SINGLE CYCLE MIPS 32 CPU

DEEPANK CHINNAM

SCU ID: W1175796

CODE FOR PC:

```
module pc(clk, reset, PCin, PCout);
input clk, reset;
input [31:0] PCin;
output reg [31:0] PCout;
always @(negedge clk) begin
if(reset) PCout <= 0;
else PCout <= PCin + 1;
end
endmodule
```

CODE FOR INSTRUCTION MEMORY:

```
module IM (clk, Address ,inst);
input [31:0] Address;
input clk;
output reg[31:0] inst;
reg[31:0] IM[255:0];
initial begin
IM[0]<=32'h00221820;
IM[1]<=32'hAC010000;
IM[2]<=32'h8C240000;
IM[3]<=32'h10210001;          //beq
IM[4]<=32'h00001820;
IM[5]<=32'h00411822;
end
always @(posedge clk) begin
inst <= IM[Address];
```

```
end
endmodule
```

CODE FOR MUX (5 bit):

```
module mux5bit(MUXsel, input0, input1, MUXout_5);
input MUXsel;
input [4:0] input0,input1;
output reg [4:0] MUXout_5;
always @(MUXsel, input0, input1)
begin
case (MUXsel)
0: MUXout_5 <= input0 ;
1: MUXout_5 <= input1;
endcase
end
endmodule
```

CODE FOR REGISTER FILE:

```
module registerfile (Read1, Read2, WriteReg, WriteData, RegWrite, Data1, Data2, clk);
input [4:0] Read1, Read2, WriteReg;
input [31:0] WriteData;
input RegWrite, clk;
output [31:0] Data1, Data2;
reg [31:0] RF [31:0];
initial begin
RF[0] = 0;
RF[1] = 5;
```

```
RF[2] = 10;
end
assign Data1 = RF[Read1];
assign Data2 = RF[Read2];
always begin
@(posedge clk) if (RegWrite) RF[WriteReg] <= WriteData;
end
endmodule
```

CODE FOR SIGN EXTEND:

```
module signextend(dataInput,dataOutput);
input [15:0] dataInput ;
output [31:0] dataOutput ;
reg [31:0] dataOutput ;
always @ (dataInput)
begin
dataOutput [15:0] = dataInput [15:0];
dataOutput [31:16] = {16 { dataInput[15] } };
end
endmodule
```

CODE FOR MUX (32 bit):

```
module mux32bit (MUXselect, i0, i1, MUXout_32);
input MUXselect;
input [31:0] i0, i1;
output reg [31:0] MUXout_32;
always @(MUXselect, i0, i1) begin //reevaluate if these change
```

```
case (MUXselect)
0: MUXout_32 <= i0 ;
1: MUXout_32 <= i1;
endcase
end
endmodule
```

CODE FOR ALU:

```
module MIPSALU (ALUctl, A, B, ALUOut, Zero);
input [3 :0] ALUctl;
input [31:0] A, B;
output reg [31:0] ALUOut;
output Zero;
assign Zero = (ALUOut==0);     // zero is true if ALUOut is 0
always @(ALUctl, A, B)
case (ALUctl)
0: ALUOut <= A & B;           // AND operation
1: ALUOut <= A | B;           // OR operation
2: ALUOut <= A + B;           // ADD operation
6: ALUOut <= A - B;           // SUB operation
7: ALUOut <= A < B ? 1:0;      // if A<B then ALUOut=1 else ALUOut=0
12: ALUOut <= ~(A | B);       // NOR operation
default: ALUOut <=0;          // default case
endcase
endmodule
```

CODE FOR AND GATE:

```
module andgate (input0, input1, out);
input input0, input1;
output out;
assign out = input0 & input1;
endmodule
```

CODE FOR ALU CONTROL:

```
module Aluctrl (ALUOp, FuncCode, ALUCntrl);
input [1:0] ALUOp;
input [5:0] FuncCode;
output reg [3:0] ALUCntrl;
always @(ALUOp or FuncCode )
if(ALUOp==0) ALUCntrl<=2;
else if(ALUOp==1) ALUCntrl<=6;
else
        case(FuncCode)
        32: ALUCntrl <= 2; //add
        34: ALUCntrl <= 6; //subtract
        36: ALUCntrl <= 0; //and
        37: ALUCntrl <= 1; //or
        39: ALUCntrl <= 12; //nor
        42: ALUCntrl <= 7; //slt
        default: ALUCntrl <= 15; //not happen
        endcase
endmodule
```

CODE FOR DATA MEMORY:

```
module DM (Address, Write_Data, ReadData, MemRead , MemWrite, clk);
input MemRead, MemWrite, clk;
input [31:0] Write_Data, Address;
output reg [31:0] ReadData;
reg [31:0] DM [255:0];
always @(posedge clk) begin
 if(MemRead && !MemWrite) ReadData <= DM[Address];
 end
always @(negedge clk) begin
if(MemWrite && !MemRead) DM[Address] = Write_Data;
end
initial begin
DM [5] = 20;
end
endmodule
```

CODE FOR SHIFT LEFT:

```
module shiftleft(dataIn,dataOut);
input [31:0] dataIn;
output [31:0] dataOut;
reg [31:0] dataOut;
always @(dataIn)
begin
dataOut = dataIn << 2;
end
endmodule
```

CODE FOR ADDER (32 bit):

```
module adder32bit(PCadder1, shiftOut, ALUresult);

input [31:0] PCadder1;

input [31:0] shiftOut;

output reg [31:0] ALUresult;

always @ (PCadder1 or shiftOut) begin

ALUresult = PCadder1 + shiftOut;

end

endmodule
```

CODE FOR CONTROL UNIT:

```
module controlunit(op, reg_dst, alu_src, mem_to_reg, reg_write, mem_read, mem_write,
branch, ALUOp);

input [5:0] op;

output reg reg_dst, alu_src, mem_to_reg, reg_write, mem_read, mem_write, branch;

output reg [1:0] ALUOp;

always @(*)

begin

case (op)

6'b000000:

begin //R-type instruction

{reg_dst, alu_src, mem_to_reg, reg_write, mem_read, mem_write, branch} <=7'b1001000;

ALUOp<=2'b10;

end

6'b100011:

begin //lw instruction

{reg_dst, alu_src, mem_to_reg, reg_write, mem_read, mem_write, branch} <=7'b0111100;
```

```
ALUOp<=2'b00;
end
6'b101011: begin //sw instruction
{reg_dst, alu_src, mem_to_reg, reg_write, mem_read, mem_write, branch} <=7'b0100010;
ALUOp<=2'b00;
end
6'b000100: begin //beq instruction
{reg_dst, alu_src, mem_to_reg, reg_write, mem_read, mem_write, branch} <=7'b0000001;
ALUOp<=2'b01;
end
endcase
end
endmodule
```

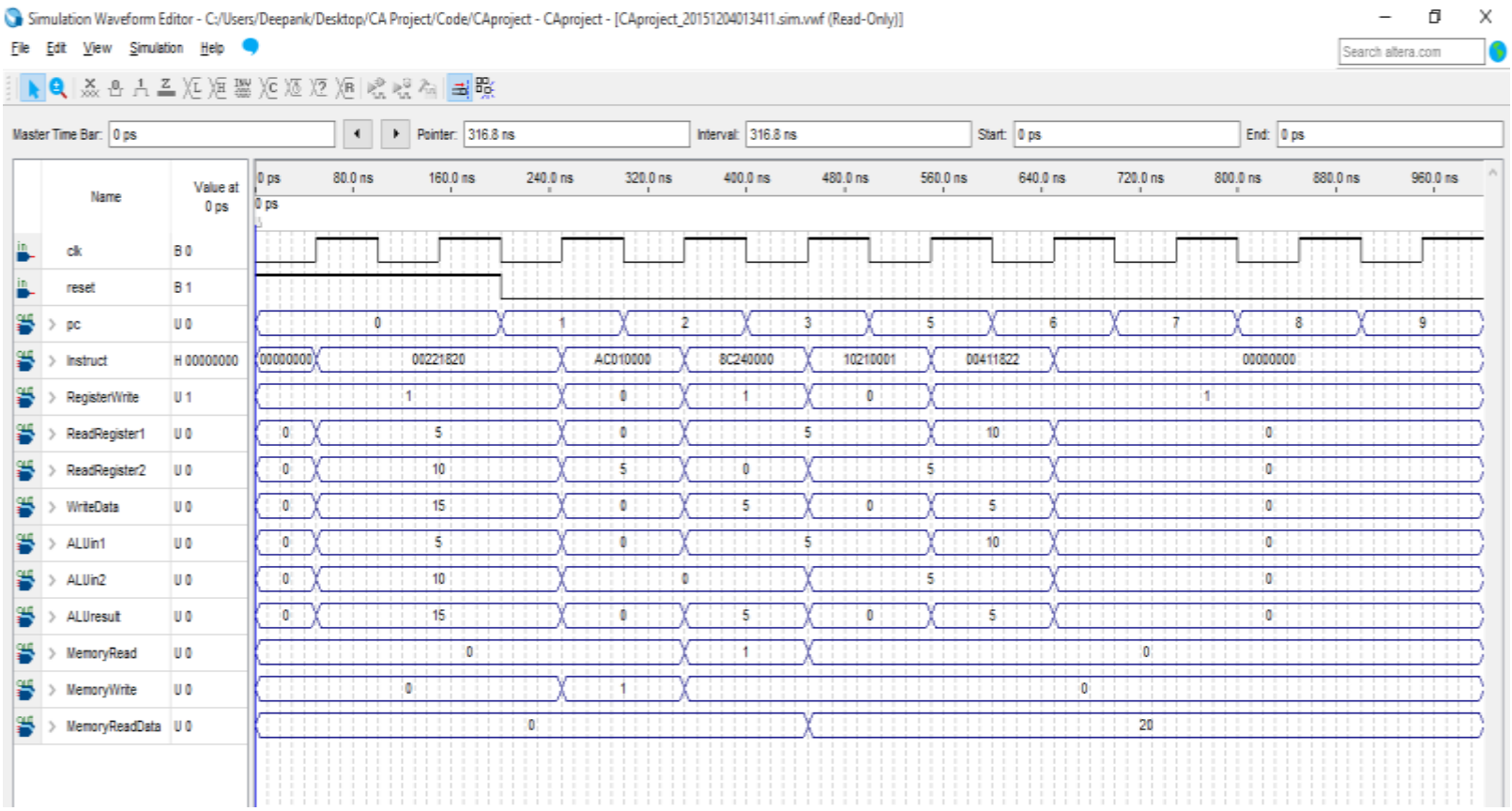
CODE FOR CONNECTIONS:

```
module connections (reset, clk, pc,
Instruct,ReadRegister1,ReadRegister2,WriteData,RegisterWrite,ALUin1,ALUin2,ALUresult,Mem
oryRead,
MemoryWrite, MemoryReadData);
input reset, clk;
output [31:0] pc,
Instruct,ReadRegister1,ReadRegister2,WriteData,RegisterWrite,ALUin1,ALUin2,ALUresult,Mem
oryRead,
MemoryWrite, MemoryReadData;
wire [31:0] PCout, instruction, Data1, Data2, signextend_out_1, ALU_i2, ALUresult_1,
Datamemory_read, mux2out,
pc_plus4, shiftOut, ALUresult_2, MUXout, MUXout_32, Read1, Read2, ALUOut;
wire RegDst, RegWrite, ALUSrc, MemWrite, MemRead, MemtoReg, Branch, zero, and_out_1,
ReadData ;
```

```
wire [4:0] WriteReg;

wire [3:0] ALU_ctrl;

wire [1:0] ALUctl;


pc pc_1 (.PCin(MUXout_32), .reset(reset), .clk(clk), .PCout(PCout));


IM IM_1 (.Address(PCout), .clk(clk),.inst(instruction));


mux5bit mux5bit_1 (.input0(instruction[20:16]), .input1(instruction[15:11]), .MUXsel(RegDst),
.MUXout_5(WriteReg));


registerfile registerfile_1 (.Read1(instruction[25:21]), .Read2(instruction[20:16]),
.WriteReg(WriteReg), .WriteData (mux2out), .Data1(Data1), .Data2(Data2),
.RegWrite(reg_write), .clk(clk));


signextend signextend_1 (.dataInput(instruction[15:0]), .dataOutput(signextend_out_1));


mux32bit mux32bit_1 (.i0(Data2), .i1(signextend_out_1), .MUXselect(ALUSrc),
.MUXout_32(ALU_i2));


Aluctrl Aluctrl_1 (.ALUOp(ALUctl), .FuncCode (instruction[5:0]), .ALUCntrl(ALU_ctrl));


MIPSALU MIPSALU_1(.A(Data1), .B(ALU_i2), .ALUctl(ALU_ctrl), .ALUOut(ALUresult_1),
.Zero(zero));


DM DM_1 (.Address(ALUresult_1), .Write_Data(Data2), .clk(clk), .MemWrite(MemWrite),
.MemRead(MemRead), .ReadData(Datamemory_read));
```

```
mux32bit mux32bit_2 (.i0(ALUresult_1), .i1(Datamemory_read), .MUXselect(mem_to_reg),
.MUXout_32(mux2out));


adder32bit adder32bit_1 (.PCadder1(PCout), .shiftOut(signextend_out_1),
.ALUresult(ALUresult_2));


andgate andgate_1 (.input0(Branch), .input1(zero), .out(and_out_1));


mux32bit mux32bit_3 (.i0(PCout), .i1(ALUresult_2), .MUXselect(and_out_1),
.MUXout_32(MUXout_32));


controlunit controlunit_1 (.op(instruction[31:26]), .mem_read(MemRead),
.mem_to_reg(MemtoReg), .branch(Branch), .reg_dst(RegDst), .reg_write(RegWrite),
.alu_src(ALUSrc), .ALUOp(ALUctl), .mem_write(MemWrite));


assign pc = PCout;

assign Instruct = instruction;

assign ReadRegister1 = Data1;

assign ReadRegister2 = Data2;

assign WriteData = mux2out;

assign RegisterWrite = RegWrite;

assign ALUin1 = Data1;

assign ALUin2 = ALU_i2;

assign ALUresult = ALUresult_1;

assign MemoryRead = MemRead;

assign MemoryWrite = MemWrite;

assign MemoryReadData = Datamemory_read;

endmodule
```

WAVEFORM:


Simulation Waveform Editor - C:/Users/Deepank/Desktop/CA Project/Code/CAproject - CAproject - [CAproject_20151204013411.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 316.8 ns
Interval: 316.8 ns
Start: 0 ps
End: 0 ps
Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
clk B 0
reset B 1
pc U 0
Instruct H 00000000
RegisterWrite U 1
ReadRegister1 U 0
ReadRegister2 U 0
WriteData U 0
ALUin1 U 0
ALUin2 U 0
ALUresult U 0
MemoryRead U 0
MemoryWrite U 0
MemoryReadData U 0
00000000
00221820
AC010000
8C240000
10210001
00411822
00000000