Digital Networks and Computer Systems

Digital Networks and Computer Systems

TAYLOR L. BOOTH

Professor of Electrical Engineering
Computer Science Group
Department of Electrical Engineering
University of Connecticut

JOHN WILEY AND SONS, INC.
New York • London • Sydney • Toronto

Library of Congress Catalogue Card Number: 71-138906

ISBN 0-471-08840-4

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Preface

During the past few years a major shift has occurred in the field of digital systems and computers. Integrated circuit technology has so sharply reduced the price of both digital computers and basic logic modules that many of the tasks that were traditionally performed by analogue circuits and systems are now carried out by using digital techniques. As a result, many engineers and scientists have found it necessary to understand the basic operation of digital systems and how these systems can be designed if they are to carry out particular information-processing tasks associated with their work.

This trend has resulted in the need for an introductory undergraduate course in the digital-system area designed to provide a unified overview of the inter-relationships between digital system design, computer organization, and machine-language-level programming techniques. Unfortunately most current texts treat each one of these topics as a separate subject. Although this approach is desirable (and even necessary) for books that are to be used in the more advanced computer engineering and computer science courses, a new approach is needed for introductory courses. That is the goal of this book.

This book provides the reader with an integrated overview of various classes of digital information-processing systems and the inter-relationship between the hardware and software techniques that can be used to solve a particular information-processing problem. The unifying theme throughout the book is the concept that the steps involved in an information processing task are representable by an algorithm and that the task of a designer is to choose the best techniques to use in executing each step of the algorithm. In some cases it is obvious that these tasks can be carried out by a simple digital network while, at other times, a complex computer program is required. However, there is an ever-increasing gray area between these two approaches in which many different alternatives must be considered before the best solution can be identified. By giving the student a view of the interdependencies of logic design, digital-system design, and machine-level programming, it is possible to provide him, early in his program, with an appreciation of how all these different areas of computer technology interact.

At the University of Connecticut this book is used in the first professional level computer science course. Since the only prerequisite to this course is an introductory programming course, many students from areas such as mathematics, statistics, the physical sciences, the life sciences, engineering, and students planning on majoring in computer science take this course. This serves the dual purpose of preparing students for advanced study in the computer science area and of giving other students an overview of digital networks and computers beyond that presented in the introductory computing course.

All electrical engineering and computer science majors take this course as a required course. Because of scheduling considerations, most of these students take the course during the first semester of their junior year. However, many sophomores have completed this course without difficulty and future changes in the curriculum are planned that will allow this course to be taken either during the sophomore or junior year. In fact, there is no reason why this course could not be taken by freshman students, since there is no specific mathematical background required of the student other than an understanding of high school level mathematics.

At other schools this book is suitable for an introductory digital systems course such as envisioned by the COSINE Committee of the Commission on Education of the National Academy of Engineering or for courses I3 or I6 of the ACM Curriculum 68 (Communications of the ACM March 1968, pp. 151–197).

The sequence in which the material is presented provides for an orderly and logical transition from the basic ideas of representing digital information and performing basic logical operations through the idea of complex information processing systems and programs. Chapter I gives a brief overview of the various topics discussed in the book and their interrelationships. Chapters II and III present a discussion of the techniques that can be used to represent and operate upon information in digital form. The material in Chapters IV, V, and VI provide an introduction to basic switching theory and combinational logic network design. The main concepts of switching theory are presented in a straightforward manner without excessive formalism. This material also illustrates many of the standard logic circuits encountered in digital systems.

Chapters VII, VIII, and IX deal with the idea of digital networks with memory. Several of the basic memory elements are first discussed and then the idea of a synchronous sequential network is introduced. No attempt is made to treat asynchronous networks. Chapter IX is particularly important since it shows how the simple digital networks treated in the earlier chapters can be combined to form complex digital systems.

Chapter X introduces the general ideas behind the operation of stored program digital computers. In particular, a special simulated educational computer, called SEDCOM, is introduced to illustrate these ideas. SEDCOM is then used in Chapter XI to illustrate the idea of machine-language programming and the various programming techniques that can be used to carry out different types of information processing tasks on a small computer. Chapters XII and XIII then discuss the general structure of assembler- and procedure-oriented languages and the translator programs that can be used to transfer source-language programs into object-language programs.

At the University of Connecticut we cover the first twelve chapters in detail and briefly discuss the material in Chapter XIII as time permits at the end of the semester. Dr. Bernard Lovell of our faculty has developed a program to simulate SEDCOM on our Computer Center's IBM 360/65. We therefore require our students actually to write and run a number of the home problems in Chapters X through XIII on this simulated computer. Students can also use special logic breadboards in our digital system laboratory to obtain additional insight into the operation of digital networks.

In order to provide an aid for the student and to help the independent reader, a number of simple exercises are included at the end of each section to illustrate the material of that section. The answers to many of these exercises are included in Appendix I. Several home problems are included at the end of each chapter. These problems are extensive in nature. They extend the material contained in the chapter and start the student thinking about one or more new concepts that will be discussed in one of the following chapters. The references at the end of the chapter guide the reader who is interested in the further exploration of a given area. A Teacher's Manual is available from the publisher upon request for those instructors who adopt the text for classroom use.

I am indebted to many people who have helped in the development of this book. The following persons read the manuscript at various stages of its development and offered a number of very helpful suggestions: William Hammond of Bradley University, Frederick Hill and Gerald Peterson of the University of Arizona, David Evans of the University of Utah, Alan Marcovitz of Florida Atlantic University, James Pugsley of the University of Maryland, Donald Dietmeyer of the University of Wisconsin, Donald Epley of the University of Iowa, Chester Carroll of Auburn University, Gale Miner of Brigham Young University, Stanley Altman of the State University of New York at Stony Brook, C. L. Coates and Garnot Metze of the University of Illinois, and A. J. Pennington of the Drexel Institute of Technology. In addition to the assistance of the aforementioned, I was very fortunate in having the opportunity to have this material Class Tested at the University of Maryland under the guidance of Professor Marshall Abrams. One very important and continuing source of suggestions has been from my colleagues

Yi-Tzuu Chien, Elliot B. Koffman, John L. C. Löf, Bernard W. Lovell, and Howard A. Sholl who have made extensive comments and suggestions during the complete evolution of this book from the first set of rather incomplete and sketchy classroom notes. The task of writing this book has been simplified considerably by the help of my Wiley editor Donald C. Ford. The ability of Mrs. Norma Gingras to transform my illegible handwriting into typed copy is also deeply appreciated. Finally, I thank my wife, Aline, for her patience and encouragement throughout the development of this book.

Mansfield, Connecticut, 1970 TAYLOR L. BOOTH

Contents

CHAPTER I

Introduction to Digital Systems

I. Introduction

Because of the increasing complexity of civilization, man has been forced to continually develop better and more efficient techniques to process and utilize information. Initial attempts at developing information processing aids centered around improving methods of carrying out mechanical manipulations of numbers. During the 17th centry many of the leading mathematicians and scientists developed calculating devices to aid them in their research. As industrial technology developed during the 18th and 19th centuries, these basic ideas were refined and extended to develop complex mechanical devices that could be used to control machines and aid businessmen in performing repetitive calculations and bookkeeping tasks.

In the early 1800's Charles Babbage proposed and attempted to construct a device that he referred to as an analytical engine. Conceptually this device was similar to our modern digital computers. Although he was able to build a simple model of his machine, he was never able to complete the construction of a machine that would handle practical problems. One of the reasons for his failure was that the design called for so many moving mechanical parts that the inherent friction between the various parts prevented satisfactory operation of the complete machine. Even though Babbage failed to develop a practical device, many of the concepts that he developed laid the foundation for the design concepts of modern computers.

Computers, as we know them today, have become practical only because we have been able to replace mechanical devices with electronic devices. In the late 1930's and early 1940's a series of relay computers were built through the joint effort of Harvard University, Bell Telephone Laboratories, and IBM. Although these computers operated satisfactorily, they were quickly superseded by electronic computers.

In 1946 J. P. Eckert and Dr. J. W. Mauchly developed the first electronic computer, the ENIAC, at the Moore School of Engineering at the University of Pennsylvania. This computer contained 18,000 vacuum tubes. Vacuum tubes were so unreliable at that time that the predicted mean time to failure was shorter than the mean time to repair the device. Nevertheless the computer did work and was used by the U.S. Army for a number of years.

As the capability of computers and digital systems became better understood, many major technical advances were made. With the introduction of the transistor in the early 1950's, it became possible to design and construct highly reliable computers. We have now reached the point where there are a large number of manufacturers that are producing computers of various sizes and capabilities with prices that range from a few thousand to many millions of dollars.

The majority of people who come in contact with computers can be classified as occasional computer users. Their main interest is to use the computer to carry out the routine data processing task or calculations needed as part of their work. By using procedure-oriented languages such as FORTRAN, COBOL, or PL/1, these people are able to carry out data processing tasks without worrying about the internal organization or structure of the computer.

The high information processing rates of modern computers, however, makes it possible to apply computers to a variety of information processing tasks that were not even conceived of before the development of modern computers. Consequently just as an engineer or a scientist must understand the limitations of the physical laws of nature he must also develop an understanding and appreciation of the laws dealing with the utilization, processing, and transmission of information.

This book has been designed for the person who has reached the point where he wishes to use a computer as more than a calculating device to solve routine problems. Consequently we first investigate the mathematical techniques that are used to describe and analyze digital networks and systems. Next, the methods that may be used to design combinational and sequential logic networks, which are found in every digital system and computer, are presented. Once the operation of these basic building blocks is understood we then consider how they can be used to form complex data processing devices and general purpose digital computers. Finally, we consider the various types of programming systems that can be used to program a computer and how they are related to the efficiency of the overall information processing system.

II. Algorithmic Processes

Two of the major problems in designing a complex digital information processing system concern:

1. The identification of the various fundamental information processing tasks that must be accomplished.

2. The specification of the component parts of the system needed to carry out these tasks.

From an abstract viewpoint, the complete computational process carried out by any digital information processor or computer can be formally represented by the mathematical relationship

$$F(x) = y$$

where x represents the data presented to the processor, $F(x)$ represents the computation performed on the data and y represents the results of this computation. The computation represented by $F(x)$ can take many forms.

In the simplest case, the processor might be a simple logic network that takes the current value of n input variables, $[x_1, x_2, \ldots, x_n]$, and immediately produces an output $f(x_1, x_2, \ldots, x_n)$. On the other hand, the processor might be a large-scale computer system that measures the status of a chemical production process and produces output signals to control the rate at which certain chemical reactions are allowed to take place.

For each of these information processing tasks or any other tasks that we might wish to perform, there is only one restriction that we must place on the computation represented by $F(x)$. We must be sure that there is an explicit and unambiguous set of instructions that tells us how to perform the computation. This set of rules is called an algorithm for the computation of $F(x)$.

Algorithm. We say that an *algorithm* for the computation $F(x) = y$ exists if there is an ordered sequence of discrete steps that can be performed mechanically by a device such that given x the device either:

(a) forms $y = F(x)$ by executing these steps in the prescribed order, or
(b) indicates that no y exists that satisfies the conditions of the computation.

The device must require only a finite number of steps to reach one or the other of these decisions.

From this definition we see that if we are to implement an algorithm on a digital device we must reduce the steps of the algorithm to a sequence of elementary operations that can be performed by the device. In some cases the device will consist of a simple digital network constructed to perform the complete computation in one step while in other cases the algorithm for the computation will be so complex that it requires a large number of steps and can only be implemented on a large-scale digital computer. We now investigate the general properties of algorithms as they relate to the design and utilization of digital information processing devices. This will, in turn, allow us to gain an insight into the interrelationship between the organization of digital networks and computers and the computational processes that can be carried out by these devices. Our first task is to define what we mean by an "elementary operation."

We automatically carry out an algorithm every time we perform a particular mathematical or logical operation. However, we seldom give any thought to the form that this algorithm takes. This is because our previous experience has taught us to associate fixed reactions and interpretations to different mathematical symbols. However, if we wish to describe how we carried out a given calculation to someone who does not have our background we must explain, in great detail, how the computation is performed.

For example assume that we wish to compute the sum of the three two-digit numbers

$$A = a_2a_1 \qquad B = b_2b_1 \qquad D = d_2d_1$$

Normally we would probably carry out the addition in our heads, write down the answer

$$Y = A + B + D = y_3y_2y_1$$

and consider our problem solved. Most computers cannot simultaneously add three numbers together. They must, instead, perform the calculation in two steps as:

Step 1 $R_1 = A + B$
Step 2 $Y = R_1 + D$

Thus, if we assume that we can use the elementary operation of adding two numbers together, our calculation can be completed by using a two-step algorithm. However, let us consider what would happen if the computing device that we had could only add two digits at a time. Should this be the case we would have to replace both step 1 and step 2 with a sequence of steps that would describe how the two numbers are to be added together digit by digit.

The elementary operation in this case would be digit addition which can be formally defined by

$$\begin{array}{cc} & u_i \\ & \underline{v_i} \\ c_i & s_i \end{array}$$

where s_i is the unit sum of the two digits and c_i is the carry. For example let $u_i = 9$ and $v_i = 5$. Then

$$\begin{array}{cc} & 9 \\ & \underline{5} \\ 1 & 4 \end{array}$$

and we see that $c_i = 1$ and $s_i = 4$.

It is possible to build a device to compute the two functions

$$s_i = F_1(u_i, v_i)$$

and

$$c_i = F_2(u_i, v_i)$$

If we must use this device to compute

$$Y = A + B + D$$

then we could use the following algorithm:

Algorithm to compute $Y = A + B + D$ First Part Compute $R = A + B$		Example of Calculation Performed Using Algorithm $Y = 25 + 34 + 98$
Step 1	$r_1 = F_1(a_1, b_1)$	$r_1 = 9 = F_1(5, 4)$
Step 2	$c_1 = F_2(a_1, b_1)$	$c_1 = 0 = F_2(5, 4)$
Step 3	$p_2 = F_1(a_2, b_2)$	$p_2 = 5 = F_1(2, 3)$
Step 4	$r_2 = F_1(p_2, c_1)$	$r_2 = 5 = F_1(5, 0)$
Step 5	$m_2 = F_2(a_2, b_2)$	$m_2 = 0 = F_2(2, 3)$
Step 6	$n_2 = F_2(p_2, c_1)$	$n_2 = 0 = F_2(5, 0)$
Step 7	$c_2 = F_1(m_2, n_2)$	$c_2 = 0 = F_1(0, 0)$
Second Part $Y = R + D$		
Step 8	$y_1 = F_1(r_1, d_1)$	$y_1 = 7 = F_1(9, 8)$
Step 9	$c_1' = F_2(r_1, d_1)$	$c_1' = 1 = F_2(9, 8)$
Step 10	$p_2' = F_1(r_2, d_2)$	$p_2' = 4 = F_1(5, 9)$
Step 11	$y_2 = F_1(p_2', c_1')$	$y_2 = 5 = F_1(4, 1)$
Step 12	$m_2' = F_2(r_2, d_2)$	$m_2' = 1 = F_2(5, 9)$
Step 13	$n_2' = F_2(p_2', c_1')$	$n_2' = 0 = F_2(4, 1)$
Step 14	$c_2 = F_1(m_2', n_2')$	$c_2 = 1 = F_1(1, 0)$
Step 15	$y_3 = F_1(c_2', c_2)$	$y_3 = 1 = F_1(1, 0)$
Result $Y = y_3y_2y_1$		Result $Y = y_3y_2y_1 = 157$

This algorithm actually represents the following very simple addition process.

$$\begin{array}{c|ccc|c}
\text{Stage 1} & 0 & 0 & & \text{Carry} \\
\text{Compute } R = A + B & 0 & 2 & 5 & A \\
 & 0 & 3 & 4 & B \\
\cline{2-4}
 & 0 & 5 & 9 & R
\end{array}$$

$$\begin{array}{c|ccc|c}
\text{Stage 2} & 1 & 1 & & \text{Carry} \\
\text{Compute } Y = R + D & 0 & 5 & 9 & R \\
 & 0 & 9 & 8 & D \\
\cline{2-4}
 & 1 & 5 & 7 & Y
\end{array}$$

Thus we see that the set of basic operations that we can use affects the complexity of an algorithm. The example also illustrates how we can solve the problem. When we are working with a system there will usually be a sequence of operations that are used enough times to justify attaching a functional name to them. Thus we could define a function

$$F_S(U, V) \rightleftharpoons U + V$$

that stands for the steps needed to form the sums in the above algorithm. The algorithm then goes back to

Step 1 $R = F_S(A, B)$
Step 2 $Y = F_S(R, D)$

This idea of taking a sequence of simple operations and defining a new operation to represent this sequence is used repeatedly throughout this book. In this way we can concentrate on the important concepts being presented without worrying about the fine details of how each step of the process is actually implemented.

Flow Chart Representation of Algorithms. One of the most convenient ways to represent an algorithm is by means of a *flow chart* or *flow diagram.* A flow chart is a graphical representation of a particular algorithm that indicates the logical sequence of operations that are to be performed by the device that executes the algorithm. The flow chart is basically a collection of specially shaped boxes and directed lines. The contents of each box indicate which operations are to be performed while the lines that interconnect the boxes indicate the sequence in which the instructions are to be performed.

A very elaborate flow-chart symbology has been evolved by computer programmers. However, for our needs in this book we will limit our flow chart symbols to those illustrated in Figure 1-1. Reference 6 at the end of this chapter presents an extensive discussion of flow-charting techniques.

Each instruction box and decision box will contain one or more expressions describing how the basic operations are used to carry out the calculations. It is assumed that the reader has been introduced to computer programming in sufficient detail to be aware of how flow charts are used. The following example will serve to review these ideas.

Assume that we wish to calculate the roots of the equation $ax^2 + bx + c$. If $a \neq 0$ then these roots are given by

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

The simplest possible flow chart for finding r_1 and r_2 is given in Figure 1-2.

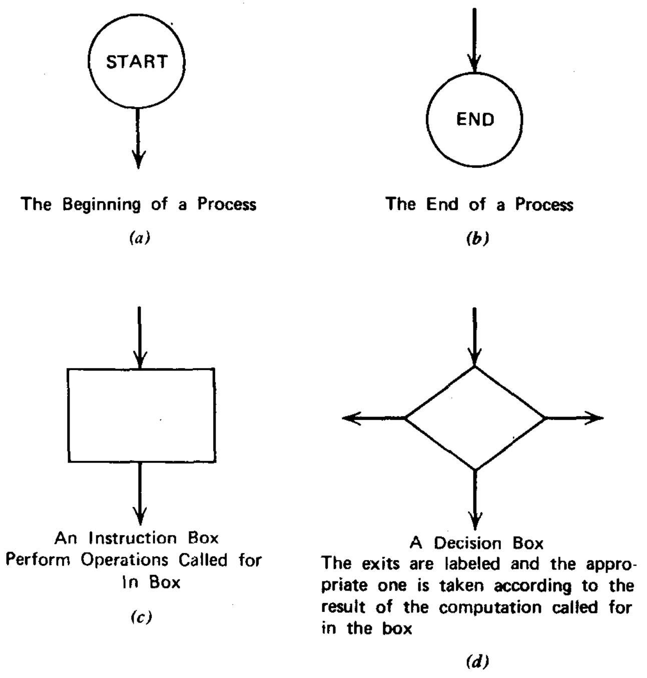
START
The Beginning of a Process
(a)
END
The End of a Process
(b)
An Instruction Box
Perform Operations Called for In Box
(c)
A Decision Box
The exits are labeled and the appropriate one is taken according to the result of the computation called for in the box
(d)

Figure 1-1 Basic flow chart notation. (*a*) The beginning of a process. (*b*) The end of a process. (*c*) An instruction box performs operations called for in box. (*d*) A decision box. The exits are labeled and the appropriate one is taken according to the result of the computation called for in the box.

However, if we examine this flow chart we see that it is not much different from our initial statement of the problem. In particular, it assumes that we have two basic operations corresponding to

$$f_1(a, b, c) = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

and

$$f_2(a, b, c) = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

that can be evaluated to find r_1 and r_2. Since these two functions are somewhat specialized, it becomes desirable to break the calculation down into smaller parts. Before we can do this we must consider some of the problems that must be overcome.