LAB 4 - Pipeline CPU I

1. 系統架構:

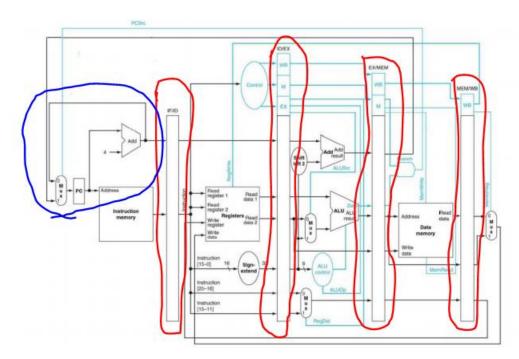
加入 pipeline 的想法,讓 CPU 更有效率。

2. 設計模組分析、設計結果:

藍色圓圈的地方,讓 PC 直接加 4,不用等 instruction 全部跑完,才能跑下一個。而是把 CPU 分成 5 個 state:

$$IF \rightarrow ID \rightarrow EX \rightarrow MEM \rightarrow WB$$

State 的中間則以 register(紅色圓圈)去儲存之後可能要用到的 data。 就能以 state 為單位執行 instruction 了



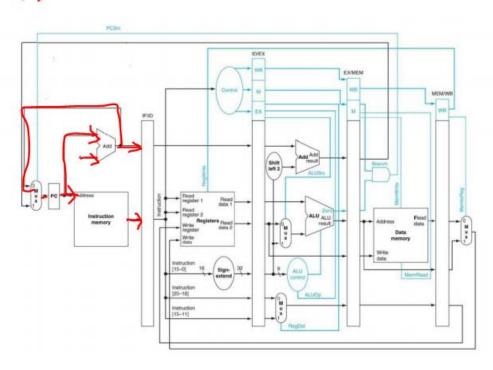
Example:

Test1.

(1) Addi \$1, \$0, 3;

將 PC 輸入 instrction memory, 並把讀取出的 instruction 暫存在 IF/ID Register 中。此外 PC+4 之後立刻傳回,才能在下一個 cycle 讀取下一個 instruction。並將 PC+4 暫存在 IF/ID Register 中。

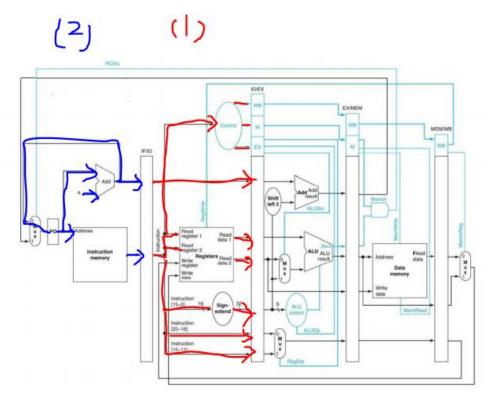
(1)



(2) Addi \$2, \$0, 4;

下一個 PC 輸出→下一個 instriction 存入 IF/ID Register,再把 PC+4 回傳。

IF/ID Register 輸出原本儲存的 instruction 進行 Register 讀取、Decoder,並儲存在 ID/EX Register。

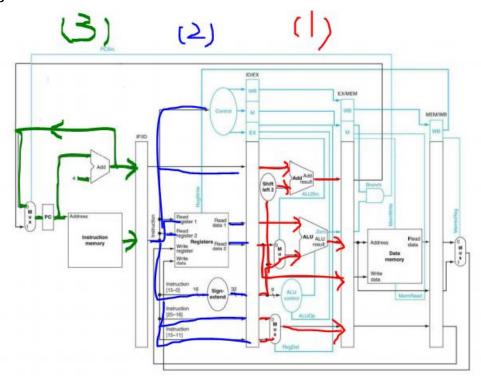


(3) Addi \$3, \$0, 1;

PC 輸出→下一個 instruction 產生並存至 IF/ID Register, PC+4 回傳。

IF/ID Register 輸出讀取 Register 的值和 Control decode。並傳入 ID/EX Register 儲存。

ID/EX Register 輸出 Rs, Rt 的值,送入 ALU 進行運算再傳入 EX/MEM Register。



以此類推, Pipeline 的功能就是分段、同時處理。

/TestBench/cpu/dk_i /TestBench/cpu/rst_n /TestBench/cpu/rst_pc /TestBench/cpu/pc_to_im_adder /TestBench/cpu/im_to_IF /TestBench/cpu/IF_ID_i /TestBench/cpu/IF_ID_o /TestBench/cpu/IF_ID_o /TestBench/cpu/Rdata1 /TestBench/cpu/Rdata2 /TestBench/cpu/WriteData			000002c 0000028 0000000 000002c 000002c00 00000088c 0000000	(32'h0000)(3 (32'h2001)(3 (32'h0000)(3 (64'h0000)(6	2'h0000 (32'h00 2'h0000 (32'h00 2'h2002 (32'h20 2'h0000 (32'h00 4'h0000 (64'h00 4'h0000 (64'h00	00 (32'h0000 03 (32'had010 00 (32'h0000 00 (64'h0000	(32h0000) (32h0021) (32h0000) (64h0000) (64h0000) (32h000	32'h0000 32'h0000 32'h00022 32'h0000 64'h0000 32'h0000000: 100003
# Register= # r0= # r8=	0, r1= 1, r9=	3, r2= 0, r10=	4, r3= 3, r11=	1, r4= 0, r12=	6, r5= 0, r13=	2, r6= 0, r14=	7, r7= 0, r15=	1 0
# r16= # r24=	0, r17= 0, r25=	0, r18= 0, r26=	0, r19= 0, r27=	0, r20= 0, r28=	0, r21= 0, r29=	0, r22= 128, r30=	0, r23= 0, r31=	

3. 遭遇的困難與解決方法:

一開始連PC都跑不進去,以為是哪條線沒有接好,一條一條寫下來慢慢對, 還是沒有發現錯誤。後來才突然看到是rst_n 寫成rst_i。

PC 跑進去之後,r0 的值一直會被改掉,發現是 WriteData = xxxxxxxxxxx 的時候 r0 會被改寫,RegWrite 從一開始就是1 了,還沒等到第一步要寫入的時候 RegWrite 就已經回傳1,才導致後面的錯誤。後來把 ALU Default 時也 output 出0 才不會被改寫。

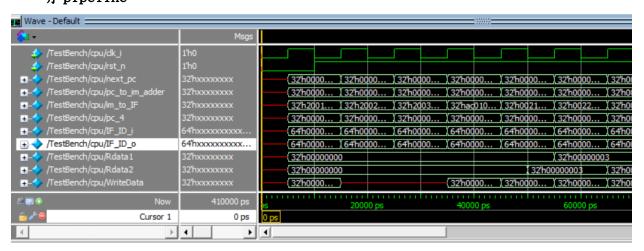
因為 Pipeline 的 intruction 是一直接下去的,所以看波形圖的時候還要算 到底甚麼時候才到某個 state。

4. 作業心得討論:

看波形圖的時候才深刻感受到 pipeline 跟沒有 pipeline 的差別: 沒有 pipeline:

ù v	Msgs												
+	32'h00000008		32'h000	. (32'h000	. (32'h000	(32'h000	32'h000.	(32'h000	32'h000	32'h000	32h000	32'h000	(32'h000)
∓- <pre> /TestBench/cpu/n_pc3 </pre>	32'h00000004		32'h000	. (32'h000	. (32'h000	32'h000	32h000.	(32'h000	32'h000	32'h000	32h000	32h000	32'h000
∓– ♦ /TestBench/cpu/instr	32'h20010001		32'h200	. (32'h200	. (32'h200	32h200	32h200.	(32'h080	32'hac0	32'hac0	32'h8c0	32'h8c8	32'h002
₽- /TestBench/cpu/Rdata1	32'd0		32'd0									(32'd4	(32'd1
₽- /TestBench/cpu/Rdata2	32'd0		32'd0	32'd0	(32'd0	(32'd0	32'd0	(32'd0	32'd1	(32'd2	32'd0	(32'd0	(32'd3
∓- /TestBench/cpu/Sign_Extend	32'd1		32'd1	32'd2	(32'd3	32'd4	32'd5	32'd8	32'd0	32'd4	32'd0		32'd16416
	32'd4	3	32'd4	32'd8	32'd12	32'd16	32'd20	32'd32	32'd0	32'd16	32'd0		32'd65664
	32'd1		32'd1	32'd2	(32'd3	32'd4	32'd5	32'd8	32'd0	32'd4	32'd0		(32'd3
± → /TestBench/cpu/ALU_result	32'd1		32'd1	32'd2	(32'd3	(32'd4	32'd5		32'd0	(32'd4	32'd0	(32'd4	
	5'd1		5'd1	(5'd2	(5'd3	(5'd4	5'd5	(5'd0	(5'd1	(5'd2	5'd6	(5'd7	(5'd8
∔	32'dx	_									32'd1	(32'd2	
+	32'd1		32'd1	32'd2	(32'd3	(32'd4	32'd5		32'd0	(32'd4	32'd1	(32'd2	(32'd4
₩	2510 ns	n i i i i	min	20 ns	minhin	40 ns	mili	60 ns	milin	80 ns	milim	100 ns	
Cursor 1	Cursor 1 5 ns		ns										

有 pipeline:



有 pipeline 的速度比較快,可以清楚的看到每個 state 的都在執行不一樣的 instruction。沒有 pipeline 就是規規矩矩的一個一個執行。

