計算機組織期末專題報告

MIPS Pipeline

Stall & Forwarding

第四組

A1095502沈品豪

A1095517黃泓茗

A1095521鄭惠心

A1095557陳柏安

1. 專案架構

我們使用 c++ 模擬出MIPS pipeline，並透過 stall 或 forwarding 解決 pipeline 時會發生的所有 hazard 問題，只需將所有MIPS組合語言指令寫成一個txt檔至指定資料夾，執行Stall.cpp或Forwarding.cpp便會讀取該指令檔案，編譯成機器碼，並執行完pipeline輸出執行結果txt檔。

1. 輸入、輸出檔案及資料夾說明
2. Stall資料夾 : 存放stall所需所有input和輸出結果
3. Forwarding資料夾 : 存放forwarding所需所有input和輸出結果
4. asm.txt : MIPS組合語言
5. input.txt : 將組合語言進行編譯，所產生的instructions
6. memory\_result.txt : 程序結束後，所有記憶體狀態
7. output.txt : 按照ppt中的呈現輸出結果方式
8. register\_result.txt : 程序結束後，所有暫存器狀態
9. state.txt : 每個cycle中，所有stages的詳細資訊
10. 程式功能說明

主要分成四部分:

**1、 Pipeline Main Function:**

(1) Stall.cpp

* 模擬具有stall功能的pipline cpu
* 與Forwarding.cpp共用所有header files

(2) Forwarding.cpp

* 模擬具有forwarding功能的pipline cpu
* 與Stall.cpp共用所有header files

**2、 CPU基本零件**

(1) Instruction\_Memory.h

* 指令以 32 bits (word) 為單位，instruction memory 以 8 bits (byte) 為單位，因此抓取下一個指令時需 PC = PC + 4
* 初始化時，針對asm.txt進行編碼
* 讀取指令並對指令進行字串處理(比如空格和逗號)
* 根據指令形成instruction format
* 最後加上32個bits的 1，代表halt指令，程序結束之位置

(2) Register\_File.h

* 讀取或寫入特定暫存器
* 輸出所有暫存器的狀態，寫檔至 register\_result.txt

(3) Data\_Memory.h

* data memory 以 byte 為單位，資料以 word 為單位，所以在讀取/寫入資料的時候都必須一次處理 4 bytes
* 讀取或寫入記憶體位置之資料
* 輸出所有記憶體的狀態，寫檔至 memory\_result.txt

**3、 Stages**

(1) IF.h

* Instruction fetch
* 儲存當前狀態指令名稱以及是否執行

(2) ID.h

* Instructioin decode & Register read
* 儲存當前狀態指令名稱、是否發生stall以及是否執行

(3) EX.h

* Execute
* 儲存當前狀態指令名稱以及是否執行
* 儲存instruction field, all input, Opcode, control sign

(4) MEM.h

* Memory write
* 儲存當前狀態指令名稱以及是否執行
* 儲存instruction field, memory input, control sign

(5) WB.h

* Write back
* 儲存當前狀態指令名稱以及是否執行
* 儲存instruction field, register input, control sign

**4. State**

(1) State.h

* data member 包含 class IF, ID, EX, MEM, WB

四、程式碼流程

* **同一cycle之Stage流程:**

[ WB Stage -> MEM Stage -> EX Stage -> ID Stage -> IF Stage ]

為防止當前state之後續Stage的變數被前面Stage的操作所影響、更新，因此倒置

* **Stall & Forwarding 流程**

//紀錄當前狀態有哪些stages須執行，以及每一個stage的資料

State current\_state;

//下一個要進入的state

State next\_state;

while(1) {

/\*

執行需執行的stages

透過執行結果與flags，來更新 next\_state 的內容

\*/

/\*-------------------WB stage-------------------\*/

更新Register File

/\*-------------------MEM stage------------------\*/

依指令更新Write\_data及Data\_memory:

-> lw: load data(Write\_data = DM.read(ALU\_result)

-> sw: store data

-> R-type: Write\_data = ALU result

/\*-------------------EX stage-------------------\*/

▲**Forwarding**

EX Hazard

MEM Hazard

▲

依指令更新ALU\_result:

-> add: Read\_data1 + Read\_data2

-> sub: Read\_data1 - Read\_data2

-> lw、sw: Read\_data1 + sign-extended(address)

/\*-------------------ID stage-------------------\*/

Data Hazard

Control Hazard

instruction decode

control sign

/\*-------------------IF stage-------------------\*/

透過IF的PC 讀取instruction memory(IM)的指令:

-> instruction = IM.read(PC)

PC = PC + 4

/\*----------------------------------------------\*/

當所有Stages都不執行時，即為結束 break

儲存每個cycle結束後的狀態

進入下一cycle，並更新當前狀態:

-> cycle = cycle + 1;

-> current\_state = next\_state;

}

五、問題與解決方法

* **有states的程式碼，debug具有一定難度**
* solution 1 : 將每個cycle中，所有stages的訊息詳細印出來，藉此判斷在哪個出現問題
* solution 2 : 設立多個中斷點，以利看出是哪一步出現問題
* **如何用github協作專題，經常發生嚴重merge conflict**
* solution 1 : 隨時更新當前進度，不要太久沒merge，若檔案有新增、刪除、修改皆須注意
* solution 2 : 將源代碼pull到local，與自己的code merge，將conflict解決後再重新發送pull request
* **思考整個程式基礎框架，需哪些標頭檔及其內容**
* solution 1 : 依上課投影片介紹，規劃每個stage的input與output，及control sign於各階段的使用情形，並且將IM、RF、DM的功能實做出來
* solution 2 : 在程式實作中，依編寫情形更新標頭檔內容，逐步完善
* **依不同指令將其轉成machine code，並用其來完成後續所有過程，在過程中判斷machine code與將其進行運算很繁瑣**
* solution 1 : 上網查詢指令之opcode與funct
* solution 2 : 須明確了解所有指令instruction field功能及差別
* solution 3 : 因錯誤通常都不明顯，花費大量時間debug
* **Stall與Forwarding的判斷**
* solution 1 : 查閱ppt與網路資料
* solution 2 : 仔細思考所有條件並瘋狂debug

六、心得

**沈品豪**

這次的專題我被分配到的是forwarding，但我只寫了一點點，我應該好好反省!!

**黃泓茗**

這次專題從0開始使用c++模擬pipeline cpu的運作，雖然過程十分折磨，但親手做出來很有成就感，還可以補足上課一知半解的部分，讓我更深入了解pipeline運作。

**鄭惠心**

這次的期末專題讓我更加了解pipeline的操作，因為在寫code之前必須非常了解整個過程，而且一開始根本不知道該如何下手，也沒有分.cpp和.h檔，把它都寫在同一個專案裡，但這樣實在是太雜亂了，所以我們後來有分成.cpp和.h檔。

同時這學期才開始接觸git的使用，剛好這一次可以使用到於是更加了解怎麼使用，覺得很好玩!!

**陳柏安**

這次的期末專題是一個非常不錯的挑戰，很高興最後能完成Stall及加分的Forwarding，雖然整個過程從構想、研究ppt、實作、debug花費了非常多的時間，但大大的加深了我對Pipeline所有指令、線路及Hazard的概念，獲益良多。

七、工作分配

**A1095502沈品豪**

Forwarding、Word

**A1095517黃泓茗**

Stall、Forwarding、README、Forwarding範例答案計算

**A1095521鄭惠心**

Stall、Word、Forwarding範例答案計算

**A1095557陳柏安**

標頭檔、Stall、Forwarding、README