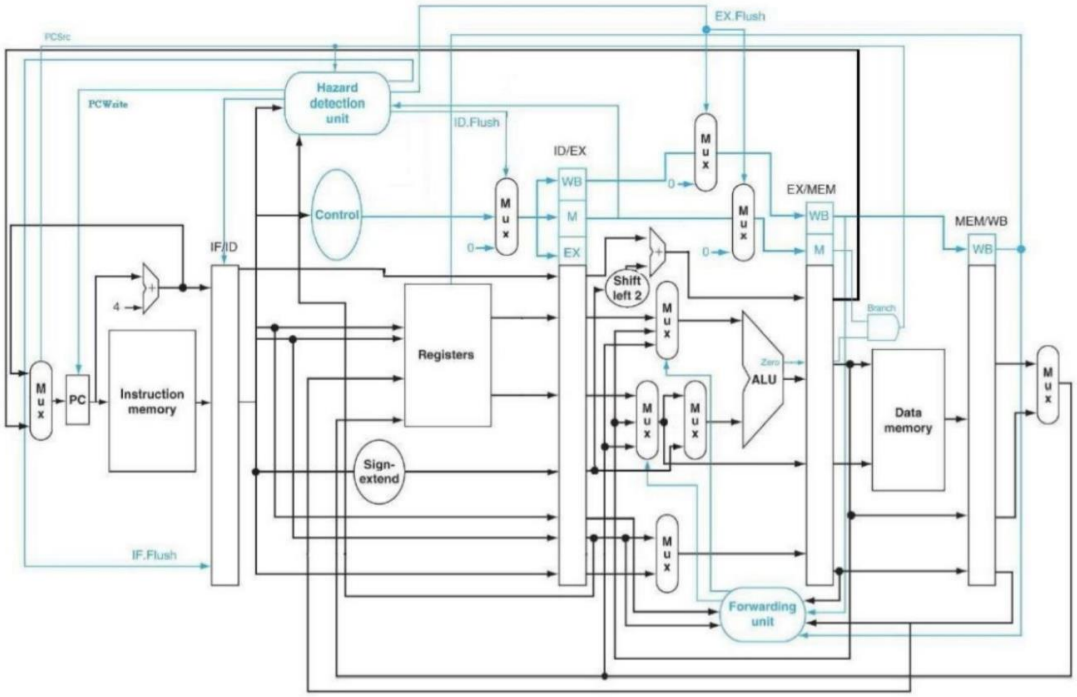
Computer Organization Lab 5

Architecture diagrams: 直接用附圖



Hardware module analysis: ModelSim

大部分同 Lab4

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。

Forwarding. v:將一些區塊利用 forward 的方式解決 Data Hazard，並設立新變數 Flush 已清空造成 stall 的 bubble

HazardDetection. v:透過教授教的方法偵測 Data Hazard(EX hazard 跟 MEM hazard 跟 double hazard 時的 MEM hazard with no EX hazard)

Decorder:

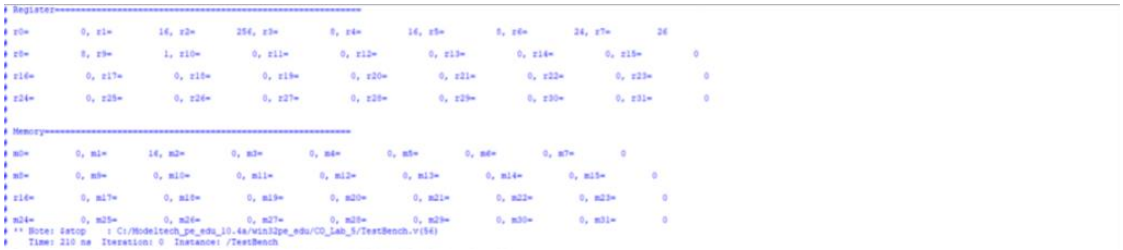
Op	ALUOp	ALUSrc	RegWrite	RegDst	Branch	MemRead	MemWrite	MemtoReg	BranchType
R	000	0	1	01	0	0	0	0	0
ADDI	001	1	1	00	0	0	0	0	0
SLTi	010	1	1	00	0	0	0	0	0
BEQ	011	0	0	00	1	0	0	0	0
LW	100	1	1	00	0	1	0	1	0
SW	100	1	0	00	0	0	1	0	0
jump	100	0	0	00	0	0	0	0	0
BNE	101	0	0	00	1	0	0	0	3
BGE	001	0	0	00	1	0	0	0	2
BGT	111	0	0	00	1	0	0	0	1

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

Result



Problems you met and solutions:

這次最大的問題就是解決新增的兩個區塊，Hazard_detection_unit 和 Forwarding_unit，而難處就在於它的線會連到許多其他區塊的地方，所以有時候資料容易讀錯，因此一定要確保 module 內是我們要的。

而 stall 的地方也很困難，因為它的出現造成更多的 control bit 和史的 MUX 變得更為複雜，而後續還有對其他區塊的相關操作就更容易有問題了，所以一定要維護好 MUX 的輸入值。

因為新增了 compare 的指令，所以在 decorder 要有其他變動好好分析，還要在硬體設計上動點手腳便可以解決。

同以往的 Lab，最困難的依然是把所有的小程式併在一起，也就是 CPU.v，何況這次還有 stall 的問題，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。

Summary:

在這次 Lab 中，主要是增加 Hazard_detection_unit 和 Forwarding_unit，以及增加更多 compare 的指令，讓程式更貼近完整的 CPU，也能解決其實應該存在的 Hazard，靠 Forwading 解決一些，以及 load use 的時候要 stall。而這次要懂得清空 controling code 將一些資料清空形成 bubble。總而言之，這次作出了很完整的有 pipeline 的簡單 CPU。配合課程內容，了解到 CPU 模擬的情形。