15213 - Recitation 1 - 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.

Each activity is designed to be solved in groups and take approximately 10 minutes.

Activity 1: Bit-level and Logical

1. 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	~0b0001 = 0b1110	0b0000 | 0b1110 = 0b1110
	0x5	0x7	~0b(0b0101 & 0b0111)=1010	1010 | 1000 = 1010
	0x3	0xC	~0000 = 1111	1100 | 0011 = 1111
0011		1100		

The next section will explore logical operations. These operations contrast with bit-level operations in that they treat the entire value as a single element. In other languages, the type of these values would be called "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 logical operators are AND (`&&`), OR (`||`), and NOT (`!`).
"`!`" is commonly termed "bang".

2. Evaluate the following expression:

```
(0x3 && 0xC) == (0x3 & 0xC)
```

L = 1
R = 0011 & 1100 = 0
L != R

3. Test whether `(!!X) == X` holds across different values of `X`. Do the same for bitwise complement: `(~~X) == X`.

	X	!X	!!X	~X	~~X
0b1111	-1	0	1	0000	1111
0b0000	0	1	0	1111	0000
0b0001	1	0	1	1110	0001
0b0010	2	0	1	1101	0010

Activity 2: Shifts, Negation and Conditional

1. Suppose we right shift the value of “-2” by 1. What value do we expect?

   -1

2. With 4-bit integers, what is the binary for -2? After right shifting by 1, what value(s) might we have?

   -2 = 0b1110
   -1

3. Fill in the following table, assuming you only have 4 bits to represent the two’s complement integer.

x	x in binary	-x in binary
1	0001	1111
2	0010	1110
7	0111	1001
-8	1000	-- 1000

4. Find an algorithm for computing the expression `(cond) ? t : f`, which evaluates to `t` if `cond` is `1` and `f` if `cond` is `0`. Assume `cond` will either be `1` or `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 = __-cond__________________________;

    /* Use the mask to toggle between returning t or returning f */
    return __(mask & t) | (mask | f)______________;
}
```

(mask & t) | (~mask & f)

cond == 0 --> mask == 0
--> mask & x == mask   mask | x == x

cond == 1 --> mask == -1
--> mask & x == x   mask | x == mask

cond == 0 --> mask == 0
--> mask & x == 0   ~mask & x == x

cond == 1 --> mask == -1
--> mask & x == x   ~mask & x == 0

Activity 3: Divide and Conquer (Bit Count)

Let’s count how many bits are set in a number. For each challenge, you can use any operator allowed in the integer problems in datalab.

Using 1 operator, we return the number of bits set in a 1-bit number:

```
int bitCount1bit(int x) {return x;}
```

1. How about if there are two bits in the input? (4 ops max)

```
int bitCount2bit(int x) {
    int bit1 = _x & b01___;
    int bit2 = ____(x>>1)&b01 _______;
    return ___bit2____ + bit1 ;
}
```

2. How about if there are four bits? (8 ops max)

```
int bitCount4bit(int x) {
    int mask = 0b0101__________;
    int halfSum = x & mask + (x>>1)&mask;
    int mask2 = 0b0011__________;
    return ___________ + ___________ ;
}
       mask2 & halfSum + mask2 & (halfSum>>2)
```

x0babcdhalfSum: 0b0b0d + 0b0a0cmask2 & halfS

3. How about if there are eight bits? (12 ops max)

```
int bitCount8bit(int x) {
    int mask = 0b01010101_;
    int quarterSum = mask & x + mask & (x>>1);
    int mask2 = 0b00110011_;
    int halfSum = mask2 & quarterSum + mask2 & (quarterSum>>2);
    int mask3 = 0b01110111_;    0b00001111
    return ___________ + ___________ ;
         mask3 & halfSum     mask3 & (halfSum >> 3)
}
                             mask3 & (halfSum >> 4)
```