University Of Waterloo

ECE 429

Task 3
Register File and Execute Stage

Stephan Van Den Heuval	sjvanden
Ravil Baizhiyenov	rbaizhiy
David Janssen	dajjanss
Ben Ridder	brridder

July 4, 2012

1 Register File Output

Listing 1: Bubble Sort Register File Output

```
# Initializing memory
# Initializing Fetch module
# opened file srec_files/BubbleSort.srec
#
# read in file
#
# Time: 1350, PC: 80020000, RS: 29, RT: 29, IR: 27bdffc8
# Time: 1360, PC: 80020004, RS: 29, RT: 31, IR: afbf0034
# Time: 1370, PC: 80020008, RS: 29, RT: 30, IR: afbe0030
# Time: 1380, PC: 8002000c, RS: 29, RT: 0, IR: 03a0f021
# Time: 1390, PC: 80020010, RS: 0, RT: 2, IR: 2402000c
# Time: 1400, PC: 80020014, RS: 30, RT: 2, IR: afc20010
# Time: 1410, PC: 80020018, RS: 0, RT: 2, IR: 24020009
# Time: 1420, PC: 8002001c, RS: 30, RT: 2, IR: afc20014
# Time: 1430, PC: 80020020, RS: 0, RT: 2, IR: 24020004
# Time: 1440, PC: 80020024, RS: 30, RT: 2, IR: afc20018
# Time: 1450, PC: 80020028, RS: 0, RT: 2, IR: 24020063
# Time: 1460, PC: 8002002c, RS: 30, RT: 2, IR: afc2001c
# Time: 1470, PC: 80020030, RS: 0, RT: 2, IR: 24020078
# Time: 1480, PC: 80020034, RS: 30, RT: 2, IR: afc20020
# Time: 1490, PC: 80020038, RS: 0, RT: 2, IR: 24020001
# Time: 1500, PC: 8002003c, RS: 30, RT: 2, IR: afc20024
# Time: 1510, PC: 80020040, RS: 0, RT: 2, IR: 24020003
# Time: 1520, PC: 80020044, RS: 30, RT: 2, IR: afc20028
# Time: 1530, PC: 80020048, RS: 0, RT: 2, IR: 2402000a
# Time: 1540, PC: 8002004c, RS: 30, RT: 2, IR: afc2002c
# Time: 1550, PC: 80020050, RS: 30, RT: 2, IR: 27c20010
# Time: 1560, PC: 80020054, RS: 2, RT: 0, IR: 00402021
# Time: 1570, PC: 80020058, RS: 0, RT: 5, IR: 24050008
# Time: 1580, PC: 8002005c, RS: 0, RT: 0, IR: 0c008020
# Time: 1590, PC: 80020060, RS: 0, RT: 0, IR: 00000000
# Time: 1600, PC: 80020064, RS: 0, RT: 0, IR: 00001021
# Time: 1610, PC: 80020068, RS: 30, RT: 0, IR: 03c0e821
# Time: 1620, PC: 8002006c, RS: 29, RT: 31, IR: 8fbf0034
# Time: 1630, PC: 80020070, RS: 29, RT: 30, IR: 8fbe0030
# Time: 1640, PC: 80020074, RS: 29, RT: 29, IR: 27bd0038
# Time: 1650, PC: 80020078, RS: 31, RT: 0, IR: 03e00008
# Time: 1660, PC: 8002007c, RS: 0, RT: 0, IR: 00000000
# Time: 1670, PC: 80020080, RS: 29, RT: 29, IR: 27bdffe8
# Time: 1680, PC: 80020084, RS: 29, RT: 30, IR: afbe0014
# Time: 1690, PC: 80020088, RS: 29, RT: 0, IR: 03a0f021
# Time: 1700, PC: 8002008c, RS: 30, RT: 4, IR: afc40018
# Time: 1710, PC: 80020090, RS: 30, RT: 5, IR: afc5001c
# Time: 1720, PC: 80020094, RS: 30, RT: 0, IR: afc00000
# Time: 1730, PC: 80020098, RS: 0, RT: 0, IR: 0800805f
# Time: 1740, PC: 8002009c, RS: 0, RT: 0, IR: 00000000
# Time: 1750, PC: 800200a0, RS: 0, RT: 2, IR: 24020001
# Time: 1760, PC: 800200a4, RS: 30, RT: 2, IR: afc20004
```

```
# Time: 1770, PC: 800200a8, RS: 0, RT: 0, IR: 08008055
# Time: 1780, PC: 800200ac, RS: 0, RT: 0, IR: 00000000
# Time: 1790, PC: 800200b0, RS: 30, RT: 2, IR: 8fc20004
# Time: 1800, PC: 800200b4, RS: 2, RT: 2, IR: 2442ffff
# Time: 1810, PC: 800200b8, RS: 0, RT: 2, IR: 00021080
# Time: 1820, PC: 800200bc, RS: 30, RT: 3, IR: 8fc30018
# Time: 1830, PC: 800200c0, RS: 3, RT: 2, IR: 00621021
# Time: 1840, PC: 800200c4, RS: 2, RT: 3, IR: 8c430000
# Time: 1850, PC: 800200c8, RS: 30, RT: 2, IR: 8fc20004
# Time: 1860, PC: 800200cc, RS: 0, RT: 2, IR: 00021080
# Time: 1870, PC: 800200d0, RS: 30, RT: 4, IR: 8fc40018
# Time: 1880, PC: 800200d4, RS: 4, RT: 2, IR: 00821021
# Time: 1890, PC: 800200d8, RS: 2, RT: 2, IR: 8c420000
# Time: 1900, PC: 800200dc, RS: 2, RT: 3, IR: 0043102a
# Time: 1910, PC: 800200e0, RS: 2, RT: 0, IR: 10400019
# Time: 1920, PC: 800200e4, RS: 0, RT: 0, IR: 00000000
# Time: 1930, PC: 800200e8, RS: 30, RT: 2, IR: 8fc20004
# Time: 1940, PC: 800200ec, RS: 2, RT: 2, IR: 2442ffff
# Time: 1950, PC: 800200f0, RS: 0, RT: 2, IR: 00021080
# Time: 1960, PC: 800200f4, RS: 30, RT: 3, IR: 8fc30018
# Time: 1970, PC: 800200f8, RS: 3, RT: 2, IR: 00621021
# Time: 1980, PC: 800200fc, RS: 2, RT: 2, IR: 8c420000
# Time: 1990, PC: 80020100, RS: 30, RT: 2, IR: afc20008
# Time: 2000, PC: 80020104, RS: 30, RT: 2, IR: 8fc20004
# Time: 2010, PC: 80020108, RS: 2, RT: 2, IR: 2442ffff
# Time: 2020, PC: 8002010c, RS: 0, RT: 2, IR: 00021080
# Time: 2030, PC: 80020110, RS: 30, RT: 3, IR: 8fc30018
# Time: 2040, PC: 80020114, RS: 3, RT: 2, IR: 00621021
# Time: 2050, PC: 80020118, RS: 30, RT: 3, IR: 8fc30004
# Time: 2060, PC: 8002011c, RS: 0, RT: 3, IR: 00031880
# Time: 2070, PC: 80020120, RS: 30, RT: 4, IR: 8fc40018
# Time: 2080, PC: 80020124, RS: 4, RT: 3, IR: 00831821
# Time: 2090, PC: 80020128, RS: 3, RT: 3, IR: 8c630000
# Time: 2100, PC: 8002012c, RS: 2, RT: 3, IR: ac430000
# Time: 2110, PC: 80020130, RS: 30, RT: 2, IR: 8fc20004
# Time: 2120, PC: 80020134, RS: 0, RT: 2, IR: 00021080
# Time: 2130, PC: 80020138, RS: 30, RT: 3, IR: 8fc30018
# Time: 2140, PC: 8002013c, RS: 3, RT: 2, IR: 00621021
# Time: 2150, PC: 80020140, RS: 30, RT: 3, IR: 8fc30008
# Time: 2160, PC: 80020144, RS: 2, RT: 3, IR: ac430000
# Time: 2170, PC: 80020148, RS: 30, RT: 2, IR: 8fc20004
# Time: 2180, PC: 8002014c, RS: 2, RT: 2, IR: 24420001
# Time: 2190, PC: 80020150, RS: 30, RT: 2, IR: afc20004
# Time: 2200, PC: 80020154, RS: 30, RT: 3, IR: 8fc3001c
# Time: 2210, PC: 80020158, RS: 30, RT: 2, IR: 8fc20000
# Time: 2220, PC: 8002015c, RS: 3, RT: 2, IR: 00621823
# Time: 2230, PC: 80020160, RS: 30, RT: 2, IR: 8fc20004
# Time: 2240, PC: 80020164, RS: 2, RT: 3, IR: 0043102a
# Time: 2250, PC: 80020168, RS: 2, RT: 0, IR: 1440ffd1
# Time: 2260, PC: 8002016c, RS: 0, RT: 0, IR: 00000000
# Time: 2270, PC: 80020170, RS: 30, RT: 2, IR: 8fc20000
```

```
# Time: 2280, PC: 80020174, RS: 2, RT: 2, IR: 24420001
# Time: 2290, PC: 80020178, RS: 30, RT: 2, IR: afc20000
# Time: 2300, PC: 8002017c, RS: 30, RT: 3, IR: 8fc30000
# Time: 2310, PC: 80020180, RS: 30, RT: 2, IR: 8fc2001c
# Time: 2320, PC: 80020184, RS: 3, RT: 2, IR: 0062102a
# Time: 2330, PC: 80020188, RS: 2, RT: 0, IR: 1440ffc5
# Time: 2340, PC: 8002018c, RS: 0, RT: 0, IR: 00000000
# Time: 2350, PC: 80020190, RS: 30, RT: 0, IR: 03c0e821
# Time: 2360, PC: 80020194, RS: 29, RT: 30, IR: 8fbe0014
# Time: 2370, PC: 80020198, RS: 29, RT: 29, IR: 27bd0018
# Time: 2380, PC: 8002019c, RS: 31, RT: 0, IR: 03e00008
# Time: 2390, PC: 800201a0, RS: 0, RT: 0, IR: 00000000
# ** Note: $finish : reg_file_tb.v(106)
# Time: 2400 ns Iteration: 0 Instance: /reg_file_tb
```

Listing 2: Simple Add Register File Output

```
# Initializing memory
# Initializing Fetch module
# opened file srec_files/SimpleAdd.srec
#
# read in file
#
# Time: 260, PC: 80020000, RS: 29, RT: 29, IR: 27bdffe8
# Time: 270, PC: 80020004, RS: 29, RT: 30, IR: afbe0014
# Time: 280, PC: 80020008, RS: 29, RT: 0, IR: 03a0f021
# Time: 290, PC: 8002000c, RS: 0, RT: 2, IR: 24020003
# Time: 300, PC: 80020010, RS: 30, RT: 2, IR: afc20000
# Time: 310, PC: 80020014, RS: 0, RT: 2, IR: 24020002
# Time: 320, PC: 80020018, RS: 30, RT: 2, IR: afc20004
# Time: 330, PC: 8002001c, RS: 30, RT: 0, IR: afc00008
# Time: 340, PC: 80020020, RS: 30, RT: 3, IR: 8fc30000
# Time: 350, PC: 80020024, RS: 30, RT: 2, IR: 8fc20004
# Time: 360, PC: 80020028, RS: 3, RT: 2, IR: 00621021
# Time: 370, PC: 8002002c, RS: 30, RT: 2, IR: afc20008
# Time: 380, PC: 80020030, RS: 30, RT: 2, IR: 8fc20008
# Time: 390, PC: 80020034, RS: 30, RT: 0, IR: 03c0e821
# Time: 400, PC: 80020038, RS: 29, RT: 30, IR: 8fbe0014
# Time: 410, PC: 8002003c, RS: 29, RT: 29, IR: 27bd0018
# Time: 420, PC: 80020040, RS: 31, RT: 0, IR: 03e00008
# Time: 430, PC: 80020044, RS: 0, RT: 0, IR: 00000000
# ** Note: $finish : reg_file_tb.v(106)
# Time: 440 ns Iteration: 0 Instance: /reg_file_tb
```

Listing 3: Simple If Register File Output

```
# Initializing memory
# Initializing Fetch module
# opened file srec_files/SimpleIf.srec
#
# read in file
#
```

```
# Time: 430, PC: 80020000, RS: 29, RT: 29, IR: 27bdffe8
# Time: 440, PC: 80020004, RS: 29, RT: 30, IR: afbe0014
# Time: 450, PC: 80020008, RS: 29, RT: 0, IR: 03a0f021
# Time: 460, PC: 8002000c, RS: 0, RT: 2, IR: 24020003
# Time: 470, PC: 80020010, RS: 30, RT: 2, IR: afc20000
# Time: 480, PC: 80020014, RS: 0, RT: 2, IR: 24020002
# Time: 490, PC: 80020018, RS: 30, RT: 2, IR: afc20004
# Time: 500, PC: 8002001c, RS: 30, RT: 0, IR: afc00008
# Time: 510, PC: 80020020, RS: 30, RT: 2, IR: 8fc20000
# Time: 520, PC: 80020024, RS: 2, RT: 2, IR: 28420005
# Time: 530, PC: 80020028, RS: 2, RT: 0, IR: 10400007
# Time: 540, PC: 8002002c, RS: 0, RT: 0, IR: 00000000
# Time: 550, PC: 80020030, RS: 30, RT: 3, IR: 8fc30000
# Time: 560, PC: 80020034, RS: 30, RT: 2, IR: 8fc20004
# Time: 570, PC: 80020038, RS: 3, RT: 2, IR: 00621021
# Time: 580, PC: 8002003c, RS: 30, RT: 2, IR: afc20000
# Time: 590, PC: 80020040, RS: 0, RT: 0, IR: 08008016
# Time: 600, PC: 80020044, RS: 0, RT: 0, IR: 00000000
# Time: 610, PC: 80020048, RS: 30, RT: 3, IR: 8fc30000
# Time: 620, PC: 8002004c, RS: 30, RT: 2, IR: 8fc20004
# Time: 630, PC: 80020050, RS: 3, RT: 2, IR: 00621023
# Time: 640, PC: 80020054, RS: 30, RT: 2, IR: afc20000
# Time: 650, PC: 80020058, RS: 30, RT: 3, IR: 8fc30000
# Time: 660, PC: 8002005c, RS: 30, RT: 2, IR: 8fc20004
# Time: 670, PC: 80020060, RS: 3, RT: 2, IR: 00621021
# Time: 680, PC: 80020064, RS: 30, RT: 2, IR: afc20008
# Time: 690, PC: 80020068, RS: 30, RT: 2, IR: 8fc20008
# Time: 700, PC: 8002006c, RS: 30, RT: 0, IR: 03c0e821
# Time: 710, PC: 80020070, RS: 29, RT: 30, IR: 8fbe0014
# Time: 720, PC: 80020074, RS: 29, RT: 29, IR: 27bd0018
# Time: 730, PC: 80020078, RS: 31, RT: 0, IR: 03e00008
# Time: 740, PC: 8002007c, RS: 0, RT: 0, IR: 00000000
# ** Note: $finish : reg_file_tb.v(106)
# Time: 750 ns Iteration: 0 Instance: /reg_file_tb
```

Listing 4: Sum Array Register File Output

```
# Initializing memory
# Initializing Fetch module
# opened file srec_files/SumArray.srec
#
# read in file
#
# Time: 570, PC: 80020000, RS: 29, RT: 29, IR: 27bdffc8
# Time: 580, PC: 80020004, RS: 29, RT: 30, IR: afbe0034
# Time: 590, PC: 80020008, RS: 29, RT: 0, IR: 03a0f021
# Time: 600, PC: 8002000c, RS: 30, RT: 0, IR: afc00000
# Time: 610, PC: 80020010, RS: 30, RT: 0, IR: afc00004
# Time: 620, PC: 80020014, RS: 30, RT: 0, IR: afc00000
# Time: 630, PC: 80020018, RS: 0, RT: 0, IR: 08008010
# Time: 640, PC: 8002001c, RS: 0, RT: 0, IR: 00000000
# Time: 650, PC: 80020020, RS: 30, RT: 2, IR: 8fc20000
```

```
# Time: 660, PC: 80020024, RS: 0, RT: 2, IR: 00021080
# Time: 670, PC: 80020028, RS: 30, RT: 2, IR: 03c21021
# Time: 680, PC: 8002002c, RS: 30, RT: 3, IR: 8fc30000
# Time: 690, PC: 80020030, RS: 2, RT: 3, IR: ac430008
# Time: 700, PC: 80020034, RS: 30, RT: 2, IR: 8fc20000
# Time: 710, PC: 80020038, RS: 2, RT: 2, IR: 24420001
# Time: 720, PC: 8002003c, RS: 30, RT: 2, IR: afc20000
# Time: 730, PC: 80020040, RS: 30, RT: 2, IR: 8fc20000
# Time: 740, PC: 80020044, RS: 2, RT: 2, IR: 2842000a
# Time: 750, PC: 80020048, RS: 2, RT: 0, IR: 1440fff5
# Time: 760, PC: 8002004c, RS: 0, RT: 0, IR: 00000000
# Time: 770, PC: 80020050, RS: 30, RT: 0, IR: afc00000
# Time: 780, PC: 80020054, RS: 0, RT: 0, IR: 08008021
# Time: 790, PC: 80020058, RS: 0, RT: 0, IR: 00000000
# Time: 800, PC: 8002005c, RS: 30, RT: 2, IR: 8fc20000
# Time: 810, PC: 80020060, RS: 0, RT: 2, IR: 00021080
# Time: 820, PC: 80020064, RS: 30, RT: 2, IR: 03c21021
# Time: 830, PC: 80020068, RS: 2, RT: 2, IR: 8c420008
# Time: 840, PC: 8002006c, RS: 30, RT: 3, IR: 8fc30004
# Time: 850, PC: 80020070, RS: 3, RT: 2, IR: 00621021
# Time: 860, PC: 80020074, RS: 30, RT: 2, IR: afc20004
# Time: 870, PC: 80020078, RS: 30, RT: 2, IR: 8fc20000
# Time: 880, PC: 8002007c, RS: 2, RT: 2, IR: 24420001
# Time: 890, PC: 80020080, RS: 30, RT: 2, IR: afc20000
# Time: 900, PC: 80020084, RS: 30, RT: 2, IR: 8fc20000
# Time: 910, PC: 80020088, RS: 2, RT: 2, IR: 2842000a
# Time: 920, PC: 8002008c, RS: 2, RT: 0, IR: 1440fff3
# Time: 930, PC: 80020090, RS: 0, RT: 0, IR: 00000000
# Time: 940, PC: 80020094, RS: 30, RT: 2, IR: 8fc20004
# Time: 950, PC: 80020098, RS: 30, RT: 0, IR: 03c0e821
# Time: 960, PC: 8002009c, RS: 29, RT: 30, IR: 8fbe0034
# Time: 970, PC: 800200a0, RS: 29, RT: 29, IR: 27bd0038
# Time: 980, PC: 800200a4, RS: 31, RT: 0, IR: 03e00008
# Time: 990, PC: 800200a8, RS: 0, RT: 0, IR: 00000000
# ** Note: $finish : reg_file_tb.v(106)
# Time: 1 us Iteration: 0 Instance: /reg_file_tb
```

2 Register File Source Code

Listing 5: Register File

```
//
// reg_file.v
//
// Register File
//
//
// Register outputs are the values stored in the regs
// Register inputs are pointers for the specific reg to use
//
```

```
module reg_file (
    clock,
    rsOut,
    rtOut,
    rsIn,
    rtIn,
    rdIn,
    regWriteEnable,
    writeBackData
);

    parameter REG_WIDTH = 32;
    parameter NUM_REGS = 32;

    input wire clock;
    input wire[4:0] rdIn;
    input wire[0:31] writeBackData;

    output reg[0:31] rsOut;
    output reg[0:31] rtOut;

    input wire[4:0] rsIn;
    input wire[4:0] rtIn;
    input wire regWriteEnable;

    reg[0:REG_WIDTH-1] registers[0:NUM_REGS-1];

    integer i;

    // Zero out all of the registers.
    initial
    begin
        for (i = 0; i  NUM_REGS; i = i + 1) begin
            registers[i] = i;
        end // for ()
    end // initial

    always @(posedge clock)
    begin
        fork
            rsOut = registers[rsIn];
            rtOut = registers[rtIn];
            //$display("time: %d Rs %d, RT: %d", $time, registers[rsIn], registers[rtIn]);
        join
        registers[rdIn] = writeBackData;
    end // always @(posedge clock)

endmodule
```

Listing 6: Register File Unit Test Bench

```
//
// reg_file_tb.v
//
// Register File Unit Test Bench
//

module reg_file_unit_tb();

reg clock;

reg[0:31] inst;
reg[0:31] data;
reg[0:31] pc;
reg[0:31] writeBackData;
reg[0:4] rdIn;

wire[0:31] rsOut;
wire[0:31] rtOut;
wire[0:31] pcOut;
wire[0:31] irOut;


reg_file U0(
    .clock (clock),
    .rsOut (rsOut),
    .rtOut (rtOut),
    .rdIn (rdIn),
    .writeBackData (writeBackData),
    .pcIn (pc),
    .pcOut (pcOut),
    .irIn (inst),
    .irOut (irOut)
);

initial begin
    clock = 1;
    inst = 0;
    data = 0;
    pc = 0;
    writeBackData = 0;
end

initial begin
    $display("\t\ttime,\tclock,\tpcIn,\tpcOut,\tirIn,\tirOut,\trsOut,\trtOut,\twriteBackData")
        ;
    $monitor("%d,\t%b,\t%h,\t%h,\t%h,\t%h,\t%h,\t%h,\t%h",
            $time, clock, pc, pcOut, inst, irOut, rsOut, rtOut, writeBackData);
end


always begin
```

```
    #5 clock = !clock;
end

initial
begin
    pc = 32'hdead_beef;
    inst = 32'h0000_0000;
    rdIn = 5'b0_0000; // $r0
    writeBackData = 32'hdeed_deed;
    @ (posedge clock);
    pc = 32'hffff_eeee;
    inst = 32'h0000_0000;
    rdIn = 5'b0_0001; // $r1
    writeBackData = 32'hbeaf_dead;
    @ (posedge clock);
    inst = 32'b000000_00000_00001_00010_00000_100000;
    rdIn = 5'b0_0100; // $r4
    writeBackData = 32'hbeef_deed;
    @ (posedge clock);
end

initial
    #100 $finish;

endmodule
```

Listing 7: Register File Test Bench

```
//
// fetch_tb.v
//
// Fetch test bench
//

module reg_file_tb;
    reg clock;

    wire[0:31] address;
    wire wren;
    wire[0:31] data_in;
    wire[0:31] data_out;

    wire[0:31] fetch_address;
    wire fetch_wren;
    wire[0:31] fetch_data_in;
    wire[0:31] fetch_data_out;

    wire[0:31] fetch_insn_decode;
    wire[0:31] fetch_pc;
    reg fetch_stall;

    wire[0:31] decode_rs_data;
```

```
wire[0:31] decode_rt_data;
wire[4:0] decode_rd_in;
wire[0:31] decode_pc_out;
wire[0:31] decode_ir_out;
wire[0:31] decode_write_back_data;
wire decode_reg_write_enable;
wire[0:`CONTROL_REG_SIZE-1] decode_control;

wire[0:31] srec_address;
wire srec_wren;
wire[0:31] srec_data_in;
wire[0:31] srec_data_out;

wire srec_done;

reg[0:31] tb_address;
reg tb_wren;
reg[0:31] tb_data_in;
wire[0:31] tb_data_out;

wire[0:31] bytes_read;
integer byte_count;
integer read_word;
reg[0:31] fetch_word;
reg instruction_valid;

mem_controller mcu(
    .clock (clock),
    .address (address),
    .wren (wren),
    .data_in (data_in),
    .data_out (data_out)
);

fetch DUT(
    .clock (clock),
    .address (fetch_address),
    .insn (fetch_data_in),
    .insn_decode (fetch_data_out),
    .pc (fetch_pc),
    .wren (fetch_wren),
    .stall (fetch_stall)
);

srec_parser #("srec_files/SimpleIf.srec") U0(
    .clock (clock),
    .mem_address (srec_address),
    .mem_wren (srec_wren),
    .mem_data_in (srec_data_in),
    .mem_data_out (srec_data_out),
    .done (srec_done),
```

```
    .bytes_read(bytes_read)
);

decode U1(
    .clock (clock),
    .insn (fetch_data_out),
        .insn_valid (instruction_valid),
    .pc (fetch_address),
    .rsData (decode_rs_data),
    .rtData (decode_rt_data),
    .rdIn (decode_rd_in),
    .pcOut (decode_pc_out),
    .irOut (decode_ir_out),
    .writeBackData (decode_write_back_data),
    .regWriteEnable (decode_reg_write_enable),
    .control (decode_control)
);

assign address = srec_done ? (fetch_stall ? tb_address : fetch_address) : srec_address;
assign wren = srec_done ? (fetch_stall ? tb_wren : fetch_wren) : srec_wren;
assign tb_data_out = data_out;
assign fetch_data_in = data_out;
assign data_in = srec_done ? (fetch_stall ? tb_data_in : fetch_data_in) : srec_data_in;

// Specify when to stop the simulation
event terminate_sim;
initial begin
    @ (terminate_sim);
    #10 $finish;
end

initial begin
    clock = 1;
    fetch_stall = 1;
    instruction_valid = 1'b0;
end

always begin
    #5 clock = !clock;
end

initial begin
    $dumpfile("reg_file_tb.vcd");
    $dumpvars;
end

initial begin
    @(posedge srec_done);
    @(posedge clock);
    byte_count = bytes_read+4;
    tb_address = 32'h8002_0000;
```

```
        tb_wren = 1'b0;
        instruction_valid = 1'b0;
        fetch_stall = 0;
        while (byte_count  0) begin
            @(posedge clock);
            if ((fetch_address - 4)  32'h8002_0000) begin
                instruction_valid = 1'b0;
            end else begin
                instruction_valid = 1'b1;
            end
                read_word = tb_data_out;
            fetch_word = fetch_data_out;

            $display("Time: %d, PC: %X, RS:%d, RT:%d, IR: %X", $time,
                   decode_pc_out, decode_rs_data, decode_rt_data, decode_ir_out);
            tb_address = tb_address + 4;
            byte_count = byte_count - 4;
        end

        // The decode runs one clock cycle behind the fetch
        @(posedge clock);
        $display("Time: %d, PC: %X, RS:%d, RT:%d, IR: %X", $time,
                 decode_pc_out, decode_rs_data, decode_rt_data, decode_ir_out);
        tb_address = tb_address + 4;

        instruction_valid = 1'b0;

        -terminate_sim;
    end

endmodule
```

3 ALU Output

Listing 8: Bubble Sort ALU Output

```
# Initializing memory
# Initializing Fetch module
# opened file srec_files/BubbleSort.srec
#
# read in file
#
# Time: 1360, Inp insn: 27bdffc8, Inp PC: 80020000, Inp RS: 29, Inp RT: 29, ALU_RESULT: 65509
# Time: 1370, Inp insn: afbf0034, Inp PC: 80020004, Inp RS: 29, Inp RT: 31, ALU_RESULT: 65509
# Time: 1380, Inp insn: afbe0030, Inp PC: 80020008, Inp RS: 29, Inp RT: 30, ALU_RESULT: 65509
# Time: 1390, Inp insn: 03a0f021, Inp PC: 8002000c, Inp RS: 29, Inp RT: 0, ALU_RESULT: 29
# Time: 1400, Inp insn: 2402000c, Inp PC: 80020010, Inp RS: 0, Inp RT: 2, ALU_RESULT: 12
# Time: 1410, Inp insn: afc20010, Inp PC: 80020014, Inp RS: 30, Inp RT: 2, ALU_RESULT: 12
# Time: 1420, Inp insn: 24020009, Inp PC: 80020018, Inp RS: 0, Inp RT: 2, ALU_RESULT: 9
# Time: 1430, Inp insn: afc20014, Inp PC: 8002001c, Inp RS: 30, Inp RT: 2, ALU_RESULT: 9
```

```
# Time: 1440, Inp insn: 24020004, Inp PC: 80020020, Inp RS: 0, Inp RT: 2, ALU_RESULT: 4
# Time: 1450, Inp insn: afc20018, Inp PC: 80020024, Inp RS: 30, Inp RT: 2, ALU_RESULT: 4
# Time: 1460, Inp insn: 24020063, Inp PC: 80020028, Inp RS: 0, Inp RT: 2, ALU_RESULT: 99
# Time: 1470, Inp insn: afc2001c, Inp PC: 8002002c, Inp RS: 30, Inp RT: 2, ALU_RESULT: 99
# Time: 1480, Inp insn: 24020078, Inp PC: 80020030, Inp RS: 0, Inp RT: 2, ALU_RESULT: 120
# Time: 1490, Inp insn: afc20020, Inp PC: 80020034, Inp RS: 30, Inp RT: 2, ALU_RESULT: 120
# Time: 1500, Inp insn: 24020001, Inp PC: 80020038, Inp RS: 0, Inp RT: 2, ALU_RESULT: 1
# Time: 1510, Inp insn: afc20024, Inp PC: 8002003c, Inp RS: 30, Inp RT: 2, ALU_RESULT: 1
# Time: 1520, Inp insn: 24020003, Inp PC: 80020040, Inp RS: 0, Inp RT: 2, ALU_RESULT: 3
# Time: 1530, Inp insn: afc20028, Inp PC: 80020044, Inp RS: 30, Inp RT: 2, ALU_RESULT: 3
# Time: 1540, Inp insn: 2402000a, Inp PC: 80020048, Inp RS: 0, Inp RT: 2, ALU_RESULT: 10
# Time: 1550, Inp insn: afc2002c, Inp PC: 8002004c, Inp RS: 30, Inp RT: 2, ALU_RESULT: 10
# Time: 1560, Inp insn: 27c20010, Inp PC: 80020050, Inp RS: 30, Inp RT: 2, ALU_RESULT: 46
# Time: 1570, Inp insn: 00402021, Inp PC: 80020054, Inp RS: 2, Inp RT: 0, ALU_RESULT: 2
# Time: 1580, Inp insn: 24050008, Inp PC: 80020058, Inp RS: 0, Inp RT: 5, ALU_RESULT: 8
# Time: 1590, Inp insn: 0c008020, Inp PC: 8002005c, Inp RS: 0, Inp RT: 0, ALU_RESULT: 8
# Time: 1600, Inp insn: 00000000, Inp PC: 80020060, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 1610, Inp insn: 00001021, Inp PC: 80020064, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 1620, Inp insn: 03c0e821, Inp PC: 80020068, Inp RS: 30, Inp RT: 0, ALU_RESULT: 30
# Time: 1630, Inp insn: 8fbf0034, Inp PC: 8002006c, Inp RS: 29, Inp RT: 31, ALU_RESULT: 30
# Time: 1640, Inp insn: 8fbe0030, Inp PC: 80020070, Inp RS: 29, Inp RT: 30, ALU_RESULT: 30
# Time: 1650, Inp insn: 27bd0038, Inp PC: 80020074, Inp RS: 29, Inp RT: 29, ALU_RESULT: 85
# Time: 1660, Inp insn: 03e00008, Inp PC: 80020078, Inp RS: 31, Inp RT: 0, ALU_RESULT: 85
# Time: 1670, Inp insn: 00000000, Inp PC: 8002007c, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 1680, Inp insn: 27bdffe8, Inp PC: 80020080, Inp RS: 29, Inp RT: 29, ALU_RESULT: 65541
# Time: 1690, Inp insn: afbe0014, Inp PC: 80020084, Inp RS: 29, Inp RT: 30, ALU_RESULT: 65541
# Time: 1700, Inp insn: 03a0f021, Inp PC: 80020088, Inp RS: 29, Inp RT: 0, ALU_RESULT: 29
# Time: 1710, Inp insn: afc40018, Inp PC: 8002008c, Inp RS: 30, Inp RT: 4, ALU_RESULT: 29
# Time: 1720, Inp insn: afc5001c, Inp PC: 80020090, Inp RS: 30, Inp RT: 5, ALU_RESULT: 29
# Time: 1730, Inp insn: afc00000, Inp PC: 80020094, Inp RS: 30, Inp RT: 0, ALU_RESULT: 29
# Time: 1740, Inp insn: 0800805f, Inp PC: 80020098, Inp RS: 0, Inp RT: 0, ALU_RESULT: 29
# Time: 1750, Inp insn: 00000000, Inp PC: 8002009c, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 1760, Inp insn: 24020001, Inp PC: 800200a0, Inp RS: 0, Inp RT: 2, ALU_RESULT: 1
# Time: 1770, Inp insn: afc20004, Inp PC: 800200a4, Inp RS: 30, Inp RT: 2, ALU_RESULT: 1
# Time: 1780, Inp insn: 08008055, Inp PC: 800200a8, Inp RS: 0, Inp RT: 0, ALU_RESULT: 1
# Time: 1790, Inp insn: 00000000, Inp PC: 800200ac, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 1800, Inp insn: 8fc20004, Inp PC: 800200b0, Inp RS: 30, Inp RT: 2, ALU_RESULT: 0
# Time: 1810, Inp insn: 2442ffff, Inp PC: 800200b4, Inp RS: 2, Inp RT: 2, ALU_RESULT: 65537
# Time: 1820, Inp insn: 00021080, Inp PC: 800200b8, Inp RS: 0, Inp RT: 2, ALU_RESULT: 8
# Time: 1830, Inp insn: 8fc30018, Inp PC: 800200bc, Inp RS: 30, Inp RT: 3, ALU_RESULT: 8
# Time: 1840, Inp insn: 00621021, Inp PC: 800200c0, Inp RS: 3, Inp RT: 2, ALU_RESULT: 5
# Time: 1850, Inp insn: 8c430000, Inp PC: 800200c4, Inp RS: 2, Inp RT: 3, ALU_RESULT: 5
# Time: 1860, Inp insn: 8fc20004, Inp PC: 800200c8, Inp RS: 30, Inp RT: 2, ALU_RESULT: 5
# Time: 1870, Inp insn: 00021080, Inp PC: 800200cc, Inp RS: 0, Inp RT: 2, ALU_RESULT: 8
# Time: 1880, Inp insn: 8fc40018, Inp PC: 800200d0, Inp RS: 30, Inp RT: 4, ALU_RESULT: 8
# Time: 1890, Inp insn: 00821021, Inp PC: 800200d4, Inp RS: 4, Inp RT: 2, ALU_RESULT: 6
# Time: 1900, Inp insn: 8c420000, Inp PC: 800200d8, Inp RS: 2, Inp RT: 2, ALU_RESULT: 6
# Time: 1910, Inp insn: 0043102a, Inp PC: 800200dc, Inp RS: 2, Inp RT: 3, ALU_RESULT: 1
# Time: 1920, Inp insn: 10400019, Inp PC: 800200e0, Inp RS: 2, Inp RT: 0, ALU_RESULT: 1
# Time: 1930, Inp insn: 00000000, Inp PC: 800200e4, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 1940, Inp insn: 8fc20004, Inp PC: 800200e8, Inp RS: 30, Inp RT: 2, ALU_RESULT: 0
```

```
# Time: 1950, Inp insn: 2442ffff, Inp PC: 800200ec, Inp RS: 2, Inp RT: 2, ALU_RESULT: 65537
# Time: 1960, Inp insn: 00021080, Inp PC: 800200f0, Inp RS: 0, Inp RT: 2, ALU_RESULT: 8
# Time: 1970, Inp insn: 8fc30018, Inp PC: 800200f4, Inp RS: 30, Inp RT: 3, ALU_RESULT: 8
# Time: 1980, Inp insn: 00621021, Inp PC: 800200f8, Inp RS: 3, Inp RT: 2, ALU_RESULT: 5
# Time: 1990, Inp insn: 8c420000, Inp PC: 800200fc, Inp RS: 2, Inp RT: 2, ALU_RESULT: 5
# Time: 2000, Inp insn: afc20008, Inp PC: 80020100, Inp RS: 30, Inp RT: 2, ALU_RESULT: 5
# Time: 2010, Inp insn: 8fc20004, Inp PC: 80020104, Inp RS: 30, Inp RT: 2, ALU_RESULT: 5
# Time: 2020, Inp insn: 2442ffff, Inp PC: 80020108, Inp RS: 2, Inp RT: 2, ALU_RESULT: 65537
# Time: 2030, Inp insn: 00021080, Inp PC: 8002010c, Inp RS: 0, Inp RT: 2, ALU_RESULT: 8
# Time: 2040, Inp insn: 8fc30018, Inp PC: 80020110, Inp RS: 30, Inp RT: 3, ALU_RESULT: 8
# Time: 2050, Inp insn: 00621021, Inp PC: 80020114, Inp RS: 3, Inp RT: 2, ALU_RESULT: 5
# Time: 2060, Inp insn: 8fc30004, Inp PC: 80020118, Inp RS: 30, Inp RT: 3, ALU_RESULT: 5
# Time: 2070, Inp insn: 00031880, Inp PC: 8002011c, Inp RS: 0, Inp RT: 3, ALU_RESULT: 0
# Time: 2080, Inp insn: 8fc40018, Inp PC: 80020120, Inp RS: 30, Inp RT: 4, ALU_RESULT: 0
# Time: 2090, Inp insn: 00831821, Inp PC: 80020124, Inp RS: 4, Inp RT: 3, ALU_RESULT: 7
# Time: 2100, Inp insn: 8c630000, Inp PC: 80020128, Inp RS: 3, Inp RT: 3, ALU_RESULT: 7
# Time: 2110, Inp insn: ac430000, Inp PC: 8002012c, Inp RS: 2, Inp RT: 3, ALU_RESULT: 7
# Time: 2120, Inp insn: 8fc20004, Inp PC: 80020130, Inp RS: 30, Inp RT: 2, ALU_RESULT: 7
# Time: 2130, Inp insn: 00021080, Inp PC: 80020134, Inp RS: 0, Inp RT: 2, ALU_RESULT: 8
# Time: 2140, Inp insn: 8fc30018, Inp PC: 80020138, Inp RS: 30, Inp RT: 3, ALU_RESULT: 8
# Time: 2150, Inp insn: 00621021, Inp PC: 8002013c, Inp RS: 3, Inp RT: 2, ALU_RESULT: 5
# Time: 2160, Inp insn: 8fc30008, Inp PC: 80020140, Inp RS: 30, Inp RT: 3, ALU_RESULT: 5
# Time: 2170, Inp insn: ac430000, Inp PC: 80020144, Inp RS: 2, Inp RT: 3, ALU_RESULT: 5
# Time: 2180, Inp insn: 8fc20004, Inp PC: 80020148, Inp RS: 30, Inp RT: 2, ALU_RESULT: 5
# Time: 2190, Inp insn: 24420001, Inp PC: 8002014c, Inp RS: 2, Inp RT: 2, ALU_RESULT: 3
# Time: 2200, Inp insn: afc20004, Inp PC: 80020150, Inp RS: 30, Inp RT: 2, ALU_RESULT: 3
# Time: 2210, Inp insn: 8fc3001c, Inp PC: 80020154, Inp RS: 30, Inp RT: 3, ALU_RESULT: 3
# Time: 2220, Inp insn: 8fc20000, Inp PC: 80020158, Inp RS: 30, Inp RT: 2, ALU_RESULT: 3
# Time: 2230, Inp insn: 00621823, Inp PC: 8002015c, Inp RS: 3, Inp RT: 2, ALU_RESULT: 1
# Time: 2240, Inp insn: 8fc20004, Inp PC: 80020160, Inp RS: 30, Inp RT: 2, ALU_RESULT: 1
# Time: 2250, Inp insn: 0043102a, Inp PC: 80020164, Inp RS: 2, Inp RT: 3, ALU_RESULT: 1
# Time: 2260, Inp insn: 1440ffd1, Inp PC: 80020168, Inp RS: 2, Inp RT: 0, ALU_RESULT:
    2147877036
# Time: 2270, Inp insn: 00000000, Inp PC: 8002016c, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 2280, Inp insn: 8fc20000, Inp PC: 80020170, Inp RS: 30, Inp RT: 2, ALU_RESULT: 0
# Time: 2290, Inp insn: 24420001, Inp PC: 80020174, Inp RS: 2, Inp RT: 2, ALU_RESULT: 3
# Time: 2300, Inp insn: afc20000, Inp PC: 80020178, Inp RS: 30, Inp RT: 2, ALU_RESULT: 3
# Time: 2310, Inp insn: 8fc30000, Inp PC: 8002017c, Inp RS: 30, Inp RT: 3, ALU_RESULT: 3
# Time: 2320, Inp insn: 8fc2001c, Inp PC: 80020180, Inp RS: 30, Inp RT: 2, ALU_RESULT: 3
# Time: 2330, Inp insn: 0062102a, Inp PC: 80020184, Inp RS: 3, Inp RT: 2, ALU_RESULT: 0
# Time: 2340, Inp insn: 1440ffc5, Inp PC: 80020188, Inp RS: 2, Inp RT: 0, ALU_RESULT:
    2147877020
# Time: 2350, Inp insn: 00000000, Inp PC: 8002018c, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 2360, Inp insn: 03c0e821, Inp PC: 80020190, Inp RS: 30, Inp RT: 0, ALU_RESULT: 30
# Time: 2370, Inp insn: 8fbe0014, Inp PC: 80020194, Inp RS: 29, Inp RT: 30, ALU_RESULT: 30
# Time: 2380, Inp insn: 27bd0018, Inp PC: 80020198, Inp RS: 29, Inp RT: 29, ALU_RESULT: 53
# Time: 2400, Inp insn: 00000000, Inp PC: 800201a0, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# ** Note: $finish : exec_stage_tb.v(129)
# Time: 2420 ns Iteration: 0 Instance: /exec_stage_tb
```

Listing 9: Simple Add ALU Output

```
# Initializing memory
# Initializing Fetch module
# opened file srec_files/SimpleAdd.srec
#
# read in file
#
# Time: 270, Inp insn: 27bdffe8, Inp PC: 80020000, Inp RS: 29, Inp RT: 29, ALU_RESULT: 65541
# Time: 280, Inp insn: afbe0014, Inp PC: 80020004, Inp RS: 29, Inp RT: 30, ALU_RESULT: 65541
# Time: 290, Inp insn: 03a0f021, Inp PC: 80020008, Inp RS: 29, Inp RT: 0, ALU_RESULT: 29
# Time: 300, Inp insn: 24020003, Inp PC: 8002000c, Inp RS: 0, Inp RT: 2, ALU_RESULT: 3
# Time: 310, Inp insn: afc20000, Inp PC: 80020010, Inp RS: 30, Inp RT: 2, ALU_RESULT: 3
# Time: 320, Inp insn: 24020002, Inp PC: 80020014, Inp RS: 0, Inp RT: 2, ALU_RESULT: 2
# Time: 330, Inp insn: afc20004, Inp PC: 80020018, Inp RS: 30, Inp RT: 2, ALU_RESULT: 2
# Time: 340, Inp insn: afc00008, Inp PC: 8002001c, Inp RS: 30, Inp RT: 0, ALU_RESULT: 2
# Time: 350, Inp insn: 8fc30000, Inp PC: 80020020, Inp RS: 30, Inp RT: 3, ALU_RESULT: 2
# Time: 360, Inp insn: 8fc20004, Inp PC: 80020024, Inp RS: 30, Inp RT: 2, ALU_RESULT: 2
# Time: 370, Inp insn: 00621021, Inp PC: 80020028, Inp RS: 3, Inp RT: 2, ALU_RESULT: 5
# Time: 380, Inp insn: afc20008, Inp PC: 8002002c, Inp RS: 30, Inp RT: 2, ALU_RESULT: 5
# Time: 390, Inp insn: 8fc20008, Inp PC: 80020030, Inp RS: 30, Inp RT: 2, ALU_RESULT: 5
# Time: 400, Inp insn: 03c0e821, Inp PC: 80020034, Inp RS: 30, Inp RT: 0, ALU_RESULT: 30
# Time: 410, Inp insn: 8fbe0014, Inp PC: 80020038, Inp RS: 29, Inp RT: 30, ALU_RESULT: 30
# Time: 420, Inp insn: 27bd0018, Inp PC: 8002003c, Inp RS: 29, Inp RT: 29, ALU_RESULT: 53
# Time: 430, Inp insn: 03e00008, Inp PC: 80020040, Inp RS: 31, Inp RT: 0, ALU_RESULT: 53
# Time: 450, Inp insn: 00000000, Inp PC: 80020048, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# ** Note: $finish : exec_stage_tb.v(129)
# Time: 460 ns Iteration: 0 Instance: /exec_stage_tb
```

Listing 10: Simple If ALU Output

```
# Initializing memory
# Initializing Fetch module
# opened file srec_files/SimpleIf.srec
#
# read in file
#
# Time: 440, Inp insn: 27bdffe8, Inp PC: 80020000, Inp RS: 29, Inp RT: 29, ALU_RESULT: 65541
# Time: 450, Inp insn: afbe0014, Inp PC: 80020004, Inp RS: 29, Inp RT: 30, ALU_RESULT: 65541
# Time: 460, Inp insn: 03a0f021, Inp PC: 80020008, Inp RS: 29, Inp RT: 0, ALU_RESULT: 29
# Time: 470, Inp insn: 24020003, Inp PC: 8002000c, Inp RS: 0, Inp RT: 2, ALU_RESULT: 3
# Time: 480, Inp insn: afc20000, Inp PC: 80020010, Inp RS: 30, Inp RT: 2, ALU_RESULT: 3
# Time: 490, Inp insn: 24020002, Inp PC: 80020014, Inp RS: 0, Inp RT: 2, ALU_RESULT: 2
# Time: 500, Inp insn: afc20004, Inp PC: 80020018, Inp RS: 30, Inp RT: 2, ALU_RESULT: 2
# Time: 510, Inp insn: afc00008, Inp PC: 8002001c, Inp RS: 30, Inp RT: 0, ALU_RESULT: 2
# Time: 520, Inp insn: 8fc20000, Inp PC: 80020020, Inp RS: 30, Inp RT: 2, ALU_RESULT: 2
# Time: 530, Inp insn: 28420005, Inp PC: 80020024, Inp RS: 2, Inp RT: 2, ALU_RESULT: 1
# Time: 540, Inp insn: 10400007, Inp PC: 80020028, Inp RS: 2, Inp RT: 0, ALU_RESULT: 1
# Time: 550, Inp insn: 00000000, Inp PC: 8002002c, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 560, Inp insn: 8fc30000, Inp PC: 80020030, Inp RS: 30, Inp RT: 3, ALU_RESULT: 0
# Time: 570, Inp insn: 8fc20004, Inp PC: 80020034, Inp RS: 30, Inp RT: 2, ALU_RESULT: 0
# Time: 580, Inp insn: 00621021, Inp PC: 80020038, Inp RS: 3, Inp RT: 2, ALU_RESULT: 5
# Time: 590, Inp insn: afc20000, Inp PC: 8002003c, Inp RS: 30, Inp RT: 2, ALU_RESULT: 5
# Time: 600, Inp insn: 08008016, Inp PC: 80020040, Inp RS: 0, Inp RT: 0, ALU_RESULT: 5
```

```
# Time: 610, Inp insn: 00000000, Inp PC: 80020044, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 620, Inp insn: 8fc30000, Inp PC: 80020048, Inp RS: 30, Inp RT: 3, ALU_RESULT: 0
# Time: 630, Inp insn: 8fc20004, Inp PC: 8002004c, Inp RS: 30, Inp RT: 2, ALU_RESULT: 0
# Time: 640, Inp insn: 00621023, Inp PC: 80020050, Inp RS: 3, Inp RT: 2, ALU_RESULT: 1
# Time: 650, Inp insn: afc20000, Inp PC: 80020054, Inp RS: 30, Inp RT: 2, ALU_RESULT: 1
# Time: 660, Inp insn: 8fc30000, Inp PC: 80020058, Inp RS: 30, Inp RT: 3, ALU_RESULT: 1
# Time: 670, Inp insn: 8fc20004, Inp PC: 8002005c, Inp RS: 30, Inp RT: 2, ALU_RESULT: 1
# Time: 680, Inp insn: 00621021, Inp PC: 80020060, Inp RS: 3, Inp RT: 2, ALU_RESULT: 5
# Time: 690, Inp insn: afc20008, Inp PC: 80020064, Inp RS: 30, Inp RT: 2, ALU_RESULT: 5
# Time: 700, Inp insn: 8fc20008, Inp PC: 80020068, Inp RS: 30, Inp RT: 2, ALU_RESULT: 5
# Time: 710, Inp insn: 03c0e821, Inp PC: 8002006c, Inp RS: 30, Inp RT: 0, ALU_RESULT: 30
# Time: 720, Inp insn: 8fbe0014, Inp PC: 80020070, Inp RS: 29, Inp RT: 30, ALU_RESULT: 30
# Time: 730, Inp insn: 27bd0018, Inp PC: 80020074, Inp RS: 29, Inp RT: 29, ALU_RESULT: 53
# Time: 740, Inp insn: 03e00008, Inp PC: 80020078, Inp RS: 31, Inp RT: 0, ALU_RESULT: 53
# Time: 760, Inp insn: 00000000, Inp PC: 80020080, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# ** Note: $finish : exec_stage_tb.v(129)
# Time: 770 ns Iteration: 0 Instance: /exec_stage_tb
```

Listing 11: Sum Array ALU Output

```
# Initializing memory
# Initializing Fetch module
# opened file srec_files/SumArray.srec
#
# read in file
#
# Time: 580, Inp insn: 27bdffc8, Inp PC: 80020000, Inp RS: 29, Inp RT: 29, ALU_RESULT: 65509
# Time: 590, Inp insn: afbe0034, Inp PC: 80020004, Inp RS: 29, Inp RT: 30, ALU_RESULT: 65509
# Time: 600, Inp insn: 03a0f021, Inp PC: 80020008, Inp RS: 29, Inp RT: 0, ALU_RESULT: 29
# Time: 610, Inp insn: afc00000, Inp PC: 8002000c, Inp RS: 30, Inp RT: 0, ALU_RESULT: 29
# Time: 620, Inp insn: afc00004, Inp PC: 80020010, Inp RS: 30, Inp RT: 0, ALU_RESULT: 29
# Time: 630, Inp insn: afc00000, Inp PC: 80020014, Inp RS: 30, Inp RT: 0, ALU_RESULT: 29
# Time: 640, Inp insn: 08008010, Inp PC: 80020018, Inp RS: 0, Inp RT: 0, ALU_RESULT: 29
# Time: 650, Inp insn: 00000000, Inp PC: 8002001c, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 660, Inp insn: 8fc20000, Inp PC: 80020020, Inp RS: 30, Inp RT: 2, ALU_RESULT: 0
# Time: 670, Inp insn: 00021080, Inp PC: 80020024, Inp RS: 0, Inp RT: 2, ALU_RESULT: 8
# Time: 680, Inp insn: 03c21021, Inp PC: 80020028, Inp RS: 30, Inp RT: 2, ALU_RESULT: 32
# Time: 690, Inp insn: 8fc30000, Inp PC: 8002002c, Inp RS: 30, Inp RT: 3, ALU_RESULT: 32
# Time: 700, Inp insn: ac430008, Inp PC: 80020030, Inp RS: 2, Inp RT: 3, ALU_RESULT: 32
# Time: 710, Inp insn: 8fc20000, Inp PC: 80020034, Inp RS: 30, Inp RT: 2, ALU_RESULT: 32
# Time: 720, Inp insn: 24420001, Inp PC: 80020038, Inp RS: 2, Inp RT: 2, ALU_RESULT: 3
# Time: 730, Inp insn: afc20000, Inp PC: 8002003c, Inp RS: 30, Inp RT: 2, ALU_RESULT: 3
# Time: 740, Inp insn: 8fc20000, Inp PC: 80020040, Inp RS: 30, Inp RT: 2, ALU_RESULT: 3
# Time: 750, Inp insn: 2842000a, Inp PC: 80020044, Inp RS: 2, Inp RT: 2, ALU_RESULT: 1
# Time: 760, Inp insn: 1440fff5, Inp PC: 80020048, Inp RS: 2, Inp RT: 0, ALU_RESULT:
    2147876892
# Time: 770, Inp insn: 00000000, Inp PC: 8002004c, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 780, Inp insn: afc00000, Inp PC: 80020050, Inp RS: 30, Inp RT: 0, ALU_RESULT: 0
# Time: 790, Inp insn: 08008021, Inp PC: 80020054, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 800, Inp insn: 00000000, Inp PC: 80020058, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 810, Inp insn: 8fc20000, Inp PC: 8002005c, Inp RS: 30, Inp RT: 2, ALU_RESULT: 0
# Time: 820, Inp insn: 00021080, Inp PC: 80020060, Inp RS: 0, Inp RT: 2, ALU_RESULT: 8
```

```
# Time: 830, Inp insn: 03c21021, Inp PC: 80020064, Inp RS: 30, Inp RT: 2, ALU_RESULT: 32
# Time: 840, Inp insn: 8c420008, Inp PC: 80020068, Inp RS: 2, Inp RT: 2, ALU_RESULT: 32
# Time: 850, Inp insn: 8fc30004, Inp PC: 8002006c, Inp RS: 30, Inp RT: 3, ALU_RESULT: 32
# Time: 860, Inp insn: 00621021, Inp PC: 80020070, Inp RS: 3, Inp RT: 2, ALU_RESULT: 5
# Time: 870, Inp insn: afc20004, Inp PC: 80020074, Inp RS: 30, Inp RT: 2, ALU_RESULT: 5
# Time: 880, Inp insn: 8fc20000, Inp PC: 80020078, Inp RS: 30, Inp RT: 2, ALU_RESULT: 5
# Time: 890, Inp insn: 24420001, Inp PC: 8002007c, Inp RS: 2, Inp RT: 2, ALU_RESULT: 3
# Time: 900, Inp insn: afc20000, Inp PC: 80020080, Inp RS: 30, Inp RT: 2, ALU_RESULT: 3
# Time: 910, Inp insn: 8fc20000, Inp PC: 80020084, Inp RS: 30, Inp RT: 2, ALU_RESULT: 3
# Time: 920, Inp insn: 2842000a, Inp PC: 80020088, Inp RS: 2, Inp RT: 2, ALU_RESULT: 1
# Time: 930, Inp insn: 1440fff3, Inp PC: 8002008c, Inp RS: 2, Inp RT: 0, ALU_RESULT:
    2147876952
# Time: 940, Inp insn: 00000000, Inp PC: 80020090, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# Time: 950, Inp insn: 8fc20004, Inp PC: 80020094, Inp RS: 30, Inp RT: 2, ALU_RESULT: 0
# Time: 960, Inp insn: 03c0e821, Inp PC: 80020098, Inp RS: 30, Inp RT: 0, ALU_RESULT: 30
# Time: 970, Inp insn: 8fbe0034, Inp PC: 8002009c, Inp RS: 29, Inp RT: 30, ALU_RESULT: 30
# Time: 980, Inp insn: 27bd0038, Inp PC: 800200a0, Inp RS: 29, Inp RT: 29, ALU_RESULT: 85
# Time: 1000, Inp insn: 00000000, Inp PC: 800200a8, Inp RS: 0, Inp RT: 0, ALU_RESULT: 0
# ** Note: $finish : exec_stage_tb.v(129)
# Time: 1020 ns Iteration: 0 Instance: /exec_stage_tb
```

4 ALU Source Code

Listing 12: ALU

```
//
// alu.v
//
//

`include "control.vh"
`include "alu_func.vh"

module alu (
    clock,
    rsData,
    rtData,
    control,
    outData,
    bt,
    insn,
    pc
);

    input wire clock;
    input wire [0:31] rsData;
    input wire [0:31] rtData;

    input wire [0:`CONTROL_REG_SIZE-1] control;
    wire [0:5] opcode;
```

```
wire [5:0] funct;
wire [0:5] sa;
wire [0:15] immediate;
input wire [0:31] insn;
wire [0:25] insn_index;
input wire [0:31] pc;
wire [0:17] offset;
wire [0:4] rt;


output reg [0:31] outData;
output reg bt; //branch taken

assign opcode = insn[0:5];
assign rt = insn[11:15];
assign sa = insn[20:25];
assign funct = insn[26:31];
assign immediate = insn[16:31];
assign offset = insn[16:31];
assign insn_index = insn[6:31];

always @(posedge clock)
  begin
      bt = 0;

    if (control[`R_TYPE]) begin
    case(funct)
    `ADD:
      //rd - rs + rt
      outData = rsData + rtData;
    `ADDU:
      //todo: do we need to treat unsigned specially?
      outData = $unsigned(rsData) + $unsigned(rtData);
    `SUB:
      outData = rsData - rtData;
    `SUBU:
      //todo: correct way to do unsigned?
      outData = $unsigned(rsData) - $unsigned(rtData);
    `SLT: // rd - 1 if rs  rt; else rd - 0
      if ($signed(rsData)  $signed(rtData)) begin
          outData = 32'h00000001;
      end else begin
          outData = 32'h00000000;
      end
    `SLTU:
       if ($unsigned(rsData)  $unsigned(rtData)) begin
           outData = 32'h0000001;
       end else begin
           outData = 32'h00000000;
       end
    `SLL:
```

```
    outData = rtData  sa;
  `SRL:
    outData = rtData  sa;
  `SRA:
    outData = rtData  sa;
  `AND:
    outData = rsData & rtData;
  `OR:
   outData = rsData | rtData;
  `XOR:
   outData = rsData ^ rtData;
  `NOR:
   outData = ~(rsData | rtData);
 endcase // case (funct)
 end else if(control[`I_TYPE]) begin
     case(opcode)
       `ADDIU:
         outData = rsData + $signed(immediate);
       `SLTI:
         if ($signed(rsData)  $signed(immediate)) begin
              outData = 32'h0000001;
         end else begin
              outData = 32'h0000000;
         end
       `LW:
         ;
       `SW:
         ;
       `LUI:
         outData = immediate  16;
       `ORI:
         outData = rsData | immediate;
     endcase // case (funct)
 end else if (control[`J_TYPE]) begin
 case(opcode)
       `J:
         //outData = insn_index;
         bt = 1'b1;

       `BEQ:
         if (rsData == rtData) begin
             outData = pc + $signed(offset  2);
             bt = 1'b1;
         end else begin
             bt = 1'b0;
         end
       `BNE:
         if (rsData != rtData) begin
             outData = pc + $signed(offset  2);
             bt = 1'b1;
         end else begin
```

```
                bt = 1'b0;
            end
          `BGTZ:
            if ($signed(rsData)  1'b0) begin
                outData = pc + $signed(offset  2);
                bt = 1'b1;
            end else begin
                bt = 1'b0;
            end
          `BLEZ:
            if ($signed(rsData) = 1'b0) begin
                outData = pc + $signed(offset  2);
            end else begin
                bt = 1'b0;
            end

          `REGIMM:
            case(rt)
              `BLTZ:
                if ($signed(rsData)  1'b0) begin
                    outData = pc + $signed(offset  2);
                    bt = 1'b1;
                end else begin
                    bt = 1'b0;
                end
              `BGEZ:
                if ($signed(rsData) = 1'b0) begin
                    outData = pc + $signed(offset  2);
                    bt = 1'b1;
                end else begin
                    bt = 1'b0;
                end
            endcase // case (rtData)

        endcase // case (funct)

    end

  end



endmodule
```

Listing 13: ALU Function Deginitions Header File

```
//
// ALU function definitions
//

`ifndef _alu_func_vh_
```

```
`define _alu_func_vh_

//R-TYPE Instructions

`define ADD 6'b100000
`define ADDU 6'b100001
`define SUB 6'b100010
`define SUBU 6'b100011
`define SLT 6'b101010
`define SLTU 6'b101011
`define SLL 6'b000000
`define SRL 6'b000010
`define SRA 6'b000011
`define AND 6'b100100
`define OR 6'b100101
`define XOR 6'b100110
`define NOR 6'b100111

//I-TYPE instructions
`define ADDIU 6'b001001
`define SLTI 6'b001010
`define LW 6'b100011
`define SW 6'b101011
`define LUI 6'b001111
`define ORI 6'b001101

//J-TYPE instructions
`define J 6'b000010
`define BEQ 6'b000100
`define BNE 6'b000101
`define BGTZ 6'b000111
`define BLEZ 6'b000110
`define REGIMM 6'b000001

`define BLTZ 5'b00000
`define BGEZ 5'b00001

`endif // `ifndef _alu_func_vh_
```

Listing 14: ALU Control Register Constant Definitions Header File

```
//
// Control register constant definitions
//

`ifndef _control_vh_
`define _control_vh_

`define CONTROL_REG_SIZE 4

`define REG_WE 0
`define I_TYPE 1
```

```
`define R_TYPE 2
`define J_TYPE 3

`endif
```

Listing 15: ALU Unit Test Bench

```
//
// alu_tb.v
//
// ALU test bench
//

`include "control.vh"
`include "alu_func.vh"

module alu_tb;
    reg clock;
    reg[0:31] rsData;
    reg[0:31] rtData;
    reg[0:`CONTROL_REG_SIZE-1] control;
    reg[0:5] funct;
    reg[0:5] sa;
    reg[0:15] immediate;
    reg[0:31] insn;
    wire [0:31] aluOutput;
    wire bt;
    reg [0:31] pc;

    alu DUT(
        .clock (clock),
            .rsData (rsData),
        .rtData (rtData),
            .control (control),
            .outData (aluOutput),
            .bt (bt),
            .insn (insn),
            .pc (pc)
    );

 initial begin
        clock = 1;
    end

    always begin
        #5 clock = !clock;
    end

    event terminate_sim;
    initial begin
        @ (terminate_sim);
        #10 $finish;
```

```
end

initial begin

    @ (posedge clock);
    control[`R_TYPE] = 1'b1;
    control[`I_TYPE] = 1'b0;
    pc = 0;
    rsData = 32'h00000005;
    rtData = 32'h00000002;

    //arithmetic function set to ADD
    insn = 32'h00000000 | `ADD;
    @ (posedge clock);
    @ (posedge clock);
    $display("ALU output for 5 + 2: %d", aluOutput);

    rsData = -32'h00000002;
    rtData = 32'h00000002;
    //arithmetic function set to ADD
    insn = 32'h00000000 | `ADD;
    @ (posedge clock);
    @ (posedge clock);
    $display("ALU output for %d + %d : %d",$signed(rsData), rtData, aluOutput);

    rsData = -32'h00000002;
    rtData = 32'h00000001;
    //arithmetic function set to ADDU
    insn = 32'h00000000 | `ADDU;
    @ (posedge clock);
    @ (posedge clock);
    $display("ALU output for %d + %d (unsigned): %d",rsData, rtData, aluOutput);

    rsData = 32'h00000004;
    rtData = 32'h00000002;
    insn = 32'h00000000 | `SUB;
    @ (posedge clock);
    @ (posedge clock);
    $display("ALU output for %d - %d : %d",rsData, rtData, aluOutput);

    rsData = -32'h00000002;
    rtData = 32'h00000007;
    insn = 32'h00000000 | `SUBU;
    @ (posedge clock);
    @ (posedge clock);
    $display("ALU output for %d - %d (unsigned): %d",rsData, rtData, aluOutput);

    rsData = -32'h00000002;
    rtData = 32'h00000007;
    insn = 32'h00000000 | `SLT;
    @ (posedge clock);
```

```
@ (posedge clock);
$display("ALU output for %d  %d : %d", $signed(rsData), rtData, aluOutput);

rsData = -32'h00000002;
rtData = 32'h00000007;
insn = 32'h00000000 | `SLTU;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for %d  %d (unsigned): %d",rsData, rtData, aluOutput);

rsData = 32'h00000002;
rtData = 32'h00000002;
sa = 5'b00010;

insn = 32'h00000000 | `SLL | (5'b00010  6);
@ (posedge clock);
@ (posedge clock);
$display("ALU output for %d  %d: %d", rtData, sa, aluOutput);

rsData = 32'h00000002;
rtData = 32'h00000002;
sa = 5'b00010;
insn = 32'h00000000 | `SRL | (5'b00010  6);
@ (posedge clock);
@ (posedge clock);
$display("ALU output for %d  %d: %d", rtData, sa, aluOutput);

rsData = 32'h00000002;
rtData = 32'h00000001;
sa = 5'b00010;
insn = 32'h00000000 | `SRA | (5'b00010  6);
@ (posedge clock);
@ (posedge clock);
$display("ALU output for %d  %d: %d", rtData, sa, aluOutput);

rsData = 32'h00040a02;
rtData = 32'hffffffff;
insn = 32'h00000000 | `AND;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for %h AND %h: %h", rsData, rtData, aluOutput);

rsData = 32'h00040a02;
rtData = 32'hffff0000;
insn = 32'h00000000 | `OR;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for %h OR %h: %h", rsData, rtData, aluOutput);

rsData = 32'h00040a02;
rtData = 32'hffffffff;
```

```
insn = 32'h00000000 | `XOR;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for %h XOR %h: %h", rsData, rtData, aluOutput);

rsData = 32'h00040a02;
rtData = 32'hffffffff;
insn = 32'h00000000 | `NOR;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for %h NOR %h: %h", rsData, rtData, aluOutput);

control[`I_TYPE] = 1'b1;
control[`R_TYPE] = 1'b0;

rsData = 32'h00000008;
immediate = 16'h0007;
insn = 32'h00000000 | (`ADDIU  26) | 16'h0007;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for %d + %d: %d (immediate unsigned)", rsData, immediate,
    aluOutput);

rsData = 32'h00000008;
immediate = 16'h0007;
insn = 32'h00000000 | (`SLTI  26) | 16'h0007;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for %d  %d: %d (immediate)", rsData, immediate, aluOutput);

rsData = 32'h00000007;
immediate = 16'h0008;
insn = 32'h00000000 | (`SLTI  26) | 16'h0008;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for %d  %d: %d (immediate)", rsData, immediate, aluOutput);


immediate = 16'h0007;
insn = 32'h00000000 | (`LUI  26) | 16'h0007;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for load upper immediate: %h - %h ",immediate, aluOutput);

rsData = 32'h0000ffff;
immediate = 16'h0007;
insn = 32'h00000000 | (`ORI  26) | 16'h0007;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for %h ORI %h: %h (immediate)", rsData, immediate, aluOutput);
```

```
control[`I_TYPE] = 1'b0;
control[`R_TYPE] = 1'b0;
control[`J_TYPE] = 1'b1;

insn = 32'h00000000 | (`J  26) | 32'h000000ff;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for J %h", aluOutput);
$display("Branch taken: (1 = yes, 0 = no) %d", bt);

rsData = 32'h00000001;
rtData = 32'h00000001;
insn = 32'h00000000 | (`BEQ  26) | 32'h0000dead;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for BEQ %h", aluOutput);
$display("Branch taken: (1 = yes, 0 = no) %d", bt);

rsData = 32'h00000101;
rtData = 32'h00000001;
insn = 32'h00000000 | (`BEQ  26) | 32'h0000dead;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for BEQ %h", aluOutput);
$display("Branch taken: (1 = yes, 0 = no) %d", bt);

       rsData = 32'h00000001;
rtData = 32'h00000001;
insn = 32'h00000000 | (`BNE  26) | 32'h0000dead;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for BNE %h", aluOutput);
$display("Branch taken: (1 = yes, 0 = no) %d", bt);

rsData = 32'h00000101;
rtData = 32'h00000001;
insn = 32'h00000000 | (`BNE  26) | 32'h0000dead;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for BNE %h", aluOutput);
$display("Branch taken: (1 = yes, 0 = no) %d", bt);

rsData = 32'h00000101;
insn = 32'h00000000 | (`BGTZ  26) | 32'h0000beef;
@ (posedge clock);
@ (posedge clock);
$display("ALU output for BGTZ %h (taken)", aluOutput);
$display("Branch taken: (1 = yes, 0 = no) %d", bt);

rsData = 32'h00000101;
insn = 32'h00000000 | (`BLEZ  26) | 32'h0000beef;
```

```
        @ (posedge clock);
        @ (posedge clock);
        $display("ALU output for BLEZ %h (not taken)", aluOutput);
        $display("Branch taken: (1 = yes, 0 = no) %d", bt);


        rsData = 32'h00000101;
        insn = 32'h00000000 | (`REGIMM  26) | 32'h0000beef;
        @ (posedge clock);
        @ (posedge clock);
        $display("ALU output for BLTZ %h (not taken)", aluOutput);
        $display("Branch taken: (1 = yes, 0 = no) %d", bt);

        rsData = 32'h00000101;
        insn = 32'h00000000 | (`REGIMM  26) | (`BGEZ  16) | 32'h0000beef;
        @ (posedge clock);
        @ (posedge clock);
        $display("ALU output for BGEZ %h (taken)", aluOutput);
        $display("Branch taken: (1 = yes, 0 = no) %d", bt);



        - terminate_sim;
    end


endmodule
```

Listing 16: Execute Stage Test Bench

```
//
// exec_stage_tb.v
//
// Testbench for the execute stage
// includes fetch, decode, and alu
//

module exec_stage_tb;
    reg clock;

    wire[0:31] address;
    wire wren;
    wire[0:31] data_in;
    wire[0:31] data_out;

    wire[0:31] fetch_address;
    wire fetch_wren;
    wire[0:31] fetch_data_in;
    wire[0:31] fetch_data_out;

    wire[0:31] fetch_insn_decode;
    wire[0:31] fetch_pc;
```

```
reg fetch_stall;

wire[0:31] decode_rs_data;
wire[0:31] decode_rt_data;
wire[4:0] decode_rd_in;
wire[0:31] decode_pc_out;
wire[0:31] decode_ir_out;
wire[0:31] decode_write_back_data;
wire decode_reg_write_enable;
wire [0:`CONTROL_REG_SIZE-1] decode_control;

wire[0:31] srec_address;
wire srec_wren;
wire[0:31] srec_data_in;
wire[0:31] srec_data_out;

wire srec_done;

reg[0:31] tb_address;
reg tb_wren;
reg[0:31] tb_data_in;
wire[0:31] tb_data_out;

wire[0:31] bytes_read;
integer byte_count;
integer read_word;
reg[0:31] fetch_word;
reg instruction_valid;

// alu
reg[0:31] alu_rs_data_res;
reg[0:31] alu_rt_data_res;
reg[0:31] alu_insn;
reg[0:31] alu_insn_res;
wire [0:31] alu_output;
wire bt;
reg [0:31] alu_pc_res;
reg [0:`CONTROL_REG_SIZE-1] alu_control_res;

mem_controller mcu(
    .clock (clock),
    .address (address),
    .wren (wren),
    .data_in (data_in),
    .data_out (data_out)
);

fetch DUT(
    .clock (clock),
    .address (fetch_address),
    .insn (fetch_data_in),
```

```
    .insn_decode (fetch_data_out),
    .pc (fetch_pc),
    .wren (fetch_wren),
    .stall (fetch_stall)
);

srec_parser #("srec_files/BubbleSort.srec") U0(
    .clock (clock),
    .mem_address (srec_address),
    .mem_wren (srec_wren),
    .mem_data_in (srec_data_in),
    .mem_data_out (srec_data_out),
    .done (srec_done),
    .bytes_read(bytes_read)
);

decode U1(
    .clock (clock),
    .insn (fetch_data_out),
        .insn_valid (instruction_valid),
    .pc (fetch_address),
    .rsData (decode_rs_data),
    .rtData (decode_rt_data),
    .rdIn (decode_rd_in),
    .pcOut (decode_pc_out),
    .irOut (decode_ir_out),
    .writeBackData (decode_write_back_data),
    .regWriteEnable (decode_reg_write_enable),
    .control (decode_control)
);

alu alu(
    .clock (clock),
        .rsData (decode_rs_data),
    .rtData (decode_rt_data),
        .control (decode_control),
        .outData (alu_output),
        .bt (alu_bt),
        .insn (alu_insn),
        .pc (decode_pc_out)
);



assign address = srec_done ? (fetch_stall ? tb_address : fetch_address) : srec_address;
assign wren = srec_done ? (fetch_stall ? tb_wren : fetch_wren) : srec_wren;
assign tb_data_out = data_out;
assign fetch_data_in = data_out;
assign data_in = srec_done ? (fetch_stall ? tb_data_in : fetch_data_in) : srec_data_in;
```

```
// Specify when to stop the simulation
event terminate_sim;
initial begin
    @ (terminate_sim);
    #10 $finish;
end

initial begin
    clock = 1;
    fetch_stall = 1;
    instruction_valid = 1'b0;
end

always begin
    #5 clock = !clock;
end

initial begin
    $dumpfile("exec_stage_tb.vcd");
    $dumpvars;
end


always @(posedge clock) begin
    alu_insn = fetch_data_out;
    alu_insn_res = alu_insn;
    alu_control_res = decode_control;
    alu_pc_res = decode_pc_out;
    alu_rs_data_res = decode_rs_data;
    alu_rt_data_res = decode_rt_data;
end


initial begin
    @(posedge srec_done);
    @(posedge clock);
    byte_count = bytes_read+4;
    tb_address = 32'h8002_0000;
    tb_wren = 1'b0;
    instruction_valid = 1'b0;
    fetch_stall = 0;

    while (byte_count  0) begin
        @(posedge clock);
        if ((fetch_address - 4)  32'h8002_0000) begin
            instruction_valid = 1'b0;
        end else begin
            instruction_valid = 1'b1;
        end
            read_word = tb_data_out;
        fetch_word = fetch_data_out;
```

```
            tb_address = tb_address + 4;
            byte_count = byte_count - 4;
        end

        // The decode runs one clock cycle behind the fetch
        @(posedge clock);
        tb_address = tb_address + 4;
        instruction_valid = 1'b0;

        // allow the last alu op to run
        @(posedge clock);
        @(posedge clock);
        -terminate_sim;
    end // initial begin

    // ALU Process
    initial begin
        @(posedge srec_done);
        @(posedge clock);

        while (byte_count  0) begin
            @(posedge clock);

            $display("Time: %d, Inp insn: %X, Inp PC: %X, Inp RS: %d, Inp RT: %d, ALU_RESULT:
                %d",
                    $time, alu_insn_res, alu_pc_res, alu_rs_data_res, alu_rt_data_res,
                        alu_output);

        end
        // allow the last decode to run
        @(posedge clock);
        @(posedge clock);
        $display("Time: %d, Inp insn: %X, Inp PC: %X, Inp RS: %d, Inp RT: %d, ALU_RESULT: %d",
                $time, alu_insn_res, alu_pc_res, alu_rs_data_res, alu_rt_data_res, alu_output)
                    ;
    end

endmodule
```