**BILKENT UNIVERSITY**

**COMPUTER SCIENCE**

**CS224**

# Preliminary Design Report - Lab5

**Section 2**

**İrem Seven**

**21704269**

**4.12.19**

## a) Hazard Types

### 1) Compute – Use Hazard

This hazard type is a data hazard. It occurs when a instruction uses a register that has been used by the previous register. Mening that if the previous instructor writes value to a register, next instructor will read the same register wrong since it is not written yet. As an example:

```
add $t0,$t1,$t2
sub $t3,$t4,$t0
```

Here hazard will occur because of register t0.

### 2) Load-Use Hazard

This hazard type is a data hazard. It occurs when an instruction as lw loads a value into a register in memory stage and the instruction after lw needs that value to use. As an example:

```
lw $t0,4($t1)
add $s0,$t0,$t2
```

Here hazard will occur because of t0 register.

### 3) Load-Store Hazard

This hazard type is a data hazard. It occurs when a data that is written in memory also will be used/read right after in the memory stage. It happens since the next instruction wants to use value in memory stage which should have already been calculated by the previous instructions writeback stage. As an example:

```
lw $t0,16($s0)
sw $t0,4($s1)
```

Here hazard will occur because of t0 register.

### 4) Branch Hazard

This hazard type is a control hazard. It occurs since pc cannot be determined, fetching, yet because whether the branch will be taken or not is not known yet. Reason is that it is determined in the fourth cycle of the clock, from the beginning. Any example can be given when there is a branch instruction followed by any instruction. An example is that:

```
beq $t0,$t1,take
addi $t3,$4,5
sub $t6,$s0,$s1
take:
```

## b) Solutions for Hazards

### 1) Control – Use Hazard

It occurs since the writeback stage of add instruction happens after the sub instructions execute stage. This problem can be solved by stalling 2 cycles in the given pipeline. However, a more efficient solution is that forwarding. Data can be forwarded from latch after alu to just before the alu.

### 2) Load - Use Hazard

This hazard can be solved by stalling followed by forwarding. Only forwarding cannot solve this problem. Thus, one stalling is needed first. After that forwarding can be made from the latch just before the writeback stage into the just before the alu.

### 3) Load-Store Hazard

This hazard can be solved directly forwarding the data from the latch which is right before the writeback stage to right before the data memory stage. Meaning that requires forwarding from lw’s writeback stage to the sw’s memory stage.

### 4) Branch Hazard

This hazard can be solved by flushing the instructions when the branch is taken. Another solution is that taking the branch earlier that fetching can be determined earlier.

## d) Logic Equations for Hazards

### 1) Stalling

```
lwstall = ((rsD == rtE) || (rtD == rtE)) && MemtoRegE

        StallF = StallD = FlushE = lwstall
```

### 2) Forwarding

#Forwarding logic for *ForwardBE* same, but replace *rsE* with *rtE.*

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

### 3) Branch Solution

```
ForwardAD = (rsD != 0) && (rsD == WriteRegM) && RegWriteM

ForwardBD = (rtD != 0) && (rtD == WriteRegM) && RegWriteM


branchstall = BranchD && RegWriteE && (WriteRegE == rsD || WriteRegE == rtD)
                   || BrachD && MemtoRegM && (WriteRegM == rsD || WriteRegM == rtD)
        StallF = StallD = FlushE = (lwstall || branchstall)
```

## e) Test Programs

### 1) Hazard Free

| Assembly | Machine Code |
|---|---|
| addi $t0,$0,1 | 0x20080001 |
| addi $t1,$0,2 | 0x20090002 |
| addi $t2,$0,3 | 0x200a0003 |
| addi $t3,$0,4 | 0x200b0004 |
| sw $t0,4($sp) | 0xafa80004 |
| addi $t0,$0,5 | 0x20080005 |
| sub $t5,$t1,$t2 | 0x012a6822 |
| addi $s0,$s0,0 | 0x22100000 |
| lw $s1,0($sp) | 0x8fb10000 |

### 2) Compute – Use Hazard

| Assembly | Machine Code |
|---|---|
| addi $t0,$0,10 | 0x2008000a |
| addi $t1,$0,5 | 0x20090005 |
| addi $t2,$t0,20 | 0x210a0014 |
| add $t3,$t2,$t0 | 0x01485820 |
| addi $t4,$t3,15 | 0x216c000f |
| sub $t2,$t3,$t4 | 0x016c5022 |

### 3) Load – Use Hazard

| Assembly | Machine Code |
|---|---|
| addi $t0,$0,1 | 0x20080001 |
| addi $t1,$0,2 | 0x20090002 |
| addi $t2,$0,3 | 0x200a0003 |
| addi $t3,$0,4 | 0x200b0004 |
| lw $t4,0($sp) | 0x8fac0000 |
| add $s0,$t4,$t0 | 0x01888020 |

**4) Load – Store Hazard**

| **Assembly** | **Machine Code** |
|---|---|
| addi $sp,$sp,-8 | 0x23bdfff8 |
| addi $t0,$0,1 | 0x20080001 |
| addi $t1,$0,2 | 0x20090002 |
| addi $t2,$0,3 | 0x200a0003 |
| lw $s0,0($sp) | 0x8fb0000 |
| sw $s0,4($sp) | 0xafb00004 |

**5) Brach Hazard**

| **Assembly** | **Machine Code** |
|---|---|
| addi $t0,$0,2 | 0x20080002 |
| addi $t1,$0,2 | 0x20090002 |
| addi $s0,$0,0 | 0x20100000 |
| addi $s0,$0,0 | 0x20100000 |
| addi $s0,$0,0 | 0x20100000 |
| beq $t0,$t1,equal | 0x11090003 |
| addi $t0,$0,3 | 0x20080003 |
| addi $t1,$0,4 | 0x20090004 |
| addi $s0,$0,-1 | 0x2010ffff |
| equal: | |