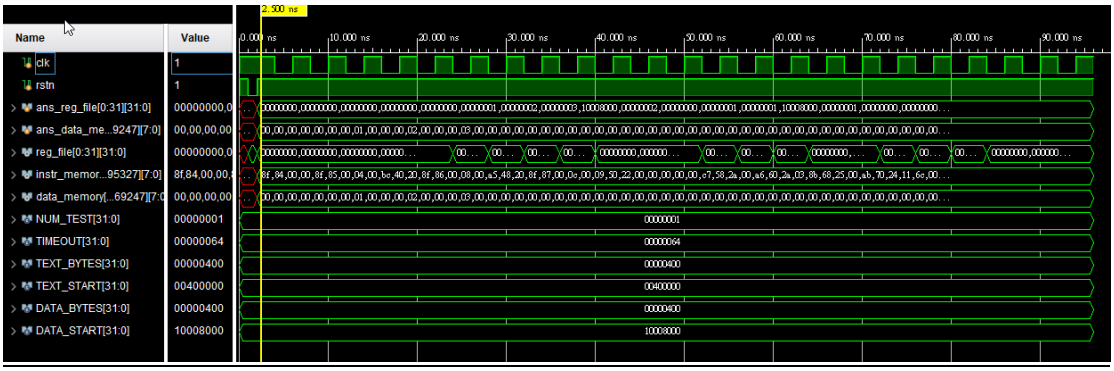
Computer Organization

Lab 3 pipelined processor-part 1

111550168 陳奕

1. Experimental Result

(1) Waveform



(2) Othercases

I reordered the instructions such that data hazards do not occur. Additionally, I implemented the addi instruction to test whether the processor hadle addi instruction correctly.

```
1   8f 85 00 04  // [00400004] lw $5, 4($28)                    ; 11: lw      $a1, 4($gp)
2   8f 86 00 08  // [00400008] lw $6, 8($28)                    ; 13: lw      $a2, 8($gp)
3   8f 87 00 0c  // [0040000c] lw $7, 12($28)                   ; 15: lw      $a3, 12($gp)
4   8f 84 00 00  // [00400010] lw $4, 0($28)                    ; 10: lw      $a0, 0($gp)
5   00 00 00 00  // [00400014] nop                              ; 17: nop
6   00 a5 48 20  // [00400018] add $9, $5, $5                   ; 14: add     $t1, $a1, $a1
7   00 c7 58 2a  // [0040001c] slt $11, $6, $7                  ; 18: slt     $t3, $a2, $a3
8   00 a6 60 2a  // [00400020] slt $12, $5, $6                  ; 19: slt     $t4, $a1, $a2
9   00 bc 40 20  // [00400024] add $8, $5, $28                  ; 12: add     $t0, $a1, $gp
10  00 09 50 22  // [00400028] sub $10, $0, $9                  ; 16: sub     $t2, $0, $t1
11  03 8b 68 25  // [0040002c] or $13, $28, $11                 ; 20: or      $t5, $gp, $t3
12  00 ab 70 24  // [00400030] and $14, $5, $11                 ; 21: and     $t6, $a1, $t3
13  20 E6 00 01  // [00400034] addi $6, $7, 0x0001              ; 22:   addi    $a2, $a3, 1
14  11 6c 00 06  // [00400038] beq $11, $12, 24 [end-0x00400030];23: beq     $t3, $t4, end
15  00 05 88 20  // [0040003c] add $17, $0, $5                  ; 24: add     $s1, $0, $a1
16  00 06 90 20  // [00400040] add $18, $0, $6                  ; 25: add     $s2, $0, $a2
17  00 07 98 20  // [00400044] add $19, $0, $7                  ; 26: add     $s3, $0, $a3
18  00 c6 a0 20  // [00400048] add $20, $6, $6                  ; 27: add     $s4, $a2, $a2
19  00 c7 a8 20  // [0040004c] add $21, $6, $7                  ; 28: add     $s5, $a2, $a3
20  00 e7 b0 20  // [00400050] add $22, $7, $7                  ; 29: add     $s6, $a3, $a3
21  02 86 b8 20  // [00400054] add $23, $20, $6                 ; 30: add     $s7, $s4, $a2
22
```

The registers value after running through the instructions are identical to the expected registers value, which is shown below.

```
00000000  // R0 (r0)
00000000  // R1 (at)
00000000  // R2 (v0)
00000000  // R3 (v1)
00000000  // R4 (a0)
00000001  // R5 (a1)
00000004  // R6 (a2)
00000003  // R7 (a3)
10008001  // R8 (t0)
00000002  // R9 (t1)
fffffffe  // R10 (t2)
00000001  // R11 (t3)
00000001  // R12 (t4)
10008001  // R13 (t5)
00000001  // R14 (t6)
00000000  // R15 (t7)
```

```
00000000  // R16 (s0)
00000001  // R17 (s1)
00000004  // R18 (s2)
00000003  // R19 (s3)
00000000  // R20 (s4)
00000000  // R21 (s5)
00000000  // R22 (s6)
00000004  // R23 (s7)
00000000  // R24 (t8)
00000000  // R25 (t9)
00000000  // R26 (k0)
00000000  // R27 (k1)
10008000  // R28 (gp)
7fffff74  // R29 (sp)
00000000  // R30 (s8)
00000000  // R31 (ra)
```

2. Questions

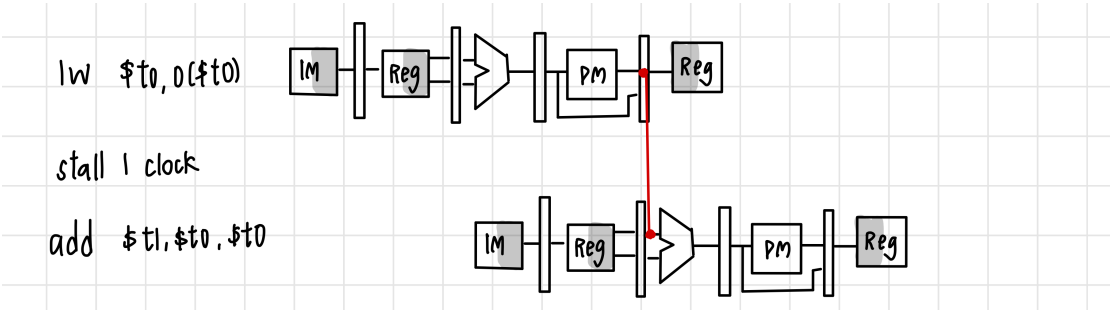
(1) Stalling or forwarding

Sequence 1	Sequence 2	Sequence 3

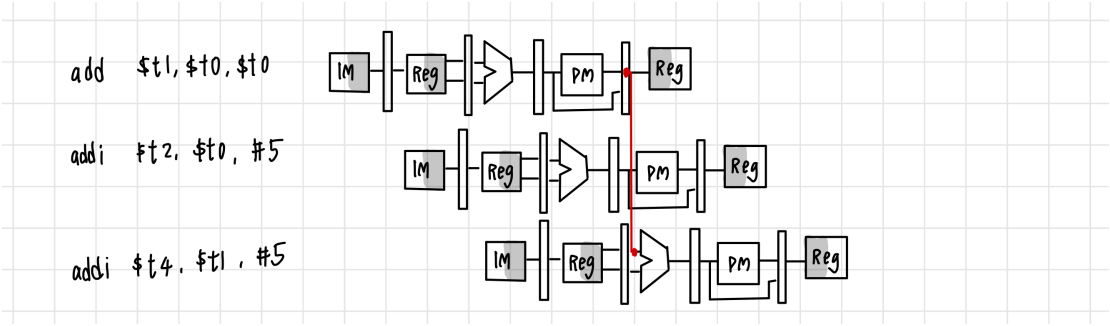
`lw $t0,0($t0)`
`add $t1,$t0,$t0`
	`add $t1,$t0,$t0`
	`addi $t2,$t0,#5`
	`addi $t4,$t1,#5`
		`addi $t1,$t0,#1`
		`addi $t2,$t0,#2`
		`addi $t3,$t0,#2`
		`addi $t3,$t0,#4`
		`addi $t5,$t0,#5`

Sequence 1: must stall 1 cycle and need forwarding.

$t0 from load will only be available after the memory read, which completes in the MEM stage of the pipeline. However, add instruction needs $t0 in the ID stage immediately after IF stage, thus it requires stalling for one cycle. And Forwarding $t0 from MEM stage.



Sequence 2: can avoid stall using only forwarding.



The first addi instruction can execute without issues since $t0 is not being modified by the preceding add instruction. The second addi requires the result of the first add instruction ($t1). This result will be available via forwarding from the MEM stage of the first add to the EX stage of the second addi.

Sequence 3: can execute without stalling or forwarding.

These instructions can execute without stalling or forwarding because there are no data dependencies between them that would affect the outcome.

(2) Explain the difference between throughput and latency.

Throughput is the number of instructions completed per unit of time. While Latency measures the time it takes for a single instruction to travel through the pipeline and complete its execution. Throughput focuses on the overall rate of instruction completion, while latency measures the time it takes for a single instruction to be processed.

(3) True or false

1. (1%) Allowing jumps, branches, and ALU instructions to take fewer stages than the five required by the load instruction will increase pipeline performance under all circumstances.

False.

Reducing the number of stages for branches and jumps can introduce new challenges, such as the resulting complexity in control logic, the efficiency of branch predictions, and the overall balance of pipeline resource usage. These complexity might lead to inefficiencies or increased hazards, outweighing the intended performance improvements.

2. (1%) Trying to allow some instructions to take fewer cycles does not help, since the throughput is determined by the clock cycle; the number of pipe stages per instruction affects latency, not throughput.

True.

The number of stages in a pipeline affects the latency of each instruction, but the throughput is governed by the clock cycles. Reducing the number of stages some instructions passing through doesn't increase the throughput, as throughput depends on how quickly each instruction can be fed through the pipeline, which is dictated by the clock cycle.

3. (1%) You cannot make ALU instructions take fewer cycles because of the writeback of the result, but branches and jumps can take fewer cycles, so there is some opportunity for improvement.

True.

ALU instructions need to write back results, and thus may not benefit from reduced cycles like branches and jumps, which can sometimes complete earlier without needing to write back data.

4. (1%) Instead of trying to make instructions take fewer cycles, we should explore making the pipeline longer, so that instructions take more cycles, but the cycles are shorter. This could improve performance.

False.

Making the pipeline longer by adding more stages (and thus potentially shortening the duration of each stage or cycle) does not necessarily improve performance. While it might help in certain high-frequency designs by enabling a higher clock rate, it can also increase latency and the complexity of handling hazards. For example, a longer pipeline may introduce more problems such as increased pipeline stalls and greater overhead in managing these stalls and hazards.