Computer Systems Lecture 2

Data Representation

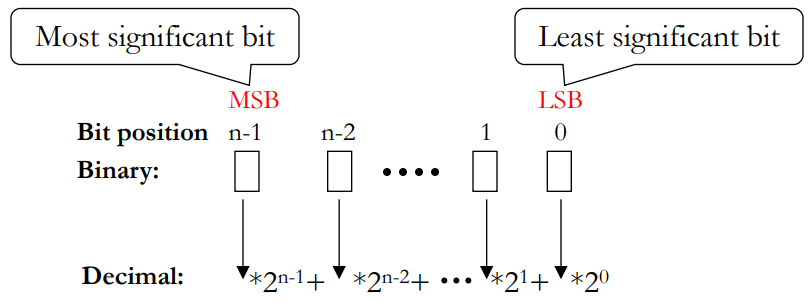
The way data is represented in computer hardware affects many things about a computer system: the complexity of circuits, cost, speed, reliability… This means we must consider how to design hardware for storing data (memory) and manipulating data (processing such as adders or multipliers).

The Bit

Information is represented as sequences of symbols, humans use letter, numerals, punctuation ect… but computers only use 1s and 0s, also known as bits (bitt is an acronym for binary digit). The advantages of this are that it’s easy to do computation with bits, its very reliable and the circuits used to process and store these are simply and highly reusable. The disadvantages are that there is little to no information per bit meaning we need to use many bits to represent not much data (512 = 100000000 or ‘A’ = 01000001).

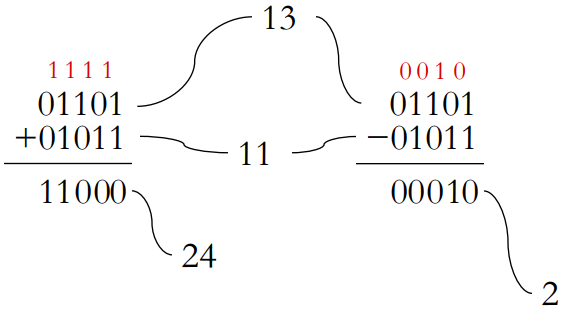
Natural Number Representation

Non-negative (unsigned) integers are very simple to represent in binary:



Basic Operations

Addition and subtraction with unsigned binary number is also easy:



Fixed Bit-Length Arithmetic

Hardware can’t handle infinitely long bit sequences so we end up with a few fixed-size data types:

* Byte: always 8 bits
* Word: the typical unit of data on which a processor operates (today 32 or 64 bits is most common)

Overflow happens when the result of a calculation doesn’t fit in the specified datatype

Negative Numbers

We have a couple ways of representing negative numbers with binary:

Sign-magnitude representation:

* Use the 1st bit as the sign with 1 meaning negative and 0 meaning positive meaning 0010 = 2 and 1010 = -2.

This makes addition and subtraction much more complicated as the operation depends on the sign, it also has two zeros (a positive 0000 = 0 and negative 1000 = -0).

There is a much better way to represent negative numbers, two’s complement.

Two’s Complement: The Intuition

Represent the negation of X as 2N-X where n is the number of bits being used. Then X + (-X) = X + (2N-X) = 2N. For example:

For a 3-bit fixed width N=3, X= 2 -> 010 in binary. 2N = 8 -> 1000 in binary. So -X = 2N-X = 8-2 = 6 -> 110 in binary. To check this we can use X + (-X) = 0: 010 + 110 = 1000 but we only look at the last 3 bits which are 000 = 0 in decimal.

Converting a number to two’s complement it actually much easier than that though, all you need to do is invert all the bits and add 1:

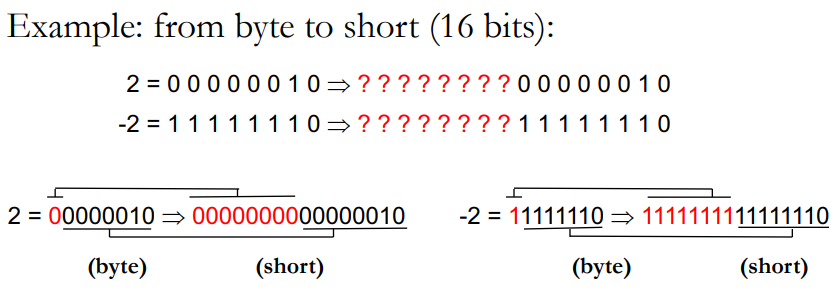
X=010, flip the bits = 101, add 1 = 110 = -X

Another way to think of this is that the MSB represents the negative of -2N-1 and then you add the number below it to that number in this case 110 is -4 + 2 = 2, this way to tell if its negative all you need to look at is the MSB.

It also works out that A-B = A + (the two’s complement of B) which makes subtraction a lot simpler as we can simply use the same circuitry as when we add. The range of two’s complement is asymmetric though, it goes from -2N-1 to -2N-1-1. For two’s complement there are two kinds of overflows, Positive overflow produces a negative number and Negative underflow produces a positive number.

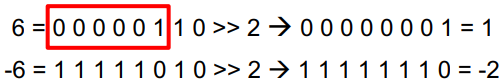
Converting Between Data Types

Converting a 2’s complement number from a smaller to a larger representation (bit length) is done by sign extension where you fill in the new gap with the MSB:



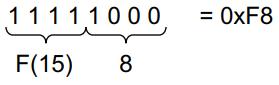
Shifting

Shifting is when you move the bits of a data type left or right (bits that fall off the edge are lost). For left shifts, 0s fill in the empty bit places, for right shifts though there are two options, you can fill with 0s (logical shift) for non-numerical data or you can fill with the MSB (arithmetic shift) for 2’s complement numbers. A shift left by n is equivalent to multiplying by 2n and shifting right by n is equivalent to dividing by 2n and rounding towards -infinity:



Hexadecimal Notation

Binary numbers and other binary-encoded information is often too long and tedious for humans to use. The solution to this is a more compact encoding, for this Hexadecimal (base 16) is most common. Hex digits go from 0-9 and A-F where A = 10, B = 11,…, F=15. We use Hex as conversion to/from binary is actually very easy as every 1 hex digit corresponds to 4 binary bits:

Its important to note that hex is just for humans, to make binary more bearable.