University of Western Ontario, Computer Science Department
CS3350B, Computer Organization

Assignment 4 **Due: March 27, 2020**

General Instructions: This assignment consists of 4 pages, 5 exercises, and is marked out of 100. For any question involving calculations you must provide your workings. You may collaborate with other students in the class in the sense of general strategies to solve the problems. But each assignment and the answers within are to be solely individual work and completed independently. Any plagiarism found will be taken seriously and may result in a mark of 0 on this assignment, removal from the course, or more serious consequences.

Submission Instructions: The answers to this assignment are to be submitted on OWL as a **single PDF file**. Ideally, the answers are to be typed. At the very least, clearly *scanned* copies (no photographs) of hand-written work. If the person correcting your assignment is unable to easily read or interpret your answer then it may be marked as incorrect without the possibility of remarking.

Useful Facts:

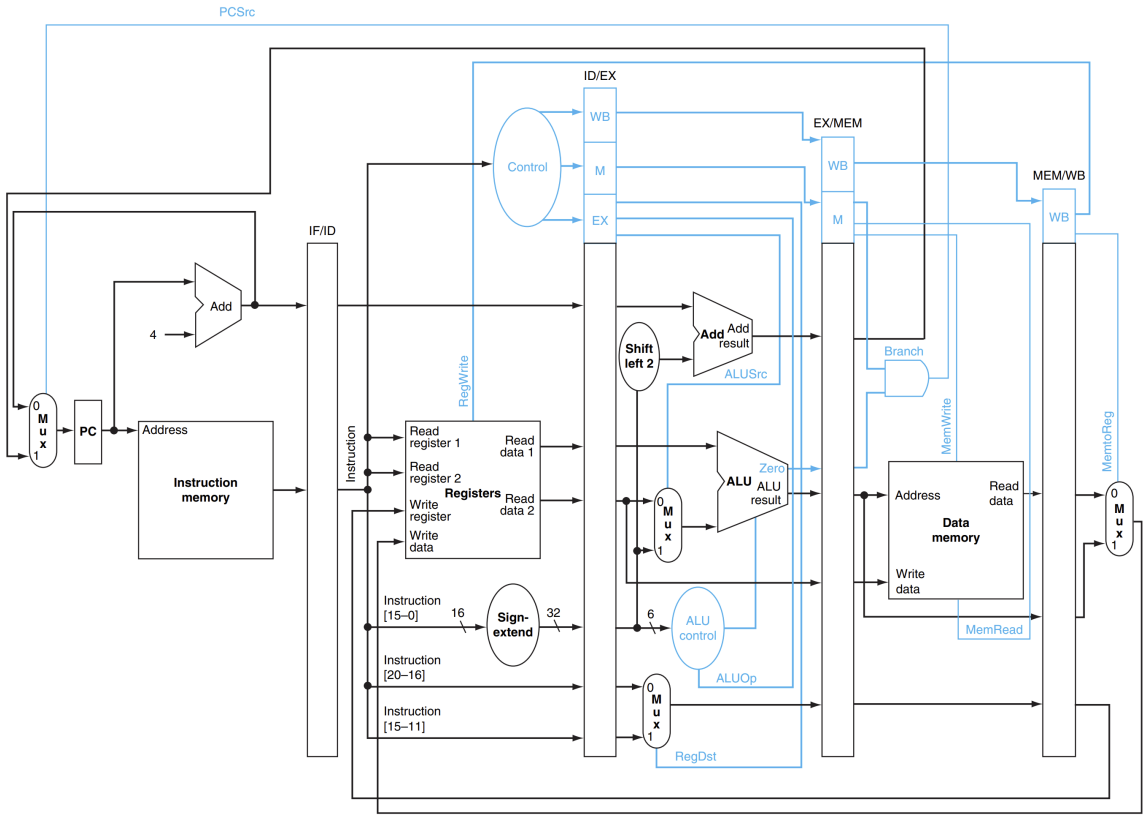

Figure 1: MIPS Datapath with Interstage Registers

Exercise 1. [20 Marks] Consider the multi-cycle MIPS datapath presented in Figure 1, it shows 4 inter-stage registers: IF/ID, ID/EX, EX/MEM, and MEM/WB. Consider also the control signals presented in the diagram in blue. Assume the ALUOp control signal is 3 bits. Ignore control signals not shown (i.e. the ones controlling forwarding). Determine the minimum size, in bits, of each of the four inter-stage registers. For part marks, ensure to include workings and not just the total values.

Exercise 2. [15 Marks] Consider a pipelined processor with s stages. Each stage except the last runs in t units of time (say pico-seconds) meanwhile the last stage runs in $r \times t$ units of time where $r > 1$. Thus, the last stage is slower than the other ones. Assume there are n tasks being processed by this pipeline.

(a) Compute an expression for the actual pipeline speedup based on the variables s, t, r, and n. Show your workings.

(b) Compute an expression for the ratio of time for which the pipeline runs at full occupancy based on the variables s, t, r, and n. Show your workings.

Exercise 3. Consider the following `C` code:

```
for (i = 0; i < n; ++i)
  a[i] = b[i] + i;
```

where `a` and `b` are 32-bit integer arrays of size `n`. We have a corresponding `MIPS` instruction sequence:

```
        add $t0, $0, $0     # $t0 = 0, which corresponds to i in C code
loop:   lw $s1, 0($s4)      # assume $s4 stores the base address of array b
        add $s0, $s1, $t0   # $s0 gets b[i] + i
        sw $s0, 0($s2)      # assume $s2 stores the base address of array a
        addi $t0, $t0, 1    # ++i
        addi $s2, $s2, 4    # get address of a[i+1]
        addi $s4, $s4, 4    # get address of b[i+1]
        slt $t2, $t0, $s5   # assume that $s5 holds n
        bne $t2, $0, loop   # if $t2 == 1, go to loop
```

Assume that the above `MIPS` instructions will be executed on a 5-stage pipelined processor. Ignore control hazards and structural hazard. Assume the branch condition is computed in the EX stage.

(a) [10 Marks] Draw the pipeline execution diagram for one iteration of the loop, that is, starting at the label `loop`. Assume there is no data forwarding. Compute the CPI (clock cycles per instruction) for that one iteration. Do not re-order the instructions.

(b) [10 Marks] Draw the pipeline execution diagram for one iteration of the loop, that is, starting at the label `loop`. This time, assume all possible data forwarding is used. For each instance of data forwarding, annotate your pipeline diagram (say, with arrows, colors, or footnotes) to indicate the source and destination of the forwarding; also say what kind of forwarding it is. Compute the CPI (clock cycles per instruction) for that one iteration. Do not re-order the instructions.

(c) [10 Marks] Apply loop unrolling (as well as instruction re-ordering, if you like) on the above `MIPS` code for two iterations. Write out the corresponding `MIPS` instruction code. Make sure to use offsets appropriately to avoid unnecessary instructions.

(d) [15 Marks] Consider a 2-issue extension of MIPS. That is, a VLIW extension of MIPS where two instructions occur in each instruction word. The issue packet has the following format: the first instruction must be arithmetic or branch, and the second instruction must be a data transfer instruction (`lw` or `sw`). Still assume all types of forwarding.

Using the code of your unrolled loop of part (c), statically schedule one iteration of the loop to run optimally on this 2-issue machine. You may use additional registers to accomplish this task. Consider using the following table as a starting point and to help guide you through the process.

	ALU or branch	Data transfer	CC
`loop:`			1
			2
			3
			4
			5
			6
			7
			8
			9
			⋮

Exercise 4. [15 Marks] Consider a shared-memory multiprocessor that consists of three processor/cache units where cache coherence is maintained by a MESI protocol. The private caches are direct mapped. Assume that words X1, X2 and X3 are in the same cache line. Given the following sequence of events, identify each access as a cold miss (CM), a true sharing miss (TM), a false sharing miss (FM), or a hit (H).

Clock	Processor 1	Processor 2	Processor 3	CM, TM, FM or H
1	Read X1			
2		Read X2		
3			Read X3	
4	Write X1			
5			Write X3	
6		Read X1		
7	Write X2			
8			Read X1	
9			Read X2	

Exercise 5. [5 Marks + 5 Bonus] An inlined function is a function augmented with the keyword `inline`. The keyword `inline` is a directive to the compiler to insert the function's code body in place of any call to the function. An example is below. Based on all of the material in this course, explain why inlined functions can allow for code to execute more quickly. A short one sentence (correct) answer will yield the 5 marks. Very good answers will yield 5 bonus marks.

```
inline int foo(int a, int b) {
    return a + b;
}

int main(int argc, char** argv) {
    int i = 10;
    int j = 12;
    int c = foo(i,j);
}
```