Computer Organization Lab5

0816146 韋詠祥

Architecture diagrams

以 Lab 4 的程式碼為基礎，加入 Forwarding unit 模組，針對各種會發生 Data hazard 的情況，嘗試在資料產生出來時立即使用。在 load-use 等無法順利修復的情況，則利用 Hazard detection unit 模組產生 `Stall` 訊號，讓程式得以產生正確執行結果。

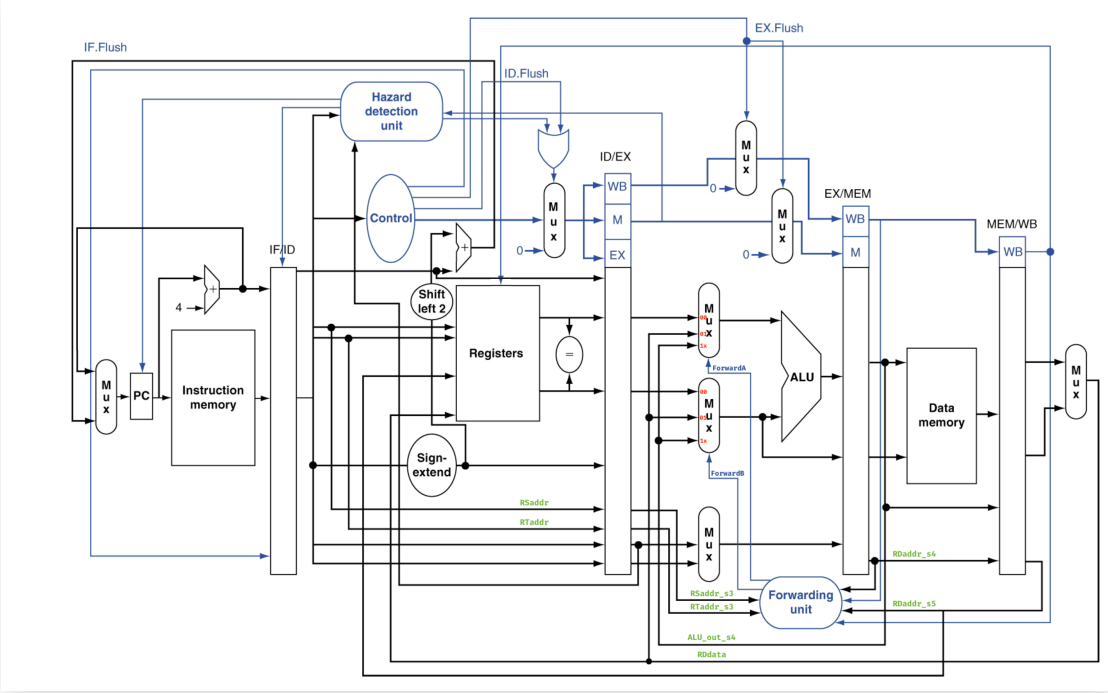

Fig 1. 架構圖，修改自第四章講義

Hardware Module Analysis

這次的 Pipe_CPU 本體從上次的 230 行增加到 360 行，主要是條件分支相關的多工器，以及 Stall 時需要的控制參數。

這次的 Forwarding unit 未完全依照講義的做法，在遇到 double data hazard 時 ForwardA 或 ForwardB 會輸出為 11 而非 10，如此可以大幅簡化程式碼的行為。

```
// I/O ports
input [4:0] RSaddr_i, RTaddr_i, RDaddr_s4_i, RDaddr_s5_i;
input RegWrite_s4_i, RegWrite_s5_i;
output wire [1:0] ForwardA_o, ForwardB_o;

// EX/MEM is s4, fwd[0]
// MEM/WB is s5, fwd[1]

// Main function
assign ForwardA_o = {RegWrite_s4_i & (|RDaddr_s4_i) & (RDaddr_s4_i == RSaddr_i),
                     RegWrite_s5_i & (|RDaddr_s5_i) & (RDaddr_s5_i == RSaddr_i)};
assign ForwardB_o = {RegWrite_s4_i & (|RDaddr_s4_i) & (RDaddr_s4_i == RTaddr_i),
                     RegWrite_s5_i & (|RDaddr_s5_i) & (RDaddr_s5_i == RTaddr_i)};
```

Fig 2. Forwarding unit 程式碼片段

而 Hazard detection unit 則依照講義上提到的 load-use hazard 情境設計，並加入對條件分支的支援。

Problems You Met and Solutions

這次 Lab 5 時間碰上期末考，到最後幾天才有空開始做，精神狀況也不是很好，所以有寫出一些不該出現的 bug，找很久才發現。

```
commit 67e395b1fd9a100707a92d97023f77c48b914cc9
Author: Sean Wei <me@sean.taipei>
Date:   Mon Jun 21 22:25:13 2021 +0800

    Add hazard detection unit

modified: Lab5-Hazard-Detection/Pipe_Reg.v

@ Lab5-Hazard-Detection/Pipe_Reg.v:6 @
always @(posedge clk_i) begin
    if (~rst_i)
        data_o ⩽ 0;
    else if (keep_i)
        data_o ⩽ keep_i;
    else
        data_o ⩽ data_i;
end
```

Fig 3. 錯誤的 Pipe_Reg 程式碼

```
commit 39a1db8f78b04087ae44a1eeae3a052b16e629e2
Author: Sean Wei <me@sean.taipei>
Date:   Tue Jun 22 03:33:27 2021 +0800

    Add stall for conditional branch

modified: Lab5-Hazard-Detection/Pipe_Reg.v

@ Lab5-Hazard-Detection/Pipe_Reg.v:20 @ output reg [size-1:0] data_o;
always @(posedge clk_i) begin
    if (~rst_i)
        data_o ⩽ 0;
    else if (keep_i)
        data_o ⩽ keep_i;
        data_o ⩽ data_i;
    else
        data_o ⩽ data_i;
end
```

Fig 4. 錯誤的 Pipe_Reg 程式碼

```
commit 0fd4dc043e2760d59be5411c223e4d73489eac62
Author: Sean Wei <me@sean.taipei>
Date:   Tue Jun 22 16:51:48 2021 +0800

    Add 2 stall for lw beq

modified: Lab5-Hazard-Detection/Pipe_Reg.v

@ Lab5-Hazard-Detection/Pipe_Reg.v:20 @ output reg [size-1:0] data_o;
always @(posedge clk_i) begin
    if (~rst_i)
        data_o ⩽ 0;
    else if (keep_i)
        data_o ⩽ data_i;
        data_o ⩽ data_o;
    else
        data_o ⩽ data_i;
end
```

Fig 5. 修正成功的 Pipe_Reg 程式碼

如圖三、圖四、圖五記錄，在 Stall 時 `Pipe_Reg` 的行為應該是維持原始數值，但最開始寫成 Stall 時設為 1，後來改成取 `data`，但未注意到 `data_i` 及 `data_o` 差異，最後才改為正確的程式。

Result

正確實作完 `Forwarding` 及 `Hazard` 兩個模組後，成功讓測試資料的輸出與作業規格上的答案相符。

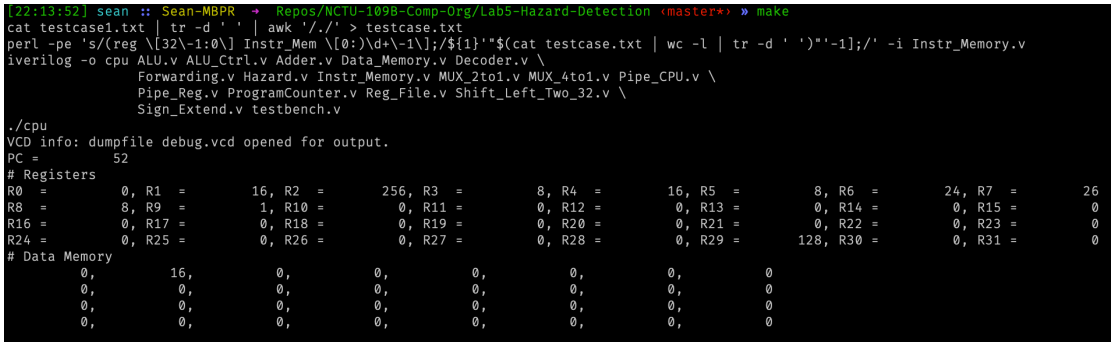

Fig 6. 實驗結果

Summary

這次 Lab 5 分基礎部分及進階部分，在搞清楚概念後，基礎部分很快就能做完了，不過進階部分要考慮的狀況比較多，在加入 Pipeline 後複雜的架構也讓 debug 困難許多，又因為自己沒產出合適的測試資料，整體上在進階部分花費不少時間，也無法確定是否有正確實作。

原本還希望把 `j` / `jal` 也一起實作，就能拿 Lab 3 的費氏數列程式碼來測試了，不過遇到不明的 bug 遲遲無法解決，最後只好作罷。

對於一些在實作中領悟的觀念，讓我很希望在期末考前就能花時間完成這次 Lab 作業。對很多題目都有一定的幫助。