|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class: | **CPE 300L – Digital System Architecture and Design - 1001** | | | Semester: | **Fall 2017** |
|  | | | | | |
| Points |  | Document author: | **Shabrya Lott** | | |
|  | Author's email: | [**lotts2@unlv.nevada.edu**](mailto:lotts2@unlv.nevada.edu) | | |
|  | | | |
| Document topic: | **Final** | | |
| Instructor's comments: | | | | | |

1. **Control Unit Design**

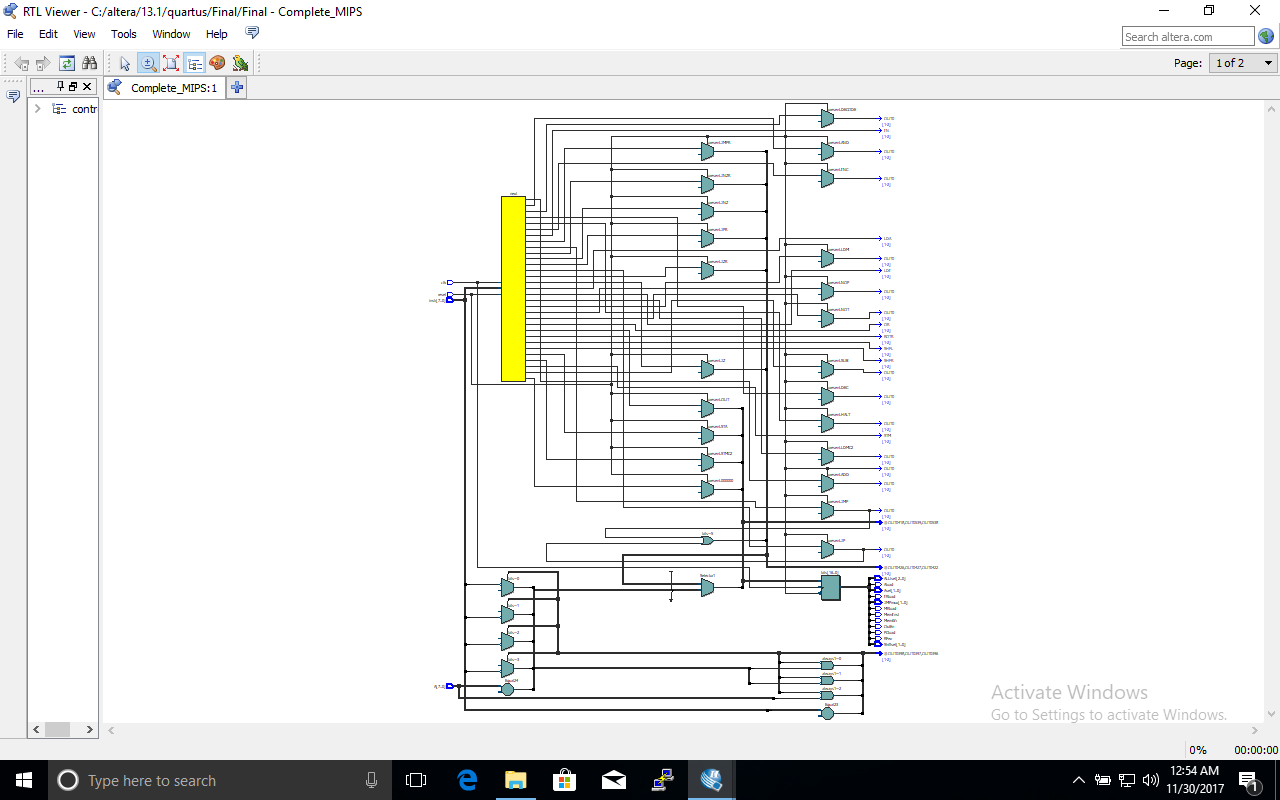


Figure 1: RTL Viewer

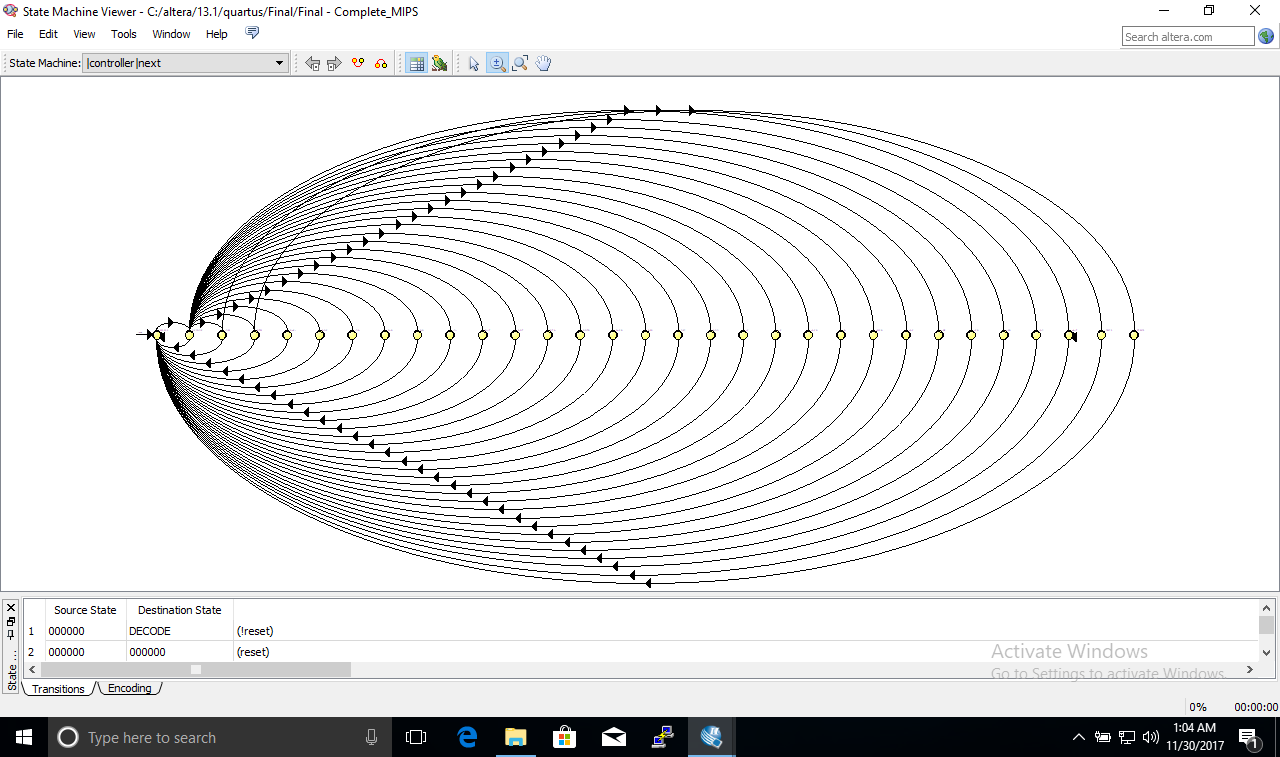


Figure 2: FSM Schematic

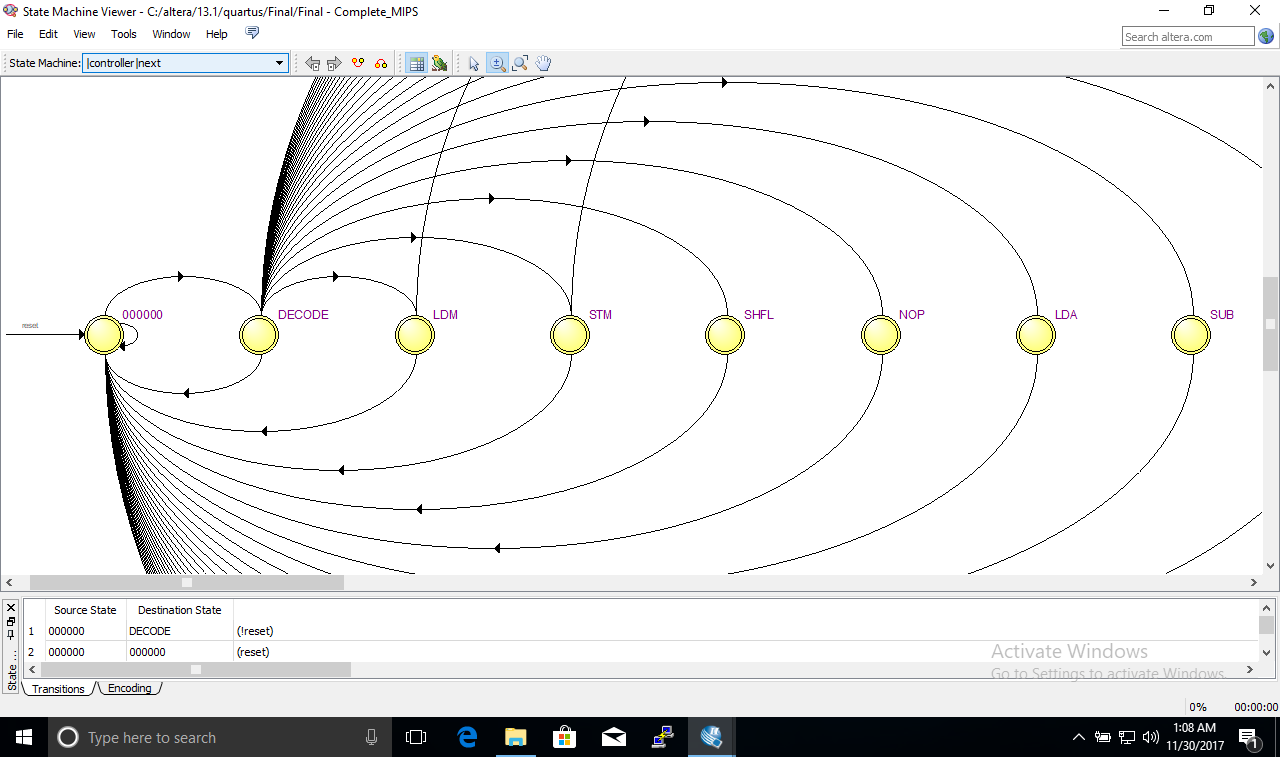


Figure 3: FSM Schematic Zoomed In

1. **Complete Design**

//top

module Complete\_MIPS(input clk, reset,

input [7:0] IN,

output [7:0] writedata,

output memWr,output[7:0] OUT,output [7:0] data,

output [5:0] addr,output Aload);

wire [5:0] address;

wire [7:0] readdata;

assign data = readdata;

assign addr = address;

my\_cpu PROCESSOR(clk,reset,address,memWr,writedata,readdata,IN,OUT,Aload);

dmem dmem(clk,memWr,address,writedata,readdata);

endmodule

module dmem(input clk,we,

input [5:0]addr,

input [7:0]d,

output[7:0] rd);

reg [7:0] RAM [63:0];//creat an array that ia 64 by 8

initial begin

$readmemh("memfile.dat",RAM);

end

always @(posedge clk)

if(we)//WRITE

RAM[addr] <= d;

assign rd = RAM[addr];

endmodule

//CPU

module my\_cpu(input clk,reset,

output[5:0] address,

output memWr,

output[7:0] writedata,

input [7:0] readdata,IN,

output [7:0] OUT,

output Aload

);

wire IRload,PCload,MRload,RFwr,OutEn;

wire MemInst;

wire [1:0] Jmpmuxsel,Shiftsel;

wire [1:0] Asel;

wire [2:0] ALUsel;

wire [7:0] instr;

controller fsm(clk,reset,instr,writedata,memWr,Jmpmuxsel,Shiftsel,Asel,IRload,MRload,PCload,

MemInst,Aload,RFwr,OutEn,ALUsel);

datapath DP(clk,reset,readdata,IRload,Jmpmuxsel,PCload,MemInst,MRload,

Asel,Aload,RFwr,ALUsel,Shiftsel,OutEn,IN,writedata,OUT,instr,address);

endmodule

//Control Unit

module controller(input clk,reset,

input [7:0] instr,

input [7:0] A,

output MemWr,

output [1:0] JMPmux,Shiftsel,Asel,

output IRload,MRload,PCload,MemInst,Aload,RFwr,OutEn,

output [2:0] ALUsel

);

reg [5:0] present;

reg [5:0] next;

reg [16:0] bits;

initial

begin

next <= 6'd0;

present <= 6'd0;

bits <= 17'd0;

end

localparam FETCH = 6'd0,

DECODE = 6'd1,

NOP = 6'd2,

LDA = 6'd3,

STA = 6'd4,

LDM = 6'd5,

STM = 6'd6,

LDI = 6'd7,

AND = 6'd8,

OR = 6'd9,

ADD = 6'd10,

SUB = 6'd11,

NOT = 6'd12,

INC = 6'd13,

DEC = 6'd14,

SHFL = 6'd15,

SHFR = 6'd16,

ROTR = 6'd17,

IN = 6'd18,

OUT = 6'd19,

HALT = 6'd20,

JMP = 6'd21,

JMPR = 6'd22,

JZ = 6'd23,

JZR = 6'd24,

JNZ = 6'd25,

JNZR = 6'd26,

JP = 6'd27,

JPR = 6'd28,

LDMC2 = 6'd29,

STMC2 = 6'd30;

always@(next)

begin

if(reset)

present = 6'd0;

else

present = next;

end

always@(posedge clk)

begin

//next = present;

if(reset)

begin

next = FETCH;

end

else

begin

case(present)

FETCH:

begin

bits = 17'b10001000000000000;

next = DECODE;

end

DECODE:

begin

bits = 17'd0;

if(instr == 8'b0000\_0000)

next = NOP;

else if(instr[7:4] == 4'b0001)

next = LDA;

else if(instr[7:4] == 4'b0010)

next = STA;

else if(instr[7:4] == 4'b0011)

next = LDM;

else if(instr[7:4] == 4'b0100)

next = STM;

else if(instr[7:4] == 4'b0101)

next = LDI;

else if(instr[7:4] == 4'b1010)

next = AND;

else if(instr[7:4] == 4'b1011)

next = OR;

else if(instr[7:4] == 4'b1100)

next = ADD;

else if(instr[7:4] == 4'b1101)

next = SUB;

else if(instr == 8'b1110\_0000)

next = NOT;

else if(instr == 8'b1110\_0001)

next = INC;

else if(instr == 8'b1110\_0010)

next = DEC;

else if(instr == 8'b1110\_0011)

next = SHFL;

else if(instr == 8'b1110\_0100)

next = SHFR;

else if(instr == 8'b1110\_0101)

next = ROTR;

else if(instr == 8'b1111\_0000)

next = IN;

else if(instr == 8'b1111\_0001)

next = OUT;

else if(instr == 8'b1111\_0010)

next = HALT;

else if(instr[7:4] == 4'b0110)

begin

if(instr[3:0] == 4'd0)

next = JMP;

else

next = JMPR;

end

else if(instr[7:4] == 4'b0111)

begin

if(instr[3:0] == 4'd0)

next = JZ;

else

next = JZR;

end

else if(instr[7:4] == 4'b1000)

begin

if(instr[3:0] == 4'd0)

next = JNZ;

else

next = JNZR;

end

else if(instr[7:4] == 4'b1001)

begin

if(instr[3:0]== 4'd0)

next = JP;

else

next = JPR;

end

else

next = FETCH;

end

NOP:

begin

bits = 17'b00000000000000000;

next = FETCH;

end

LDA:

begin

bits = 17'b00000000110000000;

next = FETCH;

end

STA:

begin

bits = 17'b00000000001000000;

next = FETCH;

end

LDM:

begin

bits = 17'b01001000000000000;

next = LDMC2;

end

STM:

begin

bits = 17'b01001000000000000;

next = STMC2;

end

LDI:

begin

bits = 17'b00001001110000000;

next = FETCH;

end

AND:

begin

bits = 17'b00000000010001000;

next = FETCH;

end

OR:

begin

bits = 17'b00000000010010000;

next = FETCH;

end

ADD:

begin

bits = 17'b00000000010100000;

next = FETCH;

end

SUB:

begin

bits = 17'b00000000010101000;

next = FETCH;

end

NOT:

begin

bits = 17'b00000000010011000;

next = FETCH;

end

INC:

begin

bits = 17'b00000000010110000;

next = FETCH;

end

DEC:

begin

bits = 17'b00000000010111000;

next = FETCH;

end

SHFL:

begin

bits = 17'b00000000010000010;

next = FETCH;

end

SHFR:

begin

bits = 17'b00000000010000100;

next = FETCH;

end

ROTR:

begin

bits = 17'b00000000010000110;

next = FETCH;

end

IN:

begin

bits = 17'b00000001010000000;

next = FETCH;

end

OUT:

begin

bits = 17'b00000000000000001;

next = FETCH;

end

HALT:

begin

bits = 17'd0;

next = HALT;

end

LDMC2:

begin

bits = 17'b00000101110000000;

next = FETCH;

end

STMC2:

begin

bits = 17'b00000110000000000;

next = FETCH;

end

JMP:

begin

bits = 17'b00011000000000000;

next = FETCH;

end

JMPR:

begin

if(instr[3:0] == 4'd0)

bits = 17'd0;

else

begin

if(instr[3] == 1'd0)//pos

bits = 17'b00111000000000000;

else//neg

bits = 17'b00101000000000000;

end

next = FETCH;

end

JZ:

begin

if(A == 8'b0)

bits = 17'b00011000000000000;

else //zero != 0 so PC+1

bits = 17'b00001000000000000;

next = FETCH;

end

JZR:

begin

if(A == 8'b0 && instr[3:0]!= 4'd0)

begin

if(instr[3] == 1'd0)//pos

bits = 17'b00111000000000000;

else//neg

bits = 17'b00101000000000000;

end

else//zero!=0 or smmm == 0 thus NOP

bits =17'd0;

next = FETCH;

end

JNZ:

begin

if(A != 8'b0)

bits = 17'b00011000000000000;

else

bits = 17'b00001000000000000;

next = FETCH;

end

JNZR:

begin

if(A!=8'b0 && instr[3:0] != 4'd0)

begin

if(instr[3] == 1'd0)//pos

bits = 17'b00111000000000000;

else//neg

bits = 17'b00101000000000000;

end

else //the top instruction was false

bits = 17'd0;

next = FETCH;

end

JP:

begin

if(A[7] == 1'd0 && A !=8'd0 )//A is positive

bits = 17'b00011000000000000;

else // A is neg

bits = 17'b00011000000000000;

next = FETCH;

end

JPR:

begin

if(A[7] == 1'd0 && instr[3:0] != 4'd0)

begin

if(instr[3] == 1'd0)//pos

bits = 17'b00111000000000000;

else//neg

bits = 17'b00101000000000000;

end

else //the top instruction was false

bits = 17'd0;

next = FETCH;

end

endcase

end

end

assign IRload = bits[16];

assign MRload = bits[15];

assign JMPmux = bits[14:13];

assign PCload = bits[12];

assign MemInst = bits[11];

assign MemWr = bits[10];

assign Asel = bits[9:8];

assign Aload = bits [7];

assign RFwr = bits [6];

assign ALUsel = bits [5:3];

assign Shiftsel = bits [2:1];

assign OutEn = bits[0];

endmodule

//Datapath

module datapath(input clk,reset,

input [7:0] readdata,

input IRload,

input [1:0] Jmpmuxsel,

input PCload,

input MemInst,

input MRload,

input [1:0] Asel,

input Aload,

input RFwr,

input [2:0] ALUsel,

input [1:0] Shiftsel,

input outen,

input [7:0] IN,

output [7:0] accout,

output [7:0]OUT,

output [7:0] INSTR,

output [5:0] address//being inputed to the RAM

);

wire [7:0] result,rfout;

//result hold the result from the SHIFTREG

wire [7:0] aluout;

wire [7:0] amux\_out;//ACC mux out

//accout hold the output of the accumlator

wire [5:0] MAout;

wire [5:0] PCout,pos\_sign,neg\_sign,PCplusone;

wire [5:0] JMPmuxOut;

//PC logic

flopr #(8) IR (clk,reset,IRload,readdata,INSTR);

flopr #(6) MA (clk,reset,MRload,readdata[5:0],MAout);

flopr #(6) PC(clk,reset,PCload,JMPmuxOut,PCout);

relplus pos(PCout,INSTR[5:0],pos\_sign);

relminus min(PCout,INSTR[5:0],neg\_sign);

pcplus pcplusone(PCout,PCplusone);

jump\_mux #(6) JMPmux(PCplusone,readdata[5:0],neg\_sign,pos\_sign,Jmpmuxsel,JMPmuxOut);

mux2 #(6) MemMux(PCout,MAout,MemInst,address);

//Accumulator logic

acc\_mux #(8) accmux4(result,rfout,IN,readdata,Asel,amux\_out);

flopr #(8) accumlator(clk,reset,Aload,amux\_out,accout);

regfile RF(accout,INSTR[2:0],RFwr,clk,rfout);

alu ALU(accout,rfout,ALUsel,aluout);

Shift\_Reg shifter(aluout,Shiftsel,result);

output\_reg OR(clk,reset,outen,accout,OUT);

endmodule

//flopr

module flopr # (parameter WIDTH = 8)

(input clk, reset,load,

input [WIDTH-1:0] d,

output [WIDTH-1:0] q);

reg [WIDTH-1:0] data;

assign q = data;

always @ (posedge clk)

begin

if(reset)

data <= 0;

else if(load)

data <= d;

end

endmodule

/relplus

module relplus(input[5:0] a,b,

output[5:0] OUT);

assign OUT = a + {3'b000,b[2:0]};

endmodule

/PCminus

module relminus(input[5:0]a,b,

output[5:0] OUT);

assign OUT = a - {3'b000,b[2:0]};

endmodule

//PCplus

module pcplus(input[5:0] d,

output [5:0] q);

assign q = d + 6'd1;

endmodule

//Jump Mux

module jump\_mux #(parameter N = 6)

(input[N-1:0] d0,d1,d2,d3,

input [1:0] sel,

output reg [N-1:0] out);

always@(\*)

begin

case(sel)

2'b00: out <= d0;

2'b01: out <= d1;

2'b10: out <= d2;

2'b11: out <= d3;

default: out <= d1;

endcase

end

endmodule

//Mux

module mux2 # (parameter WIDTH = 8)

(input [WIDTH-1:0] d0, d1, input s,

output [WIDTH-1:0] y);

assign y = s ? d1 : d0;

endmodule

//Accumulator Mux

module acc\_mux #(parameter N = 8)

(input [N-1:0]d0,d1,d2,d3,

input [1:0] sel,

output reg [N-1:0] out);

always@(\*)

begin

case(sel)

2'b00: out <= d0;

2'b01: out <= d1;

2'b10: out <= d2;

2'b11: out <= d3;

default: out <= d0;

endcase

end

endmodule

//Register File

module regfile(input[7:0] D,input[2:0] address ,input we,clock,

output [7:0] out);

reg [7:0] register [7:0]; //instantiate an array of size 8 and which can hold word of length 8 bits

//If we wish to write into the register file

always@(posedge clock)

begin

if(we)

begin

register[address] <= D;

end

end

//outputs at every clock cycle

assign out = register[address];

endmodule

//ALU

module alu (a,b,sel, out);

input [7:0] a,b;

input [2:0] sel;

output reg [7:0] out;

initial

begin

out = 0;

end

always @ (\*)

begin

case(sel)

3'b000: out = a;

3'b001: out = a & b;

3'b010: out = a | b;

3'b011: out = ~a;

3'b100: out = a + b;

3'b101: out = a - b;

3'b110: out = a + 8'd1;

3'b111: out = a - 8'd1;

default: out = a;

endcase

end

endmodule

//Shift Register

module Shift\_Reg (a,sel,out);

input [7:0] a;

input [1:0] sel;

output reg [7:0] out;

initial

begin

out = 0;

end

always@(\*)

begin

case(sel)

2'b00:out = a;

2'b01:out = a << 1;

2'b10:out = a >> 1;

2'b11:out = {a[0],a[7:1]};

default: out = a;

endcase

end

endmodule

//Output Reg

module output\_reg(clk,reset,load,d,q);

input clk,reset,load;

input [7:0]d;

output [7:0]q;

flopr #(8) OUTREG(clk,reset,load,d,q);

endmodule

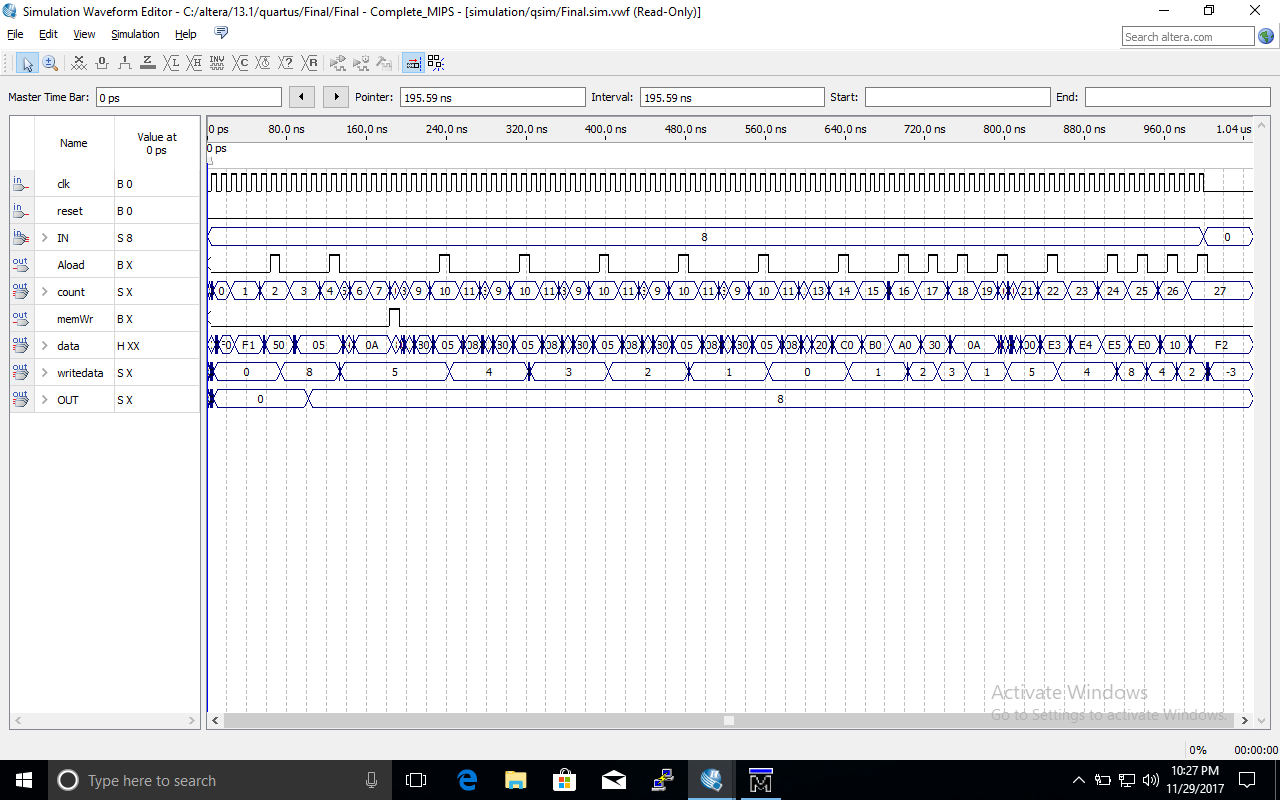


Figure 1. Waveform for memfile.dat

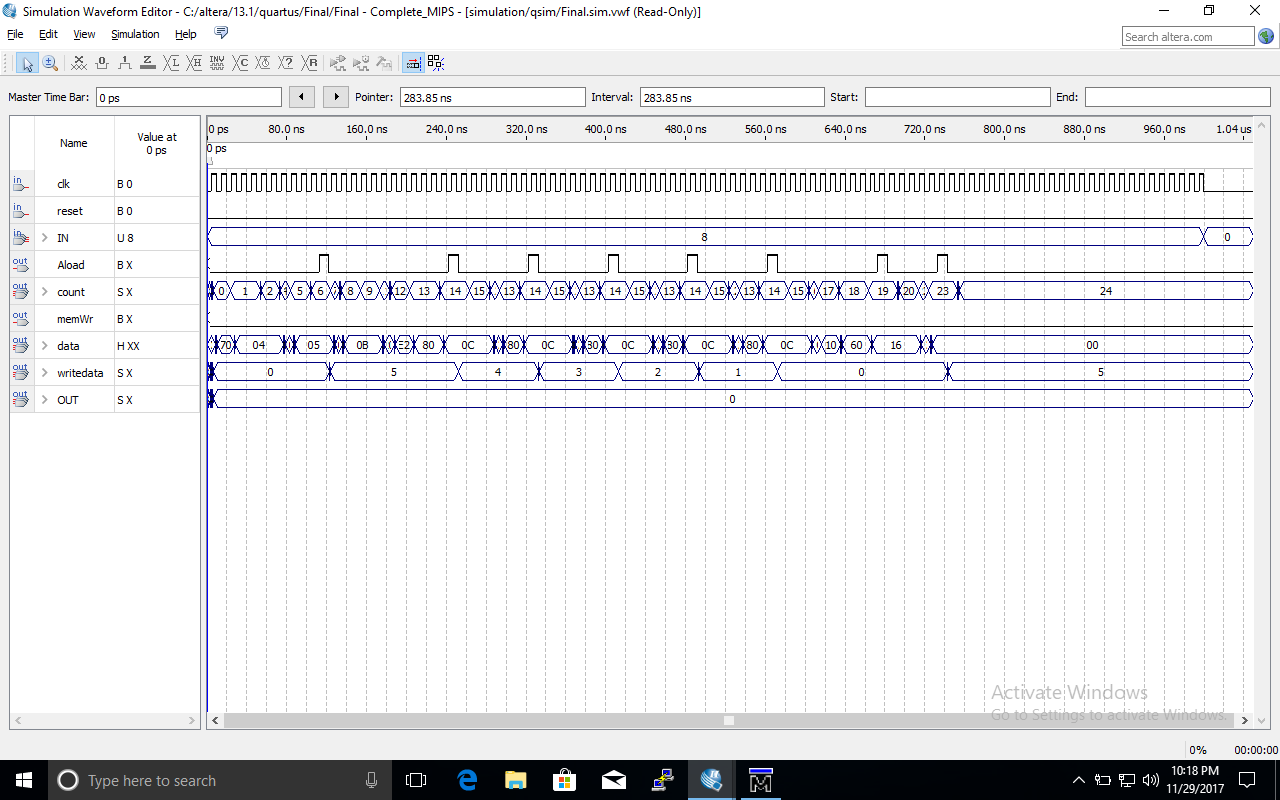


Figure 2. Waveform for memfile2.dat

1. **Testbench along with Modelsim Simulations**

There were 27 distinct instructions to be tested. Two different memfiles were created to test different instructions. The testbench generates the clock, assigns a value for IN and ends when the last instructions have been fully executed.

module test;

reg clk;

reg reset;

reg [7:0] IN;

wire [7:0] Accumulator;

wire MemWr;

wire [7:0] OUT;

wire [7:0] instr;

wire [5:0] address;

wire Aload;

// instantiate device to be tested

Complete\_MIPS U0(clk,reset,IN,Accumulator,MemWr,OUT,instr,address,Aload);

// initialize test

initial

begin

reset <= 1; # 22; reset <= 0;

IN <= 8'd8;

end

always

begin

clk <= 1;

# 10;

clk <= 0;

# 10; // clock duration

end

//check results

always@(negedge clk)

begin

if(address == 6'd29)begin //this line used for memfile.dat

// if(address == 6'd29)begin //this line used for memfile2.dat

$display("Test Finished");

$stop;

end

end

endmodule

In “memfile.dat” the following instructions were executed:

000000: In A //A <- 8

000001: Out A //Output A

000010: LDI A, 5 //A <- 5

000100: STM 001010, A //M[10] <- A = 5

000110: DEC A //A<- A -1

000111: JNZ 000101 //PC<-000101 if A != 0

001001: INC A //A <- 1

001010: STA 0, A //Reg[0]<- 1

001011: ADD A, 0 //A<- A + R[0] = 2

001111: OR A, 0 //A<- A or R[0] = 3

010000: AND A,0 //A <- A and Reg[0] = 1

010001: LDM A, 0 //A<- M[10] = 5

010011: SUB A,0 //A<- A – R[0] = 4

010100: NOP //no operation

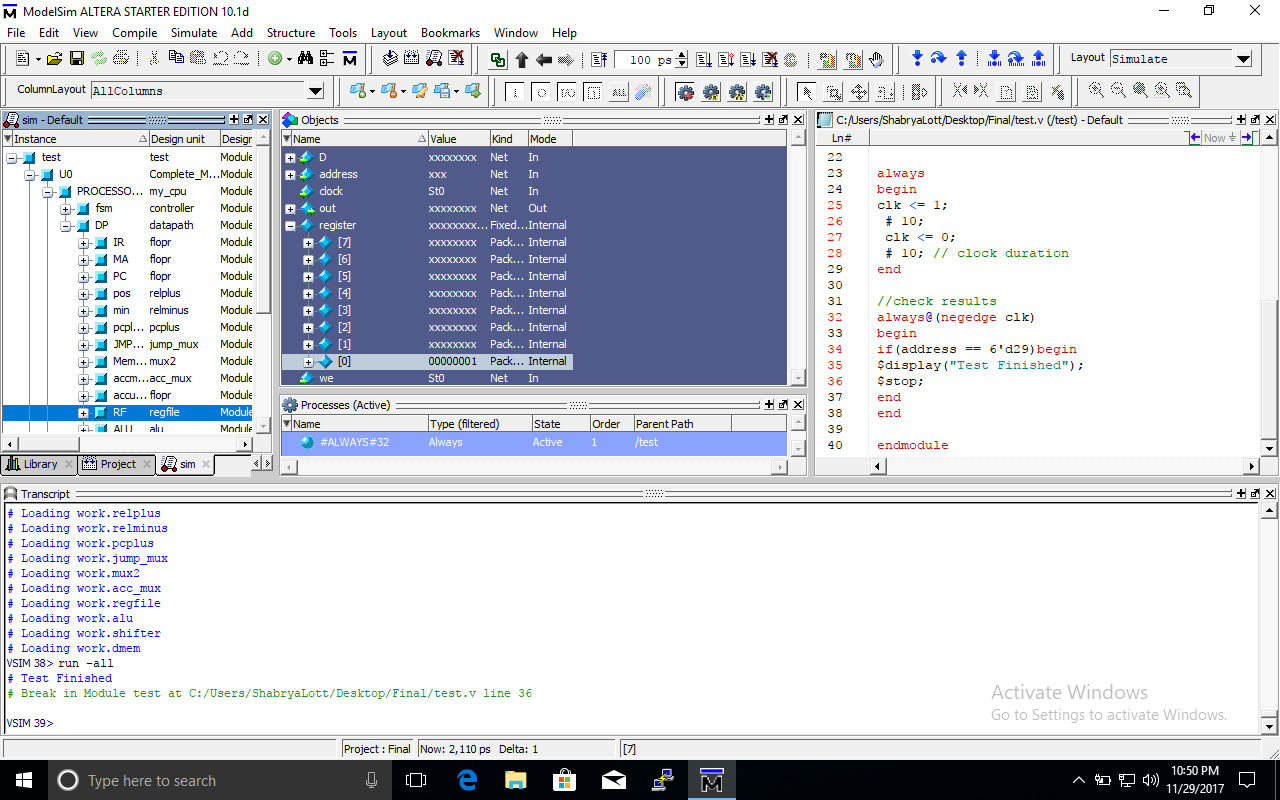
010101: SHFL A //A=<- A << 1 = 8

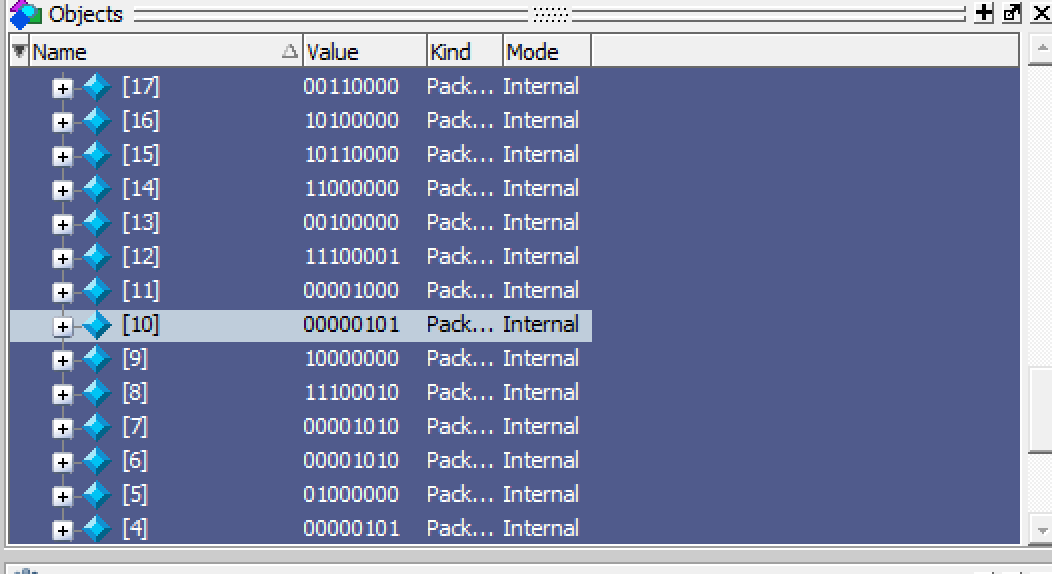
010110: SHFR A //A=<- A >> 1 = 4

010111: ROTR A //A<- 2 (Rotate A right)

011000: NOT A //A <- NOT A = -3

011001: HALT //Halt execution

Figure 3: Shows content of R[0] = 1 and testbench output



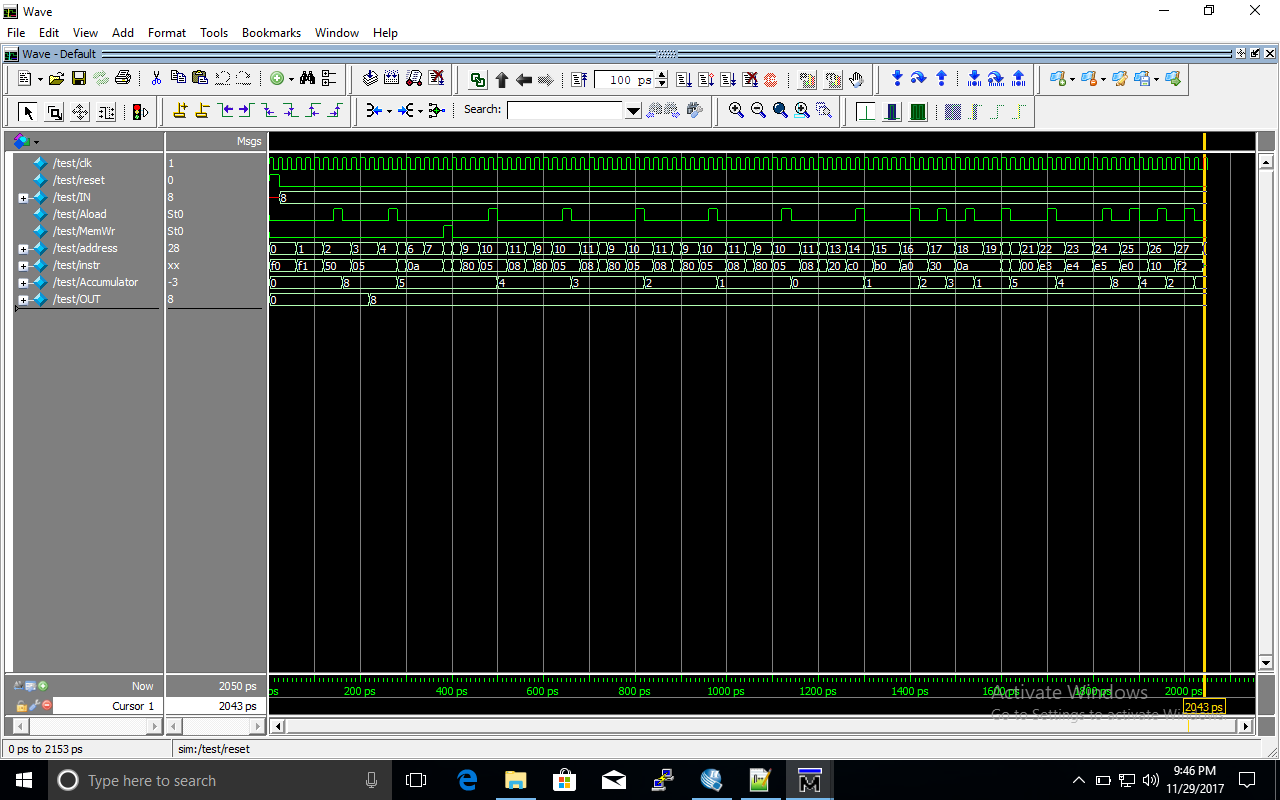
Figure 4: Shows content of Ram[10] = 5

Figure 5: ModelSim Waveform

In “memfile2.dat” the following instructions were executed:

000000: JZ 000011 //PC <- 000011 if A == 0

000010: NOT A //Shouldn’t execute

000011: LDI A, 5 //A <- 5

000101: JP 001000 //PC <- 001000 if A is positive

000111: NOT A //Shouldn’t execute

001000: STA //R[0]<- A = 5

001001: DEC A //A <- A-1

001010: JNZ 001001 //PC<-001001 if A != 0

001100: NOP //no operaton

001101: LDA //A<- R[0] = 5

001110: JMP //PC =<- 0100001

010000: NOT A //Shouldn’t execute

010001: HALT //Halt execution

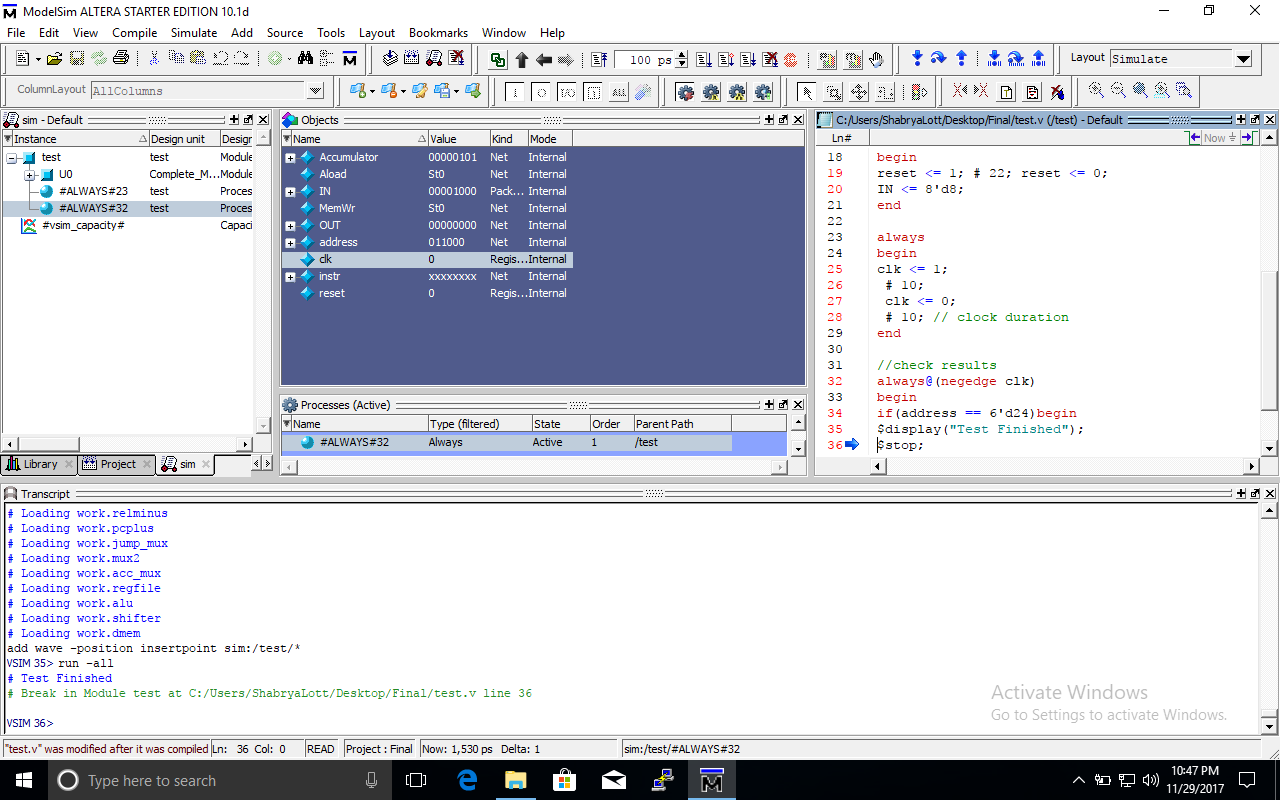


Figure 6: Testbench output

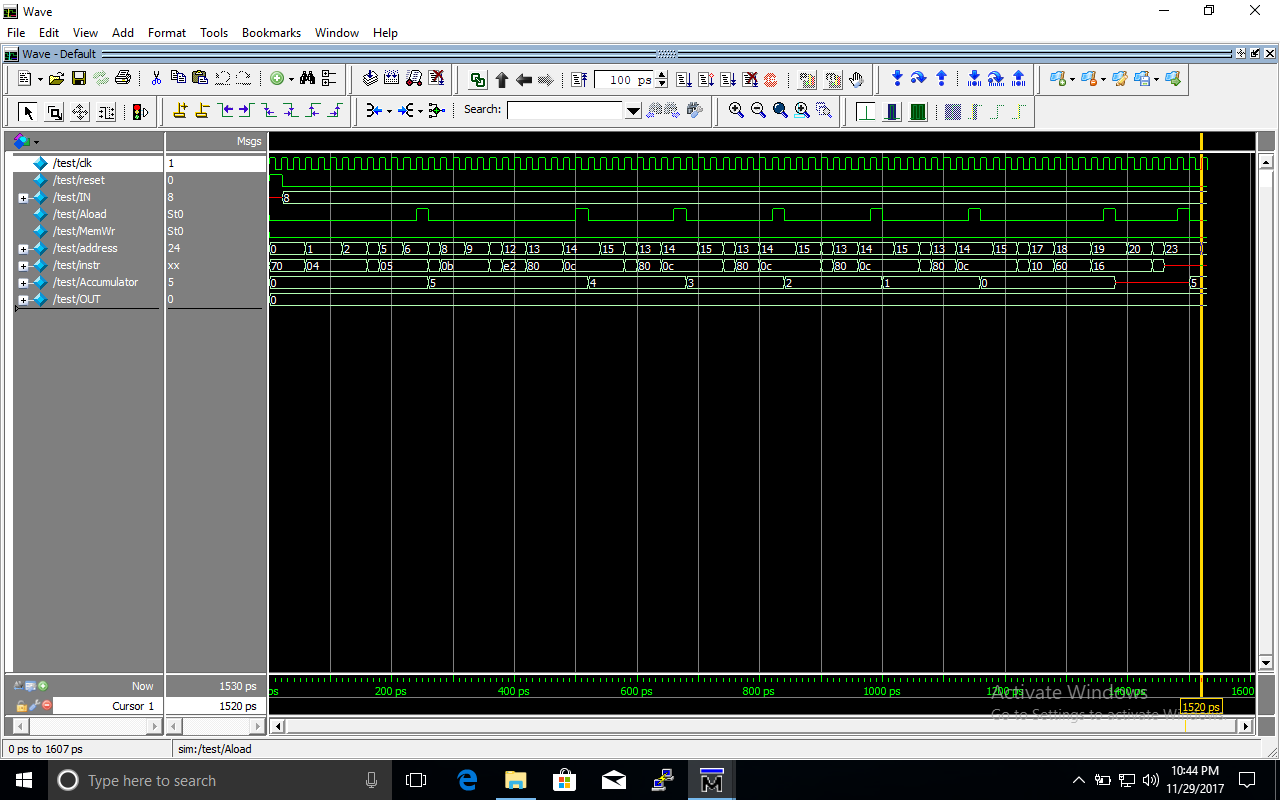


Figure 7: ModelSim Waveform