**計算機結構 Project1**

資工三 B04902009 蕭千惠

資工三 B04902027 陳 昇

資工三 B04611015 陳佳佑

1. **Pipeline Structure**

將原先single cycle的CPU module加入了IFID、IDEX、EXMEM、MEMWB這4個module，使得程序可分為5個階段進行。同時加入了ForwardUnit來避免不必要的stall，以提升整體的運算效率，並且額外設計了HazardDetection，以實現lw指令所造成data hazard時，無法避免的nop。另外，在面對beq以及j指令所產生的control hazard，也提供了flush的機制來讓我們能夠清除原來錯誤的指令。

1. **Modules Implement**
   1. **ALU**

依照ALU\_Control所給的輸入來決定該進行何種運算，不同的參數分別代表著OR、AND、加法、減法、乘法等不同的運算模式。

* 1. **ALU\_Control**

依照ALUOp以及不同的funtion來分別決定對應的ALUCtrl，並將此結果傳至ALU作為運算根據。

* 1. **Adder**

將兩輸入相加後輸出。

* 1. **Control**

根據不同指令的Op code來分別決定ALUOp、RegDst、ALUSrc、MemtoReg、RegWrite、MemWrite、MemRead、Branch、Jump等參數。Branch送入Flush以及MUX\_Add做運算，Jump送入Flush以及Mux\_Jump做運算，其餘的參數則送入MUX\_Control做處理。

* 1. **DataMemory**

建構一個32\*8 bytes的記憶體，若MemWrite為1，則將資料寫進記憶體；若MemRead為1，則將資料讀出32 bits作為輸出。

* 1. **EXMEM**

將Mem stage要用的參數傳至對應的結構中。

* 1. **Flush**

分別考慮兩種情形，會啟動flush機制。一為j指令，若Control傳入的Jump為1時會導致flush。二為beq指令，若RS和RD檔案相同，並且Control傳入的Branch為1時也會導致flush。

* 1. **ForwardUnit**

分別考慮兩種情況，會啟動forward機制。一為EX hazard，若RegWrite為1，且EXMEM的RD和IDEX的RS或RT相同時，則代表相鄰兩指令之間需要forward。若狀況一並未發生，則考慮狀況二，MEM hazard。若RegWrite為1，且MEMWB的RD和IDEX的RS或RT相同時，則代表此兩指令之間需要forward。並將輸出傳至MUXforward\_1和MUXforward\_2做運算。

* 1. **HazardDetection**

若MemRead為1，則代表我們要進行lw指令。此時倘若IDEX的RT和IFID的RT或RS相同，則代表著我們必須要stall一回合，並且通知IDEX、PC、MUX\_Control。

* 1. **IDEX**

將MUX\_Control、Registers、Sign\_Extend傳入的資料傳出至EXMEM、MUXforward\_1、MUXforward\_2、ForwardUnit、MUX32、MUX5。

* 1. **IFID**

若flush或hazard參數為1，則將輸出都設為0，反之則原封不動的輸出pc及inst。

* 1. **Instruction\_Memory**

由指令的地址取出指令。

* 1. **MEMWB**

將WB stage要用的參數傳至對應的結構中。

* 1. **MUX32**

若select參數為1，則設定輸出為輸入2；若為0，則設為輸入1。

* 1. **MUX5**

若select參數為1，則設定輸出為輸入2；若為0，則設為輸入1。

* 1. **MUXForward**

根據ForwardUnit所傳入的參數，決定是否要forward在EXMEM或MEMWB的資料作為輸出。

* 1. **MUX\_Add**

若Zero和從Control傳入的Branch同時為1時，代表指令的Op code顯示為beq，同時RS和RD相同，滿足beq的條件，此時則將輸出設為跳轉的目標地址；若否，則將輸出設為PC+4。

* 1. **MUX\_Control**

若發生hazard，則將所有signal都設為0；若沒有，則將所有signal合成EX、M、W傳出。

* 1. **MUX\_Jump**

若從Control傳入的Jump為1時，代表發生j指令，則將輸出設為目標地址；若否，則將輸出設為MUX\_Add的運算結果。

* 1. **MUX\_Write**

若MemtoReg為1，則代表指令為lw，需要將地址傳回Register來寫入資料。

* 1. **PC**

hazard\_i會判定是否發生hazard，若發生時則將下個階段的PC改為0來達成stall；若未發生則照常使用MUX\_Jump所提供的下一個PC地址。

* 1. **Register**

若RegWrite為1，則代表Register要將RD的資料寫入register當中。

* 1. **ShiftLeft26**

將輸出設為輸入擴大位數至28位元，低位元補0。

* 1. **ShiftLeft32**

將輸出設為輸入向左做2個位元的logical shift。

* 1. **Sign\_Extend**

以input的最高位來決定擴大位數至32位元的細節，若最高位為1則在前面補齊16個1；反之，則補齊16個0。

1. **Problems & Solutions**
   1. **Equal wrong location**

原先在ALU內就比較兩個輸入是否相等，並輸出對應的Zero參數，此舉會導致數值產生的時機並不正確而發生錯誤。將運算的部分改在CPU內操作，讓Zero在需要使用時才進行運算，就能改正這個問題。

* 1. **MEMWB wrong clock edge**

在MEMWB中也要放入clk\_i，才能讓數值有順著pipeline的脈絡更新，確保整體的正確性。

* 1. **Testbench clock edge**

原先single cycle的testbench直接放入pipeline時會發生錯誤，需要在clk\_i的negedge時更新，才能符合我們的pipeline設計。

* 1. **ShiftLeft32、MUX\_Add、MUX\_JUMP bug**

許多細部的接線看似十分簡單，但是不夠熟稔的編程總會在程式的字裡行間埋下大錯而難以察覺，用了GTKwave後可以快速釐清問題所在，是每個寫程式的人都該要學會使用的好工具，本組全體同學在此感謝資工系廖教授世偉無私的分享此軟件，能夠在死線前安然做完報告都實質歸功於廖教授，特以此篇幅向我們的永遠的導師、永遠的舵手致敬。

1. **Members & Team Work**

Single Cycle: 陳昇、蕭千惠

Multiple Cycle: 蕭千惠、陳佳佑

Outline: 陳昇

Debug: 蕭千惠、陳佳佑

Report: 陳昇