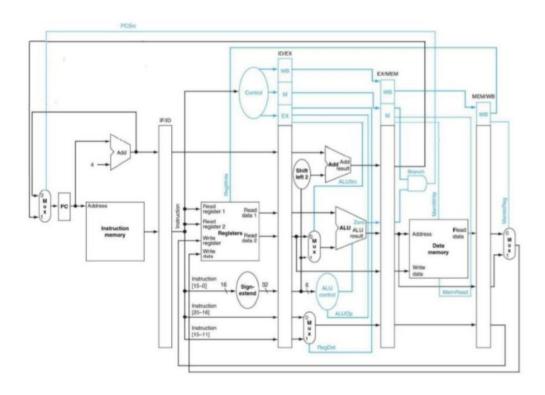
Computer Organization Lab4

Architecture diagrams: 直接用附圖



Hardware module analysis: ModelSim

Adder.v:實作加法就好,將兩個數值相加

ALU.v: 實作 ALU,可以直接參考 PDF 的 Appendix,基本的運算。

ALU_Ctrl.v: 決定在 ALU.v 的 operation,用 switch…case…實作

Decorder. v: 將 instruction 轉變成實作用的 code,以及對應的

ALU Ctrl,用 switch…case…實作

MUX_2tol.v: 兩個可能的多功器,套if…else or .. ? .. : .. 即可

Shift Left Two 32.v: 將所有位數左移 2 即可,利用"<<"

Sign Extend.v: 重複 16 次 MSB 再 concatenate input value

Pipe_CPU_1.v: 將之前的CPU 改成 Pipe 版,只是要加上 IF, ID 等區域,還是一樣將小程式組裝起來即可。

Instruction Memory. v: 紀錄所有 Instruction 資訊的記憶體,只在 Positive Clock Edge 時才可輸出值的改寫。

Pipe_Reg. v:將Reg變成Pipe版

Reg File. v: 將值存進暫存器

Program Counter(PC). v: 計數器,只在 Positive Clock Edge 時才可輸出值的改寫,決定要取得的 Instruction Memory 的 Address。

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:

0p	ALUOp	ALUSr	RegWrite	RegDst	Branch	MemRead	MemWrite	MemtoReg
		С						
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:

Ор	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

```
0010000000000010000000000010000
                           //Instrl : addi rl r0 16
//no
0010000000000110000000000001000
                           //Instr3 : addi r3 r0 8
00100000010001000000000000000100
                           //Instr2 : addi r2 r1 4
10101100000000010000000000000100
                           //Instr4 : sw
                                        r1 4(r0)
r4 4(r0)
                           //Instr5 : lw
//no
0000000011000010011000000100000
                           //Instr7 : add r6 r3 r1
00000000100000110010100000100010
                           //Instr6 : sub r5 r4 r3
0010000001001110000000000001010
                           //Instr8 : addi r7 r1 10
//no
0010000000010010000000001100100
                           //Instr10: addi r9 r0 100
                           //Instr9 : and r8 r7 r3
00000000111000110100000000100100
```

Result:

CO_P4_test_1_result: 與 result1 reg、memory 相符

z16=	0, m17-	0, m10-	0, m19-	0, =20-	0, =21-	0, m22-	0, m23-	0
m24= Register	0, m25=	0, m26=	0, m27=	0, m26=	0, m29=	0, m30=	0, m31=	.0
r0=	0, rl-	3, 12-	4, 23-	1, 14-	6, 25-	2, 16-	7, 27-	1
rt-	1, 25=	0, :10-	3, zll=	0, 112=	0, 113-	0, z14=	0, 115	. 0
r16=	0, =17=	0, 118-	0, 119-	0, r20	- 0, r21	- 0, r22-	0, z	23= 0
r24=	0, 125=	0, 226-	0, 227=	0, 128	- 0, r25	- 0, r30-	0, r	31= 0
Hemory								
m0=	0, m1-	3, m2=	0, m3-	0, m4-	0, m5= 0	, пб- 0, п	7= 0	
nt=	0, ms-	0, mlo-	0, ml1=	0, ml3-	0, m13=	0, m14-	0, mi5=	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, m26-	0, m29=	0, m30=	0, m31=	0

CO P4 test 2 result: 與 result2 reg、memory 相符

z16=	0, m17-	0, mis-	0, m19-	0, m20+	0, m21-	0, =22-	0, m23-	0
m24= Register	0, m25-	0, m26=	0, m27=	0, m20=	0, m29-	0, m30=	0, m31=	0
20-	0, ri-	3, 12-	4, 23-	1, 14-	6, I5=	2, 16-	7, 17=	1
r0+	1, 19-	0, r10=	3, :11=	0, 112=	0, ri3=	0, 214-	0, #15=	0
r16=	0, 117=	0, zis-	0, 119=	0, 120-	0, r21=	0, 122-	0, 123	
z24=	0, ±25=	0, 226=	0, 227=	0, r28=	0, r25=	0, 230=	0, 231	0
Memory								
m0=	0, ml=	3, =2=	0, ml-	0, m4=), m5= 0,	m6= 0, m7	- 0	
mi=	0, m9=	0, mil=	0, mli=	0, ml2=	0, ml3=	0, m14=	0, m15=	0
r16-	0, m17=	0, mis-	0, m19-	0, m20+	0, m21-	0, m22-	0, m23-	0
m24=	0, m25=	0, m26=	0, m27=	0, m26=	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。