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.

\[
\begin{align*}
\quad &+ \quad \quad \\
\text{Sum in binary: } &0b_______ \\
\text{Sum in decimal: } &\quad \quad \\
\text{Did overflow occur? (Yes/No): } &\quad \quad \\
\end{align*}
\]

(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.

\[
\begin{align*}
\quad &+ \quad \quad \\
\text{Sum in binary: } &0b_______ \\
\text{Sum in decimal: } &\quad \quad \\
\text{Did overflow occur? (Yes/No): } &\quad \quad \\
\end{align*}
\]
(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></td>
<td></td>
<td>0x29</td>
</tr>
<tr>
<td>8-bit 2’s complement</td>
<td>0'b1111_1001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(D) (6 points) Using 8-bit 2’s complement encoding, compute \((\neg(0xAB \& 0x55)) \div 0x04) \times 0x04\), 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 \(\gg_a\), \(\gg_1\), \(\ll_1\) 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.

\(~(0xAB \& 0x55): 0b___________________\)

Operation used for performing \(\div\) (Circle) \(\gg_a\) \(\gg_1\) \(\ll_1\)

Operation used for performing \(\times\) (Circle) \(\gg_a\) \(\gg_1\) \(\ll_1\)

\((~(0xAB \& 0x55)) \div 0x04) \times 0x04: 0b___________________\)
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.

C Program

```c
// insertion sort
void sort(int array[], int n)
{
    for (int i = 1; i < n; i = i+1)
    {
        int x = array[i];
        int j = i-1;

        // shift over elements less than x
        // to the right of x
        while (j >= 0 && x < array[j])
        {
            array[j+1] = array[j];
            j = j-1;
        }

        // before ending the loop, insert x
        array[j+1] = x;
    }
}
```

RISC V Program

```riscv
start:
slli a1, a1, 2
    // addr of end of array
add a1, a0, a1
    // addr of array[i]
addi a2, a0, 4

for:
    lw a3, 0(a2) // x
        // addr of array[j]
addi a4, a2, -4

while:

lw a5, 0(a4) // array[j]
bge a3, a5, endwhile
sw a5, 4(a4)

    j while

endwhile:

    addi a2, a2, 4

end:
ret
```
(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>
</table>
| start:  | a0: 0x__________________________  
|        | a1: 0x__________________________  
|        | a2: 0x__________________________  |
| li a0, 0xA |  
| li a1, 0x7 |  
| blt a0, a1, skip |  
| srai a0, a0, 2 |  
| skip: xor a2, a0, 0xA40 |  
| ret |  |

(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
    li a1, 4           // a1 = number of elements in array
    li a2, 0
loop:
    lw a3, 0(a0)       
    add a2, a2, a3
    addi a1, a1, -1
    lw a0, 4(a0)
    bnez a1, loop
end:
    mv a0, a2
    ret
```

a3 (after 1st iteration): 0x__________________________
a3 (after 2nd iteration): 0x__________________________
a3 (after 3rd iteration): 0x__________________________

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

Fixed instruction: ________________________________
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)

```python
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
```
(B) (9 points) 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
```

```assembly
// a0 is the starting address of array,
// a1 is the maximum number of iterations
funcB:
    addi sp, sp, -12
    sw s0, 0(sp)
    sw s1, 4(sp)
    sw s2, 8(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)
    call funcA
    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)
    addi sp, sp, 12
    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 _____________
```

(B) (1 points) 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: ________

```asm
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`.

(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________________`

(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________________`
`a2 = 0x________________`
`s1 = 0x________________`

(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________________`

(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_____________

(H) (2 points) 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_____________`
Problem 5. The $\overline{\overline{\overline{\neg (\forall)}}}$ Abstraction (12 points)

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

$$V_{out} = \begin{cases} 
5V & V_{in} \leq -3V \\
5(-V_{in} - 2V) & -3V \leq V_{in} \leq -2V \\
0V & -2V \leq V_{in} \leq 2V \\
5(V_{in} - 2V) & 2V \leq V_{in} \leq 3V \\
5V & V_{in} \geq 3V 
\end{cases}$$

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

Spec A: Noise immunity or invalid spec? _________

Brief explanation for why valid/invalid:

Spec B: Noise immunity or invalid spec? _________

Brief explanation for why valid/invalid:

Spec C: Noise immunity or invalid spec? _________

Brief explanation for why valid/invalid:
(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: $V_{OL} = \_\_\_\_ \, V$  $V_{IL} = \_\_\_\_ \, V$  $V_{IH} = \_\_\_\_ \, V$  $V_{OH} = \_\_\_\_ \, V$

Noise Immunity:  \_\_\_\_ \, V
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. \((\overline{xz} + \overline{yz})\)

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

3. \(\overline{x} + \overline{y} + xy\)

4. \(yz(\overline{y} + \overline{x}) + \overline{xy}\)
(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’).

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A’</th>
<th>B’</th>
<th>C’</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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)=________________________

Minimal sum-of-products for B’(A,B,C)=________________________

Minimal sum-of-products for C’(A,B,C)=________________________
(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>A</th>
<th>B</th>
<th>C</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 $G =$

2. Minimal sum of products for $G =$

(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.
(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>

$t_{PD}$ (ns) = _____________________________
**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.

(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.