SH20-1924-1

Edward C.
Marburg
Adams

Installed User Program

APL Decision Table Processor (DTABL) Program Description/Operations Manual

Program Number: 5796-PJB

This manual provides instructions for the use of a Decision Table Translator and auxiliary routines that have been implemented to run under several APL systems. Decision tables can be entered, displayed, edited, saved, and compiled with DTABL. The results of compilation can be displayed in several formats - - "abstracts", COBOL, PL/I, ALGOL, and APL. APL programs can be generated in executable form. Programs in other languages can be written on a data set and be used as input to other language processors. The decision tables and object code produced from them can be saved within the APL system for future reference or manipulation in separate terminal sessions.

IBM

PROGRAMMING SERVICES PERIOD

During specified number of months immediately following initial availability of each licensed program, designated as the PROGRAMMING SERVICES PERIOD, the customer may submit documentation to a designated IBM location when he encounters a problem which his diagnosis indicates is caused by a licensed program error. During this period only, IBM through the program sponsor(s), will, without additional charge, respond to an error in the current unaltered release of the licensed program by issuing known error correction information to the customer reporting the problem and/or issuing corrected or notice availability of corrected code. However, IBM does not guarantee service results or represent or warrant that all errors will be corrected. Any onsite programming services or assistance will be provided at a charge.

WARRANTY

EACH LICENSED PROGRAM IS DISTRIBUTED ON AN 'AS IS' BASIS WITHOUT WARRANTY OF ANY KIND EITHER EXPRESS OR IMPLIED.

Second Edition (March 1977)

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Palo Alto Scientific Center, P.O. Box 10500, Palo Alto, California 94304. Attention: Mr. H. J. Myers.

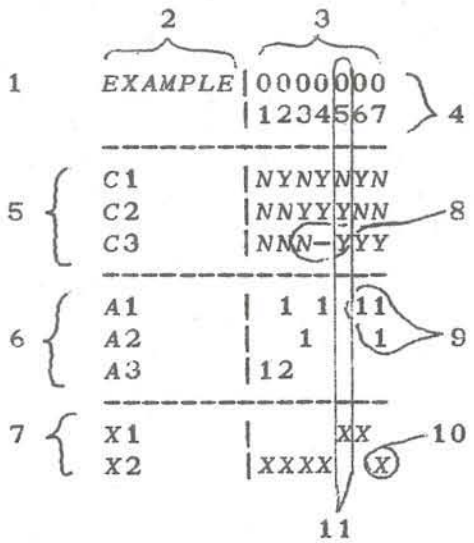
CONTENTS

ILLUSTRATIONS

INTRODUCTION AND DECISION TABLE CONCEPTS

This manual provides instructions for the use of a Decision Table Compiler and auxiliary routines that have been implemented to run under several APL systems (APL.SV, APL/CMS, and VS APL under CMS, VSPC or TSO). Decision tables can be entered, displayed, edited, saved, and compiled with *DTABL*. The results of compilation can be displayed in several formats -- "abstract", COBOL, PL/I, ALGOL, and APL. APL programs can be generated in executable form. Programs in other languages can be written on a data set and be used as input to other language processors. The decision tables and object code produced from them can be saved within the APL system for future reference or manipulation in separate terminal sessions.

Figure 1 is an example of a decision table with an indication of significant elements and terminology used in this manual.



1. Table header
2. Stub section
3. Entry section
4. Header section
5. Condition section
6. Action section
7. Exit section
8. Condition values (Y=yes, N=no, -=don't care)
9. Action sequence numbers
10. Exit indicators
11. Rule column number 5

Decision Table Anatomy
Figure 1.

The decision table allows specification of program logic, and the compiler converts this specification into an efficient procedure. The condition section of the table speci-

fies one or more conditions to be tested (but does not imply the order of testing). The results of the condition tests are recorded in the condition entries. This compiler handles only "Limited Entry Decision Tables" (LEDTs) -- i.e., those which restrict the values of the condition tests to yes (Y) or no (N). Therefore, a condition test is any expression (in some particular language) that yields only a single yes (true) or no (false) answer. Examples of condition tests are "A<B" (PL/I), ")/A=B" (APL), and "A is less than B" (English). Each column of the entry section is considered to be a rule and contains some (unique) combination of test values. These combinations must exhaust all possible combinations of test values or the table is incompletely specified. For example, if there are "n" binary (two-valued) tests, then there will be 2^n combinations in a valid table.

The action section of the table contains actions which will be performed (or not) depending upon the results of testing the conditions in the condition section. An action is any imperative expression in the language selected. Examples are, "A=B+C" (PL/I) and "Give the customer a discount" (English). Note that actions may not specify or imply flow control. (Branching or GO TO statements make no sense in a decision table context.) When a combination of test values matches those specified in a rule column, the actions in that column are performed in the order specified by the numbers indicated in that column. A blank opposite an action indicates that it is not to be performed. For example, in Figure 1 if all three conditions test to "no" (*NNN*) then rule number 1 is selected, causing the third action (*A3*) to be executed. If actions carry the same number, no implied order is required for their execution with respect to each other. There is also to be no order required between actions and tests or among tests.

The exit section of the table is analogous to the action section. However, exits specify the method of exiting the table. An exit can consist of the word *RETURN*, indicating the program generated from the table is a subroutine which will return to its caller. An exit can also indicate the name of some procedural code which can be reached by a branch from the decision table. (APL decision tables must always generate subroutines. However, this requirement does not extend to the other languages.) In the above example, the second exit (*X2*) is taken after execution of *A3* in rule 1. There must be exactly one exit selected for each rule column, and exits will always follow tests and actions when executed.

Central to the effectiveness of *DTABL* is the convention that no test or action in a given table will change the results of any test in the same table. This gives the compiler complete freedom to determine the order of tests in optimizing the object code. The ordering of actions with respect

to tests is therefore immaterial. The user can, however, specify the order of actions with respect to each other (as indicated earlier).

For convenience, rule columns may be combined by the user. The conditions allowing two columns to be combined are:

1) actions and exits in the two columns are identical
2) condition values are identical except one
3) the values in the differing condition are "Y" and "N".

To consolidate two columns into one, replace the value of the differing condition with "don't care" in one column, and discard the other column. This function can be performed by a program (*MERGE*) that is available with *DTABL* and described later in this manual. These two rules were applied to form column 4 of the decision table in Figure 1. Figure 2a shows column 4 and the two rule columns (4a and 4b) that it stands for.

	4	4a	4b
C1	Y	Y	Y
C2	Y	Y	Y
C3	-	Y	N
A1	1	1	1
A2			
A3			
X1			
X2	X	X	X

(a)

	Z	Za	Zb
C1	N	N	N
C2	Y	Y	Y
C3	-	-	-
C4	-	Y	N
A1	1	1	1
A2			
A3	1	1	1
X1	X	X	X

(b)

Rule Column Consolidation
Figure 2.

Figure 2b shows two columns (Za and Zb) from some other decision table and how they are consolidated into a single column (Z). Note that columns Za and Zb each stand for two columns, so column Z stands for four columns. (Note also that the order or actions A1 and A3 in the code produced from column Z will be immaterial because they both have the same selection number (1) in the entry positions.)

SIGN-ON PROCEDURE

There are a variety of terminals that can be attached to an APL system. The terminal discipline and sign-on procedures for each are described in [4-8] and are beyond the scope of this manual. You should be familiar with these procedures before attempting to use the APL system. For your convenience the APL keyboard is shown in Figure 3.

Keyboard Arrangement for APL
Figure 3.

Once APL has acknowledged your sign-on procedure you obtain your initial copy of *DTABL* from an APL public library by typing

)*LOAD* libno *DTABL*

followed by a carriage return or enter key. Note that you will always signal that your input is complete and ready to be acted upon by pressing the return (or enter) key. (In the above line "libno" stands for the library number -- such as 27 -- which is assigned to your installation's copy of the *DTABL* processor by the APL administrator.)

To sign off at the end of a session, type

)*OFF*

When the APL system responds with your usage statistics, you then turn off the power on the terminal. If you are using an acoustic coupler turn its power off and hang up the phone.

INITIAL TABLE ENTRY

Your first act in using the *DTABL* programs will be to enter a decision table. The initial entry of a decision table is aided by a program which you invoke by typing "*INPUT*" (followed by a carriage return or enter key). You will be prompted for each input needed to fill out each section of the table. (Note: If you are using an IBM 3270 terminal, enter "*IS*3270←1" before you enter *INPUT* so that prompting messages will appear at the proper place on your screen. At the end of this manual is a sample dialogue used to prepare and manipulate the table shown in Figure 1. The following comments give the specific rules for entering information.

The following paragraphs are headed by the prompting messages issued by the table-entry program. The paragraphs explain the significance of prompting. Figure 4, following these descriptions, shows how the table of Figure 1 was entered. A copy of this table is included with *DTABL*. In Figure 4 and examples subsequent to it the right-pointing arrows (→) indicate lines entered by the user. Other lines are produced by the *DTABL* programs.

ENTER TABLE HEADER

This table header will be placed in the top of the table and will be associated with all related output for identification. In generated programs (except APL) this header will be prefixed to all generated labels. Those programs that require both table and object code (*TESTGEN* and *PRINTHLL*) use this header to ensure compatibility of their inputs. In APL programs the table header becomes the function header. (Note that the header may not be empty. If it is the input process will be cancelled.)

ENTER CONDITION STUBS

Following this message you enter the condition stubs one line at a time. You signal the end of the condition section by entering an "empty line" (carriage return only). You can place any text that is appropriate to the target language in the decision table stubs. The stub can also contain "space" information refered to in the section COMPILING DECISION TABLES. (Note that you must enter at least one condition stub or the input process will be cancelled.)

ENTER ACTION STUBS

You respond to this message as you did to the conditions message, this time entering action stubs. An action stub can be any text suitable to the target language. You enter an empty line (carriage return only) to signal the end of the action stub secton. (Note that you must enter at least one action stub or the input process will be cancelled.)

ENTER EXIT STUBS

Your response is the same as with conditions and actions, but you enter exit stubs. Exit stubs are to be labels that are suitable to the target language. The special keyword "RETURN" in an exit stub will be recognized as a program exit and will generate a RETURN statement in the object code where the target language contains such a statement. APL programs will treat all exits as the same (as a return). As before, you enter a carriage return (empty line) to signal the end of this table section. (Note that you must enter at least one exit stub or the input

process will be cancelled.)

HOW MANY RULES ARE THERE?

You respond by entering the number of rule columns that you will require in the entry section of the table. All subsequent entry input must contain this number of columns. If you respond to the rule-count request with a zero (0) or an empty line the input program assumes that you want the maximum number of columns. It will then automatically generate all of the condition entries for you in a standard way. You will also be told how many rule columns there are. The standard condition entry sequence is as follows: The first condition row will have the first half of its columns filled with "Y" and the second half filled with "*N*". In any subsequent condition row the first half of the columns under a "Y" sequence in the previous row will be filled with "Y" and the second half with "*N*". This is also true for the columns under each "*N*" sequence in the previous row. This rule is followed for all rows so that the last condition row contains "*YNYNYN....*". For example, if there are four conditions there will be sixteen columns filled as follows:

YYYYYYYYNNNNNNNN
YYYYNNNNYYYYNNNN
YYNNYYNNYYNNYYNN
YNYNYNYNYNYNYNYN

Following your entry of the rule column count you will be prompted to supply the entry data. The prompting will consist of the low-order digit of each column number and the stub information (that you have previously entered). Figure 4 shows this. You type the entries just as you want to see them in the final table. If the "standard" conditions were requested, prompting will start with the actions, otherwise prompting will start with conditions.

There are certain restrictions on the characters that are acceptable in each entry section of the table. The condition section accepts only the characters "Y", "*N*", and "-", for "yes", "no", and "don't care" values respectively. The action section can receive numbers, letters, underscored letters, or blanks. These characters allow an ordering of the actions in a given rule column. The numbers are lowest, letters next, and underscored letters highest. A Z (underscored *Z*) has a value equivalent to 61. A blank indicates that an action does not participate in a rule. The exit section accepts an *X* to indicate that an exit participates in a particular rule, and is blank otherwise.

BAD INPUT

If any of the above restrictions is violated, your input is reprinted with the illegal character(s) replaced by an "x". You can then correct the line by typing the correct character beneath the "x". You can also revise any other characters by typing (non-blank) characters beneath them. If you want to replace a character with a blank, type a slash (/) beneath it. You will not be allowed to proceed to the next line until you have eliminated all illegal characters from the current line.

OK (NG)

After the last exit entry is accepted, the entire decision table is scrutinized for global errors. The following checks are performed:

- Completeness: All possible combinations of condition values are accounted for.
- Consistency: No combination of condition values is accounted for in more than one rule.
- Action/Exit use: Each action and exit should be used in at least one rule. (warning only)
- Exit consistency: Exactly one exit must be used in each rule column.

If the above scrutiny finds the table to be correct, the letters "*OK*" are typed. Otherwise, detailed diagnostics and the letters "*NG*" are printed. These diagnostics are self-explanatory except for the following abbreviations:

CDN -- Condition number
YCT -- Yes count (number of yes's in a row)
NCT -- No count
DIF -- Difference between YCT and NCT

The yes- and no-counts are computed taking into account the "don't cares" in each column. A column with one "don't care" stands for two columns, so each yes or no in that column is counted twice. These counts can be useful in locating typing errors in the condition section of a table. For example, if the table in Figure 1 had a Y instead of an *N* in the second column of condition 2, then you will be told that for condition 2 the yes-count is 5 and the no-count is 3. You are also told that columns 2 and 4 overlap. So the trouble is narrowed to condition 2, columns 2 and 4. You are also told that the rule "*YNN*" is missing. You can then conclude that if column 2 were changed to "*YNN*" that the table would be satisfactory.

```
→        )LOAD DTABL        ⍝LOAD DTABL WORKSPACE
   SAVED 14:56:37 08/17/76
→        INPUT
   ENTER TABLE HEADER       ⍝ARROW (→) INDICATES LINES
→  EXAMPLE                  ⍝TYPED BY THE USER.  ALL OTHER
   ENTER CONDITION STUBS    ⍝LINES ARE OUTPUT FROM THE
→  C1                       ⍝SYSTEM.  (THE ARROWS ARE NOT
→  C2                       ⍝NORMALLY PART OF THE LISTING.)
→  C3
→→
   ENTER ACTION STUBS
→  A1
→  A2
→  A3                       ⍝DOUBLE ARROWS (→→) INDICATE A
→→                          ⍝CARRIAGE RETURN PERFORMED BY
   ENTER EXIT STUBS         ⍝THE USER.
→  X1
→  X2
→→
   HOW MANY RULES ARE THERE?
→  7
   0000000
   1234567|C1
→  NYNYNYN
   1234567|C2
→  NN YYNN
   BAD INPUT                ⍝BLANK IS NOT A LEGAL CONDITION.
   NN×YYNN|C2
→    Y                      ⍝CORRECT ONLY THE BAD COLUMN.
   1234567|C3
→  NNN-YYY
   1234567|A1
→   1 1 11
   1234567|A2
→    1   1
   1234567|A3
→  12
   1234567|X1
→      XX
   1234567|X2
→  XXXX  X
   OK                       ⍝THE TABLE IS CONSISTENT.
→        SAMPLE←SAVE        ⍝TABLE SAVED AND NAMED.
```

Example of Initial Table Entry
Figure 4.

The input process is completed with the typing of *OK* or *NG*. If there were errors you can correct them as described below under EDITING A DECISION TABLE. If you should try to compile a table with errors, the results will be unpredictable.

You can cancel the input process by entering an empty line when prompted for the table header, or condition, action, or

exit stubs. (The implication is that a table must have a header and at least one condition, action and exit.) You can also cancel input if you enter an "x" (an APL multiply sign) in any entry.

Table verification can occasionally produce many error messages. To terminate these prematurely, press the attention button and enter

```
→STOPCHECK
```

You can then edit the resulting table to remove gross errors.

CORRECTION OF TYPING ERRORS

If you discover an error in typing a line before you have given the carriage return, you can correct it by backspacing to the point of error and pressing the attention button. This will erase all of the line at, and to the right of the point of error. A linefeed will acknowledge this along with a caret. You continue the line just below the caret. This facility is available on all input. (On an IBM 3270 you just backspace and type over the incorrect input.) If your error is discovered after a carriage return, it is better corrected via *EDIT*.

STORING AND DISPLAYING DECISION TABLES

After you have finished the input process described above, the decision table exists in a special internal format that allows efficient processing. We say the decision table is "loaded" when it is in this condition. To display a loaded decision table, simply type

```
SAVE
```

To place the decision table into a compact form and give it a name, type

```
tname←SAVE
```

An example of this is given as the last line of Figure 4. In future operatons of editing, compiling, etc., you can process this table by referring to its name. (e.g., *EDIT* tname, *COMPILE* tname). You can both save and display a decision table at the same time by typing

```
tname←□←SAVE
```

You are cautioned that none of these processes save the

table across terminal sessions. In naming your decision table you should avoid the names of data structures being used by the processor. Most of these start with an underscored letter. Those that don't are listed in the section TABLE OF RESERVED NAMES near the end of this manual.

A decision table can be reloaded by typing:

LOAD tname

It is seldom necessary to do this, however, because this operation is incorporated into all of the functions that manipulate decision tables.

SPREAD tname

also displays the table in compact form. In this display vertical bars are placed between the entry columns to improve readability.

SAVING TABLES ACROSS TERMINAL SESSIONS

In the APL system there is a library facility. It is implemented through the concept of "workspaces." When you sign on you have immediately available a "clear workspace" and a private library of (previously saved) workspaces. Each workspace that is to be separately saved must be uniquely named. This is done by typing

)*WSID* myws

where "myws" stands for any name (up to 8 letters long) you choose to give your workspace. (*WSID* means "workspace identifier".)

To save a workspace in your private library you type

)*SAVE* myws

If you intend to use the contents of a workspace for the next session, you should save your workspace before signing off of the current session. When you save a workspace the contents of any previously saved copy of this workspace are overwritten. To retrieve a workspace from your private library type:

)*LOAD* myws

Note that your initial workspace name (given in the sign-on instructions) was "libno *DTABL*". You must rename this workspace before you save it because the APL system will not save it under the original name.

Brief explanations of all messages are listed in the section MESSAGE SUMMARY. If you do not understand a message in your use of *DTABL*, you should save your workspace before signing off. Perhaps a person in your area who is familiar with APL can help you. If not, you can send your listings to the maintenance address indicated when you execute the function *DESCRIBE* in your *DTABL* workspace.

EDITING A DECISION TABLE

You will undoubtably have occasion to change a decision table after it has been entered -- either to correct an error, or to change the desired logic. The editing process that provides a ready means to make changes is initiated by typing:

EDIT tname

This will cause the named table to be loaded and start the editing process. If you want to edit a table that is already loaded, type:

EDIT 0

The editor allows you to insert rows or columns, delete rows or columns, and to change all or part of rows or columns, and to display all or part of the table being edited. You will be prompted with a quad-colon (⎕:) and you respond with a request as indicated below.

```
→       EDIT SAMPLE        ⍝INITIATE EDIT OF SAMPLE
   ⎕:
→       SHOW ALL           ⍝DISPLAY TABLE BEING EDITED
      ^^^^^^^
      1234567             ⍝WITH ROW AND COLUMN
      vvvvvvv             ⍝NUMBERS.
   <1>NYNYNYN|C1
   <2>NNYYYNN|C2
   <3>NNN-YYY|C3
   <1> 1 1 11|A1
   <2>  1   1|A2
   <3>12     |A3
   <1>    XX |X1
   <2>XXXX  X|X2
```

Initiate Table Editing and Display Table
Figure 5.

Requests are made up of a verb which is usually followed by row/column specifications. In Figure 5 the table *SAMPLE* is loaded for editing. In response to a prompt from the editor a request (*SHOW*) is made for a display of all rows and columns. Notice that each row and column is given a number

that is printed between angle brackets. Notice also that the stubs are printed to the right of the entries (to speed up printing). These row and column numbers are used in row/column specifications to select the section of the table to be displayed or changed. The row numbers start from 1 in each of the three lateral sections of the table (conditions, actions and entries). Figure 6 shows examples of display requests for sections of the table being edited.

```
   [:                                .     [:
→        SHOW COL 4 5                .  →        SHOW CND 1, ACT 2 3
      ^^                             .        ^^^^^^^
      45                             .        1234567
      vv                             .        vvvvvvv
   <1>YN|C1                          .     <1>NYNYNYN|C1
   <2>YY|C2                          .     <2>  1   1|A2
   <3>-Y|C3                          .     <3>12     |A3
   <1>1 |A1                          .
   <2>  |A2                          .
   <3>  |A3                          .
   <1> X|X1                          .
   <2>X |X2                          .
```

Example of Selective Display Requests
Figure 6.

(Note: Subsequent examples will constitute continued changes on the table *EXAMPLE*. The examples are taken from a sample terminal session that is listed in its entirety at the end of this manual.)

Alteration verbs are:

CHG (change existing table elements)
CHS (change existing stubs)
ADD (add new rows or columns to the table)
DEL (delete existing rows or columns from the table)
REHEAD (change the table header)

A row/column specification consists of a prefix followed by one or more numbers. The prefixes are:

COL (column numbers follow)
CND (condition row numbers follow)
ACT (action row numbers follow)
EXT (exit row numbers follow)

The following are special specifications and need no number.

ALLCOL (all columns including the stub column)
ALL (all rows and all columns)

The request

CHG ACT 2 3, *COL* 5 6

asks to change action rows 2 and 3 in columns 5 and 6. You will be prompted with the contents of these elements, one row at a time. The change is made like correcting initial input -- i.e., nonblanks replace, and a slash blanks out those characters under which they are typed. If you want to change a stub, you can refer to it as column zero (0) -- plus any appropriate row specification. You replace the contents of a row stub by merely typing new text beneath it. In this case, everything you type to the right of the vertical divider will replace the stub. (See *CHS* for a way to partially modify a stub.) Note that the row/column specifications must be separated by a comma, and that the numbers must be separated by at least one blank. If a row specification is omitted, it implies "all rows". If a column specification is omitted, it implies "all entry columns". If you want to refer to all columns (in a change -- *CHG*) including the stub column, use the specification "*ALLCOL*".

```
→       SHOW CND 2
      ^^^^^^^
      1234567
      vvvvvvv
   <2>NNYYYNN|C2
   [:
→       CHG CND 2,COL 4 7
   YN|C2
   --
   [:
→       SHOWCND            ATHE ATTN BUTTON WAS PRESSED
           v               ATO CORRECT A TYPING ERROR.
→            CND 2
      ^^^^^^^
      1234567
      vvvvvvv
   <2>NNY-YN-|C2           AELEMENTS IN COLS 4,7 REPLACED.
```

Example of a Row Change (*CHG*)
Figure 7.

The request

CHS CND 1, *ACT* 2 3

asks to perform modification of the stubs of condition 1 and actions 2 and 3. You will be prompted with the image of each stub and a right arrow. To the right of the arrow you enter a modification request of the form "/string1/string2". Every instance of "string1" will be replaced by "string2" in the stub. (In the foregoing "/" stands for any character not contained in the arbitrary character sequences "string1"

and "string2".) Multiple pairs of strings can be entered at one time. For example "/A/BB/XX/Y" changes all A to BB and all XX to Y. After each change you are again prompted by the arrow to enter additional changes to the same stub. When the right arrow prompts you, you can also display the results of the previous change by typing one space (blank) followed by a carriage return. A single slash (/) followed by a carriage return will cancel all changes made to the current stub. To move to the next stub you enter an empty line. When all stubs requested have been changed you return to the normal mode and are prompted with a □:.

If your modification request starts with a letter or number it is assumed that you want to entirely replace the stub with the text you enter. If your modification request starts with an asterisk (*) then it is assumed to be of the form "*n/string1/string2". In this case you are asking to replace the nth instance of string1 with string2. "+n" means replace the first n instances of string1, and "-n" means replace the last n instances of string1.

```
   □:
→        CHS CND 2          ⍝CHANGE STUB ONLY.
   |C2
→  →/C/COND/
→B →                        ⍝ARROW B (→B) INDICATES SINGLE
   |COND2                   ⍝BLANK TYPED TO DISPLAY NEW STUB.
→  →/                       ⍝CANCEL CHANGE, AND
   |C2                      ⍝SHOW RESTORED VALUE.
→→ →
```

Example of Stub Modification
Figure 8.

The request

ADD COL 3.5 3.6

asks that two columns be added to the table. You will be prompted to fill these columns. The new columns are to be numbered 3.5 and 3.6 respectively indicating their ordering among the old columns. (They will be between column 3 and 4.) When the new columns are added, they do not affect the numbering of any prior columns. A row or column therefore retains a unique number throughout the editing process. When the editing process begins, rows and columns are given consecutive integers starting from 1 in each section of the table. You can insert columns between 3.5 and 3.6 by adding a column numbered with any number that falls between them -- say 3.55 or 3.596. Fractional numbers with up to 15 digits to the right of the decimal point can be used in this fashion. However, when you display the table (described below) a maximum of three digits will be printed on each side of the decimal point.

```
→       ADD CND 2.5          ⍝NEW CONDITION ROW TO BE FILLED.
   xxxxxxx|x
   YYYNNNN C2.5
   □:
→       SHOW CND 1 2 2.5 3
        ^^^^^^^
        1234567
        vvvvvvv
   <1.0>NYNYNYN|C1
   <2.0>NNYNYNY|C2
   <2.5>YYYNNNN|C2.5         ⍝NOTE NEW ROW.
   <3.0>NNN-YYY|C3
```

Example of Row Addition
Figure 9.

The request

DEL ACT 6

asks for the deletion of action row number 6. If row 6 does not exist, no action will be taken and you will be so informed. If both row and column specifications are included in a request for addition or deletion, the entire row(s) and column(s) will be added or deleted. If both row and column are specified in a change request, only the elements at the intersection(s) of the row(s) and column(s) will be affected.

When you request a change or addition (implying that new material is to be placed in the table) you can specify that the added data be obtained from another part of the table. This is accomplished by using a *FROM* specification. For example,

ADD COL 4.1, *FROM COL* 4

requests that a new column (number 4.1) be added to the table, and that it is to be filled with the contents of column 4.

CHG CND 2 3, *FROM CND* 3 2

requests that condition rows 2 and 3 be changed, and filled with the contents of condition rows 3 and 2. Because such changes take place "simultaneously", the effect is to reverse the order of the entry protion of rows 2 and 3. To specify the reversal of the stubs of these rows, type

CHG CND 2 3, *COL* 0, *FROM CND* 3 2, *COL* 0

and to reverse the entire contents of the rows, type

CHG ALLCOL, CND 2 3, *FROM ALLCOL,CND* 3 2

Note from the above example, that the order of the specifications is immaterial (i.e., row-column or column-row) but that the ordering of the numbers within a specification is significant. In all cases, the *FROM* specification should be last.

```
→       CHG CND 2, COL 4 7, FROM CND 2 , COL 2 3
  □:
→       SHOW CND 2
      ^^^^^^^
      1234567
      vvvvvvv
  <2>NNYNYNY|C2          ØELEMENTS COPIED.
```

CHG request with *FROM* specification
Figure 10.

In the event that you wish to cancel a partially entered request, you can abandon it by typing *KILL* (preceded by a comma) at the end of the line. For example

CND ACT 2 3, COL 4 5, FROM CND 2,KILL

To rehead a table or otherwise modify the table header, enter the request

REHEAD

You will be prompted with the current header and a right arrow. You can modify the header using the conventions used in *CHS* (including display or cancellation of the modification). However, if you merely enter a new header it will replace the old one.

To terminate an edit session type

END

This will cause the table to be verified as it is when initial input is completed. You will receive the *OK/NG* signal as appropriate, and you can save the resulting table via the *SAVE* order. (The table is in the "loaded" state at conclusion of editing.) Before the table is verified, a check is made to see if there are any x's in the table as a result of editing. If there are, you are not allowed to terminate and will be prompted by the □:. You can use *SHOW* to see where the errors are.

Verification of the table takes place at the conclusion of editing and input processing. If the table has any serious errors it will be marked so that the compiler will know to warn you.

The editing process can be aborted at any time you are prompted with the □: by typing

ABORT

If you do this, the results of the edit are lost and there will be no resultant table. You will not lose any table you have retained by using the *SAVE* function.

Self-explanatory diagnostics will be issued if you attempt to change a non-existent row or column, or if you omitted a prefix (*COL*, *CND*, *ACT*, *EXT*), or if there is a mismatch between the number of rows and columns supplied in a *FROM* specification and the space it is to fill. In some cases you will be prompted to supply a correction. If you cannot properly correct the error, you can *KILL* the request (by entering "*KILL*") and write a new one. Any time you are prompted with a □:, you can *KILL* a request by typing "*KILL*" at the end of the line.

COLUMN MERGING

As indicated in the introduction, columns can be merged by the introduction of "don't care" values. You can do this by hand before entering the table, or you can have the *DTABL* system do it for you by typing

MERGE tname

This will cause the named table to be loaded and the appropriate columns to be merged. If the table is already loaded, type

MERGE 0

To save the table resulting from *MERGE*, use *SAVE*. The merging process will not affect any "don't cares" that you have already introduced. There is an example of the *MERGE* process in the sample terminal session at the end of this manual.

ADDING AN "ELSE" COLUMN

During the verification process, if the table conditions are found to be incomplete, you will be asked if you want to add an ELSE column. You may answer YES or NO. If you answer YES, the condition section of the table is completed automatically by the processor. You will then be asked to supply the action and exit column contents. If you must add an action stub to the table for inclusion in the ELSE column, you should answer NO and use the editor to add these rows first. The editor will then ask you again whether you want the ELSE column added and you can say YES. You will be

asked to enter an exit name for the ELSE column. You can then enter an existing or new exit stub. (Note that the ELSE column actually may be more than one column. This happens when it requires more than one column to represent all of the omitted condition combinations. The same actions and exit will be selected in all of the added columns. There is nothing special about these columns once they are added.)

You can initiate the addition of an ELSE column yourself by typing

ADDELSE tname or *ADDELSE* 0

depending on whether the named table is already loaded.

COMPILING DECISION TABLES

To compile a decision table, simply type

COMPILE tname

This will cause the table to be loaded and then compiled. If the table is already loaded, use zero (0) instead of the table name.

The compiler, using only entry information, produces a procedural code structure in which the units are condition tests (that always branch on true), actions, and exits. The code is initially in a tree structure with the exits as terminal elements. Because such a tree often contains many redundant code sequences, certain optimization processes are employed to reduce the code to a more efficient form. These optimizations (detailed in [1] and [2]) remove redundant code, reorder condition tests, and relocate actions in order to reduce the size of the resultant code. Unconditional branches may be introduced at this time, but the logic of the original form of the program will remain unchanged.

Optimizations *OPT2* and *OPT3* operate under the assumptions regarding sequence independence stated earlier. (Optimization options are listed on page 20.) You are therefore cautioned to assure that actions in a table do not modify condition test results. Program loops should be avoided unless they encompass the entire decision table, or are entirely contained within a single action.

The code resulting from compilation will be optimized primarily with respect to space. There will be a one-to-one correspondence between the number of rule columns in the source table and the number of flow paths through the object code. No path will contain more tests than necessary to isolate the particular rule that it represents. Each path

will contain the required actions in the specified order, and will terminate in the specified exit. Speed optimization is limited to the avoidance of redundant testing in a given flow path. Because the compiler does not analyze the meaning of the stub information, no attempt is made to perform classical optimizations such as common subexpression elimination or folding. Such a task is beyond the intent of this compiler.

In the absence of other information, the optimizer assumes that all actions and conditon tests take the same amount of space (8 bytes). You can override this assumption by including a space estimate in the stub section of the table. The size in bytes should follow the action and condition information (entry section) and be enclosed in "⊂ ⊃" brackets.

If the compiler detects an unverified table it will warn you but give you the option to proceed with a compilation. The results of such a compilation will not be reliable. In some cases the compiler will abort the compilation because of overlapping or missing rule columns.

Several types of output listings can be produced by the compiler -- "abstract", COBOL, PL/I, ALGOL, and APL. The non-abstract outputs are collectively called "HLL" (for Higher Level Language). Examples of these can be found in the terminal session example at the end of this manual. (If you are able to program in APL you can, with the aid of [2], readily develop your own HLL listing generator for the language of your choice.)

The abstract format is analogous to an assembly listing with a label column, code column, and operand column. The op-codes are references to the conditions, actions, and exits of the original table. "Cnn" refers to condition number "nn". "Ann" and "Xnn" similarly refer to actions and exits respectively. All labels are two-digit numbers, and appear in column 1 followed by a colon. Condition tests (which are conditional branches) and unconditional branches (three dashes) have a right arrow followed by a label which is the label of the branch-target.

If the stub section of the decision table is properly formatted, you can obtain a listing of a valid HLL procedure. The condition section should contain relational expressions; the action section should contain assignment statements; and exits should be valid HLL program labels. The HLL lister merely places the stub information (excluding space estimates) verbatim into the appropriate HLL template. The decision table header should also be a valid HLL label, both of itself, and when suffixed by a two-digit number. (Note that stubs can be in English or other natural language suitable to the user. This is usually the case when the questionnaire processor will be used. The language selected

does not affect compilation.)

There are a number of compiler options. Prior to compilation they sould be set as follows:

OPTIONS←option-list

The options in the list can be placed in any order and are separated by commas. The options selected will remain in force until they are explicitly set again. When a workspace is saved the options in force at the time it is saved are retained in the saved workspace. They will be in force again when you reload the workspace. Options can be set at any time between compilations. The list of options is below.

ALGOL	List object code in ALGOL format
APL	List object code in APL format
APLG	Generate APL program
COBOL	List object code in COBOL format
PL1	List object code in PL/I format
SLIST	List object code in abstract format
ILIST	List preoptimized code (abstract format)
FILE	Place output on a file
OPT1	Perform optimization 1 (duplicate sequence removal)
OPT2	Perform optimization 2 (duplicate path removal)
OPT3	Perform optimization 3 (hoisting)
OPTM	Perform all optimizations
DETAIL	Report details of compilation progress

The option list in the distributed version of *DTABL* is

OPTM

If an option is omitted from the list, the corresponding action will not be performed.

If listings are requested the processor will print a row of periods, pause, and unlock the keyboard. You can then position the paper and signal when you are ready for printing by giving a carriage return.

The *FILE* option can be specified in conjunction with the *ALGOL*, *COBOL* and *PL1* listing options. When this is done, the listing is placed in a file instead of appearing on the terminal. The file has the same name as the table header. The various APL systems use different qualification conventions to separate user files. There are also varying protocols for establishing a connection between the APL and operating system environments. While these protocols are simple, you must consult the APL User's Guide [4_8] relevant to your system for details.

An example of the compilation of the table in Figure 1 is shown in the terminal session example at the end of this manual.

DISPLAYING AND SAVING COMPILATION RESULTS

The results of compilation are held in an encoded matrix called *MAT*. This can be saved and loaded in a manner analogous to the saving and loading of decision tables. By typing

```
mname←SAVEMAT
```

you will cause the object matrix to be compressed, saved, and named. To reload it, type

```
LOADMAT mname
```

Note that saving a matrix with *SAVEMAT* does not save it across terminal sessions. You use *)SAVE* myws to do that.

To display object code in HLL format, set the options list for the desired language and type:

```
tname PRINTHLL mname
```

To print object code in abstract format, type

```
ABSTRACT mname
```

Note that these requirements load the code matrix, and *PRINTHLL* also loads the decision table. To request printing of a loaded matrix or table you can use zero (0) instead of the name. The code matrix is loaded and unnamed just following compilation.

After these listing routines are invoked, they pause to allow you to position the paper. You signal that you are ready by giving a carriage return. The listing routines print approximately 50 lines per page.

SORTING A DECISION TABLE

It has been discovered that by sorting a decision table according to certain "rules of dominance", one can gain a better understanding of the relationships among the conditions. *DTABL* therefore provides a utility sort function. One condition row is said to "dominate" a second row if for all of the "yes columns" of the first row, the second row has all "don't care" values; or for all of the "no columns"

of the first row the same is true.

The sort routine first sorts rows in order of increasing number of "don't cares". Then it sorts the rows so that they are in disjoint "groups". (Two rows are in the same group if they both have a non-don't care (i.e., "Y" or "N") in the same column.) Finally, the rows are sorted so that if one row dominates another, it lies above the other in the final arrangement. After the rows are sorted, the columns are sorted as though each condition entry was a digit in a base-3 number system ("Y"=0, "N"=1, and "-"=2, and as though each column was a number. The numbers so devised are sorted in ascending sequence. Figure 11 below shows a decision table before and after sorting.

To load a decision table and sort it, type:

DSORT tname

As usual, a zero (0) instead of the table name will cause sorting of the table that is already loaded. *DSORT* leaves the sorted table in loaded form.

```
SORT|000000000111          SORT|000000000111
    |123456789012              |123456789012
----------------           ----------------
C01 |YYNNNNNNNNNN          C01 |YYNNNNNNNNNN
C02 |--YYYYYYYYNN          C06 |YNYYYYYNNNNN
C03 |--YYYYYYNN--          C02 |--YYYYNYYYYN
C04 |--YYNNNN----          C03 |--YYYN-YYYN-
C05 |----YYNN----          C04 |--YNN--YNN--
C06 |YNYNYNYNYNYN          C05 |---YN---YN--
----------------           ----------------
A01 |    1111              A01 |   11   11
A02 |    1111              A02 |   11   11
A03 |    1111              A03 |   11   11
A04 |      11              A04 |    1    1
A05 |111111111111          A05 |111111111111
----------------           ----------------
X01 |X X X X X X           X01 |X XXXXX
X02 | X X X X X X          X02 | X     XXXXX

      before                     after
```

Example of Decision Table Sorting
Figure 11.

TEST CASE GENERATION

In the object code generated from a decision table there is one path for each rule column in the source table. If one test case is prepared for each rule column then every instruction in the object code will be executed, and every

conditional branch instruction will be executed for both the true (yes) and false (no) values of the corresponding condition. Because of the action of the optimizers it is often the case that both of the above test criteria can be met with fewer test cases than there are rule columns. This is because the optimizers overlap paths through the object code. The extent to which testing can be reduced depends upon the extent to which the paths overlap. To determine the minimum set of test cases enter:

tname *TESTGEN* mname

and you will be supplied with a list of rule column numbers which constitute the minimum coverage set. (As usual you can use zero (0) for tname and mname if the corresponding table or code matrix is already loaded.)

QUESTIONNAIRE PROCESSING

It is possible to use a decision table as the basis for an on-line questionnaire. Each condition stub in the table constitutes a question to be asked which can be answered "yes" or "no" by a user who is sitting at the terminal. Each answer determines which question should be asked next, or whether no further questions should be asked. When a sufficient number of answers are given to enable the isolation of a single rule column, the actions and exits selected by that column constitute a list of responses to the user. *DTABL* contains a questionnaire processor that operates in this fashion when the user enters:

Q tname

where tname can be zero (0) for a table that is already loaded. The questionnaire processor appends question marks to the condition stubs before presenting them to the user. The answers entered in response to the questions can be any word containing at least one "Y" (meaning "yes") or one "N" (meaning "no") but not both. For example OKAY, YES, YEP, Y and YEAH would be interpreted as "yes". NEIN, NOPE, N, NAY, or NO would mean "no". JA, which has neither a "Y" or "N", and NYET which has both would be rejected as ambiguous.

PROCESSOR CAPACITY

There are few programmed capacity restrictions in *DTABL*. You are limited to 255 conditions and a total of 255 actions and exits. Furthermore, this restriction applies only if you want to use *SAVEMAT* (described above).

There are, however, restrictions imposed by the APL environ-

ment under which *DTABL* is implemented. These restrictions are in input and output line widths, and in storage capacity. Experience to date shows that in practice these limitations are seldom encountered.

The maximum stub width is 130 characters. If your table width (including entries and stubs) exceeds this, table displays will deteriorate -- i.e., a single row will be placed on two or more lines. In any event neither half can exceed the terminal width because an end-of-line cuts off input.

The HLL and abstract printing routines will print only the two low-order digits of a numeric label, although the restriction on label values in *MAT* is over 16 million.

The size of the workspace can affect the size of a table that can be compiled. *DTABL* has handled tables with as many as ten conditions and 35 rule columns without spilling storage in a 50K workspace.

You can reduce the chances of a storage spill by erasing those tables or matrices which you have saved but no longer need. This can be done by typing:

```
)ERASE name1 name2 name3 etc.
```

The list of names (separated by blanks) are those named items you wish to discard.

You can *LOAD* a table and *)ERASE* the compact (unloaded) version prior to operating on it.

You can establish a workspace solely for storing tables (or object codes) and using the *)COPY* facility of APL to transfer your tables (codes) to it. After saving your compiler workspace under the name of your choice, perform the following sequence:

```
)CLEAR
)WSID mydata
)COPY myws name1 name2 ...
)SAVE mydata
```

Repeat the *)COPY* line as many times as needed to save the desired tables (codes). You can then reload your compiler workspace and erase the copied tables (codes) from it. (Don't forget that your library copy of the compiler work-space still contains the old version of tables (codes) until you *)SAVE* the updated version.)

There are three kinds of overflow messages you can get. They are *WS FULL*, *SYMBOL TABLE FULL* and *STACK FULL*. APL maintains a symbol table of all the names you create (as well as those that are used in *DTABL*). Unfortunately, when

you)*ERASE* a named item, its name is not removed from the symbol table. Your recourse will be to perform the sequence below if you received the *SYMBOL TABLE FULL* diagnostic.

```
)ERASE unneeded tables and object codes
)SAVE myws
)CLEAR
)SYMBOL 500 (= bigger than the current setting)
)COPY myws
)WSID myws
)SAVE myws
```

You can then restart the interrupted activity.

If you receive a *WS FULL* message try erasing unneeded decision tables then perform the above sequence without the ")*SYMBOL*" line. Then retry the operation that was interrupted.

If you receive a *STACK FULL* message try the above sequence with *STACK* in place of *SYMBOL*. You can find out the current size limitations by typing)*SYMBOL*or)*STACK* without a number following. The amount of free workspace is obtained by typing □*WA*. (Note that space given to the stack or symbol table is taken from the free workspace.)

Another way to obtain more space is to separate certain *DTABL* functions into several different workspaces. To assist you in doing this the *DTABL* workspace has been divided into six groups as follows:

COMPILER	all functions used by *COMPILE* (includes listing functions)
EDITOR	all functions used by *INPUT*, *EDIT* and *ADDELSE*
MISC	all functions used by *MERGE*, *DSORT*, *TESTGEN*, and *Q*
NCOMPILER	functions not used by *COMPILE*
NEDITOR	functions not used by *INPUT*, *EDIT* and *ADDELSE*
NMISC	functions not used by *MERGE*, *DSORT*, *TESTGEN*, and *Q*

Each of the first three groups can be used to copy relevant functions from *DTABL* into a clear workspace. The last three can be used to delete unneeded code from the workspace you are working with. For example, if you get a *WS FULL* during compilation, you can erase group *NCOMPILER* and continue.

If the *WS FULL* condition persists your needs have exceeded the capacity of the APL system you are running under. The remedies available are (1) to increase your virtual machine or region size, or (2) to move your workspaces to an APL system that accomodates larger workspace sizes. See your APL system administrator for assistance.

KEYWORD SUMMARY

Page	Keyword	Reminder
3	...	Sign on
4	)*LOAD* libno *DTABL*	Initialize workspace
4	)*OFF*	Sign off
4	*INPUT*	Initial table entry
4	*IS*3270←(0/1)	Terminal mode
9	→*STOPCHECK*	Interrupt checking
9	tname←*SAVE*	Save decision table
10	*LOAD* tname	Load decision table
10	*SPREAD* tname	Display table with verticals
10	)*WSID* myws	Give workspace a name (myws)
10	)*SAVE* myws	Save workspace in library
10	)*LOAD* myws	Load workspace from library
11	*EDIT* tname (0)	Edit decision table
12	*SHOW* spec	Show partially edited table
12	*CND* r r r r	Specify condition rows
12	*ACT* r r r r	Specify action rows
12	*EXT* r r r r	Specify exit rows
12	*COL* c c c c	Specify columns
12	*ALLCOL*	Specify all columns & stubs
12	*ALL*	Spec. all rows, cols & stubs
13	*CHG* spec	Request change as specified
13	*CHS* spec	Change stub
14	*ADD* spec	Request row/column additon
15	*DEL* spec	Request row/column deletion
15	*FROM* spec	Specify data source
16	*KILL*	Cancel edit command
16	*REHEAD*	Request table header change
16	*END*	Terminate editing
17	*ABORT*	Cancel editing
17	*MERGE* tname(0)	Merge table columns
18	*ADDELSE* tname (0)	Add an ELSE column to table
18	*COMPILE* tname (0)	Compile a table
20	*OPTIONS*←	Set compiler options
	ALGOL Algol format	*FILE* Place output in a file
	APL APL format	*OPT*1 Redundant Sequence
	APLG APL (executable)	*OPT*2 Redundant Path
	COBOL COBOL format	*OPT*3 Hoisting
	*PL*1 PL/I format	*OPTM* *OPT*1,*OPT*2,*OPT*3
	SLIST Abstract format	*DETAIL* Report progress
	ILIST Preoptimization Abstract Listing	
21	mname←*SAVEMAT*	Save results of compilation
21	*LOADMAT* mname (0)	Load object code matrix
21	tname(0) *PRINTHLL* mname(0)	Print in High Lev. Lang.
21	*ABSTRACT* mname (0)	Print code, abstract format
22	*DSORT* tname (0)	Sort decision table
23	tname(0) *TESTGEN* mname(0)	Generate test columns
23	*Q* tname (0)	Questionnaire processor

MESSAGE SUMMARY

Error Messages

STUB TRUNCATED TO 130 CHARACTERS	Table stub exceeded 130 characters, and was truncated to 130.
EMPTY STUB	Stub contains only space data.
NO ACTION	You tried to delete nonexistant rows or columns, or to add rows or columns that were already in the table. Or you tried to add an ELSE column to a complete table.
BAD INPUT	Incorrect input, "x" replaces the bad data shown with this message.
NOT EDITING	You tried to use the edit commands without entering the *EDIT* function.
IN EDIT. NO ACTION	You attempted to perform a non-edit function while editing.
VALUE ERROR n	n is a list of invalid row/column numbers.
MISSING PREFIX COL, CND, ACT, OR EXT	One or more of the listed prefixes was omited.
STUB MISMATCH	The *FROM* specification does not match the *CHG* specification with respect to column 0.
ROW LENGTH ERROR	The *FROM* specification does not match the *CHG* specification with respect to the number or rows.
COLUMN LENGTH ERROR	The *FROM* specification does not match the *CHG* specification with respect to the number of columns.
ERRORS	There is an empty condition, action or exit section in the table being edited.
YES-NO IMBALANCE CDN, YCT, NCT, DIF	There are not an equal number of Y's and *N*'s in some condition rows. The counts and differences are given for the imbalanced rows.

MISSING RULE: ...	The indicated rule column is not accounted for in the table.
RULE OVERLAP: n	"n" is a list of columns which have overlapping condition values.
NO RULE USES ACTION(S) n	"n" is a list of actions not used by any rule.
NO RULE USES EXIT (S) n	"n" is a list of the numbers of exits not used by any rule.
MISSING OR MULTIPLE EXITS IN RULE(S) n	"n" is a list of rule columns without exactly 1 exit.
NG	The table being entered or edited is in error.
tablename IS IN ERROR, DO YOU WISH TO PROCEED?	The compiler has detected an invalid table. You respond YES or NO.
PLEASE ENTER YES OR NO	You have given an ambiguous response to a YES/NO questions. Your response must contain either a Y or an N, at least one, not both.
RULE n OVERLAP. QUESTIONING TERMINATED RULE n OVERLAP. COMPILATION ABANDONED	"n" is a list of the rule columns that have been found by the compiler or questionaire processor to overlap. Occurs with an invalid table.
RULE MISSING. QUESTIONING TERMINATED RULE MISSING. COMPILIATION ABANDONED	Some rule column was discovered missing by the compiler or the questionnaire processor. Occurs with an invalid table.
ILLEGAL TABLE	You attempted to *LOAD* (possibly indirectly) an object that was not a decision table.
ILLEGAL OBJECT CODE	You attempted to *LOADMAT* (possibly indirectly) an object that was not a code matrix.
EMPTY TABLE	There is no table loaded (or saved).

EMPTY OBJECT CODE	There is no code matrix loaded (or saved).
INCOMPATIBLE TAELE/CODE	You have tried to use *PRINTHLL* or *TESTGEN* with an object code that does not match the decision table.
DEFINITICN ERROR: [n] line image	You used the APLG option to produce an APL object program, but it could not be defined because of an error in the line image displayed. If the line image is line 0, the error may be due to the existence in your workspace of an APL variable or group with the same name as your program. You will also get this message if the function you are trying to define is "pendant" in your workspace.
SYNTAX ERROR	You have entered a command improperly. Look for an omitted comma or blank or a mispelled keyword.
VALUE ERROR	You have mispelled something, or attempted to save a table that doesn't exist.

During an attempt to file output you may encounter one of the error messages listed below. The error number (n) accompanying the message is particular to the operating system you are using. Your APL administrator can direct you in seeking help if you cannot discover the cause of this error.

OPEN FAILURE: n	Probable cause is that your table header is not a legal file name. It is also possible that the APL shared variable interface is (temporarily) unavailable (output will be cancelled)
WRITE FAILURE: n	Probable cause is a line that is too long for the file being used.
INTEFFACE QUOTA EXHAUSTED	You don't have enough interface capacity for your file. See your APL administrator to increase your quota.
NO SHARES. SVP INACTIVE	You didn't include APL110 or APL121 when you accessed VSAPL.

SV SPACE QUOTA EXCEEDED — Too little space for Shared Variable processor. Get more when you activate APL by entering VSAPL 8K APL110.

WS FULL — Not enough free workspace to continue calculations. See the discussion under PROCESSOR CAPACITY.

SYMBOL TABLE FULL — Not enough symbol table capacity. See discussion under PROCESSOR CAPACITY.

STACK FULL — Too many recursions for the execution stack to handle. See the discussion under PROCESSOR CAPACITY.

xxxx[n] — You have stopped in function "xxxx" at line "n". This was caused by an interrupt (sometimes due to line noise). To resume execution enter "→n".

Prompting and Other Messages - - - - - - - - - - - - -

ENTER TABLE HEADER — Enter the header of the decision table.

ENTER CONDITION STUBS — Enter at least one condition stub.

ENTER ACTION STUBS — Enter at least one action stub.

ENTER EXIT STUBS — Enter at least one exit stub.

HOW MANY RULES ARE THERE? — Enter the number of rule columns for this table.

STANDARD CONDITIONS (n RULES) — You have requested "standard" conditions. There are "n" rule columns.

INPUT CANCELLED — You have cancelled the input process.

DO YOU WANT TO ADD AN ELSE COLUMN? — Your table has missing rules, do you want them added? Answer YES or NO.

nnn:ACTIONS	"nnn" is the action row numbers for the ELSE column you are adding.
ENTER EXIT NAME	Enter the exit stub for the ELSE column you are adding.
HEADER IS header	You have requested *REHEAD*. "header" is the current header expression of the table you are editing.
OK	The decision table has been verified following *INPUT* or *EDIT*.
COMPILING tablename	Compilation has started.
n LINES COMPILED IN s CPU SECONDS	Compilation is completed.
....	You are prompted to position the paper on your terminal to receive a program listing. Enter a carriage return to start printing.
FILING to tablename	Signals start of filing output
END OF FILED OUTPUT	Signals filed output complete

The following messages are printed during compilation when you have selected the DETAIL option.

n LINES GENERATED FROM tablename (prior to opts. 1 and 2)

LOCATE DUPLICATE SEQUENCES IN n LINES (opt 1)

n SEQUENCES TO BE EXAMINED

REMOVING REDUNDANT GOTOS FROM n LINES

REMOVING REDUNDANT GOTOS FROM n LINES

REMOVING REDUNDANT EXITS FROM n LINES

REMOVING DEAD CODE FROM n LINES

FURTHER SEQUENCE REDUCTION IS POSSIBLE

NO FURTHER SEQUENCE REDUCTION IS POSSIBLE

THERE ARE n LINES OF CODE LEFT (opt1)

ATTEMPTING DUPLICATE PATH REMOVAL (opt2)

TABLE OF RESERVED NAMES

ABORT	*CHS*	*DETAIL*	*INPUT*	*NMISC*	*REHEAD*
ABSTRACT	*CND*	*DSORT*	*KILL*	*OPTIONS*	*SAMPLE*
ACT	*COBOL*	*EDIT*	*LOAD*	*OPTM*	*SAVE*
ADD	*COIBM*	*EDITOR*	*LOADMAT*	*OPT1*	*SAVEMAT*
ADDELSE	*COL*	*END*	*MAT*	*OPT2*	*SHOW*
ALGOL	*COMPILE*	*EXT*	*MERGE*	*OPT3*	*SLIST*
APL	*COMPILER*	*FILE*	*MISC*	*PL1*	*SPREAD*
APLG	*DEL*	*FROM*	*NCOMPILER*	*PRINTHL*	*STOPCHECK*
CHG	*DESCRIBE*	*ILIST*	*NEDITOR*	*Q*	*TESTGEN*

REFERENCES

1. Myers, H. J., "Compiling Optimized Code from Decision Tables", IBM Journal of Research and Development, Vol 16 No. 5, September 1972, pages 489-503.

2. *DTABL* System Guide. (Licensed Material) LY20-2282

3. APL Language. GC26-3847

4. APL Shared Variables (APLSV) User's Guide. SH20-1460 or SH20-9087

5. APL/CMS User's Guide. SH20-1846

6. VS APL for CMS: Terminal User's Guide. SH20-9067

7. VS APL for VSPC: Terminal User's Guide. SH20-9066

8. VS APL for TSO: Yale University Terminal User's Guide. SH20-1872

EXAMPLE TERMINAL SESSION

```
→         )LOAD 27 DTABL     ⍝LOAD DTABL WORKSPACE
  SAVED 14:56:37 08/17/76
→         INPUT
  ENTER TABLE HEADER         ⍝ARROW (→) INDICATES LINES
→ EXAMPLE                    ⍝TYPED BY THE USER.  ALL OTHER
  ENTER CONDITION STUBS      ⍝LINES ARE OUTPUT FROM THE
→ C1                         ⍝SYSTEM.  (THE ARROWS ARE NOT
→ C2                         ⍝NORMALLY PART OF THE LISTING.)
→ C3
→→
  ENTER ACTION STUBS
→ A1
→ A2
→ A3                         ⍝DOUBLE ARROWS (→→) INDICATE A
→→                           ⍝CARRIAGE RETURN PERFORMED BY
  ENTER EXIT STUBS           ⍝THE USER.
→ X1
→ X2
→→
  HOW MANY RULES ARE THERE?
→ 7
  0000000
  1234567|C1
→ NYNYNYN
  1234567|C2
→ NN YYNN
  BAD INPUT                  ⍝BLANK IS NOT A LEGAL CONDITION.
  NN×YYNN|C2
→   Y                        ⍝CORRECT ONLY THE BAD COLUMN.
  1234567|C3
→ NNN-YYY
  12345←7|A1
→  1 1 11
  1234567|A2
→   1   1
  1234567|A3
→ 12
  1234567|X1
→     XX
  1234567|X2
→ XXXX  X
  OK                         ⍝THE TABLE IS CONSISTENT.
→         SAMPLE←SAVE        ⍝TABLE SAVED AND NAMED.
```

```
      SAMPLE              ⍝DISPLAY TABLE NAMED SAMPLE.
EXAMPLE|0000000
       |1234567
--------------
C1     |NYNYNYN
C2     |NNYYYNN
C3     |NNN-YYY
--------------
A1     | 1 1 11
A2     |  1   1
A3     |12
--------------
X1     |    XX
X2     |XXXX  X
→     EDIT SAMPLE         ⍝INITIATE EDIT OF SAMPLE
□:
→     SHOW ALL            ⍝DISPLAY TABLE BEING EDITED
   ^^^^^^^
   1234567                ⍝WITH ROW AND COLUMN
   vvvvvvv                ⍝NUMBERS.
<1>NYNYNYN|C1
<2>NNYYYNN|C2
<3>NNN-YYY|C3
<1> 1 1 11|A1
<2>  1   1|A2
<3>12     |A3
<1>    XX |X1
<2>XXXX  X|X2
□:
→     SHOW CND 1, ACT 2 3
   ^^^^^^^
   1234567
   vvvvvvv                ⍝SELECTIVE DISPLAYS
<1>NYNYNYN|C1
<2>  1   1|A2
<3>12     |A3
□:
→     SHOW COL 4 5
   ^^
   45
   vv
<1>YN|C1
<2>YY|C2
<3>-Y|C3
<1>1 |A1
<2>  |A2
<3>  |A3
<1> X|X1
<2>X |X2
```

```
   ⊏:
→        SHOW CND 2
      ^^^^^^^
      1234567
      vvvvvvv
   <2>NNYYYNN|C2
   ⊏:
→        CHG CND 2,COL 4 7
   YN|C2
   --
   ⊏:
→        SHOWCND              ATHE ATTN BUTTON WAS PRESSED
             v                ATO CORRECT A TYPING ERROR.
→             CND 2
      ^^^^^^^
      1234567
      vvvvvvv
   <2>NNY-YN-|C2              AELEMENTS IN COLS 4,7 REPLACED.
   ⊏:
→        CHG CND 2, COL 4 7, FROM CND 2 , COL 2 3
   ⊏:
→        SHOW CND 2
      ^^^^^^^
      1234567
      vvvvvvv
   <2>NNYNYNY|C2              AELEMENTS COPIED.
   ⊏:
→        CHS CND 2            ACHANGE STUB ONLY.
   |C2
→  →/C/COND/
→B →                          AARROW B (→B) INDICATES SINGLE
   |COND2                     ABLANK TYPED TO DISPLAY NEW STUB.
→  →/                         ACANCEL CHANGE, AND
   |C2                        ASHOW RESTORED VALUE.
→→ →
   ⊏:
→        ADD CND 2.5          ANEW CONDITION ROW TO BE FILLED.
   ×××××××|×
   YYYNNNN C2.5
   ⊏:
→        SHOW CND 1 2 2.5 3
         ^^^^^^^
         1234567
         vvvvvvv
   <1.0>NYNYNYN|C1
   <2.0>NNYNYNY|C2
   <2.5>YYYNNNN|C2.5          ANOTE NEW ROW.
   <3.0>NNN-YYY|C3
   ⊏:
→        DEL CND 2.5          ADELETE ROW JUST ADDED.
   ⊏:
→        ADD COL 4.5 4.6, FROM COL 1 1
```

```
   ⊏:
→        SHOW ALL
      ^^^^^^^^^
      123444567
      .........
      000056000            ⍝NOTE THAT CONDITION ROW 2.5
      vvvvvvvvv            ⍝HAS BEEN DELETED, AND COLUMNS
   <1>NYNYNNNYN|C1         ⍝4.5 AND 4.6 HAVE BEEN ADDED.
   <2>NNYNNNYNY|C2
   <3>NNN-NNYYY|C3
   <1> 1 1   11|A1
   <2>  1     1|A2
   <3>12  11   |A3
   <1>      XX |X1
   <2>XXXXXX  X|X2
   ⊏:
→        DEL COL 3.5
   NO ACTION               ⍝COLUMN 3.5 DOESN'T EXIST
   ⊏:
→        DEL COL 4.5 4.6
   ⊏:
→        ADD COL 2
   NO ACTION               ⍝COLUMN 2 ALREADY EXISTS.
   ⊏:
→        CHG CND 1 2, ALLCOL, FROM CND 2 1, ALLCOL
   ⊏:
→        SHOW ALL
      ^^^^^^^
      1234567
      vvvvvvv
   <1>NNYNYNY|C2
   <2>NYNYNYN|C1           ⍝NOTE THAT CONDITION ROWS 1
   <3>NNN-YYY|C3           ⍝AND 2 ARE REVERSED.
   <1> 1 1 11|A1
   <2>  1   1|A2
   <3>12     |A3
   <1>    XX |X1
   <2>XXXX  X|X2
   ⊏:
→        CHG CND 1 2, COL 4 7
   NY|C2
   YN
   YN|C1
   YN
   ⊏:
→        END
   OK                      ⍝TABLE IS CONSISTENT.
```

```
→       MERGEXAMPLE
MERGE|0000000001111111112222222222333
     |1234567890123456789012345678901 2
-----+--------------------------------
C1   |YYYYYYYYYYYYYYYYNNNNNNNNNNNNNNN
C2   |YYYYYYYYNNNNNNNNYYYYYYYYNNNNNNN
C3   |YYYYNNNNYYYYNNNNYYYYNNNNYYYYNNN
C4   |YYNNYYNNYYNNYYNNYYNNYYNNYYNNYYN
C5   |YNYNYNYNYNYNYNYNYNYNYNYNYNYNYNY
-----------------------------------------
A1   |11111111111111111111111111111111
A2   |1111
A3   |1111
A4   |    11111111          1111
A5   |11111111111111111111111111111111
A6   |1   1   1   1   1   1   1   1
A7   |11  11  11  11  11  11  11  11
-----------------------------------+-----
X1   |  XX  XX  XX  XX  XX  XX  XX  XX
X2   |XX  XX  XX  XX  XX  XX  XX  XX
→       MERGE MERGEXAMPLE
→       SAVE
MERGE|000000000111111
     |123456789012345
---------------------
C1   |YYY---YYY---NNN
C2   |YYYYYYNNNNNN---
C3   |YYYNNNYYYNNNYYY
C4   |YYNYYNYYNYYNYYN
C5   |YN-YN-YN-YN-YN-
---------------------
A1   |111111111111111
A2   |111
A3   |111
A4   |   111111
A5   |111111111111111
A6   |1  1  1  1  1
A7   |11 11 11 11 11
-----+---------------
X1   |  X  X  X  X  X
X2   |XX XX XX XX XX
```

```
                                    ⍝SAMPLE COMPILATION.
→         SAMPLE
    EXAMPLE|0000000
           |1234567
    ---------------
    C1     |NYNYNYN
    C2     |NNYYYNN
    C3     |NNN-YYY
    ---------------
    A1     | 1 1 11
    A2     |  1   1
    A3     |12
    ---------------
    X1     |    XX
    X2     |XXXX  X
→         OPTIONS←OPTM,COBOL,PL1,APL,SLIST
→         COMPILE SAMPLE
    COMPILING EXAMPLE
     14 LINES COMPILED IN 2 CPU SECONDS
    ..........
→→
    EXAMPLE.
                IF C1 THEN GO TO EXAMPLE03.
                IF C3 THEN GO TO EXAMPLE04.
                IF C2 THEN GO TO EXAMPLE05.
    EXAMPLE01.  A3.
                GO TO X2.
    EXAMPLE03.  A1.
                IF C2 THEN GO TO X2.
                IF C3 THEN GO TO X1.
                GO TO EXAMPLE01.
    EXAMPLE04.  IF C2 THEN GO TO X1.
                A1.
    EXAMPLE05.  A2.
    EXAMPLE06.  GO TO X2.
                COMMENT END OF EXAMPLE.
    ..........
```

```
→→
   EXAMPLE: BEGIN;
               IF C1 THEN GO TO EXAMPLE03;
               IF C3 THEN GO TO EXAMPLE04;
               IF C2 THEN GO TO EXAMPLE05;
   EXAMPLE01: A3;
               GO TO X2;
   EXAMPLE03: A1;
               IF C2 THEN GO TO X2;
               IF C3 THEN GO TO X1;
               GO TO EXAMPLE01;
   EXAMPLE04: IF C2 THEN GO TO X1;
               A1;
   EXAMPLE05: A2;
   EXAMPLE06: GO TO X2;
            END /* OF EXAMPLE */;
   ..........
→→
       EXAMPLE
  [1]  →(C1)/6
  [2]  →(C3)/10
  [3]  →(C2)/12
  [4]  A3
  [5]  →0
  [6]  A1
  [7]  →(C2)/0
  [8]  →(C3)/0
  [9]  →4
  [10] →(C2)/0
  [11] A1
  [12] A2
   ..........
→→
   EXAMPLE

   000|    C01→03
   001|    C03→04
   002|    C02→05
   003|01:A03
   004|    X02
   005|02:X01
   006|03:A01
   007|    C02→06
   008|    C03→02
   009|    ----→01
   010|04:C02→02
   011|    A01
   012|05:A02
   013|06:X02
```

APL Decision Table Processor (DTABL) Program Description/Operations Manual Printed in U.S.A. SH20-1924-1

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

APL Decision Table Processor (DTABL)
Program Description/Operations Manual

SH20-1924-1

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name and mailing address:

_____

_____

_____

(Optional Wording)

What is your occupation? _____

Number of latest Newsletter associated with this publication: _____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Reader's Comment Form

Fold and tape | Please Do Not Staple | Fold and tape

First Class
Permit 40
Armonk
New York

Business Reply Mail

No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department 825
1133 Westchester Avenue
White Plains, New York 10604

Fold and tape | Please Do Not Staple | Fold and tape

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Cut or Fold Along Line

APL Decision Table Processor (DTABL) Program Description/Operations Manual Printed in U.S.A. SH20-1924-1

APL Decision Table Processor (DTABL)
Program Description/Operations Manual

SH20-1924-1

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name and mailing address:

_____

_____

_____

(Optional Wording)

What is your occupation? _____

Number of latest Newsletter associated with this publication: _____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Reader's Comment Form

Fold and tape | Please Do Not Staple | Fold and tape

First Class
Permit 40
Armonk
New York

Business Reply Mail

No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department 825
1133 Westchester Avenue
White Plains, New York 10604

Fold and tape | Please Do Not Staple | Fold and tape

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Cut or Fold Along Line

APL Decision Table Processor (DTABL) Program Description/Operations Manual Printed in U.S.A. SH20-1924-1

SAMPLE
DEC TABLES
from GLANS

DTABL

A
DECISION TABLE
PROCESSOR

IUP 5796-PJB

THE DECISION TABLE IS A TOOL FOR

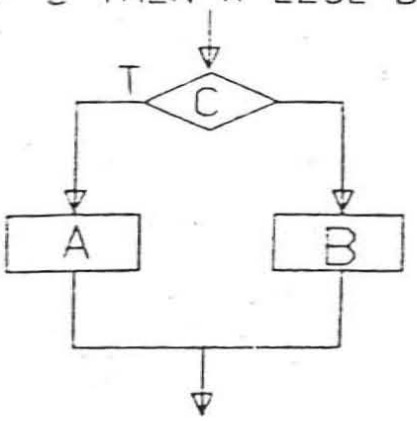
- PROGRAMMING AND
- COMMUNICATION

- LOGIC, NOT FLOW
- SIMPLE TO UNDERSTAND
- CONCISE, MEANINGFUL DOCUMENTATION
- INDUCES CHECKS ON LOGIC
  - CONSISTENCY
  - COMPLETENESS

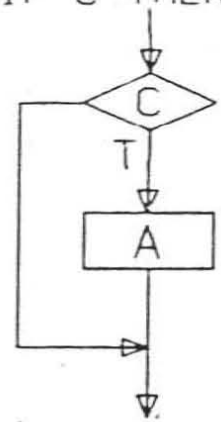
RESULTING PROGRAMS ARE

- QUICKER TO WRITE
- MORE RELIABLE
- EASIER TO MAINTAIN
- LESS COSTLY TO HAVE

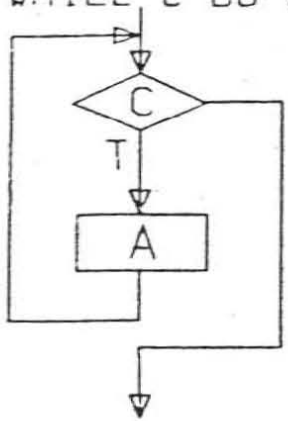
IF C THEN A ELSE B



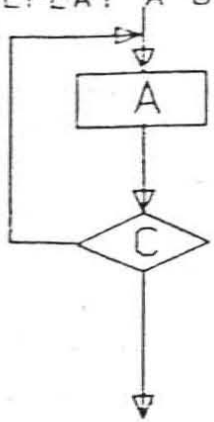
IF C THEN A



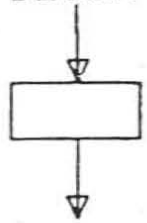
WHILE C DO A



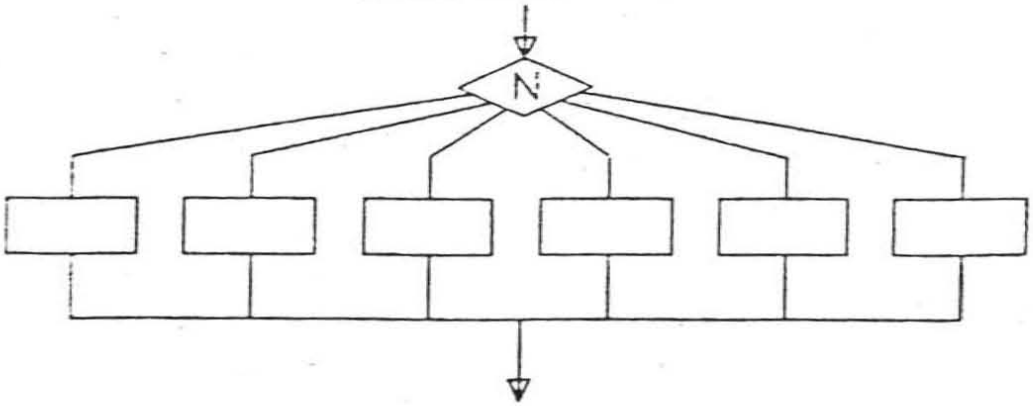
REPEAT A UNTIL C



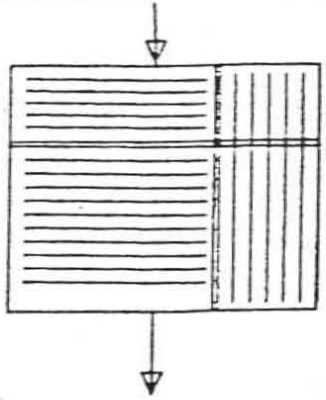
BLOCK



CASE N OF ...

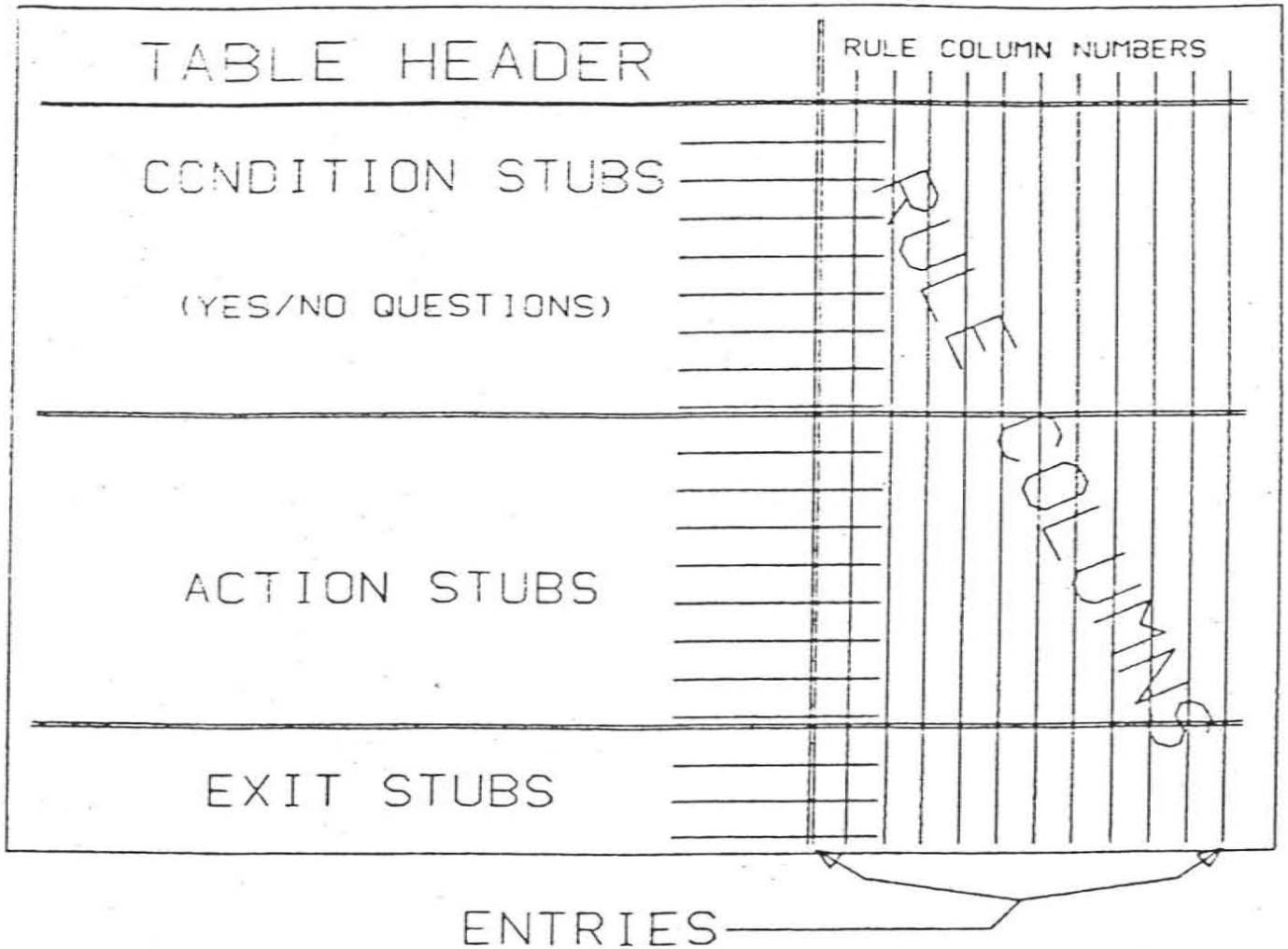


DECISION TABLE



BLOCK STRUCTURES

DECISION TABLE ANATOMY



```
D190F3                          |000000000011111
                                |123456789012345
--------------------------------|---------------
ERROR                           |YYYYYYYYYYYYYYN
TM ERROR DXCCW4                 |YYYYYYYYYYYYYYN-
RDBCK PARAM                     |YYYYYNNNNNNNN--
MOD JFCBIND2                    |-----YYYYYNN---
I/O ERROR                       |YNNNNYNNNNNYN--
RDBCK PARAM AGAIN               |-YNNN-YNNN------
MOD JFCBIND3                    |--YNN--YNN------
I/O ERROR FOR BSF               |---YN---YN------
--------------------------------|---------------
SET TO BSF                      |          11
BSP TWICE                       |1111111111
SET CCW1 TO BYPASS TM HDR       |   11    11
BSF                             | 1111 111111
SET UP ABEND 613                |1  1 1  1 1
--------------------------------|---------------
D190G1                          | XX X XX X XX
ABEND                           |X  X X  X X
D190F4                          |           X
```

COLUMN MERGING

1. ALL ELEMENTS SAME EXCEPT 1 CONDITION
2. VALUES IN CONDITION ARE YES AND NO

TWO EXAMPLES

C1	Y	Y	Y
C2	Y	Y	Y
C3	Y	N	-
A1	1	1	1
A2			
A3			
X1			
X2	X	X	X

C1	N	N	N
C2	Y	Y	Y
C3	-	-	-
C4	Y	N	-
A1	1	1	1
A2			
A3	1	1	1
X1	X	X	X

LOGIC_VALIDATION

N CONDITIONS REQUIRE
N
2 UNIQUE RULE COLUMNS

AMBIGUOUS

```
C1  I. . YY.
C2  I. . NN.
C3  I. . YY.
----------
A1  I. . 1 .
A2  I. .  1.
----------
X1  I. . XX.
```

REDUNDANT

```
C1  I. . YY.
C2  I. . NN.
C3  I. . YY.
----------
A1  I. . 11.
A2  I. . 22.
----------
X1  I. . XX.
```

INCOMPLETE

```
C1  I YNN
C2  I-YN
C3  I--Y
--------
A1  I11
A2  I2 1
--------
X1  IXXX
```

DTABL PROVIDES

- A TABLE EDITOR
- LOGIC VALIDATOR
- OPTIMIZING PRE-COMPILER
  (APL, COBOL, PL/I)
- TEST CASE ADVISOR
- TABLE MANIPULATION
  COLUMN MERGING
  TABLE SORTING
- QUESTIONNAIRE PROCESSOR

TABLE EDITING

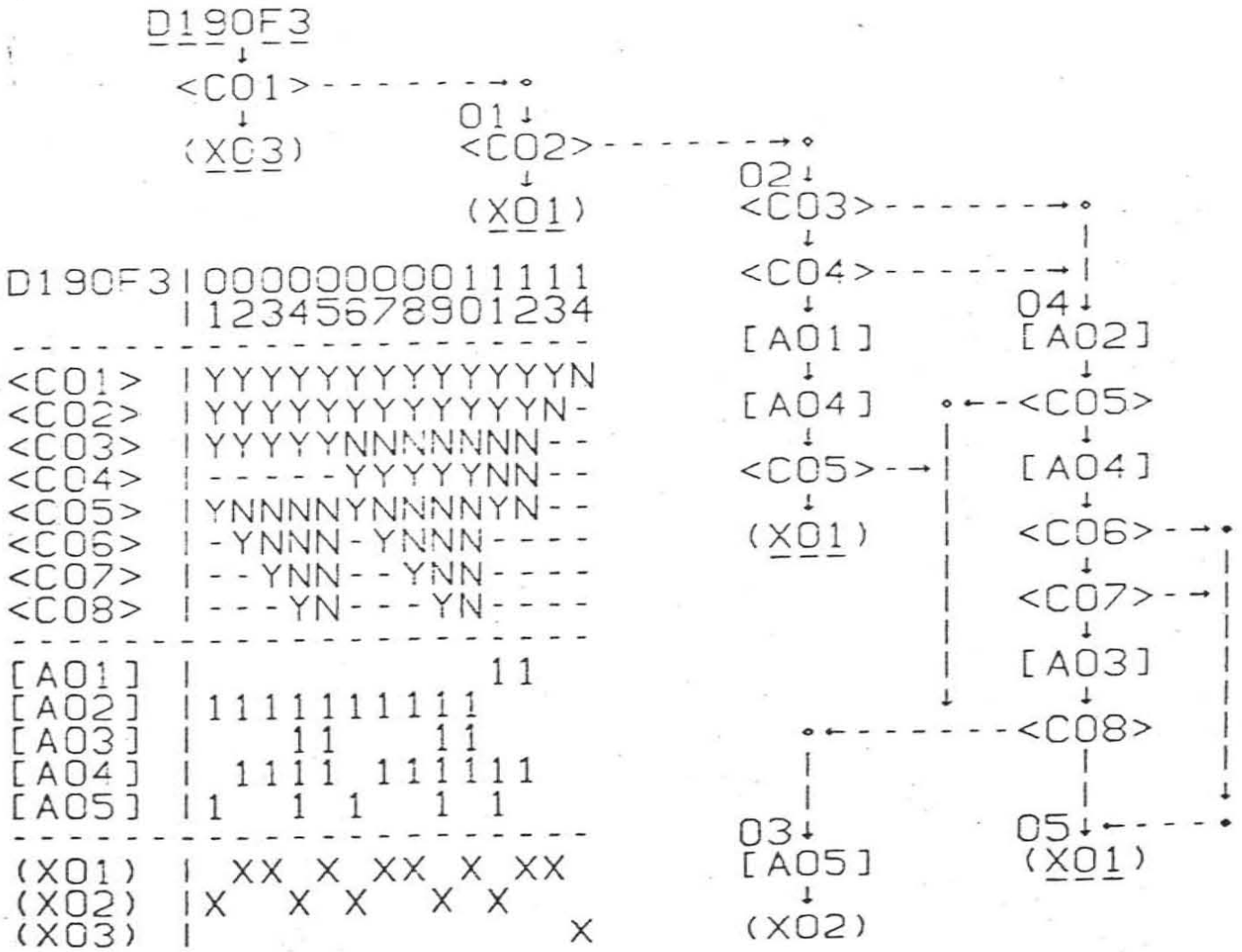
INSERTION
DELETION
MODIFICATION

BY

ROWS
COLUMNS
ELEMENTS

```
     OPTIONS-OPTM,PL1
     COMPILE D-D190F3
D190F3                          |000000000011111
                                |123456789012345
--------------------------------------------------
ERROR                           |YYYYYYYYYYYYYYN
TM ERROR OXCCW4                 |YYYYYYYYYYYYYN-
RDBCK PARAM                     |YYYYYNNNNNNNN--
MOD JFCBIND2                    |-----YYYYYNN---
I/O ERROR                       |YNNNNYNNNNYN---
RDBCK PARAM AGAIN               |-YNNN-YNNN------
MOD JFCBIND3                    |--YNN--YNN------
I/O ERROR FOR BSF               |---YN---YN------
--------------------------------------------------
SET TO BSF                      |           11
BSP TWICE                       |1111111111
SET CCW1 TO BYPASS TM HDR       |   11    11
BSF                             | 1111 111111
SET UP ABEND 613                |1  1 1  1 1
--------------------------------------------------
D190G1                          | XX X XX X XX
ABEND                           |X  X X  X X
D190F4                          |             X
COMPILING D190F3
 20 LINES COMPILED IN 1 CPU SECOND
.....

D190F3: BEGIN;
          IF ERROR THEN GO TO D190F301;
          GO TO D190F4;
D190F301: IF TM ERROR OXCCW4 THEN GO TO D190F302;
          GO TO D190G1;
D190F302: IF RDBCK PARAM THEN GO TO D190F304;
          IF MOD JFCBIND2 THEN GO TO D190F304;
          SET TO BSF;
          BSF;
          IF I/O ERROR THEN GO TO D190F303;
          GO TO D190G1;
D190F303: SET UP ABEND 613;
          GO TO ABEND;
D190F304: BSP TWICE;
          IF I/O ERROR THEN GO TO D190F303;
          BSF;
          IF RDBCK PARAM AGAIN THEN GO TO D190G1;
          IF MOD JFCBIND3 THEN GO TO D190G1;
          SET CCW1 TO BYPASS TM HDR;
          IF I/O ERROR FOR BSF THEN GO TO D190F303;
D190F305: GO TO D190G1;
        END /* OF D190F3 */;
```


D190F3
<C01>
(XC3)
O1
<C02>
(XO1)
O2
<C03>
<C04>
[A01]
[A04]
<C05>
(XO1)
O4
[A02]
<C05>
[A04]
<C06>
<C07>
[A03]
<C08>
O3
[A05]
(XO2)
O5
(XO1)
D190F3 0000000001111 1234567890123 4
<C01> YYYYYYYYYYYYYN
<C02> YYYYYYYYYYYYN-
<C03> YYYYYNNNNNNN--
<C04> -----YYYYYNN--
<C05> YNNNNYNNNNYN--
<C06> -YNNN-YNNN----
<C07> --YNN--YNN----
<C08> ---YN---YN----
[A01] 11
[A02] 11111111111
[A03] 11 11
[A04] 1111 111111
[A05] 1 1 1 1 1 1
(XO1) XX X XX X XX
(XO2) X X X X X
(XO3) X
0 TESTGEN 0
1 2 3 5 9 11 12 13 14

QUESTIONNAIRE PROCESSING

```
      Q DISCOUNT
QUANTITY>100? YES
QUANTITY>500? NO
PREFERRED CUSTOMER? YES
----------
APPLY 10 PCT DISCOUNT
APPLY  5 PCT DISCOUNT
DO BILLING NEXT
```

```
      Q DISCOUNT
QUANTITY>100? YES
QUANTITY>500? YES
PREFERRED CUSTOMER? NO
----------
APPLY 20 PCT DISCOUNT
DO BILLING NEXT
```

```
      Q DISCOUNT
QUANTITY>100? NO
PREFERRED CUSTOMER? YES
----------
APPLY  5 PCT DISCOUNT
DO BILLING NEXT
```

DISCOUNTS	000000
	123456
QUANTITY>100	NNYYYY
QUANTITY>500	--NNYY
PREFERRED CUSTOMER	YNYNYN
APPLY 10 PCT DISCOUNT	11
APPLY 20 PCT DISCOUNT	11
APPLY 5 PCT DISCOUNT	1 2 2
DO BILLING NEXT	XXXXXX

Reprinted from IBM Journal of research and development

Vol. 16 | No. 5 | September 1972

H. J. Myers

Compiling Optimized Code from Decision Tables

H. J. Myers

Compiling Optimized Code from Decision Tables

Abstract: This paper reviews the structure of decision tables and methods for converting them into procedural code. It describes new optimization methods, which are applied before, during, and after code generation. Some results from an experimental decision table processor are provided.

Introduction

Decision tables have been in use for over ten years, principally in business applications, to state problems that contain a relatively high proportion of programmed tests. Numerous compilers have been built that convert the logic expressed in decision tables into algorithms that are executable by computer. Emphasis in decision-table compilers has typically been placed on producing logically correct code, on checking the decision table for completeness and consistency, and on ordering condition tests for efficient execution. Most decision table compilers produce code that is in a higher-level language, leaving optimization of the produced object code up to the high-level language compiler. However, the structure of the code produced by typical decision-table compilers is of a type that is improved little by any of the optimizing algorithms used by commercial compilers today.

General-purpose support programs (systems programs) are also typified by a high proportion of programmed tests. It would therefore appear that effective use of decision tables could be made in describing systems programs. However, systems programs must also be comprised of highly optimal code. This paper describes some decision-table compiling algorithms that provide a program structure of high enough quality to satisfy most systems programming needs. The output can be used either by a post-processing compiler or by a programmer as a guide in hand coding.

For the project, we constructed a running compiler and support system into which were introduced numerous decision tables based on actual systems programs. From these decision tables, procedural code (in PL/I format) was produced. Some samples of the code structure produced are included in the Appendix to enable the reader to judge the effectiveness of the compiler. The system was produced in an interactive API environment, which allows considerable flexibility in revising and augmenting algorithms.

In the paper, we first review the structure of decision tables and the procedures used to map them into code. However, the main emphasis is on the optimization methods we use before, during, and after code generation.

Decision tables

In order to make this paper reasonably self-contained, we review here the structure of decision tables and the general methods used to convert them into procedural language. We attempt to emphasize those aspects of the process that provide opportunities for optimization.

A sample decision table is shown in Fig. 1.

The *stub* portion (2) of the table gives descriptions of *conditions* (5,6) *actions* (7,8) and *exits* (9,10). The format of the stub contents is generally constrained by the target language into which the table is being translated. Our processor places no constraint on the contents of the stubs; if correct PL/I code is to be produced, the condition stubs must contain one PL/I relational expression each; the actions should contain a PL/I assignment statement, call statement (or other nonbranching execut-

Figure 1 Sample decision table.

able statement); and the exit stubs should contain only a valid PL/I name. The table name should also be valid in PL/I.

Each column in the *entry* portion (3) of the table represents a *rule*. Rule numbers (two-digit numbers read vertically) and the table name appear in the *table header* (4). Rule 02 in Fig. 1, for example, indicates that if a *NAME FIELD IS PRESENT* (Y means yes) *and* if *OPERAND 1 IS NOT IN REGISTER NOTATION* (N means no) *and* if *FILE IS NOT DEFINED*, then the actions taken are *GENERATE NAME* and *ERROR 2* (in that order), followed by an exit to *FINISH*. In general, a rule specifies that for a unique combination of conditions, some selected actions are performed and a selected exit is taken.

The decision-table representation of logic does not impose any strict ordering on the sequencing of condition tests. Furthermore, actions in a given table are not allowed to modify factors that would cause a change in the outcome of a condition test in the same table. This gives the compiler more latitude in selecting an optimal ordering of condition tests with respect to each other and with respect to actions.

The ordering of actions with respect to each other can be loosely defined. In Fig. 1 the order required is specified by an integer opposite a selected action. If there is no integer, the action is not selected. Within a given rule, actions are executed in the order specified (e.g., in rule 07 *GENERATE NAME* occurs before *ERROR 1*). Exactly one exit is taken after execution of the actions of the selected rule.

In this experiment, we consider limited-entry type decision tables, in which the value of a condition is limited to yes or no. The alternative, extended entry tables, allows values that are numbers or number ranges. However, as Press [1] demonstrates, these can be converted into limited-entry tables, so that our methods apply to both types of decision tables.

The occurrence of a third value (—) in rule 04 of Fig. 1 does not contradict the two-value restriction. Rule 04 is actually a condensation of two rules (say 4a and 4b), which have identical action and exit entries, but whose condition entries are:

Condition	*4a*	*4b*	*4c*
1	*Y*	*Y*	*Y*
2	*Y*	*Y*	*Y*
3	*Y*	*N*	–

Rules 4a and 4b mean that the specified actions and exits are to be performed when conditions 1 and 2 are both yes, regardless of the value of condition 3. Therefore in rule 04, condition 3 is immaterial—a don't-care condition. As will be shown later, in the code produced by the compiler, there will be no test for condition 3 in the flow path for which conditions 1 and 2 are both true.

Because conditions in a limited-entry table are binary valued, there are 2^n unique combinations of the values of n conditions, and therefore 2^n rule columns. In practice, the number of rules is greatly reduced by the introduction of don't-care entries and consolidation of the rules as shown for rule 04. If rules 4a and 4b had been counted as part of the table in Fig. 1, that table would have 8 (2^3) rules for 3 conditions. A detailed discussion of column combining is presented later in the section on pregeneration optimization.

With the 2^n requirement on the rule count, one can program a verification of the rule entries that checks for both consistency and completeness. However, as will be seen, the check does not extend to an interpretation of the stub information and so it cannot be regarded as a complete check.

Decision-table code production

One of three methods is generally used to map decision table logic into procedural representation. In the *rule mask* method [2–4] every condition is first tested and then a *selection mask* is created. Next, for each action and exit, an *action mask* (created at compile time) is compared to the selection mask. If the masks coincide, the action (or exit) is performed.

In the second method, a variation on the rule-mask method, a unique power of 2 is assigned to each condition, which is then tested and its number added to a counter if the condition is true. At the end of condition testing, the counter is used as an index into a branch table and control is transferred to the appropriate action-exit sequence. (Note that the branch table must have 2^n

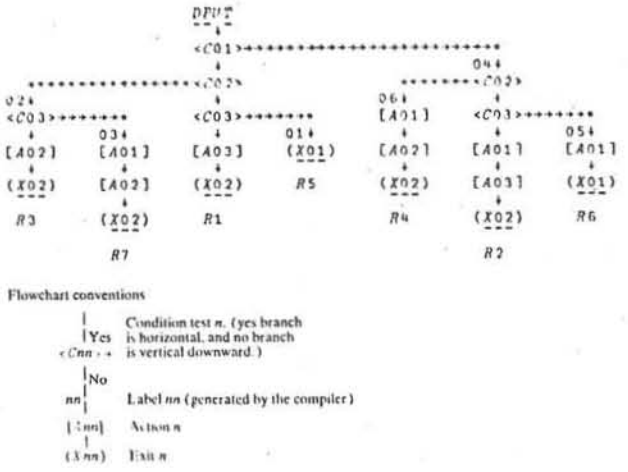


Figure 2 Tree corresponding to sample decision table.

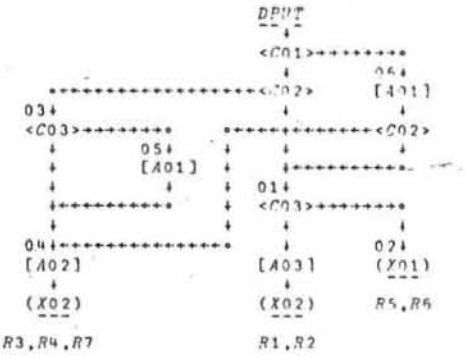


Figure 3 Optimized tree.

Figure 4 Optimized listing.

```
LINE                                    BRANCH
NUMBERS LABELS OPERATIONS OPERANDS
                                        
000|    C01→06
001|    C02→03
002|01: C03→02
003|    A03
004|    X02
005|02: X01
006|03: C03→05
007|04: A02
008|    X02
009|05: A01
010|    ---→04
011|06: A01
012|    C02→04
013|    ---→01
```

Note: *Cnn*→ indicates a conditional branch, and ---→ indicates an unconditional branch

Figure 5 Concrete listing.

```
DPUT:   BEGIN;
        IF NAME FIELD PRESENT THEN GO TO DPUT6;
        IF OPERAND 1 IN REG. NOTATION THEN GO TO DPUT3;
DPUT1:  IF FILE DEFINED THEN GO TO PUTTWO;
        ERROR 2;
        GO TO FINISH;
DPUT3:  IF FILE DEFINED THEN GO TO DPUT5;
DPUT4:  ERROR 1;
        GO TO FINISH;
DPUT5:  GENERATE NAME;
        GO TO DPUT4;
DPUT6:  GENERATE NAME;
        IF OPERAND 1 IN REG. NOTATION THEN GO TO DPUT4;
        GO TO DPUT1;
        END;
```

entries for n conditions.) These two methods require that all tests be performed regardless of the don't-care entries in the table. They tend to produce (when bit masks are used) a small program that runs longer than is usually necessary.

We use the third scheme, called the *condition tree* method, which causes the generation of a tree-structured program with condition tests at each node. Each action-exit sequence is placed at the leaves. Figure 2 shows an unoptimized tree corresponding to the decision table of Fig. 1. The rule numbers for each action-exit sequence are placed under the leaves for clarity.

Notice the one-to-one correspondence between the leaves of the tree in Fig. 2 and the rule columns of the decision table shown in Fig. 1. Notice also the large amount of redundancy in the code generated. In this case, the penalty for this redundancy is not lengthened execution sequences, but excessive storage consumption. Figure 3 shows the effect of the optimization algorithms on the tree of Fig. 2.

Figure 4 shows an *abstract* listing of the program diagrammed in Fig. 3. This listing is called abstract because nothing in it refers to the actual (concrete) conditions, actions, or exits that are described in the table stubs. In fact, the processing algorithms ignore the contents of the stubs except for listing purposes.

Because we are principally interested in compilation, the abstract format is the one used in this paper. However, with little difficulty, concrete listings can be produced for any procedural language.

Figure 5 is a concrete listing in PL/I format of the program shown in Fig. 4. Because the stubs in the original table (Fig. 1) do not follow PL/I conventions, the result is *not* a PL/I procedure.

• *Code generation*

Various methods exist for the generation of tree-form code from decision tables [5–7]. Briefly, one selects (by criteria described later) a condition. The rule columns are then partitioned into two groups—a no-group and a yes-group—according to the values in each column of the selected condition. If the condition value is don't-care in a particular column, that column is placed in both

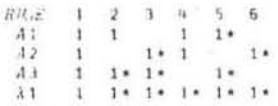


Figure 6 Cross-linked action-exit sequences.

groups. Code is then produced for the selected condition. Assume that the line

$Cnn \rightarrow$ label $- m$

is generated, where *label-m* is a created label. The code generator is then re-entered recursively with a subtable consisting of the no-group of rule columns, but with the selected condition row removed from them. Next, the generated label (*label-m*) is produced and the generator is re-entered with the yes-group columns (without the selected condition row). The resulting structure appears as follows:

$Cnn \rightarrow$ label $- m$
no – group
Label – *m*: yes – group

Because every action sequence terminates in an exit, control will not flow from the no-group into the yes-group.

It can be seen that the yes- and no-groups are composed of substructures analogous to the main structure. Recursion is terminated when there is no condition remaining to be selected. At this time, there should be exactly one rule column remaining. The action-exit sequence is then produced from that column. Because of the presence of don't-care conditions, a special check must be made before generating each conditional branch. This consists of locating any column in which all of the conditions have don't-care entries. If such a column is found, its action-exit sequence is generated and recursion terminates.

Other special tests are made for handling *else* rules and optimization procedures, which will be discussed later.

An extension of this algorithm, called *cross-linking*, is used in the Preprocessor for Encoded Tables (P.E.T.) processor [8] to reduce the number of instances of actions generated. Each action-exit sequence is compared to the action-exit sequences to its left. If some action and its selected successors exactly match the corresponding elements in some rule to the left, they are linked leftward. For example, in Fig. 6, the cross-linked action-exits are marked by an asterisk. When code is generated, the first linked action is replaced by a branch to the action corresponding to it in the rule it is linked to. This reduces redundancy in the generated actions, providing better code than other processors [8,9].

Most published accounts concerned with object code optimization concentrate on the order of condition testing. Reinwald and Soland [10,14] give algorithms (supported by proofs) for generation of optimal object code with respect to time and space. These algorithms require that storage and time costs be associated with each condition test, but do not take into account the optimization of action sequences.

The P.E.T. processor reduces redundancy in action sequences (a space optimization) but does not remove redundant testing sequences. (It relies upon the table-writer to order condition tests and to introduce don't-cares to provide adequate optimization of testing.) P.E.T. also allows the table writer to place actions into the condition portion of the table. This is called *preconditioning*, and is allowed when an action is to be performed regardless of the outcome of the conditional testing that it precedes.

In summary, previous processors have placed the optimization burden on the user to a large extent. This partially reduces the principal benefit of the decision table – the separation of the logic required from the procedural mechanism that implements it.

A recent paper by Dailey [11] has been published since this work was concluded. Although his approach is different, his optimizations seem to overlap some of those presented here. A careful comparison of his methods to ours has not yet been carried out.

The methods described in this paper attempt to optimize with a minimum amount of guidance from the user. Optimization processes are applied before, during, and after code generation, with major emphasis on the final optimization process. The basic generation process produces excellent code from the point of view of time optimization. It generates no unnecessary tests in a given test sequence. It could be further improved if frequency information were provided (see [12] for the method). Most of our optimization methods are aimed at reducing space requirements. Our experience shows considerable success in removing redundant code, and the program structure is suitable for systems programs.

Optimization techniques

Our pregeneration optimization consists of consolidating rule columns and introducing don't-care values into the rule portion of the table. The optimization methods that are carried out during code generation are concerned with the ordering of the condition tests and the "factoring" of actions. (The latter technique is called *preconditioning* in [8], but is more commonly known in compiler-writing circles as "hoisting.") Postgeneration optimization procedures include removal of duplicate sequences of code and consolidation of duplicate flow paths. These procedures often leave dead code, branch

chains, and other dross in their wake, which is cleaned up by some *scavenger* optimization methods. A final optimization process was introduced into the program that we use to create a concrete PL/I listing. This process is an example of a scavenger procedure. Its purpose is to remove unneeded conditional branches to exits, which are themselves branches. These optimization methods are discussed in the succeeding sections.

• *Pregeneration optimization*

The first optimization method used is that of consolidating rules where possible so as to introduce don't-cares into the condition entries. We call this the *merge* process. Figure 7 shows the initial state of a decision table, which we will use to illustrate the process.

Two rule columns can be combined into one if:

- They are identical except for one condition entry, and
- In the differing entry, one column has *Y* and the other has *N* (neither is a don't-care).

The algorithm used first groups rule columns according to action-exit sequences. In Fig. 7 the grouping gives

```
(07 08 09) (10 11 12 12 14 15 16)
```

(Note that groups containing only one column are ignored.) It then forms subgroups according to don't-care patterns. If two rule columns were

```
Y Y
N Y
- -
Y N
```

they would have the same don't-care pattern and would be put into the same subgroup. Because there are no don't-cares in the table in Fig. 7, all elements of each group have the same don't-care pattern, so that the groups default to subgroups. Within each subgroup, the condition entries are compared to determine if a difference exists in exactly one position. Two columns from Fig. 7 that qualify can be combined as follows:

```
1 1                 1 1
4 5                 4 5
Y Y                 Y
N N                 N
Y Y                 Y
N Y (DIFFERENT) - (DON'T CARE)
```

The value at the position of difference is replaced by a don't-care value and one of the rules is discarded. Once a pair of rules is consolidated, it is removed from the subgroup because its don't-care pattern has changed. The remaining columns in the subgroup are processed similarly until no further combinations can be found.

```
PUT1|0000000001111111
    |1234567890123456
--------------------
C1  |NYNYYYYNYNNNNNYYY
C2  |NNYYYNNYYYNNYYNNY
C3  |YNNNNYNNNNNYYYYYY
C4  |NNNNNYYYYYYNYNYYY
--------------------
A1  |          1111111
A2  | 1 11
A3  |1
A4  |  111
A5  |      111
A6  |     1111
A7  |11111111
A8  |    1
--------------------
X1  |XXXXXXXXXXXXXXXX
```

Figure 7 Decision table initial state.

After all subgroups have been processed, the combined columns are regrouped and reprocessed. This iterative process continues until no new combinations are possible. (Note that the particular combinations that occur depend upon the order in which the columns are matched.)

The effect of this optimization on basic code generation (in the absence of other optimization methods) is to remove unnecessary condition testing. It therefore improves both the space and time costs of the resulting code.

Figure 8 shows the results of the merge process and examples of its effects, as well as the effects of different ordering (in the absence of other optimization methods). On the other hand, in the presence of other optimization procedures, the effect is reduced because of overlap of optimization function. This pregeneration optimization gives the user a better insight into his decision table. It also reduces, at an early stage, the amount of work that the other optimizers have to do. This method generally results in a net reduction in compile time.

Both of the compilers that we examined [8,9] require that the user perform this optimization in order to produce better code. It is often convenient for the user to do part of this optimization himself when the situations giving rise to don't-cares are patently obvious. (It allows him to greatly reduce the number of rule columns he has to deal with.) The use of don't-cares also can be a trick that allows conditional test dependencies (a violation of decision-table ground rules); therefore the implemented merge function was designed to leave original don't-cares intact.

It is instructive to compare our merge process to the electrical engineering problem of circuit simplification. Recall that we identified two groups from Fig. 7 having the same action-exit sequence. Consider rewriting the condition entries of the second of these groups as follows so that they look like a Boolean expression.

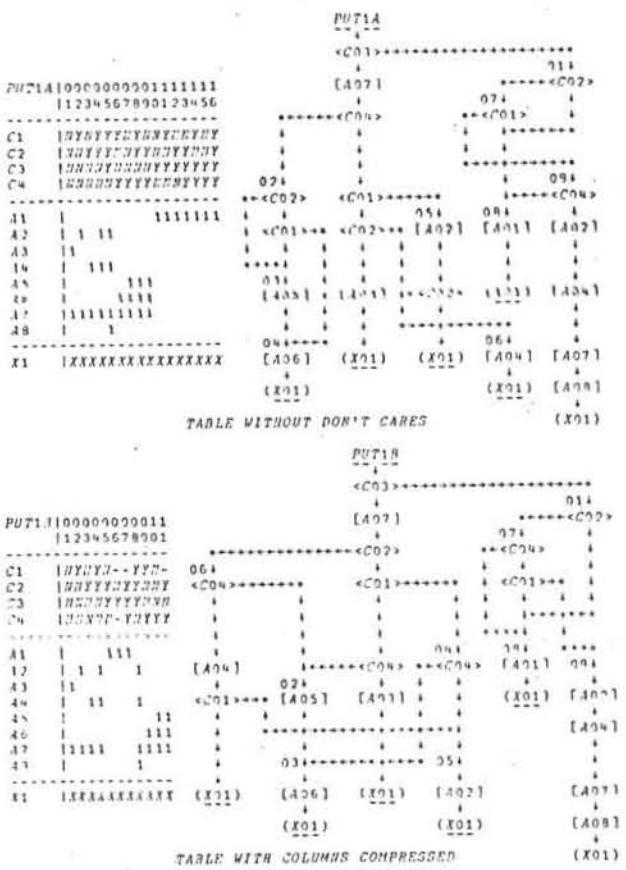


Figure 8 Effects of merge process.

1 1 1 1 1 1 1
0 1 2 3 4 5 6
N N N Y Y Y Y
N N Y Y N N Y
Y Y Y Y Y Y Y
N Y N Y N Y Y

The equivalent Boolean expression with A, B, C, and D representing conditions 1 through 4, respectively, is:

$$\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}B\bar{C}\bar{D} + \bar{A}BCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}CD + ABCD.$$

Note that each rule represents a term in the expression, and that all rules are in effect ORed (+) together. The elements within a rule column are ANDed. The decision table effectively states in this case, "If *all* of the conditions of *any* column are met, the common action/exit sequence is performed."

Now if we use the Quine-McCluskey technique [13], to find the optimal Boolean expression, the result is:

$$\bar{A}C + \bar{B}C + CD.$$

The equivalent decision-table rule columns are:

N - -
- N -
Y Y Y
- - Y

Investigation of the code generated from a table so optimized shows no improvement over that for the merge method. This is because the Quine-McCluskey method introduces don't-cares that ultimately cause the code generator to place a rule column in two subtables. It can therefore be concluded that Quine-McCluskey optimization methods are not applicable to the decision-table optimization process. Note that the first two columns actually overlap (e.g., the condition sequence *NNYY* could apply to either one of them). Indeed, if it were not for the fact that both columns specify the same action-exit sequence, the decision table would be inconsistent.

• *Optimization during code generation*

The code generation process we used has already been outlined. However, the method of selecting the "next test" was deferred to this section since it has an impact on the quality of the code produced.

The most significant work published relating to code optimization is, in the author's opinion, that of Reinwald and Soland in Refs. [10] and [14]. Primary papers have also been published by Montalbano [6] and Pollack [7], and contributions made by King [2] and Press [1]. Most recently, Shwayder [12] extends the work of Pollack. King describes the rule-mask technique, which produces compact code but requires that all condition tests be performed regardless of the logic needed. The other papers deal with optimizing methods for sequential testing procedures (the condition-tree method).

Press takes advantage of an *else* column, reducing the number of instances of tests to a minimum (both statically—presence in storage—and dynamically—presence in a flow path). However, the Reinwald and Soland work is the most general and requires that the time and space costs of performing tests be included as input to the optimization process. Reference [10] describes time optimization and Ref. [14] describes space optimization. Both papers provide formulas for calculating the extra cost involved in performing test i after test j has been performed. They next demonstrate a means for determining a lower bound on the extra costs of all tests performed after test j. Then they provide an algorithm for searching a subset of all possible generated testing sequences to find the sequence with minimal lower bound. They prove this sequential procedure to be optimal *in terms of the number of tests*. The costs of actions were not considered. This, we presume, is because action

sequences were considered by them to be atomic units and therefore could not be subjected to reorganization. It should be noted that the time consumed by the search algorithm goes up rapidly with the number of condition tests in the table. (This approach falls in the area of combinatorial mathematics and is akin to the "traveling salesman" problem.)

Our initial approach to test selection followed along the lines suggested by Pollack [7]. It was the simplest, and our attention was focused on postgeneration optimization—a subject treated only lightly elsewhere. The first criterion was the selection of the condition row with the fewest don't-care values. Beyond that, if a tie had to be broken, the row was selected that had the minimum difference between the number of *Y*'s and *N*'s (again on the advice of Pollack). Later this was compared to the Press method.

It was found that the Press method

- Required more compile time,
- Did not improve object code in the presence of postgeneration optimization, and
- Interfered with "hoisting" optimization steps.

We felt that concentration on postgeneration optimization should be continued, and no further effort was expended on enhancing condition test selection. Clearly, further investigation is warranted. The original selection algorithm is therefore retained. Because an else-column capability was desirable in a decision-table processor, it was provided. The implementation was simply to construct the missing condition rule entries and to supply them with a common action-exit sequence (specified by the user). These added columns are compressed via the merge routine and require no subsequent special treatment. Test selection and hoisting are applied equally to all columns.

Schwayder [12] shows how to incorporate frequency information into test selection. Although this was not incorporated into the processor, the impact on our current generation technique of including Schwayder's algorithm was investigated. If the frequency information specifies the rule frequencies, only one line of APL code need be added to take them into account. Another line of APL code would be required if condition test frequencies are given. More effort would be required to incorporate frequency specification in the decision table format than in making use of it in code generation. Inclusion of optional frequency information is recommended for follow-on work. (As will be described later, some postgeneration optimization methods may destroy some of the effectiveness of test selection.)

The second optimization performed during code generation is that of "hoisting." When one or more actions are to occur in all flow paths following from a single

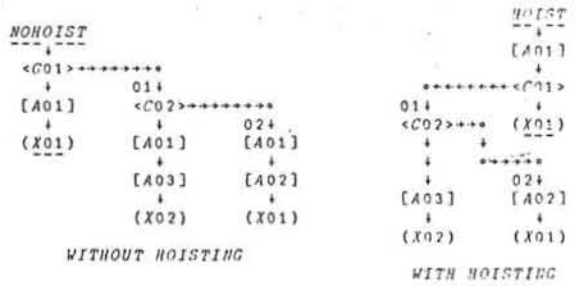


Figure 9 The effects of "hoisting."

condition test, it does not destroy program logic to move (hoist) these actions to a position in front of the condition test. This is a space optimization, because duplicate instances of these actions can be removed from all successor paths to the test. Figure 9 demonstrates the effects of hoisting.

It should be noted that an action cannot be hoisted past another action that must precede it in sequence.

Hoisting can most readily be performed during code generation. Just prior to selection of a condition test, the subtable is examined for an action that:

- Is performed in all remaining rules, and
- Is not required to follow an action that is not performed in all the remaining rules.

Any actions that fulfill these criteria are immediately generated and removed from the subtable.

The action entries accepted by the processor allow the user to indicate an ordering, a *lack of* ordering, or a partial ordering requirement on the actions. This is a degree of freedom not provided in procedural descriptions.

It is not always easy for the user to recognize hoistable actions because he cannot easily recognize the subtables. However, he always has a clear picture of the ordering requirements (or their lack) on the action. Automatic hoisting lets the user specify his logic requirements in terms most easily understood by him.

Unfortunately, hoisting can, on occasion, exert a negative influence on duplicate sequence removal, an optimization procedure that occurs after code generation. This problem is discussed after the description of the affected optimization. But a way has not been discovered to detect the situation without exhaustive (time consuming) combinatorial analysis of the entire table. Fortunately, in the practical examples we have examined, hoisting is more often good than bad. (It seldom had any effect on duplicate sequence removal.) We explicitly resist taking the route of the P.E.T. processor, which requires that the user control this optimization. As mentioned earlier, post-generation hoisting might improve the situation, but was not investigated deeply.

```
003|    C02→16          035|    C02→02
004|    A01             036|    A01
005|    A02             037|    A02
006|02:C03→03          038|16:C03→17
007|    X01             039|    X01
008|03:A07             040|17:A07
009|    C07→25          041|    C07→13

010|    ---→18          042|18:A04
```

Figure 10 Possibly duplicate sequences.

• *Postgeneration optimization*

The two principal postgeneration optimization procedures are *duplicate sequence removal* (DSR) and *duplicate path removal* (DPR). In addition, "scavenger" optimization techniques remove

- Dead (unreferenced) code,
- Redundant condition tests,
- Redundant unconditional branches (branch chains),
- Redundant exits.

The major effect of these optimization procedures is to save space.

• *Duplicate sequence removal (DSR)*

The code generation technique we use assures the performance of all testing required to isolate a rule. If don't-cares are inserted by the user or the merge process, then no tests are performed beyond those actually needed to isolate a rule. Note that rule frequencies are not taken into account, and that a reduced execution time cannot be assured. Average performance can be varied by changing the order of testing, but the minimum amount of testing required to isolate a rule cannot be varied [10]. Beyond reducing the tests to a minimum, one can attempt to restructure the generated code so as to reduce the number of duplicated code sequences. This is done by replacing one of the duplicate sequences with a branch to its equivalent (DSR). In this way a small time loss is introduced (to execute the branch) to save storage space.

DSR is easy to perform because the code generator produces code containing easily noticed patterns of duplicated code. Pairs of code sequences with identical operation codes are first isolated and then more carefully scrutinized (longer sequences first) to assure logical equivalence. If they are logically equivalent, one of them is replaced with a branch to the other. The compared sequences are considered not equivalent for the following reasons:

- They are dissimilar at any point,
- One sequence overlaps a portion of the other, or
- A portion of either sequence has been previously removed (because it was equivalent to some other sequence).

If any of these conditions occur, the unequivalent portions of the sequences are masked off and the remaining "good" portions are individually compared by a recursive procedure. Any duplicated sequence longer than one line of code is removed if it is logically equivalent to another.

Figure 10 shows two possibly duplicate sequences. Note first that lines 010 and 042 are not included in the sequences being compared, but are shown because they are germane to the detailed comparison. Operation codes in lines 003 through 009 match those of lines 035 through 041. The flow paths following these sequences pass to lines 010 and 042, which are seen to be equivalent. The branch paths from lines 003 (035) and 006 (038) are readily compared because they lie within the two sequences being compared. These can be certified as being logically equivalent by comparing the offsets of the branch targets from the beginning of the sequences. Note that in the case of the 003–035 pair the branch targets are across sequences but that this fact does not matter in the comparison. A lengthy analysis must be performed only on the paths emanating from the pair 009–041. This requires a line-by-line comparison of the code starting at label 25 with that starting at label 13.

The cross-link process performed by the P.E.T. processor is an attempt to eliminate duplicate code. Its effect, however, is to eliminate only common trailing portions of action sequences (those that end in an exit). Our algorithms remove all trailing sequences of redundant code, including redundant test trees. In addition, all non-trailing redundant sequences are removed if they flow into logically equivalent code. In general, the results have been very good when applied to actual system programs, as shown by the examples in the Appendix.

It was mentioned earlier that the hoisting optimization can have a negative effect on DSR, because hoisting of an action may remove it from one of a pair of duplicate sequences. When this happens, the pair no longer qualifies for consolidation. It is possible, for example, for hoisting to remove two lines and prevent the removal of, say, ten or twelve lines of duplicated sequences. On the other hand, the removal of an action from a sequence could also cause that sequence to match another, when it would not have done so otherwise. Some cases like this were actually encountered, such as Tables 8 and 9 in the Appendix. Through use of compiler options, selective elimination of hoisting is allowed. Further work should be done to try to establish an effective method for predicting the effect of hoisting on DSR.

Deferment of hoisting until the post-generation phase was considered. Hoisting at that time is much less convenient and will definitely lead to longer compile time. More information must be carried and maintained to keep track of the rule column(s) that were the source of

a particular action. The problem is made more complex by the folding actions of the other optimizers. On the other hand, reasonably good results were obtained with the algorithm used, and we judged the effort-to-payoff ratio too high to implement delayed hoisting.

• *Duplicate path removal (DPR)*

The second major postgeneration optimization technique is duplicate path removal (DPR). When two flow paths that emanate from a condition test have identical leading logic, the leading portion of these paths can be consolidated by moving the condition test down the paths to the point(s) where they differ. Figure 11 illustrates DPR.

As can be seen, DPR may cause reordering of condition testing. The method is to isolate as potential candidates those flow paths emanating from the same condition test and that start with the same operation code. Then a test is selected whose branch target and successor lines contain the same operation code. For each such test, the flow paths are compared and points of difference (POD) are located. (Note that if there are no POD's, the test is redundant and can be removed immediately.) The two paths emanating from the test will be referred to as the "fall-through path" and the "branch path." Conceptually, the process is as follows. Locate the POD's in each of the two paths. Remove the test and place a copy of it just in front of each POD in the fall-through path. The address of each test copied is changed to point to the corresponding POD in the branch path.

With the removal of the original test, it is expected (but not guaranteed) that a large portion of the branch path will become dead code. However, there is no simple way to determine this prior to performing DPR. Therefore a copy of the code is saved before performing DPR. If the code resulting from DPR shows improvement, it replaces the old code. If not the old code is restored. Some time was spent trying to develop some correlation between the amount of improvement and both number of POD's and number of lines in the duplicated paths. However, nothing developed that was useful. Furthermore, the conceptual algorithm described above did not work. This was because DPR was applied after DSR, and DSR could destroy the tree nature of the original unoptimized code. Therefore, DPR has to contend with the interesting possibility that the two flow paths being compared might merge, or cross over. In fact, some cases examined had a single line in the code turn up as POD's in two different paths. This caused the same test to be inserted twice in the same place.

The algorithm finally developed avoids the problems by making a separate copy of the fall-through path, and placing at each POD (in the duplicate) a copy of the original test with the target of the POD in the original

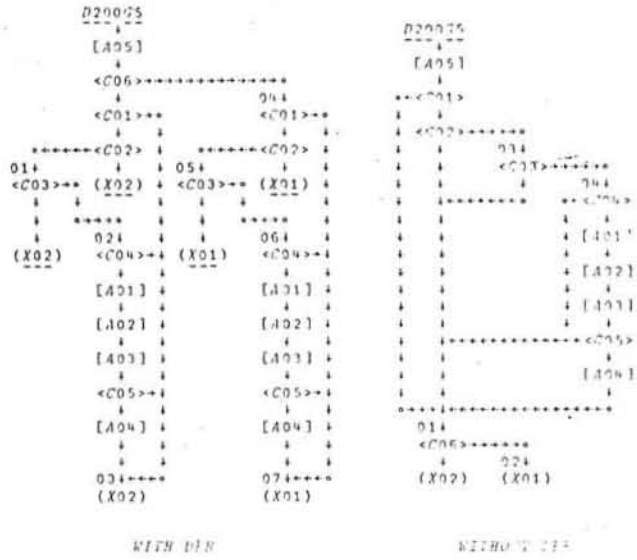


Figure 11 The effects of duplicate path removal.

branch path. This test is followed by an unconditional branch whose target is the POD in the fall-through path. At the end of the path comparison, the original test is replaced with an unconditional branch to the duplicated code, which is placed at the end of the code body. Both the branch and fall-through paths then become candidates for dead code removal.

Because DPR reorders condition testing, it may destroy the effectiveness of optimization procedures that depend upon test ordering. Since DPR never increases storage consumption, only time optimization can be adversely affected. This is a typical time-space trade-off situation. Unfortunately, the user may wish to trade off differently for different points of application of DPR within code from a single table. At present, sufficient information is not carried in the generated code to calculate the overall time costs. A recommended follow-on would be to try to include this.

DPR can recognize only those duplicate paths that emanate from the same condition test. Some tables that were processed contained duplicate paths that did not emanate from the same point and couldn't be removed by the processor. When these paths were removed manually, it was discovered that they could be removed only at the expense of inserting additional tests in the paths. The resulting code then would contain two tests for the same condition in a single path. This is typical of a situation in which the programmer would ordinarily set a switch for later testing. Table 6 in the Appendix contains this type of program structure. No algorithm was discovered that could expeditiously locate duplicate

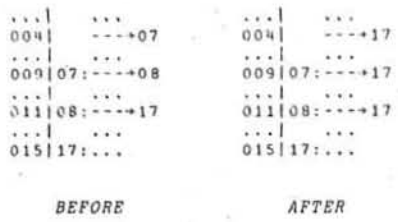


Figure 12 Branch removal.

Figure 13 Reordering.

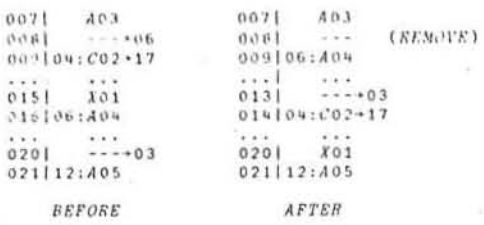


Figure 14 Backward branch movement.

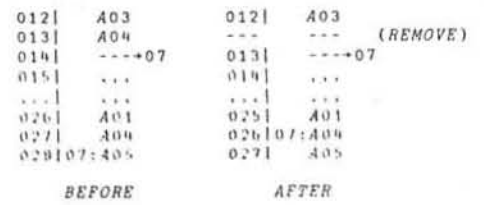
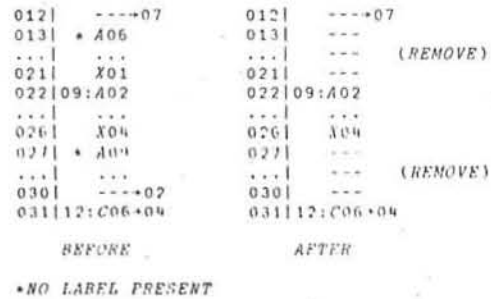


Figure 15 Dead code removal.



paths of this nature. This is also an area that needs further exploration.

• *Scavenger optimization*

Scavenger optimization procedures, as their name implies, clean up the leavings of other optimizers. The scavenger procedures to be described in the paragraphs below are

- Redundant branch removal,
- Redundant exit removal,
- Redundant test removal, and
- Dead code removal.

Redundant branch removal is a procedure for seeking out branches that receive control from other branches and removing them. (Sequences of directly connected branches are called branch chains.) Labeled branches are readily detected. All references (in other instructions) to the label on a branch are replaced with the branch operand, effectively removing the branch from the chain. The labels can then be removed. Figure 12 illustrates branch removal.

The second step is to locate all branches that are preceded by either an unconditional branch or an exit and to remove them. All branches whose targets are exits are replaced with the exits themselves. If a branch has a target that can receive control only from the branch (i.e., the target is preceded by an unconditional branch or an exit), then the branch can be eliminated by reordering the code. This is done, thereby placing the target and all code physically following it (up to a branch or exit) in place of the branch. Figure 13 indicates the method.

While cleaning up branches, this optimizer can perform one additional optimization step, which, strictly speaking, is not redundant branch elimination. It is a type of reverse hoisting. If the predecessor line to a branch target is the same as the predecessor to the branch, the branch predecessor can be removed, and the branch address reduced by one, to refer to the predecessor of the former target. Figure 14 shows this optimization method.

Redundant exit removal is a procedure to remove all exits of any one type that are preceded by a branch or another exit. Care must be taken, however, that there is always one exit of each type left. All conditional branches that have exits as their targets are set so that all references to an exit of one type refer to the same exit. This reduces the requirements for instances of exits to a minimum. (Note that unconditional branches to exits were eliminated by the branch optimizer.) No further processing by the exit optimizer is necessary.

Redundant test removal requires that the flow paths emanating from each conditional branch be examined. If the fall-through path is logically equivalent to the branch path, the test is eliminated. No other clean-up is performed by the test optimizer.

Dead code removal is a procedure that first removes all unreferenced labels. It then locates any unlabeled lines that follow exits or unconditional branches. These

lines are dead and can be removed. Any unlabeled line following a dead line is also dead and can be removed. All dead lines are located and removed at one time. However, because a dead line may have been a branch (conditional or unconditional), the removal of dead lines may give rise to more unreferenced labels. Therefore, the process is iterated until there are no more such labels. Figure 15 illustrates dead code removal.

• *Interaction of the optimizers*
The order of all optimization procedures is:

Pregeneration
    Merge
Generation
    Test selection
    Hoisting
Postgeneration
    Duplicate sequence removal (DSR)
    Scavengers
        Redundant branch removal
        Redundant exit removal
        Redundant test removal
        Dead code removal
        Duplicate path removal (DPR).

The postgeneration optimization procedures are iterative whenever their application can possibly introduce new program structure that would be susceptible to their further application. Further, the scavenger steps are called by both DPR and DSR, and DSR calls DPR. Specifically, DSR processes all of the sequences it can, then calls the scavengers. It then determines whether code was reduced. If so, it repeats. In the case of DPR, after each test instruction is processed, the scavenger procedures are called, followed by DSR. (Calling the scavenger procedures before entering DSR speeds up the latter.) Then DPR checks for code improvement as described earlier.

Concluding remarks

About seventy decision tables taken from actual application areas have been compiled by the system. The code for only two of these could be improved by hand. Furthermore, the compiler running times under the API system were short enough to allow the experimental model to operate as a production tool. Although there must be manual intervention between processing a decision table and the production of the final code, the processor as it stands has already proved to be a useful aid to some programmers by helping them organize their code. It can be concluded that the processor can be used *now* to gainful ends. If the algorithms described above were recoded into a more fully automated environment, program production could be improved even more.

As has been noted earlier, certain optimization problems are yet unsolved. These are:

- The interference among test-order selection, hoisting, DSR and DPR, and
- Duplicate path removal where the paths do not start at the same condition test.

Some additional optimization procedures could be added to the processor. These are:

- The use of frequency information in test-order selection,
- The use of timing information in conjunction with space/speed priority setting for DPR, and
- The reversal of condition tests.

An investigation of the benefits of post-generation hoisting should also be made.

The optimization methods described in this paper do not include all possible program optimization procedures. We have concentrated on optimization methods that are not usually done by higher-level language processors, namely, the gross arrangement of program flow structure. Specifically ignored are such optimization methods as loop analysis, common subexpression elimination, code motion (except for hoisting and forward code motion), and subsumption [15]. To perform these would require analysis of the information in the decision-table entries, and would require a restriction on the language permitted in the entries. It was felt that a wider service could be performed by providing a framework that would accept any language for entry statements. In this way, optimization procedures that were not performed could still be accomplished by passing the output from the decision-table compiler through another optimizing compiler. Minor revisions to the PL/I printing program can cause it to produce output acceptable to FORTRAN, ALGOL, and COBOL compilers (some of which perform optimization), and even to an assembler macroprocessor.

It should be noted that if a decision table has n rule-columns, then there are exactly n ways of traversing the generated program. This should suggest that exactly n test cases need be prepared to thoroughly test the produced program. Such a test battery would be guaranteed to execute every instruction in the program, and to execute every conditional branch for both yes and no conditions. Because the optimizers fold the program so that some paths through the program execute the (physically) same instructions, it is often possible to completely exercise the program with fewer test cases than the number of rule columns. A test-case generator could easily select a subset of the rule columns that would exercise all

of the code. It is recommended that such a generator be added to the system.

Acknowledgments

The author is indebted to D. H. Manning and P. C. Jacobs for first bringing the problems of decision table compiling to his attention. D. H. Manning, and R. E. Gaiduk are to be thanked for their enlightening discussions on the subject, and our thanks go to May Li and D. H. Manning for providing some "real" decision tables against which the processor was tested. R. H. Williams contributed to the solution of DPR problems.

Appendix: Samples of decision table compilations

This appendix contains decision tables and flow charts of code compiled from them. The reader is invited to browse them to obtain a subjective appraisal of the effectiveness of the compiler. The tables are representative of those examined during the development of the compiler. They were selected to demonstrate several points.

The range of table complexity,
Compile-time range,
The effects of various optimization procedures,
Comparison with other methods,
Some unsolved problem areas.

In addition to these points the reader should also be able to verify that:

There is a unique flow path through the object code for each rule column,
A given flow path contains no redundant tests,
Hoisting situations are not always readily seen in a decision table by the user,
The compiler produced correct code,
The code cannot be improved by hand except where noted.

Table 1 lists the sample decision tables, the time required to compile them on a System/360 Model 50, and the number of lines of object code produced.

In reference to Table 8, note that in some cases the order of condition row selection is arbitrary. When this happens, it is possible that different orderings will produce different amounts of final object code. The condition rules were reversed in D190F3 in Table 8 to create D190F3R. The code produced from D190F3R was 26 lines compared with 20 lines produced from D190F3. Careful examination of the produced code shows that the reversal of selection of conditions 3 and 5 (in the subtable for condition 2 = yes) caused a difference in the hoisting of action 4. When action 4 was hoisted (compiling D190F3), a pair of duplicate paths appeared, one of which was removed by the optimizer.

Table 1 Decision table compile times.

TABLE NAME	TIME (SECS)	LINES OF CODE
CHECK	15	17
CHECKM		15
D190F3	25	20
D190F3R	26	26
D190F4	50	31
D190F4M		26
D190P12	11	9
D200C10	7	7
D200C3	6	4
D200C9	2	3
D200G5	48	15
D200G7	19	18
D200N9	9	8
DCABCD	238	48
FED1	19	23
FED2	19	26
H9PET		10

Table 2 Examples of simple decision tables and their corresponding flow charts.

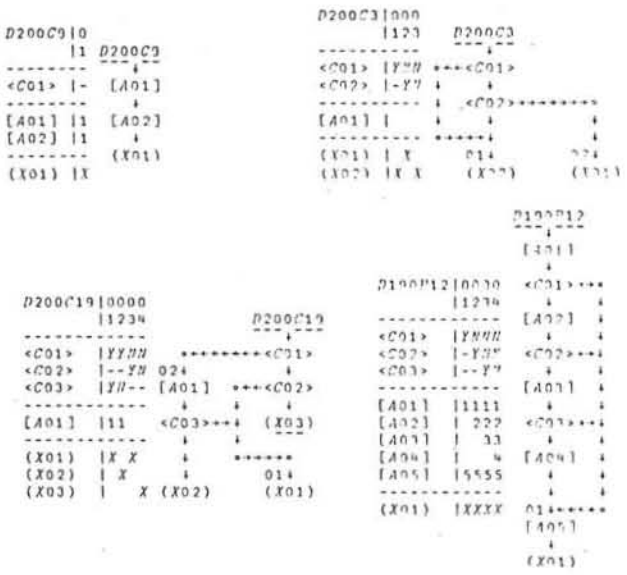


Table 3 A complex table.

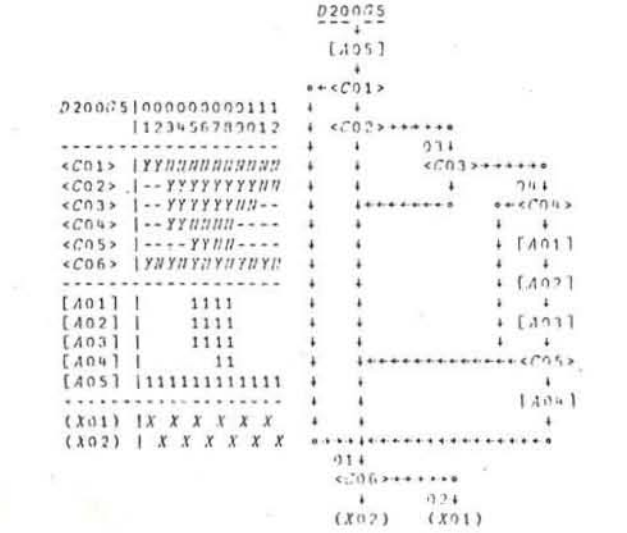


Table 4 Table compiled by our processor and by P.E.T.

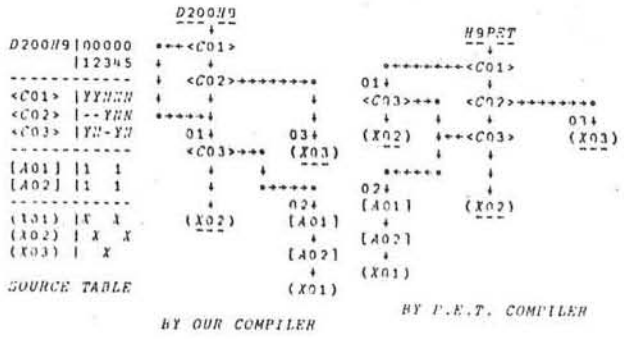


Table 5 A more complex table.

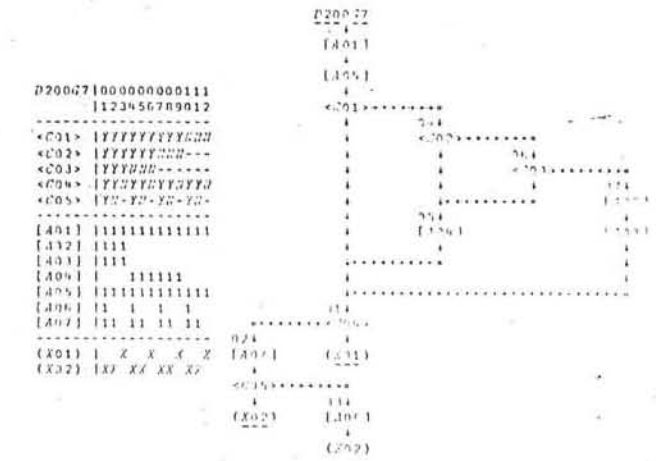


Table 6 Manually improved example.

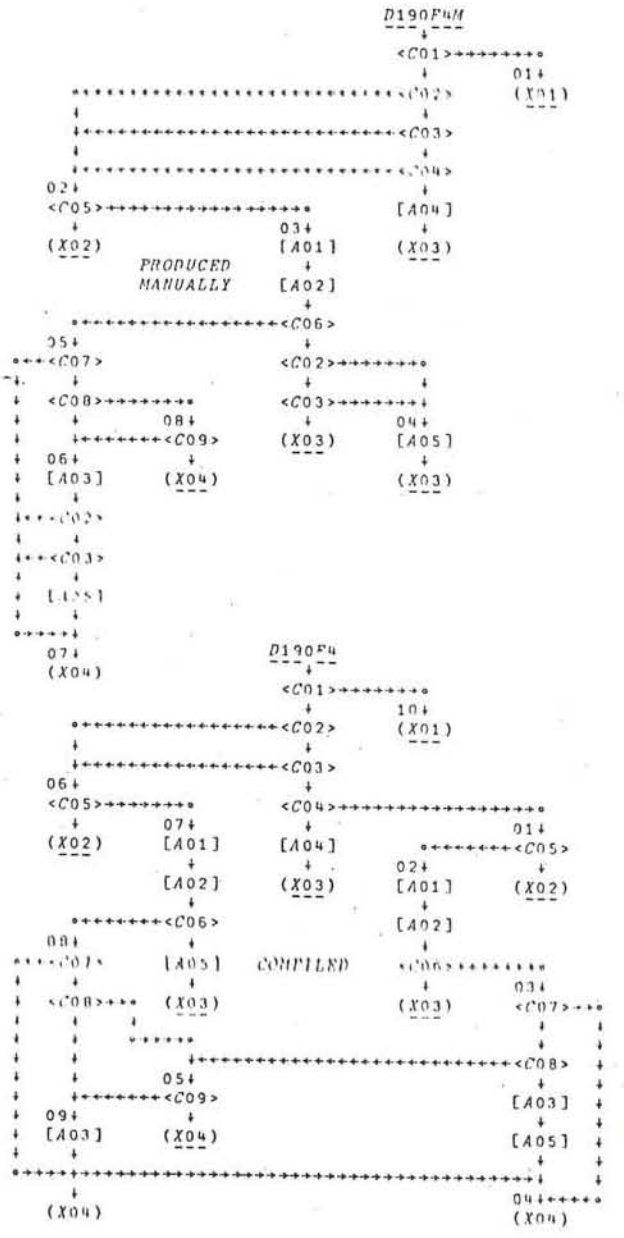


Table 7 Another example of manual improvement.

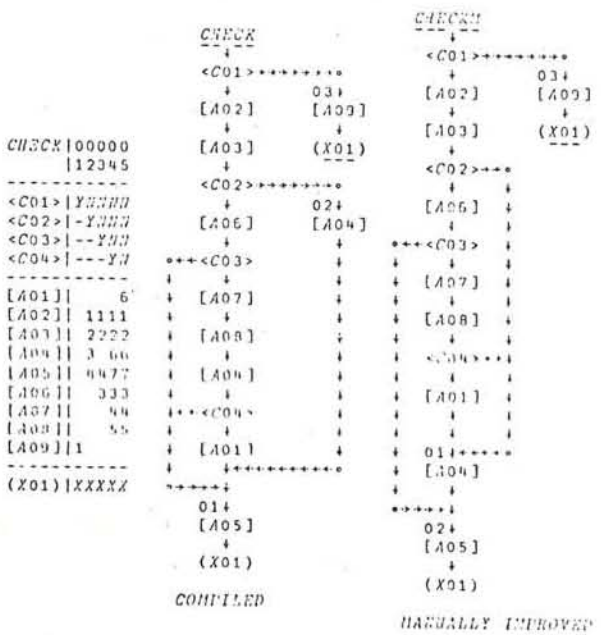


Table 8 Effects of arbitrary condition row selection.

Table 9 Effects of detrimental hoisting.

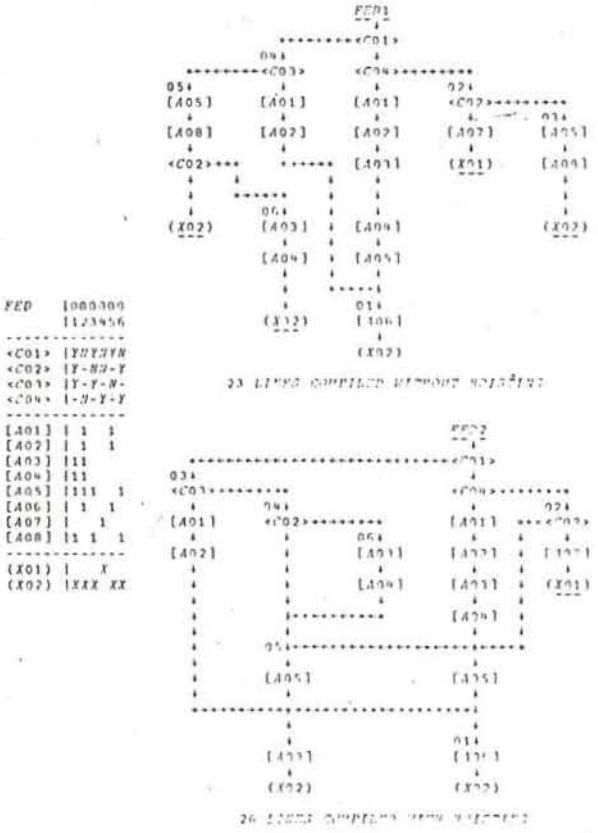


Table 10 A very complex table.

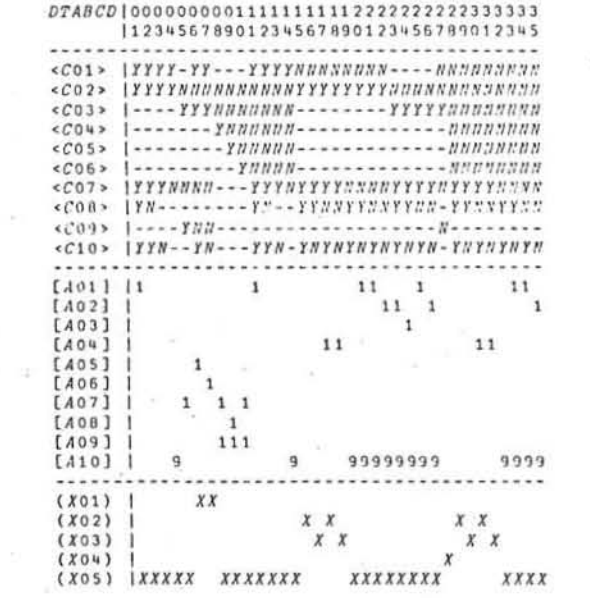
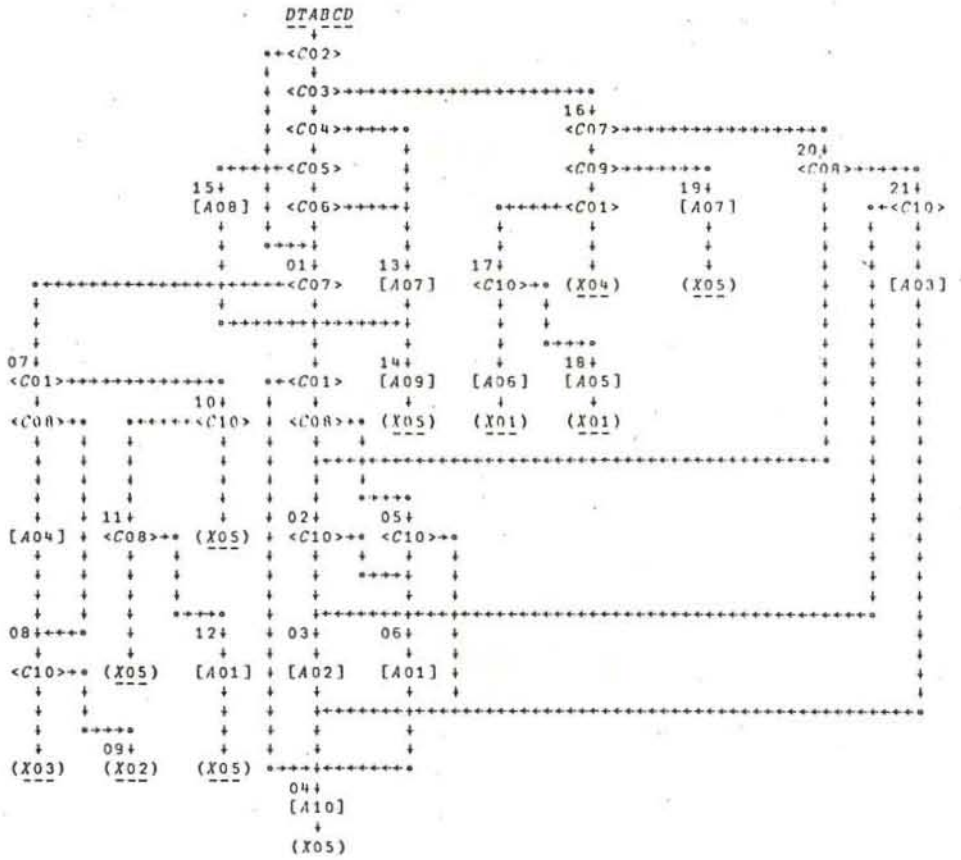


Table 10 A very complex table (continued).



References

1. L. I. Press, "Conversion of Decision Tables to Computer Programs," *Comm. ACM* 8, No. 6, 385–390 (June 1965).
2. P. J. H. King, "Conversion of Decision Tables to Computer Programs by Rule Mask Techniques," *Comm. ACM* 9, No. 11, 796–801 (Nov. 1966).
3. H. W. Kirk, "Use of Decision Tables in Computer Programming," *Comm. ACM* 8, No. 1 41–43 (Jan. 1963).
4. C. R. Muthukishnan and V. Rajaraman, "On the Conversion of Decision Tables to Computer Programs," *Comm. ACM* 6, No. 13, 347–351 (June 1970).
5. J. F. Egler, "A Procedure for Converting Logic Table Conditions into an Efficient Sequence of Test Instructions," *Comm. ACM* 6, No. 6, 510–514 (June 1963).
6. M. S. Montalbano, "Tables, Flow Charts, and Program Logic," *IBM Systems Journal* 1, 51–63 (Sept 1962).
7. S. I. Pollack, "Conversion of Limited Entry Decision Tables to Computer Programs," *Comm. ACM* 8, No. 11, 677–682 (Nov. 1965).
8. Bell Canada, "P.E.T. (Preprocessor for Encoded Tables) Processor, Users Manual".
9. H. B. Towne, LTjg USNR, "NAVTABTRANS-C", a computer program. NAVCOSSACT, 1969.
10. L. T. Reinwald and R. M. Soland, "Conversion of Limited Entry Decision Tables to Optimal Computer Programs I: Minimum Average Processing Time," *Journal ACM* 13, No. 3, 339–358 (July 1966).
11. W. H. Dailey, "Some Notes on Processing Limited Entry Decision Tables," *SIGPLAN Notes* 6, No. 8, 81–89 (Sept 1971).
12. K. Schwayder, "Conversion of Limited-Entry Decision Tables into Computer Programs," *Comm. ACM* 14, No. 2, 69–73 (Feb. 1971).
13. H. J. Myers and M. Y. Hsiao, "An APL Algorithm for Calculating Boolean Differences," *Proceedings of the IEEE Symposium on Error Recovery Systems* (1969).
14. L. T. Reinwald and R. M. Soland, "Conversion of Limited Entry Decision Tables to Computer Programs II: Minimum Storage Requirement," *Journal ACM* 14, No. 4, 742–756 (Oct. 1967).
15. E. S. Lowry and C. W. Medlock, "Object Code Optimization," *Comm. ACM* 12, No. 1, 13–22 (Jan. 1969).

Received February 18, 1972;
Revised April 14, 1972

The author is located at the IBM Systems Development Division Laboratory, San Jose, California 95114.

DATA RULES

PROCESS AREA CONTACT PERSONNEL	PREPARED BY	DATE / REVISION		PROJECT NR	DESCRIPTION

CONDITIONS | TABLE: MSYM10

Read in and place conditions in matrix

MSYM10A = ROWCTR * 5 + LOCROWS
X = MCURROW + 35

	TABLETYPE	EQ	'L'	N	N	N	N	Y	Y	
	NAME 2	EQ	BLANK	Y	Y	N	N			
	STUB OPR			EQ BLANK	UN BLANK	EQ VS	UN VS	EQ VS	UN VS	
MOVE	ON	TO		MEXT2	MEXT1	MVS	MLTD	MVS	MLTD	
MOVE	ON	TO		MSW2		MSW1	MSW3	MSW1	MSW3	
MOVE	+1	TO		CNTEXT	CNTEXT	CNTEXT	CNTLTD	CNTEXT	CNTLTD	
MOVE	+1	TO	BITSWTAG				X		X	
MOVE	BITSWTAG	TO	STUBITSW				X		X	
SET	CDCOLS1	EQ		'03 03 12'	'03 03 12'	'03 03 12'	'03 03 12'	'18 14 02'	'18 14 02'	
SET	MCURROW	EQ	MSYM10A	X	X	X	X	X	X	
SET	MISC3	EQ	X	X	X	X	X	X	X	
MOVE	IPSTUB	TO	MSTUB	X	X	X	X	X	X	
MOVE	+1	TO	ROWCTR	X	X	X	X	X	X	
MOVE	ZERO	TO	MISCTR	X	X	X	X	X	X	
GO TO	MSYM11A			X	X	X	X	X	X	

DATA RULES

PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA NR | PAGE / TO PAGE

CONDITIONS		TABLE	NEXT	ACTIONS
		msym12		Read in and place conditions in matrix

											RULE NR
	NO. RULES	EQ	MISCTR	N	N	N	N	N	N	Y	
	NO. COLS. CD 1	EQ	MISCTR1	N	N	N	N	N	Y		
	ENTRY OPR	EQ	BLANK	N	N	N	N	Y			
	MSW1	EQ		OFF	OFF	OFF	ON				
	MSW2	EQ		OFF	OFF	ON	OFF				
	MSW3	EQ		OFF	ON	OFF	OFF				
SET	MATRIX OPR	EQ		STUB OPR	IP ENTRY OPR	IP ENTRY OPR	IP ENTRY OPR				
SET	MATRIX OPND	EQ		IP ENTRY OPND	BLANK	IP ENTRY OPND	STUB NAME A				
	MISCTR	+	+1	X	X	X	Y	X			
	MISCTR1	+	+1	Y	X	X	X	X			
	MISC1	+	INCR CD	X	Y	X	X	X			
↓	MISC2	+	INCR CD	X	Y	X	Y	X			
MOVE	STUB	TO	MATRIX							X	
GET	INPUT								X	X	
GO TO				msym12	msym12	msym12	msym12	msym12	msym16	msym18	

DATA RULES

PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS | TABLE: MSYM18 | ACTIONS

X = LOCENTRY + 11
Y = LOCENTY - 515

											RULE NR.
	IP OPER	EQ	'TABLE'	Y	N	N	N	N	N		
	↓	↓	'TITLE'		Y	N	N	N	N		
			'ENDOP'			Y	N	N	N		
			'ORDER'				Y	N	N		
	IP ACT OPR	EQ	BLANKS					Y	N		
MOVE	MSG3	TO	MESSAGE		X		X				
DO	PRINT				X		X				
GO TO					SKIP TBL		SKIP TBL	MSYM10			
MOVE	ON	TO	HADACT						X		
SET	TAG NO	+	+1						X		
MOVE	(TR)	TO	ACT TAG TR						X		
MOVE	TAG NO	TO	ACT TAG TR						X		
MOVE	ZERO	TO	MISC TR	X		X			X		
MOVE	ZERO	TO	MJ	X		X			Y		
MOVE	LOCENTRY	TO	MI	X		X			X		
	X	TO	MIJ	X		X			Y		
	LOCENTRY	TO	MISC2	X		X			X		
	Y	TO	MISC1	X		X			X		
MOVE	OFF	TO	ASK SW	X		X			X		
MOVE	(*)	TO	SAVERULE-H	X		X			X		
GO TO	MSYM21			X		X			X		

Line Reference Target LRT-BE4-V2

DATA RULES

PROCESS AREA CONTACT PERSONNEL	PREPARED BY	DATE / REVISION	PROJECT NR	DESCRIPTION	PROCESS AREA / NR	PAGE / TO PAGE

CONDITIONS | TABLE: msym21 | ACTIONS: Scan for adjacent redundancies and replace with X

( ) = contents of index

X = MI + 11

													RULE NR
NO RULES	EQ	MJ	N	N	N	N	N	N	Y	Y	Y	Y	
(MIJ)	EQ	BLANK	N	N	N	N	N	Y					
(MIJ)	EQ	SAVRULE	Y	Y	N	N	N						
(misc1)	EQ	(*)	Y	N	Y	Y	Y						
(MIJ)OPND	EQ	SAV RULE OPND			Y	Y	N						
(MIJ)OPR	IN	OPPOSITE TBL			Y	Y	even if no						
OPP. OPR	EQ	SAVRULE OPR			Y	N							
ASK SW	EQ	ON							X	X	N	N	
MI	EQ	LOCENTRY									Y	N	
ROWOTR	EQ	MISCTR							N	Y			
MOVE	(*)	To	(misc2)	X									
MOVE	ON	To	ASKSW	X									
MOVE	'A'	To	(misc2)			X							
MOVE	(MIJ)	To	SAV RULE		X	X	X	X	X				
SET	MIJ	+	+15	X	X	X	X	X	X				
SET	misc1	+	+15	X	X	X	X	X	X				
SET	misc2	+	+15	X	X	X	X	X	X				
SET	MJ	+	+1	X	X	X	X	X	X				
SET	misc1	EQ	MI							X			
	MI	+	+515							X			
	misc2	EQ	MI							X			
	MIJ	EQ	X							X			
	MJ	EQ	ZERO							X			
	ASK SW	EQ	OFF							X			
MOVE	(*)	To	SAVRULE-4							X			
MOVE	msg1	To	MESSAGE								X		
DO	PRINT										X		
GO TO			msym21	msym21	msym21	msym21	msym21	msym21	msym21				

see next page

DATA RULES

PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR. | PAGE / TO PAGE

CONDITIONS | TABLE: MSym21 (Cont'd) | ACTIONS | RULE NR.

SET	MIJ	EQ	LOC ENTRY					
↓	MJ	EQ	ZERO	Y	X	X		
	MI	EQ	ZERO	X	X	X		
Go To	MSym27			X	X	X		
				X	X	X		

DATA RULES

PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE | REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA | NR. | PAGE | TO PAGE

CONDITIONS | TABLE: msym27 | ACTIONS: Set tag bits in matrix - assign tags & place in XREFLIST

X = MJ * 11 + LOC XREF
Y = MJ * 15 + LOC ENTRY
Z = MJ * 11 + LOC XREF LOC
W = LOC XREF + 10

											RULE NR.
	MJ	EQ	NO. RULES	N	N	N	N	N	N	Y	
	MI	GR	ROW CTR	N	N	N	N	N	Y		
	MJ	EQ	ELSE RULE	N	N	N	N	Y			
	(MIJ)	EQ	'*'	N	N	N	Y				
	(MIJ)	EQ	BLANK	N	N	Y					
	X	EQ	BLANK	Y	N						
MOVE	'0'	TO	(Z)						X		
SET	MIJ	+	+12	X							
MOVE	'A'	TO	(MIJ)	X							
SET	TAG NO	+	+1	X							
MOVE	TAG NO	TO	(X)	X							
SET	MJ	+	+1	X	X			X	X		
MOVE	ZERØ	TO	MI	X	X			X	X		
SET	MIJ	EQ	Y	X	X			X	X		
SET	MIJ	+	+515			X	X				
SET	MI	+	+1			X	X				
SET	MI	EQ	ZERØ							X	
SET	MJ	EQ	ZERØ							X	
SET	MISC1	EQ	W							X	
SET	MIJ	EQ	LOC ENTRY							X	
GOTO				msym27	msym27	msym27	msym27	msym27	msym27	msym44	

Line Reference Target LRT-BE4-V2

DATA RULES

PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE | REVISION | PROJECT NR. | DESCRIPTION | PROCESS AREA | NR | PAGE | TO PAGE

CONDITIONS | TABLE: MSYM41 | ACTIONS: Scan Matrix and Set-up Fixture Transfer Points

w = MJ * 15 + LOC ENTRY
x = MISC1 + 11
y = MJ + 1
z = MSJ * 15 + LOC ENTRY

													RULE NR
	MJ	EQ	NO. RULES	N	N	N	N	N	N	N	Y		
	MJ	EQ	ELSE RULE	N	N	N	N	N	Y				
	MJ	EQ	BLANK RULE	N	N	N	N	Y					
	(MISC1)	EQ	BLANK	N	N	N	Y						
	(MISC1)	EQ	'*'	N	N	Y							
	(MISC1)	EQ	'I'	N	Y								
	MI	EQ	ROW CTR	N	N	N	N	N	N	Y			
MOVE	x	To	MISC2	X									
MOVE	y	To	MSJ	X									
MOVE	ZERO	To	MSI	X									
MOVE	z	To	MSIJ	X									
SET	MIJ	+	+515		X	X	X						
MOVE	ZERO	To	MI		X	X	X	X	X	X			
SET	MISC1	+	+11		X	X	X	X	X	X			
SET	MJ	+	+1		X	X	X	X	X	X			
MOVE	w	To	MIJ		X	X	X	X	X	X			
GO TO				MSYM44	MSYM41	MSYM41	MSYM41	MSYM41	MSYM41	MSYM41	MSYM51		

DATA RULES

PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS | TABLE: msym44 | ACTIONS

x = mIJ + 13
y = msJ * 15 + LocEntry

										RULE NR.
	msJ	EQ	No. Rules	N	N	N	N	N	Y	
	msJ	EQ	ELSE Rule	N	N	N	N	Y		
	msJ	EQ	BLANK Rule	N	N	N	Y			
	msIJ	EQ	(*)	N	Y	Y				
	mI	GR	msI		Y	N				
SET	FAILTR EXT	EQ		msJ					ELSE Rule	
SET	msJ	+	+1			X	X	X		
SET	MISCR	+	+11			X	X	X		
MOVE	ZERO	To	msI			X	X	X		
SET	msIJ				+ +515	EQ y	EQ y	EQ y		
SET	mIJ	+	+515	X					X	
Go To				msym41	msym44	msym44	msym44	msym44	msym41	

DATA RULES

PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE | REVISION | PROJECT NR. | DESCRIPTION | PROCESS AREA | NR | PAGE | TO PAGE

CONDITIONS | TABLE: msym51

ACTIONS: setup & write out table info. (tag, NOP-y DO etc)

W =
X = TAB XXX — TR NEXT TABLE; TRANS *
Y =
Z = — TRIN TABLE NAME; move #GGGG# 7 TPTPCWNDSWJ

																	RULE NR.
	TBL NO	EQ	BLANK	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	
	TBL NAME	EQ	BLANK	N	N	N	N	Y	Y	Y	Y	N	N	N	N	Y	
	NEXT TABLE	EQ	'DO'	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y		
	HAD ACT	EQ	ON	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y		
MOVE	X	TO	OUTPUT	X	X	X	X										
PUT	OUTPUT			X	X	X	X										
MOVE	TBL NO.	TO	TBL. NAME					X	X	X	X						
MOVE	TBL NAME	TO	OP TAG	X	X	X	X	X	X	X	X	X	X	X	X		
MOVE	'NOP'	TO	OP OPER			X	X			X	X			X	X		
PUT	OUTPUT					X	X			X	X			X	X		
MOVE	W	TO	ACT TAG TR	X				X				X					
SET	TAG NO	+	+1		X		X		X		X		X		X		
MOVE	TAG NO	TO	CUR ACT COD		X		X		X		X		X		X		
MOVE	Y	TO	ACT TAG TR			X				X				X			
MOVE	Z	TO	OP OPER	X	X	X	X	X	X	X	X	X	X	X	X		
PUT	OUTPUT			X	X	X	X	X	X	X	X	X	X	X	X		
SET	MSG TIME	TO	ON	X	X	X	X	X	X	X	X	X	X	X	X		
SET	TAG NO	+	+1									X	Y	X	Y	X	
MOVE	TAG NO	TO	SETUP TRUE										Y	Y	X		
MOVE	ZERO	TO	SETUP FALSE										X	X	X		
MOVE	ZERO	TO	MISC TR										X	X	X		
SET	MISC1	EQ	LOG MATRIX										X	X	X		
SET	MISC 2	EQ	BIT SW TAG										X	X	X		
GoTo	msym58			↓	↓	↓	↓	↓	↓	↓	↓	↓	X	X	X		
GoTo	SKIPTBL															X	

DATA RULES

PROCESS AREA CONTACT PERSONNEL	PREPARED BY	DATE / REVISION		PROJECT NR.	DESCRIPTION	PROCESS AREA / NR.	PAGE / TO PAGE

CONDITIONS | TABLE: m54m53 | ACTIONS

make test statements for limited rows (condition)

									RULE NR.
	CNTLTD	EQ ZERO	N	N	N	Y			
	CNT LTD	EQ MISCTR	N	N	Y				
	FLAG	EQ LTD	Y	N					
MOVE	SETUPTRUE	TO SETUP TAG	X						
SET	TAGNO	+ +1	X						
MOVE	TAG NO	TO SETUPTRUE	X						
MOVE	STUB, MISC1	TO SETUPNAM1	X						
DO	TESTSTEMNT		X						
MOVE	(SETON)	TO OP OPER	X						
MOVE	(misc2)	TO OP OPND	X						
MOVE	(1)	TO OP OPND	X						
PUT	OUTPUT		X						
SET	MISCTR	+ +1	X						
SET	MISC1	+ +515	X	X					
SET	MISC2	+ +515	X	X					
SET	FLAG	+ +515	X	X					
SET	MSITIME	TO OFF				X			
SET	MII	EQ LOC ENTRY			X	X			
SET	MI	EQ Loc MATRIX			X	X			
↓	MJ	EQ ZERO			X	X			
	MISCTR	EQ ZERO			X	X			
	MISC1	EQ LOC XREF CD			X	X			
Go To			m54m53	m54m53	m54m54	m54m54			

IBM 7080

DECISION TABLE LOGIC SHEET

PROGRAM IDENTIFICATION (75–80)

PAGE	LINE	CC	TYPE	NO. RULES	ELSE	TABLE NO.	IDENTIFICATION	TABLE NAME	NEXT TABLE NAME	ERROR TABLE NAME
01 02	03 05	06	07	08 09	10 11	12 15	16 20	21 30	31 40	41 50
	0000		M	04		0005	TABLE	MTRT	MSYNER	

PAGE	LINE	CC	IDENTIFICATION	ORDER OF CONDITION ROWS
01 02	03 05	06 07 11	12 15 16 20	21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
	0001		COND	ORDER

PAGE	LINE	CC	IDENTIFICATION	ORDER OF RULES
01 02	03 05	06 07 11	12 15 16 20	21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
	0002		RULE	ORDER

PAGE	LINE	CC	IDENTIFICATION	COMMENT
01 02	03 05	06 07 15	16 20	21 74
	0003		TITLE	INIT. FOR CONDITION PROCESSING

PROGRAM _____ DATE _____

PREPARED BY _____ PAGE _____

NEXT PAGE _____

START NEXT CARD: Dup. Col's. 1–5, Punch B in Col. 6, Skip Col's. 7–33, Dup. Col's. 75–80.

PAGE	LINE	CC	ACTION OPERATOR	NAME 1		NAME 2	RULE 1		RULE 2		RULE 3		RULE 4		RULE 5		RULE 6	
							01	02 03 04 05 06	07	08 09 10 11 12	13	14 15 16 17 18	19	20 21 22 23 24	25	26 27 28 29 30	31	32 33 34 35 36
							OP	NAME 2	OP	NAME 2	OP	NAME 2	OP	NAME 2	OP	NAME 2	OP	NAME 2
01 02	03 05	06 07 11		12 21	22 23	24 33	34 35	36 37 38 39 40 41 42 43 44 45	46 47	48 49 50 51 52 53 54 55 56 57	58 59	60 61 62 63 64 65 66 67 68 69	34 35	36 37 38 39 40 41 42 43 44 45	46 47	48 49 50 51 52 53 54 55 56 57	58 59	60 61 62 63 64 65 66 67 68 69
	01	A		MIRELSE	IF	BLANK	N		Y		Y		Y					
	02	A		MIRNEXTEL	EQ	2ND	Y				N		N					
	03	A		MIRPRETAL	IS	BLANK			Y				N					
	04	A																
	05	A	MOVE	BLANK	TO	MATRIX	X		X		X		X					
	06	A				XREFLIST												
	07	A				MACTCDTAG												
	08	A				MSETASEQ												
	09	A		MRITSWINET	TO	MRITSWTAG												
	10	A		MAREAL	TO	MDEDLROWS												
	11	A		DEF	TO	MRESDESW												
	12	A		ZERO	TO	MERRCTR												
	13	A		INPUT	TO	MSAVETHDR	X		X		X		X					
	14	A	SET	TAGNO	+	41	X		X		X							
	15	A	SET	MSVELSE	-	41	X											
	16	A	SET	MTSW4	EQ	MEXAL	X											
	17	A	MOVEV	MJGEN3	TO	MTSW4	X											
	18	A	MOVE	MJGEN1	TO	MSGTEXT			X		X							
	19	A	DO	PRINT					X		X							
	20	A	SET	MELSETAG	EQ			TAGNO		TAGNO		TAGNO		MSVERRTBL				
	21	A	SET	MERRTBLSW	EQ			ON		ON		ON		OFF				
	22	A	SET	MSVELSE	EQ	(3,2)			X		X		X					
	23	A	SET	MRITAG	EQ			TAGNO		TAGNO				MSVERRTBL				
	24	A																
	25	A	DO	GET			X		X		X		X					

REMARKS:

IBM 7080

DECISION TABLE LOGIC SHEET

PROGRAM IDENTIFICATION (75–80)

PAGE	LINE	CC	TYPE	NO. RULES	ELSE	TABLE NO.	IDENTIFICATION	TABLE NAME	NEXT TABLE NAME	ERROR TABLE NAME
	0000		M			0006	TABLE	MSYM28	MSYM98	MSYM90

PAGE	LINE	CC	IDENTIFICATION	ORDER OF CONDITION ROWS
	0001		CONDORDER	

PAGE	LINE	CC	IDENTIFICATION	ORDER OF RULES
	0002		RULEORDER	

PAGE	LINE	CC	IDENTIFICATION	COMMENT
	0003		TITLE	PROCESS TABLE HEADER CARDS

PROGRAM _____ DATE _____

PREPARED BY _____ PAGE _____

NEXT PAGE _____

START NEXT CARD: Dup. Col's. 1–5, Punch B in Col. 6, Skip Col's. 7–33, Dup. Col's. 75–80.

PAGE	LINE	CC	ACTION OPERATOR	NAME 1		NAME 2	RULE 1						RULE 2						RULE 3						RULE 4						RULE 5						RULE 6					
							01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
							OP	NAME 2					OP	NAME 2					OP	NAME 2					OP	NAME 2					OP	NAME 2					OP	NAME 2				
	01	A		INDEX		EQ		'TITLE'						'ORDER'						'ORDER'						'ORDER'																
	02	A		HEADING		EQ								'COND'						'RULE'						'RULE'																
	03	A		MAXLINE		ES	ZERO																								N						N				
	04	A		MAXCHAR		IS	BLANK																								Y						N				
	05	A		MSWRULES		VS	'27'												ER						LE																
	06	A																																								
	07	A	MOVE	00		TO							MSWDSW						MRULESW						MRULESW																
	08	A	SET	MSWA		EQ	ZERO						X						X						X																
	09	A	SET	MSW2		EQ	MAXCHAR						X						X						X																
	10	A	SET	MSCAS		EQ														'27'						MSWNORULES																
	11	A	DO										MSYM64						MSYM7						MSYM7																
	12	A	MOVE	CTITLE		TO	INDREC						X						X						X																
	13	A	MOVE	INREC		TO	OUTPUT		X				X						X						X																
	14	A	DO	PUT					X				X						X						X																
	15	A	DO	GET					X				X						X						X																
	16	A	MOVE	00		TO	HEADCNT																								X										
	17	A	MOVE	ZERO		TO	MAXLTD																								Y										
	18	A	MOVE	ZERO		TO	MAXTEXT																								X										
	19	A	GOTO						MSYM68				MSYM66						MSYM32						MSYM32						MSYM10						MSYM91				
	20	A																																								
	21	A																																								
	22	A																																								
	23	A																																								
	24	A																																								
	25	A																																								

REMARKS:

IBM 7080

DECISION TABLE LOGIC SHEET

PROGRAM IDENTIFICATION (75–80): _____

PAGE	LINE	CC	TYPE	NO. RULES	ELSE	TABLE NO.	IDENTIFICATION	TABLE NAME	NEXT TABLE NAME	ERROR TABLE NAME
	0000		L	04		0007	TABLE	MSYMSP	DO	

PAGE	LINE	CC	IDENTIFICATION	ORDER OF CONDITION ROWS
	0001		CONDORDER	

PAGE	LINE	CC	IDENTIFICATION	ORDER OF RULES
	0002		RULEORDER	

PAGE	LINE	CC	IDENTIFICATION	COMMENT
	0003		TITLE	MOVE ADD ORDER INTO MSYSERRS

PROGRAM _____ DATE _____

PREPARED BY _____ PAGE _____

NEXT PAGE _____

START NEXT CARD: Dup. Col's. 1–5, Punch B in Col. 6, Skip Col's. 7–33, Dup. Col's. 75–80.

PAGE	LINE	CC	ACTION OPERATOR	NAME 1		NAME 2	RULE 1 OP	RULE 1 NAME 2	RULE 2 OP	RULE 2 NAME 2	RULE 3 OP	RULE 3 NAME 2	RULE 4 OP	RULE 4 NAME 2	RULE 5 OP	RULE 5 NAME 2	RULE 6 OP	RULE 6 NAME 2
	01	A						01 02 03 04 05 06		07 08 09 10 11 12		13 14 15 16 17 18		19 20 21 22 23 24		25 26 27 28 29 30		31 32 33 34 35 36
	02	A						34 35 36 37 38 39 40 41 42 43 44 45		46 47 48 49 50 51 52 53 54 55 56 57		58 59 60 61 62 63 64 65 66 67 68 69		34 35 36 37 38 39 40 41 42 43 44 45		46 47 48 49 50 51 52 53 54 55 56 57		58 59 60 61 62 63 64 65 66 67 68 69
	01	A		MSG7R		EQ C,16	Y	N N N										
	02	A		MSG2,I		EQ C,2		Y N N										
	03	A		MSG2,I		EQ C,3		N Y										
	04	A																
	05	A																
	06	A	MOVE	MSG2,0002		TO MSGTRI		X										
	07	A	MOVE	MEXP3		TO MEXP3,I		X										
	08	A	SET	MSG+7R		+ 16		X										
	09	A	SET	MSG+2		+ 12		X										
	10	A	COMP	ORDER				X										
	11	A	COMP	ORDER														
	12	A																
	13	A																
	14	A																
	15	A																
	16	A																
	17	A																
	18	A																
	19	A																
	20	A																
	21	A																
	22	A																
	23	A																
	24	A																
	25	A																

REMARKS: MEXP2 = MSGTRI * 515–515 + MEOCMATRIX (index of beg of rows for this condition)
MEXP3 = MSGTR * 5 + MEOCROWS (index in other row arch)
MSG2 is current rule in SWAT area.

IBM 7080

DECISION TABLE LOGIC SHEET

PROGRAM IDENTIFICATION (75–80)

PAGE	LINE	CC	TYPE	NO. RULES	ELSE	TABLE NO.	IDENTIFICATION	TABLE NAME	NEXT TABLE NAME	ERROR TABLE NAME
	0000		L			0008	TABLE	MESSTR	DD	

PAGE	LINE	CC	IDENTIFICATION	ORDER OF CONDITION ROWS
	0001		CONDORDER	

PAGE	LINE	CC	IDENTIFICATION	ORDER OF RULES
	0002		RULEORDER	

PAGE	LINE	CC	IDENTIFICATION	COMMENT
	0003		TITLE	MOVE RULE CODES INTO INDICATORS

PROGRAM _____ DATE _____

PREPARED BY _____ PAGE _____

NEXT PAGE _____

START NEXT CARD: Dup. Col's. 1–5, Punch B in Col. 6, Skip Col's. 7–33, Dup. Col's. 75–80.

PAGE	LINE	CC	ACTION OPERATOR	NAME 1		NAME 2	RULE 1						RULE 2						RULE 3						RULE 4						RULE 5						RULE 6					
							01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
							OP	NAME 2					OP	NAME 2					OP	NAME 2					OP	NAME 2					OP	NAME 2					OP	NAME 2				
	01	A																																								
	02	A																																								
	03	A																																								
	04	A																																								
	05	A																																								
	06	A	MOVE	MESSAGE		TO MESTR1			X																																		
	07	A	MOVE	MESX04		TO MESTR1			X																																		
	08	A	SET	MESSTR		+ 11			X																																		
	09	A	SET	MESSG		+ 12			X																																		
	10	A	ADD	MESSAGE					X																																		
	11	A	ADD	MESSAGE						X																																	
	12	A																																								
	13	A																																								
	14	A																																								
	15	A																																								
	16	A																																								
	17	A																																								
	18	A																																								
	19	A																																								
	20	A																																								
	21	A																																								
	22	A																																								
	23	A																																								
	24	A																																								
	25	A																																								

REMARKS: MEXP4 = MESSTR1 * 15-15
MEXP5 = MESSTR * 5 + MCODES
MES03 is No. Bullets on 27 MES02 is current addr. in INPUT area.

IBM 7080

DECISION TABLE LOGIC SHEET

PROGRAM IDENTIFICATION (75–80): TPLIST

PAGE	LINE	CC	TYPE	NO. RULES	ELSE	TABLE NO.	IDENTIFICATION	TABLE NAME	NEXT TABLE NAME	ERROR TABLE NAME
	0000		M	06		0003	TABLE		TABOOOH	NONE

PAGE	LINE	CC	IDENTIFICATION	ORDER OF CONDITION ROWS
	0001		CONDORDER	01 03 02

PAGE	LINE	CC	IDENTIFICATION	ORDER OF RULES
	0002		RULEORDER	

PAGE	LINE	CC	IDENTIFICATION	COMMENT
	0003		TITLE	INIT. PER TABLE 6TH CARD

PROGRAM _____ DATE _____

PREPARED BY _____ PAGE _____

NEXT PAGE _____

START NEXT CARD: Dup. Col's. 1–5, Punch B in Col. 6, Skip Col's. 7–33, Dup. Col's. 75–80.

PAGE	LINE	CC	ACTION OPERATOR	NAME 1	OP	NAME 2	RULE 1 OP	RULE 1 NAME 2	RULE 2 OP	RULE 2 NAME 2	RULE 3 OP	RULE 3 NAME 2	RULE 4 OP	RULE 4 NAME 2	RULE 5 OP	RULE 5 NAME 2	RULE 6 OP	RULE 6 NAME 2
	01	A		TALTYPE	EQ	(2)	Y		Y		Y		N		N		N	
	02	A		NORULES	IS		LE (28)		LE (28)		GR (28)							
	03	A		NORULES			IS ZERO		NT ZERO				IS ZERO		NT ZERO		NT ZERO	
	04	A		EXP1	GR	(00)									N		Y	
	05	A	SET	WORK	EQ	EXP1											X	
	06	A	SET	PAGENO	+	+1					X						X	
	07	A	DO	COMPCS													X	
	08	A	SET	NORULES	EQ	(6)	X						X					
	09	A	SET	ZRULESW	EQ			00		OFF		OFF		ON		OFF		OFF
	10	A	SET	ORDERCOLS	EQ			AREA3		AREA3		AREA3		AREA2		AREA2		AREA2
	11	A	DO					INITHDG3S		INITHDG3S		INITHDG3S		INITHDG3L		INITHDG3L		INITHDG3L
	12	A	GET	INPUT			X		X		X		X		X		X	
	13	A																
	14	A																
	15	A																
	16	A																
	17	A																
	18	A																
	19	A																
	20	A																
	21	A																
	22	A																
	23	A																
	24	A																
	25	A																

REMARKS: EXP1 = NORULES-6

IBM 7080

DECISION TABLE LOGIC SHEET

PROGRAM IDENTIFICATION

PAGE	LINE	CC	TYPE	NO. RULES	ELSE	TABLE NO.	IDENTIFICATION	TABLE NAME	NEXT TABLE NAME	ERROR TABLE NAME
	0000		L	06		0004	TABLE			NONE

PAGE	LINE	CC	IDENTIFICATION	ORDER OF CONDITION ROWS
	0001		CONDORDER	

PAGE	LINE	CC	IDENTIFICATION	ORDER OF RULES
	0002		RULEORDER	

PAGE	LINE	CC	IDENTIFICATION	COMMENT
	0003		TITLE	

PROGRAM _____ DATE _____

PREPARED BY _____ PAGE _____

NEXT PAGE _____

START NEXT CARD: Dup. Col's. 1–5, Punch B in Col. 6, Skip Col's. 7–33, Dup. Col's. 75–80.

PAGE	LINE	CC	ACTION OPERATOR	NAME 1	OP	NAME 2	RULE 1						RULE 2						RULE 3						RULE 4						RULE 5						RULE 6					
							01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32					
							OP	NAME 2					OP	NAME 2					OP	NAME 2					OP	NAME 2					OP	NAME 2					OP	NAME 2				
	01	A		PGLINE		ISZERO	Y	Y	Y	Y	N	N																															
	02	A		IMPER		EQ (TITLE)	N	N	Y	Y																																		
	03	A		IMPER		EQ (ORDER)	N	Y																																			
	04	A		PGTOP		ISBLANK					Y	N																															
	05	A		TITLCTR		EQ NOPAGES			Y	N																																	
	05	A																																							
	07	A	MOVE	MSG1		TO OUTPUT	X																																		
	08	A	PUT	OUTPUT			X																																		
	09	A	MOVE	MSG1		TO OUTPUT	X																																		
	10	A	PUT	OUTPUT			X																																		
	11	A	GOTO	STOPTAL			X																																		
	12	A	DO	ADDPRINTT				X																																	
	13	A	SET	XR		EQ LOCHDG2				X																																
	14	A	MOVE	INPUT		TO HDG2,XR				X																																
	15	A	SET	LOCHDG2		+ 132				X																																
	16	A	SET	TITLCTR		+ 1				X																																
	17	A	GET	INPUT				X	X	X																																
	18	A	GOTO	TABPRINT				X	X	X																																
	19	A																																							
	20	A																																							
	21	A																																							
	22	A																																							
	23	A																																							
	24	A																																							
	25	A																																							

REMARKS:

IBM 7080

DECISION TABLE LOGIC SHEET

PROGRAM IDENTIFICATION: TPLIS

PAGE	LINE	CC	TYPE	NO. RULES	ELSE	TABLE NO.	IDENTIFICATION	TABLE NAME	NEXT TABLE NAME	ERROR TABLE NAME
	0000		L	02	02		TABLE	COMPGS	DD	

PAGE	LINE	CC	IDENTIFICATION	ORDER OF CONDITION ROWS
	0001		CONDORDER	

PAGE	LINE	CC	IDENTIFICATION	ORDER OF RULES
	0002		RULEORDER	

PAGE	LINE	CC	IDENTIFICATION	COMMENT
	0003		TITLE	COMPUTE NO. PAGES REQUIRED FOR MIXED TABLE.

PROGRAM _____ DATE _____

PREPARED BY _____ PAGE _____

NEXT PAGE _____

START NEXT CARD: Dup. Col's. 1–5, Punch B in Col. 6, Skip Col's. 7–33, Dup. Col's. 75–80.

PAGE	LINE	CC	ACTION OPERATOR	NAME 1	OP	NAME 2	RULE 1		RULE 2		RULE 3		RULE 4		RULE 5		RULE 6	
							01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
							OP		NAME 2				OP		NAME 2				OP		NAME 2				OP		NAME 2				OP		NAME 2				OP		NAME 2	
	01	A		WORK	GR	000	Y																															
	02	A																																				
	03	A	SET	PAGENO	+	+1	X																															
	04	A	SET	WORK	−	+2	X																															
	05	A	GOTO	COMPGS			X																															
	06	A																																				
	07	A																																				
	08	A																																				
	09	A																																				
	10	A																																				
	11	A																																				
	12	A																																				
	13	A																																				
	14	A																																				
	15	A																																				
	16	A																																				
	17	A																																				
	18	A																																				
	19	A																																				
	20	A																																				
	21	A																																				
	22	A																																				
	23	A																																				
	24	A																																				
	25	A																																				

REMARKS:

DATA RULES

APPROVALS & DATES	PROCESS AREA CONTACT PERSONNEL	PREPARED BY	DATE / REVISION	PROJECT NR	DESCRIPTION	PROCESS AREA / NR	PAGE / TO PAGE

CONDITIONS

TABLE 10Φ - PROCESS CONTROL CARDS

ACTIONS

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME	(1) OP	(1) VALUE	(2) OP	(2) NAME	(3) OP	(3) NAME	(4) OP	(4) NAME	(5) ELSE	RULE NR
				INOPER	EQ			(DATE)		(MODEL)		(ASMBL)		(ASMBL)		
				INOPND	EQ							(ORIGINAL)		(TABLAR)		
			MOVE	INOPND		TYPEMSGH	X									
			MOVE	INOPND		PRINTMSGH	X									
			MOVE	(ORIGINAL)		INOPND							X			
			SET	CONTCDSW	OF								X			
			MOVE	INPUT		OUTPUT	X		X				X			
			DO	PUT			X		X				X			
			DO	GET			X		X				X			
			SET	ERRORIND								2 (IMPROPER CONTROL CARD)			1 (IMPROPER ASSIGN)	
			TYPE	PROGRAM TO PROGRAM USE.									X			
			DO													
			GO TO					TAB1104		TAB1204		TAB0900		TAB1401	TAB0900	

INDICATES TO GET ROUTINE TO BEGIN SEQUENCE CHECKING

(FOR CONVENIENCE IN CHARTING, ERROR SW IS SET ON - THESE ACTUALLY GO TO SEPERATE ROUTINES)

ALL INITIALIZATION, RESETING, BLANKING, ETC. (NOT SHOWN)

FORM 6004 REV AUG 1960

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS

TABLE 101 - ROUTINE SELECTION I

ACTIONS

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME	① OP	① NAME	② OP	② NAME	③ OP	③ NAME	④ OP	④ NAME	⑤ OP	⑤ NAME	⑥ OP	⑥ NAME	⑦	⑦ NAME	ELSE		RULE NR
				INOPER	EQ			(TABLE)		(TABLE)		(TABLE		(END)				(ENTRY)		(LEVEL)			
				AUTOCODER	IS			ON		OF		OF										INDICATES AUTOCODER WAS BEING BYPASSED	
				TAB 0,1	IS					ON		OF										IND. LAST TABLE WAS 0000, 0001	
				BYPASS	IS										ON		OF		OF			IND. A TABLE TYPE A OR R LAST FOUND	
			JET	BYPASS	OF		X		X		X		X										
			MOVE	NAMESAVE		OUTGND	X																
			MOVE	(STRING)		OUTOBER	X																
																						TO BUILD TRIN ENTRY TO CLOSE A 3 SUBNO	
			MOVE	(LASH)		OUTCHER			X													IF TAB 0,1 NEED A BLANK LASH FOR KNAME	
			MOVE	(UNKNOWN OR TYPE)		MESSAGE														X			
			MOVE	INPUT		OUTPUT									X		X		X	X			
			DO	PUT					X						X		X		X	X			
			DO	GET											X		X		X	X			
			DO	PRINT																X		PUTS ERROR MESSAGE ON TAPE	
			SET	AUTOCODER	OF		X																
			GO TO					TAB0102		TAB0102		TAB0102		TAB0301		TAB0101		TAB0101		TAB0101		TAB0107	

FORM 3004 REV AUG 1960

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA NR | PAGE / TO PAGE

CONDITIONS

TABLE 102 - FIND TABLE TYPE

ACTIONS

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME	OP	① NAME	OP	② NAME	OP	③ NAME	④	⑤	⑥	ELSE		RULE NR.
				TABLE TYPE	EQ			(R)		(A)		(A)	(A)	(D)	(D)		COL 7 OF TABLE HEADER	
				TAB NO					IS	BLANK	NT	BLANK	NT BLANK				CHECK TO SEE IF 2 TABS NEEDED FOR A III TABLE	
				TAB NO							LE	(0003)	GE (0003)	LE 0003	GR 0003			
			MOVE	INPUT		OUTPUT		X		X		X	X	X	X	X		
			MOVE	(TITLE)		OUTOPER		X		X		X	X	X	X	X		
			DO	PUT				X		X		X	X	X	X	X		
			SET	BYPASS	ON			X		X		X	X					
			SET	AUTOCODER	ON					X		X	X					
			SET	NAME SAVE	EQ							TABNAME	TABNAME	TABNO			SAVE VALID NAME	
			MOVE	(TRANS *)		OUTOPER							X					
			MOVE	TABNO		OUTTAB							X					
			DO	PUT									X				BUILD TRANS ENTRY IF BOTH VALID	
			MOVE	TABNAME		OUTTAB				X		X	X					
			MOVE	(NOP+++04)		OUTOPER				X		X	X					
			DO	PUT						X		X	X					
			DO	GET				X		X		X	X					
			MOVE	TABNO		NOCOMP								X			COMPUTE LOCATION OF VALIDITY TEST TZB INSTR.	
			SET	NOCOMP	*	+5								X			= (TABNO × 5) + TZB TAB0	
			SET	ADDR TAB0	+	NOCMP								X			TZB INSTRUCTION THEN MOVED TO TAB0103	
			MOVE	I, ADDR TAB0		TEST VALID								X				
			DO	GET										X				
			SET	EXFILE	ON									X				
			GO TO					TAB0101		TAB0101		TAB0101	TAB0101	TAB 0103	TAB0105	MIII-I		

FORM 10004 REV AUG. 1960

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS

TABLE 103 - OPERATOR VALIDITY CHECK AND ROUTING SELECTION II

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	① NAME	OP	① NAME	OP	② NAME	OP	② NAME	OP	③ NAME	OP	③ NAME	RULE NR
				INOPER	IS	OPTABLE	Y					N			
				VALID CHAR.			Y								

TABLE LOOK UP FOR OPER.- FUNCTION HAS BIT CODE IDENTIFYING THOSE TAB NOS. IN WHICH THIS OP. IS VALID. TESTED BY TEB INSTR MOVED IN PLACE BY TABOIOR ON BASIS OF CURRENT TAB NO. FUNCTION ALSO HAS ADDR. of ROUTINE TO GO TO FOR EACH OP TYPE

ACTION	NAME	OP	NAME	OP	NAME	OP	NAME
SET	MESSAGE			EQ (INVALID OP FOR TABLE)		EQ (UNKNOWN OP TYPE)	
DO	PRINT			X		X	
GOTO			I₂ OPTABLE FN	I₃ OPTABLE FN		TABOIOR (BYPASS)	

ROUTINES AS FOLLOWS:

FILE	200	✓
RNAME	204	✓
GNAME	207	✓
FLD	203	✓
PRE	204	✓
RLIST	205	✓
VLIST	206	✓
SEXP	208	✓
MEXP	208	
PCON	204	✓
POINT	209	
TABLE	102	✓
END	301	✓
ALL OTHERS	210	✓ Bypass as is

FORM 8004 REV AUG 1960

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS

TABLE 105 - ERROR- DATA TABLE HAS TAB NO. OVER 0003

ACTIONS

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME	① OP	NAME	② OP	NAME	③ OP	NAME	④ OP	NAME	⑤ OP	NAME	⑥ OP	NAME	⑦ OP	NAME	RULE NR
				NEXT BUFFER	IS			1		1		1		2		2		2			A BIT OVER 10'S POS OF EIA INSTR IN GET
				B1 OPEN	NS	FILE	NE		EQ		NE										ROUTINE TESTED TO SEE WHICH BUFFER IS NEXT
				B2 OPEN	NS	ORDER	EQ		NE		NE										
				B3 OPEN									NE		EQ		NE				
				B4 OPEN									EQ		NE		NE				
			MOVE	MIL-SEC 1		MESSAGE 1	X		X		X		X		X		X				(IMPROPER TAB. NO FOR DATA TABLE -)
			SET	MESSAGE 1	EQ			(ASSUME FUNCTION)		(ASSUME TAB 0000)		(ASSUME TAB 0003)		(ASSUME FUNCTION)		(ASSUME TAB 0004)		(ASSUME TAB 0003)			
			SET	TEST VALID	EQ					T=84		T&B 3				T&B 4		T&B 3			MOVE T&B INTO TAB 103 FOR VALIDITY TEST
			SET	BIG ERROR	ON						X				X						MAY BE CRITICAL ERROR
			SET	EXP FILE	ON			X													
			GO TO					WAIT 1		TAB 0103		TAB 0103		WAIT 1		TAB 0103		TAB 0103			

FORM 8004 REV AUG 1980

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS

TABLE 200 - FILE ROUTINE

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME (1)	OP	NAME (2)
				EXP FILE	ON		Y	
			MOVE	(LESS RVBNES) SIAM CALLED LAST FILE		MESSALE		X
			DO	PRINT				X
			SET	EXP FILE	OF		Y	
			SET	1ST RNAME	ON		Y	X
			MOVE	(TITLE)		INOPER	X	X
			MOVE	INNV		FILECOUNT	X	X
			MOVE	(CLASN)		OUTOPER	X	X
			DO	PUT			Y	X
			GO TO	TAB0110			X	X

TABLE 202
GNAME
ROUTINE

ACTION	NAME	OP	NAME
MOVE	(NAME8)		INOPER
GO TO	TAB0110		

TABLE 203.
FLD ROUTINE

ACTION	NAME	OP	NAME
MOVE	(FLD)		INOPER
GO TO	TAB0110		

TABLE 204
PRE &
PCON ROUTINE

ACTION	NAME	OP	NAME
MOVE	(CON)		INOPER
GO TO	TAB0110		

TABLE 210
BYPASS ROUTINE

ACTIONS

ACTION	NAME	OP	NAME	RULE NR
MOVE	INPUT		OUTPUT	
DO	PUT			
DO	GET			
GO TO	TAB0103			

FORM 9004 REV AUG 1960

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS

ACTIONS

TABLE 201 - RNAME ROUTINE

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY																		RULE NR
			1ST RNAME	ON		Y		Y		Y		Y		N		N		N		
			FILE COUNT	VS		EQ	1	HI	1	EQ	1	HI	1	EQ	Ø	EQ	1	HI	1	
			INTAG	IS	BLANK	N		N		Y		Y								
			SET	FILE COUNT	EQ	1														
			SET	MESSAGE	EQ															
			DO	PRINT										(FIRST RNAME AND TAGNO)		(WHILE RNAMES FROM FILE AGREE)				
			SET	TAGNO	+	1								X		X				
			MOVE	TAGHOLD		OUTOPNO								X						
			MOVE	(RASN)		OUTOPER										X		Y		
			DO	PUT												X		Y		
			MOVE	SAVEDNU		INNU										X		X		
			SET	TAGHOLD	EQ			INTAG		INTAG		BLANKS		TAGNO						
			SET	1STRNAME	OF		X		✓		X		X							
			MOVE	INNU		SAVEDNU	X		X		X		X							
			SET	FILECOUNT	-	1	X		✓		X		X		X		X		X	
			SET	ERRFILE	ON		X				X						X			
			MOVE	(NAME)		INOPER	X		X		X		X		X		X		X	
			GO TO	TAB210			X		X		X		X		X		X		X	

FORM 8004 REV AUG 1970

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR. | DESCRIPTION | PROCESS AREA / NR. | PAGE / TO PAGE

CONDITIONS

TABLE 205 - RLIST ROUTINE

ACTIONS

TABLE 206 - VLIST ROUTINE

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	IN	NAME	OP	NAME (1)	OP	NAME (2)	ITS	ACTION	NAME	OF	NAME		RULE NR
				INBUFCL 39-58	15	BLANK	Y		N		CHECK FOR FUNCTION PLACEMENT TAG.	MOVE	INTAG		OUTTAG		
												MOVE	(TABLE)		OUTOPER		
												MOVE	INNO		OUTOPND		
												DO	PUT				
												MOVE	IN COL 43-74		OUT COL 43-74		
			MOVE	INPUT		OUTPUT	X		X			DO	PUT				
			BLANK	OUTCOL 39-58					X		BLANK FUNCTION TAG ON OUTPUT	MOVE	INTAG		TALTAG		
			MOVE	(TABLE)		OUTOPER	X		X			SET	TBLCODE	EQ	ZERO		
			MOVE	INCOL 39-58		ASSOCTAG			X		IF FUN TAG, PUT IN PLACE TO ENTER IN ASSOCTAG TABLE	SET	TBLSEQNO	EQ	ZEROS		
			MOVE	INTAG		TBLTAG	X		X			DO	TAB 0306				
			MOVE	ZERO		TBLCODE	X		X		CODE IS Ø	DO	GET				
			SET	SEQNO	+	1			X		IF FUN TAG - UPDATE SEQ NO	GO TO	TAB 0103				
			SET	TBLSEQNO	EQ			ZERO		SEQNO	AND PLACE IN TABLE BUILD AREA						
			DO	TAB 0346			X		X		TO ENTER TAG CODE, SEQ & ASSOC TAG IN ERR TABLE						
			DO	PUT			X		X								
			DO	GET			X		X								
			GO TO	TAB 0103			X		X								

FORM RDGA REV AUG 1968

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR. | PAGE / TO PAGE

CONDITIONS

TABLE 209 - POINT ROUTING

ACTIONS

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME (1)	dP	NAME (1)	OP	NAME (2)	RULE NR.
				INTAGLS	EQ	BLANKS	Y		N		
			MOVE	(POINT TAG OVER 5 CHAR)		MESSAGE			X		
			DO	PRINT					X		
			MOVE	INTAG		TBLTAG	X		X		
			MOVE	(-)		TBLCODE	X		X		
			MOVE	INCOL3SR		TBLSEQNO-1	X		X		
			MOVE	(0)		TBLSEQNO	X		X		
			DO	TAB0306			X		X		
			MOVE	(CON a+6)		INOPER	X		X		
			MOVE	INOUT		OUTPUT	X		X		
			DO	PUT			X		X		
			DO	GET			X		X		
			GO TO	TAB0103			X		X		

FORM 3006 REV AUG 1980

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE | REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA | NR | PAGE | TO PAGE

CONDITIONS

TABLE 207 SEKP ROUTINE

RULE NUMBER CURRENT	PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME	① OP	① NAME	② OP	② NAME	③ OP	③ NAME	OP	NAME
				INEXTYP	EQ	(JOIN)	N		Y		Y		Y	
				(%)	EQ	INCOL27			Y		N		N	
				(%)	EQ	INCOL28			N		Y		N	
			MOVE	INCOL 22		LOCATOR								
			DO	NAMELOC										
			MOVE	INTAG		TBLTAG.	x		x		x		x	
			MOVE	INTAG		OUTTAG	x		x		x		x	
			SET	TBLCODE	EQ	(1)	x		x		x		x	
			SET	TBLSEQNO	EQ	ZEROS	x		x		x		x	
			MOVE	INNU		OUTNU	x		x		x		x	
			MOVE	(RCD)		OUTOPER	x		x		x		x	
			MOVE	(A)		OUTOPND	x		x		x		x	
			DO	PUT			x		x		x		x	
			MOVE	(NOP @4)		OUTOPER	x		x		x		x	
			DO	PUT			x		x		x		x	
			MOVE	(GATHR)		WORKOPER	x		x		x		x	
			MOVE	(ASSUME JOIN)		MESSAGE	x							
			DO	PRINT			x							
			SET	COL i	EQ			INCOL 21		INCOL28		INCOL29		INCOL 28
			SET	LITERAL	EQ	OFF	x		x		x		x	
			SET	LASTBLANK	EQ	OFF	x		x		x		x	
			SET	QUIT	EQ	OFF	x		x		x		x	
			SET	INCOL 7576	EQ	BLANK	x		x		x		x	
			MOVE	(ASSUME BEGIN COL28)		MESSAGE							x	
			DO	PRINT									x	
			GO TO	SEKPSCAN			x		x		x		x	

ACTIONS

TABLE NAMELOC

ACTION	NAME	OP	NAME	① OP	① NAME	② OP	② NAME	③ OP	③ NAME		RULE NR
	LOCATOR	IS			ZERO		POS		NEG		
SET	LOCATOR	-	5			X					
MOVE	(NAME)		OUTOPER	X				X			
MOVE	LOCATOR		OUTCOL22	X				X			
DO	PUT			X				X			
GO TO					( )		NAMELOC		( )		

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS

TABLE 0211

SEXP OPERAND ROUTINE

OPND = TAG left out

ACTIONS

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	① NAME	OP	② NAME	OP	③ NAME	④	⑤	⑥	⑦	⑧	RULE NR
				PLUS	IS	ON	Y		N	N	N	Y	N	N	N	} EITHER INDICATES THIS OPND IS NUMERIC
				OPHOLD	IS	NUM			Y	N	N		Y	N	N	} IF NEEDS LITERAL BRACKETS
				OPHOLD	IS	EXPTABLE				Y	N			Y	N	IF NOT NUMERIC - LOOK TO SEE IF EXPRESS.
				QUIT	IS	OFF	Y		Y	Y	Y	N	N	N	N	INDICATES ✦ WAS LAST CHAR SCANNED
			SET	NEXTWORDREAD	EQ	NEXTCHAR	X		X	X	X	X	X	X	X	BUILD R2 ADDR OF WHERE LAST CHAR. OF
			SET	NEXTWORDREAD	+	LGTHOPND	X		X	X	X	X	X	X	X	THIS OPERAND WILL BE PLACED IN WORK AREA
			SET	NEXTWORDREAD	+	2	X		X	X	X	X	X			IF NUM. - ADD 2 FOR # #,2
			SET	PLUS		OFF	X				X					-
			DO	LONGCHECK			X		X	X	X	X	X	X	X	SEE IF COMPUTED ADDR HIGHER THAN Col 75-WORK.
			MOVE	(#)		I,NEXTCHAR	X		X		X	X				L.H. LITERAL SIGN
			SET	NEXTCHAR	+	LGTHOPND	X		X	X	X	X	X	X		
			SET	NEXTCHAR	+	1	X		X		X	X				
			MOVE	OPHOLD		I,NEXTCHAR	X		X	X	X	X	X	X		PLACE OPERAND IN WORK
			MOVE	OPHOLD		OUTOPND			X				X			IF THIS OPERAND IS EXPRESSION -
			MOVE	(TPLNK)		OUTOPER			X				X			BUILD LINKAGE TO EXPRESS ROUTINE
			DO	PUT					X				X			
			SET	NEXTCHAR	+	1	X				X	X				
			MOVE	(#)		I,NEXTCHAR	X				X	X				R.H. LITERAL SIGN
			SET	NEXTCHAR	+	1	X		X	X	X	X	X	X		
			MOVE	(4)		I,NEXTCHAR	X		X	X	X	X	X	X		LOSENGE TO FINISH THIS OPND
			SET	NEXTCHAR	+	1	X		X	X	X	X	X	X		
			MOVE	(-0001)		LGTHOPND	X		X	X	X	X	X	X		OP. LGTH CTR BEGINS AT -1
			GO TO	FINISHSEXP							X	X	X	X		✦ WAS LAST CHAR. SCANNED - FINISH EXP.
			SET	COL i	+	1	X		X	X						OTHERWISE BLANK OR , LAST, STEP TO SCAN NEXT
			GOTO	SEXPSCAN			X		X	X						CONTINUE SCANNING

FORM 8002 REV AUG 1960

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS

TABLE SEKPSCAN 0212

TABLE - FINISH SEKP 0213

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	⑬	⑭	⑮	ELSE
				I,COLi	EQ		(	)	(	)	(−)	(−)	(+)	(+)	LB	(#)	(♦)	(♦)	(♦)	(♦)	(♦)	)
				LITERAL	IS		ON	OF	OF	ON	OF	ON	OF	ON	OF	ON	OF	OF	ON	OF	OF	
				LASTBLANK	IS			ON	OF												OF	
				BEGINTAG	IS				OF								ON	OF			ON	
					IS																	
			SET	LASTBLANK					ON	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	OF	ON	OF
			SET	LITERAL										OF	ON							
			SET	PLUS								ON		ON								
			SET	SETOP	EQ	LSTHOGND											X			X	X	
			SET	SETOP	+	1											X			X	X	
			SET	LOOOP	EQ	COLi											X			X	X	
			SET	LOOOP	−	1											X			X	X	
			MOVE	I,LOOOP (BEND)		OPHOLD											X			X	X	
			SET	SEKPSCANN	+	1		X														
			SET	BEGINTAG			ON				ON		OF	OF	ON	ON	OF		OF	OF	ON	
			SET	LSTHOGND	+	1	X		X	X	X	X	X	X	X	X			X		X	
			SET	COLi	+	1	X		X	X	X	X	X	X	X	X			X		X	
			SET	QUIT													ON					
			DO	GET				X														
			GO TO	GETSEKP				X														
			GO TO	TABOIII													X			X	X	
			GO TO	FINISHSEKP														X				
			GO TO	SEKPSCAN			X		X	X	X	X	X	X	X	X			X		X	

ACTION	NAME	OP	NAME	①	②	RULE NR
	NEXTIN	IS		ON	OFF	
SET	QUIT		OFF	X	X	
DO	PUTSEKP			X	X	
DO	TAB 0506			X	X	
MOVE	WORK		OUTPUT	X	X	
DO	PUT			X	X	
DO	GET			X	X	
SET	NEXTIN		OFF	X		
GOTO	TAB 0103					

IDENT RLD. 73-74

IF ON, NEXT CARD HAS ALREADY BEEN READ IN BY GETSEKP

PUT OUT ALL PROF CARDS

ENTER TAG IN EXPTABLE

PROBLEM:

Compute SET

Compute HOD ADDR

HOD OPERAND

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR. | DESCRIPTION | PROCESS AREA NR. | PAGE / TO PAGE

CONDITIONS: TABLE - GETSEXP 0214

ACTIONS: TABLE PUTSEXP 0215

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME	①	②	③	④
				INCOL 16-22	EQ	BLANKS	N	Y	Y	Y
				INCOL 23-24	EQ	BLANKS		Y	N	N
				SEXPCARDNO	GR	1Φ		N	Y	Y
			SET	NEXTIN		ON	X	X	X	X
			SET	COL-I	EQ	INCOL23			X	
			MOVE	(TITLE)		INOPER				X
			SET	BIGERROR		ON				X
			MOVE	{OVER 9 ATTENDANCES IN SCOP}		MESSAGE				X
			DO	PRINT						X
			SET	INCOL7576		BLANK			X	
			GO TO				FINISASEXP	FINISHSEXP	SEXPSCAN	FINISHSEXP

RULE NR.	ACTION	NAME	OP	NAME	①	②
		SEXPHOLD	13	ZERO	Y	N
	MOVE	I, WORKAOUT		OUTPUT		X
	DO	PUT				X
	SET	WORKAD	+	8Φ		X
	SET	SEXPHOLD	-	1		X
	SET	WORKAOUT	EQ	HIGHCARDSTR	X	
	SET	LOWERADDR	EQ	HIGHCARDSTR		
	SET	SEXPCARDNO	EQ	ZEROS	X	
	GO TO				( )	TABO215

FORM 8004-MLV AUG. 1970

DATA RULES

APPROVALS & DATES

PROCESS AREA CONTACT PERSONNEL

PREPARED BY

DATE

REVISION

PROJECT NR.

DESCRIPTION

PROCESS AREA NR. PAGE TO PAGE

CONDITIONS

TABLE LONGCHECK - 0216

ACTIONS

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME (1)	CP	NAME (2)	CP	NAME	RULE NR.
				NEXTWORKAD	GR	REC WORKLOAD	Y		N		
			MOVE	WORK		I, WORKADIN	X				
			SET	WORKADIN	+	80	X				
			SET	DEXPHOLD	+	1	X				
			SET	WORKIC	EQ	BLANKS	X				
			SET	NEXTCHAR	EQ	WORKCOL23	X				
			GO TO				X ( )		X ( )		

FORM 6004 REV AUG 1980

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE | REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA NR. | PAGE | TO PAGE

CONDITIONS: TABLE 208 MEXP

ACTIONS: TABLE - MEXP SCAN

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME
			MOVE	INTAG		TBLTAG
			SET	TBLCODE	EQ	1
			SET	TBLSEQNO	EQ	ZEROS
			MOVE	INTAG		WORKTAG
			MOVE	(RCD)		WORKOPER
			MOVE	INCOL 2329		WORKCOMP
			MOVE	INCOL 3035		WORKNU
			SET	WORKNU	+	INCOL 2829
			MOVE	WORKCOMP		LOCATOR
			DO	NAMELOK		
			MOVE	WORK		OUTPUT
			DO	PUT		
			MOVE	(HOP #4)		OUTPUT
			DO	PUT		
			MOVE	(MATH)		WORKOPER
			MOVE	INNU		WORKNU
			SET	INCOL3536		BLANKS
			GO TO	MEXPSCAN		

ACTION	NAME	OP	NAME	RULE NR
	I, COL	EQ		
	LASTBLANK	IS	ON	
	BEGINTAG	IS	ON	
	REDEFINE	IS	ON	
SET	LASTBLANK			
SET	BEGINTAG			
SET	REDEFINE			
SET	QUIT			
SET	SETUP	EQ	LSTHOLD	
SET	SETUP	-	1	
SET	LOOP	EQ	COLI	
SET	LOOP	-	1	
MOVE	I, LOOP(SUB)		OPHOLD	
SET DO	NEXTNUMERIC LOOKCHECK	EQ	NEXTCHAR	
MOVE	COLI		I, NEXTCHAR	
SET	NEXTCHAR	+	1	
SET	COLI	+	1	
SET	OPHOLDI	EQ	LSTOPHOLD	
SET	OPHOLDI	+	LSTHOLD	
SET	LSTHOLD	+	1	
GO TO	NUMBR DECK			
GO TO	MEXPOPR			
GO TO	MEXPSCAN			
DO	GET			
SET	SEXPCHAR	+	1	
DO	GETMEXP			
GO TO	MEXPSCAN			

FORM 8008 REV AUG 1980

Line Reference Target LRT-BE4-V2

DATA RULES

APPROVALS & DATES	PROCESS AREA CONTACT PERSONNEL	PREPARED BY	DATE / REVISION	PROJECT NR	DESCRIPTION	PROCESS AREA / NR	PAGE / TO PAGE

CONDITIONS

TABLE- MEXPOPR

ACTIONS

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME	OP	(1) NAME	OP	(2) NAME	OP	(3) NAME	OP	(4) NAME	OP	(5) NAME	OP	(6) NAME	OP	(7) NAME	OP	(8) NAME	RULE NR.
				REDEFINE	IS	ON	Y			Y	N		N		N		N		N		N		
				OPHOLD	IS	EXPTABLE					N		N		Y		Y		N		N		
				OPHOLD	IS	TOLTAB					N		N						Y		Y		
				QUIT	IS	ON	N			Y	N		Y		N		Y		N		Y		
			SET	NEXTWORKAD	BY	NEXTCHAR	X			X	X		X		X		X		X		X		
			SET	NEXTWORKAD	+	LGTHOUND	X			X	X		X		X		X		X		X		
			DO	LONGCHECK			X			X	Y		Y		X		X		X		X		
			SET	NEXTCHAR	+	LGTHOUND	X			X	X		X		X		X		X		X		
			MOVE	OPHOLD		I3 NEXTCHAR	X			X	X		X		X		X		X		X		
			MOVE	(CTPLNK)		OUTOPEN									X		X						
			MOVE	OPHOLD		OUTUPND									X		X						
			DO	PUT											X		X						
			MOVE	(EXP REPEAT ITSELF)		MESSAGE													X		X		
			SET	BIGERROR		ON													X		X		
			DO	PRINT															X		X		
			SET	NEXTCHAR	+	1	X			X	X		X		X		X		X		X		
			MOVE	(LOOP)		LGTHOUND	X			X	X		X		X		X		X		X		
			SET	NEXTWORKAD	BY	NEXTCHAR	X			X	X		X		X		X		X		X		
			DO	LONGCHECK			X			X	X		X		X		X		X		X		
			MOVE	COLi		I3 NEXTCHAR	X			X	X		X		X		X		X		X		
			SET	NEXTCHAR	+	1	X			X	X		X		X		X		X		X		
			SET	COLi	+	1	X			X	X		X		X		X		X		X		
			GO TO	MEXPSCAN			X				X				X				X				
			GO TO	MEXPQUIT				X				X					X				X		

FORM 8004 REV AUG 1980

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR. | DESCRIPTION | PROCESS AREA / NR. | PAGE / TO PAGE

CONDITIONS: TABLE- NUM OR DECK

ACTIONS

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME	① OP	① NAME	② OP	② NAME	③ OP	③ NAME	④ OP	④ NAME	⑤ OP	⑤ NAME	⑥ OP	⑥ NAME	⑦ OP	⑦ NAME	RULE NR.
				OPHOLDi	EQ	(,)	Y		Y		N		N		N		Y		N		
				OPHOLDi	IS	NUM					Y		Y		N				Y		
				OPHOLDi	EQ	LGOPHOLDi	N		Y		N		Y				Y		Y		
				QUIT	IS				OF				OF				ON		ON		
			SET	OPHOLDi	-	1	X				X										
			SET	NEXTWORKAD	EQ	NEXTCHAR			X				X				X		X		
			SET	NEXTWORKAD	+	LGTHOPND			X				X				Y		X		
			SET	NEXTWORKAD	+	3			X				X				X		X		
			DO	LONGCHECK					X				X				X		X		
			MOVE	(#+1)		I, NEXTCHAR			X				X				X		X		
			SET	NEXTCHAR	+	LGTHOPND			X				X				X		X		
			SET	NEXTCHAR	+	2			X				X				X		X		
			MOVE	OPHOLD		I, NEXTCHAR			X				X				X		X		
			SET	NEXTCHAR	+	1			X				X				X		X		
			MOVE	(#)		I, NEXTCHAR			X				X				Y		X		
			SET	NEXTCHAR	+	1			X				X				X		X		
			MOVE	(-0001)		LGTHOPND			X				X				X		X		
			SET	NEXTWORKAD	EQ	NEXTCHAR			X				X				X		X		
			DO	LONGCHECK					X				X				X		X		
			MOVE	COLi		I, NEXTCHAR			X				X				X		X		
			SET	NEXTCHAR	+	1			X				X				X		X		
			SET	COLi	+	1			X				X				X		X		
			GO TO	NUMOR DECK				X				X									
			GO TO	NEXPOPR												X					
			GOTO	NEXPSCAN					X												
			GOTO	NEXP QUIT													X		X		

FORM 8004 REV 11/0. 1960

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS: TABLE - GET MEXP

ACTIONS: TABLE - MEXPQUIT

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME	① OP	① NAME	② OP	② NAME	③ OP	③ NAME	④ OP	④ NAME
				INCOL 16-22	EQ	BLANKS	N		Y		Y		Y	
				INCOL 23-24	EQ	BLANKS			Y		N		N	
				SEXPCNKDNO	GR	10					N		Y	
			SET	NEXTIN		ON	X		X				X	
			SET	COLI	EQ	INCOL33					X			
			MOVE	(TITLE)		INOPER							X	
			SET	BLKERROR		ON							X	
			MOVE	(OVERIN OUTCOL33 IN)		MESSAGE							X	
			DO	PRINT									X	
			SET	INCOL7576		BLANK					X			
			GO TO				MEXPQUIT	MEXPQUIT	(	)	MEXPQUIT			

RULE NR.	ACTION	NAME	OP	NAME	① OP	① NAME	② OP	② NAME
		QUIT			Y		Y	
		NEXTIN			N		Y	
	SET	NEXTIN					OF	
	SET	QUIT			OF		OF	
	SET	LASTBLANK						
	SET	COLEPLUS1	EQ	COLC	X			
	SET	COLIPLUS1	+	1	X			
	DO	OVETAGCHK			X			
	SET	NEXTWORKRD	EQ	NEXTCHAR	X		X	
	SET	NEXTWORKRD	+	10	X		X	
	SET	NEXTWORKRD	+	3	X			
	SET	NEXTWORKRD	+	OUTTABLATH	X			
	DO	LONGCHECK			X		X	
	MOVE	(@)		I, NEXTCHAR				
	SET	NEXTCHAR	+	1				
	MOVE	TBLTAB		I, NEXTCHAR	X		X	
	SET	NEXTCHAR	+	10	X			
	MOVE	DASHES 3		I, NEXTCHAR	X			
	SET	NEXTCHAR	+	3	X			
	MOVE	I, COLI		I, NEXTCHAR	X			
	DO	TAB0306			X		X	
	SET	NEXTCHAR	EQ	BLANKS	X		X	
	SET	COLC	EQ	REINCOL33				
	SET	OUTTABLATH	EQ	ERROR				
	DO	PUTSEXP			X		X	
	MOVE	WORK		OUTPUT	X		X	
	SET	WORK	EQ	BLANKS	X		X	
	DO	PUT			X		X	
	MOVE	(TITLE)		OUTOPER	X		X	
	MOVE	TBLTAB		OUTOPND	X		X	
	MOVE	(@)		OUTCOL33	X		X	
	DO	PUT			X		X	
	DO	GET			X			
	GO TO	TAB0103			X		X	

FORM 8004 REV AUG 1980

Line Reference Target LRT-BE4-V2

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE | REVISION | PROJECT NR. | DESCRIPTION | PROCESS AREA | NR. | PAGE | TO PAGE

TABLE — OVFTAGCHK

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	CONDITIONS			ACTIONS				RULE NR.
			I_2 COLIplus1 eq (COL)			Y	N	N	N	
			I_2 COLI eq (CL)			Y	N	N	N	
			I_2 COLIplus1 eq (0)				N	Y		
			DO	GETMEXP		X				
			SET	I_2 COLIplus1	COLI	X				
			SET	I_2 COLIplus1	+ 1	X	X	X	X	
			SET	OVTAGCTR	= 1			X		
			SET	COLI	+ 1		X			
			GO TO			OVFTAGCHK	OVFTAGCHK	OVFTAGCHK	MEXPQUIT	

FORM 8004 REV AUG 1980

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS

ACTIONS

TABLE 301 - END CARD ROUTINE

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME	OP	① NAME	OP	② NAME	OP	③ NAME	OP	④ NAME	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	RULE NR
				AUTOCODER	IS			ON		ON		ON		OFF	OFF	OFF	ON	ON	ON	OFF	OFF	OFF	
				BILERROR				OFF		ON		OFF		ON	OFF	OFF	OFF	ON	OFF	ON	OFF	OFF	
				ERRORIND				OFF				ON			ON	OFF	OFF		ON		ON	OFF	
				INTAG	EQ	FINAL		OFF		OFF		OFF		OFF	OFF	OFF	ON	ON	ON	ON	ON	ON	
			SET	AUTOCODER	OF		X		X		X						X	X	X				
			MOVE	PROGID		TMSGID	X		X		X		X	X	X	X	X	X	X	X	X	X	
			MOVE	(TRINS)		OUTOPER	X		X		X						X	X	X				
			MOVE	NAME SAVE		OUTOPND	X		X		X						X	X	X				
			DO	PUT			X		X		X						X	X	X				
			SET	TMSGERRTYPE				(NO)		(MAJ)		(MIN)		(MAJ)	(MIN)	(NO)	(NO)	(MAJ)	(MIN)	(MAJ)	(MIN)	(NO)	
			MOVE	INPUT		OUTPUT																	
			DO	PUT			X		X		X		X	X	X	X	X	X	X	X	X	X	
			MOVE	(E)		TMSGAP											X	X	X	X	X	X	
			SET	ERRORIND		(3)	X		X		X		X	X	X	X							
			TYPE	TMSG			X		X		X		X	X	X	X	X	X	X	X	X	X	
			CLOSE	INPUT													X	X	X	X	X	X	
			CLOSE	OUTPUT													X	X	X	X	X	X	
			CLOSE	PRINT													X	X	X	X	X	X	
			TYPE	(EOJ#)													X	X	X	X	X	X	
			GO TO					TAB0900		TAB0800		TAB0900		TAB0900	TAB0900	TAB0500	TABSTOP	TABSTOP	TABSTOP	TABSTOP	TABSTOP	TABSTOP	

FORM 3004 REV. AUG. 1960

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS

TABLE 302 - (GET)

CLOSED TABLE

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME (1)	OP	NAME (2)	OP	NAME (3)	OP	NAME (4)
				Buffer 1	IS	ON	Y		N		Y	
				Control Sw	IS	OFF	N		N		Y	
			SET	Buffer 1				OFF		ON		OFF		ON
			SET	INPUT	EQ			Buffer 1		Buffer 2		Buffer 1		Buffer 1
			IORD	TAIN			X		X		X		X	
			DO	JSCCK			X		X					
			GO TO	( )			X		X		X		X	

TABLE 303 - (SEQCK) CLOSED TABLE

ACTION	NAME (1)	OP	NAME (2)	OP	NAME (3)	OP	NAME (4)
	INPLINE	HI	LAST PLINE	Y		N	
MOVE	INPLINE		LAST PLINE	X		X	
MOVE	(OUTDEVICE)		MESSAGE			X	
DO	PRINT					X	
GO TO	( )			X		X	

TABLE 304

(PUT)

CLOSED TABLE

ACTION	NAME	OP	NAME	RULE NR
SET	PGLNE	+	1	
MOVE	PROGID		OUTID	
IOCMP	TPOUT			
SET	OUTPUT		BLANKS	
GO TO	( )			

OUTPUT IS DUMPED TO END OF OCTANT 79930 - 79999

TABLE 305

(PRINT) CLOSED TABLE

ACTION	NAME	OP	NAME
SET	ERROKIND		ON
PRINT	MESSAGE		
SET	MESSAGE	EQ	BLANKS
GO TO	( )		

MACRO INSTRUCTION WHICH PUTS MESSAGE, CHECKS LINE COUNT. IF COUNT = 0, NEXT TIME PUT WILL BE PRECEDED BY PAGE HEADINGS

FORM 8004 REV AUG. 1970

DATA RULES

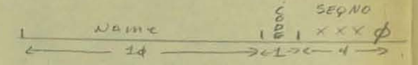
APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS

ACTIONS

TABLE 306 - ENTER ITEM IN EXPRESSION TABLE

CLOSED TABLE



RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME	OP	NAME ①	OP	NAME ②	OP	NAME ③	OP	NAME ④	OP	NAME ⑤	OP	NAME ⑥		RULE NR.
				ENTRYSEQNO	HI	25	Y		N		N		N		N		ONLY 25 RLIST ENTRIES ALLOWED - IN ASSOC TAG TABLE.	
				THISENTRYADR	VS	I, LASTTABADR			HI		EQ		LR		EQ		BEGIN AT RIGTH END OF TABLE, LOOKING FOR 1ST HI.	
				ENTRYSEQNO	IS	ZERO			Y		Y				N		IF NOT Ø - NOT RLIST AND NO ASSOCIATED TAG	
			MOVE	(OVER 25 RLIST ENTRIES)		MESSAGE	X										MORE THAN 25 RLISTS - GIVE MESSAGE & TREAT AS VLIST	
			DO	PRINT			X											
			SET	ENTRYSEQNO	EQ	ZEROS	X											
			SET	I, LASTABENT				THISENTRY		THISENTRY		I+ NEXTENTRY		THISENTRY		THISENTRY	IF LOW - MOVE ITEM IN TABLE TO RIGTH	
			SET	LASTABENT	-	15					X						SHIFT TO ADDRESS ITEM ON LEFT OF THIS ONE	
			SET	NEXTENTRY	-	15					X						FOR NEXT CMP & SAVE	
			SET	LASTABREG	-	15					X						SHIFT TO ADDRESS OF THIS ITEM FOR RCV	
			MOVE	(DUPLICATE ENTRY TAGS)		MESSAGE			X						X		DUPLICATE TAGS - GIVE ERROR MSG (CRITICAL), BUT	
			SET	BIGERROR		ON			X						X		ENTER ITEM AS IF HI.	
			DO	PRINT					X						X			
			SET	LASTABENT		NEXTENTRY		X		X				X		X	IF HI/EQ - MOVE ENTRY TO TABLE, UPDATE ADDR	
			SET	TABLECOUNT	+	1		X		X				X		X	OF LAST ITEM IN TABLE BY 15	
			SET	FUNTOLBASE	+	SEQNO		X		X								
			MOVE	ASSOCTAG		I, FUNTOLBASE		X		X							ASSOC TAG ADDR = (HI ASSOC TAG TABLE -1Ø) + SEQNO	
			SET	NEXTENTRY		LASTABENT		X		X				X				
			SET	NEXTENTRY	+	15		X		X				X		X	SET UP FIRST RCV ADDR for NEXT LOOP.	
			GO TO				TABØ306	( )		( )		TABØ306		( )		( )	LOOP OR EXIT	

FORM 8004 REV AUG 1960

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS | ACTIONS

TABLE 307 - SEARCH EXPRESSION TABLE

CLOSED TABLE

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	(1) NAME	OP	(1) NAME	IF	(2) NAME	OP	(3) NAME		RULE NR
				QUER	IS	EXPTABLE	N		Y		Y	IF YES - ARGUMENT IS PLACED IN TAGFCNADDR. -> 1 POS TAGCODE 4 POS SEQ NO.	
				TAGCODE	IS	Ø			N		Y		
			SET	TAGCODE		(+Ø)	X					SET A BIT IN TAG CODE TO INDICATE NO FIND	
			SET	TAGFUNADDR	+	FUNTBLBASE					X	IF A LIST - COMPUTE ADDR OF ASSOC TAG = (HI@ ASSOC TAG TABLE -10) + SEQ NO.	
			GO TO	( )			X		X		X	EXIT	

FORM 8004 REV AUG 1980

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA NR | PAGE / TO PAGE

CONDITIONS

ACTIONS

TABLE 900 - DECISIONS FOLLOWING TYPED MESSAGES

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY														RULE NR
			ERROR IND EQ		1	1	2	2	3	3						
			252		ON	OF	ON	OF	ON	OF						
			253			ON		ON		ON						
			MOVE	(A3MSL)	OUTOPER	X			X							
			MOVE	(ORIGINAL)	OUTOPND	X			X							
			DO	PUT		X			X							
			DO	GET			X	X		X						
			CLOSE	INPUT							X					
			CLOSE	OUTPUT							X					
			CLOSE	PRINT							X					
			TYPE	(EOJE)							X					
			DO	PROCEED TO PROGRAM HSK						X						
			GOTO			TAB0101	TAB0300	TAB0101	TABSTOP	TAB0101	TABSTOP					

FORM 9004 REV AUG 1960

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS | ACTIONS

TABLE 401 - TESTATEMENT

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME 1	OP	NAME 2	
			SET	BASEINNUMBER	EQ	ON	
			MOVE	SETUPTAB	TO	WORKTAB	
			MOVE	(TEST)	TO	WORKOPER	
			DO	TABLGTH			FINDS TAB LENGTH & STORES AT 11TH (SEE 3 BELOW)
			GO TO	TAB 404			

TABLE 405 - RLISTFIN

ACTION	NAME 1	OP	NAME 2
MOVE	(---)		NEXTCHAR, 3
SET	NEXTCHAR	+	3
MOVE	I, TABFUNNO		NEXTCHAR, 10
SET	NEXTCHAR	+	10
GO TO	TAB 406		

TABLE 407 - OPERROR

ACTION	NAME 1	OP	NAME 2	RULE NR
MOVE	SETUPOP	TO	MESSAGE	
DO	PRINT			
SET	RLCODE	EQ	(E)	
SET	RLTH	EQ	+1	
GO TO	TAB 410			

TABLE 404 - LINKAGE

ACTION	NAME 1	OP	NAME 2
MOVE	WORKTAB		OUTTAB
SET	WORKTAB	EQ	BLANKS
MOVE	(TPUNK)	TO	OUTOPER
MOVE	SETUPUNIT	TO	OUTPUNB
MOVE	(4)	TO	OUTCOL33
DO	PUT		
GO TO	( )		

TABLE 406 - TEST FIN

ACTION	NAME 1	OP	NAME 2	(1)	(2)
	SETUPFALSE	EQ	BLANKS	Y	N
	SETUPTRUE	EQ	BLANKS		Y
MOVE	(.)	TO	NEXTCHAR, 1		
SET	NEXTCHAR	+	1		
MOVE	SETUPTRUE	TO	NEXTCHAR, 10		
SET	NEXTCHAR	+	10		
MOVE	(.)	TO	NEXTCHAR, 1		
MOVE	SETUPFALSE	TO	NEXTCHAR, 10		
SET	NEXTCHAR	+	10		
MOVE	(.)	TO	NEXTCHAR, 1		
SET	NEXTCHAR	EQ	KEYWORDEND		
MOVE	WORK	TO	OUTPUT		
DO	PUT				
SET	WORK	EQ	BLANKS		
GO TO	WHILE				

FORM 8004 REV AUG. 1960

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE | REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA | NR | PAGE | TO PAGE

CONDITIONS

TABLE 402 — CODE CONDITIONS TO TEST STATEMENTS

RULE NUMBER CURRENT	PRIOR	WEEKLY FREQUENCY	ACTION	NAME 1	OP	NAME 2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
		1		L,SETUPNAM1	EQ	(#)	Y	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	ALREADY A LITERAL
		2		SETUPNAM1	IS	NUMERIC				Y	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N	NEEDS # SYMBOLS
		3		SETUPNAM1	IS	EXPTABLE							Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	IF Y, FUN = TAG FUNADDR
		4		TAGCODE	EQ	Ø							Y	Y	Y	Y	N	N	N	N	N				LIST NAME
		5		TAGCODE	EQ	1											Y	Y	Y	N	N				EXPRESSION
		6		TAGCODE	EQ	4														Y	Y				STATE RELATION
		9		SETUPOP	IS	OPTABLE		Y	N		Y	N						Y	N				Y	N	IF Y, FUN = RCODE, RLTH
		7		TAGFUNADDR	IS	ZERO							Y	Y	N	N									VLIST NAME
		8		BASEMEMBER	IS	ON	Y	N	N	Y	N	N	Y	N	Y	N	Y	N	N	Y	N	Y	N	N	INDICATES 1ST PASS
			SET	RCODE	TO									1M		1M					TF ADDR				
			SET	RLTH	TO									(42)		(42)					TF [illegible]				
		✓	GO TO	MEM2CHR									X		X					X					
			DO	LINKAGE													X	X							CREATE LINK TO EXP.
			DO	OPERROR					X			X							X					X	GIVE ERROR MSG, ASSUME EQ
			MOVE	RCODE	TO	NEXTCHAR, RLTH		X	X		X	X		X		X		X	X		X		X	X	
			SET	NEXTCHAR	+	RLTH		X	X		X	X		X		X		X	X		X		X	X	
			MOVE	(5)		NEXTCHAR, 1		X	X		X	X		X		X		X	X		X		X	X	
			SET	NEXTCHAR	+	1		X	X		X	X		X		X		X	X		X		X	X	
			MOVE	(#)		NEXTCHAR, 1				X	X	X													
			SET	NEXTCHAR	+	1				X	X	X													
			MOVE	SETUPNAM1		NEXTCHAR, TLTH	X	X	X	X	X	X		X		X	X	X	X		X	X	X	X	
			SET	NEXTCHAR	+	TLTH	X	X	X	X	X	X		X		X	X	X	X		X	X	X	>	
			MOVE	(#)		NEXTCHAR, 1				X	X	X													
			SET	NEXTCHAR	+	1				X	X	X													
			MOVE	(5)		NEXTCHAR, 1	X			X							X					X			
			SET	NEXTCHAR	+	1	X			X							X					X			
			REFLD	SETUPNAM1		SETUPNAM2	X			X							X					X			REVERSES NAMES 1 & 2
			SET	BASEMEM	TO	OFF	X			X							X					X			
			DO	RLISTFIN										X											
			GO TO	TESTFIN				X	X		X	X		X		X		X	X		X		X	X	
			GO TO	TAB0402			X			X							X					X			

ACTIONS

TABLE 403 MEM2CHK

ACTION	NAM 1	OP	NAME 2	1	2	3	4	5	6	7	RULE NR
	TAGCODE	EQ	Ø	Y	Y	Y	N	N	N	N	
	TAGCODE	EQ	1				Y	Y	Y	Y	
	SETUPNAM2	IS	EXPTABLE	N	Y	Y	N	Y	Y	Y	
	TAGCODE	IS	Ø		Y	N		N	Y	N	
	TAGCODE	IS	1					Y	N	N	
REFLD	SETUPNAM1		SETUPNAM2	X		X	X			X	
MOVE	SETUPNAM1	TO	MESSAGE		X			X	X		
MOVE	SETUPNAM2	TO	MESSAGE		X			X			
PUT	MESSAGE				X			X	X		
MOVE	SETUPNAM1		NEXTCHAR, TLTH		X			X	X		
SET	NEXTCHAR	+	TLGTH		X			X	X		
MOVE	(5)		NEXTCHAR, 1		X			X	X		
SET	NEXTCHAR	+	1		X			X	X		
SETOF	BASEMEM				X			X	X		
GO TO	TAB0401				X	X	X	X	X	X	

FORM 8004 REV AUG 1960

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS | ACTIONS

TABLE 500 - ACTION

TAB 0503 - STATE

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME1	OP	NAME2
			SET	BASENUMBER	EQ	0N
			MOVE	SETUPTAG	TO	WORKTAG
			MOVE	SETUPACT	TO	ACTIONOP
			MOVE	SETUPOP	TO	ACTIONOP+5
			GO TO	ACTOPSRCH		

TAB 500A - ACTOPSRCH

ACTION	NAME1	OP	NAME2	①	②	③	④	⑤	⑥
	ACTIONOP	IS	ACTOPTBLE	N	Y	Y	Y	Y	Y
	INDTRANS	IS	ON		Y	N	N	N	N
	REVQNDS	IS	ON			Y	N	Y	N
	QNDSREQ	IS	ON			N	Y	Y	N
REFLD	SETUPNAM1		SETUPNAM2			X		X	
MOVE	SETUPNAM1	TO	SETUPFALSE				X	X	
MOVE	MACROHEAD	TO	WORKQPER			X	X	X	X
MOVE	(NOP)	TO	WORKQPER	X					
MOVE	SETUPNAM1	TO	WORKQPND	X					
DO	PUT			X					
MOVE	ACTIONOP	TO	MESSAGE	X					
DO	PRINT			X					
SET	BIGERROR	EQ	ON	X					
GO TO	MIT#I			X					
GO TO	I, MACROHEAD				X				
GO TO	TAB#502					Y	X	X	X

FUN = 3 pos code - 1 - Revise Format
2 - Valid for op1
3 - Valid for op2

5 pos Macro hdr or Ind. Trm.
Ind. Trm for MOVEV CLOSE DO OPEN GO TO

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME1	OP	NAME2
				L, SETUPACT+1	EQ	
			JET	WORKQPER	EQ	
			MOVE	SETUPACT	TO	MESSAGE
			DO	PRINT		
			MOVE	SETUPNAM1		NEXTCHAR, TLTH
			SET	NEXTCHAR	+	TLTH
			MOVE	(C#)		NEXTCHAR, 1
			SET	NEXTCHAR	+	1
			GO TO	0502		

RULE NR	(ON)	(OF)	(ZE)	(BL)	ELSE
	(SETON)	(SETOF)	(ZERO)	(BLANK)	
					X
					X
	X	X	X	X	
					X
					X
					X
	X	X	X	X	X

FORM 5004 REV AUG 1980

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS

TABLE 501 - ACTION CODER

ACTIONS

TAB 502 - END ACTOP

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME1	OP	NAME2	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫					①	②	③	④	RULE NR
				L, SETUPNAM1	EQ		(#)	(-)	(.)											BASEMEMBER	IS	ON	Y	Y	N	N	
				SETUPNAM1	IS	NUMERIC				Y	N	N		N	N	N	N			OPER3REQ	IS	ON	Y	N			
				SETUPNAM1	IS	EXPTABLE					Y	Y	Y	Y	Y	Y	Y			OPER3REQ	IS	ON			N	Y	
				TAGCODE	EQ	(4)					Y	Y	N	N	N	N											
				TAGCODE	EQ	(L)							Y	Y	N	N											
				TAGCODE	EQ	(0)									Y	N				SET	BASEMEMBER	EQ	OFF	X	X		
				TAGCODE	EQ	(-)										Y	Y	Y		MOVE	SETUPNAM1	TO	SETUPNAM1	X			
				POINTVALID	IS	ON										Y	Y	N		MOVE	OP2	TO	OP1	X			
				EXPVALID	IS	ON								Y	N					MOVE	SETUPFALSE	TO	NEXTCHAR, 10				X
				STATEVALID	IS	ON					Y	N								SET	NEXTCHAR	+	10				X
				ACTIONOP	EQ	(SET EQ)										Y	N			MOVE	(#)	TO	NEXTCHAR, 1				X
																				MOVE	WORK	TO	OUTPUT		X	X	X
			DO	TAGLGTH			X	X	X	X	X	X	X	X	X	X	X	X	FIND TAG LENGTH & STORE AT TLTH (NOT SHOWN)	DO	PUT				X	X	X
			MOVE	SETUPNAM1		MESSAGE					X		X	X			X		PUT OPND IN ERROR-MSG, CONSIDER	SET	NEXTCHAR	EQ	LC WORK END		X	X	X
			DO	PRINT							X		X	X			X		AS TAG ONLY	SET	WORK	EQ	BLANKS		X	Y	X
			DO	LINKAGE										X					CREATE LINK TO EXPRESSION	GO TO	TAB0501			X			
			MOVE	CINIPT	TO	WORKOPER											X			GO TO	MIT#I				X	X	X
			MOVE	TAGFUNDOL	TO	NEXTCHAR, 3											X										
			SET	NEXTCHAR	+	3											X										
			MOVE	(#)	TO	NEXTCHAR, 1		X	X	X																	
			SET	NEXTCHAR	+	1		X	X	X																	
			MOVE	SETUPNAM1	TO	NEXTCHAR, TLTH	X	X	X	X		X	X	X	X	X	X	X									
			SET	NEXTCHAR	+	TLTH	X	X	X	X		X	X	X	X	X	X	X									
			MOVE	(#)		NEXTCHAR, 1		X	X	X																	
			SET	NEXTCHAR	+	1		X	X	X																	
			MOVE	(0)		NEXTCHAR, 1	X	X	X	X		X	X	X	X	X	X	X									
			SET	NEXTCHAR	+	1	X	X	X	X		X	X	X	X	X	X	X									
			GO TO	END ACTOP			X	X	X	Y		X	X	X	X	X	X	X									
			GO TO	STATE							X																

FORM 8004 REV AUG. 1980

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR. | DESCRIPTION | PROCESS AREA / NR. | PAGE / TO PAGE

CONDITIONS | ACTIONS

TAB 504 - GO TO & DO ROUTINE

TAB 506 - CLOSE ROUTINE

RULE NUMBER CURRENT	RULE NUMBER PRIOR	WEEKLY FREQUENCY	ACTION	NAME 1	OP	NAME 2	①	②	③	④
				SETUPACT	EQ	(GO TO)	Y	Y	N	N
				SETUPACT	EQ	(DO)			Y	Y
				SETUP NAM1-6	IS	NUMERIC	Y	N	Y	N
			MOVE	(TO)		OUTOPER	X	X		
			MOVE	(LINK)		OUTOPER			X	X
			MOVE	(TAB)		OUTOPND	X		X	
			MOVE	SETUPNAM1		OUTOPND		X		X
			MOVE	SETUPNAM1		OUTOPND+3	X		X	
			MOVE	(Φ)		OUTCOL33	X	X	X	X
			DO	PUT			X	X	X	X
			GO TO	MITEI			X	X	X	X

ACTION	NAME 1	OP	NAME 2	RULE NR
MOVE	(IOPUT)		OUTOPER	
MOVE	SETUPNAM 1		OUTOPND	
MOVE	(C Φ)		OUTCOL33	
MOVE	(CLSFINAL Φ)		OUTCOL35	
DO	PUT			
GO TO	MITEI			

TAB 505 - OPEN ROUTINE

ACTION	NAME 1	OP	NAME 2
MOVE	(LOGIC CLASS Φ)		OUTOPER
DO	PUT		
GO TO	MITEI		

FORM 8004 REV AUG. 1960

DATA RULES

APPROVALS & DATES | PROCESS AREA CONTACT PERSONNEL | PREPARED BY | DATE / REVISION | PROJECT NR. | DESCRIPTION | PROCESS AREA / NR | PAGE / TO PAGE

CONDITIONS | ACTIONS

TABLE 507 - MOVEV-INITIAL

RULE NUMBER CURRENT / PRIOR | WEEKLY FREQUENCY

ACTION	NAME1	OP	NAME2
MOVE	(MOVEV)	*	WORKOPER
MOVE	(LTD)		WORKNU
GO TO	TAB0508		

TABLE 508 - INITIAL AND TEST

ACTION	NAME1	OP	NAME2	①	②	③	④	⑤
	L,SETUPNAM1	EQ		(#)	(+)	(-)		
	SETUPNAM1	IS	NUMERIC				Y	N
GO TO	TAB0509			X	X	X	X	
SET	SETUPNIL	EQ	L①SETUPNAM1					X
DO	TALTH							X
GO TO	TAB0510							X

TABLE 509 - END LOOP

ACTION	NAME1	OP	NAME2	①	②
	BASEMEMBER	IS	ON	Y	N
SET	WORKNU	EQ	BLANKS	X	
REFLD	SETUPNAM1		SETUPNAM2	X	X
SET	BASEMEMBER	EQ	OFF	X	
MOVE	ACTIONOP		MESSAGE		X
DO	PRINT				X
SET	BILGELOCK	EQ	ON		X
SET	SETUPACT	EQ	(MOVE)		X
GO TO	TAB0508			X	
GO TO	TAB0500				X

ERROR MSG-NO POINT
CRITICAL ERROR
ASSUME MOVE

FORM 8004 REV AUG 1980

TABLE 510 - COMMA HUNT

ACTION	NAME1	OP	NAME2	①	②	③	④	⑤
	TLTH	IS	ZERO	N	Y	Y	Y	N
	SETUPNIL	EQ	(,)	N				Y
	SETUPNAM1	IS	EXPTABLE		Y	Y	N	
	TAGCODE	EQ	(-)		Y	N		
SET	SETUPNIL	+	1	X				
SET	TLTH	-	1	X				
GO TO	TAB0510			X				
MOVE	SETUPNIL,TLTH		WORKCOL33					X
SET	NEXTCHAR	+	12					X
SET	NEXTCHAR	+	TLTH					X
DO	TAB0511							X
MOVE	(◆)		I,NEXTCHAR					X
MOVE	(◆)		WORKCOL33		X			X
SET	SETUPNIL	EQ	BLANK,TLTH					X
MOVE	SETUPNAM1	TO	WORKOPND		X			X
GO TO	TAB0512							X
MOVE	(◆)		WORKCOL34		X			
MOVE	TALENMODE,3		WORKCOL35		X			
MOVE	(◆)		WORKCOL38		X			
SET	NEXTCHAR	+	16		X			
REFLD	SETUPNAM1		SETUPNAM2		X			
SET	BASEMEMBER	EQ	OFF		X			
GO TO	TAB0500				X			
GO TO	TAB0509					X	X	

TAB 0511 - CHECK MODIFIER FOR EXP

ACTION	NAME1	OP	NAME2	①	②	
	WORKCOL29,30	IS	EXPTABLE	Y	N	also checks for code 1
MOVE	WORKTAG	TO	OUTTAG	X		
MOVE	(TPLNK)	TO	OUTOPER	X		
MOVE	WORKCOL34,TLTH	TO	OUTOPND	X		
MOVE	(◆)		OUTCOL33	X		
DO	PUT			X		
SET	WORKTAG	EQ	BLANKS	Y		
GO TO	( )			X	X	

RULE NR

TABLE 0512 - EXP TABLE SEARCH FOR POINT NAME

ACTION	NAME1	OP	NAME2	①	②	③
	SETUPNAM1	IS	EXPTABLE	N	Y	Y
	TAGCODE	EQ	(-)		N	Y
SET	I,NEXTCHAR	EQ	(SEP)	X	X	
SET	I,NEXTCHAR	EQ	TAGENMODE,3			X
SET	NEXTCHAR	+	3	X	X	X
MOVE	(◆)	TO	I,NEXTCHAR	X	X	X
SET	NEXTCHAR	+	1	X	X	X
MOVE	SETUPNAM1	TO	MESSAGE	X	X	
DO	PRINT			X	X	
REFLD	SETUPNAM1		SETUPNAM2	X	X	X
SET	BASEMEMBER	EQ	OFF	X	X	X
GO TO	TAB0500			X	X	X