Representing / Encoding Integers

- Encoding = symbolic representation of a value, in some specified number of digits, in some specified alphabet (here, 0 & 1)
- *Unsigned binary* representation
  - Represent only positive numbers
    - Direct binary representation
  - Examples (8-bit unsigned binary)
    $+13_{10} = +1101_2 = 00001101_{2ub}$
  - Can represent 0 to 255 in 8 bits
- *Signed magnitude* representation
  - Precede number with sign bit
    0 = positive, 1 = negative
  - Examples (8-bit signed magnitude)
    $+13_{10} = +1101_2 = 00001101_{2sm}$
    $-13_{10} = -1101_2 = 10001101_{2sm}$
  - Can represent –127 to +127 in 8 bits

Encoding Integers (cont.)

- *Excess n* representation
  - Add *bias value* (*n*) to number, then represent directly in binary
  - Examples (8-bit excess 128)
    $+13_{10} = 13 + 128 = 141 = 10001101_{2ex128}$
    $-13_{10} = -13 + 128 = 115 = 01110011_{2ex128}$
  - Can represent –128 to 127 in 8 bits
- *Two's complement* representation
  - Represent positive numbers in n-bit signed magnitude form
  - Represent negative numbers as $2^n - N$
  - Examples (8-bit two's complement)
    $+13_{10} = +0001101_2 = 00001101_{2'scomp}$
    $-13_{10} = 256 - 13 = 243 = 11110011_{2'scomp}$
  - Can represent –128 to 127 in 8 bits

8-Bit Numerical Representations

Numeric Value	Unsigned Binary	Signed Magnitude	Excess 128	Excess 127	Two's Complemen
255	11111111				
254	11111110	N/A	N/A	N/A	N/A
…					
128	10000000			11111111	
127	01111111	01111111	11111111	11111110	01111111
126	01111110	01111110	11111110	11111101	01111110
…	…	…	…	…	…
2	00000010	00000010	10000010	10000001	00000010
1	00000001	00000001	10000001	10000000	00000001
0	00000000	00000000	10000000	01111111	00000000
-1		10000001	01111111	01111110	11111111
-2		10000010	01111110	01111101	11111110
…	N/A	…	…	…	…
-127		11111111	00000001	00000000	10000001
-128		N/A	00000000	N/A	10000000

Worksheet — Two's Complement Addition and Subtraction

- Perform the following arithmetic operations in 4-bit two's complement arithmetic, showing your work:
  2 + 3 =
  2 – 4 =
  (–7) – (–5) =
  (–5) + (–2) =
- To add two numbers, add as usual
- To subtract two numbers, negate the second, and then add
  - Two ways to negate a two's complement number:
    - Invert all the bits, and add 1 to the result
    - Scan right to left, keep all bit the same, but invert all bits after passing the first "1"
    - In 8 bits: –13 = –00001101 = 11110011
  - Why subtract this way??

Fixed-Point Representation

- Suppose we have a number of values to represent. One way to do so would be use a specific number of digits, assuming a decimal point to the right of the lsd:

  000500. 006000. 010000.

- Or we could assume that the decimal point is further to the right of the lsd:

  000005 _ _. 000060 _ _. 000100 _ _.

- We could also assume that the decimal point is between the msd and lsd:

  00500.0 06000.0 10000.0

- All of these are called *fixed-point* representations — ones where the decimal point is <u>fixed</u> at one specific place for encoding all numbers

Floating-Point Representation

- Consider the following ways of representing the decimal values:

5	60	100	(x100)
500	6,000	10,000	(x1)
5000	60,000	100,000	(x0.1)

- Internally, we can represent a value as:

  number, n multiplied by *scale factor*, s

000005	000060	000100	$s = 100$
000500	006000	010000	$s = 1$
005000	060000	100000	$s = 0.1$

- Encoding a value using a number (*mantissa*) and an scale factor (*characteristic*, or *exponent*, to an implicit base) is called a *floating-point* representation

Floating-Point Representation

- In general, this is all very similar to *scientific notation* — writing a number as as the product of a decimal number and a power of 10
  - 5×10^2 60×10^2 100×10^2
  - 500×10^0 6000×10^0 100000×10^0
- A number can be represented uniquely in *normal form* — where we insist that there be exactly one non-zero digit to the left of the decimal point
  - 5×10^2 6×10^3 1×10^4
- All significant digits must be included. Thus the following three values are different:
  - 5×10^2 5.0×10^2 5.000×10^2

Homework #5 — Due 11/9/98 (Part 1)

1. Perform the following arithmetic operations in 5-bit two's complement arithmetic, showing your work:

   6 + 7 =

   6 – 7 =

   6 + (–4) =

   (–2) – (–5) =