

# 計算機組織 Lab4

9617145 許晏峻

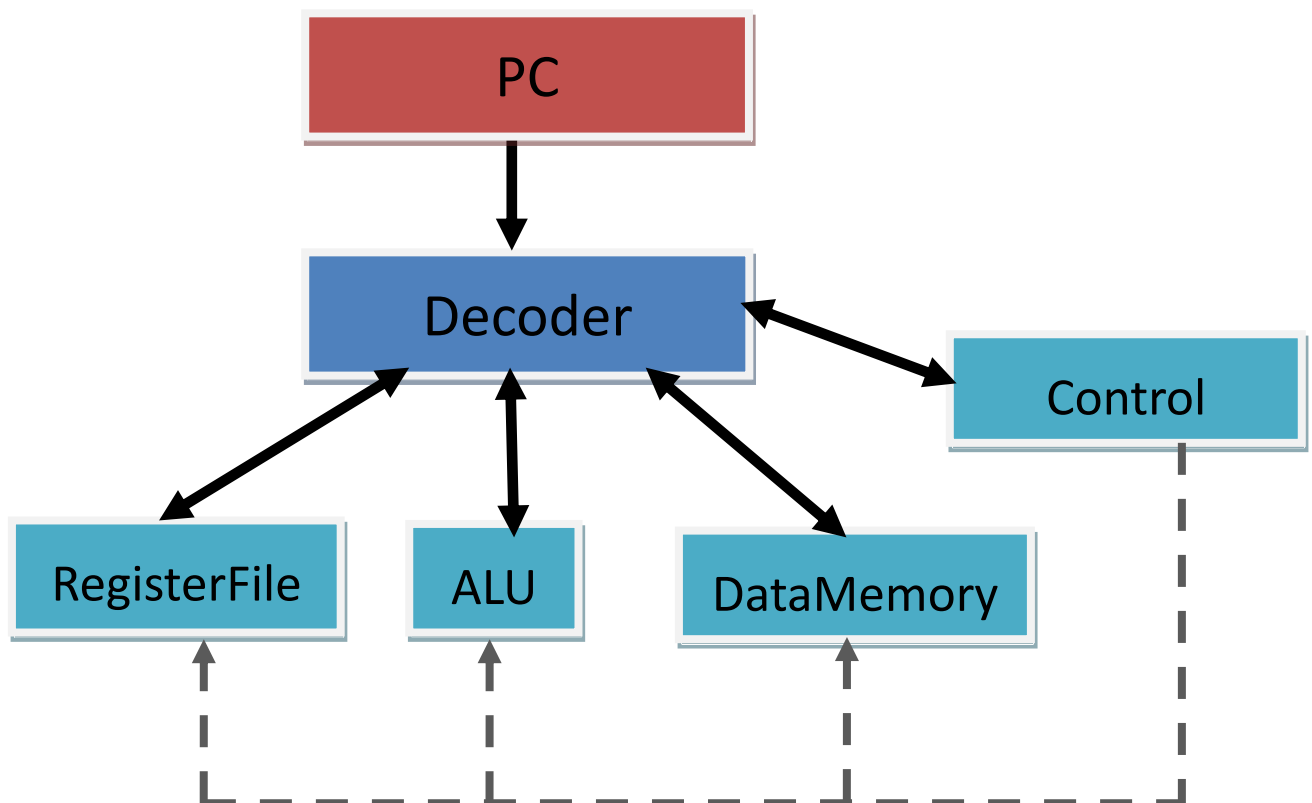
## 各元件功能：

每個 block 都代表一個獨立的 module：

1. PC 開始整個指令，依照 PC 值取出 InstructionMemory 傳進 Decoder
2. Control 負責的是所有的 control signal for MUX
3. RegisterFile 做 register 的存值和讀值，存值是在 positive edge 瞬間進行
4. DataMemory 做 memory 的存值和讀值，存值是在 positive edge 瞬間進行
5. ALU 做各種運算包括 add、sub、shift left、shift right
6. Decoder 做各種傳值的動作，依照 control signal 的值決定放進 RegisterFile、ALU、DataMemory 中的參數是 Instruction 的哪些部份

因此在 Verilog 中的架構如下圖：

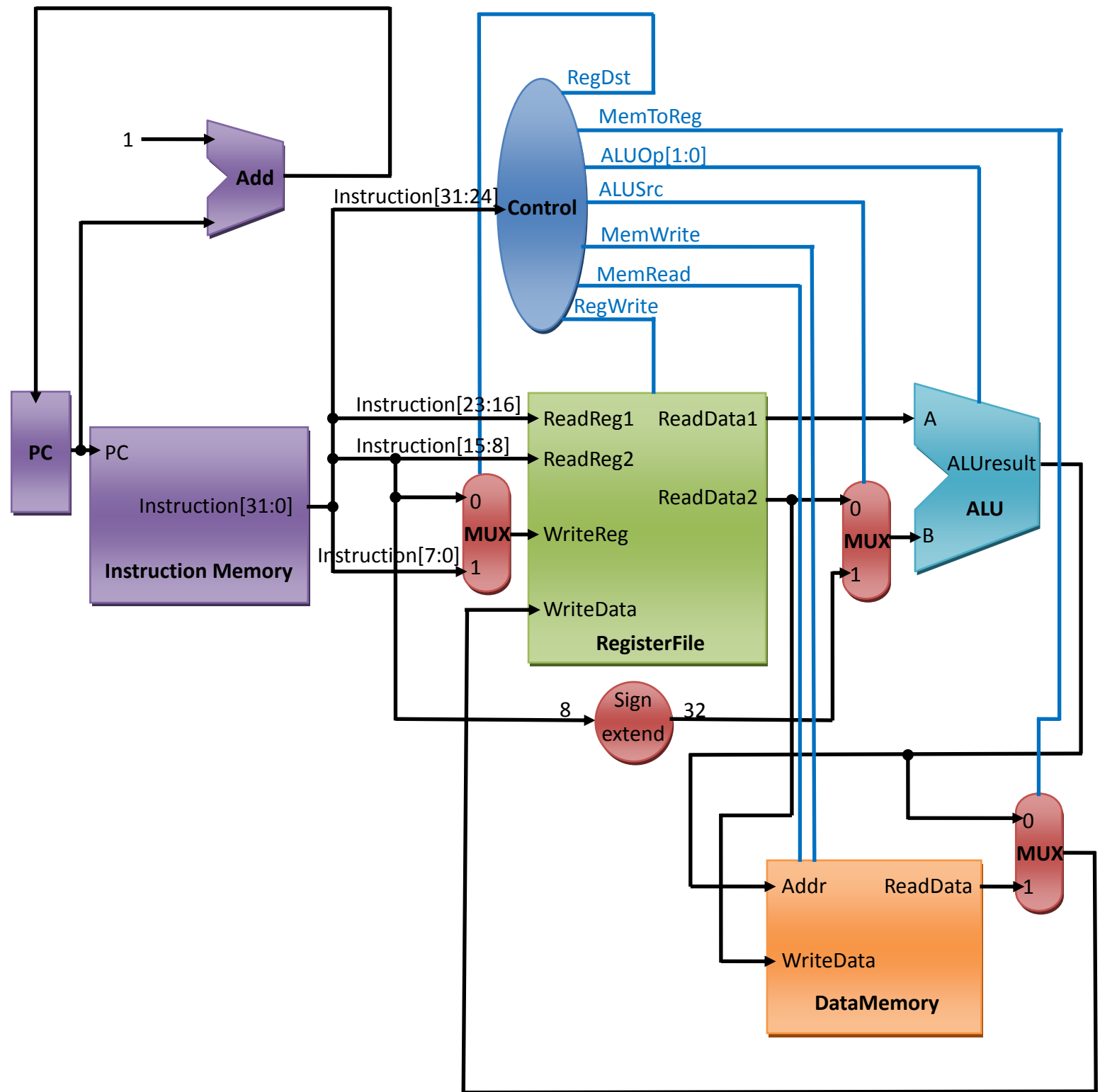
每個 module 都是 link 到 Decoder，Decoder 依照 Control 的 signal 決定該如何傳值



## 架構圖：

在整個 single-cycle 的概念上架構如下圖：
































我的設計是在 positive edge 來作 write 的動作，因此 write 會在下個 clock 來的瞬間做好  
以下將每個 module 做的事用顏色區隔開來



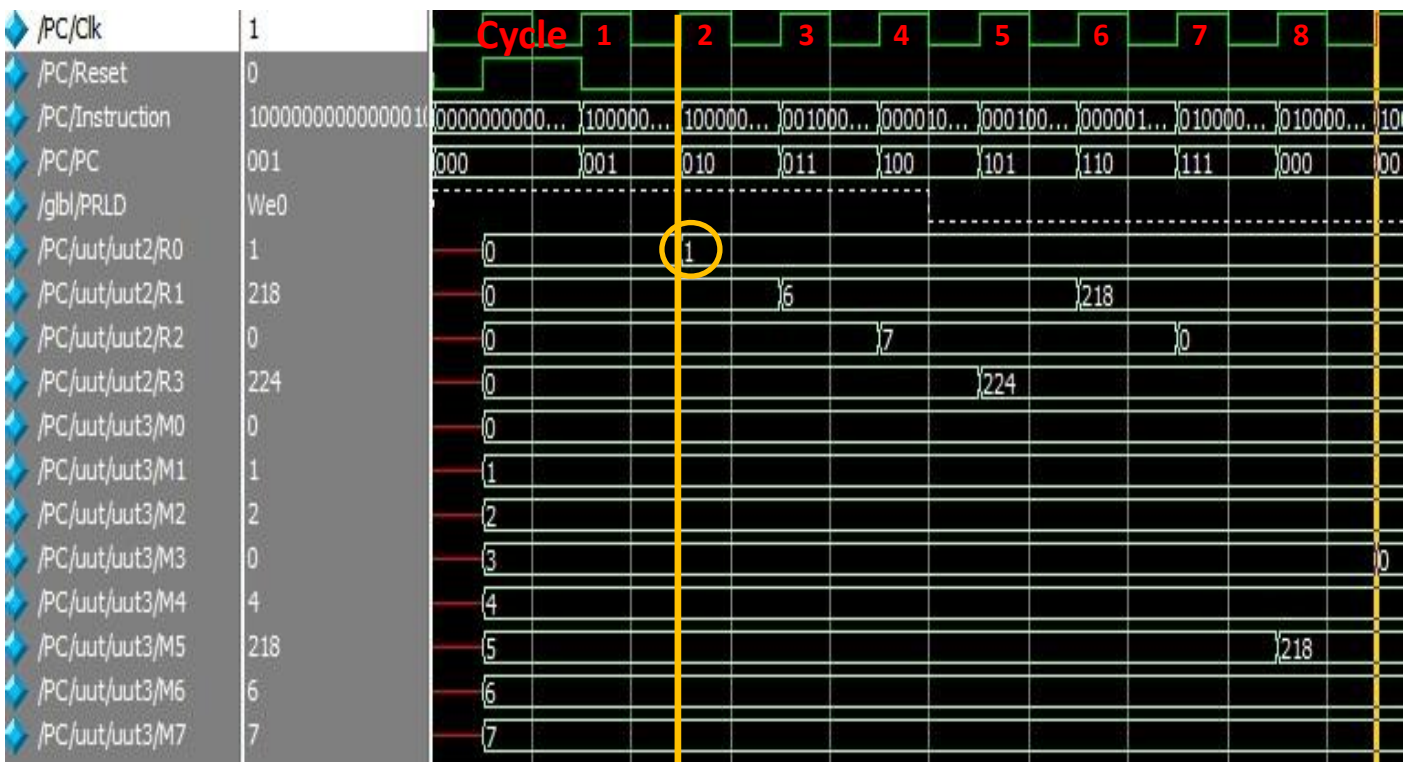
波型圖：

/PC/Clk	1
/PC/Reset	0
/PC/Instruction	10000
/PC/PC	001
/glbl/PRLD	0e0
/PC/uut/uut2/R0	1
/PC/uut/uut2/R1	218
/PC/uut/uut2/R2	0
/PC/uut/uut2/R3	224
/PC/uut/uut3/M0	0
/PC/uut/uut3/M1	1
/PC/uut/uut3/M2	2
/PC/uut/uut3/M3	0
/PC/uut/uut3/M4	4
/PC/uut/uut3/M5	218
/PC/uut/uut3/M6	6
/PC/uut/uut3/M7	7

## Final Vlaue

	/PC/Clk	1
	/PC/Reset	1
	 /PC/Instruction	00000
	 /PC/PC	000
	 /glbl/PRLD	0x1
	 /PC/uut/uut2/R0	0
	 /PC/uut/uut2/R1	0
	 /PC/uut/uut2/R2	0
	 /PC/uut/uut2/R3	0
	 /PC/uut/uut3/M0	0
	 /PC/uut/uut3/M1	1
	 /PC/uut/uut3/M2	2
	 /PC/uut/uut3/M3	3
	 /PC/uut/uut3/M4	4
	 /PC/uut/uut3/M5	5
	 /PC/uut/uut3/M6	6
	 /PC/uut/uut3/M7	7

## Initial Vlaue



Cycle1 的值在進入  
Cycle2 的瞬間  
(Positive edge)Write

## Cycle 分析：

Cycle	1	2	3	4	5	6	7	8
Signal	PC+1	PC+1	PC+1	PC+1	PC+1	PC+1	PC+1	PC+1
/Description	Read M1	Read M6	R0+ R1	SL R2 5bit	R3-R1	SR R0 2bit	Read R1	Read R2
	Write R0	Write R1	Write R2	Write R3	Write R1	Write R2	Write M5	Write M3
RegDst	0	0	1	1	1	1	X	X
MemtoReg	1	1	0	0	0	0	X	X
MemRead	1	1	0	0	0	0	0	0
MemWrite	0	0	0	0	0	0	1	1
ALUOp	X	X	00	10	01	11	X	X
ALUSrc	X	X	0	1	0	1	X	X
RegWrite	1	1	1	1	1	1	0	0

## 心得：

這次作業寫起來滿有感覺的，不像之前 lab1~lab3 感覺都有些模糊，只做部份的東西，卻不知道細節應該怎麼做設定才對。

Lab4 就像上課講的 single-cycle processor 差不多：先將 PC 設定好 address，然後去讀 Instruction Memory，接著就是利用 Decoder 來將指令解碼，設定好 Control Signal，然後開始讀取 Register 後再丟進 ALU 來作需要的運算，或著就去存取 Data Memory 的值做寫入或讀取，最後再看是否要寫回 Register；大致上就是在做這些事。

和上課時不一樣的地方：作業用的指令格式和 MIPS 的指令格式有所不同，設計得較為簡單，都是以 8-bit 為單位，所以沒有 function field 的部份，存取 DataMemory 時也不用多去計算 Memory 的位置，而是直接看 Memory 的編號，而且實作的指令也比較少，沒有 Branch、Jump 等，所以真的是簡單的許多。

經過一番思考，將上課所學的概念套進去，尤其像是 Control Signal，這個算是最重要的地方，這個部份在 lab3 已經有將 table 填過，概念有了，lab4 就不會太難將整個電路組合起來，就是在多加了個 Instruction Memory 和 PC 就算是完成了。