Bilkent University

Cs224-01

Osman Buğra Aydın

21704100

Lab05

13 May 2020

B-) Hazards

There are four hazards that we can list. Three of them are data hazards and remaning one is control hazard. To specify data hazards, they are compute-use, load-use and load-store. Branch is a control hazard.

C-) Explanation

1-) Compute-use Hazard (Data Hazard)

When Does It Occur?

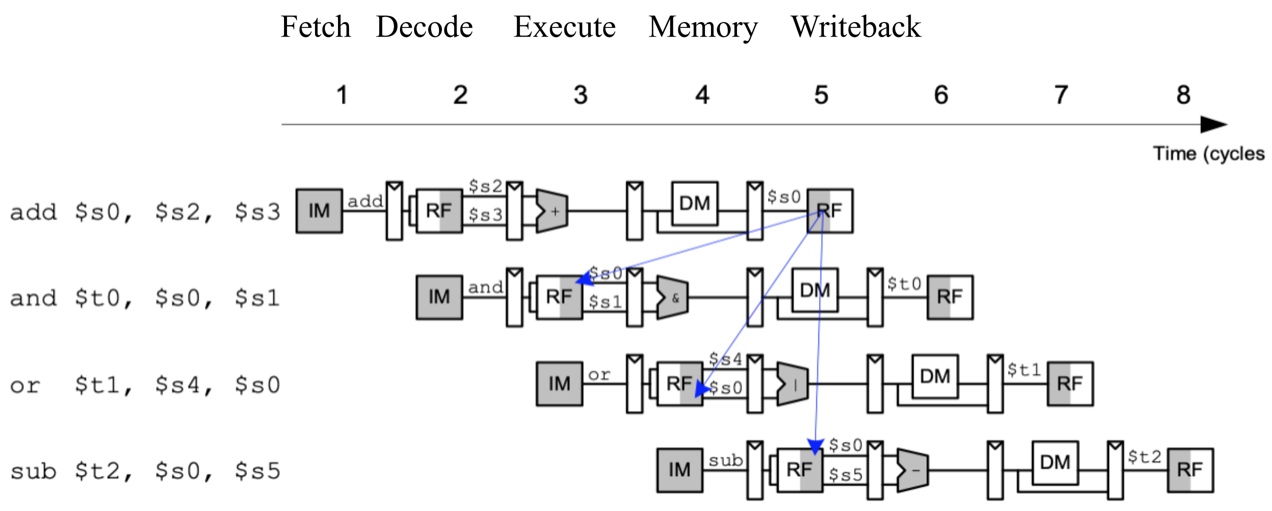
Compute-use occurs when the result of an instruction is not written back before next instruction uses it.

1-) add $t0, $t1, $t3
2-) add $s0, $t0, $t3

In this example, the result of the first instruction is not written back to rd register which is $t0 while second instruction tries to use the value of $t0.

Why Does It Occur?

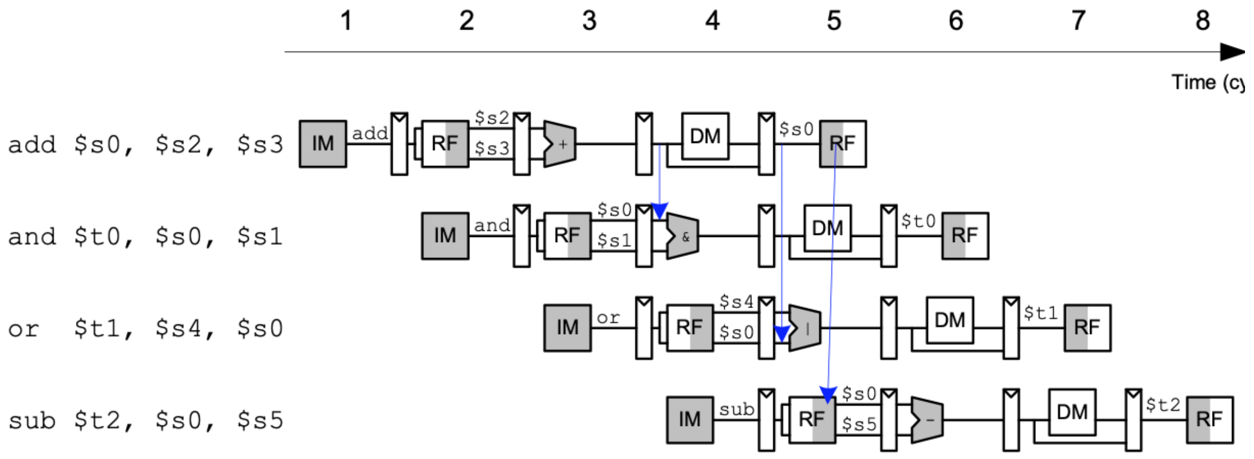
The result of the first instruction is calculated in execute stage while next instruction tries to fecth data from rd register in decode stage. The calculation will not be written back to register till the end of the writeback stage. It means, the result of calculation is not arrived to rd when next instruction tries to use it. Here is an example why first instruction cannot keep up with next instruction.



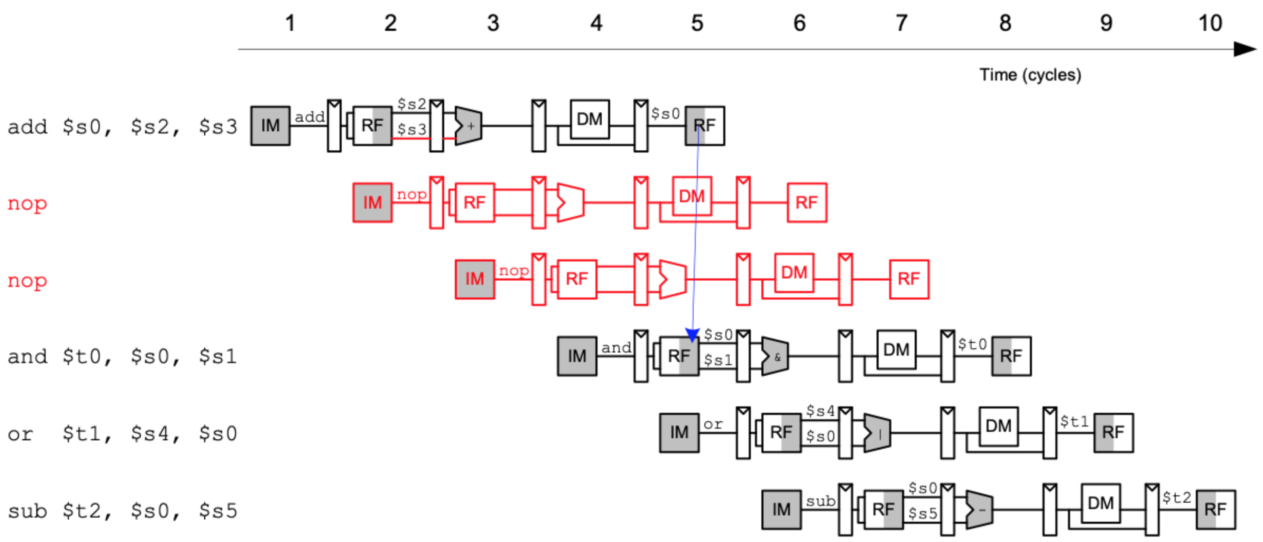
How Can It Be Solved?

As it is said previously, the result of instruction is calculated in execute stage. The problem will be solved if the result in the execute stage is forwarded to the next instruction. Stalling can be a solution for this problem but it is not as efficient as forwarding because there is no waiting in forwarding whereas stalling requires program to wait 2 stage. Furthermore, the problem can be solved with using 2 nop instructions between instructions which may create this hazard. Here is a example to solution.

1-)Solution with Forwarding



2-)Solution with Double Nops



2-) Load-use Hazard (Data Hazard)

When Does It Occur?

Load-use hazard occur when lw instruction is not finished its work and next instruction tries to use register whose result should have been result of lw instruction.

1-) lw $t0, 4($t1)
2-) add $t2, $t0, $t3

In this example, the value of $t0 is not computed yet while next instruction tries to fecth the value of $t0.

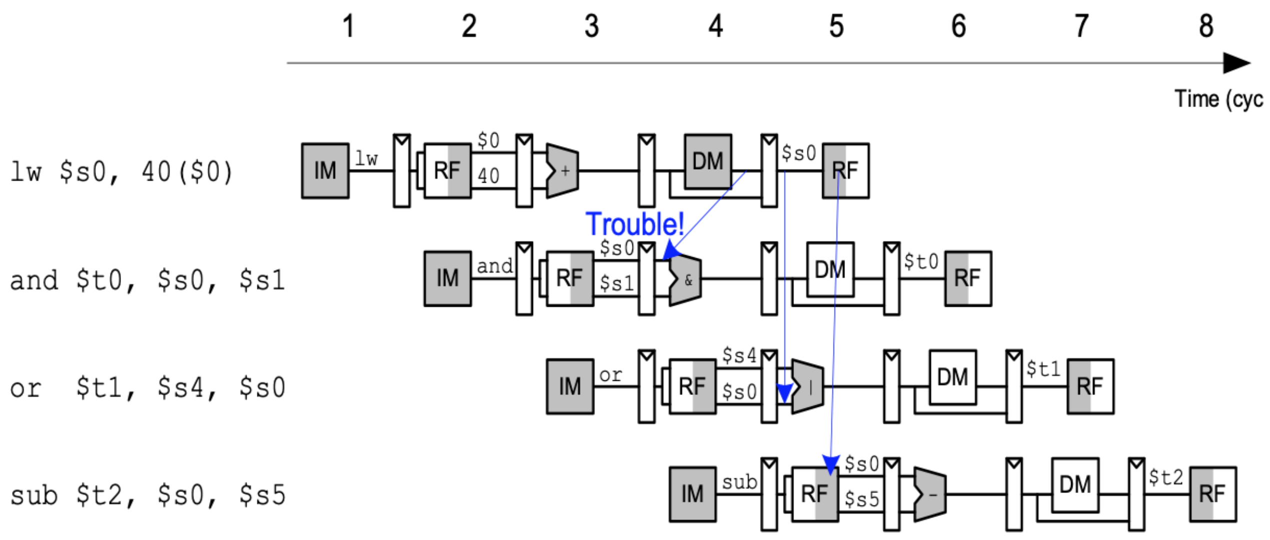
Why Does It Occur?

All instructions that read data from memory like lw require to finish memory stage. Therefore, next instructions which depend on previous loading instruction cannot fecth the right data which should have already been loaded. By this, next instructions will not calcute the right data in execute stage because memory stage is later than execute.

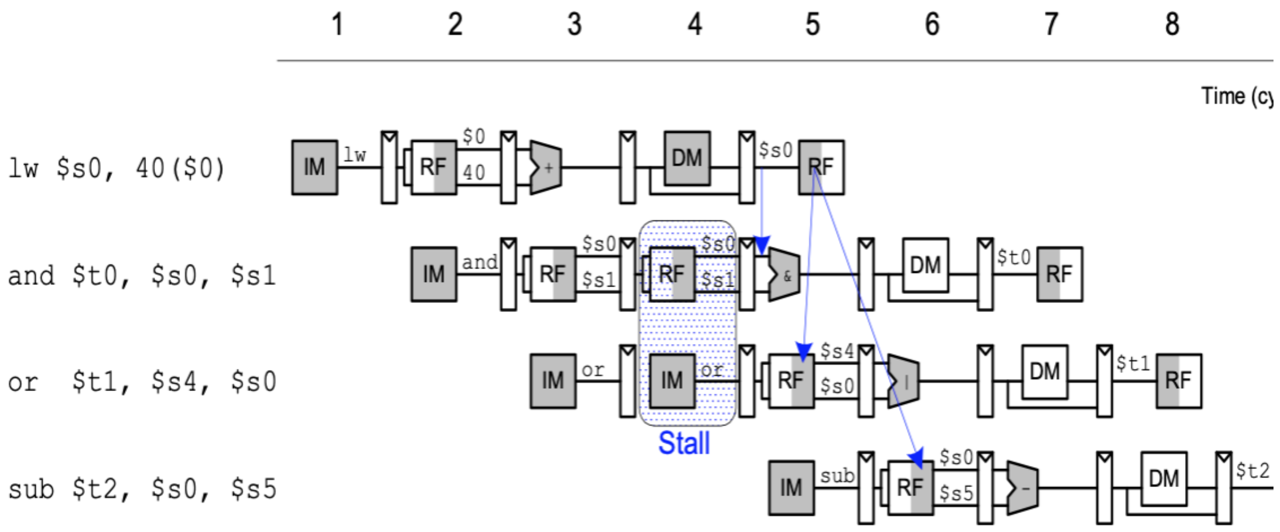
How Can It Be Solved?

Unlike previous hazard, forwarding cannot be a solution to this hazard since the result of memory stage comes after the execute stage. That's why result cannot be computed with forwarding correctly. The correct approach to fix this hazard is stalling. By stalling, execute stage will be delayed. Thus, the data in the memory can be read and used by subsequent instructions. Furthermore, nop instruction can be used to solve this problem but it will be an unefficient implementation. Here are some pictures to explain.

1-) Forwarding Is Not a Solution



2-) Stalling As a Solution



3-) Load-store Hazard (Data Hazard)

When Does It Occur?

Load-use hazard is similar to load-use hazard. It occurs when lw instruction is not finished its work and next instruction is store word instruction.

1-) lw $t0, 4($t1)
2-) sw $t0, 2($t2)

In this example, the value of $t0 is not computed yet while next instruction tries to save the value of $t0 to another address.

Why Does It Occur?

The reason of the problem is same with the load-use as it is explained but in the load-store. To summarize, instructions that read data from memory like lw require to finish memory stage. Therefore, sw instruction cannot get the correct value of rt register. Fetching the data from memory part of previous instruction countinues while subsequent sw instruction wants data from rt register.

How Can It Be Solved?

Similarly, forwarding cannot be a solution to this hazard. Using nop instruction can be a solution but it will not be efficient. Stalling method is a efficient method to fix this hazard. The implementation of this will be following, first flush the execute stage and then stall fecth and decode stages.

2-) Branch Hazard (Control Hazard)

When Does It Occur?

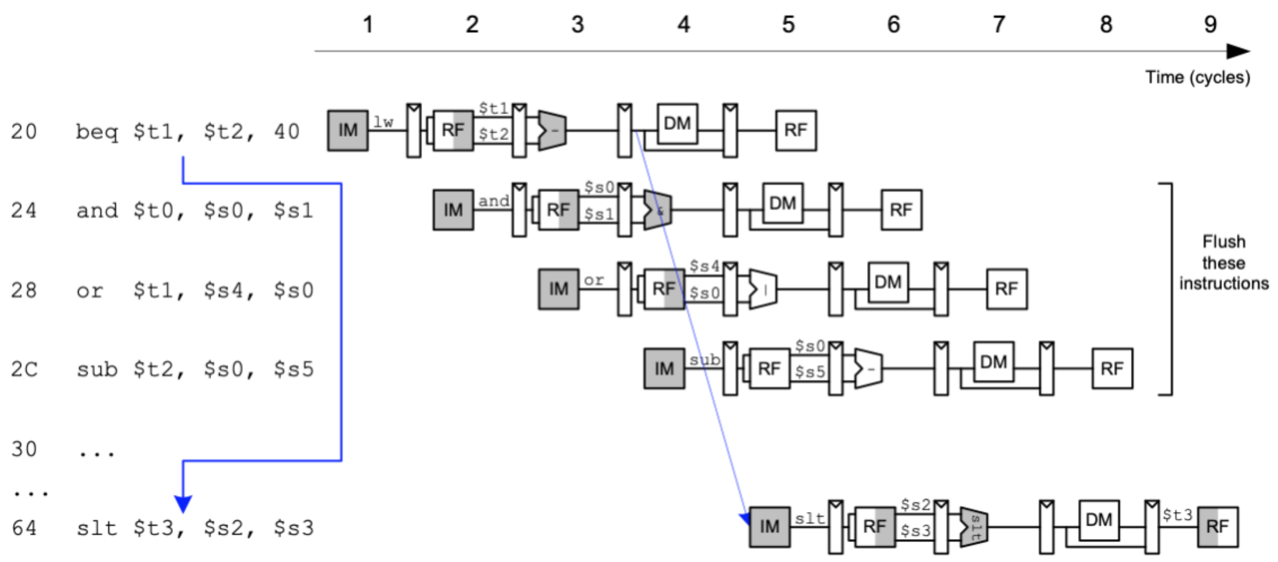
Branch hazard may occur when there is a subsequent instruction after branch equation.

1-) beq $t0, $t1, Next
2-) addi $t0, $t1, 1
3-) Next: slt $t2, $t0, $t1

In this example, it can be said that addi instruction will be executed without concerning there is a beq instruction.

Why Does It Occur?

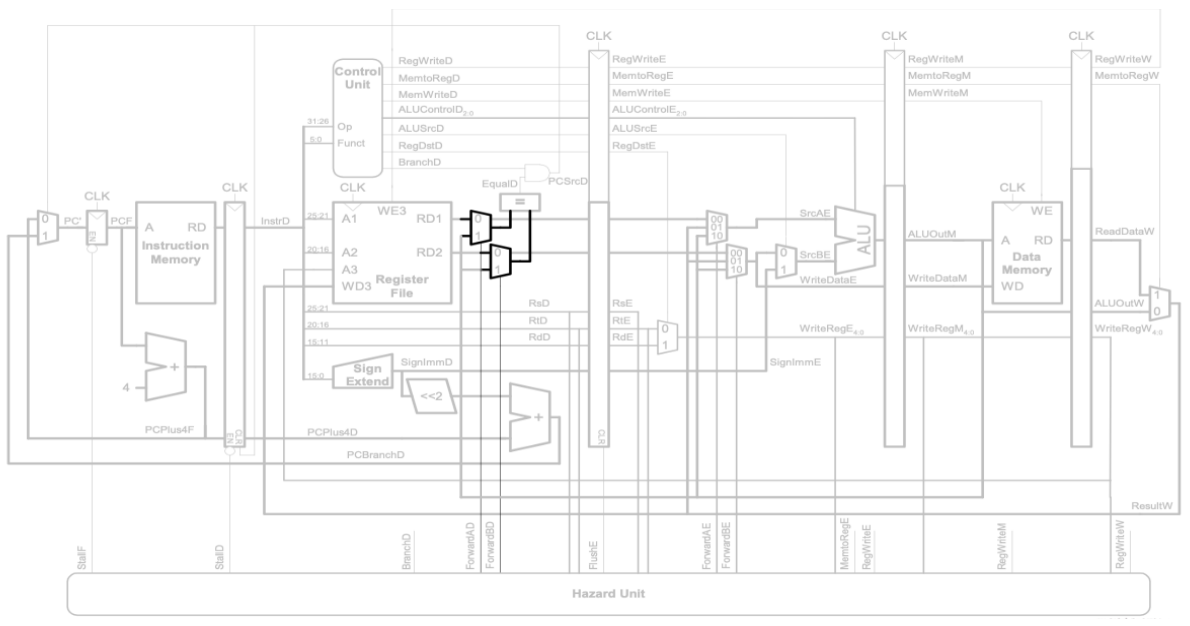
Branch instruction will calculate where to go or the value of program counter at the memory stage. Therefore, the subsequent instruction will reach the execute stage untill branch decides the program counter. That will be a problem for program. To exemplify hazard, the picture will sufficient.



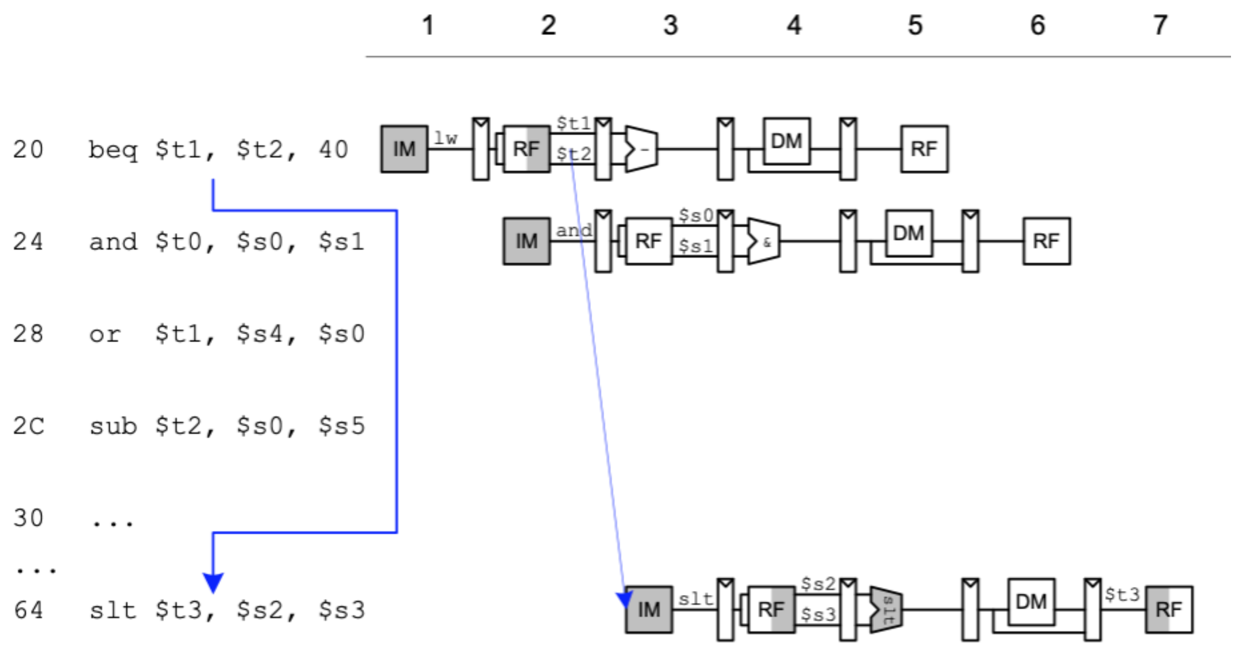
How Can It Be Solved?

Forwarding is not a solution for this hazard. To solve this, stalling for 3 times will be an adequate method or instructions can be flushed. Another solution is to add some hardware to get rid of this problem. Again, nop operations will fix the problem but it is an inefficient approach to do. Figure below shows how hardware implementation should be done.

1-) Hardware implementation



2-) Result Of The Implementation



```
20   beq $t1, $t2, 40
24   and $t0, $s0, $s1
28   or  $t1, $s4, $s0
2C   sub $t2, $s0, $s5
30   ...
...
64   slt $t3, $s2, $s3
```

D-) Logic Equations For Forwarding

1-) This is for ForwardingAE

```
if   ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM)
        then    ForwardAE = 10
else
    if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW)
        then    ForwardAE = 01
     else       ForwardAE = 00
```

2-) ForwardingBE is the same with upper equations. Just ForwardingAE needs to be replaced with Forwarding.

3-) This is for Control Forwarding

```
ForwardAD = (rsD !=0) AND (rsD == WriteRegM) AND RegWriteM
ForwardBD = (rtD !=0) AND (rtD == WriteRegM) AND RegWriteM
```

Logic Equations For Stalling

1-) This is for Load Stalling

```
lwstall =
   ((rsD==rtE) OR (rtD==rtE)) AND MemtoRegE

StallF = StallD = FlushE = lwstall
```

2-) This is for Control Stalling

```
branchstall = BranchD AND RegWriteE AND
                 (WriteRegE == rsD OR WriteRegE == rtD)
              OR
              BranchD AND MemtoRegM AND
                 (WriteRegM == rsD OR WriteRegM == rtD)

StallF = StallD = FlushE = (lwstall OR branchstall)
```

E-) Test Programs

The registers with the same color represents hazards.

1-)This program has compute-use hazard

Assembly Code **Address-MachineCode(hex)**

```
add $s0, $s1, $t1         8'h00: 0x02298020
sub  $s2, $s0, $s3        8'h04: 0x02139022
add  $t1, $t0, $s2        8'h08: 0x01124820
```

2-)This program has load-use hazard

Assembly Code **Address-MachineCode(hex)**

```
addi $t1, $zero, 50       8'h00: 0x20090032
nop                       8'h04: 0x00000000
nop                       8'h08: 0x00000000
lw    $t0, 3($t1)         8'h0c: 0x8d280003
and  $t2, $t0, $t3        8'h10: 0x010b5024
or    $t4, $s0, $t0       8'h14: 0x02086025
```

3-)This program has load-store hazard

Assembly Code **Address-MachineCode(hex)**

```
ori   $t0, $t1, 23     8'h00: 0x35280017
and  $t1, $t2, $t3     8'h04: 0x014b4824
nop                    8'h08: 0x00000000
lw    $s0, 5($t0)      8'h0c: 0x8d100005
sw    $s0, 3($t1)      8'h10: 0xad300003
```

4-)This program has branch hazard

Assembly Code	**Address-MachineCode(hex)**
li $s0, 5	8’00: 0x34100005
li $s1, 5	8’04: 0x34110005
beq $s0, $s1, Next	8’08: 0x1211fffd
*addi $t0, $zero, 39	8’0c: 0x20080027
*add$t2, $s0, $zero	8’10: 0x02005020
*and$t1, $s1, $s2	8’14: 0x02324824
add $t3, $t4, $t3	8’18: 0x018b5820
Next:	8’1c:
slt $t0, $s0, $s1	8’20: 0x0211402a

The instructions begin with the * will be fetched which cause hazards.

5-)This program has no hazards

Assembly Code	**Address-MachineCode(hex)**
addi $t0, $zero, 34	8’00: 0x20080022
addi $t1, $zero, 21	8’04: 0x20090015
addi $t2, $zero, 2	8’08: 0x200a0002
beq $zero, $t0, next	8’0c: 0x10000002
sub $t3, $t0, $t1	8’10: 0x01095822
next:	8’14:
and $s0, $t0, $t1	8’18: 0x01098024
or $s1, $t1, $t2	8’1c: 0x012a8825
lw $s2, 7($t0)	8’20: 0x8d120007
add $s0, $s0, $t3	8’24: 0x020b8020
nop	8’28: 0x00000000
sw $s2, 3($t2)	8’2c: 0xad520003
add $s1, $s2, $zero	8’30: 0x02408820