Number Systems, Base Conversions, and Computer Data Representation

Decimal and Binary Numbers

When we write decimal (base 10) numbers, we use a positional notation system. Each digit is multiplied by an appropriate power of 10 depending on its position in the number:

For example:

$$\begin{aligned} 843 &= 8 \times 10^2 + 4 \times 10^1 + 3 \times 10^0 \\ &= 8 \times 100 + 4 \times 10 + 3 \times 1 \\ &= 800 + 40 + 3 \end{aligned}$$

For whole numbers, the rightmost digit position is the one's position ($10^0 = 1$). The numeral in that position indicates how many ones are present in the number. The next position to the left is ten's, then hundred's, thousand's, and so on. Each digit position has a weight that is ten times the weight of the position to its right.

In the decimal number system, there are ten possible values that can appear in each digit position, and so there are ten numerals required to represent the quantity in each digit position. The decimal numerals are the familiar zero through nine (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

In a positional notation system, the number base is called the radix. Thus, the base ten system that we normally use has a radix of 10. The term radix and base can be used interchangeably. When writing numbers in a radix other than ten, or where the radix isn't clear from the context, it is customary to specify the radix using a subscript. Thus, in a case where the radix isn't understood, decimal numbers would be written like this:

$$127_{10} \quad 11_{10} \quad 5673_{10}$$

Generally, the radix will be understood from the context and the radix specification is left off.

The binary number system is also a positional notation numbering system, but in this case, the base is not ten, but is instead two. Each digit position in a binary number represents a power of two. So, when we write a binary number, each binary digit is multiplied by an appropriate power of 2 based on the position in the number:

For example:

$$\begin{aligned} 101101 &= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 1 \times 32 + 0 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 \\ &= 32 + 8 + 4 + 1 \end{aligned}$$

In the binary number system, there are only two possible values that can appear in each digit position rather than the ten that can appear in a decimal number. Only the numerals 0 and 1 are used in binary numbers. The term 'bit' is a contraction of the words 'binary' and 'digit', and when talking about binary numbers the terms bit and digit can be used interchangeably. When talking about binary numbers, it is often necessary to talk of the number of bits used to store or represent the number. This merely describes the number of binary digits that would be required to write the number. The number in the above example is a 6 bit number.

The following are some additional examples of binary numbers:

$$101101_2 \quad 11_2 \quad 10110_2$$

Conversion between Decimal and Binary

Converting a number from binary to decimal is quite easy. All that is required is to find the decimal value of each binary digit position containing a 1 and add them up.

For example: convert 10110_2 to decimal.

```
1 0 1 1 0
 \   \ \_____________1 x 2^1 =  2
  \   \_____________1 x 2^2 =  4
   \_________________1 x 2^4 = 16
                              ---
                               22
```

Another example: convert 11011_2 to decimal

```
1 1 0 1 1
 \ \   \ \_____________1 x 2^0 =  1
  \ \   \_____________1 x 2^1 =  2
   \ \_________________1 x 2^3 =  8
    \___________________1 x 2^4 = 16
                                 ---
                                  27
```

The method for converting a decimal number to binary is one that can be used to convert from decimal to any number base. It involves using successive division by the radix until the dividend reaches 0. At each division, the remainder provides a digit of the converted number, starting with the least significant digit.

An example of the process: convert 37_{10} to binary

```
37 / 2 = 18        remainder 1        (least significant digit)
18 / 2 = 9         remainder 0
9 / 2 = 4          remainder 1
4 / 2 = 2          remainder 0
2 / 2 = 1          remainder 0
1 / 2 = 0          remainder 1        (most significant digit)
```

The resulting binary number is: 100101

Another example: convert 93_{10} to binary

```
93 / 2 = 46        remainder 1        (least significant digit)
46 / 2 = 23        remainder 0
23 / 2 = 11        remainder 1
11 / 2 = 5         remainder 1
 5 / 2 = 2         remainder 1
 2 / 2 = 1         remainder 0
 1 / 2 = 0         remainder 1        (most significant digit)
```

The resulting binary number is: 1011101

Hexadecimal Numbers

In addition to binary, another number base that is commonly used in digital systems is base 16. This number system is called hexadecimal, and each digit position represents a power of 16. For any number base greater than ten, a problem occurs because there are more than ten symbols needed to represent the numerals for that number base. It is customary in these cases to use the

ten decimal numerals followed by the letters of the alphabet beginning with A to provide the needed numerals. Since the hexadecimal system is base 16, there are sixteen numerals required. The following are the hexadecimal numerals:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

The following are some examples of hexadecimal numbers:

10_{16} 47_{16} $3EA_{16}$ $A03F_{16}$

The reason for the common use of hexadecimal numbers is the relationship between the numbers 2 and 16. Sixteen is a power of 2 ($16 = 2^4$). Because of this relationship, four digits in a binary number can be represented with a single hexadecimal digit. This makes conversion between binary and hexadecimal numbers very easy, and hexadecimal can be used to write large binary numbers with much fewer digits. When working with large digital systems, such as computers, it is common to find binary numbers with 8, 16 and even 32 digits. Writing a 16 or 32 bit binary number would be quite tedious and error prone. By using hexadecimal, the numbers can be written with fewer digits and much less likelihood of error.

To convert a binary number to hexadecimal, divide it into groups of four digits starting with the rightmost digit. If the number of digits isn't a multiple of 4, prefix the number with 0's so that each group contains 4 digits. For each four digit group, convert the 4 bit binary number into an equivalent hexadecimal digit. (See the Binary, BCD, and Hexadecimal Number Tables at the end of this document for the correspondence between 4 bit binary patterns and hexadecimal digits)

For example: Convert the binary number 10110101 to a hexadecimal number

Divide into groups for 4 digits	1011 0101
Convert each group to hex digit	B 5
	$B5_{16}$

Another example: Convert the binary number 0110101110001100 to hexadecimal

Divide into groups of 4 digits	0110 1011 1000 1100
Convert each group to hex digit	6 B 8 C
	$6B8C_{16}$

To convert a hexadecimal number to a binary number, convert each hexadecimal digit into a group of 4 binary digits.

Example: Convert the hex number 374F into binary

	3 7 4 F
Convert the hex digits to binary	0011 0111 0100 1111
	0011011101001111_2

There are several ways in common use to specify that a given number is in hexadecimal representation rather than some other radix. In cases where the context makes it absolutely clear that numbers are represented in hexadecimal, no indicator is used. In much written material where the context doesn't make it clear what the radix is, the numeric subscript 16 following the hexadecimal number is used. In most programming languages, this method isn't really feasible, so there are several conventions used depending on the language. In the C and C++ languages, hexadecimal constants are represented with a '0x' preceding the number, as in: 0x317F, or 0x1234, or 0xAF. In assembler programming languages that follow the Intel style, a hexadecimal constant begins with a numeric character (so that the assembler can distinguish it from a variable