計算機組織

Project3

四電機三乙

劉錕笙

B10607118

程式截圖:

pipelineCPU.v

```
module PipelineCPU(
        input [31:0] Addr_in,
        input clk,
        output [31:0] Addr_o
        );

reg [31:0]WB_REG=0;
reg wr_reg=0;
reg [4:0]RD_REG=0;

wire ALUSrc,RegWrite,RegDst,MemRead,MemWrite,MemtoReg,Jump,Branch,DoBranch;
wire zero_EX,zero_MEM,stall;
wire [31:0]RT_ID,RT_EX,RT_MUX;
wire [31:0]Instruction_IF,Instrucion_ID;
wire [31:0]Immediately_ID,Immediately_EX;
wire [31:0]Result_EX,Result_MEM,Result_WB,WriteData_MEM;
IM my_IM(.Addr_in(Addr_in),.Instruction(Instruction_IF));//Fatch Instrucion

IF2ID_Register my_IF2ID(.clk(clk),.instruction(Instruction_IF),.instruction_o(Instrucion_ID)); //IF/ID

//-------------------------------------     ID      ----------------------------------------------------//

Hazard_Detection myDetection(.MemRead(M_EX[1]),.RS_addr(Instrucion_ID[25:21]),.RT_addr(Instrucion_ID[20:16]),.RT_ID(RT_addr),.stall(stall));

Control my_Ctrl(.Op(Instrucion_ID[31:26]),.ALUOp(ALUOp),.RegWrite(RegWrite),.RegDst(RegDst),.ALUSrc(ALUSrc),
        .MemWrite(MemWrite),.MemRead(MemRead),.MemtoReg(MemtoReg),.Branch(Branch),.Jump(Jump));

RF my_RF(.RegWrite(WB_WB[1]),.RS_Address(Instrucion_ID[25:21]),.RT_Address(Instrucion_ID[20:16]),
        .RD_Address(RD_WB),.RSdata(RS_ID),.RTdata(RT_ID),.RDdata(MUX32b2RDwr),.clk(clk));

SE my_SE(.data_i(Instrucion_ID[15:0]),.data_o(Immediately_ID));

ID2EX_Register my_ID2EX(.clk(clk),.WB({RegWrite,MemtoReg}),.M({Branch,MemRead,MemWrite}),.EX({RegDst,ALUSrc,ALUOp}),
                        .RS(RS_ID),.RT(RT_ID),.Immediately(Immediately_ID),.RD1(Instrucion_ID[15:11]),.RD2(Instrucion_ID[20:16]),
                        .RS_addr(Instrucion_ID[25:21]),.RT_addr(Instrucion_ID[20:16]),
                        .WB_o(WB_EX),.M_o(M_EX),.EX_o(EX_EX),.RS_o(RS_EX),.RT_o(RT_EX),.Immediately_o(Immediately_EX),
                        .RD1_o(EX_RD1),.RD2_o(EX_RD2),.RS_addr_o(RS_addr),.RT_addr_o(RT_addr));

//-------------------------------------     EX      ----------------------------------------------------//

MUX5b RD_MUX(.data0(EX_RD1),.data1(EX_RD2),.select(RegDst),.data_o(RD_EX));

ALUcrtl my_ALUcrtl(.funct(Immediately_EX[5:0]),.ALUOp(EX_EX[1:0]),.operation(operation));

Forwarding_unit myForwarding_unit(.wr_MEM(WB_MEM[1]),.wr_WB(WB_WB[1]),.wr_REG(wr_reg),.RS_ID(RS_addr),.RT_ID(RT_addr),.RD_MEM(RD_MEM),
                                  .RD_WB(RD_WB),.RD_REG(RD_REG),.ForwardA(Forward_RS),.ForwardB(Forward_RT));

MUX32b_3X1 RS_hazard(.select(Forward_RS),.ID(RS_EX),.MEM(Result_MEM),.WB(MUX32b2RDwr),.REG(WB_REG),.out(RS_MUX));

MUX32b_3X1 RT_hazard(.select(Forward_RT),.ID(RT_EX),.MEM(Result_MEM),.WB(MUX32b2RDwr),.REG(WB_REG),.out(RT_MUX));

MUX32b ALU_MUX(.data0(RT_MUX),.data1(Immediately_EX),.select(EX_EX[2]),.data_o(MUX32b2ALU));

ALU my_ALU(.Source1(RS_MUX),.Source2(MUX32b2ALU),.operation(operation),.shamt(Immediately_EX[10:6]),.result(Result_EX),.zero(zero_EX));

EX2MEM_Register my_EX2MEM(.clk(clk),.WB(WB_EX),.M(M_EX),.ALU_result(Result_EX),.WriteData(RT_MUX),.zero(zero_EX),.RD(RD_EX),
       .WB_o(WB_MEM),.M_o(M_MEM),.ALU_result_o(Result_MEM),.WriteData_o(WriteData_MEM),.zero_o(zero_MEM),.RD_o(RD_MEM));

//-------------------------------------     MEM      ----------------------------------------------------//

and a1(DoBranch,zero_MEM,M_MEM[2]);

DM my_DM(.clk(clk),.Address(Result_MEM),.data(WriteData_MEM),.MemRead(M_MEM[1]),.MemWrite(M_MEM[0]),.DM_data(DMdata_MEM));

MEM2WB_Register myMEM2WB(.clk(clk),.WB(WB_MEM),.MemData(DMdata_MEM),.ALU_result(Result_MEM),.RD(RD_MEM),.WB_o(WB_WB),.MemData_o(DMdata_WB),
                         .ALU_result_o(Result_WB),.RD_o(RD_WB));

//-------------------------------------     WB      ----------------------------------------------------//

MUX32b RDwr_MUX(.data0(Result_WB),.data1(DMdata_WB),.select(WB_WB[0]),.data_o(MUX32b2RDwr));


//-------------------------------------     WB      ----------------------------------------------------//

always @(posedge clk)begin
WB_REG<=MUX32b2RDwr;
wr_reg<=WB_WB[1];
RD_REG<=RD_WB;
end


endmodule
```

描述:依照各 **stage** 加上 **3** 個 **reg**，並連接起來

PC.v:沒有把 PC 額外獨立出來

IM.v

```
module IM(
        //input clk,
        //input stall,
        input [31:0]Addr_in,
        output[31:0]Instruction
        );
reg [7:0]Instruction_list [83:0];

assign Instruction={Instruction_list[Addr_in],Instruction_list[Addr_in+1],Instruction_list[Addr_in+2],Instruction_list[Addr_in+3]};

initial begin

        //add $t0, $t1, $t1
        {Instruction_list[0],Instruction_list[1],Instruction_list[2],Instruction_list[3]}<={6'd20,5'd9,5'd9,5'd8,5'd0,6'd21};
        //sub $t1, $t2, $t4
        {Instruction_list[4],Instruction_list[5],Instruction_list[6],Instruction_list[7]}<={6'd20,5'd10,5'd12,5'd9,5'd0,6'd22};
        //srl $t4, $t5, 2
        {Instruction_list[8],Instruction_list[9],Instruction_list[10],Instruction_list[11]}<={6'd20,5'd13,5'dz,5'd12,5'd2,6'd23};
        //sll $t6, $t6, 4
        {Instruction_list[12],Instruction_list[13],Instruction_list[14],Instruction_list[15]}<={6'd20,5'd14,5'dz,5'd14,5'd4,6'd24};
        //xor $t3, $t1, $t2
        {Instruction_list[16],Instruction_list[17],Instruction_list[18],Instruction_list[19]}<={6'd20,5'd9,5'd10,5'd11,5'd0,6'd25};
        //and $t5, $t4, $t2
        {Instruction_list[20],Instruction_list[21],Instruction_list[22],Instruction_list[23]}<={6'd20,5'd12,5'd10,5'd13,5'd0,6'd26};
        //sw $t0, 2($t7)
        {Instruction_list[24],Instruction_list[25],Instruction_list[26],Instruction_list[27]}<={6'd43,5'd15,5'd8,16'd2};
        //lw $s3, 2($t7)
        {Instruction_list[28],Instruction_list[29],Instruction_list[30],Instruction_list[31]}<={6'd35,5'd15,5'd19,16'd2};
        //sw $s4, 4($t7)
        {Instruction_list[32],Instruction_list[33],Instruction_list[34],Instruction_list[35]}<={6'd43,5'd15,5'd20,16'd4};
        //sw $t0, 2($t2)
        {Instruction_list[36],Instruction_list[37],Instruction_list[38],Instruction_list[39]}<={6'd43,5'd10,5'd8,16'd2};
        //lw $s4, 3($t2)
        {Instruction_list[40],Instruction_list[41],Instruction_list[42],Instruction_list[43]}<={6'd35,5'd10,5'd20,16'd3};

//-------------------------------------------------------------//
```

PC 16、PC 20 會發生 datahazard，因為在 RF 讀取時資料還在 WB 等待下個 clk 寫入 RF，但是寫入後 RF 已經讀取完畢進入下一級，所以取得的資料是錯的。

```
//-------------------------------------------------------------//

        //add $t6, $t5, $t4
        {Instruction_list[44],Instruction_list[45],Instruction_list[46],Instruction_list[47]}<={6'd20,5'd13,5'd12,5'd14,5'd0,6'd21};
        //sub $t7, $t6, $t5
        {Instruction_list[48],Instruction_list[49],Instruction_list[50],Instruction_list[51]}<={6'd20,5'd14,5'd13,5'd15,5'd0,6'd22};
        //add $t3, $t6, $t7
        {Instruction_list[52],Instruction_list[53],Instruction_list[54],Instruction_list[55]}<={6'd20,5'd14,5'd15,5'd11,5'd0,6'd21};
        //lw $s1, 2($t2)
        {Instruction_list[56],Instruction_list[57],Instruction_list[58],Instruction_list[59]}<={6'd35,5'd10,5'd17,16'd2};
        //sw $s1, 2($t5)
        {Instruction_list[60],Instruction_list[61],Instruction_list[62],Instruction_list[63]}<={6'd43,5'd13,5'd17,16'd2};
        //lw $t1, 2($t5)
        {Instruction_list[64],Instruction_list[65],Instruction_list[66],Instruction_list[67]}<={6'd35,5'd13,5'd9,16'd2};
        //add $t2, $t1, $t1
        {Instruction_list[68],Instruction_list[69],Instruction_list[70],Instruction_list[71]}<={6'd20,5'd9,5'd9,5'd10,5'd0,6'd21};

//-------------------------------------------------------------//*/
end
endmodule
```

PC48: $t6 還在 EX，還沒寫回 RF，但下一級要使用

PC52: $t7 還在 **EX**，還沒寫回 **RF**，但下一級要使用，且**$t6** 還在 **MEM**，還沒寫回 **RF**，但下一級要使用

PC60: $s1 在上一個 **PC** 要被更新，但是還沒更新就取值

PC68: $t1 在上一個 **PC** 要被更新，但是還沒更新就取值

IF_ID.v

```
module IF2ID_Register(
        input clk,stall,
        input [31:0]instruction,
        output reg [31:0]instruction_o);

initial begin

instruction_o=0;

end


always @(posedge clk,posedge stall)begin

instruction_o<=instruction;


end
endmodule
```

Control.v

```
module Control(
        input [5:0] Op,
        output reg [1:0] ALUOp,
        output reg RegDst,
        output reg MemRead,
        output reg MemtoReg,
        output reg MemWrite,
        output reg ALUSrc,
        output reg RegWrite,
        output reg Jump,
        output reg Branch
        );
always @(Op)begin
case(Op)
        6'd20:begin
                ALUOp<=2'b10;
                RegDst<=0;
                MemRead<=0;
                MemWrite<=0;
                MemtoReg<=0;
                ALUSrc<=0;
                RegWrite<=1;
                Jump<=0;
                Branch<=0;
        end
        6'd43:begin
                ALUOp<=2'b00;
                RegDst<=1;
                MemRead<=0;
                MemWrite<=1;
                MemtoReg<=0;
                ALUSrc<=1;
                RegWrite<=0;
                Jump<=0;
                Branch<=0;
        end
        6'd35:begin
                ALUOp<=2'b00;
                RegDst<=1;
                MemRead<=1;
                MemWrite<=0;
                MemtoReg<=1;
                ALUSrc<=1;
                RegWrite<=1;
                Jump<=0;
                Branch<=0;
        end
        6'd8:begin
                ALUOp<=2'b00;
                RegDst<=1;
                MemRead<=0;
                MemWrite<=0;
                MemtoReg<=0;
                ALUSrc<=1;
                RegWrite<=1;
                Jump<=0;
                Branch<=0;
        end
        6'd9:begin
                ALUOp<=2'b01;
                RegDst<=1;
                MemRead<=0;
                MemtoReg<=0;
                MemWrite<=0;
                ALUSrc<=1;
                RegWrite<=1;
                Jump<=0;
                Branch<=0;
        end
        6'd4:begin
                ALUOp<=2'b01;
                RegDst<=0;
                MemRead<=0;
                MemWrite<=0;
                MemtoReg<=0;
                ALUSrc<=0;
                RegWrite<=0;
                Jump<=0;
                Branch<=1;
        end
        6'd2:begin
                ALUOp<=2'b01;
                RegDst<=0;
                MemRead<=0;
                MemWrite<=0;
                MemtoReg<=0;
                ALUSrc<=0;
                RegWrite<=0;
                Jump<=1;
                Branch<=0;
        end
endcase

end
endmodule
```

RF.v

```
module RF(
        input clk,
        input RegWrite,
        input [4:0] RS_Address,
        input [4:0] RT_Address,
        input [4:0] RD_Address,
        output reg [31:0] RSdata,
        output reg [31:0] RTdata,
        input [31:0] RDdata
        );
reg [31:0] Register[31:0];

initial begin
always @(RS_Address,RT_Address)begin
RSdata=Register[RS_Address];
RTdata=Register[RT_Address];

end

always @(posedge clk)begin
if(RegWrite)Register[RD_Address]=RDdata;

end


endmodule
```

```
initial begin

Register[0]=32'd0;
Register[1]=32'd11;
Register[2]=32'd370;
Register[3]=32'd183;
Register[4]=32'd91;
Register[5]=32'd234;
Register[6]=32'd53;
Register[7]=32'd124;
Register[8]=32'd317;
Register[9]=32'd179;
Register[10]=32'd101;
Register[11]=32'd161;
Register[12]=32'd77;
Register[13]=32'd320;
Register[14]=32'd152;
Register[15]=32'd10;
Register[16]=32'd100;
Register[17]=32'd100;
Register[18]=32'd245;
Register[19]=32'd19;
Register[20]=32'd2;
Register[21]=32'd13;
Register[22]=32'd262;
Register[23]=32'd185;
Register[24]=32'd180;
Register[25]=32'd180;
Register[26]=32'd198;
Register[27]=32'd178;
Register[28]=32'd235;
Register[29]=32'd22;
Register[30]=32'd1000;
Register[31]=32'd75;


end
```

SE.v

```
module SE(
        input [15:0]data_i,
        output [31:0]data_o
        );

assign data_o=data_i[15]?{16'hFFFF,data_i}:{16'h0000,data_i};

endmodule
```

ID_EX.v

```
module ID2EX_Register(
        input clk,
        input [1:0]WB, //{RegWrite,MemToReg}
        input [2:0]M,  //{Branch,MemRead,MemWrite}
        input [3:0]EX, //{RegDst,ALUSrc,ALUOP[1:0]}
        input [4:0]RD1,RD2,RS_addr,RT_addr,
        input [31:0]RS,RT,Immediately,
        output reg [1:0]WB_o,
        output reg [2:0]M_o,
        output reg [3:0]EX_o,
        output reg [4:0]RD1_o,RD2_o,RS_addr_o,RT_addr_o,
        output reg [31:0]RS_o,RT_o,Immediately_o);

initial begin
WB_o=0;
M_o=0;
EX_o=0;
RS_o=0;
RT_o=0;
Immediately_o=0;
RD1_o=0;
RD2_o=0;
RS_addr_o=0;
RT_addr_o=0;
end

always @(posedge clk)begin

WB_o<=WB;
M_o<=M;
EX_o<=EX;
RS_o<=RS;
RT_o<=RT;
Immediately_o<=Immediately;
RD1_o<=RD1;
RD2_o<=RD2;
RS_addr_o<=RS_addr;
RT_addr_o<=RT_addr;

end
endmodule
```

MUX5b.v

```
module MUX5b(
         input [4:0]data0,
         input [4:0]data1,
         input select,
         output [4:0]data_o);

assign data_o=select?data1:data0;

endmodule

module MUX32b(
         input [31:0]data0,
         input [31:0]data1,
         input select,
         output [31:0]data_o);

assign data_o=select?data1:data0;

endmodule
```

MUX9b.v 沒有獨立成 module

ALUctrl.v

```
module ALUcrtl(
        input [5:0] funct,
        input [1:0] ALUOp,
        output reg[5:0] operation
        );

always @(ALUOp,funct)begin
if(ALUOp[1]==1)begin
        case(funct)
                6'd21:operation=6'd27;
                6'd22:operation=6'd28;
                6'd23:operation=6'd29;
                6'd24:operation=6'd30;
                6'd25:operation=6'd31;
                6'd26:operation=6'd32;
        endcase
end
else if(ALUOp[0]==0)operation=6'd27;
else if(ALUOp[0]==1)operation=6'd28;

end

endmodule
```

MUX3X1.v

```
module MUX32b_3X1(
        input [1:0]select,
        input [31:0]ID,MEM,WB,REG,
        output reg [31:0]out);

initial out=0;

always @(ID,MEM,WB,REG,select)begin

case(select)
        2'b00:out=ID;
        2'b10:out=MEM;
        2'b01:out=WB;
        2'b11:out=REG;

endcase

end
endmodule
```

在 **ID** 發現另一個 **hazard**，所以改成 **4*1**

Adder.v

```
module Adder(
        input [31:0]data1,
        input [31:0]data2,
        output [31:0]data_o
        );

assign data_o=data1+data2;

endmodule
```

ALU.v

```
module ALU(
        input[31:0] Source1,
        input[31:0] Source2,
        input[5:0]operation,
        input[4:0]shamt,
        output reg[31:0]result,
        output zero,
        output reg carry
        );

assign zero=(result==0)?1:0;

always @(Source1,Source2,operation,shamt)begin

case(operation)
        6'd27:begin
                {carry,result}=Source1+Source2;
        end
        6'd28:begin
                {carry,result}=Source1-Source2;
        end
        6'd29:begin
                result=Source1>>shamt;
        end
        6'd30:begin
                result=Source1<<shamt;
        end
        6'd31:begin
                result=Source1^Source2;
        end
        6'd32:begin
                result=Source1&Source2;
        end

endcase
end


endmodule
```

EX_MEM

```
module EX2MEM_Register(
        input clk,zero,
        input [1:0]WB, //{RegWrite,MemToReg}
        input [2:0]M,  //{Branch,MemRead,MemWrite}
        input [4:0]RD,
        input [31:0]ALU_result,WriteData,
        output reg zero_o,
        output reg [1:0]WB_o,
        output reg [2:0]M_o,
        output reg [4:0]RD_o,
        output reg [31:0]ALU_result_o,WriteData_o);

initial begin

WB_o=0;
M_o=0;
ALU_result_o=0;
WriteData_o=0;
zero_o=0;
RD_o=0;

end

always @(posedge clk)begin

WB_o<=WB;
M_o<=M;
ALU_result_o<=ALU_result;
WriteData_o<=WriteData;
zero_o<=zero;
RD_o<=RD;

end
endmodule
```

DM.v

```
module DM(
        input clk,
        input [31:0] Address,
        input [31:0] data,
        input MemRead,
        input MemWrite,
        output reg [31:0] DM_data
        );

reg [7:0]Mem[127:0];
integer i;

initial begin
        for(i=0;i<128;i=i+1)begin
                Mem[i]=8'd0;
        end
end

always @(negedge clk)begin
if(MemWrite){Mem[Address],Mem[Address+1],Mem[Address+2],Mem[Address+3]}=data;
if(MemRead)DM_data={Mem[Address],Mem[Address+1],Mem[Address+2],Mem[Address+3]};

end

endmodule
```

MEM_WB

```
module MEM2WB_Register(
        input clk,
        input [1:0]WB, //{RegWrite,MemToReg}
        input [4:0]RD,
        input [31:0]MemData,ALU_result,
        output reg [1:0]WB_o,
        output reg [4:0]RD_o,
        output reg [31:0]MemData_o,ALU_result_o);

initial begin
WB_o=0;
MemData_o=0;
ALU_result_o=0;
RD_o=0;
end

always @(posedge clk)begin

WB_o<=WB;
MemData_o<=MemData;
ALU_result_o<=ALU_result;
RD_o<=RD;

end
endmodule
```

Forwarding.v

```
module Forwarding_unit(
        input wr_MEM,wr_WB,wr_REG,
        input [4:0]RS_ID,RT_ID,RD_MEM,RD_WB,RD_REG,
        output reg [1:0] ForwardA,ForwardB);

initial begin
ForwardA=2'b00;
ForwardB=2'b00;
end

always @(wr_MEM,wr_WB,RS_ID,RT_ID,RD_MEM,RD_WB)begin

ForwardA=2'b00;
ForwardB=2'b00;
if(RD_WB!=0)begin
        if(wr_REG==1)begin

                if(RS_ID == RD_REG)ForwardA=2'b11;

                if(RT_ID == RD_REG)ForwardB=2'b11;
        end

        if(wr_WB==1)begin

                if(RS_ID == RD_WB)ForwardA=2'b01;

                if(RT_ID == RD_WB)ForwardB=2'b01;
        end

        if(wr_MEM==1)begin
                if(RS_ID == RD_MEM)ForwardA=2'b10;

                if(RT_ID == RD_MEM)ForwardB=2'b10;
        end
end


end
endmodule
```

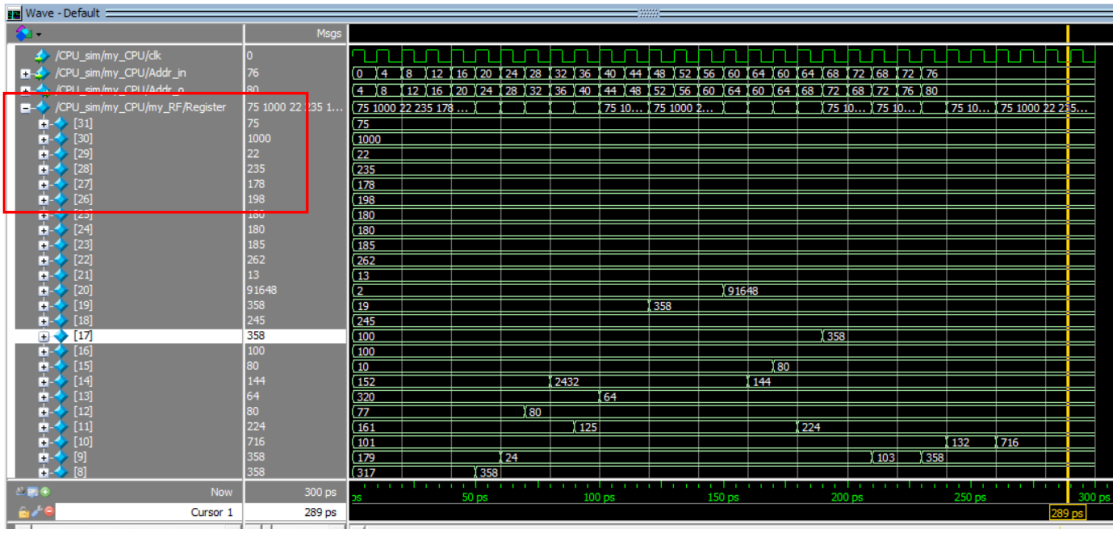
和課本一樣

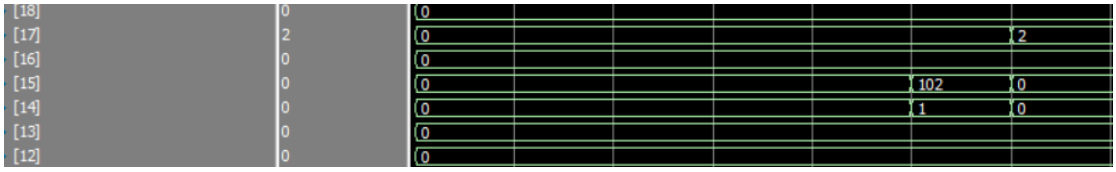
Hazard.v

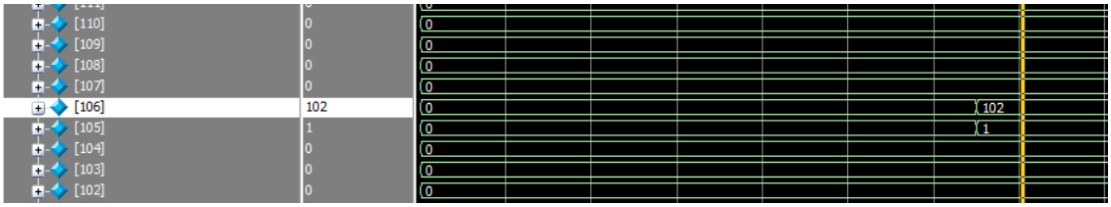
```
module Hazard_Detection(
        input MemRead,
        input [4:0]RS_addr,RT_addr,RT_ID,
        output reg stall);

initial stall=0;


always @(RS_addr,RT_addr,MemRead)begin
stall=0;
if(MemRead==1)begin
        if(RS_addr==RT_ID  || RT_addr==RT_ID)begin
                stall=1;
        end
end

end
endmodule
```

模擬結果

第一題 R-format

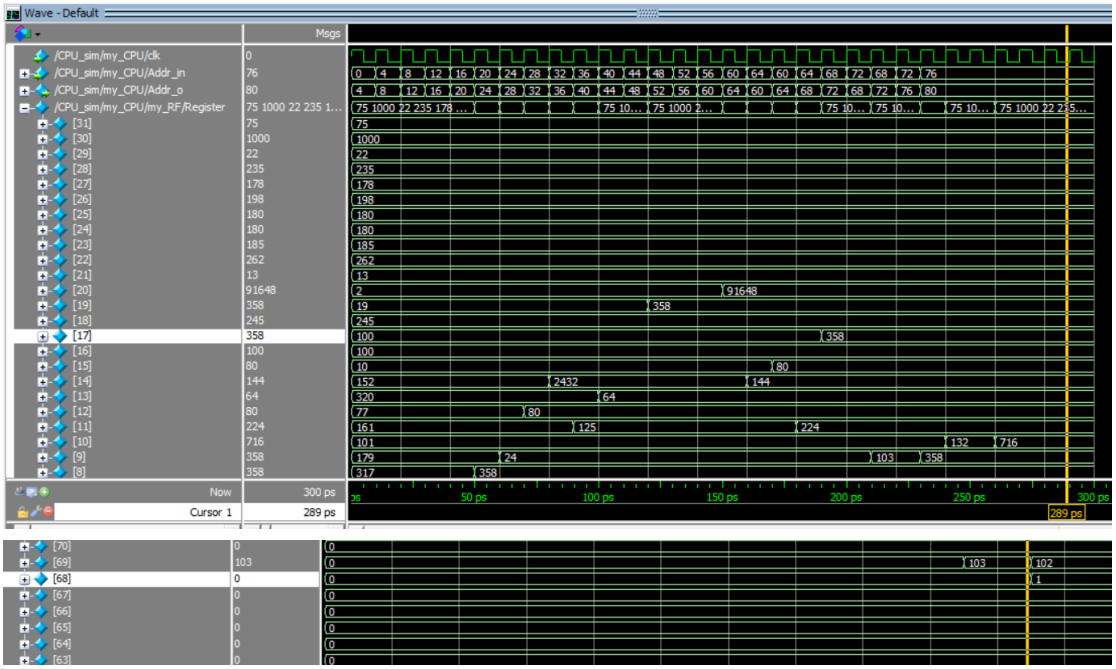






1. add $t0, $t1, $t1
   179+179=358->R[8]
2. sub $t1, $t2, $t4
   101-77=24 ->R[9]
3. srl $t4, $t5, 2
   320>>2=26 ->R[12]
4. sll $t6, $t6, 4
   152<<4=2432 ->R[14]
5. xor $t3, $t1, $t2
   24xor101 =125 ->R[11]
6. and $t5, $t4, $t2
   80and320=64 ->R[13]

1. sw $t0, 2($t7)
   R[8]->M[12] = 358->0 0 1 102
2. lw $s3, 2($t7)
   M[12]->R[19] = 0 0 1 102 = 358
3. sw $s4, 4($t7)
   R[20]->M[14] = 2->0 0 0 2
4. sw $t0, 2($t2)
   R[8]->M[103] =358 -> 0 0 1 102
5. lw $s4, 3($t2)
   M[104]->R[20]



1. add $t6, $t5, $t4

   64+80=144->R[14]
2. sub $t7, $t6, $t5

   144-64=80->R[15]
3. add $t3, $t6, $t7

   144+80=224->R[11]
4. lw $s1, 2($t2)

   M[103]->R[17]=358
5. sw $s1, 2($t5)

   R[17]->M[66] = 0 0 1 102
6. lw $t1, 2($t5)

   M[66]->R[9] = 358
7. add $t2, $t1, $t1

   358+358=716 ->R[10]

個人心得

這次的 **project** 卡了很久，真的寫到心很累，最後面 **hazard detection** 好像有些時序問題但我沒辦法解決，至少算出的值是對的就好了。覺得時序問題真的很麻煩，沒辦法找到確切的問題所在，要查很久。