計算機組織 Final Project

110學年度第2學期

老師：朱守禮 老師

學生：

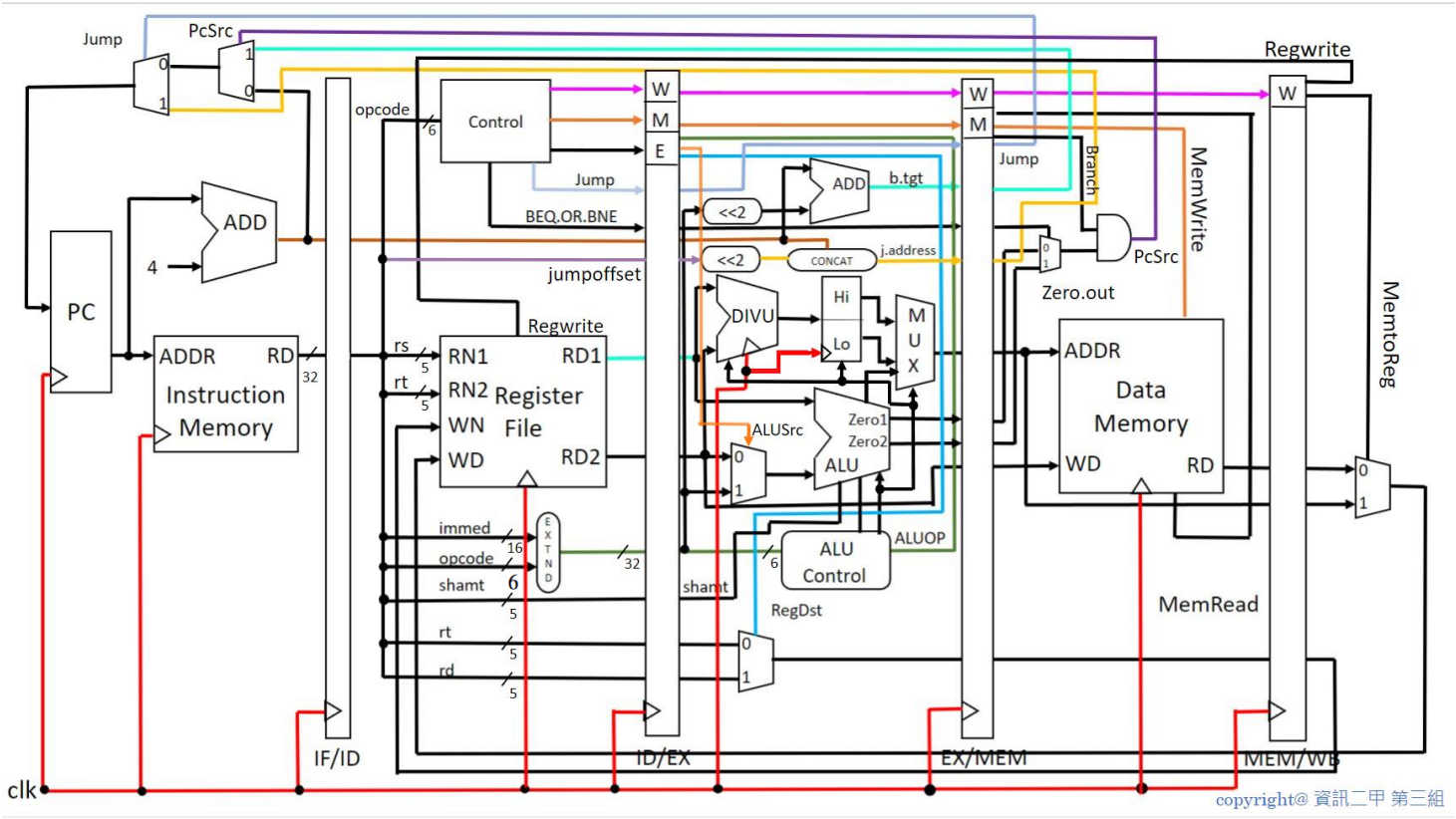
10927102 楊舒怡

10927103 楊蕙嬬

10927104 張郁琪

10927109 陳宥蓁

一、Datapath與詳細架構圖



二、設計重點說明

1.reg32：

32位元暫存器，讓PC可以暫存值。

2.add32：

執行32位元的加法運算。

用於PC及branch的位址加法。

3.memory：

用於Instruction Memory與Data Memory。

Instruction Memory只讀不寫。

Data Memory透過control_pipelined輸出的結果分別會有三種情況：

(1)可讀不可寫。

(2)可寫不可讀。

(3)不可讀也不可寫。

4.reg_file：

用RD1存rs暫存器所讀取的資料，用RD2存rt暫存器所讀取的資料。

根據RegWrite決定暫存器是否可以寫入，WN是欲寫入的暫存器編號，WD是寫入暫存器的資料。

5.extend：

傳入opcode判斷要將16位元的值進行無號數還是有號數擴充。

※ 無號數擴充 => 用16個0加上原先的16位元擴充成32位元。

※ 有號數擴充 => 取16個最高位元值加上原先的16位元擴充成32位元。

6.ALU：

包含32-bits AND、OR、ADD、SUB、SLT、SLL、ORI功能。

利用FA實現加減法運算，並將AND、OR、invert與FA結合、less，先實現1-bit ALU，再連接32個1-bit ALU實現32-bits ALU。SLL也會被包含在裡面(詳見請看Shifters說明)。

若dataA < dataB，則將slt結果(最低位元)設為1，否則設為0。

ORI是將rs暫存器的值與立即值做OR運算。

7.Shifters(SLL)：

為32-bits Barrel Shifter，達成邏輯左移運算。

利用dataB的位元的值來當作輸入2-1Mux，用來決定輸出結果。

每一層需要32個2-1多工器，該層的輸出為下層的輸入...(以此類推)

8.Division Hardware：

為32-bits 無號數除法，

採用Third Version Sequential Restoring Division Hardware 設計。

第一步驟：先將除數(dataB)暫存於除數暫存器中、被除數(dataA)暫存於rem右半部，再將rem左移 1 bit。

第二步驟：以round代表回合數。用Always Block設計，每次觸發條件(正緣向上)就將round + 1，依照round判斷是否已達32回合，(每回合要)：

先把 LHrem(rem右半部) = LHrem(rem右半部) - divr(除數暫存器) →

若rem < 0，將LHrem = LHrem + divr(剛減掉的加回來)、rem左移1位元、最低位元設為0；若rem >= 0，將rem左移1bit、最低位元設為1。

第三步驟：DONE(做了32回合)，將LHrem右移1bit。

9.HiLo暫存器：

將除法器計算完的結果儲存起來的64-bits 暫存器。

將商數存放於Lo暫存器，餘數存放於Hi暫存器。

10.mux2：

二對一多工器，傳入一個訊號用來判斷要輸出的結果。

11.mux3：

三對一多工器，傳入一個訊號判斷要輸出的是Hi or Lo or ALU運算結果。

12.ALU Control：

根據Signal來決定該完成哪一種運算。

13.control_pipelined：

根據輸入的opcode，產生對應的控制訊號。

其中BEQ_OR_BNE是我們自行設定的，用於判斷是要執行beq指令或是bne指令。

控制訊號如下表所示 ：

	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	Jump	ALUOp	BEQ_OR_BNE
R_FORMAT	1	0	1	1	0	0	0	0	10	x
ORI	0	1	0	1	0	0	0	0	11	x
LW	0	1	0	1	1	0	0	0	00	x
SW	x	1	x	0	0	1	0	0	00	x
BEQ	x	0	x	0	0	0	1	0	01	0
BNE	x	0	x	0	0	0	1	0	01	1
J	x	0	x	0	0	0	1	1	01	x

14.IF/ID ：

為 Instruction Fetch 與 Instruction Decode 之間的暫存器。當clock正緣觸發，如果rst為1，將輸出值皆設為0，反之將輸入值傳送給輸出值，藉此完成值的傳遞。

15.ID/EX ：

為 Instruction Decode 與 Execute / Address Calculation 之間的暫存器。當clock正緣觸發，如果rst為1，將輸出值皆設為0，反之將輸入值傳送給輸出值，藉此完成訊號與值的傳遞。

16.EX/MEM ：

為 Execute / Address Calculation 與 Memory Access(read/write) 之間的暫存器。當clock正緣觸發，如果rst為1，將輸出值皆設為0，反之將輸入值傳送給輸出值，藉此完成訊號與值的傳遞。

17.MEM/WB ：

為 Memory Access(read/write) 與 Write Back(results into register file) 之間的暫存器。當clock正緣觸發，如果rst為1，將輸出值皆設為0，反之將輸入值傳送給輸出值，藉此完成訊號與值的傳遞。

三、Icarus Verilog驗證結果 與 Waveform輸出圖形

※ lw $s1, $t7, 0

(IF/ID) 讀指令 => Instruction Memory
(IF/ID ~ ID/EX) 讀暫存器資料 => Register File
(ID/EX ~ EX/MEM) 計算記憶體位置 => ALU
(EX/MEM ~ MEM/WB) 讀記憶體資料 => Data Memory
(IF/ID ~ ID/EX) 寫回暫存器 => Register File

※ beq $s1, $s2, 6

(IF/ID) 讀指令 => Instruction Memory
(IF/ID ~ ID/EX) 設定條件 => Control，讀暫存器資料 => Register File
(ID/EX ~ EX/MEM) s1暫存器中的值與s2暫存器中的值利用ALU做減法運算，
如果值相同，輸出結果為1，反之則為0 => ALU
(EX/MEM ~ MEM/WB) 將輸出結果與Branch做and，and完的結果存在PCSrc中
(IF/ID)如果PCSrc為1執行跳轉(PC = (PC+4) + 24)，否則 (PC = (PC+4))。

※ sub $s2, $s0, $s2

(IF/ID) 讀指令 => Instruction Memory
(IF/ID ~ ID/EX) 讀暫存器資料 => Register File
(ID/EX ~ EX/MEM) 計算兩個暫存器($s0 & $s2)做減法的值 => ALU
(EX/MEM ~ MEM/WB) 值的傳遞(不做事)
(IF/ID ~ ID/EX) 將計算結果存回s2暫存器中 => Register File

※ sll $t0, $t1, 2

(IF/ID) 讀指令 => Instruction Memory
(IF/ID ~ ID/EX) 讀暫存器資料 => Register File
(ID/EX ~ EX/MEM) 計算t0暫存器值左移2位元的值 => ALU
(EX/MEM ~ MEM/WB) 值的傳遞(不做事)
(IF/ID ~ ID/EX) 將計算結果存回t1暫存器 => Register File

※ j 27

(IF/ID) 讀指令 => Instruction Memory。
(IF/ID ~ ID/EX) 設定條件 => Control，讀暫存器資料 => Register File。
(ID/EX ~ EX/MEM) 計算所要跳躍的位置 。
(EX/MEM ~ MEM/WB) 傳遞剛算好的要跳躍的位置。
(IF/ID) PC = 剛算好的要跳躍的位置(無條件跳躍)。

※ or $s2, $s0, $s2

(IF/ID) 讀指令 => Instruction Memory
(IF/ID ~ ID/EX) 讀暫存器資料 => Register File
(ID/EX ~ EX/MEM) 將兩個暫存器($s0 & $s2)做or運算 => ALU
(EX/MEM ~ MEM/WB) 值的傳遞(不做事)
(IF/ID ~ ID/EX) 將計算結果存回s2暫存器 => Register File

※ lw $s1, $t5, 0

(IF/ID) 讀指令 => Instruction Memory
(IF/ID ~ ID/EX) 讀暫存器資料 => Register File
(ID/EX ~ EX/MEM) 計算記憶體位置 => ALU
(EX/MEM ~ MEM/WB) 讀記憶體資料 => Data Memory
(IF/ID ~ ID/EX) 寫回暫存器 => Register File

※ ori $s2, $s0, 4

(IF/ID) 讀指令 => Instruction Memory
(IF/ID ~ ID/EX) 讀暫存器資料 => Register File，將立即值做無號數擴充
(ID/EX ~ EX/MEM) 將s2暫存器值與立即值做or運算 => ALU
(EX/MEM ~ MEM/WB) 值的傳遞(不做事)
(IF/ID ~ ID/EX) 將計算結果存回s0暫存器 => Register File

※ sw $zero, $s2, 24

(IF/ID) 讀指令 => Instruction Memory
(IF/ID ~ ID/EX) 讀暫存器資料 => Register File
(ID/EX ~ EX/MEM) 計算記憶體位置 => ALU
(EX/MEM ~ MEM/WB) 將資料存入該記憶體位置 => Data Memory
(IF/ID ~ ID/EX) 不做事

※ divu $t7, $v1

(IF/ID) 讀指令 => Instruction Memory
(IF/ID ~ ID/EX) 讀暫存器資料 => Register File
(ID/EX ~ EX/MEM) 將暫存器做除法運算 => DIVU
※ 需要32個cycle，利用nop解決hazard問題，
32個cycle後除法運算後的值會寫入hi、lo暫存器中
(EX/MEM ~ MEM/WB) 不做事
(IF/ID ~ ID/EX) 不做事

```
 0, reading data: Mem[         x] =>          x
 0, reading data: Mem[         0] => 2385444864
 0, reg_file[ 0] =>          0 (Port 1)
 0, reg_file[ 0] =>          0 (Port 2)
 0, PC:         0
 0, wd:          0
 0, NOP

 1, reading data: Mem[         4] =>  305201158
 1, reg_file[17] =>          2 (Port 1)
 1, reg_file[15] =>         21 (Port 2)
 1, PC:         4
 1, LW

 2, reading data: Mem[         8] =>          0
 2, reg_file[17] =>          2 (Port 2)
 2, PC:         8
 2, BEQ

 3, reading data: Mem[         2] =>        256
 3, reg_file[ 0] =>          0 (Port 1)
 3, reg_file[ 0] =>          0 (Port 2)
 3, PC:        12
 3, wd:          0
 3, NOP

 5, reg_file[15] <=        256 (Write)
 4, PC:        16
 4, wd:        256
 4, NOP

 5, PC:        32
 5, wd:         x
 5, NOP

 6, reading data: Mem[        36] =>   38834210
 6, PC:        36
 6, wd:          0
 6, NOP

 7, reading data: Mem[        40] =>     606336
 7, reg_file[18] =>          3 (Port 1)
 7, reg_file[16] =>          1 (Port 2)
 7, PC:        40
 7, wd:          0
 7, SUB

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

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

11, reg_file[18] <=          2 (Write)
10, PC:        52
10, wd:          2
10, NOP
```

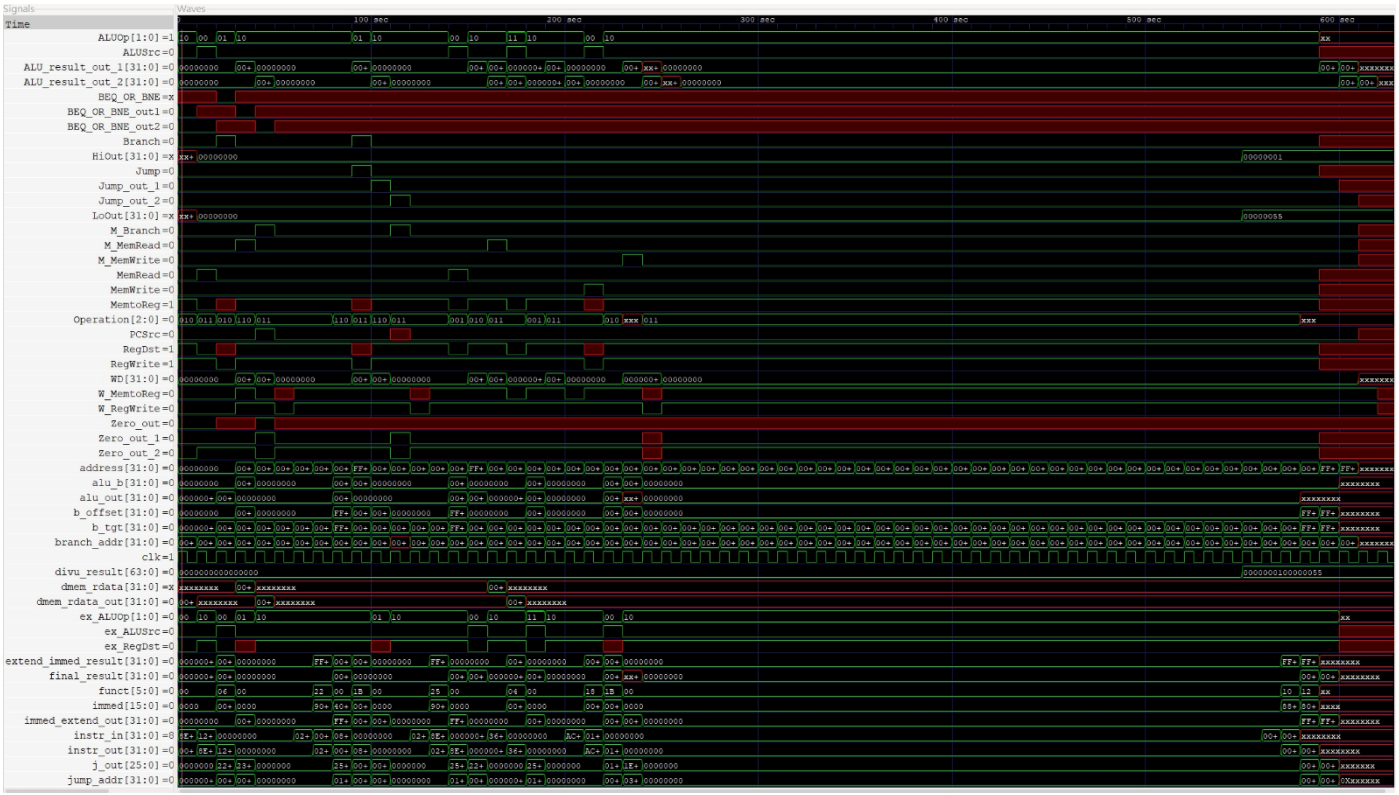
```
11, PC:        56
11, wd:          0
11, NOP

12, reg_file[ 8] <=          0 (Write)
12, reading data: Mem[       108] =>   38834213
12, PC:       108
12, wd:          x
12, NOP

13, reading data: Mem[       112] => 2385313792
13, reg_file[18] =>          2 (Port 1)
13, reg_file[16] =>          1 (Port 2)
13, PC:       112
13, wd:          0
13, OR

14, reading data: Mem[       116] =>          0
14, reg_file[17] =>          2 (Port 1)
14, reg_file[13] =>         19 (Port 2)
14, PC:       116
14, LW

15, reg_file[ 0] =>          0 (Port 1)
15, reg_file[ 0] =>          0 (Port 2)
15, PC:       120
15, wd:          0
15, NOP

16, reading data: Mem[         2] =>        256
16, reading data: Mem[       124] =>  911212548
17, reg_file[18] <=          3 (Write)
16, PC:       124
16, wd:          3
16, NOP

17, reading data: Mem[       128] =>          0
17, reg_file[18] =>          3 (Port 1)
17, reg_file[16] =>          1 (Port 2)
17, PC:       128
17, ORI

18, reg_file[13] <=        256 (Write)
18, reg_file[ 0] =>          0 (Port 1)
18, reg_file[ 0] =>          0 (Port 2)
18, PC:       132
18, wd:          0
18, NOP

19, PC:       136
19, wd:          0
19, NOP

20, reading data: Mem[       140] => 2886860824
21, reg_file[16] <=          x (Write)
20, PC:       140
20, wd:          x
20, NOP

21, reading data: Mem[       144] =>   31653915
21, reg_file[18] =>          3 (Port 2)
21, PC:       144
21, SW
```

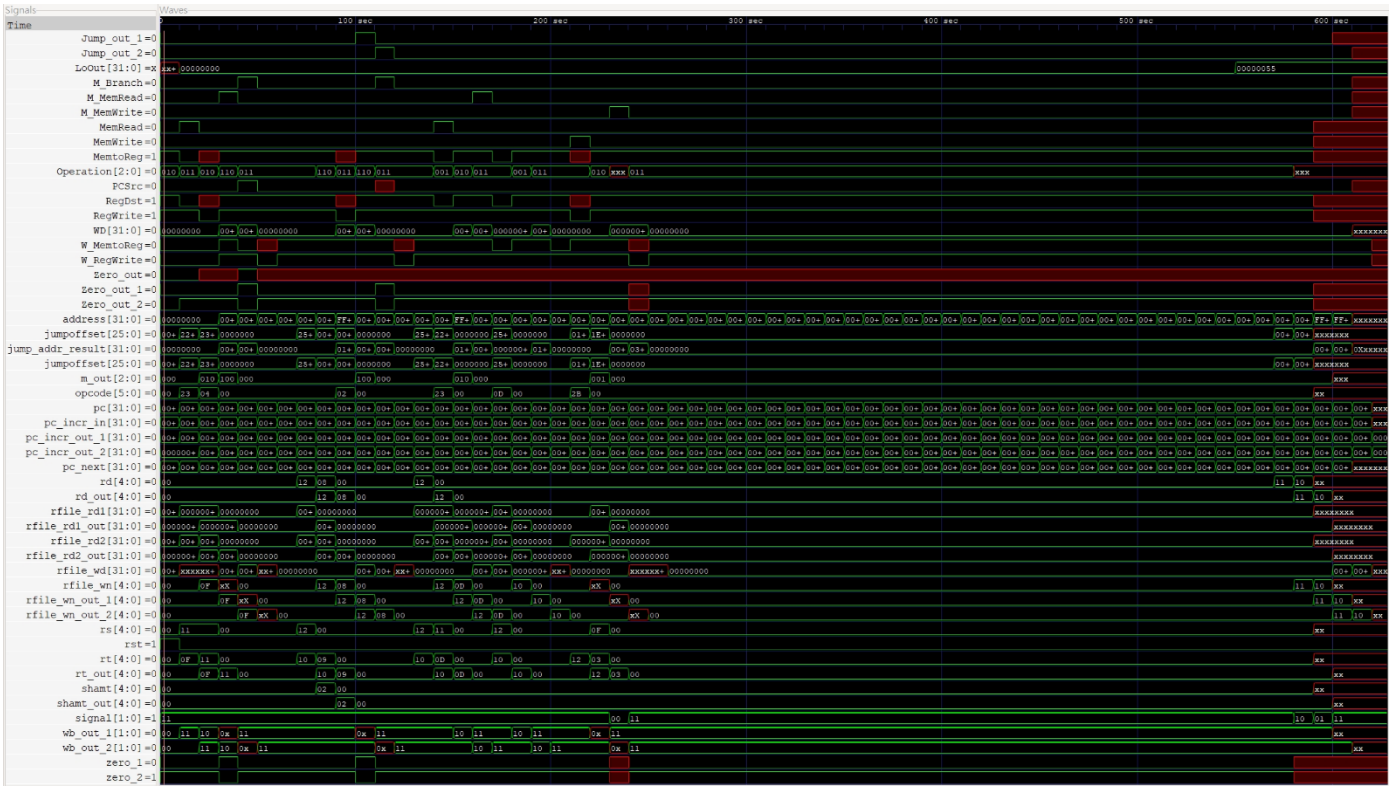
```
22, reading data: Mem[       148] =>          0
22, reg_file[15] =>        256 (Port 1)
22, reg_file[ 3] =>          3 (Port 2)
22, PC:       148
22, wd:         0
22, DIVU

23, reg_file[ 0] =>          0 (Port 1)
23, reg_file[ 0] =>          0 (Port 2)
23, PC:       152
23, wd:         0
23, NOP

24, writing data: Mem[        24] <=          3
24, PC:       156
24, wd:         x
24, NOP

25, PC:       160
25, wd:         x
25, NOP

26, PC:       164
26, wd:         0
26, NOP

27, PC:       168
27, wd:         0
27, NOP

28, PC:       172
28, wd:         0
28, NOP

29, PC:       176
29, wd:         0
29, NOP

30, PC:       180
30, wd:         0
30, NOP

31, PC:       184
31, wd:         0
31, NOP

32, PC:       188
32, wd:         0
32, NOP

33, PC:       192
33, wd:         0
33, NOP

34, PC:       196
34, wd:         0
34, NOP

35, PC:       200
35, wd:         0
35, NOP

36, PC:       204
36, wd:         0
36, NOP
```

```
37, PC:        208
37, wd:          0
37, NOP

38, PC:        212
38, wd:          0
38, NOP

39, PC:        216
39, wd:          0
39, NOP

40, PC:        220
40, wd:          0
40, NOP

41, PC:        224
41, wd:          0
41, NOP

42, PC:        228
42, wd:          0
42, NOP

43, PC:        232
43, wd:          0
43, NOP

44, PC:        236
44, wd:          0
44, NOP

45, PC:        240
45, wd:          0
45, NOP

46, PC:        244
46, wd:          0
46, NOP

47, PC:        248
47, wd:          0
47, NOP

48, PC:        252
48, wd:          0
48, NOP

49, PC:        256
49, wd:          0
49, NOP

50, PC:        260
50, wd:          0
50, NOP

51, PC:        264
51, wd:          0
51, NOP

52, PC:        268
52, wd:          0
52, NOP

53, PC:        272
53, wd:          0
53, NOP
```

```
                53, NOP

                54, PC:       276
                54, wd:         0
                54, NOP

                55, PC:       280
                55, wd:         0
                55, NOP

                56, reading data: Mem[       284] =>      34832
                56, PC:       284
                56, wd:         0
                56, NOP

                57, reading data: Mem[       288] =>      32786
                57, PC:       288
                57, wd:         0
                57, MFHI

                58, reading data: Mem[       292] =>          x
                58, PC:       292
                58, wd:         0
                58, MFLO

control_single unimplemented opcode  x
                59, reg_file[ x] =>          x (Port 1)
                59, reg_file[ x] =>          x (Port 2)
                59, PC:       296
                61, reg_file[17] <=          1 (Write)
                60, PC:       300
                61, PC:       304
                62, reg_file[16] <=         85 (Write)
                62, End of Simulation

tb.v:41: $finish called at 630 (1s)
```

- waveform圖


Signals
Waves
Time
ALUOp[1:0]
ALUSrc
ALU_result_out_1[31:0]
ALU_result_out_2[31:0]
BEQ_OR_BNE
BEQ_OR_BNE_out1
BEQ_OR_BNE_out2
Branch
HiOut[31:0]
Jump
Jump_out_1
Jump_out_2
LoOut[31:0]
M_Branch
M_MemRead
M_MemWrite
MemRead
MemWrite
MemtoReg
Operation[2:0]
PCSrc
RegDst
RegWrite
WD[31:0]
W_MemtoReg
W_RegWrite
Zero_out
Zero_out_1
Zero_out_2
address[31:0]
alu_b[31:0]
alu_out[31:0]
b_offset[31:0]
b_tgt[31:0]
branch_addr[31:0]
clk
divu_result[63:0]
dmem_rdata[31:0]
dmem_rdata_out[31:0]
ex_ALUOp[1:0]
ex_ALUSrc
ex_RegDst
extend_immed_result[31:0]
final_result[31:0]
funct[5:0]
immed[15:0]
immed_extend_out[31:0]
instr_in[31:0]
instr_out[31:0]
j_out[25:0]
jump_addr[31:0]


Signals
Waves
Time
Jump_out_1
Jump_out_2
LoOut[31:0]
M_Branch
M_MemRead
M_MemWrite
MemRead
MemWrite
MemtoReg
Operation[2:0]
PCSrc
RegDst
RegWrite
WD[31:0]
W_MemtoReg
W_RegWrite
Zero_out
Zero_out_1
Zero_out_2
address[31:0]
jumpoffset[25:0]
jump_addr_result[31:0]
jumpoffset[25:0]
m_out[2:0]
opcode[5:0]
pc[31:0]
pc_incr_in[31:0]
pc_incr_out_1[31:0]
pc_incr_out_2[31:0]
pc_next[31:0]
rd[4:0]
rd_out[4:0]
rfile_rd1[31:0]
rfile_rd1_out[31:0]
rfile_rd2[31:0]
rfile_rd2_out[31:0]
rfile_wd[31:0]
rfile_wn[4:0]
rfile_wn_out_1[4:0]
rfile_wn_out_2[4:0]
rs[4:0]
rst
rt[4:0]
rt_out[4:0]
shamt[4:0]
shamt_out[4:0]
signal[1:0]
wb_out_1[1:0]
wb_out_2[1:0]
zero_1
zero_2

四、心得感想

楊舒怡：這次的期末project比期中要複雜很多，除了要了解pipeline的概念，還要了解整個架構圖的運作方式，五個階段要做什麼事等等，小幅修改架構圖後，再利用程式實作出來。前前後後都蠻困惑的，實在是有組員的幫助才能順利完成。

楊蕙嬬：本次project比期中複雜太多了，除了ALU和DIVU的外，還增加了記憶體和暫存器之間的訊號傳遞、control 對 WB、M、EX 的作用等等，除此之外，有些指令是要記憶體讀取或寫入有些結果是寫入暫存器等等，特別是 jump 和 branch 指令，因為跳躍的方式不同所以當 PC 跳躍後，要花一些時間去檢查跳的位置是否正確，但經過本次次作業對個指令的執行路徑有更多的理解。

張郁琪：這次的跟上次比起來真的困難很多，我們剛打完要去測試時一直遇到一個很困惑的問題，然後我們不管怎麼想都覺得我們的也很合理，之後才發現要把等號改成箭等，這也讓我發現到原來這兩個雖然都是賦值但其實也是有差異的；我們也很常遇到一堆跟delay有關的問題而每次都是最後才發現，所以也每天都搞到很晚才完成。雖然很累但是也透過這次作業得到了很多收穫。

陳宥蓁：這次的final project 跟之前的midterm project 比起來確實難了很多，除了那四條管子要處理，還有hazard的問題也必須要處理。我們打程式的過程其實沒有那麼順利，有時候卡一個bug就卡了一整個晚上，還好最後都有順利找到，但是每次找到之後都會覺得為什麼這裡會打錯…（覺得不會錯的地方，但都是那裡發生問題），找到之後雖然很爽但也同時覺得自己很蠢！！！

五、各組員分工方式與負責項目

共同完成。