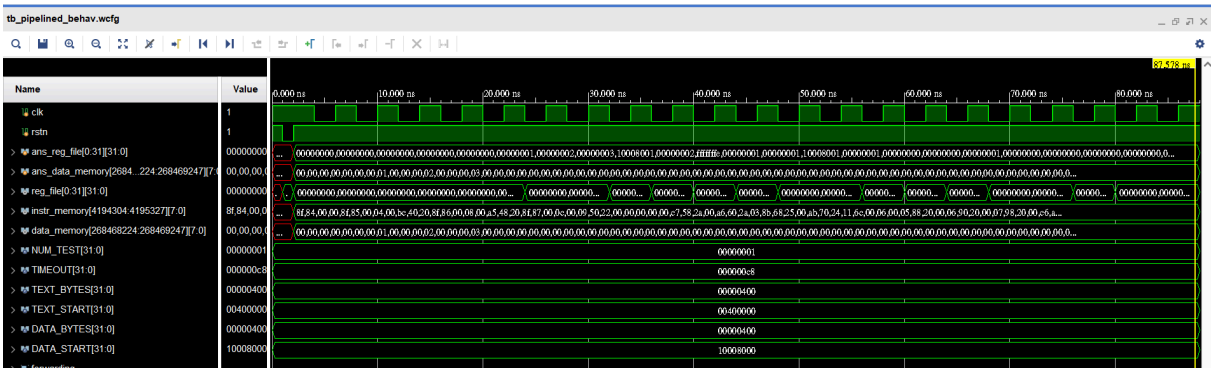
Lab 4: Pipelined Processor - Part 2

111613025 黃綵誼

Experimental Result

1. (1%)Show the waveform screenshot of the test we provided.



2. (1%)What other cases have you tested? Why did you choose them?

(1)1.s

```
1   main:   lw      $t0, 0($gp)         # Load t0 from memory t0 = 0
2           addi    $t1, $t0, 5         # test addi
3           lw      $t1, 4($gp)
4           addi    $t2, $t1, 4         # test if forward correctly from l
5           add     $t3, $t2, $t1
6           lw      $t4, 8($gp)
7           sub     $t5, $t4, $t3
8           or      $t6, $t5, $t4       # test or
9           and     $t7, $t6, $t3       # test and
10
11  loop:   beq     $t4, $t2, end       # forward t4 from previous loop
12          addi    $t4, $t4, 1         # t4 = t4 + 1
13          beq     $0,  $0,  loop
14          add     $s0, $0, $t0        # Will not execute
15          add     $s1, $0, $t1
16
17  end:    add     $s2, $t5, $t4
18          slt     $s3, $t2, $t3       # test slt
```

(2)2.s

```
1   main:   lw      $t0, 0($gp)         # Load t0 from memory t0 = 0
2           lw      $t4, 0($gp)
3           lw      $t1, 4($gp)         # Load t1 from memory t1 = 1
4           lw      $t2, 0($gp)         # t2 = 0
5           addi    $t2, 2
6
7   loop:   beq     $t0, $t2, end       # should stall 1, check if beq can
8           addi    $t0, $t0, 1
9           sub     $t3, $t1, $t0
10          lw      $t3, 8($gp)          # t3 = 2
11          beq     $t2, $t3, loop       # should stall 2, taken
12          addi    $t4, $t4, 1
13          add     $a3, $t4, $t4
14
15  end:    sw      $t4, 0($gp)       # s3 = 0
16          lw      $t2, 0($gp)       # test lw after sw
17          sw      $t2, 12($gp)      # test sw after lw
18
```

Answer the following Questions:

1. (2%) List out the equation to detect EX & MEM hazard in forwarding unit. Which part of the equation in textbook p. 369 is wrong?

   This part in p. 369 is wrong.

   ```
   if (MEM/WB.RegWrite
   and (MEM/WB.RegisterRd ≠ 0)
   and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
           and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs)
   and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

   if (MEM/WB.RegWrite
   and (MEM/WB.RegisterRd ≠ 0)
   and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
           and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRt)
   and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01
   ```

   It should be:

```
always @(*)
    if (EX_MEM_reg_write & EX_MEM_rd != 0 & EX_MEM_rd == ID_EX_rs) be
        forward_A = 'b10;
    end else if (MEM_WB_reg_write & MEM_WB_rd != 0 & MEM_WB_rd == ID_
        forward_A = 'b01;
    end else begin
        forward_A = 'b00;
    end

always @(*)
    if (EX_MEM_reg_write & EX_MEM_rd != 0 & EX_MEM_rd == ID_EX_rt) be
        forward_B = 'b10;
    end else if (MEM_WB_reg_write & MEM_WB_rd != 0 & MEM_WB_rd == ID_
        forward_B = 'b01;
    end else begin
        forward_B = 'b00;
    end
```

2. (2%) In forwarding for `beq`, is forwarding from MEM/WB to ID needed? Why?

   Yes, it's needed because in my pipelined processor design, the branch decision is made in the ID stage. This means that the beq instruction must compare the values of the registers (rs and rt) in the ID stage to determine if the branch should be taken.

   Therefore, we need forwarding from MEM/WB to ID ensures that if the beq instruction requires a value that is currently being written back to the register file, the correct value is forwarded directly to the ID stage without waiting for it to be written back to the register file. For example `beq` right after `lw`.

3. (2%) Briefly explain how you insert 2 stalls when `beq` reads registers right after `lw` writes it.

   I add some extra signal, like `IF_ID_branch`, `EX_MEM_mem_to_reg` in `hazard_detection.v` to determine if `beq` is at ID stage and if `lw` is the previous instruction.

   For lw -> beq, after the first stall, `lw` enter EXE stage and `beq` is at ID stage, then I use `((EX_MEM_mem_to_reg & IF_ID_branch) & ((EX_MEM_write_reg == IF_ID_rs) | (EX_MEM_write_reg == IF_ID_rt)))` to set the second stall.

```
assign stall = ((ID_EX_mem_read) & ((ID_EX_rt == IF_ID_rs) | (ID_EX_r
                // e.g., lw -> add
               ((EX_MEM_mem_to_reg & IF_ID_branch) & ((EX_MEM_write_r
                // e.g., lw -> beq
                ((IF_ID_branch & ID_EX_reg_write) & ((ID_EX_rt == IF_
                // e.g.,add -> beq
```

4. (2%) `sw` right after `lw` is quite common since copying and pasting data from one address to another is used frequently. In the textbook, a stall is followed by the `lw` in this case. Is it possible to remove this stall? How?

   Yes, this stall can be removed by forwarding write back data for `lw` in WB stage to write data for `sw` in MEM stage.