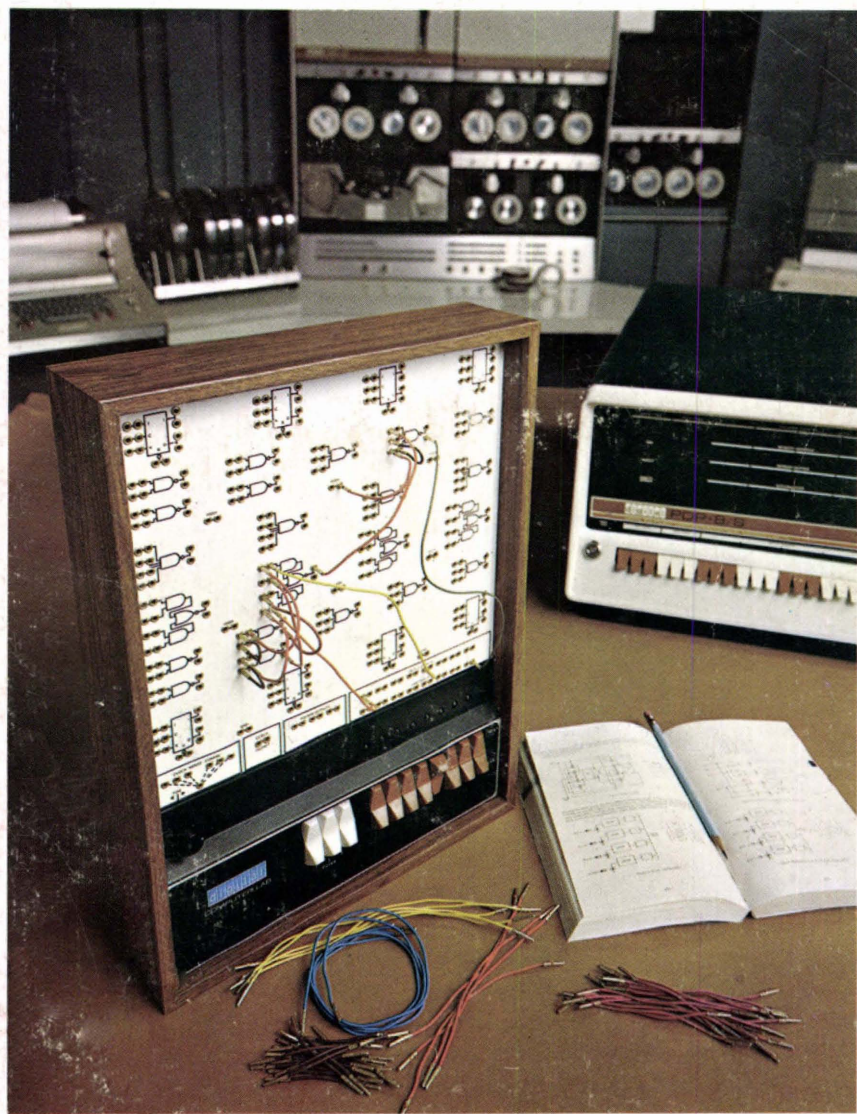


digital

COMPUTER LAB WORKBOOK



DIGITAL EQUIPMENT CORPORATION

digital

COMPUTER LAB WORKBOOK

**JOHN L. HUGHES
DESIGN ENGINEER**

**Copyright © 1969 by
Digital Equipment Corporation**

TABLE OF CONTENTS

PREFACE		V
INTRODUCTION		VII
OPERATING INSTRUCTIONS		XI
CHAPTER 1	The Binary Concept	1
	Experiment 1.1: 2-Input NAND Gate	6
	Experiment 1.2: 3-Input NAND Gate	6
	Experiment 1.3: 4-Input NAND Gate	9
	Experiment 1.4: Inverter	9
	Experiment 1.5: Substituting Gates	12
	Experiment 1.6: Decimal to Binary Encoder	15
	Experiment 1.7: Binary to Decimal Decoder	18
CHAPTER 2	Basic Logic Gates	23
	Experiment 2.1: AND/NOR Gate	25
	Experiment 2.2: NOR Gate Application of AND/NOR Gate	25
	Experiment 2.3: AND/NOR Comparator	27
	Experiment 2.4: AND/NOR Gate Used for Exclusive OR	27
	Experiment 2.5: Non-Inverting Gate	28
	Experiment 2.6: Equality Detector	29
	Experiment 2.7: Parity Bit Generator	31
CHAPTER 3	Flip-Flops	33
	Experiment 3.1: R-S Flip-Flop	34
	Experiment 3.2: Clocked R-S Flip-Flop	36
	Experiment 3.3: D Type Flip-Flop	37
	Experiment 3.4: J-K Flip-Flop	39
	Experiment 3.5: 4-Bit Shift Register	41
CHAPTER 4	Boolean Algebra to Gating Networks	45
	Experiment 4.1: Gating Circuit Simplification	59
	Experiment 4.2: Equality and Relative Magnitude Detector	64
CHAPTER 5	Binary Counters	65
	Experiment 5.1: Asynchronous Binary Up Counter	67
	Experiment 5.2: Modified Asynchronous Binary Counter	67
	Experiment 5.3: Synchronous Binary Up Counter	68
	Experiment 5.4: Synchronous Binary Up/Down Counter	70
	Experiment 5.5: Synchronous Modulo 6 Binary Counter	73
	Experiment 5.6: Asynchronous Self-Stopping Modulo 13 Binary Counter	73
	Experiment 5.7: Variable Modulus Asynchronous Binary Up Counter	74
CHAPTER 6	Serial Addition	79
	Experiment 6.1: Serial Adder	83
	Experiment 6.2: Subtraction Using the Serial Adder ..	84
	Experiment 6.3: Multiplication Using the Serial Adder	85

CHAPTER 7	Parallel Addition	87
	Experiment 7.1: Parallel Adder	90
	Experiment 7.2: Two's Complement Subtraction	92
	Experiment 7.3: One's Complement Subtraction	93
	Experiment 7.4: One Step Parallel Addition	93
CHAPTER 8	Binary Coded Decimal Operations	95
	Experiment 8.1: The 8421 Counter	95
	Experiment 8.2: The Excess 3 Counter	96
	Experiment 8.3: The 2421 Counter	96
	Experiment 8.4: The 5421 Counter	96
	Experiment 8.5: Serial BCD Addition	101
CHAPTER 9	Code Conversion and Decoding	105
	Experiment 9.1: The 2421 to 8421 Converter	110
	Experiment 9.2: The 5421 to 8421 Converter	113
	Experiment 9.3: Gray to Binary Converter	115
CHAPTER 10	System Considerations	119
	Experiment 10.1: Parallel Adder Control	122
	Experiment 10.2: Synchronizers	123
	Experiment 10.3: Maximum Frequency	126
Appendix A:	Checkout Procedures	129
Appendix B:	Karnaugh Mapping	135
Appendix C:	TTL Circuitry	139
Appendix D:	The Computer	145
Appendix E:	Glossary of Terms	156
Appendix F:	Decimal to Binary Conversion Table	163
Appendix G:	Powers of Two	165
Appendix H:	Recommended Texts	166
Appendix I:	COMPUTER LAB Hardware Specifications	167
Appendix J:	Warranty	169
Appendix K:	Logic Element Truth Tables.	170

PREFACE

This workbook contains a complete course in digital logic. It is intended to accompany courses in binary arithmetic, Boolean algebra, digital logic or computer technology. The workbook was developed to be used with Digital Equipment Corporation's COMPUTER LAB.

Much more material is presented than could be used in any full year laboratory course. The extra material is included to allow particular sections of a course to be emphasized with larger amounts of experimental time. Most chapters have a main theme experiment followed by a number of sub-experiments that can be used for emphasis. The information presented in the sub-experiments is not needed to understand later sections of the book and these sections can be considered supplementary.

The workbook material is prepared on five levels of instruction. The first level is an introduction to the binary concept. The student is shown how two states can be used to both perform logic functions and count. Basic logic functions on the COMPUTER LAB are studied, and the correlation between the binary and decimal systems is demonstrated.

The second level provides a close look at the logic elements available on the COMPUTER LAB. The student uses the simpler logic gates to construct some of the more complex devices. This way, he gains a much greater understanding of these complex mechanisms than would be possible by studying a complete unit.

At the third level, the student studies Boolean algebra, logic truth tables and how the transition from a logic requirement to a gating network is accomplished. This level uses the complete complex logic elements studied in detail in the previous level.

The fourth level deals with digital subsystems, beginning with an explanation of the mathematical process of binary addition. This level also discusses methods of complementary binary subtraction, binary coded decimal counting, and code conversion.

The final level of learning brings together the knowledge from previous levels by discussing digital systems.

My thanks go to all the people who have assisted in the writing, preparation and revision of this book. In particular, I would like to acknowledge the help given by the following individuals from Digital Equipment Corporation: Mr. G. Del Rossi, Mr. D. Doyle, Mr. R. Nelson, Mr. J. Richardson; and also, Mr. J. Knott, Thistletown Collegiate Institute; Mr. J. Miller, Ryerson Polytechnical Institute; Dr. K. Smith, University of Toronto.

GENERAL FEATURES

With the COMPUTER LAB, DIGITAL offers a complete teaching package. The unit and workbook are supplemented by the COMPUTER LAB Teacher's Guide which includes detailed course plans, complete answers for all the questions in the workbook, additional pictorial wiring diagrams, suggested test questions and supplementary instructional text.

- Teaches modern computer logic.
- Comprehensive Workbook includes basic text material and ten detailed experiments.
- Provides up to 200 hours of laboratory study.
- The ten basic chapters are divided into more than 30 experiments.
- Teacher's Guide with answers, additional text, extra problems, and course plans.
- Easy to use: each logic function is represented on the front panel by MIL STD-806 symbology. Inputs and Outputs of each internal circuit are accessible through the front panel eyelets.

INTRODUCTION

Computers and computer technology are assuming roles of increasing importance in today's technically oriented civilization. It is becoming more and more necessary that secondary schools give students a basic knowledge of computer fundamentals. For all students, instruction in basic computer concepts helps present and reinforce the important "New Math" principles of binary arithmetic and Boolean algebra. For the technical student, a knowledge of computer fundamentals can open a rewarding career in the computer industry.

The explosive growth in the use and application of computers has created a tremendous, mostly unfilled, demand for trained people. By 1970, according to a recent survey, there will be between 100,000 and 150,000 computers in use. This means that in the next few years, the computer industry will require more than 150,000 new programmers, over 200,000 new systems analysts, more than 90,000 new managers and supervisors, plus about 20,000 people to replace those lost through attrition.

As the computer gradually eliminates jobs in some areas, others are created requiring new skills and placing new demands on our educational system. Educators in ever growing numbers agree that all students should receive at least a basic introduction to the fundamentals of computer technology. All students will benefit from exposure to fundamental computer concepts, for the mathematical principles that form an important part of computer technology are also important in general mathematics programs.

Education in basic computer concepts should begin in secondary schools and be complemented by specialized training in vocational schools and advanced college study. Many educators have already accepted this responsibility and instituted computer technology courses. Texts, curricula and other materials prepared by the United States Office of Education, the National Science Teacher Association, The Association for Computing Machines and others, stress that initial education in computer science should emphasize the basic concepts and principles that apply to all digital computers. Chief among these are:

- 1) Binary arithmetic—the "machine language" of most present day computers
- 2) Principles of digital logic—the electronic "switches" that allow computers to operate and make decisions
- 3) Two-state memory devices
- 4) Boolean algebra—the rules for logic manipulation

In teaching computer technology and its associated mathematical concepts, the student benefits most if he learns the basic principles first, if he becomes thoroughly familiar with binary arithmetic, Boolean algebra and digital logic. From this foundation, all other computer-related education builds naturally and easily, allowing the student to continue his education with the least effort and best preparation into any of the many careers created by the expanding computer technology, from computer mathematics to programming to electronic engineering.

THE COMPUTER LAB

Digital Equipment Corporation has had extensive experience in computer education, with over half of its 3000 installed PDP computers in educational

institutions. DIGITAL found a need for a training device to teach basic computer concepts and so developed the COMPUTER LAB, a classroom laboratory for teaching digital logic and computer fundamentals.

The COMPUTER LAB is a complete educational package for teaching the fundamental concepts underlying the theory and operation of all digital computers. In clear experiment-lessons, the student is given step-by-step instructions in digital logic principles. After mastering the basic material presented in the COMPUTER LAB course, the student can progress easily through advanced study into many of the computer-related disciplines. The digital logic fundamentals presented in the course constitute the basic knowledge a student needs to build a career as a computer technician or electronics engineer. A programmer, if he is to use a computer to its fullest potential, must also be thoroughly familiar with a computer's basic operating principles.

An important part of computer technology is the "New Math". Once, the binary number system and Boolean algebra were interesting sideroads of mathematics. Now the computer has given them new importance, for the computer understands only the binary language and follows the laws of Boolean algebra. Everyone in school today is exposed to these two vital elements of the New Math. The COMPUTER LAB not only operates with binary numbers according to Boolean algebraic laws but graphically demonstrates these two important concepts. For every student, whether he intends to find a career within the computer industry or not, the New Math is important and the COMPUTER LAB makes it easier to understand. Students going on to college will need to use computer concepts in mathematics, science and engineering studies. For the technical or vocational student, a knowledge of the computer can lead to a career in the computer industry.

STEP-BY-STEP INSTRUCTION

The COMPUTER LAB incorporates the latest advances in computer technology and the workbook gives the student step-by-step instructions. Each instruction can be followed by wiring the unit with the easily inserted and removed patch cords and by testing final designs. The course provides a basic set of ten chapters, each divided into many complete experiments. Each chapter deals with a basic computer principle and the entire course fully illustrates the whole range of digital logic principles.

The unit is completely self-contained, compact, portable, and easy to use. Each of the internal basic digital functions is represented on the front panel by a logic symbol in standard, widely-accepted symbology. All connections for each experiment are made on the front panel with the patchcords, and the student can progress from one experiment to the next quickly and easily. Once the student has wired a logic design, he can test it by supplying signals in either of two ways. A series of three manually-operated switches provides signals to the logic. The unit also has a clock that automatically supplies test signals at any point in the system. Eight additional switches provide sustained signals to the logic and indicator lamps give a visual indication of the state of the logic at any point on the board. The unit comes complete with patchcords and workbook and is ready for immediate use. Because of the wide variety of logic elements used in the COMPUTER LAB, the student has complete freedom to design his own experiments and his own digital subsystems and range far beyond the basic instructions given in the COMPUTER LAB course.

THE COMPUTER LAB COURSE

As a laboratory instrument for classroom training in digital logic, the unit can provide a course lasting a full semester, comprising over 200 hours of laboratory work supplemented by 50 hours or more of lecture.

The basic course is organized on five levels of instruction. The instructor can follow the course material as presented in the workbook or is free to tailor the course to emphasize specific subject matter, grade levels or rates of instruction by deleting certain sections and/or adding supplementary information in other areas.

The COMPUTER LAB is designed for classroom use in high schools, technical schools, junior colleges and universities. It can be used for logic circuit experimental work in research laboratories and in industry. During laboratory work, at every level of instruction, the student has the opportunity to learn by doing. At any level, he may easily move back to a lower level to reinforce some of the more basic concepts taught earlier. As the student becomes more adept, he may connect several COMPUTER LABs together, one supervising the other, to enlarge computational power.

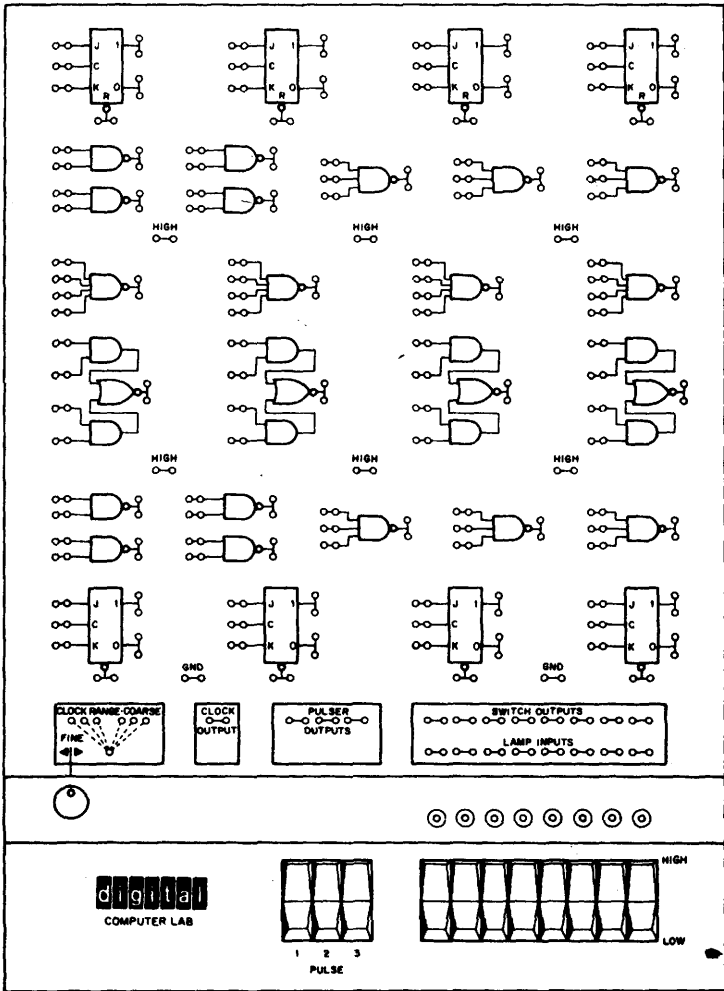


Figure 1 COMPUTER LAB Patchpanel

COMPUTER LAB OPERATING INSTRUCTIONS

I LOGIC LEVELS

There are two logic levels used on the COMPUTER LAB; HIGH (HI) and LOW (LO). Several terminals are provided on the patchpanel to give a HIGH logic level.

II ROCKER SWITCHES

Rocker switches can be used to provide either a HI or a LO logic level. If the upper side of the rocker switch is depressed, the two corresponding switch output terminals (directly in front of the switch) are taken to a HI level. If the lower side of a rocker switch is depressed, the two corresponding output terminals are taken to a LO level.

III PULSER SWITCHES

The outputs of the pulser switches are normally LO. When a pulser is depressed, the corresponding two output terminals go to a HI level. When the pulser is released its output terminals return to the LO condition. Internal circuits connected to the pulsers make sure that when a pulser is depressed or released, electrical noise generated in the switch is not transmitted to the pulser output. This special circuitry makes the pulsers useful in applications requiring noise-free transitions from one level to another. Rocker switches do not have this feature.

IV PULSES

Pulses are voltages which go from one logic level to another for a short time and then return to the original level. All pulses have a width which is defined as the length of time for which the pulse voltage is at the second or transient level. HI PULSES are ones which go from LO to HI for a short time, then back to LO. LO PULSES are ones which go from HI to LO for a short time, then back to HI. Pulses can occur one at a time or in a pulse train. Pulse trains have pulses which occur at a certain repetition rate, normally measured in pulses per second. (See Figure 2.)

V CLOCK

The clock provides a continuous train of HI pulses. Clock pulses are 50 nanoseconds wide (50×10^{-9} seconds or $\frac{50}{1,000,000,000}$ sec.). The frequency of clock pulses can be continuously varied from less than one pulse per second to over 10 million pulses per second. The slowest speed range of the clock is obtained by connecting the common clock coarse terminal to the left-most speed-selecting terminal. (See Figure 3.) The repetition rate of the clock increases with each terminal to the right. The fastest repetition rate is obtained by leaving the clock range selector disconnected. Repetition rates within each coarse range can be varied using the clock fine control. (Fully counterclockwise gives the slowest repetition rate in a range; fully clockwise provides the fastest rate.) The clock range coarse terminals are to be used only for selection of clock repetition rate. The clock output is obtained from the two terminals labeled CLOCK OUTPUT.

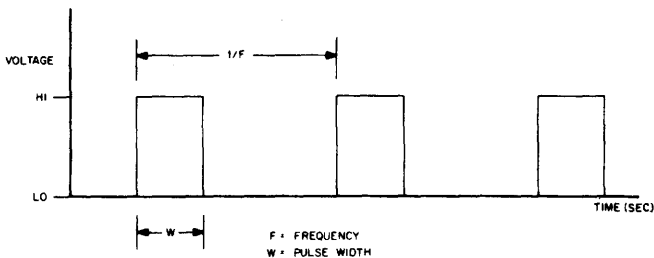


Figure 2 High Pulse Train

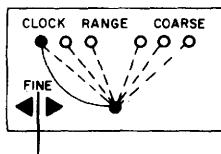


Figure 3 Clock Range Set Slow

VI LAMP INDICATORS

The operation of experiments constructed on the COMPUTER LAB patchpanel is monitored by the lamp indicators. A lamp will be ON if its corresponding input is at a HI logic level. A lamp will be OFF if the corresponding input is at a LO logic level. If no connection is provided to a lamp input, the lamp will be OFF. Lamps will respond to sustained logic levels and pulses of sufficient duration to activate the lamp filament.

VII UNUSED INPUTS

Unused inputs to gates and flip-flops should be connected to the HI terminals provided on the patchpanel. It is especially important to connect flip-flop Reset inputs to a HI terminal when they are used in counters, shift registers, etc., with no Reset provision. Unless this is done, flip-flops will not operate properly.

VIII USING TWO OR MORE COMPUTER LABS TOGETHER

Often there are applications where it is necessary to use two or more COMPUTER LABS to construct large logic circuits. To use COMPUTER LABS in this way, connect wires from the GND (ground) terminals on the patchboard of one COMPUTER LAB to GND terminals on the other units. When constructing circuits of this nature, it is best to try to build major circuit sections on each COMPUTER LAB. This sectional construction will keep the number of logic circuit interconnections between COMPUTER LABS to a minimum. Each COMPUTER LAB used in a large circuit must be plugged in and turned on.

IX WIRING ON COMPUTER LAB

Often a single output is used to drive several gate inputs. There are only two terminals for each output, but an output can be wired to drive many more than two inputs by "daisy chaining." This is done by connecting an output to the first input it has to drive, then connecting a wire from the first input to the second input, the second to the third, and so on. Figure 4 shows how all flip-flop clock inputs can be daisy chained and connected to the COMPUTER LAB clock.

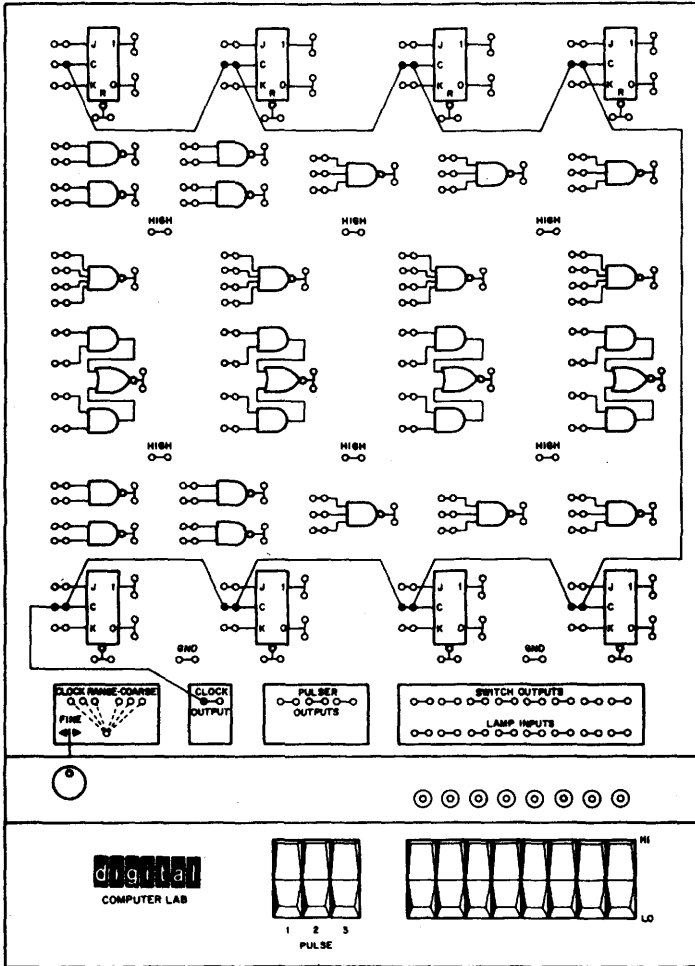


Figure 4 Daisy Chaining Clock Inputs

CHAPTER 1

THE BINARY CONCEPT

I INTRODUCTION

There are a large number of devices with only two states or possible conditions. For example, a light switch has only two states: it can be on or off. Similarly, a doorbell button can be either pressed or released, causing a bell to be on or off. The point of a ball point pen can be either extended or retracted. These two-state devices can be classed into two groups. Group 1 has memory, Group 2 does not have memory. For example, a light switch has memory; it remembers the last state it was put into. If it is turned on, it remains on until it is turned off. The ball point pen also has memory; if the point is extended it remains extended until the button at the end of the pen is pushed to retract it. Once the point is retracted, it remains retracted until it receives a command to extend. The door bell push-button is a device without memory; the bell will ring only as long as the button is depressed. Once the button is released it does not remember that it has been depressed and the bell stops ringing.

Decisions are often based on a number of YES or NO type conditions, two-state conditions. For example, if a driver sees a red light OR a stop sign OR an obstacle in the path of travel of his car, he stops. Symbolically the stop decision could be represented as in Figure 1.1. Each one of the conditions which would make the driver stop are either present or not present. They are two-state conditions: the red stop light can be on or off; the stop sign can be present or not present; the obstacle can be present or not present. If any one or more of these conditions is present, the driver will stop. All possible combinations of conditions which can occur can be represented in a table. Figure 1.2 is a table showing all the possible conditions that can occur between the stop light, stop sign and the obstacle. This type of table is known as a truth table. The number of possible conditions in a two-state truth table is equal to 2^N where N is the number of two-state variables considered.

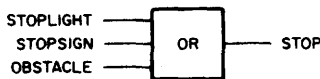


Figure 1.1 Stop Decision

CONDITIONS			RESPONSE (Stop)
Stop Light	Stop Sign	Obstacle	
No	No	No	No
No	No	Yes	Yes
No	Yes	No	Yes
No	Yes	Yes	Yes
Yes	No	No	Yes
Yes	No	Yes	Yes
Yes	Yes	No	Yes
Yes	Yes	Yes	Yes

Figure 1.2 Logical Stop Decision Truth Table

Mechanical situations can also be represented in a symbolic form. If, for instance, a door has two locks, the door would be secure if one lock or the other lock is bolted. Figure 1.3 shows how this condition can be represented symbolically. Again, all possible conditions that can exist with the locks can be represented in a truth table, as shown in Figure 1.4.

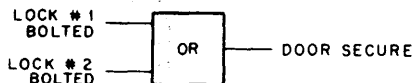


Figure 1.3 Door With Two Locks

LOCK #1 BOLTED	LOCK #2 BOLTED	DOOR SECURE
NO	NO	NO
NO	YES	YES
YES	NO	YES
YES	YES	YES

Figure 1.4 Door With Two Locks Truth Table

II GATES

The two above examples demonstrate logical decisions. The results of the first logical decision could be STOP or DO NOT STOP. The result of the second logical decision could be the door is SECURE or the door is NOT SECURE. The two-state outcome of these logical decisions depends on the state of the examined two-state input conditions. In the first example, the examined conditions were the stop light, the stop sign and the obstacle. In the second example, the examined conditions were the two locks on the door. Logical decisions can be made by human beings and by many types of devices, among them electrical devices, mechanical devices, hydraulic devices, etc. Regardless of the type of device, the decision can be represented symbolically in a standard form.

OR Gate

The OR logical decision has been demonstrated in both the above examples. The standard symbol for this logical decision is shown in Figure 1.5. This symbol is called an OR gate. The inputs or conditions examined by an OR gate are on the left of Figure 1.5. The result or output of an OR gate is on the right of Figure 1.5. OR gates may have two or more inputs but only one output. Figure 1.5 shows a two-input OR gate. The OR condition is met, or the gate is enabled, if one or more of the inputs is present. In general, a gate is enabled when its input conditions are met. The driving example described earlier can be represented in a standard symbolic form as shown in figure 1.6 by a 3-input OR gate. This gate will operate as described in the truth table in Figure 1.2.



Figure 1.5 Standard OR Symbol



Figure 1.6 OR Driving Decision

AND Gate

Another type of decision involves the AND function. For example, if a driver were at an intersection when the light turned green AND the path of travel were clear, the driver would go. Both input conditions must be present before the driver can go. Figure 1.7 shows a symbolic representation of this AND decision. Figure 1.8 is a truth table showing all the possible input conditions and the resultant output conditions. (Note: there are 2^2 possible combinations of the two two-state variables.) The AND gate can have two or more inputs, all of which must be present for the gate to be enabled.

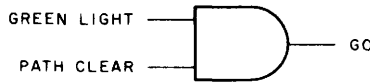


Figure 1.7 AND Driving Decision

Light Green	Path Clear	Go
No	No	No
No	Yes	No
Yes	No	No
Yes	Yes	Yes

Figure 1.8 AND Driving Decision Truth Table

Inverter

There are applications where it is convenient to take information and change it into its opposite state. A device to perform this function is known as an INVERTER. The symbols for an inverter are shown in Figure 1.9. If an inverter receives YES information at its input on the left, it would give NO information at its output on the right. If it received NO information at its input, it would give YES information at its output. Figure 1.10 is a set of truth tables which show the operation of the inverter. Figure 1.10(a) shows how the inverter can be used to invert YES or NO information.



Figure 1.9 Inverter

INPUT	OUTPUT	INPUT	OUTPUT	INPUT	OUTPUT
	NO	1	0	\bar{A}	$\overline{\bar{A}}$
	YES	0	1	\bar{A}	A
1.10(a)		1.10(b)		1.10(c)	

Figure 1.10 Inverter Truth Tables

Figure 1.10(b) shows how the inverter can be used to invert 1 or 0 information, a 1 representing a YES, a 0 representing a NO. Figure 1.10(c) shows a method of representing condition "A" and the inversion of that condition "not A." Not A, or the inversion of A, is represented by \bar{A} . If A is at the input of the inverter, its opposite, \bar{A} , is at the output. If \bar{A} is at the input of an inverter, its opposite, A, is at the output.

Summary

In an OR gate, the output is 1, or the gate is enabled, whenever one or more of the inputs are 1. The output of an OR gate is 1 if any one or more of the following conditions are met: input A is 1, OR input B is 1, OR input C is 1, etc.

In an AND gate, the output is 1, or the gate is enabled when, and only when, the inputs are 1, regardless of the number of inputs. The output of an AND gate is 1 only when input A is 1, AND input B is 1, AND input C is 1, etc.

In an Inverter, the output is always the opposite, or inverse, of the input. An output is 1 when its input is 0. An Inverter output is 0 when its input is 1.

Negated Input OR Gate

The inverter can be used with the AND and OR gates to extend their capabilities. If both inputs to an OR gate are fed through inverters, as in Figure 1.11, the OR gate functions in an opposite or complementary manner. With inverted inputs, the OR function will be present when one input, or the other, or both, are not present. In other words, the negated input OR gate will be enabled when one or both inputs are **not present**. Figure 1.12 shows the truth table for a 2-input negated input OR gate. A "1" represents a "present" or YES condition. A "0" represents a "not present" or NO condition. In standard symbology, a small circle on the input line of an OR gate symbol, as shown in Figure 1.13, is used to replace the inverter symbol used in Figure 1.11.

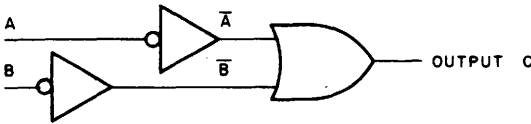


Figure 1.11 Negated Input OR Function

INITIAL CONDITIONS		INVERTED CONDITIONS		OUTPUT
A	B	\bar{A}	\bar{B}	C
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

Figure 1.12 Negated Input OR Truth Table



Figure 1.13 Negated Input OR

* See note on page 21.

NAND Gate

Similarly, an AND gate, with an inverter at its output, can be used to perform an opposite or complementary function, as shown in Figure 1.14. The truth table for this function in Figure 1.15 shows that the output from the inverted AND will not be present when both inputs are present. The inverted AND function is enabled when both inputs are present. This type of gate is represented in Figure 1.16 with a small circle attached to the output of the AND gate to replace the inverter symbol. Since the gate is performing a "not" AND function when enabled, it is called a NAND gate.

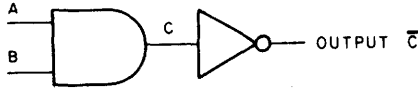


Figure 1.14 AND Gate with Inverted Output

A	B	C	OUTPUT \bar{C}
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Figure 1.15 AND Gate and Inverter Truth Table



Figure 1.16 NAND Gate

Equality of NAND and Negated Input OR Gates

If the truth tables for the negated input OR gate and the NAND gate are compared, as in Figure 1.17, it is readily evident that they are identical. For corresponding input conditions, the output conditions are equal for the NAND and negated input OR gates.

A	B	OUTPUT
0	0	1
0	1	1
1	0	1
1	1	0

Figure 1.17(a) Negated Input OR Gate Truth Table

A	B	OUTPUT
0	0	1
0	1	1
1	0	1
1	1	0

Figure 1.17(b) NAND Gate Truth Table

The operation of NAND and negated input OR gates is identical in every way, except in the way that the operation is interpreted. There are a number of NAND gates on the COMPUTER LAB. These gates can be interpreted as either NAND gates or negated input OR gates.

III COMPUTER LAB LOGIC FUNCTIONS

The COMPUTER LAB uses electronic logic functions which recognize two voltage levels, HI and LO. It is advisable at this time to review the COMPUTER LAB specifications for rocker switches and lamp indicators given in the introduction. The COMPUTER LAB uses inverting logic for a number of reasons. Chief among these is the versatility and ease of use. Also, all modern computers use inverting logic elements.

EXPERIMENT 1.1: 2-INPUT NAND GATE

Figure 1.18 shows a NAND gate with the inputs connected to rocker switches which can provide either a HI or a LO signal on either gate input. The output is connected to a lamp indicator. The same circuit is shown pictorially in Figure 1.19. Construct this circuit on the COMPUTER LAB. The rocker switch outputs are also connected to indicator lamps to give a visual indication of the input condition.

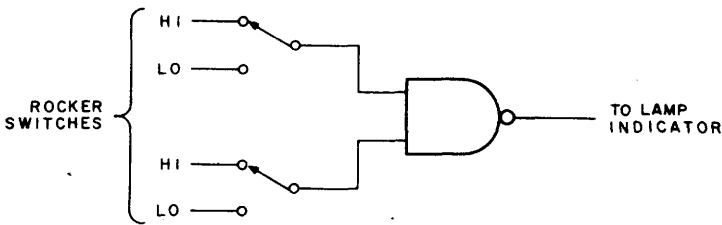


Figure 1.18 2-Input NAND Gate Test Circuit

Figure 1.20 is a truth table showing the operation of a 2-input NAND gate. Test the NAND gate by checking each condition of the truth table and filling in the final column to indicate whether the lamp was on or off, with HI representing the lamp on, and LO representing the lamp off. Compare the truth table you have made to the one in Figure 1.15.

SWITCHES		LAMP INDICATOR
A	B	
LO	LO	
LO	HI	
HI	LO	HI
HI	HI	

Figure 1.20 NAND Gate Truth Table

EXPERIMENT 1.2: 3-INPUT NAND GATE

Connect the 3-input NAND gate, as shown pictorially in Figure 1.21. Test its operation by observing the output condition for all possible input conditions; the truth table is shown in Figure 1.22. Complete the final column giving the indication obtained on the lamps. A HI will be indicated when the lamp is on; a LO, when the lamp is off. Again, the rocker switch outputs have been connected to indicator lamps to indicate visually the input conditions.

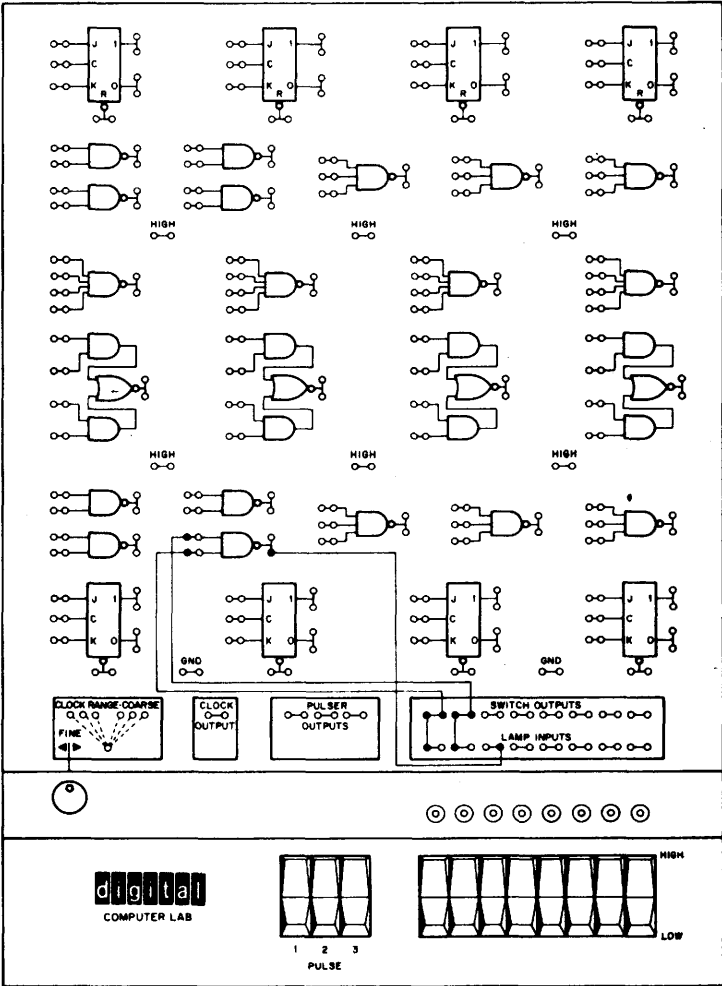


Figure 1.19 2-Input NAND Gate Test

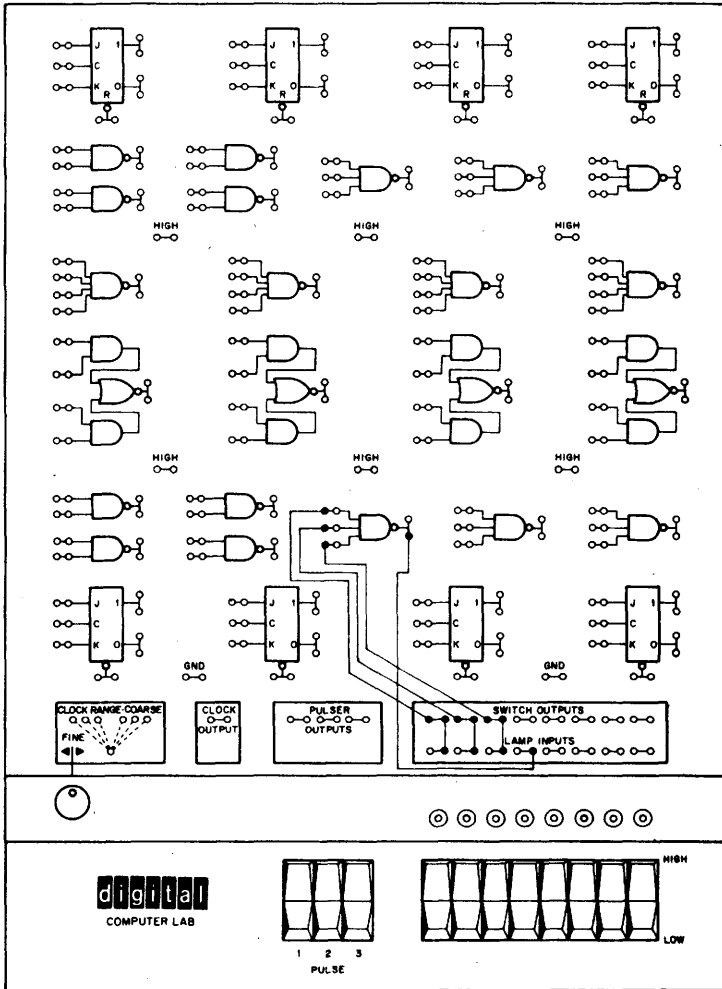


Figure 1.21 3-Input Gate Test

A	B	C	OUTPUT
LO	LO	LO	HI
LO	LO	HI	HI
LO	HI	LO	
LO	HI	HI	
HI	LO	LO	
HI	LO	HI	
HI	HI	LO	
HI	HI	HI	

Figure 1.22 3-input NAND Gate Truth Table

EXPERIMENT 1.3: 4-INPUT NAND GATE

Test the operation of the 4-input NAND gate on the COMPUTER LAB by connecting the circuit shown in Figure 1.23. Complete the final column in the truth table in Figure 1.24, with a HI indicating the lamp on, and a LO indicating the lamp off.

A	B	C	D	OUTPUT
LO	LO	LO	LO	HI
LO	LO	LO	HI	HI
LO	LO	HI	LO	HI
LO	LO	HI	HI	
LO	HI	LO	LO	
LO	HI	LO	HI	
LO	HI	HI	LO	
LO	HI	HI	HI	
HI	LO	LO	LO	
HI	LO	LO	HI	
HI	LO	HI	LO	
HI	LO	HI	HI	
HI	HI	LO	LO	
HI	HI	LO	HI	
HI	HI	HI	LO	
HI	HI	HI	HI	

Figure 1.24 4-Input NAND Gate Truth Table

EXPERIMENT 1.4: INVERTER

An inverter can be constructed as shown in figure 1.25.

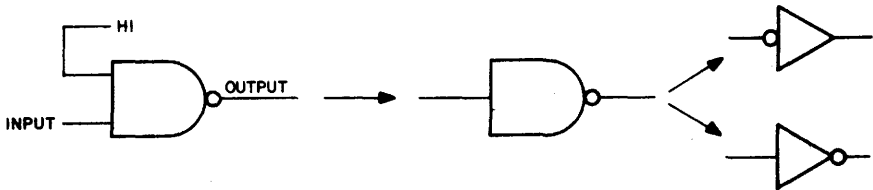


Figure 1.25 Inverting Function from 2-Input NAND

Make this inverter by connecting the circuit shown in Figure 1.26. Test its operation and complete the truth table shown in Figure 1.27. By supplying a

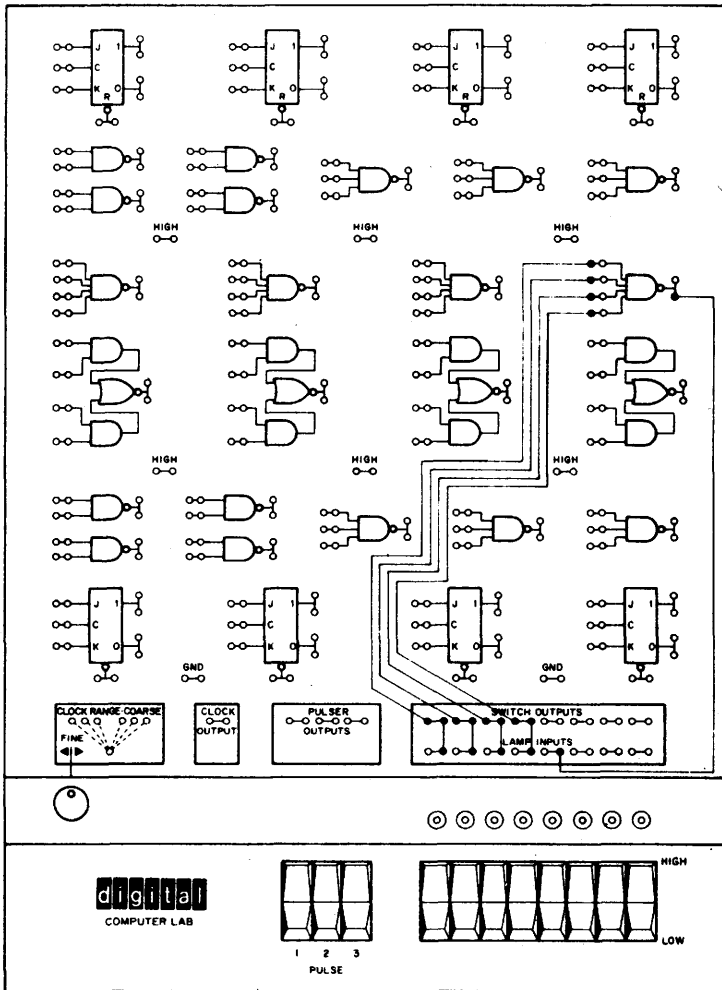


Figure 1.23 4-Input NAND Gate Test Circuit

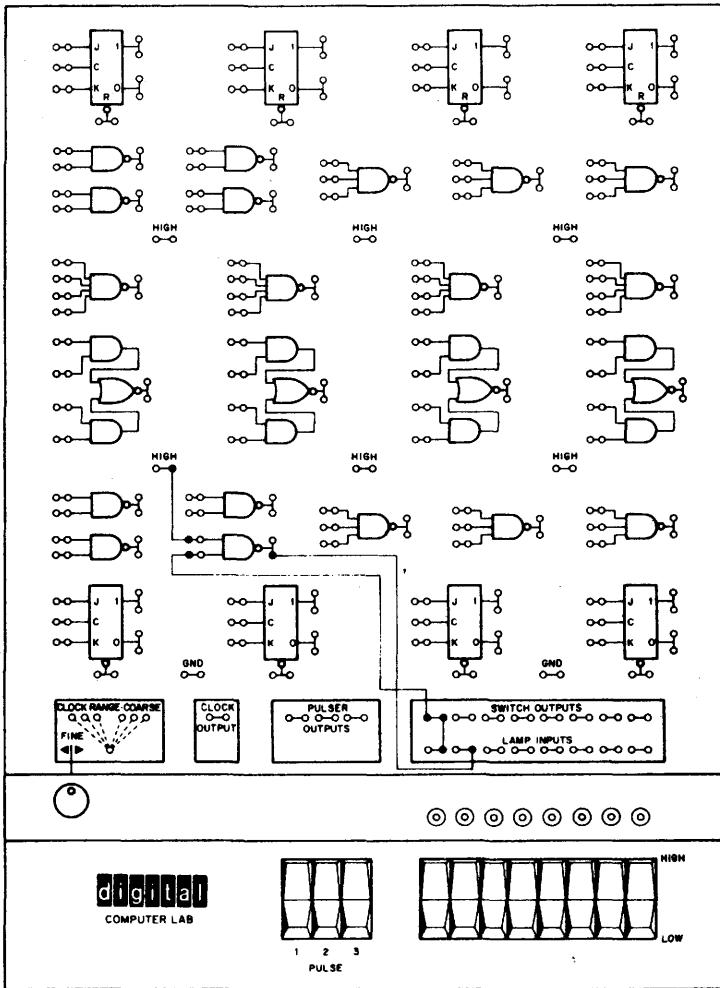


Figure 1.26 2-Input NAND Gate Used as an Inverter

constant HI to one input of a 2-input NAND gate, the gate becomes, in effect, a 1-input NAND gate. The condition of the one remaining input now controls the gate output. If the remaining input is HI, the gate output will be LO; if that input is LO, the output will be HI. The gate is now functioning as an inverter. The standard inverter symbols are shown on the right of Figure 1.25.

INPUT	OUTPUT
LO	HI
HI	LO

Figure 1.27 Inverter Truth Table

EXPERIMENT 1.5: MULTIPLE INPUT GATE USAGE

There are logic systems and subsystems to be constructed on the COMPUTER LAB which will require more NAND gates with a specific number of inputs than are available on the patch panel. In cases such as this, gates with a greater number of inputs can be substituted for gates with fewer inputs. For instance, a 4-input gate can be substituted for a 3-input gate by connecting the unused input to a HI level and using the other 3 inputs as a normal 3-input NAND gate. Connect the circuit shown pictorially in Figure 1.28 and complete the truth table for this 4-input NAND gate used to substitute for a 3-input NAND gate, as shown in Figure 1.29.

A	B	C	OUTPUT
LO	LO	LO	HI
LO	LO	HI	HI
LO	HI	LO	
LO	HI	HI	
HI	LO	LO	
HI	LO	HI	
HI	HI	LO	
HI	HI	HI	

Figure 1.29 Truth Table for 4-Input NAND Gate Used as 3-Input NAND Gate

QUESTIONS

1. Make a truth table similar to the one you have completed in Figure 1.20 for a two-input NAND gate when used as an inverter. Which conditions on the initial truth table cannot occur? Why?
2. When a 4-input NAND gate has one of its inputs connected to HI, what conditions cannot occur in the truth table in Figure 1.24?
3. Show how a 3-input NAND gate can be used as an inverter.
4. Show two techniques for using a 3-input NAND gate as a 2-input NAND gate.
5. Explain why a NAND gate can be used to perform the negated input OR function.

IV BINARY NUMBERS

The decimal system has 10 distinct states represented by the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. However, there are counting systems which use other than 10 distinct states. Consider, for example, an apartment building where there are 8 apartments on each of 7 floors. The owner wants to number each apart-

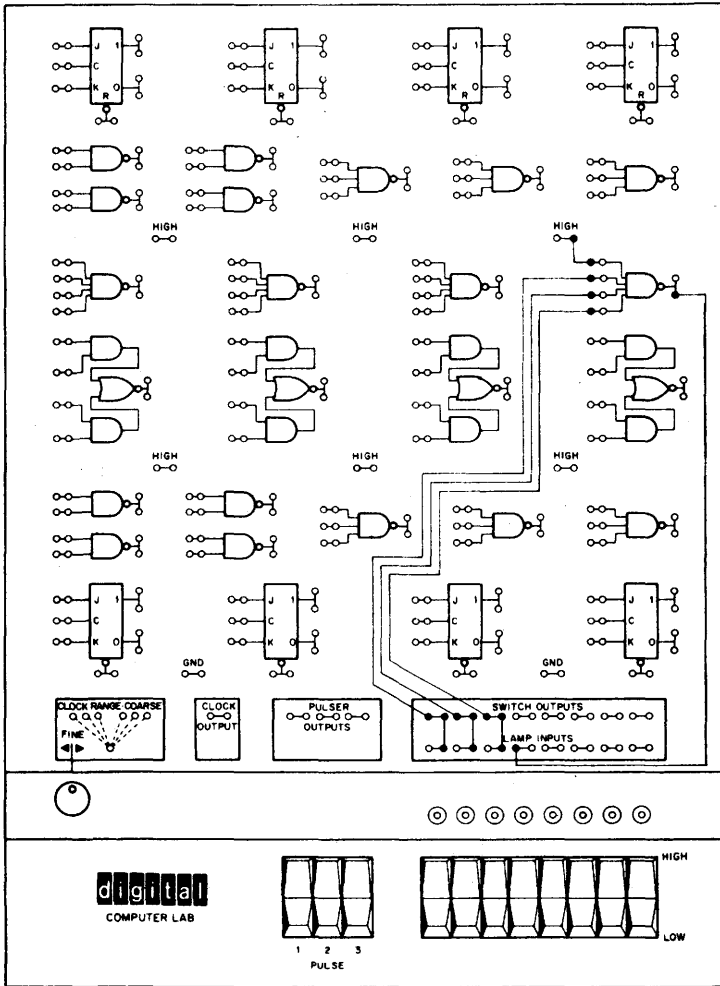


Figure 1.28 4-Input NAND Gate Used as a 3-Input NAND Gate

ment with the first digit of the number indicating the floor of that apartment and the second digit indicating the number of the apartment on that particular floor. The logical sequence of numbers would be 00, 01, 02, 03, 04, 05, 06, 07, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, . . . etc. This counting sequence has used only 8 unique states or digits. The digits used are 0, 1, 2, 3, 4, 5, 6, 7. The number system is called the octal number system, because it has a base of 8 distinct states.

BINARY							DECIMAL		
2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	10 ²	10 ¹	10 ⁰
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	1	0	0	0	2
0	0	0	0	0	1	1	0	0	3
0	0	0	0	1	0	0	0	0	4
0	0	0	0	1	0	1	0	0	5
0	0	0	0	1	1	0	0	0	6
0	0	0	0	1	1	1	0	0	7
0	0	0	1	0	0	0	0	0	8
0	0	0	1	0	0	1	0	0	9
0	0	0	1	0	1	0	0	1	0
0	0	0	1	0	1	1	0	1	1
0	0	0	1	1	0	0	0	1	2
0	0	0	1	1	0	1	0	1	3
0	0	0	1	1	1	0	0	1	4
0	0	0	1	1	1	1	0	1	5
0	0	1	0	0	0	0	0	1	6
0	0	1	0	0	0	1	0	1	7
0	0	1	0	0	1	0	0	1	8
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	0	0	1	0	0	1	0	0
1	1	0	0	1	0	1	1	0	1
1	1	0	0	1	1	0	1	0	2

Figure 1.30 Binary Counting Sequence

Similarly, if there are only two unique states to work with, a counting system can be evolved: this is the binary or two-state counting sequence. Figure 1.30 shows the binary count sequence and decimal numbers of corresponding value. Digital computers are composed of two-state logic elements which can use the binary system. Just as the decimal system is based on powers of 10, the binary system is based on powers of 2. Each binary digit carries a weight or multiplier which is a power of 2, as shown in figure 1.30. The least significant, or right-most digit, carries a weight or multiplier of 2⁰ or 1. The next most significant binary digit carries a weight or multiplier of 2¹. The next most significant binary digit carries a weight or multiplier of 2² and so on. A decimal number is weighted with the right-most column having a multiplier or weight of 10⁰. The next most significant column has a multiplier of 10¹ or 10. The next most significant column has a weight of 10² or 100, and so on. For instance, the decimal number 136 is equal to:

$$1 \times 10^2 + 3 \times 10^1 + 6 \times 10^0.$$

Likewise the binary number 1100 is:

$$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$

which is equivalent to 12 in the decimal number system. When referring to a digit in a binary number, the term "bit" (contracted from binary digit) is used.

V DECIMAL TO BINARY CONVERSION

To convert a binary number into decimal, add the decimal weights of each binary bit which is a 1. Thus the binary number 1001100 is equivalent in value to the decimal number $2^6 + 2^3 + 2^2 = 76$. This binary to decimal conversion suggests a method for converting decimal numbers into binary numbers.

That method is to subtract the largest power of 2 which is less than or equal to the decimal number being converted. The second step is the same as the first and it is performed on the remainder from the first step. The process continues until the remainder of successive subtractions equals 0. Example, convert 76 into binary.

$$\begin{aligned} 76 - 2^6 &= 12 \\ 12 - 2^3 &= 4 \\ 4 - 2^2 &= 0 \end{aligned}$$

Thus 76 equals $2^6 + 2^3 + 2^2$. As was shown in Figure 1.30, the 2^2 is the third position from the right of a binary number; 2^3 is the fourth position from the right of a binary number; 2^6 is the seventh position from the right in a binary number. The binary equivalent of a decimal number 76 is, therefore, 1001100.

QUESTIONS

6. Convert the following binary numbers into decimal:

- | | |
|-------------|--------------|
| A) 10010110 | F) 100011110 |
| B) 0110011 | G) 10011001 |
| C) 11110111 | H) 10101010 |
| D) 1000001 | I) 110110110 |
| E) 0001000 | J) 011011011 |

7. Convert the following decimal numbers into binary:

- | | |
|--------|--------------------|
| A) 148 | F) 4×10^3 |
| B) 277 | G) 49 |
| C) 53 | H) 875 |
| D) 256 | I) 94 |
| E) 512 | J) 117 |

EXPERIMENT 1.6: DECIMAL TO BINARY ENCODER

To communicate with a computer, it is necessary to convert input information into the binary language which the computer understands. One device that does this translation is an encoder. Figure 1.31 shows an encoder for converting the decimal digits 0 to 7 into a three-bit binary number. Construct the circuit on the COMPUTER LAB according to the pictorial diagram in Figure 1.32. Recall that the NAND and the negated input OR gates are the same in function so the NAND gates on the COMPUTER LAB can be used wherever the schematic in Figure 1.31 requires the negated input OR function.

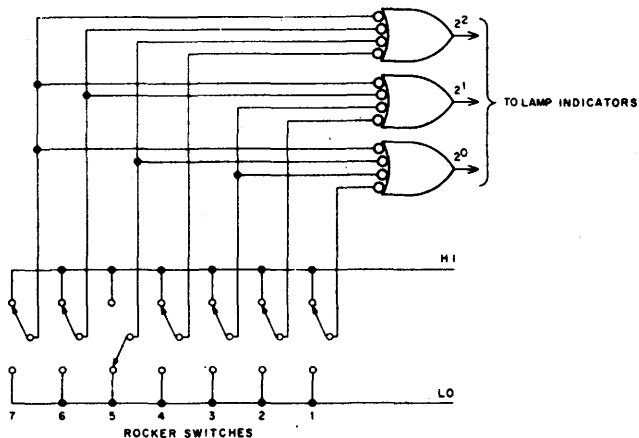


Figure 1.31 Decimal to Binary Encoder

Decimal To Binary Encoder Operating Instructions

1. Set all the rocker switches to the HI condition. Observe and record the condition of the lamps.
2. Set the '1' rocker switch to the LO condition. Observe and record the condition of the lamps.
3. Reset the '1' rocker switch to the HI condition. Set the '2' rocker switch to the LO condition. Observe and record the condition of the lamps.
4. Continue setting one rocker switch at a time to the LO condition and observe and record the indication obtained on the lamps for each of the 7 rocker switches.

QUESTIONS

8. Compare the recorded indication of the lamps which you have obtained to the binary count sequence in Figure 1.30.
9. At what level (HI or LO) should a switch input be for conversion? Why?
10. At what level must all switch inputs other than the ones being converted be? Why?
11. Explain the operation of the decimal to binary encoder by explaining the operation of each gate.
12. Design an encoder to convert the decimal numbers from 0 to 15 into binary.

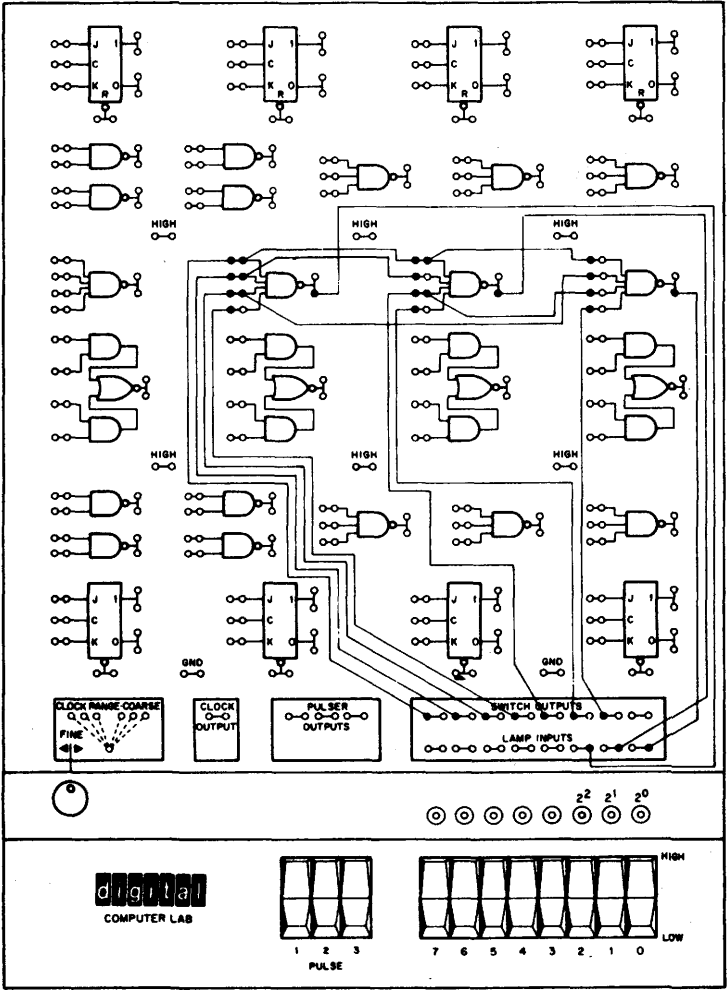


Figure 1.32 Decimal to Binary Encoder

EXPERIMENT 1.7: BINARY TO DECIMAL DECODER

When a computer has completed an operation the answer is usually given in binary form. Since most people are used to working in decimal, it is often necessary to decode the output information. One method is to use a NAND gate decoder, such as the one shown in Figure 1.33. The partial truth table in Figure 1.35 shows the operating rules for the decoder. Note that in the table the ($\bar{\quad}$) symbol is used to indicate negation.

For example, when $\bar{2}^0$ is a 1, it indicates that the 2^0 switch is in the LO position, or that the 2^0 bit is a 0 in the number being decoded. To detect the number 101, a NAND gate would look for the conditions 2^2 , $\bar{2}^1$ and 2^0 . When that NAND gate is enabled, it would indicate the presence of the number 101 by giving a LO output (which could turn a lamp out). Complete the truth table in Figure 1.35 to show the complete operating rules for the 3-bit binary to decimal decoder. Construct the decoder as shown in the pictorial diagram in Figure 1.34.

2^2	2^1	2^0	$\bar{2}^2$	$\bar{2}^1$	$\bar{2}^0$	DECIMAL
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	2
0	1	1				3
1						4
1						5
						6
						7

Figure 1.35 Binary to Decimal Decoder Operating Truth Table

Binary To Decimal Decoder Operating Instructions

1. Place all switches in the LO position. Observe and record lamp outputs.
2. Place 2^0 switch in the HI position. Observe and record lamp outputs.
3. Return 2^0 switch to the LO position. Place 2^1 switch to the HI position. Observe and record lamp outputs.
4. Continue testing the decoder by placing the switches in the positions indicated by the binary count sequence in Figure 1.30 with a HI representing a 1, and LO representing a 0.

QUESTIONS

13. Explain how the logic elements in the binary to decimal decoder operate.
14. Design and construct the circuitry necessary to decode the binary number 101 into the decimal number 5.
15. Design the circuitry to decode the binary numbers up to 1111 into decimal.
16. Why is a base 2 number system well suited for two-state devices such as those used on the COMPUTER LAB?

SUPPLEMENTARY QUESTIONS

17. There are 24 ways that two 2-state devices can be coded to represent the numbers 0 to 3. Complete the table in Figure 1.36 to show all of the possible ways.

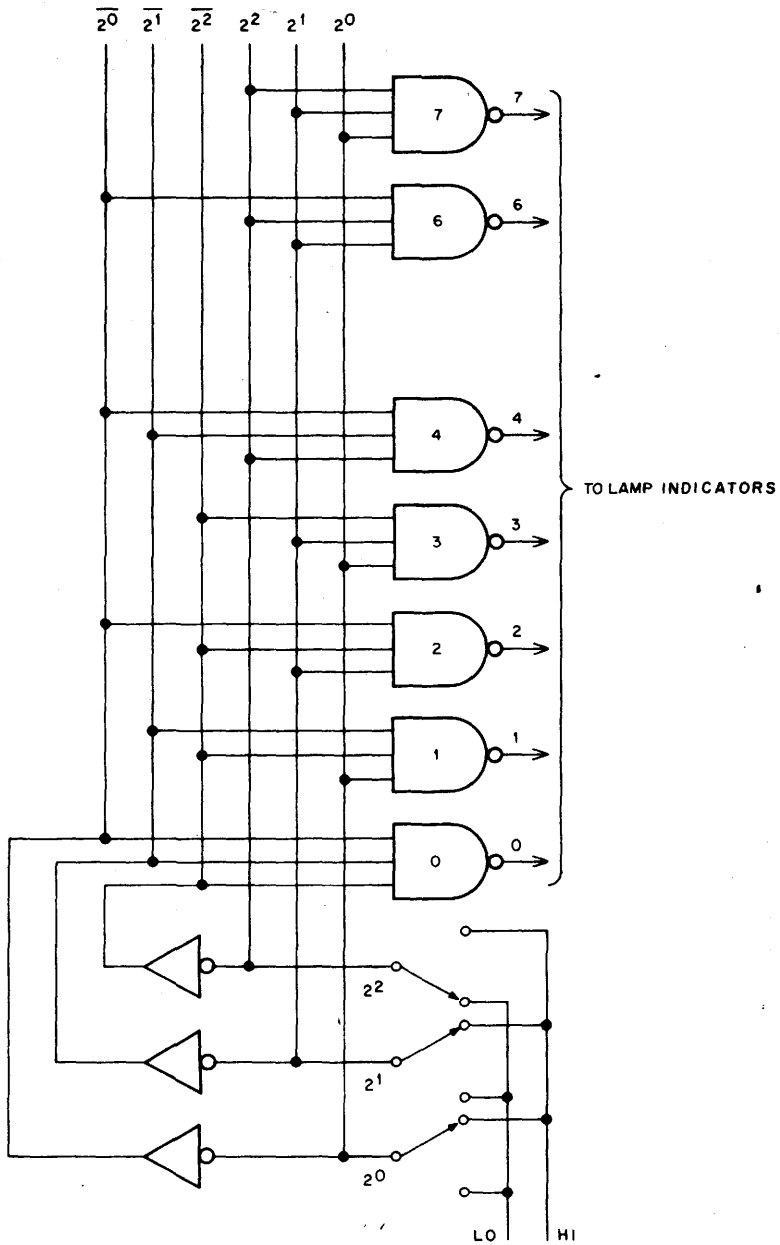


Figure 1.33 Binary to Decimal Decoder

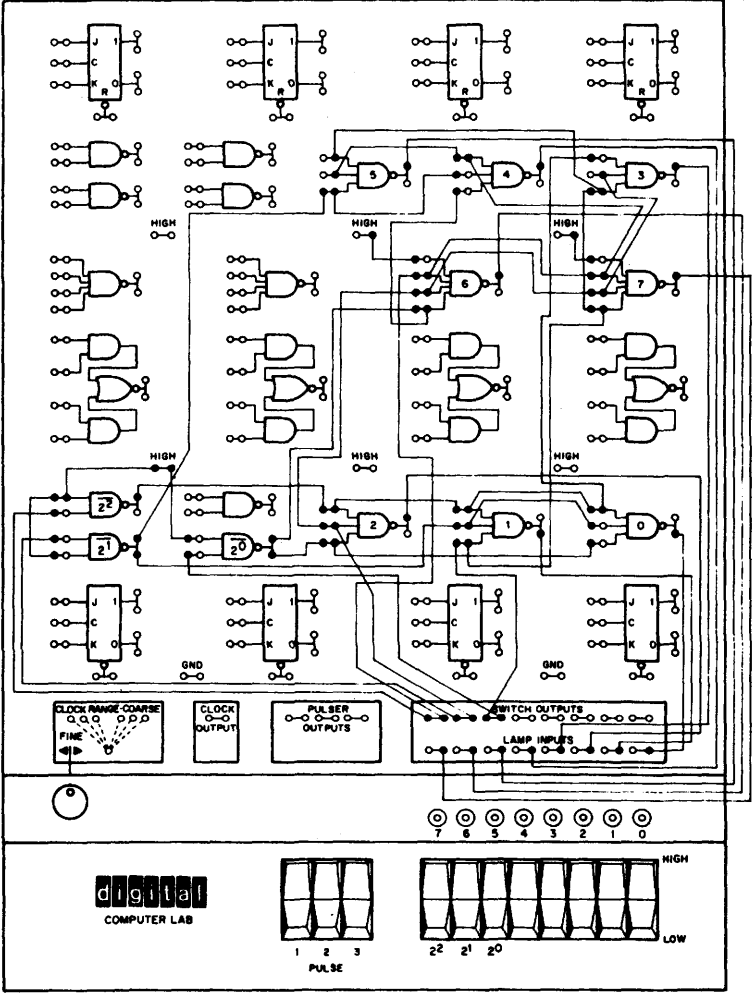


Figure 1.34 Binary to Decimal Decoder

DEVICE 1	DEVICE 2	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
LO	LO	0	0	0	0	0	0	1	1																
LO	HI	1	1	2	2	3	3	0	0																
HI	LO	2	3	1	3	1	2	2	-																
HI	HI	3	2	3	1	2	1	3	-																

Figure 1.36

Which of the 24 ways is easiest to follow and expand? Why?

Expand the code you have chosen to show the truth table for the complete count sequence available if four 2-state devices were used.

18. Construct a truth table for the numbers from 0 to 22 in the base 3 number system. (Use the digits 0, 1, 2.)
19. Design an encoder to encode the decimal numbers 0 to 8 into base 3 representation using the gates you have worked with.
20. What difficulties arise in using the COMPUTER LAB for number systems with a base other than 2?
21. The symbol for the NOR gate is shown in Figure 1.37. What alternative interpretation can be given to this gating function? Why?



Figure 1.37 NOR Gate

NOTE: A small circle(s) at the input(s) to any element indicates that the relatively low (L) input signal enables the function. Conversely, the absence of a small circle indicates that the relatively high (H) input signal enables the function.

A small circle at the symbol output indicates that the output terminal of the enabled function is relatively low (L).

CHAPTER 2

BASIC LOGIC GATES

I INTRODUCTION

The COMPUTER LAB uses electronic devices to perform the logic operations discussed in Chapter 1. When using these devices, their special properties must be considered. One of the things discussed in this experiment is the basic rules which must be followed when using the 2-state elements on the COMPUTER LAB.

The AND/NOR gate will be introduced. This is an extremely useful gate which can be used in circuits to determine the equality of two binary numbers. This experiment also covers simple applications of both the NAND and the AND/NOR gates to perform non-inverting logic functions.

II BASIC RULES FOR USING COMPUTER LAB FUNCTIONS

A Levels

A level is a voltage which is held constant for a long time. It may either be HI or LO. A HI level can be detected by a lamp; that is, a lamp will be lit if the input to the lamp is HI. If the input to a lamp is LO, it will not be lit. Sustained logic levels are provided by the rocker switches on the front of the COMPUTER LAB panel.

B Pulses

Pulses are voltages which go from one logic level to a second logic level for a short time and then return to the original level. Figure 2.1 shows a HI pulse: a voltage that goes from LO to HI, remains there for a short time, and then returns to LO. Figure 2.2 is a LO pulse: a level that goes from HI to LO, remains there for a short time and then returns to HI. HI pulses are provided by the pulser switches and the clock. A pulser switch will provide the HI level as long as the switch is depressed. The clock provides a continuous train of HI pulses of very short duration. The time between clock pulses can be varied from 1 second to less than $1/10,000,000$ of a second; that is, the clock can be slowed to provide 1 pulse per second or it can be speeded up to provide as many as 10,000,000 pulses per second.

Figure 2.1 High Pulse

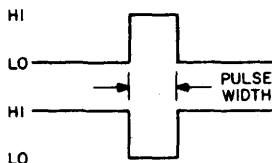


Figure 2.2. Low Pulse

C Input Loading

Each logic input on the COMPUTER LAB imposes a certain load on the output driving it. A quantitative measurement of the load imposed by an input on the output driving it is described as a number of load units. An output

must provide drive capability equal to or greater than the sum of the load units of all the inputs it is driving. All gate inputs on the COMPUTER LAB panel impose a load of one load unit. Clock and reset inputs on the flip-flops impose a load of two load units. The lamp inputs impose a load of 5 load units. If one output were driving 1 gate input, 1 flip-flop reset input, and one lamp input, that output would have to drive a total load of $1 + 2 + 5 = 8$ load units.

D Output Drive

The drive capabilities of the gates, flip-flops, clock and pulsers are limited: fan-out is a quantitative measure of the drive capability of a logic device. The fan-out of a logic element indicates the maximum sum of load units the element can drive. All the logic outputs on the patch panel of the COMPUTER LAB have a fan-out of 10. The clock, pulsers and rocker switches have a fan-out of 30. Therefore, a gate output is capable of driving any combination of inputs which imposes a load of 10 load units or less. For instance, one gate output can drive 10 gate inputs.

E Output Connections

Because of the electronic configuration of the logic elements in the COMPUTER LAB, **outputs from gates, flip-flops, pulsers, clock, switches, etc. must be connected only to inputs. Outputs must not be connected to outputs.** If outputs are connected together, an indeterminate condition will reside in the logic network and the results of an experiment would be inaccurate.

F The Clock

The clock provides a continuous train of HI pulses at its output. The repetition rate of these clock pulses is variable from 1 pulse per second up to 10,000,000 pulses per second. The clock has several available speed ranges. By connecting a wire from the common terminal to the left-most frequency selecting terminal, the clock operates at its lowest speed. By leaving the common terminal disconnected and the frequency terminals all disconnected, the clock operates at its highest speed. The clock fine control provides fine adjustment of speed within each coarse range.

G Ground Outputs

Ground outputs (labeled "gnd") are provided in the bottom corners of the COMPUTER LAB patch panel. They should be used only to connect one computer lab to another to perform complex experiments.

III THE AND/NOR GATE

The AND/NOR gate is a single logic element capable of performing compound logic functions. Figure 2.3 is a symbolic diagram of the gating functions performed by the AND/NOR gate. Since this is a single logic element it is impossible to use any of the individual functions within the AND/NOR gate separately.

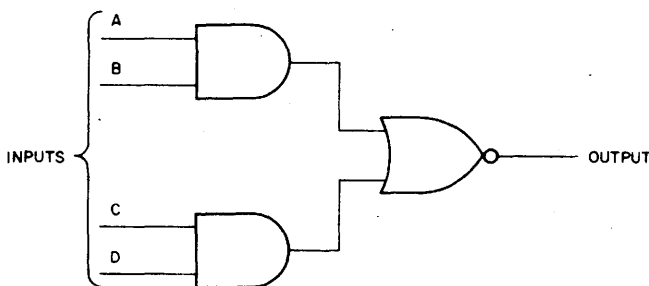


Figure 2.3 AND/NOR Gate

Only 4 inputs and 1 output are accessible. The truth table for the AND/NOR gate is shown in Figure 2.4. If either of the AND functions in the gate is enabled, by a HI at both its inputs, the output NOR gate will also be enabled and the output of the AND/NOR gate as a whole will be LO. If neither of the AND functions in the AND/NOR gate is enabled, the AND/NOR gate will be disabled and its output will be HI.

INPUTS				OUTPUT
A	B	C	D	
LO	LO	LO	LO	HI
LO	LO	LO	HI	HI
LO	LO	HI	LO	HI
LO	LO	HI	HI	LO
LO	HI	LO	LO	HI
LO	HI	LO	HI	HI
LO	HI	HI	LO	HI
LO	HI	HI	HI	LO
HI	LO	LO	LO	HI
HI	LO	LO	HI	HI
HI	LO	HI	LO	HI
HI	LO	HI	HI	LO
HI	HI	LO	LO	LO
HI	HI	LO	HI	LO
HI	HI	HI	LO	LO
HI	HI	HI	HI	LO

Figure 2.4 AND/NOR Gate Truth Table

EXPERIMENT 2.1: AND/NOR GATE

Test one of the AND/NOR gates on the COMPUTER LAB by connecting its four inputs to rocker switch outputs and its output to a lamp input. Verify that the gate operates as outlined in the truth table in Figure 2.4.

EXPERIMENT 2.2: NOR GATE APPLICATION OF AND/NOR GATE

Connect the 2 AND inputs of each half of the AND/NOR gate together as shown in Figure 2.5. Connect one rocker switch to each of the AND inputs of the AND/NOR gate. Leave the lamp driver connected to the output of the gate. Show the operation of the AND/NOR gate connected in this manner by completing the truth table in Figure 2.6.

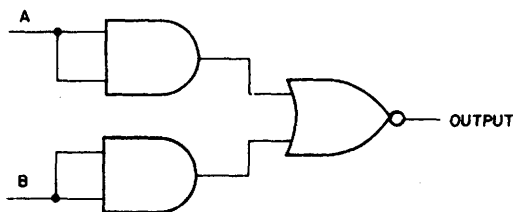


Figure 2.5 NOR Application of AND/NOR Gate

INPUTS		OUTPUT
A	B	
LO	LO	HI
LO	HI	LO
HI	LO	
HI	HI	

Figure 2.6 NOR TRUTH TABLE

The AND/NOR gate when connected as outlined above, is performing the NOR function shown symbolically in Figure 2.7. An alternate representation for the NOR function is a negated input AND, as shown in Figure 2.8.



Figure 2.7 NOR Gate

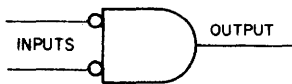


Figure 2.8 Negated Input AND Gate

QUESTIONS

1. Construct a truth table for the negated input AND gate, and compare it to the truth table you have made in Figure 2.6. Explain why the NOR gate and the negated input AND gate are different interpretations of the same logic function.
2. Make a truth table for the operation of the logic function shown in Figure 2.9.

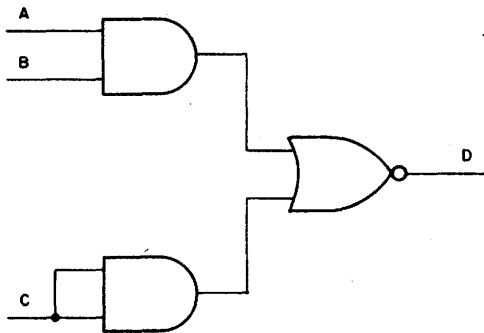


Figure 2.9

EXPERIMENT 2.3: AND/NOR COMPARATOR

The AND/NOR gate can also be used to compare two bits of binary information to determine whether or not they are equal. A simple way to perform this comparison using an AND/NOR gate and two inverters is shown in Figure 2.10. This circuit determines whether or not the two binary bits A and B are in the same state and are therefore equal. The truth table in Figure 2.11 shows that if both A and B are equal, the output is HI. If A and B are not equal, the output is LO. Construct the 2-bit comparator of Figure 2.10 on the COMPUTER LAB, connecting the inputs to rocker switches and the output to an indicator lamp. Test the comparator for each set of conditions in the truth table in Figure 2.11, and verify that it operates as outlined.

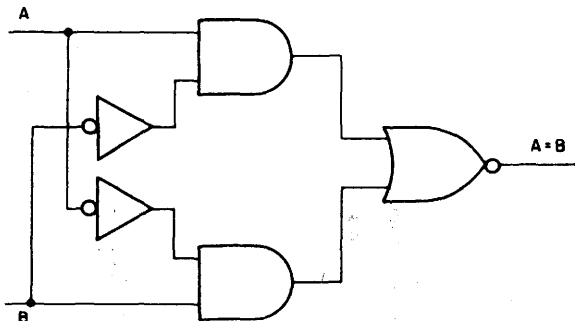


Figure 2.10 AND/NOR Comparator

A	B	A = B
LO	LO	HI
LO	HI	LO
HI	LO	LO
HI	HI	HI

Figure 2.11 Comparator Truth Table

QUESTIONS

3. Explain the operation of the gates in the comparator for each of the 4 input conditions.

EXPERIMENT 2.4: AND/NOR GATE USED FOR EXCLUSIVE OR

The normal 2-input OR function is present or enabled if one input or the other or both, are HI. Another type of OR function is the Exclusive OR. This function is enabled if one input or the other, but not both, is present. In other words, the output is HI if the inputs are different. The Exclusive OR function can be constructed using the AND/NOR gate with two inverters, as shown in Figure 2.12. Complete the truth table for the Exclusive OR function shown in Figure 2.13.

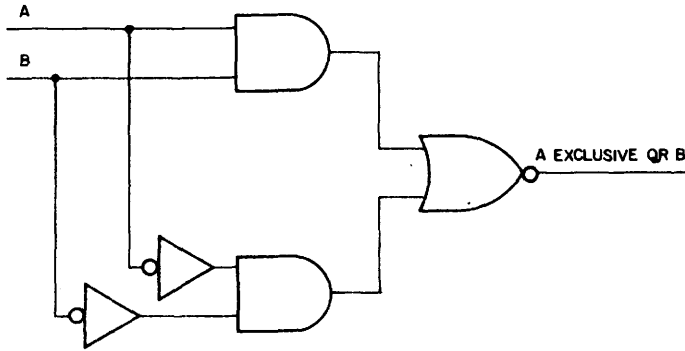


Figure 2.12 AND/NOR Exclusive OR

A	B	A Exclusive OR B
LO	LO	LO
LO	HI	HI
HI	LO	HI
HI	HI	LO

Figure 2.13 Exclusive OR Truth table

Construct the circuit shown in Figure 2.12 on the COMPUTER LAB. Connect inputs to rocker switches and connect the output to a lamp indicator. Test the circuit by verifying that the Exclusive OR circuit operates as outlined in the truth table in Figure 2.13.

QUESTIONS

4. What function would be obtained by putting an inverter on the output of the Exclusive OR circuit?

EXPERIMENT 2.5: NON-INVERTING GATES

Even though the COMPUTER LAB uses inverting gates, it is sometimes necessary to have non-inverting functions. Non-inverting functions can be constructed from inverting gates and separate inverters. To convert an inverting gate input to a non-inverting input, connect an inverter to the input. To convert an inverting output to a non-inverting output, connect an inverter to the output. For example in Figure 2.14a an AND function is constructed by inverting the output of a NAND gate. Construct this AND function on the COMPUTER LAB, connecting the two inputs to the rocker switches and the output to a lamp. Make a truth table to confirm that the function constructed is operating as an AND gate. Compare your truth table with the AND gate truth table in Figure 2.14b.

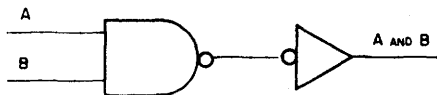


Figure 2.14a AND Function

A	B	OUTPUT A and B
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2.14b AND Gate Truth Table

QUESTIONS

- Construct an OR function using a NAND gate with inverters at the two inputs. Make a truth table for its operation.
- Make an AND function using two inverters and the AND/NOR gate in the NOR application.
- Make an OR function using an AND/NOR gate in the NOR configuration plus one inverter.
- What is the most convenient way to construct an AND function?
- What is the most convenient way to construct an OR function?

EXPERIMENT 2.6: THE EQUALITY DETECTOR

There are many applications in a computer where it is necessary to compare the value of two numbers and determine if they are equal. The comparator constructed previously in this experiment will indicate if two bits are equal. To determine if two complete binary numbers are equal, corresponding bits of the binary numbers can be compared using a comparator circuit. The outputs from a number of comparator circuits can be brought into an AND function to determine if the two numbers as a whole are equal. The circuit in Figure 2.15 is an equality detector composed of four comparator circuits which will determine if two 4-bit binary numbers are equal. If the two numbers are equal, there will be a HI or 1 at the output.

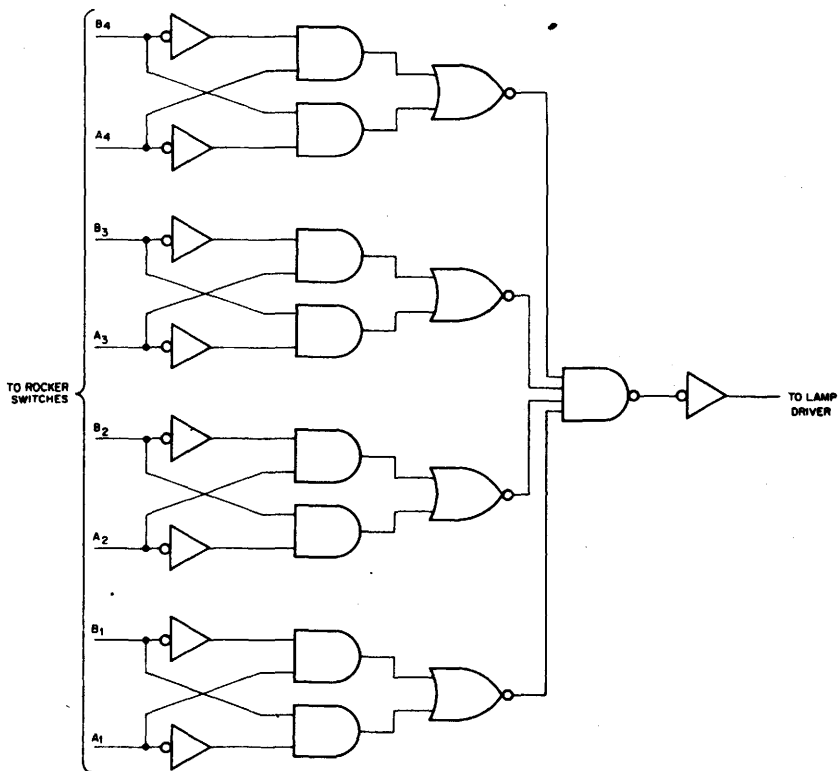


Figure 2.15 Four-Bit Equality Detector

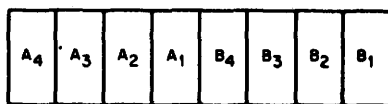


Figure 2.16 Switch Usage for Equality Detector

Construct the equality detector as shown in Figure 2.15. Connect the switch inputs shown in Figure 2.16 with number A on the left of the switch register and number B on the right of the switch register. Connect the output of the equality detector to a lamp indicator. Verify that the circuit does in fact operate as an equality detector by setting one number into the A switch register and trying the B register in all 16 possible states. Note the condition of the switches in the B register which causes the lamps to light. Repeat this procedure for two or three different settings of the number in switch register A.

QUESTIONS

10. Explain the operation of the gates in the equality detector in Figure 2.15.
11. Design an equality detector for two 8-bit switch registers using only AND/NOR gates, two-input NAND gates and four-input NAND gates.

EXPERIMENT 2.7: PARITY BIT GENERATOR

There are a number of ways to check that a binary number transmitted from one device to another is accurate. One way is to count the number of 1's in a binary number and, if there are an odd number of 1's, add another bit (called a parity bit), equal to one, to the end of the number. If the sum of the number of 1's in a binary number is even, a parity bit equal to 0 is added to the end of the number.

If a number and its parity bit are checked at any later time, there should always be an even number of 1's. The composite number is now said to have even parity. Data transmitted over a long distance often has a parity bit added so that the receiving computer or logic system can check the accuracy of a number before accepting it.

Suppose the information for preparing a pay check of \$128 was transmitted from one computer to another as the binary number 10000000. If the 2⁷ bit of information was for some reason changed, the binary number would become 00000000 and the check would be written for \$0. Had the system used parity, the number transmitted initially would have been 100000001 and the number received with the error would have been 000000001. When the receiving computer did a check for parity, the error would be discovered and the receiving computer would request the sending computer to re-transmit the information.

Complete the truth table in Figure 2.17 to show what parity bit should be added to the end of all 16 possibilities for 4 binary bits of information to maintain even parity. Construct the parity bit generator in Figure 2.18 connecting the inputs to rocker switches and the parity bit output to an indicator lamp. Connect the outputs of the four rocker switches to indicator lamps as well to give an indication of the binary number being read into the circuit. The parity bit generator uses successive Exclusive OR circuits to detect an uneven number of 1's. Test the circuit and verify that it operates as outlined for each condition of the truth table in Figure 2.17.

BINARY NUMBER				PARITY BIT
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0
1	1	1	1	1

Figure 2.17 Parity Bit Truth Table

QUESTIONS

12. Explain how successive Exclusive OR circuits can detect that an uneven number of 1's is present in the number and then generate a 1 parity bit.
13. Design a circuit for a 5-bit parity checker which could check the numbers generated by the parity generator in Figure 2.18.

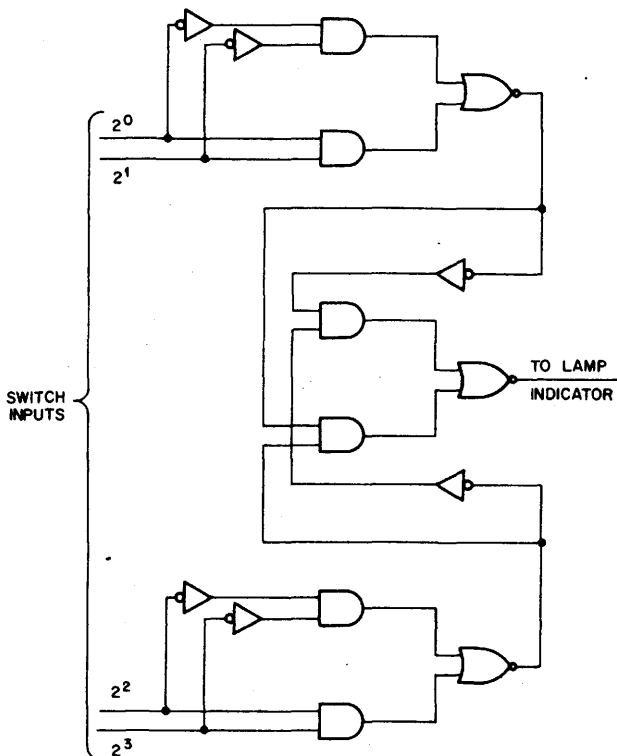


Figure 2.18 Parity Bit Generator

SUPPLEMENTARY QUESTIONS

14. Design and construct a circuit using an AND/NOR gate where the output will be LO if inputs A and B are HI, or input C is HI.
15. Design a parity bit generator for four information bits to generate a number and parity bit combination which will always give an odd number of 1's. (odd parity)
16. Design and construct a circuit using only NAND gates to perform the same function as the AND/NOR gate.
17. Will a parity bit checker be able to determine if two bits of one binary number have changed value during a transmission of information?

CHAPTER 3

FLIP-FLOPS

I INTRODUCTION

The logic devices studied to this point have all required a continuous input level to operate. Once the input level was removed, the device would not remember its previous input condition. In addition to gates, a computer requires devices that will retain their condition after an input has been removed. One example of a device without memory is a push-button. The push-button is energized only as long as the button is depressed. A light switch illustrates a device with memory. When a light switch is turned on, it will stay on until it receives an impulse to turn off. After it is turned off it will stay off until another "on" impulse is received. If there were no electrical devices with memory, a button would have to be held depressed to keep a light on.

In a computer, one device which has memory is called a flip-flop: an electronic logic element which remembers the last instruction it received. In many ways it is similar to a light switch. If a flip-flop receives an instruction to go to the "1" condition it does so and stays there until instructed to go to the "0" condition. If it is in the "1" condition and receives an instruction to go to the "0" condition, it does so and stays there until instructed to return to the "1" condition.

All flip-flops have a 1 and 0 output. If the 1 output is HI, it indicates that the flip-flop is in the "1" condition. If the 0 output is HI, the flip-flop is in the "0" condition. The 0 output always presents the opposite or complement of the information at the 1 output. There are therefore only two possible output conditions, either the 1 output HI and the 0 output LO or the 1 output LO and the 0 output HI.

Examining either output indicates the state of the other output and the condition of the flip-flop. Only the condition of one output must be known to determine the condition of the flip-flop; however, two outputs are provided for convenience. The truth table in Figure 3.1 shows the two possible output conditions for a flip-flop and the corresponding definition for the flip-flop state.

OUTPUTS		FLIP-FLOP STATE
I	O	
HI	LO	"1"
LO	HI	"0"

Figure 3.1

In this experiment several types of flip-flops will be constructed using only NAND gates. The internal logic operation of flip-flops can be more closely studied by building them from standard gating functions than would be possible using the complete flip-flops supplied on the COMPUTER LAB patch panel.

II THE R-S FLIP-FLOP

The Reset-Set(R-S) is the simplest to construct and easiest to understand flip-flop. It has two inputs, a SET input and a RESET input, and two outputs, a 1 and a 0. If a LO pulse is received on the SET input, the flip-flop goes to

the "1" condition and remains there after the pulse has been removed. If a LO pulse is received on the RESET input, the flip-flop goes to the "0" condition and remains there.

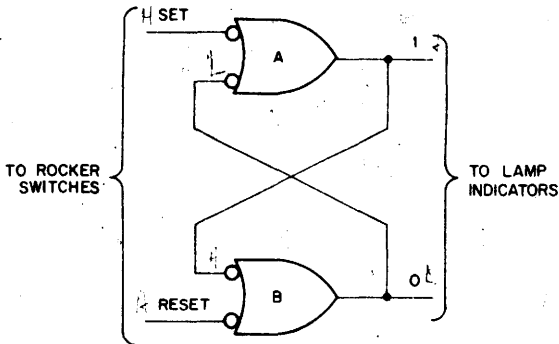


Figure 3.2 R-S Flip-Flop Using Negated Input OR (NAND) Gates

The circuit in Figure 3.2 is an R-S flip-flop composed of negated input OR (NAND) gates. Assume that both the Set and Reset inputs are HI and that the flip-flop is initially in the "1" condition with its 1 output HI and its 0 output LO. In this state gate A will have a HI output and will present a high level to the upper input of gate B. The RESET input and the SET inputs are both disabled when HI. Gate B will have two HI inputs present and its output will therefore be LO. The LO output of gate B will present a LO input to gate A which will enable gate A and produce a HI at A's output. Therefore, gate B is continually enabling gate A and gate A is continually disabling gate B. The two gates are "latched" in a condition that will remain until the input conditions are changed.

Similarly in the "0" condition, gate B is continually enabled by gate A and gate A is continually disabled by gate B. This again is a latched condition which will remain until the flip-flop receives another input.

If the flip-flop is initially in the "0" condition and a LO pulse is present on the SET input, gate A will be enabled and provide a HI at its output and a HI at the input to gate B. Gate B will be disabled because the RESET is HI and the input presented by gate A is HI. Gate B's output will therefore go LO. In the LO condition, it will enable the lower input of gate A and will establish a latched "1" condition. Similarly if the flip-flop is initially in the "1" condition, and a LO pulse is presented on the RESET line, gate B is enabled causing gate A to be disabled and creating a latched "0" condition.

If both inputs of an R-S flip-flop are simultaneously enabled with a LO pulse, both outputs will go HI for the duration of the enabling pulses. When both enabling pulses are removed, the flip-flop will go to an indeterminate state: it could latch in either the "1" or the "0" condition. Because of this indeterminate state, care must be taken when using an R-S flip-flop to insure that both inputs are not simultaneously enabled.

EXPERIMENT 3.1: R-S FLIP-FLOP

Figure 3.3. is a truth table showing the output conditions which will result when the combinations of initial conditions and signal inputs occur as shown

in columns one through four. Line one and line five will result in the indeterminate conditions previously described. Construct the R-S flip-flop as shown in Figure 3.2; connect the 1 and 0 outputs to separate lamp indicators. Connect SET and RESET inputs to rocker switches. Test the circuit you have constructed to verify each line in the truth table in Figure 3.3.

		(1)	(2)	(3)	(4)	(5)	(6)
		INITIAL CONDITIONS		PULSED SIGNAL INPUTS		OUTPUT RESULT	
		1 OUTPUT	0 OUTPUT	SET INPUT	RESET INPUT	1 OUTPUT	0 OUTPUT
flip-flop in "1" condition initially	(1)	HI	LO	LO	LO	INDETERMINATE	
	(2)	HI	LO	LO	HI	HI	LO
	(3)	HI	LO	HI	LO	LO	HI
	(4)	HI	LO	HI	HI	HI	LO
flip-flop in "0" condition initially	(5)	LO	HI	LO	LO	INDETERMINATE	
	(6)	LO	HI	LO	HI	HI	LO
	(7)	LO	HI	HI	LO	LO	HI
	(8)	LO	HI	HI	HI	LO	HI

Figure 3.3. R-S Flip-Flop Truth Table

QUESTIONS

1. Assuming both inputs are HI, explain why the R-S flip-flop stays locked in one state or the other. Outline how each gate is operating in your explanation.
2. Explain how the transition from the "1" to the "0" condition takes place when the flip-flop receives a RESET pulse. Attempt to operate the SET and RESET rocker switches simultaneously to perform steps one and five in the truth table in Figure 3.2. Were the results consistent? Why?
3. What is the fan-out of each of the R-S flip-flop outputs?

III CLOCKED R-S FLIP-FLOP

Most often when a flip-flop is used, it is best to enable, or condition, its inputs with levels first and then allow it to make the required transition after it receives a pulse from another source. The pulse source in a digital system is called a "clock." The COMPUTER LAB has a clock included. However, for this experiment, the pulser switches will be used to provide single pulses instead of the train of pulses supplied by the clock.

The clocked R-S flip-flop has two separate circuit sections as shown in Figure 3.4. The flip-flop section is identical to the R-S flip-flop previously examined. There is also a steering network which steers clock pulses to either SET or

RESET the flip-flop. If the SET input line is enabled with a HI level, gate A will be enabled when a HI pulse is presented at the clock input. When gate A is enabled, it will provide a LO SET signal to the R-S flip-flop and the flip-flop will go to the "1" condition. If the SET input line has a LO level and the RESET line is enabled with a HI level, gate B will be enabled when a HI clock pulse occurs. When gate B is enabled, a LO RESET signal will be presented to the flip-flop and it will go to the "0" condition.

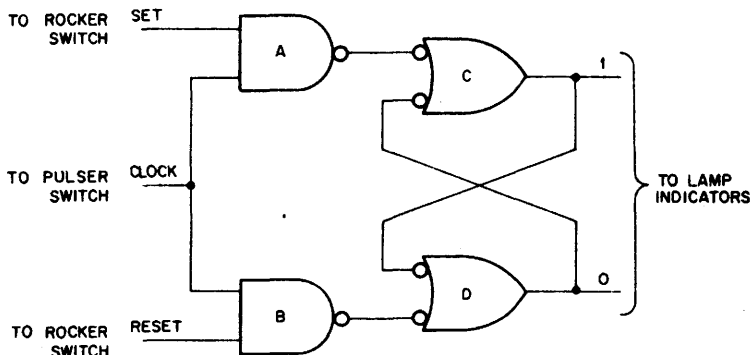


Figure 3.4 Clocked NAND Gate R-S Flip-Flop

EXPERIMENT 3.2: CLOCKED R-S FLIP-FLOP

Figure 3.5 is a truth table which shows all eight possible initial conditions that can exist prior to a clock pulse. Complete the final columns to show what conditions are present after the clock pulse. Note that if both the SET and RESET inputs are enabled before a clock pulse, the flip-flop condition is indeterminate after the pulse. Construct the circuit in Figure 3.4, connecting the SET and RESET inputs to rocker switches and the clock input to a pulser switch. Connect both the 0 and 1 outputs to lamp indicators. Test the circuit to verify that it operates as outlined in the truth table in Figure 3.5.

INITIAL CONDITIONS		SIGNAL INPUTS		AFTER CLOCK PULSE	
1 OUTPUT	0 OUTPUT	SET	RESET	1 OUTPUT	0 OUTPUT
LO	HI	LO	LO		
LO	HI	LO	HI	LO	HI
LO	HI	HI	LO	HI	LO
LO	HI	HI	HI		
HI	LO	LO	LO		
HI	LO	LO	HI	LO	HI
HI	LO	HI	LO	HI	LO
HI	LO	HI	HI		

Figure 3.5 Clocked R-S Flip-Flop Truth Table

QUESTIONS

- When does a flip-flop transition take place, on the leading edge or the trailing edge of a HI clock pulse? Why?

5. Why is the output indeterminate if both the SET and RESET inputs are enabled when the clock pulse occurs?
6. Explain the operation of the flip-flop by outlining the function of each gate.

IV D TYPE FLIP-FLOP

One way of insuring that there can be no indeterminate state in the operation of a flip-flop is to provide only one conditioning input which can either be HI or LO. The D type flip-flop has only one conditioning input, the D or data input. Whatever information is present at the D input prior to and during the clock pulse propagates to the "1" output when the leading edge of the clock pulse occurs. If the D input is HI prior to and during a clock pulse, the flip-flop goes to the "1" condition; if the D input is LO before and during a clock pulse, the flip-flop goes to the "0" condition.

EXPERIMENT 3.3 D TYPE FLIP-FLOP

Figure 3.6 is a simplified D type flip-flop. If HI information is present on the D conditioning input, gate A is enabled when a HI clock pulse occurs and SET information is passed onto the R-S flip-flop section causing the flip-flop to latch in the "1" condition. If a LO level is present on the D input, gate B is enabled when a clock pulse occurs and a RESET pulse is passed through, causing the R-S flip-flop to go to the "0" condition. Complete the truth table in Figure 3.7 to show the operation of a D type flip-flop. Construct the circuit in Figure 3.6 on the COMPUTER LAB connecting the D input to a rocker switch and the clock input to a pulser switch. Connect both the 1 and the 0 outputs to lamp indicators. Test the circuit to verify that it operates as outlined in the truth table in Figure 3.7.

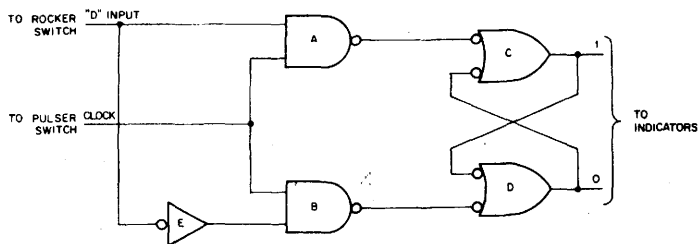


Figure 3.6 D Type Flip-Flop

INITIAL CONDITIONS		D INPUT	AFTER CLOCK PULSE	
1 OUTPUT	0 OUTPUT		1 OUTPUT	0 OUTPUT
LO	HI	LO	LO	HI
HI	LO	LO	LO	HI
LO	HI	HI		
HI	LO	HI		

Figure 3.7 D Type Flip-Flop Truth Table

QUESTIONS

7. Explain the operation of the D type flip-flop by noting the function of each of the gates.
8. Are there any indeterminate states in the truth table for this device?
9. Design a circuit to SET or RESET the flip-flop directly, independent of the state of the steering network.

V MASTER-SLAVE J-K FLIP-FLOP

One of the most useful flip-flops available is the J-K flip-flop. The two unique features of a J-K flip-flop are: A) a clock pulse will not cause any transitions in the flip-flop if neither the J nor the K input is enabled prior to the clock pulse, and B) if both the J and the K inputs are enabled before a clock pulse, the flip-flop will complement or change state when the clock pulse occurs. If a 1 is present before the clock pulse, a 0 will be present after the clock pulse and if a 0 is present before the pulse, a 1 will be present after. There is no indeterminate condition in the operation of a J-K flip-flop.

The J-K flip-flops used in the COMPUTER LAB are master-slave devices which trigger on the trailing edge of a clock pulse. They are actually two flip-flop circuits: a master flip-flop and a slave flip-flop. The information which is present at the J and K inputs is transmitted to the master flip-flop on the leading edge of a HI clock pulse and held there until the trailing edge of the clock pulse occurs when it is allowed to pass through to the slave flip-flop. If the J input is enabled and the K input is disabled prior to a clock pulse, the flip-flop will go to the "1" condition. If the K input is enabled and the J input is disabled prior to the clock pulse, the flip-flop will go to the "0" condition. If neither the J nor the K input is enabled before a clock pulse, the flip-flop will remain in whatever condition it is in. If both the J and the K inputs are enabled, the flip-flop will complement on the trailing edge of the HI clock pulse and go to the opposite of whatever state it was in.

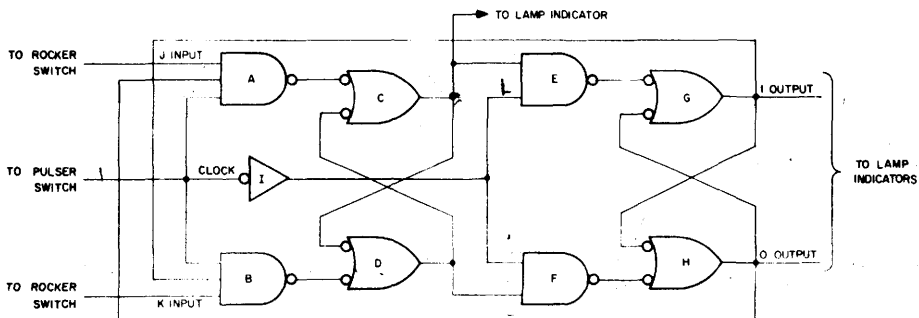


Figure 3.8 Master-Slave J-K Flip-Flop

Figure 3.8 shows a functional logic diagram of a master slave J-K flip-flop using NAND gates. Gates C and D are the master flip-flop. Gates G and H are the slave flip-flop. Gates A and B are the steering network of the master flip-flop and the steering network for the slave flip-flop is comprised of gates E, F, and I. The 1 output of the master flip-flop is point X. The operation of the flip-flop will be studied by examining the "1" to "0" transition of the flip-flops, with both the J and the K inputs enabled with a HI level before the clock pulse. When the leading edge of a HI clock pulse occurs, gate B will be enabled with three HI inputs. This will provide a RESET signal for the master flip-flop which will then go to the "0" condition. The slave flip-flop remains in the "1" condition while the clock pulse is HI because gate I is providing a LO signal to both gates E and F, thereby blocking inputs to the slave flip-flop. When the trailing edge of the clock pulse occurs, gate F will be enabled with a HI level at both its inputs and a RESET signal will be provided to the slave flip-flop, which will then go to the "0" condition. The next clock pulse, with both the J and K enabled, would cause the master flip-flop to go to the "1" condition on the leading edge of the clock pulse and cause the slave flip-flop to go to the "1" condition on the trailing edge of the pulse. Figure 3.9 is a truth table for the J-K flip-flop showing all eight possible initial conditions.

INITIAL CONDITIONS				FINAL CONDITIONS	
OUTPUTS		INPUTS		OUTPUTS	
1	0	J	K	1	0
LO	HI	LO	LO	LO	HI
LO	HI	LO	HI	LO	HI
LO	HI	HI	LO	HI	LO
LO	HI	HI	HI		
HI	LO	LO	LO		
HI	LO	LO	HI		
HI	LO	HI	LO		
HI	LO	HI	HI		

Figure 3.9 Master-Slave J-K Flip-Flop Truth Table

EXPERIMENT 3.4: J-K FLIP-FLOP

Complete the truth table in Figure 3.9 to show the final conditions that would result after a clock pulse.

Construct the flip-flop in Figure 3.8 on the COMPUTER LAB connecting the J and K inputs to rocker switches, the clock input to the pulser switch and the 1 outputs of both the master and the slave flip-flop to lamp indicators. (The 1 output of the master flip-flop is point X.) Test the operation of the flip-flop to verify that it operates as outlined for each condition of the truth table. When operating the flip-flop, note when transitions occur—on the leading edge (when the clock pulser is depressed) or on the trailing edge (when the clock pulser is released) of the clock pulse. Test the operation of one of the built in J-K flip-flops on the COMPUTER LAB to verify that it also operates according to the truth table in Figure 3.9. Connect the J, K, and RESET inputs to rocker switches, the clock input to a pulser switch and the 0 and 1 outputs to indicator lamps. The RESET overrides all other inputs and puts the flip-flop in the "0" state. Since RESET is an inverted input, a LO level enables it and a HI level disables it.

QUESTIONS

10. Are there any indeterminate states in the operation of the J-K flip-flop? Why?
11. Explain how the flip-flop makes a "0" to "1" transition if the J input is enabled prior to the clock pulse and the flip-flop is initially in the "0" condition.
12. Complete the diagram in Figure 3.10 showing the timing of the operations in the master and slave sections of the J-K flip-flop for the "0" and "1" transition. Assume that the clock pulse is 50 nano seconds (50×10^{-9} sec.) duration and that each NAND gate requires 15 nano seconds to propagate information from its inputs to its output.
13. Show how the circuit in Figure 3.8 could be modified to include a direct SET and a direct RESET input.

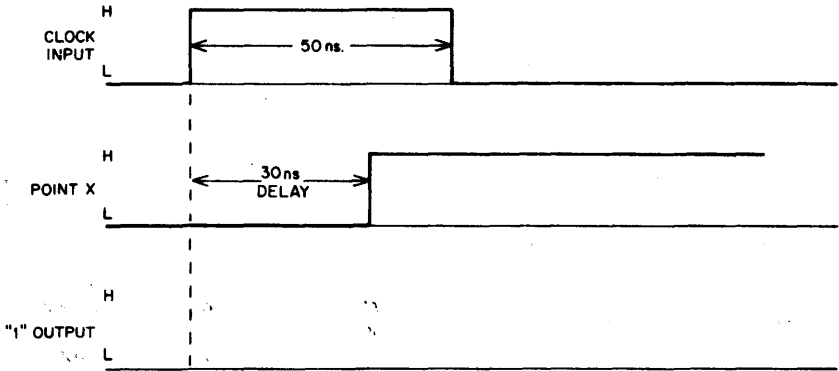


Figure 3.10 J-K Flip-Flop Timing

VI J-K SHIFT REGISTER

Frequently a digital signal has to be stored for a given number of clock pulses and, at the end of that time, be fed into another circuit in a system. For instance, a component tester on a production line would test a device and some number of positions later, if the device were defective, eject it. The information that the device had failed could be fed into a shift register and each time the device moved one position down the line, the information in the shift register would shift one position as well. When the defective device reached the ejector, the shift register would indicate that it was in position and the ejector would remove it from the production line.

Figure 3.11 is a four-bit J-K shift register which transmits information from its input to its output after four clock pulses. Assume all flip-flops are initially in the "0" condition and the rocker switch to the J input of flip-flop A is providing a HI level. When the first clock pulse occurs, flip-flop A will go to the "1" condition because its J input will be enabled with a HI level and its K input will be disabled with a LO level. Flip-flop B will remain in the "0" condition because its J input is disabled with a LO level and its K input is enabled with a HI level, both levels being presented by the outputs of flip-flop A. Similarly, flip-flop C will remain in the "0" condition, as will flip-flop

D. If the rocker switch is put in the LO condition before the next clock pulse, flip-flop A will go to the "0" condition on the following pulse. Flip-flop B will go to the "1" condition because its J input is enabled by the 1 output of flip-flop A and its K input is disabled by the 0 output of flip-flop A. Flip-flop C and flip-flop D will remain in the "0" condition because their K inputs are enabled with a HI level and their J inputs are disabled with a LO level. Successive clock pulses will shift the "1" which was initially fed into flip-flop A on the first clock pulse to flip-flop B, then to flip-flop C, and finally to flip-flop D. After the five clock pulses all flip-flops will be back in the "0" condition.

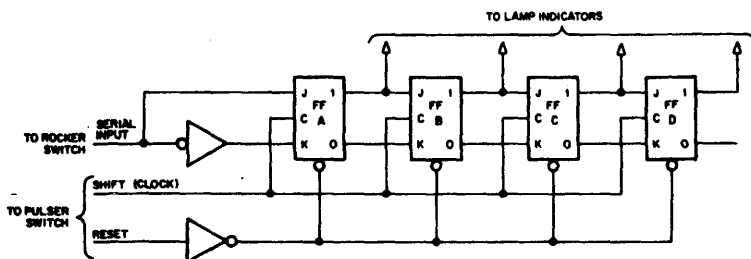


Figure 3.11 Four-Bit J-K Shift Register

EXPERIMENT 3.5: 4-BIT SHIFT REGISTER

Figure 3.12 is a truth table for the four-bit shift register. Complete the last three columns in the truth table. Construct the shift register Figure 3.11 on the COMPUTER LAB connecting the flip-flop 1 outputs to lamp indicators, the serial input to a rocker switch and the shift and RESET inputs to pulser switches. Test the shift register to verify that it operates as outlined in the truth table in Figure 3.12.

FLIP-FLOP A	FLIP-FLOP B	FLIP-FLOP C	FLIP-FLOP D
0	0	0	0
1	0	0	0
0	1	0	
0	0		
0			

Figure 3.12 Four-Bit Shift Register Truth Table

QUESTIONS

- Complete the timing diagram in Figure 3.13 to show the results obtained from operating the shift register as outlined above.
- Explain in detail how a shift register works. In your explanation describe:
 - How a "1" can propagate down the line
 - Why does a flip-flop in a "1" state with input being fed by a flip-flop in the "0" state return to the "0" state when a clock pulse is received.

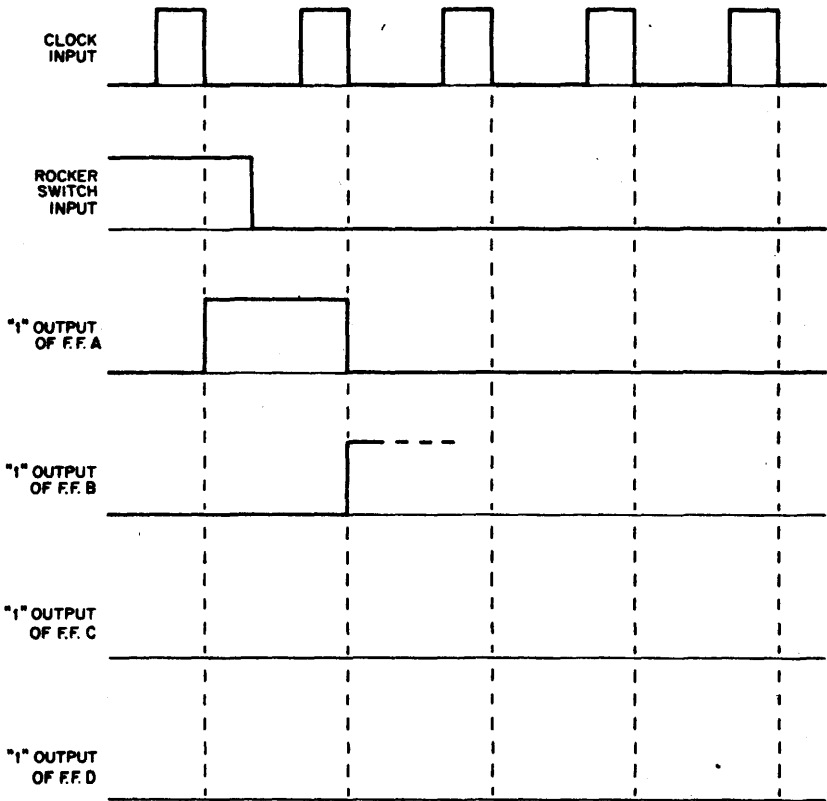


Figure 3.13 Four-Bit Shift Register Timing Diagram

SUPPLEMENTARY QUESTIONS

16. The D type flip-flop in Figure 3.6 is a simplified version of the flip-flop and has certain limitations. The most serious limitation is the fact that when the D input changes level while the clock input is still HI, the flip-flop 1 output will follow the level change. D type flip-flops used in computers do not have this problem. Whatever level is present at the D input prior to the clock pulse is transmitted to the 1 output of the flip-flop on the leading edge of a clock pulse. Further changes on the D input after the leading edge of the clock pulse has occurred do not affect the 1 output. Figure 3.14 is a functional block diagram of the actual D type flip-flop used in computers. The direct RESET and the direct SET inputs are disabled when HI. The H and L indications marked on each input and each output of the gates on the flip-flop indicate levels present in the flip-flop when it is in the "0" condition and the D input is HI. Explain how a "0" to "1" transition occurs on the leading edge of a clock pulse. Also explain why the D input has no effect on the flip-flop after the leading edge of the clock pulse has occurred.

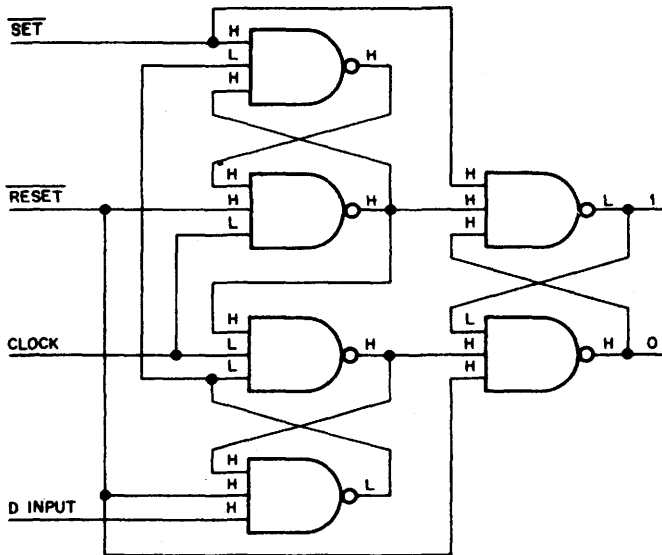


Figure 3.14 D Type Flip-Flop

17. Explain how the direct SET and the direct RESET work on the D type flip-flop. If a direct RESET signal is present and the D input is 'H' when the leading edge of a clock pulse occurs, will the flip-flop go to the "1" condition?
18. If the RESET is enabled on one of the J-K flip-flops on the COMPUTER LAB and a J is enabled when a clock pulse passes, what condition results after the clock pulse? Why?
19. Design and construct the additional circuitry to allow the 4-bit shift register to recycle any information it contains every four clock pulses. Also make provision to read a 1 into flip-flop A at any time and reset all flip-flops. This application of a shift register is known as a ring counter.
20. Design and construct additional circuitry to allow the 4-bit shift register to read information into the four flip-flops from four rocker switches simultaneously. The shift register can be RESET (to put all flip-flops in the "0" condition) by momentarily enabling the RESET line prior to reading in the information from the switch register.
21. Construct the switch tail ring counter in Figure 3.15 on the COMPUTER LAB and make a truth table for its operation.

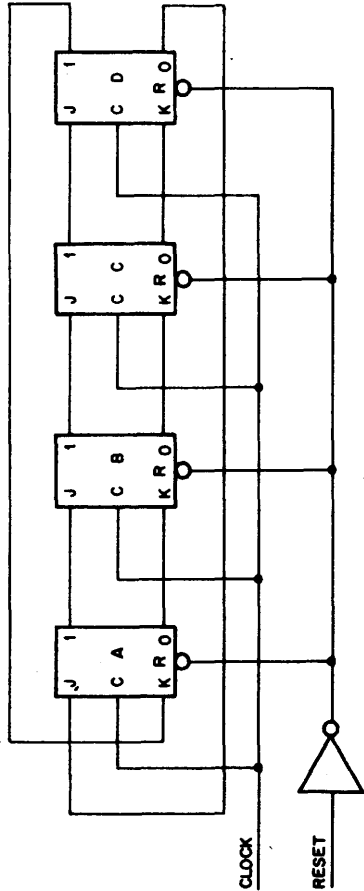


Figure 3.15 Switch Tail Ring Counter

CHAPTER 4

BOOLEAN ALGEBRA TO GATING NETWORKS

I INTRODUCTION

Boolean algebra is a form of algebra designed to deal specifically with two-state functions. The logic functions presented in previous experiments can be expressed in Boolean algebraic form. With Boolean, or two-state, algebra, two-state logic functions can be manipulated in a simple mathematical fashion rather than with cumbersome logic diagrams or truth tables. In this experiment the basic Boolean relations and the use of those relations to simplify Boolean functions will be discussed. Often the complexity of a gating network can be reduced to obtain reduced cost and greater operating speed. The following three techniques are used in the reduction process:

1. Reduce the Boolean expression to the simplest form.
2. Make a truth table and search for any simpler correlations that might not be apparent in the Boolean expression.
3. Design the gating circuit, keeping in mind the capabilities of the gates available, and search for greater reduction.

Several examples using these three simplification techniques will be developed and constructed in this experiment.

II BOOLEAN OPERATORS

The standard logic symbols for AND, OR, INVERT, etc. were presented in Chapter 1. Each symbol has a set of specific relations between input and output conditions. These same logic functions, or operators, can be represented in Boolean algebraic form. The algebraic signs used to express the logic relations between Boolean variables are the following:

- a) AND: \cdot (dot)
- b) OR: $+$ (plus sign)
- c) EXCLUSIVE OR: \oplus (circled plus sign)
- d) NEGATION: $\bar{}$ (superscript bar)
- d) EQUALITY: \equiv (equivalence sign)

Boolean expressions have several important characteristics which help to simplify their manipulation. Just as in normal algebra, a Boolean algebraic expression is composed of variables, constants and operation signs. In Boolean algebra, a variable can have only two possible states, 1 or 0. For example, the Boolean variable A can be either 0 or 1, but nothing else. Similarly, there are only two Boolean constants, 1 and 0. As with variables and constants, an entire Boolean function, or expression, can have only one of two possible values, 1 or 0. For example, the expression $[A + C \cdot (B \cdot C + D)]$ can represent only one of the two possible states, 1 or 0, depending upon the values of the variables A, B, C and D.

A Boolean function can be uniquely identified by listing the value of the function ("1" or "0") for each combination of input conditions. Such a listing is the familiar truth table. A helpful fact to know when preparing a truth table for a Boolean expression is that, if there are variables in the expression, there will be 2^n rows in the truth table. For example, the expression $A + B \cdot C \cdot D$ has four variables; therefore, its truth table will have 2^4 , or 16, rows.

Figure 4.1 shows the corresponding symbolic logic function for each of the five basic Boolean operators. The simplification possible using Boolean symbology is apparent even with these basic operators. The gating symbols are obviously much more difficult to use and manipulate than the corresponding Boolean operators.

Truth tables of all the basic Boolean operators are shown in Figures 4.2A through 4.2E.

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

The AND condition is valid only when A AND B are both valid

Figure 4.2a AND Truth Table

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

the OR condition is valid when either A OR B or both are valid

Figure 4.2b OR Truth Table

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

the EXCLUSIVE OR condition is valid if A OR B but not both are valid, i.e. if A and B are different

Figure 4.2c EXCLUSIVE OR Truth Table

A	B	$A \equiv B$
0	0	1
0	1	0
1	0	0
1	1	1

The EQUIVALENCE condition is valid if A and B are the same

Figure 4.2d EQUIVALENCE Truth Table

A	\bar{A}
0	1
1	0

NEGATION gives the binary opposite of a number or function

Figure 4.2e NEGATION Truth Table

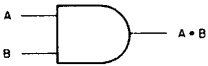

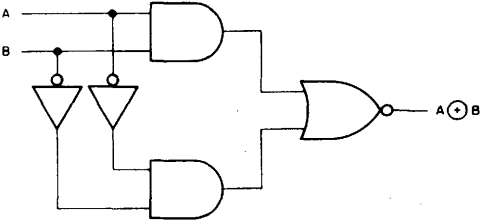
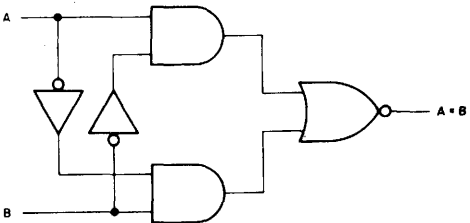
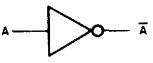
FUNCTION	BOOLEAN SYMBOL	LOGIC SYMBOL
AND	\cdot	
OR	$+$	
EXCLUSIVE OR	\oplus	
EQUIVALENCE	\equiv	
NEGATION	$\bar{}$	

Figure 4.1

The basic Boolean operators can be combined to give more complex functions. For example, the AND operator can be combined with NEGATION to obtain the "not AND" or "NAND" function, i.e.:

$$A \cdot B = \text{AND function}$$

$$\overline{A \cdot B} = \text{NAND function}$$

Negating each of the variables in an OR function gives a negated input OR function, i.e.:

$$A + B = \text{OR function}$$

$$\bar{A} + \bar{B} = \text{negated input OR function}$$

Combinations of the Boolean operators can be used to give any required logic function. For example, in Figure 4.3 all the basic gates used to this point are shown with the equivalent Boolean function.

III BOOLEAN LAWS

There are three basic laws of Boolean algebra:

- | | |
|-----------------------------|---|
| 1) It is commutative, i.e. | $A + B = B + A$ |
| | $A \cdot B = B \cdot A$ |
| 2) It is associative, i.e. | $A + B + C = A + (B + C) = (A + B) + C$ |
| | $A \cdot B \cdot C = A \cdot (B \cdot C) = (A \cdot B) \cdot C$ |
| 3) It is distributive, i.e. | $A \cdot (B + C) = A \cdot B + A \cdot C$ |
| | $A + (B \cdot C) = (A + B) \cdot (A + C)$ |

These laws can be proved by truth table techniques as shown in Figure 4.4. It should be noted that the second part of the distributive law is not valid in ordinary algebra.

A	B	C	$(B \cdot C)$	$A + (B \cdot C)$	$(A + B)$	$(A + C)$	$(A + B) \cdot (A + C)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

Figure 4.4 Truth Table of the Second Part of the Distributive Law

IV BOOLEAN IDENTITIES AND DeMORGAN'S THEOREM

There are nine basic Boolean identities of one variable and one constant using only AND, OR and INVERT operators. They are the following:

- | | | |
|----|-----------------------|-------------------|
| 1. | $\bar{\bar{A}} = A$ | } INVERT Identity |
| 2. | $A \cdot 1 = A$ | |
| 3. | $A \cdot 0 = 0$ | } AND Identities |
| 4. | $A \cdot A = A$ | |
| 5. | $A \cdot \bar{A} = 0$ | |
| 6. | $A + 1 = 1$ | } OR Identities |
| 7. | $A + 0 = A$ | |
| 8. | $A + A = A$ | |
| 9. | $A + \bar{A} = 1$ | |



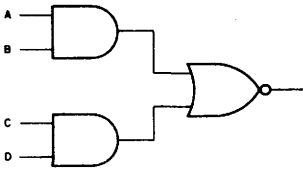



LOGIC FUNCTION	LOGIC SYMBOL	BOOLEAN EXPRESSION
NAND		$\overline{A \cdot B}$
NEGATED INPUT OR		$\overline{A + B}$
AND/NOR		$\overline{A \cdot B + C \cdot D}$
NOR		$\overline{A + B}$
NEGATED INPUT AND		$\overline{A \cdot B}$
INVERTER		\overline{A}

Figure 4.3

Each of the basic identities can be proved valid using truth table techniques. Identities 1, 2 and 9 are proved in Figures 4.4a through 4.4c

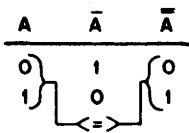


Figure 4.4a Proof of Identity #1
($\overline{\overline{A}} = A$)

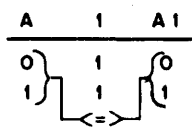


Figure 4.4b Proof of Identity #2
($A \cdot 1 = A$)

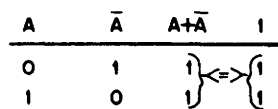


Figure 4.4c Proof of Identity #3
($A + \overline{A} = 1$)

The basic identities with one variable lead to identities with two or more variables. To simplify the writing of complex Boolean expressions, the following convention has been adopted which parallels that of ordinary algebra: $A \cdot B$ may also be written as AB . Therefore, the identity $(A+B) \cdot (C+D) = A \cdot C + A \cdot D + B \cdot C + B \cdot D$ can be rewritten as $(A+B) (C+D) = AC + AD + BC + BD$.

10. $A + AB = A$
11. $AB + A\bar{B} = A$
12. $(A+B) (A+\bar{B}) = A$
13. $A + \bar{A}B = A+B$
14. $(A+\bar{B}) B = AB$
15. $AC + AB + \bar{B}C = AC + \bar{B}C$
16. $(A+B) (B+C) (\bar{A}+C) = (A+B) (\bar{A}+C)$

The following examples demonstrate how basic one-variable identities and Boolean algebra laws can be used to prove two-variable identities:

Proof of Identity #10

$$\begin{aligned}
 & A + AB \\
 & \text{take out common factor} \\
 & = A(1+B) \left\{ \begin{array}{l} \longrightarrow \text{Identity \#6. } 1+B=1 \\ \longrightarrow \text{Identity \#1. } 1 \cdot A = A \end{array} \right. \\
 & = A
 \end{aligned}$$

Proof of Identity #12

$$\begin{aligned}
 & (A+B) (A+\bar{B}) \\
 & = AA + A\bar{B} + BA + B\bar{B} \quad \left\{ \begin{array}{l} \longrightarrow \text{IDENTITY \#5. } B\bar{B}=0 \\ \longrightarrow \text{IDENTITY \#4. } AA=A \end{array} \right. \\
 & = A + A\bar{B} + BA \\
 & = A + A(B+\bar{B}) \quad \left\{ \begin{array}{l} \longrightarrow \text{IDENTITY \#9. } B+\bar{B}=1 \\ \longrightarrow \text{IDENTITY \#2. } A+1=A \end{array} \right. \\
 & = A + A \cdot 1 \\
 & = A + A \quad \left\{ \begin{array}{l} \longrightarrow \text{IDENTITY \#8. } A+A=A \\ \longrightarrow \text{IDENTITY \#1. } A=A \end{array} \right. \\
 & = A
 \end{aligned}$$

Identities can also be proved by truth table techniques. The method is to determine the value of the function on the left side of the identity for all possible combinations of conditions of the variables and then to determine the value of the right side of the equation for all possible combinations. If the identity is true, both sides will be equivalent for corresponding conditions. The following examples (Figures 4.5 and 4.6) demonstrate the truth table method of proof for identities 14 and 15.

A	B	\bar{B}	$A+\bar{B}$	Left Side $B(A+\bar{B})$	Right Side $A \cdot B$
0	0	1	1	0	0
0	1	0	0	0	0
1	0	1	1	0	0
1	1	0	1	1	1

Figure 4.5 Truth Table of Identity #14

A	B	C	A · C	A · B	\bar{C}	B · \bar{C}	Left Side		Right Side
							A · C + A · B + B · \bar{C}	A · C + B · \bar{C}	
0	0	0	0	0	1	0	0	0	
0	0	1	0	0	0	0	0	0	
0	1	0	0	0	1	1	1	1	
0	1	1	0	0	0	0	0	0	
1	0	0	0	0	1	0	0	0	
1	0	1	1	0	0	0	1	1	
1	1	0	0	1	1	1	1	1	
1	1	1	1	1	0	0	1	1	

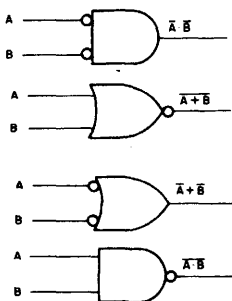
Figure 4.6 Truth Table of Identity #15

There is another very valuable theorem called De Morgan's Theorem that can be used to simplify a complex Boolean function. In its most general form, De Morgan's Theorem states "If, in a Boolean function each + is replaced by a ·, and each · is replaced by a +, and each variable is replaced by its complement, the result is the essence of the given function." The following equations form the essence of De Morgan's Theorem:

- $\overline{A \cdot B} = \bar{A} + \bar{B}$
- $\overline{A + B} = \bar{A} \cdot \bar{B}$

Repeated application of these equations can be used to simplify complex Boolean expressions.

The first equation states mathematically the equivalence of the negated input AND gate and the NOR gate. The second equation states the equivalence of the negated input OR gate and the NAND gate (Figure 4.7).



According to De Morgan's Theorem, since $\overline{A \cdot B} = \bar{A} + \bar{B}$, these two logic gates perform identical functions.

According to De Morgan's Theorem, since $\overline{A + B} = \bar{A} \cdot \bar{B}$, these two logic gates perform identical functions.

Figure 4.7 De Morgan's Theorem as Applied to Logic Gates

The above identities, laws and theorems can be used to simplify Boolean functions algebraically. They can all be verified using truth table techniques if desired. Other algebraic tools such as multiplying out and factoring can be used also to reduce a Boolean function without reference to truth tables or mapping techniques.

The following example demonstrates how a function can be reduced algebraically:

$$\overline{[A \cdot (\bar{A} \cdot \bar{B})] \cdot (C \cdot D)}$$

The initial function has several negations which can be removed in part by applying De Morgan's Theorem on the two halves of the AND expression.

$$\overline{[A \cdot (\bar{A} \cdot \bar{B})] + (C \cdot D)}$$

De Morgan's Theorem can again be applied to the first term in the function to remove another negation.

$$\overline{[A \cdot (\bar{A} + B)] + (C \cdot D)}$$

Identity #14 can now be applied to the first term in the function to give the final reduced function:

$$\overline{(A \cdot B)} + (C \cdot D)$$

QUESTIONS

1. Make truth tables for the following:
 $A \cdot B \cdot C$, $A+B+C$, $A \oplus B \oplus C$ and $A \equiv B \equiv C$.
2. Prove identities 11, 12 and 15 by means of truth tables.
3. Prove identity 14 using the single variable identities and multiplication.

V BOOLEAN FUNCTIONS TO GATING NETWORKS

There is a definite sequence of operations which can be followed to go from a required logic function to the simplest possible gating network. The first operation is to translate the logic requirement into a Boolean expression. The Boolean expression is then simplified to yield a reduced Boolean equation. Then construct a truth table of the simplified Boolean equation. Correlations not readily apparent in a Boolean expression can often be found in a truth table and used to further simplify the Boolean expression. If such correlations exist, the Boolean expression should be simplified again. The final simplified version of the Boolean expression should then be translated into a gating network. The gating network again presents possibilities for simplification to make best use of the particular logic elements available. Finally, the device should be constructed according to the simplified gating network and tested to verify that it operates as required by the original Boolean expression. These techniques are not rigid and a more experienced designer may in some cases change the order of operations or even skip steps.

A. Boolean Statement

The following example demonstrates translation of a logic requirement into a Boolean expression: A machine will operate if: "the safety switch is on and the foreman's key is in position and the operator's hand is not in the stroke path of the machine and a work piece is in position." Each of the Boolean variables should be given a name, for example, the safety switch being on will be called "A," the foreman's key being in position will be called "B," the operator's hand being in the path of stroke will be called "C," the work piece being in position will be called "D." The machine-start information will be called "F." The Boolean expression representing the condition necessary to start this machine is:

$$F = A \cdot B \cdot \overline{C} \cdot D$$

QUESTIONS

4. A bowler has three pins left up, A, B, and C. Two of the pins must be knocked down on the next bowl in order for him to win the game. Define by means of a Boolean expression all possible ways in which he can win the game. Also, in a second Boolean expression, define all the ways that the player can lose the game.
5. There are five factories close together, each one turning off its lights at a different time. Factory A turns off its lights at 5:30 pm, factory B turns off its lights at 6:00 pm, factory C turns off its lights at 5:00 pm, factory D turns off its light at 5:15 pm and factory E turns off its lights at 5:45 pm.

- Part A:** Define a Boolean function dependent on the condition of the lights which will be valid in the time interval between 5:15 pm and 5:45 pm.
- Part B:** Define a Boolean expression dependent on the condition of the lights which will be valid from 5:45 pm to 6:00 pm.
- Part C:** Define one Boolean expression which will be valid for both the following time intervals:

5:00 pm to 5:15 pm or
5:45 pm to 6:00 pm

B. Boolean Simplification

Once a Boolean function has been established the first step in translating it into a gating network is to simplify it using Boolean algebraic techniques. The identities, laws and theorems presented previously can be used in the simplification process.

QUESTIONS

6. Simplify the following functions using Boolean algebraic techniques:

- A) $\overline{A} \cdot \overline{[(B \cdot C) + D]} \cdot \overline{A}$
- B) $\overline{(A \cdot B)} + \overline{(A + B)}$
- C) $\overline{(A + AB)} + \overline{AB}$
- D) $\overline{(A + B)} \cdot \overline{(A \cdot B)}$
- E) $\overline{[(AB + \overline{AB}) + AB]} + \overline{AB}$

C. Truth Table Reduction

Even after simplifying an expression with Boolean algebra, the final result may indicate a logic design which overlooks the obvious. If a truth table is constructed, often an alternate simpler expression of the function becomes apparent. A typical example of an application for this method of simplification would be the function:

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

If a logic circuit were constructed according to the above expression, the logic network would take the form shown in Figure 4.8 with an AND gate for each term of the expression and an OR gate for the final output. But if a truth table were constructed as shown in Figure 4.9, it would become obvious that there are only two sets of conditions where $F = 0$ and it would therefore be easier to search for conditions where $F = 0$ rather than where $F = 1$.

The two conditions where $F = 0$ are $A \overline{B} \overline{C}$ and $A \overline{B} C$. A logic circuit could be constructed to search for these two conditions and if they were found, the function F would equal "zero." The output of the first condition OR the output of the second condition would be \overline{F} . To obtain the function F an extra stage of inversion could be added to the end of the logic circuit as shown in Figure 4.10.

The variable C can be either "one" or "zero" for F to be "zero." C therefore, has no effect on the value of the function F and it can be ignored in searching for the condition $F = 0$. The only requirements for F to be "zero" are that $A = 1$ and $B = 0$. The circuit can be further reduced to the form shown in Figure 4.11.

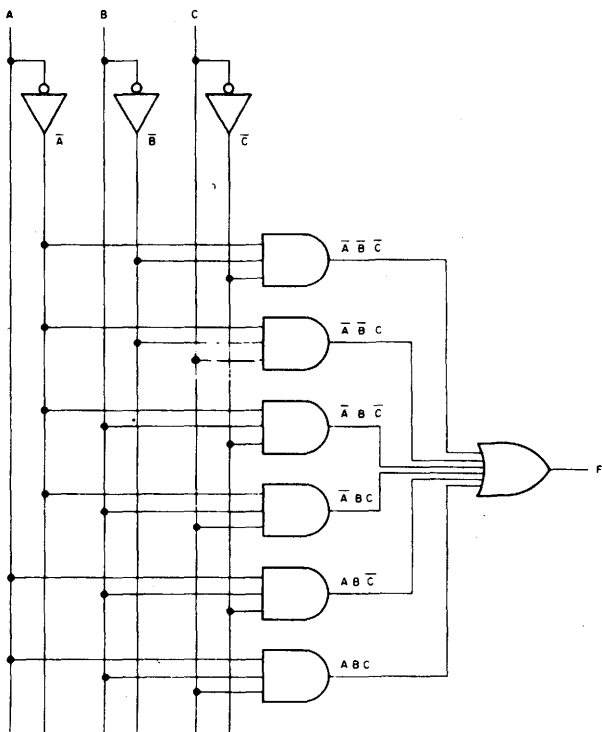


Figure 4.8 Gating Network for: $F = \bar{A} \bar{B} \bar{C} + \bar{A} \bar{B} C + \bar{A} B \bar{C} + \bar{A} B C + A \bar{B} \bar{C} + A B \bar{C}$

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

Figure 4.9 Truth Table for: $F = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC$

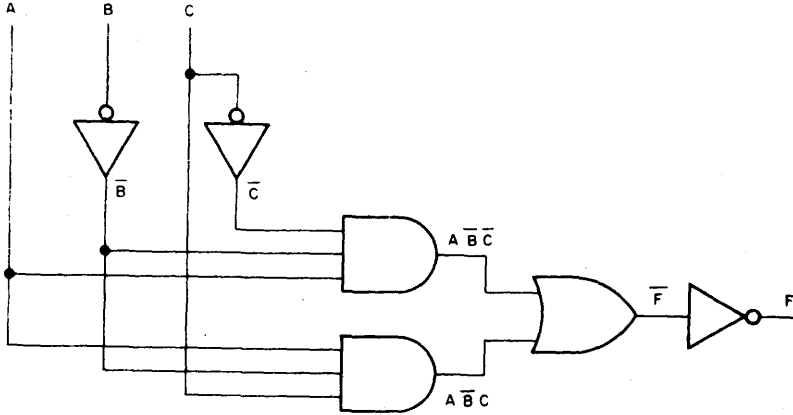


Figure 4.10 Logic Network to Detect $F = 0$

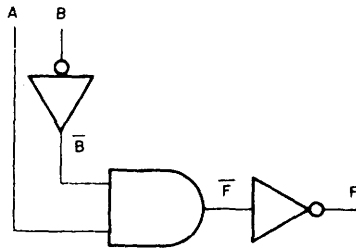


Figure 4.11 Simplified Network for $F = 0$

The rules for this type of simplification are not hard and fast. In general, one looks for the simplest correlation between the input conditions and the output. In some of the reference texts, listed in the appendix, mapping techniques, such as Karnaugh maps, are described. See the Appendix for a brief discussion of Karnaugh map simplification.

QUESTIONS

7. Make a truth table for the following functions and then rewrite the functions in reduced Boolean form.

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

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

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

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

D. Hardware Considerations

As was shown in Chapter 2, any of the basic Boolean functions can be constructed using combinations of the NAND and NOR gates available on the COMPUTER LAB. The first step in the translation is to express the Boolean function in terms of AND, OR, and INVERT functions only. The most desirable form of expression is a sum of products. For example:

$$A \oplus B = (A \cdot \bar{B}) + (\bar{A} \cdot B)$$

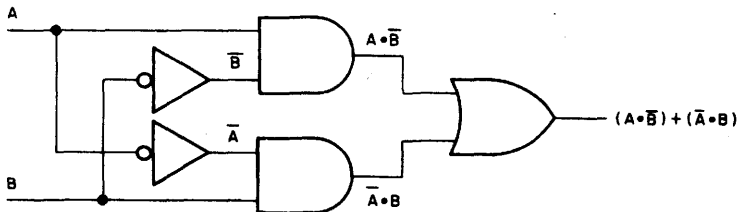


Figure 4.12 Gating Diagram for: $A \oplus B = (A \cdot \bar{B}) + (\bar{A} \cdot B)$

This expression can be translated directly into a symbolic gating diagram as shown in Figure 4.12. The functions used in the diagram in Figure 4.12 are not directly available on the COMPUTER LAB. However, these functions can be constructed from available gates as shown in Figure 4.13. Figure 4.14 is a circuit using functions available on the COMPUTER LAB to perform the operation of the non-inverting circuit in Figure 4.12. Even though the circuit

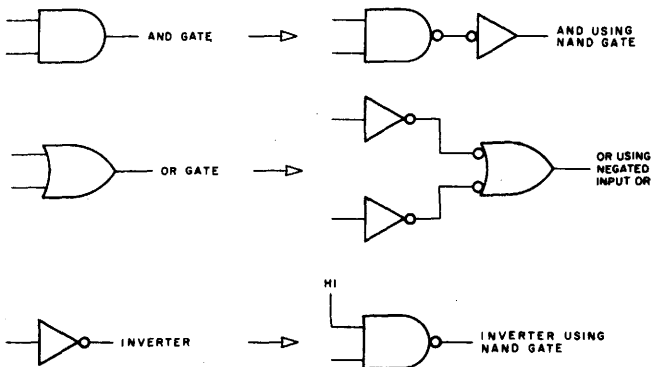


Figure 4.13 Logic Functions from Gates Available on COMPUTER LAB

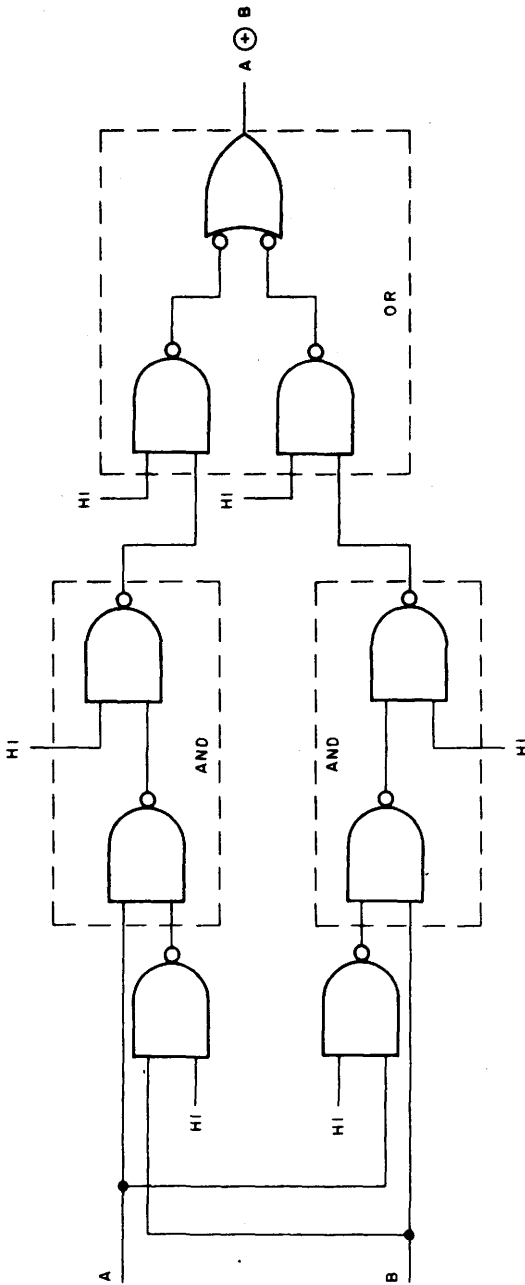


Figure 4.14 Exclusive OR Using Functions Available on the COMPUTER LAB

can now be constructed, it can still be simplified. The first obvious simplification possible is removing the two sets of series inverters. Identity #1 explains why these inverters can be removed: the double inverse of any function is that function itself. Therefore, the output of the two inverters is the same as the input and the inverters are performing no necessary function. The outputs of the two NAND gates can be connected directly to the inputs of the negated input OR gate as shown in Figure 4.15. The NAND and negated input OR gates are generating the AND/OR function or in Boolean symbology, a sum of products.

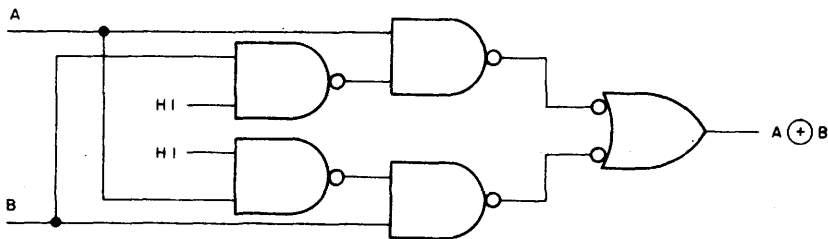


Figure 4.15 Exclusive OR Using NAND Gates

The circuit is now down to the simplest form obtainable using only NAND gates, but an AND/NOR gate with an inverter could be used to perform the AND/NOR function in two gates instead of three as shown in Figure 4.16.

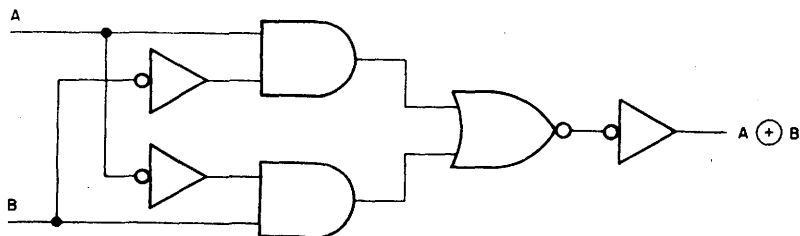


Figure 4.16 Exclusive OR Using AND/NOR and NAND Gates

Prior to inversion, the output of the AND/NOR gate in Figure 4.16 is the function $A \equiv B$. This can be confirmed algebraically by examining the truth table in Figure 2.11. Equality and Exclusive OR functions provide complementary outputs for the same input conditions. Therefore:

$$\overline{(A \equiv B)} = A \oplus B$$

If the final output inverter in Figure 4.16 was removed, the resulting circuit would detect equality. In order to make best use of the AND/NOR gate in the Exclusive OR application, the inverting property of the output NOR section of the gate should be used in the design. To further simplify the circuit, the final inverter should be removed and, to maintain the same output, complementary information must be provided to the AND/NOR gate inputs. In the circuit in

Figure 4.16 , one of the AND gates is enabled when the Exclusive OR function is "1".

To provide complementary information to the input of the AND/NOR gate, the AND sections of the gate should be enabled when the Exclusive OR function is "0." If one of the AND gates is enabled, the output NOR gate will be enabled to give a LO or "0" output indicating that the Exclusive OR condition is not present. If neither AND gate of the AND/NOR gate is enabled, the output NOR gate will be disabled to give a "1" or HI output, indicating that the Exclusive OR is present. The most simplified form of the Exclusive OR circuit is shown in Figure 4.17.

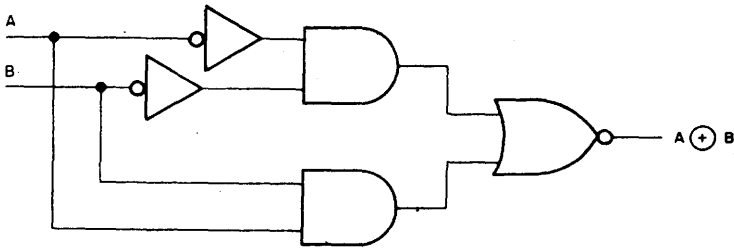


Figure 4.17 Final Simplified Exclusive OR Circuit

Reduction techniques used in this example were again not rigorous and relied on step by step reasoning based on a thorough knowledge of the gates available on the COMPUTER LAB. The simplification using only NAND gates is relatively straightforward and the circuit presented in Figure 4.15 would be completely acceptable if the AND/NOR gate were not available. However, the circuit reductions possible using AND/NOR gates make a search for applications of this gate an essential part of the design of any system. Before attempting to do reduction by using direct circuit simplification it is advisable to review Chapter 2 to be certain of the operation of each gate.

EXPERIMENT 4.1: GATING CIRCUIT SIMPLIFICATION

The following circuits (Figures 4.18-4.21) are the results of translating Boolean equations directly into gating networks. Follow the circuits through gate by gate and determine the Boolean equation for each network. Construct the unsimplified circuit and make a truth table of its operation; simplify the Boolean equations using all the techniques mentioned in this chapter including Boolean reduction, truth table reduction and direct circuit reduction. Reduce the circuit to the minimum number of gates as indicated under each diagram. Construct each reduced circuit and verify that it operates as outlined in the truth table for the unsimplified circuits. The indicator lamps can be used to detect logic levels at various points in the circuit if necessary. Consider the AND/NOR function as a single gate.

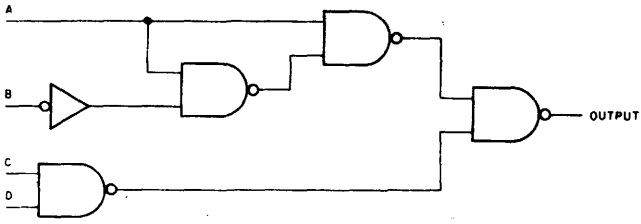


Figure 4.18 2 Gates Minimum

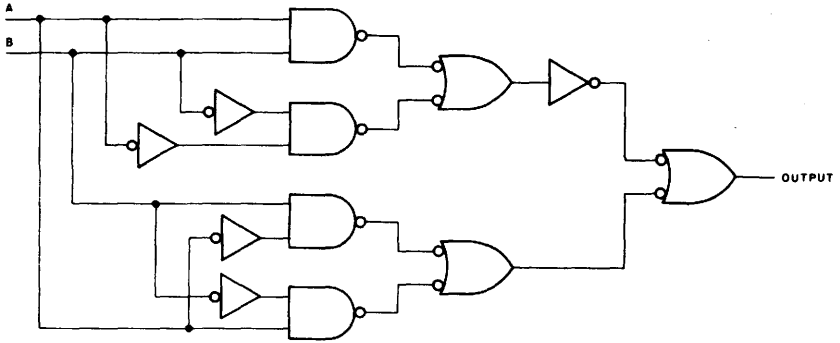


Figure 4.19 3 Gates Minimum

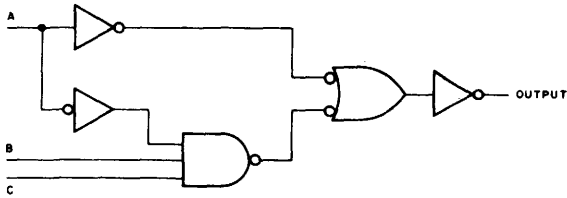


Figure 4.20 1 Gate Minimum

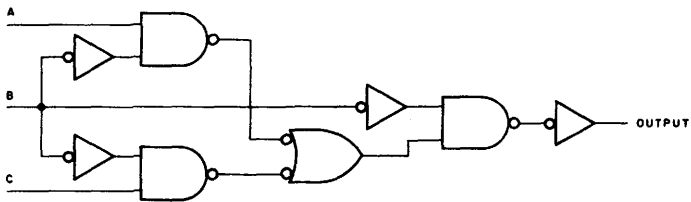


Figure 4.21 2 Gates Minimum

VI EQUALITY AND RELATIVE MAGNITUDE DETECTOR

In Chapter 2 an equality detector was built to check when the two switch registers were equal. If the two numbers were unequal there was no way of determining which number was the greater. In this experiment a device will be constructed which will tell not only if two numbers are equal but, if they are unequal, which of the two is greater.

When two numbers are unequal the one with the greater value is greater at the most significant unequal bit. For example, the number $A = 11011$ is greater than the number $B = 10111$ because the 2^3 bit of number A is greater than the 2^3 bit of number B. Since the 2^3 bit is the most significant inequality there is no need to examine any less significant bits. As a matter of fact, in this case if the 2^2 bit were examined by mistake to determine which number is greater, B would wrongly appear larger than A.

An equality and relative magnitude detector can be built in two sections: the first section determines if corresponding bits of the two numbers being compared are equal. This circuit is shown in Figure 4.22. The second circuit, as shown in Figure 4.23, determines which of the two numbers is greater if an inequality exists. The equality and relative magnitude detector in this experiment will compare two 3-bit binary numbers.

The equality detector section of this circuit will give a "1" or HI on output D if the 2^2 bits are equal, if $A_2 = B_2$. The equality detector output E will give a "1" or HI indication if $A_1 = B_1$. The equality detector output F will give a HI or "1" indication if $A_0 = B_0$.

The relative magnitude detector section of the circuit shown in Figure 4.23 takes input information from the equality detector. The operation of the relative magnitude section of the circuit will be studied through an example where number $A = 101$ and number $B = 110$. The 2^2 bits of numbers A and B are equal so the D input from the equality detector to the relative magnitude detector will be HI. The A_2 input will be HI, the B_2 input will be LO. Therefore gate 2 will be disabled and its output will be HI.

The E input of the relative magnitude detector section will be receiving a LO signal from the equality detector because the 2^1 bits of numbers A and B are not equal. Therefore gate 4 will not be enabled; its output will be HI and give a HI input to gate 3. Gate 3 will give a LO output. Gate 1 will be receiving HI information from the D input of the relative magnitude detector but both the A_1 and the B_1 inputs will be LO. Therefore gate 1 will be disabled and have a HI output. The F input from the equality detector will be LO indicating that the 2^0 bits of numbers A and B are not equal. The A_0 and B_0 inputs will both be HI but, because the input to gate 0 is also fed by the LO output of gate 3, gate 0 will not be enabled and its output will therefore be HI. Gate 6 has a HI input from D and a LO input from E and F. Consequently it will be disabled and its output will be HI. Since A does not equal B the $A \equiv B$ output is HI. Gate 5 has three HI inputs from gates 0, 1, and 2. It is therefore disabled and its output is LO. Since the $A > B$ output is LO, number A is not greater than B. Therefore, since A is not greater than B and A does not equal B, B must be greater than A.

The relative magnitude detector section of the circuit has searched for the first inequality and when that inequality was found, the rest of the relative magnitude detection circuitry was disabled. The first inequality in the example above occurred at the 2^1 bit and gate 3 provided a LO output, disabling the 2^0 relative magnitude detector section of the circuit. The output of gate 3

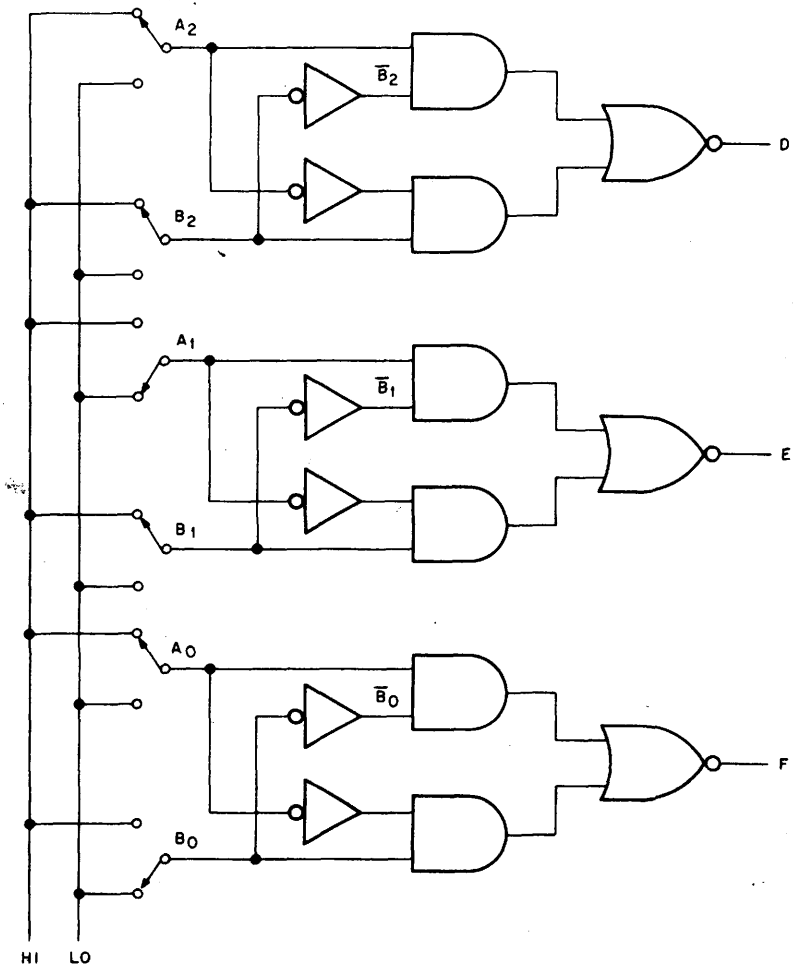


Figure 4.22 Equality Detector

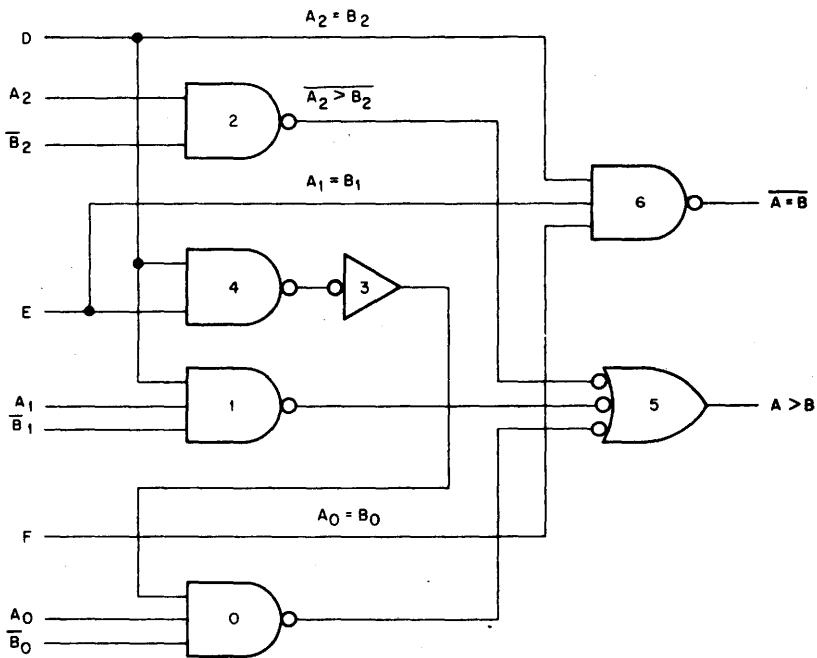


Figure 4.23 Relative Magnitude Detector

could be called "equals-so-far" output. In order to perform a check for relative magnitude, a section of the detector must receive equal-so-far information from the next most significant section.

EXPERIMENT 4.2: EQUALITY AND RELATIVE MAGNITUDE DETECTOR

Construct the equality and relative magnitude detector circuit as shown in Figures 4.22 and 4.23 on the COMPUTER LAB. Connect the A inputs to the three left-most rocker switches, connect the B inputs to the three right-most rocker switches starting with the least significant bit on the right of each group of switches. Test the operation of the circuit by setting a number on switch register A and recording the output obtained for all eight possible settings of switch register B. Repeat the procedure for several values of switch register A and confirm the circuit operates as outlined above.

QUESTIONS

8. Explain in detail how the detector works by examining the function of each gate in the circuit. Recall the method of determining the relative magnitude of the two numbers mentioned earlier in this chapter.
9. Write a Boolean equation for the output of each of the gates in the circuit in Figure 4.23.
10. How could the $A \equiv B$ output circuitry be simplified?
11. Design and construct an extension to the detector to give it three outputs, $A \equiv B$, $A > B$, $B > A$ where the $A \equiv B$ output goes HI when A is equal to B, the $A > B$ output goes HI when A is greater than B and the $B > A$ output goes HI when B is greater than A.
12. Design an 8 bit equality and relative magnitude detector using only the AND/NOR gate, 2 input NAND gates, 3 input NAND gates, and 4 input NAND gates.

SUPPLEMENTARY QUESTIONS

13. Make a truth table similar to the one in Figure 3.2 for the three Boolean variables A, B, and C showing the five Boolean functions operating on all eight possible combinations of the binary variables.
14. Write the Boolean equation that covers the operation of $A > B$ output of the three bit equality and relative magnitude detector. The equation will be of the form: Output = F ($A_0, B_0, A_1, B_1, A_2, B_2$).
15. Make a truth table showing the outputs that would be obtained from a two-bit equality and relative magnitude detector. Show all sixteen possible input combinations of the two 2-bit numbers.
16. Assuming 15×10^{-9} seconds propagation delay for each NAND and AND/NOR gate, determine the delay of the three-bit equality and relative magnitude detector.
17. With the costs and propagation delays listed below, what reductions in cost and delay have been accomplished in the simplifications referenced in question 6.

2-input NAND	\$ 9.20	15 nsec
3-input NAND	\$12.00	15 nsec
4-input NAND	\$16.00	15 nsec
AND/NOR Gate	\$16.80	15 nsec

CHAPTER 5

BINARY COUNTERS

I INTRODUCTION

The word "computer" is taken from the French word meaning "to count." As might be expected, counters of one form or another are basic subsystems in a computer. The counters presented in this chapter will use the binary counting sequence as shown in Figure 5.1. (Also see Appendix.)

2^3	2^2	2^1	2^0	DECIMAL
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Figure 5.1

J-K flip-flops can be used in binary counters. The two states of a J-K flip-flop, a "1" state, and a "0" state, can be used to represent a bit in a binary number. If the flip-flop is in the "1" state, that bit is a 1. If the flip-flop is in the "0" state, that bit is a 0.

II ASYNCHRONOUS COUNTERS

If both the J and K inputs of the J-K flip-flop are connected to a HI logic level, the output of the flip-flop changes its state, or complements, every time a HI to LO level change occurs at its clock input. This is shown in the clock input and 1 output of flip-flop O lines of the timing diagram in Figure 5.2. The 1 output of the flip-flop has divided the number of pulses at its clock input by two (i.e., the 1 output is HI on every other clock pulse). If the clock input of a second J-K flip-flop were connected to the 1 output of the first, it would similarly divide the number of pulses at its clock input by two. The 1 output of the second flip-flop would have one-half as many pulses as appear at the 1 output of the first flip-flop and one-quarter as many pulses as appear at the clock input of the first flip-flop. Additional stages could be added to this device so that each stage would divide the output of the previous stage by two. Figure 5.2 shows a timing diagram for a 4-stage device of this type. Note that each successive stage divides the signal present at the output of the previous stage by two. The circuit for this device is shown in Figure 5.3. This circuit can divide by sixteen the number of pulses appearing at the clock input of the first flip-flop, and it can also count in binary.

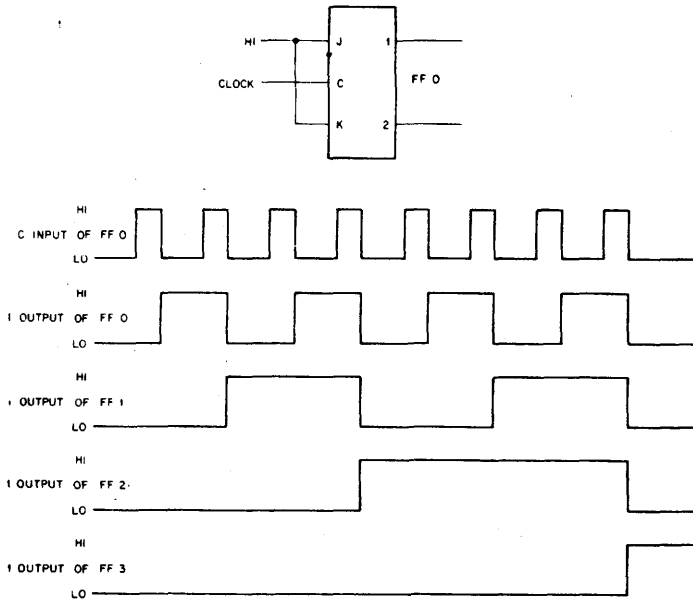


Figure 5.2 Timing Diagram

Suppose all four flip-flops are initially in the "0" state (the 1 output of each is LO). The trailing edge of the first pulse into the clock input of flip-flop 0 will cause it to complement and go from the "0" to the "1" condition. At the end of the first pulse, flip-flops 3, 2, and 1 are still in the "0" state and flip-flop 0 is in the "1" state. The condition of the flip-flops can be written as 0001.

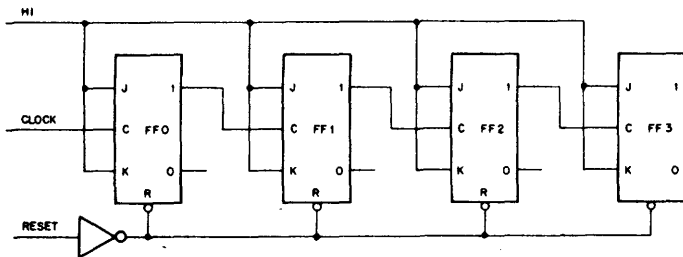


Figure 5.3 Asynchronous Up Counter

When the trailing edge of the second input clock pulse occurs, flip-flop 0 again complements, going from the "1" state to the "0" state. Flip-flop 1 sees the falling level at the output of flip-flop 0 and also complements, going from the "0" state (initial condition) to the "1" state. The condition of the

flip-flops after two clock pulses can be represented by 0010. Flip-flop conditions resulting after the third to fifteenth input clock pulses can be written as follows; (3) 0011 (4) 0100 (5) 0101 (6) 0110 (7) 0111 (8) 1000 (9) 1001 (10) 1010 (11) 1011 (12) 1100 (13) 1101 (14) 1110 (15) 1111. The counter is counting in the binary code given in Figure 5.1.

EXPERIMENT 5.1: ASYNCHRONOUS BINARY UP COUNTER

Construct the counter shown in Figure 5.3 on the COMPUTER LAB, connecting the clock and reset inputs to pulsers and the flip-flop 1 outputs to lamp indicators. Reset all flip-flops by pushing the reset pulser once. Push the Clock pulser once and record the outputs obtained on a truth table. Continue depressing the clock pulser and recording the output for all 16 possible counts. Check that the truth table is the same as Figure 5.1.

A clock in a digital system provides a continuous train of HI clock pulses. Disconnect the clock pulser switch and connect the clock on the COMPUTER LAB to the input of flip-flop 0; set the clock to run as slowly as possible. The counter will count continuously in this mode, with the lamps giving a visible count sequence. Turn the clock control to the right and observe the counting rate increases.

The name "asynchronous" means that each flip-flop of the counter does not operate simultaneously. One flip-flop causes the next to complement. This type of counter is often picturesquely called a "ripple through counter." This name is particularly apt since information goes through the counter like a wave from the least significant bit to the most significant bit.

EXPERIMENT 5.2: MODIFIED ASYNCHRONOUS BINARY COUNTER

Modify the counter by connecting the Clock inputs of each bit to the 0 output of the previous bit as shown in Figure 5.4. Reconnect the clock input of flip-flop 0 to a pulser. Reset all flip-flops to "0" by pushing the Reset pulser once. Make a truth table for the operation of the modified counter by recording the counter output after each of 16 clock pulses.

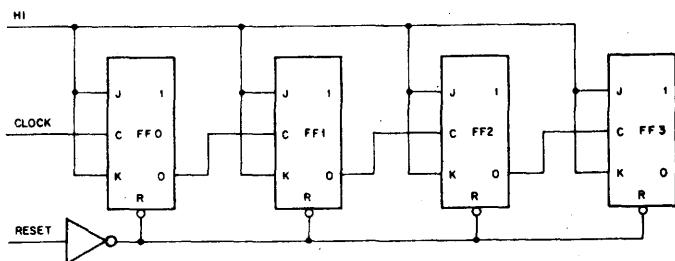


Figure 5.4 Modified Asynchronous Binary Counter

QUESTIONS

1. Describe the operation of the asynchronous binary Up Counter in Figure 5.3 for transitions between the binary numbers 1000 and 1100.
2. What function does the modified counter in Figure 5.4 perform?
3. Design and construct an 8-bit asynchronous binary Up Counter. The RESET circuitry will have to be modified to prevent overloading a gate output with the RESET inputs from 8 flip-flops.

4. Connect the J and K inputs of the 2^0 flip-flop to a rocker switch. Test the operation of the counter with the switch in the HI position. Test the operation of the counter with the switch in the LO position. What function is the switch performing?

III SYNCHRONOUS COUNTERS

The asynchronous, or ripple through, counter had to propagate information from the least significant to the most significant bit one stage or bit at a time. A synchronous counter makes all the required changes of state simultaneously. For instance, when changing from the binary number 0111 to the binary number 1000, all bits complement simultaneously. To do the same numerical transition, with the asynchronous counter previously described, first the 2^0 input would have to complement, then the 2^1 bit, then the 2^2 bit, and finally the 2^3 bit, each bit receiving its complement instruction from the next less significant bit.

A J-K flip-flop will remain in its present state when a clock pulse occurs if its J and K inputs are both disabled with a LO level. If both the J and K inputs are enabled with a HI level, the flip-flop will complement on a clock pulse. The 2^0 bit flip-flop of a synchronous counter will have its J and K inputs continuously enabled with a HI level and it will complement on every clock pulse as shown in Figure 5.1. The 2^1 bit complements on clock pulses when there is a "1" in the 2^0 bit. The J and K inputs of the 2^1 bit must be enabled when the 2^0 bit is in the "1" condition. Similarly, the 2^2 bit complements on every clock pulse if there is a "1" in both the 2^0 bit and the 2^1 bit. The 2^2 bit flip-flop must have its J and K inputs enabled when this condition occurs. The 2^3 bit must complement on clock pulses when there is a "1" condition in the 2^0 , 2^1 , and 2^2 bits. Its J and K inputs must be enabled when this condition occurs. In general, as shown in the truth table in Figure 5.1, a bit will complement on the count following all less significant bits being a "1". All bits complement simultaneously because of common clock inputs.

EXPERIMENT 5.3: SYNCHRONOUS BINARY UP COUNTER

Figure 5.5 is a four-bit synchronous binary up counter. Construct this circuit on the COMPUTER LAB connecting the Clock and Reset inputs to pulsers, and the 1 outputs of the flip-flops to lamp indicators. Reset the counter to the "0" condition by depressing the Reset pulser once. Obtain a truth table for the operation of the counter by depressing the clock pulser 16 times and recording the output after each clock pulse. Verify that the truth table constructed is identical to the one in Figure 5.1. In order to extend a synchronous binary up counter of this type, the J and K inputs of successively more significant bits will have to be enabled when all less significant bits are equal to 1.

QUESTIONS

5. Design and construct the circuitry necessary to add an extra bit to the counter in Figure 5.5.
6. Design and construct a four-bit synchronous binary down counter. To down count, a bit will complement on the count following all less significant bits being in the "0" condition.
7. Compare the advantages and disadvantages of the two types of counters constructed to this point.

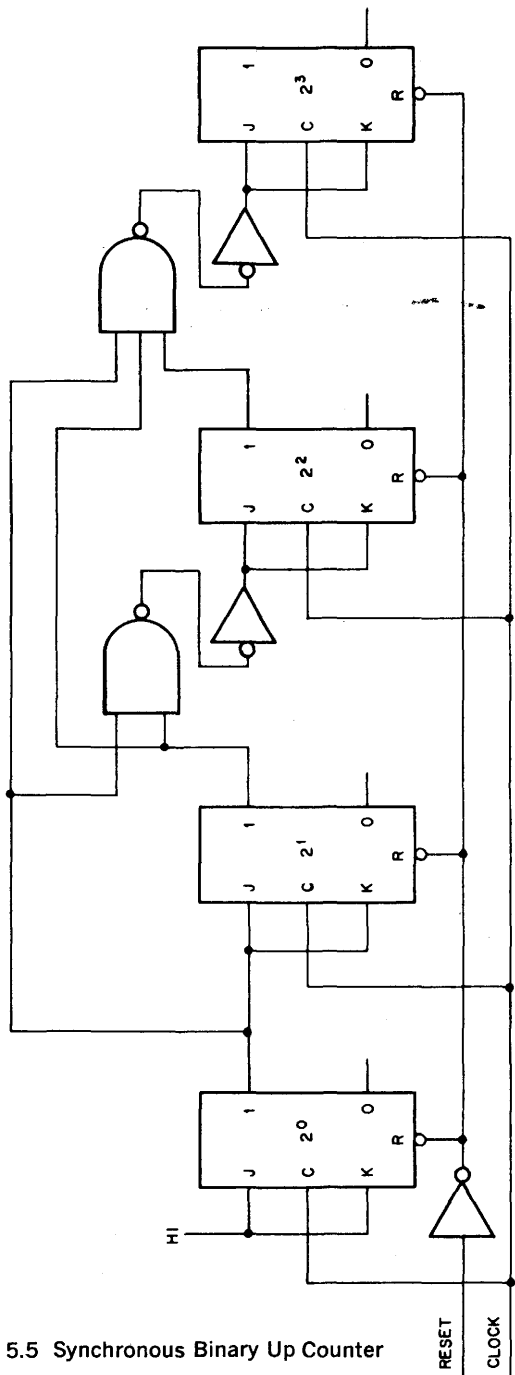


Figure 5.5 Synchronous Binary Up Counter

IV SYNCHRONOUS UP/DOWN COUNTER

In some applications it is necessary for a counter to be able to count in two directions, up or down. A synchronous up/down counter has two separate gating networks. One, to enable it to count up, and another to enable it to count down. If the up counting gating network is enabled, a flip-flop will complement when all less significant flip-flops are in the "1" condition as shown in Figure 5.1. If the down counting gating network is enabled, a flip-flop will complement when all less significant flip-flops are in the "0" condition.

EXPERIMENT 5.4: SYNCHRONOUS BINARY UP/DOWN COUNTER

Figure 5.6 is a synchronous binary Up/Down Counter. Construct the circuit on the COMPUTER LAB, connecting the Up Enable and Down Enable lines to rocker switches, the Clock and Reset inputs to pulser switches, and the 1 outputs of the flip-flops to lamp indicators. Test the circuit in the up counting mode by enabling the up count line with a HI level and disabling the down count line with a LO level, reset the counter to zero, then record the output of the counter after each of 8 successive clock pulses. Test the circuit in the down count mode by enabling the down count line with a HI level, disabling the up count line with LO level, reset the counter, and record the output of the counter after each of 8 successive clock pulses.

QUESTIONS

8. Design a control circuit which will enable the counter to count up when a rocker switch is in the HI position and will enable it to count down when the rocker switch is in the LO position.
9. What will happen if a change of direction is made from up to down or down to up when the clock input is still HI?
10. Show how the up/down counter can be used as a subtractor by resetting the counter, up counting for 6 pulses, and then down counting for 5 pulses. What binary result is obtained from this operation?
11. What happens if both the up enable and down enable lines of the synchronous up/down counter in Figure 5.6 are simultaneously enabled with a HI level when a clock pulse occurs? Why?

V MODULO N COUNTERS

Often a counter is required to count to a specific number and stop. Or the counter may be required to count to a certain number, recycle to 0, and start the count sequence again. The same basic rule applies to the design of these counters as it does to the design of normal synchronous binary counters: detect the state of the counter prior to a clock pulse and selectively enable the J and K inputs of a flip-flop to cause the desired condition following a clock pulse. There are rigorous mapping techniques described in the reference texts in the appendix which can be used to design special counter circuits. Counter design in this chapter will not be rigorous, but will be a step-by-step synthesis of the circuit based on a thorough knowledge of gate and flip-flop functions available on the COMPUTER LAB.

A. Recycling Modulo 6 Counter

A recycling Modulo 6 counter will be used as an example of a recycling counter. Figure 5.7 shows a truth table for a modulo 6 counter. Note how the sequence recycles to 000 after reaching its maximum count of 101. The maximum count of this sequence is equivalent to the decimal number 5. However the counter is rightly called the Modulo 6 counter because there are 6 unique states in this count sequence.

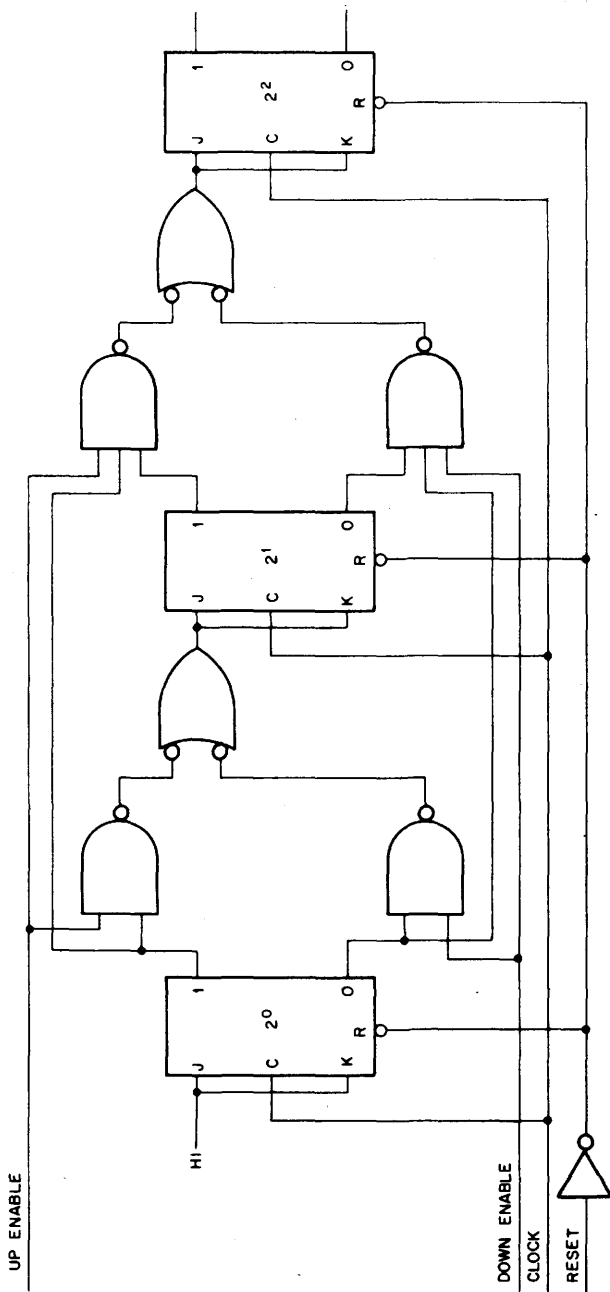


Figure 5.6 Synchronous Binary Up/Down Counter

DECIMAL	2^2 (F.F.A.)	2^1 (F.F.B.)	2^0 (F.F.C.)
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1

Figure 5.7 Modulo 6 Counter Truth Table

There is only one transition (from 101 to 000) where the recycling Modulo 6 count deviates from the binary count sequence. The starting point in the design of a Modulo 6 counter is therefore a 3-bit binary counter as shown in Figure 5.8 a. The Modulo 6 counter in Figure 5.8 b operates the same as the binary counter until the number 101 is reached. At this point the normal binary counter has the J and K inputs of flip-flops B and C enabled and the J and K inputs of flip-flop A disabled. The next clock pulse therefore causes the counter to advance to the number 0. For a Modulo 6 counter, flip-flop C can make the same transition as in the binary counter, following 101. However flip-flop B must remain in the "0" condition and flip-flop A must go to the "0" condition. Flip-flop B therefore has its J and K inputs enabled with a HI level when flip-flop C is in the "1" condition and flip-flop A is in the "0" condition. When the number 101 is reached, flip-flop A is in the "1" condition and flip-flop B will therefore remain in the "0" condition. Flip-flop A must go to "0" after the number 101 is reached. Its K input is therefore enabled with a HI level when flip-flop C is in the "1" condition. Flip-flop A is taken to the "1" condition on the clock pulse following a 1 in both flip-flop A and flip-flop B as in the normal binary counter.

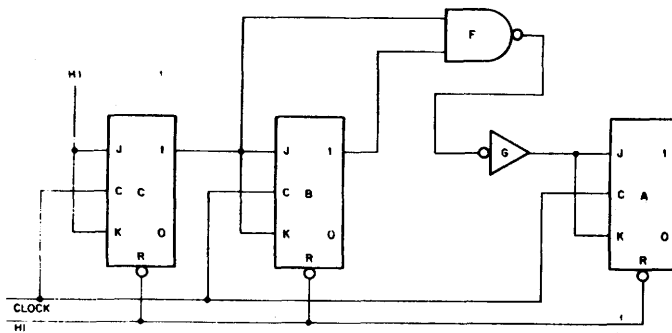


Figure 5.8a Binary Counter

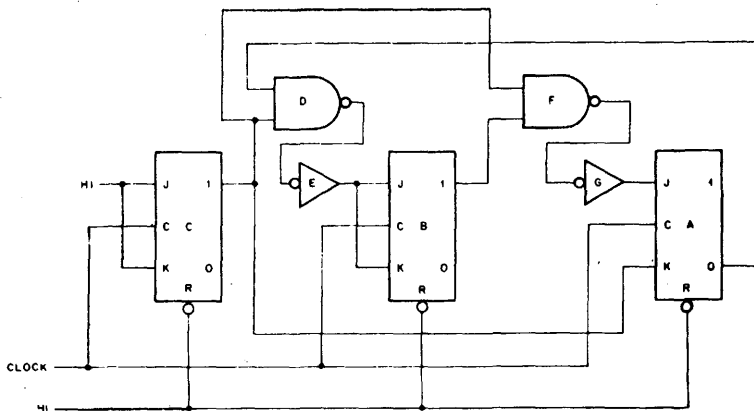


Figure 5.8b Modulo 6 Counter

0 to 5 & back to 0

EXPERIMENT 5.5: SYNCHRONOUS MODULO 6 BINARY COUNTER

Construct the Modulo 6 Counter in Figure 5.8b on the COMPUTER LAB, connecting the Clock input to a pulser switch and all 1 outputs of the flip-flops to lamp indicators. Be sure to disable all flip-flop Reset inputs with a HI level. This will keep unwanted electrical noise from causing the flip-flops to act erratically. Make a truth table showing the count sequence obtained by operating the counter one pulse at a time. Verify that the truth table made is identical to the one in Figure 5.7.

QUESTIONS

12. Design and construct a Modulo 12 counter by extending the Modulo 6 counter already built. Make a truth table to show its operation.
13. Design and construct a Modulo 5 counter which will recycle to 0 after reaching the binary number 100.
14. Extend the Modulo 5 counter to make a Modulo 10 counter.

B. Self-Stopping Counters

Instead of having a counter recycle after a specific number of counts it is sometimes necessary to have it stop and wait for an external reset signal. To make the counter stop, a gating circuit must detect the maximum count and disable the counter. Question 4 showed how the J and K inputs of the least significant bit of an asynchronous counter can disable the counter and cause it not to recognize clock pulses. Figure 5.9 is a self-stopping, asynchronous binary up counter. When this counter reaches the count of 1100, the NAND gate in the circuit will detect the ones in the 2^3 and the 2^2 bits and will disable the J and K inputs of the 2^0 bit and stop the counter. The counter must be reset before further clock pulses are recognized.

EXPERIMENT 5.6: ASYNCHRONOUS SELF-STOPPING MODULO 13 BINARY COUNTER

Construct the self-stopping modulo 13 asynchronous binary counter on the COMPUTER LAB and test to see that it operates as outlined.

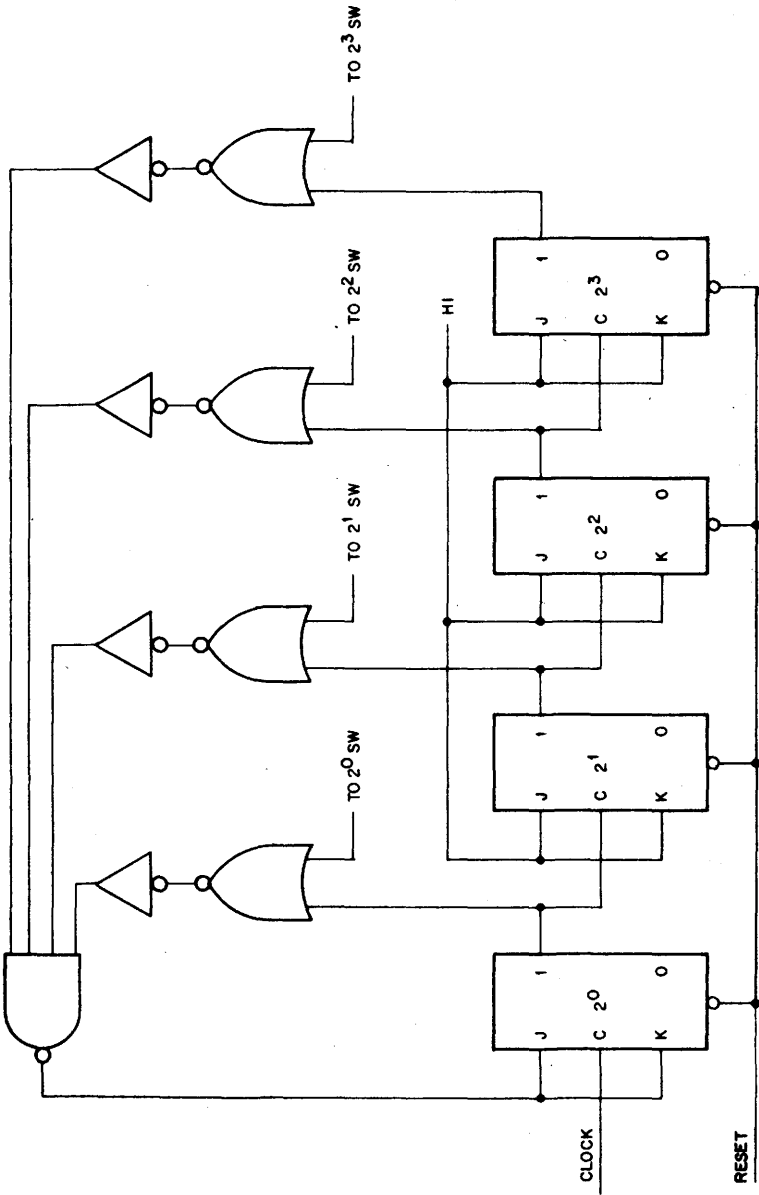


Figure 5.10 Self Stopping Asynchronous Binary Up Counter
with variable modules

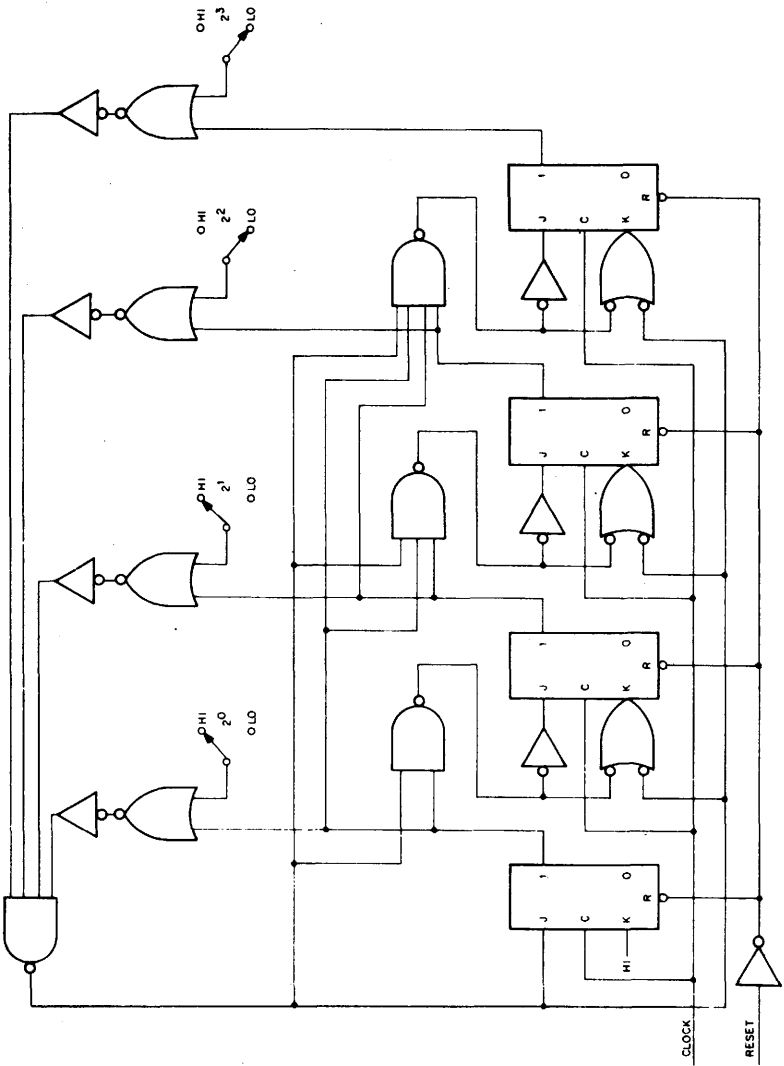


Figure 5.11 Recycling Modulo N Synchronous Binary Up Counter

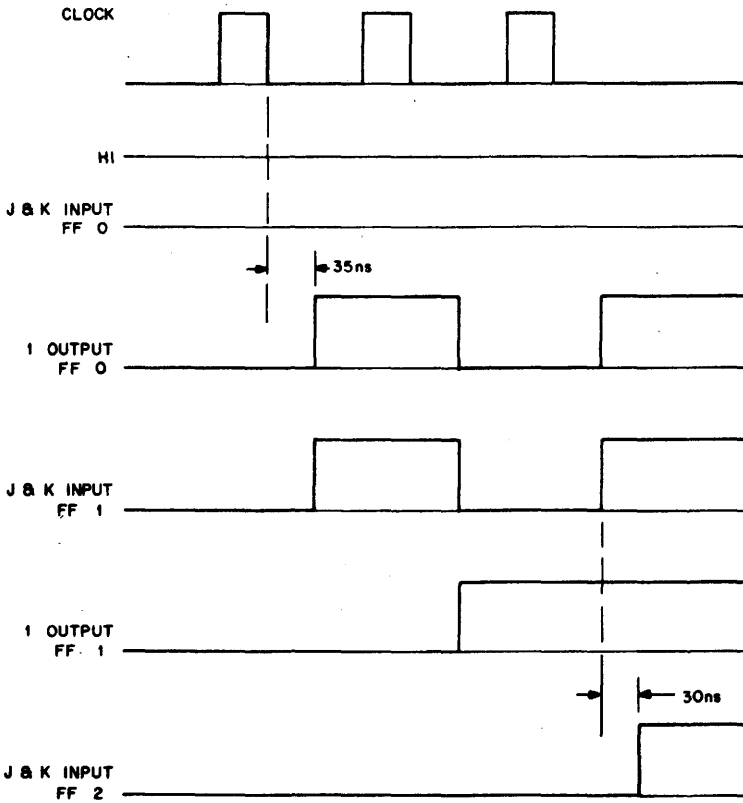


Figure 5.12 Synchronous Binary Up Counter Timing Diagram

NOTE: Propagation delay of an asynchronous counter can be demonstrated visually using a dual trace oscilloscope to look at the output of the first and fourth flip-flops while operating the counter at high clock repetition rates.

20. How much time must be allowed for a change of counting direction in the Synchronous Up/Down Counter described earlier in this chapter? Assume the same propagation delays as given in question 18.
21. Design and construct a self-stopping, synchronous four-bit binary up counter which will stop on a number selected by a four-bit rocker switch register.

CHAPTER 6

SERIAL ADDITION

I BINARY ADDITION

Addition in the binary number system follows the same type of rules as decimal addition. There are four basic rules for addition of two binary numbers;

<u>A</u>	<u>B</u>	=	<u>SUM</u>	<u>CARRY</u>
0	0	=	0	0
0	1	=	1	0
1	0	=	1	0
1	1	=	0	1

In the more general case there are two digits plus a carry digit from the next less significant addition to add together; Figure 6.1 shows addition of three digits with the resultant sum and carry information. The subscript N indicates information pertaining to the 2^N addition.

RULE NO.	CARRY _N	A _N BIT	B _N BIT	SUM _N	CARRY _{N+1}
1	0	0	0	0	0
2	0	0	1	1	0
3	0	1	0	1	0
4	0	1	1	0	1
5	1	0	0	1	0
6	1	0	1	0	1
7	1	1	0	0	1
8	1	1	1	1	1

Figure 6.1 Binary Addition Truth Table

The subscript N+1 under the carry indicates the information carried from the 2^N addition to the 2^{N+1} addition. The following addition example indicates how the truth table can be applied:

NUMBER A	0	1	0	1	1	1	0	0
NUMBER B	0	0	1	1	1	0	1	0
SUM	1	0	0	1	0	1	1	0
RULE APPLIED (REFER TO FIG. 6.1)	RULE 5	RULE 7	RULE 6	RULE 8	RULE 4	RULE 3	RULE 2	RULE 1

II SERIAL ADDITION

The addition process when done manually is serial. That is to say it is done in a step-by-step fashion performing the least significant part of the addition first and then progressively the more significant parts. For instance, in adding the binary numbers 101 and 100, the first step would be to add the digits weighted 2^0 to get a sum of 1, then add the digits weighted 2^1 to get a sum of 0, and finally add the digits weighted 2^2 to get 0 and 1 to carry. The addition is done serially, starting at the least significant digit. Following the same

process a serial adder could be made using a system such as the one in Figure 6.2.

The rectangular blocks which store number A, number B, and the sum are shift registers such as the one constructed in Chapter 3. Two binary numbers can be read into registers A and B. Each register will simultaneously shift all its information one position to the right upon receipt of a shift instruction. The adder performs an addition on one bit at a time as information appears at the output of registers A and B. The adder performs exactly the same functions as is outlined in the truth table in Figure 6.1 producing sum and carry information. The "Sum" register will read whatever information is at its serial input into the I position when a shift pulse is received. When a shift pulse is received by the carry flip-flop it will go to the state dictated by the adder carry output.

To use the adder, numbers would be read into registers A and B and five shift pulses would occur. After the first shift pulse, the sum of the 2^0 addition would appear in position I of the "Sum" register and the carry element would contain the information indicated in the Carry $N+1$ column of the truth table in Figure 6.1. After the second shift pulse the sum of the 2^1 addition would appear in position II of the "Sum" register, the sum of the 2^0 addition would have shifted to position II, and the carry flip flop would contain the information to be stored for the next addition (2^2). At the end of five shift pulses the "Sum" register would contain the sum of number A plus number B. If the sum is greater than 5 bits the carry will be "1" at the end of the addition. If number A and number B are both 4-bit numbers overflow of the sum never occurs.

A more economical design for a serial adder is shown in Figure 6.3; it uses one of the shift registers for two purposes. The upper register labeled accumulator stores one of the numbers to be added and as well as at the end of the addition will contain the sum of the numbers initially in the incident and accumulator registers. The incident register accepts numbers from an outside source. This type of adder has the advantage of being able to add more than two numbers together by accumulating the sum from successive additions. A three bit serial adder of this type can be built on the COMPUTER LAB as shown in Figure 6.4.

Flip-flops A_0 , A_1 and A_2 are the incident register. Flip-flops B_0 , B_1 and B_2 are the accumulator register. Carry information is stored in flip-flop C. Gates 1 through 7 perform the adder function and provide sum information for flip-flop B_2 in the accumulator and carry information for the carry flip-flop C.

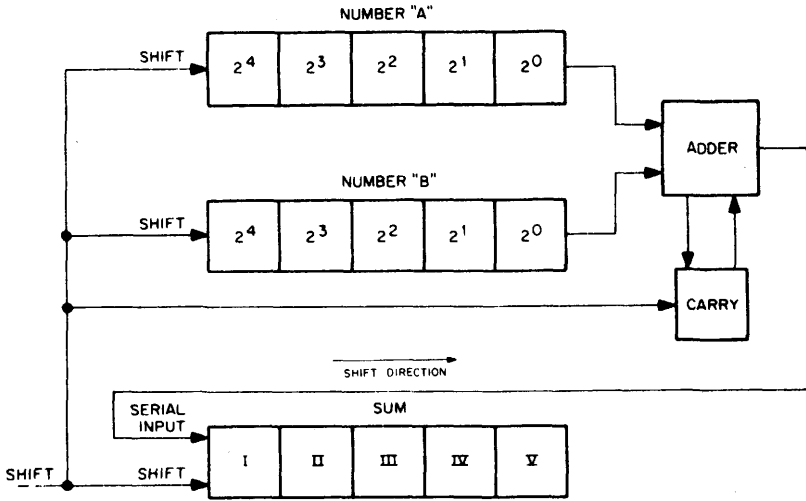


Figure 6.2 Serial Adder Function Diagram

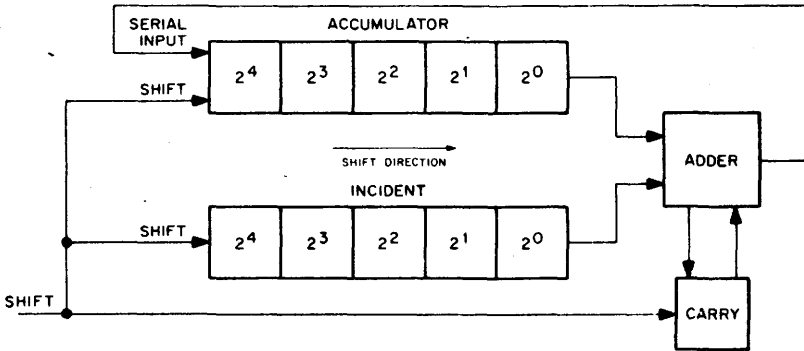
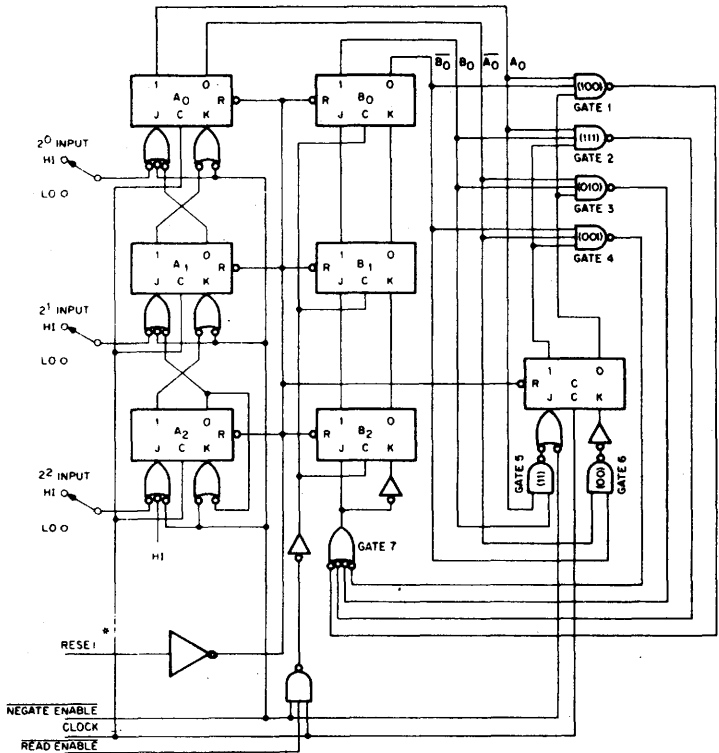


Figure 6.3 Simplified Serial Adder



*This inverter is constructed using two And/Nor Gates with all 8 inputs connected to the reset pulser. One gate drives the reset inputs of A_2 , A_1 , A_0 and the other gate drives the reset inputs of B_2 , B_1 , B_0 and the carry flip flop.

Figure 6.4 Serial Adder

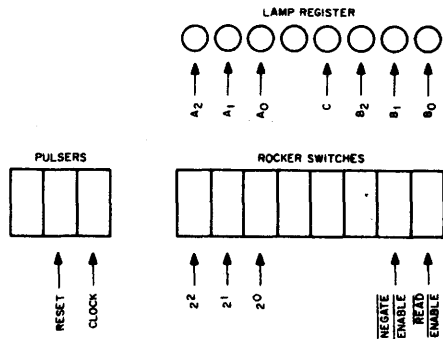


Figure 6.5 Serial Adder Switch and Lamp Usage

EXPERIMENT 6.1: SERIAL ADDER

Construct the serial adder in Figure 6.4 on the COMPUTER LAB. A convenient configuration for lamps and switches is shown in Figure 6.5. The lamp indicator inputs are driven by the "1" outputs of the flip-flops (these lines are not included in Figure 6.4). The RESET, READ ENABLE, and NEGATE ENABLE inputs are rocker switches; the clock input is a pulser switch. The disabled position for all the rocker switches is HI (upper side of rocker depressed).

The adder is used as follows:

1. Clear the incident and accumulator registers and the carry flip-flop by a momentary LO reset signal from the RESET rocker switch. Restore the output of the RESET switch to HI.
2. Make sure that the NEGATE ENABLE switch is in the HI position.
3. Read a number into the incident register by setting the input switch register to the desired number and the READ ENABLE switch LO and then pulsing the clock pulser switch once. A "1" will be read into each flip-flop in the incident register that has its corresponding rocker switch in the LO position. For example to read in 101, the 2^0 and 2^2 inputs would be LO and the 2^1 input would be HI. Restore all switches to the HI position after read in.
4. Pulsing the clock pulser switch three times will shift the number from the incident register to the accumulator register by adding the incident number to the 0's initially in the accumulator.
5. Read a second number into the incident register by the method outlined in step 3.
6. Pulse the clock pulser switch three times to add the numbers in the incident and accumulator registers. The sum will appear in the accumulator register after the addition.
7. Additional numbers can be added to the sum appearing in the accumulator by repeating steps five and six for as many times as there are numbers to add.

Perform the following four additions to test the adder. Follow the steps outlined above and record the output obtained on the lamps after each step in the addition.

Test Additions:

1	100	2	001	3	101	4	011
	<u>010</u>		<u>001</u>		<u>011</u>		<u>011</u>
	110		010		1000		110

QUESTIONS

1. Explain in detail how read-in from the switch register to the incident register takes place.
2. Explain the operation of the incident and accumulator registers as shift registers.
3. Why does the incident register fill with 0's as the information which was initially in it shifts out?
4. The adder circuit (gates 1 to 7) detects conditions giving a sum of 1 (see the truth table Figure 6.1). This information is fed back to the J and K inputs of flip-flop B_2 to cause it to go to the "1" condition on the next clock pulse if the sum is 1 or conversely to go to a "0" if the sum is 0. Design an alternate adder system that detects conditions giving a sum of 0 showing also the method of feeding into the J and K inputs of flip-flop B_2 .

- What conditions cause the carry flip-flop to go to a "1"? What conditions cause the carry flip-flop to go to "0"?
- Why must the carry flip-flop be reset when the incident and accumulator registers are reset? What happens if a "1" is resident in the carry flip-flop when an addition is begun?
- How can the adder be extended to eight bits? What change would have to be made in the operating procedure for an eight-bit adder?

III BINARY SUBTRACTION

A set of rules can be developed for binary subtraction similar to those outlined for binary addition in Figure 6.1. Serial binary subtraction can be performed on a device similar to the adder in Figure 6.4; it would be designed to operate according to the truth table for binary subtraction. However, there is a simpler way to perform a subtraction using addition techniques. If number B must be subtracted from number A, number B can be made negative and added to number A i.e. $A - B = A + (-B)$.

One method of negating a number uses two's complement arithmetic in which negation is accomplished by complementing each bit of a binary number and incrementing or adding 1 to the result. The complement of a number is obtained by changing all the 1's to 0's and all the 0's to 1's. To increment, simply add 1.

Complement each bit:	—00101
Increment:	11010
Equals:	<u>+00001</u>
	11011

Since the sign bit is 1 (11011), the number is negative. $11011 = -00101$

When dealing with binary numbers in two's complement notation, the left-most bit of the number always indicates the sign of that number and is therefore called the sign bit. If the sign bit is 1, the number is negative and if the sign bit is 0 the number is positive. The absolute value of a negative number can be obtained by negating that number.

The serial adder in Figure 6.4 can contain two two-bit numbers plus one sign bit for each number.

Taking an example of $3 - 2$ will demonstrate how two's complement subtraction is performed. The binary number 2 is 010. -2 in two's complement is 110. Adding 011 (3) plus 110 (-2) is equivalent to 3 minus 2. Any carry information at the end of the addition is ignored.

	0	1	1	3
+	1	1	0	-2
	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>

FINAL CARRY IGNORED

EXPERIMENT 6.2: SUBTRACTION USING THE SERIAL ADDER

The serial adder has provision for negating the number in the incident register by causing each bit to complement and setting the carry flip-flop to the "1" state. To do this, take the NEGATE ENABLE switch LO and pulse the clock pulser switch once. Thus the subtraction $3 - 2$ could be performed as

follows: read in 3, shift 3 to the accumulator; read in 2, negate 2, and then add (-2) to 3. Perform the following subtractions on the serial adder:

$$A \ 3 - 2, \quad B \ 3 - 1, \quad C \ 3 - 0, \quad D \ 2 - 3$$

Record the results obtained at each step in each subtraction, noting especially what condition the carry flip-flop goes to when a number is negated.

QUESTIONS

8. Explain in detail how a number in the incident register of the serial adder is negated. Use the number 2 as an example.
9. Explain how the adder can be used to determine the absolute value of a two's complement negative number. Use 110 as an example (110 is a negative number because the sign bit is 1) and test your method using the serial adder.
10. Why can the three-bit adder not be used to subtract 3-bit numbers?

EXPERIMENT 6.3: MULTIPLICATION USING THE SERIAL ADDER

A simple way to do multiplication is to perform successive additions and accumulate a sum. For instance 2×3 is equivalent to $2 + 2 + 2$. These additions could be performed on a serial adder by reading in and adding 2 three times. The incident register can be modified so that it becomes a ring counter and a number initially read into it recirculates in the register and returns to the initial condition after three shift pulses. Modify the serial adder as shown in Figure 6.6. Read in the number 010, disable the READ ENABLE switch by making it HI. Pulse the clock pulser 9 times to perform three successive additions. The accumulator should contain the number 110 and the incident register should still contain 010. The multiplication method is not generally used in computers because it is extremely slow.

QUESTIONS

11. Explain in detail the operation of the incident register with the configuration shown in Figure 6.6.
12. What happens when 5 successive additions of the binary number 010 are attempted? Why?

SUPPLEMENTARY QUESTIONS

13. Make a truth table for binary subtraction similar to the addition truth table in Figure 6.1. The sum column becomes a difference column and the carry column becomes a borrow column.
14. Design a subtractor circuit to subtract the accumulator from the incident register. The carry flip-flop in the adder will of course be equivalent to a borrow flip-flop in the subtractor.
15. How could division be accomplished using a serial adder?

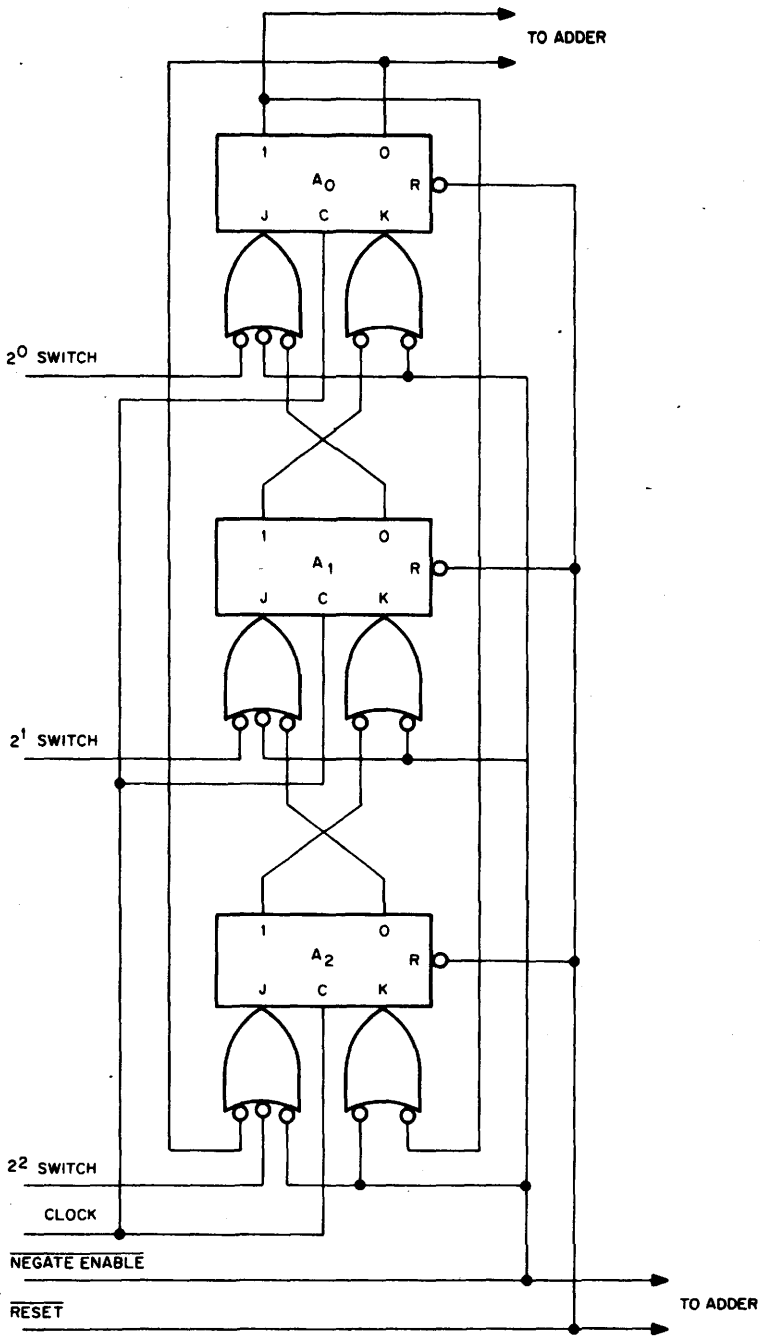


Figure 6.6

CHAPTER 7

PARALLEL ADDITION

I INTRODUCTION

A serial adder is easily expandable and inexpensive, but it is extremely slow for large numbers. If there are two 32-bit numbers to add, the addition must be performed in 32 steps, allowing the adder time to settle between steps. A parallel adder on the other hand can perform the same addition in one or two steps by adding all 32 bits of the number simultaneously.

Since the adder circuitry must be duplicated for each bit in a parallel adder, cost becomes a significant factor in the design. One of the compromises that can be made in the interest of economy is to do additions in two steps instead of in one. Both one and two step adders will be constructed in this experiment.

II TWO STEP PARALLEL ADDER

As with the serial adder the parallel adder has two registers, an incident register which takes a number from the outside and an accumulator register which stores the accumulated sum. Figure 7.1 is a general block diagram showing the organization of a parallel adder. The two step parallel adder has further control of the individual adder elements which allows the half-add and carry steps to be executed independently. The half-add operation is an addition of corresponding bits of two binary numbers neglecting carry information. Half-add causes each bit in the accumulator register to complement when the corresponding bit in the incident register is 1. Figure 7.2 is a truth table showing all possible conditions for corresponding bits in the incident and accumulator registers before the half-add operation and also in the accumulator after half-add.

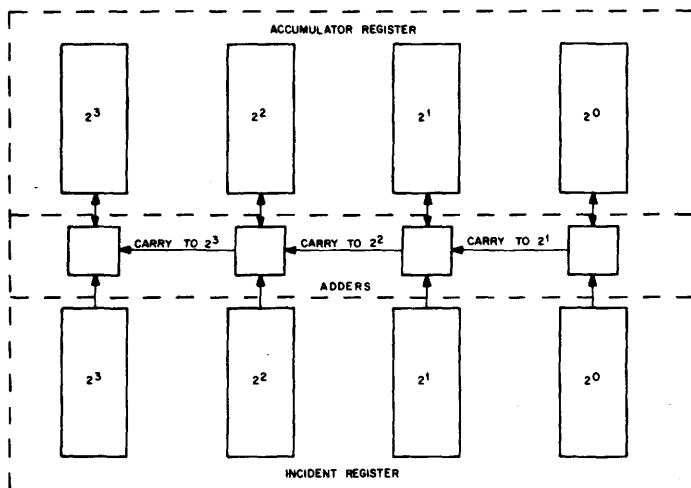


Figure 7.1 Parallel Adder

BEFORE HALF-ADD		AFTER HALF-ADD
INCIDENT BIT	ACCUMULATOR BIT	ACCUMULATOR BIT
0	0	0
0	1	1
1	0	1
1	1	0

Figure 7.2 Half Addition in a Parallel Adder

After half-add operation the carry operation is performed. The carry operation complements each bit in the accumulator which receives a carry of 1 from the next less significant bit. A bit is left unchanged if a carry of 0 is brought forward from the next less significant bit. Figure 7.3 is a truth table showing all eight possible combinations of information that can exist with the carry and corresponding bits of the accumulator and incident registers.

1	2	3	4	5	6
Bit N In Incident Register	Bit N In Accumulator Before Half Add	Bit N In Accumulator After Half Add	Carry To Bit N	Sum In Bit N Of Accumulator After Carry	Carry To Bit N + 1
0	0	0	0	0	0
1	0	1	0	1	0
0	1	1	0	1	0
1	1	0	0	0	1
0	0	0	1	1	0
1	0	1	1	0	1
0	1	1	1	0	1
1	1	0	1	1	1

Figure 7.3 Parallel Adder Operation Truth Table

Columns 1 and 2 indicate possible conditions of bits in the incident and accumulator registers prior to half-add. Column 3 indicates the accumulator condition after the half-add. Column 4 indicates the carry information available to Bit N from Bit N-1 after half-add. Column 6 indicates the carry information Bit N must generate for Bit N + 1 by examining the carry from Bit N-1, the incident bit, and the accumulator bit (columns 1, 3 & 4) after half-add. Column 5 is the sum of the incident bit plus the accumulator bit plus the carry bit which must appear in the accumulator after the carry operation.

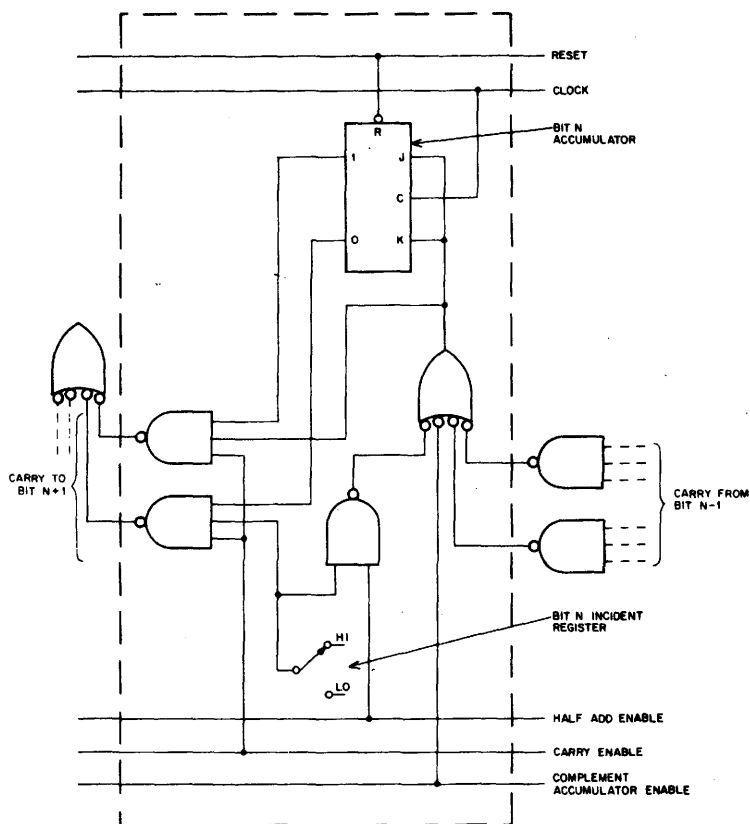


Figure 7.4 Two Step Parallel Adder

Figure 7.4 is one bit of a two step parallel adder. The accumulator bit is a flip-flop and the incident bit is provided by a switch (or a flip-flop). HALF ADD ENABLE and CARRY ENABLE lines are enabled by a HI level and disabled by a LO level. COMPLEMENT ACCUMULATOR ENABLE is enabled by a LO level and disabled by a HI level. An enabled function will be executed on the clock pulse following the enable level. Only one line can be enabled at a time and the enable information must remain during the clock pulse which executes the function enabled. Resetting is caused, as before, by a momentary LO signal on the RESET line. The incident bit is "1" if the switch is in the HI position and conversely is "0" if the switch is in the LO position. Read-in to the accumulator register can be accomplished by resetting, then half-adding the desired number from the incident register.

QUESTIONS

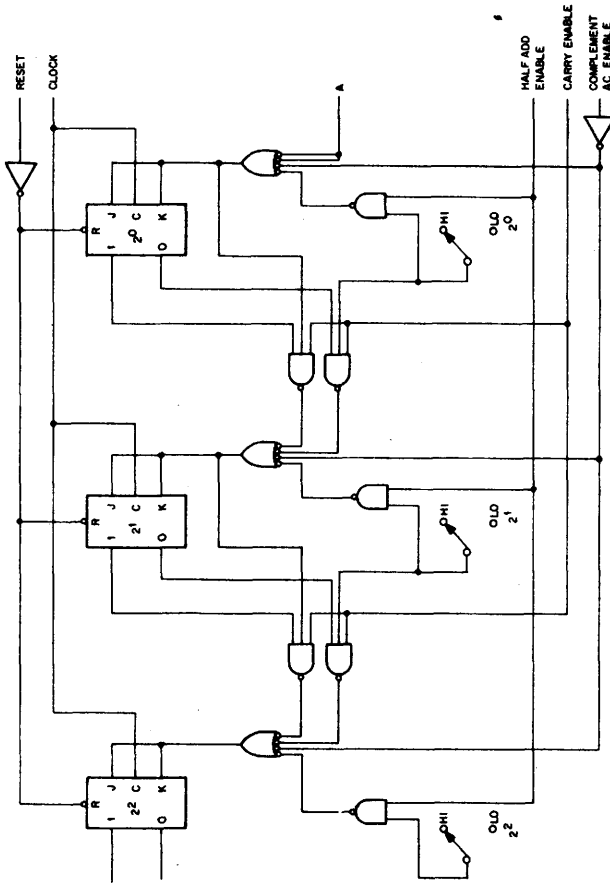
1. Explain in detail how the half-add function takes place in the adder in Figure 7.4:

- A if the incident bit is 1
- B if the incident bit is 0

2. Write a Boolean equation to indicate the operation of the carry circuitry in Figure 7.4.

EXPERIMENT 7.1: PARALLEL ADDER

Figure 7.5 is a three-bit parallel adder. Construct the circuit on the COMPUTER LAB using the switches and lamps as shown in Figure 7.6. The adder is used as shown in figure 7.6. Lamps are connected to the 1 output of flip-flops.



NOTE: Input A should be connected to a HI level unless it is being used for subtraction.

Figure 7.5 Three-Bit Parallel Adder

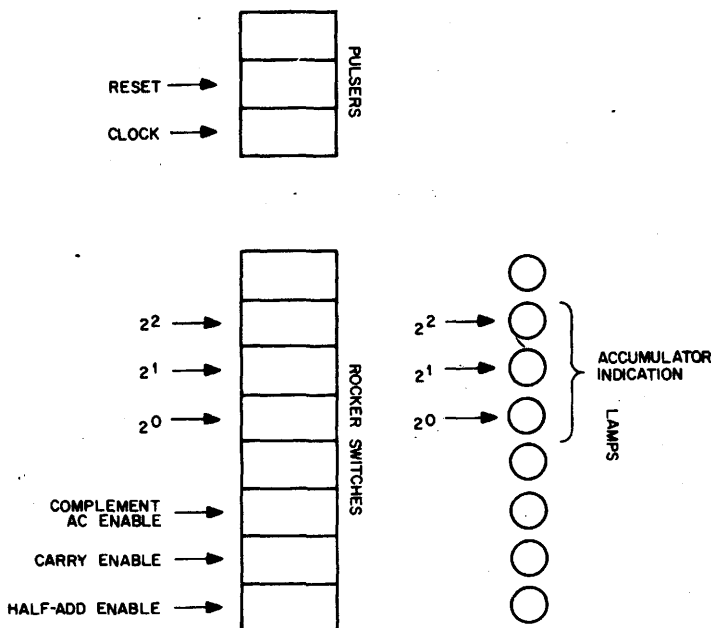


Figure 7.6 Lamp And Switch Usage For Parallel Adder

1. Depress the RESET pulser once to clear the accumulator.
2. Set a number on the incident register. Recall that a "1" is HI (upper side of rocker switch depressed) and a "0" is LO. The binary number 101 would be set in by a HI on the 2^0 and 2^2 inputs and a LO on the 2^1 input.
3. Read the number into the accumulator by putting the HALF-ADD ENABLE switch in the HI position and depressing the clock pulser once. Make sure that the other two function enabling switches are in the disable position (LO) before pulsing the clock switch.
4. Set the second number to be added into the switch register.
5. Half-add by putting the HALF-ADD ENABLE switch in the HI position and depressing the clock pulser once. Again be sure that half-add is the only function enabled when the clock pulser is depressed. After half adding return the HALF-ADD ENABLE switch to the LO position.
6. Carry by putting the CARRY ENABLE switch in the HI position and depressing the clock pulser once. Carry must be the only function enabled when the clock pulse occurs. The sum will now appear in the accumulator.
7. More numbers can be added to the sum in the accumulator by repeating steps 4 through 6 for as many times as there are numbers to add.

Test the adder by performing the following four additions by the method outlined on page 91.

$$\begin{array}{r} \text{A} \quad 100 \\ \quad 010 \\ \hline \quad 110 \end{array}$$

$$\begin{array}{r} \text{B} \quad 001 \\ \quad 001 \\ \hline \quad 010 \end{array}$$

$$\begin{array}{r} \text{C} \quad 011 \\ \quad 011 \\ \hline \quad 110 \end{array}$$

$$\begin{array}{r} \text{D} \quad 101 \\ \quad 011 \\ \hline \quad 1000 \end{array}$$

QUESTIONS

- The serial adder in Chapter 6 had a carry flip-flop which was useful for determining if the accumulator had overflowed (i.e. the final sum was larger than the capacity of the accumulator). The adder in Figure 7.5 has no provision for overflow and is consequently incapable of doing additions such as D of the set of test additions. Design and construct an overflow circuit using one of the spare J-K flip-flops.
- How much time is required to perform a three-bit addition after two of the numbers are present in the incident and accumulator registers? Keep in mind that after the carry is enabled, carry information must have time to propagate from the least significant to the most significant bit (e.g. adding 001 plus 111, carry propagates from the 2^0 addition to the 2^2 addition). How long would a 32-bit addition take? Assuming:
 - 15 ns delay per NAND gate
 - 35 ns propagation delay for J-K flip-flops (after trailing edge of clock pulse)
 - 50 ns clock pulse width
- What is the purpose of the inverters in the COMPLEMENT AC ENABLE and RESET lines? (AC is short for accumulator).

III SUBTRACTION USING THE TWO STEP PARALLEL ADDER

EXPERIMENT 7.2: TWO'S COMPLEMENT SUBTRACTION

Two's complement subtraction can be performed by complementing the accumulator and incrementing using the carry input (A) to the 2^0 flip-flop. When the COMPLEMENT AC ENABLE switch is in the HI position a clock pulse will complement the information in the accumulator regardless of the state of the incident register. Incrementing can be achieved by providing a LO level at point A when the carry takes place.

QUESTIONS

- Negating a number by the techniques described above is rather awkward as there are two steps which have to be performed at separate times. Design and construct a control circuit that will do the following:
 - Accept a start instruction when the accumulator is complemented and remember that instruction.
 - When the carry is enabled after a start instruction a LO level will be presented at point A (the carry input to the 2^0 bit).
 - When the clock pulse is received to execute the carry instruction, the control will reset and await another start instruction.

When the control circuit is added, the COMPLEMENT ENABLE becomes NEGATE ENABLE. Make sure the added control doesn't interfere with the addition function of the adder.

- Explain how the adder can now be used to determine the absolute value of a negative number in two's complement form.

EXPERIMENT 7.3: ONE'S COMPLEMENT SUBTRACTION

There is a second method of negating a binary number using one's complement arithmetic. To negate a number in one's complement the only operation required is to complement each bit in that number.

$$- 01001 = + 10110$$

The first bit in a one's complement binary number is, as in two's complement, a sign bit. A sign bit of 1 indicates a negative number and a sign bit of 0 indicates a positive number. A one's complement subtraction has one extra step called end around carry which increments the final answer if any carry information is left over from the sign bit addition. The following example demonstrates one's complement subtraction.

3	011		011
<u>-2</u>	<u>-010</u>	A) Negate	<u>+101</u>
1		B) Add	000
		C) End Around	<u>+ 1</u>
		Carry	001

QUESTIONS

8. Design and construct the additional circuitry necessary to perform end around carry on the parallel adder.
9. What restriction must be put on additions in a parallel adder with the end around carry facility?

Perform the following test subtractions with the parallel adder using one's complement subtraction:

$$\text{A } 3 - 2, \quad \text{B } 3 - 1, \quad \text{C } 3 - 0, \quad \text{D } 2 - 3$$

QUESTIONS

10. Describe a method of subtracting the incident number from the number in the accumulator.
11. Which system of subtraction do you consider to be the better, one's complement or two's complement? In your answer discuss the relative advantages and disadvantages of both systems from mathematical and circuit design points of view.

EXPERIMENT 7.4: ONE STEP PARALLEL ADDITION

Single step parallel addition is used in computer applications where speed is of prime importance. The single step parallel adder performs an addition in one clock pulse and is therefore in most cases twice as fast as a comparable length two step parallel adder. Figure 7.7 is one bit of a single step parallel adder. Construct the one bit of the adder on the COMPUTER LAB connecting the C_{N+1} output and flip-flop 1 output to lamp drivers. Connect the A_N and C_N inputs to rocker switches and the clock input to a pulser switch. The carry lines, C_N and C_{N+1} and the A_N input are in the "1" condition if HI, and are in the "0" condition if LO. Verify that the circuit operates as an adder by checking the C_{N+1} output for all 8 possible initial conditions and also checking the sum left in the flip-flop after a clock pulse for all 8 possible initial conditions.

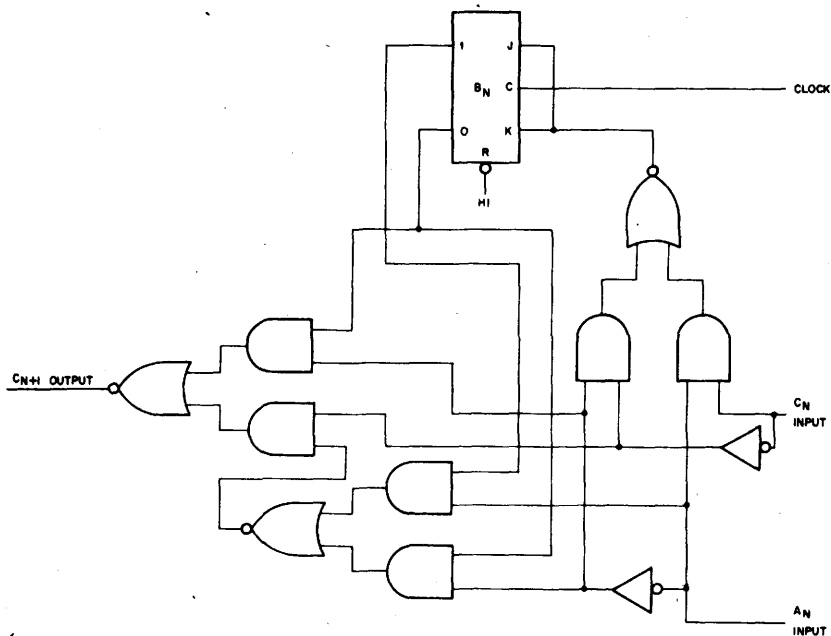


Figure 7.7 One Step Parallel Adder

QUESTIONS

12. How much time must be allowed for a 12-bit one step parallel adder to settle before the add clock pulse can occur? What rate could additions be performed at in a 12-bit one step adder? Assuming:
 - A. NAND and AND/NOR gates propagate information in 15 ns.
 - B. J-K flip-flops propagate 35 ns after the clock pulse trailing edge.
 - C. Clock pulses are 50 ns wide.
13. Define the operation of the adder by determining what Boolean functions are generated by the adder for C_{N+1} and for the J and K inputs of the flip-flop.

SUPPLEMENTARY QUESTIONS

14. Design and construct a one-bit, one step parallel subtractor to subtract the incident number from the accumulator number.
15. Design an alternate circuit for a one step parallel adder not using AND/NOR gates. What is the minimum number of NAND gates possible?

CHAPTER 8

BINARY CODED DECIMAL OPERATIONS

I INTRODUCTION

All the electronic circuits, studied to this point, worked with two states or conditions and were therefore used to perform operations using binary or two-state arithmetic. Decimal arithmetic is more common and more widely used and it is therefore often advantageous to be able to encode a decimal number into a binary form, operate on it in binary form in a computer and then decode it back into decimal form.

There are a variety of codes available to do this. The relative merits and disadvantages of the various codes will be discussed in Chapter 9. Figure 8.1 gives the count sequences for 6 of the more common BCD (binary coded decimal) codes. Each of the codes in figure 8.1 uses 4 bits to encode one decimal digit. Four bits must be used to encode the ten unique states of the decimal number system. In this experiment, counters for 4 of the codes will be built and methods for addition and subtraction will be shown.

DECIMAL	8421	EXCESS 3	2421
0	0000	0011	0000
1	0001	0100	0001
2	0010	0101	0010
3	0011	0110	0011
4	0100	0111	0100
5	0101	1000	1011
6	0110	1001	1100
7	0111	1010	1101
8	1000	1011	1110
9	1001	1100	1111
DECIMAL	5421	5311	74(-2)(-1)
0	0000	0000	0000
1	0001	0001	0111
2	0010	0011	0110
3	0011	0100	0101
4	0100	0101	0100
5	1000	1000	1010
6	1001	1001	1001
7	1010	1011	1000
8	1011	1100	1111
9	1100	1101	1110

Figure 8.1

II BCD COUNTERS

EXPERIMENT 8.1: THE 8421 COUNTER

The 8421 code follows the binary count sequence up to the binary number 1001 and then recycles to 0000. The basis of the 8421 counter is therefore a binary counter which has a breakpoint in its count sequence at 1001. Figure 8.2 is a synchronous 8421 BCD up counter. Construct it on the COMPUTER LAB. Connect the 1 outputs of the flip-flops to lamp indicators and the

clock and reset inputs to a pulser switch. Test it by advancing the counter one pulse at a time and recording the output after each pulse to verify that it has the same count sequence as indicated in Figure 8.1.

QUESTIONS

1. Describe how the counter in Figure 8.2 operates noting in particular how the transition from 1001 to 0000 takes place.
2. Construct a second 8421 decade and design and construct a circuit to advance it on each clock pulse following the number 1001 being in the first 8421 decade. Make sure both decades operate synchronously.
3. Design and construct an asynchronous 8421 up counter and an asynchronous decade cascading circuit.

EXPERIMENT 8.2: THE EXCESS 3 COUNTER

The Excess 3 code also follows the binary sequence and has only one breakpoint. An Excess 3 number is formed by adding 3 to the binary representation of the number to be encoded, therefore no weight can be assigned to each bit. Figure 8.3 is a synchronous Excess 3 up counter. Construct it on the COMPUTER LAB connecting flip-flop 1 outputs to lamp indicators and the clock and reset inputs to pulsers. Test the counter by advancing it one pulse at a time recording the output after each pulse.

QUESTIONS

4. Explain how the Excess 3 counter operates noting especially how the 1100 to 0011 transition occurs.
5. Simplify the circuit in Figure 8.3 by removing two redundant gates.
6. Design a circuit to synchronously cascade a second Excess 3 decade.
7. Design and construct a synchronous Excess 3 down counter.

EXPERIMENT 8.3: THE 2421 COUNTER

The 2421 code again follows the binary sequence with one breakpoint at the code number 0100 (decimal 4). Each position in the 2421 code has a numerical weight and the decimal value of a 2421 number can therefore be determined by summing the weights of each bit which is a 1. Figure 8.4 is a 2421 synchronous up counter. Construct it on the COMPUTER LAB connecting 1 outputs from flip-flops to lamp indicators and the clock and reset inputs to pulser switches. Test the operation of the counter by advancing it one pulse at a time and recording the output after each pulse.

QUESTIONS

8. Simplify the 2421 counter by removing two redundant gates from the circuit.
9. Design and construct a two decade asynchronous 2421 up counter.
10. How soon after the last clock pulse can an accurate read-out be obtained from the two decade asynchronous 2421 counter?

EXPERIMENT 8.4: THE 5421 COUNTER

The 5421 code follows the binary sequence with two breakpoints, one between 0100 and 1000 and another between 1100 and 0000. Surprisingly, the 5421 counter is the simplest BCD counter to construct. Figure 8.5 is a synchronous 5421 up counter. Construct it on the COMPUTER LAB connecting flip-flop 1 outputs to lamp indicators and the clock and reset inputs to pulsers. Test the counter by operating it one pulse at a time and recording the output obtained after each pulse.

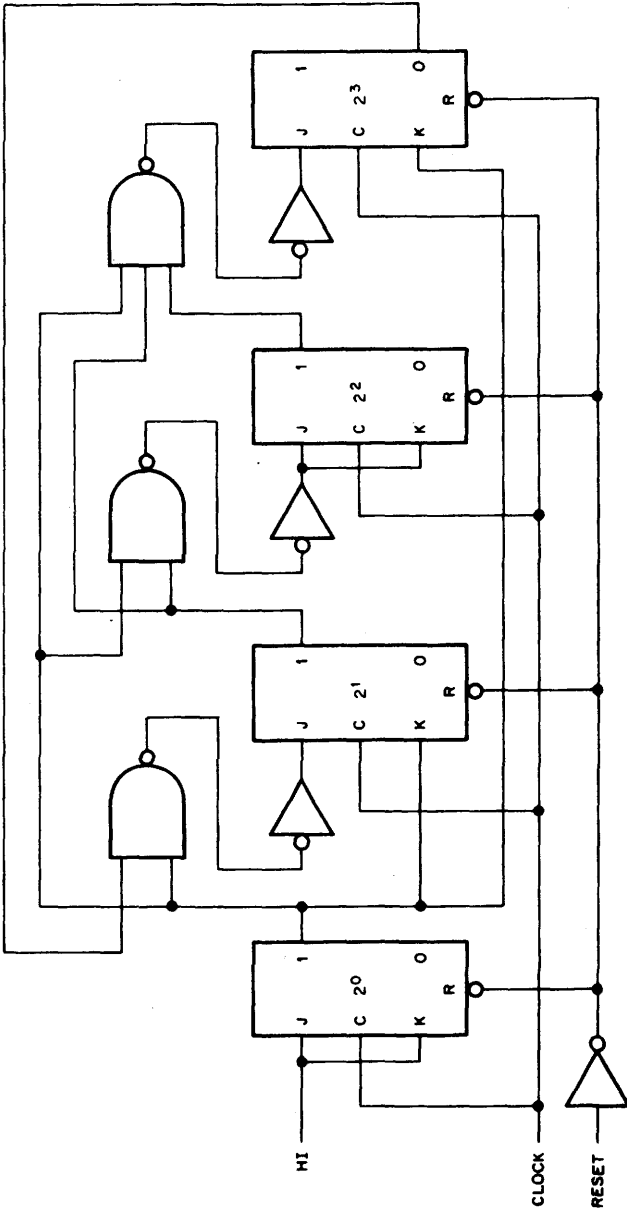


Figure 8.2 Synchronous 8421 BCD Counter

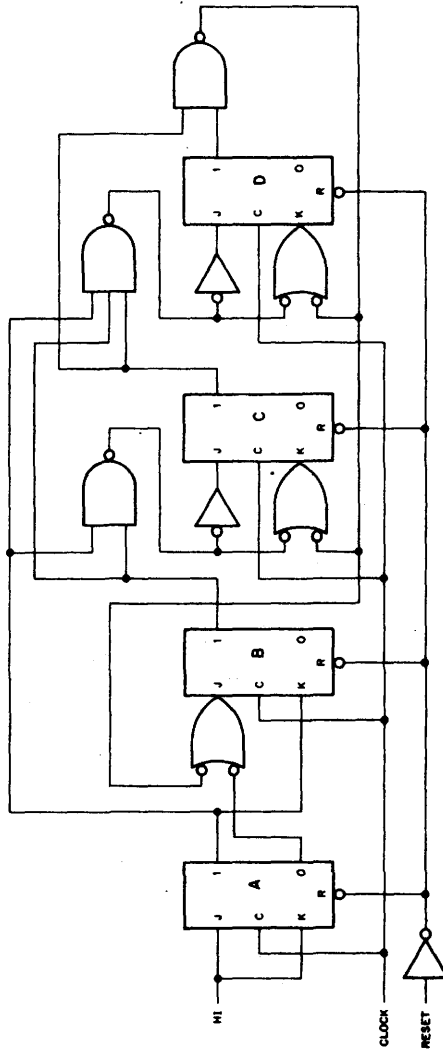


Figure 8.3 Synchronous Excess 3 Code Up Counter

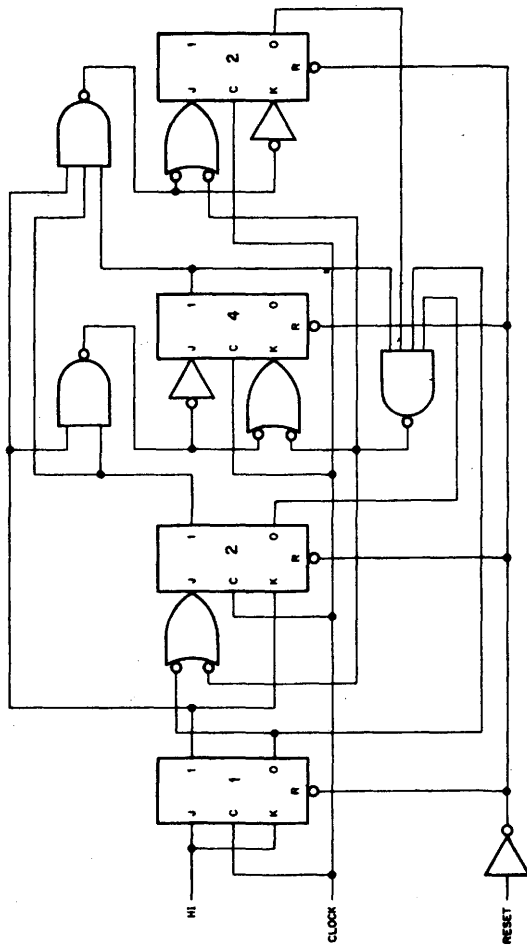


Figure 8.4 2421 BCD Counter

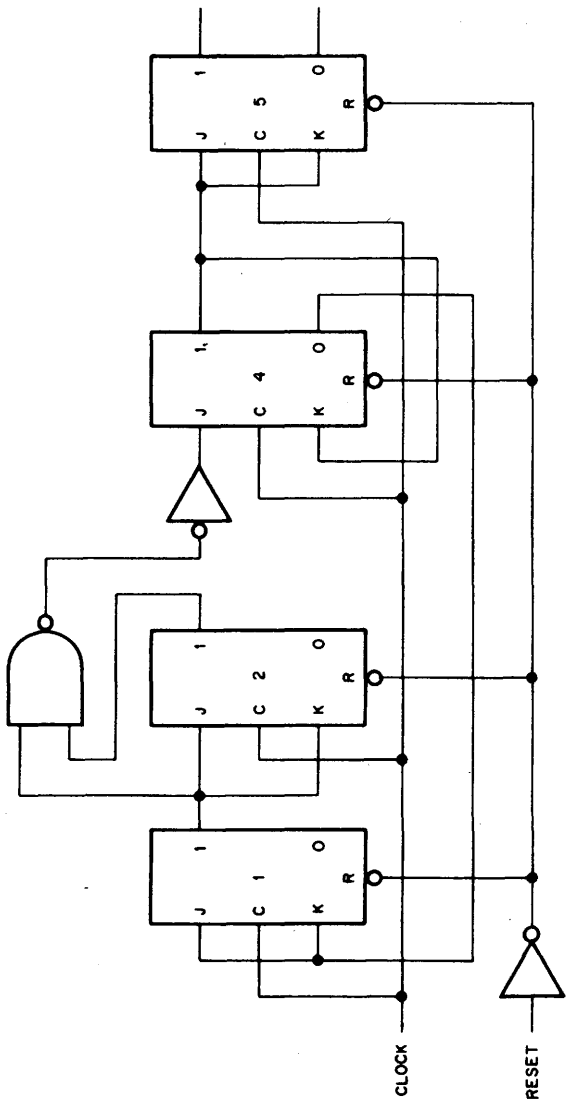


Figure 8.5 5421 BCD Counter

QUESTIONS

11. Explain how the counter operates, in particular how transitions are made at the two break points.
12. Design and construct a synchronous 5421 up/down counter.

III BCD ADDITION

Because of the binary format most BCD codes can be added in a binary adder. If necessary a correction factor can be added after the binary addition. For instance two decimal digits in 8421 code could be added in binary fashion and if the sum of the two digits were greater than 9 a correction factor of 6 would be added to that decade, e.g.:

A.
$$\begin{array}{r} 5 \\ +3 \\ \hline 8 \end{array} \qquad \begin{array}{r} 0101 \\ +0011 \\ \hline 1000 \end{array} \quad \text{Less than 9 no correction factor necessary}$$

B.
$$\begin{array}{r} 8 \\ +7 \\ \hline 15 \end{array} \qquad \begin{array}{r} 1000 \\ 0111 \\ \hline 1111 \\ 0110 \\ \hline 0101 \end{array} \quad \text{Add correction factor}$$

 0001 Carry to 10¹ decade

Excess 3 addition is also performed in binary fashion. If the decade sum is less than 9 a correction factor of -3 is subtracted, and if the sum is greater than 9 a correction factor of +3 is added, e.g.:

A.
$$\begin{array}{r} 5 \\ +3 \\ \hline 8 \end{array} \qquad \begin{array}{r} 1000 \\ 0110 \\ \hline 1110 \\ -0011 \\ \hline 1011 \end{array} \quad \text{Correction factor } -3$$

B.
$$\begin{array}{r} 8 \\ +7 \\ \hline 15 \end{array} \qquad \begin{array}{r} 0011 \\ 0011 \\ \hline 0111 \\ -0011 \\ \hline 0100 \end{array} \qquad \begin{array}{r} 1011 \\ 1010 \\ \hline 0101 \\ +0011 \\ \hline 1000 \end{array} \quad \text{Correction factors}$$

EXPERIMENT 8.5: SERIAL BCD ADDITION (SUPPLEMENTARY)

Figure 8.6 is a five-bit serial adder which could be used to add two binary coded decimal digits. The correction detect circuit indicates whether or not the sum of the two decimal digits added is greater than 9 and thereby what form of correction is required. The adder design is the same in Figure 8.6 as the serial adder in Experiment 6.1. Construct a five bit serial adder on two COMPUTER LABs wiring the incident register on one COMPUTER LAB and the remainder of the circuit on the other.

When using two or more COMPUTER LABs together establish a common ground by connecting a ground wire from one COMPUTER LAB to another using the ground terminals at the bottom corners of the patchboards. If more than two COMPUTER LABs are used for a complex circuit, connect all ground lines out from one central patchboard, to avoid ground loops.

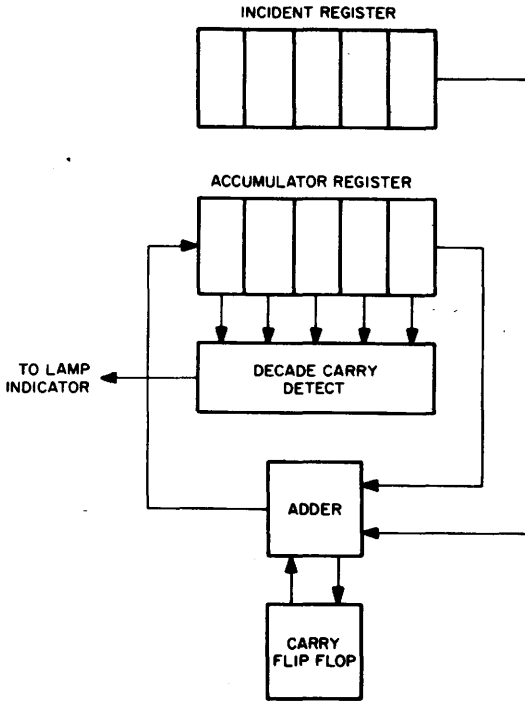


Figure 8.6 Single Decade BCD Serial Adder

If two COMPUTER LABs are not available, use one of the rocker switches and an inverter to simulate the output of the last flip-flop of the incident register. Change the switch after each clock pulse to the condition that would have been present at the output of the incident register at that time. For example, if the number to be added is 00101, initially the switch presents a 1 to the adder; after the first clock pulse it is changed to a 0, after the second clock pulse it is changed to a 1, after the third to a 0, and so on.

QUESTIONS

- Design and construct a decade carry detect circuit for 8421 additions to determine if a correction factor must be added.
- Perform the following 8421 additions, recording the results before and after adding the correction factor. Does your decade carry detect circuit detect sums over 9?

$$\begin{array}{r} 7 \\ +2 \\ \hline \end{array}$$

$$\begin{array}{r} 5 \\ +1 \\ \hline \end{array}$$

$$\begin{array}{r} 8 \\ +6 \\ \hline \end{array}$$

$$\begin{array}{r} 9 \\ +8 \\ \hline \end{array}$$

When does carry to the next most significant decade occur?

- Design and construct a decade carry detect circuit for Excess 3 additions to determine which correction factor must be added, +3 or -3.
- Perform the following Excess 3 additions recording the results before and after adding the correction factor.

Does your decade carry detect circuit detect sums over 9?
When does carry to the next most significant decade occur?

$$\begin{array}{r} 7 \\ +2 \\ \hline \end{array}$$

$$\begin{array}{r} 5 \\ +1 \\ \hline \end{array}$$

$$\begin{array}{r} 8 \\ +6 \\ \hline \end{array}$$

$$\begin{array}{r} 9 \\ +8 \\ \hline \end{array}$$

PARALLEL BCD ADDITION

Parallel addition can be performed using Excess 3 code in a normal binary parallel adder such as the one in Chapter 7. One extra flip-flop per decade is required to determine when a carry occurs from one decade to another. If a carry occurs the sum of the two decimal digits was greater than 9 and a correction factor of +3 should be added and if no carry occurs the sum is less than 9 and a correction factor of -3 should be added. Figure 8.7 is a functional block diagram of an Excess 3 parallel adder. The four-bit adder sections of the circuit can be one or two step adders.

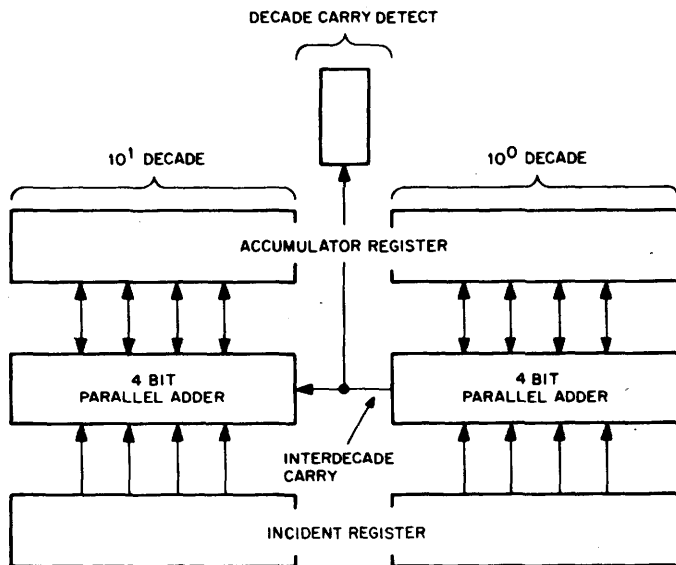


Figure 8.7 Two Decade Excess 3 Parallel Adder

QUESTIONS

- Design a circuit which would detect an interdecade carry in a two step adder like the one in Chapter 7.
- Design a single step two decade Excess 3 parallel adder.

IV BCD SUBTRACTION

Binary coded decimal numbers can be subtracted using 9's or 10's complement arithmetic in the same way as 1's and 2's complement is used for binary arithmetic. The 9's complement of a decimal number is made by subtracting each digit individually from 9; i.e., $-028 \Leftrightarrow 971$ (9's complement

form). Just as a 1's or 2's complement binary number has a sign bit, a 9's or 10's complement decimal number has a sign digit.

If the sign digit is 9 the number is negative. If the sign digit is 0, the number is positive. The rules for 9's complement decimal addition are identical to those for 1's complement binary addition, namely: add and then end around carry. For example, to subtract 28 from 35, the 9's complement arithmetic would be as follows:

$$\begin{array}{r}
 035 \qquad \qquad 035 \qquad \qquad \text{9's complement subtraction} \\
 -028 \rightarrow \text{Negate} \rightarrow +971 \\
 \hline
 \qquad \qquad \qquad 006 \\
 \text{end around carry } 1 \\
 \hline
 \qquad \qquad \qquad 007
 \end{array}$$

10's complement decimal subtraction is analogous to 2's complement binary subtraction. The 10's complement negative of a number is determined by subtracting each digit from 9 and then incrementing the number obtained. As with 2's complement any carry left from the most significant digit addition is discarded.

SUPPLEMENTARY QUESTIONS

19. Perform the following decimal subtractions using:

- a) 9's complement technique and
- b) 10's complement technique

A.
$$\begin{array}{r} 045 \\ -032 \\ \hline \end{array}$$

B.
$$\begin{array}{r} 089 \\ -036 \\ \hline \end{array}$$

C.
$$\begin{array}{r} 031 \\ -025 \\ \hline \end{array}$$

20. Write the number 46 in Excess 3 notation. Determine the 9's complement form of -46 and write -46 in Excess 3, 9's complement form without the sign digit. What rule can be made for negating an Excess 3 coded number by 9's complement arithmetic?
21. Write the number 30 and the 10's complement representation of -30 in Excess 3 form. What rule can be made for negating an Excess 3 number using 10's complement arithmetic?
22. Design a single decade Excess 3 parallel adder with provision for negating the number in the accumulator to do 9's complement subtraction. Be sure to include an end around carry facility in your design.
23. Why is the 8421 code not suitable for parallel addition techniques?
24. Negate the numbers 46 and 30 using:
 - a) 10's complement arithmetic with 8421 code
 - b) 9's complement arithmetic with 8421 code.
 Which code is more convenient for BCD arithmetic operations 8421 or Excess 3?

CHAPTER 9

CODE CONVERSION AND DECODING

I INTRODUCTION

In the preceding experiment so many different types of codes have been introduced that the question, "why such a variety of codes?" might arise. Many of the codes in use today have been developed for very good reasons, like combatting noise in intercomputer communications or accommodating more keys on a teletype keyboard, but others are surviving more through habit than anything else. This experiment covers most of the important codes in use today and their relative merits will be discussed in the light of current technology.

For purposes of discussion, codes will be grouped into three main classifications: decimal, reflective codes, error-detecting codes and miscellaneous codes.

II DECIMAL CODES

To build a decimal computer with 2-state devices, it is necessary to encode each decimal digit with binary bits. Four binary bits are needed. Although only 10 of the 16 permutations possible with the 4-bit BCD decade will be used, all are available. The number of codes that can be generated is calculated as follows:

$$16 \times 15 \times 14 \times 13 \times 12 \times 11 \times 10 \times 9 \times 8 \times 7 = \frac{16!}{6!} = 2.9 \times 10^{10}$$

The choice of a code is obviously important. Desirable features of a code are: ease in performing arithmetic operations, economy of storage space, economy of gating operations, error detection and correction, and simplicity.

The 8421 code is commonly referred to simply as binary-coded decimal because the weights of the positions are the same as in the binary number system. Arithmetic operations are easily performed using the same basic method as in binary since the number sequence is the same.

However, the 8421 code is only one of almost thirty billion possibilities. Logic designers have experimented with some of the others to see if they would be useful. Only six of them are used to any extent at all, and these six are listed in the table below.

Decimal	8421	Excess 3	2421	Decimal	5421	5311	74-2-1
0	0000	0011	0000	0	0000	0000	0000
1	0001	0100	0001	1	0001	0001	0111
2	0010	0101	0010	2	0010	0011	0110
3	0011	0110	0011	3	0011	0100	0101
4	0100	0111	0100	4	0100	0101	0100
5	0101	1000	1011	5	1000	1000	1010
6	0110	1001	1100	6	1001	1001	1001
7	0111	1010	1101	7	1010	1011	1000
8	1000	1011	1110	8	1011	1100	1111
9	1001	1100	1111	9	1100	1101	1110

Figure 9.1

In the Excess 3 code, a decimal digit D is represented by the binary equivalent of the digit D plus 3. The Excess 3 code is not a weighted code, but since it follows the same number sequence as binary, it is useful in arithmetic operations. Addition is facilitated since the need for a correction factor is easily detected and easily implemented. Because it is self-complementing, the Excess 3 code is also useful in subtraction.

The 2421 is a self-complementing weighted code which is commonly employed in counting systems. Other examples of 4-bit weighted codes include the 5421, and the 74-2-1 code.

In this discussion, only three will be compared in detail: the 8421, the 2421, and the Excess 3. It will be seen that one is superior to the other two in specific operation such as addition or subtraction, but for another operation such as digital-to-analog conversion, it is inferior to one or both of the others. Figure 9.2 is a diagrammatic illustration of the arithmetic properties of the various codes. Here the sixteen possible states of a four-bit counter are represented around the perimeter of a circle. Circular representation is used to emphasize the fact that a four-bit binary counter resets itself to 0 on a sixteenth pulse, and that a pure binary code is therefore cyclic in behavior.

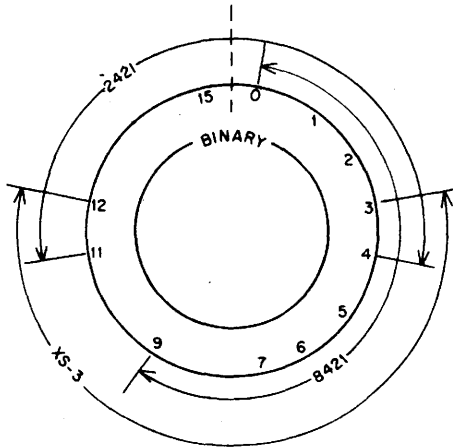


Figure 9.2 Pictorial representation of the essential characteristics of 8421, 2421, and Excess-3 codes.

The three codes under discussion are represented as segments of the circle, and it will be seen that each of them covers only ten of the sixteen possible positions of the rim of the circle. The main purpose of this discussion will be to show that the positions taken by these segments reveal a great deal about the properties of each of the three codes. As a corollary to this statement, the properties of other decimal codes may be studied by examining their positions on the circle. The 8421 code uses the first ten states and discards the last six. The Excess 3 code uses the middle ten states and discards three on either end. The 2421 code uses the first five and the last five and leaves the middle six unused. All three codes use ten sequential states. In most cases codes using broken segments are difficult to maneuver in even simple logical ele-

ments because of the need to build additional gating to skip over the unused segments. This, then, is the first obvious factor to consider in selecting a code.

The next property to consider is the symmetry of the code in comparison with the symmetry of the binary code. The binary code may be considered as being symmetrical about the 15-0 transition point, in that numbers on one side of the circle are "one's" complements of numbers directly opposite. It will be seen that both the Excess 3 and the 2421 codes are symmetrical about this transition while the 8421 code is not. It is also interesting to note that the first two are self-complementing while the latter is not. That is to say, if all of the bits in a number in either of these two codes are complemented, the "nine's" complement of the decimal number is obtained. This simplifies subtraction in decimal arithmetic just as the "one's" complementation simplifies subtraction in binary arithmetic.

A. Counting

In counting applications, all three codes are about equally efficient. The counter must be so constructed that, over ten of its sixteen states, it behaves like a binary counter; it must have built-in gating to skip over the six unused states. If the code is made up of several broken segments of the circle, then several sets of gates would normally be required to skip over the unused segments.

B. Arithmetic Operations

The symmetrical code groups (Excess 3 and 2421) have another feature that is not quite so apparent at first. If they are added in an ordinary binary adder, the proper carry will always be generated. That is to say, the four-bit register will always pass through the 1111 to 0000 transition when the sum exceeds nine. Of course, the number that remains in the adder may or may not be the right sum (i.e., it may require a correction term). Nevertheless, the generation of the carry at the right time is very important. The correction necessary then becomes very simple, particularly with the Excess 3 code. In this case, there are only two possibilities:

1. If two numbers are represented as $A + 3$ and $B + 3$ (as they are in the Excess 3 code), the sum will be $A + B + 6$ if the sum is less than nine. The sum should be $A + B + 3$, to be the correct Excess 3 notation. Therefore, three must be subtracted from the result in order to get the proper Excess 3 code representation.
2. If the sum is greater than nine, $A + B + 6$ will cancel out the six unused states in the code leaving only $A + B$. In this case, it is necessary to add three in order to return to the Excess 3 representation.

This means that the detection of a carry is all that is necessary to determine which correction is necessary. It turns out that the necessary circuitry to perform these steps is extremely simple as was demonstrated in Chapter

8. The 8421 code is more difficult to manipulate in a binary adder. The result of the addition will be correct if the sum is less than nine, but six must be added when the sum is greater than nine. This involves only one correction, but it is more difficult to tell when a correction should take place. Also, the carry may not occur until the correction takes place, which means that each decade must be corrected separately.

Since binary addition is the basis of many arithmetic operations (e.g. multiplication by successive addition), it can be safely stated that the Excess 3 code is the most efficient for general arithmetic manipulations.

C. Up/Down Counting

As expected the self-complementing codes are simpler to use than the 8421 code in up/down counting. With the latter type, special gating is required to make the counter skip over the six unused states.

D. Digital-To-Analog Conversion

Although up to this point in the discussion the Excess 3 code appears to be efficient for most applications, it does have one serious disadvantage in that it is not a weighted code. This makes it inconvenient for performing digital-to-analog conversions. The 2421 code, on the other hand, has nine as the sum of its weights. Since nine is the maximum number in a single decimal digit, a digital-analog resistive summing network will produce a maximum output voltage equal to the value of the reference voltage. The 8421 code makes use of only 10/16ths of a digital-to-analog converter since it can use only the first ten states of a four-bit binary sequence.

The 2421 code has another interesting feature. The six unused states also give valid representations of decimal numbers. (The unused segment extends up to, but not beyond, ten.) This fact is worth remembering if an 8421 code number is to be converted to an analog signal in a D-A converter set to read 2421 code. The 2421 weighted resistor network will still recognize the six states not used in the 2421 code. Thus, the 8421 code will need only to be modified in the representation of the decimal numbers eight and nine.

E. Summary Comments On Decimal Codes

The above discussion is not intended as an exhaustive comparison of all possible decimal codes. It does, however, reveal that for most operations the Excess 3 code has definite advantages over the other two discussed. Despite this, more frequent use is being made of the 8421 code in day-to-day logic design. The main reason for this is probably the fact that it can be more easily remembered than any of the others. Nevertheless, a knowledge of other code groups may often help to simplify the circuitry required to perform any given function.

III REFLECTIVE CODES

The binary sequence which has been used in the earlier experiments suffers from a number of shortcomings in complex control systems. If readout is required on the fly, for example, for a shaft angle digitizer, confusion could arise at the transition from binary 7 to binary 8. If the most significant bit changed slightly before the three least significant bits, the output would briefly indicate 15, a highly erroneous number. To eliminate such catastrophic errors, it is necessary to have a type of code where no more than one bit changes in going between two successive numbers. Such a code is shown in the table below. It is referred to as the Gray binary code.

Decimal	Binary	Reflected Binary (Gray)
0	00000	00000
1	00001	00001
2	00010	00011
3	00011	00010
4	00100	00110
5	00101	00111
6	00110	00101
7	00111	00100
8	01000	01100
9	01001	01101
10	01010	01111
11	01011	01110
12	01100	01010
13	01101	01011
14	01110	01001
15	01111	01000
16	10000	11000
17	10001	11001
18	10010	11011
19	10011	11010
20	10100	11110

Figure 9.3

In going from one piece of equipment to another, it is frequently necessary to change code. Code changes may be done whether in a parallel fashion or a serial fashion. The choice depends on whether speed or cost is more important and also on the form that the input or output data must take. A gray-to-binary converter of the parallel type is illustrated at the end of this chapter.

IV ERROR-DETECTING CODES

Although this chapter does not include an experiment dealing with error-detecting codes, they are brought to the attention of the student here because he may be faced with the task of converting such codes into some other code.

The simplest type of error-detecting code is an ordinary binary (or BCD) code with a parity bit added. In Chapter 2, it was illustrated how to generate and add a parity bit to a binary pattern.

Other more elaborate codes are also in use. They all have one characteristic in common, and that is that they use extra bits to detect and in some cases correct errors.

The Hamming code is one which permits not only error-detection, but error-correction as well. Check bits are distributed throughout a number in such a way as to provide parity checks on various bit positions. To correct an error, the parity checks are made in order, and the results of these checks are arranged in a binary pattern with the least significant bit corresponding to the first check output. The number which turns up in this binary pattern is the number of the bit that is in error. The Hamming code uses at least three redundant bits for single BCD digit error detection and at least four for double error detection.

V MISCELLANEOUS CODES

So far, we have seen that in addition to the pure binary and the pure BCD codes, there are many modified codes which are intended for specific applications. These applications were related to error detection and error prevention. In addition, there is a family of codes which has been developed specifically for the communications industry.

With the ever-increasing need for computers to "talk" to one another over telephone and telegraph facilities, and to a wide variety of electromechanical devices, it has become apparent that all equipment must be able to generate and accept the same code. This common code is called the ASCII code (for American Standard Code for Information Interchange). It is an eight-bit code that is capable of representing all of the common alphanumeric characters. It is interesting to recall that at one time, the 5-bit Teletype code was in wide use; but it suffered from the limitation that it required a keyboard that had fewer keys than standard office typewriters. The eight-bit ASCII code of today gives a typist essentially the same keyboard as is found on a standard office typewriter. The TWX data communications system uses the 8-bit ASCII code, and computer systems make wide use of this facility for remote access terminals.

VI CONVERTERS

With such an abundance of codes available, it is often convenient to convert from one code to another to use the particular advantages of each code. For instance, when accepting an analog signal input, the 2421 code is convenient for the analog-to-digital conversion circuitry, and then if arithmetic operations must be performed on that digitized input, the Excess 3 code is more convenient. Therefore in this particular example a code converter to go from 2421 code to Excess 3 code would be useful. In experiments to follow, designs will be shown for several code converters which accept a number, then, on a convert instruction, change the number from an initial code to a second code.

The method of design for code converters of this type is to first of all determine what initial code conditions have to be detected and, once those conditions have been detected, what final state should be enabled in the converter when the convert clock pulse occurs. The converter must be able to accept a number by means of a parallel transfer of information from a source. The conversion is done on the clock pulse following the receipt of information and the converted information must be cleared prior to accepting another number for conversion.

EXPERIMENT 9.1: THE 2421 TO 8421 CONVERTER

Figure 9.4a shows the initial BCD code which will enter the converter and beside it the final BCD code required after conversion. If the decimal equivalent of 0 through 4 exists in the converter, the 2421 and 8421 representations are identical and therefore no change is required when the convert instruction occurs. The representations of the numbers 5 through 9 are not identical in both codes and the converter must therefore detect the presence of any one of these numbers in 2421 code and convert the 2421 representation to 8421 representation.

DECIMAL	2421 CODE				8421 CODE			
	D	C	B	A	D	C	B	A
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	1	0	1	1	0	1	0	1
6	1	1	0	0	0	1	1	0
7	1	1	0	1	0	1	1	1
8	1	1	1	0	1	0	0	0
9	1	1	1	1	1	0	0	1

Figure 9.4a

DECIMAL	IDENTIFY BY	CHANGE TO
5	C = 0, D = 1	C = 1, B = 0, D = 0
6	B = 0, D = 1	B = 1, D = 0
7		B = 1, D = 0
8	B = 1, C = 1, D = 1	B = 0, C = 0
9		B = 0, C = 0

Figure 9.4b

To detect that 5 is present in 2421 form it is sufficient to be sure that bit D is a 1 and bit C is a 0. When the number 5 is detected the converter must cause the following changes to go from 2421 to 8421 representation: bit B becomes 0, bit C becomes 1, bit D becomes 0. In converting the numbers 6 and 7 from 2421 to 8421 the same changes are required (D becomes 0, and B becomes 1); therefore only bits D and B must be examined to identify when 6 or 7 is present in 2421 code. Similarly 8 and 9 require the same changes in going from 2421 code to 8421 code so only bits B, C and D must be examined to identify the two numbers. Figure 9.5 is a 2421 to 8421 converter. Construct it on the COMPUTER LAB connecting the A, B, C and D inputs to rocker switches (starting with D on the left). The clock and reset inputs are pulser switches and all the flip-flop 1 outputs should be connected to lamp indicators (start with flip-flop D on the left).

To operate the converter, first clear the register by depressing the reset pulser once. Read-in is accomplished by setting all inputs which are to be 1 LO and all inputs which are to be 0 HI and then pulsing the clock pulser switch once. A number will have been read into the register and will be present in 2421 code. All the input switches must be disabled by putting them in the HI position. The next clock converts the number from 2421 code to 8421 code. Convert the following numbers from 2421 to 8421 code using the method outlined above:

0001, 0011, 1011, 1101, 1111.

QUESTIONS

- Describe the function of gates E, F and G by explaining
 - what each gate detects and
 - what each gate does.

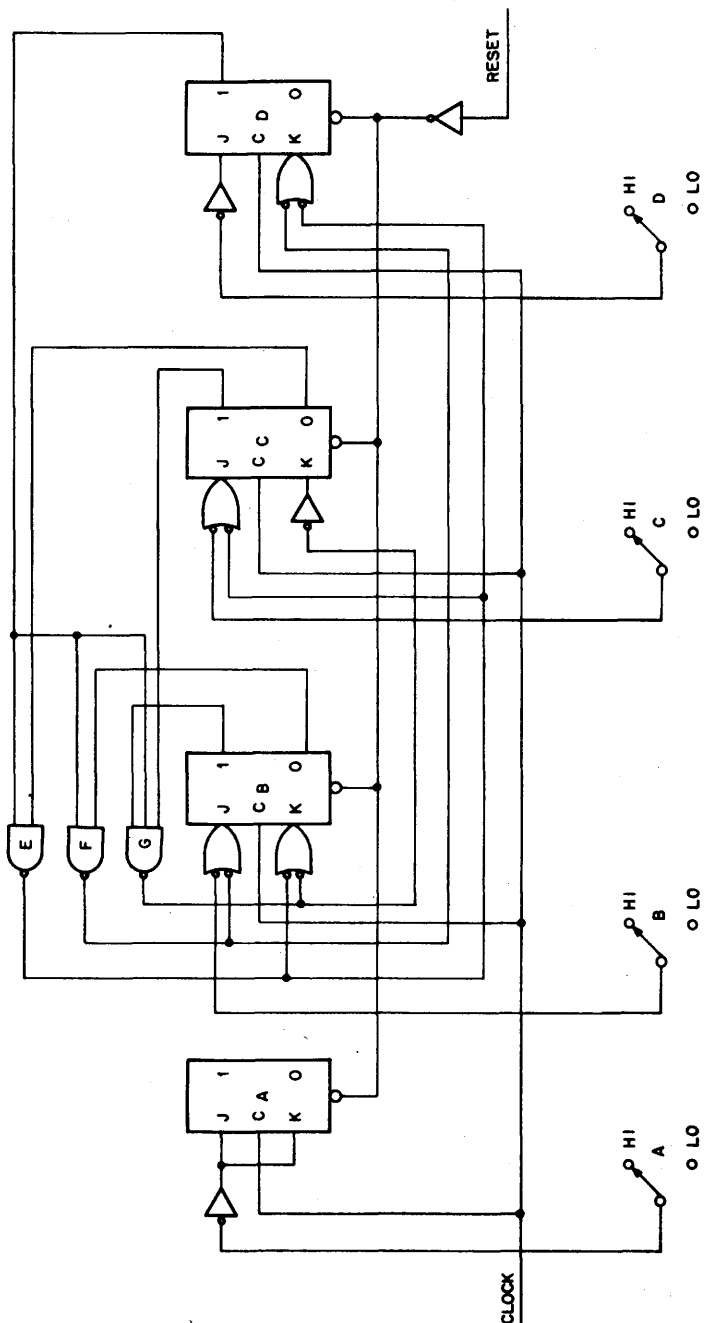


Figure 9.5 2421 — 8421 Converter

- Why are gates E and F only two-input gates?
- Describe in detail how a 2421 number is read into the register.

EXPERIMENT 9.2: THE 5421 TO 8421 CONVERTER

The 5421 to 8421 converter is more complex because each of 5, 6, 7, 8 and 9 must be separately detected and separately converted. Figure 9.6a shows corresponding states of the 5421 and 8421 codes. Figure 9.6b shows the methods of identifying each of the 5421 states representing the decimal numbers 5 through 9 and the changes which must be made when those numbers are detected. The converter to perform the functions indicated in figure 9.6b is shown in figure 9.7.

DECIMAL	5421 CODE				8421 CODE			
	D	C	B	A	D	C	B	A
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	1	0	0	0	0	1	0	1
6	1	0	0	1	0	1	1	0
7	1	0	1	0	0	1	1	1
8	1	0	1	1	1	0	0	0
9	1	1	0	0	1	0	0	1

Figure 9.6a

	IDENTIFY BY	CHANGE TO
5	$A = B = C = 0, D = 1$	$A = 1, C = 1, D = 0$
6	$A = 1, B = 0, D = 1$	$A = 0, B = 1, C = 1, D = 0$
7	$A = 0, B = 1, D = 1$	$A = 1, C = 1, D = 0$
8	$A = 1, B = 1, D = 1$	$A = 0, B = 0$
9	$C = 1, D = 1$	$A = 1, C = 0$

Figure 9.6b

Construct the converter on the COMPUTER LAB connecting inputs and outputs in the same manner as was done for the 2421 to 8421 converter. The operation of 5421-8421 converter is identical to the 2421-8421 model. Convert the following numbers from 5421 code to 8421 code using the converter:

0010, 0100, 1001, 1011, 1100

QUESTIONS

- Describe the functions of gates E through I under the following headings:
 - gate detects
 - output enables
- Why must all input switches be in the HI condition prior to the convert clock pulse?
- Design an Excess 3 to 8421 code converter.
- Design an 8421 to 2421 code converter.

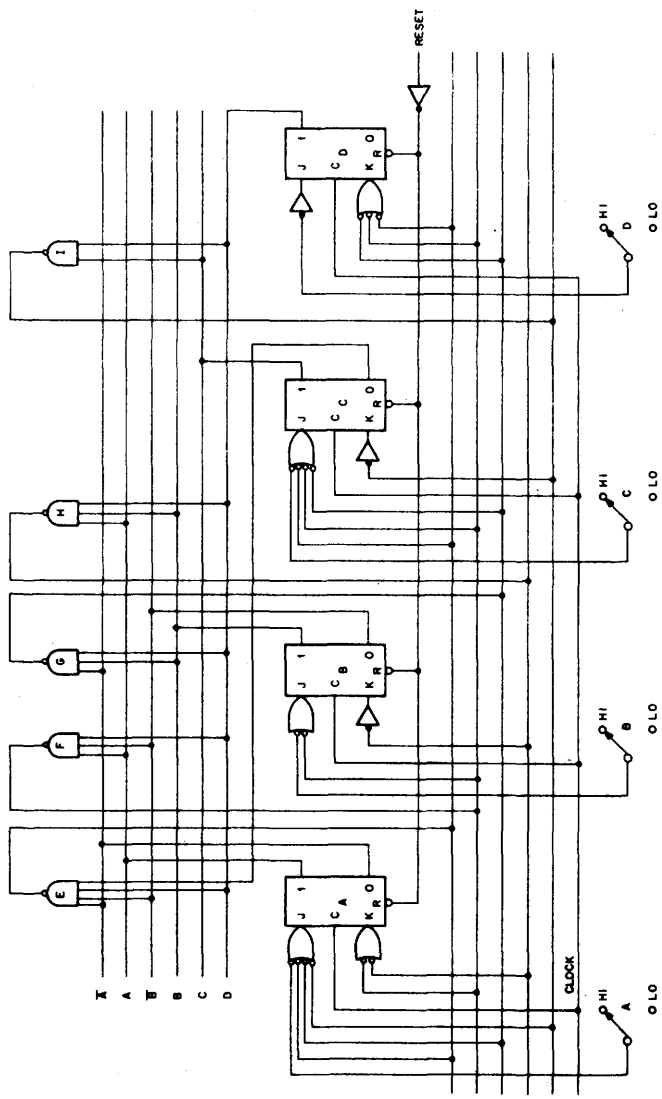


Figure 9.7 5421-8421 Converter

EXPERIMENT 9.3: GRAY TO BINARY CONVERTER

As was mentioned earlier in this experiment, gray code or reflected binary is useful in control applications because there is only one bit transition to increment or decrement by 1. However arithmetic operations are more easily done in normal binary code; consequently, there is often a requirement to convert one to another. Figure 9.8a shows four bits of regular binary code and corresponding numbers in gray code. The general conversion rule in going from gray to binary is:

$$\text{Bit N. (1)} = [\text{Bit N. (0)}] + [\text{Bit N+1 (1)}].$$

In other words the 2^N bit will be a 1 after the conversion either if bit N in gray code were a 1 before the conversion or if the 2^{N+1} bit will be a 1 after the conversion. A bracketed (0) refers to conditions prior to conversion and a bracketed (1) refers to conditions after conversion.

The following conversion example demonstrates how the conversion rule can be applied to convert the gray code number 1010 to binary code:

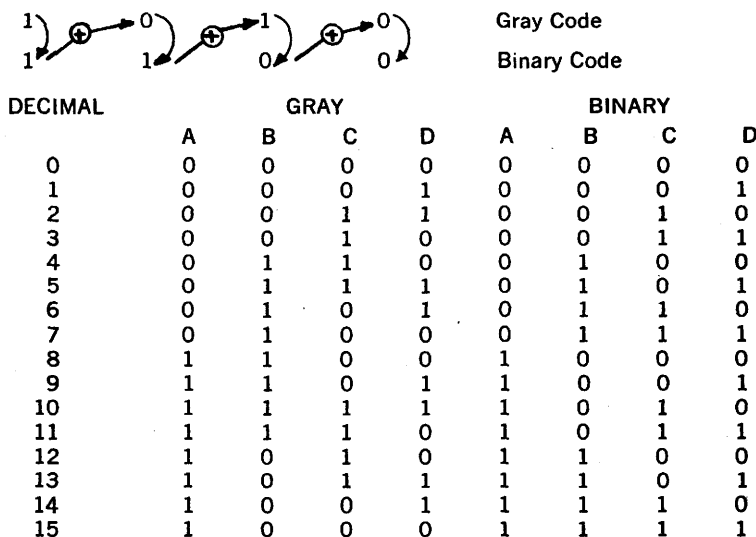


Figure 9.8a

$$\begin{aligned}
 A(1) &= A(0) \\
 B(1) &= B(0) \oplus A(1) = B(0) \oplus A(0) \\
 C(1) &= C(0) \oplus B(1) = C(0) \oplus [B(0) \oplus A(0)] \\
 D(1) &= D(0) \oplus C(1) = D(0) \oplus \{C(0) \oplus [B(0) \oplus A(0)]\}
 \end{aligned}$$

Figure 9.8b

Construct the gray to binary converter on the COMPUTER LAB. Switch inputs A through D should be connected to rocker switches starting with A at the left. Lamp indicators should be connected to the 1 outputs of flip-flops. The reset and clock inputs are driven by pulser switches. The operating sequence is the same as that for the 2421 to 8421 converter illustrated on page 111.

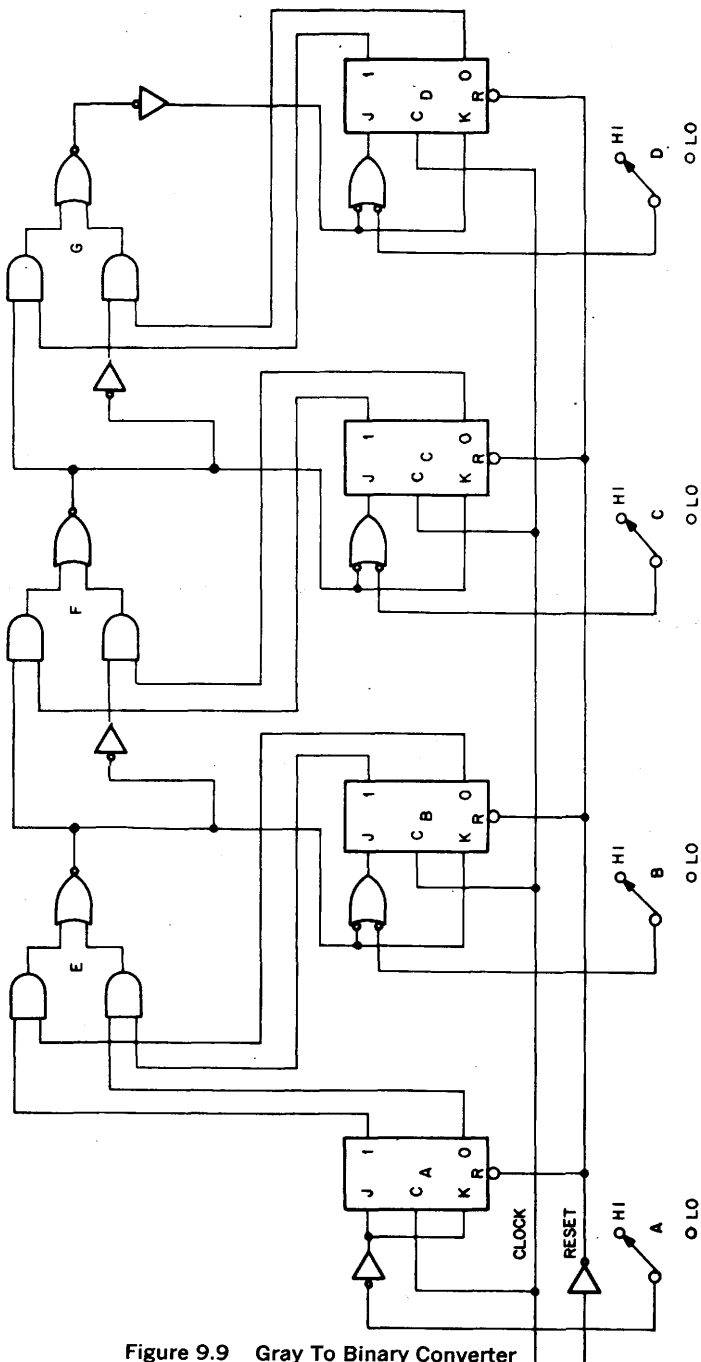


Figure 9.9 Gray To Binary Converter

Test the converter by converting the following gray code numbers into binary and noting the results.

0011, 0110, 0101, 1100, 1101, 1111, 1010, 1000.

QUESTIONS

8. What Boolean functions do gates E, F, and G detect?
9. How much time, after the gray code number has been read into the register, must be allowed for the circuitry to settle prior to receiving a convert clock pulse?

SUPPLEMENTARY QUESTIONS

10. Define a general equation to convert a binary number into a gray number.
11. Design and construct a binary to gray converter.

VII BINARY TO BCD CONVERSION

Sometimes for decoding purposes it is necessary to convert a pure binary number into a binary coded decimal number.

The most common way of accomplishing this conversion is to put the binary number in a down counter and count down to zero while simultaneously counting up from zero in a BCD counter. When zero is reached in a binary down counter the BCD up counter will be the BCD equivalent of the original binary number.

SUPPLEMENTARY QUESTIONS

12. Make a block diagram of a system which could perform a binary to 8421 BCD conversion. Outline briefly in words what each of the functional blocks in your diagram would do and how they might be constructed.

CHAPTER 10

SYSTEM CONSIDERATIONS

I INTRODUCTION

The subsystems constructed to this point are still a long way from operating anything like a computer. To add a number using the parallel adder in Experiment 7.1 required several manually controlled steps to perform a single addition. In the time one addition could be performed on the two step parallel adder, a small boy with a pencil could do 3 or 4 equally complex additions. The speed limitation of the adder is not imposed by the execution time of the device (measured in nanoseconds) but rather by the operating time of the person using that device (measured in seconds). Thus if human operation can be excluded from the requirement of the adder there will be a reduction of up to 5 orders of magnitude in the add time.

To remove human control operations from a control sequence a new device must be added to the system or subsystem, namely the control. The control must be capable of executing all the instructions in sequence which were previously initiated from outside.

In controlling subsystems and bringing different subsystem functions together into a system, consideration has to be given to the timing or synchronizing of signals coming into a control device. Synchronizers built into a system assure that signals will be brought into an operation at the correct time.

If a system has proper control and synchronizing circuitry built in there are still limits to the operating speed imposed by propagation delays which are inherent in the devices used. In this experiment methods for control and synchronizing will be discussed and built and ways of determining maximum system frequency will be shown.

II CONTROL

Each system or subsystem control will be unique but there is a definite method which can be applied to control design which is helpful in most cases. As was mentioned before, the purpose of a control is to generate an operating sequence for a subsystem or system which causes a series of operations to commence when the control is instructed to start. The first design requirement is to decide exactly what that operating sequence is. For instance for the two step adder in Experiment 7.1 the operating sequence is:

1. ENABLE HALF ADD, DISABLE CARRY
2. ALLOW CLOCK PULSE TO PASS
3. ENABLE CARRY, DISABLE HALF ADD
4. ALLOW CLOCK PULSE TO PASS
5. STOP. (DISABLE HALF ADD AND CARRY)

Figure 10.1 is the two step parallel adder. A control unit to operate this adder would have to control both the HALF ADD ENABLE and CARRY ENABLE inputs in order to perform additions. The clock input can be connected permanently to a clock line as the adder does not perform any function unless one of the enable inputs is asserted.

Now that the operating sequence has been defined and the controlling inputs have been found it is helpful to draw a timing diagram which shows what levels must be present on the controlling lines to give the desired operating sequence. Figure 10.2 is a timing diagram showing the required control information along with the time at which the two steps of the addition take place.

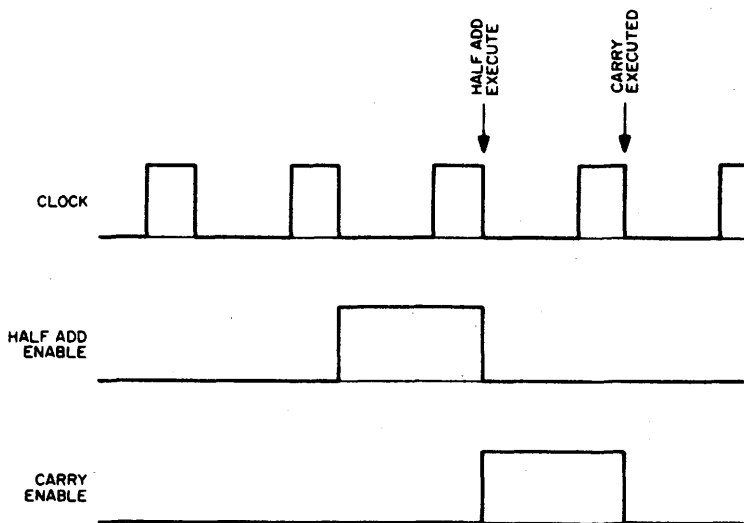


Figure 10.2 Adder Control Timing

The basis of most controls is a counter of some form or another which, in conjunction with the clock, generates the required control sequence and then stops. In this particular case a two-bit asynchronous counter will give four unique states which will be a sufficient basis for the control. The counter must count only to the binary number 11 and stop. Figure 10.3 is a counter which will start counting clock pulses after being reset to 0 and when it reaches 11 the NAND gate feeding back to the J and K inputs of the first flip-flop will disable the counter and further clock pulses will be ignored. The control sequence in Figure 10.2 can be generated by detecting the 01 condition in the counter to assert the HALF ADD ENABLE line and detecting the 10 condition to assert the CARRY ENABLE line. Gates A and B detect the 01 and 10 conditions respectively and provide control to the adder. In summary, the design method for a control is as follows:

1. Itemize the steps required to control the device.
2. Make a timing diagram showing the sequence of logic levels which has to be generated by the control.
3. Design a sequence generator or control to give the output indicated by the timing diagram.

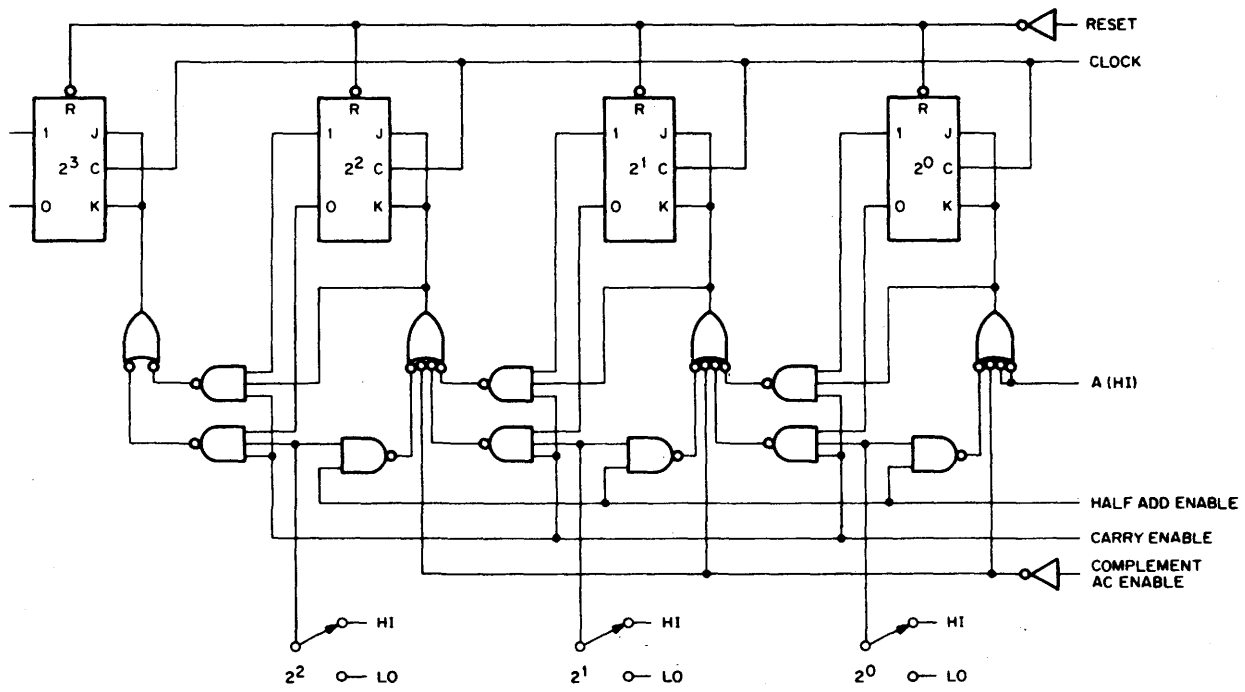


Figure 10.1 Three Bit Parallel Adder

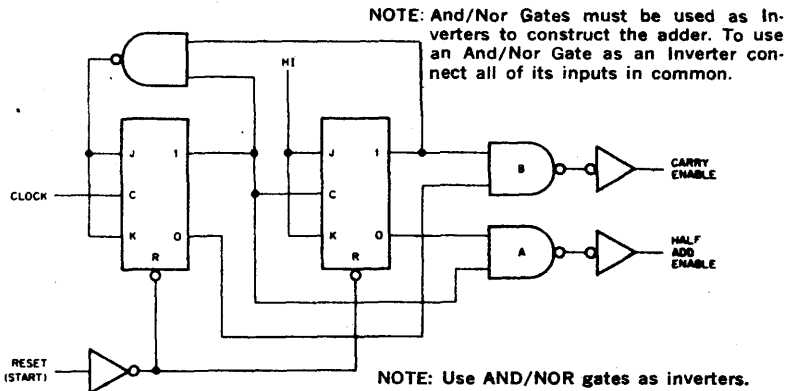


Figure 10.3 2 Step Parallel Adder Control

EXPERIMENT 10.1: PARALLEL ADDER CONTROL

Construct the parallel adder and control on the COMPUTER LAB, wiring the adder inputs and outputs as indicated in Chapter 7 (with the exception of CLOCK, HALF ADD ENABLE and CARRY ENABLE). Construct the adder control and connect it to the adder. The RESET (start) input to the control is connected to a pulser. Connect the clock inputs of both the adder and its control to the clock output on the COMPUTER LAB and set the clock to operate at its slowest speed. The operating sequence of the controlled adder will now be:

1. Clear the accumulator by depressing the adder RESET pulser once.
2. Set a number into the incident switch register.
3. Read the number in by depressing the control RESET (start) pulser once.
4. Set a second number into the input switch register.
5. Add the incident number to the accumulator by depressing the control RESET (start) pulser once.

With the clock set at its lowest speed it is possible to see the half add and carry operations taking place. Test the adder and its control by performing the following additions:

A.
$$\begin{array}{r} 011 \\ 001 \\ \hline 100 \end{array}$$

B.
$$\begin{array}{r} 010 \\ 101 \\ \hline 111 \end{array}$$

C.
$$\begin{array}{r} 011 \\ 011 \\ \hline 110 \end{array}$$

QUESTIONS

1. Explain in detail how the control works.
2. Why is the 00 condition of the control counter not used for a control operation?

SUPPLEMENTARY QUESTIONS

3. As was mentioned in Chapter 7 an adder can be used to multiply by performing a number of additions. Design an extension for the control to perform a selected number of successive additions.

4. In some cases it is necessary to transfer information in parallel from a multiple bit source to a shift register and then shift information out serially from the register. A typical application for this type of register is to read several bits of information from a computer and then shift it out serially to go down a line to another computer or read-out device. Design a four-bit register and control to:

- a) Read 4 bits of information simultaneously.
- b) Shift out the 4 bits serially on the next 4 clock pulses.

III SYNCHRONIZING

If a control receives a signal at the wrong time the result can be an error. For example, suppose the adder control received a second start instruction just after the half add occurred. The number in the accumulator would most likely be in error because the carry operation would not have been completed prior to the second start instruction. Thus it is necessary to synchronize the start instruction with the operation of the adder.

The synchronous Up/Down Counter in Chapter 5 is another example of a device requiring synchronization. If both its UP ENABLE and DOWN ENABLE lines are held in the HI condition, the counter will complement every clock pulse and possibly end up in an erroneous state. This condition can be avoided by having only one direction line which will cause the counter to count up when HI and count down when LO such as in Figure 10.4. With single line direction assertion, the counter will work perfectly well until a change of direction occurs when the clock pulse is HI. In this case the state of the counter following the clock pulse trailing edge is not certain. The change of direction on the up and down enable lines must therefore be synchronized with the operation of the clock.

EXPERIMENT 10.2: SYNCHRONIZERS

A J-K flip-flop is ideally suited for synchronizing with the clock because its output transitions take place after the trailing edge of the clock pulse regardless of when the conditioning J and K inputs change state. Figure 10.5 shows how a J-K flip-flop can be used to control direction and to synchronize the counter so that transitions do not occur on the up enable and down enable lines when the clock input is HI.

This particular synchronizer assures that level changes on the enable lines occur at the correct times. Other synchronizers direct clock pulses down one line or another dependent on the state of a flip-flop or in other cases receive a pulse and cause a change to occur at some later time in synchronism with the clock etc. Like a control, each individual synchronizer has to be tailored to a particular system. Construct the circuit shown in Figure 10.5.

QUESTIONS

5. Design and construct an Up/Down Counter similar to the one in Figure 10.5 but which will accept two separate pulse inputs to change direction. That is, a pulse on the count up line will insure that after the following clock pulse the counter will change direction and then count up. And conversely a pulse on the count down line will insure that after the next clock pulse, the counter will change direction and then count down. An R-S flip-flop might be used in this application (Chapter 3).

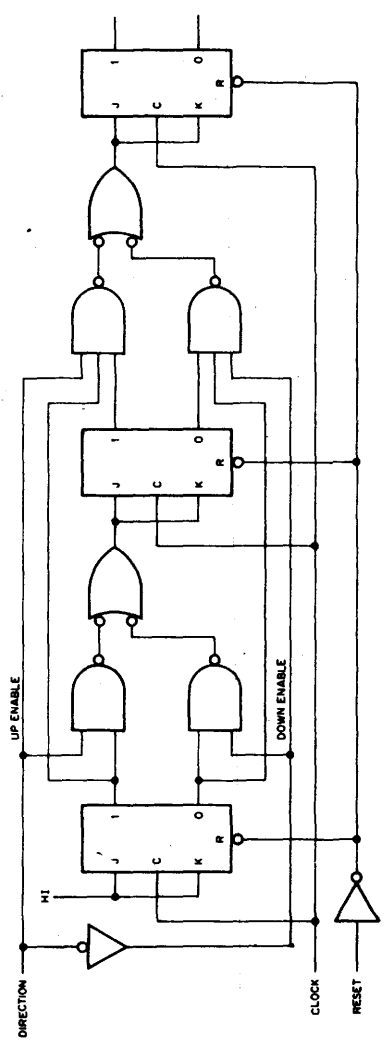


Figure 10.4 Synchronous Up/Down Counter

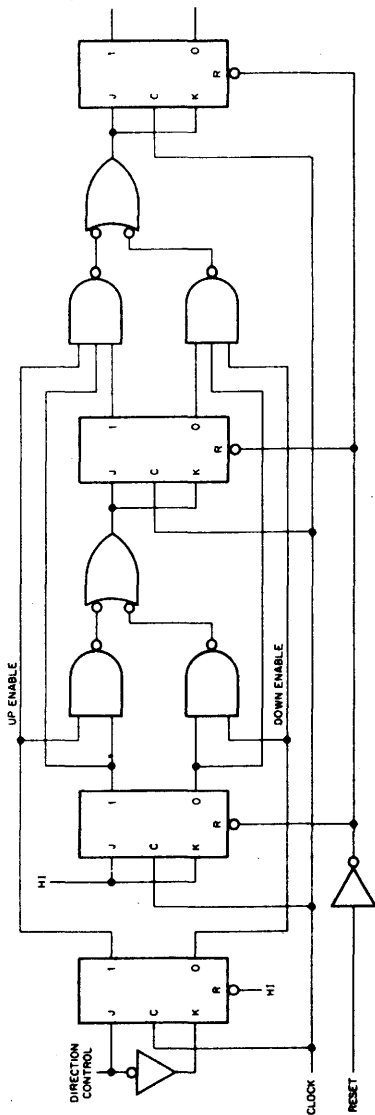


Figure 10.5 Controlled Synchronous Up/Down Counter

6. Design a synchronizer for the two step adder and control to insure that a second start instruction is not recognized until the adder has completed its cycle.
7. Design a synchronizer to direct clock pulses to one of two counters dependent on whether a counter #1 or counter #2 control pulse was received last. Be sure that the synchronizer will not permit split pulses to get through to either counter.

IV SYSTEM FREQUENCY

Because the circuits used on the COMPUTER LAB are so fast, the fact that they do take time to transmit information is often overlooked. With careful design, logic configurations can be used at speeds of 10 megacycles and higher, but there are limits. A timing diagram of the longest logic chain is the most useful tool in determining a system operating frequency. Sufficient time must be allowed from the trailing edge of a clock pulse to the leading edge of the next clock pulse for information to propagate down the longest logic chain and settle.

Figure 10.6 is a self-stopping asynchronous up counter that will count to the number 2^N and stop. The longest logic chain in this particular case is the chain of flip-flops. Each flip-flop is looking at the output of the previous one and therefore there is a delay of 35 nanoseconds per flip-flop. In order to inhibit counting the counter must do the transition from 01 111 to 100 0000, making the 2^N bit a 1 and disabling the J and K Inputs of flip-flop 2^0 . The propagation delay of the counter as a whole is thus $35(N + 1)$ ns as shown in Figure 10.7. The time between the trailing edge of one clock pulse to the leading edge of the next must therefore be $35(N + 1)$ nanoseconds. If the clock pulses are closer together, there is a good chance that the counter will not inhibit quickly enough to stop at the number 2^N but will overshoot and carry on to the number $2^N + 1$.

EXPERIMENT 10.3: MAXIMUM FREQUENCY

Construct an 8-bit counter similar to the one in Figure 10.6 on the COMPUTER LAB. Connect the RESET input to a pulser and the CLOCK input to the COMPUTER LAB clock. Connect all flip-flop outputs to lamps. When the reset pulser is depressed and held down, the counter will count up to 2^7 and stop. Operate the counter with the clock set to a fairly slow speed and verify that it performs as outlined. Disconnect the jumper from the clock range selector and turn the clock variable control to its fastest position and retest the counter.

QUESTIONS

8. Does the counter operate identically at HI and LO clock repetition rates? Why?
9. How fast can the counter be operated? What is the average delay per flip-flop?
10. Draw a timing diagram for the longest logic chain in the two step parallel adder with control (which was constructed in this experiment). What is the maximum frequency of operation for this device, assuming 50 nanosecond clock pulses?
11. How fast can the synchronous Up/Down Counter (constructed in this experiment) be operated? Do not forget that in some cases the counter may have to change direction in the time between two clock pulses.

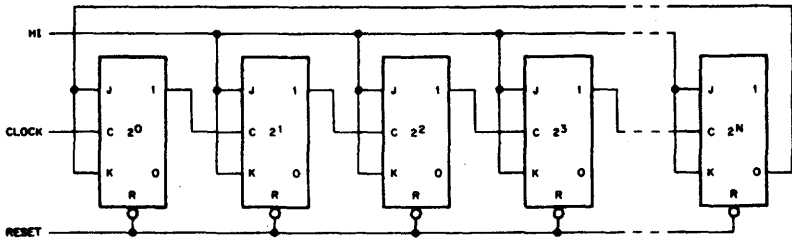


Figure 10.6 Asynchronous Self-Stopping UP Counter

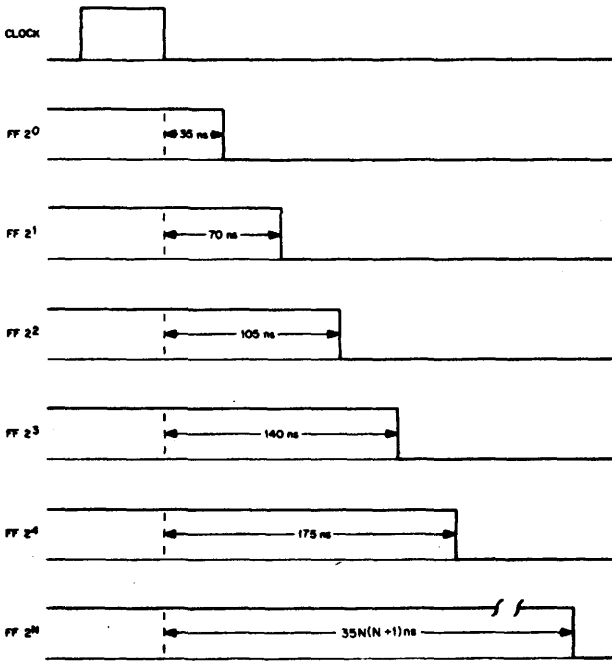


Figure 10.7 Flip-Flop "1" Output Propagation Delays



APPENDIX A

CHECKOUT PROCEDURES

Most faulty circuit operations result from loose patchcords, incorrect patchcord wiring, improper student logic designs, or other such errors that are easy to remedy. Therefore, before concluding that the COMPUTER LAB unit needs repair, please identify the specific fault and then consult the following troubleshooting guide.

If none of the suggested remedies correct the fault, then:

1. Obtain a Return Authorization Number (RA#) from Module Marketing Service, Digital Equipment Corporation, Maynard, Massachusetts, by calling (617) 897-5111.
2. Ship the COMPUTER LAB prepaid insured to:

Digital Equipment Corporation
COMPUTER LAB Repair Center
146 Main Street
Maynard, Massachusetts 01754
3. On the shipping container, list your NAME, RETURN ADDRESS and RETURN AUTHORIZATION NUMBER.
4. If the COMPUTER LAB is NOT on warranty, enclose a check or Money Order for \$35.00 payable to "Digital Equipment Corporation" or enclose a Purchase Order. The normal warranty period is 90 days. Consult the Warranty Agreement for exact terms and conditions.
5. DIGITAL will repair the unit and return it postpaid.

FAULT	CAUSE	REMEDY
Unit completely inoperative.	No power input.	—Plug into 120 VAC receptacle. —Turn power switch on.
	Wiring error.	—Check wiring.
	Damaged power supply.	—Check HI terminal for + 2.8V to ground
Intermittent failure.	Loose connections.	—Connect any free wires. —Push in loose pin connectors.
	Gate fan-out exceeded.	—Parallel gates to obtain higher fan-out.
Sectional failure of a circuit.	Incomplete wiring.	—All gates used should have at least one connection on each input and output. —If no connection is present, check circuit for completeness.
	Wiring error.	—Check for sectional errors in wiring. —Check that two outputs are not wired in parallel.
	Gate fan-out exceeded.	—Parallel gates to obtain higher fan-out.

FAULT	CAUSE	REMEDY														
HI terminal does not supply HI level.	HI terminal or rocker switch output connected to ground or another output.	—Check wiring to be sure no connection is made from HI terminal or switch output to any other outputs.														
Rocker switch will not supply HI level.	Rocker switch output or HI terminal connected to gnd or another terminal	—Check wiring to be sure no connection is made from a switch output or a HI terminal to any other output terminal.														
Lamp will not light.	Incorrectly wired circuit.	—Check wiring.														
	Bulb failure.	—Test lamp by removing input wires from circuit and connecting lamp input to HI (lamp should light). If it still fails remove the switch panel cover and replace bulb with one of the spares supplied. Note: Bulbs should be replaced with an exact replacement bulb DEC #12-5591.														
Pulser switch will not supply HI level.	Pulser output connected to another switch or logic output.	—Check the pulser output connections.														
Clock inoperative.	Clock output connected to another logic or switch output.	—Check clock output wiring.														
	Clock coarse range selector wrongly connected.	—Check clock coarse range wiring.														
2-input NAND gate inoperative.	Wiring error.	—Check wiring especially for other outputs connected in parallel. —Check for disconnected inputs.														
	Gate failure.	—Test to determine failure by connecting gate inputs to switches and gate outputs to lamp drivers to verify the following truth table: <table border="1" data-bbox="600 1254 911 1403"> <thead> <tr> <th>A</th> <th>B</th> <th>Output</th> </tr> </thead> <tbody> <tr> <td>LO</td> <td>LO</td> <td>HI</td> </tr> <tr> <td>LO</td> <td>HI</td> <td>HI</td> </tr> <tr> <td>HI</td> <td>LO</td> <td>HI</td> </tr> <tr> <td>HI</td> <td>HI</td> <td>LO</td> </tr> </tbody> </table>	A	B	Output	LO	LO	HI	LO	HI	HI	HI	LO	HI	HI	HI
A	B	Output														
LO	LO	HI														
LO	HI	HI														
HI	LO	HI														
HI	HI	LO														

FAULT	CAUSE	REMEDY																																																																																				
3-input NAND gate inoperative.	Wiring error.	<ul style="list-style-type: none"> —Check wiring (especially gate output). —Check for disconnected inputs. 																																																																																				
	Gate failure.	<ul style="list-style-type: none"> —Disconnect gate from circuit and connect inputs to rocker switches and the output to a lamp to perform the following test: <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>A</th> <th>B</th> <th>C</th> <th>Output</th> </tr> </thead> <tbody> <tr><td>LO</td><td>LO</td><td>LO</td><td>HI</td></tr> <tr><td>LO</td><td>LO</td><td>HI</td><td>HI</td></tr> <tr><td>LO</td><td>HI</td><td>LO</td><td>HI</td></tr> <tr><td>LO</td><td>HI</td><td>HI</td><td>HI</td></tr> <tr><td>HI</td><td>LO</td><td>LO</td><td>HI</td></tr> <tr><td>HI</td><td>LO</td><td>HI</td><td>HI</td></tr> <tr><td>HI</td><td>HI</td><td>LO</td><td>HI</td></tr> <tr><td>HI</td><td>HI</td><td>HI</td><td>LO</td></tr> </tbody> </table>	A	B	C	Output	LO	LO	LO	HI	LO	LO	HI	HI	LO	HI	LO	HI	LO	HI	HI	HI	HI	LO	LO	HI	HI	LO	HI	HI	HI	HI	LO	HI	HI	HI	HI	LO																																																
A	B	C	Output																																																																																			
LO	LO	LO	HI																																																																																			
LO	LO	HI	HI																																																																																			
LO	HI	LO	HI																																																																																			
LO	HI	HI	HI																																																																																			
HI	LO	LO	HI																																																																																			
HI	LO	HI	HI																																																																																			
HI	HI	LO	HI																																																																																			
HI	HI	HI	LO																																																																																			
4-input NAND gate inoperative.	Wiring error.	<ul style="list-style-type: none"> —Check wiring, especially output connections to be sure no connection is made to another output. —Check for disconnected inputs. 																																																																																				
	Gate failure.	<ul style="list-style-type: none"> —Disconnect gate from circuit and connect inputs to rocker switches and output to lamp driver. Test to see that all conditions in the truth table are valid. <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>A</th> <th>B</th> <th>C</th> <th>D</th> <th>Output</th> </tr> </thead> <tbody> <tr><td>LO</td><td>LO</td><td>LO</td><td>LO</td><td>HI</td></tr> <tr><td>LO</td><td>LO</td><td>LO</td><td>HI</td><td>HI</td></tr> <tr><td>LO</td><td>LO</td><td>HI</td><td>LO</td><td>HI</td></tr> <tr><td>LO</td><td>LO</td><td>HI</td><td>HI</td><td>HI</td></tr> <tr><td>LO</td><td>HI</td><td>LO</td><td>LO</td><td>HI</td></tr> <tr><td>LO</td><td>HI</td><td>LO</td><td>HI</td><td>HI</td></tr> <tr><td>LO</td><td>HI</td><td>HI</td><td>LO</td><td>HI</td></tr> <tr><td>LO</td><td>HI</td><td>HI</td><td>HI</td><td>HI</td></tr> <tr><td>HI</td><td>LO</td><td>LO</td><td>LO</td><td>HI</td></tr> <tr><td>HI</td><td>LO</td><td>LO</td><td>HI</td><td>HI</td></tr> <tr><td>HI</td><td>LO</td><td>HI</td><td>LO</td><td>HI</td></tr> <tr><td>HI</td><td>LO</td><td>HI</td><td>HI</td><td>HI</td></tr> <tr><td>HI</td><td>HI</td><td>LO</td><td>LO</td><td>HI</td></tr> <tr><td>HI</td><td>HI</td><td>LO</td><td>HI</td><td>HI</td></tr> <tr><td>HI</td><td>HI</td><td>HI</td><td>LO</td><td>HI</td></tr> <tr><td>HI</td><td>HI</td><td>HI</td><td>HI</td><td>LO</td></tr> </tbody> </table>	A	B	C	D	Output	LO	LO	LO	LO	HI	LO	LO	LO	HI	HI	LO	LO	HI	LO	HI	LO	LO	HI	HI	HI	LO	HI	LO	LO	HI	LO	HI	LO	HI	HI	LO	HI	HI	LO	HI	LO	HI	HI	HI	HI	HI	LO	LO	LO	HI	HI	LO	LO	HI	HI	HI	LO	HI	LO	HI	HI	LO	HI	HI	HI	HI	HI	LO	LO	HI	HI	HI	LO	HI	HI	HI	HI	HI	LO	HI	HI	HI	HI	HI
A	B	C	D	Output																																																																																		
LO	LO	LO	LO	HI																																																																																		
LO	LO	LO	HI	HI																																																																																		
LO	LO	HI	LO	HI																																																																																		
LO	LO	HI	HI	HI																																																																																		
LO	HI	LO	LO	HI																																																																																		
LO	HI	LO	HI	HI																																																																																		
LO	HI	HI	LO	HI																																																																																		
LO	HI	HI	HI	HI																																																																																		
HI	LO	LO	LO	HI																																																																																		
HI	LO	LO	HI	HI																																																																																		
HI	LO	HI	LO	HI																																																																																		
HI	LO	HI	HI	HI																																																																																		
HI	HI	LO	LO	HI																																																																																		
HI	HI	LO	HI	HI																																																																																		
HI	HI	HI	LO	HI																																																																																		
HI	HI	HI	HI	LO																																																																																		

FAULT	CAUSE	REMEDY																																																																																				
AND/NOR gate inoperative.	Wiring error.	<p>—Check wiring, especially output connections to be sure no connection is made to another output.</p> <p>—Check for disconnected inputs.</p>																																																																																				
	Gate failure.	<p>—Disconnect gate from circuit and connect inputs to rocker switches and output to lamp driver. Test to see that all conditions in the truth table are valid.</p> <table border="1" data-bbox="552 424 905 871"> <thead> <tr> <th>A</th> <th>B</th> <th>C</th> <th>D</th> <th>Output</th> </tr> </thead> <tbody> <tr><td>LO</td><td>LO</td><td>LO</td><td>LO</td><td>HI</td></tr> <tr><td>LO</td><td>LO</td><td>LO</td><td>HI</td><td>HI</td></tr> <tr><td>LO</td><td>LO</td><td>HI</td><td>LO</td><td>HI</td></tr> <tr><td>LO</td><td>LO</td><td>HI</td><td>HI</td><td>LO</td></tr> <tr><td>LO</td><td>HI</td><td>LO</td><td>LO</td><td>HI</td></tr> <tr><td>LO</td><td>HI</td><td>LO</td><td>HI</td><td>HI</td></tr> <tr><td>LO</td><td>HI</td><td>HI</td><td>LO</td><td>HI</td></tr> <tr><td>LO</td><td>HI</td><td>HI</td><td>HI</td><td>LO</td></tr> <tr><td>HI</td><td>LO</td><td>LO</td><td>LO</td><td>HI</td></tr> <tr><td>HI</td><td>LO</td><td>LO</td><td>HI</td><td>HI</td></tr> <tr><td>HI</td><td>LO</td><td>HI</td><td>LO</td><td>HI</td></tr> <tr><td>HI</td><td>LO</td><td>HI</td><td>HI</td><td>LO</td></tr> <tr><td>HI</td><td>HI</td><td>LO</td><td>LO</td><td>LO</td></tr> <tr><td>HI</td><td>HI</td><td>LO</td><td>HI</td><td>LO</td></tr> <tr><td>HI</td><td>HI</td><td>HI</td><td>LO</td><td>LO</td></tr> <tr><td>HI</td><td>HI</td><td>HI</td><td>HI</td><td>LO</td></tr> </tbody> </table>	A	B	C	D	Output	LO	LO	LO	LO	HI	LO	LO	LO	HI	HI	LO	LO	HI	LO	HI	LO	LO	HI	HI	LO	LO	HI	LO	LO	HI	LO	HI	LO	HI	HI	LO	HI	HI	LO	HI	LO	HI	HI	HI	LO	HI	LO	LO	LO	HI	HI	LO	LO	HI	HI	HI	LO	HI	LO	HI	HI	LO	HI	HI	LO	HI	HI	LO	LO	LO	HI	HI	LO	HI	LO	HI	HI	HI	LO	LO	HI	HI	HI	HI
A	B	C	D	Output																																																																																		
LO	LO	LO	LO	HI																																																																																		
LO	LO	LO	HI	HI																																																																																		
LO	LO	HI	LO	HI																																																																																		
LO	LO	HI	HI	LO																																																																																		
LO	HI	LO	LO	HI																																																																																		
LO	HI	LO	HI	HI																																																																																		
LO	HI	HI	LO	HI																																																																																		
LO	HI	HI	HI	LO																																																																																		
HI	LO	LO	LO	HI																																																																																		
HI	LO	LO	HI	HI																																																																																		
HI	LO	HI	LO	HI																																																																																		
HI	LO	HI	HI	LO																																																																																		
HI	HI	LO	LO	LO																																																																																		
HI	HI	LO	HI	LO																																																																																		
HI	HI	HI	LO	LO																																																																																		
HI	HI	HI	HI	LO																																																																																		

FAULT	CAUSE	REMEDY
J-K flip-flop inoperative.	Wiring error.	—J-K or R input not connected. —1 or 0 output connected to another output.
	Incorrect use of Clock input (clock connected to rocker switch).	—Connect Clock to pulser or COMPUTER LAB clock.
	J-K Flip-Flop failure.	—Test the flip-flop according to the procedure in Figure 1 to determine failure.*

Before Clock Pulse				After Clock Pulse	
1 Output	0 Output	J	K	1 Output	0 Output
LO	HI	LO	LO	LO	HI
LO	HI	LO	HI	LO	HI
LO	HI	HI	LO	HI	LO
LO	HI	HI	HI	HI	LO
HI	LO	LO	LO	HI	LO
HI	LO	LO	HI	LO	HI
HI	LO	HI	LO	HI	LO
HI	LO	HI	HI	LO	HI

- *1. When performing the above test, the RESET input should be connected to a HI logic level.
2. To test the RESET input, connect it to a LO logic level. The flip-flop 0 output should remain HI and the 1 output LO for all J and K input conditions.

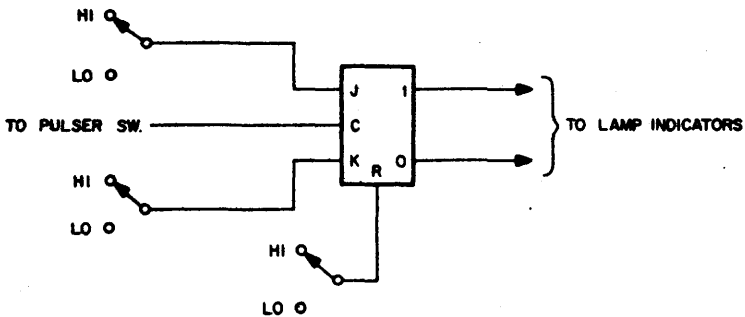


Figure 1 J-K Flip-Flop Failure Test

APPENDIX B

KARNAUGH MAPPING

A Karnaugh Map is a simple visual representation of a 2-state function and is used to obtain a simplified Boolean expression from a truth table. To make a Karnaugh Map, a Boolean function must first be in a "sum of products" form. In most cases, functions will have to be simplified through truth tables to obtain this format. Examples of truth table simplification are shown in Figures 4.6 and 4.7 of Chapter 4. As a general rule, the basic variables (A, B, C, etc.) of a Boolean function are made into columns of the truth table. These variables are then combined into the successive levels of terms until finally the function itself is made into a column of the truth table. The following example (Figure 1) illustrates how a complex example can be reduced to a sum of products by a truth table. The function F is valid for all input combinations which give F a value of 1. The last column in the truth table indicates that the function $F = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot C \cdot \overline{D}$. This sum of products form of expression is now suitable for plotting on a Karnaugh Map.

There are 16 possible conditions for any 2-state function with four variables. Each one of the combinations of input variables is called a "minterm." The function will have a value of 1 or 0 for each minterm. The truth table will give the value of each minterm of the function. A Karnaugh Map takes information from a truth table and organizes it in visual grouping of minterms for easier simplification of the function.

$$F = \overline{A} \cdot \{ \overline{[(B+C) \cdot D]} \cdot \overline{A} \}$$

A	B	C	D	B + C	(B + C) · D	[(B + C) · D] · A	[(B + C) · D] · A	F
0	0	0	0	0	0	0	1	1(1)
0	0	0	1	0	0	0	1	1(2)
0	0	1	0	1	0	0	1	1(3)
0	0	1	1	1	1	1	0	0
0	1	0	0	1	0	0	1	1(4)
0	1	0	1	1	1	1	0	0
0	1	1	0	1	0	0	1	1(5)
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

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

(1)
(2)
(3)
(4)
(5)

Figure 1

A Karnaugh Map of the truth table in Figure 1 will indicate the value of each of the 16 minterms for the function F. A convenient way to do this is with a table 4 squares by 4 squares, where each square uniquely represents one minterm.

On the Karnaugh Map there is one square for each minterm.

3 variables produce 8 squares

4 variables produce 16 squares

5 variables produce 32 squares

(Mapping over 5 variables becomes cumbersome)

The map is laid out to produce easy recognition of terms that differ only in the barred and unbarred state of one variable. As an example, if $F = abc + abc$, the two terms differ only because one term contains c and the other c . This relationship is called an adjacency, and by application of the distributive law can be simplified to ab .

$$\begin{aligned} F &= abc + abc \\ &= ab(c + c) \\ &= ab \end{aligned}$$

This adjacency is readily seen on the Karnaugh Map. A diagram for a 3 variable map is illustrated below. Each column on the map represents

		(AB)	(AB)	(AB)	(AB)		
	AB	00	01	11	10		
Rows		0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}\bar{C}$	$A\bar{B}C$	Columns
		1	$\bar{A}B\bar{C}$	$\bar{A}BC$	$AB\bar{C}$	ABC	

a specific value of A and B. Each row represents a specific value of C. The first bit in each number group to the right of AB (column digits) indicates the value of A for a particular column.

0 = not function

1 = true function

The upper row contains minterms with C valued 0. The lower row contains minterms with C valued 1.

A diagram for variables abc is shown below. A term is illustrated by placing a one in an appropriate square.

		AB	00	01	11	10
C		0	0	1	0	0
		1	0	0	0	0

A diagram for variables abc is shown below.

		AB	00	01	11	10
C		0	0	0	0	0
		1	0	1	0	0

Combining the two maps readily shows the adjacency and produces a two variable term ab , since c and c is common to the adjacency.

Therefore, $abc + abc = ab$. The circle represents the simplified term. (Term cannot contain $c = c$.)

	AB	00	01	11	10
C	0	0	1	0	0
1	0	1	0	0	

Adjacency = ab.

A diagram used to plot four variables is shown below. The number to the right of AB represent all possible combinations of AB.

	AB	00	01	11	10
CD →	00	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B C\bar{D}$
	01	$\bar{A}\bar{B}C D$	$\bar{A}B C D$	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}B C\bar{D}$
	11	$\bar{A}\bar{B}C D$	$\bar{A}B C D$	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}B C\bar{D}$
	10	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}B C\bar{D}$	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}B C\bar{D}$

The numbers to the right of CD represent all possible combinations of CD. All minterms are plotted above.

RULES

In order to simplify any expression, the following rules should be followed.

1. Make as large a grouping of adjacent minterms as possible. (Grouping must contain a number of minterms which is an even power of 2. i.e. 2, 4, 8, 16, etc.)

	AB	00	01	11	10
CD →	00	0	0	1	1
	01	0	0	1	1
	11	0	0	1	1
	10	0	0	1	1

8 SQUARE ADJACENCY

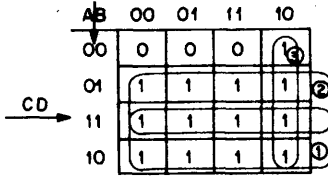
2. Overlapping groups is desirable to reduce the expression to its simplest form.

	AB	00	01	11	10
CD →	00	1	1	1	1
	01	0	0	1	1
	11	0	0	0	0
	10	0	0	0	0

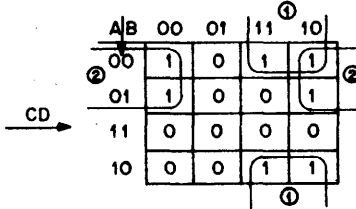
① ②

$$\bar{A}\bar{C}\bar{D} + A\bar{C}\bar{D} + A\bar{C}D = \bar{C}\bar{D} + A\bar{C}$$

3. A function may be defined by grouping all the minterms valued 1. (Or by grouping all the minterms valued 0 and negating.)

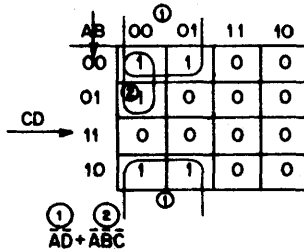


4. Determine external adjacencies by "folding" the map so that the outer sides touch or the top and bottom touch.

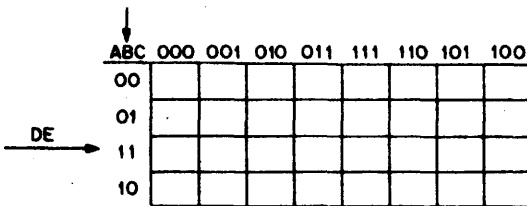


$A\bar{C}\bar{D} + A\bar{C}D + \bar{A}B\bar{C} + \bar{A}B\bar{C}D$ PLOTTED BECOMES $\bar{A}D + \bar{B}C$

Referring back to Figure 1. $F = ABCD + ABCD + ABCD + ABCD + ABCD$ is plotted and simplified as follows:



A map for five variables is illustrated below:



APPENDIX C

THE TRANSISTOR-TRANSISTOR LOGIC NAND GATE

The same basic electronic configuration is used throughout the logic functions on the Computer Lab. In this section the operation of that configuration will be studied and the methods used to make more complex gates examined.

Figure 1 is the basic two input NAND gate schematic diagram. The circuit is divided into 3 major sections, the multiple emitter input, the phase splitter and the totem pole output circuit. The two diode model of a transistor shown in Figure 2 will be used in the analysis of the circuit. A forward biased silicon junction (i.e. diode) gives a voltage drop of about 0.75 volts and a saturated silicon transistor has a collector emitter voltage of 0.4 volts average. These two figures will be used throughout the following discussion.

With either input at the LO logic level (0.0V-0.8V) the multiple emitter input transistor will be ON with its base residing at about $0.75 + 0.4 = 1.15$ volts. The three diode string consisting of Q_1 's base collector diode, Q_2 's base emitter diode, and Q_4 's base emitter diode will have only 1.15 volts across it and will therefore be conducting only leakage currents ($0.75 + 0.75 + 0.75 = 2.25$ volts required for forward bias). With no current flowing into the base emitter junction of Q_2 , the transistor will be OFF and its collector emitter voltage is allowed to rise. Similarly with no current flowing in the base emitter diode of Q_4 the transistor is OFF and its collector emitter voltage is allowed to rise. When both Q_2 and Q_4 are OFF, Q_3 is freed to pull the output voltage to a HI level. The voltage levels present in the circuit with one or more LO inputs is shown in Figure 4.

If both inputs are HI (2.4-3.6 volts) the head of the three diode string will reside at about 2.25 volts and there will be a current path from the 4K base resistor on the input transistor through the diode string to ground as shown in Figure 5. With current flowing in the base emitter junctions of both Q_2 and Q_4 , both transistors will be turned ON. Q_3 is held OFF whenever Q_2 is ON. The output is driven LO (0.0V-0.4V) by transistor Q_4 . The voltage levels present in the circuit with both inputs HI and are shown in Figure 6.

MULTIPLE INPUT GATES, THE AND/NOR GATE

More inputs are added to a NAND gate by increasing the number of emitters on the multiple emitter input transistor. The AND/NOR gate circuit has an extra input transistor and phase splitter transistor. The phase splitter of the extra section is connected in parallel with the basic NAND phase splitter as shown in Figure 7.

LOADING AND DRIVE CAPABILITIES

To take an input to ground (logical LO) requires a source capable of sinking 1.6 ma. (In calculating this figure recall that the 4K input resistor has a tolerance of 30%.) To take an input to + 3 volts (logical HI) requires a source capable of delivering 40 μ a. This combination of drive requirements is called a unit load. Each gate output is capable of driving 10 unit loads, or in other words each output has a fan-out of 10.

LOGIC LEVELS AND NOISE MARGIN

A gate input will recognize 0.0 volts to 0.8 volts as logical LO and 2.0 volts to 3.6 volts will be recognized as a logical HI. An output is between 0.0 volts and 0.4 volts in the logical LO condition. The logical HI output condition is

between 2.4 volts and 3.6 volts. Figure 7 shows diagrammatically the acceptable transistor-transistor logic levels. The worst case noise margin is 400 millivolts that is, an output would have to make at least a 400 millivolt excursion to cause an input which is connected to it to go into the indetermined voltage region. For instance if an output were at 0.4 volts (worst case logical LO) there would have to be a + 400 mv swing in voltage to cause inputs connected to it to go into their indetermined region.

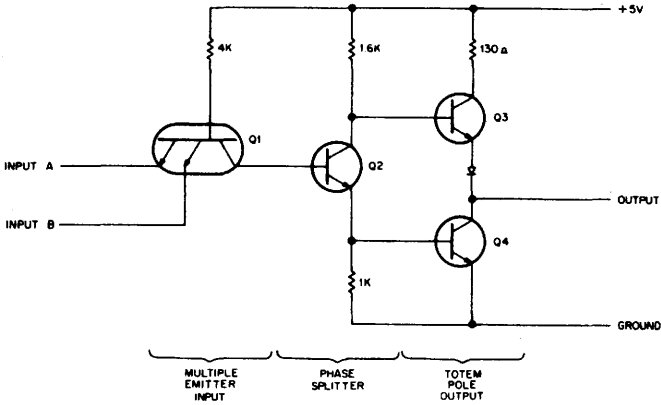


Figure 1 TTL NAND Gate Schematic Diagram

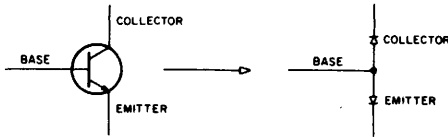


Figure 2 Two Diode Model For Transistor

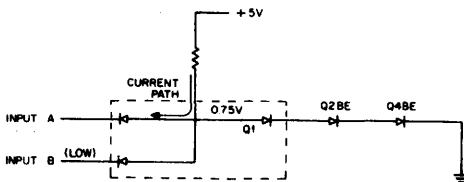


Figure 3 Diode Equivalent NAND Gate Circuit, One Input LO

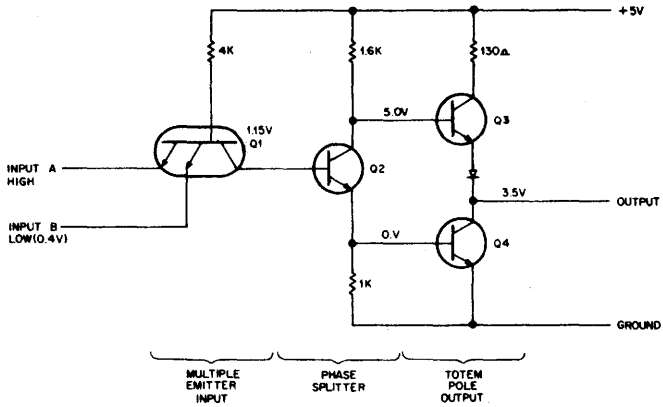


Figure 4 TTL NAND Gate Schematic Diagram, One Input LO

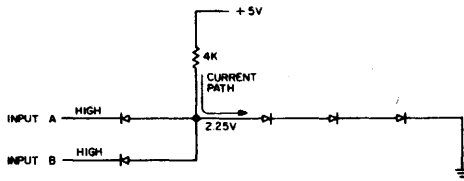


Figure 5 Diode Equivalent NAND Gate Circuit, Both Inputs HI

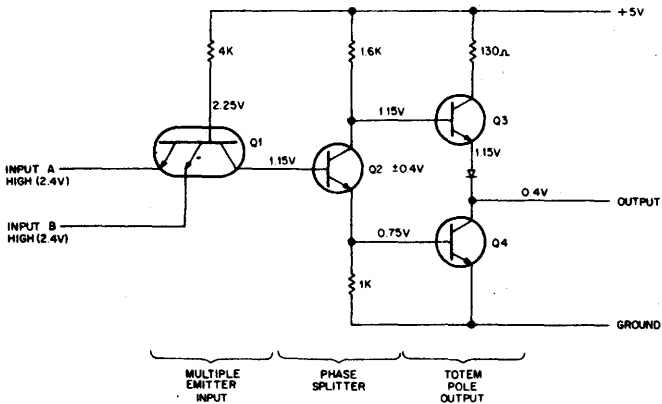


Figure 6 TTL NAND Gate Schematic Diagram, Both Inputs HI

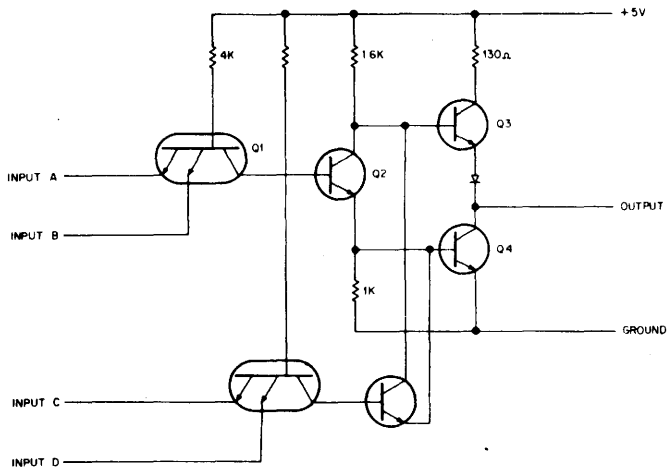


Figure 7 AND/NOR Gate Circuit

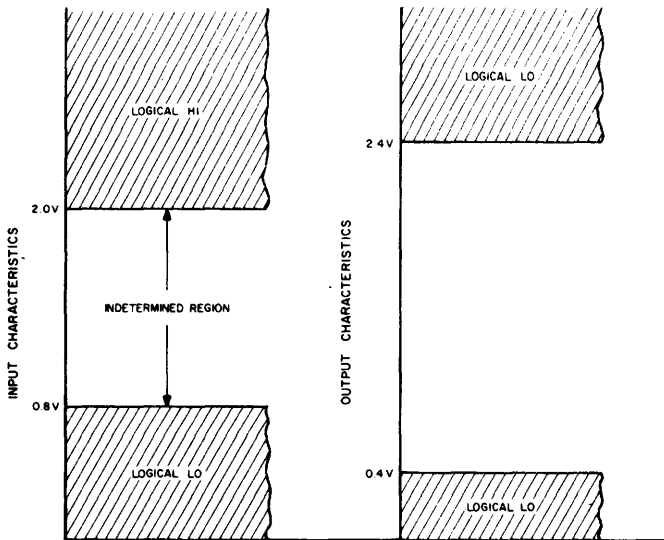


Figure 8 Logic Levels

OUTPUT CONNECTIONS

When outputs are mistakenly connected to outputs, *indetermined* levels often result because the transistor-transistor logical output circuit is a low impedance in both the logical LO and logical HI condition. Other forms of logic (e.g. diode-transistor logic) have a low impedance output in the LO condition and a high impedance output in the HI condition. Because of the varying output impedance in the two states, outputs can be paralleled in these forms of logic to give what is called the wired OR function. This cannot be done with the transistor-transistor logic on the COMPUTER LAB; outputs must be connected only to inputs.

APPENDIX D

THE COMPUTER

I INTRODUCTION

The modern digital computer is very similar to the standard desk calculator with one very basic difference: the computer performs long chains of operations without human intervention. The computer can also make certain logical "yes" or "no" decisions and change its future actions as a result. The chain of instructions which tells the computer how to solve a particular problem is called a "program." Preparing the list of operating instructions for the computer is called "programming," and the people who do this, "programmers."

The computer is a machine that has to be taught step by step exactly what to do. It can do only one step at a time, but it does each step at such a great speed that it appears as if the computer were doing an entire large scale operation at once. For example, the DIGITAL PDP-8/S computer can add two numbers in 36 microseconds—that is, in 36 one-millionths of a second. At this speed, it can perform almost 28,000 additions in one second. An even faster computer, the DIGITAL PDP-8/I, requires only 3 microseconds to perform one addition. It can perform over 300,000 additions in one second.

The computer can remember both facts and instructions. This stored information can be brought into use in a very small fraction of a second and, unlike the human memory, the computer never forgets. It can instantly remember every fact it has been told. Because of its speed and memory, the computer can provide great accuracy by solving problems the long and most accurate way.

The computer has two prime capabilities that let it work automatically: it can remember a set of instructions and it can perform these instructions in proper order without outside direction. A computer operator need only feed instructions and data into the computer. The computer works automatically, step by step, solving the problem.

II PARTS OF A COMPUTER

The working parts of the computer can be grouped into five major categories: A), input devices; B), arithmetic or calculation unit; C), memory or storage devices; D), output devices; and E), control unit.

A) Input Devices

Input devices supply the computer with basic data and also give the instructions or program that tell it what to do with the basic data. Information can be fed into the computer by a variety of means. The slowest method is manually setting a series of switches. Each switch has a certain meaning to the computer. A faster method is by the use of punched cards. Each hole in the card has a certain meaning to the computer as either a fact or an instruction that tells the computer what to do with a fact. A faster method still is with magnetic tape, similar to the tape used in tape recorders. The computer senses information on the tape and interprets this as facts or operating instructions. A still faster method is through dataphone or an electronic hookup from one computer directly to another.

B) Arithmetic Unit

The computer's calculating or arithmetic unit operates on the same principle as an adding machine. It can take two numbers and add, subtract, multiply or divide them.

C) Memory

The computer has a memory in which it stores both the data needed to solve a problem and instructions that tell it how to manipulate the data.

D) Output Devices

After the computer has finished a set of instructions and arrived at a solution, it uses various output devices to bring this information to the outside world. These devices are very similar to the input mechanisms described earlier. Output information can be punched on cards; it can be displayed by patterns of lights; it can be recorded on magnetic tape or printed on paper. The information can also be transferred from one computer to another via data-phone or it can be displayed on an oscilloscope, a device very similar to a television.

E) Control Unit

All of the elements of a modern electronic computer must work in a definite series of operations. The input and output devices must function at the proper time; the arithmetic unit must operate when the proper numbers are in it; and the storage or memory element must transfer information in and out in the proper sequence. All this is controlled by the computer's central processor or control unit. The operating instructions held in memory tell the central control unit the order in which the various parts of the computer must operate and what they must do. The unit then coordinates all parts of the computer so that events happen in the proper sequence and at the right time.

This is neither as powerful nor as mysterious as it may sound, for the control unit only does exactly what it is told to do. Numerically encoded instructions which are stored in the memory can be sent to the control unit to direct it to carry out certain basic operations. The control unit is designed to "decode" each number sent to it and to begin a chain of events designated by that number. For example, the instruction code number 7001₈ might start a counting operation in an arithmetic unit. When one chain of events has been completed, the control unit is ready to receive another number from the memory and to begin the chain of events designated by that number and so on.

Relatively few of the instructions which the control unit can interpret are actually built into the computer but there are built-in instructions which make it possible, when combined in the proper sequence, to form any specific chain of operations which possibly could have been built in. This remarkable circumstance gives the computer its versatility. This type of computer—one that stores its own instructions and provides the means of creating an indefinite array of operating sequences—is called a general purpose computer. It is the set of instructions, or program, that states the procedure the computer is to follow in solving the problem at hand. The program is a large series of operations composed of many simpler operations, namely the basic ones provided in the computer's repertoire. In using the computer to solve a problem, the operator's task is primarily one of writing an effective program based ultimately on the simple operations the computer "knows" how to do.

III WHAT A COMPUTER DOES

A computer can work automatically because it can remember instructions as well as facts. The computer's memory is divided into a number of addresses or locations. A group of digits is stored at each address and each group is handled by the computer as a unit. This unit is generally called a "word." Each memory address is numbered and the information at the address is then referred to by that number. For example, address [500] could contain the number 7,025 or address [18] could contain an operating instruction. The memory words can be either data or operating instructions, either of which can be stored at any memory address. Operating instructions are stored in the computer as a group of digits.

To illustrate how the various parts of a computer operate, think of one being used to solve the algebraic equation " $y = a + b$ " where $a = 2$ and $b = 3$. The instructions the computer will need are ERASE, ADD, STORE, PRINT, and STOP. ERASE tells the computer to remove all previous numbers from the Arithmetic Unit and start at zero. ADD instructs it to take a number stored in a memory location and add it to whatever number is in the Arithmetic Unit. The STORE instruction takes the number currently in the Arithmetic Unit and stores it in a specific memory location. At the PRINT command, the computer will print out the information it is holding in the specified memory location. STOP tells the computer that it has finished a set of commands and should return ready to FETCH the instruction at location [1] and wait for further instructions.

For solving $y = a + b$, the computer will need nine memory locations, as shown in Figure 1, with the first six holding operating instructions, location [7] holding the value of a , location [8] holding the value of b and location [9] holding the final answer, or the value of y .

ADDRESS	CONTENTS
[1]	ERASE
[2]	ADD [7]
[3]	ADD [8]
[4]	STORE [9]
[5]	PRINT [9]
[6]	STOP
[7]	2
[8]	3
[9]	0

$$\begin{aligned}y &= a + b \\ \text{where} \\ a &= 2 \\ b &= 3\end{aligned}$$

Figure 1 Memory Addresses and Contents Before Program Starts

There are two parts to every computer command: the FETCH instruction and the EXECUTE instruction. During the first part of each command, the Central Control Unit goes to the next memory address and 'fetches' the instruction. In the second part of the command, the Control Unit 'executes' the instruction it received.

For example, in Figure 2A, the Control Unit fetches the instruction ERASE from memory address [1]. It then executes the instruction as in Figure 3B by erasing the previous contents of the Arithmetic Unit, in this case 108.

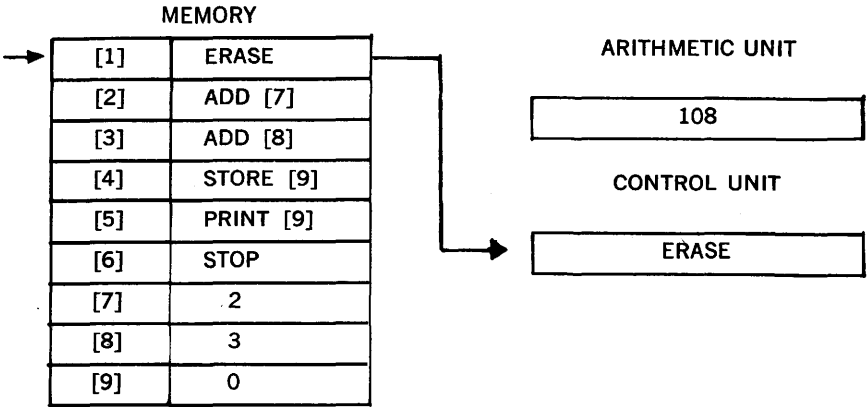


Figure 2A Fetch Instruction

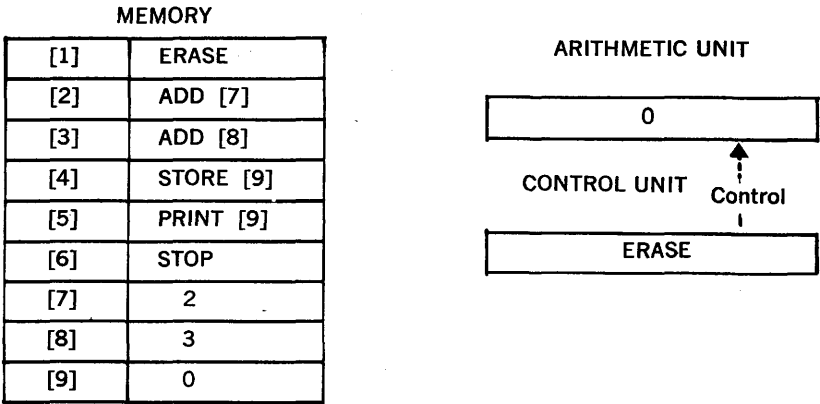


Figure 2B Execute Instruction

The Control Unit then moves to the next address location (Figure 3A) where it first fetches the instruction ADD [7] telling it to add the contents of address [7] to the number in the Arithmetic Unit (0) and it then executes the instruction (Figure 3B).

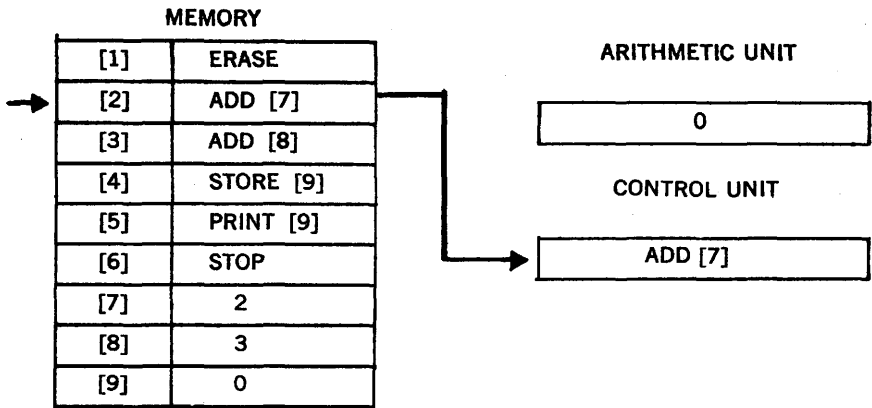


Figure 3A Fetch Instruction

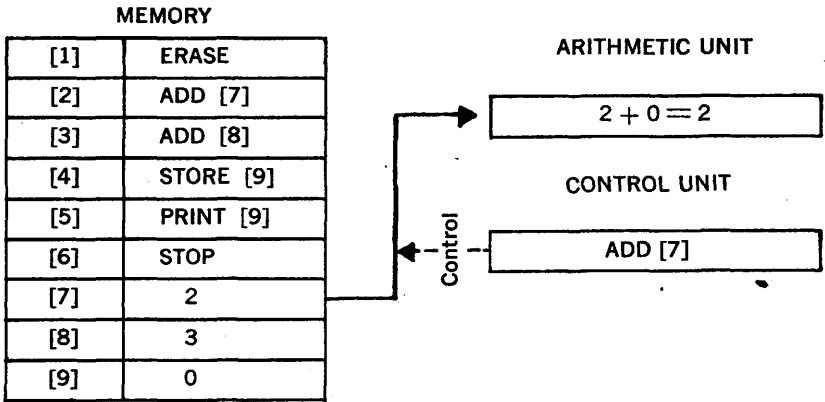


Figure 3B Execute Instruction

The next command, illustrated in Figures 4 A and 4 B, tells the computer to add the contents of address [8] to the number in the Arithmetic Unit. The unit now holds the value of $a + b$.

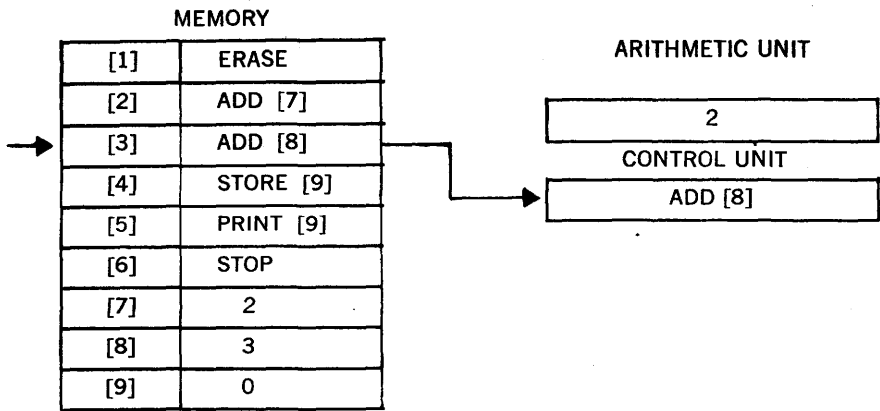


Figure 4A Fetch Instruction

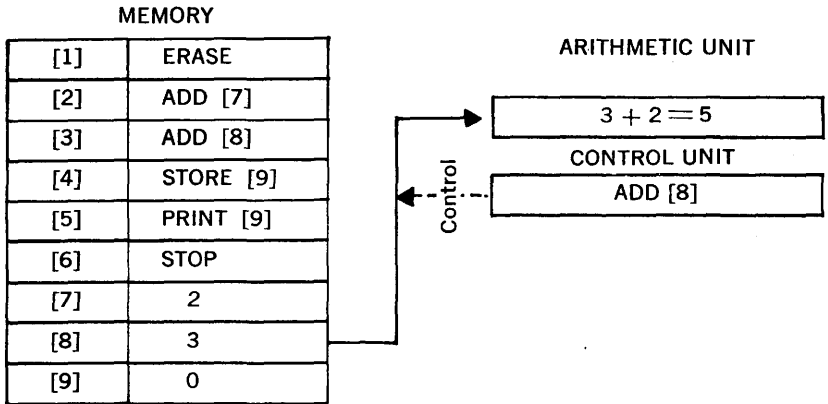


Figure 4B Execute Instruction

The computer has now solved the equation $y = a + b$ for the value of y where $a = 2$ and $b = 3$. The next task is to get this information out of the computer. Figures 5A and 5B illustrate one method of doing this. The Control Unit reads the contents of Memory Address [4] and is instructed to take the information in the Arithmetic Unit and store it in Memory Address [9], erasing any information already in [9].

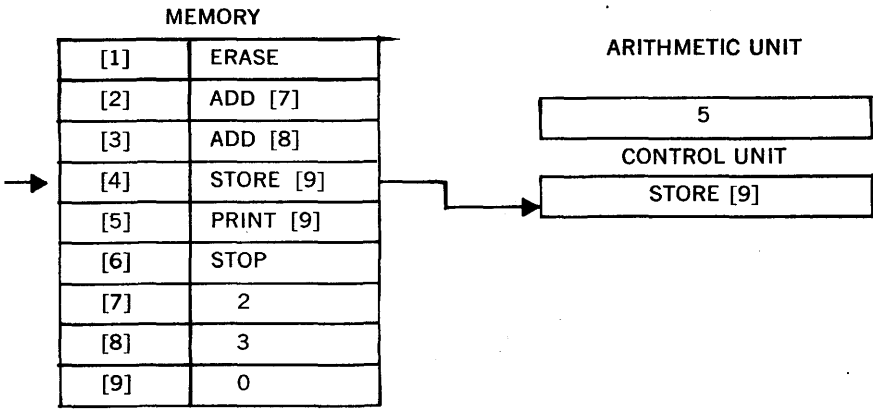


Figure 5A Fetch Instruction

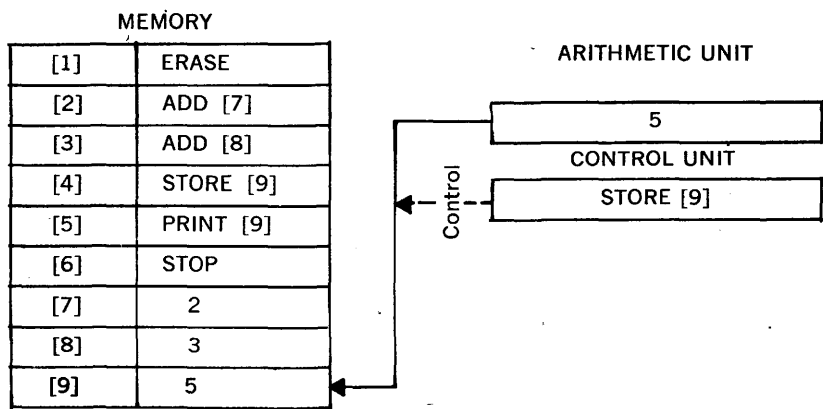


Figure 5B Execute Instruction

The next instruction (Figures 6A and 6B) tell the computer to print out the information stored in Memory Address [9].

MEMORY

[1]	ERASE
[2]	ADD [7]
[3]	ADD [8]
[4]	STORE [9]
[5]	PRINT [9]
[6]	STOP
[7]	2
[8]	3
[9]	5

ARITHMETIC UNIT

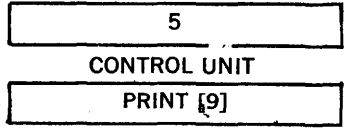
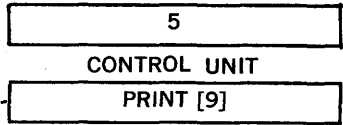


Figure 6A Fetch Instruction

MEMORY

[1]	ERASE
[2]	ADD [7]
[3]	ADD [8]
[4]	STORE [9]
[5]	PRINT [9]
[6]	STOP
[7]	2
[8]	3
[9]	5

ARITHMETIC UNIT



PRINTER

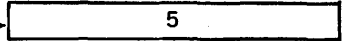


Figure 6B Execute Instruction

After executing the instruction held in memory address [5], the control unit advances to address [6] where it is told to stop, return to [1] and await a new start command. The contents of each memory address remain unchanged and new values for a and b will have to be read into addresses [7] and [8] before the control unit is started again. When told to start, the computer will go through the sequence of instructions, again solving the equation $y = a + b$ for the values of a and b. The values of a and b could be changed by manually "keying" new numbers into [7] and [8] or instructions could be added to the program which would allow the computer to accept new values for a and b from various input devices. (See Figures 7 A and 7 B)

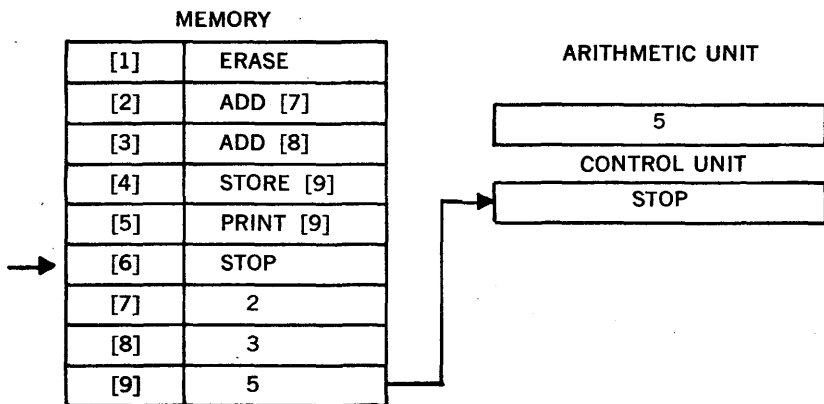


Figure 7A Fetch Instruction

1. Computer Halts
2. Returns to Address 1
3. Waits for New Values for a and b
4. Waits for Start Command

Figure 7B Execute Instruction

IV PROGRAMMING

In solving any problem, the computer cannot operate until it has been given a detailed set of instructions—the program. It can only follow instructions step by step until the task is finished. The computer functions this way for any problem it is given, whether simply adding a column of numbers or solving the complex formulae expressing planetary motion.

Before writing a program, the problem must first be completely analyzed and divided into its simplest components. One method for doing this is to construct a flow chart of the problem—a diagram showing each part of the total problem in its proper relationship to every other part. To actually work within a computer, each part of a normal flow chart usually is broken down into many smaller instructions. One part might require 35 or more individual steps within the computer.

The flow chart illustrated in Figure 8 shows the steps involved in packing oranges into a crate. This program is slightly more complicated than the example explained previously, for it requires the computer to make a decision. The boxes in the chart represent work to be done and the diamond represents a choice the computer must make. As each orange is packed into the crate, the Arithmetic Unit is increased by 1. The computer then asks if the number held in the Arithmetic Unit is equal to, or less than, 100. If it is equal to 100, the program proceeds forward, oranges are no longer added, the crate is packed, the Arithmetic Unit returns to zero, a new crate is brought into position and the program starts over again. If the count has not yet reached 100, the program flows to the left, adding additional oranges until the count does total 100.

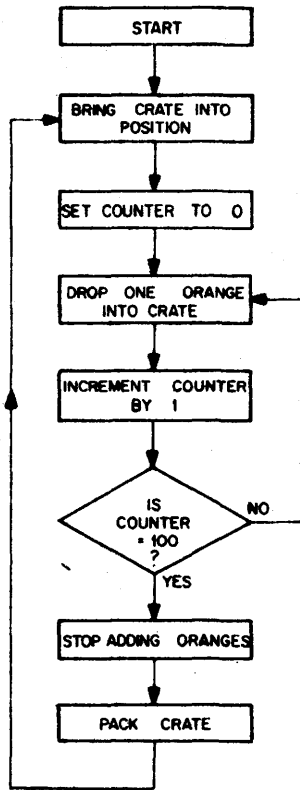


Figure 8 Flow Chart

The action of skipping part of the program or choosing which of two sets of instructions to follow is called "branching." The computer decides which to follow by asking whether the count is equal to zero. It makes a simple yes/no decision.

However, even in making decisions, the computer is simply following instructions. It is the programmer who originally wrote the program who had to decide when a decision was needed and what data would determine which choice is taken. Once the program has been detailed in a flow chart and resolved into simple operations, operating instructions are written telling the computer how to handle each task.

The computer, however, does not understand English: the instructions must be written in a binary code, intelligible to the computer. The computer only understands and operates upon a code composed of 1's and 0's. The first experiment in the COMPUTER LAB Workbook explains what the binary code is and how it operates. The programmer, however, does not have to write his

program directly in binary code because special "symbolic languages" which closely resemble English are available. The program is written in a symbolic language which is automatically converted into binary by the computer. For example, by typing in the symbol for "ADD," the computer will automatically translate ADD into the correct machine (binary) instruction, which might be 1001001. The computer can automatically translate because someone previously had written a special program called an "Assembler" which tells the computer how to translate symbolic languages into its own machine language. Once the symbolic program has been converted to binary, it moves as electronic impulses into the computer's memory.

V HOW A COMPUTER OPERATES

Inside the computer are thousands of wires connecting the different parts of the computer just as telephone lines crisscross the country from one end to the other, connecting every telephone with every other telephone. Also, within the computer are thousands of electronic devices that function as switches which can be opened or closed to form a direct line between two specific parts of the machine. When the computer reaches an instruction that tells it, for instance, "ADD the contents of address [200]," the electronic switches set a direct line between address [200] and the Arithmetic Unit. When the line is complete, electronic pulses—each conveying one bit of information—move from the memory location to the Arithmetic Unit. Because these pulses follow one another, they are often called a "pulse train." This action is very similar to the operation of the telephone network. When the number is dialed, the dial converts the number to a series of electronic pulses. These pulses move a set of mechanical switches called relays which set up a direct line between two telephones. Although the telephone system consists of millions of telephones and millions of interconnected wires, a completed telephone call connects just two telephones.

The completed connections are called circuits. In the computer, different circuits perform many types of operations. They may transfer numbers out of an address in the computer's memory and deposit them in the arithmetic unit. Then another series of circuits will cause this unit to add. Finally, circuits may send information out of the computer perhaps by punching holes in cards or paper tape. The computer operates on a series of electronic pulses. It is because the electronic switches can open and close so rapidly moving pulses through the machine at rates up to one billion per second that the computer has such fantastic speed.

APPENDIX E

GLOSSARY OF TERMS

ACCUMULATOR—An accumulator register is one used to store the sum of an addition in a binary adder.

ANALOG—A signal which is continuously variable and unlike a digital signal does not have discrete levels.

AND/NOR GATE—The AND/NOR Gate is a single logic element whose operation can be interpreted by 2 AND gates with outputs feeding into a NOR gate. Since this is a single logic element no access is provided for the internal logic elements (i.e. no connection is provided at the output of the AND gates).

ASYNCHRONOUS—An asynchronous device is one which does not have all elements of that device operating at the same time. For example, an asynchronous counter does not have all bits of the counter operating simultaneously. The information which causes a bit to complement in an asynchronous counter must ripple through all less significant bits.

BINARY—The binary number system is one which has only two states. "0" and "1" are the two binary digits.

BINARY CODED DECIMAL—Four or more bits of binary information can be used to encode one decimal digit. When a decimal digit is encoded in this way it is called a Binary Coded Decimal (BCD).

BIT—The words "binary digit" are often abbreviated to BIT.

CARRY—In performing binary additions one bit of information often has to be carried from one section of the addition to the next most significant section. This bit of information is called a "carry bit." The carry flip-flop in a serial adder stores the carry information of one addition and presents it to the next most significant addition.

CLOCK—The clock in a digital system is used to provide a continuous train of pulses. The clock on the COMPUTER LAB provides pulses which are 50 nanoseconds wide and whose repetition rate can be varied from less than 1 pulse per second to more than 10×10^6 pulses per second.

CLOCKED R-S FLIP-FLOP—The clocked R-S flip-flop has two conditioning inputs which control the state to which the flip-flop will go at the arrival of the clock pulse. If the S (Set) input is enabled, the flip-flop goes to the "1" condition when clocked. If the R (Reset) input is enabled, the flip-flop goes to the "0" condition when clocked. The clock pulse is required to change the state of the flip-flop.

COMPARATOR—A comparator is a device used to determine if two bits of information are equal.

COMPLEMENT—The complement of a variable or function is the binary opposite of that variable or function. If a variable or function is 1, its complement will be 0. If a variable or function is 0, its complement will be 1. The complement of 011010 is 100101.

CONTROL—The control in a digital system is a device used to provide a sequence of levels and/or pulses which will cause a system or subsystem to carry out a sequence of operations.

COUNTER—The counter is a device which will maintain a continuous record of the number of pulses which it has received at its input. The output of the counter indicates the sum of the number of input pulses.

D-TYPE FLIP-FLOP—A D-type flip-flop will propagate whatever information is at its D (data) conditioning input prior to the clock pulse, to the 1 output on the leading edge of a clock pulse.

DECODER—A decoder is a device used to convert information from a coded form into a more usable form (i.e., binary-to-decimal decoder).

DIGIT—A digit is one character in a number. There are 10 digits in the decimal number system. There are two digits in the binary number system.

ENABLE—A gate is enabled if its input conditions result in a specified output. The specified output varies for different gating functions. For instance, an AND gate is enabled when its output is HI and the NAND gate is enabled when its output is LO. Sub-systems described in later experiments have function enabling inputs which allow operations to be executed on a clock pulse after the function enabling input is enabled with the correct logic level.

ENCODER—An encoder is a device which takes information in one code and encodes it into another (e.g., the decimal to binary encoder).

END AROUND CARRY—The end around carry operation adds the carried information from the left-most bit to the results of the right-most addition. End around carry is used for 1's complement and 9's complement arithmetic.

EXCLUSIVE OR—The Exclusive OR function is valid, or its value is 1, if one and only one of the input variables is present. The Exclusive OR applied to two variables is present or 1, if the 2 binary input variables are different. The Exclusive OR applied to 2 input variables is often also called half-add.

FAN-OUT—The fan-out of an output is a number which indicates the number of the unit loads an output can drive.

FLIP-FLOP—A flip-flop is a storage device which can be used to retain one bit of information. A flip-flop can be in the "1" state or the "0" state. In the "1" state, its 1 output presents a HI level and its 0 output presents a LO level. In the "0" state, its 1 output presents a LO level and its 0 output presents a HI level.

FUNCTION—A relationship is a function if, and only if, for every combination of input conditions there is one unique output. (E.g., the AND function will be valid or present if all the input variables examined are valid).

GATE—A gate is a device whose operation can be defined by a binary logic function. Electronic gates are provided on the COMPUTER LAB. However, gates can be constructed using other types of devices (hydraulic, mechanical, etc.).

GRAY CODE—The Gray code is the reflective binary counting code which changes only one bit at a time when incrementing or decrementing by 1.

GROUND—Ground is the reference or base level from which all voltages are measured on the COMPUTER LAB. The ground terminals on the patch panel are to be used only for inter-connection of two or more COMPUTER LABS for large logic circuits.

HALF-ADD—The half-add operation is performed first in doing a two-step binary addition. It adds corresponding bits in two binary numbers ignoring any carry information.

HI—HI is used throughout this Workbook as an abbreviation for the logic level HIGH. The corresponding voltage range for a HI level is 2.4 volts to 3.6 volts.

INCIDENT—The Incident Register in an adder is used to accept numbers from the outside world.

INVERT—To invert a function or variable is to change the value of that function or variable to a 0 if it is 1, or to a 1 if it is 0.

INVERTER—An Inverter is a device which performs the invert operation. It will present at its output the inverse or complement of the information at its input.

J-K FLIP-FLOP—A J-K flip-flop has two conditioning inputs and one clock input. If both conditioning inputs are disabled prior to a clock pulse, the flip-flop will remain in its present condition when a clock pulse occurs. If the J input is enabled (HI) and the K input is disabled (LO), the flip-flop will go to the 1 condition on a clock pulse. If the K input is enabled, and the J input is disabled, the flip-flop will go to the 0 condition on a clock pulse. If both the J and K inputs are enabled prior to a clock pulse, the flip-flop will complement or go to the opposite state on a clock pulse. The J-K flip-flops used on the COMPUTER LAB are master-slave devices and, therefore, perform transitions on the trailing edge of a HI clock pulse.

LEADING EDGE—The leading edge of a pulse is defined as that edge or transition which occurs first. (i.e., the leading edge of a HIGH pulse is the LO to HI transition.)

LEVEL—A level is a voltage which remains constant for a long time. There are two possible levels in the COMPUTER LAB: HI or LO.

LO—LO is used throughout this Workbook as an abbreviation for the logic level LOW. The corresponding voltage range for a LO level is 0.0 volts to 0.4 volts.

LOGIC—Logic is a form of mathematics based upon two-state truth tables. Electronic logic uses two-state gates and flip-flops to perform decision making functions.

MASTER-SLAVE—A master-slave flip-flop is one which contains two flip-flops, a master flip-flop and a slave flip-flop. A master flip-flop receives its information on the leading edge of a clock pulse and the slave or output flip-flop receives its information on the trailing edge of the pulse.

MODULUS—The modulus of a counter describes the number of distinct states which that counter has. (e.g., the modulo 10 counter has a modulus of 10 and therefore has 10 distinct states).

MODULO N COUNTER—A modulo N counter has N unique states.

NAND GATE—A NAND gate is enabled when both its inputs are present or HI. When a NAND gate is enabled, its output is LO. The term NAND is a contraction of the two words NOT AND.

NEGATE—To negate a binary function or variable is to change the value of that function or variable to 1 if it is 0, or to 0 if it is 1. The symbol for negation is superscript bar (-).

NEGATED INPUT OR GATE—A negated input OR gate is enabled if one input or the other or both are LO. When enabled, the output of the gate is HI. A NAND gate is identical to a negated input OR gate in function. The difference between a NAND gate and a negated input OR gate is merely in the way that the operation of that gate is interpreted.

NINE'S COMPLEMENT—Nine's Complement arithmetic provides a method of negating a decimal number so that subtraction can be performed using addition techniques. The 9's complement negation of a number is obtained by subtracting each decimal digit individually from 9. In performing a 9's complement addition, end around carry must be used.

NOR GATE—A NOR gate is enabled when one or more of its inputs are enabled or HI. When enabled, the output of a NOR gate is LO. The word NOR is a contraction of the two words NOT OR.

OCTAL—The octal number system is one which has 8 distinct digits—namely, 0, 1, 2, 3, 4, 5, 6, 7.

ONE'S COMPLEMENT—One's Complement arithmetic provides a method of negating a binary number so that binary subtraction can be performed using addition techniques. To obtain the 1's complement of a binary number, all bits in that number must be complemented. In performing 1's complement addition, end around carry must be used.

PARALLEL ADDITION—Parallel Addition operates on each set of corresponding digits simultaneously in the addition of two numbers.

PARITY—Parity is a method by which binary numbers can be checked for accuracy. An extra bit, called a parity bit, is added to numbers in systems using parity. If even parity is used, the sum of all 1's in a number and its corresponding parity bit is always even. If odd parity is used, the sum of 1's in a number and its corresponding parity bit is always odd.

PROPAGATION DELAY—The propagation delay of an electronic digital device is the time which is required to transfer information from its input to its output.

PULSE—A pulse is a voltage which goes from one level to another, remains there for a short time and then returns to the original level. A HI pulse is one which goes from LO to HI for a short time and then returns to LO. A LO pulse is one which goes from HI to LO for a short time and then returns to HI.

PULSE WIDTH—The width of a pulse is defined as the length of time for which the pulse voltage is at the second, or transient, level.

RECYCLING MODULO N COUNTER—A recycling modulo N counter is one which has N distinct states and which counts to a maximum number and recycles, on the next input pulse, to its minimum number.

REFLECTIVE CODES—Reflective counting codes are codes which appear to be the mirror image of normal counting codes. Their most useful property is that only one digit changes at a time in incrementing or decrementing by 1. The reflective binary code is called the GRAY code.

RESET—If a Reset input to a flip-flop is enabled, the flip-flop will go to the "0" condition.

RING COUNTER—A ring counter is a device capable of storing several bits of information. A ring counter will accept shift instructions which will shift all the information one position at a time. If information is shifting left in a register, the value of the left-most bit will shift into the right-most bit of the register. Similarly, if the ring counter is shifting right, the value of the right-most bit will shift into the left-most bit in the register. The information will recycle in a ring counter every N shift pulses where N is the number of bits in the ring counter. A SWITCH TAIL RING COUNTER will contain the complement of the information it initially contained after N clock pulses and it will contain the same information as it started with initially after 2N clock pulses.

R-S FLIP-FLOP—The R-S flip-flop has two inputs, a Set input and a Reset input. If the Set input is enabled (HI), the flip-flop goes to the "1" condition. If the Reset input is enabled (HI), the flip-flop goes to the "0" condition.

SELF-STOPPING MODULO N COUNTER—A self-stopping modulo N counter has N distinct states and will stop when it reaches a predetermined maximum number. It will not accept further count pulses until it is reset to a number less than the maximum number.

SERIAL ADDER—A serial adder is one which performs additions in a series of steps. The least significant addition is performed first and progressively more significant additions are performed until finally the sum of the two numbers is obtained.

SET INPUT—When the Set input to a flip-flop is enabled, the flip-flop goes to the "1" condition.

SHIFT REGISTER—A shift register can contain several bits of information. When a shift instruction is received, all the information in that register is shifted one position.

SIGN BIT—When using complementary arithmetic, the left-most bit in a number is called the sign bit. If the sign bit is a 1, the number is negative. If the sign bit is a 0, the number is positive.

SYNCHRONIZE—To synchronize a level or a pulse is to make sure that that level or pulse is presented to a system or subsystem at the correct time.

SYNCHRONOUS—A synchronous device or subsystem is one which has all changes occurring simultaneously. For instance, a synchronous counter is one which has all required bit changes taking place at the same time.

TEN'S COMPLEMENT—Ten's Complement arithmetic is used to perform decimal subtractions using addition techniques. The 10's complement negative of a number is obtained by subtracting each digit in the number individually from 9 and adding 1 to the result.

TRAILING EDGE—The trailing edge of a pulse is that edge or transition which occurs last. The trailing edge of a HI clock pulse is the HI to LO transition.

TWO'S COMPLEMENT—Two's Complement is a form of binary arithmetic which is used to perform binary subtractions using addition techniques. The 2's complement negative of a binary number is obtained by complementing each bit in that binary number and adding 1 to the result.

UNIT LOAD—All COMPUTER LAB inputs impose a load on the outputs driving them. A unit load requires 1.6 ma at ground and + 40 μ a at + 3 volts. The load imposed upon an output by an input can be defined as a number of unit loads.

APPENDIX F

DECIMAL EQUIVALENTS OF BINARY NUMBERS TO 2⁸

Decimal	Binary	Decimal	Binary	Decimal	Binary	Decimal	Binary
0	00000000	45	00101101	90	01011010	135	10000111
1	00000001	46	00101110	91	01011011	136	10001000
2	00000010	47	00101111	92	01011100	137	10001001
3	00000011	48	00110000	93	01011101	138	10001010
4	00000100	49	00110001	94	01011110	139	10001011
5	00000101	50	00110010	95	01011111	140	10001100
6	00000110	51	00110011	96	01100000	141	10001101
7	00000111	52	00110100	97	01100001	142	10001110
8	00001000	53	00110101	98	01100010	143	10001111
9	00001001	54	00110110	99	01100011	144	10010000
10	00001010	55	00110111	100	01100100	145	10010001
11	00001011	56	00111000	101	01100101	146	10010010
12	00001100	57	00111001	102	01100110	147	10010011
13	00001101	58	00111010	103	01100111	148	10010100
14	00001110	59	00111011	104	01101000	149	10010101
15	00001111	60	00111100	105	01101001	150	10010110
16	00010000	61	00111101	106	01101010	151	10010111
17	00010001	62	00111110	107	01101011	152	10011000
18	00010010	63	00111111	108	01101100	153	10011001
19	00010011	64	01000000	109	01101101	154	10011010
20	00010100	65	01000001	110	01101110	155	10011011
21	00010101	66	01000010	111	01101111	156	10011100
22	00010110	67	01000011	112	01110000	157	10011101
23	00010111	68	01000100	113	01110001	158	10011110
24	00011000	69	01000101	114	01110010	159	10011111
25	00011001	70	01000110	115	01110011	160	10100000
26	00011010	71	01000111	116	01110100	161	10100001
27	00011011	72	01001000	117	01110101	162	10100010
28	00011100	73	01001001	118	01110110	163	10100011
29	00011101	74	01001010	119	01110111	164	10100100
30	00011110	75	01001011	120	01111000	165	10100101
31	00011111	76	01001100	121	01111001	166	10100110
32	00100000	77	01001101	122	01111010	167	10100111
33	00100001	78	01001110	123	01111011	168	10101000
34	00100010	79	01001111	124	01111100	169	10101001
35	00100011	80	01010000	125	01111101	170	10101010
36	00100100	81	01010001	126	01111110	171	10101011
37	00100101	82	01010010	127	01111111	172	10101100
38	00100110	83	01010011	128	10000000	173	10101101
39	00100111	84	01010100	129	10000001	174	10101110
40	00101000	85	01010101	130	10000010	175	10101111
41	00101001	86	01010110	131	10000011	176	10110000
42	00101010	87	01010111	132	10000100	177	10110001
43	00101011	88	01011000	133	10000101	178	10110010
44	00101100	89	01011001	134	10000110	179	10110011

DECIMAL EQUIVALENTS OF BINARY NUMBERS TO 2⁸

Decimal	Binary	Decimal	Binary	Decimal	Binary	Decimal	Binary
180	10110100	199	11000111	218	11011010	237	11101101
181	10110101	200	11001000	219	11011011	238	11101110
182	10110110	201	11001001	220	11011100	239	11101111
183	10110111	202	11001010	221	11011101	240	11110000
184	10111000	203	11001011	222	11011110	241	11110001
185	10111001	204	11001100	223	11011111	242	11110010
186	10111010	205	11001101	224	11100000	243	11110011
187	10111011	206	11001110	225	11100001	244	11110100
188	10111100	207	11001111	226	11100010	245	11110101
189	10111101	208	11010000	227	11100011	246	11110110
190	10111110	209	11010001	228	11100100	247	11110111
191	10111111	210	11010010	229	11100101	248	11111000
192	11000000	211	11010011	230	11100110	249	11111001
193	11000001	212	11010100	231	11100111	250	11111010
194	11000010	213	11010101	232	11101000	251	11111011
195	11000011	214	11010110	233	11101001	252	11111100
196	11000100	215	11010111	234	11101010	253	11111101
197	11000101	216	11011000	235	11101011	254	11111110
198	11000110	217	11011001	236	11101100	255	11111111

APPENDIX G THE POWERS OF 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.015625
128	7	0.0078125
256	8	0.00390625
512	9	0.001953125
1024	10	0.0009765625
2048	11	0.00048828125
4096	12	0.000244140625
8192	13	0.0001220703125
16384	14	0.00006103515625
32768	15	0.000030517578125
65536	16	0.0000152587890625
131072	17	0.00000762939453125
262144	18	0.000003814697265625
524288	19	0.0000019073486328125
1048576	20	0.00000095367431640625
2097152	21	0.000000476837158203125
4194304	22	0.0000002384185791015625
8388608	23	0.00000011920928955078125
16777216	24	0.000000059604644775390625
33554432	25	0.0000000298023223876953125
67108864	26	0.00000001490116119384765625
134217728	27	0.00000000745058059623828125
268435456	28	0.0000000037252902984619140625
536870912	29	0.00000000186264514923095703125
1073741824	30	0.000000000931322574615478515625
2147483648	31	0.000000000465661287307392578125
4294967296	32	0.00000000023283064365386962890625
8589934592	33	0.00000000011641532182693481453125
17179869184	34	0.0000000000582076609134874072265625
34359738368	35	0.00000000002910383045673370361328125
68719476736	36	0.00000000001455191522836685180640625
137438953472	37	0.00000000000727595761418325903203125
274877906944	38	0.00000000000363797880709171259566015625
549755813888	39	0.00000000000181898940354585647583078125
1099511627776	40	0.0000000000009094947017729282379150390625
2199023255552	41	0.00000000000045474735088646411895751953125
4398046511104	42	0.000000000000227373675443232059478759765625
8796093022208	43	0.000000000000113686837721616029739378828125
1759218604416	44	0.00000000000005684341886080801486968994140625
3518437208832	45	0.0000000000000284217094304040743484970703125
70368744177664	46	0.00000000000001421085471520077172424853515625
140737488355328	47	0.00000000000000710542735760100185871124237578125
281474976710656	48	0.000000000000003552713678800590929355621337890625
562949953421312	49	0.0000000000000017763568394002504646778106689453125
1125899906842624	50	0.00000000000000088817841970012523233890533447265625
2251799813685248	51	0.00000000000000044408920985062616169452667236328125
4503599627370496	52	0.000000000000000222044604925301308084726333181640625
9007199254740992	53	0.000000000000000111022302462515656404236316680908203125
18014398509841984	54	0.00000000000000005551115123125787270218158340454101625
36028797018963968	55	0.0000000000000000277555756156289135105907917022705078125
72057594037927936	56	0.000000000000000013877787807814567552953958113525390625
144115188075855872	57	0.000000000000000006938893903907228377647697925567626953125
288230376151711744	58	0.0000000000000000034694469519536141888238489627838137465625
576460752303423488	59	0.00000000000000000173472347597680709441192448139190673828125
1152921504606846976	60	0.000000000000000000867361737988403547205962240695953369140625
2305843009213693952	61	0.0000000000000000004336808689942017736029811203476684703125
4611686018427387904	62	0.000000000000000000216840434971008868014905601739883428515625
9223372036854757808	63	0.0000000000000000001084202172485540344007452608096941142578125
18446744073709551616	64	0.0000000000000000000542101208624275221700372640043497085712890625
3689348814741093232	65	0.0000000000000000000271050543121376108501863202174854278564453125
73786976294838206464	66	0.0000000000000000000135525271560688054250931601010742139282265625
147573952589676412928	67	0.000000000000000000006776263758034402712546580054371356964111328125
295147905179352825856	68	0.0000000000000000000033881317890172013562732900271856784820556640625
590295810358705651712	69	0.00000000000000000000169406589408600678136645011359283924102783203125
1180591620717411303424	70	0.0000000000000000000008470329425403390683215006796419620513916015625
236118341434822606848	71	0.000000000000000000000423516472715016953416125039820819810256950078125
472236648286945213696	72	0.00000000000000000000021175823618157508476708625169910490512847900390625

APPENDIX H

RECOMMENDED TEXTS

The following texts can be used to supplement the material presented in the COMPUTER LAB Workbook. The texts stress different aspects of computer technology and present information on a number of educational levels. By using these texts, or others, in conjunction with the COMPUTER LAB and workbook, the instructor can easily tailor the emphasis and complexity of his course.

- Arnold, B. H., **Logic and Boolean Algebra**, Englewood Cliffs: Prentice-Hall
- Bartee, T. C., **Digital Computer Fundamentals**, New York: McGraw-Hill
- Flores, I., **Computer Design**, Englewood Cliffs: Prentice-Hall
Computer Logic, Englewood Cliffs: Prentice-Hall
- Gillie, A. C., **Binary Arithmetic and Boolean Algebra**, New York: McGraw-Hill
- Harris, J. N., **Digital Transistor Circuits**, New York: John Wiley & Sons
- Hurley, R. B., **Transistor Logic Circuits**, New York: John Wiley & Sons
- Jacobowitz, H., **Computer Arithmetic**, New York: Hayden
- Kintner, P. M., **Electronic Digital Techniques**, New York: McGraw-Hill
- Lytel, A., **Computer Mathematics**, Indianapolis: Bobbs-Merrill
- Mandi, M., **Fundamentals of Electronic Computers**, Englewood Cliffs: Prentice-Hall
- Millman, J. and Taub, J., **Pulse, Digital and Switching Waveforms**, New York: McGraw-Hill
- Nashelsky, **Digital Computer Theory**, New York: John Wiley & Sons
- Transistor Fundamentals** Volume 1 thru 4 Indianapolis:
Howard W. Sams Publisher 1968
- Computer Basics** Volume 3 thru 6 Indianapolis: Howard W. Sams
Publisher 1968
- Barron R.C. and Piccirilli **Digital Logic and Computer Operations** McGraw Hill

APPENDIX I

COMPUTER LAB HARDWARE SPECIFICATIONS

I GENERAL SPECIFICATIONS

1. Size— $16\frac{1}{2}$ " x $12\frac{1}{2}$ " x $3\frac{1}{4}$ " (cases are stackable).
2. Finish—Simulated teak.
3. Power Requirements—H500: 50 or 60 cps, 105 to 120 VAC; H500A: 50 or 60 cps, 210 to 240 VAC.
4. Power Switch—On Variable Clock Control.

II ELECTRONIC LOGIC SPECIFICATIONS

1. Functions Available:
 - 8 J-K Flip-Flops (Master-Slave type)
 - 8 2-Input NAND Gates
 - 6 3-Input NAND Gates
 - 4 4-Input NAND Gates
 - 4 AND/NOR Gates
2. Type of Circuits:
Transistor-Transistor Logic Integrated Circuits
3. Logic Levels:
 - HI—+ 3 volts nominal (+ 2.4 to + 3.6 volts)
 - LO—Ground nominal (0.0 to + 0.4 volts)
 - Internal Supply Voltage: + 5 volts (not available on patch panel)
4. Input Loading:
 - Unit Load: 1.6 ma at GND. +40 μ a at + 3 volts.
 - Gate inputs and Flip-Flop J & K inputs present one unit load each.
 - Flip-Flop Clock and Reset inputs present two unit loads each.
 - Lamp Indicator inputs present five unit loads each.
5. Output Drive:
 - The Fan-Out of an output indicates the number of units loads that output can drive:
 - Logic outputs on the patch panel have a fan-out of 10.
 - Rocker Switches, Pulsers, the Clock, and HI terminals have a fan-out of 30.

III CONTROL AND INDICATOR SPECIFICATIONS

1. Functions Available:
 - 8 Rocker Switch logic level generators
 - 3 Pulsar Switches
 - 8 Lamp Indicators
 - 1 Variable Frequency Clock
2. Rocker Switches:
 - Give HI output when upper side of switch is depressed.
 - Give LO output when lower side of switch is depressed.
 - Fan out: 30.
3. Pulsar Switches:
 - Normally provide a LO output.
 - HI output when depressed, remain HI until released.
 - Fan-out: 30.
 - Built-in switch filters.

4. Lamp Indicators:
Normally OFF when no input.
ON when corresponding input HI.
OFF when corresponding input LO.
Load: 5 unit loads.
5. Clock:
Repetition rate: less than 1 pulse per second to more than 10×10^6 pulses per second.
Pulse width: 50 nanoseconds nominal.
Fan-out: 30.

APPENDIX J

COMPUTER LAB WARRANTY

The COMPUTER LAB is warranted against original defects in material and workmanship under normal use and service at the voltage marked thereon. This warranty shall remain in effect for a period of 90 days after receipt of COMPUTER LAB by the retail buyer. This agreement applies only within the United States and Puerto Rico.

All COMPUTER LABS must be returned prepaid to Digital Equipment Corporation. Transportation charges covering the return of the repaired COMPUTER LAB shall be paid by DEC. DEC will select the carrier, but by so doing, will not thereby assume any liability in connection with the shipment, nor shall the carrier be in any way construed to be the agent of DEC.

Please ship all units to:

Digital Equipment Corporation
COMPUTER LAB Repair Center
146 Main Street
Maynard, Massachusetts 01754

No COMPUTER LAB will be accepted for credit or exchange without the prior written approval of DEC, plus proper Return Authorization Number (DEC RA#).

The above warranty is contingent upon proper use in the application for which the COMPUTER LAB was intended and does not cover COMPUTER LABS which have been modified without DEC's prior written approval, or which have been subjected to unusual physical or electrical stress, or on which the original identification marks have been removed or altered. These warranties will not apply: if adjustment, repair, or parts replacement is required because of accident, neglect, misuse, failure of electric power, air conditioning, humidity control, transportation, or causes other than ordinary use.

Please read the instructions on the use and care of the COMPUTER LAB carefully because damage caused by failure to follow instructions is not covered by this warranty. No responsibility will be assumed for damage resulting from the use of other than DEC replacement parts.

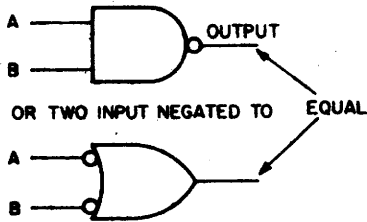
The COMPUTER LAB Warranty shall be in effect only if the retail buyer mails the COMPUTER LAB Warranty Information Card to DEC, Maynard, within ten days of receipt of COMPUTER LAB.

EXCEPT FOR THE EXPRESS WARRANTY STATED ABOVE, DEC DISCLAIMS ALL WARRANTIES INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS; and the above stated express warranty is in lieu of all obligations or liabilities on the part of DEC for damages, including but not limited to, consequential damages arising out of or in connection with the use or performance of the COMPUTER LAB.

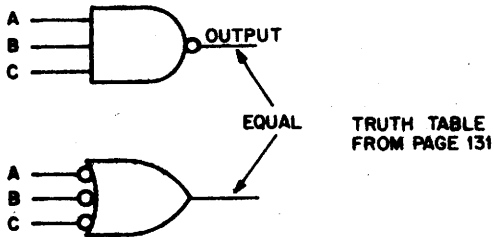
APPENDIX K

LOGIC ELEMENT TRUTH TABLES

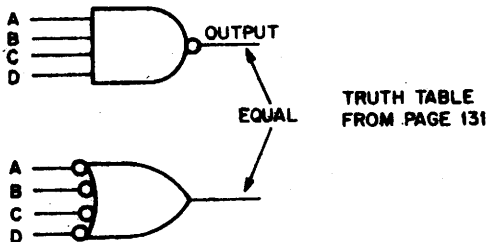
A Two Input NAND Gate



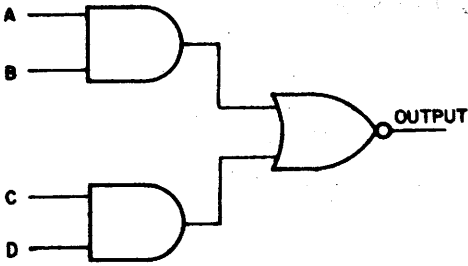
B Three input NAND Gate



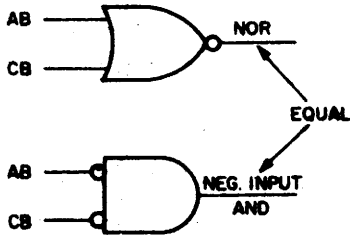
C Four input NAND Gate



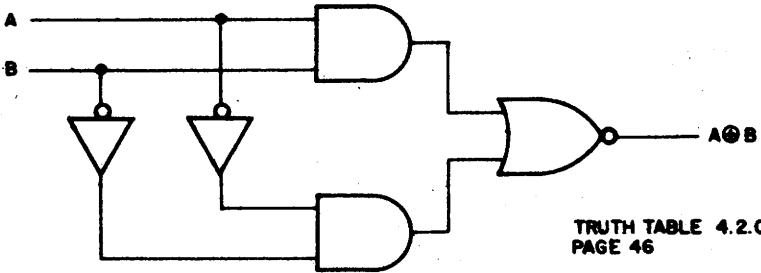
D AND/NOR Gate



TRUTH TABLE
FROM PAGE 132

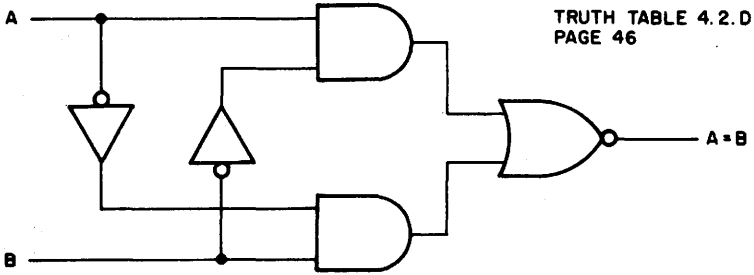


E Page 47 Figure 4.1 Exclusive OR

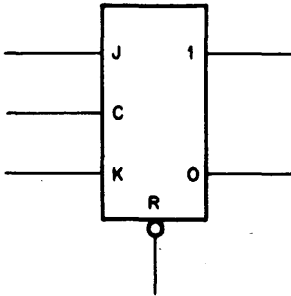


TRUTH TABLE 4.2.C
PAGE 46

F Page 47 Figure 4.1 Equivalence



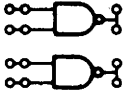
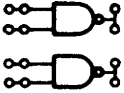
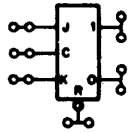
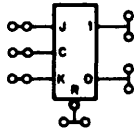
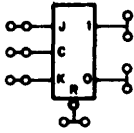
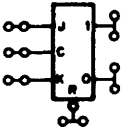
G J — K Flip Flop



TRUTH TABLE
FROM PAGE 133

DESCRIPTION	MODEL NUMBER	UNIT COST
COMPUTER LAB SETS		
Set contains COMPUTER LAB "916" Patchcord Assortment and B-350 COMPUTER LAB Workbook.	H-500	
	50-60 Hz	
	105-120 VAC	
	H-500-A	\$375.00
	50-60 Hz	
	210-240 VAC	
STANDARD PATCHCORD ASSORTMENTS		
Bundle of Taper-Pin Patchcords (107 of Assorted Lengths)	916	\$ 30.00
COMPUTER LAB TEACHERS' GUIDE		
8½" x 11" size; lies flat when opened.		\$ 5.00

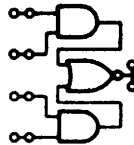
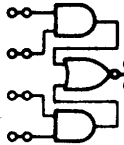
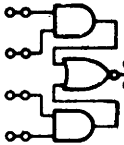
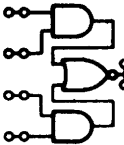
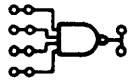
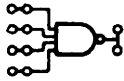
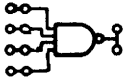
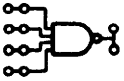
ALL PRICES SUBJECT TO CHANGE WITHOUT NOTICE



HIGH

HIGH

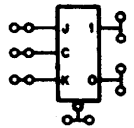
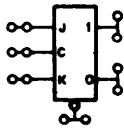
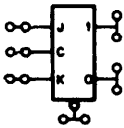
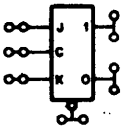
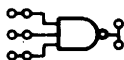
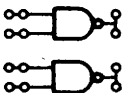
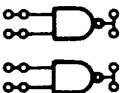
HIGH



HIGH

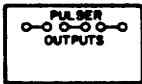
HIGH

HIGH



GND

GND



digital
COMPUTER LAB

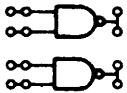
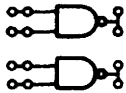
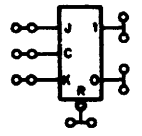
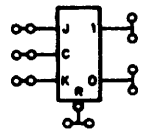
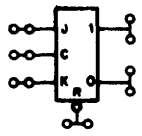
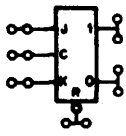


1 2 3
PULSE



HIGH

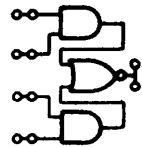
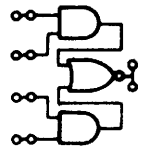
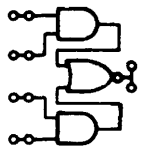
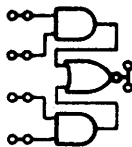
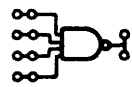
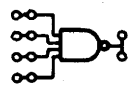
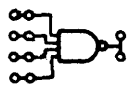
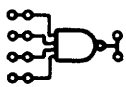
LOW



HIGH

HIGH

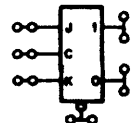
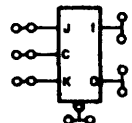
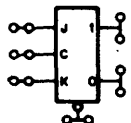
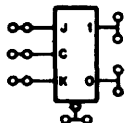
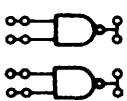
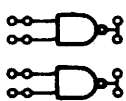
HIGH



HIGH

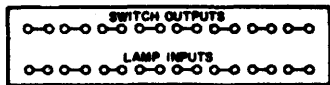
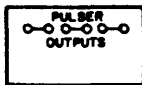
HIGH

HIGH

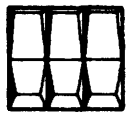


GND

GND

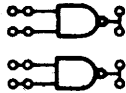
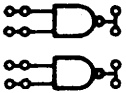
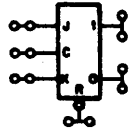
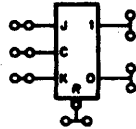
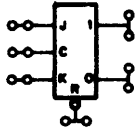
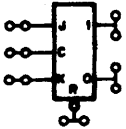


digital
COMPUTER LAB



1 2 3
PULSE

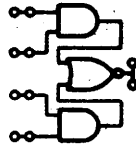
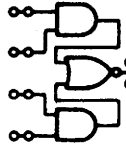
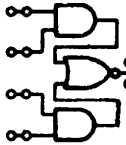
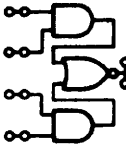
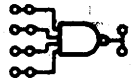
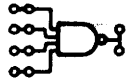
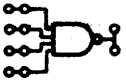




HIGH

HIGH

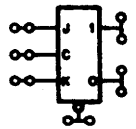
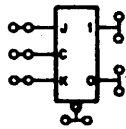
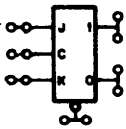
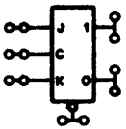
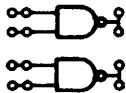
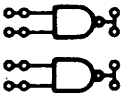
HIGH



HIGH

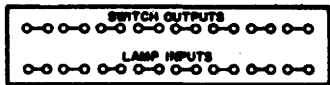
HIGH

HIGH



END

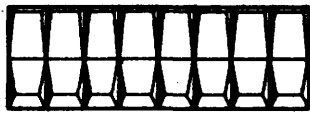
END



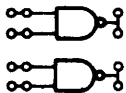
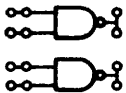
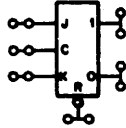
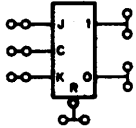
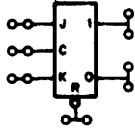
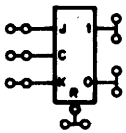
digital
COMPUTER LAB



1 2 3
PULSE



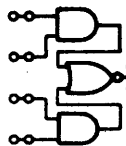
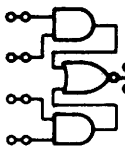
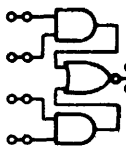
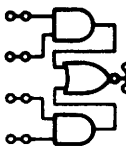
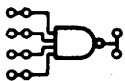
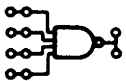
HIGH
LOW



HIGH

HIGH

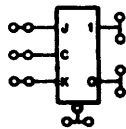
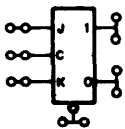
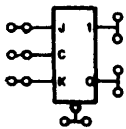
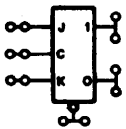
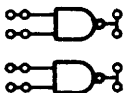
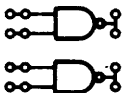
HIGH



HIGH

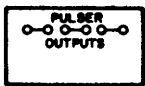
HIGH

HIGH

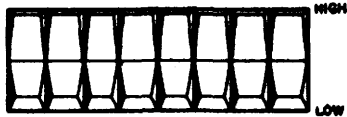
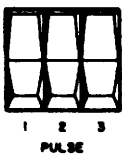


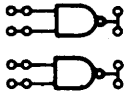
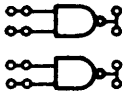
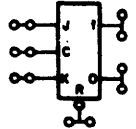
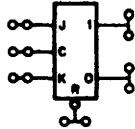
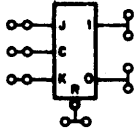
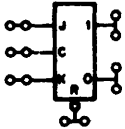
GND

GND



digital
COMPUTER LAB

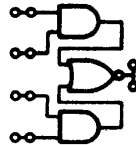
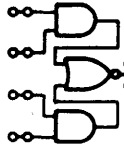
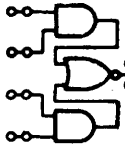
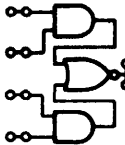
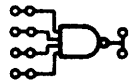
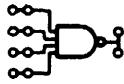
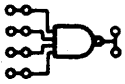




HIGH

HIGH

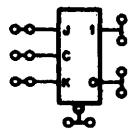
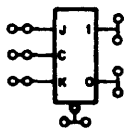
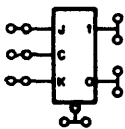
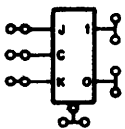
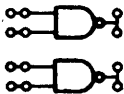
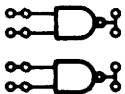
HIGH



HIGH

HIGH

HIGH



END

END



digital
COMPUTER LAB



1 2 3
PULSE



HIGH
LOW

DIGITAL EQUIPMENT CORPORATION **digital** WORLD-WIDE SALES AND SERVICE

MAIN OFFICE AND PLANT

146 Main Street, Maynard, Massachusetts, U.S.A. 01754 • Telephone: From Metropolitan Boston: 646-800 • Elsewhere: (617)-897-5111
 TWX: 710-347-0212 Cable: DIGITAL MAYN Telx: 84-8467

UNITED STATES

NORTHEAST

REGIONAL OFFICE

15 Lunda Street, Waltham, Massachusetts 02154
 Telephone: (617)-891-1020/1033 TWX: 710-324-8919

WALTHAM

15 Lunda Street, Waltham, Massachusetts 02154
 Telephone: (617)-891-8310/8315 TWX: 710-324-8919

CAMBRIDGE/BOSTON

899 Main Street, Cambridge, Massachusetts 02139
 Telephone: (617)-461-6130 TWX: 710-320-1157

ROCHESTER

130 Aliens Creek Road, Rochester, New York 14618
 Telephone: (716)-461-1700 TWX: 710-596-3211

CONNECTICUT

1 Prestige Drive, Meriden, Conn. 06450
 Telephone: (203)-237-8441/7488 TWX: 710-461-0054

MID-ATLANTIC — SOUTHEAST

REGIONAL OFFICE

U.S. Route 1, Princeton, New Jersey 08540
 Telephone: (609)-452-2940 TWX: 510-685-2338

NEW YORK

95 Cedar Lane, Englewood, New Jersey 07631
 Telephone: (201)-871-4194, (212)-584-8955, (212)-136-9447
 TWX: 710-961-9721

NEW JERSEY

1289 Route 46, Parsippany, New Jersey 07654
 Telephone: (609)-335-3330 TWX: 710-887-8219

PRINCETON

U.S. Route 1
 Princeton, New Jersey 08540
 Telephone: (609)-452-2940 TWX: 510-685-2338

LONG ISLAND

1919 Middle Country Road
 Centereach, L.I., New York 11720
 Telephone: (516)-585-5410/5413 TWX: 510-228-6505

PHILADELPHIA

Station Square Three, Paoli, Pennsylvania 19001
 Telephone: (215)-847-4607/4410 Telex: 510-886-8395

WASHINGTON

Executive Building
 6811 Kenilworth Ave., Riverdale, Maryland 20840
 Telephone: (301)-778-1800/752-8797 TWX: 710-826-9662

DURHAM/CHAPEL HILL

2724 Chapel Hill Boulevard
 Durham, North Carolina 27707
 Telephone: (919)-486-3347 TWX: 510-927-0912

CANADA

Digital Equipment of Canada, Ltd.
 CANADIAN HEADQUARTERS
 150 Rossmont Street, Carleton Place, Ontario
 Telephone: (813)-257-2815 TWX: 610-361-1851

OTTAWA

120 Holland Street, Ottawa 3, Ontario
 Telephone: (613)-725-2193 TWX: 610-562-8807

TORONTO

230 Lakeshore Road East, Port Credit, Ontario
 Telephone: (416)-276-8111 TWX: 610-492-4306

MONTREAL

9675 Cote de Liesse Road
 Dorval, Quebec, Canada T9S
 Telephone: 514-636-9383 TWX: 610-422-4124

EDMONTON

5631 - 103 Street
 Edmonton, Alberta, Canada
 Telephone: (403)-434-9333 TWX: 610-831-2248

VANCOUVER

Digital Equipment of Canada, Ltd.
 2210 West 12th Avenue
 Vancouver 8, British Columbia, Canada
 Telephone: (604)-736-5816 TWX: 610-929-2008

SOUTH AMERICA

CASIN S.A.
 Virrey del Pinar 4071 Buenos Aires, Argentina
 Telephone: 52-5185 Telex: 012-2284

EUROPEAN HEADQUARTERS

Digital Equipment Corporation International-Europe
 81 Route De L'Aire
 1227 Carouge F, Geneva, Switzerland
 Telephone: 42 78 50/58/59 Telex: 22 883

GERMANY

Digital Equipment GmbH
 COLOGNE
 5 Koeln, Bismarckstrasse 7, West Germany
 Telephone: 52 21 81 Telex: 989-2289
 Telegram: Flip Chip Koeln

FRANKFURT

A M Forthhaus 5-7
 6076 Neu-Isenburg-Gravenbruch, Germany
 Telephone: 08102-5526/5529

MID-ATLANTIC — SOUTHEAST (cont.)

ORLANDO

Suite 222, 6900 Lake Ellenor Drive, Orlando, Fla. 32809
 Telephone: (305)-851-4450 TWX: 810-850-0180

ATLANTA

2815 Cleverly Place, Suite 100,
 Atlanta, Georgia 30340
 Telephone: (404)-456-3133/3134/3135 TWX: 810-757-4223

KNOXVILLE

5731 Lyons View Pike, S.W., Knoxville, Tenn. 37919
 Telephone: (615)-588-6571 TWX: 810-583-0123

CENTRAL

REGIONAL OFFICE

1850 Frontage Road, Northbrook, Illinois 60062
 Telephone: (312)-498-2500 TWX: 910-686-0555

PITTSBURGH

402 Penn Center Boulevard
 Pittsburgh, Pennsylvania 15235
 Telephone: (412)-243-8500 TWX: 710-797-3657

CHICAGO

1850 Frontage Road, Northbrook, Illinois 60062
 Telephone: (312)-498-2500 TWX: 910-686-0555

ANN ARBOR

230 Huron View Boulevard, Ann Arbor, Michigan 48103
 Telephone: (313)-781-1150 TWX: 910-223-8953

INDIANAPOLIS

21 Beachway Drive — Suite G
 Indianapolis, Indiana 46224
 Telephone: (317)-243-8341 TWX: 610-341-3438

MINNEAPOLIS

Suite 111, 8000 Cedar Avenue South,
 Minneapolis, Minnesota 55400
 Telephone: (612)-884-4092 TWX: 910-756-2818

CLEVELAND

Park Hill Bldg., 35104 Euclid Ave.
 Wuyfighty, Ohio 44324
 Telephone: (216)-946-9484 TWX: 810-427-2608

ST. LOUIS

Suite 110, 1115 Progress Pky., Maryland Heights,
 Missouri 63043
 Telephone: (314)-872-7520 TWX: 910-764-3031

DAYTON

3101 Kettering Blvd., Dayton, Ohio 45439
 Telephone: (513)-299-7377 TWX: 910-459-1678

MILWAUKEE

Suite 207, 2825 N. Mayfair Rd., Milwaukee, Wis. 53222
 Telephone: (414)-453-3400 TWX: 910-282-1199

INTERNATIONAL

GERMANY (cont.)

MUNICH
 8 Muenchen 13, Wallnerstrasse 2, Germany
 Telephone: 0811-358011-15 Telex: 811-35-80-11

HANOVER

Digital Equipment Corporation GmbH
 3 Hannover, Germany, Podbielskistrasse 102
 Telephone: 5511-58-70-35 Telex: 922-862

ENGLAND

Digital Equipment Co., Ltd.
 READING
 8 Tessa Road, Reading, Berkshire, England
 Telephone: 0734-583833/4/5/6 Telex: 84327

MANCHESTER

5 Upper Precinct, Worley
 Manchester, England M25AZ
 Telephone: 061-790-8411 Telex: 668886

LONDON

Bilton House, Uxbridge Road, Ealing, London W.5
 Telephone: 01-579-2781 Telex: 22371

FRANCE

Equipelement Digital
 PARIS
 327 Rue de Charenton, 75 Paris 12^e Fr., France
 Telephone: 344-76-07 Telex: 21330

BENELUX

THE HAGUE
 Sir Winston Churchill N.V.
 Dir Winston Churchiln 370
 Pijpewijk (Z.H.), Netherlands
 Telephone: 70-9951-80 Telex: 32533

BRUSSELS

Digital Equipment N.V.S.A.
 108 Rue D'Arton
 1040 Brussels, Belgium Telephone: 021-39258

SWEDEN

Digital Equipment Aktiebolag
 STOCKHOLM
 Vretensgatan 2, S-171 54 Solna, Sweden
 Telephone: 96 13 90 Telex: 170 50
 Cable: Digital Stockholm

CENTRAL (cont.)

DALLAS

8655 North Stemmons Freeway
 Dallas, Texas 75247
 Telephone: (214)-838-6880 TWX: 910-861-4000

HOUSTON

3417 Millam Street, Suite A, Houston, Texas 77002
 Telephone: (713)-524-2561 TWX: 910-861-1651

WEST

REGIONAL OFFICE
 500 San Antonio Road, Palo Alto, California 94308
 Telephone: (415)-326-5940 TWX: 910-373-1206

ANAHEIM

801 E. Ball Road, Anaheim, California 92805
 Telephone: (714)-776-8022/8730 TWX: 910-581-1158

WEST LOS ANGELES

1510 Colmer Avenue, Los Angeles, California 90025
 Telephone: (213)-479-3791/4318 TWX: 910-342-8889

SAN FRANCISCO

560 San Antonio Road, Palo Alto, California 94308
 Telephone: (415)-326-5940 TWX: 910-373-1206

OAKLAND

1565 Edgewater Drive
 Oakland, California 94621
 Telephone: (415)-835-5453/7830 TWX: 910-366-7238

ALBUQUERQUE

6303 Indian School Road, N.E.
 Albuquerque, N.M. 87110
 Telephone: (505)-298-5411/5428 TWX: 910-988-0814

DENVER

2205 South Colorado Blvd., Suite #5
 Denver, Colorado 80222
 Telephone: (303)-757-3332/758-1658/758-1659
 TWX: 910-531-2650

SEATTLE

1521 130th N.E., Bellevue, Washington 98005
 Telephone: (206)-454-4058/455-5404 TWX: 910-443-2308

SALT LAKE CITY

431 South 3rd East, Salt Lake City, Utah 84111
 Telephone: (801)-328-9838 TWX: 910-925-5834

PHOENIX

307 E. Southern Ave., Tempe, Arizona 85281
 Telephone: (602)-867-1818 TWX: 910-950-4881

PORTLAND

13015 Southwest Pacific Highway, Tigard, Ore. 97223
 Telephone: (503)-638-8632/8884 TWX: 910-458-8792

SWITZERLAND

Digital Equipment Corporation S.A.
 GENEVA
 81 Route De L'Aire
 1227 Carouge F, Geneva, Switzerland
 Telephone: 42 79 50/58/59 Telex: 22 883

ZURICH

Freitagstrasse 26, 8002 Zurich, Switzerland
 Telephone: (51) 38 78 23

ITALY

Digital Equipment S.p.A.
 MILAN
 Corso Garibaldi, 46, 20121 Milano, Italy
 Telephone: 872 748, 872 804, 872 304 Telex: 33815

AUSTRALIA

Digital Equipment Australia Pty. Ltd.
 SYDNEY
 75 Alexander St., Crows Nest, N.S.W. 2085, Australia
 Telephone: 439-2566 Telex: AA20740
 Cable: Digital, Sydney

MELBOURNE

60 Park Street, South Melbourne, Victoria, 3205
 Telephone: 66-6142 Telex: AA30700

WESTERN AUSTRALIA

643 Murray Street
 West Perth, Western Australia 6005
 Telephone: 21-4683 Telex: AA82140

BRISBANE

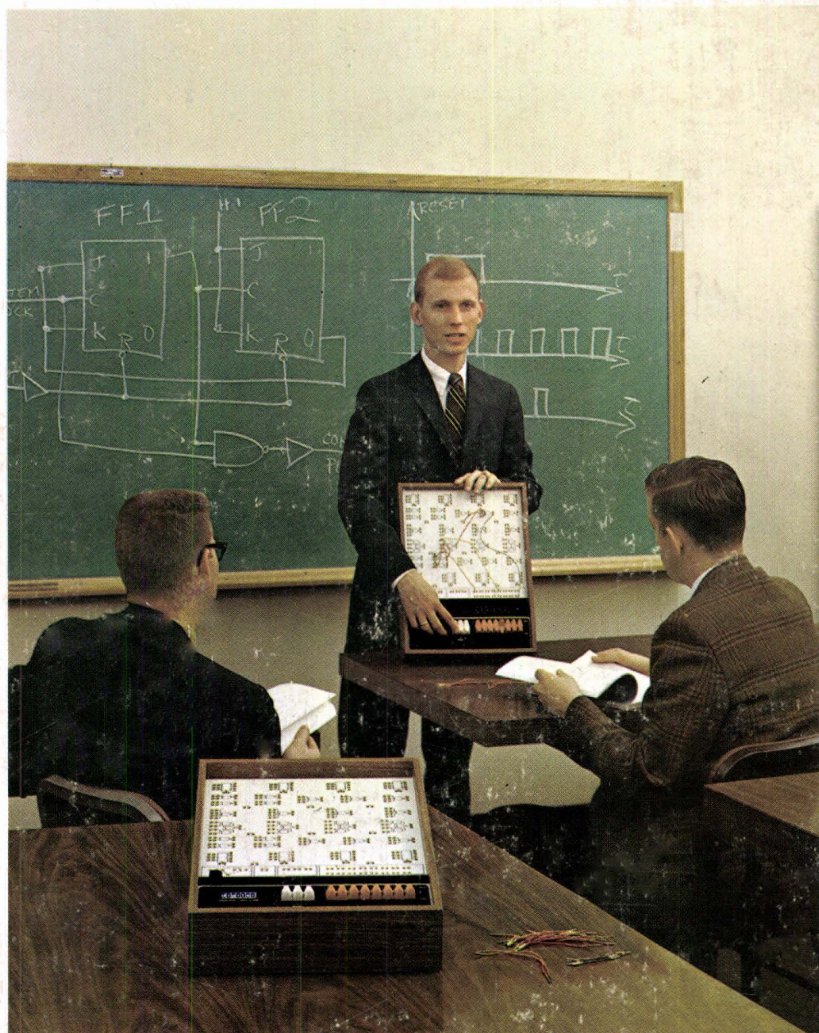
138 Merivale Street, South Brisbane
 Queensland, Australia 4101
 Telephone: 44047 Telex: AA40616

JAPAN

TOKYO
 Ritai Trading Co., Ltd. (sales only)
 Koza-to-Kaiken Bldg.
 No. 18-14, Nishishinbashi 1-chome
 Minato-Ku, Tokyo, Japan
 Telephone: 5915246 Telex: 781-4208

Digital Equipment Corporation International

Kowa Building No. 17, Second Floor
 2-7 Nishi-Azabu 1-Chome
 Minato-Ku, Tokyo, Japan
 Telephone: 404-5894/6 Telex: TX-6428



The DIGITAL COMPUTER LAB is a complete classroom laboratory for teaching computer fundamentals: digital logic, binary arithmetic and Boolean algebra. The COMPUTER LAB Workbook provides step-by-step instructions in digital logic principles. Each step is followed by wiring a logic design on the COMPUTER LAB front panel and then testing it. The course is composed of ten chapters divided into 46 individual experiments, illustrating the full range of digital logic and computer fundamentals.