===
Assignment #04 - Number Systems with Negative Numbers
===

- Ian! D. Allen - idallen@idallen.ca - www.idallen.com

Sources for these answers (thank you!):
    Ian Allen
    Jason Lavergne
Corrections by (thank you!):
     Terence Christie (#19)

1. Using a word size of four bits, list in order vertically all of the possible bit patterns from 0000(2) to 1111(2). Beside each of the 16 patterns, give:

    a) the hexadecimal and unsigned decimal equivalent
    b) the four-bit signed-magnitude decimal equivalent
    c) the four-bit one's complement decimal equivalent
    d) the four-bit two's complement decimal equivalent

BINARY	HEX	DEC	S/M	1's	2's
0000	0	0	0	0	0
0001	1	1	1	1	1
0010	2	2	2	2	2
0011	3	3	3	3	3
0100	4	4	4	4	4
0101	5	5	5	5	5
0110	6	6	6	6	6
0111	7	7	7	7	7
1000	8	8	-0	-7	-8
1001	9	9	-1	-6	-7
1010	A	10	-2	-5	-6
1011	B	11	-3	-4	-5
1100	C	12	-4	-3	-4
1101	D	13	-5	-2	-3
1110	E	14	-6	-1	-2
1111	F	15	-7	-0	-1

2. What are the largest and smallest integers (in decimal) an 8-bit word can hold using a sign-magnitude representation?

  Half the of the 2**8 bit patterns are negative; half are positive.
  2**8 divided by two is 2**7. We start from zero, therefore:

```
    + (2**7 - 1) = +127  (+0 -> +127)
    - (2**7 - 1) = -127  (-0 -> -127)
```

3. What are the largest and smallest integers (in decimal) an 8-bit word can hold using a one's complement representation?

  Half the of the 2**8 bit patterns are negative; half are positive.
  2**8 divided by two is 2**7. We start from zero, therefore:

```
    + (2**7 - 1) = +127  (+0 -> +127)
    - (2**7 - 1) = -127  (-0 -> -127)
```

4. What are the largest and smallest integers (in decimal) an 8-bit word can hold using a two's complement representation?

    Due to the "flip bits and add one", there is no "minus zero".

```
    + (2**7 - 1) = +127  (+0 -> +127)
    - (2**7)     = -128  (-1 -> -128 -- note how we start at -1 not -0)
```

```
 5.  What are the largest and smallest integers a 16-bit word can hold
     using a sign-magnitude representation?

   Half the of the 2**16 bit patterns are negative; half are positive.
   2**16 divided by two is 2**15.  We start from zero, therefore:

     + (2**15 - 1) = +32,767  (+0 -> +32,767)
     - (2**15 - 1) = -32,767  (-0 -> -32,767)

 6.  What are the largest and smallest integers a 16-bit word can hold
     using a one's complement representation?

   Half the of the 2**16 bit patterns are negative; half are positive.
   2**16 divided by two is 2**15.  We start from zero, therefore:

     + (2**15 - 1) = +32,767  (+0 -> +32,767)
     - (2**15 - 1) = -32,767  (-0 -> -32,767)

 7.  What are the largest and smallest integers a 16-bit word can hold
     using a two's complement representation?

     Due to the "flip bits and add one", there is no "minus zero".

     + (2**15 - 1) = +32,767  (+0 -> +32,767)
     - (2**15)     = -32,768  (-1 -> -32,768 -- note how we start at -1 not -0)

 8.  Generalize the above answers: For an N-bit word size, express as a
     formula using N as a power of two (i.e. using 2**N) what are the largest
     (most positive) and smallest (least positive) integers possible for:

     a) N-bit unsigned
     b) N-bit signed-magnitude
     c) N-bit one's complement
     d) N-bit two's complement

     a) N-bit unsigned

         Largest  = (2**N) - 1
         Smallest = 0

     b) N-bit signed-magnitude

         Largest  = + ( (2**(N-1)) - 1 )
         Smallest = - ( (2**(N-1)) - 1 )

     c) N-bit one's complement

         Largest  = + ( (2**(N-1)) - 1 )
         Smallest = - ( (2**(N-1)) - 1 )

     d) N-bit two's complement

         Largest  = + ( (2**(N-1)) - 1 )
         Smallest = - ( (2**(N-1))     )

 9.  Convert 23 to 8-bit 00010111 binary one's complement.

     Number is positive.  Just treat as unsigned binary:

       (+)23
         -16 = 1 x 2**4
         ---
           7
         - 0 = 0 x 2**3
         ---
```

```
          7
        - 4 = 1 x 2**2
        ---
          3
        - 2 = 1 x 2**1
        ---
          1
        - 1 = 1 x 2**0
        ---
          0

        = 0001 0111 [one's complement]

10. Convert -9 to 8-bit 11110110 binary one's complement.

    Number is negative.  Do the positive, then negate it:

       (+)9
        - 8 = 1 x 2**3
        ---
          1
        - 0 = 0 x 2**2
        ---
          1
        - 0 = 0 x 2**1
        ---
          1
        - 1 = 1 x 2**0
        ---
          0

        = 0000 1001 [unsigned 9]
        = 1111 0110 [flip bits to get one's complement -9]

11. Convert -23 to 8-bit 11101000 binary one's complement.

    Number is negative.  Do the positive, then negate it:

    = 0001 0111 [unsigned 23 from above]
    = 1110 1000 [flip bits to get one's complement]

12. How do you know that a two's-complement addition has overflowed?
    (There are at least two ways.  Give one or both.)

    Method A: positive+positive=negative or negative+negative=positive
      -OR-
    Method B: carry-in to sign bit is not equal to carry-out from sign bit

13. Convert 23 to 8-bit 00010111 binary two's complement.

    Number is positive.   Just treat as unsigned binary (above):

     = 0001 0111 [unsigned 23 from above]

14. Convert -9 to 8-bit 11110111 binary two's complement.

    Number is negative.  Treat as positive, then do one's complement
    (above), then add one:

     = 0000 1001 [unsigned 9 from above]
     = 1111 0110 [flip bits for one's complement]
     = 1111 0111 [add one for two's complement -9]

15. Convert -23 to 8-bit 11101001 binary two's complement.
```

```
    Number is negative.  Treat as positive, do one's complement (above),
    then add one:

     = 0001 0111 [unsigned 23 from above]
     = 1110 1000 [flip bits for one's complement]
     = 1110 1001 [add one for two's complement -23]

16. Convert 8-bit 10010011 binary sign-magnitude to -19 decimal (note
    the negative).

    Number is negative.  Make it positive, convert to decimal, put a
    minus sign in front of it:

      10010011 [sign-magnitude]
      00010011 [remove sign bit to make it unsigned]

        = 1 x 2**0 +
          1 x 2**1 +
          0 x 2**2 +
          0 x 2**3 +
          1 x 2**4 +
          0 x 2**5 +
          0 x 2**6    = 19 --> (add minus sign) -19

17. Convert 8-bit 10010011 binary one's complement to -108 decimal (note
    the negative).

    Number is negative.  Make it positive, convert to decimal, put a
    minus sign in front of it:

      10010011 [one's complement]
      01101100 [flip bits to make it unsigned]

        = 0 x 2**0 +
          0 x 2**1 +
          1 x 2**2 +
          1 x 2**3 +
          0 x 2**4 +
          1 x 2**5 +
          1 x 2**6    = 108 --> (add minus sign) -108

18. Convert 8-bit 10010011 binary two's complement to -109 decimal (note
    the negative).

    Number is negative.  Make it positive, convert to decimal, put a
    minus sign in front of it:

      10010011 [two's complement]
      01101100 [flip bits and then ...]
      01101101 [... add one - number is now positive]

        = 1 x 2**0 +
          0 x 2**1 +
          1 x 2**2 +
          1 x 2**3 +
          0 x 2**4 +
          1 x 2**5 +
          1 x 2**6    = 109 --> (add minus sign) -109

19. Copy the left column of ECOA2e Chapter 2 slide 54.  Fix the second sum
    to be 0100+0110.  Perform the given two's complement additions.
    Without looking, fill in the remaining four columns based on the
    results.  Add another column that states whether the result is
    correct if treated as unsigned math instead of two's complement.
```

```
                RESULT  CAR   OVF   SIGNOK  UNSIGNOK
    0100 + 0010 = 0110    No    No    Yes       Yes
    0100 + 0110 = 1010    No   Yes     No       Yes
    1100 + 1110 = 1010   Yes    No    Yes        No
    1100 + 1010 = 0110   Yes   Yes     No        No

    (Did you remember to fix the typing error in the second line?)

20. Convert 16-bit two's complement 8000h to decimal -32,768 (note the
    negative).

    Number is negative.  Make it positive, convert to decimal, put a
    minus sign in front of it:

    8000h  [two's complement]
    7FFFh  [flip bits, using hex bit flip table, and then ...]
    8000h  [... add one - number is now positive]

    Add up one power of 16 (using some exponent math to simplify):

    8 * 16**3 = 2**3 * (2**4)**3 = 2**3 * 2**12 = 2**15 = 32,768 (famous number)

    Add back the minus sign: -32,768

21. Convert 16-bit two's complement A123h to decimal -24,285 (note negative).

    Number is negative.  Make it positive, convert to decimal, put a
    minus sign in front of it:

    A123h  [two's complement]
    5EDCh  [flip bits, using hex bit flip table, and then ...]
    5EDDh  [... add one - number is now positive]

    Add up four powers of 16, from 16**0 up to 16**3:

    5 * 16**3  +   E * 16**2  +   D * 16**1  +   D * 16**0
  = 5 *  4096  +  14 *   256  +  13 *    16  +  13 *     1 = 24,285

    Add back the minus sign: -24,285

22. Convert 16-bit two's complement FFFFh to decimal -1 (note the negative).

    Number is negative.  Make it positive, convert to decimal, put a
    minus sign in front of it (use the method above):

    FFFFh -> [flip bits] 0000h -> [add one] 0001h = 1 -> [add sign] -1

23. Cross out or delete the positive numbers (16-bit two's complement),
    leaving only negative numbers:    8000h  8001h  9FC5h  A123h  BFFFh

    In 16-bits, all the above numbers have the sign bit (top bit) on.
    They are all negative numbers.

24. Add 16-bit two's complement ABCDh to 7FFFh and give the Result, Carry,
    Overflow, and Sign.  Is the result correct for two's complement?

    ABCDh + 7FFFh = 2BCCh [carry on] [overflow off] [sign off] [correct]

25. Add 16-bit two's complement 8A9Ch to ABCDh and give the Result, Carry,
    Overflow, and Sign.  Is the result correct for two's complement?

    8A9Ch + ABCDh = 3669h [carry on] [overflow on] [sign off] [wrong answer]

26. Add 18-bit two's complement 8A9Ch to ABCDh and give the Result, Carry,
    Overflow, and Sign.  Is the result correct?  (The 18 is not a mistake.)
    In 18-bits, all the above numbers have the sign bit (top bit) *off*.
    ( 8A9Ch written in 18 bits binary is:  00 1000 1010 1001 1100 )
    All the numbers have zero in the sign bit; they are all positive numbers.

    8A9Ch + ABCDh = 13669h [carry off] [overflow off] [sign off] [correct]

    Note that 18-bit 13669h = 01 0011 0110 0110 1001 --> a positive number
    The "carry" now fits in 18 bits - the carry flag does not come on.
    The sign bit (the leftmost bit) is off - the number is positive.

27. Add 16-bit two's complement 9999h to 4321h and give the Result, Carry,
    Overflow, and Sign.  Is the result correct?

    9999h + 4321h = DCBAh [carry off] [overflow off] [sign on] [correct]

28. Is it still true that shifting a binary number left one bit multiplies
    it by two, if the number is a negative two's complement number?

    Yes, iff the doubled value can be represented in the number of bits.
    Consider 4-bit two's complement:

    1111(2) = -1(10)   shifted left becomes   1110 = -2(10)
    1110(2) = -2(10)   shifted left becomes   1100 = -4(10)
    1101(2) = -3(10)   shifted left becomes   1010 = -6(10)
    1100(2) = -4(10)   shifted left becomes   1000 = -8(10)
    1011(2) = -5(10)   shifted left becomes   0110 = +6(10) [too big]

    The left shift doubles the number only if the doubled number is
    small enough to be represented in the available number of bits.

29. Cross out or delete the following positive values that will fit
    correctly in a 32bit two's complement integer with no misrepresentation
    (leave behind only the values that *cannot* be correctly converted):

    The range of 32-bit two's complement integers is (from the formula above):

    Largest  = + ( (2**(N-1)) - 1 ) = + (2**31 - 1) [ = +(2 Gi) - 1 ]
    Smallest = - ( (2**(N-1))     ) = - (2**31)     [ = -2 Gi ]

    For 32 bit two's comp. integers, anything bigger than (2**31 - 1) won't fit:

                  2**31     (2**31)+1
    (2**32)-1     2**32     (2**32)+1
    (2**33)-1     2**33     (2**33)+1

30. Build a Truth Table for two's complement "overflow", based on the rule
    that overflow occurs if the carry in to the sign bit does not equal the
    carry out.  Use these headings on the truth table columns:

      ------------ -------------
       A   B   CI   R   CO   OV
      ------------ -------------

    Columns A, B, and CI are inputs.  R, CO, and OV are outputs.
    Columns A and B are the sign bits of the two numbers being added.
    Column CI is the possible carry in to the sign bit.  CO (carry out),
    R (result), and OV (overflow flag) are outputs.  The truth table must
    have eight rows, for all possible combinations of the three inputs A,
    B, and CI.  R is the sign bit of the result (based on A, B, and CI).
    CO is the carry out (also based on A, B, and CI).  OV is whether
    or not overflow occurred (based on CI and CO).  Fill in the whole
    8x6 table.

    The table starts with every possible combination of the three input
```

```
     bits on the left, from 0 0 0 to 1 1 1.  The three outputs are:

       ------------ -------------
        A   B   CI   R   CO   OV
       ------------ -------------

        0   0    0   0    0   No
        0   0    1   1    0   Yes (because CI != CO)
        0   1    0   1    0   No
        0   1    1   0    1   No
        1   0    0   1    0   No
        1   0    1   0    1   No
        1   1    0   0    1   Yes (because CI != CO)
        1   1    1   1    1   No

--
| Ian! D. Allen  -  idallen@idallen.ca  -  Ottawa, Ontario, Canada
| Home Page: http://idallen.com/   Contact Improv: http://contactimprov.ca/
| College professor (Free/Libre GNU+Linux) at: http://teaching.idallen.com/
| Defend digital freedom:  http://eff.org/  and have fun:  http://fools.ca/
```