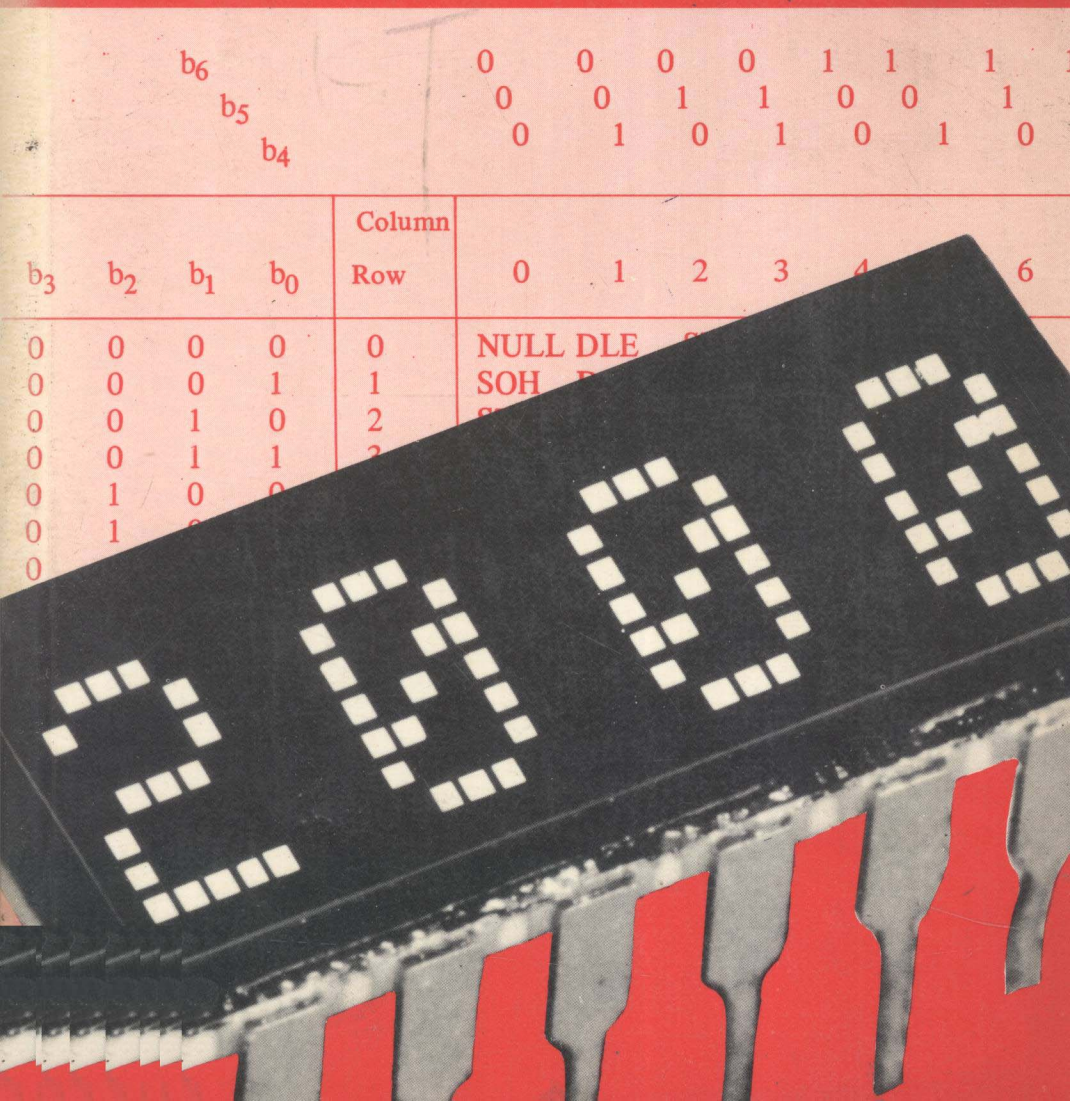
CODES
for Computers and Microprocessors

P.E. Gosling Q.L.M. Laarhov



Codes for Computers and Microprocessors

P. E. Gosling

Q. L. M. Laarhoven

First published 1980 by
THE MACMILLAN PRESS LTD
London and Basingstoke
Associated companies in Delhi Dublin
Hong Kong Johannesburg Lagos Melbourne
New York Singapore and Tokyo

Typeset in 10/12 Press Roman by
Styleset Limited, Salisbury, Wilts.
and printed in Hong Kong

British Library Cataloguing in Publication Data

Gosling, P E
Codes for computers and microprocessors.
1. Electronic digital computers
2. Coding theory
I. Title II. Laarhoven, Q L M
001.6'424 QA76.5

ISBN 0–333–25825–8

Preface

This book is about communication – communication between man and machines, and between machines themselves. It is intended to be as much a reference book as a textbook to serve as part of a course of study. On the one hand it will be useful in a practical sense for many people working, in particular, with minicomputers and now microprocessors. In both of these an understanding of what is going on inside is an aid to using the machines efficiently. On the other hand it will also be of value to students of computer science who need to be familiar with the techniques of data transmission. We hope that this book will add a practical dimension to the pure theory that is to be found in many textbooks dealing with computer science at all levels.

P. E. GOSLING
Q. L. M. LAARHOVEN

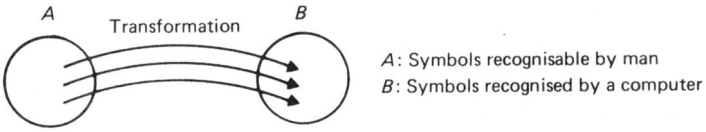
Introduction

The ability to exchange information between men and computers, and between computers themselves, relies on symbols which have a definite meaning agreed and arranged between both parties beforehand. Such symbols must have the same meaning for both partners in order to avoid any confusion or misinterpretation. The agreement on the meaning of the symbols used is usually built into the computer and learned by man. The series of agreements which represents the elements of a collection for communication is referred to as a *code*, and man has used symbols and codes for many years. For example, traffic lights and traffic signs together form a code which consist of a series of agreements about traffic behaviour. Another example is the Morse code which is a well known time-based code representing numbers and the letters of the alphabet.

The choice of a code for a given application depends on several factors. One of these is the suitability of the code for arithmetic calculation, another is the ability to detect mistakes and correct them. Modern computers only recognise the two symbols 1 and 0 because these are easily represented by electronic equipment. Our task in this book is to show some of the ways in which a variety of information can be contained in patterns which consist solely of 1s and 0s.

DATA REPRESENTATION

Codes can take any form as required by the user but in general they can be thought of as a mapping from a set of symbols A to a set of symbols B by a transformation according to some rule.



Information intelligible to man will consist of a set of symbols of which the letters of the alphabet and the digits 0 to 9 form a large part. The computer, however, operates through a series of electronic switches designed to allow current

to flow or not flow. Such a switch provides two possible options

Current/no current

which translates to

yes/no

on/off

1/0

such a unit of information is called a *bit* (*b*inary *i*nformation digi*t*).

There are a number of standard codes used to convey information through binary coding. One of these is the American Standard Code for Information Interchange (ASCII); another is the Extended Binary Coded Decimal Interchange Code (EBCDIC) and these will be discussed in detail later.

The result of coding a piece of information is called the *object* code and it is the purpose of all coding to produce an object code in binary digits for any combination of the 26 alphabetic characters, the digits 0 to 9, certain punctuation marks, graphical characters and arithmetic operators (+, −;=. . . .) together with what are termed control characters. A control character is one which represents an instruction to a certain piece of computer hardware, for example

Transmit/receive

Device control

Code extension control

Information separators

A code in binary form can represent any number of symbols provided that for every possible symbol there exists a unique binary code.

If only one binary digit can be used in an object code then there are only 2^1 combinations possible (0 or 1). If more than one binary digit is allowed then for n bits there are 2^n combinations possible. For example, if $n = 3$ there are $2^3 = 8$ combinations possible

000, 001, 010, 011, 100, 101, 110, 111

so that from 3 bits there are 8 possible object code patterns.

The more characters we need to represent, the greater the number of bits we need to use in order to distinguish between letters, figures, punctuation marks and special characters. The bits must be put together in such a way that there is no doubt as to their meaning. A sequence of bits are put together to form all or part of a computer *word*. If a word consists of eight bits

b0	b1	b2	b3	b4	b5	b6	b7

b7, the rightmost bit is called the *least significant bit* (LSB) and the leftmost bit, b0, is called the *most significant bit* (MSB).

Contents

1 *Weighted Codes*

In general a code needs to be selected systematically. Such a system could depend on

(1) time span plus frequency (as in the Morse code)
(2) Certain groups (figures together, small letters, capital letters).

A system is called a *weighted code* if there is an arithmetic connection between the code and decimal notation.

The decimal notation is itself an example of a weighted code. For example, the number 252 implies that the first 2 has a different meaning from the last 2. This is because the number is representing

2 one hundreds

5 tens

2 units

that is

$$200 + 50 + 2$$

$$= 2 \times 10^2 + 5 \times 10^1 + 2 \times 10^0$$

The *weight* of a particular digit is represented by its position relative to the other digits. This can be represented in the form of a progression as shown above.

The decimal number 641.8 can be represented by the progression

$$0 \times 10^3 + 6 \times 10^2 + 4 \times 10^1 + 1 \times 10^0 + 8 \times 10^{-1} + 0 \times 10^{-2}$$

Every real number expressed to a finite number of significant digits can therefore be represented by a series of the form

$$N = a_n \times g^n + a_{n-1} \times g^{n-1} + \cdots + a_1 \times g^1 + a_0 \times g^0 + a_{-1} \times g^{-1} + \cdots + a_{-m} \times g^{-m}$$

$$= \sum_{i=-m}^{i=n} a_i \times g^i$$

in which g represents the base, basis or radix and a the character or symbol.

To represent a number there are as many symbols as the size of the base. For $g = 10$ (decimal system) we have the ten symbols 0 to 9 inclusive. Between $a_0 \times g^0$ and $a_{-1} \times g^{-1}$ Continental practice is to use the comma, otherwise a full stop (point) is used. If $m = 0$ the number is an integer. If $n = 0$ the number is a fraction. This series is, in fact, a special form of a weighted code.

1.1 PROPERTIES

(1) If for every a_i in a number $a_i = g - 1$ (that is, the maximum value of a_i) then

$$\sum_{i=0}^{i=n-1} a_i \times g^i = g^n - 1$$

(for example, if $n = 3$, then $999 = 1000 - 1$)

$$\sum_{i=-k}^{i=n-1} a_i \times g^i = g^n - g^{-k} \text{ approaches } g^n$$

For example

$$999.99 = 1000 - 0.01$$

and

$$999.99 \cdots \rightarrow 1000$$

(2) Because

$$2 \times a_k \leqslant 2(g - 1)$$

then

$$a_k \times g^k + a_k \times g^k \leqslant 2(g - 1) \times g^k$$

so that

$$a_k \times g^k + a_k \times g^k + 1 \times g^k \leqslant 2(g - 1) \times g^k + g^k < 2 \times g^{k+1}$$

that is to say that when adding up two numbers, one does not have to carry more than one unit whatever the base.

(3)

$$g \times \sum_{i=0}^{i=n-1} a_i \times g^i = \sum_{i=0}^{i=n-1} a_i \times g^{i+1} + \sum_{i=1}^{i=n} a_{i-1} \times g^i + 0 \times g^0$$

that is to say that multiplication of an integer by the base means that the number moves one place to the left and a zero is placed in the units position.

1.2 SOME CONVERSION EXAMPLES

(1) *Binary system*: base 2; symbols: digits 0 and 1

$0 = 0 \times 2^0 = 0$

$1 = 1 \times 2^0 = 1$

$10 = 1 \times 2^1 + 0 \times 2^0 = 2$

$11 = 1 \times 2^1 + 1 \times 2^0 = 3$

$100 = 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 4$

$101 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$

(2) *Hexadecimal code*: base 16; symbols: digits 0 to 9 plus letters A to F

$0 = 0 \times 16^0 = 0$

$1 = 1 \times 16^0 = 1$

$2 = 2 \times 16^0 = 2$

$A = A \times 16^0 \ (= 10 \times 16^0) = 10$

$B = B \times 16^0 \ (= 11 \times 16^0) = 11$

$10 = 1 \times 16^1 + 0 \times 16^0 = 16$

$A4 = A \times 16^1 + 4 \times 16^0 = (10 \times 16^1 + 4 \times 16^0) = 164$

(3) *Octal code*: base 8; symbols: digits 0 to 7

$0 = 0 \times 8^0 = 0$

$1 = 1 \times 8^0 = 1$

$10 = 1 \times 8^1 + 0 \times 8^0 = 8$

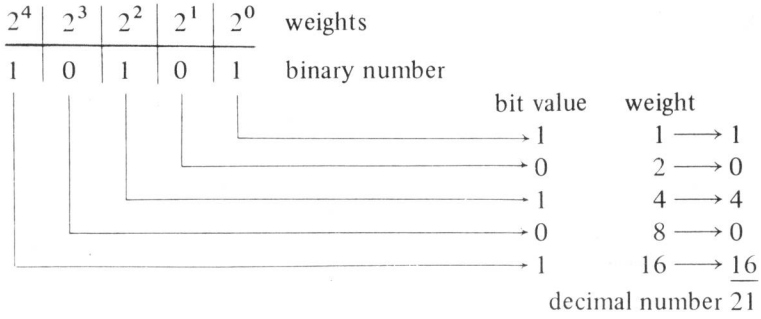
$11 = 1 \times 8^1 + 1 \times 8^0 = 9$

$100 = 1 \times 8^2 + 0 \times 8^1 + 0 \times 8^0 = 64$

If several number systems are intermixed, it is common practice to put the base as a subscript after the number, thus $101_2 = 5_{10}$.

1.3 CONVERSION OF BASE g TO DECIMAL

To convert a binary number to its decimal equivalent we make use of the weighted code as follows.



In general we find the value of a binary number by multiplying every digit by its weight and summing the products. This will, of course, apply to any weighted code. The number with the greatest weight is placed on the far left (in the most significant position) and the number with the least weight is placed on the far right (in the least significant position). Normally, leading zeros are not written.

1.4 ARITHMETIC RULES FOR BINARY NUMBERS

Addition	*Subtraction*
0 + 0 = 0	0 − 0 = 0
0 + 1 = 1	0 − 1 = 1 with borrow 1
1 + 0 = 1	1 − 0 = 1
1 + 1 = 0 plus carry 1	1 − 1 = 0
Multiplication	*Division*
0 x 0 = 0	0/1 = 0
0 x 1 = 0	1/1 = 1
1 x 0 = 0	0/0 = ?
1 x 1 = 1	1/0 = ?

1.5 WORD LENGTH

In a computer most components (memory, arithmetic and logic units, registers, data channels) can only contain a certain number of bits so that each can contain a fixed-length computer word. A computer word need not be used solely for the representation of a number; it can contain operation codes or other information

such as sign bits. The division of a word into various parts is reflected in its word *format*.

The length of word used can vary considerably. Commonly used are word lengths of 8, 12, 16, 18, 24, 32, 48 and 64 bits. Most microcomputers have a word which is made up of 8 bits, commonly referred to as a ***byte***, while the majority of mini-computers use a 16-bit word, that is, two bytes. Incidentally, in microcomputers half a byte is often referred to as a ***nibble***! Large mainframe computers in general use words with 32 or 64 bits. From this it is obvious that the word length and the format of the word determine to a large extent the architecture of a computer. In general, most designs use a word of fixed length with a double-length word to allow for an extended range of instruction words and greater accuracy of arithmetic calculations. The use of a word of variable length would complicate the way a computer works to a considerable extent.

1.6 CONVERSION OF DECIMAL TO BINARY

There are two ways in which a decimal number can be converted into its binary equivalent.

(1) *Subtraction of Powers*

Subtract the highest power of 2 possible from the decimal number which leaves a positive remainder and place a 1 in the appropriate place in the weight table. Repeat this procedure with the remainder, the remainder's remainder, and so on, until the decimal number has been reduced to zero. If after a subtraction the next power of 2 cannot be subtracted a 0 must be written in the weight table.

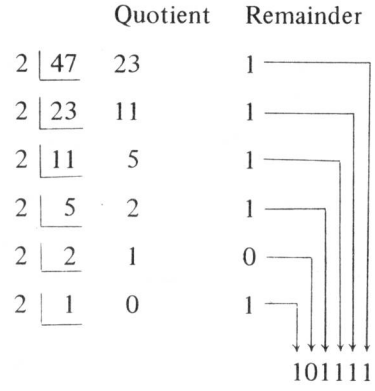
Example The binary equivalent of 42_{10}

42	10	10	2	2	0	2^5	2^4	2^3	2^2	2^1	2^0	
−32	−16	−8	−4	−2	−1	32	16	8	4	2	1	
10	−	2	−	0	−	1	0	1	0	1	0	$= 101010_2$

(2) *Division Method*

Divide the decimal number by 2. If there is a remainder, put 1 in the LSB position. When there is no remainder, put 0 in front of the previous digit. Divide the quotient from the previous division by 2 and repeat the process.

Example The binary equivalent of 47_{10}



1.7 OTHER NOTATION FORMS FOR BINARY CODES

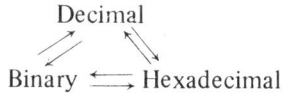
The hexadecimal system is used to present large binary numbers in a more compact form; this limits the likelihood of mistakes in transmission. The hexadecimal system has 16 digit symbols and the base number is 16. The symbols are the digits 0 to 9 and the letters A to F with weights of 0 to 15 inclusive.

The hexadecimal value of a binary number can be determined by dividing the bits up into groups of four starting with the least significant bit. Each group of 4 bits represents 16 possible numbers. For example, a number can be represented as A673D and will have a decimal equivalent of

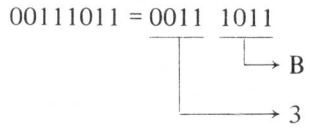
$$A \times 16^4 + 6 \times 16^3 + 7 \times 16^2 + 3 \times 16^1 + D \times 16^0$$

$$= 10 \times 16^4 + 6 \times 16^3 + 7 \times 16^2 + 3 \times 16^1 + 13 \times 16^0$$

On the basis of the conversion triangle below, an example from every conversion can be worked out.



Binary to Hexadecimal



Hence

$$00111011_2 = 3B_{16}$$

Hexadecimal to Binary

$$1D4_{16} = 0001\ \ 1101\ \ 0100$$
$$1 \quad D \quad 4$$
$$= 000111010100_2$$

Weights	*Binary* 8 4 2 1	*Hexadecimal*	*Decimal*
	0 0 0 0 =	0	0
	0 0 0 1 =	1	1
	0 0 1 0 =	2	2
	0 0 1 1 =	3	3
	0 1 0 0 =	4	4
	0 1 0 1 =	5	5
	0 1 1 0 =	6	6
	0 1 1 1 =	7	7
	1 0 0 0 =	8	8
	1 0 0 1 =	9	9
	1 0 1 0 =	A	10
	1 0 1 1 =	B	11
	1 1 0 0 =	C	12
	1 1 0 1 =	D	13
	1 1 1 0 =	E	14
	1 1 1 1 =	F	15

1.8 OCTAL NUMBER SYSTEM

The octal number system, base 8, symbols 0 to 7, is often used in minicomputers so that words containing 12 bits can be made more manageable by dividing the word into four groups of 3 bits each. If a word contains 16 bits it is usually divided up into five groups of 3 bits and one of 1 bit only.

Example By splitting the binary number up into four groups of three bits and starting with the least significant bit we get 11010111101 divided into 011 010 111 101. These binary groups can be replaced by their octal equivalents, that is

$$011_2 = 3_8 \quad 010_2 = 2_8 \quad 111_2 = 7_8 \quad 101_2 = 5_8$$

The octal equivalent of the number is therefore 3275. This method is easily used to convert an octal number to binary

$$5307_8 = 101\ 011\ 000\ 111$$

Octal–Decimal Conversion

Octal numbers can be converted to decimal by using the weights of the digits as before. The weights are, of course, powers of 8.

Example

$$2167_8 = 2 \times 8^3 + 1 \times 8^2 + 6 \times 8^1 + 7 \times 8^0$$
$$= 1024 + 64 + 48 + 7$$
$$= 1143_{10}$$

Decimal–Octal Conversion

As with binary–decimal conversion there are two important methods for this operation.

(1) *Subtraction of Powers*

Subtract from the decimal number the highest possible value of $a \times 8^n$, such that a positive remainder is left, where a is a number between 0 and 7 and n starts at one less than the number of decimal digits before the decimal point. Write down the value of a. Continue with decreasing powers of 8 until the decimal number is reduced to zero. Whenever the subtraction of powers of 8 is not possible write down a zero.

Example The octal equivalent of 2591_{10}

$$\begin{array}{rl} 2591 & \\ -\ 2560 & = 5 \times 8^3 = 5 \times 512 \\ \hline 31 & \\ -\ 0 & = 0 \times 8^2 = 0 \times 64 \\ \hline 31 & \\ -\ 24 & = 3 \times 8^1 = 3 \times 8 \\ \hline 7 & \\ -\ 7 & = 7 \times 8^0 = 7 \times 1 \\ \hline 0 & \end{array}$$

Hence

$$2591_8 = 5037_{10}$$

The following conversion table is very useful in this context.

Position of octal number	0	1	2	3	4	5	6	7
1 8^0	0	1	2	3	4	5	6	7
2 8^1	0	8	16	24	32	40	48	56
3 8^2	0	64	128	192	256	320	384	448
4 8^3	0	512	1024	1536	2048	2560	3072	3584
etc.								

(2) *Conversion by Division*

Divide the decimal number by 8 and place the remainder as the least significant digit of the octal equivalent. Repeat this with increasing weights until the remainder is zero.

Example 1376_{10} in octal

		Quotient	Remainder
8	1376	172	0
8	172	21	4
8	21	2	5
8	2	0	2

Hence

$$1376_{10} = 2540_8$$

Both of these methods can be adapted to the hexadecimal system.

1.9 FRACTIONS

In all number systems fractions are represented in the same way as in the decimal system and we will use the point (.) rather than the Continental comma.

Conversion from Decimal Fractions into Binary and Octal Fractions

Subtraction of Powers

In this case the highest possible negative power is multiplied by one of the digits in the number system and is subtracted from the decimal number, and so on. The multiplying digit is then written down in order to construct the new number.