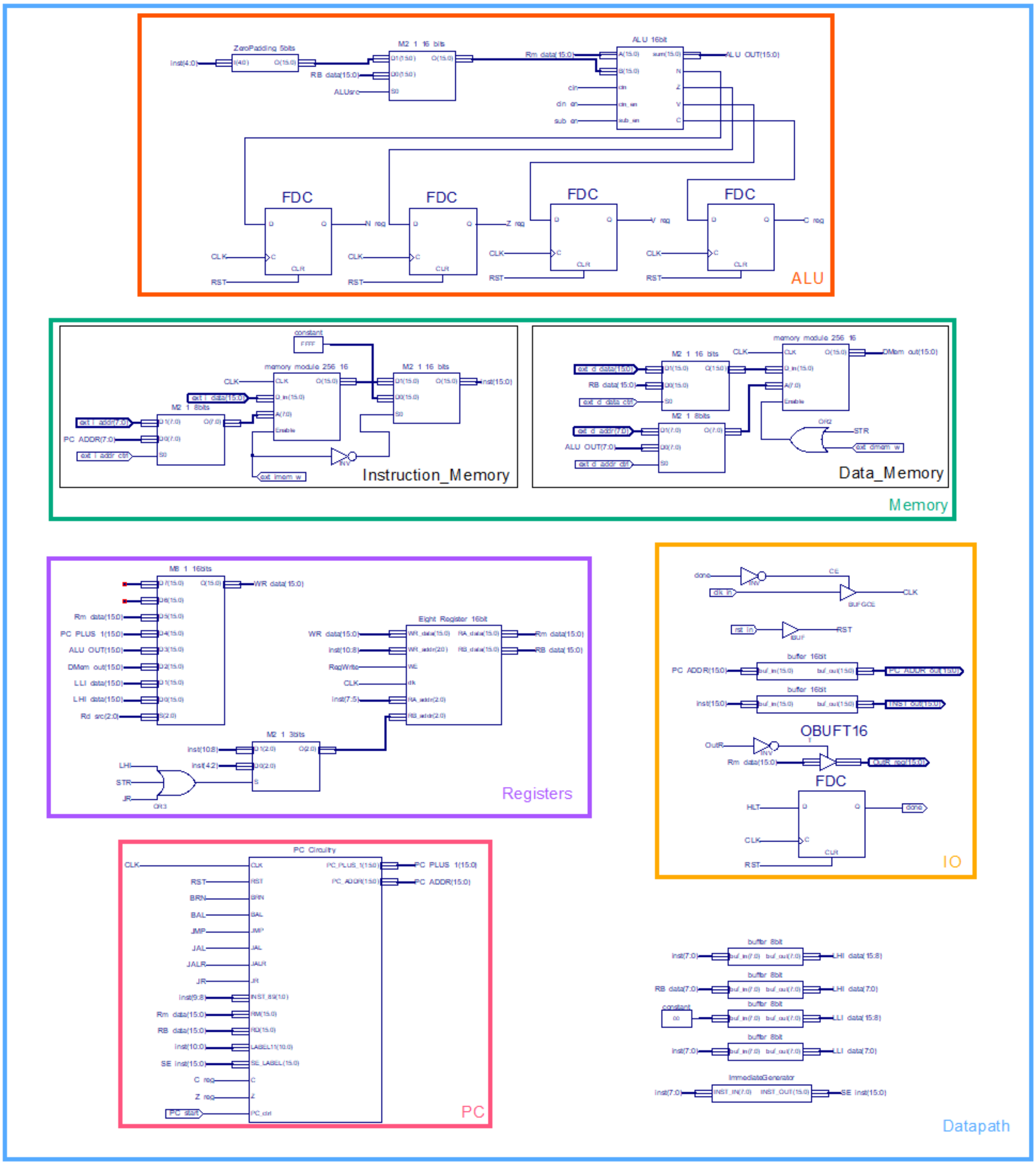
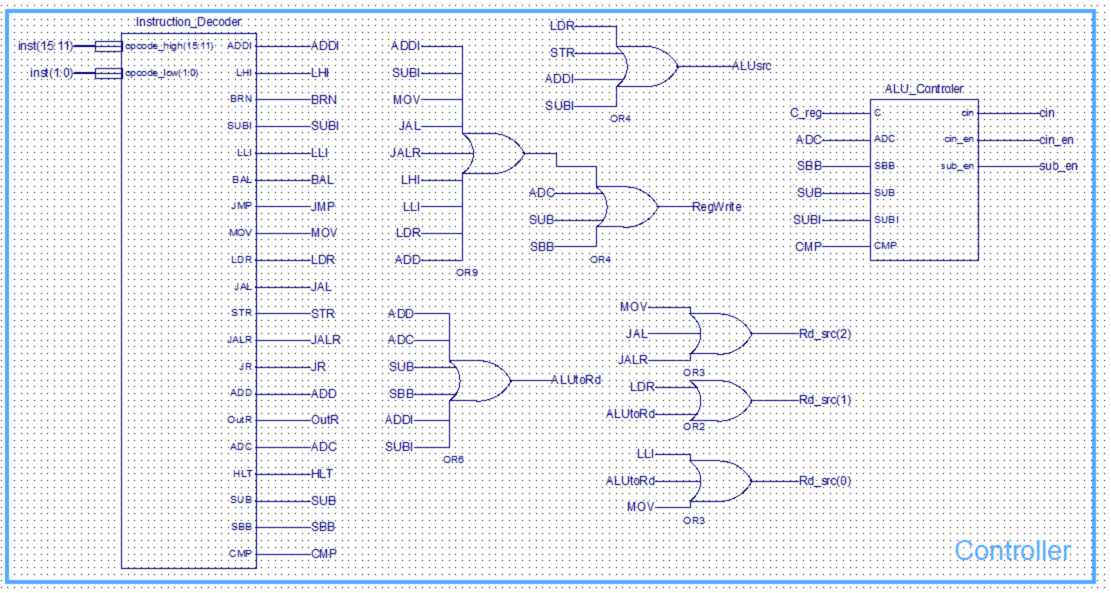
Schematic Entry

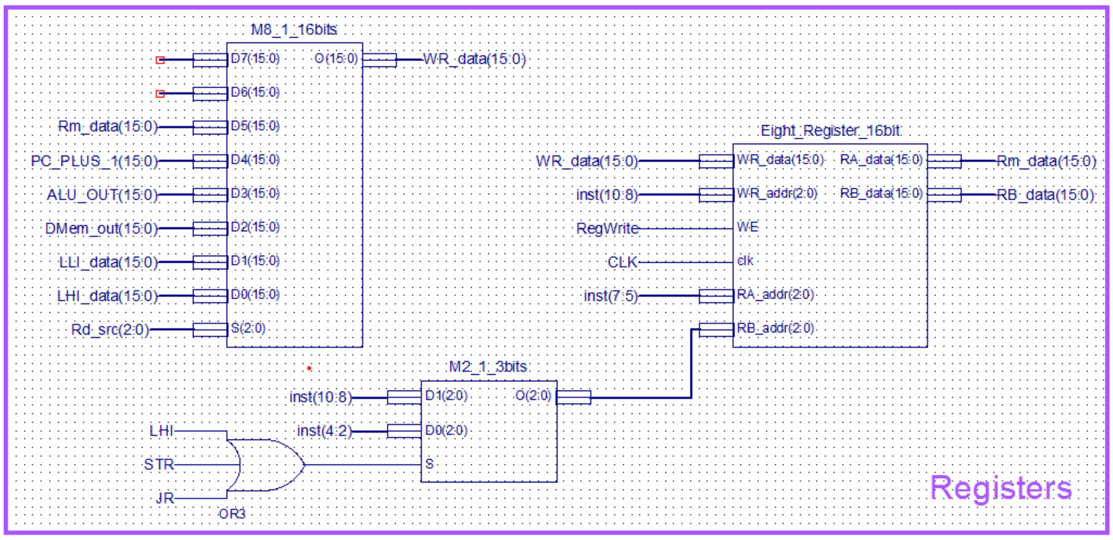
- Datapath



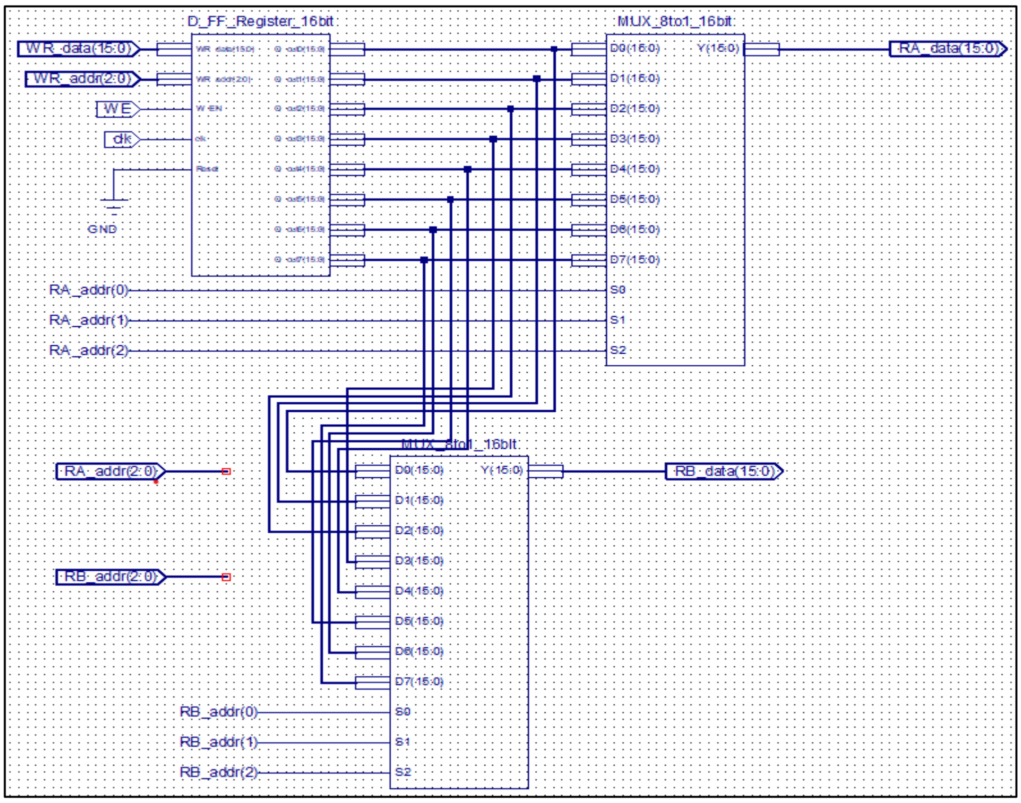
Controller



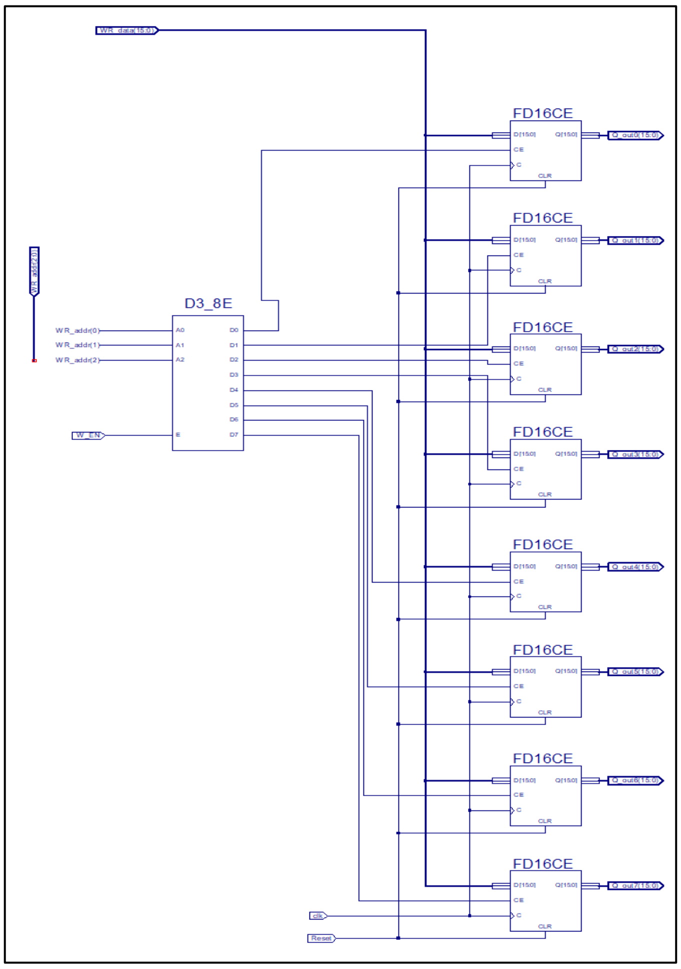
Registers



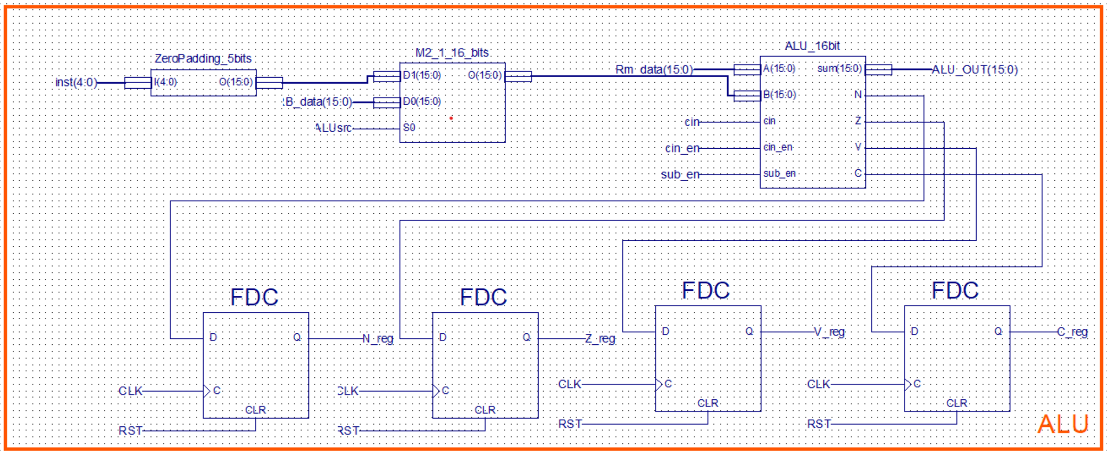
Eight_Register_16bit



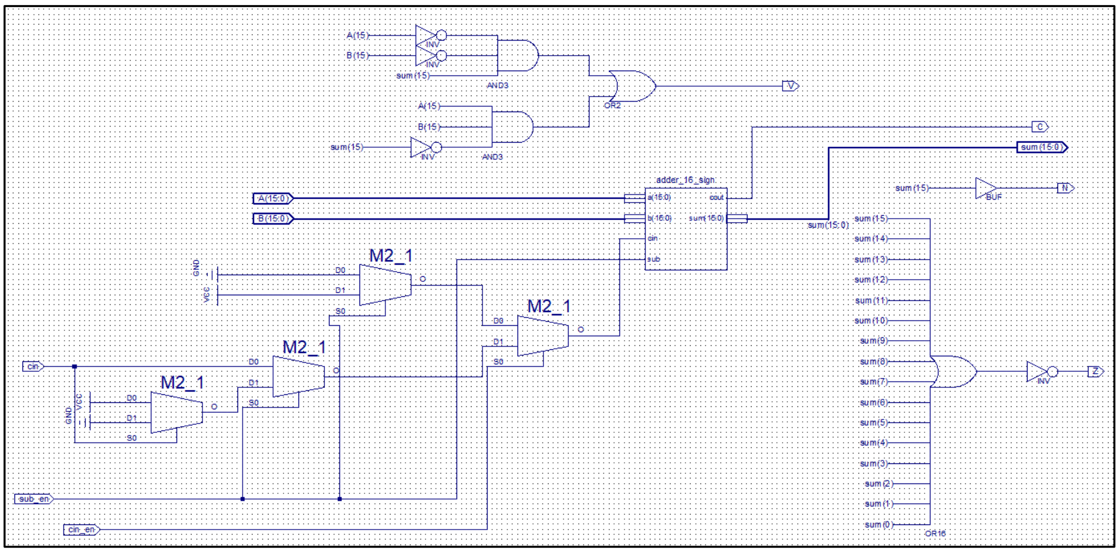
D_ff_Register_16bit



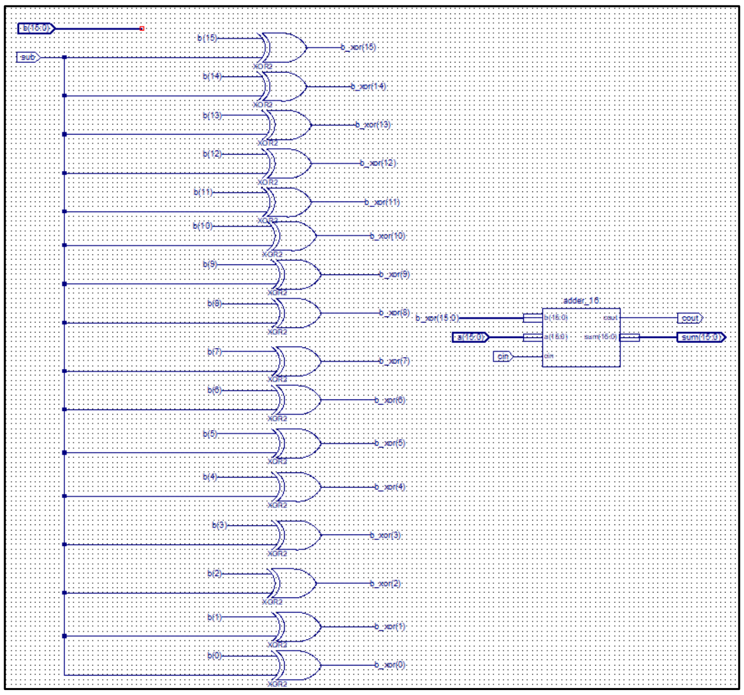
ALU



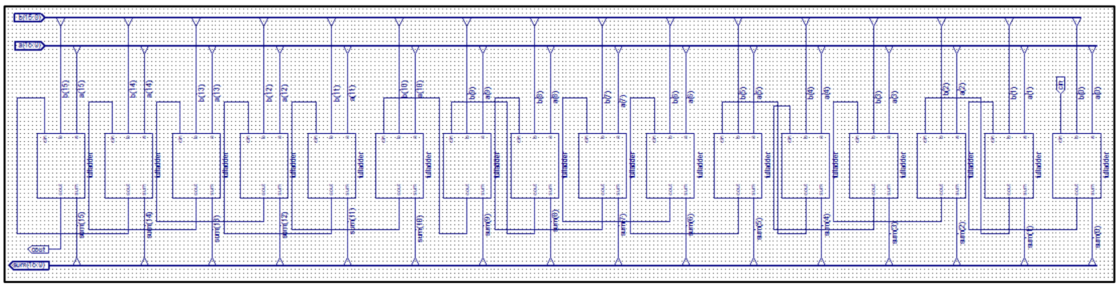
ALU_16bit



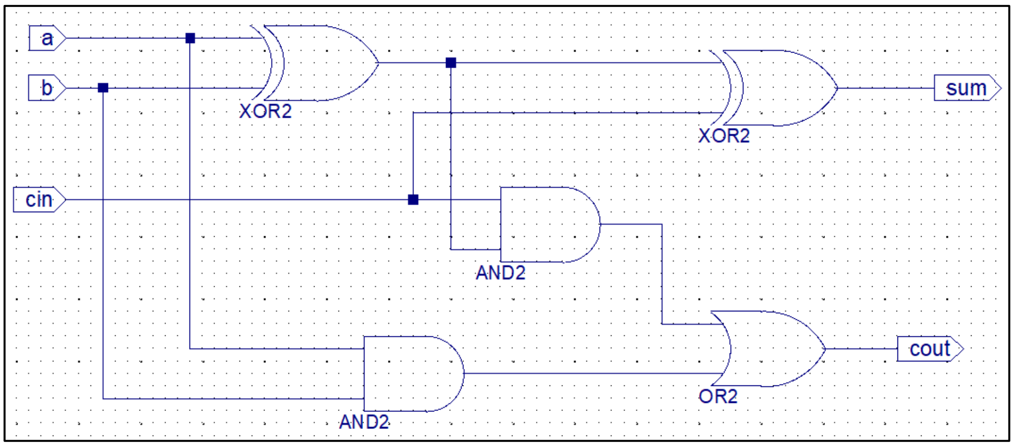
Adder_16_sign



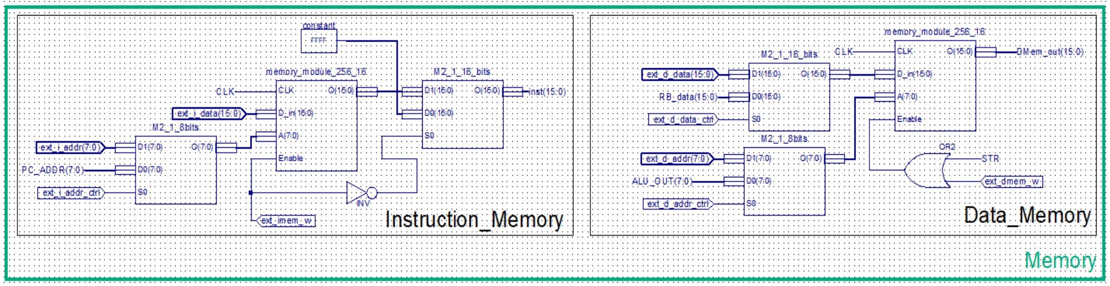
Adder_16



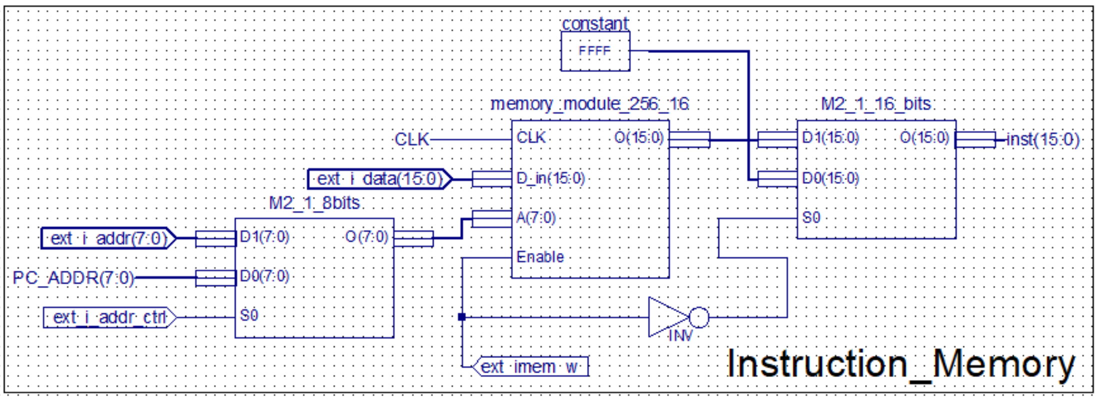
fulladder



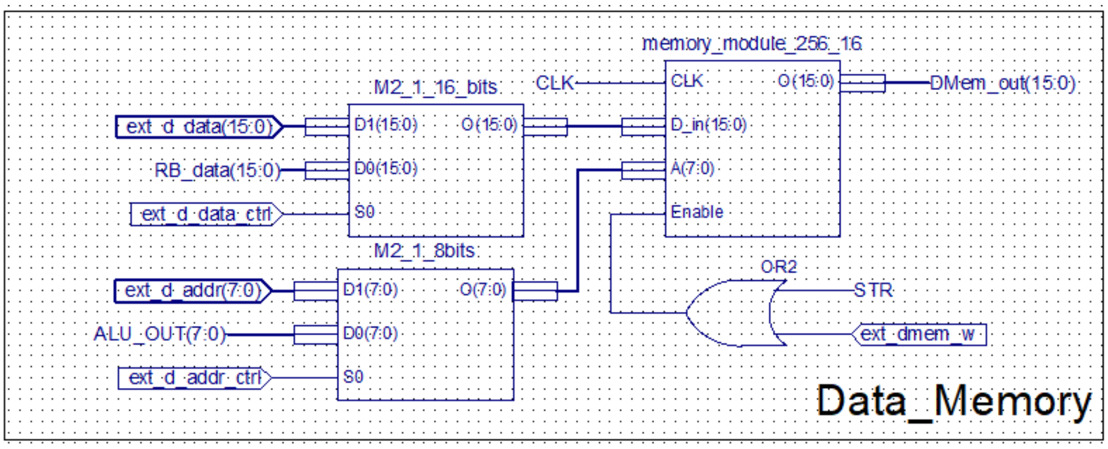
Memory



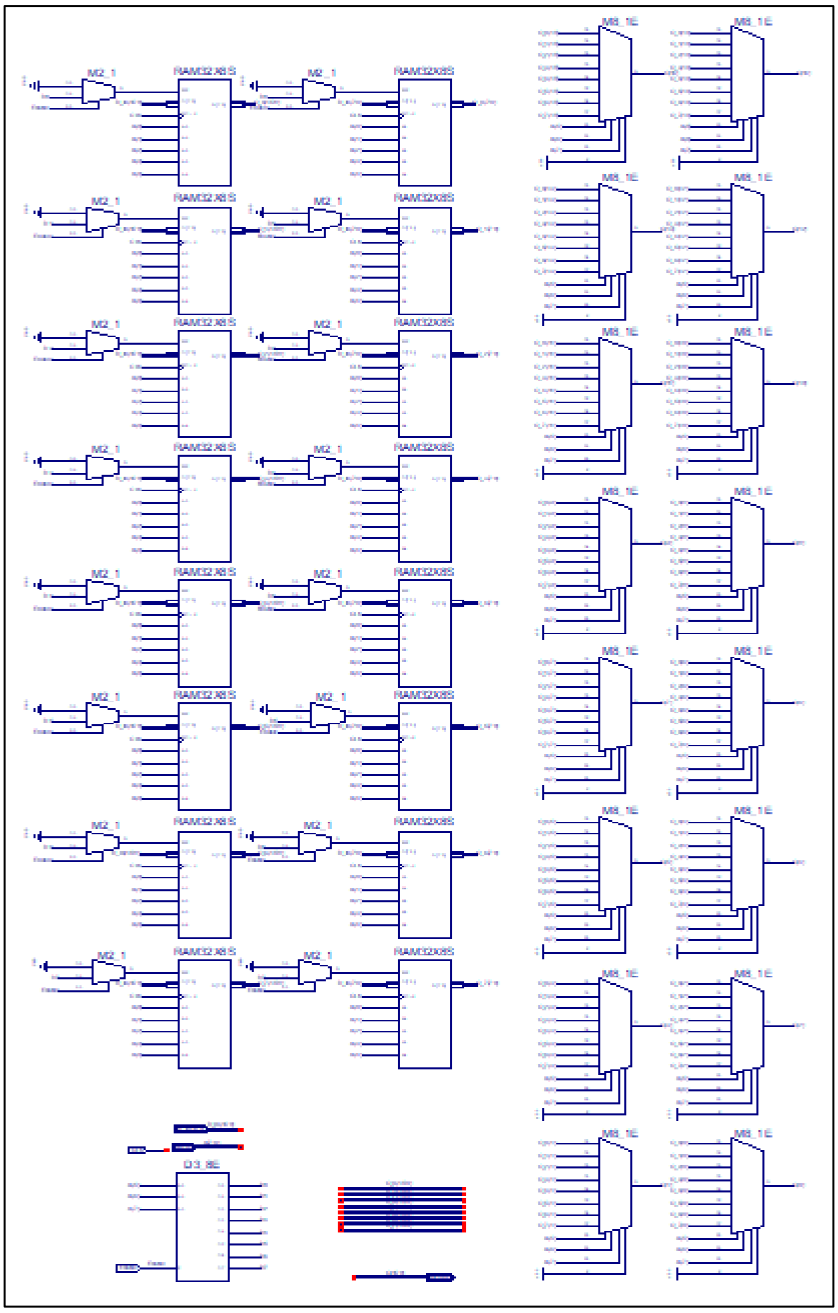
Instruction_Memory



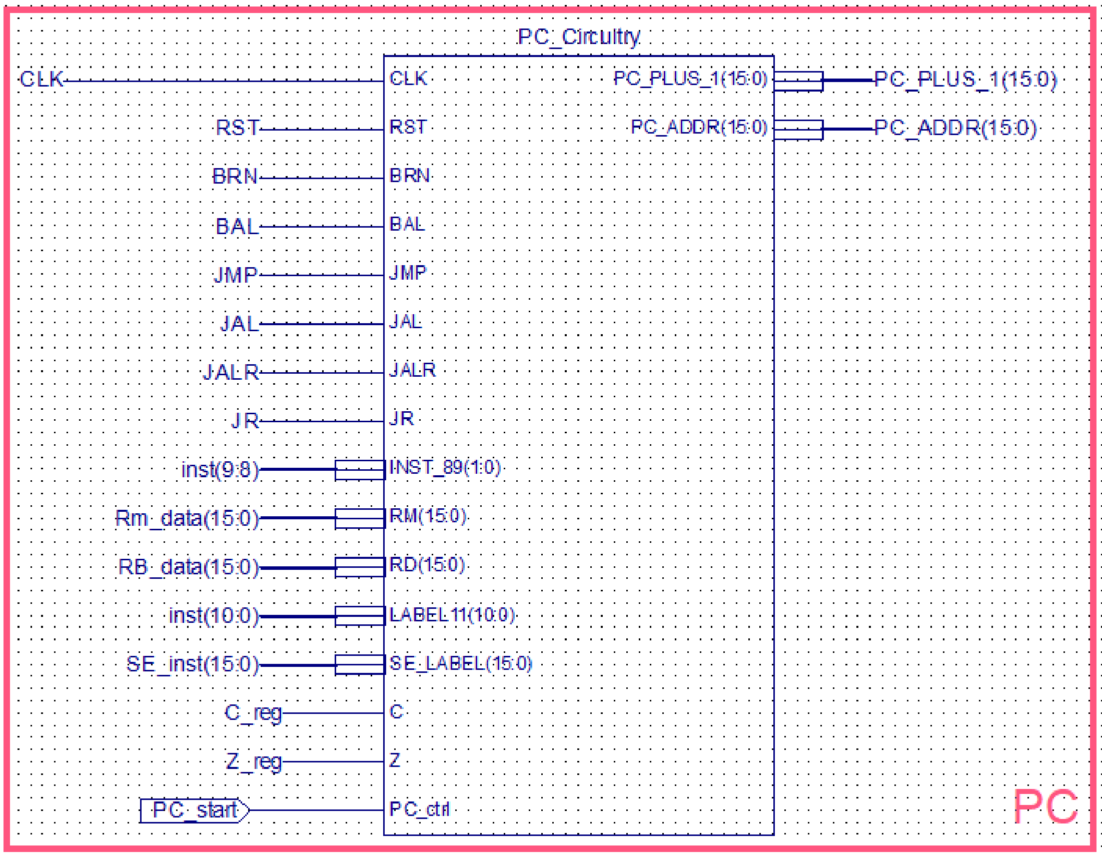
Data_Memory



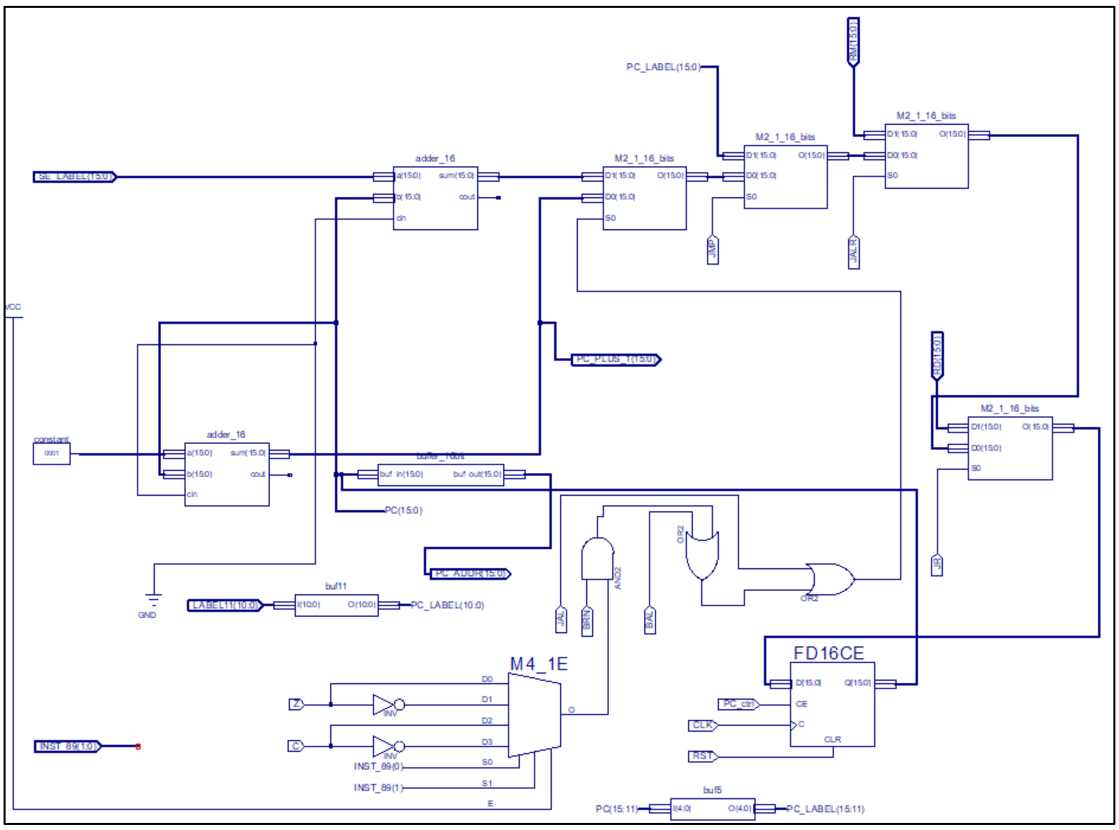
Memory_module_256_16



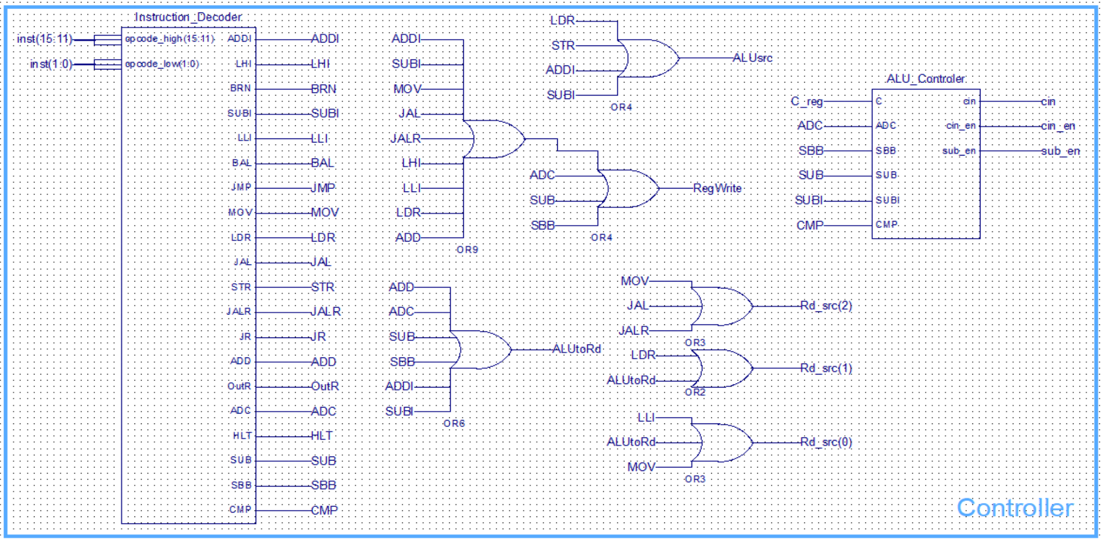
PC



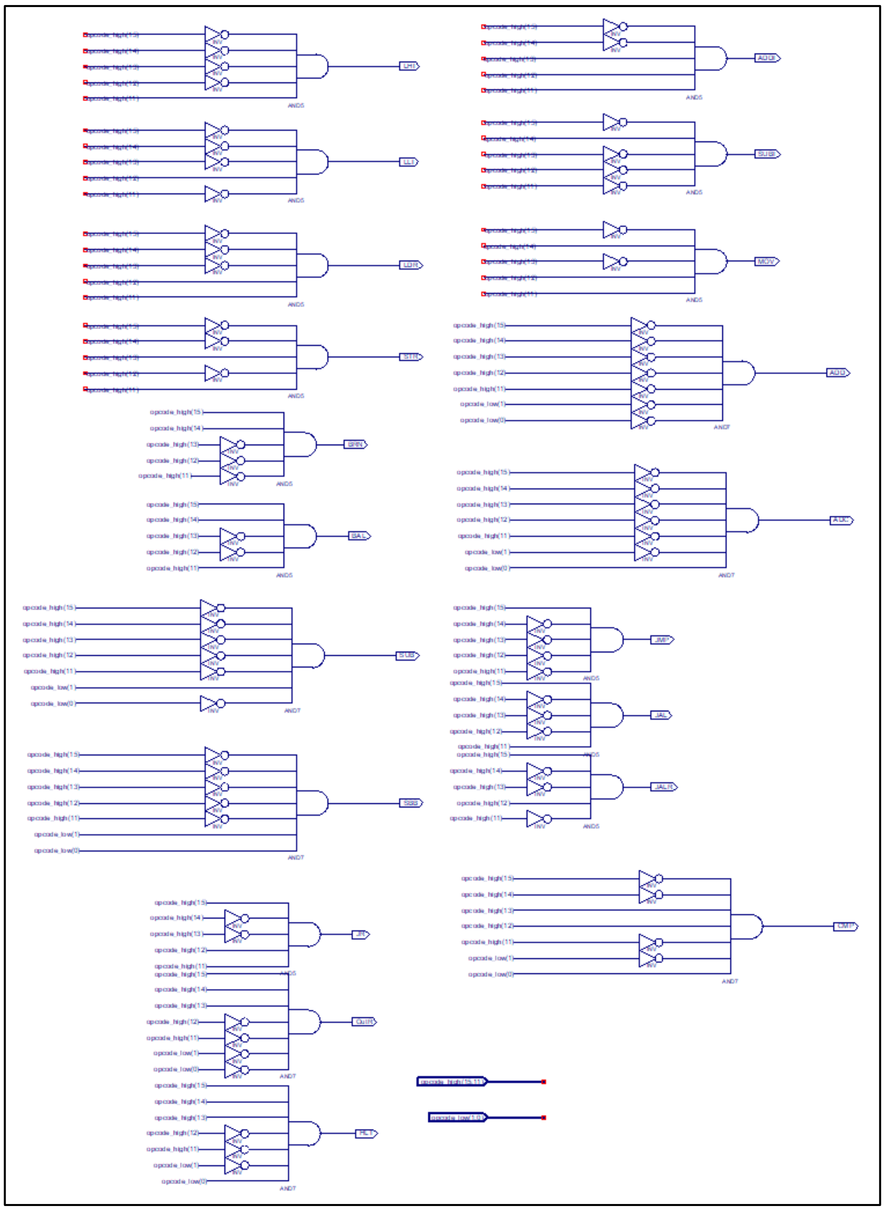
PC_Circultry



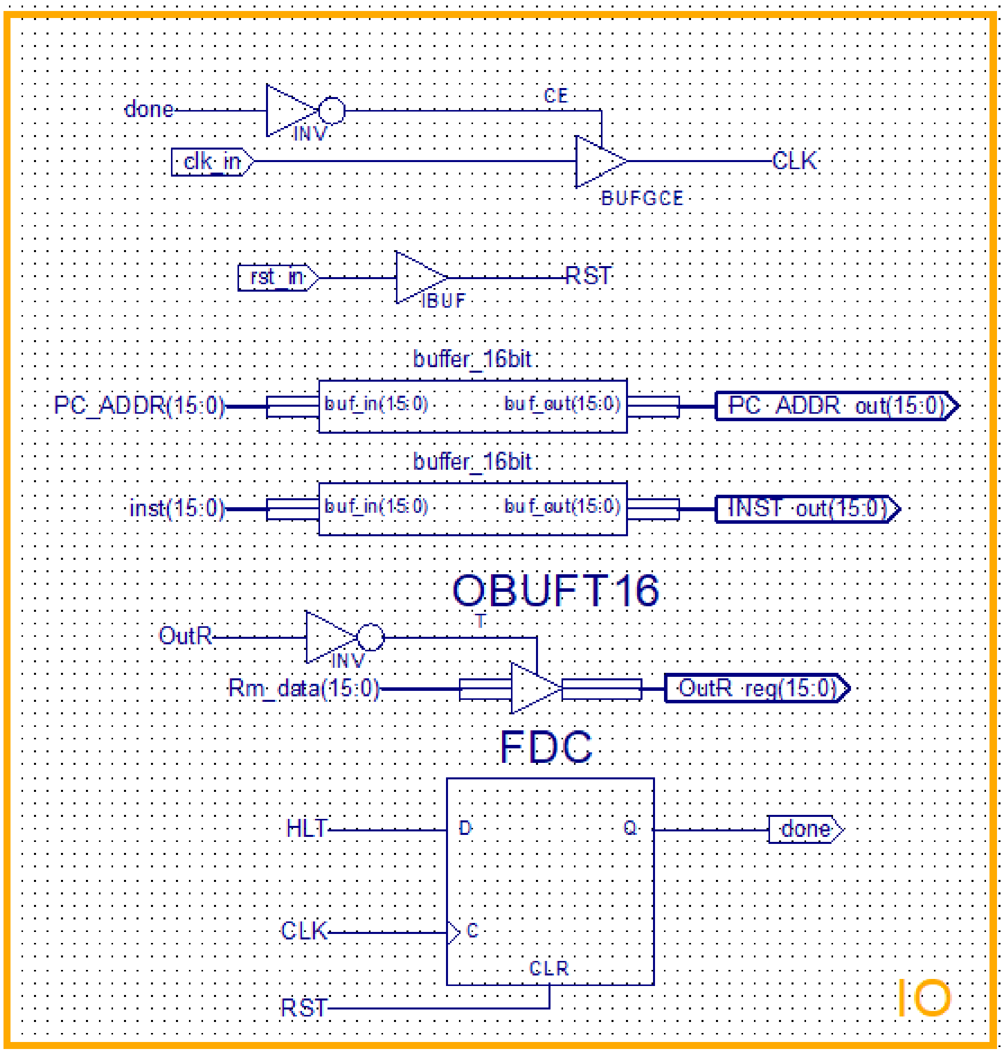
Controller



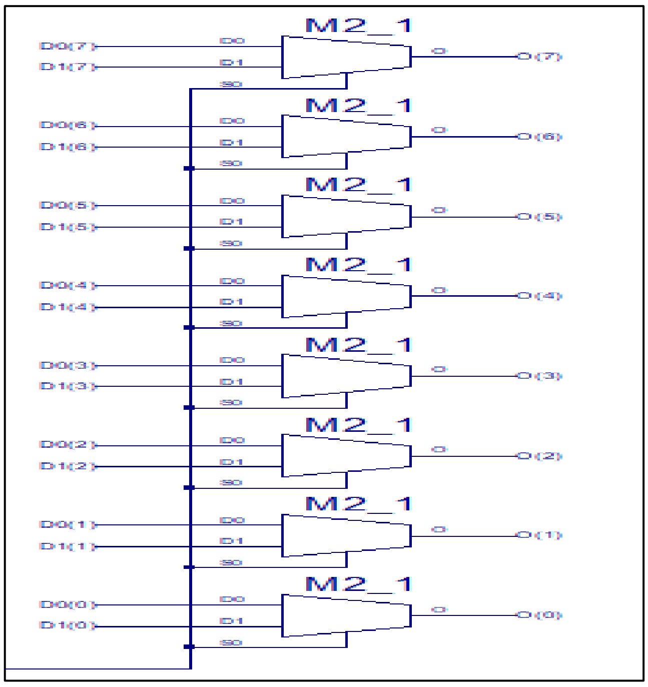
Instruction_Decoder



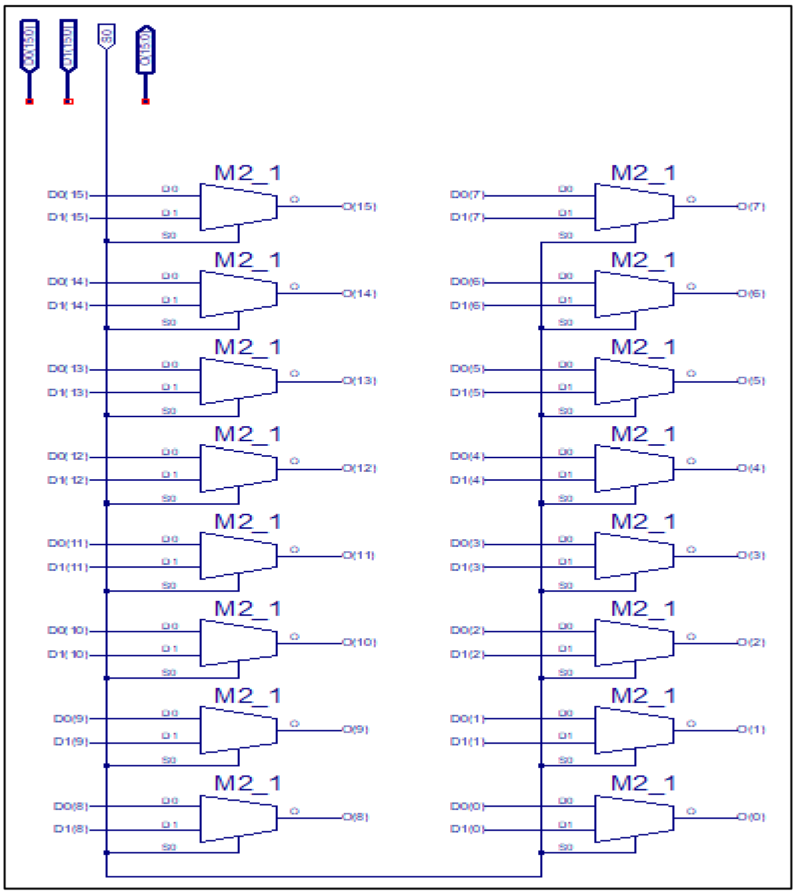
- IO



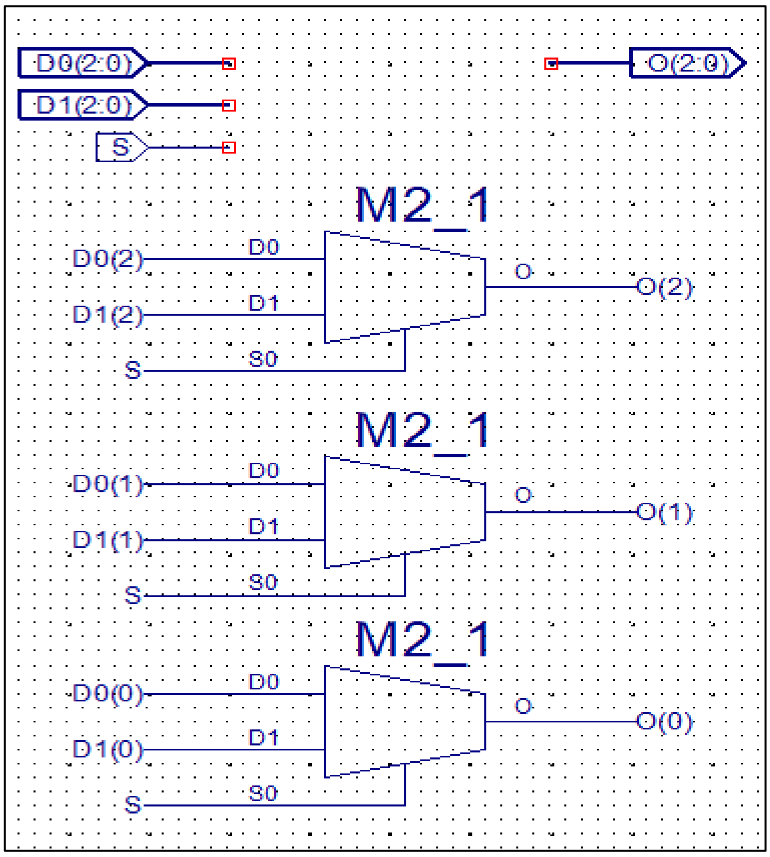
- 2 對 1 多工器(8bit)



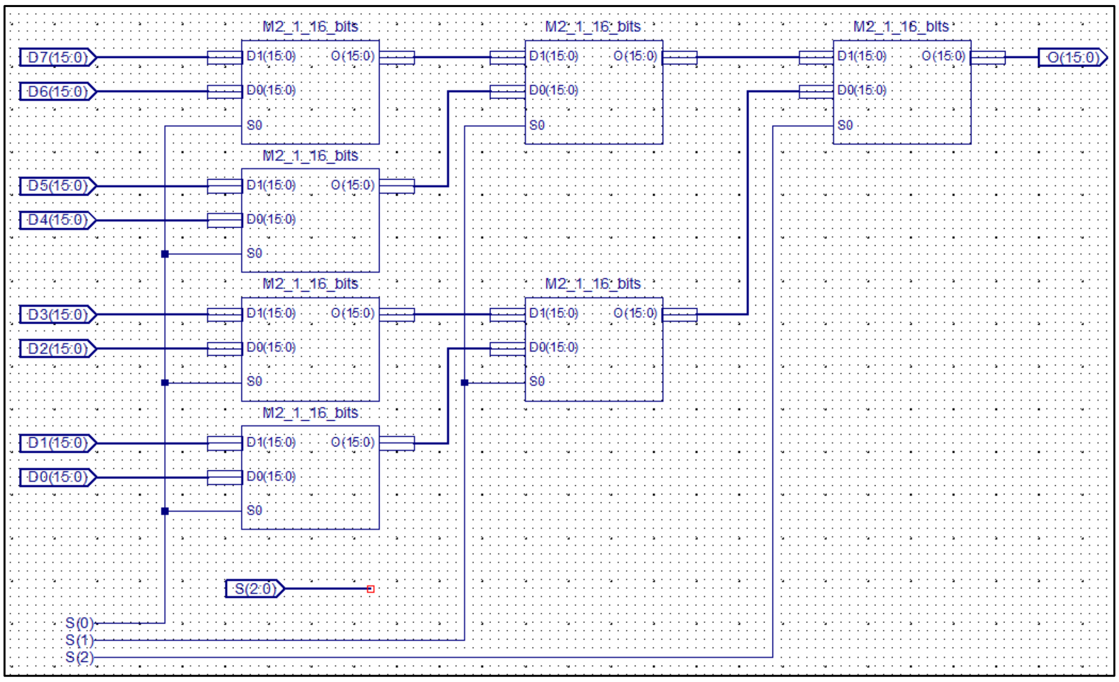
2 對 1 多工器(16bit)



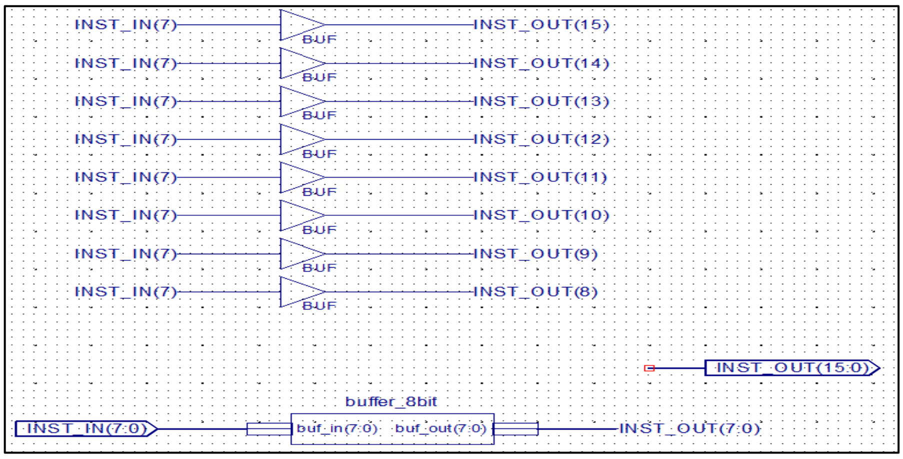
2 對 1 多工器(3bit)



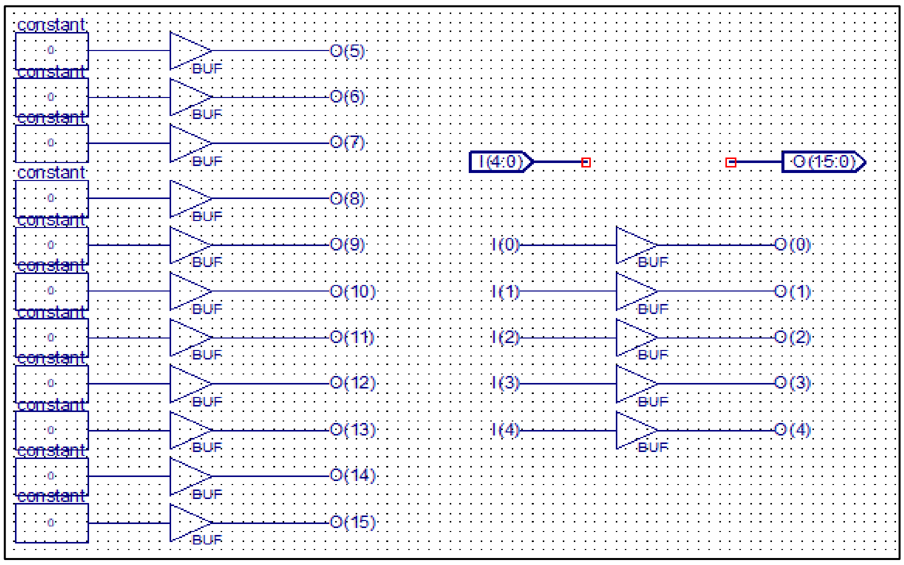
8 對 1 多工器(16bit)



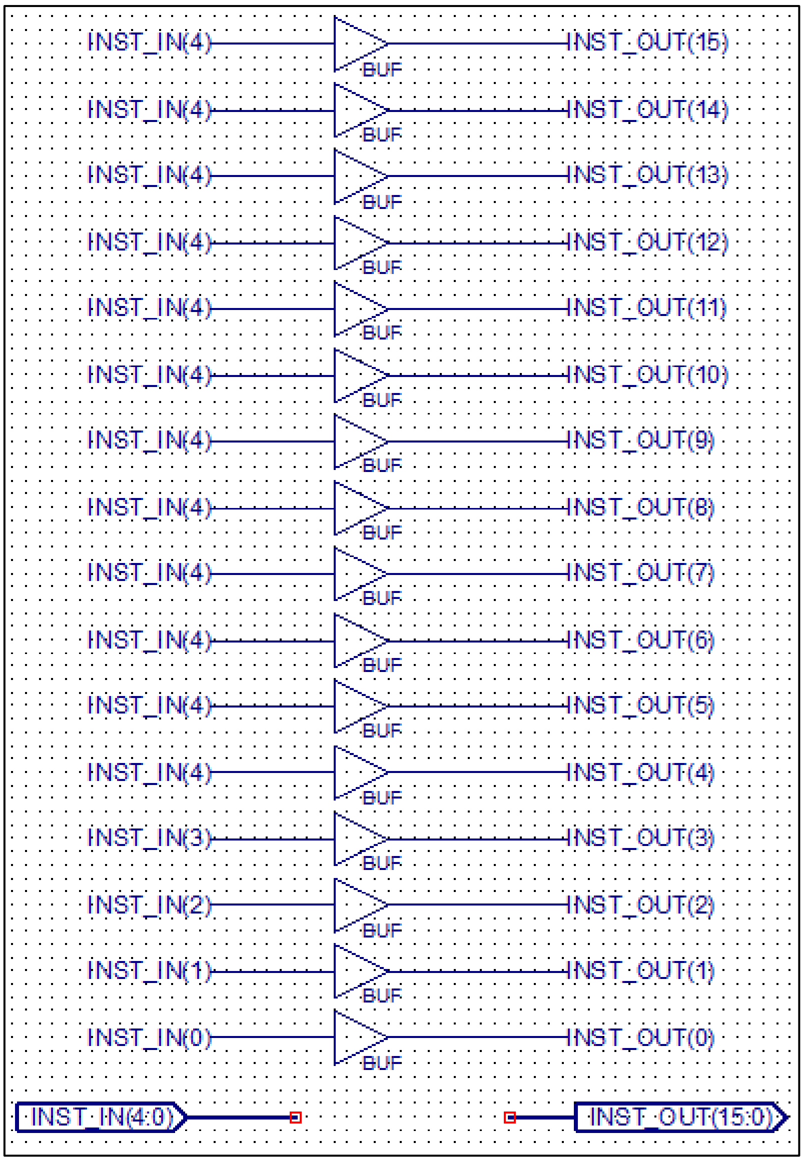
ImmediateGenerator(8to16)



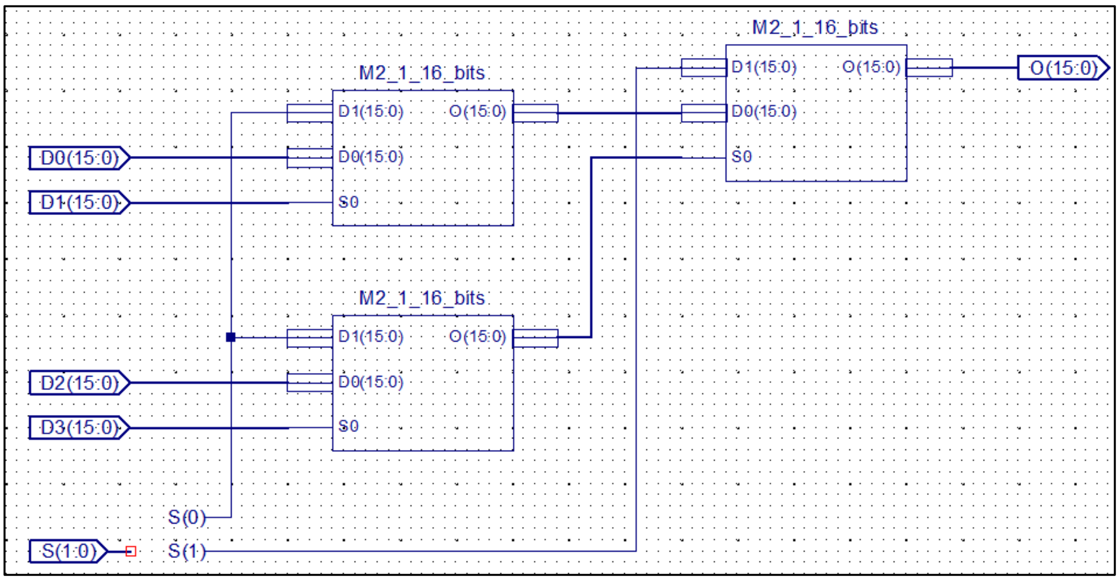
Zero extension



- ImmediateGenerator(5to16)



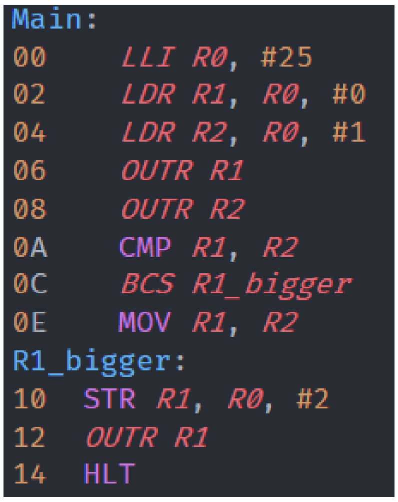
- 4 對 1 多工器(16bit)



Verification

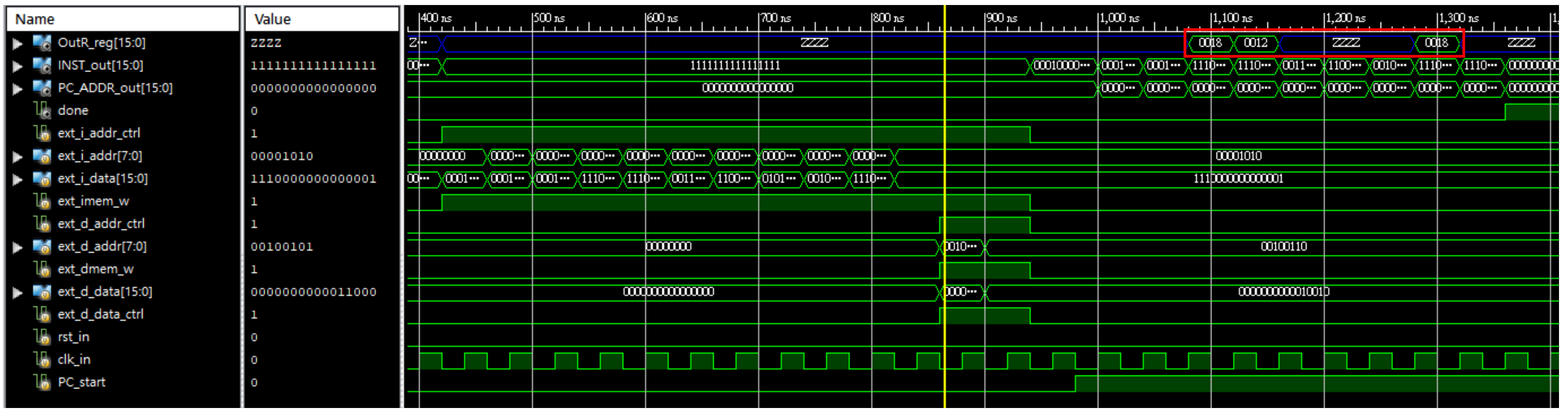
1. Find the minimum and maximum from two numbers in memory.

- Assembly code



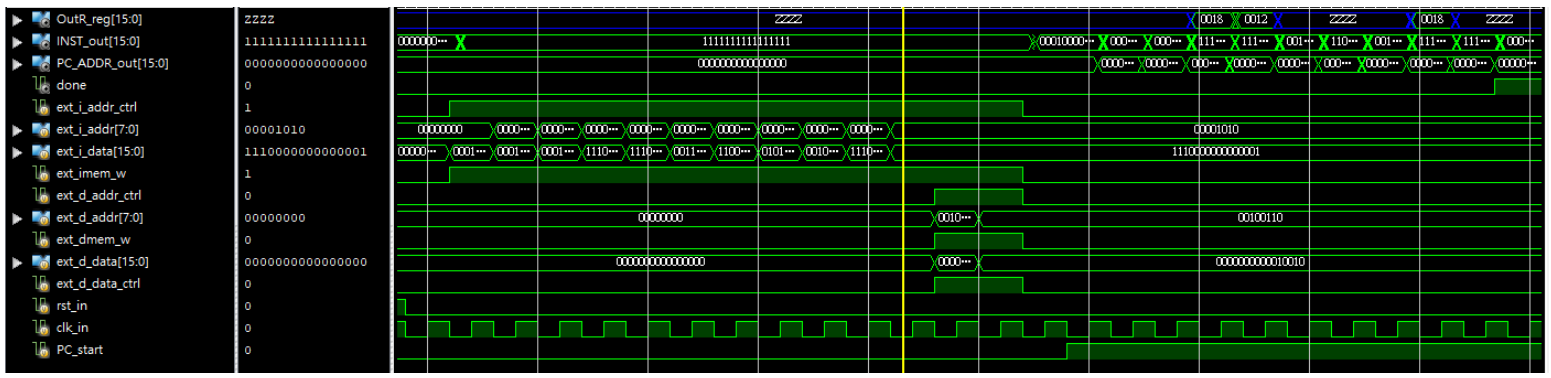
```
Main:
00      LLI R0, #25
02      LDR R1, R0, #0
04      LDR R2, R0, #1
06      OUTR R1
08      OUTR R2
0A      CMP R1, R2
0C      BCS R1_bigger
0E      MOV R1, R2
R1_bigger:
10   STR R1, R0, #2
12   OUTR R1
14   HLT
```

- Behavioral



Name	Value
OutR_reg[15:0]	ZZZZ
INST_out[15:0]	1111111111111111
PC_ADDR_out[15:0]	0000000000000000
done	0
ext_i_addr_ctrl	1
ext_i_addr[7:0]	00001010
ext_i_data[15:0]	1110000000000001
ext_imem_w	1
ext_d_addr_ctrl	1
ext_d_addr[7:0]	00100101
ext_dmem_w	1
ext_d_data[15:0]	0000000000011000
ext_d_data_ctrl	1
rst_in	0
clk_in	0
PC_start	0

- Post-Route

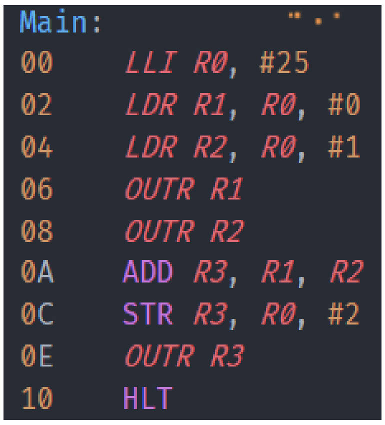


OutR_reg[15:0]	ZZZZ
INST_out[15:0]	1111111111111111
PC_ADDR_out[15:0]	0000000000000000
done	0
ext_i_addr_ctrl	1
ext_i_addr[7:0]	00001010
ext_i_data[15:0]	1110000000000001
ext_imem_w	1
ext_d_addr_ctrl	0
ext_d_addr[7:0]	00000000
ext_dmem_w	0
ext_d_data[15:0]	0000000000000000
ext_d_data_ctrl	0
rst_in	0
clk_in	0
PC_start	0

可以觀察到一開始記憶體裡面的值為 12H 和 18H，而在指令執行完後會找到最大值 18H 並輸出。

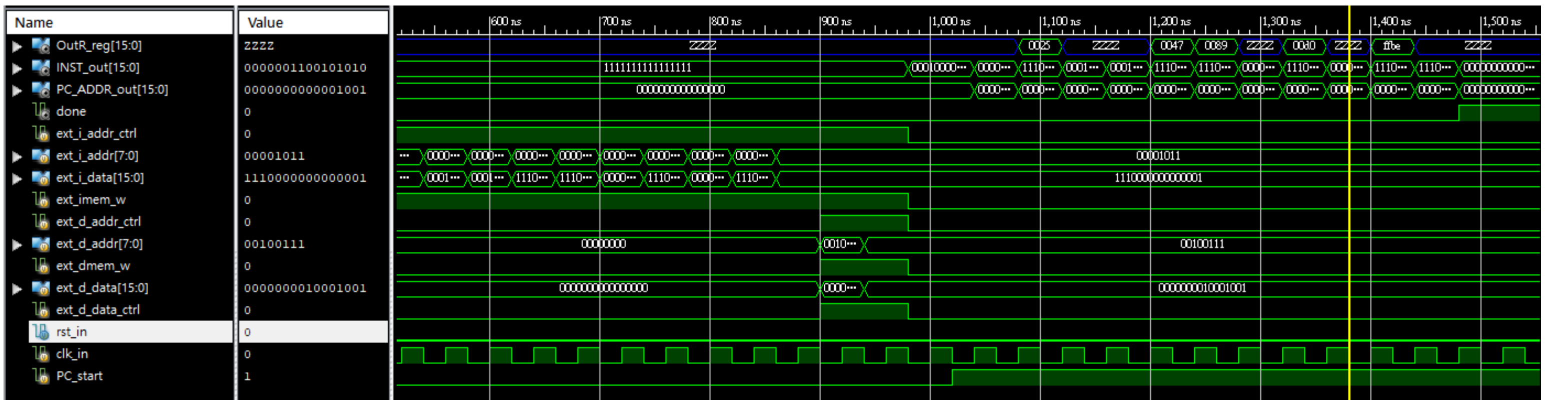
2. Add two numbers in memory and store the result in another memory location.

- Assembly code



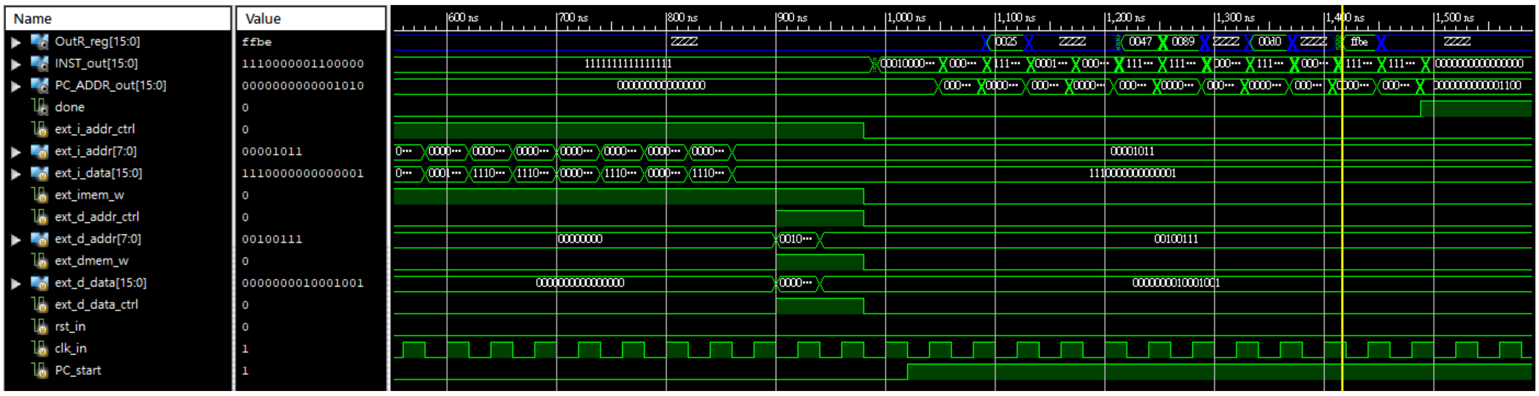
```
Main:
00      LLI R0, #25
02      LDR R1, R0, #0
04      LDR R2, R0, #1
06      OUTR R1
08      OUTR R2
0A      ADD R3, R1, R2
0C      STR R3, R0, #2
0E      OUTR R3
10      HLT
```

- Behavioral



Name	Value
OutR_reg[15:0]	zzzz
INST_out[15:0]	0000001100101010
PC_ADDR_out[15:0]	0000000000001001
done	0
ext_i_addr_ctrl	0
ext_i_addr[7:0]	00001011
ext_i_data[15:0]	1110000000000001
ext_imem_w	0
ext_d_addr_ctrl	0
ext_d_addr[7:0]	00100111
ext_dmem_w	0
ext_d_data[15:0]	0000000010001001
ext_d_data_ctrl	0
rst_in	0
clk_in	0
PC_start	1

- Post-Route

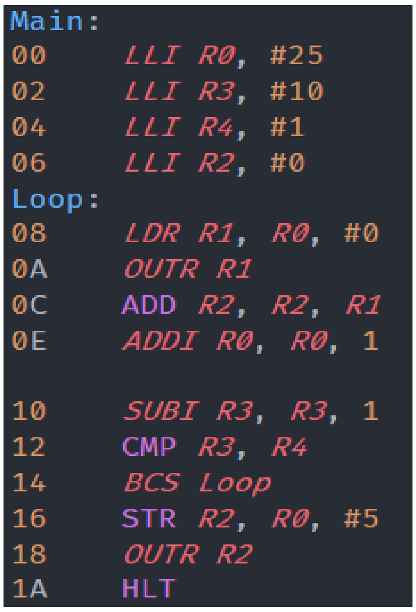


Name	Value
OutR_reg[15:0]	ffbe
INST_out[15:0]	1110000001100000
PC_ADDR_out[15:0]	0000000000001010
done	0
ext_i_addr_ctrl	0
ext_i_addr[7:0]	00001011
ext_i_data[15:0]	1110000000000001
ext_imem_w	0
ext_d_addr_ctrl	0
ext_d_addr[7:0]	00100111
ext_dmem_w	0
ext_d_data[15:0]	0000000010001001
ext_d_data_ctrl	0
rst_in	0
clk_in	1
PC_start	1

把 47H 存在記憶體位址 0025H，0089H 存在記憶體位址 0026H，將兩數相加得到結果 00D0H，相減得到結果 FFBEH。

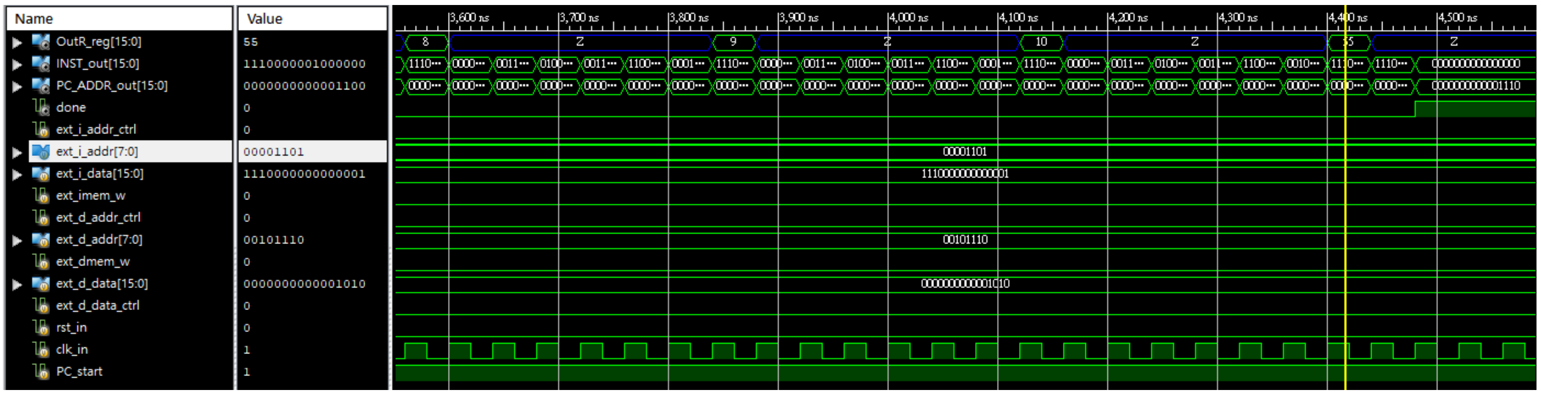
3. Add ten numbers in consecutive memory locations.

- Assembly code



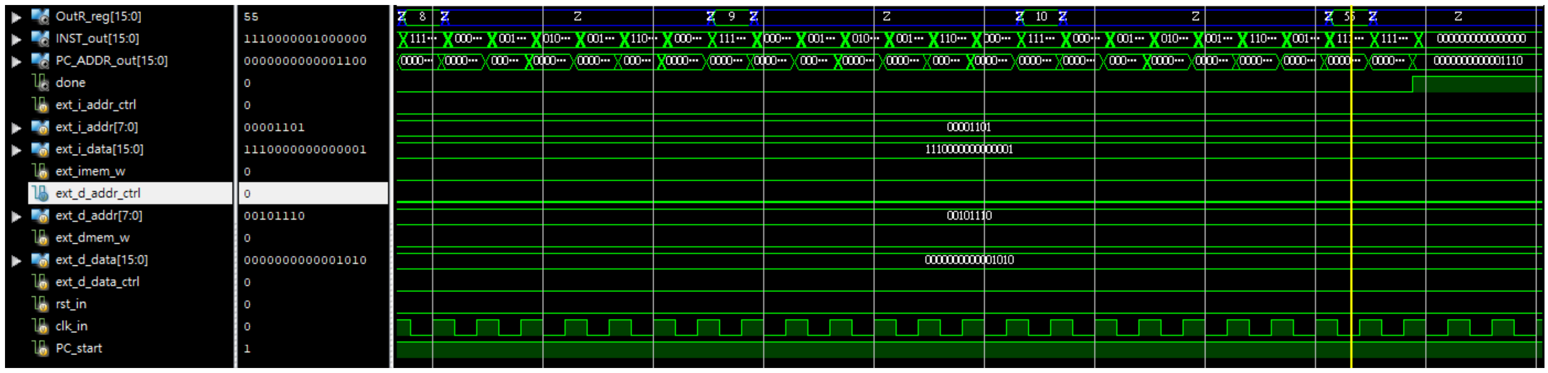
```
Main:
00      LLI R0, #25
02      LLI R3, #10
04      LLI R4, #1
06      LLI R2, #0
Loop:
08      LDR R1, R0, #0
0A      OUTR R1
0C      ADD R2, R2, R1
0E      ADDI R0, R0, 1

10      SUBI R3, R3, 1
12      CMP R3, R4
14      BCS Loop
16      STR R2, R0, #5
18      OUTR R2
1A      HLT
```

- Behavioral



Name	Value
OutR_reg[15:0]	55
INST_out[15:0]	1110000001000000
PC_ADDR_out[15:0]	0000000000001100
done	0
ext_i_addr_ctrl	0
ext_i_addr[7:0]	00001101
ext_i_data[15:0]	1110000000000001
ext_imem_w	0
ext_d_addr_ctrl	0
ext_d_addr[7:0]	00101110
ext_dmem_w	0
ext_d_data[15:0]	0000000000001010
ext_d_data_ctrl	0
rst_in	0
clk_in	1
PC_start	1

- Post-Route

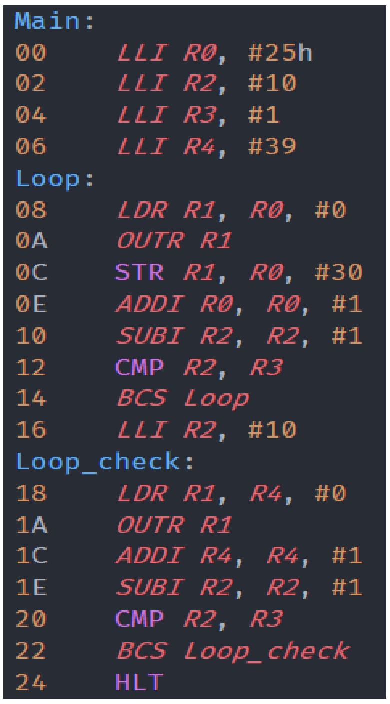


OutR_reg[15:0]	55
INST_out[15:0]	1110000001000000
PC_ADDR_out[15:0]	0000000000001100
done	0
ext_i_addr_ctrl	0
ext_i_addr[7:0]	00001101
ext_i_data[15:0]	1110000000000001
ext_imem_w	0
ext_d_addr_ctrl	0
ext_d_addr[7:0]	00101110
ext_dmem_w	0
ext_d_data[15:0]	0000000000001010
ext_d_data_ctrl	0
rst_in	0
clk_in	0
PC_start	1

存在記憶體的值為 1~10，可以看到結果會將每次運算的值輸出出來，也就是會計算(1+2+3…..9+10)=55，並且得到 55 的正確答案。

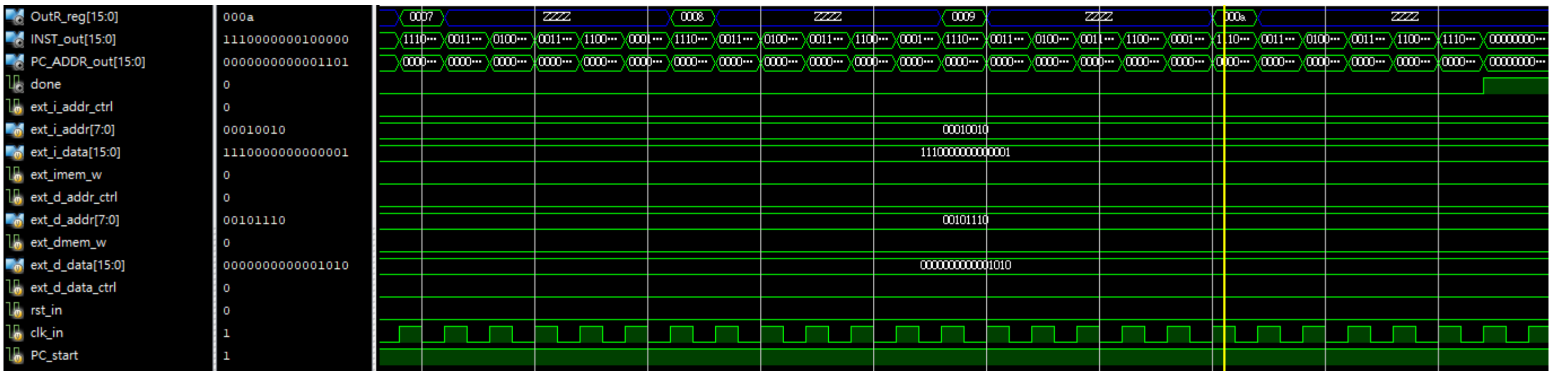
4. Mov a memory block of N words from one place to another.

- Assembly code



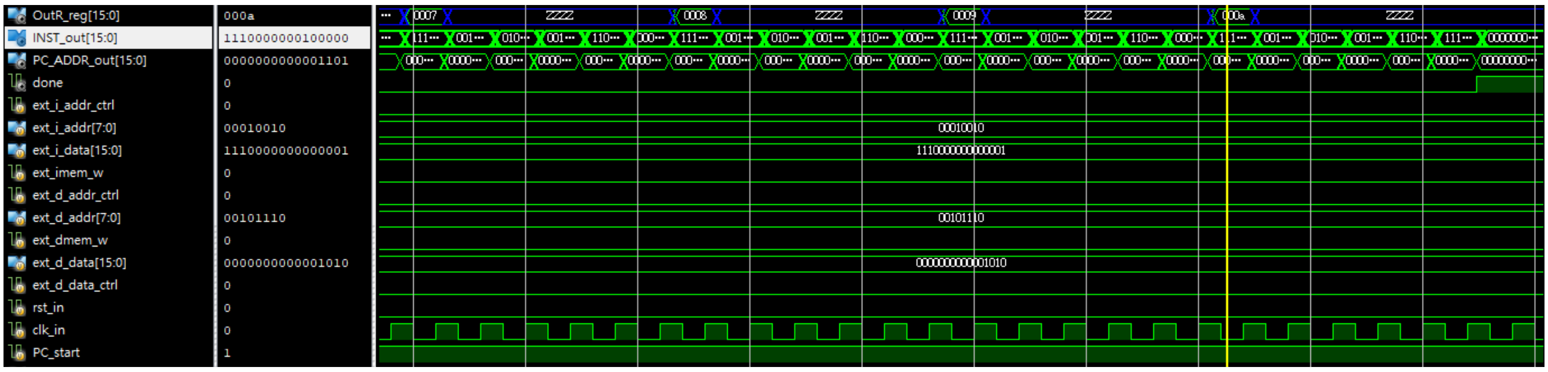
```
Main:
00      LLI R0, #25h
02      LLI R2, #10
04      LLI R3, #1
06      LLI R4, #39
Loop:
08      LDR R1, R0, #0
0A      OUTR R1
0C      STR R1, R0, #30
0E      ADDI R0, R0, #1
10      SUBI R2, R2, #1
12      CMP R2, R3
14      BCS Loop
16      LLI R2, #10
Loop_check:
18      LDR R1, R4, #0
1A      OUTR R1
1C      ADDI R4, R4, #1
1E      SUBI R2, R2, #1
20      CMP R2, R3
22      BCS Loop_check
24      HLT
```

- Behavioral



```
OutR_reg[15:0]      000a
INST_out[15:0]      1110000000100000
PC_ADDR_out[15:0]   0000000000001101
done                0
ext_i_addr_ctrl     0
ext_i_addr[7:0]     00010010
ext_i_data[15:0]    1110000000000001
ext_imem_w          0
ext_d_addr_ctrl     0
ext_d_addr[7:0]     00101110
ext_dmem_w          0
ext_d_data[15:0]    0000000000001010
ext_d_data_ctrl     0
rst_in              0
clk_in              1
PC_start            1
```

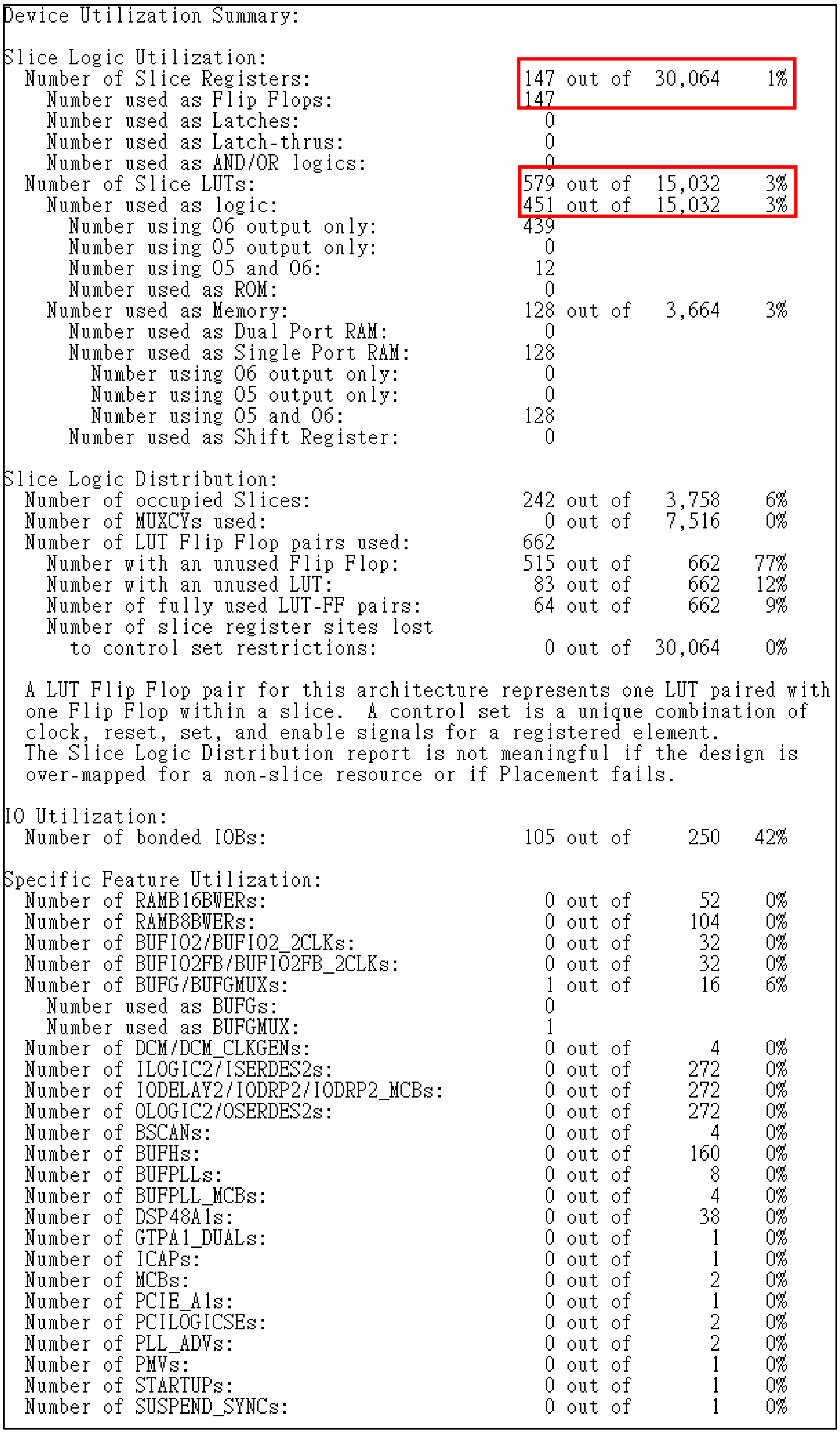
- Post-Route



```
OutR_reg[15:0]      000a
INST_out[15:0]      1110000000100000
PC_ADDR_out[15:0]   0000000000001101
done                0
ext_i_addr_ctrl     0
ext_i_addr[7:0]     00010010
ext_i_data[15:0]    1110000000000001
ext_imem_w          0
ext_d_addr_ctrl     0
ext_d_addr[7:0]     00101110
ext_dmem_w          0
ext_d_data[15:0]    0000000000001010
ext_d_data_ctrl     0
rst_in              0
clk_in              0
PC_start            1
```

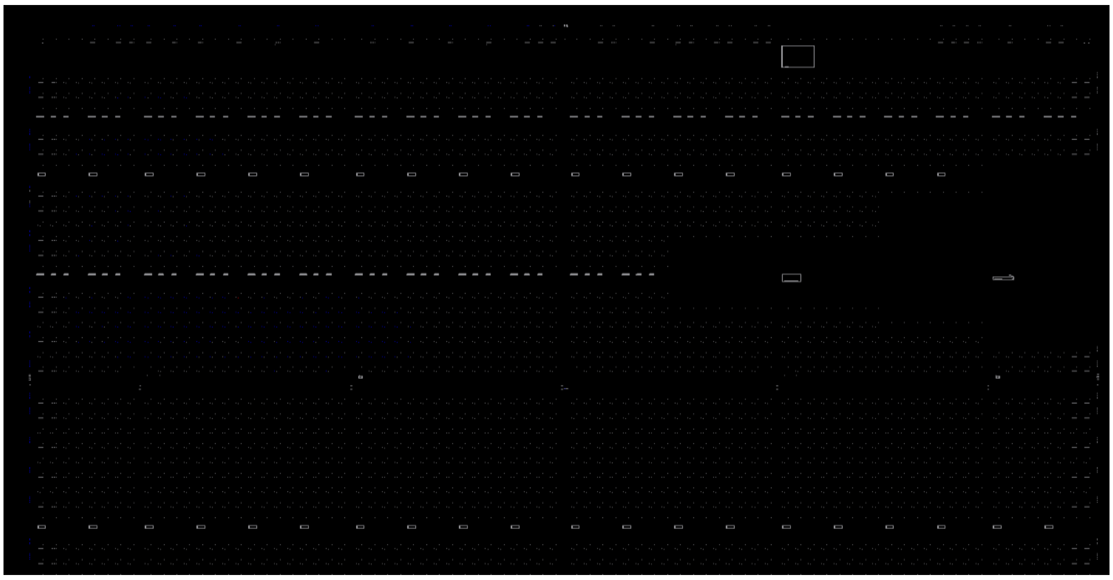
可以看到最後的輸出是 8…9….10，因為我們是將 memory 裡面的 1~10，搬到另一個記憶體位址，我們再搬完後去查看記憶體的位址裡面是否為 1~10，可以發現 move 成功。

Hardware

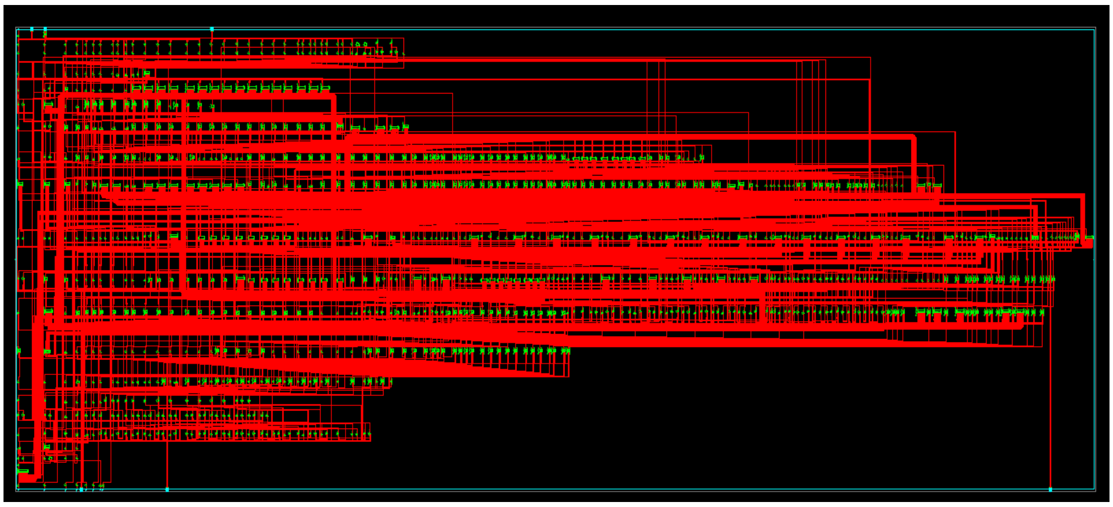


```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                   147 out of  30,064    1%
    Number used as Flip Flops:                 147
    Number used as Latches:                      0
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                        579 out of  15,032    3%
    Number used as logic:                      451 out of  15,032    3%
      Number using O6 output only:             439
      Number using O5 output only:               0
      Number using O5 and O6:                   12
      Number used as ROM:                        0
    Number used as Memory:                     128 out of   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

Slice Logic Distribution:
  Number of occupied Slices:                   242 out of   3,758    6%
  Number of MUXCYs used:                         0 out of   7,516    0%
  Number of LUT Flip Flop pairs used:          662
    Number with an unused Flip Flop:           515 out of     662   77%
    Number with an unused LUT:                  83 out of     662   12%
    Number of fully used LUT-FF pairs:          64 out of     662    9%
    Number of slice register sites lost
      to control set restrictions:               0 out of  30,064    0%

  A LUT Flip Flop pair for this architecture represents one LUT paired with
  one Flip Flop within a slice.  A control set is a unique combination of
  clock, reset, set, and enable signals for a registered element.
  The Slice Logic Distribution report is not meaningful if the design is
  over-mapped for a non-slice resource or if Placement fails.

IO Utilization:
  Number of bonded IOBs:                       105 out of     250   42%

Specific Feature Utilization:
  Number of RAMB16BWERs:                         0 out of      52    0%
  Number of RAMB8BWERs:                          0 out of     104    0%
  Number of BUFIO2/BUFIO2_2CLKs:                 0 out of      32    0%
  Number of BUFIO2FB/BUFIO2FB_2CLKs:             0 out of      32    0%
  Number of BUFG/BUFGMUXs:                       1 out of      16    6%
    Number used as BUFGs:                        0
    Number used as BUFGMUX:                      1
  Number of DCM/DCM_CLKGENs:                     0 out of       4    0%
  Number of ILOGIC2/ISERDES2s:                   0 out of     272    0%
  Number of IODELAY2/IODRP2/IODRP2_MCBs:         0 out of     272    0%
  Number of OLOGIC2/OSERDES2s:                   0 out of     272    0%
  Number of BSCANs:                              0 out of       4    0%
  Number of BUFHs:                               0 out of     160    0%
  Number of BUFPLLs:                             0 out of       8    0%
  Number of BUFPLL_MCBs:                         0 out of       4    0%
  Number of DSP48A1s:                            0 out of      38    0%
  Number of GTPA1_DUALs:                         0 out of       1    0%
  Number of ICAPs:                               0 out of       1    0%
  Number of MCBs:                                0 out of       2    0%
  Number of PCIE_A1s:                            0 out of       1    0%
  Number of PCILOGICSEs:                         0 out of       2    0%
  Number of PLL_ADVs:                            0 out of       2    0%
  Number of PMVs:                                0 out of       1    0%
  Number of STARTUPs:                            0 out of       1    0%
  Number of SUSPEND_SYNCs:                       0 out of       1    0%
```

FPGA VIEW



RTL VIEW:



Discussion

雖然之前有修過計算機組織，但是在經過這次的作業後，對於整個 CPU 的架構又更加的了解，困難的部分就是以前在寫 Verilog 時，可以直接對硬體行為做描述，但是畫 Schematic 時電路卻需要全部手動拉，所以腦中必須思考，我想要做這樣的動作，那我需要哪些硬體，因此需要更多的設計時間去想怎麼實現。另外，因為這次實作的 ISA 是之前較不熟悉的，因此在設計時，有時候會不了解這個 ISA 為什麼需要這個指令，在網路搜尋後，細細研讀，就會發現有些指令設計的精妙，也就是為什麼要如此設計，會有一種原來也可以這樣做的感覺，學習到很多，未來在設計硬體電路時，也不會一昧的亂打 Verilog，心中也會浮現那塊理想的電路。