MINICOMPUTERS AND MICROPROCESSORS

Martin Healey

Minicomputers and Microprocessors

Martin Healey, PhD, MSc, C.Eng, MIEE

Department of Electrical and Electronic Engineering,
University College, Cardiff.

Hodder and Stoughton

London Sydney Auckland Toronto

ISBN 0 340 20113 4

First printed 1976
Reprinted 1977, 1978

Printed in Great Britain for
Hodder and Stoughton Educational,
a division of Hodder and Stoughton Ltd.,
Mill Road, Dunton Green, Sevenoaks, Kent,
by J. W. Arrowsmith Ltd., Bristol BS3 2NT

Preface

There are a large number of text books on the market covering the broad field of digital computers. As an author, one must justify any extension of this list. It is not sufficient just to create an alternative, direct competitor to established standards. True, time creates obsolescence in books as it does in hardware, so that there is always room for the occasional 'new techniques' text, but this one is largely fundamental, covering basic technology that is well established. This text aims to be different in two ways, firstly it concentrates on the small and medium sized machines, known collectively as minicomputers and the modern LSI offshoot, the microprocessor, and secondly the accent is entirely one of 'How does it work?' rather than 'How do you design it?'

The book is primarily aimed at Engineers and Systems Analysts who wish to know more about the machine, which is, when all said and done, simply a component in a system to them. It is the author's opinion that all Electrical Engineering undergraduates should take a course on computer fundamentals, such is the impact of these machines on all walks of life. Those students studying computer science or computer design, require far more detail than is included here, but a study of minicomputer architecture and the associated software and peripherals will well serve as an introductory text. This concept, which is not commonly accepted in computer teaching if the standard texts are a guide, is all the more relevant nowadays since the drastic fall in the cost of 'stand-alone' minicomputers means that these machines can be made available for 'hands-on' practice, avoiding the problems of sharing a larger general purpose machine.

The microprocessor is strictly a central processor fabricated on one or two LSI chips. This means that smaller, dedicated, digital computers can be tailored to suit a particular requirement. The fundamental principles of operation of microprocessors are the same as those of minicomputers. These machines will be used in industry to replace the

larger systems currently made up of TTL logic chips. This means a drastic about face in the background requirements of the systems engineer. The detailed study of combinatorial and sequential logic will become less important, the need to program a small computer in a low level machine language moving to the fore. It is this problem which leads the author to believe that a broader knowledge of digital computers is required by the next generation of electrical and electronic engineers. It is possible, in the usual conflict for teaching time, that machine code or Assembly Language programming will be considered more important than logic minimisation techniques.

The book is presented in nine chapters. The routine arithmetic, logic, memory and electronic processes are briefly described in appendices, rather than the main text, since, with normal electronic studies, and indeed with the newer maths taught in schools, many readers will be well versed in these topics.

Chapter 1 describes a range of applications of minicomputers and microprocessors, related to larger digital computers. Attention is given to the problem of schematic and conceptual descriptions of the fundamental components used in the make up of a computer.

Chapter 2 explains the basic features of a simple minicomputer, by developing a hypothetical machine. The importance of the relation of the hardware operations that can be implemented to the possible instruction set, constrained by a fixed word length, is stressed.

Chapter 3 contains descriptions of the more refined, but fundamental, features found in modern minicomputers CPUs.

Chapter 4 expands the I/O considerations introduced in Chapter 2, including interrupt structures and block data transfer. The more practical aspects of interfacing are also introduced.

Chapter 5 describes the special features of the microprocessor, concentrating on a typical 8 bit machine. Short descriptions of the peripherals commonly used, including bulk storage devices, are given in Chapter 6.

Chapter 7 is a review of the requirements and availability of software. The scope is extensive but purely descriptive. It is not a programming course.

In attempting to present the facts in a logical progressive fashion, a number of more sophisticated hardware techniques are skipped in the earlier chapters and are grouped together in Chapter 8. In general, the material presented in this chapter covers newer, better methods of doing the more fundamental tasks previously explained.

The final chapter is included rather as a practical warning. In this chapter brief comments are made on the problems of purchasing and installing a computer system. It serves a warning that the purchaser can

get carried away with the technical complexity of digital computer CPUs and forget the more important aspects of the *system*.

As a course in minicomputers or fundamentals of any digital computer, this text can be read sequentially from beginning to end. The appendices are only intended for quick reference or revision; indeed separate textbooks are required to do justice to each topic covered.

The text is organised so that the engineer who requires an explanation of microprocessors only, may read Chapter 2 followed by Chapter 5, filling in any further details as required.

A fairly extensive bibliography, constrained largely to textbooks rather than papers, is included. In particular, however, I would like to acknowledge the 'silent men' who prepare the handbooks for the commercial machines. I have referenced the particular handbooks which I have personally used over the years, but I would like to praise the efforts of all such authors, as much as a user as a writer.

I would like to acknowledge the helpful discussions I have had with David Turtle, David Horrocks and Peter Tomlinson. I also would like to thank Bob Churchhouse for the notes which largely form Section 1.7. Finally I must admit that the inspiration to write this book stemmed from my associations with minicomputer seminars and conferences run by Richard Elliott-Green and Bob Parslow (On-line) and Gerry Cain, Yakup Paker and Peter Morse (Minicom). Finally I would like to acknowledge the help of Barbara and Jeanette in the preparation of the manuscript.

MARTIN HEALEY

Contents

Chapter 1

Digital Computers and Their Applications

1·1 Binary Representation of Information

A digital computer is an electronic programmable calculating machine which works on a binary principle. Electronic circuits are used to represent combinations of binary digits, each of which can be in one of two states, on or off, high or low, open or closed, etc. Logically these two states can be termed true or false but the useful parallel with binary arithmetic makes the terminology 0 or 1 most common; in fact using the 0/1 notation, logical (Boolean) arithmetic can be closely related to normal arithmetic using the binary system. Any reader who is not conversant with binary or logical arithmetic will find the topic well covered in modern elementary text books; Appendix 1 provides a summary of the salient features.

Inside a practical computer the two states are represented by electrical voltages. The actual voltage levels depend upon the technology used in the manufacture of the circuits and vary as new manufacturing techniques are developed. For some time now the TTL (*transistor–transistor logic*) type of circuit has prevailed and logical 0 is represented by a nominal 0 volts and logical 1 by a nominal 3·5 volts. Details of the different types of logic systems are given in Appendix 2.

The more recent MOS LSI (*metal-oxide-silicon large-scale-integration*) circuits work with different voltages, but such is the current hold of TTL that the MOS LSI circuits are being 'tailored' to be compatible at their inputs and outputs with TTL systems. Of course the original concept of computers was thought of long before the electronic age, so that mechanical representation of two state systems was envisaged; indeed punched paper tape is just such a system where the punched hole represents, say, 0 and no hole represents 1. For such a system however a tape reader is needed, a device which converts the hole/no-hole system into TTL compatible voltage levels before such information is useable by the computer.

Each *binary digit* is referred to as a *bit*. To represent a meaningful piece of information, bits must be used in combinations, e.g. to represent all possible decimal numbers from 0 to 63_{10}, that is the binary numbers 0 to 111111_2, requires 6 bits. Differing situations may well require differing combinations of bits, but some standardisation has been introduced, the advantages of which should be clear later. The most common combination is the *8 bit* group, called a *byte*. A byte is often too small a combination to be of use inside a computer, so that each computer, according to its design, has an established fixed *word* length. This means that each piece of information is handled inside the machine as a fixed length binary word. Common word lengths are 8, 12, 16, 18, 24 and 32. The more sophisticated machines will allow more than one word length by allowing groups of bytes, but not any number of bits. The pros and cons of a particular word length will be discussed in the following chapters.

The advantage of using a binary system in an electronic machine lies in the elimination of erroneous interpretation while 'transporting' information from one point of the machine to another. Consider a system in which the numbers from 0 to 10 are represented by subdividing a 5 volt supply, as in figure 1.1. In an equivalent binary system four digits are required as shown in figure 1.2. Note that three digits can represent 0 to 7 and four digits 0 to 15; the fourth digit is required to represent 0 to 10, although some combinations are 'wasted'. Thus the number 8 is represented by the voltage level 4·0 volts in the analogue form and by the four voltages 5, 0, 0 and 0 in the binary or digital form.

Now, in manipulating the information in electronic circuitry, it is likely that the voltage levels will be distorted; if the 4·0 volts falls to 3·75 volts then it is midway between the representation of 8 and 7 and an error has been introduced into the system. In the binary system the 0 and 5 volt levels must be distorted to 2·5 volts before ambiguity occurs. In other words errors occur in the analogue system for voltage level

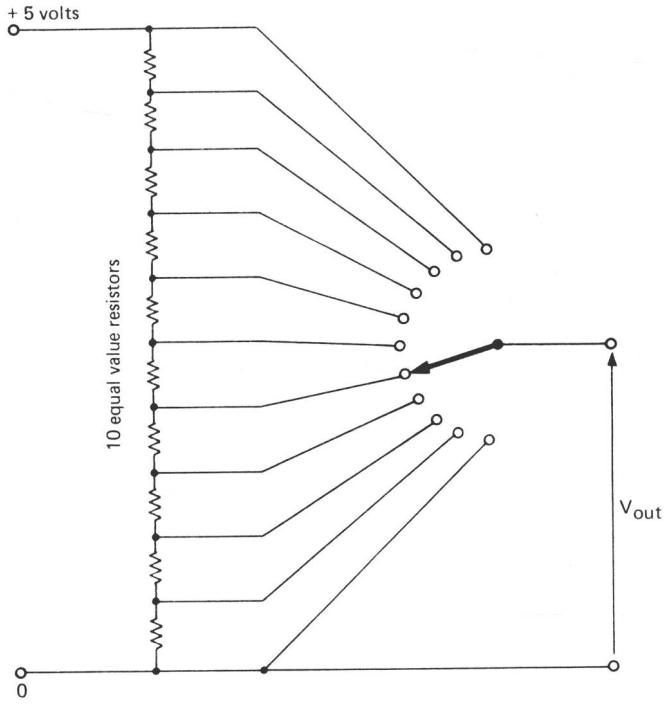

Fig. 1·1 Analogue representation of the integer numbers 0 to 10.

changes equal to about 5% of the 5 volt supply while the digital system is tolerant to changes of around 50%.

The resolution of the analogue system can be improved by subdividing each interval and that of the digital system by extending the number of digits below the LSB (*least significant bit*) as shown in figures 1.3 and 1.4. The digital system shown in figure 1.4, in which each digit

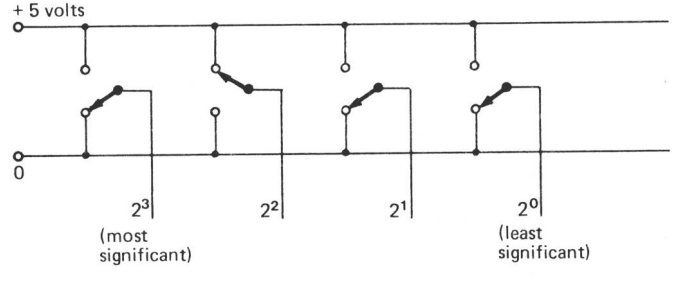

Fig. 1·2 Binary representation of the integer numbers 0 to 10.

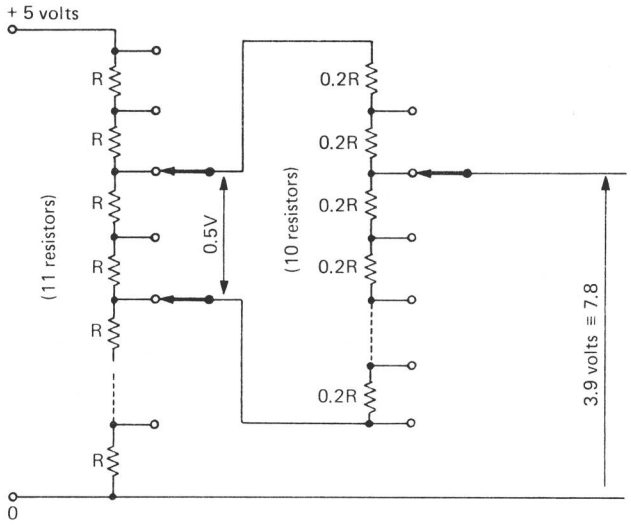

Fig. 1·3 Two decade analogue representation. Note that there are 11 resistors in the primary chain. The two Rs in parallel with the ten 0·2 R resistors is equivalent to one R as in figure 1.1.

of a decimal number is coded as four binary digits is called *binary coded decimal* (BCD). In practice most digital computers use a pure binary number code, which is more efficient than BCD, as in BCD only ten of the sixteen possible combinations of each 4 bits are used. These number systems are discussed in Appendix 1. The resolution of both systems (figures 1.3 and 1.4) can be increased by adding further sections, however, accuracy is a different problem to resolution. It is no use adding further decades to figure 1.3 if the accuracy of the most

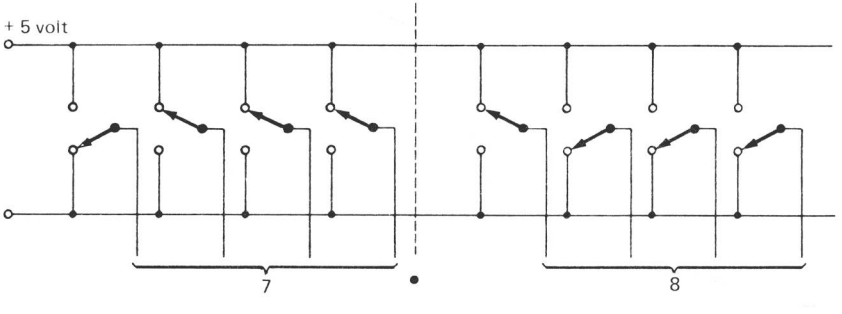

Fig. 1·4 Binary representation of two decade decimal systems using four binary digits for each decade.

significant digit cannot match that of the least significant. On the other hand the accuracy of the binary system is directly related to the resolution, since it is fair to assume that the voltage level changes will never approach the 50% mark at which errors will occur. Thus the accuracy of a digital system is directly related to the number of bits allowed in the word used inside the computer. For those readers conversant with FORTRAN, the Double Precision statement is a direct application of this principle in which the number of bits used to represent the number is doubled over the normal representation, with the attendant increase in accuracy.

In summary a digital computer uses a very simple system to represent information, the binary system, which gives extraordinarily high integrity. The penalty paid for using the binary system is the large numer of digits that are required, e.g. 4 bits to represent one decimal digit. A digital computer then is a proliferation of electronic two-state circuits.

1·2 Schematic Concept of a Digital Computer

All digital computers are capable of executing a limited number of defined operations. Typical operations are to move information from one part of the computer to another, i.e. to and from the memory and the CPU (*central processing unit*), or to take data into the computer from the input and to send data to the output. Other operations involve arithmetical and logical functions, such as add two numbers together. The computer is ordered to execute a particular operation by giving it an *instruction*; each instruction is coded as a combination of binary digits. The computer can only execute one of these instructions at a time; to perform a required task a sequence of these simple instructions must be executed. It is worth while stressing at this point just how simple the instructions actually are; on many machines even the simple operation of multiplying two numbers together is performed by repeated 'shift and add' instructions. However the speed at which these simple instructions can be performed is beyond the comprehension of the human being. For example, a machine was made to move 8000 pieces of data from one memory location to another and to add each number to an accumulator. The program was started by depressing a switch and completion indicated by causing a light bulb to be lit. Despite the 8000 data moves and 8000 additions, all done sequentially, it appears to the human operator that the bulb is lit directly by closing the switch. A high speed idiot is a most appropriate description of a digital computer!

When the computer is required to perform another task it must be asked to execute another sequence of its own simple instructions. Thus, since each and every task set by the user of the machine must be converted into a string of simple machine instructions, the same machine can be made to perform any such task without any changes to its electronic components. A sequential list of the instructions to be performed is called a *Machine Code* or *Object Program*; special languages such as FORTRAN or COBOL have been developed to simplify the task of creating the program. Each new problem posed by the user is formulated by writing a new program.

Now each instruction could be fed to the machine via an operator's console, one at a time. Such an approach however, would slow the machine down to the speed of the human operator; use must be made of the incredible speed of the electronic circuitry. Thus the developed program is entered into a store inside the machine, labelled *program memory* in figure 1.5. The first instruction is then transferred from the

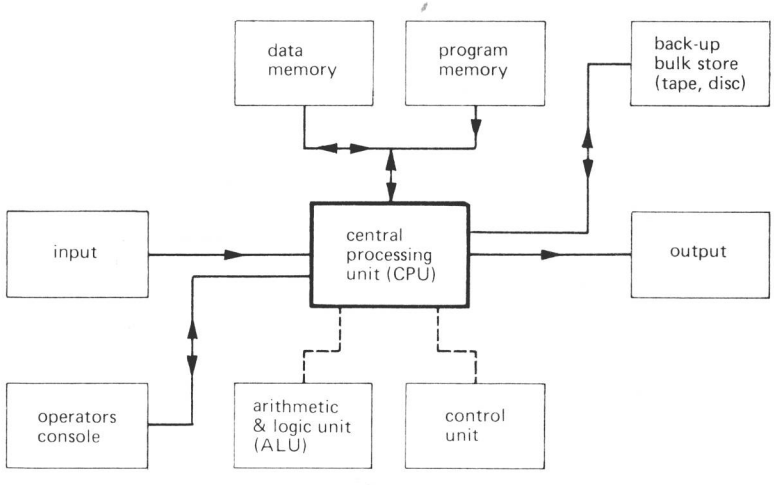

Fig. 1·5 Schematic layout of a digital computer.

memory to the CPU(this only takes around 1 microsecond) where it is 'decoded' by the control unit and executed. When each instruction is finished the next instruction is 'fetched' from the program memory and executed. Note that in executing an instruction, data may be involved and this is made rapidly available by a similar store, labelled *data memory* in figure 1.5. In a clever system the next program is 'loaded' into another part of the memory while the first program is executing, avoiding delays. This 'stored-program' concept is funda-

mental to the operation of a digital computer; it is the ability to use the same machine for a variety of problems simply by reprogramming that gives the machine its vital versatility.

The physical components of the computer are termed *hardware*, while the programs that control the machine are termed *software*.

Each instruction is simply a binary word; a particular combination of zeros and ones is decoded by the control unit in the CPU. In practice this means that a complex network of logical electronic elements is switched in a pattern determined by the bits in the instruction. A few simple examples are given in Appendix 2, but there are many textbooks available on the design of digital systems. While each instruction does not necessarily require the same number of bits to define the required action, the organisation of the program memory dictates that all instructions are coded into fixed length words, some bits of which may be redundant for a particular instruction. There are many considerations to be taken into account in arriving at a choice of word length (the choice is the machine designers, not the users) as will become clear in later chapters.

Now both the data and the instructions are stored and manipulated inside the computer as fixed length combinations of bits. The memories are 'word orientated' that is the instructions are located in the memory by individually addressing each word – the whole word is fetched down into the CPU, never particular bits of the word* In practice all machines above the programmable calculator level use the same basic word length for both data and instructions so that a common memory can be employed; in this way jobs with small programs and large amounts of data and jobs with large programs and small amounts of data can both be accommodated with a minimum total memory requirement. The program is loaded into one area of memory and data into another. The CPU is initialised by being given the address of the location of the first instruction and proceeds from there by the logical progression of the program, referring to data in other, specified locations. It must be stressed that there is no physical difference between an instruction and a piece of data; both are similar length binary words. If for any erroneous reason (usually an incorrectly written program) a piece of data is fetched into the CPU when an instruction is expected, then the control section will attempt to decode and execute it, with some very puzzling results!

It has already been noted that the memory is addressed as fixed length words and that the contents of any location, either an instruction or data, must be rapidly available on demand by the CPU. Since the

* Some machines are byte orientated, where memory is split into individually addressed 8 bit bytes. A 16 bit instruction will automatically be fetched as 2 bytes in one operation.

memory locations may be referenced in any order – a simple program will work through its instructions from consecutive locations, but a GO TO in the program will alter the sequence – it must be possible to access *any* one location in memory as quickly as *any* other. Such a memory is said to be a *random access memory* (RAM); for the most part this is magnetic core store, with LSI semi-conductor memory growing in application. RAM should be contrasted with a sequentially accessible memory such as magnetic tape; used as the main memory on a computer, magnetic tape would be forever spooling back and fore to locate specific words of information. RAM, normally referred to simply as memory or even more loosely as core, is expensive and limited in size by the word length; the reasons for this will be explained in the next chapter. Thus magnetic tape and magnetic disc are used as back-up or bulk stores; a program would be copied from the back-up store to memory as a high-speed block transfer and then executed from memory.

The number of basic instructions that the machine is capable of executing is directly related to the sophistication of the electronic logic circuitry in the CPU. The main feature is the *arithmetic and logic unit* (ALU) which actually performs the desired operations on the data. Associated with the ALU are a number of storage registers, ranging from one to say sixteen, which act as a 'scratch-pad' during arithmetic processing. These registers store data similar to specific memory locations, but are high speed TTL registers, the data from which can be accessed in tens of nanoseconds* rather than the microseconds of core memory. In fact the fixed word length must put an upper limit on the number of types of instruction that can be coded; this often sets the limit on the complexity of the CPU rather than the actual circuit design. It is of no use providing extra functions in the electronics if these extra functions cannot be 'dialled-up' by the instruction!

A word of information, say 16 bits as an example, can be transferred inside the machine, e.g. moved from memory to the ALU, in one of two modes, either *serial* or *parallel*. Serial mode means that the word is transferred one bit at a time along a single path in 16 time intervals, the 16 bits being reformed as one word before being used. Parallel mode means that 16 paths are provided and the whole word is transmitted in one time interval. Clearly serial mode is cheaper but slower than parallel mode. All modern computers; in which hardware costs have become less important, thanks to integrated circuit technology; work exclusively in parallel mode, increased speed being the prime objective. One commonly encountered example of serial transmission of

* A nanosecond is a thousandth of a microsecond, that is there are 10^9 nanoseconds per second.