Unsigned Overflow

**Overflow** occurs when two numbers are added or subtracted , and the correct result is a number that is outside of the range of allowable numbers for that precision. Both unsigned and signed overflow are possible (more on signed numbers later)  
  
 8 bits -- unsigned integers 0 to 28 -1 or 0 to 255.

16 bits -- unsigned integers 0 to 216-1 or 0 to 65535

Unsigned Overflow Example

Assume 8 bit precision; i.e. I can’t store any more than 8 bits for each number.

Lets add 255 + 1 = 256. The number 256 is OUTSIDE the range of 0 to 255 What happens during the addition?

255 = FF

+ 1 = 01  
-------------------  
 256 != 00  
  
F + 1 = 0, carry out  
F + 1 (carry) + 0 = 0, carry out  
Carry out of MSB falls off end, No place to put it.

Final answer is WRONG because could not store carry out.

**Unsigned Overflow**

A carry out of the Most Significant Digit (MSD) or Most Significant Bit (MSB) is an OVERFLOW indicator for addition of UNSIGNED numbers.

The correct result has overflowed the number range for that precision, and thus the result is incorrect.

If we could STORE the carry out of the MSD, then the answer would be correct.

But we are assuming it is discarded because of fixed precision, so the bits we have left are the incorrect answer.

**Signed Integer Representation**

We have been ignoring large sets of number types so far; i.e. the sets of signed integers, and floating point numbers.

We will not talk about floating point representation (i.e. 9.23 x1013).

We WILL talk about signed integer representation.

The *PROBLEM* with signed integers ( - 45, + 27, -99) is the SIGN. How do we encode the sign?

The sign is an extra piece of information that has to be encoded in addition to the magnitude.

**Signed Magnitude Representation**

Signed Magnitude (SM) is a method for encoding signed integers.

The Most Significant Bit is used to represent the sign. ‘1’ is used for a ‘-’ (negative sign), a ‘0’ for a ‘+’ (positive sign).

The format of a SM number in 8 bits is:  
 s m m m m m m m

where ‘s’ is the sign bit and the other 7 bits represent the magnitude.

NOTE: for positive numbers, the result is the same as the unsigned binary representation.

**Signed Magnitude Examples (8 bits)**

-5 = 1 0000101 = 85h  
+5 = 0 0000101 = 05h  
+127 = 0 1111111 = 7Fh  
 -127 = 1 1111111 = FFh  
+ 0 = 0 0000000 = 00h  
 - 0 = 1 0000000 = 80h

For 8 bits, can represent the signed integers -127 to +127.

For N bits, can represent the signed integers

-(2(N-1) – 1) to +( 2(N-1) – 1)

**Signed Magnitude comments**

* Signed magnitude easy to understand and encode (similar to that which people commonly use).
* **One problem is that it has two ways of representing 0 (-0, and +0). Mathematically speaking, no such thing as two representations for zeros.**
* Another problem is that addition of K + (-K) does not give Zero (if the numbers are operated on directly with the sign bit in place).  
   -5 + 5 = 85h + 05h = 8Ah = -10   
  In general to add two SM numbers it is necessary to subtract the smaller magnitude from the larger and use the sign of the larger number for the sign of the result. (Special case of equal magnitudes and opposite sign must be recognized).
* Example: (+5) + (-6) = -(6 – 5) = - 1

**One’s Complement Representation**

One’s complement is another way to represent signed integers.  
  
To encode a negative number, get the binary representation of its magnitude, then COMPLEMENT each bit. Complementing each bit means that 1’s are replaced with 0’s, 0’s are replaced with 1’s.

What is -5 in One’s Complement, 8 bits?

The magnitude 5 in 8-bits is 0000 0101 = 05h

Now complement each bit: 1111 1010 = FAh  
FAh is the 8-bit, one’s complement number of -5.  
  
NOTE: positive numbers in 1’s complement are simply their binary representation.

One’s Complement Examples

-5 = 1111 1010 = FAh  
+5 = 0000 0101 = 05h  
+127 = 0111 1111 = 7Fh  
 -127 = 1000 0000 = 80h  
+ 0 = 0000 0000 = 00h  
 - 0 = 1111 1111 = FFh

For 8 bits, can represent the signed integers -127 to +127.

For N bits, can represent the signed integers

-(2(N-1) – 1) to + (2(N-1) – 1)

**One’s Complement Comments**

**Still have the problem that there are two ways of representing 0 (-0, and +0) . Mathematically speaking, no such thing as two representations for zeros.**  
However, addition of K + (-K) now gives Zero!  
  
 -5 + 5 = FA + 05 = FF = -0   
  
Unfortunately, K + 0 = K only works if we use +0, does not work if we use -0.

5 + (+0) = 05h + 00h = 05 = 5 (ok)

5 + (-0) = 05h + FFh = 04 = 4 (wrong)

**Two’s Complement Representation**

Two’s complement is another way to represent signed integers.  
  
To encode a negative number, get the binary representation of its magnitude, COMPLEMENT each bit, then ADD 1. (get One’s complement, then add 1).

What is -5 in Two’s Complement, 8 bits?

The magnitude 5 in 8-bits is 0000 0101 = 05h

Now complement each bit: 1111 1010 = FAh  
Now add one: FAh + 1h = FBh  
$FB is the 8-bit, twos complement representation of -5.  
  
NOTE: positive numbers in 2’s complement are simply their binary representation.

**Two’s Complement Examples**

-5 = 1111 1011 = FBh  
+5 = 0000 0101 = 05h  
+127 = 0111 1111 = 7Fh  
 -127 = 1000 0001 = 81h  
-128 = 1000 0000 = 80h (note the extended range)  
+ 0 = 0000 0000 = 00h  
 - 0 = 0000 0000 = 00h (only one zero)

For 8 bits, can represent the signed integers -128 to +127.

For N bits, can represent the signed integers

-2(N-1) to + 2(N-1) - 1

Note that negative range extends one more than positive range.

Two’s Complement Comments

Two’s complement is the method of choice for representing signed integers.

It has none of the drawbacks of Signed Magnitude or One’s Complement.  
  
There is only one zero, and K + (-K) = 0.  
  
 -5 + 5 = FB + 05 = 00 = 0

Normal binary addition is used for adding numbers that represent twos complement integers.

Given a hex or decimal number,

how do we know if it is in 2’s complement or 1’s complement; is it already in 2’s complement or do we have put it in 2’s complement?

If a HEX number is given, it may be asked for a decimal representation based on how it is INTERPRETED the encoding as a particular method (i.e, either 2’s complement, 1’s complement, signed magnitude).

A Hex or binary number BY ITSELF can represent ANYTHING (unsigned number, signed number, character code, etc.).

You MUST HAVE additional information that tells you what the encoding of the bits mean.

FE = 1111 1110 as an 8 bit unsigned integer = 254   
FE as an 8 bit signed magnitude integer = -126   
FE as an 8 bit ones complement integer = - 1  
FE as an 8 bit twos complement integer = -2  
  
7F = 0111 1111 as an 8 bit unsigned integer = 127  
7F as an 8 bit signed magnitude integer = +127  
7F as an 8 bit ones complement integer = +127  
7F as an 8 bit twos complement integer = +127  
  
To do hex to signed-decimal conversion, we need to determine sign (Step 1), determine Magnitude (step 2), combine sign and magnitude (Step 3) … more on this on following slides.

Hex to Signed Decimal Conversion Rules

Given a Hex number, and you are told to convert to a signed integer (either as signed magnitude, 1’s complement, 2’s complement)

STEP 1: Determine the sign. If the Most Significant Bit is zero, the sign is positive. If the MSB is one, the sign is negative.

This is true for ALL THREE representations: SM, 1’s complement, 2’s complement.  
  
 F0 = 1111 0000 (MSB is ‘1’), so sign of result is ‘-’  
 64 = 0110 0100 (MSB is ‘0’), so sign of result is ‘+’.

If the Most Significant Hex Digit is > 7, then MSB = ‘1’  
(i.e. 8,9,A,B,C,D,E,F => MSB = ‘1’)

STEP 2 (positive sign): If the sign is POSITIVE, then just convert the hex value to decimal. The representation is the same for SM, 1’s complement, 2’s complement.  
 64 is a positive number, decimal value is  
 6 x 16 + 4 = 100.

Final answer is +100 regardless of whether encoding was SM, 1’s complement, or 2’s complement.  
  
 64 as an 8 bit signed magnitude integer = +100  
 64 as an 8 bit ones complement integer = +100  
 64 as an 8 bit twos complement integer = +100

STEP 2 (negative sign): If the sign is Negative, then need to compute the magnitude of the number.

We will use the trick that - (-N) = + N   
i.e. Take the negative of a negative number will give you the positive number. In this case the number will be the magnitude.  
  
If the number is SM format, set Sign bit to Zero:  
 F0 = 1111 0000 => 0111 0000 = 70h = 112  
If the number is 1’s complement, complement each bit.  
 F0 = 1111 0000 => 0000 1111 = 0Fh = 15  
If the number is 2’s complement, perform 2’s complement on it.  
 F0 = 11110000 => 00001111 + 1 = 0001 0000 = 10h = 16

(Note: found magnitudes only so far)

STEP 3 : Just combine the sign and magnitude to get the result.

F0 as 8 bit Signed magnitude number is -112  
 F0 as 8 bit ones complement number is -15   
 F0 as 8 bit twos complement number is -16  
  
 64 as an 8 bit signed magnitude integer = +100  
 64 as an 8 bit ones complement integer = +100  
 64 as an 8 bit twos complement integer = +100

**Two’s Complement Overflow**

Consider two 8-bit 2’s complement numbers. I can represent the signed integers -128 to +127 using this representation.  
  
What if I do (+1) + (+127) = +128. The number +128 is OUT of the RANGE that I can represent with 8 bits. What happens when I do the binary addition?

+127 = 7F

+ +1 = 01  
-------------------  
 128 != 80 (this is actually -128 as a twos complement number - the wrong answer)  
  
How do we know if overflow occurred?

Added two POSITIVE numbers, and got a NEGATIVE result.

Detecting Two’s Complement Overflow

Two’s complement overflow occurs when:  
  
 Add two POSITIVE numbers and get a NEGATIVE result  
 Add two NEGATIVE numbers and get a POSITIVE result  
  
We CANNOT get two’s complement overflow

if we add a NEGATIVE and a POSITIVE number together.

**Another way an overflow condition can be detected is by observing**

**the carry *into* the sign-bit position and the carry *out* of the sign-bit position.**

**If these two carries are *not equal* an overflow has occurred.**

All hex numbers represent signed decimal in two’s complement format

FFh = -1

+ 01h = + 1  
--------  
 00h = 0

Note there is a carry out, but the answer is correct. Can’t have 2’s complement overflow when adding positive and negative number.

FFh = -1

+ 80h = -128  
--------  
 7Fh = +127 (incorrect)

Added two negative numbers, got a positive number. Two’s Complement overflow.