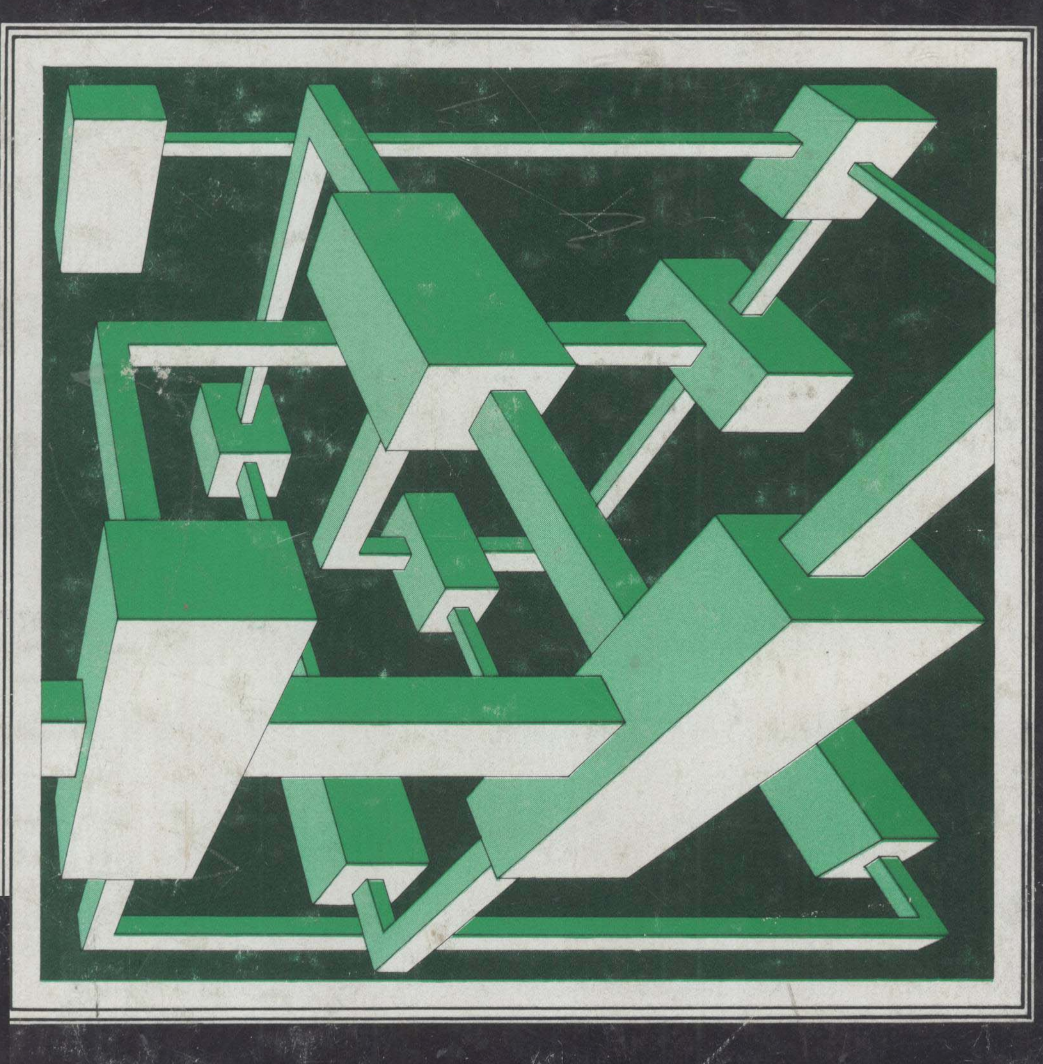
Ellis Horwood · Publishers
COMPUTERS AND THEIR APPLICATIONS 10

Fundamentals of Computer Logic

David Hutchison



FUNDAMENTALS OF COMPUTER LOGIC

FUNDAMENTALS OF COMPUTER LOGIC

DAVID HUTCHISON, B.Sc., M.Tech.
Department of Computer Science
University of Strathclyde

ELLIS HORWOOD LIMITED
Publishers · Chichester
Halsted Press: a division of
JOHN WILEY & SONS
New York · Chichester · Brisbane · Toronto

First published in 1981 by

ELLIS HORWOOD LIMITED
Market Cross House, Cooper Street, Chichester, West Sussex, PO19 1EB, England

The publisher's colophon is reproduced from James Gillison's drawing of the ancient Market Cross, Chichester.

Distributors:

Australia, New Zealand, South-east Asia:
Jacaranda-Wiley Ltd., Jacaranda Press,
JOHN WILEY & SONS INC.,
G.P.O. Box 859, Brisbane, Queensland 40001, Australia

Canada:
JOHN WILEY & SONS CANADA LIMITED
22 Worcester Road, Rexdale, Ontario, Canada.

Europe, Africa:
JOHN WILEY & SONS LIMITED
Baffins Lane, Chichester, West Sussex, England.

North and South America and the rest of the world:
Halsted Press: a division of
JOHN WILEY & SONS
605 Third Avenue, New York, N.Y. 10016, U.S.A.

British Library Cataloguing in Publication Data
Hutchison, David
Fundamentals of computer logic. –
(Ellis Horwood series in computers and their applications).
1. Electronic digital computers – Circuits
2. Logic circuits
I. Title
621.3819'5835 TK7888.4 80–42028

ISBN 0–85312–258–X (Ellis Horwood Ltd., Publishers – Library Edn.)
ISBN 0–85312–305–5 (Ellis Horwood Ltd., Publishers – Student Edn.)
ISBN 0–470–27117–5 (Halsted Press)

Typeset in Press Roman by Ellis Horwood Ltd.
Printed in Great Britain by Butler and Tanner Ltd., Frome, Somerset.

Table of Contents

Author's Preface

This book describes the structure of computers through a study of their underlying logic circuits. The approach used is to present the material in a bottom-up way, starting with a description of the basic building blocks of logic circuits and proceeding in a series of layers to build a picture of how computers are designed and constructed. Although the material is principally concerned with computer hardware, some attention is paid to the mutual dependence of hardware and software requirements in computer design.

While the book is aimed primarily at first- and second-year undergraduate students at Universities and Polytechnics – in computer science, microprocessor studies and related engineering subjects – the approach is intended also to benefit those with an interest in computers from a mainly hardware point of view. Those who wish to learn about logic building blocks and their use in designing logic circuits, but not necessarily their application in computers, are also catered for since these topics – the core of the book – are essentially self-standing. A little background knowledge is required: a familiarity with binary numbers and an acquaintance with the notion of programming. The reader who has attended a short course on computer appreciation will be well prepared.

Chapter 1 sets the scene of the book by presenting a brief history of computers and outlining the structural layers of a modern computer. In Chapter 2 the building blocks of logic are introduced from both an abstract and a physical point of view: the theory of Boolean algebra and the implementation tools of integrated circuits ('chips') are brought together. Chapter 3 classifies logic circuits into the combinational and sequential varieties, and describes techniques for designing both types of circuit, with worked examples in each case. Further worked design examples are presented in Chapter 4, along with other aspects of the uses of logic circuits in practice. Several logic circuits used in computers are introduced before Chapter 5, which illustrates how the circuits fit together to implement the major functional units in a computer. Particular attention is paid to the design of arithmetic and control units. The last Chapter, 6, discusses the interdependence of hardware and software in computers and ends by commenting on the implications for computer design of new hardware technologies and advances in software techniques.

An Appendix contains a sample of typical literature available from a semiconductor manufacturer, describing some of the integrated circuits with which logic circuits, and computers, can be constructed. In the text the importance of referring to, and understanding, such data sheets is emphasised. The Reading List contains chapter-by-chapter recommendations for further reading. No references are included in the text but the annotations with each title in the List direct the reader to suitable sources for specific topics.

Students using this book as a course text will greatly benefit from a short practical course which illustrates the use of integrated circuits in logic design and implementation. Both the practical work and the choice of any design problems for students to tackle are best left to the discretion, and ingenuity, of the course lecturer.

Thanks are due to the second-year students of Computer Science at the University of Strathclyde who have helped, wittingly or otherwise, to evolve the approach used in this book by their participation in the course on Logic Design during the past three years. My thanks in general to my colleagues in the Department of Computer Science, and in particular to Miss Agnes Wisley and Mrs Margaret McDougall for their help in typing the manuscript. Brian Meek (the Series Editor) and Michael Horwood have helped give this book shape and direction and I am most grateful to them. Lastly, the book would never have been written without the help and support of two people, Ian Campbell and Ruth Hutchison.

Glasgow, August 1980

CHAPTER 1

The structure of computers

1.1 INTRODUCTION

Computers have two major ingredients: hardware and software. *Hardware* is the collective term used to describe the physical units of the computer – the processor, memory and peripheral devices, including all mechanical and electronic components. *Software,* on the other hand, refers in general to the programs which cause the computer hardware to obey specific sequences of instructions. The physical realisation of software is either a set of instructions, written in a particular language, on a piece of paper or a pattern of binary digits (*bits*) in the memory hardware of a computer.

Very often a distinction is made between system software and applications software. *System software,* sometimes referred to confusingly as simply software, is a set of programs written by the manufacturer or supplier of the hardware (or in some cases by the users themselves). This software consists of an *operating system* which controls the basic functions of the hardware, and a set of utility programs such as language translators, editors and debugging aids. Depending on the intended application area of the computer its operating system may provide more or less extensive facilities: in a large multi-access machine the operating system may have to service the widely-different needs of its many concurrent users and support a complex file system held on magnetic discs and tape. At the opposite extreme a small single-user computer may require an operating system which simply allows single programs to be loaded and started, and memory locations to be inspected and their contents altered. All computers require some system software which enables users to operate them easily. The extent to which utility programs are provided also depends on how the computer is used: systems on which programs are constantly being developed would typically have available a variety of high-level language compilers and a file editor, whereas dedicated systems in which the programs are fixed and proven may provide no such utilities along with the operating system.

Applications software, usually written by users or software houses, tailors a computer system (already provided with system software) to a particular task or application. Typically applications software is written in a high-level programming language such as COBOL, FORTRAN or Pascal, and in this way the hardware features of the computer are hidden from the programmer. The compiler translates the user program into machine-code instructions, the form in which the computer can understand the programmer's intentions. Interaction with the outside world, via the peripheral devices, is handled on behalf of the programmer by the operating system; the programmer simply indicates the need for input or output by means of a high-level statement such as read(X) or print(X) in the program text. In special circumstances, such as cases in which the speed of operation of programs is critical, applications may be programmed in assembly language, a symbolic form of the machine-code which computers understand. The central part, or kernel, of an operating system is often written in this machine-oriented form. Applications written in assembly language also rely on the presence of an operating system to control the resources of the computer hardware and to provide a machine-independent user interface.

In very general terms the structure of a computer system can be illustrated as in Fig. 1.1. This shows the three components introduced above -- hardware, system software, and applications software – ordered in a hierarchy or a set of *layers.* The computer hardware is the lowest, or most basic, of the layers, while

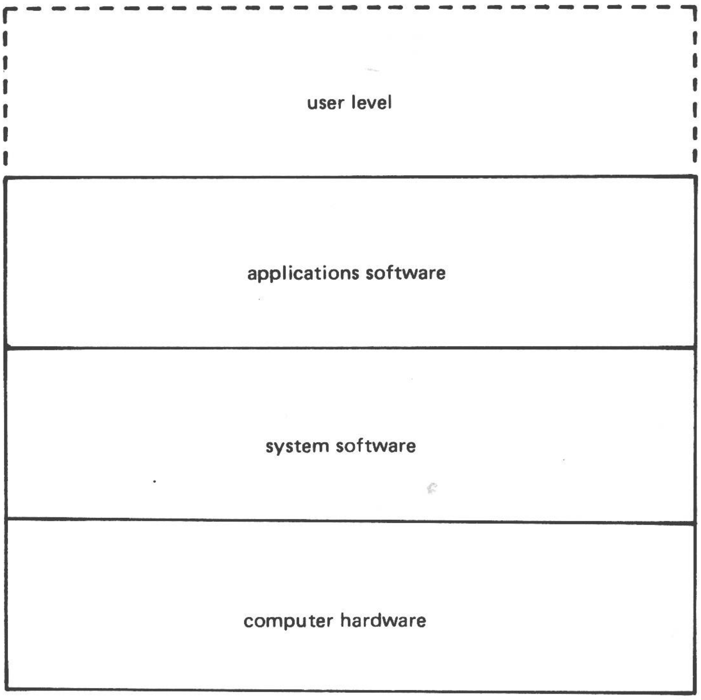


Fig. 1.1 – The broad layers of a computer system.

the highest layer in the computer hierarchy is the user level. Strictly speaking the user level is not a part of the computer as such; this is indicated by the dashed lines in the diagram.

Each layer provides a set of facilities for the one above. The architecture of the computer hardware determines the nature of the machine features which the system software has to handle; the type of use of the computer system influences the facilities which the system software offers to the applications programmer; and lastly, but very importantly, the applications level should provide a friendly set of facilities to the users. The boundary between one layer and the next is termed an *interface*: as we go from the hardware to the user level the interface facilities offered are more and more high-level, in other words further from the low-level machine features and somewhat closer to a form which people can easily understand. The *man-machine interface,* as the highest-level interface is called, is increasingly important with computer systems finding their way into every office and factory, where non-expert users are called upon to operate them.

* * *

The scope of this book falls well short of the man-machine interface, and short also of applications software. It deals with computer hardware, and to some extent with the interface between hardware and system software. More specifically the book is about *computer logic,* the electronic rather than the mechanical aspects of computer hardware – the logic circuits as opposed to the moving parts such as magnetic discs or line printers. The remainder of this chapter outlines the context in which the main material of the text is set.

1.2 COMPUTER LOGIC

To explain the meaning of computer logic let us look a little more deeply into the hardware layer of the computer system. It should be borne in mind that in this book we are dealing exclusively with *digital* as opposed to *analogue* computers. Analogue computers represent information in the form of continuously varying voltages or currents and are used in special-purpose applications like the design of automobile suspension systems where the physical system can be modelled or simulated by the computer.

A brief history

Digital computers represent information in discrete binary (two-valued) form and have evolved from the calculating machines of the last century, including Charles Babbage's design for an Analytical Engine (1837) and Hollerith's Electric Tabulating System (1889). The first electronic computer was the ENIAC (1946), inspired by a memorandum of J. W. Mauchly in 1942 within the Moore School of Electrical Engineering at the University of Pennsylvania. The ENIAC (Electronic Numerical Integrator and Calculator) took three years to build and was large-scale in every way. It contained some 19,000 valves, weighed 30 tons and consumed 200 kilowatts of electricity. It was also extremely fast by the standards of the day, being able to multiply two 10-digit decimal numbers in 3 milliseconds. However, the effort of programming the ENIAC was such as to discourage its use for any other than extensive computational problems, since it had to be programmed manually by plugging and unplugging sets of connecting wires. Data could be entered using a punched card reader, and results output on punched cards or on an electric typewriter. A large team was responsible for the design and construction of the ENIAC, most notably J. P. Eckert and J. W. Mauchly who in 1947 jointly founded a company to produce computers commercially. One of their first products was called the UNIVAC (Universal Autommatic Computer). Later their company became the UNIVAC division of the Sperry-Rand Corporation, which along with IBM began selling computers successfully in the early 1950s.

Another member of the Moore School team, John von Neumann (1903-1957), is credited with the idea now seen as the final step in the development of the general-purpose computer. This is the idea of a *stored-program* machine in which program and data share a common memory. The most important consequence is that programming is made very much easier; thus the computer possesses a generalised instruction set, fixed into its hardware, and the program – consisting of a sequence of appropriately chosen instructions – can be read in via a punched-card reader in the same way as the data. An additional consequence, one that has had less lasting significance, is that programs can be made to modify their own instructions.

There is evidence to suggest that others before von Neumann had the notion of a stored-program computer, notably Konrad Zuse (in his 1936 paper), who produced in Germany in the 1930s a series of mechanical and electro-mechanical computers called Z1, Z2 and Z3; A. M. Turing, whose 1936 abstract model of a computer – called a Turing machine – formed the basis for much of the present-day knowledge of the theory of computation; and even Charles Babbage, although in his case perhaps the suggestion is closer to speculation than in the others. However, it is certain that von Neumann's draft report on the EDVAC (Electronic Discrete Variable Computer) written in 1945 contains the earliest documented presentation of the stored-program idea. The EDVAC was the successor to ENIAC and contained several design changes which originated during the ENIAC project. The main differences were that it was a binary rather than a decimal machine and that it had a much larger memory: 1K (or 1024) 40-bit words of mercury delay-line main store, plus a secondary, slower magnetic store 20K words in size. This machine did not become operational until 1951.

Meantime a report written for the U.S. Army Ordnance Department in 1946 by Burks, Goldstine and von Neumann proposed a methodology for designing computers. This report was the first of a series which led to yet another machine called the IAS. In effect its proposals summarise the characteristics of *first-generation* digital computers. Burks, Goldstine and von Neumann suggest the following features:

- main units: control
  arithmetic
  memory
  input/output communication
- program and data sharing the memory
- binary internal forms
- a synchronous clock system
- the use of subroutines
- possible adder hardware, but multiplier software
- the use of an accumulator register
- parallel operation for memory accessing
- diagnostic/single-step provision.

All these features are to be found in the majority of present-day computers. They characterise what has become known as the *von Neumann machine.* A schematic diagram of such a machine is shown in Fig. 1.2. This gives us an outline description of the typical hardware structure of a computer.

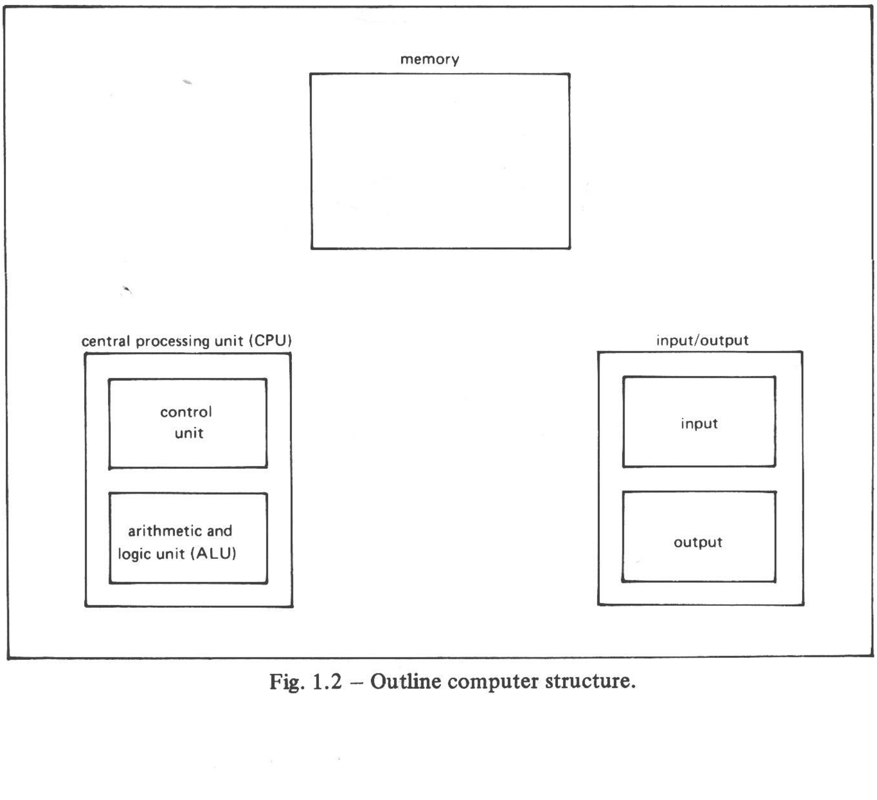

memory
central processing unit (CPU)
control unit
arithmetic and logic unit (ALU)
input/output
input
output

Fig. 1.2 – Outline computer structure.

There are three main parts:

(1) the memory, in which program and data co-reside.

(2) the central processing unit (CPU) which in turn has two components – the arithmetic and logic unit (ALU) in which all calculations are performed, the results being held in the associated accumulator register; the control unit, whose job it is to fetch and obey program instructions from the memory, and to co-ordinate the activities of the other units.

(3) input and output units, which respectively read information into the computer and print it out.

In the late 1940s and early 1950s many computers were built. In Britain there was notable work at Manchester and at Cambridge. Probably the first working stored-program computer was a small experimental machine built at Manchester University in 1948 by F. C. Williams and T. Kilburn. The range of machines produced at Manchester culminated (much later, around 1961) in the famous ATLAS computer which had a *one-level store,* the forerunner of the present-day virtual memory systems in which the (fast) main memory and the (slow) magnetic backing store are seen by the programmer as effectively a single large memory. At Cambridge M. V. Wilkes and others designed and built the EDSAC (Electronic Delay Storage Automatic Calculator), a machine modelled on the lines of the EDVAC. It was completed in 1949 and had a fixed program which could nowadays be described as an assembler and loader, an early contribution to programming aids.

Following on from the early days of computer design, from about the mid 1950s, we can identify the emergence of the *second-generation* machine. These were characterised by improvements in both hardware and programmability. Perhaps the most important hardware innovation was the replacement of the vacuum tube by the *transistor,* a semiconductor device invented at Bell Telephone Labs. by J. Bardeen and W. H. Brattain. This permitted computers to be built which were smaller and more reliable, and consumed less power. Not until the surface-barrier transistor was developed in 1954 by Philco did the operating speed of computers improve significantly, however. Thereafter the development of discrete transistor technology continued with the introduction of *logic families* called direct-coupled transistor logic (DCTL), diode-transistor logic (DTL), and resistor-transistor logic (RTL). These represented efforts continually being made by the manufacturers to improve the performance of the basic elements of computer hardware. A key aim was to reduce the cost of the elements, while nevertheless also improving their speed and reliability.

Alongside the new technology of transistors in characterising second-generation machines stands the introduction of *high-level programming languages* as a major aid to speeding up the process of computer programming. The intention of the high-level language was to permit programming to be problem-oriented and machine-independent. A *compiler* (or translator program) converts the high-level language programs into machine-code specific to the computer which will run the program. FORTRAN (Formula Translation) was the first widely-used high-level programming language. It was developed by a group at IBM under the direction of John Backus between 1954 and 1957. COBOL (Common Business Oriented Language) followed in 1959, intended mainly for business applications, in contrast to Fortran which was designed specifically to be used in scientific work. ALGOL (Algorithmic Language), specified in 1960 and revised in 1962 by an international committee including, amongst others, John Backus and Peter Naur, was another important language designed during the second generation of computers. Other languages, now obsolete, were being designed and compilers implemented. The first system software was now beginning to be produced by computer manufacturers and was supplied as part of a package along with the computers themselves. This early system software tended to consist of compilers and rudimentary operating systems.

Apart from changes in technology and software, the architecture and logical design of computers were developing too. Second-generation machines tended to have a floating-point arithmetic unit; index registers and indirect addressing became standard; with magnetic core main memory the design of the CPU tended to be strongly influenced by the timing of memory accesses; and synchronous operations (that is linked to a common timing source) became very widely used.

Previously, asynchronous operations dominated: in this scheme the component parts of an operation (some slow, others fast) were allowed to proceed at their own pace, and job completion signals indicated that the next phase could begin. In computers of the second generation onwards the cycle of events within the machine was controlled by a central clock, both CPU and memory operations being synchronised from its timing pulses. The use of index registers was pioneered by the Manchester University team: these fast-access storage locations in the CPU allowed modification of memory addresses and were particularly intended to help improve the efficiency of machine-code programs produced by compilers. Together with indirect addressing, the presence of index registers extended the memory addressing capabilities of computers in line with the requirements dictated by high-level languages. Experience with software was influencing the design of computers considerably. Applications, too, influenced their design. The requirement for very powerful computational facilities was satisfied by the widespread use of floating-point arithmetic units.

Details apart, the general structure of computers as specified by von Neumann and his colleagues was still the same – the three main parts, CPU, memory and input/output – and has remained so ever since. Moving on beyond 1960 the trend was still to improve the speed, size (and inevitably cost) and programmability of computers.

As with the previous generation, *third-generation* computers are most strongly characterised by a technological innovation, in this case the use of *integrated circuits* (ICs). Instead of the former discrete components, the semiconductor industry began producing monolithic ICs on which the equivalent of several transistors were fabricated. This newest advance took place in the early 1960s, with Fairchild and Texas Instruments well to the fore amongst the semiconductor manufacturers. It was, however, Sylvania which first produced the logic family which has remained popularly in use up to the present day: transistor-transistor logic (TTL). With higher packing density of components and improved switching speeds, ICs enabled computers (and other digital logic devices) to be much smaller and faster. It is alternatively suggested that third-generation machines are mainly characterised, from the programmer's point of view, by multiprogramming operating systems based on large capacity magnetic drum and disc stores. Certainly all of the major computer manufacturers set out to implement such operating systems, although it cannot be claimed that many had success until much later in the 1960s. One of the most successful third-generation machines was IBM's System/360, which was available in a variety of different configurations to meet the needs of the individual customer. These machines, in common with the majority being produced at the time, were very large, powerful *mainframe computers*.

About the middle of the 1960s a somewhat different type of machine began to appear on the market. This was the *minicomputer*, characterised by short word lengths (of some 12 to 24 bits) and modest hardware and software facilities. These machines were built to satisfy a new, but soon growing demand for dedicated computers in industrial applications. Digital Equipment Corporation (DEC) was one of the first manufacturers, with its PDP series, to sell minicomputers.

Although more powerful computers continued to be designed, the trend towards smaller machines accelerated as whole new applications areas in industry and commerce revealed themselves. This trend was helped along considerably by the increasing performance/cost ratio of integrated circuit technology. In 1964 Texas Instruments introduced a standard TTL product line called semiconductor network (SN) series 54. Although this was intended primarily for the military market, Texas Instruments soon produced a lower-cost, lower-specification version called series 74. This logic family originally packaged up to about twelve transistor equivalents on one IC: this *level of integration* is called small-scale integration (SSI). In 1969 medium-scale integration (MSI) was introduced, packing from twelve up to a hundred transistors onto an IC.

Large-scale integration (LSI) soon enabled upwards of a hundred transistors to be packaged together on one monolithic structure. With this level of integration manufacturers saw that they could produce an IC containing enough logic to implement a small CPU. In 1971 Intel brought the first *microprocessor* into the marketplace, the 4-bit 4004. Soon 8-bit microprocessors, notably Intel's 8080 and the Motorola 6800, became very widely used products.

The increasing levels of integration and the lowering of IC component