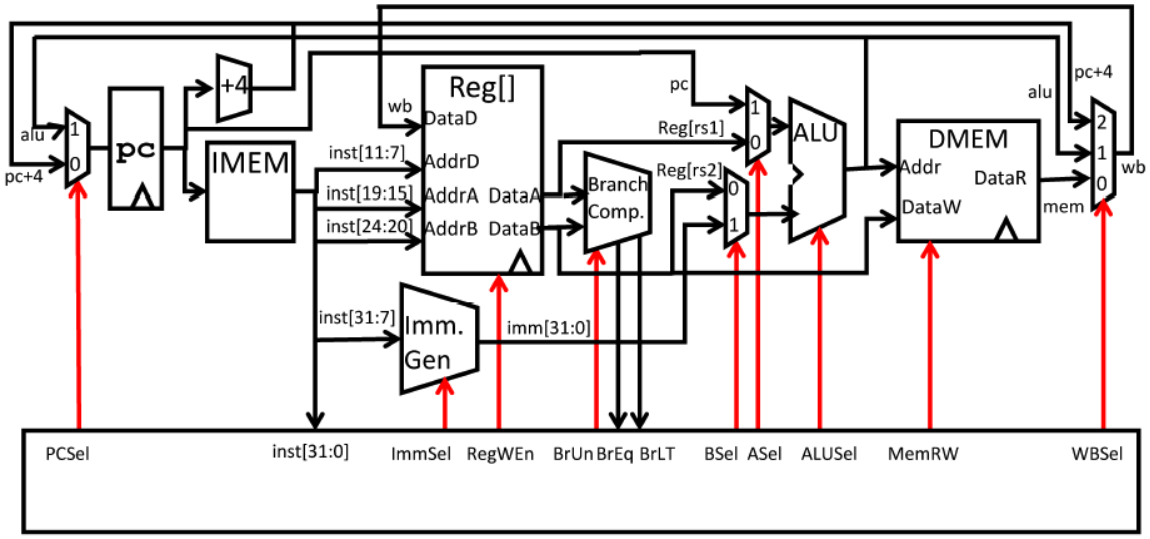
EECS 151/251A Homework 5

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

Solution:

Any instructions that use rs2 in the ALU won't work. This includes all R-type instructions. The explicit list is:
`add, slt, sltu, and, or, xor, sll, srl, sub, sra`.
Note that branches will still work as the branch computation unit is separate from the ALU. Also note stores will still work since Reg[rs2] is directly passed to the DMEM.
To get full credit for this question, you should specify the explicit instruction list, or claim that R-type instructions in general won't work.
For each instruction not on the list, -0.5pt

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

Solution:

`auipc` will no longer work. (1pt)
`branch` (or specifying `beq, bne, blt, bge, bltu, bgeu`) (2pts) and `jal` (1pt) address calculations won't work.
Note `jalr` will still work since the new PC is calculated as rs1 + imm.
For each instruction not on the list, -1pt

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

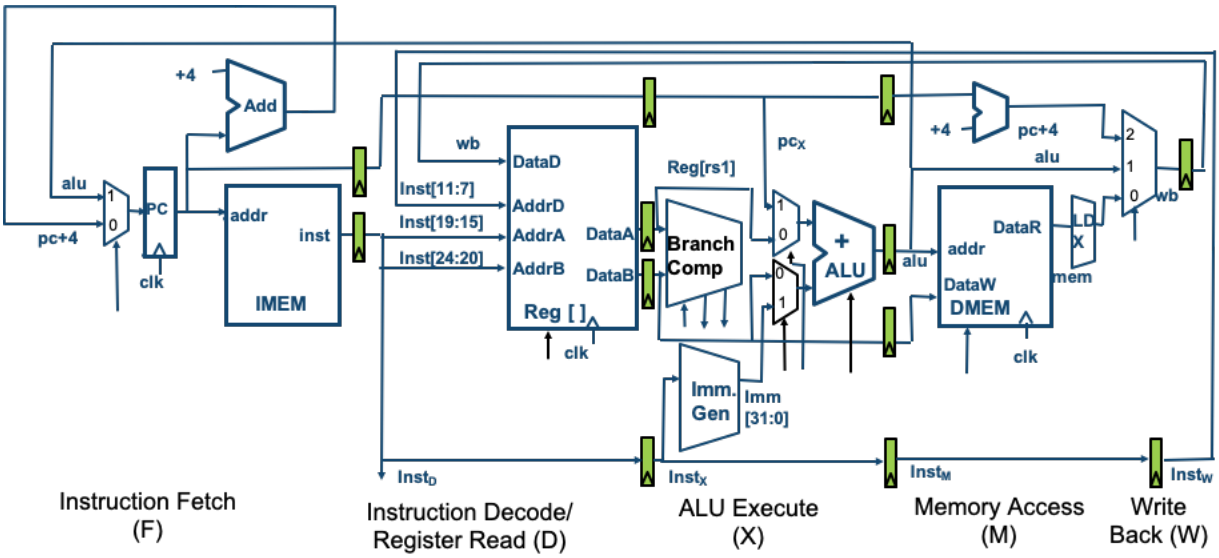
Solution:

`jal` (2pts), `jalr` (2pts) won't work.
For each instruction not on the list, -1pt

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?

Solution:

(a) Takes 19 cycles in total (3pts). The table is as below (5pts):

Cycle	IF	D	EX	M	WB	note
1	li	-	-	-	-	
2	li	li	-	-	-	
3	xor	li	li	-	-	
4	add	xor	li	li	-	
5	add	xor	-	li	li	Data hazard
6	add	xor	-	-	li	
7	slli	add	xor	-	-	
8	blt	slli	add	xor	-	Data hazard
9	blt	slli	-	add	xor	
10	addi(18)	blt	slli	-	add	Data+Ctrl hazard
11	addi(18)	blt	-	slli	-	
12	addi(18)	blt	-	-	slli	
13	beq	addi(18)	blt	-	-	
14	addi(20)	beq	-	blt	-	
15	beq	addi (K)	beq(K)	-	blt	Kill addi(20), beq
16	addi(20)	beq	-	-	-	
17	nop	addi(20)	beq	-	-	
18	-	nop	addi(20)	beq	-	
19	-	-	nop	addi(20)	beq	
20	-	-	-	nop	addi(20)	
21	-	-	-	-	nop	

(b) (2pts)

$$CPI = \frac{21 cycles}{9 instructions} \approx 2.33$$

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.

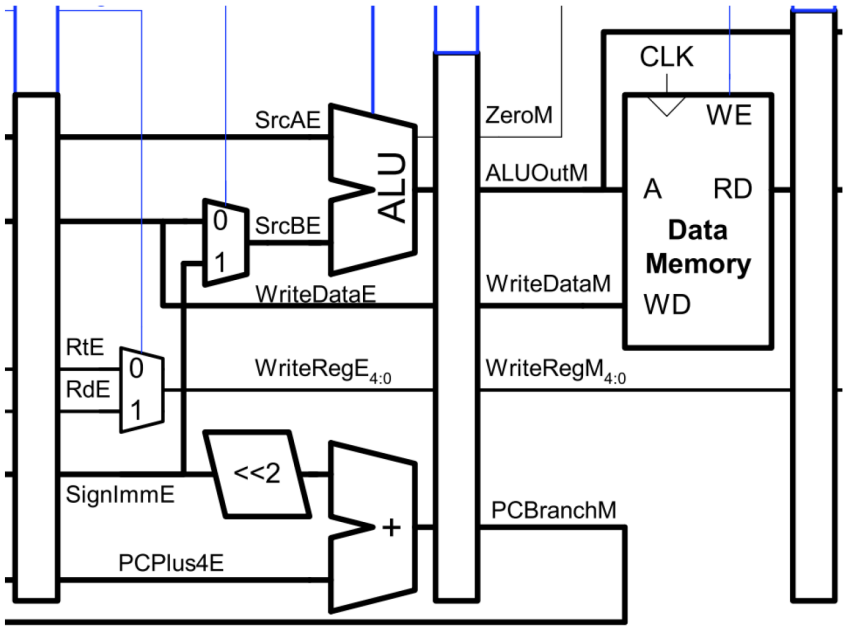
Solution:

There is a read-after-write data hazard present (2pts), involving the value written to the $x1$ register by the first instruction. We can draw a pipeline diagram to analyze this situation:

Cycle	1	2	3	4	5	6	7	8
Inst 1	IF	D	EX	M	WB			
Inst 2		IF	D	-	-	EX	M	WB

Since the result of the first instruction isn't available to the next instruction until after it has been written to the regfile (at the end of cycle 5), the second instruction has to stall until it is available. 2 extra cycles are needed due to the data hazard (2pts).

(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.



Solution:

Now the data needed by instruction 2 is available in cycle 4 and can be forwarded immediately.

Cycle	1	2	3	4	5	6	7	8
Inst 1	IF	D	EX	M	WB			
Inst 2		IF	D	EX	M	WB		

No extra cycles are caused by this hazard now. (2pts)

We can modify the datapath by routing `ALUOutM` from the memory access stage to 2 muxes that drive `SrcAE` and `SrcBE`. The muxes are controlled by one control signal each and if the control signal is 1, `ALUOutM` is forwarded to the respective ALU inputs. We can call these control signals `ALUFwdA` and `ALUFwdB`. (2pts)

(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.

Solution:

There is a data hazard concerning register `x1` similar to part (a) (1pt). As a result these instructions will take 8 cycles to execute without any additional forwarding paths (1pt).

(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`?

Solution:

We could again reduce the total cycles down to 6 (1pt). However, directly feeding the output of the data memory to the execute stage will likely yield a long ciritical path which could un-balance the pipeline and negatively impact the max frequency of operation (3pts).

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?

Solution:

The assembly code counts the 1s from memory 0x74 to 0x7f, and the result should be 35. There are 2 branch instructions:

1. 0x1c: This branch will be taken if the current word have not finished (x2 is not all 0). Considering the content in memory, this instruction will be executed 26+32+20 = 78 times in total: 75 taken, 3 not taken.
2. 0x2c: This branch will be taken if not finish. Executed 3 times in total: 2 taken, 1 not taken.

It takes **3** extra cycles if a wrong prediction is made (Branch compare result will be available at the end of EX. If use 2 for the whole problem, receive 80% credit if the rest is

correct). For this part, branch is not taken by default, so there will be $3 \times (75 + 2) = 231$ extra cycles.

(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?

Solution:

There will be extra hardware that directly calculates the value of the new PC.
For this part, branch is taken by default, so it will have $3 \times (3 + 1) = 12$ extra cycles.

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

Solution:

Many solutions will be accepted. For example, based on last choice: there will be $3 \times (1 + 1 + 1 + 1 + 1) = 15$ extra cycles. It does have more cycles comparing to (b), but it is a more general strategy.