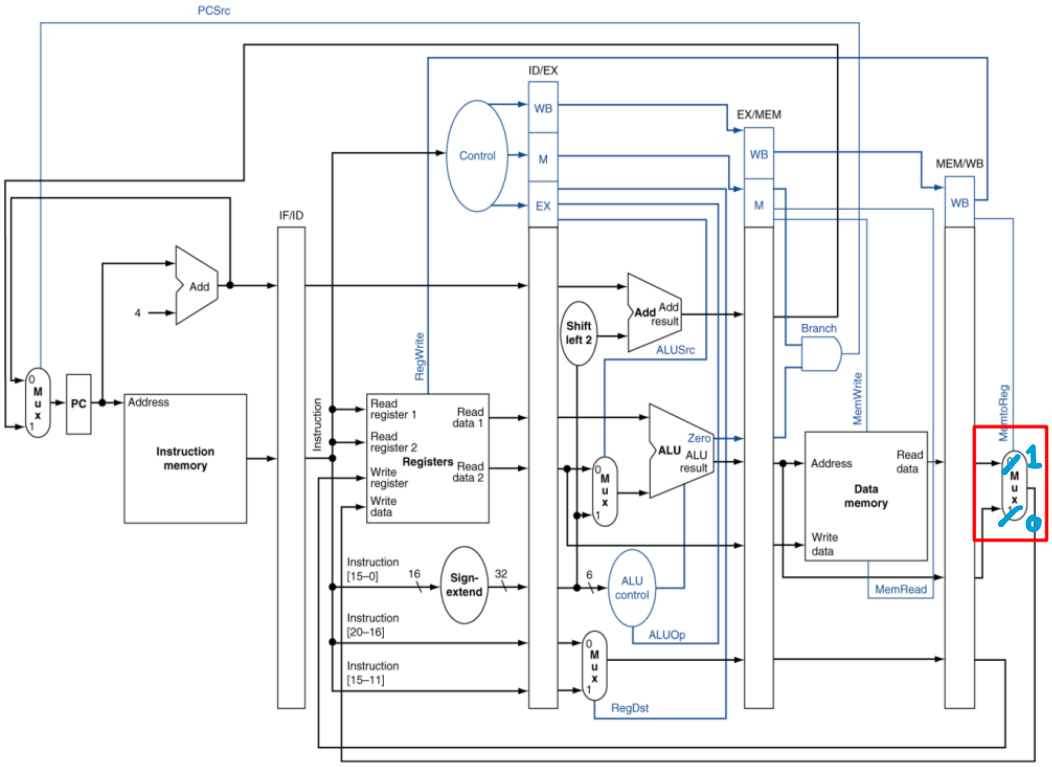
Computer Organization Lab4

ID:110550142 Name:黃芷柔

Architecture diagrams:



Hardware module analysis:

這次的 Adder, Decoder, Sign extend, Shift_Left_Two_32, MUX_2to1 都是用 lab3 時設計的。跟 Lab3 不同的是 ALU control 跟 ALU 加上 mult 和 xor 的功能，還多了 Pipe register，要把上個 stage 的資訊傳給下個 stage。

在 IF stage 有 MUX_2to1、PC、Instruction Memory 跟 Adder，進行 instruction fetching

在 ID stage 有 Decoder、Register 跟 sign extend，進行 instruction decoding 和 register reading。

在 EX stage 有 ALU、ALU Control、Adder、Shift left 2 跟 2 個 MUX，進行運算或者計算 address。

在 MEM stage 有 Data memory，對 memory 進行寫入或讀出。

在 WB stage 有一個 MUX，把結果寫回 Register。

紅框的 MUX 設計跟 SPEC 給的不同，是沿用 lab3 寫的 lab3 是 ALU result=0,Data Memory=1。

Simulation results:

Testcase 1

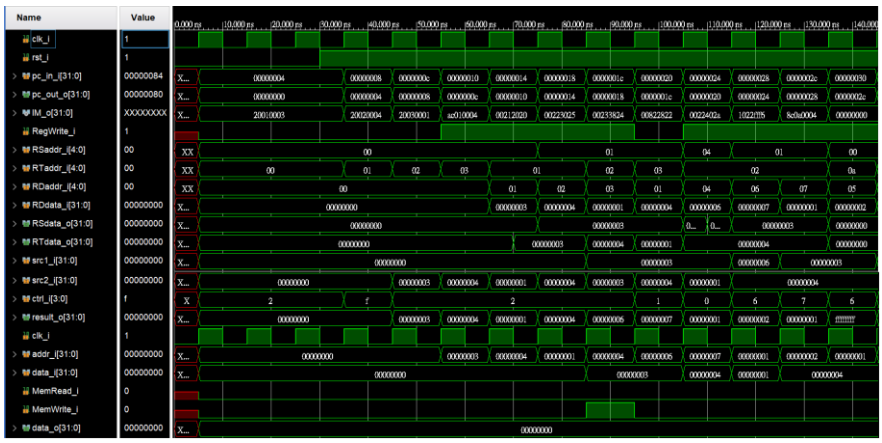
```
========== Final Result ==========
- Register File -
r0 =  0  r1 =  3  r2 =  4  r3 =  1
r4 =  6  r5 =  2  r6 =  7  r7 =  1
r8 =  1  r9 =  0 r10 =  3 r11 =  0
r12 =  0 r13 =  0 r14 =  0 r15 =  0
r16 =  0 r17 =  0 r18 =  0 r19 =  0
r20 =  0 r21 =  0 r22 =  0 r23 =  0
r24 =  0 r25 =  0 r26 =  0 r27 =  0
r28 =  0 r29 =  0 r30 =  0 r31 =  0

- Memory Data -
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
m16 =  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
```

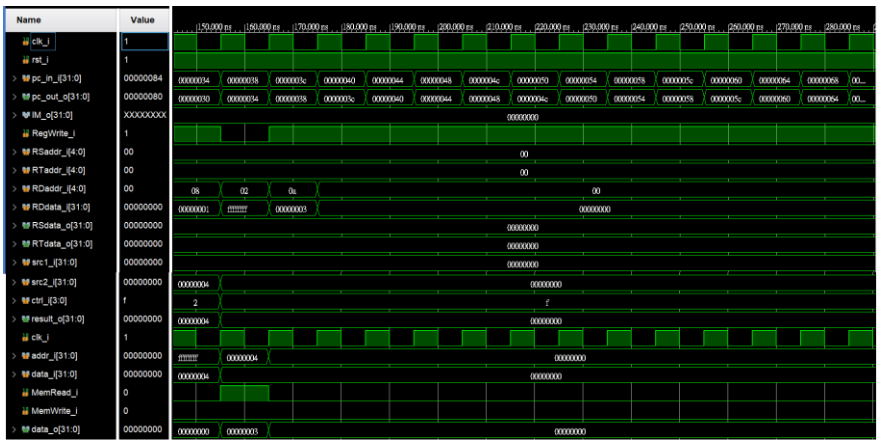
Testcase 2

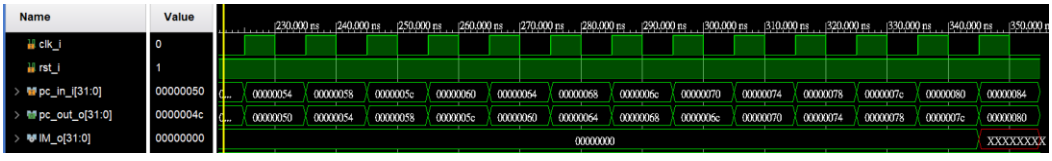
```
========== Final Result ==========
- Register File -
r0 =  0  r1 =  0  r2 =  4  r3 =  5
r4 = 49  r5 =  0  r6 =  3  r7 =  5
r8 =  1  r9 =  0 r10 =  7 r11 =  7
r12 =  0 r13 =  0 r14 =  0 r15 =  0
r16 =  0 r17 =  0 r18 =  0 r19 =  0
r20 =  0 r21 =  0 r22 =  0 r23 =  0
r24 =  0 r25 =  0 r26 =  0 r27 =  0
r28 =  0 r29 =  0 r30 =  0 r31 =  0

- Memory Data -
m0 =  0  m1 =  7  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
m16 =  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
```

和給的答案相同

Test1:(不同時間分成三張圖)

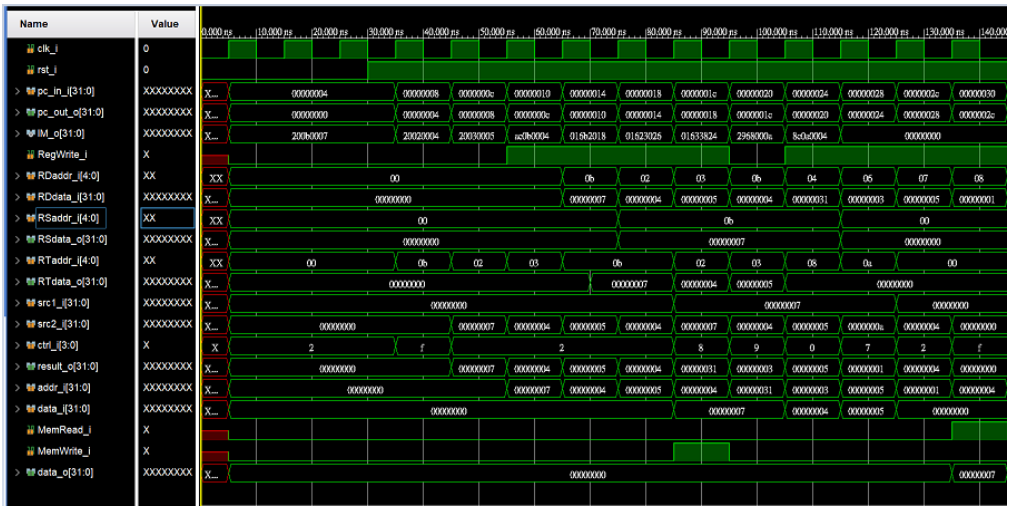


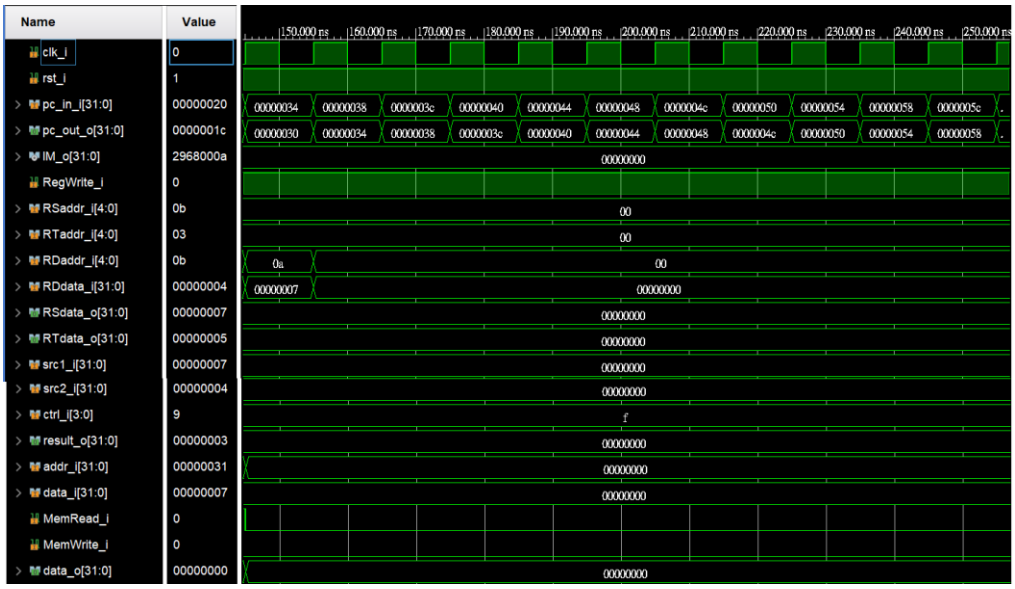


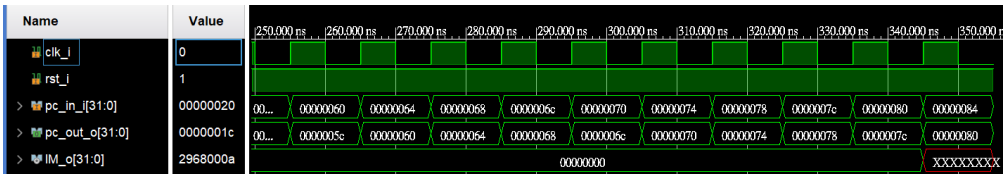


(其他項的 value 和第二張最後的值一樣)

Test2(不同時間分成三張圖)







(其他項的 value 和第二張最後的值一樣)

Bonus:

在 hazard 的兩個 instruction 中，插入兩個 instruction，優先插入後面有獨立的 instruction，沒有就插入 NOP。改完的結果如下:

addi $1, $0, 16

NOP

addi $3, $0, 8

addi $2, $1, 4

sw $1, 4($0)

lw $4, 4($0)

NOP

add $6, $3, $1

sub $5, $4, $3

addi $7, $1, 10

NOP

addi $9, $0, 100

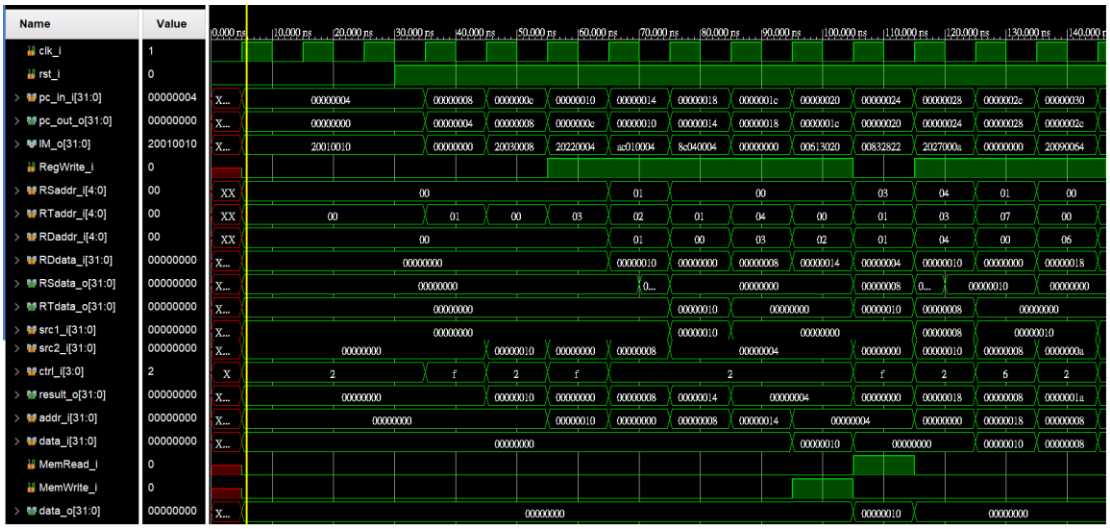
and $8, $7, $3

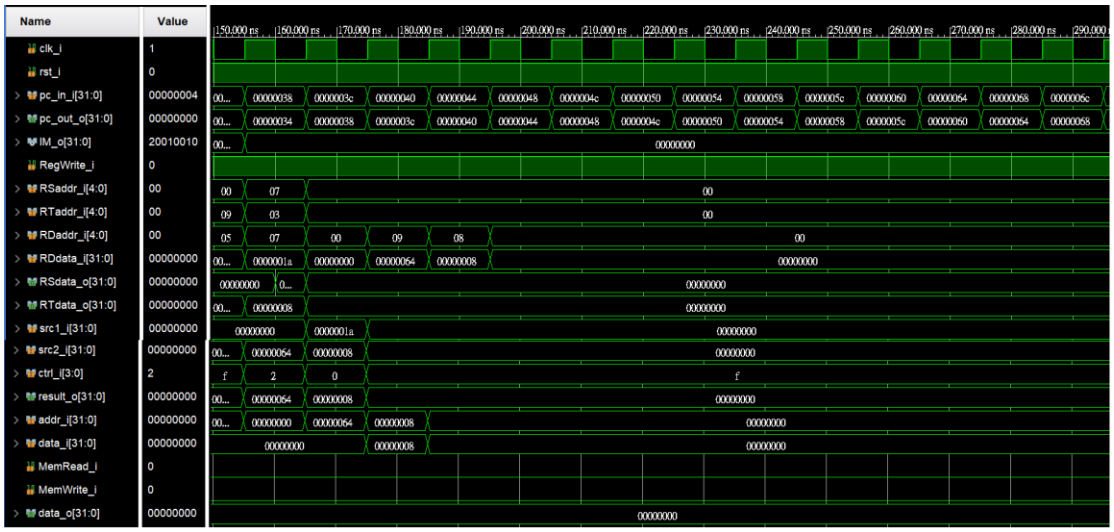
改完的 txt 檔

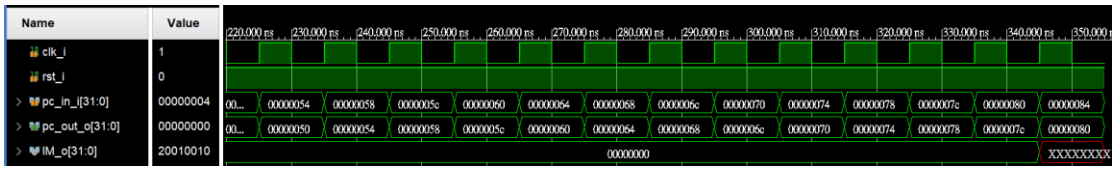
```
1  00100000000000010000000000010000
2  00000000000000000000000000000000
3  00100000000000110000000000001000
4  00100000001000100000000000000100
5  10101100000000010000000000000100
6  10001100000001000000000000000100
7  00000000000000000000000000000000
8  00000000011000010011000000100000
9  00000000100000110010100000100010
10 00100000001001110000000000001010
11 00000000000000000000000000000000
12 00100000000010010000000001100100
13 00000000111000110100000000100100
```

```
============ Final Result ============
- Register File -
 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 Data -
 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
m16 =    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
```

Waveform(不同時間分成三張圖)







(其他項的 value 和第二張最後的值一樣)

Problems you met and solutions:

結構變得複雜後，容易因為 wire 接錯或打錯字而跑不出結果，透過 SPEC 提供的 diagram，警告訊息和波形圖，檢查是哪個細節出了問題。

還有沒辦法截取完整的 Waveform，所以先截成上下兩張，用小畫家把兩張合併，原本想要全部時間軸放在同一張，但太長了看不清楚，所以分成三張。

Summary:

這次的 lab 讓我對 pipelined 的結構更加熟悉，以及如何用 NOP 跟改變 instruction 順序解決 hazard 問題，並學會如何加上 mult 跟 xor 的指令。