15213 - Lecture 3 - POGIL Activity (Integer Operations)

Introduction

In this activity you will learn about integer operations.

Before you begin, take a minute to assign roles to each member in your group. Try to switch up the roles as much as possible: please don't pick a role for yourself that you have already done more than once. 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.

For this and all future activities, the Facilitator should also take the role of the reader and read the questions aloud to the group.

- **Facilitator**: Reads question aloud; keeps track of time and makes sure everyone contributes appropriately.
- **Quality Control**: Records all answers & questions, and provides team reflection to team & instructor.
- **Spokesperson**: Talks to the instructor and other teams. Compiles and 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 0: Review of Addition / Positive

1. Add the following two binary numbers together.

$$\begin{array}{cccccc} & & 1 & 0 & 1 & 0 \\ + & & 1 & 1 & 0 & 0 \\ \hline \end{array}$$

2. How many bits were required for the *result* in the previous question?

3. How does the number of bits needed for the result compare to the number of bits in the numbers you added together?

4. If you only have 4 bits to represent the previous sum, what number(s) might you provide? Explain.

Model 0: Review of Negative Integers

1. Thinking about two's complement, what is the leftmost bit in a non-negative number?

2. 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		

3. Use your answer from the previous question to find an expression that gives the *most positive* number that can be represented by a N-bit two's complement number. Hint: This will be related to a power of two in some way.

4. Use your answer from the previous questions to find an expression that gives the *most negative* number that can be represented by a N-bit two's complement number. Hint: This will be related to a power of two in some way.

The most positive signed integer is termed TMax, and the most negative is TMin. The most positive unsigned integer is termed UMax.

5. Add the following two binary numbers together:

```
  1111000
+ 0100111
```

   Confirm that the result is correct by converting all the binary numbers to their decimal representation, both signed and unsigned. Are both results correct?

6. Given what you observed in the previous question, does the architecture need multiple adders in hardware for signed and unsigned addition?

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. Yesterday, one of the steps to negating a signed integer was to invert all of the bits. This operation, complement or "~", can be applied to any integer. For each integer, convert it from hexadecimal to binary and then apply the operation. What is the final hexadecimal integer?

   - ~0xCAFE
   - ~0x3C3C
   - ~0x0000

2. 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. Complete the table below.

OP0	OP1	AND	OR	XOR
0	0	0	0	0
1	0	0	1	1
0	1			
1	1			

3. Fill in the following table using AND:

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

4. For which numbers was the value & 0x1 not 0000? What is a common property of these integers?

5. Many times in systems programming, we want to test if a flag value is set in a given bit-pattern. The programmer will commonly use `X & FLAG == FLAG`. Give an explanation of this expression in pseudo-code.

6. Systems programmers will also regularly have to combine flag values. Describe how OR (|) is used in the following example, which creates a new file for writing:

   ```
   open(filename, (O_WRONLY | O_CREAT | O_TRUNC), ...);
   ```

7. De Morgan's Law enables one to distribute negation over AND and OR. Given the following expression, verify whether it is true over various 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. With 4-bit values, how many values are false and how many are true?
2. Evaluate the following expression: (0x3 && 0xC) == (0x3 & 0xC)
3. Test whether !!X == X holds across different values of X.

X	!X	!!X
-1		
0		
1		
2		

4. Now, for each of the previous values, substitute complement “~” for logical not. Do these results differ from the previous results?

Model 3: Multiplication and Division

1. We observed yesterday that when a 0 is appended to the right of a binary number, its value was increased. Appending these zeros has an operator, “<<”. Assume that the final integer is 32-bit.

Value	<<
0x30	1
0x5A	4
0x11D	31

2. Given the expression X = (0x1 << 2) | (0x1 << 1) , what is the value of X in decimal and binary?
3. The compiler can often detect simple multiplication and replace it with shifts and addition. Given the largest 3-bit unsigned integer, what is its value squared? How many bits does this

value require?

4. What is the result from the previous question if it must be stored in 3 bits.
5. Shift can also move the digits the other direction with ">>". Compute the following.

Value	>>
0x30	1
0x5A	4
0x11	3

6. Convert the initial and final values in the previous question to decimal. To what common operation is right shift equivalent?
7. Suppose we right shift the value of "-2" by 1. Based on the previous question, what value do we expect?
8. With 4-bit integers, what is the binary for -2? After right shifting by 1, what value(s) might we have?
9. Given the unsigned, 4-bit value of 0xA, what should the result be when right shifting by 1?
10. Yesterday, we covered converting from decimal to binary, it was suggested to divide by 2 and take the remainder. Fill in this algorithm into the following code using the operations learned today:

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