15213 - Recitation 3 - Datalab

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 (ˆ). 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 1: What is floating point?

1. Write 15213 in scientific notation. Describe the different parts of the notation.

Model 2: Binary Scientific Notation

1. Write out the following binary numbers in scientific (binary) notation. In this notation, we use powers of 2 rather than powers of 10.

Value	Binary
5 3/4	101.11
2 7/8	10.111
1 7/16	1.0111

2. What value(s) is to the left of the binary point in each number?

Model 3: IEEE Representation

This model starts the review of the IEEE floating point representation.

Value $= (-1)^s * 1.f * 2^E$

1. What does s represent in the equation? If the value for s is 1, what happens to the number?
2. The fractional bits, f, have an implicit 1 in the front. This pattern is termed, "normalized". Returning to the table in model 2, what are the values for f for each number?
3. With the 8 exponent bits of a single-precision float, what is the smallest non-zero number with 0x01 as the exponent bits? Is this value greater or less than one?
4. IEEE subtracts a bias from the exponent bits to adjust the range of representable values. For the single-precision floats, the bias is 127. With $E = exp - bias$, what exponent bits (exp) is required for the number in the previous question?

Model 4: Extreme Exponents

1. Suppose we have 5 bits to represent the fractional component, x.xxxx. If this bit pattern must be normalized, then what is the smallest number that we can store, assuming an exponent of 0?
2. For what you know of the representation (i.e. normalized) so far, can you represent 0 with the IEEE format?

One of the goals of the IEEE representation was to make the bit pattern of 0 be the value of 0. Thererfore, in the IEEE representation, an exponent value of 0 is made a special case. If the exponent value is 0, then the biased exponent is the same as having an exponent value of 1. Furthermore, the fractional part is "denormalized", such that it no longer has a leading 1.

3. How many representations for zero are there with denormalized floats?
4. If the 5-bit pattern from before does not need to be normalized, then what is the smallest non-zero number that you can represent with these bits?

Model 5: Addition

To perform addition with floating point numbers, the hardware first works to line up the binary points. And then it can add the significands together. Finally, the result must be renormalized and a new exponent computed.

1. Compute the sum of the following binary numbers: $1.010 * 2^2 + 1.110 * 2^3$. Apply each step in turn.
2. How many fractional bits were required for the result of the previous question?
3. There are several possible rounding schemes for floating point values. There are two components of rounding. First, is what to do in general? Should the float be rounded up, down, to zero, or to nearest? The second component is what to do about ties with round to nearest. So should 9/2.0 be 4 or 5? The default IEEE scheme is round to nearest even. Apply it to the following values for a system that only has three bits for the final values.

Value	Binary	Rounded	Final
1 3/32			
1 5/32			
1 7/8			
1 5/8			

Model 6: Simple Floating-point

For the following model, we are going to use a smaller representation. 1 bit for the sign. 3 bits for the exponent, with a bias of 3. And 4 fractional bits.

1. What is the largest absolute value that we can represent with these bits? Smallest non-zero?
2. Given the following value 0x5C, what is this 8-bit float's decimal value? What is the binary representation of this value?
3. Compute the sum of the following floats (hex values shown): `0x5C + 0x43`.
4. Compute the product of the following floats (hex values shown): `0x5C * 0x43`.

Model I: Bit Puzzle

Given a 32-bit float, convert the float to an integer assuming that the value is within the range (TMIN, TMAX) using integer bitwise and shift operations.

```
unsigned float2i(unsigned);
```

1. Start by identifying the fields of the float. Write code to assign each to a separate variable.
2. Convert each field from its stored form to its full value.
3. Apply the exponent to the fractional bits.
4. What other components exist in the float?

Model R: Review

1. (advanced) Will the following code ever terminate? If so, what value will sum contain?

```
float sum = 0.0f;
float inc = 1.0f;
float tsum;

do {
    tsum = sum;
    sum += inc;
} while (tsum != sum);
```

2. If the floats and constants are changed to ints, how (if at all) do your answers change?