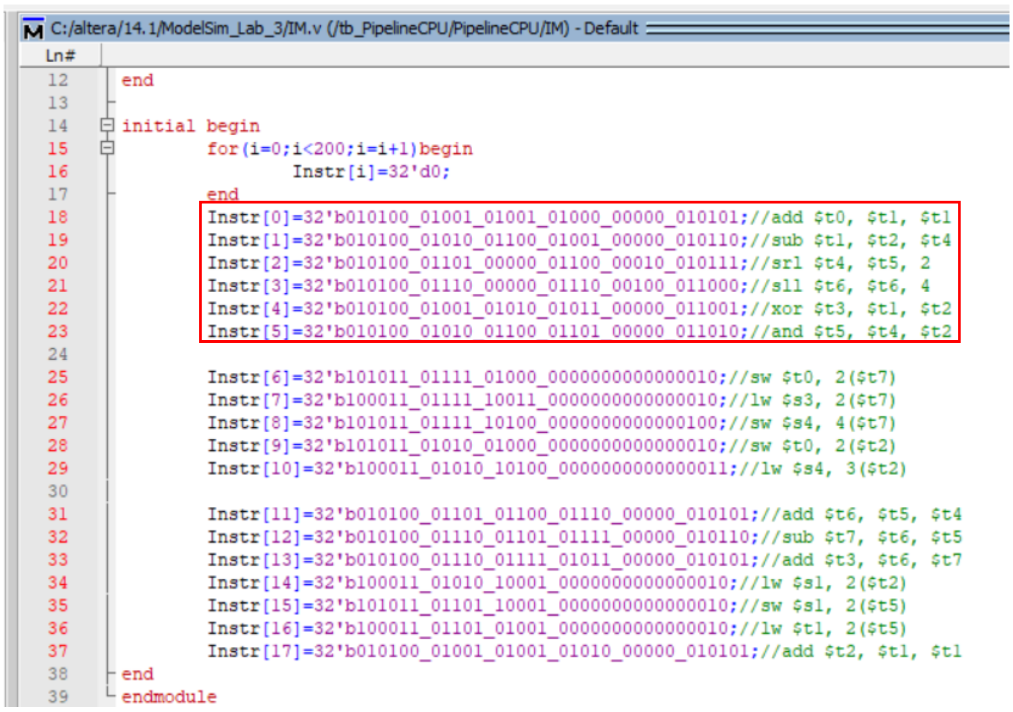
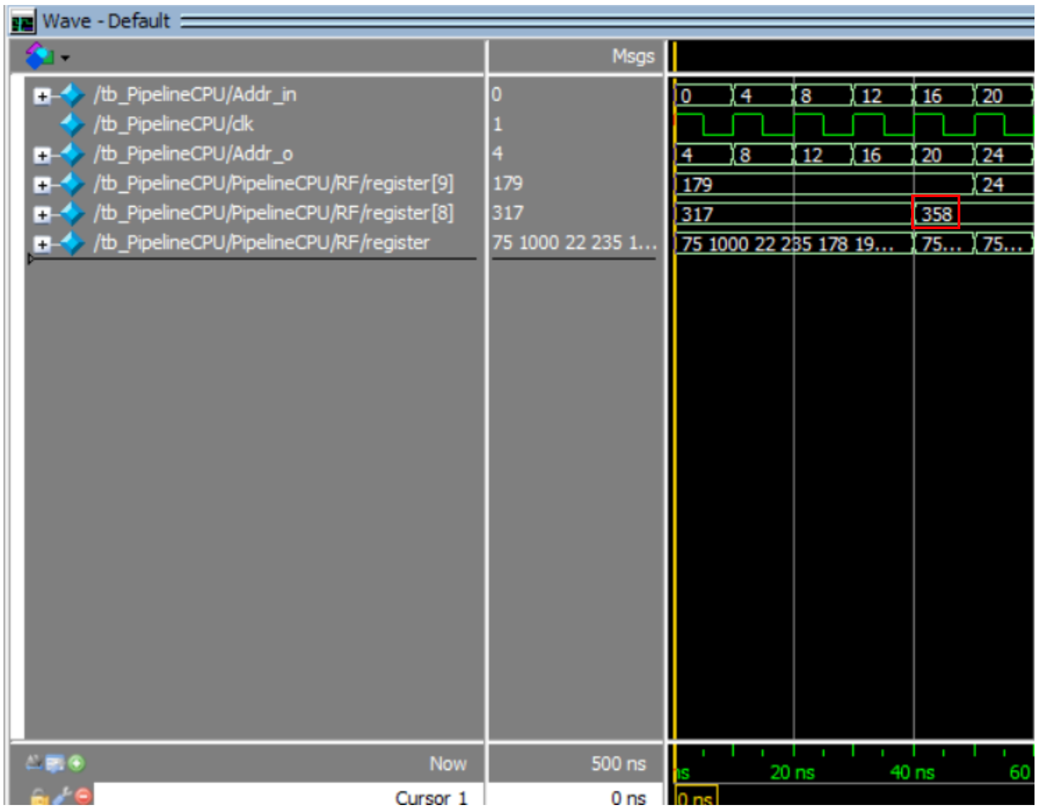
這次報告將會先展示模擬成果，最後再討論各項 module 程式碼內容

Part1: R-Format


C:/altera/14.1/ModelSim_Lab_3/IM.v (/tb_PipelineCPU/PipelineCPU/IM) - Default

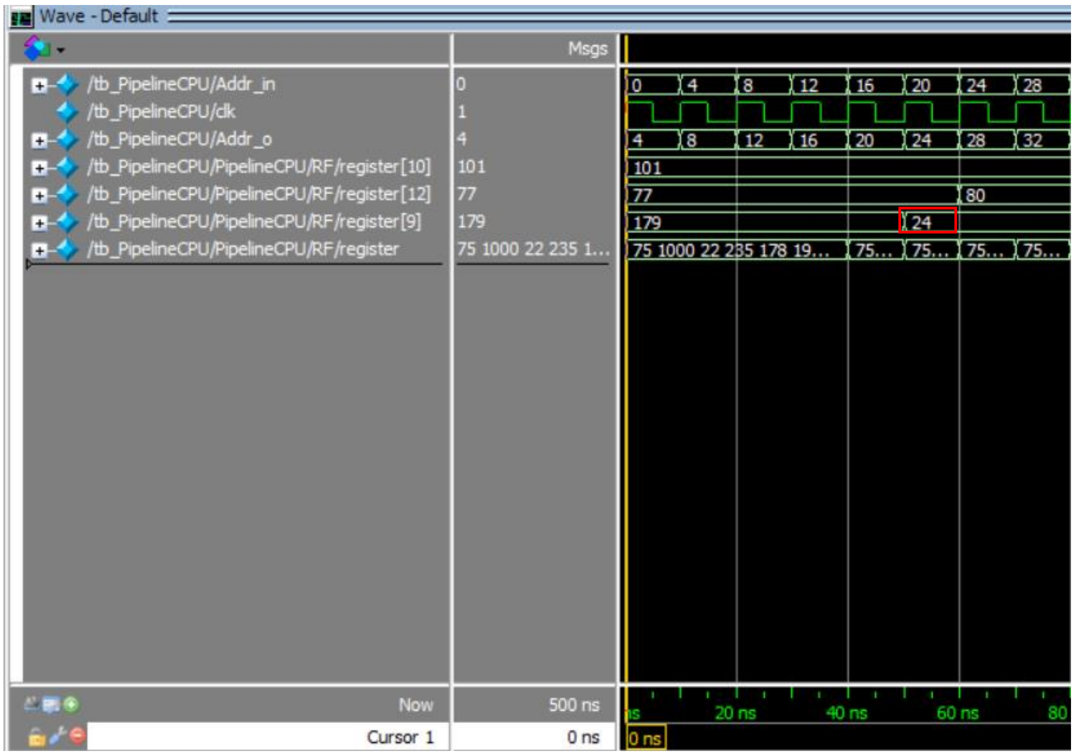
```
end

initial begin
        for(i=0;i<200;i=i+1)begin
                Instr[i]=32'd0;
        end
        Instr[0]=32'b010100_01001_01001_01000_00000_010101;//add $t0, $t1, $t1
        Instr[1]=32'b010100_01010_01100_01001_00000_010110;//sub $t1, $t2, $t4
        Instr[2]=32'b010100_01101_00000_01100_00010_010111;//srl $t4, $t5, 2
        Instr[3]=32'b010100_01110_00000_01110_00100_011000;//sll $t6, $t6, 4
        Instr[4]=32'b010100_01001_01010_01011_00000_011001;//xor $t3, $t1, $t2
        Instr[5]=32'b010100_01010_01100_01101_00000_011010;//and $t5, $t4, $t2

        Instr[6]=32'b101011_01111_01000_0000000000000010;//sw $t0, 2($t7)
        Instr[7]=32'b100011_01111_10011_0000000000000010;//lw $s3, 2($t7)
        Instr[8]=32'b101011_01111_10100_0000000000000100;//sw $s4, 4($t7)
        Instr[9]=32'b101011_01010_01000_0000000000000010;//sw $t0, 2($t2)
        Instr[10]=32'b100011_01010_10100_0000000000000011;//lw $s4, 3($t2)

        Instr[11]=32'b010100_01101_01100_01110_00000_010101;//add $t6, $t5, $t4
        Instr[12]=32'b010100_01110_01101_01111_00000_010110;//sub $t7, $t6, $t5
        Instr[13]=32'b010100_01110_01111_01011_00000_010101;//add $t3, $t6, $t7
        Instr[14]=32'b100011_01010_10001_0000000000000010;//lw $s1, 2($t2)
        Instr[15]=32'b101011_01101_10001_0000000000000010;//sw $s1, 2($t5)
        Instr[16]=32'b100011_01101_01001_0000000000000010;//lw $t1, 2($t5)
        Instr[17]=32'b010100_01001_01001_01010_00000_010101;//add $t2, $t1, $t1
end
endmodule
```

1. Instr[0]=>$t0($8)=358=$t1(179)+$t1(179)

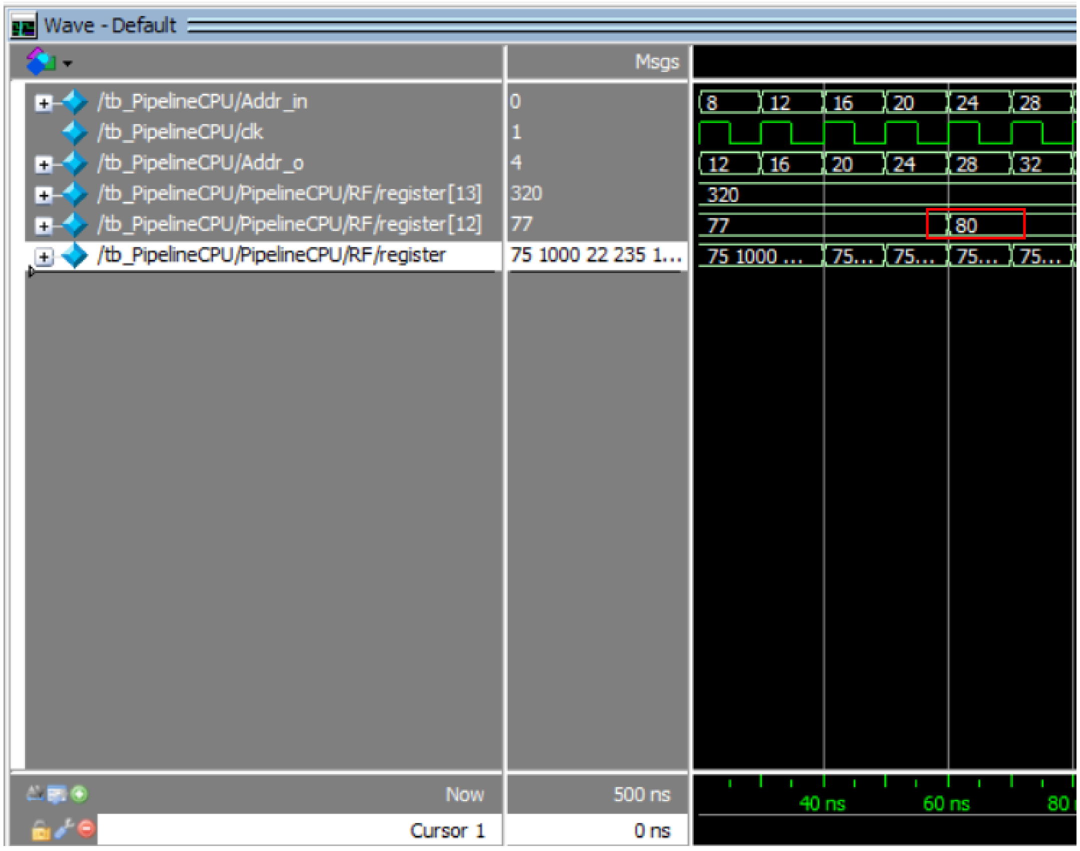

Wave - Default

```
                                           Msgs
/tb_PipelineCPU/Addr_in                    0                  0 4 8 12 16 20
/tb_PipelineCPU/clk                        1
/tb_PipelineCPU/Addr_o                     4                  4 8 12 16 20 24
/tb_PipelineCPU/PipelineCPU/RF/register[9] 179                179 24
/tb_PipelineCPU/PipelineCPU/RF/register[8] 317                317 358
/tb_PipelineCPU/PipelineCPU/RF/register    75 1000 22 235 1...  75 1000 22 235 178 19... 75... 75...
Now                                        500 ns             20 ns   40 ns   60
Cursor 1                                   0 ns               0 ns
```

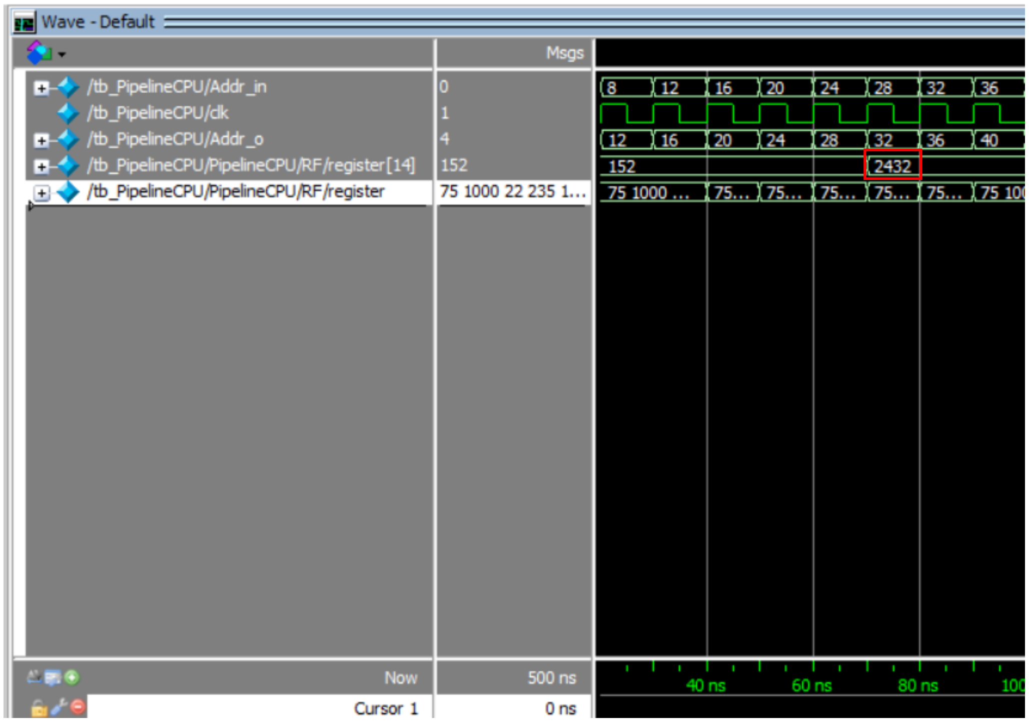
2. Instr[1]=>$t1($9)=24=$t2(101)-$t4(77)



3. Instr[2]=>$t4($12)=80=$t5(320)/4

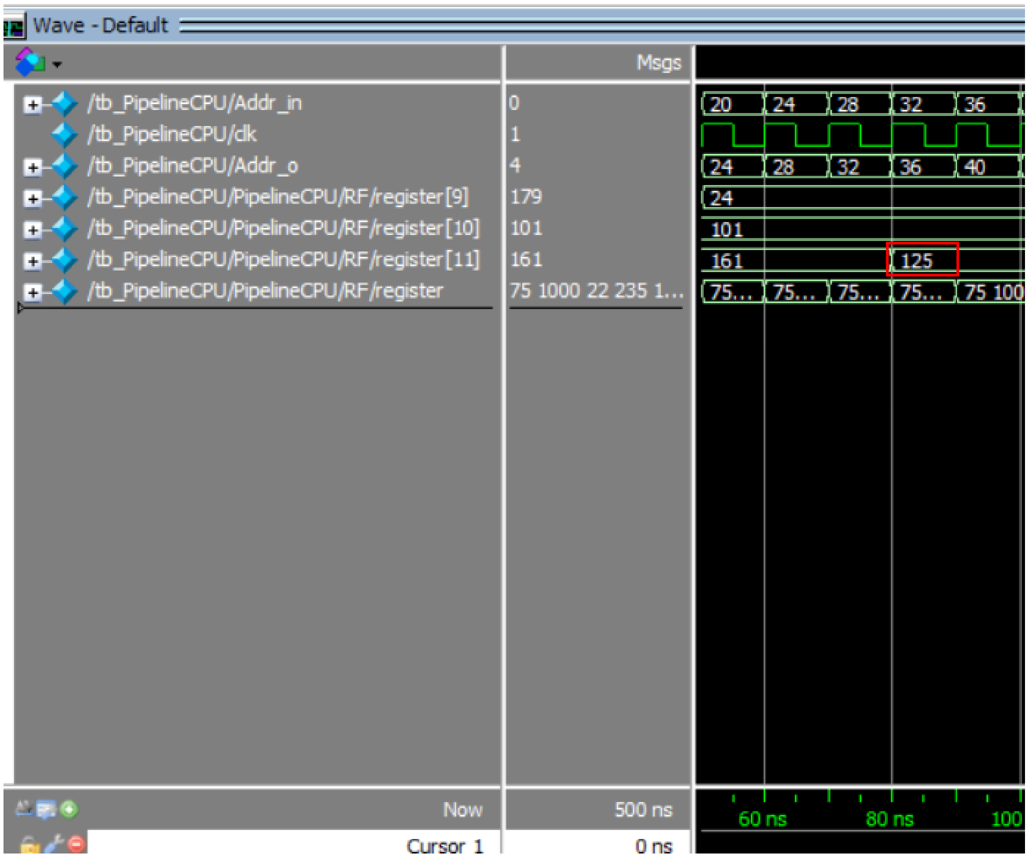


4. Instr[3]=>$t6($14)=2432=$t6(152)*16



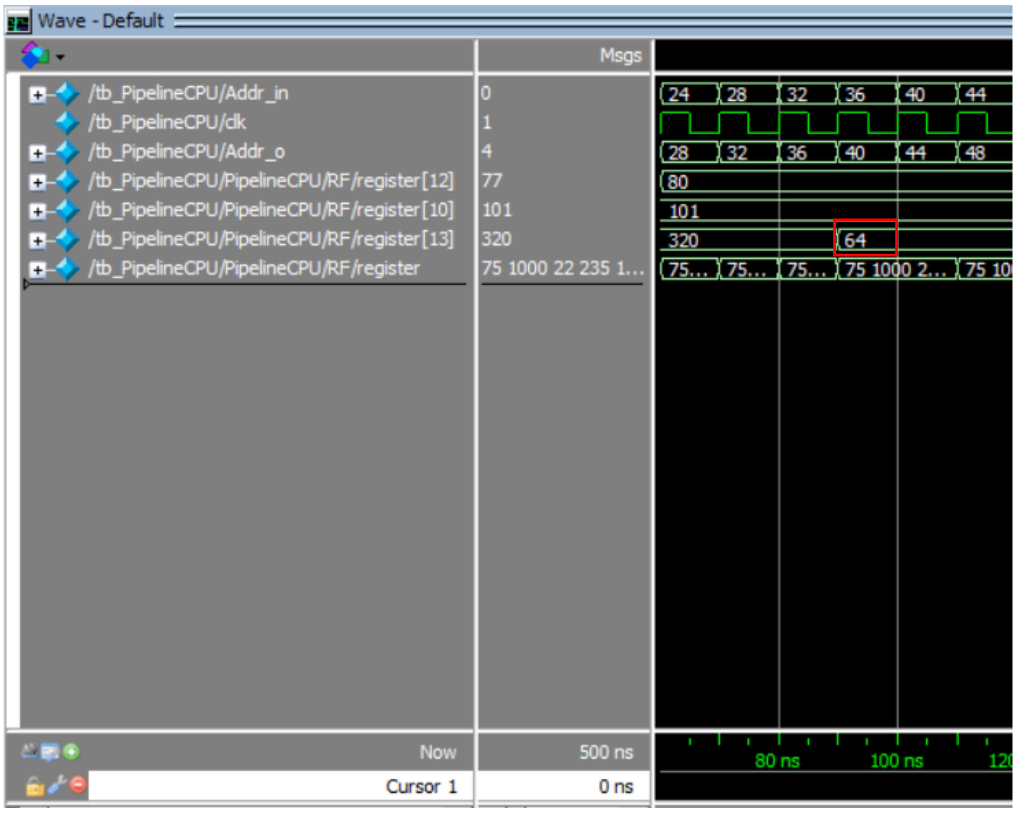
```
/tb_PipelineCPU/Addr_in                          0                 8   12  16  20  24  28  32  36
/tb_PipelineCPU/clk                              1
/tb_PipelineCPU/Addr_o                           4                 12  16  20  24  28  32  36  40
/tb_PipelineCPU/PipelineCPU/RF/register[14]      152               152                 2432
/tb_PipelineCPU/PipelineCPU/RF/register          75 1000 22 235 1...   75 1000 ...  75...  75...  75...  75...  75...  75 100
Now                                              500 ns
Cursor 1                                         0 ns
```

5. Instr[4]=>$t3($11)=125(01111101)=$t1(00011000)XOR$t2(01100101)



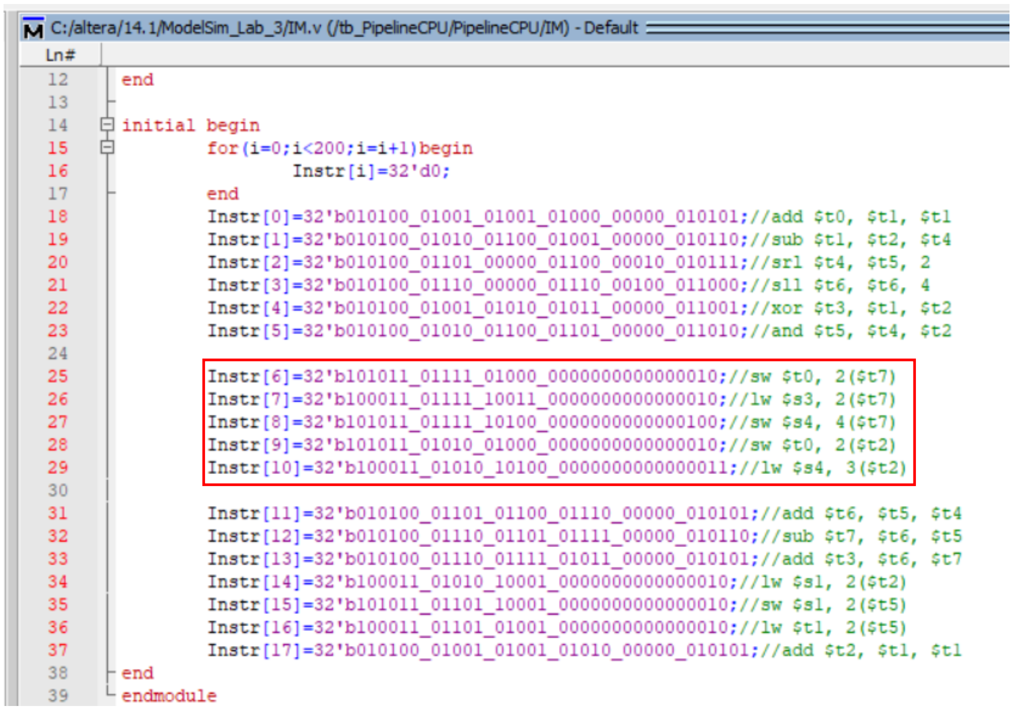
```
/tb_PipelineCPU/Addr_in                          0                 20  24  28  32  36
/tb_PipelineCPU/clk                              1
/tb_PipelineCPU/Addr_o                           4                 24  28  32  36  40
/tb_PipelineCPU/PipelineCPU/RF/register[9]       179               24
/tb_PipelineCPU/PipelineCPU/RF/register[10]      101               101
/tb_PipelineCPU/PipelineCPU/RF/register[11]      161               161         125
/tb_PipelineCPU/PipelineCPU/RF/register          75 1000 22 235 1...   75...  75...  75...  75...  75 100
Now                                              500 ns
Cursor 1                                         0 ns
```

6. Instr[5]=>$t5($13)=64(01000000)=$t4(01010000)AND$t2(01100101)



```
/tb_PipelineCPU/Addr_in                      0                  24 28 32 36 40 44
/tb_PipelineCPU/clk                          1
/tb_PipelineCPU/Addr_o                       4                  28 32 36 40 44 48
/tb_PipelineCPU/PipelineCPU/RF/register[12]  77                 80
/tb_PipelineCPU/PipelineCPU/RF/register[10]  101                101
/tb_PipelineCPU/PipelineCPU/RF/register[13]  320                320 64
/tb_PipelineCPU/PipelineCPU/RF/register      75 1000 22 235 1…  75… 75… 75… 75 1000 2… 75 10
```

Part2: LW 、SW

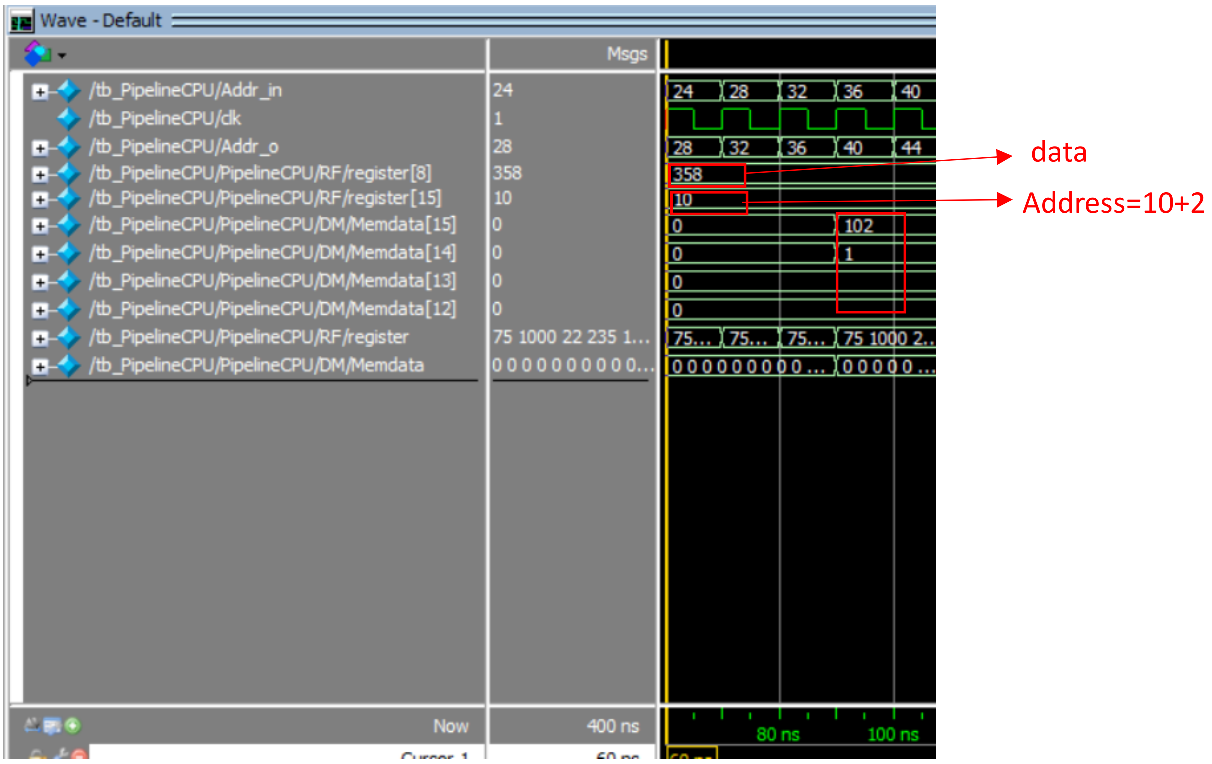


C:/altera/14.1/ModelSim_Lab_3/IM.v (/tb_PipelineCPU/PipelineCPU/IM) - Default

```
end

initial begin
        for(i=0;i<200;i=i+1)begin
                Instr[i]=32'd0;
        end
        Instr[0]=32'b010100_01001_01001_01000_00000_010101;//add $t0, $t1, $t1
        Instr[1]=32'b010100_01010_01100_01001_00000_010110;//sub $t1, $t2, $t4
        Instr[2]=32'b010100_01101_00000_01100_00010_010111;//srl $t4, $t5, 2
        Instr[3]=32'b010100_01110_00000_01110_00100_011000;//sll $t6, $t6, 4
        Instr[4]=32'b010100_01001_01010_01011_00000_011001;//xor $t3, $t1, $t2
        Instr[5]=32'b010100_01010_01100_01101_00000_011010;//and $t5, $t4, $t2

        Instr[6]=32'b101011_01111_01000_0000000000000010;//sw $t0, 2($t7)
        Instr[7]=32'b100011_01111_10011_0000000000000010;//lw $s3, 2($t7)
        Instr[8]=32'b101011_01111_10100_0000000000000100;//sw $s4, 4($t7)
        Instr[9]=32'b101011_01010_01000_0000000000000010;//sw $t0, 2($t2)
        Instr[10]=32'b100011_01010_10100_0000000000000011;//lw $s4, 3($t2)

        Instr[11]=32'b010100_01101_01100_01110_00000_010101;//add $t6, $t5, $t4
        Instr[12]=32'b010100_01110_01101_01111_00000_010110;//sub $t7, $t6, $t5
        Instr[13]=32'b010100_01110_01111_01011_00000_010101;//add $t3, $t6, $t7
        Instr[14]=32'b100011_01010_10001_0000000000000010;//lw $s1, 2($t2)
        Instr[15]=32'b101011_01101_10001_0000000000000010;//sw $s1, 2($t5)
        Instr[16]=32'b100011_01101_01001_0000000000000010;//lw $t1, 2($t5)
        Instr[17]=32'b010100_01001_01001_01010_00000_010101;//add $t2, $t1, $t1
end
endmodule
```

1.Instr[6]=>{Memdata[12],Memdata[13],Memdata[14],Memdata[15]}

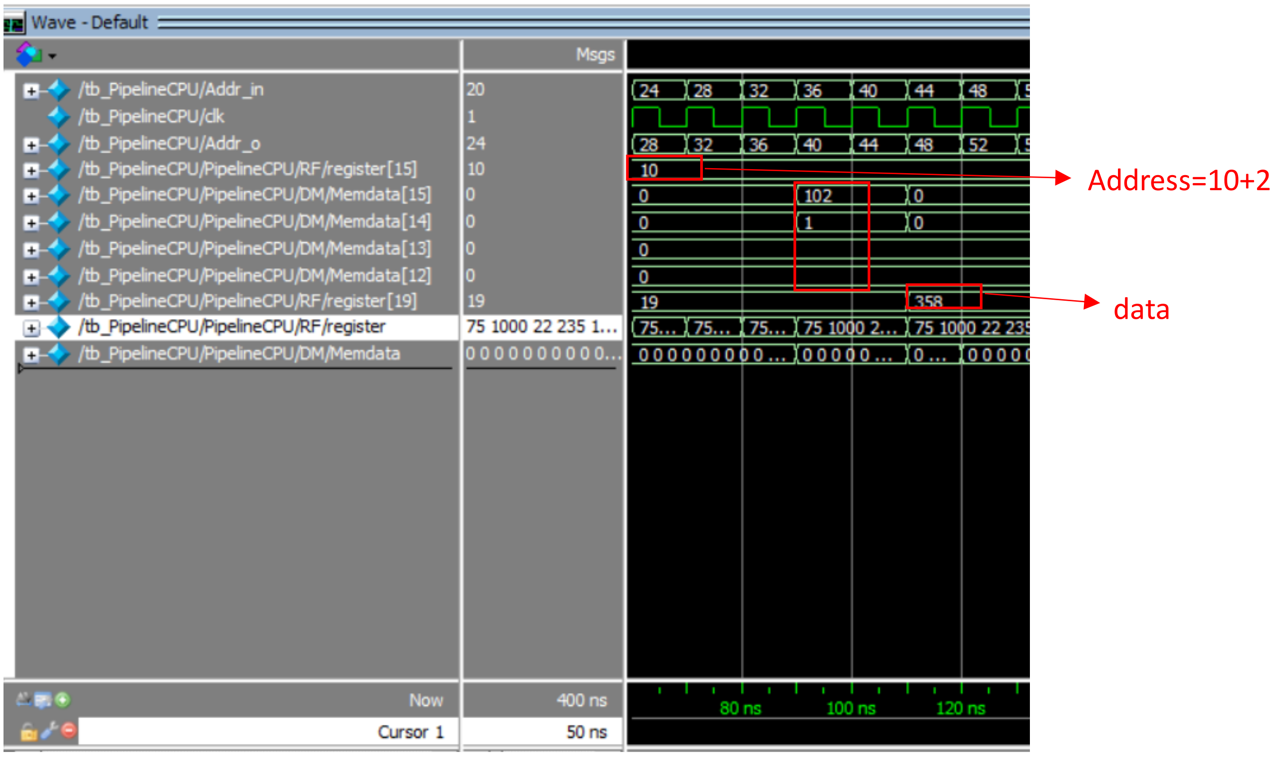
={8’d0,8’d0,8’b00000001,8’b01100110}(358)



Wave - Default

```
                                              Msgs
/tb_PipelineCPU/Addr_in                       24                 24  28  32  36  40
/tb_PipelineCPU/clk                           1
/tb_PipelineCPU/Addr_o                        28                 28  32  36  40  44
/tb_PipelineCPU/PipelineCPU/RF/register[8]    358                358
/tb_PipelineCPU/PipelineCPU/RF/register[15]   10                 10
/tb_PipelineCPU/PipelineCPU/DM/Memdata[15]    0                  0           102
/tb_PipelineCPU/PipelineCPU/DM/Memdata[14]    0                  0           1
/tb_PipelineCPU/PipelineCPU/DM/Memdata[13]    0                  0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[12]    0                  0
/tb_PipelineCPU/PipelineCPU/RF/register       75 1000 22 235 1...  75...  75...  75...  75 1000 2...
/tb_PipelineCPU/PipelineCPU/DM/Memdata        0 0 0 0 0 0 0 0 0...  0 0 0 0 0 0 0 0 ...  0 0 0 0 0 ...
Now                                           400 ns             80 ns   100 ns
Cursor 1                                      60 ns
```

2.Instr[7]=>$s3($19)={Memdata[12],Memdata[13],Memdata[14],Memdata[15]}

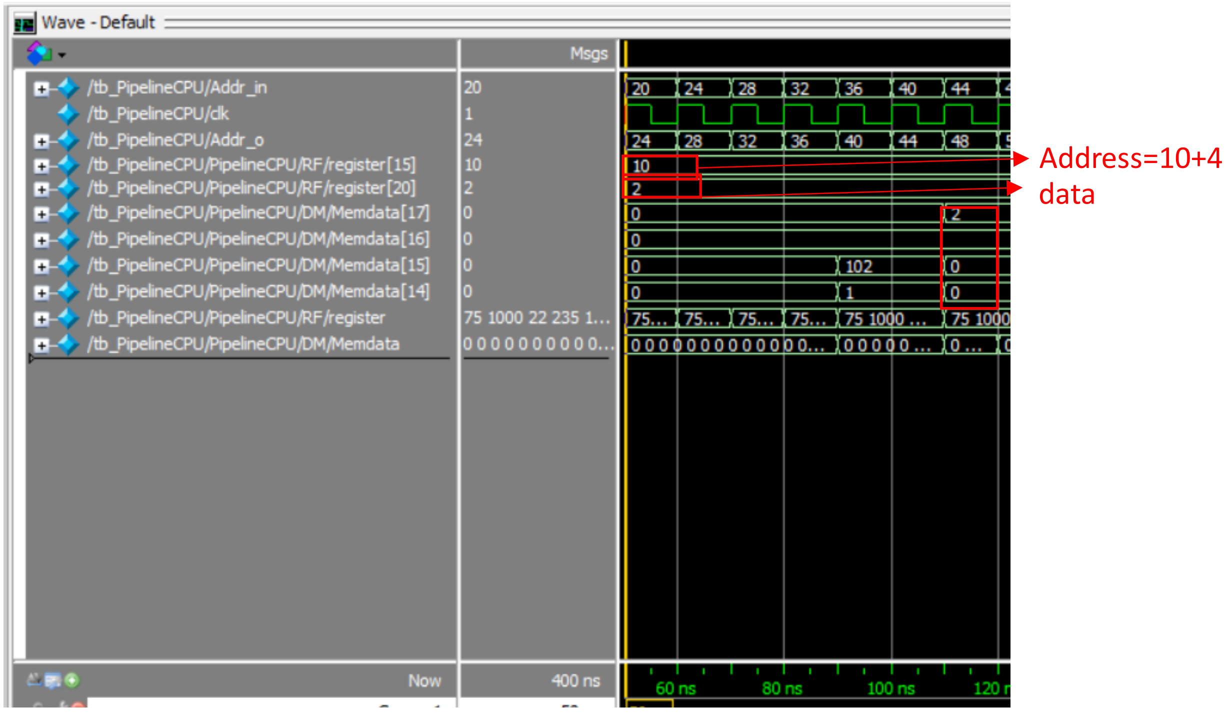

Wave - Default

	Msgs
/tb_PipelineCPU/Addr_in	20
/tb_PipelineCPU/clk	1
/tb_PipelineCPU/Addr_o	24
/tb_PipelineCPU/PipelineCPU/RF/register[15]	10
/tb_PipelineCPU/PipelineCPU/DM/Memdata[15]	0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[14]	0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[13]	0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[12]	0
/tb_PipelineCPU/PipelineCPU/RF/register[19]	19
/tb_PipelineCPU/PipelineCPU/RF/register	75 1000 22 235 1...
/tb_PipelineCPU/PipelineCPU/DM/Memdata	0 0 0 0 0 0 0 0 0 0...

Now 400 ns
Cursor 1 50 ns

3.Instr[8]=> {Memdata[14],Memdata[15],Memdata[16],Memdata[17]}

={8'd0,8'd0,8'd0,8'b00000010}(2)

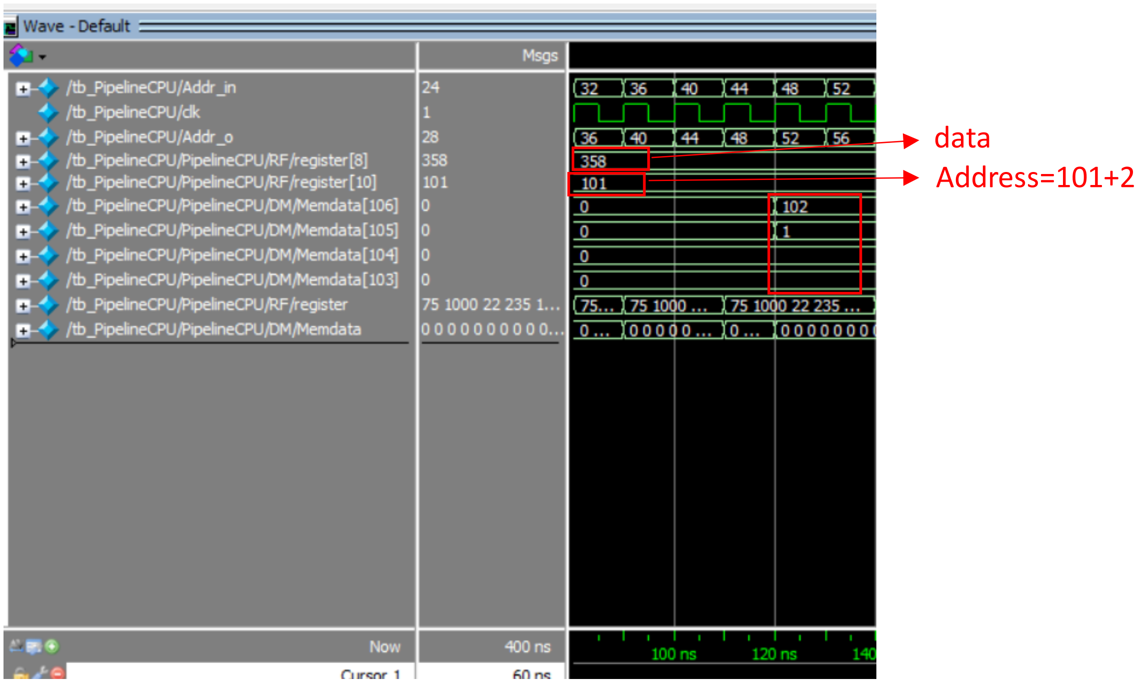

Wave - Default

	Msgs
/tb_PipelineCPU/Addr_in	20
/tb_PipelineCPU/clk	1
/tb_PipelineCPU/Addr_o	24
/tb_PipelineCPU/PipelineCPU/RF/register[15]	10
/tb_PipelineCPU/PipelineCPU/RF/register[20]	2
/tb_PipelineCPU/PipelineCPU/DM/Memdata[17]	0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[16]	0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[15]	0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[14]	0
/tb_PipelineCPU/PipelineCPU/RF/register	75 1000 22 235 1...
/tb_PipelineCPU/PipelineCPU/DM/Memdata	0 0 0 0 0 0 0 0 0 0...

Now 400 ns

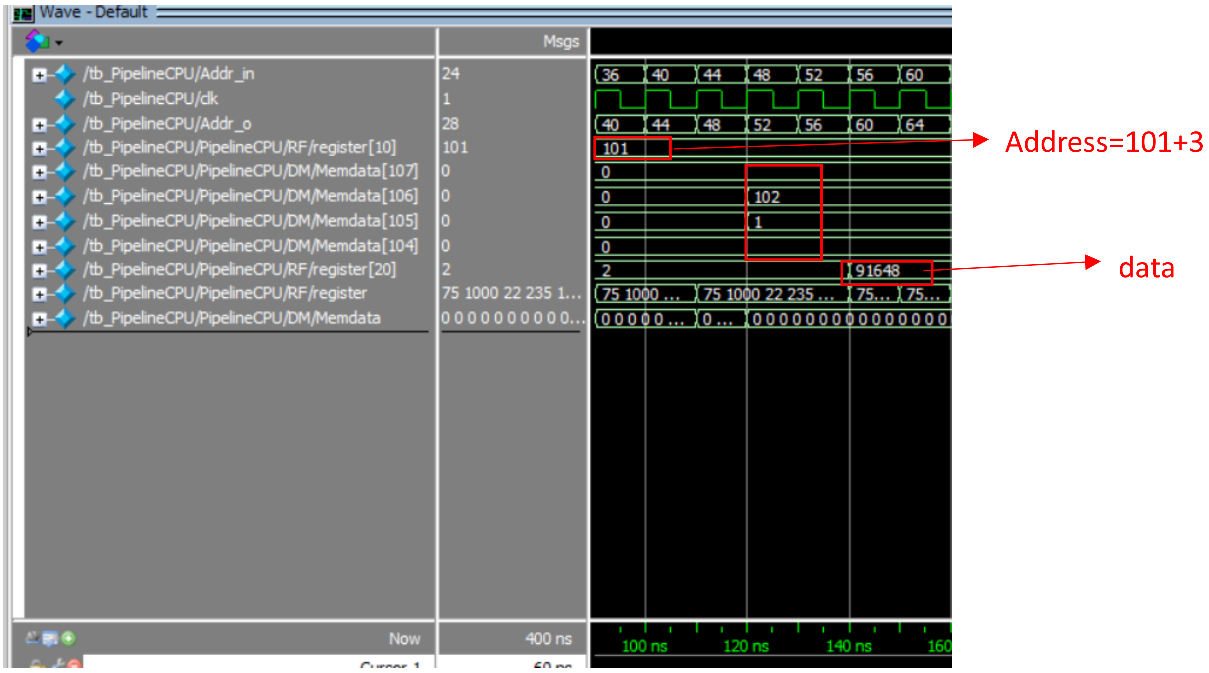
4.Instr[9]=> {Memdata[103],Memdata[104],Memdata[105],Memdata[106]}

={8’d0,8’d0,8’b00000001,8’b01100110}(358)



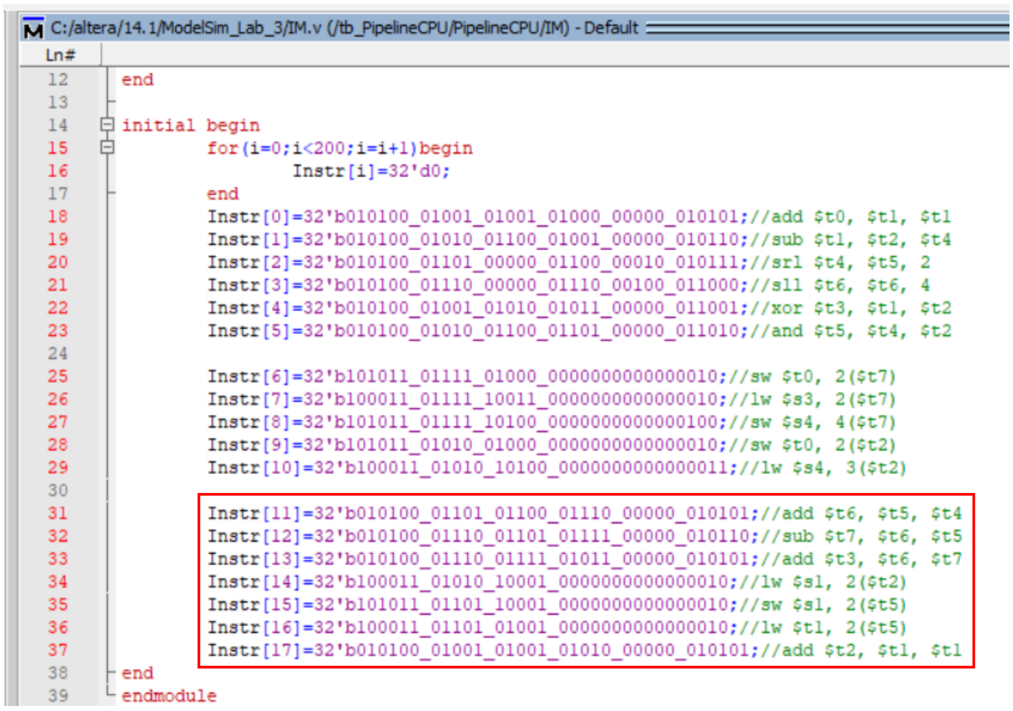
```
                                              Msgs
/tb_PipelineCPU/Addr_in                       24
/tb_PipelineCPU/clk                           1
/tb_PipelineCPU/Addr_o                        28
/tb_PipelineCPU/PipelineCPU/RF/register[8]    358
/tb_PipelineCPU/PipelineCPU/RF/register[10]   101
/tb_PipelineCPU/PipelineCPU/DM/Memdata[106]   0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[105]   0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[104]   0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[103]   0
/tb_PipelineCPU/PipelineCPU/RF/register       75 1000 22 235 1...
/tb_PipelineCPU/PipelineCPU/DM/Memdata        0 0 0 0 0 0 0 0 0 0...
Now                                           400 ns
Cursor 1                                      60 ns
```

5.Instr[10]=>$s4($20)={Memdata[104],Memdata[105],Memdata[106],Memdata[107]



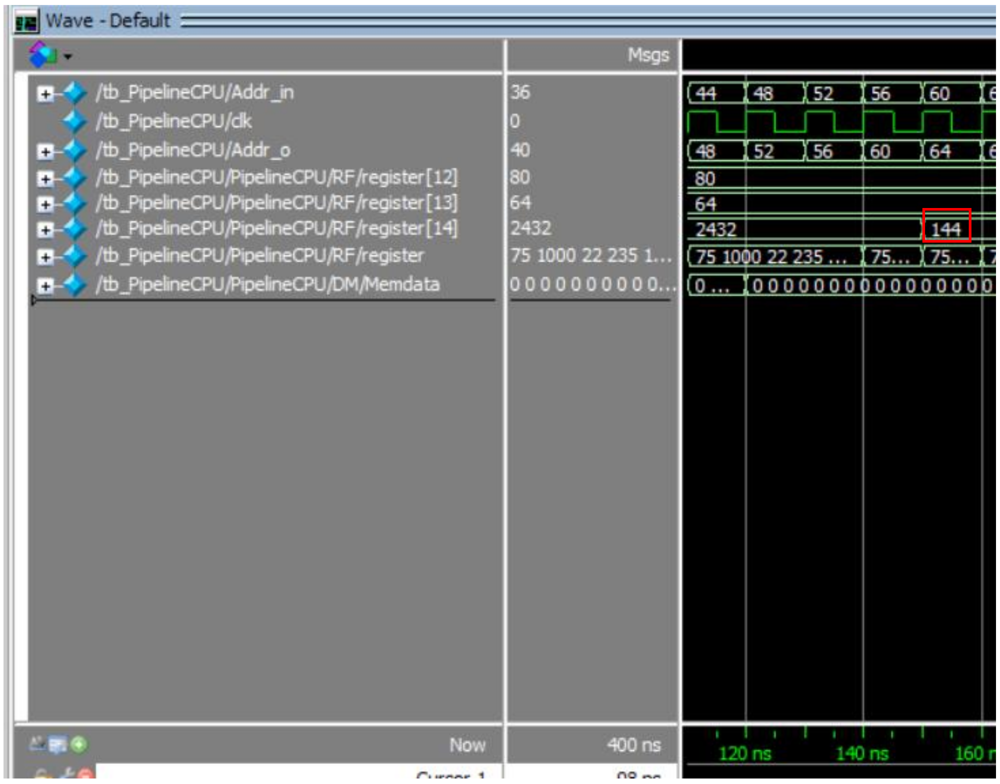
```
                                              Msgs
/tb_PipelineCPU/Addr_in                       24
/tb_PipelineCPU/clk                           1
/tb_PipelineCPU/Addr_o                        28
/tb_PipelineCPU/PipelineCPU/RF/register[10]   101
/tb_PipelineCPU/PipelineCPU/DM/Memdata[107]   0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[106]   0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[105]   0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[104]   0
/tb_PipelineCPU/PipelineCPU/RF/register[20]   2
/tb_PipelineCPU/PipelineCPU/RF/register       75 1000 22 235 1...
/tb_PipelineCPU/PipelineCPU/DM/Memdata        0 0 0 0 0 0 0 0 0 0...
Now                                           400 ns
Cursor 1                                      60 ns
```

Part3:Forwarding & Hazard


C:/altera/14.1/ModelSim_Lab_3/IM.v (/tb_PipelineCPU/PipelineCPU/IM) - Default

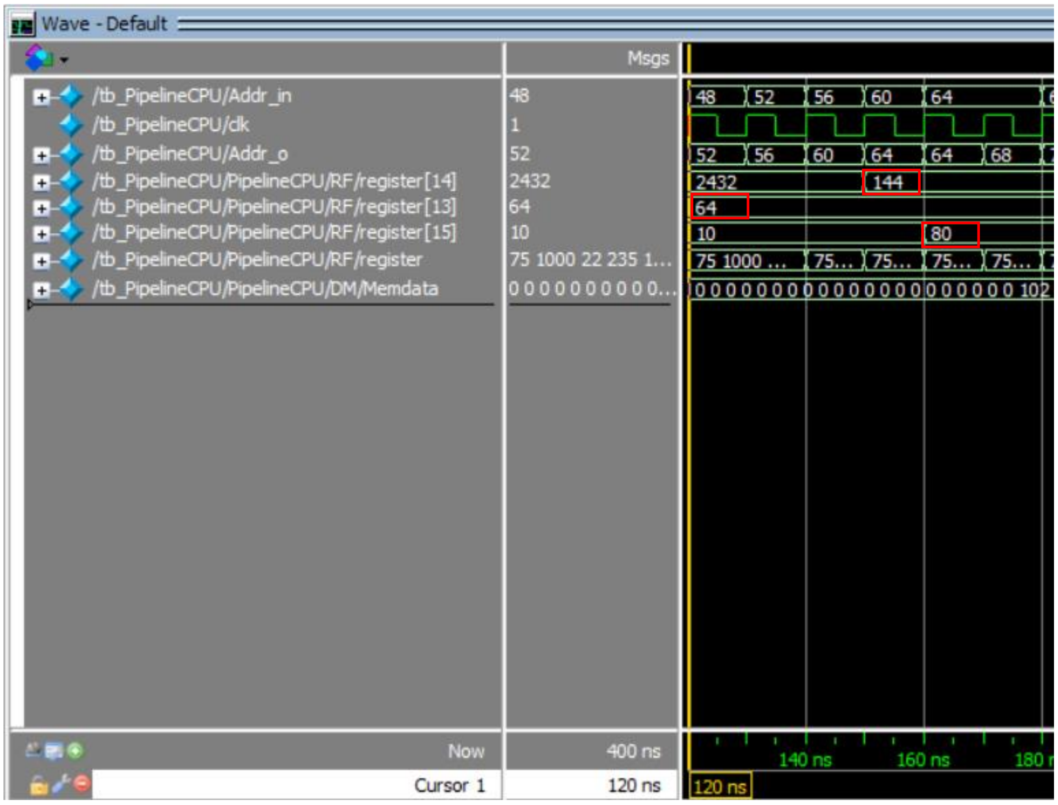
```
end

initial begin
        for(i=0;i<200;i=i+1)begin
                Instr[i]=32'd0;
        end
        Instr[0]=32'b010100_01001_01001_01000_00000_010101;//add $t0, $t1, $t1
        Instr[1]=32'b010100_01010_01100_01001_00000_010110;//sub $t1, $t2, $t4
        Instr[2]=32'b010100_01101_00000_01100_00010_010111;//srl $t4, $t5, 2
        Instr[3]=32'b010100_01110_00000_01110_00100_011000;//sll $t6, $t6, 4
        Instr[4]=32'b010100_01001_01010_01011_00000_011001;//xor $t3, $t1, $t2
        Instr[5]=32'b010100_01010_01100_01101_00000_011010;//and $t5, $t4, $t2

        Instr[6]=32'b101011_01111_01000_0000000000000010;//sw $t0, 2($t7)
        Instr[7]=32'b100011_01111_10011_0000000000000010;//lw $s3, 2($t7)
        Instr[8]=32'b101011_01111_10100_0000000000000100;//sw $s4, 4($t7)
        Instr[9]=32'b101011_01010_01000_0000000000000010;//sw $t0, 2($t2)
        Instr[10]=32'b100011_01010_10100_0000000000000011;//lw $s4, 3($t2)

        Instr[11]=32'b010100_01101_01100_01110_00000_010101;//add $t6, $t5, $t4
        Instr[12]=32'b010100_01110_01101_01111_00000_010110;//sub $t7, $t6, $t5
        Instr[13]=32'b010100_01110_01111_01011_00000_010101;//add $t3, $t6, $t7
        Instr[14]=32'b100011_01010_10001_0000000000000010;//lw $s1, 2($t2)
        Instr[15]=32'b101011_01101_10001_0000000000000010;//sw $s1, 2($t5)
        Instr[16]=32'b100011_01101_01001_0000000000000010;//lw $t1, 2($t5)
        Instr[17]=32'b010100_01001_01001_01010_00000_010101;//add $t2, $t1, $t1
end
endmodule
```

1. Instr[11]=>$t6($14)=358=$t5(64)+$t4(80)


Wave - Default

```
                                            Msgs
/tb_PipelineCPU/Addr_in                     36                  44   48   52   56   60
/tb_PipelineCPU/clk                         0
/tb_PipelineCPU/Addr_o                      40                  48   52   56   60   64
/tb_PipelineCPU/PipelineCPU/RF/register[12] 80                  80
/tb_PipelineCPU/PipelineCPU/RF/register[13] 64                  64
/tb_PipelineCPU/PipelineCPU/RF/register[14] 2432                2432                144
/tb_PipelineCPU/PipelineCPU/RF/register     75 1000 22 235 1... 75 1000 22 235 ...  75...  75...
/tb_PipelineCPU/PipelineCPU/DM/Memdata      0 0 0 0 0 0 0 0 0...  0 ...  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Now                                         400 ns              120 ns   140 ns   160
Cursor 1                                    98 ns
```

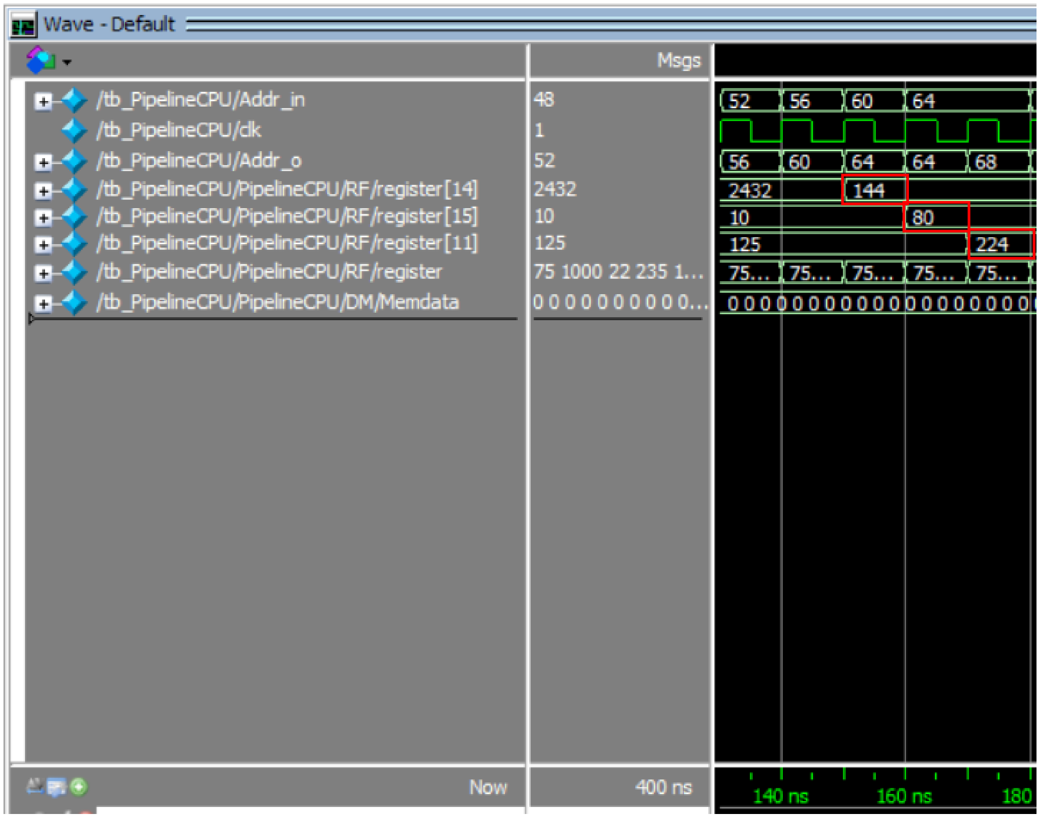
2. Instr[12]=>$t7($15)=24=$t6(144)-$t5(64)



```
Wave - Default
                                                Msgs
/tb_PipelineCPU/Addr_in                         48                 48   52   56   60   64
/tb_PipelineCPU/clk                             1
/tb_PipelineCPU/Addr_o                          52                 52   56   60   64   64   68
/tb_PipelineCPU/PipelineCPU/RF/register[14]     2432               2432           144
/tb_PipelineCPU/PipelineCPU/RF/register[13]     64                 64
/tb_PipelineCPU/PipelineCPU/RF/register[15]     10                 10                  80
/tb_PipelineCPU/PipelineCPU/RF/register         75 1000 22 235 1...  75 1000 ...  75...  75...  75...  75...
/tb_PipelineCPU/PipelineCPU/DM/Memdata          0 0 0 0 0 0 0 0 0...  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 102
                                     Now        400 ns             140 ns     160 ns     180
                                     Cursor 1   120 ns             120 ns
```

這部分會產生一個 hazard，因為前一個值還沒存進去，所以要利用 forwarding 做處理

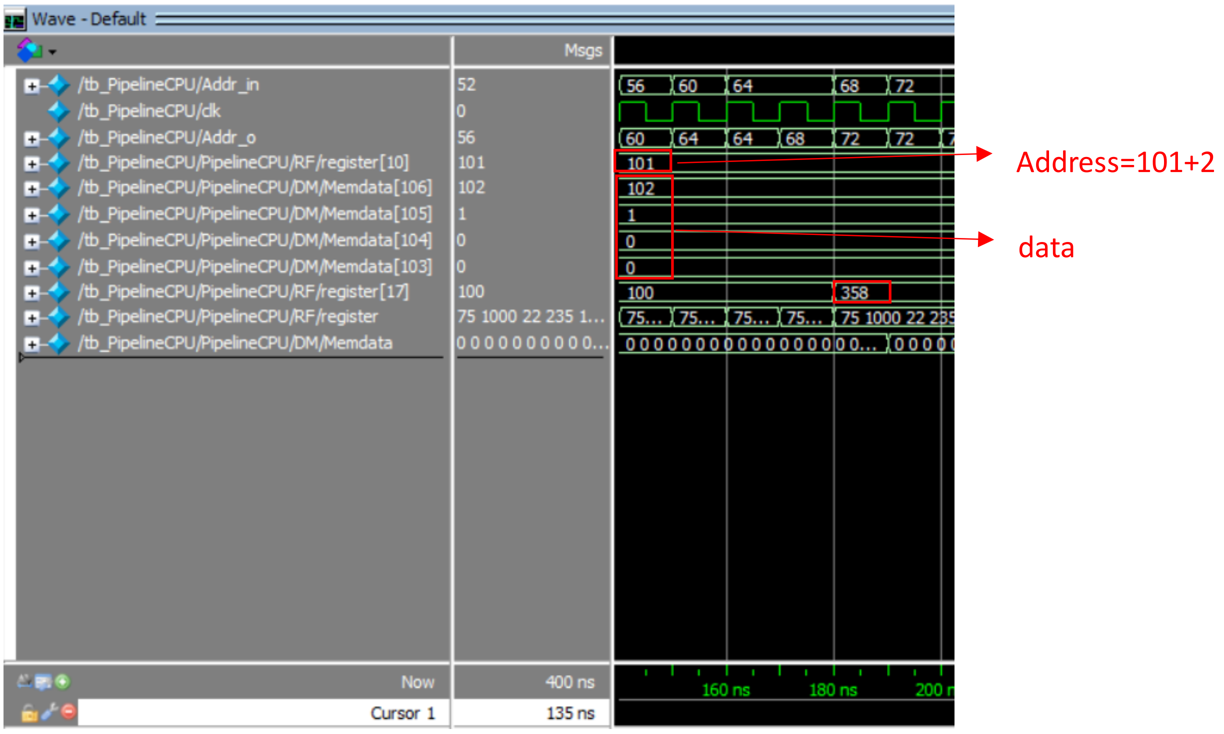
3. Instr[13]=> $t3($14)=224=$t6(144)+$t7(80)



```
Wave - Default
                                                  Msgs
/tb_PipelineCPU/Addr_in                           48                  52  56  60  64
/tb_PipelineCPU/clk                               1
/tb_PipelineCPU/Addr_o                            52                  56  60  64  64  68
/tb_PipelineCPU/PipelineCPU/RF/register[14]       2432                2432  144
/tb_PipelineCPU/PipelineCPU/RF/register[15]       10                  10  80
/tb_PipelineCPU/PipelineCPU/RF/register[11]       125                 125  224
/tb_PipelineCPU/PipelineCPU/RF/register           75 1000 22 235 1... 75... 75... 75... 75... 75...
/tb_PipelineCPU/PipelineCPU/DM/Memdata            0 0 0 0 0 0 0 0 0... 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Now                                               400 ns              140 ns  160 ns  180
```

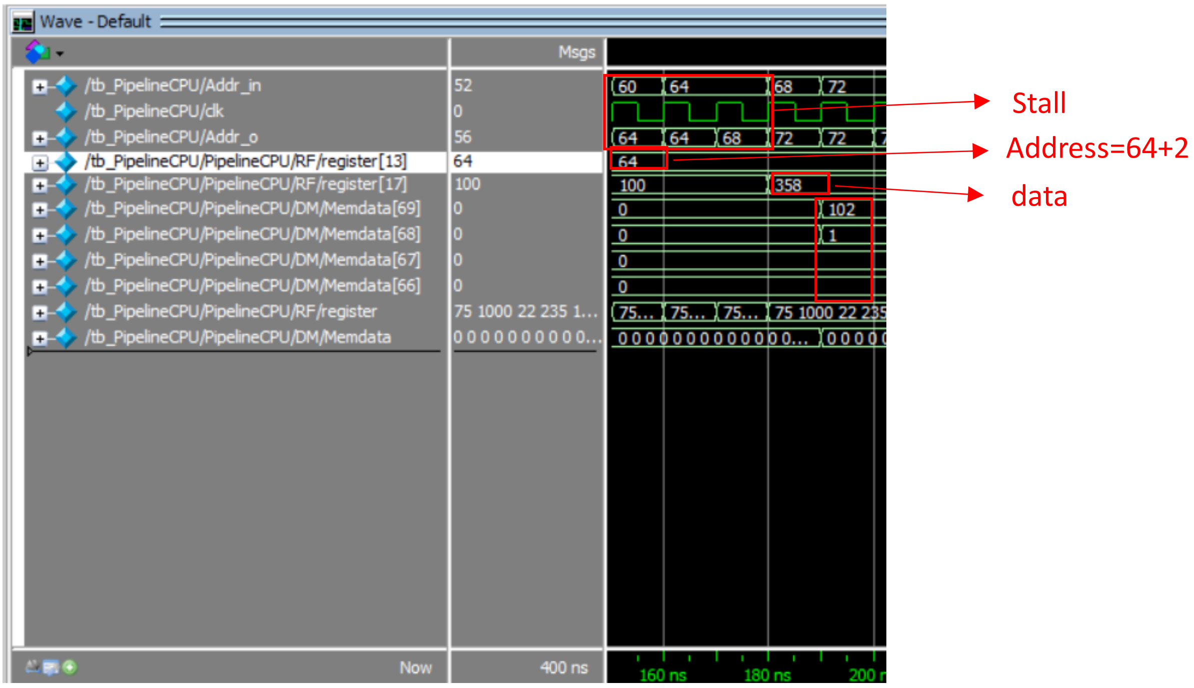
這部分一樣會產生 hazard，也是因為值還沒存進去所產生的

4.Instr[14]=>$s1($17)={Memdata[103],Memdata[104],Memdata[105],Memdata[106]}



```
Wave - Default
                                                  Msgs
/tb_PipelineCPU/Addr_in                           52                  56  60  64  68  72
/tb_PipelineCPU/clk                               0
/tb_PipelineCPU/Addr_o                            56                  60  64  64  68  72  72
/tb_PipelineCPU/PipelineCPU/RF/register[10]       101                 101
/tb_PipelineCPU/PipelineCPU/DM/Memdata[106]       102                 102
/tb_PipelineCPU/PipelineCPU/DM/Memdata[105]       1                   1
/tb_PipelineCPU/PipelineCPU/DM/Memdata[104]       0                   0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[103]       0                   0
/tb_PipelineCPU/PipelineCPU/RF/register[17]       100                 100  358
/tb_PipelineCPU/PipelineCPU/RF/register           75 1000 22 235 1... 75... 75... 75... 75... 75 1000 22 235
/tb_PipelineCPU/PipelineCPU/DM/Memdata            0 0 0 0 0 0 0 0 0... 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0... 0 0 0 0

Now                                               400 ns              160 ns  180 ns  200
Cursor 1                                          135 ns
```

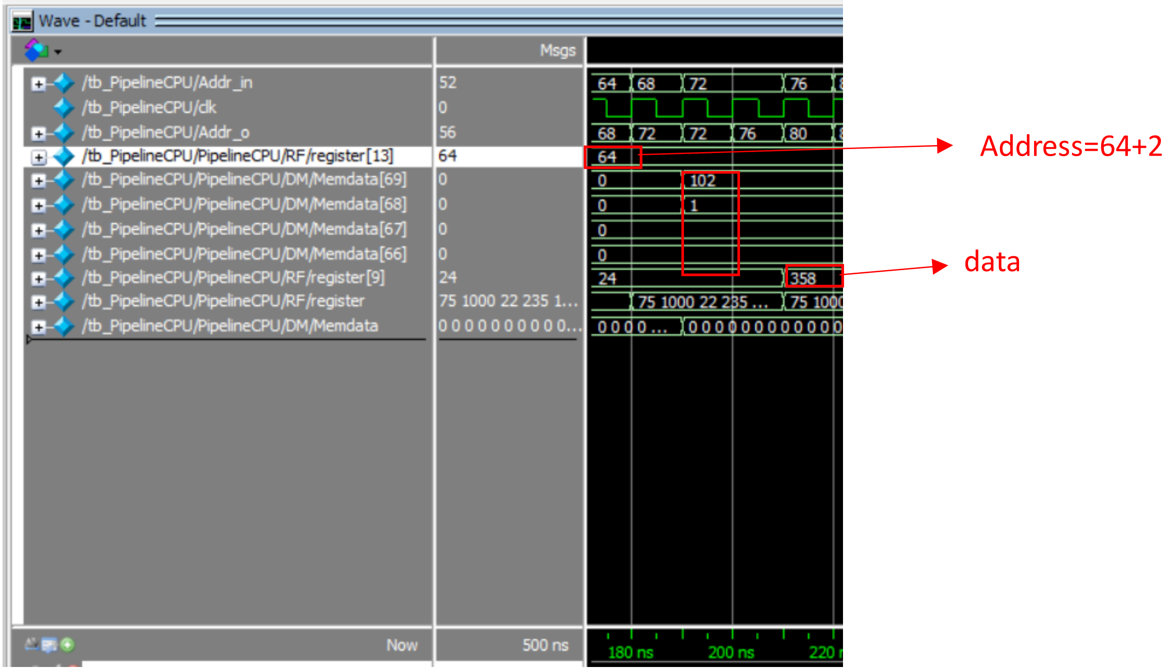
5. Instr[15]=> {Memdata[66],Memdata[67],Memdata[68],Memdata[69]}
=\{8’d0,8’d0,8’b00000001,8’b01100110\}(358)



```
Wave - Default
                                                     Msgs
/tb_PipelineCPU/Addr_in                              52
/tb_PipelineCPU/clk                                  0
/tb_PipelineCPU/Addr_o                               56
/tb_PipelineCPU/PipelineCPU/RF/register[13]          64
/tb_PipelineCPU/PipelineCPU/RF/register[17]          100
/tb_PipelineCPU/PipelineCPU/DM/Memdata[69]           0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[68]           0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[67]           0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[66]           0
/tb_PipelineCPU/PipelineCPU/RF/register              75 1000 22 235 1...
/tb_PipelineCPU/PipelineCPU/DM/Memdata               0 0 0 0 0 0 0 0 0...
Now                                                  400 ns
```

這部分會產生一個 hazard，也是可以利用 forwarding 去做處理，但是要注意的是這次 hazard 比較特別，所以會有 stall

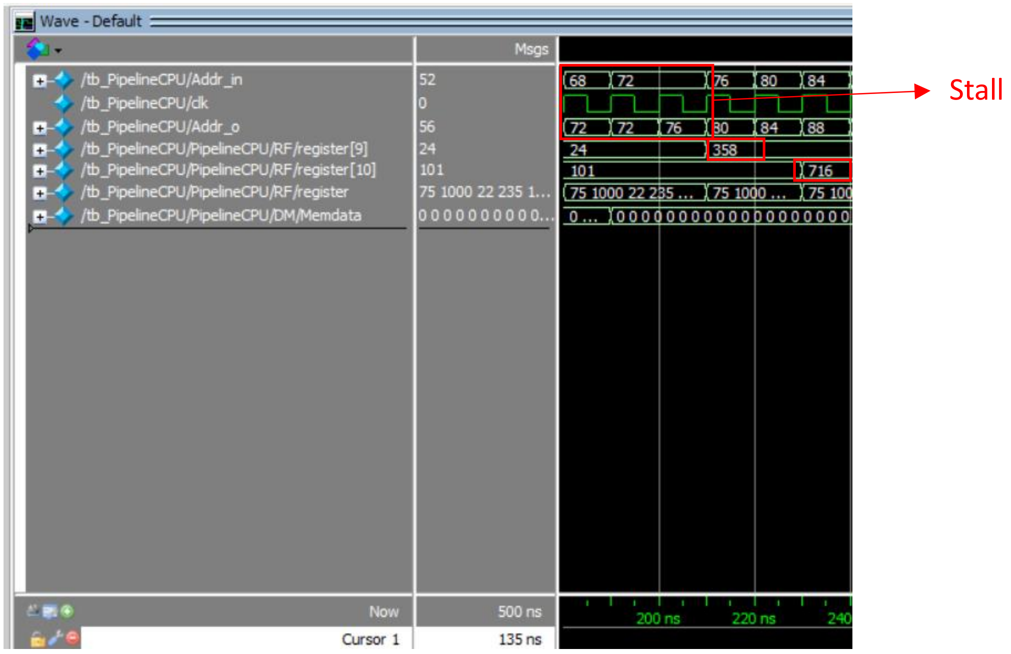
6.Instr[16]=>\$t1(\$9)={Memdata[66],Memdata[67],Memdata[68],Memdata[69]}



```
Wave - Default
                                                     Msgs
/tb_PipelineCPU/Addr_in                              52
/tb_PipelineCPU/clk                                  0
/tb_PipelineCPU/Addr_o                               56
/tb_PipelineCPU/PipelineCPU/RF/register[13]          64
/tb_PipelineCPU/PipelineCPU/DM/Memdata[69]           0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[68]           0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[67]           0
/tb_PipelineCPU/PipelineCPU/DM/Memdata[66]           0
/tb_PipelineCPU/PipelineCPU/RF/register[9]           24
/tb_PipelineCPU/PipelineCPU/RF/register              75 1000 22 235 1...
/tb_PipelineCPU/PipelineCPU/DM/Memdata               0 0 0 0 0 0 0 0 0...
Now                                                  500 ns
```

這裡也是有 hazard 的產生

7. Instr[17]=>$t2($10)=716=$t1(358)+$t1(358)

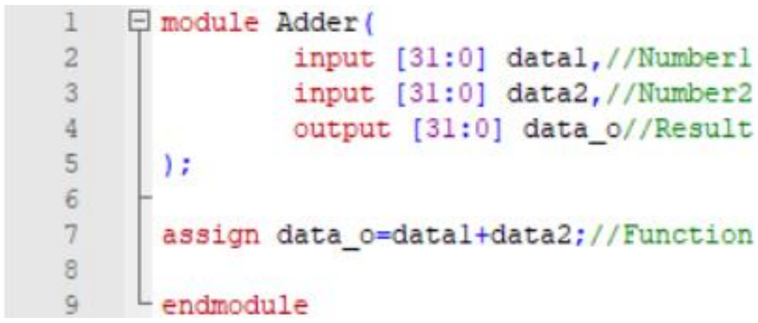


```
/tb_PipelineCPU/Addr_in                       52                     68  72  76  80  84
/tb_PipelineCPU/clk                           0
/tb_PipelineCPU/Addr_o                        56                     72  72  76  80  84  88
/tb_PipelineCPU/PipelineCPU/RF/register[9]    24                     24  358
/tb_PipelineCPU/PipelineCPU/RF/register[10]   101                    101  716
/tb_PipelineCPU/PipelineCPU/RF/register       75 1000 22 235 1...    75 1000 22 235 ...  75 1000 ...  75 100
/tb_PipelineCPU/PipelineCPU/DM/Memdata        0 0 0 0 0 0 0 0 0...   0 ...  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Now       500 ns
Cursor 1  135 ns
```

這邊跟剛剛一樣會有 hazard，也是用一樣的處理方式，但這是屬於特別版，所以會有 stall 產生

Module 程式碼

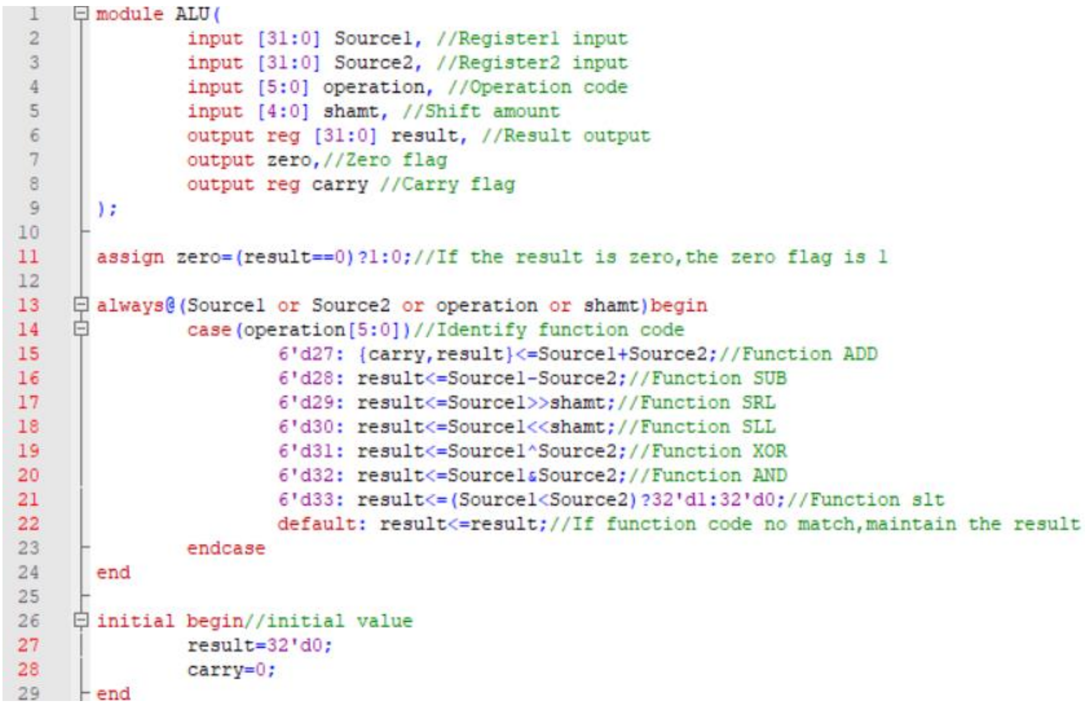
a. Adder



```
module Adder(
        input [31:0] data1,//Number1
        input [31:0] data2,//Number2
        output [31:0] data_o//Result
);

assign data_o=data1+data2;//Function

endmodule
```

這部分與上次 module 相同

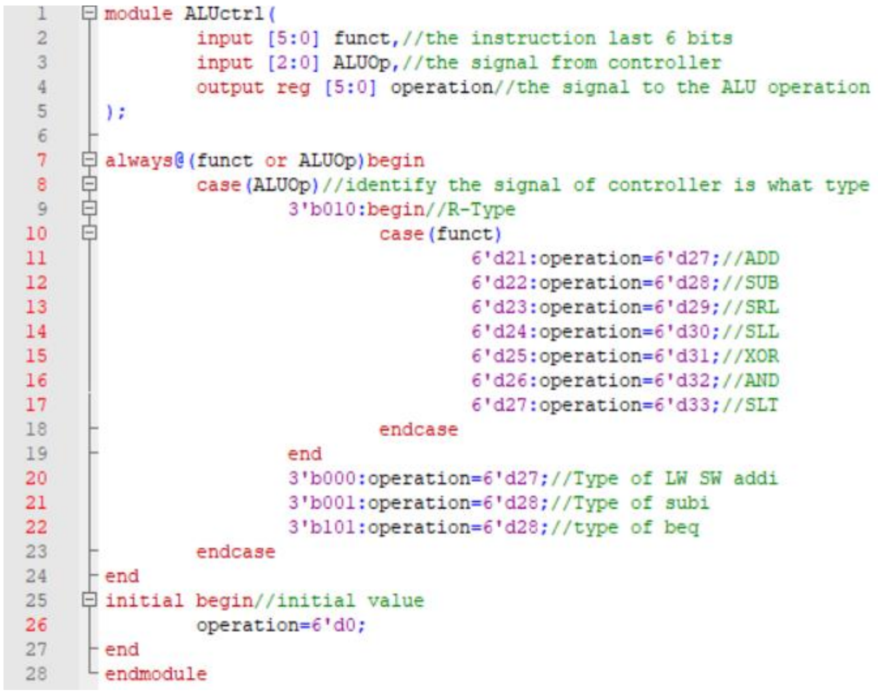
b.ALU



```
module ALU(
        input [31:0] Source1, //Register1 input
        input [31:0] Source2, //Register2 input
        input [5:0] operation, //Operation code
        input [4:0] shamt, //Shift amount
        output reg [31:0] result, //Result output
        output zero,//Zero flag
        output reg carry //Carry flag
);

assign zero=(result==0)?1:0;//If the result is zero,the zero flag is 1

always@(Source1 or Source2 or operation or shamt)begin
        case(operation[5:0])//Identify function code
                6'd27: {carry,result}<=Source1+Source2;//Function ADD
                6'd28: result<=Source1-Source2;//Function SUB
                6'd29: result<=Source1>>shamt;//Function SRL
                6'd30: result<=Source1<<shamt;//Function SLL
                6'd31: result<=Source1^Source2;//Function XOR
                6'd32: result<=Source1&Source2;//Function AND
                6'd33: result<=(Source1<Source2)?32'd1:32'd0;//Function slt
                default: result<=result;//If function code no match,maintain the result
        endcase
end

initial begin//initial value
        result=32'd0;
        carry=0;
end
```

這部分與上次 module 相同

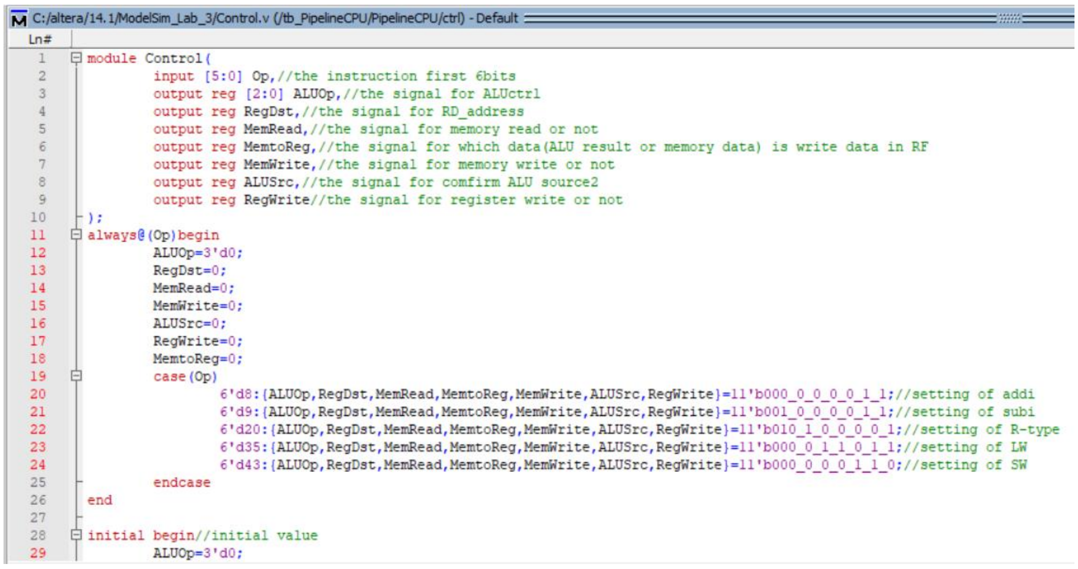
c.ALUctrl



```verilog
module ALUctrl(
        input [5:0] funct,//the instruction last 6 bits
        input [2:0] ALUOp,//the signal from controller
        output reg [5:0] operation//the signal to the ALU operation
);

always@(funct or ALUOp)begin
        case(ALUOp)//identify the signal of controller is what type
                3'b010:begin//R-Type
                        case(funct)
                                6'd21:operation=6'd27;//ADD
                                6'd22:operation=6'd28;//SUB
                                6'd23:operation=6'd29;//SRL
                                6'd24:operation=6'd30;//SLL
                                6'd25:operation=6'd31;//XOR
                                6'd26:operation=6'd32;//AND
                                6'd27:operation=6'd33;//SLT
                        endcase
                end
                3'b000:operation=6'd27;//Type of LW SW addi
                3'b001:operation=6'd28;//Type of subi
                3'b101:operation=6'd28;//type of beq
        endcase
end
initial begin//initial value
        operation=6'd0;
end
endmodule
```

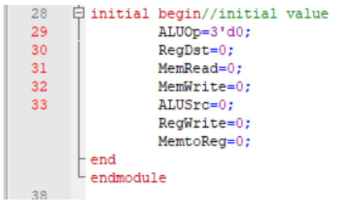
這部分與上次 module 相同

d. Control



C:/altera/14.1/ModelSim_Lab_3/Control.v (/tb_PipelineCPU/PipelineCPU/ctrl) - Default

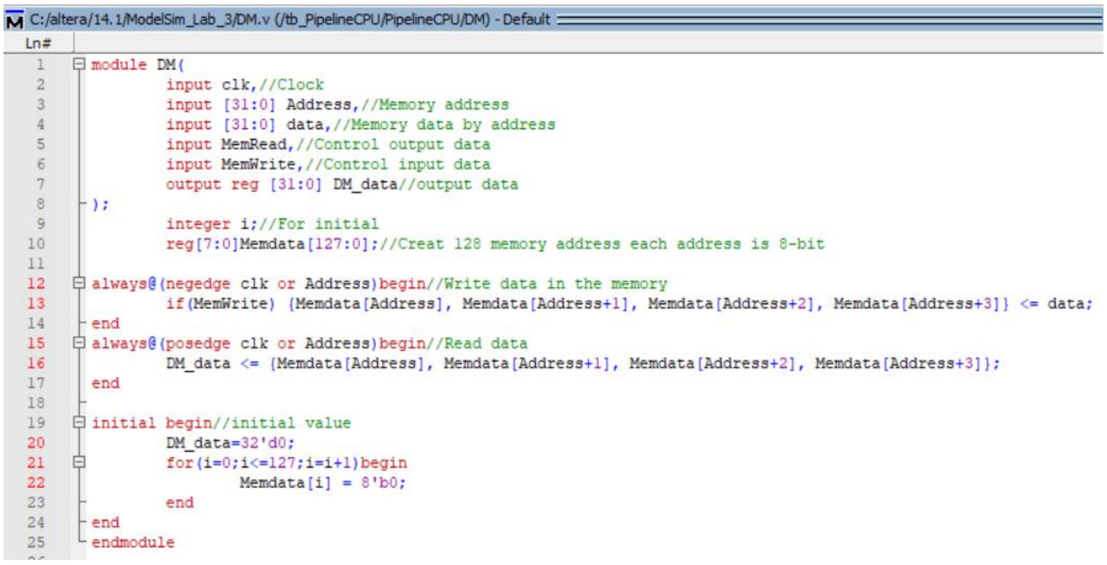
```verilog
module Control(
        input [5:0] Op,//the instruction first 6bits
        output reg [2:0] ALUOp,//the signal for ALUctrl
        output reg RegDst,//the signal for RD_address
        output reg MemRead,//the signal for memory read or not
        output reg MemtoReg,//the signal for which data(ALU result or memory data) is write data in RF
        output reg MemWrite,//the signal for memory write or not
        output reg ALUSrc,//the signal for comfirm ALU source2
        output reg RegWrite//the signal for register write or not
);
always@(Op)begin
        ALUOp=3'd0;
        RegDst=0;
        MemRead=0;
        MemWrite=0;
        ALUSrc=0;
        RegWrite=0;
        MemtoReg=0;
        case(Op)
                6'd8:{ALUOp,RegDst,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite}=11'b000_0_0_0_0_1_1;//setting of addi
                6'd9:{ALUOp,RegDst,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite}=11'b001_0_0_0_0_1_1;//setting of subi
                6'd20:{ALUOp,RegDst,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite}=11'b010_1_0_0_0_0_1;//setting of R-type
                6'd35:{ALUOp,RegDst,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite}=11'b000_0_1_1_0_1_1;//setting of LW
                6'd43:{ALUOp,RegDst,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite}=11'b000_0_0_0_1_1_0;//setting of SW
        endcase
end

initial begin//initial value
        ALUOp=3'd0;
```



```verilog
initial begin//initial value
        ALUOp=3'd0;
        RegDst=0;
        MemRead=0;
        MemWrite=0;
        ALUSrc=0;
        RegWrite=0;
        MemtoReg=0;
end
endmodule
```

這部分與上次 module 類似 只差在這次沒有 Jump 和 branch 訊號

e.DM

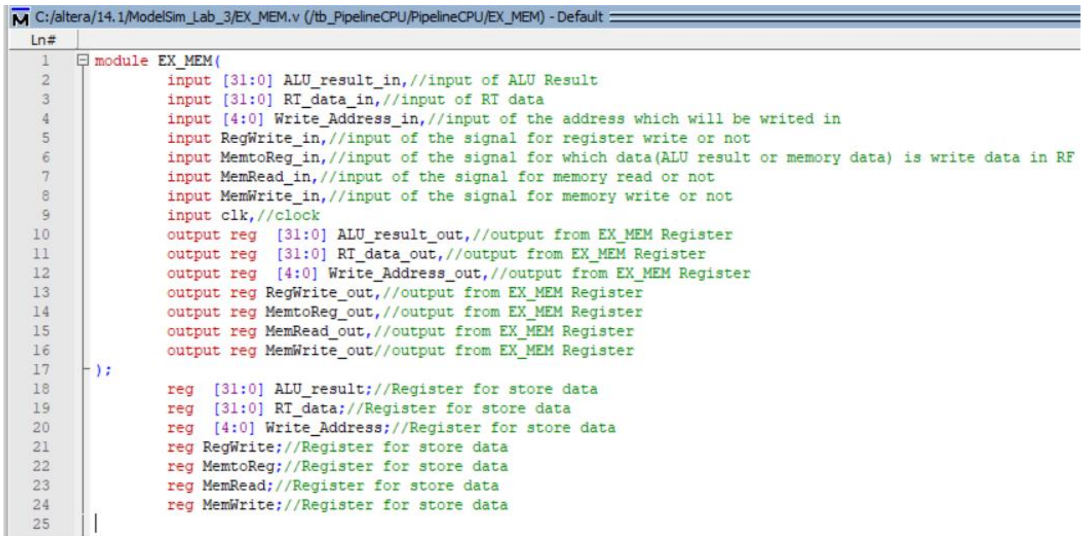


C:/altera/14.1/ModelSim_Lab_3/DM.v (/tb_PipelineCPU/PipelineCPU/DM) - Default

```verilog
module DM(
        input clk,//Clock
        input [31:0] Address,//Memory address
        input [31:0] data,//Memory data by address
        input MemRead,//Control output data
        input MemWrite,//Control input data
        output reg [31:0] DM_data//output data
);
        integer i;//For initial
        reg[7:0]Memdata[127:0];//Creat 128 memory address each address is 8-bit

always@(negedge clk or Address)begin//Write data in the memory
        if(MemWrite) {Memdata[Address], Memdata[Address+1], Memdata[Address+2], Memdata[Address+3]} <= data;
end
always@(posedge clk or Address)begin//Read data
        DM_data <= {Memdata[Address], Memdata[Address+1], Memdata[Address+2], Memdata[Address+3]};
end

initial begin//initial value
        DM_data=32'd0;
        for(i=0;i<=127;i=i+1)begin
                Memdata[i] = 8'b0;
        end
end
endmodule
```

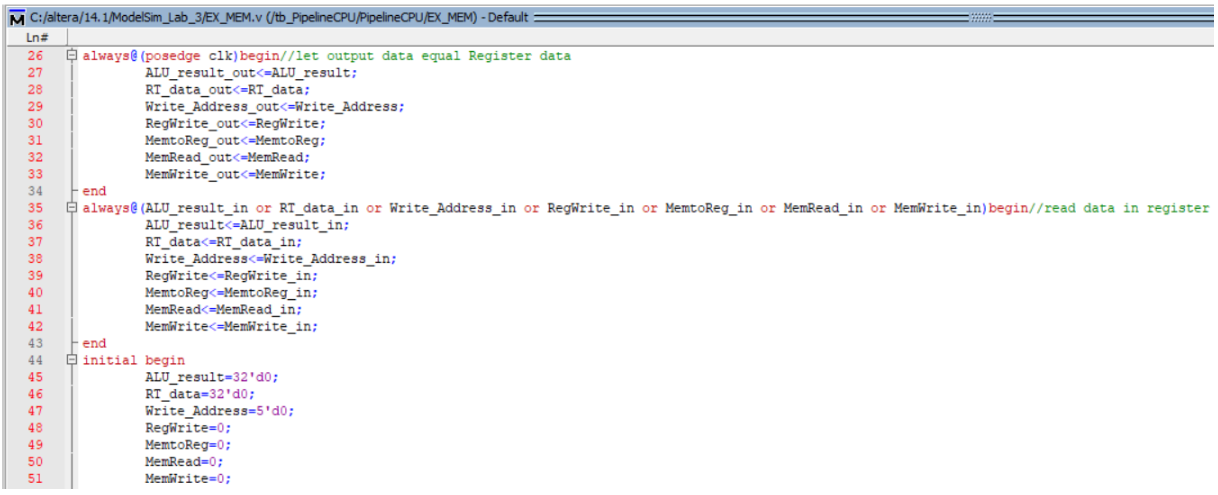
這部分與上次 module 相同

f.EX_MEM



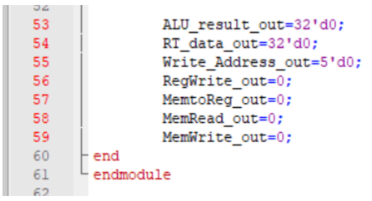
C:/altera/14.1/ModelSim_Lab_3/EX_MEM.v (/tb_PipelineCPU/PipelineCPU/EX_MEM) - Default

```verilog
module EX_MEM(
        input [31:0] ALU_result_in,//input of ALU Result
        input [31:0] RT_data_in,//input of RT data
        input [4:0] Write_Address_in,//input of the address which will be writed in
        input RegWrite_in,//input of the signal for register write or not
        input MemtoReg_in,//input of the signal for which data(ALU result or memory data) is write data in RF
        input MemRead_in,//input of the signal for memory read or not
        input MemWrite_in,//input of the signal for memory write or not
        input clk,//clock
        output reg  [31:0] ALU_result_out,//output from EX_MEM Register
        output reg  [31:0] RT_data_out,//output from EX_MEM Register
        output reg  [4:0] Write_Address_out,//output from EX_MEM Register
        output reg RegWrite_out,//output from EX_MEM Register
        output reg MemtoReg_out,//output from EX_MEM Register
        output reg MemRead_out,//output from EX_MEM Register
        output reg MemWrite_out//output from EX_MEM Register
);
        reg  [31:0] ALU_result;//Register for store data
        reg  [31:0] RT_data;//Register for store data
        reg  [4:0] Write_Address;//Register for store data
        reg RegWrite;//Register for store data
        reg MemtoReg;//Register for store data
        reg MemRead;//Register for store data
        reg MemWrite;//Register for store data

```



C:/altera/14.1/ModelSim_Lab_3/EX_MEM.v (/tb_PipelineCPU/PipelineCPU/EX_MEM) - Default

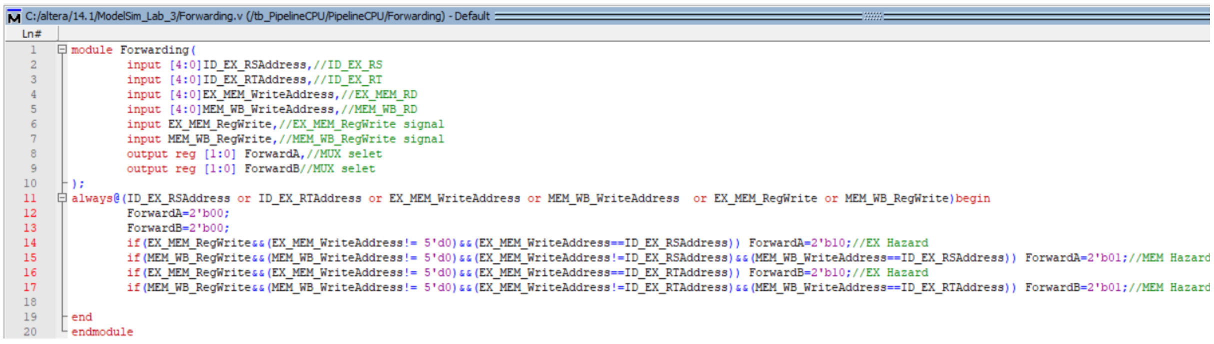
```verilog
always@(posedge clk)begin//let output data equal Register data
        ALU_result_out<=ALU_result;
        RT_data_out<=RT_data;
        Write_Address_out<=Write_Address;
        RegWrite_out<=RegWrite;
        MemtoReg_out<=MemtoReg;
        MemRead_out<=MemRead;
        MemWrite_out<=MemWrite;
end
always@(ALU_result_in or RT_data_in or Write_Address_in or RegWrite_in or MemtoReg_in or MemRead_in or MemWrite_in)begin//read data in register
        ALU_result<=ALU_result_in;
        RT_data<=RT_data_in;
        Write_Address<=Write_Address_in;
        RegWrite<=RegWrite_in;
        MemtoReg<=MemtoReg_in;
        MemRead<=MemRead_in;
        MemWrite<=MemWrite_in;
end
initial begin
        ALU_result=32'd0;
        RT_data=32'd0;
        Write_Address=5'd0;
        RegWrite=0;
        MemtoReg=0;
        MemRead=0;
        MemWrite=0;
```



```
            ALU_result_out=32'd0;
            RT_data_out=32'd0;
            Write_Address_out=5'd0;
            RegWrite_out=0;
            MemtoReg_out=0;
            MemRead_out=0;
            MemWrite_out=0;
end
endmodule
```

這部分是因為 pipeline 需要所以加上去當暫存器使用，但想法也非常簡單，只要考慮好甚麼時候存進暫存器，甚麼時候要讓暫存器的值讀出，這樣就可以搞定了

g. Forwarding

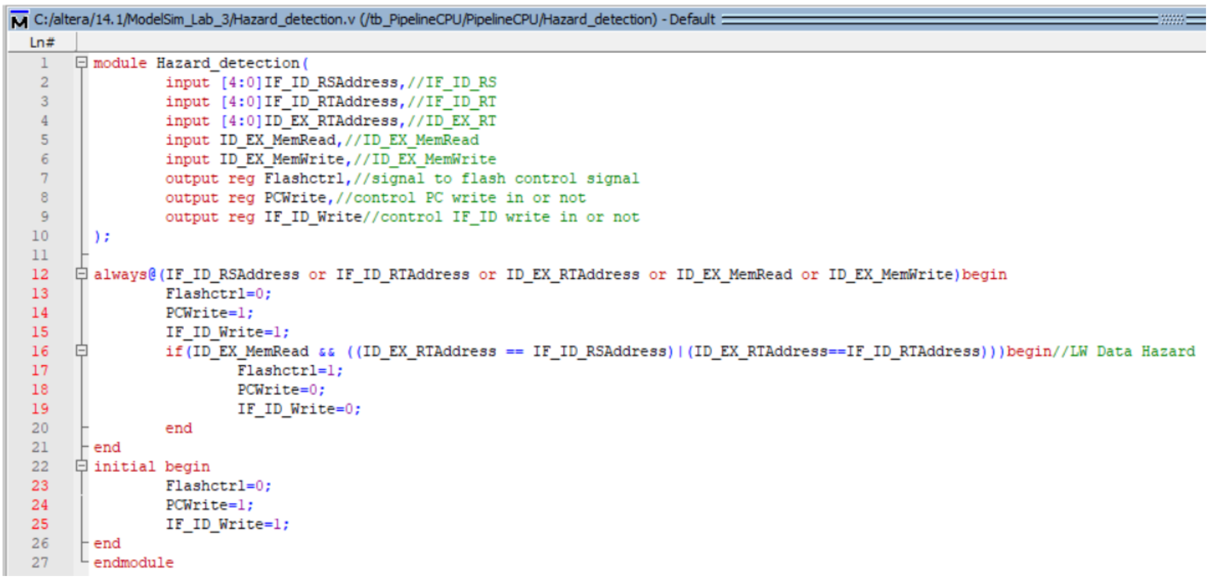


C:/altera/14.1/ModelSim_Lab_3/Forwarding.v (/tb_PipelineCPU/PipelineCPU/Forwarding) - Default

```
module Forwarding(
        input [4:0]ID_EX_RSAddress,//ID_EX_RS
        input [4:0]ID_EX_RTAddress,//ID_EX_RT
        input [4:0]EX_MEM_WriteAddress,//EX_MEM_RD
        input [4:0]MEM_WB_WriteAddress,//MEM_WB_RD
        input EX_MEM_RegWrite,//EX_MEM_RegWrite signal
        input MEM_WB_RegWrite,//MEM_WB_RegWrite signal
        output reg [1:0] ForwardA,//MUX selet
        output reg [1:0] ForwardB//MUX selet
);
always@(ID_EX_RSAddress or ID_EX_RTAddress or EX_MEM_WriteAddress or MEM_WB_WriteAddress  or EX_MEM_RegWrite or MEM_WB_RegWrite)begin
        ForwardA=2'b00;
        ForwardB=2'b00;
        if(EX_MEM_RegWrite&&(EX_MEM_WriteAddress!= 5'd0)&&(EX_MEM_WriteAddress==ID_EX_RSAddress)) ForwardA=2'b10;//EX Hazard
        if(MEM_WB_RegWrite&&(MEM_WB_WriteAddress!= 5'd0)&&(EX_MEM_WriteAddress!=ID_EX_RSAddress)&&(MEM_WB_WriteAddress==ID_EX_RSAddress)) ForwardA=2'b01;//MEM Hazard
        if(EX_MEM_RegWrite&&(EX_MEM_WriteAddress!= 5'd0)&&(EX_MEM_WriteAddress==ID_EX_RTAddress)) ForwardB=2'b10;//EX Hazard
        if(MEM_WB_RegWrite&&(MEM_WB_WriteAddress!= 5'd0)&&(EX_MEM_WriteAddress!=ID_EX_RTAddress)&&(MEM_WB_WriteAddress==ID_EX_RTAddress)) ForwardB=2'b01;//MEM Hazard

end
endmodule
```

這部分是上課有教過的部分，就是為了要處理 hazard 的問題，也是只要依照上課所講的那樣下去做處理，很快就可以解決

h. Hazard_detection

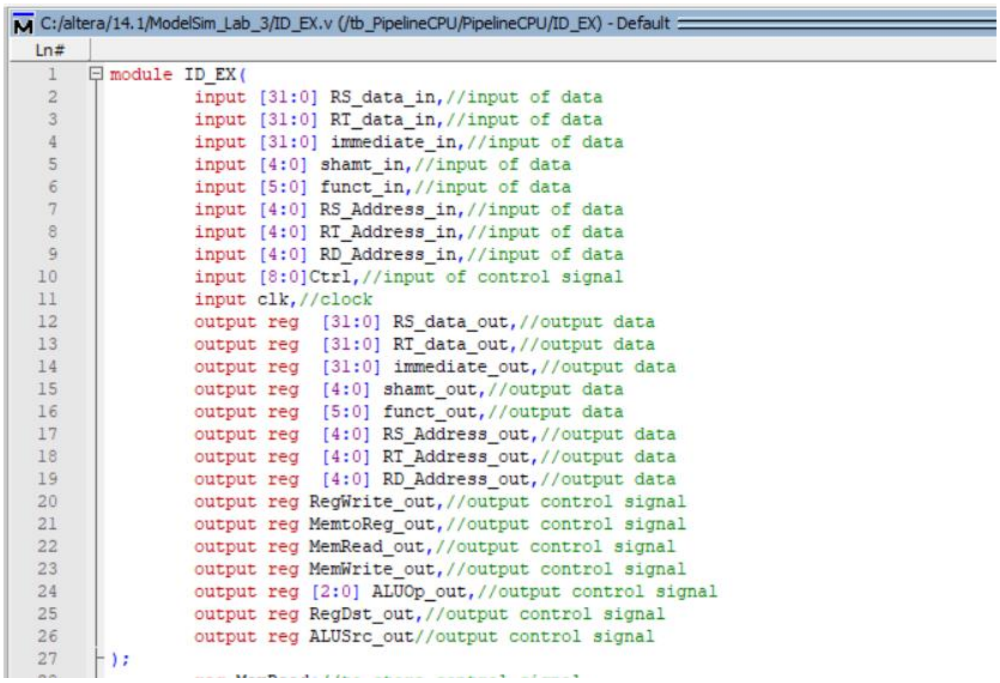


C:/altera/14.1/ModelSim_Lab_3/Hazard_detection.v (/tb_PipelineCPU/PipelineCPU/Hazard_detection) - Default

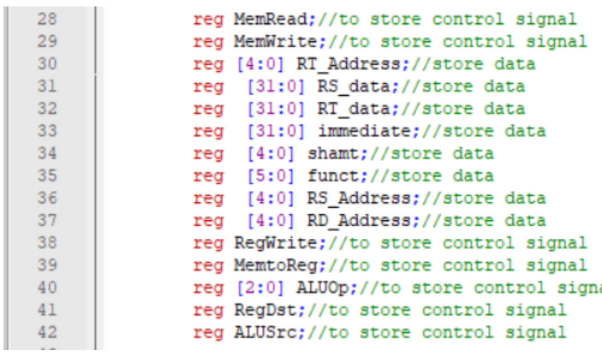
```
module Hazard_detection(
        input [4:0]IF_ID_RSAddress,//IF_ID_RS
        input [4:0]IF_ID_RTAddress,//IF_ID_RT
        input [4:0]ID_EX_RTAddress,//ID_EX_RT
        input ID_EX_MemRead,//ID_EX_MemRead
        input ID_EX_MemWrite,//ID_EX_MemWrite
        output reg Flashctrl,//signal to flash control signal
        output reg PCWrite,//control PC write in or not
        output reg IF_ID_Write//control IF_ID write in or not
);

always@(IF_ID_RSAddress or IF_ID_RTAddress or ID_EX_RTAddress or ID_EX_MemRead or ID_EX_MemWrite)begin
        Flashctrl=0;
        PCWrite=1;
        IF_ID_Write=1;
        if(ID_EX_MemRead && ((ID_EX_RTAddress == IF_ID_RSAddress)|(ID_EX_RTAddress==IF_ID_RTAddress)))begin//LW Data Hazard
                Flashctrl=1;
                PCWrite=0;
                IF_ID_Write=0;
        end
end
initial begin
        Flashctrl=0;
        PCWrite=1;
        IF_ID_Write=1;
end
endmodule
```

這部分是延伸上面那部分用的，一樣是為了處理 hazard 的，而這個部分是因為 LW 需要 stall 一個 Cycle 的時間，所以必須利用這個去處理 stall 的問題

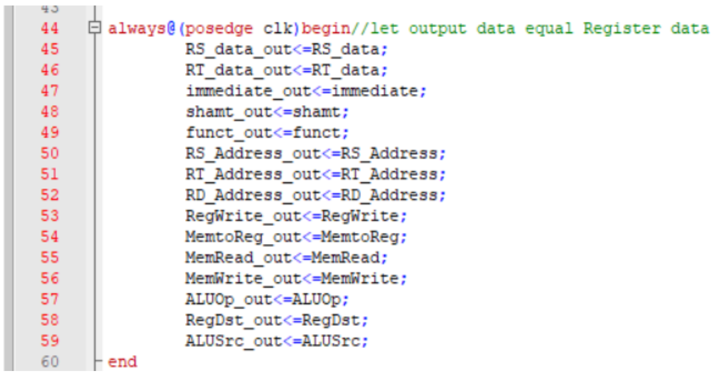
i. ID_EX


C:/altera/14.1/ModelSim_Lab_3/ID_EX.v (/tb_PipelineCPU/PipelineCPU/ID_EX) - Default
Ln#

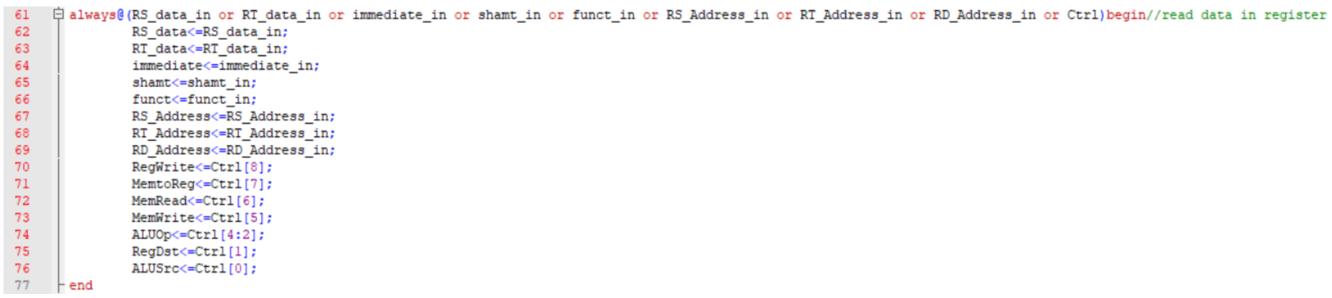
```
module ID_EX(
        input [31:0] RS_data_in,//input of data
        input [31:0] RT_data_in,//input of data
        input [31:0] immediate_in,//input of data
        input [4:0] shamt_in,//input of data
        input [5:0] funct_in,//input of data
        input [4:0] RS_Address_in,//input of data
        input [4:0] RT_Address_in,//input of data
        input [4:0] RD_Address_in,//input of data
        input [8:0]Ctrl,//input of control signal
        input clk,//clock
        output reg  [31:0] RS_data_out,//output data
        output reg  [31:0] RT_data_out,//output data
        output reg  [31:0] immediate_out,//output data
        output reg  [4:0] shamt_out,//output data
        output reg  [5:0] funct_out,//output data
        output reg  [4:0] RS_Address_out,//output data
        output reg  [4:0] RT_Address_out,//output data
        output reg  [4:0] RD_Address_out,//output data
        output reg RegWrite_out,//output control signal
        output reg MemtoReg_out,//output control signal
        output reg MemRead_out,//output control signal
        output reg MemWrite_out,//output control signal
        output reg [2:0] ALUOp_out,//output control signal
        output reg RegDst_out,//output control signal
        output reg ALUSrc_out//output control signal
);
```



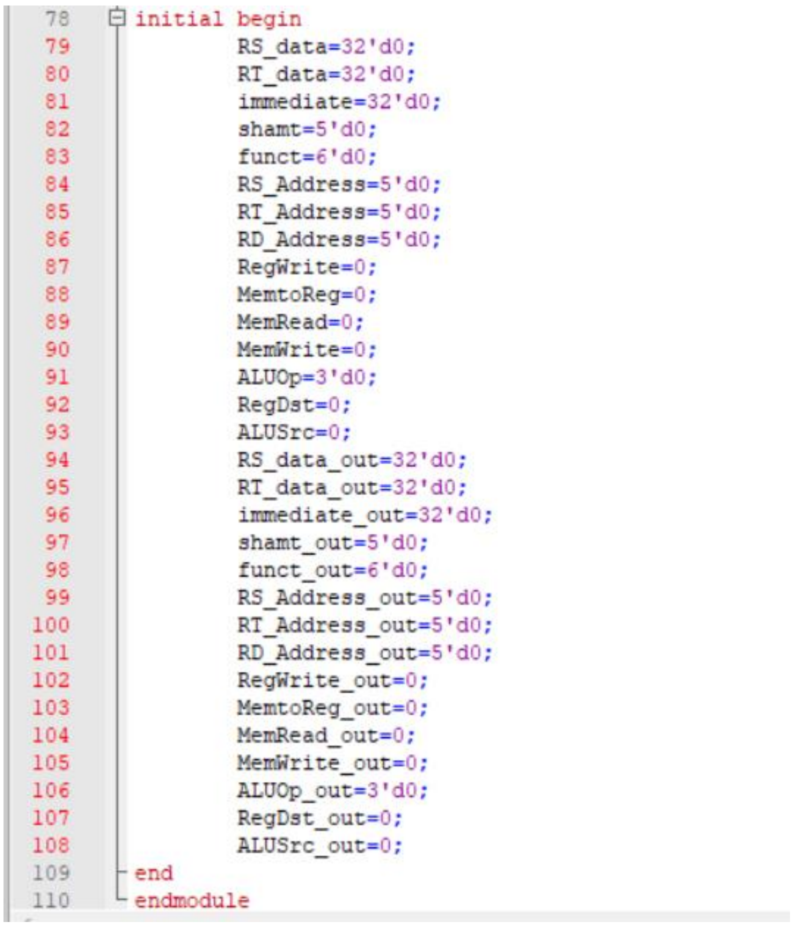
```
        reg MemRead;//to store control signal
        reg MemWrite;//to store control signal
        reg [4:0] RT_Address;//store data
        reg  [31:0] RS_data;//store data
        reg  [31:0] RT_data;//store data
        reg  [31:0] immediate;//store data
        reg  [4:0] shamt;//store data
        reg  [5:0] funct;//store data
        reg  [4:0] RS_Address;//store data
        reg  [4:0] RD_Address;//store data
        reg RegWrite;//to store control signal
        reg MemtoReg;//to store control signal
        reg [2:0] ALUOp;//to store control signal
        reg RegDst;//to store control signal
        reg ALUSrc;//to store control signal
```



```
always@(posedge clk)begin//let output data equal Register data
        RS_data_out<=RS_data;
        RT_data_out<=RT_data;
        immediate_out<=immediate;
        shamt_out<=shamt;
        funct_out<=funct;
        RS_Address_out<=RS_Address;
        RT_Address_out<=RT_Address;
        RD_Address_out<=RD_Address;
        RegWrite_out<=RegWrite;
        MemtoReg_out<=MemtoReg;
        MemRead_out<=MemRead;
        MemWrite_out<=MemWrite;
        ALUOp_out<=ALUOp;
        RegDst_out<=RegDst;
        ALUSrc_out<=ALUSrc;
end
```



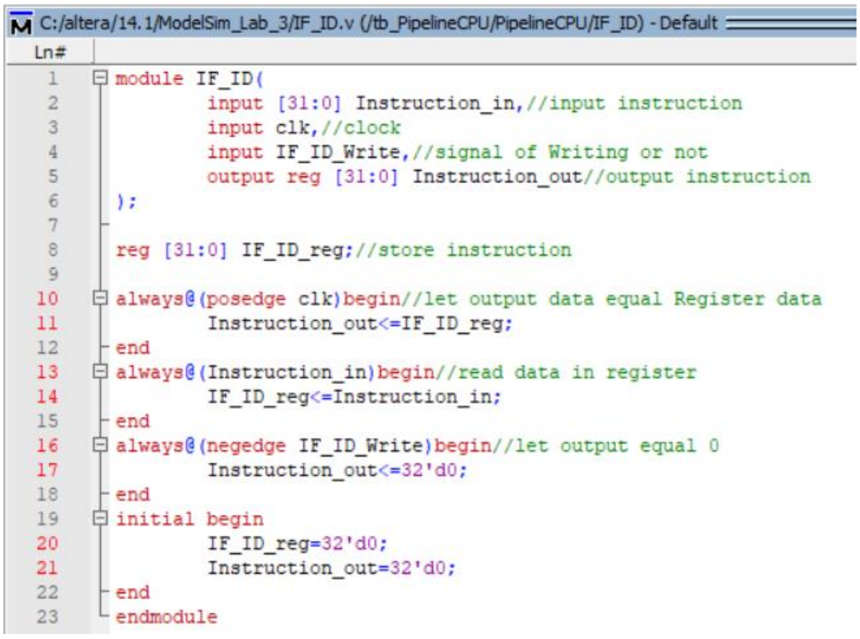
```
always@(RS_data_in or RT_data_in or immediate_in or shamt_in or funct_in or RS_Address_in or RT_Address_in or RD_Address_in or Ctrl)begin//read data in register
        RS_data<=RS_data_in;
        RT_data<=RT_data_in;
        immediate<=immediate_in;
        shamt<=shamt_in;
        funct<=funct_in;
        RS_Address<=RS_Address_in;
        RT_Address<=RT_Address_in;
        RD_Address<=RD_Address_in;
        RegWrite<=Ctrl[8];
        MemtoReg<=Ctrl[7];
        MemRead<=Ctrl[6];
        MemWrite<=Ctrl[5];
        ALUOp<=Ctrl[4:2];
        RegDst<=Ctrl[1];
        ALUSrc<=Ctrl[0];
end
```



```
initial begin
		RS_data=32'd0;
		RT_data=32'd0;
		immediate=32'd0;
		shamt=5'd0;
		funct=6'd0;
		RS_Address=5'd0;
		RT_Address=5'd0;
		RD_Address=5'd0;
		RegWrite=0;
		MemtoReg=0;
		MemRead=0;
		MemWrite=0;
		ALUOp=3'd0;
		RegDst=0;
		ALUSrc=0;
		RS_data_out=32'd0;
		RT_data_out=32'd0;
		immediate_out=32'd0;
		shamt_out=5'd0;
		funct_out=6'd0;
		RS_Address_out=5'd0;
		RT_Address_out=5'd0;
		RD_Address_out=5'd0;
		RegWrite_out=0;
		MemtoReg_out=0;
		MemRead_out=0;
		MemWrite_out=0;
		ALUOp_out=3'd0;
		RegDst_out=0;
		ALUSrc_out=0;
end
endmodule
```

這部分與剛剛的 EX_MEM 大同小異，而程式碼看起來比較多是因為這部分屬於比較前面，所以有很多的資料還沒處理以及訊號線

j.IF_ID

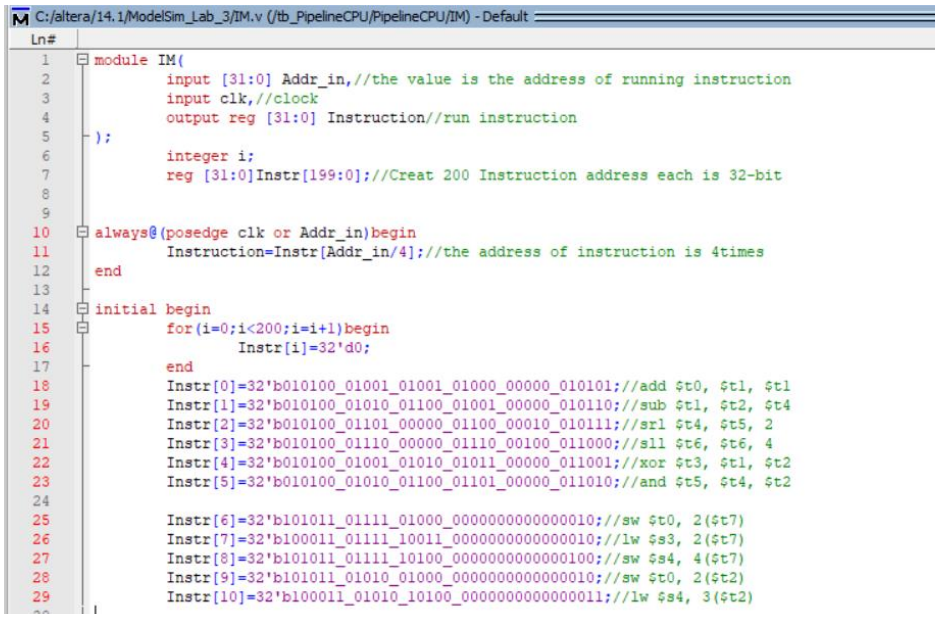


C:/altera/14.1/ModelSim_Lab_3/IF_ID.v (/tb_PipelineCPU/PipelineCPU/IF_ID) - Default

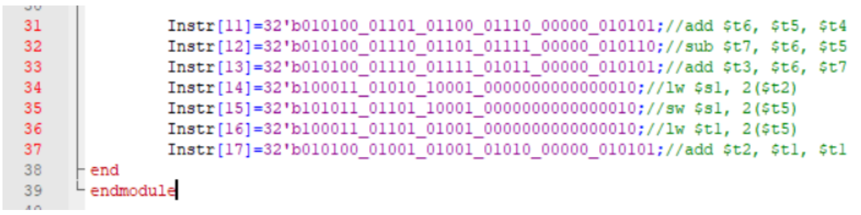
```
module IF_ID(
		input [31:0] Instruction_in,//input instruction
		input clk,//clock
		input IF_ID_Write,//signal of Writing or not
		output reg [31:0] Instruction_out//output instruction
);

reg [31:0] IF_ID_reg;//store instruction

always@(posedge clk)begin//let output data equal Register data
		Instruction_out<=IF_ID_reg;
end
always@(Instruction_in)begin//read data in register
		IF_ID_reg<=Instruction_in;
end
always@(negedge IF_ID_Write)begin//let output equal 0
		Instruction_out<=32'd0;
end
initial begin
		IF_ID_reg=32'd0;
		Instruction_out=32'd0;
end
endmodule
```

這部分算是滿重要的 register，他跟上面兩個大同小異，只差在因為 stall 的時候我必須將這個 register 清空，以免在後面的 forwarding 會有出錯的機率

k.IM


C:/altera/14.1/ModelSim_Lab_3/IM.v (/tb_PipelineCPU/PipelineCPU/IM) - Default

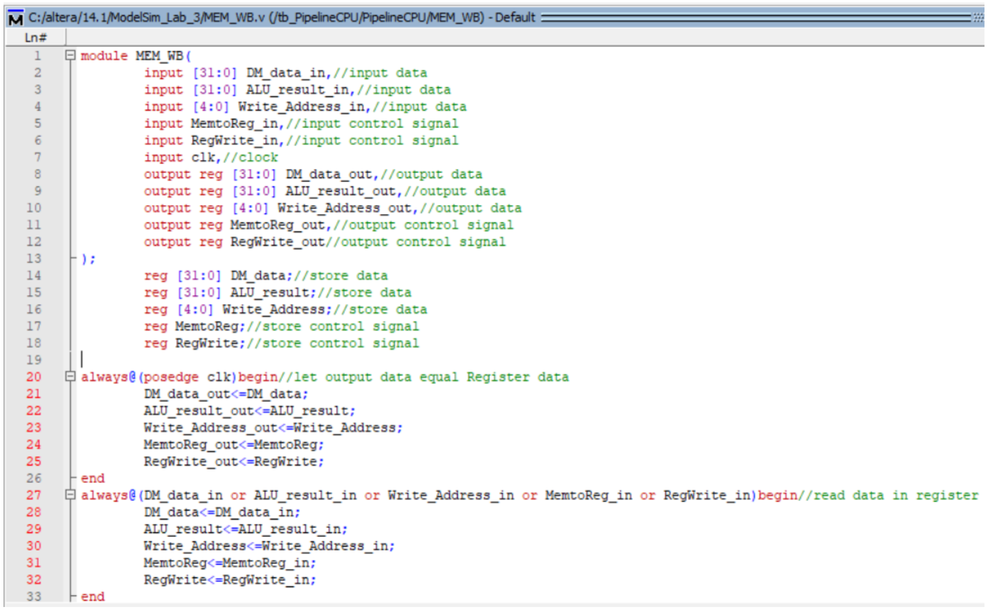
```
module IM(
        input [31:0] Addr_in,//the value is the address of running instruction
        input clk,//clock
        output reg [31:0] Instruction//run instruction
);
        integer i;
        reg [31:0]Instr[199:0];//Creat 200 Instruction address each is 32-bit


always@(posedge clk or Addr_in)begin
        Instruction=Instr[Addr_in/4];//the address of instruction is 4times
end

initial begin
        for(i=0;i<200;i=i+1)begin
                Instr[i]=32'd0;
        end
        Instr[0]=32'b010100_01001_01001_01000_00000_010101;//add $t0, $t1, $t1
        Instr[1]=32'b010100_01010_01100_01001_00000_010110;//sub $t1, $t2, $t4
        Instr[2]=32'b010100_01101_00000_01100_00010_010111;//srl $t4, $t5, 2
        Instr[3]=32'b010100_01110_00000_01110_00100_011000;//sll $t6, $t6, 4
        Instr[4]=32'b010100_01001_01010_01011_00000_011001;//xor $t3, $t1, $t2
        Instr[5]=32'b010100_01010_01100_01101_00000_011010;//and $t5, $t4, $t2

        Instr[6]=32'b101011_01111_01000_0000000000000010;//sw $t0, 2($t7)
        Instr[7]=32'b100011_01111_10011_0000000000000010;//lw $s3, 2($t7)
        Instr[8]=32'b101011_01111_10100_0000000000000100;//sw $s4, 4($t7)
        Instr[9]=32'b101011_01010_01000_0000000000000010;//sw $t0, 2($t2)
        Instr[10]=32'b100011_01010_10100_0000000000000011;//lw $s4, 3($t2)
```



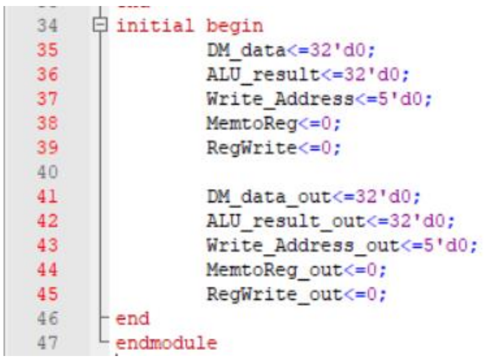
```
        Instr[11]=32'b010100_01101_01100_01110_00000_010101;//add $t6, $t5, $t4
        Instr[12]=32'b010100_01110_01101_01111_00000_010110;//sub $t7, $t6, $t5
        Instr[13]=32'b010100_01110_01111_01011_00000_010101;//add $t3, $t6, $t7
        Instr[14]=32'b100011_01010_10001_0000000000000010;//lw $s1, 2($t2)
        Instr[15]=32'b101011_01101_10001_0000000000000010;//sw $s1, 2($t5)
        Instr[16]=32'b100011_01101_01001_0000000000000010;//lw $t1, 2($t5)
        Instr[17]=32'b010100_01001_01001_01010_00000_010101;//add $t2, $t1, $t1
end
endmodule
```

這部分與上次的幾乎一模一樣，只修改了 Instruction 的部分

l.MEM_WB

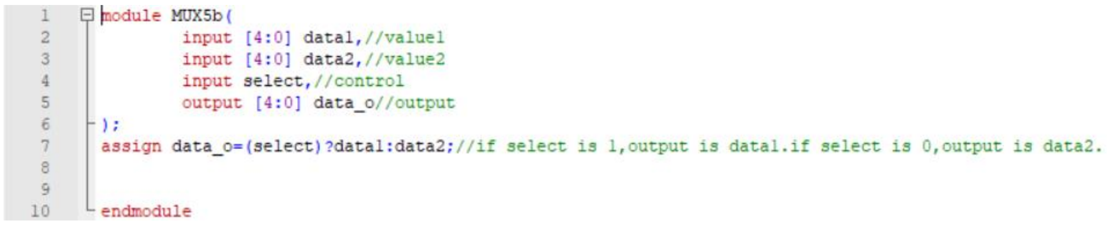

C:/altera/14.1/ModelSim_Lab_3/MEM_WB.v (/tb_PipelineCPU/PipelineCPU/MEM_WB) - Default

```
module MEM_WB(
        input [31:0] DM_data_in,//input data
        input [31:0] ALU_result_in,//input data
        input [4:0] Write_Address_in,//input data
        input MemtoReg_in,//input control signal
        input RegWrite_in,//input control signal
        input clk,//clock
        output reg [31:0] DM_data_out,//output data
        output reg [31:0] ALU_result_out,//output data
        output reg [4:0] Write_Address_out,//output data
        output reg MemtoReg_out,//output control signal
        output reg RegWrite_out//output control signal
);
        reg [31:0] DM_data;//store data
        reg [31:0] ALU_result;//store data
        reg [4:0] Write_Address;//store data
        reg MemtoReg;//store control signal
        reg RegWrite;//store control signal

always@(posedge clk)begin//let output data equal Register data
        DM_data_out<=DM_data;
        ALU_result_out<=ALU_result;
        Write_Address_out<=Write_Address;
        MemtoReg_out<=MemtoReg;
        RegWrite_out<=RegWrite;
end
always@(DM_data_in or ALU_result_in or Write_Address_in or MemtoReg_in or RegWrite_in)begin//read data in register
        DM_data<=DM_data_in;
        ALU_result<=ALU_result_in;
        Write_Address<=Write_Address_in;
        MemtoReg<=MemtoReg_in;
        RegWrite<=RegWrite_in;
end
```


```
initial begin
        DM_data<=32'd0;
        ALU_result<=32'd0;
        Write_Address<=5'd0;
        MemtoReg<=0;
        RegWrite<=0;

        DM_data_out<=32'd0;
        ALU_result_out<=32'd0;
        Write_Address_out<=5'd0;
        MemtoReg_out<=0;
        RegWrite_out<=0;
end
endmodule
```

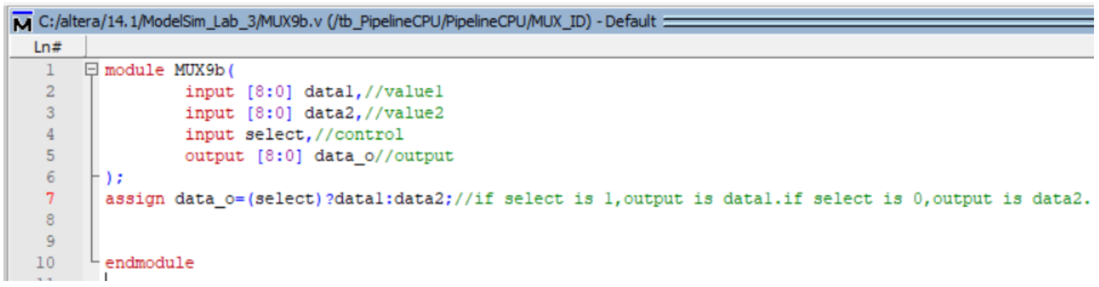
這部分就與前面的 register 一樣，就是為了 pipeline 使用，不讓資料繼續走下去

m.MUX5b


```
module MUX5b(
        input [4:0] data1,//value1
        input [4:0] data2,//value2
        input select,//control
        output [4:0] data_o//output
);
assign data_o=(select)?data1:data2;//if select is 1,output is data1.if select is 0,output is data2.


endmodule
```

這部分與上次 module 相同

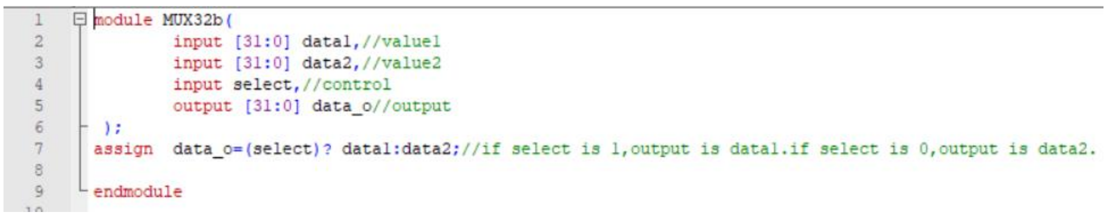
n. MUX5b


C:/altera/14.1/ModelSim_Lab_3/MUX9b.v (/tb_PipelineCPU/PipelineCPU/MUX_ID) - Default

```
module MUX9b(
        input [8:0] data1,//value1
        input [8:0] data2,//value2
        input select,//control
        output [8:0] data_o//output
);
assign data_o=(select)?data1:data2;//if select is 1,output is data1.if select is 0,output is data2.


endmodule
```

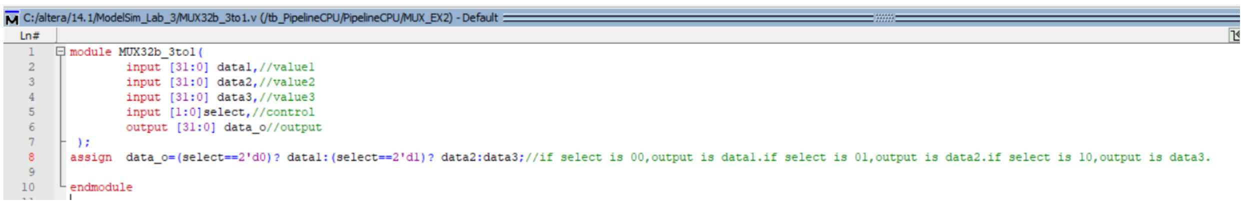
這部分跟上次 module 差不多，只差在 bit 數不同

o.MUX32b


```
module MUX32b(
        input [31:0] data1,//value1
        input [31:0] data2,//value2
        input select,//control
        output [31:0] data_o//output
 );
assign  data_o=(select)? data1:data2;//if select is 1,output is data1.if select is 0,output is data2.

endmodule
```

這部分與上次 module 相同

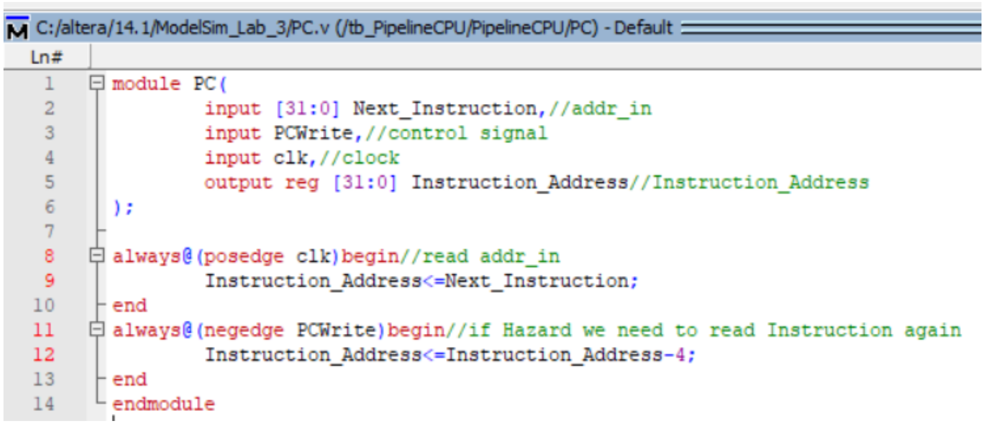
p. MUX32b_3to1


C:/altera/14.1/ModelSim_Lab_3/MUX32b_3to1.v (/tb_PipelineCPU/PipelineCPU/MUX_EX2) - Default

```
module MUX32b_3to1(
        input [31:0] data1,//value1
        input [31:0] data2,//value2
        input [31:0] data3,//value3
        input [1:0]select,//control
        output [31:0] data_o//output
 );
assign  data_o=(select==2'd0)? data1:(select==2'd1)? data2:data3;//if select is 00,output is data1.if select is 01,output is data2.if select is 10,output is data3.

endmodule
```

這部分跟上次 module 也差不多，只是 MUX 選擇線變多

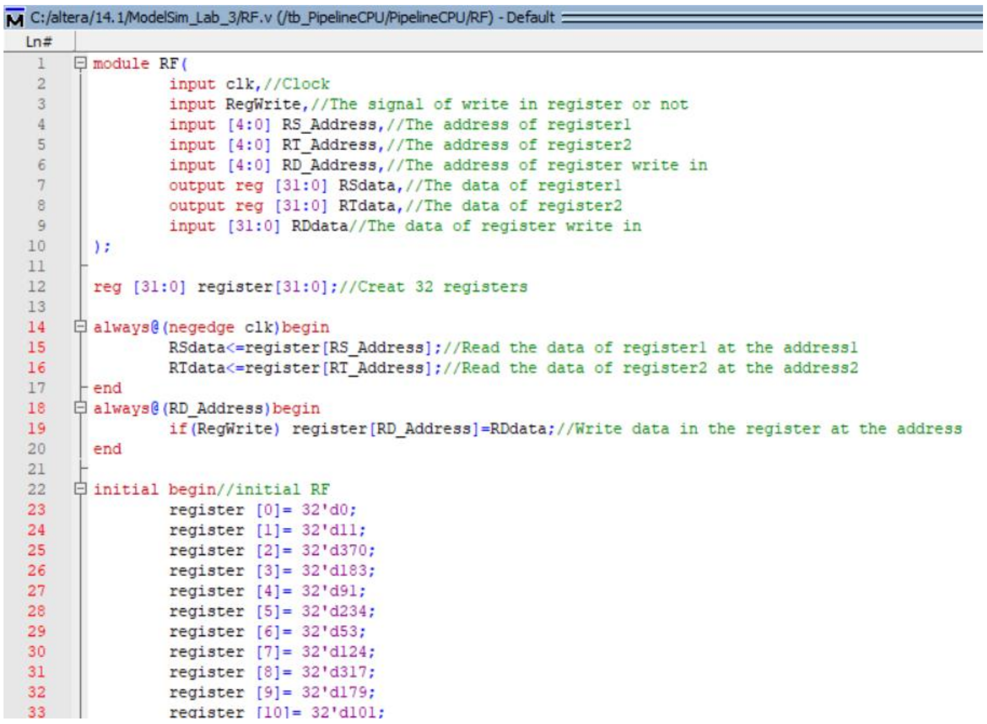
q.PC


C:/altera/14.1/ModelSim_Lab_3/PC.v (/tb_PipelineCPU/PipelineCPU/PC) - Default

```
module PC(
        input [31:0] Next_Instruction,//addr_in
        input PCWrite,//control signal
        input clk,//clock
        output reg [31:0] Instruction_Address//Instruction_Address
);

always@(posedge clk)begin//read addr_in
        Instruction_Address<=Next_Instruction;
end
always@(negedge PCWrite)begin//if Hazard we need to read Instruction again
        Instruction_Address<=Instruction_Address-4;
end
endmodule
```

這部分是 Program Counter，基本上就是加 4，但遇到 stall 時就要處理，我方法是 output-4 這樣也可以達到 stall 的效果

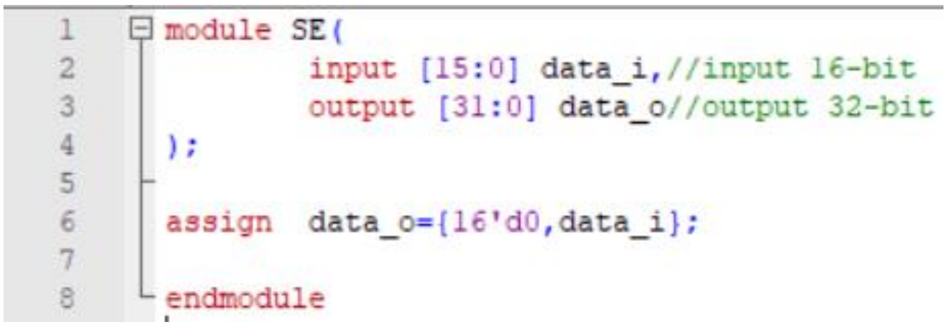
r.RF


C:/altera/14.1/ModelSim_Lab_3/RF.v (/tb_PipelineCPU/PipelineCPU/RF) - Default

```
module RF(
        input clk,//Clock
        input RegWrite,//The signal of write in register or not
        input [4:0] RS_Address,//The address of register1
        input [4:0] RT_Address,//The address of register2
        input [4:0] RD_Address,//The address of register write in
        output reg [31:0] RSdata,//The data of register1
        output reg [31:0] RTdata,//The data of register2
        input [31:0] RDdata//The data of register write in
);

reg [31:0] register[31:0];//Creat 32 registers

always@(negedge clk)begin
        RSdata<=register[RS_Address];//Read the data of register1 at the address1
        RTdata<=register[RT_Address];//Read the data of register2 at the address2
end
always@(RD_Address)begin
        if(RegWrite) register[RD_Address]=RDdata;//Write data in the register at the address
end

initial begin//initial RF
        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;
```

這部分與上次相同

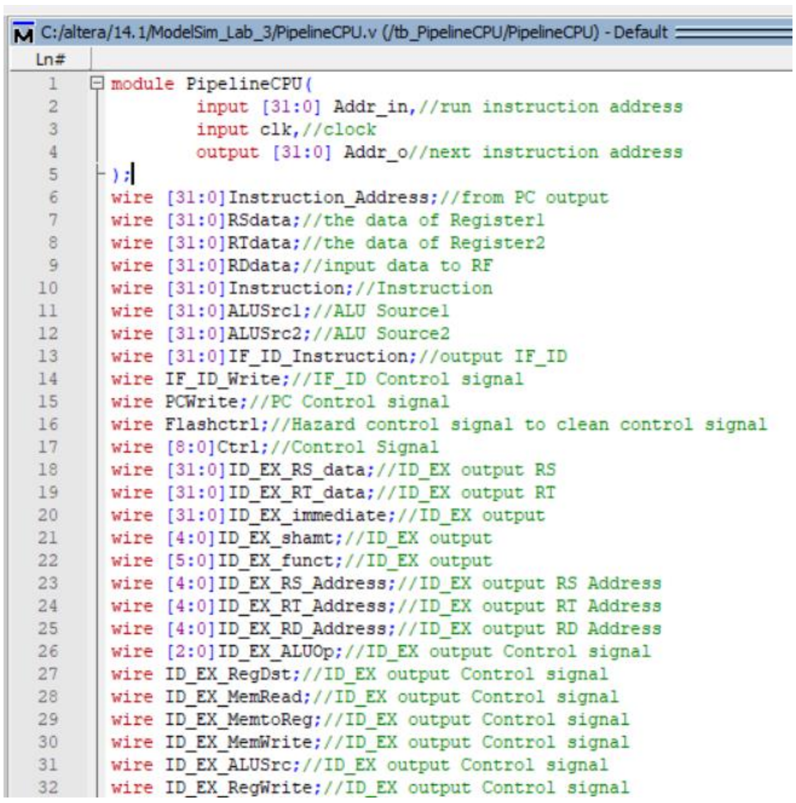
s.SE



```
module SE(
        input [15:0] data_i,//input 16-bit
        output [31:0] data_o//output 32-bit
);

assign  data_o={16'd0,data_i};

endmodule
```

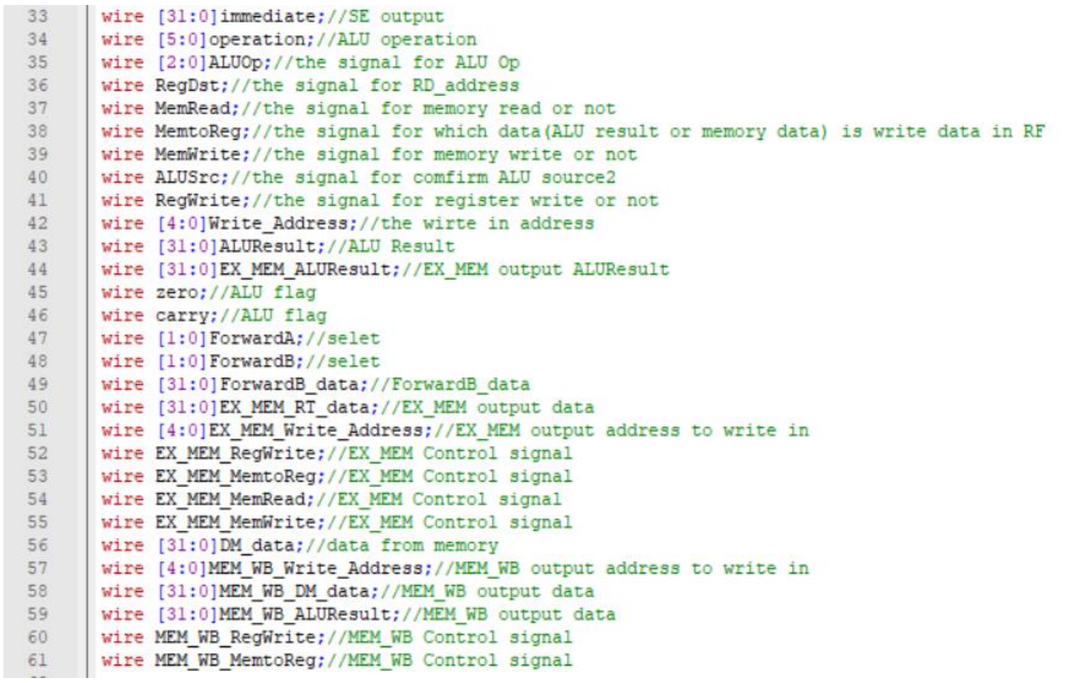
這部分與上次相同

t. PipelineCPU

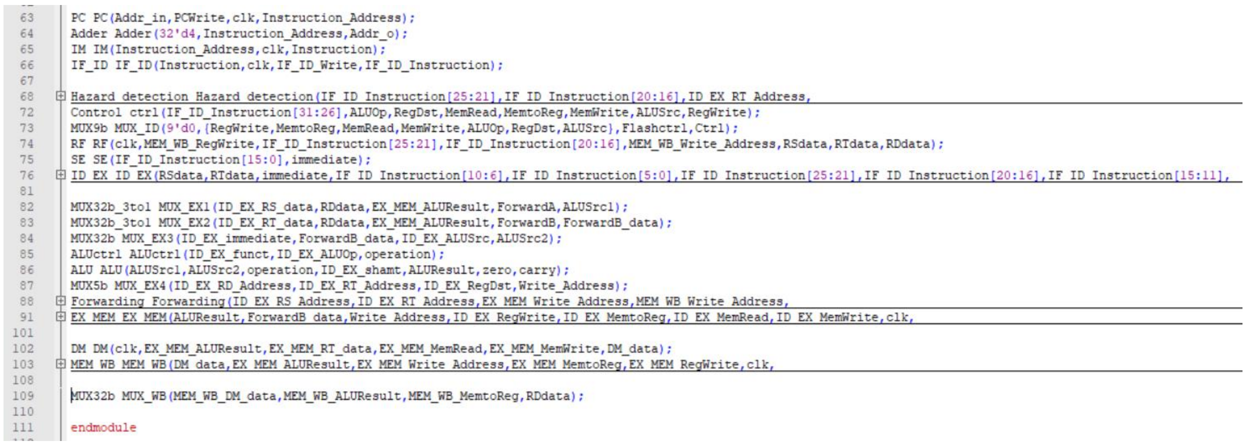


C:/altera/14.1/ModelSim_Lab_3/PipelineCPU.v (/tb_PipelineCPU/PipelineCPU) - Default

```
module PipelineCPU(
        input [31:0] Addr_in,//run instruction address
        input clk,//clock
        output [31:0] Addr_o//next instruction address
);
wire [31:0]Instruction_Address;//from PC output
wire [31:0]RSdata;//the data of Register1
wire [31:0]RTdata;//the data of Register2
wire [31:0]RDdata;//input data to RF
wire [31:0]Instruction;//Instruction
wire [31:0]ALUSrc1;//ALU Source1
wire [31:0]ALUSrc2;//ALU Source2
wire [31:0]IF_ID_Instruction;//output IF_ID
wire IF_ID_Write;//IF_ID Control signal
wire PCWrite;//PC Control signal
wire Flashctrl;//Hazard control signal to clean control signal
wire [8:0]Ctrl;//Control Signal
wire [31:0]ID_EX_RS_data;//ID_EX output RS
wire [31:0]ID_EX_RT_data;//ID_EX output RT
wire [31:0]ID_EX_immediate;//ID_EX output
wire [4:0]ID_EX_shamt;//ID_EX output
wire [5:0]ID_EX_funct;//ID_EX output
wire [4:0]ID_EX_RS_Address;//ID_EX output RS Address
wire [4:0]ID_EX_RT_Address;//ID_EX output RT Address
wire [4:0]ID_EX_RD_Address;//ID_EX output RD Address
wire [2:0]ID_EX_ALUOp;//ID_EX output Control signal
wire ID_EX_RegDst;//ID_EX output Control signal
wire ID_EX_MemRead;//ID_EX output Control signal
wire ID_EX_MemtoReg;//ID_EX output Control signal
wire ID_EX_MemWrite;//ID_EX output Control signal
wire ID_EX_ALUSrc;//ID_EX output Control signal
wire ID_EX_RegWrite;//ID_EX output Control signal
```



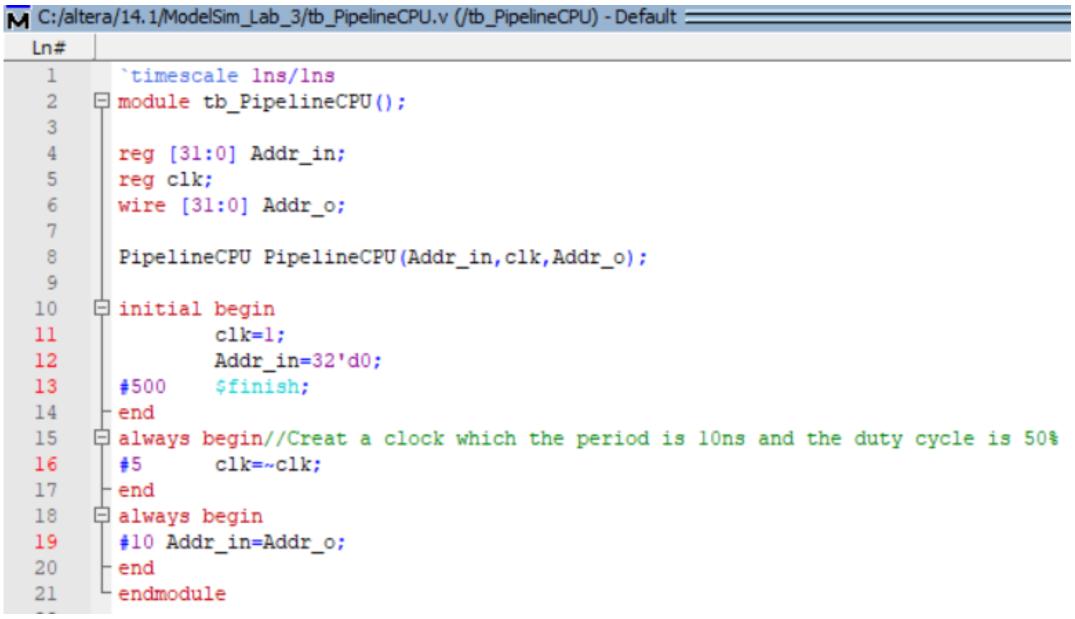
```verilog
wire [31:0]immediate;//SE output
wire [5:0]operation;//ALU operation
wire [2:0]ALUOp;//the signal for ALU Op
wire RegDst;//the signal for RD_address
wire MemRead;//the signal for memory read or not
wire MemtoReg;//the signal for which data(ALU result or memory data) is write data in RF
wire MemWrite;//the signal for memory write or not
wire ALUSrc;//the signal for comfirm ALU source2
wire RegWrite;//the signal for register write or not
wire [4:0]Write_Address;//the wirte in address
wire [31:0]ALUResult;//ALU Result
wire [31:0]EX_MEM_ALUResult;//EX_MEM output ALUResult
wire zero;//ALU flag
wire carry;//ALU flag
wire [1:0]ForwardA;//selet
wire [1:0]ForwardB;//selet
wire [31:0]ForwardB_data;//ForwardB_data
wire [31:0]EX_MEM_RT_data;//EX_MEM output data
wire [4:0]EX_MEM_Write_Address;//EX_MEM output address to write in
wire EX_MEM_RegWrite;//EX_MEM Control signal
wire EX_MEM_MemtoReg;//EX_MEM Control signal
wire EX_MEM_MemRead;//EX_MEM Control signal
wire EX_MEM_MemWrite;//EX_MEM Control signal
wire [31:0]DM_data;//data from memory
wire [4:0]MEM_WB_Write_Address;//MEM_WB output address to write in
wire [31:0]MEM_WB_DM_data;//MEM_WB output data
wire [31:0]MEM_WB_ALUResult;//MEM_WB output data
wire MEM_WB_RegWrite;//MEM_WB Control signal
wire MEM_WB_MemtoReg;//MEM_WB Control signal
```



```verilog
PC PC(Addr_in,PCWrite,clk,Instruction_Address);
Adder Adder(32'd4,Instruction_Address,Addr_o);
IM IM(Instruction_Address,clk,Instruction);
IF_ID IF_ID(Instruction,clk,IF_ID_Write,IF_ID_Instruction);

Hazard_detection Hazard_detection(IF_ID_Instruction[25:21],IF_ID_Instruction[20:16],ID_EX_RT_Address,
Control ctrl(IF_ID_Instruction[31:26],ALUOp,RegDst,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite);
MUX9b MUX_ID(9'd0,{RegWrite,MemtoReg,MemRead,MemWrite,ALUOp,RegDst,ALUSrc},Flashctrl,Ctrl);
RF RF(clk,MEM_WB_RegWrite,IF_ID_Instruction[25:21],IF_ID_Instruction[20:16],MEM_WB_Write_Address,RSdata,RTdata,RDdata);
SE SE(IF_ID_Instruction[15:0],immediate);
ID_EX ID_EX(RSdata,RTdata,immediate,IF_ID_Instruction[10:6],IF_ID_Instruction[5:0],IF_ID_Instruction[25:21],IF_ID_Instruction[20:16],IF_ID_Instruction[15:11],

MUX32b_3to1 MUX_EX1(ID_EX_RS_data,RDdata,EX_MEM_ALUResult,ForwardA,ALUSrc1);
MUX32b_3to1 MUX_EX2(ID_EX_RT_data,RDdata,EX_MEM_ALUResult,ForwardB,ForwardB_data);
MUX32b MUX_EX3(ID_EX_immediate,ForwardB_data,ID_EX_ALUSrc,ALUSrc2);
ALUctrl ALUctrl(ID_EX_funct,ID_EX_ALUOp,operation);
ALU ALU(ALUSrc1,ALUSrc2,operation,ID_EX_shamt,ALUResult,zero,carry);
MUX5b MUX_EX4(ID_EX_RD_Address,ID_EX_RT_Address,ID_EX_RegDst,Write_Address);
Forwarding Forwarding(ID_EX_RS_Address,ID_EX_RT_Address,EX_MEM_Write_Address,MEM_WB_Write_Address,
EX_MEM EX_MEM(ALUResult,ForwardB_data,Write_Address,ID_EX_RegWrite,ID_EX_MemtoReg,ID_EX_MemRead,ID_EX_MemWrite,clk,

DM DM(clk,EX_MEM_ALUResult,EX_MEM_RT_data,EX_MEM_MemRead,EX_MEM_MemWrite,DM_data);
MEM_WB MEM_WB(DM_data,EX_MEM_ALUResult,EX_MEM_Write_Address,EX_MEM_MemtoReg,EX_MEM_RegWrite,clk,

MUX32b MUX_WB(MEM_WB_DM_data,MEM_WB_ALUResult,MEM_WB_MemtoReg,RDdata);

endmodule
```

最後 PipelineCPU 就做出來了，只要按造接線圖將以上的 Module 接起來，還有一些線路的定義做好，這部分就只剩接線要接對，其他沒甚麼大問題，當然這次因為使用 pipeline 的關係所以線會很多，因此從這邊就可以知道變數名稱很重要，先不要管別人看不看得懂，但自己千萬不可以被搞亂

u. tb_PipelineCPU


C:/altera/14.1/ModelSim_Lab_3/tb_PipelineCPU.v (/tb_PipelineCPU) - Default

```
`timescale 1ns/1ns
module tb_PipelineCPU();

reg [31:0] Addr_in;
reg clk;
wire [31:0] Addr_o;

PipelineCPU PipelineCPU(Addr_in,clk,Addr_o);

initial begin
        clk=1;
        Addr_in=32'd0;
#500    $finish;
end
always begin//Creat a clock which the period is 10ns and the duty cycle is 50%
#5      clk=~clk;
end
always begin
#10 Addr_in=Addr_o;
end
endmodule
```

這是最後測試的 testbench，而寫法也很簡單，跟上次差不多，只是我改變成我習慣的正緣觸發方式