CR-119850

MCR-70-38
(Supplement 2)

COPY NO. 6

FINAL REPORT

FOR

SUPPLEMENT TWO

FORMULATION

OF A

TELEMETRY COMPUTER PROGRAM

CONTRACT NAS8-24017

MAY 1971

Prepared For

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GEORGE C. MARSHALL SPACE FLIGHT CENTER
MARSHALL SPACE FLIGHT CENTER, ALABAMA 35812

Electronics Research Department
Martin Marietta Corporation
P. O. Box 179, Denver, Colorado 80201

MCR-70-38
(Supplement 2)

FINAL REPORT

FOR

SUPPLEMENT TWO

FORMULATION

OF A

TELEMETRY COMPUTER PROGRAM

CONTRACT NAS8-24017

MAY 1971

Keith H. Hill
Robert O. Leighou
Duane L. Starner

Robert O. Leighou
Robert O. Leighou
Program Manager

Prepared For

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GEORGE C. MARSHALL SPACE FLIGHT CENTER
MARSHALL SPACE FLIGHT CENTER, ALABAMA 25812

Electronics Research Department
Martin Marietta Corporation
P. O. Box 179, Denver, Colorado 80201

FOREWORD

This supplement to the final report is presented in response to Paragraph III.2 of Exhibit A of Contract NAS8-24017.

CONTENTS

ABSTRACT

The algorithm described is a fast algorithm for the simultaneous minimization of multiple Boolean functions to a two-level AND-OR form. The major advantages of the algorithm are that it is fast, does not require excessive storage capacity, handles multiple functions, and utilizes unused input states (don't cares) for simplifying the functions. The program as written handles up to 16 variables with no limit to the number of functions; however, the resultant total number of prime implicants may not exceed 3073.

As an example of the performance of the algorithm, a problem consisting of ten functions with ten variables took about 50 seconds of CPU time on a CDC 6500 computer. These ten functions were nominal type functions with approximately 50 percent of the vertices filled. A worst case ten function 13 variable problem took 35 minutes. This problem is considered worst case because it contained 6392 don't care vertices out of the 8192 total per function. This large number of don't cares increases the search time for prime implicants and also gives a large number of prime implicants, thereby increasing the time required for final selection.

This algorithm thus provides the capability for minimizing a set of functions of a large number of variables which were previously done poorly by manual methods and could not be done by computer because of excessive time and storage requirements.

I. INTRODUCTION

This paper describes a computerized procedure for reducing, or simplifying a set of Boolean equations. This is a process which is required for determining a low cost implementation for any digital logic system to be built. For small systems or parts of systems the reduction of a single function or equation can be accomplished manually using techniques well described in several available textbooks (e.g. Phister[1] and Cauldwell[2]). For a combinational logic circuit with fewer than 7 variables the manual processes work quite well. Beyond 7 variables, approximate minimizations are usually accomplished by partitioning the problem into several smaller problems. Computer programs have been written implementing the standard methods to solve problems with more than 7 variables, however historically the memory requirements and running time has been excessive for more than 12 or 13 variables, forcing the partitioning of larger problems for approximate solutions. Memory requirements approximately double with each additional variable, and execution time increases exponentially.

The problem becomes even more complex when there are several outputs or functions of the input variables. There is no known procedure to generate a true minimum for this multiple function case. Good solutions can be found by extending the single function procedures, however extension processes are not well developed in the literature.

---

[1]Montgomery Phister, Jr., "Logical Design of Digital Computers," John Wiley & Sons, 1958.

[2]Samuel H. Caldwell, "Switching Circuits and Logical Design," John Wiley & Sons, 1958.

No known computer programs exist for simplifying multiple equations.

A hardwired telemetry formating technique developed by Martin Marietta[3] generates a set of non-reduced equations for translating word and frame counter states into 10 bit addresses for use in a remote multiplexing system. There can be up to 16 input variables and up to 8 modes or formats, each requiring 10 equations. The broad applicability of this format generation technique emphasized the importance of having a computer program available to reduce the equation set for low cost implementation. The computer program described herein is the second step towards the realization of such a program. The first step was a program for simplifying a single 16 variable equation. This program is based upon the results of the single equation program. Both programs are general purpose and can be applied to the entire spectrum of digital logic design, of which this telemetry formatter is only one significant example.

Basic knowledge of Boolean algebra is necessary for the understanding of the algorithm as discussed herein. The following paragraphs are not intended to substitute for this basic knowledge, but to serve as an introduction to terminology used later and as a brief review of basic Boolean algebra, and logic reduction principles. A simple Boolean equation of four variables is shown as:

$$F = AB\overline{CD} + \overline{A}CD + \overline{AC}D.$$

[3] R. H. Hardin, "A Multiple Format Telemetry Programmer," National Telemetering Conference, 1967, San Francisco, California.

This equation is written in a sum-of-products form. The first term, $AB\overline{CD}$, contains all four variables and is therefore a minterm. A problem with n variables has 2n possible minterms. These minterms can conceptually be visualized at the vertices of an n dimensional cube. Figure I-1 shows a pictorial representation of a 3 variable cube. Another more convenient representation of this same cube is the Karnaugh map shown in Figure I-2. Figure I-3 shows a four variable Karnaugh map with entries depicting the function given in the above equation, where X is an entry and the number in parenthesis indicates the term of the equation from which the entry came. By use of Boolean algebra or by examining the Karnaugh map, the sample equation can be reduced to

$$F = AB\overline{CD} + \overline{A}D$$

The cost of the original equation using the number of gate inputs as a criterion is a four input gate for the first term, 2 three input gates for the last two terms and a three input gate for the OR function for a total cost of 13. The cost of the reduced equation is determined to be 8 by a similar procedure. This cost criteria is commonly used in the literature.

The algorithm which is presented is an automated process for examining the Carnaugh maps of several equations and selecting a good solution. For computer representation of a Karnaugh map, it is con-

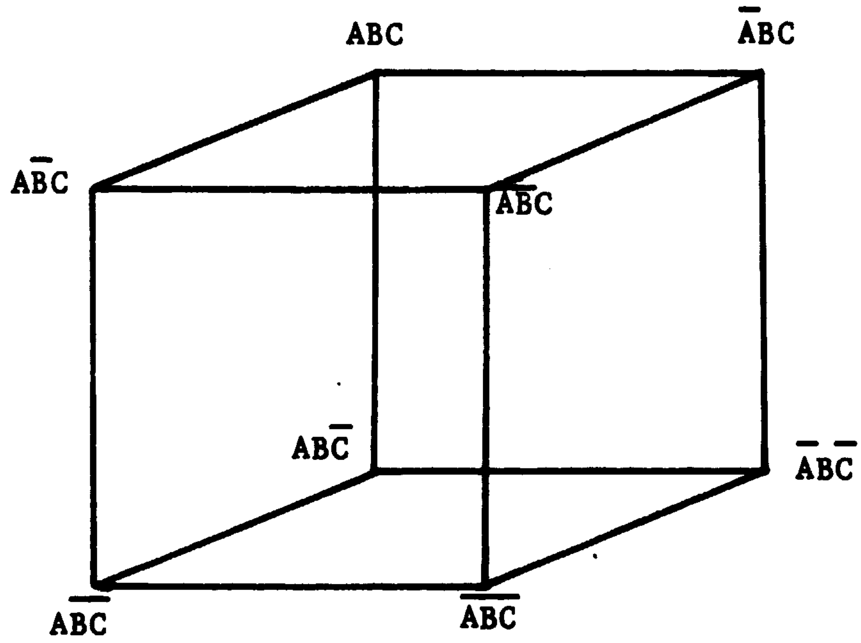


FIGURE I-1: 3-Variable n-Cube

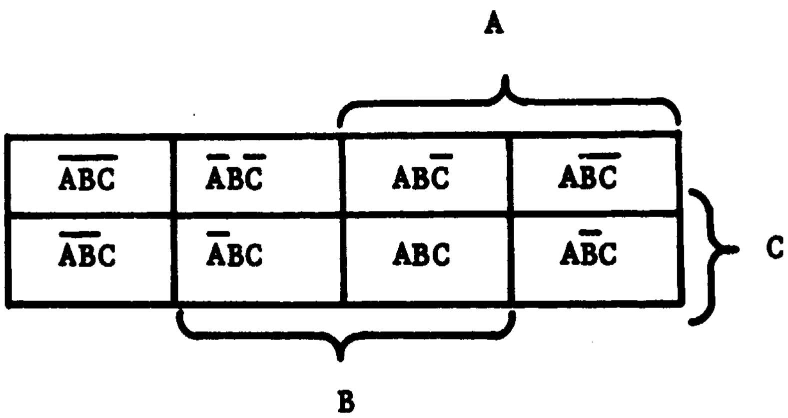


FIGURE I-2: 3-Variable Karnaugh Map

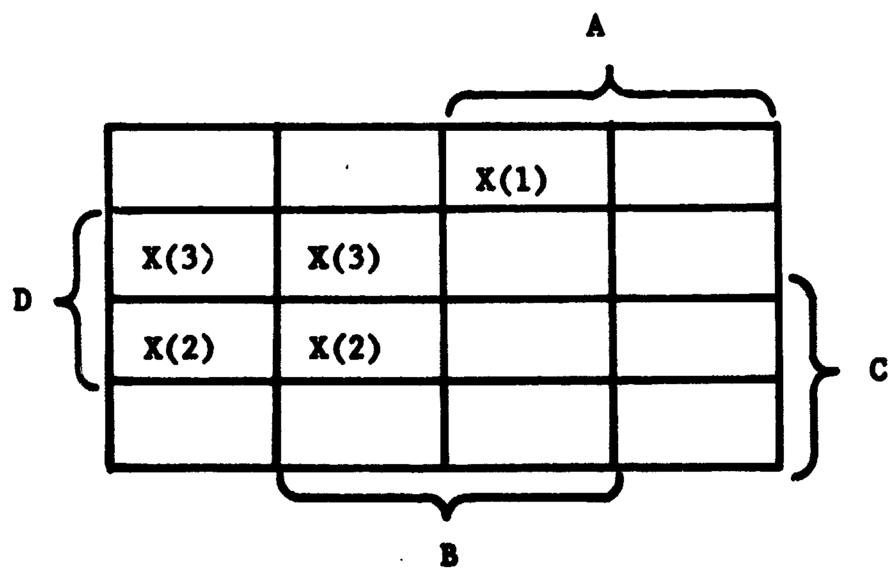


FIGURE I-3: Karnaugh Map for Sample Equation venient to represent the vertices numerically. Figure I-4 shows a four variable map with decimal entries corresponding to the binary value represented by each minterm. For example the minterm $A\bar{B}C\bar{D}$ is in binary form, 1010, which is a decimal 10. The equation of the previous example therefore occupies position 3, 5, 7, 9, and 12 in the map.

Two important concepts in the minimization process are subcubes and prime implicants. A subcube is a set of vertices which correspond to a single term in a sum of products equation. In the special case where all variables are present in the term, the subcube is a vertex. In Figure I-3 the subcubes present were $AB\overline{CD}$, $\bar{A}CD$ and $\overline{AC}D$ from the original equation. Other subcubes for this example are $\bar{A}D$, $\overline{ABC}D$, $\bar{A}\bar{B}CD$, $\overline{AB}C\bar{D}$ and $\bar{A}BCD$. A prime implicant is a subcube which is not wholly contained in another subcube of the function. For the above example the only two prime implicants are $\bar{A}D$ and $AB\overline{CD}$. The minimal solution is a sum of prime implicants, however not all prime implicants are required. For the example of Figure I-5, the prime implicants are $A\bar{C}$, $B\bar{C}D$, $\bar{A}BD$, $\bar{A}BC$ and $\bar{A}\bar{C}D$. The minimal solution is $A\bar{C} + \bar{A}BD + \bar{A}\bar{C}D$. This solution was obtained by inspection. On a larger problem a methodical procedure would be required.

When a simultaneous minimization of several equations is desired, the process changes. The individual equations may not be minimized in order that terms may be shared between equations and thus achieve

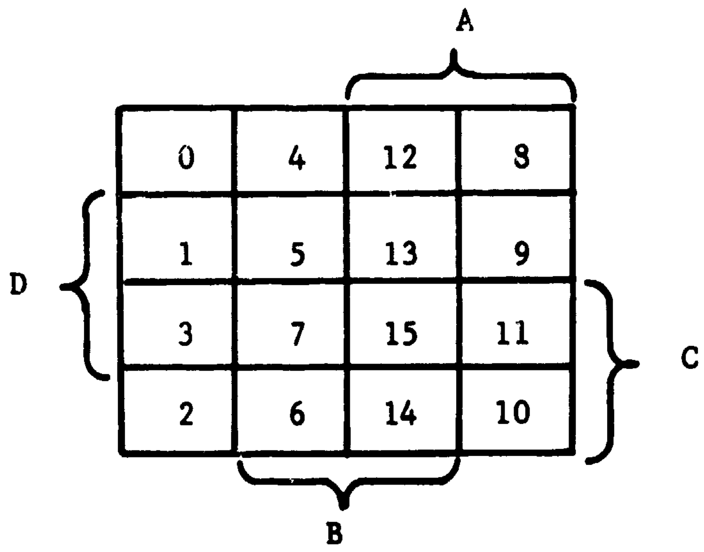


FIGURE I-4: Numerical Representation of Vertices

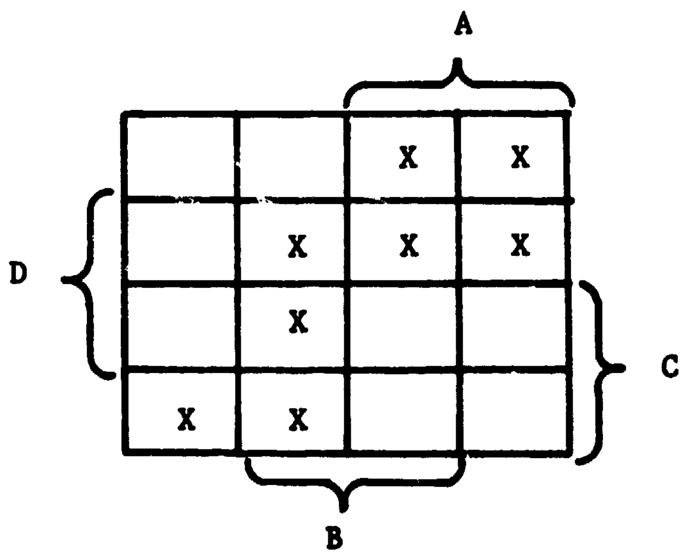


FIGURE I-5: Sample Number 2

a net overall minimum cost. Figure I-6 is an example of a two equation problem. The individual minimal solutions are:

$$\text{Equation 1} = \overline{BD} + A\overline{B}$$

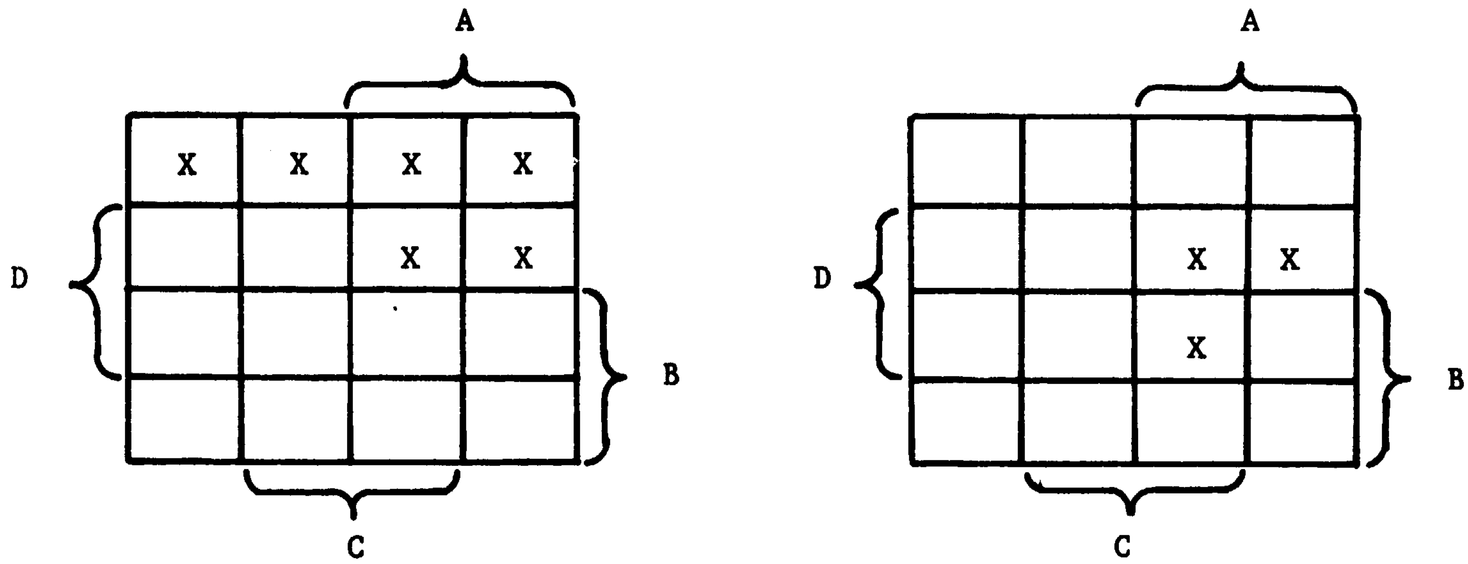
$$\text{Equation 2} = A\overline{B}D + ACD$$

The cost of equation 1 is 6 and the cost of equation 2 is 8, for a total of 14. If equation 1 is rewritten as

$$\text{Equation 1} = \overline{BD} + A\overline{B}D$$

the cost goes up to 7 when considered by itself. When considering the two equation problems, $A\overline{B}D$ is a shared term and only needs to be generated once. The total cost becomes 12. The algorithm presented attempts to maximize the cost savings possible by term sharing.

In the above examples all vertices were specified as a ONE or a ZERO, i. e. either included or excluded from the equation. For many problems there are a set of states (vertices of the n-cube), which the input variables cannot achieve because of outside constraints. These vertices are called <u>don't-care</u> vertices and may be assigned as a ONE or ZERO to simplify the equation. A simple example of such a situation is a decimal counter using four flip-flops which reset after count nine. Figure I-7 is a Karnaugh map for decimal counter with the equation being true for counts 2, 3, 6, 7 and 9. The <u>don't-cares</u> are shown by Ø. Without using the don't cares the reduced equation is $F = \overline{A}C + A\overline{BC}D$. When the don't-cares are used the equation reduces



Equation 1 Equation 2

FIGURE I-6: Two Equation Problem

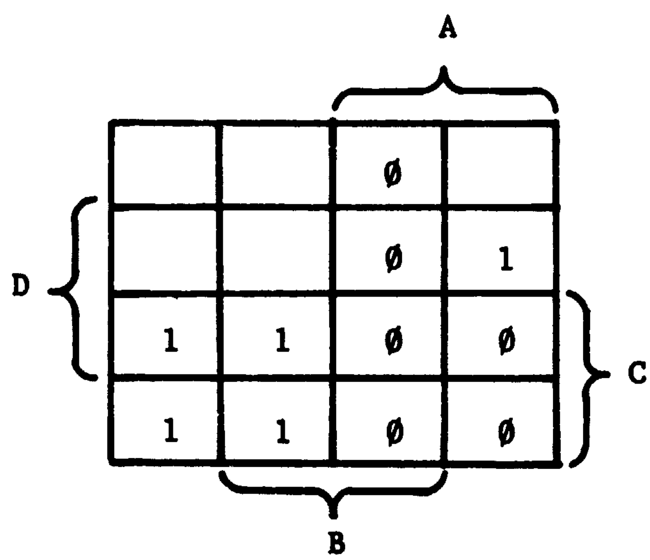


FIGURE I-7: Decimal Counter Example

to F = C + AD. The unallowed counts of 10, 11, 13, 14 and 15 are used as ONES and 12 is ZERO. If several equations were being derived from this counter, each one can use the don't-cares without regard to the manner in which the other equations used them.

II. MULTIPLE EQUATION ALGORITHM

The steps in the total process are to read each equation, including don't cares, into the Karnaugh map and generate the set of prime implicants for each equation. From this set, a merged list is generated that contains the prime implicants from all equations with duplication eliminated. The generation of prime implicants is the most costly part of the process in terms of computer running time. The algorithm for prime implicant generation is presented in Section III. The remainder of this section discusses the method used for prime implicant selection for a low cost solution to the multiple equation problem.

The selection algorithm consists of three phases. The first phase is a preliminary selection of prime implicants followed by Phase 2 which eliminates any redundant subcubes present from Phase 1. Phase 3 examines the equations from Phase 2 looking for the possibility of replacing several small cubes by one larger cube with a resulting lower cost.

Phase 1 - The first step in Phase 1 is to select all essential prime implicants. An essential prime implicant is one which covers one or more required vertices which are not covered by any other prime implicants, When this is accomplished, all remaining vertices are covered by at least two prime implicants. The selection continues by generating a comparison key for each remaining prime implicant and selecting the one that has the largest key. The keys are revised after each selection and again the remaining prime implicant with the largest key is selected. This process continues until the revised keys become all zero. The five comparison keys in decending order of importance are:

1. The total number of ONES covered from all equations.

2. Cube size.

3. The number of ONES covered in this equation.

4. The number of ONES covered in this equation plus the number of covered vertices contained which have been covered previously by subcubes.

5. Cost

KEY-1: This key being in the most significant position forces the selection of the prime implicant which has the largest contribution towards satisfying the complete set of equations being reduced.

KEY-2: Because of don't care vertices it is possible to have a small cube cover the same number of vertices as a large cube. This key forces the selection of the larger cube first, which has a lower cost. This key is modified to maximum size for those cubes which occur in more than one equation. This forces the consideration of terms which can be shared between equations.

KEY-3: For prime implicants which are equal in Keys 1 and 2, this key forces selection of the one which is most important for the equation being reduced.

KEY-4: This key takes the count for Key 3 and adds the count of vertices covered by previous subcubes. This again aids in selecting terms which are most important for the equation being reduced and prevents Key 2 from forcing an all subcube solution due to the sharing of terms from other equations.

KEY-5: This key is the number of gate inputs required to generate the prime implicant less 1. The 1 is subtracted so that the maximum cost is 15 which only takes 4 bits in the key word instead of the 5 bits required for 16. This key selects the smaller prime implicants when all other keys are equal. This can only occur when Key 2 is maximum because the term appears in more than one equation. The smaller cubes must therefore occur in more equations than the larger cube if all other keys are equal.

In the event of a tie in the comparison key, the prime implicant with the largest lower vertex is chosen. If that is also equal, the one with the smallest upper vertex is selected. This cannot be equal or the two prime implicants would be the same. This vertex selection picks the last prime implicants generated in the prime implicant generation subroutine. This implies less probability of covering vertices that can be covered by a large number of other terms. Our test problems have shown this to be a good criterion.

Phase 2 - Phase 2 examines each equation for the presence of redundant cubes and eliminates them. A redundant cube is one whose vertices are completely covered by other cubes in the equation. These redundant cubes can occur because of the high priority of shared prime implicants.

Phase 3 - This phase examines all terms of the merged prime implicant list which have a non-zero comparison key for a given equation to determine what terms of the equation could be eliminated if this prime implicant were used. When terms of the equation can be eliminated, the implementation cost of using the new term is compared with the cost of the replaced terms. If a cost savings results the new term and the cost savings are saved in a list. This continues for all prime implicants and all equations. When the cost saving analysis is complete the prime implicant with the largest cost savings is selected and replaces the appropriate terms in the equations. The cost analysis is re-entered and the high cost savings term is selected again. When no further cost savings are possible by this procedure the algorithm terminates and the final solutions for all equations are printed.

III. PRIME IMPLICANT GENERATION

This section describes the algorithm used for prime implicant generation. This is described separately since it is a very important part of the overall minimization process, requiring a significant part of the computer time for a given problem. The algorithm is based upon the results of C. C. Carroll[4]. Mr. Carroll developed two mathematical theorems which form the foundation of the prime implicant generation algorithm and are described below.

It is clear that for any subcube there is one vertex which has the largest binary value and one that has the smallest binary value. The operation "$\wedge$" between two vertices is defined as a bit by bit AND of the binary numbers (e.g. 1010 $\wedge$ 0110 = 0010). If two vertices v_1 and v_2 of an n-cube are such that $v_1 \wedge v_2 = v_1$, then this relationship is defined as $v_1 \Leftarrow v_2$ (e.g. 0101 $\wedge$ 1101 = 0101; therefore 0101 $\Leftarrow$ 1101). This can be thought of to mean v_1 is contained in v_2.

- Theorem 1: If $c \in C^n$, then min (c) $\Leftarrow$ max (c).

This theorem states that for any subcube, the minimum vertex (min (c)) is contained in the maximum vertex (max (c)).

- Theorem 2: $v \subseteq C$ if $v \Leftarrow$ max (c) and min (c) $\Leftarrow v$

This theorem states that a vertex v of the n-cube C is an element of the subcube c if and only if v is contained in the maximum vertex max (c) and the minimum vertex min (c) is contained in v.

---

[4] C. C. Carroll, "A Fast Algorithm for Boolean Function Minimization," AD680305, <u>Project Themis</u>, Auburn University for Army Missile Command, Huntsville, Alabama, December 1968.

Theorem 2 proves that the minimum and maximum vertex of a subcube are sufficient to completely specify a subcube, and Theorem 1 provides a simple test to determine if two vertices determine a subcube. It is also apparent from theorem 1 that the maximum vertices for all subcubes with a common minimum vertex can be generated directly. This can be done by taking the 0's of min (c) and letting them take on all possible combinations of 1's and 0's, keeping the 1's of min (c) fixed. Similarly all vertices of a subcube can be generated by using theorem 2. Take all 0's of min (c) which correspond to 1's of max (c) and let them take on all combinations of 1's and 0's, keeping fixed the 1's and 0's of max (c) and min (c) which correspond.

An example of subcube generation with a common min (c):

Let min (c) = 01010. the subcubes

are: 01010, 01010 (the vertex min (c))

01010, 01011

01010, 01110

01010, 01111

01010, 11010

01010, 11011

01010, 11110

01010, 11111

An example of subcube vertex generation:

Take the subcube 01010, 11011. The vertices of this subcube are:

01010

01011

11010

11011

The computer implementation of the two generation processes are straightforward iterative procedures. For the subcube generator one starts with the first max (c), which is equal to 2^n-1 for the largest subcube. The remaining max (c)'s are obtained by subtracting binary numbers called RESULT, from 2^n-1. RESULT takes on all binary values that have ZEROs in the positions corresponding to ONEs of min (c). The RESULT values are generated in ascending order which generates subcubes in descending order.

The generation of the vertices of the n-cube starts with min (c) as the first vertex. The complement of max (c) is bit by bit ORed with this vertex with a binary one being added to the result. Following the addition, a bit by bit OR with min (c) is performed followed by a bit by bit AND with max (c). This process continues until max (c) is reached.

For a given equation the first non zero vertex is used as the min (c) and all subcubes with that min (c) are generated, with all non zero max (c)s being flagged in the Karnaugh map. For each flagged max (c) the following actions are taken.

a. If MCARR(K) is a DONT CARE state (=2), the index K is saved in a list (E list) and the E bit counter is incremented.

b. If MCARR(K) is ZERO, the J flag for this max (c) is cleared and the next max (c) is calculated to form a new (I,J) cube and all lists generated for the old cube are abandoned.

c. If MCARR(K) is a CARE state (=1), the index K is saved in a list (L list) and the L count is incremented.

d. If MCARR(K) is a COVERED CARE state (=3), the index K is saved in the E list and E list count is incremented and the size of the cube covering the vertex is examined. If the old cube size is greater than the size of the cube under examination, nothing further is done. If the cube under examination is larger than the old cube, the NONCNT counter is incremented.

When a cube has passed all the MCARR(K) examinations, the cube is a prime implicant. For each element in the L list (i.e. CARE K's) the following operations are performed:

a. The size of the current cube is placed in MCARR(L).

b. The prime implicant number is placed in MCARR(L).

c. The CARE state (=1) is changed to a COVERED CARE state (=3).

d. The J flag is cleared for MCARR(L).

When all max (c)s for a given min (c) are exhausted, the next non zero min (c) is obtained and the process is continued. When min (c) exceeds the largest 1 bit set in the Karnaugh map, the process terminates.

It is important to note that a complete search of the Karnaugh map is made before the subcube generation process terminates. Our test problems indicated that this results in improved solutions in some examples over a termination process which stops generating subcubes when all required vertices have been covered. For large equations with more than 12 variables and with large numbers of don't-care vertices, a large amount of computational time can be spent searching for subcubes after all vertices have been covered. The limited experience with this type of problems indicate very little degradation of solution if the earlier termination is used. Therefore a control card is used to allow a user to select between early cutoff and no cutoff, thereby making his own cost effectiveness decision. The same control card is used to select input mode. The two input options are equation or vertex number designation.

IV. DATA PREPARATION

The program prepares for the algorithm by first reading two control cards. The first consists of two fields as shown in Figure IV-1. The type field controls the type of equation input, and the cutoff switch field controls the cutoff switch. The master control card consists of 17 fields as shown in Figure IV-2. The size field controls the size of Karnaugh Map (MCARR, Figure IV-3) used by the program. The remaining fields are used to control the interpretation of a term of an equation. The bit numbers which are active are inserted into the first fields with all other fields being zero. Thus for a four variable problem the first four fields are filled with 1, 2 3, and 4.

The next card(s) are the cards containing the information about the DONT CARE (excluded) states. This card consists of six fields as shown in Figure IV-4. The first field is an end-of-data type indicator and is used only following all cards which contain data (of which there may be none). The second field contains the first DONT CARE state expressed in a decimal number. The third field contains the last DONT CARE state expressed in a decimal number. The fourth field contains the multiplier. The fifth field contains the first number to be multiplied. The sixth field contains the last number to be multiplied.

The data preparation phase of the program first initializes MCARR to zero using the size input to the program to determine where to stop. The program then uses the CONT CARE control cards to set the DONT CARE

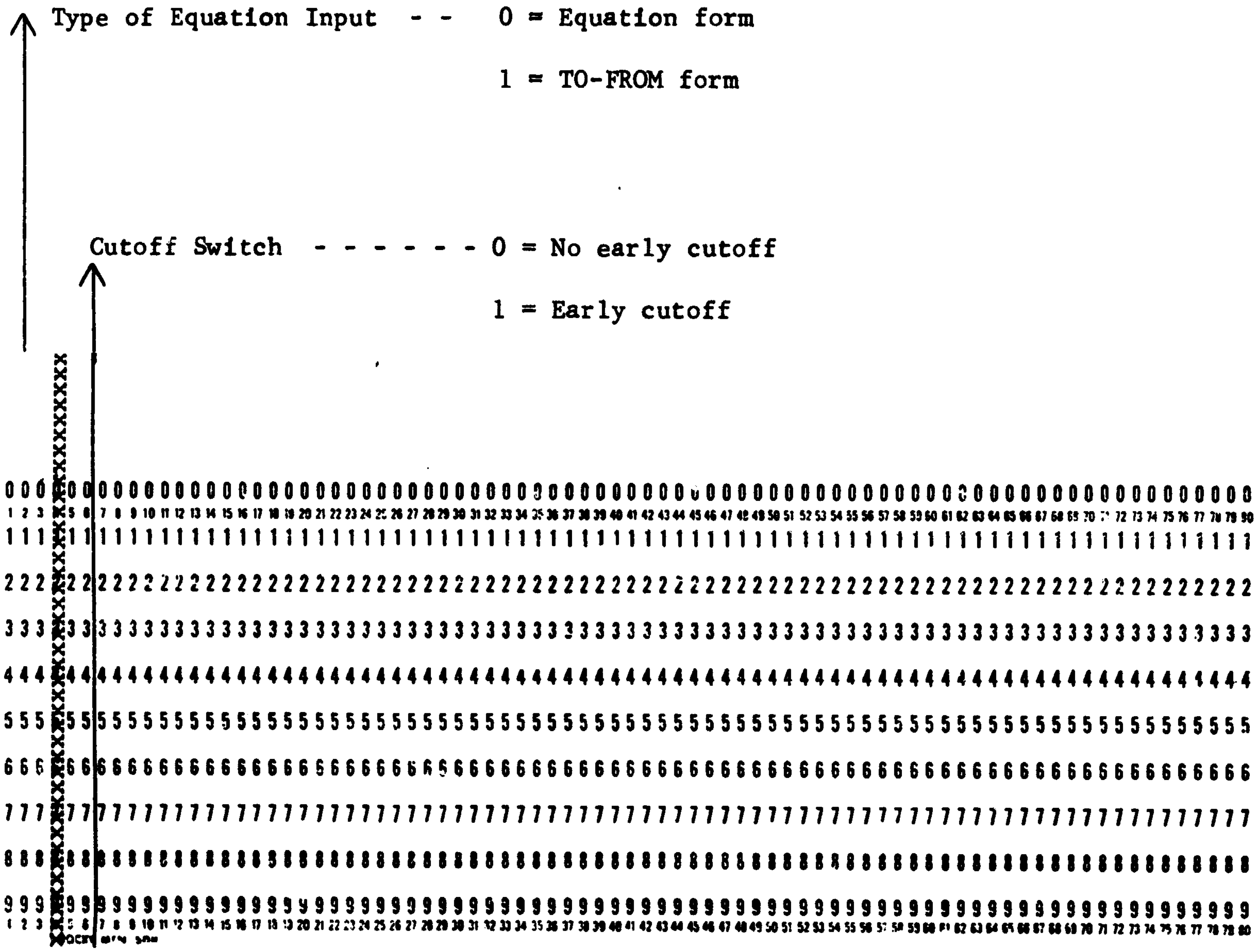


FIGURE IV-1, Type and Cutoff Control Card

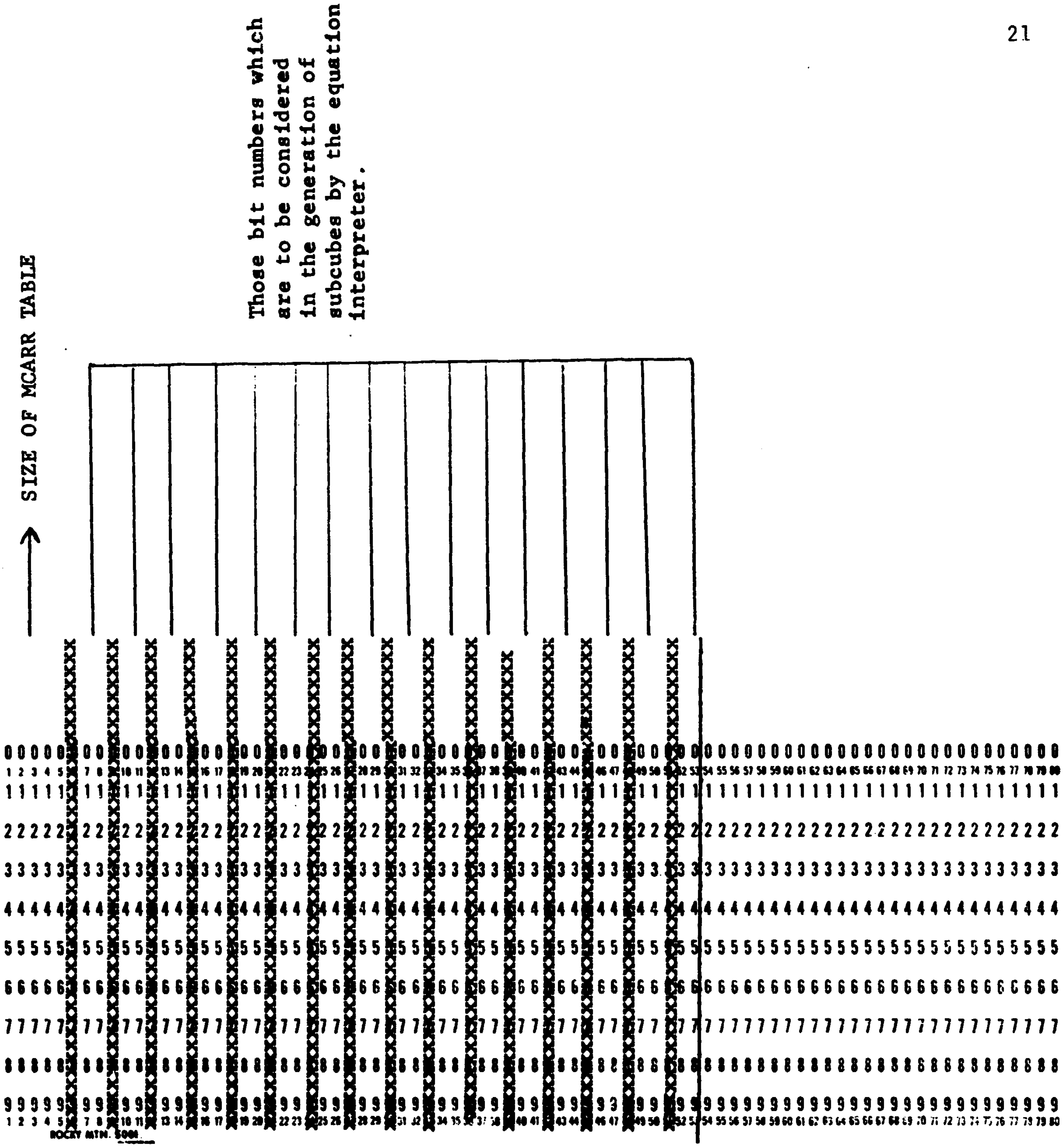


FIGURE IV-2, Master Control Card

MCARR TABLE

ENTRY 0	ENTRY 1	60 BIT WORDS
ENTRY 2	ENTRY 3	
ENTRY 4	ENTRY 5	
ENTRY 65532	ENTRY 65533	
ENTRY 65534	ENTRY 65535	

ENTRY DESCRIPTION

JF	CS	MPINUM	D	30 BIT ENTRY

JF - J FLAG - 1 octal digit - When set indicates that this is a good high vertex for a cube.

CS - CUBE SIZE - 2 octal digits - Set to the cube size which covers this vertex.

MPINUM - PRIME IMPLICANT NUMBER - 6 octal digits - This indicates the number of the cube which covers this vertex. If ZERO, the vertex has been covered by more than one cube. Used for essential prime implicant selection.

D - DESCRIPTOR - 1 octal digit - If set to ZERO indicates bit is ZERO.
If set to ONE indicates bit is ONE.
If set to TWO indicates bit is DONT CARE.
If set to THREE indicates bit was ONE. and has now been covered; CS and MPINUM are used only in this state.

FIGURE IV-3, MCARR Table

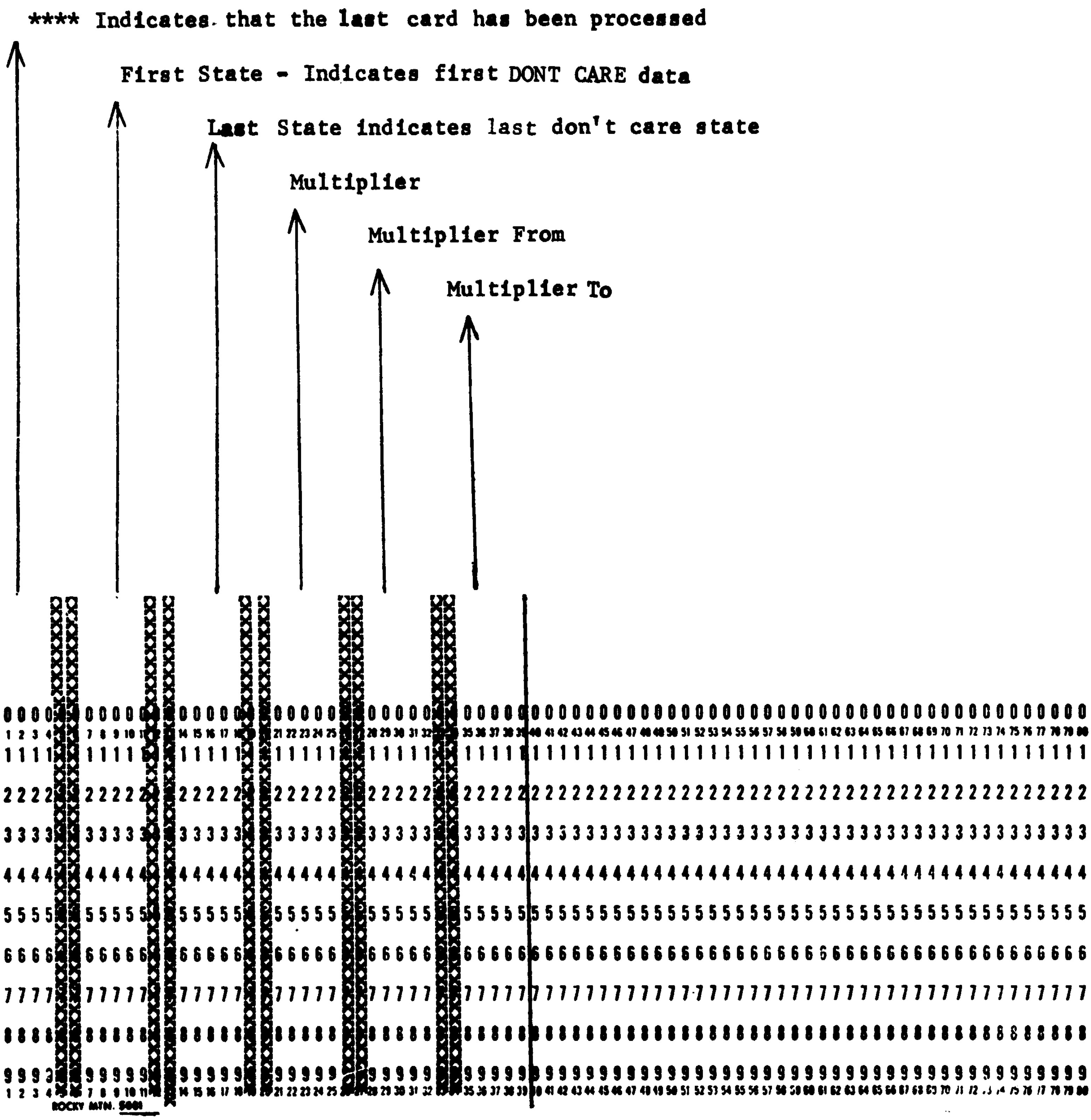


FIGURE IV-4, DONT CARE Control Card

state in MCARR. The program uses the following calculation to determine the bits to set for each DONT CARE control card:

(first state + N) + (multiplier) (Multiplier from + M)

where N = 0, 1, 2, 3, ... and M = 0, 1, 2, 3, ... and when (first state +N) = last state, then M is incremented and (first state + N) is set to (first state + 0). After (multiplier from)= (multiplier to) the next card is processed.

At this point in the data preparation phase, MCARR contains no care states. The program then determines the type of input and if equation form is indicated, the program then reads an equation term in the form :

S1 = Q1 Q2 Q3' Q4

Q1 Q2' Q3

The equation term may be placed in any card column but may not extend to the next card. There may not be more than one equation term per card.

The program reads the card and, using the bit numbers input in the master control card, interprets the term in the following manner. If all the bits called out in the master control card are contained in the equation term, then the bit pattern is used as a binary number pointing to that single care state. If all the bits called out in the master control card are not used in the term, then the unused bits are considered as X state bits and are taken through all possible states and all the resulting states are set into MCARR.

If the type indicated is TO-FROM form, the next card (Figure IV-5) is read and the appropriate vertices in MCARR are set. The program then generates all the necessary prime implicants. The program then determines by looking at the next card to be read if another equation is to be reduced.

The program determines the last care (ONE) bit set before it enters the prime implicant generation routine.

If another equation is to be reduced MCARR is initialized again and the same DONT CARE control cards are used to generate the DONT CARE states. If the card contains **** in the first four columns, the program terminates.

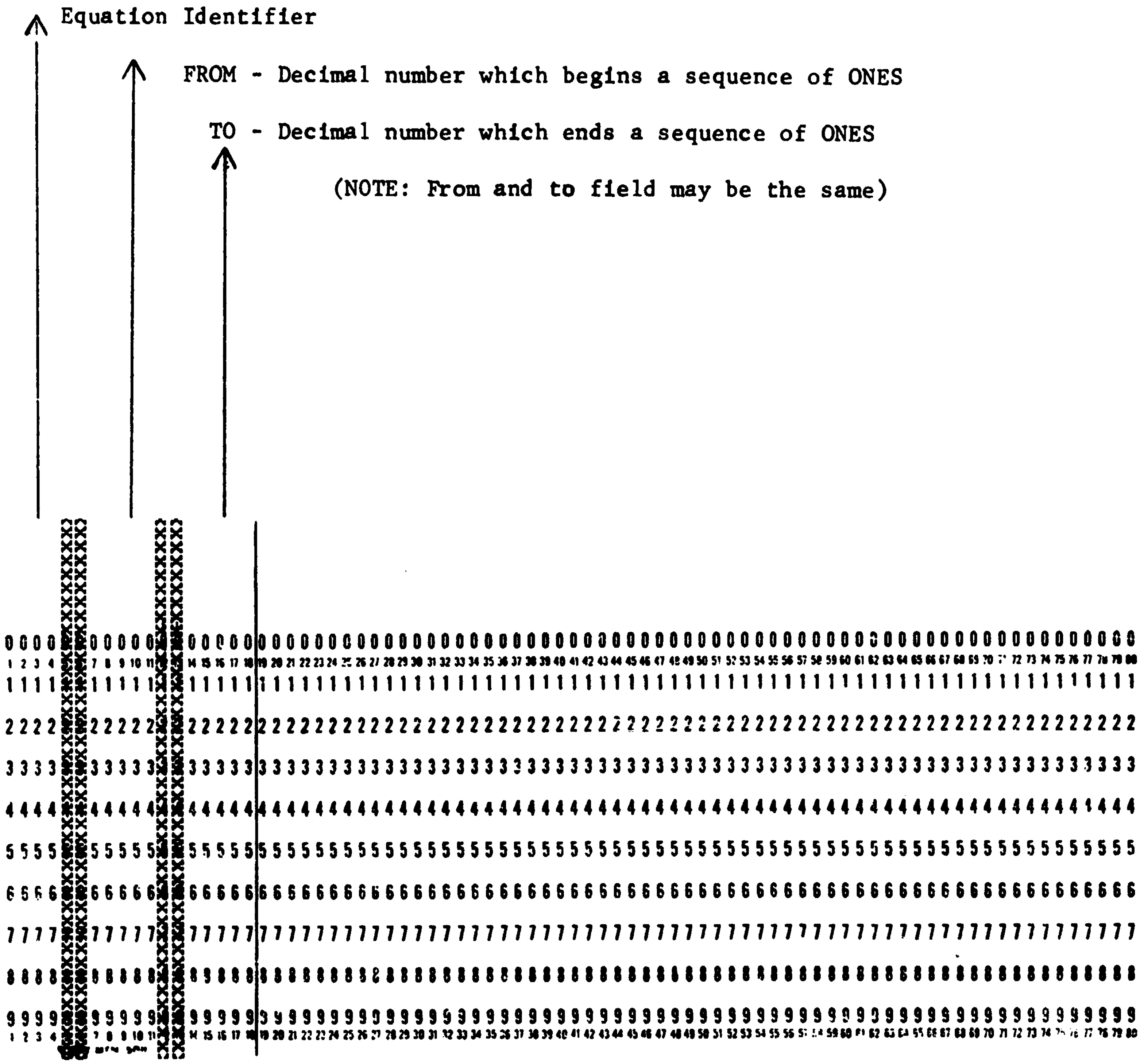


FIGURE IV-5, TO - FROM Equation Card

V. CONCLUSIONS

The computer program that has been developed shows considerable promise in the minimization of Boolean functions. In particular, it provides the capability to minimize multiple Boolean functions with up to 16 variables. In addition, it has the capability to make use of the forbidden states when the function is for non-binary systems. Tests showed that the algorithm was indeed very fast, that it did not require excessive storage capability, and that it found the minimum two level AND-OR representation in all of the test problems. It may be that the algorithm will always find the minimum but the proof of this would require considerable effort. Since the algorithm will always provide a solution that is close to the minimum, this additional effort would not be warranted except for purely academic reasons.

The results of this program to date have been the achievement of an algorithm which is both good, in terms of solution quality, and practical, in terms of computer time required, for a classical two level AND-OR minimization of multiple functions of a large number of variables. The results are so good that the necessary steps should be taken to make the algorithm even more useful. These steps are:

1. Modify the input and output routines to allow a large flexibility in problem specification formats.

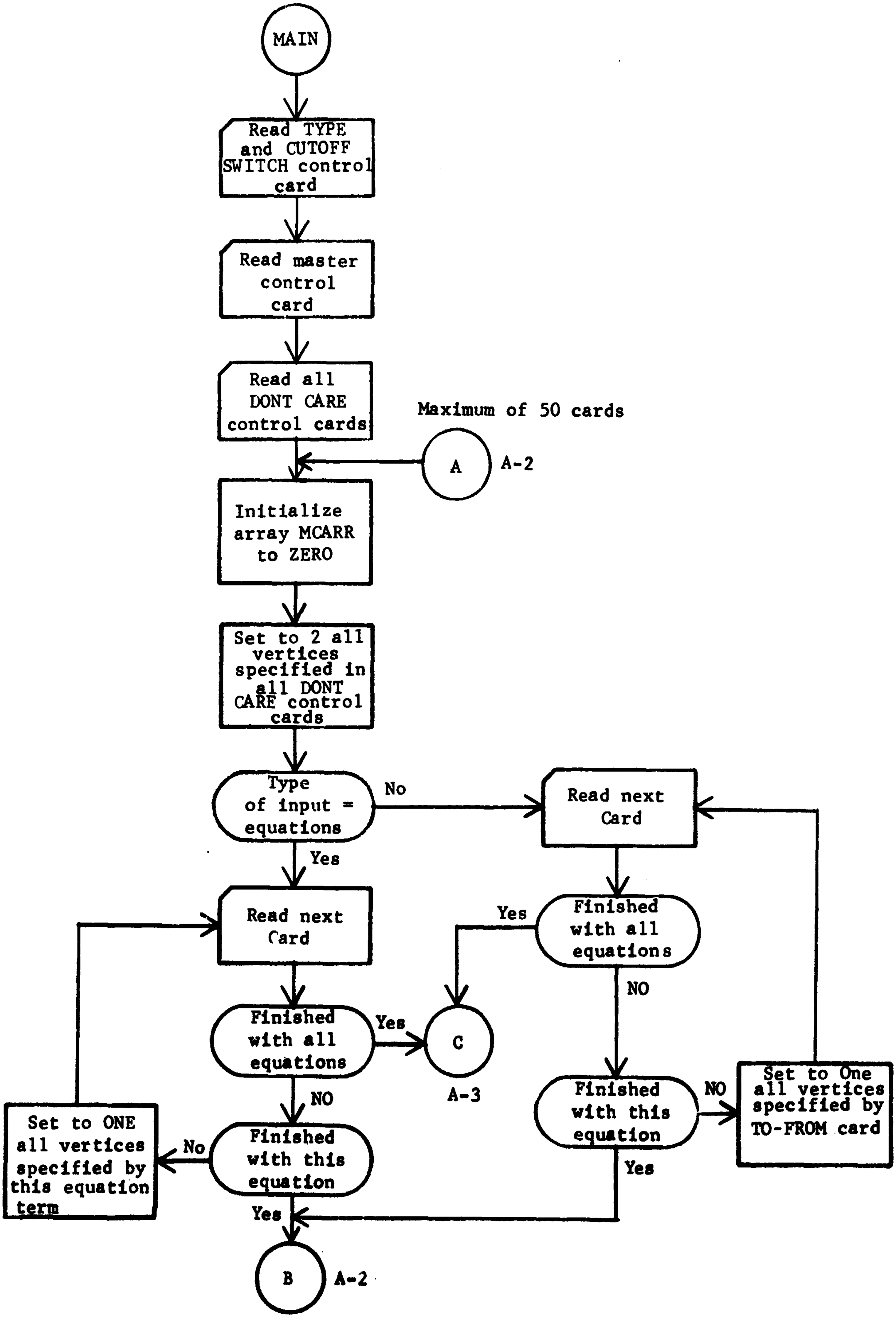
2. Extend the minimization to a multiple level solution which looks for common subterms which can be shared. This is partially accomplished now, in that terms of one equation which are subterms of another equation are used. As an example of subterms of a single equation consider $F = A\overline{B}CD\overline{EF} + A\overline{B}CEFG$, which can be factored as $F = A\overline{B}C(D\overline{EF} + EFG)$ indicating a sharing of the subterms $A\overline{B}C$.
3. Customize the solution to a particular logic family, taking into account fan-in and fan-out capabilities as well as incorporating special functions where applicable.

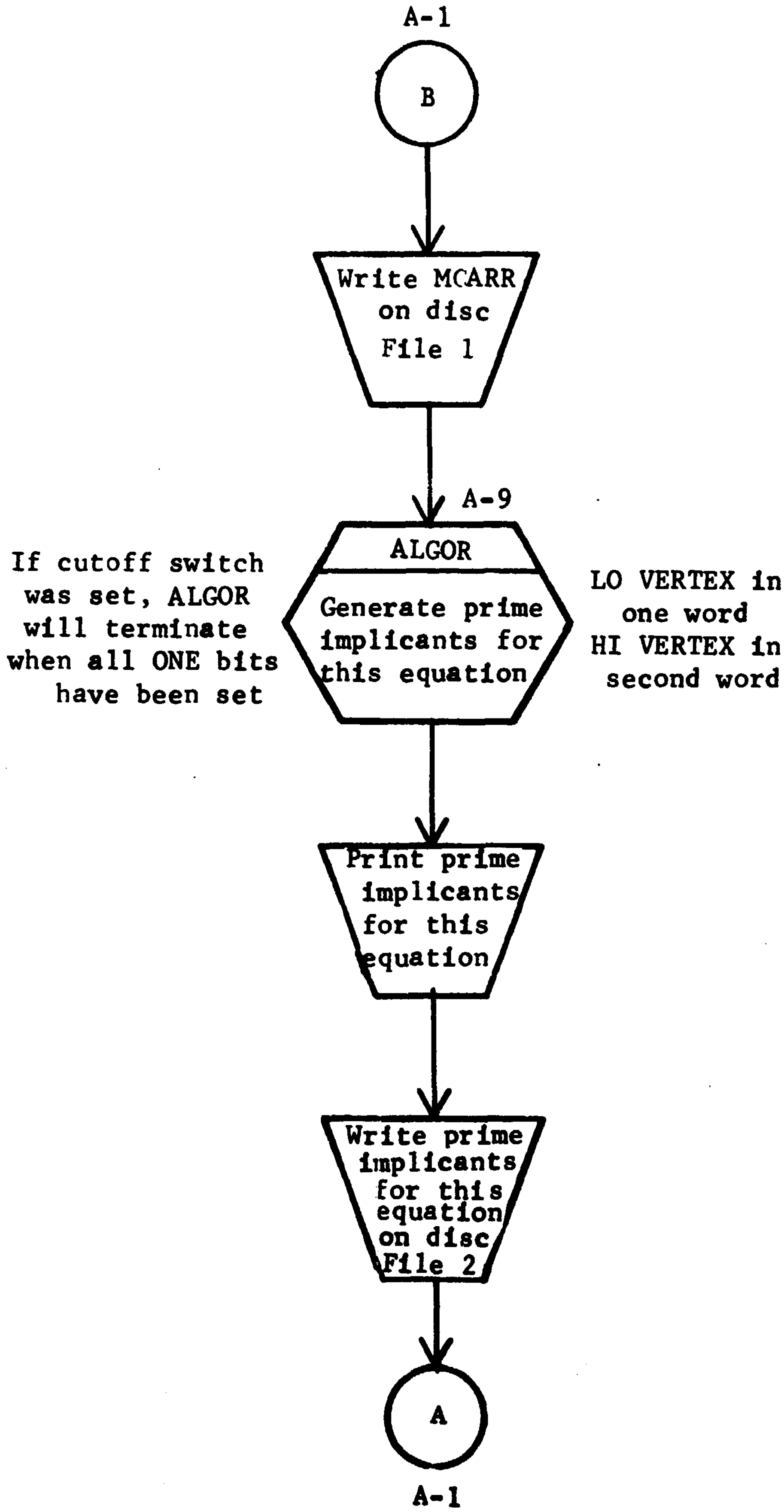
These three steps are not completely separate in that the improved input/output format is desirable for any useful program and the customization for a logic family implies multiple level solutions because of fan-in limitations. The steps 2 and 3 can be taken separately, but could be more efficiently accomplished together.

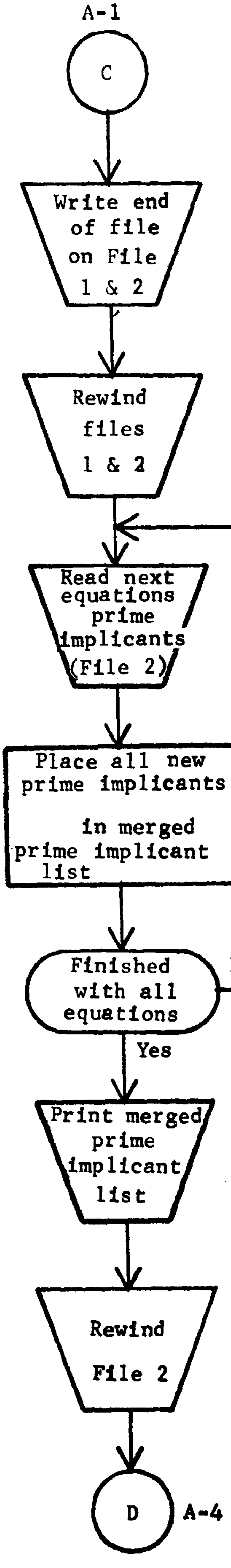
APPENDIX

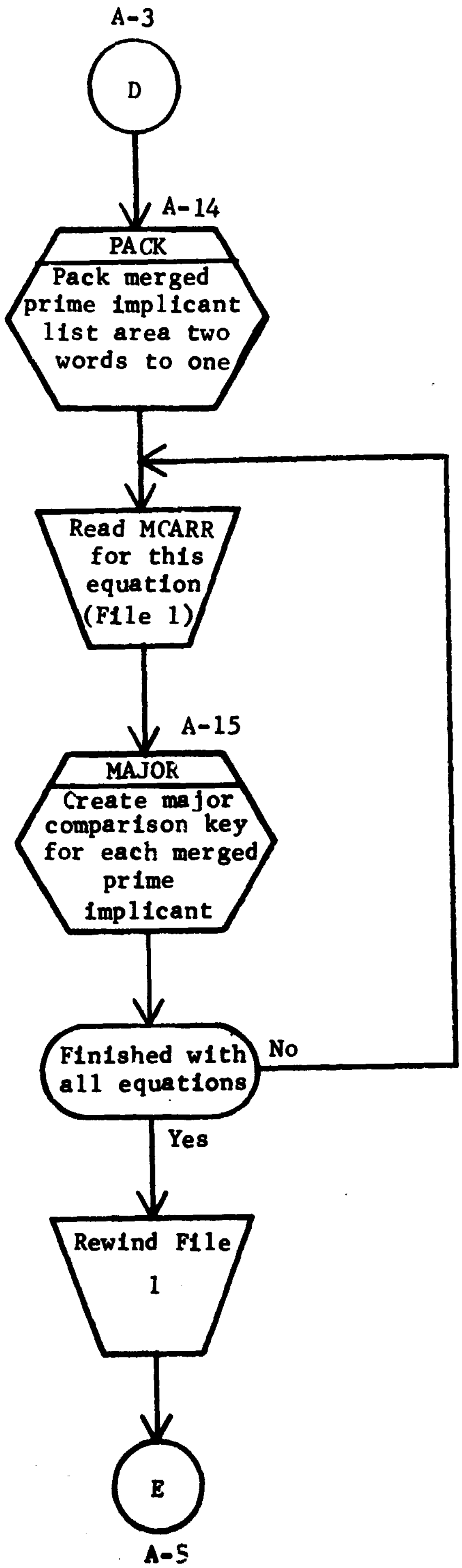
This Appendix contains the flow charts for the computer programs which implement the algorithms discussed in the main body of the report.

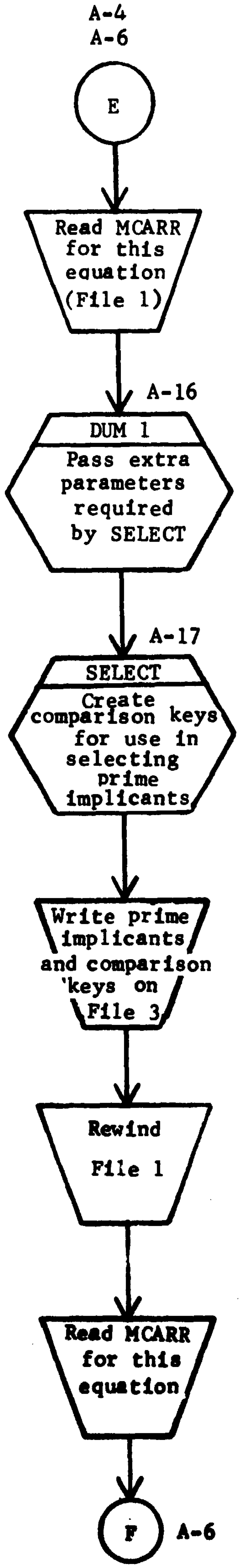
APPENDIX CONTENTS


MAIN
Read TYPE and CUTOFF SWITCH control card
Read master control card
Read all DONT CARE control cards
Maximum of 50 cards
A
A-2
Initialize array MCARR to ZERO
Set to 2 all vertices specified in all DONT CARE control cards
Type of input = equations
No
Yes
Read next Card
Read next Card
Finished with all equations
Yes
NO
Finished with all equations
Yes
C
A-3
NO
Finished with this equation
No
Set to ONE all vertices specified by this equation term
Yes
Finished with this equation
NO
Set to One all vertices specified by TO-FROM card
Yes
B
A-2

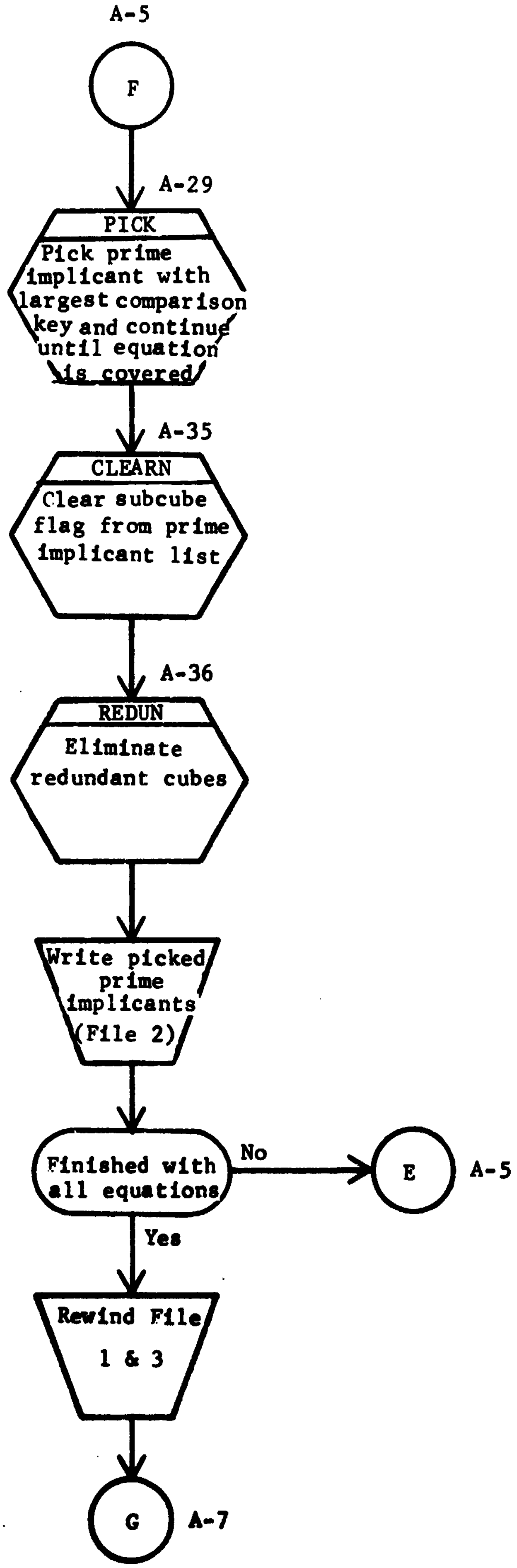

A-1
B
Write MCARR on disc File 1
A-9
ALGOR
Generate prime implicants for this equation
If cutoff switch was set, ALGOR will terminate when all ONE bits have been set
LO VERTEX in one word HI VERTEX in second word
Print prime implicants for this equation
Write prime implicants for this equation on disc File 2
A
A-1

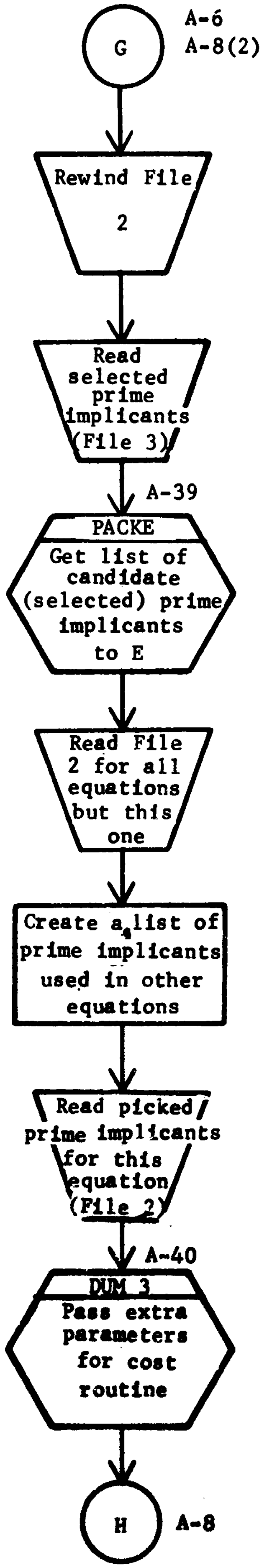

A-1
C
Write end of file on File 1 & 2
Rewind files 1 & 2
Read next equations prime implicants (File 2)
Place all new prime implicants in merged prime implicant list
Finished with all equations
NO
Yes
Print merged prime implicant list
Rewind File 2
D
A-4

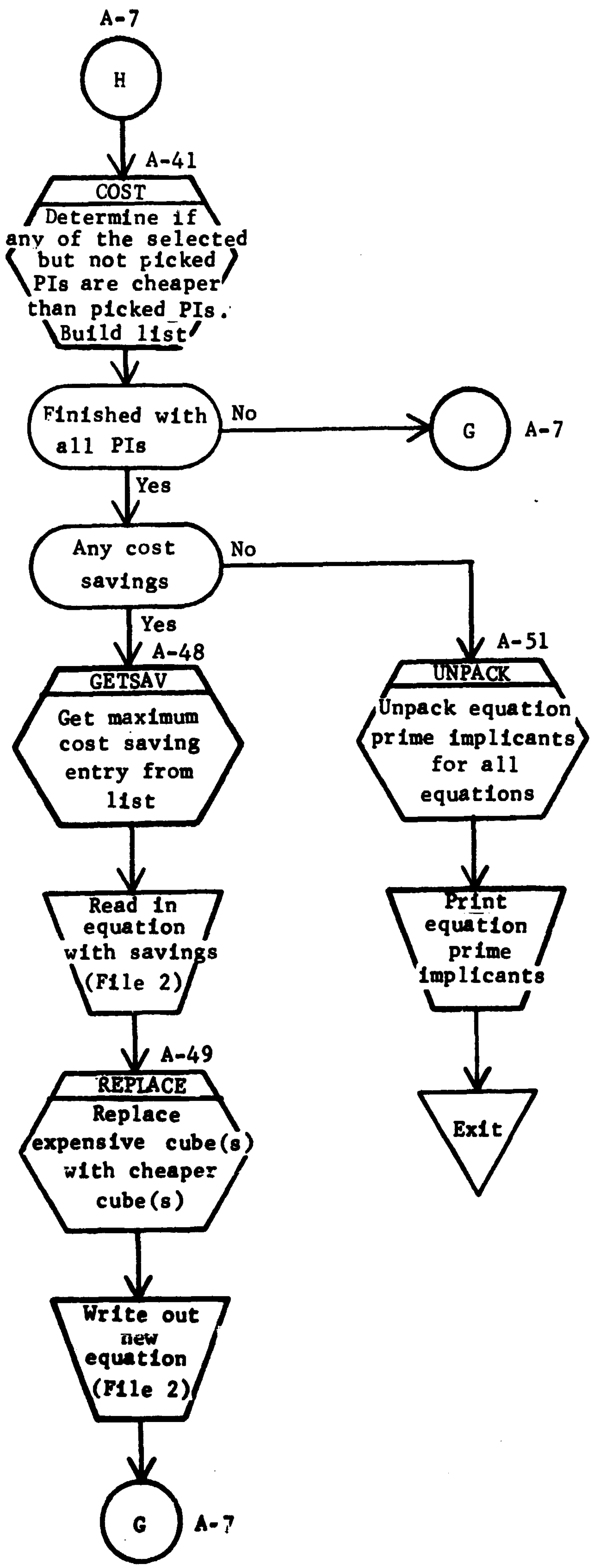

A-3
D
A-14
PACK
Pack merged prime implicant list area two words to one
Read MCARR for this equation (File 1)
A-15
MAJOR
Create major comparison key for each merged prime implicant
Finished with all equations
No
Yes
Rewind File 1
E
A-5

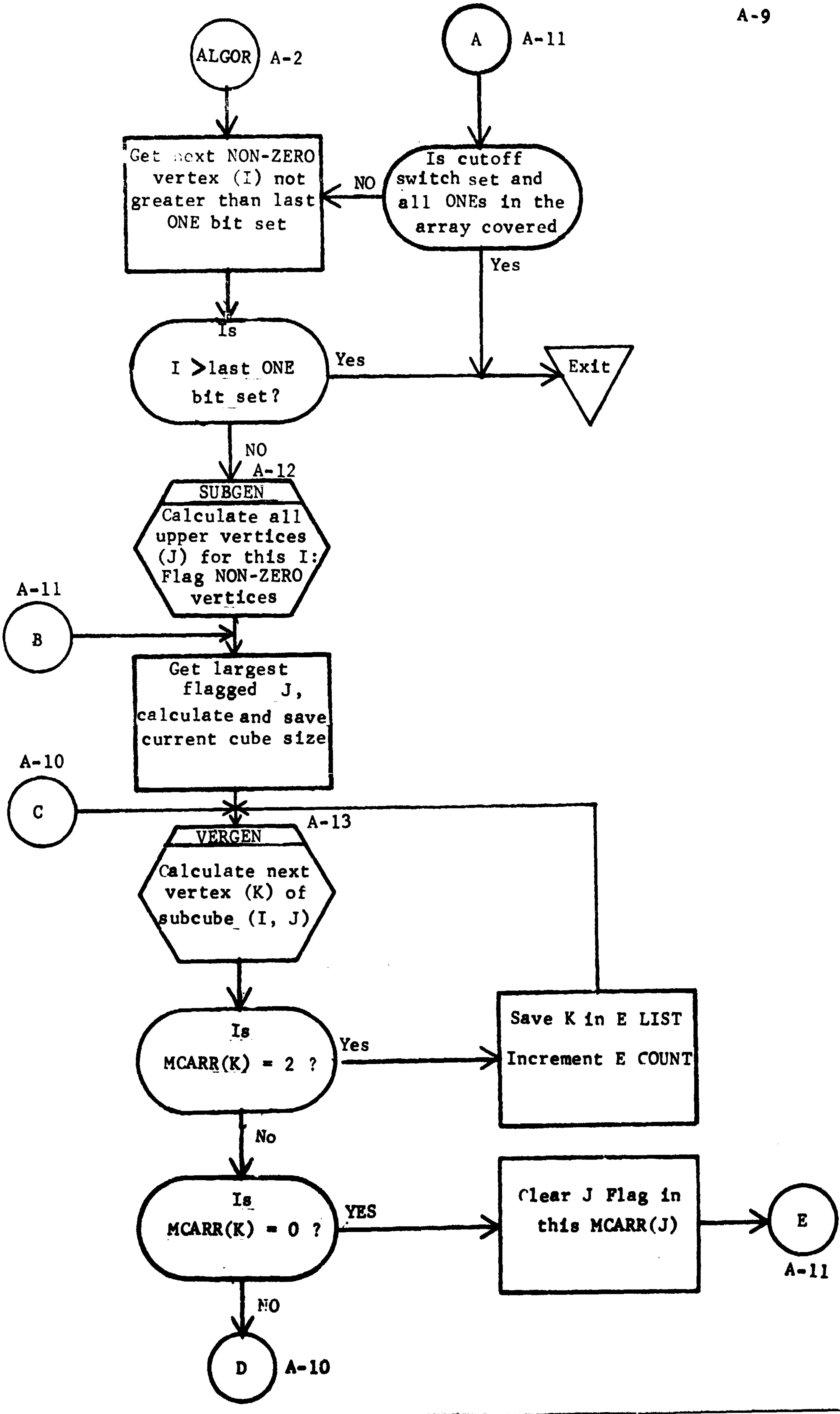


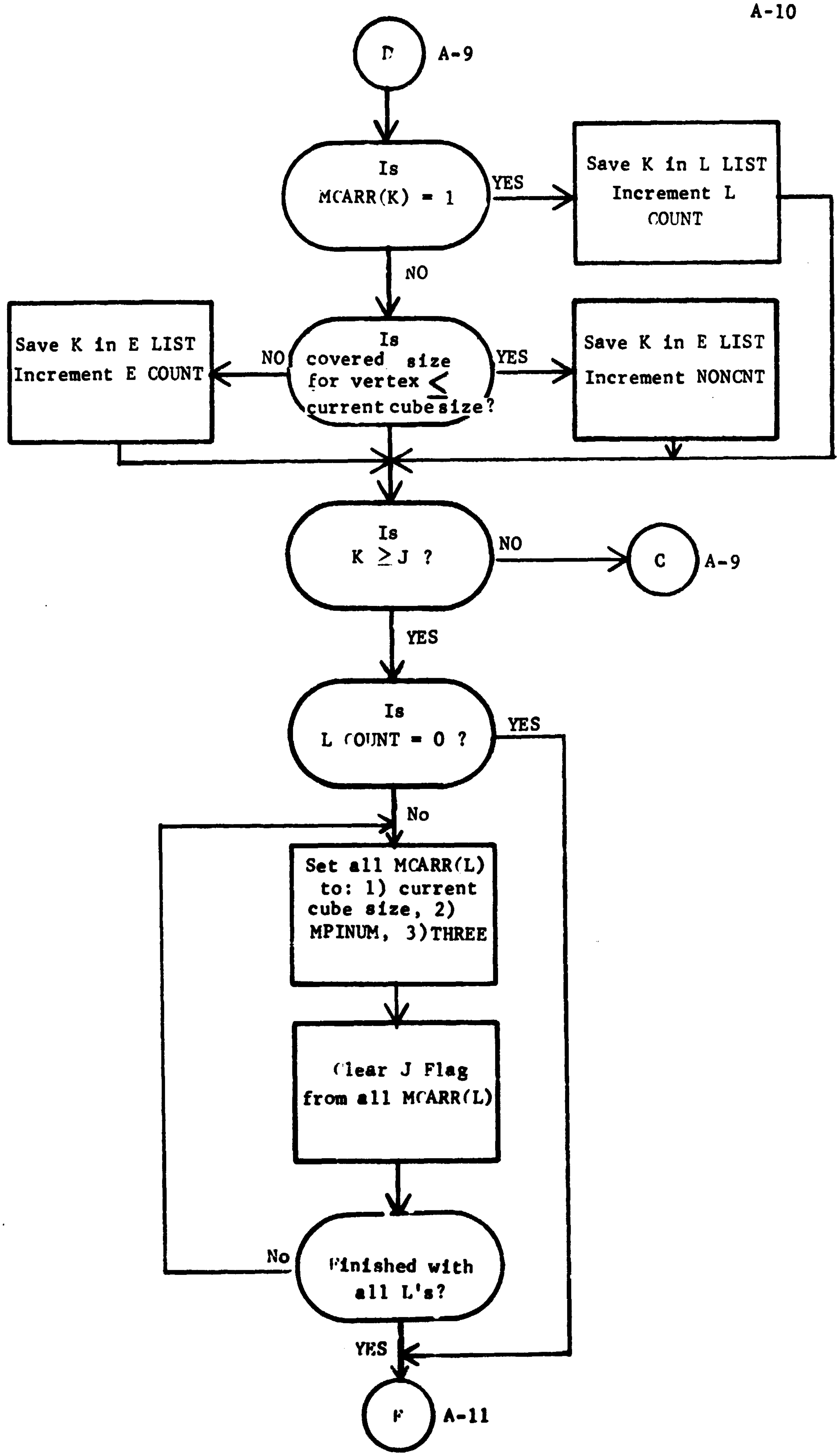
A-4
A-6
E
Read MCARR for this equation (File 1)
A-16
DUM 1
Pass extra parameters required by SELECT
A-17
SELECT
Create comparison keys for use in selecting prime implicants
Write prime implicants and comparison keys on File 3
Rewind File 1
Read MCARR for this equation
F
A-6

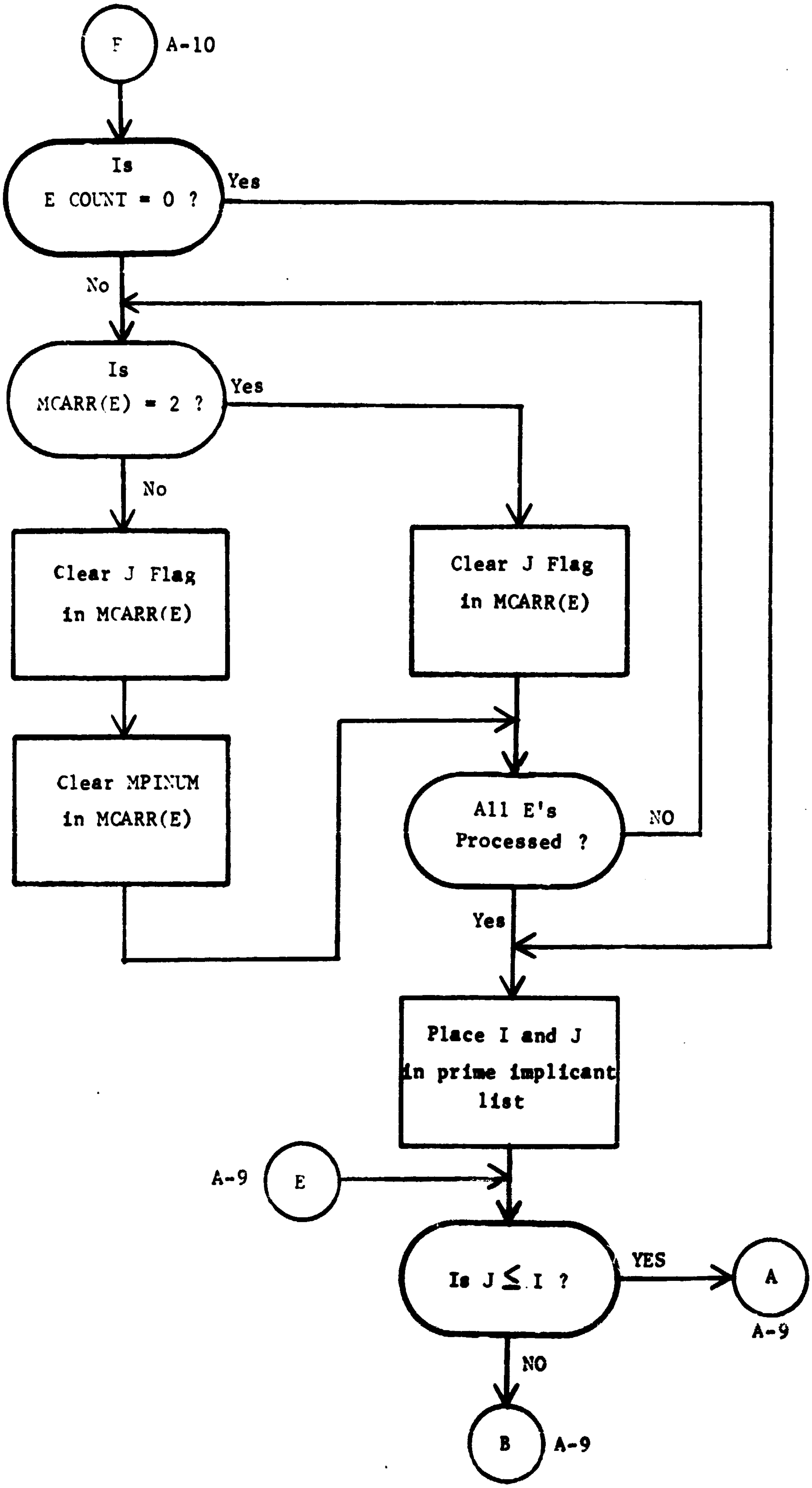

A-5
F
A-29
PICK
Pick prime implicant with largest comparison key and continue until equation is covered
A-35
CLEARN
Clear subcube flag from prime implicant list
A-36
REDUN
Eliminate redundant cubes
Write picked prime implicants (File 2)
Finished with all equations
No
E
A-5
Yes
Rewind File 1 & 3
G
A-7


G
A-6
A-8(2)
Rewind File 2
Read selected prime implicants (File 3)
A-39
PACKE
Get list of candidate (selected) prime implicants to E
Read File 2 for all equations but this one
Create a list of prime implicants used in other equations
Read picked prime implicants for this equation (File 2)
A-40
DUM 3
Pass extra parameters for cost routine
H
A-8

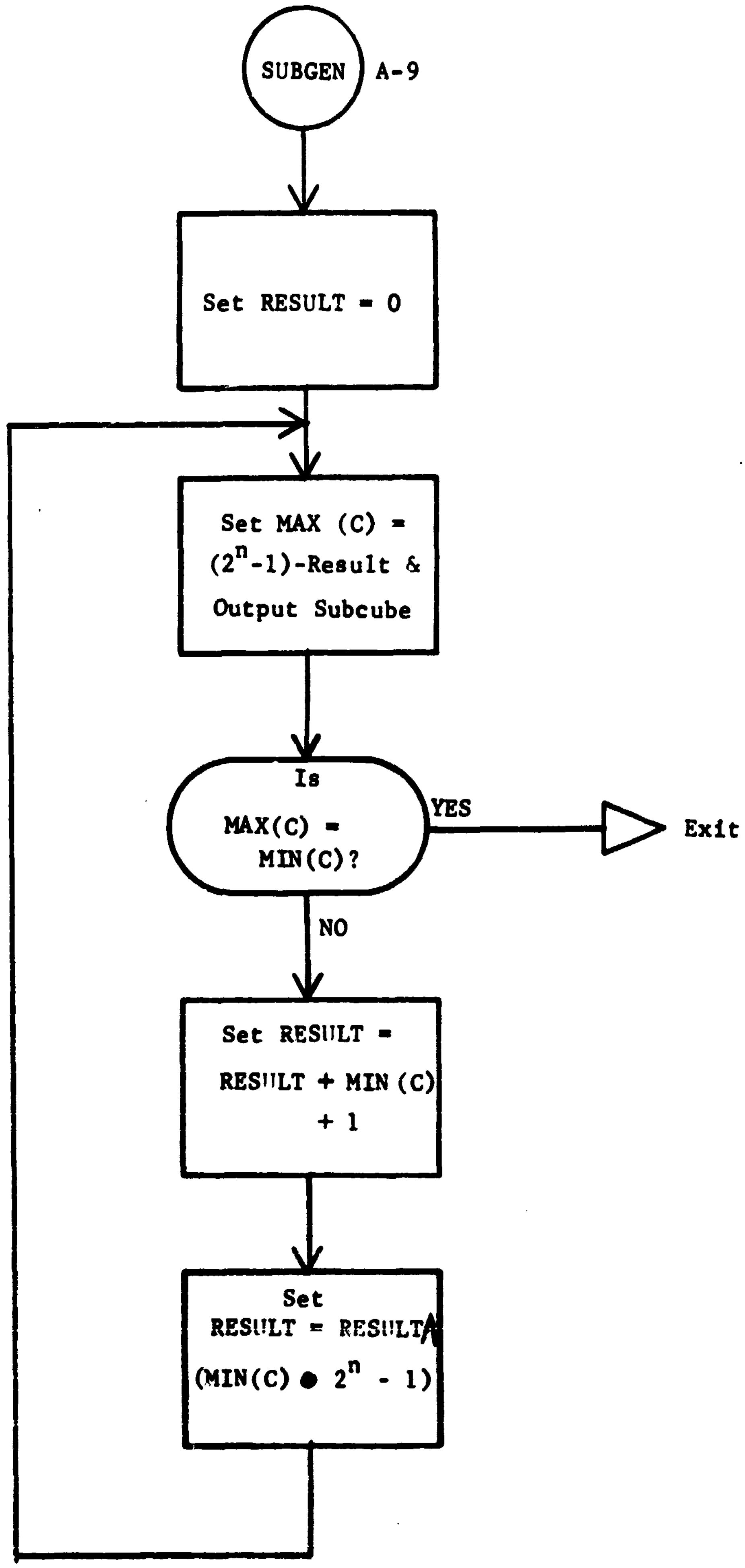

A-7
H
A-41
COST
Determine if any of the selected but not picked PIs are cheaper than picked PIs. Build list
Finished with all PIs
No
G
A-7
Yes
Any cost savings
No
Yes
A-48
GETSAV
Get maximum cost saving entry from list
A-51
UNPACK
Unpack equation prime implicants for all equations
Read in equation with savings (File 2)
Print equation prime implicants
A-49
REPLACE
Replace expensive cube(s) with cheaper cube(s)
Exit
Write out new equation (File 2)
G
A-7

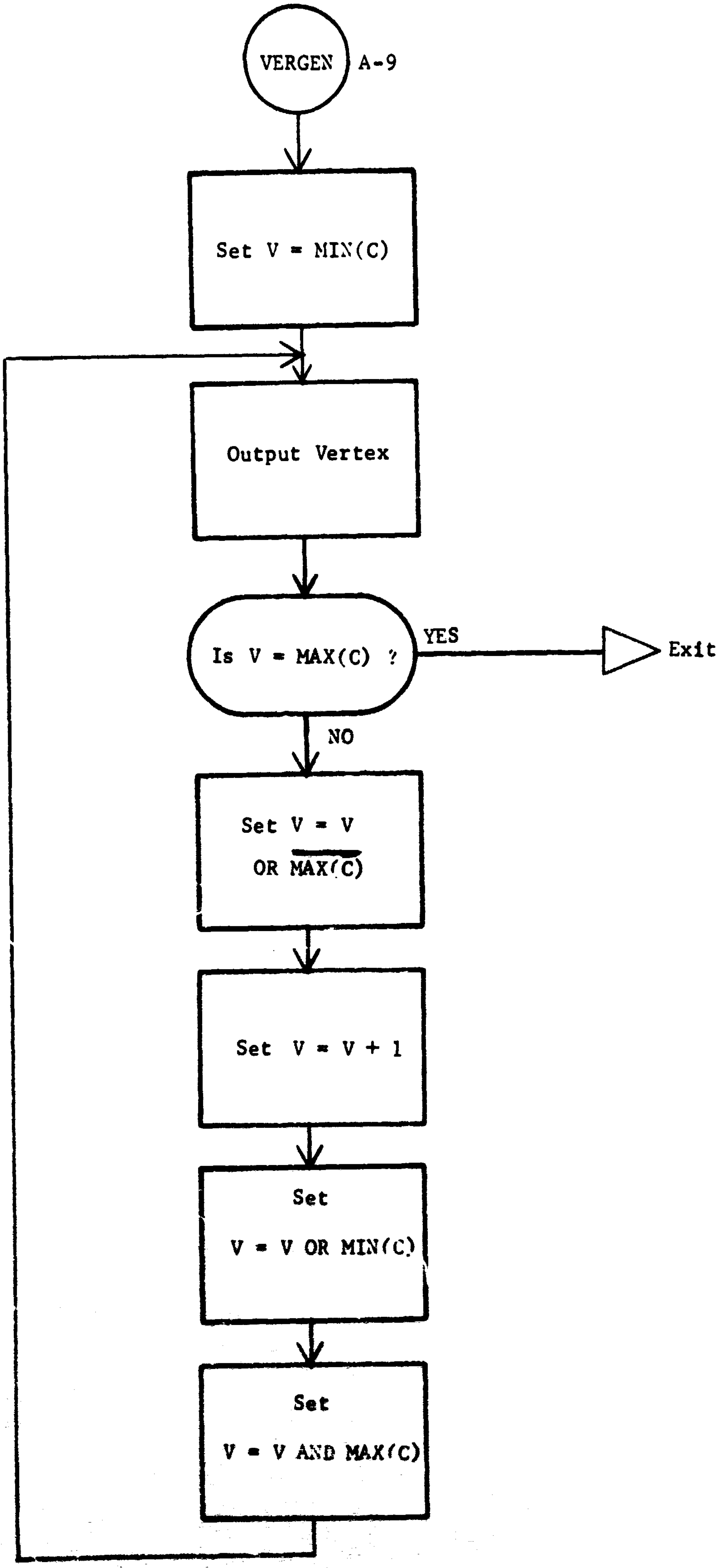

ALGOR
A-2
A
A-11
Get next NON-ZERO vertex (I) not greater than last ONE bit set
NO
Is cutoff switch set and all ONEs in the array covered
Yes
Is I > last ONE bit set?
Yes
Exit
NO
A-12
SUBGEN
Calculate all upper vertices (J) for this I: Flag NON-ZERO vertices
A-11
B
Get largest flagged J, calculate and save current cube size
A-10
C
A-13
VERGEN
Calculate next vertex (K) of subcube (I, J)
Is MCARR(K) = 2 ?
Yes
Save K in E LIST
Increment E COUNT
No
Is MCARR(K) = 0 ?
YES
Clear J Flag in this MCARR(J)
E
A-11
NO
D
A-10

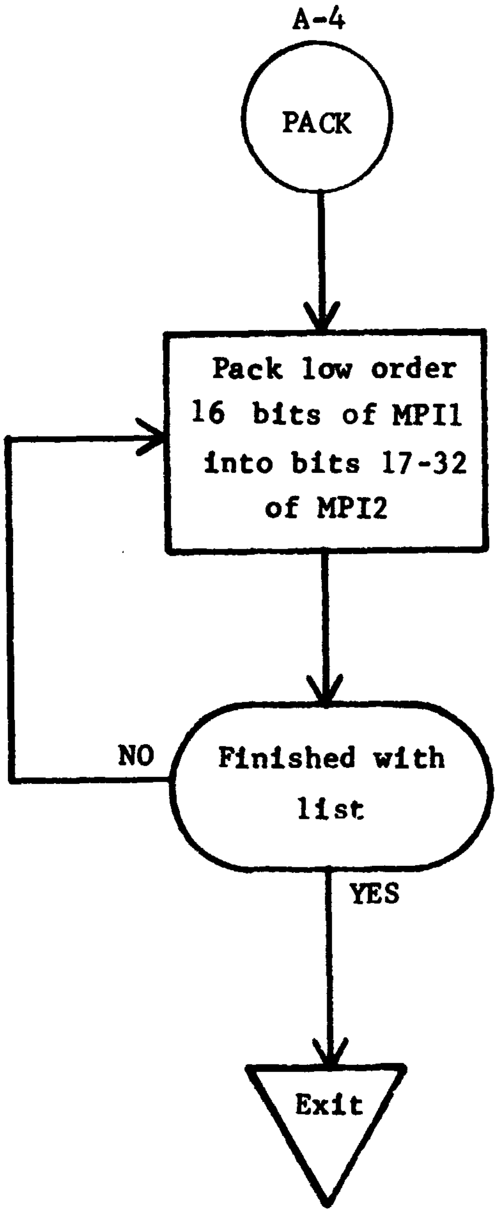

D
A-9
Is
MCARR(K) = 1
YES
Save K in L LIST
Increment L
COUNT
NO
Save K in E LIST
Increment E COUNT
NO
Is
covered size
for vertex ≤
current cube size?
YES
Save K in E LIST
Increment NONCNT
Is
K ≥ J ?
NO
C
A-9
YES
Is
L COUNT = 0 ?
YES
No
Set all MCARR(L)
to: 1) current
cube size, 2)
MPINUM, 3) THREE
Clear J Flag
from all MCARR(L)
No
Finished with
all L's?
YES
F
A-11

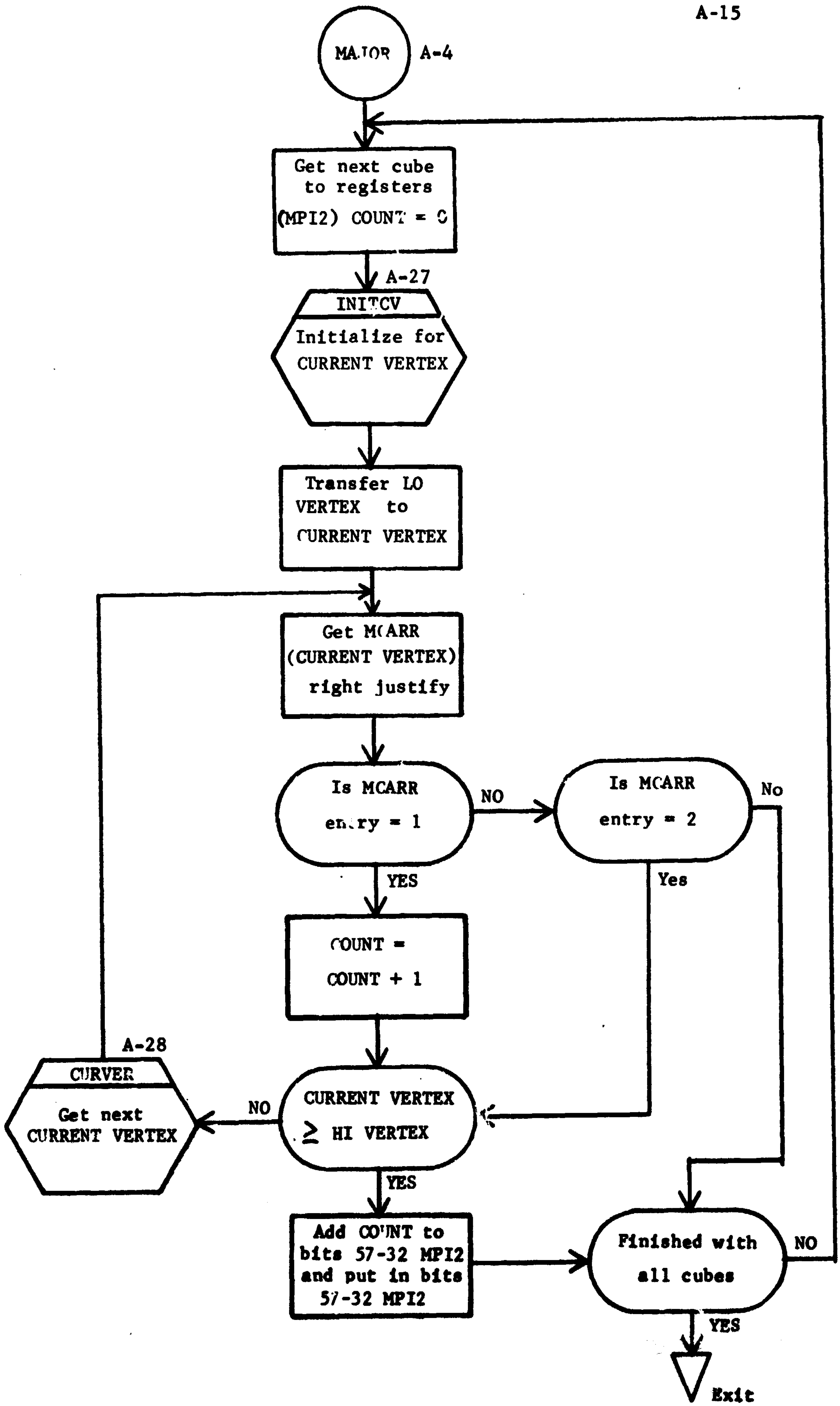


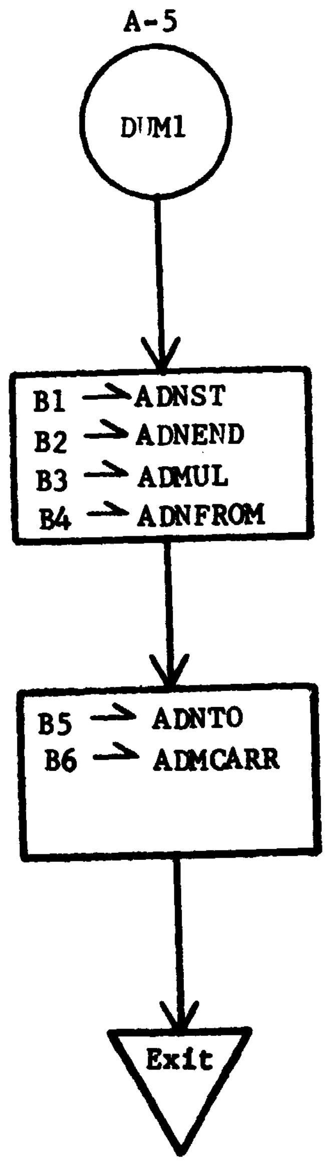
F
A-10
Is
E COUNT = 0 ?
Yes
No
Is
MCARR(E) = 2 ?
Yes
No
Clear J Flag
in MCARR(E)
Clear J Flag
in MCARR(E)
Clear MPINUM
in MCARR(E)
All E's
Processed ?
NO
Yes
Place I and J
in prime implicant
list
A-9
E
Is J ≤ I ?
YES
A
A-9
NO
B
A-9

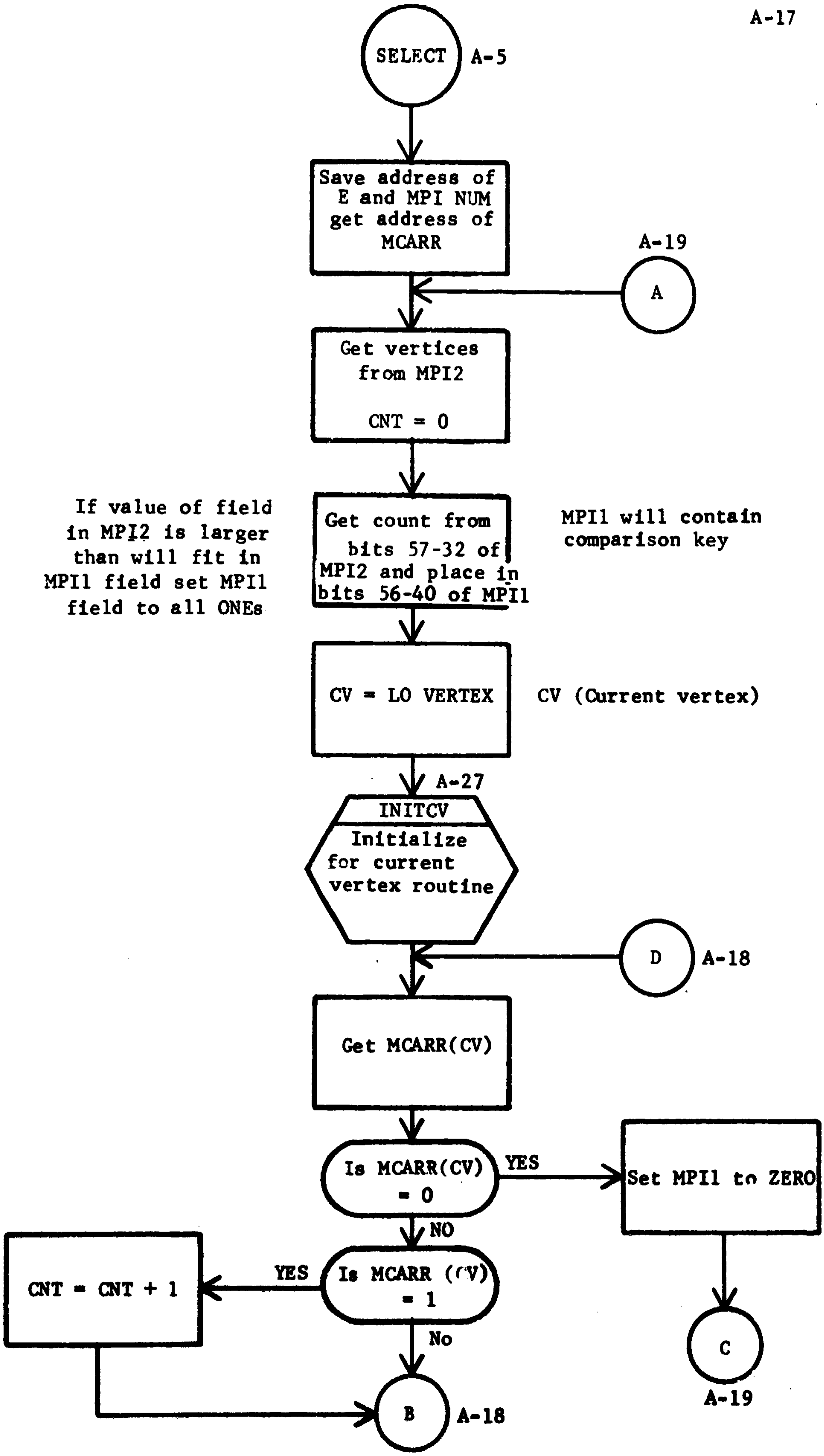

SUBGEN
A-9
Set RESULT = 0
Set MAX (C) =
(2^n-1)-Result &
Output Subcube
Is
MAX(C) =
MIN(C)?
YES
Exit
NO
Set RESULT =
RESULT + MIN (C)
+ 1
Set
RESULT = RESULT
(MIN(C) ● 2^n - 1)

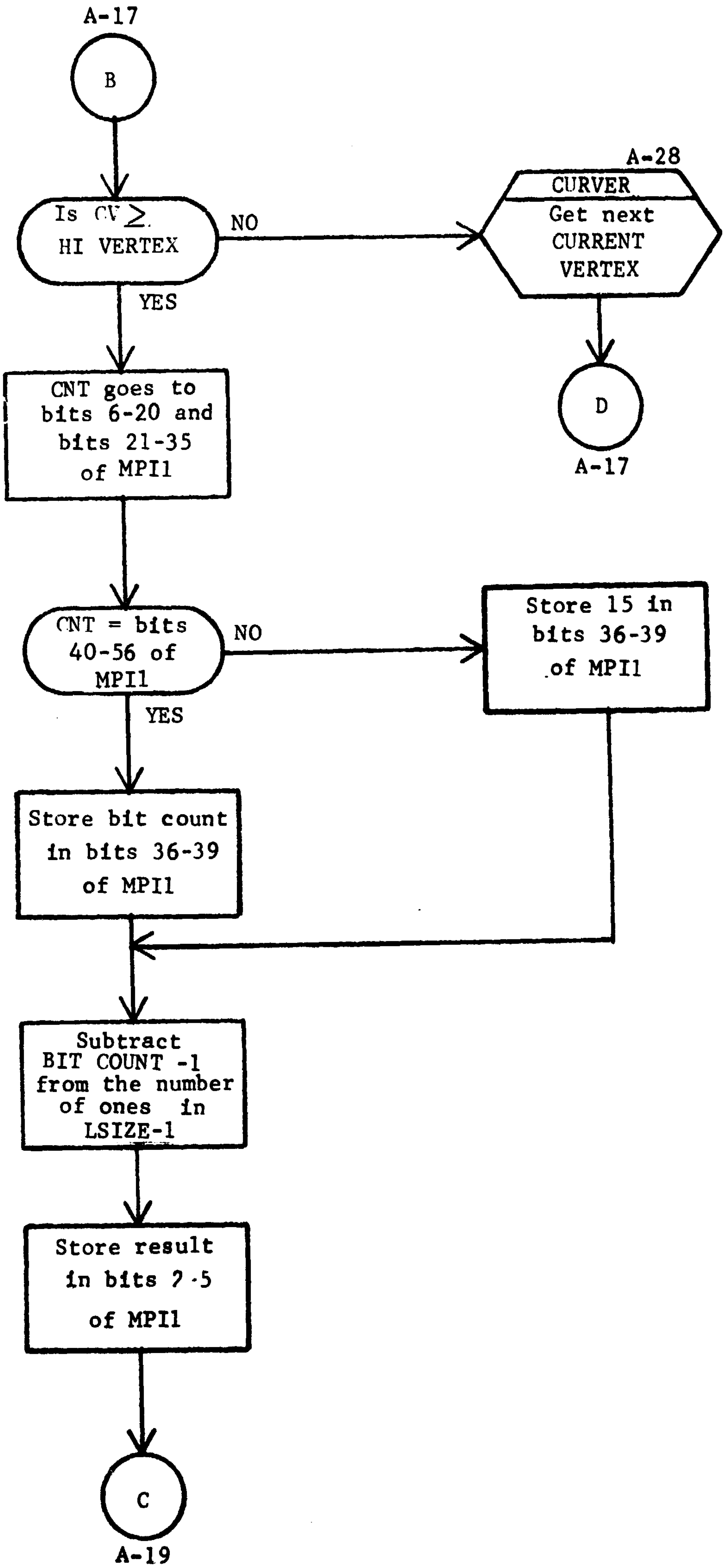

VERGEN
A-9
Set V = MIN(C)
Output Vertex
Is V = MAX(C) ?
YES
Exit
NO
Set V = V OR MAX(C)
Set V = V + 1
Set V = V OR MIN(C)
Set V = V AND MAX(C)

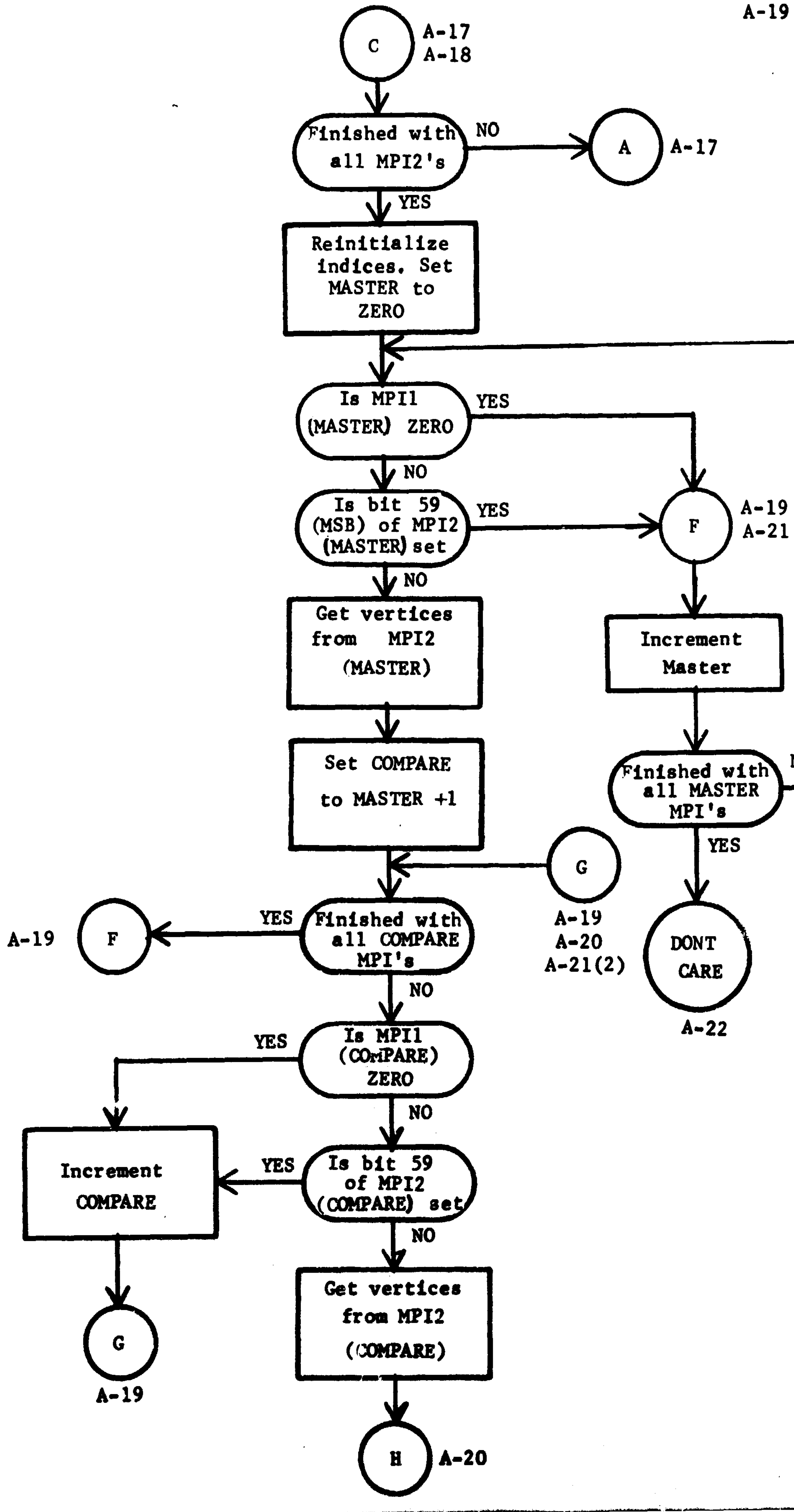

A-4
PACK
Pack low order 16 bits of MPI1 into bits 17-32 of MPI2
NO
Finished with list
YES
Exit

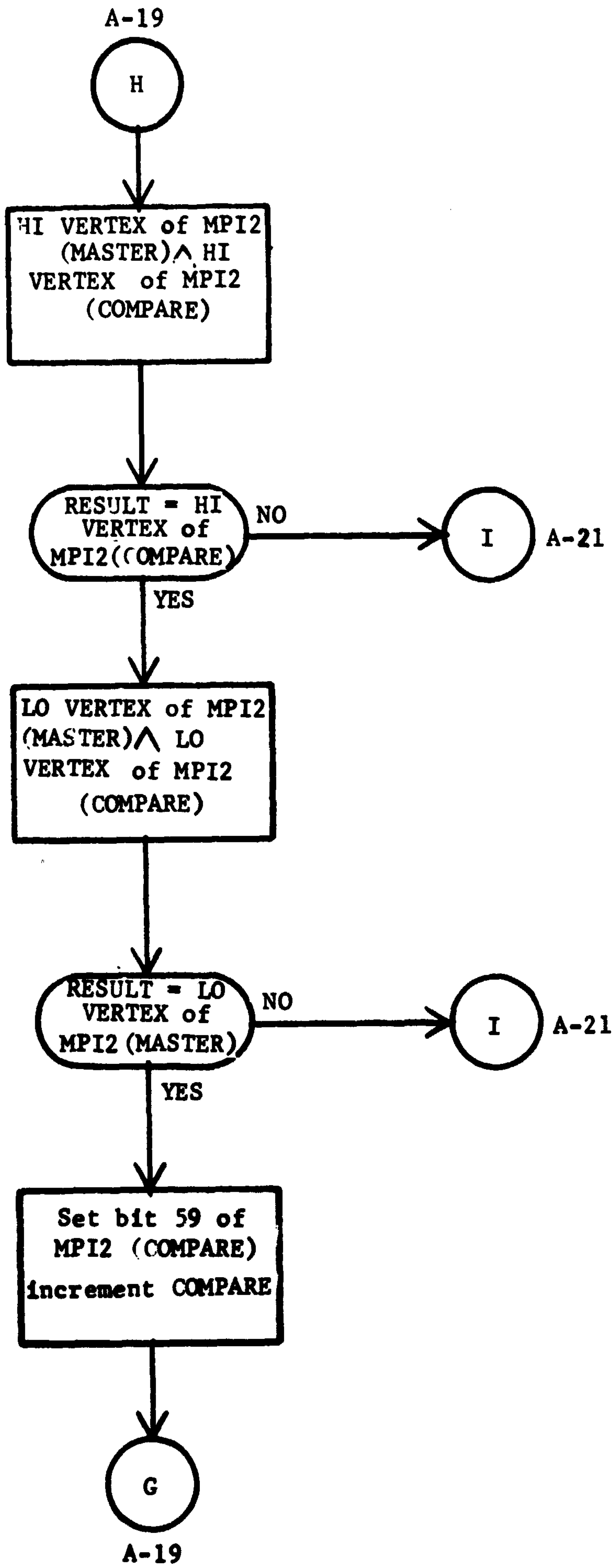

MAJOR
A-4
Get next cube to registers
(MPI2) COUNT = 0
A-27
INITCV
Initialize for CURRENT VERTEX
Transfer LO VERTEX to CURRENT VERTEX
Get MCARR (CURRENT VERTEX) right justify
Is MCARR entry = 1
NO
Is MCARR entry = 2
No
YES
Yes
COUNT = COUNT + 1
A-28
CURVER
Get next CURRENT VERTEX
NO
CURRENT VERTEX ≥ HI VERTEX
YES
Add COUNT to bits 57-32 MPI2 and put in bits 57-32 MPI2
Finished with all cubes
NO
YES
Exit

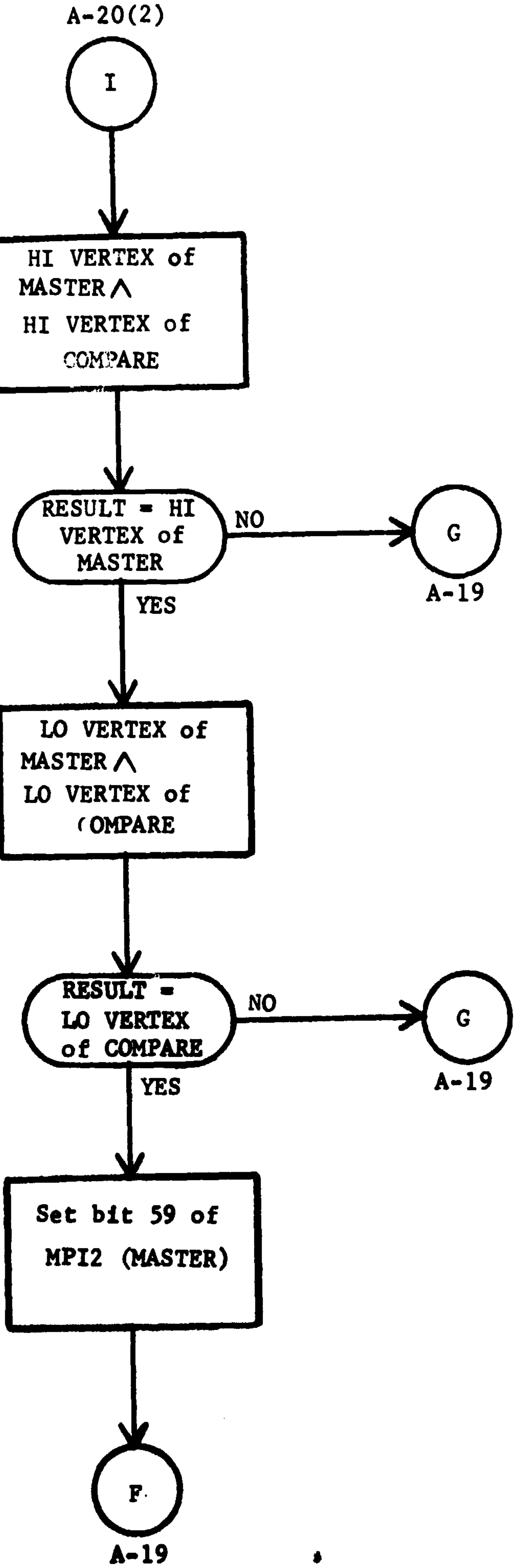

A-5
DUM1
B1 → ADNST
B2 → ADNEND
B3 → ADMUL
B4 → ADNFROM
B5 → ADNTO
B6 → ADMCARR
Exit

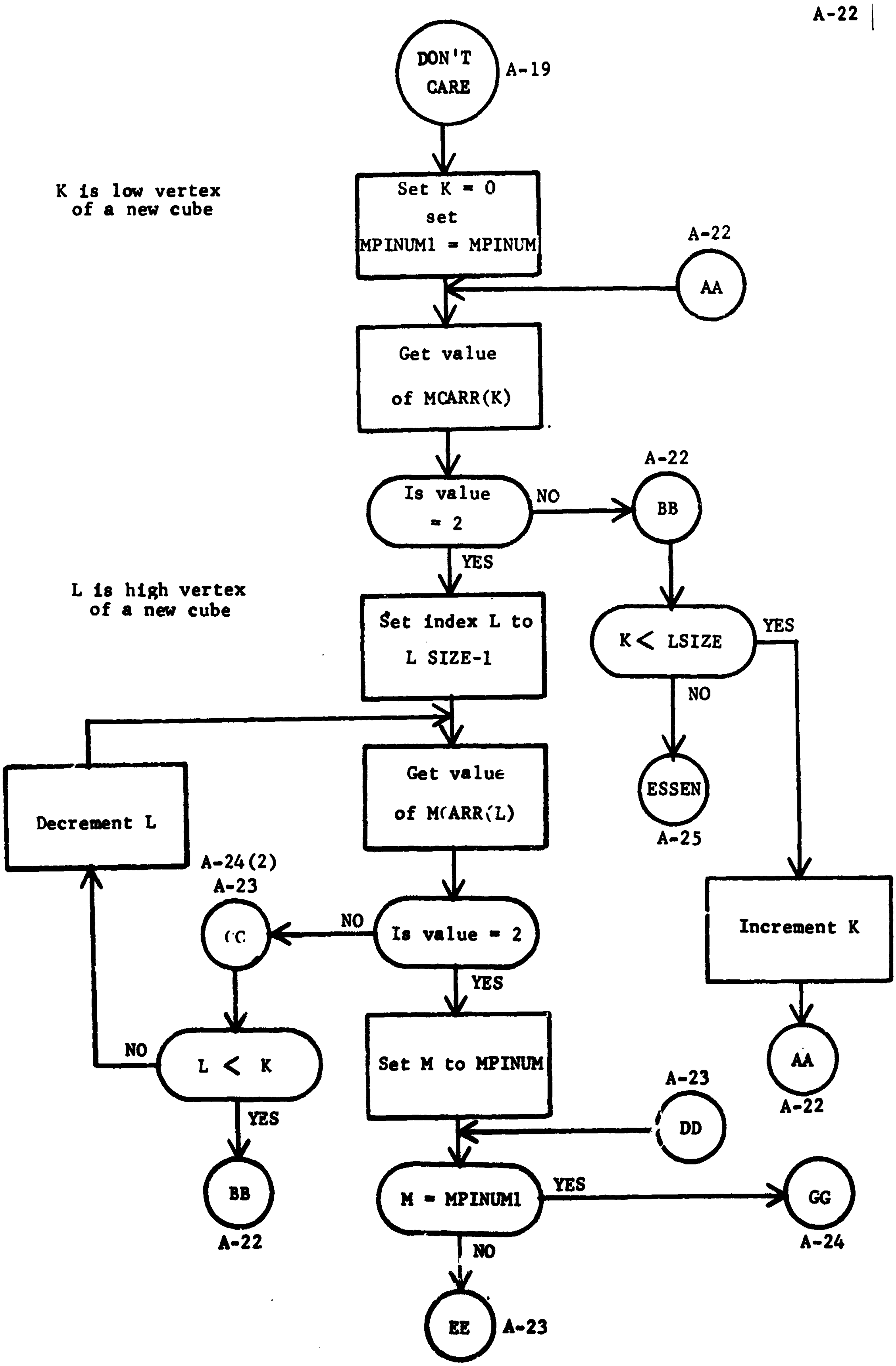

SELECT
A-5
Save address of E and MPI NUM get address of MCARR
A-19
A
Get vertices from MPI2
CNT = 0
If value of field in MPI2 is larger than will fit in MPI1 field set MPI1 field to all ONEs
Get count from bits 57-32 of MPI2 and place in bits 56-40 of MPI1
MPI1 will contain comparison key
CV = LO VERTEX
CV (Current vertex)
A-27
INITCV
Initialize for current vertex routine
D
A-18
Get MCARR(CV)
Is MCARR(CV) = 0
YES
Set MPI1 to ZERO
NO
Is MCARR (CV) = 1
YES
CNT = CNT + 1
No
C
A-19
B
A-18

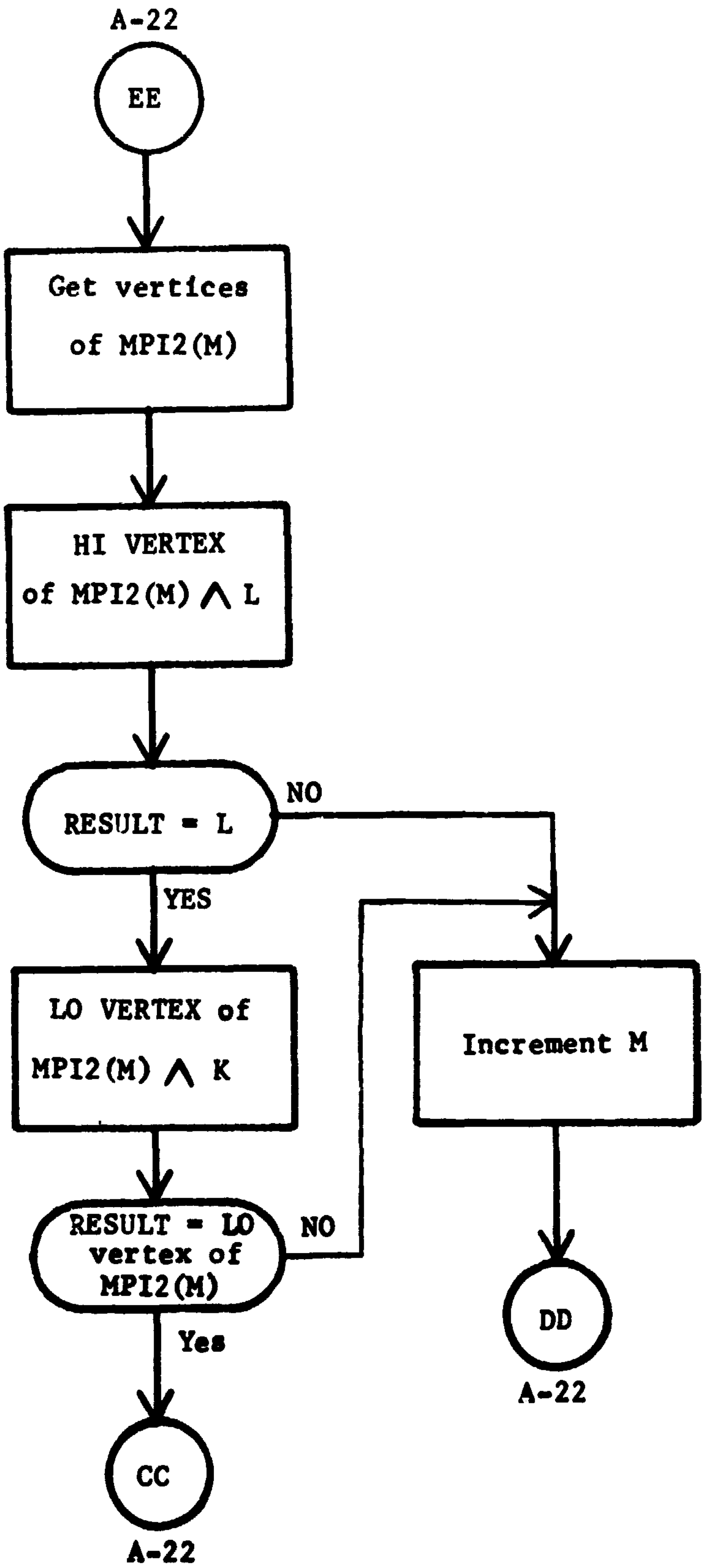

A-17
B
Is CV ≥
HI VERTEX
NO
A-28
CURVER
Get next
CURRENT
VERTEX
D
A-17
YES
CNT goes to
bits 6-20 and
bits 21-35
of MPI1
CNT = bits
40-56 of
MPI1
NO
Store 15 in
bits 36-39
of MPI1
YES
Store bit count
in bits 36-39
of MPI1
Subtract
BIT COUNT -1
from the number
of ones in
LSIZE-1
Store result
in bits 2-5
of MPI1
C
A-19

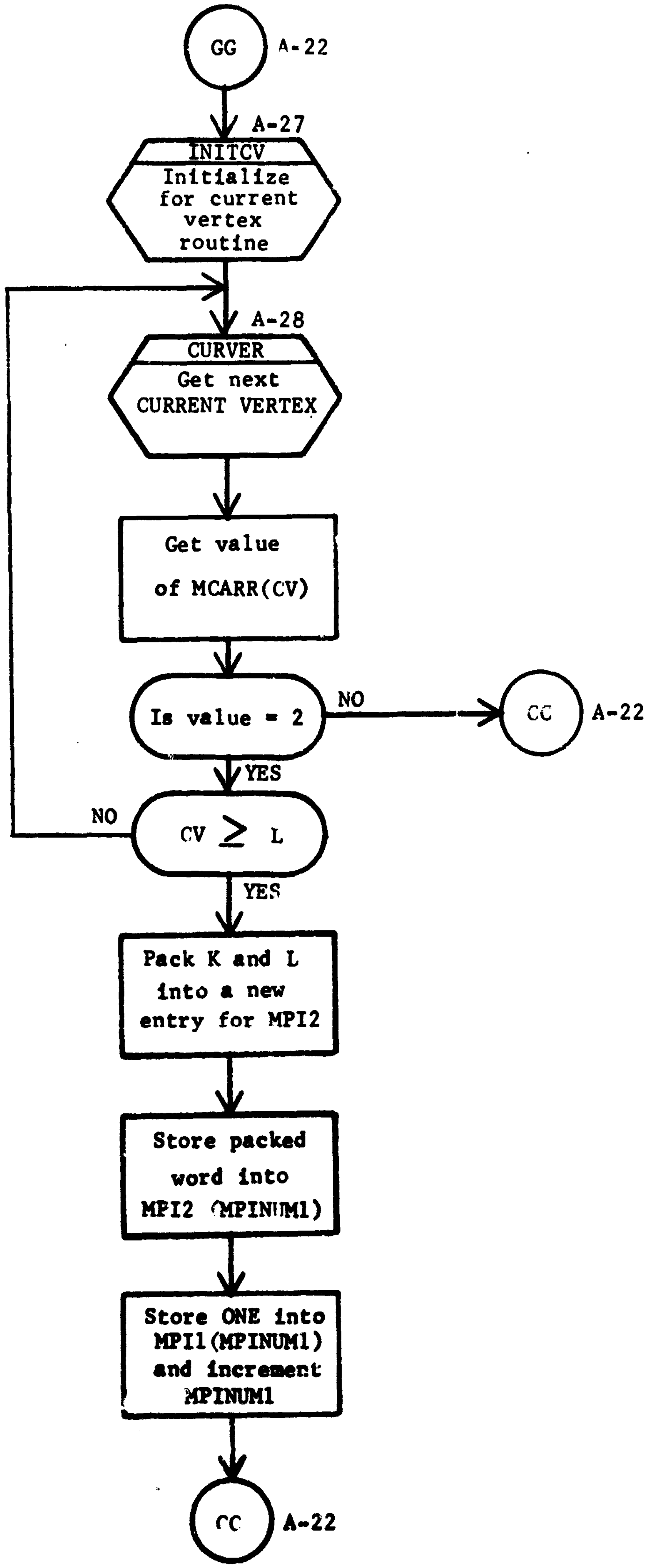

C
A-17
A-18
Finished with all MPI2's
NO
A
A-17
YES
Reinitialize indices. Set MASTER to ZERO
Is MPI1 (MASTER) ZERO
YES
NO
Is bit 59 (MSB) of MPI2 (MASTER) set
YES
F
A-19
A-21
NO
Get vertices from MPI2 (MASTER)
Increment Master
Set COMPARE to MASTER +1
Finished with all MASTER MPI's
NO
YES
G
A-19
A-20
A-21(2)
DONT CARE
A-22
A-19
F
YES
Finished with all COMPARE MPI's
NO
YES
Is MPI1 (COMPARE) ZERO
NO
Increment COMPARE
YES
Is bit 59 of MPI2 (COMPARE) set
NO
Get vertices from MPI2 (COMPARE)
G
A-19
H
A-20

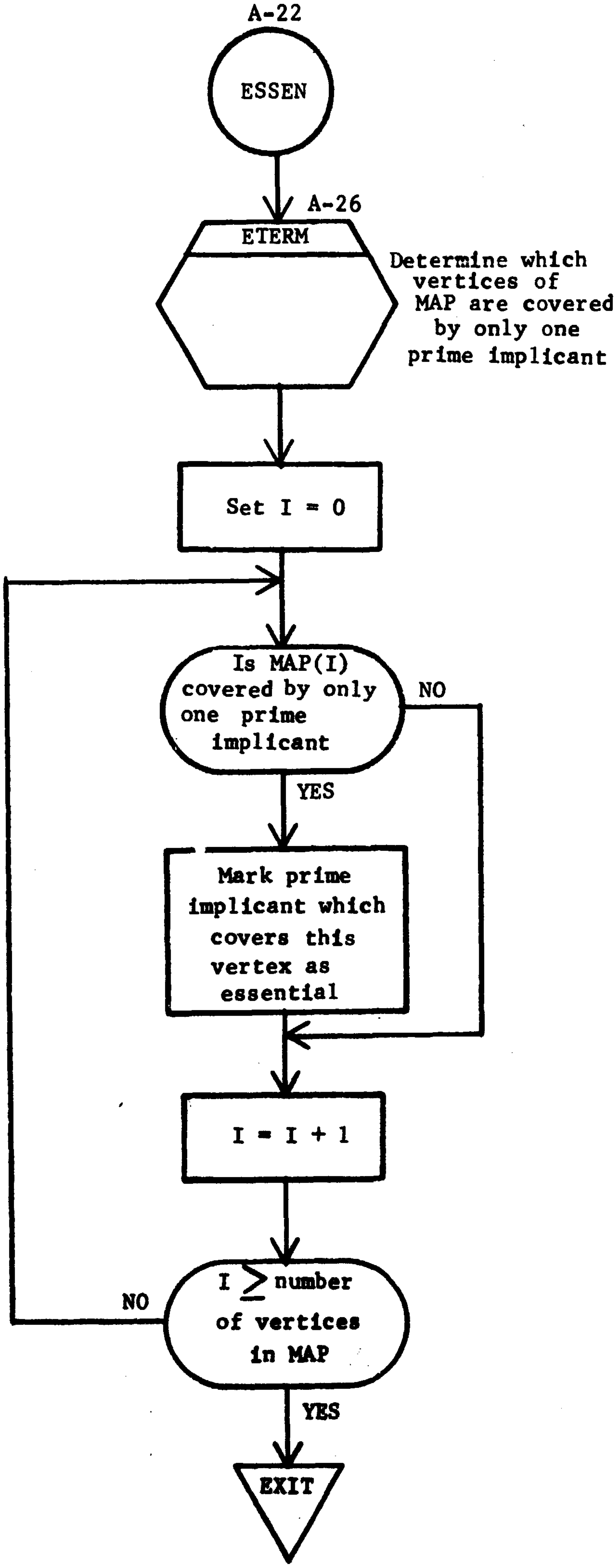

A-19
H
HI VERTEX of MPI2 (MASTER)∧ HI VERTEX of MPI2 (COMPARE)
RESULT = HI VERTEX of MPI2(COMPARE)
NO
I
A-21
YES
LO VERTEX of MPI2 (MASTER)∧ LO VERTEX of MPI2 (COMPARE)
RESULT = LO VERTEX of MPI2 (MASTER)
NO
I
A-21
YES
Set bit 59 of MPI2 (COMPARE) increment COMPARE
G
A-19

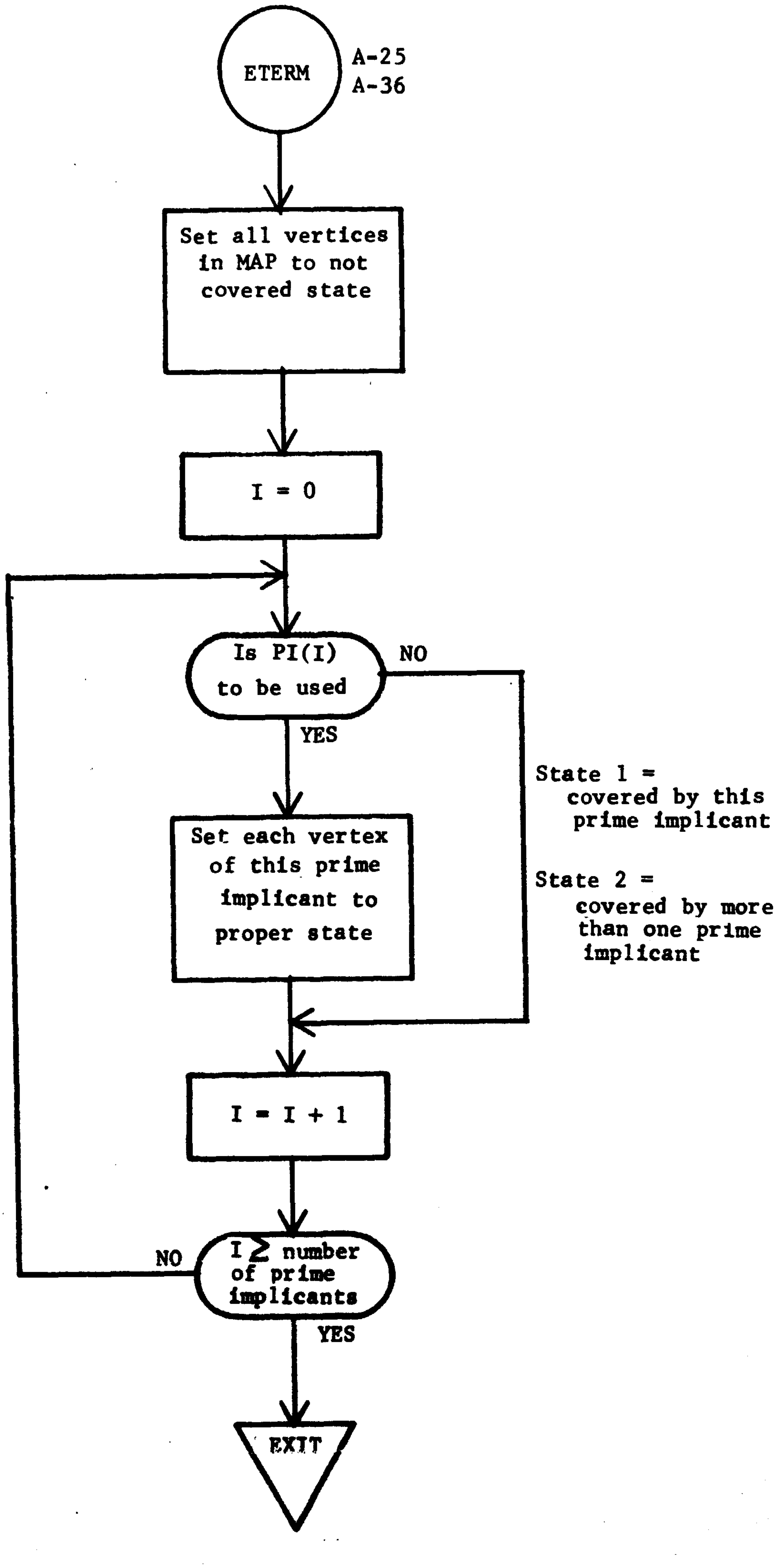

A-20(2)
I
HI VERTEX of MASTER ∧ HI VERTEX of COMPARE
RESULT = HI VERTEX of MASTER
NO
G
A-19
YES
LO VERTEX of MASTER ∧ LO VERTEX of COMPARE
RESULT = LO VERTEX of COMPARE
NO
G
A-19
YES
Set bit 59 of MPI2 (MASTER)
F
A-19

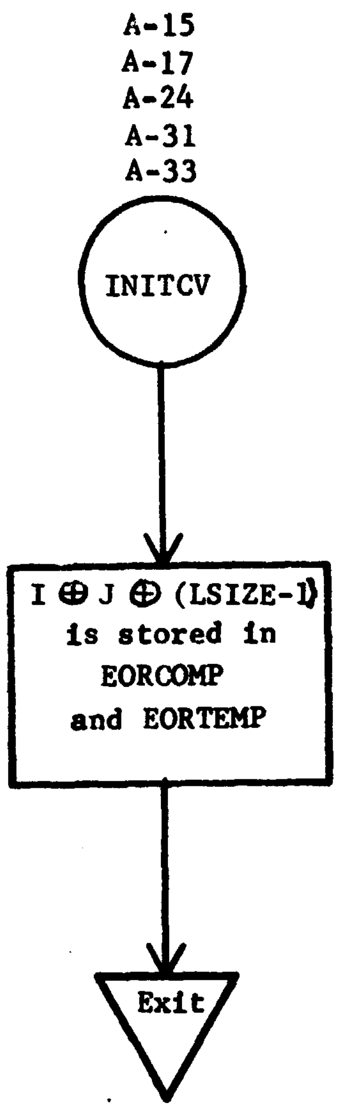

DON'T CARE
A-19
K is low vertex
of a new cube
Set K = 0
set
MPINUM1 = MPINUM
A-22
AA
Get value
of MCARR(K)
A-22
Is value
= 2
NO
BB
YES
L is high vertex
of a new cube
Set index L to
L SIZE-1
K < LSIZE
YES
NO
ESSEN
A-25
Get value
of MCARR(L)
Decrement L
A-24(2)
A-23
CC
NO
Is value = 2
YES
Increment K
NO
L < K
Set M to MPINUM
AA
A-22
A-23
DD
YES
BB
A-22
M = MPINUM1
YES
GG
A-24
NO
EE
A-23


A-22
EE
Get vertices of MPI2(M)
HI VERTEX of MPI2(M) ∧ L
RESULT = L
NO
YES
LO VERTEX of MPI2(M) ∧ K
Increment M
RESULT = LO vertex of MPI2(M)
NO
Yes
DD
A-22
CC
A-22


GG
A-22
A-27
INITCV
Initialize for current vertex routine
A-28
CURVER
Get next CURRENT VERTEX
Get value of MCARR(CV)
Is value = 2
NO
CC
A-22
YES
NO
CV ≥ L
YES
Pack K and L into a new entry for MPI2
Store packed word into MPI2 (MPINUM1)
Store ONE into MPI1(MPINUM1) and increment MPINUM1
CC
A-22


A-22
ESSEN
A-26
ETERM
Determine which vertices of MAP are covered by only one prime implicant
Set I = 0
Is MAP(I) covered by only one prime implicant
NO
YES
Mark prime implicant which covers this vertex as essential
I = I + 1
I ≥ number of vertices in MAP
NO
YES
EXIT


ETERM
A-25
A-36
Set all vertices in MAP to not covered state
I = 0
Is PI(I) to be used
NO
YES
State 1 = covered by this prime implicant
State 2 = covered by more than one prime implicant
Set each vertex of this prime implicant to proper state
I = I + 1
NO
I ≥ number of prime implicants
YES
EXIT



NOTE:

Required parameters

LSIZE = size of Karnaugh map

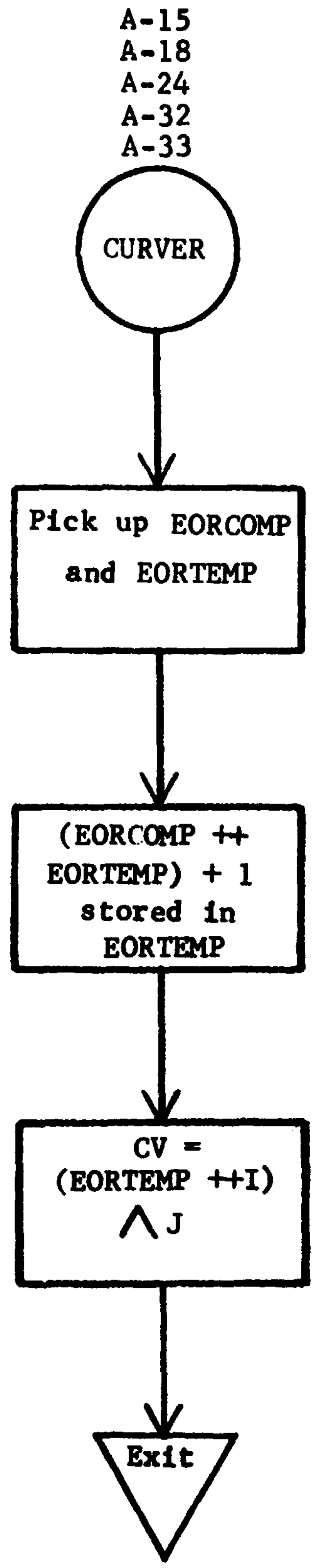
J = HI VERTEX

I = LO VERTEX

$\wedge$ = AND

++ = OR

$\oplus$ = Exclusive OR



NOTE:

Required Parameters

I = LOW VERTEX

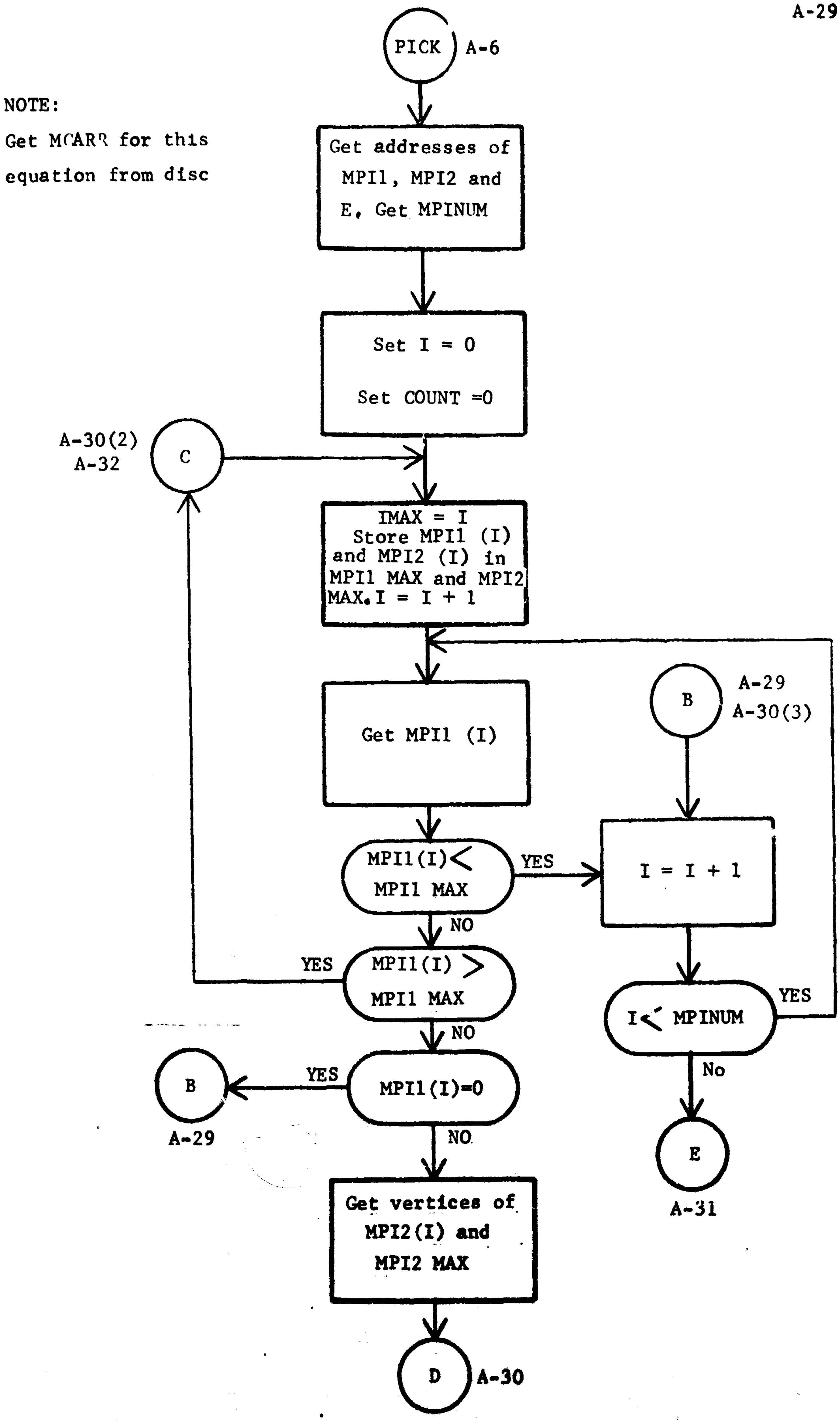
J = HI VERTEX

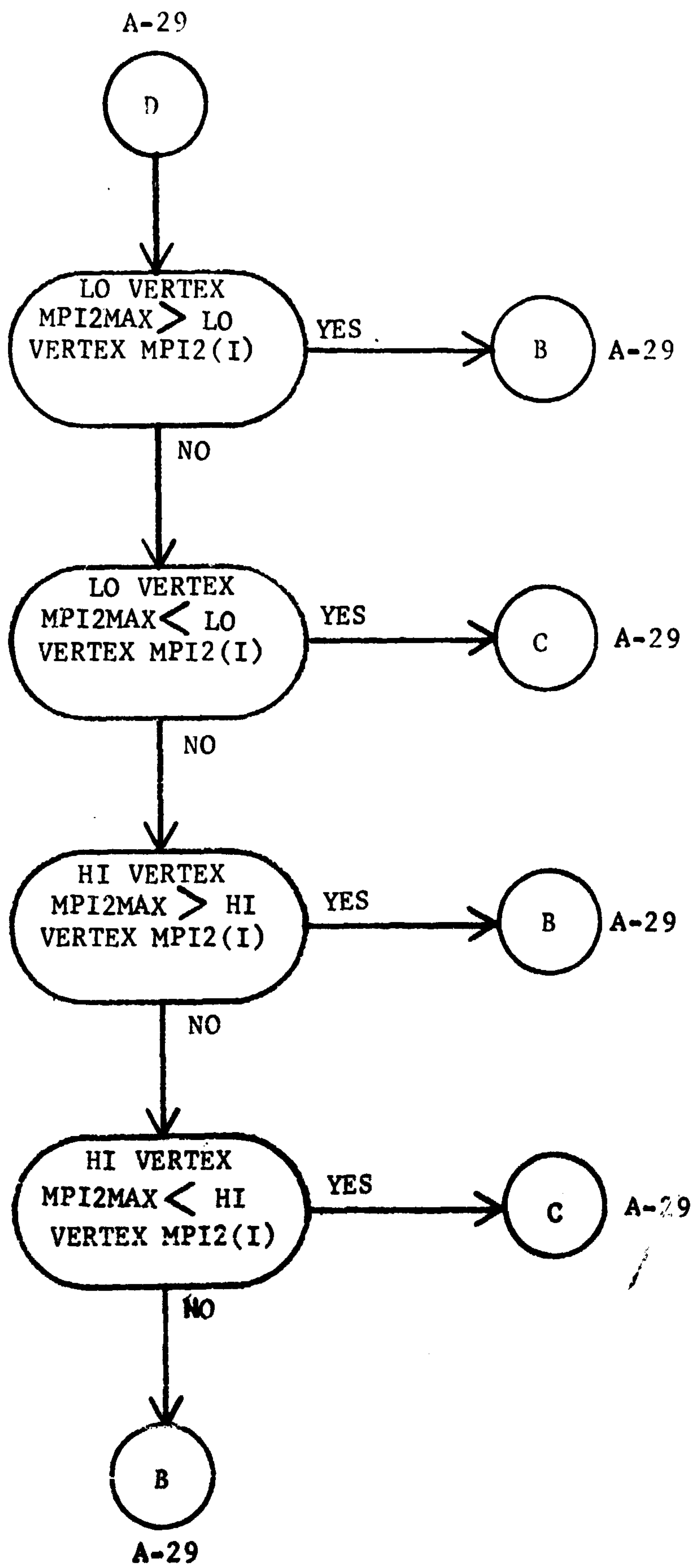
CV = CURRENT VERTEX

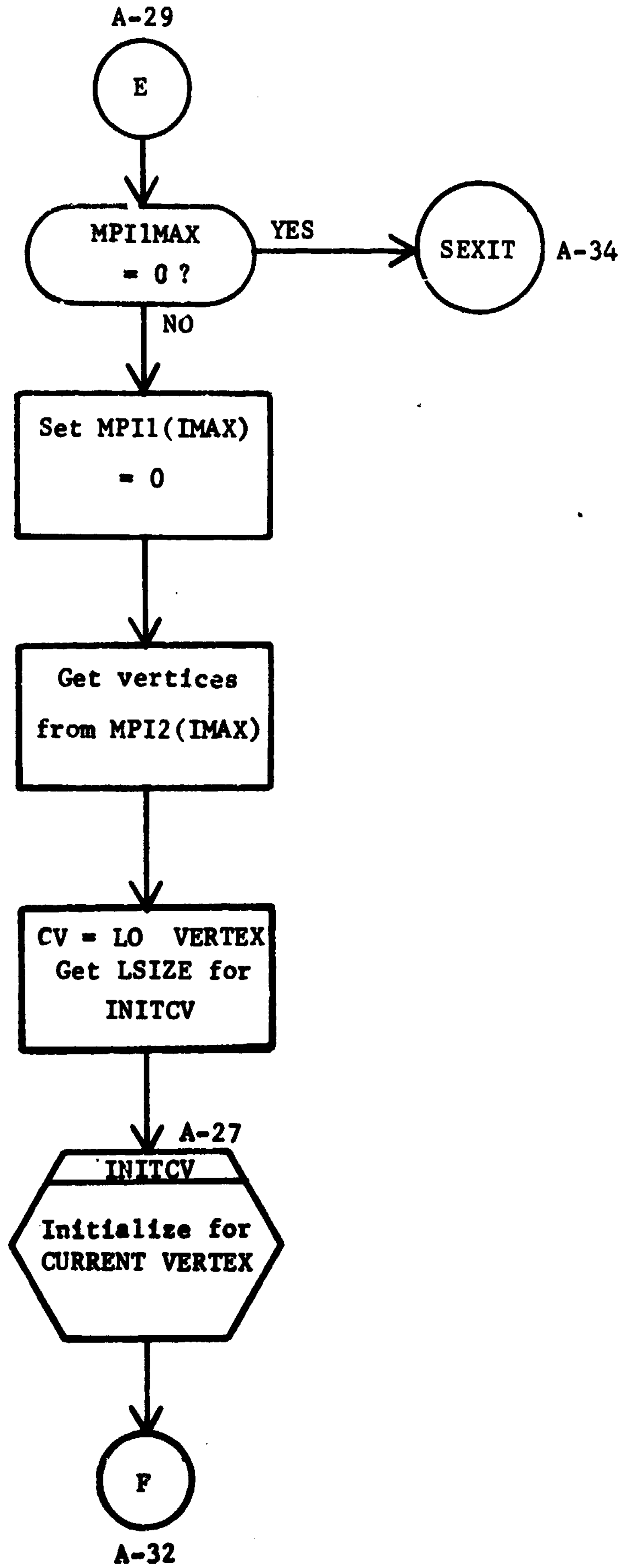
⋀ = AND

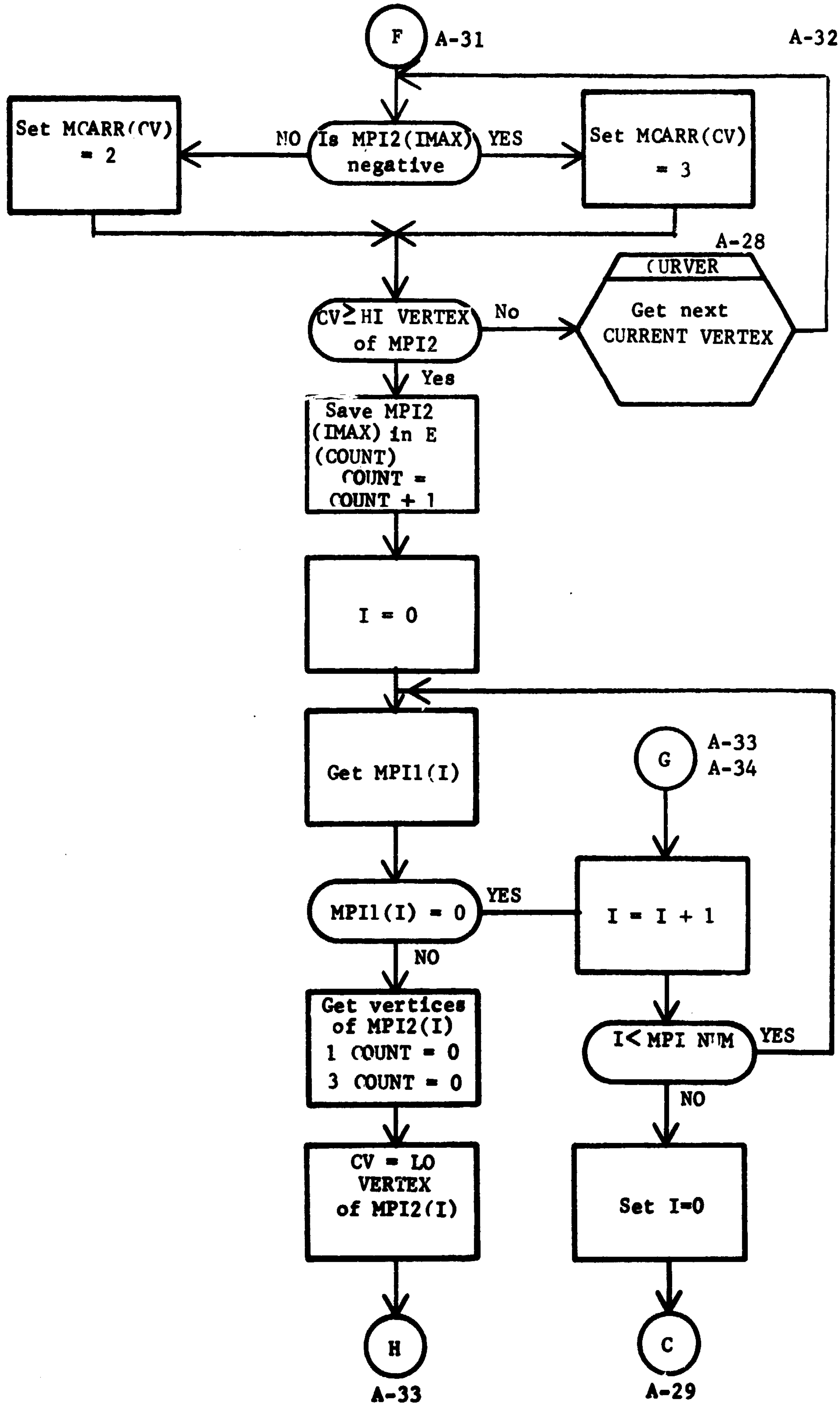
++ = OR

⊕ = Exclusive OR

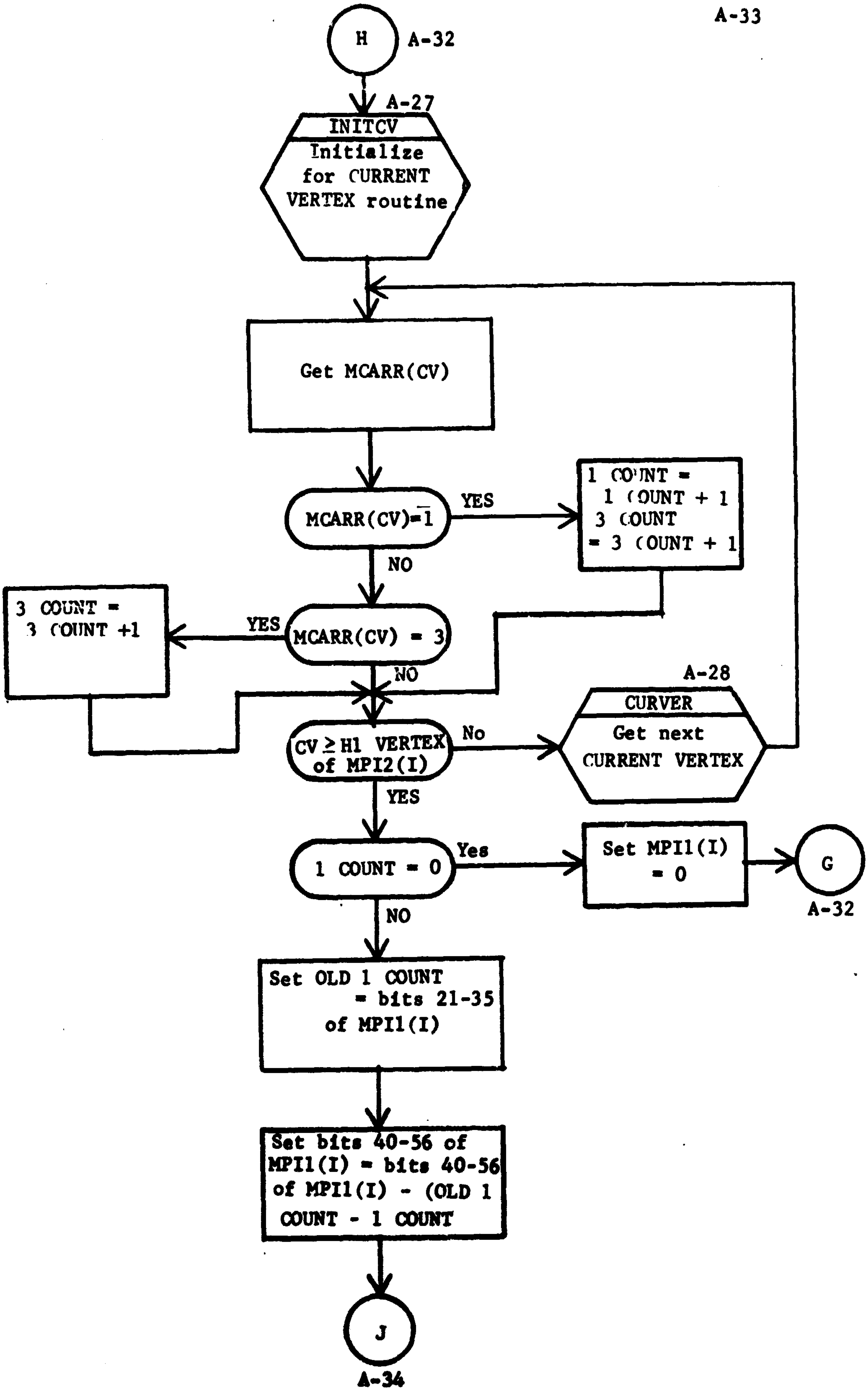

NOTE:
Get MCARR for this equation from disc
PICK
A-6
Get addresses of MPI1, MPI2 and E, Get MPINUM
Set I = 0
Set COUNT =0
A-30(2)
A-32
C
IMAX = I Store MPI1 (I) and MPI2 (I) in MPI1 MAX and MPI2 MAX. I = I + 1
Get MPI1 (I)
B
A-29
A-30(3)
MPI1(I)< MPI1 MAX
YES
NO
I = I + 1
YES
MPI1(I) > MPI1 MAX
NO
I < MPINUM
YES
No
B
A-29
YES
MPI1(I)=0
NO
E
A-31
Get vertices of MPI2(I) and MPI2 MAX
D
A-30

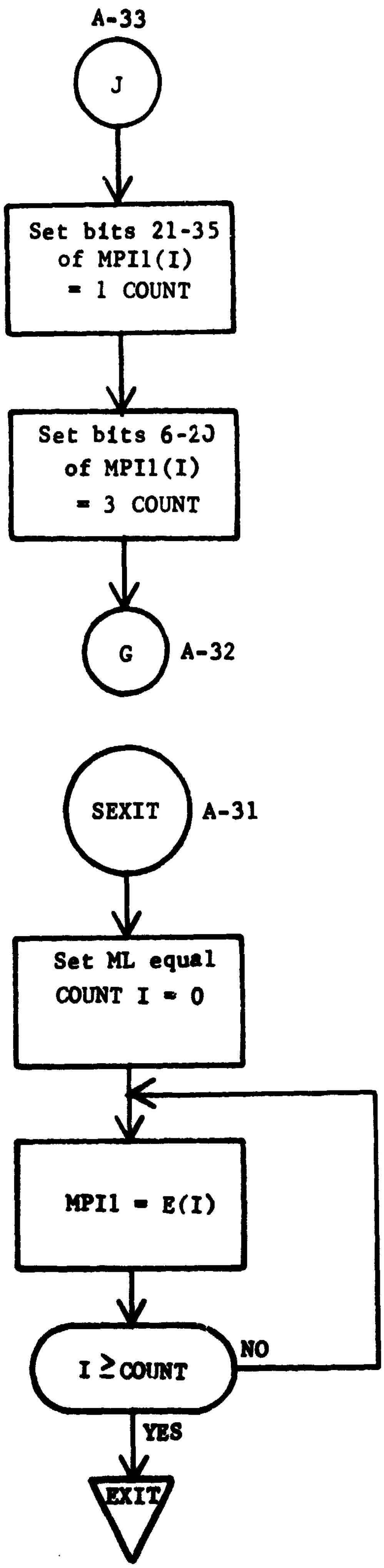

A-29
D
LO VERTEX MPI2MAX > LO VERTEX MPI2(I)
YES
B
A-29
NO
LO VERTEX MPI2MAX < LO VERTEX MPI2(I)
YES
C
A-29
NO
HI VERTEX MPI2MAX > HI VERTEX MPI2(I)
YES
B
A-29
NO
HI VERTEX MPI2MAX < HI VERTEX MPI2(I)
YES
C
NO
B
A-29

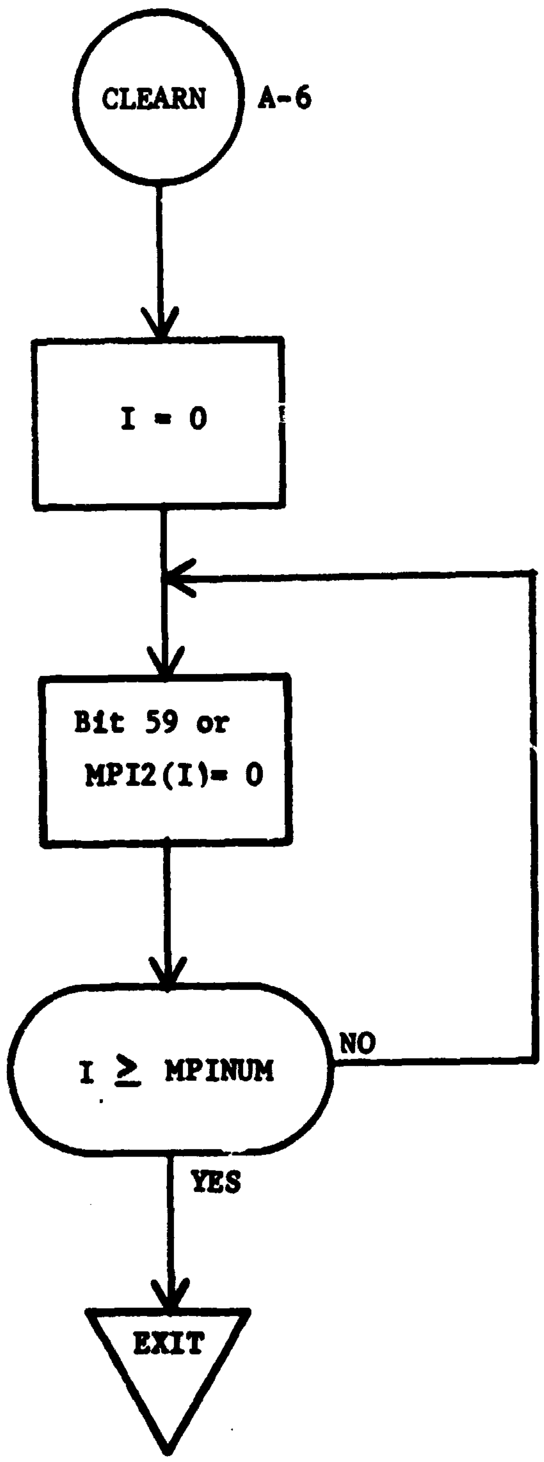

A-29
E
MPI1MAX
= 0 ?
YES
SEXIT
A-34
NO
Set MPI1(IMAX)
= 0
Get vertices
from MPI2(IMAX)
CV = LO VERTEX
Get LSIZE for
INITCV
A-27
INITCV
Initialize for
CURRENT VERTEX
F
A-32

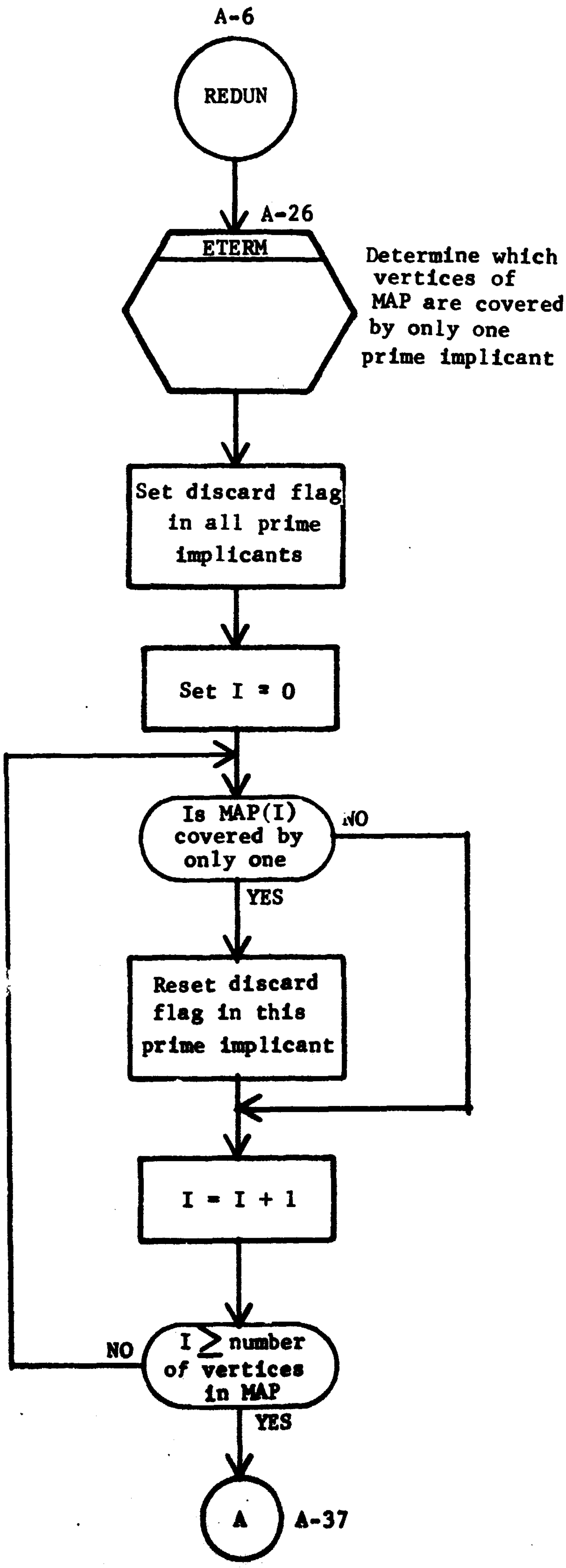


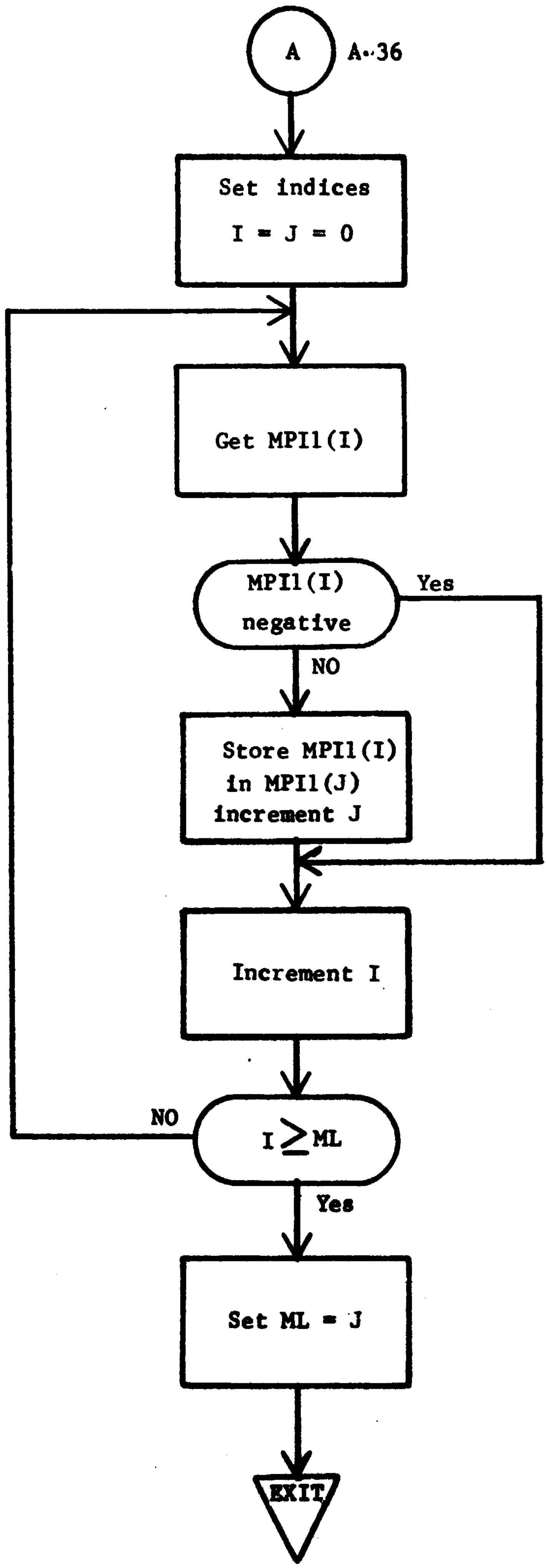
F
A-31
A-32
Set MCARR(CV) = 2
NO
Is MPI2(IMAX) negative
YES
Set MCARR(CV) = 3
A-28
CURVER
Get next CURRENT VERTEX
CV≥HI VERTEX of MPI2
No
Yes
Save MPI2 (IMAX) in E (COUNT) COUNT = COUNT + 1
I = 0
Get MPI1(I)
G
A-33
A-34
MPI1(I) = 0
YES
I = I + 1
NO
Get vertices of MPI2(I) 1 COUNT = 0 3 COUNT = 0
I< MPI NUM
YES
NO
CV = LO VERTEX of MPI2(I)
Set I=0
H
A-33
C
A-29

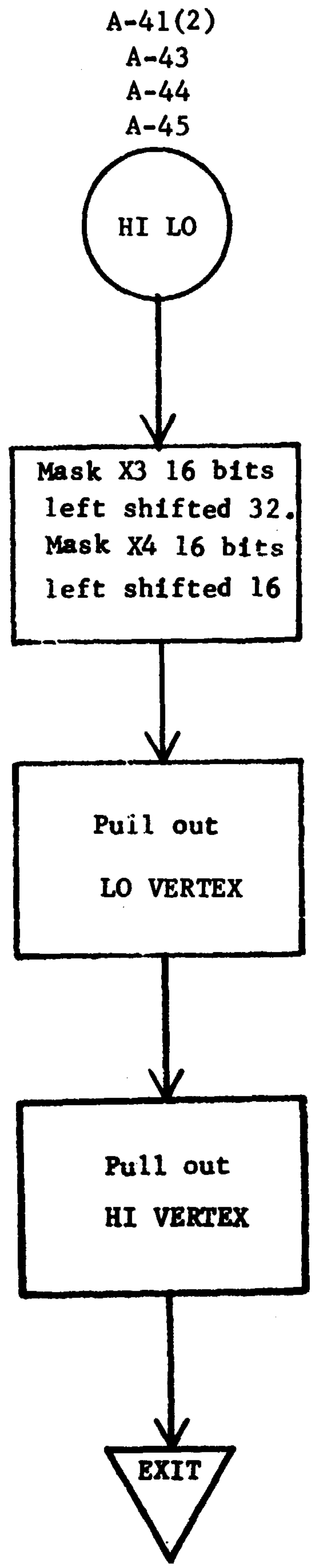

H
A-32
A-27
INITCV
Initialize for CURRENT VERTEX routine
Get MCARR(CV)
MCARR(CV)=1̄
YES
1 COUNT = 1 COUNT + 1
3 COUNT = 3 COUNT + 1
NO
3 COUNT = 3 COUNT +1
YES
MCARR(CV) = 3
NO
A-28
CURVER
Get next CURRENT VERTEX
CV ≥ H1 VERTEX of MPI2(I)
No
YES
1 COUNT = 0
Yes
Set MPI1(I) = 0
G
A-32
NO
Set OLD 1 COUNT = bits 21-35 of MPI1(I)
Set bits 40-56 of MPI1(I) = bits 40-56 of MPI1(I) - (OLD 1 COUNT - 1 COUNT
J
A-34

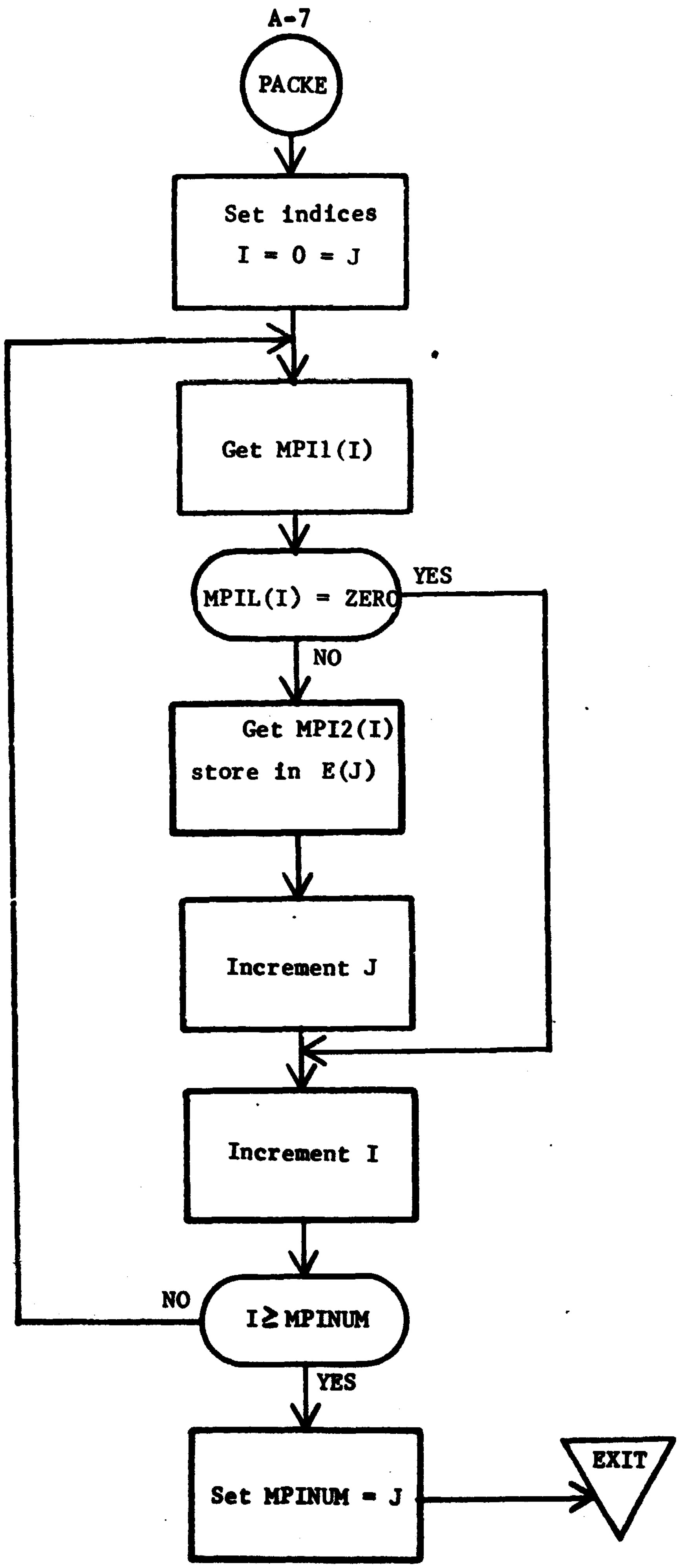

A-33
J
Set bits 21-35
of MPI1(I)
= 1 COUNT
Set bits 6-20
of MPI1(I)
= 3 COUNT
G
A-32
SEXIT
A-31
Set ML equal
COUNT I = 0
MPI1 = E(I)
I ≥ COUNT
NO
YES
EXIT


CLEARN
A-6
I = 0
Bit 59 or
MPI2(I)= 0
I ≥ MPINUM
NO
YES
EXIT

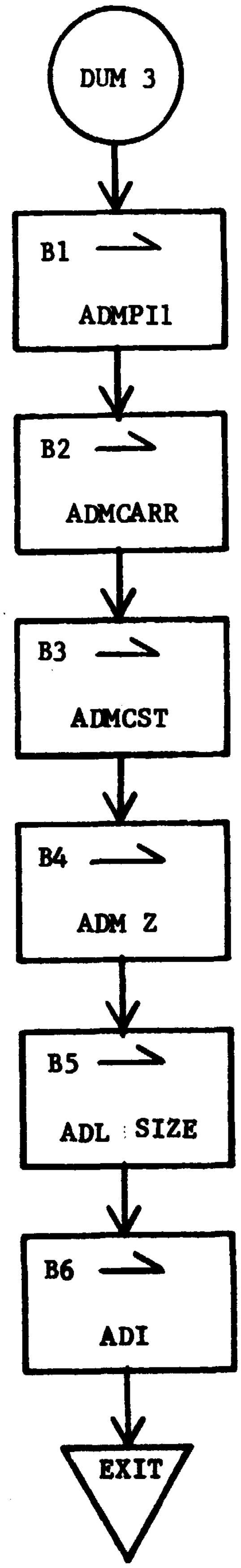

A-6
REDUN
A-26
ETERM
Determine which vertices of MAP are covered by only one prime implicant
Set discard flag in all prime implicants
Set I = 0
Is MAP(I) covered by only one
NO
YES
Reset discard flag in this prime implicant
I = I + 1
I ≥ number of vertices in MAP
NO
YES
A
A-37


A
A-36
Set indices
I = J = 0
Get MPI1(I)
MPI1(I)
negative
Yes
NO
Store MPI1(I)
in MPI1(J)
increment J
Increment I
NO
I ≥ ML
Yes
Set ML = J
EXIT


A-41(2)
A-43
A-44
A-45
HI LO
Mask X3 16 bits left shifted 32.
Mask X4 16 bits left shifted 16
Pull out
LO VERTEX
Pull out
HI VERTEX
EXIT


A-7
PACKE
Set indices
I = 0 = J
Get MPI1(I)
MPIL(I) = ZERO
YES
NO
Get MPI2(I)
store in E(J)
Increment J
Increment I
NO
I ≥ MPINUM
YES
Set MPINUM = J
EXIT

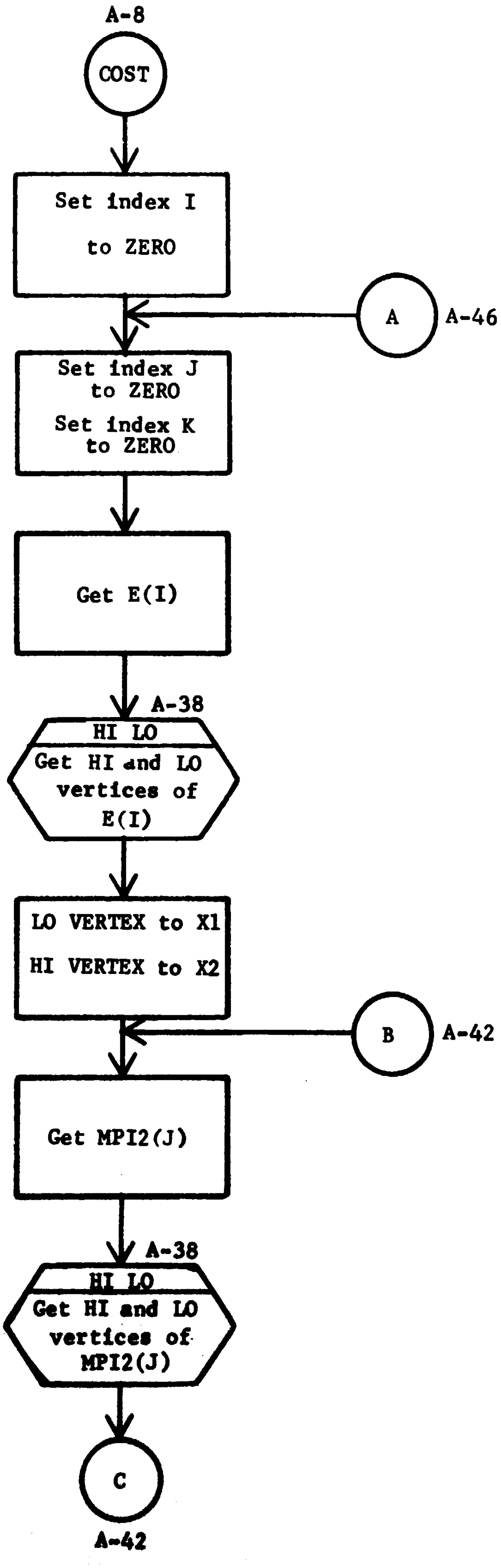
A-7

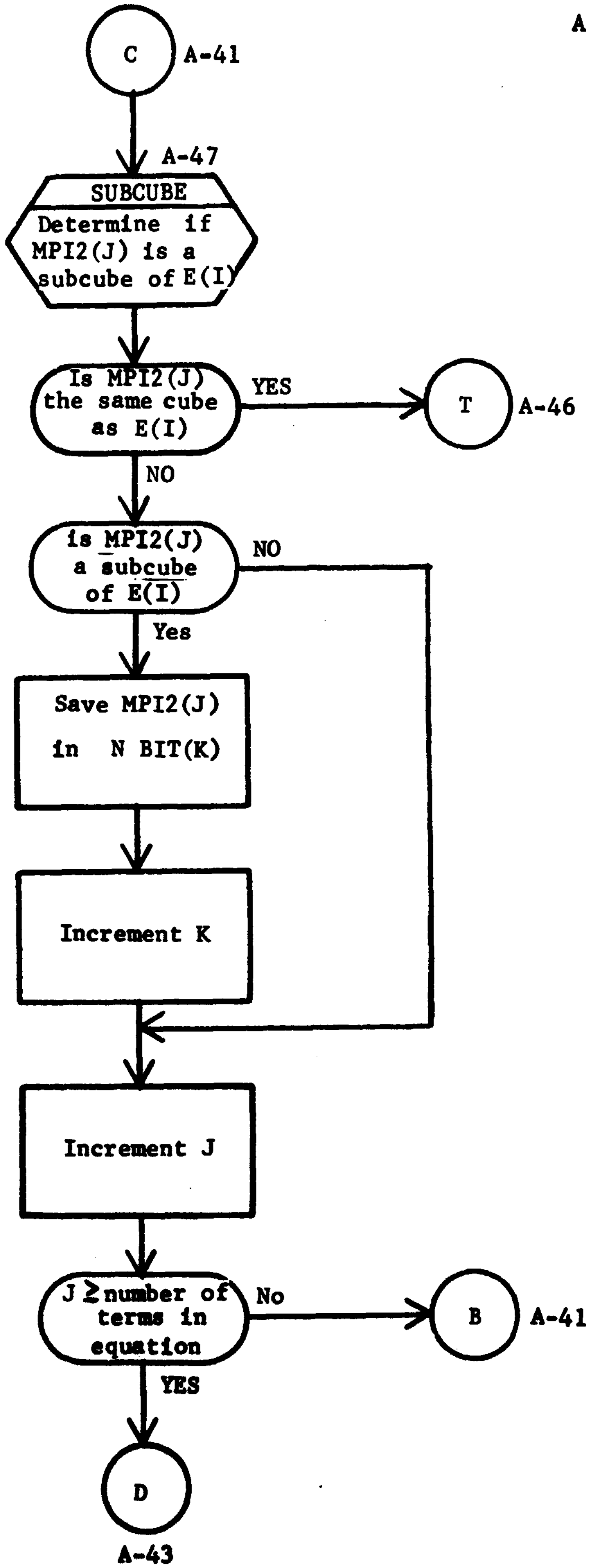

DUM 3
B1
ADMPI1
B2
ADMCARR
B3
ADMCST
B4
ADM Z
B5
ADL SIZE
B6
ADI
EXIT

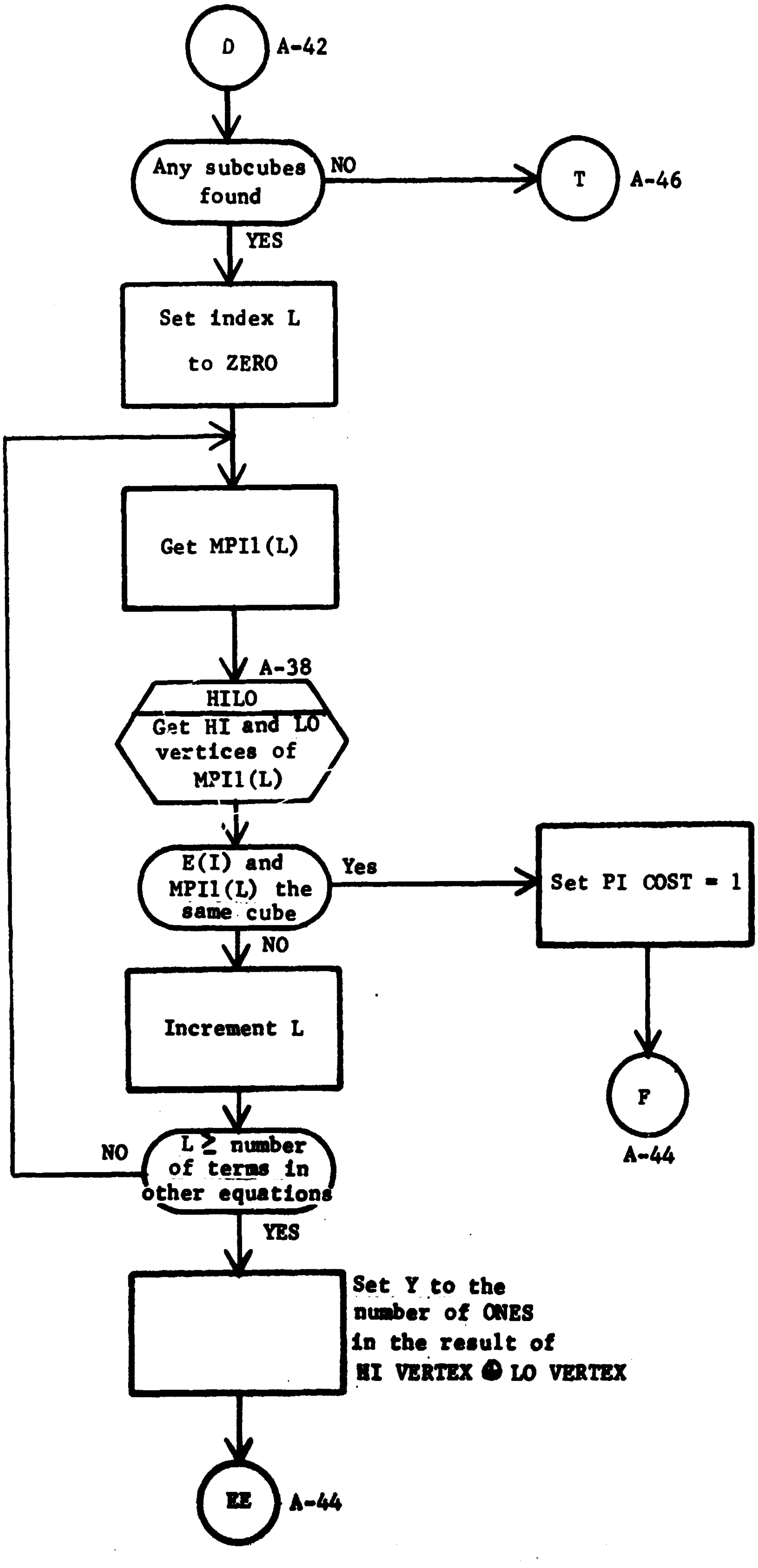
NOTE:

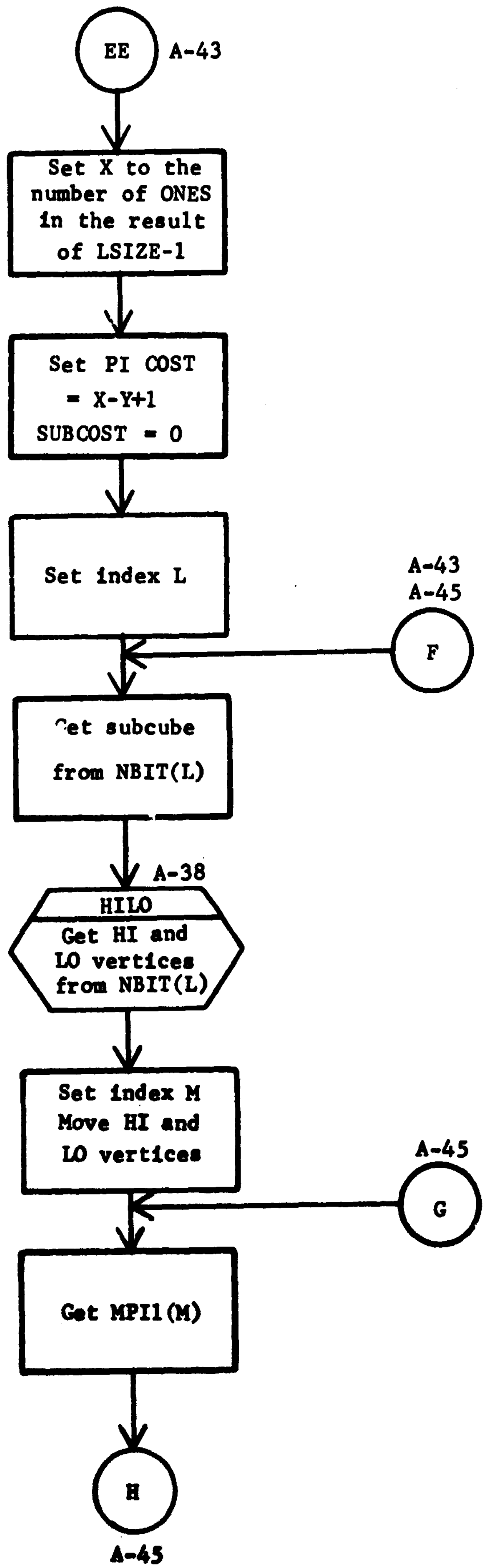
MPI2 contains prime implicants for this equation

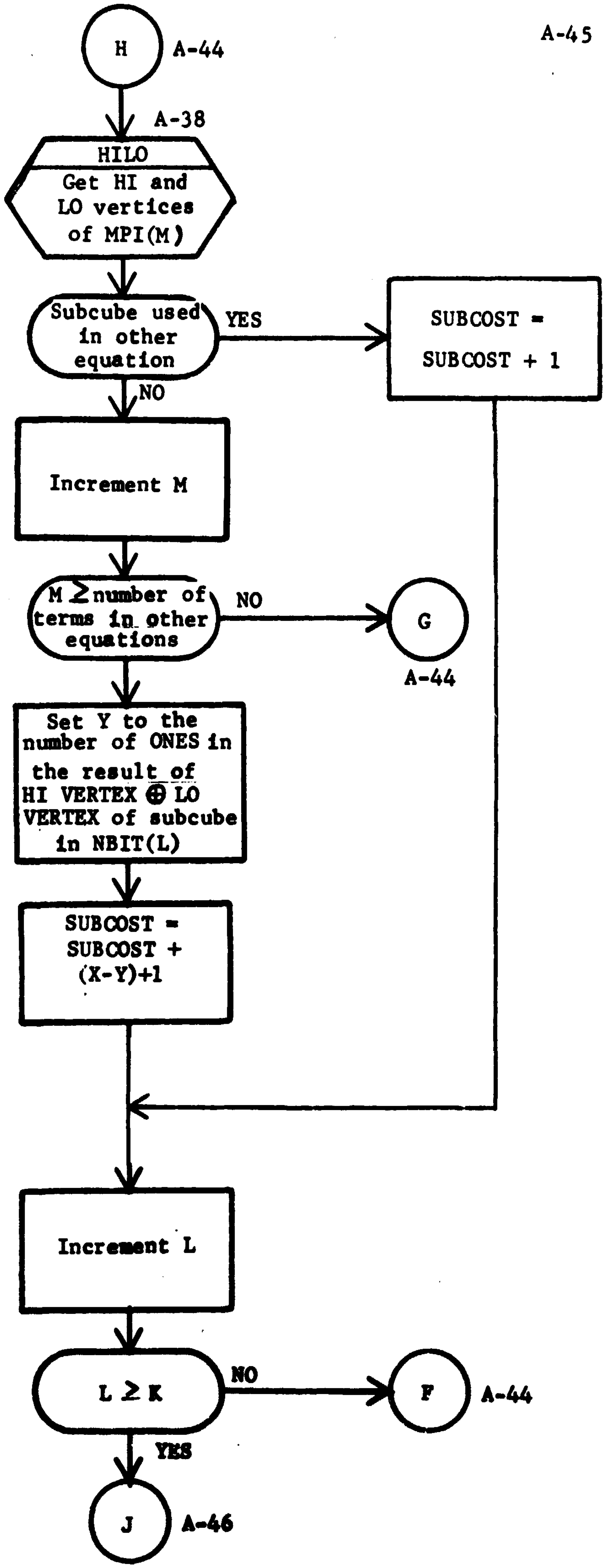
MPI1 contains prime implicants for all other equations

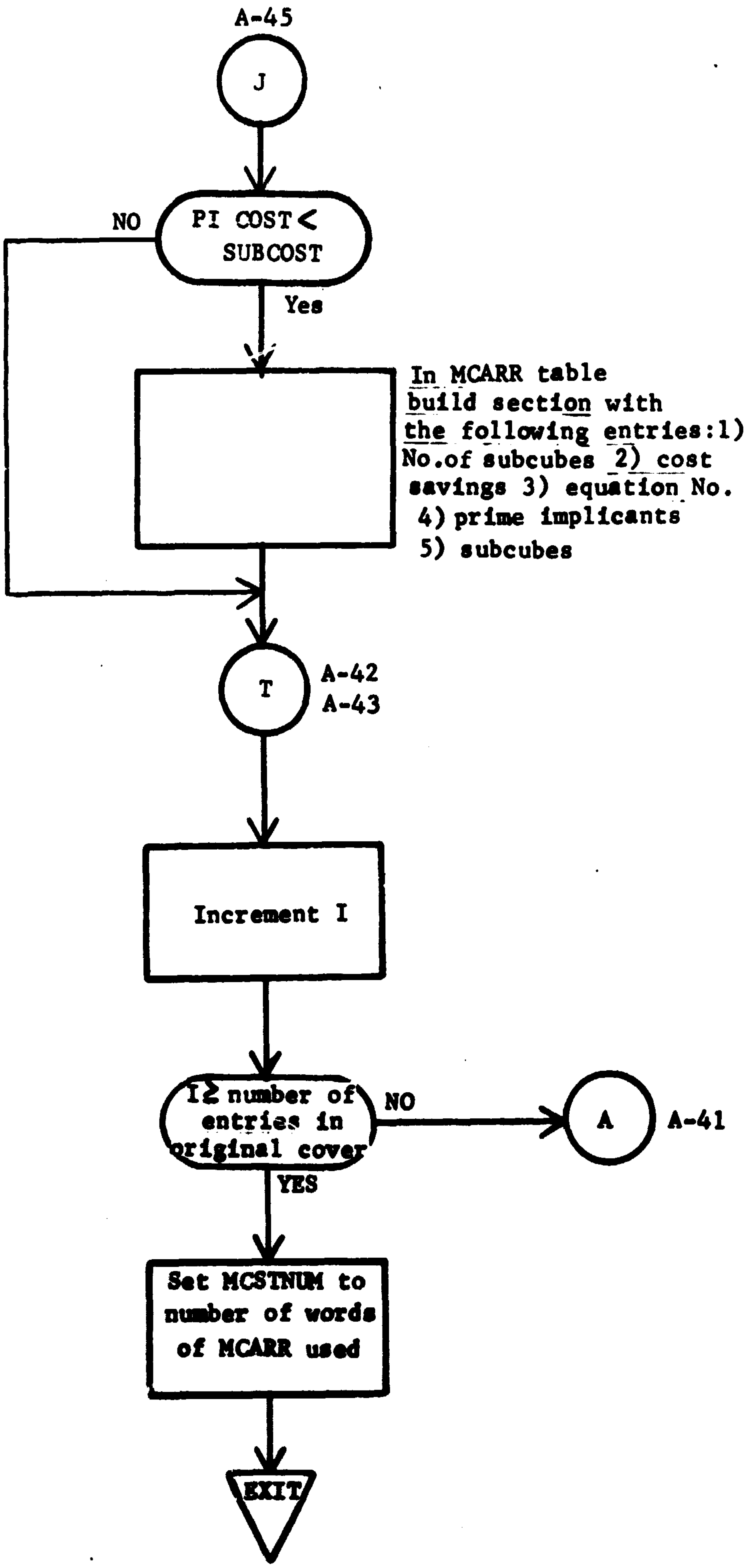


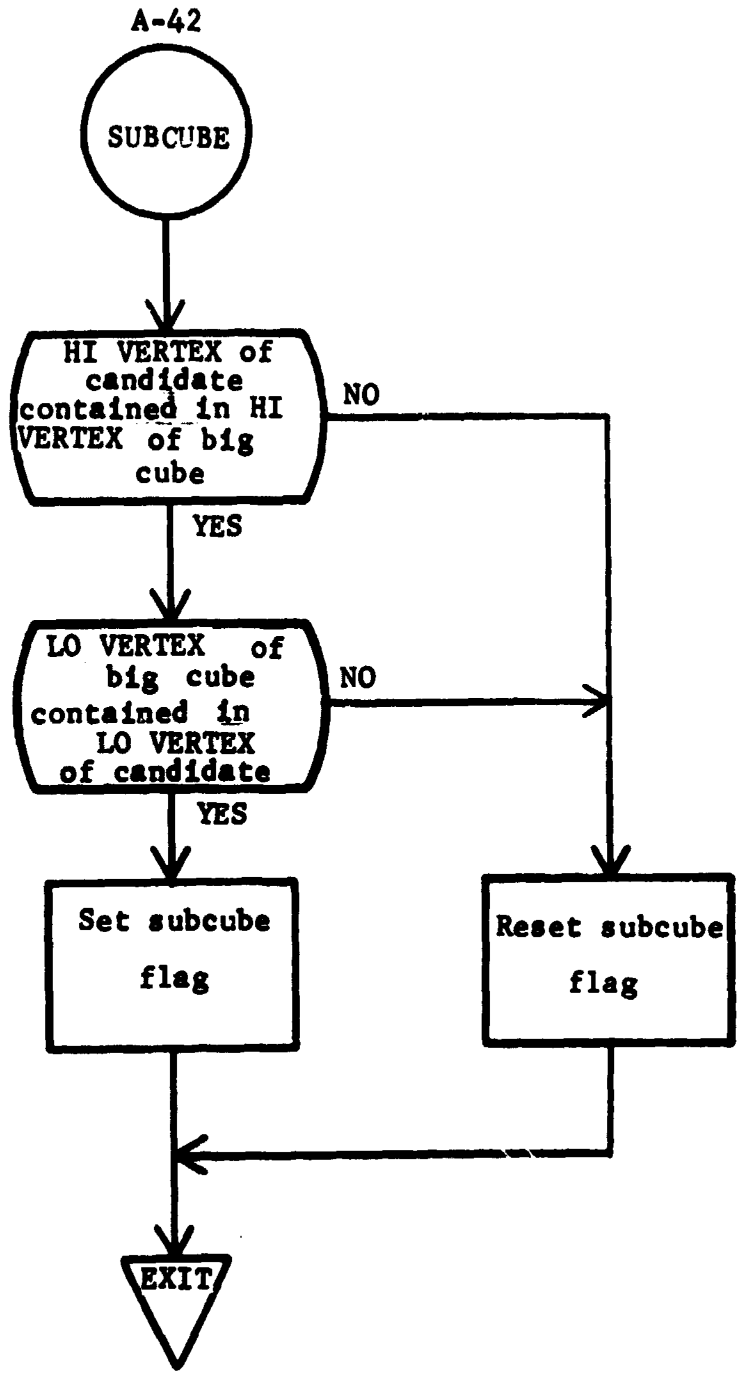

C
A-41
A-47
SUBCUBE
Determine if MPI2(J) is a subcube of E(I)
Is MPI2(J) the same cube as E(I)
YES
T
A-46
NO
is MPI2(J) a subcube of E(I)
NO
Yes
Save MPI2(J) in N BIT(K)
Increment K
Increment J
J ≥ number of terms in equation
No
B
A-41
YES
D
A-43

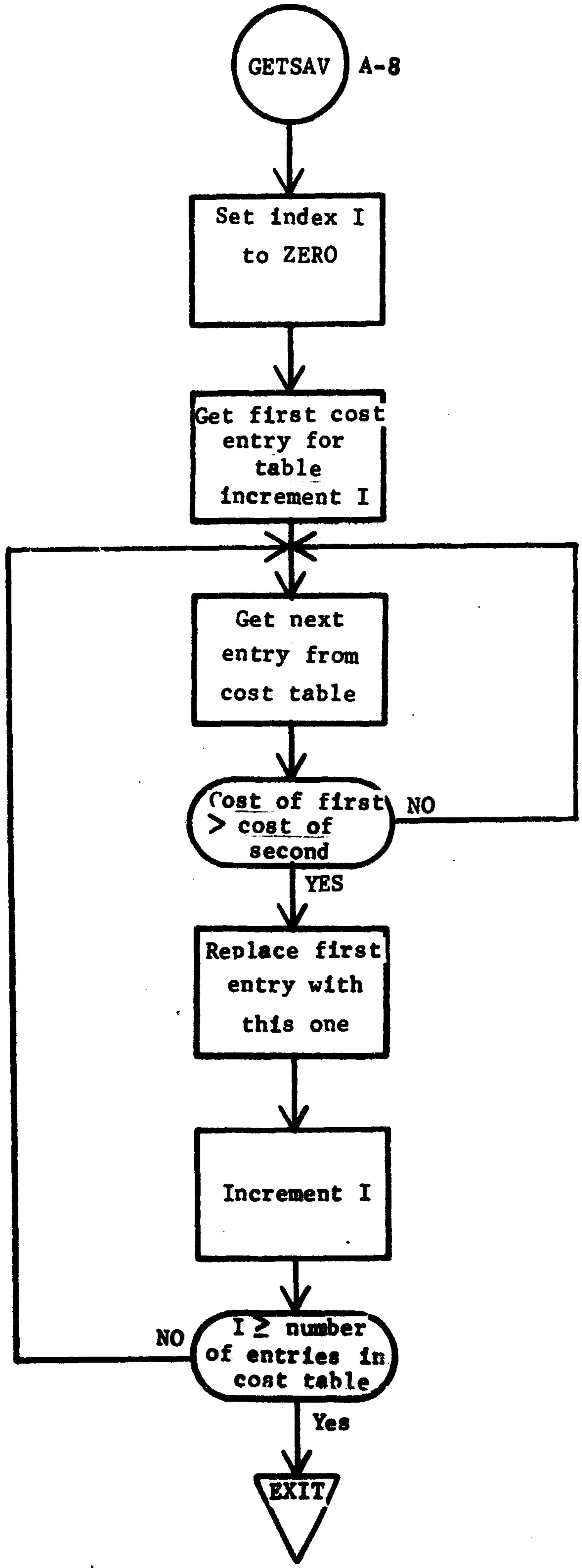

D
A-42
Any subcubes found
NO
T
A-46
YES
Set index L to ZERO
Get MPI1(L)
A-38
HILO
Get HI and LO vertices of MPI1(L)
E(I) and MPI1(L) the same cube
Yes
Set PI COST = 1
F
A-44
NO
Increment L
NO
L ≥ number of terms in other equations
YES
Set Y to the number of ONES in the result of HI VERTEX ⊕ LO VERTEX
EE
A-44

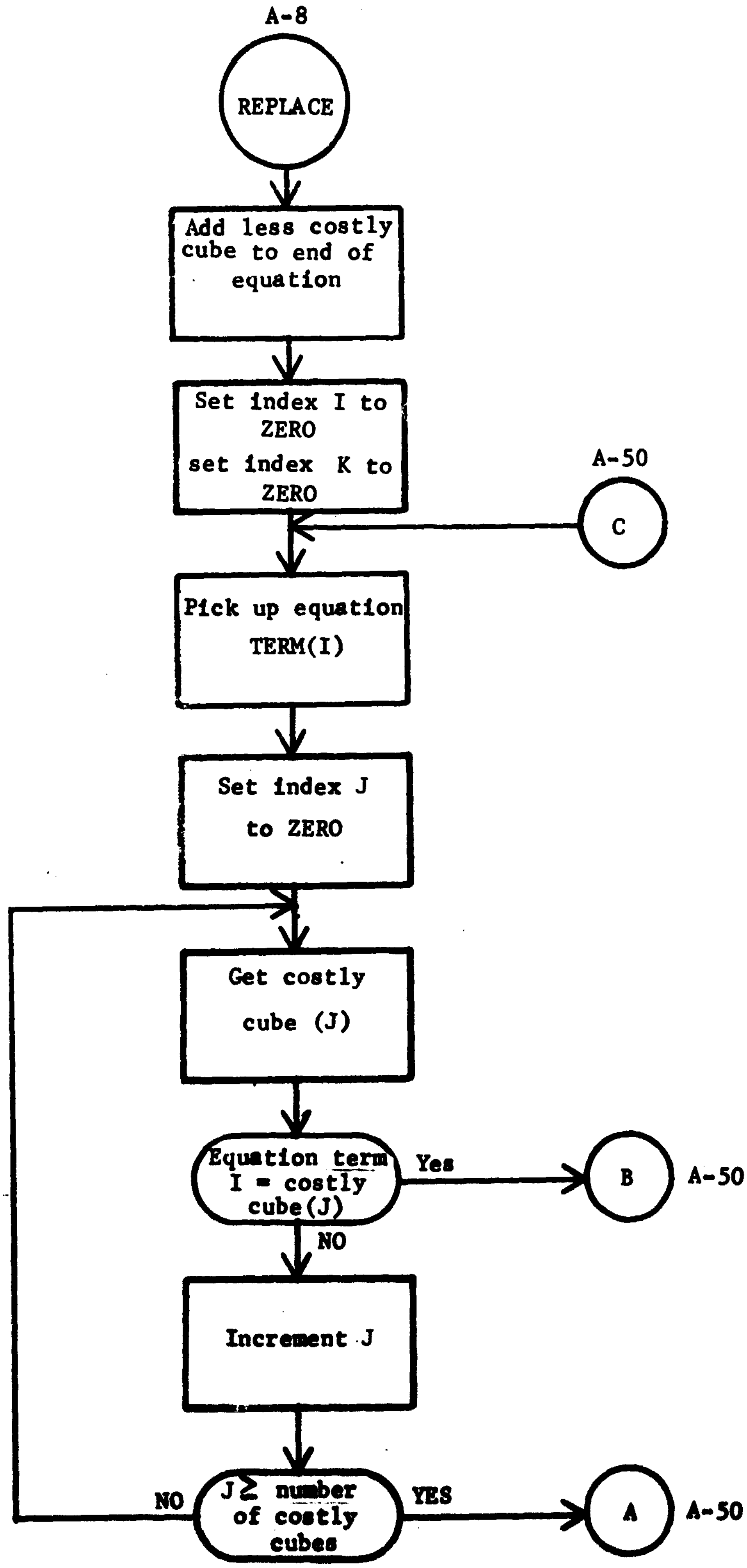

EE
A-43
Set X to the number of ONES in the result of LSIZE-1
Set PI COST = X-Y+1
SUBCOST = 0
Set index L
A-43
A-45
F
Get subcube from NBIT(L)
A-38
HILO
Get HI and LO vertices from NBIT(L)
Set index M
Move HI and LO vertices
A-45
G
Get MPI1(M)
H
A-45

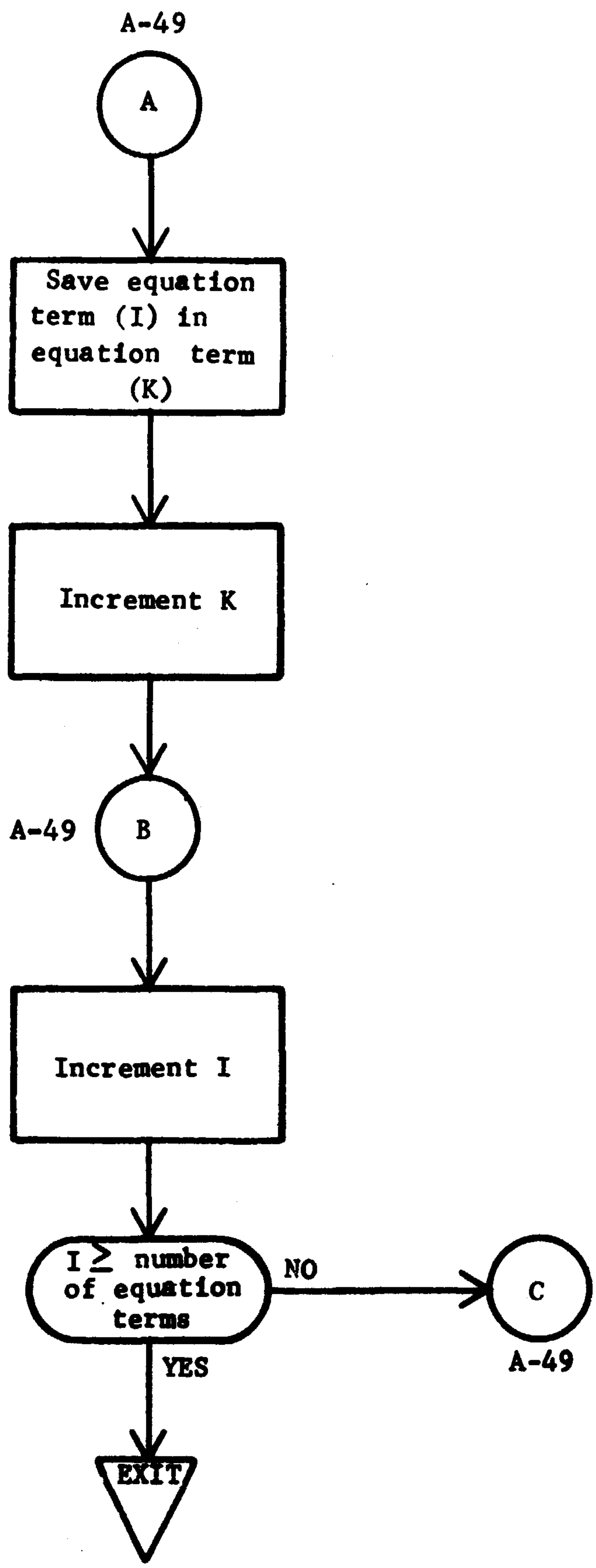

H
A-44
A-38
HILO
Get HI and LO vertices of MPI(M)
Subcube used in other equation
YES
SUBCOST = SUBCOST + 1
NO
Increment M
M ≥ number of terms in other equations
NO
G
A-44
Set Y to the number of ONES in the result of HI VERTEX ⊕ LO VERTEX of subcube in NBIT(L)
SUBCOST = SUBCOST + (X-Y)+1
Increment L
L ≥ K
NO
F
A-44
YES
J
A-46

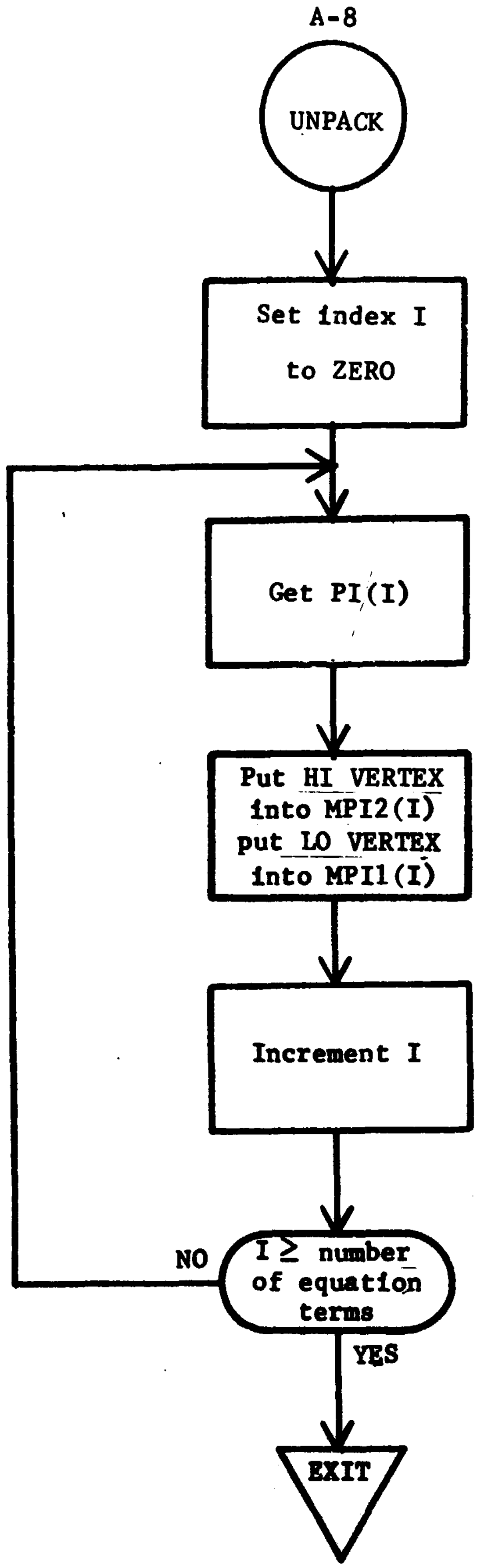

A-45
J
NO
PI COST <
SUBCOST
Yes
In MCARR table
build section with
the following entries:1)
No.of subcubes 2) cost
savings 3) equation No.
4) prime implicants
5) subcubes
T
A-42
A-43
Increment I
I≥ number of
entries in
original cover
NO
A
A-41
YES
Set MCSTNUM to
number of words
of MCARR used
EXIT


A-42
SUBCUBE
HI VERTEX of candidate contained in HI VERTEX of big cube
NO
YES
LO VERTEX of big cube contained in LO VERTEX of candidate
NO
YES
Set subcube flag
Reset subcube flag
EXIT


GETSAV
A-8
Set index I to ZERO
Get first cost entry for table increment I
Get next entry from cost table
Cost of first > cost of second
NO
YES
Replace first entry with this one
Increment I
I ≥ number of entries in cost table
NO
Yes
EXIT


A-8
REPLACE
Add less costly cube to end of equation
Set index I to ZERO
set index K to ZERO
A-50
C
Pick up equation TERM(I)
Set index J to ZERO
Get costly cube (J)
Equation term I = costly cube(J)
Yes
B
A-50
NO
Increment J
J ≥ number of costly cubes
NO
YES
A
A-50


A-49
A
Save equation term (I) in equation term (K)
Increment K
A-49
B
Increment I
I ≥ number of equation terms
NO
C
A-49
YES
EXIT


A-8
UNPACK
Set index I to ZERO
Get PI(I)
Put HI VERTEX into MPI2(I) put LO VERTEX into MPI1(I)
Increment I
I ≥ number of equation terms
NO
YES
EXIT