2019-2020 SPRING SEMESTER
CS 224 – DIGITAL DESIGN
LAB 5 – 12.05.2020

SECTION: 3
NAME: DENIZHAN
SURNAME: KEMEROZ
STUDENT ID: 21703471

b) Hazards that can occur in this pipeline

1. Compute-use hazard which is a data hazard is caused by getting data from **execute stage** to use in the other instructor's stage that it is required. If that is not fixed, in the **decode stage** the required data cannot be used and in the other stages wrong data will be used.
2. Load-use hazard which is a data hazard is caused by getting data from **memory stage** to use in the other instructor's stage that it is required. If that is not fixed, in the **decode stage** the required data cannot be used.
3. Load-store hazard which is a data hazard is caused by getting data from **memory stage** to use in the other instructor that stores it. If that is not fixed, in the decode stage the required data cannot be used and the loaded data cannot be stored in the **memory stage**.
4. Branch hazard which is a control hazard is caused by getting data from **decode stage** to decide which instruction is going to be used (which instruction will be **fetched**).
5. Compute-load which is a data hazard is caused by getting storing address from **execute stage** to use in the other instructor that load in that address. If that is not fixed, in the decode stage the required address cannot be used and the value cannot be loaded from that address in the **memory stage.**
6. Compute-store which is a data hazard is caused by getting data from **execute stage** to use in the other instructor that stores it. If that is not fixed, in the decode stage the required data cannot be used and the loaded data cannot be stored in the **memory stage**.

c) Solution

1. When there is a situation like using a register which is the solution of the comparison before the using instruction, the using instruction can be stalled the times that it requires, and after using ALU to compute, the value can be forwarded to the next instruction's Decode or Execute part (depends on the action) so that it can reduce the requirement time of stalling or even can work by itself if the instruction is not immediately after the instruction that computes.
2. When there is a situation like using a register which is the solution of the loading from the memory before the using instruction, the using instruction can be stalled the times that it requires, and after getting value from Data Memory, the value can be forwarded to the next instruction's Decode or Execute part (depend on the action) so that it can reduce the requirement time of stalling.
3. When there is a situation like using a register which is the solution of the loading from the memory before the instruction that stores that register, the storing instruction can be stalled the times that it requires, and after getting value from Data Memory, the value can be forwarded to the next instruction's Memory Stage's WriteDataM so that it can reduce the requirement time of stalling.
4. When there is a situation like using a register which to decide which instruction will be fetched, the next instructions can be stalled one the time because of the resolution, or the next instructions can be flushed till the new destination that branch leads.
5. When there is a situation like using a register which is the solution of the comparison before the loading data from it, the using instruction can be stalled the times that it requires, and after using ALU to compute, the value can be forwarded to the next instruction's Memory

part to ALUOutM so that it can reduce the requirement time of stalling or even can work by itself if the instruction is not immediately after the instruction that computes.

6. When there is a situation like using a register which is the solution of the computation before the instruction that stores that register, the storing instruction can be stalled the times that it requires, and after using ALU to compute, the value can be forwarded to the next instruction's Memory Stage's WriteDataM so that it can reduce the requirement time of stalling.

d) The logic equations

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

if ((rtE != 0) AND (rtE == WriteRegM) AND RegWriteM) then

ForwardBE = 10

else if ((rtE != 0) AND (rtE == WriteRegW) AND RegWriteW) then

ForwardBE = 01

else ForwardBE = 00

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

ForwardAD = (rsD != 0) AND (rsD == WriteRegM) AND RegWriteM

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

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) Small test programs

1. Compute-use

```
.text
main:
   addi $t0, $0, 3
   addi $t1, $0, 1
   add $s0, $s0, $0
   add $s0, $s0, $0
   add $s0, $s0, $0
   add $s0, $s0, $0
   add $s0, $s0, $0
   add $t2, $t1, $t0       0x01285020
   add $t0, $t1, $t2       0x012a4020
```

```
8'h00: instr = 32'h20080003;
8'h04: instr = 32'h20090003;
8'h08: instr = 32'h02008020;
8'h0c: instr = 32'h02008820;
8'h10: instr = 32'h02009020;
8'h14: instr = 32'h02009820;
8'h18: instr = 32'h0200a020;
8'h1c: instr = 32'h01285020;
8'h20: instr = 32'h012a4020;
```

2. Load-use

```
        .text
main:
         addi $t0, $0, 3
         addi $t1, $0, 1
        la $t2, array
        sw $t0, 0($t2)
        add $s0, $zero, $0
```

```
        add $s0, $zero, $0
        add $s0, $zero, $0
        add $s0, $zero, $0
        add $s0, $zero, $0
        lw $t1, 0($t2)          0x8d490000
        add $t0, $t1, 3         0x21280003
        .data
array: .space 80 #,2,3,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
```

```
8'h00: instr = 32'h21080003;
8'h04: instr = 32'h21290001;
8'h08: instr = 32'h3c011001;
8'h0c: instr = 32'h342a0000;
8'h10: instr = 32'had480000;
8'h14: instr = 32'h00008020;
8'h18: instr = 32'h00008020;
8'h1c: instr = 32'h00008020;
8'h20: instr = 32'h00008020;
8'h24: instr = 32'h8d490000;
8'h28: instr = 32'h21280003;
```

3. Load-store

```
        .text
main:
        addi $t0, $0, 3
        addi $t1, $0, 1
        la $t2, array
        sw $t0, 0($t2)
        add $s0, $zero, $0
        add $s0, $zero, $0
        add $s0, $zero, $0
        add $s0, $zero, $0
```

```
        add $s0, $zero, $0
        lw $t1, 0($t2)          0x8d490000
        sw $t0, 4($t2)          0xad480004

        .data
array: .space 80 #,2,3,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20

8'h00: instr = 32'h21080003;
8'h04: instr = 32'h21290001;
8'h08: instr = 32'h3c011001;
8'h0c: instr = 32'h342a0000;
8'h10: instr = 32'had480000;
8'h14: instr = 32'h00008020;
8'h18: instr = 32'h00008020;
8'h1c: instr = 32'h00008020;
8'h20: instr = 32'h00008020;
8'h24: instr = 32'h8d490000;
8'h28: instr = 32'h ad480004;
```

4. NO HAZARD

```
        .text
main:
        addi $t0, $0, 3
        addi $t1, $0, 1
        addi $t4, $0, 1
        addi $t6, $0, 4
        la $t2, array
        la $t3, array2
        sw $t0, 0($t2)
        sw $t1, 0($t3)
        add $s0, $zero, $0
```

```
        add $s1, $zero, $0
        add $s2, $zero, $0
        add $s3, $zero, $0
        add $s4, $zero, $0
        lw $t1, 0($t2)          0x8d490000
        sw $t2, 0($t3)          0xad6a0000
        add $s0, $zero, $0
        add $s1, $zero, $0
        add $s2, $zero, $0
        add $s3, $zero, $0
        add $s4, $zero, $0
        lw $t0, 0($t3)          0x8d680000
        add $t1, $t1, $t4       0x012C4820
        add $t5, $t6, $0        0x01C06820

        .data
array: .space 80 #,2,3,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
array2: .space 80 #,2,3,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
```