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

Assignment 3 **Due: March 13, 2020**

General Instructions: This assignment consists of 5 pages, 6 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 Things:

A MIPS simultator, *spim*: http://pages.cs.wisc.edu/~larus/spim.html

List of MIPS isntructions: http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html

Example MIPS code which runs on spim is provided in OWL under resources.

Labels can be used in assembly in replace of calculating exact values for branch and jump instructions. The following is an example.

```
int isNeg(int a0) {
    if(a0 < 0) {
        return 1;
    } else {
        return 0;
    }
}
```

```
isNeg:
    slt  $t0 $a0 $0
    beq  $t0 $0  isPos
    addi $v0 $0  1
    jr   $ra
isPos:
    add  $v0 $0  $0
    jr   $ra
```

When translating code to and from assembly, one should usually consider local variables as being stored in registers and memory accesses as being done through pointers or arrays.

```
int loadEx(int* a0) {
    int t0 = a0[0];
    return t0;
}
```

```
loadEx:
    lw   $t0 0($a0)
    add  $v0 $t0 $0
    jr   $ra
```

Exercise 1. Consider the following MIPS function.

```
foo1:
    addi  $t0  $a0  37
    addi  $t1  $a1  -12
    sll   $t2  $t1  2
    slt   $t3  $t2  $t0
    beq   $t3  $0   a
    addi  $v0  $0   1
    jr    $ra
a:
    addi  $v0  $0   2
    jr    $ra
```

(a) [**3 marks**] Assuming \$a0 = 12 and \$a1 = 4 what is the value of \$t2, \$t3, \$v0 at the end of this code fragment.

(b) [**8 marks**] Translate the MIPS function into C-style pseudo-code. You may use any variable names you wish in place of register names. Ensure variables are declared using the proper data type.

Exercise 2. [**12 marks**] Consider the following MIPS function.

```
foo2:
    sltiu $t0  $a1  3
    add   $v0  $0   $0
    bne   $t0  $0   b
    lw    $t0  0($a0)
    lw    $t1  4($a0)
    lw    $t2  8($a0)
    sw    $t0  8($a0)
    sw    $t2  4($a0)
    sw    $t1  0($a0)
    addi  $v0  $v0  1
b:
    jr    $ra
```

Translate the MIPS function into C-style pseudo-code. You may use any variable names you wish in place of register names. Ensure variables are declared using the proper data type.

Exercise 3. [14 Marks]

Implement a function in MIPS which *recursively* computes the **factorial** of a number. You do not need to create an entire working MIPS program, just a function to compute factorial. Hint 1: don't worry about overflow. Hint 2: you may use the *pseudo-isntruction* `mul`.

$$\texttt{mul}\ \$a\ \$b\ \$c \equiv \begin{array}{l}\texttt{mult}\ \$b\ \$c\\ \texttt{mflo}\ \$a\end{array}$$

Exercise 4. [30 Marks]

Implement a function (or two functions) in MIPS which implements **bubblesort**. You do not need to create an entire working MIPS program, just the function for bubblesort. Your MIPS function should have a signature equivalent to the following C function. Hint: the lecture slides already give MIPS assembly for `swap` and most of bubblesort.

```
//sort the array a in place. a has n elements.
void bubblesort(int* a, int n);
```

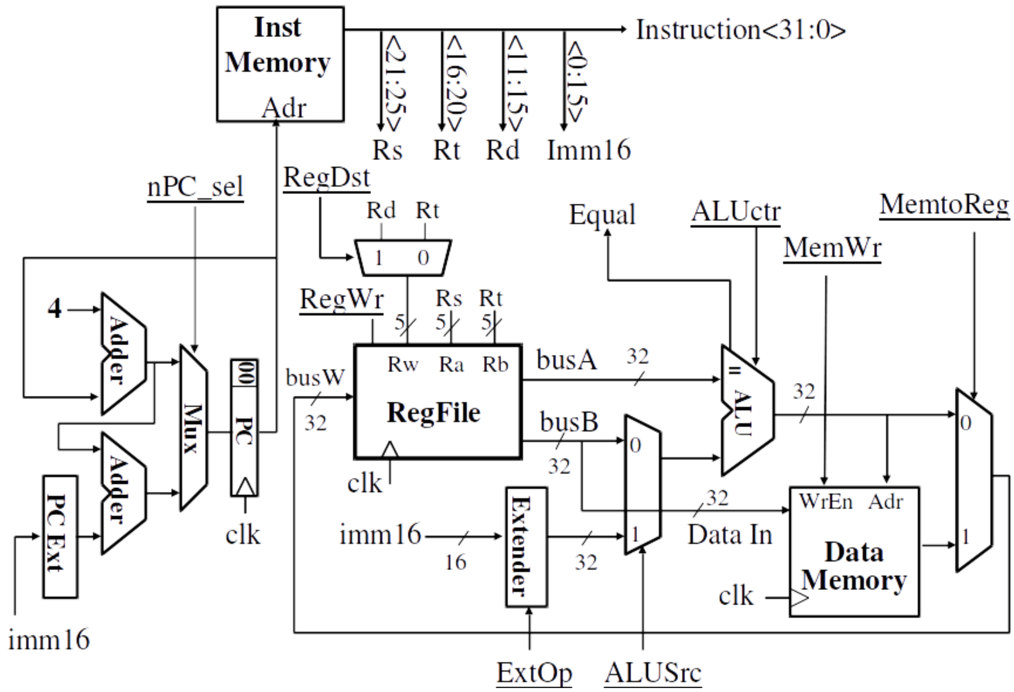


Figure 1: MIPS datapath with control signals

Exercise 5. Consider the MIPS datapath with control signals as presented in Figure 1.

(a) [**8 marks**] Give the control signals required to execute the instruction:

```
bne $s0 $s1 L
```

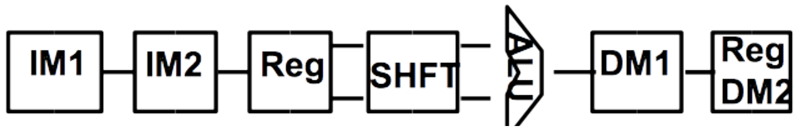
You may use "x" to denote a "don't care" value. You may use the semantic meaning for each control signal (e.g. ALUCtr = "add").

(b) [**8 marks**] Give the control signals required to execute the instruction:

```
sw $s0 12($t0)
```

You may use "x" to denote a "don't care" value. You may use the semantic meaning for each control signal (e.g. ALUCtr = "add").

Exercise 6. Consider the following 7 stage datapath and the timings for each stage.



Stage	IM1	IM2	REG	SHFT	ALU	DM1	DM2
Time	200ps	300ps	100ps	100ps	400ps	300ps	200ps

(a) [**2 marks**] Using single-cycle clocking for this datapath what is the minimum clock cycle possible for this datapath?

(b) [**2 marks**] If this datapath were to be pipelined, what is the minimum clock cycle which would be possible?

(c) [**5 marks**] What is the ideal speedup of this datapath using pipelining? What is the actual speedup obtained when executing 100 instructions? Assume that there are no dependencies or hazards between instructions.

(d) [**8 marks**] Let us assume this datapath is not pipelined but follows a multi-cycle clocking method, using the clock cycle as in **(b)**. For this part of the exercise, let us add to the datapath an optimization where instructions can skip stages that are unused for that instruction. Consider the following set of instructions to be executed on this datapath along with the stages in which meaningful work is performed.

`add`: IM1, IM2, REG, ALU, DM2

`sll`: IM1, IM2, REG, SHFT, DM2

`sw`: IM2, REG, ALU, DM1

`lw`: IM1, IM2, REG, ALU, DM1, DM2

Given a program composed of 400 `add`, 100 `sll`, 300 `sw`, and 400 `lw` instructions, calculate the average CPI of the program and the time to execute the program. Assume that there are no dependencies or hazards between instructions. (Hint: how many clock cycles are required for each stage and each instruction type?)