4 Integer arithmetic

(4.1) In C,

- Integer data is stored in blocks of
  - 2 bytes, 'short' or 'short int'
  - 4 bytes, 'int',
  - 4 bytes, 'long' or 'long int' on a 32-bit machine,
  - 8 bytes, 'long' or 'long int' on a 64-bit machine.
- Short integers are never used except where memory is scarce.
- At *face value,* these integers have ranges

  $$0 \ldots 2^{16} - 1, \quad 0 \ldots 2^{32} - 1, \quad 0 \ldots 2^{64} - 1,$$

  respectively. But *negative numbers* must be allowed for.
- Where the *high-order bit* is 1, a *negative number* is represented.

Negative integers. A short integer with high-order bit 1 represents the negative integer

$$v - 2^{16}$$

where v is its face value (from 32768 to 65536). Where the high-order bit is 0, it represents the number given by its face value.

Conversely, a negative number s is represented as a short integer as

$$s + 2^{16}.$$

This representation is called *2s complement*.

Thus if the face value is 32000 then this represents a positive number, but 33000 represents $33000 - 2^{32} = -32536$.

Short integers have values in the range

$$-2^{15} \ldots 2^{15} - 1, \, - 32768 \ldots 32767, \, - (1000000000000000)_2 \ldots (-80)_{16} \ldots (ff)_{16}.$$

This is called *short integer range.*

In general, an N-bit integer has range

$$-2^{N-1} \ldots 2^{N-1} - 1$$

The range of these kinds of integer are roughly

- 16 bit short int: $\pm 32,000$.
- 32 bit int; $\pm$ 2 billion.

- 64 bit long int: $\pm 9,223,372,036,854,775,808$.
  $\pm 9 \times 10^{18}$

respectively.

Converting a number s, in short integer range, to a short int.

- If s is nonnegative, just convert it to 4 hex digits. Otherwise,
- Suppose $s = -t$.
- Convert t to 4 hexadecimal digits.
- Subtract from $(ffff)_{16}$.
- Add 1.

Example. Convert 3276 to short int.

```
      204          12
16 ) 3276     16 ) 204
     32           16
      07           44
       0           32
       76     rmndr 12
       64
rmndr  12                answer 0ccc
```

Little endian. The maths machines are all Dell computers using Intel processors, which store numerical data 'little endian.' That is, the *bytes* are stored low-order byte first, but within the byte face value is observed. The integer 3276 is stored as $cc\ 0c$.

Example. Convert -2768 to short int.

```
      173           10              ffff
16 ) 2768      16 ) 173           - 0ad0
     16            16               f52f
     116            13            +    1
     112             0        answer f530
       48      rmndr 13
       48
rmndr   0         2768 = 0a d0
```

The answer is $f5$ 30, or 30 $f5$ little-endian.

Addition of ints. The computer effectively adds N-bit integers modulo 2^N, where $N = 16, 32, 64$ for 16-, 32-, and 64-bit respectively.

Note that if $x = 32,000$ then it is in short integer range, but $x + x$ corresponds to a negative number. However,

(4.2) Proposition *If x and y are in short integer range, and $x + y$ is in short integer range, then $x + y$ will be computed correctly as short (16-bit) integers.*

The same goes for 32-bit and 64-bit integers. (No proof.) ∎

Example. Convert 3276 and -2768 to short integers, add them (big-endian), and convert the result to decimal.

```
 3276   is   0ccc
-2768   is   f530
             01fc
```

One can check this by converting to decimal.

```
01fc = (16 x 1 + 15 ) x 16 + 12 = 508

 3276
-2768
  508 ------ results agree.
```

It is almost as easy to calculate 32-bit integer representations. For example, Convert 31415 and 31415 to int (big endian), and add

```
byte to hex 183 is b 7
byte to hex 122 is 7 a
byte to hex 0 is 0 0
byte to hex 0 is 0 0
31415: 00 00 7a b7
int 31415 hex  b7 7a 00 00

byte to hex 183 is b 7
byte to hex 122 is 7 a
byte to hex 0 is 0 0
byte to hex 0 is 0 0
31415: 00 00 7a b7
int 31415 hex  b7 7a 00 00

byte to hex 110 is 6 e
byte to hex 245 is f 5
byte to hex 0 is 0 0
byte to hex 0 is 0 0
The sum 62830: 00 00 f5 6e
int 62830 hex  6e f5 00 00
```