VLSI System Design (Graduate Level)

Fall 2021

HOMEWORK I

REPORT

Must do self-checking before submission:

Compress all files described in the problem into one tar

All SystemVerilog files can be compiled under SoC Lab environment

All port declarations comply with I/O port specifications

Organize files according to File Hierarchy Requirement

No any waveform files in deliverables

Student name: \_\_顧芳宜\_\_\_\_\_\_\_\_\_

Student ID: \_\_P78101556\_\_\_\_\_\_\_

**Summary:**

這個作業是一個5-stage pipelined CPU的實作，包含IF、ID、EX、MEM、WB stages。其中在ID和MEM 個別實作了一個記憶體：instruction memory和data memory。控制訊號是由Control Unit輸出，stalls則是由Hazard Unit實現，另外計算則是由計算單元ALU運算，最後forwarding則是透過Forward Unit去控制。

**Lesson Learnt:**

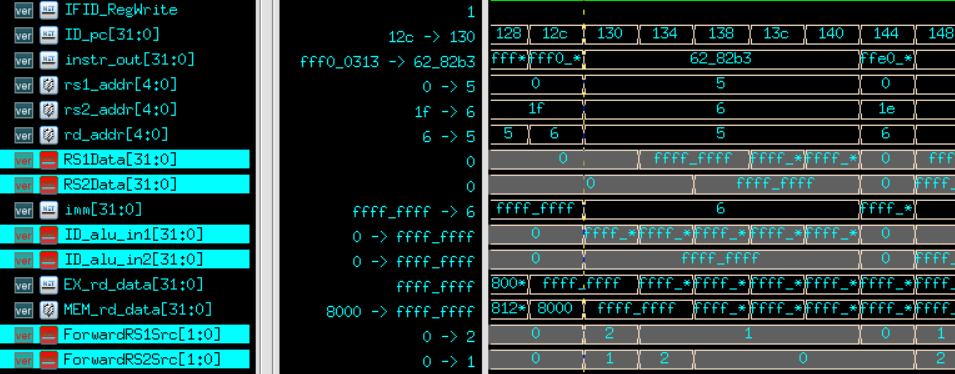
* 深入學習pipelined CPU的運作模式和其中需要考慮的問題，如forwarding、hazard(如何實作stalls)等等。並更加了解記憶體存取資料的方式。
* 了解riscv的指令和其運作方式。
* Register File 剛開始是以combinational circuit實作，但這樣的方式實作會有**latch**產生的問題。若是直接將Register File改成flip-flop，便可以解決latches產生的問題。這代表WB stage是直接實作在Register File裡面，因此需要將MEMWB\_Reg這個flip-flop移除。

**Block Diagram:**

****

**Waveform Verification:**

1. **測試R-Type功能**。R-Type功能皆須經由ALU運算，功能包括ADD、SUB、shift、SLT、XOR、OR、AND。運算後會再存回暫存器。Register File會先存取rs1和rs2的address，並輸出其相對應的值(RS1Data&RS2Data)。再來將*RS1Data*和*RS2Data*傳入ALU運算前需確認是否這個指令跟前兩個指令有Data Dependency，是否需要forwarding。此時Forward Unit的兩個控制項*ForwardRS1Src*和*ForwardRS2Src*會用來判斷是否forward data：0代表無forwarding，1代表forward ALU的計算結果，2代表forward從Data Memory輸出的結果。



另外rs2的部分也需要透過*ALUSrc*判斷到底傳入ALU的值是從immediate generator出來的還是register file (或是forwarding的值)。

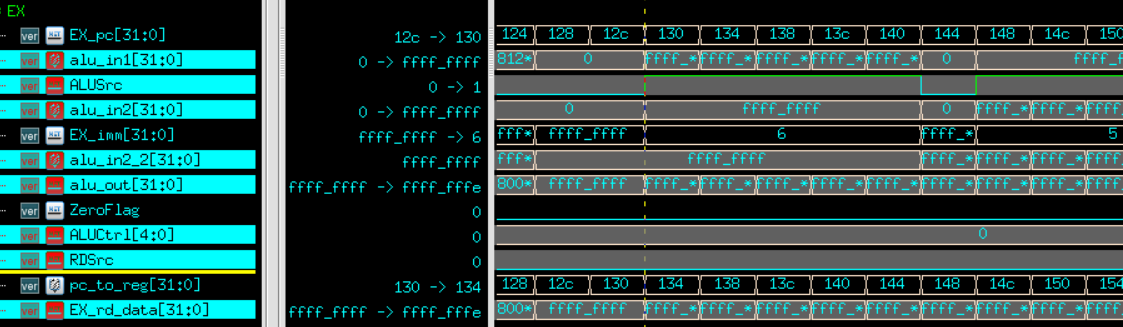
最終傳入ALU，ALU透過ALU Control傳入的控制指令*ALU\_Ctrl*得知需要進行哪種運算。以*add t0,t0,t1*為例(pc=130, instr\_out=6282b3)。rs1\_addr=5,rs2\_addr=6,RS1Data=0,RS2Data=0。由於ForwardRS1Src=2, ForwardRS2Src=1, ALUSrc=1，輸入ALU的兩個值是從EX和MEM forward而來的：

**alu\_in1 = 32’h ffff\_ffff = -1**

**alu\_in2\_2 = 32’h ffff\_ffff = -1**

**結果為 alu\_out = 32’h ffff\_fffe = -2**

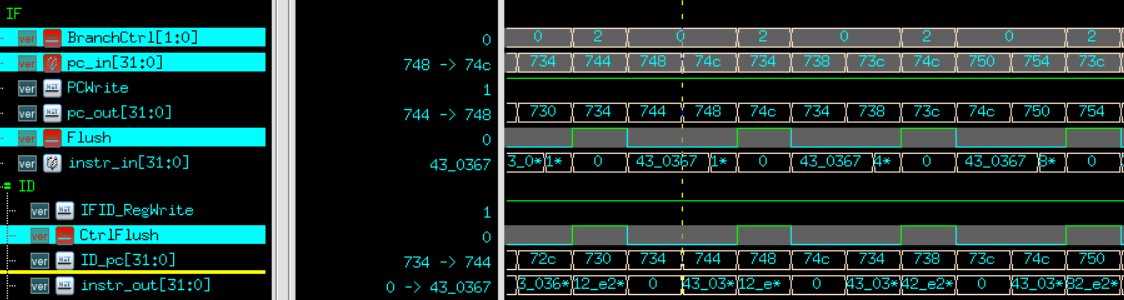
根據RDSrc和MemtoReg，回傳到rd的值可以是進行jump之後的return pc (RDSrc=1)、ALU的計算結果(RDSrc=0)或是從DM輸出的值(MemtoReg=1)。當RegWrite=1時可成功寫入暫存器。



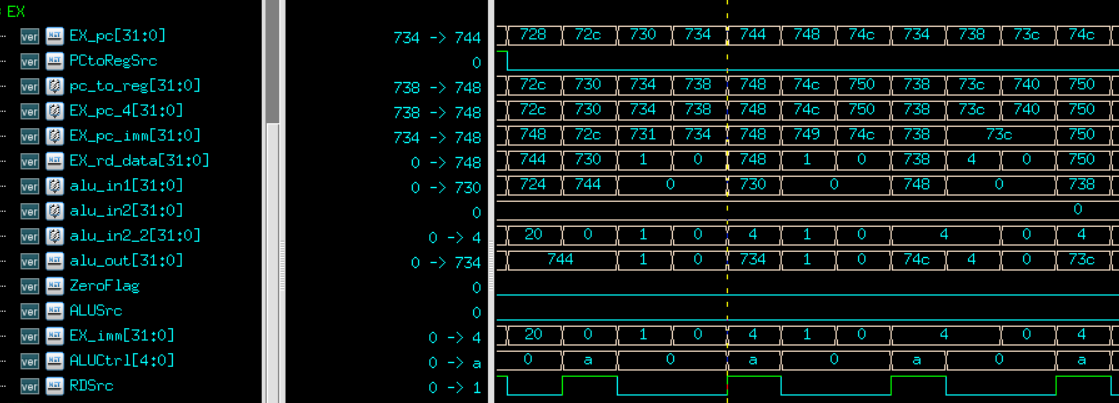
1. **測試JALR。**

由於JALR、JAL和branch所跳到的下一個PC是由位於EX stage的Branch Control Unit來決定，因此需要兩個stalls，而這兩個stalls則是由Hazard Unit輸出的*Flush*和*CtrlFlush*所產生。Flush=1時，instruction會變成0，而CtrlFlush=1時，Control Unit所產生的控制指令如*ALUOP、ALUSrc*等等會被設為0。

以pc=744的00430367— *jalr t1,4(t1)*為例。首先，須經由ALU算出下一個PC的位置(PC=rs1+imm=730+4)，因此ALUSrc=0將imm作為ALU的輸入值。另外，須在Jump之前將原本下一個pc的位置存入暫存器(rd=pc+4=744+4)，而與pc相關的運算並非在ALU中進行，而是透過另外的加法器運算，而*PCtoRegSrc*決定回傳的值為pc+4還是pc+imm。接下來， RDSrc=1、MemtoReg=0。

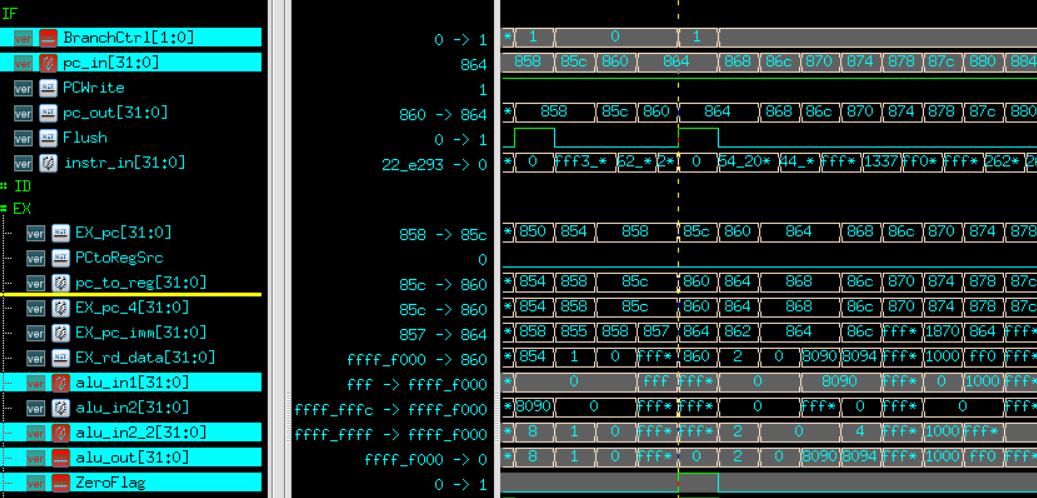


2 stalls



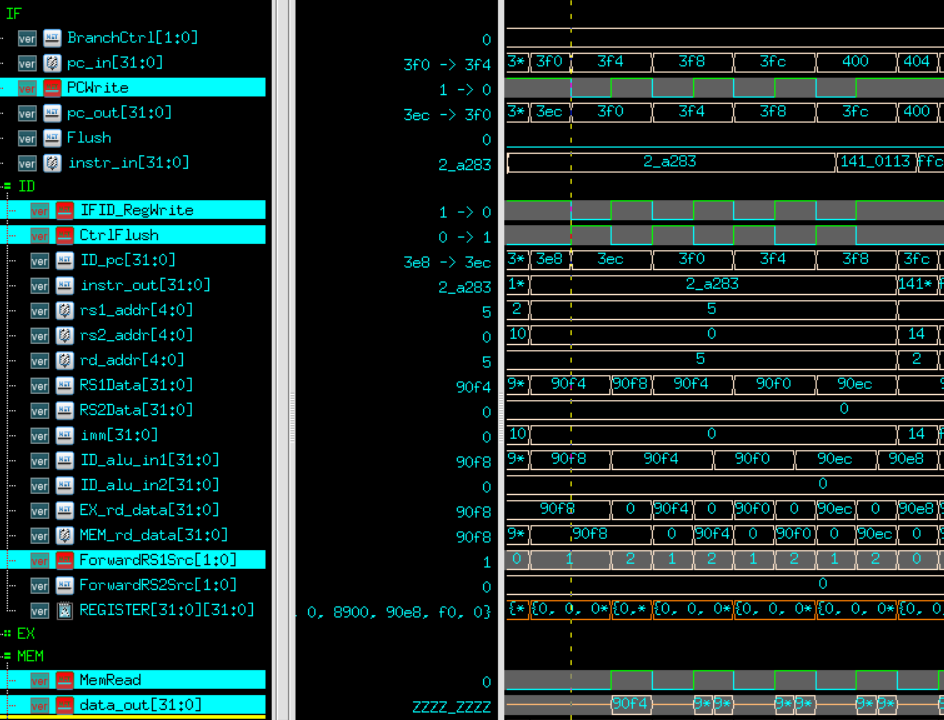
1. **測試B-Type。**

是否Branch是由ALU計算之後所決定。若條件符合，那麼ALU所產生的Zeroflag就會是1’b1，告訴Branch Control Unit這個指令是可以跳的，而Branch Control Unit會輸出*BranchCtrl以*決定下一個pc。下圖是beq從pc=85c成功跳到pc=864的例子。



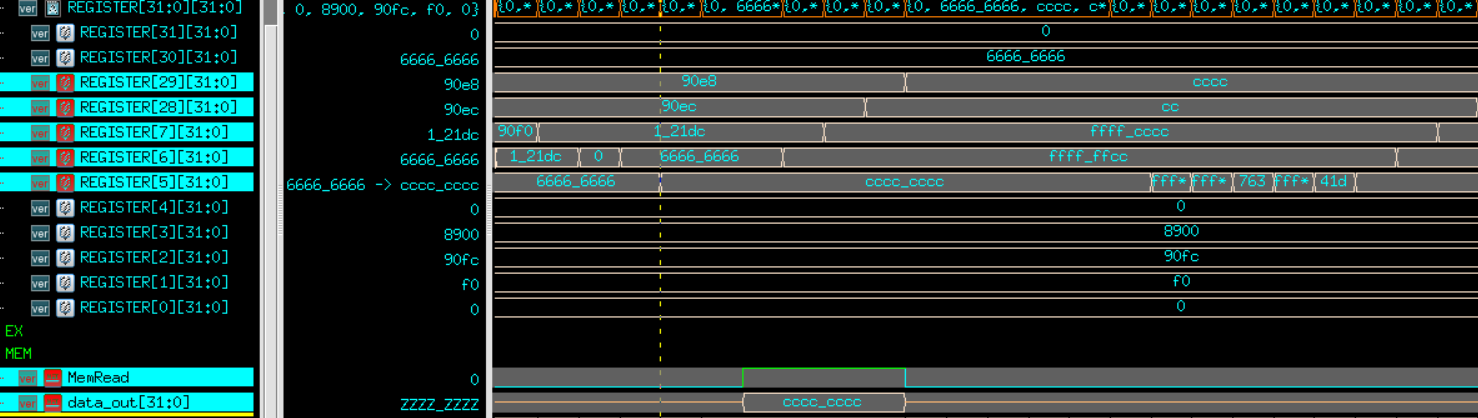
1. **測試Load。**

Load一個值時，MemRead會等於1，且MemtoReg=1使memory輸出的值傳回register。Load要注意的狀況是lw-use的問題。因為從記憶體輸出的資料無法立刻提供給在下一個需要這個資料的指令，因此必須產生一個stall。這個stall是由CtrlFlush產生，並將PCWrite和IFID\_RegWrite設為0。



**stall**

另外，lb、lh、lbu、lhu是再記憶體輸出之後擷取其中片段資料。如下圖記憶體輸出32’h cccc\_cccc，分別輸入lw、lb、lh、lbu、lhu到暫存器$5、$6、$7、$28、$29，也就是32’h cccc\_cccc、32’h ffff\_ffcc、32’h ffff\_cccc、32’hcc、32’h cccc。



1. **測試S-Type。**

Store分為sw、sb、sh，分別是store word、store byte、store halfword，存入32bits、8bits跟16bits。並且需要根據存入的記憶體位置決定資料存入的形式。如下圖為sw、sb、sh的例子，分別將32’b1234\_5678、32’b78、32’b5678存入。而根據存入記憶體的起始位置，需要將byte、halfword的值進行相對應的shift。



