Please write your name and Kerberos on your solutions.

The only material you may access for this exam is this pdf, the RISC-V ISA handout (https://6004.mit.edu/web/_static/spring21/resources/references/6004_isa_reference.pdf) and the quiz 1 answer template if you would like to use it (https://6004.mit.edu/web/_static/spring21/resources/quizzes/q1_answer_template.makeup.pdf).

Each subproblem is labelled with (Label: X), (e.g., Label: 2A_1). If you are writing your solutions on blank paper, please make sure you write down the label of each subproblem together with its solution and circle each answer. So, for example, if your computed value for problem 2A_1 is 5, you would write:

2A_1: 5

NOTE: You must show your work to get credit for your answers.

Problem 1. Honor Code Agreement (1 point)

We are using the honor system during this exam, and ask that you accept the following terms of this honor system:

1. You will not share the exam with anyone
2. You will not discuss the material on the exam with anyone until after solutions are released
3. You understand that the exam is closed book and that you may only use reference material provided with the exam
4. You understand that you must show your work to get credit for your answers.

(label: 1) I will abide by the above terms (circle one):   Agree / Do Not Agree
Problem 2. Binary Arithmetic (11 points)

(A) (4 points) Express the following decimal values as hexadecimal numbers encoded in 8-bit 2’s complement. If a value cannot be expressed in 8-bit 2’s complement, write “Not Possible”.

76 = 0b0100_1100 (4+8+64)
Invert: 0b1011_0011
Add 1: -76 = 0b1011_0100 = 0xB4

(label: 2A_1) -76 in 8-bit 2’s complement hexadecimal (0x):____0xB4_______

The smallest value that can be represented in 8-bit 2’s complement notation is -2^7 = -128.

(B) (2 points) Calculate the 8-bit 2’s complement sum of the numbers 0xC7 and 0x74 using 8-bit 2’s complement arithmetic. Provide your answer in 8-bit 2’s complement notation. If the result cannot be expressed in 8-bit 2’s complement, write “Not Possible”. You must show your work using 2’s complement arithmetic.

0xC7 = 0b1100_0111
0x74 = 0b0111_0100
0b10011_1011

(label: 2B) 0xC7 + 0x74 in 8-bit 2’s complement binary notation (0b):____0011 1011____

(C) (1 point) What is the minimum number of bits required to represent the number 27 in 2’s complement notation?

27 = 16 + 8 + 2 + 1
To represent this value in 2’s complement notation, the MSB must be a 0 because 27 is a positive number. Thus 27 expressed with the smallest number of bits is = 0b011011 so the minimum number of bits is 6.

(label: 2C) Minimum number of bits to represent 27 in 2’s complement notation:____6____
(D) (4 points) Perform an arithmetic right-shift by 3 on the 12-bit 2’s complement number 0xC31. What is the result in hexadecimal, and what is the equivalent mathematical operation of this shift operation?

0xC31 >> 3 = 0b 1100_0011_0001 >> 3 = 0b 1111_1000_0110 = 0xF86

(label: 2D_1)  Result of shift in hexadecimal (0x): _____ F86 __________

(label: 2D_2)  Equivalent mathematical operation of right-shift by 3: _____ divide by 8 _____
Problem 3. RISC-V Assembly (16 points)

(A) (3 points) Fill in the blanks in the following C code given its translated RISC-V assembly code.

```c
// arr = an array of integers
// n = length of array
int fn(int arr[], int n) {
    int count = 0;
    for (int i = 0; i < n; i = i + 1) {
        if (arr[i] & 1 == 0) {
            count = count + 1;
        }
    }
    return count;
}
```

RISC-V:

```
fn:    li a2, 0
       li a3, 0
L:     slli a6, a3, 2
       add a6, a6, a0
lw a4, 0(a6)
       andi a5, a4, 1
       bne a5, zero, endL
       addi a2, a2, 1
endL:  addi a3, a3, 1
       blt a3, a1, L
       mv a0, a2
       ret
```

(label: 3A_1) if (arr[i] & 1 == 0)

...  

(label: 3A_2) return count;
(B) (1 point) What is the hexadecimal encoding of the highlighted instruction: \texttt{lw a4, \theta(a6)} in the assembly code snippet from part (A)? You may use the template below to help with the encoding. \textbf{Recall that a4 translates to x14 and a6 translates to x16.}

<table>
<thead>
<tr>
<th>[31: 20]</th>
<th>[19:15]</th>
<th>[14:12]</th>
<th>[11:7]</th>
<th>[6:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>funct3</td>
<td>rd</td>
<td>opcode</td>
</tr>
<tr>
<td>0000 0000 0000</td>
<td>10000</td>
<td>010</td>
<td>01110</td>
<td>000011</td>
</tr>
</tbody>
</table>

= 0x00082703

\textbf{(label: 3B) Instruction encoding (0x): \quad \quad 00082703 \quad \quad}

For the following code snippets, provide the value left in each register \textbf{just prior to executing the “nop” operation} (i.e., when the processor reaches the instruction at the \texttt{end} label), or specify \textbf{CAN’T TELL} if it is impossible to tell the value of a particular register. The code snippets are independent of both each other and of the first two parts of this question. Assume that the \texttt{nop} operation is an operation that does nothing.

(C) (6 points)

\begin{align*}
\text{addi } x1, x0, 0xE3 & \quad x0 = 0xFFFFFE35\\
\text{slli } x1, x1, 12 & \quad x1 = 0xFFFF5000\\
\text{li } x2, 0x1 & \quad x2 = 1\\
\text{bgt } x2, x1, L1 & \quad x2 > x1 \text{ so branch taken}\\
\text{addi } x2, x2, 1 & \\
\text{L1:} & \\
\text{addi } x2, x2, 4 & \quad x2 = 5\\
\text{end:} & \\
\text{nop} & \\
\end{align*}

\textbf{(label: 3C_1) \quad x1: (0x)\quad \quad FFE35000 \quad \quad}

\textbf{(label: 3C_2) \quad x2: (0x)\quad \quad 5 \quad \quad}

\textbf{(label: 3C_3) \quad pc: (0x)\quad \quad CAN’T TELL \quad \quad}
(D) (6 points)

```assembly
.data
    . = 0x100
    addi x3, zero, 0x308  x3 = 0x308
    lw  x4, -4(x3)       x4 = M[0x304] = 0x60046004
    slli x4, x4, 4       x4 = 0x00460040
    sw  x4, -8(x3)       M[0x300] = 0x00460040
.end:
    nop

.text
    . = 0x300
    .word 0x12345678
    .word 0x60046004
    .word 0xAAAAAAAA

(label: 3D_1) Which address in memory is written to: (0x) 300

(label: 3D_2) What value is written to memory: (0x) 00460040

(label: 3D_3) pc: (0x) 110
Problem 4. Corn, Crows, and Calling Conventions (19 points)

You are writing a function in RISC-V assembly called `sim_budget` that will be used as part of a special new RISC-V processor for farmers to budget their scarecrow purchases. The `sim_budget` function returns the profit for the season for a given scarecrow-purchasing budget. During this computation `sim_budget` calls the function `protect_harvest`, which takes in the number of scarecrows and returns the amount of corn left over after a crop season.

(A) (6 points) An implementation of the desired behavior of `sim_budget` is shown below in C syntax. A `uint32` is an unsigned 32-bit integer. A `const` defines a constant value. Assume `budget` is an integer multiple of `SCARECROW_COST`.

```c
const CORN_PRICE = 2;
const SCARECROW_COST = 16;
uint32 sim_budget(uint32 budget) {
    uint32 scarecrows = budget / SCARECROW_COST;
    uint32 corn_yield = protect_harvest(scarecrows);
    return corn_yield * CORN_PRICE - budget;
}
```

Implement the `sim_budget` function in RISC-V assembly such that its behavior matches that of the C implementation, while adhering to the calling convention. Assume that `protect_harvest` also follows the calling convention. Assume that the processor is a 32-bit RISC-V so all the registers are 32-bits in length, and all the values involved fit within 32 bits (so do NOT handle over/under-flow). We have intentionally provided the entire next page for your `sim_budget` implementation to give everyone plenty of space to write their function.
(label: 4A)

```
sim_budget:
    addi sp, sp, -8
    sw ra, 0(sp)
    sw a0, 4(sp)
    srli a0, a0, 4    // compute scarecrows
    call protect_harvest
    slli a0, a0, 1    // compute corn_yield * CORN_PRICE
    lw t1, 4(sp)
    sub a0, a0, t1
    lw ra, 0(sp)
    addi sp, sp, 8
    ret
```
Below is one possible implementation of `protect_harvest` shown in C syntax. The function `crows_attack` returns the amount of corn left over after `num_crows` crows attack the farmland for one day, and it follows the calling convention. Assume `INITIAL_CROWS` is always strictly greater than `scarecrows`.

```c
const SEASON_LENGTH = 50;
const INITIAL_CROWS = 300;
const INITIAL_CROP = 200000;

uint32 protect_harvest(uint32 scarecrows){
    uint32 remains = INITIAL_CROP;
    uint32 num_crows = INITIAL_CROWS - scarecrows;
    for (uint32 day = SEASON_LENGTH - 1; day >= 0; day = day - 1 {  
        remains = crows_attack(remains, num_crows);
    }
    return remains;
}
```

Here is a RISC-V assembly implementation that your friend wrote for `protect_harvest`. The code below ("Imp #1") follows the calling convention. However, it has two issues with it. 1) It is restoring one register which makes the assembly computation not match the C implementation. 2) Your friend mentioned that it is slow and is throttling the new farming processor.

```assembly
protect_harvest:  
    addi sp, sp, -16  
    sw ra, 0(sp)  
    sw s0, 4(sp)  
    sw a0, 8(sp)  
    sw t0, 12(sp)  
    li s0, 300  
    sub s0, s0, a0  
    li a0, 200000  
    li t0, 49  
    for:
        mv a1, s0  
        addi sp, sp, -16  
        sw ra, 0(sp)  
        sw s0, 4(sp)  
        sw a0, 8(sp)  
        sw t0, 12(sp)  
        call crows_attack  
        lw ra, 0(sp)  
        lw s0, 4(sp)  
        lw a0, 8(sp)  
        lw t0, 12(sp)  
        addi sp, sp, 16  
        addi t0, t0, -1  
        bgez t0, for
```
(B) (3 points) Your friend asks for your help in solving both issues. Which memory access(es) should be removed in "Imp #1" to make the code correctly compute the return value as specified by the C code. **Do not address the performance issues here.** Use the tags (1A, 1B, etc.) in the comments to the right of each line to identify the line. Also, explain why removing the line(s) will result in correcting the assembly implementation.

**(label: 4B)**

Lines 3C and 4C should be removed in order to correct the code. If a0 is restored after a function call, and before saving the return value, then the return value gets overwritten.

(C) (6 points) After correcting the code, which additional memory accesses can be removed from "Imp #1" and still have it consistently compute the correct answer? Include all memory accesses that can safely be removed and explain why it is safe to remove it. Use the tags (1A, 1B, etc.) in the comments to the right of each line to identify each line.

**(label: 4C)**

2A, 3A ⇒ ra only needs to be restored right before returning
2B, 3B ⇒ s0 is a callee-saved register, so it won't be modified by crows_attack
1C, 2C ⇒ a0 is caller-saved, but it's supposed to be modified by crows_attack and protect_harvest
1D, 4D ⇒ t0 is a caller-saved register, so protect_harvest is not responsible for saving it. However, because the value of “day” needs to be maintained across calls to crows_attack, 2D and 3D are required.
Your friend takes some of your advice for optimizing his code, but got stuck and needs your help again. The code below ("**Imp #2**") is faster than the previous implementation, but it doesn't return the correct value every time.

```assembly
protect_harvest:
    addi sp, sp, -16
    sw ra, 0(sp)
    li t0, 300
    sub t0, t0, a0
    li t1, 200000
    li t2, 49

for:
    mv a0, t1
    mv a1, t0
    call crows_attack
    mv t1, a0
    addi t2, t2, -1
    bgez t2, for
forend:

    lw ra, 0(sp)
    addi sp, sp, 16
    ret
```

(D) (4 points) What is causing "**Imp #2**" to sometimes be wrong? Name one circumstance in which "**Imp #2**" would return the correct answer.

(label: 4D)

Imp #2 is sometimes wrong because it's not saving the t-registers before or restoring the t-registers after the call to crows_attack in situations when it wants the value to be saved. (In this case, that's t0 and t2).

One circumstance in which Imp #2 would return the correct answer is if crows_attack didn't modify t0 or t2.
Problem 5. Stack Detective (16 points)

A mad scientist created a special rabbit (which he named “RAD-BIT”) that can reproduce without a mate and live forever! Every RAD-BIT gives birth to a new RAD-BIT every day.

The software engineer that works for him wrote the `radbit` function in C below to calculate how many RAD-BITs exist after x days from creation of the first RAD-BIT.

```c
/* recursively count the number radbit after x days */
int radbit(unsigned x) {
    if (x == 1) {
        return 2;
    } else {
        return 2 * radbit(x - 1);
    }
}
```

Two RAD-BITs are created. One RAD-BIT (and its offspring) were able to reproduce for n*2 days, and the other RAD-BIT (and its offspring) were able to reproduce for n*4 days.

The following function calculates the total number of RAD-BITs.

```c
int math(unsigned n) {
    unsigned math_x = radbit(n * 2);
    unsigned math_y = radbit(n * 4);
    return math_x + math_y;
}
```

```c
math: addi sp, sp -12
sw ra, 0(sp)
sw a0, 4(sp)
sw s0, 8(sp)
_____________________
first: call radbit
mv s0, a0
lw a0, 4(sp)
slli a0, a0, 2

second: call radbit
add a0, a0, s0
lw ra, 0(sp)
lw s0, 8(sp)
addi sp, sp, 12
jr ra
```
The engineer is paid by the hour, so they wrote the programs in assembly as well to kill time (shown above on the right).

(A) (2 points) Fill in the missing line in the assembly code for the math function to match the C implementation.

(label: 5A) Missing assembly line: ___ slli a0, a0, 1 _____

The mad scientist calls the math function. The program is halted during the second (i.e., after the call radbit at label second) call to radbit, right before executing the line labeled stop.

The stack pointer was located at SP_start when the math assembly program first started running. At the time the program halted, it was located at SP_stop

The radbit and the math functions share the same stack.

<table>
<thead>
<tr>
<th>SP_stop → s0</th>
<th>0x10</th>
</tr>
</thead>
<tbody>
<tr>
<td>radbit ra</td>
<td>0xF0</td>
</tr>
<tr>
<td>radbit 0</td>
<td>0x10</td>
</tr>
<tr>
<td>radbit ra</td>
<td>0xF0</td>
</tr>
<tr>
<td>radbit s0</td>
<td>0x10</td>
</tr>
<tr>
<td>radbit ra</td>
<td>0x40</td>
</tr>
<tr>
<td>math ra</td>
<td>0x08</td>
</tr>
<tr>
<td>math a0</td>
<td>0x02</td>
</tr>
<tr>
<td>math s0</td>
<td>0x01</td>
</tr>
<tr>
<td>SP_start →</td>
<td>0x14</td>
</tr>
<tr>
<td>Larger addresses</td>
<td>0x10</td>
</tr>
</tbody>
</table>

(B) (3 points) Fill out the three missing spaces in the stack.

(C) (1 point) What is the value of unsigned n passed into the original math function call?

(label: 5C) Value of n passed into original math function call: ______ 2 _____________
(D) (2 points) What is the value inside unsigned math_x after the first call to radbit returns?

(label: 5D)  Value of math_x after the first call to radbit (0x): ______ 10 ____________

(E) (2 points) What is the address of the original math function call?

(label: 5E)  Address of original math function call (0x): _____4 ____________

(F) (2 points) What is the address of the label first in the math function?

(label: 5F)  Address of label first in math function (0x): ____2C ____________

(G) (2 points) What is the address of the label stop in the radbit function?

(label: 5G)  Address of label stop in radbit function (0x): _____FC ____________

(H) (2 points) The mad scientist noticed that one of the registers the engineer saved into the stack doesn’t need to be saved. Which register is this and why?

(label: 5H_1)  Unnecessarily saved register : _____s0 ____________

(label: 5H_2)  Why doesn’t this register need to be saved on the stack?  it’s not used in radbit
Problem 6. Static Discipline (14 points)

Many things besides transistors can be used to create logic gates. Ben Bitbio, is investigating the use of biological neurons as computing devices. He characterizes some cells and comes up with the following simple model for a neuron, N. N takes three input signals, modeled as voltages $V_A$, $V_B$, and $V_C$. It produces one output voltage, $V_O$ based on the sum of the inputs.

When the sum of the inputs to the neuron is greater than or equal to 270 mV, the neuron produces a 100 mV output. When the sum of the inputs is less than or equal to 250 mV, the neuron produces 0 mV on its output. Otherwise, the output is an unknown value between 0 and 100 mV. Note that all voltages in this system will be between 0 and 100 mV inclusive. The behavior is summarized below.

$$V_O = \begin{cases} 
100 \text{ mV}, & V_A + V_B + V_C \geq 270 \text{ mV} \\
0 \text{ mV}, & V_A + V_B + V_C \leq 250 \text{ mV} \\
0 \leq ? \leq 100 \text{ mV}, & \text{otherwise} 
\end{cases}$$

Ben realizes that N can implement a 3-input AND gate. To do so, he needs to determine an appropriate signaling specification. For this entire problem, you should assume that low voltages represent logic level low values and high voltages represent logic level high values.

(A) (6 points) Create a signaling specification that allows for this gate to behave as a 3-input AND gate, follows the static discipline, and maximizes the noise margins. Remember that no single voltage signal will exceed 100 mV.

Since the outputs of N are constant, we can immediately determine $V_{OH} = 100 \text{ mV}$ and $V_{OL} = 0 \text{ mV}$. $V_{OH}$ does not need to be any lower since N can produce a 100 mV output, and lowering $V_{OH}$ needlessly reduces the high noise margin. If it is any higher, N could not be used with this signaling specification since it is incapable of producing over 100 mV. Similar reasoning applies to the choice of $V_{OL}$.

To determine input thresholds, we rely on the static discipline: all valid inputs must produce valid outputs. In this case, no valid inputs can cause the sum of the inputs to be between 250 and 270 mV. First, we choose the smallest possible value of $V_{IH}$. For three high inputs, the sum must be at least 270 mV. Therefore, $3 \cdot V_{IH} \geq 270 \text{ mV}$, so $V_{IH} \geq 90 \text{ mV}$. To maximize the high noise margin, we choose $V_{IH} = 90 \text{ mV}$.

Lastly, we determine the highest possible value of $V_{IL}$. Consider the worst-case scenario that might violate the static discipline. When two of the inputs are high and one is low, the sum
must still be at or below 250 mV. If two of the inputs are as high as possible (100 mV, as
given in the problem), the sum is 200 mV. To stay below the 250 mV threshold, the last input
must be at or below 50 mV. So in general, $V_{IL} \leq 50 \text{ mV}$. We choose $V_{IL} = 50 \text{ mV}$ to
maximize the low noise margin.

\begin{align*}
\text{(label: 6A\textunderscore1)} & \quad V_{IL} = ____ 50 _____ \text{ mV} \\
\text{(label: 6A\textunderscore2)} & \quad V_{IH} = ____ 90 _____ \text{ mV} \\
\text{(label: 6A\textunderscore3)} & \quad V_{OL} = ____ 0 _____ \text{ mV} \\
\text{(label: 6A\textunderscore4)} & \quad V_{OH} = ____ 100 _____ \text{ mV}
\end{align*}
(B) (3 points) Ignoring the results from Part A, Ben decides to use the following sub-optimal signaling specification that obeys the static discipline for Neuron N.

\[
\begin{align*}
V_{IL} &= 40 \text{ mV} \\
V_{IH} &= 90 \text{ mV} \\
V_{OL} &= 5 \text{ mV} \\
V_{OH} &= 95 \text{ mV}
\end{align*}
\]

What are the noise margins and noise immunity of this signaling specification? Noise immunity is defined as the minimum of the high and low noise margins.

\[
\begin{align*}
\text{Low Noise Margin} &= V_{IL} - V_{OL} = 40 - 5 = 35 \text{ mV} \\
\text{High Noise Margin} &= V_{OH} - V_{IH} = 95 - 90 = 5 \text{ mV} \\
\text{Noise Immunity} &= \min\{\text{Low Noise Margin, High Noise Margin}\} = 10 \text{ mV}
\end{align*}
\]

- Low Noise Margin: ________ 35 ________ mV
- High Noise Margin: ________ 5 ________ mV
- Noise Immunity: ________ 5 ________ mV

(C) (1 point) Ben wants to connect Neuron N to a Device M. Device M has the following VTC.

Assuming a valid signaling specification, what kind of logic gate is implemented by Device M?

- (label 6C) LOGIC GATE (circle one): BUFFER INVERTER NEITHER
(D) (3 points) Can Ben use Device M with Neuron N and the given signaling specification in Part B (reproduced below)? If he 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 M can be used with Neuron N while obeying the static discipline and maximizing noise margins when possible.

\[ V_{IL} = 40 \text{ mV} \]
\[ V_{IH} = 90 \text{ mV} \]
\[ V_{OL} = 5 \text{ mV} \]
\[ V_{OH} = 95 \text{ mV} \]

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: ______ 93 ______ mV

(E) (1 point) Re-calculate the noise immunity of the resulting system, regardless of whether or not you changed any thresholds.

Low Noise Margin = \( V_{IL} - V_{OL} \) = 40 \(- 5\) = 35 mV
High Noise Margin = \( V_{OH} - V_{IH} \) = 93 \(- 90\) = 3 mV

Noise Immunity = \( \min \{ \text{Low Noise Margin}, \text{High Noise Margin} \} \) = 3 mV

Noise Immunity: ______ 3 ______ mV
Problem 7. Boolean Algebra (11 points)

(A) (5 points) Simplify the following equation into its minimum sum of products form.

\[
ec(ab + \overline{a} + d + \overline{b} + c + \overline{d}bc + \overline{b})
\]

\[
= ec(ab + \overline{a} + b) + \overline{d}bc + \overline{b})
\]

\[
= ec((\overline{a} + \overline{b} + \overline{c}) (a) + \overline{d}b + \overline{b})
\]

\[
= ec(\overline{b} + \overline{c}a + \overline{db})
\]

\[
= ec(\overline{b} + \overline{db})
\]

\[
= ec(\overline{b} + \overline{d})
\]

\[
= ec\overline{b} + ec\overline{d}
\]

(label: 7A) Minimal sum of products expression: \[ ec\overline{b} + ec\overline{d} \]

(B) (3 points) If a gate G implements the function \( G(a, b, c, d) = ab\overline{c} + \overline{b}d \). Is that gate universal? Explain why or why not.

(label: 7B)

\[ G(a, b, 0, d) = ab + \overline{b}d = ba + \overline{b}d = \text{mux with inputs } a \text{ and } d \text{ and select } b. \]

Since a mux is a universal gate then so is G.
(C) (3 points) Show how to build a 2-input OR gate out of 2-input NAND gates.

(label: 7C)

\[ a + b = \overline{\overline{a} + \overline{b}} = \overline{\overline{a} \overline{b}} = \text{NAND}(\overline{a}, \overline{b}) = \text{NAND}(\text{NAND}(A, A), \text{NAND}(B, B)) \]
Problem 8. CMOS (12 points)

(A) (2 points) One day at work, Bob comes running up to your desk with two random truth tables, Table 1 and Table 2 (below). He exclaims, “One of these circuits will change the world! Unfortunately, I don’t remember which.” He states that you can implement one of these circuits using a single CMOS gate, but does not remember which of the two. Being a CMOS expert, you offer to help look at these tables to identify which one can be built using a single CMOS gate. Please identify which of these two functions, \( Q(A, B, C) \) and \( R(A, B, C) \) you can build using a single CMOS gate. Also, please explain why the other function cannot, and point to specific rows of the table below in your explanation.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

(label: 8A_1) Which function can you build with a single CMOS gate (select one)?

\( Q(A, B, C) \)

\( R(A, B, C) \)

(label: 8A_2) Why are you unable to build the other circuit with a single CMOS gate?

You cannot build \( R(A, B, C) \) because you have a rising input leading to a rising output (row 1 \( \rightarrow \) row 3)
(B) (4 Points) After evaluating the two functions, Bob rushes out in a seeming stroke of brilliance and returns with yet another function, \( S(A, B, C) \) shown in Table 3 (below). Bob is certain you can implement \( S \) with a single CMOS gate, but asks you to do the actual design instead. In the space below, please draw a complete CMOS gate using a minimal number of transistors for \( S(A, B, C) \). You see that all he did was invert all of the outputs. Unsure of his CMOS skills, Bob asks you to try to implement this new function \( S \). In the space below, please draw a CMOS implementation of \( S(A, B, C) \). For full credit minimize the total number of transistors used in your implementation.

Table 3:

<table>
<thead>
<tr>
<th>Row</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(label: 8B) CMOS implementation of \( S(A,B,C) \):

The SOP is \( \overline{A} \overline{B} + \overline{B} \overline{C} \), which can be rewritten as \( \overline{B}(\overline{A} + \overline{C}) \).
(C) (6 points) With your CMOS implementation, Bob proposes making a new module, \( S \) using your CMOS gate. He asserts that this new module is universal! Not wanting to make any mistakes, you set out to prove this. Using your implementation for \( S \), please show how you would build a two-input NAND gate, a two-input OR gate, and a two-input AND gate. Please write your answers using function composition (e.g. \( S(X, Y, S(Y, 0, 0)) \)).

\[
\text{(label: 8C_1) NAND}(X, Y) = \quad \quad S(X, 0, Y) \quad \quad \quad \quad
\]

\[
\text{(label: 8C_2) OR}(X, Y) = \quad \quad S(S(X, Y, 1), 0, 1) \quad \quad \quad \quad
\]

NOT: \( S(X, 0, 1) \) or \( S(1, 0, X) \) or \( S(1, X, 0) \) or \( S(0, X, 1) \) or \( S(0, X, 0) \)
NOR: \( S(X, Y, 1) \) or \( S(1, X, Y) \)
OR: NOT(NOR(X, Y))

\[
\text{(label: 8C_3) AND}(X, Y) = \quad \quad S(S(X, Y, 0), 1) \quad \quad \quad \quad
\]

NOT(NAND(X, Y))

END OF QUIZ 1!