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	**0b1110**	**0xE**
0x5	0x7	**0b1010**	**0xA**
0x3	0xC	**0b1111**	**0xF**

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)
```

0x1 =/= 0x0

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

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?

   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 two's complement integer.

x	x in binary	-x in binary
1	**0b0001**	**0b1111**
2	**0b0010**	**0b1110**
7	**0b0111**	**0b1001**
-8	**0b1000**	**-int min is int min**

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 = 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));
}
```

Activity 4: Divide and Conquer (isPalindrome) - dont give out the answer - hint at it though - walk through an example

Let's check to see if a number is a palindrome. For each challenge, you can use any operator allowed in the integer problems in datalab.

Using 1 operator, we return the 1 if the number is a palindrome, 0 otherwise:

1. How about if there are two bits in the input? (5 ops max)

```
int isPalindrome2bit(int x) {
    int hibit = 0b01 & (x >> 1);
    int lobit = 0b01 & x;
    return !(hibit ^ lobit); }
```

2. How about if there are four bits? (10 ops max)

```
int isPalindrome4bit(int x) {
    int mask2 = 0b11;
    int hi2 = mask2 & (x >> 2);
    int lo2 = mask2 & x;
    int mask1 = 0b01;
    int newhi1 = (lo2 & mask1) << 1;
    int newlo1 = (lo2 >> 1) & mask1;
    int lo2Reverse = newhi1 | newlo1;
    return !(hi2 ^ lo2Reverse); }
```

3. How about if there are eight bits? (15 ops max)

```
int isPalindrome8bit(int x) {
    int mask4 = 0b1111;
    int hi4 = mask4 & (x >> 4);
    int lo4 = mask4 & x;
    int mask2 = 0b11;
    int newhi2 = (lo4 & mask2) << 2;
    int newlo2 = (lo4 >> 2) & mask2;
    int lo4R2 = newhi2 | newlo2;
    int mask1 = 0b0101;
    int newhi1 = (lo4R2 & mask1) << 1;
    int newlo1 = (lo4R2 >> 1) & mask1;
    int lo4R4 = newhi1 | newlo1;
    return !(hi4 ^ lo4R4);
}
```