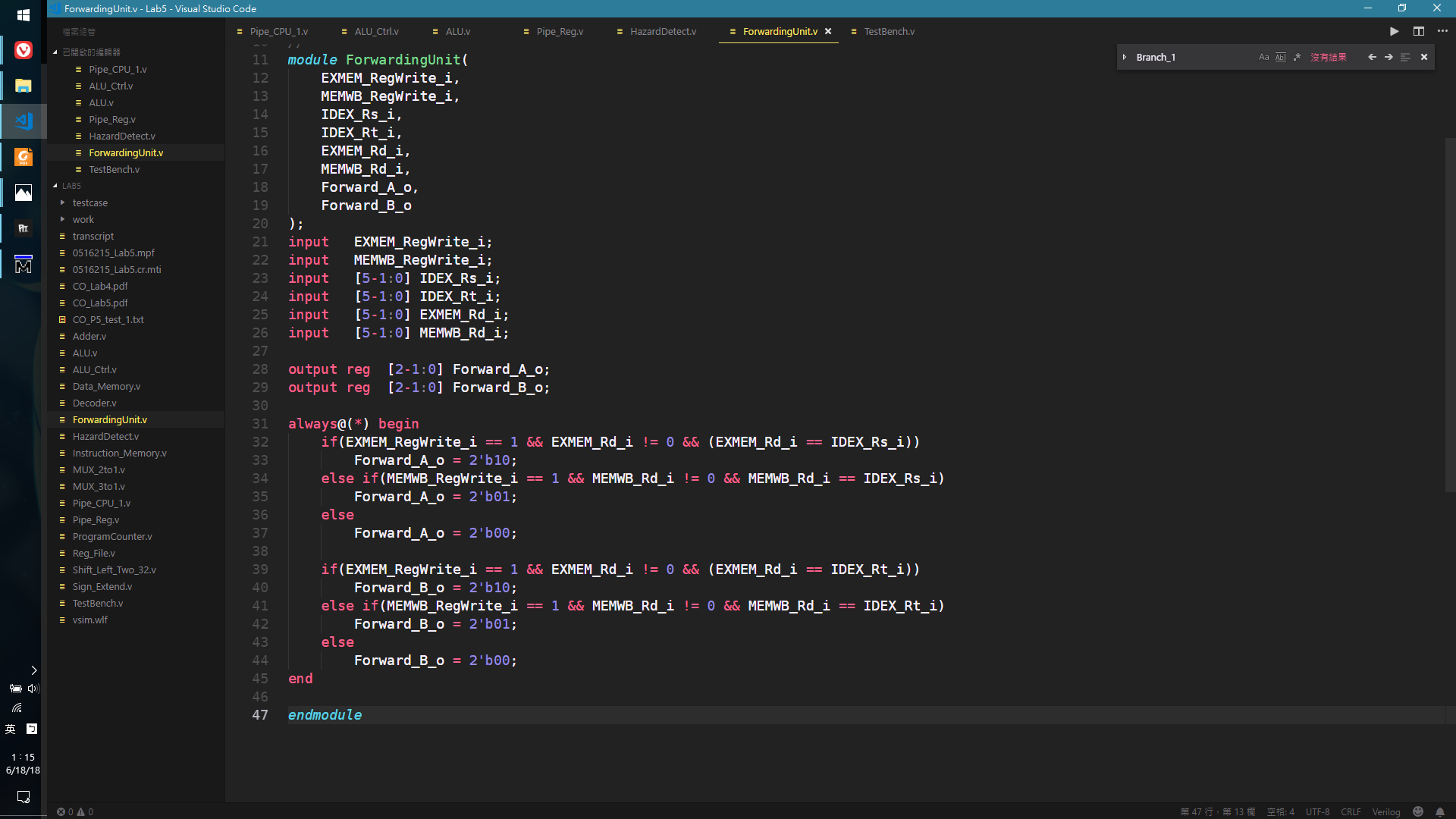
**Computer Organization**

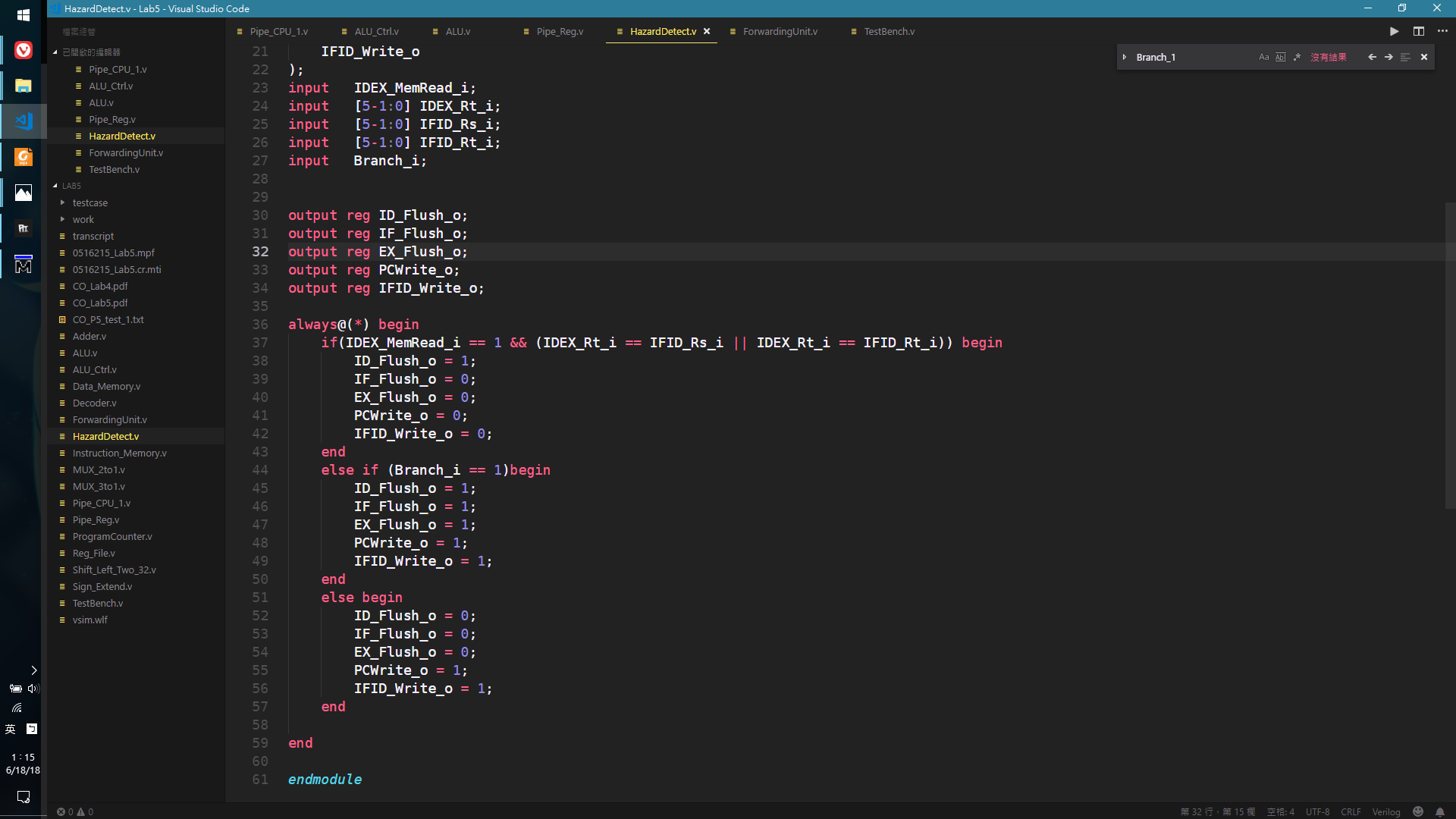
LAB5 0516215 林亮穎

**Source code and the note:**

這次作業最大的不同就是加入了Hazard\_Detection和Forwarding\_Unit，我大概簡單講一下我是怎樣來做的：



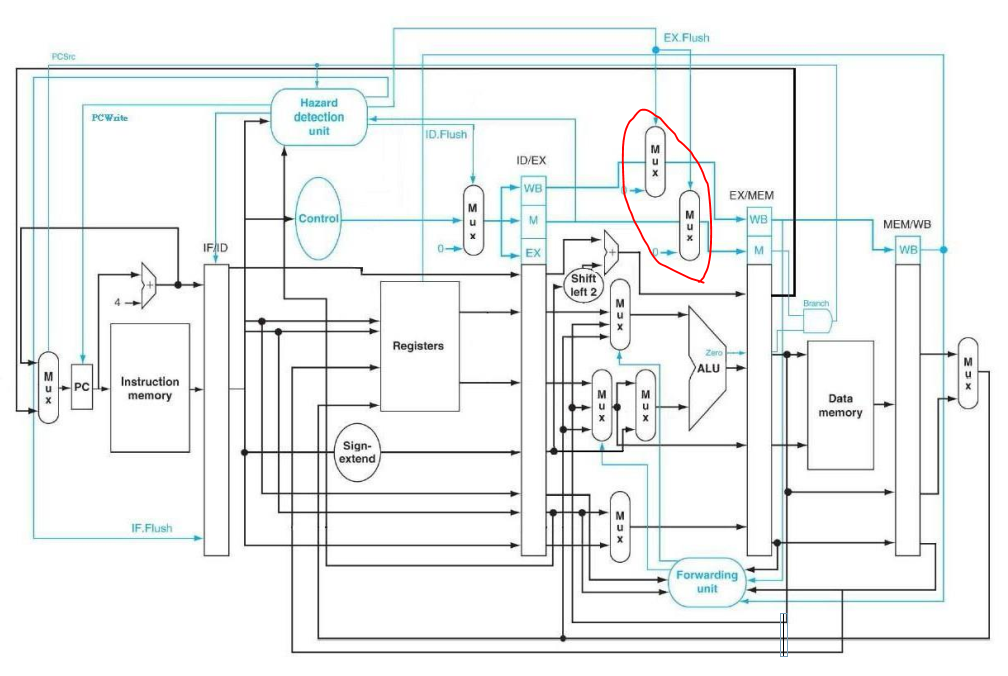
Forwarding\_Unit主要是在處理data hazard，我的偵測方式和課本上差不多，以Register RS為例，當發生EX Hazard時，Forward\_A設為10，ALU的input1會從MEM stage forward過來；而當MEM Hazard時，Forward\_A設為01，ALU的input1會從WB stage forward過來，這邊由於用了else if，判別是不用寫得像課本上那麼長。

****

Hazard\_Detection\_Unit主要是在處理load-use data hazard以及control hazard發生時，要讓pipeline裡面部分的stage作stall，當load-use hazard發生時，flush stage ID，並且將PC和IF/ID pipe\_reg”冰凍”起來，禁止去update他們，以便抓到的instruction可以在stall完後繼續使用。而control hazard是在MEM stage做完branch的判定後，結果丟回來這個

Hazard\_Detection\_Unit，若是要做branch，就將branch instruction後面三個已經被抓進來pipeline裡面的instruction清掉，也就是flush stage IF、stage ID、stage EX這三個stage，之後才能讓branch address進到PC。若是沒碰到上述兩種hazard，則pipeline就如常執行，不需要作stall，PC和IF/ID pipe\_reg皆可正常update。

**Architecture diagrams:**

1. pipeline CPU電路圖

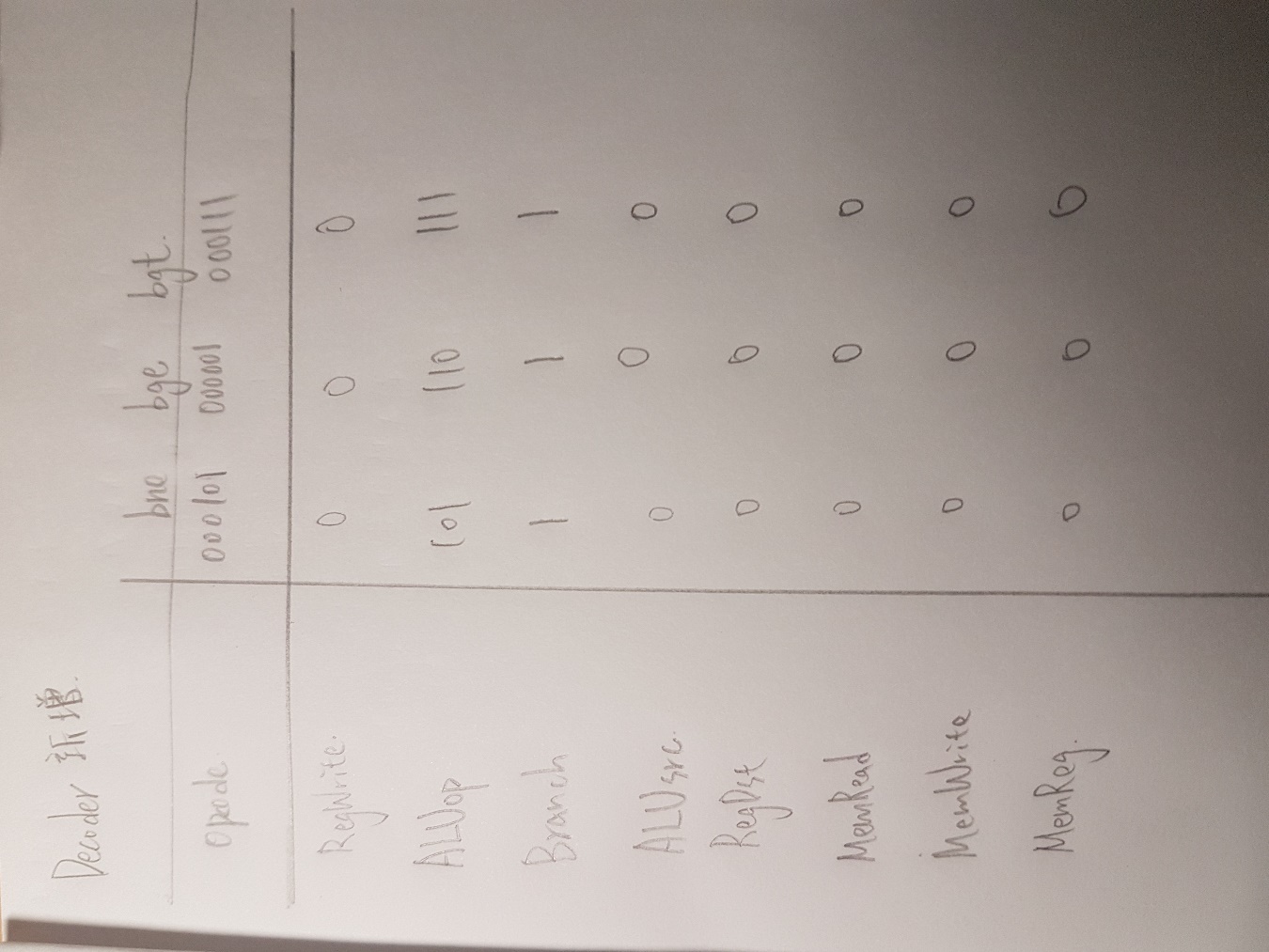
此圖參考自spec上的電路圖，但是有作一些修改：

(1) 紅圈處用來flush EX stage的2個MUX，由於他們總是會被同時flush，我把它合併成一個MUX。

(2) stage IF的flush我並不是像圖中直接去flush IF/ID pipe\_reg，而是將pc+4和PC的data接到一個新的MUX，再將IF flush的signal接到MUX上。這樣做可以不用修改pipe\_reg的架構，稍微方便一點。

2. Decoder真值表

加入了bne、bgt、bge



其中ALUop分別設為5、6、7，方便之後設計ALUctrl和丟入ALU作各自特別的運算

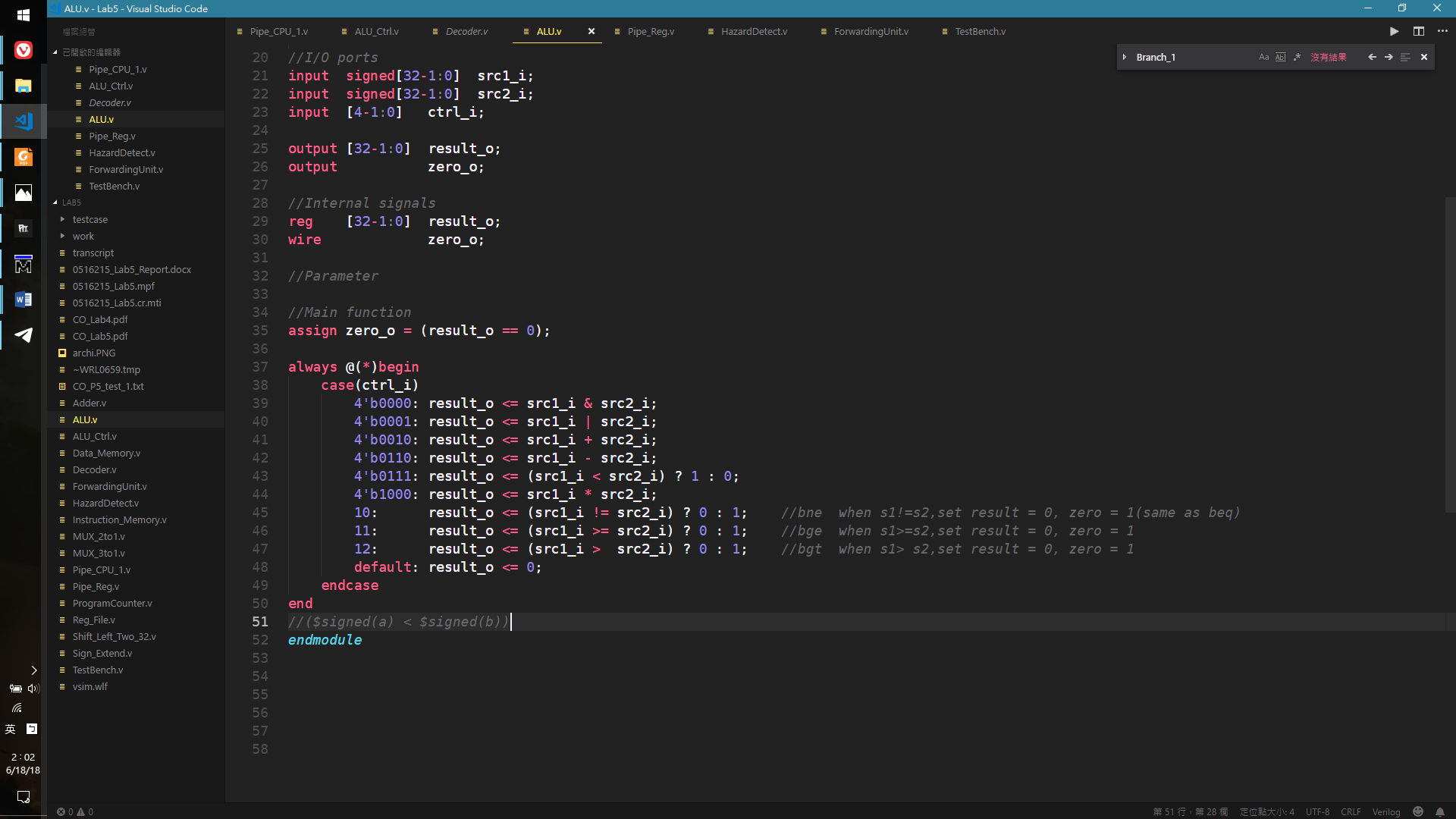
以下僅列出本次作業新增加或修改的Module：

3. ALU\_Ctrl

加入了bne、bgt、bge，input的ALUop分別為5、6、7，對應output的ALUctrl分別是10、11、12。

4. ALU

放上code比較方便說明，如下：



以bne為例，對應的input ALUctrl是10，當rs != rt成立時，也就是符合bne要跳去branch address時，我故意將result設為0，這樣對應的output signal **Zero會是1**，先前用來判斷beq是否成立，PC要抓PC+4還是branch address的signal “**PCSrc = MEM\_Branch & MEM\_Zero**”

就可以繼續沿用來判斷bne是否要跳過去branch address。Bgt和bge也是同理。

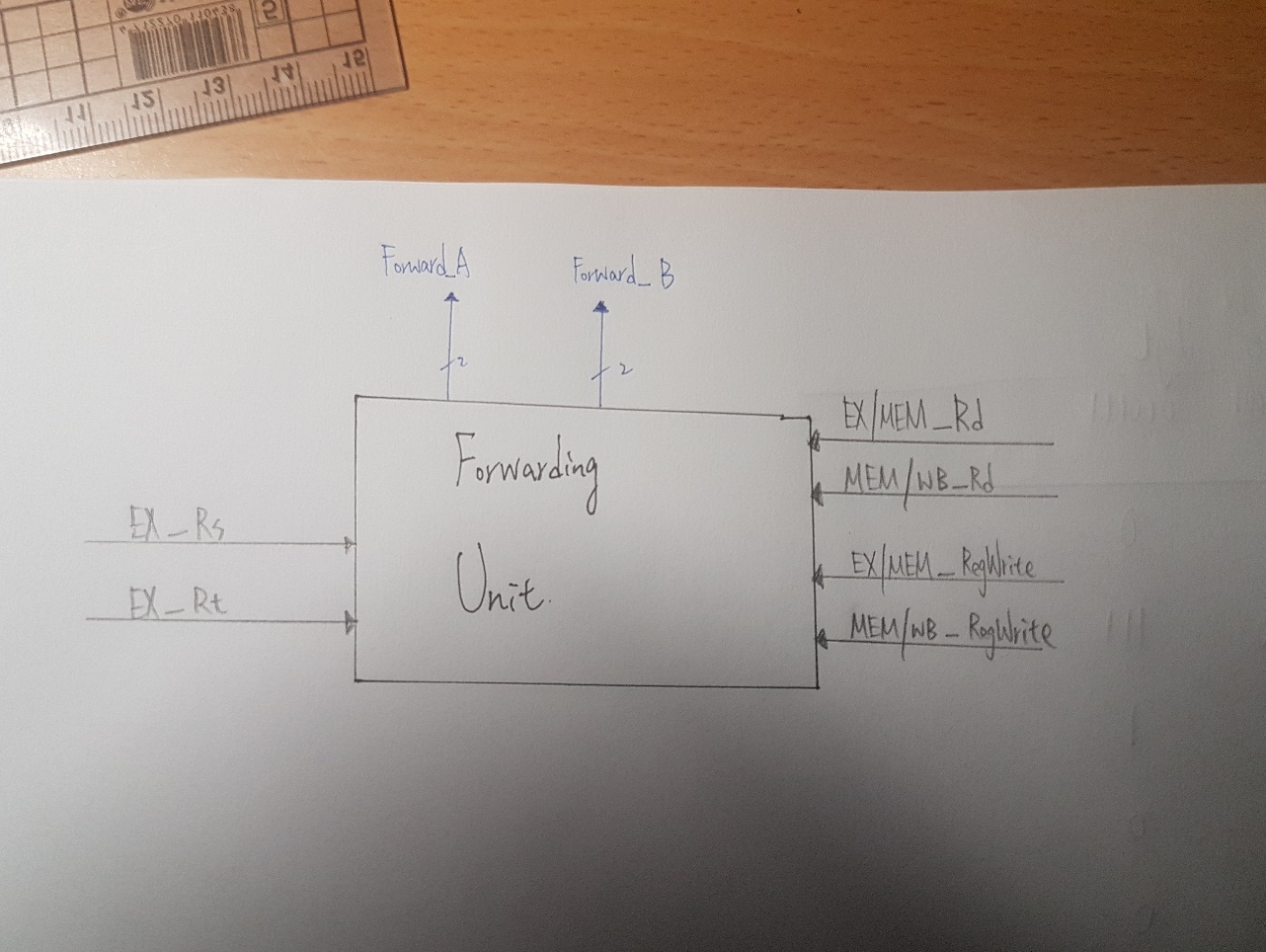
之所以想說要這樣設計，是因為前面的lab提供的判斷bne、bgt、bge的架構圖，必須加入一個MUX，以及一個判斷branch type的signal(可能要另外設計unit)，實在是有點複雜，因此這邊就用這種方法來盡量配合beq，盡量用原有的branch方法來作。

5. pipe\_reg

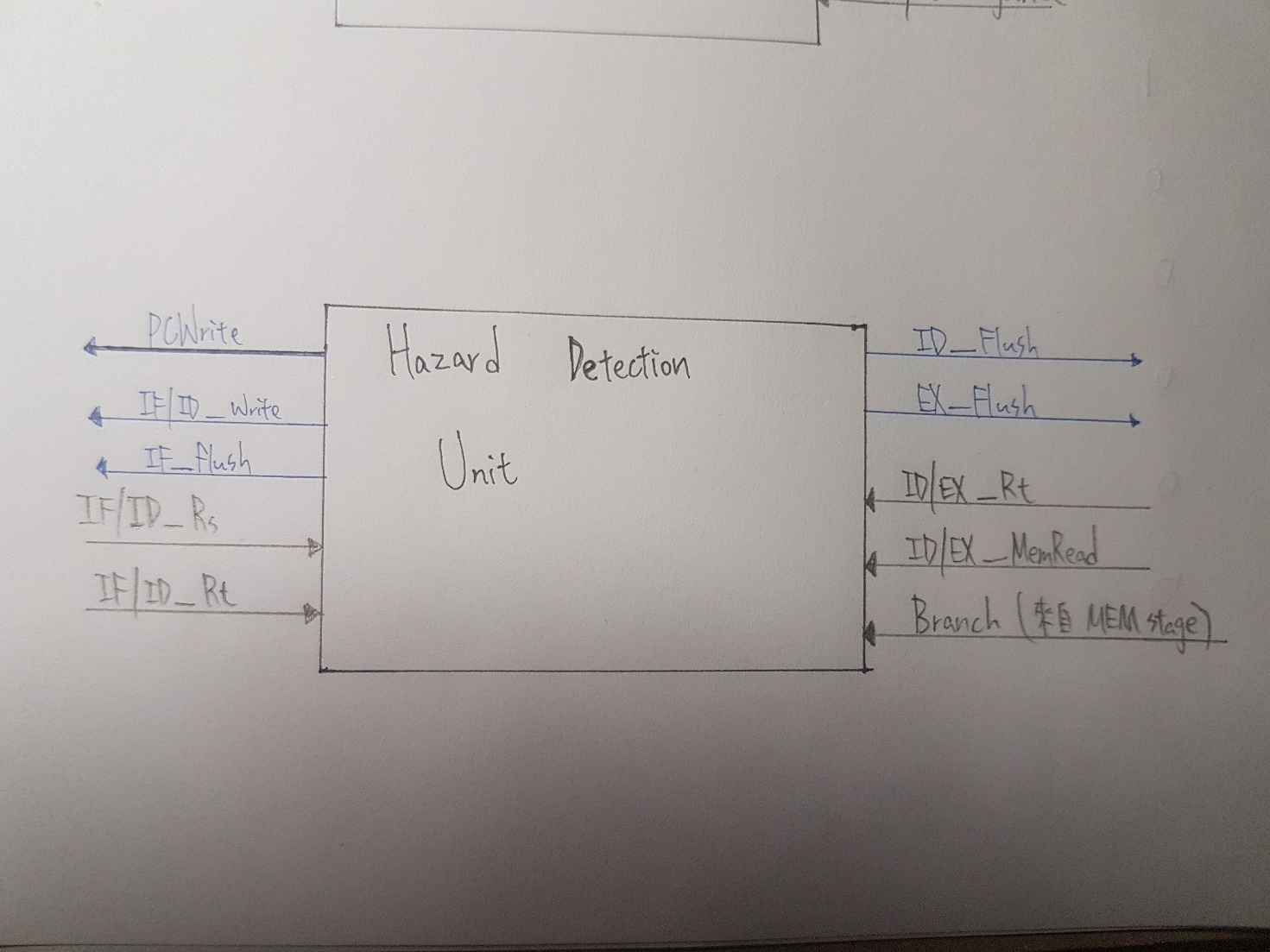
新增pipereg\_write的input，控制能不能寫入data到此pipe\_reg。

圖沒什麼改變，就不畫了。

6. Forwarding Unit



7. Hazard Detecton Unit

****

**Hardware module analysis:**

大多新增的module的功能及設計想法都在前面的段落提過了，這邊針對本次新增的兩個最重要的module總結一下：

1. Forwarding Unit

針對data hazard中的EX hazard以及MEM Hazard以forwarding的方式來解決，藉此來避免stall。對於EX Hazard，ALU的input會從MEM stage forward過來；對於MEM Hazard，ALU的input會從WB stage forward，而偵測這兩種hazard的判別式就如同前面段落描述的那樣

2. Hazard Detecton Unit

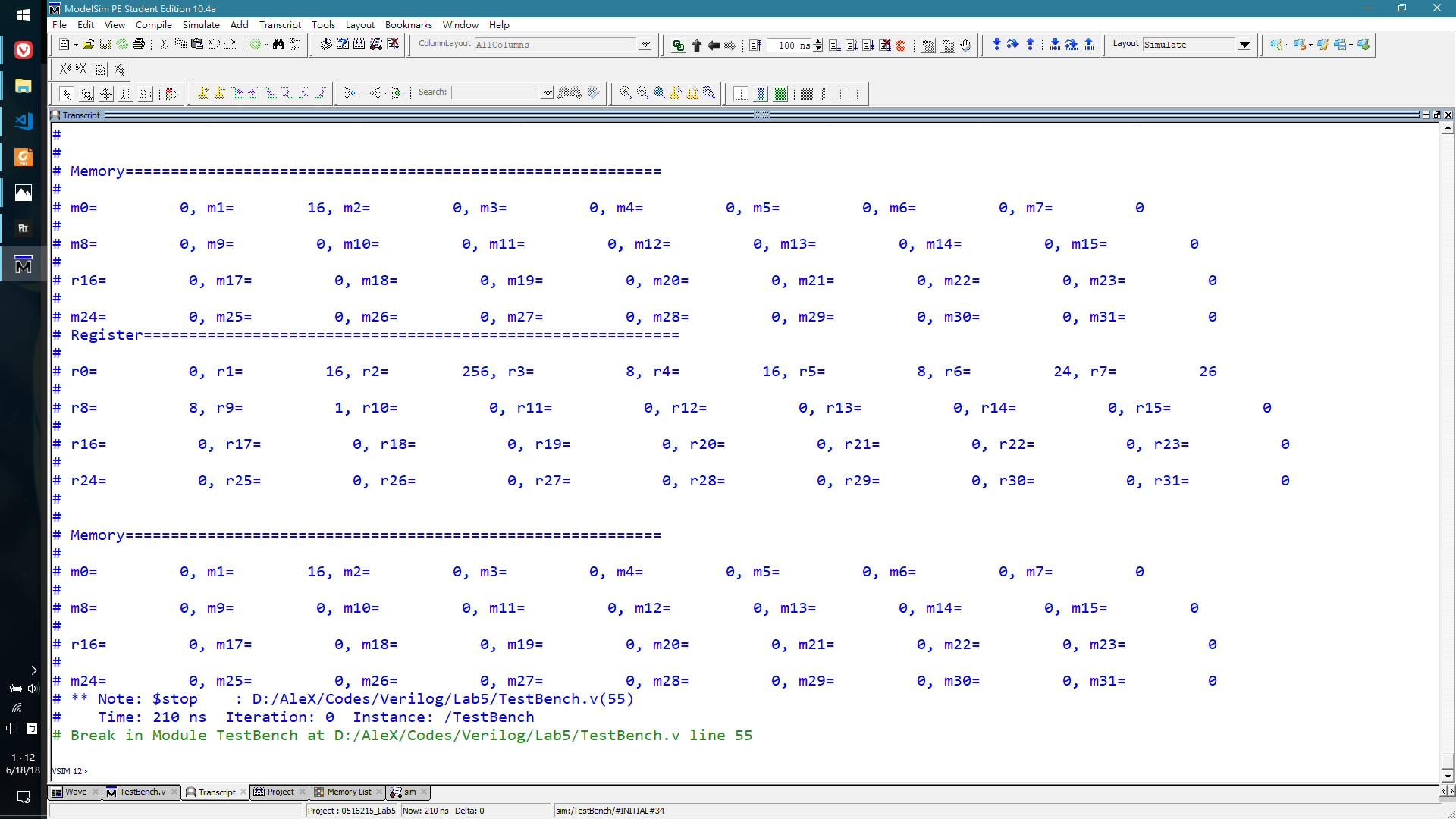
針對load-use data hazard以及branch hazard這兩種無法單純透過forwarding來解決的hazard，必須在pipeline中加入適當的bubble來處理。

load-use data hazard中，必須將IF/ID pipe\_reg以及Program Counter的保留下來，禁止作update，並且對ID stage作flush(將control signal設為0)。

branch hazard中，我猜測branch不會執行，因此會接著執行後面的instruction，直到stage MEM確定了到底要不要跳到branch address。若是要跳的話，必須對stage IF、stage ID、stage EX作flush，後面才能接著執行branch address的instruction。

**Finished part:**

1. 測資一輸出結果



**Problems you met and solutions:**

1. 上次pipe\_reg的寫法太冗長，因此這次改將多個訊號合併在同一個pipe\_reg。

2. PC和pipe\_reg這些之前給的元件都得加入新的input，debug的過程中有出過一些問題。

3. bne、bgt等新加的branch類指令，一開始沒想到要怎麼實作他們的branch，和朋友討論過後才想到可以用ALU 的output來作，避免新增元件和訊號。

**Division of work: (one-member teams don’t need to write)**

X

**Summary:**

這次作業是要完成一個可forwarding以及處理hazard並stall的pipeline CPU，做起來其實比前面的Lab複雜不少，不僅要接的線變的很多，許多原有的元件也要做修改，之前設計時想的不夠周全的元件也要改變寫法，也因此產生不少bug，實在是花了不少時間。再加上有許多科目都在期末考，所以做的非常趕，希望不要出什麼大問題。