Computer Organization Lab4

0816146 韋詠祥

Architecture diagrams

以 Lab 3 的程式碼為基礎，移除 `j/jal/jr` 等指令的支援後，切分為不同的處理階段，並在各階段中間加入暫存器，從 `IF`、`ID`、`EX`、`MEM`、`WB` 分別編號為 `s1` 到 `s5`

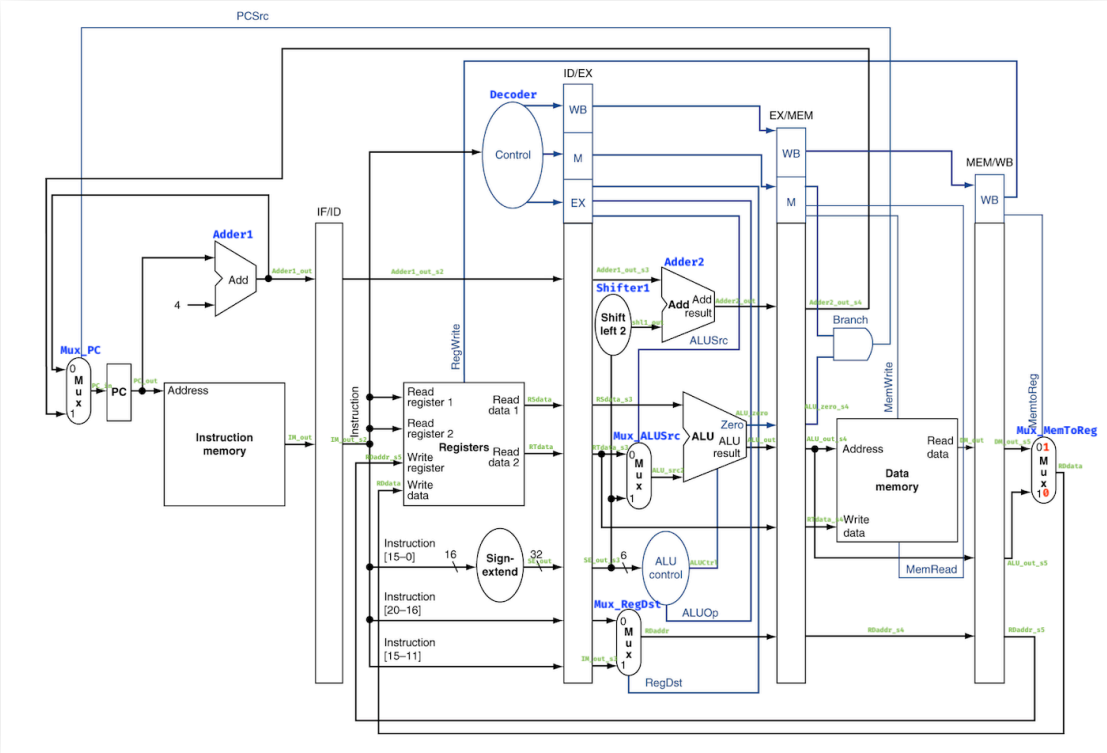

Fig 1. 架構圖，修改自第四章講義

圖 1 中藍色字為模組名稱、綠色小字為 `Pipe_CPU` 主模組中各條連接線的名稱，模組輸出不加後綴，經過暫存器後加上後綴如 `IM_out_s2`

Hardware Module Analysis

這次的 `Pipe_CPU` 本體從上次的 160 行增加到 230 行，少了幾個跟條件分支相關的多工器，新加入的 `Pipe_Reg` 多了 80 行

唯一與作業規格附圖不同的是 `Mux_MemToReg` 模組兩個輸入的順序，這邊將兩個輸入來源交換，以符合上次 Lab 3 使用的順序

Problems You Met and Solutions

在處理測資 1 時，原本的設計會讓第 4 個指令碼 `sw r1, 4(r0)` 吃不到第 1 個指令碼 `addi r1, r0, 3` 所儲存的 `r1` 值，經過分析後發現是 Reg_File 模組當時設計有問題，如果透過 `RDaddr/RDdata` 儲存資料，要在下一個時脈才能從 `RS/RT` 正常讀取，後來加上判斷 `RSaddr/RTaddr` 是否與 `RDaddr` 相同就解決了

處理測資 2 時，如同作業規格所述會遇到 data hazard，嘗試重新排序成圖 2 所示順序後，成功讓每個值寫入暫存器、從暫存器讀取中間都隔至少 2 個指令碼，因此不需要插入空指令就能成功運作，得到正確的輸出成果

```
 1 _00:  addi  r1, r0, 16   # r1 = 16
 2 _04:  addi  r3, r0, 8    # r3 = 8
 3 _08:  addi  r9, r0, 100  # r9 = 100
 4 _0C:  sw    r1, 4(r0)    # A[1] = r1 = 16
 5 _10:  addi  r2, r1, 4    # r2 = r1 + 4 = 20
 6 _14:  lw    r4, 4(r0)    # r4 = A[1] = 16
 7 _18:  addi  r7, r1, 10   # r7 = r1 + 10 = 26
 8 _1C:  add   r6, r3, r1   # r6 = r3 - r1 = -8
 9 _20:  sub   r5, r4, r3   # r5 = r4 - r3 = 8
10 _24:  and   r8, r7, r3   # r8 = r7 & r3 = 8
```

Fig 2. 測資 2 修改結果

這次作業規格說要支援 11 種指令碼，但 `slti` 及 `mult` 不在這次提供的測資中，我相信他是好的。

Result

實作完 Pipe_CPU 並解決測資 2 的 data hazard 狀況後，成功讓兩筆測試資料的輸出都與作業規格上的答案相符

```
[17:49:26] sean :: Sean-MBPR  ➜  Repos/NCTU-109B-Comp-Org/Lab4-Pipelined-CPU (master*) » make
cat testcase1.txt | tr -d ' ' | awk '/./' > testcase.txt
perl -pe 's/(reg \[32\-1:0\] Instr_Mem \[0:)\d+\-1\];/${1}"$(cat testcase.txt | wc -l | tr -d ' ')"-1];/' -i Instr_Memory.v
iverilog -o cpu ALU.v ALU_Ctrl.v Adder.v Data_Memory.v Decoder.v \
                Instr_Memory.v MUX_2to1.v MUX_4to1.v Pipe_CPU.v Pipe_Reg.v \
                ProgramCounter.v Reg_File.v Shift_Left_Two_32.v Sign_Extend.v \
                testbench.v
./cpu
VCD info: dumpfile debug.vcd opened for output.
PC =         56
# Registers
R0  =         0, R1  =         3, R2  =         4, R3  =         1, R4  =         6, R5  =         2, R6  =         7, R7  =         1
R8  =         1, R9  =         0, R10 =         3, R11 =         0, R12 =         0, R13 =         0, R14 =         0, R15 =         0
R16 =         0, R17 =         0, R18 =         0, R19 =         0, R20 =         0, R21 =         0, R22 =         0, R23 =         0
R24 =         0, R25 =         0, R26 =         0, R27 =         0, R28 =         0, R29 =         0, R30 =       128, R31 =         0
# Data Memory
         0,          3,          0,          0,          0,          0,          0,          0
         0,          0,          0,          0,          0,          0,          0,          0
         0,          0,          0,          0,          0,          0,          0,          0
         0,          0,          0,          0,          0,          0,          0,          0

cat testcase2.txt | tr -d ' ' | awk '/./' > testcase.txt
perl -pe 's/(reg \[32\-1:0\] Instr_Mem \[0:)\d+\-1\];/${1}"$(cat testcase.txt | wc -l | tr -d ' ')"-1];/' -i Instr_Memory.v
iverilog -o cpu ALU.v ALU_Ctrl.v Adder.v Data_Memory.v Decoder.v \
                Instr_Memory.v MUX_2to1.v MUX_4to1.v Pipe_CPU.v Pipe_Reg.v \
                ProgramCounter.v Reg_File.v Shift_Left_Two_32.v Sign_Extend.v \
                testbench.v
./cpu
VCD info: dumpfile debug.vcd opened for output.
PC =         56
# Registers
R0  =         0, R1  =        16, R2  =        20, R3  =         8, R4  =        16, R5  =         8, R6  =        24, R7  =        26
R8  =         8, R9  =       100, R10 =         0, R11 =         0, R12 =         0, R13 =         0, R14 =         0, R15 =         0
R16 =         0, R17 =         0, R18 =         0, R19 =         0, R20 =         0, R21 =         0, R22 =         0, R23 =         0
R24 =         0, R25 =         0, R26 =         0, R27 =         0, R28 =         0, R29 =       128, R30 =         0, R31 =         0
# Data Memory
         0,         16,          0,          0,          0,          0,          0,          0
         0,          0,          0,          0,          0,          0,          0,          0
         0,          0,          0,          0,          0,          0,          0,          0
         0,          0,          0,          0,          0,          0,          0,          0
```

Fig 3. 實驗結果

從波形圖來看，測資 1 及測資 2 各時刻的輸出值也與人工計算的結果相同

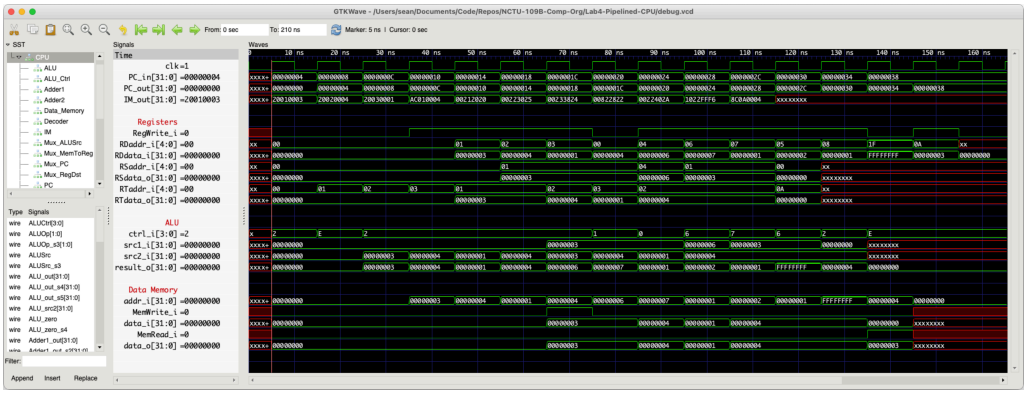

Fig 4. 測資 1 波形圖

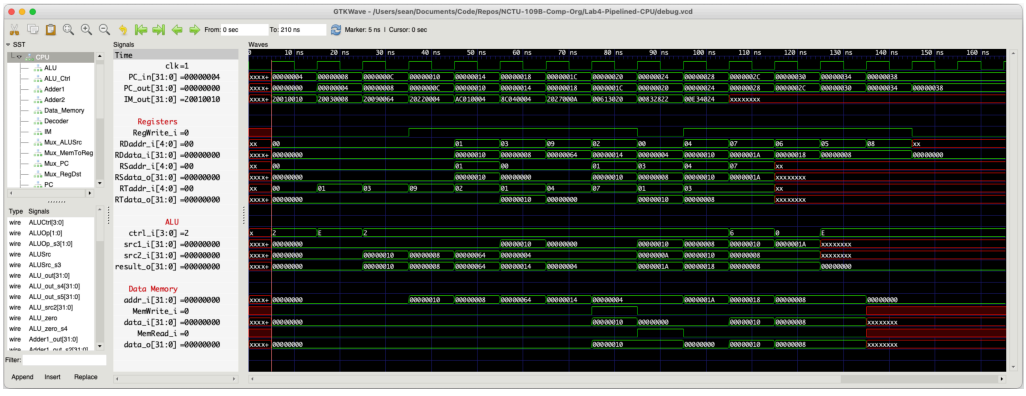

Fig 5. 測資 2 波形圖

Summary

這次雖然把 Pipe_CPU 分成 5 個處理階段，但需要支援的指令比起上次 Lab 3 少了很多，因此 Decoder 出現 BranchType/Jump 兩個沒用到的控制訊號，目前做法是 Decoder 照樣輸出，但傳到 Pipe_CPU 的連接線後不再往下傳遞，讓下次重新實作時不需要再次更動 Decoder 內的邏輯

實際開始寫這份作業以前，看到架構圖的中間 4 條 Pipe_Reg 突然不知道該從何下手，參考範例程式碼後，認為連接線的命名規則很合理，有助於程式碼可讀性，因此仿照同一套規則，在架構圖上標註每一條連接線的變數名稱後，看著修改個模組的輸入輸出變數，實作完後覺得這次 Lab 4 Pipelined CPU 反而比上次 Lab 3 Signle Cycle CPU 簡單很多

Reference

MIPS Converter: https://www.eg.bucknell.edu/~csci320/mips_web/

MIPS CPU implemented in Verilog: https://github.com/jmahler/mips-cpu/