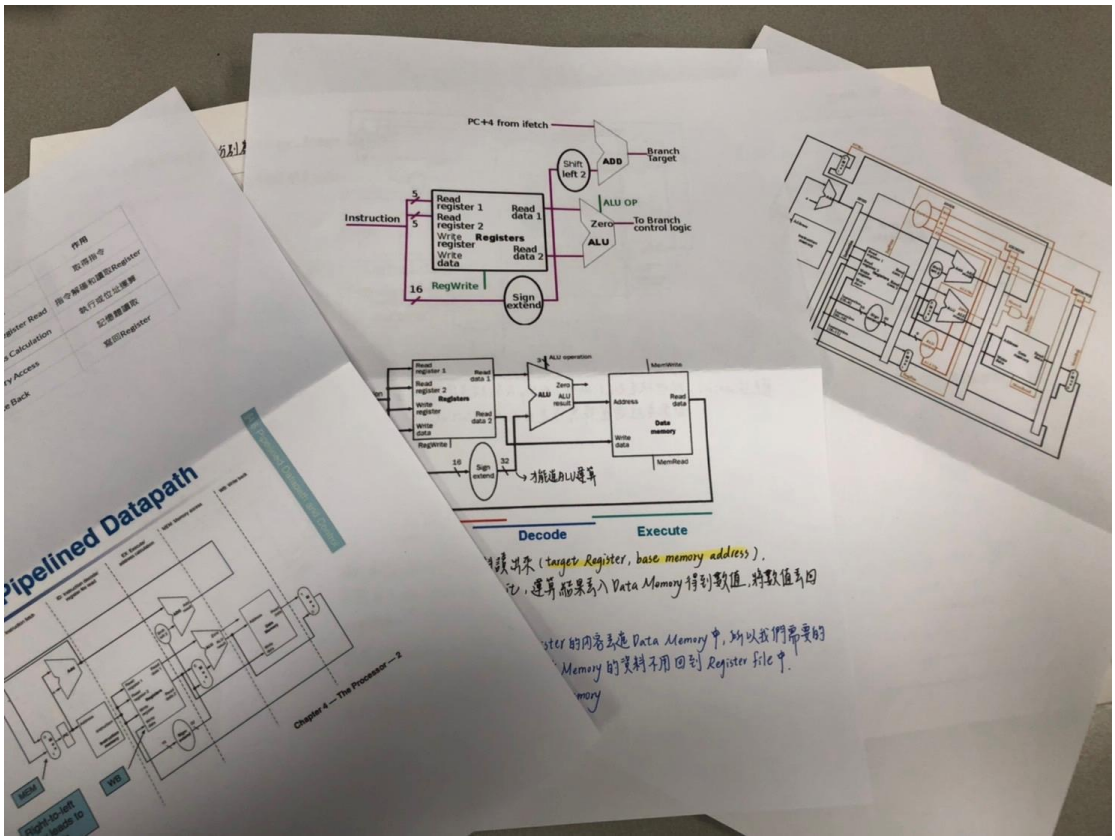
# FPGA 系統設計實務

期中報告

M11102150

劉家瑜

# Design



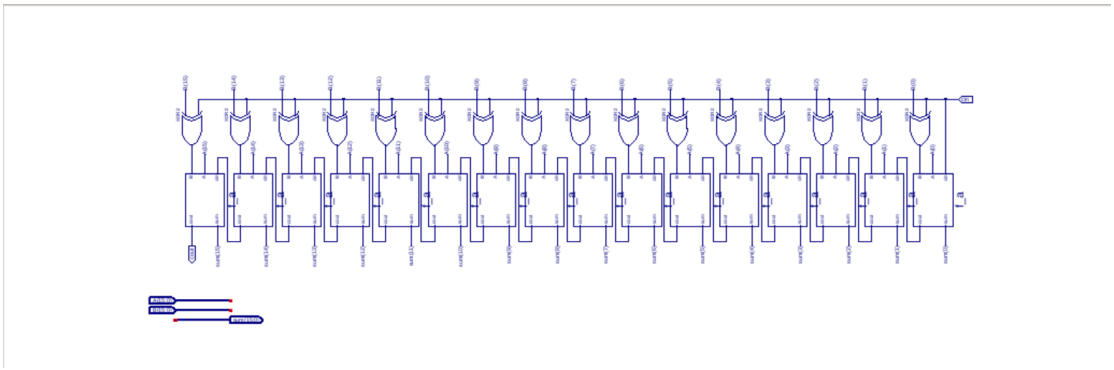

| | | regEn | regSel | regData | reg 外 | readA | readB | outB_in | outImm | LHI | add/sub | logicm | memread | memEn | memAddr | memData | done | pcen | pcext | pc inc bra |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00001 | LHI | 1 | instr[10:8] | ALUout | 0 | instr[10:8] | | instr[7:0] | 1 | 1 | 0 | 1 | 0 | 0 | x | x | 0 | | 1 | 0 |
| 00010 | LLI | 1 | instr[10:8] | ALUout | 0 | x | | instr[7:0] | 1 | 0 | 0 | 1 | 0 | 0 | x | x | 0 | | 1 | 0 |
| 00011 | LDR | 1 | instr[10:8] | MEMout | 1 | instr[7:5] | | instr[4:0] | 1 | 0 | 0 | 0 | 1 | 0 | ALUout | x | 0 | | 1 | 0 |
| 00101 | STR | 0 | x | x | 0 | instr[7:5] | instr[10:8] | instr[4:0] | 1 | 0 | 0 | 0 | 0 | 1 | ALUout | RB | 0 | | 1 | 0 |
| 00000 | ADD | 1 | instr[10:8] | ALUout | 0 | instr[7:5] | instr[4:2] | | 0 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | 1 | 0 |
| 00000 | ADC | 1 | instr[10:8] | ALUout | 0 | instr[7:5] | instr[4:2] | | 0 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | 1 | 0 |
| 00000 | SUB | 1 | instr[10:8] | ALUout | 0 | instr[7:5] | instr[4:2] | | 0 | 0 | 1 | 0 | 0 | 0 | x | x | 0 | | 1 | 0 |
| 00000 | SBB | 1 | instr[10:8] | ALUout | 0 | instr[7:5] | | | 0 | 0 | 1 | 0 | 0 | 0 | x | x | 0 | | 1 | 0 |
| 00110 | CMP | 0 | x | x | 0 | instr[7:5] | instr[4:2] | | 0 | 0 | 1 | 0 | 0 | 0 | x | x | 0 | | 1 | 0 |
| 00111 | ADDI | 1 | instr[10:8] | ALUout | 0 | instr[7:5] | | instr[4:0] | 1 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | 1 | 0 |
| 01000 | SUBI | 1 | instr[10:8] | ALUout | 0 | instr[7:5] | | instr[4:0] | 1 | 0 | 1 | 0 | 0 | 0 | x | x | 0 | | 1 | 0 |
| 01011 | MOV | 1 | instr[10:8] | ALUout | 0 | instr[7:5] | | 0x0000 | 1 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | 1 | 0 |
| C3 | BCC | 0 | x | x | 0 | x | x | | 0 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | disp | 0 |
| C2 | BCS | 0 | x | x | 0 | x | x | | 0 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | disp | 0 |
| C1 | BNE | 0 | x | x | 0 | x | x | | 0 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | disp | 0 |
| C0 | BEQ | 0 | x | x | 0 | x | x | | 0 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | disp | 0 |
| CE | B[AL] | 0 | x | x | 0 | x | x | | 0 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | disp | 0 |
| 10000 | JMP | 0 | x | x | 0 | x | x | | 0 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | instr[10:0] | 1 |
| 10001 | JAL | 1 | instr[10:8] | pcaddr | 1 | x | x | | 0 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | disp | 0 |
| 10010 | JAL | 1 | instr[10:8] | pcaddr | 1 | instr[7:5] | | 0x0000 | 1 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | ALUout | 1 |
| 10011 | JR | 0 | x | x | 0 | instr[7:5] | | 0x0000 | 1 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | ALUout | 1 |
| 11100 | OutR | 0 | x | x | 0 | instr[7:5] | | 0x0000 | 1 | 0 | 0 | 0 | 0 | 0 | x | x | 0 | | 1 | 0 |
| 11100 | HLT | 0 | x | x | 0 | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | 1 | | 0 | 0 |

設計部分，一開始選這堂課的時候，就知道這堂課需要具備計算機組織與數位邏輯還有寫硬體描述語言

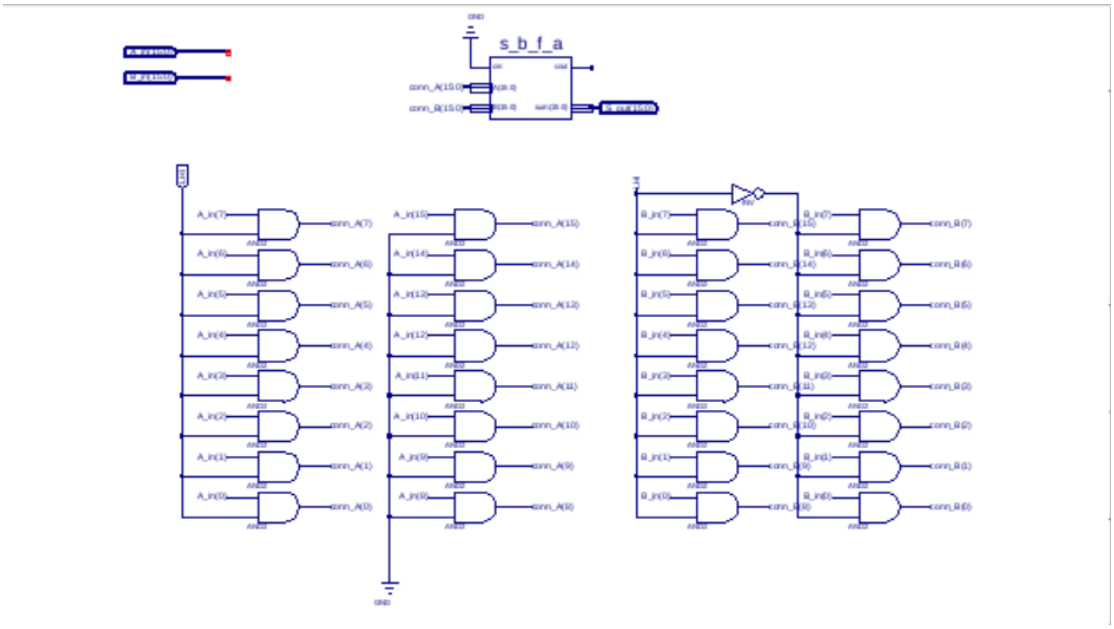
的能力，因此架構的部分除了上網查之外，也上了清大 OCW 的課程，所以在開始拉電路之前，資料該怎麼流與怎麼控制，已有初步的想法與規劃。

# Schematic

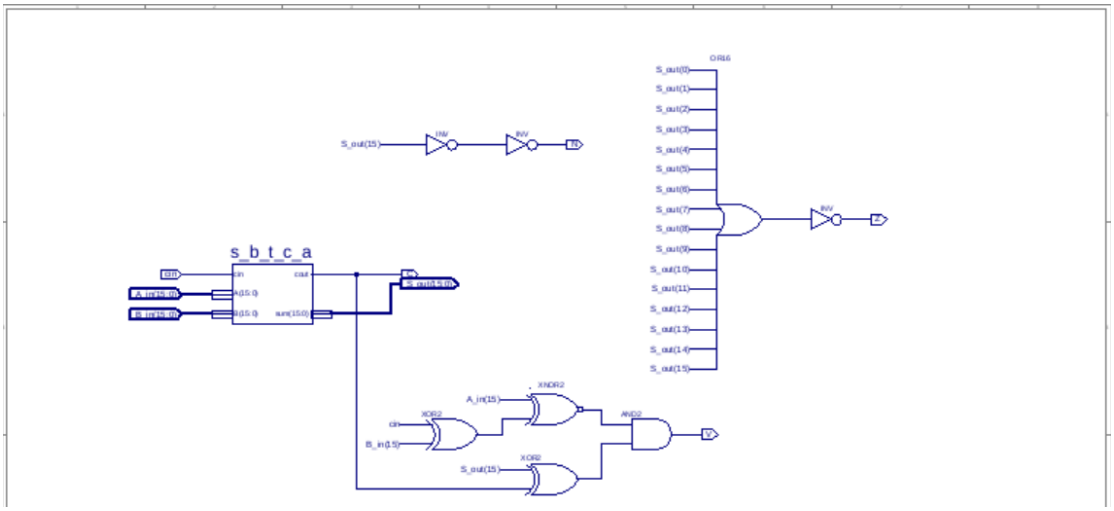
## 1.16-Bit Complement Adder



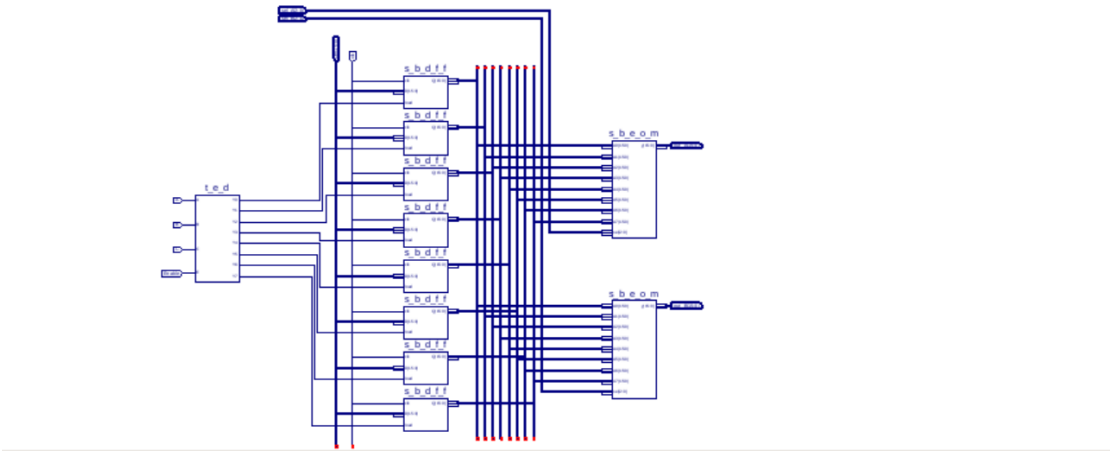
## 2.LHI_LLI(用來處理 LHI 和 LLI 指令)



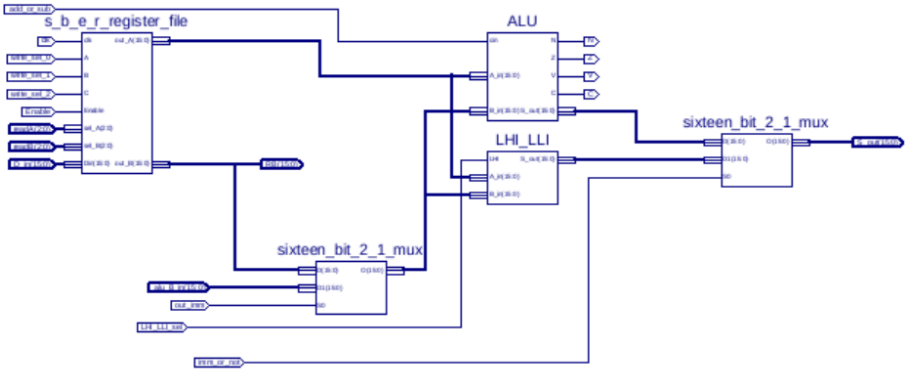
## 3.ALU



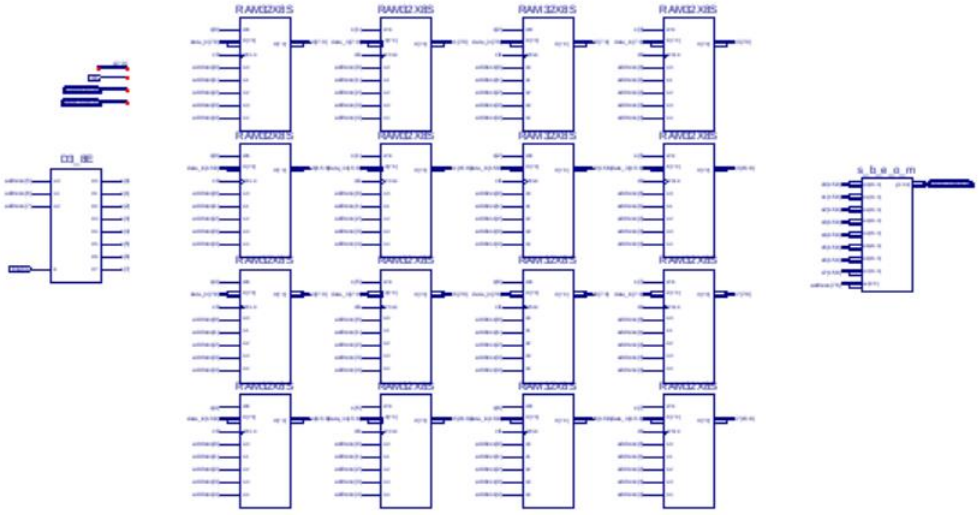

## 4.16-Bit Eight-Register Register File



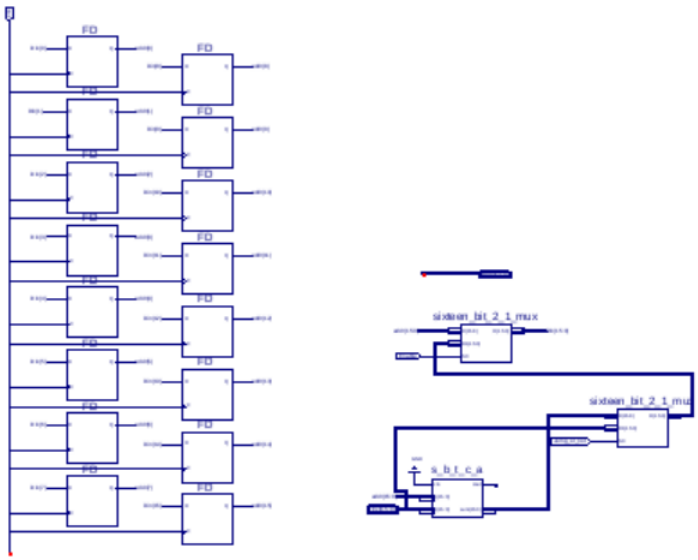
## 5.16-Bit RF-plus-ALU



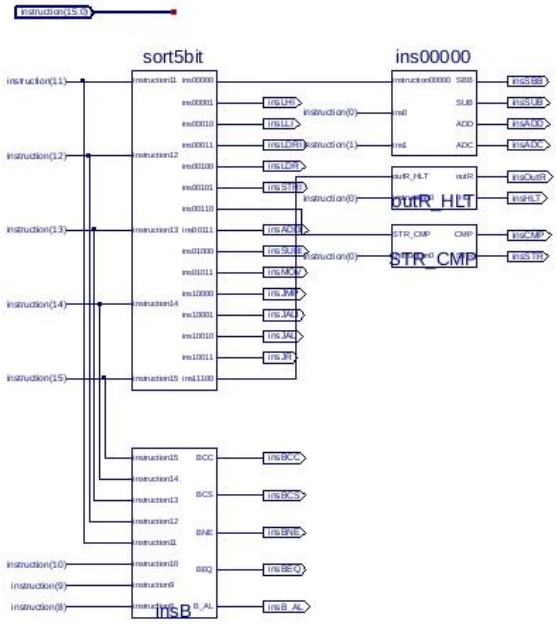

## 6.256*16 Memory Module



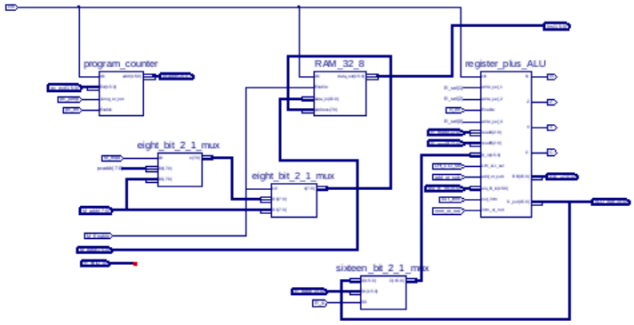
## 7.Program Counter



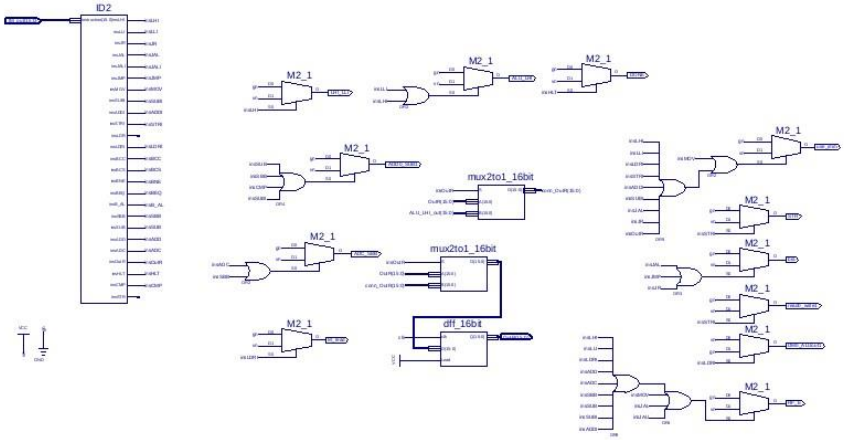

## 8.Instruction Decoder



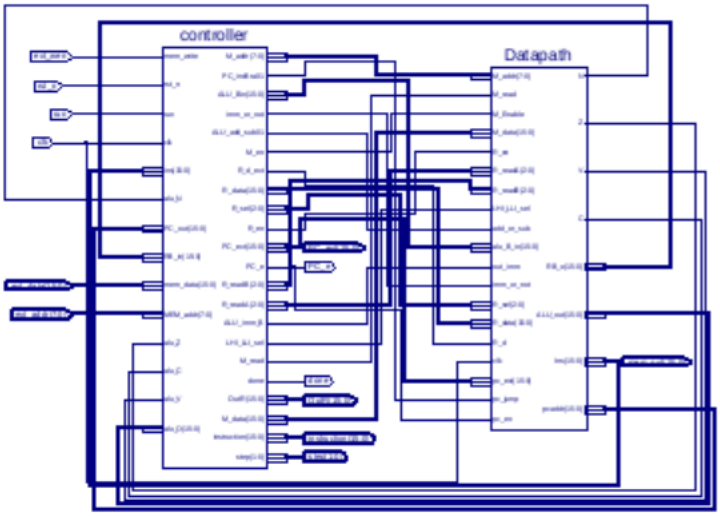
## 9.Datapath




## 10.Controller(partial)

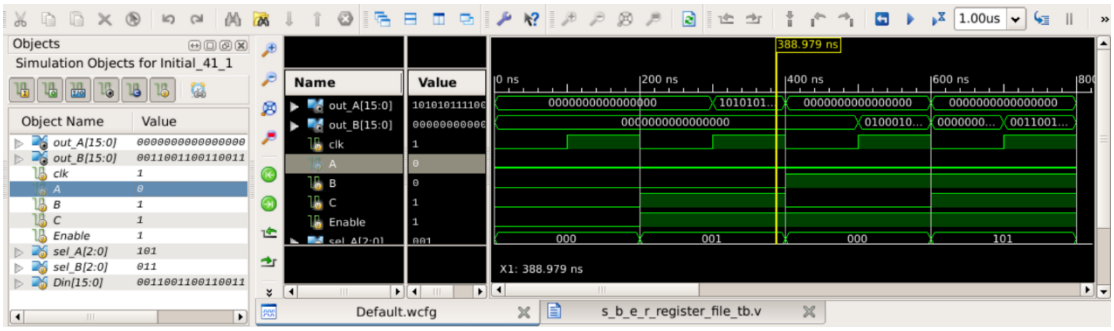


# 11.Complete Computer

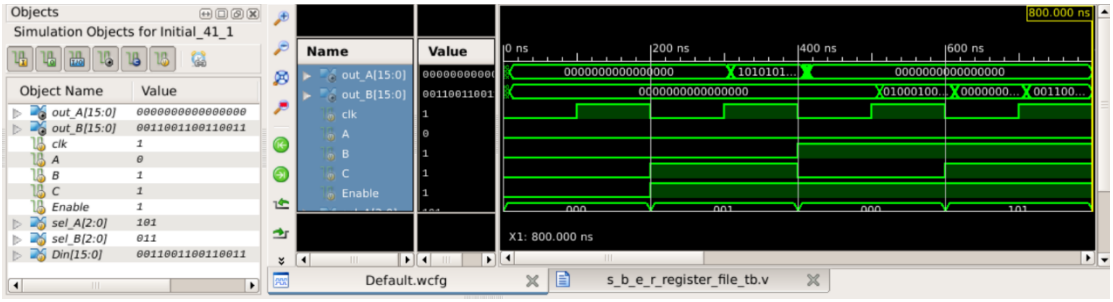


# Verification

## 1.16-Bit Eight-Register Register File Test Bench

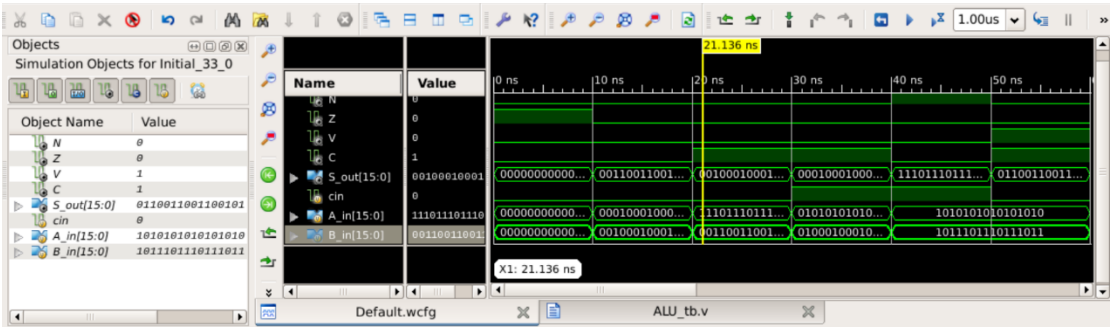



## Post-Route

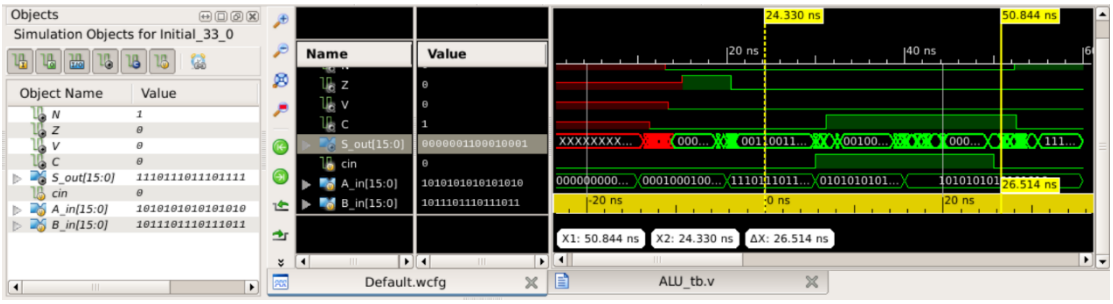



```
initial begin
    clk=0;
end
forever #100 clk=~clk;
initial begin
    A=0;
    B=0;
    C=0;
    Enable=0;
    sel_A[2:0]=0;
    sel_B[2:0]=0;
    Din[15:0]=0;
    #200;

    A=0;
    B=0;
    C=1;
    Enable=1;
    Din[15:0]=16'habcd;
    sel_A[2:0]=3'b001;
    sel_B[2:0]=3'b010;
    #200;

    A=0;
    B=1;
    C=0;
    Enable=1;
    Din[15:0]=16'h4444;
    sel_A[2:0]=3'b000;
    sel_B[2:0]=3'b010;
    #200;

    A=0;
    B=1;
    C=1;
    Enable=1;
    Din[15:0]=16'h3333;
    sel_A[2:0]=3'b101;
    sel_B[2:0]=3'b011;
    #200;

    $finish;
end
```

# 2.16-bit ALU Test Bench

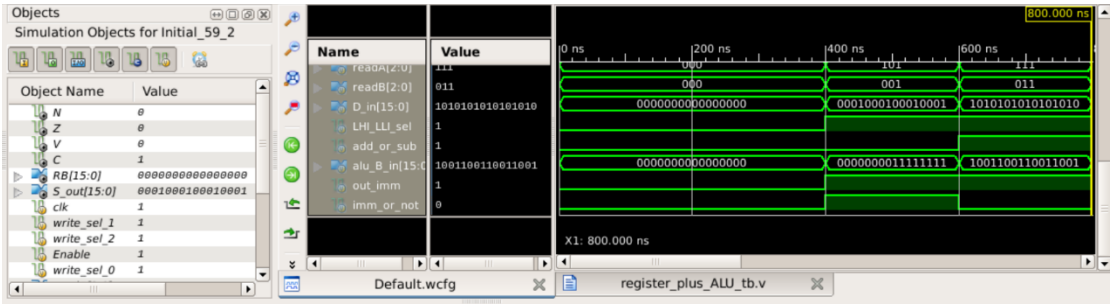


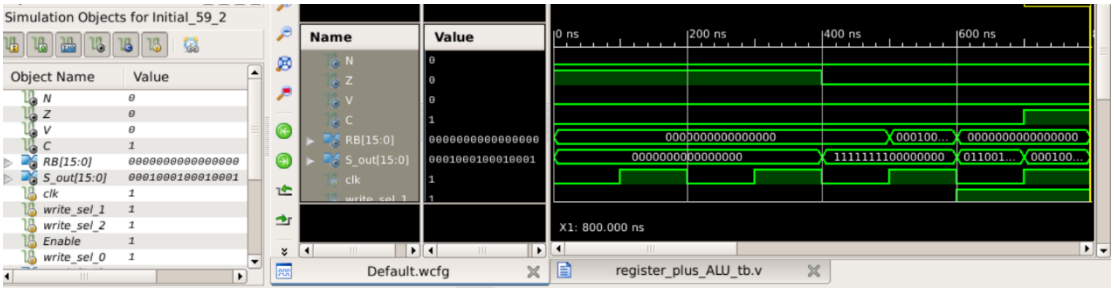

# Post-Route



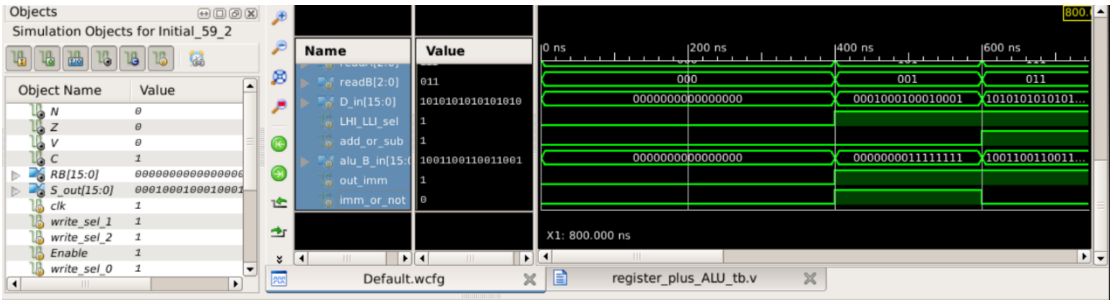

```
initial begin
    cin=0;
    A_in[15:0]=0;
    B_in[15:0]=0;
    #10;

    cin=0;
    A_in[15:0]=16'h1111;
    B_in[15:0]=16'h2222;
    #10;

    cin=0;
    A_in[15:0]=16'heeee;
    B_in[15:0]=16'h3333;
    #10;

    cin=1;
    A_in[15:0]=16'h5555;
    B_in[15:0]=16'h4444;
    #10;

    cin=1;
    A_in[15:0]=16'haaaa;
    B_in[15:0]=16'hbbbb;
    #10;

    cin=0;
    A_in[15:0]=16'haaaa;
    B_in[15:0]=16'hbbbb;
    #10;

end
```

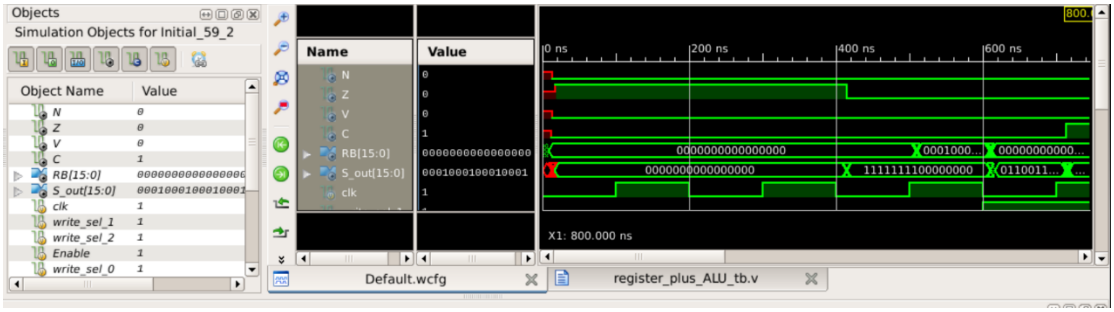
# 3.16-Bit RF-plus-ALU Test Bench



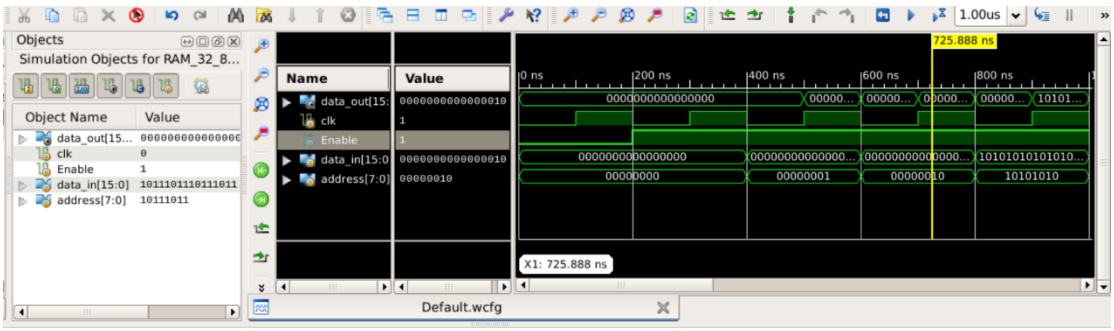




# Post-Route

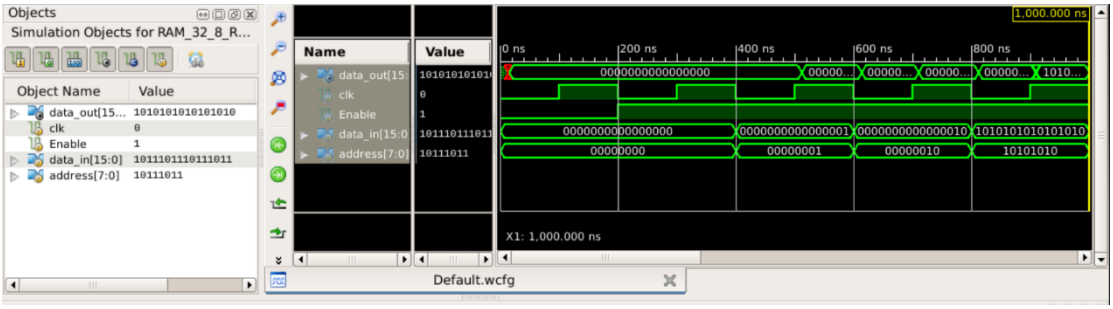


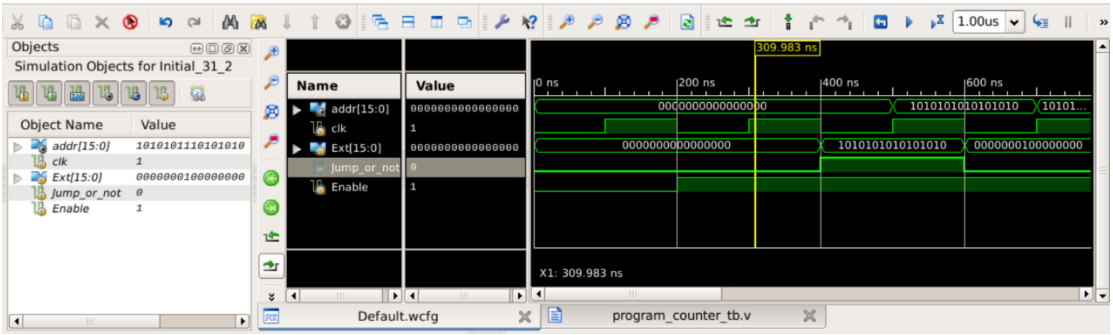
# 4.256*16 Memory Module Test Bench



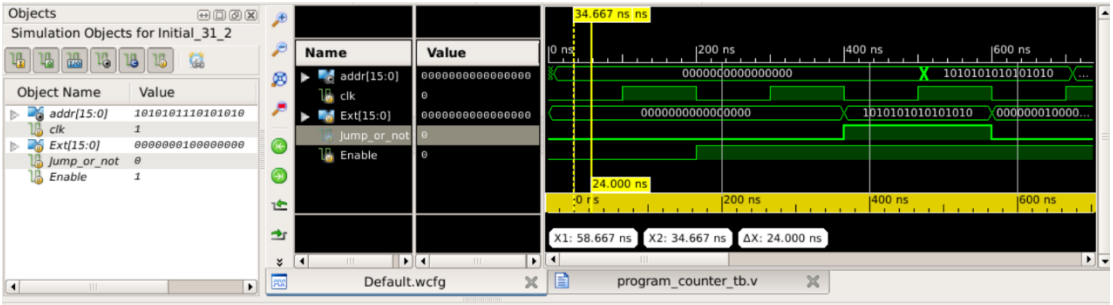

# Post-Route




# 5.Program Counter Test Bench

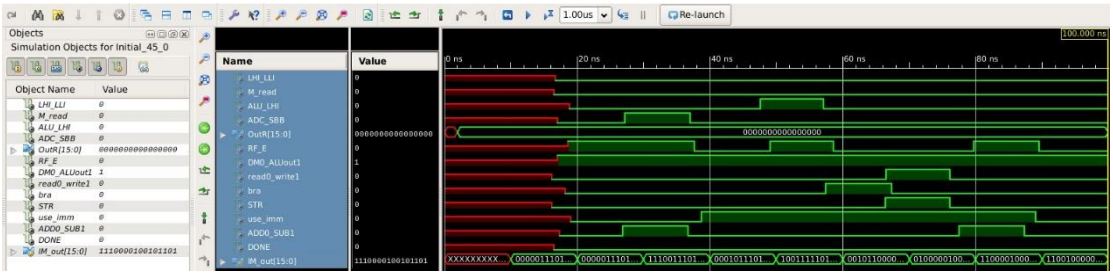



# Post-Route



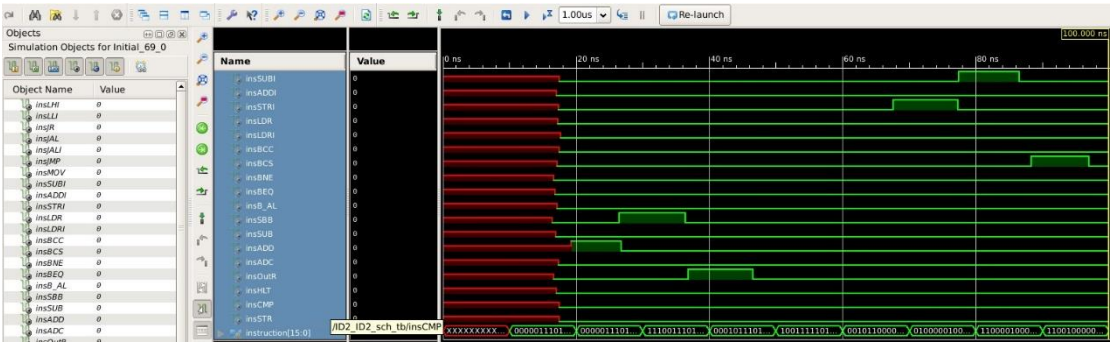
# 6.Controller Test Bench

# Post-Route



# 7.Instruction Decoder Test Bench

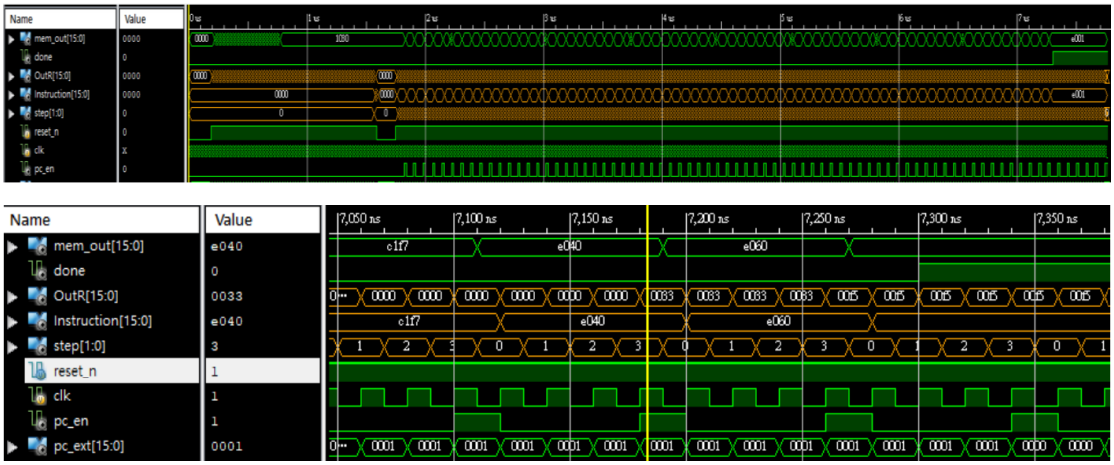
# Post-Route



1. Find the minimum and maximum from two numbers in memory

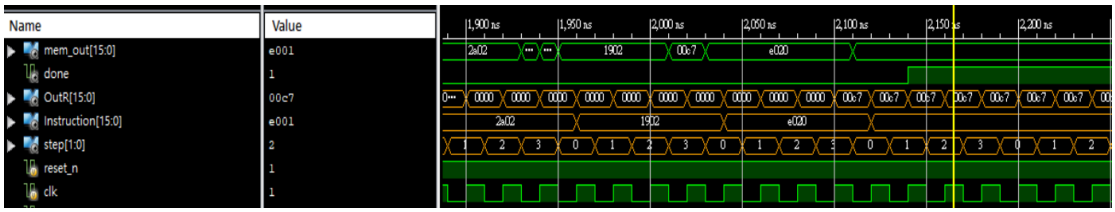
| | A | B | C | D |
|---|---|---|---|---|
| 1 | HEX | label | ASSEMBLY | COMMAND |
| 2 | 0 | | LLI R0,#30H | 00010_000_00110000 |
| 3 | 1 | | LLI R1,#37H | 00010_001_00110111 |
| 4 | 2 | | LLI R2,#255 | 00010_010_11111111 |
| 5 | 3 | | LHI R2,#7fH | 00001_010_01111111 |
| 6 | 4 | | LLI R3,#0 | 00010_011_00000000 |
| 7 | 5 | lab0 | LDR R4,R0,#0 | 00011_100_000_00000 |
| 8 | 6 | | CMP R2,R4 | 00110_000_010_100_01 |
| 9 | 7 | | BCC lab1 | 11000011_00000010 |
| 10 | 8 | | MOV R2,R4 | 01011_010_100_00000 |
| 11 | 9 | lab1 | CMP R3,R4 | 00110_000_011_100_01 |
| 12 | a | | BCS lab2 | 11000010_00000010 |
| 13 | b | | MOV R3,R4 | 01011_011_100_00000 |
| 14 | c | lab2 | ADDI R0,R0,#1 | 00111_000_000_00001 |
| 15 | d | | CMP R1,R0 | 00110_000_001_000_01 |
| 16 | e | | BNE lab0 | 11000001_11110111 |
| 17 | f | | OutR R2 | 11100_000_010_000_00 |
| 18 | 10 | | OutR R3 | 11100_000_011_000_00 |
| 19 | 11 | | HLT | 11100_00000000_01 |

```
write_mem(16'h0,16'b00010_000_00110000 ) ;//LLI R0,#30H
write_mem(16'h1,16'b00010_001_00110111 ) ; //LLI R1,#37H
write_mem(16'h3,16'b00010_010_11111111 ) ; //LLI R2,#255
write_mem(16'h4,16'b00001_010_01111111 ) ; //LHI R2,#7fH
write_mem(16'h5,16'b00010_011_00000000 ) ; //LLI R3,#0
write_mem(16'h6,16'b00011_100_000_00000 ) ; //LDR R4,R0,#0
write_mem(16'h7,16'b00110_000_010_100_01 ) ; //CMP R2,R4
write_mem(16'h8,16'b11000011_00000010 ) ; //BCC labl
write_mem(16'h9,16'b01011_010_100_00000 ) ; //MOV R2,R4
write_mem(16'hA,16'b00110_000_011_100_01 ) ; //CMP R3,R4
write_mem(16'hA,16'b11000010_00000010 ) ; //BCS lab2
write_mem(16'hB,16'b01011_011_100_00000 ) ; //MOV R3,R4
write_mem(16'hB,16'b00111_000_000_00001 ) ; //ADDI R0,R0,#1
write_mem(16'hB,16'b00110_000_001_000_01 ) ; //CMP R1,R0
write_mem(16'hB,16'b11000001_11110111 ) ; //BNE lab0
write_mem(16'hB,16'b11100_000_010_000_00 ) ; //OutR R2
write_mem(16'hB,16'b11100_000_011_000_00 ) ; //OutR R3
write_mem(16'hB,16'b11100_000000000_01 ) ; //HLT
```




2. Add two numbers in memory and store the result in memory location

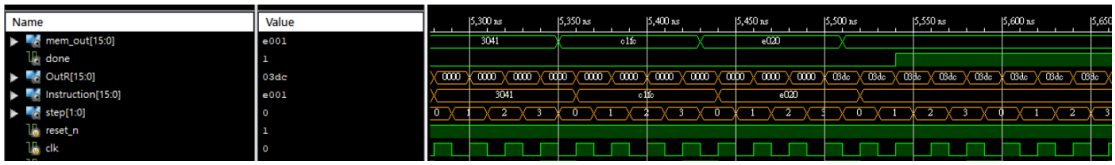
```
write_mem(16'h0,16'b00010_000_00110000 ) ;//LLI R0,#30H
write_mem(16'h1,16'b00011_011_000_00000 ) ; //LDR R3,R0,#0
write_mem(16'h2,16'b00011_100_000_00001 ) ; //LDR R4,R0,#1
write_mem(16'h3,16’b00000_010_011_100_00 ) ; //ADD R2,R3,R4
write_mem(16'h4,16'b00101_010_000_00010 ) ; //STR R4,R0,#0
write_mem(16'h5,16'b00011_001_000_00010 ) ; //LDR R1,R0,#2
write_mem(16'h6,16'b11100_000_001_000_00 ) ; //OutR R1
write_mem(16'h7,16'b11100_000000000_01 ) ; //HLT
```




3. Add ten numbers in consecutive memory locations

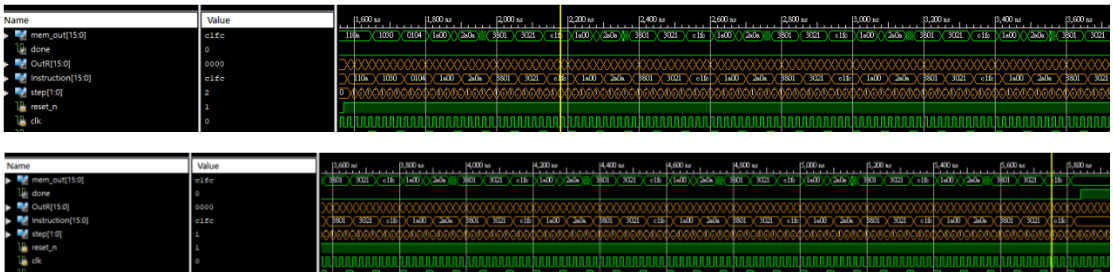
```
write_mem(16'h0,16'b00010_000_00110000 ) ;//LLI R0,#30H
write_mem(16'h1,16'b00010_010_00111001 ) ;//LLI R2,#39H
write_mem(16'h2,16'b00011_001_000_00000 ) ; //LDR R1,R0,#0
write_mem(16'h3,16’b00111_001_001_011_00 ) ; //ADDI R0,R0,#1
write_mem(16'h4,16'b00011_011_000_00000 ) ; //LDR R3,R0,#0
write_mem(16'h5,16’b00000_001_001_011_00 ) ; //ADD R1,R1,R3
write_mem(16'h6,16'b00110_000_010_000_01 ) ; //CMP R2,R0
write_mem(16'h7,16'b11000001_11110111 ) ; //BNE lab0
```

write_mem(16'h8,16'b11100_000_001_000_00 ) ; //OutR R1
write_mem(16'h9,16'b11100_000000000_01 ) ; //HLT




4. Mov a memory block of N words from one place to another

write_mem(16'h0,16'b00010_000_00110000 ) ;//LLI R0,#30H
write_mem(16'h1,16'b00010_010_00111001 ) ;//LLI R2,#39H
write_mem(16'h2,16'b00011_001_000_00000 ) ; //LDR R1,R0,#0
write_mem(16'h3,16’b00111_001_001_011_00 ) ; //ADDI R0,R0,#1
write_mem(16'h4,16'b00011_011_000_00000 ) ; //LDR R3,R0,#0
write_mem(16'h5,16’b00000_001_001_011_00 ) ; //ADD R1,R1,R3
write_mem(16'h6,16'b00110_000_010_000_01 ) ; //CMP R2,R0
write_mem(16'h7,16'b11000001_11110111 ) ; //BNE lab0
write_mem(16'h8,16'b11100_000_001_000_00 ) ; //OutR R1
write_mem(16'h9,16'b11100_000000000_01 ) ; //HLT



# Hardware Cost

## 16-bits RISC CPU hardware cost

| Device Utilization Summary | | | |
|---|---|---|---|
| **Slice Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slice Registers | 136 | 30,064 | 1% |
| Number used as Flip Flops | 136 | | |
| Number used as Latches | 0 | | |
| Number used as Latch-thrus | 0 | | |
| Number used as AND/OR logics | 0 | | |
| Number of Slice LUTs | 643 | 15,032 | 4% |
| Number used as logic | 515 | 15,032 | 3% |
| Number using O6 output only | 515 | | |
| Number using O5 output only | 0 | | |
| Number using O5 and O6 | 0 | | |
| Number used as ROM | 0 | | |
| Number used as Memory | 128 | 3,664 | 3% |
| Number used as Dual Port RAM | 0 | | |
| Number used as Single Port RAM | 128 | | |
| Number using O6 output only | 0 | | |
| Number using O5 output only | 0 | | |
| Number using O5 and O6 | 128 | | |
| Number used as Shift Register | 0 | | |
| Number of occupied Slices | 220 | 3,758 | 5% |

Total number of LUTs 643

Total number of FFs 136

# Discussion

## 1. Design Entry

一開始我以為設計一個 RISC CPU 應該不難，因為網路上其實有很多類似的架構圖，但是再按圖施工的時候才發現，其實大部分有些東西都精簡過，所以如果沒先想好資料流動方式，其實是很容易卡關的。我花了很多時間在修改錯誤與理解每塊 module 的運作方式與功能，此外我大學的時候有學過數位邏輯和稍微寫過 verilog，但是電路是第一次拉，一開始蠻好玩的，可是遇到挫折的時候，其實蠻需要與人討論的，不然真的要花超多時間研究。最重要的地方是我認為我對每塊 module 的功能與溝通還不夠熟悉，Term Project 中也只有幾塊有很確切的解釋裡面該有哪些小模塊，其他就需要靠你對 CPU 架構的理解了。

## 2. Timing Simulation Error

另一方面就是我對時序的方面可能還不夠了解，我認為我的 behavior simulation 沒有太大的問題，但是在 timing simulation 上好像有些許錯誤，但是我可能無法如期找出問題把它修改好，接下來還要準備期中

考，希望我的期末 Project 可以把它改進並做得更好。