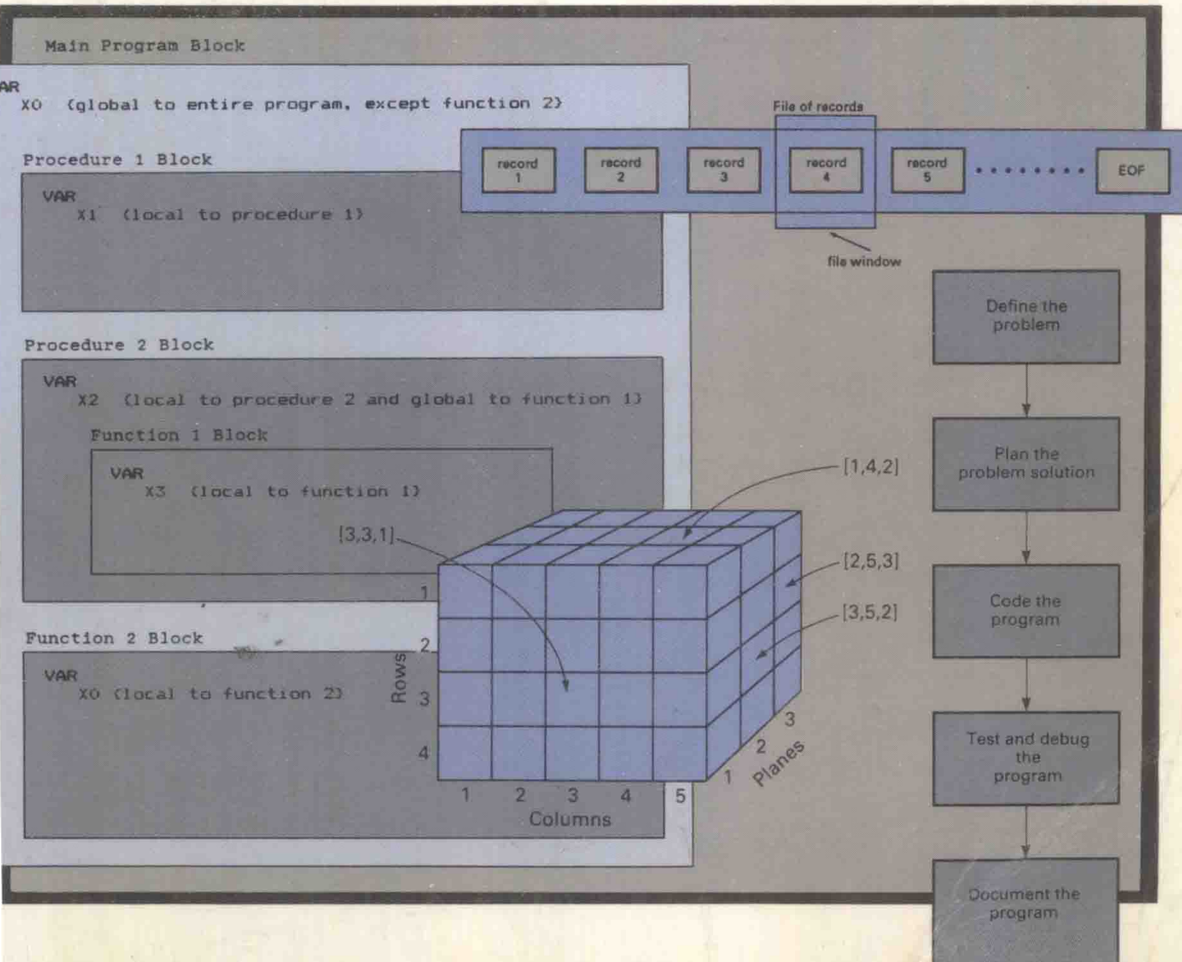
ANDREW C. STAUGAARD, JR.

DISK INCLUDED

STRUCTURING TECHNIQUES

AN INTRODUCTION USING TURBO PASCAL



STRUCTURING TECHNIQUES AN INTRODUCTION USING TURBO PASCAL

Andrew C. Staugaard, Jr.
The School of the Ozarks
Department of Computer Science

[illegible]

Prentice Hall, Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

Staugaard, Andrew C.
Structuring techniques.

Includes index.
1. Pascal (Computer program language) 2. Turbo Pascal (Computer program) 3. Structured programming.
I. Title.
QA76.73.P2S736 1989 005.26 88-12482
ISBN 0-13-853425-X

Editorial/production supervision and interior design: LISA SCHULZ GARBOSKI
Cover design: DIANE SAXE
Manufacturing buyer: MARY NOONAN and SALLYE SCOTT

Printed in the United States of America

10 9 8 7 6 5 4 3 2

ISBN 0-13-853425-X

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

To the one who has given me the life, strength,
and wisdom to write this book

PREFACE

With this text, you will learn how to create computer solutions to common problems found in mathematics, science, and business. In other words, this text will not only teach you the Pascal language, but more important, it will teach you how to define problems and plan their solution so that they can be easily ''coded'' using the Pascal language, or any computer language for that matter. Good problem definition and solution planning via algorithmic development are discussed early-on and carried as a theme throughout the text.

This text has been written to provide the undergraduate student a first exposure to problem solving using the Pascal language as vehicle. The text is designed to be used at the introductory level, assuming that the student has no previous background in programming of any kind. However, it is assumed that the student is familiar with basic algebra and right angle trigonometry concepts such as those that might be found in high school algebra and trig courses.

Each chapter includes numerous example programs and problems that specifically apply Pascal programming to simple problems found in mathematics, science, and business. The program examples and problems have been written in short, understandable modules that stress the fundamental concepts being discussed. In addition, several real world programming ''tasks'' have been included so that the student learns how to integrate simple program modules into comprehensive computer program solutions.

The text begins with a general overview of computer hardware and software technology in the first part of Chapter 1. This material can be covered rather rapidly if the students have already had some programming experience. However, the later part of Chapter 1 discusses what I call the ''Programmer's Algorithm'' and should be covered thoroughly. The Programmer's Algorithm is a step-by-step procedure that I have used to get students started on the right programming track by considering good problem definition, solution planning, and documentation. Of particular importance is solution planning via algorithmic development. I feel that once the student has defined the problem and planned a solution via an algorithmic structure, the actual program coding is secondary to the problem solving task. As a result, I stress good algorithmic development throughout the text. I have used a pseudocode algorithmic language that is simple and whose structure is very similar to Pascal. This approach allows for easy translation of the algorithm to the coded Pascal program.

Chapter 2 familiarizes the student with concepts in Pascal that are not found in non-structured languages like BASIC. It is important here that the student grasp the idea of data typing, since this is a primary feature of Pascal. In addition, the idea of program structure and the modular top-down design approach of a structured language are introduced in this chapter.

The student really begins to get his/her hands dirty in Chapter 3 where they learn how to get information in and out of a Pascal program. Special consideration is given to interactive user-friendly programming. The discussion in Chapter 3 is extended into Chapter 4 where the student learns how to perform standard arithmetic, Boolean, and function operations in Pascal. Several simple program tasks are illustrated at the end of this chapter to illustrate these operations.

Up to this point, the student has been writing simple straight-line programs. In Chapter 5 the If/Then, If/Then/Else, and Case operations are discussed to introduce the decision making element into the program structure. This is followed with a discussion of the Pascal iteration operations of While/Do, Repeat/Until, and For/Do in Chapter 6.

In Chapter 7, students really get the flavor of a structured language when they learn how to write their own functions and procedures in Pascal. From this point on, the idea of a modular top-down design approach to problem solving is emphasized. Such an approach using a structured language, like Pascal, is mandatory if the student plans on tackling complex programming tasks like those that are typically found in industry.

Chapter 8 begins with a discussion of user-defined data, another important feature of Pascal. Here, the student learns how to tailor a program to meet the application. This is followed by a comprehensive discussion of the array data structure. Simple one-dimensional arrays are discussed first, followed by a discussion of multi-dimensional arrays. The last section of the chapter applies arrays to the solution of simultaneous equations via Cramer's rule.

Records are the topic of Chapter 9. Here the student learns about the limitation of an array for storing different data types, and discovers the record as a solution. Record declaration and access in Pascal are covered thoroughly in this chapter using several real world business applications.

Files are discussed in Chapter 10 to conclude the text. The student first learns how to declare and access disk files through several simple examples. Then, the student learns how to construct general procedures that can be used to create, expand, read and change disk files to meet any given application.

The Pascal compiler employed is TURBO Pascal. TURBO Pascal was chosen due to its wide usage in the educational market. It is readily available at an affordaole price and runs on just about any microcomputer, including the IBM PC. Although TURBO

Pascal is the primary vehicle for this text, the text is also appropriate for use with other versions of Pascal, such as Standard Pascal, UCSD Pascal, or MacPascal. I have made an effort to discuss the differences between TURBO and the others where appropriate.

A unique feature of this text is the inclusion of a student disk which contains the answers to the odd chapter questions and programming problems. The programming problem solutions are filed individually on the disk so that the student can call the program from disk, examine it, and actually execute it to observe its operation. Many of these working programs can be further used by the student to solve common problems found in mathematics, science and business. In addition, the student disk can also serve as a work disk for the student's own programs, thus eliminating the need for the student to purchase a separate work disk.

Enjoy!

Andrew C. Staugaard, Jr.

CONTENTS

1

GETTING ACQUAINTED WITH COMPUTERS, PROGRAMS, AND PASCAL

INTRODUCTION

This first chapter has been written to provide you with an introduction to computers, computer programs, and Pascal in general. You will learn about the relationship between the computer system and the computer programs that operate the system. In particular, you will study the steps required to solve just about any programming problem.

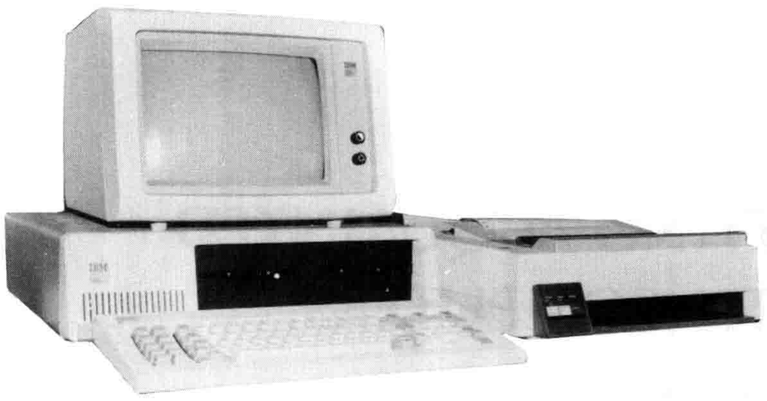
Of extreme importance is the last section of this chapter, which teaches you how to develop algorithms. Make sure you understand this material and work the related problems at the end of the chapter. As you become more experienced in programming, you will find that the ''secret'' to successful programming is good planning through the use of algorithms.

Any computer system, regardless of its size can be broken down into two major components: hardware and software. So, let's begin with a comprehensive overview of each.

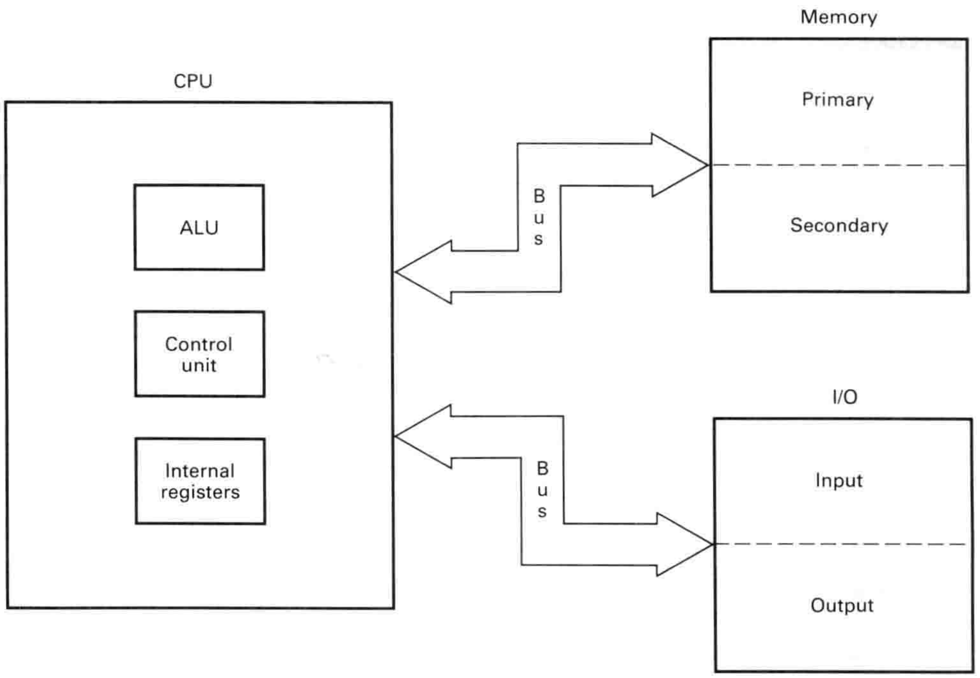
1-1 THE HARDWARE

You undoubtedly have seen some of the hardware components of a computer. These are the physical devices that you can see and touch, such as those shown in Figure 1-1a. This typical microcomputer system obviously has a keyboard for user input, a display screen for output, and magnetic disk drives for program and data storage. Two very important parts of the system that cannot be seen, because they are inside the console, are the ***central processing unit*** and its ***working memory.***

The block diagram in Figure 1-1b shows all of the major hardware sections of the



(a)



(b)

Figure 1-1 (a) A typical microcomputer system and (b) its hardware structure, or architecture.

system. From this figure, you see that the system can be divided into five functional parts: the central processing unit (CPU), main working or primary memory, secondary memory, input, and output.

The Central Processing Unit (CPU)

The central processing unit (CPU) is the brains of the entire system. This is where all of the calculations and decisions are made. In a microcomputer system, the entire CPU is contained within a single integrated circuit (IC) chip called a ***microprocessor.*** In fact, this is what distinguishes a microcomputer from a mini or mainframe computer. In mini and mainframe computers, several ICs make up the CPU, not just one as in a microcomputer. A typical microprocessor IC is pictured in Figure 1-2, along with a magnified view of the chip itself.

There are three basic functional regions within the CPU that you should know about. They are the ***arithmetic logic unit (ALU),*** the ***control unit,*** and the ***internal registers.***

The Arithmetic Logic Unit (ALU)

As its name implies, the arithmetic logic unit performs all of the arithmetic and logic operations within the CPU. The arithmetic operations performed by the ALU include addition, subtraction, multiplication, and division. These four basic arithmetic operations can be combined to perform just about any mathematical calculation, from simple arithmetic to calculus.

Logic operations performed by the ALU are comparison operations that are used to compare numbers, letters, and special characters. The three basic logic comparison operations are equal to (=), less than (<), and greater than (>). These three basic operations can be combined to form the three additional logic operations of not equal (<>), less than or equal to (<=), and greater than or equal to (>=).

Table 1-1 summarizes the arithmetic and logic operations performed by the ALU. Notice the operations symbols listed in the table. These are the symbols that you will use later to perform arithmetic and logic operations when writing Pascal programs.

The Control Unit

The control unit section of the CPU directs and coordinates the activity of the entire system. This section interprets program instructions and generates electrical signals to the other parts of the system in order to execute those instructions. The control unit communicates with other sections of the CPU via internal signal paths called ***buses.*** The control unit often is likened to a traffic cop or orchestra leader, because it directs the activity of the entire system.

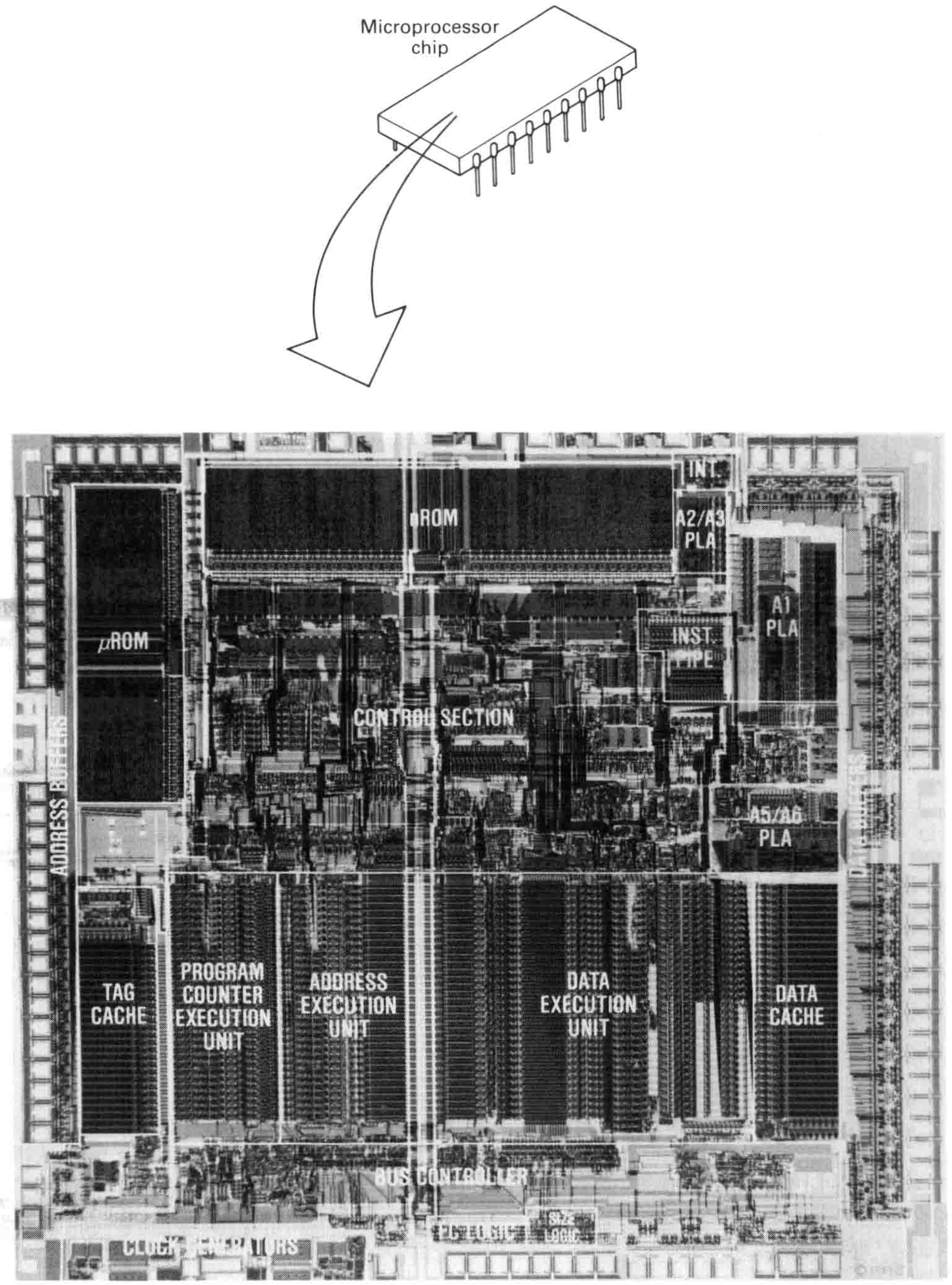


Figure 1-2 A microprocessor chip is the CPU of a microcomputer system. (Copyright by Motorola, Inc. Used by Permission.)

TABLE 1-1

A Summary of Arithmetic and Logic Operations Performed by the ALU Section of the CPU

Arithmetic Operations	*Symbol*
Addition	+
Subtraction	−
Multiplication	*
Division	/
Logic Operations	*Symbol*
Equal to	=
Not equal to	<>
Less than	<
Less than or equal to	<=
Greater than	>
Greater than or equal to	>=

Internal Registers

The internal register section of the CPU contains temporary storage areas for program instructions and data. In other words, these registers temporarily hold information while it is being processed by the CPU. Several different types of registers are employed. They include ***accumulators***, ***data registers***, ***address registers***, and ***general-purpose registers.*** It is not important that you know the precise operation of each of these registers for high-level Pascal programming. However, they will become important to you if you ever do any assembly language programming. More about this later.

Primary Memory

Primary memory often is called main working memory. The reason for this is that primary memory is used to store programs and data while they are being ''worked,'' or executed, by the CPU. As Figure 1-3 shows, there are two basic types of primary memory: ***random access memory*** **(*RAM*)** and ***read-only memory*** **(*ROM*).**

Random Access Memory (RAM)

Random access memory is memory for you, the user. When you enter a program or data into the system, it goes into RAM. This is why the amount of RAM often is quoted when you buy a computer system. You most likely have heard the terms *64K*, *128K*, *256K*, and so on, when describing a microcomputer system. This is the amount of RAM, or user memory, that the system contains. Here, the letter *K* stands for the value 1,024. Thus, a

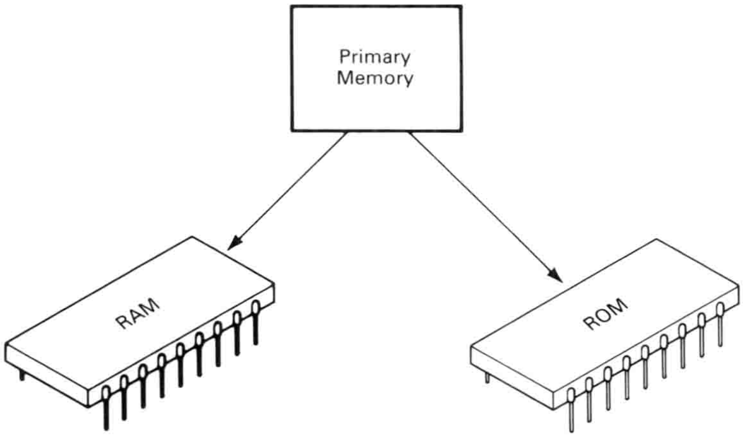


Figure 1-3 Primary memory consists of RAM (user memory) and ROM (system memory).

64K system has 64 × 1,024 = 65,536 *bytes* of RAM, a *128K* system has 128 × 1,024 = 131,072 *bytes* of RAM, and so forth. The more bytes of RAM a system has, the more room there is for your programs and data. As a result, larger and more complex programs can be executed with larger amounts of RAM.

By definition, RAM is ***read/write*** memory. This means that information can be written into, or stored, into RAM and read from, or retrieved, from it. When writing new information into a given area of RAM, any previous information in that area is destroyed. Fortunately, you don't have to worry about this when entering programs, since the system makes sure that the new program information is not written over any important old information.

Once information has been written into RAM, it can be read, or retrieved, by the CPU during program execution. A read operation is nondestructive. Thus, when data are read from RAM, the RAM contents are not destroyed and remain unchanged.

One final point about RAM: It is ***volatile.*** This means that any information stored in RAM is erased when power is removed from the system. As a result, any programs that you have entered in main working memory (RAM) will be lost when you turn off the system. You must remember to save your programs on a secondary memory device, such as a disk, before turning off the system power.

Read-Only Memory (ROM)

Read-only memory often is called ***system memory*** because it stores system-related programs and data. These system programs and data take care of system-related tasks such as reset, cursor control, binary conversions, and so on. All of these system programs are part of a larger ***operating system*** program that is permanently stored in ROM or on a disk. We say that the ROM and the operating system programs that it contains are ''transparent'' to you, the user. The word *transparent* is appropriate, since you do not ''see'' or concern yourself with the operating system programs during the course of programming in a high-level language, like Pascal.

As its name implies, read-only memory can only be read from and *not* written into. Consequently, information stored in ROM is permanent and cannot be changed. Since the information is permanent, ROM must be ***nonvolatile.*** This means that any information stored in ROM is not lost when power is removed from the system. Due to this feature, ROM programs often are called ***firmware.***

You might have heard the terms ***mask-programmed ROM, EPROM,*** or ***EEPROM.*** These are different forms of ROM that might be part of a system. Mask-programmed ROMs are programmed by the ROM chip manufacturer and can never be altered. EPROM stands for erasable, programmable read-only memory. Information stored in these chips can be erased using ultraviolet light. After erasure, the chip can be reprogrammed. EEPROM stands for electrically erasable, programmable read-only memory. These chips can be erased with electrical signals and reprogrammed.

Both EPROMs and EEPROMs usually are used during the initial development phases of a system. The reason is that any operating system *bugs* can be corrected by reprogramming the EPROM and EEPROM. Once the system is developed and all of the bugs are removed, the EPROMs and EEPROMs are usually replaced with less expensive mask-programmed ROMs.

Secondary Memory

Secondary memory, sometimes called ***mass storage,*** is used to hold programs and data on a semipermanent basis. The most common type of secondary memory used in a microcomputer system is the magnetic floppy disk shown in Figure 1-4. The floppy disk in Figure 1-4 gets its name from the fact that it is flexible, rather than rigid, or hard. The disk is coated with a magnetic material and enclosed within a plastic jacket. When inserted into a ***disk drive*** the disk is spun on the drive at about 300 rpm. A read/write recording head within the drive reads and writes information on the disk through the access slot, or window, in the disk jacket.

Notice that there is a small write-protect notch on one side of the disk jacket. When covered with a small sticker or piece of tape, new information cannot be written, or recorded, on the disk. This feature is provided so that you can protect the disk from accidental erasure.

When writing programs, you will first enter the program code into the system primary memory, or RAM. When entering TURBO Pascal programs, you must create a ***work file*** name for the program. This work file name creates an area, or ***file,*** on the disk where your program will be saved. As you enter and work with your program in RAM, you will periodically ***save*** the program on the disk so that it is permanently stored. When you save the program, the system simply copies the program from RAM and saves it on the disk under the work file name that you created. This process is illustrated in Figure 1-5a.

Saving a program on disk allows you to retrieve it easily later. To read a program from a disk, you simply insert the disk in the disk drive and enter the work file name assigned to the program. This tells the system to read the work file and transfer it into