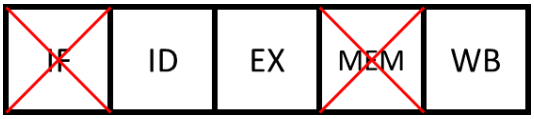
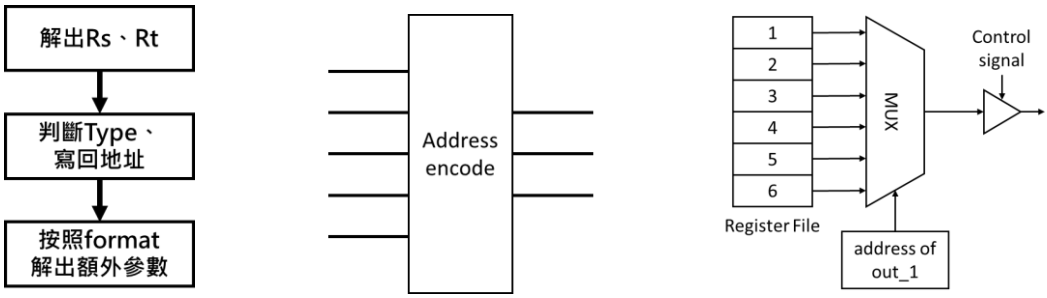
數位電路與系統 HW5 Report

110511233 李承宗

第五次作業要實現一個簡化版的 MIPS CPU，由於這學期我剛好有修計算機組織，因此這次作業的敘述跟目標我覺得還不算難懂。了解目標之後，有多修課的優勢就出來了，我馬上就想到在計組所學的 5-stage pipeline CPU 架構，因此我就先把這個架構拿出來用：



但是，如果真的要像計組課本那樣把 5-stage pipeline CPU 做出來的話肯定得花上不少時間。所以我就先從修改架構下手，發現其實作業並不需要 MEM 的那層，IF 也是因為這次 design 沒有接到 Memory 之類的外部電路，因此可以稍作簡化，直接從 ID 開始即可。

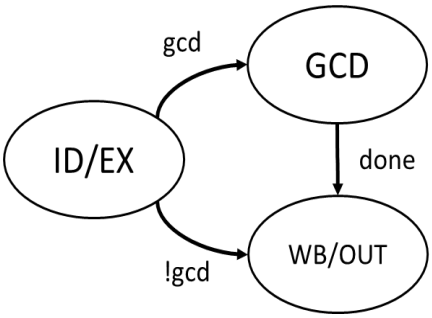


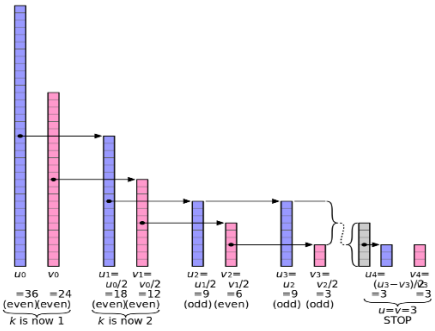
因此接下來只需要把剩下三個 stage 完成即可。ID 的部分由於指令並沒有很多，因此我的解法是直接使用嵌套式的 decoder 來實現：先確定動作是 R-Type 或 I-Type，再將所需要執行的指令以及一些額外參數（*imm*、*funct*、*shamt*等等）給解出來。完整流程是先將共同需要的 Rs、Rt 先解出來並判斷是否為有效指令（雖然 R-Type 指令不需要 Rs，但還是需要拿來判斷是否為有效指令），接下來再用*opcode*來判斷指令的格式以及要寫回的地址是 Rd 還是 Rt，最後將所有參數存起來，以供 EX stage 使用。

接下來兩個 stage 就相對要簡單許多，在 EX stage 要做的事就只需要將結果算出來，最後在 WB 的部分將值寫回*register*，並順便將要求的值輸出至*out*_1,2,3,4即可。

其中我發現有效的地址其實只有 6 個，因此我做了幾個小電路專門來處理地址，分別是：在 ID 的時候將地址*encode*成 0～6 的值（由於我在設計的時候將 instruction fail 的訊號也塞在地址的部分，這樣做可以讓對地址做解析時候順便判斷是否為有效的指令，所以需要七個值來列出全部情況），而且*output_reg*的值也可以透過這個電路來進行壓縮，這樣一來可以節省許多儲存地址的額外開銷，所以我覺得非常值得。而且讀取*register file*的時候部分就只需要輸入1~6來當作地址，一舉兩得。輸出的地方我使用組合邏輯來實現，再加上一個*control signal*，當*out_valid*為 1 並且*instruction_fail*為 0 時才會啟用，並且將*register file*的值輸出到*out*_1,2,3,4。

在實際將 EX stage 做出來的時候，我發現 R-Type 裡面有求最大公因數的指令，因為計算需要的 cycle 並不是固定的，這樣一來肯定會破壞 pipeline 的結構（如果碰到 GCD 還是要持續 pipeline 下去的話就得加很多將 instruction 存起來的儲存空間），而且我發現輸出的波形其實是在*out_valid*輸出 1 的下一個 clock 才會給資料，所以在到這裡的時候我就放棄原先 pipeline 的架構，改成使用狀態機來控制現在 CPU 跑到哪一個 stage。在設計的時候我也簡化了狀態，將本來的 ID/EX 合併成一個 stage，這樣就可以少存很多額外的值，同時因為少了一個 stage，延遲也低了不少。





因為 GCD 並不是一個 cycle 就能解決的事，所以在 ID/EX 結束之後會有兩個 branch，分別是：(1) 如果指令要做的是 GCD，就將值塞給跑計算 GCD 的電路，待運算完成之後再跳回 WB/OUT；(2) 除了 GCD 之外的指令都可以在一個 cycle 完成，因此只要直接轉到 WB/OUT 即可。

解決了所有控制、流程的部分，最後就只剩下如何將 GCD 算出來了。在設計的時候我就有想到用相減的方式肯定很吃虧，取餘數會合出很大的除法器電路（16/16bits）肯定也不值得，因此我就開始上網研究有沒有更適合電路運算的方式。最終，我找到了一個叫做 Binary GCD（Stein's Algorithm）的方法，其實原理非常簡單，就是利用二進制的特性來優化的輾轉相除法，判斷兩個數的 LSB，來做出相對應的動作：(1) 兩個都是 0，那麼這兩個數一定有 2 的公因數，就可以提出來（也就是對兩個數同時除以 2）；(2) 0 跟 1（或 1 跟 0），那麼這兩個數一定沒有 2 的公因數，就可以將 LSB 為 0 的數字往右 shift（除以 2）；(3) 兩個數都是 1，那麼這兩個數可以直接相減，並且減出來的結果又可以做 shift（除以 2）。做完這些動作之後就可以進入下一次運算，直到兩個值相等，或是其中一個數為 0，就可以得到我們要的結果。

藉由這些敘述可以發現這套演算法非常適合拿來在電路上實現，而且我在測試的時候發現：這套電路的 critical path 只會經過 mux 跟減法器，所以其實延遲很低。接下來我就有了大膽的想法，也就是在一個 cycle 內做三四次的迭代。實際出來結果也很理想，電路執行的時間下降了許多。至此整個電路設計完畢。

由於這次 HW5 剛好跟我的比賽、期中考試，以及系上活動的事情夾雜在一起，所以這次作業我沒有辦法花時間做額外的優化，都是在設計的時候想到怎麼做就直接寫下去了，一定還有可以改進的地方是我沒想到的，有點可惜。