has four files :

1. reg_alu.v
2. alu.v
3. mylib.v
4. tb_reg_alu.v

1. reg_alu.v :

This file mainly accounts for reading data from the registers and writing data to the registers.
The register file has two read ports and a write port. Each read port consists of an address line of 3 bit and the read output d_out 16 bits . The write port consists of address line of 3 bits and write input d_in of 16 bits to the register only when wr=1.

Modules used :

a. reg_16 : This module is used to construct one 16-bit register. This module instantiates the dfrl module 16-times since 16-bit of data is to be stored.

b. reg16_8 : Since reg_16 module accounts for one 16-bit register, it has to be instantiated 8 times to construct 8 16-bit registers. Therefore, this module instantiates reg_16 module 8 times.

c. reg_file : This is the main module used to read/write data from/to the registers. This consists of inputs rd_addr_a , rd_addr_b , wr_addr ,d_in ,d_out_a and d_out_b .
Based on the addresses rd_addr_a and rd_addr_b, the data is read from the registers and based on wr_addr, the data is written to the registers. This module instantiates the reg16_8 module once to construct the register.

d. reg_alu : This module either stores the data-in provided or the result of the ALU to one of the 8 registers. This consists of a select line called as “sel”. When sel=0, data-in is written to the register specified by wr_addr. And when sel=1, the result of the ALU operation performed as d_out_a and d_out_b as inputs is stored in the register specified by wr_addr.

2. alu.v :

This file performs many ALU operations like ADD,SUB,AND,OR,SHIFT,SLT on the inputs given by the reg_alu module. This takes in two 16-bit data as input and performs operations on them and returns one 16-bit output.

Modules used :

a. fa : This module is the Full Adder module which just adds A,B,Cin and returns output sum. Sum is computed by A xor B xor Cin and Carry-out is computed by A*B+B*Cin+Cin*A.

b. addsub : This module mainly decides whether addition/subtraction is to be performed. It can decide that by doing B xor op[0]. If the result is op[0]=0, then B is passed else 2's complement of B is passed to fa module. Therefore, this instantiates the fa module once.

c. others : This takes two 16-bit data as inputs and slices the inputs 16 times. And it performs all other operations like AND,OR,ADD and SUB.

d. shift : This module takes one 16-bit data as input and performs left/right shifts on it. The shift_opcode of 4 bits denotes the shift-amount i.e., the number of bits it has to shift.

It consists of two control lines :

- sft_sel :

To select between left and right shifts.
sft_sel=0 indicates left-shift and sft_sel=1 indicates right-shift.

- ryt_sft_sel :

To select between logical and arithmetic right shifts.

ryt_sft_sel=0 indicates logical right shift and ryt_sft_sel indicates arithmetic right-shift.
Here, we use two mux: mux1 and mux2. mux1 selects the output with shift-right logical and shift-right arithmetic as input lines based on ryt_sft_sel. And mux2 selects between output of mux1 and shift left based on sft_sel. The output of mux2 is the final output of the Shift block.

Since the data is 16-bit, the maximum number of times it can be shifted is only 15. Therefore, we need 4-bit shift_opcode. For each of the sll, srl and sra; we require 16 16:1 Multiplexer because we select each bit out of 16 bits and this must be done for 16-times.

e. sll : It takes in 3-bit opcode and 16-bit data as the input and shifts left by sft_opcode bits and provides the output.

f. srl : It takes in 3-bit opcode and 16-bit data as the input and performs logical shift right by sft_opcode bits and provides the output.

g. sra : It takes in 3-bit opcode and 16-bit data as the input and performs arithmetic shift right by sft_opcode bits and provides the output.

h. alu : This module consists of two 16-bit data as input and provides one 16-bit data as output.

It has two control lines :

- main_sel :

This select line is used to select between the shift module and the others.
If main_sel=0; the output of others will be selected and if main_sel=1; the output of the Shift module will be selected.

- slt_sel :

This select line is used to select between the set less than and other operations. If slt_sel=0; the output of others module will be selected and if slt_sel=1; the output of the SLT module will be selected.

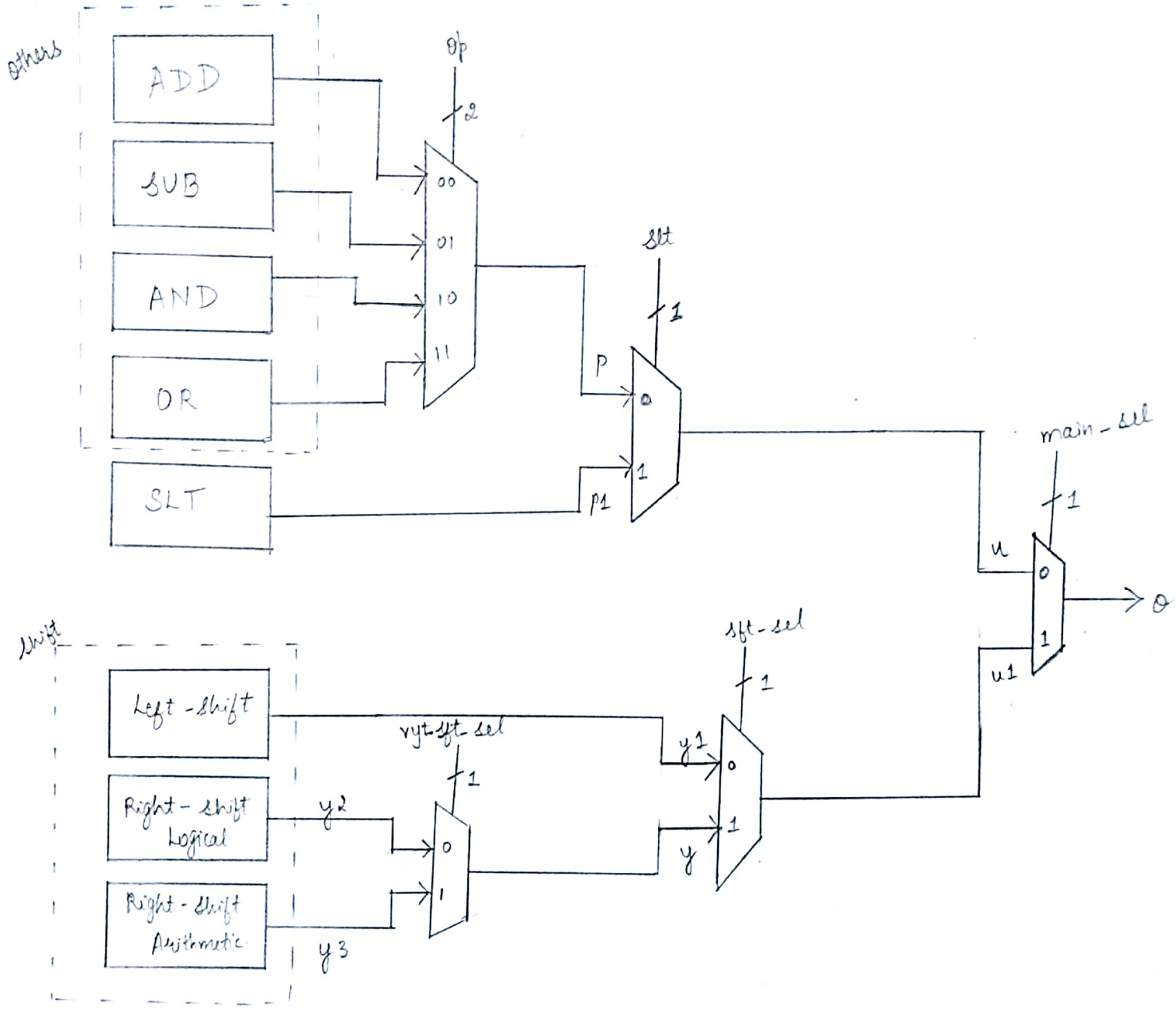
i. slt : This module takes two inputs A and B and computes A-B by passing opcode 01 to the existing ALU to perform ALU subtraction. Then the Most significant bit of A-B is checked. If it is 1, slt flag is high. Else if it is zero, slt flag will be set to low.

3. mylib.v :

This file consists of all the basic structural modules. In this program, we used the below modules from the file:
and2,or2,or3,mux2,mux8,mux16 and demux 8.

4. tb_reg_alu.v :

This is a test-bench file which consists of 29 inputs and each input is given in the form of a test_vector. Each input is of size 37 bits. The first 8 inputs demonstrate the register store part, the next 8 inputs demonstrate the left-shift part, the next 8 for right shift part and last four for slt part.



```
module fa (input wire i0, i1, cin, output wire sum, cout);
wire w0,w1,w2;
xor3 x1(i0,i1,cin,sum);
and2 A1(i0,i1,w0);
and2 A2(i1,cin,w1);
and2 A3(cin,i0,w2);
or3 o1(w0,w1,w2,cout);
endmodule

module addsub (input wire addsub, i0, i1, cin, output wire sumdiff,
cout);
wire w4;
xor2 x3(i1,addsub,w4);
fa d(i0,w4,cin,sumdiff,cout);
endmodule

module alu_slice (input wire [1:0] op, input wire i0, i1, cin, output
wire o, cout);
wire c1,c2,mo1,mo2,o1;
addsub a1(op[0],i0,i1,cin,o1,cout);
and2 A4(i0,i1,c1);
or2 o2(i0,i1,c2);
mux2 m1(c1,c2,op[0],mo1);
mux2 m2(o1,mo1,op[1],o);
endmodule

module shift(input wire sft_sel,ryt_sft_sel,input wire[3:0]op ,input
wire[15:0]a,output wire [15:0]o);
wire [15:0]y,y1,y2,y3;
sll S4(op,a,y1);
srl S5(op,a,y2);
sra s6(op,a,y3);
assign y= (ryt_sft_sel==1'b0)? y2 : y3;
assign o= (sft_sel==1'b0)? y1 : y;
endmodule

module sll(input wire [3:0] op,input wire [15:0]a, output wire [15:0]y);
mux16
A({a[0],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,
1'b0,1'b0},op[0],op[1],op[2],op[3],y[0]);
mux16
A1({a[1],a[0],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0
,1'b0,1'b0},op[0],op[1],op[2],op[3],y[1]);
mux16
A2({a[2],a[1],a[0],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0
,1'b0,1'b0},op[0],op[1],op[2],op[3],y[2]);
mux16
A3({a[3],a[2],a[1],a[0],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0
,1'b0,1'b0},op[0],op[1],op[2],op[3],y[3]);
mux16
A4({a[4],a[3],a[2],a[1],a[0],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0
,1'b0,1'b0},op[0],op[1],op[2],op[3],y[4]);
mux16
A5({a[5],a[4],a[3],a[2],a[1],a[0],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0
,1'b0,1'b0},op[0],op[1],op[2],op[3],y[5]);
mux16
A6({a[6],a[5],a[4],a[3],a[2],a[1],a[0],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0
,1'b0,1'b0},op[0],op[1],op[2],op[3],y[6]);
```

```
mux16
A7({a[7],a[6],a[5],a[4],a[3],a[2],a[1],a[0],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0
,1'b0,1'b0},op[0],op[1],op[2],op[3],y[7]);
mux16
A8({a[8],a[7],a[6],a[5],a[4],a[3],a[2],a[1],a[0],1'b0,1'b0,1'b0,1'b0,1'b0
,1'b0,1'b0},op[0],op[1],op[2],op[3],y[8]);
mux16
A9({a[9],a[8],a[7],a[6],a[5],a[4],a[3],a[2],a[1],a[0],1'b0,1'b0,1'b0,1'b0
,1'b0,1'b0},op[0],op[1],op[2],op[3],y[9]);
mux16
A10({a[10],a[9],a[8],a[7],a[6],a[5],a[4],a[3],a[2],a[1],a[0],1'b0,1'b0,1'
b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y[10]);
mux16
A11({a[11],a[10],a[9],a[8],a[7],a[6],a[5],a[4],a[3],a[2],a[1],a[0],1'b0,1
'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y[11]);
mux16
A12({a[12],a[11],a[10],a[9],a[8],a[7],a[6],a[5],a[4],a[3],a[2],a[1],a[0],
1'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y[12]);
mux16
A13({a[13],a[12],a[11],a[10],a[9],a[8],a[7],a[6],a[5],a[4],a[3],a[2],a[1]
,a[0],1'b0,1'b0},op[0],op[1],op[2],op[3],y[13]);
mux16
A14({a[14],a[13],a[12],a[11],a[10],a[9],a[8],a[7],a[6],a[5],a[4],a[3],a[2
],a[1],a[0],1'b0},op[0],op[1],op[2],op[3],y[14]);
mux16
A15({a[15],a[14],a[13],a[12],a[11],a[10],a[9],a[8],a[7],a[6],a[5],a[4],a[
3],a[2],a[1],a[0]},op[0],op[1],op[2],op[3],y[15]);
endmodule

module srl(input wire [3:0] op,input wire [15:0]a1, output wire
[15:0]y1);
mux16 B
({a1[0],a1[1],a1[2],a1[3],a1[4],a1[5],a1[6],a1[7],a1[8],a1[9],a1[10],a1[1
1],a1[12],a1[13],a1[14],a1[15]},op[0],op[1],op[2],op[3],y1[0]);
mux16
B1({a1[1],a1[2],a1[3],a1[4],a1[5],a1[6],a1[7],a1[8],a1[9],a1[10],a1[11],a
1[12],a1[13],a1[14],a1[15],1'b0},op[0],op[1],op[2],op[3],y1[1]);
mux16
B2({a1[2],a1[3],a1[4],a1[5],a1[6],a1[7],a1[8],a1[9],a1[10],a1[11],a1[12],
a1[13],a1[14],a1[15],1'b0,1'b0},op[0],op[1],op[2],op[3],y1[2]);
mux16
B3({a1[3],a1[4],a1[5],a1[6],a1[7],a1[8],a1[9],a1[10],a1[11],a1[12],a1[13]
,a1[14],a1[15],1'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y1[3]);
mux16
B4({a1[4],a1[5],a1[6],a1[7],a1[8],a1[9],a1[10],a1[11],a1[12],a1[13],a1[14
],a1[15],1'b0,1'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y1[4]);
mux16
B5({a1[5],a1[6],a1[7],a1[8],a1[9],a1[10],a1[11],a1[12],a1[13],a1[14],a1[1
5],1'b0,1'b0,1'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y1[5]);
mux16
B6({a1[6],a1[7],a1[8],a1[9],a1[10],a1[11],a1[12],a1[13],a1[14],a1[15],1'b
0,1'b0,1'b0,1'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y1[6]);
mux16
B7({a1[7],a1[8],a1[9],a1[10],a1[11],a1[12],a1[13],a1[14],a1[15],1'b0,1'b0
,1'b0,1'b0,1'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y1[7]);
mux16
B8({a1[8],a1[9],a1[10],a1[11],a1[12],a1[13],a1[14],a1[15],1'b0,1'b0,1'b0,
1'b0,1'b0,1'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y1[8]);
```

```
mux16
B9({a1[9],a1[10],a1[11],a1[12],a1[13],a1[14],a1[15],1'b0,1'b0,1'b0,1'b0,1
'b0,1'b0,1'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y1[9]);
mux16
B10({a1[10],a1[11],a1[12],a1[13],a1[14],a1[15],1'b0,1'b0,1'b0,1'b0,1'b0,1
'b0,1'b0,1'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y1[10]);
mux16
B11({a1[11],a1[12],a1[13],a1[14],a1[15],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b
0,1'b0,1'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y1[11]);
mux16
B12({a1[12],a1[13],a1[14],a1[15],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,
1'b0,1'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y1[12]);
mux16
B13({a1[13],a1[14],a1[15],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'
b0,1'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y1[13]);
mux16
B14({a1[14],a1[15],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0
,1'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y1[14]);
mux16
B15({a1[15],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1
'b0,1'b0,1'b0},op[0],op[1],op[2],op[3],y1[15]);
endmodule

module sra(input wire [3:0] op,input wire [15:0]a, output wire [15:0]t);
mux16
C({a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12],a[
13],a[14],a[15]},op[0],op[1],op[2],op[3],t[0]);
mux16
C1({a[1],a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12],a[13],
a[14],a[15],a[15]},op[0],op[1],op[2],op[3],t[1]);
mux16
C2({a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12],a[13],a[14]
,a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[2]);
mux16
C3({a[3],a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12],a[13],a[14],a[15
],a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[3]);
mux16
C4({a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12],a[13],a[14],a[15],a[1
5],a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[4]);
mux16
C5({a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12],a[13],a[14],a[15],a[15],a[
15],a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[5]);
mux16
C6({a[6],a[7],a[8],a[9],a[10],a[11],a[12],a[13],a[14],a[15],a[15],a[15],a
[15],a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[6]);
mux16
C7({a[7],a[8],a[9],a[10],a[11],a[12],a[13],a[14],a[15],a[15],a[15],a[15],
a[15],a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[7]);
mux16
C8({a[8],a[9],a[10],a[11],a[12],a[13],a[14],a[15],a[15],a[15],a[15],a[15]
,a[15],a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[8]);
mux16
C9({a[9],a[10],a[11],a[12],a[13],a[14],a[15],a[15],a[15],a[15],a[15],a[15
],a[15],a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[9]);
mux16
C10({a[10],a[11],a[12],a[13],a[14],a[15],a[15],a[15],a[15],a[15],a[15],a[
15],a[15],a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[10]);
```

```
mux16
C11({a[11],a[12],a[13],a[14],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[
15],a[15],a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[11]);
mux16
C12({a[12],a[13],a[14],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[
15],a[15],a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[12]);
mux16
C13({a[13],a[14],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[
15],a[15],a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[13]);
mux16
C14({a[14],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[
15],a[15],a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[14]);
mux16
C15({a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[15],a[
15],a[15],a[15],a[15],a[15]},op[0],op[1],op[2],op[3],t[15]);
endmodule

module slt(input wire [15:0] i0, i1,output wire [15:0] r);
output wire [15:0]temp;
others O9(2'b01,i0,i1,temp);
assign r= (temp[15]==1'b1)?16'd1:16'd0;
endmodule

module others(input wire [1:0]op, input wire [15:0]i0,i1, output
wire[15:0]o);
output wire [14:0]c;
alu_slice S1(op,i0[0],i1[0],op[0],o[0],c[0]);
alu_slice S2(op,i0[1],i1[1],c[0],o[1],c[1]);
alu_slice S3(op,i0[2],i1[2],c[1],o[2],c[2]);
alu_slice S4(op,i0[3],i1[3],c[2],o[3],c[3]);
alu_slice S5(op,i0[4],i1[4],c[3],o[4],c[4]);
alu_slice S6(op,i0[5],i1[5],c[4],o[5],c[5]);
alu_slice S7(op,i0[6],i1[6],c[5],o[6],c[6]);
alu_slice S8(op,i0[7],i1[7],c[6],o[7],c[7]);
alu_slice S9(op,i0[8],i1[8],c[7],o[8],c[8]);
alu_slice S10(op,i0[9],i1[9],c[8],o[9],c[9]);
alu_slice S11(op,i0[10],i1[10],c[9],o[10],c[10]);
alu_slice S12(op,i0[11],i1[11],c[10],o[11],c[11]);
alu_slice S13(op,i0[12],i1[12],c[11],o[12],c[12]);
alu_slice S14(op,i0[13],i1[13],c[12],o[13],c[13]);
alu_slice S15(op,i0[14],i1[14],c[13],o[14],c[14]);
alu_slice S16(op,i0[15],i1[15],c[14],o[15],cout);
endmodule


module alu (input wire slt_sel,main_sel,sft_sel,ryt_sft_sel,input wire
[1:0] op,input wire [3:0] sft_op, input wire [15:0] i0, i1,output wire
[15:0] o, output wire cout);
wire [15:0]u,u1,p,p1;
others O(op,i0,i1,p);
slt S0(i0,i1,p1);
assign u=(slt_sel==1'b0)?p:p1;
shift S9(sft_sel,ryt_sft_sel,sft_op,i0,u1);
assign o =(main_sel == 1'b0)? u : u1;
endmodule
```

```
`include "mylib.v"
`include "alu.v"
// Write code for modules you need here
module reg_16(input wire clk,reset,load,input wire [15:0] in, output wire[15:0]
out);
      dfrl d0(.clk(clk),.reset(reset),.load(load),.in(in[0]),.out(out[0]));
      dfrl d1(.clk(clk),.reset(reset),.load(load),.in(in[1]),.out(out[1]));
      dfrl d2(.clk(clk),.reset(reset),.load(load),.in(in[2]),.out(out[2]));
      dfrl d3(.clk(clk),.reset(reset),.load(load),.in(in[3]),.out(out[3]));
      dfrl d4(.clk(clk),.reset(reset),.load(load),.in(in[4]),.out(out[4]));
      dfrl d5(.clk(clk),.reset(reset),.load(load),.in(in[5]),.out(out[5]));
      dfrl d6(.clk(clk),.reset(reset),.load(load),.in(in[6]),.out(out[6]));
      dfrl d7(.clk(clk),.reset(reset),.load(load),.in(in[7]),.out(out[7]));
      dfrl d8(.clk(clk),.reset(reset),.load(load),.in(in[8]),.out(out[8]));
      dfrl d9(.clk(clk),.reset(reset),.load(load),.in(in[9]),.out(out[9]));
      dfrl d10(.clk(clk),.reset(reset),.load(load),.in(in[10]),.out(out[10]));
      dfrl d11(.clk(clk),.reset(reset),.load(load),.in(in[11]),.out(out[11]));
      dfrl d12(.clk(clk),.reset(reset),.load(load),.in(in[12]),.out(out[12]));
      dfrl d13(.clk(clk),.reset(reset),.load(load),.in(in[13]),.out(out[13]));
      dfrl d14(.clk(clk),.reset(reset),.load(load),.in(in[14]),.out(out[14]));
      dfrl d15(.clk(clk),.reset(reset),.load(load),.in(in[15]),.out(out[15]));
endmodule

module reg16_8(input wire clk,reset,input wire [0:7] load,input wire [15:0]
u,output wire [15:0] q0,q1,q2,q3,q4,q5,q6,q7);
      reg_16 r1(clk,reset,load[0],u,q0);
      reg_16 r2(clk,reset,load[1],u,q1);
      reg_16 r3(clk,reset,load[2],u,q2);
      reg_16 r4(clk,reset,load[3],u,q3);
      reg_16 r5(clk,reset,load[4],u,q4);
      reg_16 r6(clk,reset,load[5],u,q5);
      reg_16 r7(clk,reset,load[6],u,q6);
      reg_16 r8(clk,reset,load[7],u,q7);
endmodule

module mux8_16(input wire [15:0] i0,i1,i2,i3,i4,i5,i6,i7,input wire [2:0]
rd_addr,output wire [15:0] o);
      mux8
m0({i0[0],i1[0],i2[0],i3[0],i4[0],i5[0],i6[0],i7[0]},rd_addr[0],rd_addr[1],rd_ad
dr[2],o[0]);
      mux8
m1({i0[1],i1[1],i2[1],i3[1],i4[1],i5[1],i6[1],i7[1]},rd_addr[0],rd_addr[1],rd_ad
dr[2],o[1]);
      mux8
m2({i0[2],i1[2],i2[2],i3[2],i4[2],i5[2],i6[2],i7[2]},rd_addr[0],rd_addr[1],rd_ad
dr[2],o[2]);
      mux8
m3({i0[3],i1[3],i2[3],i3[3],i4[3],i5[3],i6[3],i7[3]},rd_addr[0],rd_addr[1],rd_ad
dr[2],o[3]);
      mux8
m4({i0[4],i1[4],i2[4],i3[4],i4[4],i5[4],i6[4],i7[4]},rd_addr[0],rd_addr[1],rd_ad
dr[2],o[4]);
      mux8
m5({i0[5],i1[5],i2[5],i3[5],i4[5],i5[5],i6[5],i7[5]},rd_addr[0],rd_addr[1],rd_ad
dr[2],o[5]);
      mux8
m6({i0[6],i1[6],i2[6],i3[6],i4[6],i5[6],i6[6],i7[6]},rd_addr[0],rd_addr[1],rd_ad
dr[2],o[6]);
      mux8
m7({i0[7],i1[7],i2[7],i3[7],i4[7],i5[7],i6[7],i7[7]},rd_addr[0],rd_addr[1],rd_ad
dr[2],o[7]);
      mux8
m8({i0[8],i1[8],i2[8],i3[8],i4[8],i5[8],i6[8],i7[8]},rd_addr[0],rd_addr[1],rd_ad
dr[2],o[8]);
```

```
      mux8
m9({i0[9],i1[9],i2[9],i3[9],i4[9],i5[9],i6[9],i7[9]},rd_addr[0],rd_addr[1],rd_ad
dr[2],o[9]);
      mux8
m10({i0[10],i1[10],i2[10],i3[10],i4[10],i5[10],i6[10],i7[10]},rd_addr[0],rd_addr
[1],rd_addr[2],o[10]);
      mux8
m11({i0[11],i1[11],i2[11],i3[11],i4[11],i5[11],i6[11],i7[11]},rd_addr[0],rd_addr
[1],rd_addr[2],o[11]);
      mux8
m12({i0[12],i1[12],i2[12],i3[12],i4[12],i5[12],i6[12],i7[12]},rd_addr[0],rd_addr
[1],rd_addr[2],o[12]);
      mux8
m13({i0[13],i1[13],i2[13],i3[13],i4[13],i5[13],i6[13],i7[13]},rd_addr[0],rd_addr
[1],rd_addr[2],o[13]);
      mux8
m14({i0[14],i1[14],i2[14],i3[14],i4[14],i5[14],i6[14],i7[14]},rd_addr[0],rd_addr
[1],rd_addr[2],o[14]);
      mux8
m15({i0[15],i1[15],i2[15],i3[15],i4[15],i5[15],i6[15],i7[15]},rd_addr[0],rd_addr
[1],rd_addr[2],o[15]);
endmodule

module reg_file (input wire clk, reset, wr, input wire [2:0] rd_addr_a,
rd_addr_b, wr_addr, input wire [15:0] d_in, output wire [15:0] d_out_a,
d_out_b);

// Declare wires here
      wire [0:7] load;
      wire[15:0] q0,q1,q2,q3,q4,q5,q6,q7;
// Instantiate modules here
      demux8
d8(.i(wr),.j2(wr_addr[2]),.j1(wr_addr[1]),.j0(wr_addr[0]),.o(load));
      reg16_8 r16(clk,reset,load,d_in,q0,q1,q2,q3,q4,q5,q6,q7);
      mux8_16 m8(q0,q1,q2,q3,q4,q5,q6,q7,rd_addr_a,d_out_a);
      mux8_16 m9(q0,q1,q2,q3,q4,q5,q6,q7,rd_addr_b,d_out_b);
endmodule


module mux2_16(input wire op,input wire [15:0]i0,i1,output wire [15:0]o);
mux2 m1(i0[15],i1[15],op,o[15]);
mux2 m2(i0[14],i1[14],op,o[14]);
mux2 m3(i0[13],i1[13],op,o[13]);
mux2 m4(i0[12],i1[12],op,o[12]);
mux2 m5(i0[11],i1[11],op,o[11]);
mux2 m6(i0[10],i1[10],op,o[10]);
mux2 m7(i0[9],i1[9],op,o[9]);
mux2 m8(i0[8],i1[8],op,o[8]);
mux2 m9(i0[7],i1[7],op,o[7]);
mux2 m10(i0[6],i1[6],op,o[6]);
mux2 m11(i0[5],i1[5],op,o[5]);
mux2 m12(i0[4],i1[4],op,o[4]);
mux2 m13(i0[3],i1[3],op,o[3]);
mux2 m14(i0[2],i1[2],op,o[2]);
mux2 m15(i0[1],i1[1],op,o[1]);
mux2 m16(i0[0],i1[0],op,o[0]);
endmodule

module reg_alu (input wire clk, reset,
slt_sel,sel,main_sel,sft_sel,ryt_sft_sel,wr, input wire [1:0] op,input
wire[3:0]sft_op, input wire [2:0] rd_addr_a,rd_addr_b, wr_addr, input wire
[15:0] d_in, output wire [15:0] d_out_a, d_out_b, output wire cout);
// Declare wires here
wire [15:0] reg_write_ip,alu_wire_op;
```

```
wire cout1;
// Instantiate modules here
reg_file
rf(clk,reset,wr,rd_addr_a,rd_addr_b,wr_addr,reg_write_ip,d_out_a,d_out_b);
mux2_16 m216(sel,d_in,alu_wire_op,reg_write_ip);
alu
A(slt_sel,main_sel,sft_sel,ryt_sft_sel,op,sft_op,d_out_a,d_out_b,alu_wire_op,cou
t1);
dfr d(clk,reset,cout1,cout);
endmodule
```

```
module invert (input wire i, output wire o);
   assign o = !i;
endmodule

module and2 (input wire i0, i1, output wire o);
  assign o = i0 & i1;
endmodule

module or2 (input wire i0, i1, output wire o);
  assign o = i0 | i1;
endmodule

module xor2 (input wire i0, i1, output wire o);
  assign o = i0 ^ i1;
endmodule

module nand2 (input wire i0, i1, output wire o);
   wire t;
   and2 and2_0 (i0, i1, t);
   invert invert_0 (t, o);
endmodule

module nor2 (input wire i0, i1, output wire o);
   wire t;
   or2 or2_0 (i0, i1, t);
   invert invert_0 (t, o);
endmodule

module xnor2 (input wire i0, i1, output wire o);
   wire t;
   xor2 xor2_0 (i0, i1, t);
   invert invert_0 (t, o);
endmodule

module and3 (input wire i0, i1, i2, output wire o);
   wire t;
   and2 and2_0 (i0, i1, t);
   and2 and2_1 (i2, t, o);
endmodule

module or3 (input wire i0, i1, i2, output wire o);
   wire t;
   or2 or2_0 (i0, i1, t);
   or2 or2_1 (i2, t, o);
endmodule

module nor3 (input wire i0, i1, i2, output wire o);
   wire t;
   or2 or2_0 (i0, i1, t);
   nor2 nor2_0 (i2, t, o);
endmodule

module nand3 (input wire i0, i1, i2, output wire o);
   wire t;
   and2 and2_0 (i0, i1, t);
   nand2 nand2_1 (i2, t, o);
endmodule
```

```
module xor3 (input wire i0, i1, i2, output wire o);
   wire t;
   xor2 xor2_0 (i0, i1, t);
   xor2 xor2_1 (i2, t, o);
endmodule

module xnor3 (input wire i0, i1, i2, output wire o);
   wire t;
   xor2 xor2_0 (i0, i1, t);
   xnor2 xnor2_0 (i2, t, o);
endmodule

module mux2 (input wire i0, i1, j, output wire o);
  assign o = (j==0)?i0:i1;
endmodule

module mux4 (input wire [0:3] i, input wire j1, j0, output wire o);
  wire  t0, t1;
  mux2 mux2_0 (i[0], i[1], j1, t0);
  mux2 mux2_1 (i[2], i[3], j1, t1);
  mux2 mux2_2 (t0, t1, j0, o);
endmodule

module mux8 (input wire [0:7] i, input wire j2, j1, j0, output wire o);
  wire  t0, t1;
  mux4 mux4_0 (i[0:3], j2, j1, t0);
  mux4 mux4_1 (i[4:7], j2, j1, t1);
  mux2 mux2_0 (t0, t1, j0, o);
endmodule

module mux16 (input wire[0:15]i, input wire j3,j2,j1,j0, output wire o);
  wire t0,t1;
  mux8 mux8_0 (i[0:7], j3, j2, j1,t0);
  mux8 mux8_1 (i[8:15], j3, j2, j1,t1);
  mux2 mux2_9 (t0,t1,j0,o);

endmodule



module demux2 (input wire i, j, output wire o0, o1);
  assign o0 = (j==0)?i:1'b0;
  assign o1 = (j==1)?i:1'b0;
endmodule

module demux4 (input wire i, j1, j0, output wire [0:3] o);
  wire  t0, t1;
  demux2 demux2_0 (i, j1, t0, t1);
  demux2 demux2_1 (t0, j0, o[0], o[1]);
  demux2 demux2_2 (t1, j0, o[2], o[3]);
endmodule

module demux8 (input wire i, j2, j1, j0, output wire [0:7] o);
  wire  t0, t1;
  demux2 demux2_0 (i, j2, t0, t1);
  demux4 demux4_0 (t0, j1, j0, o[0:3]);
  demux4 demux4_1 (t1, j1, j0, o[4:7]);
endmodule
```

```
module df (input wire clk, in, output wire out);
  reg df_out;
  always@(posedge clk) df_out <= in;
  assign out = df_out;
endmodule

module dfr (input wire clk, reset, in, output wire out);
  wire reset_, df_in;
  invert invert_0 (reset, reset_);
  and2 and2_0 (in, reset_, df_in);
  df df_0 (clk, df_in, out);
endmodule

module dfrl (input wire clk, reset, load, in, output wire out);
  wire _in;
  mux2 mux2_0(out, in, load, _in);
  dfr dfr_1(clk, reset, _in, out);
endmodule
```

```
`timescale 1 ns / 100 ps
`define TESTVECS 29

module tb;
  reg clk, reset, slt_sel,wr, sel,main_sel,sft_sel,ryt_sft_sel;
  reg [1:0] op;
  reg [3:0]sft_op;
  reg [2:0] rd_addr_a, rd_addr_b, wr_addr; reg [15:0] d_in;
  wire [15:0] d_out_a, d_out_b;
  reg [36:0] test_vecs [0:(`TESTVECS-1)];
  integer i;
  initial begin $dumpfile("tb_reg_alu.vcd"); $dumpvars(0,tb); end
  initial begin reset = 1'b1;    #0 reset = 1'b0;    end
  initial clk = 1'b0; always #5 clk =~ clk;
  initial begin
                                      // reg-store
    test_vecs[0][36] = 1'b0;
    test_vecs[0][35] = 1'b0;      test_vecs[0][34] = 1'b1;
test_vecs[0][33] = 1'b0;       test_vecs[0][32] = 1'b0;
    test_vecs[0][31] = 1'b1;      test_vecs[0][30:29] = 2'b00;
test_vecs[0][28:25] = 4'b0000; test_vecs[0][24:22] = 3'o0;
    test_vecs[0][21:19] = 3'ox;   test_vecs[0][18:16] = 3'h0;
test_vecs[0][15:0] = 16'd1023;

    test_vecs[1][36] = 1'b0;
    test_vecs[1][35] = 1'b0;      test_vecs[1][34] = 1'b1;
test_vecs[1][33] = 1'b0;       test_vecs[1][32] = 1'b0;
    test_vecs[1][31] = 1'b1;      test_vecs[1][30:29] = 2'b00;
test_vecs[1][28:25] = 4'b0000; test_vecs[1][24:22] = 3'o0;
    test_vecs[1][21:19] = 3'ox;   test_vecs[1][18:16] = 3'h1;
test_vecs[1][15:0] = 16'd0907;

    test_vecs[2][36] = 1'b0;
    test_vecs[2][35] = 1'b0;      test_vecs[2][34] = 1'b1;
test_vecs[2][33] = 1'b0;        test_vecs[2][32] = 1'b0;
    test_vecs[2][31] = 1'b1;      test_vecs[2][30:29] = 2'b00;
test_vecs[2][28:25] = 4'b0000;  test_vecs[2][24:22] = 3'o1;
    test_vecs[2][21:19] = 3'o7;   test_vecs[2][18:16] = 3'h2;
test_vecs[2][15:0] = 16'd32769;

    test_vecs[3][36] = 1'b0;
    test_vecs[3][35] = 1'b0;      test_vecs[3][34] = 1'b1;
test_vecs[3][33] = 1'b0;        test_vecs[3][32] = 1'b0;
    test_vecs[3][31] = 1'b1;      test_vecs[3][30:29] = 2'b00;
test_vecs[3][28:25] = 4'b0000;  test_vecs[3][24:22] = 3'o2;
    test_vecs[3][21:19] = 3'o5;   test_vecs[3][18:16] = 3'h3;
test_vecs[3][15:0] = 16'd0678;

    test_vecs[4][36] = 1'b0;
    test_vecs[4][35] = 1'b0;      test_vecs[4][34] = 1'b1;
test_vecs[4][33] = 1'b0;         test_vecs[4][32] = 1'b0;
    test_vecs[4][31] = 1'b1;      test_vecs[4][30:29] = 2'b00;
test_vecs[4][28:25] = 4'b0000;   test_vecs[4][24:22] = 3'o3;
    test_vecs[4][21:19] = 3'o5;   test_vecs[4][18:16] = 3'h4;
test_vecs[4][15:0] = 16'd41245;

    test_vecs[5][36] = 1'b0;
```

```
    test_vecs[5][35] = 1'b0;      test_vecs[5][34] = 1'b1;
test_vecs[5][33] = 1'b0;          test_vecs[5][32] = 1'b0;
    test_vecs[5][31] = 1'b1;      test_vecs[5][30:29] = 2'b00;
test_vecs[5][28:25] = 4'b0000;    test_vecs[5][24:22] = 3'o4;
    test_vecs[5][21:19] = 3'o5;   test_vecs[5][18:16] = 3'h5;
test_vecs[5][15:0] = 16'd37119;

    test_vecs[6][36] = 1'b0;
    test_vecs[6][35] = 1'b0;      test_vecs[6][34] = 1'b1;
test_vecs[6][33] = 1'b0;          test_vecs[6][32] = 1'b0;
    test_vecs[6][31] = 1'b1;      test_vecs[6][30:29] = 2'b00;
test_vecs[6][28:25] = 4'b0000;    test_vecs[6][24:22] = 3'o5;
    test_vecs[6][21:19] = 3'o7;   test_vecs[6][18:16] = 3'h6;
test_vecs[6][15:0] = 16'd33809;

    test_vecs[7][36] = 1'b0;
    test_vecs[7][35] = 1'b0;      test_vecs[7][34] = 1'b1;
test_vecs[7][33] = 1'b0;          test_vecs[7][32] = 1'b0;
    test_vecs[7][31] = 1'b1;      test_vecs[7][30:29] = 2'b00;
test_vecs[7][28:25] = 4'b0000;    test_vecs[7][24:22] = 3'o6;
    test_vecs[7][21:19] = 3'o5;   test_vecs[7][18:16] = 3'h7;
test_vecs[7][15:0] = 16'd0101;

                                              //left-shift

    test_vecs[8][36] = 1'b0;
    test_vecs[8][35] = 1'b1;      test_vecs[8][34] = 1'b1;
test_vecs[8][33] = 1'b0;          test_vecs[8][32] = 1'b0;
    test_vecs[8][31] = 1'b1;      test_vecs[8][30:29] = 2'b00;
test_vecs[8][28:25] = 4'd0;       test_vecs[8][24:22] = 3'o2;
    test_vecs[8][21:19] = 3'o5;   test_vecs[8][18:16] = 3'h1;
test_vecs[8][15:0] = 16'd0101;

    test_vecs[9][36] = 1'b0;
    test_vecs[9][35] = 1'b1;      test_vecs[9][34] = 1'b1;
test_vecs[9][33] = 1'b0;          test_vecs[9][32] = 1'b0;
    test_vecs[9][31] = 1'b1;      test_vecs[9][30:29] = 2'b00;
test_vecs[9][28:25] = 4'd1;       test_vecs[9][24:22] = 3'o0;
    test_vecs[9][21:19] = 3'o5;   test_vecs[9][18:16] = 3'h1;
test_vecs[9][15:0] = 16'd0101;

    test_vecs[10][36] = 1'b0;
    test_vecs[10][35] = 1'b1;     test_vecs[10][34] = 1'b1;
test_vecs[10][33] = 1'b0;         test_vecs[10][32] = 1'b0;
    test_vecs[10][31] = 1'b1;     test_vecs[10][30:29] = 2'b00;
test_vecs[10][28:25] = 4'd2;      test_vecs[10][24:22] = 3'o1;
    test_vecs[10][21:19] = 3'o5;  test_vecs[10][18:16] = 3'h1;
test_vecs[10][15:0] = 16'd0101;

    test_vecs[11][36] = 1'b0;
    test_vecs[11][35] = 1'b1;     test_vecs[11][34] = 1'b1;
test_vecs[11][33] = 1'b0;         test_vecs[11][32] = 1'b0;
    test_vecs[11][31] = 1'b1;     test_vecs[11][30:29] = 2'b00;
test_vecs[11][28:25] = 4'd3;      test_vecs[11][24:22] = 3'o2;
    test_vecs[11][21:19] = 3'o5;  test_vecs[11][18:16] = 3'h1;
test_vecs[11][15:0] = 16'd0101;

    test_vecs[12][36] = 1'b0;
```

```
    test_vecs[12][35] = 1'b1;    test_vecs[12][34] = 1'b1;
test_vecs[12][33] = 1'b0;        test_vecs[12][32] = 1'b0;
    test_vecs[12][31] = 1'b1;    test_vecs[12][30:29] = 2'b00;
test_vecs[12][28:25] = 4'd4;     test_vecs[12][24:22] = 3'o3;
    test_vecs[12][21:19] = 3'o5; test_vecs[12][18:16] = 3'h1;
test_vecs[12][15:0] = 16'd0101;

    test_vecs[13][36] = 1'b0;
    test_vecs[13][35] = 1'b1;    test_vecs[13][34] = 1'b1;
test_vecs[13][33] = 1'b0;        test_vecs[13][32] = 1'b0;
    test_vecs[13][31] = 1'b1;    test_vecs[13][30:29] = 2'b00;
test_vecs[13][28:25] = 4'd5;     test_vecs[13][24:22] = 3'o4;
    test_vecs[13][21:19] = 3'o5; test_vecs[13][18:16] = 3'h1;
test_vecs[13][15:0] = 16'd0101;

    test_vecs[14][36] = 1'b0;
    test_vecs[14][35] = 1'b1;    test_vecs[14][34] = 1'b1;
test_vecs[14][33] = 1'b0;        test_vecs[14][32] = 1'b0;
    test_vecs[14][31] = 1'b1;    test_vecs[14][30:29] = 2'b00;
test_vecs[14][28:25] = 4'd6;     test_vecs[14][24:22] = 3'o5;
    test_vecs[14][21:19] = 3'o5; test_vecs[14][18:16] = 3'h1;
test_vecs[14][15:0] = 16'd0101;

    test_vecs[15][36] = 1'b0;
    test_vecs[15][35] = 1'b1;    test_vecs[15][34] = 1'b1;
test_vecs[15][33] = 1'b0;        test_vecs[15][32] = 1'b0;
    test_vecs[15][31] = 1'b1;    test_vecs[15][30:29] = 2'b00;
test_vecs[15][28:25] = 4'd7;     test_vecs[15][24:22] = 3'o6;
    test_vecs[15][21:19] = 3'o5; test_vecs[15][18:16] = 3'h1;
test_vecs[15][15:0] = 16'd0101;

    test_vecs[16][36] = 1'b0;
    test_vecs[16][35] = 1'b1;    test_vecs[16][34] = 1'b1;
test_vecs[16][33] = 1'b0;        test_vecs[16][32] = 1'b0;
    test_vecs[16][31] = 1'b1;    test_vecs[16][30:29] = 2'b00;
test_vecs[16][28:25] = 4'b0010;  test_vecs[16][24:22] = 3'o7;
    test_vecs[16][21:19] = 3'o5; test_vecs[16][18:16] = 3'h1;
test_vecs[16][15:0] = 16'd0101;

                                                        //    logical-
right-shift

    test_vecs[17][36] = 1'b0;
    test_vecs[17][35] = 1'b1;    test_vecs[17][34] = 1'b1;
test_vecs[17][33] = 1'b1;        test_vecs[17][32] = 1'b0;
    test_vecs[17][31] = 1'b1;    test_vecs[17][30:29] = 2'b00;
test_vecs[17][28:25] = 4'd3;     test_vecs[17][24:22] = 3'o5;
    test_vecs[17][21:19] = 3'o5; test_vecs[17][18:16] = 3'h1;
test_vecs[17][15:0] = 16'd0101;

    test_vecs[18][36] = 1'b0;
    test_vecs[18][35] = 1'b1;    test_vecs[18][34] = 1'b1;
test_vecs[18][33] = 1'b1;        test_vecs[18][32] = 1'b0;
    test_vecs[18][31] = 1'b1;    test_vecs[18][30:29] = 2'b00;
test_vecs[18][28:25] = 4'd2;     test_vecs[18][24:22] = 3'o0;
    test_vecs[18][21:19] = 3'o5; test_vecs[18][18:16] = 3'h1;
test_vecs[18][15:0] = 16'd0101;
```

```
    test_vecs[19][36] = 1'b0;
    test_vecs[19][35] = 1'b1;    test_vecs[19][34] = 1'b1;
test_vecs[19][33] = 1'b1;        test_vecs[19][32] = 1'b0;
    test_vecs[19][31] = 1'b1;    test_vecs[19][30:29] = 2'b00;
test_vecs[19][28:25] = 4'd3;     test_vecs[19][24:22] = 3'o7;
    test_vecs[19][21:19] = 3'o5; test_vecs[19][18:16] = 3'h1;
test_vecs[19][15:0] = 16'd0101;

    test_vecs[20][36] = 1'b0;
    test_vecs[20][35] = 1'b1;    test_vecs[20][34] = 1'b1;
test_vecs[20][33] = 1'b1;        test_vecs[20][32] = 1'b0;
    test_vecs[20][31] = 1'b1;    test_vecs[20][30:29] = 2'b00;
test_vecs[20][28:25] = 4'd4;     test_vecs[20][24:22] = 3'o2;
    test_vecs[20][21:19] = 3'o5; test_vecs[20][18:16] = 3'h1;
test_vecs[20][15:0] = 16'd0101;


                                                        //arith- right-
shift

    test_vecs[21][36] = 1'b0;
    test_vecs[21][35] = 1'b1;    test_vecs[21][34] = 1'b1;
test_vecs[21][33] = 1'b1;        test_vecs[21][32] = 1'b1;
    test_vecs[21][31] = 1'b1;    test_vecs[21][30:29] = 2'b00;
test_vecs[21][28:25] = 4'd5;     test_vecs[21][24:22] = 3'o4;
    test_vecs[21][21:19] = 3'o5; test_vecs[21][18:16] = 3'h1;
test_vecs[21][15:0] = 16'd0101;

    test_vecs[22][36] = 1'b0;
    test_vecs[22][35] = 1'b1;    test_vecs[22][34] = 1'b1;
test_vecs[22][33] = 1'b1;        test_vecs[22][32] = 1'b1;
    test_vecs[22][31] = 1'b1;    test_vecs[22][30:29] = 2'b00;
test_vecs[22][28:25] = 4'd6;     test_vecs[22][24:22] = 3'o5;
    test_vecs[22][21:19] = 3'o5; test_vecs[22][18:16] = 3'h1;
test_vecs[22][15:0] = 16'd0101;

    test_vecs[23][36] = 1'b0;
    test_vecs[23][35] = 1'b1;    test_vecs[23][34] = 1'b1;
test_vecs[23][33] = 1'b1;        test_vecs[23][32] = 1'b1;
    test_vecs[23][31] = 1'b1;    test_vecs[23][30:29] = 2'b00;
test_vecs[23][28:25] = 4'd7;     test_vecs[23][24:22] = 3'o6;
    test_vecs[23][21:19] = 3'o5; test_vecs[23][18:16] = 3'h1;
test_vecs[23][15:0] = 16'd0101;

    test_vecs[24][36] = 1'b0;
    test_vecs[24][35] = 1'b1;    test_vecs[24][34] = 1'b1;
test_vecs[24][33] = 1'b1;        test_vecs[24][32] = 1'b1;
    test_vecs[24][31] = 1'b1;    test_vecs[24][30:29] = 2'b00;
test_vecs[24][28:25] = 4'd2;     test_vecs[24][24:22] = 3'o2;
    test_vecs[24][21:19] = 3'o5; test_vecs[24][18:16] = 3'h1;
test_vecs[24][15:0] = 16'd0101;

                                                        // slt

    test_vecs[25][36] = 1'b1;
    test_vecs[25][35] = 1'b1;    test_vecs[25][34] = 1'b0;
test_vecs[25][33] = 1'b1;        test_vecs[25][32] = 1'b1;
```

```
    test_vecs[25][31] = 1'b1;    test_vecs[25][30:29] = 2'b00;
test_vecs[25][28:25] = 4'd5;     test_vecs[25][24:22] = 3'o7;
    test_vecs[25][21:19] = 3'o3; test_vecs[25][18:16] = 3'h1;
test_vecs[25][15:0] = 16'd0101;

    test_vecs[26][36] = 1'b1;
    test_vecs[26][35] = 1'b1;    test_vecs[26][34] = 1'b0;
test_vecs[26][33] = 1'b1;        test_vecs[26][32] = 1'b1;
    test_vecs[26][31] = 1'b1;    test_vecs[26][30:29] = 2'b00;
test_vecs[26][28:25] = 4'd6;     test_vecs[26][24:22] = 3'o3;
    test_vecs[26][21:19] = 3'o7; test_vecs[26][18:16] = 3'h1;
test_vecs[26][15:0] = 16'd0101;

    test_vecs[27][36] = 1'b1;
    test_vecs[27][35] = 1'b1;    test_vecs[27][34] = 1'b0;
test_vecs[27][33] = 1'b1;        test_vecs[27][32] = 1'b1;
    test_vecs[27][31] = 1'b1;    test_vecs[27][30:29] = 2'b00;
test_vecs[27][28:25] = 4'd7;     test_vecs[27][24:22] = 3'o0;
    test_vecs[27][21:19] = 3'o3; test_vecs[27][18:16] = 3'h1;
test_vecs[27][15:0] = 16'd0101;

    test_vecs[28][36] = 1'b1;
    test_vecs[28][35] = 1'b1;    test_vecs[28][34] = 1'b0;
test_vecs[28][33] = 1'b1;        test_vecs[28][32] = 1'b1;
    test_vecs[28][31] = 1'b1;    test_vecs[28][30:29] = 2'b00;
test_vecs[28][28:25] = 4'd2;     test_vecs[28][24:22] = 3'o7;
    test_vecs[28][21:19] = 3'o0; test_vecs[28][18:16] = 3'h1;
test_vecs[28][15:0] = 16'd0101;

  end
  initial {slt_sel,sel,main_sel,sft_sel,ryt_sft_sel, wr, op,
sft_op,rd_addr_a, rd_addr_b, wr_addr, d_in} = 0;
  reg_alu ALU(clk, reset,slt_sel, sel, main_sel,sft_sel,ryt_sft_sel,wr,
op, sft_op, rd_addr_a, rd_addr_b, wr_addr, d_in,
  d_out_a, d_out_b, cout);
  initial begin
    #0
 for(i=0;i<`TESTVECS;i=i+1)
      begin #10 {slt_sel,sel,main_sel,sft_sel,ryt_sft_sel, wr, op,
sft_op, rd_addr_a, rd_addr_b, wr_addr, d_in}=test_vecs[i]; end
    #100 $finish;
  end
endmodule
```

GTKWave - /home/vinayaka/ddcoproj/Final_proj/tb_reg_alu.vcd

File Edit Search Time Markers View Help

From: 0 sec To: 390 ns Marker: 179800 ps | Cursor: 155 ns

SST

- tb
  - ALU

Type	Signals
wire	op[1:0]
wire	rd_addr_a[2:0]
wire	rd_addr_b[2:0]
wire	reg_write_ip[15:0]
wire	reset
wire	ryt_sft_sel
wire	sel
wire	sft_op[3:0]
wire	sft_sel
wire	slt_sel
wire	wr
wire	wr_addr[2:0]

Filter:

Append Insert Replace

Signals

Time

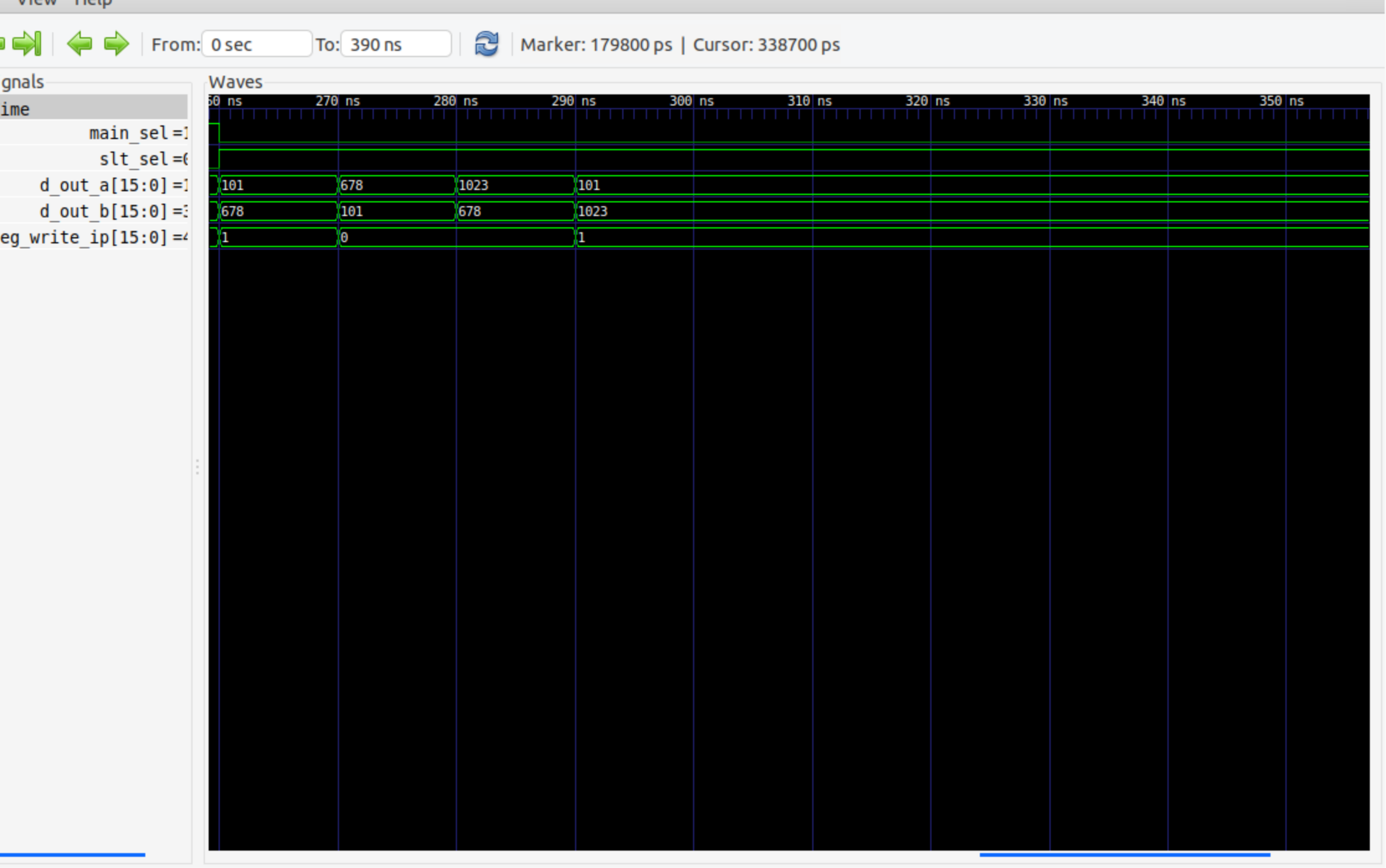
sel =1
sft_sel =0
ryt_sft_sel =0
d_out_a[15:0] =1
sft_op[3:0] =2
reg_write_ip[15:0] =4

Waves

ns 100 ns 110 ns 120 ns 130 ns 140 ns 150 ns 160 ns 170 ns 180 ns

d_out_a[15:0]	32769	1023	2046	8184	32769	678	41245	37119	33809	101	37119
sft_op[3:0]	0	1	2		0	4	0			2	3
reg_write_ip[15:0]	32769	2046	8184	32736	32769	10848	41245	37119	33809	404	4639


Signals
Waves
Time
sel
sft_sel
ryt_sft_sel
d_out_a[15:0]
sft_op[3:0]
reg_write_ip[15:0]
80 ns
190 ns
200 ns
210 ns
220 ns
230 ns
240 ns
250 ns
260 ns
270 ns
37119
1023
101
32769
41245
37119
33809
32769
101
678
3
2
3
4
5
6
7
2
5
6
4639
255
12
2048
64776
65091
65288
57344
1
0


View Help
From: 0 sec
To: 390 ns
Marker: 179800 ps | Cursor: 338700 ps
gnals
ime
main_sel
slt_sel
d_out_a[15:0]
d_out_b[15:0]
eg_write_ip[15:0]
Waves
50 ns
270 ns
280 ns
290 ns
300 ns
310 ns
320 ns
330 ns
340 ns
350 ns
101
678
1023
101
678
101
678
1023
1
0
1