CAS CS 210 - Computer Systems

Spring 2017

Problem Set #1 (Logic, Data Representation & Arithmetic)
Out: Friday, February 3
Due: Friday, February 10, 1:00 pm

To be completed individually. For all questions, show your work in obtaining the final answers. Submit a hard copy in the "CS 210" inbox at MCS. Please review the academic conduct rules mentioned in the syllabus and in class.

In answering the first five problems, your code should follow these rules:

- Assume integers are represented in two's complement.
- Assume data type `int` is w bits long. Unless w is given a specific value, your code should work as long as w is a multiple of 8. You can use the expression `sizeof(int)<<3` to compute w.
- Your code should be straight-line with no conditionals, loops, or function calls.
- You are not allowed to use division, modulus, multiplication, or relative comparison operators ($<, >, <=$, and $>=$).
- You are allowed to use all bit-level and logic operations, left and right shifts (but only with shift amounts between 0 and $w-1$), addition and subtraction, and integer constants `INT_MIN` and `INT_MAX`.
- You are allowed to use casting between data types `int` and `unsigned`, either explicitly or implicitly. *Note:* C allows so-called "type casting" to convert between unsigned and signed. This means that you can interpret the same bit pattern according to a magnitude-only convention or two's complement. For example, consider a 3-bit ($w = 3$) representation:

  ```
  int x = -4; // bit pattern 100
  unsigned y;

  y = x; // y is assigned the decimal value 4 based on magnitude-only convention
  ```

  See page 75 in the CMU text for details.
- Unless otherwise specified, you are allowed to use equality (==) and inequality (!=) tests.

Even with these rules, you should try to make your code readable by choosing descriptive variable names and using comments to describe the logic behind your solutions.

You may find it helpful to use a technique called "masking". A *bitmask* is a bit vector that indicates which bits within a variable to change or query. For example, the following code uses the mask `0xFF` to extract the most significant byte from integer argument `x`:

```
/* Get most significant byte from x */
int get_msb(int x) {
    /* Shift by w-8 */
    int shift_val = (sizeof(int)-1)<<3;
    /* Arithmetic shift */
    int xright = x >> shift_val;
    /* Zero all but LSB */
    return xright & 0xFF;
}
```

1. Suppose we number the bytes in a w-bit word from 0 (least significant) to $w/8 - 1$ (most significant). Write code for the following C function, which will return an unsigned value in which byte `i` of argument `x` has been replaced by byte `b`:

   ```
   unsigned replace_byte (unsigned x, int i, unsigned char b);
   ```

   Here are some examples showing how the function should work:

   ```
   replace_byte(0x12345678, 2, 0xAB) --> 0x12AB5678
   replace_byte(0x12345678, 0, 0xAB) --> 0x123456AB
   ```

   You may use constants but no larger than 255 (*i.e.*, `0xFF`).

2. Write C expressions that evaluate to 1 when the following conditions are true, and to 0 when they are false. Assume `x` is of type `int`.

   A. Any bit of `x` equals 1.

   B. Any bit of `x` equals 0.

   C. Any bit in the least significant byte of `x` equals 1.

   D. Any bit in the most significant byte of `x` equals 0.

   In this problem, you may not use equality (==) or inequality (!=) tests. You may use constants but no larger than 255.

3. Write C code to implement the following function:

   ```
   /* Return 1 when any odd bit of x equals 1; 0 otherwise.
       Assume w=32. */

   int any_odd_one(unsigned x);
   ```

   In this problem, you may assume that data type `int` has w=32 bits. You may also use 32-bit constants. Note that bit 0 is the least significant bit, and 0 is even!

4. You just started working for a company that is implementing a set of procedures to operate on a data structure where 4 signed bytes are packed into a 32-bit `unsigned`. Bytes within the word are numbered from 0 (least significant) to 3 (most significant). You have been assigned the task of implementing a function for a machine using two's-complement arithmetic and arithmetic right shifts with the following prototype:

```
/* Declaration of data type where 4 bytes are packed
   into an unsigned */
typedef unsigned packed_t;

/* Extract byte from word.  Return as signed integer */
int xbyte(packed_t word, int bytenum);
```

That is, the function will extract the designated byte and sign extend it to be a 32-bit `int`.

Your predecessor (who was fired for incompetence) wrote the following code:

```
/* Failed attempt at xbyte */
int xbyte(packed_t word, int bytenum)
{
   return (word >> (bytenum << 3)) & 0xFF;
}
```

A. What is wrong with this code?

B. Give a correct implementation of the function that uses only left and right shifts, along with one subtraction. You may use constants but no larger than 255.

5. Write a function with the following prototype:

```
/* Determine whether arguments can be subtracted without overflow */
int tsub_ok(int x, int y);
```

This function should return 1 if the computation `x - y` does not overflow.

6. Convert the following numbers as indicated, using as few digits in the results as necessary.

(a) $(010101)_2$ to base 10

(b) $(011100)_2$ to base 16

(c) $(54.125)_{10}$ to binary

(d) $(122.3)_4$ to base 16

7. Show the results of adding the following pairs of five-bit two's complement numbers and indicate whether or not overflow occurs for each case. (a) 11110 + 11101; (b) 10111 + 10111; (c) 11111 + 01011

8. Assuming a 64-bit machine and two's complement representation for integers, show the total number of bits used, and all the bit values, for the following variables:

```
char c = 35;
char d = 'G';
int x = -42;
```

9. Complete the following code to perform the conversion from ASCII bit representation for the variable `n` to the magnitude-only bit representation for the variable `x`. You may assume that the user enters exactly three digits for the input. (Hint: what are the magnitude-only values for the bit patterns that represent the ASCII numeric symbols?)

```
#include <stdio.h>

main()
{
  char n[4];
  int x;

  printf("Enter a 3-digit non-negative number: ");
  scanf("%s", n);

  ...
  ...
  ...

  printf("The number is %i \n", x);
}
```