1	/10
2	/14
3	/15
4	/16
5	/13
6	/19
7	/13

M A S S A C H U S E T T S   I N S T I T U T E   O F   T E C H N O L O G Y
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.004 Computation Structures
Spring 2020

Quiz #1

Name	*Athena login name*	*Score*

Recitation section

☐ WF 10, 34-302 (Domenic)	☐ WF 12, 34-302 (Alan)	☐ WF 2, 34-302 (Miles)
	☐ WF 12, 35-310 (Brian)	☐ WF 2, 8-205 (Shahul)
☐ WF 11, 34-302 (Domenic)	☐ WF 1, 34-302 (Alan)	☐ WF 3, 34-302 (Miles)
	☐ WF 1, 35-310 (Brian)	☐ WF 3, 8-205 (Shahul)

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

Problem 1. Binary Arithmetic (10 points)

(A) (2 points) What is the maximum decimal value that can be represented in **7-bit unsigned binary**? What is the minimum decimal value that can be represented in **6-bit 2's complement**?

Largest 7-bit unsigned binary (in decimal):____________________

Smallest 6-bit 2's complement number (in decimal):____________________

(B) (4 points) What is the value of (~(0x56) & 0x3D) | 0x42, where **~** is bitwise NOT, & is bitwise AND, and **|** is bitwise OR? Provide your result in both 8-bit binary and hexadecimal. Show your work.

Result in binary (0b):________________________

Result in hexadecimal (0x):________________________

(C) (2 points) Express the values 75 and 23 in 8-bit 2's complement

75 in 8-bit 2's complement notation (0b):____________________

23 in 8-bit 2's complement notation (0b):____________________

(D) (2 points) Compute 75 – 23 using 8-bit 2's complement arithmetic. **You must show your work using 2's complement arithmetic.**

75 – 23 in 8-bit 2's complement notation (0b):____________________

Problem 2. RISC-V Assembly Language (14 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) (5 points)

```
. = 0x0
     lui x1, 3
     srai x2, x1, 1
     li x1, 0x6
     slli x3, x1, 2
     xor x4, x1, x3
     mv x0, x1
end: unimp
```

Value left in x0: 0x______________

Value left in x1: 0x______________

Value left in x2: 0x______________

Value left in x3: 0x______________

Value left in x4: 0x______________

(B) (3 points)

```
. = 0x0
     jal x5, L1
     jal x6, end
L1:  j L2
     jal x6, end
L2:  jr x5
end: unimp
```

Value left in x5: 0x______________

Value left in x6: 0x______________

Address of label L2: 0x______________

Sequence C refers to certain locations in memory. Assume that the first 4 memory locations starting from address 0x600 have been initialized with the following 4 words.

```
    . = 0x600
    // First 4 words at address 0x600
    .word 0x60046004
    .word 0x87654321
    .word 0x12345678
    .word 0x00000001
```

(C) (6 points)

```
    . = 0x0
          li x7, 0x600
          mv x8, x7
    loop: addi x8, x8, 4
          lw x9, 0(x8)
          sw x9, -4(x8)
          blez x9, loop
          lw x7, 0(x7)
    end:  unimp
```
	Value left in x7: 0x______________
	
	Value left in x8: 0x______________
	
	Value left in x9: 0x______________

Problem 3. RISC-V Calling Conventions (15 points)

Two programmers, ZoomBa and Nim, are trying to write a certain array-processing program together. Both ZoomBa and Nim wrote their own functions in Python.

However, due to MTI's collaboration policy, ZoomBa and Nim are only allowed to share code written in RISC-V Assembly Language. Moreover, Nim was terrible at translating their code into the Assembly Language, so their code crashes when run together.

You just successfully hacked MTI's super secure and fast WIFI, and you saw ZoomBa and Nim sending these Assembly Codes around:

ZoomBa's Code:

```
// convert an input integer with
// a certain function
// I am sure this function works
// correctly!
map:
  li a1, 1
  sll a0, a1, a0
  ret
```

Nim's Code:

```
// array processing
// pseudocode is provided in the
// next page
array_process:
  li t1, 0
  mv s2, a0
  mv s3, a0
loop:
  beq t1, a1, end
  lw a0, 0(s2)
  call map
  sw a0, 0(s2)
  addi s2, s2, 4
  addi t1, t1, 1
  j loop
end:
  mv a0, s3
  ret
```

You tried testing their assembly code together, and of course, it crashed. You then started to decipher their codes and fix them.

(A) (3 points) Luckily, you hacked MTI's WIFI again and got some parts of their pseudo code written in Python, but some parts are still missing. **Please fill in the blank to make the following Python code have the same functionality as what ZoomBa and Nim intended to write.**

The part in the blank **should be a mathematical expression of x alone and should involve using only Python mathematical operations of `+, -, *, /, // (integer division), or ** (power)`**.

ZoomBa's Code:	**Nim's Code:**

`// convert an input integer with`
`// a certain function`
`// I am sure this function works`
`// correctly!`
`def map(x):`
`  return ____________________`
	`// array processing`
	`// pseudo code is provided below`
	`def array_process(array, size):`
	`  for i in range(size):`
	`    array[i] = map(array[i])`
	`  return array`

(B) (12 points) Now since you notice that these codes violate calling conventions in many places, you decided to fix them all by saving additional registers onto the stack and loading them back appropriately.

Please add appropriate instructions (**either Increment/Decrement stack pointer, Load word from stack, or Save word to stack only**) into the blank spaces on the next page to make these codes follow the calling convention. You may not need to use all the spaces provided.

Your answer should still follow calling convention **even if ZoomBa changes his map function** to perform something else (that follows the calling convention).

For full credit, you should **only save registers that must be saved onto the stack and avoid unnecessary loads and stores** while following the calling conventions.

Nim’s Code:

```
array_process:
  li t1, 0
  mv s2, a0
  mv s3, a0
loop:
  beq t1, a1, end
  lw a0, 0(s2)
  call map
  sw a0, 0(s2)
  addi s2, s2, 4
  addi t1, t1, 1
  j loop
end:
  mv a0, s3
  ret
```

Answer:

```
array_process:
  li t1, 0

  ____________________

  ____________________

  ____________________

  ____________________

  ____________________

  ____________________
  mv s2, a0
  mv s3, a0
loop:
  beq t1, a1, end

  ____________________

  ____________________
  lw a0, 0(s2)
  call map
  sw a0, 0(s2)

  ____________________

  ____________________
  addi s2, s2, 4
  addi t1, t1, 1
  j loop
end:
  mv a0, s3
  ____________________

  ____________________

  ____________________

  ____________________

  ____________________
  ret
```

Problem 4. Stack Detective (16 points)

Below is the Python code for a recursive implementation of binary search, which finds the index at which an element should be inserted into a sorted array to ensure that the array is still sorted after the insertion. To the right is a not so elegant, but valid, implementation of the function using RISC-V assembly.

```
/* find where to insert element in arr */
def binary_search(arr, start, end, element):
    if (start == end):
       return end
    mid = (start + end)//2
    if element < arr[mid]:
       end = mid
    else:
       start = mid + 1
    return binary_search(arr, start,
                         end, element)
```

```
binary_search: addi sp, sp, -8
               sw ra, 4(sp)
               sw s0, 0(sp)
               mv s0, a2
               beq a1, a2, done
               add t0, a1, a2
               srli t0, t0, 1
               slli t1, t0, 2
               add t1, a0, t1
               lw t1, 0(t1)
if:            bge __________
               mv a2, t0
               j recurse
else:          addi t0, t0, 1
               mv a1, t0
recurse:       call binary_search
               mv s0, a0
done:          mv a0, s0
               lw s0, 0(sp)
               lw ra, 4(sp)
L1:            addi sp, sp, 8
               ret
```

(A) (2 points) What should be in the blank on the line labeled **if** to make the assembly implementation match the Python code?

if: bge ____________________________

(B) (2 points) How many words will be written to the stack before the program makes each recursive call to the function **binary_search**?

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

The program's initial call to function **binary_search** occurs outside of the function definition via the instruction '**call binary_search**'. The program is interrupted *during a recursive call* to **binary_search**, *just prior* to the execution of '**addi sp, sp, 8**' at label **L1**. The diagram on the right shows the contents of a region of memory. All addresses and data values are shown in hex. **The current value in the SP register is 0xEB0 and points to the location shown in the diagram.**

Memory Contents

	Address	Data
	0xEA4	0x0
	0XEA8	0x5
	0XEAC	0xC4
SP→	0XEB0	0x6
	0xEB4	0xC4
	0xEB8	0x6
	0xEBC	0xC4
	0xEC0	0xA
	0xEC4	0xC4
	0xEC8	0x3E
	0xECC	0xCA4
	0xED0	0xCED

(C) (4 points) What were the values of arguments `arr` and `end` at the beginning of *the initial call* to **binary_search**? Write CAN'T TELL if you cannot tell the value of an argument from the stack provided.

Arguments at beginning of this call : `arr` = **0x**_______________

`end` = **0x**_______________

(D) (4 points) What are the values in the following registers right when the execution of **binary_search** is interrupted? Write CAN'T TELL if you cannot tell.

Current value of s0: 0x_______________

Current value of ra: 0x_______________

(E) (2 points) What is the hex address of the '**call binary_search**' instruction that made the *initial call* to **binary_search**?

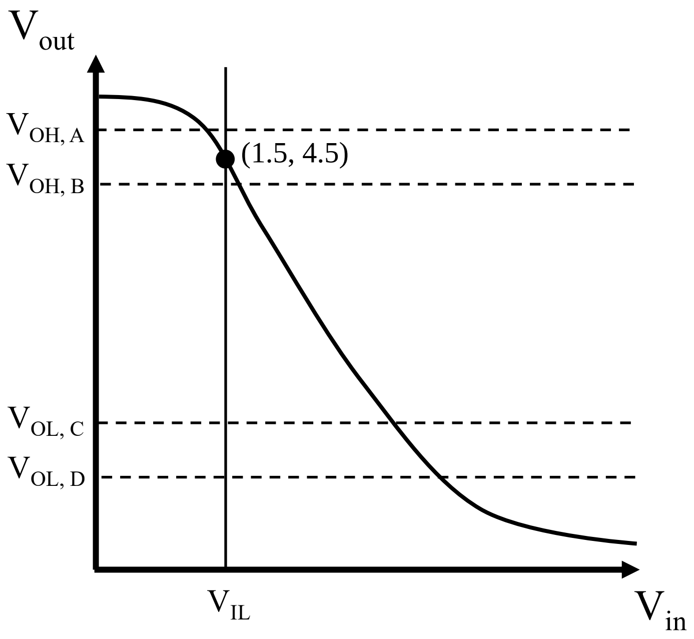
Address of instruction that made initial call to binary_search: 0x _______________

(F) (2 points) What is the hex address of the `ret` instruction?

Address of `ret` instruction: 0x _______________

Problem 5. Static Discipline (13 points)

Consider the voltage transfer characteristic (VTC) of a hypothetical Device F shown below (not to scale). Ben Bitdiddle is considering two different choices for each of V_{OH} and V_{OL}, as indicated by the dashed lines on the graph below. The precise values of these thresholds are listed in the table. V_{IL} is known to be 1.5V.



Variable	**Value (V)**
$V_{OH,A}$	4.7
$V_{OH,B}$	4.2
$V_{OL,C}$	1.8
$V_{OL,D}$	1.0
V_{IL}	1.5

(A) (4 points) V_{IH} is not known right now but is guaranteed to be between 3.0 V and 4.0 V. For each choice of V_{OH}, circle **YES** if it both follows the static discipline and has a positive **high noise margin**, or circle **NO** if it does not satisfy both of these conditions. If the answer depends on knowing values that are not provided, then circle **DON'T KNOW**.

$V_{OH,\ A}$ (circle one): **YES** **NO** **DON'T KNOW**

$V_{OH,\ B}$ (circle one): **YES** **NO** **DON'T KNOW**

(B) (4 points) For each choice of V_{OL}, circle **YES** if it both follows the static discipline and allows for a positive **low noise margin**, or circle **NO** if it does not satisfy both of these conditions. If the answer depends on knowing values that are not provided, then circle **DON'T KNOW**.

$V_{OL,\ C}$ (circle one): **YES** **NO** **DON'T KNOW**

$V_{OL,\ D}$ (circle one): **YES** **NO** **DON'T KNOW**

(C) (2 points) Suppose Ben settles on the following signaling specification which follows the static discipline:

$V_{IL} = 1.5$ V
$V_{IH} = 3.8$ V
$V_{OL} = 1.4$ V
$V_{OH} = 4$ V

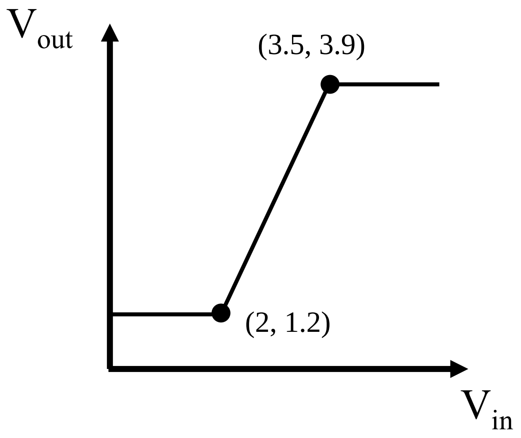
Calculate the high and low noise margins as well as the noise margins of the device as a whole.

High Noise Margin: ____________________ V

Low Noise Margin: ____________________ V

Overall Noise Margin: ___________________ V

(D) (3 points) Alyssa P. Hacker has another Device G with the following VTC that she would like to use with Device F. Can she use Device G with Device F and the signaling specifications described in part (C)? If she can, circle **NO CHANGES**. Otherwise, circle **CHANGES NEEDED** and change **exactly one** of the thresholds of the signaling specification to a new value such that Device G can be used while obeying the static discipline and maximizing noise margins. Also calculate the overall noise margin of the resulting system, regardless of whether or not you changed any thresholds. Keep in mind that VTCs may touch the edge of, but not enter the forbidden region.



Are changes needed? (circle one): **NO CHANGES** **CHANGES NEEDED**

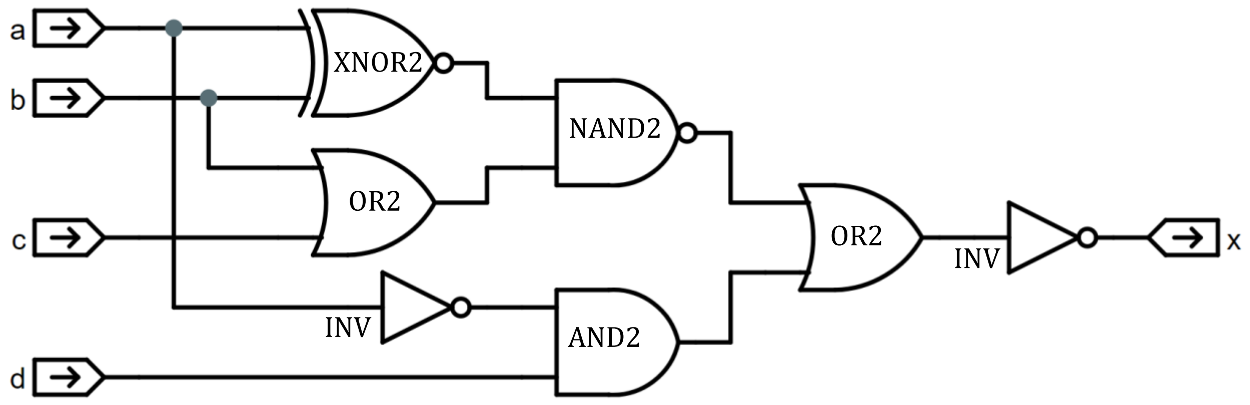
Threshold to Change (circle one): **V_{IL}** **V_{IH}** **V_{OL}** **V_{OH}**

Value to change to: __________________ V

Overall Noise Margin: __________________ V

Problem 6. Boolean Algebra and Combinational Logic (19 points)

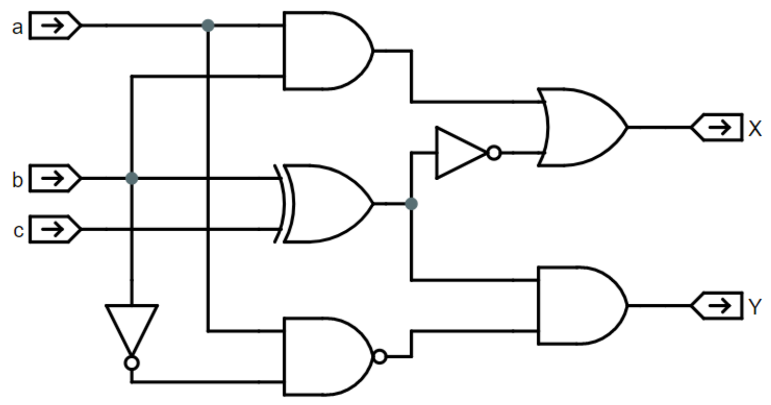
(A) (3 points) Consider the logic diagram below, which includes XNOR2, OR2, NAND2, AND2, and INV. Using the t_{PD} information for the gate components shown in the table below, compute the t_{PD} for the circuit.



Gate	t_{PD}
XNOR2	7.0ns
OR2	5.5ns
NAND2	3.0ns
AND2	5.0ns
INV	2.0ns

t_{PD} (ns) = ______________________________

(B) (6 points) Given the circuit shown below, construct the truth table for outputs **X** and **Y.**



a	b	c	X	Y
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

(C) (4 points) Find a minimal sum-of-products expression for output **X** of the circuit described by the truth table shown below.

a	b	c	d	X
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Minimal sum of products for X = ______________________________

(D) (6 points) For each of the following expressions determine if it is satisfiable. If satisfiable, provide a minimal sum-of-products. Otherwise, show why it is not satisfiable.

1. $\overline{\bar{c}(a+b)(a+d)}(ab\bar{c})$

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

Problem 7. CMOS (13 points)

One day, you are given a mysterious circuit, $D(a, b, c)$.You are told that it is implemented using a single CMOS gate, consisting of an output connected to a single pFET-based pullup circuit and a single nFET-based pulldown circuit.

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

(circle one) $\boldsymbol{D(1, 1, 1) =}$ **0 … 1 … (can't say)**

(circle one) $\boldsymbol{D(0, 0, 1) =}$ **0 … 1 … (can't say)**

(circle one) $\boldsymbol{D(0, 1, 0) =}$ **0 … 1 … (can't say)**

(B) (6 points) You find another circuit, $E(a, b, c)$, and are told that it can be implemented **using a single CMOS gate**. You are given a truth table for two new functions F and G. For each function, determine whether it is possible for the function to be E. If it is possible, draw the CMOS gate.

A	B	C	F	G
0	0	0	1	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	1
1	1	1	0	0

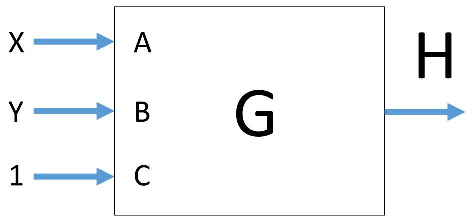
i. **F** **(circle one) Possible Not Possible**

CMOS GATE (if possible):

ii. **G** **(circle one) Possible Not Possible**

CMOS GATE (if possible):

(C) (4 points) Now, we are interested in $H(x, y)$, which computes $G(x, y, 1)$.



i. Can H be implemented as a single CMOS gate?

Circle one: YES NO

ii. If yes, draw the CMOS gate below. If not, explain why not.

END OF QUIZ 1!