February 1974

To the Graduate Council:

I am submitting herewith a thesis written by Loreene Knoph Cantwell entitled "Computer Minimization of Multiple-Output Boolean Functions." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Robert E. Bodenheimer, Major Professor

We have read this thesis
and recommend its acceptance:

Lora R. Jordan

Robert M. Aiken

Accepted for the Council:

Vice Chancellor
Graduate Studies and Research

COMPUTER MINIMIZATION OF MULTIPLE-OUTPUT BOOLEAN FUNCTIONS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee

Loreene Knoph Cantwell

March 1974

ACKNOWLEDGMENTS

The author takes this opportunity to express her sincere gratitude to Dr. Robert E. Bodenheimer for his suggestion of the topic and for his advice and encouragement during the preparation of this thesis. Also, the author wishes to express her appreciation to Dr. Robert M. Aiken and Dr. Sara R. Jordan for their services on the master's degree committee. Special thanks go to Dr. George L. Jensen, Director of Analytical and Data Services, and to Mr. Frederick L. Stiles, Director of the Computer Assisted Registration System, for their suggestions and thoughts during the course of this thesis; to Mr. George Miles, thesis consultant; and to the computer operators of The University of Tennessee Computing Center, Mr. Bruce Delaney, Mr. Ted Sherrod, and Mr. Jim Calfee, for their cooperation and help during the programming phases of this research. Finally, the author is deeply indebted to her parents, Mr. and Mrs. Nic L. Knoph, and to her husband, Frank P. Cantwell, Jr., for their encouragement and assistance which has made this computer science education both possible and worthwhile.

1136607

ABSTRACT

This thesis describes two basic methods for minimizing two-level, multiple-input, multiple-output Boolean functions. The two methods are: (1) the map method; and (2) the tabular method. Two general procedures for each method are presented, along with illustrative examples of each. In addition, other more recently introduced extensions of each method, each with an example, are briefly described.

The formation, properties, and uses of maps and of the prime implicant table are discussed. A cost function is used in each case to determine the minimality of the solution.

Finally, an IBM System 360 MODEL 65 PL/1 computer program is presented. The basis of this program is the Quine-McCluskey method. The program allows up to ten inputs and ten outputs and determines a minimal solution according to user-specified cost factors. If the problem is too large to be handled completely within the main memory of the computer, a sub-optimal solution, which may be minimal, is found. The value of this program to the user, either a student or a circuit designer, is its ability to rapidly determine a minimal solution for a given problem and to free the user from the tedious and repetitive steps involved in determining that minimal solution.

TABLE OF CONTENTS

CHAPTER		PAGE
I.	INTRODUCTION	1
	Statement of Problem	2
	Scope of Investigation	3
II.	MAP METHODS	8
	Veitch Charts	11
	Karnaugh Maps	13
	Extension of the Karnaugh Map	19
III.	TABULAR METHODS	23
	Quine's Method	23
	Quine-McCluskey Method	26
	Quine-McCluskey Method for Multiple-Output Functions	32
IV.	THE PRIME IMPLICANT TABLE	38
	Construction	38
	Reduction of the Table	41
	Cyclic Prime Implicant Table	48
V.	MINIMIZATION WITH THE AID OF A DIGITAL COMPUTER	62
VI.	SUMMARY AND CONCLUSIONS	76
LIST OF REFERENCES		78
APPENDICES		80
A.	GENERAL PROGRAM FLOWCHART	81
B.	PROGRAM SOURCE LISTING	90

CHAPTER PAGE

C. SAMPLE INPUTS AND OUTPUTS 159

D. CONTROL CARD SEQUENCE USING LOAD MODULE 166

VITA 167

LIST OF TABLES

TABLE		PAGE
I.	Table of Combinations for Example 1	9
II.	Rearrangement of Table of Combinations for Example 2 for Formation of the Quine-McCluskey Procedure	28
III.	Results of Combining Terms for Example 2	29
IV.	Quine-McCluskey Method Applied to Example 2	31
V.	Quine-McCluskey Method Applied to Example 3	35
VI.	Prime Implicants for Example 3	36
VII.	Prime Implicant Table for Example 2	40
VIII.	Prime Implicant Table for Example 3	42
IX.	Reduced Prime Implicant Table for Example 2	47
X.	Prime Implicant Table for Example 2 after Partial Completion of Bowman-McVey Method	55
XI.	Reduced Prime Implicant Table for Example 3	58

LIST OF FIGURES

FIGURE		PAGE
1.	Logic Gate Symbols	4
2.	A Typical Two-Level AND-OR Gating Network for a Single-Output Function	5
3.	Veitch Charts	12
4.	Karnaugh Maps	14
5.	Karnaugh Map for Example 1	16
6.	Karnaugh Map for Example 2	18
7.	The Extended Karnaugh Map for Example 2	21
8.	Circuit Diagram for Example 3	61
9.	Control Card Sequence for Source Deck	73
10.	Control Card Sequence for Object Deck	74

CHAPTER I

INTRODUCTION

Boolean algebra, originally introduced in the mid-1800's by English mathematician George Boole, was developed as the symbolic interpretation of logical reasoning to be used in the solution of everyday problems. Since dealing with problems involving four or five variables involved a considerable amount of computation that necessitated the use of skillful methods, interest in solving such large-scale problems was lost (1).

However, with the coming of digital computers, there was a rebirth of Boolean algebra. In 1938, Shannon made the first use of this theory in circuit design and simplification. Boolean algebra, previously known as the algebra of propositional logic, became known as the algebra of digital computer circuits and is now the foundation for the analysis of switching circuits, both combinational and sequential.

The methods of symbolic logic that are concerned with deduction (or simplification, in terms of circuit minimization) must lead from true sentences to true sentences. A proposition is a statement that is said to be either true or false. There are three key propositional operators that aid in the manipulation of the propositions: AND, OR, and NOT. The first two operators combine two propositions; the latter is associated with a single proposition. The AND combination is true if and only if both propositions are true. The OR combination is true if and only if either or both propositions are true. The negation of a proposition is true only if the proposition itself is false.

Propositions that are combined by the propositional operators can be determined as either true or false and are called Boolean functions.

I. STATEMENT OF PROBLEM

Switching circuits perform logical functions. The design of such circuits is called logical design. These circuits can be as simple as a light switch in a closet or as complex as the circuit that performs arithmetic operations in an electronic digital computer. The purpose of switching theory when applied to digital computer circuitry is to determine the voltage level on a wire. Each proposition then has the basic form, "Wire X has a high voltage signal."

The task of the designer is to properly connect the logic circuit elements or the logic blocks, AND, OR, and NOT, so that the resulting circuit will perform the desired function. There are many different ways to connect these logic blocks so that the cricuit will perform this function. The designer's main problem is to realize that circuit in some efficient, reliable, and economical form.

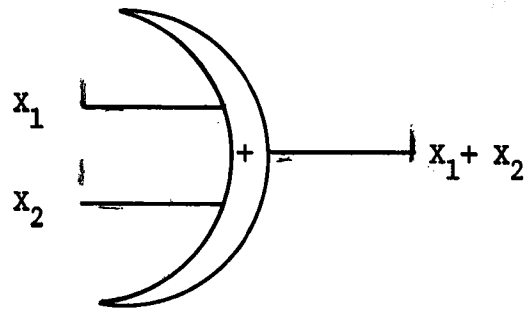
Although intuition can sometimes be used in designing switching circuits, a circuit does not need to become too complicated before intuitive thinking can fail to yield the best network. As networks become larger and circuits more complex, the designer must rely on something more than intuition. Formalized procedures have been developed in recent years to aid the designer in obtaining the desired circuitry. These methods lead to simpler circuits which are more easily understood and implemented. Having fewer circuit components increases the reliability of the circuit operation and therefore decreases the amount of down-time, as well as the cost of constructing the network.

II. SCOPE OF INVESTIGATION

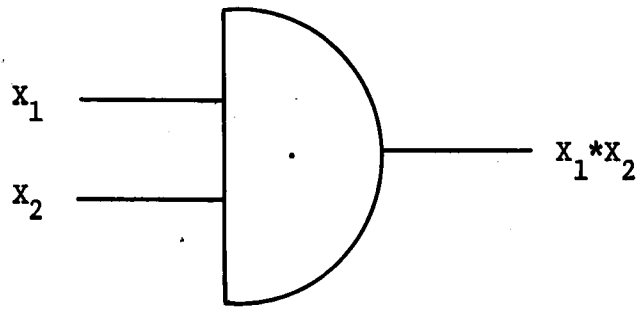
Obtaining algorithms for the design of efficient combinational circuits is a major area in switching theory research. The more successful algorithms deal with the design of circuits that allow only two levels of gating. These networks are either a series of AND-gates connected to an output OR-gate or a series of OR-gates connected to an output AND-gate. The result of such algorithms is a Boolean algebra expression for each output. If the output gate is an OR-gate, this expression is in the form of a sum of several product terms and is called a minimal sum. If the output gate is an AND-gate, the expression is called a minimal product since it is in the form of a product of sum terms. The techniques for obtaining such minimal forms are obtained from the input combinations for which the function has either a value of "zero" (called "0-terms") or a value of "one" (called "1-terms") and the input combinations for which the value of the function is unspecified (called the "don't-care terms" or "Ø-terms") (2).

Implementation of combinational switching circuits is not limited to electronic gates. It is possible to use relay contact networks, but this thesis will discuss circuit implementation with gates. There are a variety of symbols for the gates that are used today in the design of combinational circuits. Figure 1 shows the symbols for the AND, OR, and NOT gate that will be used in this research.

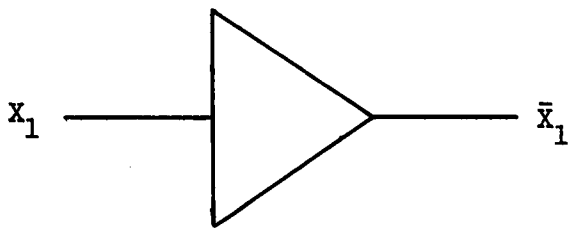
The author has chosen to limit the subject matter of this thesis to single-output and multiple-output minimization techniques for circuits with two levels of gating. Figure 2 shows a diagram of a typical



(a) OR gate



(b) AND gate



(c) NOT gate or inverter

Figure 1. Logic gate symbols.

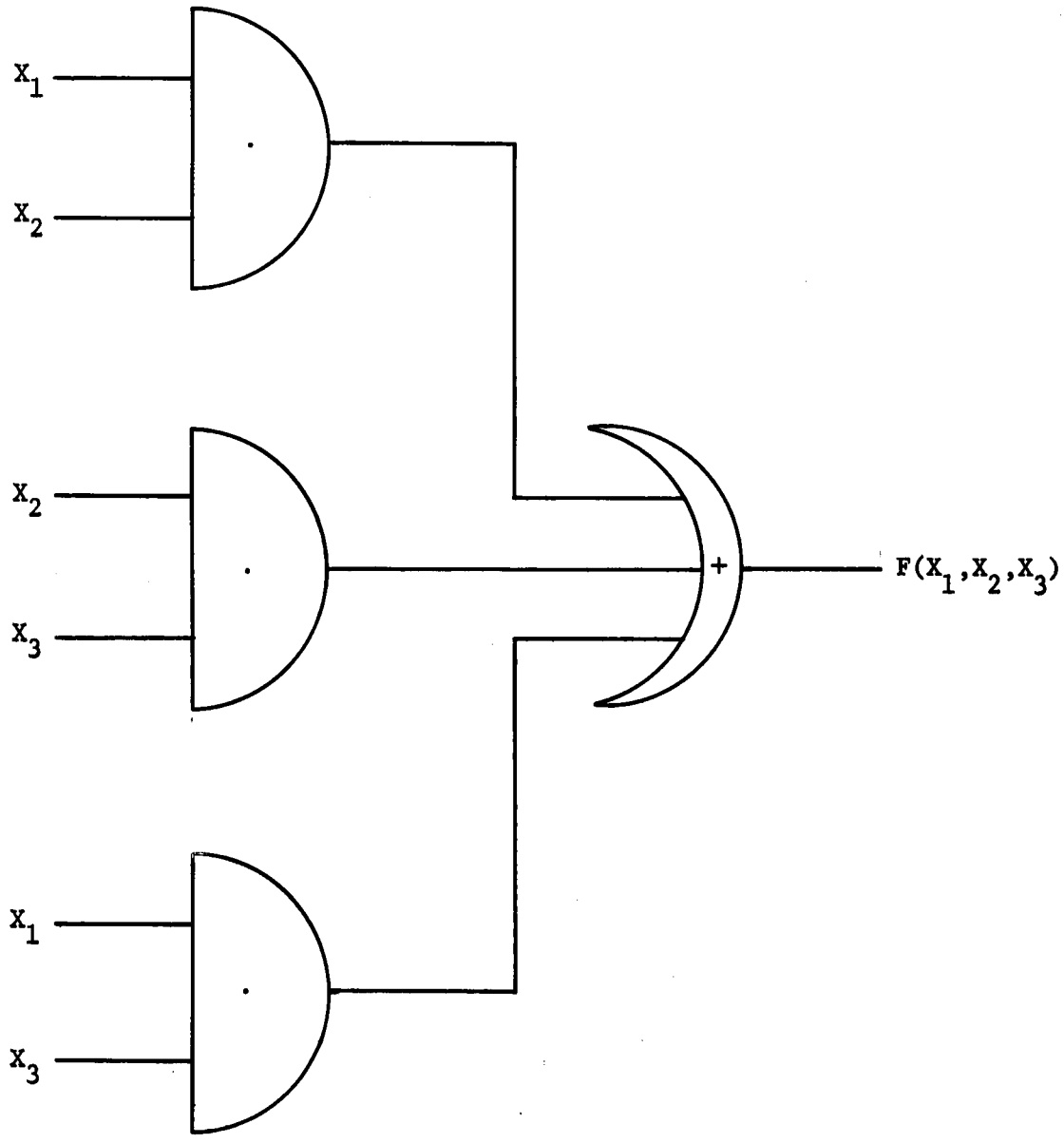


Figure 2. A typical two-level AND-OR gating network for a single-output function.

two-level AND-OR gating network for a single-output function. The analysis is limited to the two-level combinational switching circuit, defined as one for which the outputs are dependent only on the present circuit inputs, as opposed to the sequential switching circuit, for which the outputs also depend on previous input combinations.

The results of the techniques described in this thesis can be applied only to the steady-state behavior of combinational circuits. The author has chosen to ignore static hazards, since their presence in a combinational circuit might cause only a momentarily incorrect output before the output becomes constant. Such hazards may be avoided in the circuit design by introducing certain redundancies within the network. However, if a combinational network that contains a hazard is used in the design of a sequential circuit, that circuit can enter an incorrect internal state, resulting in a false output that remains as long as the circuit remains in that incorrect internal state. Therefore, in order to design a properly functioning sequential circuit, having hazard-free networks in the combinational parts of the circuit is a necessity.

A problem may be minimized in either the sum-of-products form or the product-of-sums form. Although the discussion in this thesis will use the sum-of-products form, all the minimization methods are applicable to the product-of-sums form by the principle of duality.

Two general minimization techniques are presented. Chapter II deals with map methods. The Veitch chart, one of the earliest attempts at establishing a formalized procedure in analyzing switching networks, is presented and is followed by the Karnaugh map method, an attempt

to improve the Veitch chart method. A relatively new procedure for extending the Karnaugh map is also presented. Chapter III deals with tabular methods. Quine's method was the first attempt at a step-by-step procedure. McCluskey added a few more steps, resulting in the Quine-McCluskey method. Since a true minimal form can be assured only by using the prime implicant table, Chapter IV covers only the formation and use of this table and its adaptation to multiple-output functions.

In determining the minimal realization of a function, the designer must consider some cost criterion. There are several different cost definitions presently used in minimizing prime implicant tables. The cost factor to be used in this thesis is a function of the number of logic gates, the number of inputs, and the number of interconnections between the two levels of gating. Reducing the number of gates usually causes a decrease in the number of interconnections. Both reductions mean fewer possible failures, an accompanying increase in reliability, and easier detection of faults in a network that is operating incorrectly (3).

A digital computer program, written by the author, is presented in Chapter V. This program accepts Boolean functions as input and determines a minimal solution. The computations are based on the Quine-McCluskey method. Input format and limitations are explained in that chapter.

In recent years, use of the electronic digital computer has increased tremendously (4,5,6). As shown in Chapter V, the computer can rapidly determine minimal solutions that formerly required large amounts of human time and effort.

CHAPTER II

MAP METHODS

Design of a combinational circuit can be brought about by using the theorems of Boolean algebra in a straight-forward manner. The desired circuit performance is specified by generating a table of combinations. An algebraic expression is formed from this table, and a simplified circuit realization is derived from that function. The table of combinations, which completely specifies a combinational circuit, is generated by specifying all possible combinations of values of the input variables and determining the appropriate circuit output for those input combinations. The formation of the input combinations is called a standard basis; a one-to-one correspondence exists between the assignment and the binary representation of the decimal numbers. A table of combinations for a combinational circuit is shown in Table I.

A concise means of specifying the table of combinations is to list the numbers for which the output has a value of "one," if the result is to be in the sum-of-products form, or a value of "zero," if the result is to be in the product-of-sums form. For the table of combinations shown in Table I, the function can be defined as follows:

Example 1.

$$F(X_1, X_2, X_3, X_4) = \Sigma(0, 1, 2, 3, 4, 5, 10, 11, 13, 15)$$

or

$$F(X_1, X_2, X_3, X_4) = \Pi(6, 7, 8, 9, 12, 14)$$

TABLE I

TABLE OF COMBINATIONS FOR EXAMPLE 1

Decimal Value	Input Variables				Output
	X_1	X_2	X_3	X_4	F
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

After the table of combinations has been formed, an algebraic expression for the output function can be written directly from this table. For each row of the table that specifies an output of "one," there is an associated fundamental product term, often called a "p-term," a "1-term," or a minterm. Each such term contains all the input variables. The rule for forming the fundamental product term for a certain row is to prime any input variable which equals "zero" in that row and to leave unprimed any input variable that equals "one" in that row. The product of these literals so formed is the corresponding fundamental product term for that row of the table of combinations. The sum of all the fundamental product terms for the function is called a canonical sum. The canonical sum for the problem shown in Example 1 is listed below.

$$F(X_1, X_2, X_3, X_4) = \bar{X}_1\bar{X}_2\bar{X}_3\bar{X}_4 + \bar{X}_1\bar{X}_2\bar{X}_3X_4 + \bar{X}_1\bar{X}_2X_3\bar{X}_4 + \bar{X}_1\bar{X}_2X_3X_4 + \bar{X}_1X_2\bar{X}_3\bar{X}_4 + \bar{X}_1X_2\bar{X}_3X_4 + X_1\bar{X}_2X_3\bar{X}_4 + X_1\bar{X}_2X_3X_4 + X_1X_2\bar{X}_3X_4 + X_1X_2X_3X_4$$

For a function of "n" input variables, there are 2^n rows in the table of combinations. Since the output in each row can have either a value of "zero" or a value of "one," there are 2^{2^n} possible output functions for these "n" variables. If "n" gets to be larger than four, a large number of switching functions must be considered. Therefore, the need for the development of a systematic way to generate all the different output functions or to generate only the minimal solution, according to what the circuit designer thinks should be minimal, became necessary.

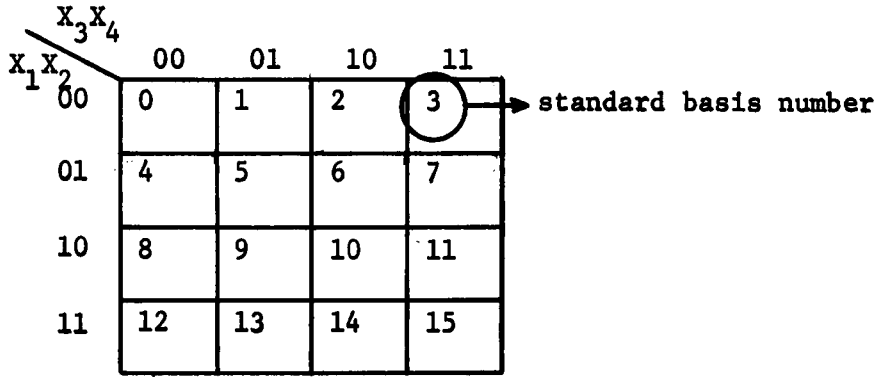
I. VEITCH CHARTS

The Veitch chart is actually a table of combinations that is arranged in a more convenient form. A map for a single-output function of "n" variables consists of 2^n squares or cells, each of which represents one possible combination of the input variables. The column and row headings are labeled with combinations of the input variables. Each square is assigned the decimal number corresponding to the binary value of the input variables or the input state it represents. The function value, either "zero," "one," or "phi," associated with each input combination is entered in the corresponding square. Veitch labeled the map in order that the standard basis numbers could be read horizontally across the map. Figure 3(a) shows a Veitch chart for a four variable problem.

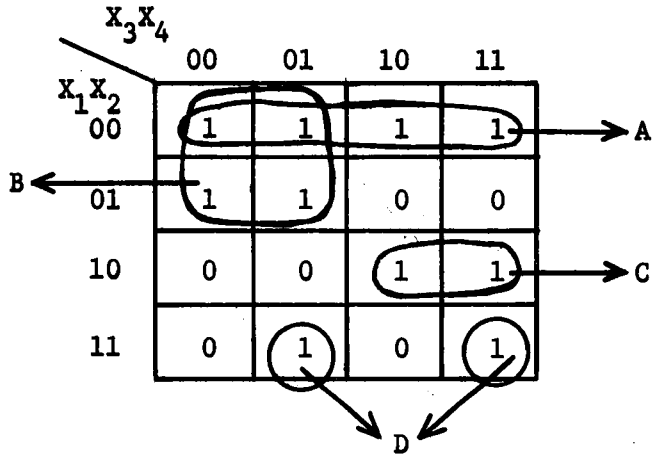
Figure 3(b) shows a Veitch chart for Example 1. Circle A corresponds to all possible combinations of X_3 and X_4 and hence will be covered by $\bar{X}_1 \cdot \bar{X}_2$. Circle B corresponds to all possible combinations of X_2 and X_4 and will be covered by $\bar{X}_1 \cdot \bar{X}_3$. The two units in circle C can be covered by $X_1 \cdot \bar{X}_2 \cdot X_3$; and the two units in circle D by $X_1 \cdot X_2 \cdot X_4$. Hence, a sum-of-products solution is of the following form.

$$F(X_1, X_2, X_3, X_4) = (\bar{X}_1 \cdot \bar{X}_2) + (\bar{X}_1 \cdot \bar{X}_3) + (X_1 \cdot \bar{X}_2 \cdot X_3) + (X_1 \cdot X_2 \cdot X_4)$$

The Veitch chart is impractical because of the difficulties in determining the cover terms for a complex function. Even maps for only four variables are confusing. Such a graphical technique could not be relied upon to produce a minimal result, especially for a large function.



(a) Veitch chart for four variables.



(b) Veitch chart for Example 1.

Figure 3. Veitch charts.

II. KARNAUGH MAPS

In 1953, one year after the introduction of the Veitch chart, Karnaugh improved the single-output map method by rearranging the column and row labels for the input variables so that the designer could see more easily which combinations of variables covered other combinations, thereby eliminating some of the variables. Figure 4 shows Karnaugh maps for three, four, and five variable problems, with each standard basis number in the corresponding cell.

The power of this method is its use of the human ability to recognize a pattern in a graphical representation more rapidly than in a tabular representation. With the cyclic code which Karnaugh used, successive characters differ in only a single position. The use of this code to label the columns and rows of the map means that cells which share a side correspond to input variable combinations that differ by the value of a single variable. Such cells are said to be adjacent. This adjacency plays a key part in the simplification process because two adjacent cells may combine to eliminate a variable by the following Boolean theorem:

> $Aa + Aa' = A$ where $\underline{A}$ denotes a product of literals and $\underline{a}$ represents a single literal.

Adjacencies always exist between squares which are physically side by side. For example, in Figure 4(b) cell number 15 is adjacent to cells 7, 11, 13, and 14. Cells on the edge of the map are also adjacent to the cell that would be adjacent if the two opposite sides of the map were joined together. In Figure 4(a) cell 2 is adjacent to cells 0, 3, and 6; cell 4 is adjacent to cells 5, 6, and 0. Note that

C \ AB	00	01	11	10
0	0	2	6	4
1	1	3	7	5

(a) Map for three variables.

CD \ AB	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

(b) Map for four variables.

A=0

DE \ BC	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

A=1

DE \ BC	00	01	11	10
00	16	20	28	24
01	17	21	29	25
11	19	23	31	27
10	18	22	30	26

(c) Map for five variables.

Figure 4. Karnaugh maps.

in the five variable map, adjacencies exist between the two maps. In Figure 4(c), cell 15 is adjacent to cells 7, 11, 13, 14, and 31.

Groups containing a number of adjacent cells that is some power of two may be combined. This collection of 2^m cells, each adjacent to "m" cells of the group, is called a subcube. This subcube is said to cover those "m" cells of the map and can be expressed by a product term that contains "n-m" literals, where "n" is the number of input variables for the function. In other words, "m" variables have been eliminated. In the Karnaugh map for Example 1, shown in Figure 5, cells 0, 1, 2, and 3 combine to form subcube A. Since this subcube has four members and since the problem has four input variables, the subcube is expressed by product term $\bar{X}_1 \cdot \bar{X}_2$, which contains two literals ($n = 4$ and $m = 2$).

In covering a map, every cell that contains a "one" entry must be accounted for or covered at least once and can be used in as many groups as desired. A subcube should be as large as possible, and the largest ones correspond to prime implicants. A prime implicant is redundant in the cover of the function if its omission from the cover does not change the function. A prime implicant that is the only implicant that covers a particular cell becomes an essential prime implicant. This essential prime implicant covers at least one cell that is not covered by any other prime implicant of the function.

In Figure 5, subcubes A and B clearly combine four terms each. Subcube A corresponds to prime implicant $\bar{X}_1 \cdot \bar{X}_2$ since variables X_3 and X_4 change values and can be eliminated. Likewise subcube B corresponds to prime implicant $\bar{X}_1 \cdot \bar{X}_3$ with variables X_2 and X_4 having been eliminated.

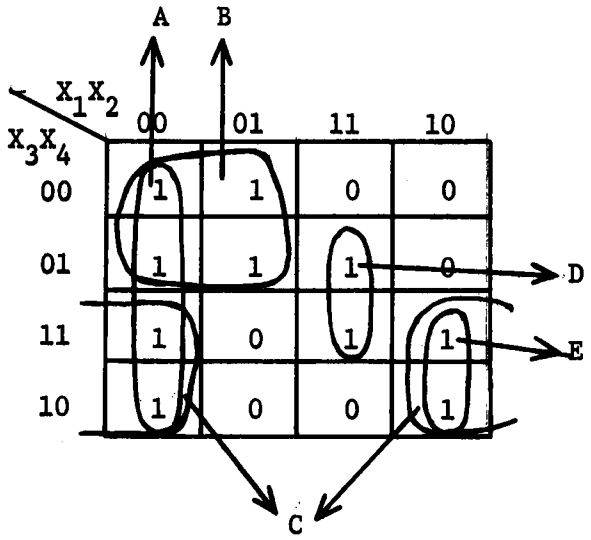


Figure 5. Karnaugh map for Example 1.

In subcube C, cells 2 and 3 are adjacent to cells 10 and 11 and hence may be combined into one cube with four members, corresponding to prime implicant $\bar{X}_2 \cdot X_3$. Subcube D corresponds to prime implicant $X_1 \cdot X_2 \cdot X_4$. Prime implicant $X_1 \cdot \bar{X}_2 \cdot X_3$, covering minterms 10 and 11, is redundant since circle E is included in the larger circle C. Prime implicants B and C are essential since they cover minterms that no other prime implicants cover (terms 4 and 10, respectively).

In minimizing a function by a Karnaugh map, the combination of as many adjacent cells with "one" entries (sometimes called "on-cells" and representing the minterms of the function) and helpful "don't-care" entries as possible is the basic task. Then each prime implicant will cover the maximum number of adjacent cells, a number which must be a power of two. When all the cells with "one" entries are covered by the smallest number of subcubes, the resulting expression will be a minimum sum-of-products, containing "X" product terms, where "X" is the number of subcubes.

Consider the following five-variable problem.

Example 2.

$$F(X_1,X_2,X_3,X_4,X_5) = \Sigma(1, 4, 7, 14, 17, 20, 21, 22, 23) + \Phi(0, 3, 6, 19, 30)$$

The Karnaugh map for this function, shown in Figure 6, is represented by two four-variable maps side by side and contains a total of 32 cells, where 32 is "two" raised to the power of the number of input variables, which is "five." In this case, there are adjacencies between corresponding cells of the two maps as well as between cells within a map. Of the five prime implicants that are designated as circles A, B, C, D, and

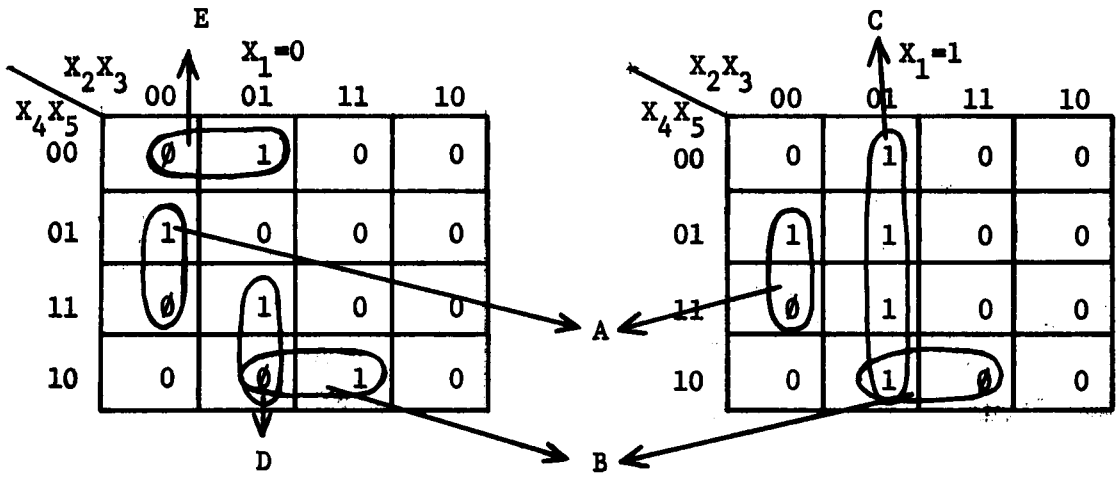


Figure 6. Karnaugh map for Example 2.

E, only circle B corresponds to an essential prime implicant since no other prime implicant covers input combination 14. These five prime implicants cover the function, but this cover is not necessarily a minimal cover. Many other prime implicants can be found from this map, but the designer must redraw the map several times in order to be sure that he has discovered all the prime implicants.

Applying minimization techniques to a map of more than six variables is extremely difficult because of the inability of the eye to detect the patterns in such a complex map structure. For these large structures, inclusion of redundant prime implicants and failure to detect maximum subcubes is an unavoidable result.

III. EXTENSION OF THE KARNAUGH MAP

In 1969, Alan B. Marcovitz and Charles M. Shub suggested the use of unique identifiers on the Karnaugh map in order to help the designer in recognizing redundant covers (7). This added feature is most useful if the function has many terms that are not covered by essential prime implicants. When the Karnaugh map is used in the simplification of rather complex functions, a redundant product term may not be easily detected. This redundancy can be eliminated by the assignment of unique identifiers to each prime implicant in the cover and by placing that identifier in each cell of the map that is covered by that prime implicant.

This algorithm consists of four basic steps. After a cover is determined, each of the prime implicants is assigned a unique identifier or letter. The map is then redrawn with the letters assigned to the

appropriate cells, and this new map is checked to see if any redundant prime implicants are apparent. Consider again Example 2. One cover for the function consists of the following set of prime implicants, each of which has been assigned a unique letter.

$$\bar{X}_1\bar{X}_2\bar{X}_4\bar{X}_5 = U$$

$$\bar{X}_2\bar{X}_3X_5 = V$$

$$\bar{X}_1\bar{X}_2X_4X_5 = W$$

$$X_3X_4\bar{X}_5 = X$$

$$X_1\bar{X}_2X_5 = Y$$

$$X_1\bar{X}_2X_3 = Z$$

Redrawing the map with the appropriate letters in the ON-cells results in the Karnaugh map shown in Figure 7.

A redundant prime implicant covers minterms that are already covered by other prime implicants. Therefore, the identifier corresponding to that redundant prime implicant will not appear in any cell by itself. If all redundant prime implicants are removed from the cover of the function, each identifier will appear alone in at least one cell. In the map in Figure 7, the product term represented by identifier Y is redundant since the four cells that contain a Y also contain some other identifier. This algorithm will aid in the minimization of a function, but is not guaranteed to find the minimal solution.

Map methods, in general, aid in the simplification of Boolean functions. However, use of these trial and error techniques does not guarantee the optimal solution. For more complex problems with more

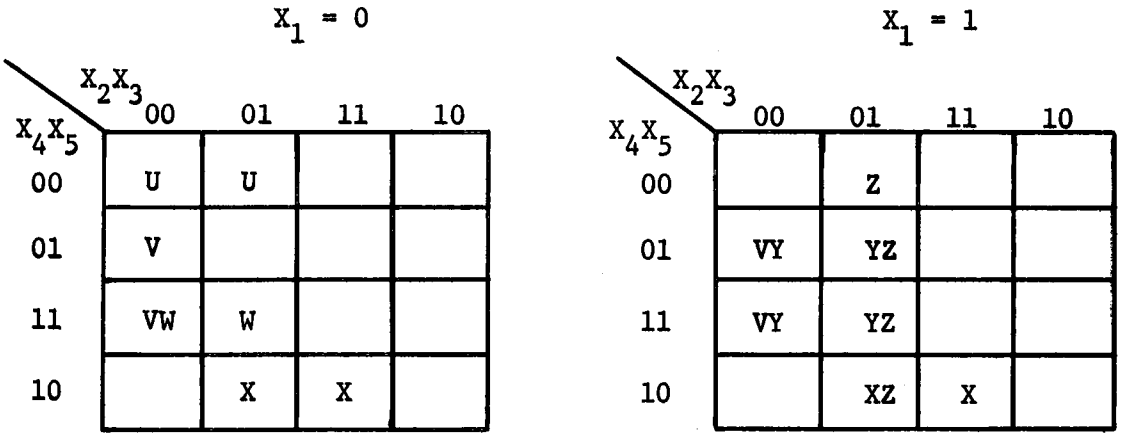


Figure 7. The extended Karnaugh map for Example 2.

than six variables, this type of method is extremely impractical because of the reliance on the designer's ability to discern the correct patterns within the map. For larger problems, a more formalized method to obtain minimal solutions is desirable. Tabular methods provide such a step-by-step means of generating minimal solutions to Boolean functions.

CHAPTER III

TABULAR METHODS

A tabular method that is based mainly on the work of W. U. Quine and E. J. McCluskey corrects the problems associated with the map methods. Quine developed the initial approach, and McCluskey later perfected the method. The resulting Quine-McCluskey procedure guarantees a two-level circuit realization that contains the fewest number of product terms, which correspond to gates, and the fewest number of literals, corresponding to gate inputs. Quine's procedure involved an exhaustive search of all possible combinations of minterms. McCluskey reduced this lengthy search and formalized a method to choose the minimal combination of the prime implicants required to realize the function.

I. QUINE'S METHOD

In 1952 Quine introduced a method for finding prime implicants to serve as an aid in the simplification of Boolean functions (8). By the application of the Boolean theorem $Aa + Aa' = A$ to the minterms generated from the table of combinations, variables from some of the terms of the canonical expansion can be eliminated and the circuit thus simplified. By repeated use of this theorem, several variables may be eliminated from each term of the expansion. The theorem is applied to every pair of terms; to every pair of terms generated from the original group of terms; and so on until the theorem can be applied no longer. Then the remaining terms are the prime implicants and

can be used in forming a minimal sum-of-products expression for the function.

Consider the table of combinations in Table I, page 9, for the function given in Example 1. From that table, the following canonical expansion can be formed.

$$F(X_1, X_2, X_3, X_4) = \bar{X}_1\bar{X}_2\bar{X}_3\bar{X}_4 + \bar{X}_1\bar{X}_2\bar{X}_3X_4 + \bar{X}_1\bar{X}_2X_3\bar{X}_4 + \bar{X}_1\bar{X}_2X_3X_4 + \bar{X}_1X_2\bar{X}_3\bar{X}_4 + \bar{X}_1X_2\bar{X}_3X_4 + X_1\bar{X}_2X_3\bar{X}_4 + X_1\bar{X}_2X_3X_4 + X_1X_2\bar{X}_3X_4 + X_1X_2X_3X_4$$

The following step-by-step procedure illustrates the method by which a sum which is more minimal than the canonical expansion is obtained.

Step 1. By the application of the Boolean theorem to the minterms of the table of combinations, pairs of terms that differ in only one variable can be combined.

Combining Terms	Input Combinations	Resulting Terms
$\bar{X}_1\bar{X}_2\bar{X}_3\bar{X}_4$ and $\bar{X}_1\bar{X}_2\bar{X}_3X_4$	0000 and 0001	$\bar{X}_1\bar{X}_2\bar{X}_3$
$\bar{X}_1\bar{X}_2X_3\bar{X}_4$ and $\bar{X}_1\bar{X}_2X_3X_4$	0010 and 0011	$\bar{X}_1\bar{X}_2X_3$
$\bar{X}_1X_2\bar{X}_3\bar{X}_4$ and $\bar{X}_1X_2\bar{X}_3X_4$	0100 and 0101	$\bar{X}_1X_2\bar{X}_3$
$X_1\bar{X}_2X_3\bar{X}_4$ and $X_1\bar{X}_2X_3X_4$	1010 and 1011	$X_1\bar{X}_2X_3$
$X_1X_2\bar{X}_3X_4$ and $X_1X_2X_3X_4$	1101 and 1111	$X_1X_2X_4$

Step 2. Since each input combination combined with another input combination, only the newly formed terms need be considered now. Determine if any two of these new terms can combine to eliminate another variable.

Combining Terms	Input Combinations	Resulting Terms
$\bar{X}_1\bar{X}_2\bar{X}_3$ and $\bar{X}_1\bar{X}_2X_3$	000- and 001-	$\bar{X}_1\bar{X}_2$

Step 3. No more terms will combine, and each input combination is covered by at least one of these terms. Thus, a solution to the function is the following transmission equation, formed by summing the terms that were not combined.

$$F(X_1, X_2, X_3, X_4) = \bar{X}_1X_2\bar{X}_3 + X_1\bar{X}_2X_3 + X_1X_2X_4 + \bar{X}_1\bar{X}_2$$

However, there are several different ways in which the input terms can be combined. Consider combining the new terms generated in Step 1 in a different manner.

Step 2'.

Combining Terms	Input Combinations	Resulting Terms
$\bar{X}_1\bar{X}_2\bar{X}_3$ and $\bar{X}_1X_2\bar{X}_3$	000- and 010-	$\bar{X}_1\bar{X}_3$
$\bar{X}_1\bar{X}_2X_3$ and $X_1\bar{X}_2X_3$	001- and 101-	$\bar{X}_2X_3$

Then the transmission equation becomes $F(X_1, X_2, X_3, X_4) = \bar{X}_1\bar{X}_3 + \bar{X}_2X_3 + X_1X_2X_4$. This new equation, which has one less product term and four fewer literals, represents a more minimal solution than the previous solution.

As the previous example illustrates, Quine's method can become very long and involved for complex problems that have either many input variables or many minterms. Its weakness lies in the lack of a short, step-by-step procedure for applying the theorem to all possible pairs of rows of the table of combinations.

II. QUINE-McCLUSKEY METHOD

In 1956 McCluskey proposed a modification for increasing the efficiency of finding the prime implicants of Boolean functions. This method, which became known as the Quine-McCluskey method, is actually a step-by-step algorithm for determining all possible prime implicants of a function and for selecting some minimal group of these prime implicants that realizes the function.

In this tabular method, the first step is to form the table of combinations and to rearrange the rows of the table that produce an output of "one" into a more suitable tabular form. Instead of forming all possible combinations of pairs of rows of this table in order to generate the prime implicants, the task may be simplified by a rearrangement of the rows into groups. If two terms are to combine, they must be identical in all positions except one. Thus, one term must have one additional "one" entry in its binary representation than the other term has. If the table is rearranged into groups of terms which have the same numbers of "ones" in the binary representation, then only terms within adjacent groups of the table need to be compared. Each term within one group must be compared with each term of the next adjacent group. For each pair of binary representations differing in only one coordinate, a new term which has the same coordinates as both the original terms in each position except one is formed. That one position in which the terms disagree contains a "dash." The formation of this new term is analogous to combining two cells in a Karnaugh map to eliminate a variable. The two terms that combined to form the new

term are checked off in the first table, since the fact that they combined to form another term signifies that they are not prime implicants. Since all prime implicants must be found, all pairs of terms must be compared, including those terms that were previously checked off.

After all comparisons for the first table have been completed, the terms in the new table must be compared in the same manner. In this second table, a new term is formed whenever two terms differ in only one position and have dashes in the same positions. This procedure is continued until no more pairs of terms can be combined. Any term in the tables which is not checked off is a prime implicant.

Consider the two-step procedure for generating all the prime implicants for the problem shown in Example 2.

Step 1. From the table of combinations, arrange the terms that have an output of "one" and the "don't-care" terms in order by the number of ones contained in the binary representation of the minterms. Horizontal lines are placed between the groupings to aid in identification. Table II shows the result of this operation.

Step 2. Compare each term of a group with each term of the group below it to determine if there are any pairs of terms that can be combined. If such a pair is found, check off those terms in the table. Repeat this step until no more terms can be combined. Table III shows the result of the continual application of this step.

Note that any term that has more than one dash can be formed in more than one way. Terms 00-11 and 10-11 combine to form term -0-11.

TABLE II

REARRANGEMENT OF TABLE OF COMBINATIONS FOR EXAMPLE 2
FOR FORMATION OF THE QUINE-McCLUSKEY PROCEDURE

Number of Ones	Binary Representation of Minterms
0	00000
1	00001
1	00100
2	00011
2	00110
2	10001
2	10100
3	00111
3	01110
3	10011
3	10101
3	10110
4	10111
4	11110

TABLE III

RESULTS OF COMBINING TERMS FOR EXAMPLE 2

Column 1		Column 2		Column 3	
Number of Ones	Binary Representation	Number of Ones	Binary Representation	Number of Ones	Binary Representation
0	00000 ✓	0	0000-	1	-00-1
1	00001 ✓	0	00-00	1	-01-0
1	00100 ✓	1	000-1 ✓	2	-0-11
2	00011 ✓	1	-0001 ✓	2	-011-
2	00110 ✓	1	001-0 ✓	2	--110
2	10001 ✓	1	-0100 ✓	2	10--1
2	10100 ✓	2	00-11 ✓	2	101--
3	00111 ✓	2	-0011 ✓		
3	01110 ✓	2	0011- ✓		
3	10011 ✓	2	0-110 ✓		
3	10101 ✓	2	-0110 ✓		
3	10110 ✓	2	100-1 ✓		
4	10111 ✓	2	10-01 ✓		
4	11110 ✓	2	1010- ✓		
		2	101-0 ✓		
		3	-0111 ✓		
		3	-1110 ✓		
		3	10-11 ✓		
		3	101-1 ✓		
		3	1011- ✓		
		3	1-110 ✓		

In addition, terms -0011 and -0111 also combine to form the same term. Term -0-11 is not written twice in the third column of Table III, but all four of the combining terms are checked off in second column of that table.

For processing by hand, working with the binary representation of the minterms has its drawbacks. Errors will occur when trying to compare zeroes and ones for a large number of terms. With a little experience the Quine-McCluskey method can be performed much more quickly and efficiently with the decimal integer representation of the minterms.

Use of this representation means that terms that combine must differ by a power of two. The term with the fewer number of ones in its binary representation is subtracted from the term with the additional one in its representation. This difference, which must be positive, must be recorded in the table, since implicants with different variables deleted cannot be combined.

For example, 5(2) is a shorthand way to denote implicant 001-1, which was formed by combining terms 5 and 7. The "5" is the minterm with the fewer number of ones, and the "2" indicates the position of the dash, the variable that has been eliminated. During subsequent iterations, only those terms that have the same value in parentheses need to be compared. The terms 3(4) and 19(4) combine to form the term 3(4,16). The "3" indicates the smaller minterm, and the "4" is the original difference, or the position for which the variable has been eliminated. The "16" is the new difference between 19 and 3. The generation of the prime implicants for Example 2 is shown again in Table IV. The terms that are not checked off are the prime implicants.

TABLE IV

QUINE-McCLUSKEY METHOD APPLIED TO EXAMPLE 2

Column 1	Column 2	Column 3
0 ✓	0(1)	1(2,16)
1 ✓	0(4)	4(2,16)
4 ✓	1(2) ✓	3(4,16)
3 ✓	1(16) ✓	6(1,16)
6 ✓	4(2) ✓	6(8,16)
17 ✓	4(16) ✓	17(2,4)
20 ✓	3(4) ✓	20(1,2)
7 ✓	3(16) ✓	
14 ✓	6(1) ✓	
19 ✓	6(8) ✓	
21 ✓	6(16) ✓	
22 ✓	17(2) ✓	
23 ✓	17(4) ✓	
30 ✓	20(1) ✓	
	20(2) ✓	
	7(16) ✓	
	14(16) ✓	
	19(4) ✓	
	21(2) ✓	
	22(1) ✓	
	22(8) ✓	

III. QUINE-McCLUSKEY METHOD FOR MULTIPLE-OUTPUT FUNCTIONS

For the multiple-output function, the generation of the prime implicants is similar to the method used for the single-output function, but with the addition of a tag for each minterm. The minterms for all the functions are merged into one table. Each of these terms is identified by a tag, usually a string of letters a, b, c, . . . , which indicates to which output function or functions each minterm belongs.

The procedure for generating the first table is the same as before. The terms and their respective tags are ordered according to the number of ones in the binary representation, and comparisons are made to determine the pairs of terms that differ in only one position. Once such a pair of terms is found, their tags must be compared. If the tags for the two minterms have no common letter, the two terms cannot be combined. If the two minterms can be combined and their respective tags have at least one common letter, then the new term is formed just as in the single-output case, and the new tag consists of the letter or letters shared by the two combining tags. If the tag of either or both of the combining terms is identical to the tag for the new term, then that combining term is checked off, indicating that the minterm is not a prime implicant. For example, the term 2(1) with tag "c" and the term 10(1) with tag "abc" can be combined. The new term is 2(1,8) and the new tag is "c." Term 2(1) can be checked off since its tag matches the new tag, but term 10(1) cannot be checked off since 10(1) may still be used as a prime implicant for functions a and b.

The multiple-output prime implicants are generated by continuing the process of comparing pairs of terms, using the single-output method for forming new terms and the above method for forming the corresponding tags. Note that if two terms have tags with no letters in common, then those two terms cannot combine. The terms that remain unchecked when no more pairs of terms may be combined are the prime implicants for the multiple-output function.

To illustrate this procedure, consider the step-by-step generation of prime implicants for the following three-output function.

Example 3.

$F_a(X_1, X_2, X_3, X_4) = \Sigma(2, 4, 10, 11, 12, 13) + \Phi(5)$

$F_b(X_1, X_2, X_3, X_4) = \Sigma(4, 5, 10, 11, 13)$

$F_c(X_1, X_2, X_3, X_4) = \Sigma(1, 2, 3, 10, 11, 12) + (13, 15)$

Step 1. Merge all the terms together and form the tag for each term.

Term	Tag
2	ac
4	ab
10	abc
11	abc
12	ac
13	abc
5	ab
1	c
3	c
15	c

Step 2. Arrange the terms in order by the number of ones in the binary representation. Draw lines between the groups to aid in the generation of prime implicants in Step 3.

Term	Tag
1	c
2	ac
4	ab
3	c
5	ab
10	abc
12	ac
11	abc
13	abc
15	c

Step 3. Determine pairs that can be combined. Form the new term and its tag from each appropriate pair. Repeat this step until no more pairs of terms can be combined. Table V shows the application of this step to Example 3, and Table VI shows the resulting prime implicants.

The Quine-McCluskey method generates all the prime implicants for a given function. A maximum number of literals has been deleted from each prime implicant. At this point, the task of the designer is to express the function as a sum of a number of these prime implicants such that the "cost" of implementing the circuit is minimal. Finding this minimal set that covers every input combination is not easy. A

TABLE V

QUINE-McCLUSKEY METHOD APPLIED TO EXAMPLE 3

Column 1			Column 2			Column 3	
1	c	✓	1(2)	c		2(1,8)	c
2	ac	✓	2(1)	c	✓	4(1,8)	a
4	ab	✓	2(8)	ac			
3	c	✓	4(1)	ab			
5	ab	✓	4(8)	a	✓		
10	abc	✓	3(8)	c	✓		
12	ac	✓	5(8)	ab			
11	abc	✓	10(1)	abc			
13	abc		12(1)	ac			
15	c	✓	11(4)	c			
			13(2)	c			

TABLE VI

PRIME IMPLICANTS FOR EXAMPLE 3

Terms	Binary Representation	Prime Implicant	Covered Terms	Functions Occurring In
13	1101	$X_1X_2\bar{X}_3X_4$	13	a,b,c
1(2)	00X1	$\bar{X}_1\bar{X}_2X_4$	1,3	c
2(8)	X010	$\bar{X}_2X_3\bar{X}_4$	2,10	a,c
4(1)	010X	$\bar{X}_1X_2\bar{X}_3$	4,5	a,b
5(8)	X101	$X_2\bar{X}_3X_4$	5,13	a,b
10(1)	101X	$X_1\bar{X}_2X_3$	10,11	a,b,c
12(1)	110X	$X_1X_2\bar{X}_3$	12,13	a,c
11(4)	1X11	$X_1X_3X_4$	11,15	c
13(2)	11X1	$X_1X_2X_4$	13,15	c
2(1,8)	X01X	$\bar{X}_2X_3$	2,3,10,11	c
4(1,8)	X10X	$X_2\bar{X}_3$	4,5,12,13	a

convenient method to aid the designer in reaching the minimal solution is the formation of the prime implicant table, which will be discussed in the next chapter.

CHAPTER IV

THE PRIME IMPLICANT TABLE

After all the prime implicants have been generated by one of the methods previously presented, the designer must choose some minimal group of these that will cover the function. This minimization can be more easily accomplished if the minterms and the prime implicants are arranged in some sort of table. The resulting table contains all the information required for determining the minimal cover of the function and is called a prime implicant table or a cover table.

I. CONSTRUCTION

In the prime implicant table, all the minterms of the given function appear in a row at the top of the table. The prime implicants appear in a column at the left of the table. If a prime implicant covers a certain minterm in the top row, an "X" is placed at the intersection of that row and that column in the prime implicant table. Thus, each column can contain any number of "X's," but every column must have at least one "X." The designer's task is to choose the smallest subset of prime implicants such that each column of the table contains at least one "X" in those rows contained in this subset and such that the total cost of the selected prime implicants is as small as possible. If these requirements are met, the union of the prime implicants contained in that subset covers the functions, and no other expression which costs less and is equivalent to the functions can be found.

The "don't-care" terms, those for which the designer desires no specific circuit response, should not be listed as column headings in the top row of the prime implicant table since these terms do not need to be covered by the minimal expression. Their omission leaves the specification of the "don't-care" terms open. If the final minimal expression contains a prime implicant that is derived from a "don't-care" combination, then that combination has been assigned an output value of "one"; otherwise, an output value of "zero." The handling of "don't-care" terms in this manner insures that the minimal solution of the function obtained from the prime implicant table covers only those minterms for which an output was specified.

On the right-hand side of the table, the designer constructs columns pertaining to the "cost" of each prime implicant. These are to aid in the selection of the minimal circuit design. The first column is the gate cost, C_G, for the first level of gating. The second column is the gate-input cost for inputs to this first level of gating, denoted by C_{I1}. The third column is the gate-interconnection cost for inputs to the second level of gating, denoted by C_{I2}. Table VII shows the prime implicant table for Example 2, with the cost columns on the right-hand side.

For the multiple-output problem, the prime implicant table is formed in much the same way as for the single-output problem. The only difference is that there must be one column for each minterm of each output function. In order to aid the designer in the reduction of the table, the table is subdivided into groups of columns for each function so that all the minterms corresponding to one of the

TABLE VII

PRIME IMPLICANT TABLE FOR EXAMPLE 2

	1	4	7	14	17	20	21	22	23	C_G	C_{I1}	C_{I2}
0(1) = A	X									1	4	1
0(4) = B		X								1	4	1
1(2,16) = C	X				X					1	3	1
4(2,16) = D		X				X		X		1	3	1
3(4,16) = E			X						X	1	3	1
6(1,16) = F			X					X	X	1	3	1
6(8,16) = G				X				X		1	3	1
17(2,4) = H					X		X		X	1	3	1
20(1,2) = I						X	X	X	X	1	3	1

output functions are represented by a set of adjacent columns in the prime implicant table. Again, there is one row in the table for each prime implicant, and there must be at least one "X" per column for each minterm of each function.

The three columns pertaining to cost are again drawn at the right-hand end of the prime implicant table. The gate cost C_G and cost of first-level inputs, C_{I1}, are the same as in the single-output problem. The cost of gate interconnections differs, however. The value of C_{I2} will vary according to the number of functions for which that prime implicant can be used. For each function for which that prime implicant is contained in the minimal circuit, one additional interconnection to feed the output gate for that function must be added. Table VIII shows the prime implicant table for Example 3. Note that prime implicant I, which covers only "don't-care" terms, is included only for the sake of completeness. The value of C_{I2} for prime implicant A indicates that that prime implicant may be used in the minimal circuit for one or two of the functions.

II. REDUCTION OF THE TABLE

If any column in the prime implicant table contains a single "X," the prime implicant that corresponds to that row is an essential prime implicant and must be included in all irredundant expressions for that function, since the input combination for that column is covered by that prime implicant alone and no other. The "X" in that column is circled and an asterisk is placed next to the prime implicant to indicate its essentiality. Also, all the minterms in the corresponding

TABLE VIII

PRIME IMPLICANT TABLE FOR EXAMPLE 3

Prime Implicants	Minterms																		Cost Factors		
	F_a						F_b					F_c							C_G	C_{I1}	C_{I2}
	2	4	10	11	12	13	4	5	10	11	13	1	2	3	10	11	12				
13 = A						X					X								1	4	1,2
1(2) = B												X		X					1	3	1
2(8) = C	X		X										X		X				1	3	1,2
4(1) = D		X					X	X											1	3	1,2
5(8) = E						X		X			X								1	3	1,2
10(1) = F			X	X					X	X					X	X			1	3	1,2,3
12(1) = G					X	X											X		1	3	1,2
11(4) = H																X			1	3	1
13(2) = I																			-	-	-
2(1,8) = J													X	X	X	X			1	2	1
4(1,8) = K		X			X	X													1	2	1

columns for the appropriate function that the essential prime implicant also covers are checked off at the top of the table. In Table VII, page 40, the prime implicant corresponding to row G is essential. That term covers minterms 14 and also 22, both of which may be checked off in the top row of column headings.

The prime implicant table can be further reduced in size by the application of the theorems of dominance. Two rows of a prime implicant table that have "X's" in exactly the same columns are said to be equal. One of the pair of rows can be removed from the table, and the one whose corresponding prime implicant has less total cost is the one that is chosen for removal.

A row of a prime implicant table dominates another row of the same table if the first row contains an "X" in every column that the second row contains an "X" and also has at least one "X" in a column in which the second row does not have an "X." That is, the first row covers every column covered by the second row. If the prime implicant corresponding to the first row does not "cost" any more than the prime implicant which corresponds to the second row, then that second row, called the dominated row, can be deleted from the prime implicant table. That first row can replace the second row without increasing the "cost" of the solution, since the "cost" of the first row is no greater than that of the second row.

For example, in Table VII, row E is dominated by row F. Since the cost of the prime implicant corresponding to row F is no greater than that of the prime implicant corresponding to row E, row E may be eliminated from the table. Row A is dominated by row C, and, since

the "cost" of C is no more than the "cost" of A (in fact, the cost of C is less), row A may also be eliminated from the table. Likewise, row B, dominated by row D, may be eliminated, too.

One column of a prime implicant table is said to dominate another column of that table if the first column has "X's" in all the rows that the second column has "X's" and if the first column has an "X" in at least one row that the second column does not. Consider the columns corresponding to minterms 21 and 23, which will be referred to as "column 21" and "column 23," in Table VII, page 40, as an aid in the understanding of column dominance. If column 23 were removed, in order to cover column 21, either row H or row I must be selected; column 23 then will automatically be covered since that column contains "X's" in row H and row I. But if column 21 were removed, row E or row F could be selected to cover column 23, and column 21 would not be covered at all. Thus, the dominant column of a pair of columns in the prime implicant table can be removed without affecting the minimal solution if there exists another column with "X's" in every row that the dominant column has "X's." The "cost" of the prime implicant is not involved in column dominance.

In Table VII, in addition to the removal of the column corresponding to minterm 23, the column for minterm 22 may also be removed. This column dominates the column for minterm 20.

In the case of the multiple-output function, a prime implicant may be a prime implicant for any number of output functions, but may or may not be essential to all these functions. Consider the prime implicant table for Example 3, shown in Table VIII, page 42. The prime

implicant corresponding to row C is essential to function F_a but is not essential to function F_c. Minterms 2 and 10 may be checked off in function F_a, but they may not be checked off in function F_c. Since prime implicant C is essential to function F_a, and since that prime implicant can possibly be used in another circuit implementation, its three corresponding cost factors must be changed. A gate is already necessary for row C since the prime implicant is essential to F_a. If row C is chosen later for function F_c, another gate with three inputs is not necessary; only one additional interconnection from that gate to the output gate for function F_c is necessary. Hence, for the prime implicant corresponding to row C, C_G goes to "zero"; C_{I1} becomes "zero" also; and C_{I2} is now set to "one," its value having been decreased by one.

For the multiple-output function, the same basic dominance rules apply as for the single-output function. Row dominance is handled in exactly the same way. In Table VIII, page 42, row H may be eliminated since row J covers every minterm that row H covers and since the cost of implementing the prime implicant corresponding to row J is less.

Column dominance has one main difference: a dominant column may be removed only if the column it dominates refers to the same function. That is, column dominance may only be applied to columns within each function. In Table VIII, the column for minterm 13, which dominates the column for minterm 12, may be removed from function F_a, and the column for minterm 5, which dominates the column for minterm 4, may be removed from function F_b. In function F_c, the columns

for minterms 3 and 10, dominating the columns for minterms 1 and 2, respectively, can be eliminated.

At any point after the removal of essential prime implicants or after the removal of dominant columns or dominated rows from the prime implicant table, the simplification process is somewhat easier for the designer if a reduced prime implicant table is formed. In this table, only the minterms that are currently left uncovered are shown as columns, and only the prime implicants that cover any of these minterms are shown as rows. This reduction in the size of the table makes the selection of the appropriate terms for the minimal solution an easier task for the designer.

Consider again the prime implicant table for Example 2, shown in Table VII, page 40. This table can be reduced considerably by applying some of the reduction principles. Row G is essential; therefore, the columns corresponding to minterms 14 and 22 may be checked off. Row A, dominated by row C, may be removed, as may row B, dominated by row D. The column for minterm 17 dominates the column for minterm 1, since the latter column contains only the "X" in row C (after the removal of row A), and thus can be eliminated. Likewise, the column for minterm 23, dominate over the column for minterm 21, can be removed from the table. The resulting reduced prime implicant table is shown in Table IX.

The same principles of reduction can be applied to the reduced prime implicant table, also. In Table IX, C and D are now essential; they are called secondary essential prime implicants. Since the column for minterm 20 is also covered by D, only columns for minterms

TABLE IX

REDUCED PRIME IMPLICANT TABLE FOR EXAMPLE 2

Prime Implicants	Minterms					Cost Factors		
	1	4	7	20	21	C_G	C_{I1}	C_{I2}
1(2,16) = C	X					1	3	1
4(2,16) = D		X		X		1	3	1
3(4,16) = E			X			1	3	1
6(1,16) = F			X			1	3	1
17(2,4) = H					X	1	3	1
20(1,2) = I				X	X	1	3	1

7 and 21 remain. Each column is covered by two prime implicants, each of which has the same cost. Therefore, any of the pairs E and H, E and I, F and H, or F and I can be chosen to cover the remaining portion of the table. The minimal sums for this problem, of which there are four, are shown below. The letters have been replaced by the corresponding prime implicants.

$$F(X_1, X_2, X_3, X_4, X_5) = X_3X_4\bar{X}_5 + \bar{X}_2\bar{X}_3X_5 + \bar{X}_2X_3\bar{X}_5 + \begin{cases} \bar{X}_2X_4X_5 + X_1\bar{X}_2X_5 \\ \bar{X}_2X_4X_5 + X_1\bar{X}_2X_3 \\ \bar{X}_2X_3X_4 + X_1\bar{X}_2X_5 \\ \bar{X}_2X_3X_4 + X_1\bar{X}_2X_3 \end{cases}$$

Often, as in the case of Example 2, a minimal solution is reached after all essential prime implicants, dominated rows, and dominant columns have been detected and properly handled according to the rules for the reduction of the prime implicant table. Every minterm of every function will be covered by at least one of the prime implicants.

However, sometimes the "solution" of the prime implicant table may not be obvious to the designer. Perhaps every column of the table contains at least two "X's," and the table can be reduced no further. Such a table is called a cyclic prime implicant table.

III. CYCLIC PRIME IMPLICANT TABLES

Since no rows or columns can be removed from a cyclic prime implicant table, another method must be used to determine the minimal sum. For the single-output function, use of Petrick's method produces

all irredundant sums for the table in the form of a prime implicant function, often called a "p-function" (9). The sum with minimal "cost," according to the designer's definition of "cost," is the one chosen as the minimal solution.

The identifying letters corresponding to the prime implicants are interpreted as Boolean variables in the p-function. Each such variable is set equal to "one" if the corresponding prime implicant for that row is selected and to "zero" if not selected. For these variables, a new switching function, which is equal to "one" only when each column of the reduced prime implicant table has an "X" in at least one of the selected rows, is specified. The resulting function, in which each variable corresponds to a prime implicant of the original function, is the p-function. The only time that the p-function equals "zero" is if there is some column of the reduced prime implicant table that does not have an "X" in any selected row.

This p-function, formed directly from the reduced prime implicant table, can be obtained from an unreduced prime implicant table, giving the same results, but its use is much easier if the designer reduces the table as much as possible first. Each column contributes a factor to the p-function, which is in the form of a product-of-sums, and each sum factor lists the prime implicants that cover a certain minterm. After all the factors have been "multiplied through," a sum of several product terms results. This resulting p-function is then reduced by using the two basic theorems of switching algebra:

$$a(a + b) = aa + ab = a + ab = a(1 + b) = a(1) = a$$

$$a + ab = a(1 + b) = a(1) = a$$

In the resultant sum-of-products expression, each product term stands for a combination of prime implicants representing an irredundant solution of the reduced prime implicant table to which it was applied. Each prime implicant that was initially determined to be a part of the solution during the application of the reduction techniques must be included with these product terms. This technique thus generates all irredundant solutions to the function.

Consider the reduced prime implicant table for Example 2, shown in Table IX, page 47. Although this table is not cyclic, a p-function can be generated to show all irredundant solutions to the table. The designer must form a product-of-sums expression that indicates which prime implicants cover which minterms. Only prime implicant C covers minterm 1, and only prime implicant D covers minterm 4. Prime implicants E and F, D and I, and H and I cover minterms 7, 20, and 21, respectively. Forming these terms into a p-function results in the following product-of-terms expression, which is then expanded according to the rules of Boolean algebra.

$$P = (C)\ (D)\ (E + F)\ (D + I)\ (H + I)$$
$$= (CD)\ (E + F)\ (DH + I)$$
$$= (CDE + CDF)\ (DH + I)$$
$$= (CDEH + CDEI + CDFH + CDFI)$$

The last expression indicates that the reduced prime implicant table can be covered in four irredundant ways: (1) by prime implicants C, D, E, and H; (2) by prime implicants C, D, E, and I; (3) by prime implicants C, D, F, and H; or (4) by prime implicants C, D, F, and I. Since prime implicants C and D are essential, they necessarily appear in all four terms of the expression.

In order to determine which of the four solutions to the reduced prime implicant table is indeed minimal, the designer must consider the cost of each. By looking at the three cost columns in the prime implicant table in Table IX, page 47, the designer notes that each prime implicant has the same implementation cost. Hence, all four solutions are minimal.

The minimal solution to the function can now be obtained. The application of the reduction principles revealed only that prime implicant G is essential initially. Therefore, this prime implicant must be included in any minimal sum. Conversion of the letters representing the prime implicants is done, and the four minimal solutions, identical to the ones found earlier, are shown below.

$$F_1(X_1, X_2, X_3, X_4, X_5) = X_3X_4\bar{X}_5 + \bar{X}_2\bar{X}_3X_5 + \bar{X}_2X_3\bar{X}_5 + \bar{X}_2X_4X_5 + X_1\bar{X}_2X_5$$

$$F_2(X_1, X_2, X_3, X_4, X_5) = X_3X_4\bar{X}_5 + \bar{X}_2\bar{X}_3X_5 + \bar{X}_2X_3\bar{X}_5 + \bar{X}_2X_4X_5 + X_1\bar{X}_2X_3$$

$$F_3(X_1, X_2, X_3, X_4, X_5) = X_3X_4\bar{X}_5 + \bar{X}_2\bar{X}_3X_5 + \bar{X}_2X_3\bar{X}_5 + \bar{X}_2X_3X_4 + X_1\bar{X}_2X_5$$

$$F_4(X_1, X_2, X_3, X_4, X_5) = X_3X_4\bar{X}_5 + \bar{X}_2\bar{X}_3X_5 + \bar{X}_2X_3\bar{X}_5 + \bar{X}_2X_3X_4 + X_1\bar{X}_2X_3$$

In early 1970, Robert M. Bowman and E. S. McVey introduced a technique for obtaining quite rapidly an approximate solution to the cyclic prime implicant table (10). In this method, the designer must eliminate the essential prime implicants before applying this new technique of incomplete branching, with the branching point being the column that contains the fewest number of "one" entries. This selection of the branching point means that the smallest number of possible branches will be considered. The technique involves choosing the

"difficult-to-cover" column first; that is, the one that contains the fewest number of "one" entries. The proper prime implicant to cover that column is chosen based upon the value of its "figure of merit." For each column, there is a weighting function which is the total number of prime implicants divided by the number of "one" entries in that column. The figure of merit for a particular prime implicant is the sum of the weighting functions for each column which that prime implicant covers. The row which has the largest figure of merit is chosen, and its corresponding prime implicant is selected as a member of the final cover. The authors, Bowman and McVey, indicate a savings of 98 percent in computation time for finding this near-minimal cover of the cyclic prime implicant table and that, if the prime implicant table is not cyclic, the solution that results is a minimal one.

The steps involved in this method are as follows:

Step 1. Choose all essential prime implicants as members of the final cover. Remove from the table all minterms covered by these essential prime implicants.

Step 2. Beginning with N = 2, look for all columns containing N "one" entries. If there is more than one such column, the choice is arbitrary. One of the rows that corresponds to one of the "one" entries in the selected column must be a member of the final cover.

Step 3. For the N prime implicants (one for each "one" entry) that cover the chosen column, search the rows corresponding to those

prime implicants and locate all the "one" entries in each row. Determine the weighting functions for the columns that contain those "one" entries. Add these weighting functions to determine the figure of merit for each of the N rows.

Step 4. Select the prime implicant row that corresponds to the largest figure of merit. This prime implicant will be a member of the final cover for the table. If there is more than one figure of merit that is the largest, the choice is arbitrary.

Step 5. Delete from the table all columns that the chosen prime implicant covers.

Step 6. Repeat steps three through five for the other columns that have exactly N "one" entries until all these columns are covered by some prime implicant in the final cover.

Step 7. Increase N to 3, 4, 5, . . . and apply steps two through six until every column has been covered by some prime implicant that is contained in the final cover.

Consider again Example 2 and its prime implicant table, which is shown in Table VII, page 40. Application of the Bowman-McVey technique to this problem will result in one of the minimal solutions that was previously found by using the p-function or by the application of the reduction principles. The steps as applied to Table VII are shown below.

Step 1'. Prime implicant G is essential and therefore must be a member of the final cover. Minterms 14 and 22 may be checked off in the prime implicant table.

Step 2'. Set N = 2. Columns 1, 4, 7, 17, 20, and 21 all have only two "one" entries. Arbitrarily select column 17.

Step 3'. Column 17 is covered by prime implicants C and H. The figure of merit must be found for these two rows.

figure of merit for row C: $\frac{8}{2} + \frac{8}{2} = 8$

figure of merit for row H: $\frac{8}{2} + \frac{8}{2} + \frac{8}{4} = 10$

Step 4'. Choose prime implicant H with the larger figure of merit, and check this one off in the prime implicant table.

Step 5'. Columns 17, 21, and 23 may be checked off in the table. Table X shows the prime implicant table at this point.

Step 2'a. Arbitrarily choose column 7, another of the columns containing only N "one" entries.

Step 3'a. Column 7 is covered by prime implicants E and F.

figure of merit for row E: $\frac{8}{2} = 4$

figure of merit for row F: $\frac{8}{2} = 4$

Step 4'a. Arbitrarily select and check off prime implicant F.

Step 5'a. Column 7 may be checked off in the prime implicant table.

Step 2'b. Column 20 is chosen next.

Step 3'b. Prime implicants D and I cover this column.

TABLE X

PRIME IMPLICANT TABLE FOR EXAMPLE 2 AFTER PARTIAL COMPLETION OF BOWMAN-McVEY METHOD

Prime Implicant	Minterms 1	4	7	14 ✓	17 ✓	20	21 ✓	22 ✓	23 ✓
0(1) = A	X								
0(4) = B		X							
1(18) = C	X				X				
4(18) = D		X				X		X	
3(20) = E			X						X
6(17) = F			X					X	X
*6(24) = G				X				X	
√17(6) = H					X		X		X
20(3) = I						X	X	X	X

figure of merit for row D: $\frac{8}{2} + \frac{8}{2} = 8$

figure of merit for row I: $\frac{8}{2} = 4$

Step 4'b. Select and check off prime implicant D, which has the larger figure of merit.

Step 5'b. Check off columns 4 and 20 in the prime implicant table.

Step 2'c. Column 1 is chosen next.

Step 3'c. Prime implicants A and C cover this column.

figure of merit for row A: $\frac{8}{2} = 4$

figure of merit for row C: $\frac{8}{2} = 4$

Step 4'c. Prime implicant C is chosen arbitrarily and is checked off in the prime implicant table.

Step 5'c. Column 1 can be checked off.

At this point, every minterm has been covered. The prime implicants that belong to the final cover are G, H, F, D, and C. These prime implicants correspond to the following sum-of-products expression, which, in this case, happens to be a minimal solution for Example 2, as was shown earlier.

$$F(X_1, X_2, X_3, X_4, X_5) = X_3X_4\bar{X}_5 + X_1\bar{X}_2X_5 + \bar{X}_2X_3X_4 + \bar{X}_2X_3\bar{X}_5 + \bar{X}_2\bar{X}_3X_5$$

Consider again Example 3. The original prime implicant table is shown in Table VIII, page 42. Previous steps have denoted which prime implicants are essential to each of the three functions, and

adjustments have been made in the cost columns to accomodate these changes. Dominated rows, as well as dominant columns, have been removed from the table. The resulting reduced prime implicant table is shown in Table XI. Prime implicants which are essential to a particular function and cover minterms that occur only in that function have been eliminated for the sake of clarity.

Since every minterm is covered by at least two prime implicants, this resulting reduced prime implicant table is cyclic. Row A is equal to row E, but its cost is more than that of row E. Therefore, row A can be removed from the table. Then prime implicant E becomes essential to function F_b and need no longer be considered since it covers minterms in no other functions. Thus, function F_b has a solution consisting of prime implicants D, F, and E.

The remaining part of the prime implicant table for functions F_a and F_c is still cyclic. Since the table can be reduced no further, the designer must use the p-function to aid in obtaining the minimal circuits. This p-function is listed and expanded below.

$$p = (D + K)\ (G + K)\ (C + J)\ (F + J)$$

$$= (K + DG)\ (J + CF)$$

$$= (JK + CFK + DGJ + CDFG)$$

The next step is to check each term of the p-function to determine if any prime implicant within that term is not essential to the particular function. If any such term is found, that corresponding prime implicant should be removed from the minimal circuit only for the appropriate function. Assuming that the gate cost, the cost of gate inputs to the first level of gating, and the cost of

TABLE XI

REDUCED PRIME IMPLICANT TABLE FOR EXAMPLE 3

Prime Implicants	Minterms					Cost Factors		
	F_a		F_b	F_c				
	4	12	13	2	11	C_G	C_{I1}	C_{I2}
13 = A			X			1	4	1
2(8) = C	X			X		0	0	1
4(1) = D						0	0	1
5(8) = E			X			1	3	1
10(1) = F					X	0	0	1
12(1) = G		X				0	0	1
2(1,8) = J				X	X	1	2	1
4(1,8) = K	X	X				1	2	1

gate interconnections are all of equal importance, the designer can simply find the sum of those three cost factors for each term of the p-function. Based on this cost criterion, the solution having the least cost would be selected.

Consider the first term of the p-function, JK. Prime implicant J covers the remaining columns of function 3, and prime implicant K covers the remaining columns of function 1. However, since neither of these prime implicants has been used before, two more gates, four more inputs, and the two interconnections must be used. This adds a cost of 8 to the final solution.

For the second term of the p-function, CFK, C and F cover the remaining columns of function 3, while K covers the remaining columns of function 1. Since prime implicant C is essential to function 1, only 1 interconnection must be added in order to use that prime implicant in function 3. The use of prime implicant F in function 3 involves the same situation, since F is essential to function 2. Since prime implicant K, which covers the remaining columns of function 1, has not been used before, a cost of 4 (1 gate, 2 inputs, and 1 interconnection) must be added to the cost of implementing the final solution. The total cost of using term CFK is then 6.

The use of the third term of the p-function, DGJ, involves the same situation as the use of term CFK. The total cost is 6, since prime implicants D and G have been used before, and prime implicant J has not.

Consider the fourth term of the p-function, CDFG. All four prime implicants have been used previously, and, therefore, only four

interconnections must be added in order to use these prime implicants in the final solution. This term represents the solution of minimal cost. Prime implicants D and G are used in function 1, and prime implicants C and F are used in function 3.

The minimal solution for Example 3 is shown below, and the circuit diagram is shown in Figure 8.

$$F_a(X_1, X_2, X_3, X_4) = \bar{X}_2X_3\bar{X}_4 + X_1\bar{X}_2X_3 + \bar{X}_1X_2\bar{X}_3 + X_1X_2\bar{X}_3$$

$$F_b(X_1, X_2, X_3, X_4) = \bar{X}_1X_2\bar{X}_3 + X_2\bar{X}_3X_4 + X_1\bar{X}_2X_3$$

$$F_c(X_1, X_2, X_3, X_4) = \bar{X}_1\bar{X}_2X_4 + X_1X_2\bar{X}_3 + \bar{X}_2X_3\bar{X}_4 + X_1\bar{X}_2X_3$$

The use of the prime implicant table is a tremendous aid to the designer in determining all possible solutions to the problem and in choosing the solution that is minimal, according to his own cost criteria. One point that should be noted is that the previously mentioned methods do not consider fan-in and fan-out. Physical limitations of the logic gates as far as fan-in and fan-out limits can prohibit the implementation of a solution that has been deemed a minimal one.

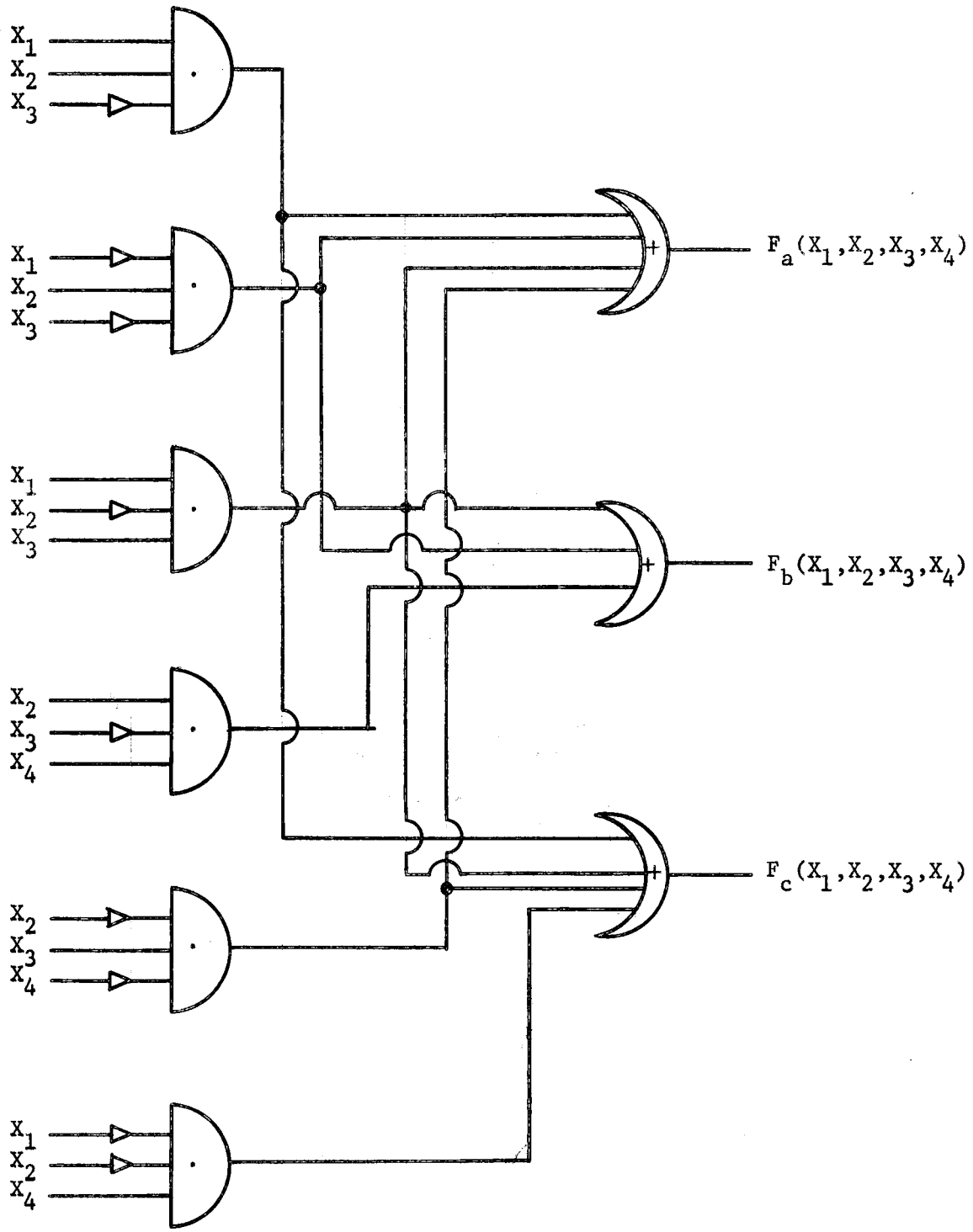


Figure 8. Circuit diagram for Example 3.

CHAPTER V

MINIMIZATION WITH THE AID OF A DIGITAL COMPUTER

The digital computer program which is described in this chapter is designed to accept minterms and maxterms of a switching function as input and to compute a minimal solution which is based on user-specified cost factors. The program is written in PL/1 for use with the IBM/360 MODEL 65 computer. This language was chosen because of its versatility in handling bit strings and its capability of dynamically allocating storage. Since the Quine-McCluskey minimization method as outlined in Chapter III is the theoretical basis of the program, no further elaboration will be given here concerning this aspect of the program. This chapter will explain the program's input sequence, general techniques used in writing the program, and interpretation of the output. A general flowchart for the program is included in Appendix A. The program listing is shown in Appendix B, and sample inputs and outputs are shown in Appendix C.

The limitations of the program should be pointed out before any explanation of the specific input sequence is given. Only one problem may be solved on a given computer run. The program will accept a minimum of two and a maximum of ten input variables. A minimum of one and a maximum of ten output functions can be handled. For any given problem, the input terms can be either minterms or maxterms or both. Any combination is permissible. These terms will be converted according to the desired form of the solution. The result will be given in the sum-of-products form, product-of-sums form, or both,

depending on the request of the user. The minimal solution is one which minimizes the following cost function:

$$\text{Cost} = (X_1 \cdot A) + (X_2 \cdot B) + (X_3 \cdot C) = \text{minimum.}$$

The values of A, B, and C, which are determined by the program, and the values of variables X_1, X_2, and X_3, which are set by the user, are as follows:

A = Number of gates needed to implement the solution.

B = Number of inputs to the first level of gating.

C = Number of inputs to the second level of gating, called interconnections.

X_1 = Cost factor for AND-gates, if the result is to be in the sum-of-products form, or OR-gates, if the result is to be in the product-of-sums form.

X_2 = Cost factor for inputs (inverted or uninverted) to the first level of gating.

X_3 = Cost factor for inputs or interconnections to the second level of gating.

The input card sequence used with the program is shown below. All cards are required in the proper sequence. If any card is not in the proper sequence or contains any invalid information, an appropriate diagnostic will be printed, and execution will terminate. After the processing of the input terms begins, if any error occurs, the remaining input cards will be processed and checked only for valid characters. Upon completion of the input phase, execution will then terminate.

Card 1. User identification card containing any identifying information the user desires.

Column 1--The character "X." This character is required, regardless of whether or not the user desires any identification line.

Columns 2-80--Any information chosen by the user to identify the problem. This information, which may be blank, will be printed at the top of each page of the output listing.

Card 2. Card identifying type of output listing desired.

Column 1--The character "X."

Columns 2-9--Either the word "COMPLETE" or the word "PARTIAL" must be in the appropriate columns. If the user desires his output listing to consist of only the input data, the prime implicant table, and the solution, "PARTIAL" should be specified. Otherwise, the output listing will also include information as to how the minimal solution was obtained.

Card 3. Number of input variables.

Columns 1-2--The number of input variables. This number must be right-justified in the field if its value is less than ten. The number may have a leading zero or a leading blank.

Columns 3-80--Any identification that the user wishes.

Card 4. Number of output functions.

Columns 1-2--The number of output functions. This number, if less than ten, may be preceded by a leading blank or zero.

Columns 3-80--Any identification that the user wishes.

Card 5. Cost factor card. The maximum value for each cost factor is 9999.999. Each cost factor must be contained in a seven column field. If the decimal point is included, the value may be anywhere within the appropriate seven columns. If no decimal point is present, the value will be considered as seven digits, with the decimal point between the fourth and fifth digits from the left. All blanks are converted to zeroes.

Column 1--The character "X."

Columns 10-16--The cost factor for AND gates. If the result is to be minimized in the sum-of-products form, this cost factor must have a positive value.

Columns 20-26--The cost factor for OR gates. If the result is to be minimized in the product-of-sums form, this cost factor must have a positive value.

Columns 30-36--The cost factor for inputs to the first level of gating. Must always be a positive number.

Columns 40-46--The cost factor for inputs to the second level of gating. Must always be a positive number.

Card 6. Solution specification card. This card is free-form. The terms "SUM OF PRODUCTS" or "PRODUCT OF SUMS" or both terms must appear somewhere on the card. The input terms will be converted to the necessary form. If a solution in both forms is desired, the form for which the most terms of one kind are given as input will be determined first. If an equal number of minterms and maxterms for

the output functions is given, the sum-of-products solution is arbitrarily determined first.

Card 7. Input term cards. There must be one set of cards specifying the input terms for each function, and there must be at least three cards within each set. The first card of the set must contain either the word "MIN" for minterms or "MAX" for maxterms in columns 1-3. Any other identifying information may be contained in columns 4-80. The next cards following this card contain the minterms or maxterms for the appropriate function. These cards have a very free format. The individual terms must be separated by either a blank or a comma or any combination of the two. Consecutive terms may be represented by a hyphen optionally surrounded by blanks between the first and last terms of the series. A hyphenated combination may be broken on two cards. These terms need not be in sorted order. The next card following the last of the minterms or maxterms must contain the word "PHI" in columns 1-3, with identifying information in columns 4-80. Even if there are no phi-terms for the function, the "PHI" card must be present. Following this card are the phi-terms, with the same format as the minterm or maxterm cards. There must be one set of these cards, consisting of a "MIN" or "MAX" card, the appropriate terms, followed by a "PHI" card and the appropriate terms, for each output function. If there are any duplicate terms within either the care-terms or the don't-care-terms for a particular function, the duplicate term will be eliminated, and execution will continue. If, by accident, a term is included as both a care-term and a don't-care-term within the same function, an appropriate message will be printed, and execution will terminate.

The program consists of a main routine, which in turn calls five overlaid subroutines and two internal subroutines. Since each of the five overlaid subroutines can be executed independently of the others, only one division needs to be in the computer's main memory at one time. Therefore, the overlay process was used in order to lessen the amount of the main memory required by the program.

In the main routine, the first five data cards are read. The tables used to handle the input terms are allocated in this routine, and their size is based on the number of input variables and output functions. Dynamic allocation of all storage is used throughout the program. Its use assures that a small problem will run in less core than a large problem since the relevant tables are only as large as necessary.

The first subroutine, PREPRO, processes the input terms, verifies that each term is within the proper range of values based on the number of input variables, makes various checks on the input data, and, finally, sorts the terms in preparation for a later subroutine. The input data as verified by this subroutine is then printed.

After the printing, the main routine determines if the type of input terms corresponds to the requested form of the solution. If not, subroutine CONVERT changes the terms to the appropriate type. The resulting terms after the conversion are printed by this routine.

Subroutine MERGE combines each care and don't-care term from all the different functions into one table of unique terms. These terms are sorted; the primary sort is on the number of ones contained in the binary representation of the term, and the secondary sort is

on the decimal value of the term itself. This sort is in preparation for the generation of the prime implicants, which is done by the next subroutine, GENERPI.

This subroutine generates the prime implicants exactly as the Quine-McCluskey method does. However, instead of continually generating new tables, the program uses only two tables for the prime implicant generation and one table for the storage of the prime implicants. From the first table, which is set up by a previous subroutine to contain all the different input terms and their respective tags in sorted order, the second table of possible prime implicants is formed. All terms from the first table that cannot combine to form new terms and their respective tags are then stored in the table of prime implicants, called PRIME. Then, after switching the external pointers that point to these two tables, the generation of prime implicants continues. However, what was the second table is now the first table since the pointers were exchanged. This process of storing the prime implicants as they are generated and of generating a new table after the pointers have been switched continues until no more terms can combine.

The representation of a prime implicant was made considerably easier by using a single value to represent the variables that have been eliminated. For example, the prime implicant denoted by 0(8,4,1) is stored as 0(13). By this method, only one value is needed for each prime implicant to represent all the variables that have been eliminated.

At this point, one limitation of the program should be pointed out. The two tables that generate the prime implicants and the final

table containing all the prime implicants are limited to 500 entries each. No documentation of the possible number of prime implicants generated by a particular problem with a given number of inputs and outputs could be found. Therefore, the author chose to arbitrarily select the size of 500. In the generation of the prime implicants, no more than 500 implicants can be generated at any one time in one of the tables, and the total number of prime implicants cannot exceed 500. Should either situation occur, the problem can be eliminated by increasing the size of all three tables.

The next subroutine, COVERIT, determines the terms that each prime implicant covers, and with this information, sets up the prime implicant table structure. This table contains one extra row and one extra column of flag bits which are initially turned off. Whenever a row or a column is eliminated from the table, the flag bit is turned on, indicating such. This subroutine prints the prime implicant table exactly as shown in Table IV, page 31, and also sets up the table of costs for the prime implicants. This table contains the total cost of implementing each prime implicant based on the number of input variables, the number of gates, the number of interconnections, and the user-supplied cost factors.

Subroutine REDUCE applies the various reduction techniques to the prime implicant table. Initially, all essential prime implicants and all the terms that they cover are removed. Then dominant columns and dominated rows are eliminated. This process repeats until either the functions are completely covered or until no more reduction can be done. In the latter case, a cyclic prime implicant table is the

result. In subroutine CYCLIC, the p-function for each output function is individually determined. Redundant terms are eliminated as they occur in order that no memory is used for the storage of unnecessary terms. Only three tables are used in producing the final p-function for each output function. The size of one table is set according to the maximum number of terms that could be generated if none were redundant. However, the other table in which this information is later stored is allocated according to the actual number of members that remain after all the redundant terms have been eliminated. This method was used in order to most efficiently utilize available memory.

After the p-function for each different output function has been found and stored in a stack, these results are popped out of the stack and into a table for finding the final p-function. For a fairly large problem of, for example, seven or more outputs, the final p-function can contain many thousands of terms. However, PL/1 limits the size of a dimensioned table to 32,767 entries. In order to avoid this problem and to arrive at a guaranteed minimal solution, the only solution remaining to the author was to write much of this information on disk. However, this act would cause a tremendous increase in time necessary to arrive at the solution, since the disk file would have to be rewound several times during the processing. Therefore, the author chose to avoid this approach and to partition the functions in order to arrive at a sub-optimal solution in much less time.

The determination of the optimum method of partitioning the output functions in order to provide the minimal solution is itself worthy of much study. This author has made no attempt to select a

partitioning method that would yield such a result. However, neither was the straightforward method of partitioning the first N/2 functions and the last N/2 functions, a very simple approach, chosen. Rather the author has chosen to partition the functions in such a way as to use the least amount of memory possible. In this method, each column of a table contains all entries for the p-function for the appropriate output function. Before the beginning of the "multiplication" of two columns, corresponding to the multiplication of the p-functions for two different functions, the columns are sorted into descending order according to the number of entries in that column. Then the multiplication routine of subroutine CYCLIC processes from opposite sides of the table. In this manner, the p-function with the most number of terms is multiplied by the p-function with the least number of terms, and so on. By this method, the columns of the newly formed table are more equal in length. Had this sorting not been done, the p-function for two of the output functions, each having a very large number of terms, and the p-function for two other output functions, each with a very small number of terms, might in turn be multiplied together, resulting in a very one-sided table.

This sort-and-multiply method is used repeatedly until either the entire p-function is found or the "multiplication" of any two columns yields a maximum of more than 32,767 entries in the table. At this point, the number of partitions and the functions belonging to each partition are printed along with a message indicating that only a sub-optimal solution will result. This partitioning may or may not lead to the minimal solution, but will certainly result in

a near-minimal solution more quickly than would be obtained by not partitioning the output function. It was found during testing of the problem that only problems consisting of seven or more outputs must be partitioned. If no partitioning is necessary, then the solution obtained is indeed a minimal one.

The remaining portion of subroutine CYCLIC checks each member of the final p-function for each partition in order to determine which one can be implemented at the least cost. If there is more than one partition, the cost table is updated to reflect the selection of one of the p-function members for that partition as a member of the final solution. This final solution is printed in the main routine.

A listing of all control cards and the relative position of these cards when using the source deck is shown in Figure 9. Figure 10 shows the relative position of the control cards when using an object deck. Since comments are provided within the source program listing in Appendix B, no specific details are shown in the flowcharts in Appendix A. Appendix D shows how the load module for the program would be loaded onto disk and executed.

An estimation of the running time for a particular problem is difficult. The time element is dependent on the number of output functions, the number of prime implicants, and the difficulty in solving the prime implicant table. The first of these three factors is the only one known to the user. The largest problem attempted during program testing required only 164K of main memory. Smaller problems require as little as 108K. As a general rule for running with a source deck, for problems of four or fewer output functions, one

```
//THESIS JOB  (27401,33188,T-3,R-128K),CANTWELL
//STEP1  EXEC PL1FCLG,GOREG=128K,LKEDOPT='OVLY,LET'
//PL1.SYSIN DD *

       Main routine
       Subroutine GENERPI
       Subroutine REDUCE

*  PROCESS;
          Subroutine PREPRO
*  PROCESS;
          Subroutine CONVERT
*  PROCESS;
          Subroutine MERGE
*  PROCESS;
          Subroutine COVERIT
*  PROCESS;
          Subroutine CYCLIC

//LKED.SYSIN DD *
  ENTRY IHENTRY
  OVERLAY ALPHA
  INSERT PREPRO
  OVERLAY ALPHA
  INSERT CONVERT
  OVERLAY ALPHA
  INSERT MERGE
  OVERLAY ALPHA
  INSERT COVERIT
  OVERLAY ALPHA
  INSERT CYCLIC
//GO.SYSIN  DD *
       data cards
```

Figure 9. Control card sequence for source deck.

```
//THESIS  JOB (27401,33188,T-3,R-128K),CANTWELL
//STEP1   EXEC  PL1FLG,GOREG=128K,LKEDOPT='OVLY,LET'
//LKED.SYSIN DD *

       object deck

  ENTRY IHENTRY
  OVERLAY ALPHA
  INSERT PREPRO
  OVERLAY ALPHA
  INSERT CONVERT
  OVERLAY ALPHA
  INSERT MERGE
  OVERLAY ALPHA
  INSERT COVERIT
  OVERLAY ALPHA
  INSERT CYCLIC
//GO.SYSIN DD *
       data cards
```

Figure 10. Control card sequence for object deck.

should allow two minutes; for five or six output functions, five minutes; and for seven to ten output functions, 15 minutes.

CHAPTER VI

SUMMARY AND CONCLUSIONS

Problems in switching theory are stated in the form of Boolean functions, but these functions are rarely in a minimal form. For the sake of reliability of operation, accuracy of result, and economy in design, these Boolean functions must be converted to a minimal form before implementation. This thesis has presented two general methods for achieving such a minimal circuit network: map methods and tabular methods. Each method has certain advantages and disadvantages, and the choice of which method to use depends on the size of the particular problem and the familiarity of the designer with the method.

Because of the tremendous growth of automatic digital computers and the increasing complexity of automatic control systems and business machines, logical design by intuition has become a more and more difficult and time-consuming task. Therefore, the use of the computer in developing efficient design algorithms has become increasingly widespread. A computer program, written by the author, will determine a minimal or, in the case of a very large problem, a near-minimal solution quite rapidly. The main value of this program is its ability to rapidly determine that solution and to indicate for the user the steps taken to arrive at the final answer. The user is relieved of the time-consuming step-by-step operations that are required in order to find such a solution for a large problem. The user of the program needs to know only the format of the input data and how to interpret the output. This program does not consider the fan-in or fan-out limitations

of the electronic gates. If the designer should modify the resulting minimal two-level solution into a representation with more than two levels, a knowledge of the potential dynamic and static hazards is required. If a combinational circuit is used as part of a network containing sequential circuitry, erroneous operation will result unless all hazards can be eliminated.

Since the field of switching theory is such a large one, no attempt has been made to cover any more than one small segment in this brief analysis. Hopefully this research has served to show the importance of this special area of switching theory.

LIST OF REFERENCES

LIST OF REFERENCES

1. Ledley, Robert S. Digital Computing and Control Engineering. New York, 1960.

2. McCluskey, E. J., Jr. "Minimal Sums for Boolean Functions Having Many Unspecified Fundamental Products," Transactions AIEE (Communications and Electronics), part I, Vol. 81, 1962, pp. 387-391.

3. Baugh, Charles R., C. S. Chandersekaran, Richard S. Suill, and Saburd Muroga, "Optimal Networks of NOR-OR Gates for Functions of Three Variables," IEEE Transactions on Computers, Vol. C-21, No. 2, 1972, pp. 153-160.

4. Marcus, Mitchell P. Switching Circuits for Engineers. Englewood Cliffs, New Jersey, 1967.

5. McCluskey, E. J. Introduction to the Theory of Switching Circuits. New York, 1965.

6. Dietmeyer, Donald L. Logic Design of Digital Systems. Boston, 1971.

7. Marcovitz, Alan B. and Charles M. Shub. "An Improved Algorithm for the Simplification of Switching Functions Using Unique Identifiers on a Karnaugh Map," IEEE Transactions on Computers, Vol. C-18, No. 4, 1969, pp. 376-378.

8. McCluskey, E. J., Jr. "Minimization of Boolean Functions," The Bell System Technical Journal, Vol. 35, 1956, pp. 1417-1444.

9. Hill, Fredrick J. and Gerald R. Peterson. Introduction to Switching Theory and Logical Design. New York, 1968.

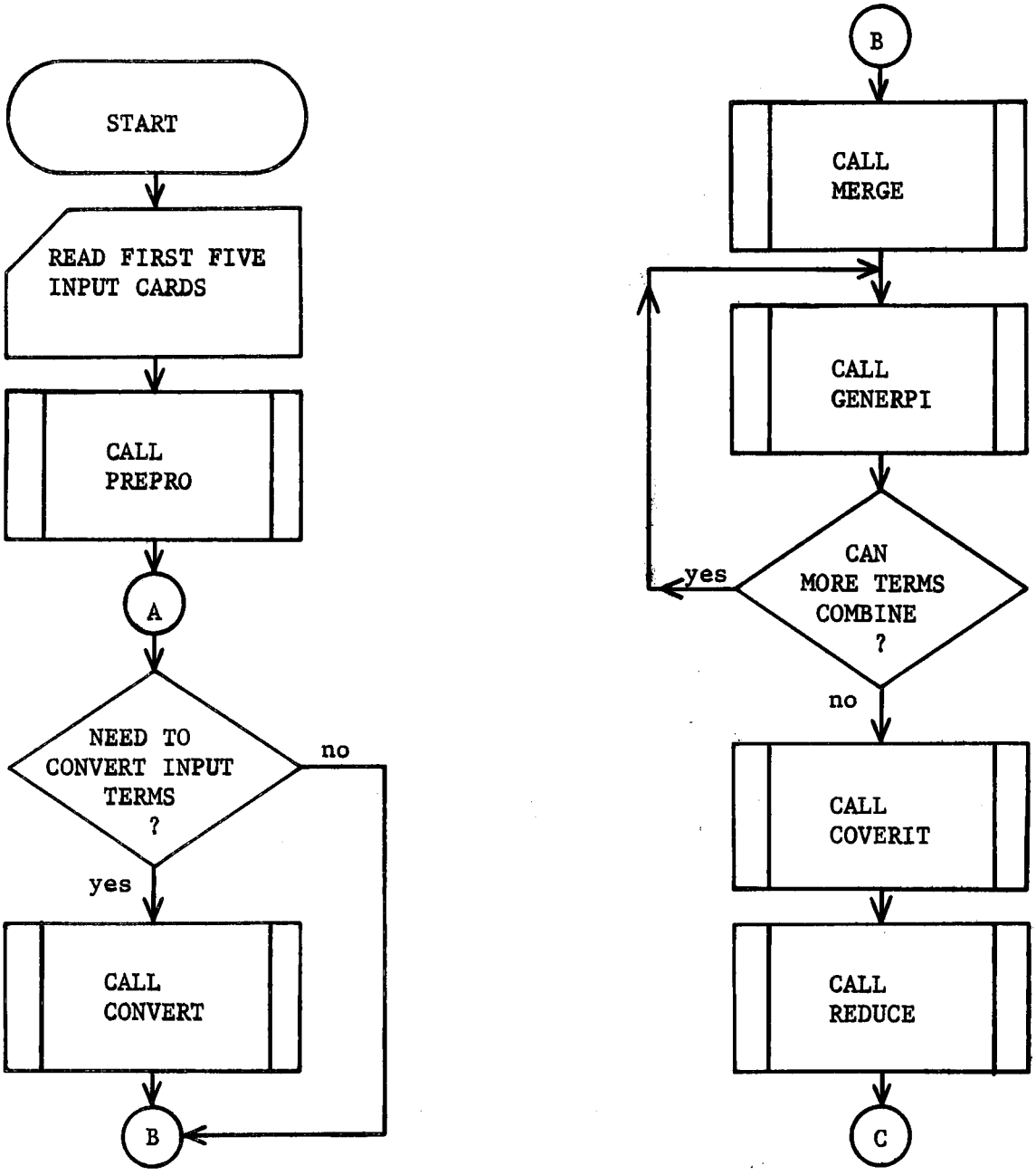
10. Bowman, Robert M. and E. S. McVey. "A Method for the Fast Approximate Solution of Large Prime Implicant Charts," IEEE Transactions on Computers, Vol. C-19, No. 2, 1970, pp. 169-173.

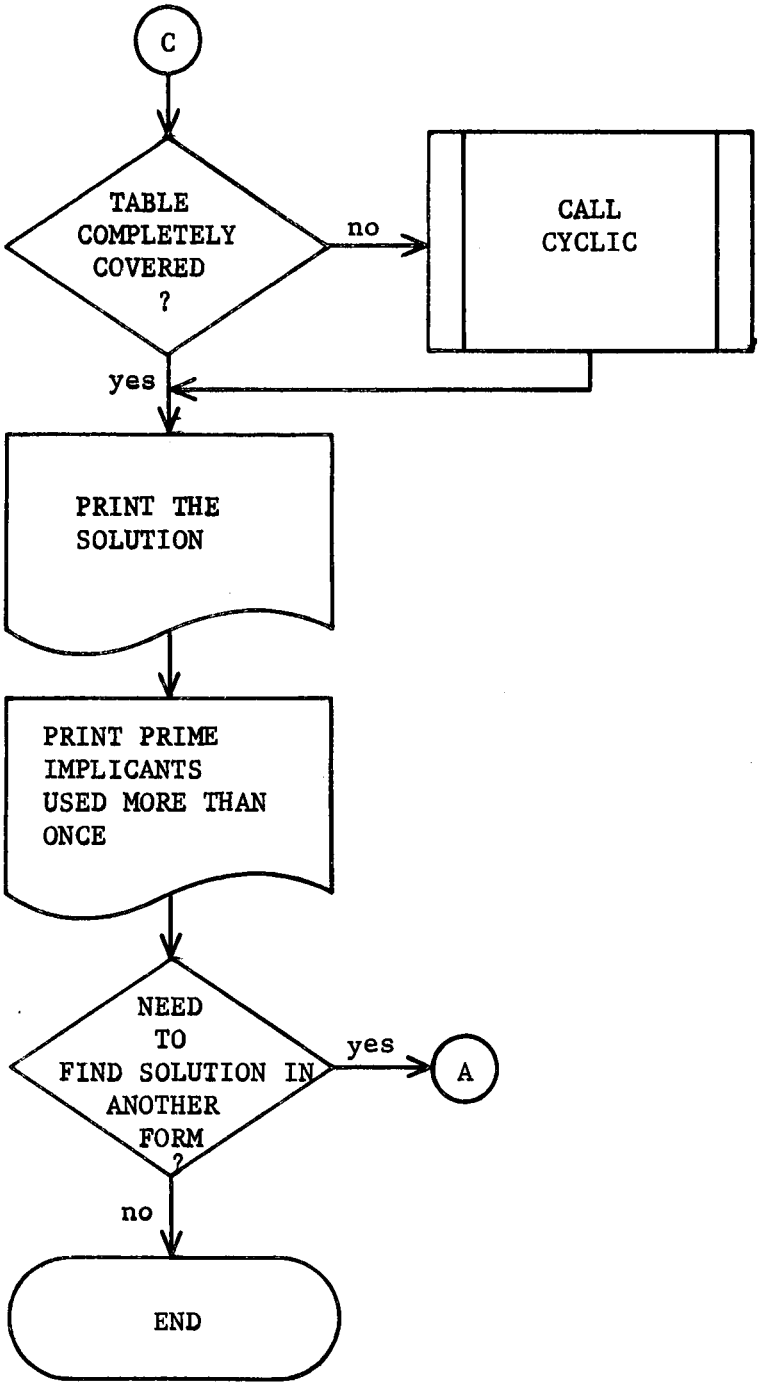
APPENDICES

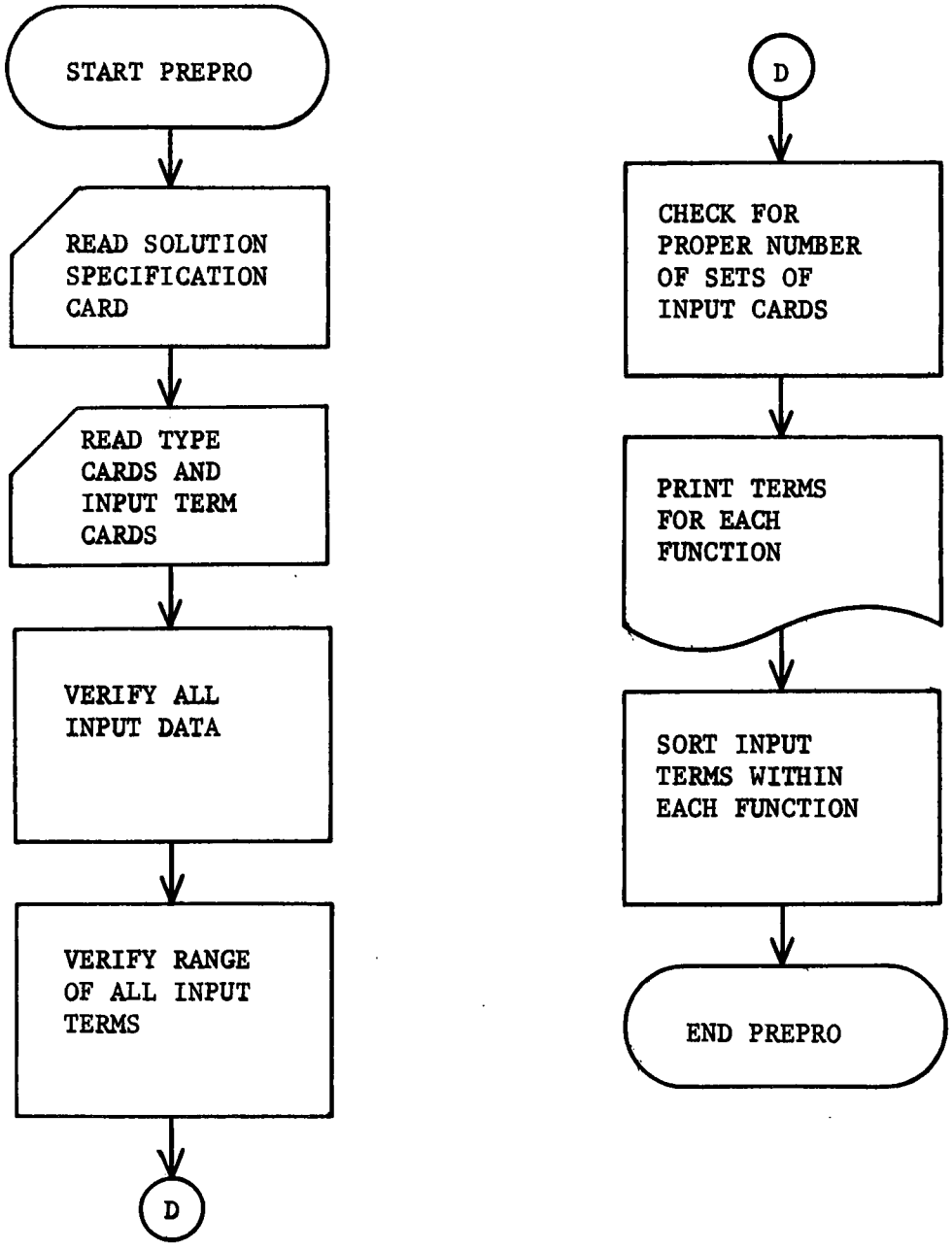
APPENDIX A

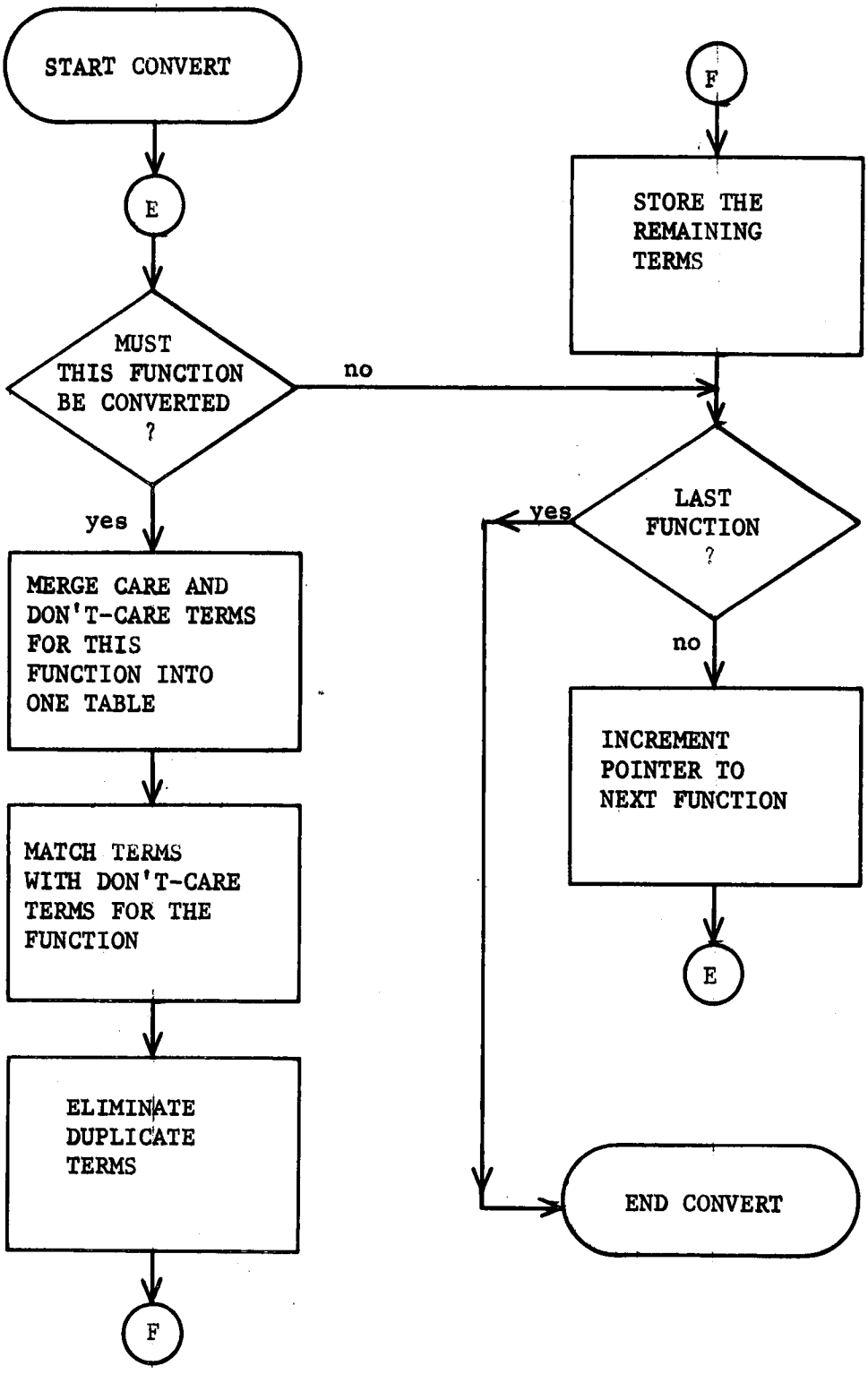
GENERAL PROGRAM FLOWCHART

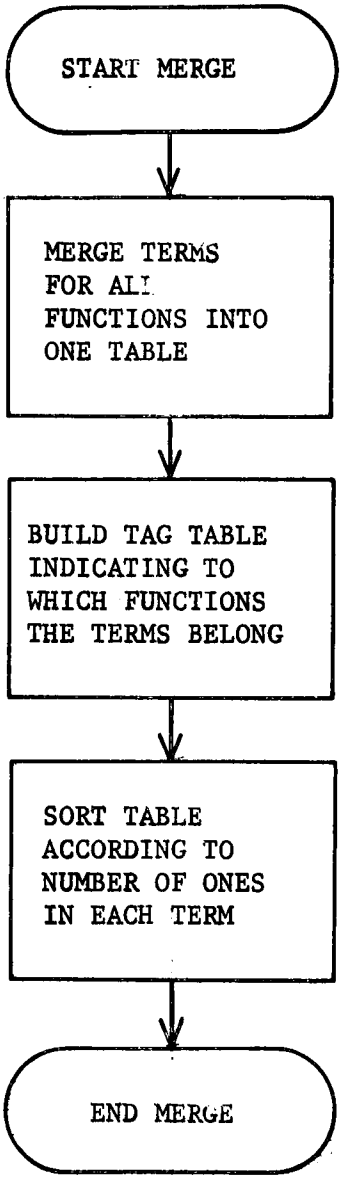
Flowchart for Main Routine

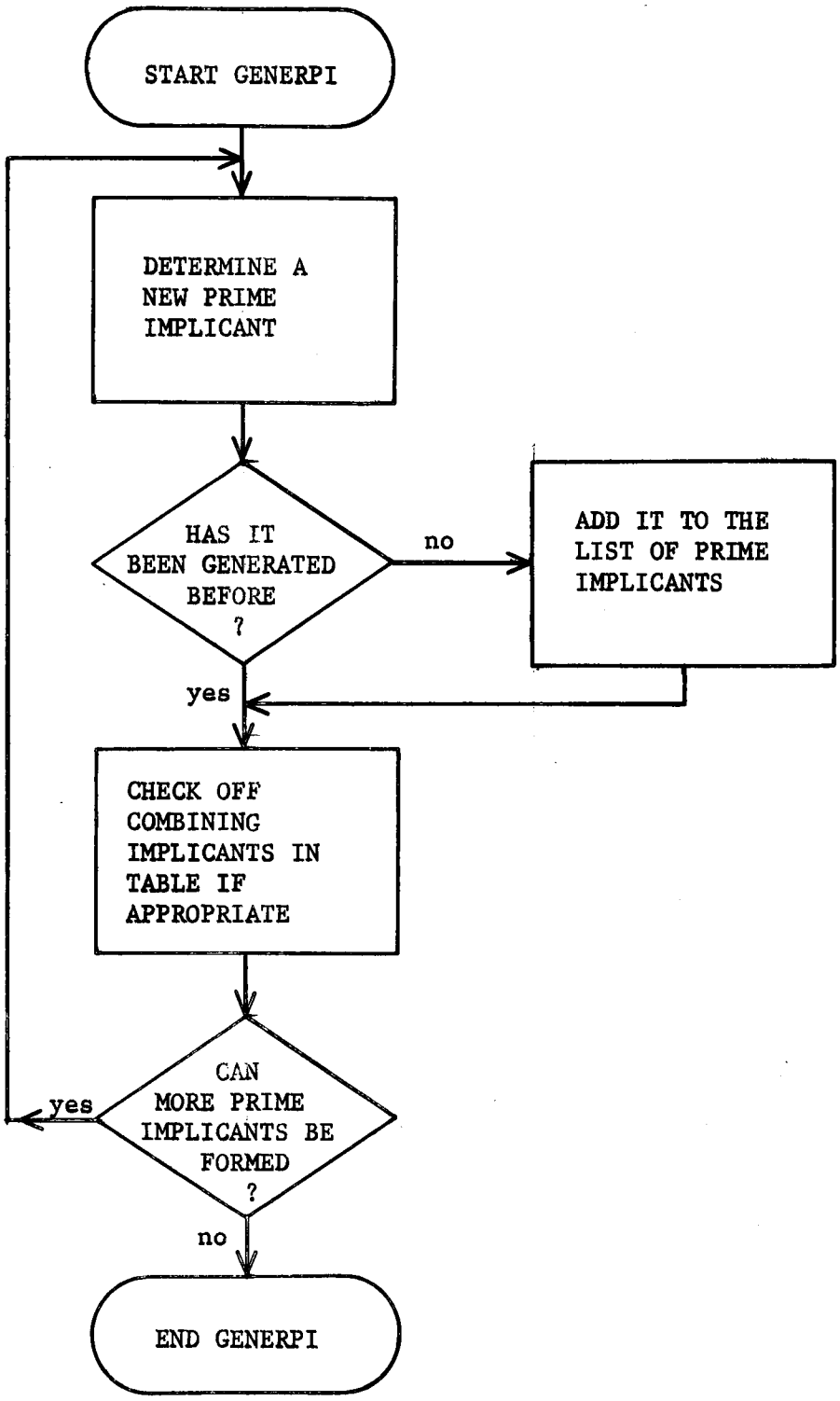


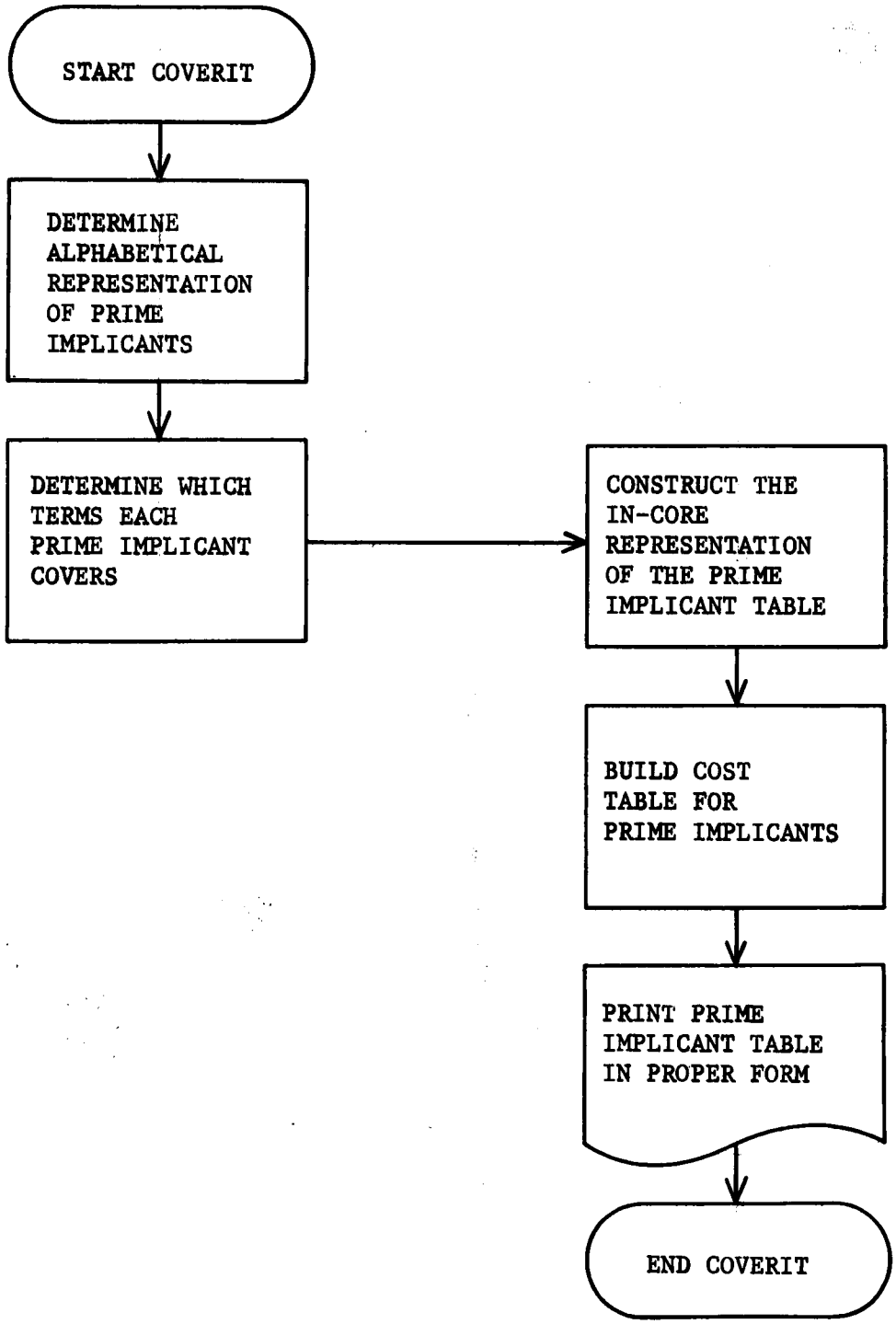

C
TABLE COMPLETELY COVERED ?
no
CALL CYCLIC
yes
PRINT THE SOLUTION
PRINT PRIME IMPLICANTS USED MORE THAN ONCE
NEED TO FIND SOLUTION IN ANOTHER FORM ?
yes
A
no
END

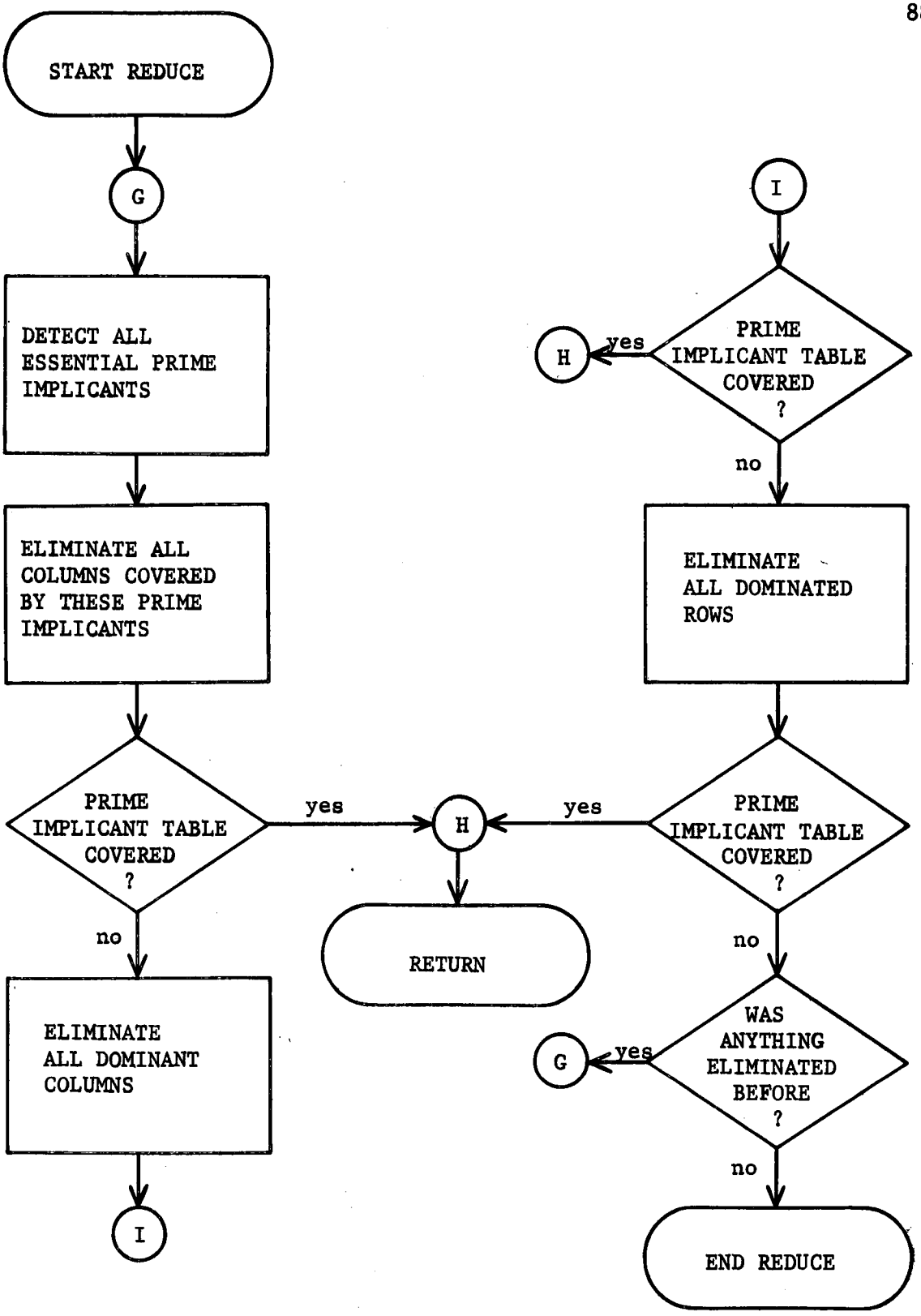

START PREPRO
READ SOLUTION SPECIFICATION CARD
READ TYPE CARDS AND INPUT TERM CARDS
VERIFY ALL INPUT DATA
VERIFY RANGE OF ALL INPUT TERMS
D
D
CHECK FOR PROPER NUMBER OF SETS OF INPUT CARDS
PRINT TERMS FOR EACH FUNCTION
SORT INPUT TERMS WITHIN EACH FUNCTION
END PREPRO

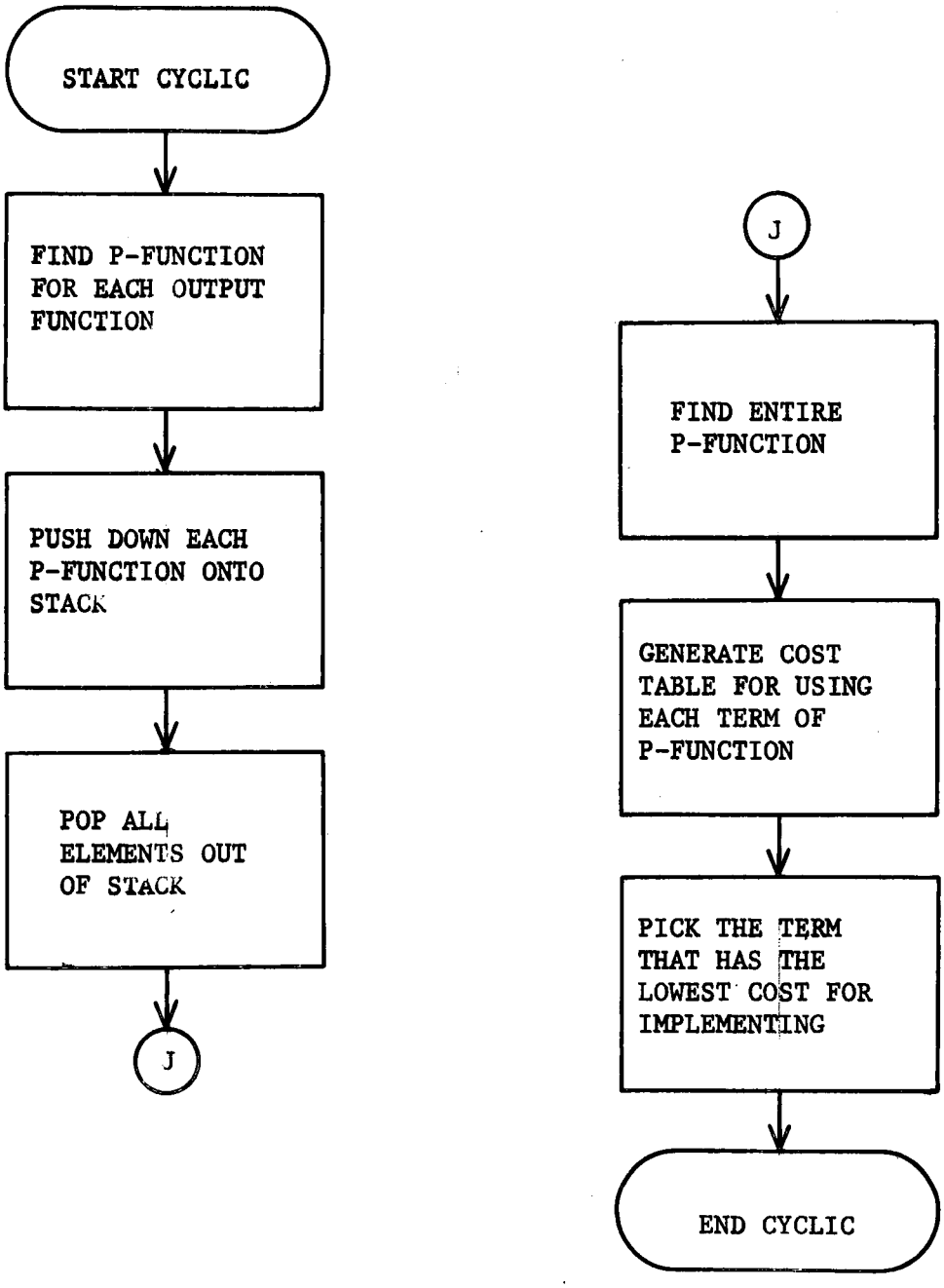

START CONVERT
E
MUST THIS FUNCTION BE CONVERTED ?
no
yes
MERGE CARE AND DON'T-CARE TERMS FOR THIS FUNCTION INTO ONE TABLE
MATCH TERMS WITH DON'T-CARE TERMS FOR THE FUNCTION
ELIMINATE DUPLICATE TERMS
F
F
STORE THE REMAINING TERMS
LAST FUNCTION ?
yes
no
INCREMENT POINTER TO NEXT FUNCTION
E
END CONVERT


START MERGE
MERGE TERMS FOR ALL FUNCTIONS INTO ONE TABLE
BUILD TAG TABLE INDICATING TO WHICH FUNCTIONS THE TERMS BELONG
SORT TABLE ACCORDING TO NUMBER OF ONES IN EACH TERM
END MERGE


START GENERPI
DETERMINE A NEW PRIME IMPLICANT
HAS IT BEEN GENERATED BEFORE ?
no
ADD IT TO THE LIST OF PRIME IMPLICANTS
yes
CHECK OFF COMBINING IMPLICANTS IN TABLE IF APPROPRIATE
CAN MORE PRIME IMPLICANTS BE FORMED ?
yes
no
END GENERPI


START COVERIT
DETERMINE ALPHABETICAL REPRESENTATION OF PRIME IMPLICANTS
DETERMINE WHICH TERMS EACH PRIME IMPLICANT COVERS
CONSTRUCT THE IN-CORE REPRESENTATION OF THE PRIME IMPLICANT TABLE
BUILD COST TABLE FOR PRIME IMPLICANTS
PRINT PRIME IMPLICANT TABLE IN PROPER FORM
END COVERIT


START REDUCE
G
DETECT ALL ESSENTIAL PRIME IMPLICANTS
ELIMINATE ALL COLUMNS COVERED BY THESE PRIME IMPLICANTS
PRIME IMPLICANT TABLE COVERED ?
yes
H
RETURN
no
ELIMINATE ALL DOMINANT COLUMNS
I
I
PRIME IMPLICANT TABLE COVERED ?
yes
H
no
ELIMINATE ALL DOMINATED ROWS
PRIME IMPLICANT TABLE COVERED ?
yes
G
no
WAS ANYTHING ELIMINATED BEFORE ?
no
END REDUCE


START CYCLIC
FIND P-FUNCTION FOR EACH OUTPUT FUNCTION
PUSH DOWN EACH P-FUNCTION ONTO STACK
POP ALL ELEMENTS OUT OF STACK
J
J
FIND ENTIRE P-FUNCTION
GENERATE COST TABLE FOR USING EACH TERM OF P-FUNCTION
PICK THE TERM THAT HAS THE LOWEST COST FOR IMPLEMENTING
END CYCLIC

APPENDIX B

PROGRAM SOURCE LISTING

```
THESIS: PROCEDURE OPTIONS (MAIN) REORDER;
DECLARE
01  MASTER                      CONTROLLED EXTERNAL,
    03  TABLE (SIZEA,SIZEB),
        05  TERM                BINARY FIXED (15),
    03  CNTTABLE (SIZEB),
        05  NUMTERMS            BINARY FIXED (15),
    03  BINARRAY (0:NUMBINTAB),
        05  NUMONES             BINARY FIXED (15),
01  PRIME (500)                 EXTERNAL,
    03  BASE                    BINARY FIXED (15),
    03  DIFFER                  BINARY FIXED (15),
    03  OUTPUTTAG               BIT (10) ALIGNED,
01  REFCOL (NUMOUTPUTS)         CONTROLLED EXTERNAL,
    03  FIRSTCOL                BINARY FIXED (15),
    03  LASTCOL                 BINARY FIXED (15),
01  REFER                       STATIC EXTERNAL,
    03  NUMVAR                  BINARY FIXED (15),
    03  NUMOUTPUTS              BINARY FIXED (15),
    03  NUMBINTAB               BINARY FIXED (15),
    03  TOPOFPAGE               CHAR (79),
    03  SIZEA                   BINARY FIXED (15),
    03  SIZEB                   BINARY FIXED (15),
    03  ESSENROW                BINARY FIXED (15),
    03  WIDTH                   BINARY FIXED (15),
    03  SIZE                    BINARY FIXED (15),
    03  POFS                    BIT (1),
    03  SOFP                    BIT (1),
01  TABLIST                     CONTROLLED EXTERNAL,
    03  PITABLE (0:ESSENROW,0:WIDTH) BIT (1),
    03  REFTAB (WIDTH),
```

```
        05  TOPROW             BINARY FIXED (15),
        05  COLREF             BINARY FIXED (15),
    03  ALLCOSTS (ESSENROW),
        05  GATECOST           BINARY FIXED (10),
        05  INPUT              BINARY FIXED (10),
        05  INTER              BINARY FIXED (10),
        05  COST               BINARY FIXED (31,12),
        05  MEMBER             BIT (NUMOUTPUTS) ALIGNED,
    03  PIPOFS (ESSENROW,SIZE) CHAR (1),
    03  PISOFP (ESSENROW,SIZE) CHAR (1),
01  SOURCETAB (500) BASED (PS),
    03  SBASENUM               BINARY FIXED (15),
    03  SNUMBERONES            BINARY FIXED (15),
    03  STOTDIFF               BINARY FIXED (15),
    03  TAG                    BIT (10) ALIGNED,
    03  ESSENSW                BIT (1) ALIGNED,
01  REDUCEDTAB (500) BASED (PR),
    03  RBASENUM               BINARY FIXED (15),
    03  RNUMBERONES            BINARY FIXED (15),
    03  RTOTDIFF               BINARY FIXED (15),
    03  RTAG                   BIT (10) ALIGNED,
    03  RESSENSW               BIT (1) ALIGNED,
ALPHABET (10) CHAR (1) INITIAL ('A','B','C','D','E','F','G','H','I',
                                'J'),
ANDCOST                        BINARY FIXED (31,12),
BLANK                          CHAR (1) INITIAL (' '),
BOTH                           BIT (1) INITIAL ('0'B),
CHECKVAL                       BINARY FIXED (31),
(CNTMAX,CNTMIN)                BINARY FIXED (15),
CNTSOL                         BINARY FIXED (15),
CCLONE                         CHAR (1),
COUNT1                         BINARY FIXED (15),
COUNT2                         BINARY FIXED (15),
(I,J,K,L,M)                    BINARY FIXED (15),
ICOUNT                         BINARY FIXED (15),
INPUTCOST                      BINARY FIXED (31,12),
```

```
INTERCOST                    BINARY FIXED (31,12),
IROW                         BINARY FIXED (15),
LISTALL                      BIT (1),
LUCKY                        BIT (1),
MATCHTYPE                    CHAR (3),
MAXK                         BINARY FIXED (15),
MAXNUMPI                     BINARY FIXED (15),
MAXROWS                      BINARY FIXED (15),
MERGENUM                     BINARY FIXED (15),
MUSTCONVERT                  BIT (1),
NUMCHAR                      BINARY FIXED (15),
NUMCOVER                     BINARY FIXED (15),
ORCOST                       BINARY FIXED (31,12),
(PS,PR,T)                    POINTER EXTERNAL,
SOLVED                       BIT (1) INITIAL ('0'B),
TYPEOUTPUT                   CHAR (8),
TYPETERM1 (1:10)             CHAR (3),
TYPETERM2 (1:10)             CHAR (3);

    OPEN FILE (SYSPRINT) LINESIZE (133);
    ON ENDPAGE (SYSPRINT) BEGIN;
      PUT FILE (SYSPRINT) EDIT (TOPOFPAGE) (PAGE,X(25),A);
      PUT FILE (SYSPRINT) SKIP(2);  END;

    POFS='0'B;  SOFP='0'B;
    OPEN FILE (SYSIN) INPUT;
    GET FILE (SYSIN) EDIT (COLONE,TOPOFPAGE) (COLUMN(1),A(1),A(79));
    IF COLONE ¬= 'X' THEN DO;
      PUT FILE (SYSPRINT) EDIT ('THE FIRST DATA CARD SHOULD CONTAIN ',
      'AN X IN COLUMN 1 AND ANY PROGRAM IDENTIFICATION THAT YOU ',
      'DESIRE ON THE REST OF THE CARD.','YOUR FIRST DATA CARD DOES ',
      'NOT HAVE AN X IN COLUMN 1.  THE CARD APPEARS ON THE NEXT LINE.',
      COLONE,TOPOFPAGE) (SKIP(3),A,A,A,SKIP,A,A,A(1),A(79));
      STOP;
      END;
    SIGNAL ENDPAGE (SYSPRINT);
```

```
GET FILE (SYSIN) EDIT (COLONE,TYPEOUTPUT) (COLUMN(1),A(1),A(8));
IF COLONE='X' THEN IF TYPEOUTPUT='COMPLETE' THEN LISTALL='1'B;
                ELSE IF TYPEOUTPUT='PARTIAL ' THEN LISTALL='0'B;
                ELSE IF TYPEOUTPUT=' PARTIAL' THEN LISTALL='0'B;
                          ELSE DO;
                            PUT FILE (SYSPRINT) EDIT ('THE CARD ',
                            'SPECIFYING TYPE OF OUTPUT LISTING IS ',
                            'INCORRECT.  THE X IS IN COLUMN 1, BUT',
                            ' THE NEXT WORD CANNOT BE DECODED.',
                            'THE CARD APPEARS ON THE NEXT LINE.',
                            COLONE,TYPEOUTPUT) (SKIP(3),A,A,A,A,
                            SKIP,A,SKIP,A,A);
                            STOP;  END;
                ELSE DO;
                  PUT FILE (SYSPRINT) EDIT ('SECOND DATA CARD ',
                  'INDICATING TYPE OF OUTPUT LISTING DESIRED IS ',
                  'EITHER OMITTED OR DOES NOT CONTAIN AN X IN ',
                  'COLUMN 1.','CHECK DATA CARDS.') (SKIP(3),A,A,A,A,
                  SKIP,A);  STOP;
                  END;
GET FILE (SYSIN) EDIT (NUMVAR) (COLUMN(1),F(2));
IF NUMVAR < 2 | NUMVAR > 10 THEN DO;
  PUT FILE (SYSPRINT) EDIT ('THIS PROGRAM LIMITS THE NUMBER OF',
  ' INPUTS TO A NUMBER BETWEEN 2 AND 10,INCLUSIVE.  YOUR SECOND ',
  'DATA CARD SPECIFIED ',NUMVAR,' INPUTS.') (SKIP,A,A,A,F(3),A);
  STOP;
  END;
PUT FILE (SYSPRINT) EDIT ('NUMBER OF INPUTS=',NUMVAR)
(SKIP(2),A,F(3));
GET FILE (SYSIN) EDIT (NUMOUTPUTS) (COLUMN(1),F(2));
IF NUMOUTPUTS < 1 | NUMOUTPUTS > 10 THEN DO;
  PUT FILE (SYSPRINT) EDIT ('THIS PROGRAM LIMITS THE NUMBER OF ',
  'OUTPUTS TO A NUMBER BETWEEN 1 AND 10, INCLUSIVE.  YOUR THIRD ',
  'DATA CARD SPECIFIED ',NUMOUTPUTS,' OUTPUTS.')
  (SKIP,A,A,A,F(3),A);
  STOP;
```

```
  END;
PUT FILE (SYSPRINT) EDIT ('NUMBER OF OUTPUTS=',NUMOUTPUTS)
(SKIP(2),A(18),F(3));
IF NUMVAR > NUMOUTPUTS THEN NUMBINTAB=2**NUMVAR-1;
                       ELSE NUMBINTAB=2**NUMOUTPUTS-1;
SIZEA=2**NUMVAR;
SIZEB=2*NUMOUTPUTS;
ALLOCATE MASTER;
ALLOCATE REFCOL;
CNTTABLE=0;
TABLE=-99;
GET FILE (SYSIN) EDIT (COLONE,ANDCOST,ORCOST,INPUTCOST,INTERCOST)
(COLUMN(1),A(1),COLUMN(10),F(7,3),COLUMN(20),F(7,3),
COLUMN(30),F(7,3),COLUMN(40),F(7,3));
IF COLONE='X' THEN;  ELSE DO;
  PUT FILE (SYSPRINT) EDIT ('THE CARD SPECIFYING COST FACTORS ',
  'DOES NOT CONTAIN AN X IN COLUMN 1.',
  'THE FIRST CHARACTER OF THE CARD IS ''',COLONE,'''.',
  'EXECUTION WILL TERMINATE.') (SKIP(3),A,A,SKIP,A,A,A,SKIP,A);
  STOP;
  END;
IF INPUTCOST <= 0 THEN DO;
  PUT FILE (SYSPRINT) EDIT ('COST FACTOR FOR INPUTS TO FIRST ',
  'LEVEL OF GATING, WHICH SHOULD BEGIN IN COLUMN 30, IS ',
  'EITHER NEGATIVE OR ZERO.','CHECK COST FACTOR CARD.  EXECUTION ',
  'WILL TERMINATE.') (SKIP(2),A,A,A,SKIP,A,A);
  STOP;  END;
IF INTERCOST <= 0 THEN DO;
  PUT FILE (SYSPRINT) EDIT ('COST FACTOR FOR INPUTS TO SECOND ',
  'LEVEL OF GATING, WHICH SHOULD BEGIN IN COLUMN 40, IS NOT ',
  'POSITIVE.  CHECK COST FACTOR DATA CARD.') (SKIP(2),A,A,A);
  STOP;  END;
PUT FILE (SYSPRINT) EDIT ('COST FACTOR FOR AND GATES IS ',ANDCOST)
(SKIP(4),A,F(5,2));
PUT FILE (SYSPRINT) EDIT ('COST FACTOR FOR OR GATES IS ',ORCOST)
(SKIP(2),A,F(5,2));
```

```
PUT FILE (SYSPRINT) EDIT ('COST FACTOR FOR EACH FIRST LEVEL INPUT',
' IS ',INPUTCOST) (SKIP(2),A,A,F(5,2));
PUT FILE (SYSPRINT) EDIT ('COST FACTOR FOR EACH SECOND LEVEL ',
'INPUT IS ',INTERCOST) (SKIP(2),A,A,F(5,2));

CALL PREPRO (TYPETERM1,TYPETERM2,COUNT1,COUNT2,ORCOST,ANDCOST);
PUT FILE (SYSPRINT) SKIP(2);
PUT FILE (SYSPRINT) EDIT ('INPUT TERMS AS READ IN')
(SKIP(3),X(15),A);
DO I=1 TO NUMOUTPUTS;
  J=2*I-1;
  PUT FILE (SYSPRINT) EDIT  (TYPETERM1(I),'TERMS FOR FUNCTION ',
  I,':') (SKIP(2),A(3),A,F(3),A);
  IF NUMTERMS(J)=0 THEN PUT FILE (SYSPRINT) EDIT ('NONE') (X(4),A);
  ELSE PUT FILE (SYSPRINT) EDIT ((TERM(K,J) DO K=1 TO NUMTERMS(J)))
  (20 (X(1),F(4)),SKIP(1),X(26));
  J=J+1;
  PUT FILE (SYSPRINT) EDIT (TYPETERM2(I),'TERMS FOR FUNCTION ',
  I,':') (SKIP,A(3),A,F(3),A);
  IF NUMTERMS(J)=0 THEN PUT FILE (SYSPRINT) EDIT ('NONE') (X(4),A);
  ELSE PUT FILE (SYSPRINT) EDIT ((TERM(K,J) DO K=1 TO NUMTERMS(J)))
  (20 (X(1),F(4)),SKIP(1),X(26));
  END;

CNTSOL=0;  /*  COUNTS NUMBER OF SOLUTIONS TO FIND                  */
CNTMIN=0;  /*  WILL COUNT NUMBER OF FUNCTIONS THAT HAVE            */
CNTMAX=0;  /*  MAXTERMS OR MINTERMS AS INPUT                       */
DO I=1 TO COUNT1;
  IF TYPETERM1(I)='MAX' THEN CNTMAX=CNTMAX+1;
  ELSE CNTMIN=CNTMIN+1;
  END;  /*  END LOOP I  */
MUSTCONVERT='1'B;  /* IF SWITCH REMAINS ON, THEN THE SUBROUTINE
                      TO CONVERT THE INPUT TERMS TO CORRESPOND
                      TO THE SPECIFIED RESULT MUST BE CALLED.   */
IF SOFP & POFS THEN BOTH='1'B;  ELSE BOTH='0'B;
IF ¬BOTH THEN IF SOFP THEN IF CNTMIN=NUMOUTPUTS
```

```
                                    THEN MUSTCONVERT='0'B;
                                    ELSE MATCHTYPE='MIN';
                               ELSE IF CNTMAX=NUMOUTPUTS
                                    THEN MUSTCONVERT='0'B;
                                    ELSE MATCHTYPE='MAX';
            ELSE DO;
              IF CNTMIN=NUMOUTPUTS THEN DO;
                MUSTCONVERT='0'B;  POFS='0'B;  END;
                ELSE IF CNTMAX=NUMOUTPUTS THEN DO;
                  MUSTCONVERT='0'B;  SOFP='0'B;  END;
                  ELSE IF CNTMIN >= CNTMAX THEN DO;
                    POFS='0'B;  MATCHTYPE='MIN';  END;
                    ELSE DO;
                      SOFP='0'B;  MATCHTYPE='MAX';  END;
            END;
CONVERSION:
    CNTSOL=CNTSOL+1;
    IF MUSTCONVERT THEN CALL CONVERT (TYPETERM1,COUNT1,TYPETERM2,
    COUNT2,BOTH,MATCHTYPE);
    ALLOCATE SOURCETAB SET (PS);
    ALLOCATE REDUCEDTAB SET (PR);
    CALL MERGE (MERGENUM);
    ESSENROW=0;
    MAXNUMPI=0;
    MAXROWS=0;

CALL_AGAIN:
    REDUCEDTAB=0;

    CALL GENERPI (MERGENUM,IROW,LUCKY,MAXNUMPI,MAXROWS);
/*  IF LUCKY RETRUNS TRUE, THEN SWITCH THE POINTERS TO TABLES
    SOURCETAB AND REDUCEDTAB AND CALL SUBROUTINE GENERPI AGAIN      */
    IF LUCKY THEN DO;
      T=PS;
      PS=PR;
      PR=T;
```

```
      MERGENUM=IROW;
      GO TO CALL_AGAIN;
      END;
    FREE SOURCETAB;
    FREE REDUCEDTAB;
    CALL COVERIT (NUMCOVER,CRCOST,ANDCOST,INPUTCOST,INTERCOST);
    CALL REDUCE (INTERCOST,LISTALL,SOLVED);
    IF ¬SOLVED THEN CALL CYCLIC (INTERCOST);

SOLUTION:
/*  PRINT THE SOLUTION TO THE PROBLEM  */
    IF LINENO(SYSPRINT)+5+(2*NUMOUTPUTS) > 56 THEN
    SIGNAL ENDPAGE (SYSPRINT);
    PUT FILE (SYSPRINT) EDIT ('SOLUTION') (SKIP(5),X(20),A);
    MAXK=0;  /*  USED IN DETERMINING LONGEST PRIME IMPLICANT          */
    DO I=1 TO NUMOUTPUTS;
      ICOUNT=0; /*  USED TO SUPPRESS INITIAL * OR + IN SOLUTION      */
      NUMCHAR=0;  /*  USE NUMCHAR TO COUNT COLUMNS ACROSS THE PAGE   */
      PUT FILE (SYSPRINT) EDIT ('FUNCTION ',I,':',BLANK) (SKIP(2),
      A,F(3),A,X(1),A);
      NUMCHAR=NUMCHAR+15;
      DO J=1 TO ESSENROW;
        IF SUBSTR(MEMBER(J),I,1) THEN DO;
          ICOUNT=ICOUNT+1;
          IF POFS THEN DO;
            DO K=1 TO SIZE;
              IF PIPOFS(J,K)=' ' THEN GO TO ENDK1;
              END;  /*  END LOOP K  */
ENDK1:
            K=K-1;
            IF K > MAXK THEN MAXK = K;
            NUMCHAR=NUMCHAR+K+5; /* 5 SPACES INCLUDE SIGN AND PARENS */
            IF NUMCHAR>132 THEN DO;  /* GO TO NEXT LINE              */
              PUT FILE (SYSPRINT) EDIT (BLANK) (SKIP,X(14),A);
              NUMCHAR=15;
              END;
```

```
            IF ICOUNT ¬= 1 THEN PUT FILE (SYSPRINT) EDIT (' * ') (A);
            PUT FILE (SYSPRINT) EDIT ('(') (A);
            PUT FILE (SYSPRINT) EDIT ((PIPOFS(J,L) DO L=1 TO K)) (A);
            PUT FILE (SYSPRINT) EDIT (')') (A);
            END;
          ELSE DO;
            DO K=1 TO SIZE;
              IF PISOFP(J,K)=' ' THEN GO TO ENDK2;
              END;  /*  END LOOP K  */
ENDK2:
            K=K-1;
            IF K > MAXK THEN MAXK = K;
            NUMCHAR=NUMCHAR+K+5;
            IF NUMCHAR > 132 THEN DO;  /*  GO TO NEXT LINE          */
              PUT FILE (SYSPRINT) EDIT (BLANK) (SKIP,X(14),A);
              NUMCHAR=15;
              END;
          IF ICOUNT ¬= 1 THEN PUT FILE (SYSPRINT) EDIT (' + ') (A);
            PUT FILE (SYSPRINT) EDIT ('(') (A);
            PUT FILE (SYSPRINT) EDIT ((PISOFP(J,L) DO L=1 TO K)) (A);
            PUT FILE (SYSPRINT) EDIT (')') (A);
            END;
          END;
        END;  /*  END LOOP J  */
      END;  /*  END LOOP I  */
/*  NOW USE NUMCHAR IN DETERMINING IF THERE IS AT LEAST ONE PRIME
    IMPLICANT THAT IS USED IN MORE THAT 1 FUNCTION.                 */
    DO I=1 TO ESSENROW;
      NUMCHAR=0;
      DO J=1 TO NUMOUTPUTS;
        IF SUBSTR(MEMBER(I),J,1) THEN NUMCHAR=NUMCHAR+1;
        IF NUMCHAR=2 THEN GO TO PRINTMORE;
        END;  /*  END LOOP J  */
      END;  /*  END LOOP I  */
    GO TO SEE_IF_ALL;
/*  IF FALL THROUGH THIS LOOP, NO PRIME IMPLICANT IS USED IN MORE
```

```
    THAN 1 OUTPUT FUNCTION, SO GO TO SEE_IF_ALL;                    */

PRINTMORE:
/*  PRINT PRIME IMPLICANTS USED IN MORE THAN ONE FUNCTION.          */
    PUT FILE (SYSPRINT) SKIP(3);
    MAXK=MAXK+1;
    DO I=1 TO ESSENROW;
      CHECKVAL=MEMBER(I);
      IF NUMONES(CHECKVAL)=0 THEN GO TO INCI;
      IF NUMONES(CHECKVAL)=1 THEN GO TO INCI;
      /*  USED IN MORE THAN ONE FUNCTION                            */
      PUT FILE (SYSPRINT) SKIP;
      IF POFS THEN DO;
        DO K=1 TO SIZE;
          IF PIPOFS(I,K)=' ' THEN GO TO ENDK3;
          END;  /*  END LOOP K  */
ENDK3:
        K=K-1;
        PUT FILE (SYSPRINT) EDIT ((PIPOFS(I,L) DO L=1 TO K)) (A);
        END;
      ELSE DO;
        DO K=1 TO SIZE;
          IF PISOFP(I,K)=' ' THEN GO TO ENDK4;
          END;  /*  END LOOP K  */
ENDK4:
        K=K-1;
        PUT FILE (SYSPRINT) EDIT ((PISOFP(I,L) DO L=1 TO K)) (A);
        END;
    PUT FILE (SYSPRINT) EDIT (' IS USED IN FUNCTIONS ') (X(MAXK-K),A);
      DO J=1 TO NUMOUTPUTS;
        IF SUBSTR(MEMBER(I),J,1) THEN PUT FILE (SYSPRINT) EDIT (J)
        (X(1),F(3));
        END;  /*  END LOOP J  */
INCI: END;  /*  END LOOP I  */

SEE_IF_ALL:
```

```
      IF BOTH THEN DO;
        IF CNTSOL=2 THEN GO TO FINIS;
        IF POFS THEN DO;
          SOFP='1'B;  POFS='0'B;
          MUSTCONVERT='1'B;  MATCHTYPE='MIN';
          END;
        ELSE DO;
          POFS='1'B;  SOFP='0'B;
          MUSTCONVERT='1'B;  MATCHTYPE='MAX';
          END;
        GO TO CONVERSION;
        END;
FINIS:
      FREE MASTER;
      FREE TABLIST;
      FREE REFCOL;
```

```
GENERPI: PROC (MERGENUM,IROW,LUCKY,MAXNUMPI,MAXROWS);

/*  THIS SUBROUTINE GENERATES ALL THE PRIME IMPLICANTS.              */

DECLARE
CURRENTBASE                    BINARY FIXED (15),
CURRENTDIF                     BINARY FIXED (15),
CURRENTNUMIS                   BINARY FIXED (15),
CURRENTTAG                     BIT (10) ALIGNED,
EXCLOR                         BIT (4) INITIAL ('0110'B) STATIC,
HOLDTAG                        BIT (10) ALIGNED,
IROW                           BINARY FIXED (15),
LUCKY                          BIT (1),
MAXNUMPI                       BINARY FIXED (15),
MAXROWS                        BINARY FIXED (15),
MAYBEPI                        BIT (15),
MERGENUM                       BINARY FIXED (15),
NEWDIFF                        BINARY FIXED (15);

    LUCKY='1'B;
    IROW=0;
    IF MERGENUM > MAXROWS THEN MAXROWS=MERGENUM;
    DO I=1 TO MERGENUM-1;
      CURRENTBASE=SBASENUM(I);  CURRENTNUMIS=SNUMBERONES(I);
      CURRENTDIF=STOTDIFF(I);   CURRENTTAG=TAG(I);
/*  FIRST SEE IF THE BASE VALUE IS IN THE GROUP WITH ONE LESS 'ONE'
    THAN THE GROUP BEING CHECKED                                     */
      DO J=I+1 TO MERGENUM;
        IF CURRENTNUMIS<(SNUMBERONES(J)-1) THEN GO TO ENDI;
        IF CURRENTNUMIS¬=(SNUMBERONES(J)-1) THEN GO TO ENDJ;
        IF CURRENTDIF¬=STOTDIFF(J) THEN GO TO ENDJ;
        IF (CURRENTTAG & TAG(J)) THEN DO;
          IF CURRENTBASE>SBASENUM(J) THEN GO TO ENDJ;
          MAYBEPI=BOOL(CURRENTBASE,SBASENUM(J),EXCLOR);
          IF NUMONES(MAYBEPI)=1 THEN DO;
/*  HAVE A PRIME IMPLICANT - FIRST SEE IF IT
```

```
    HAS ALREADY BEEN GENERATED                                          */
          HOLDTAG=CURRENTTAG & TAG(J);
          NEWDIFF=CURRENTDIF+SBASENUM(J)-CURRENTBASE;
          DO K=1 TO IROW;
            IF RTAG(K)=HOLDTAG THEN IF RBASENUM(K)=CURRENTBASE THEN
            IF RTOTDIFF(K)=NEWDIFF THEN GO TO TAG_IT;
            END;  /*  END LOOP K  */
            IROW=IROW+1;
    IF IROW>500 THEN DO;
      PUT FILE (SYSPRINT) EDIT ('MORE THAN 500 IMPLICANTS HAVE ',
      'BEEN GENERATED AT ONE TIME - THE PROGRAM CANNOT HANDLE THAT ',
      'MANY.') (SKIP,A,A,A);
      STOP;
      END;
            /*  STORE THE NEW PRIME IMPLICANT                           */
            RBASENUM(IROW)=CURRENTBASE;
            RNUMBERONES(IROW)=CURRENTNUMIS;
            RTOTDIFF(IROW)=NEWDIFF;  RTAG(IROW)=HOLDTAG;
TAG_IT:
            /*  LOOK AT THE TAGS TO DETERMINE IF THE PRIME
                IMPLICANT CAN BE CHECKED OFF                            */
            IF CURRENTTAG=HOLDTAG THEN ESSENSW(I)='1'B;
            IF TAG(J)=HOLDTAG THEN ESSENSW(J)='1'B;
            END;
            END;
ENDJ:      END;  /*  END LOOP J */
ENDI:    END;   /*  END LOOP I  */
    MAXNUMPI=MAXNUMPI+MERGENUM;
    DO K=1 TO MERGENUM;
      IF ESSENSW(K) THEN GO TO ENDK;
      ESSENROW=ESSENROW+1;
          IF ESSENROW > 500 THEN DO;
            PUT FILE (SYSPRINT) EDIT ('A TOTAL OF MORE THAN 500 ',
            'PRIME IMPLICANTS HAS BEEN GENERATED FOR THIS PROBLEM. ',
            ' EXECUTION WILL TERMINATE DUE TO LIMITATIONS OF THE ',
            'PROGRAM.') (SKIP(2),A,A,SKIP,A,A);
```

```
            STOP;  END;
        BASE(ESSENROW)=SBASENUM(K);
        DIFFER(ESSENROW)=STOTDIFF(K);
        OUTPUTTAG(ESSENROW)=TAG(K);
ENDK: END;  /*  END LOOP K  */
      IF IROW=0 THEN DO;
        LUCKY='0'B;
        PUT FILE (SYSPRINT) EDIT ('A TOTAL OF ',MAXNUMPI,' IMPLICANTS ',
        'HAVE BEEN GENERATED FOR THIS PROBLEM.  ',ESSENROW,
        ' OF THESE ARE PRIME IMPLICANTS.') (SKIP(3),A,F(4),A,A,F(4),A);
      PUT FILE (SYSPRINT) EDIT ('THE LARGEST NUMBER OF IMPLICANTS ',
      'GENERATED AT ONE TIME IS ',MAXROWS,'.  THIS NUMBER SHOULD ',
      'NEVER BE GREATER THAN 500.') (SKIP,A,A,F(3),A,A);
        END;
END GENERPI;
```

```
REDUCE: PROC (INTERCOST,LISTALL,SOLVED);

/*  THIS SUBROUTINE ATTEMPTS TO REDUCE THE PRIME IMPLICANT TABLE BY
    ELIMINATING ESSENTIAL PRIME IMPLICANTS, DOMINANT COLUMNS, AND
    DOMINATED ROWS.                                                  */

DECLARE
CHANGE                          BIT (1),
DOMCOL                          BINARY FIXED (15),
INTERCOST                       BINARY FIXED (31,12),
(JDOM,KDOM)                     BIT (1),
(JNOTDOM,KNOTDOM)               BIT (1),
LISTALL                         BIT (1),
ONLYONE                         BIT (1),
PREVCOUNT                       BIT (1),
(ROWNATED,ROWNANT)              BINARY FIXED (15),
SAVEJ                           BINARY FIXED (15),
SOLVED                          BIT (1),
VALUE                           BINARY FIXED (15);

TRYESSENTIAL:
/*  FIRST DETECT ALL ESSENTIAL PRIME IMPLICANTS, AND ELIMINATE
    ALL COLUMNS COVERED BY THESE ESSENTIAL PRIME IMPLICANTS.         */
    CHANGE='0'B;  /*  THIS SWITCH WILL BE TURNED ON IF A CHANGE IS
    MADE TO THE PRIME IMPLICANT TABLE, INDICATING TO REPEAT THE
    STEPS FOR REDUCTION OF THE TABLE.                                */
    DO I=1 TO WIDTH;
        IF PITABLE(0,I) THEN GO TO NEXTCOL;
        ICOUNT=0;
        PREVCOUNT='0'B;
        DO J=1 TO ESSENROW;
            IF PITABLE(J,I) THEN ICOUNT=ICOUNT+1;
            /*  ICOUNT IS NO. OF ONES IN THAT COLUMN  */
            IF ¬PREVCOUNT & ICOUNT=1 THEN DO;
                SAVEJ=J;
                PREVCOUNT='1'B; /* PREVCOUNT IS TURNED ON AFTER THE
```

```
                                        FIRST ONE BIT HAS BEEN ENCOUNTERED  */
                    END;
                END;    /*  END LOOP J  */
            IF ICOUNT ¬= 1 THEN GO TO NEXTCOL;
/*  YEA - THIS PRIME IMPLICANT IS ESSENTIAL.
    INDICATE WHICH FUNCTION IT IS ESSENTIAL TO  */
            SUBSTR(MEMBER(SAVEJ),COLREF(I),1)='1'B;
            IF LISTALL THEN PUT FILE (SYSPRINT) EDIT ('PRIME IMPLICANT ',
            SAVEJ,' IS ESSENTIAL TO FUNCTION ',COLREF(I)) (SKIP,A,F(3),
            A,F(3));
/*  CHANGE THE COST TABLE FOR THAT ESSENTIAL PRIME IMPLICANT        */
            GATECOST(SAVEJ)=0;   INPUT(SAVEJ)=0;
            INTER(SAVEJ)=INTER(SAVEJ)-1;
            COST(SAVEJ)=INTERCOST*INTER(SAVEJ);
/*  ELIMINATE OTHER TERMS THAT THE PRIME IMPLICANT COVERS           */
            VALUE=COLREF(I);
            DO K=FIRSTCOL(VALUE) TO LASTCOL(VALUE);
                IF PITABLE(0,K) THEN GO TO STOPK;
                IF PITABLE(SAVEJ,K) THEN DO;
                    PITABLE(*,K)='0'B;  PITABLE(0,K)='1'B;
                    END;
STOPK:
                END;  /*  END LOOP K  */
            PITABLE(0,I)='1'B;  /*  INDICATE THAT COLUMN HAS BEEN ELIMIN.*/
            CHANGE='1'B;  /*  INDICATE THAT SOME CHANGE WAS MADE
                              TO THE PRIME IMPLICANT TABLE              */
            GO TO NEXTCOL;
NEXTCOL:
        END;  /*  END LOOP I  */
/*  SEE IF THERE IS ANY TERM (COLUMN OF PRIME IMPLICANT TABLE) THAT
        IS NOT COVERED.  THIS IS INDICATED BY A ZERO IN THE TOP
        POSITION OF THAT COLUMN.                                    */
        DO I=1 TO WIDTH;
            IF ¬PITABLE(0,I) THEN GO TO TRYCOL;
            END;  /*  END LOOP I  */
        SOLVED='1'B;
```

```
    RETURN;

/*  ELIMINATE ALL DOMINANT COLUMNS  */
TRYCOL:
    DO I=1 TO NUMOUTPUTS;
      DO J=FIRSTCOL(I) TO LASTCOL(I)-1;
        IF  PITABLE(0,J) THEN GO TO NEXTJ; /* COL. ALREADY ELIMINATED*/
        DO K=J+1 TO LASTCOL(I);
          IF PITABLE(0,K) THEN GO TO NEXTK; /* COL. ALREADY ELIM.    */
          JDOM='0'B;  KDOM='0'B;
          JNOTDOM='0'B;  KNOTDOM='0'B;
/*  JNOTDOM AND KNOTDOM ARE TURNED ON AS SOON AS A COLUMN CANNOT
    BE DOMINANT - SO NO NEED TO CHECK ANY MORE OF THAT COLUMN.      */
          DO L=1 TO ESSENROW;
            IF KNOTDOM THEN GO TO CHECKJ;
            /* IF WHENEVER THERE IS A 1 IN COLUMN J, THERE IS ALSO
               A 1 IN COLUMN K, THEN COLUMN K DOMINATES COLUMN J    */
            IF PITABLE(L,J) THEN IF PITABLE(L,K) THEN KDOM='1'B;
            ELSE KNOTDOM='1'B;  ELSE;
CHECKJ:
            IF JNOTDOM THEN GO TO CHECKJK;
            /* IF WHENEVER THERE IS A 1 IN COLUMN K, THERE IS ALSO
               A 1 IN COLUMN J, THEN COLUMN J DOMINATES COLUMN K    */
            IF PITABLE(L,K) THEN IF PITABLE(L,J) THEN JDOM='1'B;
            ELSE JNOTDOM='1'B;  ELSE;
CHECKJK:
/*  KNOW THAT NEITHER CAN BE DOMINANT SO JUMP OUT OF LOOP           */
            IF JNOTDOM & KNOTDOM THEN GO TO NEXTK;
            END;  /*  END LOOP L  */
/*  CHECK FOR THE CASE THAT ONE OR BOTH COLUMNS CONTAINED ALL ZEROES */
          IF JNOTDOM THEN JDOM='0'B;
          IF KNOTDOM THEN KDOM='0'B;
          IF ¬JDOM & ¬KDOM THEN GO TO NEXTK;
          CHANGE='1'B;  /*  INDICATE A CHANGE IN PITABLE.           */
          IF KDOM & JDOM THEN DO;
            IF LISTALL THEN PUT FILE (SYSPRINT) EDIT ('CCLUMN ',K,
```

```
            ' (TERM ',TOPROW(K),') WHICH EQUALS COLUMN ',J,
            ' (TERM ',TOPROW(J),') WILL BE REMOVED FROM FUNCTION ',
            I) (SKIP,2 (A,F(3),A,F(5)),A,F(3));
            DOMCOL=K;
            GO TO ELIMCOL;
            END;
          IF KDOM THEN DO;
            DOMCOL=K;
            IF LISTALL THEN PUT FILE (SYSPRINT) EDIT ('COLUMN ',K,
            ' (TERM ',TOPROW(K),') WHICH DOMINATES COLUMN ',J,
            ' (TERM ',TOPROW(J),') WILL BE REMOVED FROM FUNCTION ',
            I) (SKIP,2 (A,F(3),A,F(5)),A,F(3));
            END;
          ELSE DO;
            DOMCOL=J;
            IF LISTALL THEN PUT FILE (SYSPRINT) EDIT ('COLUMN ',J,
            ' (TERM ',TOPROW(J),') WHICH DOMINATES COLUMN ',K,
            ' (TERM ',TOPROW(K),') WILL BE REMOVED FROM FUNCTION ',
            I) (SKIP,2 (A,F(3),A,F(5)),A,F(3));
            END;
/*  AT THIS POINT, MUST DETERMINE HOW TO REDUCE THE INTERCONNECTION
    COST BASED ON THE NUMBER OF MINTERMS COVERED BY EACH PRIME
    IMPLICANT CORRESPONDING TO AN X IN THAT DOMINANT COLUMN THAT
    WILL BE ELIMINATED.  ASSUME THERE IS ONLY ONE X IN THAT ROW FOR
    EACH FUNCTION. - TURN SWITCH ON.  TURN SWITCH OFF AS SOON AS
    ANOTHER X IS FOUND AND JUMP OUT OF LOOP.  ONLY NEED TO REDUCE
    INTERCONNECTION COST IF THERE IS ONLY ONE X WITHIN THAT FUNCTION
    FOR THAT PRIME IMPLICANT.  THE INTERCONNECTION COST MUST BE
    REDUCED BY 1 SINCE, BY ELIMINATING THE DOMINANT COLUMN, ALL
    COLUMNS COVERED BY THAT PRIME IMPLICANT WERE REMOVED FROM THE
    FUNCTION.  THEREFORE, THERE IS ONE LESS FUNCTION THAT THE
    PRIME IMPLICANT CAN BE USED IN - HENCE, ITS INTERCONNECTION COST
    CAN BE REDUCED BY 1.                                          */
          DO L=1 TO ESSENROW;
            ONLYONE='1'B;
            IF PITABLE(L,DOMCOL) THEN IF INTER(L) <= 1 THEN GO TO
```

```
            NEXTL;  ELSE DO M=FIRSTCOL(I) TO LASTCOL(I);
              IF PITABLE(0,M) THEN GO TO NEXTM; /* COL. ALREADY EL. */
              IF M=DOMCOL THEN GO TO NEXTM; /* DON'T CONSIDER X IF
                                               IT IS IN DOM. COLUMN     */
              IF ¬PITABLE(L,M) THEN GO TO NEXTM; /* IS IT THE ONLY
                                    ONE IN THAT ROW FOR THAT FUNCTION?  */
              ONLYONE='0'B;  GO TO EARLYOUT;
NEXTM:        END;  /*  END LOOP M  */
            ELSE GO TO NEXTL;
EARLYOUT:
            IF ONLYONE THEN IF INTER(L) > 1 THEN DO;
              INTER(L)=INTER(L)-1;
              COST(L)=COST(L)-INTERCOST;
              END;
NEXTL:      END;  /*  END LOOP L  */
ELIMCOL:
/*  NOW ELIMINATE THE COLUMN BY TURNING OFF ALL ONE BITS IN IT     */
          PITABLE(*,DOMCOL)='0'B;
          PITABLE(0,DOMCOL)='1'B;
          CHANGE='1'B;
NEXTK:
          END;  /*  END LOOP K  */
NEXTJ:
        END;  /*  END LOOP J  */
NEXTI:
      END;  /*  END LOOP I  */
/*  SEE IF THERE IS ANY TERM (COLUMN OF PRIME IMPLICANT TABLE) THAT
    IS NOT COVERED.  THIS IS INDICATED BY A ZERO IN THE TOP
    POSITION OF THAT COLUMN.                                       */
    DO I=1 TO WIDTH;
      IF ¬PITABLE(0,I) THEN GO TO TRYROW;
      END;  /*  END LOOP I  */
    SOLVED='1'B;
    RETURN;

/*  NOW TRY TO ELIMINATE DOMINATED ROWS FROM THE TABLE             */
```

```
TRYROW:
    DO J=1 TO ESSENROW-1;
      IF PITABLE(J,0) THEN GO TO EXITJ;
      DO K=J+1 TO ESSENROW;
        IF PITABLE(K,0) THEN GO TO EXITK;
        JDOM='0'B;  KDOM='0'B;
        JNOTDOM='0'B;  KNOTDOM='0'B;
/*  JNOTDOM AND KNOTDOM ARE TURNED ON AS SOON AS A ROW CANNOT BE
    DOMINATED - SO NO NEED TO CHECK ANY MORE OF THAT ROW.          */
        DO L=1 TO WIDTH;
          IF JNOTDOM THEN GO TO LOOKATJ;
/*  IF WHENEVER THERE IS A ONE IN ROW J, THERE IS ALSO A ONE IN
    ROW K, THEN ROW K DOMINATES ROW J AND ROW J CAN BE REMOVED.    */
          IF PITABLE(J,L) THEN IF PITABLE(K,L) THEN JDOM='1'B;
          ELSE JNOTDOM='1'B;  ELSE;
LOOKATJ:
          IF KNOTDOM THEN GO TO LOOKATJK;
/*  IF WHENEVER THERE IS A ONE IN ROW K, THERE IS ALSO A ONE IN
    ROW J, THEN ROW J DOMINATES ROW K AND ROW K CAN BE REMOVED.    */
          IF PITABLE(K,L) THEN IF PITABLE(J,L) THEN KDOM='1'B;
          ELSE KNOTDOM='1'B;  ELSE;
LOOKATJK:
          IF JNOTDOM & KNOTDOM THEN GO TO EXITK;
          END;  /*  END LOOP L  */
        IF JNOTDOM THEN JDOM='0'B;
        IF KNOTDOM THEN KDOM='0'B;
        IF ¬JDOM & ¬KDOM THEN GO TO EXITK;
        IF KDOM & JDOM THEN DO;
/*      ROWS ARE EQUAL - ELIMINATE ONE OF HIGHER COST    */
          IF COST(J) >= COST(K) THEN DO;
            ROWNATED=J;  ROWNANT=K;  GO TO ELIMROW;
            END;
          ROWNATED=K;  ROWNANT=J;
          GO TO ELIMROW;
          END;
        IF KDOM THEN DO;
```

```
          ROWNATED=K;  ROWNANT=J;
          END;
        ELSE DO;
          ROWNATED=J;  ROWNANT=K;
          END;

CHECKCOST:
/*  MUST EXAMINE THE COST OF THE TWO ROWS IN ORDER TO
    DECIDE IF THE DOMINATED ROW CAN BE REMOVED.                    */
        IF COST(ROWNATED) < COST(ROWNANT) THEN GO TO EXITK;
ELIMROW:
/*  ELIMINATE THE DOMINATED ROW BECASUE IT COSTS AT LEAST
    AS MUCH AS THE ROW THAT IT IS DOMINATED BY.                    */
        PITABLE(ROWNATED,*)='0'B;
        COST(ROWNATED)=0.0;
        IF LISTALL THEN PUT FILE (SYSPRINT) EDIT ('ROW ',ROWNATED,
        ', DOMINATED BY OR EQUAL TO ROW ',ROWNANT,', IS ELIMINATED')
        (SKIP,A,F(2),A,F(2),A);
        PITABLE(ROWNATED,0)='1'B;
        CHANGE='1'B;
EXITK:
        END;  /*  END LOOP K  */
EXITJ:
      END;  /*  END LOOP J  */
/*  NOW SEE IF EVERYTHING IS COVERED - IF SO, THEN A SOLUTION HAS
    BEEN OBTAINED.  IF NOT AND CHANGE SWITCH IS ON, THEN APPLY THE
    STEPS TO REDUCE THE TABLE AGAIN;  OTHERWISE, THE TABLE IS
    CYCLIC AND A P-FUNCTION MUST BE USED TO RESOLVE IT.            */
    DO I=1 TO WIDTH;
      IF PITABLE(0,I) THEN GO TO IAGAIN;
      /*  SOME TERM NOT COVERED - REPEAT THE REDUCTION STEPS.      */
      IF CHANGE THEN GO TO TRYESSENTIAL;
      RETURN;  /* HAVE TO USE P-FUNCTION - RETURN SOLVED OFF  */
IAGAIN:
      END;  /*  END LOOP I  */
    SOLVED='1'B;
```

```
    END REDUCE;
END THESIS;
```

```
PREPRO: PROC (TYPETERM1,TYPETERM2,COUNT1,COUNT2,CRCOST,ANDCOST);

/*  THIS SUBROUTINE PROCESSES AND VERIFIES THE INPUT TERMS          */

DECLARE
01  MASTER                     CONTROLLED EXTERNAL,
    03  TABLE (SIZEA,SIZEB),
        05  TERM               BINARY FIXED (15),
    03  CNTTABLE (SIZEB),
        05  NUMTERMS           BINARY FIXED (15),
    03  BINARRAY (0:NUMBINTAB),
        05  NUMONES            BINARY FIXED (15),
01  REFCOL (NUMOUTPUTS)        CONTROLLED EXTERNAL,
    03  FIRSTCOL               BINARY FIXED (15),
    03  LASTCOL                BINARY FIXED (15),
01  REFER                      STATIC EXTERNAL,
    03  NUMVAR                 BINARY FIXED (15),
    03  NUMOUTPUTS             BINARY FIXED (15),
    03  NUMBINTAB              BINARY FIXED (15),
    03  TOPOFPAGE              CHAR (79),
    03  SIZEA                  BINARY FIXED (15),
    03  SIZEB                  BINARY FIXED (15),
    03  ESSENROW               BINARY FIXED (15),
    03  WIDTH                  BINARY FIXED (15),
    03  SIZE                   BINARY FIXED (15),
    03  POFS                   BIT (1),
    03  SOFP                   BIT (1),
ANDCOST                        BINARY FIXED (31,12),
COUNT                          BINARY FIXED (15),
COUNT1                         BINARY FIXED (15),
COUNT2                         BINARY FIXED (15),
ENDVAL                         BINARY FIXED (15),
ERRORLINE (80)                 CHAR (1),
EQTERM                         BIT (1),
HOLDTERM                       BINARY FIXED (15),
HOLDVAL                        BINARY FIXED (15),
```

```
(I,II,J,K,KK,L,LL,NN)          BINARY FIXED (15),
IMAGE                          CHAR (80),
(IROW,ICOL)                    BINARY FIXED (15),
ITYPE                          CHAR (3),
MAXVAL                         BINARY FIXED (15),
MESSAGE1                       CHAR (46),
MESSAGE2                       CHAR (77),
NONALPHA                       CHAR (107),
ORCOST                         BINARY FIXED (31,12),
STARTVAL                       BINARY FIXED (15),
TEMP                           BINARY FIXED (15),
TYPETERM1 (1:10)               CHAR (3),
TYPETERM2 (1:10)               CHAR (3),
VALUE                          BIT (15);


/*  INITIALIZE THE ERROR MESSAGES  */
NONALPHA='THE DATA CARD THAT SPECIFIES WHAT FORM THE ANSWER WILL BE IN
CONTAINS A NON-ALPHABETIC CHARACTER IN COLUMN ';
MESSAGE1='DATA CARD CONTAINS TWO HYPHENS BETWEEN DIGITS.';
MESSAGE2='A PORTION OF THAT CARD IS PRINTED STARTING WITH THE INVALID H
YPHEN IN COL. 1.';

/*  GENERATE THE BINARY TABLE TO BE USED IN SORTING THE INPUT TERMS
    BY THE NUMBER OF ONES CONTAINED IN THE BINARY REPRESENTATION OF
    THE DECIMAL VALUE OF THE TERM.  THIS MUST BE DONE IN ORDER TO
    GENERATE THE PRIME IMPLICANTS BY THE QUINE-MC CLUSKEY METHOD.   */

    IF NUMVAR > NUMOUTPUTS THEN HOLDVAL=NUMVAR;
                           ELSE HOLDVAL=NUMOUTPUTS;
    STARTVAL=17-HOLDVAL;
    DO I=0 TO NUMBINTAB;
      VALUE=SUBSTR(UNSPEC(I),STARTVAL,HOLDVAL);
      COUNT=0;
      DO J=1 TO HOLDVAL;
        IF SUBSTR(VALUE,J,1) THEN COUNT=COUNT+1;  ELSE;
```

```
    END;  /*  END LOOP J  */
  NUMONES(I)=COUNT;
  END;  /*  END LOOP I  */
GET FILE (SYSIN) EDIT (IMAGE) (COLUMN(1),A(80));
J=VERIFY(IMAGE,'ABCDEFGHIJKLMNOPQRSTUVWXYZ ');
IF J>0 THEN DO;
  PUT FILE (SYSPRINT) EDIT ('NONALPHA',J) (SKIP(3),A,F(3));
  GO TO BAD_DATA;  END;
J=INDEX(IMAGE,'PRODUCT OF SUMS');
IF J>0 THEN POFS='1'B;
J=INDEX(IMAGE,'SUM OF PRODUCTS');
IF J>0 THEN SOFP='1'B;
PUT FILE (SYSPRINT) SKIP(2);
IF POFS & SOFP THEN PUT FILE (SYSPRINT) EDIT ('THE RESULT WILL ',
'BE MINIMIZED IN BOTH THE SUM-OF-PRODUCTS AND PRODUCT-OF-SUMS ',
'FORMS.') (SKIP(2),A,A,A);
IF POFS & ¬ SOFP THEN PUT FILE (SYSPRINT) EDIT ('THE RESULT WILL ',
'BE MINIMIZED IN THE PRODUCT-OF-SUMS FORM.') (SKIP(2),A,A);
IF SOFP & ¬ POFS THEN PUT FILE (SYSPRINT) EDIT ('THE RESULT WILL ',
'BE MINIMIZED IN THE SUM-OF-PRODUCTS FORM.') (SKIP(2),A,A);
IF ¬ POFS & ¬ SOFP THEN DO;
  PUT FILE (SYSPRINT) EDIT ('DATA CARD SPECIFYING FORM OF ',
  'SOLUTION CONTAINS AN INVALID WORD.  CHECK THE DATA CARD WHICH',
  ' IS PRINTED BELOW.') (SKIP(3),A,A,A);
  PUT FILE (SYSPRINT) EDIT (IMAGE) (SKIP(3),A);
  GO TO BAD_DATA;  END;
IF POFS THEN IF ORCOST <= 0 THEN DO;
  PUT FILE (SYSPRINT) EDIT ('THE COST FACTOR FOR OR-GATES IS ',
  'NOT POSITIVE - THIS CONFLICTS WITH THE REQUESTED FORM OF THE ',
  'OUTPUT, WHICH IS PRODUCT-OF-SUMS.') (SKIP(2),A,A,A);
  GO TO BAD_DATA;  END;
IF SOFP THEN IF ANDCOST <= 0 THEN DO;
  PUT FILE (SYSPRINT) EDIT ('THE COST FACTOR FOR AND-GATES IS ',
  'NOT POSITIVE - THIS CONFLICTS WITH THE REQUESTED FORM OF THE ',
  'OUTPUT, WHICH IS SUM-OF-PRODUCTS.') (SKIP(2),A,A,A);
  GO TO BAD_DATA;  END;
```

```
    ON ENDFILE (SYSIN) GO TO ENDOFDATA;
/*  MUST READ 2*NO. OF OUTPUTS PHI AND MAX-MIN CARDS
    COUNTS CONTROL THE SETS OF CARDS READ IN  */
    COUNT1=0;
    COUNT2=0;
    IROW=0;
    ICOL=1;

/*  BEGIN INPUT TERMS  */
    GET FILE (SYSIN) EDIT (IMAGE) (COLUMN(1),A(80));
TYPECARD:
    IF SUBSTR(IMAGE,1,3)='MAX' | SUBSTR(IMAGE,1,3)='MIN' THEN DO;
      COUNT1=COUNT1+1;
      IF COUNT1 > NUMOUTPUTS THEN DO;
        PUT FILE (SYSPRINT) EDIT ('THERE ARE TOO MANY ''MAX'' CARDS',
        ' OR ''MIN'' CARDS.  YOU SPECIFIED ',NUMOUTPUTS,' OUTPUT ',
        'FUNCTIONS,','BUT CARD NUMBER ',COUNT1,' OF THE CARE-TERMS ',
        'WAS JUST READ.') (SKIP(3),A,A,F(3),A,A,SKIP,A,F(3),A,A);
        GO TO END_OF_LINE;
        END;
      TYPETERM1(COUNT1)=SUBSTR(IMAGE,1,3);
      NUMTERMS(ICOL)=IROW;
      ICOL=2*COUNT1-1;
      IROW=0;
      GO TO NEXTCARD;
      END;
    IF SUBSTR(IMAGE,1,3)='PHI' THEN DO;
      COUNT2=COUNT2+1;
      IF COUNT2 > NUMOUTPUTS THEN DO;
        PUT FILE (SYSPRINT) EDIT ('THERE ARE TOO MANY ''PHI'' CARDS',
        '.  YOU SPECIFIED ',NUMOUTPUTS,' OUTPUT FUNCTIONS, BUT CARD ',
        'NUMBER ',COUNT1,' OF THE DON''T-CARE TERMS WAS JUST READ.')
        (SKIP(3),A,A,F(3),A,A,F(3),A);
        GO TO END_OF_LINE;
        END;
```

```
      TYPETERM2(COUNT2)=SUBSTR(IMAGE,1,3);
      NUMTERMS(ICOL)=IROW;
      ICOL=2*COUNT2;
      IROW=0;
      GO TO NEXTCARD;
      END;
    PUT FILE (SYSPRINT) EDIT ('EXPECTING KEYWORDS ''MAX'' OR ''MIN''',
    ' OR ''PHI'' - UNABLE TO DECODE WORD ON CARD WHICH IS ''',
    (SUBSTR(IMAGE,1,3)),'''.') (SKIP(3),A,A,A,A);
    GO TO BAD_DATA;

/*  GET AN INPUT M-TERM OR PHI-TERM CARD  */
NEXTCARD:
    GET FILE (SYSIN) EDIT (IMAGE) (COLUMN(1),A(80));
    J = VERIFY (IMAGE, '0123456789 -,');
    IF J=1 THEN GO TO TYPECARD;
    IF J>0 THEN DO;
        PUT FILE (SYSPRINT) EDIT (IMAGE) (SKIP(3),A);
        ERRORLINE=' ';
        ERRORLINE(J)='$';
        PUT FILE (SYSPRINT) SKIP;
        PUT FILE (SYSPRINT) EDIT (ERRORLINE) (A);
        PUT FILE (SYSPRINT) EDIT ('INVALID DATA CARD - CHECK POSITION',
        J) (SKIP,A(34),X(2),F(5));
        GO TO BAD_DATA;
        END;

AGAIN:
    K=VERIFY(IMAGE,' ,'); /* POS. OF NEXT DIGIT OR HYPHEN  */
    IF K=0 THEN GO TO NEXTCARD;
    IF SUBSTR(IMAGE,K,1) = '-' THEN GO TO HAVE_HYPHEN;
    IMAGE=SUBSTR(IMAGE,K);  /*  BREAK OFF UP TO THE NEXT DIGIT  */
    K=1;

   /*  L IS THE POSITION OF THE NEXT BLANK OR COMMA  */
    L=VERIFY(IMAGE,'0123456789'); /* FIRST DELIMITER AFTER DIGITS */
```

```
/*  CHECK FOR DUPLICATE TERMS IN THE CARE-TERMS AND IN THE
    DON'T-CARE TERMS OF EACH FUNCTION.  THIS DOES NOT CHECK
    BETWEEN THE CARE-TERMS AND THE DON'T-CARE TERMS.  IF THERE IS
    A DUPLICATE TERM IN THE CARE-TERMS AND THE DON'T-CARE TERMS
    OF A FUNCTION, THE TERM IS TREATED AS A CARE-TERM.              */
    HOLDTERM=SUBSTR(IMAGE,K,L-K);
    DO NN=1 TO IROW;
      IF HOLDTERM=TERM(NN,ICOL) THEN GO TO POSTN;
      END;  /*  END LOOP NN  */
    IROW=IROW+1;
    TERM(IROW,ICOL)=HOLDTERM;
POSTN:
    IMAGE=SUBSTR(IMAGE,L); /* START STRING AFTER LAST DIGIT  */
    GO TO AGAIN;

HAVE_HYPHEN:
    IMAGE=SUBSTR(IMAGE,K+1); /* NEW IMAGE STARTS 1 PAST HYPHEN  */
    KK=VERIFY(IMAGE,' ,');
    IF KK=0 THEN DO;
        ON ENDFILE (SYSIN) BEGIN;
        PUT FILE (SYSPRINT) EDIT ('UNABLE TO FIND NEXT DIGIT',
        ' AFTER HYPHEN - CHECK DATA CARDS FOR POSSIBLE STRAY PUNCH.',
        ' TERMINATE EXECUTION.') (SKIP(3),A,A,A);
        STOP;
        END;

    /*  HAVE HYPHENATED STRING ON TWO CARDS.                        */
    GET FILE (SYSIN) EDIT (IMAGE) (COLUMN(1),A(80));
    J = VERIFY (IMAGE, '0123456789 -,');
    IF J>0 THEN DO;
        PUT FILE (SYSPRINT) EDIT (IMAGE) (SKIP(3),A);
        ERRORLINE=' ';
        ERRORLINE(J)='$';
        PUT FILE (SYSPRINT) SKIP;
        PUT FILE (SYSPRINT) EDIT (ERRORLINE) (A);
        PUT FILE (SYSPRINT) EDIT ('INVALID DATA CARD - CHECK POSITION',
```

```
        J) (SKIP,A(34),X(2),F(5));
        GO TO BAD_DATA;
        END;
      KK=VERIFY(IMAGE,' ,');
      IMAGE=SUBSTR(IMAGE,KK);
      KK=1;
      ON ENDFILE (SYSIN) GO TO ENDOFDATA;
      END;

    /*  KK IS POS. OF FIRST DIGIT AFTER THE HYPHEN  */
    IMAGE=SUBSTR(IMAGE,KK); /* BREAK STRING OFF UP TO THE DIGIT  */
    KK=1;
    IF SUBSTR(IMAGE,KK,1) = '-' THEN DO;
        PUT FILE (SYSPRINT) EDIT (MESSAGE1,MESSAGE2,IMAGE,
        'TERMINATE EXECUTION') (SKIP(3),A,SKIP,A,SKIP,A,SKIP,A);
        GO TO BAD_DATA;  END;

    /*  LL IS POSITION OF FIRST NON-DIGIT AFTER THE HYPHEN          */
    LL = VERIFY(IMAGE,'0123456789');
    IF IROW=0 THEN DO;
        PUT FILE (SYSPRINT) EDIT ('HAVE HYPHEN IN COLUMN 1 OF ',
        'FIRST TERM CARD.  TERMINATE EXECUTION.') (SKIP(3),A,A);
        GO TO BAD_DATA;
        END;
    STARTVAL=TERM(IROW,ICOL)+1;
    ENDVAL=SUBSTR(IMAGE,KK,LL-KK);
    DO II=STARTVAL TO ENDVAL;
            HOLDTERM=II;
            /*  CHECK DUPLICATE TERMS IN THE FUNCTION               */
            DO NN=1 TO IROW;
              IF HOLDTERM=TERM(NN,ICOL) THEN GO TO AFTERN;
              END;  /*  END LOOP NN  */
            IROW=IROW+1;
            TERM(IROW,ICOL)=HOLDTERM;
AFTERN:
            END;  /*  END LOOP II  */
```

```
    IMAGE=SUBSTR(IMAGE,LL); /* BREAK OFF DIGITS  */
    GO TO AGAIN;

BAD_DATA:
    PUT FILE (SYSPRINT) EDIT ('THE PROGRAM WILL TERMINATE ',
    'DUE TO A SEVERE ERROR NOTED PREVIOUSLY.  THE REMAINING DATA ',
    'CARDS WILL BE CHECKED FOR VALID CHARACTERS ONLY.')
    (SKIP(3),A,A,SKIP,A);
    ON ENDFILE (SYSIN) GO TO END_OF_LINE;
LOOP:
    GET FILE (SYSIN) EDIT (IMAGE) (COLUMN(1),A(80));
    J=VERIFY(IMAGE,'0123456789 -,');
    IF J=1 THEN DO;
      IF SUBSTR(IMAGE,1,3)='MAX' THEN GO TO PRINT_IT;
      IF SUBSTR(IMAGE,1,3)='MIN' THEN GO TO PRINT_IT;
      IF SUBSTR(IMAGE,1,3)='PHI' THEN GO TO PRINT_IT;
      PUT FILE (SYSPRINT) EDIT (IMAGE,'  CARD HAS AN UNIDENTIFIABLE',
      ' WORD BEGINNING IN COLUMN 1.') (SKIP(3),A,A,A);
      GO TO LOOP;  END;
    IF J>1 THEN DO;
      PUT FILE (SYSPRINT) EDIT (IMAGE,'  CARD CONTAINS AN INVALID ',
      'CHARACTER IN COLUMN',J) (SKIP,A,A,A,F(3));
      GO TO LOOP;  END;
PRINT_IT:
    PUT FILE (SYSPRINT) EDIT ('VALID CARD: ',IMAGE) (SKIP,A,A);
    GO TO LOOP;
END_OF_LINE:
    SIGNAL ERROR;
    STOP;
ENDOFDATA:
    NUMTERMS(ICOL)=IROW;

/*  MAKE SURE THERE IS AT LEAST ONE CARE-TERM FOR EACH FUNCTION.    */
    DO J=1 TO NUMOUTPUTS*2 BY 2;
      IF NUMTERMS(J)=0 THEN DO;
        PUT FILE (SYSPRINT) EDIT ('THERE ARE NO CARE-TERMS FOR ',
```

```
        'FUNCTION ',J,'.','YOU HAVE OMITTED THE TERMS FOLLOWING ',
        'THE ''MAX'' OR ''MIN'' CARD FOR THIS FUNCTION.',
        'EXECUTION WILL TERMINATE.') (SKIP(3),A,A,F(3),A,
        SKIP,A,A,SKIP,A);
        STOP;  END;
      END;  /*  END LOOP J  */
/*  VERIFY THAT THERE ARE CARE AND DON'T-CARE TERMS FOR EACH
    FUNCTION (EVEN IF THERE ARE NO PHI-TERMS).                    */
    IF COUNT1¬=COUNT2 THEN DO;
      PUT FILE (SYSPRINT) EDIT ('INVALID DATA - DID YOU OMIT A PAIR',
      ' OF MIN-MAX AND PHI CARDS?','REMEMBER - EVEN IF THERE ARE ',
      'NO PHI-TERMS, YOU MUST HAVE A PHI-CARD IN YOUR DATA.')
      (SKIP(3),A,A,SKIP,A,A);
      STOP;  END;
/*  VERIFY THAT EACH TERM IS WITHIN THE PROPER RANGE              */
    MAXVAL=2**NUMVAR-1;
    DO I=1 TO NUMOUTPUTS*2;
      DO J=1 TO NUMTERMS(I);
        IF TERM(J,I)>MAXVAL THEN DO K=1 TO NUMOUTPUTS;
          IF I=2*K THEN DO;
            ITYPE=TYPETERM2(K);
            GO TO TOOBIG;
            END;
          IF I=2*K-1 THEN DO;
            ITYPE=TYPETERM1(K);
            GO TO TOOBIG;
            END;
          END;  /*  END LOOP K  */
        END;  /*  END LOOP J  */
      END;  /*  END LOOP I  */
    GO TO CHECKIT;
TOOBIG:
    PUT FILE (SYSPRINT) EDIT ('A ',ITYPE,'TERM FOR FUNCTION ',K,
    ', WITH VALUE OF',TERM(J,I),', EXCEEDS THE MAXIMUM VALUE OF',
    MAXVAL,'.') (SKIP(3),A,A(3),A,F(3),A,F(5),A,F(5),A);
    PUT FILE (SYSPRINT) EDIT ('EXECUTION WILL TERMINATE.') (SKIP,A);
```

```
    STOP;
CHECKIT:
/*  VERIFY THE NUMBER OF GROUPS OF CARDS -
    SHOULD EQUAL THE NUMBER OF OUTPUT FUNCTIONS                    */
    IF NUMOUTPUTS¬=COUNT1 | NUMOUTPUTS¬=COUNT2 THEN DO;
      PUT FILE (SYSPRINT) EDIT ('YOU DO NOT HAVE THE CORRECT NUMBER ',
      'OF MAX-MIN OR PHI CARDS.') (SKIP,A,A);
      PUT FILE (SYSPRINT) EDIT ('THERE ARE ',COUNT1,' MAX OR MIN ',
      'CARDS AND ',COUNT1,' PHI CARDS.') (SKIP(3),A,F(4),A,A,F(4),A);
      STOP;
      END;
/*  NOW SORT THE INPUT TERMS.  THIS MUST BE DONE SO THAT THE
    PHI-TERMS CAN BE MERGED IN WITH THE CARE-TERMS IN SUB. MERGE   */
    DO I=1 TO NUMOUTPUTS*2;
      DO J=1 TO NUMTERMS(I)-1;
        DO K=J+1 TO NUMTERMS(I);
          IF TERM(J,I) > TERM(K,I) THEN DO;
            TEMP=TERM(J,I);  TERM(J,I)=TERM(K,I);  TERM(K,I)=TEMP;
            END;
          END;  /*  END LOOP K  */
        END;  /*  END LOOP J  */
      END;  /*  END LOOP I  */
/*  NOW SEE IF THERE ARE ANY DUPLICATE CARE AND DON'T-CARE TERMS
    WITHIN ANY OF THE OUTPUT FUNCTIONS.                            */
    EQTERM='0'B;
    PUT FILE (SYSPRINT) SKIP(2);
    DO I=1 TO NUMOUTPUTS*2 BY 2;   /*  I IS COLUMN OF CARE-TERMS   */
      J=I+1;   /*  J IS COLUMN OF DON'T-CARE TERMS                 */
      K=1;  /*  K POINTS TO CARE-TERM ROW                  */
      L=1;  /*  L POINTS TO DON'T-CARE TERM ROW            */
COMPARE:
      IF TERM(K,I) < TERM(L,J) THEN GO TO INCK;
      IF TERM(K,I) > TERM(L,J) THEN GO TO INCL;
      /*  HAVE TWO EQUAL TERMS - PRINT THEM AND SET FLAG           */
      PUT FILE (SYSPRINT) EDIT ('IN FUNCTION ',I,' TERM ',TERM(K,I),
      ' IS BOTH A CARE-TERM AND A DON''T-CARE TERM.  THIS IS NOT ',
```

```
      'ALLOWED.') (SKIP,A,F(3),A,F(4),A,A);
      EQTERM='1'B;
INCL:
      L=L+1;
      IF L>NUMTERMS(J) THEN GO TO INCI;
      GO TO COMPARE;
INCK:
      K=K+1;
      IF K>NUMTERMS(I) THEN GO TO INCI;
      GO TO COMPARE;
INCI:
      END;  /*  END LOOP I  */
    IF EQTERM THEN STOP;
    END PREPRO;
```

```
CONVERT: PROC (TYPETERM1,COUNT1,TYPETERM2,COUNT2,BOTH,MATCHTYPE);

/*  THIS SUBROUTINE CONVERTS THE TERMS TO EITHER MINTERMS OR
    MAXTERMS, DEPENDING ON WHAT FORM THE SOLUTION SHOULD BE IN,
    AND TEMPORARILY STORES THEM IN ARRAY VAL.                    */

DECLARE
01  MASTER                    CONTROLLED EXTERNAL,
    03  TABLE (SIZEA,SIZEB),
        05  TERM              BINARY FIXED (15),
    03  CNTTABLE (SIZEB),
        05  NUMTERMS          BINARY FIXED (15),
    03  BINARRAY (0:NUMBINTAB),
        05  NUMONES           BINARY FIXED (15),
01  REFCOL (NUMOUTPUTS)       CONTROLLED EXTERNAL,
    03  FIRSTCOL              BINARY FIXED (15),
    03  LASTCOL               BINARY FIXED (15),
01  REFER                     STATIC EXTERNAL,
    03  NUMVAR                BINARY FIXED (15),
    03  NUMOUTPUTS            BINARY FIXED (15),
    03  NUMBINTAB             BINARY FIXED (15),
     03 TOPOFPAGE             CHAR (79),
    03  SIZEA                 BINARY FIXED (15),
    03  SIZEB                 BINARY FIXED (15),
    03  ESSENROW              BINARY FIXED (15),
    03  WIDTH                 BINARY FIXED (15),
    03  SIZE                  BINARY FIXED (15),
    03  POFS                  BIT (1),
    03  SOFP                  BIT (1),
BOTH                          BIT (1),
COUNT1                        BINARY FIXED (15),
COUNT2                        BINARY FIXED (15),
ENDVAL                        BINARY FIXED (15),
(I,J,K,M)                     BINARY FIXED (15),
(ICOL,IROW)                   BINARY FIXED (15),
(JLIMIT,LIMITLL,LIMITL)       BINARY FIXED (15),
```

```
(L,LL)                          BINARY FIXED (15),
MATCHTYPE                       CHAR (3),
NEWROW                          BINARY FIXED (15),
PHICOL                          BINARY FIXED (15),
STARTVAL                        BINARY FIXED (15),
TYPETERM1 (1:10)                CHAR (3),
TYPETERM2 (1:10)                CHAR (3),
VAL (SIZEA)                     BINARY FIXED (15) CONTROLLED;

    ALLOCATE VAL;
/*  DETERMINE WHICH FUNCTIONS MUST BE CONVERTED
    BY COMPARING WITH THE VALUE STORED IN MATCHTYPE                    */
    DO I=1 TO COUNT1;
      IF TYPETERM1(I)=MATCHTYPE THEN GO TO STOPI;
      TYPETERM1(I)=MATCHTYPE;
      ICOL=2*I-1;
      JLIMIT=NUMTERMS(ICOL);
      IROW=0;
      VAL=0;
      STARTVAL=0;
      DO J=1 TO JLIMIT;
        ENDVAL=TERM(J,ICOL)-1;
        IF STARTVAL > ENDVAL THEN GO TO INCSTART;
        DO K=STARTVAL TO ENDVAL;
          IROW=IROW+1;
          VAL(IROW)=K;
          END;
INCSTART:
        STARTVAL=ENDVAL+2;
        TERM(J,ICOL)=-99;
        END;  /*  END LOOP J  */
      ENDVAL=2**NUMVAR-1;
      IF STARTVAL>ENDVAL THEN GO TO MATCH;
      ELSE DO K=STARTVAL TO ENDVAL;
          IROW=IROW+1;
          VAL(IROW)=K;
```

```
          END;
MATCH:
/* NOW MATCH TERMS IN VAL ARRAY WITH PHI TERMS,
    ELIMINATE DUPLICATES, AND STORE OTHERS IN TERM.                    */
    NEWROW=0;
    PHICOL=ICOL+1;
    LIMITLL=NUMTERMS(PHICOL);
    LIMITL=IROW;
    LL=1;
    L=1;
    IF LIMITLL=0 THEN GO TO MOVEREST;
NEXTONE:
    IF VAL(L)=TERM(LL,PHICOL) THEN GO TO INCBOTH;
    IF VAL(L)<TERM(LL,PHICOL) THEN DO;
      NEWROW=NEWROW+1;
      TERM(NEWROW,ICOL)=VAL(L);
      L=L+1;
      IF L>LIMITL THEN GO TO ENDI;  ELSE GO TO NEXTONE;
      END;
    ELSE GO TO INCLL;
INCBOTH:
    L=L+1;
    IF L>LIMITL THEN GO TO ENDI;
INCLL:
    LL=LL+1;
    IF LL>LIMITLL THEN GO TO MOVEREST;  ELSE GO TO NEXTONE;
MOVEREST:
    DO M=L TO IROW;
      NEWROW=NEWROW+1;
      TERM(NEWROW,ICOL)=VAL(M);
      END;
ENDI:
    NUMTERMS(ICOL)=NEWROW;
STOPI:
    END;  /*  END LOOP I  */
```

```
/*  PRINT THE NEW TERMS AFTER SOME HAVE BEEN CONVERTED                  */
    IF (LINENO(SYSPRINT)+6+(NUMOUTPUTS*3)) >= 60 THEN SIGNAL ENDPAGE
    (SYSPRINT);  ELSE PUT FILE (SYSPRINT) SKIP(4);
    PUT FILE (SYSPRINT) EDIT ('INPUT TERMS AFTER CONVERSION TO ',
      'MATCH RESULT SPECIFICATIONS') (SKIP(2),A,A);
    DO I=1 TO NUMOUTPUTS;
      J=2*I-1;
      PUT FILE (SYSPRINT) EDIT  (TYPETERM1(I),'TERMS FOR FUNCTION ',
      I,':') (SKIP(2),A(3),A,F(3),A);
      IF NUMTERMS(J)=0 THEN PUT FILE (SYSPRINT) EDIT ('NONE') (X(4),A);
      ELSE PUT FILE (SYSPRINT) EDIT ((TERM(K,J) DO K=1 TO NUMTERMS(J)))
      (20 (X(1),F(4)),SKIP(1),X(26));
      J=J+1;
      PUT FILE (SYSPRINT) EDIT (TYPETERM2(I),'TERMS FOR FUNCTION ',
      I,':') (SKIP,A(3),A,F(3),A);
      IF NUMTERMS(J)=0 THEN PUT FILE (SYSPRINT) EDIT ('NONE') (X(4),A);
      ELSE PUT FILE (SYSPRINT) EDIT ((TERM(K,J) DO K=1 TO NUMTERMS(J)))
      (20 (X(1),F(4)),SKIP(1),X(26));
      END;
    FREE VAL;
    END CONVERT;
```

```
MERGE: PROC (MERGENUM);

/*  THIS SUBROUTINE MERGES ALL THE TERMS FOR EACH FUNCTION IN
    PREPARATION FOR THE GENERATION OF THE PRIME IMPLICANTS          */

DECLARE
01  MASTER                     CONTROLLED EXTERNAL,
    03  TABLE (SIZEA,SIZEB),
        05  TERM               BINARY FIXED (15),
    03  CNTTABLE (SIZEB),
        05  NUMTERMS           BINARY FIXED (15),
    03  BINARRAY (0:NUMBINTAB),
        05  NUMONES            BINARY FIXED (15),
01  REFCOL (NUMOUTPUTS)        CONTROLLED EXTERNAL,
    03  FIRSTCOL               BINARY FIXED (15),
    03  LASTCOL                BINARY FIXED (15),
01  REFER                      STATIC EXTERNAL,
    03  NUMVAR                 BINARY FIXED (15),
    03  NUMOUTPUTS             BINARY FIXED (15),
    03  NUMBINTAB              BINARY FIXED (15),
     03 TOPOFPAGE              CHAR (79),
    03  SIZEA                  BINARY FIXED (15),
    03  SIZEB                  BINARY FIXED (15),
    03  ESSENROW               BINARY FIXED (15),
    03  WIDTH                  BINARY FIXED (15),
    03  SIZE                   BINARY FIXED (15),
     03 POFS                   BIT (1),
     03 SOFP                   BIT (1),
01  SOURCETAB (500) BASED (PS),
    03  SBASENUM               BINARY FIXED (15),
    03  SNUMBERONES            BINARY FIXED (15),
    03  STOTDIFF               BINARY FIXED (15),
    03  TAG                    BIT (10) ALIGNED,
    03  ESSENSW                BIT (1) ALIGNED,
CHECKONES                      BINARY FIXED (15),
FUNC1               BIT (10) INITIAL ('1000000000'B) STATIC ALIGNED,
```

```
FUNCNUM                          BINARY FIXED (15),
(I,J,K,L,M)                      BINARY FIXED (15),
HOLDIT                           BINARY FIXED (15),
HOLDTAG                          BIT (10) ALIGNED,
IROW                             BINARY FIXED (15),
MERGENUM                         BINARY FIXED (15),
MERGETAB (SIZEA)                 BINARY FIXED (15) CONTROLLED,
NUMTIMES                         BINARY FIXED (15),
ONECNT                           BINARY FIXED (15),
PS                               POINTER EXTERNAL,
VALUE1                           BINARY FIXED (15),
VALUE2                           BINARY FIXED (15);

/*  TABLE MERGETAB CONTAINS ALL THE DIFFERENT CARE-TERMS AND
    DON'T-CARE TERMS FOR ALL THE DIFFERENT OUTPUT FUNCTIONS.
    THERE ARE NO DUPLICATE TERMS IN THIS LIST.  THE SIZE OF THE
    TABLE IS 2**NUMVAR, WHICH WOULD ALLOW FOR EVERY POSSIBLE TERM
    FOR NUMVAR INPUT VARIABLES.                                    */
    ALLOCATE MERGETAB;
    NUMTIMES=NUMOUTPUTS*2;
    SOURCETAB=0;
    TAG(*)='0'B;
    ESSENSW(*)='0'B;
    /*  MOVE IN THE MINTERMS OR MAXTERMS FOR THE FIRST FUNCTION.   */
    DO J=1 TO NUMTERMS(1);
      MERGETAB(J)=TERM(J,1);
      TAG(J)=FUNC1;
      END;  /*  END LOOP J  */
    IROW=J-1;
    MERGENUM=IROW;

    /*  NOW MERGE IN THE TERMS FOR THE OTHER FUNCTIONS.  IF J IS
        EVEN, THEN MERGE IN THE DON'T-CARE-TERMS FOR FUNCTION NUMBER
        FUNCNUM.  IF J IS ODD, THEN MERGE IN THE CARE-TERMS FOR THE
        FUNCTION NUMBER FUNCNUM.                                   */
    DO J=2 TO NUMTIMES;
```

```
      FUNCNUM=(J+1)/2;
      DO L=1 TO NUMTERMS(J);
        DO M=1 TO MERGENUM;
          IF MERGETAB(M)=TERM(L,J) THEN DO;
            /*  INDICATE WHICH FUNCTION THE TERM OCCURRED IN        */
            SUBSTR(TAG(M),FUNCNUM,1)='1'B;
            GO TO ENDL;
            END;
          ELSE;
          END;  /*  END LOOP M  */
        /*  FOUND A TERM THAT HAS NOT OCCURRED IN ANY OTHER FUNCTION*/
        IROW=IROW+1;
        MERGETAB(IROW)=TERM(L,J);
        SUBSTR(TAG(IROW),FUNCNUM,1)='1'B;
        MERGENUM=IROW;
ENDL:   END;  /*  END LOOP L   */
ENDJ: END;  /*  END LOOP J  */

MOVEIN:
    DO I=1 TO MERGENUM;
      SBASENUM(I)=MERGETAB(I);
      END;  /*  END LOOP I  */

/* NOW DO A PRIMARY SORT ON THE NUMBER OF 1'S IN THE BINARY
    REPRESENTATION OF THE NUMBER AND A SECONDARY SORT ON THE
    DECIMAL VALUE OF THE NUMBER.                                  */
    DO I=1 TO MERGENUM-1;
      VALUE1=SBASENUM(I);
      ONECNT=NUMONES(VALUE1);
      DO K=I+1 TO MERGENUM;
        VALUE2=SBASENUM(K);
        CHECKONES=NUMONES(VALUE2);
        IF ONECNT<CHECKONES THEN GO TO ENDK;
        IF ONECNT=CHECKONES THEN GO TO CHECKVALUE;
        HOLDIT=SBASENUM(I);
        SBASENUM(I)=SBASENUM(K);
```

```
          SBASENUM(K)=HOLDIT;
          HOLDTAG=TAG(I);
          TAG(I)=TAG(K);
          TAG(K)=HOLDTAG;
          ONECNT=CHECKCNES;
          VALUE1=SBASENUM(I);
          GO TO ENDK;
CHECKVALUE:
          IF VALUE1<VALUE2 THEN GO TO ENDK;
          IF VALUE1=VALUE2 THEN DO;
               PUT FILE (SYSPRINT) EDIT ('DUPLICATE TERMS') (SKIP,A);
               PUT FILE (SYSPRINT) EDIT ('VALUE1=',VALUE1,'VALUE2=',
               VALUE2) (X(10),A(7),F(3),X(3),A(7),F(3));
               STOP;
               END;
          HOLDIT=SBASENUM(I);
          SBASENUM(I)=SBASENUM(K);
          SBASENUM(K)=HOLDIT;
          HOLDTAG=TAG(I);
          TAG(I)=TAG(K);
          TAG(K)=HOLDTAG;
          VALUE1=SBASENUM(I);
ENDK: END;    /*  END LOOP K    */
     END;     /*  END LOOP I     */

/*   APPEND NUMBER OF 1'S TO EACH ELEMENT IN SOURCETAB   */
     DO J=1 TO MERGENUM;
          SNUMBERONES(J)=NUMCNES(SBASENUM(J));
          END;
     FREE MERGETAB;
     END MERGE;
```

```
COVERIT: PROC (NUMCOVER,ORCOST,ANDCOST,INPUTCOST,INTERCOST);

/*  THIS SUBROUTINE SETS UP THE ACTUAL PRIME IMPLICANTS IN THE PROPER
    FORM, DETERMINES WHICH TERMS EACH PRIME IMPLICANT COVERS,
    CONSTRUCTS THE IN-CORE REPRESENTATION OF THE PRIME IMPLICANT
    TABLE, BUILDS A COST TABLE FOR ALL THE PRIME IMPLICANTS, AND
    PRINTS THE PRIME IMPLICANT TABLE.                              */

DECLARE
01  MASTER                     CONTROLLED EXTERNAL,
    03  TABLE (SIZEA,SIZEB),
        05  TERM               BINARY FIXED (15),
    03  CNTTABLE (SIZEB),
        05  NUMTERMS           BINARY FIXED (15),
    03  BINARRAY (0:NUMBINTAB),
        05  NUMONES            BINARY FIXED (15),
01  PRIME (500)                EXTERNAL,
    03  BASE                   BINARY FIXED (15),
    03  DIFFER                 BINARY FIXED (15),
    03  OUTPUTTAG              BIT (10) ALIGNED,
01  REFCOL (NUMOUTPUTS)        CONTROLLED EXTERNAL,
    03  FIRSTCOL               BINARY FIXED (15),
    03  LASTCOL                BINARY FIXED (15),
01  REFER                      STATIC EXTERNAL,
    03  NUMVAR                 BINARY FIXED (15),
    03  NUMOUTPUTS             BINARY FIXED (15),
    03  NUMBINTAB              BINARY FIXED (15),
    03  TOPOFPAGE              CHAR (79),
    03  SIZEA                  BINARY FIXED (15),
    03  SIZEB                  BINARY FIXED (15),
    03  ESSENROW               BINARY FIXED (15),
    03  WIDTH                  BINARY FIXED (15),
    03  SIZE                   BINARY FIXED (15),
    03  POFS                   BIT (1),
    03  SOFP                   BIT (1),
01  TABLIST                    CONTROLLED EXTERNAL,
```

```
    03  PITABLE (0:ESSENROW,0:WIDTH) BIT (1),
    03  REFTAB (WIDTH),
        05  TOPROW             BINARY FIXED (15),
        05  COLREF             BINARY FIXED (15),
    03  ALLCOSTS (ESSENROW),
        05  GATECOST           BINARY FIXED (10),
        05  INPUT              BINARY FIXED (10),
        05  INTER              BINARY FIXED (10),
        05  COST               BINARY FIXED (31,12),
        05  MEMBER             BIT (NUMCUTPUTS) ALIGNED,
    03  PIPOFS (ESSENROW,SIZE) CHAR (1),
    03  PISOFP (ESSENROW,SIZE) CHAR (1),
ALPHABET (10) CHAR (1) INITIAL ('A','B','C','D','E','F','G','H','I',
                                'J'),
ANDCOST                         BINARY FIXED (31,12),
BITNUM                          BINARY FIXED (15),
BITROW                          BINARY FIXED (15),
BITVAL (ESSENROW,NUMBIT)        BINARY FIXED (15) CONTROLLED,
BLANK                           CHAR (1) INITIAL (' '),
CENTER                          BINARY FIXED (15),
CONTINUED                       CHAR (10),
(COUNT1,COUNT2)                 BINARY FIXED (15),
COVER (ESSENROW,NUMCOVER)       BINARY FIXED (15) CONTROLLED,
COVERCOL                        BINARY FIXED (15),
COVERROW                        BINARY FIXED (15),
ENDVAL                          BINARY FIXED (15),
(HISWITCH,LOSWITCH)             BIT (1),
(HIVAL,LOVAL)                   BINARY FIXED (15),
(I,J,JJ,K,KK,L,M,N)             BINARY FIXED (15),
ICOL                            BINARY FIXED (15),
ICOUNT                          BINARY FIXED (15),
INPUTCOST                       BINARY FIXED (31,12),
INTERCOST                       BINARY FIXED (31,12),
MAXNUM                          BINARY FIXED (15) INITIAL (0),
MORE                            BINARY FIXED (15),
NEWBASE                         BINARY FIXED (15),
```

```
NEWDIFF                         BINARY FIXED (15),
NONE                            BIT (1),
NUMBER                          BIT (15),
NUMBIT                          BINARY FIXED (15),
NUMCOVER                        BINARY FIXED (15),
NUMFUNC                         BINARY FIXED (15),
ORCOST                          BINARY FIXED (31,12),
PRINTX                          BIT (1),
(START,END)                     BINARY FIXED (15),
(STARTCOL,ENDCOL)               BINARY FIXED (15),
(STRING1,STRING2)               BIT (10),
TOTSUM                          BINARY FIXED (15),
VALUE                           BINARY FIXED (15);

    STRING1='0'B;
    ENDVAL=17-NUMVAR;
    SIZE=3*NUMVAR;
    WIDTH=0;
    DO I=1 TO NUMOUTPUTS*2 BY 2;
        WIDTH=WIDTH+NUMTERMS(I);
        END;
    ALLOCATE TABLIST;
    PISOFP=' ';
    PIPOFS=' ';
    NUMBIT=NUMVAR+2;
    ALLOCATE BITVAL;
    BITVAL=-9;
    BITROW=0;

/*  BREAK DOWN THE TOTAL DIFFERENCE STORED WITH THE NUMERICAL
    REPRESENTATION OF THE PRIME IMPLICANT INTO THE ACTUAL VARIABLES
    THAT HAVE BEEN ELIMINATED.                                      */
    DO M=1 TO ESSENROW;
      NEWBASE=BASE(M);  NEWDIFF=DIFFER(M);
      BITROW=BITROW+1;
      ICOL=2;
```

```
      BITVAL(BITROW,ICOL)=NEWBASE;
/*   STRING1 IS FOR THE BASE OF THE NUMBER.
     STRING2 IS FOR THE VARIABLES THAT HAVE BEEN ELIMINATED.
     FOR EXAMPLE, FOR PRIME IMPLICANT 8(4,2), THE VALUE OF
     STRING1 IS '1000000000' AND THE VALUE OF STRING2 IS '0110000000'.
     IF FINDING THE SUM OF PRODUCTS, THEN A 1 IN STRING1 REPRESENTS
     AN UNCOMPLEMENTED VARIABLE;  A 0, A COMPLEMENTED VARIABLE.
     IF FINDING A PRODUCT OF SUMS, A 1 IN STRING1 REPRESENTS A
     COMPLEMENTED VARIABLE;  A 0, AN UNCOMPLEMENTED VARIABLE.        */
     STRING1=SUBSTR(UNSPEC(NEWBASE),ENDVAL,NUMVAR);
     STRING2='0'B;
       IF NEWDIFF=0 THEN GO TO LEAVEM;
       DO I=NUMVAR TO 0 BY -1;
         IF NEWDIFF=2**I THEN DO;
            ICOL=ICOL+1;
            BITVAL(BITROW,ICOL)=NEWDIFF;
            BITNUM=NUMVAR-I;
            SUBSTR(STRING2,BITNUM,1)='1'B;
            GO TO LEAVEM;  END;
          IF NEWDIFF>2**I THEN DO;
            VALUE=2**I;
            ICOL=ICOL+1;
            BITVAL(BITROW,ICOL)=VALUE;
            BITNUM=NUMVAR-I;
            SUBSTR(STRING2,BITNUM,1)='1'B;
            NEWDIFF=NEWDIFF-VALUE;
            END;
         END;  /*  END LOOP I  */
       PUT FILE (SYSPRINT) EDIT ('ERROR - FELL THROUGH LOOP - ',
       'NEWBASE=',NEWBASE,'NEWDIFF =',NEWDIFF,'NEWDIFF IS NOT A POWER',
       ' OF 2.') (SKIP(3),A,A,F(3),X(2),A,F(3),X(2),A,A);
       STOP;
LEAVEM:
     BITVAL(BITROW,1)=ICOL-2;
     IF MAXNUM<(ICOL-2) THEN MAXNUM=ICOL-2;
     COUNT1=0;
```

```
    COUNT2=0;

/*  FORM THE 'ALPHABETICAL' REPRESENTATION OF THE PRIME IMPLICANTS  */
    DO K=1 TO NUMVAR;
        IF SOFP THEN DO;
            /*  WAS THAT VARIABLE ALREADY ELIMINATED?               */
            IF SUBSTR(STRING2,K,1) THEN GO TO ENDK;
            COUNT1=COUNT1+1;  /*  COLUMN OF TABLE PISCFP          */
            PISOFP(M,COUNT1)=ALPHABET(K);
            IF ¬SUBSTR(STRING1,K,1) THEN DO;
                COUNT1=COUNT1+1;
                PISCFP(M,COUNT1)='''';
                END;
            COUNT1=COUNT1+1;
            PISOFP(M,COUNT1)='*';
            END;
        IF POFS THEN DO;
            /*  WAS THAT VARIABLE ALREADY ELIMINATED?               */
            IF SUBSTR(STRING2,K,1) THEN GO TO ENDK;
            COUNT2=COUNT2+1;  /*  COLUMN OF TABLE PIPOFS          */
            PIPOFS(M,COUNT2)=ALPHABET(K);
            IF SUBSTR(STRING1,K,1) THEN DO;
                COUNT2=COUNT2+1;
                PIPOFS(M,COUNT2)='''';
                END;
            COUNT2=COUNT2+1;
            PIPOFS(M,COUNT2)='+';
            END;
ENDK:
        END;  /*  END LOOP K  */
      PISOFP(M,COUNT1)=' ';  PIPOFS(M,COUNT2)=' ';
    END;  /*  END LOOP M  */

/*  NOW DETERMINE WHAT EACH PRIME IMPLICANT COVERS                  */
    NUMCOVER=2**MAXNUM+1;
    ALLOCATE COVER;
```

```
    COVER=-9;
/*  ESSENROW AND BITROW SHOULD BE EQUAL  */
    COVERROW=0;
    DO M=1 TO BITROW;
    ENDVAL=16-BITVAL(M,1);
    L=2**(BITVAL(M,1))-1;
    NEWBASE=BITVAL(M,2);
      COVERCOL=2;
      COVERROW=COVERROW+1;
        COVER(COVERROW,COVERCOL)=NEWBASE;
      DO I=1 TO L;
      TOTSUM=NEWBASE;
        NONE='1'B;
        NUMBER=SUBSTR(UNSPEC(I),2,15);
        DO J=15 TO ENDVAL BY -1;
          IF SUBSTR(NUMBER,J,1) THEN DO;
            K=J-ENDVAL+3;
            NONE='0'B;
            TOTSUM=TOTSUM+BITVAL(COVERROW,K);
            END;  ELSE;
      END;  /*  END LOOP J  */
          IF NONE THEN GO TO ENDI;
          COVERCOL=COVERCOL+1;
          COVER(COVERROW,COVERCOL)=TOTSUM;
ENDI:     END;
      COVER(COVERROW,1)=COVERCOL-1;
      END;  /*  END LOOP M  */
    MEMBER(*)='0'B;
    PITABLE='0'B;

/*  THE TOP ROW (ROW ZERO) AND THE LEFT-MOST COLUMN (COLUMN ZERO)
    OF PITABLE ARE FLAGS INDICATING WHETHER OR NOT THAT ROW OR
    COLUMN HAS BEEN ELIMINATED BY ONE OF THE REDUCTION TECHNIQUES.
    IF A ROW OR COLUMN HAS BEEN REMOVED, THEN THE FLAG BIT IS
    TURNED ON.  IF THE PRIME IMPLICANT CORRESPONDING TO A ROW
    COVERS ONLY PHI-TERMS, THEN THE FLAG BIT FOR THAT ROW IS TURNED
```

```
     ON WHEN THE TABLE IS CONSTRUCTED.  SINCE EVERY COLUMN MUST
     CONTAIN AT LEAST ONE X WHEN THE TABLE IS CONSTRUCTED THEN THE
     FLAG BIT FOR EACH COLUMN WILL ALWAYS BE OFF WHEN THE TABLE IS
     CONSTRUCTED.  IN INITIALIZING THE PITABLE, THE FLAG BITS ARE
     ALL TURNED ON.  THEN AS THE PITABLE IS CONSTRUCTED AND THE
     X'S PUT INTO THE PROPER PLACES, THE CORRESPONDING FLAG BITS ARE
     TURNED OFF, SIGNIFYING THAT THERE IS AN X IN THAT ROW OR COLUMN.*/
     PITABLE(0,*)='1'B;
     PITABLE(*,0)='1'B;
     ICOUNT=0;
     DO I=1 TO NUMOUTPUTS*2 BY 2;
         DO K=1 TO NUMTERMS(I);
             ICOUNT=ICOUNT+1;
             TOPROW(ICOUNT)=TERM(K,I);
             END;  /*  END LOOP K  */
         END;  /*  END LOOP I  */

/*   BUILD A COLUMN-REFERENCE TABLE THAT WILL TELL WHAT FUNCTION
     A COLUMN OF THE PRIME IMPLICANT TABLE BELONGS TO.
     BUILD THE IN-CORE REPRESENTATION OF THE PRIME IMPLICANT
     TABLE, AND ALSO BUILD THE TABLE OF COSTS.                    */
     DO N=1 TO ESSENROW;  /*  PROCESS THRU TABLE PRIME  */
         INPUT(N)=NUMVAR-BITVAL(N,1);
         INTER(N)=0;
         GATECOST(N)=0;  /*  IN CASE A P.I. COVERS ONLY PHITERMS    */
         ICOUNT=COVER(N,1)+1;
         DO I=1 TO NUMOUTPUTS;  /* PROCESS THRU OUTPUTTAG  */
             IF ¬(SUBSTR(OUTPUTTAG(N),I,1)) THEN GO TO ENDI1;
             GATECOST(N)=1;
             INTER(N)=INTER(N)+1;
             J=2*I-3;
             STARTCOL=1;
             DO K=1 TO J BY 2;
                 STARTCOL=STARTCOL+NUMTERMS(K);
                 END;  /*  END LOOP K  */
             ENDCOL=STARTCOL+NUMTERMS(J+2)-1;
```

```
            M=2;  /*  M CONTROLS WHICH ELEMENT IN COVER  */
            FIRSTCOL(I)=STARTCOL;
            LASTCOL(I)=ENDCOL;
            DO KK=STARTCOL TO ENDCOL;
                COLREF(KK)=I;
                END;
NEXTONE:
            IF COVER(N,M)>TOPROW(STARTCOL) THEN DO;
                STARTCOL=STARTCOL+1;
                IF STARTCOL>ENDCOL THEN GO TO ENDI1;
                GO TO NEXTONE;
                END;
            IF COVER(N,M)<TOPROW(STARTCOL) THEN DO;
                M=M+1;
            IF M>ICOUNT THEN GO TO ENDI1;
                GO TO NEXTONE;
                END;
/*  HAVE FOUND A MATCHING TERM - INSERT IT IN PRIME IMPLICANT TABLE.
    TURN OFF BIT IN THAT ROW AND COLUMN TO INDICATE THAT THERE IS
    SOMETHING IN THAT COLUMN OR ROW                                  */
            PITABLE(0,STARTCOL)='0'B;  PITABLE(N,0)='0'B;
            PITABLE(N,STARTCOL)='1'B;
            STARTCOL=STARTCOL+1;
            M=M+1;
            IF M>ICOUNT | STARTCOL>ENDCOL THEN GO TO ENDI1;
            GO TO NEXTONE;
 ENDI1:     END;  /*  END LOOP I  */
      /*  IF A PRIME IMPLICANT CONSISTS OF ONLY ONE VARIABLE, IN
          WHICH CASE INPUT(N)=1, THEN NO GATES ARE NEEDED TO
          IMPLEMENT THAT PRIME IMPLICANT AND THE VALUE OF
          GATECOST(N) SHOULD BE SET TO 0.  ALSO, SINCE A SINGLE
          VARIABLE IS CONSIDERED AS AN INPUT TO THE SECOND LEVEL OF
          GATING, OR AN INTERCONNECTION, THEN SET INPUT(N)=0 AND LET
          INTER(N) REMAIN AS IS.                                 */
        IF INPUT(N)=1 THEN DO;
          GATECOST(N)=0;  INPUT(N)=0;
```

```
        END;
      IF POFS THEN COST(N)=(ORCOST*GATECOST(N))+(INPUTCOST*INPUT(N))+
      (INTERCOST*INTER(N)); ELSE COST(N)=(ANDCOST*GATECOST(N))+
      (INPUTCOST*INPUT(N))+(INTERCOST*INTER(N));
      END;  /*  END LOOP N  */
    FREE COVER;

/*  NOW READY TO PRINT THE PRIME IMPLICANT TABLE  */
    CONTINUED=(10)' ';
    ENDCOL=0;
    NUMFUNC=1;
/*  JJ IS THE FUNCTION NUMBER THAT THE STARTING COLUMN PERTAINS TO.
    NUMFUNC IS THE FUNCTION NUMBER THAT THE ENDING COLUMN PERTAINS TO*/
NEXT_PART:
    IF (ESSENROW + LINENO(SYSPRINT) + 10) >= 60 THEN
    SIGNAL ENDPAGE (SYSPRINT);
    PRINTX='0'B;  /* TURNED ON IF NEED TO PRINT A BREAK BAR AFTER  */
                  /* THE LAST CHARACTER ON THAT LINE                */
    STARTCOL=ENDCOL+1; /* THE COLUMN IN PITABLE TO START WITH  */
    ENDCOL=STARTCOL+17;  /* THE COLUMN IN PITABLE TO END WITH  */
    LOSWITCH='0'B;  HISWITCH='0'B;
/*  CAN FIT A MAXIMUM OF 17 TERMS ON A LINE INCLUDING BREAK BARS
    THE TOTAL NUMBER OF MINTERM COLUMNS THAT WILL BE PRINTED IS
    ((ENDCOL-STARTCOL+1)-(HIVAL-LOVAL+1)+1).  THE OTHER COLUMNS
    ARE BREAK BARS FOR EACH FUNCTION PLUS ONE EXTRA BREAK BAR FOR
    SEPARATING THE START OF A FUNCTION FROM THE PRIME IMPLICANTS.   */
/*  DETERMINE THE STARTING FUNCTION AND THE ENDING FUNCTION
    BASED ON THE VALUES OF STARTCOL AND ENDCOL.                     */
    DO J=NUMFUNC TO NUMOUTPUTS;
        IF LOSWITCH THEN GO TO FINDHI;
        IF STARTCOL >= FIRSTCOL(J) & STARTCOL <= LASTCOL(J) THEN DO;
            LOVAL=J;  LOSWITCH='1'B;  END;
        IF HISWITCH THEN GO TO BOTHSW;
FINDHI:
        IF ENDCOL >= FIRSTCOL(J) & ENDCOL <= LASTCOL(J) THEN DO;
            HIVAL=J;  HISWITCH='1'B;  END;
```

```
/*  IF BOTH SWITCHES ARE ON, THEN HAVE FOUND THE RIGHT FUNCTION
    NUMBERS FOR THE STARTING AND ENDING COLUMNS.                    */
BOTHSW:
        IF LOSWITCH & HISWITCH THEN GO TO LEAVEJ;
        END;  /*  END LOOP J  */
/*  AT THIS POINT, ENDCOL > WIDTH SO READY TO PRINT LAST OF TABLE   */
    HIVAL=NUMOUTPUTS;
LEAVEJ:
/*  ALLOW FOR BREAK BARS BETWEEN THE FUNCTIONS  */
    ENDCOL=ENDCOL-(HIVAL-LOVAL+1);
    IF ENDCOL >= WIDTH THEN DO;
        ENDCOL=WIDTH;  PRINTX='1'B;  GO TO TRYAGAIN;
        END;
    ELSE IF COLREF(ENDCOL) ¬= COLREF(ENDCOL+1) THEN PRINTX='1'B;
    ELSE IF COLREF(ENDCOL) ¬= COLREF(ENDCOL-1) THEN DO;
        ENDCOL=ENDCOL-1;  HIVAL=HIVAL-1;  PRINTX='1'B;
        END;
    ELSE;
/*  NOW MAKE SURE THAT ENDCOL IS BETWEEN THE FIRSTCOL AND
    LASTCOL VALUES FOR FUNCTION NUMBER HIVAL.                       */
TRYAGAIN:
    IF ENDCOL >= FIRSTCOL(HIVAL) & ENDCOL <= LASTCOL(HIVAL) THEN;
    ELSE DO;
        HIVAL=HIVAL-1;  GO TO TRYAGAIN;  END;
    NUMFUNC=HIVAL;
    JJ=LOVAL;
/*  CENTER THE TABLE HEADING OVER THE TABLE BASED ON ITS WIDTH  */
    CENTER=((ENDCOL-STARTCOL+1+HIVAL-LOVAL+1)*5)+29;
    PUT FILE (SYSPRINT) EDIT ('PRIME IMPLICANT TABLE',CONTINUED)
    (SKIP(3),X(CENTER/2),A,A);
    IF SOFP THEN PUT FILE (SYSPRINT) EDIT ('MINTERMS') (SKIP(2),
    X(CENTER/2+7),A);  ELSE PUT FILE (SYSPRINT) EDIT ('MAXTERMS')
    (SKIP(2),X(CENTER/2+7),A);
    PUT FILE (SYSPRINT) EDIT (BLANK) (SKIP,X(27),A);
    DO I=JJ TO NUMFUNC;
        IF STARTCOL <= FIRSTCOL(I) THEN
```

```
          IF ENDCOL >= LASTCOL(I)
            THEN CENTER=(LASTCOL(I)-FIRSTCOL(I)+1)*5+5;
            ELSE CENTER=(ENDCOL-FIRSTCOL(I)+1)*5+5;
        ELSE IF ENDCOL >= LASTCOL(I)
          THEN CENTER=(LASTCOL(I)-STARTCOL+1)*5+5;
          ELSE CENTER=(ENDCOL-STARTCOL+1)*5+5;
        IF (MOD(CENTER,2)) = 1 THEN CENTER=CENTER-1;
        IF CENTER <= 10 THEN
          IF I=JJ THEN
            IF STARTCOL > FIRSTCOL(I) THEN PUT FILE (SYSPRINT) EDIT
            ('FUNCTION',I) (X(1),A,F(2));   /* HAVE SMALL PART OF
                          FUNCTION THAT HAS BEEN BROKEN ON THE PAGE. */
            ELSE DO;
/*  HAVE A VERY SMALL FUNCTION TO START PRINTING ON THE PAGE        */
              PUT FILE (SYSPRINT) EDIT (BLANK) (X(5),A);
              PUT FILE (SYSPRINT) EDIT ('FUNCTION',I) (X(1),A,F(2));
              END;
          ELSE DO;
/*  HAVE A VERY SMALL FUNCTION TO PRINT IN MIDDLE OF THE PAGE       */
            PUT FILE (SYSPRINT) EDIT ('FUNCTION',I) (X(1),A,F(2));
            END;
        ELSE DO;
/*  CENTER > 10 SO USE REGULAR PRINTING METHOD TO PRINT  */
          IF I=JJ THEN PUT FILE (SYSPRINT) EDIT (BLANK) (X(5),A); ELSE;
          PUT FILE (SYSPRINT) EDIT ('FUNCTION',I,BLANK)
          (X(CENTER/2-8),A,F(2),X(CENTER/2-3),A);
          END;
/*  THIS FORMAT ALLOWS FOR CENTER SPACES FOR THE FUNCTION  */
        END;  /*  END LOOP I  */
    PUT FILE (SYSPRINT) EDIT ('PRIME IMPLICANTS',BLANK)
    (SKIP,A,X(13),A);
    DO I=STARTCOL TO ENDCOL;
        IF JJ > NUMOUTPUTS THEN GO TO PRINTTOP;
        IF STARTCOL > FIRSTCOL(JJ) THEN JJ=JJ+1;  /* TO ALLOW FOR
                                    BREAKING A FUNCTION ON TWO LINES. */
        IF I=FIRSTCOL(JJ) THEN DO;
```

```
            PUT FILE (SYSPRINT) EDIT ('|') (X(4),A);
            JJ=JJ+1;
            END;
PRINTTOP:
        PUT FILE (SYSPRINT) EDIT (TOPROW(I)) (X(1),F(4));
        END;  /*  END LOOP I  */
    IF PRINTX THEN PUT FILE (SYSPRINT) EDIT ('|') (X(4),A);

/*  HAVE PRINTED MINTERMS OR MAXTERMS AT TOP OF TABLE;
    NOW PRINT THE REST OF THE PRIME IMPLICANT TABLE.                    */
    DO I=1 TO ESSENROW;
    JJ=LOVAL;
        PUT FILE (SYSPRINT) SKIP;
        IF SOFP THEN PUT FILE (SYSPRINT) EDIT ((PISOFP(I,K) DO K=1 TO
        SIZE)) (A);  ELSE PUT FILE (SYSPRINT) EDIT ((PIPOFS(I,K) DO K=1
        TO SIZE)) (A);
        PUT FILE (SYSPRINT) EDIT (BLANK) (X(29-SIZE),A);
        DO K=STARTCOL TO ENDCOL;
            IF JJ > NUMOUTPUTS THEN GO TO PRINTBIT;
            IF STARTCOL > FIRSTCOL(JJ) THEN JJ=JJ+1;  /*  ALLOW FOR
                                    BREAKING A FUNCTION ON TWO LINES. */
            IF K=FIRSTCOL(JJ) THEN DO;
                PUT FILE (SYSPRINT) EDIT ('|') (X(4),A);
                JJ=JJ+1;  END;
PRINTBIT:
            IF PITABLE(I,K) THEN PUT FILE (SYSPRINT) EDIT ('X') (X(4),
            A);  ELSE PUT FILE (SYSPRINT) EDIT (BLANK) (X(4),A);
            END;  /*  END LOOP K  */
        IF PRINTX THEN PUT FILE (SYSPRINT) EDIT ('|') (X(4),A);
        END;  /*  END LOOP I  */
    PRINTX='0'B;
/*  HAVE PRINTED THE FIRST PART OF THE PRIME IMPLICANT TABLE.
    NOW SEE IF THERE IS ANY MORE OF IT TO PRINT.                        */
    IF ENDCOL < WIDTH THEN DO;
        CONTINUED=' (CONT''D.)';
        GO TO NEXT_PART;
```

```
        END;
/*  NOW THE COMPLETE PRIME IMPLICANT TABLE HAS BEEN PRINTED.          */
    PUT FILE (SYSPRINT) SKIP(3);
    FREE BITVAL;
END COVERIT;
```

```
CYCLIC: PROC (INTERCOST);

/*  THIS SUBROUTINE FORMS THE P-FUNCTION TO FIND A SOLUTION
    TO A CYCLIC PRIME IMPLICANT TABLE.                          */

DECLARE
01  FUNCTNS (NEWCOL)          CONTROLLED,
    03  OLDFUNC               BIT (NUMOUTPUTS),
    03  NEWFUNC               BIT (NUMOUTPUTS),
01  REFCOL (NUMOUTPUTS)       CONTROLLED EXTERNAL,
    03  FIRSTCOL              BINARY FIXED (15),
    03  LASTCOL               BINARY FIXED (15),
01  REFER                     STATIC EXTERNAL,
    03  NUMVAR                BINARY FIXED (15),
    03  NUMOUTPUTS            BINARY FIXED (15),
    03  NUMBINTAB             BINARY FIXED (15),
    03  TOPOFPAGE             CHAR (79),
    03  SIZEA                 BINARY FIXED (15),
    03  SIZEB                 BINARY FIXED (15),
    03  ESSENROW              BINARY FIXED (15),
    03  WIDTH                 BINARY FIXED (15),
    03  SIZE                  BINARY FIXED (15),
    03  POFS                  BIT (1),
    03  SOFP                  BIT (1),
01  SIZETAB (NUMOUTPUTS)      CONTROLLED,
    03  NUMROWS               BINARY FIXED (31),
01  STRINGS                   CONTROLLED,
    03  PRODTERM              BIT (ESSENROW),
    03  SAVE                  BIT (ESSENROW),
01  TABLEA                    CONTROLLED,
    03  OLDCNT (NUMCOL)       BINARY FIXED (31),
    03  OLDTAB (MAXNUMA,NUMCOL) BIT (ESSENROW),
01  TABLEB                    CONTROLLED,
    03  RANK(NEWCOL)          BINARY FIXED (31),
    03  NEWCNT (NEWCOL)       BINARY FIXED (31),
    03  NEWTAB (MAXNUMB,NEWCOL) BIT (ESSENROW),
```

```
01  TABLEC                          CONTROLLED,
    03  CCOUNT                      BINARY FIXED (31),
    03  CTAB (MAXNUMA)              BIT (ESSENROW),
01  TABLIST                         CONTROLLED EXTERNAL,
    03  PITABLE (0:ESSENROW,0:WIDTH) BIT (1),
    03  REFTAB (WIDTH),
        05  TOPROW                  BINARY FIXED (15),
        05  COLREF                  BINARY FIXED (15),
    03  ALLCOSTS (ESSENROW),
        05  GATECOST                BINARY FIXED (10),
        05  INPUT                   BINARY FIXED (10),
        05  INTER                   BINARY FIXED (10),
        05  COST                    BINARY FIXED (31,12),
        05  MEMBER                  BIT (NUMOUTPUTS) ALIGNED,
    03  PIPOFS (ESSENROW,SIZE) CHAR (1),
    03  PISOFP (ESSENROW,SIZE) CHAR (1),
01  WHICHPI                         CONTROLLED,
    03  NOWTAB (NUMOUTPUTS,ESSENROW)  BIT (1),
    03  DUPTAB (NUMOUTPUTS,ESSENROW)  BIT (1),
AGAIN                               BIT (1),
BLANK                               CHAR (1) INITIAL (' '),
CHOICEPI                            BINARY FIXED (15),
CURRENT                             BINARY FIXED (15),
FINALP                              BIT(1),
HOLDCOL                             BINARY FIXED (15),
HOLDROW                             BINARY FIXED (15),
(I,J,K,L,M,N,NN)                    BINARY FIXED (15),
ICOL                                BINARY FIXED (15),
INTERCOST                           BINARY FIXED (31,12),
IROW                                BINARY FIXED (15),
KEEPNEWCOL                          BINARY FIXED (15),
LOWCOST                             BINARY FIXED (31,12),
(MAXNUMA,MAXNUMB,MAXNUMC)           BINARY FIXED (31),
NEWCOL                              BINARY FIXED (15),
NEWROW                              BINARY FIXED (15),
NEXT                                BINARY FIXED (15),
```

```
NUMCHOICE                       BINARY FIXED (15),
NUMCOL                          BINARY FIXED (15),
NUMLEFT                         BINARY FIXED (15),
NUMPERCOL                       BINARY FIXED (15),
PFNCOST (MAXNUMB)               BINARY FIXED (31,12) CONTROLLED,
PICHOICE                        BINARY FIXED (15),
PTRC                            BINARY FIXED (15),
TEMP                            BINARY FIXED (15),
TEMPPI (0:WIDTH)                BIT (1) CONTROLLED,
USEDPI (ESSENROW)               BIT (1) CONTROLLED;

PFUNCTION:
    ALLOCATE STRINGS;
    FINALP='0'B;  /*  TURNED ON WHEN FINDING THE P-FUNCTION FOR THE
                                                    ENTIRE FUNCTION. */
    PTRC=0;  /*  POINTS TO THE APPROPRIATE COLUMN OF SIZETAB THAT
                        RECORDS THE STACKED ALLOCATIONS OF TABLEC.   */
    MAXNUMC=0;
    ALLOCATE SIZETAB;
    SIZETAB=0;
    I=1;  /*  I CONTROLS THE NUMBER OF THE OUTPUT FUNCTION          */

/*  SET UP THE FIRST TABLE FROM THE PART OF THE
    PRIME IMPLICANT TABLE FOR FUNCTION I.                           */
NEXTFN:
    MAXNUMA=0;  NUMCOL=0;
    DO J=FIRSTCOL(I) TO LASTCOL(I);
      IF PITABLE(0,J) THEN GO TO INCJ;
      /*  NUMCOL IS THE NUMBER OF UNCOVERED COLUMNS.
          MAXNUMA IS EQUAL TO LARGEST NUMBER OF ONES IN
          ANY COLUMN OF THE PRIME IMPLICANT TABLE.                  */
      NUMCOL=NUMCOL+1;  NUMPERCOL=0;
      DO K=1 TO ESSENROW;
        IF PITABLE(K,J) THEN NUMPERCOL=NUMPERCOL+1;
        END;  /*  END LOOP K  */
      IF NUMPERCOL > MAXNUMA THEN MAXNUMA = NUMPERCOL;
```

```
INCJ:
      END;  /*  END LOOP J  */
    IF NUMCOL=0 THEN DO;
      /*  THAT FUNCTION IS ALREADY COVERED SO IT WILL
          NOT BE INCLUDED IN THE P-FUNCTION EXCEPT AS A DUMMY.      */
      MAXNUMA=0; /* DUMMY LENGTH FOR THE TABLE                      */
      NEWCOL=1;  MAXNUMB=1;
      ALLOCATE TABLEB;
      NEWCNT(1)=0;
      GO TO STOREP;
      END;
    ALLOCATE TABLEA;
    OLDTAB='0'B;
    ICOL=0;
/*  BUILD TABLE OLDTAB FROM THE PITABLE  */
    OLDCNT=0;
    DO J=FIRSTCOL(I) TO LASTCOL(I);
      IF PITABLE(0,J) THEN GO TO ADDJ; /* COLUMN ALREADY COVERED    */
      ICOL=ICOL+1;  IROW=0;
      DO K=1 TO ESSENROW;
        IF ¬PITABLE(K,J) THEN GO TO ADDK;  /*  A ZERO IN PITABLE    */
        IROW=IROW+1;
        SUBSTR(OLDTAB(IROW,ICOL),K,1)='1'B;
        OLDCNT(ICOL)=OLDCNT(ICOL)+1;
ADDK:
        END;  /*  END LOOP K  */
ADDJ:
      END;  /*  END LOOP J  */
    IF NUMCOL=1 THEN DO;
      /*  THERE IS ONLY ONE COLUMN IN THIS FUNCTION THAT IS NOT
          COVERED.  SINCE THERE IS NOT ANOTHER COLUMN BY WHICH TO
          MULTIPLY IT, JUST MOVE THE ONE COLUMN INTO TABLEB, AND
          THEN GO TO STOREP.                                        */
      NEWCOL=NUMCOL;  MAXNUMB=MAXNUMA;
      ALLOCATE TABLEB;
      NEWCNT(NEWCOL)=OLDCNT(NEWCOL);
```

```
      DO J=1 TO NEWCNT(NEWCOL);
        NEWTAB(J,NEWCOL)=OLDTAB(J,NEWCOL);
        END;  /*  END LOOP J  */
      GO TO STOREP;
      END;

MULTAGAIN:
/*  NOW READY TO MULTIPLY IN PAIRS - WORK TOWARD THE MIDDLE
    OF OLDTAB FROM THE ENDS OF THE TABLE.                          */
    MAXNUMB=0;
    DO J=1 TO NUMCOL/2 BY 1;
      IF (OLDCNT(J)*OLDCNT(NUMCOL-J+1)) > MAXNUMB THEN
      /*  USE MAXNUMB TO DETECT THE MAXIMUM NUMBER OF ENTRIES
          THAT CAN BE GENERATED IN TABLEB                          */
      MAXNUMB=OLDCNT(J) * OLDCNT(NUMCOL-J+1);
      END;  /*  END LOOP J  */
    IF MOD(NUMCOL,2)=1 THEN IF OLDCNT(NUMCOL/2+1) > MAXNUMB
    THEN MAXNUMB=OLDCNT(NUMCOL/2+1);  ELSE;  ELSE;
    NEWCOL=NUMCOL/2;
    IF MOD(NUMCOL,2)=1 THEN NEWCOL=NEWCOL+1;
    KEEPNEWCOL=NEWCOL;
    IF FINALP THEN IF MAXNUMB > 32767 THEN DO;

/*  DUE TO LIMITATIONS OF AVAILABLE CORE, ONLY A NEAR-MINIMAL
    SOLUTION CAN BE OBTAINED.  THE SOLUTION WILL BE MINIMIZED
    ACCORDING TO WHAT HAS BEEN DONE UP TO THIS POINT - WITH NUMCOL
    PARTITIONS.  MOVE TABLEA INTO TABLEB AND END THE
    P-FUNCTION ROUTINE.                                            */
      PUT FILE (SYSPRINT) EDIT ('DUE TO THE SIZE OF THE PROBLEM, THE',
      ' PROBLEM CANNOT BE MINIMIZED AS A WHOLE.  IT WILL BE MINIMIZED',
      ' IN ',NUMCOL,' PARTITIONS.') (SKIP(3),A,A,A,F(3),A);
      PUT FILE (SYSPRINT) EDIT ('THEREFORE, THE SOLUTION IS NOT ',
      'GUARANTEED TO BE THE MINIMAL SOLUTION, ALTHOUGH IT FREQUENTLY ',
      'IS.') (SKIP,A,A,A);
      NEWCOL=NUMCOL;  MAXNUMB=MAXNUMA;
      ALLOCATE TABLEB;
```

```
      DO J=1 TO NEWCCL;
        NEWCNT(J)=OLDCNT(J);  NEWFUNC(J)=OLDFUNC(J);
        DO K=1 TO NEWCNT(J);
          NEWTAB(K,J)=OLDTAB(K,J);
          END;  /*  END LOOP K  */
        END;  /*  END LOOP J  */
      DO J=1 TO NEWCOL;
        PUT FILE (SYSPRINT) EDIT ('PARTITION ',J,' CONTAINS THE ',
        'FOLLOWING FUNCTIONS:') (SKIP(2),A,F(3),A,A);
        DO K=1 TO NUMOUTPUTS;
          IF SUBSTR(NEWFUNC(J),K,1) THEN PUT FILE (SYSPRINT) EDIT (K)
          (X(2),F(3));
          END;  /*  END LOOP K  */
        PUT FILE (SYSPRINT) EDIT ('THERE ARE ',NEWCNT(J),
        ' DIFFERENT TERMS WITHIN THE P-FUNCTION FOR THIS PARTITION.')
        (SKIP,A,F(8),A);
        END;  /*  END LOOP J  */
      GO TO ENDPFN;
      END;  /*  END TEST FOR FINALP BIT ON                              */
    ALLOCATE TABLEB;
    MAXNUMA=0;
    DO J=1 TO NUMCOL/2 BY 1;
      TEMP=NUMCOL-J+1;
      NEWCOL=J;
      NEWROW=0;  /*  POINTER TO CURRENT ROW IN NEWTAB                  */
      DO K=1 TO OLDCNT(J);
        DO L=1 TO OLDCNT(TEMP);
          PRODTERM=OLDTAB(K,J) | OLDTAB(L,TEMP);
 /*  SEE IF THE NEW PRODUCT TERM HAS ALREADY BEEN GENERATED OR
     CAN BE ELIMINATED OR ELIMINATES A PREVIOUSLY GENERATED TERM.    */
          DO M=1 TO NEWROW;
            SAVE=PRODTERM | NEWTAB(M,NEWCOL);
            IF SAVE=PRODTERM THEN GO TO ADDL;
LOOP1:
            IF SAVE=NEWTAB(M,NEWCOL) THEN DO;
/*  SUBSTITUTE THE TERM WITH THE FEWER NUMBER OF LETTERS FOR A
```

```
    PREVIOUSLY GENERATED TERM.  THEN CHECK ALL THE OTHER TERMS TO
    SEE IF ANY OTHERS CAN BE ELIMINATED BY THIS 'SMALLER' TERM.
    START FROM THE BOTTOM OF THE LIST OF TERMS AND WORK UP.
    IF ONE IS ABLE TO BE ELIMINATED, THEN ALL THE TERMS BELOW IT
    MUST BE MOVED UP ONE ROW IN THE LIST.                          */
              NEWTAB(M,NEWCOL)=PRODTERM;
              HOLDROW=NEWROW;
              DO N=NEWROW TO M+1 BY -1;
                IF (PRODTERM | NEWTAB(N,NEWCOL)) = NEWTAB(N,NEWCOL)
                THEN DO; /* FOUND ONE TO ELIMINATE - MUST SHIFT UP  */
                  DO NN=N+1 TO HOLDROW;
                    NEWTAB(NN-1,NEWCOL)=NEWTAB(NN,NEWCOL);
                    END;  /*  END LOOP NN  */
                  HOLDROW=HOLDROW-1;
                  END;
                END;  /*  END LOOP N  */
              NEWROW=HOLDROW;       GO TO ADDL;
              END;  /*  END LOOP1  */
            END;  /*  END LOOP M  */
          NEWROW=NEWROW+1;  /*  ADD THIS TERM TO THE LIST OF TERMS  */
          NEWTAB(NEWROW,NEWCOL)=PRODTERM;
ADDL:
          END;  /*  END LOOP L  */
        END;  /*  END LOOP K  */
      NEWCNT(NEWCOL)=NEWROW;
      RANK(NEWCOL)=NEWCOL;
      IF FINALP THEN NEWFUNC(NEWCOL)=OLDFUNC(J) | OLDFUNC(TEMP);
/*  NOW MAXNUMA WILL BE USED TO DETECT THE LARGEST
    COLUMN IN TABLEB.  THIS WILL BE USED WHEN THE
    CONTENTS OF TABLEB ARE MOVED INTO TABLEA.                      */
      IF NEWROW > MAXNUMA THEN MAXNUMA=NEWROW;
      END;  /*  END LOOP J  */

ODDROWS:
/*  THERE IS AN ODD NUMBER OF ROWS IN TABLEA - INSERT THE
    MIDDLE ROW OF TABLEA AT THE END OF TABLEB.                     */
```

```
    IF KEEPNEWCOL ¬= NEWCOL THEN DO;
      TEMP=NUMCOL/2+1;
      NEWCOL=NEWCCL+1;
      NEWCNT(NEWCCL)=CLDCNT(TEMP);
      RANK(NEWCOL)=NEWCOL;
      IF FINALP THEN NEWFUNC(NEWCOL)=CLDFUNC(TEMP);
      DO J=1 TO NEWCNT(NEWCOL);
        NEWTAB(J,NEWCCL)=CLDTAB(J,TEMP);
        END;  /*  END LOOP J  */
      END;
CHECKALL:
    IF NEWCOL=1 THEN GO TO STOREP;

SRTMOVE:
/*  SORT THE COLUMNS OF TABLEB INTO DESCENDING ORDER.  THEN
    MOVE THE COLUMNS INTO THE APPROPRIATE COLUMN OF TABLEA.
    CURRENT POINTS TO THE COLUMN TO WHICH EVERY OTHER COLUMN THAT
    HAS A GREATER RANK IS BEING COMPARED.  NEXT POINTS TO THE
    COLUMN THAT IS CURRENTLY BEING COMPARED WITH COLUMN CURRENT.    */
    CURRENT=1;  NEXT=2;
COMPARE:
    IF NEWCNT(CURRENT) < NEWCNT(NEXT) THEN
      IF RANK(CURRENT) > RANK(NEXT) THEN GO TO INCNEXT;  ELSE DO;
        TEMP=RANK(CURRENT); RANK(CURRENT)=RANK(NEXT); RANK(NEXT)=TEMP;
        CURRENT=NEXT;  NEXT=NEXT+1;
        IF NEXT > NEWCOL THEN GO TO NEWCURRENT; /* AT END OF ROW    */
        ELSE GO TO COMPARE;
        END;
INCNEXT:
    NEXT=NEXT+1;
    IF NEXT > NEWCOL THEN GO TO NEWCURRENT;
    GO TO COMPARE;
NEWCURRENT:
    IF RANK(CURRENT)+1=NEWCOL THEN GO TO ENDSORT;
    DO K=1 TO NEWCOL;
      IF RANK(CURRENT)+1 = RANK(K) THEN DO;
```

```
        CURRENT=K;  NEXT=1;
        GO TO COMPARE;  END;
      END;  /*  END LOOP K  */
ENDSORT:
/*  NOW FREE TABLEA AND MOVE TABLEB INTO IT WITH COLUMNS SORTED
    ACCORDING TO THE NUMBER OF ELEMENTS IN THAT COLUMN.          */
    FREE TABLEA;
    NUMCOL=NEWCOL;
    ALLOCATE TABLEA;
/*  NOW TRANSFER TABLEB INTO TABLEA ACCORDING TO THE RANK IN TABLEB */
    DO J=1 TO NUMCOL;
      TEMP=RANK(J);
      OLDCNT(TEMP)=NEWCNT(J);
      IF FINALP THEN OLDFUNC(TEMP)=NEWFUNC(J);
      DO K=1 TO NEWCNT(J);
        OLDTAB(K,TEMP)=NEWTAB(K,J);
        END;  /*  END LOOP K  */
      END;  /*  END LOOP J  */
    FREE TABLEB;
    GO TO MULTAGAIN;

STOREP:
/*  THE P-FUNCTION FOR EITHER ONE OF THE OUTPUT FUNCTIONS OR THE
    ENTIRE FUNCTION (SIGNIFIED BY FINALP BEING TURNED ON) HAS BEEN
    GENERATED.  AT THIS POINT, NEWCOL=1.  IF THERE IS ONLY ONE OUTPUT
    FUNCTION, THEN THE P-FUNCTION FOR THE ENTIRE PROBLEM IS NOW IN
    TABLEB, AND THERE IS NO NEED TO STACK ANYTHING IN TABLEC.    */
    FREE TABLEA;
    IF NUMOUTPUTS=1 THEN DO;
      ALLOCATE FUNCTNS;
      SUBSTR(NEWFUNC(1),1,1)='1'B;
      GO TO ENDPFN;
      END;
    IF FINALP THEN GO TO ENDPFN;
    ALLOCATE TABLEC;
    PTRC=PTRC+1;  /*  COUNT NUMBER OF ALLOCATIONS OF TABLEC      */
```

```
    IF NEWCNT(1)=0 THEN GO TO SKIP;
    DO J=1 TO NEWCNT(1);
      CTAB(J)=NEWTAB(J,1);
      END;  /*  END LOOP J  */
SKIP:
    CCOUNT=NEWCNT(1);
    NUMROWS(PTRC)=MAXNUMA;
    IF MAXNUMA > MAXNUMC THEN MAXNUMC=MAXNUMA;  /*  KEEP TRACK OF
    MAXIMUM NUMBER OF ROWS OF TABLEC - THIS WILL BE USED LATER TO
    ALLOCATE TABLEB AFTER P-FUNCTIONS FOR THE INDIVIDUAL FUNCTIONS
    HAVE BEEN FOUND.                                                */
    FREE TABLEB;
ADDI:
    I=I+1;
    IF I <= NUMOUTPUTS THEN GO TO NEXTFN;
/*  WHEN THIS POINT IS REACHED, THE P-FUNCTIONS FOR EACH INDIVIDUAL
    OUTPUT FUNCTION ARE STACKED IN TABLEC, WITH THE P-FUNCTION FOR
    THE FIRST FUNCTION ON THE BOTTOM OF THE STACK.  THERE ARE PTRC
    ELEMENTS IN THIS STACK.  POP THE P-FUNCTION FOR EACH FUNCTION
    OUT OF THE STACK, AND PUT IT IN A COLUMN IN TABLEB, WHICH TABLE
    WILL HAVE A MAXIMUM OF MAXNUMC ROWS IN ANY COLUMN.  THEN SORT
    THE COLUMNS, MOVE THEM INTO TABLEA, AND GO THROUGH THE
    MULTIPLICATION PROCEDURE AGAIN.                                 */
    MAXNUMA=0;  MAXNUMB=MAXNUMC;
    NEWCOL=PTRC;
    ALLOCATE TABLEB;
    ICOL=NEWCOL+1;  /*  POINTER TO COLUMN OF TABLEB  */
POP:
    ICOL=ICOL-1;
    IF ICOL=0 THEN GO TO ENDPOP;
    IF CCOUNT > MAXNUMA THEN MAXNUMA=CCOUNT;
    NEWCNT(ICOL)=CCOUNT;  RANK(ICOL)=ICOL;
    DO J=1 TO CCOUNT;
      NEWTAB(J,ICOL)=CTAB(J);
      END;  /*  END LOOP J  */
    FREE TABLEC;
```

```
    GO TO POP;
ENDPOP:
/*  NOW READY TO SORT COLUMNS OF TABLEB AND MOVE INTO TABLEA          */
    FINALP='1'B;  /*  TURNED ON WHEN FINDING P-FUNCTION FOR          */
                  /*  THE ENTIRE PROBLEM                             */
    ALLOCATE FUNCTNS;
    OLDFUNC='0'B;  NEWFUNC='0'B;
    DO J=1 TO NEWCOL;
      SUBSTR(NEWFUNC(J),RANK(J),1)='1'B;
      END;  /*  END LOOP J  */
SQUISH:
    HOLDCOL=NEWCOL;
    DO J=NEWCOL TO 1 BY -1;
      IF NEWCNT(J)=0 THEN DO;
        /*  WANT TO MOVE OUT THE DUMMY COLUMN AND SHIFT THE OTHER
            COLUMNS OVER TO TAKE UP ITS PLACE.                        */
        DO K=J+1 TO HOLDCOL;
          NEWCNT(K-1)=NEWCNT(K);
          NEWFUNC(K-1)=NEWFUNC(K);
          DO L=1 TO NEWCNT(K);
            NEWTAB(L,K-1)=NEWTAB(L,K);
            END;  /*  END LOOP L  */
          END;  /*  END LOOP K  */
          HOLDCOL=HOLDCOL-1;
        END;
      END;  /*  END LOOP J  */
    NEWCOL=HOLDCOL;
/*  REASSIGN THE RANKS          */
    DO J=1 TO NEWCOL;
      RANK(J)=J;
    END;
    MAXNUMA=MAXNUMB;
    GO TO SRTMOVE;

ENDPFN:
/*  AT THIS POINT, THE P-FUNCTION HAS BEEN COMPUTED.  IF NEWCOL=1
```

```
    THEN THE ENTIRE FUNCTION HAS BEEN MINIMIZED WITH NC PARTITIONS.
    IF NEWCOL IS NOT EQUAL TO 1, THEN ONLY A NEAR-MINIMAL SOLUTION
    HAS BEEN OBTAINED DUE TO THE LIMITS OF AVAILABLE MEMORY.  IN
    THIS CASE, IT IS NECESSARY TO MINIMIZE THE FUNCTION IN NEWCOL
    PARTITIONS.  THE 1-BITS IN NEWFUNC INDICATE THE FUNCTIONS
    BELONGING TO EACH PARTITION.                                    */
    IF NEWCOL=1 THEN PUT FILE (SYSPRINT) EDIT ('THERE ARE ',NEWCNT(1),
    ' TERMS IN THE P-FUNCTION FOR THIS PROBLEM.') (SKIP(3),A,F(7),A);
    FREE STRINGS;
/*  ESTABLISH A TOP ROW INTO WHICH A COPY OF THE TOP ROW OF THE
    PRIME IMPLICANT TABLE IS MOVED.  AS COLUMNS ARE ELIMINATED BY
    CHOOSING A MEMBER WITHIN A TERM OF THE P-FUNCTION SOLUTION, THEN
    TURN THE FLAG BIT ON, INDICATING THE COLUMN HAS BEEN COVERED.   */
    ALLOCATE TEMPPI;
    ALLOCATE WHICHPI;
    DUPTAB='0'B;
    DO I=1 TO NEWCOL; /* PROCESS THRU EACH PARTITION OF P-FUNCTION  */
      ALLOCATE PFNCOST;  ALLOCATE USEDPI;
      PFNCOST=0.0;      USEDPI='0'B;      LOWCOST=99999.99;
      DO J=1 TO NEWCNT(I); /* PROCESS THRU EACH ELEMENT IN PARTITION*/
        NOWTAB='0'B;
        AGAIN='0'B;
        TEMPPI=PITABLE(0,*); /* MOVE IN TOP ROW OF FLAG BITS        */
        NUMLEFT=0;
LOOKAGAIN:
        DO K=1 TO NUMOUTPUTS; /* PROCESS THRU EACH OUTPUT FUNCTION  */
          /* DOES THE CURRENT PARTITION COVER THIS OUTPUT?          */
          IF ¬SUBSTR(NEWFUNC(I),K,1) THEN GO TO KEND;
          DO L=FIRSTCOL(K) TO LASTCOL(K); /* PROCESS COL. FOR FN. K */
            IF TEMPPI(L) THEN GO TO LEND; /* COL. ALREADY COVERED  */
            NUMCHOICE=0;
            DO M=1 TO ESSENROW; /* LOOK FOR A 1 IN THE COLUMN       */
                IF ¬PITABLE(M,L) THEN GO TO MEND;
        /*  IF THAT PRIME IMPLICANT CORRESPONDING TO ROW M IS THE
            ONLY ONE THAT OCCURS IN THE CURRENT P-FUNCTION TERM,
            THEN CHOOSE THAT PRIME IMPLICANT AND ELIMINATE ALL
```

```
            COLUMNS THAT IT COVERS.  IF THERE IS MORE THAN ONE PRIME
            IMPLICANT FOR THAT COLUMN, THEN MUST DECIDE LATER.      */
                IF SUBSTR(NEWTAB(J,I),M,1) THEN DO;
                  NUMCHOICE=NUMCHOICE+1;
                  IF NUMCHOICE=1 THEN PICHOICE=M;
                  END;
MEND:             END;  /*  END LOOP M  */
              IF NUMCHOICE=1 THEN DO; /* CHOOSE PI M TO COVER COL L */
                M=PICHOICE;
                DO N=FIRSTCOL(K) TO LASTCOL(K);
                  IF TEMPPI(N) THEN GO TO NEND;
                  IF PITABLE(M,N) THEN DO;
        /*    IF N=L THEN THE COLUMN IS THE ONE THAT WAS ORIGINALLY
              DETECTED TO CHOOSE.  IF N < L THEN IT IS A COLUMN THAT
              COULD NOT BE COVERED BEFORE, SO SUBTRACT 1 FROM THE
              COUNTER OF THESE COLUMNS (NUMLEFT).  IF N > L THEN IT IS
              A COLUMN THAT HAS NOT BEEN ENCOUNTERED YET, BUT GO AHEAD
              AND TURN ITS FLAG BIT ON SINCE IT IS COVERED BY
              PRIME IMPLICANT M.                                    */
                    TEMPPI(N)='1'B; /* SET FLAG BIT                */
                    IF N<L THEN NUMLEFT=NUMLEFT-1;
                    END;
NEND:             END;  /*  END LOOP N  */
        /*    IF PRIME IMPLICANT M HAS ALREADY BEEN USED IN THIS
              PARTICULAR TERM OF THE P-FUNCTION SOLUTION, THEN ADD A
              COST OF 1 TO THE COST OF IMPLEMENTING THIS SOLUTION.
              OTHERWISE, ADD THE COST OF COST(M).                   */
                IF USEDPI(M) THEN DO;
                  PFNCOST(J)=PFNCOST(J)+(1*INTERCOST);
                  GO TO PIUSED;
                  END;
                PFNCOST(J)=PFNCOST(J)+COST(M);
                USEDPI(M)='1'B;
PIUSED:
                NOWTAB(K,M)='1'B;
                END;  /*  END LOOP FOR NUMCHOICE=1                  */
```

```
              ELSE IF AGAIN THEN DO;
          /*  THERE IS MORE THAN ONE '1' IN COLUMN L OF PITABLE.  AT
              LEAST ONE OF THE PRIME IMPLICANTS MUST HAVE BEEN USED
              PREVIOUSLY.  LOOK AT THEM AND CHOOSE THE FIRST ONE THAT
              HAS ALREADY BEEN USED IN IMPLEMENTING ANOTHER FUNCTION. */
                  DO N=1 TO ESSENROW;
                    IF PITABLE(N,L) THEN IF USEDPI(N) THEN DO;
                      NOWTAB(K,N)='1'B;
                      NUMLEFT=NUMLEFT-1;
                      IF NUMLEFT ¬= 0 THEN GO TO LEND;
                      GO TO POSTK;
                      END;
                    END;  /*  END LOOP N  */
                    PUT FILE (SYSPRINT) EDIT ('ERROR - FELL THRU LOOP ',
                    ' - HAVE MORE THAN 1 X IN A COLUMN AND NONE HAS ',
                    'BEEN PREVIOUSLY USED','COLUMN ',L,' IS THE PROBLEM')
                    (SKIP,A,A,A,SKIP,A,F(3),A);
    STOP;
                    END;
                  ELSE NUMLEFT=NUMLEFT+1;
LEND:           END;  /*  END LOOP L  */
KEND:         END;  /*  END LOOP K  */
POSTK:
          IF NUMLEFT ¬= 0 THEN DO;
            AGAIN='1'B;  GO TO LOOKAGAIN;
            END;
          USEDPI='0'B;  IF PFNCOST(J) < LOWCOST THEN DO;
            LOWCOST=PFNCOST(J);  CHOICEPI=J;;
            /* MOVE IN THE SOLUTION ONLY FOR THE APPROPRIATE
               FUNCTIONS.  THIS IS DONE SO THAT IF THERE IS MORE THAN
               ONE PARTITION, THEN WHAT IS ALREADY STORED IN DUPTAB
               WILL NOT BE DESTROYED.                                  */
            DO NN=1 TO NUMOUTPUTS;
              IF SUBSTR(NEWFUNC(I),NN,1) THEN DUPTAB(NN,*)=NOWTAB(NN,*);
            END;  /*  END LOOP NN  */
            END;
```

```
JEND:
        END;  /*  END LOOP J  */
    IF NEWCOL ¬= 1 THEN DO;
      FREE PFNCOST;  FREE USEDPI;
      END;
IEND:
/*  BEFORE TAKING CARE OF THE NEXT PARTITION, IF THERE IS ONE,
    CHANGE THE COST TABLE TO REFLECT THE USAGE OF THE PRIME
    IMPLICANTS IN THE SOLUTION FOR THE CURRENT PARTITION I.          */
    IF I < NEWCOL THEN DO NN=1 TO ESSENROW;
      IF SUBSTR(NEWTAB(CHOICEPI,I),NN,1) THEN COST(NN)=1.0;
      END;  /*  END LOOP NN  */
      END;  /*  END LOOP I  */
    DO J=1 TO NUMOUTPUTS;
      DO K=1 TO ESSENROW;
        IF DUPTAB(J,K) THEN SUBSTR(MEMBER(K),J,1)='1'B;
        END;  /*  END LOOP K  */
      END;  /*  END LOOP J  */
    FREE TABLEB;
    FREE FUNCTNS;
    FREE WHICHPI;
    FREE TEMPPI;
END CYCLIC;
```

APPENDIX C

SAMPLE INPUTS AND OUTPUTS

EXAMPLE 1 FROM THIS THESIS

```
NUMBER OF INPUTS=  4

NUMBER OF OUTPUTS=  1

COST FACTOR FOR AND GATES IS  1.00

COST FACTOR FOR OR GATES IS  1.00

COST FACTOR FOR EACH FIRST LEVEL INPUT IS  1.00

COST FACTOR FOR EACH SECOND LEVEL INPUT IS  1.00

THE RESULT WILL BE MINIMIZED IN THE SUM-OF-PRODUCTS FORM.

                    INPUT TERMS AS READ IN

MINTERMS FOR FUNCTION   1:    0    1    2    3    4    5    10    11    13    15
PHITERMS FOR FUNCTION   1:    NONE

A TOTAL OF  26 IMPLICANTS HAVE BEEN GENERATED FOR THIS PROBLEM.
6 OF THESE ARE PRIME IMPLICANTS.
THE LARGEST NUMBER OF IMPLICANTS GENERATED AT ONE TIME IS  13.
THIS NUMBER SHOULD NEVER BE GREATER THAN 500.
```

PRIME IMPLICANT TABLE

	MINTERMS FUNCTION 1									
PRIME IMPLICANTS	0	1	2	3	4	5	10	11	13	15
B*C'*D						X			X	
A*C*D								X		X
A*B*D									X	X
A'*B'	X	X	X	X						
A'*C'	X	X			X	X				
B'*C			X	X			X	X		

PRIME IMPLICANT 5 IS ESSENTIAL TO FUNCTION 1
PRIME IMPLICANT 6 IS ESSENTIAL TO FUNCTION 1
ROW 1, DOMINATED BY OR EQUAL TO ROW 3, IS ELIMINATED
ROW 2, DOMINATED BY OR EQUAL TO ROW 3, IS ELIMINATED
PRIME IMPLICANT 3 IS ESSENTIAL TO FUNCTION 1

SOLUTION

FUNCTION 1: (A*B*D) + (A'*C') + (B'*C)

EXAMPLE 2 FROM THIS THESIS

```
NUMBER OF INPUTS=  5

NUMBER OF OUTPUTS=  1

COST FACTOR FOR AND GATES IS  1.00

COST FACTOR FOR OR GATES IS  1.00

COST FACTOR FOR EACH FIRST LEVEL INPUT IS  1.00

COST FACTOR FOR EACH SECOND LEVEL INPUT IS  1.00

THE RESULT WILL BE MINIMIZED IN THE SUM-OF-PRODUCTS FORM.

                INPUT TERMS AS READ IN

MINTERMS FOR FUNCTION   1:    1    4    7    14    17    20    21    22    23
PHITERMS FOR FUNCTION   1:    0    3    6    19    30

A TOTAL OF  42 IMPLICANTS HAVE BEEN GENERATED FOR THIS PROBLEM.
9 OF THESE ARE PRIME IMPLICANTS.
THE LARGEST NUMBER OF IMPLICANTS GENERATED AT ONE TIME IS  21.
THIS NUMBER SHOULD NEVER BE GREATER THAN 500.
```

PRIME IMPLICANT TABLE

	MINTERMS FUNCTION 1								
PRIME IMPLICANTS	1	4	7	14	17	20	21	22	23
A'*B'*C'*D'	X								
A'*B'*D'*E'		X							
B'*C'*E	X				X				
B'*C*E'		X				X		X	
B'*D*E			X						X
B'*C*D			X					X	X
C*D*E'				X				X	
A*B'*E					X		X		X
A*B'*C						X	X	X	X

PRIME IMPLICANT 7 IS ESSENTIAL TO FUNCTION 1
COLUMN 9 WHICH DOMINATES COLUMN 3 WILL BE REMOVED FROM FUNCTION 1
ROW 1, DOMINATED BY OR EQUAL TO ROW 3, IS ELIMINATED
ROW 2, DOMINATED BY OR EQUAL TO ROW 4, IS ELIMINATED
ROW 5, DOMINATED BY OR EQUAL TO ROW 6, IS ELIMINATED
PRIME IMPLICANT 3 IS ESSENTIAL TO FUNCTION 1
PRIME IMPLICANT 4 IS ESSENTIAL TO FUNCTION 1
PRIME IMPLICANT 6 IS ESSENTIAL TO FUNCTION 1
ROW 8, DOMINATED BY OR EQUAL TO ROW 9, IS ELIMINATED
PRIME IMPLICANT 9 IS ESSENTIAL TO FUNCTION 1

SOLUTION

FUNCTION 1: (B'*C'*E) + (B'*C*E') + (B'*C*D) + (C*D*E') + (A*B'*C)

EXAMPLE 3 FROM THIS THESIS

```
NUMBER OF INPUTS=  4

NUMBER OF OUTPUTS=  3

COST FACTOR FOR AND GATES IS  1.00

COST FACTOR FOR OR GATES IS  1.00

COST FACTOR FOR EACH FIRST LEVEL INPUT IS  1.00

COST FACTOR FOR EACH SECOND LEVEL INPUT IS  1.00

THE RESULT WILL BE MINIMIZED IN THE SUM-OF-PRODUCTS FORM.

                INPUT TERMS AS READ IN

MINTERMS FOR FUNCTION   1:    2    4    10    11    12    13
PHITERMS FOR FUNCTION   1:    5

MINTERMS FOR FUNCTION   2:    4    5    10    11    13
PHITERMS FOR FUNCTION   2:    NONE

MINTERMS FOR FUNCTION   3:    1    2     3    10    11    12
PHITERMS FOR FUNCTION   3:   13   15

A TOTAL OF  23 IMPLICANTS HAVE BEEN GENERATED FOR THIS PROBLEM.
11 OF THESE ARE PRIME IMPLICANTS.
THE LARGEST NUMBER OF IMPLICANTS GENERATED AT ONE TIME IS  11.
THIS NUMBER SHOULD NEVER BE GREATER THAN 500.
```

PRIME IMPLICANT TABLE

	MINTERMS																	
	FUNCTION 1						FUNCTION 2					FUNCTION 3						
PRIME IMPLICANTS	2	4	10	11	12	13	4	5	10	11	13	1	2	3	10	11	12	
A*B*C'*D						X					X							
A'*B'*D												X		X				
B'*C*D'	X		X										X		X			
A'*B*C'		X					X	X										
B*C'*D						X		X			X							
A*B'*C			X	X					X	X					X	X		
A*B*C'					X	X											X	
A*C*D																X		
A*B*D																		
B'*C													X	X	X	X		
B*C'		X			X	X												

PRIME IMPLICANT 3 IS ESSENTIAL TO FUNCTION 1
PRIME IMPLICANT 6 IS ESSENTIAL TO FUNCTION 1
PRIME IMPLICANT 4 IS ESSENTIAL TO FUNCTION 2
PRIME IMPLICANT 6 IS ESSENTIAL TO FUNCTION 2
PRIME IMPLICANT 2 IS ESSENTIAL TO FUNCTION 3
PRIME IMPLICANT 7 IS ESSENTIAL TO FUNCTION 3
COLUMN 6 WHICH DOMINATES COLUMN 5 WILL BE REMOVED FROM FUNCTION 1
COLUMN 15 WHICH DOMINATES COLUMN 13 WILL BE REMOVED FROM FUNCTION 3
ROW 1, DOMINATED BY OR EQUAL TO ROW 5, IS ELIMINATED
ROW 8, DOMINATED BY OR EQUAL TO ROW 6, IS ELIMINATED
PRIME IMPLICANT 5 IS ESSENTIAL TO FUNCTION 2

THERE ARE 4 TERMS IN THE P-FUNCTION FOR THIS PROBLEM.

SOLUTION

FUNCTION 1: (B'*C*D') + (A'*B*C') + (A*B'*C) + (A*B*C')

FUNCTION 2: (A'*B*C') + (B*C'*D) + (A*B'*C)

FUNCTION 3: (A'*B'*D) + (B'*C*D') + (A*B'*C) + (A*B*C')

B'*C*D'	IS USED IN FUNCTIONS	1	3	
A'*B*C'	IS USED IN FUNCTIONS	1	2	
A*B'*C	IS USED IN FUNCTIONS	1	2	3
A*B*C'	IS USED IN FUNCTIONS	1	3	

APPENDIX D

CONTROL CARD SEQUENCE FOR PLACING LOAD MODULE ON DISK

```
//THESIS  JOB (27401,33188,T-2),CANTWELL,MSGLEVEL=(1,1)
//STEP1   EXEC PL1FCL,LKEDOPT='OVLY,LET'
//PL1.SYSIN DD  *

       source deck

//LKED.SYSLMOD  DD  DSN=J27401.P33188.THESIS,
//    DISP=(NEW,KEEP),UNIT=2314,VOL=SER=ONLIN1,
//    SPACE=(TRK,(68,10),RLSE)
//LKED.SYSIN  DD *
  ENTRY  IHENTRY
  OVERLAY  ALPHA
  INSERT  PREPRO
  OVERLAY  ALPHA
  INSERT  CONVERT
  OVERLAY  ALPHA
  INSERT  MERGE
  OVERLAY  ALPHA
  INSERT  COVERIT
  OVERLAY  ALPHA
  INSERT  CYCLIC
  NAME  SWITCH(R)
```

CONTROL CARD SEQUENCE FOR EXECUTING PROGRAM FROM LOAD MODULE

```
//THESIS JOB (27401,33188,T-2,R-128K),CANTWELL
//STEP1  EXEC   PGM=SWITCH,REGION=128K
//STEPLIB  DD   DSN=J27401.P33188.THESIS,UNIT=2314,
//     VOL=SER=ONLIN1,DISP=SHR
//SYSPRINT  DD  SYSOUT=A
//SYSIN  DD *

       data cards
```

VITA

Loreene Knoph Cantwell was born in Knoxville, Tennessee, on December 13, 1949. She attended elementary school in that city and was graduated from the Webb School of Knoxville in 1967. She received a Bachelor of Science degree in Mathematics from The University of Kentucky in May 1971.

She entered the Graduate School at The University of Tennessee in September 1971. In the fall of 1972, she was employed by the Computer Assisted Registration System and received the Master of Science degree with a major in Computer Science in March 1974. She is currently residing in Morristown, Tennessee, where her husband is associated with a law firm.