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

Assignment 3 **Due: March 16, 2023**

General Instructions: This assignment consists of 8 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 to Gradescope. 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 simulator, *spim*: http://pages.cs.wisc.edu/~larus/spim.html

List of MIPS instructions: https://inst.eecs.berkeley.edu/ cs61c/resources/MIPS_help.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. [**16 marks**] For each of the following MIPS assembly instructions, give the instruction format, the decimal value of each bit field for that instruction format, and the 32-bit instruction word encoding that instruction *written as a single hexadecimal number*.

- `add $v1, $a2, $t8`
- `lb $s2, 37($t7)`
- `bne $v0, $t4, 732`
- `sra $s4, $s6, 11`

Exercise 2. [**10 marks**]

(a) Consider the following MIPS assembly code fragment. What is the correct value of `L2` to be used in the `beq` instruction? Assume the `lw` instruction labelled by `L1` is located at address `0x89aa1b`. *Show your workings.*

```
L1:
        lw   $t0, 0($a0)
        lw   $t1, 4($a0)
        lw   $t2, 0($a1)
        lw   $t3, 4($a1)
        slt  $t4, $t0, $t2
        slt  $t5, $t1, $t3
        and  $t4, $t4, $t5
        beq  $t4, $0, L2
        sub  $t3, $t3, $t2
        j    L3
L2:
        sub  $t3, $t2, $t3
L3:
        sw   $t3, 0($a0)
        jr   $ra
```

(b) Assume that, for a particular clock cycle in a single-cycle MIPS datapath, the program counter is `0x1258AB91` and the fetched instruction word is `0x089F01A7`. What is the value of the program counter on the next clock cycle? Write you answer in hexadecimal. *Show your workings.*

Exercise 3. [20 marks] Consider the following MIPS assembly code fragment. The `sort` function implements bubble sort using a helper function `swap`.

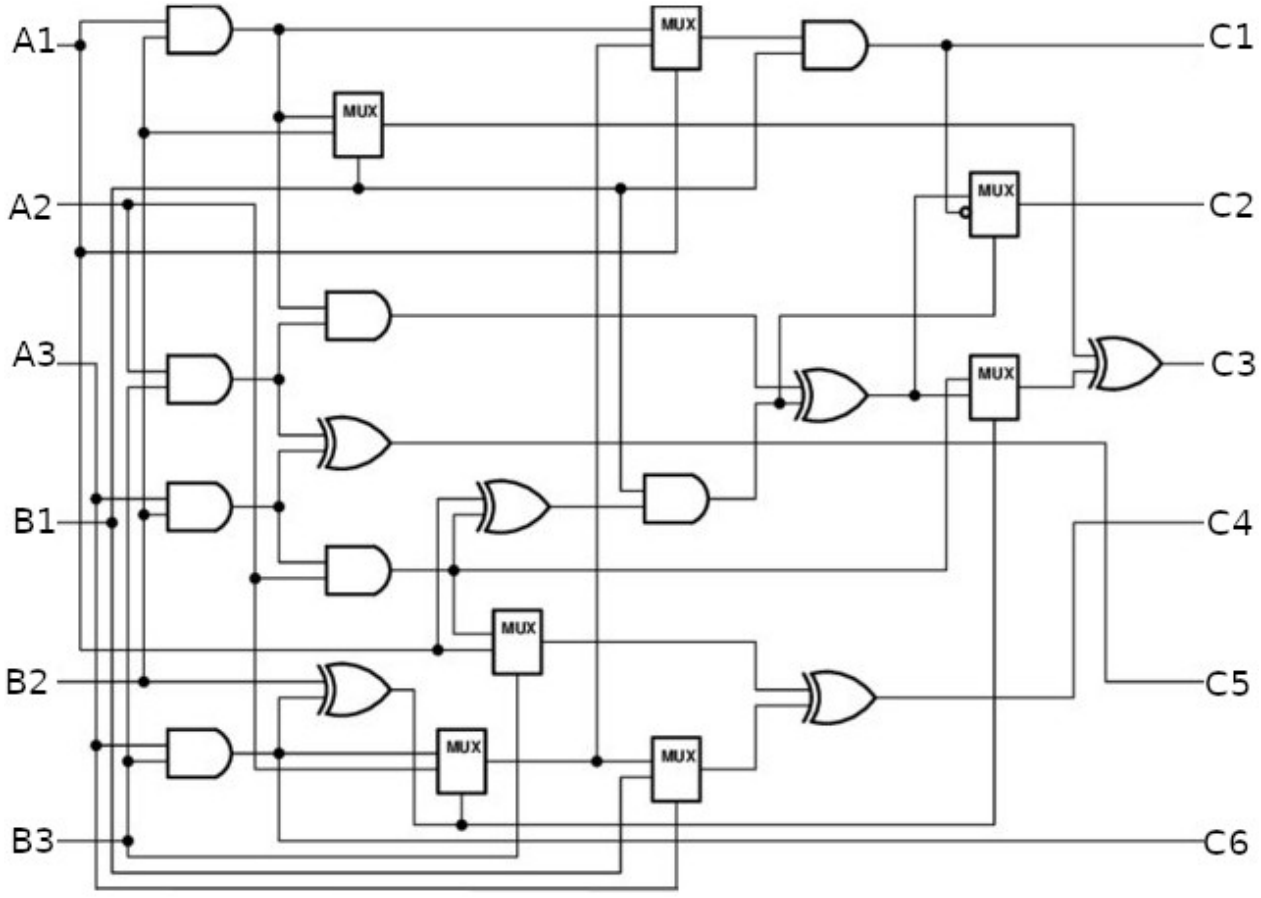
Assume there are no syntax errors. But, there are 10 bugs in this code with respect to the semantics of MIPS and runtime bugs. That is, there are exactly 10 lines of code that either need to be modified or added to get this code working correctly.

- If an instruction needs to be modified, state the instruction as is and then state what it should be modified to be.
- If an instruction needs to be added, state before which instruction it should be added and the instruction to add.

```
# swap k and k+1 of an array where a0 is array addreess and a1 is k
swap:
    add $t0, $v1, $zero #t0 = k
    add $t0, $v0, $t0   #t0 = v[k]
    lw $t1, 0($t0)      #load v[k]
    lw $s1, 4($t0)      #load v[k+1]
    sw $s1, 0($t0)      #v[k] = v[k+1]
    sw $t1, 4($t0)      #v[k+1] = v[k]
    jr $ra

# Bubble sort an array of ints with address a0 and length a1
sort:
    addi $sp, $sp, 16       #make room for ints on stack
    sw $s3, 12($sp)
    sw $s2,  8($sp)
    sw $s1,  4($sp)
    sw $s0,  0($sp)
    add $s2, $a0, $zero     #s2 = a0 (address of v array)
    add $s3, $a1, $zero     #s3 = a1 (length of v array)
    add $s0, $zero, $zero   #s0 = i = 0
for1tst:
    slt $t0, $s0, $s3       #t0 = (s0 < s3)
    beq $t0, $zero, exit1
    addi $s1, $s0, -1       #s1 = j = i-1
for2tst:
    slti $t0, $s1, 0        #t0 = (s1 < 0)
    bne $t0, $zero, exit2   #go to exit2 if (s1 < 0)
    sll $t1, $s1, 2         #t1 = j * 4
    add $t2, $s2, $t1       #t2 = v + (j*4)
    lw $t3, 0($t2)          #t3 = v[j]
    lw $t4, 4($t2)          #t4 = v[j+1]
    slt $t0, $t4, $t3       #t3 = v[j+1] < v[j]
    beq $t0, $zero, exit2   #go to exit2 if v[j+1] >= v[j]
    add $v0, $s2, $zero     #set args for swap proc
    add $v1, $s1, $zero
    j swap                  #call swap
    addi $s1, $s1, -1       #decrement j
    j for2tst               #iterate inner loop
exit2:
    addi $s0, $s0, 1        #increment i
    j for1tst               #iterate outer loop
exit1:
    lw $s0,  0($sp)
    lw $s1,  4($sp)
    lw $s2,  8($sp)
    lw $s3, 12($sp)
    addi $sp, $sp, -16
    jr $ra
```

Exercise 4. [14 marks]



Consider the above circuit, and assume the following timings for its constituent pieces.

- Transfer along wires is instantaneous.
- NOT operations are instantaneous (the small circle before the top-right MUX).
- Propagation delay of AND gate: 20ns
- Propagation delay of XOR gate: 45ns
- Propagation delay of MUX: 120ns

(a) Assume the values of A1, A2, A3, B1, B2, B3 all change to new values at time $t = 0$. For each of C1, C2, C3, C4, C5, C6, determine the time at which its new value becomes available. That is, determine each output's critical path.

(b) What is the propagation delay of this circuit as a whole?

Exercise 5. [20 Marks] Consider the single-cycle MIPS datapath with control signals as presented in Figure 1. We want to add a new instruction to the MIPS instruction set architecture: `foo`. Its specification is as follows:

MIPS assembly	RTL
`foo $rt $rs IMM`	Mem[Reg[$rs]] ← Reg[$rt] Reg[$rt] ← Mem[Reg[$rs] + Reg[$rt]] Reg[$rs] ← Reg[$rs] + IMM PC ← PC + 4

*Note that this is the normal "PC + 4" as for any non-branch, non-jump instruction.

Your task is to modify the MIPS datapath so that it can fulfill this new instruction `foo`. To do that, you should:

(i) add new wires, ports, circuitry, MUX, control signals, etc. to the datapath so that it can execute the new instruction `foo` (see, e.g., Slides 16-23 in L11-CPUControl);

(ii) ensure that any newly added circuitry and control signals do not hinder the execution of any existing operations in the MIPS ISA (i.e. your modified datapath should still be able to successfully execute all the preexisting instructions in the MIPS ISA).

Assume that `IMM` is a signed integer. Assume that memory is fast enough to read and write within one clock cycle, and that the read from memory occurs before the write to memory. To structure your solution to this exercise, perform the following:

(a) Modify (using Photoshop, Gimp, OneNote, etc.) the MIPS datapath (supplied as `MIPSDatapath.png`) with new circuits and control signals so that it can perform the `foo` instruction. That is, simply draw your new circuits and wires on top of the original image. Add your modifications in a colour other than black so they are clearly distinguishable from the original data path. Ensure you include bit-widths and labels.

(b) Give a brief English description of the modifications you have made. Explain how data flows through the modified datapath when the `foo` instruction is being executed. Explain how data flows through the modified datapath when any instruction besides `foo` is being executed.

(c) Give the values of the control signals required to execute your instruction. That is, when executing the `foo` instruction, give the values of the 8 preexisting control signals as well as the values of any new control signals you have added.

(d) If you have added any new control signals, give the values of these control signals when any instruction *besides* `foo` is being executed.

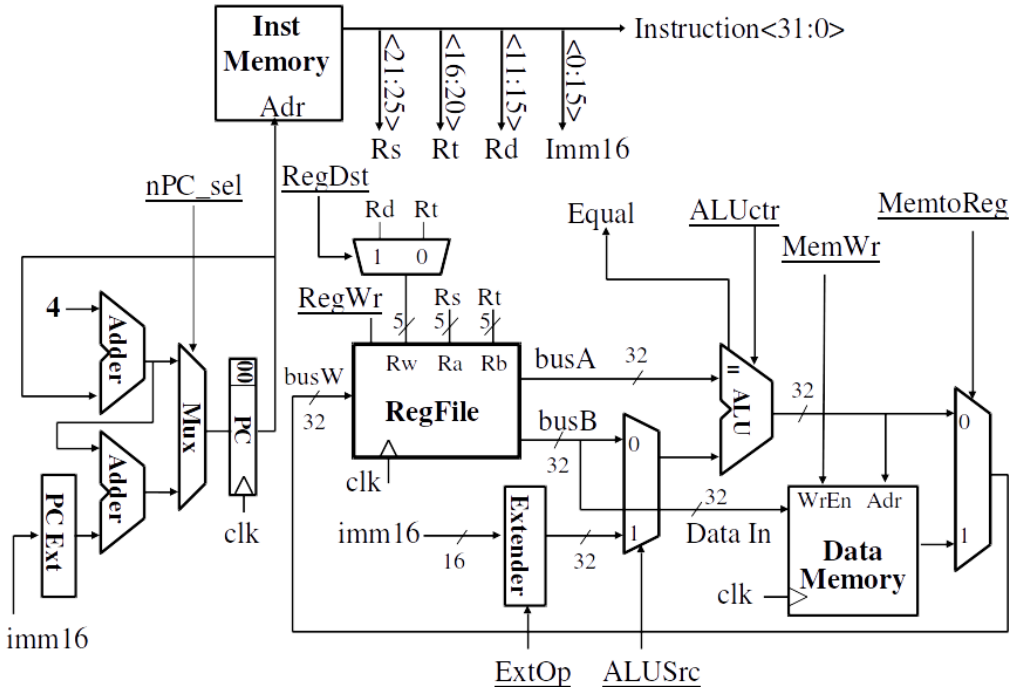
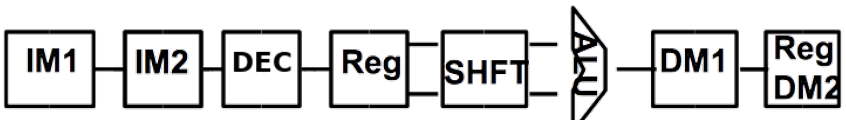


Figure 1: MIPS datapath with control signals

Exercise 6. Consider the following 8-stage datapath and the timings for each stage.



Stage	IM1	IM2	DEC	REG	SHFT	ALU	DM1	DM2
Time	115ps	225ps	175ps	200ps	150ps	425ps	200ps	225ps

(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) [6 marks] Assume this datapath was pipelined. What is the ideal speedup? Assuming no data dependencies or pipeline hazards, what is the actual speedup obtained when executing 300 instructions? Round your answers to 3 decimal places. As always, show your workings.

(d) [10 marks] Let us assume this datapath is not pipelined but still follows a multi-cycle clocking method. Let us further assume that the datapath has an additional optimization in which instructions skip stages that are unused for that instruction. Consider the following set of instructions to be executed on this datapath along with the stages which are used by that instruction.

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

`sll`: IM1, REG, SHFT, DM2

`sw`: IM2, DEC, REG, ALU, DM1

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

`beq`: IM1, IM2, REG

Instruction	*Instruction Count*
`add`	1500
`sll`	250
`sw`	500
`lw`	400
`beq`	350

Table 1: Program Instructions

Consider also a program composed of instructions as detailed in Table 1. If this program was to be executed on this datapath, determine: (*i*) the ideal CPI of this program, and (*ii*) the total execution time of this program. Assume that there are no dependencies or hazards between instructions. Assume there are no memory stalls.