14 Pipelined CPU Design with Stall Capability

14.1 Objectives

After completing this lab, you will:

- Implement pipeline stall to handle RAW hazard after Load instruction
- Implement pipeline stall for handling control hazards

14.2 Handling RAW Hazard after Load

Unfortunately, not all data hazards can be forwarded. Load has a delay that cannot be eliminated by forwarding. In the example shown in Figure 14.1, the LW instruction does not read data until the end of CC4. Thus, data cannot be forward to ADD instruction at the end of CC3. However, the Load instruction can forward data to the 2nd next and later instruction.

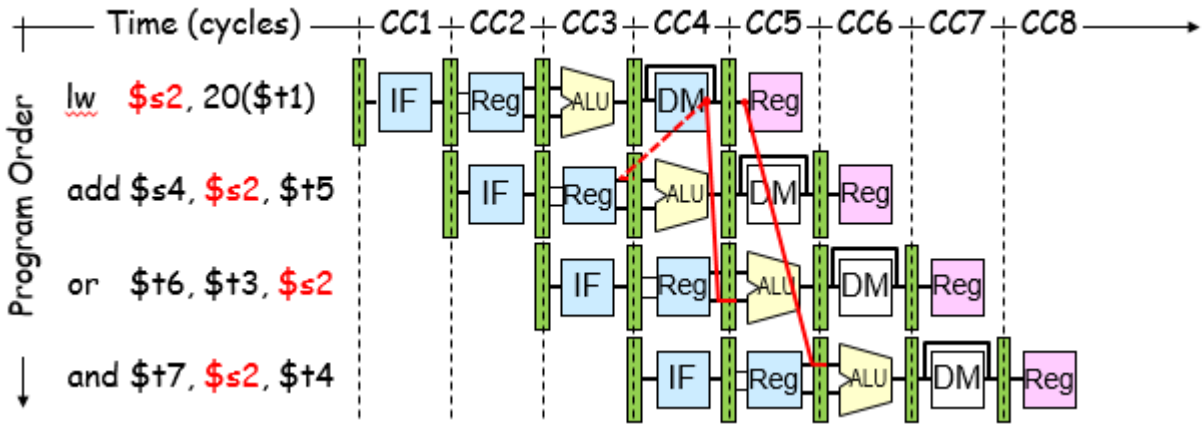

Figure 14.1: RAW data hazard following a Load instruction.

Whenever a RAW hazard after a Load instruction occurs, we have no choice but to stall the pipeline. Stalling the pipeline requires that the instruction in the ID stage be replaced by a NOP (No Operation) instruction and freezing the content of PC and Instruction registers for one clock cycle. This is achieved by replacing the control signals by 0 signals (called a Bubble) and disabling the PC and IR registers as illustrated in Figure 14.2. The Condition for stalling the pipeline is as follows:

if ((EX.MemRead == 1) // Detect Load in EX stage

and (ForwardA==1 or ForwardB==1)) Stall // RAW Hazard

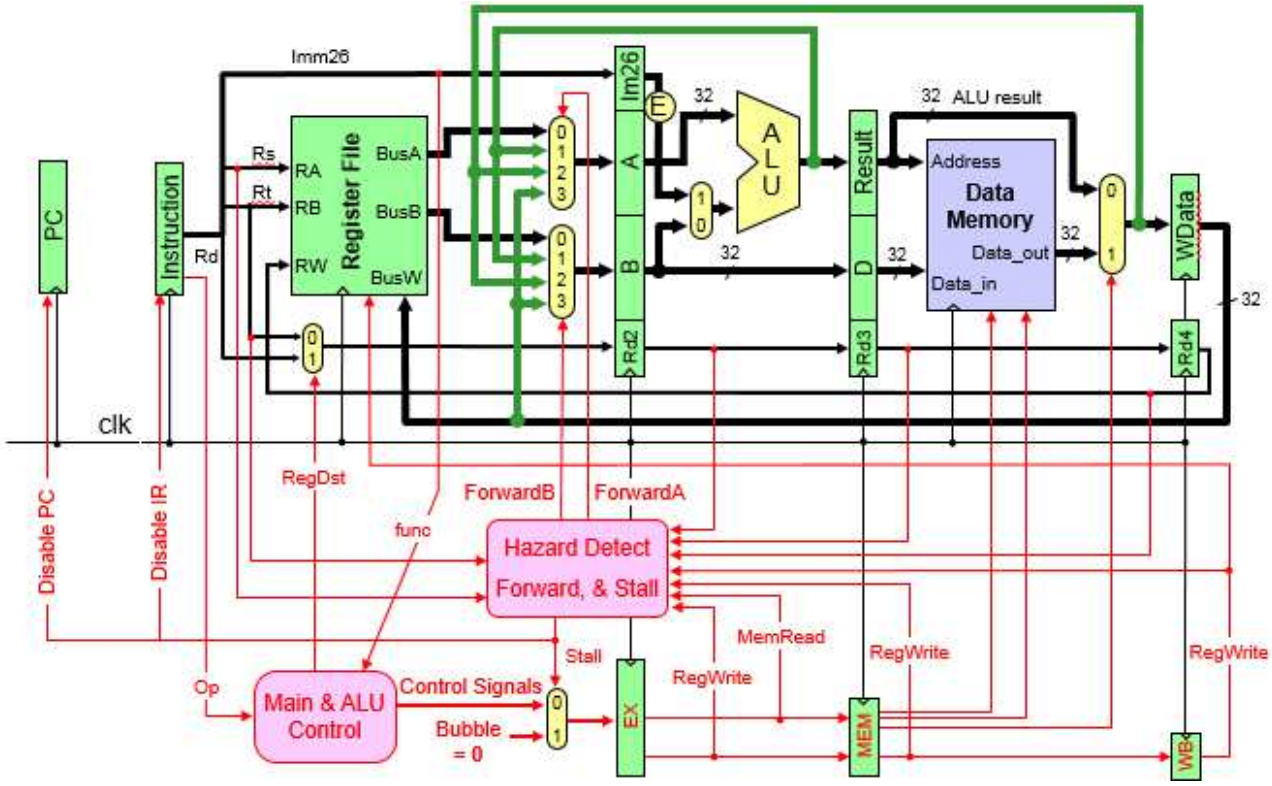

Figure 14.2: Pipeline stall due to data hazard following a Load instruction.

14.3 Handling Control Hazards

Jump and Branch can cause great performance loss. Jump instruction needs only the jump target address while Branch instruction needs two things:

- Branch Result Taken or Not Taken
- Branch Target Address
  - PC + 4 If Branch is NOT taken
  - PC + 4 + 4 × immediate If Branch is Taken

Jump and Branch targets are computed in EX stage at which point two instructions have already been fetched. For Jump, the two instructions need to be flushed. For Branch, the two instructions are flushed if Branch is Taken. Figure 14.3 illustrates an example of handling a control hazard by replacing the two fetched instructions by bubbles. Control logic detects a Branch instruction in the 2nd Stage. The ALU computes the Branch outcome in the 3rd Stage. Next1 and Next2 instructions will be fetched anyway. Thus, we need to convert Next1 and Next2 into bubbles if branch is taken.

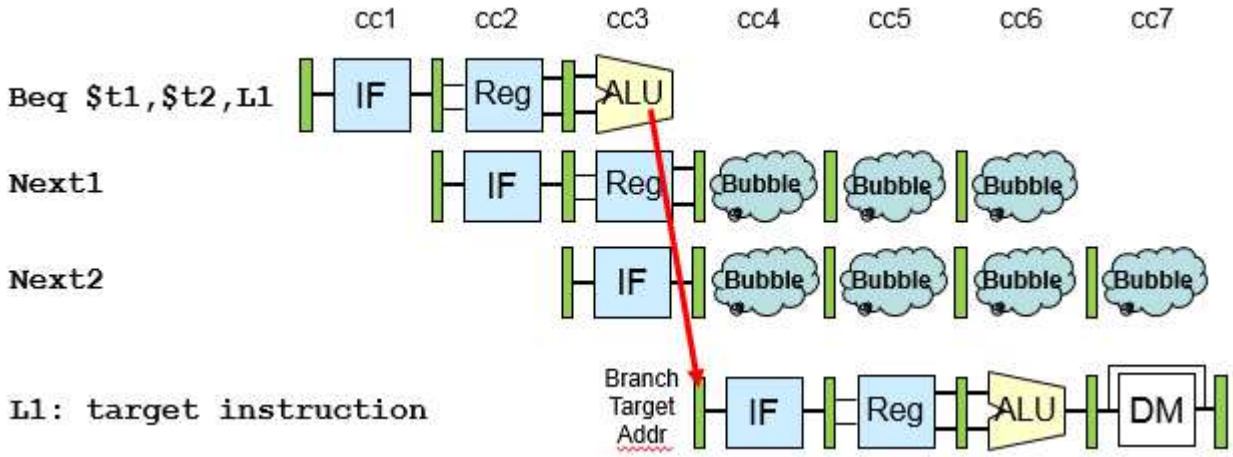

Figure 14.3: Example of handling a control hazard.

Converting the instruction in the 2nd stage into a bubble can be achieved by replacing the control signals by a bubble. Converting the instruction in 1st stage into a NOP instruction is achieved by Resetting the IR register using synchronous reset. The reset is signal is connected to the PCSrc signal which is set when the branch is taken. Figure 14.4 illustrates the implementation of handling control hazards.

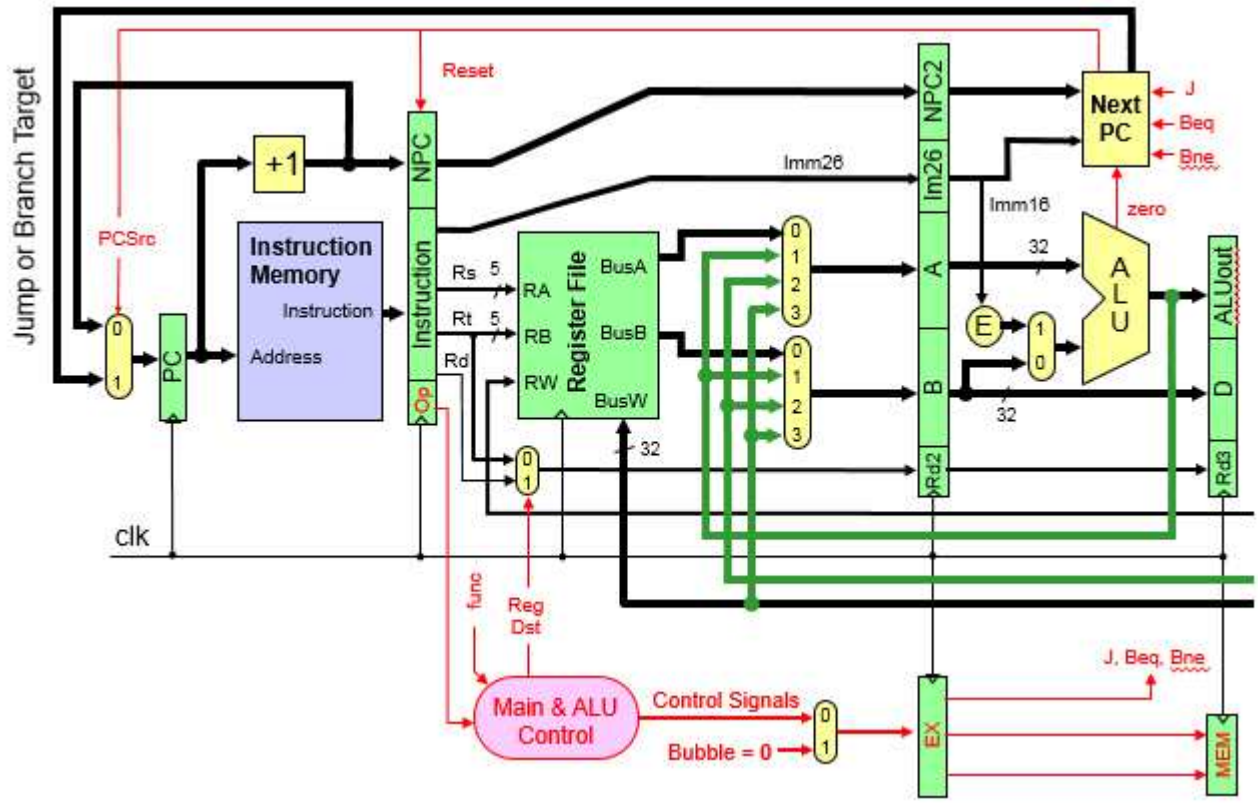

Figure 14.4: Handling control hazards.

14.4 In-Lab Tasks

1. Implement handling of RAW hazard occurring after LOAD instruction in your pipelined CPU design.
2. Test the correctness of your implementation by running the MIPS code fragment given below and verify its correct execution:

```
I1:   ADDI $s0, $0, 10
I2:   ADDI $s1, $0, 5
I3:   SLL  $s0, $s0, 4
I4:   LW   $s2, 0($s0)
I5:   ADD  $s2, $s2, $s1
I6:   SW   $s2, 4($s0)
```

3. Implement control hazard handling in your pipelined CPU design.
4. Test the correctness of your implementation of handling control hazards by running the following MIPS code fragment and verify its correct execution:

```
I1:   ADDI $a1, $0, 10
I2:   ADD $t5, $0, $0
I3:   LW $t2, 0($t5)
I4:   ADD $t3, $t2, $t2
I5:   SW $t3, 0($t5)
I6:   ADDI $a1, $a1, -1
I7:   ADDI $t5, $t5, 4
I8:   BNE $a1, $0, I3
```