**專案報告**

1. **工作分配:**

**鄭俊安: 設計程式架構、最終的程式碼整理、技術顧問**

**廖習驊: stalling的程式之元件實作、專案報告**

**張家展: forwarding的程式之元件實作、README**

**高士然: stalling 與 forwarding程式中machine.h實作**

1. **製作過程以及遭遇的困難:**

**一開始在設計架構時一直想不到好的解法，想用最純粹的寫法把元件一一實現，但這樣工作量太大了，而且C++不像verilog可以以bit為單位去做存取及調整，資料型別一下設char一下設int會讓整個程式看起來雜亂無章，因此在前中期時，我們嘗試過乾脆以verilog去實現這份專案。**

**事實上針對這份專案，在前中期(元件實作)時verilog比C++友善太多了，能針對單一bit去做處理、連接，加上某些元件在電路實作上是以繞線的方式去做處理的，若用C++來寫只能以左移右移的方式去解決，並不能說是很好的模擬。**

**但在包裝verilog元件的過程中還是遇到了難以解決的問題，set up timing problem，在clock正緣觸發時，元件的input電容可能還沒充滿，導致元件的輸出為上一次輸入產生的結果，是有想過很多方法來解決，改為正緣觸發時做input電容的充電，負緣觸發時去對input做處理，但在register file又會發生問題，原因是必須在前半個cycle做寫入後半cycle做讀取，等於說上述的解法並不對於每個元件皆可行，另外一個解決辦法是多加一個cycle，但是覺得都用verilog寫了，還不能真正的模擬出電路就有點無趣，所以後來又改回用C++寫。**

**既然打算用C++來寫，就打算將全部的資料型態設為int，雖然這樣使用的位元數目跟電路所需的位元數目相差不小，但在整體程式碼的可讀性以及彈性上面都加了不少的分。**

**那有之前寫verilog的經驗，我們可以把一些verilog的想法給套在C++上，像是在verilog裡不會看到 A\_unit.input\_val = B\_unit.output\_val，那我在C++中為了去實現類似的操作，我將各個元件的input都設為一個pointer，去指向前一個元件的output，這樣就不需要用assign的方法去將前一個元件的output傳給後面元件的input。**

**在成員變數的命名上，也是有一些小小的講究，對於input的成員變數，變數名稱會以下畫線** “\_” **做開頭，而output的成員變數則是以下畫線做結尾。**

1. **專案實作心得**

**高士然:**

**在寫完我的pipeline模擬之後，我覺得這是一個非常有趣且富有挑戰性的項目。**

**在開始進行模擬之前，我需要瞭解pipeline的基本概念和運作原理， 在實作過程中，我遇到了一些挑戰。首先，我必須確保我的模擬能夠正確地將指令放入階段並進行處理，並且在每個階段之間正確地傳遞資料。其次，我必須處理延遲和分支轉移的問題，這需要我在每個階段之間正確地傳遞控制信號。**

**總的來說，這個項目讓我學到了許多有關pipeline運作原理和C++程式設計的知識，並且增強了我的架構設計的能力。我希望以後能夠進一步探索和應用這些知識。**

**張家展:**

**首先，我學習到了pipeline是如何讓CPU在執行指令時更有效率的。在pipeline中，每個指令都會在不同的階段被處理，例如取指令、解碼、執行、記憶體存取和寫回。這樣的設計讓CPU可以同時處理多個指令，大大提高了效率。**

**其次，我還學習到了如何使用C++模擬pipeline。在寫程式的過程中，我必須了解每個指令的功能，並使用C++的物件導向程式設計原則來建立各個指令的類別。我也必須考慮如何在不同的階段之間傳遞資料，以及如何處理像分支延遲之類的流程問題。**

**最後，我也學習到了如何使用資料結構儲存指令的資訊，並使用指標操作來改變變數操作的運用。**

**總的來說，模擬pipeline是一個有趣且富有挑戰性的經驗，讓我對pipeline有了更深刻的理解。**

**廖習驊:**

**在寫了一份pipeline模擬的程式之後，我對pipeline的運作有了更透徹的了解。**

**在模擬的過程中，我們需要對每個階段（如IF、ID、EX、MEM、WB）的流程進行模擬，並記錄每個階段的資料，以便在接下來的流程中使用。透過這樣的方式，我們可以看出pipeline在實際執行指令時的流程，並且更能理解pipeline背後的原理。**

**在實作過程中，我們還需要注意分支延遲、數據延遲以及流水線暫停等問題，並適當地設計暫存器以解決這些問題。在處理完這些問題後，我們才能真正執行流水線指令。**

**總的來說，透過模擬pipeline的過程，我對pipeline執行指令的原理有了更深刻的了解，也對如何解決pipeline執行過程中遇到的問題有了更多的瞭解。**

**鄭俊安:**

**這次專題原本打算用verilog來寫，雖然最後失敗了但也讓我更熟悉verilog的語法及細節，像是避免latch、priority mux...等等與電路比較相關的東西。**

**在後來用C++設計架構的同時也把在verilog所用到的想法給實際使用在C++上，算是鑽研出了一種之前沒有想過的架構，在之後設計其他問題時，也可以把這次所學給應用進去。**