Computer Science M151B, Homework 5

Michael Wu
UID: 404751542

May 7th, 2018

Problem 1

Yes, the `PCSrc` signal could be computed in the `EX` stage, as the zero of the ALU output and the `Branch` signal are both available in `EX`. An advantage of making this change would be to reduce the branch penalty from three cycles to two cycles, since we can update the program counter one cycle earlier. However, since this adds an `and` gate to the output of the ALU, this could potentially lead to a higher cycle time. The book shows a better implementation where the branch computed in the `ID` stage, reducing the branch penalty to one cycle, which is why this change is not used.

Problem 2

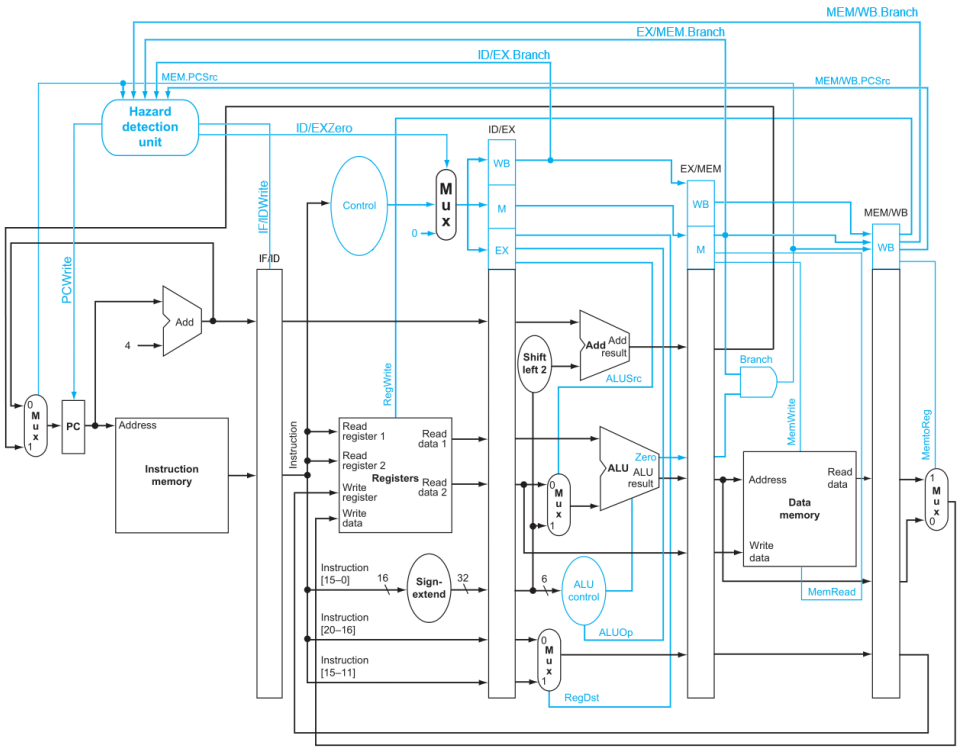
a) The value of `PCSrc` is 0 when this instruction is in the `MEM` stage. This is because there is no branching going on, so the program counter should update normally.

b) The value of `PCSrc` is 1 when this instruction is in the `MEM` stage. This is because this branch instruction checks if register `$5` is equal to itself, which must be true. So the branch will be taken and the program counter will be updated to the address given in the branch instruction.

Problem 3

This series of instructions will execute over 9 cycles. On the fifth cycle, the first `add` instruction is in the `WB` stage, so register `$2` is written. Additionally, the third `add` instruction is in the `ID` stage, so the registers `$6` and `$1` are being read.

Problem 4



The modified hazard detection unit is shown above. The goal of this modification is to always stall for three cycles after every branch. I have added the `Branch` control signal to the `MEM/WB` registers, so that a signal is available to determine if a branch instruction is in the `WB` stage. I have also added the `MEM/WB.PCSrc` signal to aid in fetching a new instruction if

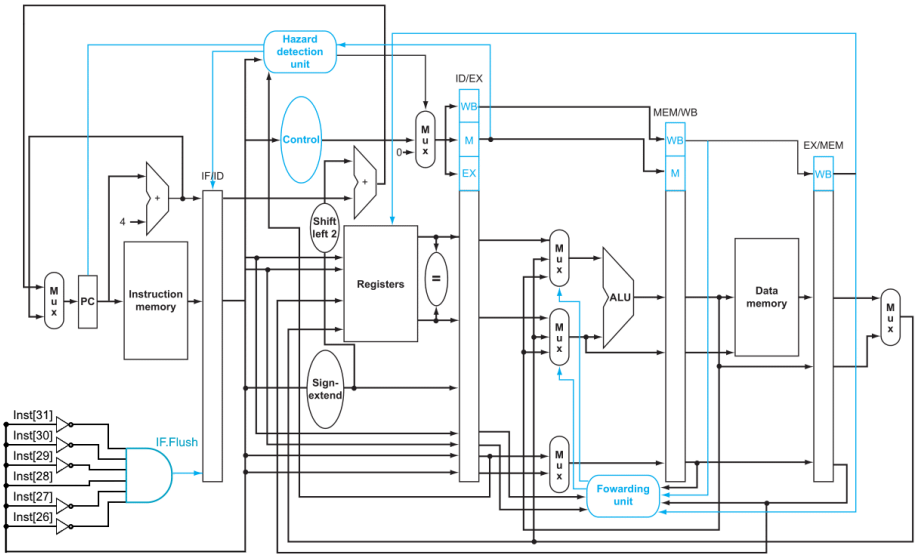
the branch is taken. The hazard detection unit stalls by injecting 0 into the `ID/EX` registers. It also sets `IF/IDWrite` and `PCWrite` to 0 to prevent a new instruction from being loaded.

The logic for the outputs of the hazard detection unit is shown below.

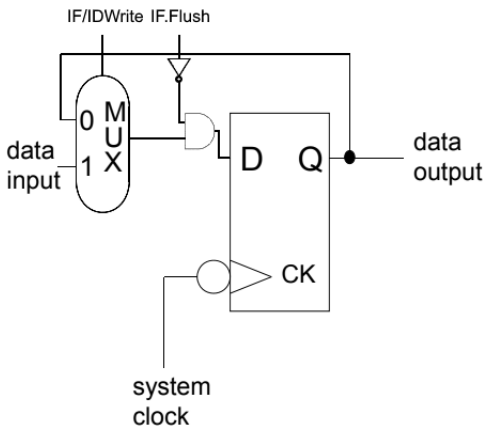
```
if (ID/EX.Branch or EX/MEM.Branch or MEM/WB.Branch)
  then ID/EXZero = 1 else ID/EXZero = 0

if (MEM/WB.PCSrc != 1 and
  (ID/EX.Branch or EX/MEM.Branch or MEM/WB.Branch))
    then IF/IDWrite = 0 else IF/IDWrite = 1

if (MEM.PCSrc != 1 and MEM/WB.PCSrc != 1 and
  (ID/EX.Branch or EX/MEM.Branch or MEM/WB.Branch))
    then PCWrite = 0 else PCWrite = 1
```

Note that I set `PCWrite = 1` when `MEM.PCSrc = 1` to allow for the branch to update the program counter if it is taken. I then set `PCWrite = 1` and `IF/IDWrite = 1` when `MEM/WB.PCSrc = 1` to allow for the instruction at the branch target address to be read and the program counter to be updated in the next cycle.

Problem 5



a) Since the `IF.Flush` signal is asserted whenever a branch instruction occurs, this corresponds to asserting it when an opcode of 4 is read. So this circuit implements that logic.



b) The `IF.Flush` signal will be used to clear out the register by using a gate as shown in the figure above. The gate is placed after the multiplexer controlled by `IF/IDWrite`, so that `IF.Flush` has precedence over `IF/IDWrite`.

Problem 6

The updated code will be

```
addi $r6,  $r6,  12
sw   $r16, $r6
addi $r6,  $r6,  -4
lw   $r16, $r6
beq  $r5,  $r4,  Label
add  $r5,  $r1,  $r4
slt  $r5,  $r15, $r4
```

which increases the number of instructions to 7. Since the length of the pipeline decreases by one cycle, this will require 10 cycles to execute. Previously with the 5 stage pipeline, there were 5 instructions which required 9 cycles to execute. So our speedup is

$$\frac{9}{10} = 0.9$$

which indicates that we actually get worse performance by making this change.

Problem 7

With stall-on-branch and the branch outcome being determined in the `EX` stage, the code will take 11 cycles to execute. Determining the branch outcome in the `ID` stage reduced the branch penalty by one, which makes the code take 10 cycles to execute. Since we assume that the latencies of each stage are as shown in the given table, our clock cycles will be the same for either implementation. Thus determining the branch outcome in the `ID` stage gives a speedup of

$$\frac{11}{10} = 1.1$$

relative to determining the branch outcome in the `EX` stage.

Problem 8

The opcode must contain at least 8 bits, and each register field must contain at least 6 bits. So that leaves at most 12 bits for the immediate field. Thus the maximum range of the immediate value is from -2048 to 2047 in decimal.

Problem 9

Implementing this code in MIPS using indexed addressing would yield the following code.

```
add  $t0, $zero, $zero
addi $t1, $zero, 10
sll  $t2, $t0,   2
add  $t2, $t2,   $a1
sll  $t3, $t0,   3
add  $t3, $t3,   $a0
lw   $t4, 0($t2)
add  $t4, $t4,   $t0
sw   $t4, 0($t3)
addi $t0, $t0,   1
bne  $t0, $t1,   -9
```

Implementing this code in PowerPC using update addressing would yield the following code.

```
      xor   r1, r1, r1
      addi  a1, a1, -4
      addi  a0, a0, -8
loop: lwzu  r2, 4(a1)
      add   r2, r2, r1
      stwu  r2, 8(a0)
      addi  r1, r1, 1
      cmpwi r1, 10
      bne   loop
```