計算機組織期末專題報告

第九組

成員

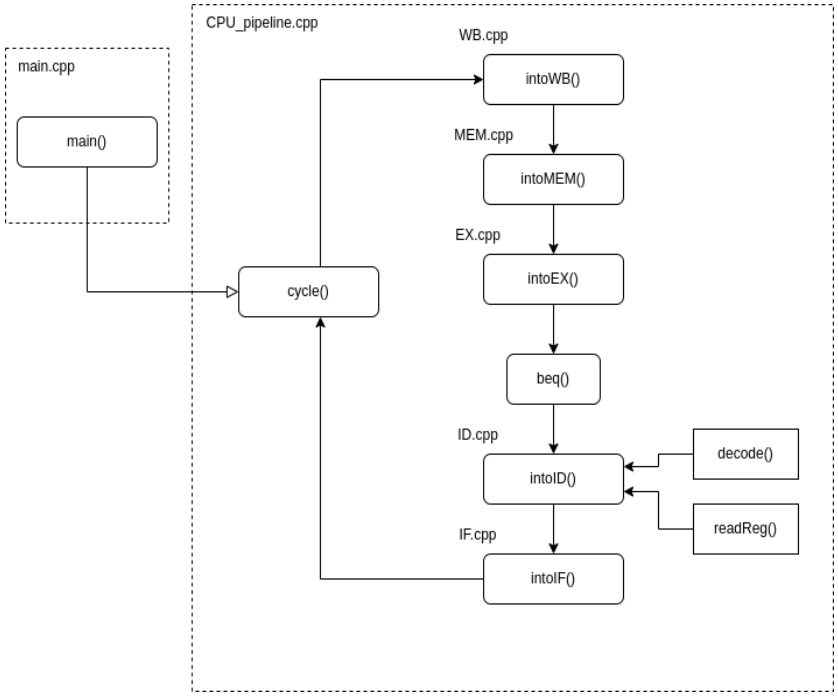
A1095509　李品妤

A1095514　朱祐誼

A1095550　莊郁誼

A1095551　廖怡誠

一、 程式架構



程式主要分成三個部份，main、CPU_pipeline 和五個階段 IF、ID、EX、MEM、WB。

我們將這次的專題實做成一個 Class CPU_pipeline，使用者只需要從 main 將 MIPS 指令傳入 class 中，就可以得到對應的執行結果。

實做的主體在於 cycle 階段執行的內容，我們將 PC 的更動和是否需要執行 Stall 的判斷，此安排是用來解決無法同步執行的問題，接著正式進入 pipeline 中，循環的順序為 WB->MEM->EX->ID->IF，這樣的安排是為了確保每次該階段執行時所拿的值，都是還沒被更新過的值，如果想反過來，就會出現像 WB 要拿 MEM/WB register 的值出來，但此暫存器的值已經被先執行的 MEM 更新，導致 WB 獲取的會是下一個 cycle 要用的值。

在循環的過程中，我們將本要在 ID 中判斷的 beq 改放到 ID 和 EX 之間，這樣設計的原因與前面新舊值影響的問題相同，且因為我們是

將 PC 的更動放在 cycle 中，因此 beq 的判斷也放在同一層會比較好處理，可以透過改變 index 移動目標指令。

不斷循環直到傳入的所有指令都執行完 cycle function 就會結束，並輸出結果。

二、 程式說明

撰寫基礎:

- 在 ID 所使用的值為 IF/ID register 中的值，在 EX 所用的值為 ID/EX register 中的值,以此類推。
- 為了方便處理每個 regiter 都一定會傳入 rs,rt,rd 和 offset，沒有用到的就會填 0。
- stall 或是目前 stage 沒有指令，均會填入 null 做區別。

功能說明:

- Main
  - 程式的進入點。
  - 讀取 MIPS 指令並做字串處理。
- CPU_pipeline
  - 用 index 模擬 PC 的變化。
  - 將記憶體、暫存器與 cycle 數初始化。
  - 讀取的指令進行循環，並計算最後一個指令結束時所花費的 cycle 數。
  - 在一輪 cycle 結束後，判斷 EX hazard 和 MEM hazard 情況，如果有 hazard 的話，會將 EX 的傳入值改為 null，代表不將 ID 的值往下傳，ID 只執行更新 WB 回傳的值，IF 不變，模擬 stall。

- 如果沒有 hazard 的話，正常運行。若在執行 ID 前 opcode 為 beq，就會檢查 beq 條件是否成立。若成立就會改變下個指令的位置，並將 ID 傳入 null，代表原本的 IF 沒有往下傳，而在寫入新位置到 IF。

- IF
  - 將讀入的指令字串切割，例如：讀入 add $1,$2,$3,處理後為 [add,$1,$2,$3,]。

- ID
  - Decode 將讀入指令，根據對應的操作轉換成 signal，例如：讀到指令 lw，轉換成 0101011、讀到指令 add 或是 sub，轉換成 1000010，而指令 rs,rt,rd 或是 offset，則直接用字串處理取得使用，過程不會轉換成 machine code。
  - 利用 rs, rt 取出要使用的暫存器存到 reg1, reg2。
- EX
  - 判斷 opcode 是 add,sub,lw,sw，決定要執行哪種操作，存到 ALUresult。
- MEM
  - 如果 MemRead 為 1，從記憶體對應的位置取出資料，對應的位置為在 EX 計算完的值。
  - 如果 MemWrite 為 1，則將資料寫入記憶體對應的位置，對應的位置為在 EX 計算完的值。
- WB
  - 如果 MemtoReg 為 1，則從記憶體取出的值，且 RegWrite 為 1，更新到暫存器 rt 中。

■ 如果 MemtoReg 為 0，則把從 ALU 中計算出的值，且 RegWrite 為 1，更新到暫存器 rd 中。

三、 遇到問題&解決

1. 指令讀取處理

問題:

正常處理流程，傳入 MIPS 指令後需要先將指令組譯成機械碼，再放入 pipeline 中開始循環，但因為我們在轉換成機械碼後會是用字串儲存，反而在 ID 階段要獲取暫存器值時，會因為要將二進位字串轉回整數，變得更難處理。

解決:

最後，我們放棄原本的方式，不把傳入的指令轉換成二進制，直接將指令做字串切割，分出要獲取的數值，就不需要再做轉換的動作。

2. Data Hazard

問題 1:

在課堂的內容中，只有提及在有 forwarding 的情況下，EX hazard 和 MEM hazard 是如何判斷及處理，但這次的專題是實做沒有 forwarding 的版本，所以當初對於要放在哪個階段和怎麼設計猶豫了很久。

解決:

最後，我們決定參考 forwarding 電路圖的架構，將判斷機制放在 ID 階段，與原本的差異在於在判別後，我們不會提前把值傳回去做更新，只會再出現情況後，將傳入 EX 的值改為 null，模擬 stall。

問題 2:

使用軟體模擬硬體環境最大的痛點是無法模擬同步執行，這導致判別 data hazard 的操作沒辦法一起放在 ID 中，因為程式的流程是從後往前執行，代表在 ID 階段獲取 EX 和 MEM 階段的值，實際上已經是下一個 cycle 的值，會造成判斷錯誤，如果要從前往後執行的話，又會導致前面更新的值蓋過舊的值，要更改架構的話，也只會變得更複雜。

解決:

因此，最後我們決定這個部份就不按照硬體設計，等全部的階段執行完，在一次做判斷，決定下一個 cycle 是否需要 stall。

3. Beq 的判斷

問題:

Beq 原本是在 ID 中做判斷的，但會因為執行先後順序的影響，破壞到指令運行的步驟

解決:

因此，我們把它移到循環的地方做判斷，因為那邊可以直接處理指令運行的順序。

4. lw & sw Offset

在寫的時候，指令 lw, sw 的 offset 我們是直接讀取存入，但取得 register 或 memory 值之前，需要先除以四，位置才會對，因為單位是 word。網路上查了很多資料，有些沒有除，有些有除，所以當時我們花了許多時間討論。

四、 分工

因為這次的專題程式碼為一體成型，所以我們工作分配的方式為輪流寫，在討論完整體架構後，我們將工作分為前半部，架構撰寫包含 pipelined 的五個階段，後半部為 beq 和 data hazard 的判斷及彙整程式碼。會這樣分配的原因是從頭建立架構比較麻煩，所以將較麻煩的 stall 處理拆出來，順便進行程式碼的檢查並彙整。

其餘的報告、Readme、makefile 均為兩人合力討論寫出來的內容。

莊郁誼	程式架構設計、撰寫、寫報告、Readme、makefile
廖怡誠	程式架構設計、撰寫、寫報告、Readme、makefile

五、 心得

莊郁誼：

一開始在寫的時候不知道要從哪裡下手，所以就先把他全部都寫在主程式，之後再分成不同階段的 class。而程式在進入每個階段後可以得到不同的值，而這個就要花很多時間去研究、搞懂其運行的原理，再把他寫成程式，是一項挑戰。但經過這次專題我也更理解什麼時候需要加 stall，還有每個階段的功用，會產生哪些值，什麼時後會影響暫存器、記憶體...，還有 signal 的運作！總覺得應該要把每個階段所有的元件都寫成一個物件來處理，但這樣好像又太大費周章，所以最後沒有這樣處理。在處理傳送到每個階段的時候，我覺得特別酷，因為不像是以我們學到的去讓他照順序處理下來，而是先從後面一個階段往後推，再往前一個階段後，再往後推。

還有在寫的過程中，有用到 git，git 真的相當方便，終於不是在用 LINE 來做版本更新了呵呵呵，而且也更熟悉 git 的指令。還有學到怎麼寫 readme，還有 makefile。

總之寫完，學到了很多，也感到很有成就感，相信這個專題帶給我相當多的知識！

廖怡誠：

我認為最困難的部份是架構設計，因為要用軟體模擬硬體的實際情況有很多的條件限制，像是無法同步處理、模擬元件的大小較難設計成一樣的，且上課教過比較完整的電路架構只有 forwarding 的版本，因此在 data hazard 如何處理的部份，花了許多的心思，最後也只能突發奇想的設計一下。雖然在實做之前已經有好好想過要怎麼設計，但開始做之後又會發現許多的問題，像是 stall 和 beq 的判別，等一輪的 cycle 結束後再一次做判別，才不會出現問題，否則就會有遇到困難中提到的問題發生，在這個環節改了很多次的架構，有幾次的更動反而讓整體的結構都被影響到，越寫越亂，經過一番波折才決定用現在的處理方式，這部份應該是整份專題最有挑戰性的地方，所以在做出來之後覺得很有成就感。

在 Readme 的設計中，為了要讓助教可以快速編譯執行，還嘗試寫了 Makefile，覺得還蠻有趣的，在其他課程上看到其他助教使用，就覺得是一個能夠避免環境問題的好方法，所以這次硬是抽出時間把 Makefile 做出來，成果也相當不錯。