RISC-V Calling Conventions:

- Caller places arguments in registers a0-a7
- Caller transfers control to callee using jal (jump-and-link) to capture the return address in register ra
  - jal ra, label: R[ra] <= pc + 4; pc <= label
  - jal label

- Callee runs, and places results in registers a0 and a1
- Callee transfers control to caller using jr (jump-register) instruction
  - ret: pc <= R[ra]
  - jr ra
  - jalr x0, 0(ra)

Push register xi onto stack
addi sp, sp, -4
sw xi, 0(sp)

Pop value at top of stack into register xi
lw xi, 0(sp)
addi sp, sp, 4

Assume 0(sp) holds valid data.

**Stack discipline:** can put anything on the stack, but leave stack the way you found it

- Always save s registers before using them
- Save a and t registers if you will need their value after procedure call returns.
- Always save ra if making nested procedure calls.
Note: A small subset of essential problems are marked with a red star (★). We especially encourage you to try these out before recitation.

Problem 1.

Integer arrays season1 and season2 contain points Ben Bitdiddle had scored at each game over two seasons during his time at MIT Intramural Basketball Team. Please write a RISC-V assembly program which counts the number of games he scored more than 20 points. An equivalent Python program is given below. Note that the base addresses for arrays season1 and season2 along with their size are passed down to function greaterthan20.

```python
import numpy as np

def main():
    season1 = np.array([18, 28, 19, 33, 25, 11, 20])
    season2 = np.array([30, 12, 13, 33, 37, 19, 22])
    result = greaterthan20(season1, season2, len(season1))
    print(result)

def greaterthan20(a, b, size):
    count = 0
    for i in range(size):
        if a[i] > 20:
            count += 1
        if b[i] > 20:
            count += 1
    return count

# Beginning of your assembly code
greaterthan20:
    li t0, 0 // t0 <= count
    li t1, 0 // t1 <= index
    li t2, 20
loop :
    ble a2, t1, endloop
    sll t1, t1, 2 // 4*index
checka:
    add t3, a0, t1
    lw t4, 0(t3)
    ble t4, t2, checkb // if a[i] <= 20, then check b[i]
    addi, t0, t0, 1 // increment counter
checkb:
    add t3, a1, t1
    lw t4, 0(t3)
    ble t4, t2, endcompare // if b[i] <= 20, then go to endcompare
    addi, t0, t0, 1 // increment counter
endcompare:
    add t1, t1, 1 // increment index
    j loop
endloop:
    mv a0, t0
    ret
```

// Beginning of your assembly code

```assembly
greaterthan20:
    li t0, 0 // t0 <= count
    li t1, 0 // t1 <= index
    li t2, 20
loop :
    ble a2, t1, endloop
    sll t1, t1, 2 // 4*index
checka:
    add t3, a0, t1
    lw t4, 0(t3)
    ble t4, t2, checkb // if a[i] <= 20, then check b[i]
    addi, t0, t0, 1 // increment counter
checkb:
    add t3, a1, t1
    lw t4, 0(t3)
    ble t4, t2, endcompare // if b[i] <= 20, then go to endcompare
    addi, t0, t0, 1 // increment counter
endcompare:
    add t1, t1, 1 // increment index
    j loop
endloop:
    mv a0, t0
    ret
```
Problem 2. ★

The following C program computes the log base 2 of its argument. The assembly code for the procedure is shown on the right, along with a stack trace showing the execution of ilog2(10). The execution has been halted just as it's about to execute the instruction labeled “rtn:” The SP label on the stack shows where the SP is pointing to when execution halted.

```c
/* compute log base 2 of arg */
int ilog2(unsigned x) {
    unsigned y;
    if (x == 0) return 0;
    else {
        /* shift x right by 1 bit */
        y = x >> 1;
        return ilog2(y) + 1;
    }
}
```

(A) Please fill in the values for the two blank locations in the stack trace shown on the right. Please express the values in hex.

**Fill in values (in hex!) for 2 blank locations**

(B) What are the values in a0, s0, sp, and pc at the time execution was halted? Please express the values in hex or write “CAN'T TELL”.

<table>
<thead>
<tr>
<th>SP →</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x93</td>
</tr>
<tr>
<td>0x240</td>
</tr>
<tr>
<td>0x1</td>
</tr>
<tr>
<td>0x240</td>
</tr>
<tr>
<td>0x2</td>
</tr>
<tr>
<td>0x240</td>
</tr>
<tr>
<td>0x5</td>
</tr>
<tr>
<td>0x1108</td>
</tr>
<tr>
<td>0x37</td>
</tr>
</tbody>
</table>

Value in a0: 0x__________ in s0: 0x__________
Value in sp: 0x__________ in pc: 0x__________

a0 = 2, s0 = 2, sp = Can't tell, pc = 0x250

(C) What was the address of the original ilog2(10) function call?

**Original ilog2(10) address:** 0x__________

0x1104
Problem 3. ★

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

```c
int fn(int x) {
    int lowbit = x & 1;
    int rest = x >> 1;
    if (x == 0) return 0;
    else return ???;
}
```

(C) What is the missing C source corresponding to ??? in the above program?

C source code: ________________________________

```c
fn(rest) + lowbit
```

```assembly
fn: addi sp, sp, -12
    sw s0, 0(sp)
    sw s1, 4(sp)
    sw ra, 8(sp)
    andi s0, a0, 1
    srai s1, a0, 1
    yy: beqz a0, rtn
    mv a0, s1
    jal ra, fn
    add a0, a0, s0

rtn: lw s0, 0(sp)
    lw s1, 4(sp)
    lw ra, 8(sp)
    addi sp, sp, 12
    jr ra
```
The procedure `fn` is called from an external procedure and its execution is interrupted just prior to the execution of the instruction tagged `yy:`. The contents of a region of memory are shown on the left below. If the answer to any of the below problems cannot be deduced from the provided information, write “CAN’T TELL”.

(B) What was the argument to the most recent call to `fn`?

**Most recent argument (HEX):** \( x=0x11 \)

(C) What is the missing value marked `??` for the contents of location 1D0?

**Contents of 1D0 (HEX):** CAN’T TELL

(D) What is the hex address of the instruction tagged `rtn`?

**Address of rt (HEX):** 0x50

(E) What was the argument to the first recursive call to `fn`?

**First recursive call argument (HEX):** \( x=0x23 \)

(F) What is the hex address of the `jal` instruction that called `fn` originally?

**Address of original call (HEX):** 0xC0

(G) What were the contents of `s1` at the time of the original call?

**Original s1 contents (HEX):** 0x22

(H) What value will be returned to the original caller if the value of `a0` at the time of the original call was 0x47?

**Return value for original call (HEX):** 0x4

counts the number of 1’s in original number

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SP→</td>
<td>0x1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rest</td>
<td>0x11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ra</td>
<td>0x4C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lowbit</td>
<td>0x1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rest</td>
<td>0x23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ra</td>
<td>0x4C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orig s0</td>
<td>0x3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orig s1</td>
<td>0x22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orig ra</td>
<td>0xC4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>