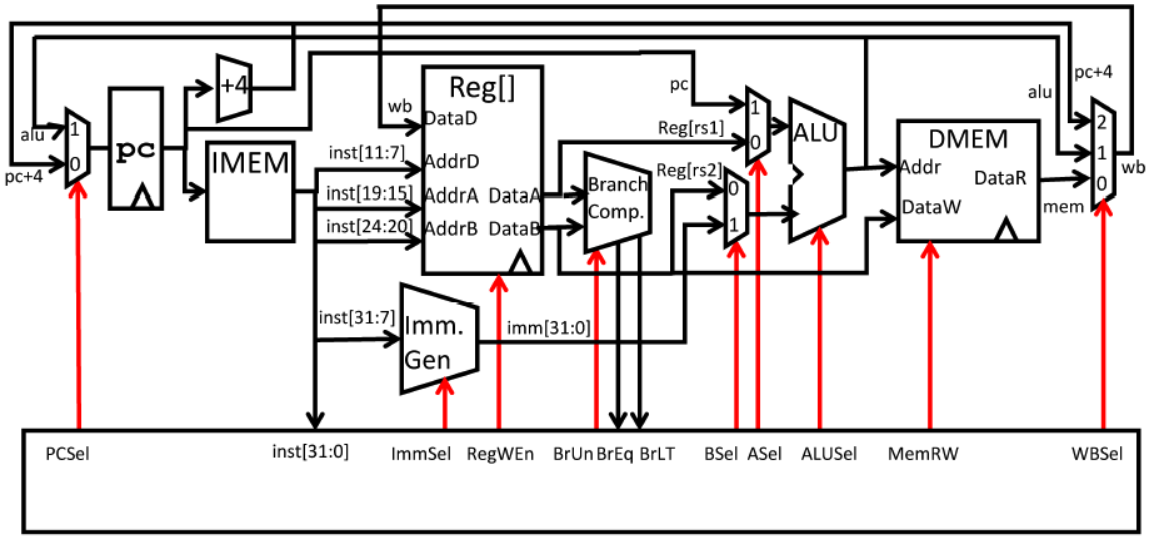
EECS 151/251A Homework 5

Due Friday, Oct 16th, 2020

Problem 1: Control Logic [12 pts]

In the fabrication of any digital circuit, there may be manufacturing defects. One type of defect involves a signal being shorted to GND or VDD (stuck-at-zero or stuck-at-one). Consider the following datapath and control signals, please specify the RISC-V instructions that will no longer work for the following stuck signals.



(a) (4 pts) BSel is stuck-at-one

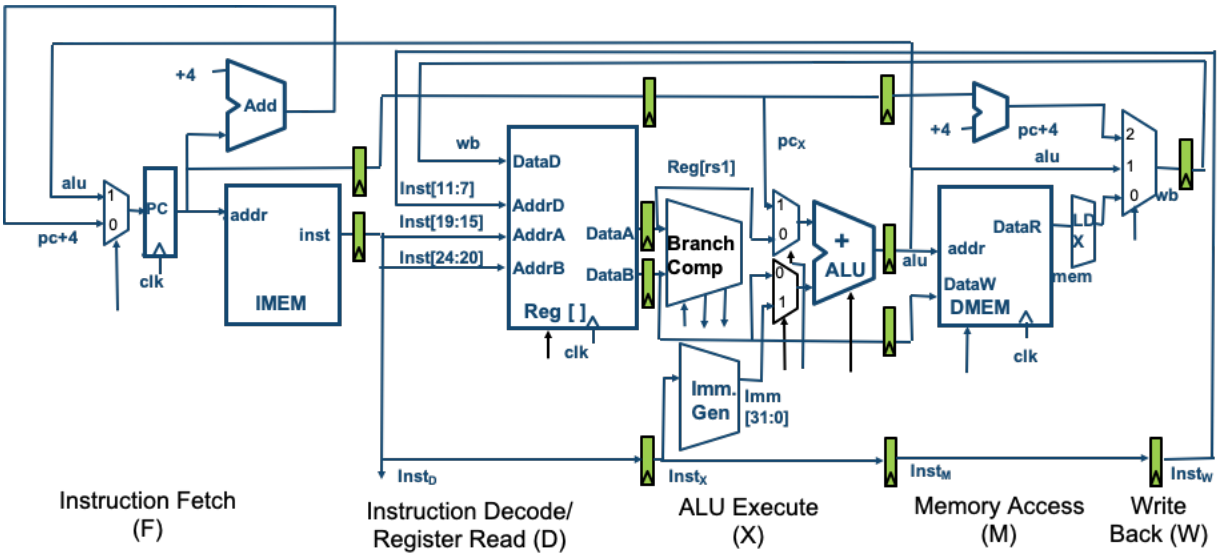
(b) (4 pts) ASel is stuck-at-zero

(c) (4 pts) WBSel[1] is stuck-at-zero

Problem 2: Hazard [10 pts]

Assume we have a 5 stage pipelined processor with the following stages:

1. Instruction fetch (`IF`)
2. Decode (`D`)
3. Execute (`EX`)
4. Memory Access (`M`)
5. Writeback (`WB`)



(a) (8 pts) Assume that data forwarding isn't implemented in this datapath (use stall), and branch is always not taken (take `PC+4`). How many cycles will the following assembly take to execute? Please make a timing table showing the stage of each cycle, which kind(s) of hazard do you observe? (hint: The column should be: `Cycle | IF | D | EX | M | WB`. At the final cycle, `nop` should be in the last stage)

```
0x00:   li   x1, 3
0x04:   li   x2, 5
0x08:   xor  x3, x1, x2
0x0c:   add  x4, x1, x2
0x10:   slli x5, x3, 1
0x14:   blt  x4, x5, 0x08
0x18:   addi x1, x4, 4
0x1c:   beq  x5, x1, 0x08
0x20:   addi x1, x4, 4
0x24:   nop
```

(b) (2 pts) What is the CPI of this process for this block of code?

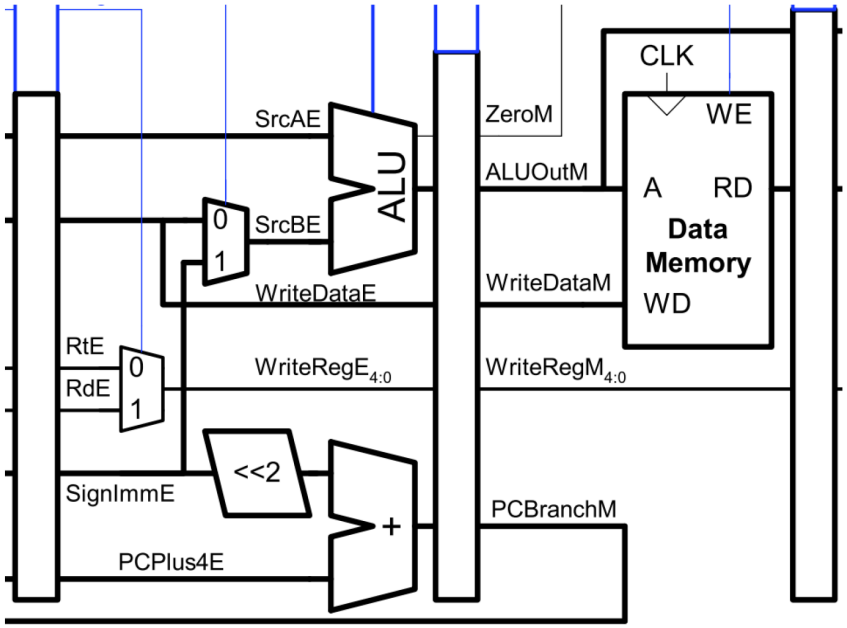
Problem 3: Forwarding [14 pts]

Now we have a typical **5** stage pipeline (instruction fetch (IF) | decode(D) | execute (EX) | memory access (M) | writeback (WB)) RISC-V processor. You are given the assembly code shown below.

```
add x1, x2, x3
xor x2, x1, x4
```

(a) (4 pts) Explain what kind of hazard is present in the code above, and how many extra cycles are introduced if the processor has no data-forwarding. Assume all state elements in the processor are positive edge triggered.

(b) (4 pts) To improve the performance, forwarding is implemented such that the inputs to the ALU can be pulled from the memory access stage. Now, how many extra cycles are caused by the hazard? How would you modify the part of the datapath shown below for this to be possible? Name any new control signals you have added.



(c) (2 pts) Now consider this set of instructions:

```
lw x1, 0(x2)
add x5, x4, x1
```

Assuming an asynchronous read data memory, how many cycles will this set of instructions take to execute? Identify any data hazards.

(d) (4 pts) If you could add another forwarding path from the output `RD` of the data memory, how many cycles will these instructions take to execute? What could be a disadvantage of forwarding from the output of the data memory versus from the pipeline register clocking `RD`?

Problem 4: Branch Prediction [14 pts]

We have a **5** stage pipeline (`IF` | `D` | `EX` | `M` | `WB`) with all necessary forwarding (i.e. no load/data hazard). You are given the assembly code shown below.

```
0x00:   addi x1,  x0, 0
0x04:   addi x5,  x0, 0x70
0x08:   addi x6,  x0, 0x7c
0x0c:   lw   x2,  x5, 4
0x10:   andi x3,  x2, 1
0x14:   add  x1,  x1, x3
0x18:   srli x2,  x2, 1
0x1c:   bne  x2,  x0, -12
0x20:   addi x5,  x5, 4
0x24:   bne  x5,  x6, -24
0x28:   nop
```

The data sent back from a sensor is stored in data memory, as shown below:

Addr	Data
...	...
0x74	0x03e007ff
0x78	0xfc003c00
0x7c	0x000ff800
...	...

Consider the following cases:

(a) (5 pts) `PC+4` is always taken for branch prediction. How many **extra** cycles does the assembly code take?

(b) (5 pts) Branch is always taken for branch prediction. To make sure the branch address is available immediately next cycle, how would you modify the datapath? How many **extra** cycles does the assembly code take?

(c) (4 pts) Please propose a branch prediction strategy. How would that work in this case?