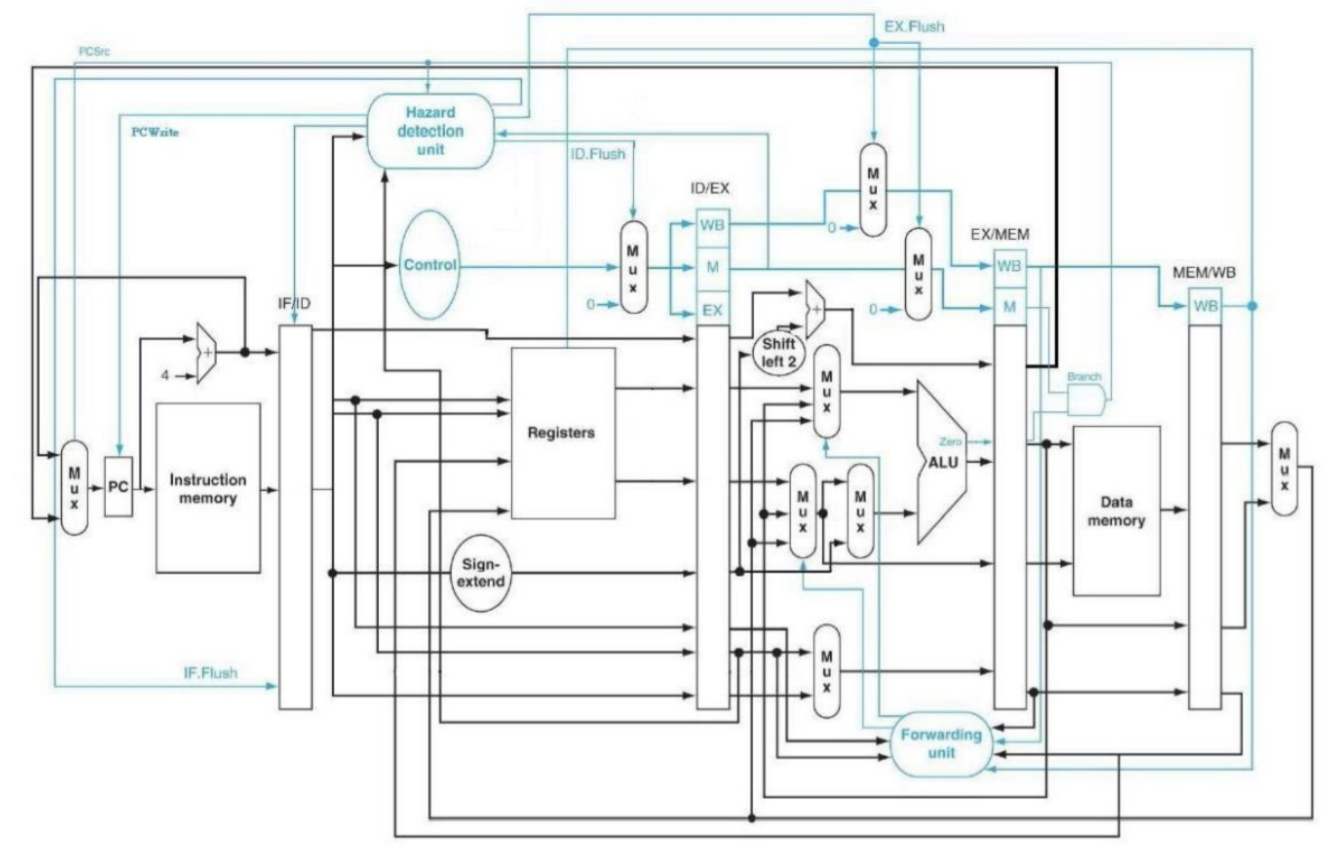
**Computer Organization Lab 5**

**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\_2to1.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)

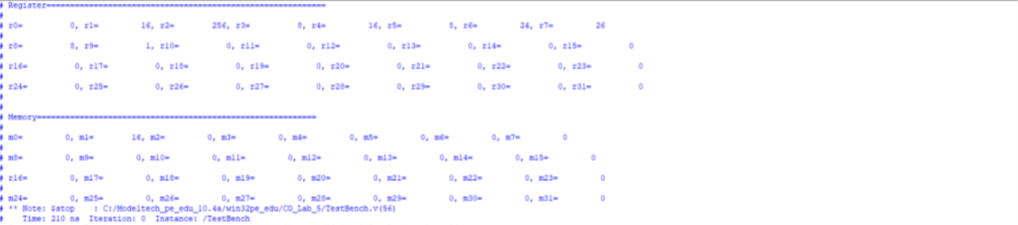
**Decorder:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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。