計算機組織
Project2

四電機三乙

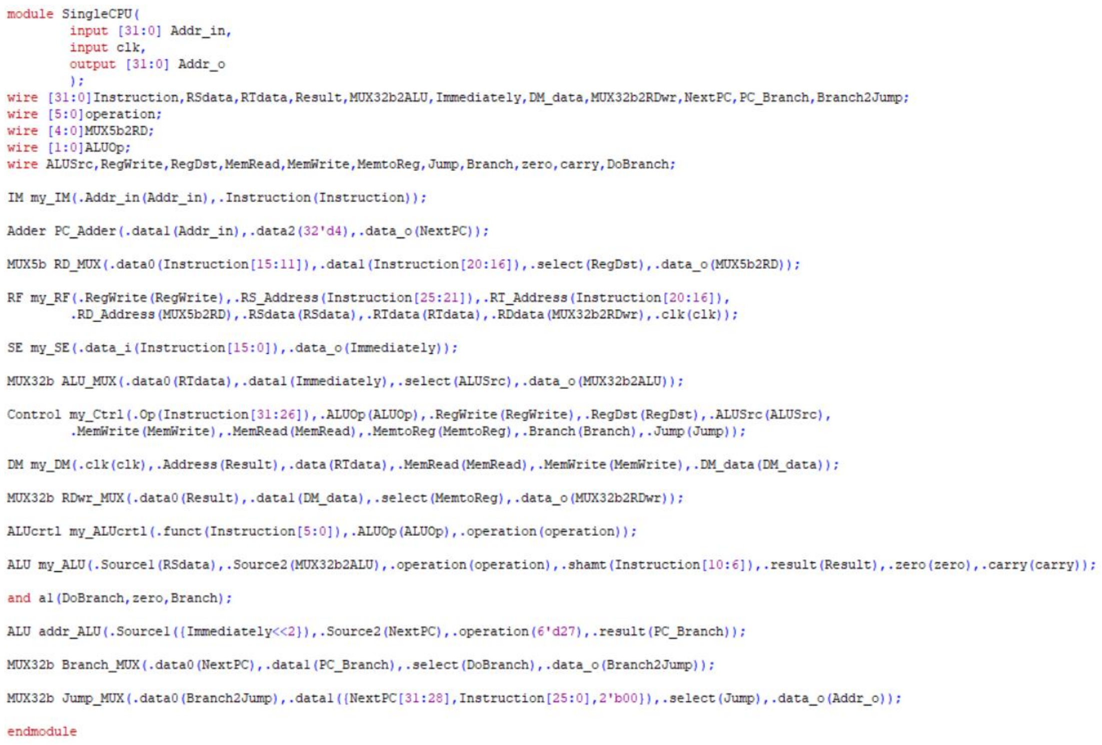
劉錕笙

B10607118

目錄

各 **Module** 程式截圖與說明:

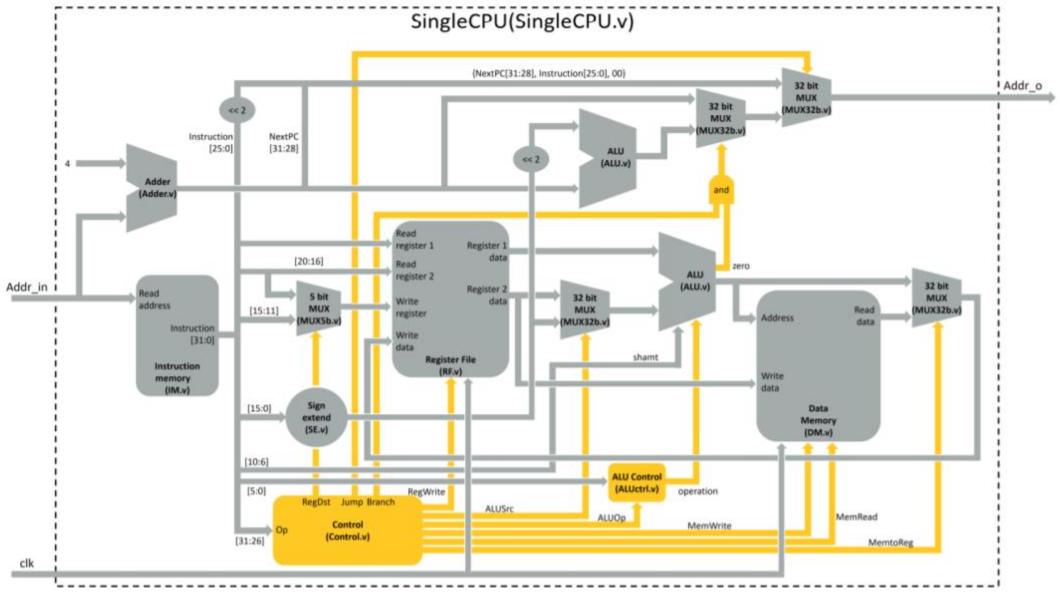
1. SingleCPU.v



```verilog
module SingleCPU(
        input [31:0] Addr_in,
        input clk,
        output [31:0] Addr_o
        );
wire [31:0]Instruction,RSdata,RTdata,Result,MUX32b2ALU,Immediately,DM_data,MUX32b2RDwr,NextPC,PC_Branch,Branch2Jump;
wire [5:0]operation;
wire [4:0]MUX5b2RD;
wire [1:0]ALUOp;
wire ALUSrc,RegWrite,RegDst,MemRead,MemWrite,MemtoReg,Jump,Branch,zero,carry,DoBranch;

IM my_IM(.Addr_in(Addr_in),.Instruction(Instruction));

Adder PC_Adder(.data1(Addr_in),.data2(32'd4),.data_o(NextPC));

MUX5b RD_MUX(.data0(Instruction[15:11]),.data1(Instruction[20:16]),.select(RegDst),.data_o(MUX5b2RD));

RF my_RF(.RegWrite(RegWrite),.RS_Address(Instruction[25:21]),.RT_Address(Instruction[20:16]),
        .RD_Address(MUX5b2RD),.RSdata(RSdata),.RTdata(RTdata),.RDdata(MUX32b2RDwr),.clk(clk));

SE my_SE(.data_i(Instruction[15:0]),.data_o(Immediately));

MUX32b ALU_MUX(.data0(RTdata),.data1(Immediately),.select(ALUSrc),.data_o(MUX32b2ALU));

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

DM my_DM(.clk(clk),.Address(Result),.data(RTdata),.MemRead(MemRead),.MemWrite(MemWrite),.DM_data(DM_data));

MUX32b RDwr_MUX(.data0(Result),.data1(DM_data),.select(MemtoReg),.data_o(MUX32b2RDwr));

ALUcrtl my_ALUcrtl(.funct(Instruction[5:0]),.ALUOp(ALUOp),.operation(operation));

ALU my_ALU(.Source1(RSdata),.Source2(MUX32b2ALU),.operation(operation),.shamt(Instruction[10:6]),.result(Result),.zero(zero),.carry(carry));

and a1(DoBranch,zero,Branch);

ALU addr_ALU(.Source1({Immediately<<2}),.Source2(NextPC),.operation(6'd27),.result(PC_Branch));

MUX32b Branch_MUX(.data0(NextPC),.data1(PC_Branch),.select(DoBranch),.data_o(Branch2Jump));

MUX32b Jump_MUX(.data0(Branch2Jump),.data1({NextPC[31:28],Instruction[25:0],2'b00}),.select(Jump),.data_o(Addr_o));

endmodule
```

上圖為 R_I_J-format 整體的 CPU 接線，主要實踐下圖，因為按圖施

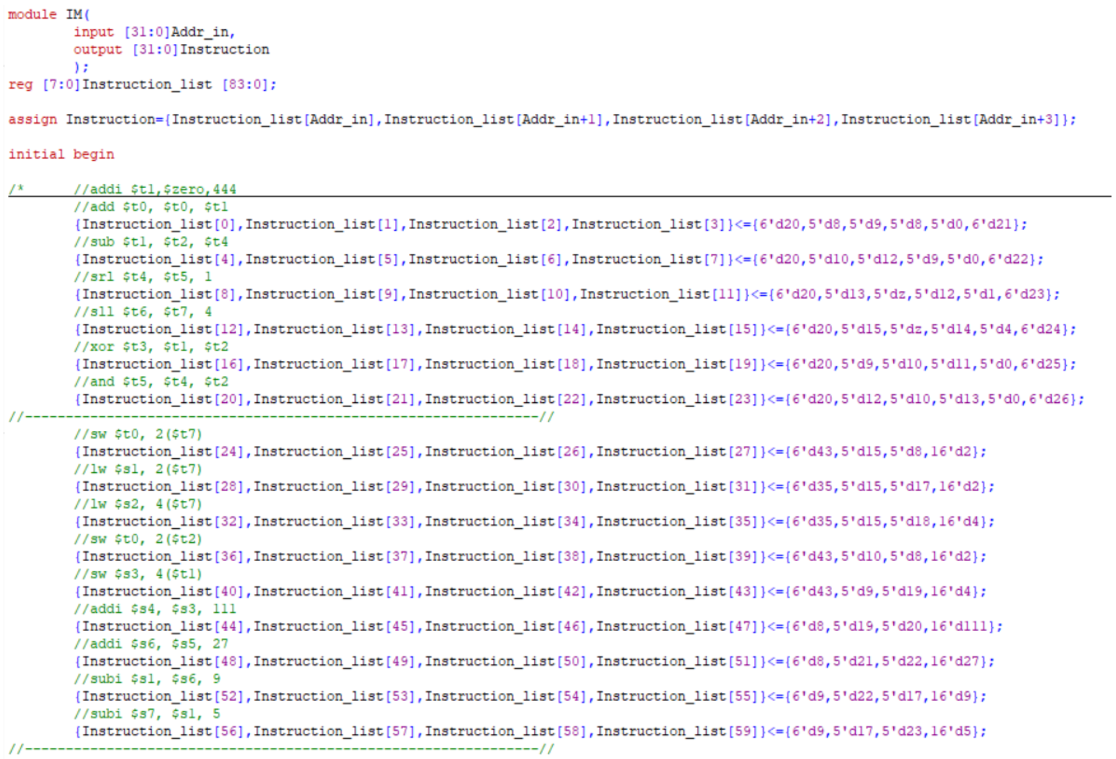
工故不多加贅述。



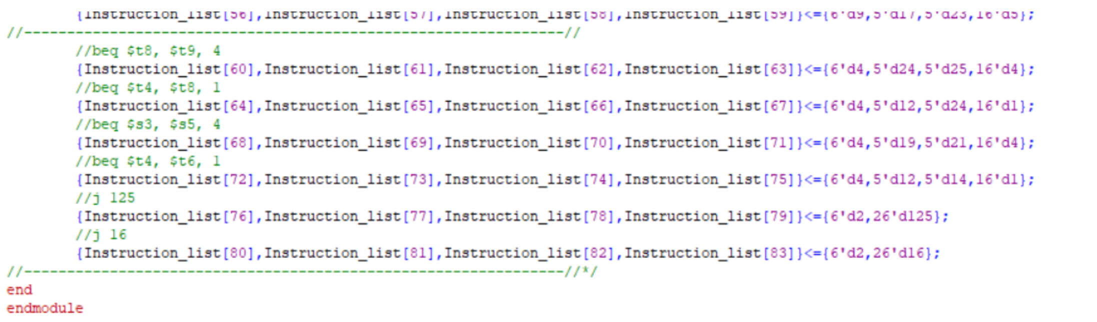
2. IM.v

主要功能指令

這邊是 C 語言的指令，先註解掉



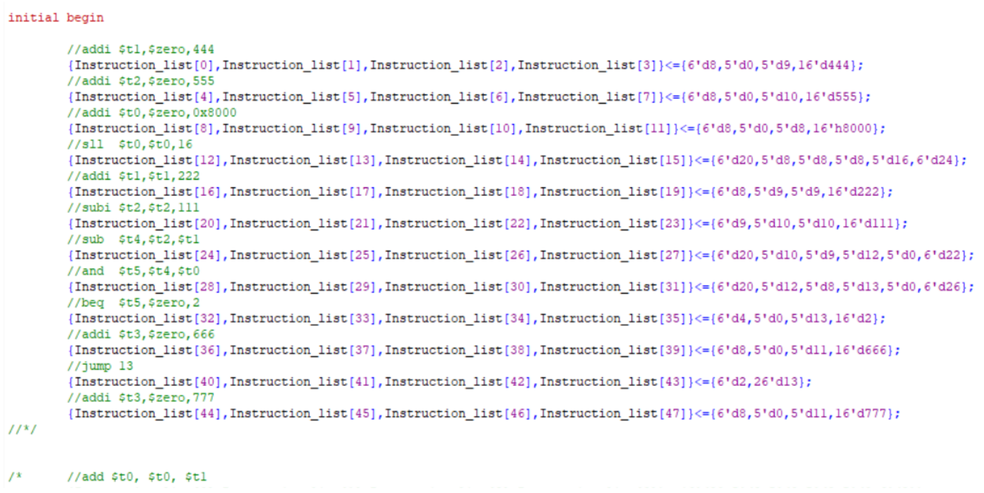
```
module IM(
        input [31:0]Addr_in,
        output [31:0]Instruction
        );
reg [7:0]Instruction_list [83:0];

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

initial begin

/*      //addi $t1,$zero,444
        //add $t0, $t0, $t1
        {Instruction_list[0],Instruction_list[1],Instruction_list[2],Instruction_list[3]}<={6'd20,5'd8,5'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, 1
        {Instruction_list[8],Instruction_list[9],Instruction_list[10],Instruction_list[11]}<={6'd20,5'd13,5'dz,5'd12,5'd1,6'd23};
        //sll $t6, $t7, 4
        {Instruction_list[12],Instruction_list[13],Instruction_list[14],Instruction_list[15]}<={6'd20,5'd15,5'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 $s1, 2($t7)
        {Instruction_list[28],Instruction_list[29],Instruction_list[30],Instruction_list[31]}<={6'd35,5'd15,5'd17,16'd2};
        //lw $s2, 4($t7)
        {Instruction_list[32],Instruction_list[33],Instruction_list[34],Instruction_list[35]}<={6'd35,5'd15,5'd18,16'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};
        //sw $s3, 4($t1)
        {Instruction_list[40],Instruction_list[41],Instruction_list[42],Instruction_list[43]}<={6'd43,5'd9,5'd19,16'd4};
        //addi $s4, $s3, 111
        {Instruction_list[44],Instruction_list[45],Instruction_list[46],Instruction_list[47]}<={6'd8,5'd19,5'd20,16'd111};
        //addi $s6, $s5, 27
        {Instruction_list[48],Instruction_list[49],Instruction_list[50],Instruction_list[51]}<={6'd8,5'd21,5'd22,16'd27};
        //subi $s1, $s6, 9
        {Instruction_list[52],Instruction_list[53],Instruction_list[54],Instruction_list[55]}<={6'd9,5'd22,5'd17,16'd9};
        //subi $s7, $s1, 5
        {Instruction_list[56],Instruction_list[57],Instruction_list[58],Instruction_list[59]}<={6'd9,5'd17,5'd23,16'd5};
//-----------------------------------------------------------------//
```



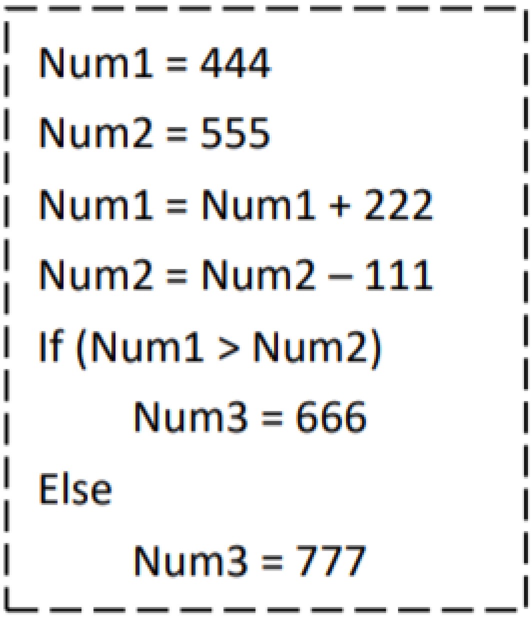
```
        {Instruction_list[56],Instruction_list[57],Instruction_list[58],Instruction_list[59]}<={6'd9,5'd17,5'd23,16'd5};
//-----------------------------------------------------------------//
        //beq $t8, $t9, 4
        {Instruction_list[60],Instruction_list[61],Instruction_list[62],Instruction_list[63]}<={6'd4,5'd24,5'd25,16'd4};
        //beq $t4, $t8, 1
        {Instruction_list[64],Instruction_list[65],Instruction_list[66],Instruction_list[67]}<={6'd4,5'd12,5'd24,16'd1};
        //beq $s3, $s5, 4
        {Instruction_list[68],Instruction_list[69],Instruction_list[70],Instruction_list[71]}<={6'd4,5'd19,5'd21,16'd4};
        //beq $t4, $t6, 1
        {Instruction_list[72],Instruction_list[73],Instruction_list[74],Instruction_list[75]}<={6'd4,5'd12,5'd14,16'd1};
        //j 125
        {Instruction_list[76],Instruction_list[77],Instruction_list[78],Instruction_list[79]}<={6'd2,26'd125};
        //j 16
        {Instruction_list[80],Instruction_list[81],Instruction_list[82],Instruction_list[83]}<={6'd2,26'd16};
//-----------------------------------------------------------------//*/
end
endmodule
```

上圖將題目要求的指令轉換成 Instruction

C 語言功能指令



```
initial begin

        //addi $t1,$zero,444
        {Instruction_list[0],Instruction_list[1],Instruction_list[2],Instruction_list[3]}<={6'd8,5'd0,5'd9,16'd444};
        //addi $t2,$zero,555
        {Instruction_list[4],Instruction_list[5],Instruction_list[6],Instruction_list[7]}<={6'd8,5'd0,5'd10,16'd555};
        //addi $t0,$zero,0x8000
        {Instruction_list[8],Instruction_list[9],Instruction_list[10],Instruction_list[11]}<={6'd8,5'd0,5'd8,16'h8000};
        //sll  $t0,$t0,16
        {Instruction_list[12],Instruction_list[13],Instruction_list[14],Instruction_list[15]}<={6'd20,5'd8,5'd8,5'd8,5'd16,6'd24};
        //addi $t1,$t1,222
        {Instruction_list[16],Instruction_list[17],Instruction_list[18],Instruction_list[19]}<={6'd8,5'd9,5'd9,16'd222};
        //subi $t2,$t2,111
        {Instruction_list[20],Instruction_list[21],Instruction_list[22],Instruction_list[23]}<={6'd9,5'd10,5'd10,16'd111};
        //sub  $t4,$t2,$t1
        {Instruction_list[24],Instruction_list[25],Instruction_list[26],Instruction_list[27]}<={6'd20,5'd10,5'd9,5'd12,5'd0,6'd22};
        //and  $t5,$t4,$t0
        {Instruction_list[28],Instruction_list[29],Instruction_list[30],Instruction_list[31]}<={6'd20,5'd12,5'd8,5'd13,5'd0,6'd26};
        //beq  $t5,$zero,2
        {Instruction_list[32],Instruction_list[33],Instruction_list[34],Instruction_list[35]}<={6'd4,5'd0,5'd13,16'd2};
        //addi $t3,$zero,666
        {Instruction_list[36],Instruction_list[37],Instruction_list[38],Instruction_list[39]}<={6'd8,5'd0,5'd11,16'd666};
        //jump 13
        {Instruction_list[40],Instruction_list[41],Instruction_list[42],Instruction_list[43]}<={6'd2,26'd13};
        //addi $t3,$zero,777
        {Instruction_list[44],Instruction_list[45],Instruction_list[46],Instruction_list[47]}<={6'd8,5'd0,5'd11,16'd777};
//*/


/*      //add $t0, $t0, $t1
```



```
Num1 = 444
Num2 = 555
Num1 = Num1 + 222
Num2 = Num2 – 111
If (Num1 > Num2)
        Num3 = 666
Else
        Num3 = 777
```

1. 先將 444、555 寫進記憶體內，我使用的是 I-format

   addi,$t1,$zero,444

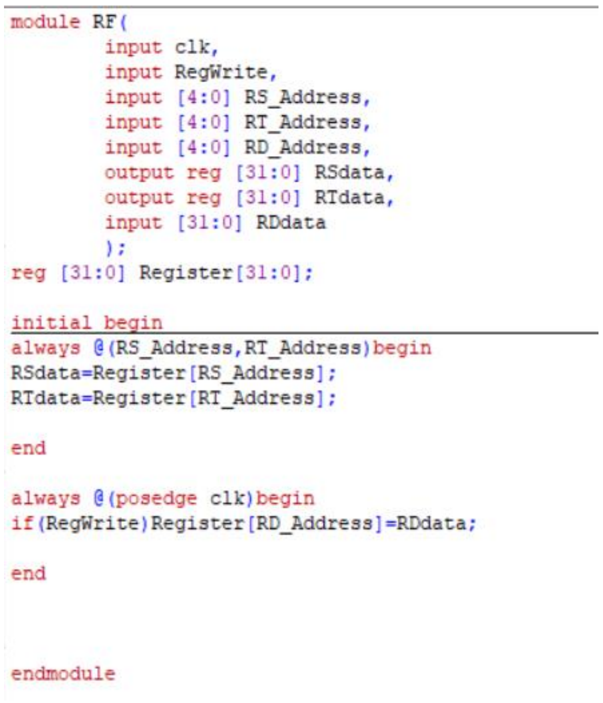
2. 對 Num1、Num2 進行加減運算，I-format

   Addi $t1,$t1,222

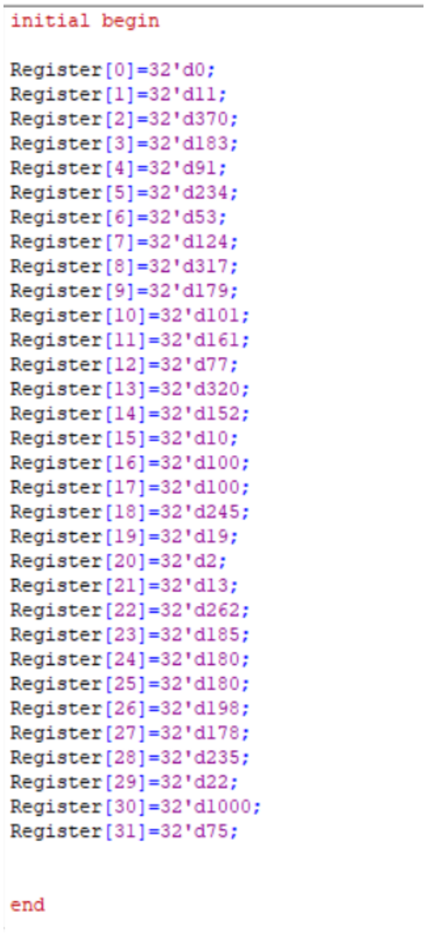
3. 要進行判斷，由於 Instruction 內只有 beq 指令，因此我先做了轉換，先將 Num2-Num1，再判斷結果的正負號。判斷正負號的方式是將結果 AND 一個 0x80000000 的值，再用 beq 判斷結果結果是否為零，若為零則表示 Num2>=Num1，則跳至 else

```
addi $t0,$zero,0x8000
sll    $t0,$t0,16
sub    $t4,$t2,$t1
and    $t5,$t4,$t0
beq    $t5,$zero,2
addi $t3,$zero,666
jump 13
addi $t3,$zero,777
```

3. RF.v



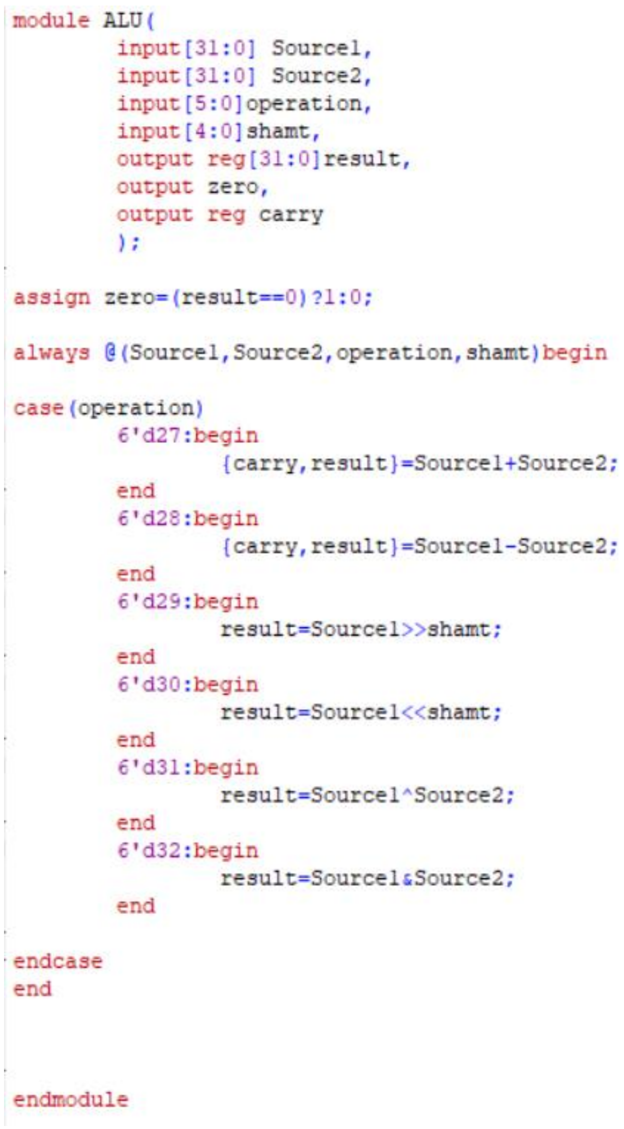
```
module RF(
        input clk,
        input RegWrite,
        input [4:0] RS_Address,
        input [4:0] RT_Address,
        input [4:0] RD_Address,
        output reg [31:0] RSdata,
        output reg [31:0] RTdata,
        input [31:0] RDdata
        );
reg [31:0] Register[31:0];

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
```

將值初始化，若要寫入記憶體，則在進入下一個 CLK 前將這一個 CLK 的值儲存起來。因為 posedge clk 動作時使用的是前一個狀態的 RegWrite，因此可以當作在此 clk 結束的瞬間，對記憶體寫入，也可以確保資料已經計算完成，也可以提供讀取 memery 的裕度。

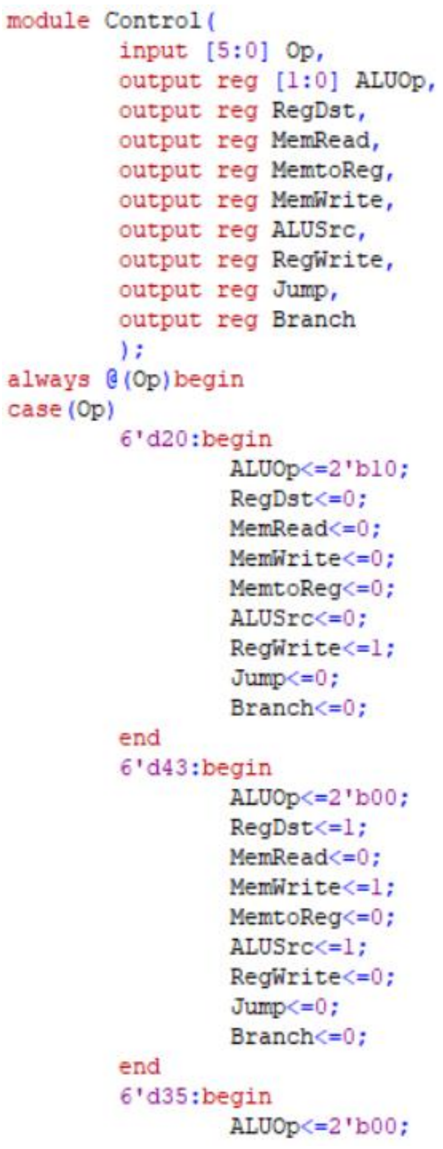
4. ALU.v



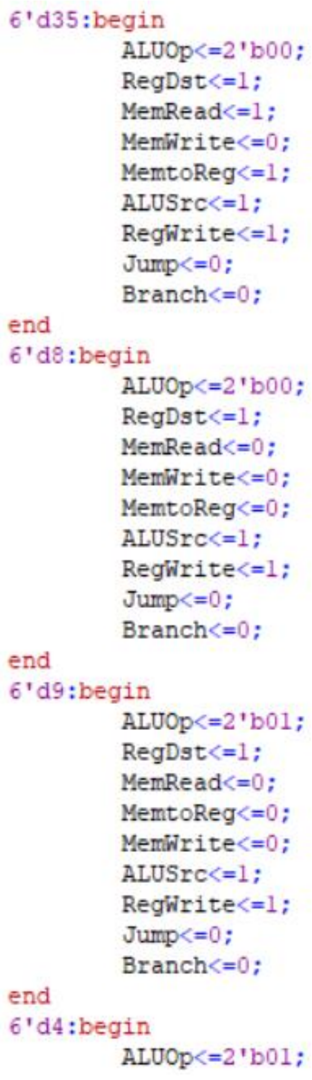
```
module ALU(
        input[31:0] Source1,
        input[31:0] Source2,
        input[5:0]operation,
        input[4:0]shamt,
        output reg[31:0]result,
        output zero,
        output reg carry
        );

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
```

將 ALU_control 的 OP 資料寫入，並進行運算。

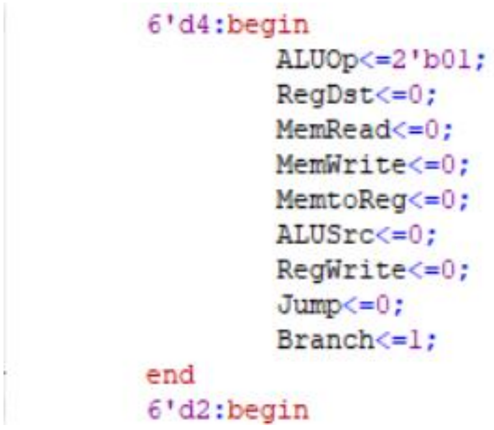
5. Control.v



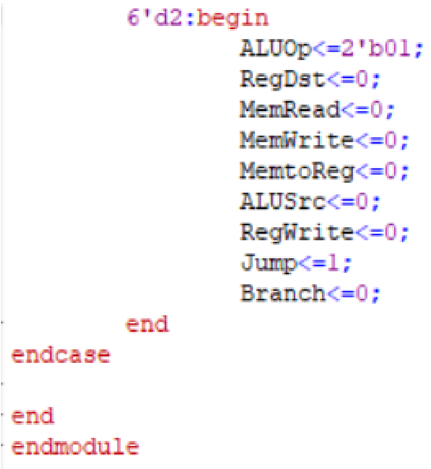
```
module Control(
        input [5:0] Op,
        output reg [1:0] ALUOp,
        output reg RegDst,
        output reg MemRead,
        output reg MemtoReg,
        output reg MemWrite,
        output reg ALUSrc,
        output reg RegWrite,
        output reg Jump,
        output reg Branch
        );
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;
```



```
        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;
```



```
        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
```



```
        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
```

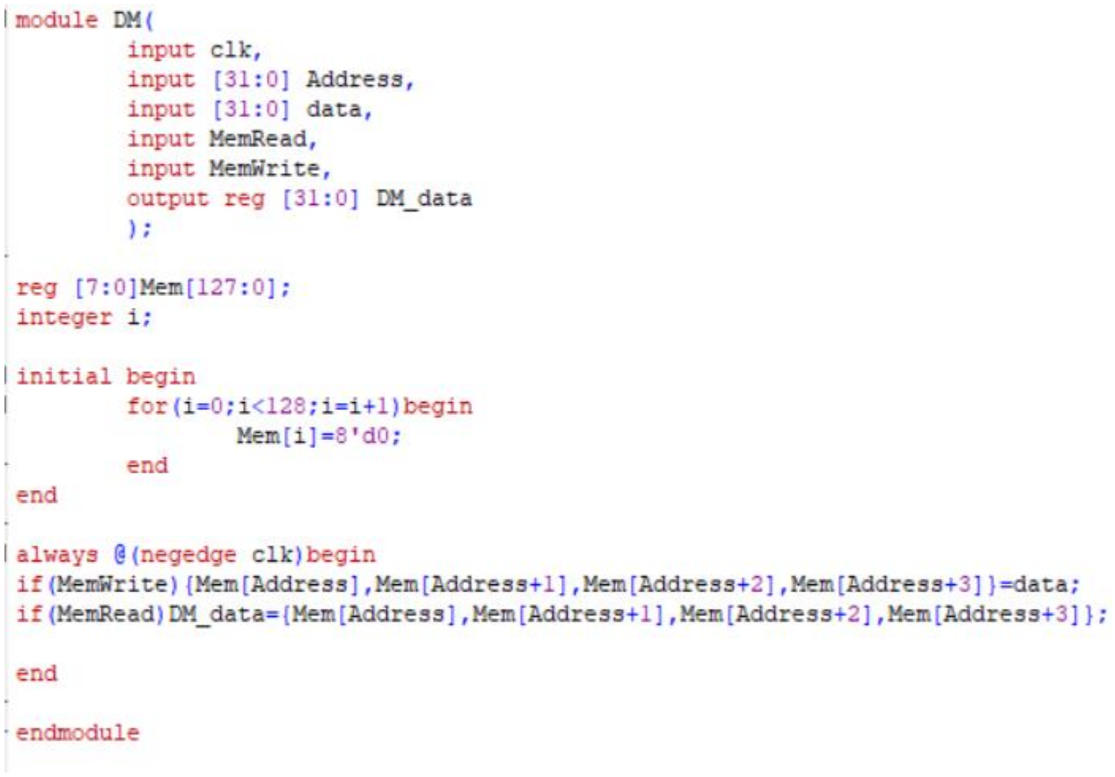
Control path 的控制。詳細如下表

	R-type	**SW**	**LW**	**Addi**	**Subi**	**Beq**	**Jump**
RegDst	0	1	1	1	1	0	0
MemRead	0	0	1	0	0	0	0
MemWrite	0	1	0	0	0	0	0
Mem2Reg	0	0	1	0	0	0	0
ALUsrc	0	1	1	1	1	0	0
RegWrite	1	0	1	1	1	0	0
OP	10	00	00	00	01	01	01
Branch	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1

這邊對 OP 的訊號線進行調整，將原[2:0]改為[1:0]，因為

ALUOp[2]與控制訊號 Branch 相同，故只留下 branch

6. DM.v



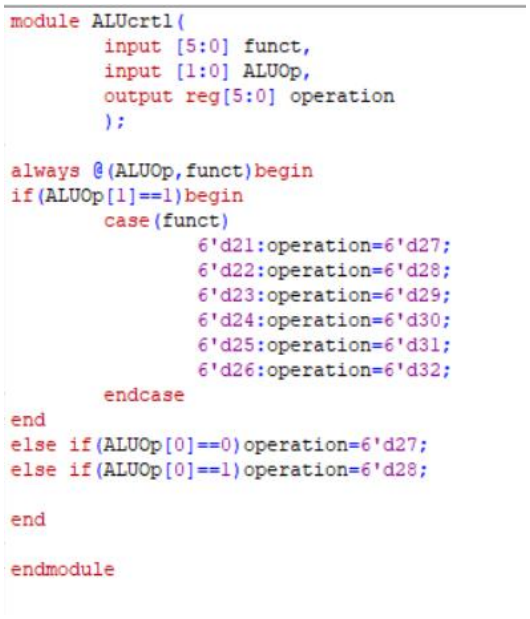
```
module DM(
        input clk,
        input [31:0] Address,
        input [31:0] data,
        input MemRead,
        input MemWrite,
        output reg [31:0] DM_data
        );

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
```

Initial 將記憶體空間清除歸零。

資料在 negedge 輸出與寫入，配合 RF 在下一個 clk posedge 寫入，使資料不會出現差錯。

資料讀取與寫入依照題目要求，單位 byte，但因為沒有設計 lb 指令，因此在寫入/讀取時直接設計為 word 讀取。並且以 MSB 編碼。

7. ALUctrl.v

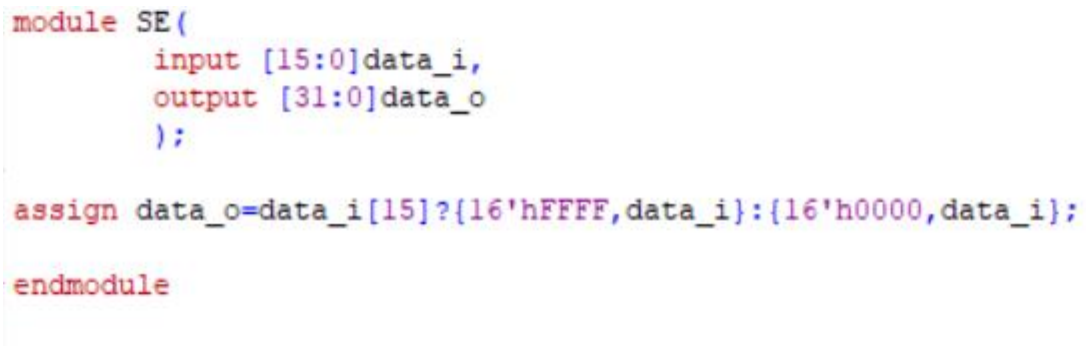


```
module ALUcrtl(
        input [5:0] funct,
        input [1:0] ALUOp,
        output reg[5:0] operation
        );

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
```

針對 OP 指令進行轉換，並在指向 R-format 時將 funct 轉碼輸出。

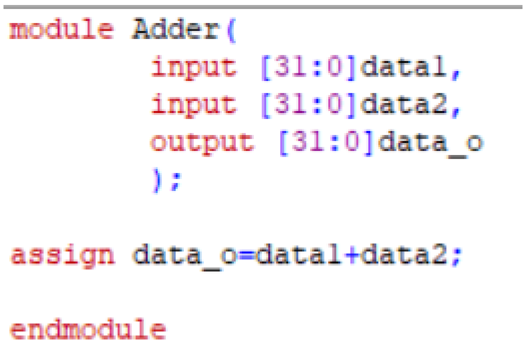
這邊我有對 OP 的格式做調整，因為原本 2:0 共 3bits 的資料只會使用到兩條，而 ALUOp[2]=branch 又會在 control path 獨立出一個控制訊號，對 ALUctrl 沒有影響，所以改成 ALUOp[1:0]兩條

8. SE.v



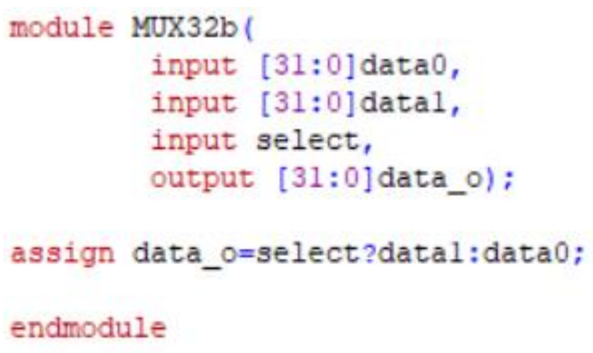
```
module SE(
        input [15:0]data_i,
        output [31:0]data_o
        );

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

endmodule
```

9. Adder.v



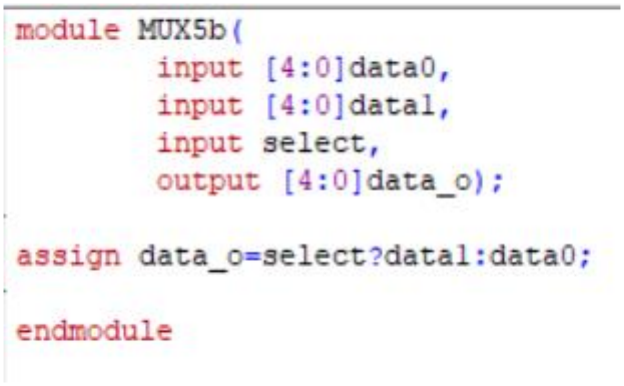
```
module Adder(
        input [31:0]data1,
        input [31:0]data2,
        output [31:0]data_o
        );

assign data_o=data1+data2;

endmodule
```

10.MUX32b.v



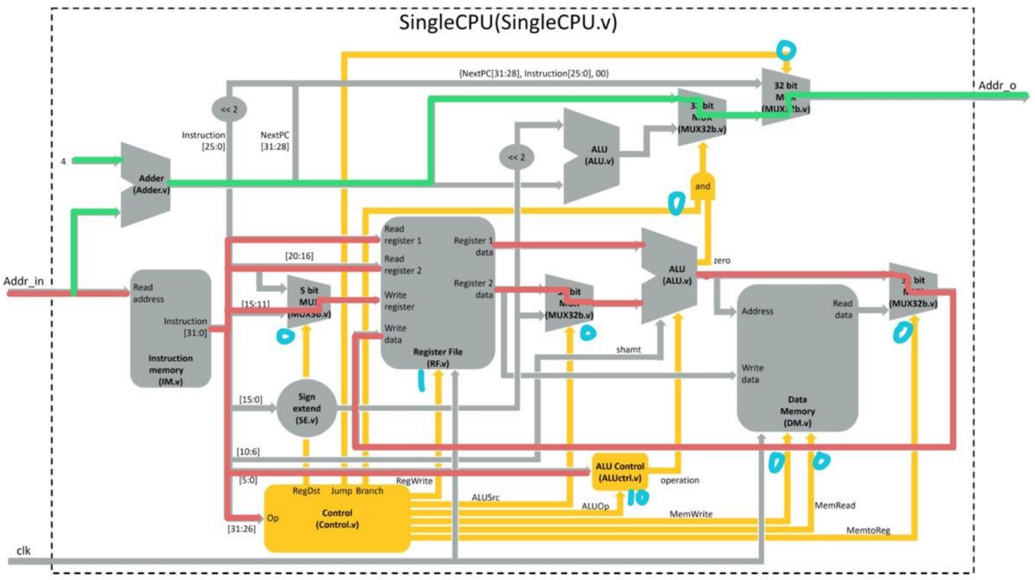
```
module MUX32b(
        input [31:0]data0,
        input [31:0]data1,
        input select,
        output [31:0]data_o);

assign data_o=select?data1:data0;

endmodule
```

11.MUX5b.v

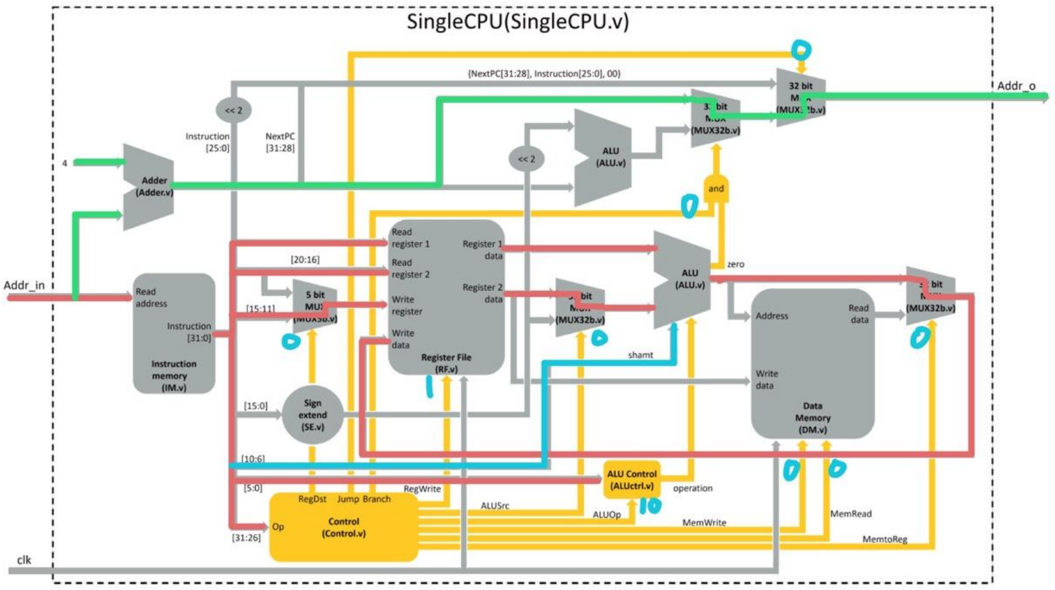


```
module MUX5b(
        input [4:0]data0,
        input [4:0]data1,
        input select,
        output [4:0]data_o);

assign data_o=select?data1:data0;

endmodule
```

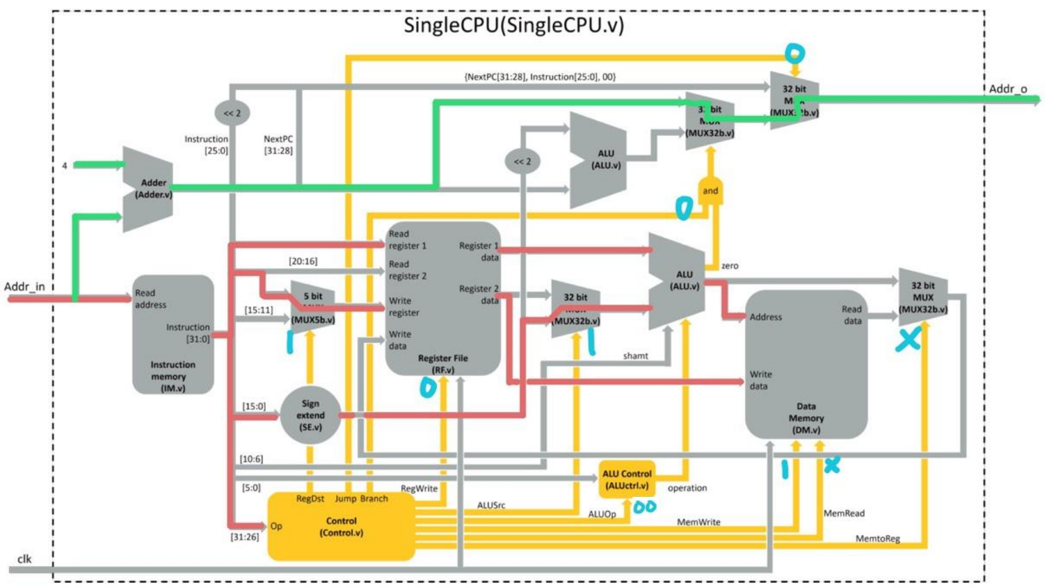
Data/Control Path



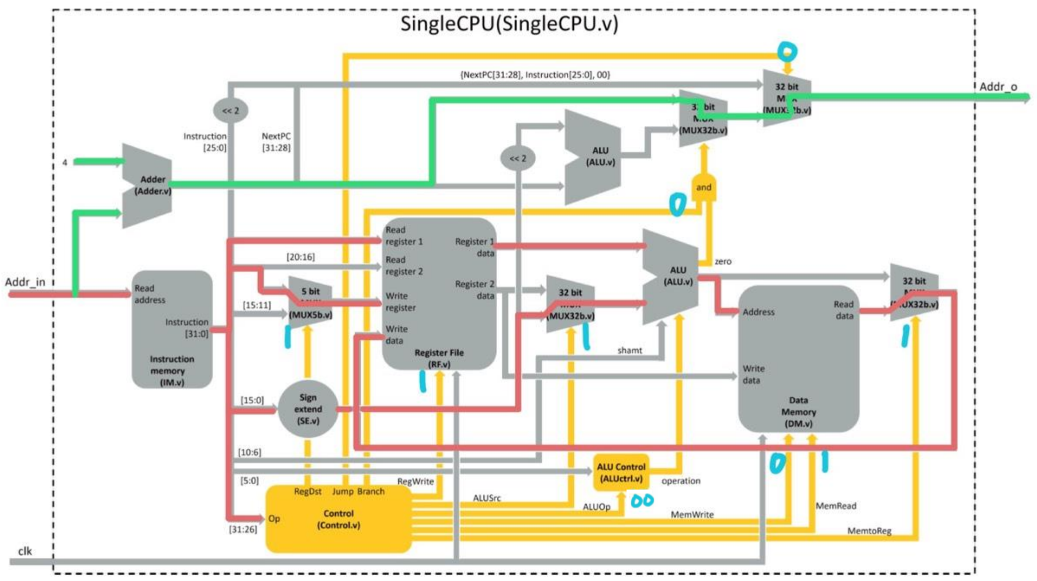
R_format，(add,sub,and,xor)



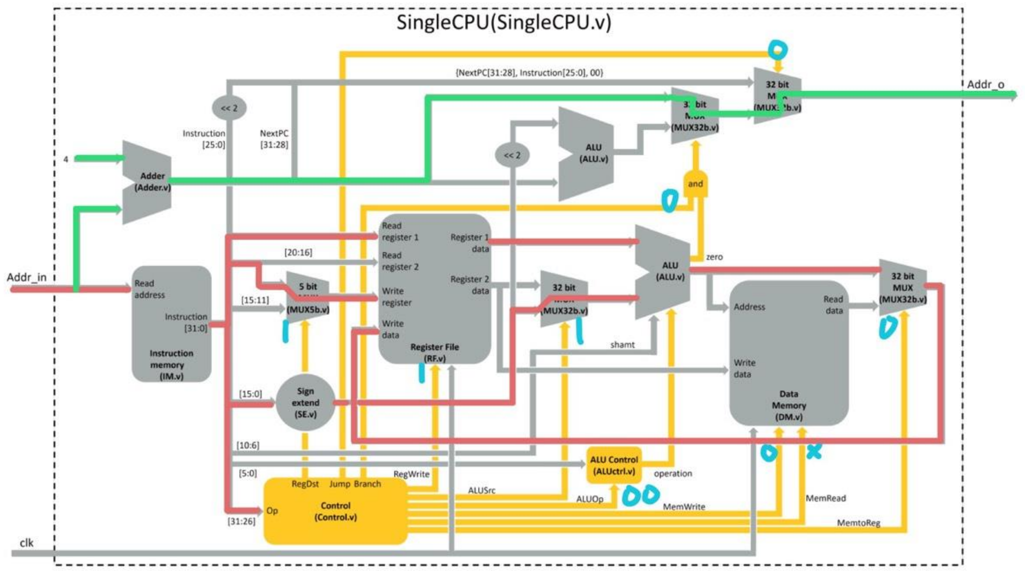
R_format，(sll,srl)



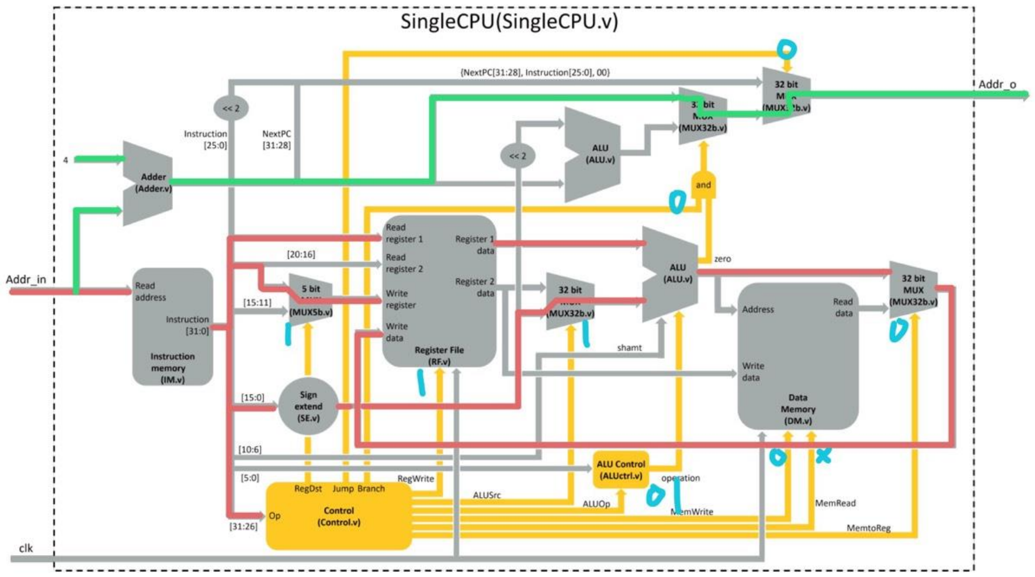
I-format，sw



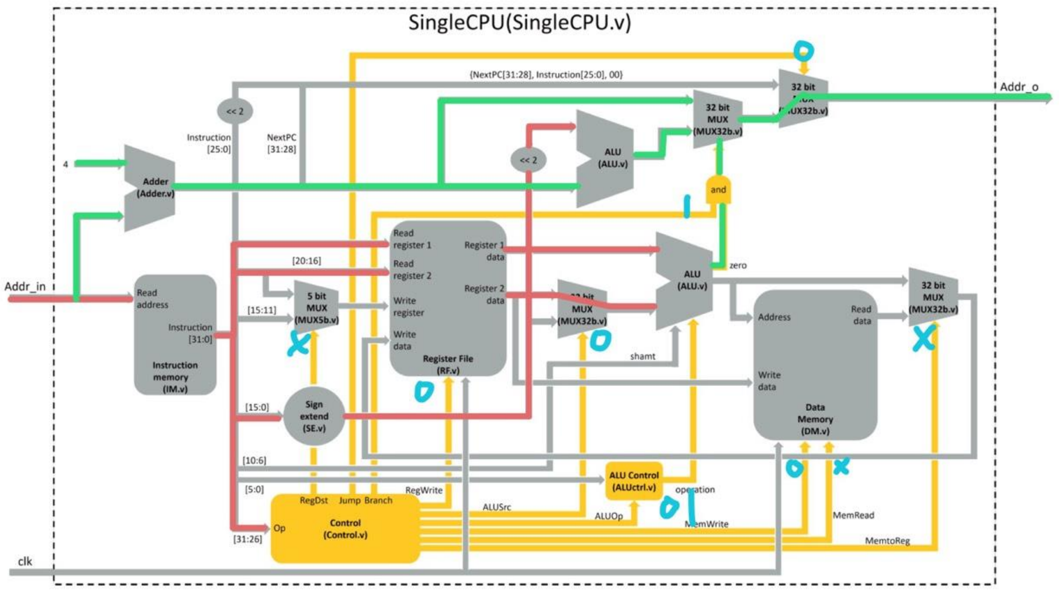
I-format，lw



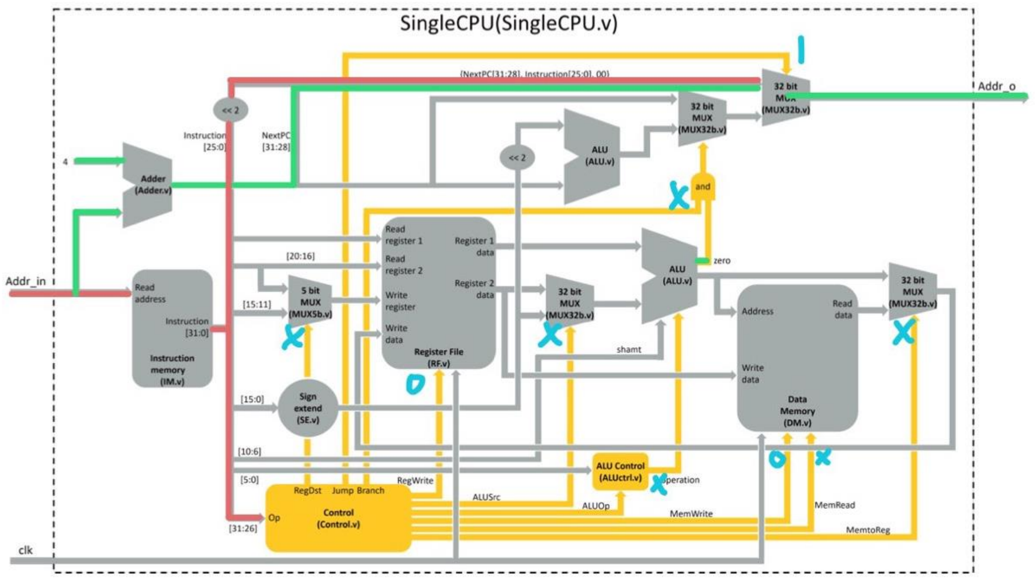
I-format，addi



I-format，subi



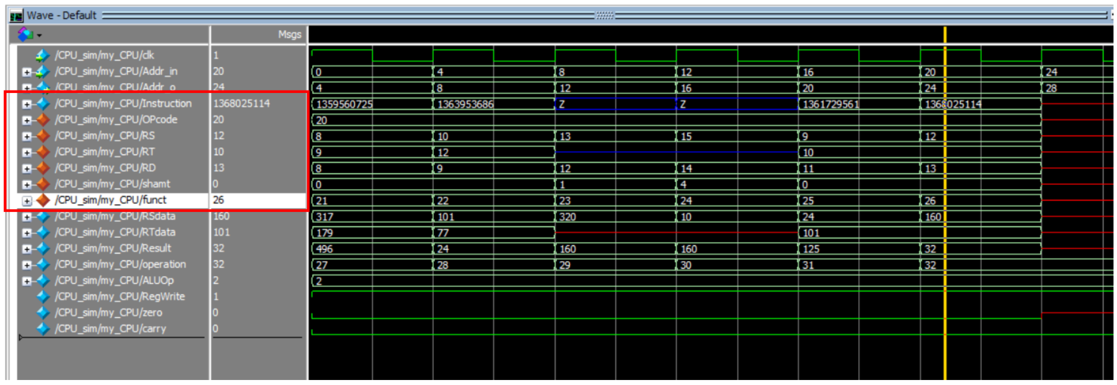
I-format，Branch



J-format，Jump

模擬結果

第一題 R-format


Wave - Default
Msgs
/CPU_sim/my_CPU/clk 1
/CPU_sim/my_CPU/Addr_in 20
/CPU_sim/my_CPU/Addr_o 24
/CPU_sim/my_CPU/Instruction 1368025114
/CPU_sim/my_CPU/OPcode 20
/CPU_sim/my_CPU/RS 12
/CPU_sim/my_CPU/RT 10
/CPU_sim/my_CPU/RD 13
/CPU_sim/my_CPU/shamt 0
/CPU_sim/my_CPU/funct 26
/CPU_sim/my_CPU/RSdata 160
/CPU_sim/my_CPU/RTdata 101
/CPU_sim/my_CPU/Result 32
/CPU_sim/my_CPU/operation 32
/CPU_sim/my_CPU/ALUOp 2
/CPU_sim/my_CPU/RegWrite 1
/CPU_sim/my_CPU/zero 0
/CPU_sim/my_CPU/carry 0

1.R-format，取暫存器 8、9 的值 317、179，相加=496，存到暫存器 8

2.R-format，取暫存器 10、12 的值 101、77，相減=24，存到暫存器 9

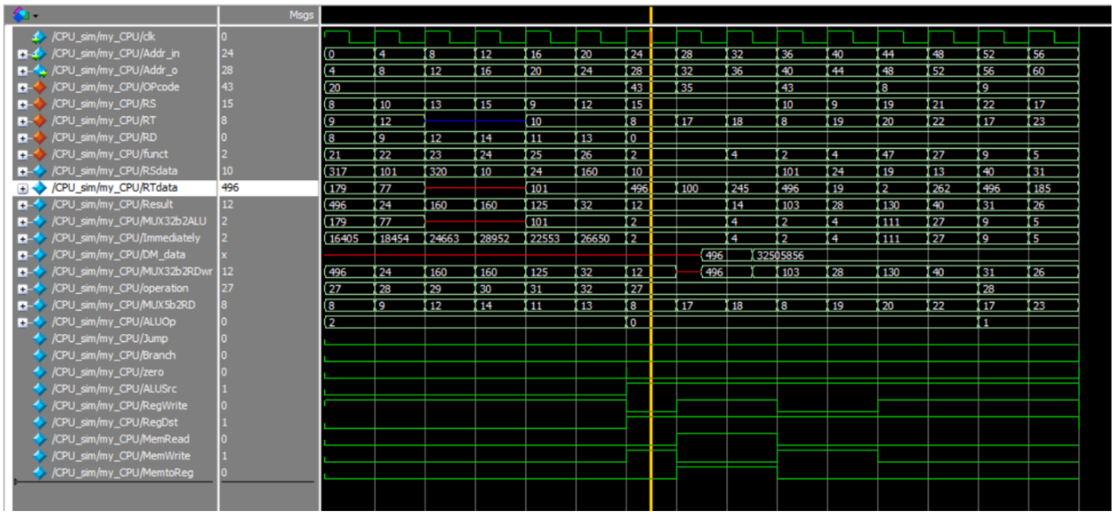
3.R-format，取暫存器 13 的值 320，右移 1，320/2=160，存到暫存器 12

4.R-format，取暫存器 15 的值 10，右移 4，10*16=160，存到暫存器 14

5.R-format，取暫存器 9、10 的值 24、101，XOR=125，存到暫存器 11

6.R-format，取暫存器 12、10 的值 160、101，AND=32，存到暫存器 13

第二題 I-format


Msgs
/CPU_sim/my_CPU/clk 0
/CPU_sim/my_CPU/Addr_in 24
/CPU_sim/my_CPU/Addr_o 28
/CPU_sim/my_CPU/OPcode 43
/CPU_sim/my_CPU/RS 15
/CPU_sim/my_CPU/RT 8
/CPU_sim/my_CPU/RD 0
/CPU_sim/my_CPU/funct 2
/CPU_sim/my_CPU/RSdata 10
/CPU_sim/my_CPU/RTdata 496
/CPU_sim/my_CPU/Result 12
/CPU_sim/my_CPU/MUX32b2ALU 2
/CPU_sim/my_CPU/Immediately 2
/CPU_sim/my_CPU/DM_data x
/CPU_sim/my_CPU/MUX32b2RDwr 12
/CPU_sim/my_CPU/operation 27
/CPU_sim/my_CPU/MUX5b2RD 8
/CPU_sim/my_CPU/ALUOp 0
/CPU_sim/my_CPU/Jump 0
/CPU_sim/my_CPU/Branch 0
/CPU_sim/my_CPU/zero 0
/CPU_sim/my_CPU/ALUSrc 1
/CPU_sim/my_CPU/RegWrite 0
/CPU_sim/my_CPU/RegDst 1
/CPU_sim/my_CPU/MemRead 0
/CPU_sim/my_CPU/MemWrite 1
/CPU_sim/my_CPU/MemtoReg 0

1.I-format，暫存器[8]的值 496 存進暫存器[15]的值 10+2=12 記憶體[12]=496

2.I-format，暫存器[15]的值 10+2=12，取出記憶體[12]的值 496 存進暫存器[17]

3.I-format，暫存器[15]的值 10+4=12，取出記憶體[14]的值 496<<16=32505856 存進暫存器[18]

4.I-format，暫存器[8]的值 496 存進暫存器[10]的值 101 位移 2=103 記憶體[103]=496

5.I-format，暫存器[19]的值 19 存進暫存器[9]的值 24 位移 4=28 記憶體[28]=19

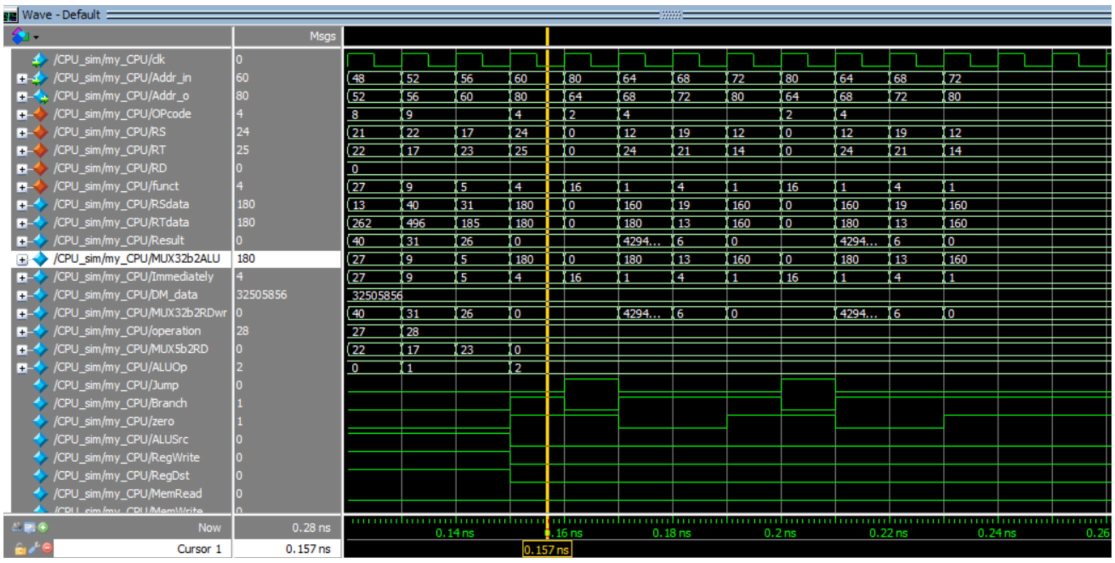
6.I-format，暫存器[19]的值 19+111=130，存到暫存器[20]=130

7.I-format，暫存器[21]的值 13+27=40，存到暫存器[22]=40

8.I-format，暫存器[22]的值 40+9=31，存到暫存器[17]=31

8.I-format，暫存器[17]的值 31+5=26，存到暫存器[23]=26

第三題 J-format



```
Wave - Default
                                    Msgs
/CPU_sim/my_CPU/clk                 0
/CPU_sim/my_CPU/Addr_in             60
/CPU_sim/my_CPU/Addr_o              80
/CPU_sim/my_CPU/OPcode              4
/CPU_sim/my_CPU/RS                  24
/CPU_sim/my_CPU/RT                  25
/CPU_sim/my_CPU/RD                  0
/CPU_sim/my_CPU/funct               4
/CPU_sim/my_CPU/RSdata              180
/CPU_sim/my_CPU/RTdata              180
/CPU_sim/my_CPU/Result              0
/CPU_sim/my_CPU/MUX32b2ALU          180
/CPU_sim/my_CPU/Immediately         4
/CPU_sim/my_CPU/DM_data             32505856
/CPU_sim/my_CPU/MUX32b2RDwr         0
/CPU_sim/my_CPU/operation           28
/CPU_sim/my_CPU/MUX5b2RD            0
/CPU_sim/my_CPU/ALUOp               2
/CPU_sim/my_CPU/Jump                0
/CPU_sim/my_CPU/Branch              1
/CPU_sim/my_CPU/zero                1
/CPU_sim/my_CPU/ALUSrc              0
/CPU_sim/my_CPU/RegWrite            0
/CPU_sim/my_CPU/RegDst              0
/CPU_sim/my_CPU/MemRead             0
/CPU_sim/my_CPU/MemWrite            0
                          Now       0.28 ns
                     Cursor 1       0.157 ns
```

1.I-format，比較暫存器 24、25 的值，180=180，跳轉 4 行指令(4*4=16)，

New_PC=NextPC+4*4=PC+4+16=60+20=80

2.J-format，跳轉至第 16 個 instruction，也就是 PC=16*4=64 位置

3.I-format，比較暫存器 12、24 的值，160!=180，不執行 Branch，

New_PC=NextPC=PC+4=64+4=68

4.I-format，比較暫存器 19、21 的值，19!=13，不執行 Branch，

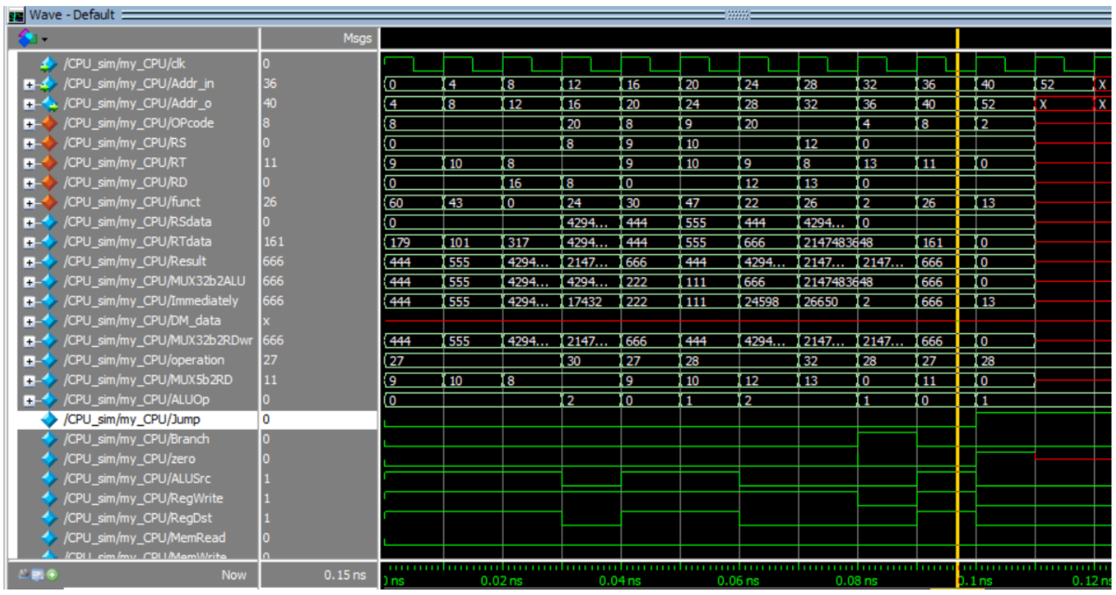
New_PC=NextPC=PC+4=68+4=72

5.I-format，比較暫存器 12、14 的值，160=160，跳轉 1 行指令(1*4=4)，

New_PC=NextPC+1*4=PC+4+4=72+8=80

6.J-format，跳轉至第 16 個 instruction，也就是 PC=16*4=64 位置

以下 2-5 循環...



addi $t1,$zero,0x444	將暫存器[9]設定為 444
addi $t2,$zero,0x555	將暫存器[10]設定為 555
addi $t0,$zero,0x8000	將暫存器[8]設定為 0x8000(H)
sll $t0,$t0,16	將暫存器[8]左移 16bit=0x80000000(H)
addi $t1,$t1,222	暫存器[9]=444+222=666
subi $t2,$t2,111	暫存器[10]=555-111=444
sub $t4,$t2,$t1	將暫存器[12]= 暫存器[10]- 暫存器[9] 這邊藉由判斷相減後的正負號，讓 beq 判斷從 A=B 轉換成 A>B。
and $t5,$t4,$t0	留下結果的正負號，並存入暫存器[13]
beq $t5,$zero,2	如果 B>=A，則原條件不成立，跳至值行 num3=777;
addi $t3,$zero,666	如果 B-A 為負號，則上式 branch 不成立，則暫存器[11]=666
jump 13	執行完跳至 num=777 之後繼續執行，由於共有 12 條 Instruction，故跳至第 PC=13*4=52 位置
addi $t3,$zero,777	暫存器[11]=777

個人心得

我覺得這些題目都沒有很難，只要把課本上教過的電路實踐出來就好了。比較麻煩的就是要如何在一個 **clk** 兩個 **edge** 中處理三份資料。打報告的時間比打程式還要久**…..**。我覺得最後 **C** 轉組語蠻有趣的，想了一下如何把 **beq** 延伸到 **if else** 的判斷式裡面。