Problem 1. Binary Arithmetic (10 points)

(A) (4 points) What is \(\lnot(0x5A) \oplus 0x3D\), where \(\lnot\) is bitwise NOT and \(\oplus\) is bitwise XOR? Provide your result in both binary and hexadecimal.

Result in binary (0b): ______________

Result in hexadecimal (0x): ______________

(B) (4 points) What is 15 in 8-bit 2’s complement notation? What is \(-22\) in 8-bit 2’s complement notation? Show how to compute 15–22 using 2’s complement addition. What is the result in 8-bit 2’s complement notation?

15 in 8-bit 2’s complement notation (0b): ______________

\(-22\) in 8-bit 2’s complement notation (0b): ______________

15–22 in 8-bit 2’s complement notation (show your work) (0b): ______________

(C) (2 points) What range of numbers encoded using two’s complement representation can be expressed using 5 bits? Provide your answer in decimal.

Smallest 5-bit two’s complement number (in decimal): ______________

Largest 5-bit two’s complement number (in decimal): ______________
Problem 2. Boolean Expressions (12 points)

(A) (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.)

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

2. $a \cdot \overline{(b + c)}(c + a)$

(B) (6 points) There are Boolean expressions for which no assignment of values to variables can produce True (e.g., $a \cdot \overline{a}$). These Boolean expressions are said to be non-satisfiable.

Are the following Boolean expressions satisfiable? If the expression is satisfiable, give an assignment to variables that makes the expression evaluate to True. If the expression is non-satisfiable, explain why.

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

2. $(\overline{x} + yz) \cdot (\overline{y}x + z) \cdot (\overline{z}y + x) + (\overline{x} + yz) \cdot (\overline{y}x + z) \cdot (\overline{z}y + x)$
Problem 4. Combinational Logic in BSV (25 points)

(A) (4 points) The following BSV function f performs a basic operation using a and b. We want f2 to implement the same function as f. Fill in the blank in f2 to make the two functions equivalent. Use a single expression. Assume n is a power of 2.

```haskell
// TLog#(n) is the log base 2 of n
function Bit#(1) f(Bit#(n) a, Bit#(TLog#(n)) b);
    Bit#(n) x = a;
    for (Integer i = 0 ; i < valueOf(TLog#(n)) ; i = i+1) begin
        // (2**i) is 2 to the ith power
        x = (b[i] == 1) ? x >> (2**i) : x;
    end
    return x[0];
endfunction

function Bit#(1) f2(Bit#(n) a, Bit#(TLog#(n)) b);
    return ______________;
endfunction
```
(B) (6 points) Write the truth table for the combinational device described by the function below.

```plaintext
function Bit#(2) f(Bit#(1) a, Bit#(1) b, Bit#(1) c);
    Bit#(8) x = 8'hB7; // hex value 0xB7
    Bit#(2) ret = 1;
    if (c == 1) begin
        x = x + 1;
    end
    case ({a, b})
        1: ret = x[1:0];
        2: ret = x[3:2];
        3: ret = x[7:6] ^ 2'b11;
    endcase
    return ret;
endfunction
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>ret[1]</th>
<th>ret[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
(C) (4 points) The following BSV function \( g \) performs a specific arithmetic operation on \( n \)-bit operands \( a \) and \( b \). We want the function \( g_2 \) to implement \( g \) in a single line of code. Fill in the blank with an expression of the form “\( a \) operation \( b \)” to make \( g_2 \) equivalent to \( g \).

```hs
function Bit#(n) g(Bit#(n) a, Bit#(n) b);
    Bit#(TAdd#(n,1)) c = 1; // note initial value
    Bit#(n) ret = 0;
    for (Integer i = 0 ; i < valueOf(n) ; i = i+1) begin
        ret[i] = a[i] ^ ~b[i] ^ c[i];
        c[i+1] = (a[i] & ~b[i]) | (a[i] & c[i]) | (~b[i] & c[i]);
    end
    return ret;
endfunction

function Bit#(n) g2(Bit#(n) a, Bit#(n) b);
    return __________;
endfunction
```

(D) (5 points) For \( n = 2 \), manually compile the BSV function \( g \) above into a combinational circuit. Please draw its logic diagram. You can use inverters, AND, OR, XOR, NAND, and NOR gates, as well as multiplexers. Please label the inputs and outputs bit by bit (\( a[0] \), \( a[1] \), \( b[0] \), \( b[1] \), \( ret[0] \), and \( ret[1] \)). You do not need to optimize or simplify the circuit.
(E) (6 points) Show that 1-bit 2-to-1 muxes can be used to implement any combinational circuit by implementing an inverter, an AND gate, and an OR gate using only 1-bit 2-to-1 muxes. You may tie inputs to 1 or 0 if necessary, and may use one or multiple muxes. Clearly label all inputs and outputs.

Logic diagram of inverter implementation using 2-input mux:

\[
Z = A \cdot S + B \cdot \bar{S}
\]

Logic diagram of AND gate implementation using 2-input mux:

Logic diagram of OR gate implementation using 2-input mux:
Problem 1. Assembly Language (15 points)

For the RISC-V instruction sequences below, provide the hex values of the specified registers after each sequence has been executed. Assume that each sequence execution ends when it reaches the `unimp` instruction. Also assume that all registers are initialized to 0 before execution of each sequence begins.

(A) (7 points)

\[
\begin{array}{l}
. = \text{0x}0 \\
\quad \text{li x11, 0x400} \\
\quad \text{lw x11, 0x0(x11)} \\
\quad \text{bge x11, x0, L1} \\
\quad \text{xori x12, x11, 0xffff} \\
\quad \text{j end} \\
\end{array}
\]

\[
\begin{array}{l}
\text{L1: srl x12, x11, 8} \\
\text{end: unimp} \\
. = \text{0x}400 \\
X: \quad \text{.word 0xC0C0A0A0} \\
\end{array}
\]

Value left in x11: 0x__________

Value left in x12: 0x__________
(B) (8 points)

```
. = 0x0
    lw x11, 0x2000(x0)
    mv x12, x0
loop:
    andi x13, x11, 1
    add x12, x12, x13
    srl x11, x11, 1
    bnez x11, loop
    unimp

. = 0x200
X: .word 0x00140083
```

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Value left in x11:</th>
<th>Value left in x12:</th>
<th>Value left in x13:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td></td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td></td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>
Problem 2. RISC-V Assembly and Calling Conventions (20 points)

For each of the code segments below, specify whether or not the RISC-V assembly code properly implements the desired functionality described in C and satisfies the RISC-V calling convention. If either the functionality or the calling convention are not satisfied, then provide code that is functionally correct and satisfies the calling convention.

Note that ‘g’ refers to another function whose specification is not provided to you. Also, note that there may be multiple errors in the code. You should correct them all.

(A) (7 points)

```c
int f(int a, int b) {
    return g(a + b, b) + a;
}
```

<table>
<thead>
<tr>
<th>f:</th>
<th>Assembly code produces expected results and follows calling convention:</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv s1, a0</td>
<td>YES NO</td>
</tr>
<tr>
<td>add a0, a1, a0</td>
<td></td>
</tr>
<tr>
<td>jal ra, g</td>
<td></td>
</tr>
<tr>
<td>add a0, a0, s1</td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td></td>
</tr>
</tbody>
</table>

```c
Assembly code produces expected results and follows calling convention: |
YES NO |
If not, provide correct code here: |
G: |
(B) (6 points)

```c
int f(int a, int b, bool c) {
    if (c) return a * 2;
    else return a - b;
}
```

<table>
<thead>
<tr>
<th>Assembly code produces expected results and follows calling convention:</th>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
</table>

If not, provide correct code here:

```
f:   bnez a2, L1
    slli a0, a0, 2
    ret
L1:  sub a0, a1, a0
    ret
```

---

(C) (7 points)

```c
unsigned int f(unsigned int a, int b) {
    if (b < 0) {
        return g(a) + b;
    }
    else return a >> b;
}
```

<table>
<thead>
<tr>
<th>Assembly code produces expected results and follows calling convention:</th>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
</table>

If not, provide correct code here:

```
f:   slt t0, a1, x0
    beqz t0, L1
    addi sp, sp, -4
    sw ra, 0(sp)
    jal ra, g
    add a0, a0, a1
    ret
L1:  sra a0, a0, a1
    ret
g:   
```
Problem 3. Procedures and Stacks (20 points)

You are given an incomplete listing of a C procedure and its translation to RISC-V assembly code (shown on the right):

```c
int f(int a, int b) {
    if (a < b)
        return ???
    else
        return a | b;
}
```

(A) (3 points) Give the HEX encoding of the ‘sw ra, 8(sp)’ instruction.

Hex encoding of sw ra, 8(sp): 0x __________

(B) (3 points) What is the missing C expression corresponding to the ‘???’ in the above program?

(give C code expression)

(C) (2 point) How many words are stored on the stack every time we want to execute function \( f \) recursively?

Number of words pushed onto the stack for each recursive call to \( f \)? ______
The program’s initial call to function \( f \) occurs outside of the function definition via the instruction ‘jal ra, \( f \)’. The program is interrupted at an execution (not necessarily the first) of function \( f \), just prior to the execution of the \texttt{addi sp, sp, 12} instruction 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 \( 0x40 \) and points to the location shown in the diagram.

(D) (3 points) What are the arguments to the original call to \( f \)? Write CAN’T TELL if you can’t tell.

\[
\text{Original arguments to } f, \ a = \ \ \ \ \ \ ; \ b = \ \ \ \ \ \ \ \\
\]

(E) (3 points) What are the arguments to the current call to \( f \)? Write CAN’T TELL if you can’t tell.

\[
\text{Current arguments to } f, \ a = \ \ \ \ ; \ b = \ \ \ \ \\
\]

(F) (2 points) What value was in SP just prior to the initial call to \( f \)?

\[
\text{Initial contents of SP: } 0x \ \ \ \ \\
\]

(G) (2 points) What is the address of the ‘jal ra, \( f \)’ instruction that made the original call to \( f \)?

\[
\text{Address of original call to } f: 0x \ \ \ \ \\
\]

(H) (2 points) What is the hex address of the instruction at L1?

\[
\text{Address of instruction at L1: } 0x \ \ \ \\
\]

END OF QUIZ 1!