REPORT – Assignment 9

In this assignment we use forwarding to prevent hazards and avoid stalling as much as possible. Forwarding is done by using the data of a later stage's latch instead of a possibly hazardous data available in this stage's latch.

Differentiating register updates:

Also in this assignment we need to differentiate between the two types updated that are made to registers:

1. Those that involve ALU operations: In this type of instructions we get the data to be written to the registers at the end of th.e EXE stage from where it can be forwarded.
2. Those that involve load from memory: The data to be written is available in the MEM stage when memory access is performed so in this case we have data 1 cycle later than the previous.

Differentiating register reads:

1. Lets call a register "to be read" when it is to be read for an ALU operation or comparison i.e. its value is required in the EXE stage.
2. Lets call a register "to be stored" when it is to be read for storing in the memory i.e. its value is requierd in the MEM stage.

Post-WB latch:

If a register was updated in previous to previous instruction and its data is to be stored in memory in this instruction then we need a forward to MEM stage but in this cycle no latch has that instruction's data so we need a new latch at the end of WB stage from where forward can be made to this stage.

Preventing hazards using forwarding

In the ID stage if we detect that there the register to be read was updated in the previous cycle or the previous to previous cycle then we update the **forwrding control signals** accordingly.
Lets call a register written when it is updated by an ALU operation type instruction and loaded when it is updated by lw instruction.

1. When a register to be **read** is **written** in the previous cycle:
   We forward from **EX-MEM** latch to **EX stage**
2. When a register to be **read** is **written** in the previous to previous cycle:
   We forward from **MEM-WB** latch to **EX stage**
3. When a register to be **read** is **loaded** in the previous cycle:
   We **stall** for 1 cycle and forward from **MEM-WB** latch to **EX stage**
4. When a register to be **read** is **loaded** in the previous to previous cycle:
   We forward from **MEM-WB** latch to **EX stage**
5. When a register to be **stored** is updated in the previous cycle:
   We forward from **MEM-WB** latch to **MEM stage**
6. When a register to be **stored** is updated in the previous to previous cycle:
   We forward from **post-WB** latch to **MEM stage**

Testing:

TestCases ($a1 <= 1 in all test cases)

Register written in previous cycle is to be read: (Forward from EX-MEM latch to EX stage):

```
add $a0 $zero $zero
add $a1 $a0 $a0
```

Register written in previous to previous cycle is to be read: (Forward from MEM-WB latch to EX stage):

```
add $a0 $zero $zero
sub $a1 $zero $zero
add $a1 $a0 $a0
```

Register loaded in previous cycle is to be read: (Stall + Forward from MEM-WB latch to EX stage):

```
sub $sp $sp $a1
lw $a0 0 $sp
add $a1 $a0 $a0
```

Register loaded in previous cycle is to be read: (Forward from MEM-WB latch to EX stage):

```
sub $sp $sp $a1
lw $a0 0 $sp
add $a2 $a2 $a2
add $a1 $a0 $a0
```

Register updated(written/loaded) in previous cycle is to be stored:(Forward from MEM-WB latch to MEM stage):

```
sub $sp $sp $a1
add $a0 $zero $zero
sw $a0 0 $sp
```

Register updated(written/loaded) in previous to previous cycle is to be stored:(Forward from post-WB latch to MEM stage):

```
add $a0 $zero $zero
sub $sp $sp $a1
sw $a0 0 $sp
```

Testcases for comparison with assignment 8:

```
    reg[reg_stoi("$a1")] = 1;
    reg[reg_stoi("$at")] = 1;
    reg[reg_stoi("$v1")] = 10;
    reg[reg_stoi("$a3")] = 10;
```

Finding sum of first $a3 numbers:

```
add $a3 $a3 $a1
add $a3 $a3 $a1
add $a3 $a3 $a1
add $a3 $a3 $a1
jal findsum
j exit
findsum:
sub $sp $sp $a1
sub $sp $sp $a1
sw $a0 0 $sp
sw $a3 1 $sp
```

```
add $a0 $zero $zero
for:
add $a0 $a0 $a3
sub $a3 $a3 $a1
bne $a3 $zero for
add $v0 $a0 $zero
lw $a3 1 $sp
lw $a0 0 $sp
add $sp $sp $a1
add $sp $sp $a1
jr $ra
exit:
```

Result for assignment 8:
Toal Cycles: 119
No of instruction executed: 59
Average instruction per cycle: **0.495798**

Result for assgnment 9:
Toal Cycles: 77
No of instruction executed: 59
Average instruction per cycle: **0.766234**

****

```
//Storing first $v1 fibonacci numbers in the momory from (4095..4095 - $v0 + 1)
$a1 <= 1, $a1 <= 1

label1:
sub $sp $sp $at
sw $a0 0 $sp
sub $sp $sp $at
sw $a1 0 $sp
add $v0 $v0 $at
label2:
add $a2 $a1 $a0
sub $sp $sp $at
sw $a2 0 $sp
lw $a0 1 $sp
lw $a1 0 $sp
add $v0 $v0 $at
beq $v0 $v1 label3
j label2
label3:
```

Result for assignment 8:
Toal Cycles: 129
No of instruction executed: 76
Average instruction per cycle: **0.589147**

Result for assignment 9:
Toal Cycles: 89

No of instruction executed: 76
Average instruction per cycle: **0.853933**