Bitwise Computations in C

The purpose of this lab is to introduce you to the use of C's bitwise operators for manipulating binary sequences in C. It is also an opportunity for you to review and assess your C programming knowdledge. While the C programming requirements of this course will be quite basic, there is unlikely to be any review of C programming during the lectures. For further assistance with C check out links such as:

https://cvw.cac.cornell.edu/Cintro/

http://www.w3schools.in/c/

Representation of Binary Sequences

1. The most common way of representing a binary sequence in C is with the "`unsigned int`" data type. Since this data type provides 4 bytes of data storage, the operations provided in C for bit-strings are applied to binary sequences of length 32. Shorter sequences can be represented using the least significant bits of a 32 bit sequence. Remember that bit positions are identified using "little endian notation" with bit position 0 be the rightmost or least significant bit position.

   Using your preferred test editor, open a new file called `bitlib.c` and enter:

   ```
   #include <stdio.h>
   ```

   You will be needing I/O functions in some of the exercises to follow.

2. Next, define a data type called "`bit32`" as follows:

   ```
   typedef unsigned int bit32;
   ```

3. Instances of this data type will have values that can be interpreted as binary sequences. The value of a "`bit32`" can be expressed as an 8-digit hexadecimal value. To output a "`bit32`" value, define the following function in your file `bitlib.c`:

   ```
   void printhex(bit32 x){
      printf("%.8x ", x);
   }
   ```

   Compile your file `bitlib.c` to obtain an object file, `bitlib.o`. correcting any syntax errors.

4. Hexadecimal constants are defined in C by a sequence of hexadecimal digits preceded by "0x" (i.e.," zero x", not "oh x"). For example, `0xFFFFFFFF` defines a binary sequence of 32 one's. When fewer than 8 hex digits are required, the left most bits are set to 0. For example, `0xF` is equivalent to `0000000F`.

   In a separate file called "`test_bitlib.c`" write a main program that tests your subprogram, `printhex` in the file `bitlib.o`. As a test of the function, define a loop that prints out the hex digits from 0 to F using `printhex`. If you don't know how to do this, ask the TA.

   You should note that the hex digits "a" through "f" are lower case. Both upper and lower case are acceptable representations of these digits.

5. Compile your file `test_bitlib.c`, linking it with `bitlib.o` to obtain a runtime file called "`ex_main`."

Operations on Binary Sequences

In what follows, you will be using the C bitwise operators to effect various changes to values of type `bit32`:

1. The "and" operator, "&," is used to extract the values in specific bit positions. For example, the following statement can be used to obtain the least significant byte of the 8 digit hexadecimal value 0x012345678:

   ```
   bit32 least_byte;
   least_byte = 0x12345678 & 0xff;
   printhex(least_byte);
   ```

   The value printed will be "00000078". The value, "0xff", is called a "mask."

   Similarly, to obtain the most significant (left-most) byte:

   ```
   bit32 left_byte;
   left_byte = 0x12345678 & 0xff000000;
   printhex(left_byte);
   ```

   The value printed would be "12000000". In this case the mask used is "0xff000000".

   In order to obtain the actual value of the byte, it is necessary to shift it to the right 3 byte positions; that is 24 bit positions. C provides an operator, ">>" , called the "right shift operator." The following statement will shift the value of `left_byte` 24 bit positions to the right:

   ```
   bit32 left_byte_value;
   left_byte_value = left_byte >> 24;
   printhex(left_byte_value):
   ```

   The value printed will be "00000012".

   In the file "`bitlib.c`" add a C function "`get_byte(bit32 hex_val, int n)`" that returns the "`bit32`" value of the byte in `hex_val` at position `n`, right justified as shown above. Compile and debug the updated `bitlib.c` file.

2. Rewrite your "`main`" program in `test_bitlib.c` so that the `bit32` value, `0xfedcba98`, is parsed into its 4 bytes, with the least significant byte output first. The output should be as follows:

   ```
   98
   ba
   dc
   fe
   ```

   This order of the bytes, with the least significant byte first, is called "little endian byte order."

3. The "left-shift" operator, "<<", allows you to move a binary sequence to the more significant bit positions of a "`bit32`" value. For example, the byte `0x12` can be repositioned to the most significant byte position with:

   ```
   bit32 bin_seq;
   bin_seq = 0x12 << 24;
   printhex( bin_seq);
   ```

   the value printed would be "12000000". Recall that 0x12 = 0x00000012.

The "or" operator, "|", is used to combine two binary sequences by setting to 0 only those corresponding bit positions of the two operands that are both 0. It is particularly useful in concatenating the two or more `bit32` values where the non-zero bits are in different positions. For example:

```
bit32 bin_seq;
bin_seq = 0x21 | 0x4300;
printhex(bin_seq);
```

The value printed should be "00004321".

4. In the file "`bitlib.c`" add a C function "`place_byte(bit32 hex_val, bit32 byte_val, int n)`" that returns the "`bit32`" value `hex_val` with `byte_val` inserted at byte position `n`. The previous value of the byte in position `n` of `hex_val` is replaced by the new value, `byte_val`. Compile and debug the updated `bitlib.c` file.

5. Revise your "`main`" program in `test_bitlib.c` so that the `bit32` value, `0x12`, is placed beginning in byte position 3 of a 32 bit bnary sequence, the value `0x56` is placed in byte position 1 of the same sequence. If the original value is `0xabcd0000`, then the output should be as follows:

```
12cd5600
```