```
############################################################
#           cs315 Week 3
#
#   -> Signed number system (sign-magnitude, 1's complement, 2's complement)
#
############################################################

Signed number systems:
    * can represent both positive and negative numbers (unsigned binary is positive only)
    * left-most bit is the sign

In the binary systems we will be using:
    * 0 -> positive (+)
    * 1 -> negative (-)

IMPORTANT: Binary representation for positive numbers is THE SAME for sign-magnitude, 1's complement, and 2's complement systems. Thus, if the value is positive (i.e. >= 0), COPY BITS ONLY, no conversion is needed.

sign-magnitude:
    sign-magnitude is made out of sign and magnitude
    |-- sign (1 bit) ---|--- magnitude (varies) --|

    --> decimal to sign-magnitude conversion <--
    convert decimal magnitude to binary
    set sign bit accordingly (i.e. 0 for positive, 1 for negative)

    Ex: -27 to sign-magnitude binary
    convert magnitude: 27 (in base 10) is equal to 11011 (in base 2)
    set sign: negative --> 1

    result: 111011


    --> sign-magnitude to decimal conversion <--
    convert magnitude to decimal
    set sign accordingly

    Ex: 01101
        ^
    sign bit

    magnitude: 1101 (in base 2) is equal to 13 (in base 10)
    sign: 0 --> positive
    result: +13


1's complement:
    * used primarily as a step for converting to 2's complement

    --> conversion from sign-magnitude to 1's complement <--
    * if value is negative:
        --> flip all bits EXCEPT the sign bit
    * if value is positive:
        --> copy all the bits

    Ex: 111011 in sign-magnitude to 1's complement

    sign is a 1 (number is negative), must convert
    111011 (in sign-magnitude) is equal to 100100 (in 1's complement)
                                           ^
                                    note that sign of result is still 1 (number was originally negative and it is still negative. It verifies correctness of conversion)

    Ex: 01101 in sign-magnitude to 1's complement

    sign is a 0 (value is positive), copy only
    01101 (in sign-magnitude) is equal to 01101 (in 1's complement)
```

```
    --> conversion from 1's complement to sign-magnitude <--
    The same process
    * if value is negative:
        --> flip all bits EXCEPT the sign bit
    * if value is positive:
        --> copy all the bits

    Ex: 11110 in 1's complement to sign magnitude

    sign is a 1 (value is negative), must convert
    11110 (in 1's complement) is equal to 10001 (in sign-magnitude)


    Ex: 01111 in 1's complement to sign magnitude

    sign is a 0 (value is positive), copy only
    01111 (in 1's complement) is equal to 01111 (in sign-magnitude)


2's complement:
    * very widely used in computer hardware

    --> conversion from 1's complement to 2's complement <--
    * if value is positive, COPY ONLY
    * if value is negative, add 1 to the 1's complement value

    Ex: 100111 in 1's complement to 2's complement
    value is negative, convert: 100111 (1's complement) + 1 = 101000 (2's complement)

                               100111
                          +         1
                               ------
                               101000

    Ex: 011001 in 1's complement to 2's complement
    value is positive, copy only: 011001 (1's complement) = 011001 (2's complement)


    --> conversion from 2's complement to unsigned binary <--
    * if value is positive, COPY ONLY
    * if value is negative, flip all the numbers and then add 1 to it

    Ex: 100001 in 2's complement to unsigned binary
    value is negative, convert: 100001
                                011110
                           +         1
                                ------
                                011111  <-- unsigned binary (positive)


    Ex: 011101 in 2's complement to unsigned binary
    value is positive, copy only: 011101 (2's complement) = 011101 (unsigned binary)


concepts:
    sign extension:
        * it may be necessary to extend the number of bits in a binary number
        * extending number of bits should not change value

    sign-magnitude sign extension:
        pad 0's between the sign and the value as needed

    Ex: sign extend 11011 (4 bits of magnitude) to 7 bits of magnitude
    11011 --> 10001011

    1's complement and 2's complement sign extension:
        * copy sign bit to the left as needed
```

```
    Ex: sign extend 11011 (4 bits of magnitude) to 7 bits of magnitude
    11011 --> 11111011


signed overflow (2's complement arithmetic):
    * NOT the same as unsigned overflow
    * adding two values of the same sign may yield a result with a different sign bit (signed overflow)

    * determining whether overflow occurred or not:
        if the operand signs are different, there will be no overflow
        if the operand signs are the same, there MAY be overflow
        if result sign is the same sign as the operands, no overflow
        if the result sign is different from the operands, overflow occurred

    Ex: 00000101 + 01000000 = 01000101
        ^          ^
    positive    positive

    same sign operands
    result sign is the same as the operands
    no overflow


    Ex: 10000010 + 11111101 = 01111111
        ^          ^
    negative    negative

    same sign operands
    result sign is different from operands
    overflow occurred


    Ex: 10000110 + 00100101 = 10101011
        ^          ^
    negative    positive

    different sign operands
    no overflow (overflow cannot occurs because numbers have different sign. In other words, magnitude is getting small as we "subtract")

Note: on homework, questions will ask to state whether overflow occurred or not. State what happened either way (i.e. do not say so only when overflow occurs)

Summary:

    * decimal to sign-magnitude:
        convert decimal magnitude to binary
        set sign bit accordingly (i.e. 0 for positive, 1 for negative)

    * sign-magnitude to decimal:
        convert magnitude to decimal
        set sign accordingly

    * sign-magnitude to 1's complement:
        flip all bits EXCEPT the sign bit

    * 1's complement to sign-magnitude:
        flip all bits EXCEPT the sign bit

    * 1's complement to 2's complement:
        add 1 to the 1's complement value

    * 2's complement to unsigned binary:
        flip all the numbers and then add 1 to the result
```