1983

Development of the computer systems management instructional laboratory at the Naval Postgraduate School

Mills, Kenneth J.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/19743

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

DEVELOPMENT OF THE COMPUTER SYSTEMS MANAGEMENT INSTRUCTIONAL LABORATORY AT THE NAVAL POSTGRADUATE SCHOOL

by

Kenneth J. Mills
Jesse M. Richards
Glen F. Tilley

June 1983

Thesis Advisor: N. F. Schneidewind

Approved for public release; distribution unlimited

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Development of the Computer Systems Management Instructional Laboratory at the Naval Postgraduate School		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June 1983 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Kenneth J. Mills Jesse M. Richards Glen F. Tilley		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June, 1983 13. NUMBER OF PAGES 295
14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Technology, Instructional Laboratory

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The ability to converse effectively with technicians has been recognized as a critical skill for managers of data processing activities. This need has been addressed by the Association for Computing Machinery in their recommended curricula for the education of Information Systems specialists. Members of the Association have also described the functions of a graduate of those curricula to be that of a boundary spanner and a (Cont)

ABSTRACT (Continued) Block # 20

change agent. Other authors have identified that these skills need to be gained in practical environments, and that the manager needs to know at least a minimum of the technical language in order to select good technicians for his staff, and to communicate with that staff effectively. At the Naval Postgraduate School a course of instruction in technical aspects of the computer was designed into a newly constructed microcomputer laboratory. This thesis is the report of the evolution of that laboratory and course of instruction.

Development of the Computer Systems Management Instructional Laboratory at the Naval Postgraduate School

by

Kenneth J. Mills
Lieutenant, United States Navy
B.S., University of New Mexico, 1976

Jesse M. Richards
Commander, United States Navy
B.A., University of Virginia, 1967

Glen F. Tilley
Lieutenent, United States Navy
B.S., University of Washington, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
June 1983

ABSTRACT

The ability to converse effectively with technicians has been recognized as a critical skill for managers of data processing activities. This need has been addressed by the Association for Computing Machinery in their recommended curricula for the education of Information Systems specialists. Members of the Association have also described the functions of a graduate of those curricula to be that of a boundary spanner and a change agent. Other authors have identified that these skills need to be gained in practical environments, and that the manager needs to know at least a minimum of the technical language in order to select good technicians for his staff, and to communicate with that staff effectively. At the Naval Postgraduate School a course of instruction in technical aspects of the computer was designed into a newly constructed microcomputer laboratory. This thesis is the report of the evolution of that laboratory and course of instruction.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION TO THE LABORATORY

A. RATIONALE FOR THE LABORATORY

In 1972 Oliver Wight identified a problem for executives who knew too little about how computers work.

> "What a great job the technicians have done in creating a computer "mystique."...the computer technicians have sold {the executive} a bill of goods that he must understand how the computer works... But what he really needs to know about how the computer works is very limited indeed, and when technicians create a "mystique" around the machine--a barrier for the manager--they not only make him dependent upon them, but they also seriously impair his ability to make intelligent decisions about the use of the computer." [Ref. 1]

Wight's warning was that the executive should not try to become a technician himself, but that he needs to know enough about what the computer system could do and how it does it to make sound managerial decisions. This requires some knowledge of the technical jargon used by specialists, but not a full technical competency. At a minimum, he must be able to overcome the mystique of the computer and understand it as a management tool.

Despite this warning that the executive must not be surrounded by computer mystique, the current state of affairs is such that many top executives are not always fully able to penetrate that mystique and make those intelligent decisions. In a recent article Debra Zahay indicated that many businesses which are hiring graduates with Masters in Business Administration degrees are using them in data processing functions, principally to improve communication between technical and non-technical staff. This is a result of the "shortage of programmers and technical people who can communicate with nontechnical staff." [Ref. 2]

Obviously, the inability of the executive to communicate effectively with the technical staff in the computer operations area is an area of growing concern to top management executives. In one of the latest textbooks on management of information systems the author states

> "The complexities of developing IS {information services} systems has forced the creation of specialized departments resulting in a series of strained relationships with the users of their service. . . . IS has specialized in order to harness the various necessary technical skills to get the job done. The specialists have appropriately developed their own language systems. They speak of bits, bytes, DOS, CICS, and so on to communicate among each other. General management, however, has a quite different language, featuring words such as sales growth, return on investment, and productivity." [Ref. 3]

The writers of that statement do not argue that the language created by the technicians is to blame for the fact that communication is poor between technicians and managers. They term the language systems that have been developed "appropriate." The fault for the lack of communication lies partly with managers who cannot understand the most basic vocabulary of computers and partly with technicians who cannot understand the most basic vocabulary of management. The education of business administrators in basic technical vocabulary can be addressed by assigning MBAs to jobs in the management of data processing operations as an entry-level position, but as Zahay points out, "{This practice} is often a stopgap solution to the problem of communication between functional areas and systems staff." [Ref. 4] It is apparent that there is a problem of poor communication between the non-EDP business manager and the computer scientist and programmer.

The Association for Computing Machinery views the role of the information systems specialist as a bridge between these two diverse areas. One educator in IS said, "The

information system designer (and) implementer is a <u>boundary spanner</u> and a <u>change agent</u>. Therefore, the organizational knowledge should include an understanding of the typical problems encountered by boundary spanners and change agents and the common concepts, strategies and tools required of the individuals enacting such roles." [Ref. 5] In order for the manager to act as this boundary spanner, he must have a working knowledge of the areas he is to span. In fact, the ACM delineates the graduate of the recommended IS curriculum thusly:

> "1. The Information Systems curriculum teaches information system concepts and processes with the two contexts of organizational functions and management knowledge and technical information systems knowledge...
>
> 2. The Information Systems graduate is expected to work within the environment of an organization and to interact with both organizational functions and computer technology.
>
> 3. In technical expertise, the Information Systems curriculum places substantial emphasis on the ability to develop an information system structure for an organization and to design and implement applications." [Ref. 6]

This is not to imply that the technical expert need not be mindful of the need to communicate with non-technical staff. Indeed, the need for the technicians to be able to communicate with lay persons has already been clearly identified [Ref. 7]. However, the training of technicians to communicate with the lay individual does not relieve management of the responsibility of having some basic skill in the technical area, if only in the terminology. This training obviously need not be so technical as that of the technician, but should be deep enough that the manager may reasonably communicate with the technician and be able to evaluate and hire a technically competent staff [Ref. 8]. Ideally this training should be consistant between information system specialists, or problems will undoubtedly develop both

within and without the organization as the translators need someone to translate between them as well.

In the curriculum description for the Graduate Education program the ACM called for "... knowledge of basic hardware {and} software components of computer systems, and their patterns of configuration [Ref. 9]. In the description of the recommended course content, the ACM further specified "Processor, memory, input/output, mass storage, remote transmission modules; function and possible realization of each" were to be the subjects of the Computer Systems course of the curriculum [Ref. 10]. Additionally, the ACM provided this rationale for the inclusion of a course in computer concepts in the curriculum, "It is important for the student to possess a broad familiarity with fundamental concepts and terminology associated with computer hardware systems and operating systems." [Ref. 11]

The desire to enable the information system manager to be conversant with the technical language as well as the financial language creates some unique problems for the schools which offer an information systems curriculum. At the Naval Postgraduate School, for instance, the students have a variety of educational backgrounds, ranging from the more technical degrees in computer science, physics, engineering, etc., to the liberal arts degrees in such diverse majors as psychology, English, etc. Some of the students have highly skilled financial backgrounds, including a few with MBAs and many from the Supply Corps. Some have a wide experience in working with computers on a daily basis as a result of previous tours of duty in data processing centers of various sizes. The challenge for the curriculum managers is to provide sufficient course work in both the financial and managerial arenas as well as the technical information on the functioning of computers to provide the graduates with at least the minimum skills called for by the ACM

curriculum guidelines. This challenge is made more difficult by the requirement that the student officers be returned to non-scholastic duty as soon as possible. The standard course of instruction, then, must have the requisite coursework, but at such a level that the somewhat experienced student has a challenge and the novice is not left behind.

At the Naval Postgraduate School the traditional approach of classroom lectures has been used to provide the courses which cover the hardware and operating systems. Only one of the courses normally in the Computer Systems Management curriculum has a technical laboratory associated with it: CS 2810, which is an elementary structured programming course in which PASCAL is taught on an IBM 3033 as an adjunct to the structured programming concepts [Ref. 12]. None of the required courses has any practical exposure to the subject hardware or software, although some do offer exposure to development of software as a product of a design program. The school does offer courses in other curricula that can provide the information systems student with this technical background. Due to scheduling conflicts, however, it is not always practical for every information systems student who wishes to include these courses in his studies to be able to do so.[1] In a similar circumstance, where hardware concepts are taught in the

---

[1] The curriculum presently allows the "typical" student four quarters in which he may take one elective course per quarter, and that elective must normally come from one of several predetermined subsets of course offerings called Emphasis Areas (EA). This system is driven somewhat by the military nature of the school, in that the various warfare sponsors of the curriculum have education needs which should be met as well as the traditional academic requirements for a Masters Degree in Information Systems. If a student wishes to take a course outside the normal sequence for his EA, it must be taken as an overload, or as a replacement for a course in which the student can receive validation for previous study or experience. Validation is carefully monitored to maintain academic integrity, and the more usual circumstance is that the student must overload to move into non-traditional courses.

classroom by lecture, Cook described the computer science curriculum at Central Michigan University as having "A major failing...the absence of a digital logic laboratory for the course. The design problems and the operation of the Arithmetic/Logic Unit and control unit could be made much more understandable to the student if such a laboratory were available." [Ref. 13] It is therefore logical to decide that the implementation of a hands-on laboratory for logical device training would be of major benefit to the students in an information science curriculum.

B. ESTABLISHMENT OF THE LABORATORY

In 1980 it was decided that the Administrative Science Department, the department of the Naval Postgraduate School responsible for administering the degree of Master of Science in Information Systems, would install a microcomputer laboratory for the students and faculty of the school to use for research. The opportunity was seen to incorporate into this laboratory a course of instruction in the technical area that the faculty could use to supplement the classroom work and that the student could use to explore further the technical aspects of computing and computing equipment. In addition the laboratory would provide the student with the opportunity to work with microcomputers and desk-top computers. The laboratory would also support thesis work by students as well as the faculty research taking place at the school.

The laboratory was envisaged as spanning the technological levels from the simplest logic circuits to the most complex microcomputing system and local networks with peripheral I/O and telecommunication equipment. The laboratory was to have a coherent course of instruction to assist the student in learning as much as he wished on each of the

technological levels. In addition, it was planned to incorporate sufficient equipment that ultimately the laboratory could be used to simulate the functioning of a full computer center and thus be used as a teaching aid in the course for computer center operations.

It was decided to supplement the users manuals that came with the equipment of the laboratory with additional educational and training materials, principally because the general quality of users manuals provided with the systems was poor. It appeared that those manuals were written with the assumption that the user was to be knowlegeable of the subject area as a prerequisite to using the manual. The overall intent of the laboratory was that the novice student would be able to learn the technical language and operations without having to decipher an intensely technical journal of instructions. Additionally, the laboratory was unique to the Naval Postgraduate School, and few texts were available for self-paced work of this kind. Therefore, the decision was made to create the texts in-house, using a team of students to produce them. The same team of text-writers was also to manage the installation and construction of the various computer systems that were to go in the laboratory. This thesis is the report of that development effort.

II. CONSTRUCTION OF THE EQUIPMENT AND THE ROOM

A. BACKGROUND

The basic groundwork for the Laboratory was initiated in 1980 when Professor N. Schneidewind proposed and had approved the fundamental concept of a student and faculty learning center. At that time the NPS computer center was installing a new mainframe computer and would require all available room in Ingersoll Hall as remote terminal sites. In September of 1982 the room originally chosen for the instructional laboratory space was cleared of the terminals and construction was started on the laboratory. The authors were introduced to the project during the two months preceding the construction phase and began by updating the two year old work request that had initiated the action. The problems encountered and mistakes made during the construction phase will be discussed in this chapter.

B. EVOLUTION OF THE LABORATORY

During the development of the laboratory, the United States Coast Guard offered to locate a multi-user microcomputer in the laboratory. In the interest of obtaining a system with high order languages and application programs installed, the Administrative Sciences department accepted the offer. There was no delay in the development of the laboratory as a result, however a re-alignment of goals and objectives was required of the authors in order to accommodate the introduction of this additional system to the laboratory. The authors were directed by Professor Schneidewind to plan the physical placement of the Coast Guard system in the front room of the two room laboratory. On 19 November

the Coast Guard system was moved into the lab and placed in operation. Of the original 8 work stations present in the front room, all but two were taken by the system, leaving little space for other types of equipment.

The first piece of equipment received for the laboratory was the Intel Prompt-80 microprocessor design and training device. One of the authors was assigned the tasks of reviewing all the documentation accompanying the Prompt-80 and developing a user manual that would allow a computer novice to begin self-paced education at the machine language level.

The next equipment received was the Heathkit Digital Logic Trainer (in an unassembled kit form). The assignment for one of the authors was to assemble the kit and prepare a user manual that would allow a computer novice to educate himself on the digital electronics level of computers. The same author received the Heathkit Digital Techniques self-instruction course for review and evaluation.

The third author was assigned the tasks of reviewing a series of pre-recorded cassette tapes prepared by SYBEX, Inc, as a tutorial on microcomputers, interfacing techniques and computer architecture, and preparing a synopsis on each tape selected for the laboratory.

The above mentioned projects were performed in parallel with the construction of the room which began in August, 1982. By 21 October the tutorial on the Prompt-80 was nearing completion, and by 6 November the Heathkit Digital Logic Trainer was assembled and tested. The Prompt-80 manual was submitted to Professor Schneidewind on 11 November for examination and recommendations. The tutorial for the Heathkit Digital Logic Trainer was submitted on 25 November and by 29 November the Heathkit Digital Techniques self instructional course had been thoroughly reviewed and was returned to Professor Schneidewind. The pre- recorded

cassette tapes were reviewed and the synopsis prepared for the laboratory by 15 December 1982.

A second round of equipment construction was begun at the end of December with the assembly of a Heathkit H-9 video terminal. A Heathkit H-25 dot matrix printer and H-89 computer with H-17 external disk drives followed in January. This work was completed by one of the authors, and he began writing introductory tutorials for these additional pieces of equipment. Another of the authors was assigned the task of writing a user's manual for the Intel SDK-85 microcomputer experimentation device. The third author began assembling a series of digital electronics experiments that could be performed on the Heathkit Digital Logic Trainer or the E & I Instruments, Inc. "Digi Designer" device.

By 28 February all construction and preliminary writing was completed. The authors then began compilation of information and data required to include in this thesis.

It should be mentioned at this time that during the academic quarter from January to March, 1983, the front room of the laboratory was opened for student use, with the Coast Guard multi-user system and three modem equipped terminals installed to permit access to the ARPANET for the course on telecommunications.

A brief summarization of events and cumulative time required for each is listed as figure 2.1. The time summary for the tutorial preparation is included in a later chapter.

C. DESIGN AND CONSTRUCTION CONSIDERATIONS

One prime consideration in the development of the laboratory was to present a friendly, well defined setting for anyone interested in learning about microcomputers and digital electronics. The importance of a friendly user atmosphere cannot be over emphasized, particularly since most

EVENT	MANHOURS
Collecting components for experiments	58
Interfacing equipment	8
Planning, paperwork and discussions with advisors	49
Monitoring of construction progress	27
Total	142 hrs.

Figure 2.1 **MANHOUR ALLOCATION.**

people facing unfamiliar equipment feel a certain level of apprehension. With this goal in mind, the authors planned for equipment that would provide a logical learning continuity from the digital electronics level to higher level languages and application programs.

At the outset of the project, the Naval Postgraduate School already owned some of the equipment to be used, a Heathkit H-8 computer, Heathkit H-9 terminal, an Intel Prompt-80, and an Intel SDK-85 system design kit. The authors requested that additional computer equipment be logically related to this inventory. As a result a Heathkit H-89 computer and external disk drives were ordered for the laboratory. This choice provided the laboratory with a contiguous line of equipment that was from the same family of central processor units.

The sequence of events during the construction of the laboratory and the equipment could have, at times, been described as fraught with problems. This report should assist the reader in developing and building an instructional laboratory by presenting some of the pitfalls encountered and considerations necessary for a successful installation.

The following sections will highlight those items that require planning and decisions based on the desired use of the laboratory.

1. Equipment and Software

Although it would appear that equipment and software selection would contain the bulk of decisions concerning a project of this type, that is not necessarily the case. The choice of software is, of course, very significant if a particular application is important to the use of the lab. Care should be taken to select software and hardware that meets all projected needs, is relatively easy to learn and use, and is popular to the extent that it has a good history of use and maintenance.

2. Physical Layout of Room

In the design of a computer laboratory, there are some specific considerations concerning the physical layout of the room. In a laboratory like the one at the Naval Postgraduate School, it will be necessary to allocate space for computer workstations, peripheral devices such as printers and disk drives, laboratory equipment such as meters and oscilloscopes, digital training devices, and associated documentation. Sufficient storage space for unused equipment should also be provided. If the laboratory is supposed to support several courses, as it does at NPS, there will be different equipment required at different times, so large storage cabinets should be included in the lab. The counter tops for the work stations should be designed to make maximum use of available wall space. When laying out the floor plan, it is important to remember that people need leg room and elbow room . A collision may occur if there are adjacent work stations located around an inside corner.

One of the most difficult decisions will be the placement of shared devices such as printers and plotters. The work stations utilizing these devices will need a reasonable path for the connecting cables. Another factor for consideration is the expected traffic flow and possible interference between doors, counters, and equipment.

3. Cooling, Heating, and Ventilation

The room will contain electronic equipment, and therefore adequate heating, cooling, and ventilation should be provided. Each computer by itself will generate only a small amount of heat, but in the aggregate a room full of equipment may become warm enough to cause damage to the devices. One of the largest sources of heat will be the number of people in the room. Twenty people in a small room will have a definite effect on the room temperature. Generally speaking, computers function better in a cooler environment with low humidity. The trade-off to be considered is that people may not use the lab if they are uncomfortably cold. The best source of required temperature and humidity levels is the manufacturer's literature.

4. Electrical Power Requirements

Most digital electronic equipment contains an internal power supply and is designed to be plugged into a standard 110-120 volt three pronged (grounded) outlet. For a laboratory, at least two outlets per work station should be installed. Some computers provide auxiliary outlets to power peripheral devices, but not all can supply the heavy power requirements of high current devices such as printers. The outlets should be located in a convenient location, keeping in mind that most power cords extend from the rear of the device. Another consideration for electrical power is that computers are sensitive to voltage spikes and fluctuations that occur on a random but frequent interval. There are

filters available to suppress voltage spikes and constant voltage transformers or uninteruptable power supplies to protect against fluctuations.

5. Work Station Requirements

Each workstation should have enough room for a CRT display, a keyboard, a computer, and a printer or space for a modem and telephone. Since it may be unrealistic to fully equip all stations, the temptation may be to reduce the workstation space allocation in an attempt to save room. If the work stations spaces are too small to move things around, a serious degradation of flexibility can occur.

6. Cleaning

A small but significant problem of a computer laboratory deals with routine cleaning. Methods should be provided to adequately remove waste paper and trash from the lab. If the room is normally locked, an arrangement with the cleaning service will have to be made. Special cleaning solvents and equipment are needed for CRT screens, computer cases, and peripheral devices. Disk drives and other equipment are extremely sensitive to smoke and dirt. It would be a good idea to provide a whiteboard and felt tip markers instead of a standard chalk board. No smoking signs should be prominently displayed. Cleaning instructions are normally included with each piece of equipment.

7. Security

Security of a computer laboratory falls into two catagories. First, considerations must be made concerning the physical security of the equipment and software in the room. The NPS Instructional laboratory is protected with cipher locks on the doors and keyed locks on the storage cabinets. The combinations for the cipher locks are released

only to persons who read and sign a non-disclosure statement. Secondly, the software disks for proprietary software are issued in a similar manner, with an agreement not to copy proprietary software being signed prior to issuance.

D. REMARKS

The seven topics discussed above were all significant considerations in the development of the NPS Instructional laboratory. The list is by no means intended to be a comprehensive indicator of all possible problems. The chapter on lessons learned will discuss several problems encountered by the authors in these areas.

III. INSTRUCTIONAL MATERIAL DESIGN AND TESTING

During the formulation phase of the development of the laboratory instructional materials, several areas of consideration were evaluated. It was determined that due to the variety of equipment incorporated into the lab, uniformity in text style and instructional method should be a major factor in the design of the instructional materials.

A. METHOD

The two instructional methodologies considered for implementation were Computer Aided Instruction (CAI), and hard-copy, printed tutorials. The CAI method of instruction is primarily used for direct institutional support. Typical examples of CAI are Drill and Practice, Tutorials, Simulation/Gaming, Inquiry/Dialogue, Information Retrieval, and Problem Solving [Ref. 14]. CAI is accomplished through interactive computer tutorial sessions and thus requires the availability and use of a computer system. This requirement, coupled with the goal of uniformity in method and style, lead to the decision to utilize hard-copy, printed tutorials for all instructional equipment used in the laboratory. It was decided that printed tutorials would provide greater access to the learning materials and would allow greater mobility of the tutorials for independent study.

B. STYLE

When approproiate for the equipment type, the primary style used in designing the tutorials was an adaptation of the "Prompt and Response" style [Ref. 15]. This type of

instruction is designed to prompt the reader to respond to a stimulus presented in a frame type format. As adapted for use in this laboratory, the response is in the form of an action taken by the reader, thus leading the reader through the tutorials in a step-by-step manner. This instructional style provides the reader with immediate feedback concerning the correctness of the action taken. Additionally, this style allows the reader to skip lessons previously covered or undesired, and to review any material covered which is unclear.

C. TUTORIAL USER'S NEEDS

The first step in designing the tutorials was to consider the qualifications and background of the users of the laboratory. Their ability level and background in electronics, mathematics, and computer systems were evaluated so as to design instructional materials best suited to the users' needs and to supplement education received through other courses taken at the Naval Postgraduate School. Because the tutorials were being written primarily for graduate students the authors could assume a high level of scholastic and verbal ability, relatively high motivation (participation in this lab may be voluntary), and varying acquaintance with the terminology and concepts of electronics, mathematics, logic design, and microcomputer theory. There are no prerequisites for the material presented in this lab. It was designed to be studied independently or in conjunction with courses such as CS2810, CS3010, CS3030, CS3200, IS2000, IS3100, IS4183, and others.

D. TUTORIAL DESCRIPTION

Before the authours could begin work on the tutorial manuals, they had to learn the equipment and its operations sufficiently well to be able to teach it to others. This task was made more difficult by the poor manuals that accompanied some of the equipment. Having mastered the equipment, the authors then had to become proficient at the creation of programmed texts, and combine the machine skills with the writing skills. The final stage of the labor was the actual creation of the tutorials. A total of 340 manhours were spent on the research and preparation of the 7 tutorial sets for use on the laboratory equipment. A brief description of each tutorial set is listed below.

1. DD-1 Digi-Designer

The Digi-Designer tutorial contains a functional and physical description of the equipment and its use in the design of logic circuits. Included in the tutorial are the following topics:

a. Binary Mathematics

The basic concepts of binary addition, subtraction, and multiplication is provided for those readers who desire to review this topic.

b. Logic Design

A review of the concepts of logic design utilizing AND, OR, XOR, and NAND gates is provided.

c. Karnaugh Maps

The use and techniques of Karnaugh mapping as a tool for reducing Boolean equations are discussed.

d. Laboratory Experiments

Several laboratory experiments are included to familiarize the reader with the Digi-Designer and the physical concepts of logic design.

2. Heathkit Digital Logic Training Device

The tutorial for the Heathkit Digital Logic Training Device was written in the "Prompt and Response" style discussed earlier and contains three sections. Part one of the tutorial is a functional and physical description of the digital console. Part two contains experiments designed to demonstrate correct procedures for operation of the digital console. Part three contains experiments utilizing logic gates. These experiments are designed to provide a basic introduction to logic design concepts and digital logic "breadboarding".

3. Prompt 80

The tutorial for the Prompt 80 computer is a programmed text written in the "Prompt and Response" linear style for ease of use with the computer. Section one of the manual contains a physical and functional description of the Prompt 80 console and peripheral ports. Section two provides instruction on modifying the registers and memory and introduces the reader to the task of entering a machine language program into the computer. In section three, this concept is expanded by showing the reader how to write a machine language program when given an algorithm. Section four contains instruction on the advanced functions of the Prompt 80, reading and writing to a PROM, debugging machine language programs, and some advanced concepts in machine language programming.

4. SDK-85

The tutorial for the SDK-85 was written in the "Prompt and Response" style and contains three sections. Part one is a general description of the SDK-85 computer. Part two contains a component-by-component functional description of the SDK-85. Part three contains assembly language sample programs and explanations of the additional capabilities provided by the 8085 CPU in comparison to the 8080 CPU used in the Prompt 80.

5. Sybex Self-Study Tape Library

Three courses from the Sybex tape library were reviewed and selected for inclusion in the laboratory. The courses selected were:

SB3 - Military Microprocessor Systems
SB5 - Bit Slice
SB7 - Microprocessor Interfacing Techniques

The manual for the tape library contains a description of the library system and, for each course, an outline describing the course goal, the topics, and the material covered within those topics. The brief synopsis of each course allows the reader to review the material contained in the courses and to determine the applicability of the courses to the reader's abilities and field of study.

6. Heathkit H-9 Terminal

The manual for the Heathkit H-9 terminal explains the effects of each of the control keys of the terminal and describes the functioning of the terminal. In addition, the user is taken through a "Prompt and Response" tutorial demonstrating the procedure to utilize the H-9 (via a modem) as a remote terminal for the IBM 3033 computer system located at the NPS W. R. Church Computer Center.

7. Heathkit H-89 Microcomputer

The manual for the Heathkit H-89 Microcomputer explains the general outline of the computer. It describes the steps necessary to boot the CP/M operating system. It also gives a very brief overview of the imbedded commands of CP/M, the utility programs that came with CP/M and the working of the function keys of CP/M as installed in the H-89. No applications software is described in this manual, as that is left to the user to learn. The manufacturer manuals on the interfaces, monitors and other specific electronic issues are available for reference. The style of the tutorial is traditional text.

E. TUTORIAL TESTING

The completed tutorials were tested by a member of the faculty, members of the project design team, and selected "non-technical" students of the Naval Postgraduate School The experience level of the evaluators ranged from readers with little or no knowledge of microcomputer systems to those who were highly experienced in the concepts covered in the laboratory. During the testing, weaknesses noted in the tutorials were evaluated and the tutorials were modified for improvement and re-evaluated. Both experienced and inexperienced evaluators were able to complete the tutorials with little difficulty. The inexperienced evaluators were generally impressed that the tutorials were not written in a highly technical language, thus providing a better conceptual understanding for them. It was judged by these evaluators that the laboratory could provide a useful and worthwhile learning experience.

F. TUTORIAL CONSIDERATIONS

The authors found that one of the most difficult tasks was to insure that all the tutorials were consistent for style and format. With three authors and seven different manuals it was not a trivial task to make them so. The authors attempted to keep some consistency by proofreading each other's work, using conferences to decide format policy and through intense communications. This sharing of the labor and talent made it possible to achieve the consistency demonstrated in the appendices.

IV. LESSONS LEARNED FROM THE INSTALLATION

A. PROBLEM AREAS ENCOUNTERED

In reviewing the original work requests and discussing the initial project with the sponsor, the authors discovered several items that had not been originally considered in those requests, as well as areas in which technological changes had made the plans obsolete. For example, the countertop height for the laboratory as specified was too high for comfortable typing for long periods. In addition, there was no provision made for storage of materials in the laboratory, nor for security of the more pilferable items of the laboratory. The laboratory had been designed to be partitioned into two rooms. Lockable cabinets and cipher locks on both the inner and outer doors were also provided. Although this change was made late in the development cycle, and was made to plans that had been approved for two years, the Public Works Department was able to respond to the needs and provide the facility as desired. The division of the room into two work areas complicated the situation by forcing a change in the ventilation system of the area to provide exhaust and inflow to both areas. Again, the response of Public Works was gratifying.

The enthusiastic response of Public Works to the changing requirements was not entirely without pitfalls. Some problems developed in the actual execution of the design of the cabinets and countertops. The authors were able to correct the communications failures by personal intervention, and the ultimate product was most suitable for the purpose.

In general, the difficulties faced in the production effort of the physical facility all stemmed from poor communication on the parts of both the transmitting individual and the receiving individual. In those areas where communication was clear and effective, no delays or errors were made. Once again, communications have proven to be critical in program development. Also, the authors appreciate the fact thano complex project can be implemented without some degree of uncertainty and ambiguity. That is, some aspects must be learned by actually forging ahead, doing the work and obtaining experience. Hindsight then allows one to state how the project could have been done "perfectly".

B. PITFALLS TO BE AVOIDED

It is impossible to rigidly define the areas in which any project will experience delays and failures, principally because the conditions in which the project is undertaken will be unique. It is, however, possible to identify the general areas in which close personal supervision will help avoid some of the pitfalls and failures. In a most general way, the communications mentioned above apply to all of the areas in which failures occur.

Communications failures can occur between any of the elements of the design and production team: the design supervisor, his workers, the supervisor of the producing workers, and the actual technicial staff performing the physical work. Failures at the junction of design supervisor and his workers leads to mis-designed or inappropriatly designed plans. Failures at the junction of design and production teams can lead to incorrectly followed plans or improperly drawn plans followed to the letter. Failures between production supervisors and their workers results in improperly installed facilities or delays in installation.

It is not possible to overemphasize the need for accurate, timely and clearly understood communications between all members of the team.

Another area of concern for the designer of a similar laboratory must be the compatability of the planned equipment. In the NPS Instructional Laboratory the original plan was to use the Heathkit H-9 Terminal both with the H-8 computer and as an additional terminal for ARPANET and the IBM 3033 at NPS. Unfortunately, the design of the H-9 terminal makes that impossible, in that the terminal has no capability for lower case characters and if a lower case ASCII code is received, the terminal displays a control character in its place. This discrepancy was discovered when the terminal was first used on the ARPANET, and has made the terminal less attractive than it might otherwise have been. Designers of laboratories similar to this one should carefully screen all hardware for full compatability, including the obtaining of manuals in advance, if that is what is required to investigate fully the capabilities and limitations of a machine.

V. FUTURE PLANS FOR THE LABORATORY

A. SUPPORT OF COURSES AT THE SCHOOL

There are two areas of concern for the immediate future of the instructional laboratory: the use of the laboratory and the equipment to be added to the laboratory. The first issue to be discussed is the future use of the laboratory.

It is the intention of the school to increase the usage of the laboratory in direct support of classwork in both the Information Systems course area and in the Computer Science course area. In particular there are seven courses in which the laboratory can be made an integral part of the coursework and to which the laboratory represents a significant improvement in facilities. Each of these courses will be discussed in detail below. It is recognized that the use of a physically small laboratory to assist in teaching classes with sometimes as many as 40 students or more may be fraught with problems of crowding and scheduling clashes. However, the fact that the laboratory is available to the user twenty-four hours a day, seven days a week, all year long, significantly reduces the problem of crowding and scheduling to a lesser problem of having some users come to the laboratory at unconventional hours. As discussed in the initial chapter, the motivation of the students at NPS is high, and the maturity of the students makes it possible to accept the smallness of the laboratory and still use it as a primary teaching aid.

The lowest level course to be supported in the laboratory is IS 2000, Introduction to Computer Management. The NPS Catalogue [Ref. 12] states that this course covers the elementary hardware and software concepts of Computer

Management. In IS 2000 the laboratory can be used to introduce to the novice student the various terms of data processing, with a chance for the student to actually see and use computers for perhaps the first time. The equipment in the laboratory that might be used very well in this course includes the Digi-designer, the SDK-85, the Prompt 80, the Heath 8 microcomputer, and the Heath 89 microcomputer. Although the instructional material for those devices may well be more advanced than the student needs at this level, the devices can be used to introduce the concepts of registers, memory, storage devices and media, hexidecimal, octal, and binary arithmetic, busses, CPU, "chips", etc.

The IS 3100 course, Survey of Contemporary Computer Systems, has as part of its course coverage the comparison of microcomputers and their price and performance characteristics. With the Heath 89 as a demonstration of a relatively advanced 8-bit machine and the USCG system as an example of a typical 16-bit system, the laboratory can demonstrate the change in performance and price which occurs across this range of computers.

IS 3220, Computer Center Operations, was designed to be taught using the W.R. Church Computer Center as a training site for the student to actually manage a large system. However, with the recent installation of the new IBM 3033 equipment at the Center, that arrangement has been eliminated. In its place, the laboratory can be used to simulate a large computer center. All the roles in a typical large center can be emulated in the laboratory, and problems placed before the students to solve pertaining to allocation of assets and priorities, production scheduing and control, operational procedures, and computer performance evaluation can all be taught through simulation. The use of the laboratory in this way is virtually open-ended. Its success

depends entirely on the response of the students and the inventiveness of the simulation designers.

Application of Database Management, IS 4183, can be assisted in this laboratory in both the use of microcomputing in database management and in the teaching of relational database management systems. The laboratory has CONDOR (tm) relational data base management software available for the student to experiment with and actually see a database system at work. It is possible that problems in database design could be given with the assets of the laboratory available for the student to use in their solution. In addition, role playing could also be used to demonstrate to the students the functions of a database administrator in a simulation environment.

The final Information Systems course to be supported directly is IS 4185, Computer-based Information Systems. In this course the student is presently required to prepare a small decision support system for part of the course credit. The laboratory can be used as a resource for that project, as well as a demonstration site for microcomputer-based decision support and management information systems.

In the Compter Science course area, there are two courses that the laboratory could support. One of these is CS 3010, Computing Devices and Systems. In this course the student is taught computing at the bits and bytes level, with emphasis on the hardware and the interconnections between hardware, rather than on software. The Digi-designer, logic trainer and both the Prompt 80 and the SDK-85 will be significant teaching tools in this course. The lower level devices can be used to teach the concepts of logical circuits, while the SDK-85 and Prompt 80 can be used to demonstrate the effect of clock pulses, timing circuits, and data transfers. In the latter part of the course, the Heath 89 can be used in the final integration of the logical

training and to help the student see that the principles of the lower machines apply equally to the higher.

Finally, CS 3020, Software Design, can use the laboratory as an asset with high level languages such as FORTRAN, COBOL, etc., to assist the student in the design of software that meets the currently accepted criteria of modularity, changeability, etc.

B. HARDWARE PLANS FOR THE LABORATORY

Most of the initial equipment has already been installed in the laboratory. However, several pieces of equipment will be ordered, or are at NPS and not installed. These include a Heath 8 microcomputer to be used with the Heath 9 terminal, an IBM Personal Computer, an Apple microcomputer, a microcomputer development system, a small local network and a microcomputer interfacing system. In addition, more software is planned, including some of the more recently developed electronic spreadsheets, some other database management systems and perhaps different operating systems. With the state of the art in microcomputers in such flux, the present plans are to remain flexible, and to add to the laboratory whatever hardware and software seems to be gaining acceptability in the Department of Defense (DoD) as a whole, with a view to keeping the student and faculty abreast of those developments.

C. CONCLUSION

The laboratory has been a long time in development, and during its development technological changes have provided new opportunities for upgrading its technical capabilities. However, now that the laboratory is a reality, it will be maintained with the latest in hardware and software, and should be used by the faculty and students of NPS for

research to benefit the DoD as a whole. That use alone will justify its existance, but the more important use of the laboratory is in the instructional mode, to reduce the mystique of computers mentioned in Chapter 1, and to provide to the DoD Information Systems specialists who can bridge the communications gap between the technician and the manager.

LIST OF REFERENCES

1. Wight, C., *The Executive's New Computer--Six Keys to Systems Success*, p. 45, Reston, 1972

2. Zahay, D., "Carving a Systems Niche", *Datamation*, Volume 27, Number 2, pp. 100-104, February, 1981

3. Cash, J. I. Jr., McFarlan, F. W., and McKenny, J. L., *Corporate Information Systems Management: Text and Cases*, Richard D. Irwin, Inc, 1983

4. Zahay, p. 100.

5. Couger, J. Daniel, "Improving the Effectiveness of Campus Recruiting", *Computing Newsletter for Instructors of Data Processing*, Volume XV, No. 8, p. 1, April, 1982.

6. Ibid., p. 4.

7. Mein, W. J., "On the Need for Students to Present Technical Material to Non-technical Audiences in a Computer Science Curriculum", *SIGSCE BULLETIN*, Volume 14, Number 1, pp. 97-101, February, 1982

8. Lucas, H. C. Jr., "Preparing Executives for Corporate Information Management", *Infosystems*, pp. 114-117, October, 1979.

9. Association for Computing Machinery, *ACM Recommended Curricula for Computer Science and Information Processing Programs in Colleges and Universities, 1968-1981*, 1981, p. 57.

10. Ibid., p. 77.

11. Ibid., p. 5.

12. Naval Postgraduate School Catalogue, Naval Postgraduate School, 1982-83.

13. Cook, R. N., "A Hardware Course for a Software Curriculum", *SIGSCE BULLETIN*, Volume 13, number 2., pp. 17-22, June, 1981.

14. Milner, Stuart D., "How To Make The Right Decisions About Microcomputers", Instructional Innovator, v. 25, no. 6, pp. 12-19, September 1980, p. 13.

15. Markle, Susan M., Good Frames and Bad: A Grammar For Frame Writing , New York: Wiley, 1964.

APPENDIX A

DD-1 DIGI-DESIGNER TUTORIAL

```
  *********************************
 ***********************************
*************************************
***                               ***
***   INSTRUCTIONAL LABORATORY    ***
***                               ***
***       DD-1 DIGI-DESIGNER:     ***
***                               ***
***                               ***
***                               ***
***          LOGIC CIRCUIT        ***
***                               ***
***     DESIGN METHODOLOGIES      ***
***                               ***
*************************************
 ***********************************
  *********************************
```

TABLE OF CONTENTS

Section 1

DD-1 DIGI-DESIGNER

1.1 INTRODUCTION

This series of instruction is designed to teach the reader to understand and to design logic circuits utilizing the DD-1 Digi-Designer and logic components. A review of the basic mathematics requirements is also provided.

1.2 DD-1 DESCRIPTION

The DD-1 Digi-Designer, produced by E & L Instruments Incorporated, is a complete digital circuit design instrument that will meet your requirements for digital circuit design laboratory experiments. It will handle both Integrated Circuit (IC) and discrete components without soldering: connections are made using any 22 gage insulated wire. The unit includes a regulated 5 volt (+5V) direct current power supply, a selectable frequency clock (pulse generator), dual bounce-free pushbuttons (pulsers), four switches for applying voltage or ground as required, four Light Emitting Diode (LED) logic lamp monitors, and the SK10 universal component socket.

1.3 SK-10 SOCKET DESCRIPTION

The SK10 socket is basically a matrix of 64 pairs of common contacts (5 per strip) arranged symmetrically; combined with 8 buss strips running along the length of the socket (40 contacts per strip). The socket allows the user to insert all electronic components required for the experiments with lead diameters up to 20 gage wire. For very large components, the E & L BP24 adapter pins, which accept leads up to 16 gage wire, should be used. When inserting DIP ICs, be certain to preset the leads at the correct spacing. Insert one side partially in, then roll the second set of leads into the other side , then press squarely down seating the IC properly.

1.4 MANUAL DESCRIPTION

This manual is primarily an adaptation of the notes and labs for the Hewlett Packard HP 5035T Logic Lab as taught in the EE-2810 course at NPS. The manual contains some useful review information on binary mathematics, logic design, Karnaugh Maps, theory behind flip-flop circuits, and some specification sheets for TTL ICs. Following the review information, you will find five laboratory experiments that should prove helpful in making use of the Digi-Designer to conveniently design, assemble, and test relatively complex circuits, without soldering, and in only a few minutes. It should be noted that since the advent of large scale integration, the types of design involved in these experiments are not the main concern of computer system designers; however, these experiments can be very useful in learning the basic concepts involved in logical and digital circuit design.

1.5 GENERAL LABORATORY INSTRUCTIONS

If you desire to make notes of experimental results or to answer the questions contained within the experiments, please obtain a copy of the experiment you wish to perform.

DO NOT WRITE ON THE PAGES WITHIN THIS MANUAL!

Be sure to inventory all of the Digi-Designer equipment when checking the equipment in or out. An inventory sheet and the components necessary for completing the experiments are contained in this manual following the introduction.

1.6 EQUIPMENT INVENTORY FOR DIGI-DESIGNER LAB EXPERIMENTS

QUANTITY	ITEM
1	DD-11 DIGI DESIGNER
1	MANUAL
2	7400 QUAD 2-INPUT POSITIVE NAND GATES
2	7402 QUAD 2-INPUT POSITIVE NOR GATES
2	7408 QUAD 2-INPUT POSITIVE AND GATES
2	7432 QUAD 2-INPUT POSITIVE OR GATES
1	7404 HEX INVERTER
1	7486 QUAD 2-INPUT POSITIVE XOR GATE
2	7483 DUAL JK MASTER/SLAVE FLIP-FLOPS
2	7474 DUAL D TYPE EDGE TRIGGERED FLIP-FLOPS
1	7411 TRIPLE 3-INPUT POSITIVE AND GATE
1	7420 DUAL 4-INPUT NAND GATE
1	7442 BCD-TO-DECIMAL DECODER DRIVER
1	7482 2-BIT BINARY FULL-ADDER
10	330 OHM RESISTORS (1/4 WATT)
2	ALLIGATOR CLIP JUMPER LEADS
1	IC EXTRACTOR CLIP
	ASSORTED PIECES #22 WIRE

Section 2

BINARY MATHEMATICS--TWO'S COMPLEMENT ARITHMETIC

For purposes of the following example, we will use a register length of four bits. Three bits are necessary to represent the numbers zero through seven in binary form and the additional bit is used to represent the sign of the number, whether negative or positive. The left-most digit is the most significant digit (MSD) and the right-most digit is the least significant digit (LSD).

```
Most Significant Digit<-------------->Least Significant Digit

           (MSD)---->|A1 | A2 | A3 | A4 |<----(LSD)
                     |   |    |    |    |
                     ---------------------
                        3    2    1    0
         weights---->  2    2    2    2
```

The binary representation for -8 through +7 is as follows:

7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

Notice that all negative numbers have a MSD of value 1. This bit is known as the sign bit.

Note also, that $A + (-A) = 0$.

For example,

$$\begin{array}{rcr} 6 & & 0110 \\ +(-6) & & +1010 \\ \hline 0 & 1 \longleftarrow & 0000 \end{array}$$

In this case, the 1 carried out of the register is lost leaving 0000 as the (correct) result of the calculation.

The largest possible positive number is $0111 = 2^3 - 1$ (in general, $2^{(k-1)} - 1$, where k is the register length). The negative number with the greatest magnitude is $1000 = -2^3$ (in general, $-(2^{(k-1)})$).

It is easy to see, by example, that to complement a binary number (i.e. to change it's sign) we have only to complement every bit (four in the example above), and add one to the result. For example:

$$6 = 0110$$

$$-6 = -0110 = (1001 + 0001) = 1010$$

where 1001 is the bit by bit complement.

The result is called the <u>two's complement</u>. (The bit-by-bit complement is called the <u>one's complement</u>.) The process for forming either one's or two's complements is easily implemented in computer hardware.

2.1 BINARY ADDITION

Two binary numbers are added (as in the above example) just as one would add decimal numbers, except that we use the binary "addition tables",

BINARY ADDITION TABLES

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ \underline{+0} & \underline{+1} & \underline{+0} & \underline{+1} \\ 0 & 1 & 1 & 10 \end{array}$$

However, because of the finite (4-bit) register length, there are five cases we must consider. Illustrations of the five cases are given on the following page.

(a) $A,B > 0;\ (A+B) \le 2^3-1$

3	0011
+2	+0010
5	0101

(b) $A,B > 0;\ (A+B) > 2^3-1$

3	0011
+6	+0110
9	1001

↑
Overflow

(c) $A > 0;\ B < 0$

3	0011
+(-6)	+1010
-3	1101

Overflow cannot occur
when A and B have
opposite signs.

(d) $A,\ B < 0;\ (A + B) > -2^3$

-3	1101
+(-2)	+1110
-5	1<--- 1011

Carry-out is lost;
4-bit answer is
correct.

(e) $A,B < 0;\ (A+B) < -2^3$

-3	1101
+(-6)	+1010
-9	1<--- 0111

Carry-out is lost;
Overflow condition;
Compare with (d).

Clearly cases (b) and (e) lead to erroneous 4-bit answers. We say that <u>overflow</u> has occurred. Computer hardware (a logic circuit) must be used to detect this condition.

2.2 BINARY SUBTRACTION

To compute A - B we actually calculate A + (-B). That is, we take the 2's complement of B and then add. The addition hardware is unchanged -- including the overflow indicator.

2.3 BINARY MULTIPLICATION

One method for accomplishing multiplication could be by repeated addition. This method would be very slow for large numbers. An alternative is to use the shift-and-add-method. For example, consider the unsigned binary multiplication of the decimal problem 11 x 13 = 143. The binary representation for 11, 13, & 143 is 1011, 1101, & 10001111 respectively. The multiplication is performed as follows:

	Set memory cell for C equal to zero
1011=A	
x 1101=B	
1011	B_0 = 1, so add A to C (C previously zero)
0000	B_1 = 0, so shift A but don't add to C
1011	B_2 = 1, so shift A and add to C
1011	B_3 = 1, so shift A and add to C
C = 10001111	
	The product is now in memory cell for C

We can detect a 1 (or a 0) in each successive digit of B by ANDing B with a shifting MASK. Thus,

B		MASK		
1101	•	0001	≠ 0, so B_0 = 1	Initial mask = 0001
1101	•	0010	= 0, so B_1 = 0	Mask shifted left
1101	•	0100	≠ 0, so B_2 = 1	Mask shifted again
1101	•	1000	≠ 0, so B_3 = 1	And again

Section 3

LOGIC DESIGN

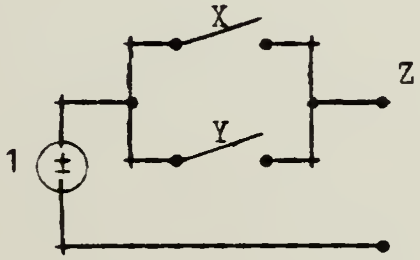
3.1 INTRODUCTION

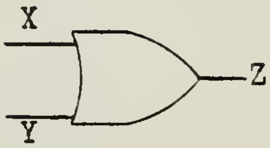
Our objective is to develop procedures for the design of logic networks which will perform specified logical tasks (e.g. to turn on a light in your home from either of two switches).

3.2 SWITCHING ALGEBRA

The basis for such design is Boolean algebra (1847), which was applied to switching circuits in 1938. You will already appreciate, for example, that

Input		Output
X	Y	Z=X+Y
0	0	0
0	1	1
1	0	1
1	1	1





all represent "logical addition" or the "OR switching operation". (An open switch is a "0"; a closed switch is a "1".)

So our first task is to postulate the properties of switching algebra (which is one possible Boolean algebra).

We postulate that each variable can take only two values, 0 or 1, and that the fundamental operations are negation, logical sum, and the logical product.

NEGATION		LOGICAL SUM			LOGICAL PRODUCT		
X	$\bar{X}$	X	Y	$Z=X+Y$	X	Y	$Z=X\bullet Y$
0	1	0	0	0	0	0	0
1	0	0	1	1	0	1	0
		1	0	1	1	0	0
		1	1	1	1	1	1

From these postulates the following theorems may be proven:

$$1 + X = 1 \tag{1}$$

$$0 \bullet X = 0 \tag{2}$$

$$0 + X = X \tag{3}$$

$$1 \bullet X = X \tag{4}$$

$$X + X = X \tag{5}$$

$$X \bullet X = X \tag{6}$$

$$\bar{X} + X = X + \bar{X} = 1 \tag{7}$$

$$\bar{X} \bullet X = X \bullet \bar{X} = 0 \tag{8}$$

$$\bar{\bar{X}} = X \tag{9}$$

$$X + Y = Y + X \tag{10}$$

$$X \bullet Y = Y \bullet X \tag{11}$$

$$X + (Y + Z) = (X + Y) + Z \tag{12}$$

$$X \bullet (Y \bullet Z) = (X \bullet Y) \bullet Z \tag{13}$$

$$X + (X \bullet Y) = X \tag{14}$$

$$X \bullet (X + Y) = X \tag{15}$$

$$X \bullet (Y + Z) = X \bullet Y + X \bullet Z \tag{16}$$

$$(X + Y) \bullet (X + Z) = X + Y \bullet Z \tag{17}$$

$$X + (\bar{X} \bullet Y) = X + Y \tag{18}$$

$$X \bullet (\bar{X} + Y) = X \bullet Y \tag{19}$$

Problem 1

Use truth tables and the postulates to prove

(i) Theorem (1) above,

(ii) Theorem (7) above,

(iii) Theorem (14) above,

(iv) Theorem (17) above.

Problem 2

Show the switch arrangement corresponding to:

(i) $X \bullet Y$ (Hint: See the diagram for $X+Y$ above.)

(ii) $1 + X$ (What switch position corresponds to 1?)

(iii) $X + \overline{X}$.

Problem 3

Use switch diagrams to show that $X+(X \bullet Y)=X$

Finally we can list two theorems due to DeMorgan:

$$\overline{X + Y} = \overline{X} \bullet \overline{Y} \tag{20}$$

$$\overline{X \bullet Y} = \overline{X} + \overline{Y} \tag{21}$$

These too may be proved with the help of truth tables. They are easily extended to more variables; e.g.

$$\overline{X + Y + Z} = \overline{X} \bullet \overline{Y} \bullet \overline{Z}.$$

3.3 SWITCHING FUNCTIONS

Consider the statement,

$$F = AB\overline{C} + A(\overline{B} + C) + \overline{A}$$

We say that F is a function of the three variables A, B, and C. (A, B, and C are "input" or "independent" variables,

each of which can take on the values 0 or 1 independently of the others.) For any combination of values of A, B, and C (e.g. 1 0 1) we could evaluate the value of the function, F.

A switching function is a (boolean) algebraic statement, and, like any other algebraic expression, can often be simplified by applying the appropriate theorems. For example,
(*We will omit the "•" which represents the AND operation)

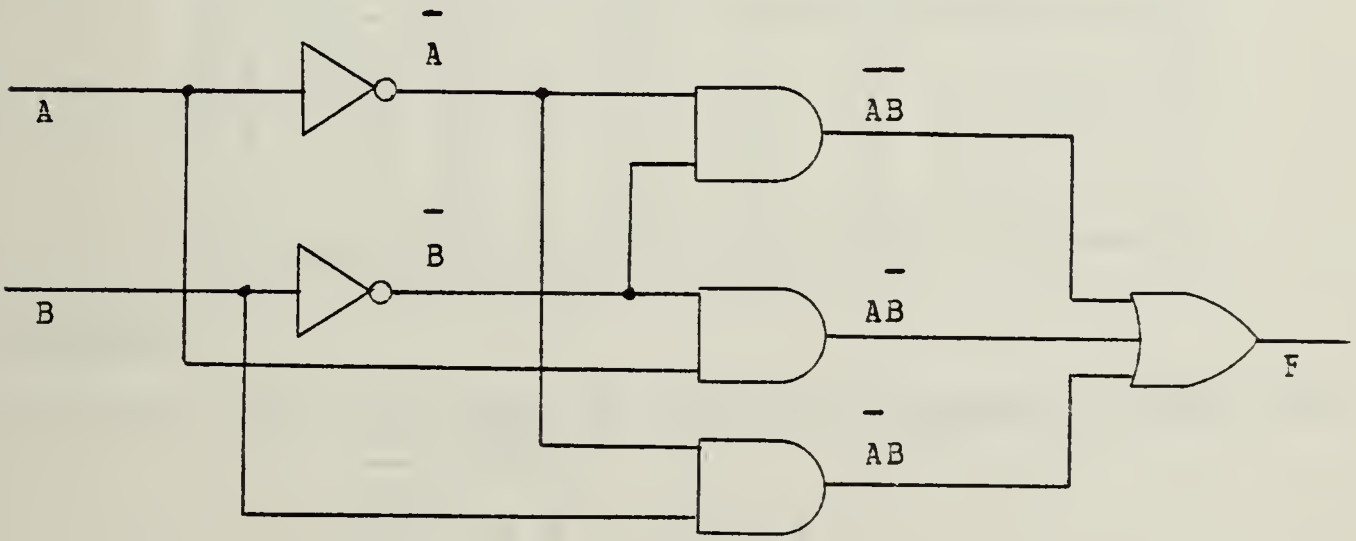
$$\bar{A}\bar{B} + A\bar{B} + \bar{A}B = \bar{B}(\bar{A} + A) + \bar{A}B \qquad \text{(Theorem 16)}$$

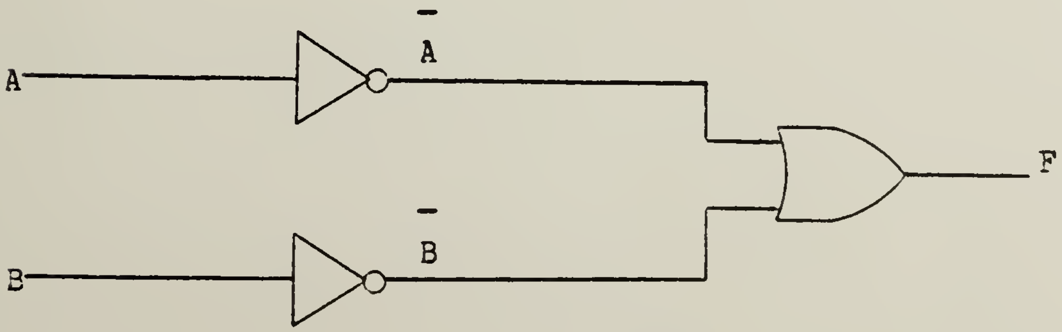
$$= \bar{B}1 + \bar{A}B \qquad \text{(Theorem 7)}$$

$$= \bar{B} + \bar{A}B \qquad \text{(Theorem 4)}$$

$$= \bar{B} + \bar{A} \qquad \text{(Theorem 18)}$$

Let us now draw a logic circuit corresponding to this switching function, both before and after simplification.





Clearly we have saved three AND gates.

Problem 4

Simplify the circuit even further, by using a NAND gate.

Problem 5

Apply the rules (theorems) of switching algebra to simplify:

$$WXYZ + W\bar{X}YZ + \bar{W}XYZ.$$

Then draw the logic circuits for the given and the simplified logic functions.

Problem 6

Simplify $AB\bar{C} + A(\bar{B} + C) + \bar{A}$. Check your result by completing the following truth table.

A B C	$\bar{B} + C$	$A(\bar{B} + C)$	$AB\bar{C}$	$\bar{A}$	F
0 0 1	1	0	0	1	1
0 1 0	0	0	0	1	1
0 1 1	1	0	0		

Problem 7

Use the rules (theorems) of switching algebra to show that:

$$\overline{\bar{A}B + A\bar{B}} = \bar{A}\bar{B} + AB$$

<u>Comments</u>:

(i) You may work on either or both sides of the equation algebraically until you get equality between left and right sides.

(ii) There are many ways that you could do this.

(iii) One hint, which makes it easier to apply the <u>very useful</u> DeMorgan theorems, is to complement both sides of the equation:

$$\overline{\overline{\bar{A}\,B + A\,\bar{B}}} = \overline{\bar{A}\,\bar{B} + A\,B}$$

Then start algebraic manipulation according to the rules.

(The left-hand side becomes simply $\bar{A}\,B + A\,\bar{B}$, while you can apply DeMorgan's theorem to the right-hand side.)

3.4 LOGIC SPECIFICATIONS

Often the logic specifications will be in the form of an English language statement describing the desired objective. For example,

> On a democratic desert island, three people, A, B, and C decide to build a voting machine. The inputs are A, B, and C (which will be 1 for a yes vote and 0 for a no vote). The output must light an LED marked M if the majority vote yes and must light an LED marked U (as well as M) if they vote yes unanimously.

Problem 8

Complete the truth table for this device (three input variables A, B, and C, and two output functions M, and U).

Inputs			Outputs	
A	B	C	M	U
0	0	0	0	0
0	0	1	0	0
0	1	0		

As another example of a performance specification, consider the following.

> The seat-belt interlock system for a two-seat automobile is to prevent starting unless the driver and the passenger (if any) are buckled in. The state of the passenger's seat-belt is to have no effect if there is no passenger.

The first step in setting up the truth table is the definition of the appropriate variables. Here, for example,

W = 1 if there is a passenger,

W = 0 if no passenger

B_d and B_p = 1 if seat-belts on,

B_d and B_p = 0 if seat-belts not connected

F, the output function, must be 1 if the car is to be allowed to start.

Problem 9

Set up the truth table for this seat-belt interlock system.

3.5 IMPLEMENTATION OF A LOGIC OR SWITCHING FUNCTION

Given that we have obtained a truth table corresponding to the logic specifications, the next step is to design an appropriate logic circuit. Here we will demonstrate some systematic approaches to this problem.

3.5.1 The Sum-of-Products Form of the Logic Function

Suppose that we have obtained the following truth table (logic specification) for some system.

A	B	C	F	COMMENTS
0	0	0	0	
0	0	1	1	<-- $\bar{A}\,\bar{B}\,C = 1$ only when A=0, B=0, C=1
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	<-- $A\,\bar{B}\,C = 1$ only when A=1, B=0, C=1
1	1	0	1	<-- $A\,B\,\bar{C} = 1$ only when A=1, B=1, C=0
1	1	1	0	

Looking at the rows with 1's in the F column, we can obviously make the statements in the Comments column. It then follows that

$$F = \bar{A}\,\bar{B}\,C + A\,\bar{B}\,C + A\,B\,\bar{C}$$

will be 1 only if A = 0, B = 0, C = 1 or if A = 1, B = 0, C = 1 or if A = 1, B = 1, C = 0. That is, the expression for F, above, will have the same truth table as does F in the logic specification.

Problem 10

Obtain the truth table for $F = \bar{A}\ \bar{B}\ C + A\ \bar{B}\ C + A\ B\ \bar{C}$, and confirm that it agrees with the table above.

A logic function in the form of our example, $F = \bar{A}\ \bar{B}\ C + A\ \bar{B}\ C + A\ B\ \bar{C}$, is for obvious reasons called the <u>sum-of-products form</u> of the logic function. If you now reread the present section you should have no difficulty in seeing that the sum-of-products form of F can always be written down by inspection of the truth table.

Notice, particularly, that although we looked at only three rows of the above truth table (those rows for which F = 1), the resulting expression for F is correct for <u>any</u> combination of values of A, B, and C. (We made sure that F = 1 for the proper three cases <u>only</u>, and so F naturally took the only other possible value, 0, for all the other cases.)

Problem 11

Obtain the logic function in sum-of-products form for the following system (actually an XOR gate).

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Problem 12

Obtain the sum-of-products form of the logic functions M and U in the voting machine example (Problem 8).

Problem 13

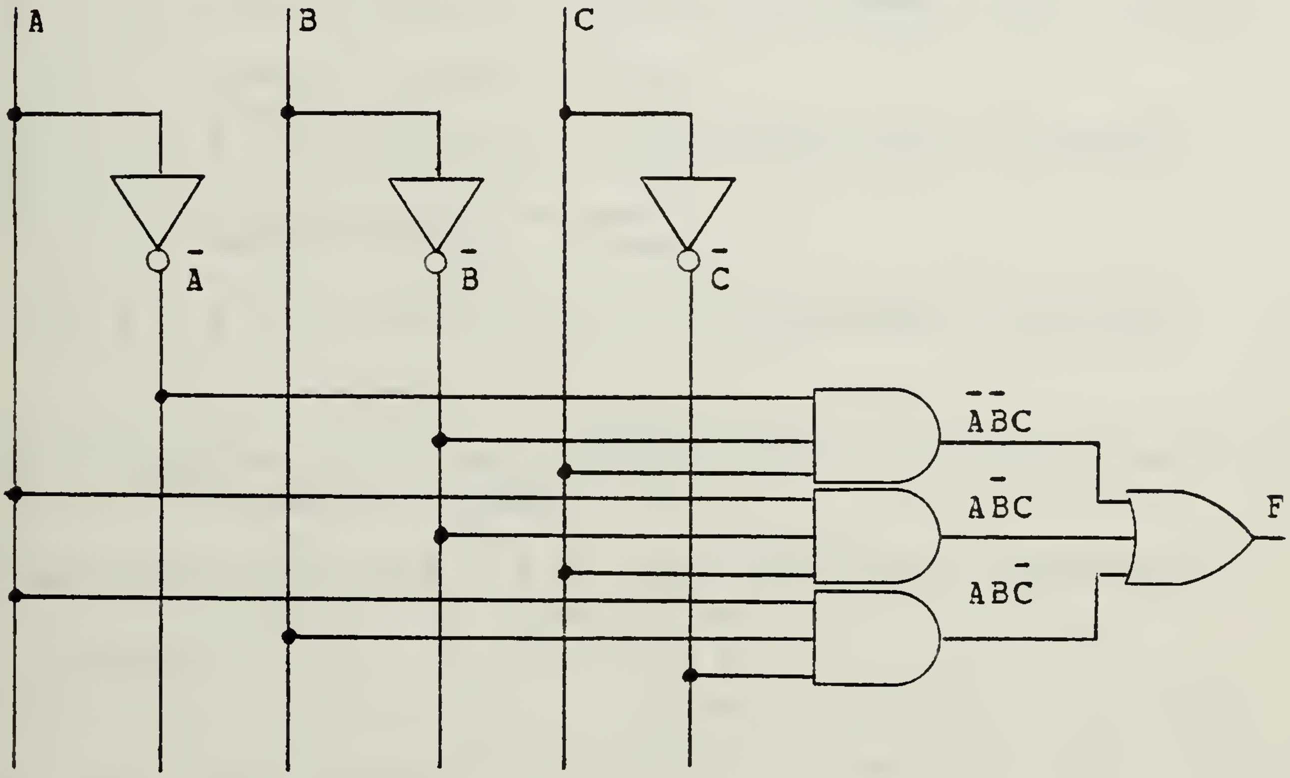
Repeat the sum-of-products form for the seat-belt interlock system (Problem 9).

3.5.2 Implementation of the Sum-of-Products Logic Function

Having found that

$$F = \bar{A}\ \bar{B}\ C + A\ \bar{B}\ C + A\ B\ \bar{C}$$

for the truth table at the start of this section, it is easy to see that the AND/OR/INVERTER implementation is as follows.



Actually, this sum-of-products function can be simplified by Boolean algebra or by Karnaugh mapping (discussed later). That is,

$$F = \bar{A}\ \bar{B}\ C + A\ \bar{B}\ C + A\ B\ \bar{C} = \bar{B}\ C + A\ B\ \bar{C}$$

This is also easily implemented, with some saving of gates.

Problem 14

Starting with the sum-of-products logic functions, obtain the AND/OR/INVERTER implementations corresponding to Problems 11 and 12. Optionally, for Problem 13 also.

3.5.2.1 NAND Gate Implementation

Sometimes it is convenient to use NAND gates only. (We will see in the next section how to use NOR gates only.) Consider the following manipulation.

$$F = \bar{A}\,\bar{B}\,C + A\,\bar{B}\,C + A\,B\,\bar{C}$$

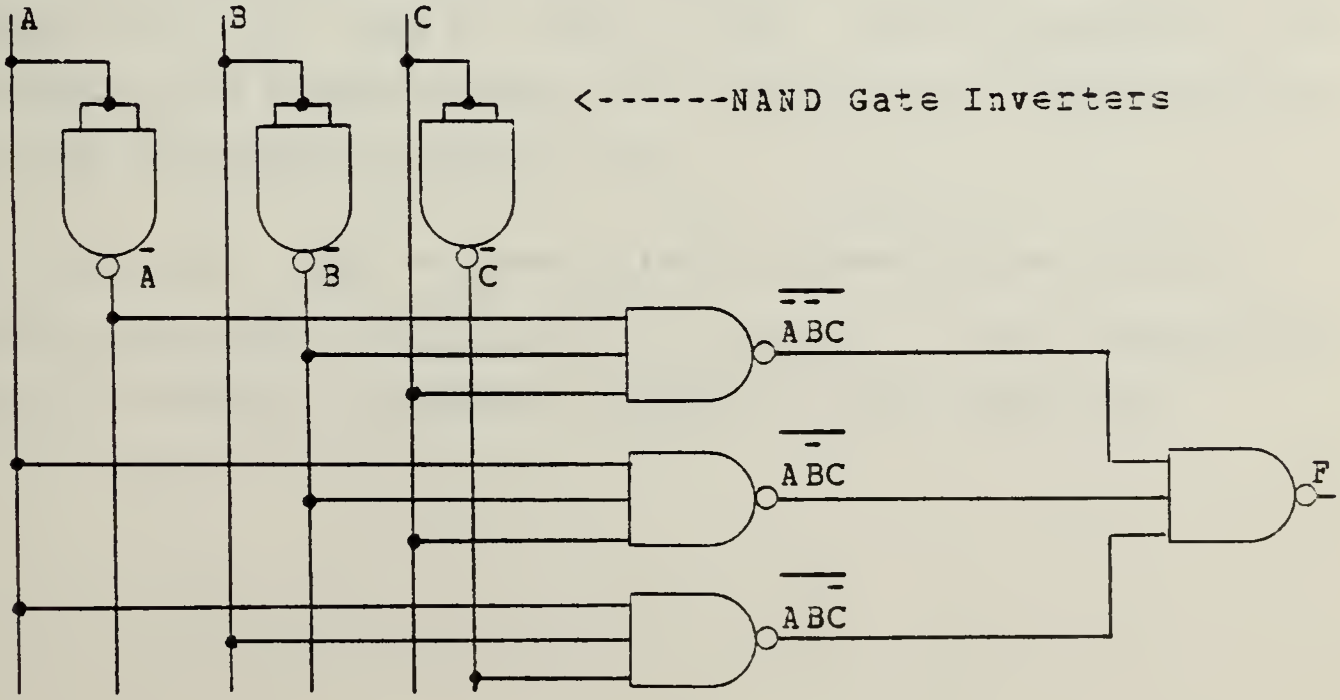
$$\bar{F} = \overline{\bar{A}\,\bar{B}\,C + A\,\bar{B}\,C + A\,B\,\bar{C}} \quad \text{(complement both sides)}$$

$$= \overline{\bar{A}\,\bar{B}\,C} \bullet \overline{A\,\bar{B}\,C} \bullet \overline{A\,B\,\bar{C}} \quad \text{(DeMorgan's theorem)}$$

$$\bar{\bar{F}} = F = \overline{\overline{\bar{A}\,\bar{B}\,C} \bullet \overline{A\,\bar{B}\,C} \bullet \overline{A\,B\,\bar{C}}} \quad \text{(complement theorem)}$$

We can recognize $\overline{\bar{A}\,\bar{B}\,C}$ as a NAND operation on $\bar{A}$, $\bar{B}$, and C. Similarly for $\overline{A\,\bar{B}\,C}$ and $\overline{A\,B\,\bar{C}}$. Then the entire expression may be thought of as $F = \overline{a \bullet b \bullet c}$

Which is a NAND operation on $a = \overline{\bar{A}\,\bar{B}\,C}$, $b = \overline{A\,\bar{B}\,C}$, $c = \overline{A\,B\,\bar{C}}$.



It then follows that

$$F = (A + B + C)(A + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + \bar{C})$$

will be 0 if A = 0, B = 0, C = 0 and if A = 0, B = 1, C = 0 and if A = 0, B = 1, C = 1, and if A = 1, B = 0, C = 0 and if A = 1, B = 1, C = 1. That is, this <u>product-of-sums</u> logic function, F, will have the same truth table as does F in the logic specification. (F will be 0 for all the above listed values of A, B, and C only, and must therefore be 1 for all other values of A, B, and C.)

<u>Problem 15</u>

Obtain the truth table for

$$F = (A + B + C)(A + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + \bar{C})$$

and confirm that it agrees with the table above.

If you now reread the present section, you should have no difficulty in seeing that the product-of-sums form of F can always be written down by inspection of the truth table.

Notice that although we looked at only five rows of the above truth table (those rows for which F = 0), the resulting expression for F is correct for any combination of values of A, B, and C. Note, too, that by comparing and combining the present result with that in the first section we have incidentally shown that

$$F = \bar{A}\bar{B}C + A\bar{B}C + AB\bar{C} = (A+B+C)(A+\bar{B}+C)(A+\bar{B}+\bar{C})(\bar{A}+B+C)(\bar{A}+\bar{B}+C)$$

emphasizing that there are, in general, many alternative ways of writing a (Boolean) algebraic logic function.

Problem 16

Obtain the logic function in product-of-sums form for the following system (an XOR gate again).

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Problem 17

Obtain the product-of-sums form of the logic functions M and U in the voting machine example (see Problem 8).

Problem 18

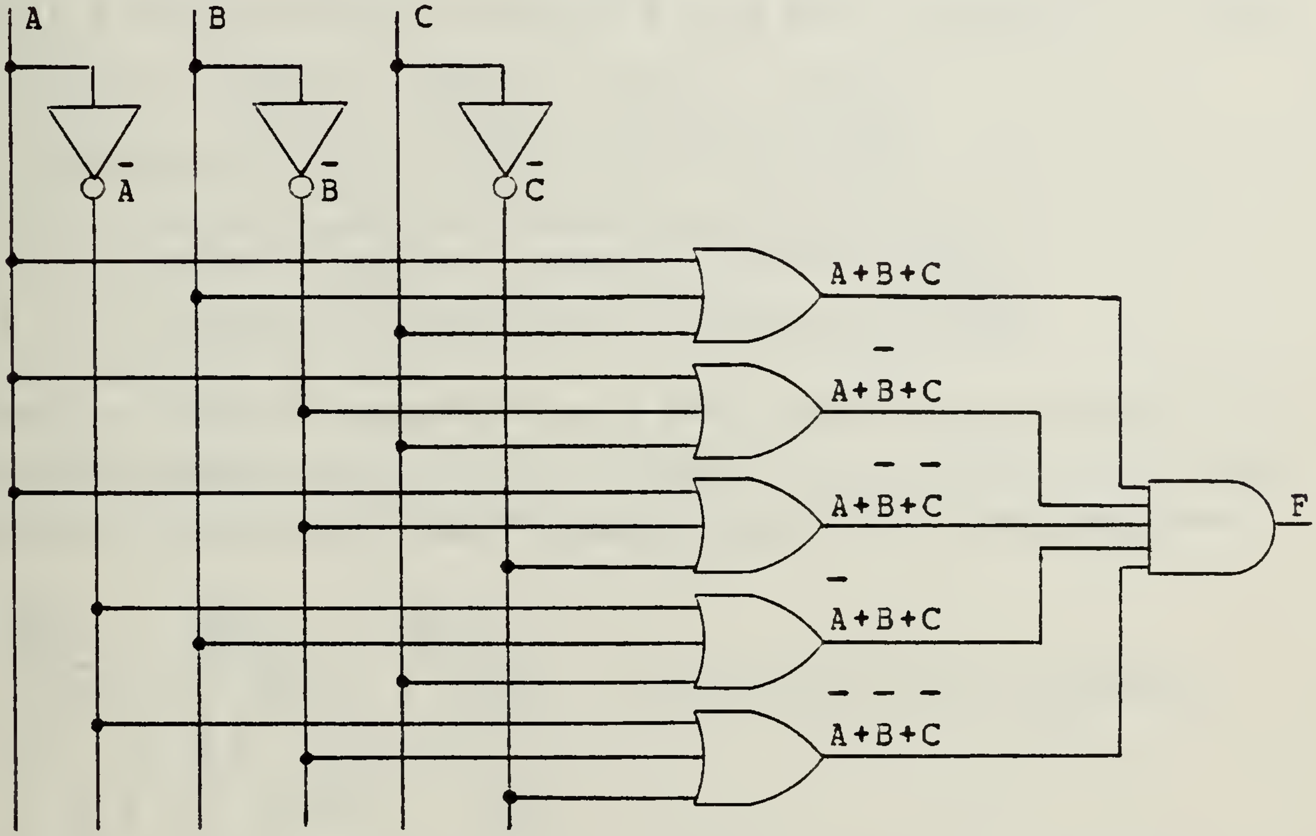
Repeat, for the seat-belt interlock system of Problem 9.

3.5.3 Implementation of the Product-of-Sums Logic Function

Having found that

$$F = (A + B + C)(A + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + \bar{C})$$

for our example system, it is easy to see that the AND/OR/INVERTER implementation is as follows.



Actually, this product-of-sums function can be simplified by Boolean algebra or by Karnaugh mapping (discussed later). That is

$$F = (A + B + C)(A + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + \bar{C})$$
$$= (\bar{B} + \bar{C})(B + C)(A + C)$$

This is much more easily implemented, with a considerable saving of gates. (We would need two inverters, three 2-input OR gates, and one 3-input AND gate.)

Problem 19

Starting with the product-of-sums logic functions, obtain the AND/OR/INVERTER implementations corresponding to Problems 16 and 17. Optionally, for Problem 18 also.

3.5.3.1 NOR Gate Implementation

Sometimes it is convenient to use NOR gates only. Consider the following manipulation.

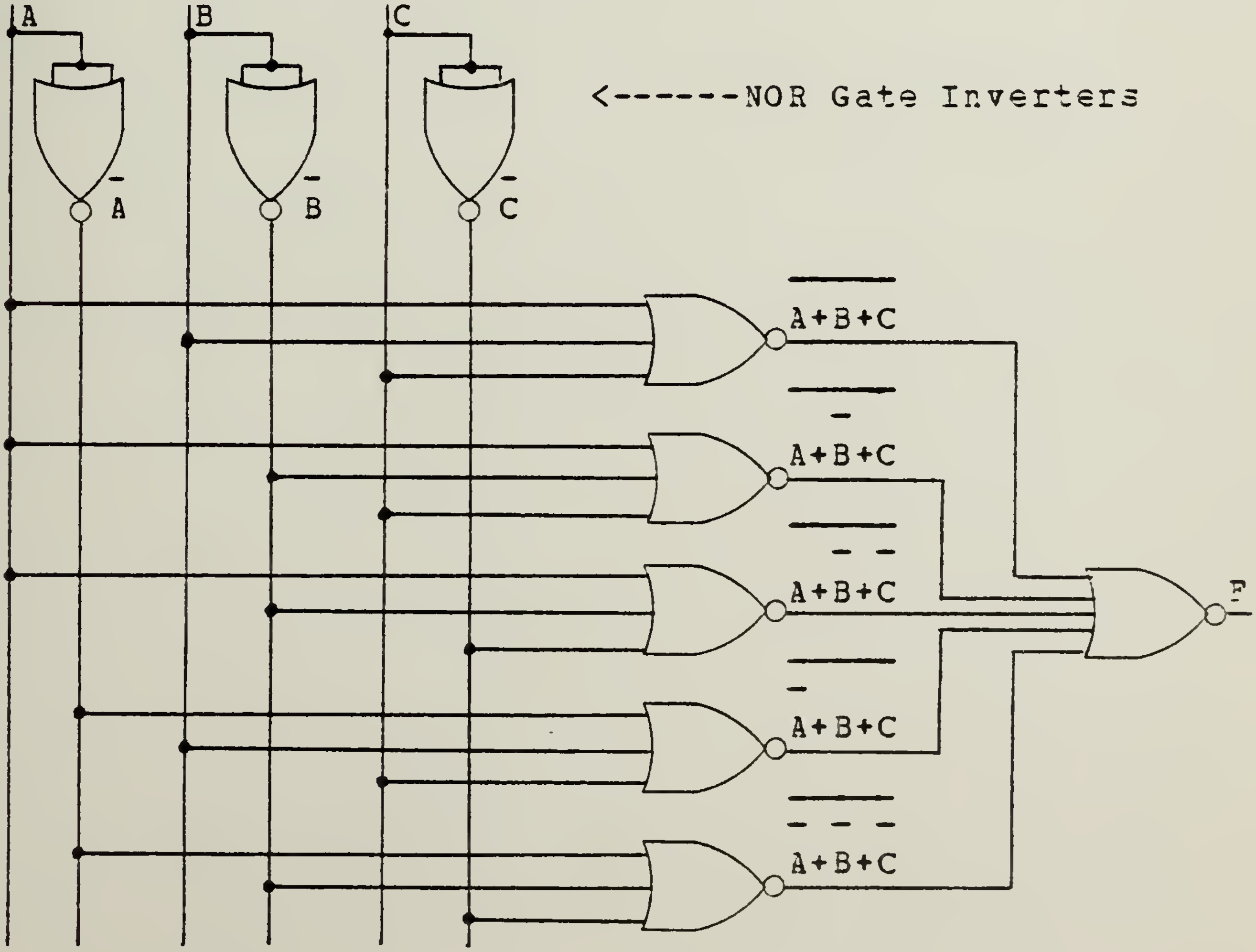
$$F = (A + B + C)(A + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + \bar{C})$$

$$\bar{F} = \overline{(A + B + C)(A + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + \bar{C})}$$

$$= \overline{(A+B+C)} + \overline{(A+\bar{B}+C)} + \overline{(A+\bar{B}+\bar{C})} + \overline{(\bar{A}+B+C)} + \overline{(\bar{A}+\bar{B}+\bar{C})}$$

$$\bar{F} = F = \overline{\overline{(A+B+C)} + \overline{(A+\bar{B}+C)} + \overline{(A+\bar{B}+\bar{C})} + \overline{(\bar{A}+B+C)} + \overline{(\bar{A}+\bar{B}+\bar{C})}}$$

which you should recognize as a NOR operating on each of the bracketed quantities, each of which in turn is a NOR operating on the three "added" quantities within the brackets.



Problem 20

Starting with the product-of-sums logic functions, obtain the NOR gate implementations corresponding to Problems 16 and 17.

Review Question

Consider a binary adder, which will add the nth digit of a binary number A, the nth digit of a binary number B, and the "carry" into the nth position. Let these quantities be: A_n, B_n, and C_n. The sum, s_n, will be 1 if any one of these is 1, or if all three are 1. The carry to the next stage, C_{n+1}, will be 1 if any two or all three are 1.

3.6 ANSWERS TO PROBLEMS

1. (i)

1	X	1 + X
1	0	1
1	1	1

i.e. $1 + X = 1$

(ii)

X	$\bar{X}$	$X + \bar{X}$
0	1	1
1	0	1

i.e. $\bar{X} + X = 1$

(iii)

X	Y	X•Y	X + X•Y
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

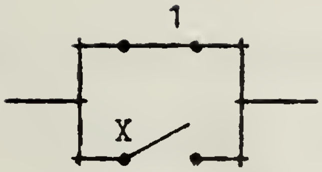
The last column clearly equals X.

(iv)

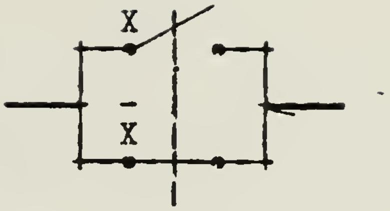
X	Y	Z	X+Y	X+Z	(X+Y) (X+Z)	Y•Z	X+Y•Z
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

equal

2.



1 + X
One switch permanently closed.



X + $\bar{X}$
One switch opens as other closes.

3.

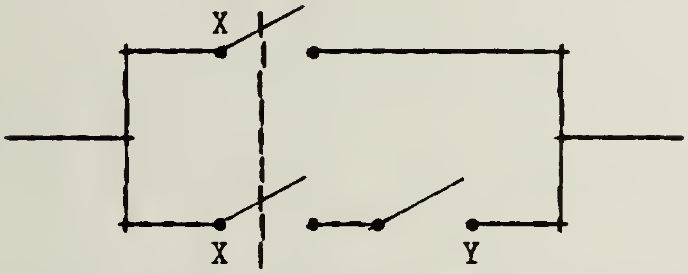
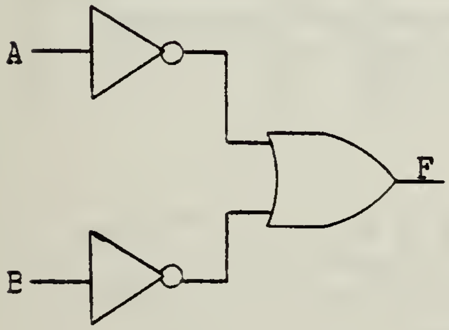


Diagram for X+(X•Y)

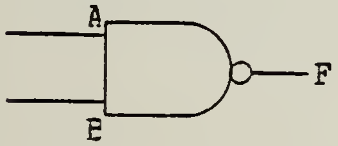
This is equivalent to a short (1) when X = 1 (closed)
and to an open (0) when X = 0 (open)
So the combination is equivalent to X itself

4.



$F = \overline{A} + \overline{B} = \overline{A \bullet B}$
by DeMorgan's theorem

====>



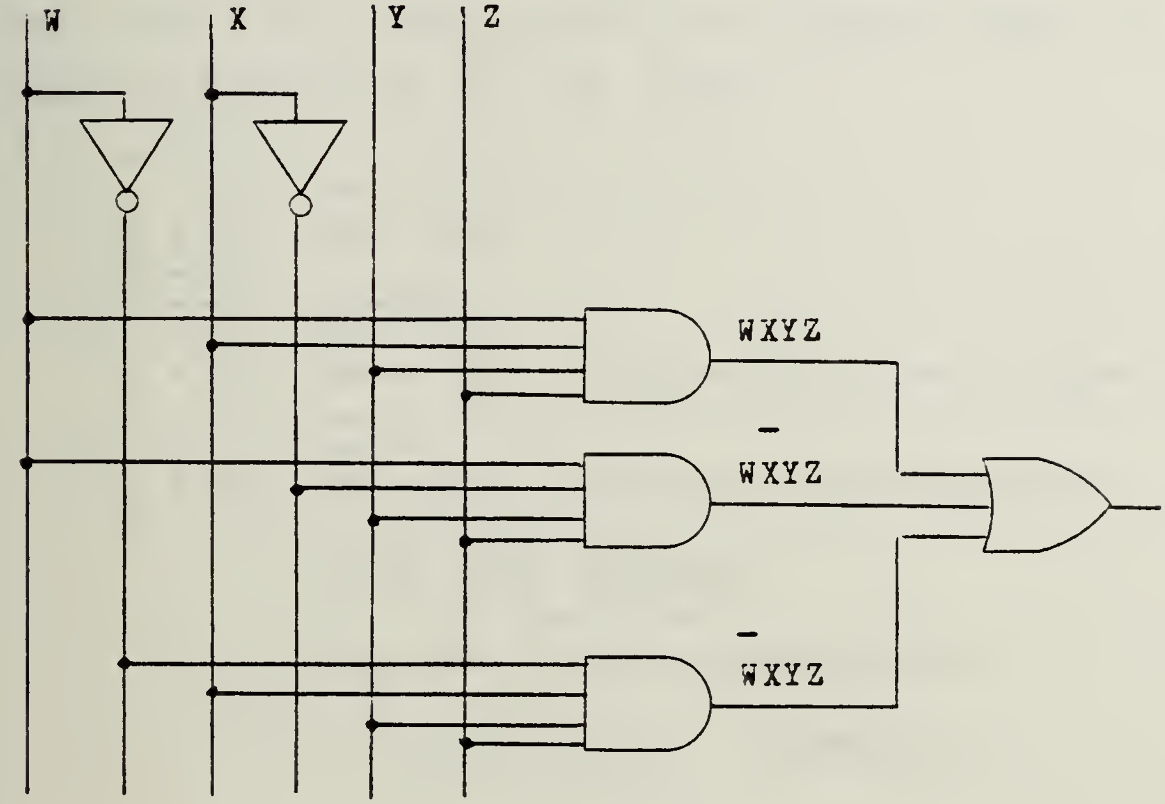
5. $WXYZ + W\bar{X}YZ + \bar{W}XYZ = (WX+W\bar{X}+\bar{W}X)YZ$

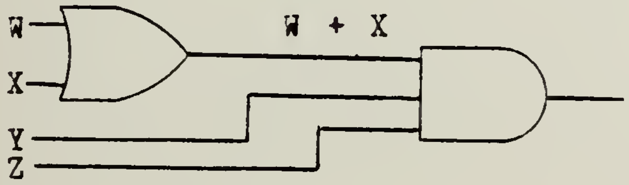
$= [W(X+\bar{X}) + \bar{W}X]YZ$

$= (W + \bar{W}X)YZ$

$= (W + X)YZ$

There are other possibilities.





6. $AB\overline{C} + A(\overline{B}+C) + \overline{A} = A(B\overline{C}+\overline{B}+C) + \overline{A}$

$= A(\overline{B}+\overline{C}+C) + \overline{A}$

$= A(\overline{B}+1) + \overline{A}$

$= A \bullet 1 + \overline{A} = A + \overline{A} = 1$

The truth table should show the function equal to 1 for all eight combinations of A, B, and C.

7.

$\overline{\overline{A}B + A\overline{B}} = \overline{AB} + AB$

$\overline{\overline{\overline{A}B + A\overline{B}}} = \overline{\overline{AB} + AB}$ complementing both sides

$\overline{A}B + A\overline{B} = \overline{\overline{AB}} + \overline{AB}$ Theorem 9 and DeMorgan

$= (\overline{A}+\overline{B})(A+B)$ DeMorgan

$= \overline{A}A + \overline{A}B + \overline{B}A + \overline{B}B$ Theorem 16

$= 0 + \overline{A}B + \overline{B}A + 0$ Theorem 6

8.

A	B	C	M	U
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

9.

W	B_d	E_p	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

10.

A	B	C	$\bar{A}\bar{B}C$	$A\bar{B}C$	$AB\bar{C}$	F
0	0	0	0	0	0	0
0	0	1	1	0	0	1
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	0
1	0	1	0	1	0	1
1	1	0	0	0	1	1
1	1	1	0	0	0	0

11. $F = \bar{A}B + A\bar{B}$

12. $M = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

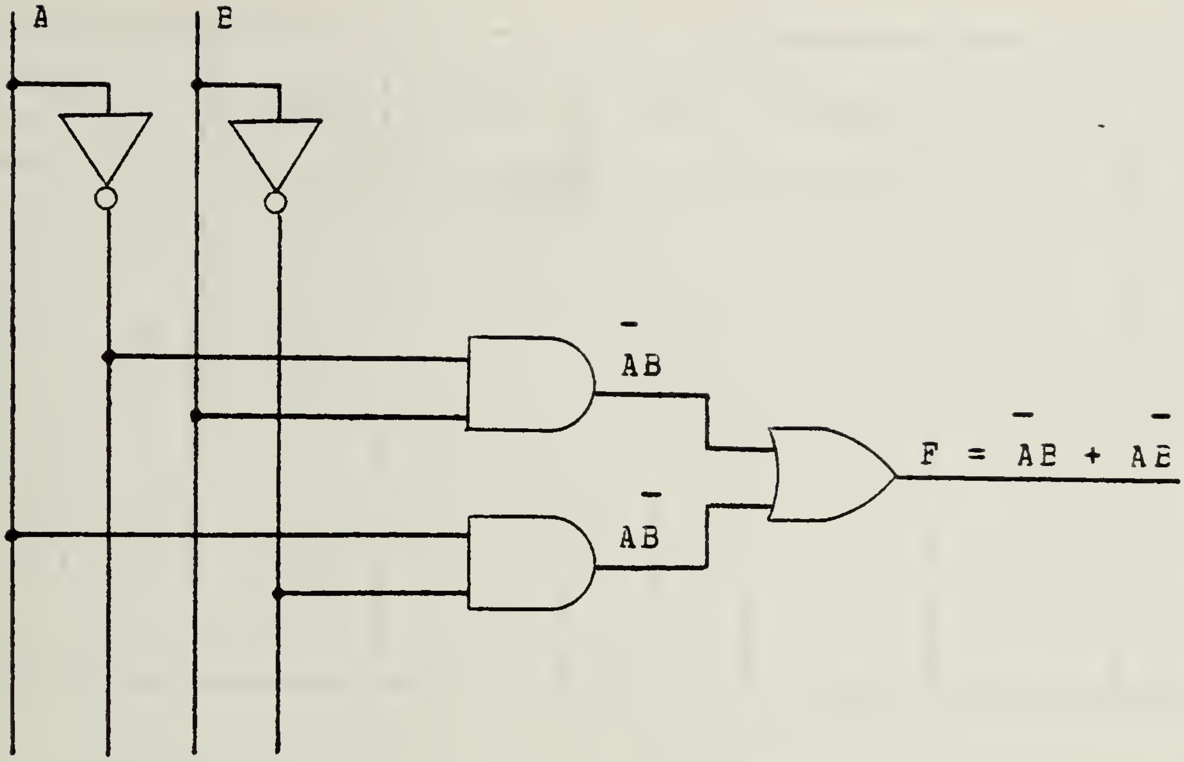
$U = ABC$

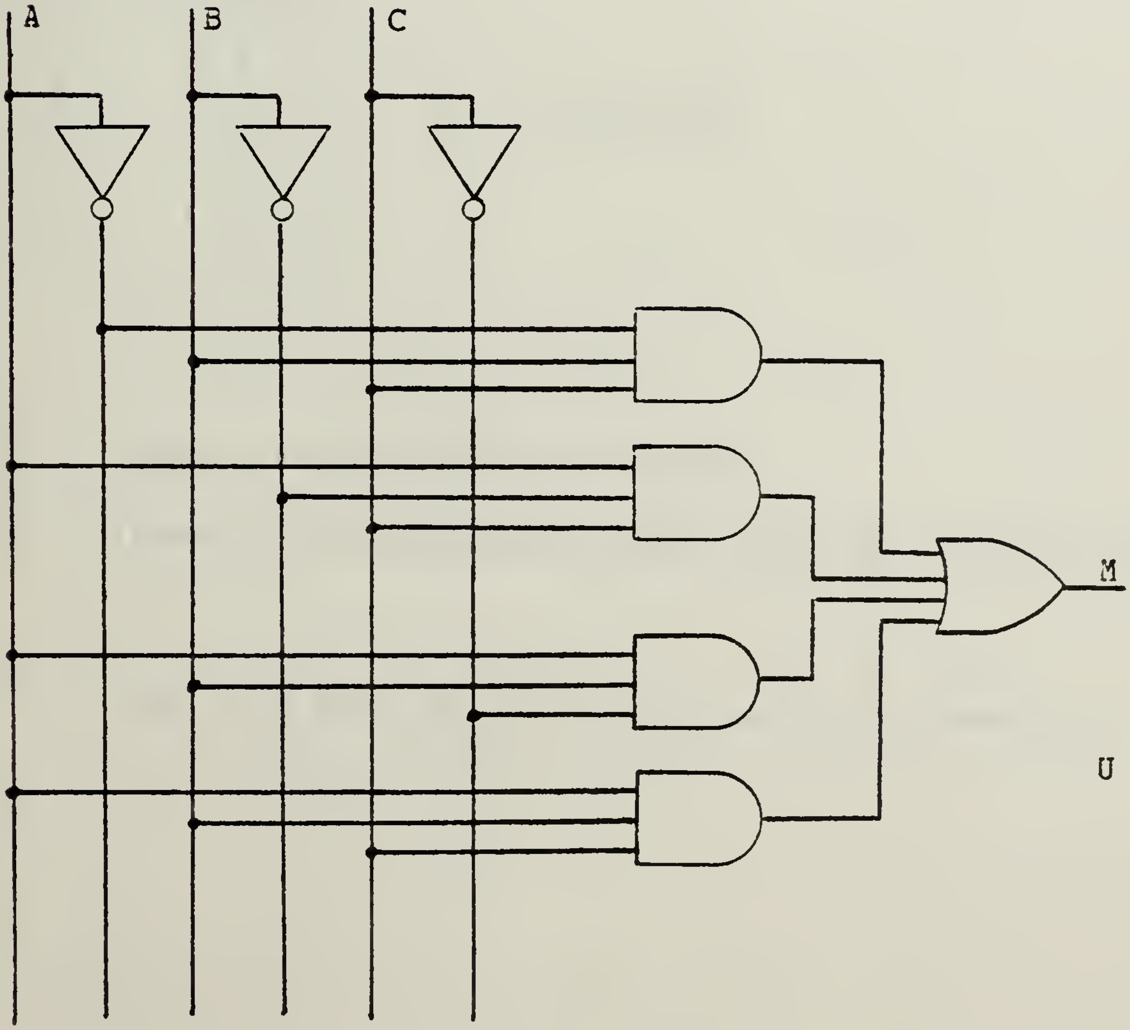
13. (a) $F = \bar{W}B_d\bar{B}_p + W\bar{B}_dB_p + WB_dB_p$

(b) Your answer will depend on your definition of variables. Our answer is

$F = \bar{S}D\bar{T} + S\bar{D}\bar{T} + SD\bar{T}$

14.





15.

A	B	C	$A+B+C$	$A+\bar{B}+C$	$A+\bar{B}+\bar{C}$	$\bar{A}+B+C$	$\bar{A}+\bar{B}+\bar{C}$	F
0	0	0	0	1	1	1	1	0
0	0	1	1	1	1	1	1	1
0	1	0	1	0	1	1	1	0
0	1	1	1	1	0	1	1	0
1	0	0	1	1	1	0	1	0
1	0	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	0	0

16.

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

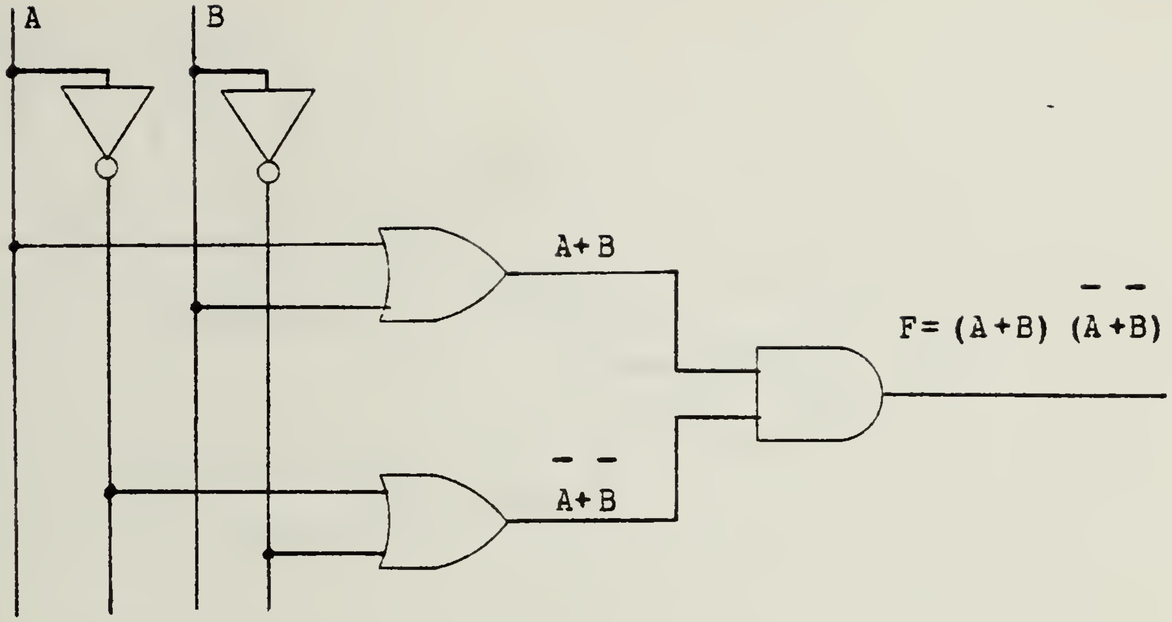
$F = (A+B)(\bar{A}+\bar{B})$

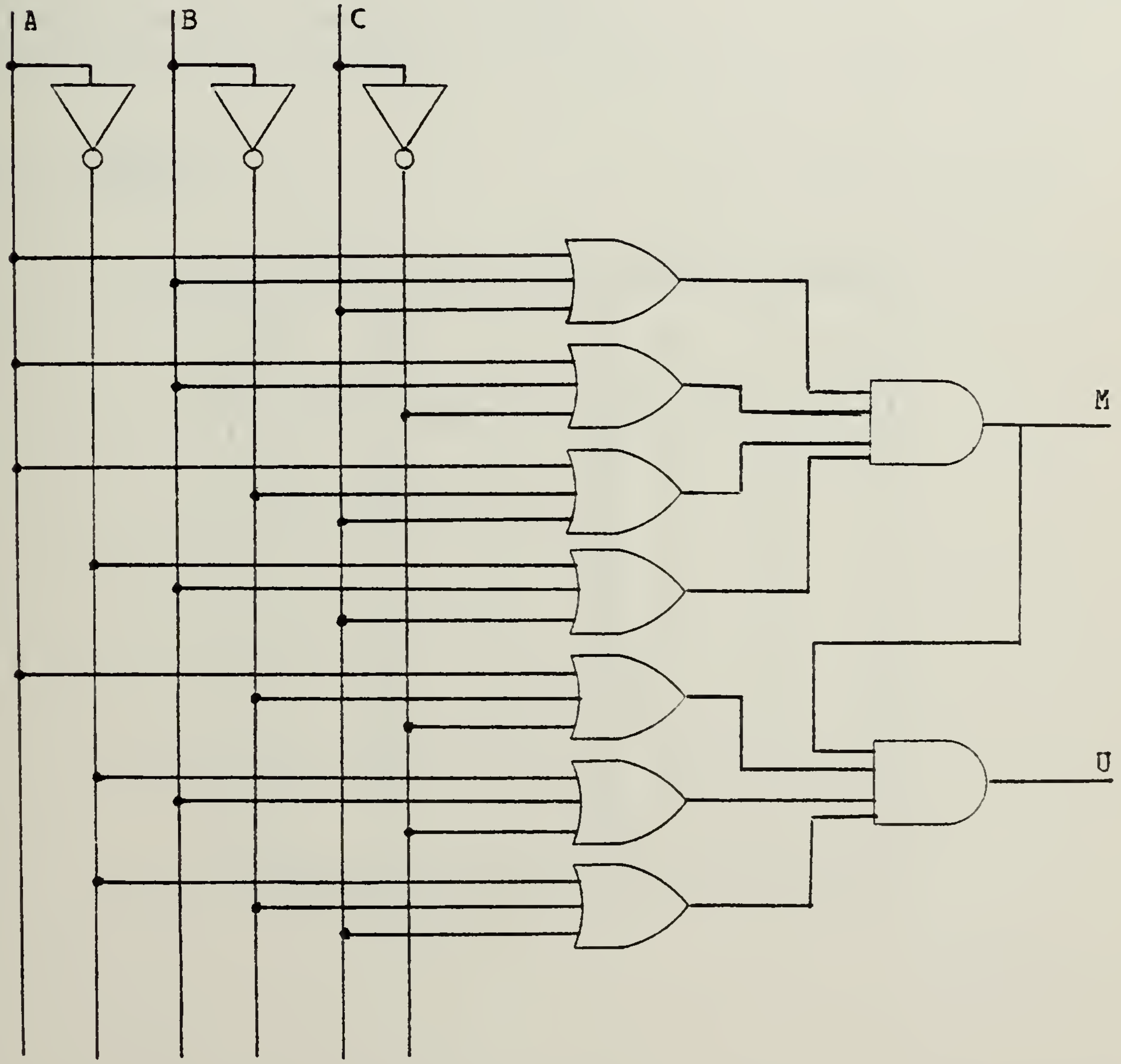
17. $M = (A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(\bar{A}+B+C)$

$U = (A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(A+\bar{B}+\bar{C})(\bar{A}+B+C)(\bar{A}+B+\bar{C})(\bar{A}+\bar{B}+C)$

18. $F = (W+B_d+B_p)(W+B_d+\bar{B}_p)(\bar{W}+B_d+B_p)(\bar{W}+B_d+\bar{B}_p)(\bar{W}+\bar{B}_d+B_p)$

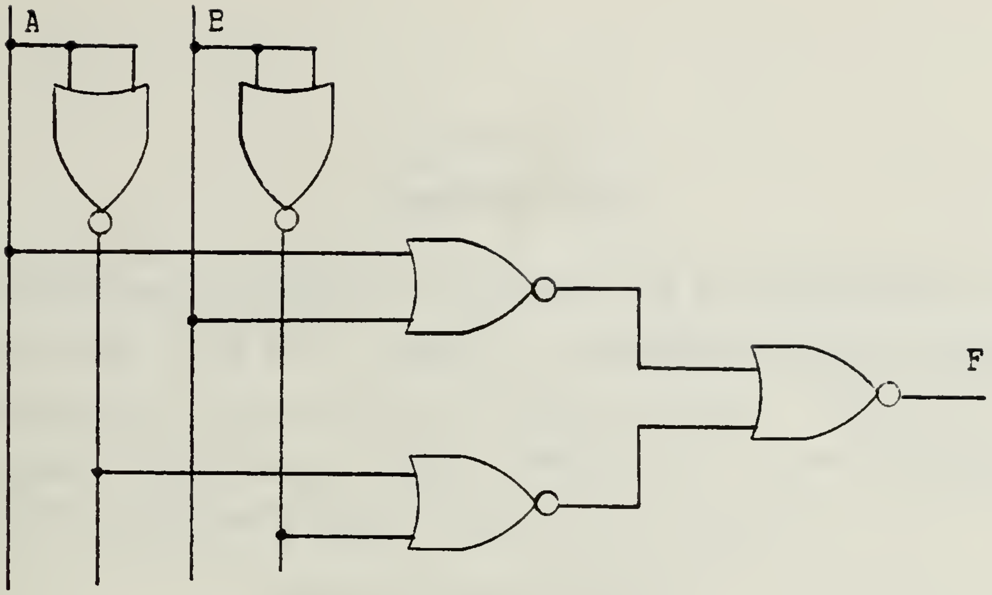
19.





NOTE There is no need to repeat the upper four OR gates to obtain U.

20.



Similarily for the voting machine.

Review Question.

A_n	B_n	C_n	S_n	C_{n+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	0	1
1	1	1	1	1

Section 4

KARNAUGH MAPS

Karnaugh maps visually portray the properties of Boolean functions and can be used to systematically simplify the combinational logic circuits (functions). To start with, we will assume that the logic function to be simplified has been put in standard sum-of-products form.

Example: $F = \bar{A} \bullet \overline{[(B + C) \bullet D] \bullet \bar{A}}$

To put this in sum-of-products form we set up the truth table.

A	B	C	D	B+C	(B+C) •D	$[(B+C)\bullet D]\bullet\bar{A}$	$\overline{[(B+C)\bullet D]\bullet\bar{A}}$	F
0	0	0	0	0	0	0	1	1
0	0	0	1	0	0	0	1	1
0	0	1	0	1	0	0	1	1
0	0	1	1	1	1	1	0	0
0	1	0	0	1	0	0	1	1
0	1	0	1	1	1	1	0	0
0	1	1	0	1	0	0	1	1
0	1	1	1	1	1	1	0	0
1	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	1	0
1	0	1	0	1	0	0	1	0
1	0	1	1	1	1	0	1	0
1	1	0	0	1	0	0	1	0
1	1	0	1	1	1	0	1	0
1	1	1	0	1	0	0	1	0
1	1	1	1	1	1	0	1	0

It now follows, from the last column, that

$$F = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}C\overline{D} + \overline{A}B\overline{C}\,\overline{D} + \overline{A}BC\overline{D}$$

There are 16 possible input combinations of a function of four variables: $A\ B\ C\ D$, $A\ B\ C\ \overline{D}$, $A\ B\ \overline{C}\ D$, $A\ B\ \overline{C}\ \overline{D}$,..., of which only five appear in the sum-of-products expression. Each possible input combination is called a minterm.

A Karnaugh map for a function of four variables is merely a four by four table of 16 squares or cells, one for each minterm.

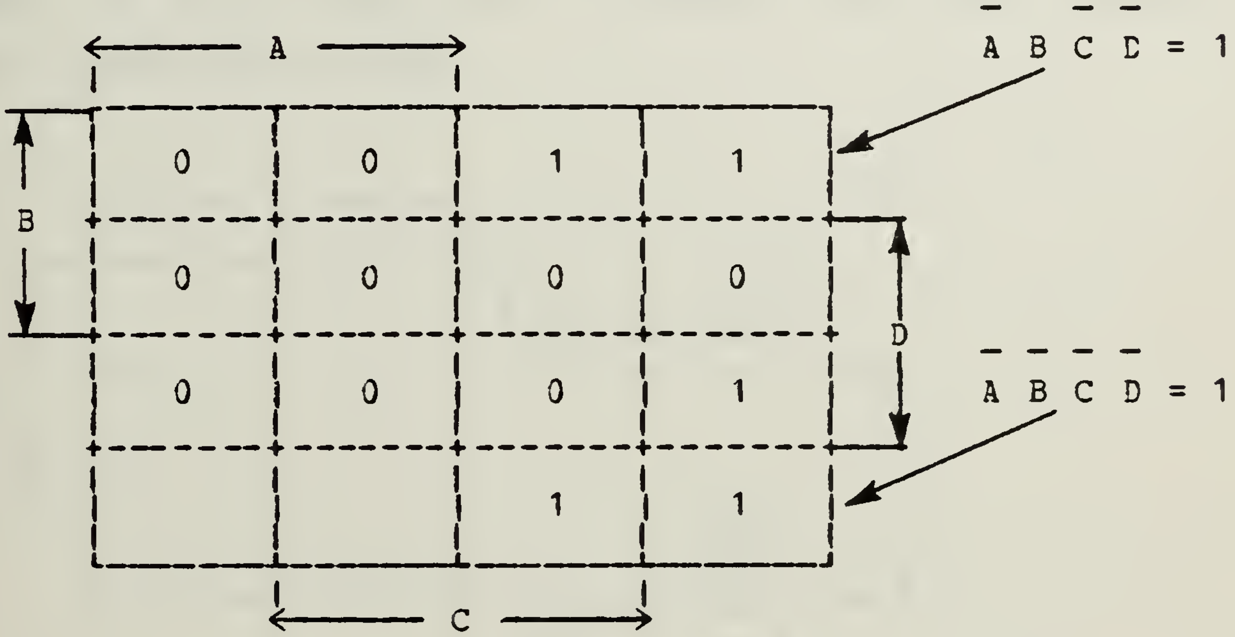


Figure 1

The Karnaugh map is a function of four variables (for example, Figure 1) is divided into regions A, B, C, and D as shown. This clearly implies the assignment of regions $\overline{A}$, $\overline{B}$, $\overline{C}$, and $\overline{D}$ also. Then each cell will correspond to one minterm, whose value may be entered in the cell as in Figure 1.

Figure 1 has a "cyclic" structure. If you were to wrap the diagram around a vertical cylinder, the two isolated $\overline{C}$ columns would join and become adjacent. Similarly, if the map were wrapped around a horizontal cylinder, the two

isolated $\bar{D}$ columns would join and become adjacent. The importance of these observations will become apparent in a moment.

The next step is to group adjacent 1-cells -- cells containing 1's. Each group must be "rectangular", and must contain exactly $2^{(k)}$ 1-cells. That is, 1x1, 1x2, 1x4, 2x2, 2x4,...groupings are legitimate, a 1x3 group or an odd-shaped group is not. It is possible that a given 1-cell will be a member of more than one group. There will, in general, be more than one way to group the 1-cells. For our purposes we will find that we should use the largest possible legitimate groups.

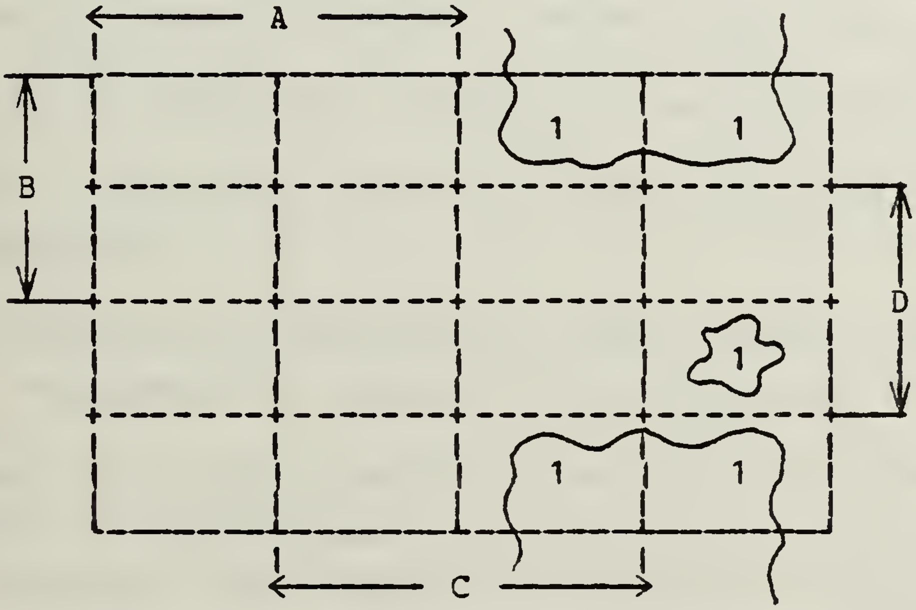


Figure 2

The five 1-cells correspond to the five minterms in the above sum-of-products expression for the example function, F. In Figure 2, the 1-cells corresponding to the above example have been grouped (circled). Note that the cyclic property has been used. The two 1-cells in the top row are adjacent to the two 1-cells in the bottom row.

The grouping focuses attention on particular cells. With With this grouping, we can do nothing with the minterm $\overline{A}\overline{B}\overline{C}D$, which appears in a group of 1. But consider the 4 minterms in the other group:

$$\overline{A}BC\overline{D} + \overline{A}B\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}\overline{C}\overline{D} = \overline{A}B\overline{D}\,(C + \overline{C}) + \overline{A}\overline{B}\overline{D}\,(C + \overline{C})$$

$$= \overline{A}B\overline{D} + \overline{A}\overline{B}\overline{D} = \overline{A}\overline{D}\,(B + \overline{B})$$

$$= \overline{A}\overline{D}$$

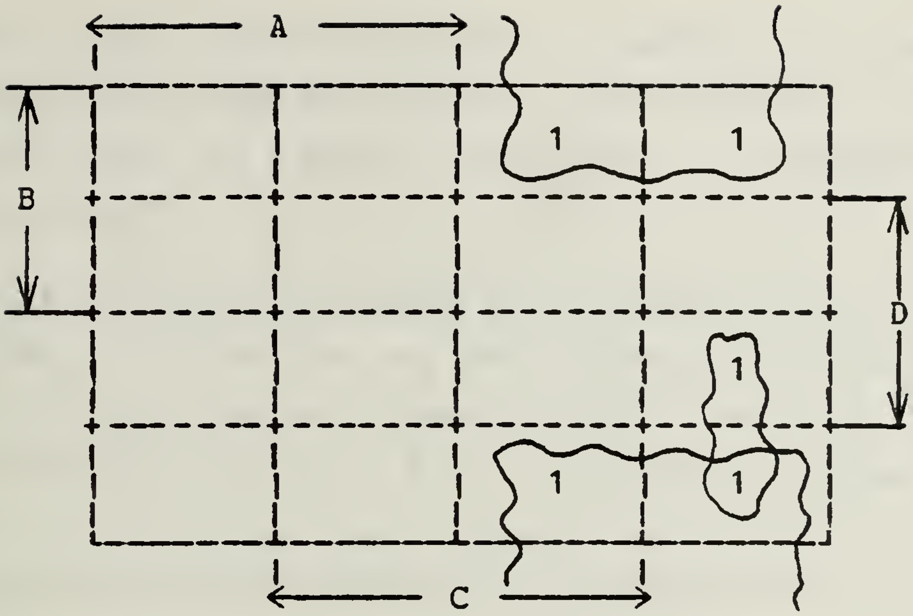
A considerable simplification! Thus we can now write a more simplified expression for the function F in the example:

$$F = \overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}\,\overline{D}$$

Actually, the Karnaugh map allows us to skip the above algebra of simplification. ** Note that the 2x2 group of Figure 2 is entirely in the $\overline{A}$ region and entirely in the $\overline{D}$ region. So we can now write down the simplification $\overline{A}\,\overline{D}$ by inspection!

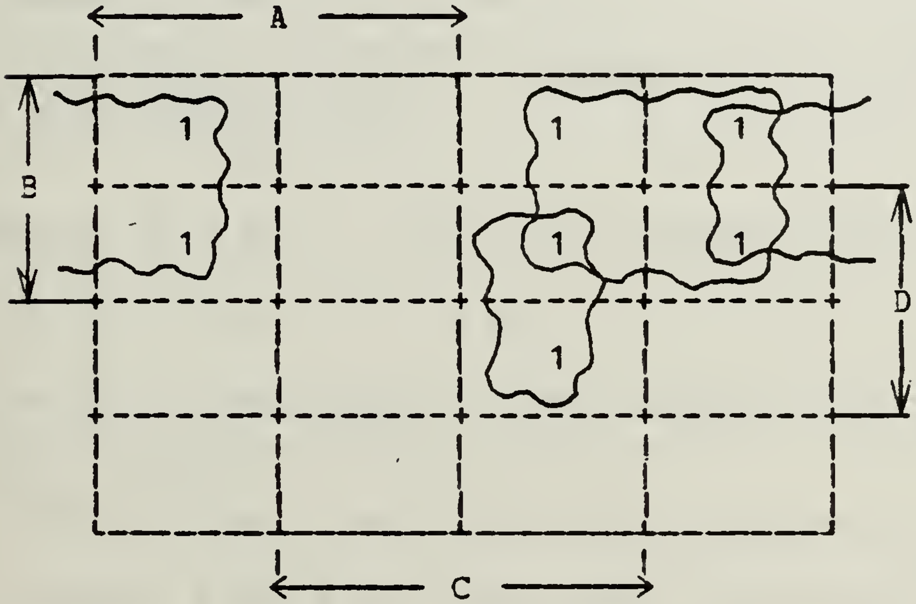
Let's examine the logic behind this "rule". Recall that any 1-cell represents a minterm of the function, F. For example, in Figure 2 the third cell in the top row represents $\overline{A}\,B\,C\,\overline{D}$ This will be unity when, A = 0; B = 1; C = 1; and D = 0. Now, suppose you wished to locate the 1-cells which will contain 1's when A=0 *and* D=0 *regardless of the values of B and C.* This describes all the cells in the region common to A and $\overline{D}$, that is the 2x2 group of Figure 2.

Some other examples are in order. We will assume that the functions have been put into sum-of-products form, and will only show the maps and the corresponding simplified functions.

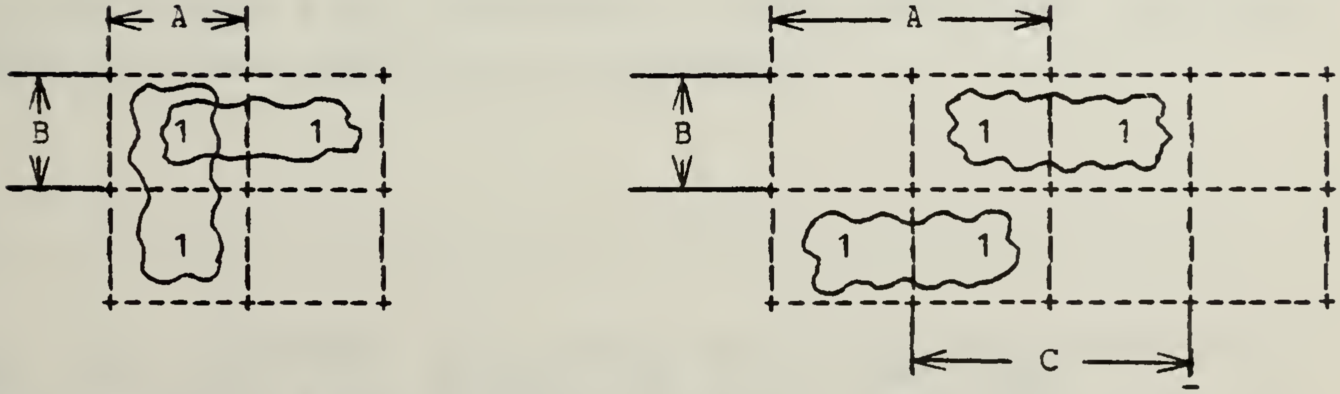


$F = \bar{A}\,\bar{D} + \bar{A}\,\bar{B}\,\bar{C}$

(A better grouping than that of the previous example.)



$F = B\,\bar{C} + \bar{A}\,B + \bar{A}\,C\,D$



$F = A + B$
(A function of 2 variables)

$F = B\,C + A\,\bar{B}$
(A function of 3 variables)

Figure 3

Karnaugh maps can be used for functions of more than four variables, but those functions will not be illustrated here. For functions of "many" variables, computer-based methods can be used.*

Now let us turn to the product-of-sums form of the switching functions. Returning to the example, with its truth table, on the first page of these notes, we can write down the product-of-sums form of F by inspection of the final column:

$$\begin{aligned} F = {} & (A+B+\bar{C}+D)(A+\bar{B}+C+\bar{D})(A+\bar{B}+\bar{C}+\bar{D})(\bar{A}+B+C+D) \\ & (\bar{A}+B+C+\bar{D})(\bar{A}+B+\bar{C}+D)(\bar{A}+B+\bar{C}+\bar{D})(\bar{A}+\bar{B}+C+D) \\ & (\bar{A}+\bar{B}+C+\bar{D})(\bar{A}+\bar{B}+\bar{C}+D)(\bar{A}+\bar{B}+\bar{C}+\bar{D}) \end{aligned}$$

Each of the sums is called a <u>maxterm</u> and corresponds to a <u>zero</u> in the Karnaugh map.

The maxterm $(A + B + \bar{C} + \bar{D})$, for example, will be zero when

$A = 0$; $B = 0$; $C = 1$; $D = 1$

The 0-cells corresponding to this maxterm will be

<u>not</u> in A; <u>not</u> in B; in C; in D

in the Karnaugh map (see Figure 4). Note the inversion!

If you compare Figure 2, where the 1-cells are obtained from the sum-of-products minterms, with Figure 4, where the 0-cells follow from the product-of-sums maxterms, you should agree that the results are equivalent.

---

* E. J. McCluskey, Jr., Minimization of Boolean Function, Bell Syst. Tech. J., vol.25, pp. 1417-1444, 1956.
W. V. Quine, A way to Simplify Truth Functions, Am. Math. Monthly, vol. 62, pp. 627-631, 1955.

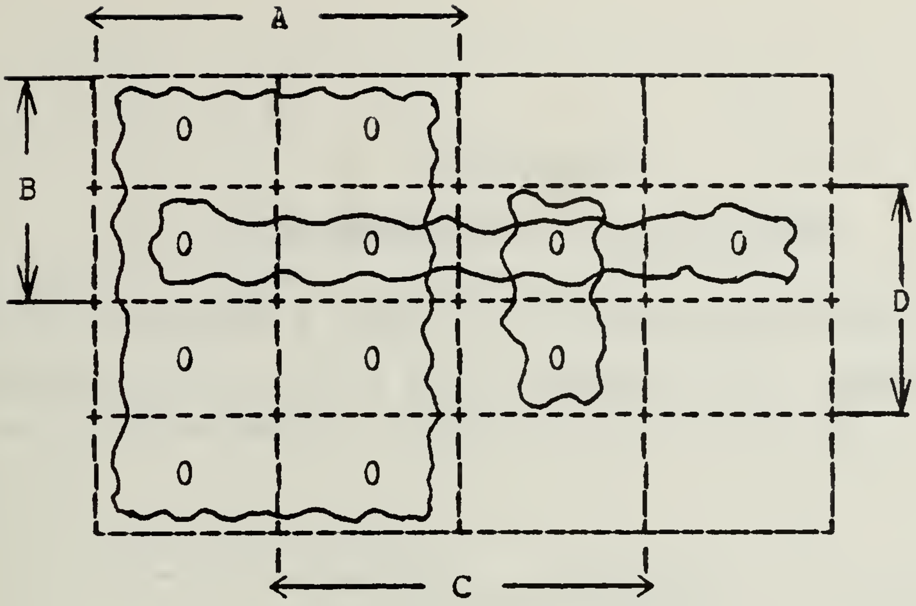


Figure 4

The 0-cells can be grouped as shown, and by inspection,

$$F = \bar{A}(\bar{B} + \bar{D})(A + \bar{C} + \bar{D})$$

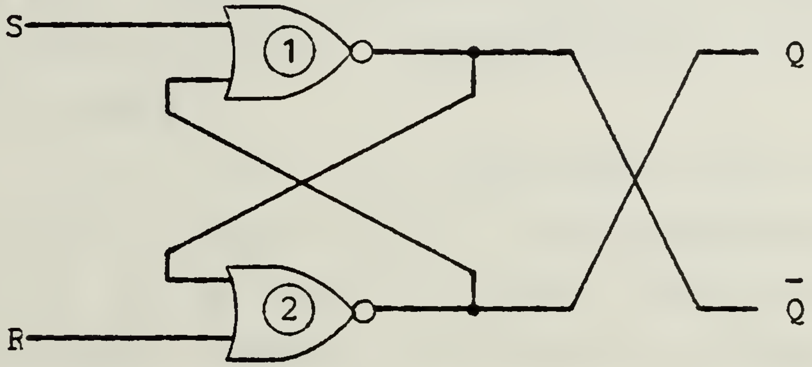
Again, a considerable simplification! Note that if a group is entirely in one region, say A, then the corresponding term in the logical sum is $\bar{A}$.

The justification of the inspection "rule" is very similar to that given for the minterm or sum-of-products method. You should be able to do this yourself.

Section 5

INTRODUCTION TO FLIP-FLOPS

Let us consider a pair of cross-coupled NOR gates as shown below. The inputs are S (*set*) and R (*reset or clear*), and the (complementary) outputs are Q and $\bar{Q}$.



If the inputs are S = 1 and R = 0, the output of NOR gate number 1 *must* be 0; that is, $\bar{Q}$ = 0. If $\bar{Q}$ = 0 (and, as given, R = 0), the output of NOR gate number 2 must be 1; that is Q = 1. In summary, a *set* input (S = 1, R = 0) will *set* Q to 1 (and $\bar{Q}$ to 0).

Exercise: Show, similarly, that a reset input (S = 0, R = 1) will clear Q to 0 (and $\bar{Q}$ to 1).

The case of S = R = 0 can be bewildering. No longer does data dictate the state of the outputs Q and $\bar{Q}$. Do you agree? Whenever this happens - and it *often* happens with circuits containing flip-flops - we overcome the difficulty by *postulating* an output state, say Q = 1 and $\bar{Q}$ = 0. Then we check the validity of the postulate. In other words, we must check to see if we are violating any of the properties of the circuit. Here with Q = 1, it follows that the output of NOR gate number 1 *must* be 0. That is, $\bar{Q}$ = 0, as postulated. Finally, since *both* inputs to NOR gate number 2 are 0, its output (Q), *must* be 1, as postulated.

In summary, if some previous event left ($Q = 1$, $\bar{Q} = 0$), then the input pair $S = R = 0$ will leave the output state unaltered. We say that the flip-flop remains *latched* in the set state as long as $S = R = 0$.

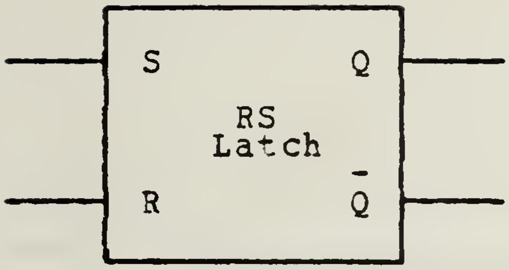
Exercise: Show, similarly, that if some previous event left ($Q = 0$, $\bar{Q} = 1$), then the flip-flop will remain *latched* in the cleared state as long as $S = R = 0$.

In total, the input pair $S = R = 0$ *latches* the *previous output state* (no matter whether this is the set or cleared state) into the flip-flop. Consequently, this circuit is often called a *latch*.

The input $S = R = 1$ is undesireable for two reasons. First, it *is* easy to see that the corresponding outputs are $Q = 0$ and $\bar{Q} = 0$. So we could no longer use the Q, $\bar{Q}$ notation for the two outputs. Second, if we were to change the inputs from $S = R = 1$ to $S = R = 0$, what would happen? Suppose S changes to zero a little quicker than R. Then we would have the input sequence S = R = 1--->(S = 0, R = 1)--->S = R = 0, which would clear the flip-flop and then latch the *cleared* state. On the other hand, if R changes to zero a little faster than S, then we would latch the *set* state into the flip-flop. In general, we won't know whether R or S will change more rapidly, and so *we won't know what will happen* when we switch from $S = R = 1$ to $S = R = 0$.

In practice, we will call the input condition $S = R = 1$ *ambiguous* and avoid it like the plague!

This has been quite a mouthful. We can put it all together as in the following diagram and table, where Q- is the previous state of the *RS latch* and Q+ is its state *after* the specified inputs have been applied.



S	R	Q+	Action
0	0	Q-	stays latched
0	1	0	clears Q
1	0	1	sets Q
1	1	*	*

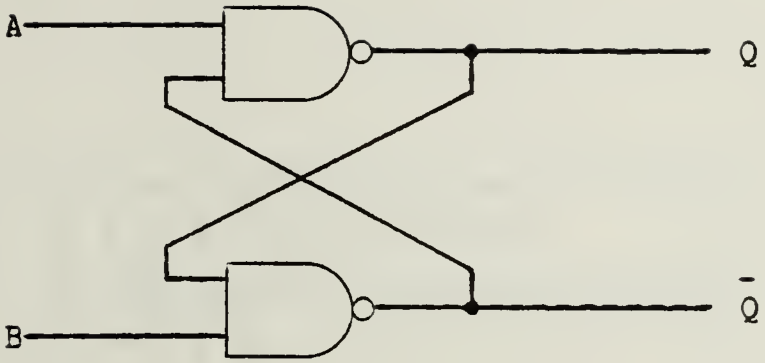
*The ambiguous case will be avoided.

Note that the action of this device depends on history - that which has gone before. For example; (S = 0, R = 1) will have no effect if Q were previously set to 0. But from the same input, (S = 0, R = 1), will change Q from a previously-set 1 to 0.

This device can be, and often is, used as one cell of a memory. We can write one bit of data into the cell by setting (S = 1, R = 0) or (S = 0, R = 1) and we can hold (i.e., memorize) the data bit, now represented by Q = 1 or Q = 0, by setting S = R = 0.

Finally, all the flip-flops which follow are built up around this basic RS latch, and all will be designed to avoid the ambiguous S = R = 1 input combination.

Exercise: See if you can complete the table of states for the cross-coupled NAND gate latch. Hint: Leave the first line in the table until the end.



A	B	Q+	Action
0	0		
0	1		
1	0		
1	1		

** Answer on next page

Answer

S	R	Q+	Action
0	0	*	*
0	1	1	sets Q
1	C	0	clears Q
1	1	Q-	stays latched

*This is opposite of the cross-coupled NOR gate latch. With this latch, the case where S = R = 0 is the ambiguous case that is to be avoided.

Section 6

LABORATORY EXPERIMENT #1

GATES

Objectives

1. To become familiar with the operating features of the DD-1 Digi-Designer.
2. To investigate the operation of AND, OR, NAND, and NOR gates.

Equipment

1. DD-1 Digi-Designer.
2. One each of the following circuits:
   7400 - Quad 2-Input Positive NAND Gate
   7402 - Quad 2-Input Positive NOR Gate
   7408 - Quad 2-Input Positive AND Gate
   7432 - Quad 2-Input Positive OR Gate
3. Card showing IC pin assignments.
4. Assortment of hook-up wire.
5. IC Extractor Clip

6.1 USE OF THE DIGI-DESIGNER

The Digi-Designer consists of a +5V logic power supply, four logic switches, two pulsers, four light emitting diode (LED) lamp monitors, a clock capable of generating square waves of six different frequencies, two pairs of terminals for external connections, a BNC (co-axial) connector, and a breadboarding assembly.

The terminal next to each logic switch provides access to the test signal available at the switch. If a switch is in

the +5V (up) position, the corresponding terminal will be at +5V potential (a logical 1); in the GRD (down) position the terminal will be at ground potential (a logical 0).

Each pulser has two terminals which have complementary outputs; when one terminal is in the logical 1 state (+5V) the other terminal is in the logical 0 state (0V), and vice-versa. When a pulser's button is depressed, the terminals reverse states; when the button is released, the terminal outputs return to their former states.

Each of the LED lamp monitors will light when a signal connected to its terminal is +5V; when the signal connected is 0V, the LED is extinguished.

The pulse generator, or clock, produces six frequencies--- 1 Hertz (Hz), 10 Hz, 100 Hz, 1 kHz, 10 kHz, and 100 kHz---which can be selected by a rotary switch.

The clock output terminals provide complementary outputs; when the left-most terminal is logical 1, the right-most terminal is logical 0, and vice-versa.

The BNC connector and the two pairs of jack terminals may be used to route signals to or from the DD-1.

The breadboarding assembly consists of two symmetrical halves separated by a groove which runs from left to right. Consider the upper half. There are sixty-four vertically running columns each having five holes. The five holes in a column are connected internally; the columns are all isolated from one another. Above the sixty-four columns are four horizontal sets of twenty-five interconnected tie points.

In this part of the experiment you are to investigate the operation of the DD-1.

6.1.1 Logic Switches and Lamp Monitors

i. Connect one logic switch output connector to one of the lamp monitor input connectors.

ii. Move the logic switch from the +5V to GRD and back again and note the illumination level of the lamp monitor.

iii. Repeat, using other lamp monitors and other logic switches.

6.1.2 Pulsers

i. Connect the "1" output connector of one of the pulsers to a lamp monitor and the "0" output of this same pulser to another lamp monitor.

ii. Note the illumination levels of the lamp monitors; depress the pulser switch and again observe the illumination levels of the lamp monitors.

iii. Repeat, for the other pulser.

6.1.3 Clock

i. Connect one of the clock's output connectors to a lamp monitor and the other clock output to another lamp monitor.

ii. Observe the LEDs at each of the clock frequencies paying particular attention to the 1 Hz case. Time this with your watch. What do you observe?

6.2 THE INTEGRATED CIRCUITS

Pick up one of the integrated circuits (ICs). It will have either 14 or 16 pins. The top surface of the IC should have two numbers marked on it. One of them, with four or five digits, starting with 74, identifies the type of IC.

(There may be one or more letters preceding this number, for example, DM7432N.) In this course you will use the TTL-family or 74-series of IC exclusively. The second number marked on the IC is a manufacturer's date code. Since many ICs were manufactured in 1974, confusion is a very real possibility!

Now search out a 7408, a Quad 2-Input AND Gate, from your IC assortment. Locate the "1-end" of the IC, which is marked by a notch, a small hole, or the like in its surface. Match the orientation of the top of the IC with the (top view) of the 7408 in the pin-assignment card (located in the manual after this tutorial). Note that the numbers run from the "1-end" down the length of the IC and then back to the "1-end) along the other side.

* Turn the DD-1 off.

* Never do any wiring unless the power is OFF. Turn the DD-1 ON <u>only</u> when you are ready to test a circuit.

* Make sure that all of the 7408's pins are straight and then place it, "<u>1-end</u>" <u>to the left</u>, so that its pins <u>rest gently</u> in holes above and below the <u>central</u> gap of the breadboard. That is each pin will make contact with a 5-hole vertical bus. Then, <u>gently</u> and firmly press the IC down with a slight rocking motion until the pins enter their holes. Continue until the body of the IC has come into contact with the breadboard.

* Wiring errors will be less likely if you habitually mount ICs "1-end" to the left.

* Take a fairly short wire and connect +5V to one of the horizontal buses. <u>Leave this bus connected</u>. It will always be convenient to construct a full-width +5V bus when working with logic circuits on the DD-1.

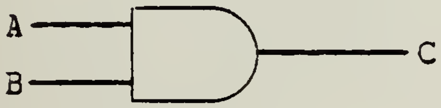
- * Construct a full-width COMMON (Ground) bus in a similar manner. Leave this bus connected.

- * Locate the Vcc and GND pins for the 7408 from the pin-assignment card. Connect these pins to the previously-wired +5V and COMMON buses, respectively. (Use the 5-pin buses into which the appropriate IC pins have been plugged).

- * Develop the habit of connecting +5V and COMMON (GND) to all ICs before you do any other wiring. These connections are usually omitted from wiring diagrams, so that only unvarying habit can keep you from error.

6.3 THE AND GATE

The operation of a (positive-logic) AND gate is defined by the truth table, below. That is, both inputs must be HIGH (1) to yield a HIGH (1) output.

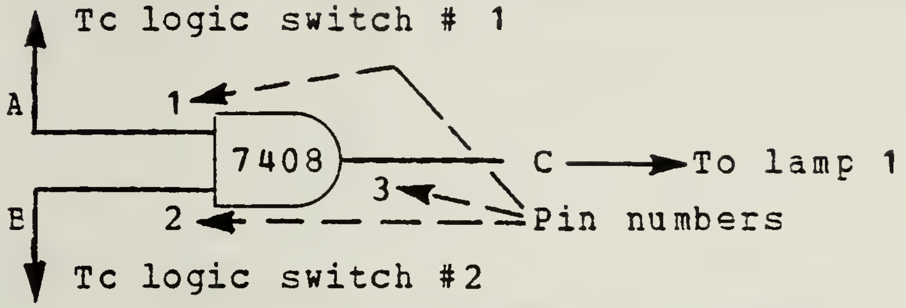


$C = A \bullet B$

AND Gate Truth Table

Inputs		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

A. Wire one of the AND Gates as shown below. Also, connect pin 7 to COMMON and pin 14 to +5V.



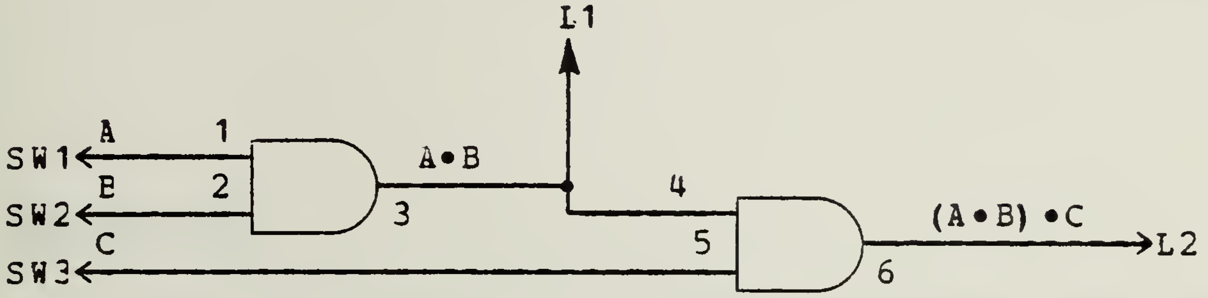
B. Turn on the power, and for each pair of switch settings (columns A and B in the table below) <u>record</u> the output that you observe in column C of the table.
(LED on = 1; LED off = 0.) Turn the DD-1 off when you have finished.

2-Input AND Gate Truth Table

Inputs		Output
A	B	C
0 (LO)	0 (LO)	
0	1 (HI)	
1	0	
1	1	

6.3.1 A 3-Input AND Gate

A. Make sure that the power is off and then construct a 3-input AND gate from two, 2-input AND gates as shown below.



B. Turn on the DD-1, set the switches as shown in the table below, and record the outputs that you observe.

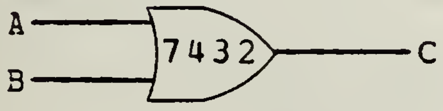
3-Input AND Gate Truth Table

Inputs A B C	L1 = A•B	L2 = A•B•C
0 0 0		
0 0 1		
0 1 0		
0 1 1		
1 0 0		
1 0 1		
1 1 0		
1 1 1		

C. Turn off the power and remove the wires from the breadboard. (You can always leave the full-width +5V and COMMON buses connected, since you will use them in every experiment.) Then carefully remove the IC using the IC extractor clip.

6.4 THE OR GATE

If at least one input to an OR gate is HIGH (1) the output will be HIGH (1). The operation of a 2-input OR gate is illustrated below.

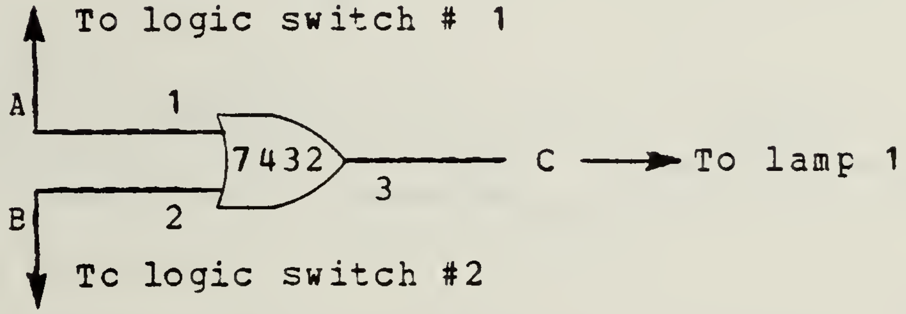


$C = A + B$

OR Gate Truth Table

Inputs A	Inputs B	Output C
0	0	0
0	1	1
1	0	1
1	1	1

A. Insert a 7432 quad, 2-input OR gate into the breadboard. Connect +5V and COMMON. Wire one of the OR gates as shown below.



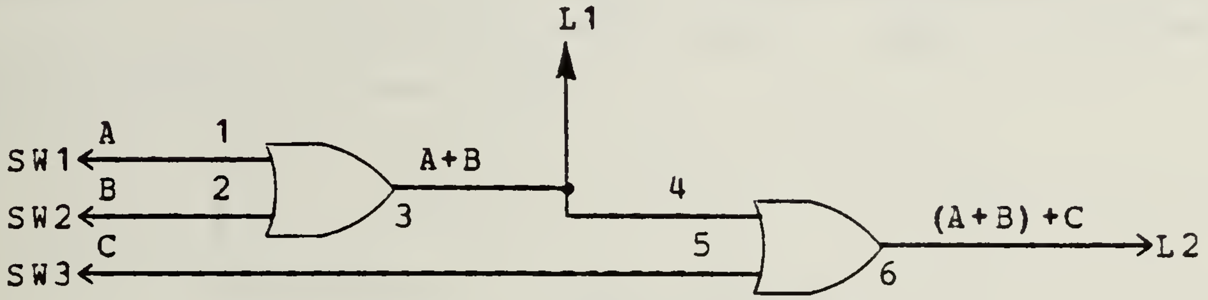
B. Turn on the power, and for each pair of switch settings (columns A and B in the table below) <u>record</u> the output that you observe in column C of the table.

2-Input OR Gate Truth Table

Inputs A	Inputs B	Output C
0 (LO)	0 (LO)	
0	1 (HI)	
1	0	
1	1	

6.4.1 A 3-Input OR Gate

A. Make sure that the power is off and then construct a 3-input OR gate from two, 2-input OR gates as shown below.



B. Turn on the DD-1, set the switches as shown in the table below, and record the outputs that you observe.

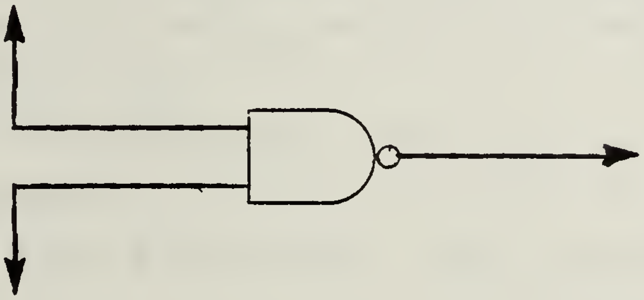
3-Input OR Gate Truth Table

Inputs A	 B	 C	L1 = A+B	L2 = A+B+C
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

C. Turn off the power and remove the wires and the IC from the breadboard.

6.5 THE NAND GATE

A. Complete the following circuit diagram showing how you could test one of the 7400's NAND gates. Show pin numbers and switch and LED connections.



B. Plug in a 7400 and wire it according to your circuit diagram. Then test it, completing the following table as you do so.

2-Input NAND Gate Truth Table

Inputs		Output
A	B	C
0 (LO)	0 (LO)	
0	1 (HI)	
1	0	
1	1	

6.6 THE NOR GATE

Repeat the preceding procedure, using a 7402 NOR-gate. Draw the circuit diagram (showing pin numbers, etc.) and draw up and complete a table for your test results.

6.7 INVERTERS

a. Use the DD-1 to obtain the truth tables for

1. A 7400 NAND gate with its two inputs connected together, and
2. A 7402 NOR gate with its two inputs connected together.

(Draw your circuits diagrams, first, showing pin numbers.)

Circuit for 7400
NAND-Gate Inverter

Circuit for 7402
NOR-Gate Inverter

NAND-Gate Inverter

Inputs A = B	Output C
0	
1	

NOR-Gate Inverter

Inputs A = B	Output C
0	
1	

6.8 AN OPTIONAL DESIGN PROBLEM

A Consider the following logic function

$$X = \bar{A}\bar{B}C + A\bar{B}C + \bar{A}BC + A\bar{B}\bar{C}$$

a. Simplify the function

b. Construct a truth table.

c. Design a logic circuit (using the ICs investigated in the lab) to realize this function.

B. Wire up your circuit and verify that it actually satisfies the function.

This logic function <u>can</u> be simplified to the point where one requires only two separate ICs and five gates, including those used as inverters. (It is <u>not</u> essential that you reach this degree of "perfection".)
Attach a separate page showing your work and results. Include the circuit diagram with pin connections.

Section 7

LABORATORY EXPERIMENT #2

THE XOR GATE, FULL-ADDERS AND HALF-ADDERS

Objectives

1. To design and test several realizations of an XOR gate.
2. To design and test half-adders and full-adders.

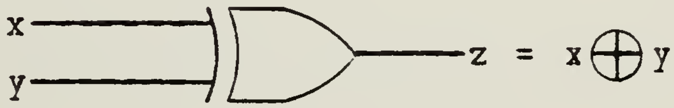
Equipment

1. DD-1 Digi-Designer.
2. The following integrated circuits:
   2 - 7400 Quad 2-Input Positive NAND Gates
   2 - 7402 Quad 2-Input Positive NOR Gates
   1 - 7404 Hex Inverter
   1 - 7408 Quad 2-Input Positive AND Gate
   1 - 7432 Quad 2-Input Positive OR Gate
   1 - 7486 Quad 2-Input Positive XOR Gate
   1 - 7482 2-Bit Binary Full-Adder
3. IC Extractor Clip.
4. Assorted hook-up wire.
5. IC pin-assignment card

7.1 THE EXCLUSIVE-OR GATE

An EXCLUSIVE OR (XOR) gate functions so that its output is 1 whenever an odd number of inputs is 1, otherwise its output is 0. The truth table for a two-input XOR gate is shown below.

Inputs X	Inputs Y	Output Z
0	0	0
0	1	1
1	0	1
1	1	0



A. Mark pin numbers on the above diagram corresponding to the 7486 XOR gate. Connect Vcc and GND to +5V and COMMON; connect the inputs to logic switches; and the output to a LED monitor. Test the 7486's logic: does it agree with the above truth table?

B. Design a sum-of-products realization of the 2-input XOR gate using AND, OR, and NOT (inverter) logic. Draw the circuit below or on separate paper. (It will always be helpful to include pin numbers and, of course, the numbers of the ICs.) Then test the circuit using two logic switches and a LED. Put your data in a truth table. Does the latter agree with that given for an XOR gate above?

C. Repeat the preceding exercise, but use <u>only</u> NAND gates (IC 7400).

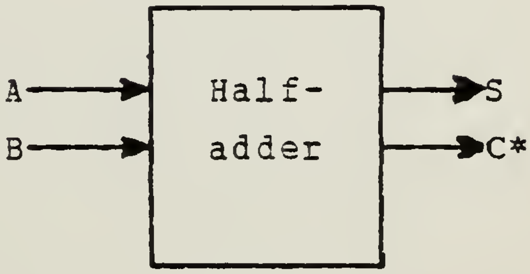
D. Design a product-of-sums realization of the 2-input XOR gate using AND/OR/INVERTER logic. Draw the circuit and test it to obtain its truth table. Does the latter agree with the given truth table for an XOR gate?

E. Repeat the preceding exercise, but use <u>only</u> NOR gates (IC 7402).

7.2 THE HALF-ADDER

A half-adder has only two inputs, A and B, and thus does not include any carry which may have resulted from adding together less significant bits. The outputs of the half-adder are the sum S, and carry, C*, for which the truth table is shown below.

Inputs		Outputs	
A	B	S	C*
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	1



A. Design and test a half-adder using AND/OR/INVERTER logic. (Draw the circuit, and complete the truth table, experimentally.)

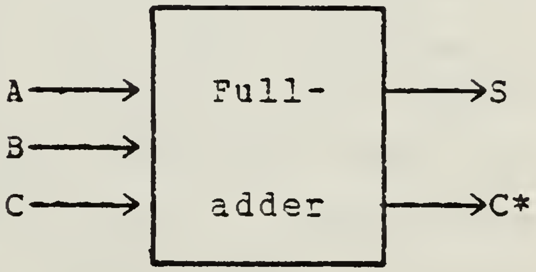
B. Repeat the preceding exercise, but use an AND gate (7408) and an XOR gate (7486).

* DO <u>NOT</u> DISASSEMBLE THIS CIRCUIT. YOU WILL USE IT AGAIN.

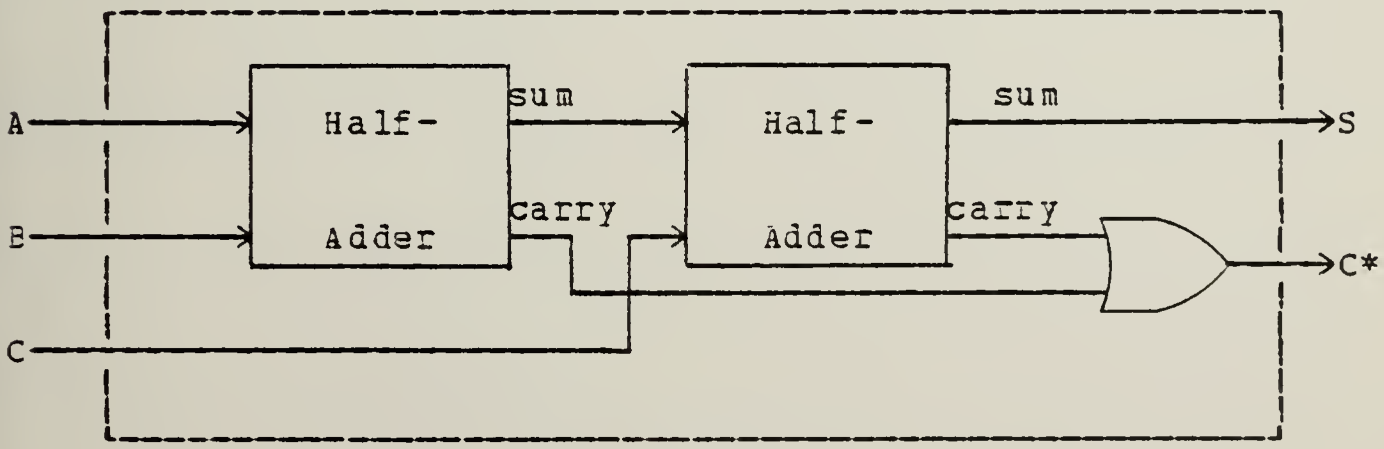
7.3 THE FULL-ADDER

The truth table for one bit of a binary full-adder is shown below. The bits to be added are denoted by A and B and the carry from the previous bits by C. The two outputs are the sum bit, S, and carry, C*, for the next stage.

Inputs			Outputs	
C	A	B	S	C*
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



A full-adder can be constructed from two half-adders as shown below.



Full-adder

A. Construct another half-adder as in the previous exercise, but using an AND and an XOR gate. Then, combine the two half-adders and an OR gate (7432) to complete a full-adder. (A complete circuit diagram, with pin numbers, will help you avoid wiring errors.) Test the full-adder, putting your data in a truth table. Does the latter agree with the given table above?

7.4 OPTIONAL FULL-ADDER EXERCISE

From the full-adder's truth table, set up the sum-of-products and product-of-sums forms of the logic functions S and C*. See if you can simplify them, being on the lookout for terms common to S and C*, which will therefore need only be generated once. (Don't spend a lot of time! Very little simplification is, in fact, possible.) If you wish, draw an AND/OR/INVERTER logic or a NAND/NOR logic realization. Your circuit diagrams will be quite involved, and constructing and testing such a circuit would not teach you much. (If you do try, you may have to ask for additional ICs)

If you wish, take a 7482 IC (a 2-bit binary full-adder), try to untangle its logic diagram (on its data sheet), and test it.

Section 8

LABORATORY EXPERIMENT #3

RS LATCH, D-TYPE FLIP-FLOPS, AND SHIFT REGISTERS

Objectives

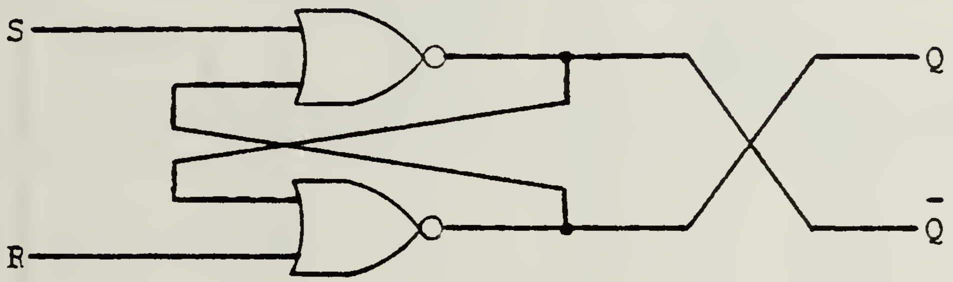
1. To investigate the operating characteristics of the RS latch and D-type flip-flop.
2. To design and test some registers using D-type flip-flops.

Equipment

1. DD-1 Digi-Designer.
2. The following integrated circuits:
   1 - 7402 Quad 2-Input Positive NOR Gates
   1 - 7404 Hex Inverter
   1 - 7408 Quad 2-Input Positive AND Gate
   2 - 7474 Dual D-type, Edge-triggered Flip-flops
3. IC Extractor Clip.
4. Assorted hook-up wire.
5. IC pin-assignment card

8.1 THE RS LATCH

The RS latch may be thought of as the basic building block from which various types of flip-flops can be constructed. (RS stands for reset-set. Sometimes SC is used, meaning set-clear. Occasionally the term flip-flop is used instead of latch.) The basic configuration is shown in the following figure.



A. Mark pin numbers on the above diagram, assuming the use of 7402 NOR gates. Plug in a 7402 IC about midway across the breadboard. Then wire up an RS latch with S and R coming from logic switches, and with LEDs monitoring the outputs Q and $\bar{Q}$. Now "play" with the circuit. Observe that as you operate the switches you can cause the LEDs to "flip-flop". You should never see both LEDs lighted at the same time, and only when both switches are set HI should both LEDs be unlit.

When Q is 1 we say that the latch is <u>set</u>. When Q is 0, we say that the latch is <u>reset</u> or <u>cleared</u>. Make sure you know how to set or reset the latch. The condition S = R = 1 is said to be <u>ambiguous</u>, since Q = $\bar{Q}$ = 0 and the latch is then neither set nor reset. Furthermore, when you switch from S = R = 1 to S = R = 0, you cannot foretell which way the latch will "flip". Try it a few times. We will go to some trouble to avoid this condition in practical circuits.

Finally, note that whenever you switch to S = R = 0, the state of the latch remains unchanged (except in the ambiguous case).

B. Run through this again, systematically recording your observations in the following table. Here Q- and Q+ are the values of Q before (-) and after (+) the inputs are applied.

Inputs		Previous State	New State	
S	R	Q-	Q+	$\overline{Q}$
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

<u>Hint</u>: How should you obtain a desired previous state (Q-)? Suppose for example, that you are ready to complete the fourth line in the table. You can

(i) set S = 1 and R = 0 to obtain Q- = 1,
(ii) set S = R = 0, a neutral starting point, and then
(iii) set the inputs S = 0 and R = 1.

Now you can record the new outputs, Q+ and $\overline{Q}$+.

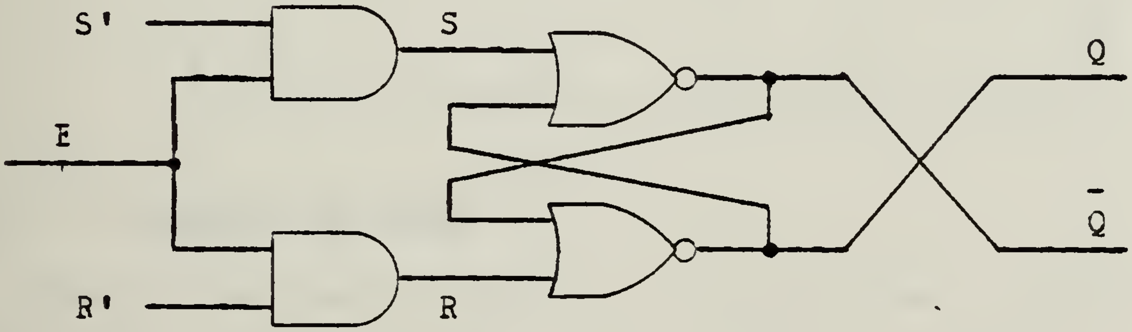
Does your completed table agree with the following short table?

Inputs S	R	Q+	Action
0	0	Q-	Stays latched
0	1	0	Resets (clears) Q
1	0	1	Sets Q
1	1	Q+=Q+=0	Ambiguous

* DO <u>NOT</u> DISCONNECT THE LATCH CIRCUIT.

8.2 THE RS LATCH WITH ENABLE

A. Add an enable provision to your RS latch by inserting AND gates as shown below. Use logic switches for the S', E, and R' inputs and connect LEDs to the outputs Q and $\bar{Q}$. Test the circuit and complete the truth table. (To obtain each desired previous state, Q-, you may have to set or reset the latch with the logic switches.)



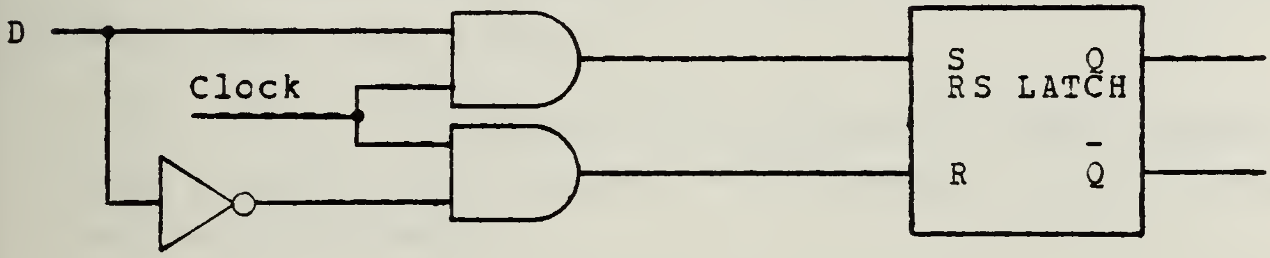
	Inputs		Previous State	New State
E	S'	R'	Q-	Q+
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

8.2.1 <u>Clocked RS Latch</u>

Connect the enable input of the previous circuit to the 1-Hz clock instead of the logic switch. Connect a LED to monitor the clock. Then observe the effects of changing the S' and R' inputs at various times during the clock cycle. Confirm that the latch performs as in the truth table above, except that the clock now takes the place of the input E.

8.3 THE D-TYPE FLIP-FLOP

Add an inverter (IC 7404, or one of the NOR gates as an inverter) to your clocked RS latch, as shown. The result is a level-clocked, D-type flip-flop. Test the circuit and complete the truth table (in which X = "don't care"). Unless your reactions are very fast, you will find it convenient to use a logic switch instead of the 1-Hz clock to enable the flip-flop.



Inputs		Previous State	New State
CLOCK	D	Q-	Q+
0	X	0	
0	X	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

At this point you should turn off the DD-1 and remove all of the wires and ICs.

8.3.1 The 7474 D-type Flip-flop

Set up a 7474 edge-clocked D flip-flop with a 1-Hz clock input (CP on the pin-assignment card). Use logic switches to set the data input (D), $\overline{\text{preset}}$ ($\overline{S}$), and $\overline{\text{reset}}$ ($\overline{R}$). Connect LEDs to monitor Q, $\overline{Q}$, and the clock. (Note the fact that the set and reset inputs are inverted--bubbled.) Investigate the flip-flop's operation. In particular, try to answer the following questions experimentally.

(i) At what time during a clock cycle can (or does) a data input take effect?

(ii) Is the 7474 positive or negative edge-clocked? In this respect, how does the 7474 compare with the standard D-type flip-flop?

(iii) When can (or does) a preset or reset input take effect?

(iv) Does the state of the data input and/or the clock influence the effect of the preset or reset inputs?

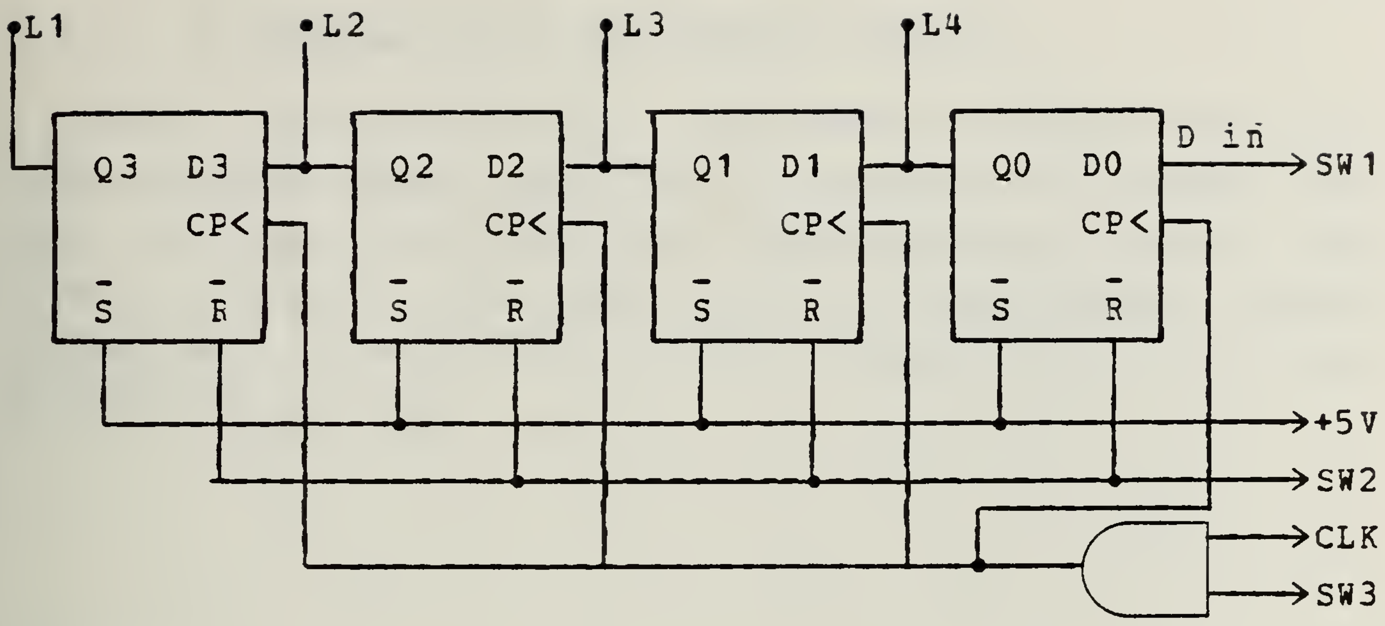
(v) To preset a 7474 flip-flop, must the input, $\overline{S}$, be zero or one? To clear a 7474 flip-flop, must $\overline{R}$ be zero or one?

8.4 SOME APPLICATIONS OF D-TYPE FLIP-FLOPS

D-type flip-flops are very common in digital systems. Three typical circuits follow.

8.4.1 Serial-load, Left-shift Register

A. Use two dual 7474 flip-flops to construct the following serial-load, left-shift register. As always, numbering the pins in the diagram will help you avoid wiring errors.



B. Test the register, observing that you can

(i) retain the register contents indefinitely by disabling the clock with SW3,

(ii) shift data left on each clock pulse, entering the new LSB from SW1, and loosing the MSB off the left end of the register, and

(iii) clear the register at any time with SW2.

8.4.2 A Ring Counter

Make two changes in the previous circuit so that

(i) the clear operation (SW2) now enters the number 0001 into the register, and

(ii) successive clock pulses then left-shift the single 1-bit circularly around the register.

Make the changes in the diagram and test the ring counter. Did it perform as required?

8.4.3 A Parallel-load, Left-shift Register

Redraw the serial-load, left-shift register with the modifications needed to allow a synchronous parallel load (all four input bits loaded on one clock edge when the LOAD input is high). When LOAD is low, the register should operate as in the circuit above. Construct and test the circuit if you like.

Section 9

LABORATORY EXPERIMENT #4

THE JK FLIP-FLOP AND ASYNCHRONOUS COUNTERS

Objectives

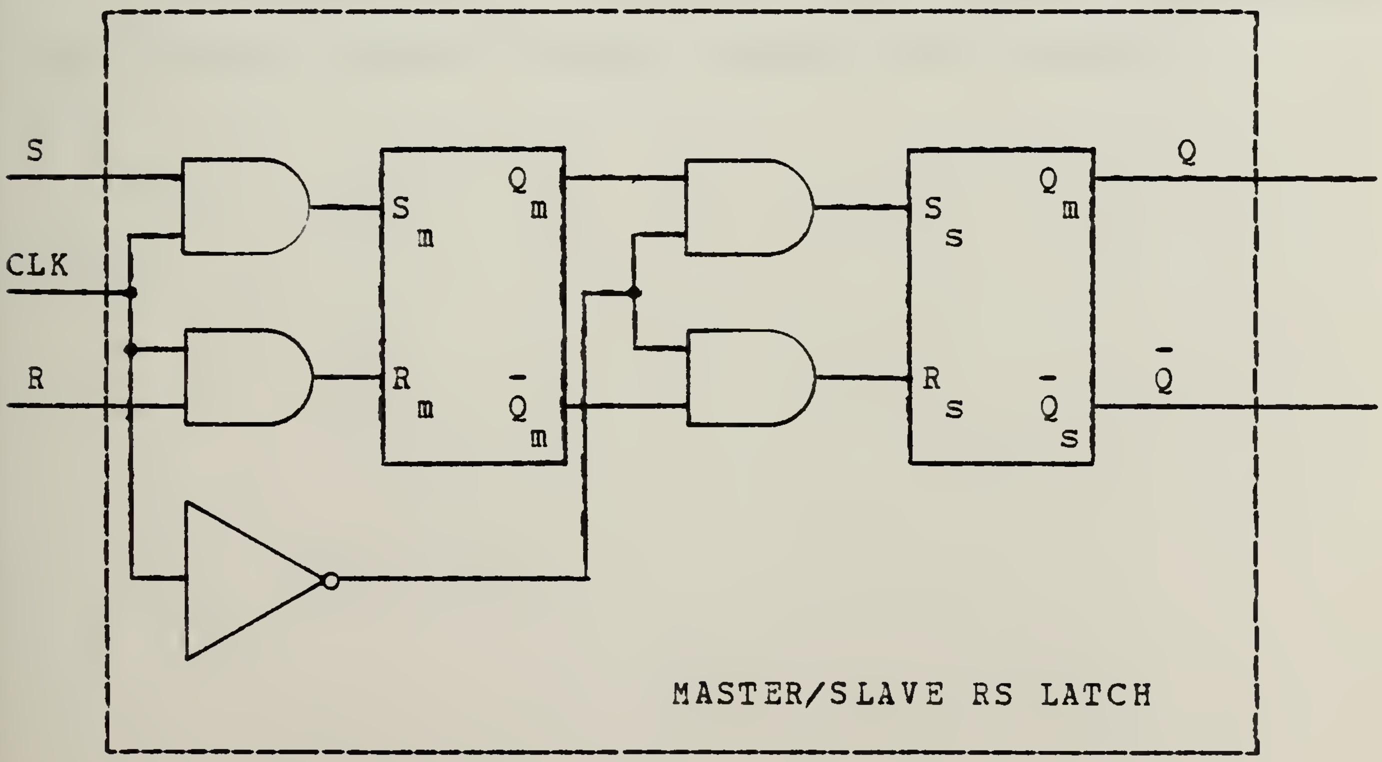
1. To investigate the operating characteristics of the JK flip-flop.
2. To design and test some ripple (i.e. asynchronous) counters.

Equipment

1. DD-1 Digi-Designer.
2. The following integrated circuits:
   1 - 7402 Quad 2-Input Positive NOR Gates
   1 - 7404 Hex Inverter
   2 - 7408 Quad 2-Input Positive AND Gate
   1 - 7432 Quad 2-Input Positive OR Gates
   2 - 7473 Dual JK Master/Slave Flip-Flops with Separate Clears and Clocks
   2 - 7474 Dual D-type, Edge-triggered Flip-flops
3. IC Extractor Clip.
4. Assorted hook-up wire.
5. IC pin-assignment card

9.1 THE MASTER/SLAVE CONFIGURATION

If one or more flip-flops in a logic circuit are driven by the outputs of flip-flops (directly, or through gates) there are potential timing difficulties with the basic clocked RS latch constructed in experiment 3 (RS LATCH, D FLIP-FLOP, AND SHIFT REGISTERS). Specifically, the clock pulse must be narrow enough so that no flip-flop responds to the "new" output of a flip-flop clocked at the same time. On the other hand, the clock pulse must be long enough to ensure that every flip-flop has time to respond reliably to its legitimate inputs. One way to avoid this difficulty is to use the master/slave configuration shown below. The master latch responds to its inputs only when the clock is high. When the clock goes low, the master is disabled first and immediately thereafter the slave is enabled and responds to the master's outputs. The new output (coming from the slave) cannot affect a master until the clock next goes high. Thus a short clock pulse is not necessary, and a square-wave clock signal can be used.



9.1.1 The Master/Slave RS Latch

A. Use two 7402 NOR gates to build each of two RS latches as in experiment 3. Then complete the circuit as shown above using 7408 AND gates and a 7404 inverter. (Be careful! Make sure that Q_m not $\bar{Q}_m$, leads to the slave's set input, and be certain to identify $Q \equiv Q_s$ correctly.) It will help if you draw the complete circuit, with pin numbers marked. Connect logic switches to S and R, the 1-Hz clock to CLK, and use LEDs to monitor Q_m, Q_s ($\equiv Q$), and CLK.

B. "Play" with this circuit until you are sure that it is functioning correctly, and that you understand the master/slave idea. (You will probably find it convenient to replace the 1-Hz clock with a logic switch.) In particular, do you agree that

(i) A set or reset input can take effect (on the master) only when CLK is HI?

(ii) The set or reset input then takes effect at the slave's output (Q and $\bar{Q}$) at the instant that the CLK next goes LO?

C. Test the circuit, completing the full and abbreviated logic tables, below. DO NOT DISMANTLE YOUR CIRCUIT.

Inputs S	R	Q-	Q+	$\bar{Q}$+
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

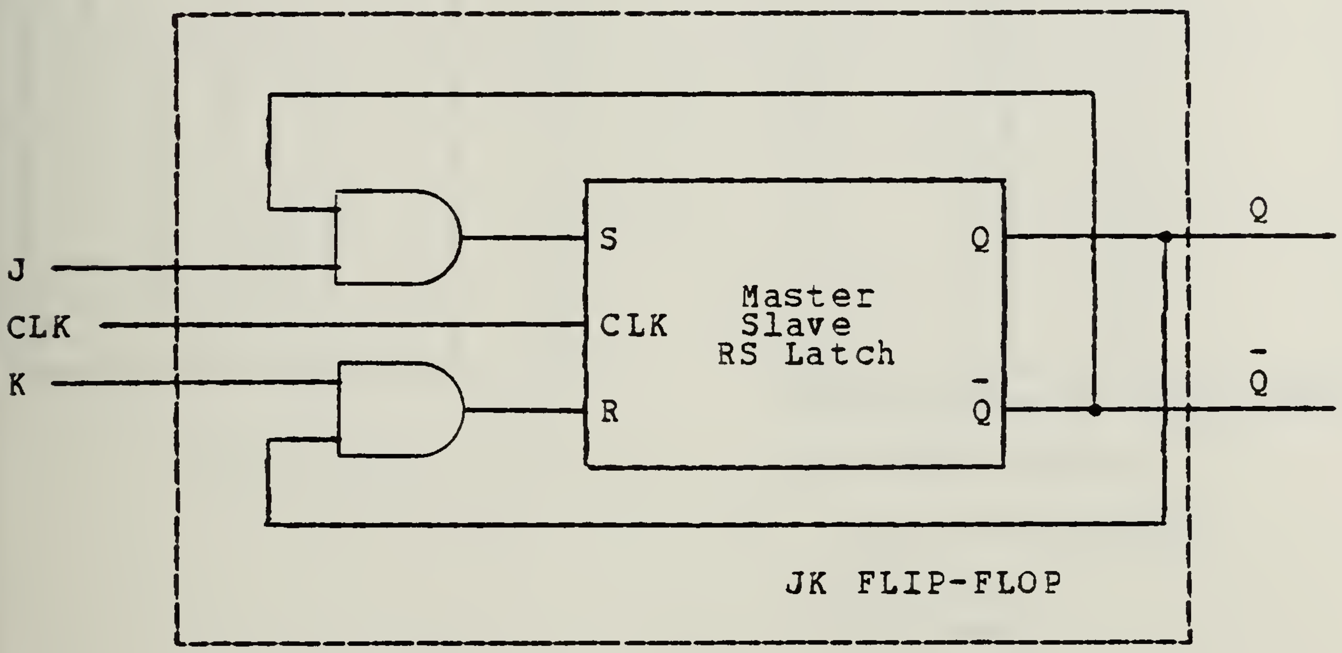
Inputs S	R	Q+	Action*
0	0		
0	1		
1	0		
1	1		

*The action of the circuit (if any) can be:

- To stay latched
- To set
- To reset
- Ambiguous

9.2 THE JK FLIP-FLOP

A. Use two additional 7408 AND gates to convert your master slave RS latch into a JK master/slave flip-flop. (Note that Q feeds back to the reset side of the master; $\bar{Q}$ feeds back to the set side.)



B. Connect logic switches to J and K, the 1-Hz clock or a logic switch to CLK, and monitor Q, $\bar{Q}$, and CLK with LEDs. "Play" with the circuit until you are satisfied that it is behaving as you expect. Complete the logic tables, below.

To avoid confusion, J and K should be changed only when CLK is HI. Then the outputs Q and $\bar{Q}$ will reflect such a change at the instant that the CLK next goes LO - a negative edge.

Inputs S	R	Q-	Q+	$\overline{Q}$+
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Inputs S	R	Q+	Action*
0	0		
0	1		
1	0		
1	1		

*The action of the circuit (if any) can be:

To stay latched
To set
To reset
Ambiguous

NOW YOU SHOULD DISASSEMBLE ALL YOUR CIRCUITS FROM THE DD-1.

9.2.1 The Dual JK FLIP-flop

Set up one of the 7473 JK flip-flops with logic switches to set J, K, and $\overline{R}$.[1] Use the 1-Hz clock or a logic switch for $\overline{CP}$ (CP = clock pulse). Monitor $\overline{CP}$, Q, and $\overline{Q}$ with LEDs. Test the flip-flop to see if its behavior agrees with that that of your home-grown unit of the previous section.

> Does the output (slave) transition occur on a positive clock edge or a negative edge?
>
> Does a LO or a HI input to $\overline{R}$ clear the flip-flop?
>
> Does a reset signal clear the flip-flop regardless of the state of J, K, and $\overline{CP}$?

9.3 ASYNCHRONOUS COUNTERS

There are two basic types of counter - synchronous and asynchronous. The latter is also known as a ripple or serial counter. In it, one flip-flop changes state, triggering a second flip-flop, which triggers a third, and so on, ...The effect ripples through the array of flip-flops.

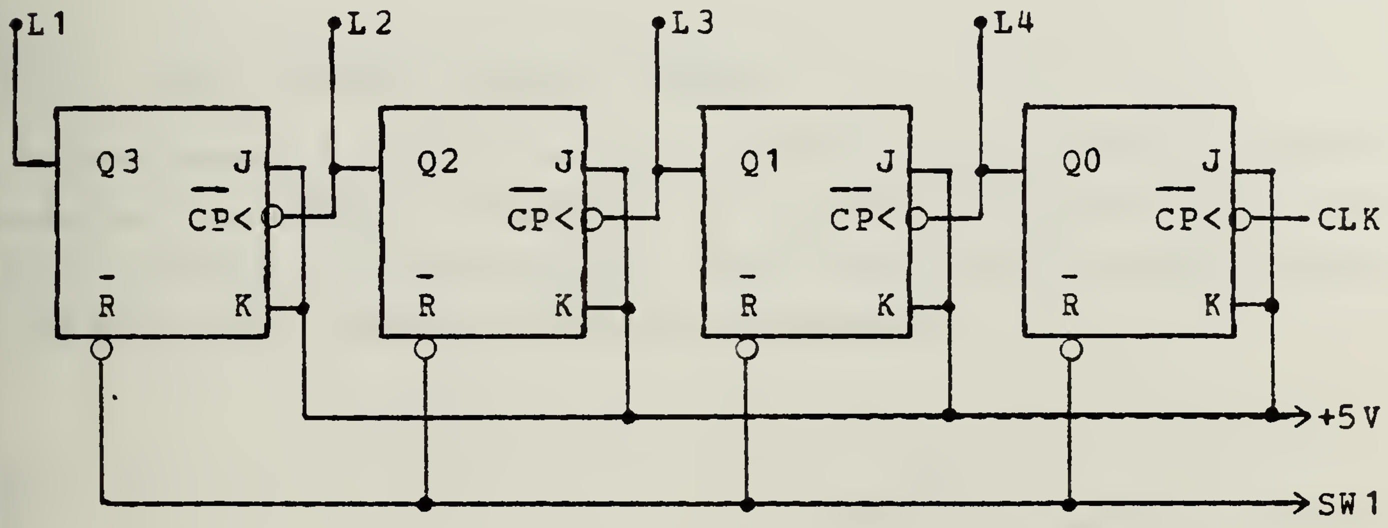
---

[1] Note the non-standard Vcc and GND connections with a 7473 JK flip-flop.

9.3.1 The Binary Ripple Up-Counter

A. Insert two 7473 dual JK flip-flops on the DD-1 and construct the counter shown below.

* Note that the J and K inputs must be connected to +5V (logical 1) even though some texts suggest that they may be left floating (i.e., unconnected). If you do leave them unconnected, erratic or even non-operation may result.



B. Make sure that the counts runs correctly from 0000 (after clearing) to 1111 (binary) before automatically reseting to 0000 for the next cycle.

Compare the frequency at the output of each stage with the input clock frequency. This circuit is often called a frequency divider.

f_{CLK} = 1-Hz
f_0 =
f_1 =
f_2 =
f_3 =

DO NOT DISMANTLE YOUR COUNTER.

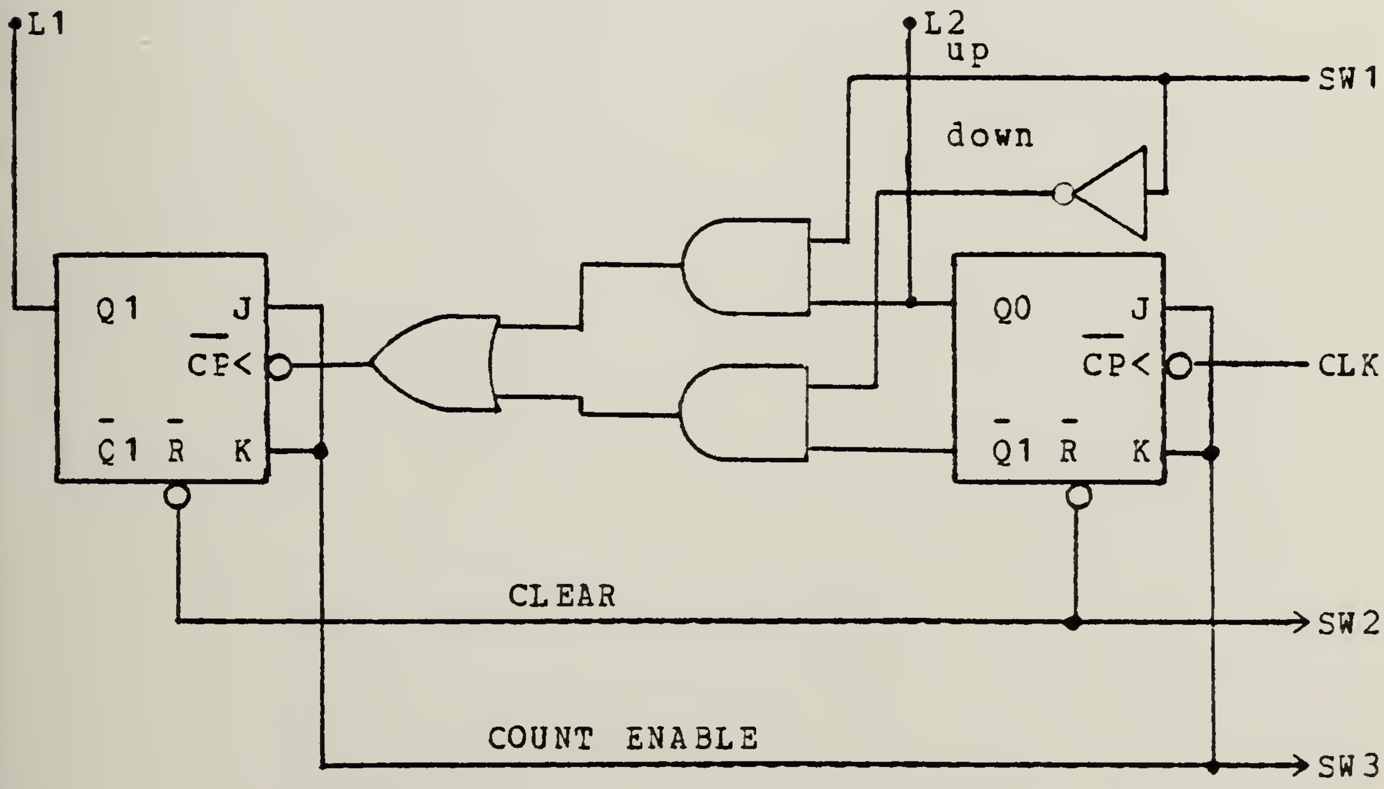
9.3.2 The Binary Ripple-Down Counter

Convert your circuit into a down-counter. You need not draw the circuit, but you should describe the change(s) that you made. (Changes to LED connections are not allowed.)

Did the counter operate correctly? What was the count sequence following a clear signal?

9.3.3 The Ripple Up/Down Counter

The counting actions of the previous two circuits can be combined by adding a suitable arrangement of gates. One such configuration is shown below. (You could use four NAND gates instead of the AND/OR/INVERTER arrangement)



Construct and test this modulo-4, binary, up/down ripple counter. (A modulo-N counter is defined to be a counter with N states. Here N = 4 and the four states are 00, 01, 10, and 11.) Did this counter perform as you expected?

There will be more on counters in the next experiment.

Section 10

LABORATORY EXPERIMENT #5

MORE COUNTERS

Objectives

1. To investigate the characteristics of an asynchronous (i.e. ripple) decade counter.
2. To investigate the properties of several synchronous counters.

Equipment

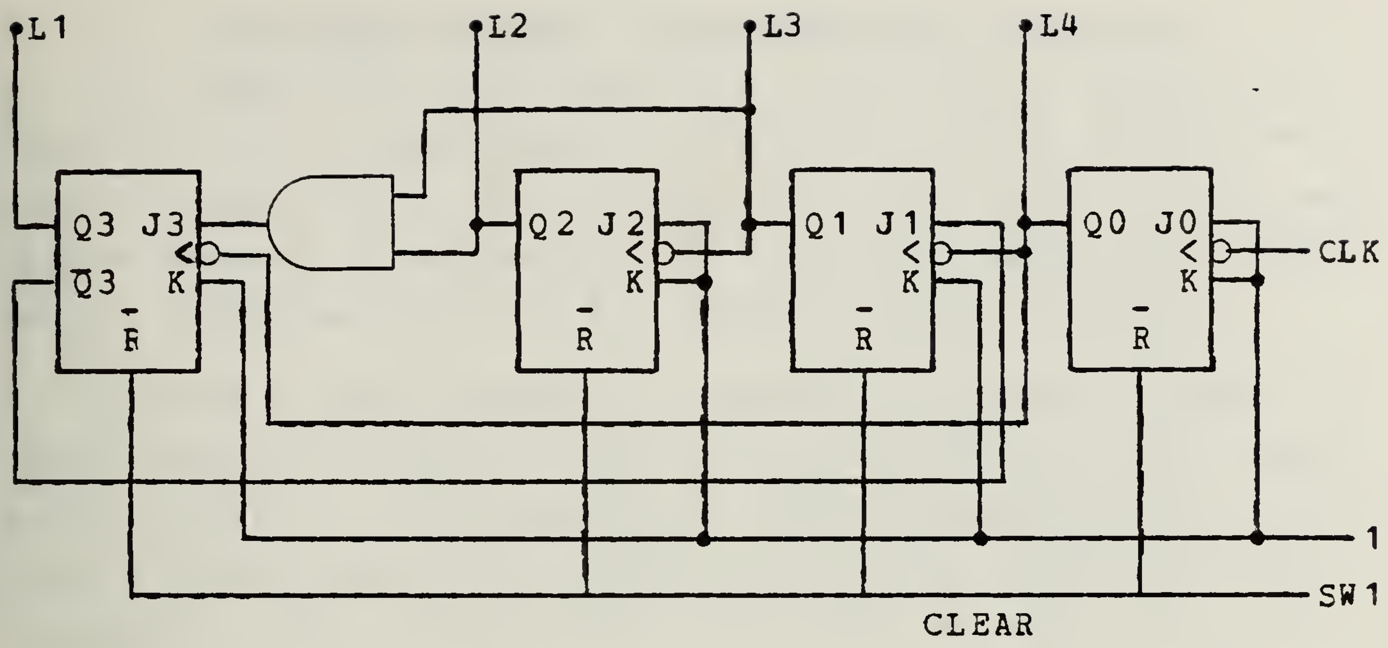
1. DD-1 Digi-Designer.
2. The following integrated circuits:
   - 1 - 7408 Quad 2-Input Positive AND Gate
   - 1 - 7411 Triple 3-Input Positive AND Gates
   - 2 - 7473 Dual JK Master/Slave Flip-flops with separate clears and clocks
3. IC Extractor Clip.
4. Assorted hook-up wire.
5. IC pin-assignment card

10.1 ASYNCHRONOUS COUNTERS (CONCLUDED)

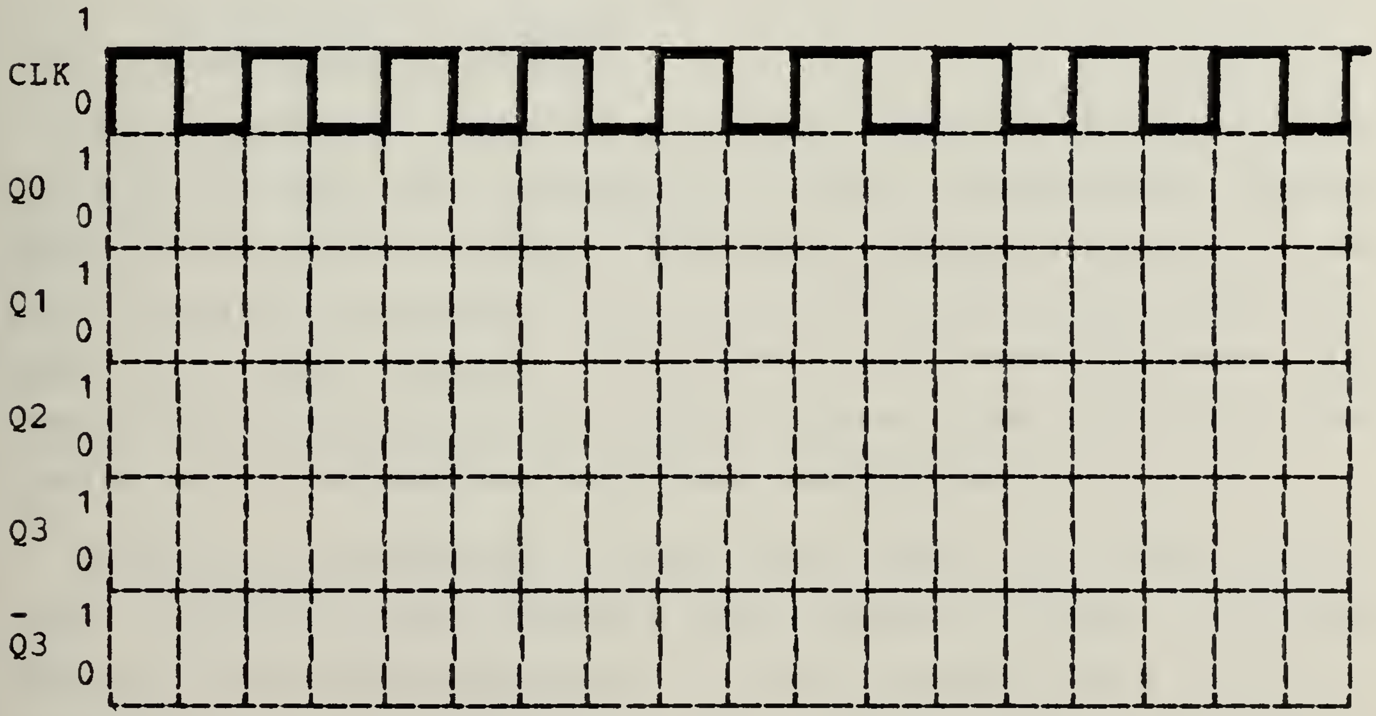
In the previous laboratory (THE JK FLIP-FLOP AND ASYNCHRONOUS COUNTERS) you investigated a number of ripple counters. Here is one more.

10.1.1 The Ripple BCD Decade Counter

BCD (binary coded decimal) implies that the digits in this counter are assigned the usual binary weights of 8-4-2-1. Decade and decimal imply modulo-10. So the counter must be designed to count up normally from 0000 to 1001, and must then automatically reset to 0000 on the next (the tenth) clock pulse. One such circuit follows.



A. Show that the above circuit operates as a BCD decade counter by completing the timing diagram, below. Make sure that your diagram shows a proper count sequence: 0000, 0001, ..., 1001, 0000, 0001, ...



B. Set up the circuit, using two 7473 dual JK flip-flops and a 7408 AND gate. Does your counter behave as your timing diagram predicts?

10.1.2 The Decade Counter (continued and optional)

A. If more than one decade of counting is required, a "carry" must be generated by each of the lower decades as it resets. This carry will act as the clock input to the next higher decade. Show how you would obtain this carry signal (which should go low when the decade resets to 0000).

B. Design a logic system to "decode" the four outputs of a decade counter. That is, the four signals Q0, Q1, Q2, and Q3 must generate a high output in the appropriate one of ten output lines (one corresponding to each decimal digit 0 through 9). All other output lines must be low.

C. Design a circuit which will generate an output voltage proportional to the count in a decade counter. This voltage must look like a staircase as the count rises from 0000 to 1001, and the voltage must return to zero on the next count (great accuracy is not required). This is basically a digital-to-analog converter.

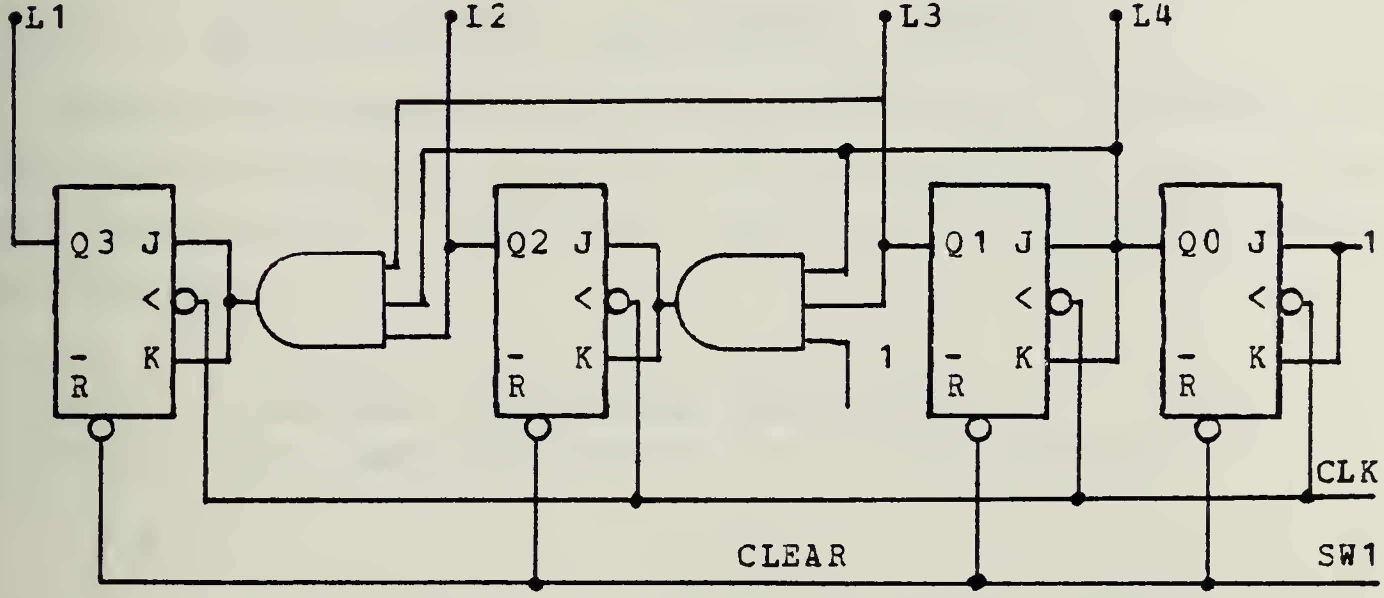
10.2 SYNCHRONOUS COUNTERS

The principal limitation of an asynchronous (ripple) counter is that the frequency at which the counter can be driven is limited by the number of flip-flops and their delay times. The reason for this is that the clock pulse for each flip-flop (except the first) is received from the preceding flip-flop in the chain. Thus, one flip-flop must change state before the next can, and so on.

Synchronous counters, on the other hand, are designed so that all flip-flops receive common clock pulses, and hence change state simultaneously. Gate networks are added to selectively control the inputs to the flip-flops and thereby provide the counting action.

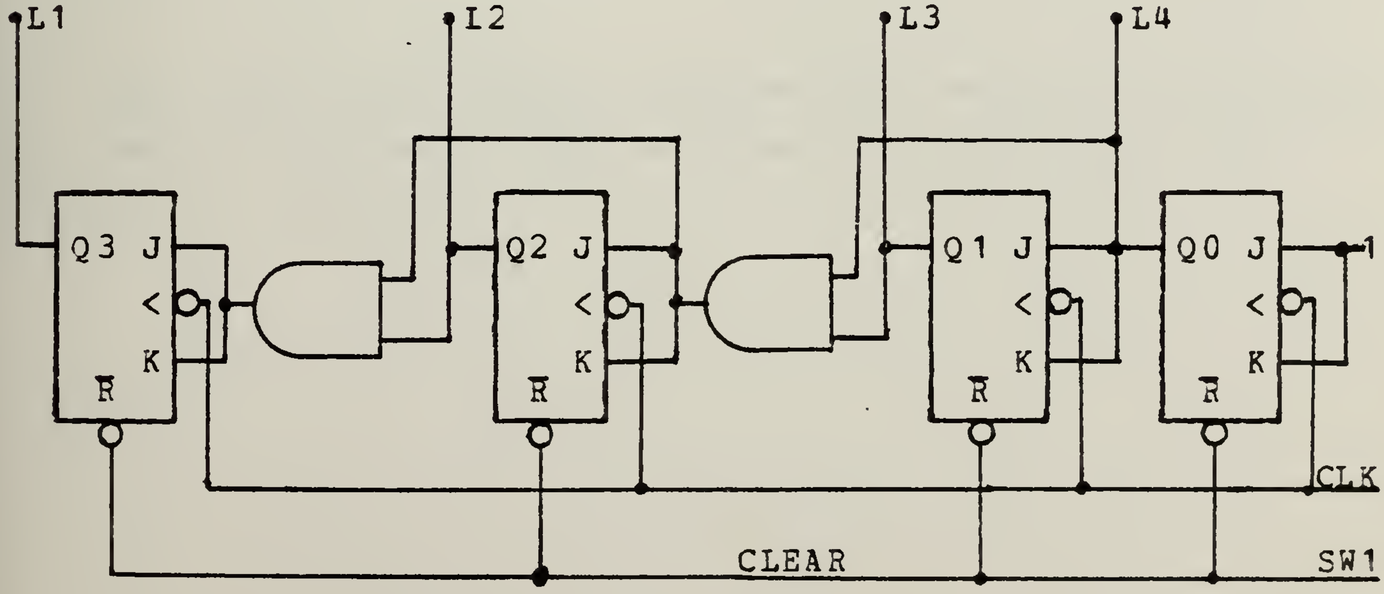
10.2.1 The Synchronous Binary Up-counter

A. Use two 7473's and a 7411 to construct the following counter. Does it perform as a modulo-16 binary up-counter?



10.2.2 The Synchronous Binary Up-counter with Ripple Carry

A. A counter which offers a compromise between the simplicity of a ripple counter and the speed of a synchronous counter is the synchronous counter with ripple carry shown below. Construct and test this counter. Does it perform as a modulo-16 binary up-counter?



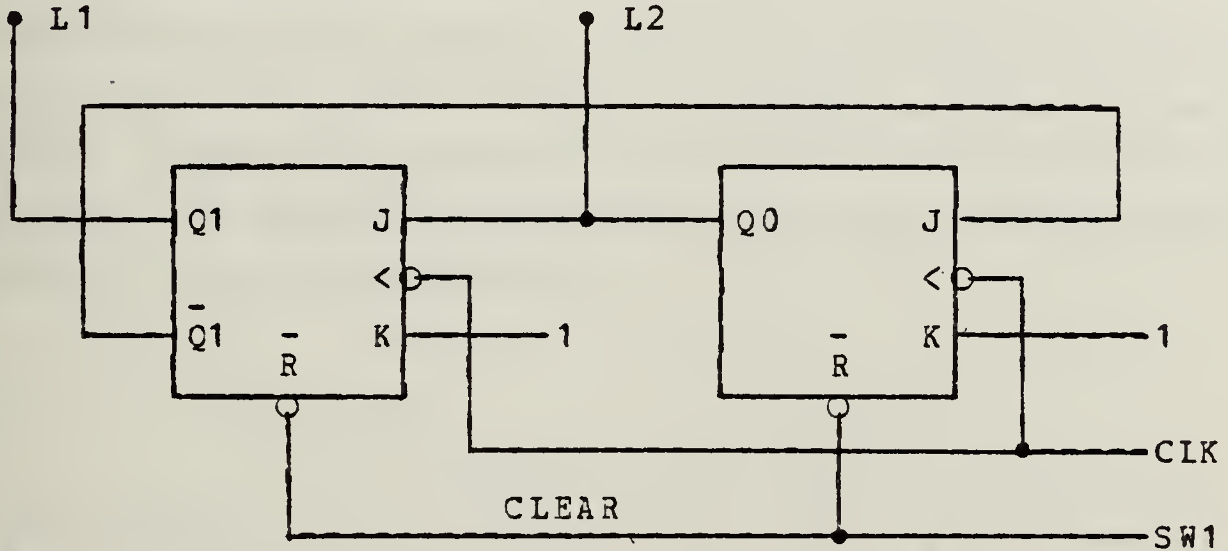
B. Discuss (or comment on) the speed (i.e., maximum clock frequency) and complexity of ripple counters; synchronous counters with ripple carry. Note: A flip-flop is slower than a simple gate.

10.2.3 The Synchronous Down-counter (optional)

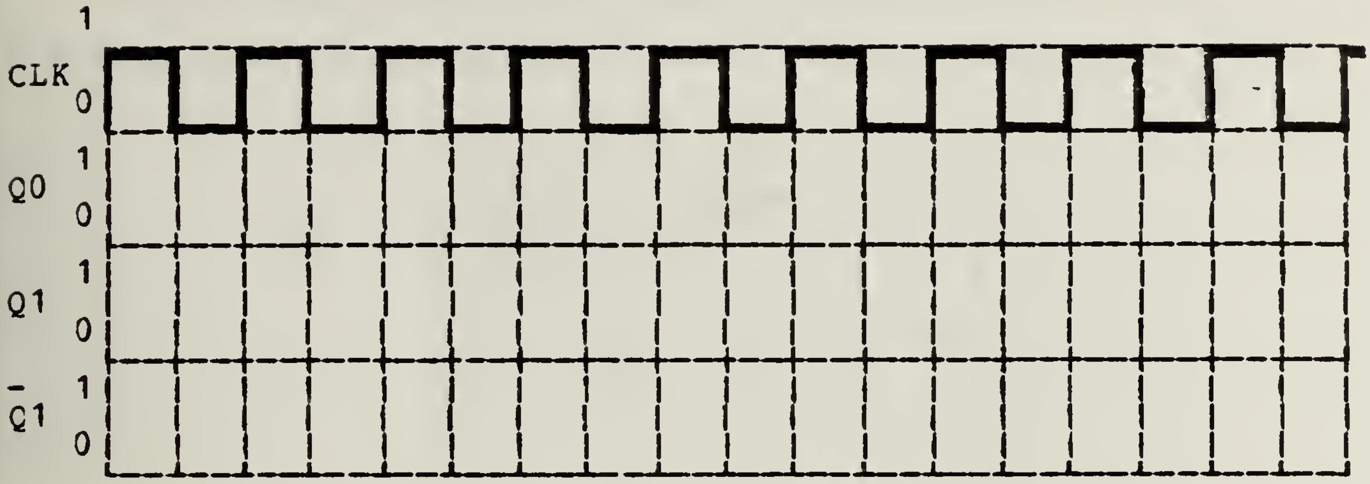
Modify (and describe the modifications of) the counter of the previous section so that it will count down. Don't alter the connections to the LEDs. Does the new counter operate correctly?

10.2.4 A Modulo-3 Synchronous Up-counter

A circuit for a modulo-3 up-counter is shown below.



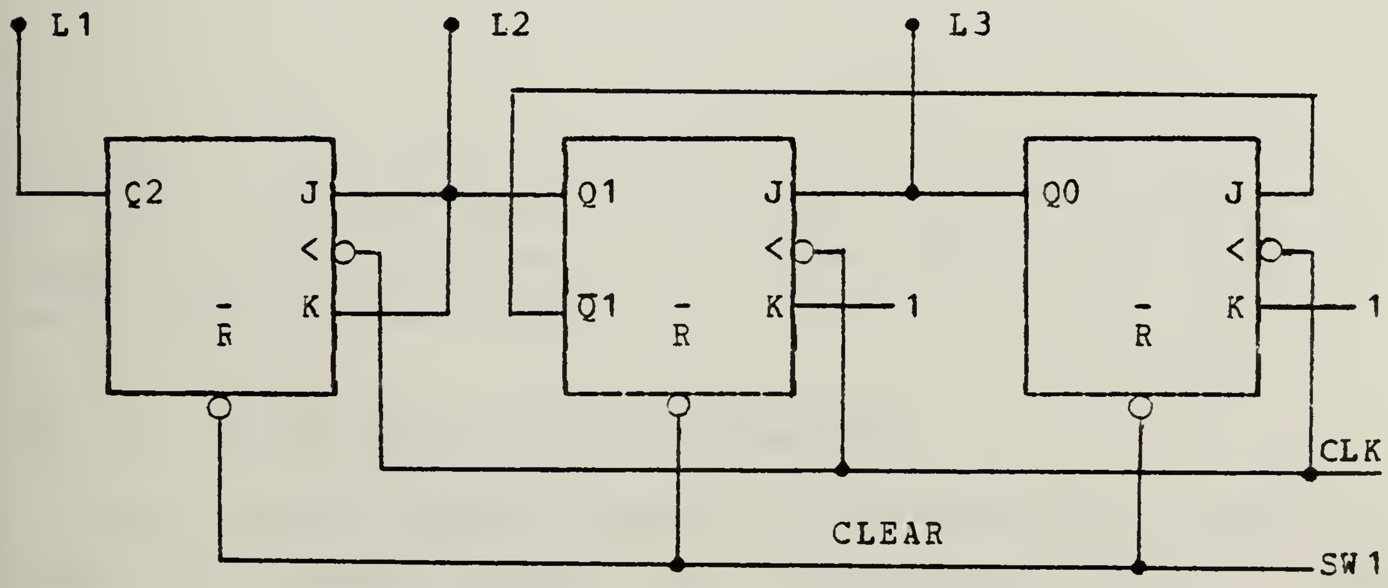
A. Show that this circuit operates as a modulo-3 up-counter by drawing the timing diagram. Does it count correctly: 00, 01, 10, 00, 01, ...?



E. Construct and test this counter. Does it behave as predicted by your timing diagram?

10.2.5 A Modulo-6 Counter

A modulo-6 counter can be obtained by adding an ordinary binary stage to the modulo-3 counter of the previous section. (Modulo-6 counters can be used to build modulo-12, modulo-24, and modulo-60 counters, which have obvious applications in time-of-day clocks.)



A. Build this counter and show, experimentally, that it is a modulo-6 counter. Complete the following table of counts.

Clock Pulse	Q2	Q1	Q0
0	0	0	0
1			
2			
3			
4			
5			
6			
7			

B. What code does this circuit use? In other words, what weights must be assigned to the three digits? (It is not the usual binary 4-2-1 code.)

10.2.6 A Modulo-12 Counter (optional)

Add another binary stage to yield modulo-12 counting. (This is not quite so easy as in the previous section.) The counter should still be synchronous. Draw the circuit. What weights must be assigned to the digits in this counter? If you built and tested this unit, did it perform properly?

10.2.7 A Modulo-5 Counter (optional)

See if you can design a modulo-5 up-counter along the lines of the modulo-3 counter you designed earlier. As well as three JK flip-flops, you will need one AND gate.

Section 11

ABBREVIATIONS

Hz: Hertz

IC: Integrated Circuit

kHz: kiloHertz

LED: Light Emitting Diode

LSB: Least Significant Bit

LSD: Least Significant Digit

MSB: Most Significant Bit

MSD: Most Significant Digit

TTL: Transistor/Transistor Logic

BCD: Binary Coded Decimal

Section 12

DEFINITIONS

Discrete: Consisting of distinct or unconnected elements.

Gate: A device that outputs a signal when specified input conditions are met.

Hertz: A unit of frequency equal to one cycle per second.

Integrated Circuit: A tiny complex of electronic components and their connections that is produced in or on a small slice of material (as silicon).

Inverter: A circuit that realizes negation. A circuit that performs logical complement.

Karnaugh Map: A mathematical tool used to visually portray the properties of Boolean functions and to simplify combinitional logic circuits or functions.

Latch: Name often used for flip-flop circuits that hold the circuit outputs at their previous state when the inputs are set to zero.

Section 13

TABLE OF DECIMAL MULTIPLES AND SUBMULTIPLES

Multiples And Submultiples	Prefixes	Symbols
10^{18}	exa	E
10^{15}	pecta	P
10^{12}	tera	T
10^{9}	giga	G
10^{6}	mega	M
10^{3}	kilo	k
10^{2}	hecto	h
10	deca	da
10^{-1}	deci	d
10^{-2}	centi	c
10^{-3}	milli	m
10^{-6}	micro	(Greek mu)
10^{-9}	nano	n
10^{-12}	pico	p
10^{-15}	femto	f
10^{-18}	atto	a

APPENDIX B
HEATHKIT DIGITAL LOGIC TRAINING DEVICE TUTORIAL

```
  **********************************
 ************************************
**************************************
***                                ***
***    INSTRUCTIONAL LABORATORY    ***
***                                ***
***                                ***
***                                ***
***                                ***
***            HEATHKIT            ***
***                                ***
***         DIGITAL LOGIC          ***
***                                ***
***        TRAINING DEVICE         ***
***                                ***
***                                ***
***                                ***
**************************************
 ************************************
  **********************************
```

TABLE OF CONTENTS

INTRODUCTION

Welcome to the Instructional Laboratory. This booklet is to assist you in familiarizing yourself with the HEATHKIT DIGITAL DESIGN EXPERIMENTER. This device can be used to breadboard (build) digital circuits using integrated circuits and connecting wires. The use of this device requires a fundamental knowledge of digital switching theory. A minimum understanding of Boolean Algebra may be sufficient if all you desire is a device to assist you in understanding or learning digital theory. Check out the HEATHKIT DIGITAL EXPERIMENTER and the HEATHKIT DIGITAL TECHNIQUES instruction books in Ingersol, room 224. For NPS students, these books and the DIGITAL DESIGN EXPERIMENTER form a complete digital electronics training course. Recommended courses of instruction to augment this book and the other material mentioned are: IS-2000, EE-2810, and CS-3010.

This book is written in programmed instruction format, so please follow the page prompts for maximum benefit.

WHEN READY TO CONTINUE, TURN THE PAGE.

1. CAUTION

A precautionary message must be inserted at this time. This design console is not itself designed to handle more components than will easily fit on the large terminal strip at the bottom. Therefore, do not "jumper" to components not on the HEATHKIT device.

WHEN READY TO CONTINUE, TURN THE PAGE.

In order to familiarize you with the DIGITAL TRAINING DEVICE, we will first describe the physical layout. In order to follow along, please use the device itself and the HEATHKIT assembly manual. Place the trainer where it is convenient to look at and open the manual to page 34.

WHEN READY TO CONTINUE, TURN THE PAGE

Section 1

HEATHKIT LOGIC TRAINING DEVICE

1.1 PHYSICAL LAYOUT OF CONSOLE

As you look at the top of the console, it is apparent that it is divided into seven sections. We will start at the top left and describe the function of each of the areas. First, however, notice that each of the top six sections has several plastic blocks mounted on them and each block has four holes in the top. These are called connector blocks and are used to make the electrical connections between sections and components. Each of the holes is electrically connected to the others in the same block, thus any hole in a block will connect to the signal or component as per the label directly above the connector. For example, each of the holes in the connector under the +12 label will provide a positive 12 volt signal.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.1.1 POWER SWITCH

Looking at the top left hand corner of the console, you will find the power on-off switch. This switch is on when the rocker is pushed down on the left and off when pushed to the right. Do not plug in or turn on the console until instructed to do so.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.1.2 LOGIC INDICATORS

The first area we will look at is labeled LOGIC INDICATORS. You will see four connector blocks ,as described earlier, and four light emitting diodes (LED) labeled L1,L2, L3, and L4. When connected to a logic circuit, these LED's will turn on or glow when a logical "1" or "HIGH" signal is applied. A logical "0" or "LOW" will extinguish the led.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.1.3 POWER SUPPLY SECTION

Directly below the LOGIC INDICATORS, you will see a section marked POWER SUPPLY. This is the part of the console that provides the required operating voltages for the integrated circuits you will be using to do designs and experiments. Please ensure that you understand this section prior to connecting any circuits together on the console.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.1.3.1 +12

This block provides a positive twelve volt source. It should only be connected to the +12 volt input pin (normally labeled Vcc) on integrated circuits that require +12 volts.

1.1.3.2 GND

The function of this block is to provide a complete connection for the operating power by allowing current to flow back to the power supply from the integrated circuits. The connection is normally to the GND pin of the integrated circuit.

1.1.3.3 -12

The -12 volt source is similar to the +12 source, and should only be connected to components that require -12 volts.

1.1.3.4 +5

This is a different kind of power source. It provides the +5 volt operating power as required by some integrated circuits and thus is quite similar to the +12 and -12 volt sources. The difference is that this block is also capable of providing a constant logic signal of "HIGH" or "1". This feature is necessary in some digital applications.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.1.4 LINE SOURCE SECTION

To the right of the POWER SUPPLY section you will find the LINE SOURCE section. The function of this section is to provide a digital signal (square wave) that varies at the wall socket frequency, normally around 60 HZ (cycles per second). An associated ground connection is provided for this signal also. This ground is the same as the ground provided in the POWER SUPPLY SECTION.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.1.5 CLOCK SECTION

The CLOCK section provides a source of constantly changing or switching logic signals, at one of three speeds. Experiment #2 demonstrates the clock operation.

1.1.5.1 CLK

This is the clock output that normally will be used. The signal is a square wave that switches at the frequency selected by the switch.

1.1.5.2 GND

This is the same ground connection as before.

1.1.5.3 CLK NOT

This signal is the logical complement of the CLK signal. It is provided for the instances when the "FALSE" or inverted clock signal is needed.

1.1.5.4 SWITCH

The switch allows selection of one of three operating frequencies for the clock; 1HZ, 1 KHZ, or 100 KHZ. A HERTZ (HZ) is one cycle per second, a KILOHERTZ (KHZ) is 1000 cycles, and thus 100 KHZ represents 100,000 cycles per second. If you have the LED's connected, you can see the effects of the switching circuitry at 1 HZ but the other speeds are too fast for the human eye to respond.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.1.6 LOGIC SWITCHES SECTION

The LOGIC SWITCHES section will be looked at next. These connection blocks and switches supply selectable logic level to the positions they are in now (A NOT and B NOT).

1.1.6.1 A and A NOT

These two blocks provide complementary signals, that is, when A is "HIGH", $\overline{A}$ is "LOW", and vice versa. A will be "HIGH" when the A switch is in the A position, and $\overline{A}$ will be "HIGH" when the switch is in the $\overline{A}$ (normal) position.

1.1.6.2 B and B NOT

The operation of these connectors and switch is the same as for A and $\overline{A}$.

A simple exercise will be performed later to clarify the operation of this section.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.1.7 DATA SWITCHES SECTION

The section below the LOGIC SWITCHES SECTION is called the DATA SWITCHES section. It consists of four connectors and switches. Each performs in exactly the same way, so only one will be explained. The DATA SECTION will be the source of logic input signals for your circuits.

1.1.7.1 SW1 (SW2,SW3,SW4)

The operation of these switches is such that if the switch is in the "UP" position (moved toward the LOGIC SWITCH section) a +5 volt logic signal is applied to the connector directly below the switch. Conversely, the "DOWN" position connects GND or a logical "LOW" to the connector.

As with the LOGIC SWITCHES section, an experiment will be performed that will clarify these features.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.1.8 BREADBOARD SOCKET

The last section of the console is the BREADBOARDING SOCKET, which is not labeled but is the long object full of holes located at the bottom of the console. This is where you will be inserting integrated circuits for design or experimentation. Look carefully at the socket and observe that it resembles several of the connectors we have seen elsewhere all connected together. It performs the same functions as the connectors but has one major difference. Notice that the socket is divided horizontally by a slot that separates two horizontal rows of holes. The socket has five holes per vertical section, and as before, all five holes are electrically connected to each other internally. When inserting integrated circuits in the BREADBOARD SOCKET, always straddle the slot with the two sides of the chip. NEVER make connections to external devices or sockets, since damage to the power supply and console may result.

WHEN READY TO CONTINUE, TURN THE PAGE.

Section 2
EXPERIMENTS

2.1 BEFORE PERFORMING EXPERIMENTS

Read this page before performing the following experiments.

1. It is now time to plug the HEATHKIT LOGIC DESIGN EXPERIMENTER in and apply power to it. First, ensure that the power switch is in the OFF (to the right) position. Now plug it in to any 110 volt grounded outlet.

2. Turn the power switch on and observe that the red power indicator light to the left of the switch illuminates.

3. When inserting and removing components, it is best to turn power off.

4. After components are in place, you can connect or remove wires with the power on or off without any danger.

5. If at any time you suspect the console is not operating correctly, TURN IT OFF, and report the problem to Professor Schneidewind or your instructor.

6. Remove integrated circuits with the extraction tool provided (looks like a tweezer) in order to prevent damage.

WHEN READY TO CONTINUE, TURN THE PAGE.

2.2 LOGIC INDICATORS EXPERIMENT

Refer to page 27 of the HEATHKIT MANUAL and connect one end of a wire to +5. Then connect the other end of the wire first to L1, then L2, L3, and L4. Each LED should light when you connect the wire to it.

WHEN READY TO CONTINUE, TURN THE PAGE.

2.3 CLOCK EXPERIMENT

Refer to page 28 and position the CLOCK switch to the 1HZ position. Connect a wire from L4 to CLK, and another wire from L3 to CLK NOT. L3 and L4 should now alternate on-off at a one cycle per second rate. L3 should be on when L4 is off and vice versa. Now position the switch to the 1KHZ position. You should observ second rate. L3 should be on when L4 is off and vice versa. Now position the switch to the 1KHZ position. You should observe that the LED's appear to be on continuously.In fact they are switching faster than your eyes can detect. With the switch in the 100 KHZ position, the LED's will also appear to light continuously but at a brighter intensity.

WHEN READY TO CONTINUE, TURN THE PAGE.

2.4 LOGIC SWITCHES EXPERIMENT

Refer to page 29 and connect one wire from L4 to LOGIC switch A. Connect another wire from L3 to logic switch A NOT. Operate the switch and observe that when the switch is in the A NOT position L3 is lighted, and when the switch is in the A position, L4 is lighted.

WHEN READY TO CONTINUE, TURN THE PAGE.

2.5 DATA SWITCHES EXPERIMENT

Refer to page 31 and connect a wire from L4 to DATA SWITCH 1. Operate the switch and observe that in the UP position, L4 is on. In the DOWN position, L4 should be off.

WHEN READY TO CONTINUE, TURN THE PAGE.

2.6 D-TYPE FLIP FLOP EXPERIMENT

Refer to page 35 and carefully connect the wires required to build a D-TYPE FLIP FLOP, using the DM7400N integrated circuit. Page 36 has a schematic representation of the circuit. In order to ensure correctness of your connections, you can follow the following listing.

PIN 14 TO +5
PIN 13 SEE PIN 10
PIN 12 TO SW-1
PIN 11 TO PINS 9 AND 1
PIN 10 TO PIN 13 AND LOGIC SWITCH A
PIN 9 SEE 11
PIN 8 TO PIN 5
PIN 7 TO GND
PIN 6 SEE PIN 2
PIN 5 SEE PIN 8
PIN 4 SEE PIN 3
PIN 3 TO PIN 4 AND L1
PIN 2 TO PIN 6 AND L2
PIN 1 SEE PIN 11 To test the operation of the D- TYPE FLIP-FLOP, apply power to the console and observe L1 or L2 is on. Put Data switch 1 in the "up" position and cycle Logic Switch A. At this time L1 should be lighted. Now put Data Switch 1 in the "down" position and cycle Logic Switch A again. L2 should be lighted and L1 off. For this experiment, the Data Switch acted as the input signal and the Logic Switch acted as the Clock input. The flip flop has performed as expected, since a D-TYPE FLIP-FLOP is supposed to have as it's output the same signal that was at the input when the Clock signal arrived.

Additional information on the D-TYPE FLIP-FLOP and other circuits can be found in the HEATHKIT DIGITAL TECHNIQUES instructional material. For the adventurous, see experiment 11 of section 6 for an example of a J-K FLIP-FLOP frequency divider.

Section 3

CLOSING REMARKS

This booklet , the Heathkit Digital Design Experimenter Console, and the Heathkit Digital Techniques material are provided to assist you in learning and understanding how digital electronic circuits operate. If you discover an area where improvement is needed, please notify Professor Schneidewind or your instructor.

APPENDIX C

PROMPT 80 TUTORIAL

```
  *************************************
 ***************************************
*****************************************
***                                   ***
***                                   ***
***    INSTRUCTIONAL LABORATORY       ***
***                                   ***
***                                   ***
***                                   ***
***                                   ***
***                                   ***
***           PROMPT 80               ***
***                                   ***
***                                   ***
***                                   ***
***                                   ***
***                                   ***
*****************************************
 ***************************************
  *************************************
```

TABLE OF CONTENTS

INTRODUCTION

Welcome to the Instructional Laboratory. In this laboratory you may work with digital devices on a level from logic gates and the elementary electronics of computers to the fully integrated level of advanced microcomputer systems.

Through this series of texts you can progress from little or no knowledge of digital equipment to a working familiarity with advanced Automated Data Processing. However, this course of instruction was not designed to make an expert of the student. Extensive outside study is needed for that. For that reason, the text will present only simple examples and problems for demonstration. For the more serious student other books and reference manuals are available in the Computer Center Library and the Knox Library.

You should have been given the following material for this tutorial:

This textbook on the Prompt 80
The Prompt 80 machine
The Prompt 80 Microcomputer User's Manual by Intel
A PROM (programmable read-only-memory chip)

In this phase of the instructional series you will be exposed to a low level digital computer, the Prompt 80 by INTEL CORP. The Prompt 80 is based on the 8080 Central Processing Unit (CPU) chip, and is programmed directly from the keyboard. As a prerequisite to this manual you should have a working knowledge of the basics of computer arithmetic, including binary and hexidecimal notation, and a basic understanding of the functions of the computer.

At the end of this course you will be able to:

1. Turn on and initialize the Prompt 80.
2. Load and run given programs on the Prompt 80.
3. Understand the basics of machine language programs.
4. Write a simple machine language program from a given algorithm, enter, debug and run the machine language code on the Prompt 80.

Additionally, you will be given instructions on how to save your programs by "burning" a Programmable Read/Only Memory chip (PROM) with the Prompt 80.

Do not plug in the Prompt 80 until instructed to do so. Put the Prompt 80 in front of you and turn to Section I.

Section 1

THE PROMPT 80 COMPUTER

1.1 INTRODUCTION TO THE PROMPT 80

This section is a self-paced programmed guide to the Prompt 80. Each page has a short section to read, some action for you to perform, and further instructions. If you get confused return to the last page which you fully understood and try again.

When Ready, turn the page.

1.2 MAJOR DIVISIONS OF THE KEYBOARD

For this section of the tutorial you will be referred to the keyboard of the Prompt 80 itself and to the labels of the keys. There is a picture of the keyboard on page 1-6 of the Prompt 80 User's Manual. You should fold that page out of the manual and have it ready for reference to help you find the appropriate sections of the keyboard as you go along.

With the Prompt 80 in front of you, notice that there are six major divisions indicated on the face of the computer:

1. Register Display Group

2. Command Function Group

3. Reset, Interrupt Group (Unmarked group of three)

4. PROM socket

5. I/O Ports connector

6. Input/Output Group

Locate each of these 6 divisions.

When Ready, turn the page.

1.2.1 Register Display Group

The Register Display Group is used to display the contents of the CPU registers four bytes at a time, using hexidecimal notation. (If you do not understand hexidecimal notation, stop now and read the section in the Prompt 80 User's Manual on computer arithmetic.) The four bytes represent four registers in the 8080 CPU. The printing below the digits show the names of the registers. The selector lights beside the titles show which four of the registers are being shown at the time. In the first row are the labels B,C,D and E. In row 2 are H,L,Flags and A (Accumulator). The third row shows the two bytes of the Program Counter and the two bytes of the Stack Pointer. These are the names of the registers available in the Prompt 80's 8080 CPU. They are normally associated in pairs, B and C, D and E, H and L, and A and Flags. The Program Counter (PC) and Stack Pointer (SP) are always treated as pairs.

If these terms are unfamiliar, or you do not understand the concept of Program counter and Stack Pointer, read the User's manual, pages 3-1 to 3-13.

When Ready, turn the page.

1.2.2 Command Function Group

To change the register display from one set of registers to another, use the SCROLL REGISTER DISPLAY key in the COMMAND FUNCTION GROUP.

The COMMAND FUNCTION GROUP has eight digital readouts, 16 numeric keys and 8 command keys. Two of the command keys have two markings--Next and (,) and Execute/End and (.). These keys are the delimiting keys, and will be important in data entry.

The digital readout has three fields: The function field; the address field and the data field. Data from the numeric keys are entered into these fields until a delimiter key is pressed. More on data entry later in this section.

When Ready, turn the page.

The numeric keys have the 16 hexidecimal digits 0-9,A-F. These keys are used to enter data into the address field and data field of the display group. The F key also is a command key when pressed from the monitor state to select one of the 8 internal functions available to the user. These functions will be discussed later, in the advanced operations section.

When Ready, turn the page.

The command keys are used to command the built in monitor program in the Prompt 80. Four of the keys cause the function field in the display to change after clearing:

Examine/Modify Register	produces	Er
Display/Modify Memory	produces	dn
GO	produces	GO
Single Step	produces	SS (then PC)

The SCROLL REGISTER DISPLAY changes the display in the Register Display Group as previously discussed. The PREVIOUS/CLEAR ENTRY key either erases the current entry or backsteps to the next lower address, depending on the sequence and mode in which it is used.

When Ready, turn the page.

1.2.3 Interrupt/Reset Group

The Interrupt/Reset key group controls the monitor program built into the Prompt 80. This program actually monitors the state of the CPU and provides the housekeeping functions to enable the user to concentrate on the program he/she wishes to operate without concern for a starting program. The monitor controls the input and output for the keys on the machine, and interprets the key strokes of the user into the functions discussed in this section. The MON INT key interrupts the currently operating program and returns control to the monitor. The USR INT key interrupts the present operation and steps to the location 3C02 H. The programmer may install at that location a jump to any address desired to indicate the beginning of the interrupt handling routine. The SYS RST key reinitializes the Prompt 80 to initial turn on conditions. This key will "rescue" the computer from uninterruptable lock ups.

When Ready, turn the page.

1.2.4 PROM Socket

The PROM socket holds PROMs for reading and writing. It is a zero insertion force socket. The movable handle locks the PROM in the socket and releases it when desired.

1.2.5 I/O Port Connector

The I/O PORTS connector is provided to connect the Prompt 80 to an external device. This connector is used to connect to card or tape readers and printers, for example. The use of the I/O connector is beyond the scope of this text. See the User's manual for more information.

When Ready, turn the page.

1.2.6 Input/Output Group

The Input/Output Group has 16 LEDs, 8 each for Input and Output. The 9 keys below the lights control them. A lighted LED represents a binary 1 and an unlit LED represents a binary 0. Pressing the key below the appropriate LED changes the input from 0 to 1. To erase the 1's, the RST key resets ALL bits to 0.

When Ready, turn the page.

This concludes the basic description of the keys. You should now know where the keys are, and roughly what they do on the panel of the Prompt 80. If you need to you can repeat this phase. The foldout page of the Prompt 80 User's Manual can be kept folded out as a referance for the rest of this tutorial. If you are referred to a section of the board and cannot remember where it is the picture can help you find it.

If you wish to repeat this phase, do so now.

When Ready, turn the page.

1.3 APPLYING POWER TO THE PROMPT80

Move the Prompt 80 near an electrical outlet with 110 v. AC (normal mains). DO NOT PLUG THE PROMPT 80 IN UNTIL INSTRUCTED TO DO SO.

The On/Off switch is a rocker switch found near the fuse socket on the back of the unit. Before plugging the unit in to the wall socket, check to see that the selector switch next to the ON/OFF switch is in the 115 v. position. Turn the On/Off switch to Off and then plug the Prompt 80 into the wall socket.

Move the On/Off switch to the On position. The fan for air cooling should now come on, indicating power is applied. The digital readouts should light and quickly stabilize. If this does NOT happen, return the On/Off switch to OFF.

When the unit is properly turned on you will be ready to continue.

When Ready, turn the page.

The Register Display Group should display the following digits:

1 2 3 4 F F A A

And the selector light should indicate that this represents the data in the HL and FLAGS, A pairs of the CPU.

Depress the SCROLL REGISTER DISPLAY key in the COMMAND Group once. The group indicator should move down to indicate the registers now displayed are the Program Counter and Stack Pointer. The digits should read:

6 7 8 9 3 F 9 0

indicating that the Program Counter points to 6789 H* and the Stack Pointer is pointing to 3F90 H, the first available stack address. The PC value is not usable, but is an initializing value only.

Press the SCROLL key again. The selector LED now shows that the BC, DE pairs are displayed. The digits should read:

b b C C d d E E

Again, these are initializing values.

Press the SCROLL key again to return the display to the HL, Flags, A display.

When Ready, turn the page.

---

* The "H" indicates that the preceding number was written in hexidecimal notation. This convention continues throughout this text.

1.4 MODIFYING A REGISTER'S CONTENTS

To demonstrate how to modify the contents of a register pair we shall use the HL register pair. The Register Display Group now indicates that that pair contains 1234 H.

To modify that pair, or any pair, the Examine/Modify register command key is used. Press that key now, once.

The Function field of the digital command readout should change from a hyphen and now display "Er" indicating that we are going to examine or modify a register.

If it does, turn the page.

If it does not, press SYS RST and then Examine/Modify register. It should now be as described above. If it still is not, seek assistance.

The monitor now anticipates the input of which register to modify. The registers are numbered from 0 to B hexidecimal (0 to 11 decimal). To indicate the register to modify, you will press one of the numeric keys from 0 to B. To determine which key to press for each register, look at the Register Display group now. The registers are identified below the Register Display Group in three rows of four each. In addition to the name of the register, there is a small number beside the register name as they are displayed that indicates which numeric key to press for the corresponding register. In other words the B register is number 0, the C register is number 1, etc. Using this scheme the H register is number 4.

Press the 4 key once now.

When Ready, turn the page.

The address field of the command/function group should now display a 4, indicating the address of the register that is to be displayed and modified. To indicate that the data entry is finished, we must now press a delimiter key. Since we will be entering more information, the correct key to press is the Next (,) key. Press that key now.

When Ready, turn the page.

The data field of the command/function group now displays the number 12, the current contents of the H register. This is another confirmation that you have selected the proper register for modification. Look at the Register Display Group and confirm that the H register is filled with a "12".

Now that the monitor has the function and register information the last thing needed is the data to insert. Press any numeric key and observe that the number is displayed in the data field of the command display. Press another number and the first number moves to the left, with the new number showing up in the right most place. Press a third number and the left one disappears, shifted left out of the display. The second digit moves to the left and the new digit is again in the right most place. The shifted out digit is lost, and not remembered by the Prompt 80. This feature of shifting out can be used to correct incorrect entries without having to go through the sequence fully. Additionally, this will continue as long as you press keys. To make sure we are still together press the A key followed by the B key. The data field should now display the digits Ab. This is the value to be inserted to the H register.

When Ready, turn the page.

To accomplish the insertion of the data into the H register we must press a delimiter key. Either the (,) or (.) key will do. The difference between them is that the (,) does not end data entry and the (.) does. Since we are going to enter data into the L register, in addition to entering data into the H register, press the (,) key now.

The Ab in the data field should now go away, and the display of the HL register should now display that the H register has that value in it. The Register Display Group should now read Ab34FFAA. The command/function group now displays

Er. 5.34.

If it does, turn to the next page.

If it does not, redo the previous five pages until you understand and have the right results.

The monitor program has now stepped to the next register in an ascending sequence. That is, the monitor is ready to modify register number 5, as seen by the "5" in the address display, (which is the the L register) and is currently indicating the value in that register, 34. Check the Register Display Group again to verify that this is the contents of the L register. Press the C and D numeric keys in that order, observe that the data field of the command display now has the two characters in it, and then press the (.) key to indicate to the monitor that the data in the data field is ready for the L register, and that no more data will be entered.

The display in the command register will display the characteristic hyphen indicating no function selected, and the Register Display Group should read AbCdFFAA.

If it does, proceed to the next page.

If it does not, re-enter the data until the L register indicates the proper data.

When ready, turn the page.

1.5 ERROR MESSAGES

You may have discovered that the monitor may give you the message "Error" in the command display if you do not press the correct type of key. This is a simple alerting device to advise the user that the sequence of key presses is not proper. The monitor does NOT test for the validity of an acceptable key, only that an acceptable key has been pressed. If the Error message appears, the Clear Entry key will correct it and return you to the monitor program.

As a demonstration, press the Examine/Modify Register key twice now. At the second press the Error message appears, indicating that the second key press was inappropriate. To clear the Error message, press Previous/Clear Entry. The hyphen reappears.

When Ready, turn the page.

1.6 MODIFYING MEMORY LOCATIONS

The Display/Modify Memory works in much the same way as the Examine/Modify Register function. The difference between the two is that the Memory function must be given an address to modify or display, which will be present in the address field of the command display, and that data field will display the previous contents of that memory address before modification.

To modify an address in memory, the sequence of key presses is:

Press Display/Modify Memory
(produces dn in display)
Enter the address into the address field.
Terminate address with (,)
Enter data to go into address in data field.
Press (,) if more data for next address is to be entered, (.) if no more data is ready.

The PREVIOUS key will backstep through memory in the display mode. To step forward, simply enter no data to modify the address and press either (,) or (.) for the next address.

As an example, press the Display/Modify Memory key now. Observe that the display shows dn. Now input a four digit address (any number below 3000 will do. To end the address, press (,). The display will immediately show the contents of that address. Press PREVIOUS to see the address decrease by one and the contents change. Press NEXT (,) to return to the initial address and p ress (,) again to step to the next higher address.

When Ready, turn the page.

1.7 GO

The GO key transfers control to either an address entered from the keyboard into the address field or to the address in the PC register if no data is entered into the address field. The command is executed when the delimiter (.) is pressed.

The sequence of key presses is:

Press GO, observe GO in the function field
Enter an address, press (.) to go there or
Press (.) to go to the address in the PC.

An example of this function will be given in a later discussion.

When Ready, turn the page.

1.8 SINGLE STEP

SINGLE STEP moves through the program similarly to GO, except that each press of the key moves the PC one step. This can be a useful key to debug or examine the functioning of the program in a slow manner. The register contents can be checked at each step to see if the program is working as desired, or to locate the mistake.

An example of this will be given later in the text.

When Ready, turn the page.

This concludes the introduction phase of this text. In the next section you will be asked to enter a simple program from the keyboard, to run and test it, and then to load the same program from the PROM you have been given.

If you desire to stop in the middle of the text, this is a convenient place to stop.

Continue when you are ready.

Section 2

RUNNING A PROGRAM IN THE PROMPT 80

2.1 ENTERING A PROGRAM

Now that you know how to modify memory and registers, it is time to enter and run a program which is given to you. The sequence of operations is the same as in the previous section, using the Display/Modify Memory key of the Command/Function Group.

To begin, the first address of the program is 3DE0 H. To enter this address, press Display/Modify Memory, and observe the "dn" in the Function field. In turn press the 3, d, e, and 0 numeric key. The address field should now display 3dE0. The data for that address is 3E. Press the Next (,) key to indicate that there is more data to follow, then press the 3 and E keys in turn. The Command display should now read:

```
d n 3 d E 0 3 E
```

To enter the data to memory, press the Next (,) key.

The command field should now read:

```
d n 3 d E 1 X X
```

where the X's indicate that any value could be present. The address field has been incremented to the next higher address, and is ready for data entry to that address. The data for this address is 8b. Enter that data.

Now that you know how to enter data, turn the page and enter all the data as shown in the table. You have already completed the first 3 steps. Begin with step four. Note

the difference in step 21 where you enter a (.) to end data entry.

When Ready, turn the page.

1. ( ) Display/Modify Memory
2. (3) (d) (e) (0) (,) (3) (e)
3. (,) (8) (b)
4. (,) (d) (3)
5. (,) (e) (b)
6. (,) (0) (e)
7. (,) (0) (f)
8. (,) (5) (9)
9. (,) (d) (b)
10. (,) (e) (9)
11. (,) (c) (d)
12. (,) (f) (1)
13. (,) (0) (7)
14. (,) (0) (d)
15. (,) (f) (2)
16. (,) (e) (6)
17. (,) (3) (d)
18. (,) (c) (3)
19. (,) (e) (4)
20. (,) (3) (d)
21. (.) (this ends program loading)

When Ready, turn the page.

Now that the program is in memory you are ready to execute it. Press the GO key and observe that the function field reads GO.

To provide the proper address for the GO instruction, recall that the program you entered started at address 3ED0 H. Press the 3, e, d, and 0 keys in order and observe that the address field reads correctly. If it does, you are ready to begin the operation of the program. Before you execute the program, here is what it does:

> Each of the digital LEDs on the display board is made up of 8 sections, 7 straight bars and one period for decimal points. These 8 sections can be lit individually by this program. To choose which sections to light the input/output group keys will be used. One of the bars, or the decimal point, will illuminate when the corresponding I/O key is pressed. Pressing another key will illuminate another section. When you run the program you can experiment and find which keys light which sections. To extinguish all sections you will press the reset key. That is the sole function of the program you have entered. It tests all of the segments of all of the LEDs at once.

When Ready, turn the page.

Now you may begin the program execution by pressing the Execute/End (.) key.

You may experiment with this program as you wish. Make as many patterns with the I/O Group keys as you can. It is an interesting effort to determine the binary combinations that make up the decimal digits. Note also that not all segment combinations have any meaning, and that not all alphabetic letters can be formed with the led segments given.

When you are ready to end the program, turn to the next page.

To end the program, press the SYS RST key and the initial state of the computer is restored. If you wish you can examine the memory contents and see that the reset did not destroy your program and thus a GO(,) 3DE0(.) will start the program again.

WHEN READY, TURN THE PAGE.

This concludes the introductory phases of this manual. The following portion of the text is written in a more classic style, without the directions to perform any actions directly. The remainder of this manual will give you information on the Instruction Set of the 8080 CPU, how to convert an algorithm to machine language, and an introduction to the advanced functions of the Prompt 80 (such as reading and writing a PROM of your own.)

Section 3

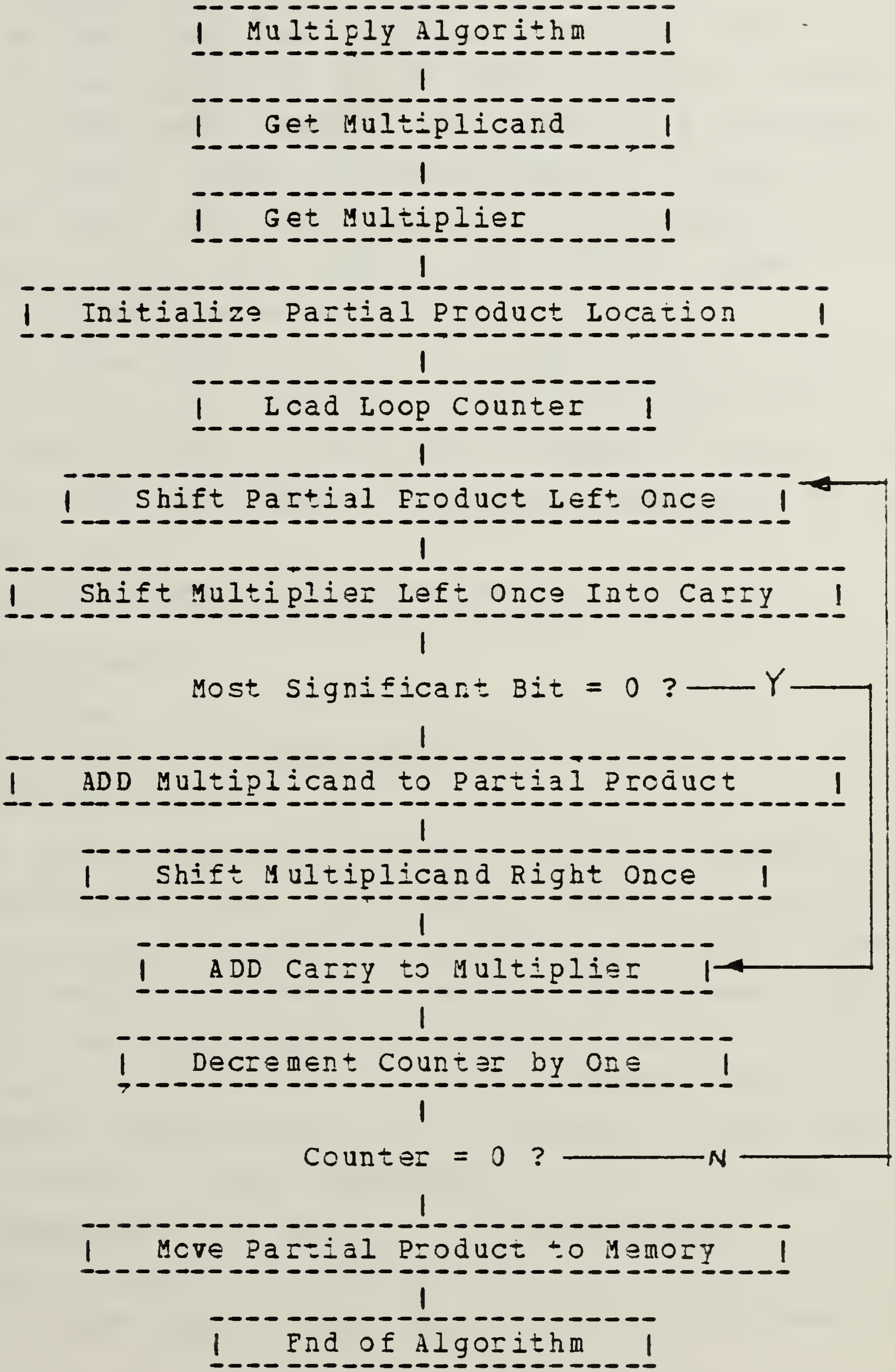
WRITING ASSEMBLY LANGUAGE PROGRAMS

3.1 ASSEMBLY LANGUAGE PROGRAMMING

Now that you can enter data into and manipulate the memory of the Prompt 80 you can begin to write your own programs for the machine. If you already know how to program in machine language, you may skip this chapter and continue in Section IV. If not, this chapter will present a simple example of how to program the Prompt 80. More independent study will be needed for the serious student of machine language programming.

To begin the process of writing a program one needs to analyze the problem to determine the input, process and desired output the program is to have. In this example we shall use a simple task as the process; the multiplication of a 16 bit number by an 8 bit number to produce a number whose bit size is no larger than 16 bits. The input will be the two numbers, in binary, and the output will be in binary (note that the Prompt 80 will display the answer as hexidecimal, but that the program will handle the data one bit at a time.) The process will be the typical binary multiplication , as shown in the figure on the next page.

3.2 MULTIPLY FLOW CHART



The first step in writing the program is to write (usually in longhand) the steps that we wish to perform, in order, and to desk check the solution. In this case, the first step in the algorithm is to fetch the multiplicand from memory. The second step is to fetch the multiplier from memory. Write these two steps down in a column on a piece of paper now. At this time we are not concerned with the specifics of HOW the Prompt 80 will do these two actions, but with WHAT we wish it to do.

Continuing, the third step is to initialize a partial product location in the computer that will be used later on. As the last step in this initializing process, we want to load a counter with the number of times the process is to be repeated, that is, 8.

Our plain text program should now read:

Get multiplicand.
Get multiplier
Initialize partial product location.
Set counter to 8.

With this much completed, we are ready to perform the actual multiplication loop. The first step in the loop is to shift the partial product to the left one place. Then the multiplier word is shifted to strip off the most significant byte. The first branch is about to occur. IF the most significant byte is a 0, the program jumps to the point at which the most significant byte is restored to the multiplier. The counter is then decreased by one. IF it is a 1, the intervening step of adding the multiplicand to the partial product occurs. In either event, the multiplier is then shifted to the right, and the counter is decreased by one.

If the counter is now NOT equal to 0, the loop is performed again from the point at which the partial product

was shifted left, as above. This process is continued until the counter reaches 0. At that point the answer is now in the partial product location, and all that remains to do is to store it in memory. As a test, write out the steps above, and then look at the next page to see if your program agrees with the one provided by the author.

3.3 MULTIPLY ALGORITHM

PROGRAM TO MULTIPLY TWO BINARY NUMBERS

```
Get multiplicand.
Get multiplier.
Initialize the partial product location.
Initialize counter to 8.
Shift left the partial product.  <----------------
Strip multiplier bit.                            |
IF it is 0, jump to ---------------------------  |
   Else, add multiplicand to partial product. |  |
Restore the multiplier bit.   <----------------  |
Decrement the loop counter.                      |
If NOT zero, repeat from -------------------------
Store in memory.

End of program.
```

Now that we have a program in English, we have to translate it into machine code, 0's and 1's. To assist in this process the developers of the 8080 CPUs provide a system of mnemonic devices to assist the programmer. The first step is to translate the English into the mnemonics, then transfer the mnemonics to hexidecimal equivalents. This avoids the necessity of making a much larger jump from English to machine language directly.

In the Prompt 80 manual you will find a summary of the mnemonics for the 8080 chip on pages 3-20 to 3-37. Although at first the code locks imposing, study will reveal that the instructions fall into several categories--accumulator instructions, byte instructions, word instructions and control instructions. The manual has divided the mnemonics within the four groups into functional areas with add, subtract, etc, grouped together.

The first step in our program was to get the multiplicand from somewhere in memory. For the time being we shall assume that the location will come later. For now we shall refer to the word instruction table for an instruction to fetch a word from memory. On page 3-31 we see the instruction mnemonic "LHLD." This machine instruction will load the HL register pair with the contents of a memory address. The memory address is to follow the mnemonic and has four digits. Of importance is the fact that the L register gets the information at that address and the H register the information at the address-1. This is a pattern followed throughout the 8080 family of CPU's. Note also that the next instruction on page 3-31 is "SHLD" and performs the inverse of "LHLD", i.e., it stores the contents of the HL pair in an address in memory. Looking at our program we see that we shall want to do that as a last step, so we should arrange to have the answer in the HL pair so we can store it conveniently using SHLD. Since the partial product will eventually be the final product, then the HL pair should

hold the partial product of the process, rather than holding the multiplicand or multiplier. After the LHLD instruction the HL pair will hold the multiplicand, so we have to move the data somewhere else to make room for the partial product. Between "Get multiplicand" and "Get Multiplier" now comes "Move HL to somewhere".

We have now added a step to our program, namely, MOVE THE HL PAIR TO (somewhere). The (somewhere) can be our free choice of any of the registers that can be addressed by the 8080 program instruction set. To make that decision, look at the word instructions that move data and pick one. The quickest of them is on page 3-30, "XCHG". It trades the value in the HL registers with the value in the DE registers and only takes 4 cycles to accomplish. This is efficient, and will be used from here on. This is NOT the only way to do it, and may not be the best in all applications. It works here and we shall use it.

Now that the multiplicand is safely in the DE pair, we have but to get the multiplier from memory. The program calls for the multiplier to be shifted eight times, and looking at the instruction set we see that the accumulator can be shifted directly (pg.3-23). Therefore, it is best to have it in the accumulator. Looking through the accumulator instructions we see "LDA" as one that loads the accumulator from an address in memory. This will be the next step in the program. The final step prior to the multiplication is the initialization of the partial product and counter. For the initial value of the partial product, recall that the partial product will be in the HL pair. We want to ensure that this pair is at 0 now. Looking for an instruction to put a value into HL we find "LXI reg16, data 16". This loads the register identified by reg16 with the number data16. In other words, "LXI H,0" will put zeroes in both H and L registers, as we want. Similarly, if we use the B

register as the counter, putting an 8 in it can be accomplished using the byte instruction "MVI B, 8."

Since we are now ready to loop it is useful to identify the position of the beginning of the loop with some temporary name (label) so we can rapidly find it again. For this exercise the loop will simply be called "LOOP" and that name should be written beside the instruction to shift the partial product left, since that is the first instruction in the loop. At this point we should reorganize the program into three columns, one for labels like "LOOP", one for mnemonics and one for the plain English description.

Do that now, and then turn to the next page for a check of your work so far.

```
LABELS     MNEMONICS       ENGLISH

Start      LHLD            Get multiplicand.
           XCHG            Move multiplicand to DE registers.
           LDA             Get multiplier.
           LXI H,0         Set Partial product to 0.
           MVI B,8         Load loop counter.
LOOP                       Shift left the partial product.
                           Strip multiplier bit.
                           IF 0, jump around next instruction.
                             else, add multiplicand to P.Prod.
                           Restore the multiplier bit.
                           Decrement loop counter.
                           If NOT zero, jump to LOOP
                           Store answer in memory.
```

Now that you have the process, see if you can find instructions to shift the partial product to the left. (As a hint, remember that a shift to the left of one position doubles the value of a binary number!)

Although many ways of accomplishing that task are available to the programmer, we have chosen was "DAD H", which simply adds the HL pair to itself, thereby doubling it. The result is that the pair is shifted to the left one position, as desired. Tricks like this one are frequent in machine language programming. Remember that the most obvious solution may NOT be the most effective. Search for efficient code whenever possible, either to decrease memory requirements or to speed up the operation of the program.

To strip off the multiplier bit the accumulator command "RAL" will shift the accumulator left through the carry bit. In this way the carry bit holds the most significant bit of the accumulator, the bit we are interested in at the moment. Since the carry bit can be tested for its value, we can branch at this point depending on the state of the carry bit. The desire is to jump around the next step if the carry is 0 (NOT SET). The mnemonic is "JNC addr16". Right now we do not know the address we wish to jump to, but it is the statement which says "Shift the multiplicand to the right." For now give it the name "NOADD" and the mnemonic will be "JNC NOADD" for the time being.

If the carry is 1 (SET), the next step is executed, namely, add the multiplicand to the partial product. The partial product is in HL, the multiplicand in DE. "DAD D" will add DE to HL, which is what we want. Now we want to restore the carry bit to the number in the accumulator. (This is not needed to perform the multiplication, but will restore the multiplier at the end of the program, a desirable result in some applications. This is call "non-destructive".) The command ACI 0 will add the carry bit, the accumulator, and 0 together, resulting in putting the carry bit back into its place in the accumulator.

To decrement the counter (the B register), at NOADD, we use "DCR B". After decrementing it, we test for the value

of B and if it is 0 we jump back to LOOP to continue. Like "JNC" above, the command "JNZ LOOP" will jump to LOOP if the Z flag (showing a 0 resulted from the DCR B) is NOT set. When the DCR B results in a 0 the Z flag is SET and the jump does NOT take place.

After looping 8 times, the B register is 0, the Z flag SET and the JNZ falls through the LOOP command (does not perform it) to the Store in memory. Remember that we planned to use "SHLD" to move the answer to memory. That concludes the programming of this multiplication. You should have a program that looks like the one on the next page to work with at this time.

```
LABEL     MNEMONIC         ENGLISH

Start     LHLD             Get multiplicand.
          XCHG             Move it to DE.
          LDA              Get multiplier.
          LXI H,0          Initialize partial product.
          MVI B,8          Initialize counter
LOOP      DAD H            Shift left partial prod.
          RAL              Shift the most sig. bit to carry.
          JNC NOADD        Skip if 0 in carry.
          DAD  D           Add DE to HL.(m'cand to P.prod.)
          ACI 0            Restores carry bit.
NOADD     DCR B            Decrement counter
          JNZ LOOP         LOOP if 0.
          SHLD             Store answer in memory.
```

End of program.

This is the heart of the program. Our task is not completed, however. All of those jumps and memory reads and writes need addresses. In addition, the 8080 CPU does not work directly on the mnemonics, but needs binary (or hexidecimal) code. That is the final stage of programming in machine language--coding the mnemonics into hexidecimal.

As you can see, each mnemonic in the Prompt 80 manual has associated with it a hexidecimal representation in one byte. In addition, some codes require additional bytes as information for the instruction. For example, our first instruction "LHLD" is shown as 2A XXXX, where the X's represent two additional bytes as an address. Since we have not decided where to put the input information, simply transfer the "2A____" to the paper with the program. We will fill in the addresses later. Now finish looking up the codes for the rest of the mnemonics, being careful to get the additional bytes correctly. When done, turn to the next page and compare your answer to that of the author.

```
MACHINE CODE   LABEL     MNEMONIC   ENGLISH
2A____         Start     LHLD       Get Multiplicand.
EB                       XCHG       Put it in DE.
3A____                   LDA        Get multiplier.
210000                   LD H,0     Initialize HL.
0608                     MVI B,8    Initialize B.
29             LOOP      DAD H      Shift HL left.
17                       RAL        Strip MSB of A.
D2____                   JNC NOADD  Jump if carry=0.
19                       DAD D      ADD DE to HL.
CE00                     ACI 0      Restores carry bit.
05             NOADD     DCR B      Decrement counter.
C2____                   JNZ LOOP   Jump if NOT zero.
22____                   SHLD       Store in memory
```

Now that we have a program with fixed length, we can choose a start address and fill in the blanks we had to leave before. For this problem, set the start address to 3D39 H and assume that upon commencement of the program the multiplicand will be at address 3E01 H and the multiplier at 3E00 H. We will store the answer back into 3E01 H when finished.

To insert these addresses remember the discussion about the process used to load registers and that the 8080 family of CPU's looks at information in reversed order. Therefore the addresses should follow the instructions in reversed byte order. That is, instead of 3E01, we write 013E. The CPU will reverse the address as a process of fetching the information. Hence, the first line now becomes "2A013E", indicating that the function "2A" is to be performed on memory location 3E01 H. Similarly, "3A____" becomes "3A003E", and the SHLD instruction is "22013E".

That leaves only the jump addresses to examine. For this program the start address will be 3D39 H. That means that the first byte in the program will be loaded in 3D39 H, and

each following byte goes into successive locations. To indicate the start address, the mnemonic "ORG" (for ORiGin) is used, with the address of the start of the program. To compute the JUMP addresses simply count down byte-by-byte to the JUMP destination and insert the data.

Finally, we must end the program somehow or it will continue to execute beyond the code we wrote. If this happens we cannot anticipate what will happen! To prevent this trip to never-never land, we insert a command that will perform some step we KNOW. RST 6 is the mnemonic for a breakpoint, and can be used here to return control to the monitor program so we can examine our results. The code for RST 6 is "F7" and is the final byte of the program.

Number the locations, fill in the jumps and then compare your work to the final product on the next page.

3.4 FINAL MACHINE LANGUAGE PROGRAM

```
ADD.  MACHINE CODE   LABEL     MNEMONIC   ENGLISH
                               ORG 3D39H
3D39  2A013E         Start     LHLD       Get Multiplicand.
3D3C  EB                       XCHG       Put it in DE.
3D3D  3A003E                   LDA        Get multiplier.
3D40  210000                   LD H,0     Initialize HL.
3D43  0608                     MVI B,8    Initialize B.
3D45  29             LOOP      DAD H      Shift HL left.
3D46  17                       RAL        Strip MSB of A.
3D47  D24D3D                   JNC NOADD  Jump if carry=0.
3D4A  19                       DAD D      ADD DE to HL.
3D4B  CE00                     ACI 0      Restores carry bit.
3D4D  05             NOADD     DCR B      Decrement counter.
3D4E  C2453D                   JNZ LOOP   Jump if NOT zero.
3D51  22013E                   SHLD       Store in memory
3D54  F7                       RST 6      Breakpoint to monitor.
```

Final program, with addresses.

Now that we have the program, we are ready to enter it, insert data in the appropriate locations, and run it. Using the techniques already discussed, enter the program as it is written above. When you have put the final byte into 3D54 H remember to press the (.) key to end the input phase. When you have done that turn the page.

Now that the program is entered into the machine we have to give it some data with which to work. In this program the data is read from two addresses in memory, and the assumption was made that the data would be placed there by some other action external to the program itself. For the first problem we will multiply 7 by 4, and will expect to get 28 as a result. Using the Display/Modify Memory key put the 4 into 3E00 H, and the 7 into 3E01 and 3E02. Since the 8080 CPU addresses memory in inverse order, the Least Significant Byte of the number goes into 3E01, and the Most Significant Byte goes into 3E02. The four digit representation of 7 decimal in Hexidecimal is 0007. The first two zeroes are the Most Significant Byte (MSB) and the 07 is the Least Significant Byte (LSB). In the case of the multiplier, 4, the byte to insert into 3E00 is 04 H. Enter these numbers now, and when done turn to the next page.

To run the program press the Go key, enter the address of the program, 3D39, and press (.).

The Command/Function readout should have changed to read:

P C. 3 d 5 5. 7 F

If it does not, re-check that you have entered the program properly. If the readout does not light, press the SYS RST key to break the program running and return to the turn on condition. The program should then be checked for entry errors using the Display/Modify memory function.

The answer is in the location we stored it, 3E01 H. Look there now, using Display/Modify Memory, and see if you got the correct result. Remember, the answer is in hexidecimal notation. You should see that 3E01 has 1C and 3E02 has 00. Putting these values together in proper order yields 001C H, and that evaluates to 28 decimal. If you cannot find these values, check that the program is properly entered and try again.

When Ready, turn the page.

Once you have the program properly running you can see that it does not take long to compute the answer. In fact, it is so fast that you cannot really see what is happening. To slow the process down we can use the Single Step function to move one step at a time through the program. We shall do that in a moment, but before that we should consider what the 8080 CPU actually does in each single step.

In the internal program of the 8080 CPU the instructions implement micro-instructions that actually carry out the data manipulations necessary to accomplish the events called for. This takes place through the fetch-execute cycle. In each step of the program we have written the 8080 CPU actually performs many operations, each directed by the micro-instruction set in the CPU. Look at the first instruction in our program "LHLD 3E01", transformed to hexidecimal code as "2A013E". At the start of the instruction execution, the program counter points to the location in memory where the first byte is stored. We gave that address as a part of the Go key instruction. The CPU first fetches from memory that byte, loads it into an internal instruction register and translates it into internal code. Since the instruction which results needs the address of the information to be put in the HL register pair, the CPU returns to the memory at one higher location than the 2A was found and fetches from that location the LSB of the address it needs. This process is repeated to get the MSB of the address from the next memory location. Now that the CPU has the address of the byte to fetch, it sends that address to the I/O unit, directing that the byte be fetched and placed into the internal data register. Once the I/O unit has completed this fetch, the CPU then transfers the data to the HL register. The last step is to update the program counter to point to the next instruction. Since the instruction "2A" actually takes 3 bytes, one for the 2A and two for address,

the program counter is incremented by 3, to point to the next instruction byte at 3D3C H. The fetch-execute cycle is complete for that first instruction.

This cycle repeats for every step in the program. When the instruction at location 3D54 is executed (RST 6) the control is returned to the monitor program and the cycling stops. In the Prompt 80 the Command/Function Group readout shows the value in the Program counter and the value that the program counter is pointing to. This is a function of the monitor program that is designed to aid the programmer.

Now we can single-step through the program to watch the data flow through the registers and memory.

When ready, turn the page.

Reinitialize the Prompt 80 by pressing the SYS RST key and then enter the data in 3E00 to 3E02 H as above. Use A0 H for address 3E00, 70 for 3E01 and 00 for 3E02. In this single step demonstration we shall multiply the numbers 70 H (112) by A0 H (160) to get 4600 H (17920).

To single-step press the Single Step key, and enter the address of the start of the program, 3D39. Do NOT press the (.) key at the end, but press (,) instead.

The Command/Function Group read-out should now read:

P C. 3 d 3 C. E b

indicating the the Program counter is pointing to 3D3C, and that that address contains the byte EB H. That is the second step in the program we wrote. Look at the value in the HL register as shown in the Register Display Group. It should read 0070, the bytes that were in the memory location 3E01 and 3E02. To check that the Program counter has actually moved, press the Scroll Register Display key once, and see that the Program Counter registers display 3d3C, as expected. Press the Scroll key once and see that the DE registers now have the value ddEE in them. Then press the Scoll key once again to see the HL pair.

Now press the Next (,) key once. Look at the HL register pair display now. It shows the value ddEE that was in the DE register. Press the Scroll key twice and note that the DE register now has the value 0070, transfered from the HL pair. The instruction was XCHG, and the instruction has been carried out. Press the Scroll key once to return to the HL register pair.

Press the Next key once. The instruction was LDA, load the Acculator with the contents of memory. See that the A register now has the value A0 in it. The program counter has again been incremented, as usual, and points to the next instruction at 3d40 H. Press Next again.

The instruction at 3d40 was to load the HL pair with zeroes. The display shows that this has been done. Press the Next key then Scroll to view the B register to see that it has the loop counter 08 in it.

Now all the variables have been initialized. Press the Scroll key again to view the HL pair. The Program Counter (PC) now points to the memory location 3d45, the first byte in the multiplication loop. The byte at that address is 29, an instruction to add HL to HL and put the answer in HL. This doubles the value in HL. Since that is presently 0, doubling it will not change the value. Press the Next key once and see that all that changes is the PC. The next instruction, 17, shifts the Accumulator right, putting the Most Significant BIT into the carry register. The value A0 is representative of the bit pattern 1 0 1 0 0 0 0 0. The Most Significant BIT is a 1, so that value should enter the carry bit of Flags. That bit will change the Flags Register to read an odd number, since the carry bit is the 0 bit in the Flags register, and the shift of the remaining bits in the A register will produce a pattern 0 1 0 0 0 0 0 0, or 40 H. Press the Next key once and see that these values show up. Any odd number in the Flags register shows that the carry bit is 1. /

The PC now points to 3d47, an instruction to jump to another address unless the carry bit is equal to 1. Since the carry bit IS equal to one, the jump will not take place. Instead, the PC will change to point to the next address in the program at 3d4A. Press the Next key and observe the change.

The instruction at 3d4A is DAD D, the command to add the value in the DE register to the value in the HL register, and put the results in the HL register. Using the scroll key note that right now the HL register = 0000 and the DE register = 0070. Press the Next key and see that the sum of these two shows up in the HL register.

The PC now points to 3d4b, an instruction to add the carry bit to the A register and the value 0. Press Next. Observe that unlike we planned, the 1 that we carried out did NOT get added to the A register. We shall come to this "BUG" in the program later. It is not fatal, and does not affect the result in this simple program. The "BUG" is in the program to demonstrate the difficulty of writing flawless code in machine language.

The PC now points to the byte at 3d4d, the decrement B instruction. To watch it work, Scroll the register display to the B register, and then press Next. The 08 should change to 07. The PC points to 3d4E, a Jump unless the zero flag is set. The Zero flag is NOT set at this time, so the jump occurs. Press the Next key to execute this step.

The PC now points to 3d45, the first step in the loop again. Press Next. The HL pair is doubled to 00E0. Press Next. The A register shifts left, shows 80, and the previously Most Significant BIT is in the carry bit. Since the Carry bit is 0, the Flags register is an even number.

Press Next. Because the carry bit was NOT SET, the jump to NOADD occurred, and the adding of DE to HL did NOT take place. Press Next twice. The B register has been decremented to 06, and the PC now points again to the beginning of the loop.

You may now continue pressing Next and watching the flow of data. When the B register finally reaches 00, the jump to LOOP at 3d4E does NOT happen, and the program continues at 3d51, an instruction to transfer the information from HL to the memory location 3E01. After that instruction is executed, the PC points to 3d54, a RST instruction. Pressing Next at that point transfers control out of the program we wrote, and sets in motion a sequence of instructions that begins at 0030 H. You may follow if you wish through the sequence, but it is not germane to this course. To stop, press the Execute/End key.

This concludes this section of the text. The BUG we identified will be addressed in the next section of the text. If you have not done so, you should execute the program you have entered with the data we were working with to its conclusion, and see that you get the correct answer.

This program not only contains the BUG discussed above, but also has limitations on the size of numbers. To demonstrate the limitation, what is the largest number you can have as a result? (Hint: when all the ones are set in the HL register, the next 1 added will recycle them to all 0's.) In addition, what should you get when you multiply 255 decimal times 512 decimal (FF H times 0200 H)? Try it and see what you get. Finally, can the program handle negative numbers? If so, how, and would it mean a change in limitations? If not, can it be made to do so?

The answers to these, and other questions, will be discussed in the next section.

This is a convenient place to stop if you wish to.

Section 4

ADVANCED OPERATIONS WITH THE PROMPT 80

4.1 ADVANCED CONCEPTS AND FUNCTIONS

In the previous section we left unanswered some questions about the program that was constructed to multiply two integers. Those questions concerned the limitations and usefulness of that program. In addition, there was a BUG in the program in that it did not restore the A register to the original value that it had had in it. These questions and the bug will be discussed in this section. In addition, the latter portions will discuss the advanced functions of the Prompt 80 that were deferred from the first section. If you have not entered the program to which we will refer, turn to page 42 and enter the program found there into the Prompt 80. This section will refer to page 42 and the program flow in the Prompt 80 computer.

4.1.1 Debugging

The purpose of the ACI 0 intruction at 3D4B was to restore the carry bit to the A register so that the A register would contain the same data at the end that it did when the program started through the looping process. In running the program, however, the A register was NOT restored, and ended up with 00 H. We will now attempt to correct that bug.

The instruction ACI 0 was originally presented as adding the carry bit, the immediate data of the 0 and the value in the A register together. That is the seeming definition given on page 3-20 of the Prompt 80 User's manual. However, the actual function of ACI 0 is to add the value 0 to the A register, put the result in the A register and IF THERE IS A CARRY, put it in the carry bit of the Flags register. That is NOT what we wanted. Looking through the instruction set we see no add instruction that adds the carry bit by itself to the A register. So what can we do to get what we want?

Since the ACI only occurs when we go through the loop, and we only go through the loop when the carry bit was set, we can safely assume that if we are in the loop the carry was set to 1. Since we can assume that, we can then add that 1 to the A register without having to add the carry bit. Simply change the ACI 0 to ACI 1. Now every time we go through the loop the A register gets a 1 added to it, as we wanted. To make the change, use the Display/Modify memory function to change memory location 3E4C to 01 instead of 00. Now load any data you wish into the locations at 3E00 to 3E02, and run the program. You will see that the A register ends up with the value originally fetched from 3E00, and that was our desire. We have de-bugged the program.

This process of de-bugging is necessary for almost every program written. This bug was simple to fix, but more often the program has to have some bytes added or deleted or both. Those changes may then change the addresses of labels

further down in the program, producing even more modifications that have to be made to the program. To avoid making too many changes to a program that has been translated into machine language, the programmer should design his program very carefully, and desk-check the flow of data before he encodes the program. In this way he can reduce the extensive program modification to a minimum.

4.1.2 Limitations

The program now runs, but still has limitations. The final page of Section III listed these problems to be considered:

1. What is the largest value that can be the result of the multiplication?
2. What happens when you exceed that number?
3. What about negative numbers?

In the first question the issue of RANGE is raised. The largest number expressable in 8 bits is 255 decimal (FFH). That is the largest number that can be held in the A register as the multiplier. The largest 16 bit number is 65535 decimal (FFFFH). That is the largest value that can be held in the register pairs, and is the largest multiplicand that can be accepted. However, the largest answer is NOT the product of these two, but the same as the largest multiplicand, 65535. That is because the same registers hold the answer and have the same limitations as the multiplicand.

But what does the program produce when the limits are exceeded? Use the program to multiply FFH times FFFFH (255 times 65535 decimal). The answer should be 16711425 decimal (FEFF01H), but that is beyond the ability of the registers and memory locations set aside by the program to hold. Since there is no provision in the program for this sizing error in the output, the Prompt 80 merely presses on in the manipulation of the data. It then shows the answer as FF01H, or 65281 decimal, and gives no indication that the capability of the program has been exceeded.

This problem of the range of accuracy of the program is one that must be faced by every programmer. The answer to the problem is usually taken in two ways--a limitation on the program is documented for the user to learn it and the program itself is set up to detect such over- and under-

flows and to issue warning messages that this has occurred. In integer work this is much easier to do than in floating point notations, and thus many simpler machines restrict operations to integers.

The final question raised the issue of negative numbers. The program as written can only handle positive numbers. If we use the two's complement notation, then the positive limits are reduced to 127 decimal in the multiplier and 32767 in the multiplicand. The negative numbers will be represented by the binary numbers with the most significant bit set to 1, i.e., in the multiplier by the numbers that in standard notation would have been 128- and in the multiplicand by those that would have been 32768 and higher. But does that make it possible to multiply negatives using this program? Try multiplying -1 times 255 (81H times 00FFH) The answer should be -255 decimal or (FF01H). What does the program give?

It is obvious that the program is inadequate for the multiplication of negative numbers. Since the fix for that problem is beyond our intention in this text it will not be discussed further. It is sufficient to say that the program, and indeed any program, has limitations built in to it and that these limitations need to be understood by the programmer and the user to prevent misuse.

4.2 ADVANCED OPERATIONS OF THE PROMPT80

The final discussion on the Prompt 80 is the description of the advanced functions of the machine. In the discussion on the function of the numeric keys it was stated that the 16 numeric keys were used to enter the hexidecimal digits and to chose the functions. Those functions selectable from the numeric keys are:

0. Read hexidecimal tape.
1. Write hexidecimal tape.
2. Write a PROM from memory.
3. Compare a PROM to memory.
4. Transfer PROM to memory.
5. Move memory (block).
6. Hexidecimal add/subtract function.
7. Byte search function.
8. Word search function.

To select one of these function, you have merely to press the appropriate numeric key with the hyphen in the command function group display. The prompt of the letter F followed by the number you have selected will appear in the command function group display. For more details on the other parameters that must be provided, see pages 4-15 to 4-21 in the Prompt 80 User's manual.

Of the functions available, the ones you are most likely to be interested in are the Read, Write and Compare PROM functions. These functions are used to transfer programs from memory to a PROM, from the PROM to memory, and to compare, byte for byte, the program in the PROM and the program in the memory. The next few paragraphs will discuss these functions.

To prepare for this section turn the Prompt 80 OFF and turn the page.

4.2.1 PROM Operations

To insert the PROM you have been given into the PROM socket, move the handle on the top left-hand side of the socket to the vertical position. This unlocks the socket. Examine the PROM carefully, without removing it from the protective foam. One end of the PROM has a distinguishing notch or dot on it. This designates the end of the PROM on which the #1 pin occurs. The end of the PROM with the dot or notch goes to the back of the socket. There are numbers painted around the socket to show where the 1, 12, 13 and 24 pins go. As long as the notch or dot is at the 1, 24 end the PROM is inserted correctly. Now that you know which way the PROM is to go, you are almost ready to install it.

The PROM is sensitive to static, and measures to protect it must be taken. Avoid handling the PROM by the pins, and if the day is such that static electricity is a problem, ground yourself by touching the chassis of the Prompt 80 while removing the PROM from the protective foam.

Remove the PROM from the foam, insert it into the socket in the proper direction and push the locking handle toward the back until the PROM is locked into the socket. IT SHOULD NOT BE NECESSARY TO USE FORCE ON THE CHIP OR SOCKET HANDLE.

When ready, turn the page.

Now that the PROM is safely in the socket you can turn the Prompt 80 back on. The display should light up exactly as before. The presence of the PROM makes no difference in the basic operation of the machine.

To transfer a program from the chip to the memory the #4 function is used. We shall now transfer the same multiply program that was written earlier from the chip to memory. First use the Display/Modify memory function to observe that the memory locations 3D39H to 3D55H presently do not have the program in them. Since the machine was turned off to insert the PROM the memory was erased and now has only "garbage" in it.

To invoke the function press the number 4 key on the keypad in the command/function group. The display should show:

F 4

Now we have to enter the address to which the PROM is to be loaded in memory. In the Prompt 80 this is address 3C00H, the lowest address available to the user. Enter 3C00, press (,). The command/function group display should show the number 3C00 until the (,) is pressed then it should show:

F 4.

Now enter the LAST memory location to be entered from the PROM, the last address available in memory, 3D55H. Press the (,) key. The display again shows:

F 4.

Now we enter the PROM address for the program to be loaded. In the PROM you have been given the address is 0000H. Enter that data but do not press the (,) key this time. The display should show:

F 4. 0 0 0 0

Press the (.) key. The program will load into memory from the PROM. The process takes a very short time and the display will flicker when finished. The Input/output group lights will flicker as the data is transfered from the PROM to the memory. To check that the program has loaded properly examine the memory locations from 3D39 to 3D55 and compare them to the program as we wrote it. It should be exactly the same.

To Write a program to a PROM, called in the vernacular "BURNING a PROM", the function F2 is used. There are a number of restrictions on the use of this function. For future reference the procedure can be found on pages 4-16 and 4-17, as well as a discussion on page 5-6 of the User's manual.

This concludes the manual on the Prompt 80 microcomputer. There are numerous reference manuals in the Laboratory for the serious student of machine language programming to use. In the next volume of the series in the laboratory you will be introduced to software which does the functions of assembling the mnemonics and allocating the addresses for you. These tools, called "assemblers," are an invaluable aid to the dedicated programmer. It is these tools which allow rapid development of higher languages, and increase programmer output.

APPENDIX D

SDK-85 TUTORIAL

INSTRUCTIONAL LABORATORY

SDK-85

TABLE OF CONTENTS

INTRODUCTION

Welcome to the Instructional Laboratory. In this laboratory you may work with digital devices on a level from logic gates and the elementary electronics of computers to the fully integrated level of advanced microcomputer systems.

Through this series of texts you can progress from little or no knowledge of digital equipment to a working familiarity with advanced Automated Data Processing. However, this course of instruction was not designed to make an expert of the student. Extensive outside study is needed for that. For that reason, the text will present only simple examples and problems for demonstration. For the more serious student other books and reference manuals are available in the Computer Center Library and the Knox Library.

WHEN READY TO CONTINUE, TURN THE PAGE.

Section 1

SDK-85 SYSTEM DESIGN KIT

1.1 THE SDK-85 SYSTEM

The SDK-85 microprocessor development system is designed around the INTEL 8085A family of integrated circuits. This set of components represents a major technological improvement over it's predecessor, the 8080 family. In the interest of retaining its investment, INTEL designed the 8085 series to be upward compatible with all existing 8080 devices and software. What this means is that any equipment and programs that were developed for the 8080 will function (and do so more efficiently) on the 8085A. The 8085A family was developed with newer technology and thus only 3 integrated circuits will replace 26 components which were required in an 8080 circuit that performed the same job.

The 8085A is a much faster CPU than the 8080, and it provides the user with two additional instructions.

WHEN READY TO CONTINUE, TURN THE PAGE

1.2 CAUTION

Before you apply any electrical power to the SDK-85, you should read this page in its entirety.

In order to use the SDK-85 you must provide a source of +5 volts DC electrical power. In the Instructional Laboratory this may be accomplished by the use of the DIGI DESIGNER console which has a built in power supply.

First, ensure that power to the DIGI DESIGNER is turned off, then connect the red wire from the SDK-85 power supply section to the +5 volt terminal post on the DIGI DESIGNER. Connect the green wire from the SDK-85 to the GND (ground) terminal of the DIGI DESIGNER. Verify your connections before applying power to the DIGI DESIGNER. You can now use the SDK-85 for experimentation and design.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.3 EQUIPMENT NEEDED

You may need the following items in order to complete this tutorial:

INTEL SDK-85 User'S Manual

Microcomputer Experimentation With The
INTEL SDK-85 by Leventhal and Walsh.

INTEL MCS-85 User's Manual

INTEL SDK-85 Design Kit

DIGI-DESIGNER Console

If you do not have these items , please acquire them from your instructor. -

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4 FUNCTIONAL COMPONENTS

In order to familiarize you with the SDK-85, it is appropriate that you first gain some knowledge of the component parts of the design kit itself. Therefore, at this time, place the SDK-85 where you can easily refer to it while you are reading.

Notice that the right side of the board is semi-populated with electronic devices and the left side is unpopulated. The left side is available for design, experimentation, and expansion of the basic board. Advanced students may find a use for the left side of the board, and for guidance are referred to the publications listed previously.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.1 POWER SUPPLY SECTION

The POWER SUPPLY section is located at the top right corner of the circuit board. It is the place that electrical power from an external device is provided to the SDK-85. As discussed earlier, +5 volts dc and a return path (ground) are required.

If you have not done so, you may now connect the SDK-85 to the DIGI DESIGNER. Use the instructions in the CAUTION page at the beginning of this tutorial.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.2 TTY INTERFACE

Slightly below and to the right of the power supply section is a group of components labeled "TTY INTERFACE". These components form the interface circuits needed to connect the SDK-85 to a Teletype terminal. This feature is not implemented in this laboratory, so no more will be said about it.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.3 CLOCK CIRCUITS

Near the center of the board, and above the large integrated circuit labeled "CPU", you will see some discrete circuit components and a flat metal box that is the crystal. These items form the external timing circuitry for the operation of the 8085A CPU.

This area is one of the major improvements of the 8085A over the 8080 family of components. The 8085A contains the majority of clock circuitry on the integrated circuit itself, while the 8080 required many more external components to generate the necessary clock and timing signals.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.4 ADDRESS DECODER

To the right of the timing crystal is the small integrated circuit known as the ADDRESS DECODER. The function of the decoder is to determine what address in random access memory (RAM) the CPU is trying to read from or write to. A chip enable signal is then generated to select the appropriate memory chip. In addition, the address decoder will enable the read only memory (ROM) and the keyboard decoder circuitry when they are selected by the CPU.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.5 CPU

The 8085A Central Processing Unit (CPU) is located below the timing circuitry and the address decoder. It is the large, 40 pin integrated circuit labeled CPU on the circuit board.

As in all computers, the CPU is the "BRAIN" that performs the work for the system. All other components are in support of the CPU chip. The 8085A CPU will control the input and output of instructions and data. It also de-codes and executes instructions and acts as the system controller.

For a complete set of 8085A instructions, see the MCS-85 User's manual.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.6 READ ONLY MEMORY

Directly below the 8085A CPU chip you will find the system Read Only Memory (ROM) chip. This component is labeled "PROM (ROM) I/O" and contains the system monitor. The monitor will be discussed later in this tutorial.

Also provided on this integrated circuit are two ports which can be individually programmed as either input or output ports.

The ROM resides between memory address locations 0000 and 07FF (hexadecimal). It is a permanent or non volatile memory chip and retains its information when electrical power is removed.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.7 RAM I/O

Below the ROM you will see a 40 pin integrated circuit that is labeled "RAM I/O". This is 2K bits (K=1024) of random access memory, which equates to 256 words of 8 bits each. This memory is NOT permanent and will lose any information stored in it if power is removed. The system RAM is used to store instructions for the CPU to execute, data to be operated on, and the results after computations are performed.

The installed RAM resides at memory locations 2000 to 27FF (hexadecimal). You will see a place above and below this memory chip which is provided for expansion by the addition of two more RAM chips.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.8 KEYBOARD AND DISPLAY

The remaining section of the board contains the Keyboard and light emitting diode (LED) display device. The two integrated circuits perform the keyboard decoding and provide the correct signals to the display. The group of discrete components directly above the LED unit provides the driver voltages necessary for the LED segments.

The display consists of a six-digit LED, and can be used to view input or output data, CPU registers, instructions, and the contents of memory locations. The display can function under user control or by CPU commands. The different keys will be explained separately.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.8.1 NUMERALS AND THE EXAM REG KEY

On the bottom right corner of the SDK-85 you will find 24 white keys arranged in 4 rows of 6 columns. The 4 right-most columns are the Numeric Keys and are labeled "0" through "F" (Hexadecimal). You will notice that some of the numeric keys have additional writing on them; for instance, the 8 key is also marked "H". These keys can be used in conjunction with the EXAM REG (EXAMINE REGISTER) key to determine the contents of the CPU registers. See below for a listing of all dual function keys. As an example, pressing the EXAM REG key followed by the 4 key will display the contents of the SPH register which is the eight most significant bits of the CPU stack pointer (stack pointer high).

The keys A through F will also display CPU register contents when used with the EXAM REG key, but they are not double marked because the registers they are associated with correspond to their Hexadecimal notation. Key A can represent the number 10 (Hex) or register A of the CPU.

Functions Of the Keys

KEY	FUNCTION
3	I (Interrupt)
4	SPH (Stack pointer-high)
5	SPL (Stack pointer-low)
6	PCH (Program counter-high)
7	PC (Program counter-low)
8	H (Memory address-high)
9	L (Memory address-low)
A	A (Accumulator)
B	B (Cpu register B)
C	C (CPU register C)
D	D (CPU register D)
E	E (CPU register E)
F	F (CPU flags byte)

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.8.2 RESET KEY

The RESET key is used to generate a control signal that will cause the computer to enter a 'start-up' program. When the RESET key is pressed, the display will read - 80 85 and control of the computer is passed to the monitor program in the system ROM (Read Only Memory). The Monitor program will allow the user to place programs and data in memory, execute programs, examine and modify the contents of RAM, and examine the contents of the CPU registers.

The RESET key also resets all registers and flags, sets all I/O (Input/Output) ports to Input mode, and disables interrupts. If you should want to examine CPU registers or flags after executing a program, DO NOT PRESS RESET after the program is finished.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.8.3 SINGLE STEP KEY

The SINGLE STEP key will allow you to execute a program one step at a time. Pressing the SINGLE STEP key will first cause the computer to enter the single step mode. Then pressing the NEXT key will cause execution of the instruction that was in the LED display, and the display is updated to show the next instruction to be executed. This mode of operation is good for de-bugging and to allow examination of CPU registers and flags at a specific point in the program.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.8.4 SUBST MEM KEY

The SUBST MEM (Substitute Memory) key is used to examine the contents of memory. In order to examine a memory location, the SUBST MEM key is pressed and then the Hexadecimal address of a memory location is keyed in. As you enter an address, notice that the address is displayed starting on the right side of the display and moves to the left as subsequent digits are entered. Once an address is entered, the contents of that address are displayed when the NEXT key is pressed.

It is important that you remember that all addresses have 4 digits and all data has 2 digits. The display always indicates information in Hexadecimal form.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.8.5 NEXT KEY

After an address has been entered by use of the SUBST MEM key, the contents of that address is displayed when the NEXT key is pressed. From that point on, successive presses of the NEXT key causes the contents of succeeding memory locations to be displayed.

The NEXT key also functions as the single step execution key as mentioned earlier in this tutorial.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.8.6 VECT INTR

The VECT INTR (vectored interrupt) key is used to cause a keyboard initiated interrupt to a program in execution. This key provides a jump to a RAM location which must hold the starting address of the interrupt handling routine. If this does not make sense to you--don't worry. This feature is normally used at a more advanced stage of programming.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.8.7 GO KEY

The GO key is used in conjunction with the EXEC key to tell the computer to execute a program. First the GO key is pressed, then the starting address of the program is entered, and the EXEC key is pressed. At this point the computer attempts to execute the program you specified. The GO key simply tells the computer to go to an address and do what it is told to do by the contents of that address.

WHEN READY TO CONTINUE, TURN THE PAGE.

1.4.8.8 EXEC KEY

The EXEC (execute) key tells the computer that it should execute a program. As already mentioned, the GO command would have already been used to set the computer to the starting address.

WHEN READY TO CONTINUE, TURN THE PAGE.

Section 2

CLOSING REMARKS

By this time, you should have some familiarty with the features and functions of the SDK-85. In order to gain some actual experience, please perform the laboratory experiments outlined in the Leventhal and Walsh book. That book provides an excellent presentation of the SDK-85 and the 8085A assembly language programming commands.

In addition, you can learn how a computer obtains data from an outside source, outputs data to an external device, responds to interrupts, and executes programs. The book provides examples of binary, hexadecimal, and decimal arithmetic as well as logical comparisons such as AND, OR, and NOT.

Additional information can be obtained from the two INTEL books provided.

You are now invited to turn on the power and begin assembly language programming on the SDK-85.

APPENDIX E
SYBEX SELF-STUDY TAPE LIBRARY

```
  ************************************
 **************************************
****************************************
***                                  ***
***    INSTRUCTIONAL LABORATORY    - ***
***                                  ***
***                                  ***
***              SYBEX               ***
***                                  ***
***            SELF-STUDY            ***
***                                  ***
***             LIBRARY              ***
***                                  ***
***                                  ***
***                                  ***
***      MICROPROCESSOR COURSE       ***
***                                  ***
****************************************
 **************************************
  ************************************
```

TABLE OF CONTENTS

Section 1

INTRODUCTION

The Sybex Self-Study Library is a set of independent study courses prepared by Sybex, Incorporated of Berkeley, California. Each course on microprocessors consists of a set of cassette tapes accompanied by a text. The time required to complete each course varies from 2.5 hours to 12 hours. These courses require a fundamental knowledge of microcomputer components and architecture, and may be beneficial for concurrent study with NPS courses: EE-2810, CS-3010, and CS-3200.

When a course of study has been selected, check out a cassette player and the appropriate tape / text set from In-224. The student may wish to bring pencil and paper for taking notes. Please do not write in the text books.

Section 2

LIST OF SELF-STUDY MICROPROCESSOR COURSES AVAILABLE

S1	Introduction To Microprocessors
S2	Programming Microprocessors
S3	Designing A Microprocessor System
SB1	Microprocessors
SB2	Microcomputer Programming
SB3	Military Microprocessor Systems
SB5	Bit-Slice
SB6	Industrial Microprocessor Systems
SB7	Microprocessor Interfacing Techniques
S10	An Introduction To Personal And Business Computing

This lab currently has available courses S3, SB3, SB5, and SB7. A brief overview of each course available in this lab follows.

2.1 SEMINAR 3 DESIGNING A MICROPROCESSOR SYSTEM

Presented by Rodnay Zaks

Time required: 2.5 hours

This seminar addresses how to interconnect a complete microprocessor system, wire by wire, including: Read Only Memory (ROM), Random Access Memory (RAM), Programmable Input-Output (PIO), Universal Asychronous Receiver Transmitter (UART), Microprocessing Unit (MPU), and clocks. Additionally, tradeoffs in addressing techniques and techniques applicable to all standard microprocessors are discussed.

TOPIC	MATERIAL COVERED
1. Comparative Microprocessor Evaluation	-Comparisons of Microprocessor classes including: -4 Bit Microprocessors -8 Bit Microprocessors -16 Bit Microprocessors -Bit Slices -8008 vs. 8080 CPU -AMD Microprocessors: 9080 -Z80 CPU vs. 8080 -Motorola 6800 -Intel 8085 -Intel 8048 / 8748
2. System Component Characteristics and Interfacing	-Static and Dynamic RAMS -ROM's including Field-Programmable (PROM), Fusible Links, Reprogrammable Memory (EPROM), and Electrically Erasable ROM (EAROM).

	-UART -PIO -Direct Memory Access (DMAC) -Programmable Interrupt Controller (PIC) -Programmable Interval Timer (PIT) -Asynchronous and Synchronous Interfacing
3. System Design	-Typical system organization -Typical microprocessor pinouts and signals -Connecting a system: i.e. CPU, Multiplexing, Data Bus, Address Bus, Memory, I/O -Standard microcomputer architecture: 6800, MCS-85 -One and two chip systems -Expanding the memory -Three I/O techniques: Polling, Interrupt, and DMA
4. Systems Development	-Cost / performance tradeoffs -How to speed up development -Hardware cost analysis -Basic software development -Software costs -Typical time-sharing prices -Use of emulators in developing a system -Debugging aids available

2.2 SEMINAR B3 MILITARY MICROPROCESSOR SYSTEMS

Presented by Rodnay Zaks

Time required: 6 hours

This seminar addresses topics on military or severe environment microprocessor systems utilized in military avionics, aerospace, naval, and industrial applications. The goal of the course is to cover all the main concepts, techniques, and some simple systems used in such militarized systems. Problems normally encountered in such designs are addressed and typical solution principles and practical implementations are proposed.

TOPIC	MATERIAL COVERED
1. Technical Introduction	-Definitions of terms
2. LSI Technologies	-Main goal is to underline the specific properties of some LSI technologies as they relate to possible choices of equipment. -Which kinds of technologies may be radiation hardened -Which kinds of technologies will be suitable for the portable systems such as aerospace applications.
3. Militarized Microprocessor Systems	-Several typical militarized militarized boards are presented. -Suitable features and design weaknesses of these boards are covered.

4. Militarized Microprocessors	-Which microprocessor chips (components) qualify for military applications.
5. Standardization	-Guidelines available for selecting such equipment and how to use the guidelines in choosing the components utilized in the system.
6. Building a system	-Procedures normally used to make the system ruggedized and resistant to the environment as per military specifications
7. Applications	-Various architectures used for military applications are discussed.
8. Reliability	-How to measure and predict reliability. -Methods used in military contracts for measuring and predicting reliability.
9. Testing	-The main concepts and testing techniques to ensure that systems meet specifications are covered.
10. Summary and Perspective	-The evolution of such products -What expectations one may have of forthcoming designs. -Differences between military systems and the current commercial/industrial systems.

2.3 SEMINAR B5 BIT-SLICE

Presented by Rodnay Zaks

Time required: 6 hours

The goal of this course is to show you how to use Bit-Slice components to implement efficient computer architectures with both traditional and non-conventional Bit-Slice applications.

The purpose of this course is:

1. To explain what Bit-Slices do and why they exist.
2. To demonstrate the procedure for designing with Bit-Slice.
3. To survey Bit-Slice devices on the market.
4. To survey the applications of Bit-Slice devices.

	TOPIC	MATERIAL COVERED
1.	Introduction	-Definitions of terms.
2.	Brief history of CPU Design	-The evolution of Bit-Slices
3.	Bit-Slice Principles	-The technological principles behind the architecture implemented in Bit-Slices.
4.	Bit-Slice In Detail / Building with Bit-Slice	-How to build a complete high performance central processing unit using an AMD - 2901 Bit-Slice chip.
5.	Other Bit-Slice Devices	-Bit-Slice devices available on the market, their merits and applications.

6.	Bit-Slice Applications	-Non-conventional applications -Cascaded slices on data paths paths for purposes such as very efficient high speed arithmetic word processing, string processing, and multi-channel memory searches through multi-port memories.
7.	Development Aids	-Simulators, PROMS, Assemblers
8.	Conclusions	-Questions and answers.
9.	Appendices	-Reference data on technologies circuitry, and components.

2.4 SEMINAR B7 MICROPROCESSOR INTERFACING TECHNIQUES

Presented by Rodnay Zaks

Time required: 6 hours

The goal of this course is to provide a comprehensive look at all the basic techniques required to interface a microprocessor system to the most commonly used peripherals. The student will learn:

1. How to assemble, interface, and connect a system.
2. How to assemble a complete CPU.
3. Input / output techniques.
4. Basic interfacing.
5. How to connect the peripherals: keyboard, LED, teletype, printer, cassette, floppy-disk, and CRT display.

TOPIC	MATERIAL COVERED
1. Introduction	-Basic concepts.
2. CPU Interfacing	-Assembly of the basic microcomputer board with the microprocessor clocks, drivers, memory, etc. -Connecting the basic board with all the peripherals.
3. Input / Output	-Review of basic input / output techniques and interconnects.
4. Peripheral Interfacing	-Interfacing with keyboards, LED's, teletypes, printers, floppy-disk, cassette, and CRT -Techniques and difficulties are addressed.

5. Communications	-Problems and solutions available for interfacing with communications equipment, time division multiplexing, modems, data links, etc.
6. Bus Standards	-Solutions available to simplify interfacing by use of standardized buses. -The IEEE 488, 583 CAMAC, and S-100 hobbyist buses are discussed.
7. Testing	-Brief coverage of testing and troubleshooting techniques associated with interfacing.
8. Evolution	-Summary of the trends of evolution and predictions of future interfacing techniques.

APPENDIX F

HEATHKIT H-9 TERMINAL TUTORIAL

```
  ************************************
 **************************************
****************************************
***                                  ***
***    INSTRUCTIONAL LABORATORY      ***
***                                  ***
***                                  ***
***                                  ***
***                                  ***
***            HEATHKIT              ***
***                                  ***
***          H-9 TERMINAL            ***
***                                  ***
***                                  ***
***                                  ***
***                                  ***
***                                  ***
****************************************
 **************************************
  ************************************
```

TABLE OF CONTENTS

INTRODUCTION

Welcome to the Instructional Laboratory. In this laboratory you may work with digital devices on a level from logic gates and the elementary electronics of computers to the fully integrated level of advanced microcomputer systems.

Through this series of texts you can progress from little or no knowledge of digital equipment to a working familiarity with advanced Automated Data Processing (ADP). However, this course of instruction was not designed to make an expert of the student. Extensive outside study is needed for that. For that reason, the text will present only simple examples and problems for demonstration. For the more serious student other books and reference manuals are available in the Computer Center Library and the Knox Library.

Section 1

HEATHKIT H-9 TERMINAL

1.1 INTRODUCTION

The Heathkit H-9 terminal is a dumb terminal with an internal RS-232 port that enables it to be used with a MODEM to communicate to any computer similarly equipped. The display is 80 columns wide, 12 lines high, upper case letters only. The screen is a white on black, with a protective cover. A repeating key, separate line feed and carriage return key, and an on/off line key make this a versatile terminal.

To use the terminal, ensure that the small connector on the back of the terminal is securely locked in place. (This connector is kept in place by a locking tab. It is unlikely that it should come loose. It is also keyed so that it can be installed in only one way.) The other end of the cable attached to the connector should be attached to the modem. Again, the connector is firmly attached by screws on this end, and only fits one way.

If the connectors are firmly attached, turn the terminal on using the on/off switch on the back of the machine. The terminal will warm up in a few moments and the cursor on the screen will be visible. While waiting for the terminal to warm up, check to see that the baud rate switch on the back of the terminal is set to "300". The "Baud rate" key on the keyboard should be down (depressed). If the "Baud rate" key is UP, the baud rate will be 110, but if it is DOWN, the rate selected on the back (300 baud) will be selected. The modem is designed for 300 baud and will not work at any other speed. The terminal itself is capable of 1200 baud.

To get this speed, move the switch on the back of the terminal from the 300 position to the "preset" position. In that position, the baud rate key will select 110 baud when UP and 1200 baud when DOWN.

1.2 <u>KEYS</u>

By now the terminal should be warmed up and ready to use. The top row of keys are function keys. The functions are:

Baud Rate--already discussed

Full Duplex--UP for half duplex, DOWN for Full duplex

Off Line--UP for terminal on line, DOWN for Off line

Xmit Page--The page, as displayed, is transmitted, starting at the cursor and continuing to the end of the page.

Plot--a diagnostic key, no function in normal use.

Auto Carry--UP for the cursor to stop at the end of a line, DOWN for the cursor to continue on the next line automatically.

Break--Terminates the Xmit page function, transmits a continuous "Space" at the serial output. Used to interrupt the sending computer from the terminal.

Erase Page--erases the page, returns the cursor to the upper left corner of the screen.

Erase EOL--erases the line the cursor is in from the cursor position to the end of the line.

In addition to these keys, there are control keys in the lower part of the keyboard as well. These keys are:

ESC--This key transmits an ASCII escape code.

CTRL--Used to transmit special control codes.

SHIFT--Shifts from UPPER to lower case (Note: this is the reverse of a normal typewriter. In addition, the display does NOT show lower case letters.)

SCROLL--after 12 lines are entered, if this key is depressed, an additional input will move the top line out and move the remaining 11 up one line, creating a new blank line at the bottom. If the key is UP, the screen will not scroll and additional data cannot be entered.

Line Feed--the cursor will move down one line and the ASCII character for LF will be transmitted.

Return--moves cursor to the first position of the line it is currently in, transmits the RT ASCII code.

Short Form--when DOWN, the display is changed to 12 lines, 4 columns of 20 characters. When UP, the display is 80 X 12 characters.

Rub Out--transmits a DEL ASCII character.

Rept--when used with another key, this causes the same character to be transmitted until the key is released. Normally the keys will transmit only one character for a keypress.

Home--returns the cursor to the upper left position, does not erase screen. Not transmitted.

Arrows--move the cursor the direction pointed to, one position per keypress. Not transmitted.

Section 2

COMMUNICATIONS WITH THE IBM 3033 AT NPS USING THE H9 TERMINAL

1. Turn on the terminal, checking to see that 300 baud rate is selected.

2. Turn on the modem.

3. When the CRT has the cursor visible, the terminal is warmed up and ready to use. Make sure that the FULL DUPLEX button is DOWN (Full Duplex) and that the modem is selected for Full duplex as well.

4. Dial the number of the NPS IBM 3033 (presently x3025).

5. When the tone is heard, cradle the handset in the modem with the cord at the end marked for it.

6. The screen should begin to display the following message: "VM/370 ONLINE".

7. When the message is fully visible, press any letter key on the keyboard.

8. The IBM will respond with a "!" and then a "." (this is the indication that the computer is ready to receive input.

9. Logon using your account number exactly as at a terminal in the center "L ####P", followed by "RETURN".

10. The IBM will respond by presenting the message, "ENTER PASSWORD", then type "********", return and overtype "HHHHHHHH" and then return again and overtype "SSSSSSSS". (On a Decwriter this produces a blob character like this S.) This serves to protect your password on that device, but does not protect it on the terminal.

11. Type your password followed by a "RETURN" and the console will eventually respond with a signon message and whatever profile exec's you have in your account. Note that some of the execs are very slow to start, and may require prompting with a keypress of some sort. Eventually the machine should yield the "R;" message, followed by the "." prompt that you may enter data.

12. For information, if this is your first contact via telephone, type "Q TERM", RETURN and see what characters do what functions. Particularly note which key is the character delete key, since this is the key that you must use to "erase" your typing mistakes.

13. FLIST and XEDIT are NOT available over the modem, but LIST and EDIT are. LIST produces a list of file names, with the usual ability to define the list by adding "LIST <filename> <filetype> <filemode>".

14. EDIT is a one-line text editor that uses the XEDIT commands that work on one line--in addition, CLocate, CFirst, CHange, etc, work. The display is limited to one line at a time, but you can type more than one line at a time by using "t#", where the # is the number of lines to type. Numbers alone will move you up or down the file appropriately.

15. Because the terminal does not display lower case letters, you may be surprised by the output in upper and lower case. The terminal CAN send lower case, and does so when the SHIFT key is depressed. This is the reverse of a normal typewriter, and is difficult to use for most people. If you intend to use the EDIT function to create text files for SCRIPTing, then you would do better with a different terminal than the H9.

APPENDIX G

HEATHKIT H-89 MICROCOMPUTER TUTORIAL

```
  ************************************
 **************************************
****************************************
***                                  ***
***    INSTRUCTIONAL LABORATORY      ***
***                                  ***
***                                  ***
***                                  ***
***                                  ***
***            HEATHKIT              ***
***                                  ***
***      H-89 MICROCOMPUTER          ***
***                                  ***
***                                  ***
***                                  ***
***                                  ***
***                                  ***
****************************************
 **************************************
  ************************************
```

TABLE OF CONTENTS

INTRODUCTION

Welcome to the Instructional Laboratory. In this laboratory you may work with digital devices on a level from logic gates and the elementary electronics of computers to the fully integrated level of advanced microcomputer systems.

Through this series of texts you can progress from little or no knowledge of digital equipment to a working familiarity with advanced Automated Data Processing (ADP). However, this course of instruction was not designed to make an expert of the student. Extensive outside study is needed for that. For that reason, the text will present only simple examples and problems for demonstration. For the more serious student other books and reference manuals are available in the Computer Center Library and the Knox Library.

In this manual you will be given a short course on the H-89 Microcomputer from Heathkit and the H-25 printer. In addition, you will be given a short course in the operating system which is used on that machine, CP/M from Digital Research, Inc. It is strongly recommended that you read this entire text before turning any of the equipment on or removing any of the diskettes from their jackets. If you are not familiar with the use of floppy diskettes, you should pay particular attention to the suggestions on the next to the last page of the text.

It is not the intent of this course to make you an expert on the intimate workings of the H-89, nor is it designed to make you an expert on CP/M. However, it is designed to provide you with sufficient information to allow you to work comfortably in the laboratory with the CP/M system and the H-89. As you use the system your confidence should grow.

Section 1

INTRODUCTION TO THE H-89

1.1 DESCRIPTION OF THE H-89

The Heath H-89 processor is based on the Z-80 CPU from Zilog, Inc. The operating system that the Laboratory has purchased is the CP/M system from Digital Research, Inc. This system is popular, and has the clever design that it supports transportable programs. A program written for CP/M will run on any machine that has CP/M, regardless of manufacturer, as long as it does not violate the rules of standard CP/M. In the commercial market there are over 500 programs available from vendors to run under CP/M.

Before applying power to the H-89, make sure no diskettes are in any of the drives, since the application of power to a drive while a diskette is in may damage the diskette or alter the data recorded on it.

1.2 POWERING UP THE H-25 PRINTER AND EXTERNAL DRIVES

To turn on the Heath H-89 you need to turn on the H-25 printer and the external disk drives first. This is a good rule for any system--power the peripherals first. The power switch for the H-25 and the external drives are on the back of the respective unit. From the front of the printer the switch is in the rear upper right corner, set in a small indentation of the outer case. On the external drives the switch is also on the back, in the lower right side. In each case the switch is a rocker switch. Position the switch to the ON position. On the printer the lights on the control panel will light, and the ribbon begin to wind to

the start position. On the disk drives there is no indication, but you may be able to detect a slight hum from the transformer in the power supply.

The control panel on the front right corner of the printer has 7 buttons and 4 lights for control and indications. The ON/OFF LINE button switch alternately places the printer in an on-line and off-line condition. In the on-line condition the printer will accept data and in the off-line condition it will not. Note: to operate the "form" switches of the printer it must be OFF-LINE. The TEST switch allows you to test the printer operations. In the interest of the laboratory only the operator should test the printer. As a user you should not have to operate the test switch. See the operation manual for the printer for details. The CLEAR BUFFER switch has two functions: if you press it for less than about 1/2 second it will clear the buffer of the printer; if you hold it in more that 1/2 second it will PRINT the buffer, then clear it. The RESET switch will reset any alarm from the printer and restart it. This switch is used to reset the printer after an out-of-paper, jammed paper or fault condition. The FORMS ALIGN switches will move the paper in the direction of the arrows near them. Use these switches to move the paper one line at a time in the direction of the arrow. These switches can be used to align the top of the paper with the print head. The TOP OF FORM switch is used to advance the paper to what the printer thinks is the top of the next page. Once there, use the FORMS ALIGN switches to actually line up the print head with the top of the paper. From that point on the printer should keep track of the top of the page.

The POWER light indicates that power is applied to the printer. The ON LINE indicator is lit whenever the printer is ready to accept data from the computer. The PAPER indicator indicates either an out-of-paper or jammed paper

condition. The FAULT indicator lights when the print unit (inside the printer) is open, the carriage is in an over-travelled condition (beyond physical limits) or the printer is overheated. IF THE FAULT LIGHT COMES ON SEEK ASSISTANCE BEFORE CONTINUING TO OPERATE. UNLESS THE FAULT IS CORRECTED THE PRINTER MAY BE DAMAGED.

1.3 POWERING UP THE H-89

Once the printer is on and operational (the POWER and ON-LINE lights lit only) and the external drives are powered, you may power up the H-89 itself. The power switch for the H-89 is in the back, at the right side of the machine as seen from the front. Again, a rocker switch is use for the ON/OFF switch. Move the ON/OFF switch to the ON position. The machine should issue a single beep, the disk drive may turn momentarily, then the screen will light up with the single prompt in the upper left corner,"H:". This indicates that you are in the internal monitor program of the terminal.

Load the SYSTEMS disk that you got from the operator into the external drive left side slot, labeled "A Drive," with the cut cut notch of the package down, the oval slot pointing toward the back of the machine, and the label pointing toward the left side of the drives. When the disk is fully inserted, close the door of the drive. It should not require any force to close the door. If you meet resistance, check to see that the disk is FULLY inserted into the drive.

Press the "B" key of the keyboard. On the screen you should see the word "Boot" appear beside the "H:" prompt. If it does NOT, press the OFF LINE key that is found in the upper left position of the keyboard. Press the "B" key again. If the word "Boot" does not appear, seek assistance.

Once the screen says "Boot", press the RETURN key on the keyboard. The external drive with the disk in it should begin to turn, the light on the drive door will light and after a few moments you will be given some information on the screen about the configuration of the system. When the system is fully booted, the standard CP/M prompt will appear "A>". This prompt indicates that CP/M is in operation, and that the presently active disk is disk A, the left hand external drive. The external drives are configured as drives A, B and C. The internal drive is configured as drives D, E and F. NOTE: THE INTERNAL AND EXTERNAL DRIVES DO NOT USE THE SAME TYPE OF DISKETTE. DO NOT PLACE A DISKETTE MARKED FOR INTERNAL USE ONLY IN THE EXTERNAL DRIVE, AND DO NOT USE DISKETTES MARKED FOR EXTERNAL USE ONLY IN THE INTERNAL DRIVE. THE ONLY DISKETTE YOU NEED FOR THE INTERNAL DRIVE IS THE SYSTEMS DISKETTE WHICH YOU HAVE BEEN GIVEN AND IS MARKED AS SUCH.

If the system fails to boot, try again. Press the SHIFT and RESET keys simultaneously to force the computer back to the "H:" prompt and type "B", followed by RETURN, again. If the computer will NOT boot at all, seek assistance from the operator.

Section 2

CP/M

2.1 BASICS

Once the computer has booted the operating system, you are in the CP/M environment. There are many excellent books on the CP/M operating system. If you wish to learn more about the system you are encouraged to read some of them. This manual will only provide you with the information necessary to run the applications packages provided with the system.

The "A>" prompt indicates that the active drive of the system is the A drive. To change drives simply type in the letter of the drive and a colon and press RETURN. The system will check to see that that drive has a disk in it and change the prompt to the letter of the new active disk. For instance, load an external disk in drive B and close the door. Press the B key, the colon key (:) and then the RETURN key. After a short interval in which the drive turns briefly, the screen will show the new prompt "B>". To return to the A disk type "A:" and RETURN. The A> prompt will return immediately because the system already knows that there is a disk in drive A. To read a list of files on the disk in the addressed drive, type the word "DIR", followed by the RETURN key. Upper and lower case do not usually matter to CP/M--it converts all commands to upper case. The screen will show a list of all the files on the active disk. For the directory of any other disk than the active one, type the word "DIR", a space, and the letter of the desired drive, followed by a ":" and then RETURN. The directory of the designated diskette will be displayed.

A filename in CP/M consists of three parts: the disk specification, the filename and the filetype, in the format "d:filename.filetype" where "d" is the disk drive name, filename is the primary name of up to 8 alphabetic or numeric characters, and filetype is the optional filetype of up to three alphabetic or numeric characters separated from the filename by a period. Legal variations are:

filename (a file on the current drive, filetype "   ")
d:filename (a file on drive "d", filetype "   ")
filename.typ (a file on current drive, filetype "typ"
d:filename.typ (a file on drive "d", filetype "typ")

If the drive specification is missing, CP/M will look for the file on the presently active drive only. To look at drive B from the "A>" prompt, for example, the "B:" MUST be in the file specification. To view all the files on the B disk from the "A>" prompt type "dir b:" and press return. To view files on drive C, type "dir c:", etc.

At this point it is appropriate to discuss briefly the concept of ambiguous and unambiguous file specifications. Unambiguous specifications are of the forms previously displayed. The name is specific to the drive, filename and filetype. In the ambiguous file specification some element of the specification is replaced by an asterisk "*" or a question mark "?". In these instances the system will perform the operation directed on all files with names that match the unambiguous part of the name, without regard to the part substituted for by the asterisk. The drive specification will NOT accept the asterisk. For example from the sequence "A>dir *.com" will display all the files on the A disk with the filetype ".com". Some CP/M functions will allow ambiguous specifications, and some will not. See the Digital Research literature on CP/M for specifics. The question mark is a "wild card" replacement for any letter or number in a filename. "A>dir stats?.fil" will display all

the files on the diskette which match the format, including variations such as:

```
stats1.fil
stats2.fil
statsd.fil
statsa.fil
```

It will NOT match "stats11.fil" however, because the length is longer on the file than the designated pattern. (Note that "A>dir ????????.???" is the same as "A>dir *.*")

To erase a file on a disk type "ERA " followed by the file you desire to erase. If the disk is not write protected or read-only, the file will be erased immediately. If you use an ambiguous specification, all files meeting the specification will be erased. (A>era *.* will erase ALL files on the A disk, A>era *.doc will erase all files with the filetyp ".doc" on the A disk, etc.)

To rename a file, use the command "REN". "REN" requires unambiguous specifications. The syntax for REN is "A>ren d:newname.typ=d:oldname.typ". (The convention of the new coming first is common to ALL CP/M commands and functions.)

To save a file to memory use the "SAVE" command. The syntax for the command is "A>save ## d:filename.typ". The ## indicates the number of "pages" of memory to save. A page of memory is 256 bytes. CP/M uses the first page for itself, and therefore the pages begin at the second page for the user. The SAVE command will move to the disk indicated the number of pages indicated, starting at the second physical page of memory and continuing to the page number "##" plus one.

TYPE d:filename.typ will display on the console the data of the file named. Names must be unambiguous. To stop display, press ANY key.

2.2 CONTROL CHARACTERS OF CP/M

The following table indicates the control characters in CP/M and their function. For more detail, see the CP/M users manual from Digital Research, Inc.

BACKSPACE	moves cursor one space back, erasing character
CTRL-C	aborts a running program, causes a warm boot from the D> prompt
DEL	same as RUB
CTRL-E	forces a physical carriage return, does NOT pass to CP/M
CTRL-H	same as BACKSPACE
CTRL-J	Linefeed, terminates input at console
CTRL-M	same as carriage return
CTRL-P	echoes all screen data to printer, the second CTRL-P terminates the function
CTRL-R	re-types the current line, as corrected
RETURN	carriage return
RUB	erases character immediately to the left of cursor echoes the character to the screen
CTRL-S	stops listing of file to screen temporarily, second CTRL-S resumes the listing
CTRL-U	cancels the present line
CTRL-X	deletes the present line
CTRL-Z	string or field separator

2.3 UTILITIES

There are several utilities included with the standard CP/M. These utility programs are on the disk which is marked "SYSTEMS DISK, Use in INTERNAL drive only". To run these programs you must address the programs with the drive designation "D:". The programs supplied are:

PIP.COM	(Peripheral Interface Program)
ED.COM	(EDitor program)
STAT.COM	(STATus of disks, files, etc.)

ASM.COM	(ASseMbler program for 8080 mnemonics)
DDT.COM	(Dynamic Debugging Tool)
FORMAT.COM	(Formats new diskettes)
SYSGEN.COM	(Installs system tracks on diskettes)

Of these utilities, PIP is the one most often used. It can be used to transfer information from one peripheral to another. The uses of PIP include printing to paper the contents of a file on diskettes, printing to the screen the input from a reader device, copying disk files from one source diskette to another destination disk, making backup copies of a diskette, etc. For more information on the uses of PIP, see the Digital Research literature on the subject.

The second most popular utility is STAT. STAT returns the status of files, diskettes, drives, peripherals, etc. You can use STAT to determine the size of existing files, the space left on a diskette, the size and type of drive connected, the logical input and output devices addressed by CP/M, etc. Again, see the Digital Research manuals for information.

ASM and DDT are tools for the programmer who wishes to write assembly language programs. The ASM program assembles standard Intel 8080 mnemonic language into machine code. DDT will display any portion of memory, allow it to be modified, and run with breakpoints and controls in the sequences. See the manual from Digital Research for more information. If you wish to learn about machine language code, see the Prompt80 tutorial of this series of tutorials.

FORMAT and SYSGEN are tools normally not needed by the applications user or programmer. If you need to format a new diskette, FORMAT is self documenting. SYSGEN is similarly self documenting. Note that NO USER DISKETTE SPACE is consumed by the system of CP/M. For that reason, there is nothing to be saved by NOT SYSGENing every diskette as it is

formatted. The policy of the laboratory is that EVERY diskette will have CP/M SYSGENed to it. If you need help, see the Digital Research literature.

2.4 POWERING DOWN THE SYSTEM

The last issue to be covered is the power-down sequence. It is important to remove power in a logical sequence to prevent inadvertent erasure of data on diskettes.

The first step in shutting down the H-89 is to remove all diskettes from the drives. Note that if you remove a diskette from a drive with a file still OPEN, the directory for that file is not accurate, and the file will be lost or damaged. To be sure, a good policy is to always return to the CP/M prompt "A>" before shutting down. This way all files are closed and diskettes are ready to be removed.

Once the diskettes are removed, and properly stored, turn off the equipment in the inverse of the power on sequence--main computer first, followed by peripherals. Once all equipment is turned off, close the disk drive doors to reduce the entry of dirt to the drives. The printer requires no special attention at shutdown.

The diskettes of a microcomputer are the key to the utility of the installation. They do, however, require certain care in handling. Do not touch the magnetic material visible through the holes in the covering with your hands or with any foreign object. Virtually undistinguishable dust particles can ruin a diskette and the read/write heads of the drive in which it is installed. Beyond physical abuse, the diskettes are also susceptible to magnetic fields. One of the most common mistakes is to put the diskettes on top of the computer, in the magnetic field of the Cathode Ray Tube of the display. Another enemy of diskettes is the telephone. When the bell rings, the magnetic field around the instrument is strong enough to erase a diskette if it is nearby. Always return the diskette to its jacket when out of the machine, and store carefully, even if it is needed again soon. These lessons have been learned with considerable "pain" by others. Be wise!

This concludes the tutorial on the H-89 Microcomputer and peripherals. For more detail, see the Heathkit operating manuals for the specific equipment. For applications packages, see the individual program instructions and manuals that accompany the software.

BIBLIOGRAPHY

Bork, A., *Learning with Computers*, Digital Equipment Corporation, 1981.

Buckingham, R.A.(ed), *Education and Large Information Systems*, North-Holland Publishing Co., 1977.

Couger, J.D., *Computers and the Schools of Business*, Business Research Division, School of Business Administration, University of Colorado, 1967.

Dinerstein, N., "On the Education of Information System Specialists", *SIGCSE BULLETIN*, v. 13, n. 2, pp. 21-25, June, 1982.

Glass, R.L., and de Nim, Sue, *The Second Coming: More Computing Projects Which Failed*, Computing Trends, 1980.

Gruenberger, F. (ed.), *The EDP People Problem*, Data Processing Digest, Inc., 1971.

Kearsley, G.P., Hillelsohn, M.J., and Sidel, R.J., "Microcomputer-based Training in Business and Industry: Present Status and Future Prospects", *Journal of Educational Technology Systems*, v. 10, n. 2, pp. 101-108, 1981.

Kraft, P., *Programmers and Managers, The Routinization of Computer Programming in the United States*, Springer-Verlag, 1977.

McCluskey, E. J., Jr., "Minimization of Boolean Function", *Bell System Technical Journal*, v. 25, pp. 1417-1444, 1956.

Mein, Wm. J., "On Students Presenting Technical Material to Non-technical Audiences in a Computer Science Curriculum", *SIGCSE BULLETIN*, v. 14, n. 2, pp. 97-101, February 1982.

Pratt, L.J., and Davis, L.D., "The Use of Computer Aided Instruction in the Teaching of Macroeconomic Principles", *SIGCUE BULLETIN*, v. 15, n. 1, pp. 2-14, January, 1981.

Pratt, L.J., and Davis, L.D., "The Use of Computer Aided Instruction in the Teaching of Macroeconomic Principles--an Update", *SIGCUE BULLETIN*, v. 16, n. 1, pp. 16-21, Summer, 1982.

Quine, W. V., "A way to Simplify Truth Functions", American Mathematics Monthly, v. 62, pp. 627-631, 1955.

Sayle, E.F., "Assessing your Data Management Needs", Desktop Computing, n. 18, pp. 38-41, March, 1983.

Sheppard, J.G., "Automation in the Office: What can it do for you?", Desktop Computing, n. 18, pp. 50-58, March, 1983.

Sherrard, John C., Hayes, John R., "A Computer Aided Instruction Tutorial for the RAMTEK 9400 Color Graphics Display System at the Naval Postgraduate School Monterey, California", Naval Postgraduate School Monterey, Ca., December 1981.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Prof. N. Schneidewind Code 54Ss Naval Postgraduate School Monterey, California 93940	4
4.	Prof. C. R. Jones Code 54Js Naval Postgraduate School Monterey, California 93940	1
5.	Prof. R. S. Elster Code 54Ea Naval Postgraduate School Monterey, California 93940	1
6.	Asst. Prof. D. R. Dolk Code 54Dk Naval Postgraduate School Monterey, California 93940	1
7.	Computer Technology Curriculum Office Code 37 Naval Postgraduate School Monterey, California 93940	1
8.	CDR Jesse M. Richards, III 4132 Minton Drive Fairfax, Virginia 22032	1
9.	LT Glen F. Tilley 620 Thomas McKeen Street Orange Park, Florida 32073	1
10.	LT Kenneth J. Mills 56 White Pine Court California, Maryland 20619	1

201620

Thesis
M5957 Mills
c.1
Development of the computer systems management instructional laboratory at the Naval Postgraduate School.