Digital Systems and Information

Markus Gasser
TU Wien
markus.gasser@tuwien.ac.at

Omid Mirmotahari
Universitetet i Oslo
omidmi@ifi.uio.no

August 25, 2016

Abstract

This paper will give a brief overview of digital systems and information representations therein, mainly the binary, octal and hexadecimal number representations. It shows how to perform arithmetic operations and introduces codes for several applications.

I. Information Representation

Information values, also called signals can be broadly divided into two types: *analog* signals, which can take all possible values over a defined range and *discrete* signals, that can take only a finite number of values (e.g. integers) in a specific range.

An extreme case of the discrete signal is the *binary* signal, which can only take two values. In a physical information system these 2 values are typically represented by "high" (H) and "low" (L) voltage. Completely equivalent they can also be represented from a more theoretical viewpoint by "true" and "false" or just "0" and "1". To map these different representations to each other a convention called *positive logic* can be used, where H corresponds to true and 1, and L corresponds to false and 0.

A single binary digit in a system is called *bit* and groups of bits represent information according to their coding or even instructions to be executed within the system.

The dominant use of binary signals within systems instead of the use of higher order discrete signals or even analog signals is due to it's simplicity, ease of of design and reliability.

II. Number Systems

From everyday arithmetic we are already familiar with the *decimal number system* that represents numbers by strings of digits each an integer between 0 and 9 and representing the value of an integer raised to the power of 10 based on it's position in the string. As an example the the value of the string 24.5 can be computed as $2 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1}$. More general each number in an arbitrary number system can be represented by a string of coefficients $A_{n-1}A_{n-2} \dots A_1 A_0 . A_{-1} A_{-2} \dots A_{-m+1} A_{-m}$ and equivalently evaluated by the power series in r: $A_{n-1}r^{n-1} + A_{n-2}r^{n-1} + \cdots + A_1 r^1 + A_0 r^0 + A_{-1}r^{-1} + A_{-2}r^{-2} + \cdots + A_{-m+1}r^{-m+1} + A_{-m}r^{-m}$ where r is called the *radix* or *base*, "." the *radix point* and the coefficients are from the range of r values and they are multiplied by powers of r. The coefficient A_{n-1} is called *most significant digit* (MSD) and A_{-m} is called *least significant digit* (LSD). To distinguish between numbers of different bases the base can be indicated in subscript e.g. $(A_{n-1} \dots A_{-m})_r$.

i. Binary Numbers

In the binary number system $r = 2$ meaning it is a base 2 system with digits 0 and 1. To convert a binary number to it's decimal equivalent it suffices to evaluate the power series with a base of 2, e.g. $(110.1)_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} = (6.5)_{10}$. Decimal numbers can be converted to their binary representation by successively subtracting smaller powers of two and starting with the greatest one that is smaller than the decimal number e.g. $(13)_{10} = 2^3 + 2^2 + 2^0 = (1101)_2$

ii. Octal and Hexadecimal Numbers

Octal and Hexadecimal number systems are useful because they let us represent binary numbers indirectly since their bases are both powers of 2.

The *octal* number system is base $8 = 2^3$ and therefore let's us represent 3 binary digits with each octal digit. An example for a octal number is $(706)_8 = 7 \times 8^2 + 0 \times 8^1 + 6 \times 8^0 = (454)_{10}$.

The *hexadecimal* number system is base $16 = 2^4$ and thus each hexadecimal digit represents 4 binary digits. The digits 10, 11, 12, 13, 14 and 15 are represented by the letters A, B, C, D, E and F. An exemplary hexadecimal number is $(F3A)_{16} = 15 \times 16^2 + 3 \times 16^1 + 10 \times 16^0 = (3898)_{10}$.

Conversion from octal or hexadecimal to binary is performed by grouping the binary digits into groups of 3 or 4 respectively and then converting these groups individually. Converting from octal or hexadecimal to binary is done by reversing this procedure.

III. Arithmetic Operations

Arithmetic operations with numbers in base r follow the same rules as for the decimal number system. However one has to be careful only to use only the *r* allowed digits to perform all the operations. In the following sections the operations are explained by means of the binary number system.

i. Addition

The sum of two binary numbers is calculated the same way as in the decimal system however only the digits 1 and 0 are used. A *carry* to the higher position is obtained if the sum at any given position is greater than 1. As an example we calculate the sum of 0011 and 0110 as shown in table 1.

Augend	0010
Addend	+0110
Carries	1100
Sum	1000

Table 1: *Binary addition example*

Minuend	0110
Subtrahend	-0011
Borrows	0010
Difference	0011

Table 2: *Binary subtraction example*

ii. Subtraction

Subtraction in binary works analogous to the decimal system except that a *borrow* into a given digit adds 2 to the minuend bit. As an example we subtract 0011 from 0110 as shown in table 2.

iii. Multiplication

The binary multiplication is really simple, because the multiplier digits are always either 1 or 0. This the partial products are equal either to the multiplicand or 0.

IV. Decimal Codes

We now want to investigate how common decimal numbers can be saved in computers, i.e. being represented by binary digits. Thus we are searching for a *code* that converts our decimal digits into binary digits.

A *n-bit binary code* is a group of n bits that can assume up to 2^n distinct combinations of 1s and 0s. Each combination represents one element of the set being coded.

The code *binary-coded decimal* assigns the decimal digits $0, 1, \dots 9$ to a 4 bit string as shown in table 3.

According to this code the exemplary number 420 is represented by the 12 bits 0100 0001 0000.

Decimal digit	BCD digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Table 3: *BCD code*

Decimal digit	Binary Code	Gray Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Table 4: *Gray and binary code*

Note that a decimal number in BCD is generally *not* the same as its equivalent binary number, e.g. our example number $(420)_{10} = (1\,1010\,0100)_2 \neq (0100\,0001\,0000)_{\text{BCD}}$. Note also that generally a BCD number needs more bits than its equivalent binary value. It is most accurate to think of BCD numbers of decimal numbers written with binary digits, and not as binary numbers.

V. Alphanumeric Codes

Binary codes can not only represent digits as we discovered i the last section, but also letters.

i. ASCII Character Code

The most prominent code for letters is called *ASCII* (American Standard Code for Information Interchange). It consists of 7 bits and thus can code 128 characters. It entails the letters a-z, a-Z, the decimal digits 0-9, special printable and non-printable characters and control characters. As most computer systems use groupings of 8 bit length, also called one *byte*, a MSD is usually added. This digit can be set to 0, used for encoding international characters or used as a parity bit.

ii. Unicode and UTF-8

Since ASCII does not provide support for international character sets *Unicode* has been developed. Its goal is to provide a *code point* and a unique name for each character of the worlds languages. To encode this code points with binary digits multiple codes are available. The most prominent one is *UTF-8*, a variable length encoding ranging from 1 to 4 bytes per code point.

iii. Parity Bit

To add redundancy to a code and thus be able to detect errors during the transmission of a bit sequence a *parity bit* can be used. It is an extra bit that makes the total number of 1s in each codeword either even or odd. For example when adding even parity to the sequence 01011 at the MSD, the codeword becomes 101011.

VI. Gray Codes

There are application were when counting down or up, changes in multiple bits are problematic. The Gray code solves this problem. As can be seen in the table 4 two adjacent code symbols always only differ at one bit position.

VII. Summary

We introduced the basis of digital systems, the binary number system. We gave an overview over this system and the octal and hexadecimal number systems. We have shown how to perform arithmetic operations with them. We have shown multiple codes for different applications, most notably BCD, ASCII and Gray Code.

References

[Kime and Mano, 2004] Kime, Charles R and Mano, M Morris (2004). Logic and computer design fundamentals. *Pearson Education*.