Problem 1. Binary Arithmetic (11 points)

(A) (4 points) Express the following decimal values as hexadecimal numbers encoded in 9-bit 2’s complement. If you cannot express a value in 9-bit 2’s complement, write “Not Possible.”

(i) -259

-259 in 9-bit 2’s complement (hexadecimal): 0x______________

(ii) 147

147 in 9-bit 2’s complement (hexadecimal): 0x______________
(B) (4 points) Calculate the result of 0xCB – 0xD9, where both values are in 8-bit 2’s complement. **You must show your work using 2’s complement arithmetic.** Provide your answer in 3 forms: 1) as an 8-bit 2’s complement number, 2) as a 2’s complement number using the fewest bits necessary to represent the same numerical value, 3) in decimal.

\[ 0xCB - 0xD9 \text{ in 8-bit 2’s complement binary notation: } 0b____________ ]

\[ 0xCB - 0xD9 \text{ in fewest number of bits 2’s complement binary notation: } 0b____________ ]

\[ 0xCB - 0xD9 \text{ in decimal notation: } ______________ \]

(C) (3 points) Compute the result of evaluating \( ((\sim0x18) ^ 0x5B) >>_a 2 \), where \( \sim \) is bitwise NOT, ^ is bitwise XOR, and \( >>_a \) is arithmetic right shift. Provide both the result of the computation prior to shifting by 2, and the final result in hexadecimal notation.

\[ \text{Result of } (\sim0x18) ^ 0x5B: \ 0x____________ ]

\[ \text{Final result of } ((\sim0x18) ^ 0x5B) >>_a 2: \ 0x____________ ]
Problem 2. RISC-V Assembly (14 points)

(A) (2 points) An assembled RISC-V binary uses an R-type instruction with hexadecimal encoding 0x00560533. The binary encoding of an R-type instruction is shown below. What is the assembly instruction? Use the **symbolic names** (e.g., a0-a7, t0-t6) of the registers in your answer.

![Binary Encoding Table]

**Assembly instruction:** ________________________

For each of the following code snippets, give the values in the registers after executing the code from the start up to the end label. Write **CAN'T TELL** if it is impossible to tell the value of a register. The code snippets are independent of each other.

(B) (3 points)

```
start:
    li x1, 0x14
    lui x2, 0x33
    slli x1, x1, 2
    addi x2, x2, -1
    bge x2, x1, end
    mv x3, x1
end:
```

x1: 0x_______________________________

x2: 0x_______________________________

x3: 0x_______________________________
(C) (5 points) For this code snippet, provide the resulting register values and the resulting memory values.

```
start:
    li x1, 0x1004
    lw x2, -4(x1)
    lw x3, 4(x1)
    sw x2, 4(x1)
    sw x3, -4(x1)
end:

. = 0x1000
.word 0x60046004
.word 0xB0BABABE
.word 0x01234567
```

x2: 0x_______________________________

x3: 0x_______________________________

Memory values : . = 0x1000

.word _______________________________

.word _______________________________

.word _______________________________
(D) (4 points)

. = 0x1000

start:
  li x2, 4
  jal x1, foo
  addi x2, x2, 1
  j end

bar:
  srl x2, x2, 1
  jalr x3, 0(x1)

foo:
  addi x1, x1, 4
  jal x0, bar

end:

x0: 0x_______________________________
x1: 0x_______________________________
x2: 0x_______________________________
x3: 0x_______________________________
Problem 3. RISC-V Calling Convention (16 points)

Nick, Joe, and Kevin have time-traveled to the year 3000. However, they had only written code that allows their time machine to travel forward in time! In order to get back to the 21st century, they must update their programs using RISC-V assembly language.

Nick writes the following RISC-V assembly procedure, negate_array_elements, that loops through an array and negates each element. Conveniently, they had already written a procedure, negate, that returns -A given an input A, so the procedure makes use of it.

However, Nick’s program doesn’t work as expected. Joe suggests that it may not be following the calling convention, as Nick tends to forget about that when programming in assembly.

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 right to make negate_array_elements follow the calling convention. If the procedure already follows the calling convention, write NO INSTRUCTIONS NEEDED. For full credit, you should only save registers that must be saved onto the stack and avoid unnecessary loads and stores. You do not need to use all the blank lines.

We can assume that the program will work correctly if it follows the calling convention. Do not remove or modify any of the original instructions. We can also assume that negate works as expected and follows the calling convention.

<table>
<thead>
<tr>
<th>Nick’s Code</th>
<th>Answer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>// Behavior: loops through array and negates each element.</td>
<td>negate_array_elements:</td>
</tr>
<tr>
<td>// Inputs:</td>
<td>____________________________</td>
</tr>
<tr>
<td>// a0: base address</td>
<td>____________________________</td>
</tr>
<tr>
<td>// a1: size of array (n).</td>
<td>____________________________</td>
</tr>
<tr>
<td>negate_array_elements:</td>
<td>____________________________</td>
</tr>
<tr>
<td>mv s0, a0</td>
<td>____________________________</td>
</tr>
<tr>
<td>li a2, 0</td>
<td>____________________________</td>
</tr>
<tr>
<td>loop:</td>
<td>____________________________</td>
</tr>
<tr>
<td>lw a4, 0(s0)</td>
<td>____________________________</td>
</tr>
<tr>
<td>mv a0, a4</td>
<td>____________________________</td>
</tr>
<tr>
<td>call negate</td>
<td>____________________________</td>
</tr>
<tr>
<td>sw a0, 0(s0)</td>
<td>____________________________</td>
</tr>
<tr>
<td>addi s0, s0, 4</td>
<td>____________________________</td>
</tr>
<tr>
<td>addi a2, a2, 1</td>
<td>____________________________</td>
</tr>
<tr>
<td>blt a2, a1, loop</td>
<td>____________________________</td>
</tr>
<tr>
<td>endloop:</td>
<td>____________________________</td>
</tr>
<tr>
<td>ret</td>
<td>____________________________</td>
</tr>
</tbody>
</table>
call negate

sw a0, 0(s0)

addi s0, s0, 4
addi a2, a2, 1
blt a2, a1, loop

endloop:

ret
While Nick and Joe were dealing with that, Kevin decided to check that one of the procedures he had previously written actually followed the calling convention.

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 right to make the procedure follow the calling convention. If the procedure already follows the calling convention, write **NO INSTRUCTIONS NEEDED**. For full credit, you should only save registers that **must be saved onto the stack** and avoid unnecessary loads and stores. You do not need to use all the blank lines.

*We can assume that the program will work correctly if it follows the calling convention. Do not remove or modify any of the original instructions.*

<table>
<thead>
<tr>
<th>Kevin’s Code</th>
<th>Answer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>// Behavior:</td>
<td>reached_destination_year:</td>
</tr>
<tr>
<td>// returns 1 if current =</td>
<td></td>
</tr>
<tr>
<td>destination else 0</td>
<td></td>
</tr>
<tr>
<td>// Inputs:</td>
<td></td>
</tr>
<tr>
<td>// a0: current year</td>
<td></td>
</tr>
<tr>
<td>// a1: destination year</td>
<td></td>
</tr>
<tr>
<td>reached_destination_year:</td>
<td></td>
</tr>
<tr>
<td>li t0, 0</td>
<td>li t0, 0</td>
</tr>
<tr>
<td>bne a0, a1, end</td>
<td>________________</td>
</tr>
<tr>
<td>li t0, 1</td>
<td>bne a0, a1, end</td>
</tr>
<tr>
<td>end:</td>
<td></td>
</tr>
<tr>
<td>mv a0, t0</td>
<td>li t0, 1</td>
</tr>
<tr>
<td>ret</td>
<td>end:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| | ret |
Problem 4. Stack Detective (18 points)

Below is the Python code implementing a recursive function \( f \) which takes two arguments \( a \) and \( b \). To the right is an implementation of the function using RISC-V assembly.

```python
def f(a, b):
    if b == 0:
        return a
    elif a == 0:
        return b
    else:
        return f(a - 1, b) + f(a, b - 1)
```

(A) (2 points) What should be in the blank on the line labeled L1 to make the assembly implementation match the Python code?

```
L1: addi ________________
```

(B) (1 points) How many words will be written to the stack if the function makes recursive calls to the function \( f \)?

**Number of words pushed onto stack before recursive call:** ______

f:

```riscv
bnez a1, elif
ret
elif:
bnez a0, else
mv a0, a1
ret
else:
addi sp, sp, -16
sw ra, 0(sp)
sw s0, 4(sp)
sw s1, 8(sp)
sw s2, 12(sp)
mv s0, a0
mv s1, a1
```

```
L1:
addi _____________
mv a1, s1
call f
mv s2, a0
mv a0, s0
addi a1, s1, -1
call f
add a0, s2, a0
lw ra, 0(sp)
lw s0, 4(sp)
lw s1, 8(sp)
lw s2, 12(sp)
```

```
L2:
addi sp, sp, 16
ret
```
The program’s initial call to the function \( f \) occurs outside of the function definition via the instruction `call f`. The program is interrupted during a recursive call to \( f \), just prior to the execution of `addi sp, sp, 16` at label `L2`. 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 0xEB4 and points to the location shown in the diagram.

(C) (4 points) What were the values of arguments \( a \) and \( b \) at the beginning of the initial call to \( f \)? Write CAN’T TELL if you cannot tell from the stack provided.

Arguments at beginning of initial call:

\[
\begin{align*}
\text{a} &= 0x\_\_\_\_\_\_\\
\text{b} &= 0x\_\_\_\_\_\_
\end{align*}
\]

(D) (4 points) What are the values in the following registers right when the execution of \( f \) is interrupted? Write CAN’T TELL if you cannot tell from the stack provided.

Current value of:

\[
\begin{align*}
ra &= 0x\_\_\_\_\_\_\\
\text{s0} &= 0x\_\_\_\_\_\_\\
\text{s1} &= 0x\_\_\_\_\_\_\\
\text{s2} &= 0x\_\_\_\_\_\_
\end{align*}
\]

(E) (2 points) What is the value in register \( a0 \) right when the execution of \( f \) is interrupted? Write CAN’T TELL if you cannot tell from the stack provided. \( \text{Note: Answering this question may take you longer than others, so you may want to leave it for the end of the exam and come back to it.} \)

\[
\begin{align*}
a0 &= 0x\_\_\_\_\_\_
\end{align*}
\]

(F) (2 points) What is the hex address of the `call f` instruction that made the initial call to \( f \)? Write CAN’T TELL if you cannot tell from the stack provided.

Address of instruction that made initial call to \( f \): 0x\_\_\_\_\_\_

(G) (3 points) What are the hex addresses of the `call f` instructions for the recursive calls to \( f \)? Write CAN’T TELL if you cannot tell from the stack provided.

Address of first recursive `call f` instruction: 0x\_\_\_\_\_\_

Address of second recursive `call f` instruction: 0x\_\_\_\_\_\_
Problem 5. Static Discipline (14 points)

Ben Bitdiddle has been given a hypothetical Device F and told to come up with signaling specifications for it. Consider the voltage transfer characteristic (VTC) of Device F shown below (not to scale). Ben 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. $V_{IH}$ is known to be 4.6V.

(A) (4 points) For each choice of $V_{OH}$, circle **YES** if it both follows the static discipline and allows for 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**. Explain each of your answers.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{OH,A}$</td>
<td>5.2</td>
</tr>
<tr>
<td>$V_{OH,B}$</td>
<td>4.2</td>
</tr>
<tr>
<td>$V_{OL,C}$</td>
<td>1.2</td>
</tr>
<tr>
<td>$V_{OL,D}$</td>
<td>0.8</td>
</tr>
<tr>
<td>$V_{IH}$</td>
<td>4.6</td>
</tr>
</tbody>
</table>

Explaination:

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

Explaination:

$V_{OH,B}$ (circle one): **YES** **NO** **DON’T KNOW**
(B) (4 points) $V_{IL}$ is not known right now but is guaranteed to be between 1.5 V and 2.5 V. In addition, all valid low inputs are guaranteed to produce valid high outputs. For each choice of $V_{OL}$, circle **YES** if it both follows the static discipline and has 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**. Explain each of your answers.

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

**Explanation:**

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

**Explanation:**

(C) (3 points) Suppose Ben settles on the following signaling specification which follows the static discipline:

\[
\begin{align*}
V_{IL} &= 1.5 \text{ V} \\
V_{IH} &= 4.7 \text{ V} \\
V_{OL} &= 1.1 \text{ V} \\
V_{OH} &= 4.9 \text{ V}
\end{align*}
\]

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

High Noise Margin: \(\underline{\text{____________________}}\) V

Low Noise Margin: \(\underline{\text{____________________}}\) V

Overall Noise Immunity: \(\underline{\text{____________________}}\) 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 the signaling specifications described in part (C)? If she can, circle NO CHANGES and explain why no changes are needed. 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. 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

If No Changes:
Provide explanation: __________________________________________________________

If Changes Needed:

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

Value to change to: __________________ V
Problem 6. Boolean Algebra and Combinational Logic (11 points)

(A) (2 points) Consider the logic diagram below, which takes 4 inputs \{a,b,c,d\} and computes two outputs \{x,y\}. Using the t_{PD} information for the gate components shown in the table below, compute the t_{PD} for the circuit.

\[
\begin{array}{|c|c|}
\hline
\text{Gate} & \text{t}_{PD} \\
\hline
\text{XNOR2} & 5.5\text{ns} \\
\text{AND2} & 3.5\text{ns} \\
\text{OR2} & 2.5\text{ns} \\
\text{INV} & 1.0\text{ns} \\
\hline
\end{array}
\]

\[t_{PD} (\text{ns}) = \]
(B) (3 points) Find the normal form expression and a minimal sum-of-products expression for output $F$ of the circuit described by the truth table shown below.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Normal form for $F =$ __________________________

Minimal sum of products for $F =$ __________________________
(C) (6 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.) Make sure that your final answer is in **minimal sum-of-products** form.

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

2. \( \overline{x}y\overline{z} + x \)

3. \( xy + y\overline{z}(x + \overline{z}) \)
Problem 7: CMOS Logic (16 points)

For each of the circuits shown below, please fill out the truth table that the circuit implements. Then provide a logical function for Out in terms of its inputs (a, b, c) or (a, b). Assume that you are given all inputs \(a, \overline{a}, b, \overline{b}, c,\) and \(\overline{c}\).

(A) (4 points)

\[
\begin{array}{ccc|c}
 a & b & c & \text{Out} \\
 0 & 0 & 0 & \\
 0 & 0 & 1 & \\
 0 & 1 & 0 & \\
 0 & 1 & 1 & \\
 1 & 0 & 0 & \\
 1 & 0 & 1 & \\
 1 & 1 & 0 & \\
 1 & 1 & 1 & \\
\end{array}
\]

Out\((a, b, c) = \boxed{}

(B) (4 points)

\[
\begin{array}{cc|c}
 a & b & \text{Out} \\
 0 & 0 & \\
 0 & 1 & \\
 1 & 0 & \\
 1 & 1 & \\
\end{array}
\]

Out\((a, b) = \boxed{}

6.004 Fall 2021 - 17 of 18 - Quiz #1
Please implement each of the following Boolean expressions as a single static CMOS gate. **You can assume that inputs and their complements are available as inputs for the gate (just as in the diagram in part B).** Please carefully draw your nFETs and pFETs and label all inputs and outputs in your transistor diagrams.

(C) (4 points) \[ F = (\overline{A} + \overline{B}) \cdot \overline{C} \]

(D) (4 points) \[ F = (A + B) \cdot \overline{D} + C \]

END OF QUIZ 1!