Name - Shubham Goyal
SCU ID - W1190952

1 .Verilog Codes for Different Module :

Program Counter

```
module PC(reset , clk, 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
```

AND Gate

```
module AndGate ( input1 , input2 , AndOut);
input input1 , input2;
output AndOut;
assign AndOut = input1 & input2;
endmodule
```

Sign Extension

```
module SignExtend(SignInputData , SignOutputData);
input [15:0] SignInputData;
output [31:0] SignOutputData;
reg [31:0] SignOutputData;
always @ ( SignInputData)
begin
SignOutputData[15:0] <= SignInputData[15:0];
SignOutputData[31:16] <= {16{SignInputData[15]}};
end
endmodule
```

MUX 32-Bit

```
module MUX ( MuxA, MuxB, MuxSel,MuxOut);
input [31:0] MuxA,MuxB;
input MuxSel;
output reg [31:0] MuxOut;
always @(MuxSel or MuxA or MuxB)
if( MuxSel) MuxOut <= MuxA;
else MuxOut <= MuxB;
endmodule
```

MUX 5-Bit

```
module MUX5 ( MuxA, MuxB, MuxSel,MuxOut);
input [4:0] MuxA,MuxB;
input MuxSel;
output reg [31:0] MuxOut;
always @(MuxSel or MuxA or MuxB)
if( MuxSel) MuxOut <= MuxA;
else MuxOut <= MuxB;
endmodule
```

ALU

```
module MIPSALU (ALUctl, ALU_A, ALU_B, ALUOut, Zero);
input [3 :0] ALUctl;
input [31:0] ALU_A, ALU_B;
output reg [31:0] ALUOut;
output Zero;
assign Zero = (ALUOut==0);
always @(ALUctl, ALU_A, ALU_B)
case (ALUctl)
0: ALUOut <= ALU_A & ALU_B;
1: ALUOut <= ALU_A | ALU_B;
2: ALUOut <= ALU_A + ALU_B;
6: ALUOut <= ALU_A - ALU_B;
7: ALUOut <= ALU_A < ALU_B ? 1:0;
12: ALUOut <= ~(ALU_A | ALU_B);
default: ALUOut <=0;
endcase
endmodule
```

Register file

```
module MIPSReg (Read1, Read2, WriteReg, WriteData, RegWrite, Data1, Data2, clock);
input [4:0] Read1, Read2, WriteReg;
input [31:0] WriteData;
input RegWrite, clock;
output [31:0] Data1, Data2;
reg [31:0] RF [31:0];
initial begin
RF[0] = 10;
RF[1] = 20;
RF[2] = 30;
RF[3] = 40;
RF[5] = 50;
end
assign Data1 = RF[Read1];
assign Data2 = RF[Read2];
always begin
@(negedge clock) if (RegWrite) RF[WriteReg] <= WriteData;
end
endmodule
```

Instruction Memory

```
module InstructMemory (clk, InstructAddress , InstructOutput);
input [31:0] InstructAddress;
input clk;
output reg[31:0] InstructOutput;
reg[31:0] IM[1023:0];
initial begin
IM[0] = 32'h00221820;        // add
IM[1] = 32'hAC010000;        // sw
IM[2] = 32'h8c240000;        // ld
IM[3] = 32'h10210001;        //beq
IM[4] = 32'h00001820;
IM[5] = 32'h00411822;        //sub
end
always @(posedge clk) begin
InstructOutput <= IM[InstructAddress];
end
endmodule
```

Full- Adder for PC

```
module FullAdder ( PcAdd4 , shiftOut, ALUOut);
input [31:0] PcAdd4;
input [31:0] shiftOut;
output reg [31:0] ALUOut;
always @ (PcAdd4 or ALUOut ) begin
ALUOut = PcAdd4 + shiftOut;
end
endmodule
```

ALU Control Unit

```
module ALUCtrl (ALUOp, FuncCode, ALUCtl);
input [1:0] ALUOp;
input [5:0] FuncCode;
output reg [3:0] ALUCtl;
always @(ALUOp or FuncCode )
if(ALUOp==0) ALUCtl<= 2;
else if(ALUOp==1) ALUCtl<=6;
else
        case(FuncCode)
        32: ALUCtl <= 2; //add
        34: ALUCtl <= 6; //subtract
        36: ALUCtl <= 0; //and
        37: ALUCtl <= 1; //or
        39: ALUCtl <= 12; //nor
        42: ALUCtl <= 7; //slt
        default: ALUCtl <= 15;
        endcase
endmodule
```

Main Control Unit

```
module ControlUnit(Opcode,RegWrite,MemRead,MemWrite,
branch,RegDst,ALUsrc,MemtoReg,ALUOp, jump);
input [5:0]Opcode;
output reg [1:0] ALUOp;
output reg RegWrite, MemWrite, MemRead, branch,RegDst,ALUsrc,MemtoReg, jump ;
always @(Opcode)
```

```
if(Opcode==0) begin
branch <=1'b0;
RegDst <=1'b1;
ALUsrc <=1'b0;
MemtoReg <=1'b0;
RegWrite <=1'b1;
MemRead <=1'b0;
MemWrite <=1'b0;
ALUOp <= 2'b10;
end
else if (Opcode==35) // LOAD
begin
branch <=1'b0;
RegDst <=1'b0;
ALUsrc <=1'b1;
MemtoReg <=1'b1;
RegWrite <=1'b1;
MemRead <=1'b1;
MemWrite <=1'b0;
ALUOp <=2'b00; // add
jump <=1'b0;
end
else if (Opcode==43) //STORE
begin
branch <=1'b0;
RegDst <=1'bz;
ALUsrc <=1'b1;
MemtoReg <=1'bz;
RegWrite <=1'b0;
MemRead <=1'b0;
MemWrite <=1'b1;
ALUOp <=2'b00; // add
jump <=0;
end
else if (Opcode==4) // BEQ
begin
branch <=1'b1;
RegDst <=1'bz;
ALUsrc <=1'b0;
MemtoReg <=1'bz;
RegWrite <=1'b0;
MemRead <=1'b0;
MemWrite <=1'b0;
```

```
ALUOp <=2'b01; //sub
jump <=1'b0;
end
else if (Opcode==8) //Addi
begin
branch <=1'b0;
RegDst <=1'b0;
ALUsrc <=1'b1;
MemtoReg <=1'b0;
RegWrite <=1'b1;
MemRead <=1'b0;
MemWrite <=1'b0;
ALUOp <=2'b0010; // add
jump <=1'b0;
end
else if (Opcode==2) //JUMP
begin
branch <=1'b0;
RegDst <=1'bz;
ALUsrc <=1'bz;
MemtoReg <=1'bz;
RegWrite <=1'b0;
MemRead <=1'b0;
MemWrite <=1'b0;
ALUOp <=2'bzz;
jump <=1'b1;
end
endmodule
```

Data Memory

```
module DataMemory (Address, WriteData, ReadData, MemRead , MemWrite, Clock);
input [31:0] WriteData, Address;
input MemRead, MemWrite, Clock;
output reg [31:0] ReadData;
reg [31:0] DM [255:0];
always @(posedge Clock) begin
 if(MemRead) ReadData = DM[Address];
 end
always @(negedge Clock) begin
if(MemWrite && !MemRead) DM[Address] = WriteData;
end
initial begin
```

```
DM[10] = 4;
DM[20] = 5;
DM[30] = 8;
DM[40] = 10;
DM[50] = 15;
end
endmodule
```

---

2. MIPS CPU With Connections

```
module MIPSCPU(reset, clk, INST, ALUOUT , MEMORYREAD, PCOUT, REGREAD1,
REGDATA1 , REGREAD2, REGDATA2, REGWRITE , MEMWRITE , MEMREAD,
WRITEADDREG, REGWRITEDATA, ALUIN, MEMWRITEDATA);
input reset , clk;
output [31:0] INST , ALUOUT , MEMORYREAD, PCOUT, REGDATA1 , REGREAD1,
REGDATA2, REGREAD2, REGWRITEDATA, ALUIN, MEMWRITEDATA;
output REGWRITE , MEMWRITE , MEMREAD;
output [4:0] WRITEADDREG;


wire [31:0] instruction , PcPlus1, RegData1 , RegData2 , SE_1_Out , ALUIn_2, ALUResult,
ReadDataMem , Mux3_Out , ALUOut_PC, Mux2_Out ;
wire [4:0] WriteRegAddr;
wire [3:0] ALUControl;
wire [1:0] ALUOpCode;
wire RegDst1 , RegWrite1 , ALUsrc1 , MemWrite1 , MemRead1 , MemtoReg1 , branch1 , Zero1 ,
AND_1_Out;

PC pc_1 (.PcIn(Mux3_Out) , .reset(reset) , .clk(clk) , .PcOut(PcPlus1));

InstructMemory IM_1 (.InstructAddress(PcPlus1) , .clk(clk), .InstructOutput(instruction));

MUX5 MUX5_1 (.MuxB(instruction[20:16]) , .MuxA(instruction[15:11]) , .MuxSel(RegDst1) ,
.MuxOut(WriteRegAddr));

MUX MUX_1 (.MuxB(RegData2) , .MuxA(SE_1_Out) , .MuxSel(ALUsrc1) , .MuxOut(ALUIn_2));

MUX MUX_2 (.MuxB(ALUResult) , .MuxA(ReadDataMem) , .MuxSel(MemtoReg1) ,
.MuxOut(Mux2_Out));

MUX MUX_3 (.MuxB(PcPlus1) , .MuxA(ALUOut_PC) , .MuxSel(AND_1_Out) ,
.MuxOut(Mux3_Out));
```

```
SignExtend SE_1(.SignInputData(instruction[15:0]) , .SignOutputData(SE_1_Out));

ALUCtrl ALUCtrl_1 (.ALUOp(ALUOpCode) , .FuncCode(instruction[5:0]) ,
.ALUCtl(ALUControl) );

FullAdder FullAdder_1 ( .PcAdd4(PcPlus1) , .shiftOut(SE_1_Out), .ALUOut(ALUOut_PC));

AndGate AND_1( .input1(Zero1) , .input2(branch1) , .AndOut(AND_1_Out));

ControlUnit CU_1 (.Opcode(instruction[31:26]), .RegWrite(RegWrite1), .MemRead(MemRead1),
.MemWrite(MemWrite1), .branch(branch1), .RegDst(RegDst1) , .ALUsrc(ALUsrc1),
.MemtoReg(MemtoReg1),  .jump(jump), .ALUOp(ALUOpCode));

MIPSReg Register_1 (.Read1(instruction[25:21]) , .Read2(instruction[20:16]) ,
.WriteReg(WriteRegAddr) , .WriteData(Mux2_Out), .Data1(RegData1) , .Data2(RegData2) ,
.RegWrite(RegWrite1) , .clock(clk) ) ;

MIPSALU ALU_1 (.ALUctl(ALUControl), .ALU_A(RegData1), .ALU_B(ALUIn_2),
.ALUOut(ALUResult), .Zero(Zero1));

DataMemory DM_1 (.Address(ALUResult), .WriteData(RegData2), .ReadData(ReadDataMem),
.MemRead(MemRead1) , .MemWrite(MemWrite1), .Clock(clk));

                assign INST = instruction;
                assign PCOUT = PcPlus1;
                assign MEMREAD = MemRead1;
                assign MEMWRITE = MemWrite1;
                assign REGREAD1 = instruction[25:21]; assign REGDATA1 = RegData1;
                assign REGREAD2 = instruction[20:16]; assign REGDATA2 = RegData2;
                assign WRITEADDREG = WriteRegAddr; assign REGWRITEDATA = Mux2_Out;
                assign REGWRITE = RegWrite1;
                assign ALUIN = ALUIn_2;
                assign ALUOUT = ALUResult;
                assign MEMORYREAD = ReadDataMem;
                assign MEMWRITEDATA = RegData2;

endmodule
```

---

3 .Waveform for MIPS CPU: