15213 - Recitation 2 - 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	**0b1110**	**0xE**
0x5	0x7	**0b1010**	**0xA**
0x3	0xC	**0b1111**	**0xF**

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".

2. Evaluate the following expression: (0x3 && 0xC) == (0x3 & 0xC) **0x1 =/= 0x0**
3. Test whether (!!X) == X holds across different values of X. Do the same for bitwise complement.

X	!X	!!X	~X	~~X
-1	**0**	**1**	**0**	**-1**
0	**1**	**0**	**-1**	**0**
1	**0**	**1**	**-2**	**1**
2	**0**	**1**	**-3**	**2**

Activity 2: Shifts, Negation and Conditional

1. Suppose we right shift the value of "-2" by 1. What value do we expect? **-2 / 2 = -1**
2. With 4-bit integers, what is the binary for -2? After right shifting by 1, what value(s) might we have? **0b1110 » 1 = 0b1111**
3. Fill in the following table, assuming you only have 4 bits to represent the 2s complement

integer.

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

4. 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 = ________(cond<<31)>>31____________;

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

 return ____(mask "and" t) "or" (~mask "and" f)_____;
}
```

Activity 3: Floating-point

1. How many representations for zero are there with denormalized floats? Are any of these representations the same for zero as an integer? **2; 1 (all zeros)**
2. Which is larger, 2^{127} or $+\inf$? Does this ordering hold when these numbers are floats (i.e., if just the bit patterns are compared)? **+inf; yes 0b0|1111 1110|0... vs 0b0|1111 1111|0...**
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 fractional component of the final value, so final binary value should be `1.xyz`.

Value	Binary	Rounded	Final
1 3/32	**1.00011**	**11**	**1.001**
1 5/32	**1.00101**	**01**	**1.001**
1 7/8	**1.111**	**-**	**1.111**
1 5/8	**1.101**	**-**	**1.101**

Activity 4: Divide and Conquer (Bit Parity)

Let's determine whether a number has an even number of 1-bits or an odd number. You can use any operator allowed in datalab. Return a 0 if there's an even number and 1 if odd. Using 1 op, you can return the parity of a 1-bit number. `int bitParity1bit(int x) {return x;}`

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

```
int bitParity2bit(int x)
{
```

```
  int bit1 = __0b01 & x_________;
  int bit2 = ____0b01 & (x >> 1)_______;
  return ____bit1_______ ^ ____bit2_______ ;
}
```

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

```
int bitParity4bit(int x)
{
  int mask = ___0b0101________;


  int halfParity = ____(mask & x) ^ (mask & (x >> 1))_______;


  int mask2 = ____0b0011_______;


  return ___(mask2 & halfParity)________ ^ ___(mask2 & (halfParity >> 2))________ ;
}
```

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

```
int bitParity8bit(int x)
{
  int mask = ____0b01010101_______;


  int quarterParity = ____(mask & x) ^ (mask & (x >> 1))_______;


  int mask2 = ___0b00110011________;


  int halfParity = ___(mask2 & quarterParity) ^ (mask2 & (quarterParity >> 2))________;


  int mask3 = ____0b00001111_______;


  return __(mask3 & halfParity)_________ ^ ____(mask3 & (halfParity >> 4))_______ ;
}
```

Activity 5: Divide and Conquer (Bit Count)

Let's count how many bits are set in a number. For each challenge, you can use any allowed operator allowed in the integer problems in datalab. Using 1 op, 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 = __0b01 & x_________;
  int bit2 = ____0b01 & (x >> 1)_______;
  return ____bit1_______ + ____bit2_______ ;
```

```
}
```

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

```
int bitCount4bit(int x)
{
  int mask = ___0b0101________;


  int halfSum = ____(mask & x) + (mask & (x >> 1))_______;


  int mask2 = ____0b0011_______;


  return ___(mask2 & halfSum)________ + ___(mask2 & (halfSum >> 2))________ ;
}
```

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 = ____0b00001111_______;


  return __(mask3 & halfSum)_________ + ____(mask3 & (halfSum >> 4))_______ ;
}
```