CS 251, Spring 2023, Assignment 4

Due Friday, July 7, 10:00 PM
Late submission accepted until Monday, July 10, 10:00PM with no penalty.

You are required to read, complete and sign, and submit (as well as follow) the following statement of Academic Integrity.

Statement of Academic Integrity for CS 251 Spring 2023, **Assignment 4**
I declare the following statements to be true:

- I have not used any unauthorized aids.
- I recognize that while I can discuss the questions in this assignment on Piazza and other forums with the instructors and with other students in the class, the write up that I am submitting is my own.
- I am aware that misconduct related to course work can result in significant penalties, including failing the course and suspension (this is covered in Policy 71:)

  `https://uwaterloo.ca/secretariat/policies-procedures-guidelines/policy-71`

Student Name:

UW ID#:

Signature:

Date:

The purpose of this assignment is to help you understand the implementation of the single cycle processor.

Coverage material for this assignment can be found on the course website as:

- On the Lecture Notes webpage in the
  - June 29th - July 6th Lectures on Pipelined Computers, Control and Data Hazards, NOPs, Forwarding, Stalls, Branch Prediction and Code Rearrangement

For this assignment, make note of the following details:

- Any diagrams that are part of your solution must be drawn neatly, using rectilinear lines and clearly labelled inputs and outputs.
- **You should assume the branch prediction method is *predict not taken* unless otherwise stated in the question.**

We have an A4 Official Post on piazza. In this post, we will include all A4 assignment related information such as

- updates/corrections made to the assignment, if any
- remark request due date
- FAQs

1. (10 points) This question refers to the pipelined datapath without forwarding, shown below.

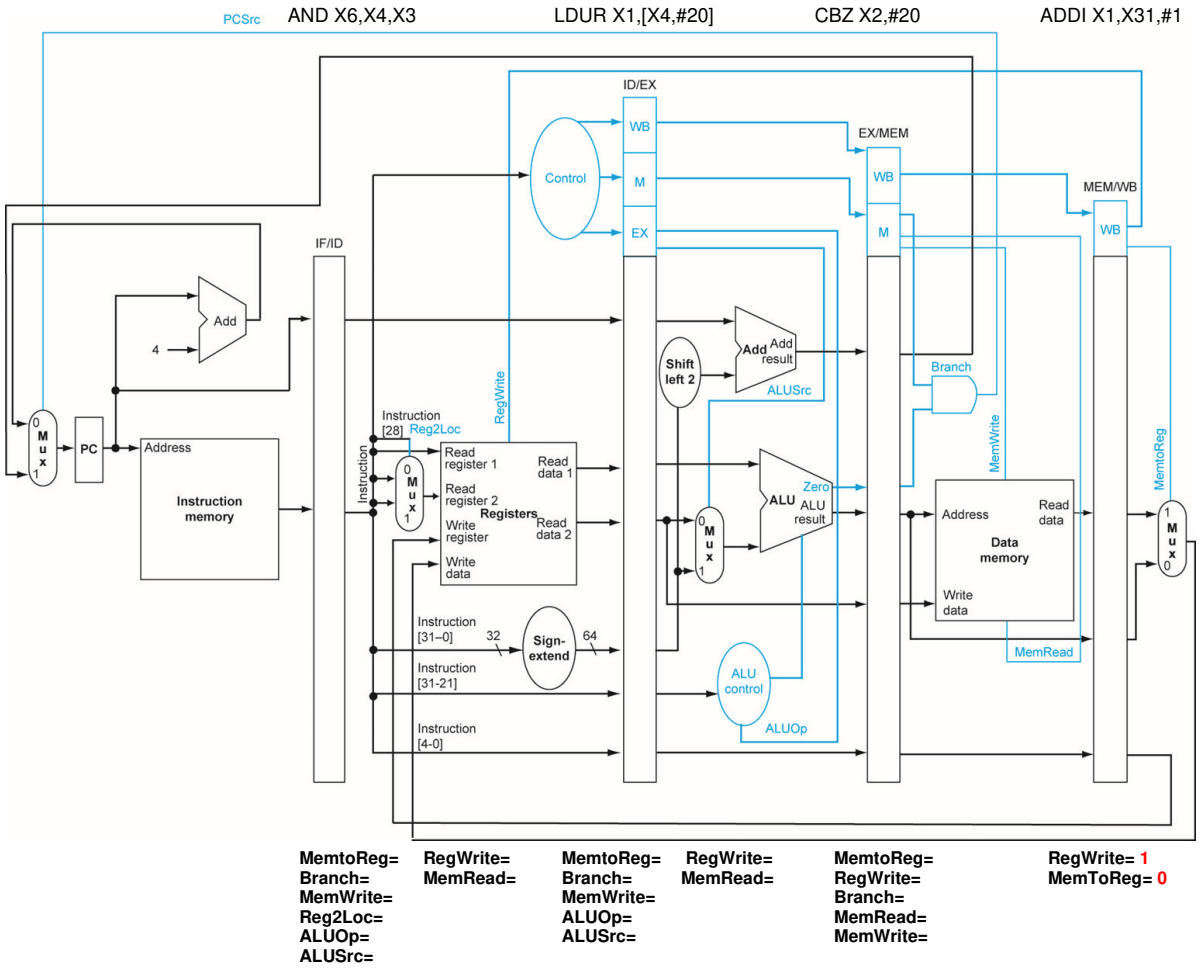
   Consider the following instructions:

```
100 ADDI X1, X31, #1     ; WB stage
104 CBZ  X2, #20         ; MEM stage
108 LDUR X1,[X4, #20]    ; EX stage
112 AND  X6, X4, X3      ; ID stage
```

   The instructions are executing in the stages indicated in the comments and in the figure below.

   In the figure below, label all of the *control signals* (including both those coming directly out of the control unit and those coming out of the pipeline registers) with their appropriate values, using don't cares where appropriate.

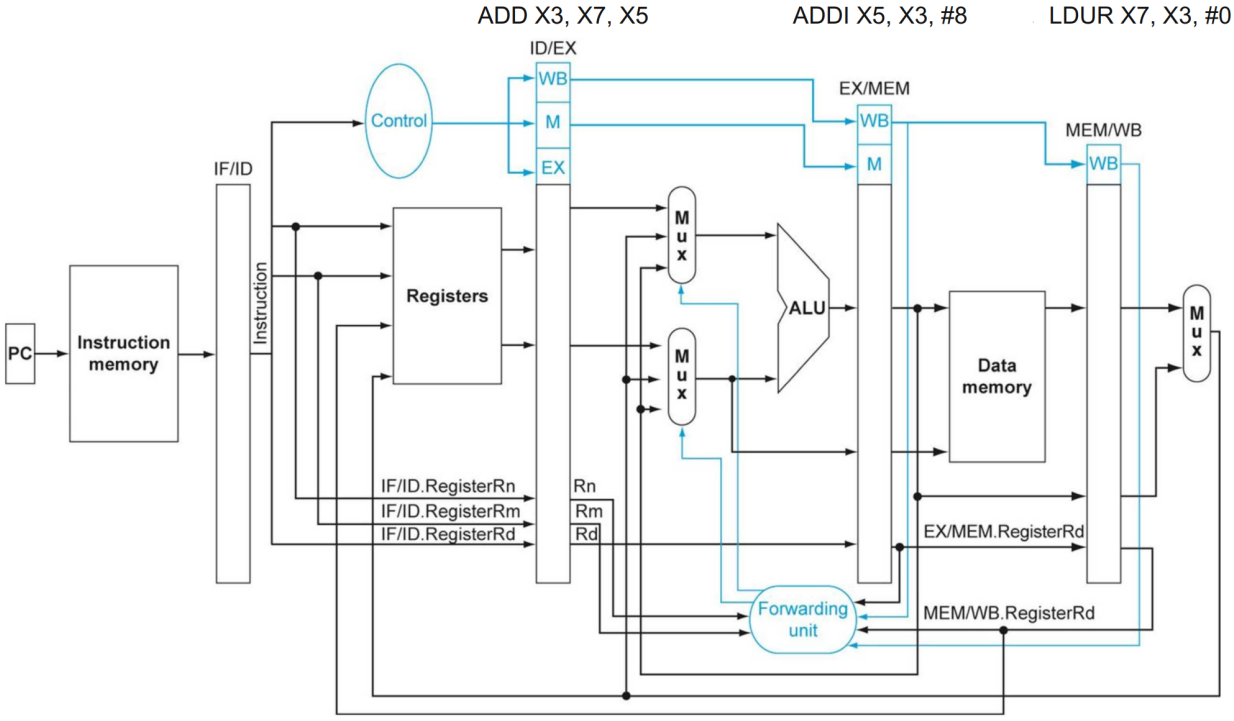
   In this figure, the instructions have been drawn above the appropriate set of pipeline registers. We have also filled in the solution for the WB stage, and given the names of the control signals whose values you need to determine for the other three stages.



2. (4 points) Consider the instructions

```
320 LDUR X7, [X3, #0]
324 ADDI X5, X3, #8
328 ADD X3, X7, X5
```

Consider the situation when the `320 LDUR` instruction is in the WB stage, the `324 ADDI` instruction is in the MEM stage, and the `328 ADD` instruction is in the EX stage. In the figure below, for the current **two inputs to the ALU**, trace the buses back through the MUXes to the appropriate set of **pipeline registers** (i.e., trace paths from the ALU's input back to the pipeline registers where the appropriate operands are stored). You may use bold dark lines to clearly indicate your work.



3. (8 points) Consider the following ARM code sequence where Data Memory at address 16 stores the value 2.

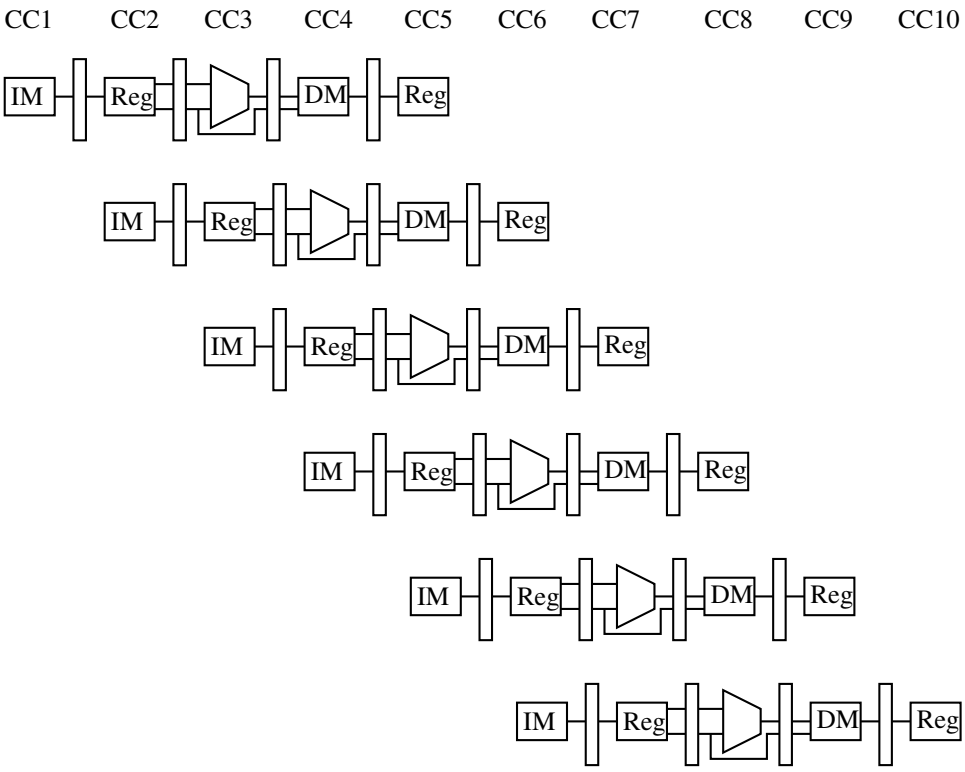
```
100     ADDI X1, XZR, #16
104     LDUR X2, [X1, #0]
108     ADDI X3, X1, #8
112     CBNZ X2, #-2
116     SUBI X1, X1, #8
```

Assume this code sequence is run on a datapath with branch in the ID-stage, with **data forwarding** (to both the EX stage and the '=0' unit in the ID stage), **load-use stalling**, and **branch-data stalling**. In the figure below, to the left of each row, write the instruction that is executed in that row. If an instruction is stalled, write the instruction that is stalled and then write "STALLED" below that. If an instruction is flushed, write the name of the instruction and then write "FLUSHED" below that.

Also draw where the forwarding occurs, similar to Figure 4.52 of the textbook, in the execution of these instructions.

Your straight line must clearly go to the ALU input (Rn or Rm) that requires forwarding (Rn is the top input to the ALU; Rm is the bottom input).

The execution of these instructions (including stalls and flushing) may require more rows than provided. If so, just show the first six rows.


CC1
CC2
CC3
CC4
CC5
CC6
CC7
CC8
CC9
CC10
IM
Reg
DM
Reg

4. (4 points) Given the code segment below, compute the total execution time in clock cycles and picoseconds (ps) for the execution of this code on the Single Cycle and Pipelined datapaths. You may assume a clock cycle time of 800ps on the Single Cycle datapath and 200ps on the Pipelined datapath. The Pipelined datapath **determines branch in the ID stage** and implements both forwarding and stalling (load-use stall) as shown in the textbook and lectures. The datapath also implements flushing for the instruction following the branch. You do **not** need to add pipeline start-up time.

```
04 SUB  X1,XZR,XZR
08 ADDI X2,XZR,#400
12 ADDI X3,XZR,#0
16 LDUR X4,[X1,#200]
20 ADDI X1,X4,#8
24 ADDI X2,X2,#8
28 SUBI X5,X3,#10
32 ADDI X3,X3,#1
36 CBNZ X5,#-5
40 ADD  X5,X1,X1
```

Datapath	Total Number of Clock Cycles (cc)	Total Execution time picoseconds (ps)
Single Cycle		
Pipelined		

5. (4 points) CPI is a measure of clock cycles per instruction that is used to compare Instruction Set Architectures. Find the average CPI in the following situations. Assume an instruction mix of

```
15% Load words
10% Store words
60% R-format
10% Conditional Branch
5%  Unconditional Branch
```

Suppose this instruction mix is executed on the pipeline datapath where **Branch is determined in the ID stage** and the datapath implements data forwarding, load-use stalling, branch data stalling and branch flushing when necessary. Assume 10% of all load-words are followed by a use and generate a load-use hazard; one quarter of all branch instructions are mispredicted and 20% of all branch instructions generate a branch data hazard requiring one stall.

State the average CPI. Show your work and the formula you used.

6. (8 points) The following code executes on the pipelines computer with **branch in the ID stage** with data forwarding, load-use stalls, branch flushing, and branch data stalling.

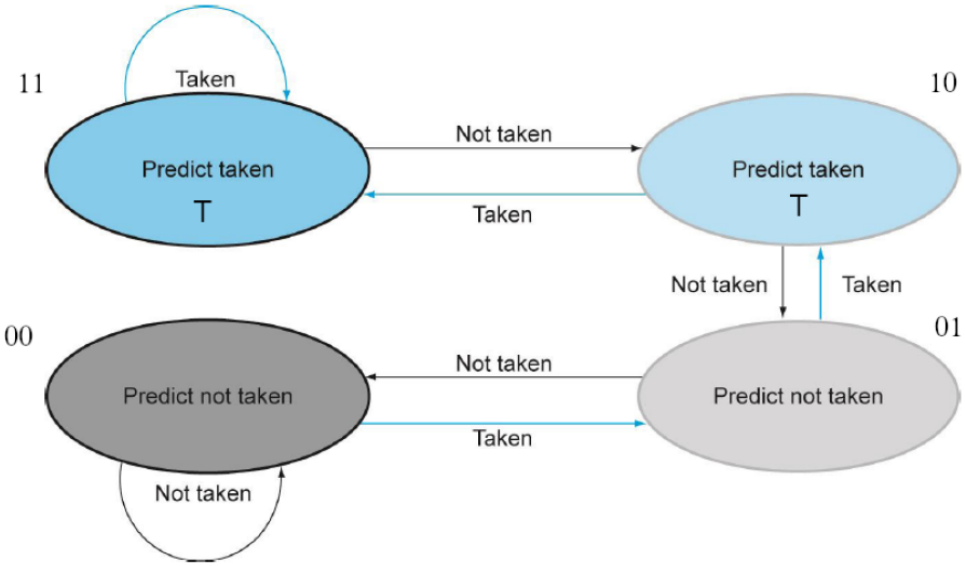
```
396 ADD  X4,XZR,XZR
400 ADDI X5,X31,#24
404 ADDI X7,X31,#71
408 LDUR X6,[X7,#0]
412 ADD  X4,X4,X6
416 ADD  X4,X4,X5
420 ADDI X7,X7,#8
424 SUBI X5,X5,#2
428 CBNZ X5,#-5
432 ADDI X9,X6,X31
```

(a) (5 points) Compute how many correct/incorrect branch predictions are made using various schemes for branch prediction.

For all branch prediction methods, assume that the branch destination table has already been built and you begin with branch not taken for 1-bit prediction. The start state for 2-bit prediction is indicated below (see also the FSM on the following page).

	Branch Prediction	
Next branch instruction	Number Correct	Number Incorrect
PC+4		
Branch destination		
1-bit Branch Prediction		
2-bit start at state 01		
2-bit start at state 11		

(b) (3 points) Here is the FSM for the 2-bit branch predictor:




```
400 ADDI X5,X31,#8
404 ADDI X7,X31,#56
408 LDUR X6,[X7,#0]
412 ADDI X7,X7,#8
416 SUBI X5,X5,#1
420 CBNZ X5,#-3
424 ADDI X9,X6,X31
```

Trace the state of the 2-bit branch predictor for line `420 CBNZ` in the code above. Assume that the FSM starts in state 00, and that it generates an output T that is high (1) when predicting that the branch should be taken. Note whether each branch prediction was correct (C) or incorrect (I). You should trace the code until it completes execution of line 424.

Current State	00
T	0
Correct/**I**ncorrect	I
Next State	