6.004 Recitation Problems
L20 – Pipelined Processors

Problem 1.

The program shown on the right is executed on a 5-stage pipelined RISC-V processor with full bypassing.

The program has been running for a while and execution is halted at the end of cycle 108.

The pipeline diagram shown below shows the history of execution at the time the program was halted.

```
x- = 0
   addi x11, x0, 16    // initialize loop index J
   addi x12, x0, 0

loop:                  // add up elements in array
   addi x11, x11, -1   // decrement index
   slli x13, x11, 2    // convert to byte offset
   lw x14, 0x310(x13)  // load value from A[J]
   add x12, x12, x14   // add to sum
   bnez x11, loop

   j outer_loop        // perform test again!
```

(A) Please indicate on which cycle(s), 100 through 105, each of the following actions occurred. If the action did not occur in any cycle, write "NONE".

cycle	*100*	*101*	*102*	*103*	*104*	*105*
IF	slli	lw	add	bnez	bnez	bnez
DEC	addi	slli	lw	add	add	add
EXE	NOP	addi	slli	lw	NOP	NOP
MEM	NOP	NOP	addi	slli	lw	NOP
WB	bnez	NOP	NOP	addi	slli	lw

Register value used from Register File: _____100, 105____

Register vale bypassed from EXE stage to DEC stage: _____101, 102____

Register value bypassed from MEM stage to DEC stage: _____NONE____

Register value bypassed from WB stage to DEC stage: _____105_____

(B) Why is the NOP instruction inserted in cycle 104?
The add must wait until the lw is in the WB stage in order to be able to get the results of the lw via a bypass path.

Problem 2.

The following program fragments are being executed on the 5-stage pipelined RISC-V processor with full bypassing. For each fragment, the pipeline diagram shows the state of the pipeline for cycle 1000 of execution. Please fill in the diagram for cycle 1001; use "?" if you cannot tell what opcode to write into a stage. Then for **both** cycles use arrows to indicate any bypassing from the EXE/MEM/WB stages back to the DEC stage.

(A)

```
…
sw x1,  0(x0)
lw x17, 0xC(x1)
addi x2, x2, -4
slli x11, x17, 2
sw x11, 0(x2)
jal ra, fact
…
```

Cycle	*1000*	*1001*
IF	sw	sw
DEC	slli	slli
EXE	addi	NOP
MEM	lw	addi
WB	sw	lw

(B)

```
…
xor x11, x11, x12
slli x12, x12, 3
sub x13, x12, x11
and x12, x13, x11
add x13, x12, x13
sw x13, 0x100(x0)
…
```

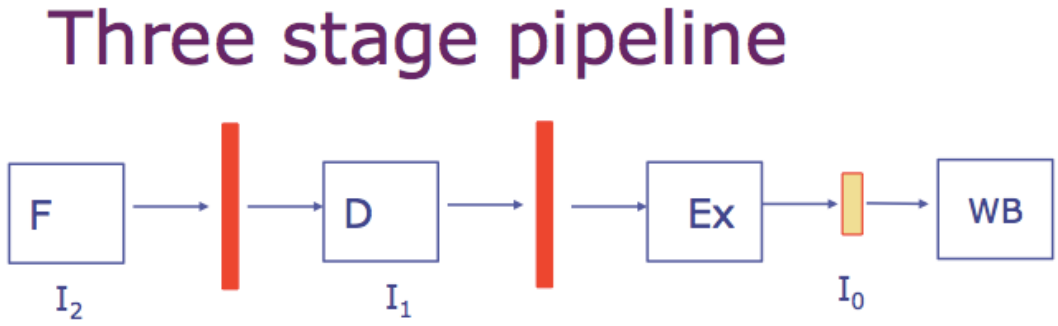
Cycle	*1000*	*1001*
IF	add	sw
DEC	and	add
EXE	sub	and
MEM	slli	sub
WB	xor	slli

Problem 3.

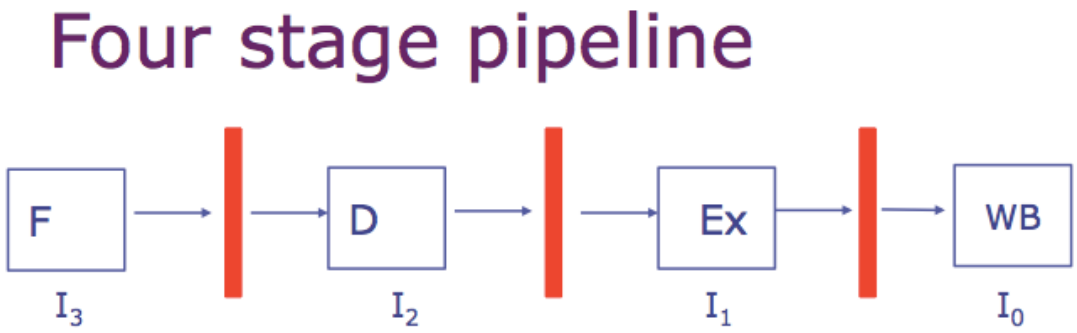
```
    …
L1: add x10, x11, x12
    lw x13, 0(x10)
    beqz x11, L1
    I1 x1, …
    I2 x2, …
    I3 x3, …
    …
```

Consider the performance of the loop on the right on various different processor configurations shown below. *For parts A-C of this problem, assume that no rules conflict.* Assume that there is no bypassing, and assume that a scoreboard is used to deal with data hazards.

P1: Three stage pipeline consisting of a Fetch, Decode, and Execute/Write Back stages. The EX/WB stage takes 2 cycles. Assume that this processor has infinite sized FIFOs between stages.



P2: Four stage pipeline with consisting of Fetch, Decode, Execute, and WB stages. Assume that this processor has infinite sized FIFOs between stages.



P3: Four stage pipeline with consisting of Fetch, Decode, Execute, and WB stages. Assume that this processor has 2-element FIFOs between pipeline stages.

(A) Draw a pipeline diagram showing what repeated executions of the loop would look like for each of the three processor configurations.

For processor P1, also show the contents of the scoreboard between every cycle.

For processor P3, also show the contents of the 2-element I2D FIFO between every cycle.

(B) Now consider how the timing diagram would change on processor P3 to account for writes to the PC occurring in both the IF and EX stages. In other words, you should no longer assume that no rules conflict. To address this issue, split the EX rule into two and demonstrate its effect on the timing diagram.

A)
P1: 3 stage pipeline with infinite sized FIFOs.

Cycle	1	2	3	4	5	6	7	8	9	10
Epoch	0	0	0	0	0	0	0	0	0	1
IF	add	lw	beqz	I1	I2	I3	I4	I5	I6	add
DEC		add	lw	lw	lw	beqz	I1	I2	I3	K(I4)
EX			add	NOP	NOP	lw	NOP	beqz	NOP	P(I1)
WB				add	NOP	NOP	lw	NOP	beqz	NOP

Scoreboard		x10	x10		x13	x0, x13	x1, x0	x2, x1, x0	x3, x2, x1	x3, x2, x1

The rd of each instruction enters scoreboard after that instruction has left the DEC stage. If the instruction has no rd, this is indicated by using x0 in the scoreboard.

K = Kill, P=Poisoned
Killed instructions never make it out of the DEC stage. Poisoned instructions already entered the scoreboard so we need to keep track of them to ensure they are properly removed from scoreboard without modifying any register state. **Do not worry about this distinction.** Treat poisoned instructions as killed.

Note that the 3 stage pipeline processor shown in lecture does the pc redirection after the WB stage.

Cycle	11	12	13	14	15	16	17	18	19	20
Epoch	1	1	1	1	1	1	1	1	1	1
IF	lw	beqz	I1	I2	I3	I4	I5	I6	I7	I8
DEC	K(I5)	K(I6)	add	lw	lw	lw	lw	lw	beqz	I1
EX	NOP	P(I2)	NOP	P(I3)	NOP	add	NOP	NOP	lw	NOP
WB	P(I1)	NOP	P(I2)	NOP	P(I3)	NOP	add	NOP	NOP	lw

Scoreboard	x3, x2	x3, x2	x10, x3	x10, x3	x10	x10		x13	x0, x13	x1, x0

P2: 4 stage pipeline with infinite sized FIFOs

Cycle	1	2	3	4	5	6	7	8	9	10
Epoch	0	0	0	0	0	0	0	1	1	1
IF	add	lw	beqz	I1	I2	I3	I4	add	lw	beqz
DEC		add	lw	lw	lw	beqz	I1	K(I2)	K(I3)	K(I4)
EX			add	NOP	NOP	lw	beqz	P(I1)	NOP	NOP
WB				add	NOP	NOP	lw	beqz	P(I1)	NOP

Scoreboard		x10	x10		x13	x0, x13	x1, x0	x1		

Note that the 4 stage pipeline processor shown in lecture does the pc redirection after the EX stage.

Cycle	11	12	13	14	15	16	17	18	19	20
Epoch	1	1	1	1	1	1	0	0	0	0
IF	I1	I2	I3	I4	I5	I6	add	lw	beqz	beqz
DEC	add	lw	lw	lw	beqz	I1	K(I2)	K(I3)	K(I4)	K(I5)
EX		add	NOP	NOP	lw	beqz	P(I1)	NOP	NOP	NOP
WB			add	NOP	NOP	lw	beqz	P(I1)	NOP	NOP

Scoreboard		x10	x10		x13	x0, x13	x1, x0	x1		

P3: 4 stage pipeline with 2-element FIFOs

I2D FIFO		add	lw	beqz lw	beqz lw	beqz	I1	I2	add	lw	beqz

Cycle	1	2	3	4	5	6	7	8	9	10	11
Epoch	0	0	0	0	0	0	0	1	1	1	1
IF	add	lw	beqz	I1	I1	I1	I2	add	lw	beqz	I1
DEC		add	lw	lw	lw	beqz	I1	K(I2)	add	lw	lw
EX			add	NOP	NOP	lw	beqz	P(I1)	NOP	add	NOP
WB				add	NOP	NOP	lw	beqz	P(I1)	NOP	add

Note that enq into FIFO can only occur if at the beginning of the cycle the FIFO was not full (i.e., had fewer than 2 elements). This is why I1 gets stalled in IF from cycle 4-6. Once the lw is removed from the FIFO then there is room for I1 to enq.

B)
P3: 4 stage pipeline with 2-element FIFOs and split EX rules to reduce conflicts.

I2D FIFO		add	lw	beqz lw	beqz lw	beqz	I1	I2	add	lw	beqz

Cycle	1	2	3	4	5	6	7	8	9	10	11
Epoch	0	0	0	0	0	0	0	1	1	1	1
IF	add	lw	beqz	I1	I1	I1	X	add	lw	beqz	I1
DEC		add	lw	lw	lw	beqz	I1	NOP	add	lw	lw
EX Normal			add	NOP	NOP	lw		P(I1)	NOP	add	NOP
EX Redirect							beqz				
WB				add	NOP	NOP	lw	beqz	P(I1)	NOP	add

Note that IF and EX Redirect conflict because they both update the PC. So in cycle 7, IF cannot fire because the beqz is redirecting the PC.

Problem 4.

The loop on the right has been executing for a while on our standard 5-stage pipelined RISC-V processor with branch annulment and full bypassing.

```
    …
L1: addi x10, x10, -4
    slti x11, x10, 10
    beqz x11, L2
    lw x12, 0x200(x10)
    j L3
L2: lw x12, 0x300(x10)
L3: sw x12, 0x400(x0)
    bnez x10, L1
    addi x12, x12, 1
    xor x12, x12, x0
    …
```

Cycle #	*300*	*301*	*302*	*303*	*304*	*305*	*306*	*307*	*308*	*309*
IF	addi	slti	beqz	lw	j	lw	sw	bnez	bnez	bnez
DEC	NOP	addi	slti	beqz	lw	NOP	lw	sw	sw	sw
EXE	NOP		addi	slti	beqz	NOP	NOP	lw	NOP	NOP
MEM	bnez			addi	slti	beqz	NOP	NOP	lw	NOP
WB	sw				addi	slti	beqz	NOP	NOP	lw

(A) **Fill in the pipeline diagram** for cycles 300-309 assuming that at cycle 300 the instruction at L1 is fetched. Also, assume that the branch to L2 is taken, as well as the final branch back to L1. **Indicate which bypass/forwarding paths are active in each cycle by drawing a vertical arrow in the pipeline diagram** from pipeline stage X in a column to the RF stage in the same column if an operand would be bypassed from stage X back to the RF stage that cycle. Note that there may be more than one vertical arrow in a column.

Fill in pipeline diagram including bypass arrows in pipeline diagram above

(B) Assume that the previous iteration of the loop executed the same instructions as the iteration shown here. Please complete the pipeline diagram for cycle 300 by filling in the OPCODEs for the instructions in the DEC, EXE, MEM, and WB stages.

Fill in OPCODEs for Cycle 300

(C) Indicate which branches are taken by providing the cycle in which the taken branch instruction enters the IF stage.

Cycle number(s) or NONE: __302, 307_____

(D) During which cycle(s), if any, do we have stalled instructions?

Cycle number(s) or NONE: __307, 308_____

Now consider a modified processor, P2, which has extra hardware in the decode stage (DEC) to resolve simple branches one cycle earlier: the decode stage includes both a circuit to check whether a register is equal to zero, and an extra adder to compute the branch target for taken branches. This processor can thus compute nextPC for `beqz` and `bnez` in DEC instead of EXE.

(E) Redo part A using processor P2 assuming the same path is taken through the code.

Cycle #	*300*	*301*	*302*	*303*	*304*	*305*	*306*	*307*	*308*	*309*
IF	addi	slti	beqz	lw	lw	sw	bnez	bnez	bnez	addi
DEC	NOP	addi	slti	beqz	NOP	lw	sw	sw	sw	bnez
EXE	bnez		addi	slti	beqz	NOP	lw	NOP	NOP	sw
MEM	sw			addi	slti	beqz	NOP	lw	NOP	NOP
WB	NOP				addi	slti	beqz	NOP	lw	NOP

(F) Compare the number of cycles per loop iteration using the original processor and the modified processor.

Cycles per loop in original processor: ___12_____

Cycles per loop in processor P2: ___10_____