Computer Organization HW4

110550088 李杰穎

June 15, 2023

1 The input fields of each pipeline register

值得注意的是，每條 wire 後面都會標明這條 wire 是屬於哪個 stage，讓 wire 不會誤用。

1.1 IF/ID

`PC_add1_IF, instr, (instr == 0 ? 1'b0 : 1'b1)`

其中：

- `PC_add1_IF` 為 PC+4
- `instr` 為 instruction
- `(instr == 0 ? 1'b0 : 1'b1)` 為後文會提到的 `enable` 的 wire

IF/ID Register 大小為 32+32+1=65

1.2 ID/EX

`RegWrite_ID, MemtoReg_ID, Branch_ID, MemRead_ID, MemWrite_ID, RegDst_ID, ALUOp_ID, ALUSrc_ID, PC_add1_ID, ReadData1_ID, ReadData2_ID, sign_ID, instr_ID[20:16], instr_ID[15:11], enable_ID`

其中:

- `PC_add1_ID` 為 PC+4
- `ReadData1_ID, ReadData2_ID` 為 register file 讀到的 `rs, rt` 資料
- `sign_ID` 為經過 sign extended 的 offset
- `RegDst_ID, ALUOp_ID, ALUSrc_ID` 為 EX stage 會用到的三個 control signal
- `Branch_ID, MemRead_ID, MemWrite_ID` 為 MEM stage 會用到的三個 control signal
- `RegWrite_ID, MemtoReg_ID` 為 WB stage 會用到的兩個 control signal
- `instr_ID[20:16]` 為 `rt`
- `instr_ID[15:11]` 為 `rd`
- `enable_ID` 為後文會提到的 `enable` 的 wire

ID/EX Register 大小為 1+2+1+1+1+2+3+1+32+32+32+32+5+5+1=151

1.3 EX/MEM

`WB_EX, MEM_EX, PC_t_EX, zero_EX, ALUResult_EX, ReadData2_EX, WriteReg_addr_EX,`

其中:

- `WB_EX` 為 WB stage 會用到的 control signal
- `MEM_EX` 為 MEM stage 會用到的 control signal
- `PC_t_EX` 為 branching 的 target address
- `zero_EX` 為 ALU 輸出 zero 的 wire
- `ALUResult_EX` 為 ALU 的計算結果
- `ReadData2_EX` 為 `rt` 的資料
- `WriteReg_addr_EX` 為 WB stage 寫入的 register address
- `enable_EX` 為後文會提到的 `enable` 的 wire

EX/MEM Register 大小為 3+3+32+1+32+32+5+1=109

1.4 MEM/WB

`WB_MEM, DM_ReadData_MEM, ALUResult_MEM, WriteReg_addr_MEM, enable_MEM`

- `WB_MEM` 為 WB stage 會用到的 control signal
- `DM_ReadData_MEM` 為 data memory 讀取到的資料
- `ALUResult_MEM` 為 ALU 的計算結果
- `WriteReg_addr_MEM` 為 WB stage 寫入的 register address
- `enable_MEM` 為後文會提到的 `enable` 的 wire

MEM/WB Register 大小為 3+32+32+5+1=73

2 Explain your control signals in sixth cycle

我們首先看到 Figure 1，可以發現在第六個 cycle 中，進到 ID stage 的 instruction 為 `0x00202816` 對應到 MIPS instruction 為 `or r5, r1, r0`。故 `RegWrite` 為 `1`，代表在 WB stage 時會將資料重新寫回 register 中、`ALU_Op` 為 `010` 代表為 R-type 的 instruction、`ALUSrc` 為 `0`，代表 ALU 的第二個 input 為 `rt`、`RegDst` 為 `1` 代表寫入到 `rd`、`Jump, Branch, BranchType, MemWrite, MemRead` 皆為 `0`，代表皆不會用到這些功能。最後 `MemtoReg` 為 `1`，代表直接將 ALU 的輸出寫入到 `rd` 中。

再來看到 Figure 2，在第六個 cycle 中，進到 ID stage 的 instruction 為 `0x8c020004` 對應到 MIPS instruction 為 `sw r2, 4(r0)`。故 `RegWrite` 為 `0`，代表在 WB stage 時不會將資料重新寫回 register 中、`ALU_Op` 為 `000` 代表為 `lw`，此時 ALU 可以視加法器，將 `rs` 儲存的位址加上 offset、`ALUSrc` 為 `1`，代表 ALU 的第二個 input 為 signed extend offset、、`RegDst, MemtoReg` 可以視為 don't care，因為我們不會將資料寫回到 register file、`MemWrite` 則是設為 `1`，因為我們要將 `rs` 的資料透過 Data Memory 寫到 `rt+4*4` 的 address 中，最後 `Jump, Branch, BranchType, MemRead` 皆為 `0`，代表皆不會用到這些功能。

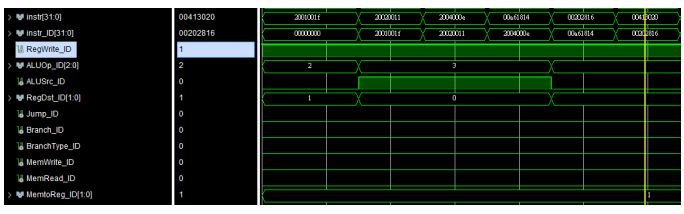

Figure 1: CO_P5_test_data1 之波形圖，圖中黃色線所在的 cycle 為第六個 cycle

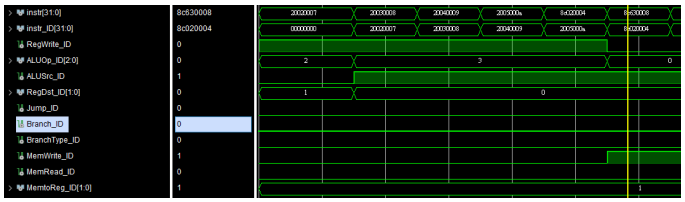

Figure 2: CO_P5_test_data2 之波形圖，圖中黃色線所在的 cycle 為第六個 cycle

3 Problems you met and solutions

當第一個指令仍在 IF stage，因為 pipeline register 的預設值皆為 0，故在第一個 cycle 中，ID stage 會 decode `0x00000000` 這個 instruction，這個 instruction 會被我的 decoder decode 為 `sll r0, r0, 0` 而且因為在 HW4 中，原本的架構有 shifter 和 zero_filled 的設計，所以有一個 `FURslt` 的 MUX，所以 sll 的 `ALU_operation` 我並沒有認真設計，而是直接打為 `1000`，故最後 `ALU` 算出的結果會是 `~r0 or r0 = 0xffffffff`，且這個結果會被存回去 `r0` 中，導致後面的計算發生錯誤，為了解決這個問題，我後來的解法是在每個 pipeline register 中加入 `enable` 這個 wire，當資料在 data memory 或是 register file 寫入時，會去跟 `enable` AND，如果 `enable` 是 1 時，資料才有可能寫入，而這個值是在 IF stage 中檢查目前進來的 instruction 是不是 `0x00000000` 決定的，如果是 `0x00000000`，則這個值會設為 0，代表計算結果不會寫入到 memory 中。經過以上的操作後，結果就會是正確的。

4 Summary

在本次作業中，我了解如何設計 Pipeline Register，並透過在 Single Cycle CPU 中加入 Pipeline Register 達到 Pipeline 的效果。也透過接線的過程，了解 CPU 中會需要哪些 control signal，各個 control signal 是在哪個 stage 被用到。在這次作業中，更加了解 CPU 整個 data flow 的流程。