1) RISC-V Assembly Programs and Translation

C version:
```c
int f(int *a, int b, int length) {
    int res = 0;
    for (int i = 0; i <= (length); i=i+1) {
        int c = a[i] + b;
        if (c & 1 == 0){
            res += 2;
        }
        else {
            res += c - 1;
        }
    }
    return res;
}
```

Convert this program into RISC-V assembly

**ANSWER:**
```riscv
// a0 = a, array base pointer  
// a1 = b  
// a2 = length

f:        li   a4, 0  // a4 = res
        li   a5, 0  // a5 = c
        li   a6, 0  // a6 = i
        j    check  // need to check for loop
              // condition before first iter

for:      addi a7, a0, a6  // a7 has a[i] address
        lw    a5, 0(a7)  // c = a[i]
        addi a6, a6, 4  // i+=4, offset coordinates
        addi a2, a2, -1  // decrement length to track iters
        add    a5, a5, a2  // c = a[i] + b
        andi a7, a7, 1  // a7 = c & 1
        beqz a7, a5, 1  // if a7 is 0, goto if, o.w. do else

else:     addi a5, a5, -1  // a5 = c - 1
        add    a4, a4, a5  // res += c - 1
        j    check  // goto for loop check

if:        addi a4, a4, 2  // res += 2, fall thru to check

check:     bgt   a2, zero, for  // length = 0 when done length iters

end:       mv    a0, a4  // answer returned in a0
        ret    // return
```
The function `drawFrame` draws one frame from a 17 second animation of the solar system onto a 512×512 pixel display. It takes two arguments; the first is the frame number and the second is brightness. The source for this function (and the additional functions it calls) is on the next page, but for reference, a version in C is given below.

```c
int time = time % 512; // 17 sec × 30 fps = 510 frames
drawTerraLuna(time, intensity);
point(0, 0, intensity); // draw Sol
}

drawTerraLuna(int time, int intensity) {
    int ex = EARTHPOS[2*time];
    int ey = EARTHPOS[2*time+1];
    int mx = MOONPOS[2*time];
    int my = MOONPOS[2*time+1];
    point(ex, ey, intensity);
    point(mx, my, intensity);
}

point(int x, int y, int intensity) {
    VRAM[y<<9 + x] = intensity; // 512×512 display
}
```

**Question A**  Draw the function call tree of `drawFrame(62, 999)`.

```
drawFrame(62, 999)
    |  
    |   --- drawTerraLuna(62, 999)
    |       
    |       |   --- point(?, ?, 999)
    |       |       
    |       |       |   --- point(?, ?, 999)
    |       |       |
    |       |       +-- point(0, 0, 999)
```

**Question B**  Execution is halted at the label `BREAK`, and a dump of the stack is given in `LISTING 2`. For each word shown, indicate the source register of the `lw` instruction that placed the word on the stack, or, if the source register cannot be determined, write “UNKNOWN”.

See `LISTING 2` for solution.

**Question C**  Rewrite `drawTerraLuna` to use only caller-saved registers (excepting `sp`).

See `LISTING 3` for solution.
LISTING 1

**drawFrame:**
andi a0, a0, 0x1FF
addi sp, sp, -8
sw ra, 4(sp)
sw a1, 0(sp)
call drawTerraLuna
li a0, 0
li a1, 0
lw a2, 0(sp)
call point
lw ra, 4(sp)
addi sp, sp, +8
ret

drawTerraLuna:
addi sp, sp, -12
sw ra, 8(sp)
sw s0, 4(sp)
sw s1, 0(sp)
slli s0, a0, 3  # s0 ← 2×time (address offset)
mv s1, a1  # s1 ← intensity
li t0, 0x4000  # start of EARTHPOS data
add t0, t0, s0
lw a0, 0(t0)
lw a1, 4(t0)
mv a2, s1
call point

BREAK:
li t0, 0x5000  # start of MOONPOS data
add t0, t0, s0
lw a0, 0(t0)
lw a1, 4(t0)
mv a2, s1
call point

lw s1, 0(sp)
lw s0, 4(sp)
lw ra, 8(sp)
addi sp, sp, +12
ret

**point:**
li t0, 0x8000  # start of VRAM (video memory)
slli a0, a0, 2
add t0, t0, a0
slli a1, a1, 11
add t0, t0, a1
sw a2, 0(t0)
ret
LISTING 2

ADDRESS 0

↑

0x0000 0000  UNKNOWN
0x0000 0000  UNKNOWN
0x0000 0724  # od1828  UNKNOWN

SP →

0x0000 0000  s1 from line 4 of drawTerraLuna
0x0000 0000  s0 from line 3 of drawTerraLuna
0x0000 0724  # od1828  ra from line 2 of drawTerraLuna
0x0000 03E7  # od999  a1 from line 4 of drawFrame
0x0000 038C  # od908  ra from line 3 of drawFrame

↓

ADDRESS ∞

LISTING 3

drawTerraLuna:  addi sp, sp, -12
             slli a0, a0, 3  # s0 ← 2 × time (address offset)
             mv   a1, a1
             sw   ra, 8(sp)
             sw   a0, 4(sp)
             sw   a1, 0(sp)
             li   t0, 0x4000  # start of EARTHPOS data
             add  t0, t0, a0
             mv   a2, a1
             lw   a1, 4(t0)
             lw   a0, 0(t0)
             call point

             li   t0, 0x5000  # start of MOONPOS data
             lw   a0, 4(sp)
             add  t0, t0, a0
             lw   a2, 8(sp)
             lw   a1, 0(t0)
             lw   a0, 8(t0)
             call point

             lw   ra, 8(sp)
             addi sp, sp, +12
             ret
3 Premonitions
Boolean simplification

Simplify the following expressions; specifically, into minimal sums of products.

\[ AB + (C + AB) \]
\[ = A(B + B) + C \]
\[ = A + C \]

\[ (ABC)(A(\bar{A} + B) + C(A + B)) \]
\[ = ABC(A \bar{A} + AB + AC + BC) \]
\[ = ABC(A \bar{A} + AB + AC + BC) \]
\[ = ABC(AB + BC + CA) \]

We can read this as “true if 1 at least two of the inputs are true, and 2 the inputs are not all true,” or in other words, “true only if two (and exactly two) of the inputs are true.” If we keep going,
\[ = (\bar{A} + B + \bar{C})(AB + BC + CA) \]
\[ = A\bar{A}B + ABC + A\bar{A}C + AB\bar{B} + B\bar{B}C + ABC + AB\bar{C} + BC\bar{C} + AC\bar{C} \]
\[ = ABC + A\bar{B}C + AB\bar{C} \]

which explicitly enumerates all the cases where two and only two inputs are true.
4) Composing Bluespec

Convert the following into Bluespec:

*Self-Masking Circuit:*

*This circuit takes a 4-bit input A and performs a masking operation on it by xor-ing it with itself shifted left once, right twice, and left 3 times.*

**ANSWER:**

```plaintext
function Bit#(4) foo(Bit#(4) a);
    Bit#(4) a1l = a << 1;
    Bit#(4) a2r = a >> 2;
    Bit#(4) a3l = a << 3;
    Bit#(4) res = a ^ a1l ^ a2r ^ a3l;
    return res;
endfunction
```

**Parameterized Functions**

Convert the 4-bit Self-Masking Circuit into an n-bit version that xors A with 1 through n-1 shifts of A in alternating directions

**ANSWER:**

```plaintext
function Bit#(w) foo(Bit#(w) a);
    Bit#(4) res = a;
    for (Integer i = 1; i < valueof(w); i=i+1) begin
        if (i % 2 == 0) begin
            res = res ^ (a >> i);
        end
        else begin
            res = res ^ (a << i);
        end
    end
    return res;
endfunction
```
5) Circuit Synthesis

Create circuits for the following boolean expressions

1) \((AB) + (C + (A \overline{B}))\)

ANSWER:

2) \((\overline{ABC})(A + B)(\overline{A} + C)\)
3) \((AB\overline{CD}) \land (\overline{A}BCD)\)

Give a boolean expression for each output bit of this circuit

\[
\text{ANSWER:}
\]
\[
\begin{align*}
\text{out}[0] &= (a[0] \land (\overline{a}[2] \land a[1])) \\
\text{out}[1] &= (a[3] \land (\overline{a}[2] \land a[1]))
\end{align*}
\]