0510008 藍挺毓
0510026 陳司瑋

Computer Organization

Source code and the note:

- alu.v

```
assign zero_o = (result_o==0);
always @(ctrl_i, src1_i, src2_i) begin
    case (ctrl_i)
        0: result_o<=src1_i&src2_i;
        1: result_o<= src1_i | src2_i;
        2: result_o<=src1_i + src2_i;
        6: result_o<=src1_i - src2_i;
        7: result_o<=( $signed (src1_i) < $signed(src2_i)) ? 32'd1:32'd0 ;
        12: result_o<= ~(src1_i | src2_i);
        default: result_o<=0;
    endcase
end
```

- ALU_Ctrl.v

```
ALUCtrl_o[3]= 0;
ALUCtrl_o[2] = sub|slt|slti|beq;
ALUCtrl_o[1] = add|sub|slt|addi|slti|beq|lw|sw;
ALUCtrl_o[0] = Or|slt|slti;
```

lab2 因為 slt 和 slti 的部分被扣分數，因為沒有 slt 負數相關測資，所以 lab2 時沒有發現，而 lab3 也不太清楚要改甚麼，這次因為有相關測資，好好檢視 code 才發現是因為control 的部分是拿 lab1 的control，而alu是拿助教提供的code，他們之間 slt 的差異在於 lab1 的 slt 是用減法實現，所以 control 是直接接給減法，而在直接做 32bit alu 時，slt 有自己的 decode 代碼”7”，lab2 接給”6”所以讓 slt 會出現負數。這次做好修改了。

- Pipe_CPU_1.v

這次大部分的 code 都和之前一樣，這次主要是依照下面的 architecture diagram 接 Pipe_CPU_1.v
而這份 code 架構助教也都打好了，只是填空接線。
code 有點長，所以我只貼一小部分上來

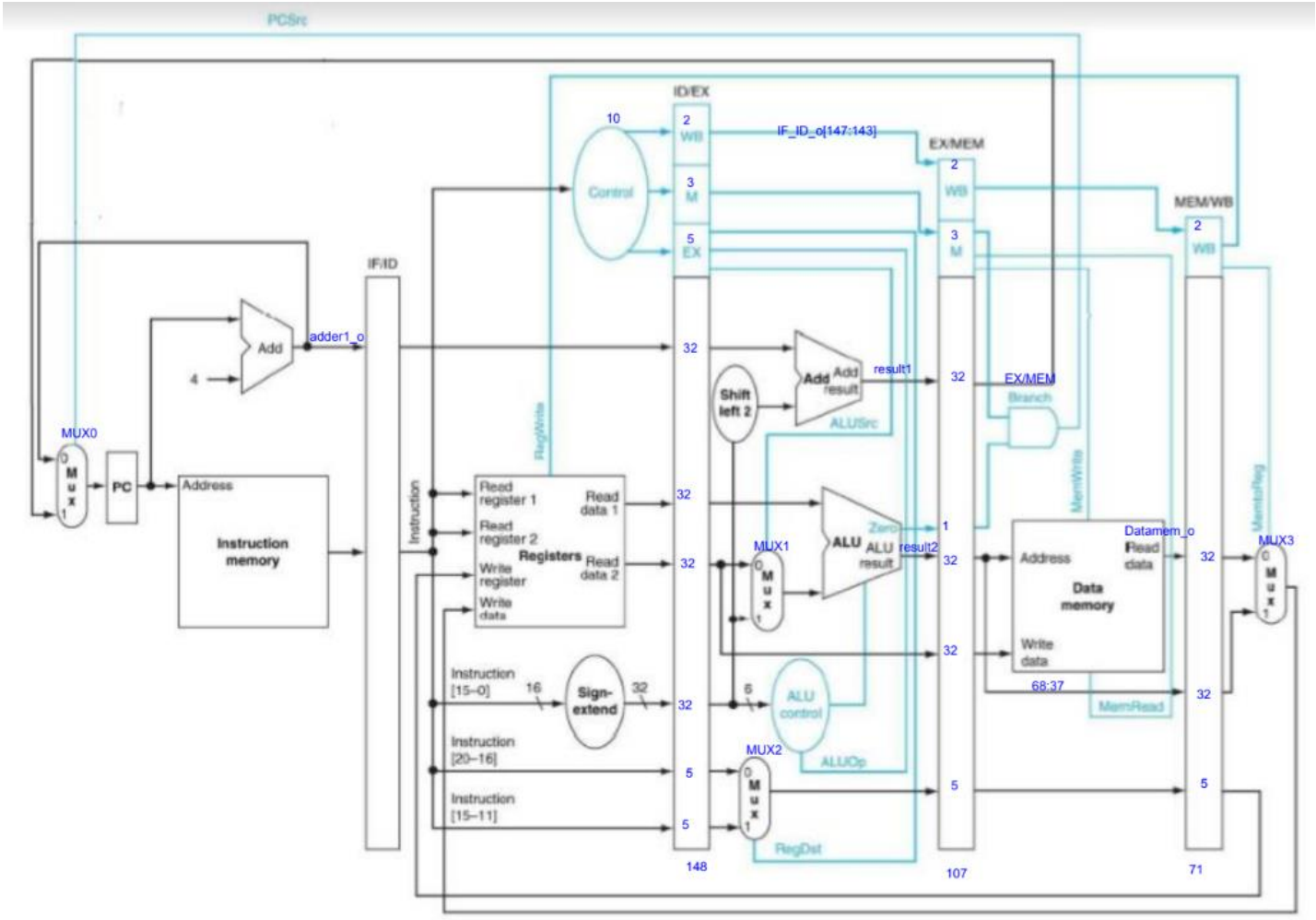
```
Pipe_Reg #(.size(148) )ID_EX(
  .clk_i(clk_i),
  .rst_i(rst_i),
  .data_i({Regwrite,MemtoReg,branch,Memread,Memwrite,RegDst,ALUOp,ALUSrc,IF_ID_o[63:32],read_data1_o,read_data2_o,extend,IF_ID_o[20:11]}),
  .data_o(ID_EX_o)//N is the total length of input/output
```

我們把東西放進 stage 中間的 register 後，就都直接用 register 編號接線，舉例如下

0510008 藍挺毓
0510026 陳司瑋

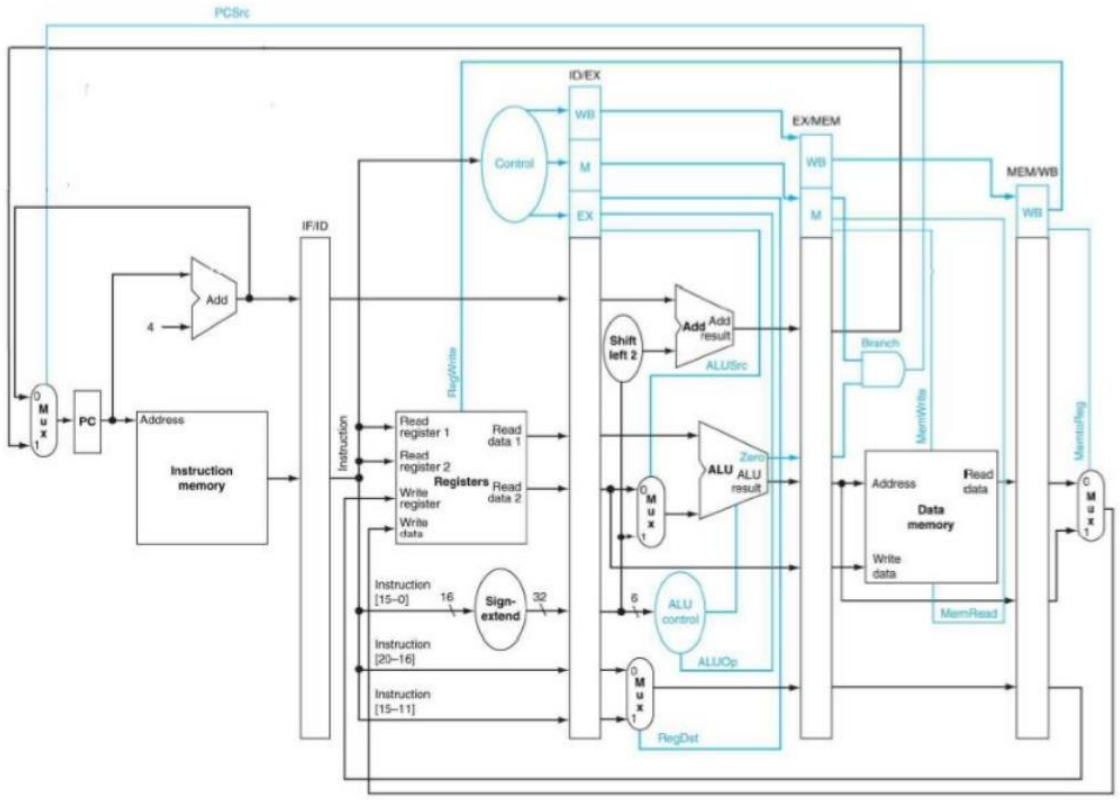
```
Data_Memory DM(
    .clk_i(clk_i),
    .addr_i(EX_MEM_o[68:37]),
    .data_i(EX_MEM_o[36:5]),
    .MemRead_i(EX_MEM_o[103]),
    .MemWrite_i(EX_MEM_o[102]),
    .data_o(Datamem_o)

);
```

這樣做的優點是 register 很清楚乾淨，不會有很多的名稱，不過缺點是數字要算得很清楚，知道哪個功能的值在第幾位，比較容易接錯線。
下圖是我們自己做的筆記，reg 上面的數字是這條線有幾位，而最下面再 reg 外的數字是整個 reg 的大小。我們依照圖的順序，照上面標記的數字安排 reg 每個位數放甚麼。



0510008 藍挺毓
0510026 陳司瑋

Architecture diagrams:



Hardware module analysis:

從上面的 Architecture diagram，與之前做的 single cycle 唯一也最明顯的差別是各個 stage 之間加了一個 register 讓不同的 instruction 在 cycle 與 cycle 帶著上一個 stage 的值，實作出 pipeline 的 module。比起上次 lab 的也少了 jump、jr、jal 的指令，所以 control 跟 mux 變得比較單純。所以 decoder 的 moudle 少了一些條件及指令。

如 Source code and the note 所述，在 Pipe_CPU_1 的 module 中，我們用四個 Pipe_Reg 的 module 分別命名為 IF_ID、ID_EX、EX_MEM、MEM_WB。再計算接進 Pipe_Reg 的 input 會是 output 的哪幾個 bits，依據 Architecture diagram 與其他 module 接線。

大部分的截圖跟說明，都有在上幾題清楚解釋。

Finished part:

CO_P4_test_1_result

0510008 藍挺毓
0510026 陳司瑋

```
m24=          0, m25=          0, m26=          0, m27=          0, m28=          0, m29=          0, m30=          0, m31=          0
Register=============================================================

r0=           0, r1=           3, r2=           4, r3=           1, r4=           6, r5=           2, r6=           7, r7=           1

r8=           1, r9=           0, r10=          3, r11=          0, r12=          0, r13=          0, r14=          0, r15=          0

r16=          0, r17=          0, r18=          0, r19=          0, r20=          0, r21=          0, r22=          0, r23=          0

r24=          0, r25=          0, r26=          0, r27=          0, r28=          0, r29=          0, r30=          0, r31=          0


Memory=============================================================

m0=           0, m1=           3, m2=           0, m3=           0, m4=           0, m5=           0, m6=           0, m7=           0

m8=           0, m9=           0, m10=          0, m11=          0, m12=          0, m13=          0, m14=          0, m15=          0

r16=          0, m17=          0, m18=          0, m19=          0, m20=          0, m21=          0, m22=          0, m23=          0

m24=          0, m25=          0, m26=          0, m27=          0, m28=          0, m29=          0, m30=          0, m31=          0
INFO: [USF-XSim-96] XSim completed. Design snapshot 'TestBench_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:13 . Memory (MB): peak = 940.426 ; gain = 0.000
```

C0_P4_test_1_result

```
Register=============================================================

r0=           0, r1=           16, r2=          20, r3=          8, r4=           16, r5=          8, r6=           24, r7=          26

r8=           8, r9=           100, r10=        0, r11=          0, r12=          0, r13=          0, r14=          0, r15=          0

r16=          0, r17=          0, r18=          0, r19=          0, r20=          0, r21=          0, r22=          0, r23=          0

r24=          0, r25=          0, r26=          0, r27=          0, r28=          0, r29=          0, r30=          0, r31=          0


Memory=============================================================

m0=           0, m1=           16, m2=          0, m3=           0, m4=           0, m5=           0, m6=           0, m7=           0

m8=           0, m9=           0, m10=          0, m11=          0, m12=          0, m13=          0, m14=          0, m15=          0

r16=          0, m17=          0, m18=          0, m19=          0, m20=          0, m21=          0, m22=          0, m23=          0

m24=          0, m25=          0, m26=          0, m27=          0, m28=          0, m29=          0, m30=          0, m31=          0
INFO: [USF-XSim-96] XSim completed. Design snapshot 'TestBench_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:03 ; elapsed = 00:00:08 . Memory (MB): peak = 940.559 ; gain = 0.133
```

Bonus:

- 更改後的順序

I1: addi $1, $0, 16
I3: addi $3, $0, 8
I10: addi $9, $0, 100
I4: sw $1, 4($0)
I5: lw $4, 4($0)
I8: addi $7, $1, 10
I7: add $6, $3, $1
I2: addi $2, $1, 4

0510008 藍挺毓
0510026 陳司瑋

I6: sub $5, $4, $3
I9: and $8, $7, $3

- 更改後的 machine code

00100000000000010000000000010000
00100000000000110000000000001000
00100000000010010000000001100100
10101100000000010000000000000100
10001100000001000000000000000100
00100000001001110000000000001010
00000000011000010011000000100000
00100000001000100000000000000100
00000000100000110010100000100010
00000000111000110100000000100100

Problems you met and solutions:

這次的 lab 不難，基本上只要照 architecture 接線就完成了，我們一開始比較不知道如何下手的是 bonus 的部分。
一開始也因為 testbench 的路徑問題，而沒讀到值，但社團有人有人提出相關的問題，所以很快解決了。
在改 machine code 的時候才發現我們其實對每個指令彼此間隔幾個 cycle 才不會有 hazard 沒有很清楚，所以我們開始一張一張 stage 畫，看每個指令中間要隔多遠，畫一畫算好 cycle 後就是把所有的 hazard 要隔的 cycle 寫出來，就很清楚 machine code 要怎麼調整了。

Division of work: (one-member teams don't need to write)

我們這次還是跟之前一樣，約幾個時間一起討論跟打 code，所以仍沒辦法劃分出明顯分工。在接線時，我們一個人打 code 一個人在 Architecture diagram 上標數字及 wire 跟 reg 的名稱，以免搞混。
Debug 也是一起 de，report 的部分是藍挺毓先做整理，而陳司瑋再做加強，或是把挺毓沒寫到的部分補完。

Summary:

一開始要寫 pipeline 的時候其實心裡有點緊張，感覺好像會複雜很多，或是很難找 bug。但真的實作的時候發現其實很單純，尤其這次沒有 jump 等的指令感

0510008 藍挺毓
0510026 陳司瑋

覺變得比較容易，只是重新把線接上去花了一點時間，其餘很順利的做完了。