計算機組織 Final Project

110學年度第2學期

老師：朱守禮 老師

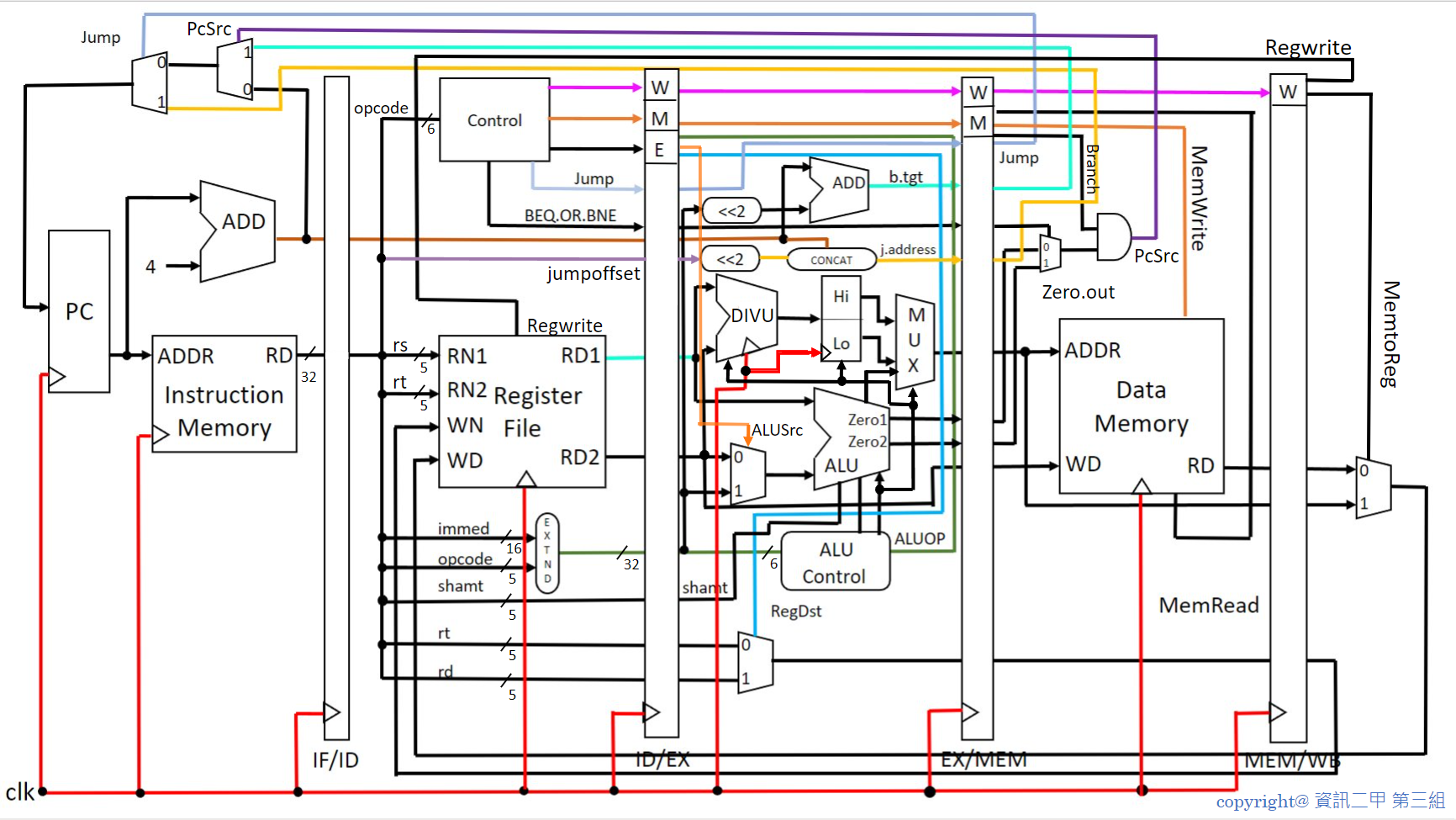
學生：

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

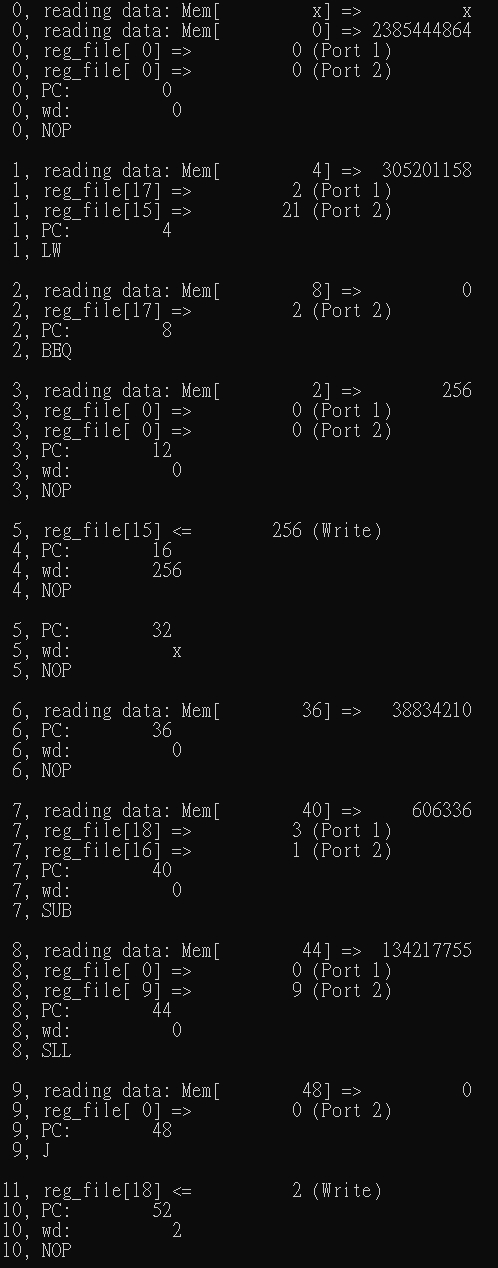
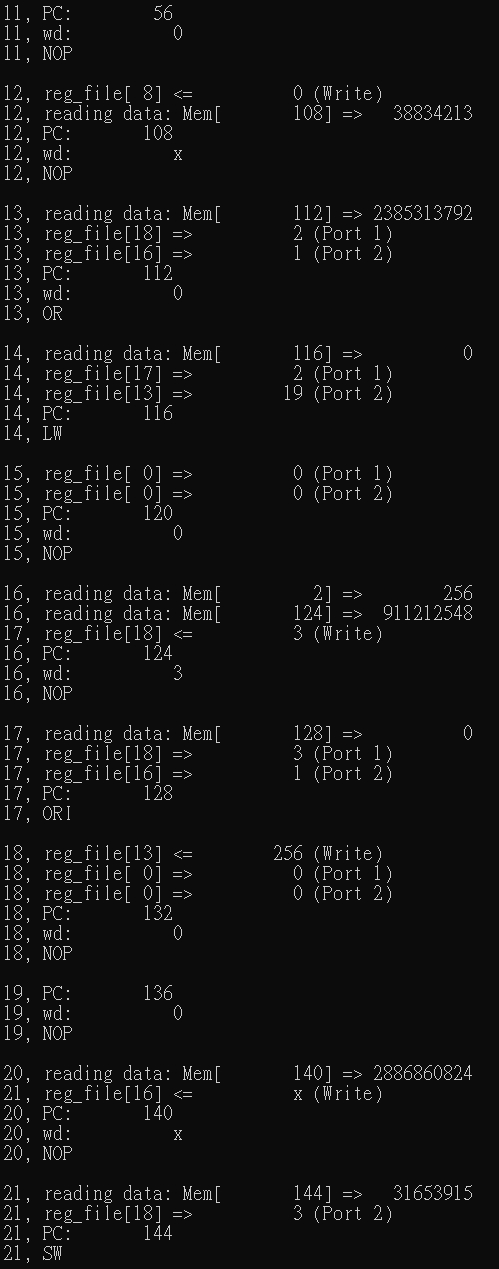
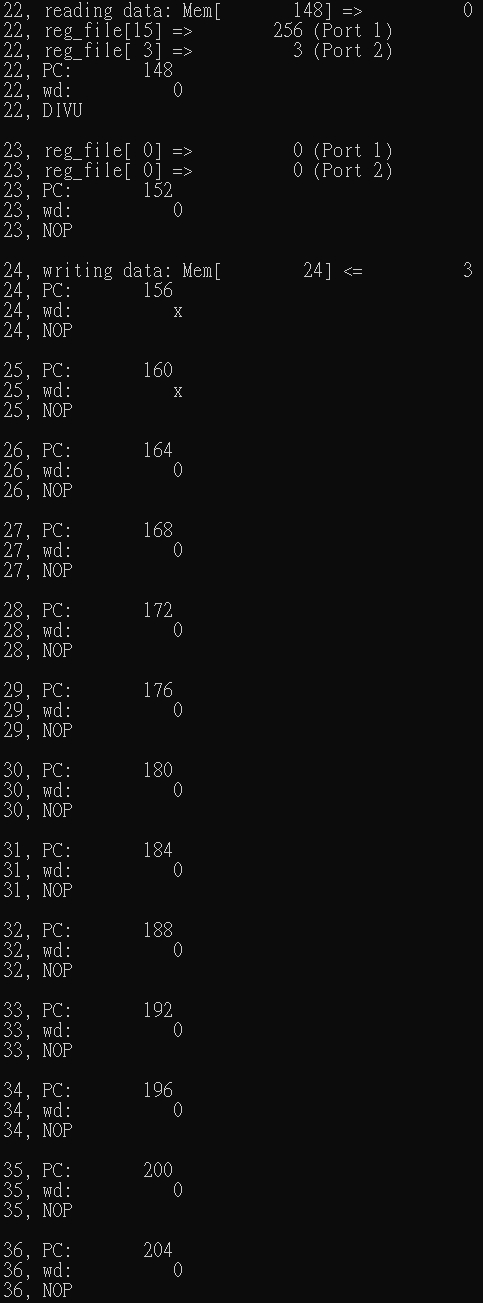
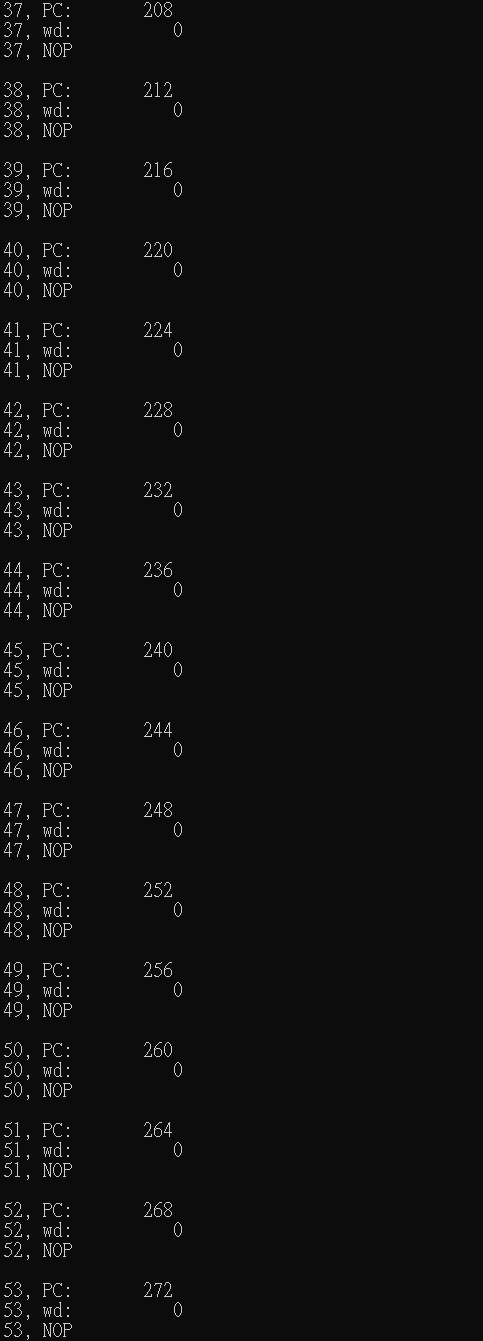
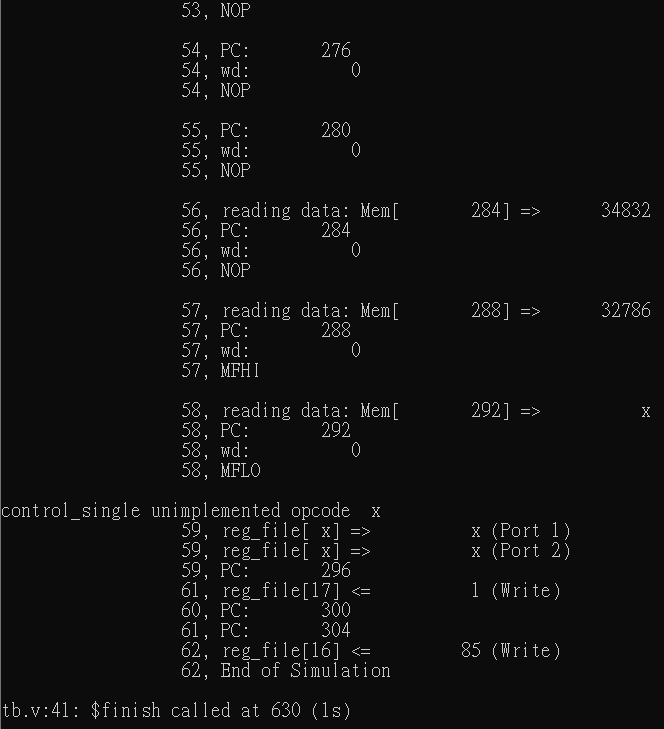
(IF/ID) 讀指令 => Instruction Memory  
(IF/ID ~ ID/EX) 讀暫存器資料 => Register File，將立即值做無號數擴充  
(ID/EX ~ EX/MEM) 將s2暫存器值與立即值做or運算 => ALU  
(EX/MEM ~ MEM/WB) 值的傳遞(不做事)   
(IF/ID ~ ID/EX) 將計算結果存回s0暫存器 => Register File

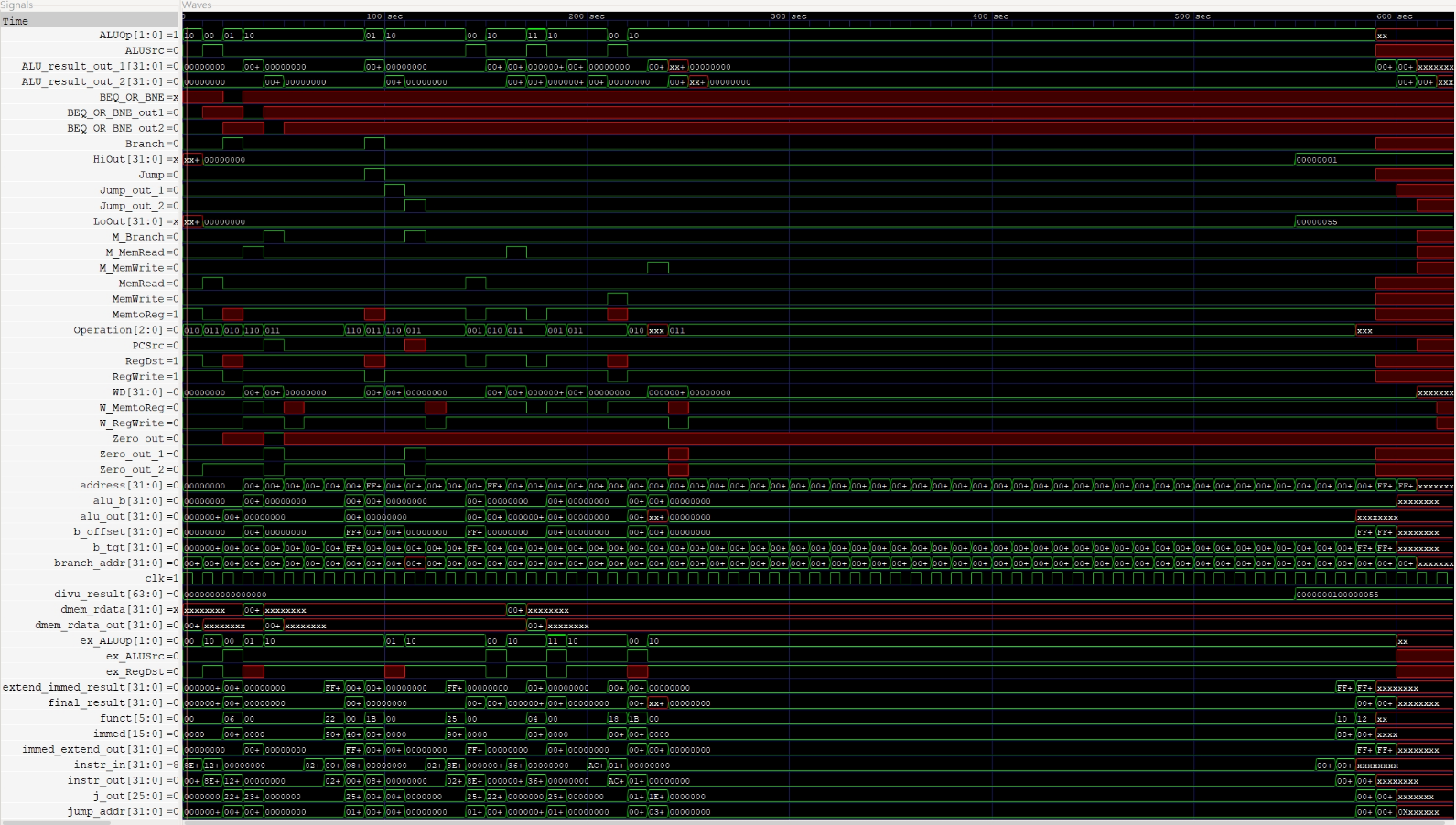
※ sw $zero, $s2, 24

(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

*  **waveform圖**

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。