A SYSTEMS SIMULATOR PROGRAMMING LANGUAGE FOR THE IBM 1620 COMPUTER

by

445

THOMAS ALLEN WEBB III

B. S., Kansas State University, 1965

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1966

Approved by:

L. E. Grosh

Major Professor

TABLE OF CONTENTS

INTRODUCTION

Through simulation the performance of an organization or some part of an organization can be represented over a certain period of time (Martin, 1961). This is accomplished by devising mathematical models representing the components of the organization and providing decision systems for these components to represent the various interrelationships. A digital computer may be used to progress through the model for various intervals of time, pausing at the end of each interval to compute the interactions between certain components. In this way it is possible to obtain a "simulated" history of the organization under certain specified conditions. By changing the conditions and repeating the process it is possible to compare various parameters or decision systems. In this manner it is possible to experiment with changes in an organization without affecting the actual organization being studied.

Random variations are characteristic of many organizations. In many situations, the randomness is so essential that the common simplifying technique of using average values simply "assumes away" the problem. Whenever uncertainty is an essential consideration in an organization, a method must be available to produce a similar uncertainty in the simulation model of the organization.

The use of a computer in such problems depends to a great extent on one's ability to structure a simulation in a machine or systems language. More specifically, it depends a great deal on the conformance of certain programming languages to the needs of simulation problems. For the programmer the most important need is that the language be problem oriented, since an organization will be reduced to a mathematical model.

Inherent in all organizations are many and varied decisions, i. e. rules of action. Decisions may more accurately be described as decision systems since a single decision may be dependent on many factors. Decision systems, due to their complexities, normally occupy the greatest portion of the simulation routing. Thus, the ability of a programming language to implement decision systems is important.

Speed and efficiency are also primary factors. Speed refers to the time required by a compiled decision system to make the correct decision. This time must be minimized without occupying excessive amounts of computer memory.

Efficiency refers to the amount of computer memory required. Since the memory must provide room for the compiled mainline program, and many, sometimes large, areas for working and storage, the amount occupied by any one of these should be minimized. The size of the mainline program and, to a certain extent, the size of the working areas are variable and depend upon the simulation being performed. However, since the subroutines do the bulk of the work, it seems only natural to make them as efficient as possible. This is true especially for the decision process.

Another important and sometimes over-looked factor, is flexibility. The process of trying several different conditions or changes within the organization is inherent to simulation. These changes are typically within the decision process itself. Thus, provisions should be made to make these changes in the form of data on the object level. This should reduce the need to recompile a program each time a change must be made.

Three important factors in the selection of a programming language for use with simulation problems have been discussed. They are speed, efficiency in the size of compiled programs and their subroutines, and flexibility on the object level of the compiled programs. Another requirement in Monte Carlo simulations is the ability to generate and use of random numbers[1] (Hull, 1962).

Chance variation is generally introduced into systems models by probability distributions. The distribution function may be a standard one (i.e. normal, exponential, Poisson, etc.) or an arbitrary observed distribution derived from historical data. The most important of these distributions is the rectangular or uniform distribution because any distribution may be represented by random numbers drawn from a rectangular distribution by use of a probability integral transformation (Freund, 1963).

IBM 1620 Fortran II is not a programming language that satisfies all of the above needs. Thus it is the objective of this thesis to augment the Fortran II programming language so that the needs specified above are satisfied.

---

[1]Random numbers generated on a computer are called "pseudo-random" numbers and only appear to be drawn at random from certain probability distributions. However, they are expected to be non-repeating for a "long" sequence of numbers.

FORTRAN II

Fortran ("formula translation") II is a language that closely resembles algebra (IBM, 1962). It is a programming language designed primarily for scientific and engineering calculations. Since it is problem oriented, it provides engineers with a method of communication that is more familiar, easier to learn, and easier to use than actual machine language. In this light, Fortran II meets one of the aforesaid qualifications for a programming language that is suited for simulation problems.

In addition, Fortran II object programs are generally as efficient as those which might be written by an experienced programmer (IBM, 1962). A bank of efficient subroutines is included in every Fortran program. These subroutines are used throughout the entire program. The mainline program consists of branches to the subroutines. Since the subroutines occupy such an important position and make up the majority of the object program, great pains have been taken to make them as short and as fast as possible.

In addition to speed and brevity, Fortran II is equipped with floating point capabilities. Although this is not necessary, it is convenient and makes programming considerably easier.

Perhaps the greatest disadvantage associated with Fortran II, is the complexity of the decision process for multiple decisions. The basic element of the decision process is the "IF" statement which permits the programmer to change the sequence of statement execution, depending upon the value of an arithmetic expression.

The general form is:

```
IF(a)N1,N2,N3
```

where (a) is an expression, and N1, N2, and N3 are statement numbers. Control is transferred to statement number N1, N2, or N3 depending on whether the value of (a) is less than, equal to, or greater than zero respectively (IBM, 1962). The "IF" statement will test only one decision parameter at a time.

As long as only one parameter need be tested at a time, the "IF" statement is quite sufficient. However, the number of "IF" statements required increases exponentially with the number of parameters in the decision system. As an example, suppose A and B need to be tested independently. A flow diagram of the process is as in figure 1.

This means that there are nine possible configurations of the two variables. In tabular form these are as follows.

A	+	+	+	-	-	-	0	0	0
B	+	-	0	+	-	0	+	-	0

Each possible configuration will require a seperate branch to a seperate or identical position within the program. A program following the logical flow listed above would contain the following "IF" statements:

```
   IF (A) 11,12,13
11 IF (B)  1, 2, 3
12 IF (B)  4, 5, 6
13 IF (B)  7, 8, 9
```

It is possible to generalize the number of possible paths as follows:

$$\text{Number of Paths} = 3^n,$$

where n = the number of decision parameters. The number of "IF" statements necessary to program a decision on (n) parameters would be

$$\sum_{i=1}^{n} 3^{i-1}.$$

This can be seen by summing the following series, representing a decision tree.

$$1 + 3^1 + 3^2 + \ldots\ldots\ldots + 3^{n-1}$$

Thus if twenty parameters are necessary, there would be 3,486,784,401 possible paths and 1,743,392,110 "IF" statements would be necessary to make the decision as to which path should be followed. In addition to this difficulty in implementing complex decisions, the machine language programs resulting from "IF" statements are relatively slow and inefficient.

Another disadvantage to Fortran II is that random numbers are not readily available. Certainly, there are several models by which random number generators may be inserted into Fortran programs, but these are relatively time consuming if they are programmed in the Fortran language. The same models when programmed in SPS (Symbolic Programming System) are much faster and occupy much less space.

The choice of Fortran II for further study and implementation is quite logical since it is "easily implemented to problem solving applications and still compiles relatively efficient machine language programs". It remains, however, to implement a method by which decisions may be made more efficiently, both in terms of the memory occupied and in terms of the time necessary for execution. Implementation of an efficient "random number generator" will also be necessary.

Fortran II is capable of using "functions" or subroutines. Functions are divided into three types:

1. Library functions
    a. Non-relocatable library functions
    b. Relocatable library functions
2. Arithmetic statement functions
3. Fortran subprograms
    a. Subprogram functions
    b. Subprogram subroutines.

The most elementary type of function is the "non-relocatable" library function. These, along with the relocatable library functions, occupy the subroutine deck of the Fortran II compiler. This type of subroutine is used for the input and output functions as well as the basic arithmetic operations (such as add or multiply). They are under the control of the processor only and may not be called directly from a Fortran source program.

Non-relocatable library functions occupy a fixed area in core storage. They will be present in all Fortran programs and in the same location unless the processor is changed. These subroutines may be changed only with direct changes to the processor. It would be better to rewrite the entire processor than to attempt to work at the level of the "non-relocatable" library function. The rewards for direct work on the processor would be about the same for a greater investment in time and effort. Hereafter, when reference is made to library functions, it will be to the exclusion of the non-relocatable variety.

Relocatable library functions will be compiled into the object program only if they are called from within the source program (with the exception of the exponential and logarithm library functions), and they will not occupy a fixed location. Certain functions, for instance the square root function, are not always required and would only occupy space in the computers memory. These are included as relocatable library functions so that core storage is not needlessly taken up.

Library functions are preprogrammed and exist in a prepared deck, referred to as the subroutine deck of Fortran II. Instead of appearing in the object program every time they are called, they appear only once and only if they are called. They may be called by including the name of the function in an arithmetic statement within the source program. The name is followed by a single argument enclosed in parentheses which may be a constant, a variable (subscripted or not subscripted), or an expression.

An example of a statement which will call two of the library functions, Viz, SINF and SQRTF, is:

```
Y = A-SINF(B*SQRTF(C)).
```

In this case, the assembled instructions of the object program will:

1. Branch to the square root subroutine to compute the value

of the square root of C.

2. Multiply the value of the square root of C by B.
3. Branch to the SINF subroutine to compute the value of the sine of the product.
4. Subtract this value from A.
5. Replace the present value of Y with the value obtained in step 4.

Only one value is produced by a library function and only one value may be used as an argument. Fortran II has the capability of using up to fifty library functions. The compiler furnished by IBM has seven. The process by which a user may add library functions will be discussed later.

An arithmetic statement function is defined by one arithmetic statement and applies only to the program in which it appears. The name of the function determines the mode of the value that is computed. The mode must be either fixed point or floating point as determined by the first letter in the name of the function.

The arithmetic statement function is defined as follows:

```
NAME (ARG) = EXPRESSION
```

where NAME is the name of the function, ARG is the argument and consists of one or more variable names of non-subscripted variables; and EXPRESSION is an arithmetic expression that must conform to the rules for forming expressions. Examples of arithmetic statement functions are as follows:

```
FIRST(X) = A * X + B
```

```
SECOND(X,B) = COS(X) * FIRST(B)
Y = SECOND(C,D).
```

This sequence of functions could be replaced (losing a great deal of versatility but saving much space and computing time) by the following statement.

```
Y = COS(C) * (A * D + B)
```

The appearance of the name of an arithmetic statement function in an arithmetic statement serves to call the function.

As many of the variables appearing within the expression of a function as desired may be stated on the left hand side of the arithmetic statement function as arguments. The arguments are really only dummy variables so their names are unimportant except to specify the mode. They may even be the same as names appearing later in the program.

Those variables which are not stated as arguments are treated as parameters. Thus, if FIRST is defined in a function statement as

```
FIRST(X) = A * X + B,
```

then a later reference to FIRST(Y) will cause

```
A * Y + B,
```

based on the current values of A, B, and Y, to be computed.

The arguments of an arithmetic statement function may be expressions and may involve subscripted variables. Thus, a reference to

```
FIRST(Z + Y(I)),
```

as a result of the previous definition of FIRST, will cause

A * (Z + Y(I)) + B,

to be computed on the basis of the current values of A, B, Y(I), and Z.

Functions defined by arithmetic statements are always compiled as closed subroutines. A closed subroutine is used only by calling for it in the source program and will be used only in the program in which it was compiled. This means that the machine language instructions are compiled only once in the object program. Calling is accomplished either with the use of a "CALL" statement or by mentioning the name of a subroutine in the source program.

Fortran subprograms consist of subroutines that are not used frequently enough to be library functions, yet due to their size or complexity, they cannot be defined in a single arithmetic statement.

Fortran subprograms are compiled seperately but the object program cannot be executed by itself. For execution they must be called and loaded by a main program. For this reason they are called subprograms. Since each subprogram is compiled seperately, the arithmetic function and variable names used in the main program are completely independent of the function and variable names used in the subprogram. This means that very general subprograms can be created and used with any main Fortran II program.

Subprograms may be divided into two types; function subprograms and subroutine subprograms. Four statements are necessary for their definition and use. They are:

FUNCTION,

SUBROUTINE,

CALL, and

RETURN,

and will be described later.

Although function and subroutine subprograms are similar and are treated together in this thesis, they differ in two significant respects:

1. Function subprograms can compute only one value, whereas subroutine subprograms can compute many values and return them to the main program.
2. Function subprograms may be called by an expression containing its name but subroutine subprograms are called by the use of a CALL statement. This means that function subprograms may be used in arithmetic statements.

The FUNCTION statement is always the first in a function subprogram and defines it as such. The general form of a FUNCTION statement is as follows:

FUNCTION "Name" (a1, a2, ... , an).

"Name" is the symbolic name of a single-valued function, and each argument (a1, a2, ... , an) is a non-subscripted variable name, i.e. must not be a member of a dimensioned array.

In a function subprogram, the name of the function must appear in an input list or on the left-hand side of an arithmetic statement. An example indicating the use of a FUNCTION statement and a RETURN statement and which conforms to this rule is as follows:

```
FUNCTION SUM (A,B)
SUM = A + B
RETURN.
```

The RETURN statement terminated the function subprogram and returns the value of the function to the ealling program. Any number of these statements may be used but control must end with one of them.

The arguments which follow the function referenee in the calling program must agree in number, mode, and order with the arguments listed in the FUNCTION statement in the funetion subprogram. When the argument in the reference statement is an array name the eorresponding argument in the FUNCTION statement must also be an array name and both must be dimensioned within their respective programs. None of the dummy variables may appear in EQUIVALENCE statements in the function subprogram.

The SUBROUTINE statement is used in a subroutine subprogram in the same manner a FUNCTION statement is used in a function subprogram. The general form of the statement is as follows:

SUBROUTINE "Name" (a1, a2, ... , an).

"Name" is the symbolic name of a subprogram, and each argument, if any is specified, is a non-subseripted variable name. The SUBROUTINE statement defines the subroutine subprogram and must be the first statement in the subprogram. The subprogram must be a Fortran program and may use any Fortran II statement exeept a FUNCTION statement or another subroutine statement.

A ealling program may refer to a subroutine subprogram by a

CALL statement which specifies the name of the subprogram and its arguments. The subroutine subprogram uses one or more of its arguments to return results. Therefore, the arguments that are used for this purpose must appear on the left-hand side of an arithmetic statement in the subprogram or in an input list within the subprogram.

The correspondence between arguments applys in the case of the subroutine subprogram just as it did for the function subprogram. That is, the arguments listed in the CALL statement must correspond in number, order, mode, and dimension.

The CALL statement refers only to the subroutine subprogram whereas the RETURN statement is used by both the function and subroutine subprograms. The general form of the CALL statement is as follows:

CALL "Name" (a1, a2, ... , an) .

Name is the name of the subroutine subprogram being called and (a1, a2, ... , an) are the arguments. Each argument may be a fixed or floating point constant or variable (with or without subscripts), or an expression.

A priority list of the functions in order of their power and flexibility would be as follows:

1. Non-relocatable Functions
2. Library Functions
3. Subroutine Subprograms
4. Function Subprograms
5. Arithmetic Statement Functions.

This list would have the same priority of difficulty of programming, the first being difficult and the last being normal Fortran programming.

A symbol table is a straightforward list of all encountered symbols such as subroutine names, variable names, and statement numbers (Leeson, 1962). A symbol table look-up operation occurs each time a constant, variable, or a statement number is encountered. If the symbol is in the table its object time address is found in the corresponding "Table of Addresses of Encountered Symbols" and is used in the generated object program. If the symbol is not in the table it is placed there and its object time address is determined and stored in the corresponding address table. The address is then used as above.

A brute force comparison is used to determine if a symbol is in the symbol table until a successful comparison is made, the temporary end of the symbol table is found, or the symbol table is found to be full. This way, the mere mention of a symbol defines it.

The first symbols in the symbol table are the names and alternate names of the library functions.

With the use of the symbol table and the table of encountered addresses, Fortran II generates a series of branch and transmit instructions, of the regular, immediate, and floating varieties, for the mainline program. The P address depends upon the operation desired and directs control to a subroutine. The Q address depends upon which symbol wants action at that time. Both will come from the table of addresses with the exception of operations requiring the use of non-relocatable subroutines. These have a fixed location and their addresses may be found more directly. The symbol table and the table of addresses are both compiling aids and therefore are not used in an object program.

If the basic processor is changed we lose the ability of incorporating revisions furnished by IBM. If we work at the library function level or higher with very little additional work our developed subroutines may be used by a different processor (on a machine other than the 1620). For this reason, alterations and additions will be made in the form of library functions.

DECISION SYSTEMS

A decision system is the algorithm for relating interacting variables in a problem solution. Decision systems can be simplified and put in a logical flow. For this discussion, then, a decision system will be described as a logical and orderly method by which decisions may be made.

The inefficiencies of a decision tree have already been demonstrated in the discussion of the "IF" statement. For simplification, this discussion will be reduced to one of "limited entry", or binary decisions. Even when using "IF" statements in Fortran, the two-way decision is often adequate.

The decision tree, for limited entry decisions, is shown in figure 2, where Y and N stand for Yes and no respectively. The use of two decision parameters yields four paths and requires three "IF" statements to accurately describe the system. Thus 20 parameters yield 1,048,576 possible paths and requires 1,048,575 "IF" statements to describe the system.

An example of a decision system is the problem of credit approval (Kirk, 1965). In this case three parameters are chosen on which to base a decision. They are:

1. Is the credit limit OK?
2. Is the pay experience favorable?
3. Has special clearance been obtained?

The decision tree to describe these parameters is shown in figure 2. The three parameters yield eight possible branches and require seven comparisons to describe the system.

Structure tables are an advantageous method for unambiguously describing complex, multi-variable, multi-rule decision systems (Schmidt, 1964). Each table is a precise statement of the logical and quantitative relationships supporting a particular elementary process. They are developed in terms of the criteria or parameters affecting the problem and the various outcomes which may result. An example of a decision table incorporating the above parameters is described in figure 3.

This table may be shortened by introducing another symbol other than the Y or N. This symbol is "-" and means "not pertinent". Whenever this symbol is used, the parameter in question (the row) has no significance in that rule (the column). Using this new symbol the decision table in figure 3 is reduced to figure 4. Using this new decision table we may shorten the tree in figure 2 to that shown in figure 5. This shortens the problem to one of four paths and three "IF" statements by eliminating redundancies.

Another economy in the decision table is gained by introducing the concept of "ELSE". This concept merely states: if no rule in a decision table can be matched then the action specified by "ELSE" will be executed. Using this idea we may reduce the decision table above to two rules, one a formal rule and the second an implied rule (figure 6).

The conversion of a decision table into a decision tree is not always as easy as above. The usual case has been to construct the general tree with all of its redundancies and inefficiencies. Several algorithms by which a fairly consistant and efficient tree may be constructed have been devised. One such algorithm will be presented here. This algorithm minimizes computer storage space required for the resultant program. It is also designed to pin-point any contradictions or redundacies among the rules in a table (Pollack, 1965).

Figure 7 presents an example of a limited-entry decision table that could easily present redundancies and inefficiencies in the normal procedure of constructing a tree. The A_1 are the actions, the C_1 are the conditions or parameters, and the R_1 are the rules. Before converting the decision table to individual comparisons and the series of branches associated with each path, the number of written decision rules should be reduced to a minimum to simplify the use of the algorithm. As an example, figure 7 may be reduced to the table described by figure 8.

It would be helpful to present a general description of how to convert the decision table to tree form before the algorithm due to Pollack is presented. The procedure is as follows.

One row of the original decision table is selected. The criterion for selection is given in the algorithm. The condition in that row becomes the first comparison of the flowchart. The original decision table is then decomposed into two subtables (containing one less row), or a subtable and a rule, or only two rules; and each of these is associated with each branch of the comparison.

This is continued with each subtable until the branches lead only to completed rules or "ELSE", or until a subtable indicates that the original table contained redundant or contradictory rules.

The algorithm is presented without proof, however when applied, it has proven to be effective (Pollack, 1965). The objective of the algorithm is to convert a decision table to a computer program and have this program use the minimum number of storage locations, and still be relatively fast.

Step one of the algorithm is a check for redundancy or contradiction. If at any stage, a pair of rules does not contain at least one (Y, N) pair in any of its row, redundancy or contradiction exists. Such is the case for rules 1 and 2 in figure 9. If the actions for rules 1 and 2 are identical a redundancy exists and rule 2 may be eliminated. Figure 10 presents an example of this. Where the actions for rule 1 differ from those of rules 2 and 3, a contradiction exists. Where the actions of rule 1 are identical with those of rules 2 and 3, a redundancy exists.

Step two of the algorithm is to make a dash count and determine those rows which have a minimum dash count. A dash that appears in a rule (column) that contains r dashes is counted as 2^r dashes. For each rule, 2^r is denoted as the column count. In each row, the sum of the column counts corresponding to the dashes in that row is called the dash count. A row dash count is the sum of the column counts of those rules that have dash entries in the row. This is illustrated in figure 11 in which row 2 has the minimum dash count.

Step three is used in the event that more than one row has a minimum dash count. It is to select that row which has a maximum delta, which is the absolute value of the difference of the Y-count and the N-count. The Y-count is the sum of the column-counts corresponding to the Y's in the row. The N-count is similar. Figure 12 is an example of a decision table with a minimum dash count in all of the rows. Since C_1 had the maximum delta, it was selected. The selected row is called k. In figure 12 $C_1=C_k$.

Step four is to discriminate on the condition in row k. This discrimination has two branches, each of which leads to a subtable which contains one or more rules, with one row less than the original table (row k is deleted). The Y-branch had a Y in row k. The N-branch works similarly. In addition both subtables contain those rules that had a dash in row k of the previous table.

Step five is to go back to step one if the subtable of interest contains more than one rule.

Step six may be divided into four smaller steps each of which handles a different situation. They are as follows:

a. If a branch leads to a subtable containing one rule, and if that rule contains all dashes, then replace the subtable with the rule itself.

b. If a branch leads to a subtable with one rule, and that rule does not have all dashes but has more than one Y, N, or a combination of Y or N, choose as row k any row that has no dash. The selected branch will indicate a subtable with one less row in it. The opposing branch will indicate "ELSE".

c. If a branch does not lead to a subtable, it leads to "ELSE".

d. If a branch leads to a subtable containing only one rule, and that rule contains only one condition whose value is Y or N, then one branch of the discrimination on that condition leads to the rule, the other branch leads to "ELSE".

As an example of the use of this algorithm, refer to the table in figure 8. The application of the algorithm yields figure 13.

Using this algorithm one may reduce a decision table to the minimum tree and program it in Fortran. This is then a valuable programming aid. This does not, however, allow a programmer to program decision tables into his Fortran program. This means, also, that the programmer must recalculate the entire decision tree, reprogram it, and reprocess the program in order to make any change in the decision process. A possible program might perform the process of reducing the table to a tree, and might even generate the completed "IF" statements on cards. The compiling process would remain, however. It would be more desirable to change the decision table at object level.

It is possible to program decision tables into Fortran with the use of the computed "GO TO" statement and a simple arithmetic statement (Veinott, 1966). The computed "GO TO" statement indicates the statement that is to be executed next. However, the statement number that the program control is transferred to can be altered during the program. The general form of the computed "GO TO" statement is

```
GO TO (N1,N2,...,Nm),I
```

where N1, N2,..., Nm are statement numbers and I is a non-subscripted fixed point variable. This statement causes transfer of control to the first, second, third, etc., statement in the list depending on whether the value of I is 1, 2, 3, etc.

The approach taken considers a decision table as a multiple branch within a program. The procedure is to calculate a unique number for each possible set of conditions. The unique numbers must be an unbroken series of consecutive numbers in order to be used as a branching variable. Figure 4 will serve as an example. A new table is constructed, adding a "value" column. Since there are $2^3=8$ possible combinations of three conditions, 8 columns will be provided, one for each possible combination. The 8 columns are numbered from 0 to 7 inclusive. X's are placed in the columns so that the corresponding "values" add to the number heading the column. This is represented in figure 14. This provides a unique number for each possible combination with a consecutive order. Since the series contains a zero it is necessary to add one to make it a branching variable. Ordinarily yes is denoted by 1 and no is denoted by 0. This convention is carried over in this case. In the above table the following notation is used:

I1=1,	Credit Limit OK
=0,	Otherwise
I2=1,	Pay Experience OK
=0,	Otherwise
I3=1,	Special Clearance
=0,	Otherwise

N1= Statement number to initiate the action "do not approve order"

N2= Statement number to initiate the action "approve order"

The Fortran program for this table is as follows:

```
JUMP=1+I1+2*I2+4*I3
GO TO (N1,N2,N2,N2,N2,N2,N2,N2),JUMP
```

It may be desirable to represent one or more conditions by more than two states. Again, let there be m conditions, the states of each of which is indicated by the value of variables I1,I2,... ,Im. Let the various conditions have K1,K2,...,Km mutually exclusive states. That is, the conditions themselves are represented by the "I" variables: each of these "I" variables can take on different values, starting with zero, to express the state of this particular condition. The number of states of any condition.

Since the states, for any condition, are mutually exclusive, by definition, only one state can exist at a time for any given condition. The number of combinations or "rules" that exist will be:

Number of rules = (K1) (K2)...(Km)=R. (Veinott, 1966)

For convenience, let KNL equal the number of states of the next-to-the-last condition.

The procedure in programming such a table is to set up R rules and identify each combination. To each combination a statement number must be assigned. The Fortran program would be:

```
JUMP=1+I1+K1*I2+K1*K2*I3+....+(K1*K2...*KNL)*Im
GO TO (N1,N2,.....,Nr),JUMP
```

For an example, see figure 15.

This technique requires the exclusion of the "-" or "not pertinent" element as well as the use of "ELSE". This means that a table must be expanded to cover all possible combinations of conditions. For the limited entry decision table 2^n rules are required for n decision parameters. Finally, changes in the decision table cannot be made on the object level. The user must recompile his program to make any such changes.

A more efficient and faster method is desirable, as well as one which will allow changes on the object level. An algorithm which should best fit these requirements was presented by Kirk in January, 1965. It is as follows.

The first step is to prepare a binary image of the condition portion of the decision table. This image, called the "table vector" and consisting of "rule vectors" is shown in figure 16 for the credit-approval table in figure 3.

The table matrix and the data vector alone do not have sufficient information for solving the problem because of the use of a zero for nonpertinent conditions in the table matrix. A masking matrix is used to screen out nonpertinent conditions from the data vector prior to scanning the table matrix. The masking matrix is produced by replacing a Y and an N with a 1, and "-" with 0. Therefore, a logical multiplication will insure a zero in the element of the data vector corresponding with a nonpertinent element in the original table. The masking matrix for the credit approval table is as in figure 17.

A table of logical multiplication, shown in figure 18 is used to multiply the data vector, element by element, by the appropriate masking vector prior to scanning. The result is then compared to the corresponding table vector. If a match is found the resulting action is executed; if not the next masking vector and table vector are tried until the table is exhausted. If the table is exhausted then "ELSE" is executed.

Since the IBM 1620 computer is not a binary computer, the use of characters is necessary. The logic used in the programming of this algorithm is presented in a subsequent section of this thesis.

RANDOM NUMBERS

A Monte Carlo calculation makes systematic use of random numbers. The simulation is provided with internally generated data that has the same characteristics as the actual data (Bowman and Fetter, 1961). A schematic diagram of a process might consist of a jagged, irregular figure representing the collection of facts as observed. An idealized model, represented by a smooth curve may be derived from the schematic diagram of observations. This model, used with a random number generator, produces an artificially irregular figure, representing the simulated experience. The simulated data is used in mathematical models for further simulations. In addition to providing more extensive data, you can generate your data according to any known rule or give it any special characteristics that you wish in order to study the effects on your simulation model.

The scheme choosen for generating random numbers has the following features: (a) the numbers generated are uniformly distributed between 0 and 1; (b) there is no serial correlation between successive numbers in the sequence; (c) about 50 million numbers can be generated before the sequence repeats itself; (d) the calculation is easy to perform on an electronic computer (Hull, 1962).

The scheme is this: Start with any ten-digit number of the form xyz0000001; call it R_0. Thereafter

$$R_n = K \cdot R_{n-1} \pmod{10^{10}}$$

where K is a fixed multiplier, which should be a ten digit odd power of a prime that is relatively prime to 10. Possible values for K include (Hull, 1962),

$$3^{19} = 1,162,261,467$$
$$7^{11} = 1,977,326,743$$
$$11^{9} = 2,357,947,691.$$

The recursion relation expressed above in symbols can be translated: To get the next number in the sequence, multiply the previous number by K. The result will be a 20 digit number. Take the last (right-hand) ten digits of the product as the next number in the sequence. Treat the number as a ten-digit decimal.

This scheme will generate rectangular or uniform random numbers which in turn may be used to sample from any distribution desired. It was coded in the Symbolic Programming System (SPS) language for the 1620 and compiled as a library function for Fortran II (see page).

Non-uniform random numbers are occasionally needed in order to more properly describe a particular process. For one dimensional distributions we need only to solve the equation $x=F(y)$ for y, where x is uniformly distributed, and where F(y) is the required (cumulative) distribution function. For example, if y is to be exponentially distributed, the following distribution function is used:

$$x = 1 - e^{y} \quad \text{and}$$
$$y = \ln(1 - x).$$

An arithmetic statement function to generate random numbers representing an exponential distribution could be as follows.

EXPRN(X) = LOGF(1. - RNDM(X))

The use of these functions has been previously described.

Many simulations require the use of arbitrary distributions, which may be used with a table look-up operation. For a detailed study on the use of arbitrary distributions, see Starr and Miller (12).

Since its developement, the random number generator has been tested by Wichlan (14). The numbers were found to be rectangular on the five per-cent level. The Chi-squared and Serial Correlation tests were used.

IMPLEMENTATION OF THE MODIFIED PROCESSOR

The Decision Table subroutine has been coded into a relocatable library subroutine. There is present a need for the capability to set up data vectors, i.e. present the conditions as they actually exist in the form of a vector. This capability must be in the form of relocatable subroutines to be compatible with the Decision Table subroutine. A one parameter data vector, with the help of the proper subroutine, could be set up with the following Fortran statement.

```
DATA = EQ(T-S)
```

If T and S are equal, a 1 will become the entire data vector; if not a 0 will be produced. The functions available for constructing the data vector are:

EQ	Equal
UN	Not equal
GR	Greater than
LR	Less than
GE	Greater than or equal
LE	Less than or equal
NC	No change (for data vectors that have already been produced)
YES	Always generates a 1
NO	Always generates a 0
RDATA	The entire data vector will be read in.

An example of the use of these functions is illustrated by the following Fortran statement.

```
DATA=EQ(T-S)+GR(U-T+LRU-S)+GE(T-U)+LE(U-T)
```

if T=1, S=2, and U=3 the following data vector will be produced.

```
011000
```

The data vector may then be tested after a table has been read into core. This is accomplished by the following statement.

```
ACTION=RDTB(1)
```

This will cause a decision table to be read into the proper place in core (common). It will also be named table number 1. Thereafter a table may be referred to by its number in testing as follows:

```
JUMP=TEST(1).
```

The last data vector produced will then be tested by decision table number 1. The resulting value will be assigned to the branching variable, JUMP. JUMP is then used in a computed GO TO statement.

The credit approval table will serve as an example (figure 4). The following variables will be used in the example:

JUMP =	the branching variable
ORDLMT =	the customers credit limit
CUSDBT =	the amount owed by the customer
ORDAMT =	the dollar amount of the order
X =	a dummy variable

The Fortran program is as follows:

```
DATA=GE(CRDLMT-(CUSDBT+ORDAMT))+YES(X)+NO(X)
ACTION=RDTB(1)
JUMP=TEST(1)
GO TO (1,1,1,2),JUMP
```

The DATA statement may vary and options may be placed under switch control. When DATA is executed a data vector is produced and stored in core according to the function names and the values of their arguments. When the function RDTB(1) is executed a table will be read into common and labeled with a 1. In this case the table will look like the following:

```
0403
1--‡
01-‡
001‡
000‡
```

where 0403 means that there are four rules and three conditions in the table. For each table the limit on conditions is 79 and the limit on rules is 99. The rules may be presented in any order but the desired actions must be expressed in the same order in the computed GO TO statement used in the source program. ACTION and DATA are dummy variables but must be floating point variables. The data vector must fit the table it is intended for. The decision table and data vector must be based on the same number of conditions or parameters. In addition, the data vector must be used before another data vector is produced. However, a decision system may contain more than one decision table. Tables may also vary in length.

When the arithmetic statement defining JUMP is encountered a value will be assigned to the variable, JUMP. Assuming this particular customer has exceeded his credit limit, the value of the data vector is as follows.

010

The resulting value of JUMP will be 2 and statement number 1 will be the next statement executed within the program.

If a data vector is to be read in, the entire data vector must be ready and must be right justified on the data card with a flag over the left-most element on the card as follows.

columns 78-80　　　010

The data vector should have the same number of conditions as does the table used for testing.

The purpose of the "data arranging" subroutines is to set up the data vectors in a place in core which is accessable to all of the relocatable subroutines. In this way any subroutine can make changes to or use the data vector as it must. The purpose of the "table reading and testing" subroutine is to read a table into core, label it and set up the table and masking matrices. At the time of testing the logical multiplication is performed on the data vector to mask it and the result is compared to the table vector. Each time a comparison is made a counter is incremented by one. When a comparison is successfully made this counter becomes the value of the branching variable. The subroutines and their flow-charts are presented in the appendix.

In order to implement relocatable library functions, they must first be coded into a 1620 SPS source program. The origin is assigned at location 10,000. All P and Q addresses that are relative to the origin must be labeled as such by placing a flag in the O_1 and O_2 positions of that same instruction, respectively.

The resulting condensed object, with the first 2 and last 7 cards removed, must be preceded by a header card punched with the following information:

Columns 1- 5	XXXXX	Total number of storage locations required by the subroutine. This number must be even.
Columns 11-12	YY	The alternate subroutine number if any.
Columns 16-20	WWWWW	The alternate entry point if any.
Column 63		A record mark.
Columns 75-80	XX0001	Card sequence number, where XX is the subroutine number.

The object deck, minus the header card, must be renumbered starting with XX0002, where XX is the subroutine number.

If a subroutine contains two entry points the card sequence numbering does not change and is done as if no alternate entry were in effect. The information given on the header is sufficient for the use of the alternate entry point. The packet of cards can be inserted between any two subroutines in the library subroutine deck.

It is also necessary to increase the number in columns 1-2 of card 03000 of the Pass I deck to reflect a new total number of relocatable subroutines. Then add the subroutine name(s) at the end of the last card. If a new card is required, it must be numbered consecutively.

Linkage into the subroutine is provided with one of the "Branch and Transmit" instructions. For instance:

```
BTM   SUBR,   A
```

will branch to SUBR and carry A with it. It will place the field represented by A into core just before the entry point of SUBR. When the subroutine is finished a "Branch Back" will be encountered and control will be transferred back to the mainline program.

TRIAL PROGRAMS

In order to test the effectiveness of the decision processes and illustrate the use of the random number generator, a system was chosen for simulation. This system entails a pump and a tank. The pump is used to fill the tank and may have various capacities. The capacity of the pump is stated in terms of the depth of liquid replaced in the tank per pass through the simulation. Thus, a continuous process is simplified into a discrete process. The pump will also have a tendency to heat during the process of pumping. It is assumed that the process of pumping will increase the temperature of the pump by one degree during one pass through the simulation. Initially the pump is 70 degrees and the upper safety limit is 180 degrees. It is assumed that the atmosphere is 70 degrees.

The loss of heat from the pump depends upon the temperature of the pump. The computer program removes heat from the pump by finding the temperature gradient between the pump and the atmosphere and subtracting one hundreth of it from the pump temperature during each pass through the simulation when the pump is inactive.

The tank has a valve which may be used to remove liquid from the tank at the rate of "B" feet per pass through the simulation. The control of this valve is placed under a user of users. The program places the control of this valve under a switch on the 1620 console or will simulate the use of the valve with the random number generator at the discretion of the operator. It is considered desirable to keep the level of the tank between four and five feet.

In addition, two alarms are provided. The first is a heat alarm and will be turned on when the temperature of the pump exceeds 180 degrees. The second is a "tank level high" alarm and will be turned on when the liquid level in the tank exceeds five feet. The decision table used to govern the pump and the alarms is depicted by figure 19.

This entire process has been simulated in four different ways each of which differs in the decision process. The general flow diagram is as presented in figure 20. They are as follows:

1. Using a tree constructed by eliminating the table (figure 19) one condition at a time in the order of their occurrence.
2. Using a tree constructed by eliminating the table according to the "dash count" of the conditions (thus constructing an optimal tree).
3. Programming the table in the Fortran language.
4. Using the table on the object level, thus using the aforementioned library subroutines.

The tree which resulted from reducing the table one condition at a time is illustrated in figure 21. This reduction was accomplished by first discriminating on condition number one (tank in use) and progressing directly through the rest of the conditions until the conversion was complete. The resulting program is illustrated by figure 22.

The tree which resulted from the reduction of the table according to the dash count is as described by figure 23. According to the proponents of the algorithm used, this is the smallest tree that can be produced. The resulting program is illustrated by figure 24.

Programming the decision table in the Fortran language is accomplished by expanding the decision table into the form indicated in figure 25. This is then programmed with the use of a single arithmetic statement and a computed "GO TO" statement. The resulting program is illustrated by figure 26.

The use of the decision table on the object level calls for a reduction for the decision table as depicted in figure 27. The resulting program is illustrated by fiugre 28.

The resulting programs are compared in figure 29. This comparison is made to illustrate the difference in programming, time and core storage necessary to perform the simulation with 1000 passes. Although the tank use was controlled by the random number generator, the same tank usage was experienced since the same argument controlled the random number gnerator in all of the programs in the comparison.

The fastest programs were the programs which used the decision trees. Both had times of 1.901 minutes. The slowest program was the program using the SPS subroutines with a time of 5.222 minutes. The shortest mainline program is the program using the SPS subroutines. However, when the subroutines are taken into consideration this program becomes the longest. The shortest program (including the required subroutines) is the decision tree produced by Pollack's algorithm. However, the subroutines are of a constant length and the mainline programs are of a variable length. For programs requiring more complicated decision systems the program produced with Pollack's algorithm would soon prove to be shorter. There would be a break-even point where the length of the improved decision tree program would be the same as the length of a program utilizing the decision table subroutines. Any larger systems would yield a shorter program with these subroutines.

CONCLUSION

The comparison of the different trial programs yields the following information.

1) Small to medium decision systems yield shorter programs when reduced to a tree with Pollack's algorithm[2].
2) Larger decision systems would probably yield the shortest programs using decision tables and the special subroutines.
3) The particular example shown above indicates that faster programs are feasible, using a decision tree. There is, however, nothing on which to base a generalization about the speed of resulting programs.

In addition, the SPS subroutines allow changes to be made on the object level of compiled programs, thus allowing a greater flexibility. This is important for simulations because you can try several different rules and arrangements and study the effects they have on an organization without recompiling any programs.

The SPS subroutines are of the following lengths.

Random Number Generator	240
Read Table and Test	2164
Greater than or equal and Less than equal	290
Greater than and Less than	290
Equal and Unequal	280

---

[2]Pollack has met with recent criticism of his algorithm. His assumption has been proven not infallable by Sprague (11).

Yes and No..194

No Change..90

Read Data..36

Some of these are quite small and efficient; others (due to their complexity) are large and bulky. It is quite probable that some of the subroutines could be shortened and/or made to operate faster by another approach, either to the problem solution or to the techniques used in programming the solution. Due to its length and bulk the "Read Table and Test" subroutine could probably be shortened considerably.

FIGURES

PLATE I

Figure 1 is an example of a decision tree in which each condition has three states. Figure 2 illustrates a binary or limited-entry decision tree.

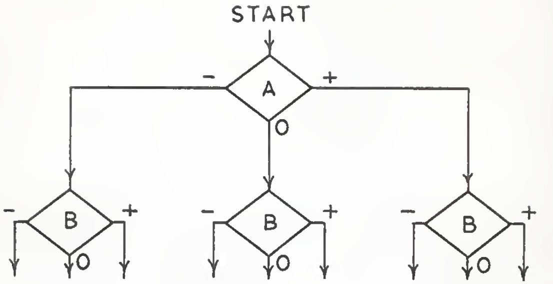
START
A
−
+
O
B
−
+
O
B
−
+
O
B
−
+
O

Fig. 1

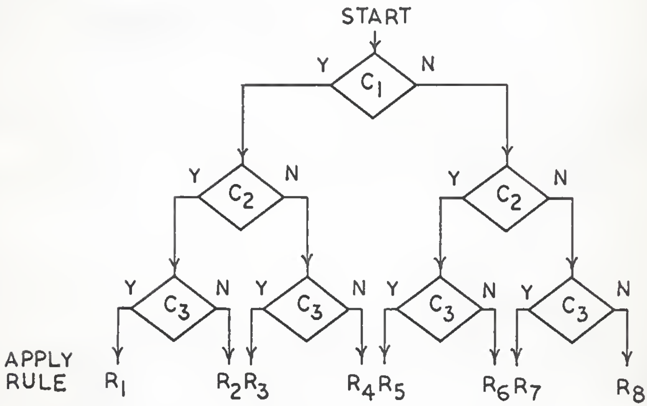
START
Y
C_1
N
Y
C_2
N
Y
C_2
N
Y
C_3
N
Y
C_3
N
Y
C_3
N
Y
C_3
N
APPLY
RULE
R_1
R_2
R_3
R_4
R_5
R_6
R_7
R_8

Fig. 2

PLATE II

Figure 3 is a credit approval decision table which is expanded to include all possible cases. Figure 4 is the same table after the rules have been combined with "not-pertinent" elements. Figure 5 is the credit approval table illustrated in the form of a tree.

	CONDITIONS	Logic Rules 1	2	3	4	5	6	7	8
C_1	Credit Limit OK	Y	Y	Y	Y	N	N	N	N
C_2	Pay Experience OK	Y	Y	N	N	Y	Y	N	N
C_3	Special Clearance	Y	N	Y	N	Y	N	Y	N
	ACTIONS								
A_1	Approve Order	X	X	X	X	X	X	X	
A_2	Disapprove Order								X

Fig. 3

	CONDITIONS	Logic Rules 1	2	3	4
C_1	Credit Limit OK	Y	N	N	N
C_2	Pay Experience OK	-	Y	N	N
C_3	Special Clearance	-	-	Y	N
	ACTIONS				
A_1	Approve Order	X	X	X	
A_2	Disapprove Order				X

Fig. 4

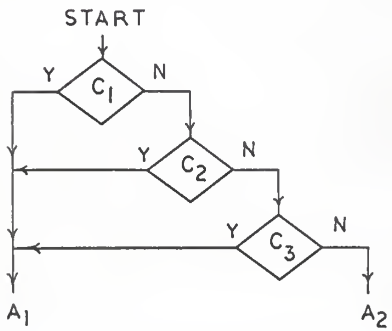

Fig. 5

PLATE III

Figure 6 is the credit approval table simplified by using the implied decision rule - "ELSE". Figure 7 is any decision table. Figure 8 is the simplified version of figure 7 after the introduction of the "not-pertinent" element.

	CONDITIONS	Logic Rules 1	E
C_1	Credit Limit OK	N	
C_2	Pay Experience OK	N	
C_3	Special Clearance	N	
	ACTIONS		
A_1	Approve Order		X
A_2	Disapprove Order	X	

Fig. 6

	R_1	R_2	R_3	R_4	R_E
C_1	Y	Y	N	N	
C_2	N	Y	Y	N	
C_3	Y	N	Y	Y	
A_1	X			X	
A_2		X	X		
A_E					X

Fig. 7

	$R_{1,4}$	R_2	R_3	R_E
C_1	-	Y	N	
C_2	N	Y	Y	
C_3	Y	N	Y	
A_1	X			
A_2		X	X	
A_E				X

Fig. 8

PLATE IV

Figures 9 and 10 illustrate decision tables which contain redundant or inconsistant rules. Figure 11 is a decision table which indicates the use of a dash count. Figure 12 illustrates the use of a dash count and the delta of each condition.

	R_1	R_2
C_1	-	N
C_2	N	N

Fig. 9

	R_1	R_2	R_3
C_1	-	Y	N

Fig. 10

	R_1	R_2	R_3	Dash Count
Column Count	4	4	2	
C_1	N	-	Y	4
C_2	N	Y	-	2
C_3	-	Y	N	4
C_4	-	-	N	8

Fig. 11

	R_1	R_2	R_3	R_4	R_5	R_6	Dash Count	Delta	Y Count	N Count
Column Count	1	2	1	2	2	2				
C_1	Y	N	N	N	N	-	2	6	1	7
C_2	Y	N	Y	-	Y	N	2	0	4	4
C_3	N	N	N	Y	-	Y	2	0	4	4
C_4	N	-	N	N	Y	Y	2	0	4	4

Fig. 12

PLATE V

Figure 13 illustrates the application of Pollack's algorithm for the expansion of decision tables into minimum decision trees.

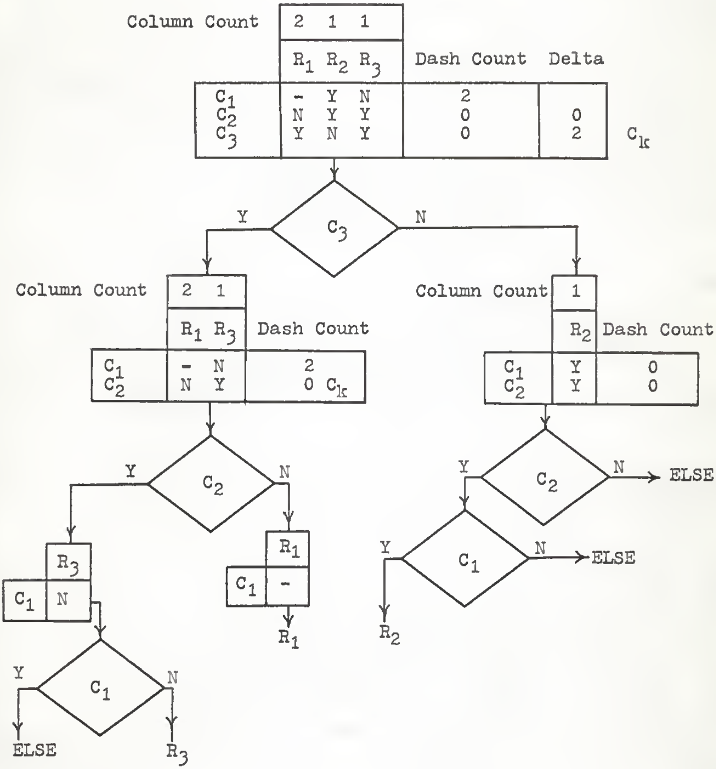
Column Count
2 1 1
R1 R2 R3
Dash Count
Delta
C1 C2 C3
- Y N
N Y Y
Y N Y
2 0 0
0 2
Ck
C3
Y
N
Column Count
2 1
R1 R3
Dash Count
C1 C2
- N
N Y
2 0 Ck
Column Count
1
R2
Dash Count
C1 C2
Y Y
0 0
C2
ELSE
R3
C1 N
R1
C1 -
C1
ELSE
R2

Fig. 13

PLATE VI

Figure 14 illustrates the credit approval table modified for programming in Fortran. Figure 15 illustrates a table containing conditions with more than two states modified for Fortran programming.

	CONDITIONS	Value	0	1	2	3	4	5	6	7
			Logic Rules							
C_1	Credit Limit OK	1		X		X		X		X
C_2	Pay Experience OK	2			X	X			X	X
C_3	Special Clearance	4					X	X	X	X
	ACTIONS									
A_1	Approve Order			X	X	X	X	X	X	X
A_2	Disapprove Order		X							

Fig. 14

CONDITIONS	Number States	Local Value	Multiplier	Net Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
					Logic Rules																							
Condition 1																												
State 1		0		0	X			X			X			X			X			X			X			X		
State 2	3	1	1	1		X			X			X			X			X			X			X			X	
State 3		2		2			X			X			X			X			X			X			X			X
Condition 2																												
State 1		0		0	X	X	X										X	X	X									
State 2	4	1	3	3				X	X	X										X	X	X						
State 3		2		6							X	X	X										X	X	X			
State 4		3		9										X	X	X										X	X	X
Condition 3																												
State 1	2	0	3·4=12	0	X	X	X	X	X	X	X	X	X	X	X	X												
State 2		1		12													X	X	X	X	X	X	X	X	X	X	X	X

Fig. 15

PLATE XII

Figure 16 is the table matrix for the credit approval table. Figure 17 is the masking matrix for the credit approval table. Figure 18 illustrates a table of logical multiplication.

	CONDITIONS	Logic Rules 1	2	3	4
C_1	Credit Limit OK	1	0	0	0
C_2	Pay Experience OK	0	1	0	0
C_3	Special Clearance	0	0	1	0

Fig. 16

	CONDITIONS	Logic Rules 1	2	3	4
C_1	Credit Limit OK	1	1	1	1
C_2	Pay Experience OK	0	1	1	1
C_3	Special Clearance	0	0	1	1

Fig. 17

	1	0
1	1	0
0	0	0

Fig. 18

PLATE VIII

Figure 19 is the decision table for the "pump and tank" simulation.

	CONDITIONS	Logic Rules 1	2	3	4	5	6	7	8	9	10
C_1	Tank In Use	-	-	-	-	0	1	1	1	1	0
C_2	Pump Running	1	0	1	0	1	0	0	1	1	0
C_3	Level Below 4 Feet	-	-	-	-	-	1	0	1	0	-
C_4	Level Above 5 Feet	1	1	0	0	0	0	0	0	0	0
C_5	Pump Temperature Above 180 F	-	-	1	1	0	0	0	0	0	0
	ACTIONS										
A_1	Start Pump	-	-	-	-	-	1	-	-	-	-
A_2	Stop Pump	1	-	1	-	1	-	-	-	1	-
A_3	Tank Level High Alarm	1	1	0	0	0	0	0	0	0	0
A_4	Pump Temperature High Alarm	-	-	1	1	0	0	0	0	0	0
	Action Set Number	1	2	3	4	5	6	7	7	5	7

Fig. 19

PLATE IX

Figure 20 is the flow logic used to simulate the pump and tank.

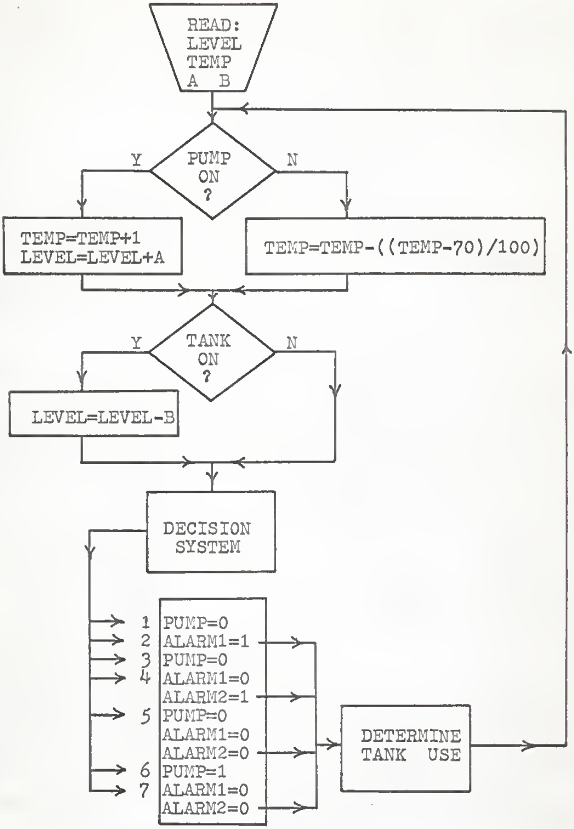
READ:
LEVEL
TEMP
A B
PUMP ON ?
Y
N
TEMP=TEMP+1
LEVEL=LEVEL+A
TEMP=TEMP-((TEMP-70)/100)
TANK ON ?
Y
N
LEVEL=LEVEL-B
DECISION SYSTEM
1 PUMP=0
2 ALARM1=1
3 PUMP=0
4 ALARM1=0
ALARM2=1
5 PUMP=0
ALARM1=0
ALARM2=0
6 PUMP=1
7 ALARM1=0
ALARM2=0
DETERMINE TANK USE

Fig. 20

PLATE X

Figure 21 is the unimproved decision tree for the pump simulation table.

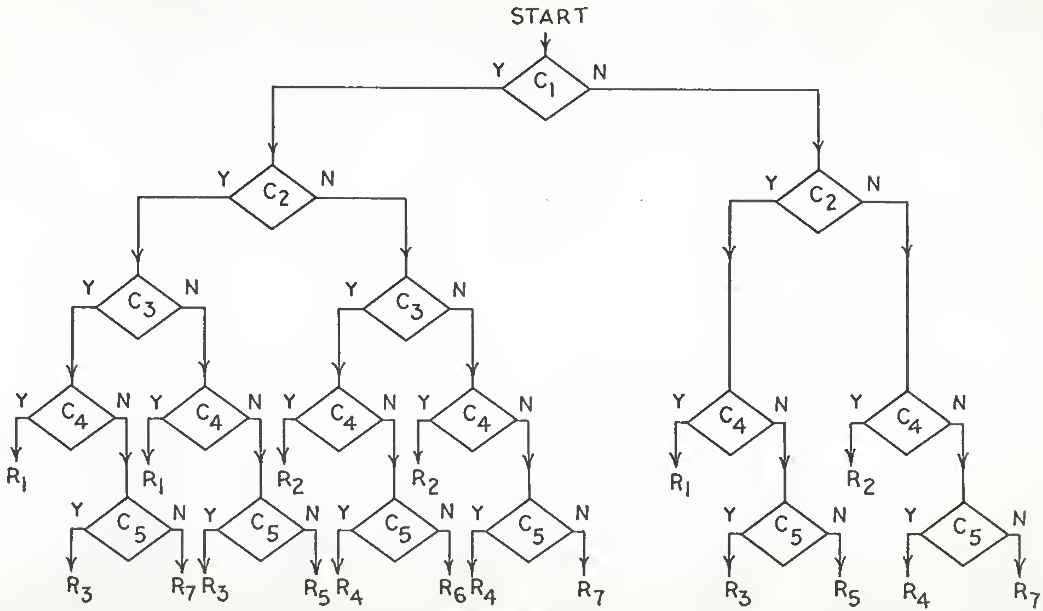
START
Y
C1
N
Y
C2
N
Y
C2
N
Y
C3
N
Y
C3
N
Y
C4
N
Y
C4
N
Y
C4
N
Y
C4
N
Y
C4
N
Y
C4
N
R1
R1
R2
R2
R1
R2
Y
C5
N
Y
C5
N
Y
C5
N
Y
C5
N
Y
C5
N
Y
C5
N
R3
R7
R3
R5
R4
R6
R4
R7
R3
R5
R4
R7

Fig. 21

PLATE XI

Figure 22 (3 pages) is the program for the unimproved decision tree for the pump simulation.

```
**      PUMP AND TANK SIMULATION (UNIMPROVED DECISION TREE)
   17 FORMAT(F7.0,F12.0,F13.1,F12.0,F11.0,F12.0)
   23 FORMAT(F6.3,F6.0,F6.3,F6.0,F6.3)
   60 FORMAT(3X7HTANK ON5X7HPUMP ON6X5HLEVEL7X4HTEMP5X11HLEVEL ALARM2X10
     1HTEMP ALARM)
   80 FORMAT(2HA=F6.3,5X2HB=F6.3)
C
C       A IS THE CAPACITY OF THE PUMP
C     B IS THE CAPACITY OF THE TANK VALVE
C     SIZE IS THE SIZE OF THE SIMULATION
C
   71 READ 23,A,SIZE,B,TEMP,ALEVEL
      PUNCH 80,A,B
      PUNCH 60
      TANK=0.
      PUMP=0.
      COUNT=0.
   70 IF(COUNT-SIZE)22,71,71
   22 IF(PUMP)8,8,9
    9 TEMP=TEMP+1.
      ALEVEL=ALEVEL+A
      GO TO 24
    8 TEMP=TEMP-((TEMP-70.)/100.)
   24 IF(TANK)10,10,11
   11 ALEVEL=ALEVEL-B
   10 IF(ALEVEL)12,13,13
   12 ALEVEL=0.
C
C     DECISION SYSTEM
C
   13 IF(TANK)52,52,41
   41 IF(PUMP)47,47,42
   42 IF(ALEVEL-4.)43,45,45
   43 IF(ALEVEL-5.)44,44,1
   44 IF(TEMP-180.)7,7,3
   45 IF(ALEVEL-5.)46,46,1
   46 IF(TEMP-180.)5,5,3
```

PLATE XII

Figure 22 continued.

```
   47 IF(ALEVEL-4.)48,50,50
   48 IF(ALEVEL-5.)49,49,2
   49 IF(TEMP-180.)6,6,4
   50 IF(ALEVEL-5.)51,51,2
   51 IF(TEMP-180.)7,7,4
   52 IF(PUMP)55,55,53
   53 IF(ALEVEL-5.)54,54,1
   54 IF(TEMP-180.)5,5,3
   55 IF(ALEVEL-5.)56,56,2
   56 IF(TEMP-180.)7,7,4
C
C  CHANGE THE VARIABLES ACCORDING TO THE DECISION
C
    1 PUMP=0.
    2 ALARM1=1.
      GO TO 14
    3 PUMP=0.
    4 ALARM1=0.
      ALARM2=1.
      GO TO 14
    5 PUMP=0.
      ALARM1=0.
      ALARM2=0.
      GO TO 14
    6 PUMP=1.
    7 ALARM1=0.
      ALARM2=0.
   14 COUNT=COUNT+1.
C
C  SWITCH 1 ON TO PUNCH STATUS
C  SWITCH 2 ON - TANK USE IS RANDOM
C           OFF - TANK CONTROLLED BY SWITCH 3
C  SWITCH 3 ON - TANK IN USE
C           OFF - TANK NOT IN USE
C
      IF(SENSE SWITCH 1) 15,16
   15 PUNCH 17,TANK,PUMP,ALEVEL,TEMP,ALARM1,ALARM2
```

PLATE XIII

Figure 22 concluded.

```
16 IF(SENSE SWITCH 2)18,19
18 IF(.5-RAN(.243))20,20,21
20 TANK=0.
   GO TO 70
21 TANK=1.
   GO TO 70
19 IF(SENSE SWITCH 3)21,20
   END
```

PLATE XIV

Figure 23 is the improved decision tree for the pump simulation.

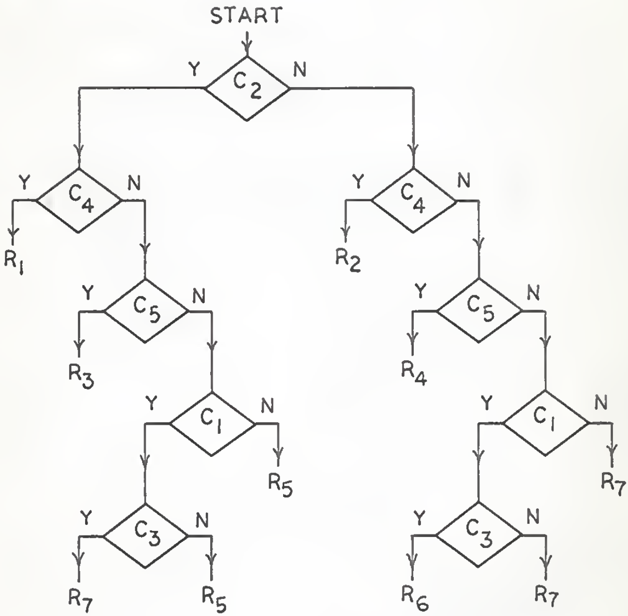
START
Y
C2
N
Y
C4
N
R1
Y
C4
N
R2
Y
C5
N
R3
Y
C5
N
R4
Y
C1
N
R5
Y
C1
N
R7
Y
C3
N
R7
R5
Y
C3
N
R6
R7

Fig. 23

PLATE XV

Figure 24 is the Fortran program for the improved decision tree for the pump simulation.

```
**    PUMP AND TANK SIMULATION (IMPROVED DECISION TREE)
   17 FORMAT(F7.0,F12.0,F13.1,F12.0,F11.0,F12.0)
   23 FORMAT(F6.3,F6.0,F6.3,F6.0,F6.3)
   60 FORMAT(3X7HTANK ON5X7HPUMP ON6X5HLEVEL7X4HTEMP5X11HLEVEL ALARM2X10
     1HTEMP ALARM)
   80 FORMAT(2HA=F6.3,5X2HB=F6.3)
C
C  A IS THE CAPACITY OF THE PUMP
C  B IS THE CAPACITY OF THE TANK VALVE
C  SIZE IS THE SIZE OF THE SIMULATION
C
   71 READ 23,A,SIZE,B,TEMP,ALEVEL
      PUNCH 80,A,B
      PUNCH 60
      TANK=0.
      PUMP=0.
      COUNT=0.
   70 IF(COUNT-SIZE)22,71,71
   22 IF(PUMP)8,8,9
    9 TEMP=TEMP+1.
      ALEVEL=ALEVEL+A
      GO TO 24
    8 TEMP=TEMP-((TEMP-70.)/100.)
   24 IF(TANK)10,10,11
   11 ALEVEL=ALEVEL-B
   10 IF(ALEVEL)12,13,13
   12 ALEVEL=0.
C
C  DECISION SYSTEM
C
   13 IF(PUMP)30,30,31
   30 IF(ALEVEL-5.)33,33,2
   33 IF(TEMP-180.)35,35,4
   35 IF(TANK)7,7,36
   36 IF(ALEVEL-4.)6,7,7
   31 IF(ALEVEL-5.)32,32,1
   32 IF(TEMP-180.)34,34,3
```

PLATE XVI

Figure 24 concluded.

```
   34 IF(TANK)5,5,37
   37 IF(ALEVEL-4.)7,7,5
C
C  CHANGE THE VARIABLES ACCORDING TO THE DECISION
C
    1 PUMP=0.
    2 ALARM1=1.
      GO TO 14
    3 PUMP=0.
    4 ALARM1=0.
      ALARM2=1.
      GO TO 14
    5 PUMP=0.
      ALARM1=0.
      ALARM2=0.
      GO TO 14
    6 PUMP=1.
    7 ALARM1=0.
      ALARM2=0.
   14 COUNT=COUNT+1.
C
C  SWITCH 1 ON TO PUNCH STATUS
C  SWITCH 2 ON - TANK USE IS RANDOM
C           OFF - TANK CONTROLLED BY SWITCH 3
C  SWITCH 3 ON - TANK IN USE
C           OFF - TANK NOT IN USE
C
      IF(SENSE SWITCH 1) 15,16
   15 PUNCH 17,TANK,PUMP,ALEVEL,TEMP,ALARM1,ALARM2
   16 IF(SENSE SWITCH 2)18,19
   18 IF(.5-RAN(.243))20,20,21
   20 TANK=0.
      GO TO 70
   21 TANK=1.
      GO TO 70
   19 IF(SENSE SWITCH 3)21,20
      END
```

PLATE XVII

Figure 25 is the expanded decision table for programming the pump simulation decision table in Fortran.

		Logic Rules																															
	value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
C_1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
C_2	2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
C_3	4	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C_4	8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
C_5	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Action Set		7	7	5	5	7	6	5	7	2	2	1	1	2	2	1	1	4	4	3	3	4	4	3	3	2	2	1	1	2	2	1	1

Fig. 25

PLATE XVIII

Figure 26 is the program for the pump simulation using the decision table in Fortran.

```
**     PUMP AND TANK SIMULATION (DEC TABLES WITHOUT MOD. PROCESSOR)
    17 FORMAT(I6,I12,F14.1,F12.0,F11.0,F12.0)
    23 FORMAT(F6.3,F6.0,F6.3,F6.0,F6.3)
    60 FORMAT(3X7HTANK ON5X7HPUMP ON6X5HLEVEL7X4HTEMP5X11HLEVEL ALARM2X10
      1HTEMP ALARM)
    80 FORMAT(2HA=F6.3,5X2HB=F6.3)
C
C   A IS THE CAPACITY OF THE PUMP
C   B IS THE CAPACITY OF THE TANK VALVE
C   SIZE IS THE SIZE OF THE SIMULATION
C
    71 READ 23,A,SIZE,B,TEMP,ALEVEL
       PUNCH 80,A,B
       PUNCH 60
       ITANK=0
       IPUMP=0
       COUNT=0.
    70 IF(COUNT-SIZE)22,71,71
    22 IF(IPUMP)8,8,9
     9 TEMP=TEMP+1.
       ALEVEL=ALEVEL+A
       GO TO 24
     8 TEMP=TEMP-((TEMP-70.)/100.)
    24 IF(ITANK)10,10,11
    11 ALEVEL=ALEVEL-B
    10 IF(ALEVEL)12,13,13
    12 ALEVEL=0.
C
C   DECISION SYSTEM
C
    13 IF(ALEVEL-4.)40,41,41
    40 I3=1
       GO TO 42
    41 I3=0
    42 IF(ALEVEL-5.)43,43,44
    44 I4=1
       GO TO 45
```

PLATE XIX

Figure 26 continued.

```
   43 I4=0
   45 IF(TEMP-180.)46,46,47
   47 I5=1
      GO TO 48
   46 I5=0
   48 JUMP=1+ITANK+2*IPUMP+4*I3+8*I4+16*I5
      GO TO (7,7,5,5,7,6,5,7,2,2,1,1,2,2,1,1,4,4,3,3,4,4,3,3,2,2,1,1,2,2
     1,1,1),JUMP
C
C  CHANGE THE VARIABLES ACCORDING TO THE DECISION
C
    1 IPUMP=0
    2 ALARM1=1.
      GO TO 14
    3 IPUMP=0
    4 ALARM1=0.
      ALARM2=1.
      GO TO 14
    5 IPUMP=0
      ALARM1=0.
      ALARM2=0.
      GO TO 14
    6 IPUMP=1
    7 ALARM1=0.
      ALARM2=0.
   14 COUNT=COUNT+1.
C
C  SWITCH 1 ON TO PUNCH STATUS
C  SWITCH 2 ON - TANK USE IS RANDOM
C           OFF - TANK CONTROLLED BY SWITCH 3
C  SWITCH 3 ON - TANK IN USE
C           OFF - TANK NOT IN USE
C
      IF(SENSE SWITCH 1) 15,16
   15 PUNCH 17,ITANK,IPUMP,ALEVEL,TEMP,ALARM1,ALARM2
   16 IF(SENSE SWITCH 2)18,19
   18 IF(.5-RAN(.243))20,20,21
```

PLATE XX

Figure 26 concluded.

```
20 ITANK=0
   GO TO 70
21 ITANK=1
   GO TO 70
19 IF(SENSE SWITCH 3)21,20
   END
```

PLATE XXI

Figure 27 is the pump decision table simplified by the use of "ELSE" for use by the modified processor.

	Logic Rules							
	1	2	3	4	5	6	7	E
C_1	-	-	-	-	0	1	1	
C_2	1	0	1	0	1	0	1	
C_3	-	-	-	-	-	1	0	
C_4	1	1	0	0	0	0	0	
C_5	-	-	1	1	0	0	0	
A_1	-	-	-	-	-	1	-	-
A_2	1	-	1	-	1	-	1	-
A_3	1	1	0	0	0	0	0	0
A_4	-	-	1	1	0	0	0	0

Fig. 27

PLATE XXII

Figure 28 is the program for simulating the pump and tank with the modified processor.

```
**    PUMP AND TANK SIMULATION (DEC TABLES WITH MODIFIED PROCESSOR)
   17 FORMAT(F7.0,F12.0,F13.1,F12.0,F11.0,F12.0)
   23 FORMAT(F6.3,F6.0,F6.3,F6.0,F6.3)
   60 FORMAT(3X7HTANK ON5X7HPUMP ON6X5HLEVEL7X4HTEMP5X11HLEVEL ALARM2X10
     1HTEMP ALARM)
   80 FORMAT(2HA=F6.3,5X2HB=F6.3)
C
C  A IS THE CAPACITY OF THE PUMP
C  B IS THE CAPACITY OF THE TANK VALVE
C  SIZE IS THE SIZE OF THE SIMULATION
C
   71 READ 23,A,SIZE,B,TEMP,ALEVEL
      PUNCH 80,A,B
      PUNCH 60
      ACTION=RDTB(1)
      TANK=0.
      PUMP=0.
      COUNT=0.
   70 IF(COUNT-SIZE)22,71,71
   22 IF(PUMP)8,8,9
    9 TEMP=TEMP+1.
      ALEVEL=ALEVEL+A
      GO TO 24
    8 TEMP=TEMP-((TEMP-70.)*.01)
   24 IF(TANK)10,10,11
   11 ALEVEL=ALEVEL-B
   10 IF(ALEVEL)12,13,13
   12 ALEVEL=0.
C
C  DECISION SYSTEM
C
   13 DATA=UN(TANK)+UN(PUMP)+LR(ALEVEL-4.)+GR(ALEVEL-5.)+GR(TEMP-180.)
      JUMP=TEST(1)
      GO TO (1,2,3,4,5,6,5,7),JUMP
C
C  CHANGE THE VARIABLES ACCORDING TO THE DECISION
C
```

PLATE XXIII

Figure 28 continued.

```
   1 PUMP=0.
   2 ALARM1=1.
     GO TO 14
   3 PUMP=0.
   4 ALARM1=0.
     ALARM2=1.
     GO TO 14
   5 PUMP=0.
     ALARM1=0.
     ALARM2=0.
     GO TO 14
   6 PUMP=1.
   7 ALARM1=0.
     ALARM2=0.
  14 COUNT=COUNT+1.
C
C   SWITCH 1 ON TO PUNCH STATUS
C   SWITCH 2 ON - TANK USE IS RANDOM
C            OFF - TANK CONTROLLED BY SWITCH 3
C   SWITCH 3 ON - TANK IN USE
C            OFF - TANK NOT IN USE
C
     IF(SENSE SWITCH 1) 15,16
  15 PUNCH 17,TANK,PUMP,ALEVEL,TEMP,ALARM1,ALARM2
  16 IF(SENSE SWITCH 2)18,19
  18 IF(.5-RAN(.243))20,20,21
  20 TANK=0.
     GO TO 70
  21 TANK=1.
     GO TO 70
  19 IF(SENSE SWITCH 3)21,20
     END
```

PLATE XXIX

Figure 28 concluded. Decision table input for the modified processor. Figure 29 is a comparison of the four different pump simulations.

```
0705
-1-1-#
-0-1-#
-1-01#
-0-01#
01-00#
10100#
11000#
```

Fig. 28 (concluded)

	Time for 1000 Passes (Minutes)	Length of Mainline Program	Length of Subroutines	Total Length
Program 1	1.901	14134	240	14374
Program 2	1.901	13590	240	13830
Program 3	2.762	13872	240	14112
Program 4	5.222	13514	3066	16580

Fig. 29

APPENDIX 1

Subroutines

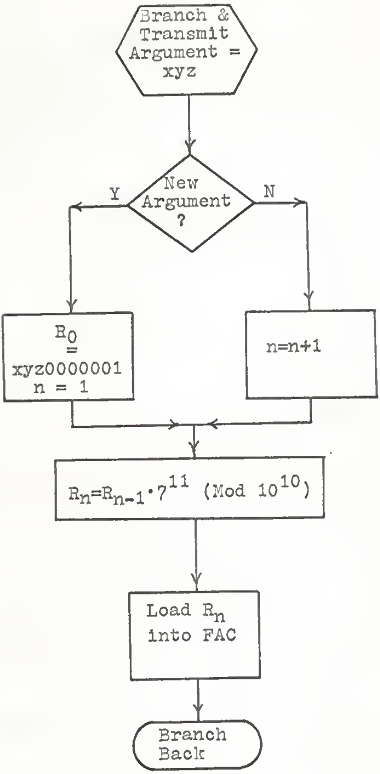
Branch &
Transmit
Argument =
xyz
New
Argument
?
Y
N
R_0
=
xyz0000001
n = 1
n=n+1
$R_n=R_{n-1}\cdot 7^{11}$ (Mod 10^{10})
Load R_n
into FAC
Branch
Back

Fig. A1

```
10000    00000            2          DORG 10000      *****  PROGRAM A1  *****
10000 06 00485 09999      4 RNDM     TFL  485        ,RNDM-1    ,1    ,
10012 24 10204 00483      6          C    STORE      ,483       ,0    ,DOES NEW ARGUMENT EQUAL OLD
10024 46 10084 01200      8          BE   CALC       ,          ,0    ,ARGUMENT.  IF NOT PUT THE NEW
10036 26 10204 00483     10          TF   STORE      ,483       ,0    ,ARGUMENT IN STORE AND
10048 26 10214 10222     12          TF   RNUM       ,CONST     ,01   ,DEFINE THE NEW MULTIPLICAND
10060 26 10207 10204     14          TF   RNUM-7     ,STORE     ,01   ,AS ABC0000001
10072 33 10207 00000     16          CF   RNUM-7     ,          ,0    ,
10084 23 10214 10232     18 CALC     M    RNUM       ,K         ,01   ,CALCULATE A RANDOM NUMBER
10096 32 00090 00000     20          SF   90         ,          ,     ,(UNIFORM DISTRIBUTION)
10108 26 10214 00099     22          TF   RNUM       ,99        ,0    ,STORE RN AS NEXT MULTIPLICAND.
10120 26 00483 00097     24          TF   483        ,97        ,     ,
10132 16 00485 00000     26          TFM  485        ,00        ,10   ,LOAD INTO FAC.
10144 43 10200 00476     28 NORM     BD   BACK       ,476       ,0    ,NORMALIZING ROUTINE
10156 05 00475 00483     30          FSL  475        ,483       ,     ,IF NECESSARY MOVE
10168 32 00476 00000     32          SF   476        ,          ,     ,THE MANTISSA
10180 12 00485 00001     34          SM   485        ,1         ,10   ,CHANGE THE CHARACTERISTIC
10192 49 10144 00000     36          B    NORM       ,          ,0    ,BRANCH TO TEST FOR ANOTHER MOVE
10200    00000           38          DORG *-3
10200 42 00000 00000     40 BACK     BB              ,          ,     ,BRANCH BACK TO MAINLINE PROGRAM
10202    00000           42          DORG *-9
10204    00003           44 STORE    DS   3
10214    00010           46 RNUM     DS   10
10222    00008           48 CONST    DC   8,1
10232    00010           51 K        DC   10,1977326743
10000    00000           54          DEND RNDM
```

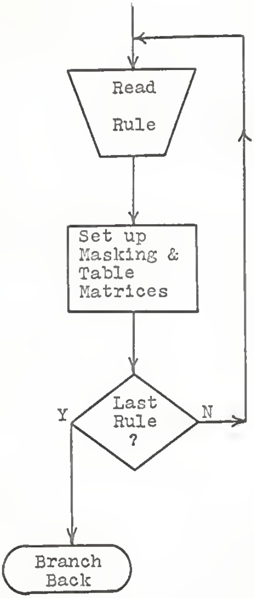
Read
Rule
Set up
Masking &
Table
Matrices
Last
Rule
?
Y
N
Branch
Back

Fig. A2(a)

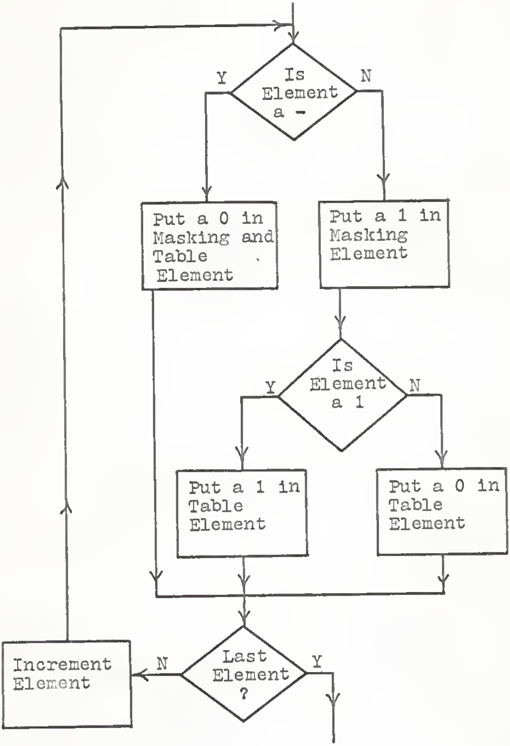
Is
Element
a -
Y
N
Put a 0 in
Masking and
Table
Element
Put a 1 in
Masking
Element
Is
Element
a 1
Y
N
Put a 1 in
Table
Element
Put a 0 in
Table
Element
Increment
Element
N
Last
Element
?
Y

Fig. A2(b)

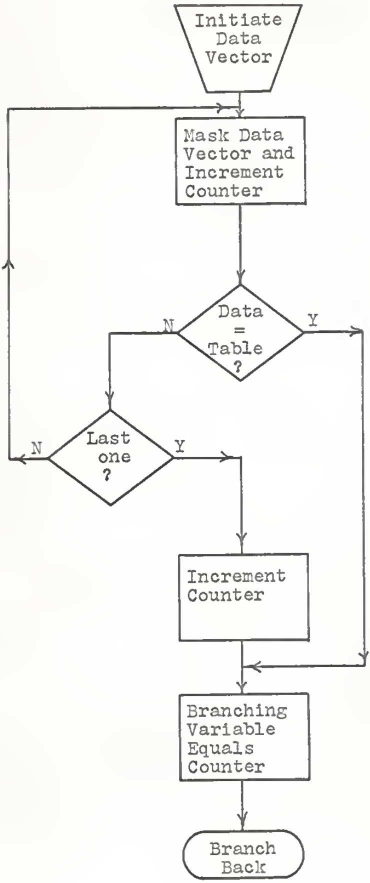
Initiate
Data
Vector
Mask Data
Vector and
Increment
Counter
Data
=
Table
?
N
Y
Last
one
?
N
Y
Increment
Counter
Branching
Variable
Equals
Counter
Branch
Back

Fig. A3(a)

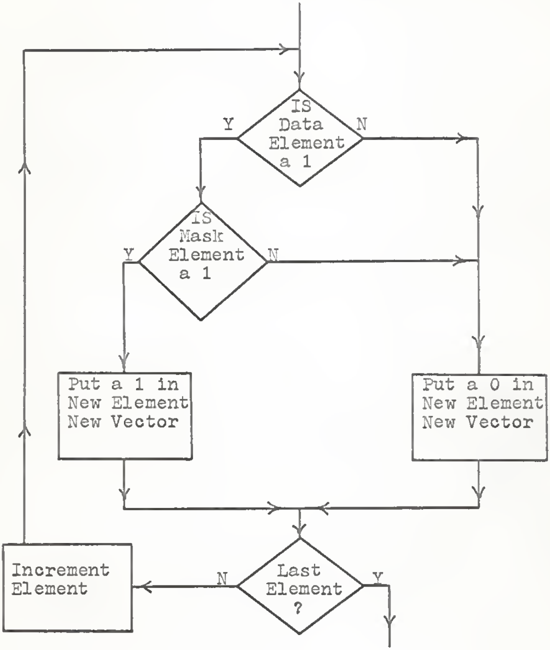
IS Data Element a 1
Y
N
IS Mask Element a 1
Y
N
Put a 1 in New Element New Vector
Put a 0 in New Element New Vector
Last Element ?
N
Y
Increment Element

Fig. A3(b)

```
10000    00000              2         DORG 10000        *****  PROGRAM A2  *****
10000 26 09999 09999        4 RDTB    TF   *-1          ,*-1        ,0111,
10012 32 09997 00000        6         SF   RDTB-3       ,           ,0    ,IF THIS IS NOT THE FIRST TABLE
10024 14 12240 00001        8         CM   CNT          ,1          ,0    ,
10036 47 10084 01200       10         BNE  ENDBLK-12    ,           ,0    ,
10048 11 12240 00001       12         AM   CNT          ,1          ,07   ,
10060 15 59999 00000       14         TDM  59999        ,0          ,     ,
10071    00001             16         DC   1            ,-          ,*    ,
10072 15 59998 00000       19         TDM  59998        ,0          ,     ,
10083    00001             21         DC   1            ,-          ,*    ,
10084 45 10116 59998       24         BNR  XX           ,59998      ,0    ,
10096 26 12250 09999       26 ENDBLK  TF   -EVEN        ,RDTB-1     ,01   ,GO TO RDHD.
10108 49 10204 00000       28         B    RDHD         ,           ,0    ,
10116    00000             30         DORG *-3
10116 45 10172 12245       32 XX      BNR  X            ,-ODD       ,01   ,IF THIS IS THE WRONG POSITION
10128 45 10148 12250       34 ENDTBL  BNR  COMP         ,-EVEN      ,01   ,GO TO X. IF THIS IS THE END OF
10140 49 10096 00000       36         B    ENDBLK       ,           ,0    ,THE BLOCK GO TO ENDBLK. IF NOT
10148    00000             38         DORG *-3                            ,GO TO COMP.
10148 24 12250 09999       40 COMP    C    -EVEN        ,RDTB-1     ,01   ,IF THE NEW TABLE NUMBER IS THE
10160 46 10204 01200       42         BE   RDHD         ,           ,0    ,SAME AS THIS ONE GO TO RDHD.
10172 12 12245 00002       44 X       SM   ODD          ,2          ,07   ,IF NOT CHANGE YOUR POSITION IN
10184 12 12250 00002       46         SM   EVEN         ,2          ,07   ,THE TABLE.  GO TO XX.
10196 49 10116 00000       48         B    XX           ,           ,0    ,
10204    00000             50         DORG *-3
10204 12 12250 00004       52 RDHD    SM   EVEN         ,4          ,07   ,
10216 12 12245 00004       54         SM   ODD          ,4          ,07   ,
10228 26 10434 12245       56         TF   B1+6         ,ODD        ,01   ,
10240 26 10446 12250       58         TF   B2+6         ,EVEN       ,01   ,
10252 26 10466 12250       60         TF   TN+6         ,EVEN       ,01   ,
10264 26 10490 12245       62         TF   B3+6         ,ODD        ,01   ,
10276 26 10510 12245       64         TF   TY+6         ,ODD        ,01   ,
10288 26 10698 12245       66         TF   FLAG+6       ,ODD        ,01   ,
10300 11 10698 00001       68         AM   FLAG+6       ,1          ,07   ,
10312 26 10766 12245       70         TF   MARK+6       ,ODD        ,01   ,
10324 36 11906 00500       72         RNCD CHK2-1       ,           ,0    ,READ THE HEADER CARD (NUMBER
10336 32 11906 00000       74         SF   CHK2-1       ,           ,0    ,RULES AND CONDITIONS)
10348 32 11908 00000       76         SF   CHK2+1       ,           ,0    ,
```

```
10360 22 10766 11909      78        S     MARK+6     ,CHK2+2   ,01   ,
10372 22 10766 11909      80        S     MARK+6     ,CHK2+2   ,01   ,
10384 36 11910 00500      82 RDRL   RNCD  RULE       ,         ,0    ,READ A RULE.
10396 45 10416 ±1910      84 A1     BNR   TX         ,RULE     ,017  ,SET UP THE TABLE AND MASKING
10408 49 10620 00000      86        B     CUT        ,         ,0    ,MATRICES UNTIL A RECORD MARK
10416    00000            88        DORG  *-3                        ,STOPS YOU.
10416 44 10460 ±1910      90 TX     BNF   TN         ,RULE     ,017  ,
10428 15 59995 00000      92 B1     TDM   59995      ,0        ,2    ,
10440 15 59994 00000      94 B2     TDM   59994      ,0        ,2    ,
10452 49 10516 00000      96        B     NWEL       ,         ,0    ,
10460    00000            98        DORG  *-3
10460 15 59994 00001     100 TN     TDM   59994      ,1        ,2    ,
10472 43 10504 ±1910     102 A2     BD    TY         ,RULE     ,017  ,
10484 15 59995 00000     104 B3     TDM   59995      ,0        ,2    ,
10496 49 10516 00000     106        B     NWEL       ,         ,0    ,
10504    00000           108        DORG  *-3
10504 15 59995 00001     110 TY     TDM   59995      ,1        ,2    ,
10516 ±2 10434 00002     112 NWEL   SM    B1+6       ,2        ,010  ,
10528 ±2 10446 00002     114        SM    B2+6       ,2        ,010  ,
10540 ±2 10466 00002     116        SM    TN+6       ,2        ,010  ,
10552 ±2 10490 00002     118        SM    B3+6       ,2        ,010  ,
10564 ±2 10510 00002     120        SM    TY+6       ,2        ,010  ,
10576 ±1 10407 00001     122        AM    A1+11      ,1        ,010  ,
10588 ±1 10427 00001     124        AM    TX+11      ,1        ,010  ,
10600 ±1 10483 00001     126        AM    A2+11      ,1        ,010  ,
10612 49 10396 00000     128        B     A1         ,         ,0    ,
10620    00000           130        DORG  *-3
10620 ±1 11991 00001     132 CUT    AM    CHK1       ,1        ,010  ,
10632 ±6 10407 ±1910     134        TFM   A1+11      ,RULE     ,017  ,
10644 ±6 10427 ±1910     136        TFM   TX+11      ,RULE     ,017  ,
10656 ±6 10483 ±1910     138        TFM   A2+11      ,RULE     ,017  ,
10668 22 10698 11909     140 GOBK   S     FLAG+6     ,CHK2+2   ,01   ,
10680 22 10698 11909     142        S     FLAG+6     ,CHK2+2   ,01   ,
10692 32 59996 00000     144 FLAG   SF    59996      ,         ,2    ,
10704 24 11991 11907     146        C     CHK1       ,CHK2     ,01   ,IF THIS IS THE LAST RULE GO TO
10716 46 10760 01200     148        BE    MARK       ,         ,0    ,MARK. IF NOT GO BACK TO RDRL.
10728 22 10766 11909     150        S     MARK+6     ,CHK2+2   ,01   ,
```

```
10740 22 10766 11909   152        S     MARK+6    ,CHK2+2   ,01   ,
10752 49 10384 00000   154        B     RDRL      ,         ,0    ,
10760    00000         156        DORG  *-3
10760 15 59997 00000   158 MARK   TDM   59997     ,         ,2    ,SET UP TWO RECORD MARKS TO
10771    00001         160        DC    1         ,-        ,*    ,INDICATE THE END OF THE BLOCK
10772 26 10802 10766   163        TF    *+30      ,MARK+6   ,01   ,OF TABLES.
10784 12 10802 00001   165        SM    *+18      ,1        ,07   ,
10796 15 00000 00000   167        TDM             ,         ,2    ,
10807    00001         169        DC    1         ,-        ,*    ,
10808 12 10434 00002   172        SM    B1+6      ,2        ,010  ,
10820 12 10446 00002   174        SM    B2+6      ,2        ,010  ,
10832 12 10466 00002   176        SM    TN+6      ,2        ,010  ,
10844 12 10490 00002   178        SM    B3+6      ,2        ,010  ,
10856 12 10510 00002   180        SM    TY+6      ,2        ,010  ,
10868 16 11991 00000   182        TFM   CHK1      ,0        ,010  ,
10880 16 00485 00000   184        TFM   FAC       ,0        ,10   ,
10892 16 00483 00111   186        TFM   FAC-2     ,111      ,9    ,
10904 42 00000 00000   188        BB              ,         ,     ,BRANCH BACK
10916 26 10915 10915   190 DEC    TF    *-1       ,*-1      ,0111,
10928 32 10913 00000   192        SF    DEC-3     ,         ,0    ,FIND OUT WHICH TABLE TO USE IN
10940 24 10915 59998   194        C     DEC-1     ,59998    ,07   ,THE TESTING AND FIND IT.  THEN
10952 46 11024 01200   196        BE    DEVT      ,         ,0    ,GO TO DEVT.
10964 49 10992 00000   198        B     INC       ,         ,0    ,
10972    00000         200        DORG  *-3
10972 45 10992 59999   202 Y      BNR   INC       ,59999    ,07   ,
10984 49 10928 00000   204        B     DEC+12    ,         ,0    ,
10992    00000         206        DORG  *-3
10992 12 10951 00002   208 INC    SM    DEC+35    ,2        ,07   ,
11004 12 10983 00002   210        SM    Y+11      ,2        ,07   ,
11016 49 10972 00000   212        B     Y         ,         ,0    ,
11024    00000         214        DORG  *-3
11024 32 11090 00000   216 DEVT   SF    11090     ,         ,6    ,
11036 26 12231 11080   218        TF    REG4      ,11080    ,0    ,MOVE DATA VECTOR INTO REG4.
11048 12 11174 00001   220        SM    INV+6     ,'        ,07   ,
11060 12 11083 00001   222        SM    *+23      ,1        ,07   ,
11072 44 11048 11081   224        BNF   DEVT+24   ,11081    ,07   ,
11084 26 11275 10951   226        TF    DET+11    ,DEC+35   ,01   ,
```

```
11096 ±2 11275 θ0004   228        SM   DET+11      ,4        ,07   ,
11108 25 11319 11275   230        TF   NEXT+11     ,DET+11   ,01   ,
11120 26 11419 11275   232        TF   TEST+11     ,DET+11   ,01   ,
11132 ±1 11419 θ0001   234        AM   TEST+11     ,1        ,07
11144 26 11467 11275   236        TF   RECMRK+11   ,DET+11   ,01   ,
11156 ±1 11467 θ0002   238        AM   RECMRK+11   ,-        ,07   ,
11168 25 ±1081 ±2231   240 INV    TD   11081       ,REG4     ,127  ,
11180 ±2 11179 θ0001   242        SM   INV+11      ,1        ,07   ,
11192 ±1 11174 θ0001   244        AM   INV+6       ,1        ,07   ,
11204 ±2 11227 θ0001   246        SM   *+23        ,1        ,07   ,
11216 44 11168 ±2232   248        BNF  INV         ,REG4+1   ,017  ,
11228 26 11246 11090   250        TF   *+18        ,11090    ,0    ,
11240 32 00000 00000   252        SF   0           ,         ,     ,
11252 33 11080 00000   254        CF   11080       ,         ,     ,
11264 43 11296 59994   256 DET    BD   DIG         ,59994    ,0    ,
11276 ±5 ±2071 00000   258 B4     TDM  REG1        ,0        ,02   ,
11288 49 11308 00000   260        B    NEXT        ,         ,0    ,
11296    00000         262        DORG *-3
11296 25 ±2071 ±2231   264 DIG    TD   REG1        ,REG4     ,0127,
11308 44 11340 59994   266 NEXT   BNF  C           ,59994    ,07   ,
11320 26 11080 12231   268        TF   11080       ,REG4     ,1    ,
11332 49 11408 00000   270        B    TEST        ,         ,0    ,
11340    00000         272        DORG *-3
11340 ±2 11275 000θ2   274 C      SM   DET+11      ,2        ,010  ,
11352 ±2 11282 000θ1   276        SM   B4+6        ,1        ,010  ,
11364 ±2 11307 000θ1   278        SM   DIG+11      ,1        ,010  ,
11376 ±2 11302 000θ1   280        SM   DIG+6       ,1        ,010  ,
11388 ±2 11319 000θ2   282        SM   NEXT+11     ,2        ,010  ,
11400 49 11264 00000   284        B    DET         ,         ,0    ,
11408    00000         286        DORG *-3
11408 25 ±2151 59995   288 TEST   TD   REG3        ,59995    ,027  ,MASK THE DATA VECTOR IN REG1.
11420 ±2 11419 θ0002   290        SM   TEST+11     ,2        ,07   ,SET UP THE TABLE VECTOR IN
11432 ±2 11467 θ0002   292        SM   RECMRK+11   ,2        ,07   ,REG3.
11444 ±2 11414 θ0001   294        SM   TEST+6      ,1        ,07   ,
11456 44 11408 59996   296 RECMRK BNF  TEST        ,59996    ,07   ,
11468 26 11498 11414   298        TF   D1+6        ,TEST+6   ,01   ,
11480 ±1 11498 000θ1   300        AM   D1+6        ,1        ,010  ,
```

```
11492 32 00000 00000     302 D1      SF   0          ,            ,    ,
11504 26 11522 11282     304         TF   D2+6       ,B4+6        ,01  ,
11516 32 00000 00000     306 D2      SF   0          ,            ,    ,
11528 24 12071 12151     308         C    REG1       ,REG3        ,01  ,COMPARE REG1 TO REG3.
11540 46 11700 01200     310         BE   OUT1       ,            ,0   ,IF THEY ARE EQUAL GO TO OUT1.
11552 26 11587 11467     312         TF   E+11       ,RECMRK+11,01     ,IF NOT TRY THE NEXT SET OF
11564 12 11587 00001     314         SM   E+11       ,1           ,07  ,VECTORS.
11576 45 11608 00000     316 E       BNR  F          ,0           ,0   ,
11588 11 12235 00001     318         AM   SVE3       ,1           ,08  ,ADVANCE SVE3 FOR EACH ITERATION
11600 49 11700 00000     320         B    OUT1       ,            ,0   ,
11608    00000           322         DORG *-3
11608 16 11307 12231     324 F       TFM  DIG+11     ,REG4        ,017 ,
11620 16 11302 12071     326         TFM  DIG+6      ,REG1        ,017 ,
11632 16 11414 12151     328         TFM  TEST+6     ,REG3        ,017 ,
11644 12 11275 00002     330         SM   DET+11     ,2           ,07  ,
11656 12 11319 00002     332         SM   NEXT+11    ,2           ,07  ,
11668 16 11282 12071     334         TFM  B4+6       ,REG1        ,017 ,
11680 11 12235 00001     336         AM   SVE3       ,1           ,08  ,
11692 49 11264 00000     338         B    DET        ,            ,0   ,
11700    00000           340         DORG *-3
11700 16 11282 12071     342 OUT1    TFM  B4+6       ,REG1        ,017 ,RESTORE THE SUBROUTINES.
11712 16 11307 12231     344         TFM  DIG+11     ,REG4        ,07  ,
11724 16 11302 12071     346         TFM  DIG+6      ,REG1        ,017 ,
11736 16 11414 12151     348         TFM  TEST+6     ,REG3        ,017 ,
11748 16 00485 00002     350         TFM  FAC        ,2           ,10  ,
11760 16 00483 00000     352         TFM  FAC-2      ,0           ,7   ,
11772 16 00478 00000     354         TFM  FAC-7      ,0           ,9   ,
11784 16 11083 11081     356         TFM  DEVT+59    ,11081       ,07  ,
11796 16 11179 12231     358         TFM  INV+11     ,REG4        ,017 ,
11808 16 11227 12232     360         TFM  DET-37     ,REG4+1      ,017 ,
11820 16 10951 59998     362         TFM  DEC+35     ,59998       ,07  ,
11832 16 10983 59999     364         TFM  Y+11       ,59999       ,07  ,
11844 16 11174 11081     366         TFM  INV+6      ,11081       ,07  ,
11856 33 00479 00000     368         CF   FAC-6      ,            ,    ,SET UP THE BRANCHING VARIABLE.
11868 26 00485 12235     370         TF   FAC        ,SVE3        ,1   ,THEN RESTORE SVE3.
11880 16 12235 00001     372         TFM  SVE3       ,1           ,08  ,
11892 16 11090 11080     374         TFM  11090      ,11080       ,7   ,
```

```
11904 42 00000 00000   376 BACK   BB                 ,          ,      ,BRANCH BACK
11906    00000         378        DORG *-9
00485    00001         380 FAC    DS   1,485
11080    00080         382 REG2   DS   80            ,11080     ,      ,
11907    00002         384 CHK2   DS   2,
11909    00002         386        DS   2,
11910    00080         388 RULE   DSS  80
11991    00002         390 CHK1   DC   2,0
12071    00080         393 REG1   DS   80
12151    00080         395 REG3   DS   80
12231    00080         397 REG4   DS   80
12235    00004         399 SVE3   DC   4,1
12240    00005         402 CNT    DC   5             ,1
12245    00005         405 ODD    DC   5,59999
12250    00005         408 EVEN   DC   5,59998
10000    00000         411        DEND RDTB
```

```
10000       00000                 2         DORG 10000       *****   PROGRAM A3   *****
10000 06 00485 09999              4 GE      TFL   485        ,*-1       ,111 ,IF ARG IS POSITIVE
10012 32 11086 00000              6         SF    11086      ,          ,    ,
10024 14 11090 00000              8         CM    11090      ,0         ,7   ,
10036 47 10060 01200             10         BNE   *+24       ,          ,0   ,
10048 16 11090 11080             12         TFM   11090      ,11080     ,7   ,
10060 44 10220 00483             14         BNF   YES        ,483       ,0   ,GO TO YES.  IF NOT
10072 49 10188 00000             16         B     NO         ,          ,0   ,GO TO NO.
10084 06 00485 10083             18 LE      TFL   485        ,*-1       ,111 ,
10096 32 11086 00000             20         SF    11086      ,          ,    ,
10108 14 11090 00000             22         CM    11090      ,0         ,7   ,
10120 47 10144 01200             24         BNE   *+24       ,          ,0   ,
10132 16 11090 11080             26         TFM   11090      ,11080     ,7   ,
10144 44 10164 00483             28         BNF   SEE        ,483       ,0   ,IF ARG IS POSITIVE
10156 49 10220 00000             30         B     YES        ,          ,0   ,GO TO SEE.  IF NOT
10164       00000                32         DORG *-3                         ,GO TO YES.
10164 14 00483 00000             34 SEE     CM    483        ,0         ,9   ,IF ARG IS ZERO GO
10176 46 10220 01200             36         BE    YES        ,          ,0   ,TO YES.  IF NOT GO TO NO.
10188 26 10206 11090             38 NO      TF    *+18       ,HOLD1     ,0   ,GENERATE A ZERO.
10200 15 00000 00000             40         TDM              ,0         ,    ,
10212 49 10244 00000             42         B     YES+24     ,          ,0   ,
10220       00000                44         DORG *-3
10220 26 10238 11090             46 YES     TF    *+18       ,HOLD1     ,0   ,GENERATE A ONE.
10232 15 00000 00001             48         TDM   ,1
10244 12 11090 00001             50         SM    HOLD1      ,1         ,10  ,DECREMENT HOLD1
10256 16 00485 00000             52         TFM   485        ,00        ,10  ,
10268 16 00483 00111             54         TFM   483        ,111       ,9   ,
10280 42 00000 00000             56         BB               ,          ,    ,BRANCH BACK
10281       00000                58         DORG *-10
11090       00005                60 HOLD1   DC    5          ,11080     ,11090
10000       00000                63         DEND GE
```

```
10000       00000              2          DORG 10000      *****  PROGRAM A4  *****
10000 06 00485 09999           4 GR       TFL  485        ,*-1       ,111 ,IF ARG IS NEGATIVE
10012 32 11086 00000           6          SF   11086      ,
10024 14 11090 00000           8          CM   11090      ,0         ,7   ,
10036 47 10060 01200          10          BNE  *+24       ,          ,0   ,
10048 16 11090 11080          12          TFM  11090      ,11080     ,7   ,
10060 44 10164 00483          14          BNF  SEE        ,483       ,0   ,GO TO NO.  IF NOT
10072 49 10188 00000          16          B    NO         ,          ,0   ,GO TO SEE.
10084 06 00485 10089          18 LR       TFL  485        ,*-1       ,111 ,IF ARG IS POSITIVE
10096 32 11086 00000          20          SF   11086      ,
10108 14 11090 00000          22          CM   11090      ,0         ,7   ,
10120 47 10144 01200          24          BNE  *+24       ,          ,0   ,
10132 16 11090 11080          26          TFM  11090      ,11080     ,7   ,
10144 44 10188 00483          28          BNF  NO         ,483       ,0   ,GO TO NO.
10156 49 10220 00000          30          B    YES        ,          ,0   ,IF NOT GO TO YES.
10164       00000             32          DORG *-3
10164 14 00483 00000          34 SEE      CM   483        ,0         ,9   ,IF ARG IS ZERO GO TO YES.
10176 47 10220 01200          36          BNE  YES        ,          ,0   ,IF NOT GO TO NO.
10188 26 10206 11090          38 NO       TF   *+18       ,HOLD1     ,0   ,GENERATE A ZERO.
10200 15 00000 00000          40          TDM             ,0         ,    ,
10212 49 10244 00000          42          B    YES+24     ,          ,0   ,
10220       00000             44          DORG *-3
10220 26 10238 11090          46 YES      TF   *+18       ,HOLD1     ,0   ,GENERATE A ONE.
10232 15 00000 00001          48          TDM  ,1
10244 12 11090 00001          50          SM   HOLD1      ,1         ,10  ,DECREMENT HOLD1
10256 16 00485 00000          52          TFM  485        ,00        ,10  ,
10268 16 00483 00111          54          TFM  483        ,111       ,9   ,
10280 42 00000 00000          56          BB              ,          ,    ,BRANCH BACK
10281       00000             58          DORG *-10
11090       00005             60 HOLD1    DC   5          ,11080     ,11090
10000       00000             63          DEND GR
```

```
10000       00000             2         DORG 10000     *****  PROGRAM  A5  *****
10000 06 00485 09999          4 EQ      TFL   485      ,*-1       ,111 ,IF ARG IS ZERO GO TO YES.
10012 32 11086 00000          6         SF    11086    ,          ,    ,
10024 14 11090 00000          8         CM    11090    ,0         ,7   ,
10036 47 10060 01200         10         BNE   *+24     ,          ,0   ,
10048 16 11090 11080         12         TFM   11090    ,11080     ,7   ,
10060 14 00483 00000         14         CM    483      ,0         ,9   ,
10072 47 10180 01200         16         BNE   NO       ,          ,0   ,IF NOT GO TO NO.
10084 49 10212 00000         18         B     YES      ,          ,0   ,
10096 06 00485 10095         20 UN      TFL   485      ,*-1       ,111 ,IF ARG IS ZERO GO TO NO.
10108 32 11086 00000         22         SF    11086    ,          ,
10120 14 11090 00000         24         CM    11090    ,0         ,7   ,
10132 47 10156 01200         26         BNE   *+24     ,          ,0   ,
10144 16 11090 11080         28         TFM   11090    ,11080     ,7   ,
10156 14 00483 00000         30         CM    483      ,0         ,9   ,
10168 47 10212 01200         32         BNE   YES      ,          ,0   ,IF NOT GO TO YES.
10180 26 10198 11090         34 NO      TF    *+18     ,HOLD1     ,0   ,GENERATE A ZERO.
10192 15 00000 00000         36         TDM            ,0         ,    ,
10204 49 10236 00000         38         B     YES+24   ,          ,0   ,
10212       00000            40         DORG *-3
10212 26 10230 11090         42 YES     TF    *+18     ,HOLD1     ,0   ,
10224 15 00000 00001         44         TDM            ,1         ,    ,GENERATE A ONE.
10236 12 11090 00001         46         SM    HOLD1    ,1         ,10  ,DECREMENT HOLD1.
10248 16 00485 00000         48         TFM   485      ,00        ,10  ,
10260 16 00483 00111         50         TFM   483      ,111       ,9   ,
10272 42 00000 00000         52         BB             ,          ,    ,BRANCH BACK
10273       00000            54         DORG *-10
11090       00005            56 HOLD1   DC    5        ,11080     ,11090
10000       00000            59         DEND EQ
```

```
10000       00000               2          DORG 10000      *****  PROGRAM A6  *****
10000 32 11086 00000            4 NO       SF   11086      ,            ,     ,
10012 14 11090 θ0000            6          CM   11090      ,0           ,7    ,
10024 47 10048 01200            8          BNE  *+24       ,            ,0    ,
10036 16 11090 ±1080           10          TFM  11090      ,-1080       ,7    ,
10048 26 10066 11090           12          TF   *+18       ,HOLD1       ,0    ,GENERATE A ZERO.
10060 15 00000 00000           14          TDM             ,0           ,     ,
10072 49 10152 00000           16          B    YESS+24    ,            ,0    ,
10080       00000              18          DORG *-3
10080 32 11086 00000           20 YES      SF   11086      ,            ,     ,
10092 14 11090 θ0000           22          CM   11090      ,0           ,7    ,
10104 47 10128 01200           24          BNE  *+24       ,            ,0    ,
10116 16 11090 ±1080           26          TFM  11090      ,11080       ,7    ,
10128 26 10146 11090           28 YESS     TF   *+18       ,HOLD1       ,0    ,GENERATE A ONE.
10140 15 00000 00001           30          TDM             ,1           ,     ,
10152 12 11090 000θ1           32          SM   HOLD1      ,1           ,10   ,DECREMENT HOLD1.
10164 16 00485 000θ0           34          TFM  485        ,00          ,10   ,
10176 16 00483 00±11           36          TFM  483        ,111         ,9    ,
10188 42 00000 00000           38          BB              ,            ,     ,BRANCH BACK
10189       00000              40          DORG *-10
11090       00005              42 HOLD1    DC   5          ,11080       ,11090
10000       00000              45          DEND NO
```

```
10000       00000          2          DORG 10000      *****  PROGRAM A7  *****
10000 32 11086 00000       4 NC       SF   11086      ,           ,    ,
10012 14 11090 θ0000       6          CM   11090      ,0          ,7   ,
10024 47 10048 01200       8          BNE  *+24       ,           ,0   ,
10036 16 11090 ±1080      10          TFM  11090      ,11080      ,7   ,
10048 12 11090 000θ1      12          SM   HOLD1      ,1          ,10  ,DECREMENT HOLD1.
10060 16 00485 000θ0      14          TFM  FAC        ,00         ,10  ,LOAD DUMMY INTO FAC.
10072 16 00483 00±11      16          TFM  FAC-2      ,111        ,9   ,
10084 42 00000 00000      18          BB              ,           ,    ,
11090       00005         20 HOLD1    DC   5          ,11080      ,11090
00485       00001         23 FAC      DS   1          ,00485
10000       00000         25          DEND NC
```

```
10000       00000          2          DORG 10000      *****  PROGRAM A8  *****
10000 36 11001 00500       4 RDATA    RNCD 11001      ,           ,    ,READ CARD INTO REG.
10012 16 00485 000θ0       6          TFM  485        ,00         ,10  ,
10024 16 00483 00±11       8          TFM  483        ,111        ,9   ,
10036 42 00000 00000      10          BB              ,           ,    ,BRANCH BACK
10000       00000         12          DEND RDATA
```

APPENDIX 2

Pump and Tank Simulation Results

A= Pump Capacity

B= Tank Valve Capacity

A= .100 B= .100

TANK ON	PUMP ON	LEVEL	TEMP	LEVEL ALARM	TEMP ALARM
0.	0.	4.5	99.	0.	0.
1.	0.	4.4	99.	0.	0.
1.	0.	4.3	99.	0.	0.
0.	0.	4.3	98.	0.	0.
1.	0.	4.2	98.	0.	0.
1.	0.	4.1	98.	0.	0.
0.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
0.	0.	4.0	98.	0.	0.
1.	1.	3.9	97.	0.	0.
1.	1.	3.9	98.	0.	0.
0.	0.	4.0	99.	0.	0.
1.	1.	3.9	99.	0.	0.
0.	0.	4.0	100.	0.	0.
1.	1.	3.9	100.	0.	0.
1.	1.	3.9	101.	0.	0.
0.	0.	4.0	102.	0.	0.
1.	1.	3.9	101.	0.	0.
0.	0.	4.0	102.	0.	0.
0.	0.	4.0	102.	0.	0.
1.	1.	3.9	102.	0.	0.
1.	1.	3.9	103.	0.	0.
0.	0.	4.0	104.	0.	0.
1.	1.	3.9	103.	0.	0.
0.	0.	4.0	104.	0.	0.
1.	1.	3.9	104.	0.	0.
0.	0.	4.0	105.	0.	0.
0.	0.	4.0	105.	0.	0.
1.	1.	3.9	104.	0.	0.
1.	1.	3.9	105.	0.	0.

1.	1.	3.9	106.	0.	0.
0.	0.	4.0	107.	0.	0.
0.	0.	4.0	107.	0.	0.
1.	1.	3.9	107.	0.	0.
0.	0.	4.0	108.	0.	0.
0.	0.	4.0	107.	0.	0.
0.	0.	4.0	107.	0.	0.
1.	1.	3.9	106.	0.	0.
1.	1.	3.9	107.	0.	0.
0.	0.	4.0	108.	0.	0.
1.	1.	3.9	108.	0.	0.
1.	1.	3.9	109.	0.	0.
1.	1.	3.9	110.	0.	0.
0.	0.	4.0	111.	0.	0.
0.	0.	4.0	111.	0.	0.
0.	0.	4.0	110.	0.	0.
0.	0.	4.0	110.	0.	0.
1.	1.	3.9	109.	0.	0.
1.	1.	3.9	110.	0.	0.
1.	1.	3.9	111.	0.	0.
0.	0.	4.0	112.	0.	0.
0.	0.	4.0	112.	0.	0.
0.	0.	4.0	112.	0.	0.
0.	0.	4.0	111.	0.	0.
1.	1.	3.9	111.	0.	0.
0.	0.	4.0	112.	0.	0.
1.	1.	3.9	111.	0.	0.
1.	1.	3.9	112.	0.	0.
0.	0.	4.0	113.	0.	0.
0.	0.	4.0	113.	0.	0.
1.	1.	3.9	112.	0.	0.
1.	1.	3.9	113.	0.	0.
1.	1.	3.9	114.	0.	0.
1.	1.	3.9	115.	0.	0.

0.	0.	4.0	116.	0.	0.
0.	0.	4.0	116.	0.	0.
0.	0.	4.0	116.	0.	0.
0.	0.	4.0	115.	0.	0.
0.	0.	4.0	115.	0.	0.
0.	0.	4.0	114.	0.	0.
1.	1.	3.9	114.	0.	0.
1.	1.	3.9	115.	0.	0.
1.	1.	3.9	116.	0.	0.
0.	0.	4.0	117.	0.	0.
1.	1.	3.9	116.	0.	0.
0.	0.	4.0	117.	0.	0.
0.	0.	4.0	117.	0.	0.
1.	1.	3.9	116.	0.	0.
0.	0.	4.0	117.	0.	0.
0.	0.	4.0	117.	0.	0.
0.	0.	4.0	116.	0.	0.
0.	0.	4.0	116.	0.	0.
0.	0.	4.0	115.	0.	0.
0.	0.	4.0	115.	0.	0.
0.	0.	4.0	114.	0.	0.
1.	1.	3.9	114.	0.	0.
0.	0.	4.0	115.	0.	0.
0.	0.	4.0	115.	0.	0.
1.	1.	3.9	114.	0.	0.
0.	0.	4.0	115.	0.	0.
1.	1.	3.9	115.	0.	0.
0.	0.	4.0	116.	0.	0.
1.	1.	3.9	115.	0.	0.
0.	0.	4.0	116.	0.	0.
1.	1.	3.9	116.	0.	0.
0.	0.	4.0	117.	0.	0.
0.	0.	4.0	116.	0.	0.
1.	1.	3.9	116.	0.	0.

A= .150 B= .100

TANK ON	PUMP ON	LEVEL	TEMP	LEVEL ALARM	TEMP ALARM
0.	0.	4.5	99.	0.	0.
1.	0.	4.4	99.	0.	0.
1.	0.	4.3	99.	0.	0.
0.	0.	4.3	98.	0.	0.
1.	0.	4.2	98.	0.	0.
0.	0.	4.2	98.	0.	0.
1.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
0.	0.	4.0	98.	0.	0.
1.	1.	3.9	98.	0.	0.
1.	0.	4.0	99.	0.	0.
1.	1.	3.9	98.	0.	0.
0.	0.	4.0	99.	0.	0.
0.	0.	4.0	99.	0.	0.
0.	0.	4.0	99.	0.	0.
0.	0.	4.0	98.	0.	0.
1.	1.	3.9	98.	0.	0.
0.	0.	4.1	99.	0.	0.
1.	0.	4.0	99.	0.	0.
1.	1.	3.9	99.	0.	0.
1.	1.	3.9	100.	0.	0.
0.	0.	4.1	101.	0.	0.
0.	0.	4.1	100.	0.	0.
1.	0.	4.0	100.	0.	0.
1.	1.	3.9	100.	0.	0.
0.	0.	4.0	101.	0.	0.
1.	1.	3.9	100.	0.	0.
0.	0.	4.1	101.	0.	0.
0.	0.	4.1	101.	0.	0.
0.	0.	4.1	101.	0.	0.
1.	0.	4.0	100.	0.	0.

0.	0.	4.0	100.	0.	0.
1.	1.	3.9	100.	0.	0.
0.	0.	4.0	101.	0.	0.
1.	1.	3.9	100.	0.	0.
0.	0.	4.1	101.	0.	0.
0.	0.	4.1	101.	0.	0.
1.	0.	4.0	101.	0.	0.
0.	0.	4.0	101.	0.	0.
1.	1.	3.9	100.	0.	0.
0.	0.	4.0	101.	0.	0.
1.	1.	3.9	101.	0.	0.
1.	0.	4.0	102.	0.	0.
1.	1.	3.9	102.	0.	0.
1.	1.	3.9	103.	0.	0.
0.	0.	4.1	104.	0.	0.
0.	0.	4.1	103.	0.	0.
0.	0.	4.1	103.	0.	0.
0.	0.	4.1	103.	0.	0.
1.	0.	4.0	102.	0.	0.
0.	0.	4.0	102.	0.	0.
0.	0.	4.0	102.	0.	0.
1.	1.	3.9	101.	0.	0.
0.	0.	4.0	102.	0.	0.
0.	0.	4.0	102.	0.	0.
1.	1.	3.9	102.	0.	0.
1.	0.	4.0	103.	0.	0.
1.	1.	3.9	102.	0.	0.
1.	1.	3.9	103.	0.	0.
0.	0.	4.1	104.	0.	0.
0.	0.	4.1	104.	0.	0.
1.	0.	4.0	104.	0.	0.
0.	0.	4.0	103.	0.	0.
1.	1.	3.9	103.	0.	0.
0.	0.	4.0	104.	0.	0.

0.	0.	4.0	104.	0.	0.
1.	1.	3.9	103.	0.	0.
0.	0.	4.1	104.	0.	0.
0.	0.	4.1	104.	0.	0.
0.	0.	4.1	104.	0.	0.
0.	0.	4.1	103.	0.	0.
1.	0.	4.0	103.	0.	0.
0.	0.	4.0	103.	0.	0.
0.	0.	4.0	102.	0.	0.
0.	0.	4.0	102.	0.	0.
1.	1.	3.9	102.	0.	0.
0.	0.	4.0	103.	0.	0.
1.	1.	3.9	102.	0.	0.
0.	0.	4.1	103.	0.	0.
1.	0.	4.0	103.	0.	0.
0.	0.	4.0	103.	0.	0.
1.	1.	3.9	102.	0.	0.
1.	1.	3.9	103.	0.	0.
1.	0.	4.0	104.	0.	0.
1.	1.	3.9	104.	0.	0.
1.	1.	3.9	105.	0.	0.
0.	0.	4.1	106.	0.	0.
1.	0.	4.0	105.	0.	0.
0.	0.	4.0	105.	0.	0.
1.	1.	3.9	105.	0.	0.
0.	0.	4.0	106.	0.	0.
0.	0.	4.0	105.	0.	0.
0.	0.	4.0	105.	0.	0.
0.	0.	4.0	105.	0.	0.
0.	0.	4.0	104.	0.	0.
0.	0.	4.0	104.	0.	0.
1.	1.	3.9	104.	0.	0.
0.	0.	4.1	105.	0.	0.
1.	0.	4.0	104.	0.	0.

A= .200 B= .100

TANK ON	PUMP ON	LEVEL	TEMP	LEVEL ALARM	TEMP ALARM
0.	0.	4.5	99.	0.	0.
1.	0.	4.4	99.	0.	0.
1.	0.	4.3	99.	0.	0.
0.	0.	4.3	98.	0.	0.
1.	0.	4.2	98.	0.	0.
1.	0.	4.1	98.	0.	0.
0.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
1.	0.	4.0	98.	0.	0.
0.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
0.	0.	4.1	98.	0.	0.
0.	0.	4.1	98.	0.	0.
0.	0.	4.1	98.	0.	0.
1.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
1.	0.	4.0	98.	0.	0.
0.	0.	4.0	98.	0.	0.
0.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
1.	1.	3.9	96.	0.	0.
0.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
0.	0.	4.0	96.	0.	0.
0.	0.	4.0	96.	0.	0.
0.	0.	4.0	96.	0.	0.
0.	0.	4.0	96.	0.	0.

0.	0.	4.0	95.	0.	0.
1.	1.	3.9	95.	0.	0.
0.	0.	4.1	96.	0.	0.
1.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
0.	0.	4.1	97.	0.	0.
1.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
0.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
1.	0.	4.0	98.	0.	0.
0.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
1.	0.	4.0	98.	0.	0.
0.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
1.	0.	4.0	98.	0.	0.
0.	0.	4.0	98.	0.	0.
0.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
0.	0.	4.1	98.	0.	0.
0.	0.	4.1	98.	0.	0.
0.	0.	4.1	97.	0.	0.
0.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
1.	1.	3.9	96.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.

1.	0.	4.0	98.	0.	0.
0.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
0.	0.	4.1	98.	0.	0.
0.	0.	4.1	97.	0.	0.
0.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
0.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
1.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
1.	0.	4.0	98.	0.	0.
1.	1.	3.9	97.	0.	0.
1.	0.	4.0	98.	0.	0.
0.	0.	4.0	98.	0.	0.
0.	0.	4.0	98.	0.	0.
1.	1.	3.9	98.	0.	0.
0.	0.	4.1	99.	0.	0.
0.	0.	4.1	98.	0.	0.
0.	0.	4.1	98.	0.	0.
0.	0.	4.1	98.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
1.	0.	4.0	98.	0.	0.
1.	1.	3.9	98.	0.	0.
0.	0.	4.1	99.	0.	0.
0.	0.	4.1	98.	0.	0.
1.	0.	4.0	98.	0.	0.
1.	1.	3.9	98.	0.	0.
1.	0.	4.0	99.	0.	0.
1.	1.	3.9	98.	0.	0.

A= .250 B= .100

TANK ON	PUMP ON	LEVEL	TEMP	LEVEL ALARM	TEMP ALARM
0.	0.	4.5	99.	0.	0.
1.	0.	4.4	99.	0.	0.
0.	0.	4.4	99.	0.	0.
1.	0.	4.3	98.	0.	0.
1.	0.	4.2	98.	0.	0.
0.	0.	4.2	98.	0.	0.
1.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
0.	0.	4.1	98.	0.	0.
0.	0.	4.1	98.	0.	0.
0.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
0.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
1.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
1.	1.	3.9	97.	0.	0.
0.	0.	4.2	98.	0.	0.
1.	0.	4.1	97.	0.	0.
0.	0.	4.1	97.	0.	0.
0.	0.	4.1	97.	0.	0.
0.	0.	4.1	97.	0.	0.
0.	0.	4.1	96.	0.	0.
0.	0.	4.1	96.	0.	0.
0.	0.	4.1	96.	0.	0.

1.	0.	4.0	95.	0.	0.
1.	1.	3.9	95.	0.	0.
1.	0.	4.0	96.	0.	0.
0.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
1.	0.	4.1	97.	0.	0.
0.	0.	4.1	96.	0.	0.
1.	0.	4.0	96.	0.	0.
0.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
1.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	96.	0.	0.
0.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
0.	0.	4.0	96.	0.	0.
0.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
0.	0.	4.2	97.	0.	0.
0.	0.	4.2	97.	0.	0.
1.	0.	4.1	96.	0.	0.
0.	0.	4.1	96.	0.	0.
1.	0.	4.0	96.	0.	0.
0.	0.	4.0	96.	0.	0.
1.	1.	3.9	95.	0.	0.
0.	0.	4.1	96.	0.	0.
1.	0.	4.0	96.	0.	0.
0.	0.	4.0	96.	0.	0.
0.	0.	4.0	95.	0.	0.

0.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
1.	1.	3.9	95.	0.	0.
1.	0.	4.1	96.	0.	0.
0.	0.	4.1	95.	0.	0.
1.	0.	4.0	95.	0.	0.
1.	1.	3.9	95.	0.	0.
0.	0.	4.1	96.	0.	0.
0.	0.	4.1	96.	0.	0.
0.	0.	4.1	95.	0.	0.
1.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
0.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.2	95.	0.	0.
1.	0.	4.1	95.	0.	0.
1.	0.	4.0	95.	0.	0.
0.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
1.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
1.	1.	3.9	95.	0.	0.
1.	0.	4.1	96.	0.	0.
1.	0.	4.0	95.	0.	0.
1.	1.	3.9	95.	0.	0.
0.	0.	4.1	96.	0.	0.
0.	0.	4.1	96.	0.	0.
1.	0.	4.0	96.	0.	0.
1.	1.	3.9	95.	0.	0.
0.	0.	4.2	96.	0.	0.
1.	0.	4.1	96.	0.	0.
1.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.

A= .300 B= .100

TANK ON	PUMP ON	LEVEL	TEMP	LEVEL ALARM	TEMP ALARM
0.	0.	4.5	99.	0.	0.
1.	0.	4.4	99.	0.	0.
0.	0.	4.4	99.	0.	0.
1.	0.	4.3	98.	0.	0.
0.	0.	4.3	98.	0.	0.
1.	0.	4.2	98.	0.	0.
0.	0.	4.2	97.	0.	0.
1.	0.	4.1	97.	0.	0.
0.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
0.	0.	4.2	97.	0.	0.
0.	0.	4.2	97.	0.	0.
0.	0.	4.2	97.	0.	0.
0.	0.	4.2	96.	0.	0.
0.	0.	4.2	96.	0.	0.
1.	0.	4.1	96.	0.	0.
0.	0.	4.1	95.	0.	0.
1.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
1.	1.	3.9	95.	0.	0.
1.	0.	4.1	96.	0.	0.
1.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
1.	1.	3.9	95.	0.	0.
1.	0.	4.1	96.	0.	0.
1.	0.	4.0	96.	0.	0.
0.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
1.	1.	3.9	95.	0.	0.
0.	0.	4.2	96.	0.	0.

1.	0.	4.1	96.	0.	0.
1.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
1.	1.	3.9	95.	0.	0.
0.	0.	4.2	96.	0.	0.
0.	0.	4.2	95.	0.	0.
1.	0.	4.1	95.	0.	0.
1.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
0.	0.	4.0	94.	0.	0.
0.	0.	4.0	94.	0.	0.
0.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
1.	0.	4.1	95.	0.	0.
1.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.2	95.	0.	0.
1.	0.	4.1	95.	0.	0.
0.	0.	4.1	95.	0.	0.
1.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.2	95.	0.	0.
1.	0.	4.1	95.	0.	0.
0.	0.	4.1	95.	0.	0.
0.	0.	4.1	94.	0.	0.
1.	0.	4.0	94.	0.	0.
0.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
1.	0.	4.1	95.	0.	0.
0.	0.	4.1	94.	0.	0.
1.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.2	95.	0.	0.

1.	0.	4.1	95.	0.	0.
1.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
1.	0.	4.1	95.	0.	0.
0.	0.	4.1	95.	0.	0.
1.	0.	4.0	95.	0.	0.
0.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.2	95.	0.	0.
1.	0.	4.1	95.	0.	0.
1.	0.	4.0	95.	0.	0.
0.	0.	4.0	94.	0.	0.
0.	0.	4.0	94.	0.	0.
0.	0.	4.0	94.	0.	0.
0.	0.	4.0	94.	0.	0.
1.	1.	3.9	93.	0.	0.
0.	0.	4.2	94.	0.	0.
0.	0.	4.2	94.	0.	0.
1.	0.	4.1	94.	0.	0.
1.	0.	4.0	94.	0.	0.
0.	0.	4.0	93.	0.	0.
0.	0.	4.0	93.	0.	0.
0.	0.	4.0	93.	0.	0.
0.	0.	4.0	93.	0.	0.
0.	0.	4.0	92.	0.	0.
1.	1.	3.9	92.	0.	0.
0.	0.	4.2	93.	0.	0.
1.	0.	4.1	93.	0.	0.
1.	0.	4.0	93.	0.	0.
1.	1.	3.9	92.	0.	0.
0.	0.	4.2	93.	0.	0.
1.	0.	4.1	93.	0.	0.
0.	0.	4.1	93.	0.	0.
1.	0.	4.0	93.	0.	0.

A= .350 B= .100

TANK ON	PUMP ON	LEVEL	TEMP	LEVEL ALARM	TEMP ALARM
0.	0.	4.5	99.	0.	0.
1.	0.	4.4	99.	0.	0.
0.	0.	4.4	99.	0.	0.
0.	0.	4.4	98.	0.	0.
1.	0.	4.3	98.	0.	0.
1.	0.	4.2	98.	0.	0.
0.	0.	4.2	97.	0.	0.
0.	0.	4.2	97.	0.	0.
0.	0.	4.2	97.	0.	0.
1.	0.	4.1	97.	0.	0.
1.	0.	4.0	96.	0.	0.
0.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
1.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
1.	1.	3.9	96.	0.	0.
0.	0.	4.3	97.	0.	0.
0.	0.	4.3	97.	0.	0.
0.	0.	4.3	97.	0.	0.
0.	0.	4.3	96.	0.	0.
0.	0.	4.3	96.	0.	0.
0.	0.	4.3	96.	0.	0.
1.	0.	4.2	96.	0.	0.
1.	0.	4.1	95.	0.	0.
1.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
0.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.2	95.	0.	0.
0.	0.	4.2	95.	0.	0.
1.	0.	4.1	95.	0.	0.

0.	0.	4.1	94.	0.	0.
1.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
1.	0.	4.2	95.	0.	0.
1.	0.	4.1	95.	0.	0.
1.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.2	95.	0.	0.
1.	0.	4.1	95.	0.	0.
0.	0.	4.1	95.	0.	0.
1.	0.	4.0	94.	0.	0.
0.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.3	95.	0.	0.
1.	0.	4.2	95.	0.	0.
1.	0.	4.1	94.	0.	0.
1.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.2	95.	0.	0.
1.	0.	4.1	95.	0.	0.
1.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.3	95.	0.	0.
1.	0.	4.2	95.	0.	0.
0.	0.	4.2	95.	0.	0.
1.	0.	4.1	94.	0.	0.
0.	0.	4.1	94.	0.	0.
1.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
1.	0.	4.1	95.	0.	0.
1.	0.	4.0	94.	0.	0.
0.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
1.	0.	4.2	95.	0.	0.

0.	0.	4.2	95.	0.	0.
1.	0.	4.1	94.	0.	0.
0.	0.	4.1	94.	0.	0.
1.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.2	95.	0.	0.
1.	0.	4.1	94.	0.	0.
0.	0.	4.1	94.	0.	0.
0.	0.	4.1	94.	0.	0.
1.	0.	4.0	94.	0.	0.
0.	0.	4.0	93.	0.	0.
0.	0.	4.0	93.	0.	0.
1.	1.	3.9	93.	0.	0.
1.	0.	4.2	94.	0.	0.
0.	0.	4.2	94.	0.	0.
0.	0.	4.2	93.	0.	0.
0.	0.	4.2	93.	0.	0.
0.	0.	4.2	93.	0.	0.
0.	0.	4.2	93.	0.	0.
0.	0.	4.2	92.	0.	0.
1.	0.	4.1	92.	0.	0.
0.	0.	4.1	92.	0.	0.
0.	0.	4.1	92.	0.	0.
1.	0.	4.0	92.	0.	0.
1.	1.	3.9	91.	0.	0.
1.	0.	4.1	92.	0.	0.
1.	0.	4.0	92.	0.	0.
0.	0.	4.0	92.	0.	0.
0.	0.	4.0	92.	0.	0.
0.	0.	4.0	91.	0.	0.
1.	1.	3.9	91.	0.	0.
1.	0.	4.2	92.	0.	0.
0.	0.	4.2	92.	0.	0.
1.	0.	4.1	92.	0.	0.

A= .400 B= .100

TANK ON	PUMP ON	LEVEL	TEMP	LEVEL ALARM	TEMP ALARM
0.	0.	4.5	99.	0.	0.
0.	0.	4.5	99.	0.	0.
0.	0.	4.5	99.	0.	0.
1.	0.	4.4	98.	0.	0.
0.	0.	4.4	98.	0.	0.
0.	0.	4.4	98.	0.	0.
1.	0.	4.3	97.	0.	0.
1.	0.	4.2	97.	0.	0.
1.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
1.	0.	4.2	97.	0.	0.
0.	0.	4.2	97.	0.	0.
0.	0.	4.2	97.	0.	0.
0.	0.	4.2	96.	0.	0.
1.	0.	4.1	96.	0.	0.
0.	0.	4.1	96.	0.	0.
0.	0.	4.1	95.	0.	0.
1.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
0.	0.	4.0	95.	0.	0.
0.	0.	4.0	94.	0.	0.
0.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.3	95.	0.	0.
1.	0.	4.2	95.	0.	0.
1.	0.	4.1	94.	0.	0.
1.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.3	95.	0.	0.
1.	0.	4.2	95.	0.	0.

0.	0.	4.2	94.	0.	0.
1.	0.	4.1	94.	0.	0.
1.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
1.	0.	4.2	95.	0.	0.
0.	0.	4.2	94.	0.	0.
0.	0.	4.2	94.	0.	0.
1.	0.	4.1	94.	0.	0.
0.	0.	4.1	94.	0.	0.
1.	0.	4.0	93.	0.	0.
0.	0.	4.0	93.	0.	0.
1.	1.	3.9	93.	0.	0.
1.	0.	4.2	94.	0.	0.
0.	0.	4.2	94.	0.	0.
0.	0.	4.2	94.	0.	0.
0.	0.	4.2	93.	0.	0.
0.	0.	4.2	93.	0.	0.
1.	0.	4.1	93.	0.	0.
0.	0.	4.1	93.	0.	0.
1.	0.	4.0	92.	0.	0.
0.	0.	4.0	92.	0.	0.
1.	1.	3.9	92.	0.	0.
0.	0.	4.3	93.	0.	0.
0.	0.	4.3	93.	0.	0.
0.	0.	4.3	92.	0.	0.
1.	0.	4.2	92.	0.	0.
1.	0.	4.1	92.	0.	0.
1.	0.	4.0	92.	0.	0.
0.	0.	4.0	92.	0.	0.
1.	1.	3.9	91.	0.	0.
1.	0.	4.2	92.	0.	0.
0.	0.	4.2	92.	0.	0.
0.	0.	4.2	92.	0.	0.
1.	0.	4.1	92.	0.	0.

1.	0.	4.0	91.	0.	0.
1.	1.	3.9	91.	0.	0.
0.	0.	4.3	92.	0.	0.
1.	0.	4.2	92.	0.	0.
1.	0.	4.1	92.	0.	0.
1.	0.	4.0	91.	0.	0.
0.	0.	4.0	91.	0.	0.
0.	0.	4.0	91.	0.	0.
1.	1.	3.9	91.	0.	0.
0.	0.	4.3	92.	0.	0.
0.	0.	4.3	92.	0.	0.
0.	0.	4.3	91.	0.	0.
0.	0.	4.3	91.	0.	0.
1.	0.	4.2	91.	0.	0.
0.	0.	4.2	91.	0.	0.
0.	0.	4.2	91.	0.	0.
0.	0.	4.2	90.	0.	0.
1.	0.	4.1	90.	0.	0.
1.	0.	4.0	90.	0.	0.
0.	0.	4.0	90.	0.	0.
1.	1.	3.9	90.	0.	0.
1.	0.	4.2	91.	0.	0.
1.	0.	4.1	90.	0.	0.
0.	0.	4.1	90.	0.	0.
0.	0.	4.1	90.	0.	0.
1.	0.	4.0	90.	0.	0.
0.	0.	4.0	89.	0.	0.
1.	1.	3.9	89.	0.	0.
0.	0.	4.3	90.	0.	0.
1.	0.	4.2	90.	0.	0.
1.	0.	4.1	90.	0.	0.
0.	0.	4.1	90.	0.	0.
0.	0.	4.1	89.	0.	0.
0.	0.	4.1	89.	0.	0.

A= .450 B= .100

TANK ON	PUMP ON	LEVEL	TEMP	LEVEL ALARM	TEMP ALARM
0.	0.	4.5	99.	0.	0.
0.	0.	4.5	99.	0.	0.
1.	0.	4.4	99.	0.	0.
0.	0.	4.4	98.	0.	0.
1.	0.	4.3	98.	0.	0.
0.	0.	4.3	98.	0.	0.
1.	0.	4.2	97.	0.	0.
1.	0.	4.1	97.	0.	0.
1.	0.	4.0	97.	0.	0.
0.	0.	4.0	97.	0.	0.
1.	1.	3.9	96.	0.	0.
1.	0.	4.2	97.	0.	0.
0.	0.	4.2	97.	0.	0.
0.	0.	4.2	97.	0.	0.
1.	0.	4.1	97.	0.	0.
1.	0.	4.0	96.	0.	0.
0.	0.	4.0	96.	0.	0.
1.	1.	3.9	96.	0.	0.
0.	0.	4.4	97.	0.	0.
1.	0.	4.3	96.	0.	0.
0.	0.	4.3	96.	0.	0.
0.	0.	4.3	96.	0.	0.
0.	0.	4.3	96.	0.	0.
1.	0.	4.2	95.	0.	0.
1.	0.	4.1	95.	0.	0.
0.	0.	4.1	95.	0.	0.
1.	0.	4.0	95.	0.	0.
0.	0.	4.0	94.	0.	0.
0.	0.	4.0	94.	0.	0.
0.	0.	4.0	94.	0.	0.
1.	1.	3.9	94.	0.	0.
0.	0.	4.3	95.	0.	0.

0.	0.	4.3	94.	0.	0.
0.	0.	4.3	94.	0.	0.
0.	0.	4.3	94.	0.	0.
0.	0.	4.3	94.	0.	0.
1.	0.	4.2	93.	0.	0.
1.	0.	4.1	93.	0.	0.
1.	0.	4.0	93.	0.	0.
1.	1.	3.9	93.	0.	0.
1.	0.	4.3	94.	0.	0.
0.	0.	4.3	93.	0.	0.
1.	0.	4.2	93.	0.	0.
1.	0.	4.1	93.	0.	0.
0.	0.	4.1	93.	0.	0.
0.	0.	4.1	93.	0.	0.
1.	0.	4.0	92.	0.	0.
0.	0.	4.0	92.	0.	0.
0.	0.	4.0	92.	0.	0.
0.	0.	4.0	92.	0.	0.
1.	1.	3.9	91.	0.	0.
1.	0.	4.2	92.	0.	0.
0.	0.	4.2	92.	0.	0.
1.	0.	4.1	92.	0.	0.
0.	0.	4.1	92.	0.	0.
1.	0.	4.0	91.	0.	0.
1.	1.	3.9	91.	0.	0.
1.	0.	4.3	92.	0.	0.
0.	0.	4.3	92.	0.	0.
0.	0.	4.3	92.	0.	0.
0.	0.	4.3	92.	0.	0.
1.	0.	4.2	91.	0.	0.
1.	0.	4.1	91.	0.	0.
0.	0.	4.1	91.	0.	0.
0.	0.	4.1	91.	0.	0.
1.	0.	4.0	91.	0.	0.

0.	0.	4.0	90.	0.	0.
1.	1.	3.9	90.	0.	0.
1.	0.	4.2	91.	0.	0.
1.	0.	4.1	91.	0.	0.
0.	0.	4.1	91.	0.	0.
0.	0.	4.1	90.	0.	0.
1.	0.	4.0	90.	0.	0.
0.	0.	4.0	90.	0.	0.
1.	1.	3.9	90.	0.	0.
1.	0.	4.3	91.	0.	0.
1.	0.	4.2	91.	0.	0.
0.	0.	4.2	90.	0.	0.
0.	0.	4.2	90.	0.	0.
1.	0.	4.1	90.	0.	0.
1.	0.	4.0	90.	0.	0.
1.	1.	3.9	90.	0.	0.
0.	0.	4.3	91.	0.	0.
1.	0.	4.2	90.	0.	0.
1.	0.	4.1	90.	0.	0.
0.	0.	4.1	90.	0.	0.
0.	0.	4.1	90.	0.	0.
1.	0.	4.0	90.	0.	0.
0.	0.	4.0	89.	0.	0.
1.	1.	3.9	89.	0.	0.
1.	0.	4.3	90.	0.	0.
1.	0.	4.2	90.	0.	0.
1.	0.	4.1	90.	0.	0.
0.	0.	4.1	90.	0.	0.
0.	0.	4.1	89.	0.	0.
0.	0.	4.1	89.	0.	0.
0.	0.	4.1	89.	0.	0.
0.	0.	4.1	89.	0.	0.
0.	0.	4.1	89.	0.	0.
0.	0.	4.1	88.	0.	0.

A= .500 B= .100

TANK ON	PUMP ON	LEVEL	TEMP	LEVEL ALARM	TEMP ALARM
0.	0.	4.5	99.	0.	0.
1.	0.	4.4	99.	0.	0.
0.	0.	4.4	99.	0.	0.
0.	0.	4.4	98.	0.	0.
0.	0.	4.4	98.	0.	0.
0.	0.	4.4	98.	0.	0.
0.	0.	4.4	97.	0.	0.
0.	0.	4.4	97.	0.	0.
0.	0.	4.4	97.	0.	0.
0.	0.	4.4	97.	0.	0.
0.	0.	4.4	96.	0.	0.
0.	0.	4.4	96.	0.	0.
0.	0.	4.4	96.	0.	0.
1.	0.	4.3	96.	0.	0.
0.	0.	4.3	95.	0.	0.
0.	0.	4.3	95.	0.	0.
1.	0.	4.2	95.	0.	0.
0.	0.	4.2	95.	0.	0.
0.	0.	4.2	94.	0.	0.
0.	0.	4.2	94.	0.	0.
0.	0.	4.2	94.	0.	0.
0.	0.	4.2	94.	0.	0.
1.	0.	4.1	93.	0.	0.
0.	0.	4.1	93.	0.	0.
0.	0.	4.1	93.	0.	0.
1.	0.	4.0	93.	0.	0.
1.	1.	3.9	92.	0.	0.
1.	0.	4.3	93.	0.	0.
0.	0.	4.3	93.	0.	0.
0.	0.	4.3	93.	0.	0.
0.	0.	4.3	93.	0.	0.
0.	0.	4.3	92.	0.	0.

1.	0.	4.2	92.	0.	0.
0.	0.	4.2	92.	0.	0.
0.	0.	4.2	92.	0.	0.
1.	0.	4.1	92.	0.	0.
1.	0.	4.0	91.	0.	0.
1.	1.	3.9	91.	0.	0.
0.	0.	4.4	92.	0.	0.
1.	0.	4.3	92.	0.	0.
0.	0.	4.3	92.	0.	0.
1.	0.	4.2	91.	0.	0.
1.	0.	4.1	91.	0.	0.
0.	0.	4.1	91.	0.	0.
0.	0.	4.1	91.	0.	0.
1.	0.	4.0	91.	0.	0.
1.	1.	3.9	90.	0.	0.
0.	0.	4.4	91.	0.	0.
1.	0.	4.3	91.	0.	0.
0.	0.	4.3	91.	0.	0.
0.	0.	4.3	91.	0.	0.
0.	0.	4.3	90.	0.	0.
1.	0.	4.2	90.	0.	0.
1.	0.	4.1	90.	0.	0.
0.	0.	4.1	90.	0.	0.
0.	0.	4.1	90.	0.	0.
1.	0.	4.0	89.	0.	0.
1.	1.	3.9	89.	0.	0.
1.	0.	4.3	90.	0.	0.
0.	0.	4.3	90.	0.	0.
0.	0.	4.3	90.	0.	0.
0.	0.	4.3	90.	0.	0.
1.	0.	4.2	89.	0.	0.
0.	0.	4.2	89.	0.	0.
0.	0.	4.2	89.	0.	0.
1.	0.	4.1	89.	0.	0.

0.	0.	4.1	89.	0.	0.
1.	0.	4.0	88.	0.	0.
1.	1.	3.9	88.	0.	0.
1.	0.	4.3	89.	0.	0.
0.	0.	4.3	89.	0.	0.
1.	0.	4.2	89.	0.	0.
1.	0.	4.1	89.	0.	0.
0.	0.	4.1	88.	0.	0.
0.	0.	4.1	88.	0.	0.
0.	0.	4.1	88.	0.	0.
1.	0.	4.0	88.	0.	0.
0.	0.	4.0	88.	0.	0.
0.	0.	4.0	88.	0.	0.
1.	1.	3.9	87.	0.	0.
0.	0.	4.4	88.	0.	0.
1.	0.	4.3	88.	0.	0.
1.	0.	4.2	88.	0.	0.
1.	0.	4.1	88.	0.	0.
1.	0.	4.0	88.	0.	0.
0.	0.	4.0	87.	0.	0.
0.	0.	4.0	87.	0.	0.
0.	0.	4.0	87.	0.	0.
1.	1.	3.9	87.	0.	0.
0.	0.	4.4	88.	0.	0.
1.	0.	4.3	88.	0.	0.
1.	0.	4.2	88.	0.	0.
0.	0.	4.2	87.	0.	0.
0.	0.	4.2	87.	0.	0.
0.	0.	4.2	87.	0.	0.
1.	0.	4.1	87.	0.	0.
1.	0.	4.0	87.	0.	0.
1.	1.	3.9	86.	0.	0.
0.	0.	4.4	87.	0.	0.
1.	0.	4.3	87.	0.	0.

BIBLIOGRAPHY

1 Bowman, Edward H. and Robert B. Fetter, Analysis for Production Management: Richard D. Irwin, Inc., 1961.

2 Brown, Robert C., Statistical Forecasting For Inventory Control: McGraw-Hill, 1959.

3 Freund, John E., Mathematical Statistics, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1963.

4 Hull, T. E. and A. R. Dobell, "Random Number Generators". SIAM Review, Vol. 4, #3, July, 1962.

5 International Business Machines, IBM 1620 Fortran II Programming System, Reference Manual, File No. 1620-25: IBM, 1962.

6 Kirk, H. W., "Use of Decision Tables in Computer Programming". Communications of the ACM, January, 1965.

7 Leeson, Daniel N. and Donald L. Dimitry, Basic Programming Concepts and the IBM 1620 Computer, Holt, Rinehart & Winston, 1962.

8 Martin, E. Wainright Jr., Electronic Data Processing, An Introduction, Irwin, 1961.

9 Pollack, Soloman L., "Conversion of Limited Entry Decision Tables to Computer Programs". Communications of the ACM November, 1965.

10 Schmidt, D. T. and T. F. Kavanaugh, "Using Decision Structure Tables". Datamation, February, 1964.

11 Sprague, V. G., "Letters to the Editor-on Storage Space of Decision Tables". Communications of the ACM, May, 1966.

12 Starr, Martin K. and David W. Miller, Inventory Control: Theory and Practice, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1962.

13 Veinott, Cyril G., "Programming Decision Tables in Fortran, Cobol or Algo". Communications of the ACM, January, 1966.

14 Wichlan, Daniel Jospeh, A Study of the Effect of Non-Normal Distribution Upon Simple Lenear Regression, Kansas State University: Thesis, 1966.

A SYSTEMS SIMULATOR PROGRAMMING LANGUAGE FOR THE IBM 1620 COMPUTER

by

THOMAS ALLEN WEBB III

B. S., Kansas State University, 1965

AN ABSTRACT OF A MASTER'S THESIS

Submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1966

The objective of this thesis was to develope a system for the IBM 1620 computer which would facilitate the simulation of systems. The Fortran II programming language was chosen as a basis for further study and implementation. Four methods for programming decision systems were studied and implemented in Fortran either as Fortran programs or as subroutines to Fortran. The four methods were each used to program a simulation and the results were compared. The four methods were: normal Fortran Programming (using a decision tree), an algorithm for minimum Fortran programming (improved decision tree), a decision tables programmed in Fortran, and decision tables used with Fortran object programs (using SPS subroutines).

The results indicate that shorter and probably faster programs can be written using the minimum decision tree programmed in Fortran for small to medium decision systems. However, for large decision systems, shorter and possible faster programs may be written if use is made of the SPS subroutines.