15213 - Recitation 1 - Datalab - S17

Introduction

In this activity you will review the material on integers, binary, and floating-point necessary for datalab. This activity was based on material developed by Professor Saturnino Garcia of the University of San Diego. It is used here with permission.

Before you begin, take a minute to assign roles to each member in your group. Below is a summary of the four roles; write the name of the person taking that role next to the summary.

If your group only has three members, combine the roles of Facilitator and Process Analyst.

- **Facilitator**: Reads question aloud; keeps track of time and makes sure everyone contributes.
- **Quality Control**: Records all answers & questions, and provides the same to team & instructor.
- **Spokesperson**: Talks to the instructor and other teams. Runs programs when applicable.
- **Process Analyst**: Considers how the team could work and learn more effectively.

Fill in the following table showing which group member is performing each role:

Role	**Person**
Facilitator	
Quality Control	
Spokesperson	
Process Analyst	

Model 010: Representing Negative Values in Binary

While there are several approaches to represent negative binary numbers, we'll focus on the dominant representation: *two's complement*, which is capable of representing both positive and negative numbers.

1. What is the leftmost bit in a non-negative number? Of a negative number?
2. What is the two's complement representation of of the following decimal numbers?
   - 3:
   - -8:
3. Complete the following table to indicate the most positive (i.e. largest) and most negative (i.e. smallest) number that can be represented with a given number of bits *when using two's complement representation*. Use the table above (which you simplified by removing unnecessary bits) to help you answer this question.

Bits	Most Positive	Most Negative
1	0	-1
2	1	-2
3		
4		

Model 1: Bit-Level Operations

In the 1800s, George Boole proposed a logic-system based on two values: 0 and 1. We will work through how these operations are exposed in C, and by extension the underlying system. The first set of operations are performed by treating the integer as a bit-vector.

1. There are three other bit-wise operations: AND (&), OR (|), and XOR (^). Each is applied between two bits. AND gives the value 1 when both operands are 1. OR gives the value of 1 when at least one operand is 1. And XOR gives the value of 1 when only 1 operand is 1.
   Fill in the following table using bit-wise AND:

Dec	Bin	X & 0x1
-2	1110	0000
-1		
0	0000	0000
1		
2		

2. For which numbers was X & 0x1 not 0000? What is a common property of these integers?

3. De Morgan's Law enables one to distribute negation over AND and OR. Given the following expression, complete the following table to verify for the 4-bit inputs. ~(x & y) == (~x) | (~y)

x	y	~(x & y)	(~x) | (~y)
0xF	0x1		
0x5	0x7		
0x3	0xC		

Model 2: Logical Operations

This section will explore logical operations. These operations contrast with bit-level in that they treat the entire value as a single element. In other languages, the type of these values would be termed, "bool" or "boolean". C does not have any such type. Instead, the value of 0 is false and all other values are true.

The three operators are AND (&&), OR (||), and NOT (!). "!" is commonly termed "bang".

1. Evaluate the following expression: (0x3 && 0xC) == (0x3 & 0xC)

2. Test whether !!X == X holds across different values of X. Do the same for bitwise complement.

X	!X	!!X	~X	~~X
-1				
0				
1				
2				

Model 3: Shifts, Multiplication and Division

1. You may recall that when a 0 is appended to the right of a binary number, its value is increased. Given the expression X = (0x1 << 2) | (0x1 << 1) , what is the value of X in decimal and binary?

The compiler can often detect simple multiplication and replace it with shifts and addition.

2. Suppose we right shift the value of "-2" by 1. What value do we expect?
3. With 4-bit integers, what is the binary for -2? After right shifting by 1, what value(s) might we have?
4. Given the unsigned, 4-bit value of 0xA, what should the result be when right shifting by 1?
5. In converting / printing from decimal to binary, it might be suggested to divide by 2 and take the remainder. Fill in this algorithm into the following code using bit-wise and shift operations:

```
unsigned x = ...;
while (x != 0)
{
    int rem = x _______ ;
    x = __________;
}
```

Model 4: Integrating the operations

As the previous questions show, an `int` can be used to represent a number, an ordered sequence of bits (also known as a bit vector), or even a boolean value. It might seem like `{+, -, *, /}` are only used for arithmetic, `{&, |, ^, >>, <<}` are only used for bit manipulation, and `{!, &&, ||}` are only used for boolean logic. However, it can be useful to mix operations between these groups.

1. Compare the bit-level representations of each number and its negative.

`x`	`x` in binary	`-x` in binary
`15`		
`22`		
`-8`		
`0x15213000`		

Now write an expression that isolates the least significant 1 bit of an `int`.
(E.g. `leastSigOneBit(0x9C0) = 0x40`)

```
leastSigOneBit(x) = ____________________________________________;
```

2. Find an algorithm for computing the expression `(cond) ? t : f`, which equals `t` if `cond` is `1` and `f` if cond is `0`.

```
int conditional(int cond, int t, int f) {
  /* Compute a mask that equals 0x00000000 or
       0xFFFFFFFF depending on the value of cond */

  int mask = ____________________________________________;

  /* Use the mask to toggle between returning t or returning f */

  return ____________________________________________;
}
```