**Computer Organization 2017**

**HOMEWORK III**

系級: 資訊108 學號: E94046157 姓名: 蔡宇軒

**實驗結果圖(snapshot of the results)**

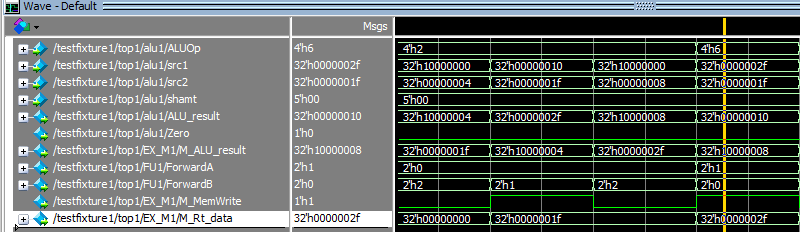
****

**指令波型圖(Snapshot of the instruction execution waveforms.)**

(Please explain why your snapshot is correct, including the wires, signals. The description should be as detailed as possible, e.g. why this situation occurs, and in waveform where does it occur?)

1. Instruction with Forwarding

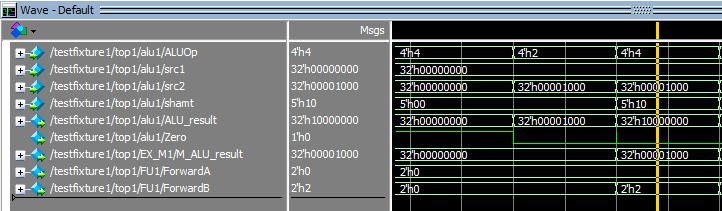
R-type:



Description:

我舉的例子是NO13、14指令，add $s2, $s0, $s1與sw $s2, 8($a0)。看到做完add指令後的結果是0000002f(47)，而後黃線處是sw指令，傳到Mem級後memwrite=1，而因為前面的forwardB=10，傳到M\_Rt\_data才會是正確的值。

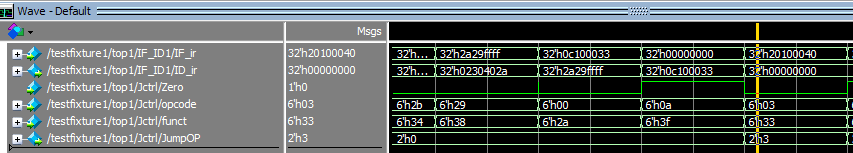
I-type:



Description:

我舉的例子是NO7、8指令，addi $a0, $0, 4096與sll $a0, $a0, 16，可以看到addi做完時，ALUresult是00001000(4096)，而後ALU做到SLL指令時，透過ForwardB = 10，也就是來自於前一個ALU結果的Forwarding，把4096再input，並左移16bit得到10000000

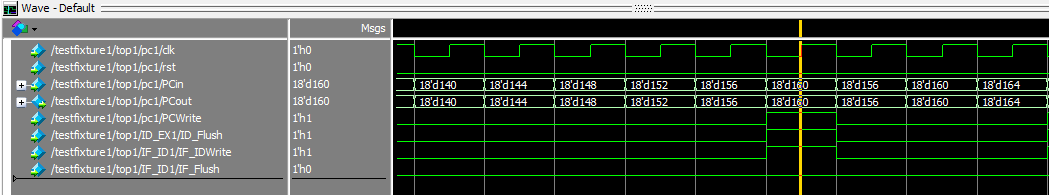
J-type:



Description:

Jtype的forwarding，除非牽扯到R[31]，否則只單純計算jump的地址，需要用到的應該只有hazard detection unit。

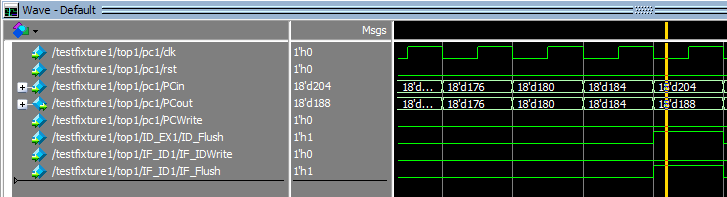
1. Load Stall:



Description:

我舉的例子是NO37，lw $s0, 40($a0)。由圖可以看到經過lw(160)時並沒有繼續往164走，而是觸發PC\_Write、IFID\_Write及Flush指令，把指令停滯一個cycle。

1. Branch Delay (& Flush):



Description:

我舉的是jal 0x004000cc [fun1]，如圖可以發現，176是jal進入pc的編號，而後會再往前兩個指令，到EX級才跳往正確的指令，預測是不做branch，因此會往180、184做兩個指令。當Jump\_Ctrl偵測到要跳指令時，觸發flush指令，使多做的兩個指令清除，而後Jump\_Ctrl會把pc帶往正確的位置。

If you CPU data is from Figure 5 in the home, show your CPU datapath and explain why you want to design your CPU this way.

Reason:

我的CPU datapath與講義上相同。

**心得(Report)**

這次花的總時間比做第一次作業時還少，因為第一次作業對modelsim還不熟，連怎麼debug都花了一段時間才搞懂。但是這次的作業難處在於，不像第一次作業一次只要看一個指令，因為pipelined cpu是一次處理多個指令，要花比較多時間看出現在到底指令在哪一級。最困擾我的還是一些小細節粗心導致錯誤。這也讓我更了解了軟體工程最忌諱粗心，因為一個小錯誤就可能使整個程式錯誤。