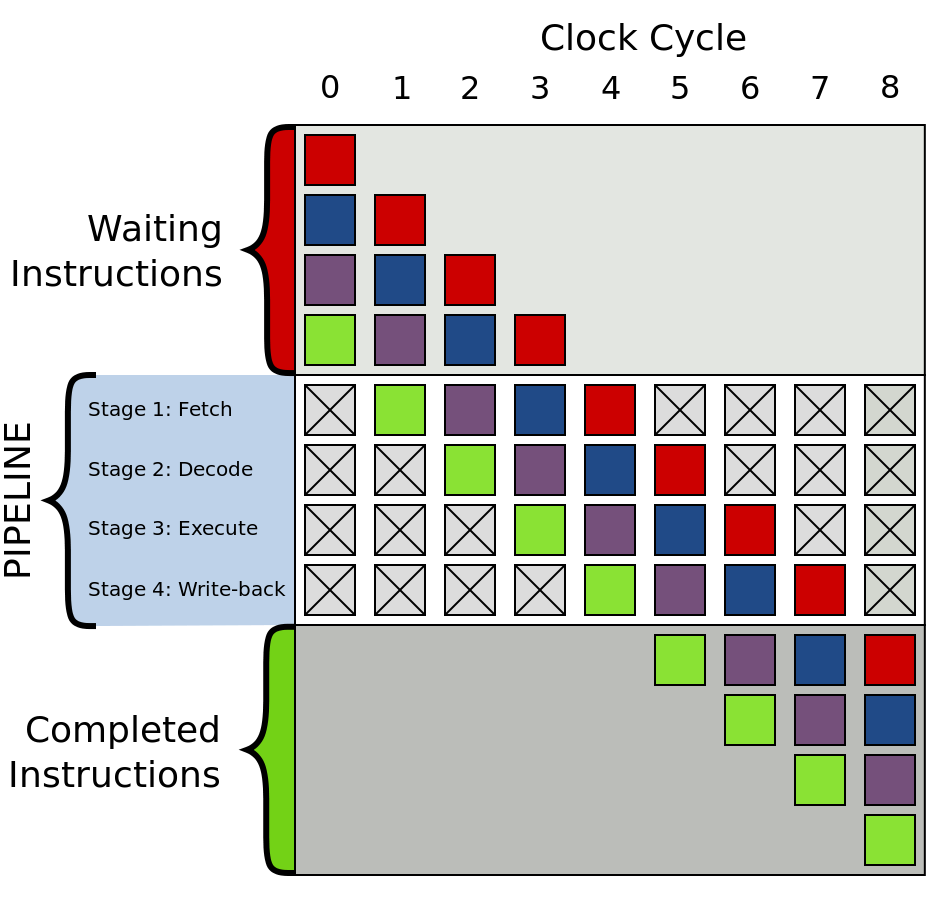
**計算機組織期末專案報告**

**Pipelined CPU Implementation**

**成員名單：**

A1005511 陳界亨

A1005523 姜博允

A1005526 王均元

A1005527 楊東祐

**指導老師：**

郭錦福教授

**一、大綱**

1.主程式碼說明

2.遭遇到的難題與解決方法

3.工作分配

4.專案報告之心得

**二、內容**

**1.主程式碼說明**

**a)準備階段(包含讀檔)**

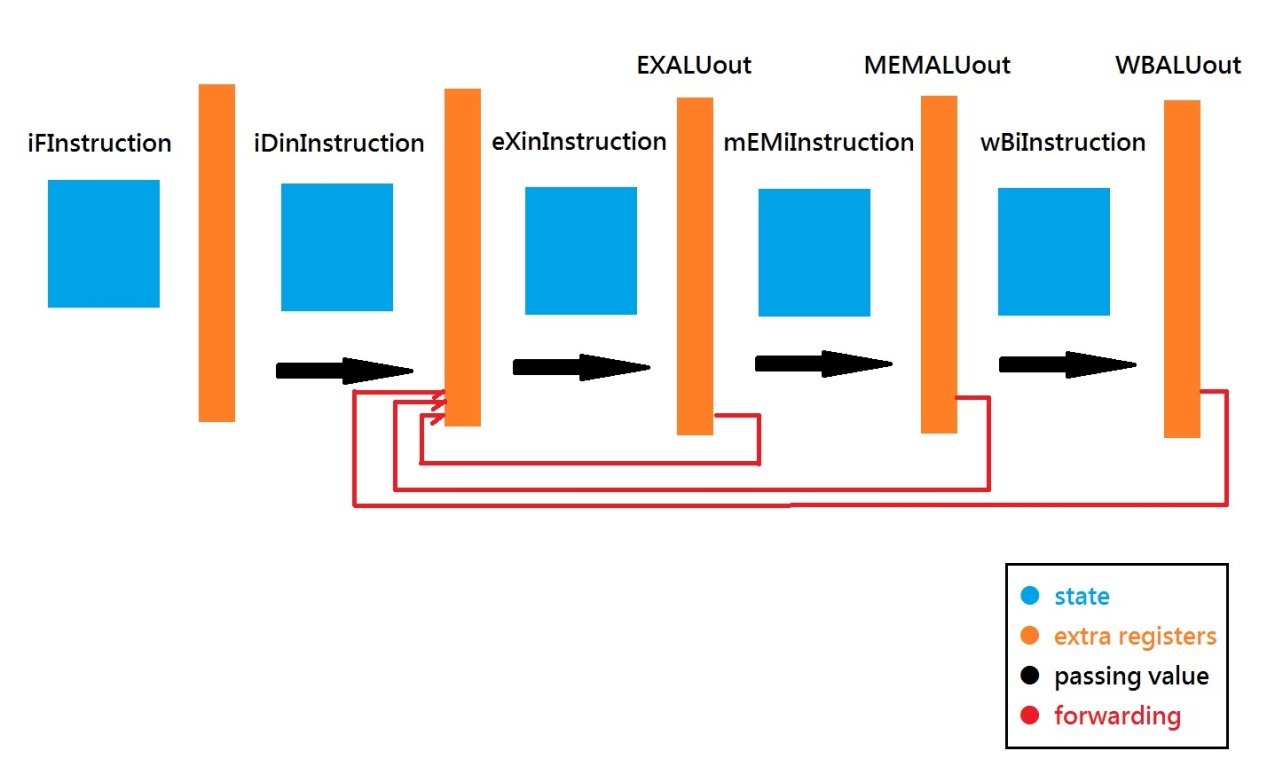
前置參數：

(1) [struct ]instruction

(2) [vector<instruction>]ArrayListOFinstructionlist

主要目的：讀入一行指令，儲存在instruction型態的struct中，再丟進vector的資料結構實作ArrayList

**b)執行階段**

****

前置參數：[ instruction ]zeroInstruction

[ instruction ]iFInstruction

[ instruction ]iDinInstruction

[ instruction ]eXinInstruction

[ instruction ]mEMiInstruction

[ instruction ]wBiInstruction

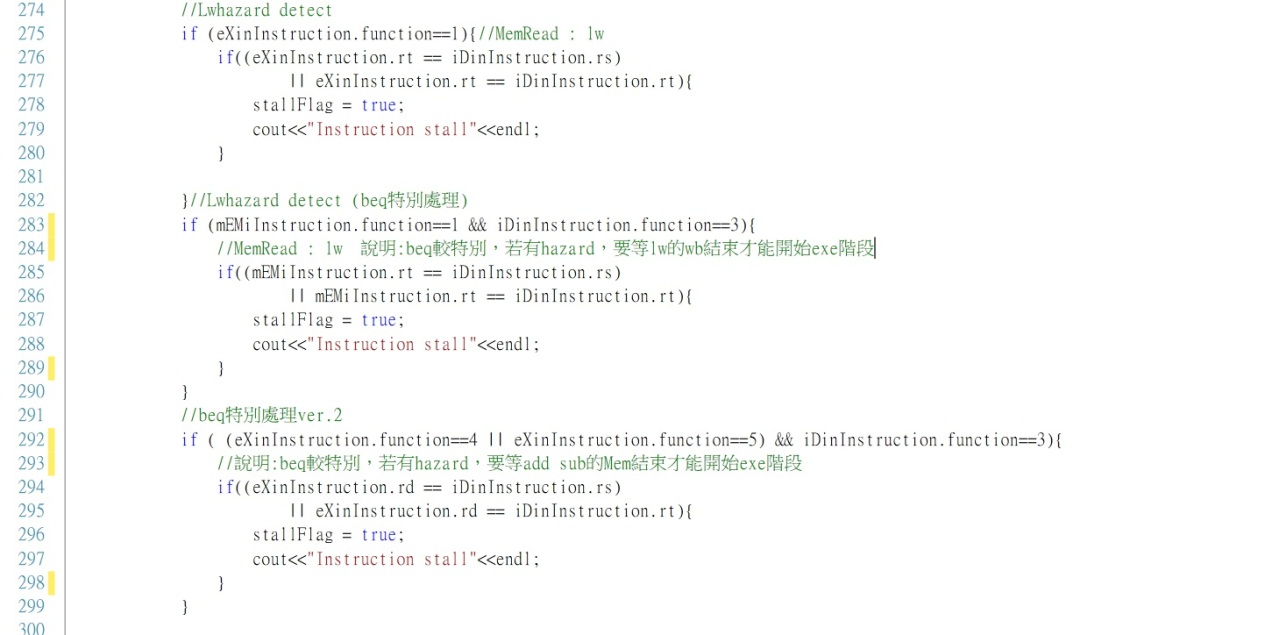
[ bool ]stallFlag

[ int ]EXALUout、MEMALUout、WBALUout

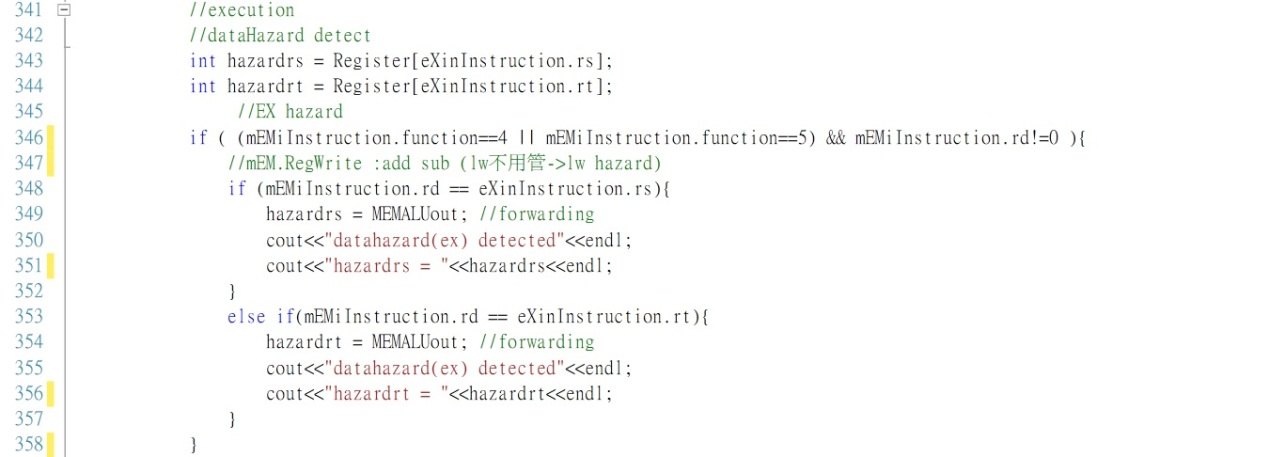
前置函數：Struct1EStruct2

主要目的 : 判斷兩struct是否相等，主要用來與zeroInstruction相比，判斷是否該階段已無指令。

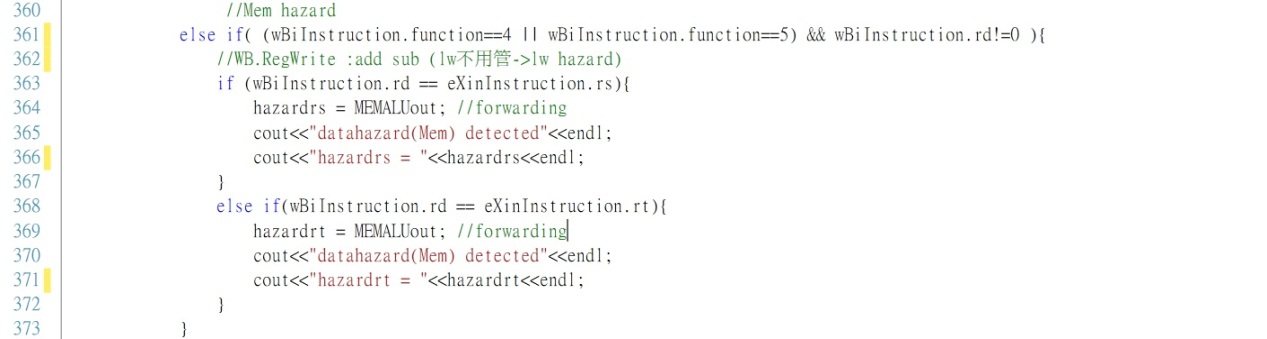
圖一~圖三主要目的：進行pipeline動作，檢查是否有hazard，依類型判斷做stall或forwarding



圖一、lw、lw +beq和beq特殊情況的hazard偵測



圖二、ex hazard偵測



圖三、mem hazard

**c)輸出階段**

主要目的：對運算結果輸出。

分為兩個階段：第一階段寫入cycle；第二階段寫入register、memory，並且以append模式寫入，防止洗掉先前的fileinput。

**2.遭遇到的難題與解決方法**

(1)原本不是很清楚memory和register的value和base address的關係，還沒開始實作就碰到了觀念上的大瓶頸

***解決辦法:***

詢問同學，釐清了原來register的值就是當base address

，因此memory的觀念問題總算解決，得以開始實作。

(2)這次每個階段牽扯到的變數太多難以一次處理，且常常有解決不完的記憶體錯誤

***解決辦法:***

把每個階段都建成一個名為instruction的struct，然後再用vector去模擬二維struct (類似ArrayList)的方式去處理這次的變數，不僅減少了原本使用動態二維陣列常產生的記憶體空間錯誤，也讓整個流程可以很順利的下去。

(3)由於不是給machine code，所以得做較複雜的字串處理，但原本只想到能用streamstring，碰到了讀檔的大瓶頸

***解決辦法:***

去C++ PLUS網站查看字串處理的相關函示庫，發現了可以使用sscanf去解決複雜的字串處理，只要先把rs 、rt 、rd欄位處理後再用空白隔開，即可丟到新的自訂變數上，因為sscanf連型態都幫忙轉好了，也解決了型態轉換這另一大問題。

(4)在處理完讀檔後不知道如何去實作pipeline的五階段，在傳遞指令的觀念上卡了很久

***解決辦法:***

經過幾天深思熟慮後，發現想法被拘泥在ppt上五階段的"左→右"方向了，應該看的是階段當下的"上→下"才對，意即應該按照先檢查beq和hazard的情況後，從WB階段處理回EXE階段才對，因為先進的指令要先做，而先做的指令理所當然的階段是在較後面的，因此改變了處理的順序後，總算突破了這個處理順序上的盲點。

(5)關於這次的beq處理方式較一般來說比較不同，因此一開始卡在不知道要在哪去特別處理beq的指令。

***解決辦法:***

經過幾天深思熟慮後，發現其實就是多做幾個例外的判斷去做stall而已，並沒什麼特別。

(6)寫完stall後想開始著手處理data hazard的forwarding，但不知道如何把上一個做完的馬上丟給下一個指令用

***解決辦法:***

後來發現其實是一直忘記要做extra register去每個階段結果的值，因此定義了EXEout、MEMout、WBout這幾個extra register去存每個回合的值，並讓他們每回合結束後互相傳遞下去，如此一來只要偵測到data hazard時便可以去上述變數抓值。

(7)beq的指令在exe階段常常跳錯位置，不知道為何卡了很久

***解決辦法:***

後來發現原來是因為"在stall發生時我有去把回合參數做減一的動作"[變因一]，而"每回合結束後回合參數又會加一"(如同PC+4)[變因二]，所以我在算跳的位置時忘記去減掉已經加過的一，因此才會每次都多跳了一行指令(不論beq有沒有成立)

(8)寫檔常常碰到只寫出最後一次cycle的結果，過程的輸出皆無出現

***解決辦法:***

後來發現其實有出現，只是因為採用的是ios:out模式，他會覆蓋的原本資料，因此當然只會出現最後的，所以改採用append模式寫檔，不覆蓋原資料而是從原資料最後寫入，此問題終於不復存。

**3.工作分配**

主程式碼撰寫：姜博允

主程式碼修訂：姜博允、楊東祐

專案報告撰寫：楊東祐

專案上台報告：陳界亨、王均元

**4.專案報告之心得**

**姜博允 :**

這次專題的最大難題就在於要去實作一個同時執行多指令又要去想辦法處理當中的hazard，而hazard又分為很多種讓原本觀念就不是很清楚的我，在實作一開始去補齊正確的觀念花了一番工夫，雖然是不小也不短時間的折磨，但相對的也得到了莫大的收穫，對pipeline這topic有更深一層的體悟，尤其是在每個階段什麼指令什麼環節處理的若干細節都瞭若指掌了，而且今年新增的可forwarding就不能stall的限制確實讓我沒有可以使用一些小技巧的空間，因此腳踏實地的去完成了這項專題也讓我得到了莫大的成就感，對計算機組織的博大精深更加新領神會到。

**王均元 :**

這次專題要我們實作出一個pipelined的CPU模擬器，理論上是很簡單的概念，但是實際上在做的時候，卻須要考慮到許多的情況，向什麼時候要forwarding，什麼時候要做stall或著什麼時候要兩者一起用等等，許多因素都必須要考慮進去，寫成程式卻是個有些困難的工作，不過這次的專題是四個人一組，遇到自己想不出來的東西時，其他組員可以幫你解答，能夠使得自己更加進步，這也是為什麼要寫專題的原因吧，最後希望自己未來能將自己在這次專題所學道的東西，運用在其它的地方，也感謝老師藉由實作來使我們更加進步。

**楊東祐：**

這次的專題在實作上面臨許多挑戰，一開始的時候我對題目並不是很清楚，直到自己去找資料才有些頭緒，而且我自己預想的模擬器不太相同。在課堂上講述的pipeline的概念我們都很能夠理解，但是如果要實作出一個MIPS模擬器是不一樣的兩件事情。每一個步驟都必須清楚需要做什麼動作，讀入一個指令後要不要做stall或是forwarding。不同的hazard都必須要有相對應的解決方法，並不是先前自己所想的那麼簡單。最後感謝老師給我們這個實作機會，能夠讓我們藉由實作使我們更加了解pipeline的運作。

**陳界亨 :**

最近課程上教的forward跟stall我認為自己雖然有些小地方不清楚不過大致上了解其運作方式，對於相關問題應該可以處理的不錯，然而在做專題的過程中這些小地方出現的各種問題讓人難以理解，這才發現到原來對於這兩個東西我還有許多情況仍模糊不清，剛好趁著這個專題釐清我不少的疑惑點，雖然debug的過程是痛苦的，不過在做完這個專題後對forward跟stall也會有進一步的了解了。

**三、參考資料**

**C++plus：**<http://www.cplusplus.com/>

**pipeline的部分:** <http://ocw.nthu.edu.tw/ocw/index.php?page=course&cid=76>