Programming for Microprocessors

ANDREW COLIN MA, FBCS, CEng, FIEE

Programming for Microprocessors

ANDREW COLIN MA, FBCS, CEng, FIEE
Professor of Computer Science,
University of Strathclyde, Scotland

NEWNES – BUTTERWORTHS
LONDON BOSTON
Sydney Wellington Durban Toronto

The Butterworth Group

United Kingdom	**Butterworth & Co (Publishers) Ltd** London: 88 Kingsway, WC2B 6AB
Australia	**Butterworths Pty Ltd** Sydney: 586 Pacific Highway, Chatswood, NSW 2067 *Also at Melbourne, Brisbane, Adelaide and Perth*
Canada	*Butterworth & Co (Canada) Ltd* Toronto: 2265 Midland Avenue, Scarborough, Ontario M1P 4S1
New Zealand	**Butterworths of New Zealand Ltd** Wellington: T & W Young Building, 77–85 Customhouse Quay, 1, CPO Box 472
South Africa	**Butterworth & Co (South Africa) (Pty) Ltd** Durban: 152-154 Gale Street
USA	**Butterworth (Publishers) Inc** Boston: 19 Cummings Park, Woburn, Mass. 01801

First published 1979

British Library Cataloguing in Publication Data

Colin, Andrew John Theodore
Programming for microprocessors.
1. Microprocessors–Programming
I. Title
001.6'42 QA76.6 78-40694

ISBN 0-408-00320-0

Typeset by Reproduction Drawings Ltd., Sutton, Surrey
Printed in England by Cox & Wyman Ltd, London,
Fakenham and Reading

1

General introduction

The microprocessor

Many major advances in human knowledge have resulted from the cross-fertilisation of two different disciplines. One product of such a union is the microprocessor, which represents the flowering both of electronics and of computer science.

The microprocessor brings a new dimension of freedom, capability and power to both areas. The electronic engineer can now design control and signal-processing systems of unparalleled reliability, speed and simplicity, while the computer scientist can obtain as much computer power as he needs at a small fraction of its previous cost.

The mixed parentage of the microprocessor gives it two entirely different aspects, each of which tells only part of the truth. An engineer schooled in conventional electronics might think of it as the culmination of a line of digital components starting with elementary logic gates, and including registers, adders, arithmetic units and stores. This view is correct as far as it goes, but fails to take into account the 'computer' dimension of the microprocessor. An engineer who saw the device solely as an electronic component would not be able to put it to effective use.

The other aspect of the microprocessor is seen by the computer scientist. When he first reads a description, the computer scientist can see no essential difference between the structure of the microprocessor and the large mainframes or minicomputers he has been using hitherto. It has basically the same arrangement of store, arithmetic and control units, and it is clear that all the programming techniques and tools such as assemblers, editors and high-level languages are as applicable to microprocessors as they are to conventional computers. The main stumbling block to the computer scientist is that the microprocessor is only a component, and to be of any use it must be built into a complete system with other components. This severely practical point has so far prevented some computer scientists from taking a serious interest in microprocessors.

It appears, then, that to make full use of microprocessors the system designer must be grounded both in electronics and in computer science subjects that traditionally cover different areas. Unlike computer scientists, electronic engineers have shown a great deal of enthusiasm for the new development; but they have often encountered great difficulties because of their inexperience in the techniques of computer science. To the computer specialist's eye, engineers often painfully re-invent methods that were fully understood 20 years ago.

The present writer is a computer scientist with some knowledge of digital electronics. This book attempts to bridge the crevasse between these two subjects, by starting from the side of the engineer, taking for granted a basic knowledge of digital electronics and concentrating on the computer-like aspects of microprocessors.

Hardwired and programmed systems

The basic feature that makes a microprocessor more than just a component is its generality, and its ability to switch between several different functions within a few microseconds.

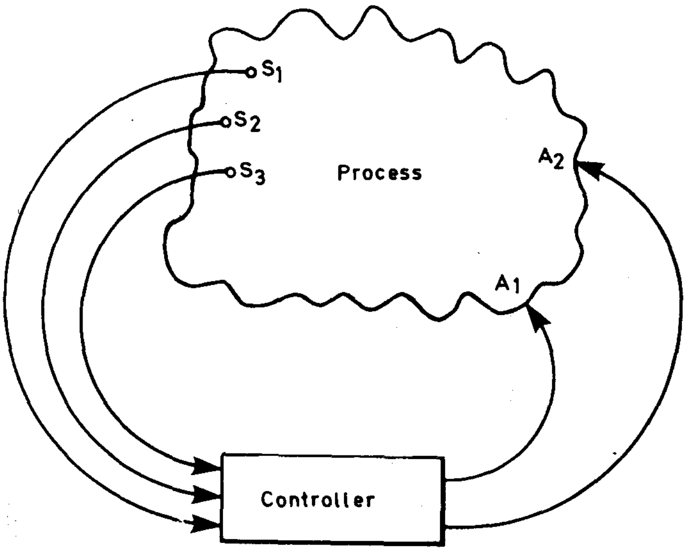

Figure 1.1

Consider a digital controller for a chemical process (*Figure 1.1*). Sensors S_1, S_2 and S_3 supply essential information about temperatures, pressures, flow rates and so on. Actuators A_1 and A_2 control valve settings and motor speeds so as to keep the process running at the right conditions for the best yield. The controller may also double as a 'safety device', shutting down the entire process or opening a relief valve if one of the sensors reports a value that exceeds some predetermined mean.

In the past, a controller of this type could well have been built using traditional components such as relays or elementary logic gates. The inputs would have been converted from analogue to digital form by A/D converters, and the outputs would have been handled by D/A converters whenever necessary. This type of construction is called 'hardwired', because the control function is totally embodied in the components and their connections. A common feature of a hardwired design is that every aspect of a specification is translated directly into its own assembly of components. Thus a safety requirement involving five inputs would be implemented by five connected relays or a five-input logic network.

Today, the same control function can be provided by a microprocessor. The controller still has the same connections to the system being controlled, and the signals must still be converted to and from the digital form; but the internal arrangement is now totally different. Instead of many components each dedicated to a specific purpose, the system has only a few parts; but they are configured so that they can carry out all the necessary functions one after the other in sequence. When the system has executed all the control functions once, it loops back and executes them again, and so on continuously. Each control function can be executed every few milliseconds, giving a close approximation to continuous control.

The information governing the control functions is stored as a series of 'instructions' in a store attached to the control system. As soon as it has finished executing any one function, the system fetches the instructions governing the next function from this store. A typical set of control instructions might carry the meaning:

1. If temperature $> 250^\circ$ ring alarm
2. If pressure > 50 kg/cm^2 ring alarm
3. If flow > 102 litre/s close inlet valve one step
4. If flow < 98 litre/s open valve one step

19. Wait 2 seconds
20. Go back to step 1

It should be noted that the actual form in which the instructions are stored is considerably different from the one shown above; the intention, however, is the same.

Figure 1.2 shows the internal structure of a control system based on a microprocessor. A vital feature of this structure is that it is the *same* for any controller. Different control functions can be obtained simply by changing the instructions held in the store. In the limit a microprocessor can implement any control function, no matter how complex.

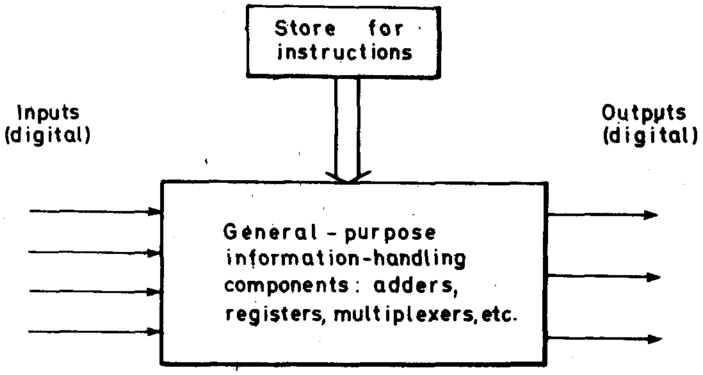
Store for instructions
Inputs (digital)
Outputs (digital)
General - purpose information-handling components : adders, registers, multiplexers, etc.

Figure 1.2

The only condition is that the function be capable of complete and unambiguous definition.

The set of instructions that produces any control function is called a 'program'. A microprocessor is therefore often referred to as a 'programmed' device, to distinguish it from one that is hardwired.

Some contrasting properties of hardwired and programmed devices

It is worth comparing hardwired and programmed systems, since they differ in several important ways.

Complexity of control

One measure of control function is its complexity – the number of inputs and outputs, the rules that relate them, and the number of exceptions. Some control functions, like automatic aircraft landing devices, are inherently complex and others can be made complex by providing alternative courses of action if any part of the system should fail.

The overall size of a hardware system is directly connected to the complexity of the control function. This fact forces a practical limit to complexity, since a system that is too large will be difficult to test and maintain, or even too bulky physically to fulfil its requirements. On the other hand the complexity of a microprocessor system can in principle be increased indefinitely, simply by adding more instructions to the program. There is practically no increase in size, or in the number of physical components.

Cost

Microprocessors now cost a negligible amount if bought in large quantities. They normally need to be equipped with power supplies and with other components such as stores; but even so, the component cost of a small microprocessor system is competitive with all but the very simplest electromechanical devices.

The design cost is a different matter. The most difficult part of designing a microprocessor system is the correct specification of the program. This needs relatively expensive equipment and some knowledge of computer science on the part of the design engineer. Furthermore, the need to learn a new technique has made the design costs of many microprocessor systems seem very high, but this can be regarded as an investment for the future. Nevertheless, it can still be argued that, to produce a simple control system in small quantities, a traditional hardwired approach is sometimes good economics.

Reliability

The reliability of a hardwired system depends on its size. Published statistics for the failure rates of individual components, soldered and wrapped joints can be used to calculate the 'Mean Time Between Faults' of any hardwired system. This figure can often be adjusted to any desired value by using more reliable components, improving ventilation, or duplicating or triplicating the whole system. This analysis assumes that the basic design is right; but since the behaviour of each separate control function can usually be checked independently, the assumption is usually valid.

Superficially, a system based on a microprocessor can be analysed in the same way. Since it has fewer components, it will appear far more reliable than its hardwired counterpart; but this again assumes that the design is correct. Here the assumption is dangerous, for most of the design in the system is in the controlling program, and it is far from easy to ensure that a series of instructions will always do exactly what they are intended to. There are two particular points to observe.

Firstly, mistakes in a program are errors in *design*. The failures they cause cannot be guarded against by duplication of the system (since both sets of equipment would make the same error at the same time), and they always come without prior warning.

Secondly, programming is an activity fraught with psychological difficulty. It has been compared to compulsive gambling. Most people are unreasoning optimists about their programs, and just as the gambler 'knows' that this time he has at last understood the mysterious forces behind the ivory ball, and that he must win on the next spin, so the programmer feels a moral certainty that his program is right. This conviction continues even though error after error comes to light; the programmer is always sure that every mistake is the last one. This universal tendency makes it very difficult to adopt the pessimistic, cynical viewpoint that alone can lead to programs that are genuinely correct. In practice, many control systems are fitted with programs that are only partially tested and riddled with mistakes. The accuracy of the program is the greatest single factor in determining the reliability of any microprocessor system. Methods of checking programs and verifying their correctness will be discussed at length later.

A classification of microprocessors

The market for microprocessors has expanded rapidly, and many different kinds are available.

Two important ways of making electronic circuitry are the 'bipolar' and 'MOS' technologies. Bipolar integrated circuits are about ten times faster than those made with MOS, but each transistor takes more room and dissipates more heat. This places a strict limit on the number of transistors on a chip, and therefore on the complexity of the chip as a whole. Most bipolar microprocessors take the form of 'bit slices' – components that can be put together in various ways to build powerful computers. The application areas and design methods for bit slices are somewhat specialised and entirely different from those associated with MOS microprocessors, and will not be considered further here.

With MOS technology, one chip can contain enough components to form a self-contained computer, albeit one that runs rather slower than a minicomputer or 'mainframe' machine. Most existing microprocessors are based on MOS.

Within this group, there is still a huge selection of devices. They share a common basic architecture, but differ in speed, in the amount of information they can handle at any time, in the complexity of the operations they actually do, in the way they are connected to the external world, in their power-supply requirements, and in their immunity to mechanical vibration and electrical noise. Other factors that might affect the selection of a microprocessor for a particular task are the number of manufacturers (to ensure continuity of supply) and the provision of software aids. Perhaps the most important single point is the degree to which the system design engineer is familiar with a particular type of microprocessor; the sheer labour of learning about a new family of microprocessors is so great that brand loyalty becomes a virtue.

Summary

This introductory chapter has made the following points:

- Microprocessors have a very low cost, and are extremely flexible in use. This combination offers many large application areas for exploitation.
- Microprocessors can be used to implement control functions of any complexity.
- The reliability of a microprocessor system depends chiefly on the accuracy of the controlling program. Correct programs are difficult and expensive to design.

2

Representation of data

This and the next few chapters deal with the intellectual and practical tools that the designer of a microprocessor system will need. The reader is advised to scan the material once for an overview – to look, as it were, into the tool cupboard – and then to return to it later, when he has begun work on an actual problem.

The binary system

Microprocessors are essentially devices for handling and processing *information*. In a digital system information is represented by binary digits, each of which can have only two possible values: 1 and 0. (These values often have other names, like 'true and false', 'set and clear', or 'mark and space'. The actual names used are not important, so long as it is understood that, for example, '1', 'true', 'set' and 'mark' all mean the same thing.)

In most circumstances, binary digits (or 'bits') are not handled singly, but in groups of fixed size called 'bytes' or 'words'. Various microprocessors have words of 4, 6, 8, 12 or 16 bits. However, eight bits is a useful and common word size, and we shall assume it in our examples.

The bits in a computer word have no predetermined meaning. Their significance always depends on the context, and is fully controlled by the system designer. There are, however, four basic types of information, each with its own conventions and methods of being handled: Boolean information, numerical information, character codes, and machine code.

Boolean information

Boolean information is the name used for data items that can have only two possible values, such as 'set' or 'clear'. On the input side, Boolean items may be derived from switch settings, relays or comparators. On output, Boolean signals control lamps, alarms, and other devices with simple 'on/off' characteristics.

Clearly, a Boolean data item can be mapped on to a single binary digit. One possible coding is for 'on' to be represented by 1 and 'off' by 0.

A word of eight binary digits can be used to contain up to eight different independent Boolean data items. If the items are related to one another (like the pushbuttons in a lift car) they are often packed into a single word; but otherwise it is more convenient to give each Boolean item a word to itself. By convention, the value is replicated, so that 'on' is represented by 11111111, and 'off' by 00000000.

Numerical information

Much of the data in any system is in the form of numbers. The values can be derived from external sources through A/D converters, or they can be generated internally whilst a program is being executed.

In a word with eight binary digits, there are 2^8 or 256 different combinations of 0's and 1's. *In principle*, these combinations can be used to represent numbers in any arbitrary code whatsoever. In practical terms, however, it is best to keep to some variant of the binary (radix two) notation, because microprocessors are designed to do arithmetic on this assumption.

The binary notation is a positional system, identical in principle to the denary system used in everyday life. A denary number is implicitly written with a radix of ten; it is understood that the various digits refer to units, tens, hundreds, thousands, etc., which are all powers of ten. Thus '1978' is interpreted as:

$$1 \times 10^3 + 9 \times 10^2 + 7 \times 10^1 + 8 \times 10^0$$

(The reader will remember that $10^0 = 1$.) An important point about the denary system is that it uses only ten different symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. Numbers greater than nine are written using combinations of the basic symbols.

In the binary system, the radix is two, and the only symbols needed are 0 and 1. A number like '10110' is interpreted as:

$$1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 \times 1 \times 2^1 + 0 \times 2^0$$

which equals 'twenty-two'.

Binary numbers are in general about three times longer (in terms of digits) than their decimal counterparts. The advantage of using binary numbers within a computer is that arithmetic is very simple.

Addition, for example, is governed by the table:

$$\boxed{\begin{array}{l} 0+0=0 \\ 0+1=1 \\ 1+0=1 \\ 1+1=0,\ \text{carry } 1 \end{array}}$$

The addition of two binary numbers could be written as follows:

$$\begin{array}{r} 101101 \\ 10111\ + \\ \hline 1000100 \end{array} \qquad \left(\begin{array}{r} 45 \\ 23\ + \\ \hline 68 \end{array}\right)$$

It is sometimes necessary to convert numbers between their denary and binary forms. Conversion from denary is best made by repeatedly dividing by two, and noting down the remainder each time. Finally the remainder digits are assembled, the first on the right and the last on the left. For example, to convert 87 to binary, we put:

$$\begin{array}{cccc} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} \\ \begin{array}{r} 43 \\ 2\overline{)87} \end{array}\ \text{r}\,1 & \begin{array}{r} 21 \\ 2\overline{)43} \end{array}\ \text{r}\,1 & \begin{array}{r} 10 \\ 2\overline{)21} \end{array}\ \text{r}\,1 & \begin{array}{r} 5 \\ 2\overline{)10} \end{array}\ \text{r}\,0 \\ \textcircled{5} & \textcircled{6} & \textcircled{7} & \\ \begin{array}{r} 2 \\ 2\overline{)5} \end{array}\ \text{r}\,1 & \begin{array}{r} 1 \\ 2\overline{)2} \end{array}\ \text{r}\,0 & \begin{array}{r} 0 \\ 2\overline{)1} \end{array}\ \text{r}\,1 & \end{array}$$

$$\therefore 87 = \overset{\textcircled{7}\ldots\textcircled{1}}{1010111}$$

To make the conversion the other way, it is easiest to use a table of the powers of two:

$2^0 = 1$	$2^5 = 32$	
$2^1 = 2$	$2^6 = 64$	
$2^2 = 4$	$2^7 = 128$	
$2^3 = 8$	$2^8 = 256$	
$2^4 = 16$	$2^9 = 512$	etc.

The conversion is made by adding up those powers of two represented by 1's in the binary number. For example:

$$1010111 = 2^6 + 2^4 + 2^2 + 2^1 + 2^0 = \begin{array}{r} 64 \\ 16 \\ 4 \\ 2 \\ 1\ + \\ \hline 87 \end{array}$$

These methods are suitable for humans, who normally think in the denary system. For a being (or a machine) that normally worked in the binary system the methods would be inverted, relying on division by ten, and a table of powers of ten.

The binary system is used in a number of variants, three of which will now be described.

Unsigned binary numbers

In this notation, numbers are represented in straightforward binary code. The only difference is that each number always has a fixed number of digits (eight in our examples). This implies that small numbers have leading zeros, and that there is a maximum size to the numbers that can be represented at all. For example:

00000011 = three

11111111 = two hundred and fifty-five (the largest number in an eight-bit system)

The presence of the leading zeros matters very little, but programmers must constantly be aware of the size limitation. Any calculation that could give a result greater than this limit may simply go wrong unless special precautions are used. Thus:

```
   11111010        (250)
   00001010   +     (10)
(1)00000100
```

The carry from the most significant stage of the addition drops off the end, leaving the (incorrect) answer 4.

Binary coded decimal ('BCD')

If a microprocessor is controlling a display made up of decimal digits, or reading decimal numbers from a keyboard, it is often convenient to allow each eight-bit word to represent one decimal digit. The binary code normally used for each digit is:

0 = 00000000	3 = 00000011
1 = 00000001	4 = 00000100
2 = 00000010	5 = 00000101

6 = 00000110 8 = 00001000

7 = 00000111 9 = 00001001

This system has two advantages: conversion to the true decimal form is trivial, and by using enough words (one for each digit) it is possible to represent numbers of any size. On the other hand, BCD is wasteful of space, and arithmetic is slow. The addition of two BCD digits is governed by the rule:

1. Add the two numbers as if they were (simple) binary quantities.
2. Add the carry from the previous stage (if any).
3. If the result is greater than ten, then:
   (a) subtract ten from the result,
   (b) carry 1 to the next stage.

For example:

```
 0011        / 3   \
 0101  +    (  5 +  )
 ----       |  --   |
 1000        \ 8   /
```

This is correct; but

```
  1000       / 8   \
  1001  r   (  9 +  )
 -----      | ---   |
 10001       \17   /
```

This exceeds ten, so 1010 (ten in binary) is subtracted:

```
 10001
  1010
 -----
   111
```

giving 0111 (seven), and 1 is carried to the next stage.

In an eight-bit system, it is clearly possible to 'pack' two BCD digits into one word. For example, 10010111 in packed BCD means '97'.

Some combinations of bits would imply 'decimal digits' greater than nine, and are not permitted. Thus the eight-bit word '10101011' would be meaningless in packed BCD.

Signed binary numbers

In many applications it is important to be able to use *signed* numbers – numbers that can have negative or positive values. Humans normally use the sign-magnitude notation, where the absolute value of the number is preceded by its sign. This leads to complicated rules for the addition and subtraction of signed numbers, such as:

> To add two signed numbers *a* and *b*:
> - If the signs are the same, add the magnitudes and attach the common sign.
> - If the signs are different, then:
>   if the magnitude of *a* is greater than the magnitude of *b*, subtract *b* from *a* and attach the sign of *a*; *otherwise* subtract *a* from *b* and attach the sign of *b*.

The system used in microprocessors is quite different and much simpler. It is called the 'two's complement' notation.

In the discussion on unsigned binary numbers, it was noted that, where quantities grew too large, carries were lost and the results of addition were wrong. This was represented as a serious drawback; but in the two's complement notation the same feature is actually turned to advantage.

One way of defining a negative number $-n$ is to specify that it is the number that, when added to $+n$, will give zero. In a system with a limited number of digits, every number has its negative. For example:

```
    00000001
    11111111   +
   ----------
 (1)00000000
```

Here the carry is lost from the end, and the sum of the two numbers being added is zero. Since the first number is +1, the second is evidently −1. Similar arguments can be used to show that:

```
11111110 = −2
11111101 = −3
11111100 = −4   etc.
```

With this representation, the addition and subtraction of signed numbers is trivially easy: no account at all need be taken of the sign! For example, +7 + (−4) gives:

```
    00000111
    11111100   +
   ----------
 (1)00000011   = +3
```

The one drawback of this system is its potential ambiguity; does 11111010 mean '250' or '−6'? The answer to this question is fundamentally a matter of convention; but to arrive at a sensible rule we make use of a circle diagram.

Consider a variable with integral (whole number) values. Each value can be written down in a small circle, and the operation of adding 1 can be shown as an arrow. A mathematician would show part of the set of integers as in *Figure 2.1*, it being understood (by the mathematician) that the line of numbers extends to infinity in both directions.

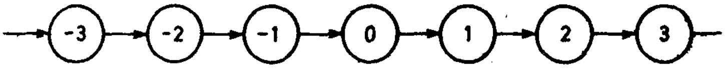

Figure 2.1

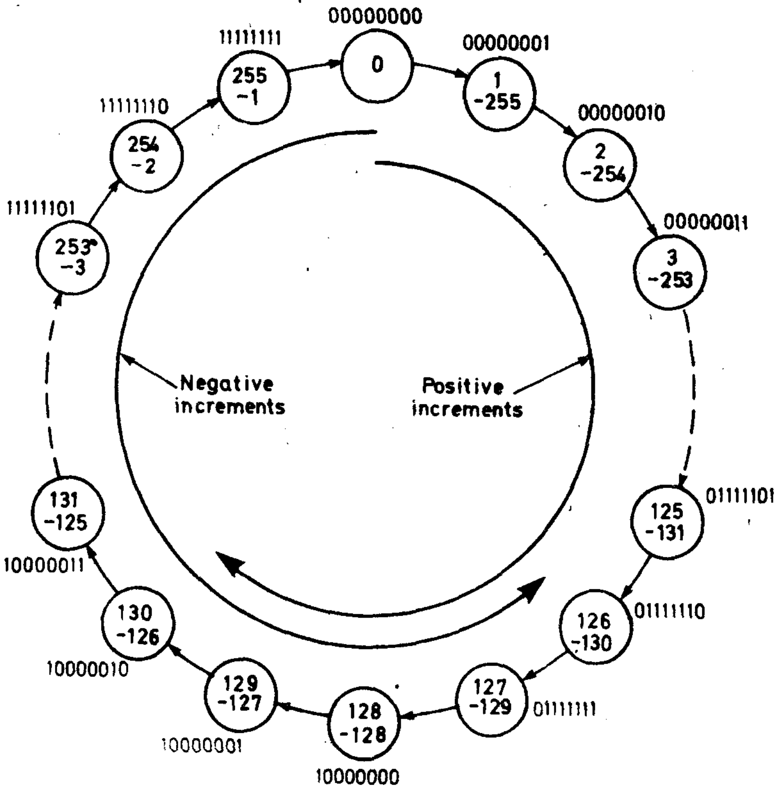

Figure 2.2