CSci 2021 Spring 2020 Section 010 Problem Set 1

Due on paper at the beginning of lecture (3:35pm) on Wednesday, February 12, 2020. We recommend that you type and print out your solutions: we won't be able to give you credit for answers if we can't read your handwriting. Please label your assignment with your name, UMN email address, and the time of your lab section (12:10 pm, 1:25 pm, or 2:30 pm).

Problem 1

In this problem, assume the variables a and b are signed 32-bit integers and that the machine uses two's complement representation, with the same overflow behavior we described in lecture. `TMAX` is the largest signed integer, and `TMIN` is the most negative signed integer. Match each numbered expression on the left with the corresponding expression on the right that is equivalent for all values of a and b. There is exactly one correct match for each expression on the left.

_____	1.	`a | b`	a) `a ^ -(TMIN + TMAX)`
			b) `~(~a & ~b)`
_____	2.	`-a`	c) `(a >> 31) - ((~a) >> 31)`
			d) `(a | b) & (~a | ~b)`
_____	3.	`a * 16`	e) `~TMIN`
			f) `((1 << 31) >> 1)`
_____	4.	`a ^ 1`	g) `a << 4`
			h) `1 << (31 - 1)`
_____	5.	`TMAX`	i) `((a < 0) ? (a + 7) : a) >> 3`
			j) `~a + 1`
_____	6.	`(a >= 0) ? 1 : -1`	k) `((a >> 31) << 1)`
			l) `~(a << 31)`

Problem 2

Adding a number to itself, multiplying by 2, and shifting left by 1 are all the same operation on bits. The way CPUs normally implement this operation, it has an overflow behavior in which the result wraps around when it is too big. Sometimes, though, instead of overflow behavior, it is desirable to have operators with what's called "saturating' ' behavior: for a saturating operation, if the result is too large or small to represent, it is replaced with the largest or smallest value that can be represented.

Your job for this problem is to build a saturating version of this operation of multiplying by 2. You will need to give separate unsigned and signed versions, because even though the non-overflow behavior is the same between unsigned and signed, the definition of overflow and the saturated results will be different. We'll call the two functions `USatTimesTwo` for unsigned and `TSatTimesTwo` for signed.

First, think about what you want using 4-bit integers. Make a table where the first column is all 16 possible 4-bit patterns, and then two more columns show the desired result of `USatTimesTwo` and `TSatTimesTwo` when run on 4-bit numbers with that bit pattern. Specifically, the `USatTimesTwo` should be the correct result of multiplying by two if it can be represented without overflowing to be too large for an unsigned value, and it should be UMax otherwise. The result of `TSatTimesTwo` should be the correct result of multiplying by two if it can be represented without overflowing as a twos-complement value, TMin if it is too negative to be represented, and TMax if it is too large to be represented.

Second, implement the same principles by writing C code for a function `USatTimesTwo` that operates on an `unsigned int` and a function `TSatTimesTwo` that operates on a signed `int`, assuming those types are 32 bits. Note that it won't work well to check for overflow after multiplying by two, because you will have already lost some information in that result. Instead, check the value before multiplying it by 2.

Problem 3

Assume you have a machine with a 16-bit word size and 65536 bytes of memory, so that pointers (memory addresses) take up 16 bits (2 bytes). `short`s are also 2 bytes, as they usually are. Assume that the variables in this code are placed in memory contiguously in the order they are declared, starting at memory address `0x800` and increasing. Given the following code segment:

```
short x = 100;
short *px = &x;
short y = 23;
short *py = &y;
int main(void) {
printf("starting x = %d\n", x);
printf("starting y = %d\n", y);
printf("a) %p\n", px);
printf("b) %p\n", py);
px = py;
printf("c) %d\n", *px);
printf("d) %p\n", &px);
*px = -232 + *px; y = 41 + *py;
printf("e) %d\n", *px);
printf("f) %d\n", *py);
printf("g) %d\n", x);
printf("h) %d\n", y);
}
```

Give the output of each `printf()` assuming the `%p` format specifier prints out a memory address in hexadecimal.

Problem 4

Assume 8-bit two's complement integer representation for this problem. Do all calculations by hand and show your work for full credit.

A. Give the bit sequence for the largest positive number and the most negative number. (I.e., TMax and TMin).

B. Give the base 10 integer equivalent value for TMax and Tmin.

For C-H, compute the value and answer these questions:

- What is the resulting bit sequence?
- What is the corresponding base 10 value?
- Did this computation result in a signed overflow? If yes, explain why.

C. 10110001 + 00110111

D. 01010001 + 00101011

E. 11100000 + 11111001

F. TMin - TMax

G. 0 - TMax

H. TMin / 2

Problem 5

In this problem, show your work for full credit. You can use a calculator for basic decimal arithmetic (addition, subtraction, multiplication, division), but show the operands and result of each such operation.

Convert the following hexadecimal numbers to octal (base 8).

A. $AB32_{16}$

B. $C23E_{16}$

C. $B393_{16}$

D. $243A_{16}$

Convert the following numbers (bases shown with subscripts) to decimal. All numbers are unsigned.

E. $B321_{16}$

F. 87788_{9}

G. 1001100110001001_{2}

Convert the following 10-bit binary numbers to decimal, assuming they are unsigned. Also, convert them to decimal representation, assuming they are signed and use 2's complement representation.

H. 1010101100_{2}

I. 1010100010_{2}

J. 1100011100_{2}