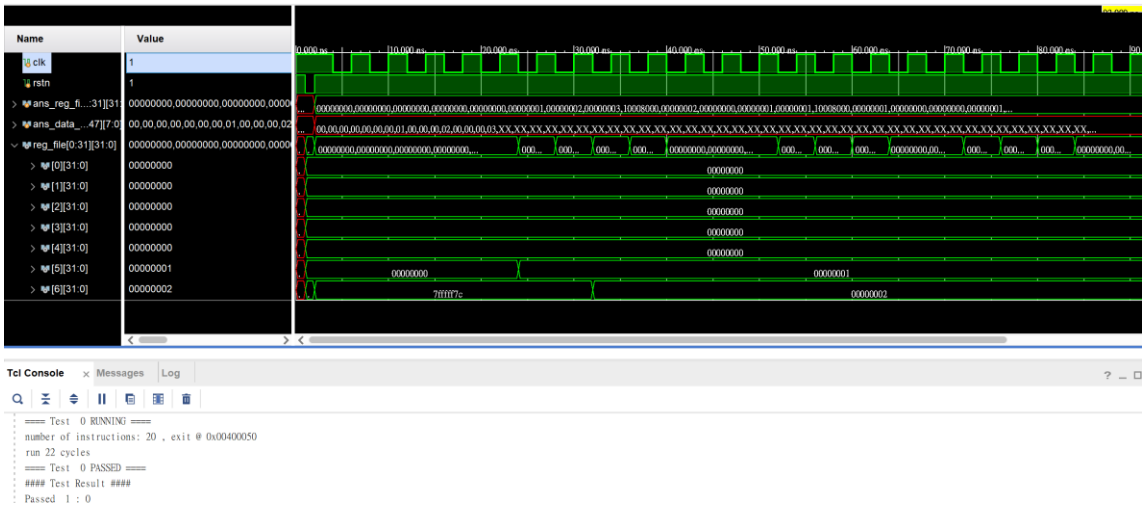
Lab3: Pipelined Processor-Part1

111550142 尤瑋辰

1. Experimental Result
   a. Show the waveform screen shot of the test we provided.

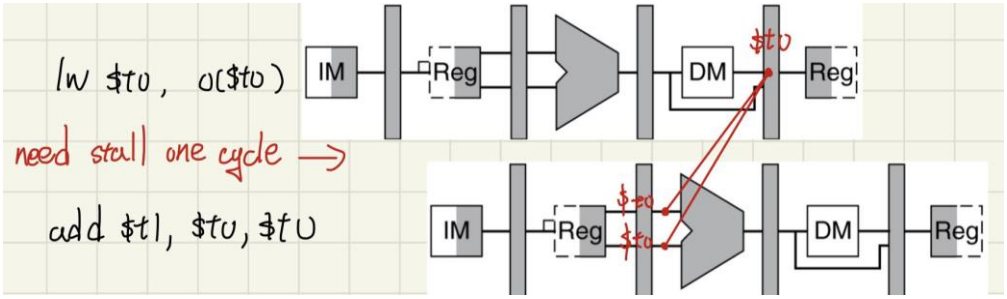


   b. What other cases you've tested? Why you choose them?

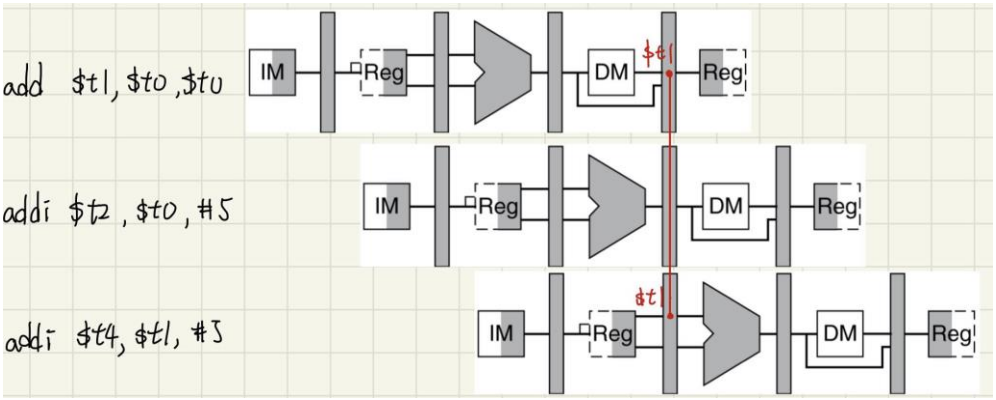
```
        .data   0x10008000      # start of Dynamic Data (pointed by $gp)
        .word   0x0             # 0($gp)
        .word   0x1             # 4($gp)
        .word   0x2             # 8($gp)
        .word   0x3             # 12($gp)
        .word   0x0             # 16($gp)
        .text   0x00400000      # start of Text (pointed by PC),
                                # Be careful there might be some other instructions in JsSPIM.
                                # Recommend at least 9 instructions to cover out those other instructions.
main:   addi    $v1, $v1, 0x1111 # v3 = v3 + 0x1111
        lw      $a0, 0($gp)
        lw      $a1, 4($gp)
        sw      $v1, 16($gp)    # save word v3 to gp+16, since addi result has been write to v3
        add     $t0, $a1, $gp   # t0 = gp, since `lw $a1` is at ID
        lw      $a2, 8($gp)
        add     $t1, $a1, $a1   # t1 = 2, since `lw $a1` is at MEM (when this enter ID, lw enter WB)
        lw      $a3, 12($gp)
        sub     $t2, $0, $t1    # t2 = 0, since `add $t1` is at MEM
        beq     $v1, $0, end    # not taken
        slt     $t3, $a2, $a3   # t3 = 1, Read after write need to wait for ? more instructions?
        slt     $t4, $a1, $a2   # t4 = 1
        or      $t5, $gp, $t3   # t5 = gp, since t3 has not been written yet
        and     $t6, $a1, $t3   # t6 = 1, since t3 is written in this cycle
        beq     $t3, $t4, end   # taken
        addi    $s1, $0, 0x0001 # will this executed even branch is taken? How about others?
        sw      $gp, 0($gp)     # gp =  0x10008000;
        addi    $s3, $0, 0x0003
        add     $s4, $a2, $a2
        add     $s5, $a2, $a3
        add     $s6, $a3, $a3
end:    add     $s7, $s4, $a2   # s7 = 2
```

I tested sw, addi, and beq instructions (with the branch not taken) to ensure they work.

2. Answer the following Questions:
   a. For each code sequence below, state whether it must stall, can avoid stalls using only forwarding, or can execute without stalling or forwarding.
      1. Sequence 1 must stall for one cycle.



      2. Sequence 2 can use only forwarding (MEM/WB.Rd = ID/EX.Rs)



      3. Sequence 3 can execute without stalling or forwarding since all 'addi' instructions are reading data from $t0, and no other instruction is writing data to $t0.
   b. Explain the difference between throughput and latency.
      Throughput, the number of instructions completed per second.
      Latency, the time it takes for instructions to be executed.
   c. A group of students were debating the efficiency of the five-stage pipeline when one student pointed out that not all instructions are active in every stage of the pipeline. After deciding to ignore the effects of hazards, they made the following five statements. Which ones are correct? Explain why or why not.
      1. False. First, ALU instruction need to write data back to register. Thus, it can not take fewer stages than five. Second, the pipeline speedup is due to increased throughput not latency.

2. True. The reason a pipelined processor can achieve speedup is by increasing throughput. However, reducing the number of cycles instructions take will improve latency but won't impact throughput.
3. True. Branch and jump only need 3 stages (IF, ID, EX), so there is some opportunity for improvement.
4. True. Without considering hazards, lengthening the pipeline can enable more instructions to be executed within a unit of time, thus enhancing overall throughput.