Problem 1. Binary Arithmetic (14 points)

(A) (2 points) Write 7 and 4 in 4-bit 2’s complement notation, then add them together using fixed width 2’s complement arithmetic. Show your work. Provide your result in binary, and decimal. For each computation also specify whether or not overflow occurred.

\[
1
+ 0100
\]

Sum in binary: \(0b\) \underline{1011} \\
Sum in decimal: \(-5\) \\
Did overflow occur? (Yes/No): Yes

(B) (2 points) Write -3 and -4 in 4-bit 2’s complement notation, then add them together using fixed width 2’s complement arithmetic. Show your work. Provide your result in binary, and decimal. For each computation also specify whether or not overflow occurred.

\[
3 = 0b0011, \hspace{1cm} -3 = 0b1100 + 1 = 0b1101 \\
4 = 0b0100, \hspace{1cm} -4 = 0b1011 + 1 = 0b1100
\]

\[
11\hspace{1cm}1101
+ \hspace{1cm}1100
\]

\[
\hspace{1cm}1\hspace{1cm}0101
\]

Sum in binary: \(0b\) \underline{1011} \\
Sum in decimal: \(-5\) \\
Did overflow occur? (Yes/No): Yes
Sum in binary: 0b______1001______

Sum in decimal: _______−7________

Did overflow occur? (Yes/No): _______No______

(C) (4 points) Fill in the following table with corresponding representations in binary, decimal, and hexadecimal.

<table>
<thead>
<tr>
<th>Encoding type</th>
<th>Binary</th>
<th>Decimal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit unsigned</td>
<td>0b0010_1001</td>
<td>41</td>
<td>0x29</td>
</tr>
<tr>
<td>8-bit 2’s complement</td>
<td>0'b1111_1001</td>
<td>−7</td>
<td>0xF9</td>
</tr>
</tbody>
</table>

(D) (6 points) Using 8-bit 2’s complement encoding, compute \((\neg(0x\text{AB} \& 0x55)) \div 0x04) \times 0x04\), where \(\neg\) represents bitwise negation, \(\&\) represents a bitwise AND, \(\div\) represents integer division (ignores remainder), and \(\times\) represents multiplication.

For performing \(\div\) and \(\times\), the only operations allowed are \(\gg_a\), \(\gg_l\), \(\ll\) which correspond to arithmetic right shift, logical right shift and logical left shift respectively. Clearly specify which operation to use for performing \(\div\) and \(\times\). Write the intermediate and final answers in 8-bit 2’s complement.

\[
\begin{align*}
0x\text{AB} &= 0'b1010_1011 \\
0x55 &= 0'b0101_0101 \\
0x\text{AB} \& 0x55 &= 0'b0000_0001 \\
\neg(0x\text{AB} \& 0x55) &= 0'b1111_1110 \\
\neg(0x\text{AB} \& 0x55) \div 0x04 &= 0'b1111_1111 \\
\neg(0x\text{AB} \& 0x55) \div 0x04 \times 0x04 &= 0'b1111_1100
\end{align*}
\]

\(~(0x\text{AB} \& 0x55): 0b______1111_1110______\

Operation used for performing “\(\div\)” (Circle) \(\gg_a\) \(\gg_l\) \(\ll\)

Operation used for performing “\(\times\)” (Circle) \(\gg_a\) \(\gg_l\) \(\ll\)

\((\neg(0x\text{AB} \& 0x55)) \div 0x04) \times 0x04: 0b______1111_1100______\)
Problem 2. RISC-V Assembly (15 points)

(A) (8 points) Ben Bitdiddle has a list of ticket prices for an upcoming concert that he’s trying to sort in ascending order. While implementing his sorting algorithm, he accidentally deleted a couple lines. Help Ben fix his program by filling in the missing lines. An equivalent C program is available below. Assume that the address to the array is stored in a0 and the number of elements in the array is stored in a1. You can assume that a1 is > 1.

<table>
<thead>
<tr>
<th>C Program</th>
<th>RISC V Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>// insertion sort</td>
<td>start:</td>
</tr>
<tr>
<td>void sort(int array[], int n) {</td>
<td>slli a1, a1, 2</td>
</tr>
<tr>
<td>for (int i = 1; i &lt; n; i = i+1) {</td>
<td>// addr of end of array</td>
</tr>
<tr>
<td>int x = array[i];</td>
<td>add a1, a0, a1</td>
</tr>
<tr>
<td>int j = i-1;</td>
<td>// addr of array[i]</td>
</tr>
<tr>
<td>// shift over elements less than x</td>
<td>addi a2, a0, 4</td>
</tr>
<tr>
<td>// to the right of x</td>
<td>for:</td>
</tr>
<tr>
<td>while (j &gt;= 0 &amp;&amp; x &lt; array[j]) {</td>
<td>lw a3, 0(a2)</td>
</tr>
<tr>
<td>array[j+1] = array[j];</td>
<td>// x</td>
</tr>
<tr>
<td>j = j-1;</td>
<td>addi a4, a2, -4</td>
</tr>
<tr>
<td>}</td>
<td>while:</td>
</tr>
<tr>
<td>// before ending the loop, insert x</td>
<td>blt a4, a0, endwhile</td>
</tr>
<tr>
<td>array[j+1] = x;</td>
<td>lw a5, 0(a4)</td>
</tr>
<tr>
<td>}</td>
<td>// array[j]</td>
</tr>
<tr>
<td></td>
<td>bge a3, a5, endwhile</td>
</tr>
<tr>
<td></td>
<td>sw a5, 4(a4)</td>
</tr>
<tr>
<td></td>
<td>addi a4, a4, -4</td>
</tr>
<tr>
<td></td>
<td>j while</td>
</tr>
<tr>
<td></td>
<td>endwhile:</td>
</tr>
<tr>
<td></td>
<td>sw a3, 4(a4)</td>
</tr>
<tr>
<td></td>
<td>addi a2, a2, 4</td>
</tr>
<tr>
<td></td>
<td>blt a2, a1, for</td>
</tr>
<tr>
<td></td>
<td>end:</td>
</tr>
<tr>
<td></td>
<td>ret</td>
</tr>
</tbody>
</table>
(B) (3 points) Ben Bitdiddle’s friend Carol Compiler needs help with her code! Help Ben debug Carol’s code by providing the values in the registers when the program reaches the `ret` instruction. Assume all registers are initialized to zero. All code snippets are independent of each other.

<table>
<thead>
<tr>
<th>Program</th>
<th>Register values when the program reaches the <code>ret</code> instruction (in hexadecimal).</th>
</tr>
</thead>
<tbody>
<tr>
<td>start:</td>
<td></td>
</tr>
<tr>
<td><code>li a0, 0xA</code></td>
<td>a0 = 0xA</td>
</tr>
<tr>
<td><code>li a1, 0x7</code></td>
<td>a1 = 0x7</td>
</tr>
<tr>
<td><code>bli a0, a1, skip</code></td>
<td>a0 = 0x2</td>
</tr>
<tr>
<td><code>sra a0, a0, 2</code></td>
<td>a2 = 0xFFFFFA42</td>
</tr>
<tr>
<td><code>skip: xor a2, a0, 0xA40</code></td>
<td></td>
</tr>
<tr>
<td><code>ret</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a0: 0x_2-------------------------------------</td>
</tr>
<tr>
<td></td>
<td>a1: 0x_7-------------------------------------</td>
</tr>
<tr>
<td></td>
<td>a2: 0x_FFFFFFA42--------------------------</td>
</tr>
</tbody>
</table>

(C) (4 points) After some searching, Ben has found the source of Carol’s bug! The snippet below was intended to find the sum of an array stored at `0x740`. However, the function does not work as intended. What value is loaded into `a3` on the first three iterations of the loop?

i.

```
start:  
  li a0, 0x740       // a0 = start of array       a0 = 0x740
  li a1, 4           // a1 = number of elements in array   a1 = 4
  li a2, 0            a2 = 0
loop:       iteration 1       iteration 2       iteration 3
  lw   a3, 0(a0)    a3 = 0x2EB      a3 = 0x7F1      a3 = 0x2EB
  add  a2, a2, a3  sum of a3s      sum of a3s      sum of a3s
  addi a1, a1, -1  a1 = 3          a1 = 2          a1 = 1
  lw   a0, 4(a0)    a0 = 0x748      a0 = 0x740      a0 = 0x748  
  bnez a1, loop    
end:       
  mv   a0, a2
  ret
```

```
.a=0x740
.word 0x000002EB
.word 0x00000748
.word 0x00000740

.a3 (after 1st iteration): 0x____2EB______
.a3 (after 2nd iteration): 0x____7F1______
.a3 (after 3rd iteration): 0x____2EB______
```

ii. **Circle** the incorrect line of code above and provide a replacement instruction, below, that will fix Carol’s function.

```
Fixed instruction: _____addi a0, a0, 4_________
```
Problem 3. RISC-V Calling Conventions (12 points)

Guy and Dude are working on a project that requires them to code in RISC-V Assembly Language. Since this is their first time working in assembly, they are looking for a RISC-V expert to check whether their code is working correctly. As a friend of Guy and Dude, you volunteered to help them.

Please add or cross out appropriate instructions (either increment/decrement stack pointer, load word from stack, or save word to stack only) to make funcA and funcB follow the RISC-V calling convention. You may not change which registers are being used. 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 cross out unnecessary loads and stores.

Assume that all values are unsigned 32-bit integers.

(A) (3 points)

```
def funcA(x,y):
    return x*y

//two arguments are stored in a0,a1

funcA:
    addi sp, sp, -8
    sw t0, 0(sp)
    sw t1, 4(sp)
    mv t0, zero
next:
    andi t1, a1, 1
    srl a1, a1, 1
    beq t1, zero, skip
    add t0, t0, a0
skip:
    slli a0, a0, 1
    bne a1, zero, next
    mv a0, t0
    lw t0, 0(sp)
    lw t1, 4(sp)
    addi sp, sp, -8
    ret
```
You may assume that the `funcA` called below follows the calling conventions.

```python
def funcB(array, max_iter):
    result = 0
    for i in range(0, max_iter):
        result += funcA(array[i], array[i+1])
    return result
```

// a0 is the starting address of array,
// a1 is the maximum number of iterations
funcB:

```assembly
addi sp, sp, -20
addi sp, sp, -12
sw s0, 0(sp)
sw s1, 4(sp)
sw s2, 8(sp)
sw ra, 12(sp)
mv s0, zero
mv s1, a0
mv s2, a1
mv t0, zero
while:
    bge s0, s2, end
    slli a2, s0, 2
    add a2, s1, a2
    lw a0, 0(a2)
    lw a1, 4(a2)
    sw t0, 16(sp)
call funcA
    lw t0, 16(sp)
    add t0, t0, a0
    addi s0, s0, 1
    j while
end:
    mv a0, t0
    lw s0, 0(sp)
    lw s1, 4(sp)
    lw s2, 8(sp)
    lw ra, 12(sp)
addi sp, sp, -12
addi sp, sp, 20
ret
```
Problem 4. Stack Detective (17 points)

Below is the Python code implementing a recursive function `count8` which counts the number of 8’s (base 16) that are in the input `in`. For example, `count8(0x988) = 2` and `count8(0x81) = 1`. To the right is an implementation of the function using RISC-V assembly.

```python
def count8(in):
    if in < 8:
        return 0
    end = in & 0xF
    current_8 = 0
    if end == 8:
        current_8 = 1
    return current_8 + count8(in >> 4)
```

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

L1: `bne __a1, a2, continue` 

(B) (1 point) How many words will be written to the stack before the program makes each recursive call to the function `count8`?

Number of words pushed onto stack before recursive call: ___3____

```python
count8:
    li a1, 8
    bgt a1, a0, base
    addi sp, sp, -12
    sw ra, 0(sp)
    sw s1, 4(sp)
    andi a2, a0, 0xF
    li s1, 0
L1:
    bne _________
    addi s1, s1, 1
continue:
    srl a0, a0, 4
    sw a0, 8(sp)
    call count8
    add a0, a0, s1
L2:
    lw ra, 0(sp)
    lw s1, 4(sp)
    addi sp, sp, 12
    j done
base:
    li a0, 0
done:
    ret```
The program’s initial call to the function `count8` occurs outside of the function definition via the instruction `call count8`. The program is interrupted during a recursive call to `count8`, just prior to the execution of `lw ra, 0(sp)` 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 `0xEAC` and points to the location shown in the diagram.

(C) (3 points) Fill in the blanks for the stack to the right for `0xEA0`, `0xEA4`, and `0xEA8`.

Each stack frame stores `ra`, `s1`, `a0`. The current stack frame corresponds to a recursive call to `count8`. Therefore, `0x9E4` is the recursive `ra` value. This means that `0xC8` is the original `ra` value. `S1` is a 1 if the value shifted out in the previous recursion was an 8. `a0` is the previous value of `a0` shifted to the right by 4. `0x82` from address `0xEB4` shifted to the right by 4 gives us `0x8` for the value in `0xEA8`. The `s1` value corresponds to the value that was shifted out in the previous recursive call. Since that value was an 8, `0xEA4` is a 1.

We see that `0xE94` through `0xE9C` are from `count8(0x8)`, the 3 below are from `count8(0x82)`, then `0xEAC` to `0xEB4` are from `count8(0x828)`, and 3 below are from `count8(0x8288)`.

(D) (2 points) What is the initial input `in` at the initial call to `count8`?
Write CAN’T TELL if you cannot tell from the stack provided.

Argument at beginning of initial call: `in = 0x_____8288________`

In each iteration `s1` is set to 1 if the least significant 4 bits equal 8 and 0 otherwise. The stack frame beginning at `0xEB8` corresponds to the initial call to `count8`. The value stored at `0xEC0` is the initial input shifted to the right by 4 (0x828). The `s1` in the next stack frame (`0xEB0`) is the value of `s1` corresponding to the 4 bits that were just shifted out. Since this value is 1, it means that the least significant 4 bits equal 8. Therefore, the initial input was `0x8288`.

(E) (3 points) What are the values in the following registers right when the execution of `count8` is interrupted? Write CAN’T TELL if you cannot tell from the stack provided.

Current value of: `ra = 0x______9E4________`

`a2 = 0x______8________`

`s1 = 0x______1________`

(F) (2 points) What is the value in register `a0` right when the execution of `count8` is interrupted? Write CAN’T TELL if you cannot tell from the stack provided.
a0 = 0x________2________

(G) (2 points) What is the hex address of the call f instruction that made the initial call to count8? Write CAN’T TELL if you cannot tell from the stack provided.

Address of instruction that made initial call to f: 0x_____C4_____

(H) (2 point) What is the hex address of the continue label? Write CAN’T TELL if you cannot tell from the stack provided.

Address of continue label: 0x_____9D8______
Problem 5. The $\neg (\forall) \neg$ Abstraction (12 points)

Device X has the Voltage Transfer Characteristic (VTC) given below:

![VTC Diagram]

We want to use Device X to build an XOR gate.

(A) (9 points) Consider the circuit shown below. We want to analyze three candidate signaling specifications. We consider a signaling specification valid if and only if the circuit behaves like a digital XOR gate. Find whether each signaling specification is valid, briefly explaining why or why not. If the specification is valid, give its noise immunity (smallest noise margin).

For this analysis, assume that input voltages are always between 0V and 5V (otherwise, a very low or very high voltage at one of the inputs could make the circuit misbehave).

<table>
<thead>
<tr>
<th>Spec</th>
<th>$V_{OL}$</th>
<th>$V_{IL}$</th>
<th>$V_{IH}$</th>
<th>$V_{OH}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0V</td>
<td>0.5V</td>
<td>4.5V</td>
<td>5V</td>
</tr>
<tr>
<td>B</td>
<td>0V</td>
<td>1V</td>
<td>4V</td>
<td>5V</td>
</tr>
<tr>
<td>C</td>
<td>0V</td>
<td>2V</td>
<td>3V</td>
<td>5V</td>
</tr>
</tbody>
</table>

Spec A: Noise immunity or invalid spec? _____ 0.5V _____

Brief explanation for why valid/invalid:

Vin in range (-0.5V, 0.5V) if both inputs are the same, at least 4.5V-0.5V=4V for $A=1\ B=0$, and at most -4V for $A=0\ B=1$

Spec B: Noise immunity or invalid spec? _____ 1V _____

Brief explanation for why valid/invalid:

Vin in range (-1V, 1V) if both inputs are the same, at least 4V-1V=3V for $A=1\ B=0$, and at most -3V for $A=0\ B=1$

Spec C: Noise immunity or invalid spec? __INVALID__

Brief explanation for why valid/invalid:

With $A=1\ B=0$, Vin can be as low as 3V-2V=1V, which gives a digital 0 at the output ($\neg$ not a XOR). Moreover, valid digital inputs can produce invalid outputs (e.g., $VA=4V\ VB=1.5V$).
(B) (3 points) Consider the circuit shown below. Find the signaling specification that makes this circuit behave like an XOR gate and maximizes noise immunity. As before, assume that input voltages are always between 0V and 5V.

Signaling specification: $V_{OL} = 0\, \text{V}$ $V_{IL} = 2.2\, \text{V}$ $V_{IH} = 2.8\, \text{V}$ $V_{OH} = 5\, \text{V}$

Noise Immunity: $2.2\, \text{V}$

With $V_{in}=2.2\, \text{V}$ (3.8V) an X device produces $V_{out}=1\, \text{V}$ (4V), which exercise the maximum voltage ranges at the inputs of the circuit from part (A) that make it behave like an XOR gate.
Problem 6. Boolean Algebra (18 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. \((\bar{x}z + \bar{y}z) = (\bar{x} + \bar{y})z = xy + \bar{z}\)

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

3. \(\bar{x} + \bar{y} + xy = (\bar{xy}) + xy = 1\)

4. \(yz(\bar{y} + \bar{x}) + \bar{x}y = \bar{y}yz + \bar{x}yz + \bar{x}y = \bar{x}y(z + 1) = \bar{x}y\)
(B) (2 points) You are given the truth table for a circuit that takes a 3-bit unsigned binary input (X = ABC), multiplies it by 2 mod 8 and adds 1 mod 8 to it to produce a 3-bit unsigned binary output (Y = A'B'C').

\[
\begin{array}{ccc|ccc}
A & B & C & A' & B' & C' \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

For the above truth table, write out a **minimal sum-of-products** for each function A'(A,B,C), B'(A,B,C), and C'(A,B,C)

**Minimal sum-of-products for A'(A,B,C)=________ B ______________**

**Minimal sum-of-products for B'(A,B,C)=________ C ______________**

**Minimal sum-of-products for C'(A,B,C)=________ 1 ______________**
(C) (3 points) Now consider a new function $G(A, B, C)$ defined by the truth table below. Find the normal form and a minimal sum-of-products expression for $G(A, B, C)$.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

1. Normal form for $G = \overline{a}\overline{b}\overline{c} + \overline{a}bc + \overline{a}b\overline{c} + a\overline{b}c + \overline{a}bc + abc$

2. Minimal sum of products for $G = \overline{b} + \overline{a}\overline{c} + ac$

\[ \overline{a}\overline{b}\overline{c} + \overline{a}bc + \overline{a}b\overline{c} + a\overline{b}c + \overline{a}bc + abc = \overline{b} + \overline{a}\overline{c} + ac \]
(D) (3 points) Draw the circuit that implements the minimal sum of products you derived for $G$ using the fewest number of gates. You may use 2-input OR, NOR, AND, NAND, XOR, and XNOR, and inverters in your circuit.

![Circuit Diagram]

(E) (2 points) Below you are given the delays for the different gates you were permitted to use in part D above. Compute the propagation delay of your circuit from D.

<table>
<thead>
<tr>
<th>Gate</th>
<th>$t_{PD}$ (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XNOR2</td>
<td>7.0</td>
</tr>
<tr>
<td>XOR2</td>
<td>6.5</td>
</tr>
<tr>
<td>NOR2</td>
<td>6.0</td>
</tr>
<tr>
<td>OR2</td>
<td>5.5</td>
</tr>
<tr>
<td>AND2</td>
<td>5.0</td>
</tr>
<tr>
<td>NAND2</td>
<td>3.0</td>
</tr>
<tr>
<td>INV</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Longest path:

$\overline{a}/c \rightarrow \overline{XOR2} \rightarrow \overline{NAND2} \rightarrow G$

$6.5 + 3.0 = 9.5$ns

$t_{PD}$ (ns) = \underline{9.5}
Problem 7. CMOS Logic (12 points)

Ben invented a light-speed spaceship last night, using a CMOS gate as the critical component. He wrote down the truth table for the Boolean expression implemented by this gate. Unfortunately, his nemesis replaced one of the entries in his truth table and Ben can’t remember which one it was! Fortunately, you have taken 6.191 and can help Ben reconstruct the truth table.

Below is the truth table for Ben’s Boolean expression, $F(a, b, c)$. One entry of the truth table has been modified. Ben surmises that his nemesis has flipped one of the outputs from a 1 to a 0.

(A) (3 points) Circle the entry in the truth table which has been modified and explain why it must have been incorrect.

<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>1</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>0</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>1</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>0</td>
</tr>
</tbody>
</table>

F(0, 1, 0) -> F(1, 1, 0) should follow the “rising inputs lead to falling outputs” rule. However, the truth table has F(0, 1, 0) = 0 and F(1, 1, 0) = 1, which is not possible under this rule.

Since we know Ben’s nemesis flipped an output from a 1 to a 0, it must be that F(0, 1, 0) is the incorrect line – its output should really be a 1.

(B) (1 point) Ben thinks he can make the spaceship even faster if he sets $F(1, 1, 1) = 1$. Can Ben still implement this with a single CMOS gate?

(circle one) Yes  No

(C) (4 points) Ben wants to add an input d to his CMOS gate to implement a new function, G. Ben sets $G(a, b, c, 0) = F(a, b, c)$. Given that G can be implemented as a single CMOS gate, what are the following values?

(circle one) $G(0, 0, 1, 1) =$ : 0 ... 1 ... (can’t say)

(circle one) $G(1, 0, 1, 1) =$ : 0 ... 1 ... (can’t say)
(D) (4 points) Alice thinks Ben should use a different design. She proposes using the Boolean expression $\overline{B}(\overline{A} + \overline{C})$. Draw the CMOS gate for Alice’s Boolean expression.

END OF QUIZ 1!