Automatic Design of Switching Networks

Darryl Dhein

AUTOMATIC DESIGN OF SWITCHING NETWORKS

by

Darryl D. Dhein

A Thesis Submitted

in

Partial Fulfillment

of the

Requirements for the Degree of

MASTER OF SCIENCE

in

Electrical Engineering

Approved by:

Prof. Swaminathan Madhu
S. Madhu

Prof. George A. Brown
G. Brown

Prof. George L. Thompson
G. Thompson

Prof. W. F. Walker
W. F. Walker

DEPARTMENT OF ELECTRICAL ENGINEERING
COLLEGE OF APPLIED SCIENCE
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK
JANUARY 1972

692309

ABSTRACT

This thesis develops a method for automatically selecting an optimum set of prime implicants of a Boolean function. The optimization algorithm is based on a minimum cost of mechanization of the simplified function. A FORTRAN IV computer program to implement this approach was written amd is included as part of this thesis. This program was developed within the framework of an overall theory for the automation of the design of switching networks. A programing structure as well as the theory for the automation of design is given. Also included is an outline of further areas of study which would be worth exploring as an extension of the present work.

TABLE OF CONTENTS

LIST OF TABLES

-LIST OF FIGURES

LIST OF FIGURES (Cont.)

LIST OF SYMBOLS

Symbol	General Usage
$\overline{X}$	A bar over a logical variable denotes negation
$+$	Is used to denote logic addition (Union of Boolean variables)
XY	Adjacency of logic variables denotes logical multiplication (intersection of Boolean variables)
	Special Program Subroutines
DATAEN	Data Entry Subroutine
SORT	High Speed Sorting Subroutine for DATAEN
CONV11	Format Conversion Subroutine for SORT
PRIMEI	Prime Implicant Determination Subroutine
ESSPI	Essential Prime Implicant Determination Subroutine
CONV12	Format Conversion Subroutine for ESSPI
FORMPI	Reformat, Weight and Order Prime Implicants
CONV13	One Word Format Conversion Subroutine for FORMPI
CONV23	Multiword Format Conversion Subroutine for FORMPI
OPTMPI	Determination and Optimization of Solution Subroutine
CONV1	One Word Format Conversion Subroutine for OPTMPI
CONV2	Multiword Format Conversion Subroutine for OPTMPI

CHAPTER 1. INTRODUCTION

1.1 Thesis Definition

This thesis develops a detailed approach to the problem of optimum selection of a set of prime implicants for minimization of switching functions. The algorithm developed allows considering a minimum cost optimization with provision for non-uniform weighting of the prime implicants and the inclusion of multiple output functions. The weighting used for the prime implicants is a cost based on the number of logic gate inputs required to mechanize the function. The computer program used to accomplish this was structured to be part of a continuing development in other areas of automatic design of switching networks. In Chapter Three an outline of the areas recommended for further development are presented.

The model used in this thesis and one which has been extensively used in logic design is the two level AND-OR logic model with a uniform cost per input for either of the gate types. This model was highly developed for its ease in solution and because it very closely represented the true design restrictions for some time. This was the time when discrete elements were used for the logic (i.e. gates were made up of individual diodes). During this earlier period, expensive amplifiers, composed of a number of discrete components, had to be inserted after every other state of passive circuitry to maintain wave shape if high speeds (by prevalent standards) were to be maintained with any reliability. With the advent of integrated circuits, the practical limitations of two levels has been virtually eliminated. Also, there is at present a greater variety of gates available which in most cases provide a cost savings over exclusive use of AND-OR gates. Another factor which affects logic design is that memory units, or flip flops, used to be many times the cost of a simple gate and therefore the procedure was to minimize the memory states to an absolute minimum independent of the gate structure and then minimize the gates. However, with modern integrated circuits a flip-flop including some built-in gating or other complex functions may be purchased for a price comparable with a few individual gates. For this reason designs of non-minimum states are sometimes less expensive because of an associated simpler gating requirement.

The two level AND-OR gating structure, however, still has the real advantage of being one of the most natural and easiest to understand and work with on a manual design basis. For the same reasons it is best adapted to teaching switching theory. Additionally, there are well developed and relatively fool-proof minimization procedures for this model. These procedures include mapping and the method known as the Quine-McCluskey method. A historical review of the developments in this area are given in the next section.

1.2 Historical Review

The starting point for most of the early work in switching networks was the Algebra of Classes set up as a formal deductive system (Boolean Algebra). Many alternate postulate sets have been proposed. One which was well developed is attributed(1) to E. V. Huntington in an article published in 1904 "Sets of Independent Postulates for the Algebra of Logic."(2) The algebra itself was named after George Boole who published two papers on it; one in 1848 and another in 1853.(3 & 4) A major development of the application of this algebra to switching circuits has been attributed(5) to C. E. Shannon for his paper on "A Symbolic Analysis of Relay Switching Circuits"(6) which was published in 1938. The postulates of this development were shown to be derivable from a subset of the calculus of propositions which in turn was developed from the algebra originated by George Boole.

Later, Shannon developed his ideas further and published a paper "The Synthesis of Two Terminal Switching Circuits" in 1949.(7) In 1951 a chart or tabular method was published for simplification of Boolean functions. This method became known as the Harvard Method.(8) This was followed by a systematic algebraic method for simplification of Boolean functions by W. V. Quine in 1952 and later improved upon.(9 & 10)

While the postulates of Boolean algebra in a mathematical sense were presented over a hundred years ago and well formulated sixty-five years ago, it is still the basis for virtually all works in switching theory and is included as a starting point for almost every text on the subject. The method which forms the basis for the two level AND-OR minimization section of this thesis was presented by E. J. McCluskey as his doctoral thesis in Electrical Engineering at Massachusetts Institute of Technology in June 1956.(11 & 12) This work was an improvement on Quine's earlier work and the method has come to be known as the Quine-McCluskey method; it is now considered the classical approach to the problem of two level AND-OR simplification through the use of Boolean Algebra. The key equation on which this method is based is given below as equation 1.

(1) $XY + X\overline{Y} = X$

Basically the method consists of first expanding all terms to a sum of terms of their lowest level "minterms" and then systematically using equation 1 to simplify the result.

Since these early developments, there have been a number of papers on the subject of optimizing the selection of the prime implicants developed by the Quine-McCluskey method. As noted by F. Luccio, these include two later papers by I. B. Pyne and E. J. McCluskey published in 1961 and 1962;(13 & 14) also, two papers by J. F. Gimpel, one in 1964 and the other in 1965(15 & 16) and Luccio's paper in 1966.(17)

The advantages of the method presented in this thesis include the fact that certain large problems, including variable cost of the different prime implicants and multiple outputs may be solved by relatively straightforward methods yielding the optimum or near optimum solution. The optimization algorithm developed for this thesis can be set to give the absolute optimum solution by use of a method of testing all solutions for minimum cost. For small size problems this would be provided automatically. For problems of any significant size the all combination approach becomes less desirable from the standpoint of computer time used. The increase in required computer time is very rapid as the number of nonessential prime implicants is increased, being similar to a factorial type of function. The program is currently written to consider all combinations of solution for a maximum of ten nonessential prime implicants. For sizes above this the weighting algorithm selects the combinations to be considered. The final solution printed is the best solution upon completion of the extent of analysis specified by the user.

There are also graphical methods to solve the two level AND-OR minimization problem. The method in common use was published by E. W. Veitch(18) in its basic form and later in the currently more popular improved form by M. Karnaugh.(19) These graphical methods tend to replace well defined routines with visual insight and are therefore not as directly applicable to automatic solution by a digital computer.

1.3 Scope of Thesis

This thesis develops an algorithm for the optimum selection of prime implicants of a Boolean function. The optimization algorithm is based on a minimum cost of mechanization of the simplified function. The results of a number of sample problems are discussed, giving the strong features and limitations of the approach. This subject matter is covered in Chapter Two. Chapter Three presents an outline of other areas recommended for future development. Chapter Four discusses the conclusions derived from the present investigation. The program presented was developed for this thesis as an original program. Appendix I provides a flow chart of the program and Appendix II provides a detailed computer listing of the program.

CHAPTER 2. OPTIMUM SELECTION OF PRIME IMPLICANTS OF BOOLEAN FUNCTIONS

The method used in selection of the prime implicants is given below. This is followed by a description of the program used in solving the AND-OR combinational logic problem with uniform cost per input. The flow charts for the program are included in Appendix I.

2.1 Optimized Prime Implicant Selection Method

The method used is the Quine-McCluskey method with an additional algorithm for optimized selection of non-essential prime implicants and special features to match the RIT 360 computer configuration. A number of provisions are incorporated for ease and naturalness of job entry. Details of the program and its use are described in section 2.3. A number of sample problems and their results are given in section 2.4.

The prime implicants are first determined by the Quine-McCluskey method as described in Cadwell.(5) After determination of the prime implicants, the essential prime implicants are selected. Essential prime implicants are ones which are required because they are the only ones that contain a particular minterm. The optimum (minimum cost) set of the remaining prime implicants necessary to specify the required function is then selected. This is accomplished by weighting the prime implicants in roughly the order of their probability of being included in an optimum solution. The most probable are then considered first in a search for solutions which continues until a user defined number of correct solutions has been achieved by the computer. The best is then printed as the required solution. The user may specify the number of prime implicants to be considered in combination and the weighting factor to be used for the prime implicant ordering.

2.2 Special Program Features

There are incorporated in the program a number of features including a storage saving techniqûe for FORTRAN programs using octal coding of logical data. In BASIC FORTRAN $\overline{\underline{\text{IV}}}$ which is used on the RIT 360 computer four bytes of information are required to store the state of a variable as 0 or 1. Four bytes is one computer word. Even in the full FORTRAN $\overline{\underline{\text{IV}}}$ employing logical variables one byte is required for the storage of the equivalent information. By using the integer format and coding the information in octal, the program used stores the state of up to eighteen literals, plus some additional information, in one word. This saves memory and allows a higher theoretical limit on the size of problems to be run. A description of the program's data input routine which includes the above encoding method is given below in section 2.3.1.

2.3 Program Description

The program is broken down into a number of functional areas. The first is the program entry section. In this section the basic information which has to be entered into the computer and the method used to encode it is described. In the next section the prime implicant development is presented, and the final section describes the method used in making an optimum selection of the prime implicants.

2.3.1 Data Entry

The program is described starting with the data entry. The first deck of cards is the computer system cards and the program deck which are provided the user as a package. Next come the data cards which are described in order of entry as follows:

Table 1

1st Data Card Entries

Column	Entry
1	Blank if only one problem is to be run or if this is the last problem. A 1 is entered if another problem is to be run
2-5	Machine Type Specification; Enter a 1 in column 5 for a combinational logic design problem.

Note: All columns not indicated should be left blank. All entries must be right justified in columns indicated. These notes apply to all card entries.

Table 2

2nd Data Card Entries

Column	Entry
1-5	No. of literals used per minterm (i.e. $A\bar{B}C\bar{D}\bar{E}\bar{F}$ contains six literals). A maximum of eighteen may be specified.
6-10	No. of outputs in the problem. A maximum of six are allowed. (i.e. a number 1-6 must be entered in column 10).
11-13	Output Definition: Enter a 1 in each of the columns associated with a desired output. Column 11 Full development of prime implicants. 12 Listing of prime implicants. 13 Listing of essential prime implicants.

The optimized prime implicant selection, the number of gate input lines and a listing of the input is provided automatically.

The third and succeeding data cards define the logic to be simplified. Provision is made for entering optional ("don't care") as well as required terms. Also, a multiplicity of input terms may be entered by a single statement. This is accomplished by leaving literals blank when all combinations of the literal are to be entered (i.e. $Abb\overline{D}$ enters $A\overline{BCD}$, $A\overline{B}C\overline{D}$, $AB\overline{CD}$ and $ABC\overline{D}$). When a term is to be specified for more than one output, all or any subset of the outputs may be specified on one card. Remaining outputs would be specified on additional cards as desired. The format for card three and all remaining cards is as follows:

Table 3

3rd Data Card Entries

Column	Entry
1	Enter a 1 if another card follows. Leave Column one blank if this is the last card of data set three.
2	Column two is left blank for clarity in reading the printed data on the punched card.
3	Leave blank if this is a required term. Enter a minus sign if it is an optional term.
Next N columns	For each literal enter a 1 if it is the true form, a 2 if in the negated form and a 3 if blank. Note: N is the number entered in Columns 1-5 of Card two.
Next column	Leave blank.
Next M columns	Enter the numbers of the outputs associated with this term. Note: M is the number entered in Column ten of Card two. If only one output is used it need not be indicated (i.e. if M is 1, these columns would be left blank as an optional entry).

As an example, if $A\overline{BC}D\overline{E}$ was a required term for outputs two and three, the card format would be "1bb12212b23." The first 1 denotes another card is to follow.

As the input is read in, the first card causes the

AND-OR logic simplification routine to be entered. The second card sets up the indices used in reading the succeeding data cards. Each succeeding data card is read into a one card buffer. This input is then reduced to one number (computer word) per minterm. These numbers are generated by entering the octal equivalent of each literal, a literal at a time, into a tempqrary buffer. Considering the part of the input denoting the literals, if the ith literal is 1 (a true valued literal) the octal value of $2^{(i-1)}$ is added to each number in the temporary storage. If it is a 2 (a negated literal) nothing is added. If it is a 3 (an all combinations specification) a new number is created for each number already in storage which is that number plus the octal value of $2^{(i-1)}$. The sign of the number(s) is plus for a required term and minus for an optional term. The number of ones in the literal of each term is entered as the two most significant digits. The octal equivalent of the sum of the weighted output numbers is the least significant two digits. Each output is weighted as zero if not applicable and as $2^{(n-1)}$ if applicable, where n is the output number. The resulting integer has the following structure:

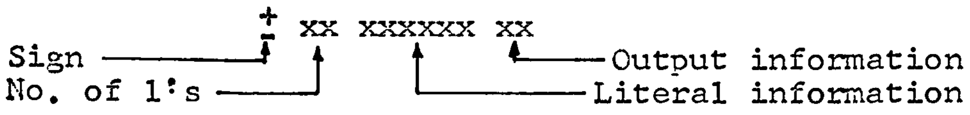


Figure 1
Word Format

The temporary buffer is overlapped on the upper 512 words of the main buffer allowing a maximum of nine blanks to be inserted in a term. After each input card is processed all the resulting minterms in temporary storage are transferred to the main storage. If there are more than one thousand minterms, storage buffers would normally be exceeded during problem solution; therefore the solution is terminated at the input phase in this case.

Upon completion of reading the problem description the main register is sorted in order of the number of literals in the true state for each minterm. Those with the least number are entered first. A standard sort approach would be to scan the register, select the least value, put it in the next position of a second buffer until all values were in ascending order. For n terms in the register there would be required a number of comparisons equal to the combinations of n terms taken two at a time, or $\frac{n!}{2(n-2)!} = \frac{1}{2}n(n-1)$ comparisons would be required. To improve the speed, a high speed binary sort is used which requires a maximum of $ni - \frac{n}{2}$ comparisons where "i" is the smallest integer for which $2^i \geq n$. For a hundred minterms the respective number of comparisons required for the two approaches would be 4,950 and 650 respectively. The ratio between the two methods would increase for a greater number of minterms and decrease for a smaller number. While an indication of the relative ratio of computer time involved, this ratio is not a true ratio of computer speed due to the fact the second approach does require more indexing and memory transfers per comparison. To save time in computing the number of ones in a minterm on each comparison, the storage number as described above is sorted directly in ascending order. The two most significant digits of this number contain the number of ones in the minterm and therefore when sorted in order provide the required ordering except for sign. One final ordering is then required to interpose the negative numbers within the positive numbers.

2.3.2 Prime Implicant Development

The ordered group of minterms resulting from the completed sort is denoted the first or starting level of the reduction. This level is divided into blocks containing a common number of ones in their minterms. By noting the position in the above ordering where the number composed of the first two digits changes value, the blocks are determined. The locations are saved at the upper end of the main register as pointers to the block changes. Each term is then compared with all terms of the next higher block. Those differing by a binary number are entered in the next level. Where two numbers differ by a binary number 2^{i-1} the literals in the ith position can be reduced by the relation $XI + X\bar{I} = X$ where X represents all literals other than the ith and I represents the ith. The numeric value of X is entered in the block of the next level. Where not all of the outputs are common between the two terms XI and $X\bar{I}$, only the common outputs are entered in the two least significant positions of the number denoting X in the next level. If all outputs match, both terms XI and $X\bar{I}$ in the current level are flagged. For level two through six a second integer number is associated with each reduced set of minterms. This number is denoted a tag and is divided into five 2 digit partitions in which the literal that was removed at each level is stored. If there are more than six levels in the reduction, additional tag words are added as required. In making comparisons for entry into levels three and up, the tags must be the same in addition to the entries differing by a binary number.

It may be noted that this requirement assures the previously removed literals are identical as a requirement of the comparison (i.e. that the X in XI and $X\bar{I}$ are the same).

After all possible reductions are made, the full development of the reduction process is printed if requested. Storage is then compressed by removing all flagged entries except those of the first level. The nonflagged entries are the prime implicants and are printed if requested by the user. For an optimum selection of prime implicants each minterm is scanned. If a minterm is contained in only one prime implicant with a common output, that prime implicant is flagged as an essential prime implicant. Also all the required minterms included in any essential prime implicants are flagged for all common outputs.

2.3.3 Optimum Prime Implicant Selection

In the next step all the minterms flagged on each of their outputs are deleted from storage. If there are no remaining minterms the essential prime implicants are printed as the final solution. If there are remaining minterms all essential prime implicants are grouped in a separate section of storage. The remaining prime implicants are assigned a weighting of one for each output of each of the remaining minterms which it contains plus an additional weight of four if the minterm for that output is contained in only one other prime implicant. This weighting has a tendency to indicate the relative probability that a prime implicant would be included in an optimum solution. The four weight may be optionally assigned a value other than four by the user. The prime implicants are then sorted in order of this weighting with the highest weighted entered first. Each of the prime implicants is then tested one at a time to see if they include all the remaining minterms. If there is one or more, the one requiring the least number of gate inputs is selected as the optimum. If not, all combinations of the prime implicants taken two at a time are tested to see if the remaining minterms are included in the other. Assuming thirty remaining prime implicants, 435 pairs would have to be considered and each pair tested to see if it contained all of the prime implicants. With the procedure used, the computer time has been reduced by effectively making the 435 scans of the remaining minterms changing a single prime implicant at a time rather than a pair of prime implicants. However, the consideration of more minterms in combination would generally not be practical from the standpoint of computer time. Therefore, only the first thirty are considered two at a time. The maximum number of prime implicants considered three at a time is fifteen; four at a time is twelve; five, six, seven, eight, nine, or ten at a time is ten. After a solution has been achieved each solution is weighted: one for each literal in each prime implicant (equivalent of one AND gate input) and one for each output it is used in (equivalent of one OR gate input). This solution is compared against any previous solution and the solution with the minimum number of gate inputs (minimum weighting) is selected and saved. If twenty five or more solutions have been achieved the best is printed as the optimum solution. If less than twenty five solutions have been achieved the first prime implicant is selected as a required prime implicant. It is then treated as an essential prime implicant and the process repeated. If there are ten or less prime implicants the absolute best solution is guaranteed, as all possible combinations would have been considered. The 25 solution rule applies after the specified combinations are done.

To enable use of this algorithm in varying situations, optional entries for the number of solutions and number of items to be considered at a time may be entered on Card 2 as follows:

Table 4

Additional Data Card 2 Entries

Column	Entry	
16-20	The number of solutions to be sought (25 is the default option if left blank). Allowable maximum is 99.	
	The maximum number of prime implicants for which all combinations are taken X at a time.	
	X	Default option
21-25	2	30
26-30	3	15
31-35	4	12
36-40	5	10
41-45	6	10
46-50	7	10
51-55	8	10
56-60	9	10
61-65	10	10
66-70	Weight factor for prime implicants	4

Note: entries must be right justified.

The weighting function for ordering of the prime implicants may be varied from the standard. The extra weight for prime implicants where only two include a minterm may be changed to any value 0-99 by entering the value in columns 66-70 on Card 2. The default option is four. If any of the options of Table 4 are used, all must be specified even if they are the same as the default option.

Additional work with this algorithm showed the initial estimates used for the standard numbers of combinations that could be practicably tested were overly optimistic; therefore, standard conditions should be used only for short problems. Some time indications and special cases are given at the end of Section 2.4 "Program Results".

There are several special means to request specific job functions by changing the number of solutions. For large problems that would require too much computer time, the user may specify a negative number of solutions. This will enable the user to receive the prime implicant development, prime implicant listing and essential prime implicant listing. It would allow an orderly progression to the next problem and use the minimum amount of computer time rather than simply putting a time limit on the job. The number zero should not be specified for the number of solutions. Any number of solutions less than ten limits the search at the first set of combinations of prime implicants from all prime implicants to one more than is specified for the second set (Columns 21-25, Card 2) as shown in Table 4.

The next section gives the results of a number of sample problems programed and a detailed example of the method used.

2.4 Program Results

Methods used and the results achieved are illustrated through the use of eight sample problems. These are described and actual output illustrated in the following sections.

2.4.1 Problem 1

Problem 1 is a basic problem which illustrates the problem specification, type of results provided by the program, encoding methods used and the problem solution method. The problem is stated as follows:

Find the optimum AND-OR mechanization for

(2) $A = X_1\bar{X}_2\bar{X}_3X_4X_5 + X_1\bar{X}_3\bar{X}_5 + X_3X_5$

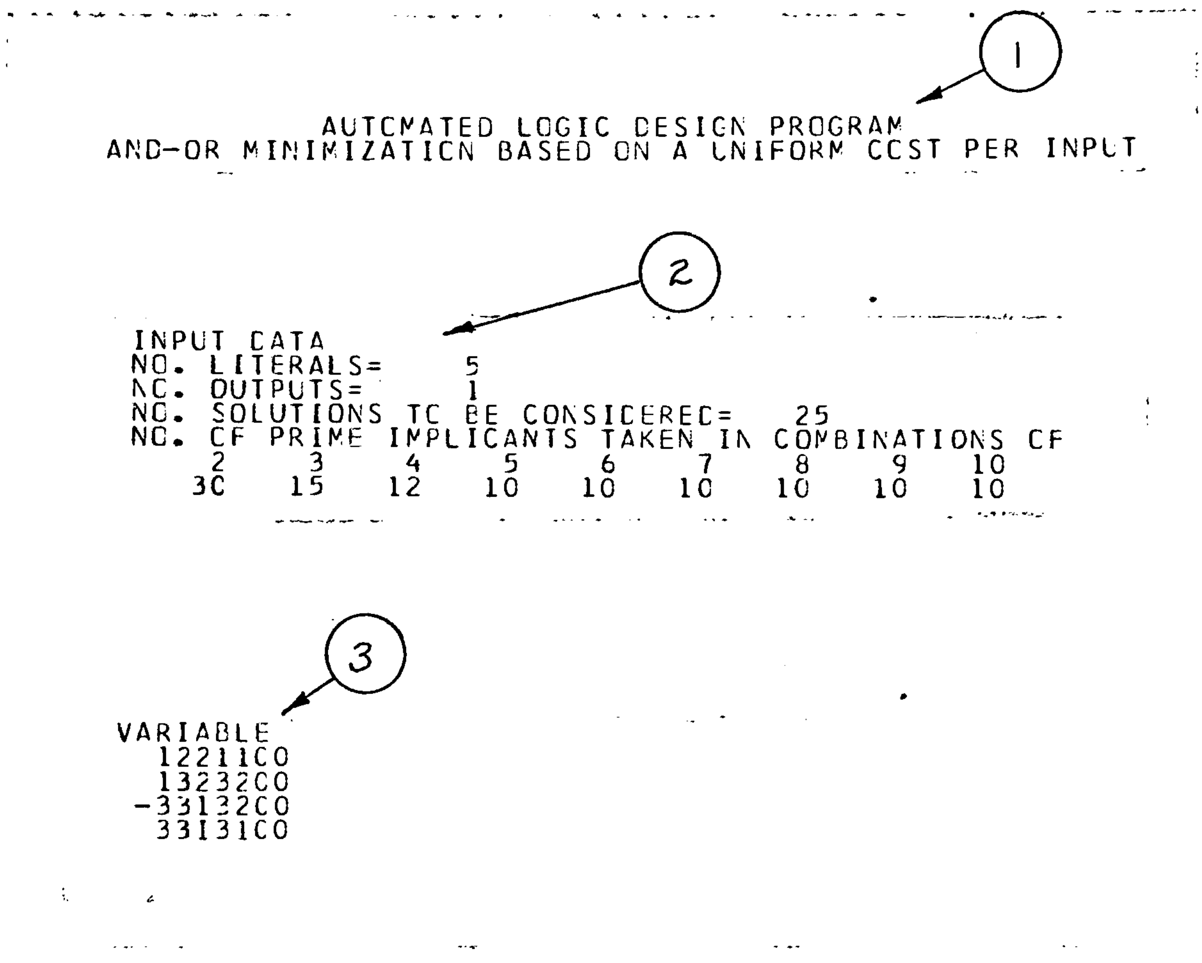
with the added provision the condition $X_3\bar{X}_5$ can not occur (i.e. $X_3\bar{X}_5$ is an optional term).

The mechanization for A as stated in Equation 2 would require a five input AND gate to form the first term, a three input AND gate for the second term and a two input AND gate for the third. All would then be OR connected with a three input OR gate to form A. As may be seen from the above example one AND gate input is required for each variable in a term and one OR gate for each term. The total number of inputs is thirteen. The object of the analysis is to reduce the mechanization cost by reducing the number of inputs required. With this relatively simple example a reduction could be effected through the use of Boolean Algebra. However, with this approach it is generally difficult to achieve an optimum solution or to know how near optimum the solution is. This first problem illustrates the Quine-McCluskey method as a systematic approach to finding a solution.

The input data for an automatic analysis of this problem has a one entered in column five of the first card. This specifies the problem type. Item one of Figure 2 shows the computer acknowledgment of this specification.

The input data for the second card includes the number of literals (five) entered in column five and the number of outputs (one) entered in column ten. Ones were entered in columns eleven, twelve, and thirteen to acquire a full set of computer output. The number of solutions to be considered before selecting the best and the number of terms to be considered in combination were not specified. The program therefore automatically selected the default options. This is shown as item two of Figure 2.

The data of Equation 2 is specified to the computer program for each of the terms as shown in Table 5. A one



```
              AUTOMATED LOGIC DESIGN PROGRAM
  AND-OR MINIMIZATION BASED ON A UNIFORM COST PER INPUT


   INPUT DATA
   NO. LITERALS=      5
   NO. OUTPUTS=       1
   NO. SOLUTIONS TO BE CONSIDERED=    25
   NO. OF PRIME IMPLICANTS TAKEN IN COMBINATIONS OF
        2     3     4     5     6     7     8     9    10
       30    15    12    10    10    10    10    10    10


  VARIABLE
    1221100
    1323200
   -3313200
    3313100
```

Figure 2

Problem 1 Specification

is entered for each literal in the true state, a two for a literal in the false (negated) state and a three for a literal that is absent (optional).

Table 5

Input Variables Problem 1

Status of Term	Term	Entry Card Column 1	2	3	4	5	6	7
Required Term	$X_1\bar{X}_2\bar{X}_3X_4X_5$	1		1	2	2	1	1
Required Term	$X_1\bar{X}_3\bar{X}_5$	1		1	3	2	3	2
Optional Term	$X_3\bar{X}_5$	1	-	3	3	1	3	2
Required Term	X_3X_5			3	3	1	3	1

It may be noted the optional (negative) term could have been omitted and a logically correct expression would have resulted; however, this type of term is used by the program to enable a reduction where possible but excluded where additional hardware would be required for its inclusion. It is thus used to advantage in simplifying the hardware mechanization. The ones in column one denote another entry follows. The blank in column one of the last card denotes the last entry. The minus sign in column two denotes the optional entry. As is seen, the equation and optional terms may be entered in any order. The number of output lines is equal to the number of equations. Item three of Figure 2 shows acknowledgement of the data entry for the one equation. As only one equation was used the output or equation number was not entered. This is shown by the last two digits being zero for each entry. The program as encoded has provision for a maximum of six outputs which may be optimized simultaneously. Most automated methods published are limited to optimizing the equations one at a time and do not mechanize for an overall minimal hardware solution with maximum effective sharing of components. The program will select a nonminimum solution for any equation if it can more than offset the difference in hardware cost with a saving in the hardware used for another equation, another output network, or group of equations by sharing components.

The first step in the optimization is to expand the terms of Equation 2 into their minterms (primary terms). This is accomplished through repeated application of the Boolean Algebra identity of equation.

(3) $X = XA + X\overline{A}$

This identity is used until all literals are present for each term. This is what is called a minterm. The first term $X_1\overline{X}_2\overline{X}_3X_4X_5$ is already in this format. The second term is expanded as follows:

(4) $X_1\overline{X}_3\overline{X}_5 = X_1X_2\overline{X}_3\overline{X}_5 + X_1\overline{X}_2\overline{X}_3\overline{X}_5$

(5) $X_1X_2\overline{X}_3\overline{X}_5 = X_1X_2\overline{X}_3X_4\overline{X}_5 + X_1X_2\overline{X}_3\overline{X}_4\overline{X}_5$

(6) $X_1\overline{X}_2\overline{X}_3\overline{X}_5 = X_1\overline{X}_2\overline{X}_3X_4\overline{X}_5 + X_1\overline{X}_2\overline{X}_3\overline{X}_4\overline{X}_5$

The two remaining input terms $X_3\overline{X}_5$ (optional term) and X_3X_5 would be expanded in a like manner. Upon completion of the expansion the terms resulting are sorted in the order of number of nonnegated literals they contain; also, they are flagged when optional. For use in the computer input each minterm is encoded by assigning it an octal value determined, as shown, in Equation 7.

(7) $$V(m) = \sum_{i=1}^{NL} \delta_i \cdot 2^{(i-1)}$$

Where

NL is the number of literals in each minterm

δ_i = 0 if the logical value of Xi is negative

δ_i = 1 if the logical value of Xi is true

For example, the value of the first term $V(X_1\overline{X}_2\overline{X}_3X_4X_5) = 2^0+2^3+2^4 = 25$, or 31 base 8. The advantage in the use of base 8 is that minterms may be constructed directly from the octal value by noting the weighting of each literal as shown in Figure 3 below.

$$\begin{tikzcd}[column sep=tiny] X_6 \arrow[d] & X_5 \arrow[d] & X_4 \arrow[d] & X_3 \arrow[d] & X_2 \arrow[d] & X_1 \arrow[d] & \\ 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & \text{Literal Weighting} \\ & 2^{\underline{nd}} & & & 1^{\underline{st}} & & \text{Octal Digit} \end{tikzcd}$$

Figure 3
Literal Weighting

For example, 31 above would have the minterm constructed by 1 giving $\overline{X}_3\overline{X}_2X_1$ and the 3 giving X_5X_4 or $X_5X_4\overline{X}_3\overline{X}_2X_1$. The first level of the prime implicant development is the ordered list of minterms. This list is given in Figure 4 for problem one. The first column has stars which are flags used in the prime implicant development as explained later. The second column contains a letter "O" to denote those minterms which are optional (i.e. an expansion of the optional term entered). The next column contains the octal value of the minterms. The last column contains the equation or output network number. For this problem there was only one output, so all the values are one.

It may be noted the minterms are grouped. This grouping is by the number of nonnegated literals in each. For example, the first is "1" which is $\overline{X}_5\overline{X}_4\overline{X}_3\overline{X}_2X_1$ and "4" which is $\overline{X}_5\overline{X}_4X_3\overline{X}_2\overline{X}_1$, both of which have one nonnegated literal. The next term, which is of the following group is "3" which is $\overline{X}_5\overline{X}_4\overline{X}_3X_2X_1$ and has two nonnegated literals.

```
PRIME IMPLICANT DEVELOPMENT



LEVEL 1
*          1  1
*C         4  1

*          3  1
*C         5  1
*C         6  1
*         11  1
*C        14  1
*         24  1

*C         7  1
*         13  1
*C        15  1
*C        16  1
*         25  1
*         26  1
*         31  1
*         34  1

*C        17  1
*         27  1
*         35  1
*         36  1

*         37  1
```

Figure 4

Prime Implicant Development
Problem 1 Level 1

Referring to Equation 1 it is obvious the reduction method used is applicable only when the number of literals of the two terms to be combined differ by one nonnegated literal. By the above grouping these terms would always be in adjacent groups. It is, therefore, necessary to search only the next group for possible reductions if one starts with the first. The procedure then, is to start with the first term of the first group and compare it to each term of the next group for a possible reduction by Equation 1. Where there is a reduction the reduced result is noted in the next level of the reduction, as shown in Figure 5. In the first group, for example, the following reduction is possible.

$$(8) \qquad 1 + 3 = \overline{X}_5\overline{X}_4\overline{X}_3\overline{X}_2X_1 + \overline{X}_5\overline{X}_4\overline{X}_3X_2X_1$$
$$= \overline{X}_5\overline{X}_4\overline{X}_3X_1$$
$$= 1$$

The above expression is tagged with a 2 denoting the second literal was removed from the terms. This result is shown in the first line of results in Figure 5. It may be noted that a simplification of the type used in this method is possible if, and only if, the terms differ by one literal only being negated in one term and not in the other. By use of the encoding as shown in Figure 3, this condition occurs when the encoded value of the terms

```
LEVEL   2
*          1   1    2
*          1   1    3
*          1   1    4
*          4   1    1
*          4   1    2
*          4   1    4
*          4   1    5

*          3   1    3
*          3   1    4
*          5   1    2
*          5   1    4
*          5   1    5
*          6   1    1
*          6   1    4
*          6   1    5
*         11   1    2
*         11   1    3
*         11   1    5
*         14   1    1
*         14   1    2
*         14   1    5
*         24   1    1
*         24   1    2
*         24   1    4

*          7   1    4
*          7   1    5
*         13   1    3
*         15   1    2
*         15   1    5
*         16   1    1
*         16   1    5
*         25   1    2
*         25   1    4
*         26   1    1
*         26   1    4
*         31   1    3
*         34   1    1
*         34   1    2

*         17   1    5
*         27   1    4
*         35   1    2
*         36   1    1
```

Figure 5

Prime Implicant Development
Problem 1 Level 2

differs by a power of 2. Our reduction procedure is then simplified to taking each term one at a time and comparing it to each term of the next group to determine if it differs by a power of 2. For example, the first term of Figure 4 has a value of 1. Comparing it with the terms of the next group it is seen that it differs by a power of 2 with the following octal numbers.

Table 6

Table of Differences of Minterms

Term	Octal Difference	Power of 2 of Difference
3	2	1
5	4	2
11	10	3

For the encoding system used it may be noted, as shown in Figure 3, that the literal represented as a difference is X_i where i is one greater than the power of 2 of the difference. The literal by which the term is reduced is called the "tag" and is shown in the last column of Figure 5. The results shown in Table 6 are given in the first three lines of the computer output in Figure 5. The remainder of the first group of Figure 5 is completed by comparing the term $\bar{X}_5\bar{X}_4X_3\bar{X}_2\bar{X}_1$

as represented by the octal term 4 with the terms of the second group in Figure 4. Each succeeding group of the second level is likewise formed by comparing the terms of the equivalent group in the first level one at a time with all the terms of the next group in the first level. Both terms in the first level for which there is a comparison differing by a power of 2 are starred (flagged) if, and only if, on that comparison all of the same outputs are included in both. If one term contains outputs not included in the other, only the common outputs are noted in the second level and the terms in the first level are not starred based on that comparison. In Figure 5, output for level 2, the first column is the flag denoting a comparison in the next level for those cases where the term is starred. The starring of a term flags it as a term that is included in a term of a higher level of the development.

The terms of the higher levels have fewer literals and require less gate inputs to mechanize; therefore they would be used rather than the starred terms in any optimum solution. For this reason the starred terms are removed from consideration as part of the final solution as they are flagged.

The fact that a term was derived from optional minterms is not noted as this information will not be used until completion of all of the levels and is available in the level one output data storage area in the computer. Therefore, the "O's" of the second column of Figure 4 are not included in any of the remaining levels. The next column is the code of the literals of the reduced term. Throughout, this data is in octal form. The octal encoding provides the convenience that each digit represents exactly three literals as shown in Figure 3. The last column is the tag (number of the literal which was reduced from the term).

The third level of the prime implicant development is derived in a similar manner. The one exception is that, in addition to differing by a power of two, terms must have the same tag to be reduced and entered in the next level. As was seen in the method of encoding and illustrated in Figure 3, the fact that two terms differ by a power of two denotes that one literal appears in the negated form in one term and in the nonnegated form in the other. However, if the tag indicates there are different literals removed from previous reductions, there would be in the original minterms of the derivation other differing literals and the basic reduction as given in Equation 1 would not be applicable. Hence, the tag must match in the reduction process.

As an example, the first term "1 1 2" ($\overline{X}_5\overline{X}_4\overline{X}_3X_1$, output 1, tag X_2) of level two differs by a power of two with "5 1 2" ($\overline{X}_5\overline{X}_4X_3X_1$, output 1, tag X_2) and "11 1 2" ($\overline{X}_5X_4\overline{X}_3X_1$, output 1, tag X_2) of the second group, yielding "1 1 2 3" ($\overline{X}_5\overline{X}_4X_1$, output 1, tags X_2 and X_3) and "1 1 2 4" ($\overline{X}_5\overline{X}_3X_1$, output 1, tags X_2 and X_4) respectively. The reduction for the first group of level three is completed in a similar manner and is given in Table 7 below.

Table 7

Reductions Forming First Group of Level 3

1st Group Level 2	2nd Group Level 2	Result
1 1 2	5 1 2	1 1 23
1 1 2	11 1 2	1 1 24
1 1 3	3 1 3	1 1 32+
1 1 3	11 1 3	1 1 34
1 1 4	3 1 4	1 1 42+
1 1 4	5 1 4	1 1 43+
4 1 1	6 1 1	4 1 12
4 1 1	14 1 1	4 1 14
4 1 1	24 1 1	4 1 15
4 1 2	5 1 2	4 1 21+
4 1 2	14 1 2	4 1 24
4 1 2	24 1 2	4 1 25
4 1 4	5 1 4	4 1 41+
4 1 4	6 1 4	4 1 42+
4 1 4	24 1 4	4 1 45
4 1 5	5 1 5	4 1 51+
4 1 5	6 1 5	4 1 52+
4 1 5	14 1 5	4 1 54+

It may be noted that the first and third terms are the same except for the order of the tags. The order of the tagged literals is the order in which literals are removed. As the order in which literals are removed in the reduction is of no importance to the result, these two terms are identical. There are a number of other terms which are also common in Table 7. The set of unique terms which are entered in level three, Figure 6, are indicated by a plus in Table 7.

One method of reducing the results of the algorithm to the unique terms would be to start with the total list for each group and compare each term with all succeeding terms and eliminate all but one in the case of identical terms. However, this would require storing all of the terms and making $\frac{1}{2}(n^2-n)$ comparisons, where n is the number of terms in the group. Also, the individual comparisons are relatively complex, entailing a comparison which would in effect unscramble the order of the tag or compare separately on each literal of the tag. The computer time is reduced and the need for storing all terms is eliminated by the algorithm used. With this algorithm the tag is tested for literals in ascending order, right to left. If a term does not fulfill this specification it is dropped upon generation, eliminating the need of buffer storage and a lengthy set of comparisons. The validity of this approach is shown below. In level two, where there

```
LEVEL   3
*        1   1     3   2
*        1   1     4   2
*        1   1     4   3
*        4   1     2   1
*        4   1     4   1
*        4   1     4   2
*        4   1     5   1
*        4   1     5   2
*        4   1     5   4

*        3   1     4   3
*        5   1     4   2
*        5   1     5   2
*        5   1     5   4
*        6   1     4   1
*        6   1     5   1
*        6   1     5   4
*       11   1     3   2
        11   1     5   3
*       14   1     2   1
*       14   1     5   1
*       14   1     5   2
*       24   1     2   1
*       24   1     4   1
*       24   1     4   2

*        7   1     5   4
*       15   1     5   2
*       16   1     5   1
*       25   1     4   2
*       26   1     4   1
*       34   1     2   1
LEVEL   4
         1   1     4   3   2
*        4   1     4   2   1
*        4   1     5   2   1
*        4   1     5   4   1
*        4   1     5   4   2

*        5   1     5   4   2
*        6   1     5   4   1
*       14   1     5   2   1
*       24   1     4   2   1
LEVEL   5
         4   1     5   4   2   1
```

Figure 6

Prime Implicant Development
Problem 1 Levels 3, 4, and 5

is only one tag, all terms are unique and are retained. For level three there are two literals which have been removed as common. Considering two generalized terms for which a reduction is possible we have

$$Y_1 \; \overline{X}_i \; Y_2 \qquad \text{tag} \; X_j$$

$$Y_1 \; X_i \; Y_2 \qquad \text{tag} \; X_j,$$

where Y_1 represents all the literals with a subscript greater than i and Y_2 all the literals with a subscript less than i. The above terms reduce to

$$Y_1 \; Y_2 \quad \text{tag} \quad X_j \; X_i.$$

It is noted, however, that the above terms being present implies that both the X_j and $\overline{X}_j$ literals are present with each of the terms $Y_1 \; X_i \; Y_2$ and $Y_1 \; \overline{X}_i \; Y_2$ and therefore there would also be in level two terms of the form

$$Y_3 \; \overline{X}_j \; Y_4 \qquad \text{tag} \; X_i$$

$$Y_3 \; X_j \; Y_4 \qquad \text{tag} \; X_i,$$

where the combined literals of Y_3 and Y_4 are the same as those of Y_1 and Y_2. These terms reduce to

$Y_3 \; Y_4 \quad \text{tag} \quad X_i \; X_j$ or, equivalently,

$$Y_1 \; Y_2 \quad \text{tag} \quad X_i \; X_j.$$

From this it is to be seen that for level three the basic algorithm will always yield pairs of equivalent terms. By the algorithm used, the term with the higher subscripted literal on the left (tag in ascending order, right to left) is selected. For the kth level there are (k-1) literals in the tag, which is represented as Z. Two terms of the kth level which are of the form that can be reduced for the (k+1) level are

$Y_1 \ \overline{X}_i \ Y_2$ tag Z

$Y_1 \ X_i \ Y_2$ tag Z. This reduces to

$Y_1 \ Y_2$ tag $Z \ X_i$. Now, if i is a subscript of smaller numerical value than any subscript of the (k-1) literals of Z, the subscripts will be in ascending order, right to left, since Z from the previous steps was in ascending order. In this case the term will be retained. In the case where i is numerically greater than the smallest subscript in Z, the smallest subscript is denoted j. As in the argument for the case of two terms, the possibility of a reduction for the literal X_j in a previous level implies that both the X_j and $\overline{X}_j$ literals are present with each of the terms

$Y_1 \ X_i \ Y_2$ tag Z

and $Y_1 \ \overline{X}_i \ Y_2$ tag Z and therefore there would also be in the kth level two terms of the form

$Y_3 \ \overline{X}_j \ Y_4$ tag Z'

$Y_3 \ X_j \ Y_4$ tag Z', where Z' contains all the literals of Z, except X_i is included and X_j is not. These two terms reduce to $Y_3 \ Y_4$ tag $Z' \ X_j$.

As the literals of the tag $Z' X_j$ are the same as tag $Z X_i$, and the combined remaining literals of $Y_3 Y_4$ are the same as the combined remaining literals of $Y_1 Y_2$, the two (k+1) level reduced terms $Y_1 Y_2$ tag $Z X_i$ and $Y_3 Y_4$ tag $Z' X_j$ are equivalent. By the above argument it is shown that for any term with the tag subscripts not in ascending order there will be an equivalent term with the tag subscripts in ascending order. Therefore, terms, where the tag subscripts are not in ascending order, may be deleted without further evaluation.

The remainder of level three and levels four and five are developed in a like manner. The starred terms in Figures 4, 5, and 6 are the terms which are wholly included in a reduction, resulting in a term on the next level. The unstarred terms which remain include, therefore, all of the original minterms and are denoted the prime implicants. In Problem 1 these terms are:
11 1 5 3 in level three, 1 1 4 3 2 in level four, and 4 1 5 4 2 1 in level five. As described in the input data, the user may optionally select a prime implicant listing as part of the output from the computer. This includes all the information as shown in Figure 7 for Problem 1. Included is all of the information for level one on the minterms, as shown in Figure 4, and a listing of the prime implicants.

```
PRIME IMPLICANT LISTING


LFVFL   1
*          1   1
*0         4   1

*          3   1
*0         5   1
*0         6   1
*         11   1
*0        14   1
*         24   1

*0         7   1
*         13   1
*0        15   1
*0        16   1
*         25   1
*         26   1
*         31   1
*         34   1

*0        17   1
*         27   1
*         35   1
*         36   1

*         37   1
LEVEL   2



LFVFL   3
          11   1     5   3

LEVFL   4
           1   1     4   3   2

LFVEL   5
           4   1     5   4   2   1
```

Figure 7

Prime Implicant Listing
Problem 1

A prime implicant is termed an essential prime implicant when it is the only one in which a required minterm is included. Such a prime implicant must, of course, be included as part of the solution in order to include the required minterm. To determine the essential prime implicants each minterm is tested against all prime implicants for its inclusion in prime implicants. If it is included in two or more prime implicants it does not require an essential prime implicant for its inclusion. If there is only one prime implicant in which it is included, that prime implicant is an essential prime implicant. In this case, all minterms which are included in this prime implicant are excluded from the test for further essential prime implicants by the computer program. If such a minterm were included in only this prime implicant it would indicate that this prime implicant was essential for more than one minterm; however, it is still an essential prime implicant. If an excluded minterm were included in another prime implicant it would be included in at least two prime implicants and, therefore, would not require an essential prime implicant to include it. In either case there is no loss in excluding the other minterms included in essential prime implicants from further testing to save computer time. If a minterm is not included in any prime implicant (unstarred in level one) it is treated as an essential prime implicant. The essential prime implicants for Problem 1 are shown in Figure 8. The literals are denoted by the numbers 1, 2, and 3 as shown in Table 8 below.

Table 8

Essential Prime Implicant Codes

Literal Code	Meaning
1	Literal included in negated form (i.e. $\bar{X}_i$).
2	Literal included in non-negated form (i.e. X_i).
3	Literal not included.
Output Code	**Meaning**
1	Not included for this output.
2	Included as part of this output.
3	Essential for this output.

The literals are in the format $X_5X_4X_3X_2X_1$. There is only one output for the network of this problem; in all cases the essential prime implicants for this output are coded "3" (essential for this output). Other outputs, had there been any, would have been listed in ascending order from right to left.

The listing of the essential prime implicants is an

```
ESSENTIAL PRIME IMPLICANTS


          LITERALS         OUTPUTS
             13332               3
             33233               3
             32312               3
```

Figure 8

Essential Prime Implicants .
Problem 1

optional listing which the user may select upon problem entry. For this problem all the minterms are included in the essential prime implicants and therefore a listing of essential prime implicants is the problem solution. When this occurs, the computer program states the fact and gives the listing of the essential prime implicants as shown in Figure 9. This solution to the problem is represented in literal form as

$$A = \overline{X}_5X_1 + X_3 + X_4\overline{X}_2X_1. \qquad (9)$$

This form requires two AND gate inputs for the first term and three for the third, plus three OR gate inputs, for a total of eight gate inputs as compared to the original form of the problem which required thirteen. It may be noted the second term, being a single term, does not require an AND gate input but may be wired direct to an OR gate input. The solution as determined by the computer is a minimum solution. When all the prime implicants are required as essential the solution is, of course, also the minimum cost solution.

2.4.2 Problem 2

Problem 2 provides a case where the desired network includes terms other than essential prime implicants.

```
ALL TERMS ARE COVERED BY THE ESSENTIAL PRIME IMPLICANTS
          LITERALS          OUTPUTS
             13332               3
             33233               3
             32312               3
```

Figure 9

Problem 1 Solution

Problem 2 is to find the optimum AND-OR mechanization of Equation 10.

$$(10) \qquad A = \overline{X}_1\overline{X}_2\overline{X}_4 + X_1\overline{X}_2X_3 + \overline{X}_1X_2\overline{X}_3\overline{X}_4$$

The data input and computer acknowledgement is as explained in Section 2.3.1, Data Entry, and is illustrated for Problem 1, Section 2.4.1. The computer acknowledgement for Problem 2 is shown in Figure 10. The prime implicant development and prime implicant listing are shown in Figures 11 and 12 respectively. It may be noted that because there is no reduction possible past the second level the prime implicant development and prime implicant listing are the same. The essential prime implicants are shown in Figure 13. The solution to Problem 2 is shown in Figure 14. The problem solution is provided separately by network output. First, the prime implicants that are included exclusive of the essential prime implicants are given. For Problem 2 there is one 1311 ($\overline{X}_4\overline{X}_2\overline{X}_1$). Also, the number of literals in the term (LIT 3), weight (WT 5), and output status (OUTPUT 2) is given. The weight is the weighting of the prime implicant. In this case a weighting of five was used. As described in Section 2.3.3, Optimum Prime Implicant Selection, a weight of one is assigned for the single minterm not included in the

```
                AUTOMATED LOGIC DESIGN PROGRAM
AND-OR MINIMIZATION BASED ON A UNIFORM COST PER INPUT


    INPUT DATA
    NO. LITERALS=      4
    NO. OUTPUTS=       1
    NO. SOLUTIONS TO BE CONSIDERED=    25
    NO. OF PRIME IMPLICANTS TAKEN IN COMBINATIONS OF
          2     3     4     5     6     7     8     9    10
         30    15    12    10    10    10    10    10    10

    VARIABLE
       223200
       121300
       212200
```

Figure 10

Problem 2 Specification

```
   PRIME IMPLICANT DEVELOPMENT



      LEVEL 1
      *          0   1

      *          2   1
      *          4   1

      *          5   1

      *         15   1
      LEVEL  2
                 0   1      2
                 0   1      3

                 4   1      1

                 5   1      4
```

Figure 11

Prime Implicant Development
Problem 2

```
PRIME IMPLICANT LISTING

LEVEL   1
*            0   1

*            2   1
*            4   1

*            5   1

*           15   1

LEVEL   2
             0   1        2
             0   1        3

             4   1        1

             5   1        4
```

Figure 12

Prime Implicant Listing
Problem 2

```
ESSENTIAL PRIME IMPLICANTS

          LITERALS            OUTPUTS
              1131                  3
              3212                  3
```

Figure 13

Essential Prime Implicants
Problem 2

```
PROBLEM SOLUTION




OUTPUT NO. 1

   PRIME IMPLICANT      LIT WT OUTPUT
              1311        3  5      2

ESSENTIAL PRIME IMPLICANTS
              1131                  3
              3212                  3

NO. OF 'AND' GATE INPUTS REQUIRED =         9
NO. OF 'OR' GATE INPUTS REQUIRED  =         3
TOTAL=      12

TOTAL NO. OF SOLUTIONS=     3
THIS WAS THE     1 TH
```

Figure 14

Problem 2 Solution

essential prime implicants and an additional weight of four because there were only two prime implicants that included this minterm. The output code 2 denotes that the prime implicant is included in this output network but is not an essential prime implicant. All the literal and output codes used in the final solution are the same as those detailed for the essential prime implicants in Table 8. The order of the literals is from right to left (i.e. $X_4X_3X_2X_1$), the same as for the essential prime implicants. The essential prime implicants 1131 ($\overline{X}_4\overline{X}_3\overline{X}_1$) and 3212 ($X_3\overline{X}_2X_1$) are given next, along with their output code of 3 denoting they are essential for this output network. For the case of a network requiring a single output, the output coding is somewhat redundant as the titles and grouping would provide the same information; however, where a network has multiple outputs, this information gives the status of each prime implicant in reference to each output network. This is better seen and is explained in detail in the next sample problem. For the programing convenience of using one less print format, the output information is printed for the single output network as well as for multiple output networks. The problem solution is

$$(11) \qquad A = \overline{X}_4\overline{X}_2\overline{X}_1 + \overline{X}_4\overline{X}_3\overline{X}_1 + X_3\overline{X}_2X_1.$$

The number of AND gate inputs required are three for each of the three terms, or nine. The number of OR gate inputs required is one for each term, or three, for a total of twelve gate inputs. For the case where a solution contained a term with a single literal, the correct solution would be indicated; however, the AND gate input count would be one greater than required since this single term could be connected direct to the OR gate inputs. Also, for the case where the solution is one term, the OR gate would not be required.

The computer also states the total number of solutions found and which one was best. As the total number of solutions was less than the twenty five requested by the default option (see Figure 10, Problem 2 Specification), it is known the search was exhausted and the solution is optimum. For the case where the number of nonessential prime implicants is less than the number of terms taken in combination, this test is conclusive. For a large problem, all of the combinations specified may have been tested and the number of solutions still be less than the number specified; in this case a solution is printed but the fact the number of solutions was not achieved would not indicate an exhaustive search of all combinations. The solution number for the best solution is given as an aid to the user in getting a feel for when he is over or under specifying for long problems and for what weighting factors would seem to best fit his problem. This is an optional part of the input, as specified in Section 2.3.3, Optimum Prime Implicant Selection.

2.4.3 Problem 3

Problem 3 is solution of a network requiring two outputs. In the case of a multiple output network an overall minimum cost solution is sought: that is, the cost of generating each output may not necessarily be minimum if the added cost is more than offset by savings in making part of the network more usable in generating one or more of the other output functions. This approach is, of course, the optimum approach as compared to simply using those sections of the network, when available, which happen to exist for another output function. Problem 3 is to find the optimum AND-OR mechanization of Equation 12 and 13.

$$A_1 = \overline{X}_1\overline{X}_3\overline{X}_4 + X_1\overline{X}_2X_3X_4 + \overline{X}_2X_3\overline{X}_4 \tag{12}$$

$$A_2 = \overline{X}_2X_3\overline{X}_4 + X_1\overline{X}_3\overline{X}_4 + X_1X_2X_3\overline{X}_4 \tag{13}$$

The data input and computer acknowledgement is as explained in Section 2.3.1, Data Entry, and is illustrated for Problem 1. The computer acknowledgement is shown in Figure 15. A difference which may be noted is the printing of the associated outputs with each term. For example, the first term, 2322 ($\overline{X}_1\overline{X}_3\overline{X}_4$), is followed by 010. The first zero is a separator; the one denotes it is associated with with the first output network, or Equation 12; the next zero may be regarded as a blank, indicating this term is specified for only one of the outputs. The prime implicant development and prime implicant listing are given in Figures 16 and 17 respectively. The output column is the second numeric column. It is coded the same as the literals, as shown in Figure 3 for the literals. That is, each output is assigned a value σ as determined by Equation 14.

$$(14) \qquad \sigma = \sum_{i=1}^{NO} \delta_i 2^{i-1}$$

Where NO is the number of outputs

$\delta_i = 0$ if the term is not included in the ith output

$\delta_i = 1$ if the term is included in the ith output

As with the literals, the result is expressed in octal. For Problem 3, which has two outputs, terms included in the first output only are coded one, terms included in the second only are coded two, and terms included in both

```
                AUTOMATED LOGIC DESIGN PROGRAM
AND-OR MINIMIZATION BASED ON A UNIFORM COST PER INPUT



INPUT DATA
NO. LITERALS=      4
NO. OUTPUTS=       2
NO. SOLUTIONS TO BE CONSIDERED=    25
NO. OF PRIME IMPLICANTS TAKEN IN COMBINATIONS OF
     2     3     4     5     6     7     8     9    10
    30    15    12    10    10    10    10    10    10

VARIABLE
   2322010
   1211010
   3212012
   1322020
   1112020
```

Figure 15

Problem 3 Specification

```
             PRIME IMPLICANT DEVELOPMENT


LEVEL 1
*              0    1

*              1    2
*              2    1
*              4    3

*              3    2
*              5    3

*              7    2
              15    1
LFVFL   2
               0    1       2
               0    1       3

*              1    2       2
*              1    2       3
               4    3       1

*              3    2       3
*              5    2       2
               5    1       4
LFVFI   3
               1    2       3    2
```

Figure 16

Prime Implicant Development
Problem 3

```
            PRIME IMPLICANT LISTING



LEVEL    1
*           0    1

*           1    2
*           2    1
*           4    3

*           3    2
*           5    3

*           7    2
           15    1

LEVEL    2
            0    1       2
            0    1       3

            4    3       1

            5    1       4

LEVEL    3
            1    2       3    2
```

Figure 17

Prime Implicant Listing
Problem 3

are coded three. A brief summary of the computer printout is now given. The codings used for input and output acknowledgement are outlined in Tables 1 through 4. The literal and output codes for the prime implicant development and prime implicant listing are outlined in Figure 3. The tag is simply the literals that have been removed from the terms by repeated application of Equation 1. The prime implicants for the multiple output case are developed as for the single output except, when all the outputs are not included in the various terms of a reduction, only those outputs common to all terms are listed for the reduced term.

The essential prime implicants for Problem 3 are listed in Figure 18 and the final solution in Figure 19. In each of these listings there is one column for each output. The columns for the outputs are listed from right to left and coded as outlined in Table 8. For the final solution, the first output network includes one prime implicant which is not of the class of essential prime implicants for the first output network. This is the prime implicant 1213 ($\overline{X}_4X_3\overline{X}_2$). The literal cost is given as zero since the AND gating for the generation of this term is also used in the second output network. The prime implicant was weighted one because it contained one min-

```
ESSENTIAL PRIME IMPLICANTS


     LITERALS              OUTPUTS
         1332                   31
         1131                   13
         1213                   32
         3212                   13
```

Figure 18

Essential Prime Implicants
Problem 3

```
                    PROBLEM SOLUTION

OUTPUT NO. 1

    PRIME IMPLICANT        LIT WT OUTPUT
              1213          0   5      22

ESSENTIAL PRIME IMPLICANTS
              1131                     13
              3212                     13



                    PROBLEM SOLUTION

OUTPUT NO. 2

    PRIME IMPLICANT        LIT WT OUTPUT

ESSENTIAL PRIME IMPLICANTS
              1332                     31
              1213                     32

NO. OF 'AND' GATE INPUTS REQUIRED =          11
NO. OF 'OR' GATE INPUTS REQUIRED  =           5
TOTAL=        16

TOTAL NO. OF SOLUTIONS=     3
THIS WAS THE      1 TH
```

Figure 19

Problem 3 Solution

term for one output that was not included in the essential prime implicants, plus an additional four because there was only one other prime implicant that also included this minterm. The output code 22 denotes the prime implicant could be used in either of the two output networks. The essential prime implicants for the first output networks are 1131 ($\overline{X}_4\overline{X}_3\overline{X}_1$) and 3212 ($X_3\overline{X}_2X_1$), both of which are applicable only to the first output network. For the second output network there are just two essential prime implicants, 1332 ($\overline{X}_4X_1$) and 1213 ($\overline{X}_4X_3\overline{X}_2$), of which the former is applicable only to the second output network and the latter is included in both. As noted earlier, this was the term which was included at no additional cost for the literals (AND inputs) in the first output network. The equation form of the solution is as follows:

(15) $$A_1 = \overline{X}_4X_3\overline{X}_2 + \overline{X}_4\overline{X}_3\overline{X}_1 + X_3\overline{X}_2X_1$$

(16) $$A_2 = \overline{X}_4X_1 + \overline{X}_4X_3\overline{X}_2$$

The number of gate inputs is sixteen as compared to twenty three for mechanization of the equations as stated in the input form (Equations 12 and 13). This solution was the first of three possible solutions as noted in Figure 19. It is also known to be the best possible solution for the same reasons given in Section 2.4.2 for Problem 2.

2.4.4 Problem 4

Problem 4 is part of a test of the operation of the program. It is the same as Problem 3 except the third input is entered separately for each of the output networks. This problem checks the program's ability to combine like entries, encoding them with the various output networks they may be associated with whether or not they were separately specified in the problem input. The problem specification, prime implicant listing, and problem solution are given in Figures 20, 21, and 22 respectively. The prime implicant development and essential prime implicant list were not requested and were therefore omitted from the computer output. As they should be, the prime implicant listing and problem solution are identical with those of Problem 3.

2.4.5 Problem 5

Problem 5 tests the feature which allows minterms to be entered any number of times, as long as the specifications are consistent. The input specification is like Problem 3, except the last term is redundant since it is included as part of the third term. The input specification is shown in Figure 23, the prime implicant development and essential prime implicants in Figure 24, and the

```
                  AUTOMATED LOGIC DESIGN PROGRAM
AND-OR MINIMIZATION BASED ON A UNIFORM COST PER INPUT



    INPUT DATA
    NO. LITERALS=      4
    NO. OUTPUTS=       2
    NO. SOLUTIONS TO BE CONSIDERED=    25
    NO. OF PRIME IMPLICANTS TAKEN IN COMBINATIONS OF
          2      3      4      5      6      7      8      9     10
      30     15     12     10     10     10     10     10     10

    VARIABLE
      2322010
      1211010
      3212010
      3212020
      1322020
      1112020
```

Figure 20

Problem 4 Specification

```
        PRIME IMPLICANT LISTING

LEVEL    1
*           0    1

*           1    2
*           2    1
*           4    3

*           3    2
*           5    3

*           7    2
           15    1
LEVEL   2
            0    1      2
            0    1      3

            4    3      1

            5    1      4
LEVEL   3
            1    2      3    2
```

Figure 21

Prime Implicant Listing
Problem 4

```
                    PROBLEM SOLUTION


OUTPUT NO. 1

    PRIME IMPLICANT        LIT WT OUTPUT
                1213         0   5     22

ESSENTIAL PRIME IMPLICANTS
                1131                   13
                3212                   13



                    PROBLEM SOLUTION


OUTPUT NO. 2

    PRIME IMPLICANT        LIT WT OUTPUT

ESSENTIAL PRIME IMPLICANTS
                1332                   31
                1213                   32

NO. OF 'AND' GATE INPUTS REQUIRED  =        11
NO. OF 'OR' GATE INPUTS REQUIRED   =         5
TOTAL=        16

TOTAL NO. OF SOLUTIONS=      3
THIS WAS THE       1 TH
```

Figure 22

Problem 4 Solution

```
                AUTOMATED LOGIC DESIGN PROGRAM
   AND-OR MINIMIZATION BASED ON A UNIFORM COST PER INPUT


    INPUT DATA
    NO. LITERALS=      4
    NO. OUTPUTS=       2
    NO. SOLUTIONS TO BE CONSIDERED=    25
    NO. OF PRIME IMPLICANTS TAKEN IN COMBINATIONS OF
          2     3     4     5     6     7     8     9    10
       30    15    12    10    10    10    10    10    10

    VARIABLE
       2322010
       1211010
       3212012
       1322020
       1112020
       1212020
```

Figure 23

Problem 5 Specification

```
PRIME IMPLICANT DEVELOPMENT


   LEVEL 1
   *          0   1

   *          1   2
   *          2   1
   *          4   3

   *          3   2
   *          5   3

   *          7   2
             15   1
   LEVEL  2
              0   1   2
              0   1   3

   *          1   2   2
   *          1   2   3
              4   3   1

   *          3   2   3
   *          5   2   2
              5   1   4
   LEVEL  3
              1   2   3   2



ESSENTIAL PRIME IMPLICANTS
              LITERALS        OUTPUTS
                  1332             31
                  1131             13
                  1213             32
                  3212             13
```

Figure 24

Prime Implicant Development and
Essential Prime Implicants
Problem 5

```
                 PROBLEM SOLUTION


OUTPUT NO. 1

   PRIME IMPLICANT       LIT WT OUTPUT
              1213         0  5     22

ESSENTIAL PRIME IMPLICANTS
              1131                  13
              3212                  13


                 PROBLEM SOLUTION


OUTPUT NO. 2

   PRIME IMPLICANT       LIT WT OUTPUT

ESSENTIAL PRIME IMPLICANTS
              1332                  31
              1213                  32

NO. OF 'AND' GATE INPUTS REQUIRED  =        11
NO. OF 'OR' GATE INPUTS REQUIRED   =         5
TOTAL=        16

TOTAL NO. OF SOLUTIONS=      3
THIS WAS THE      1 TH
```

Figure 25

Problem 5 Solution

problem solution in Figure 25. The prime implicant listing was not requested and was therefore omitted from the computer output. The development and solution are identical to Problem 3, as they should be.

2.4.6. Problem 6

Problem 6 is the same as problem 5 except that the last term, which was redundant in Problem 5 has been specified as an optional term. This creates a conflicting specification for the part of term three that includes the last term. The entry and results of Problem 6 are shown in Figure 26. The duplicate minterm entry for which there is a conflicting specification is entered on the last line. The two least significant digits give the octal value of the output network (i.e. 02 denotes the second output network and 03 both the first and second output networks). The next six digits are the octal value of the literals. The value 5 denotes $\overline{X}_4X_3\overline{X}_2X_1$. The most significant places are the number of true literals in base ten numbers (i.e. 2 denotes two true literals, X_3 and X_1). The sign denotes whether the minterm is required or optional; a negative sign denotes an optional minterm. The last term, -1212020 ($X_1\overline{X}_2X_3\overline{X}_4$ output 2), specified as an optional term, yielded the first term listed of the

```
                    AUTOMATED LOGIC DESIGN PROGRAM
  AND-OR MINIMIZATION BASED ON A UNIFORM COST PER INPUT


INPUT DATA
NO. LITERALS=     4
NO. OUTPUTS=      2
NO. SOLUTIONS TO BE CONSIDERED=    25
NO. OF PRIME IMPLICANTS TAKEN IN COMBINATIONS OF
     2     3     4     5     6     7     8     9    10
    30    15    12    10    10    10    10    10    10

VARIABLE
   2322010
   1211010
   3212012
   1322020
   1112020
  -1212020
DUPLICATE MINTERM ENTRY      -200000502       200000503
```

Figure 26

Problem 6 Entry and Result

duplicate minterms. The third term, 3212012 ($\overline{X}_2X_3\overline{X}_4$ outputs 1 and 2), yielded the second minterm listed ($\overline{X}_4X_3\overline{X}_2X_1$ outputs 1 and 2). The term $\overline{X}_2X_3\overline{X}_4$ from Equation 1 is seen to be composed of the two minterms, $\overline{X}_1\overline{X}_2X_3\overline{X}_4$ and $X_1\overline{X}_2X_3\overline{X}_4$. The optional specification of the last entry for the minterm $X_1\overline{X}_2X_3\overline{X}_4$ on output 2 is in conflict with the specification of the third term which states both minterms $\overline{X}_1\overline{X}_2X_3\overline{X}_4$ and $X_1\overline{X}_2X_3\overline{X}_4$ are required terms for both outputs. As a specification that a term is both optional and required is inconsistent, the duplicate minterm entry is noted to the user and the problem run is terminated.

2.4.7 Problem 7

Problem 7 consists of a test on the maximum number of allowable all combination literals (literals entered with a code 3). Each term with k such entries is composed of 2^k minterms. For example, if four literals are used the entry of 1332 ($X_1\overline{X}_4$) is in fact representative of the four minterms $X_1\overline{X}_2\overline{X}_3\overline{X}_4$, $X_1X_2\overline{X}_3\overline{X}_4$, $X_1\overline{X}_2X_3\overline{X}_4$, and $X_1X_2X_3\overline{X}_4$. If more than a thousand minterms are used there is a good possibility of storage space in the computer being exceeded. Ten all combination literals would result in 2^{10} or 1024 minterms and are therefore excluded with a note printed to the user that more than nine all combination literals have been used. Figure 27 provides an

```
              AUTOMATED LOGIC DESIGN PROGRAM
AND-OR MINIMIZATION BASED ON A UNIFORM COST PER INPUT


 INPUT DATA
 NO. LITERALS=    11
 NO. OUTPUTS=      1
 NO. SOLUTIONS TO BE CONSIDERED=    25
 NO. OF PRIME IMPLICANTS TAKEN IN COMBINATIONS OF
      2     3     4     5     6     7     8     9    10
     30    15    12    10    10    10    10    10    10

 VARIABLE
   33331111111100
   33333333333200
 MORE THAN 9 ALL COMBINATION LITERALS USED
```

Figure 27

Problem 7 Entry and Result

example of this type of output.

2.4.8 Problem 8

Problem 8 provides an example of an entry with more than a thousand minterms. If more than a thousand minterms are entered, the computer program notifies the user. Figure 28 provides an example of the computer printout for Problem 8. This problem size restriction is a practical limit based on the memory limits of the computer used. By use of larger amounts of computer memory there is no theoretical limit to the size of job which can be run. While there is no theoretical limit, there is a practical limit in the amount of computer time used. This is more of a limiting factor than the memory size. The amount of time used for all the problems shown in this report combined was only one minute and fifty eight seconds, including link editing and printout. Compilation and printing of the computer listing as shown in the appendix took nine minutes and twenty eight seconds. This, of course, could be reduced considerably by use of an object deck of the final program. With larger problems, containing more prime implicants, the time for a solution increases very rapidly since the number of combinations to be analyzed tends to grow in a factorial type expansion to the limits specified by the number of nonessential prime implicants

```
                AUTOMATED LOGIC DESIGN PROGRAM
AND-OR MINIMIZATION BASED ON A UNIFORM COST PER INPUT


     INPUT DATA
     NO. LITERALS=     11
     NO. OUTPUTS=       1
     NO. SOLUTIONS TO BE CONSIDERED=    25
     NO. OF PRIME IMPLICANTS TAKEN IN COMBINATIONS OF
          2      3      4      5      6      7      8      9     10
        30     15     12     10     10     10     10     10     10

     VARIABLE
       33333333331100
       33333333331200
     MORE THAN 1000 MINTERMS USED
```

Figure 28

Problem 8
Entry and Result

and the combinations in which they are considered.

For analysis of each output network of each solution, the same basic approach used in the process of finding the solution is repeated, except only strings of the prime implicants known to be in the overall solution are included. This process is used to provide the minimum cost circuit, which includes the sharing of hardware, not only for what may already exist, but rather developing the circuitry so the cost is minimum, considering all outputs.

As was noted earlier, if the problem is of a size where all combinations are not tested the solution printed is the best of those tested. In the case of multiple outputs, at each comparison during the prime implicant development terms reduced for some outputs, but not all, are left for further consideration (not flagged). It is therefore possible to have included in the final solution a term that would be included in another term (not really a prime implicant). Such terms should be removed by visual scanning of the solution by the user. Where equivalent terms are in the range of the combinations used for the problem the computer will automatically select the best solution avoiding this problem.

Considering the number of prime implicants taken in combination, it would be at least as many as defined in conjunction with table 4 if a lower X is not given an optional value less than a higher X. Basically, at each X level the number of prime implicants considered is that specified (ie. at the third X level, X=3, using the default option of 15, each of the first 15 prime implicants would be considered in turn with all unique combinations of two other terms, where the other two may include prime implicants above the 15th based on the lower X level specifications).

As an example of a longer problem, a problem with seven literals and 34 prime implicants, including five essential prime implicants, was run with a reduced search. The program was stopped by the operator with an elapsed time of one hour, six and a half minutes. By use of printout at selected steps, it was found that the computer had proceeded correctly to the first solution and was in the process of analyzing this solution. While the long time could possibly be attributed to an undetected programing "bug", a consideration of the amount of computation indicates a time limitation.

A significant improvement in the running time for larger problems would be achieved if the prime implicants and the minterms were stored in an expanded form for the last section of the program, subroutine OPTMPI. Also, if a separate list of the minterms not included by the current group of test solution prime implicants was maintained, the running time would be reduced. With this method, as each combination is analyzed, only the one new prime implicant normally used to replace one of the previous ones need be tested to see if it includes the still missing minterms.

The present method looks at each new set of prime implicants separately to see if they include the required minterms. The minterms and prime implicants are stored in a compact format which requires expansion for convenient operation. Due to the above considerations, more than an order of magnitude time savings would be anticipated for the shorter approach on large problems. While the theoretical maximum size problem that could be run with a given size memory would be reduced, due to the extra storage required, the upper practical limit would be increased. For very short problems or problems requesting only prime implicant listings there would be no significant difference from the present program. In the next chapter a further development of the overall design problem is treated on a broad basis and a sample machine design problem is presented.

CHAPTER 3. FURTHER DEVELOPMENT OF THE AUTOMATIC DESIGN PROBLEM

In the course of the thesis work an integrated approach to the development of automatic design techniques in the field of switching networks was developed to a limited extent. This is a program in which computer aided design would be carried to higher level functions and total devices. The approach proposed is a hierarchy of supervisor routines which would call basic optimization programs similar to the AND-OR minimization program of this thesis. The modules of this program are envisioned as containing models including all the significant real life problems of design so as to require a minimum of user interpretation of the results.

Designs which include all of the applicable real life problems, such as variations of temperature, power supply voltage, circuit loading, stray capacitance, deterministic noise (predictable undesired short term pulses), statistical noise, etc., are by their nature many times more difficult - if not impossible - to solve by a single algorithm. The practical approach to this problem is to consider the many developed approaches to the design, evaluate the classes which are most likely to lead to a solution, and search these, selecting the best. These are the steps in a good manual design. With an automated design these same steps can be performed much faster without the problems of clerical error. The resulting computer aided design would therefore be developed at a lower cost and be basically error free. Due to the higher speed of automated design more possible approaches may be considered, providing for still greater economies. It would in this way be practical to develop more specialized and improved techniques for various classes of problems due to the wide usage such a program would have. When the person developing the program is not very certain of the best approaches, the program itself may be equipped with memory of past experience in finding optimum solutions along various routes. This information is used to self-modify the program to guide in its future approaches to solutions of similar problems. However, if this approach is used in lieu of a direct approach where one exists, a less than optimum search will generally result. This was pointed out by M. Minsky(20) who discussed the shortcomings of the "Logic Theory" program of Newell, Shaw and Simon(21, 22) in the light of the criticism by Wang(23).

It was pointed out by M. A. Breuer(24) and E. J. McCluskey(25) that this topic, effective automatic generation of logic, is one of the major classes of automated computer design which remains unsolved. Due to the enormous scope of this project it is part of this thesis to set up a program of a continuing nature which can be further developed in future investigations. In the next section, the program structure is discussed and in the section following a sample design problem is presented.

3.1 Program Structure

The program is planned around a hierarchy of specifications which are to be implimented by a set of library programs. These programs are in turn designed to search for an optimized solution in their specific areas, after which control is returned to the higher level routines. The higher level routines are given the capability to call on the design level routines iteratively or in combination changing the specification, in order to get an optimized solution in cases where trade off is necessary and solution of the equations involved simultaneously is not practical. The "Machine Type Specification" is the highest level and is used to call the basic routines involved in the problem solution. This specification states the type of unit that is to be designed. Secondary specifications are used for such items as the input interface, output interface, items which directly affect the logic design but are not part of it (lumped cost items), the general specification of what the machine is to do with the input, etc. A description of these specifications follows, with their program implementation implied in Figures 29 and 30 and in Section 3.1.9.

3.1.1 Machine Type Specification

The type of machine to be designed is specified. This specification states whether it is primarily a computer main frame, medium size computer in total, desk top computer, card reader or punch, magnetic tape transport, disk memory, core memory, drum storage unit, paper tape reader or punch, data buffer, data transmission or terminal device, cash register, etc. The basic approach to design and the decisions to be considered would, of course, vary widely within the above types of machines. To handle this problem a supervisory routine is called by the entry of the code that describes the machine. The supervisory routine in turn calls other routines which are common to the various types of devices. This provision saves computer memory and allows a modular approach to building and addition to the overall program.

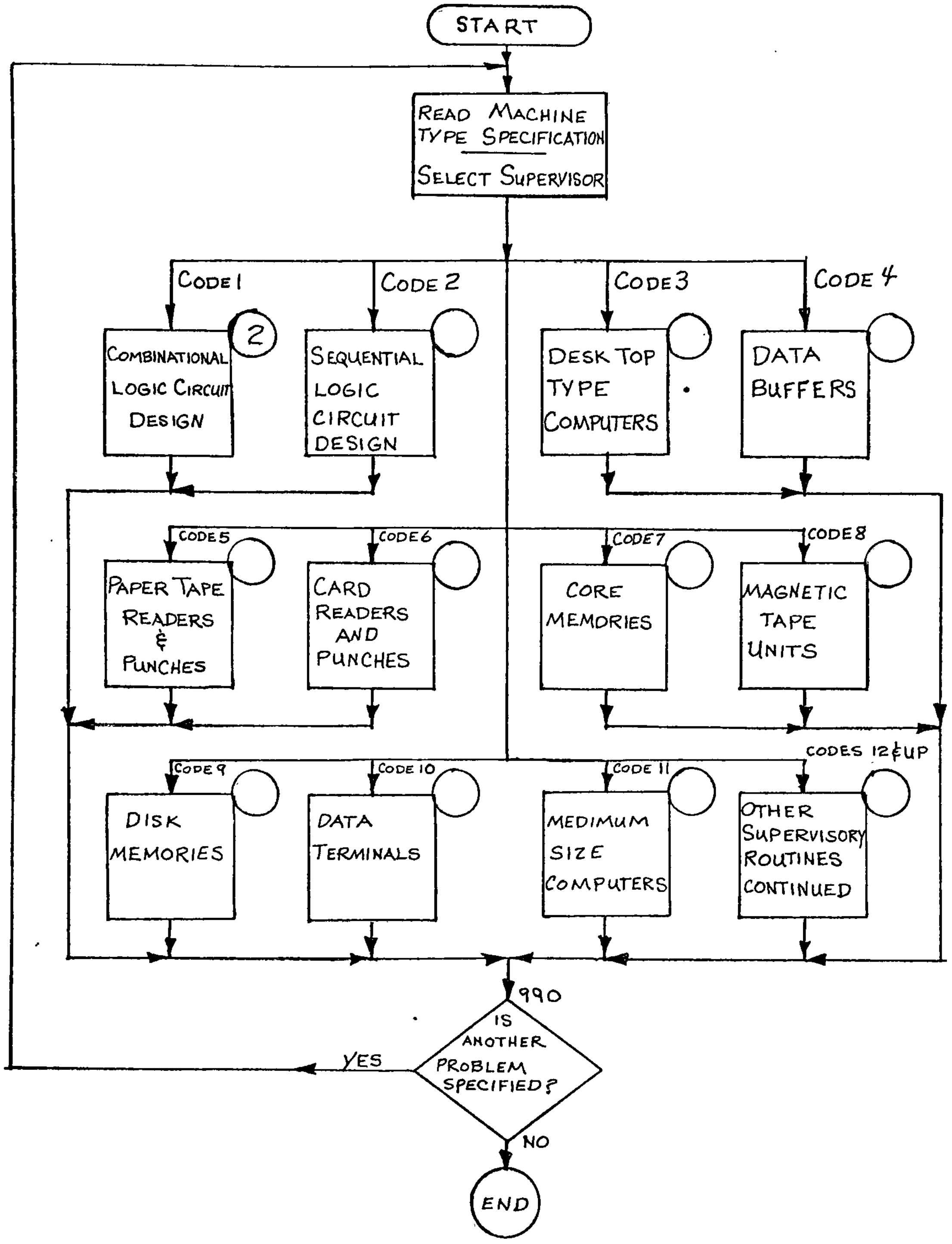


Figure 29

Automated Logic Design Program

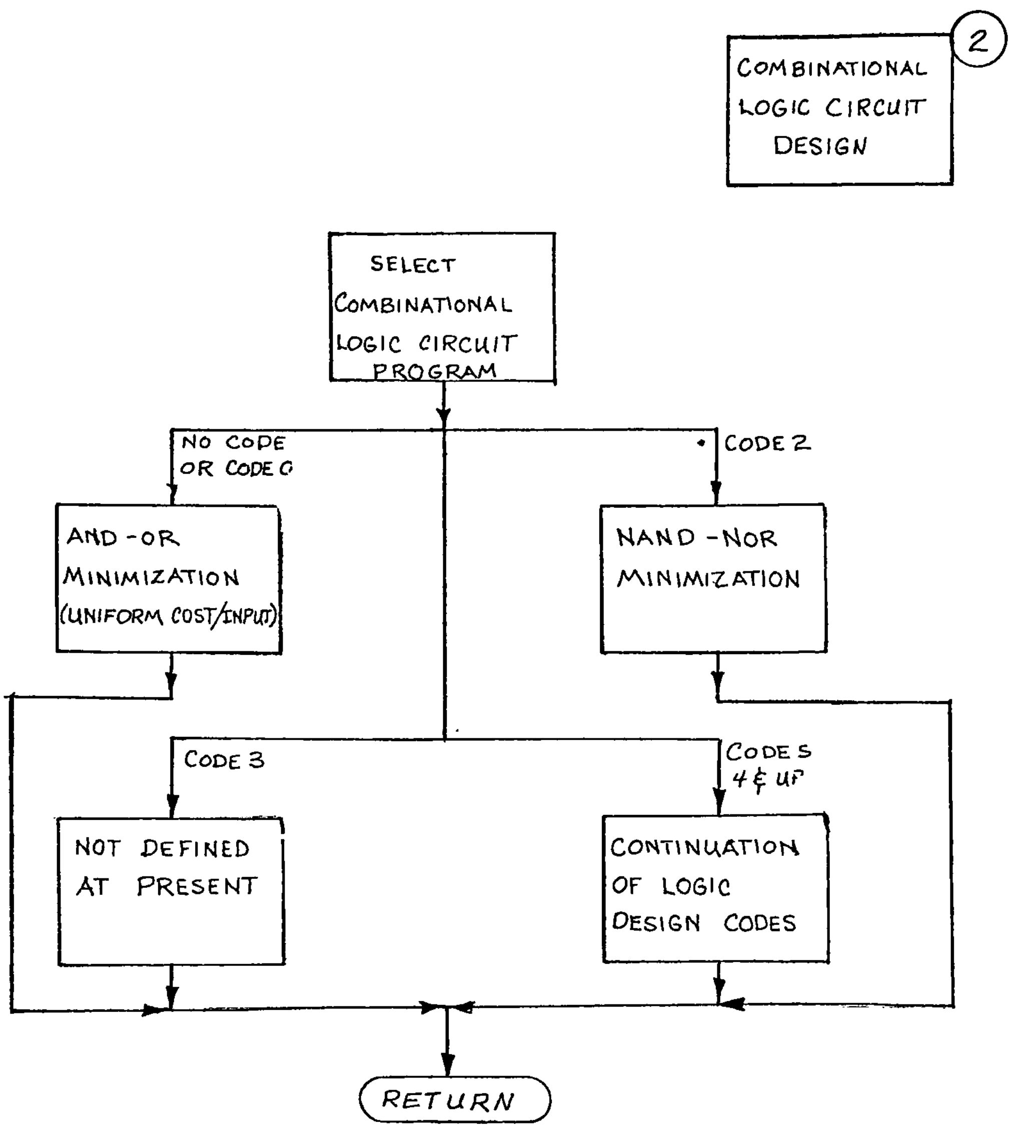

2
COMBINATIONAL LOGIC CIRCUIT DESIGN
SELECT COMBINATIONAL LOGIC CIRCUIT PROGRAM
NO CODE OR CODE 0
CODE 2
AND-OR MINIMIZATION (UNIFORM COST/INPUT)
NAND-NOR MINIMIZATION
CODE 3
CODES 4 & UP
NOT DEFINED AT PRESENT
CONTINUATION OF LOGIC DESIGN CODES
RETURN

Figure 30

Combinational Logic Circuit Design

3.1.2 Input Interface Specification

This includes, unless optional, the coding specifications; voltage levels; drive capability; timing; rise and fall times; required "don't care" timing zones and predicted input error rate. The required items to be provided as input specifications will be determined by the supervisory routine. It is to be noted that this is an automatic logic design program and would, for example, consider twelve parallel lines from the read head on a card reader as an input. It does not consider the detailed mechanical design of the card transport even though it is part of the system. As this program contains extensive cost effectiveness provisions, other electronic devices such as magnetic tape read amplifiers are handled in a manner similar to the logic units. Mechanical and other units are handled as lumped cost units (i.e. several alternate electro-mechanical card read heads could be automatically considered as to their overall effectiveness on system performance and cost, including optimizing the logic design for each. However, their individual designs would not be developed by the computer program).

3.1.3 Output Interface Specification

This includes, unless optional, the coding specifications of the output device; voltage levels, drive requirements, timing, rise and fall times, allowable "don't care" timing zones, and numbers of lines. Required error rates are covered as part of the General Specifications of item five.

3.1.4 Lumped Cost Items

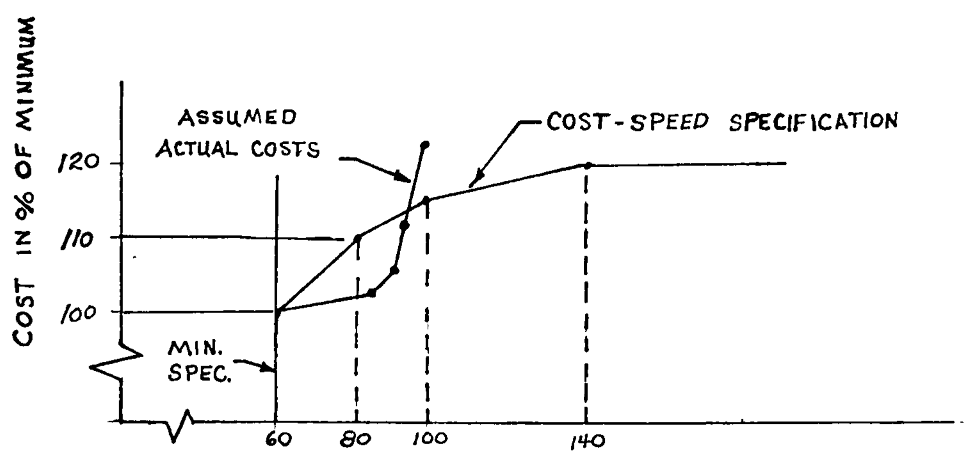
There are items which have a very direct affect on the logic design but are not logic elements. Examples of this are magnetic read and write heads and power supplies. For example, by using different types of logic the amount of power used and the cost of the power supply is greatly affected. Alternately, under different power drains and power supply tolerances the maximum reliable speed of the same logic elements will vary considerably. The specifications of the lumped cost items to be considered are called from a library by entering their identification number. These specifications, along with the General Specification, are used to determine an optimized unit selection and to optimize the mode of use.

3.1.5 General Specification

The General Specification basically states what the machine being designed is to do with the input before putting its results on the output lines. This is accomplished by providing the General Specification program a list of inputs of alpha-numeric symbols, special characters and commands for the machine being designed. It is to be noted that commands are specified to the program by their library number. When there are specific input-output specifications (items 3.1.2 and 3.1.3) associated with a command line, reference to these specifications is included here. As an example of a command specification, consider the case of specifying the command of multiplication. Assume the General Specification code for the class of multiplications to be considered in our example has a library number X01. Also assume that it is desired to provide as a built in function the multiplication of the contents of register 02 by the contents of register 01 with the results placed in 02. Assume registers 01 and 02 are registers previously requested to be implemented as output registers. Giving the command "X01, 02, 01, 02, YYY" is all that is necessary to provide for the design of this function. YYY is a command identification number. The library program X01 will automatically provide all additional information to upgrade the logic control of register 01 and 02 from output registers to arithmetic registers or provide a more desirable alternate. The type of arithmetic, error checking, error correction, precision, associated index registers and control logic are also automatically provided by the library program. The overall accuracy of computation is also entered as part of the General Specification. This may be overridden for any specific command where it would be desired.

3.1.6 Detailed Specification

Two items are specified as detailed specifications. Those are speed and reliability. Generally, there is a minimum speed requirement. This in turn determines the types of logic most appropriate and whether parallel or serial operations as well as, to a certain extent, whether synchronous or asynchronous operations are optimum. Very frequently, improved capability (speed) above the minimum specified is worth something but the percent increase in value per percent increase in speed will vary depending on application. In our example problem of Section 3.2, in the COMPUTE mode 0.2 seconds is just about as fast as a person can operate the keys. Therefore, if a person hits a divide key and it took 0.2 seconds before the answer was on the screen this would be satisfactory and going faster would be of no value. In the RUN mode, however, a whole series of arithmetic operations is most probably going to be performed before a display pause or DATA entry command is reached; therefore, an increase in speed would be of value. This increase in value is specified by stating the percent of increase in value that would result for a specific increase in speed. Provision is made for twenty specification points with either linear or logrithmetic interpolation between specification points. As an example it could be specified that a twenty five percent increase in speed is worth ten percent, fifty percent in speed, fifteen percent, and two hundred percent in speed, twenty percent in value with linear interpolation between points. The design program in this case would continue to increase the speed until the increase in cost equaled the above cost effectiveness specification curve. See Figure 31 for an example specification.



Slowest Operation Speed
Operations per Minute

Figure 31

Cost vs Speed Decision Data

In the above example, the design would be implemented for approximately 85 operations per minute based on the speed of the slowest operation. As different operations may increase the value of the machine differently for the same increase in speed, and as it is not always the slowest that should be the determining factor, provision is to be made to weight the different operations. All operations are grouped by the YYY command identification number provided by the user in the General Specification. Each YYY number may be given a different cost vs. speed specification and thereby each group of operations may be effectively weighted as determined by a market analysis. The reliability may be specified as an overall mean time between failure and/or as a probability or error on a single operation. Increased value from improving the reliability over the minimum specified is handled as a percentage in the same way as for the value of increased speed. The probability of error specification for a single operation may also be defined by the YYY coding.

3.1.7 Particular Specification

Here, items particular to a specific machine, such as options, are considered. To consider a built-in square-root operation to add value to one machine may be of so little value it would not be worth considering. However, in the case of our example computer of Section 3.2 the manufacturer may like to know how much this feature would add to the cost either as a model modification or as a plug-in unit. In the case of the plug-in unit he probably would like to know how much cost is added to units where the plug-in is not supplied. The particular specification is also used to provide marketing data on price vs. expected sales volume and manufacturing costs vs. volume. This volume data is specified similar to the way added value of the speed-cost data is specified. All cost data would be considered as a unit and a design implemented for the combination that produces the greatest value over cost (profit) for the total production. Additionally, this specification is used to provide cost data on items of fixed cost or those of only minor importance in the logic design, such as painting costs, packaging costs, etc. If any cost is significantly affected by the logic design it is provided in the lumped cost library and an appropriate optimization of logic to minimize the total cost is effected.

3.1.8 Output Specification

The program will provide the following information as requested:

* Statement as to the feasibility of meeting the specifications using components currently in the library
* Cost per Unit
* Projected Volume
* Total Cost
* Sales Value
* Projected Profit
* Materials List (parts used and item costs)
* Manufacturing Costs
* Reliability Data (overall and for all operations specified separately)
* Speed (for all classes of operations)

* Logic List (logic expressions defining the design)

* Connection/Wiring List

* Simulated Machine Program
  This program will allow testing the machine on another computer before manufacture to get a feel for its actual use.

* Any or all of the above may be provided for any of the options considered.

3.1.9 Program Outline

A completely modularized program approach is used because of its ease in expansion and development in future investigations. While a completely modularized program offers flexibility in development and ease of expansion, it also necessitates the user knowing the routines available and how to call them in detail if this function were not handled by a supervisory routine. The supervisory routine is determined by the user's machine type specification. This is done by entering a code which corresponds to a machine type. This code is also the library number of the supervisor that will process the program. The supervisor will call the appropriate input routines, component selection routines, logic development and minimization routines and the output routines. In cases where a routine is very simple, it will be incorporated directly into the supervisor. The overall program flow chart is shown in Figure 29. It may also be noted that the more complex supervisors will call upon other supervisory routines. A secondary calling arrangement is within the specific sub-program that does the calling. In such cases control is returned to the routine that did the calling rather than the main program.

In addition to specifying the machine type, the user may wish to further specify the type of problem. Consider the case of a combinational logic design problem. The problem could be one of simplifying an expression in terms of the least number of input lines for AND/OR logic, or it could be to find a minimum cost set of logic from a selected library of logic elements compatible with the equipment this item of logic is to be part of. Or, the problem may be to find the minimum cost logic design to meet a certain specified speed requirement. To allow this further definition of the problem the first data card contains a number of entries. The first column is the continuation instruction. A blank denotes this is the last problem. A "1" denotes another problem will follow. The next four columns are the library number of the machine specification. The types of machines to be implemented are assigned a library number at the time it is decided to incorporate a class of machine in the program. This arrangement allows unlimited expansion of the system without modifying the previous structure. The machine types to be assigned code numbers at this time are given below.

Table 9

Machine Types

Code	Type of Machine
1	Combinational Logic Circuit Design
2	Sequential Logic Circuit Design
3	Desk Top Type Computers
4	Data Buffers
5	Paper Tape Readers and Punches
6	Card Readers and Punches
7	Core Memories
8	Magnetic Tape Units
9	Disk Memories
10	Data Terminals
11	Medium Size Computers

Succeeding five column numbers denote sub-classification of the problem specification library. So that the user does not have to know or make a specification for options in which he is not interested, a standardizations of options is adopted. In this standardization a blank number denotes the program is to perform the design in the simplest way possible for this level and any remaining sub-levels of the specification. A "1" will always denote the most general solution (lowest cost solution) out of all possible solutions the program is set up to consider.

In the design of combinational logic a "0" or blank secondary code denotes a logic minimization for two level AND/OR logic with either gate type having its cost directly proportional to its number of inputs. Code "1" denotes the entire combinational library would be searched for a minimum cost mechanization meeting the circuit specification. Code "2" denotes NAND/NOR type logic having its cost directly proportional to its number of inputs. It is intended that codes "0" and "2" are for student use or what might be classified as a theoretical circuit specification. Code "3" and up are the production codes and specify libraries with information on propagation delay, rise and fall times, speed, reliability, drive capability and input loading. Variations of these parameters as a function of circuit loading, operating temperature, stray capacitance and power supply specification are included. Also costs of assembly, interconnection and test are included. Required don't care or "dead" times will automatically be calculated and integrated into meeting the overall specification. Codes "3" and up are based on various groupings of compatible manufacturer's logic lines which would include various combinations of number of inputs and number of gates for NAND/NOR, AND/OR, EXCLUSIVE OR, INVERTERS, etc. Flip-flops, shift registers and other devices with memory will be considered under the sequential circuit supervisory routines.

The program mechanization for the combinational logic design approach is shown in Figure 30. It is to be noted that if another program calls the combinational subprogram the calling program will automatically supply all the necessary specifications without any additional requirement on the part of the user. Even when the logic code is not code "1" a calling program may in sequence request a number of combinational logic design sub-program codes and then select the design which gives the best results. A sample problem was developed to test the program after it had been expanded and to indicate the type of problems to be considered. A description of this sample problem is presented in the following section.

3.2 Development of Sample Computer Design Problem

A sample computer specification was developed which would enable testing of the overall program after its major core sections have been written and are operative. This specification also illustrates the capabilities intended for the overall program. For our example, the case of a manufacturer who would like to have designed a general purpose desk top calculator-computer is considered.

The computer is to be able to add, subtract, multiply and divide as direct key entry operations. It is also to include program capability so that any other mathematical function such as the trigometric, inverse trigometric, or hyperbolic functions could be used as programable functions. Also, this machine is to provide the user with general purpose programing capability so that user-designed programs may be entered for repetitive calculations. Nine digits with adjustable decimal point are to be provided in the readout. The output is to be three registers displayed on a cathode ray tube. It has been decided this is to be a low cost machine limited to sixty four words of core memory for the main memory. The word size is to be nine decimal digits plus sign in a format to be defined by the program. The entry keys are described below:

Table 10

Example Computer Entry Keys

Keys	Function
0 through 9	Numeric Entry
+	Addition
-	Subtraction and Negative Number Entry
x	Multiplication
/	Division

In addition to the above mathematical entry keys the following programing functions are to be provided as direct single key entry.

Table 11

Programming Functions

Keys	Function
Move TO	Copies from one location of memory to another
Test	Similar to Fortran IF Statement
Repeats TO	Similar to Fortran DO Statement
GO TO	Similar to Fortran GO TO Statement
Data	Call for data entry
Clear	To clear registers or correct errors

Table 12

Control Functions

Switches:	
On-Off	Power On-Off
Program-Run-Compute	Mode selector
0-9 Thumb-wheel switch	Locates decimal point from far right to far left

The output consists of three registers on a cathode ray tube, each displaying nine digits plus sign and decimal point. Also, an error light is provided; when the error light is lit the keyboard locks except for the clear key.

In the compute mode the machine is to perform as a desk calculator. In this mode, the keys have the following functions. The three memory registers which are displayed are denoted 1, 2, and 3. The overall specification is as denoted in Table 13 as follows:

Table 13

Compute Mode Specification

Key	Compute Mode Detailed Specification
Numeric Keys	Enters number at right-most position in register 1 with an automatic shift with each entry. Displays error if more that nine numbers are entered, previous contents of register remain unchanged.
+	Adds the number in 1 to the number in 2, puts result in 2 and clears 1. Displays error on either positive or negative overflow, not changing 1 or 2.
-	Subtracts the number in 1 from the number in 2, puts the results in 2 and clears 1. Displays error on overflow, not changing 1 or 2.

Table 13 (cont)

x	Multiplies the number in 1 by the number in 2, puts the result in 2 and clears 1. Displays error on overflow, not changing 1 or 2.
/	Divides number in 2 by the number in 1, puts result in 2 and clears 1. Displays error on overflow or divide by zero, not changing 1 or 2.
X Move to Y	X and Y are register numbers. The contents of X are copied into Y. X is left unchanged. X, a two digit number, is entered first, then the Move To key is depressed at which time the words MOVE TO will be displayed in 1 to the right of the number X. Y is a two digit number that is entered last. Upon display of Y the memory transfer is complete. Register 1 will automatically be cleared with the next entry. If X or Y are not numbers corresponding to memory locations, an error will be displayed and no transfer takes place.
Clear X	Turns off the error light and frees the keyboard if needed. X is a three digit number. If X is 000 all registers (all of memory) are cleared. If a number corresponding to a memory location is entered that register or memory location is cleared. If X does not correspond to a memory position or 000 (i.e. 999) no registers are changed.

Test, Repeat To, Go To and Data have no effect (do nothing) in the compute mode.

In the program mode, programs may be written into the computer memory for later processing. In this mode the keys will have the following functions:

Table 14

Program Mode Arithmetic Operations

Key	Program Mode Detailed Specification
Numeric Keys	Enters number of memory location
X + Y X - Y X x Y X / Y	X and Y are memory locations. The indicated operation will be performed in the run mode. At that time the contents of X will be put in 2, the contents of Y in 1, the specified operation performed and the results put in 2. Register 1 will be cleared.

In the program mode all instructions are written in register 1 and shifted up as new instructions are entered. In this way the last two plus the current instruction will appear on the screen. The instructions are described in Table 15.

Table 15

Programing Functions Detailed

Key	Program Mode Detailed Specification
X Y Test Z	Transfers program operation to the Xth step entered if the contents of Z are negative and to the Yth step if positive. Zero is considered positive.
X Repeats to Y	Repeats the next series of steps through the Yth step X times.
Go to X	Unconditional transfer to the Xth step.
Data X	Denotes in the run mode the computer will pause for X items of data to be entered. X may be any two digit number other than 00. 00 is a display pause. In the run mode, depressing the data key denotes an item of data is entered. Data entered will be ignored if X is 00.
Clear X	If there is a program entry error the error light will light and the keyboard, other than the clear key, will lock. Clear turns off the error light and frees the keyboard if needed. If X is 000 all of memory is cleared. If X is any number corresponding to a program step number, that step is cleared. If X is anything else no registers are changed. The next entry will automatically clear register 1 (the display "CLEAR XXX").
X Move To Y	Same as in compute mode

In the run mode only the numeric keys, data key, and clear key will be used. These will be used to enter data as requested by the program and to make corrections of data in register 1. If a program error is uncovered, it is to be corrected by switching the toggle back to the program mode. The final specification for our test machine is that results should appear instantaneous to the user. By that is meant all single operations should be completed in less than 0.2 seconds.

CHAPTER 4. CONCLUSIONS

A two level AND-OR logic simplification program was developed that has certain advantages over other approaches which appear in the literature. This program should prove desirable for student use in solving logic simplification problems which are more extensive than those normally solved by the Quine-McCluskey method unaided by such a computer program.

A method has been illustrated for the structuring of an automatic design program for switching networks. This is still in its infancy, representing one of the major areas of the industry yet to be developed to anything approaching its full potential. This area holds one of the best futures for automation of design since much of the design task is centered around deductive type logic decisions and is highly repetitive. Both of these attributes are the leading prerequisites to an efficient solution by a digital computer. The only drawback is the large amount of actual development work required.

APPENDIX I

PROGRAM FLOW CHARTS

To enable use of less space in the computer memory the program was organized in phases which are overlaid during program operation. Phase one is the main program which has the function of calling the working routines. It is always in memory. Each of the other phases is overwritten as the next phase is called in. The program structure is shown in Figure 32. Subroutines called by another routine are shown under the calling routine. All the special subroutines in a program phase are shown under the phase title block. The main program flow chart is shown in Figure 33. Subprogram flow charts are shown in Figures 34 through 41. CONV1, CONV11, CONV12 and CONV13 are alike except for their use in different phases of the program. Also, CONV2 and CONV23 are alike. Flow charts for CONV1 and CONV2 are shown as Figures 40 and 41 respectively.

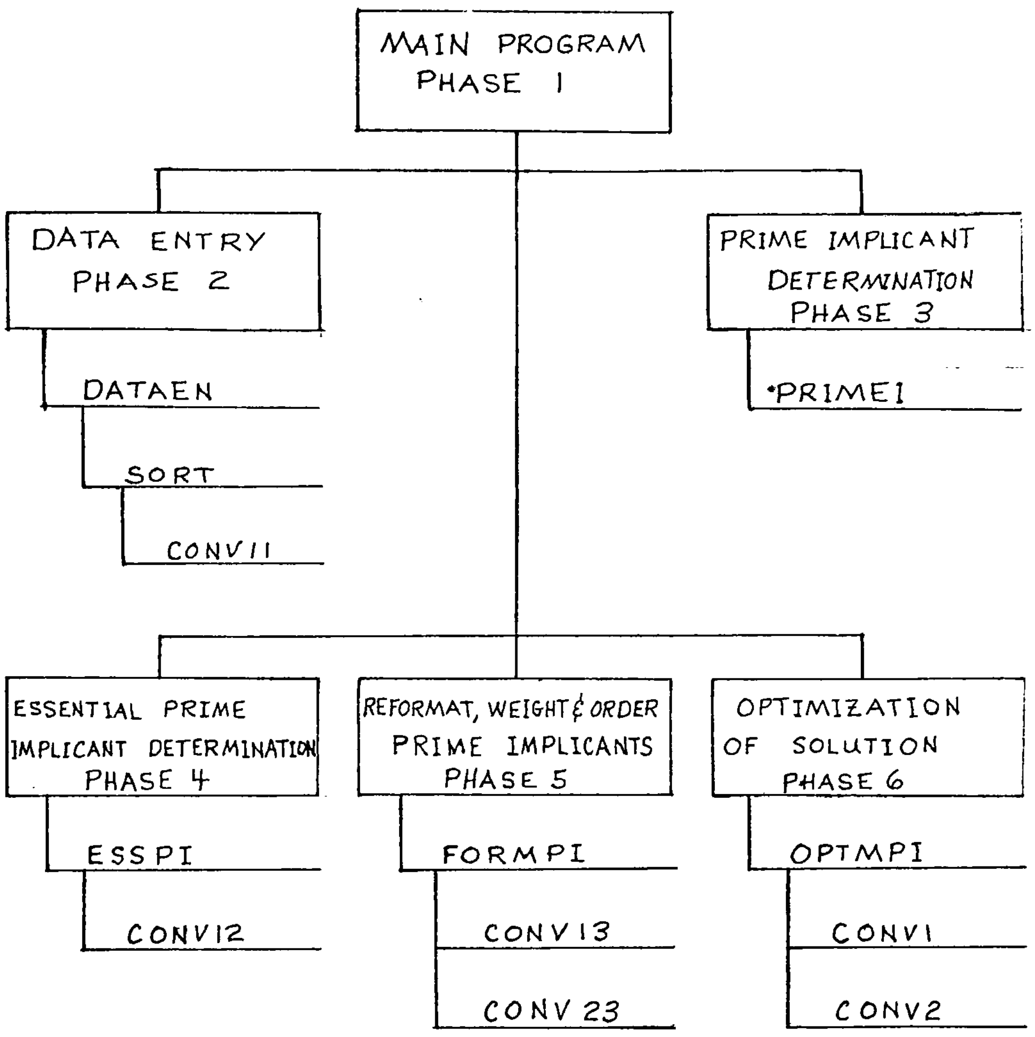

MAIN PROGRAM PHASE 1
DATA ENTRY PHASE 2
DATAEN
SORT
CONV11
PRIME IMPLICANT DETERMINATION PHASE 3
PRIME1
ESSENTIAL PRIME IMPLICANT DETERMINATION PHASE 4
ESSPI
CONV12
REFORMAT, WEIGHT & ORDER PRIME IMPLICANTS PHASE 5
FORMPI
CONV13
CONV 23
OPTIMIZATION OF SOLUTION PHASE 6
OPTMPI
CONV1
CONV2

Figure 32

Program Overlay Structure

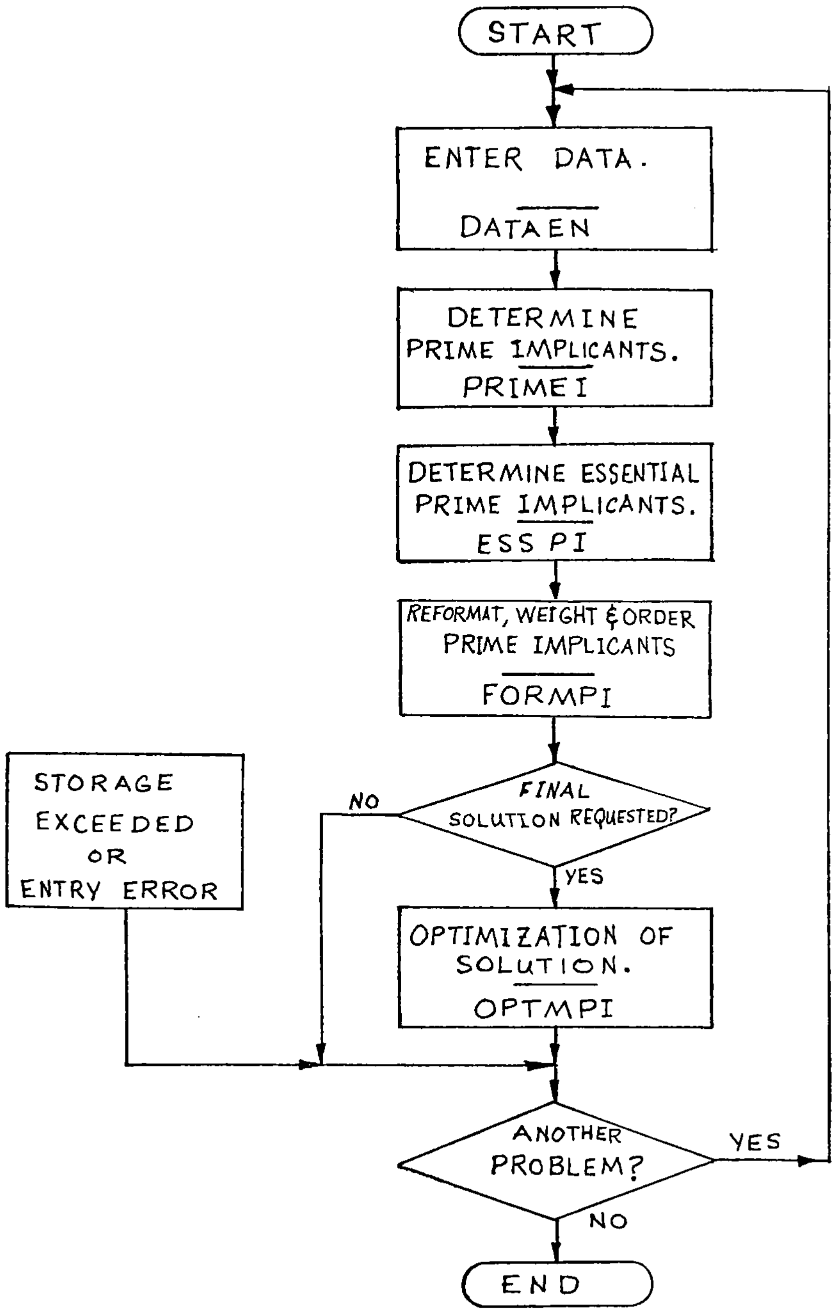

START
ENTER DATA.
DATAEN
DETERMINE PRIME IMPLICANTS.
PRIMEI
DETERMINE ESSENTIAL PRIME IMPLICANTS.
ESS PI
REFORMAT, WEIGHT & ORDER PRIME IMPLICANTS
FORMPI
FINAL SOLUTION REQUESTED?
NO
YES
OPTIMIZATION OF SOLUTION.
OPTMPI
STORAGE EXCEEDED OR ENTRY ERROR
ANOTHER PROBLEM?
YES
NO
END

Figure 33

Main Program Flow Chart

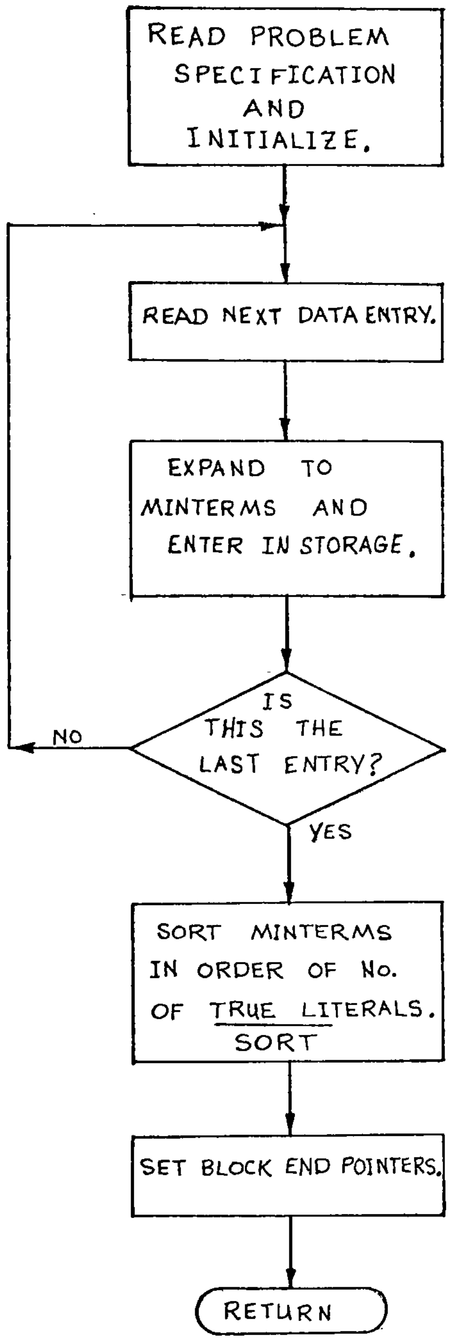

READ PROBLEM SPECIFICATION AND INITIALIZE.
READ NEXT DATA ENTRY.
EXPAND TO MINTERMS AND ENTER IN STORAGE.
IS THIS THE LAST ENTRY?
NO
YES
SORT MINTERMS IN ORDER OF No. OF TRUE LITERALS.
SORT
SET BLOCK END POINTERS.
RETURN

Figure 34

DATAEN Flow Chart

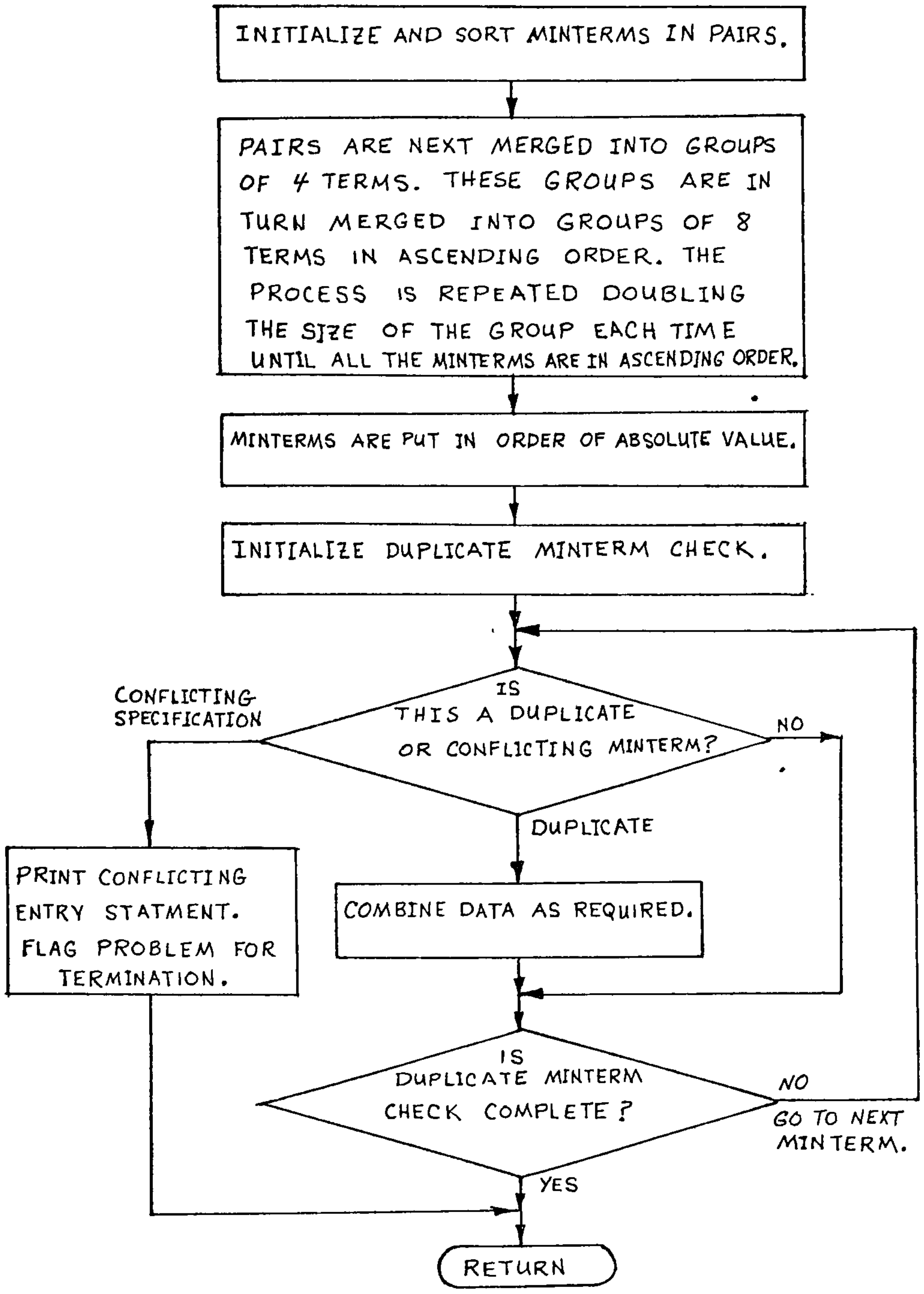

INITIALIZE AND SORT MINTERMS IN PAIRS.
PAIRS ARE NEXT MERGED INTO GROUPS OF 4 TERMS. THESE GROUPS ARE IN TURN MERGED INTO GROUPS OF 8 TERMS IN ASCENDING ORDER. THE PROCESS IS REPEATED DOUBLING THE SIZE OF THE GROUP EACH TIME UNTIL ALL THE MINTERMS ARE IN ASCENDING ORDER.
MINTERMS ARE PUT IN ORDER OF ABSOLUTE VALUE.
INITIALIZE DUPLICATE MINTERM CHECK.
IS THIS A DUPLICATE OR CONFLICTING MINTERM?
CONFLICTING SPECIFICATION
NO
DUPLICATE
PRINT CONFLICTING ENTRY STATMENT. FLAG PROBLEM FOR TERMINATION.
COMBINE DATA AS REQUIRED.
IS DUPLICATE MINTERM CHECK COMPLETE?
NO
GO TO NEXT MINTERM.
YES
RETURN

Figure 35

SORT Flow Chart

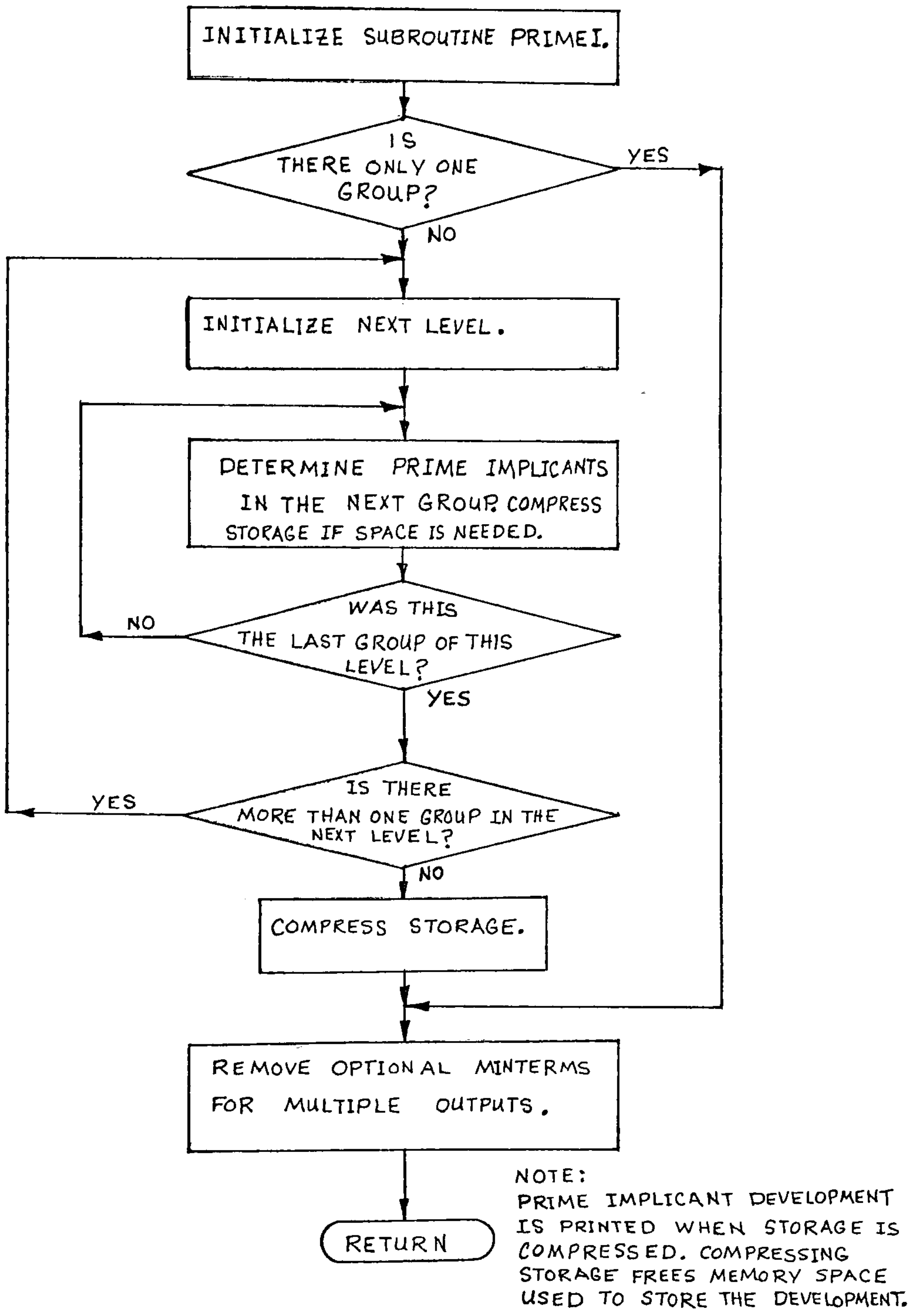

INITIALIZE SUBROUTINE PRIMEI.
IS THERE ONLY ONE GROUP?
YES
NO
INITIALIZE NEXT LEVEL.
DETERMINE PRIME IMPLICANTS IN THE NEXT GROUP. COMPRESS STORAGE IF SPACE IS NEEDED.
WAS THIS THE LAST GROUP OF THIS LEVEL?
NO
YES
IS THERE MORE THAN ONE GROUP IN THE NEXT LEVEL?
YES
NO
COMPRESS STORAGE.
REMOVE OPTIONAL MINTERMS FOR MULTIPLE OUTPUTS.
RETURN
NOTE:
PRIME IMPLICANT DEVELOPMENT IS PRINTED WHEN STORAGE IS COMPRESSED. COMPRESSING STORAGE FREES MEMORY SPACE USED TO STORE THE DEVELOPMENT.

Figure 36

PRIMEI Flow Chart

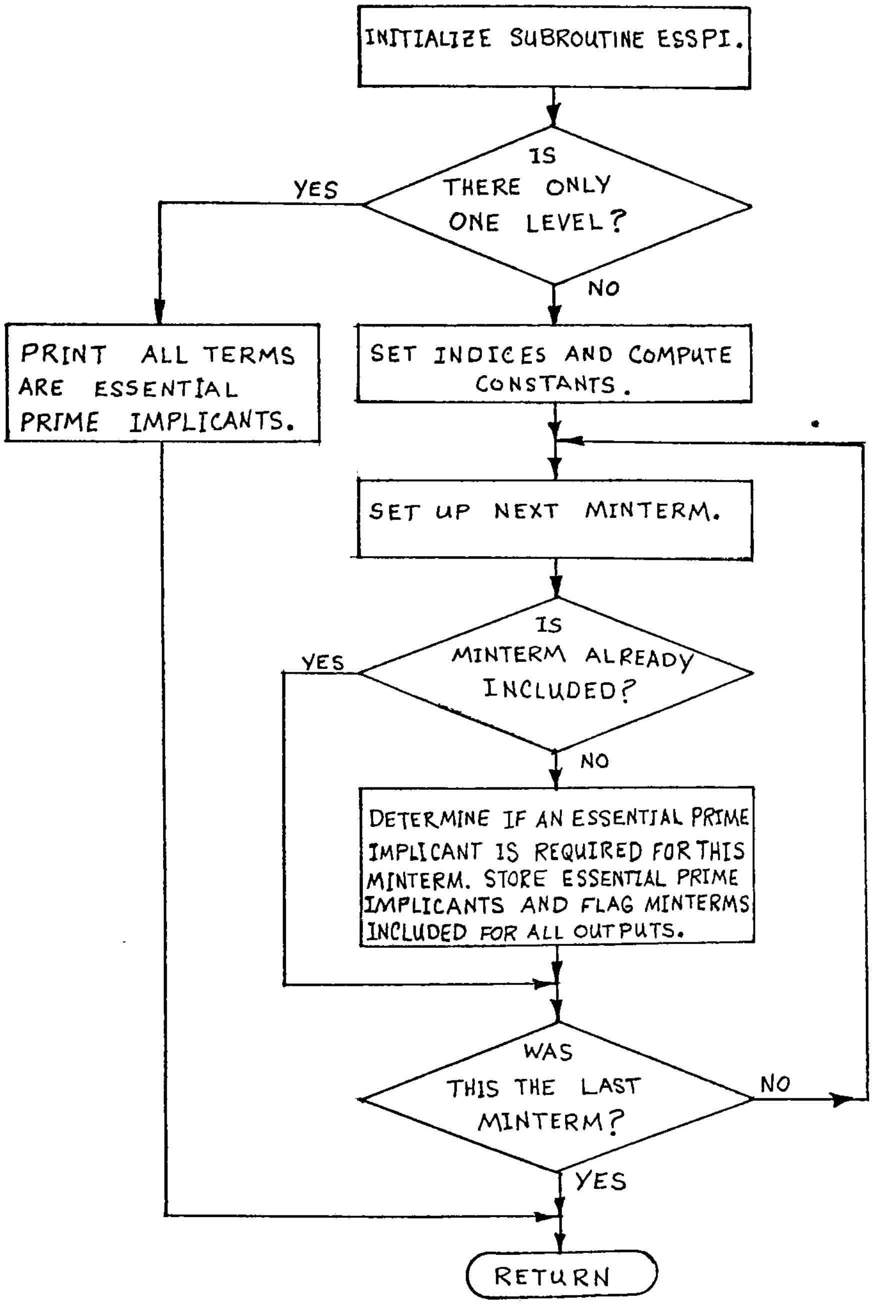

INITIALIZE SUBROUTINE ESSPI.
IS THERE ONLY ONE LEVEL?
YES
NO
PRINT ALL TERMS ARE ESSENTIAL PRIME IMPLICANTS.
SET INDICES AND COMPUTE CONSTANTS.
SET UP NEXT MINTERM.
IS MINTERM ALREADY INCLUDED?
YES
NO
DETERMINE IF AN ESSENTIAL PRIME IMPLICANT IS REQUIRED FOR THIS MINTERM. STORE ESSENTIAL PRIME IMPLICANTS AND FLAG MINTERMS INCLUDED FOR ALL OUTPUTS.
WAS THIS THE LAST MINTERM?
NO
YES
RETURN

Figure 37

ESSPI Flow Chart

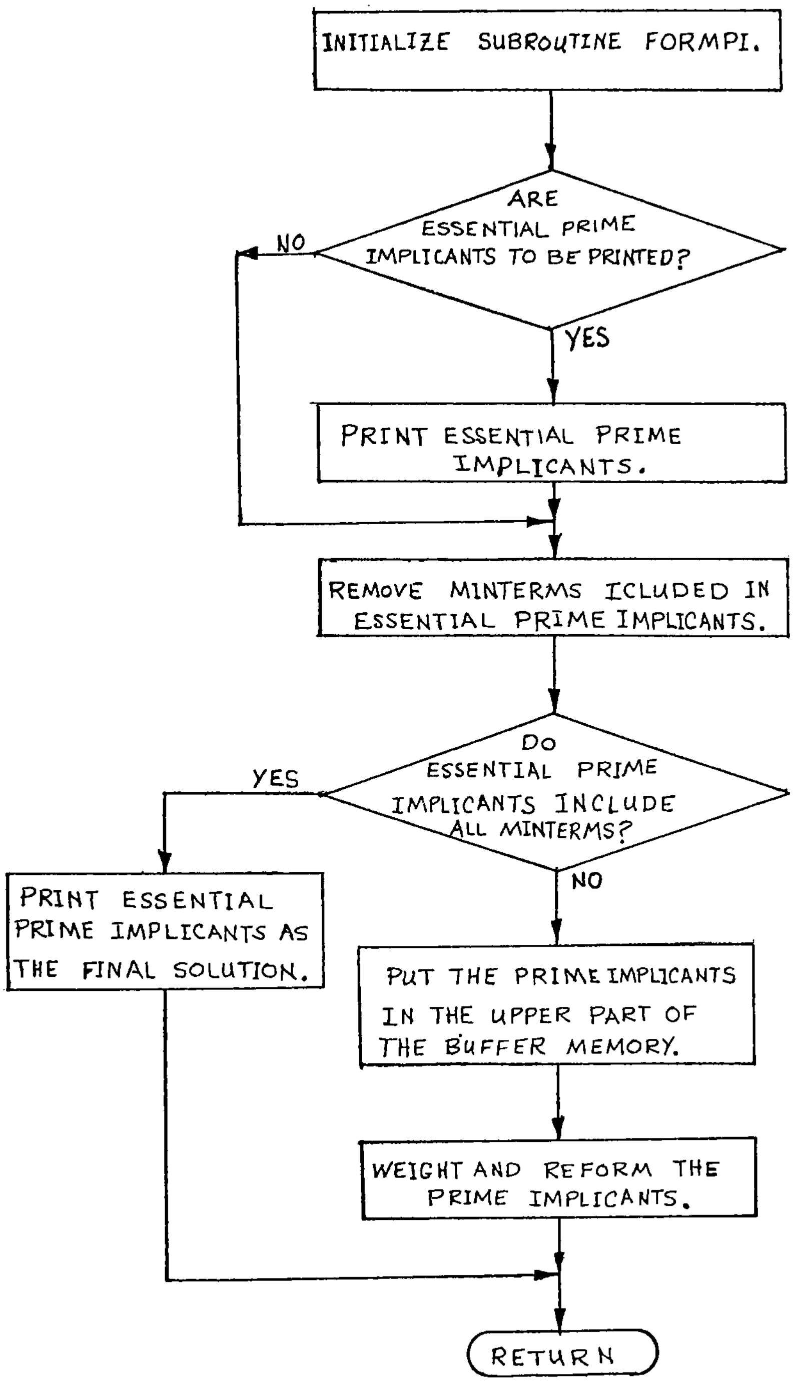

INITIALIZE SUBROUTINE FORMPI.
ARE ESSENTIAL PRIME IMPLICANTS TO BE PRINTED?
NO
YES
PRINT ESSENTIAL PRIME IMPLICANTS.
REMOVE MINTERMS ICLUDED IN ESSENTIAL PRIME IMPLICANTS.
DO ESSENTIAL PRIME IMPLICANTS INCLUDE ALL MINTERMS?
YES
NO
PRINT ESSENTIAL PRIME IMPLICANTS AS THE FINAL SOLUTION.
PUT THE PRIME IMPLICANTS IN THE UPPER PART OF THE BUFFER MEMORY.
WEIGHT AND REFORM THE PRIME IMPLICANTS.
RETURN

Figure 38

FORMPI Flow Chart

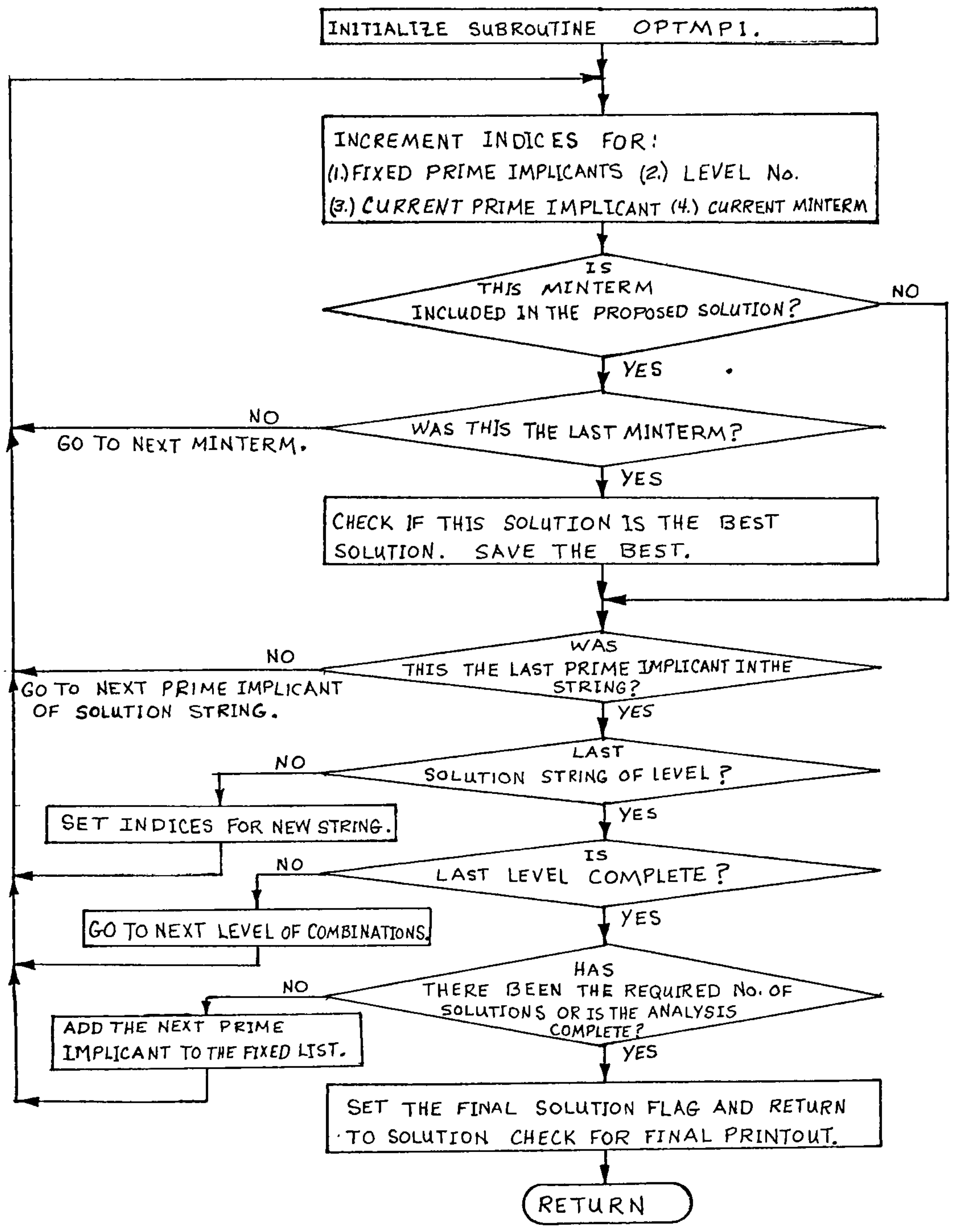

INITIALIZE SUBROUTINE OPTMPI.
INCREMENT INDICES FOR: (1.) FIXED PRIME IMPLICANTS (2.) LEVEL No. (3.) CURRENT PRIME IMPLICANT (4.) CURRENT MINTERM
IS THIS MINTERM INCLUDED IN THE PROPOSED SOLUTION?
NO
YES
WAS THIS THE LAST MINTERM?
NO
GO TO NEXT MINTERM.
YES
CHECK IF THIS SOLUTION IS THE BEST SOLUTION. SAVE THE BEST.
WAS THIS THE LAST PRIME IMPLICANT IN THE STRING?
NO
GO TO NEXT PRIME IMPLICANT OF SOLUTION STRING.
YES
LAST SOLUTION STRING OF LEVEL?
NO
SET INDICES FOR NEW STRING.
YES
IS LAST LEVEL COMPLETE?
NO
GO TO NEXT LEVEL OF COMBINATIONS.
YES
HAS THERE BEEN THE REQUIRED No. OF SOLUTIONS OR IS THE ANALYSIS COMPLETE?
NO
ADD THE NEXT PRIME IMPLICANT TO THE FIXED LIST.
YES
SET THE FINAL SOLUTION FLAG AND RETURN TO SOLUTION CHECK FOR FINAL PRINTOUT.
RETURN

Figure 39

OPTMPI Flow Chart

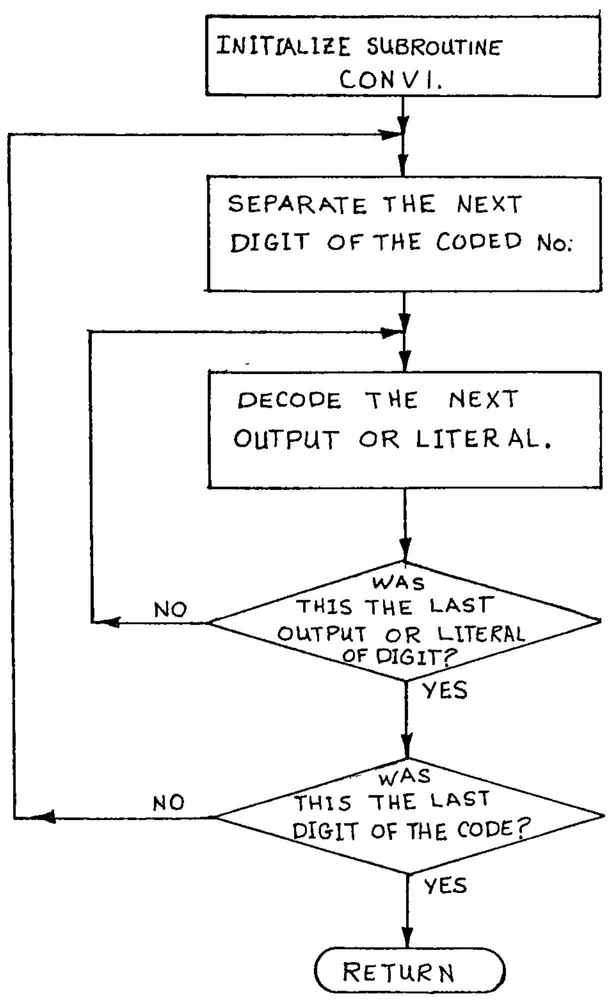

INITIALIZE SUBROUTINE CONV1.
SEPARATE THE NEXT DIGIT OF THE CODED No:
DECODE THE NEXT OUTPUT OR LITERAL.
WAS THIS THE LAST OUTPUT OR LITERAL OF DIGIT?
NO
YES
WAS THIS THE LAST DIGIT OF THE CODE?
NO
YES
RETURN

Figure 40

CONV1 Flow Chart

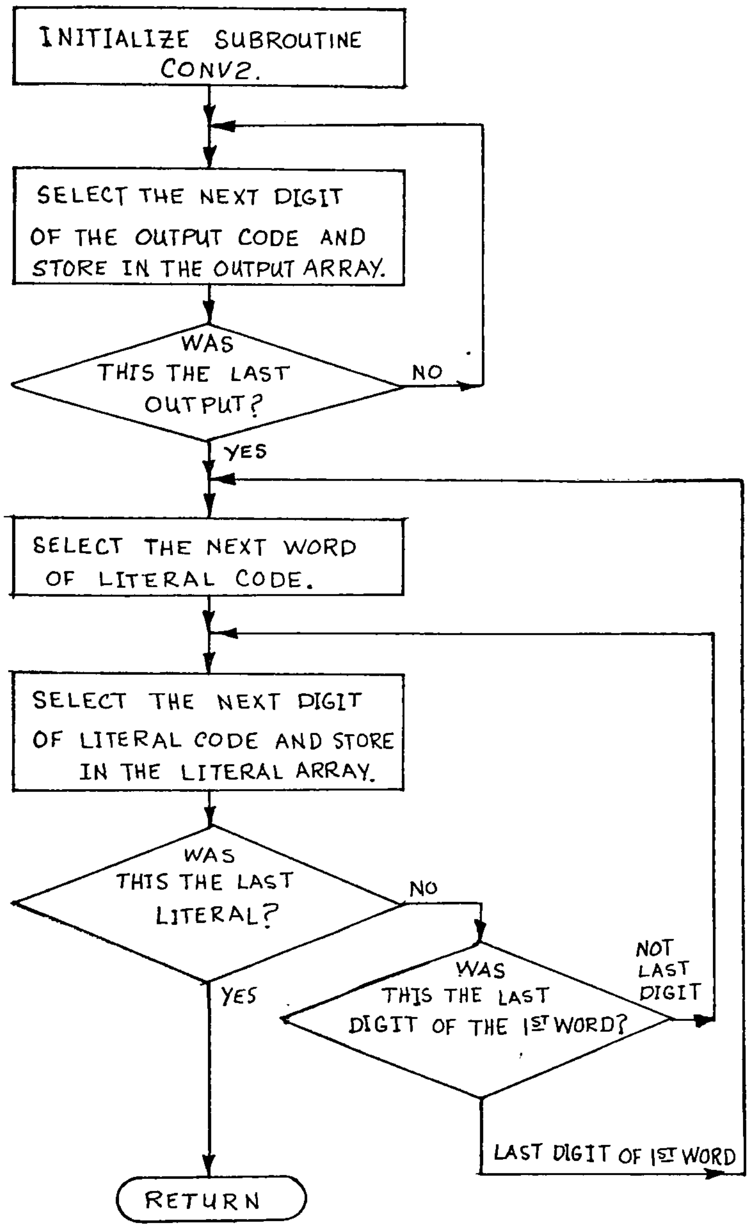

INITIALIZE SUBROUTINE CONV2.
SELECT THE NEXT DIGIT OF THE OUTPUT CODE AND STORE IN THE OUTPUT ARRAY.
WAS THIS THE LAST OUTPUT?
NO
YES
SELECT THE NEXT WORD OF LITERAL CODE.
SELECT THE NEXT DIGIT OF LITERAL CODE AND STORE IN THE LITERAL ARRAY.
WAS THIS THE LAST LITERAL?
NO
YES
WAS THIS THE LAST DIGIT OF THE 1ST WORD?
NOT LAST DIGIT
LAST DIGIT OF 1ST WORD
RETURN

Figure 41

CONV2 Flow Chart

APPENDIX II

PROGRAM LISTING

The computer program was written to run in FORTRAN IV under an IBM 360 DOS system with a minimum of 50K words of core memory. As the program is in a widely used language it would be adaptable to most systems of equivalent or larger size with a minimum of change. A source listing of the program statements follows.

```
DOS FORTRAN IV 360N-FO-479 3-3          MAINPGM            DATE    12/13/70

C       AUTOMATED LOGIC DESIGN PROGRAM
C
          COMMON IP.IT.IS.IU.L.M.IPROB.IW
          COMMON LB(2000).N(10).NB.LP(18).LXBI.LXB(100)
          COMMON NL.NO.LPID.LPI.LEPI.KOMP.LPOINT.LBLOCK.LLEVEL.IHL
          COMMON IPS(150).LM.LX(18).KPS
          COMMON    IHP.IFP.KOP(24).KOM(24).KP1.KC(24).KM
        1 FORMAT ('1'.45X.'AUTOMATED LOGIC DESIGN PROGRAM')
        2 FORMAT (I1.I4.I5)
        3 FORMAT ('1')
        4 FORMAT (' '/(' '.5I15))
        5 FORMAT ('1'/(' '.5I15))
          L=1
          M=3
          IU=1
          LXBI=0
      100 WRITE (M.1)
          IPROB=0

          READ (L.2)IP.IT.IS
          GO TO (101.992           ).IT
      101 J=IS+1
          GO TO (102.992     ).J
      102 CALL LINK (2)
          IF(IPROB)103.103.990
      103 CALL LINK (3)
          IF(IPROB)104.104.990
      104 CALL LINK (4)
          IF(IPROB)105.105.990
      105 CALL LINK (5)
          IF(IPROB)106.106.990
      106 IF(N(1))990.990.107
      107 CALL LINK (6)
      990 IF(IP)991.991.100
      991 WRITE (M.3)
          IF(N(1)-100)992.9911.992
     9911 WRITE(M.4)LB
          J=M**5
      992 CALL EXIT
          END
      SUBROUTINE DATAEN
      COMMON IP,IT,IS,IU,L,M,IPROB,IW
      COMMON LB(2000),N(10),NB,LP(18),LXBI,LXB(100)
      COMMON NL,NO,LPID,LPI,LEPI,KOMP,LPOINT,LBLOCK,LLEVEL,IHL
      COMMON IPS(150),LM,LX(18),KPS
      COMMON    IHP,IFP,KOP(24),KOM(24),KP1,KC(24),KM
      DIMENSION KV(25)
    1 FORMAT (35X,'AND-OR MINIMIZATION BASED ON A UNIFORM COST PER',
     1' INPUT'/'0INPUT DATA')
    2 FORMAT (2I5,3I1,2X,11I5)
    3 FORMAT (I1,I3,24I1)
    4 FORMAT(' ',I3,24I1)
    5 FORMAT(' NO. LITERALS=',I5/' NO. OUTPUTS=',I6/' NO. SOLUTIONS TO B
     1E CONSIDERED=',I5/' NO. OF PRIME IMPLICANTS TAKEN IN COMBINATIONS
     2OF'/'       2    3    4    5    6    7    8    9   10'/' ',9I5/
     3'0VARIABLE')
    6 FORMAT (' MORE THAN 9 ALL COMBINATION LITERALS USED')
    7 FORMAT(' MORE THAN 1000 MINTERMS USED')
   99 FORMAT (' '/(' ',5I15))
  100 NB=0
      WRITE (M,1)
      READ (L,2)NL,NO,LPID,LPI,LEPI,N,IW
      IF (N(1)) 1002,1001,1002
 1001 N(1)=25
      N(2)=30
      N(3)=15
      N(4)=12
      N(5)=10
      N(6)=10
      N(7)=10
      N(8)=10
      N(9)=10
      N(10)=10
      IW=4
 1002 WRITE (M,5)NL,NO,N
  101 READ(L,3)ID,KV
      J1=NL+NO+1
      WRITE (M,4)(KV(J),J=1,J1)
      NBLK=0
      J1=IABS(KV(1))
      KSIGN=KV(1)/J1
      KV(1)=J1
      KT=2000
      MT=2000
      LB(2000)=0
      K=1
  110 KL=(K-1)/3
      IF(KV(K)-2)111.117.113
  111 DO 112 J=MT.2000
  112 LB(J)=LB(J)+2**(K-1-3*KL)*10**KL+1000000
      GO TO 117
  113 IF(NBLK-9) 115.114.114
  114 WRITE (M.6)
      IPROB=1
      RETURN
  115 J1=2**NBLK
      NBLK=NBLK+1
      J2=2001-J1*2
      J3=MT-1
      MT=J2
      DO 116 J=J2.J3
      J4=J+J1
      LB(J)=LB(J4)
  116 LB(J4)=LB(J4)+2**(K-1-3*KL)*10**KL+1000000
  117 K=K+1
      IF(K-NL)110.110.118
  118 JB=NB+2**NBLK
      IF(JB-1000)120.120.119
  119 WRITE(M.7)
      IPROB=1
      RETURN
  120 IF(NO-1)121.121.122
  121 J1=1
      GO TO 130
  122 J1=0
      DO 129 J=1.NO
      K=K+1
      J2=KV(K)
      IF(J2)130,130,1221
 1221 GO TO (123,124,125,126,127,128),J2
  123 J1=J1+1
      GO TO 129
  124 J1=J1+2
      GO TO 129
  125 J1=J1+4
      GO TO 129
  126 J1=J1+10
      GO TO 129
  127 J1=J1+20
      GO TO 129
  128 J1=J1+40
  129 CONTINUE
  130 DO 131 J=MT,2000
      NB=NB+1
  131 LB(NB)=(LB(J)*100+J1)*KSIGN
  135 IF(ID)136,136,101
  136 CALL SORT
C
C     MARK BLOCKS
  171 KFM=IABS(LB(1))/100000000
      KS=2000
      J=1
  172 J=J+1
      IF(J-NB)173,173,175
  173 KF=IABS(LB(J))/100000000
      IF(KF-KFM)172,172,174
  174 LB(KS)=J-1
      KS=KS-1
      KFM=KF
      GO TO 172
  175 LB(KS)=NB
      IP(1)=KS
  176 RETURN
      END
      SUBROUTINE SORT
      COMMON IP,IT,IS,IU,L,M,IPROB,IW
      COMMON LB(2000),N(10),NB,LP(18),LXBI,LXB(100)
      COMMON NL,NO,LPID,LPI,LEPI,KOMP,LPOINT,LBLOCK,LLEVEL,IHL
      COMMON IPS(150),LM,LX(18),KPS
      COMMON    IHP,IFP,KOP(24),KOM(24),KP1,KC(24),KM
    8 FORMAT (' DUPLICATE MINTERM ENTRY',2I15)
   99 FORMAT (' '/(' ',5I15))
  136 DO 139 J=2,NB,2
      J1=LB(J-1)
      J2=LB(J)
      IF(J2-J1)137,139,139
  137 LB(J-1)=J2
      LB(J)=J1
  139 CONTINUE
  140 IHBS=2
      IBS=4
      KFM=1
      KFX=2
      KF=1
      KSM=3
      KSX=4
      KS=3
      KT=NB+1
  141 IF(LB(KF)-LB(KS))142,142,145
  142 LB(KT)=LB(KF)
      KT=KT+1
      KF=KF+1
  143 IF(KF-KFX)141,141,150
  145 LB(KT)=LB(KS)
      KT=KT+1
      KS=KS+1
  146 IF(KS-KSX)147,147,148
  147  IF(KS-NB)141,141,148
  148 J1=KFX-KF+1
       DO 149 J=1,J1
      LB(KT)=LB(KF)
      KT=KT+1
  149 KF=KF+1
  150 J3=KT-NB-1
      J1=KFM
      J2=NB+1
      DO 151 J=1.J3
      LB(J1)=LB(J2)
      J1=J1+1
  151 J2=J2+1
  152 KFX=KFX+IBS
      IF(KFX-NB)153.155.155
  153 KFM=KFM+IBS
      KF=KFM
      KSM=KSM+IBS
      KSX=KSX+IBS
      KS=KSM
      KT=NB+1
      GO TO 141
  155 IF(IBS-NB)156.160.160
  156 IHBS=IBS
      IBS=IBS+IBS
      KFM=1
      KFX=IHBS
      KF=1
      KSM=IHBS+1
      KSX=IBS
      KS=KSM
      KT=NB+1
      GO TO 141
  160 KS=1
  161 IF(LB(KS))162.163.163
  162 KS=KS+1
      GO TO 161
  163 IF(KS-1)1701.1701.1631
 1631 KF=KS-1
      KT=NB
  164 KT=KT+1
      IF(LB(KS)+LB(KF))166.165.165
  165 LB(KT)=LB(KF)
      KF=KF-1
      IF(KF)169.169.164
  166 LB(KT)=LB(KS)
      KS=KS+1
      IF(KS-NB)164.164.167
  167 J1=KF
      DO 168 J=1.J1
      KT=KT+1
      LB(KT)=LB(KF)
  168 KF=KF-1
  169 J1=NB+1
      J2=1
      DO 170 J=J1.KT
      LB(J2)=LB(J)
  170 J2=J2+1
 1701 J2=IABS(LB(1))/100
      J=1
 1702 J=J+1
      J1=J2
      J2=IABS(LB(J))/100
      IF(J1-J2)175.171.171
  171 KK1=IABS(LB(J-1))
      CALL CONV11(KK1.KOM.2)
      KK1=IABS(LB(J))
      CALL CONV11(KK1.KOP.2)
 1711 DO 1713 JN=1.NO
      IF(KOM(JN))1713.1713.1712
 1712 IF(KOP(JN))1713.1713.174
 1713 CONTINUE
 1714 J3=IABS(LB(J-1))
      JF=1
      JF3=J3
      DO 17145 JN=1.NO
      IF(KOP(JN)-KOM(JN))17142.17142.17141
17141 JF3=JF3+JF
17142 IF(JF-4)17144.17143.17144
17143 JF=10
      GO TO 17145
17144 JF=JF*2
17145 CONTINUE
      IF(LB(J-1))17146.17149.17149
17146 IF(LB(J))17149.17149.17147
17147 LXBI=LXBI+1
      LXB(LXBI)=IABS(LB(J-1))
```

```
       IF(LXBI-100)17149,17149,17148
17148  IPROB=1
17149  LB(J-1)=(LB(J-1)/J3)*JF3
       NB=NB-1
 1715  IF(LB(J-1))1716,1718,1718
 1716  IF(LB(J))1718,1718,1717
 1717  LB(J-1)=IABS(LB(J-1))
 1718  IF(J-NB)172,172,175
  172  DO 173 J4=J,NB
  173  LB(J4)=LB(J4+1)
       J=J-1
       J2=IABS(LB(J))/100
       GO TO 175
  174  IF(LB(J-1)/100-LB(J)/100)1741,1714,1741
 1741  WRITE(M,8)LB(J-1),LB(J)
       IPROB=1
  175  IF(J-NB)1702,176,176
  176  RETURN
       END
```

```
      SUBROUTINE CONV11(KK1.KK2.KK)
      COMMON IP.IT.IS.IU.L.M.IPROB.IW
      COMMON LB(2000).N(10).NB.LP(18).LXBI.LXB(100)
      COMMON NL.NO.LPID.LPI.LEPI.KOMP.LPOINT.LBLOCK.LLEVEL.IHL
      COMMON IPS(150).LM.LX(18).KPS
      COMMON    IHP.IFP.KOP(24).KOM(24).KP1.KC(24).KM
      DIMFNSION KK2(24)
      JKK1=KK1
      J=1
  100 DO 105 J1=1.KK
      JKK=JKK1/10
      JO=JKK1-JKK*10
      JKK1=JKK
  101 DO 105 J2=1.3
      JJ=JO/2
      KK2(J)=JO-JJ*2
      JO=JJ
  105 J=J+1
      RFTURN
      FND
      SUBROUTINE PRIMEI
      COMMON IP.IT.IS.IU.L.M.IPROB.IW
      COMMON LB(2000).N(10).NB.LP(18).LXBI.LXB(100)
      COMMON NL.NO.LPID.LPI.LEPI.KOMP.LPOINT.LBLOCK.LLEVEL.IHL
      COMMON IPS(150).LM.LX(18).KPS
      COMMON    IHP.IFP.KOP(24).KOM(24).KP1.KC(24).KM
      DIMENSION  KON1(6).KON2(6).LIST(20)
    1 FORMAT (' PROBLEM EXCEEDS MEMORY---END OF ANALYSIS')
    2 FORMAT ('1'.46X.'PRIME IMPLICANT DEVELOPMENT'/'OLEVEL 1')
    3 FORMAT (' ')
    4 FORMAT (' LEVEL'.I3)
    5 FORMAT (I15)
    6 FORMAT ('1'.48X.'PRIME IMPLICANT LISTING'/'OLEVEL  1')
    7 FORMAT('+ 0')
    8 FORMAT('+*')
    9 FORMAT('+'.I10.I3.I5.18I3)
   97 FORMAT('0',5I15)
   98 FORMAT ('OLOWER STORAGE'/(' '.5I15))
   99 FORMAT(' '/(' '.5I15))
  180 KL=0
      IHL=1
      KI=0
      KIN=0
      KB=0
      KOMP=0
      LPOINT=1
      LBLOCK=2000
      LLEVEL=1
      KNB=LP(1)
      IVF=1
      IF(KNB-2000) 181.299.299
  181 KT=KL
      KL=KL+1
      KLI=KIN-KI
      KI=(KT+9)/5
      KIN=(KT+10)/5
      KNL=KL+1
      IF(KL-1)190.190.182
  182 KB=2001-LP(KL-1)
  190 KB=KB+1
      IF(IVF)1902.1902.1901
 1901 KNB=KNB-1
```

```
      LB(KNB)=LB(KNB+1)
 1902 J=2001-KB
      KFX=LB(J)
      KSM=KFX+KI
      KSX=LB(J-1)
      LP(KNL)=KNB+1
      IVE=0
      IF(KB-1)191,191,192
  191 KFM=1
      KF=0
      GO TO 200
  192 KF=LB(J+1)-KLI
      KLI=0
      KFM=KF+KI
  195 IF(KSX-KSM)260,200,200
  200 KF=KF+KI
      IKF=IABS(LB(KF))
      KS=KSM
      IKS=IABS(LB(KS))
  201 IF(KT)205,205,202
  202 J1=KI-1
      DO 203 J=1,J1
      J2=KF+J
      J3=KS+J
      IF(LB(J2)-LB(J3))250,203,250
  203 CONTINUE
  205 J1=-1
      J2=(NI+8)/3
      JDIV=100
      DO 220 J=3,J2
      JF=IKF/JDIV
      JF=JF-(JF/10)*10
      JS=IKS/JDIV
      JS=JS-(JS/10)*10
      JD=JS-JF
      JDIV=JDIV*10
  206 IF(JD)250,220,207
  207 GO TO (208,209,250,210,250,250,250),JD
  208 MV=1
      GO TO 218
  209 MV=2
      GO TO 218
  210 MV=3
  218 MV=MV+(J-3)*3
      J1=J1+1
  219 IF(J1)220,220,250
  220 CONTINUE
  221 IF(NB+KIN+1-LP(KNL))240,222,222
  222 IF(KOMP)225,225,223
  223 WRITE (M,1)
      IPROB=1
      RETURN
  225 IR=-1
      GO TO 900
C     DETERMIN MATCHED PAIRS
C
  240 MO=0
      KON1(1)=1
      KON1(2)=2
      KON1(3)=4
      KON1(4)=10
      KON1(5)=20
      KON1(6)=40
      KON2(1)=10
      KON2(2)=10
      KON2(3)=10
      KON2(4)=100
      KON2(5)=100
      KON2(6)=100
      JJ=1
      DO 243 J=1,NO
      J1=(J-1)/3
      J2=KON2(J)
      JD=KON1(J)
      JF1=(IKF-(IKF/J2)*J2)/JD
      JF2=JF1-(JF1/2)*2
      JS1=(IKS-(IKS/J2)*J2)/JD
      JS2=JS1-(JS1/2)*2
```

```
              IF(JS2-JF2)241,242,241
         241  JJ=0
              GO TO 243
         242  MO=MO+JD*JS2
         243  CONTINUE
C
C             FLAG MATED PAIRS--FLAG IS NOTATION OF 20 ONES.
        2431  IF(JJ)2444,2444,244
         244  IF(IABS(LB(KF))-2000000000)2441,2442,2442
        2441  LB(KF)=(LB(KF)/IKF)*(IKF-(IKF/100000000)*100000000+2000000000)
        2442  IF(IABS(LB(KS))-2000000000)2443,2444,2444
        2443  LB(KS)=(LB(KS)/IKS)*(IKS-(IKS/100000000)*100000000+2000000000)
        2444  IF(MO)250,250,245
C
C             ENTER IN NEXT LEVEL
         245  IF(KL-1)2452,2452,2451
        2451  JD=(KL+3)/5
              JKF=KF+JD
              JF=100**(KL-2-5*(JD-1))
              IF(MV-LB(JKF)/JF)2452,250,250
        2452  KOMP=0
              NB=NB+1
              JKF=IKF-(IKF/100000000)*100000000
              LB(NB)=(JKF/100)*100+MO
              J2=KF
              J1=KIN-2
              IF(J1)248,248,246
         246  DO 247 J=1,J1
              J2=J2+1
              NB=NB+1
         247  LB(NB)=LB(J2)
         248  NB=NB+1
              JMV=10**(2*(KT-(KIN-2)*5))
              IF(JMV-1)249,249,2491
         249  LB(NB)=MV
              GO TO 2492
```

```
 2491 LB(NB)=LB(J2+1)+MV*JMV
 2492 LP(KNL)=KNB
      IVF=1
      LB(KNB)=NB-KIN+1
  250 IF(KS-KSX)251,255,255
  251 KS=KS+KI
      IKS=IABS(LB(KS))
      GO TO 201
  255 IF(KF-KFX)200,260,260
  260 IF(KB-2000+LP(KL))190,261,261
  261 J1=LP(KNL)
      J2=LP(KL)
      IF(J2-J1-2)262,181,181
  262 IF(J2-J1)263,263,264
  263 IHL=KI
      IR=0
      GO TO 900
  264 IHI=KNL
      IR=0
      GO TO 900
  299 IF(LXBI)310,310,300
  300 J1=1
      J2=0
      JF1=LXB(1)/100
      JF=JF1-(JF1/1000000)*1000000
  302 J2=J2+1
      JS=LB(J2)/100
      JS=JS-(JS/1000000)*1000000
  303 IF(JS-JF)302,305,305
  305 LB(J2)=LB(J2)-(LXB(J1)-100*JF1)
  306 IF(J1-LXBI)307,310,310
  307 J1=J1+1
      JF1=LXB(J1)/100
      JF=JF1-(JF1/1000000)*1000000
      GO TO 303
  310 RETURN
  900 IF(KB-1)9001,9001,9002
 9001 FPOINT=NB
      GO TO 9003
 9002 JKB=2002-KB
      FPOINT=LB(JKB)
 9003 IF(LPID)910,910,901
```

```
  901 IF(LPOINT-1)9011,9011,9012
 9011 WRITE (M,2)
 9012 J1=FPOINT
      IF(IR)9014,9013,9013
 9013 J1=NB
 9014 KPOINT=LPOINT
 9015 IF(KPOINT-LB(LBLOCK))902,902,9016
 9016 LBLOCK=LBLOCK-1
      WRITE(M,3)
 9017 IF(LBLOCK-LP(LLEVEL))9018,9015,9015
 9018 LLEVEL=LLEVEL+1
      WRITE (M,4)LLEVEL
      GO TO 9015
  902 J2=IABS(LB(KPOINT))
      J3=J2-(J2/100000000)*100000000
      WRITE (M,3)
      LIST(1)=J3/100
      LIST(2)=J3-LIST(1)*100
      IF (LB(KPOINT))9021,9022,9022
 9021 WRITE(M,7)
 9022 IF(J2-2000000000)9024,9023,9023
 9023 WRITE(M,8)
 9024 J2=LLEVEL-1
      J3=3
 9025 KPOINT=KPOINT+1
      J4=5
      J5=LB(KPOINT)
 9026 IF(J2)9029,9029,9027
 9027 IF(J4)9025,9025,9028
 9028 J6=J5/100
      LIST(J3)=J5-J6*100
      J2=J2-1
      J3=J3+1
      J4=J4-1
      J5=J6
      GO TO 9026
 9029 IF(J4-5)903,904,904
  903 KPOINT=KPOINT+1
  904 J3=J3-1
      WRITE (M,9)(LIST(J),J=1,J3)
  905 IF(KPOINT-J1)9015,9015,906
  906 IF(IR)910,910,299
  910 J=LP(1)
      K1=LB(J)+1
      IF(KL-1)930,930,9102
 9102 IF(K1-LPOINT)911,912,912
  911 K1=LPOINT
  912 IF(IR)9122,9121,9121
 9121 K2=NB
      GO TO 9123
 9122 K2=FPOINT
 9123 IB=LP(1)-1
      II=2
      INK=2
  913 IF(LB(IB)-K1)914,915,915
  914 IB=IB-1
      GO TO 913
  915 IF (LP(IL)-IB)920,920,916
  916 IL=II+1
      INK=(IL+8)/5
      GO TO 915
  920 IF(LB(K1)-2000000000)9201,921,921
 9201 K1=K1+INK
      GO TO 924
  921 JLAST=NB-INK
      JN=K1+INK
      DO 922 J=K1,JLAST
      LB(J)=LB(JN)
  922 JN=JN+1
      NB=NB-INK
      J1=LP(KNL)
      DO 923 J=J1,IB
  923 LB(J)=LB(J)-INK
      K2=K2-INK
  924 IF(K1-K2)913,913,929
  929 LPOINT=K2+INK
  930 IF(IR)221,931,931
  931 IF(LPI)299,299,932
  932 WRITE (M.6)
      KPOINT=1
      LBLOCK=2000
      LLEVEL=1
      IR=1
      J1=NB
      GO TO 9015
      END
      SUBROUTINE FSSPI
      COMMON IP.IT.IS.IU.L.M.IPROB.IW
      COMMON LB(2000).N(10).NB.LP(18).LXBI.LXB(100)
      COMMON NL.NO.LPID.LPI.LEPI.KOMP.LPOINT.LBLOCK.LLEVEL.IHL
      COMMON IPS(150).LM.LX(18).KPS
      COMMON    IHP.IFP.KOP(24).KOM(24).KP1.KC(24).KM
      DIMENSION KO(6).IEPI(6)
    1 FORMAT ('0 ALL INPUT ENTERIES ARE MINTERMS AND ESSENTIAL PRIME IMP
     1LICANTS')
    2 FORMAT ('0 MORE THAN 50 ESSENTIAL PRIME IMPLICANTS --------
     1STORAGE EXCEEDED'/(' '.2I9.I15))
    3 FORMAT(' '.5I15)
   93 FORMAT(' T6'/(' '.5I15))
   94 FORMAT(' T5'/(' '.5I15))
   95 FORMAT(' T4'/(' '.5I15))
   96 FORMAT(' T3'/(' '.5I15))
   97 FORMAT(' T2'/(' '.5I15))
   98 FORMAT(' T1'/(' '.5I15))
   99 FORMAT(' LX'/(' '.5I15))
  300 IF (IHL-1)3001.3001.3002
 3001 WRITE (M.1)
      IPROB=1
      WRITE(M.3)(LB(J).J=1.NB)
      GO TO 399
 3002 K=0
      KPS=0
      NX=(NI +2)/3+2
      KNT=1
      KML=6+NL
      NX1=(NO+2)/3
      DO 301 J=1.IHL
      J1=LP(J)
  301 LX(J)=LB(J1)
      LM=LX(1)
  302 K=K+1
      NP=1
      KP=LM
      KI =2
      KI X=LX(2)
      KMT=LB(K)
      KMT1=(KMT-(KMT/100000000)*100000000)/100
  303 IF(KMT)390.390.304
```

```
304 CALL CONV12(KMT,KOM,NX)
305 DO 308 J=1,NO
    IF(KOM(J))306,306,307
306 KO(J)=-1
    GO TO 308
307 KO(J)=1
308 CONTINUE
310 KP=KP+1
    KPI=LB(KP)
311 CALL CONV12(KPI,KOP,NX1)
314 IF(KLX-KP)315,316,316
315 KL=KL+1
    KLX=LX(KL)
    KNT=(KL+3)/5
    GO TO 314
316 DO 318 J=1,KNT
    KP=KP+1
    JP=LB(KP)
    DO 318 JJ=1,5
    JP1=JP-(JP/100)*100
    IF(KOM(JP1+6))318,318,317
317 JP2=(JP1-1)/3
    KPI=KPI+10**(2+JP2)*2**(JP1-JP2*3-1)
318 JP=JP/100
320 KPI1=(KPI-(KPI/100000000)*100000000)/100
    IF(KPI1-KMT1)340,325,340
325 JS=0
    DO 330 J=1,NO
    IF(KOP(J))330,330,326
326 KO(J)=KO(J)-1
    IF(KO(J))330,327,327
327 JS=1
330 CONTINUE
    IF(JS)332,332,335
332 DO 333 J=1,NO
    IF(KO(J))333,340,340
```

```
  333 CONTINUE
      GO TO 390
  335 IEPI(NP)=100*(KP-KNT)+KL
      NP=NP+1
  340 IF(KP-NB)310,341,341
C
C     SET UP PRIME IMPLICANT FOR ESSENTIAL PRIME IMPLICANT STORAGE
  341 NP=NP-1
 3411 IF(NP)375,375,342
  342 JI=IEPI(NP)
      KP=JI/100
      KPI=LB(KP)
      KL=JI-KP*100
      KNT=(KL+3)/5
      CALL CONV12(KPI,KOP,NX)
C     ADD REDUCED LITERALS
  343 DO 345 J1=1,KNT
      KP=KP+1
      JP=LB(KP)
      DO 345 J2=1,5
      JP1=JP-(JP/100)*100
      IF(JP1)346,346,344
  344 KOP(JP1+6)=2
  345 JP=JP/100
  346 KP=KP-KNT
C     TEST PRIME IMPLICANT FOR ESSENTIAL PRIME IMPLICANT STORAGE
  349 DO 370 J1=1,NO
      IF (KO(J1))370,350,370
  350 IF(KOP(J1))370,370,351
  351 KOP(J1)=2
      IF(KPS-150)353,352,352
  352 WRITE (M,2)(IPS(J+1),IPS(J),IPS(J+2),J=1,KPS,3)
      IPROB=1
      RETURN
C     PUT PRIME IMPLICANT IN ESSENTIAL PRIME IMPLICANT STORAGE
  353 IF (LB(KP)-2000000000)3531,3545,3545
 3531 J=7
      DO 354 J2=1,2
      JF=1
      KPS=KPS+1
      IPS(KPS)=0
      DO 354 J3=1,9
      IF(J-KML)3532,3532,354
 3532 IPS(KPS)=JF*(KOP(J)+1)+IPS(KPS)
      JF=JF*10
  354 J=J+1
 3541 KPS=KPS+1
      IPS(KPS)=0
      JF=1
      DO 3542 J2=1,NO
      IPS(KPS)=JF*(KOP(J2)+1)+IPS(KPS)
 3542 JF=JF*10
      IPS(KPS)=IPS(KPS)+KL*100000000
      GO TO 356
 3545 DO 3552 J2=1,KPS,3
      J3=7
      JP=IPS(J2)
 3546 JP1=JP/10
      JP2=JP-JP1*10
      IF (JP2-KOP(J3)-1)3552,3547,3552
 3547 J3=J3+1
      IF(J3-KML)3548,3548,3551
 3548 IF(J3-16)3549,3550,3549
 3549 JP=JP1
      GO TO 3546
 3550 JP=IPS(J2+1)
      GO TO 3546
 3551 JJ=J2+2
      GO TO 3555
 3552 CONTINUE
 3555 JP=IPS(JJ)
      J3=10**J1
      JP1=J3/10
      JP2=(JP-(JP/J3)*J3)/JP1
      IF(JP2-3)3556,356,356
 3556 IPS(JJ)=IPS(JJ)+JP1
C     FLAG PRIME IMPLICANT WITH 20 IN MOST SIGNIFICANT POSITION
  356 J=LB(KP)
```

```
      LB(KP)=J-(J/100000000)*100000000+2000000000
C     FLAG MINTERM OUTPUTS
  357 J=LB(K)
      J2=(J1-1)/3
      J=J-(10**J2)*(2**(J1-J2*3-1))
C     FLAG MINTERM IF ALL OUTPUTS ARE FLAGGED (NEGATIVE)
      IF (J-(J/100)*100)360,361,360
  360 LB(K)=J
      GO TO 370
  361 LB(K)=-1*J
  370 CONTINUE
  371 NP=NP-1
      IF(NP)390,390,342
C
C     ENTER MINTERM AS AN ESSENTIAL PRIME IMPLICANT
  375 J=7
      DO 376 J2=1,2
      JF=1
      KPS=KPS+1
      IPS(KPS)=0
      DO 376 J3=1,9
      IF(J-KML)3751,3751,376
 3751 IPS(KPS)=JF*KOM(J)
      JF=JF*10
  376 J=J+1
      KPS=KPS+1
      IPS(KPS)=0
      JF=1
      DO 380 J2=1,NO
      IPS(KPS)=JF*(2*KOM(J2)+1)
  380 JF=JF*10
      IPS(KPS)=IPS(KPS)+100000000
      LB(K)=-1*LB(K)
  390 IF(K-IM)302,399,399
  399 RETURN
      END
      SUBROUTINE CONV12(KK1.KK2.KK)
      COMMON IP.IT.IS.IU.L.M.IPROB.IW
      COMMON LB(2000).N(10).NB.LP(18).LXBI.LXB(100)
      COMMON NL.NO.LPID.LPI.LEPI.KOMP.LPOINT.LBLOCK.LLEVEL.IHL
      COMMON IPS(150).LM.LX(18).KPS
      COMMON     IHP.IFP.KOP(24).KOM(24).KP1.KC(24).KM
      DIMENSION KK2(24)
      JKK1=KK1
      J=1
  100 DO 105 J1=1.KK
      JKK=JKK1/10
      JO=JKK1-JKK*10
      JKK1=JKK
  101 DO 105 J2=1.3
      JJ=JO/2
      KK2(J)=JO-JJ*2
      JO=JJ
  105 J=J+1
      RETURN
      END
      SUBROUTINE FORMPI
      COMMON IP.IT.IS.IU.L.M.IPROB.IW
      COMMON LB(2000).N(10).NB.LP(18).LXBI.LXB(100)
      COMMON NL.NO.LPID.LPI.LEPI.KOMP.LPOINT.LBLOCK.LLEVEL.IHL
      COMMON IPS(150).LM.LX(18).KPS
      COMMON    IHP.IFP.KOP(24).KOM(24).KP1.KC(24).KM
      DIMENSION MTS(6.2)
    1 FORMAT ('1FSSENTIAL PRIMF IMPLICANTS'/'0',
     110X.'LITERALS'.8X.'OUTPUTS')
    2 FORMAT (' '.2I9.I15)
    3 FORMAT ('1ALL TERMS ARE COVERED BY THE ESSENTIAL PRIME IMPLICANTS'
     1/'0'.10X.'LITERALS'.8X.'OUTPUTS')
    4 FORMAT (' STORAGE EXCFEDED TOO MANY PRIME IMPLICANTS')
    5 FORMAT(' NO ESSENTIAL PRIME IMPLICANTS')
    6 FORMAT(' '.I18.I15)
   94 FORMAT(' T5'/(' '.5I15))
   95 FORMAT(' T4'/(' '.5I15))
   96 FORMAT(' T3'/(' '.5I15))
   97 FORMAT(' T2'/(' '.5I15))
   98 FORMAT(' T1'/(' '.5I15))
      IWW=IW*1000000
      NLIP=6+NL
      NX=(NLIP+2)/3
  400 IF(LEPI)405.405.401
  401 WRITE (M.1)
      IF(KPS)402.402.403
  402 WRITE(M.5)
      GO TO 405
  403 DO 404 J1=1.KPS.3
      J=IPS(J1+2)
      J=J-(J/1000000)*1000000
      IF(NL-9)4031.4031.4032
 4031 WRITE(M.6)IPS(J1).J
      GO TO 404
 4032 WRITE(M.2)IPS(J1+1).IPS(J1).J
  404 CONTINUE
      IF(IPROB)405.405.499
  405 IFP=LM+1
 4051 IF(KPS)4072.4072.4052
 4052 DO 4071 J1=1.KPS.3
      JP=IPS(J1)
      DO 4053 J2=7.15
      JP1=JP/10
      KOP(J2)=JP-JP1*10
 4053 JP=JP1
      JP=IPS(J1+1)
      DO 4054 J2=16,24
      JP1=JP/10
      KOP(J2)=JP-JP1*10
 4054 JP=JP1
      JP=IPS(J1+2)
      DO 4055 J2=1,6
      JP1=JP/10
      KOP(J2)=JP-JP1*10
 4055 JP=JP1
      J3=0
 4056 J3=J3+1
      IF(LB(J3))4070,4070,4057
 4057 KK1=LB(J3)
      CALL CONV13(KK1,KOM,NX)
 4058 DO 4060 J4=7,NLIP
      IF(KOP(J4)-KOM(J4)-1)4070,4060,4059
 4059 IF(KOP(J4)-3)4070,4060,4070
 4060 CONTINUE
 4061 DO 4069 J5=1,NO
      IF(KOM(J5))4069,4069,4062
 4062 IF(KOP(J5)-3)4069,4063,4069
 4063 JB=LB(J3)
      GO TO (40641,40642,40643,40644,40645,40646),J5
40641 JB=JB-1
      GO TO 4066
40642 JB=JB-2
      GO TO 4066
40643 JB=JB-4
      GO TO 4066
40644 JB=JB-10
      GO TO 4066
40645 JB=JB-20
```

```
             GO TO 4066
      40646 JB=JB-40
       4066 JJB=JB-((JB/100)*100)
            IF(JJB)4067,4067,4068
       4067 LB(J3)=-1*JB
            GO TO 4069
       4068 LB(J3)=JB
       4069 CONTINUE
       4070 IF(J3-LM)4056,4071,4071
       4071 CONTINUE
       4072 K=1
            DO 410 J=1,LM
            IF(LB(J))410,410,408
        408 LB(K)=LB(J)
            K=K+1
        410 CONTINUE
            LM=K-1
        411 IF(LM)412,412,415
        412 IPROB=1
            WRITE (M,3)
            GO TO 403
        415 JD=2000-LX(IHL)-(IHL+3)/5
            JDD=IFP-1
            IFP=IFP+JD
            DO 418 J=IFP,2000
            JDD=JDD+1
        418 LB(J)=LB(JDD)
            DO 419 J=1,IHL
        419 LX(J)=LX(J)+JD
        420 KL=2
            KNT=1
            KLX=LX(2)
            KPC=NL-1
            KP=IFP-1
            KP1=LM
            DO 4201 J1=1,NO
            KC(J1)=0
            DO 4201 J2=1,2
       4201 MTS(J1,J2)=0
            GO TO 441
C           NOTE 441 RETURNS TO 421 AFTER SETTING LEVEL
        421 KP=KP+1
      KP1=KP1+1
  422 KK1=LB(KP)
      CALL CONV13(KK1.KOP.8)
  423 DO 425 J1=1.KNT
      KP=KP+1
      JP=LB(KP)
      DO 425 J2=1.5
      JP1=JP-(JP/100)*100
      IF (JP1)430.430.424
  424 KOP(JP1+6)=2
  425 JP=JP/100
  430 J2=7
      DO 432 J=1.2
      JF=1
      LB(KP1)=0
      DO 431 J1=1.9
      LB(KP1)=(KOP(J2)+1)*JF+LB(KP1)
      J2=J2+1
  431 JF=JF*10
  432 KP1=KP1+1
      JF=1
      LB(KP1)=0
      DO 433 J=1.6
      LB(KP1)=(KOP(J)+1)*JF+LB(KP1)
  433 JF=JF*10
  434 J=KP-KNT
      IF(LB(J)-2000000000)436.437.437
  436 LB(KP1)=        KPC*100000000+LB(KP1)
  437 IF(KP1-1999)438.439.439
  438 IF(KP1-KP-2)440.439.439
  439 WRITE (M.4)
      IPROB=1
      GO TO 499
  440 IF(KP-2000)441.450.450
  441 IF(KP-KLX)421.442.442
  442 KL=KL+1
```

```
      KLX=LX(KL)
      KNT=(KL+3)/5
      KPC=KPC-1
      GO TO 441
  450 K=0
      IFP=LM+1
      IHP=KP1-2
      NB=KP1
  455 K=K+1
      J=LB(K)
      CALL CONV13(J.KOM.8)
      KP1=IFP-3
      DO 456 J=1.NO
  456 KC(J)=0
  460 KP1=KP1+3
  462 CALL CONV23
      J1=6+NL
  465 DO 469 J=7.J1
      IF (KOP(J)-KOM(J)-1)480.469.466
  466 IF(KOP(J)-3)480.469.480
  469 CONTINUE
  470 DO 478 J=1.NO
      IF (KOP(J)-1)478.478.471
  471 IF(KOM(J))478.478.472
  472 J8=L8(KP1+2)
      J1=JB/1000000
      J1=J1-(J1/100)*100
      IF(J1-99)473.475.475
  473 LB(KP1+2)=JB+1000000
  475 KC(J)=KC(J)+1
      J2=KC(J)
      IF(J2-2)476.476.478
  476 MTS(J.J2)=KP1+2
  478 CONTINUE
  480 IF(KP1-IHP)460.481.481
  481 DO 488 J=1.NO
      J1=KC(J)
      IF(J1-2)482.482.488
  482 IF(J1)488.488.483
  483 DO 487 JJ=1.J1
      J2=MTS(J.JJ)
      JB=LB(J2)
      J3=JB/1000000
      J3=J3-(J3/100)*100
      IF(J3-100+IW)485,484,484
  484 LB (J2)=LB(J2)+(99-J3)*1000000
      GO TO 487
  485 LB(J2)=LB(J2)+IWW
  487 CONTINUE
  488 CONTINUF
  489 IF(K-LM)455,490,490
  490 JF=IFP+2
      JL=IHP-1
      JL1=JL+3
      DO 496 J=JF,JL,3
      JF1=J+3
      JB=LB(J)
      JJB=JB-(JB/100000000)*100000000
      DO 493 J1=JF1,JL1,3
      JB1=LB(J1)
      JJB1=JB1-(JB1/100000000)*100000000
      IF(JJB1-JJB)493,493,491
  491 LB(J)=JB1
      LB(J1)=JB
      JB=JB1
      JJB=JJB1
      JT=LB(J-1)
      LB(J-1)=LB(J1-1)
      LB(J1-1)=JT
      JT=LB(J-2)
      LB(J-2)=LB(J1-2)
      LB(J1-2)=JT
  493 CONTINUE
      IF(JJB-999999)494,494,496
  494 IHP=J-5
      NB=IHP+2
      GO TO 499
  496 CONTINUE

  499 RETURN
      END
      SUBROUTINE CONV13(KK1.KK2.KK)
      COMMON IP.IT.IS.IU.L.M.IPROB.IW
      COMMON LB(2000).N(10).NB.LP(18).LXBI.LXB(100)
      COMMON NL.NO.LPID.LPI.LEPI.KOMP.LPOINT.LBLOCK.LLEVEL.IHL
      COMMON IPS(150).LM.LX(18).KPS
      COMMON    IHP.IFP.KOP(24).KOM(24).KP1.KC(24).KM
      DIMENSION KK2(24)
      JKK1=KK1
      J=1
  100 DO 105 J1=1.KK
      JKK=JKK1/10
      JO=JKK1-JKK*10
      JKK1=JKK
  101 DO 105 J2=1.3
      JJ=JO/2
      KK2(J)=JO-JJ*2
      JO=JJ
  105 J=J+1
      RETURN
      END
      SUBROUTINF CONV23
      COMMON IP.IT.IS.IU.L.M.IPROB.IW
      COMMON LB(2000).N(10).NB.LP(18).LXBI.LXB(100)
      COMMON NL.NO.LPID.LPI.LEPI.KOMP.LPOINT.LBLOCK.LLEVEL.IHL
      COMMON IPS(150).LM.LX(18).KPS
      COMMON    IHP.IFP.KOP(24).KOM(24).KP1.KC(24).KM
  100 JPI=LB(KP1+2)
      NLIP=NL+6
      DO 101 J=1.NO
      JPI1=JPI/10
      KOP(J)=JPI-JPI1*10
  101 JPI=JPI1
  105 J1=7
      J2=KP1
      DO 111 J=1.2
      JPI=LB(J2)
      DO 110 JJ=1.9
      JPI1=JPI/10
      KOP(J1)=JPI-JPI1*10
      IF(J1-NLIP)106.115.115
  106 JPI=JPI1
  110 J1=J1+1
  111 J2=J2+1
  115 RFTURN
      FND
      SUBROUTINE OPTMPI
      COMMON IP,IT,IS,IU,L,M,IPROB,IW
      COMMON LB(2000),N(10),NB,LP(18),LXBI,LXB(100)
      COMMON NL,NO,LPID,LPI,LEPI,KOMP,LPOINT,LBLOCK,LLEVEL,IHL
      COMMON IPS(150),LM,LX(18),KPS
      COMMON    IHP,IFP,KOP(24),KOM(24),KPI,KC(24),KM
      DIMENSION MC(10),MC1(10),KLP(10),KLP1(10),KBS(10)
    1 FORMAT ('1',52X,'PROBLEM SOLUTION')
    2 FORMAT ('OOUTPUT NO.',I2/'0   PRIME IMPLICANT      LIT WT OUTPUT')
    3 FORMAT (' ',2I9,I8,I3,I7)
    4 FORMAT ('ONO. OF ''AND'' GATE INPUTS REQUIRED =',I8/
     1' NO. OF ''OR'' GATE INPUTS REQUIRED  =',I8/' TOTAL=',I8/
     2'OTOTAL NO. OF SOLUTIONS=',I5/' THIS WAS THE',I5,' TH')
    5 FORMAT ('OESSENTIAL PRIME IMPLICANTS')
    6 FORMAT (' ',2I9,I15)
    7 FORMAT(' ',I18,I8,I3,I7)
    8 FORMAT(' ',2I18)
    9 FORMAT(' ',2I9,I18)
   71 FORMAT(' T01'/(' ',5I15))                                              TEST
   72 FORMAT(' T02'/(' ',5I15))
   73 FORMAT(' T03'/(' ',5I15))                                              TEST
   74 FORMAT(' T04'/(' ',5I15))                                              TEST
   75 FORMAT(' T05'/(' ',5I15))                                              TEST
   76 FORMAT(' T06'/(' ',5I15))                                              TEST
   77 FORMAT(' T07'/(' ',5I15))                                              TEST
   78 FORMAT(' T08'/(' ',5I15))                                              TEST
   79 FORMAT(' T09'/(' ',5I15))                                              TEST
   80 FORMAT(' T10'/(' ',5I15))                                              TEST
   81 FORMAT(' T11'/(' ',5I15))                                              TEST
   82 FORMAT(' T12'/(' ',5I15))                                              TEST
   83 FORMAT(' T13'/(' ',5I15))                                              TEST
   84 FORMAT(' T14'/(' ',5I15))                                              TEST
   85 FORMAT(' T15'/(' ',5I15))                                              TEST
   86 FORMAT(' T16'/(' ',5I15))                                              TEST
   87 FORMAT(' T17'/(' ',5I15))                                              TEST
   88 FORMAT(' T18'/(' ',5I15))                                              TEST
   89 FORMAT(' T19'/(' ',5I15))                                              TEST
   90 FORMAT(' T20'/(' ',5I15))                                              TEST
   91 FORMAT(' T21'/(' ',5I15))                                              TEST
   92 FORMAT(' T22'/(' ',5I15))                                              TEST
   93 FORMAT(' T23'/(' ',5I15))                                              TEST
   94 FORMAT(' T24'/(' ',5I15))                                              TEST
   95 FORMAT(' T25'/(' ',5I15))                                              TEST
```

```
      KBSC=999999999
      IFINAL=0
  500 IPC=IFP-6
      ISOL=0
      KLMAX=10
      NLIP=NL+6
      NLIT=(NL+2)/3+2
      J1=IFP-6
      MC(1)=IHP-3
      DO 501 J=2,10
  501 MC(J)=N(J)*3+J1
  505 IPC=IPC+3
      KL=0
      J1=0
      DO 509 J=1,10
      MC(J)=MC(J)+3
      IF(IHP-J1-MC(J))506,509,509
  506 IF(IHP-J1-IPC)507,507,508
  507 KLMAX=J-1
      GO TO 510
  508 MC(J)=IHP-J1
  509 J1=J1+3
  510 IF(N(1)-10)5101,5101,5102
 5101 MC(1)=MC(2)+3
 5102 KL=KL+1
      J1=KL*3
      DO 511 J=1,KL
      KLP(J)=IPC+J1
  511 J1=J1-3
  512 IF(KL-10)513,515,515
  513 DO 514 JL=KL,9
  514 KLP(JL+1)=-100
  515 KLX=MC(KL)
      KLP(1)=KLP(1)-3
  520 KLP(1)=KLP(1)+3
```

```
      KM=0
      KSN=1
  530 KM=KM+1
      KK1=LB(KM)
      CALL CONV1 (KK1.KC.NLIT)
  540 KP1=IFP
      KPB=KI
  541 IF(KP1-IPC)542.542.543
  542 CALL CONV2
      KP1=KP1+3
      GO TO 545
  543 IF(KPB)700.700.544
  544 KP1=KIP(KPB)
      CALL CONV2
      KPB=KPB-1
  545 DO 548 J=7.NLIP
      IF (KOP(J)-KC(J)-1)541.548.546
  546 IF(KOP(J)-3)541.548.541
  548 CONTINUE
  550 DO 552 J=1.NO
      IF(KOP(J)-KC(J)-1)552.551.552
  551 KC(J)=0
  552 CONTINUE
  553 DO 554 J=1.NO
      IF(KC(J))541.554.541
  554 CONTINUE
  555 IF(KM-LM)530.560.560
  560 IF(IFINAL)563.563.561
  561 IPC=K8S1
      DO 562 J=1.10
  562 KIP(J)=KBS(J)
      KI=KBSN
  563 NPIB1=(IPC-IFP)/3+1
      NPIB2=NPIB1+1
      NPIB=NPIB1+KL
      KO=0
      NSC=0
  564 KO=KO+1
      KMC=1
      IPC1=IFP-6
      NPIBA=NPIB
  565 IPC1=IPC1+3
```

```
      KL1=0
      MC1(1)=KLP(1)
      J=1
      J1=2
      J2=NPIB-NPIBA
  566 JN=NPIBA-J
      IF(JN)573.573.567
  567 IF(N(J1)-JN)568.569.569
  568 JN=N(J1)
  569 IF(JN-NPIB1+J2)570.570.571
  570 MC1(J1)=IFP+(JN-1+J2)*3
      GO TO 572
  571 JN=KL-(JN-NPIB2+J2)
      MC1(J1)=KLP(JN)
  572 J=J+1
      J1=J1+1
      IF(J1-10)566.566.573
  573 KLMAX1=J1-1
      NPIBA=NPIBA-1
  575 KL1=KL1+1
      KLX1=MC1(KL1)
      J=KL1
      JP=IPC1
      JL=KL
  576 IF(JP-IPC)577.578.578
  577 JP=JP+3
      KLP1(J)=JP
      GO TO 579
  578 KLP1(J)=KLP(JL)
      JL=JL-1
  579 J=J-1
      IF(J)5791.5791.576
 5791 IF(KL1-10)5792,580.580
 5792 DO 5793 J=KL1.9
 5793 KLP1(J+1)=-200
  580 IF(KLP1(1)-IPC)581.581.582
 581 KLP1(1)=KLP1(1)-3
     KPB=KI +1
     GO TO 585
 582 KPB=JI +2
 585 IF(KLP1(1)-IPC)590.595.595
 590       KP1=KLP1(1)+3
     KLP1(1)=KP1
     GO TO 600
 595 KPB=KPB-1
     KP1=KIP(KPB)
     KLP1(1)=KP1
 600 KM=0
     KSN1=1
 605 KM=KM+1
     KK1=LB(KM)
     CALL CONV1 (KK1.KC.NLIT)
 610 KP1=IFP
     KPB1=KL1
6101 IF(KC(KO))650.650.611
 611 KMC=0
6111 IF(KP1-IPC1)612.612.613
 612 CALL CONV2
     KP1=KP1+3
     GO TO 615
 613 IF(KPB1)620.620.614
 614 KP1=KIP1(KPB1)
     KPB1=KPB1-1
     CALL CONV2
 615  DO 617 J=7.NLIP
     IF(KOP(J)-KC(J)-1)611.617.616
 616 IF(KOP(J)-3)611.617.611
 617 CONTINUE
     IF(KC(KO)-KOP(KO)+1)650.650.611
 620 IF(KLP1(1)-KLP(1))585.625.625
 625 KSN1=KSN1+1
 626 IF(KLP1(KSN1)-KLP(KSN1))627.640.640
 627 IF(KLP1(KSN1))640.640.630
 630 JP=KLP1(KSN1)
     JL=KL
 631 IF(JP-IPC)632.633.633
 632 JP=JP+3
     GO TO 636
```

```
  633 IF(JP-KLP(JL))635,634,634
  634 JL=JL-1
      GO TO 633
  635 JP=KLP(JL)
      JL=JL-1
  636 KLP1(KSN1)=JP
      KSN1=KSN1-1
      IF(KSN1)580,580,631
  640 IF(KSN1-KL1)625,645,645
  645 IF(KL1-KLMAX1)575,565,565
  650 IF(KM-LM)605,655,655
  655 IF(KMC)6553,6553,6551
 6551 IF(IFINAL)665,665,6552
 6552 WRITE(M,1)
      WRITE(M,2)KO
      GO TO 661
 6553 NSC=NSC+KL1+(IPC1-IFP)/3+1
 6554 IF  (IFINAL)665,665,656
  656 WRITE(M,1)
      WRITE  (M,2)KO
      J=IFP
      JL=KL1
  657 IF(IPC1-J)658,660,660
  658 IF(JL)661,661,659
  659  J=KLP1(JL)
      JL=JL-1
  660 JO=LB(J+2)
      JO1=JO/1000000
      JOT=JO-JO1*1000000
      JF=1
      JFFF=NO
      DO 66000  JFF=1,JFFF
66000 JF=JF*10
66001 IF(JF-1000000)    66002,66003,66003
66002 JOT=JOT-JF
      JF=JF*10
      GO TO 66001
66003 JLI=JO1/100
      JWT=JO1-JLI*100
      JO=LB(J)
      JO1=LB(J+1)
      IF(NL-9)6601.6601.6602
 6601 JF=1
      JFFF=NL
      DO 66010 JFF=1.JFFF
66010 JF=JF*10
66011 IF(JF-1000000000)66012.66013.66013
66012 JO=JO-JF
      JF=JF*10
      GO TO 66011
66013 WRITE(M.7)JO.JLI.JWT.JOT
      GO TO 6603
 6602 JF=1
      JFFF=NL-9
      DO 66020 JFF=1.JFFF
66020 JF=JF*10
66021 IF(JF-1000000000)66022.66023.66023
66022 JO1=JO1-JF
      JF=JF*10
      GO TO 66021
66023 WRITE(M.3)JO1.JO.JLI.JWT.JOT
 6603 J=J+3
      GO TO 657
  661 IF(KPS) 665.665.662
  662 J=1
      J2=1
      DO 6621 JJ=1.KO
 6621 J2=J2*10
      J1=J2/10
      WRITE (M.5)
  663 JI=IPS(J+2)
      IF((JI-(JI/J2)*J2)/J1-3)6644.664.6644
  664 JO=IPS(J)
      JO1=IPS(J+1)
      JOT=IPS(J+2)
      JOT=(JOT-(JOT/1000000)*1000000)
      IF(NL-9)6641.6641.6642
 6641 WRITE(M.8)JO.JOT
```

```
      GO TO 6643
6642  WRITE(M,9)JO1,JO,JOT
6643  KBSC=KBSC+1
6644  J=J+3
      IF(J-KPS)663,665,665
 665  IF(KO-NO)564,666,666
 666  JPC=0
      DO 671 J=1,KL
      JP=KLP(J)
 671  JPC=JPC+LB(JP+2)/100000000
      IF( IPC-IFP)674,672,672
 672  JS=IFP+2
      JF= IPC+2
      DO 673 J=JS,JF,3
 673  JPC=JPC+LB(J)/100000000
 674  IF(KPS)677,677,675
 675  DO 676 J=3,KPS,3
 676  JPC=JPC-IPS(J)/100000000+NL+1
 677  NSC=NSC+JPC
6771  IF(IFINAL)680,680,6772
6772  JOR=KBSC-JPC
      WRITE (M,4)JPC,JOR,KBSC,ISOL,ISOL1
 678  RETURN
 680  ISOL=ISOL+1
 681  IF(NSC-KBSC)682,700,700
 682  KBSC=NSC
      KBSN=KL
      KBS1=IPC
      ISOL1=ISOL
      DO 683 J=1,10
 683  KBS(J)=KLP(J)
 700  IF(KLP(1)-MC(1))520,705,705
 705  KTLX=IHP
 706  KSN=KSN+1
      KTLX=KTLX-3
 710  IF(KLP(KSN)-MC(KSN))711,725,725
 711 IF(KLP(KSN))725,725,720
 720 JP=KLP(KSN)+3
     KLP(KSN)=JP
 721 KSN=KSN-1
 722 IF(KSN)724,724,723
 723 JP=JP+3
     KLP(KSN)=JP
     GO TO 721
 724 KLP(1)=KLP(1)-3
     GO TO 520
 725 IF(KSN-KL)706,727,727
 727 IF(KL-KLMAX)510,730,730
 730 IF(ISOL-N(1))735,740,740
 735 IF(IPC+10-IHP)505,740,740
 740 IFINAL=1
     GO TO 560
     END
```

```
      SUBROUTINE CONV1 (KK1,KK2,KK)
      COMMON IP,IT,IS,IU,L,M,IPROB,IW
      COMMON LB(2000),N(10),NB,LP(18),LXBI,LXB(100)
      COMMON NL,NO,LPID,LPI,LEPI,KOMP,LPOINT,LBLOCK,LLEVEL,IHL
      COMMON IPS(150),LM,LX(18),KPS
      COMMON     IHP,IFP,KOP(24),KOM(24),KP1,KC(24),KM
      DIMENSION KK2(24)
      JKKI=KK1
      J=1
  100 DO 105 J1=1,KK
      JKK=JKK1/10
      JO=JKK1-JKK*10
      JKK1=JKK
  101 DO 105 J2=1,3
      JJ=JO/2
      KK2(J)=JO-JJ*2
      JO=JJ
  105 J=J+1
      RETURN
      END
      SUBROUTINE CONV2
      COMMON IP,IT,IS,IU,L,M,IPROB,IW
      COMMON LB(2000),N(10),NB,LP(18),LXBI,LXB(100)
      COMMON NL,NO,LPID,LPI,LEPI,KOMP,LPOINT,LBLOCK,LLEVEL,IHL
      COMMON IPS(150),LM,LX(18),KPS
      COMMON    IHP,IFP,KOP(24),KOM(24),KP1,KC(24),KM
  100 JPI=LB(KP1+2)
      NLIP=NL+6
      DO 101 J=1,NO
      JPI1=JPI/10
      KOP(J)=JPI-JPI1*10
  101 JPI=JPI1
  105 J1=7
      J2=KP1
      DO 111 J=1,2
      JPI=LB(J2)
      DO 110 JJ=1,9
      JPI1=JPI/10
      KOP(J1)=JPI-JPI1*10
      IF(J1-NLIP)106,115,115
  106 JPI=JPI1
  110 J1=J1+1
  111 J2=J2+1
  115 RETURN
      END
```

REFERENCES

1. I. Copi; Symbolic Logic, MacMillin Company, 1954, p. 326.

2. E. V. Huntington; "Sets of Independent Postulates for the Algebra of Logic," Transactions of the American Mathematical Society, Vol. 5, 1904, p. 288.

3. G. Boole; The Mathematical Analysis of Logic, Cambridge, England, 1847 (reprinted in 1948, Oxford, Basil Blackwell).

4. G. Boole; An Investigation of the Laws of Thought, London, 1854.

5. S. H. Caldwell; Switching Circuits and Logic Design, John Wiley & Sons, Inc., New York, 1958.

6. C. E. Shannon; "A Symbolic Analysis of Relay Switching Circuits," Trans. AIEE, Vol. 57, 1938, pp. 713-723.

7. C. E. Shannon; "The Synthesis of Two Terminal Switching Circuits", Bell System Technical Journal, Vol. 28, No. 1, January, 1949, pp. 59-98.

8. Staff of the Computation Labratory; Synthesis of Electronic Computing and Control Circuits, Harvard University Press, Cambridge, Massachusetts, 1951.

9. W. V. Quine; "The Problem of Simplifying Truth Functions," American Mathematical Monthly, Vol. 59, October, 1952, pp. 521-531.

10. W. V. Quine; "A Way to Simplify Truth Functions," American Mathematical Monthly, Vol. 62, November, 1955, pp 627-631.

11. E. J. McCluskey; "Algebraic Minimization and the Design of Two-Terminal Contact Networks," Doctoral Thesis, Dept. of Electrical Engineering, Massachusetts Institute of Technology, June, 1956.

12. E. J. McCluskey; First part of the above Doctoral Thesis (Ref. 11) Bell System Technical Journal, Vol. 35, Nov., 1956, pp. 1417-1444.

13. I. B. Pyne and E. J. McCluskey; "An Essay on Prime Implicant Tables," J. Society Ind. Applied Math., Vol. 9, December, 1961, pp. 604-631.

14. I. B. Pyne and E. J. McCluskey; "The reduction of Redundancy in Solving Prime Implicant Tables," IRE Trans. on Electronic Computers, Vol. EC-11 pp. 473-482, August, 1962.

15. J. F. Gimpel; "A Reduction Technique for Prime Implicant Tables," 1964 Proc. Fifth Annual Symp. on Switching Theory and Logical Design pp. 183-191.

16. J. F. Gimpel; "A Method of Producing Boolean Functions Having an Arbitrarily Prescribed Prime Implicant Table," IEEE Trans. on Electronic Computers, Vol. EC-14, pp. 485-488, June, 1965.

17. F. Luccio; "A Method for the Selection of Prime Implicants," IEEE Transactions on Electronic Computers, Vol. EC-15, pp. 205-212. April, 1966.

18. E. W. Veitch; "A chart Method for Simplifying Truth Functions," Proceedings of Association for Computing Machinery; Pittsburg, Pennsylvania Meeting May 2 and 3, 1952, pp. 127-133.

19. M. Karnaugh; "The Map Method for Synthesis of Combinational Logic Circuits," AIEE Trans. Part I Communications and Electronics, Vol. 72, November, 1953, pp. 593-599.

20. M. Minsky; "Steps Toward Artificial Intelligence," Proceedings of the IRE, Vol. 49 No. 1, pp. 8-30, January, 1961.

21. A. Newell and H. A. Simon; "The Logic Theory Machine," IRE Trans. on Information Theory, Vol. if-2, September, 1956.

22. A. Newell, J. C. Shaw and H. Simon; "Empirical Exploration of the Logic Theory Machine," Proc. WJCC pp. 218-230, 1957.

23. H. Wang; "Toward Mechanical Mathematics" IBM J. Res. & Dev., Vol. 4 pp. 2-22, January, 1960.

24. M. A. Breuer; "General Survey of Design Automation of Digital Computers," Proc. IEEE December, 1966, pp. 1708-1721.

25. E. J. McCluskey; "Review of the above paper,"(24) Transactions of the IEEE November, 1967.

BIBLIOGRAPHY

T. L. Booth; Sequential Machines and Automa Theory John Wiley 1967.

L. Brillouin; Science and Information Theory, Academic Press 1956.

W. B. Davenport and W. L. Root; Random Signals and Noise, McGraw-Hill, 1956.

R. M. Fano; Transmission of Information, MIT Press, 1961.

F. C. Hennie; Finite State Models for Logical Machines, John Wiley and Sons, 1968.

V. L. Landing and R. H. Battin; Random Process in Automatic Control, McGraw-Hill, 1956.

E. J. McCluskey; Introduction to the Theory of Switching Circuits, McGraw-Hill, 1956.

M. P. Marcus; Switching Circuits for Engineers, Prentice-Hall, 1962.

R. E. Miller; Switching Theory Vol. I and II, Wiley, 1965.

B. Ostle; Statistics in Research, Iowa State College Press, 1954.

W. W. Peterson; Error Correcting Codes, MIT Press, 1961.

M. Phister; Logic Design of Digital Computers, Wiley, 1958.

F. M. Reza; An Introduction to Information Theory, McGraw-Hill, 1961.

R. K. Richards; Arithmetic Operations in Digital Computers, Van Nostrand, 1955.

R. K. Richards; Digital Computer Components and Circuits, Van Nostrand, 1957.

M. Schwartz; Communication Systems and Techniques, McGraw-Hill, 1966.

C. V. Smith; Electronic Digital Computers, McGraw-Hill, 1959.

P. E. Wood, Switching Theory, McGraw-Hill, 1968.

Note: A quite extensive bibliography is presented by M. A. Breuer (24) listing by subject type 287 works.