♢ CSCI 2500 — Computer Organization ♢
Fall 2019 Quiz 3 (October 16, 2019)

Xinhao Luo	luox6@rpi.edu Lab section: 3
Room: Sage 3303 Zone: BLUE Row: 4 Seat: 7	7:00 pm - 7:50 pm

Please silence and put away all laptops, notes, books, phones, electronic devices, etc. This quiz is designed to take 50 minutes; therefore, for 50% extra time, the expected time is 1 hour and 15 minutes and 100% extra time is 1 hour and 40 minutes. Questions will not be answered except when there is a glaring mistake or ambiguity in the statement of a question. Please do your best to interpret and answer each question.

Xinhao Luo luox6@rpi.edu

1. **(20 POINTS)** Recall that prime numbers between 2 and 25 are as follows: 2, 3, 5, 7, 11, 13, 17, 19, and 23.

   Part a: (12/20 points) Write MIPS code to determine whether a given unsigned integer (passed as an argument) is prime or not:

   - Implement this as a procedure labeled `isprime`. The procedure must return 1 if prime and 0 otherwise.
   - You may assume the argument belongs to the range [2; 25].
   - You may hardcode the list of prime numbers in the data segment.
   - Your implementation must use register `$t0` but no other *t*-registers. (You may use any of the *s*-registers, if necessary.)
   - Make sure you follow all register usage conventions and save certain registers on the stack when it is required.

```
        .data
num:    .word 2,3,5,7,11,13,17,19,23,0

        .text  #$a0
isprime: addi $sp, $sp, -4
         sw $s0 ,0($sp)
         sw
         la $s0, num
         li $v0, 0
for:     lw $t0, 0($s1)
         beq $t0, 0, exit
         beq $t0, $a0, istrue
         addi $s0, $s0, 4
         j for
istrue:  li $v0, 1
exit:    lw $s0, 0($sp)
         addi $sp, $sp, 4
         jr $ra
```

Part b: (8/20 points) Consider the following code of the `main` procedure which calls your `isprime`:

```
        .data
primes:  .word 2, 3, 5, 7, 11, 13, 17, 19, 23
prime_str: .asciiz "PRIME\n"
not_prime_str: .asciiz "NOT PRIME\n"

        .text
        .globl main
main:
        # 1
        addi $sp, $sp, -8
        sw $ra, 0($sp)
        sw $a0, 4($sp)

        li   $v0, 5        # read_int
        syscall
        move $a0, $v0
        jal  isprime

        beq  $v0, $zero, p_np
        la   $a0, prime_str
        j    print
p_np:   la   $a0, not_prime_str
print:  li   $v0, 4        # print_str
        syscall

        # 2
        lw $a0, 4($sp)
        lw $ra, 0($sp)
        addi $sp, $sp, 8

        jr   $ra
```

Fill in the code below comment lines #1 and #2 to save/restore registers on the stack, when required.

2. **(20 POINTS)** Recall the following MIPS instructions:

Format: LB rt, offset(base) **MIPS I**

Purpose: To load a byte from memory as a signed value.

Description: rt ← memory[base+offset]

The contents of the 8-bit byte at the memory location specified by the effective address are fetched, sign-extended, and placed in GPR *rt*. The 16-bit signed *offset* is added to the contents of GPR *base* to form the effective address.

Format: LBU rt, offset(base) **MIPS I**

Purpose: To load a byte from memory as an unsigned value.

Description: rt ← memory[base+offset]

The contents of the 8-bit byte at the memory location specified by the effective address are fetched, zero-extended, and placed in GPR *rt*. The 16-bit signed *offset* is added to the contents of GPR *base* to form the effective address.

Format: SB rt, offset(base) **MIPS I**

Purpose: To store a byte to memory.

Description: memory[base+offset] ← rt

The least-significant 8-bit byte of GPR *rt* is stored in memory at the location specified by the effective address. The 16-bit signed *offset* is added to the contents of GPR *base* to form the effective address.

Xinhao Luo luox6@rpi.edu

Consider the program below executed on a little-endian processor:

```
        .data
val:    .word 0x08ff8005
nl:     .asciiz "\n"

        .text
        .globl main
main:   la    $s1, val
        lbu   $t0, 0($s1)
        lb    $t1, 0($s1)
        lbu   $t2, 1($s1)
        lb    $t3, 1($s1)
        # lw   $t1, 1($s1)
        move  $a0, $t0
        li    $v0, 1
        syscall                 # print_int
        la $a0, nl
        li $v0, 4
        syscall                 # print_str
        move  $a0, $t1
        li    $v0, 1
        syscall                 # print_int
        la $a0, nl
        li $v0, 4
        syscall                 # print_str
        move  $a0, $t2
        li    $v0, 1
        syscall                 # print_int
        la $a0, nl
        li $v0, 4
        syscall                 # print_str
        move  $a0, $t3
        li    $v0, 1
        syscall                 # print_int
        la $a0, nl
        li $v0, 4
        syscall                 # print_str

        jr $ra
```

Xinhao Luo luox6@rpi.edu

Part a: (6/20 points) What would be the values of registers `$t0`, `$t1`, `$t2`, and `$t3` after the program executes? Use binary or hexadecimal notation.

Register	Value
$t0	0x00000005
$t1	0x00000005
$t2	0xaaaaa80
$t3	0xffff80

Part b: (8/20 points) What would be the console output of this program?

5
5
128
-127

Part c: (2/20 points) If you uncomment the following line:

```
# lw   $t1, 1($s1)
```

and try executing the program, you get an error. Explain why this instruction is incorrect:

The given instruction takes a whole "word" and put into the given register. However, the offset 1 ~~does not~~ make the given address no longer a valid address for a full ".word". (4 bytes)

Part d: (4/20 points) MIPS ISA includes instructions `lb` (load byte) and `lbu` (load byte unsigned). There is also a matching `sb` (store byte) instruction. However there is no "unsigned" version of the store instruction. Give an explanation for why "unsigned" store byte instruction is not included in MIPS ISA whereas "unsigned" load instruction is.

Unsigned store byte will ~~lose~~ discard value's original value when storing it. ~~The~~ Storing should not change the sign of the value.

3. **(25 POINTS)** For all parts of this problem, clearly show all work to get full credit. You are given the following Boolean formula:

$$\overline{\overline{((A * \overline{B}) + C * 1 + B * 0)} + \overline{(A + \overline{C})} * (B + C)}$$

Part a: (5/25 points) Express the same function as a Sum of Products using only algebraic transformations (i.e., only applying axioms, laws, and identities of Boolean algebra):

$\Rightarrow \overline{(\bar{A}+B+C+0)+(\bar{A}\cdot C)\cdot(B+C)}$

$(A\cdot\bar{B}\cdot\bar{C})+(\bar{A}\cdot C)\cdot(B+C)$ $\quad \bar{A}\cdot B$

$(\bar{A}\cdot(A\cdot\bar{B}\cdot\bar{C})+C\cdot(A\cdot\bar{B}\cdot\bar{C}))\cdot(B+C)$

$0\cdot(B+C)$

0 (False) Circle

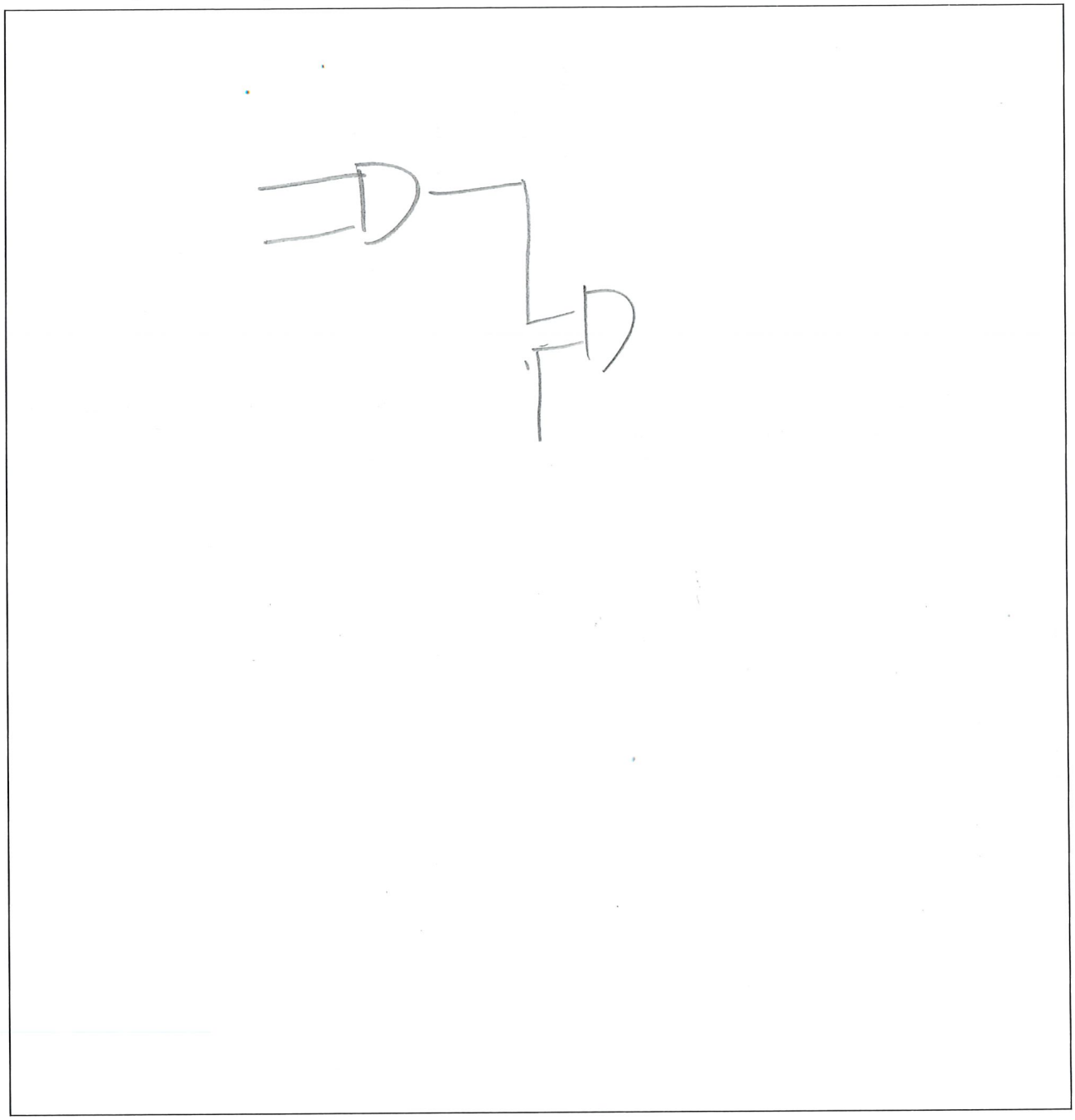
Part b: (5/25 points) Use truth tables to prove that your Sum of Products formula from Part a defines the same Boolean function as the one given by the original formula:

A	B	C	Sum	origin
T	T	T	F	F
T	T	F	F	F
T	F	T	F	F
T	F	F	F	F
F	T	T	F	F
F	T	F	F	F
F	F	T	F	F
F	F	F	F	F

the same.

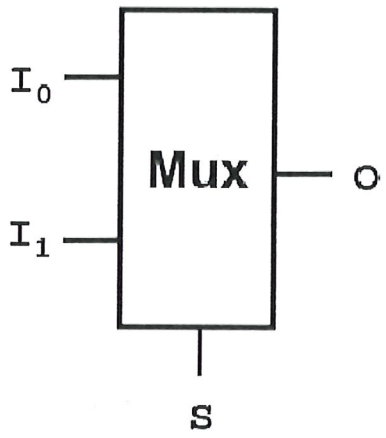
Part c: (5/25 points) Use the truth table from Part b to express this Boolean function as the Product of Sums:

Part d: (10/25 points) Implement this Boolean function (use any of the formulas but specify which formula you used) as a logic circuit using only 2-input AND, 2-input OR, and NOT gates. Clearly mark all input and output signals:

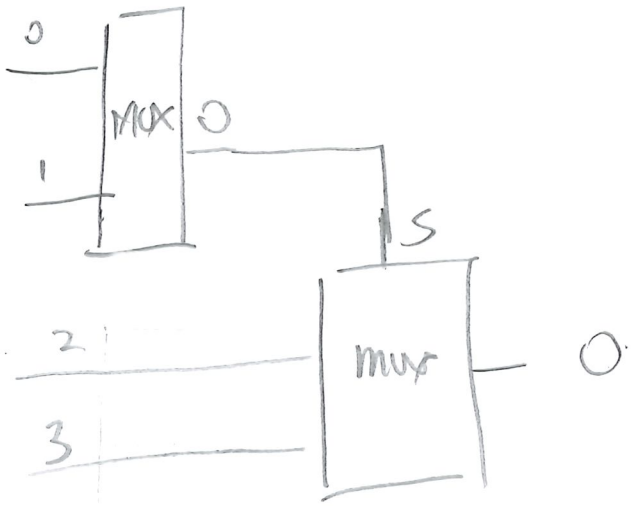


Xinhao Luo luox6@rpi.edu

4. **(15 POINTS)** Recall the 2-input multiplexor (2-Mux):



Design a logic circuit that implements a 4-input multiplexor using only 2-Mux'es (i.e., no other gates or components are allowed):



5. **(10 POINTS)** Which general form guarantees that the implementation of a Boolean function uses the minimal number of gates? Circle one:

   (a) Product of Sums
   (b) Sum of Products
   (c) Both Product of Sums and Sum of Products
   (d) Neither Product of Sums nor Sum of Products

6. **(10 POINTS)** Recall that a functionally complete set of Boolean operators is such a set that is sufficient to express any Boolean function.

   Which of the following operators are functionally complete sets (not necessarily minimal)? Circle all that apply:

   (a) AND, OR
   (b) AND, OR, NOT
   (c) AND, NOT
   (d) NOR
   (e) NAND
   (f) NAND, NOR
   (g) NAND, AND, OR, NOT