

Data Memory. v : 模擬外部記憶體。需要允許寫入的訊號才可以在 Positive Clock Edge 時將想寫入的數值寫到指定的記憶體位址。一樣也需要允許讀取的訊號才能讀取。只要允許讀取，便會輸出記憶體位置所儲存的值。

TestBench. v : 設計實驗進行模擬。

Decorder:

Op	ALUOp	ALUSrc	RegWrite	RegDst	Branch	MemRead	MemWrite	MemoReg
R	000	0	1	01	0	0	0	0
ADDI	001	1	1	00	0	0	0	0
SLTi	010	1	1	00	0	0	0	0
BEQ	011	0	0	00	1	0	0	0
LW	100	1	1	00	0	1	0	1
SW	100	1	0	00	0	0	1	0
jump	100	0	0	00	0	0	0	0

ALU_Ctrl:

Op	ALUOp	function	ALUCtrl('b)
ADD	0	32	0010
SUB	0	34	0110
AND	0	36	0000
OR	0	37	0001
SLT	0	42	0111
JR	0	8	0000
MULT	0	24	1111

ADDi	1	X	0010
SLTi	2	X	0111
BEQ	3	X	0110
LW, SW	4	X	0010
ANDi	5	X	0000

Bonus

```

001000000000000100000000000010000 //Instr1 : addi r1 r0 16
000000000000000000000000000000000 //no
001000000000000110000000000001000 //Instr3 : addi r3 r0 8
001000000010001000000000000000100 //Instr2 : addi r2 r1 4
101011000000000100000000000000100 //Instr4 : sw r1 4(r0)
100011000000010000000000000000100 //Instr5 : lw r4 4(r0)
000000000000000000000000000000000 //no
000000000110000100110000001000000 //Instr7 : add r6 r3 r1
00000000100000110010100000100010 //Instr6 : sub r5 r4 r3
00100000001001110000000000001010 //Instr8 : addi r7 r1 10
000000000000000000000000000000000 //no
00100000000010010000000001100100 //Instr10: addi r9 r0 100
00000000111000110100000000100100 //Instr9 : and r8 r7 r3

```

Result :

CO_P4_test_1_result: 與 result1 reg、memory 相符

```

# r16= 0, r17= 0, r18= 0, r19= 0, r20= 0, r21= 0, r22= 0, r23= 0
# r24= 0, r25= 0, r26= 0, r27= 0, r28= 0, r29= 0, r30= 0, r31= 0
# Register-----
# r0= 0, r1= 3, r2= 4, r3= 1, r4= 4, r5= 2, r6= 7, r7= 1
# r8= 1, r9= 0, r10= 3, r11= 0, r12= 0, r13= 0, r14= 0, r15= 0
# r16= 0, r17= 0, r18= 0, r19= 0, r20= 0, r21= 0, r22= 0, r23= 0
# r24= 0, r25= 0, r26= 0, r27= 0, r28= 0, r29= 0, r30= 0, r31= 0
# Memory-----
# m0= 0, m1= 3, m2= 0, m3= 0, m4= 0, m5= 0, m6= 0, m7= 0
# m8= 0, m9= 0, m10= 0, m11= 0, m12= 0, m13= 0, m14= 0, m15= 0
# r16= 0, m17= 0, m18= 0, m19= 0, m20= 0, m21= 0, m22= 0, m23= 0
# m24= 0, m25= 0, m26= 0, m27= 0, m28= 0, m29= 0, m30= 0, m31= 0

```

CO_P4_test_2_result: 與 result2 reg、memory 相符

```
# r16= 0, m17= 0, m18= 0, m19= 0, m20= 0, m21= 0, m22= 0, m23= 0
# m24= 0, m25= 0, m26= 0, m27= 0, m28= 0, m29= 0, m30= 0, m31= 0
# Register=====
# r0= 0, r1= 3, r2= 4, r3= 1, r4= 6, r5= 2, r6= 7, r7= 1
# r8= 1, r9= 0, r10= 3, r11= 0, r12= 0, r13= 0, r14= 0, r15= 0
# r16= 0, r17= 0, r18= 0, r19= 0, r20= 0, r21= 0, r22= 0, r23= 0
# r24= 0, r25= 0, r26= 0, r27= 0, r28= 0, r29= 0, r30= 0, r31= 0
# Memory=====
# m0= 0, m1= 3, m2= 0, m3= 0, m4= 0, m5= 0, m6= 0, m7= 0
# m8= 0, m9= 0, m10= 0, m11= 0, m12= 0, m13= 0, m14= 0, m15= 0
# r16= 0, m17= 0, m18= 0, m19= 0, m20= 0, m21= 0, m22= 0, m23= 0
# m24= 0, m25= 0, m26= 0, m27= 0, m28= 0, m29= 0, m30= 0, m31= 0
```

Problems you met and solutions:

Decorder.v 的部分從 lab2 去更改,新增了一些參數,也順便更動 ALU.v 跟 ALU_C.v, 而且畢竟它是 CPU 的核心部分,解碼錯會導致它全盤錯掉,所以要好好賦值。一個值的影響很大,只好不斷檢驗。

同 Lab2, 最困難的依然是把所有的小程式併在一起,也就是 CPU.v, 而且這次新增了新區塊,還要弄成 Pipe 版本,相當燒腦費神,還是一樣一個沒弄好,輸出都有問題,依然要花很多時間思考研究和整理每條電路的連接,然後宣告跟填上相對應的參數,常常想錯參數就會擺錯位置,就導致輸出有問題,完全弄懂 CPU 後參數才放對,也才能看到輸出,命名也很重要,因為這樣才比較不會亂掉。這次更重要的是要弄對 reg 的大小,真的很容易數錯。還有 result 要去了解 Bonus,不然會一直覺得不合。

Summary:

這次依然要模擬外部記憶體。一樣是只要把每個小部分完成,project 就大致做完了,每一小份不難,但把他們合在一起就很有難度,整併很棘手尤其是這次程式又大了,對每個細節必須更用心。先弄懂 CPU 的實作後,比較困難的只有接線填參數(Pipe_CPU_1.v)和解碼(Decorder.v),還有要記得讀寫檔的路徑問題。

在 Lab 中,因為有記憶體的加入,更貼近完善的 CPU,我更加理解 CPU,也更了解 CPU 的實作方法,越來越像是一台電腦的核心,能用軟體模擬硬體,這真的提供不小的便利性。