VE370 Introduction to Computer Organization

Project 2 Report

Author:	Group:	Student Number:
Chenyu Zhu	27	516370910131
Ruoxin Geng	27	516370910077
Xinhao Liao	27	516370910037

I. Objectives

In this part, we are going to design both a single-cycle CPU and a pipelined MIPS CPU in Verilog, which supports a main subset of MIPS instruction set including:

(1) The memory-reference instructions: load word (lw) and store word (sw).

(2) The arithmetic-logical instructions: add, addi, sub, and, andi, or and slt.

(3) The jump instructions: branch equal (beq), branch not equal (bne), and jump (j).

To successfully do this, we need:

(a) To understand all the related MIPS instructions.

(b) To understand the working principles of both a single-cycle computer and a pipelined computer.

(c) To design the architecture for both the single-cycle and the pipelined CPUs.

(d) To implement all the components we need in Verilog and then build up the CPUs.

(e) To deal with possible data hazards and structure hazards with some control signals.

II. Top Level Block Diagram

(i). Single Cycle

We use the following figure (Figure 1) for our design of single cycle.

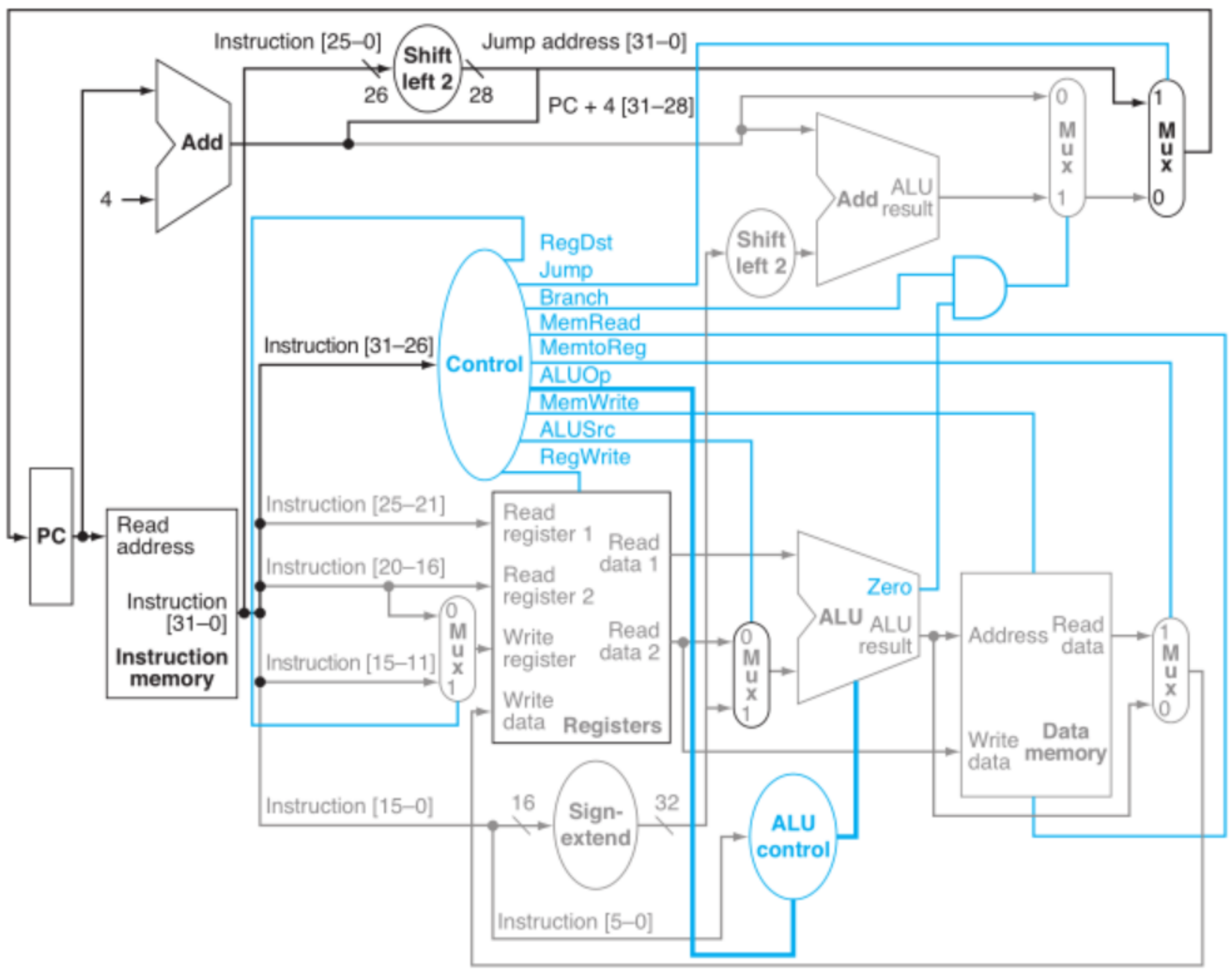

Figure 1: Single Cycle Diagram.

(ii). Pipeline

We use the following figure(Figure 2) for our design of pipeline. The design of pipeline composes of four pipelines: IF/ID , ID/EX , EX/MEM and MEM/WB. And the control signal stored in each pipeline is shown in Figure 3.

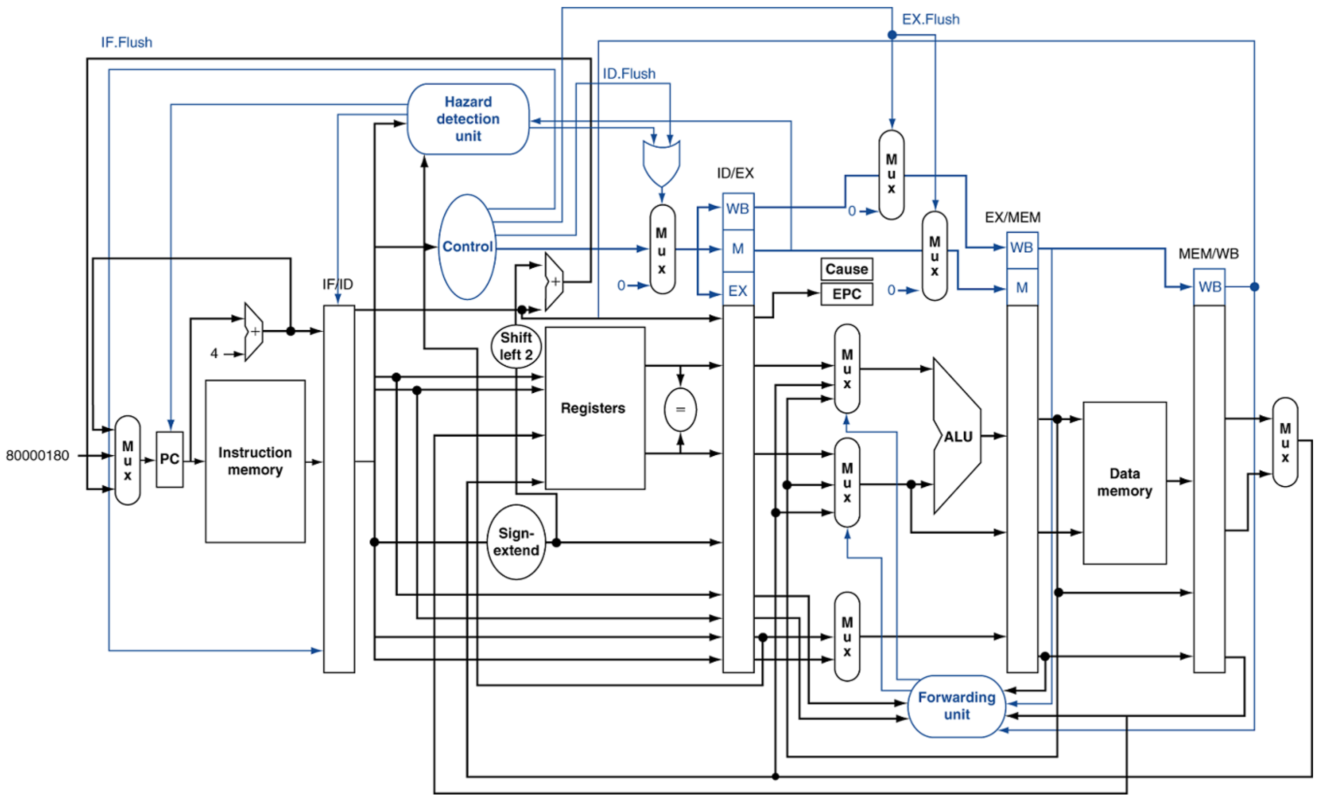

Figure 2: Pipeline Diagram

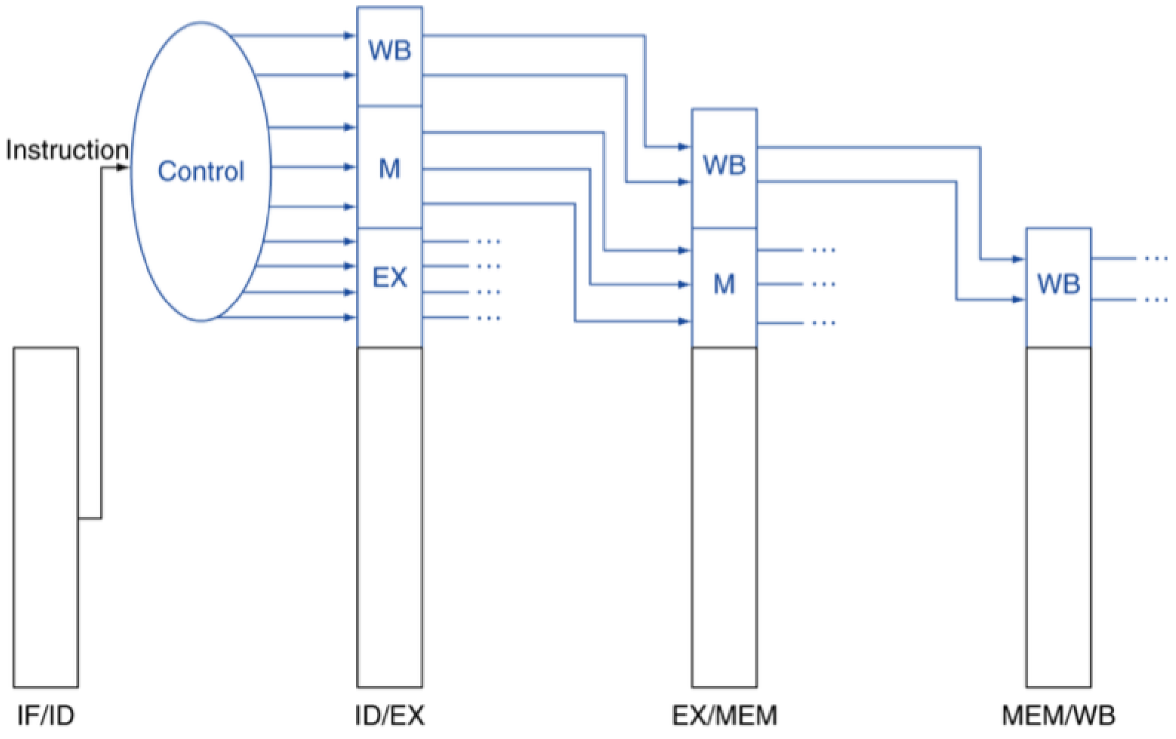

Figure 3: Control Signals in the Pipelined Architecture

III. Design of Components

Since almost all the components used in the pipelined architecture can be used in the single-cycle architecture, we only talk about the cases in the pipelined one. And the codes for the single-cycled CPU are attached in the appendix.

(i). IF stage

The modules we design in this stage are PC and the instruction memory. The PC stores the address of the instructions and the instruction memory stores all the machine codes for the instructions. The Verilog codes for the PC is

```
module PC(
input clk,
input PCwrite,
input [31:0] PCin,
output reg [31:0] PCout);
always@(posedge clk)begin
if(PCwrite)
PCout <= PCin;
end
endmodule
```

And the Verilog codes for the instruction memory is

```
module IM(Instruction,PC);
output [31:0]Instruction;
input [31:0] PC;
wire [7:0] index;
reg [31:0] memory[0:63];
integer i;
initial begin
memory[0] = 32'b00100000000010000000000000100000; //addi $t0, $zero, 0x20
memory[1] = 32'b00100000000010010000000000110111; //addi $t1, $zero, 0x37
memory[2] = 32'b00000001000010011000000000100100; //and $s0, $t0, $t1
memory[3] = 32'b00000001000010011000000000100101; //or $s0, $t0, $t1
memory[4] = 32'b10101100000100000000000000000100; //sw $s0, 4($zero)
memory[5] = 32'b10101100000010000000000000001000; //sw $t0, 8($zero)
memory[6] = 32'b00000001000010011000100000100000; //add $s1, $t0, $t1
memory[7] = 32'b00000001000010011001000000100010; //sub $s2, $t0, $t1
memory[8] = 32'b00010010001100100000000000001001; //beq $s1, $s2, error0
memory[9] = 32'b10001100000100010000000000000100; //lw $s1, 4($zero)
memory[10]= 32'b00110010001100100000000001001000; //andi $s2, $s1, 0x48
memory[11] =32'b00010010001100100000000000001001; //beq $s1, $s2, error1
memory[12] =32'b10001100000100110000000000001000; //lw $s3, 8($zero)
memory[13] =32'b00010010000100110000000000001010; //beq $s0, $s3, error2
memory[14] =32'b00000010010100011010000000101010; //slt $s4, $s2, $s1 (Last)
memory[15] =32'b00010010100000000000000000001111; //beq $s4, $0, EXIT
memory[16] =32'b00000010001000001001000000100000; //add $s2, $s1, $0
memory[17] =32'b00001000000000000000000000001110; //j Last
memory[18] =32'b00100000000010000000000000000000; //addi $t0, $0, 0(error0)
memory[19] =32'b00100000000010010000000000000000; //addi $t1, $0, 0

memory[20] =32'b00001000000000000000000000011111; //j EXIT
```

```
memory[21] =32'b00100000000010000000000000000001; //addi $t0, $0, 1(error1)
memory[22] =32'b00100000000010010000000000000001; //addi $t1, $0, 1
memory[23] =32'b00001000000000000000000000011111; //j EXIT
memory[24] =32'b00100000000010000000000000000010; //addi $t0, $0, 2(error2)
memory[25] =32'b00100000000010010000000000000010; //addi $t1, $0, 2
memory[26] =32'b00001000000000000000000000011111; //j EXIT
memory[27] =32'b00100000000010000000000000000011; //addi $t0, $0, 3(error3)
memory[28] =32'b00100000000010010000000000000011; //addi $t1, $0, 3
memory[29] =32'b00001000000000000000000000011111; //j EXIT
for(i=30;i<64;i=i+1) memory[i] <= 32'd0;
end
assign index = PC[9:2];
assign Instruction = memory[index];
//$display("the instruction now is %d", Instruction);
endmodule
```

(ii). ID stage

There are four modules in this stage: hazard-detection unit, control, registers and sign-extension. The hazard-detection unit will detect possible data and control hazard in three circumstances: an R-type instruction followed by a branch instruction, a load-word instruction followed by some instruction needing its result, and only one single instruction is between a load word instruction and a branch instruction, where the load word instruction occurs first. The control module generates all the control signals needed in the current instruction and IF.Flush to deal with possible structure hazard which happens when branch instructions take place. The registers stores the numbers in all registers. Sign-extension sign-extends the immediate numbers.

The Verilog codes for the hazard-detection unit is

```
module Hazard(
input [31:0] Ins,
input ID_EX_MemRead, MemRead,
input [4:0] ID_EX_rt,ID_EX_rd, dst_MEM,
input [1:0] ALUOp_EX,
output reg PC_write, IF_ID_write, nop);
wire [5:0] opcode;
wire [4:0] rs, rt;
assign opcode = Ins[31:26];
assign rs = Ins[25:21];
assign rt = Ins[20:16];
initial begin
PC_write <= 1;
IF_ID_write <= 1;
nop = 0;
end
always @(*)begin
if(ID_EX_MemRead && (rs == ID_EX_rt || rt == ID_EX_rt))begin //lw
PC_write <= 0;
IF_ID_write <= 0;
nop <= 1;

end
```

```
else if ((opcode == 6'b000100 || opcode == 6'b000101) &&  ALUOp_EX == 2'b10  && (rs == ID_EX_rd
|| rt == ID_EX_rd))
begin
PC_write <= 0;
IF_ID_write <= 0;
nop <= 1;
end
else if ((opcode == 6'b000100 || opcode == 6'b000101) && MemRead && (rs == dst_MEM || rt ==
dst_MEM))
begin
PC_write <= 0;
IF_ID_write <= 0;
nop <= 1;
end
else
begin
PC_write <= 1;
IF_ID_write <= 1;
nop <= 0;
end
end

endmodule
```

The Verilog codes for the control unit is

```
module Control(
input equal, IF_ID_write,
input [1:0] ALUOp_EX,
input [5:0]opcode,
output reg Jump, Branch, Bne, IF_flush,
output [7:0] CtrlSig);
reg RegDst, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite;
reg [1:0] ALUOp;
assign CtrlSig = {RegWrite,MemtoReg,MemRead, MemWrite,RegDst, ALUOp, ALUSrc};
initial begin
IF_flush <= 0;
RegDst <= 0;
Jump <= 0;
Branch <= 0;
Bne <= 0;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 0;
RegWrite <= 0;
ALUOp <= 2'b00;
end
always @(*) begin
case(opcode)
6'b000000://R

begin
```

```
IF_flush <= 0;
RegDst <= 1;
Jump <= 0;
Branch <= 0;
Bne <= 0;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 0;
RegWrite <= 1;
ALUOp <= 2'b10;
end
6'b001000://addi
begin
IF_flush <= 0;
RegDst <= 0;
Jump <= 0;
Branch <= 0;
Bne <= 0;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 1;
RegWrite <= 1;
ALUOp <= 2'b00;
end
6'b001100://andi
begin
IF_flush <= 0;
RegDst <= 0;
Jump <= 0;
Branch <= 0;
Bne <= 0;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 1;
RegWrite <= 1;
ALUOp <= 2'b11;
end
6'b000100://beq
begin
if(IF_ID_write) IF_flush<=equal ? 1:0;
else IF_flush<=0;
RegDst <= 0;
Jump <= 0;
Branch <= 1;
Bne <= 0;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 0;

RegWrite <= 0;
```

```
ALUOp <= 2'b01;
end
6'b000101://bne
begin
if(IF_ID_write) IF_flush<=equal ? 0:1;
else IF_flush<=0;
RegDst <= 0;
Jump <= 0;
Branch <= 0;
Bne <= 1;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 0;
RegWrite <= 0;
ALUOp <= 2'b01;
end
6'b000010://j
begin
IF_flush <= 1;
RegDst <= 0;
Jump <= 1;
Branch <= 0;
Bne <= 0;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 0;
RegWrite <= 0;
ALUOp <= 2'b00;
end
6'b100011://lw
begin
IF_flush <= 0;
RegDst <= 0;
Jump <= 0;
Branch <= 0;
Bne <= 0;
MemRead <= 1;
MemtoReg <= 1;
MemWrite <= 0;
ALUSrc <= 1;
RegWrite <= 1;
ALUOp <= 2'b00;
end
6'b101011://sw
begin
IF_flush <= 0;
RegDst <= 0;
Jump <= 0;
Branch <= 0;
Bne <= 0;

MemRead <= 0;
```

```
MemtoReg <= 0;
MemWrite <= 1;
ALUSrc <= 1;
RegWrite <= 0;
ALUOp <= 2'b00;
end
endcase
end
endmodule
```

The Verilog codes for the register files is

```
module Registers(clk,clk2, rs, rt,RegWrite, Data1, Data2, writeData,
writeReg,observe,Data_ob,reset);
input clk,clk2, RegWrite, reset;
input [4:0] rs;
input [4:0] rt;
input [4:0] observe;
output [31:0] Data1, Data2, Data_ob;
input wire [4:0] writeReg;
input wire [31:0] writeData;
reg [31:0] register[0:31];
integer i;
initial begin
    for (i = 0; i < 32; i = i+1) register[i] <= 0; //Initialize the registers
end

assign Data_ob = register[observe];

always @(posedge clk2)
begin
if (reset)
begin
    for (i = 0; i < 32; i = i+1) register[i] <= 0; //Initialize the registers
end
else if(RegWrite == 1 && writeReg != 0) register[writeReg] <= writeData;
end
assign Data1 = register[rs];
assign Data2 = register[rt];

endmodule
```

And the Verilog codes for the sign-extend unit is given by

```
module signExtend(in, out);
input [15:0] in;
output [31:0] out;
assign out = {{16{in[15]}}, in[15:0]};

endmodule
```

(iii). EX stage

The modules in this stage are given by: ALU_Control, ALU and forwarding unit. ALU_control generates the ALUop for the ALU arithmetic. ALU will calculate the results based on the two inputs and ALUop generated by ALU_control and forwarding unit will generate forwarding control signals to forward the ALU result to proper places when possible data hazard will occur. Below are the codes for them.

The Verilog codes for the ALU_Control unit is

```
module ALUControl(
input [5:0] funct,
input [1:0] ALUOp,
output reg [3:0] ALUControl);
always @(funct or ALUOp)
begin
    case(ALUOp)
    2'b00: ALUControl <= 4'b0010;
    2'b01: ALUControl <= 4'b0110;
    2'b11: ALUControl <= 4'b0000;
    2'b10:begin
        case(funct)
        6'b100000:ALUControl <= 4'b0010;
        6'b100010:ALUControl <= 4'b0110;
        6'b100100:ALUControl <= 4'b0000;
        6'b100101:ALUControl <= 4'b0001;
        6'b101010:ALUControl <= 4'b0111;
        endcase
    end
    endcase
end
endmodule
```

The Verilog codes for the ALU is given by

```
module ALU(
input [31:0] in1,in2,
input [3:0] Control,
output [31:0] Result);
assign Result = (Control == 4'b0010) ? (in1 + in2):
                (Control == 4'b0110) ? (in1 - in2):
                (Control == 4'b0000) ? (in1 & in2):
                (Control == 4'b0001) ? (in1 | in2):
                ((Control == 4'b0111) && (in1 < in2)) ? 1:0;

endmodule
```

The Verilog codes for the forwarding unit is given by

```
module Forwarding(

input MEM_RegWrite,WB_RegWrite,
input [4:0] MEM_rd,WB_rd,EX_rs,EX_rt,
output reg [1:0] ForwardA, ForwardB);
initial begin
ForwardA <= 2'b00;
ForwardB <= 2'b00;
end
always @(*)
begin
if(MEM_RegWrite && (MEM_rd !=0) && (MEM_rd == EX_rs)) ForwardA <= 2'b10;
else if (WB_RegWrite && (WB_rd != 0) && (WB_rd == EX_rs)) ForwardA <= 2'b01;
else ForwardA <= 2'b00;

if(MEM_RegWrite && (MEM_rd !=0) && (MEM_rd == EX_rt)) ForwardB <= 2'b10;
else if (WB_RegWrite && (WB_rd != 0) && (WB_rd == EX_rt)) ForwardB <= 2'b01;
else ForwardB <= 2'b00;

end

endmodule
```

(iv). MEM stage

In this stage, there is one module: data memory. It stores all the data and the storage of the data memory is much larger than that of registers. Here are the Verilog codes for it

```
module dataMem(clock, address, write, read, writeData,  readData, reset);
input wire read, write, reset, clock;
input wire[31:0] writeData, address;
output reg [31:0] readData;
reg [31:0] memory [0:63];
integer i;

initial begin
for(i=0;i<64;i=i+1) memory[i] <= 32'd0;
end
always @(negedge clock)
if(read)
readData <= memory[address];
else readData <= 0;

always @(posedge clock)
begin
if (reset)
begin
for(i=0;i<64;i=i+1) memory[i] <= 32'd0;
end
else if (write)
        memory[address] <= writeData;
end
endmodule
```

(v). Pipeline Registers

There are four pipeline registers, namely IF/ID, ID/EX, EX/MEM, MEM/WB, respectively.

The codes for the IF/ID pipeline register is

```
module IF_ID(
input clk,reset,
input IF_ID_write,IF_flush,
input [31:0] PCin, InsIn,
output reg [31:0] PCout, InsOut);

always@(posedge clk)
begin
if(reset) InsOut <= 0;
else if(IF_flush) InsOut <= 0;
else if(IF_ID_write)
begin
PCout <= PCin;
InsOut <= InsIn;
end
end
endmodule
```

The Verilog codes for the ID/EX pipeline register is

```
module ID_EX(
input clk,reset,
input [7:0] CtrlSig,
input [31:0] data1in, data2in, extendedin,
input [4:0] rs_ID, rt_ID, rd_ID,
output reg [1:0] WBSig,
output reg [1:0] MSig,
output reg RegDst,
output reg [1:0] ALUOp,
output reg ALUSrc,
output reg [31:0] data1out,data2out,extendedout,
output reg [4:0]  rs_EX, rt_EX, rd_EX);

always@(posedge clk)
if (reset){WBSig,MSig,RegDst,ALUOp,ALUSrc,data1out,data2out,extendedout,rs_EX,rt_EX,rd_EX} <= 0;
else
begin
WBSig <= CtrlSig[7:6];
MSig <= CtrlSig[5:4];
RegDst <= CtrlSig[3];
ALUOp <= CtrlSig[2:1];
ALUSrc <= CtrlSig[0];
data1out <= data1in;
data2out <= data2in;

extendedout <= extendedin;
```

```
rs_EX <= rs_ID;
rt_EX <= rt_ID;
rd_EX <= rd_ID;
end

endmodule
```

The Verilog codes for the EX/MEM pipeline register is

```
module EX_MEM(
input clk, reset,
input [1:0] WBSig,
input [1:0] MSig,
input [31:0] ALURes,WriteDataIn,
input [4:0] dstIn,
output reg [1:0] CtrlLeft,
output reg MemRead, MemWrite,
output reg [31:0] Address, WriteDataOut,
output reg [4:0] dstOut);


always@(posedge clk)
begin
if(reset)
{CtrlLeft,MemRead, MemWrite,Address, WriteDataOut,dstOut} <= 0;
else begin
CtrlLeft <= WBSig;
MemRead <= MSig[1];
MemWrite <= MSig[0];
Address <= ALURes;
WriteDataOut <= WriteDataIn;
dstOut <= dstIn;
end
end
endmodule
```

The Verilog codes for the MEM/WB pipeline register is

```
module MEM_WB(
input clk,reset,
input [1:0] CtrlSig,
input [31:0] ReadDataIn, ALUResIn,
input [4:0] dstIn,
output reg [4:0] dstOut,
output reg RegWrite,MemtoReg,
output reg [31:0] ReadDataOut,ALUResOut);

always@(posedge clk)
if(reset)
{dstOut,RegWrite,MemtoReg,ReadDataOut,ALUResOut} <= 0;

else begin
```

```
RegWrite <= CtrlSig[1];
MemtoReg <= CtrlSig[0];
ReadDataOut <= ReadDataIn;
ALUResOut <= ALUResIn;
dstOut <= dstIn;
end
```

(vi). The main function

The main function puts all the modules we talked about together and it makes our project work like a real CPU supporting a main subset of MIPS instruction set. The whole architecture is generally based on the figures in the textbook. Unless certain kinds of data or control hazard that enforces a nope instruction is detected (namely, IF_Flush==1), the PC module always refreshes itself every clock cycle. And the necessary control signals will be generated in the ID stage depending on the current instruction and whether hazards are detected. The forwarding unit will also generate some signals selecting the comparators in the ID stage (if branch instruction is in ID stage and some data hazard is detected) and also some mux signals selecting the input of ALU (when some data hazard is detected).

The Verilog codes for our main function is

```
module main(
input clk, clk2, reset,
input [4:0] observe,
output [31:0] data_ob
);

wire [31:0] newPC, PC, PC_IF, Ins_IF, PC_ID, Ins_ID, Data1_ID, Data2_ID, WriteDataReg;
wire [31:0] ExtendedIn, ExtendedOut, Data1_EX, Data2_EX;
wire [31:0] ALURes, Address, WriteDataMem, ReadDataMem, ALUinput2;
wire [7:0] CtrlSig, CtrlSig_ID;
wire [1:0] WBSig_EX, MSig_EX, WBSig_MEM;
wire [4:0] rs_EX,rt_EX, rd_EX,dst_EX,dst_MEM, dst;
wire [3:0] ALUCtrlSig;
wire [1:0] ALUOp, ForwardA, ForwardB, ForwardCmp1, ForwardCmp2;
wire RegDst,ALUSrc,RegWrite,MemtoReg,Jump,Branch,Bne,MemRead,MemWrite;

wire [1:0] move;
wire PCwrite,IF_ID_write, IF_flush, nop, equal, MemReadEX, RegWriteMEM, RegWriteEX;
wire [31:0] JumpAddr, A, B, ReadDataWB, ALUresWB,shifted;
wire [31:0] compare1, compare2;


assign MemReadEX = MSig_EX[1];
assign RegWriteMEM = WBSig_MEM[1];
assign RegWriteEX = WBSig_EX[1];
assign PC_IF = PC + 4;

Forwarding fwd_cmp(RegWriteEX,RegWriteMEM,dst_EX, dst_MEM, Ins_ID[25:21], Ins_ID[20:16],
ForwardCmp1, ForwardCmp2);
assign compare1 = (ForwardCmp1 == 2'b00) ? Data1_ID :

                  (ForwardCmp1 == 2'b01) ? Address :
                    (ForwardCmp1 == 2'b10) ? ALURes : 0;
assign compare2 = (ForwardCmp2 == 2'b00) ? Data2_ID :
                  (ForwardCmp2 == 2'b01) ? Address :
                  (ForwardCmp2 == 2'b10) ? ALURes : 0;

assign equal = (compare1 == compare2) ? 1 : 0;

assign move = (Jump || (Branch && equal) || (Bne && !equal)) ? 2'b10 :
              reset ? 2'b01 : 2'b00;
assign newPC = (move == 2'b00) ? (PC+4):
               (move == 2'b01) ? 0:
               (move == 2'b10) ? JumpAddr : 0;
assign shifted = {ExtendedIn[29:0],{2'b00}};
assign JumpAddr = Jump ? {PC_ID[31:28],Ins_ID[25:0],{1'b0,1'b0}}:
                  (Branch && equal) || (Bne && !equal) ? (shifted + PC_ID) : 0;

assign CtrlSig_ID = nop ? 0 : CtrlSig;
assign A = (ForwardA == 2'b00) ? Data1_EX:
           (ForwardA == 2'b01) ? WriteDataReg:
           (ForwardA == 2'b10) ? Address : 0;
assign B = (ForwardB == 2'b00) ? Data2_EX:
           (ForwardB == 2'b01) ? WriteDataReg:
           (ForwardB == 2'b10) ? Address : 0;

assign ALUinput2 = (ALUSrc) ? ExtendedOut : B;

//assign ALUSource = ALUSrc ? ExtendedOut : Data2_EX;
assign dst_EX = RegDst ? rd_EX : rt_EX;
assign WriteDataReg = MemtoReg ? ReadDataWB : ALUresWB;

PC pc(clk,PCwrite,newPC, PC);
IM InsMem(Ins_IF, PC);
IF_ID if_id(clk,reset,IF_ID_write,IF_flush,PC_IF, Ins_IF, PC_ID, Ins_ID);
Hazard hzd(Ins_ID,MemReadEX,MemRead,rt_EX, rd_EX, dst_MEM,ALUOp,PCwrite,IF_ID_write,nop);
Control ctrl(equal,IF_ID_write,ALUOp,Ins_ID[31:26],Jump, Branch, Bne, IF_flush,CtrlSig);
Registers regs(clk,clk2, Ins_ID[25:21],
Ins_ID[20:16],RegWrite,Data1_ID,Data2_ID,WriteDataReg,dst,observe, data_ob,reset);
signExtend sE(Ins_ID[15:0], ExtendedIn);
ID_EX
id_ex(clk,reset,CtrlSig_ID,Data1_ID,Data2_ID,ExtendedIn,Ins_ID[25:21],Ins_ID[20:16],Ins_ID[15:11
],
WBSig_EX,MSig_EX,RegDst,ALUOp,ALUSrc,Data1_EX, Data2_EX, ExtendedOut,rs_EX,rt_EX, rd_EX);
ALUControl aluctrl(ExtendedOut[5:0],ALUOp,ALUCtrlSig);
ALU alu(A,ALUinput2,ALUCtrlSig,ALURes);
Forwarding fwd(RegWriteMEM,RegWrite,dst_MEM,dst,rs_EX,rt_EX,ForwardA,ForwardB);
EX_MEM
ex_mem(clk,reset,WBSig_EX,MSig_EX,ALURes,B,dst_EX,WBSig_MEM,MemRead,MemWrite,Address,WriteDataMe
m,dst_MEM);
dataMem dm(clk2, Address, MemWrite, MemRead,WriteDataMem,ReadDataMem,reset);
MEM_WB mem_wb(clk,reset, WBSig_MEM, ReadDataMem, Address,dst_MEM,dst,RegWrite,MemtoReg,
ReadDataWB, ALUresWB);


endmodule
```

(vii). The simulation module

To simulate our codes, we need to add a module called "sim". It is very similar to the main function. Here are the Verilog codes for it.

```
module sim();
reg reset;
reg clk, clk2;
reg [4:0] observe;
wire [31:0] data_ob;
wire [31:0] newPC, PC, PC_IF, Ins_IF, PC_ID, Ins_ID, Data1_ID, Data2_ID, WriteDataReg;
wire [31:0] ExtendedIn, ExtendedOut, Data1_EX, Data2_EX;
wire [31:0] ALURes, Address, WriteDataMem, ReadDataMem, ALUinput2;
wire [7:0] CtrlSig, CtrlSig_ID;
wire [1:0] WBSig_EX, MSig_EX, WBSig_MEM;
wire [4:0] rs_EX,rt_EX, rd_EX,dst_EX,dst_MEM, dst;
wire [3:0] ALUCtrlSig;
wire [1:0] ALUOp, ForwardA, ForwardB, ForwardCmp1, ForwardCmp2;
wire RegDst,ALUSrc,RegWrite,MemtoReg,Jump,Branch,Bne,MemRead,MemWrite;

wire [1:0] move;
wire PCwrite,IF_ID_write, IF_flush, nop, equal, MemReadEX, RegWriteMEM, RegWriteEX;
wire [31:0] JumpAddr, A, B, ReadDataWB, ALUresWB;
wire [31:0] compare1, compare2;

assign MemReadEX = MSig_EX[1];
assign RegWriteMEM = WBSig_MEM[1];
assign RegWriteEX = WBSig_EX[1];
assign PC_IF = PC + 4;

Forwarding fwd_cmp(RegWriteEX,RegWriteMEM,dst_EX, dst_MEM, Ins_ID[25:21], Ins_ID[20:16],
ForwardCmp1, ForwardCmp2);
assign compare1 = (ForwardCmp1 == 2'b00) ? Data1_ID :
                  (ForwardCmp1 == 2'b01) ? Address :
                  (ForwardCmp1 == 2'b10) ? ALURes : 0;
assign compare2 = (ForwardCmp2 == 2'b00) ? Data2_ID :
                  (ForwardCmp2 == 2'b01) ? Address :
                  (ForwardCmp2 == 2'b10) ? ALURes : 0;

assign equal = (compare1 == compare2) ? 1 : 0;
assign move = (Jump || (Branch && equal) || (Bne && !equal)) ? 2'b10 :
              reset ? 2'b01 : 2'b00;
assign newPC = (move == 2'b00) ? (PC+4):
               (move == 2'b01) ? 0:
               (move == 2'b10) ? JumpAddr : 0;
assign JumpAddr = Jump ? {PC_ID[31:28],Ins_ID[25:0],{1'b0,1'b0}}:
                  (Branch && equal) || (Bne && !equal) ? (ExtendedIn * 4 + PC_ID) : 0;

assign CtrlSig_ID = nop ? 0 : CtrlSig;
assign A = (ForwardA == 2'b00) ? Data1_EX:
           (ForwardA == 2'b01) ? WriteDataReg:
              (ForwardA == 2'b10) ? Address : 0;
assign B = (ForwardB == 2'b00) ? Data2_EX:
              (ForwardB == 2'b01) ? WriteDataReg:
              (ForwardB == 2'b10) ? Address : 0;

assign ALUinput2 = (ALUSrc) ? ExtendedOut : B;

//assign ALUSource = ALUSrc ? ExtendedOut : Data2_EX;
assign dst_EX = RegDst ? rd_EX : rt_EX;
assign WriteDataReg = MemtoReg ? ReadDataWB : ALUresWB;

PC pc(clk,PCwrite,newPC, PC);
IM InsMem(Ins_IF, PC);
IF_ID if_id(clk,reset,IF_ID_write,IF_flush,PC_IF, Ins_IF, PC_ID, Ins_ID);
Hazard hzd(Ins_ID,MemReadEX,MemRead,rt_EX, rd_EX, dst_MEM,ALUOp,PCwrite,IF_ID_write,nop);
Control ctrl(equal,IF_ID_write,ALUOp,Ins_ID[31:26],Jump, Branch, Bne, IF_flush,CtrlSig);
Registers regs(clk,clk2, Ins_ID[25:21],
Ins_ID[20:16],RegWrite,Data1_ID,Data2_ID,WriteDataReg,dst,observe, data_ob,reset);
signExtend sE(Ins_ID[15:0], ExtendedIn);
ID_EX
id_ex(clk,reset,CtrlSig_ID,Data1_ID,Data2_ID,ExtendedIn,Ins_ID[25:21],Ins_ID[20:16],Ins_ID[15:11
],
WBSig_EX,MSig_EX,RegDst,ALUOp,ALUSrc,Data1_EX, Data2_EX, ExtendedOut,rs_EX,rt_EX, rd_EX);
ALUControl aluctrl(ExtendedOut[5:0],ALUOp,ALUCtrlSig);
ALU alu(A,ALUinput2,ALUCtrlSig,ALURes);
Forwarding fwd(RegWriteMEM,RegWrite,dst_MEM,dst,rs_EX,rt_EX,ForwardA,ForwardB);
EX_MEM
ex_mem(clk,reset,WBSig_EX,MSig_EX,ALURes,B,dst_EX,WBSig_MEM,MemRead,MemWrite,Address,WriteDataMe
m,dst_MEM);
dataMem dm(clk2, Address, MemWrite, MemRead,WriteDataMem,ReadDataMem,reset);
MEM_WB mem_wb(clk,reset, WBSig_MEM, ReadDataMem, Address,dst_MEM,dst,RegWrite,MemtoReg,
ReadDataWB, ALUresWB);

always@(clk) begin
$display("time %0d\t CLK=%b PC=%h\n",$time,clk,PC);
$display("[$s0]=%h,[$s1]=%h,[$s2]=%h \n",regs.register[16],regs.register[17],regs.register[18]);
$display("[$s3]=%h,[$s4]=%h,[$s5]=%h \n",regs.register[19],regs.register[20],regs.register[21]);
$display("[$s6]=%h,[$s7]=%h,[$t0]=%h \n",regs.register[22],regs.register[23],regs.register[8]);
$display("[$t1]=%h,[$t2]=%h,[$t3]=%h \n",regs.register[9],regs.register[10],regs.register[11]);
$display("[$t4]=%h,[$t5]=%h,[$t6]=%h \n",regs.register[12],regs.register[13],regs.register[14]);
$display("[$t7]=%h,[$t8]=%h,[$t9]=%h \n",regs.register[15],regs.register[24],regs.register[25]);
end


initial begin
#0 clk = 0; clk2=0; reset = 1;observe = 17;
#51 reset = 0;
end

always #50 clk = ~clk;
always #10 clk2 = ~clk2;
initial #3000 $stop;

endmodule
```

IV. Instruction Implementation

We test thirty instructions listed below by comparing the display of content in registers with the theoretical result.

```
memory[0] = 32'b00100000000010000000000000100000; //addi $t0, $zero, 0x20
memory[1] = 32'b00100000000010010000000000110111; //addi $t1, $zero, 0x37
memory[2] = 32'b00000001000010011000000000100100; //and $s0, $t0, $t1
memory[3] = 32'b00000001000010011000000000100101; //or $s0, $t0, $t1
memory[4] = 32'b10101100000100000000000000000100; //sw $s0, 4($zero)
memory[5] = 32'b10101100000010000000000000001000; //sw $t0, 8($zero)
memory[6] = 32'b00000001000010011000100000100000; //add $s1, $t0, $t1
memory[7] = 32'b00000001000010011001000000100010; //sub $s2, $t0, $t1
memory[8] = 32'b00010010001100100000000000001001; //beq $s1, $s2, error0
memory[9] = 32'b10001100000100010000000000000100; //lw $s1, 4($zero)
memory[10]= 32'b00110010001100100000000001001000; //andi $s2, $s1, 0x48
memory[11] =32'b00010010001100100000000000001001; //beq $s1, $s2, error1
memory[12] =32'b10001100000100110000000000001000; //lw $s3, 8($zero)
memory[13] =32'b00010010000100110000000000001010; //beq $s0, $s3, error2
memory[14] =32'b00000010010100011010000000101010; //slt $s4, $s2, $s1 (Last)
memory[15] =32'b00010010100000000000000000001111; //beq $s4, $0, EXIT
memory[16] =32'b00000010001000001001000000100000; //add $s2, $s1, $0
memory[17] =32'b00001000000000000000000000001110; //j Last
memory[18] =32'b00100000000010000000000000000000; //addi $t0, $0, 0(error0)
memory[19] =32'b00100000000010010000000000000000; //addi $t1, $0, 0
memory[20] =32'b00001000000000000000000000011111; //j EXIT
memory[21] =32'b00100000000010000000000000000001; //addi $t0, $0, 1(error1)
memory[22] =32'b00100000000010010000000000000001; //addi $t1, $0, 1
memory[23] =32'b00001000000000000000000000011111; //j EXIT
memory[24] =32'b00100000000010000000000000000010; //addi $t0, $0, 2(error2)
memory[25] =32'b00100000000010010000000000000010; //addi $t1, $0, 2
memory[26] =32'b00001000000000000000000000011111; //j EXIT
memory[27] =32'b00100000000010000000000000000011; //addi $t0, $0, 3(error3)
memory[28] =32'b00100000000010010000000000000011; //addi $t1, $0, 3
memory[29] =32'b00001000000000000000000000011111; //j EXIT
```

> The first instruction:

```
memory[0] = 32'b00100000000010000000000000100000; //addi $t0, $zero, 0x20
```

The content of register $t0 is changed to 0x20.

> The second instruction:

```
memory[1] = 32'b00100000000010010000000000110111; //addi $t1, $zero, 0x37
```

The content of register $t1 is changed to 0x37.

> The third instruction:

```
memory[2] = 32'b00000001000010011000000000100100; //and $s0, $t0, $t1
```

The content of register $s0 is changed to 0x20. The data hazard is solved with forwarding. The result of $t0 is passed from MEM/WB pipeline and the result of $t1 is passed from EX/MEM pipeline.

> The forth instruction:

```
memory[3] = 32'b00000001000010011000000000100101; //or $s0, $t0, $t1
```

The content of $s0 is changed to 0x37. The data hazard is solved with forwarding. The result of $t1 is passed from MEM/WB pipeline.

> The fifth instruction:

```
memory[4] = 32'b10101100000100000000000000000100; //sw $s0, 4($zero)
```

Nothing is changed in the register. The instruction is to store the content of $s0 to memory with offset 4. The data hazard is solved with forwarding. The result of $s0 is passed from EX/MEM pipeline.

> The sixth instruction:

```
memory[5] = 32'b10101100000010000000000000001000; //sw $t0, 8($zero)
```

Nothing is changed in the register. The instruction is to store the content of $t0 to memory with offset 8.

> The seventh instruction:

```
memory[6] = 32'b00000001000010011000100000100000; //add $s1, $t0, $t1
```

The content of $s1 is changed to 0x57.

> The eighth instruction:

```
memory[7] = 32'b00000001000010011001000000100010; //sub $s2, $t0, $t1
```

The content of $s2 is changed to 0xFFE9.

> The ninth instruction:

```
memory[8] = 32'b00010010001100100000000000001001; //beq $s1, $s2, error0
```

Since the content of $s1 is 0x57 and the content of $s2 is 0xFFE9, the address will not jump to error0.

> The tenth instruction:

```
memory[9] = 32'b10001100000100010000000000000100; //lw $s1, 4($zero)
```

The content of $s1 is changed to 0x37.

> The eleventh instruction:

```
memory[10]= 32'b00110010001100100000000001001000; //andi $s2, $s1, 0x48
```

The content of $s2 is changed to 0. When the load-use hazard is detected, stall and insert bubbles. The result of $s1 is passed from MEM/WB pipeline.

> The twelfth instruction:

```
memory[11] =32'b00010010001100100000000000001001; //beq $s1, $s2, error1
```

Since the content of $s2 is not equal to the content of $s1, the address will not jump to error1.

> The thirteenth instruction:

```
memory[12] =32'b10001100000100110000000000001000; //lw $s3, 8($zero)
```

The content of $s3 is changed to 0x20.

> The fourteenth instruction:

```
memory[13] =32'b00010010000100110000000000001010; //beq $s0, $s3, error2
```

Since the content of$ s0 is not equal to the content of $s3, the address will not jump to error2.

> The fifteenth instruction:

```
memory[14] =32'b00000010010100011010000000101010; //slt $s4, $s2, $s1 (Last)
```

Since the content of $s2 is less than s1, the content of $s4 is changed to 1.

> The sixteenth instruction:

```
memory[15] =32'b00010010100000000000000000001111; //beq $s4, $0, EXIT
```

Since the content of $s2 is less than s1, the content of $s4 is changed to 1. When the data hazard is detected, stall and insert bubbles. The address will not jump to EXIT.

> The seventeenth instruction:

```
memory[16] =32'b00000010001000001001000000100000; //add $s2, $s1, $0
```

The content of $s2 is changed to 0x37.

> The eighteenth instruction:

```
memory[17] =32'b00001000000000000000000000001110; //j Last
```

The address jumps to memory[14]. Since the content of $s2 is equal to the content of $s1, the content of $s4 is changed from 1 to 0. When PC reads instruction from memory[15], the program will jump to EXIT.

V. SSD and Top Module

The SSD module deals with the four digits on the FPGA board. When there is a difference between the anode and the cathode, then the corresponding diode will turn on, and it will be lighted on. The anode is always high voltage. For example, 4'b0000 corresponds to the digit "0". The middle diode will be off, and the others will be on. So only g=1 (no voltage difference on the diode "g") and a,b,c,d,e,f are all 0. The Verilog codes for the SSD module is

```
module SSDDriver(Q, Cathodes);
input [3:0]Q;
output [6:0] Cathodes;
reg a,b,c,d,e,f,g;
always @(Q) begin
case (Q)
4'b0000:  begin a=0;b=0;c=0;d=0;e=0;f=0;g=1; end
4'b0001:  begin a=1;b=0;c=0;d=1; e=1; f=1; g=1; end
4'b0010:  begin a=0;b=0;d=0;e=0;g=0;c=1; f=1; end
4'b0011:  begin a=0;b=0;c=0;d=0;g=0;e=1; f=1; end
4'b0100:  begin a=1;b=0;c=0;g=0;e=1; d=1; f=0; end
4'b0101:  begin a=0;c=0;d=0;g=0;f=0;b=1; e=1; end
4'b0110:  begin a=0;g=0;b=1;c=0;d=0;e=0;f=0; end
4'b0111:  begin a=0;b=0;c=0; d=1; e=1; f=1; g=1; end
4'b1000:  begin a=0;b=0;c=0;d=0;e=0;f=0; g=0; end
4'b1001:  begin a=0;b=0;c=0;f=0;g=0; d=0; e=1; end
4'b1010:  begin a=0;b=0;c=0;f=0;g=0; d=1; e=0; end
4'b1011:  begin a=1;b=1;c=0;f=0;g=0; d=0; e=0; end
4'b1100:  begin a=0;b=1;c=1;f=0;g=1; d=0; e=0; end
4'b1101:  begin a=1;b=0;c=0;f=1;g=0; d=0; e=0; end
4'b1110:  begin a=0;b=1;c=1;f=0;g=0; d=0; e=0; end
4'b1111:  begin a=0;b=1;c=1;f=0;g=0; d=1; e=0; end
default  begin  a=0;b=0;c=0;d=0;e=0;f=0;g=0; end
endcase
end

assign Cathodes={a,b,c,d,e,f,g};

endmodule
```

The top module uses two self-defined clocks. Since the actual frequency in the FPGA board is too high for human eyes to recognize, we must generate a slower clock. We use the counter that counts from zero up to 18'b0. The Clock_divider_500 module generates a 500Hz clock to accomplish this goal. The Clock module generates the clock signal that can be applied to register files and data memory, corresponds to the values in our sim module. Additionally, actually the four digits on the FPGA board can only demonstrate one single value. So in order to demonstrate something like 0037, we use the ring counter generating a four-bit sequence containing one zero and three ones. Zero corresponds to the working digit. Actually, every certain time only one digit on the board is on. But its clock cycle is to small for us to recognize. The Verilog codes for the top module is given by

```
module Clock_divider_500(clk_100M,clk_500);
reg [17:0]counter;
input clk_100M;
output clk_500;
reg clk_500;

initial begin
counter<=18'b0;
end

always@(posedge clk_100M) begin
if (counter==18'd200000)
    begin
        counter<=18'b0;
        clk_500<=1;
    end
else
    begin
        counter<=counter+1;
        clk_500<=0;
    end
end
endmodule



module top(
input clk_internal,clk, reset, sel,
input [4:0] observe,
output A3,A2,A1,A0,
output a,b,c,d,e,f,g);
wire [31:0] data_ob;
wire clk_500,clk_50;
wire [6:0]Cathodes0;
wire [6:0]Cathodes1;
wire [6:0]Cathodes2;
wire [6:0]Cathodes3;
wire [6:0]Cathodes4;
wire [6:0]Cathodes5;
wire [6:0]Cathodes6;
wire [6:0]Cathodes7;

Clock_divider_500 CD(clk_internal,clk_500);
Clock_divider_50 CD2(clk_500,clk_50);
main MAIN(clk, clk_50, reset, observe, data_ob);
```

```
SSDDriver UUT1(.Q(data_ob[3:0]),.Cathodes(Cathodes0));
SSDDriver UUT2(.Q(data_ob[7:4]),.Cathodes(Cathodes1));
SSDDriver UUT3(.Q(data_ob[11:8]),.Cathodes(Cathodes2));
SSDDriver UUT4(.Q(data_ob[15:12]),.Cathodes(Cathodes3));

SSDDriver UUT5(MAIN.PC[3:0],Cathodes4);
SSDDriver UUT6(MAIN.PC[7:4],Cathodes5);
SSDDriver UUT7(MAIN.PC[11:8],Cathodes6);
SSDDriver UUT8(MAIN.PC[15:12],Cathodes7);

four_bit_Ring_Counter UUT(clk_500,{A3,A2,A1,A0});

assign {a,b,c,d,e,f,g} = ({A3, A2, A1, A0}==4'b0111 && sel) ? Cathodes3 :
                         ({A3, A2, A1, A0}==4'b1011 && sel) ? Cathodes2 :
                         ({A3, A2, A1, A0}==4'b1101 && sel) ? Cathodes1 :
                         ({A3, A2, A1, A0}==4'b1110 && sel) ? Cathodes0 :
                         ({A3, A2, A1, A0}==4'b0111 && sel == 0) ? Cathodes7 :
                         ({A3, A2, A1, A0}==4'b1011 && sel == 0) ? Cathodes6 :
                         ({A3, A2, A1, A0}==4'b1101 && sel == 0) ? Cathodes5 :
                         ({A3, A2, A1, A0}==4'b1110 && sel == 0) ? Cathodes4 : Cathodes3;

endmodule
```

VI. RTL Schematic

Figure 4 shows our RTL schematic for a single-cycle CPU.

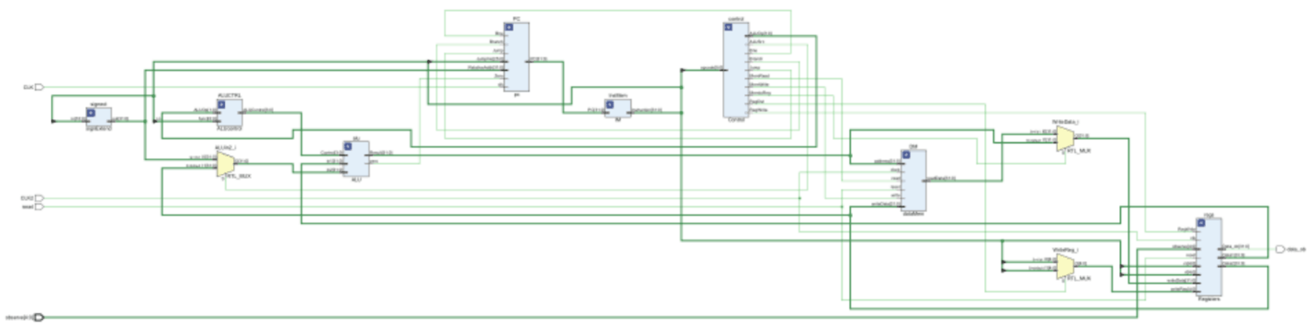

Figure 4: Single Cycle Diagram.

Figure 5 shows our RTL schematic for the top module of a pipelined CPU.

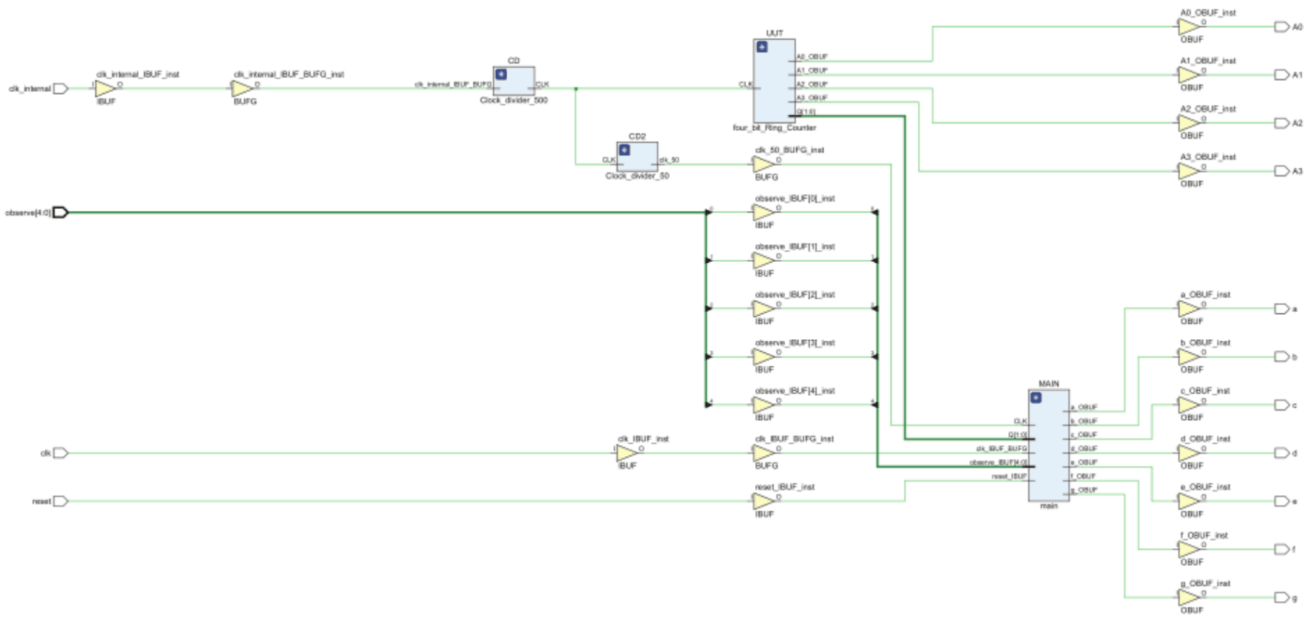

Figure 5: Schematic for Pipeline Top module.

Figure 6 shows our RTL schematic for the main module of the pipelined CPU.

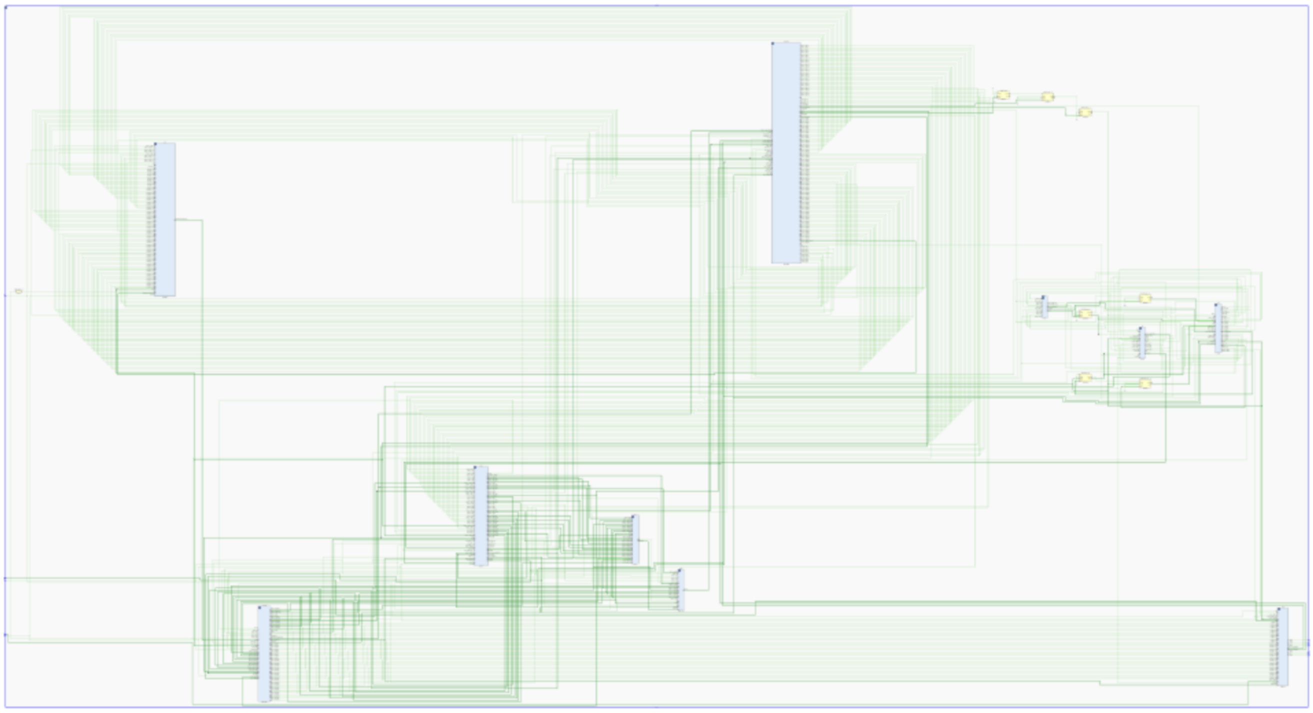

Figure 6: Schematic for Pipeline Main module.

VII. Textual Result

```
time 0   CLK=0 PC=xxxxxxxx

[$s0]=xxxxxxxx,[$s1]=xxxxxxxx,[$s2]=xxxxxxxx

[$s3]=xxxxxxxx,[$s4]=xxxxxxxx,[$s5]=xxxxxxxx
```

```
[$s6]=xxxxxxxx,[$s7]=xxxxxxxx,[$t0]=xxxxxxxx

[$t1]=xxxxxxxx,[$t2]=xxxxxxxx,[$t3]=xxxxxxxx

[$t4]=xxxxxxxx,[$t5]=xxxxxxxx,[$t6]=xxxxxxxx

[$t7]=xxxxxxxx,[$t8]=xxxxxxxx,[$t9]=xxxxxxxx

time 50  CLK=1 PC=xxxxxxxx

[$s0]=00000000,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000000

[$t1]=00000000,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 100     CLK=0 PC=00000000

[$s0]=00000000,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000000

[$t1]=00000000,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 150     CLK=1 PC=00000000

[$s0]=00000000,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000000

[$t1]=00000000,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 200     CLK=0 PC=00000004
```

```
[$s0]=00000000,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000000

[$t1]=00000000,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 250     CLK=1 PC=00000004

[$s0]=00000000,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000000

[$t1]=00000000,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 300     CLK=0 PC=00000008

[$s0]=00000000,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000000

[$t1]=00000000,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 350     CLK=1 PC=00000008

[$s0]=00000000,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000000

[$t1]=00000000,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000


[$t7]=00000000,[$t8]=00000000,[$t9]=00000000
```

```
time 400     CLK=0 PC=0000000c

[$s0]=00000000,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000000

[$t1]=00000000,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 450     CLK=1 PC=0000000c

[$s0]=00000000,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000000

[$t1]=00000000,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 500     CLK=0 PC=00000010

[$s0]=00000000,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000000,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 550     CLK=1 PC=00000010

[$s0]=00000000,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000000,[$t2]=00000000,[$t3]=00000000
```

```
[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 600     CLK=0 PC=00000014

[$s0]=00000000,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 650     CLK=1 PC=00000014

[$s0]=00000000,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 700     CLK=0 PC=00000018

[$s0]=00000020,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 750     CLK=1 PC=00000018

[$s0]=00000020,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000


[$s6]=00000000,[$s7]=00000000,[$t0]=00000020
```

```
[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 800     CLK=0 PC=0000001c

[$s0]=00000037,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 850     CLK=1 PC=0000001c

[$s0]=00000037,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 900     CLK=0 PC=00000020

[$s0]=00000037,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 950     CLK=1 PC=00000020

[$s0]=00000037,[$s1]=00000000,[$s2]=00000000
```

```
[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1000    CLK=0 PC=00000024

[$s0]=00000037,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1050    CLK=1 PC=00000024

[$s0]=00000037,[$s1]=00000000,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1100    CLK=0 PC=00000024

[$s0]=00000037,[$s1]=00000057,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000


time 1150    CLK=1 PC=00000024
```

```
[$s0]=00000037,[$s1]=00000057,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1200    CLK=0 PC=00000028

[$s0]=00000037,[$s1]=00000057,[$s2]=ffffffe9

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1250    CLK=1 PC=00000028

[$s0]=00000037,[$s1]=00000057,[$s2]=ffffffe9

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1300    CLK=0 PC=0000002c

[$s0]=00000037,[$s1]=00000057,[$s2]=ffffffe9

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000
```

```
[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1350     CLK=1 PC=0000002c

[$s0]=00000037,[$s1]=00000057,[$s2]=ffffffe9

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1400     CLK=0 PC=0000002c

[$s0]=00000037,[$s1]=00000057,[$s2]=ffffffe9

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1450     CLK=1 PC=0000002c

[$s0]=00000037,[$s1]=00000057,[$s2]=ffffffe9

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1500     CLK=0 PC=00000030

[$s0]=00000037,[$s1]=00000037,[$s2]=ffffffe9

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020


[$t1]=00000037,[$t2]=00000000,[$t3]=00000000
```

```
[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1550    CLK=1 PC=00000030

[$s0]=00000037,[$s1]=00000037,[$s2]=ffffffe9

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1600    CLK=0 PC=00000034

[$s0]=00000037,[$s1]=00000037,[$s2]=ffffffe9

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1650    CLK=1 PC=00000034

[$s0]=00000037,[$s1]=00000037,[$s2]=ffffffe9

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1700    CLK=0 PC=00000038

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000
```

```
[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1750    CLK=1 PC=00000038

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1800    CLK=0 PC=00000038

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1850    CLK=1 PC=00000038

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000000,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1900    CLK=0 PC=00000038


[$s0]=00000037,[$s1]=00000037,[$s2]=00000000
```

```
[$s3]=00000020,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 1950    CLK=1 PC=00000038

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000020,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2000    CLK=0 PC=0000003c

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000020,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2050    CLK=1 PC=0000003c

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000020,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000
```

```
time 2100    CLK=0 PC=00000040

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000020,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2150    CLK=1 PC=00000040

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000020,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2200    CLK=0 PC=00000040

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000020,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2250    CLK=1 PC=00000040

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000020,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000


[$t4]=00000000,[$t5]=00000000,[$t6]=00000000
```

```
[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2300    CLK=0 PC=00000044

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000020,[$s4]=00000001,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2350    CLK=1 PC=00000044

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000020,[$s4]=00000001,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2400    CLK=0 PC=00000048

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000020,[$s4]=00000001,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2450    CLK=1 PC=00000048

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000020,[$s4]=00000001,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020
```

```
[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2500    CLK=0 PC=00000038

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000020,[$s4]=00000001,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2550    CLK=1 PC=00000038

[$s0]=00000037,[$s1]=00000037,[$s2]=00000000

[$s3]=00000020,[$s4]=00000001,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2600    CLK=0 PC=0000003c

[$s0]=00000037,[$s1]=00000037,[$s2]=00000037

[$s3]=00000020,[$s4]=00000001,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2650    CLK=1 PC=0000003c

[$s0]=00000037,[$s1]=00000037,[$s2]=00000037


[$s3]=00000020,[$s4]=00000001,[$s5]=00000000
```

```
[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2700    CLK=0 PC=00000040

[$s0]=00000037,[$s1]=00000037,[$s2]=00000037

[$s3]=00000020,[$s4]=00000001,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2750    CLK=1 PC=00000040

[$s0]=00000037,[$s1]=00000037,[$s2]=00000037

[$s3]=00000020,[$s4]=00000001,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2800    CLK=0 PC=00000040

[$s0]=00000037,[$s1]=00000037,[$s2]=00000037

[$s3]=00000020,[$s4]=00000001,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2850    CLK=1 PC=00000040
```

```
[$s0]=00000037,[$s1]=00000037,[$s2]=00000037

[$s3]=00000020,[$s4]=00000001,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2900    CLK=0 PC=0000007c

[$s0]=00000037,[$s1]=00000037,[$s2]=00000037

[$s3]=00000020,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000

time 2950    CLK=1 PC=0000007c

[$s0]=00000037,[$s1]=00000037,[$s2]=00000037

[$s3]=00000020,[$s4]=00000000,[$s5]=00000000

[$s6]=00000000,[$s7]=00000000,[$t0]=00000020

[$t1]=00000037,[$t2]=00000000,[$t3]=00000000

[$t4]=00000000,[$t5]=00000000,[$t6]=00000000

[$t7]=00000000,[$t8]=00000000,[$t9]=00000000
```

VIII. Conclusion and Discussion

In this big project, we were asked to model the working principle of both a single cycle and a pipelined CPU in Verilog that support a subset of MIPS instruction set including lw, sw, R-type, slt, jump, beq and bne. We successfully accomplished this project and everyone did a great job. The codes we write follow what is required in the project manual. The simulation results were right and the demonstration went on well.

When debugging our project, we first run the whole testing program in our mind, and we drew a diagram showing the pipeline stage every instruction should be in in every clock cycle. Also, we drew another diagram showing the theoretical values in the registers. So our debugging is quite efficient: we do not need to double check all parts of our codes, but we only need to focus on our simulation results and compare it with our hand-drawn diagrams, find the difference, and try to find out the reason and fix the problem.

The greatest difficulties during the debugging stage we met were related to data hazards and control hazards. When we just finished writing our codes, the forwarding unit could only forward data to EX stage. But we found that when we encounter some branch instruction, we may need the result of the last instruction during the ID stage of the branch instruction. In this case, we will need to forward some data to ID stage, which was omitted by us earlier.

Another problem we faced was when we encountered some instruction sequence where a beq instruction follows a lw instruction, we may need to add two nope instructions in between. But the original code would flush the branch instruction when detecting a control hazard, so it was not correct. We fixed the problem by adding another input to the control module so that the branch instruction could be reserved when the first nope was added, and during the very next clock cycle, a hazard could still be detected, thus adding another nope.

Overall, having great partners to work with is a very great joy for all of us. This is indeed the true spirit for team works.

Appendix: Codes for the Single-Cycle CPU

```
module ALU(
input [31:0] in1,in2,
input [3:0] Control,
output zero,
output [31:0] Result);
assign Result = (Control == 4'b0010) ? (in1 + in2):
                (Control == 4'b0110) ? (in1 - in2):
                (Control == 4'b0000) ? (in1 & in2):
                (Control == 4'b0001) ? (in1 | in2):
                ((Control == 4'b0111) && (in1 < in2)) ? 1:0;
assign zero = (Result==0) ? 1:0;

endmodule





module ALUcontrol(
input [5:0] funct,
input [1:0] ALUOp,
output reg [3:0] ALUControl);
always @(funct or ALUOp)
begin
    case(ALUOp)
    2'b00: ALUControl <= 4'b0010;
    2'b01: ALUControl <= 4'b0110;
    2'b11: ALUControl <= 4'b0000;
    2'b10:begin
        case(funct)

        6'b100000:ALUControl <= 4'b0010;
        6'b100010:ALUControl <= 4'b0110;
        6'b100100:ALUControl <= 4'b0000;
        6'b100101:ALUControl <= 4'b0001;
        6'b101010:ALUControl <= 4'b0111;
        endcase
    end
    endcase
end
endmodule


module Control(
input [5:0]opcode,
output reg RegDst,Jump, Branch, Bne, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite,
output reg [1:0] ALUOp);
initial begin
RegDst <= 0;
Jump <= 0;
Branch <= 0;
Bne <= 0;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 0;
RegWrite <= 0;
ALUOp <= 2'b00;
end
always @(opcode) begin
case(opcode)
6'b000000://R
begin
RegDst <= 1;
Jump <= 0;
Branch <= 0;
Bne <= 0;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 0;
RegWrite <= 1;
ALUOp <= 2'b10;
end
6'b001000://addi
begin
RegDst <= 0;
Jump <= 0;
Branch <= 0;
Bne <= 0;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 1;

RegWrite <= 1;
```

```
ALUOp <= 2'b00;
end
6'b001100://andi
begin
RegDst <= 0;
Jump <= 0;
Branch <= 0;
Bne <= 0;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 1;
RegWrite <= 1;
ALUOp <= 2'b11;
end
6'b000100://beq
begin
RegDst <= 0;
Jump <= 0;
Branch <= 1;
Bne <= 0;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 0;
RegWrite <= 0;
ALUOp <= 2'b01;
end
6'b000101://bne
begin
RegDst <= 0;
Jump <= 0;
Branch <= 0;
Bne <= 1;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 0;
RegWrite <= 0;
ALUOp <= 2'b01;
end
6'b000010://j
begin
RegDst <= 0;
Jump <= 1;
Branch <= 0;
Bne <= 0;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 0;
ALUSrc <= 0;
RegWrite <= 0;

ALUOp <= 2'b00;
```

```
end
6'b100011://lw
begin
RegDst <= 0;
Jump <= 0;
Branch <= 0;
Bne <= 0;
MemRead <= 1;
MemtoReg <= 1;
MemWrite <= 0;
ALUSrc <= 1;
RegWrite <= 1;
ALUOp <= 2'b00;
end
6'b101011://sw
begin
RegDst <= 0;
Jump <= 0;
Branch <= 0;
Bne <= 0;
MemRead <= 0;
MemtoReg <= 0;
MemWrite <= 1;
ALUSrc <= 1;
RegWrite <= 0;
ALUOp <= 2'b00;
end
endcase
end
endmodule


module dataMem(clock, address, write, read, writeData,  readData, reset);
input wire read, write, reset, clock;
input wire[31:0] writeData, address;
output reg [31:0] readData;
reg [31:0] memory [0:63];
integer i;

initial begin
for(i=0;i<64;i=i+1) memory[i] <= 32'd0;
end
always @(negedge clock)
if(read && readData != memory[address])
readData <= memory[address];
else readData <= 0;

always @(posedge clock)
begin
if (reset)
begin
for(i=0;i<64;i=i+1) memory[i] <= 32'd0;

end
```

```
else if (write)
        memory[address] <= writeData;
end
endmodule


module IM(Instruction,PC);
output reg [31:0]Instruction;
input [31:0] PC;
wire [7:0] index;
reg [31:0] memory[0:63];
integer i;
initial begin
memory[0] = 32'b00100000000010000000000000100000; //addi $t0, $zero, 0x20
memory[1] = 32'b00100000000010010000000000110111; //addi $t1, $zero, 0x37
memory[2] = 32'b00000001000010011000000000100100; //and $s0, $t0, $t1
memory[3] = 32'b00000001000010011000000000100101; //or $s0, $t0, $t1
memory[4] = 32'b10101100000100000000000000000100; //sw $s0, 4($zero)
memory[5] = 32'b10101100000010000000000000001000; //sw $t0, 8($zero)
memory[6] = 32'b00000001000010011000100000100000; //add $s1, $t0, $t1
memory[7] = 32'b00000001000010011001000000100010; //sub $s2, $t0, $t1
memory[8] = 32'b00010010001100100000000000001001; //beq $s1, $s2, error0
memory[9] = 32'b10001100000100010000000000000100; //lw $s1, 4($zero)
memory[10]= 32'b00110010001100100000000001001000; //andi $s2, $s1, 0x48
memory[11] =32'b00010010001100100000000000001001; //beq $s1, $s2, error1
memory[12] =32'b10001100000100110000000000001000; //lw $s3, 8($zero)
memory[13] =32'b00010010000100110000000000001010; //beq $s0, $s3, error2
memory[14] =32'b00000010010100011010000000101010; //slt $s4, $s2, $s1 (Last)
memory[15] =32'b00010010100000000000000000001111; //beq $s4, $0, EXIT
memory[16] =32'b00000010001000001001000000100000; //add $s2, $s1, $0
memory[17] =32'b00001000000000000000000000001110; //j Last
memory[18] =32'b00100000000010000000000000000000; //addi $t0, $0, 0(error0)
memory[19] =32'b00100000000010010000000000000000; //addi $t1, $0, 0
memory[20] =32'b00001000000000000000000000011111; //j EXIT
memory[21] =32'b00100000000010000000000000000001; //addi $t0, $0, 1(error1)
memory[22] =32'b00100000000010010000000000000001; //addi $t1, $0, 1
memory[23] =32'b00001000000000000000000000011111; //j EXIT
memory[24] =32'b00100000000010000000000000000010; //addi $t0, $0, 2(error2)
memory[25] =32'b00100000000010010000000000000010; //addi $t1, $0, 2
memory[26] =32'b00001000000000000000000000011111; //j EXIT
memory[27] =32'b00100000000010000000000000000011; //addi $t0, $0, 3(error3)
memory[28] =32'b00100000000010010000000000000011; //addi $t1, $0, 3
memory[29] =32'b00001000000000000000000000011111; //j EXIT
for(i=30;i<64;i=i+1) memory[i] <= 32'd0;
end
assign index = PC[9:2];
always @(PC)
begin
Instruction <= memory[index];
end
//$display("the instruction now is %d", Instruction);
endmodule
```

```
module main(CLK, CLK2, reset, observe, data_ob);
input CLK,CLK2,reset;
input [4:0] observe;
output data_ob;
wire [31:0] Instruction, RelativeValue;
wire [31:0] PCin,PCout;
wire [31:0]ReadData1,ReadData2,WriteData, DataMemRead,ALURes;
wire [31:0] ALUin1, ALUin2;
wire RegDst, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite;
wire Jump, Branch, Bne,Zero;
wire [1:0] ALUOp;
wire [3:0] ALUControl;
wire [4:0] ReadReg1, ReadReg2, WriteReg;

assign WriteReg = RegDst ? {Instruction[15:11]}:{Instruction[20:16]};
assign ReadReg1 = {Instruction[25:21]};
assign ReadReg2 = {Instruction[20:16]};
assign WriteData = MemtoReg ? DataMemRead : ALURes;
assign ALUin1 = ReadData1;
assign ALUin2 = ALUSrc ? RelativeValue : ReadData2;

signExtend signext(.in(Instruction[15:0]), .out(RelativeValue));
pc PC(.clk(CLK), .JumpIns(Instruction[25:0]), .RelativeAddr(RelativeValue), .Jump(Jump),
.Branch(Branch), .Bne(Bne), .Zero(Zero), .newPC(PCin), .PC(PCout));
IM InsMem(.Instruction(Instruction), .PC(PCout));
Control control(Instruction[31:26], RegDst, Jump, Branch,Bne, MemRead, MemtoReg, MemWrite,
ALUSrc, RegWrite, ALUOp);
Registers regs(CLK2, ReadReg1, ReadReg2, RegWrite, ReadData1, ReadData2, WriteData,
WriteReg,observe,data_ob,reset);
ALUcontrol ALUCTRL(Instruction[5:0], ALUOp, ALUControl);
ALU alu(ALUin1, ALUin2, ALUControl, Zero, ALURes);
dataMem DM(CLK2, ALURes, MemWrite, MemRead, ReadData2, DataMemRead, reset);

endmodule


module pc(clk, JumpIns, RelativeAddr, Jump, Branch, Bne, Zero, newPC,PC);
output reg [31:0] newPC, PC;
input clk, Jump, Branch, Bne, Zero;
input [25:0] JumpIns;
input [31:0] RelativeAddr;
wire Jump,Branch,Zero;
integer i;
wire [31:0] JumpAddr, BranchAddr;
wire [31:0] PCadd, ALUResult;


assign PCadd = PC + 4'b0100;
assign ALUResult = PCadd + RelativeAddr * 4;
assign JumpAddr = {PCadd[31:28],JumpIns[25:0],{1'b0,1'b0}};


assign BranchAddr = ({Branch,Zero} == {1'b1,1'b1} || {Bne,Zero} == {1'b1,1'b0}) ? ALUResult :
```

```
PCadd;
initial begin
PC = 0;
end

always@(*)begin
case(Jump)
    1'b0: newPC <= BranchAddr;
    1'b1: newPC <= JumpAddr;
    default:;
endcase
end

always @(posedge clk) begin
    begin
    PC <= newPC;
    end
end
endmodule


module Registers(clk, rs, rt,RegWrite, Data1, Data2, writeData, writeReg,observe,Data_ob,reset);
input clk,RegWrite, reset;
input [4:0] rs;
input [4:0] rt;
input [4:0] observe;
output [31:0] Data1, Data2, Data_ob;
input wire [4:0] writeReg;
input wire [31:0] writeData;
reg [31:0] register[0:31];
integer i;
initial begin
    for (i = 0; i < 32; i = i+1) register[i] <= 0; //Initialize the registers
end

assign Data_ob = register[observe];

always @(posedge clk)
begin
if (reset)
begin
    for (i = 0; i < 32; i = i+1) register[i] <= 0; //Initialize the registers
end
else if(RegWrite == 1 && writeReg != 0) register[writeReg] <= writeData;
end
assign Data1 = register[rs];
assign Data2 = register[rt];

endmodule


module signExtend(in, out);

input [15:0] in;
```

```
output [31:0] out;
assign out = {{16{in[15]}}, in[15:0]};

endmodule


module sim;
reg CLK,CLK2,reset;
reg [4:0] observe;
wire [31:0] Instruction, RelativeValue, data_ob;
wire [31:0] PCin,PCout;
wire [31:0]ReadData1,ReadData2,WriteData, DataMemRead,ALURes;
wire [31:0] ALUin1, ALUin2;
wire RegDst, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite;
wire Jump, Branch, Bne,Zero;
wire [1:0] ALUOp;
wire [3:0] ALUControl;
wire [4:0] ReadReg1, ReadReg2, WriteReg;

assign WriteReg = RegDst ? {Instruction[15:11]}:{Instruction[20:16]};
assign ReadReg1 = {Instruction[25:21]};
assign ReadReg2 = {Instruction[20:16]};
assign WriteData = MemtoReg ? DataMemRead : ALURes;
assign ALUin1 = ReadData1;
assign ALUin2 = ALUSrc ? RelativeValue : ReadData2;

signExtend signext(.in(Instruction[15:0]), .out(RelativeValue));
pc PC(.clk(CLK), .JumpIns(Instruction[25:0]), .RelativeAddr(RelativeValue), .Jump(Jump),
.Branch(Branch), .Bne(Bne), .Zero(Zero), .newPC(PCin), .PC(PCout));
IM InsMem(.Instruction(Instruction), .PC(PCout));
Control control(Instruction[31:26], RegDst, Jump, Branch,Bne, MemRead, MemtoReg, MemWrite,
ALUSrc, RegWrite, ALUOp);
Registers regs(CLK2, ReadReg1, ReadReg2, RegWrite, ReadData1, ReadData2, WriteData,
WriteReg,observe,data_ob,reset);
ALUcontrol ALUCTRL(Instruction[5:0], ALUOp, ALUControl);
ALU alu(ALUin1, ALUin2, ALUControl, Zero, ALURes);
dataMem DM(CLK2, ALURes, MemWrite, MemRead, ReadData2, DataMemRead, reset);

initial begin
#0 CLK = 0;CLK2 = 0; reset = 0; observe = 20;
#51 reset=0;
end


always #50 CLK = ~CLK;
always #10 CLK2 = ~CLK2;
initial #3000 $stop;
endmodule
```