

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.004 Computation Structures

Fall 2020

Practice Quiz #1C

(Adapted from Quiz #1 Fall 2019)

1	/11
2	/12
3	/16
4	/17
5	/9
6	/12
7	/13

<i>Name</i>	<i>Athena login name</i>	<i>Score</i>
<i>Recitation section</i>		
<input type="checkbox"/> WF 10, 34-301 (Kathy)	<input type="checkbox"/> WF 12, 35-308 (Brian)	<input type="checkbox"/> WF 2, 34-303 (Quan)
<input type="checkbox"/> WF 10, 13-4101 (Ileana)	<input type="checkbox"/> WF 12, 36-155 (Jason)	<input type="checkbox"/> WF 2, 13-5101 (Domenic)
<input type="checkbox"/> WF 11, 34-301 (Kathy)	<input type="checkbox"/> WF 1, 35-308 (Brian)	<input type="checkbox"/> WF 3, 34-303 (Quan)
<input type="checkbox"/> WF 11, 13-4101 (Ileana)	<input type="checkbox"/> WF 1, 36-155 (Jason)	<input type="checkbox"/> WF 3, 13-5101 (Domenic)

Please enter your name, Athena login name, and recitation section above. Enter your answers in the spaces provided below. Show your work for partial credit. You can use the extra white space and the backs of the pages for scratch work.

Problem 1. Binary Arithmetic (11 points)

(A) (4 points) Express the values 53 and -14 in 8-bit 2's complement in both binary and hexadecimal.

53 in binary, 8-bit 2's complement encoding (0b):_____

53 in hexadecimal (0x):_____

-14 in binary, 8-bit 2's complement encoding (0b):_____

-14 in hexadecimal (0x):_____

(B) (2 points) Compute $53 - 14$ using 8-bit 2's complement arithmetic. You must show your work.

53-14 in 8-bit 2's complement encoding (show your work) (0b):_____

(C) (2 points) Could 53 and -14 both still be represented if we only used 5-bit 2's complement? For each value, if it can be done then express it in 5-bit 2's complement. If not, write **NOT POSSIBLE**.

53 in 5-bit 2's complement encoding (0b):_____

-14 in 5-bit 2's complement encoding (0b):_____

(D) (3 points) Compute the value of $\sim(0xAF \& 0xD7)$, where \sim is bitwise NOT and $\&$ is bitwise AND. Provide your result in both binary and hexadecimal. Show your work for partial credit.

Result in binary (0b):_____

Result in hexadecimal (0x):_____

Problem 2. RISC-V Assembly Language (12 points)

For each RISC-V instruction sequence below, provide the hex values of the specified registers after each sequence has been executed. **Assume that all registers are initialized to 0 prior to each instruction sequence.** Each instruction sequence begins with the line `(. = 0x0)` which indicates that the first instruction of each sequence is at address 0. Assume that each sequence execution ends when it reaches the `unimp` instruction.

(A) (3 points)

<pre>. = 0x0 lui x2, 2 addi x3, x2, 9 xori x4, x3, 0xffff end: unimp</pre>	<p>Value left in x2: 0x_____</p> <p>Value left in x3: 0x_____</p> <p>Value left in x4: 0x_____</p>
---	--

Sequences B and C refer to certain locations in memory. Assume that these memory locations have been initialized with the following data. The data consists of 3 words beginning at address 0x100, another 3 beginning at address 0x200, and finally a third set beginning at address 0x1000.

```
// Shared Data
. = 0x100
// First 3 words at address 0x100
.word 0x60046004
.word 0x12345678
.word 0x87654321

. = 0x200
// First 3 words at address 0x200
.word 0x11110000
.word 0xA0A0FFFF
.word 0x77773333

. = 0x1000
// First 3 words at address 0x1000
.word 0x11111111
.word 0xCCCCCCCC
.word 0xFFFFFFFF
```

(B) (6 points)

<pre> . = 0x0 li x1, 0x100 slli x2, x1, 1 lw x3, 0x4(x2) srai x4, x3, 4 bgez x4, L1 addi x5, x1, 4 L1: addi x5, x5, 4 unimp </pre>	<p>Value left in x1: 0x_____</p> <p>Value left in x2: 0x_____</p> <p>Value left in x3: 0x_____</p> <p>Value left in x4: 0x_____</p> <p>Value left in x5: 0x_____</p> <p>Address of label L1: 0x_____</p>
--	--

(C) (3 points)

<pre> . = 0x0 li x1, 0x100 li x2, 0x208 lw x6, 0x4(x1) mv x7, x1 sw x2, 0(x7) lw x8, 0(x2) unimp </pre>	<p>Value left in x6: 0x_____</p> <p>Value left in x7: 0x_____</p> <p>Value left in x8: 0x_____</p>
---	--

Problem 3. RISC-V Calling Conventions (16 points)

The following functions are missing code to save/restore registers to/from the stack. Identify the registers that need to be saved to the stack *at any point* in the function. **You do not need to provide the missing code. If the RISC-V calling convention does not require any registers to be saved to / restored from the stack, write "none".**

(A) (2 points)

```
// register save/restore code missing
function:
    add    zero, s0, s1
    ret
```

registers needing saving (or "none"):

(B) (2 points)

```
// register save/restore code missing
funciona:
    beqz   a0, done
    mv     s0, a0
    addi   a0, a0, -1
    jal    funciona
    // use return value of funciona:
    add    a0, s0, a0
done:
    ret
```

registers needing saving (or "none"):

(C) (2 points)

```
// register save/restore code missing
function:
    li     t0, 0x5F3759DF
    xor    t0, a0, t0
loop:
    beqz   t0, end
    srli   t0, t0, 1
    andi   t0, t0, 0x1
    beqz   t0, lsb_zero
    addi   s0, s0, 1
    j      loop
lsb_zero:
    addi   s1, s1, 1
    j      loop
end:
    ret
```

registers needing saving (or "none"):

(D) (2 points)

// register save/restore code missing
functio:

// all t registers

lw t0, 0(a0)

lw t1, 4(a0)

lw t2, 8(a0)

lw t3, 12(a0)

lw t4, 16(a0)

lw t5, 20(a0)

lw t6, 24(a0)

// all s registers

lw s0, 28(a0)

lw s1, 32(a0)

lw s2, 36(a0)

lw s3, 40(a0)

lw s4, 44(a0)

lw s5, 48(a0)

lw s6, 52(a0)

lw s7, 56(a0)

lw s8, 60(a0)

lw s9, 64(a0)

lw s10, 68(a0)

lw s11, 72(a0)

// all a registers except a0

lw a1, 76(a0)

lw a2, 80(a0)

lw a3, 84(a0)

lw a4, 88(a0)

lw a5, 92(a0)

lw a6, 96(a0)

lw a7, 100(a0)

lw a0, 104(a0)

ret

registers needing saving (or "none"):

One variant of the *Ackermann function* is defined for non-negative integers m and n :

```
int ackermann(int m, int n) {  
    if (m == 0)  
        return n + 1;  
    else if (n == 0) // m_nonzero  
        return ackermann(m - 1, 1);  
    else // both_nonzero  
        return ackermann(m - 1, ackermann(m, n - 1));  
}
```

Ben Bitdiddle has implemented the function in RISC-V assembly. However, his lack of stack discipline means that his implementation doesn't quite work.

(E) (2 points) What is the **minimum** value of $\langle X \rangle$, the number of bytes needed on the stack?

Minimum bytes on stack needed = $\langle X \rangle$ = _____ bytes

(F) (6 points) Fill in the instructions needed to make Ben's code implement the function above by employing the RISC-V calling convention.

The only missing code has to do with the calling convention.

You may use no more than one RISC-V instruction (or pseudoinstruction) per line.

You may not insert instructions where there are no lines. You may not need all lines.

For full credit, use the fewest number of instructions needed to implement the function.

```
ackermann:  
    _____  
    bnez a0, m_nonzero  
    addi a0, a1, 1  
    _____  
    ret  
m_nonzero:  
    addi sp, sp, -<X> // part (E)  
    _____  
    bnez a1, both_nonzero  
    addi a0, a0, -1  
    li a1, 1  
    call ackermann  
    _____  
    _____  
    addi sp, sp, <X> // part (E)  
    ret  
both_nonzero:  
    _____  
    _____  
    _____  
    // inner call  
    addi a1, a1, -1  
    call ackermann  
    _____  
    _____  
    _____  
    // outer call  
    addi a0, a0, -1  
    call ackermann  
    _____  
    _____  
    addi sp, sp, <X> // part (E)  
    ret
```

Problem 4. Stack Detective (17 points)

Below is the C code for a recursive version of the `factorial` function from Lab 2. To the right is a translation to RISC-V assembly language. (As in Lab 2, the `mul` function is provided to you and obeys the RISC-V calling convention, but you can't see its implementation.)

```
int mul(int a, int b) {
    return a * b;
}

int factorial(int n) {
    if (n <= 1) return 1;
    else {
        int f1 = factorial(n-1);
        return mul(f1, n);
    }
}
```

```
factorial:
    addi a1, zero, 1
    bgt a0, a1, recurse
L2:    mv a0, a1
    ret

recurse:
    addi sp, sp, -8
    sw ra, 0(sp)
    sw a0, 4(sp)
    addi a0, a0, -1
    call factorial
    lw a1, 4(sp)
    call mul
L1:    lw _____
    addi sp, sp, 8
    ret
```

- (A) (2 points) What should be in the blank on the line labeled **L1** to make the function operate correctly?

L1: lw _____

- (B) (1 point) How many words will be written to the stack before the program makes each recursive call to the factorial function?

Number of words pushed onto stack before each recursive call? _____

The factorial code above and a program that calls it once (using 'call factorial' and **with an argument greater than 1**) are loaded into the RISC-V simulator. A breakpoint is set on the line labeled **L2** and the program is executed until it reaches the breakpoint. The diagram below shows a small piece of the stack at this point. All addresses and data values are shown in hex. **The current value in the SP register is 0xF7D8 and points to the location shown in the diagram.**

Memory Contents

- (C) (3 points) Fill in each of the blanks to the right of the diagram to indicate which register the value came from when it was pushed onto the stack. Write an X if it is not possible to determine from the information you have.

Address	Data	
0xF7D4	0x2	_____
SP → 0xF7D8	0x80	_____
0xF7DC	0x2	_____

(D) (2 points) What is the hex address of the instruction at label **L1**?

Address of instruction at label **L1**: 0x _____

The program's execution is resumed and it is interrupted some time later, *just prior* to the execution of the instruction you completed at label **L1**. (**Note that this may not be the first time the simulator reached this instruction**). As before, the diagram on the right shows the current contents of a portion of the stack and the current location pointed to by SP.

(E) (4 points) What is the address of the instruction that made the initial call to factorial and what was the value of argument *n* for that call? Write CAN'T TELL if the argument does not show up in the stack.

Initial argument to factorial: *n* = _____

Address of instruction that made initial call to factorial:

0x _____

(F) (3 points) Based on what you see in the stack, does the mul function save anything there? Circle the appropriate response.

Memory Contents

Address	Data
0xF800	0x100
0xF804	0x44
SP→ 0xF808	0x80
0xF80C	0x8
0xF810	0x80
0xF814	0x9
0xF818	0xABC
0xF81C	0xA
0xF820	0xCBA
0xF824	0xB

YES NO CAN'T TELL

(G) (2 points) What is the address of the mul function. Write CAN'T TELL if you cannot tell.

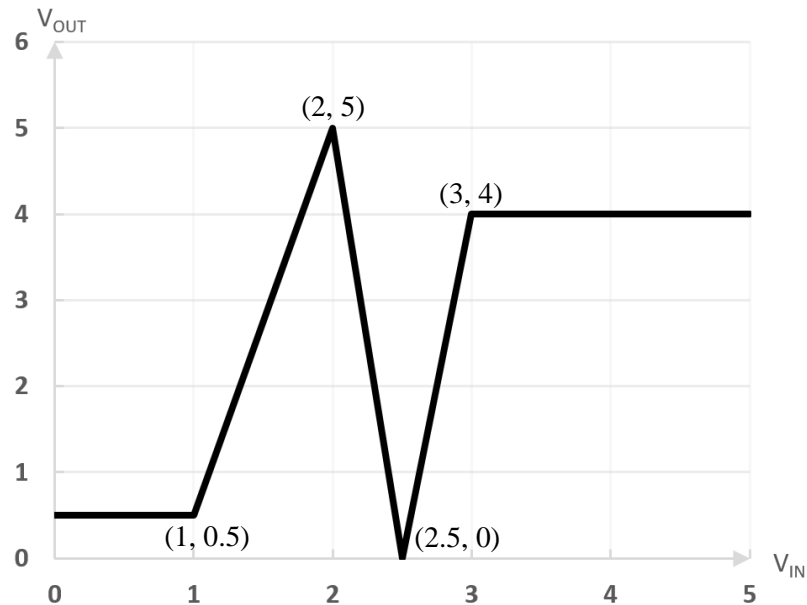
Address of mul function: 0x _____

Problem 5. Static Discipline (9 points)

At work, you are given a device by your boss, who says “This is the device that will change buffers forever!” You measure its voltage transfer characteristic (VTC), shown below.

You are then given three different signaling specifications for logic values to voltages, denoted by the incomplete table below. Noise immunity is defined as the smaller of the low and high margins.

Complete the table by filling in values to maximize the noise margin of the respective scheme. **If the numbers in a scheme cannot be chosen to allow the device to function as a buffer with positive noise margins, fill in the entries for that column with Xs.**



	Scheme A	Scheme B	Scheme C
V_{OL}	0.5		
V_{IL}		2	1
V_{IH}	2.5		3
V_{OH}		4	
Noise immunity			

Problem 6. Boolean Algebra (12 points)

(A) (8 points) Simplify the following Boolean expressions by finding a minimal sum-of-products expression for each one. (Note: These expressions can be reduced into a minimal SOP by repeatedly applying the Boolean algebra properties we saw in lecture.)

1. $\overline{(\bar{x} + \bar{y} + \bar{y}z)}$

2. $x + y(x + \overline{xz})$

3. $\bar{x} + \bar{y} + xy$

4. $xy(\bar{x} + z) + \bar{y}z$

(B) (4 points) For each of the following sets of Boolean gates, decide if they are functionally complete: can one implement any Boolean function using only the specified gates?

1. {AND, OR} **Yes No**

2. {AND, NOT} **Yes No**

3. {NOR} **Yes No**

4. {XOR, NOT} **Yes No**

Problem 7. CMOS Logic (13 points)

We have a mystery function $F(a,b,c)$ that is implemented from a single CMOS gate. Recall from lecture that a CMOS gate consists of an output node connected to a single pFET-based pullup circuit and a single nFET-based pulldown circuit.

(A) (3 points) We know that $F(0, 1, 1) = 1$. What can you say about the following values?

(circle one) $F(0, 0, 1) = : 0 \dots 1 \dots$ (can't say)

(circle one) $F(1, 1, 1) = : 0 \dots 1 \dots$ (can't say)

(circle one) $F(1, 1, 0) = : 0 \dots 1 \dots$ (can't say)

(B) (6 points) Now let us consider two Boolean expressions, $G_1(a,b,c)$ and $G_2(a,b,c)$. Given that F can be implemented from a single CMOS gate, state whether it is possible for each expression to be F . If it is possible, then draw the CMOS gate.

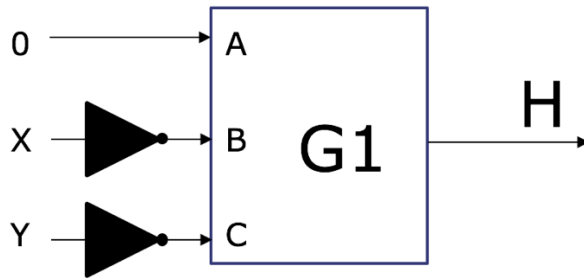
(i) $G_1(a, b, c) = (a \wedge b) + c$ **Possible** **Not possible**

CMOS gate (if possible):

(ii) $G_2(a, b, c) = \overline{(a \cdot b)} + \bar{c}$ **Possible** **Not possible**

CMOS gate (if possible):

(C) (4 points) We are now interested in function $H(x,y,z)$, which computes $G_1(0, \bar{x}, \bar{y})$.



(i) Can H be implemented as a single CMOS gate?

Yes No

(ii) If yes, draw the CMOS gate below. If no, explain why not.

END OF QUIZ 1!