Integer Representation: Recap

Unsigned Integers

A memory location stores an unsigned integer as a binary encoding of the corresponding decimal number: 0 = 0_2, 1 = 1_2, 2 = 10_2, 3 = 11_2, 4 = 100_2, ..., 2^n-1 = 111...11_2 (a binary number with $n \times 1_2$), where n is the width of the number representation (typically 8, 16, 32, or 64 bits). Range 0 ... UMAX.

Signed Integers

Typically, but not always, stored as a two's complement number. (The C programming language standard does not say how a signed number should be encoded.) A positive number encoded the same an unsigned integer. A negative number encoded (typically) as a two's complement encoding:

Assume width of the number representation is n (e.g., 8, 16, 32, or 64 bits)
Take the number: −N (e.g., −1)
Remove the sign: N (+1 = 1_2)
Compute the complement: ~N (111...10_2, $n - 1 \times 1_2$ followed by 0_2)
Add one: ~N + 1 (111...10_2 + 1_2 = 111...11_2, $n \times 1_2$)

Range: −MAX − 1 ... MAX.

Why two's complement? Because the same adder (the piece of hardware that can compute the sum of two values) can be used for both unsigned and signed operands.

C Definitions

```
#include <limits.h>

SCHAR_MIN
SCHAR_MAX
UCHAR_MAX
SHRT_MIN
SHRT_MAX
USHRT_MAX
INT_MIN
INT_MAX
UINT_MAX
LONG_MAX
LONG_MIN
ULONG_MAX
```