Computer Systems

Martin de Spirlet

Week	Unit	Title
1	1	Data Representation and Manipulation

1 Data Representation and Manipulation

1.1 Binary

Computers consist of a collection of switches called transistors, which can either be on or off. This leads to binary representation, where off is 0 and on is 1. All data and instructions are stored in binary. Binary digits are known as 'bits'. It is possible to build computers that use other number systems, but it is less expensive and more reliable to use basic components with only two states.

1.2 Characters

Characters are represented as binary values using text encoding schemes such as ASCII and Unicode. Modern schemes support internationalisation and tend to use 8 bit, 16 bit or 32 bit to encode characters. Strings are sequences of characters.

1.3 Number Bases

1.3.1 Decimal to Binary, Octal and Hexadecimal

234_{10} :

$$\left.\begin{array}{r|l} 2\lfloor 234 & 0 \\ 2\lfloor 117 & 1 \\ 2\lfloor 58 & 0 \\ 2\lfloor 29 & 1 \\ 2\lfloor 14 & 0 \\ 2\lfloor 7 & 1 \\ 2\lfloor 3 & 1 \\ 2\lfloor 1 & 1 \end{array}\right\} = 11101010_2$$

234_{10} :

$$\left.\begin{array}{r|l} 8\lfloor 234 & 2 \\ 8\lfloor 29 & 5 \\ 8\lfloor 3 & 3 \end{array}\right\} = 352_8$$

234_{10} :

$$\left.\begin{array}{r|l} 16\lfloor 234 & 10 \\ 16\lfloor 14 & 14 \end{array}\right\} = EA_{16}$$

1.3.2 Binary, Octal and Hexadecimal to Decimal

101011_2 :

$$\begin{aligned} 1 \times 2^0 &= 1 \\ 1 \times 2^1 &= 2 \\ 0 \times 2^2 &= 0 \\ 1 \times 2^3 &= 8 \\ 0 \times 2^4 &= 0 \\ 1 \times 2^5 &= \underline{32} \\ &= 43_{10} \end{aligned}$$

724_8 :

$$\begin{aligned} 4 \times 8^0 &= 4 \\ 2 \times 8^1 &= 16 \\ 7 \times 8^2 &= \underline{448} \\ &= 468_{10} \end{aligned}$$

ABC_{16} :

$$\begin{aligned} C \times 16^0 &= 12 \\ B \times 16^1 &= 176 \\ A \times 16^2 &= \underline{2560} \\ &= 2748_{10} \end{aligned}$$

1.3.3 Binary to Octal and Octal to Binary

$1011010111_2 = 1327_8$:

1 011 010 111

1 3 2 7

$705_8 = 111000101_2$:

7 0 5

111 000 101

1.3.4 Binary to Hexadecimal and Hexadecimal to Binary

$1010111011_2 = 2BB_{16}$:

10 1011 1011

2 B B

$10AF_{16} = 1000010101111_2$:

1 0 A F

0001 0000 1010 1111

1.3.5 Octal to Hexadecimal and Hexadecimal to Octal

Conversions between octal and hexadecimal can be performed by first converting to binary.

1.4 Integers in Binary

Counting in powers of 2.

Prefix	Power of 2	Value
kibi	2^{10}	1024
mebi	2^{20}	1048576
gibi	2^{30}	1073741824
tebi	2^{40}	1099511628000

1.4.1 Overflow

In a computer, an integer is represented by a fixed number of bits. The maximum value that can be stored in an unsigned integer of n bits is $2^n - 1$. It is possible that the addition of two n bit numbers yields a result that requires $n + 1$ bits.

In Java, the 'overflow' bits are lost with no error. The remaining bits give the wrong answer. It is important to make sure the data type used is big enough for the values it will represent.

1.4.2 Two's Complement

In 8 bit arithmetic, $255 + 1$ appears to be 0 due to overflow. In this case, 255 is behaving like -1. This leads to the 'two's complement' representation of negative integers in binary.

The two's complement representation of $-x$ is equivalent to the unsigned binary representation of $2^n - x$. Another method to find the representation is to flip all the bits of the binary representation of x and add 1 to the least significant bit.

Using two's complement, a signed binary integer of n bits can hold any value from -2^{n-1} to $2^{n-1}-1$, inclusive. The most significant bit represents -2^{n-1}. If the number of bits in a signed number is increased, the new bits are given the same value as the most significant bit. This is known as sign extension.

In Java, all integers are signed.

1.5 Real Numbers in Binary

1.5.1 Fixed Point Decimal to Binary

The integral part is converted as usual. The fractional part is converted by doubling as follows.

$$
\begin{aligned}
0.537_{10} = 0.100010_2 :& \\
0.537 \times 2 &= \underline{1}.074 \\
0.074 \times 2 &= \underline{0}.148 \\
0.148 \times 2 &= \underline{0}.296 \\
0.296 \times 2 &= \underline{0}.592 \\
0.592 \times 2 &= \underline{1}.184 \\
0.184 \times 2 &= \underline{0}.368
\end{aligned}
$$

1.5.2 Fixed Point Binary to Decimal

$$
\begin{aligned}
1101.0101_2 :& \\
1 \times 2^{-4} &= \ 0.0625 \\
0 \times 2^{-3} &= \ 0 \\
1 \times 2^{-2} &= \ 0.25 \\
0 \times 2^{-1} &= \ 0 \\
1 \times 2^{0} &= \ 1 \\
0 \times 2^{1} &= \ 0 \\
1 \times 2^{2} &= \ 4 \\
1 \times 2^{3} &= \ \underline{8} \\
&= 13.3125_{10}
\end{aligned}
$$

1.5.3 Floating Point Numbers

Fixed point is convenient and intuitive, but has two major problems.

1. Numerical precision — only values that are multiples of the smallest used power of two can be represented.
2. Numerical range — fractional precision comes at the expense of numerical range.

Floating point representation in binary is similar to scientific notation in decimal. In binary, real numbers can be represented in the form $\pm m \times 2^e$. Floating point numbers consist of a sign bit ($\pm$, 0 for positive or 1 for negative), a mantissa (m, the significant bits) and an exponent (e, two's complement signed binary).

S	Offset exponent, e'	Normalised mantissa, m'

The stored representation of a floating point number.

Since the mantissa is normalised (unless the value is small enough to represent without an exponent), the leading bit is almost always 1. It is therefore omitted from the stored representation. The exponent has a bias (offset) of $2^{n-1} - 1$ added to it for engineering purposes.

Floating point data types in Java.

Type	Bits				Bytes	Exponent bias
	Sign	Mantissa	Exponent	Total		
`float`	1	23	8	32	4	127
`double`	1	52	11	64	8	1023

With the 52 bit mantissa of a `double` (and the additional hidden bit), $2^{53} \approx 8 \times 10^{15}$. Hence, the `double` data type can be used to represent 15 significant decimal digits.

In Java, special values are returned for floating point overflow and other unusual circumstances. For example, if the result is too large for the `double` data type, `Double.POSITIVE_INFINITY` or `Double.NEGATIVE_INFINITY` are returned. If the result is indistinguishable from zero but known to be negative, `-0.0` is returned. If the result is not a real number, `Double.NaN` is returned.

1.5.4 Numerical Precision

Floating point arithmetic loses accuracy in its less significant digits. This is an issue even in values of type `double`.

It is a bad idea to use floating point representation for money — i.e. with integral pounds and fractional pence — as this will result in rounding errors. Taking £0.10 as an example, $0.10_{10} = 0.0\overline{0011}_2$.

Since currency is inherently integral, integer types with suitable range, such as `int`, `long` or `java.math.BigInteger`, should be used to represent multiples of the smallest denomination.