Final Project: Pipeling CPU Design

110 學年度第 2 學期

組別: 第 34 組

班級: 資訊二乙

組員 1: 10927207 蒲品憶

組員 2: 10927234 吳凱鈺

組員 3: 10927256 姜美羚

一、設計重點說明

利用 Single Cycle 的架構，將其切分成 5 個階段，此時每個階段將可以分別執行該階段之功能，不會互相干擾，藉由此技巧，達成 Pipeling CPU 的設計目的。

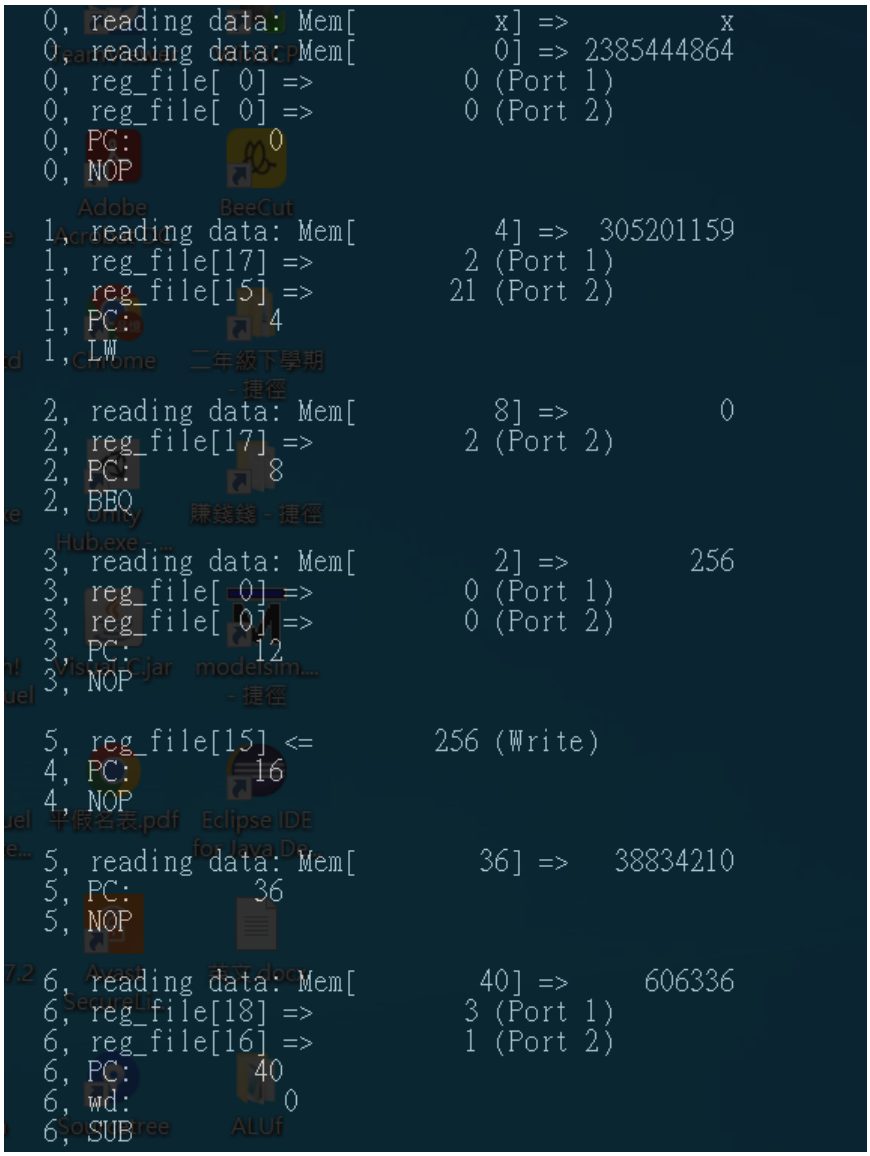
add、sub、or、and、sll、slt、divu 等 7 道指令皆依照上次的 midterm Project，但是 divu 有改，因為這次除法指令只會進來一次，使用另外寫的 count 來計算，從零開始，每做一回合就加一，直到加到 32。到達第 32 次之前都要做除法並且每次都要記錄過程中的除數、商數和餘數，直到做到第 32 次輸出計算結果，放至 HiLo 暫存器。IF/ID、ID/EX、EX/MEM 以及 MEM/WB 都是我們自己寫的，裡面會暫存 control Unit 解碼後的控制訊號，以及保留一些上一層的計算結果。這樣，當有新的指令進入時，前一指令的相關訊息才會被保留且得以傳遞到下一個階段。register file 跟 memory 是參考 CH4-2 的程式碼沒有變。由於 ori 指令需要做無號數擴充，所以我們多加了一個 Mux2，用來選擇是要有號數擴充後的值或是無號數擴充後的值，然後將選擇結果存在 ID/EX 的 register 中 ，因為要判斷訊號所以我們也多加了一個 ExtendSel 在 control Unit，用來控制是要選擇無號數擴充值或是有號數擴充值。

另外，跟課本上的 datapath 有些許不同的是，我們的 Zero 並不是從 ALU 輸出，而是另外寫一個 module 判斷是否要做跳躍指令(branch)，並將結果存到 EX/MEM 。進入下一層後，才再跟控制訊號 Branch 做 AND，告訴 PCSrc 是否要 branch。

我們將 jump Mux 放在第一層，由於我們要到第二層才會解譯指令，control unit 才會給出此指令是否為 jump 的控制訊號，送回第一層的 jump Mux，等下一次的 clk 敲起來時，才會 JUMP。

二、 結果

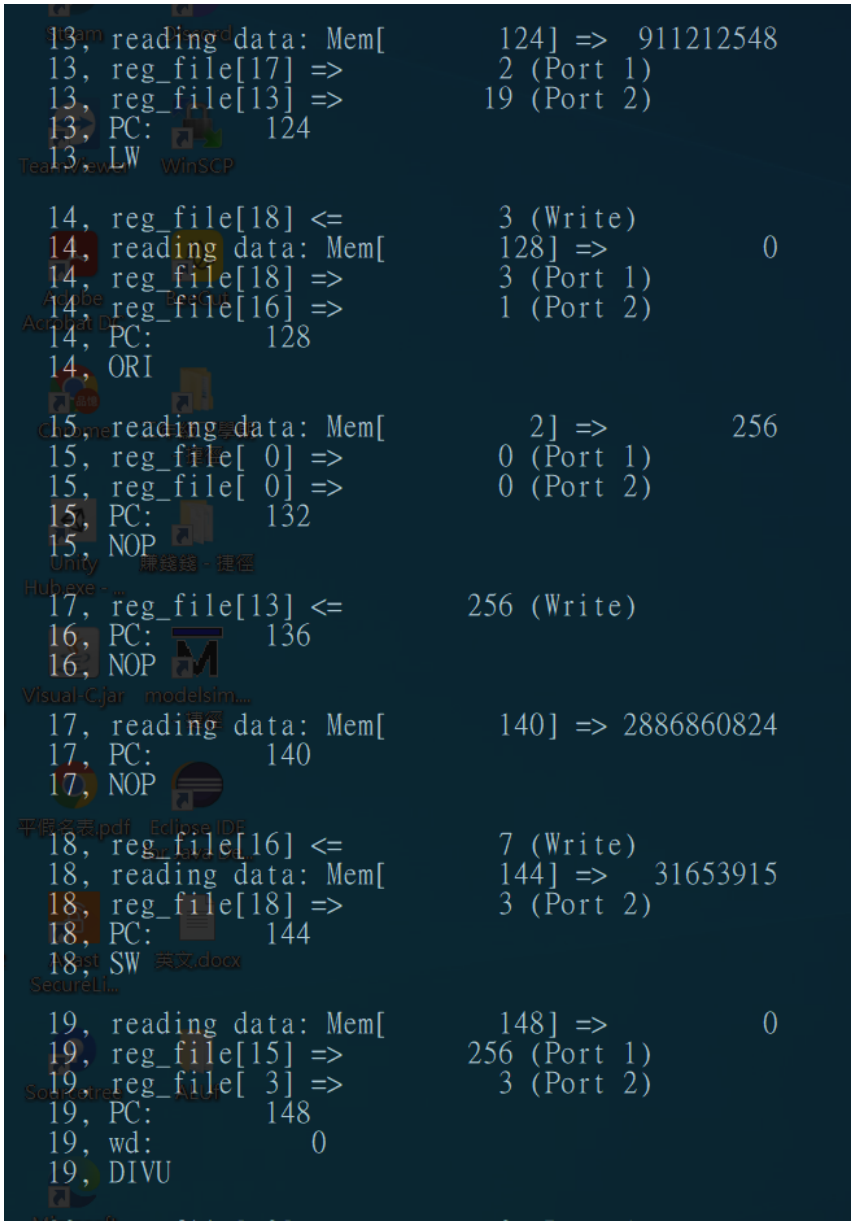
Icarus Verilog 驗證結果



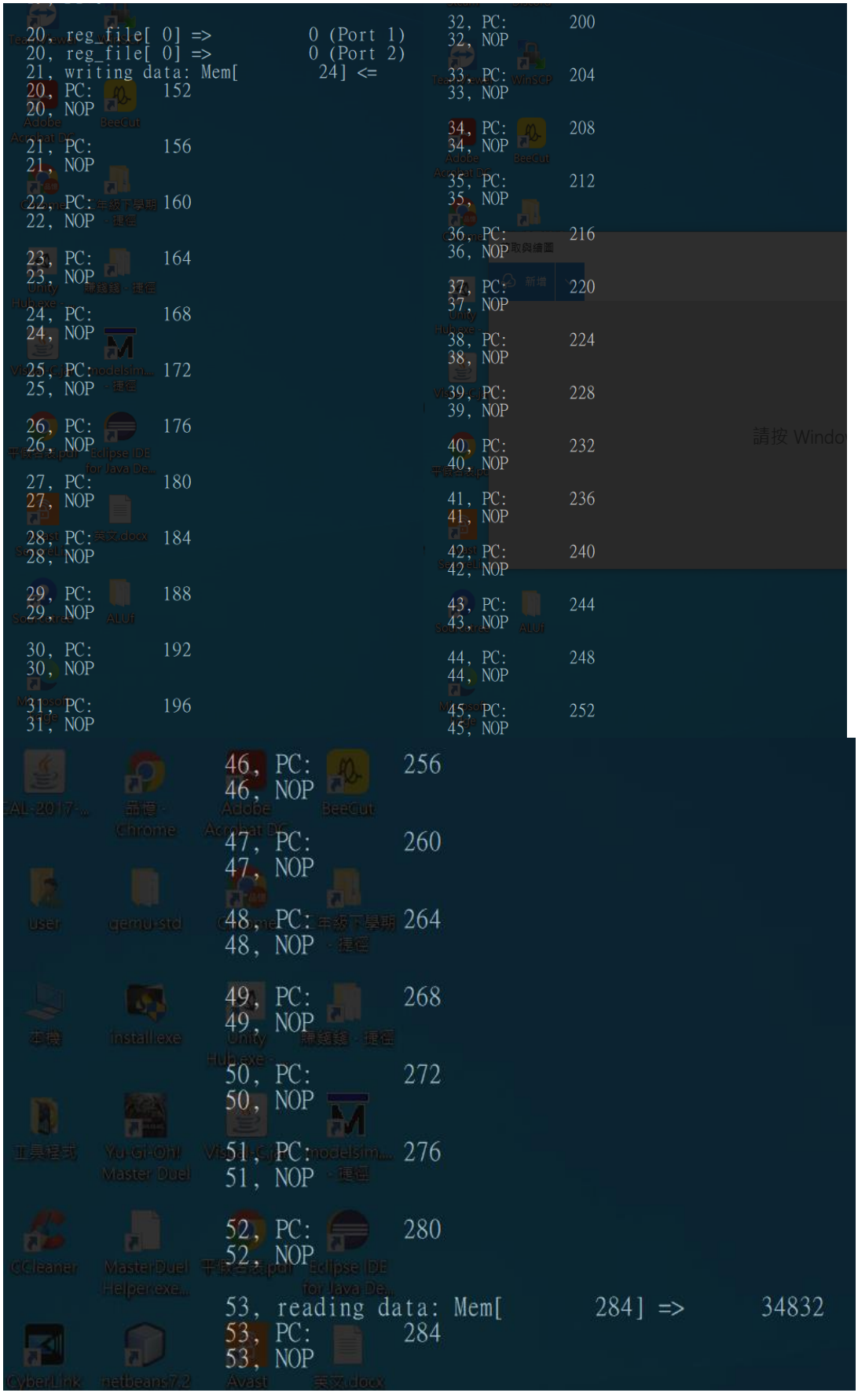
IF	ID	EX	MEM	WB	
NOP					
LW	NOP				
BEQ	LW	NOP			[因為 BEQ 會造成 DataHazards]
NOP	NOP	NOP	BEQ	LW	[所以需要三個 NOP 進來]
SUB	NOP	NOP	NOP	BEQ	[BEQ 跳到 SUB]

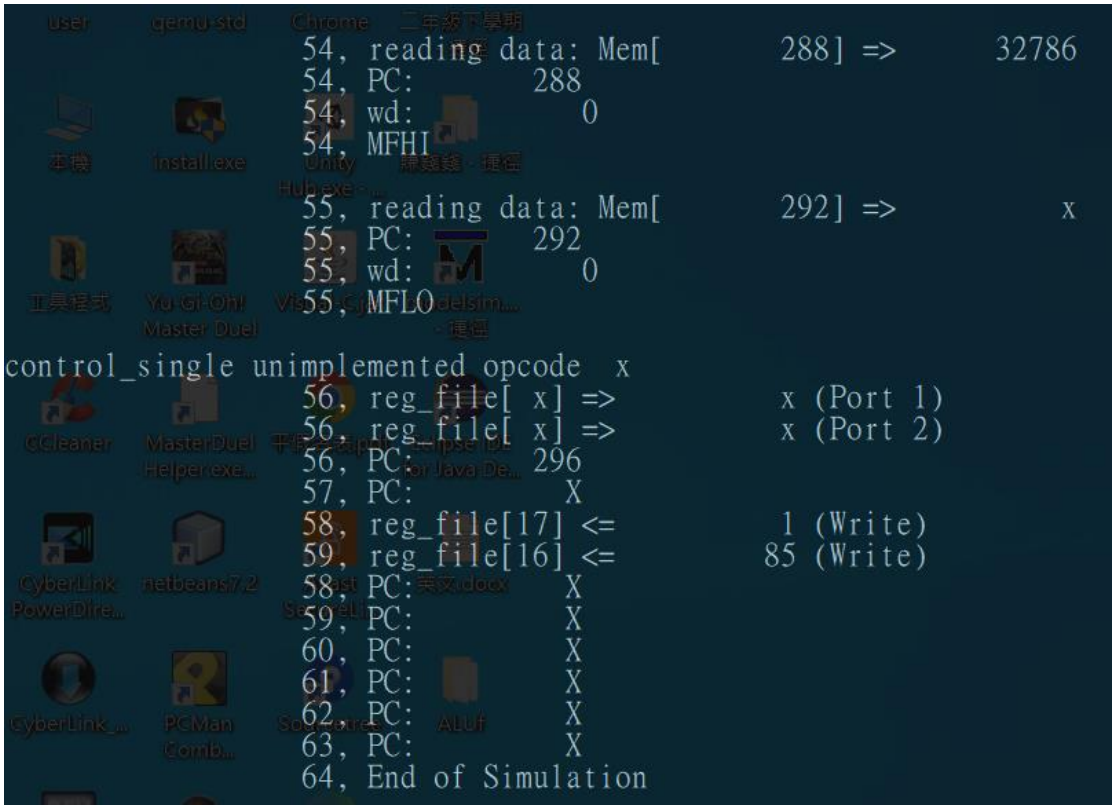
```
 7, reading data: Mem[         44] =>  134217755
 7, reg_file[ 0] =>           0 (Port 1)
 7, reg_file[ 9] =>           9 (Port 2)
 7, PC:          44
 7, wd:           0
 7, SLL

 8, reading data: Mem[         48] =>          0
 8, reg_file[ 0] =>           0 (Port 2)
 8, PC:          48
 8, J

 9, reading data: Mem[        108] =>   38834213
 9, PC:         108
 9, NOP

10, reg_file[18] <=           2 (Write)
10, reading data: Mem[        112] =>          0
10, reg_file[18] =>           2 (Port 1)
10, reg_file[16] =>           1 (Port 2)
11, reg_file[ 8] <=          36 (Write)
10, PC:         112
10, wd:          36
10, OR

11, reg_file[ 0] =>           0 (Port 1)
11, reg_file[ 0] =>           0 (Port 2)
11, PC:         116
11, NOP

12, reading data: Mem[        120] => 2385313792
12, PC:         120
12, NOP
```

IF	ID	EX	MEM	WB	
SLL	SUB	NOP	NOP	NOP	
J	SLL	SUB	NOP	NOP	[因為 J 會造成 DataHazards]
NOP	J	SLL	SUB	NOP	[所以需要一個 NOP 進來]
OR	NOP	J	SLL	SUB	[因為 OR 會造成 DataHazards]
NOP	NOP	OR	NOP	J	[所以需要兩個 NOP 進來]



LW	NOP	NOP	OR	NOP	
ORI	LW	NOP	NOP	OR	[因為 ORI 會造成 DataHazards]
NOP	NOP	NOP	ORI	LW	[所以需要三個 NOP 進來]
SW	NOP	NOP	NOP	ORI	
DIVU	SW	NOP	NOP	NOP	


20, reg_file[0] => 0 (Port 1)
20, reg_file[0] => 0 (Port 2)
21, writing data: Mem[24] <=
20, PC: 152
20, NOP
21, PC: 156
21, NOP
22, PC: 160
22, NOP
23, PC: 164
23, NOP
24, PC: 168
24, NOP
25, PC: 172
25, NOP
26, PC: 176
26, NOP
27, PC: 180
27, NOP
28, PC: 184
28, NOP
29, PC: 188
29, NOP
30, PC: 192
30, NOP
31, PC: 196
31, NOP
32, PC: 200
32, NOP
33, PC: 204
33, NOP
34, PC: 208
34, NOP
35, PC: 212
35, NOP
36, PC: 216
36, NOP
37, PC: 220
37, NOP
38, PC: 224
38, NOP
39, PC: 228
39, NOP
40, PC: 232
40, NOP
41, PC: 236
41, NOP
42, PC: 240
42, NOP
43, PC: 244
43, NOP
44, PC: 248
44, NOP
45, PC: 252
45, NOP
46, PC: 256
46, NOP
47, PC: 260
47, NOP
48, PC: 264
48, NOP
49, PC: 268
49, NOP
50, PC: 272
50, NOP
51, PC: 276
51, NOP
52, PC: 280
52, NOP
53, reading data: Mem[284] => 34832
53, PC: 284
53, NOP



最後算出除法結果，256 除以 3 ，商為 85 餘 1 。

LW:
1: 從 instruction memory 取值
2: 解譯指令碼
3: 執行
4: 進 data memory 取值
5: 將值存回 register

BEQ:
2: 從 instruction memory 取值
3: 解譯指令碼
4: 判斷是否滿足 eq 且為 beq 指令
5: 為跳躍指令 branch 且跳躍條件成立時執行
6: X

SUB:
6: 從 instruction memory 取值
7: 解譯指令碼
8: 執行
9: X
10: 將值存回 register

SLL:
7: 從 instruction memory 取值
8: 解譯指令碼
9: 執行
10: X
11: 將值存回 register

J:
8: 從 instruction memory 取值
9: 解譯指令碼，執行無條件跳躍
10: X
11: X
12: X

OR:
10: 從 instruction memory 取值
11: 解譯指令碼
12: 執行
13: X
14: 將值存回 register

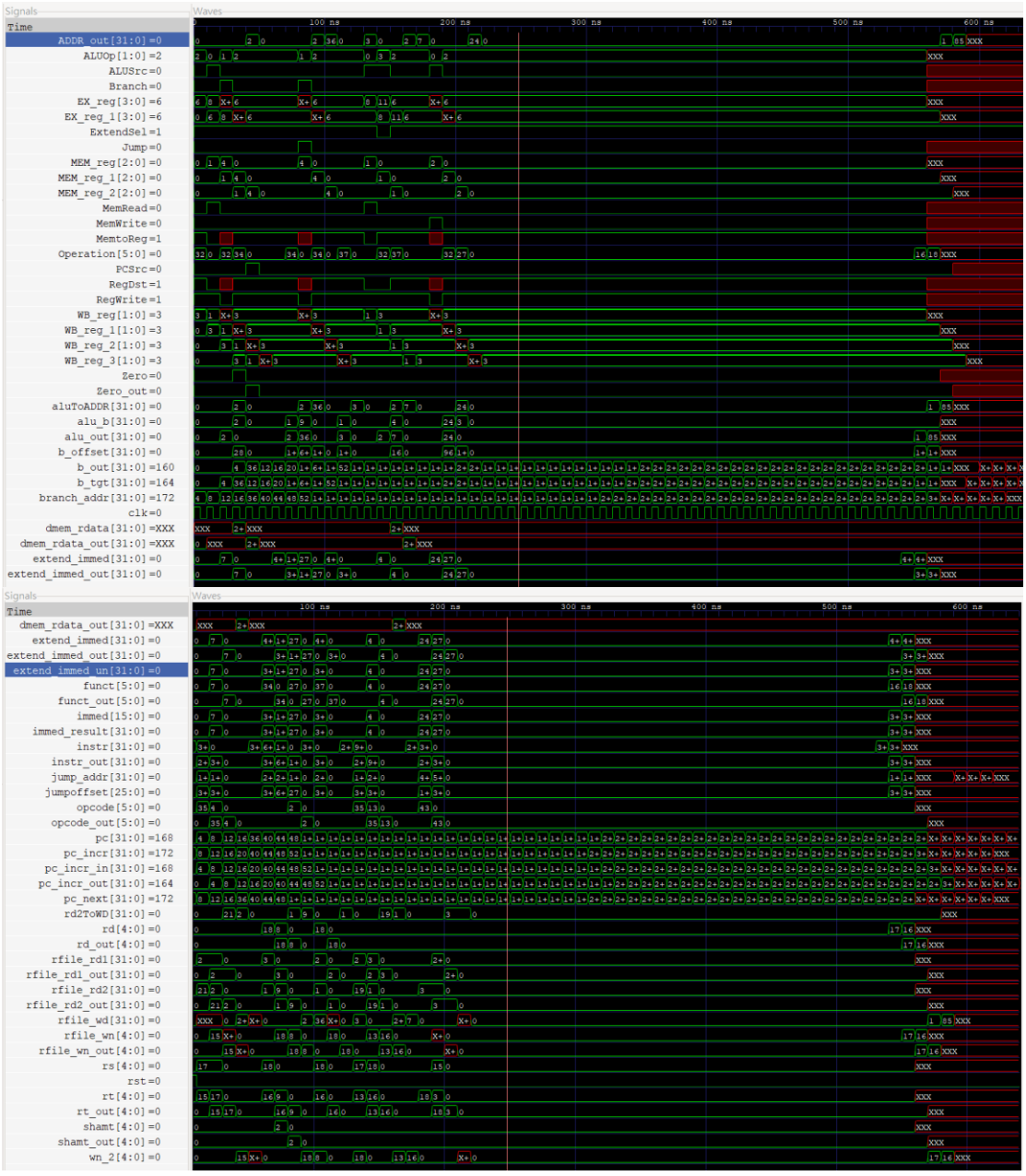
LW:
13: 從 instruction memory 取值
14: 解譯指令碼
15: 執行
16: 進 data memory 取值
17: 將值存回 register

ORI:
14: 從 instruction memory 取值
15: 解譯指令碼
16: 執行
17: X
18: 將值存回 register

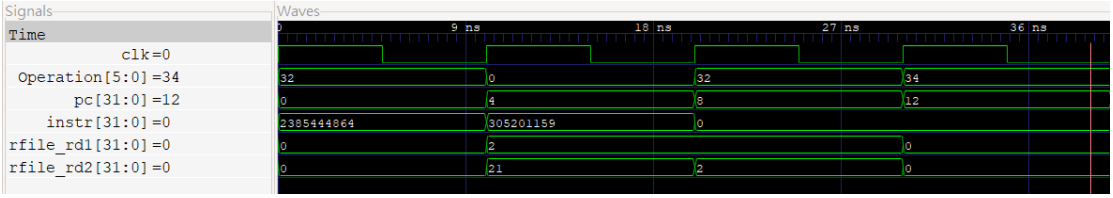
SW:
18: 從 instruction memory 取值
19: 解譯指令碼
20: 執行
21: 進 data memory 取值，將值存入 data memory
22: X

DIVU:
19: 從 instruction memory 取值
20: 解譯指令碼
經過。。。32 次之後，進行 MFHI、MFLO 指令
兩到指令在各自經過 4 次後，得到結果 商為 85 餘為 1

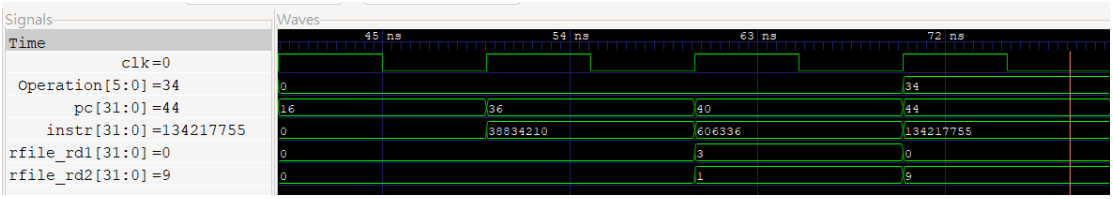
waveform 結果



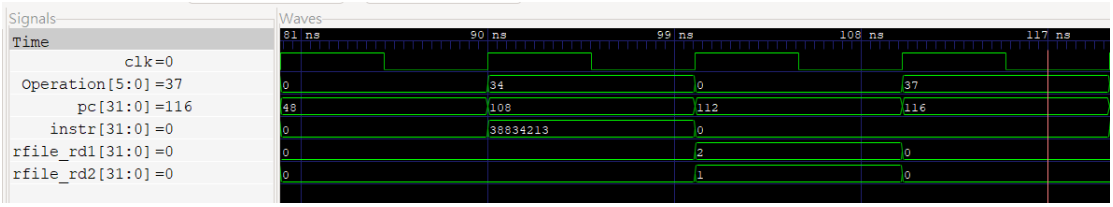
pc: 0 4 8 12



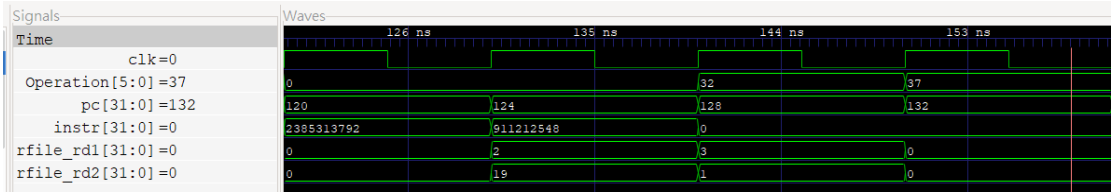
pc: 16 36 40 44



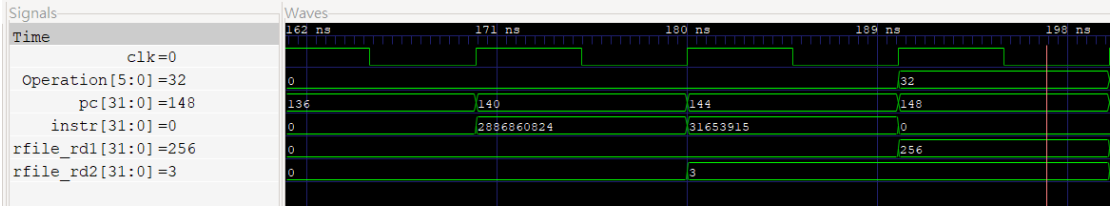
pc: 48 108 112 116



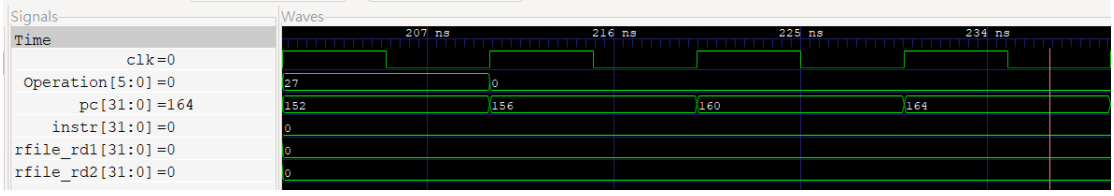
pc: 120 124 128 132



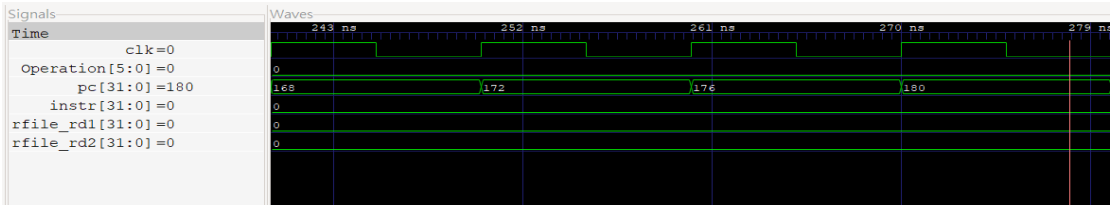
pc: 136 140 144 148



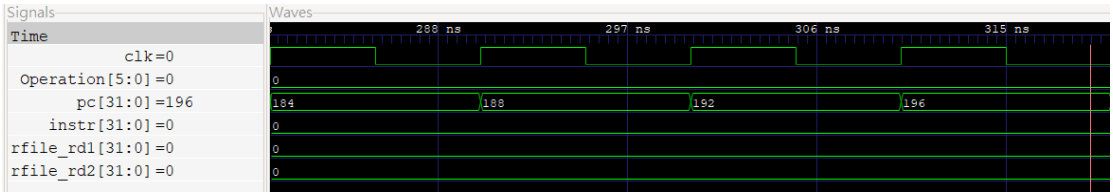
pc: 152 156 160 164



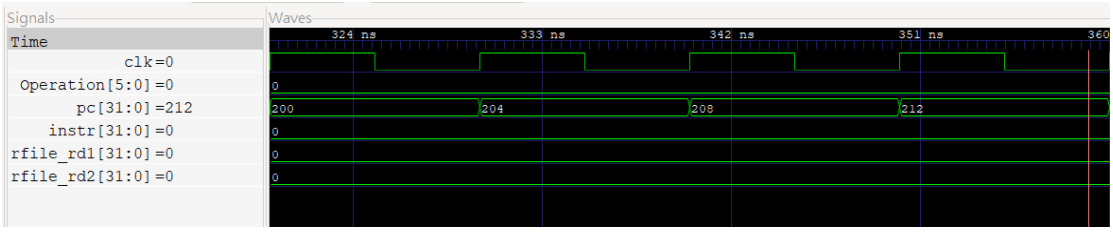
pc: 168 172 176 180



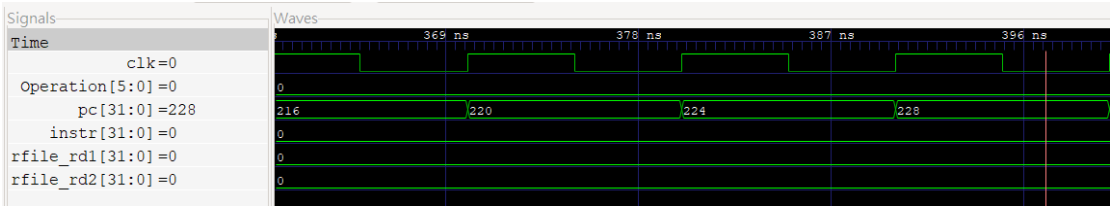
pc: 184 188 192 196



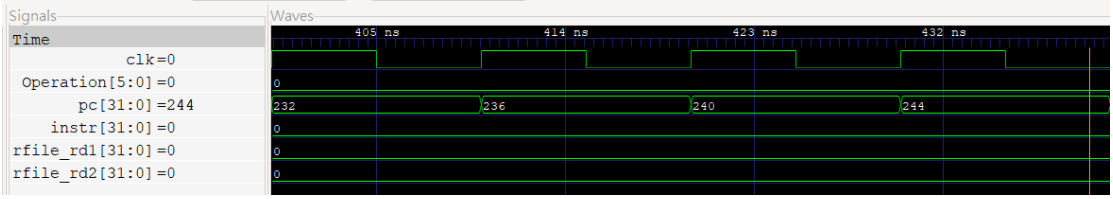
pc: 200 204 208 212



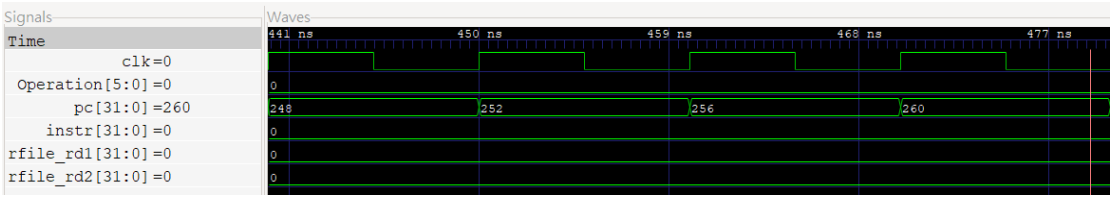
pc: 216 220 224 228



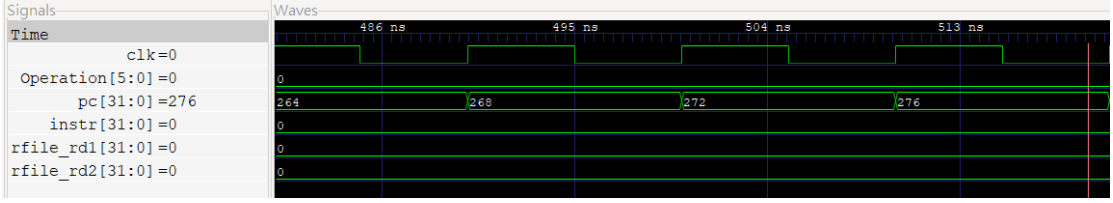
pc: 232 236 240 244



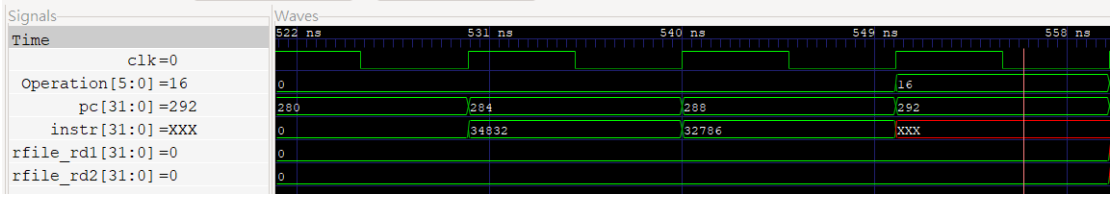
pc: 248 252 256 260



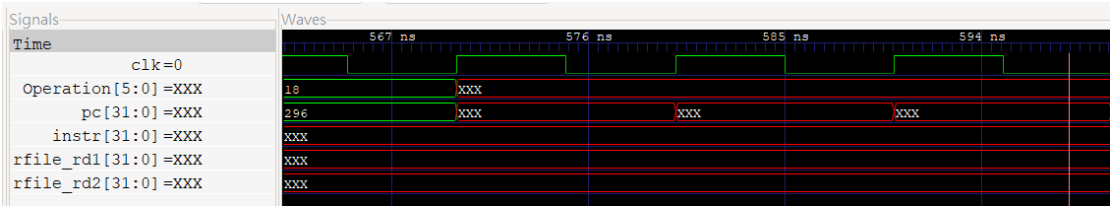
pc: 264 268 272 276



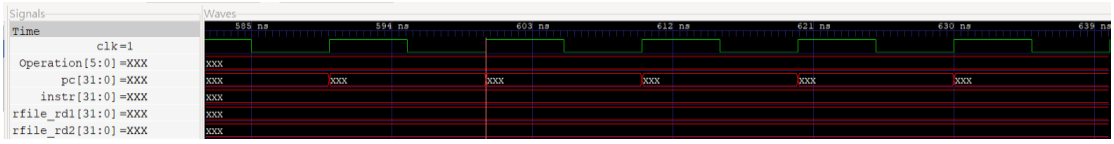
pc: 280 284 288 292



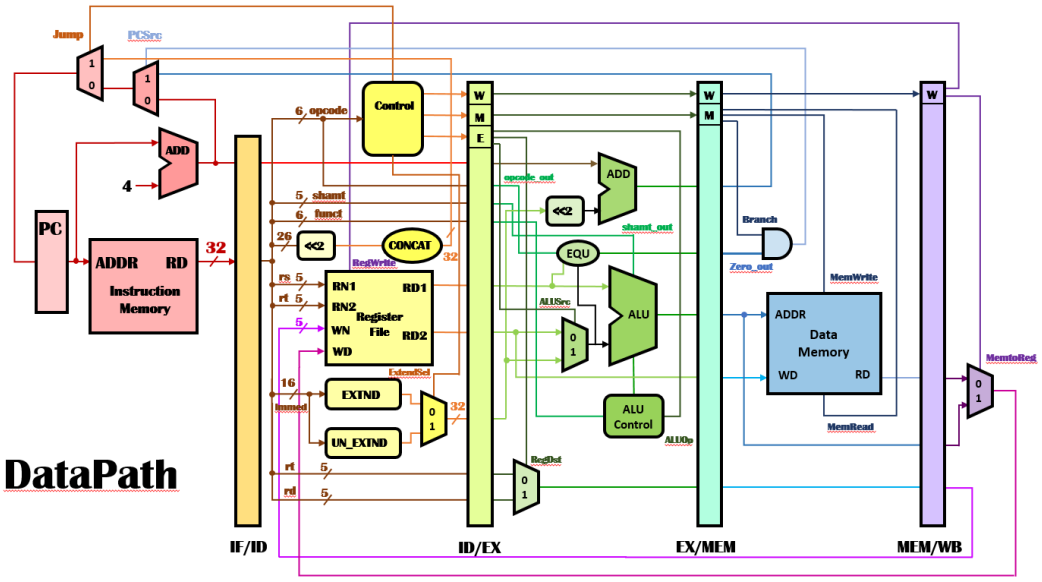
pc：296 x



pc：x



三、流程圖



四、心得感想

這次的作業有點困難，在接線及 debug 花了很久的時間，很難想像他到底是怎麼接的，就算有課本的 datapath 的圖可以參考，接線的過程還是非常坎坷。由於這是 pipeline 的 CPU，每一道指令，在每一個階段中都只會執行一小部分，所以在 debug 看哪裡線沒有接好或是有沒有送對訊號時，我們只能依靠 wavefrom，一個一個的去推敲和檢查，他每個階段是不是都是對的。

pipeline 的 CPU，在聽老師上課的時候，感覺好像沒有什麼，好像只要把訊號送過去，把線和元件照著 datapath 的圖接一接，這樣就好了。結果沒想到，實際執行時，遇到了很多困難，要考慮的東西很多，像是 CLK 以及如何保留上一層的訊號.....，還有使用 nop 來處理 DataHazards 問題。寫的過程中，其實也沒有很了解自己在寫什麼，但等到完成的那一刻，重新檢視一次 code，才完全清楚整個架構，以及明白 pipeline CPU 的執行過程。

儘管過程中，困難重重，但有耐心的一一克服後，完成了 final project，除了覺得開心，也在過程學到了很多東西。

五、分工方式

監督(複審): 姜美羚
Word 檔 : 蒲品憶
ppt 架構圖: 吳凱鈺
程式設計: 姜美羚、吳凱鈺、蒲品憶