6.004 Tutorial Problems
L19 – Control Hazards in Pipelined Processors

Problem 1. ★

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									
DEC										
EXE										
MEM										
WB										

(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: _____

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

Cycle number(s) or NONE: _____

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`									
DEC										
EXE										
MEM										
WB										

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

Cycles per loop in original processor: _____

Cycles per loop in processor P2: _____

Problem 2. ★

You've discovered a secret room in the basement of the Stata center full of discarded 5-stage pipelined RISC-V processors. Unfortunately, many have certain defects. You discover that they fall into four categories:

C1: A modified, completely functional 5-stage RISC-V processor with working bypass paths, annulment, and other components, as well as extra hardware support that allows the nextPC to be calculated in the Decode stage.
C2: A defective version of **C1** with a bad register file: all data read from the register file is zero.
C3: A defective version of **C1** with broken bypass muxes: all source operands come from the register file, even if they should be read from bypassed paths.
C4: A defective version of **C1** without annulment of instructions following branches.

To help sort the processors into the above classes, you write the following small test program:

```
. = 0x0
// Start at 0x0, with ZERO in all registers…
      addi x10, x0, 4
      jal x12, X
      slli x12, x12, 1
X:    addi x12, x12, -4
      add x13, x12, x10
      jr x13
```

Your plan is to single-step through the program using each processor, carefully noting the address the final `jr` loads into the PC. Your goal is to determine which of the above four classes a chip falls into by this `jr` address.

For each class of RISC-V processor described above, specify the value that will be loaded into the PC by the final `jr` instruction.

Pipeline diagram showing first 7 cycles of test program executing on C1:

cycle	*0*	*1*	*2*	*3*	*4*	*5*	*6*
IF	addi	jal	slli	addi	add	jr	
DEC		addi	jal	NOP	addi	add	jr
EXE			addi	jal	NOP	addi	add
MEM				addi	jal	NOP	addi
WB					addi	jal	NOP

C1: jr goes to address: _____

C2: jr goes to address: _____

C3: jr goes to address: _____

C4: jr goes to address: _____

Problem 3. ★

(A) How many cycles does it take to run each iteration of the following loop on a standard 5-stage pipelined RISC-V processor?

```
loop: lw x10, 0x100(x0)
      beqz x10, loop
      add x12, x10, x11
      sub x13, x12, x1
```

Number of cycles per loop iteration: _____

(B) Assuming a defective 5-stage pipelined RISC-V processor where the instructions following a taken branch are not annulled, which of the following statements would be true?

1. The add instruction would be executed each time through the loop.
2. The loop would take 5 cycles to execute
3. The value of the register x10 that is tested by the beqz instruction comes from a bypass path.
4. The value of register x10 that is accessed by the add instruction comes from the register file.

(C) Consider a modified processor, P2, which has extra hardware for the special case of checking if a register is equal to zero or not in the decode stage. What would be the number of cycles per loop iteration in this case?

Number of cycles per loop iteration on processor P2: _____

(D) Now consider a third processor, P3, whose instruction and data memories are pipelined and take 2 clock cycles to respond (instead of a single cycle as usual). Assume that P3 also has the extra hardware for checking if a register is equal to zero or not in the decode stage. What would be the number of cycles per loop iteration using P3?

Number of cycles per loop iteration on processor P3: _____

Problem 4.

You've been given a 5-stage pipelined RISC-V processor. Unfortunately, the processor you've been given is defective: it has no bypass paths, annulment of instructions in branch delay slots, or pipeline stalls.

```
        nop
        nop
        nop
        nop

loop:
        lw x10, 0x0(x10)
AA:
        sll x14, x10, x11
BB:
        bnez x10, loop
CC:
        add x13, x10, x13

        nop
        nop
        nop
        nop
```

You undertake to convert some existing code, designed to run on an unpipelined RISC-V, to run on your defective pipelined processor. The fragment of code above is a sample of the program to be converted. Add the **minimum** number of **NOP** instructions at the various tagged points in this code to make it give the same results on your defective pipelined RISC-V as it gives on a normal, unpipelined RISC-V.

Note that the code fragment begins and ends with sequences of **NOPs**; thus, you don't need to worry about pipeline hazards involving interactions with instructions outside of the region shown.

(A) Specify the minimal number of **NOP** instructions (*defined as addi x0, x0, 0*) to be added at each of the labeled points in the above program.

NOPs at Loop: _____

NOPs at AA: _____

NOPs at BB: _____

NOPs at CC: _____

(B) On a **fully functional** 5-stage pipeline (with working bypass, annul, and stall logic), the above code will run fine with no added **NOP**s. How many clock cycles of execution time are required by the fully functional 5-stage pipelined RISC-V **for each iteration** through the loop?

Clocks per loop iteration: _____