

The Interpretation of Binary Information

Brendan McKay

bdm@cs.anu.edu.au

Department of Computer Science, Australian National University, Canberra, ACT 0200, Australia

Abstract

Information within a computer is generally stored, transferred and processed in digital form. A single bit makes up the basic building block of this information. These bits are grouped together to form larger data types. Examples of these data types are: signed and unsigned integers, ASCII characters, floating point numbers, and address pointers. This article explains and illustrates some of these basic types.

Table of Contents

1 Introduction.....	1
• 1.1 Numbers.....	1
• 1.2 Unsigned Integers.....	1
• 1.3 Signed Integers	1
• 1.4 Text stolen from web pages.....	2
• 1.5 Acknowledgement.....	2
• 1.6 Conclusion	2

1 Introduction

A single bit has one of two possible states either 0 or 1. 8 bits are collected together to form one byte. Whereas 4 bits are call a nibble. This information is referred to as binary information – given a bit is limited to only one of two possible states. Within the computer it is processed using Boolean logic. Note that name comes from George Boole (1847) who formalized Aristotle's logic into an algebraic language for making logical inference.

A word consists of 2, 4, or 8 bytes. The word size is dependent on the architecture in question. For example a particular architecture may transfer and process information in blocks of 32 bits. Hence the word size for this architecture would be 4 bytes.

An interpretation is placed upon this binary information giving a particular combination of 0's and 1's a particular meaning. For example consider the following binary pattern.

0	1	1	0	0	0	0	1
---	---	---	---	---	---	---	---

If this byte is interpreted as an 8 bit integer then it would be the number 97. Whereas, if it is interpreted as an ASCII character then it would be considered the lowercase letter 'a'. The information within the byte has not changed, only the interpretation placed upon it.

Usually an interpretation involves creating a mapping from the bit pattern to the item of the interpretation. This places a bound on the number of different items within the interpretation. That is with 8 bits we can represent at most 2^8 items. Whereas with say 32 bits we can represent at

most 2^{32} items.

Brookshear (2005) provides a comprehensive introduction to these matters.

1.1 Numbers

Much of the computation undertaken by a computer involves numbers. Hence, we begin with numbers as an interpretation of binary information.

1.2 Unsigned Integers

The non-negative integers are the numbers 0, 1, 2, 3, Note that, this is an infinite set and hence can not be represented with a fixed sized bit pattern. However, if we limit the maximum integer we wish to represent then we can represent non-negative integers with a fixed size bit pattern. For example we can represent the set {0, 1, 2, ... 255} with a bit pattern of 8 bits. Also the simplest way of mapping the bit pattern to a number is to interpret the binary pattern as a binary number. So for example the binary pattern:

0	1	1	0	0	0	0	1
---	---	---	---	---	---	---	---

is interpreted as the binary number 01100001 which is in turn the number 97. This is calculated by changing the base of the number from 2 to 10:

$$0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 97.$$

The beauty of such an interpretation is that it simplifies indexing, comparison, and arithmetic operations for the computer's architecture.

One difficulty that often arises is determining which way around to interpret the bit pattern. Is the first bit the least or most significant bit? Moreover, when a number of bytes are stored contiguously in memory, to represent a number with a larger bound, a question remains over the ordering of bytes in terms of least significant to most significant. There are a number of different standards that are used so it is important there is consistency in the interpretation across all the interactions with this information. Fortunately, most of the work is done by the architecture and compiler/interpreter in question, problems only usually arise when information is transferred between systems.

1.3 Signed Integers

We also often wish to deal with negative numbers. One approach is to separate off a single bit within the binary number to indicate its sign. For example the first bit could

indicate the sign and the remaining bits are the number. So if the first bit is 0 then the number is positive and if it is 1 then number is negative. So the bit pattern 01100001 would be interpreted as 97, whereas, the bit pattern 11100001 would be interpreted as -97.

One problem with this approach is that you end up with two bit patterns for the number 0. That is 10000000 and 00000000 both represent 0. This not only wastes a bit pattern but it also makes the 'equality' operation more complex as $10000000 = 00000000$ even though the bit patterns are different.

The most common approach to represent signed integers is using a twos-complement approach. This both eliminates the overlapping mapping problem and simplifies a number of arithmetic operations.

The mapping of a four-bit number is shown in the following table. This can be simply extended to any number of bits. Note that the first bit still indicates whether or not the number is negative.

0111	→	7
0110	→	6
0101	→	5
0100	→	4
0011	→	3
0010	→	2
0001	→	1
0000	→	0
1111	→	-1
1110	→	-2
1101	→	-3
1100	→	-4
1011	→	-5
1010	→	-6
1001	→	-7
1000	→	-8

There is a simple rule for determining what the twos-complement number is representing. This rule is:

If the first bit is 0 then simply interpret it as a positive binary number.

If the first bit is 1 then invert all bits and add one, and then interpret it as a negative binary number.

For example with 0101 the first bit is 0 hence it represents the number 5. Whereas, with 1101 the first bit is 1, so we invert all bits, giving 0010, and add one, giving 0011. Now we interpret this as a negative binary number, hence we have -0011 (in base 2) or -3 (in base 10).

1.4 Text stolen from web pages

Most modern computer systems (including the IBM PC) operate using binary logic. The computer represents values using two voltage levels (usually 0V for logic 0 and either +3.3 V or +5V for logic 1). With two levels we can represent exactly two different values. These could be any two different values, but by convention we use the values zero and one. These two values, coincidentally, correspond to the two digits used by the binary number system.

Unfortunately, the storage of binary numbers in computers is not entirely standardized. Because computers store information in 8-bit bytes (where a bit is a single binary digit), depending on the "word size" of the machine, numbers requiring more than 8 bits must be stored in multiple bytes. The usual FORTRAN77 integer size is 4 bytes long. However, a number represented as (byte1 byte2 byte3 byte4) in a VAX would be read and interpreted as (byte4 byte3 byte2 byte1) on a Sun. The situation is even worse for floating-point (real) numbers, which are represented in binary as a mantissa and characteristic, and worse still for long (8-byte) reals!

1.5 Acknowledgement

Most of this document, apart from the purloined text in Section 1.4, was originally written by Eric McCreath.

1.6 Conclusion

All types of information can be stored in binary form. The key to unlocking this information is the way it is interpreted. Fortunately at the low level much of this work is done in hardware, however, more complex types and formats must be interpreted in software.

We have only looked at a few ways of interpreting binary information. There is also a large number of other common ways binary information is interpreted within a computer system. These include: characters, dates, floating point numbers, flags, records, arrays, lists, objects, bit-maps, colours, word documents, etc. We will leave it to the reader to investigate these possibilities.

References

- Boole, G. (1847). The mathematical analysis of logic, being an essay towards a calculus of deductive reasoning. Macmillan, Barclay, and Macmillan, Cambridge, UK.
- Brookshear, J. Glenn (2005). Computer Science, An overview, Addison Wesley, 8th edition.