File No. S360-29
Form Y33-7000-0

Program Logic

IBM System/360 Conversion Aids:
FORTRAN IV-to-PL/I Language Conversion Program
for IBM System/360 Operating System

Program Number 360C-CV-710

This document describes the internal logic of the FORTRAN IV-to-PL/I (F) Language Conversion Program for the IBM System/360 Operating System.

Program logic manuals are intended for use by IBM system engineers involved in program maintenance, and by system programmers involved in altering the program design. Program logic information is not necessary for program operation and use; therefore, distribution of this manual is limited to persons with program maintenance or modification responsibilities.

PREFACE

This document describes the structure and functions of the FORTRAN IV-to-PL/I Language Conversion Program: its components, their functions, and the control flow among them. The detailed organization of each component and the instructions used to implement its functions are described in the program listing. This manual is intended to help the user find any portion of the listing he requires.

The manual consists of eight sections. Section 1 is an introduction to the Language Conversion Program. Section 2 describes the Control Phase of the program; sections 3, 4, 5, and 6 describe the four logical phases into which the program is divided.

The reader should have some familiarity with the contents of the following publications:

- _IBM System/360 FORTRAN IV Language,_ Form C28-6515
- _IBM System/360 Operating System, PL/I (F) Programmer's Guide,_ Form C28-6594
- _IBM System/360 Conversion Aids: FORTRAN IV-to-PL/I Language Conversion Program for IBM System/360 Operating System,_ Form C33-2002
- _IBM System/360 PL/I Reference Manual_ Form C28-8201

In addition, the reader would find it useful to be acquainted with the publications:

- _IBM System/360 Operating System, Job Control Language,_ Form C28-6539
- _IBM System/360 Operating System, PL/I Language Specifications,_ Form Y33-6003.

First Edition (January, 1969)

Specifications contained herein are subject to change from time to time. Any such changes will be reported in subsequent revisions or Technical Newletters.

Request for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM France, Centre d'Etudes et Fecherches, Programming Publications, Department 843, 06 - La Gaude, France.

FIGURES

The FORTRAN IV-to-PL/I Language Conversion Program, henceforth referred to in this publication as "the LCP," is intended to facilitate transition to PL/I by converting "error-free" System/360 Operating System FORTRAN IV source programs into PL/I (F) programs. (For the purposes of this publication, the converted PL/I programs are referred to as "target" programs.) The LCP is itself expressed principally in PL/I (F) language, and can be compiled, using a System/360 Operating System PL/I (F) compiler, to give a System/360 machine code version. Note that certain routines are expressed in System/360 Operating System Assembly Language and are compiled using a System/360 Operating System (F) Assembler. The latter can then be used to convert into PL/I the FORTRAN programs written by the user.

An "error-free" source program is one that meets the following requirements:

1. it consists of statements that are in strict conformity with the syntactic rules laid down in the publication IBM System/360 FORTRAN IV Language, Form C28-6515

2. It has been successfully compiled by the user's FORTRAN IV compiler, giving a System/360 machine code version that produces results in accordance with the user's intentions.

FORTRAN statements that do not conform to these rules of syntax (note 2 above) are not converted, and a message to that effect is printed in the output listing. A message is also issued when the conversion of a source statement is impossible for technical reasons.

Note that, the LCP will convert FORTRAN IV source programs written for current IBM systems other than the System/360. In this connection, however, the user should note that:

- From the point of view of language, all FORTRAN IV source programs can be converted by the LCP, subject to the restrictions indicated in Appendix C of IBM System/360 Conversion Aids: FORTRAN IV-to-PL/I Language Conversion Program for IBM System/360 Operating System, Form C33-2002.

- There is no guarantee that the converted programs will be correctly executed. This is due to differences in implementation (storage allocation, magnitude of data, etc.).

The LCP is divided into a Control Phase and the following four logical phases:

1. Initialization phase, which scans the FORTRAN source program and classifies the FORTRAN statements.

2. Phase 10, which converts executable FORTRAN statements and builds various tables from the specification statement.

3. Phase 20, (optional) which rearranges certain tables created during Phase 10.

4. Phase 30, which generates the PL/I declarations for the identifiers used in the program, together with all comments, statements and messages.

It should be noted that, as far as possible:

- Executable statements are converted one at a time.

- FORTRAN specification statements are converted by collecting declarations of source program identifiers.

TABLES

The LCP contains a number of tables. The way in which these tables are used by the individual routines is described in the relevant sections; the layout of the tables is shown in Appendix A.

OVERALL LOGIC OF THE LCP

Figure 1 illustrates the overall internal logic of the LCP and shows the relationship among the Control Phase and the 4 logical phases: Initialization Procedure, Phase 10, Phase 20, and Phase 30.

After receiving control from the System/360 Operating System, the Control Phase calls the Initialization Procedure, whose input is the FORTRAN source program on SYSIN. This phase examines the source program to determine whether the FORTRAN statements are executable or not, initializes various tables, and, as output, places FORTRAN statements on SYSUT1. Optionally, a listing of the source program can be printed on SYSPRNT. Control returns to the Control Phase which calls Phase 10.

Phase 10 converts, if possible, the executable FORTRAN statements and comments. Certain parts of these statements are placed in the tables with the contents of the specification statements. If the size of the tables exceeds the amount of main storage available, the tables are placed on SYSUT2.

Control again returns to the Control Phase, which either calls Phase 20 (under conditions described in the section 'Phase 20'), or calls Phase 30 directly.

Phase 20, if called, rearranges EQUIVALENCE and DATA tables and returns control to the Control Phase, which then calls Phase 30.

Phase 30 takes the converted statements (on SYSUT1) and the tables (which may be on SYSUT2) as its input. It generates the PL/I declarations for the identifiers used in the program and edits the converted statements, together with all comments and messages; (its output is in the form specified by the user).

On completion of Phase 30, control returns to the Return Procedure of the Control Phase which either calls the Initialization Procedure if another FORTRAN program must be converted, or returns to the Operating System.

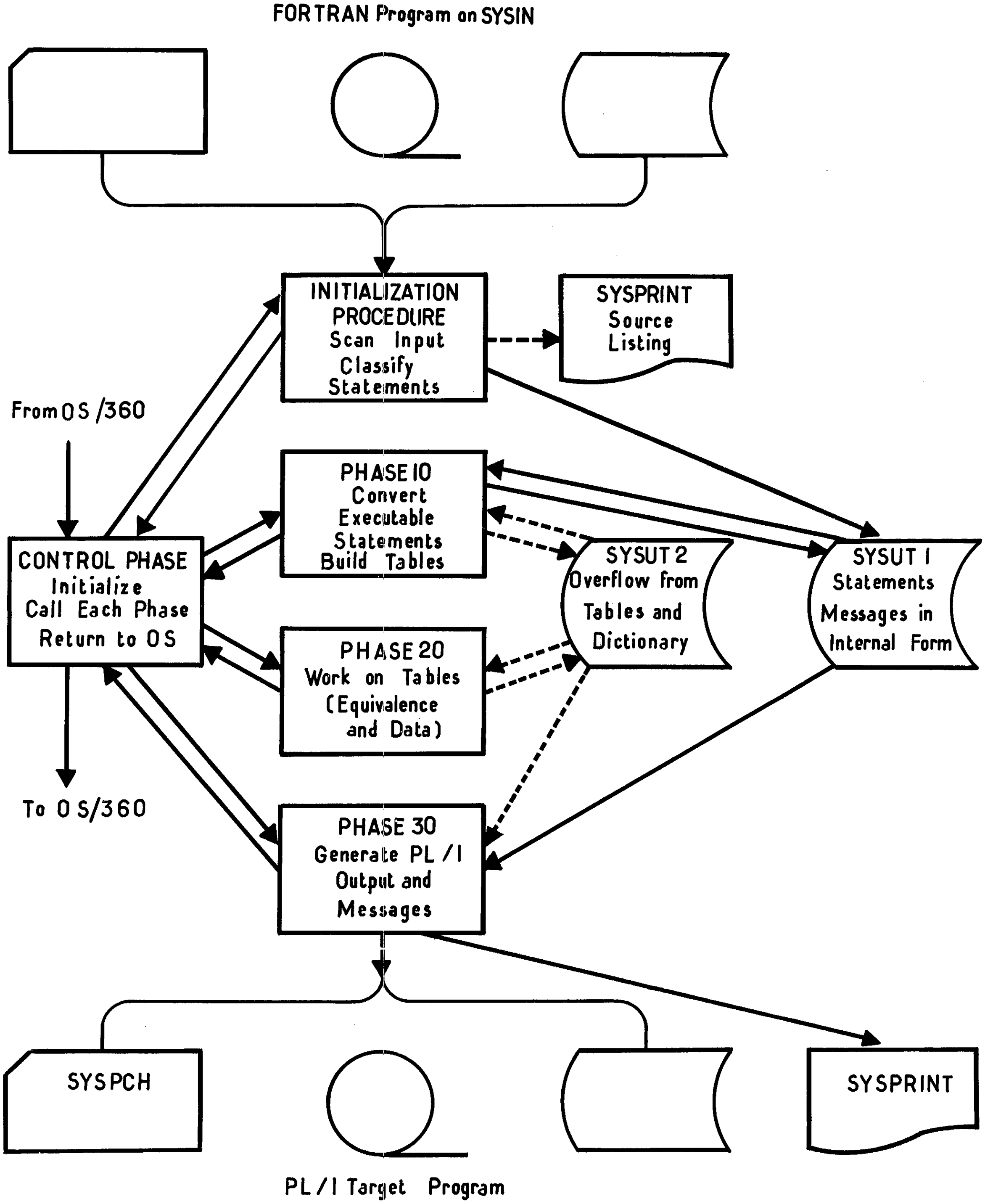

FORTRAN Program on SYSIN
INITIALIZATION PROCEDURE Scan Input Classify Statements
SYSPRINT Source Listing
From OS /360
PHASE 10 Convert Executable Statements Build Tables
CONTROL PHASE Initialize Call Each Phase Return to OS
SYSUT 2 Overflow from Tables and Dictionary
SYSUT 1 Statements Messages in Internal Form
PHASE 20 Work on Tables (Equivalence and Data)
To OS/360
PHASE 30 Generate PL/I Output and Messages
SYSPCH
SYSPRINT
PL/I Target Program

Figure 1. Overall Logic of the LCP

CONTROL PHASE

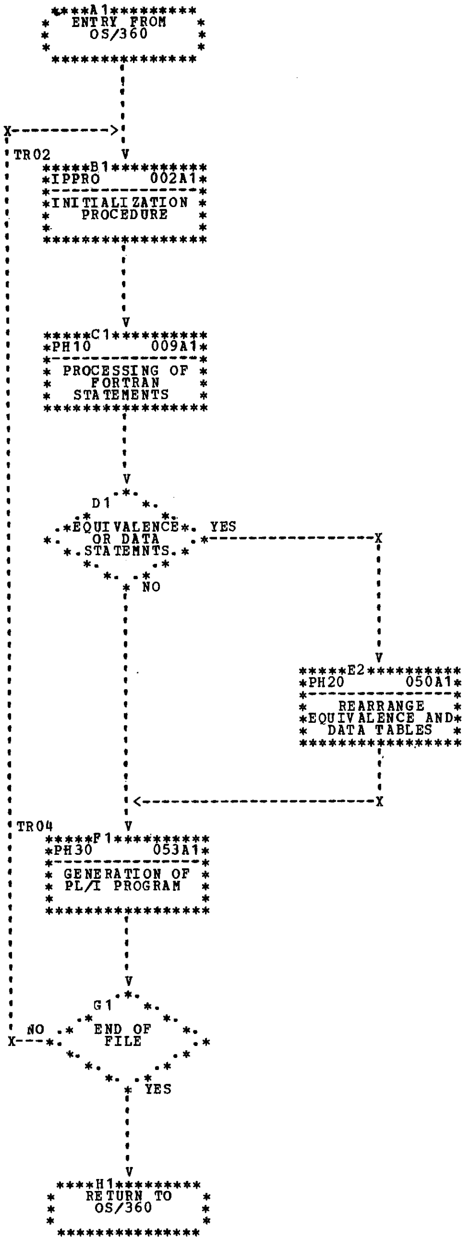
The LCP Control Phase (see Chart 001) receives control from the System/360 Operating System by means of an EXEC control card.

The Control Phase invokes the execution of the Initialization Procedure which is the first logical phase of the LCP.

On completion of each logical phase of the program, the following phase is called.

If, on completion of Phase 30, another FORTRAN program is next in the job stream, the LCP retains control and the Control Phase recalls the Initialization Procedure. After the last consecutive FORTRAN program in the job stream has been processed, the Control Phase restores control to the Operating System.

Chart 001. Overall Logic of Control Phase



INITIALIZATION PHASE

The Initialization phase is divided in three processing routines:

1. IPPRO, which initializes the settings of the switches in the LCP communication area.
2. GETCRD, which scans the source program and classifies the FORTRAN statement by type.
3. IMPRO, which processes the IMPLICIT specification statement.

The Initialization phase also makes use of the utility routines GETWRD, GETTBL, and PUTTBL. These routines are also used by Phase 10 and are described in the next chapter.

The input to the Initialization phase is the FORTRAN source program on SYSIN. The output of this phase is the FORTRAN source statements on SYSUT1.

If a FORTRAN source statement cannot be classified or if a syntactical error is detected during the scan, the statement is treated as a comment.

PROCESSING ROUTINES

This section contains detailed descriptions of the routines IPPRO, GETCRD and IMPRO. They are described in the following manner:

- Purpose of the routine
- Calling phase or routine (s)
- Entries (if more than one)
- Processing
- Routine (s) called
- Exit from the routine

IPPRO Chart 002

Purpose: To initialize the status of the switches in the LCP communication area, according to the options specified by the user in his control cards.

Called by: Control Phase

Processing: For the first FORTRAN program in a given batch, the contents of the PARM field in the EXEC control card indicates the status of the various switches. These are not modified when IPPRO is recalled to process subsequent FORTRAN programs in this batch.

The LCP control cards, if any, are scanned:

- any numeric field of two digits (maximum) is considered as a data-set reference number, and is marked as a PRINT file. This information is kept for use in the other phases.
- any FORTRAN mathematical function name is considered as a reserved word; this name is not converted into its equivalent built-in function name.

The original LCP control information is kept during processing of a single batch unless the user specifies new LCP control cards for a subsequent FORTRAN program in the batch.

For further information concerning the status of the switches, refer to the publication FORTRAN IV-to-PL/I Language Conversion Program for IBM System/360 Operating System, Form C33-2002, Appendix E, "Control Card Options." (Note that, for the sake of brevity, this publication will henceforth be referred to as "the language conversion manual.")

Routines called: None.

Exit: Go to GETCRD

GETCRD Charts 003, 004, 005, 006, 007

Purpose: To get a FORTRAN statement and classify it according to type.

Called by: IPPRO

Processing: The input to this routine consists of the FORTRAN source statements from SYSIN. Each FORTRAN statement is collected into a work area and two scans are performed.

1. Classification scan: the statement is examined one character at a time from

left to right. It is classified as non-arithmetic if:

- The following character sequence indicating the presence of an literal field is found:

  ,nH or /nH or (nH

  where n is a numerical value. Switches CSNU for n and CSHL for H are used.

- The sequence A(.....)x where x is not an equal sign, is encountered. Switch CSNA is used.

- There is a comma outside the parentheses.

- There is an equal sign inside the parentheses.

In the absence of these conditions, the statement is classified as arithmetic.

2. Scan of Keyword Dictionary (KEYDIC): the first four characters of each statement classified as non-arithmetic are compared to a list of FORTRAN keywords. This comparison determines the type of FORTRAN statement. Each source statement is given a statement code from 1 through 42 according to its type.

If an IMPLICIT statement is detected, the routine IMPRO is invoked.

When a FORTRAN statement is recognized as a READ, WRITE, PRINT, or PUNCH statement, the following additional processing takes place:

- The data set and FORMAT reference numbers are collected, if present.
- The LCP creates a table (TFM) containing FORMAT reference numbers, and corresponding data set attributes (PRINT, non-PRINT or both).

In addition, GETCRD builds table TPD and uses it to generate an END statement at the end of each DO loop. Note that a logical IF statement is treated as two separate statements.

The output from GETCRD is the FORTRAN source program on SYSUT1; FORTRAN statements with the same statement code are chained together. When an END card is encountered, this routine returns to the Control Phase.

Routines called: GETWRD, GETTBL, IMPRO, PUTTBL.

Exit: Control Phase

IMPRO Chart 008

Purpose: To convert IMPLICIT statements.

Called by: GETCRD.

Processing: The input to this routine is the IMPLICIT statement in the work area WKBF. The two alphabetic characters used as limits of an implicit type declaration are saved. The original entry in the implicit table (TIM), which contains the standard type of any name beginning with an alphabetic character, is updated to conform with the type given in the IMPLICIT statement. Any modification in length is also entered in this table.

Routines called: GETWRD.

Exit: GETCRD

Chart 002. IPPRO

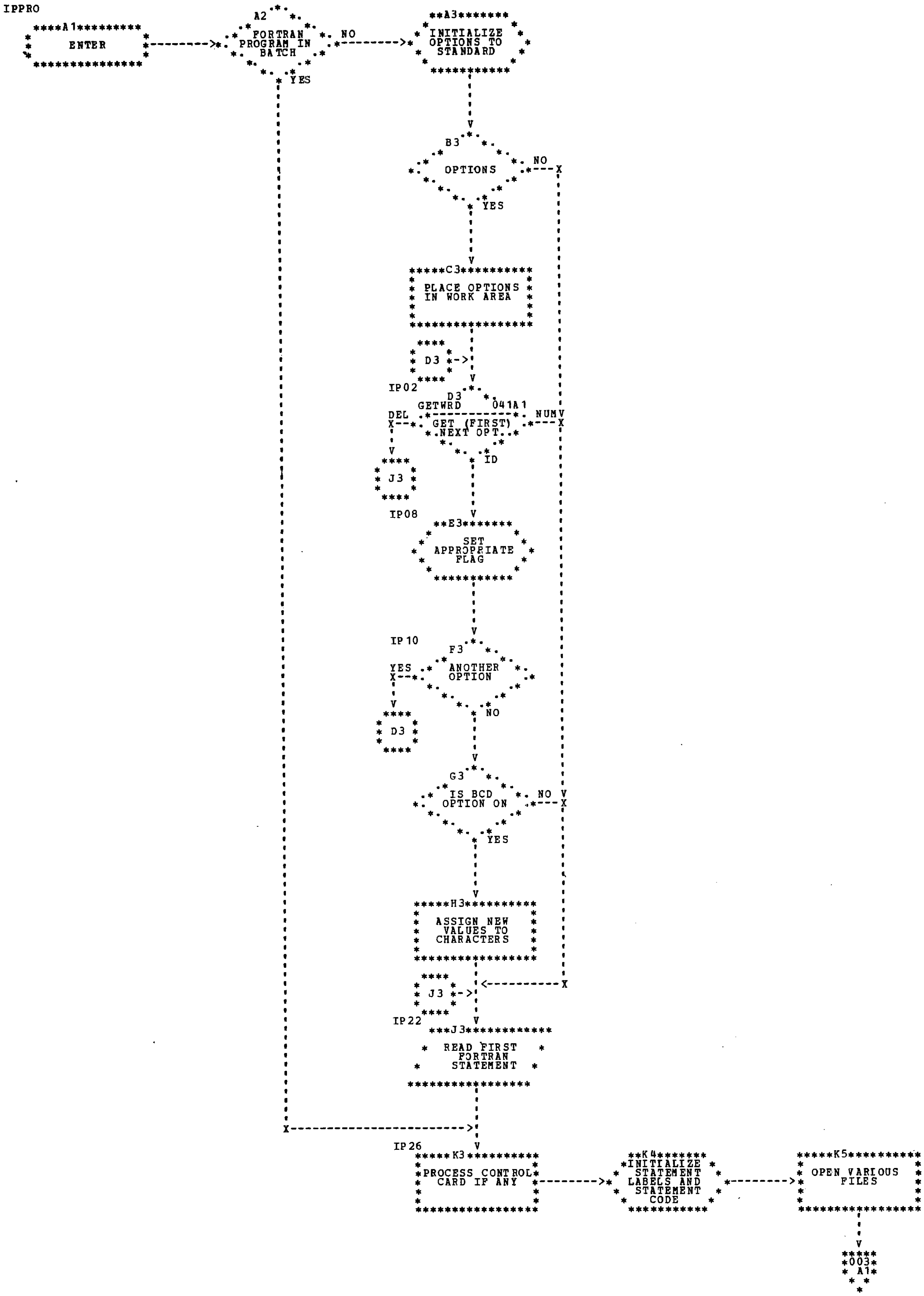


Chart 003. GETCRD (Part 1)

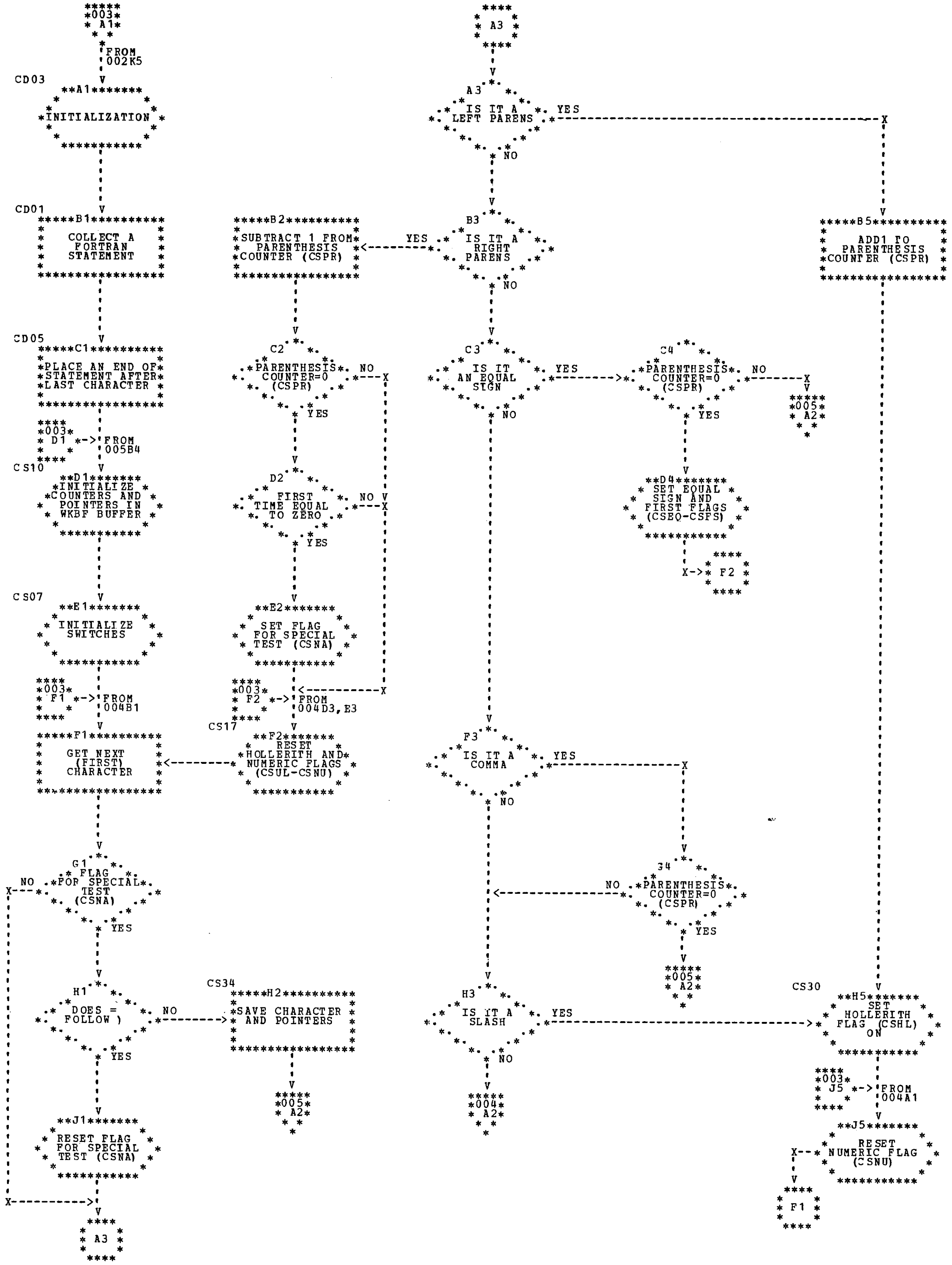

003 A1
FROM 002K5
CD03
A1 INITIALIZATION
CD01
B1 COLLECT A FORTRAN STATEMENT
CD05
C1 PLACE AN END OF STATEMENT AFTER LAST CHARACTER
003 D1
FROM 005B4
CS10
D1 INITIALIZE COUNTERS AND POINTERS IN WKBF BUFFER
CS07
E1 INITIALIZE SWITCHES
003 F1
FROM 004B1
F1 GET NEXT (FIRST) CHARACTER
G1 FLAG FOR SPECIAL TEST (CSNA)
NO
YES
H1 DOES = FOLLOW)
NO
YES
J1 RESET FLAG FOR SPECIAL TEST (CSNA)
A3
B2 SUBTRACT 1 FROM PARENTHESIS COUNTER (CSPR)
C2 PARENTHESIS COUNTER=0 (CSPR)
NO
YES
D2 FIRST TIME EQUAL TO ZERO
NO
YES
E2 SET FLAG FOR SPECIAL TEST (CSNA)
003 F2
FROM 004D3, E3
CS17
F2 RESET HOLLERITH AND NUMERIC FLAGS (CSUL-CSNU)
CS34
H2 SAVE CHARACTER AND POINTERS
005 A2
A3
A3 IS IT A LEFT PARENS
YES
NO
B3 IS IT A RIGHT PARENS
YES
NO
C3 IS IT AN EQUAL SIGN
YES
NO
F3 IS IT A COMMA
NO
H3 IS IT A SLASH
YES
NO
004 A2
C4 PARENTHESIS COUNTER=0 (CSPR)
NO
YES
005 A2
D4 SET EQUAL SIGN AND FIRST FLAGS (CSEQ-CSFS)
F2
G4 PARENTHESIS COUNTER=0 (CSPR)
NO
YES
005 A2
B5 ADD1 TO PARENTHESIS COUNTER (CSPR)
CS30
H5 SET HOLLERITH FLAG (CSHL) ON
003 J5
FROM 004A1
J5 RESET NUMERIC FLAG (CSNU)
F1

Chart 004. GETCRD (Part 2)

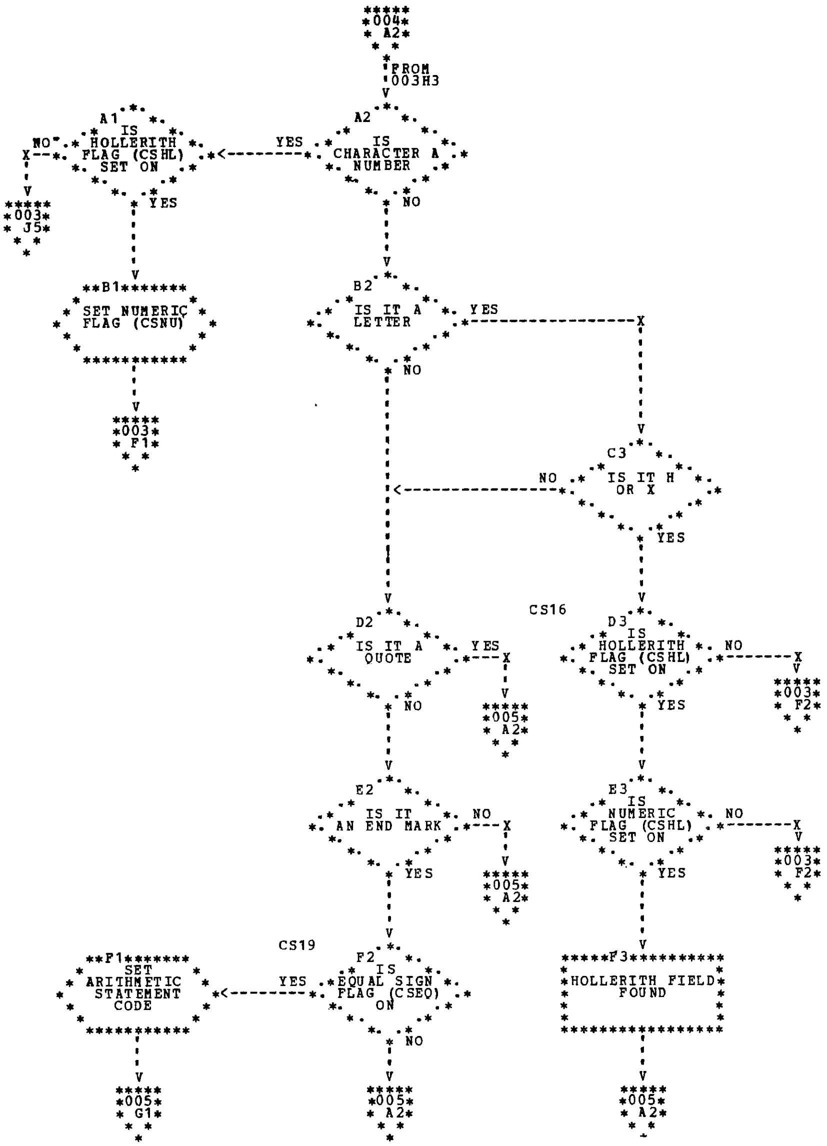


Chart 005. GETCRD (Part 3)

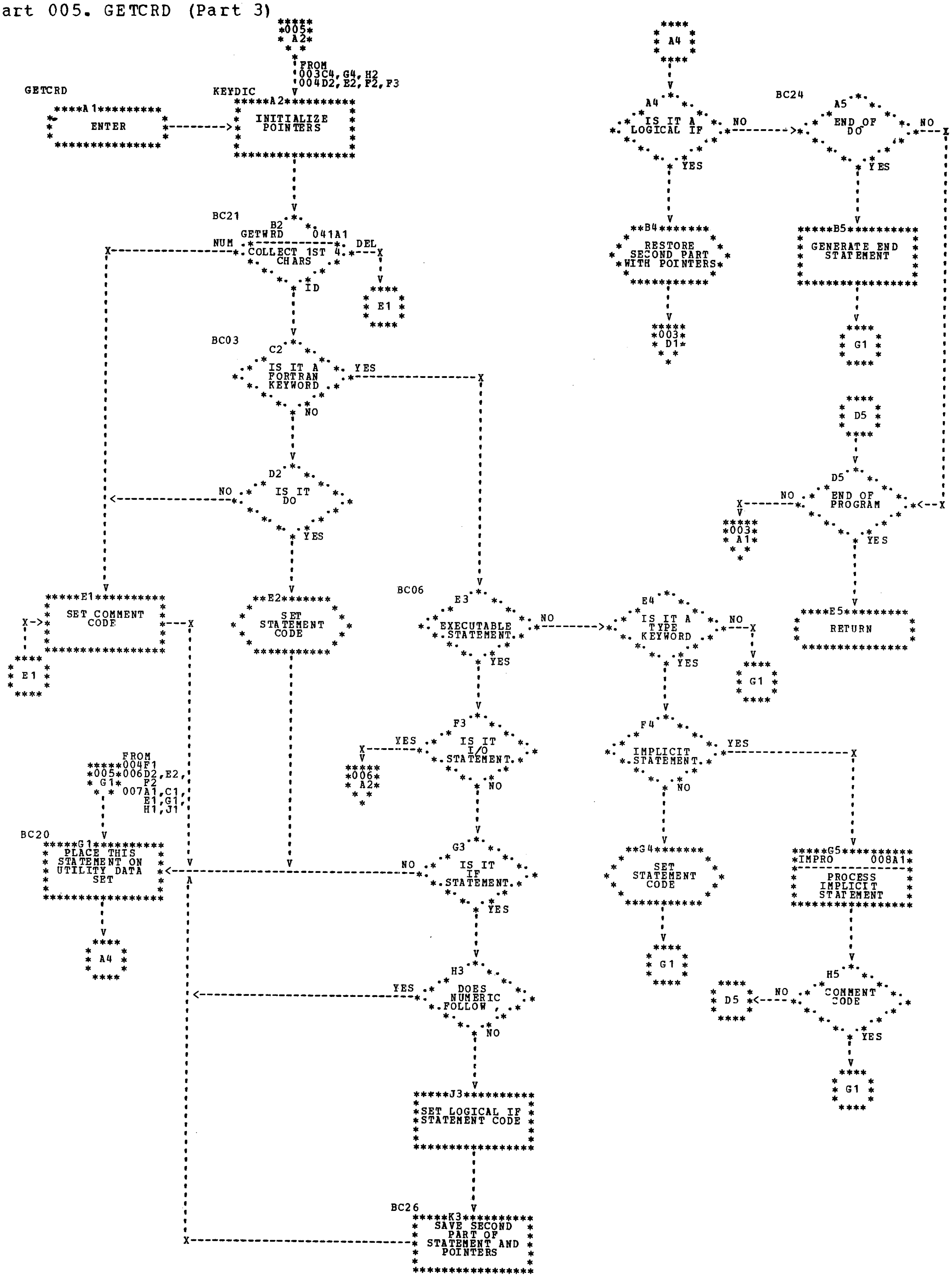


Chart 006. GETCRD (Part 4)

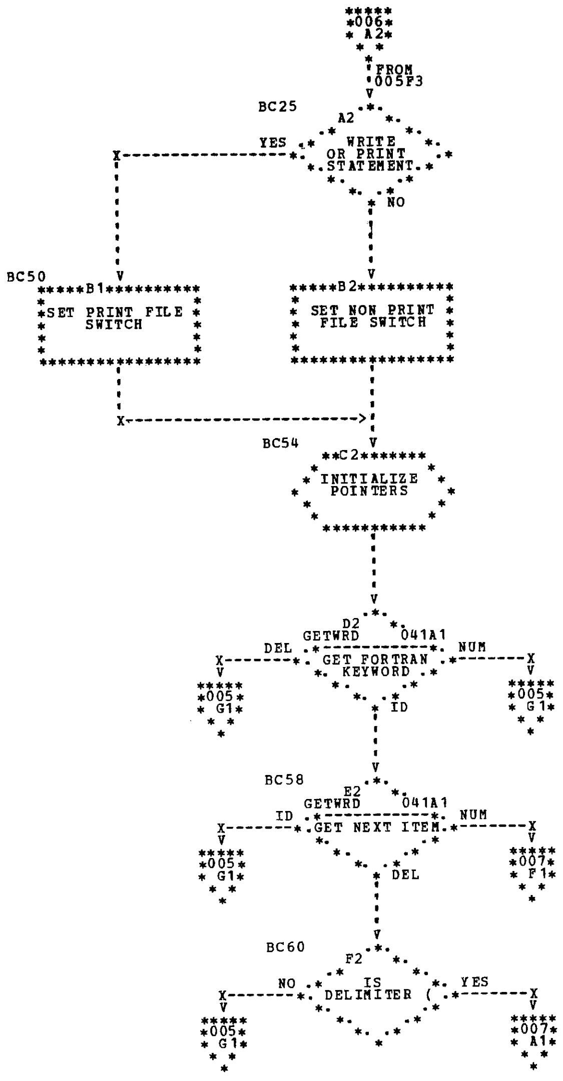


Chart 007. GETCRD (Part 5)

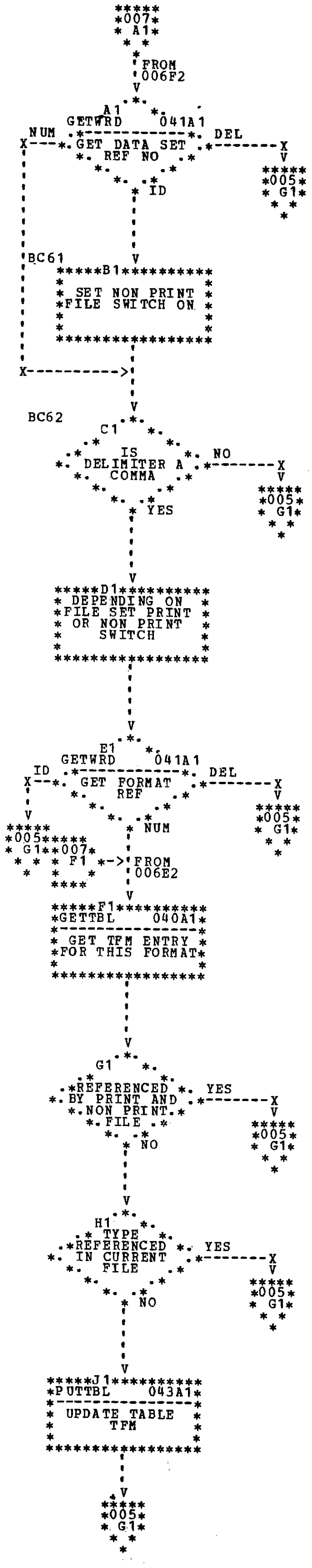
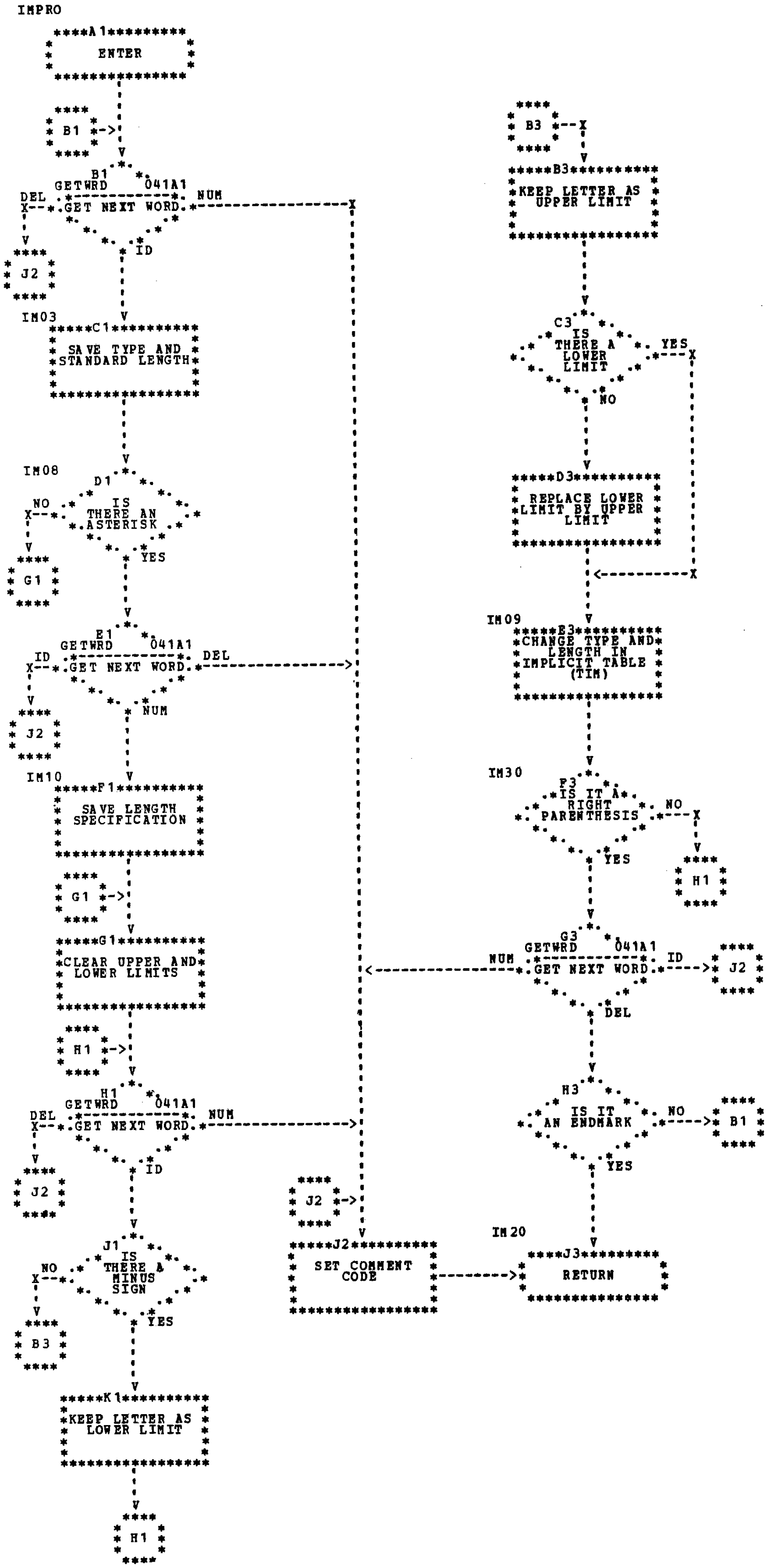


Chart 008. IMPRO Routine



Phase 10 (see Chart 009) converts executable FORTRAN statements and builds tables from the specification statements. Certain information collected during the scan is saved in tables to be processed subsquently by Phase 20 and/or Phase 30.

The input to Phase 10 is the FORTRAN statements placed on SYSUT1 by the Initialization Procedure.

Phase 10 passes control to the appropriate statement processing routine, which processes a chain of statements at a time by attempting to convert as much of the statements as possible into the corresponding PL/I statements. These routines fall into one of two categories:

1. Specification statement processing routines, which enter in the dictionary (TDI) any identifier that appears within a specification statement. The dictionary includes specification information and pointers to tables containing additional information for each entry. The routines included in this category are the following:

   BLPRO
   CMPRO
   DAPRO
   DMPRO
   EDPRO
   EQPRO
   EXPRO
   FNPRO
   NLPRO
   YIPRO

2. Executable statement processing routines, which convert executable statements and place the result on SYSUT1 in external (printable) form. The routines included in this category are the following:

   ALPRO
   ASPRO
   CAPRO
   COPRO
   CTPRO
   DOPRO
   FTPRO
   GTPRO
   IFPRO
   IOPRO
   PSPRO
   RTPRO
   STPRO

In addition, Phase 10 makes use of a number of utility routines which perform simple and frequently recurring functions, such as collecting characters, placing information in tables, etc. These routines are the following:

ARPRO
ENTTDI
ERMS
GETTBL
GETWRD
LTCOL
PUTTBL
SLPAR
SPDIM
SPDTA
SPPRO
SPSUB
XTBPRO

Note: Routines GETTBL and PUTTBL are also used by Initialization Procedure, Phases 20 and 30.

A detailed description of each routine will be found in the individual section allotted to it.

Executable statements that cannot be converted at all, i.e. the statements BACKSPACE and ENDFILE (see Appendix D of the language conversion manual) are placed on SYSUT1, together with an appropriate message.

Comments cards in the FORTRAN program are placed on SYSUT1, in the same way as executable statements. If the sequence */ appears in the text of a comment, it is replaced by the sequence *-.

Any syntactical error in a source statement detected during the scan terminates the conversion of the statement concerned. The statement is treated as a comment and a diagnostic message is issued. No further assumption is made concerning the possible effect of the erroneous statement on the rest of the program.

Source program identifiers are checked to ensure against conflict with PL/I built-in function names or reserved words. In the event of possible conflict, the identifier is entered in the dictionary (TDI) with an appropriate flag. Moreover, if the identifier occurs within the text of an executable statement, a substitution name created by the LCP is automatically generated during conversion.

On completion of Phase 10, control returns to the Control Phase.

SPECIFICATION STATEMENT PROCESSING ROUTINES

This section contains detailed descriptions of the routines used by Phase 10 to process FORTRAN specification statements. In order to simplify reference, the routines appear in alphabetical order.

BLPRO Chart 010

Purpose: To convert BLOCK DATA statements.

Called by: PH10

Processing: The input to this routine is the BLOCK DATA statement in the work area WKBF. A switch (BLSW) is set in this routine to be tested during the processing of the END statement (EDPRO) in order to determine whether the next END statement is part of a BLOCK DATA subprogram. A BEGIN statement is generated on SYSUT1.

Routine called: XTBPRO.

Exit: Calling routine.

CMPRO Chart 011

Purpose: To convert COMMON statements.

Called by: PH10

Processing: The input to this routine is the COMMON statement in the work area WKBF. The common block name is collected and placed in table TBK with a pointer to the list of variables. The list of variables is placed in table TCM and the names of variables are placed in the dictionary.

Routines Called: ENTTDI, ERMS, GETWRD, GETTBL, SPDIM, PUTTBL.

Exit: Calling routine.

DAPRO Chart 012

Purpose: To convert DATA statements.

Called by: PH10

Processing: The input to this routine is the DATA statement in the work area WKBF. The list of elements to be initialized is collected and if necessary, the address of an element, relative to the beginning of the array, is computed. The element and its address are placed in table TDA. The output from this routine is an entry in the dictionary (TDI) for each new name and an entry in table TDA for the names of variables and their literal value.

Routines Called: ENTTDI, ERMS, GETWRD, GETTBL, PUTTBL, SPSUB, SPDTA.

Exit: Calling routine.

DMPRO Chart 013

Purpose: To convert DIMENSION statements.

Called by: PH10

Processing: The input to this routine is the DIMENSION statement in the work area WKBF. The part of the statement following the keyword DIMENSION is scanned and the array name is entered in the dictionary (TDI) if it is not already there. The dimension information is collected and placed in table TDM and a pointer to this table entry is placed in the dictionary. In the case of an array name already in the dictionary, the existing entry is updated.

Routines called: ENTTDI, ERMS, GETWRD, GETTBL, PUTTBL, SPDIM.

Exit: Calling routine.

EDPRO Chart 014

Purpose: To convert END statements.

Called by: PH10

Processing: The input to this routine is the END statement in the work area WKBF. The END statement is converted and placed on SYSUT1.

Routine called: XTBPRO.

Exit: Calling routine.

EQPRO Chart 015

Purpose: To convert EQUIVALENCE statements.

Called by: PH10

Processing: The input to this routine is the EQUIVALENCE statement in the work area WKBF. The elements of the statement are collected and the equivalenced variables are placed in the equivalence table TEQ and in the dictionary TDI. If necessary, the address of an element relative to the beginning of an array is computed. The element and its address are placed in table TEQ.

Routines called: ENTTDI, ERMS, GETWRD, GETTBL, PUTTBL, SPSUB.

Exit: Calling routine.

EXPRO Chart 016

Purpose: To convert EXTERNAL statements.

Called by: PH10

Processing: The input to this routine is the EXTERNAL statement in the work area WKBF. The part of the statement following the keyword EXTERNAL is scanned. The subprogram names are checked for validity and added to the dictionary (TDI) with flags indicating their class, type and usage.

Routines called: ENTTDI, ERMS, GETWRD, PUTTBL.

Exit: Calling routine.

FNPRO Chart 017

Purpose: To convert FUNCTION, SUBROUTINE, and ENTRY statements.

Called by: PH10

Entries: There are three entry points to this routine: FNPRO, SBPRO, and ETPRO. The input, at entry point FNPRO, is the FUNCTION statement in the work area WKBF; at entry point SBPRO it is the SUBROUTINE statement; at entry point ETPRO it is the ENTRY statement.

Processing: The procedure name is placed on SYSUT1 with the keyword PROCEDURE (or ENTRY in the case of an ENTRY statement), followed by the formal parameters (unconverted) and the nonstandard return specifications (converted). A name followed by a number is assigned to each nonstandard return. This number represents the sequential number of each return within the parameter list. An additional formal parameter is created for a function subprogram and this name is entered in the dictionary (TDI). The converted statement is placed on SYSUT1.

Routines called: ENTTDI, ERMS, GETWRD, PUTTBL, SLPAR, XTBPRO.

Exit: Calling routine.

NLPRO Chart 018

Purpose: To convert NAMELIST statements.

Called by: PH10

Processing: The input to this routine is the NAMELIST statement in the work area WKBF. The namelist name is collected and placed in table TNL with a pointer to the list of variables. This list is placed in table TNV and the variable names are placed in the dictionary (TDI).

Routines called: ENTTDI, ERMS, GETTBL, GETWRD, PUTTBL.

Exit: Calling routine.

YIPRO Chart 019

Purpose: To process explicit type statements.

Called by: PH10

Entries: This routine has five entry points; the input to it consists of the following type statements: INTEGER (entry point YIPRO), REAL (entry point YRPRO), COMPLEX (entry point YCPRO), LOGICAL (entry point YLPRO), DOUBLE PRECISION (entry point YDPRO).

Processing: The type and length specifications of the input statement are saved. The length specification, the variable names, and the initial values, if any, are then scanned. The variable names are entered in the dictionary (TDI), with specification information and pointers to table TDM, and the initial values are processed by the routine SPDTA.

Routines called: ENTTDI, ERMS, GETWRD, PUTTBL, SPDTA.

Exit: Calling routine.

EXECUTABLE STATEMENT PROCESSING ROUTINES

This section contains detailed descriptions of the routines used by Phase 10 to process executable statements in the FORTRAN source program. The organization of this section is the same as that of the preceding section.

ALPRO Chart 020

Purpose: To convert assignment statements.
Called by: PH10

Processing: The input to this routine is the assignment statement in the work area WKBF. The part of the statement located to the left of an equal sign is tested to determine if it is a variable, an array or a statement function name. Then:

1. a) If it is a variable, the name is placed in the dictionary (TDI).

   b) If it is an array, the subscript routine (SPPRO) is called.

2. If the name appearing to the left of an equal sign is followed by elements between parentheses and has not been given any dimension, it is a statement function name. A PL/I procedure statement is generated and the parameter list is processed. The pointers to parameter entries in the dictionary (TDI) are saved in an argument table. These pointers are placed on SYSUT1 after the PL/I PROCEDURE statement; they are separated by commas and followed by a semi-colon. The LCP , then, generates the following statements:

   ```
   RETURN (<expression>);

   END;
   ```

The routine ARPRO is called to process the expression located to the right of the equal sign.

The converted statement is placed on SYSUT1.

Routines called: ARPRO, ENTTDI, ERMS, GETWRD, PUTTBL, SPPRO, XTBPRO.

Exit: Calling routine.

ASPRO Chart 021

Purpose: To convert ASSIGN statements.

Called by: PH10

Processing: The input to this routine is the ASSIGN statement in the work area WKBF. The statement number following the keyword ASSIGN is collected and saved. The word TO is bypassed and the variable is placed in the dictionary (TDI). The converted statement is placed on SYSUT1.

Routines called: ENTTDI, ERMS, GETWRD, PUTTBL, XTBPRO.

Exit: Calling routine.

CAPRO Chart 022

Purpose: To convert CALL statements.

Called by: PH10

Processing: The input to this routine is the CALL statement in the work area WKBF. The FORTRAN CALL statement is converted into the PL/I CALL statement. The latter is placed on SYSUT1 followed by the subprogram name and its arguments, which are processed in the routine ARPRO.

Routines called: ARPRO, ERMS, ENTTDI, GETWRD, XTBPRO, LABPRO, PUTTBL.

Exit: Calling routine.

COPRO Chart 023

Purpose: To process comments cards.

Called by: PH10

Processing: The input to this routine is the comments card. The comments card is copied onto SYSUT1, the delimiters used being: /* for the beginning and */ for the end.

The */ sequence, where it occurs within the text of a comment, is replaced by the *- sequence.

Routine called: LTCOL.

Exit: Calling routine.

CTPRO — Chart 024

Purpose: To convert CONTINUE statements.

Called by: PH10

Processing: The input to this routine is the CONTINUE statement in the work area WKBF. If a label is associated with the CONTINUE statement, this statement is converted into a semi-colon. The label and the semi-colon are placed on SYSUT1. If no label is attached to the statement, this statement is ignored.

Routines called: ERMS, GETWRD, XTBPRO, LABPRO.

Exit Calling routine.

DOPRO — Chart 025

Purpose: to convert DO statements.

Called by: PH10

Processing: The input to this routine is the DO statement in the work area WKBF. The statement number specifying the end of the DO loop is ignored. The DO index is collected and placed on SYSUT1. The DO parameters (PAR1, PAR2, PAR3) are collected and saved. If one of the two limits is variable, switch DOSW is set on. Then, one of the following three PL/I statements is generated:

1. If one of the limits is variable:

   DO index = PAR1 TO MAX(PAR1, PAR2) BY PAR3;

2. If the limits are constant and PAR1 is less than PAR2:

   DO index = PAR1 TO PAR2 BY PAR3;

3. If the limits are constant and PAR1 is greater than or equal to PAR2:

   DO index = PAR1;

The output from this routine is the converted DO statement on SYSUT1.

Routines called: ERMS, GETWRD, PUTTBL, XTBPRO, LABPRO, ENTTDI.

Exit: Calling routine.

FTPRO — Chart 026

Purpose: To convert FORMAT statements.

Called by: PH10

Processing: The input to this routine is the FORTRAN FORMAT statement in the work area WKBF. The FORTRAN format codes are translated as follows:

A to A

D E

E E

F F with call to LBLNK if BLKZR option is on (see Appendix B)

G G with a warming message

I F with call to LBLNK if BLKZR option is on (see Appendix B)

L B with a warning message

Z Z with a warning message.

/ SKIP(n) where n is the number of consecutive slashes

T COLUMN( )

X X

A string of characters appearing with H and quotes codes are replaced by an A-format code with the number of characters contained in the string.

The field count and the decimal count are enclosed in parentheses and the period is replaced by a comma.

COLUMN(1) is generated at the beginning of each PL/I FORMAT statement, to force the end of the current record.

The carriage control character read at the beginning of the FORMAT statement and after a slash is converted into the appropriate PL/I statement (PAGE, SKIP or COLUMN).

The contents of the FORMAT statement is saved in a packed form, "packed FORMAT", in the following way:

- n (format-code.

  The left parenthesis is given an internal code '*' and saved with the associated iteration factor.

- Character-strings ('-' or nH)

The string is collected and placed in table TDT.

The pointer to the entry in table TDT is then associated with an LCP-created variable which is saved in table TDU.

A reference to this entry in table TDU is saved in packed format following the internal code P.

- A format-code

  The total number of successive A-format codes not separated by any other code, by a character string, or by a left parenthesis, is placed in packed FORMAT after the internal code A.

- D, E, F, G, I, L, Z format-codes

  The total number of successive items associated with these codes and not separated by any of the previous ones is saved in packed FORMAT with the internal code N.

- Right parenthesis

  A right parenthesis is used as a packed FORMAT internal code to indicate the end of an iteration group and of packed FORMAT.

  Packed FORMAT information is placed in the table TDT and the pointer to the corresponding entry is saved with the entry in table TFM.

  If the FORMAT statement does not contain any A-format code or character string, the packed FORMAT is not saved.

  The converted PL/I FORMAT statement is placed on SYSUT1. The tables TFM and TDU are updated (see IOPRO for use of these tables).

Figure 2 illustrates tables TFM, TDT, and TDU.

Routines called: ERMS, GETWRD, LTCOL, XTBPRO, GETTBL, PUTTBL, LABPRO.

Exit: Calling routine.

GTPRO Chart 027

Purpose: To convert GO TO statements.

Called by: PH10

Processing: The input to this routine is the GO TO statement in the work area WKBF. This routine first tests what type of GO TO statement is concerned. Then,

1. In the case of an unconditional GO TO statement, the PL/I statement "GO TO EXTLABn" is placed on SYSUT1.

2. In the case of an assigned GO TO statement, the PL/I statement "GO TO variable" is placed on SYSUT1. An entry for the variable is made in the dictionary (TDI).

3. In the case of a computed GO TO statement, the following PL/I statement is placed on SYSUT1:

   IF (<index><=<number of parameters> &<index>>0) THEN GO TO BRANCHi (<index>)

   The index is placed in the dictionary. An entry in the dictionary is also created for the sequence number associated with the label BRANCH, with a pointer to table TDT, which contains the list of the GO TO parameters.

The output from this routine is an updated entry in the dictionary (TDI), an entry in table TDT in the case of a computed GO TO statement, and a converted PL/I statement on SYSUT1.

Routines called: ENTTDI, ERMS, GETWRD, PUTTBL, XTBPRO, LABPRO, BRNPRO.

Exit: Calling routine.

IFPRO Chart 028

Purpose: To convert IF statements.

Called by: PH10

Processing: The input to this routine is the IF statement in the work area WKBF. The FORTRAN IF statement is converted into a PL/I IF statement and placed on SYSUT1, followed by the arithmetic expression processed by the routine ARPRO. In addition, if the IF is an arithmetic one, the three branch labels are analyzed and converted, taking into account the relationship of the three branch labels and the label of the statement immediately following. The output is the PL/I statement on SYSUT1.

Routines called: ARPRO, ERMS, GETWRD, XTBPRO, LABPRO, BRNPRO.

Exit: Calling routine.

Purpose: To convert input/output statements.

Called by: PH10

Processing: The input to this routine is input/output statement in the work area WKBF.

The first part of the converted statement, for READ or WRITE, respectively, is:

PUT FILE (FT nF01) or GET FILE (FTnF01)

where n is the data-set reference number of the FORTRAN statement.

This part of the converted statement may be preceded by ON conditions, if necessary.

If the ERROR option is present, the routine generates the statement ON TRANSMIT.

If the END option is present, the routine generates the statement ON ENDFILE.

If the NAMELIST option is present, the routine generates DATA followed by the list of variables. If this option is not present, the routine generates the EDIT option followed by the data list.

The output from this routine consists of the appropriate PL/I input/output statements on SYSUT1. The data set and FORMAT reference numbers are saved. The corresponding table TFM and packed FORMAT entries are collected.

Each element of the I/O list is associated with a packed FORMAT element.

- N code

  An N code indicates the number of variables in the I/O list which are placed into the PL/I converted statement.

- A code

  An A code indicates the number of variables in the I/O list which are placed into the converted PL/I statement. The DICTIONARY entries of these variables are flagged as type CHARACTER.

- P code

  When a P code is encountered, the corresponding entry is table TDU is collected. The LCP-created variable is inserted in the I/O list.

  If the end of packed FORMAT information is reached before the end of the I/O list, the processing described above goes on from the last * which corresponds to the last pair of parentheses of level-1.

  This processing stops when the end of the I/O list is reached or when a DO loop in the I/O list is encountered.

- Empty I/O list

  If there is no I/O list, the packed FORMAT information is checked to determine whether an LCP-created variable or only a carriage control option (SKIP, PAGE) need be generated.

  If there is no packed FORMAT information associated with a FORMAT reference, the I/O statement is converted directly into PL/I.

  No conversion occurs if data set or FORMAT references are not integer constants, or if there is no FORMAT reference.

Figure 2 illustrates tables TFM, TDT, and TDU.

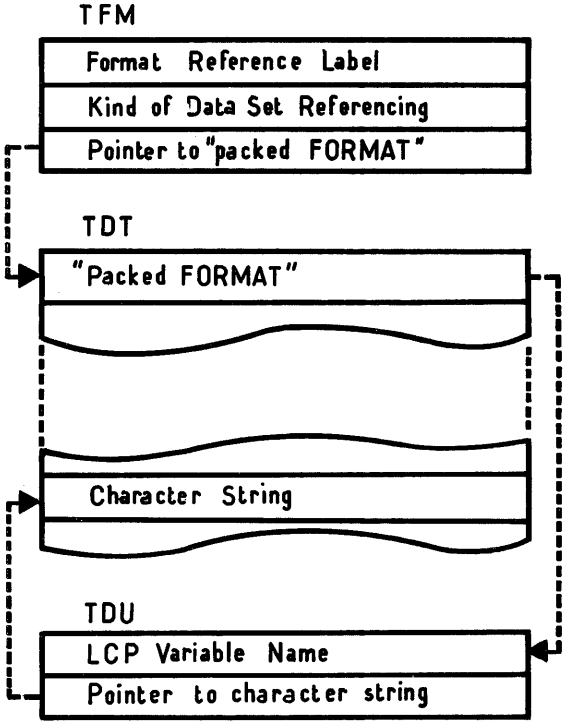

TFM
Format Reference Label
Kind of Data Set Referencing
Pointer to "packed FORMAT"
TDT
"Packed FORMAT"
Character String
TDU
LCP Variable Name
Pointer to character string

Figure 2. Tables TFM, TDT, TDU

Routines called: ENTTDI, ERMS, GETWRD, PUTTBL, XTBPRO, LABPRO, BRNPRO, GETTBL.

Exit: Calling routine.

PSPRO ________________________________ Chart 031

Purpose: To convert PAUSE statements.

Called by: PH10

Processing: The input to this routine is the PAUSE statement in the work area WKBF. The field following the keyword PAUSE is placed in the converted statement on SYSUT1. This field can be a blank, a number or a character string.

Routines called: ERMS, GETWRD, LTCOL, XTBPRO, LABPRO.

Exit: Calling routine.

RTPRO ________________________________ Chart 032

Purpose: To convert RETURN statements.

Called by: PH10

Processing: The input to this routine is the RETURN statement in the work area WKBF. The data following the RETURN statement is collected. If this field is blank, the RETURN statement is placed on SYSUT1. If the field is a constant or a variable, a transfer is made to the return parameter created or to an array of returns. The names created, together with a reference to their initial values, are entered in the dictionary (TDI).

Routines called: ENTTDI, GETWRD, PUTTBL, XTBPRO, LABPRO.

Exit: Calling routine.

STPRO ________________________________ Chart 033

PURPOSE: To convert STOP statements.

Called by: PH10

Processing: The input to this routine is the STOP statement in the work area WKBF. The field following the keyword STOP is scanned for significant data. If a character string is present, it is placed with the converted statement on SYSUT1.

Routines called: ERMS, GETWRD, XTBPRO, LABPRO

Exit: Calling routine.

UTILITY ROUTINES

This section contains detailed descriptions of the utility routines used by Phase 10. The organization is the same as that of the two preceding sections.

ARPRO ______________ Charts 034, 035, 036, 037

Purpose: To convert arithmetic or logical expressions.

Called by: ALPRO, CAPRO, IFPRO.

Processing: The input to this routine is the arithmetic or the logical expression in the work area WKBF. The variable names are placed in the dictionary (TDI) and the elements of the arithmetic or logical expression are placed on SYSUT1.

The expression is scanned twice in the FORTRAN statement. For a procedure CALL statement or a function call, all the arguments are scanned by each scan.

First Scan: (Charts 034, 035, 036) During the first scan, a "pushdown table" is used. Each entry in this table contains: the pointers to the beginning of a sub-expression, the type of the sub-expression (initialized integer), and the previous delimiter (operator or other).

An entry is added to the "pushdown table" when one of the following is encountered:

- a left parenthesis
- a function name
- a slash
- a NOT operator

These items constitute the beginning of a subexpression.

An entry is withdrawn from the "pushdown table" when one of the following is encountered:

- a right parenthesis
- a comma in a list of arguments
- a logical or relational operator
- an end of expression

These items constitute the end of a sub-expression.

During this scan, the entries in the pushdown table are updated according to the type of variable and the priority of the operator.

If integer division is found, pointers that indicate where to make an insert are taken from the pushdown table and are saved in an internal area (Insert Table).

The Insert Table is also used for inserts other than those for integer division. It contains pointers to the source expression that indicate where to make the insertion and the kind of insertion, as follows:

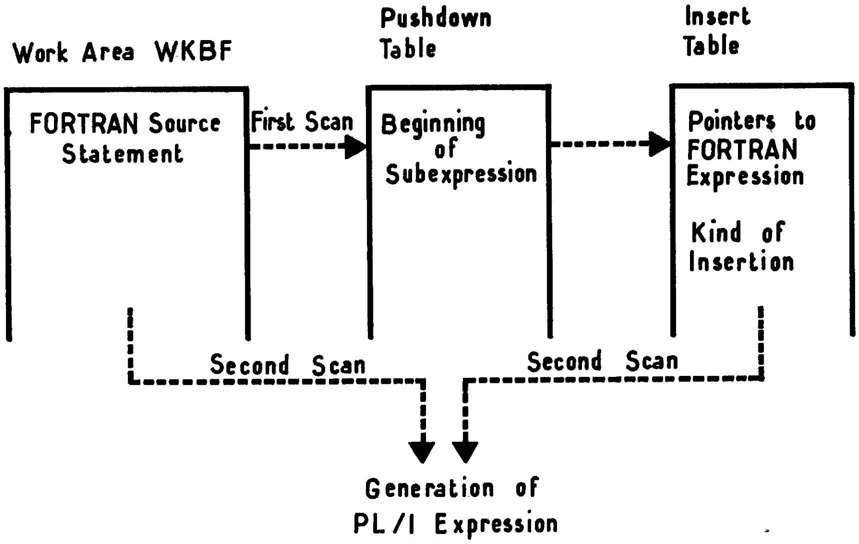
- TRUNC function for integer division.
- BINARY function for integer constants used as arguments.
- The sign between the two parts of a complex constant.
- The letter 'I' after the imaginary part of a complex constant.
- Right parenthesis at the end of an expression under the scope of a NOT operator or corresponding to the end of an expression used as an argument of the TRUNC or BINARY PL/I function.

Second Scan: (Chart 037) When the end of an expression has been reached, the source expression is scanned again. While the scan continues, the expression is converted using information saved in the Insert Table. When the scan pointers equal the pointers in the Insert Table, the insertion is made in the translated expression.

Figure 3 describes the flow of ARPRO processing.

Routines called: ENTTDI, ERMS, GETWRD, PUTTBL, SPPRO, XTBPRO, LTCOL.

Exit: Calling routine.



Fiqure 3. ARPRO Processing

ENTTDI Chart 038

Purpose: To extract the characteristics of a variable from the dictionary (TDI) according to the variable name.

Called by: All statement processing routines except BLPRO, CTPRO, EDPRO, PSPRO, STPRO.

Processing: The input to this routine is the name of a variable. The dictionary is scanned to determine whether it contains the variable. If it does, the return label is used; if not, the next sequential instruction is executed. Table entry pointer TBLCW is then updated.

Routine called: None.

Exit: Calling routine.

ERMS Chart 039

Purpose: To assign an error number to an erroneous statements.

Called by: Any routine in Phase 10.

Processing: The input to this routine is any FORTRAN statement in the work area WKBF. The number assigned to a particular error is placed on SYSUT1.

Routine called: LTCOL, XTBPRO.

Exit: Calling routine.

GETTBL Chart 040

Purpose: To get a table from SYSUT1.

Called by: CMPRO, DAPRO, DMPRO, EQPRO, FTPRO, NLPRO, IOPRO, PH30, PH20, Initialization Procedure.

Processing: The input to this routine is the table whose name appears as the argument of the statement CALL GETTBL. For example, in the statement CALL GETTBL (TAB1), TAB1 is the name of the table from which the information is fetched. The table entry pointer TBLCW contains the address of the table entry. The elements of the entry are fetched and placed in the entry format. In the case of a common block name, a namelist name, and a label format, a search is made by comparing these names with the identifiers placed in tables TBK, TNL, and TFM respectively. On output, the entry of the referenced table is placed in a buffer.

Routines called: None.

Exit: Calling routine.

GETWRD Chart 041

The characters collected are placed, with their type and length, in an area called WORD.

Purpose: To get a constant, a name or a delimiter.

Called by: Any routine in Phase 10, Initialization Procedure.

Processing: The input to this routine is the source statement in the work area WKBF. This routine divides the source statement into groups of characters classified as numeric, alphabetic or delimiter depending upon the first character collected.

The characters collected are placed, with their type and length, in an area called WORD. If a delimiter is collected, XTDLM contains the delimiter itself. If an alphabetic group is collected, XTDLM contains the character that follows the group (which is a delimiter). If a numeric group is collected, XTDLM contains the character that follows the group (which is either a delimiter or an alphabetic character).

Routine called: None.

Exit: Return to one of three labels, depending on whether a numeric string, an alphameric string, or a delimiter was collected.

LTCOL Chart 042

Purpose: To collect the alphameric data in FORTRAN statements.

Called by: ARPRO, COPRO, FTPRO, PSPRO, SPDTA.

Processing: The input to this routine consists of the alphameric literals in the work area WKBF. The routine collects the literals and places them on SYSUT1 or in table TDT. The character count is placed on SYSUT1 before the character-string, if necessary.

Routines called: XTBPRO.

Exit: Calling routine.

PUTTBL Chart 043

Purpose: To copy tables on SYSUT1

Called by: ALPRO, ARPRO, ASPRO, CMPRO, DAPRO, DMPRO, DOPRO, EQPRO, EXPRO, FNPRO, IOPRO, NLPRO, RTPRO, SPDIM, SPPRO, YIPRO in Phase 10; DTPRO, EVPRO in Phase 20, Initialization Procedure, Phase 30.

Processing: The input to this routine is the name of the table entry. In the statement: CALL PUTTBL (TAB1), TAB1 is the name of a table where the entry will be placed. The entry is placed in a buffer. All the entries of the same table are chained together using a pointer in each entry. The table entry pointer TBLCW is analyzed to see if its value is equal to zero: if it is, a new entry is placed in the table; if it is not, the existing entry is updated.

Routines called: None.

Exit: Calling routine.

SLPAR Chart 044

Purpose: To collect the parameter list of an entry point

Called by: FNPRO.

Processing: The input to this routine is the parameter list in the work area WKBF. Each parameter is placed on SYSUT1.

Routines called: ERMS, GETWRD, XTBPRO ENT-TDI, PUTTBL.

Exit: Calling routine.

SPDIM Chart 045

Purpose: To process the list of dimensions of a declarative statement.

Called by: CMPRO, DMPRO, YIPRO.

Processing: The input to this routine is the specification statement in the work area WKBF. An entry in the dimension table (TDM) is made for each dimension collected. The pointer to this entry is placed in the overflow table (TOV).

Routines called: ENTTDI, ERMS, GETTBL, GETWRD, PUTTBL.

Exit: Calling routine.

SPDTA Chart 046

Purpose: To collect DATA literals and place them in table TDT.

Called by: DAPRO, YIPRO.

Processing: The input to this routine consists of the DATA and TYPE statements in WKBF. Literals are collected and rearranged to be entered in table TDT.

Routines called: LTCOL, PUTTBL.

Exit: Calling routine.

SPPRO Chart 047

Purpose: To process subscripts.

Called by: ALPRO, ARPRO, (first, second scan) IOPRO.

Processing: The input to this routine is the FORTRAN statement in work area WKBF. The pointers to the commas and to the last right parenthesis of the subscript are collected. Using the pointers, the subscript parameters are generated in reverse order and placed on SYSUT1. The pointers are then positioned on the last right parenthesis.

Routines called: ENTTDI, GETWRD, PUTTBL, XTBPRO.

Exit: Calling routine.

SPSUB Chart 048

Purpose: To compute the position of an element within a block, relative to the first element of the block.

Called by: DAPRO, EQPRO.

Processing: The input to this routine is the FORTRAN statement in the work area WKBF. Given DIMENSION A (a, b... n*), where A has subscripts (i, j... r*), the position of the element is computed using the following formula:

i+(j-1)a+(k-1)ab+...(r*-1)ab...n*

The result is passed to the calling routine.

Routines called: ERMS, GETTBL, GETWRD.

Exit: Calling routine.

XTBPRO Chart 049

Purpose: To place converted executable statements on SYSUT1.

Called by: LTCOL and all statement processing routines except CMPRO, DAPRO, DMPRO, EQPRO, EXPRO, NLPRO, YIPRO.

Entries: This routine has three entry points: XTBPRO, BRNPRO, and LABPRO. At entry points XTBPRO and BRNPRO the input is a character-string in the area WORD; at entry point LABPRO it is the label in the area LCCRT.

Processing: If the input character-string conflicts with a PL/I built-in function name, it is modified (see section "Form of LCP Substitution Names" in the language conversion manual). Each time a line in the buffer is full, this line is placed on SYSUT1, and the next line is initialized.

At entry point BRNPRO, switch BRSW is set off.

At entry point LABPRO, switch BRSW is set on. Leading zeros and all blanks are skipped.

A delimiter is placed on SYSUT1. This delimiter is a semicolon for BRNPRO and a colon for LABPRO.

Routines called: None.

Exit: Calling routine.

Chart 009. Overall Logic of Phase 10

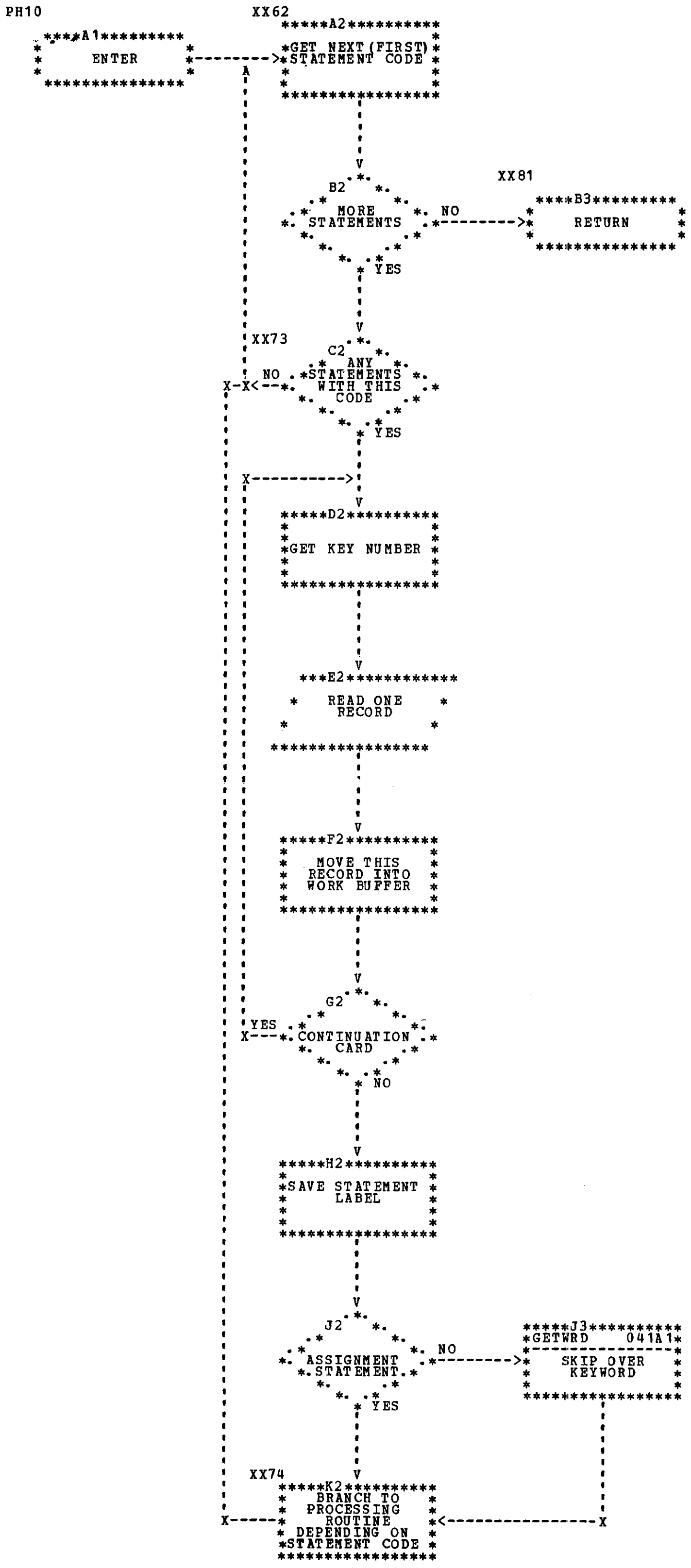


Chart 010. BLPRO Routine

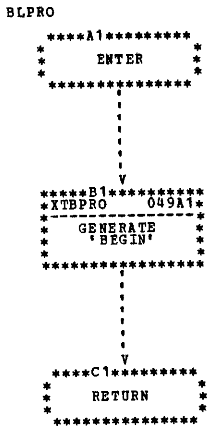


Chart 011. CMPRO Routine

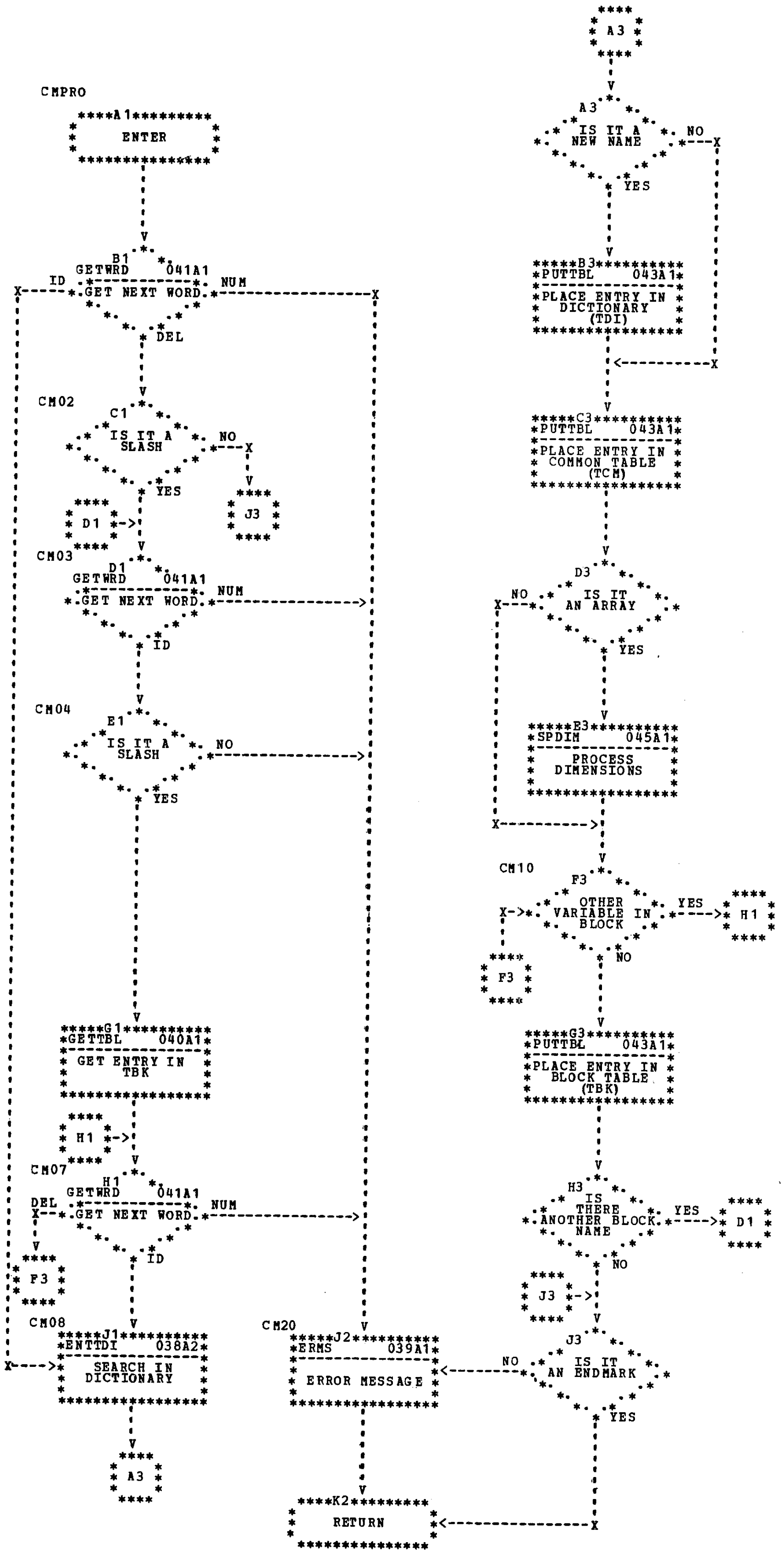


Chart 012. DAPRO Routine

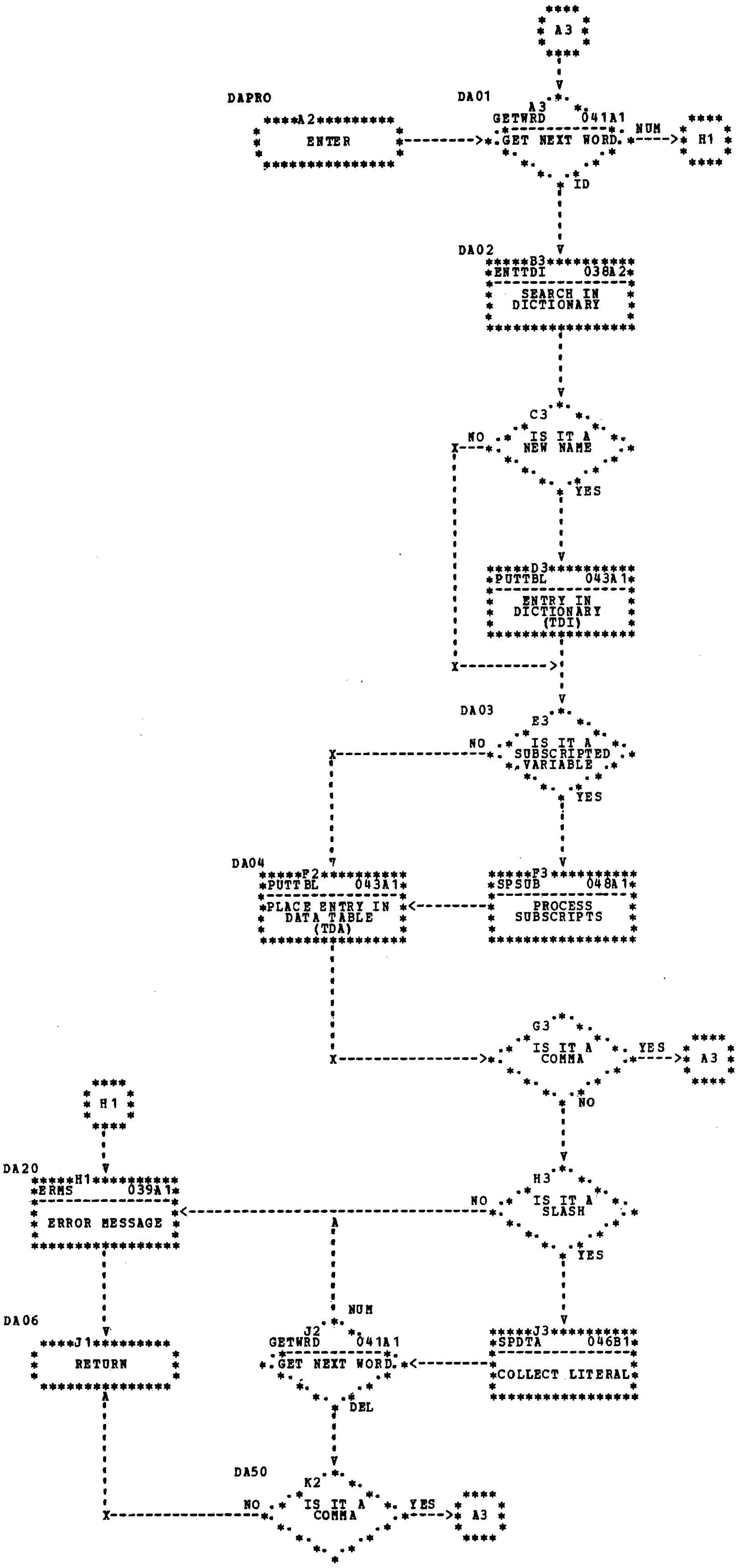


Chart 013. DMPRO Routine

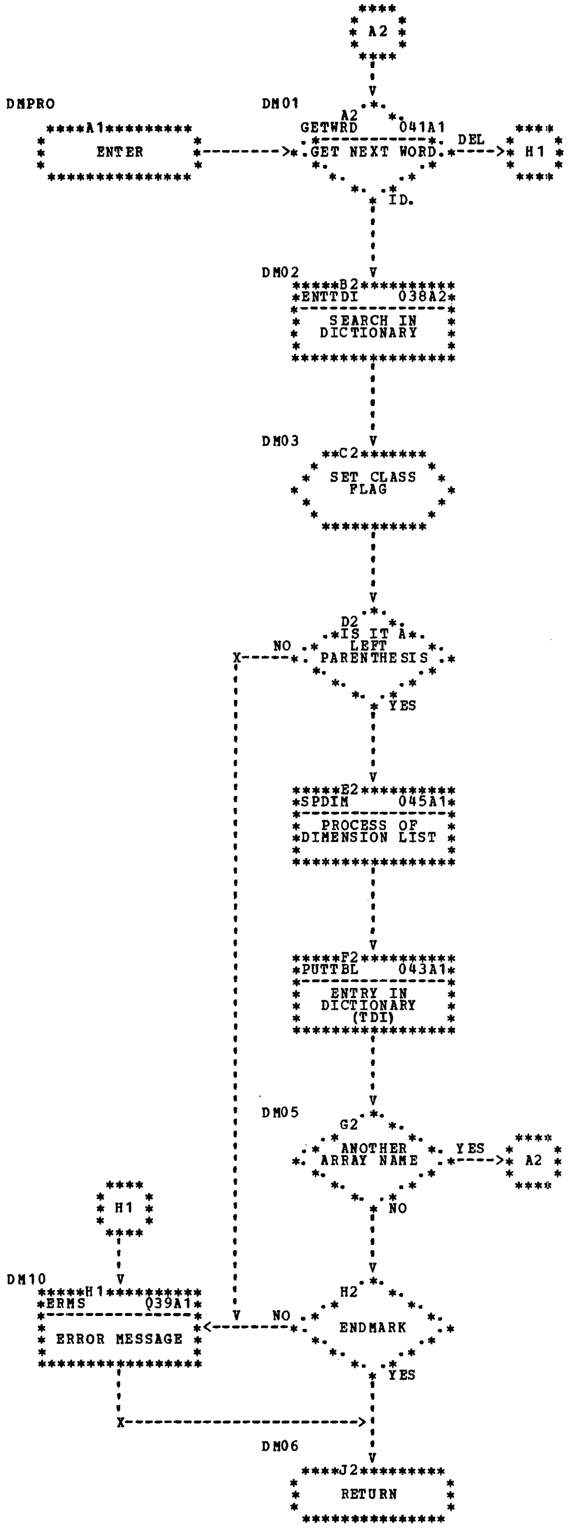


Chart 014. EDPRO Routine

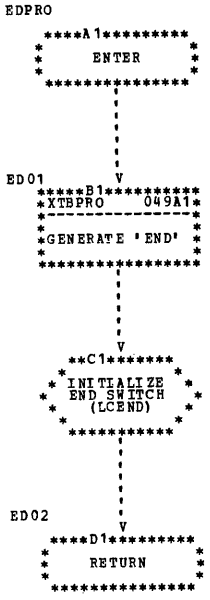


Chart 015. EQPRO Routine

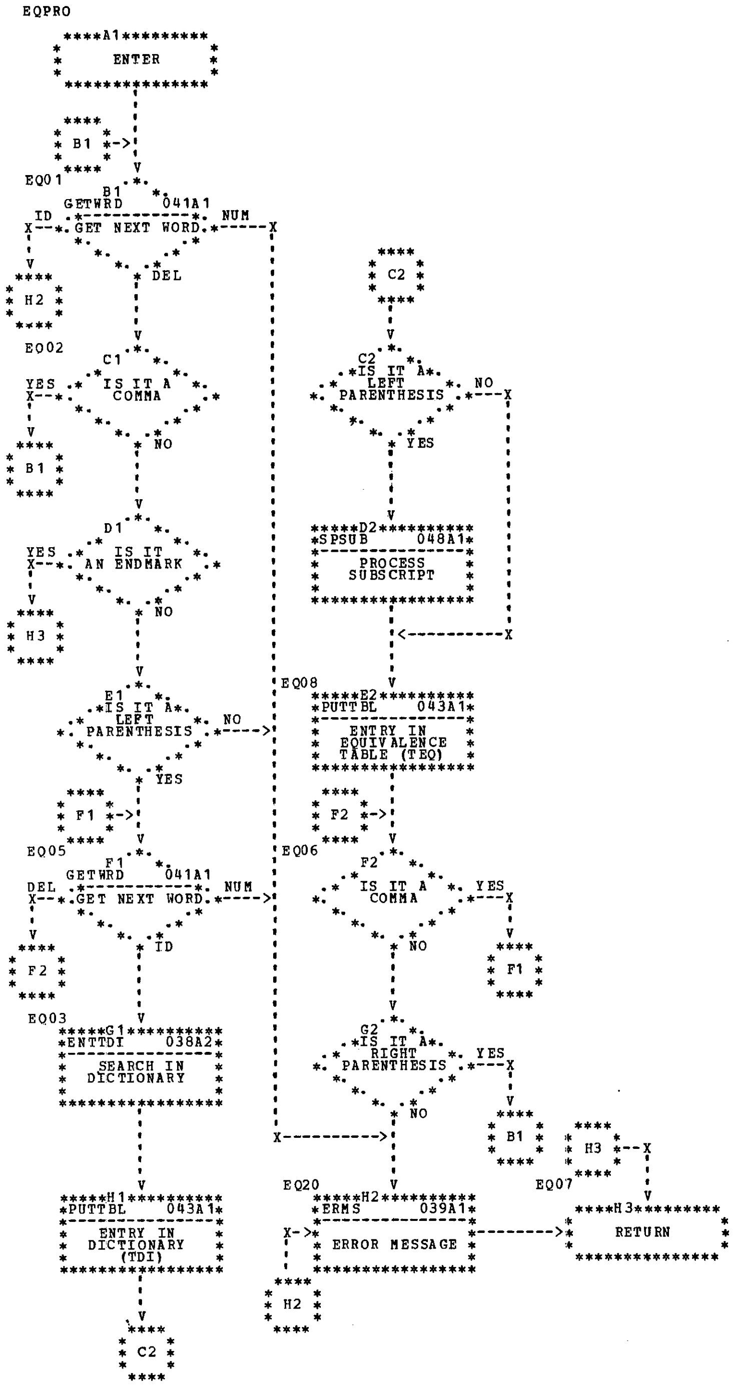


Chart 016. EXPRO Routine

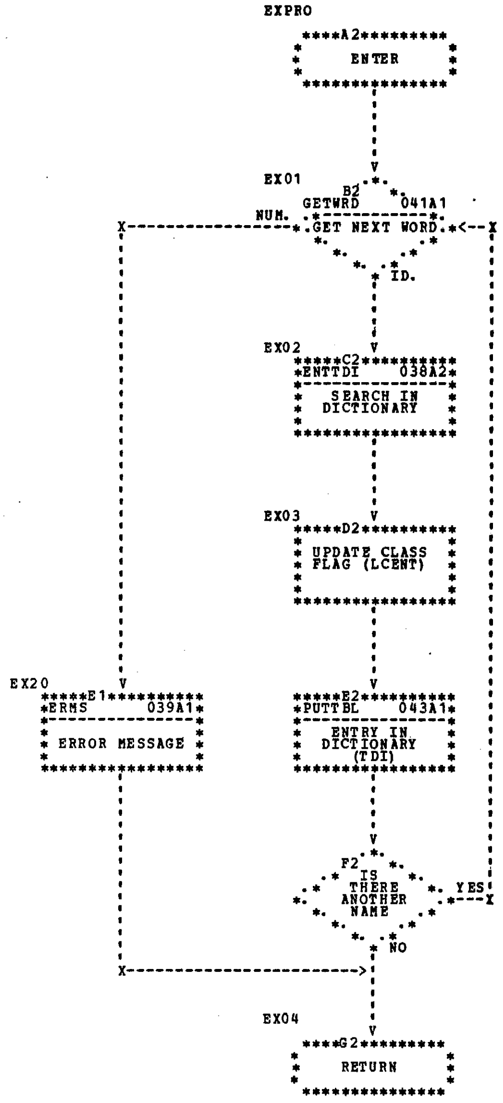


Chart 017. FNPRO Routine

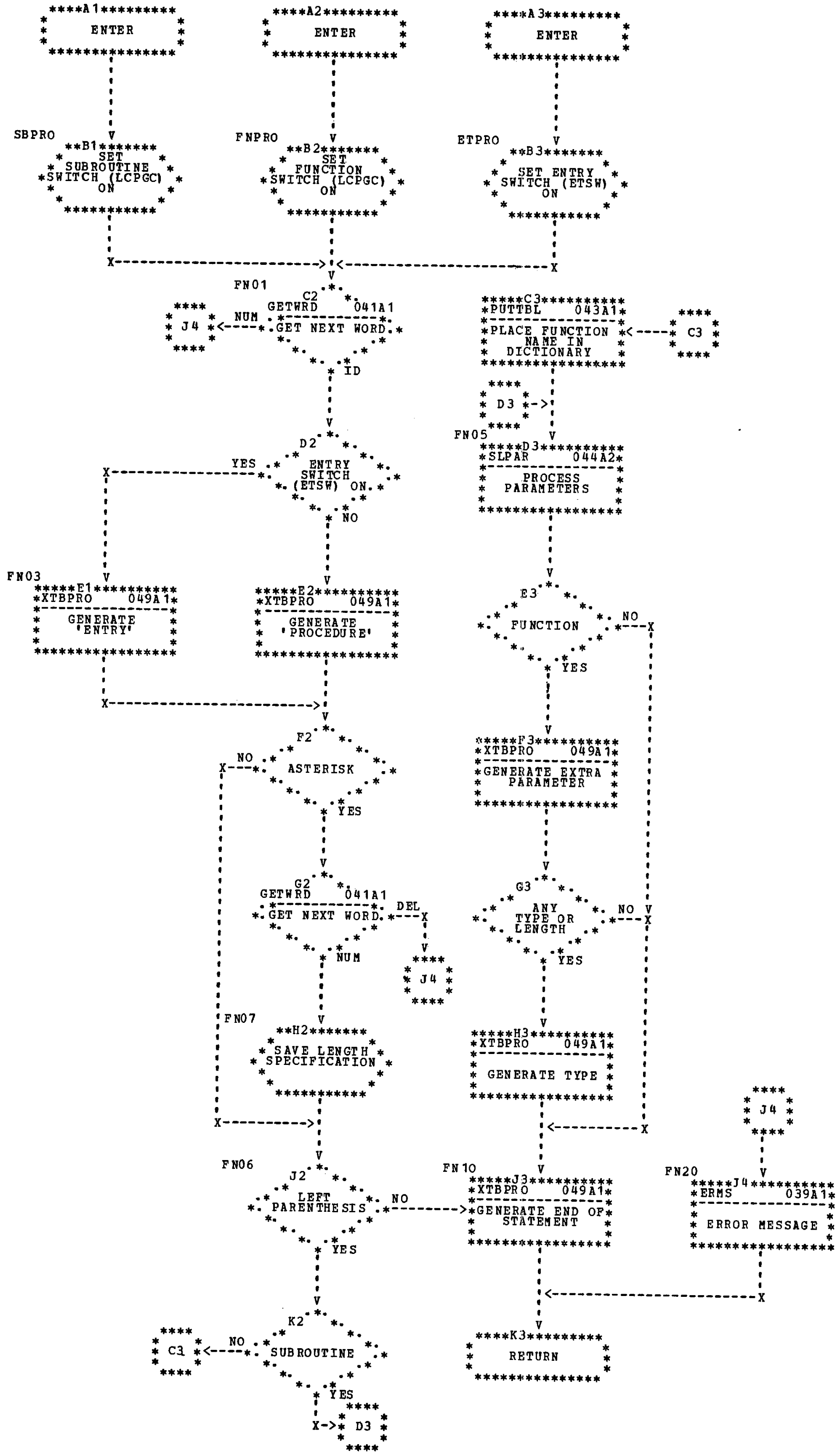


Chart 018. NLPRO Routine

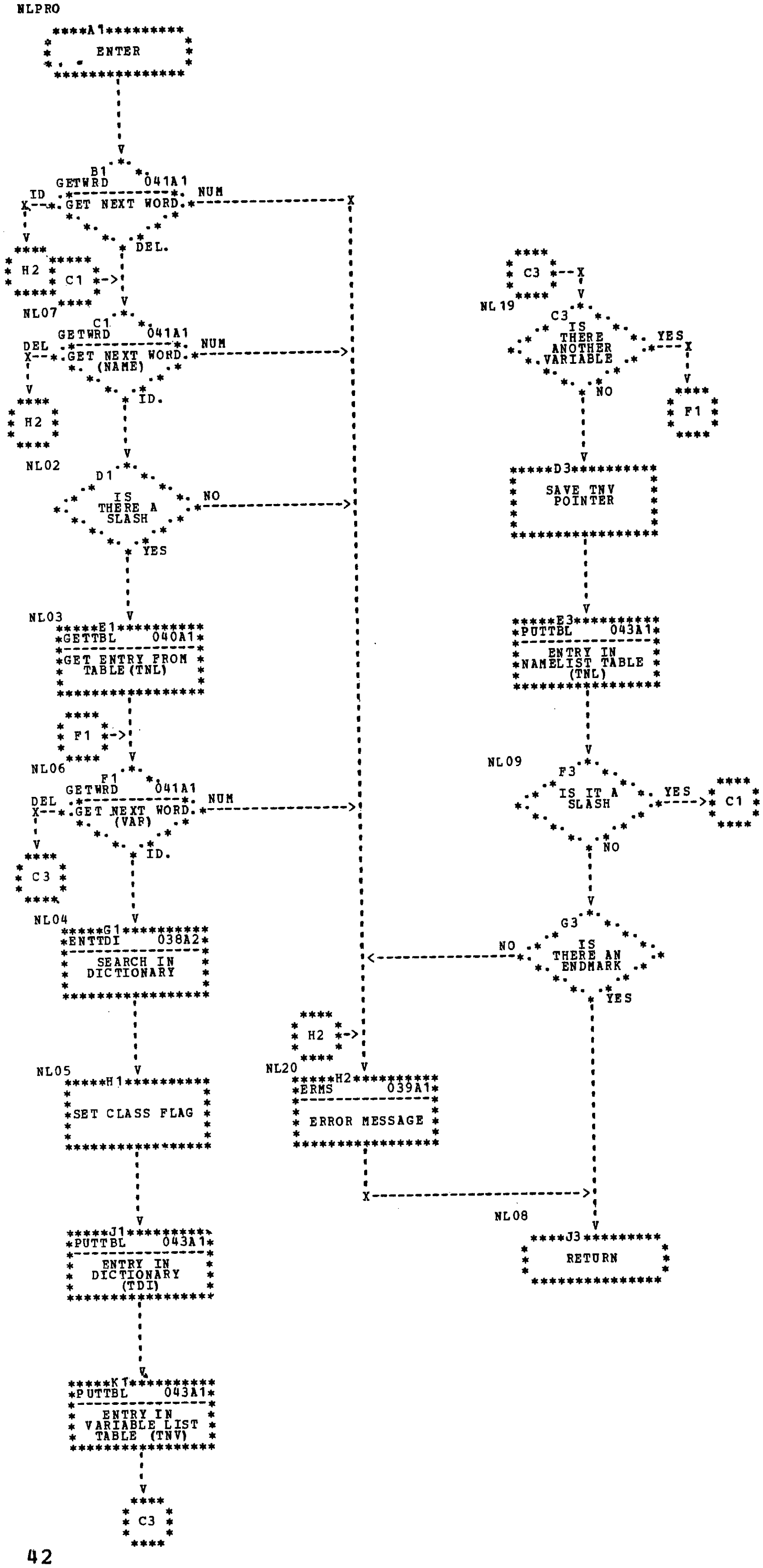


Chart 019. YIPRO Routine

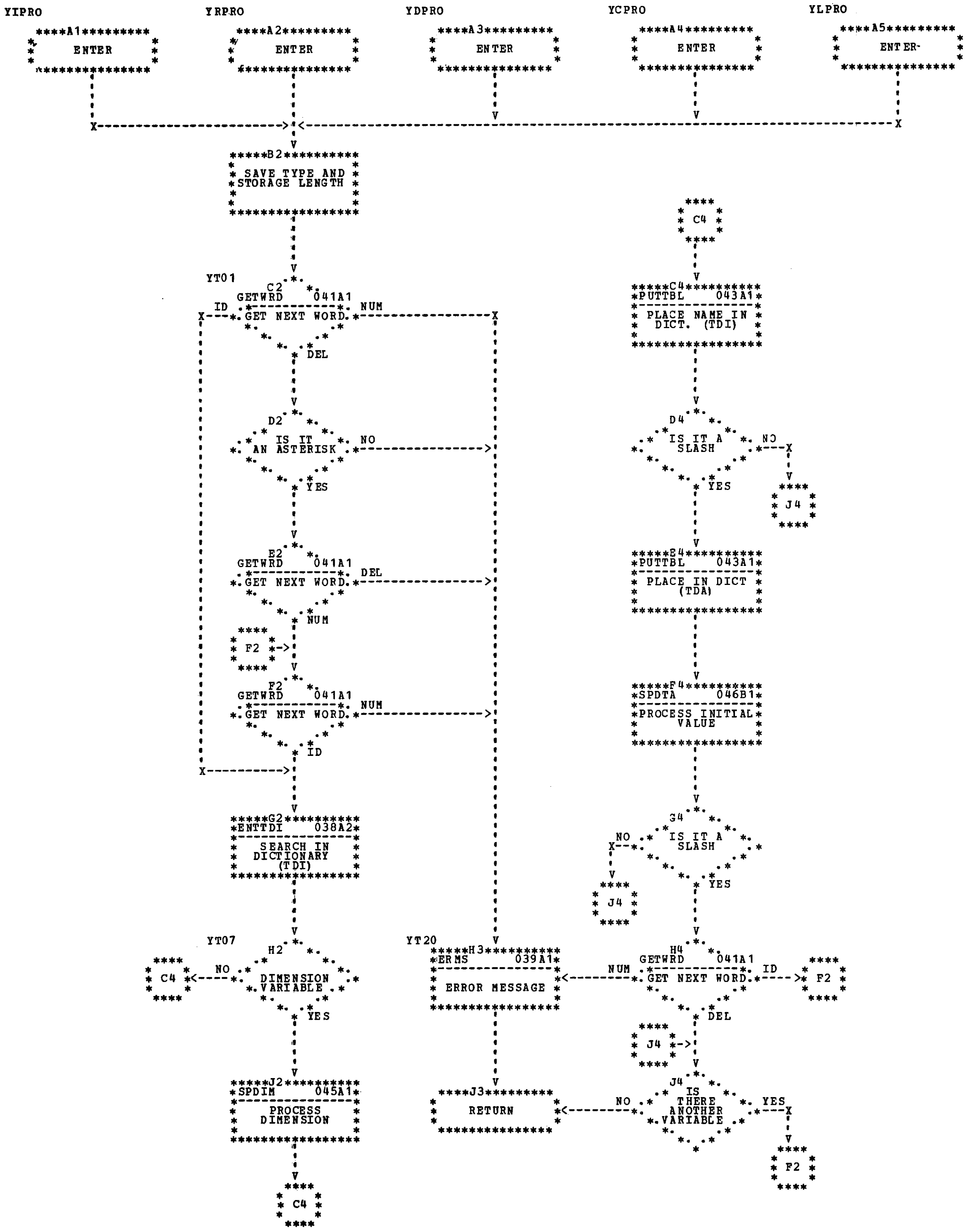


Chart 020. ALPRO Routine

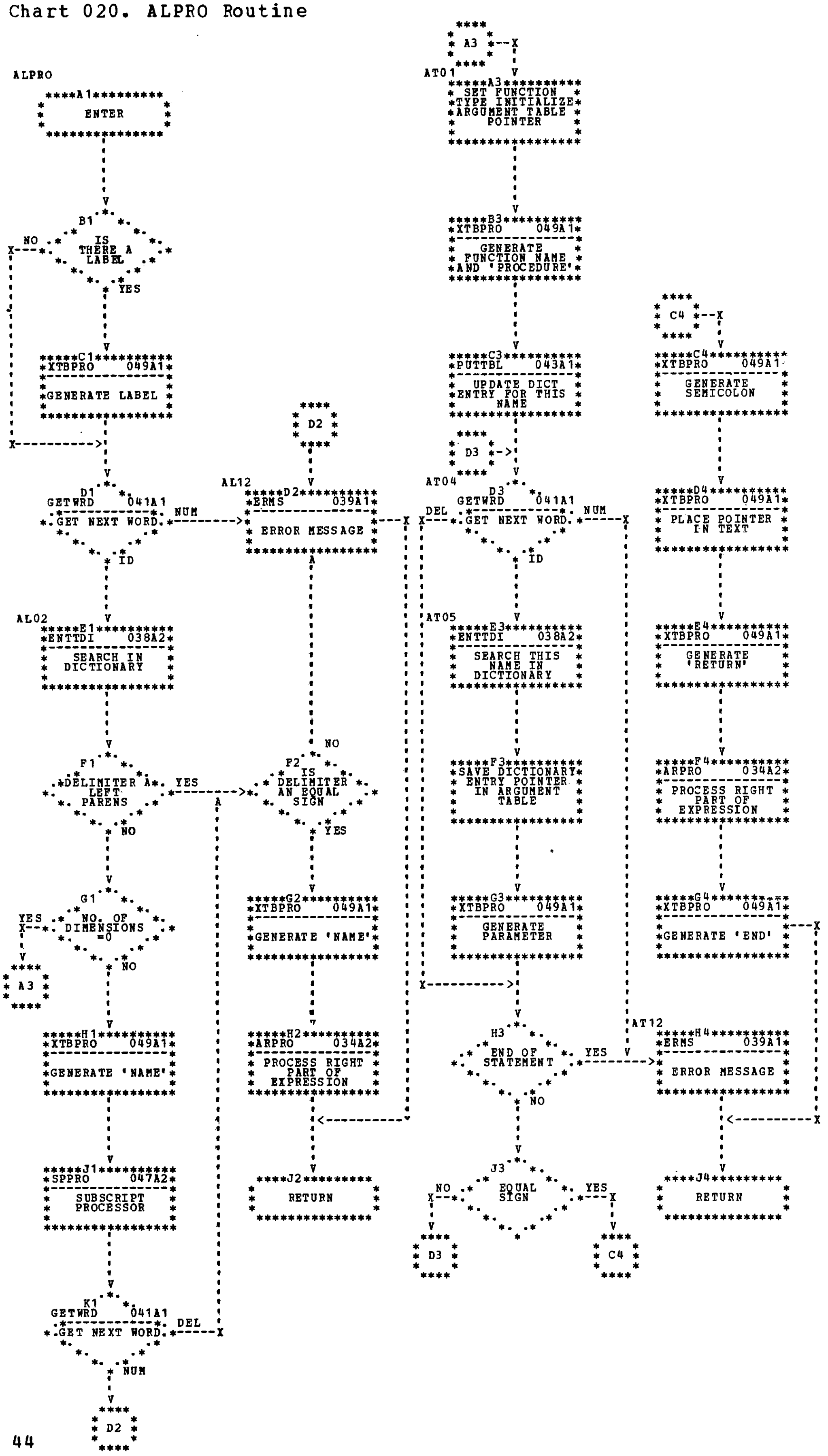


Chart 021. ASPRO Routine

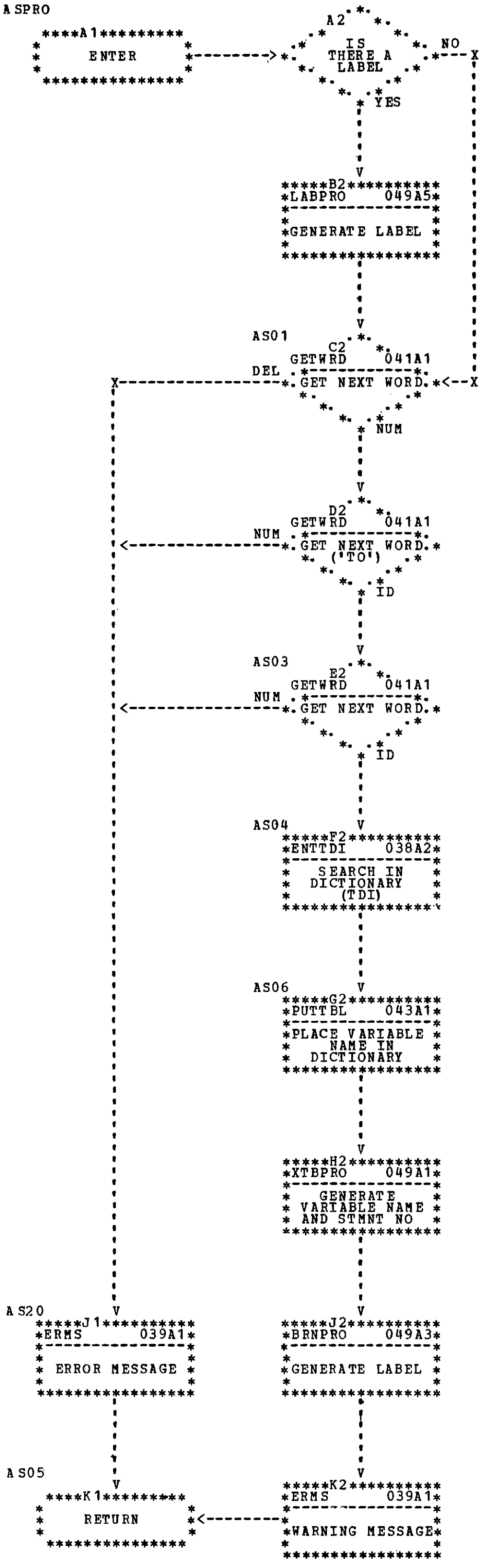


Chart 022. CAPRO Routine

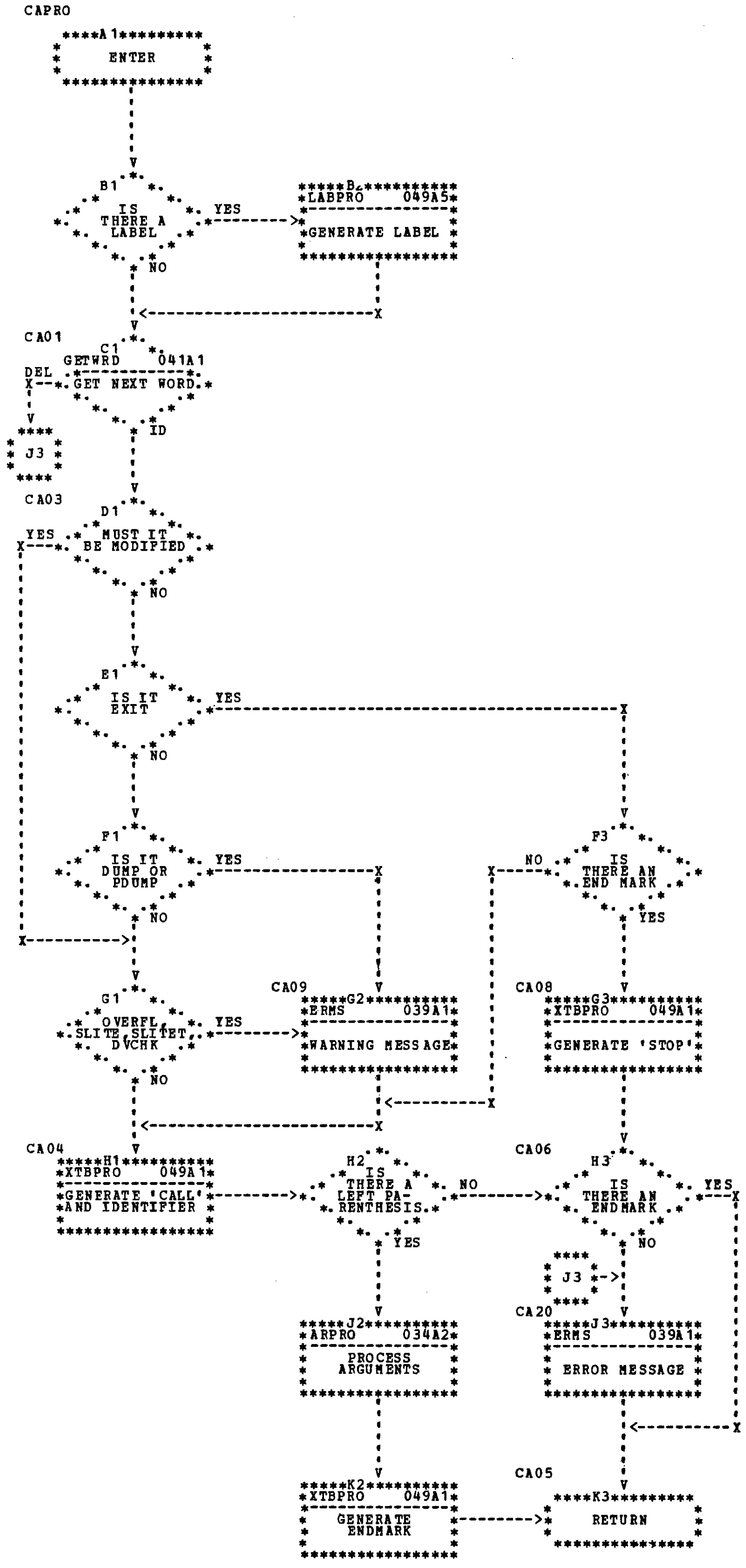


Chart 023. COPRO Routine

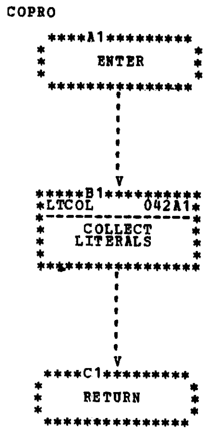


Chart 024. CTPRO Routine

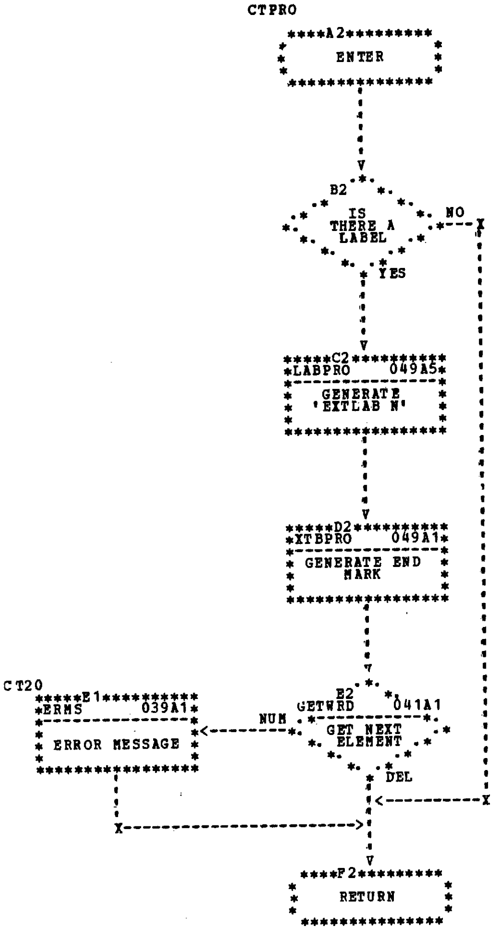


Chart 025. DOPRO Routine

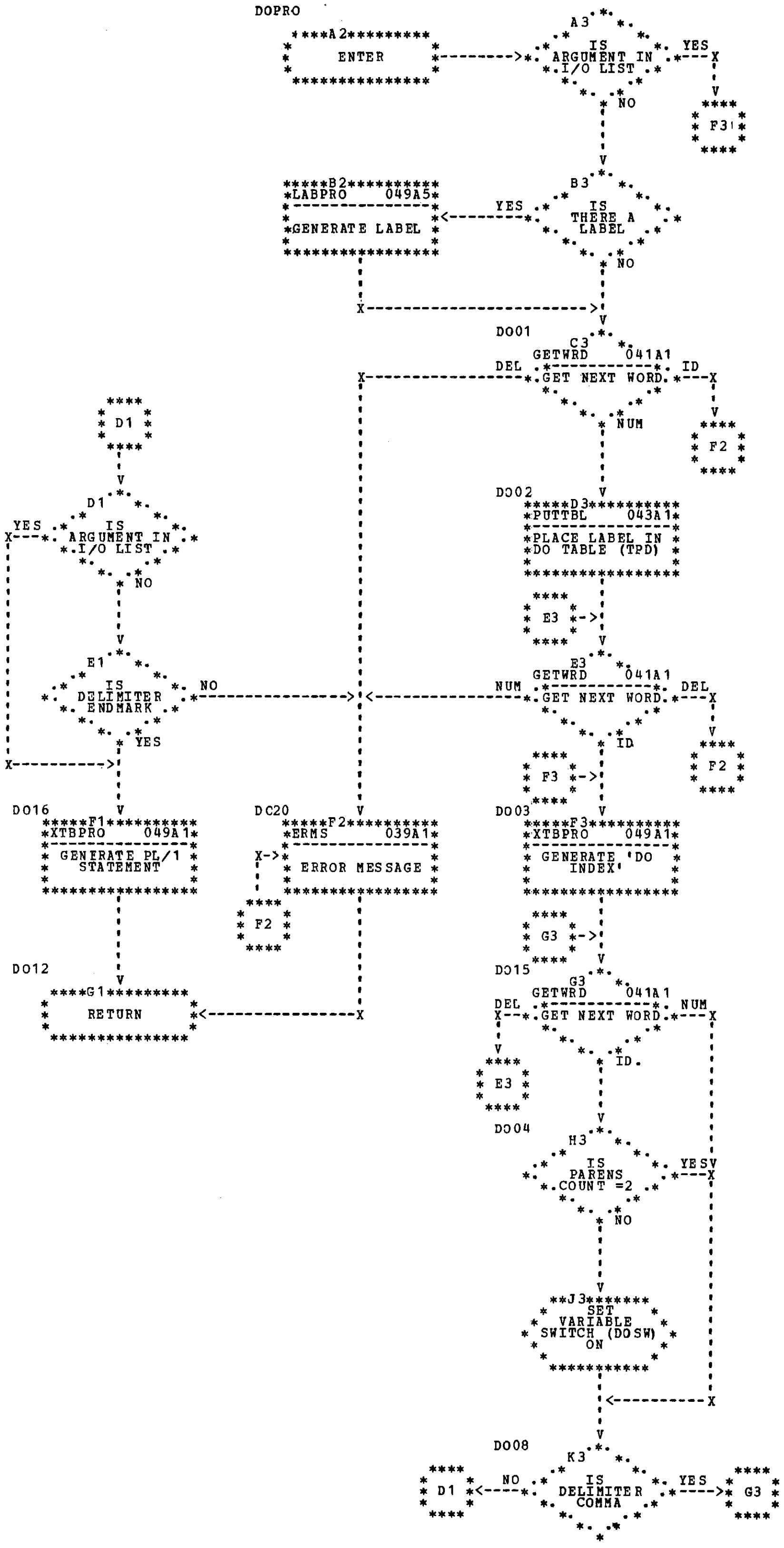


Chart 026. FTPRO Routine

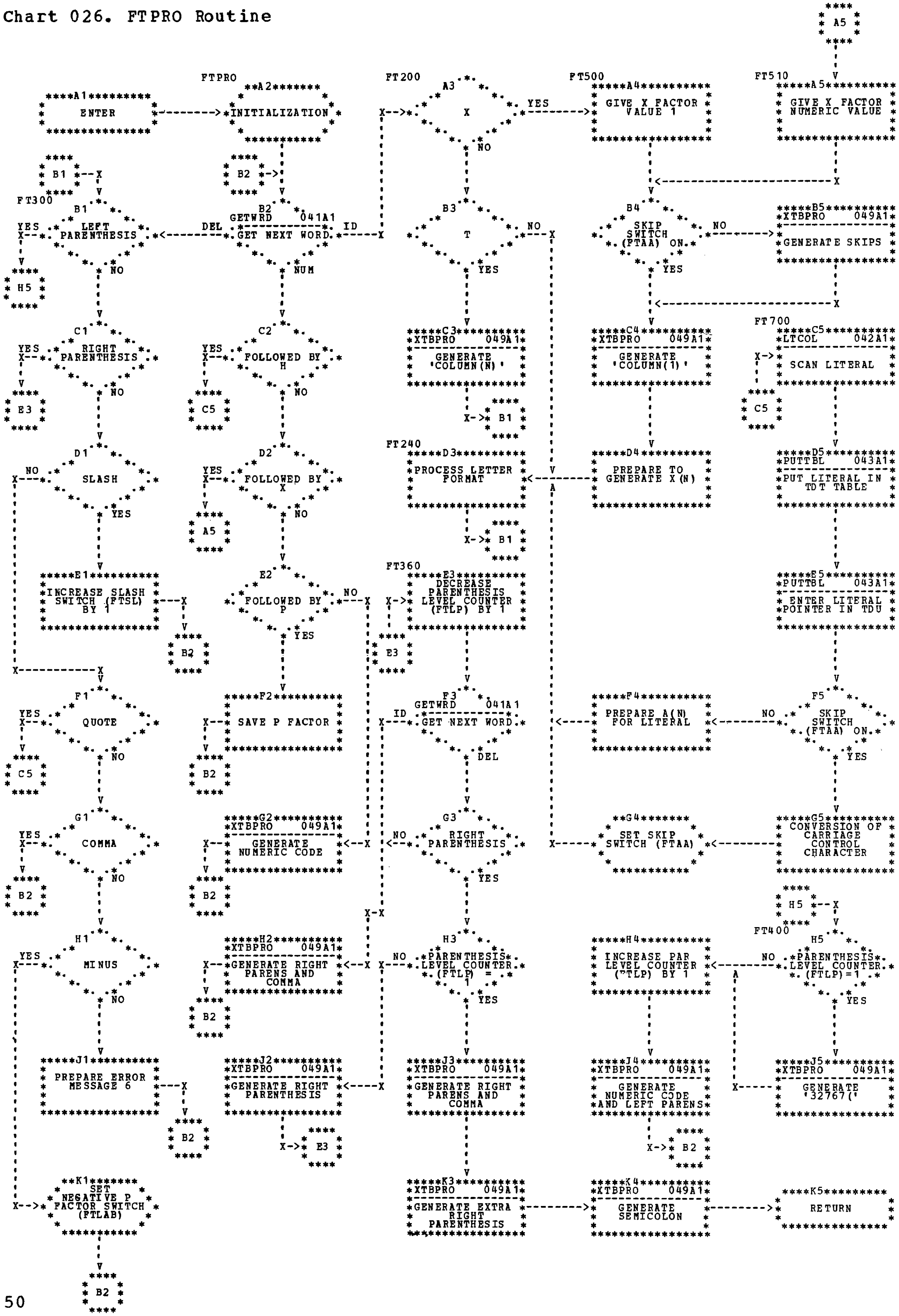

A5
A1 ENTER
FTPRO
A2 INITIALIZATION
FT200
A3 X
YES
NO
FT500
A4 GIVE X FACTOR VALUE 1
FT510
A5 GIVE X FACTOR NUMERIC VALUE
B1
B2
FT300
B1 LEFT PARENTHESIS
YES
H5
NO
B2 GETWRD 041A1 GET NEXT WORD
DEL
ID
NUM
B3 T
NO
YES
B4 SKIP SWITCH (FTAA) ON
NO
YES
B5 XTBPRO 049A1 GENERATE SKIPS
C1 RIGHT PARENTHESIS
YES
B3
NO
C2 FOLLOWED BY H
YES
C5
NO
C3 XTBPRO 049A1 GENERATE 'COLUMN(N)'
B1
C4 XTBPRO 049A1 GENERATE 'COLUMN(1)'
FT700
C5 LTCOL 042A1 SCAN LITERAL
C5
D1 SLASH
NO
YES
D2 FOLLOWED BY X
YES
A5
NO
FT240
D3 PROCESS LETTER FORMAT
B1
D4 PREPARE TO GENERATE X(N)
D5 PUTTBL 043A1 PUT LITERAL IN TDT TABLE
E1 INCREASE SLASH SWITCH (FTSL) BY 1
B2
E2 FOLLOWED BY P
NO
YES
FT360
E3 DECREASE PARENTHESIS LEVEL COUNTER (FTLP) BY 1
E3
E5 PUTTBL 043A1 ENTER LITERAL POINTER IN TDU
F1 QUOTE
YES
C5
NO
F2 SAVE P FACTOR
B2
F3 GETWRD 041A1 GET NEXT WORD
ID
DEL
F4 PREPARE A(N) FOR LITERAL
F5 SKIP SWITCH (FTAA) ON
NO
YES
G1 COMMA
YES
B2
NO
G2 XTBPRO 049A1 GENERATE NUMERIC CODE
B2
G3 RIGHT PARENTHESIS
NO
YES
G4 SET SKIP SWITCH (FTAA)
G5 CONVERSION OF CARRIAGE CONTROL CHARACTER
H5
H1 MINUS
YES
NO
H2 XTBPRO 049A1 GENERATE RIGHT PARENS AND COMMA
B2
H3 PARENTHESIS LEVEL COUNTER (FTLP) = 1
NO
YES
H4 INCREASE PAR LEVEL COUNTER ("TLP) BY 1
FT400
H5 PARENTHESIS LEVEL COUNTER (FTLP)=1
NO
YES
J1 PREPARE ERROR MESSAGE 6
B2
J2 XTBPRO 049A1 GENERATE RIGHT PARENTHESIS
E3
J3 XTBPRO 049A1 GENERATE RIGHT PARENS AND COMMA
J4 XTBPRO 049A1 GENERATE NUMERIC CODE AND LEFT PARENS
B2
J5 XTBPRO 049A1 GENERATE '32767('
K1 SET NEGATIVE P FACTOR SWITCH (FTLAB)
B2
K3 XTBPRO 049A1 GENERATE EXTRA RIGHT PARENTHESIS
K4 XTBPRO 049A1 GENERATE SEMICOLON
K5 RETURN

Chart 027. GTPRO Routine

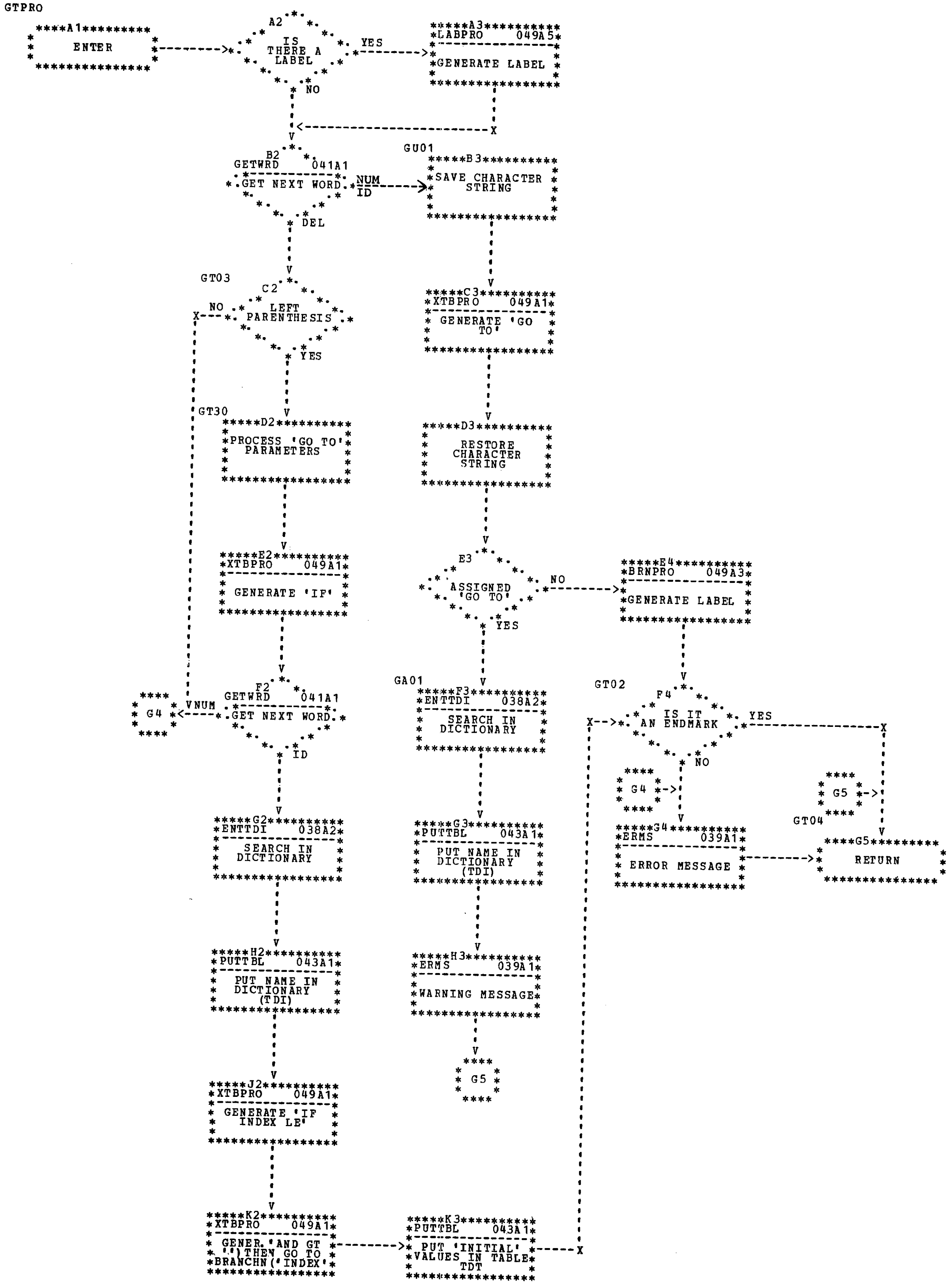


Chart 028. IFPRO Routine

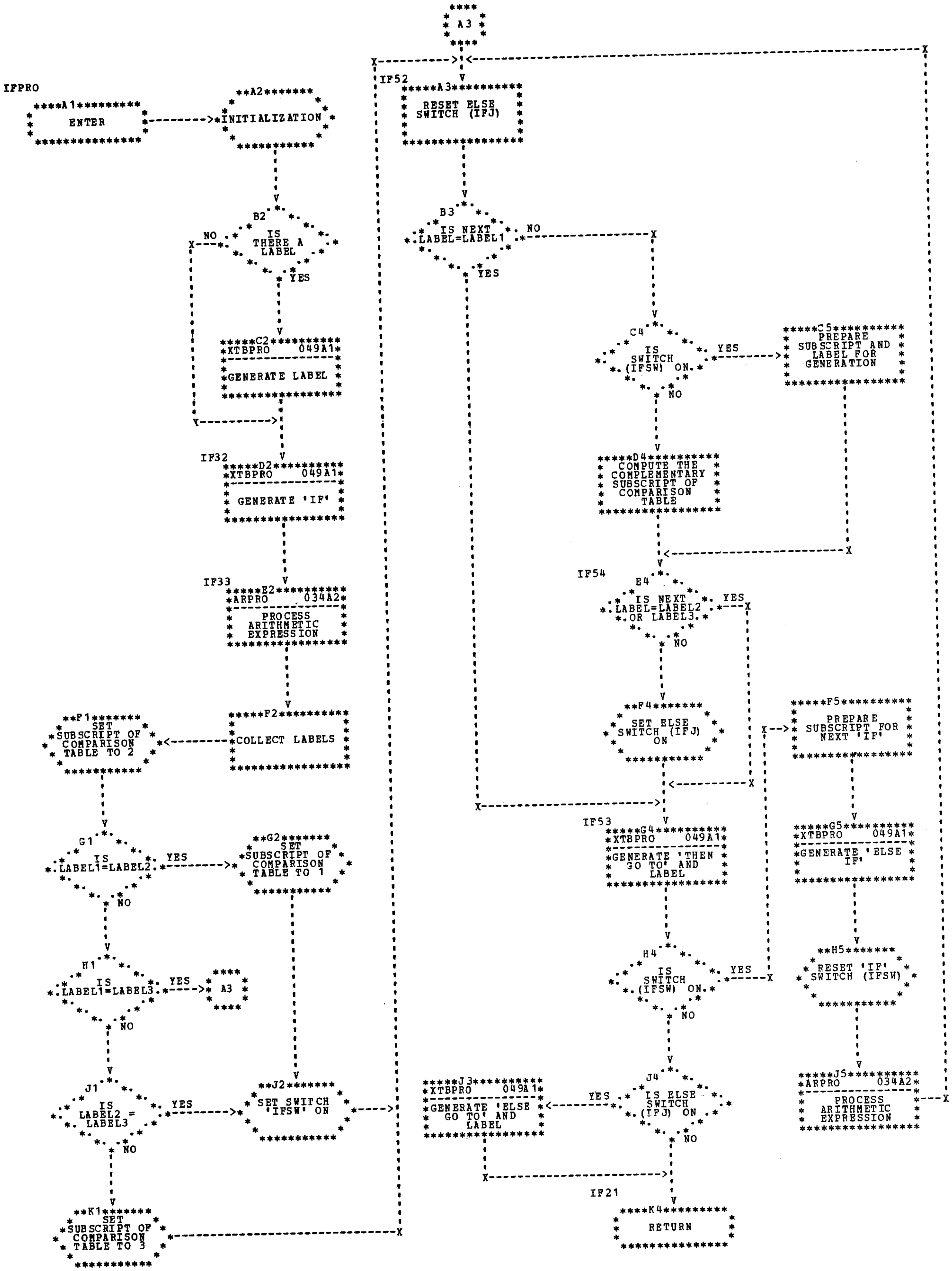


Chart 029. IOPRO Routine, Part 1 of 2

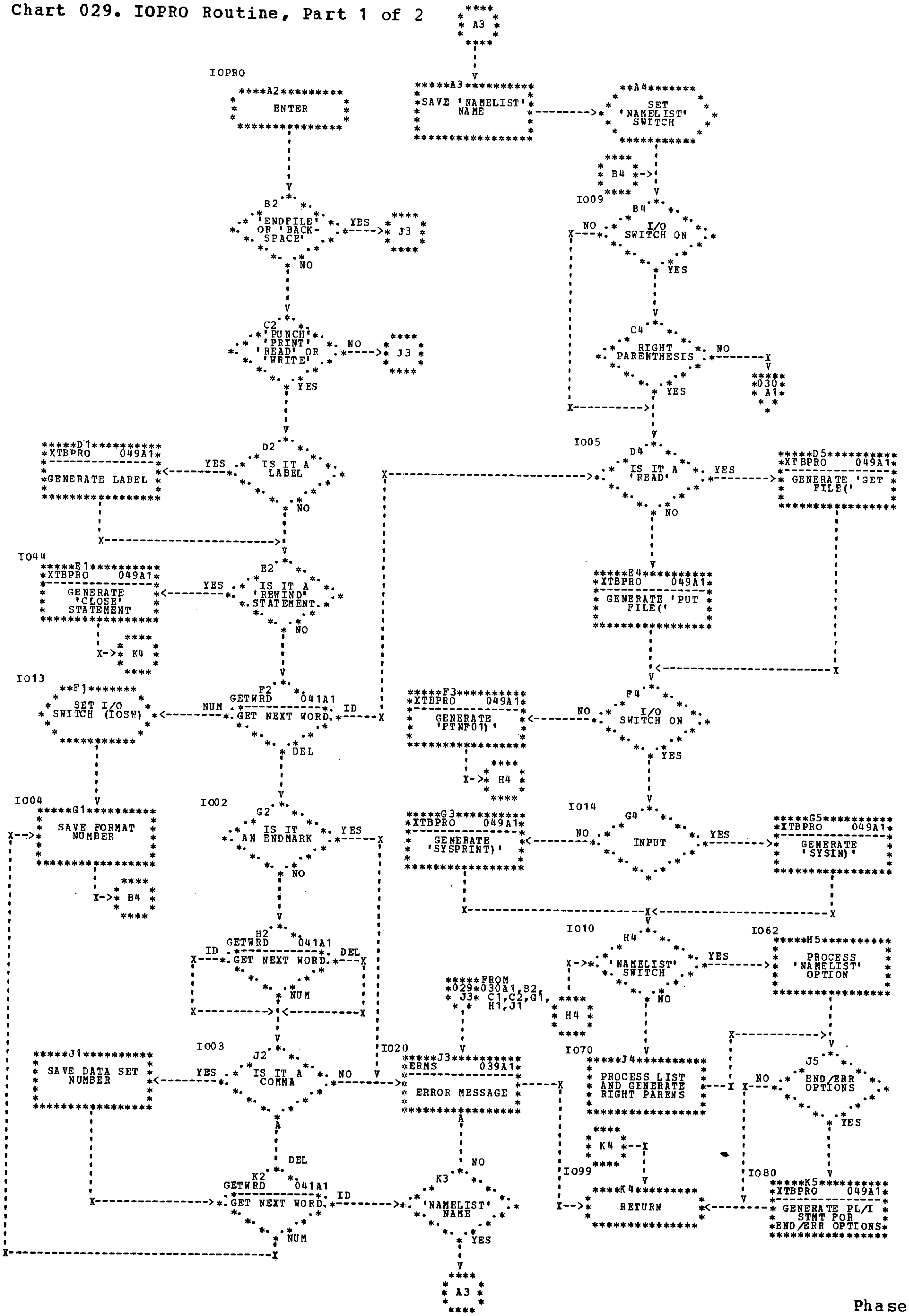

A3
IOPRO
A2 ENTER
A3 SAVE 'NAMELIST' NAME
A4 SET 'NAMELIST' SWITCH
B4
I009
B2 'ENDFILE' OR 'BACK-SPACE'
YES
J3
NO
B4 I/O SWITCH ON
NO
YES
C2 'PUNCH' 'PRINT' 'READ' OR 'WRITE'
NO
J3
YES
C4 RIGHT PARENTHESIS
NO
030 A1
YES
D1 XTBPRO 049A1 GENERATE LABEL
D2 IS IT A LABEL
YES
NO
I005
D4 IS IT A 'READ'
YES
D5 XTBPRO 049A1 GENERATE 'GET FILE('
NO
I044
E1 XTBPRO 049A1 GENERATE 'CLOSE' STATEMENT
E2 IS IT A 'REWIND' STATEMENT
YES
NO
K4
E4 XTBPRO 049A1 GENERATE 'PUT FILE('
I013
F1 SET I/O SWITCH (IOSW)
F2 GETWRD 041A1 GET NEXT WORD
NUM
ID
DEL
F3 XTBPRO 049A1 GENERATE 'FTNF01)'
F4 I/O SWITCH ON
NO
YES
H4
I004
G1 SAVE FORMAT NUMBER
B4
I002
G2 IS IT AN ENDMARK
YES
NO
G3 XTBPRO 049A1 GENERATE 'SYSPRINT)'
I014
G4 INPUT
NO
YES
G5 XTBPRO 049A1 GENERATE 'SYSIN)'
H2 GETWRD 041A1 GET NEXT WORD
ID
DEL
NUM
I010
H4 'NAMELIST' SWITCH
YES
NO
I062
H5 PROCESS 'NAMELIST' OPTION
FROM 029 030 A1,B2, J3 C1,C2,G1, H1,J1
H4
J1 SAVE DATA SET NUMBER
I003
J2 IS IT A COMMA
YES
NO
I020
J3 ERMS 039A1 ERROR MESSAGE
I070
J4 PROCESS LIST AND GENERATE RIGHT PARENS
J5 END/ERR OPTIONS
NO
YES
DEL
K2 GETWRD 041A1 GET NEXT WORD
ID
NUM
K3 'NAMELIST' NAME
NO
YES
A3
K4
I099
K4 RETURN
I080
K5 XTBPRO 049A1 GENERATE PL/I STMT FOR END/ERR OPTIONS

Chart 030. IOPRO Routine, Part 2 of 2

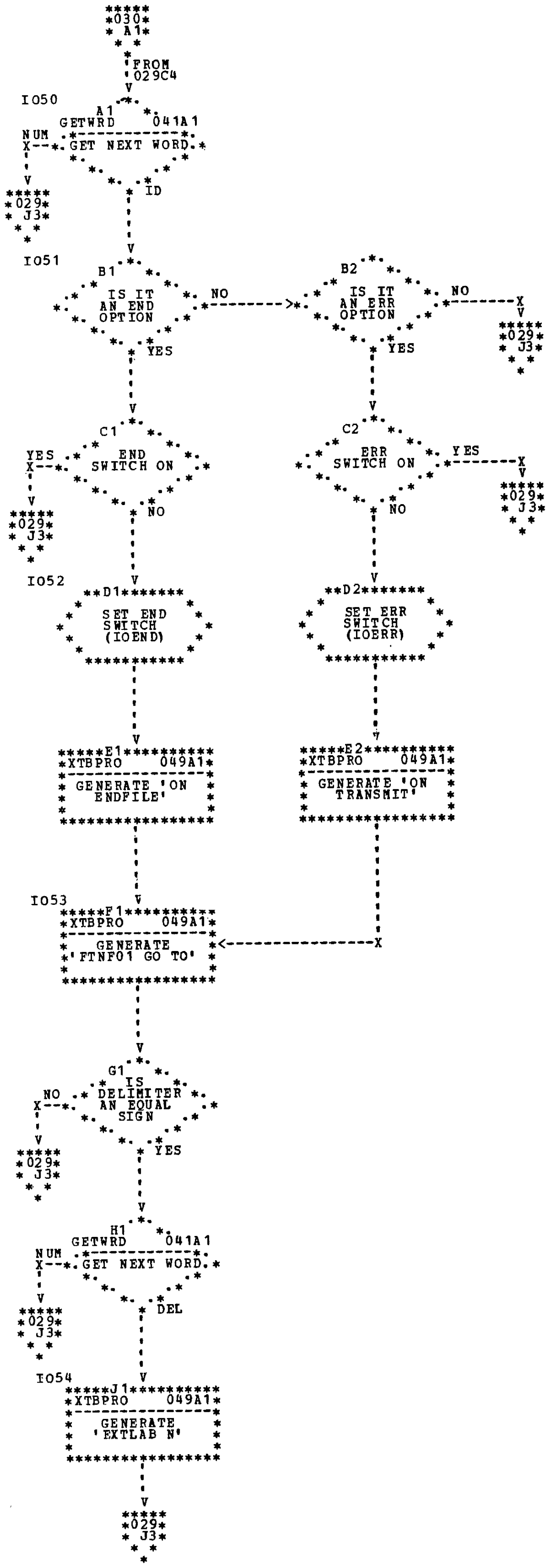


Chart 031. PSPRO Routine

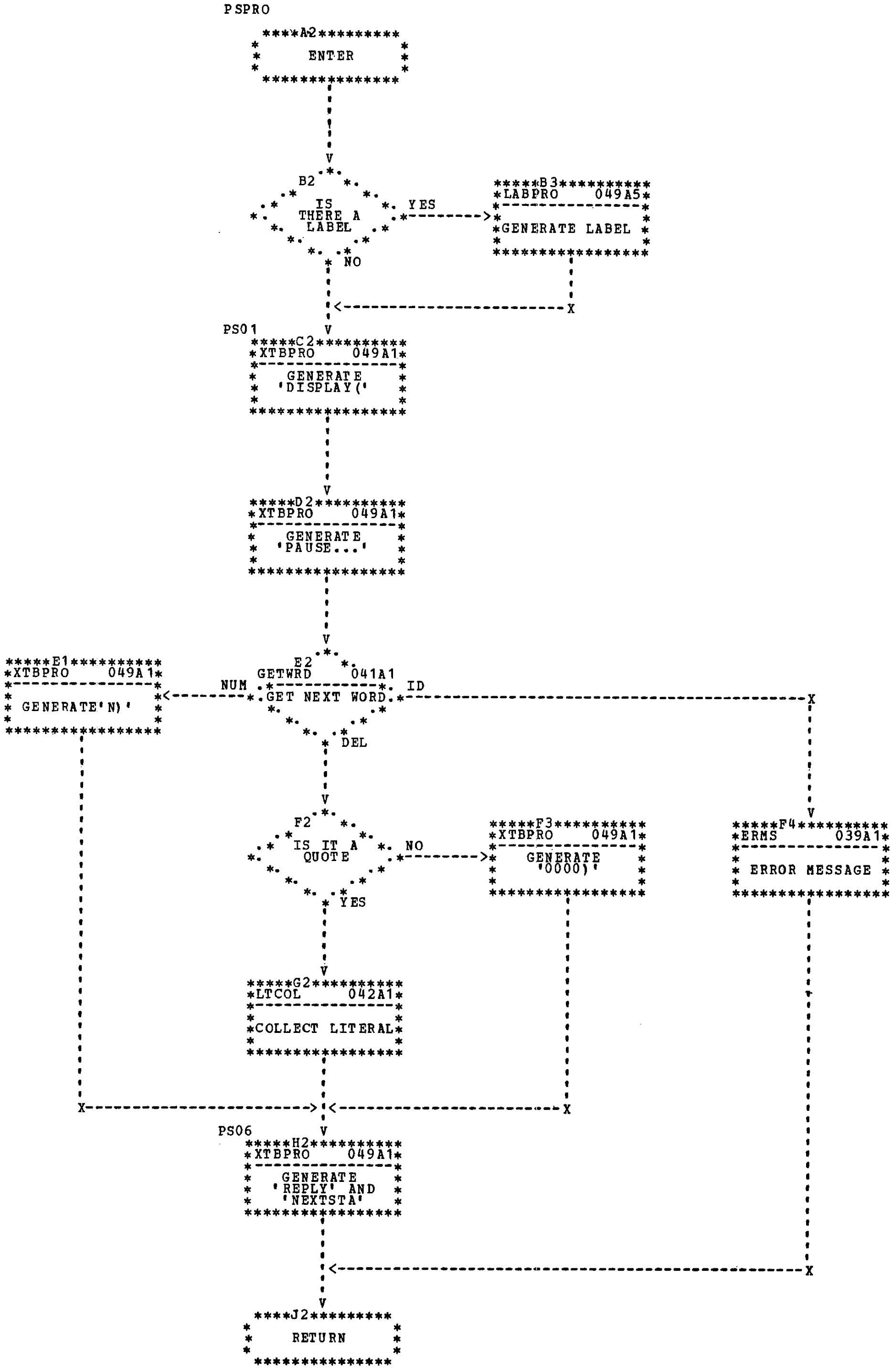


Chart 032. RTPRO Routine

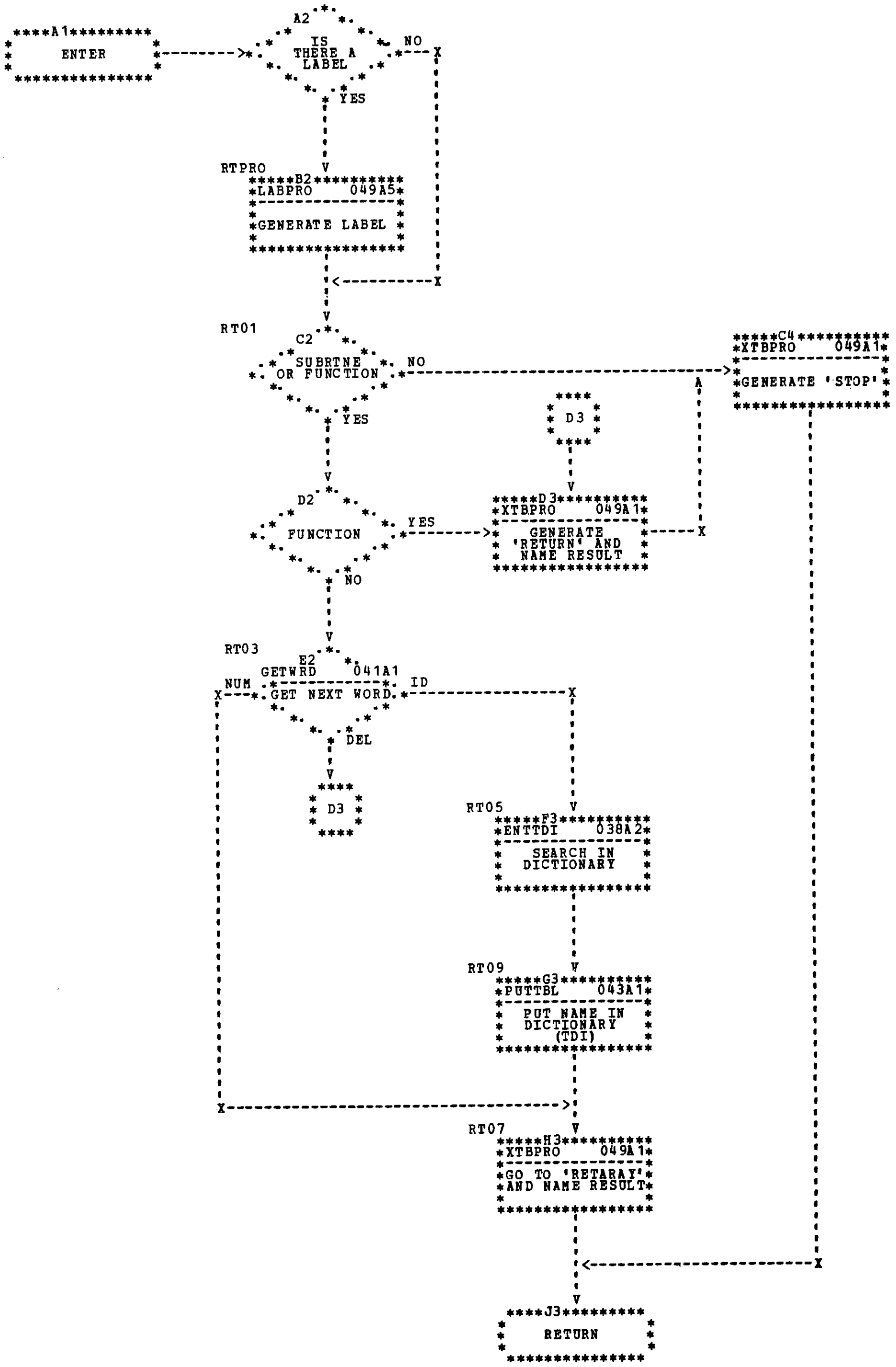


Chart 033. STPRO Routine

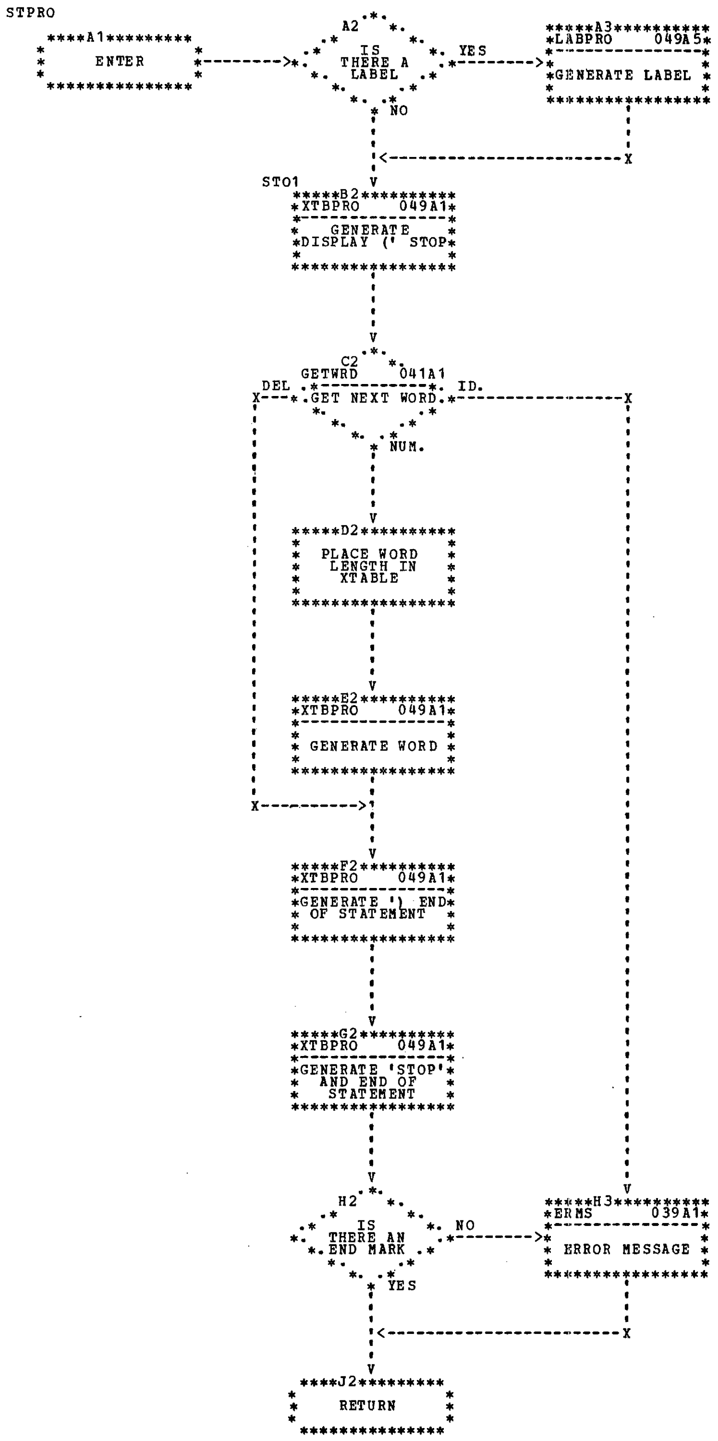
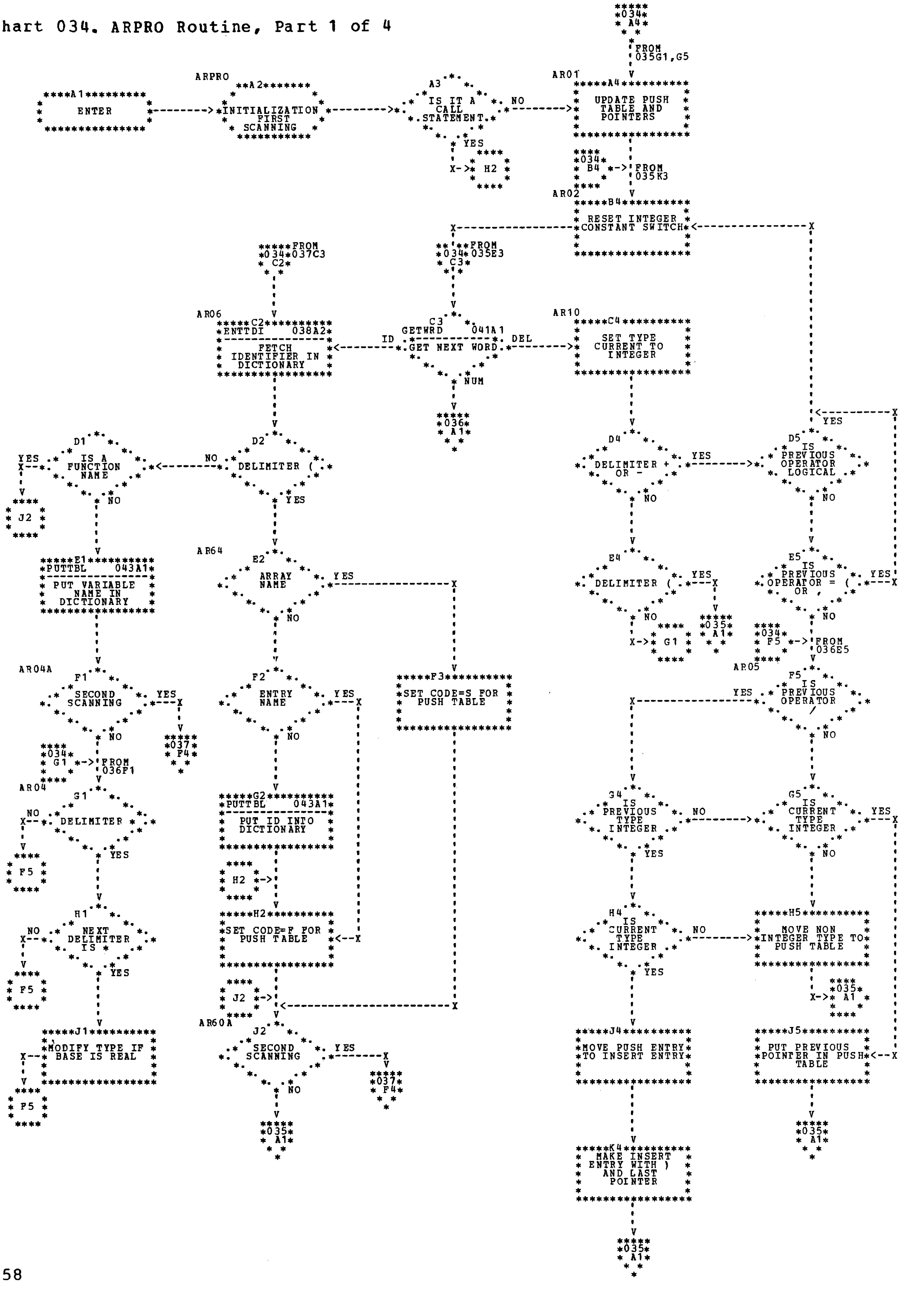
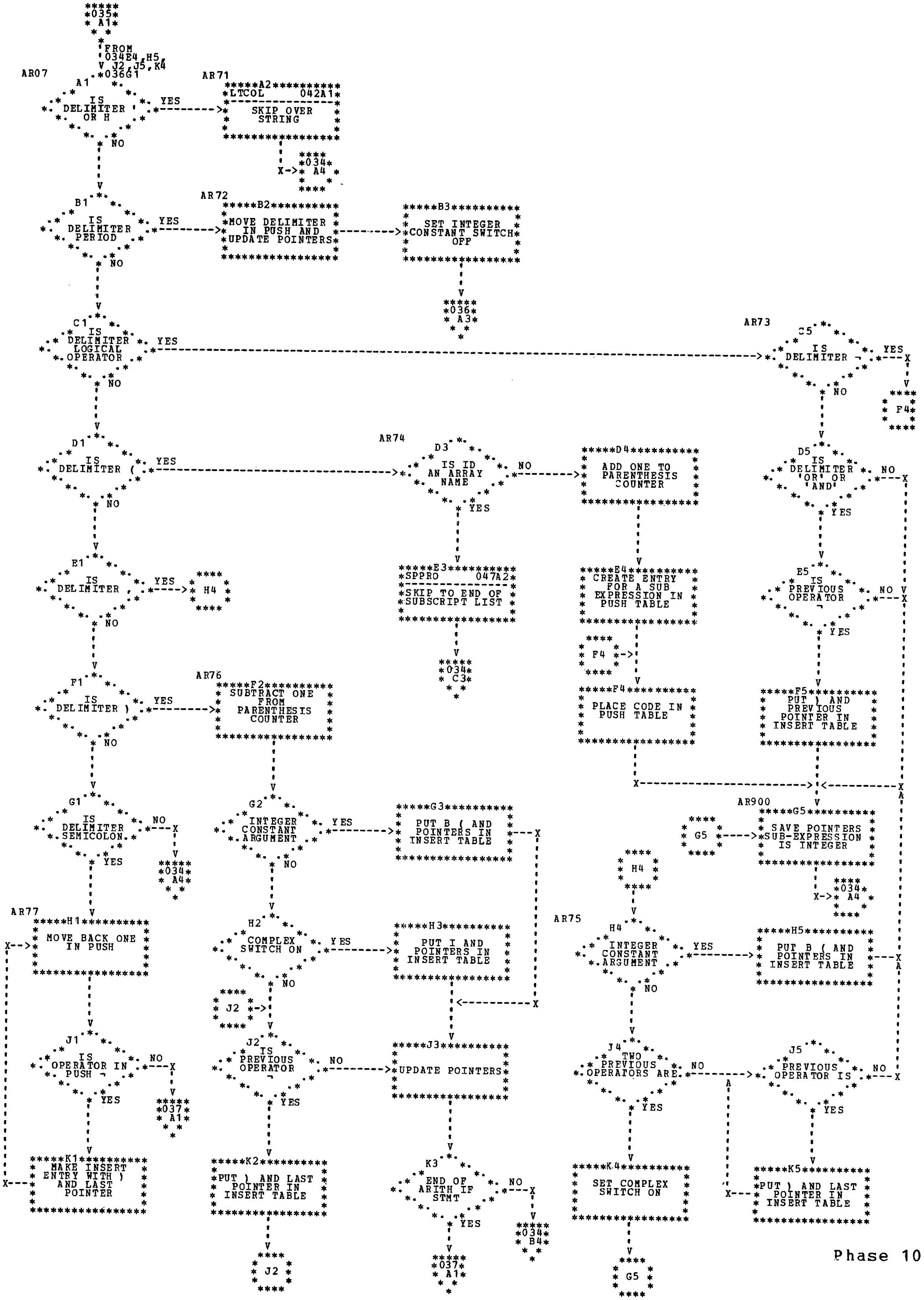


Chart 034. ARPRO Routine, Part 1 of 4



034 A4
FROM 035G1,G5
A1 ENTER
ARPRO
A2 INITIALIZATION FIRST SCANNING
A3 IS IT A CALL STATEMENT
NO
YES
H2
AR01
A4 UPDATE PUSH TABLE AND POINTERS
034 B4
FROM 035K3
AR02
B4 RESET INTEGER CONSTANT SWITCH
034 C2 FROM 037C3
034 C3 FROM 035E3
AR06
C2 ENTTDI 038A2 FETCH IDENTIFIER IN DICTIONARY
C3 GETWRD 041A1 GET NEXT WORD
ID
DEL
NUM
036 A1
AR10
C4 SET TYPE CURRENT TO INTEGER
D1 IS A FUNCTION NAME
YES
NO
J2
D2 DELIMITER (
NO
YES
D4 DELIMITER + OR -
YES
NO
D5 IS PREVIOUS OPERATOR LOGICAL
YES
NO
E1 PUTTBL 043A1 PUT VARIABLE NAME IN DICTIONARY
AR64
E2 ARRAY NAME
YES
NO
E4 DELIMITER (
YES
NO
G1
035 A1
E5 IS PREVIOUS OPERATOR = (OR ,
YES
NO
034 F5
FROM 036E5
AR04A
F1 SECOND SCANNING
YES
NO
037 F4
F2 ENTRY NAME
YES
NO
F3 SET CODE=S FOR PUSH TABLE
AR05
F5 IS PREVIOUS OPERATOR /
YES
NO
034 G1
FROM 036F1
AR04
G1 DELIMITER *
NO
YES
F5
G2 PUTTBL 043A1 PUT ID INTO DICTIONARY
G4 IS PREVIOUS TYPE INTEGER
NO
YES
G5 IS CURRENT TYPE INTEGER
YES
NO
H1 NEXT DELIMITER IS *
NO
YES
F5
H2
H2 SET CODE=F FOR PUSH TABLE
H4 IS CURRENT TYPE INTEGER
NO
YES
H5 MOVE NON INTEGER TYPE TO PUSH TABLE
035 A1
J2
AR60A
J1 MODIFY TYPE IF BASE IS REAL
F5
J2 SECOND SCANNING
YES
NO
037 F4
035 A1
J4 MOVE PUSH ENTRY TO INSERT ENTRY
J5 PUT PREVIOUS POINTER IN PUSH TABLE
035 A1
K4 MAKE INSERT ENTRY WITH) AND LAST POINTER
035 A1

Chart 035. ARPRO Routine, Part 2 of 4



035 A1
FROM 034E4,H5, J2,J5,K4 036G1
AR07 A1 IS DELIMITER ' OR H
YES
NO
AR71 A2 LTCOL 042A1 SKIP OVER STRING
034 A4
AR72 B2 MOVE DELIMITER IN PUSH AND UPDATE POINTERS
B1 IS DELIMITER PERIOD
B3 SET INTEGER CONSTANT SWITCH OFF
036 A3
C1 IS DELIMITER LOGICAL OPERATOR
AR73 C5 IS DELIMITER ¬
F4
D1 IS DELIMITER (
AR74 D3 IS ID AN ARRAY NAME
D4 ADD ONE TO PARENTHESIS COUNTER
D5 IS DELIMITER 'OR' OR 'AND'
E1 IS DELIMITER ,
H4
E3 SPPRO 047A2 SKIP TO END OF SUBSCRIPT LIST
034 C3
E4 CREATE ENTRY FOR A SUB EXPRESSION IN PUSH TABLE
E5 IS PREVIOUS OPERATOR ¬
F1 IS DELIMITER)
AR76 F2 SUBTRACT ONE FROM PARENTHESIS COUNTER
F4 PLACE CODE IN PUSH TABLE
F5 PUT) AND PREVIOUS POINTER IN INSERT TABLE
G1 IS DELIMITER SEMICOLON
034 A4
G2 INTEGER CONSTANT ARGUMENT
G3 PUT B (AND POINTERS IN INSERT TABLE
AR900 G5 SAVE POINTERS SUB-EXPRESSION IS INTEGER
G5
034 A4
AR77 H1 MOVE BACK ONE IN PUSH
H2 COMPLEX SWITCH ON
H3 PUT I AND POINTERS IN INSERT TABLE
AR75 H4 INTEGER CONSTANT ARGUMENT
H5 PUT B (AND POINTERS IN INSERT TABLE
J2
J1 IS OPERATOR IN PUSH ¬
037 A1
J2 IS PREVIOUS OPERATOR ¬
J3 UPDATE POINTERS
J4 TWO PREVIOUS OPERATORS ARE ¬
J5 PREVIOUS OPERATOR IS ¬
K1 MAKE INSERT ENTRY WITH) AND LAST POINTER
K2 PUT) AND LAST POINTER IN INSERT TABLE
J2
K3 END OF ARITH IF STMT
037 A1
034 B4
K4 SET COMPLEX SWITCH ON
G5
K5 PUT) AND LAST POINTER IN INSERT TABLE

Chart 036. ARPRO Routine 3 of 4

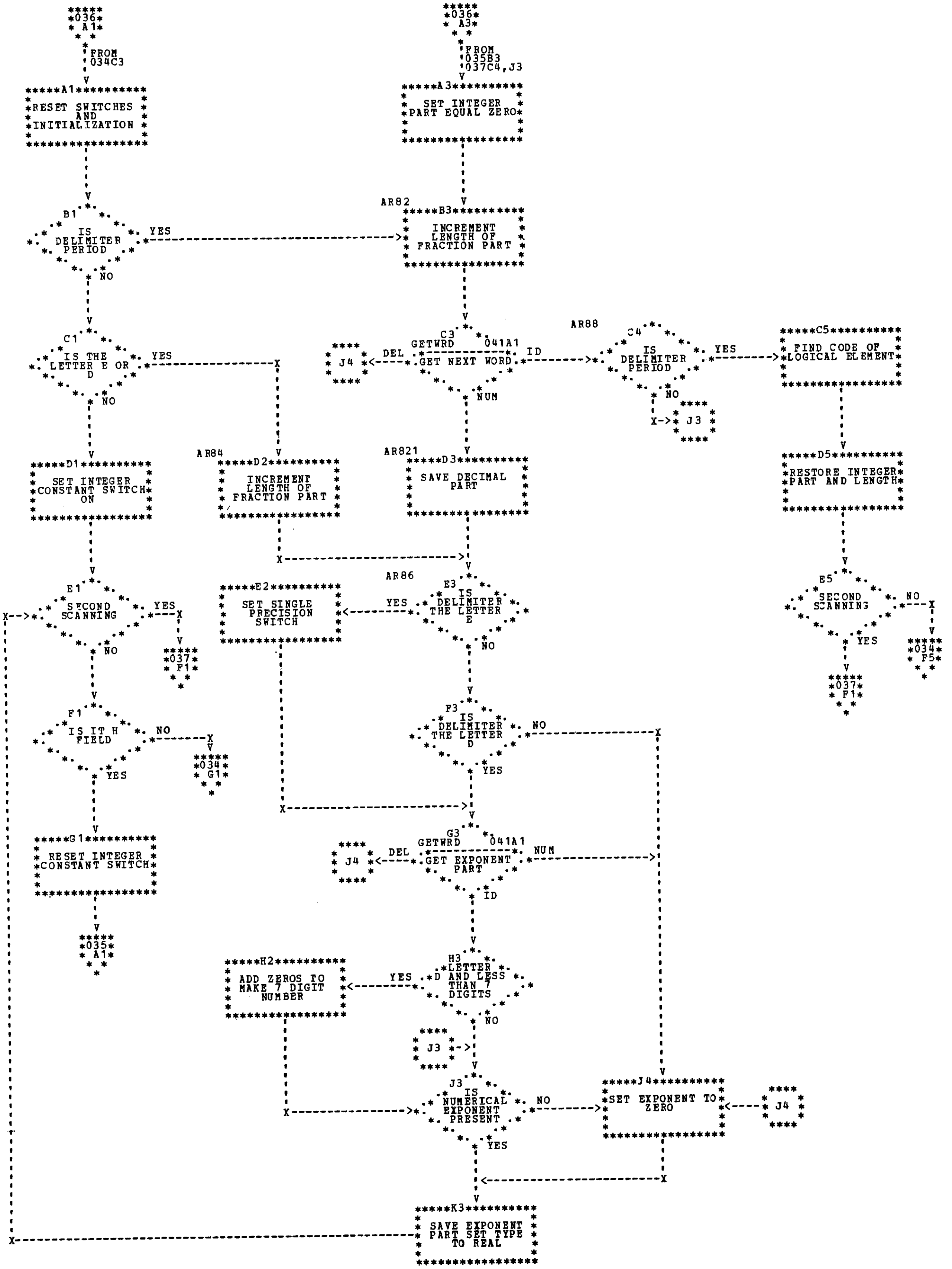

036 A1
FROM 034C3
A1
RESET SWITCHES AND INITIALIZATION
B1
IS DELIMITER PERIOD
YES
NO
C1
IS THE LETTER E OR D
YES
NO
D1
SET INTEGER CONSTANT SWITCH ON
E1
SECOND SCANNING
YES
NO
037 F1
F1
IS IT H FIELD
NO
YES
034 G1
G1
RESET INTEGER CONSTANT SWITCH
035 A1
AR84
D2
INCREMENT LENGTH OF FRACTION PART
E2
SET SINGLE PRECISION SWITCH
H2
ADD ZEROS TO MAKE 7 DIGIT NUMBER
036 A3
FROM 035B3 037C4,J3
A3
SET INTEGER PART EQUAL ZERO
AR82
B3
INCREMENT LENGTH OF FRACTION PART
C3
GETWRD 041A1
GET NEXT WORD
DEL
J4
ID
NUM
AR821
D3
SAVE DECIMAL PART
AR86
E3
IS DELIMITER THE LETTER E
YES
NO
F3
IS DELIMITER THE LETTER D
NO
YES
G3
GETWRD 041A1
GET EXPONENT PART
DEL
J4
NUM
ID
H3
LETTER D AND LESS THAN 7 DIGITS
YES
NO
J3
J3
IS NUMERICAL EXPONENT PRESENT
NO
YES
J4
SET EXPONENT TO ZERO
J4
K3
SAVE EXPONENT PART SET TYPE TO REAL
AR88
C4
IS DELIMITER PERIOD
YES
NO
J3
C5
FIND CODE OF LOGICAL ELEMENT
D5
RESTORE INTEGER PART AND LENGTH
E5
SECOND SCANNING
NO
YES
037 F1
034 F5

Chart 037. ARPRO Routine, Part 4 of 4

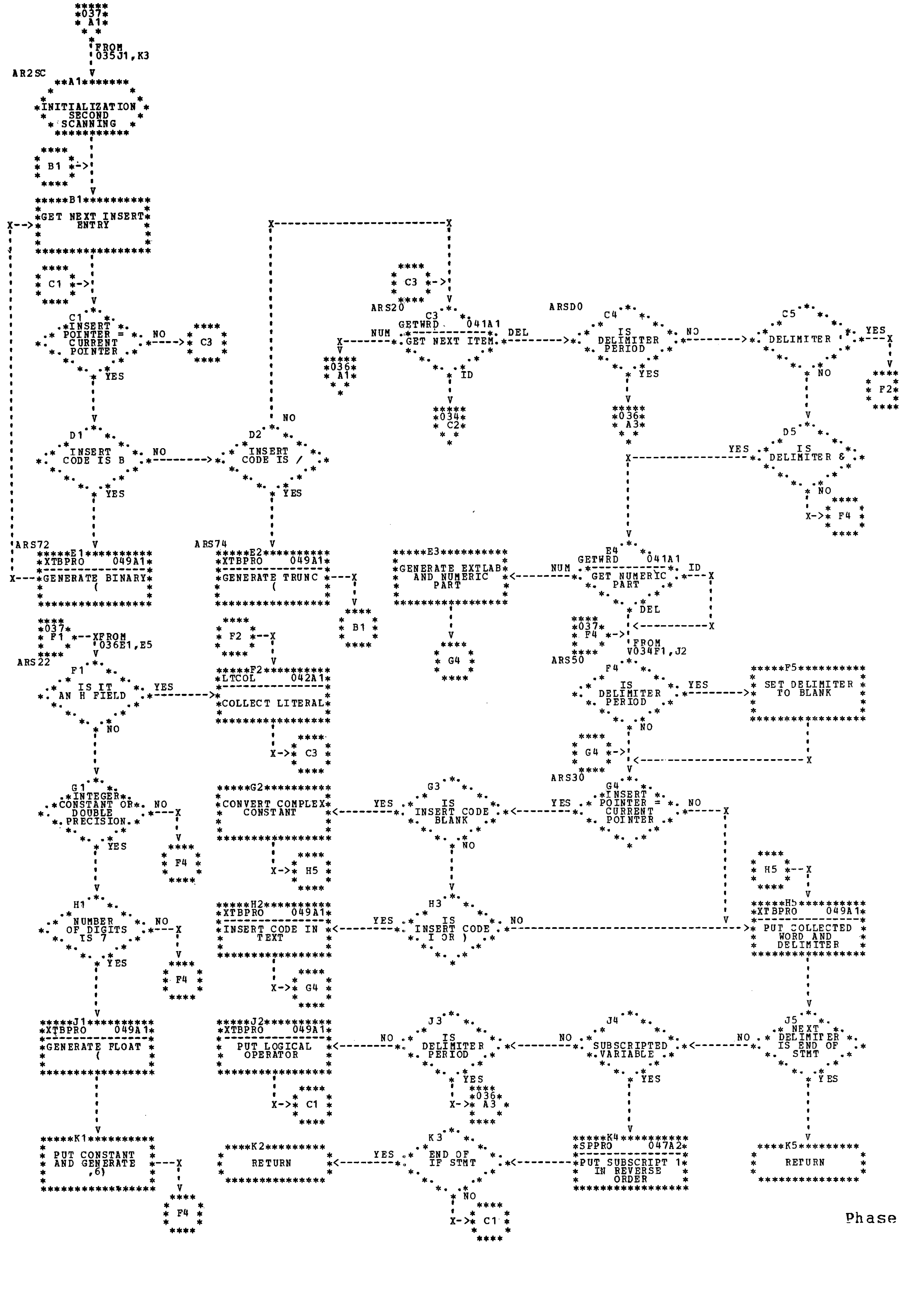

037 A1
FROM 035J1,K3
AR2SC
A1 INITIALIZATION SECOND SCANNING
B1
B1 GET NEXT INSERT ENTRY
C1
C1 INSERT POINTER = CURRENT POINTER
NO
C3
YES
D1 INSERT CODE IS B
NO
YES
D2 INSERT CODE IS /
NO
YES
ARS72
E1 XTBPRO 049A1 GENERATE BINARY (
ARS74
E2 XTBPRO 049A1 GENERATE TRUNC (
B1
037 F1
FROM 036E1,E5
ARS22
F1 IS IT AN H FIELD
YES
NO
F2
F2 LTCOL 042A1 COLLECT LITERAL
C3
G1 INTEGER CONSTANT OR DOUBLE PRECISION
NO
F4
YES
G2 CONVERT COMPLEX CONSTANT
H5
H1 NUMBER OF DIGITS IS 7
NO
F4
YES
H2 XTBPRO 049A1 INSERT CODE IN TEXT
G4
J1 XTBPRO 049A1 GENERATE FLOAT (
J2 XTBPRO 049A1 PUT LOGICAL OPERATOR
C1
K1 PUT CONSTANT AND GENERATE ,6)
F4
K2 RETURN
C3
ARS20
C3 GETWRD 041A1 GET NEXT ITEM
NUM
DEL
ID
036 A1
034 C2
ARSD0
C4 IS DELIMITER PERIOD
NO
YES
036 A3
C5 DELIMITER '
YES
F2
NO
D5 IS DELIMITER &
YES
NO
F4
E3 GENERATE EXTLAB AND NUMERIC PART
G4
E4 GETWRD 041A1 GET NUMERIC PART
NUM
ID
DEL
037 F4
FROM V034F1,J2
ARS50
F4 IS DELIMITER PERIOD
YES
NO
F5 SET DELIMITER TO BLANK
G4
ARS30
G4 INSERT POINTER = CURRENT POINTER
YES
NO
G3 IS INSERT CODE BLANK
YES
NO
H3 IS INSERT CODE I OR)
YES
NO
H5
H5 XTBPRO 049A1 PUT COLLECTED WORD AND DELIMITER
J3 IS DELIMITER PERIOD
NO
YES
036 A3
J4 SUBSCRIPTED VARIABLE
NO
YES
J5 NEXT DELIMITER IS END OF STMT
NO
YES
K3 END OF IF STMT
YES
NO
C1
K4 SPPRO 047A2 PUT SUBSCRIPT 1 IN REVERSE ORDER
K5 RETURN

Chart 038. ENTTDI Routine

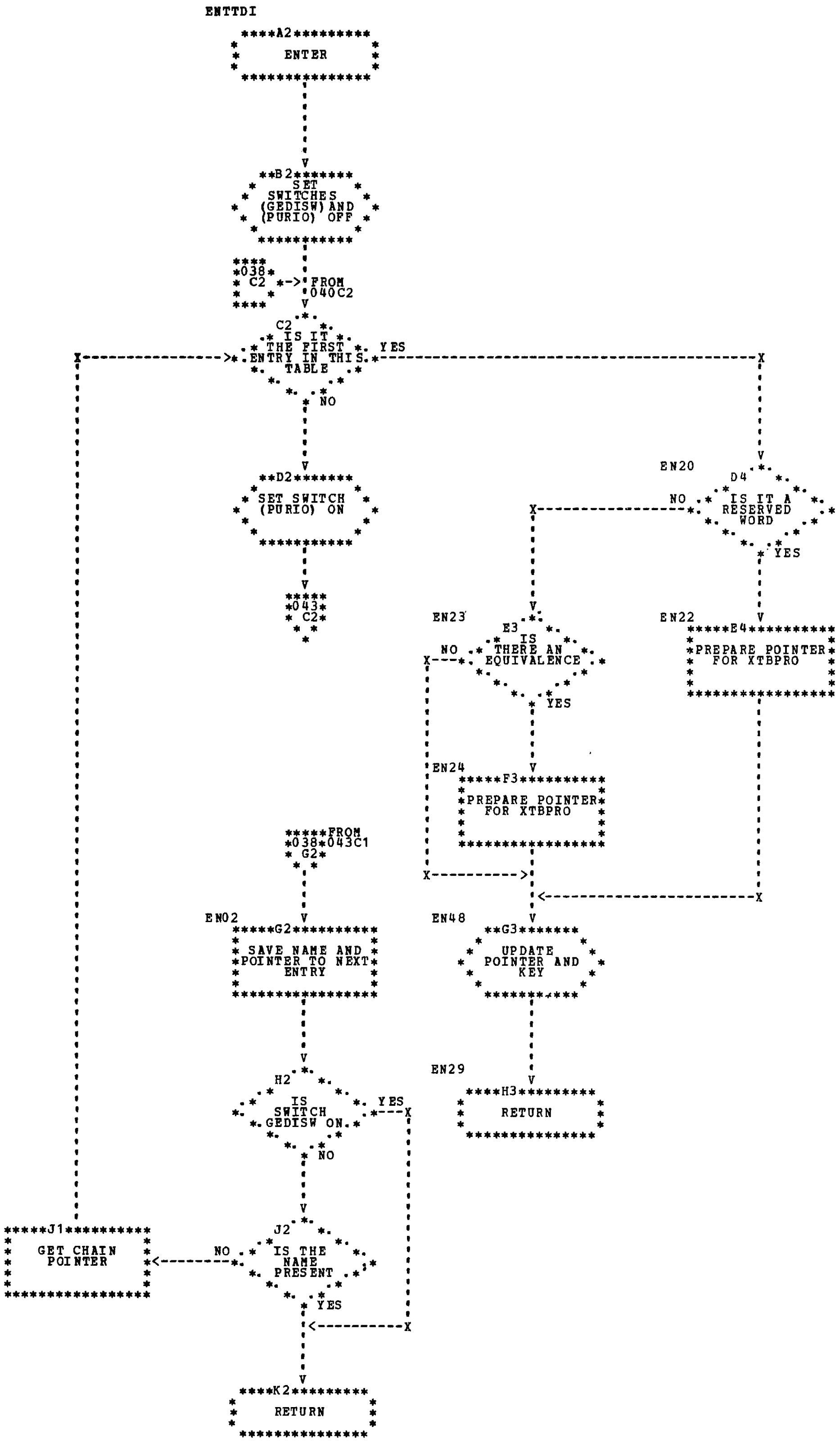


Chart 039. ERMS Routine

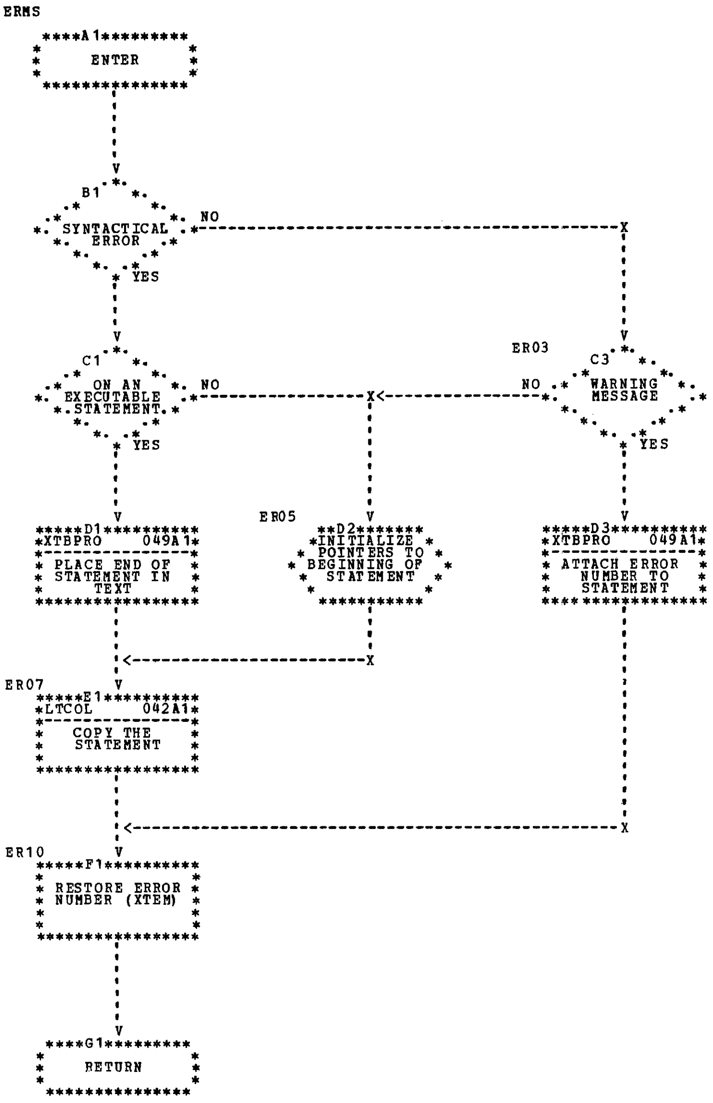

ERMS
A1
ENTER
B1
SYNTACTICAL ERROR
NO
YES
C1
ON AN EXECUTABLE STATEMENT
NO
YES
ER03
C3
WARNING MESSAGE
NO
YES
D1
XTBPRO 049A1
PLACE END OF STATEMENT IN TEXT
ER05
D2
INITIALIZE POINTERS TO BEGINNING OF STATEMENT
D3
XTBPRO 049A1
ATTACH ERROR NUMBER TO STATEMENT
ER07
E1
LTCOL 042A1
COPY THE STATEMENT
ER10
F1
RESTORE ERROR NUMBER (XTEM)
G1
RETURN

Chart 040. GETTBL Routine

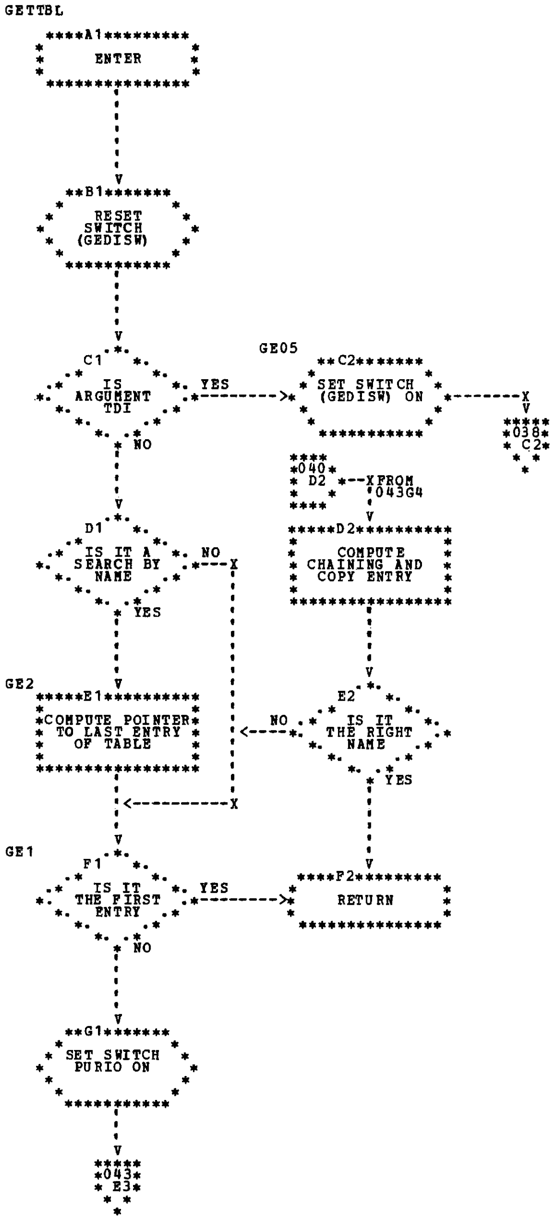


Chart 041. GETWRD Routine

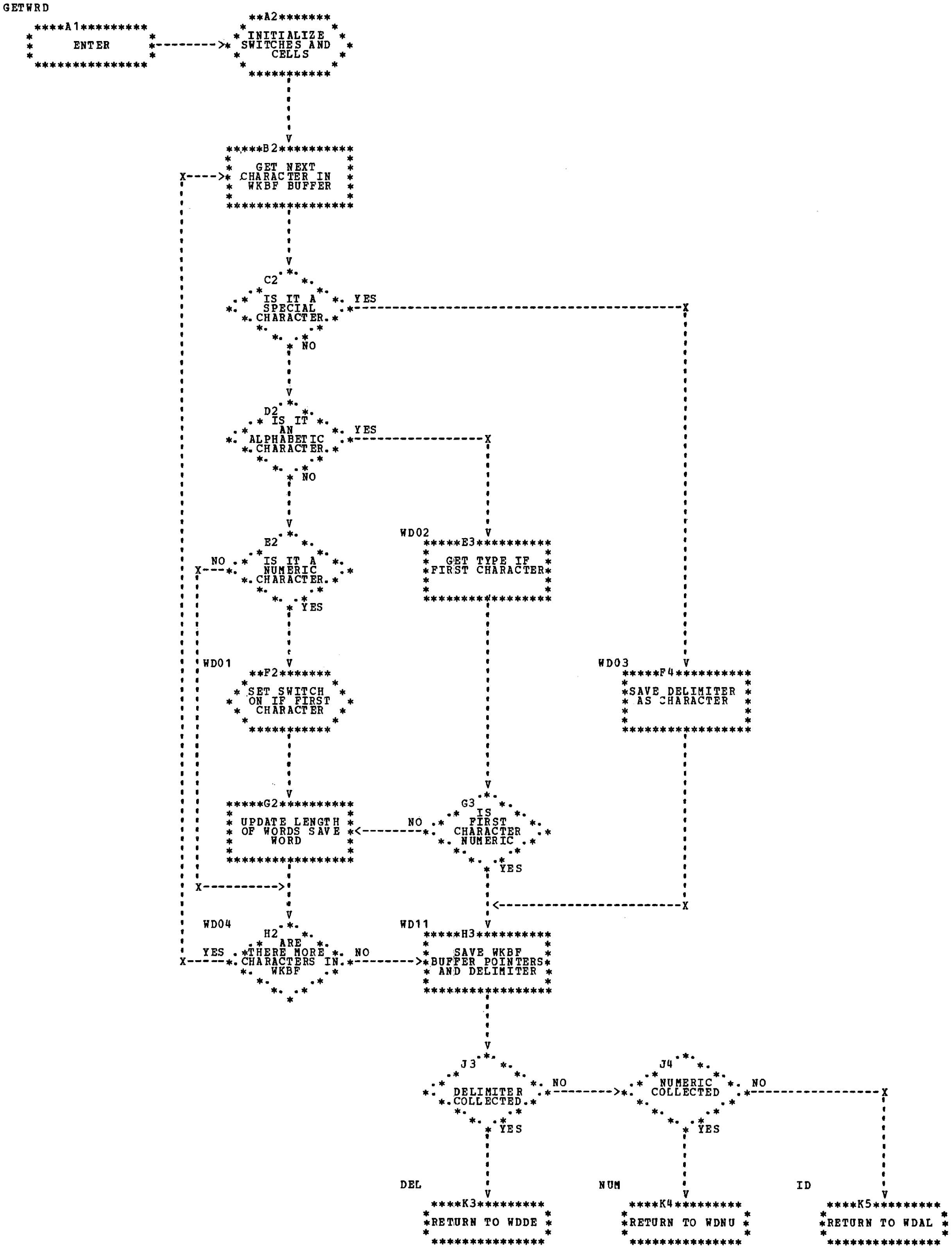
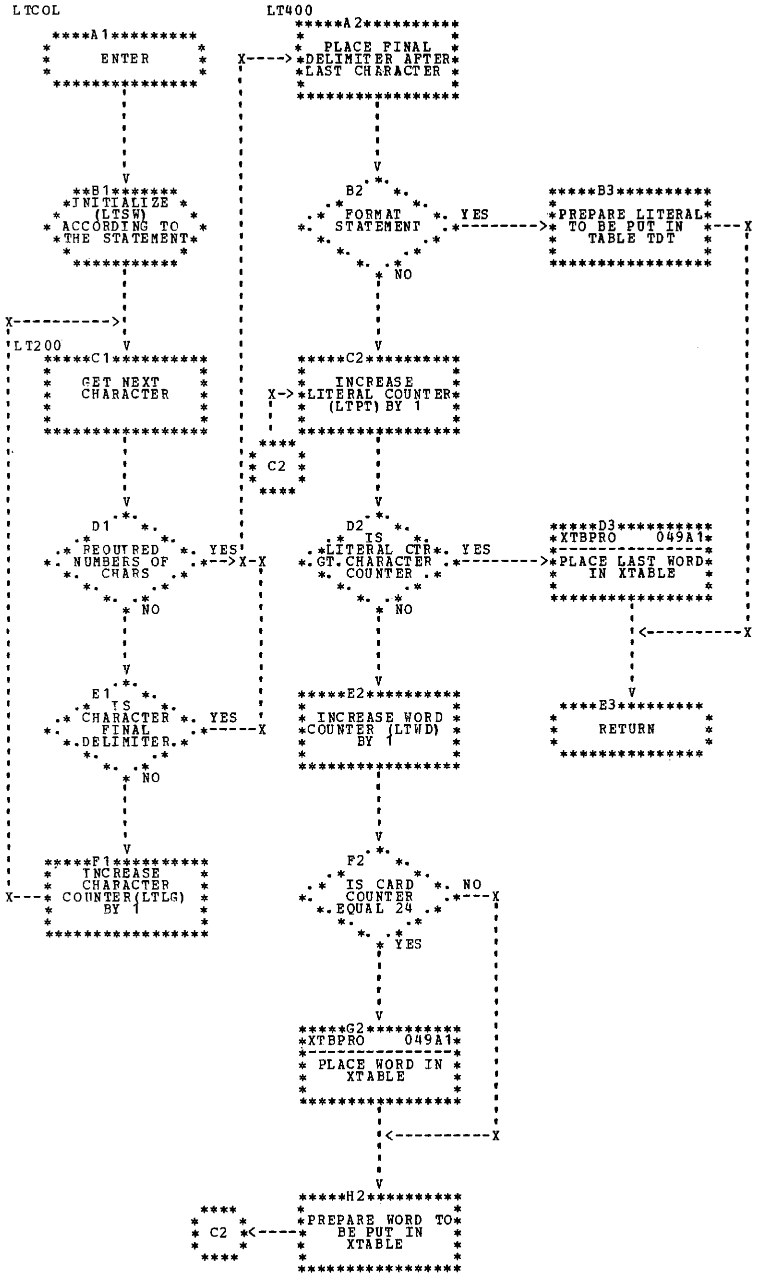

GETWRD
A1
ENTER
A2
INITIALIZE SWITCHES AND CELLS
B2
GET NEXT CHARACTER IN WKBF BUFFER
C2
IS IT A SPECIAL CHARACTER.
YES
NO
D2
IS IT AN ALPHABETIC CHARACTER.
YES
NO
E2
IS IT A NUMERIC CHARACTER.
NO
YES
WD02
E3
GET TYPE IF FIRST CHARACTER
WD01
F2
SET SWITCH ON IF FIRST CHARACTER
WD03
F4
SAVE DELIMITER AS CHARACTER
G2
UPDATE LENGTH OF WORDS SAVE WORD
G3
IS FIRST CHARACTER NUMERIC
NO
YES
WD04
H2
ARE THERE MORE CHARACTERS IN WKBF
YES
NO
WD11
H3
SAVE WKBF BUFFER POINTERS AND DELIMITER
J3
DELIMITER COLLECTED
NO
YES
J4
NUMERIC COLLECTED
NO
YES
DEL
K3
RETURN TO WDDE
NUM
K4
RETURN TO WDNU
ID
K5
RETURN TO WDAL

Chart 042. LTCOL Routine



LTCOL
A1
ENTER
B1
INITIALIZE (LTSW) ACCORDING TO THE STATEMENT
LT200
C1
GET NEXT CHARACTER
D1
REQUIRED NUMBERS OF CHARS
YES
NO
E1
IS CHARACTER FINAL DELIMITER.
YES
NO
F1
INCREASE CHARACTER COUNTER(LTLG) BY 1
LT400
A2
PLACE FINAL DELIMITER AFTER LAST CHARACTER
B2
FORMAT STATEMENT
YES
NO
C2
INCREASE LITERAL COUNTER (LTPT) BY 1
C2
D2
IS LITERAL CTR. GT. CHARACTER COUNTER
YES
NO
E2
INCREASE WORD COUNTER (LTWD) BY 1
F2
IS CARD COUNTER EQUAL 24
NO
YES
G2
XTBPRO 049A1
PLACE WORD IN XTABLE
H2
PREPARE WORD TO BE PUT IN XTABLE
C2
B3
PREPARE LITERAL TO BE PUT IN TABLE TDT
D3
XTBPRO 049A1
PLACE LAST WORD IN XTABLE
E3
RETURN

Chart 043. PUTTBL Routine

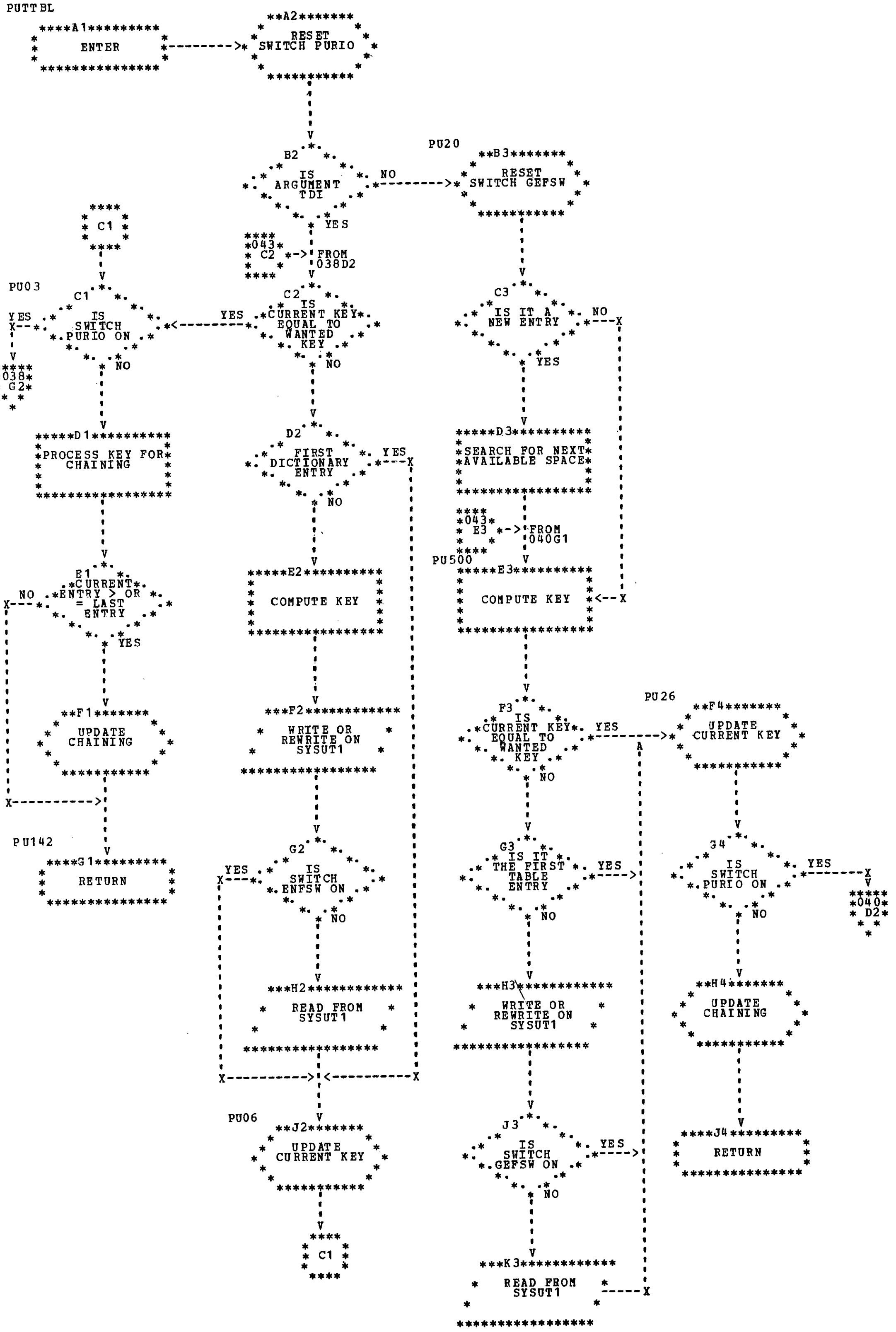


Chart 044. SLPAR Routine

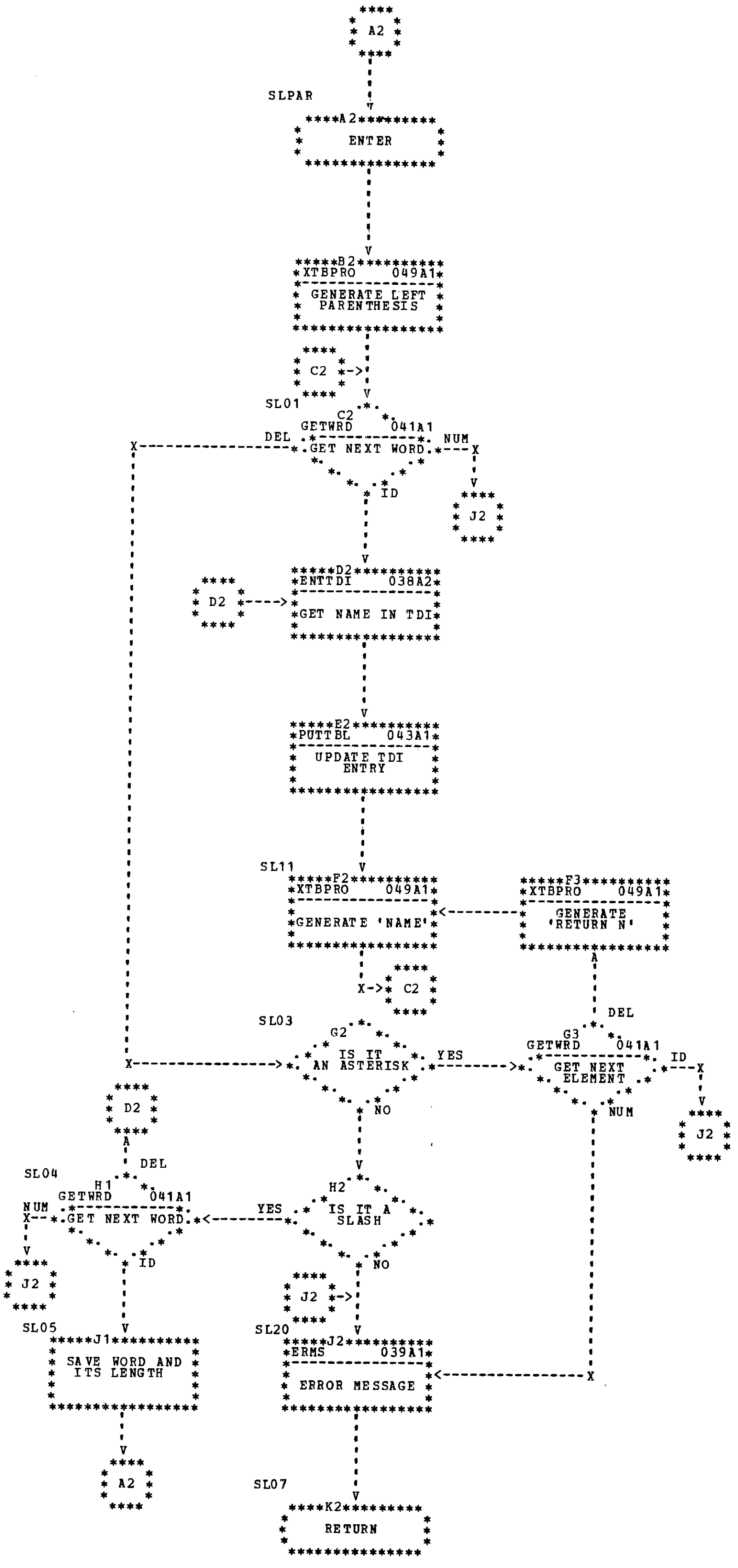


Chart 045. SPDIM Routine

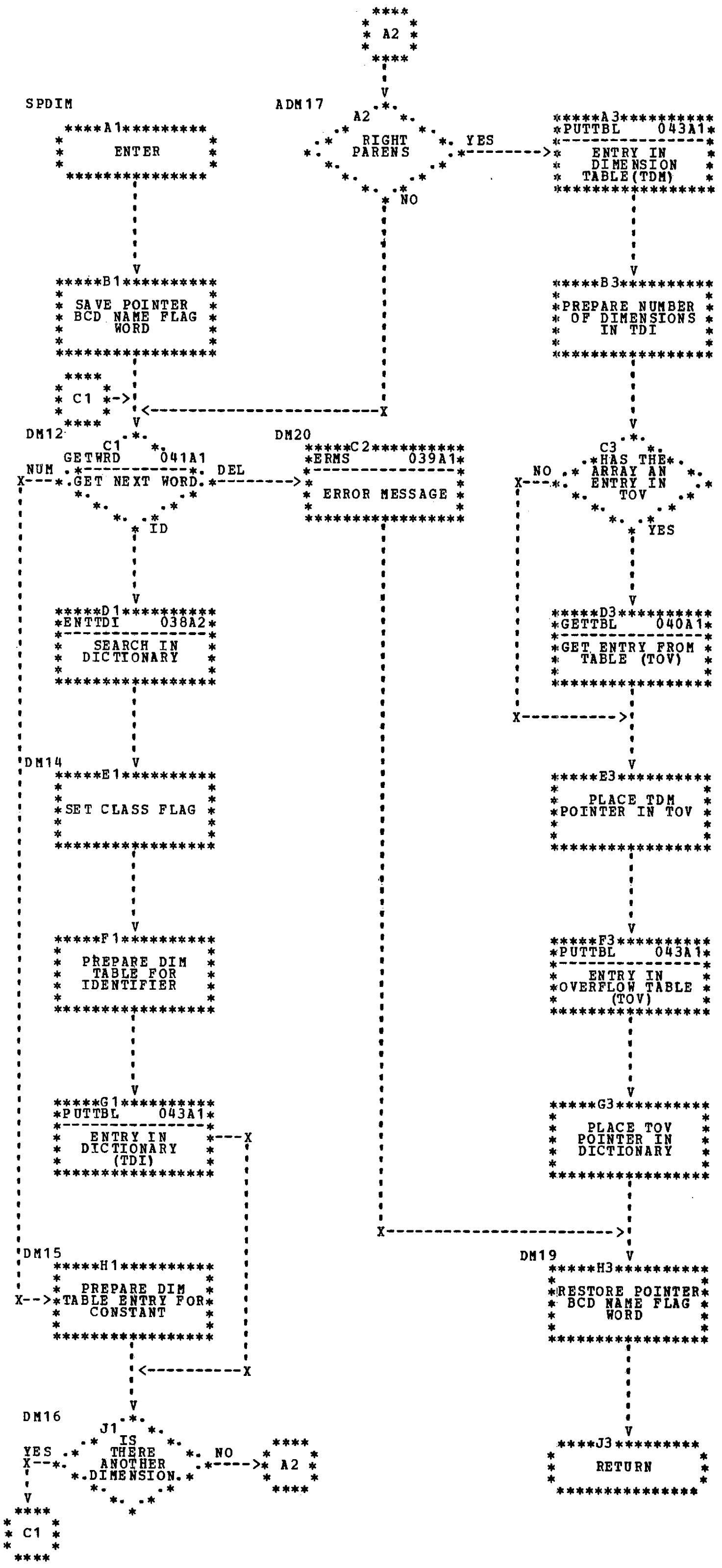

A2
SPDIM
A1
ENTER
ADM17
A2
RIGHT PARENS
YES
NO
A3
PUTTBL 043A1
ENTRY IN DIMENSION TABLE(TDM)
B1
SAVE POINTER BCD NAME FLAG WORD
B3
PREPARE NUMBER OF DIMENSIONS IN TDI
C1
DM12
C1
GETWRD 041A1
GET NEXT WORD.
NUM
DEL
ID
DM20
C2
ERMS 039A1
ERROR MESSAGE
C3
HAS THE ARRAY AN ENTRY IN TOV
NO
YES
D1
ENTTDI 038A2
SEARCH IN DICTIONARY
D3
GETTBL 040A1
GET ENTRY FROM TABLE (TOV)
DM14
E1
SET CLASS FLAG
E3
PLACE TDM POINTER IN TOV
F1
PREPARE DIM TABLE FOR IDENTIFIER
F3
PUTTBL 043A1
ENTRY IN OVERFLOW TABLE (TOV)
G1
PUTTBL 043A1
ENTRY IN DICTIONARY (TDI)
G3
PLACE TOV POINTER IN DICTIONARY
DM15
H1
PREPARE DIM TABLE ENTRY FOR CONSTANT
DM19
H3
RESTORE POINTER BCD NAME FLAG WORD
DM16
J1
IS THERE ANOTHER DIMENSION.
YES
NO
A2
C1
J3
RETURN

Chart 046. SPDTA Routine

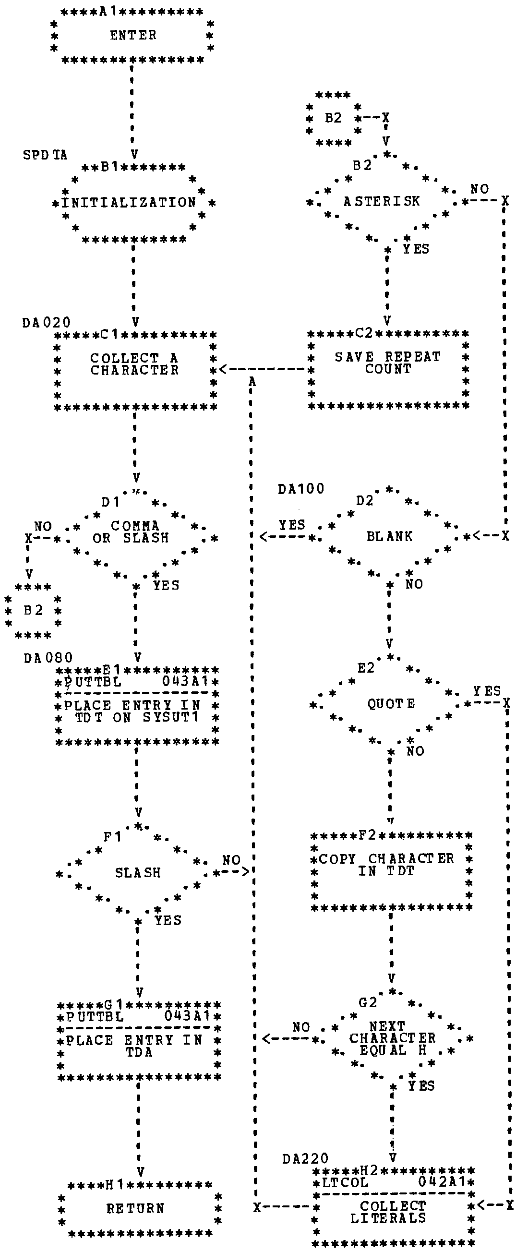


Chart 047. SPPRO Routine

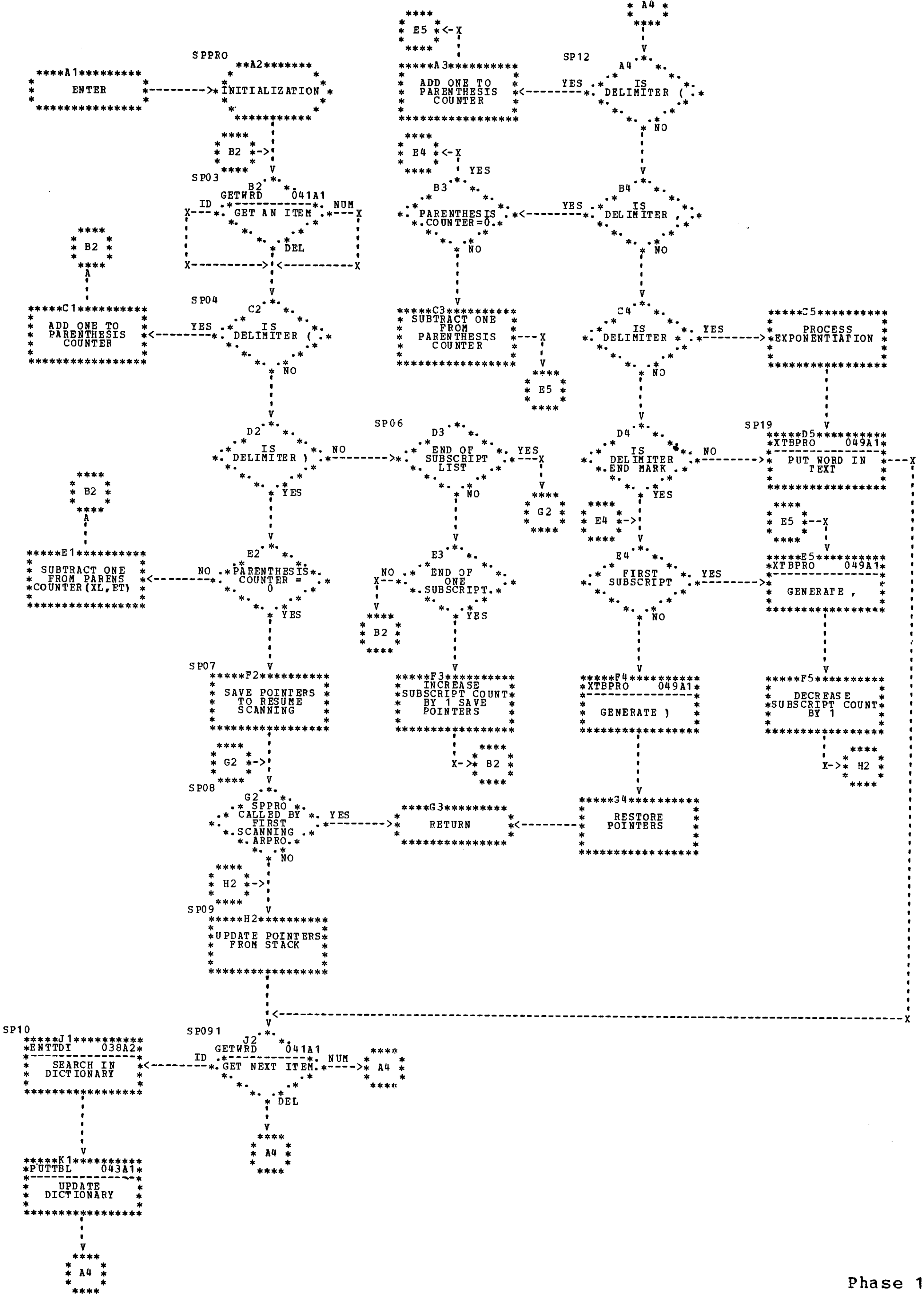


Chart 048. SPSUB Routine

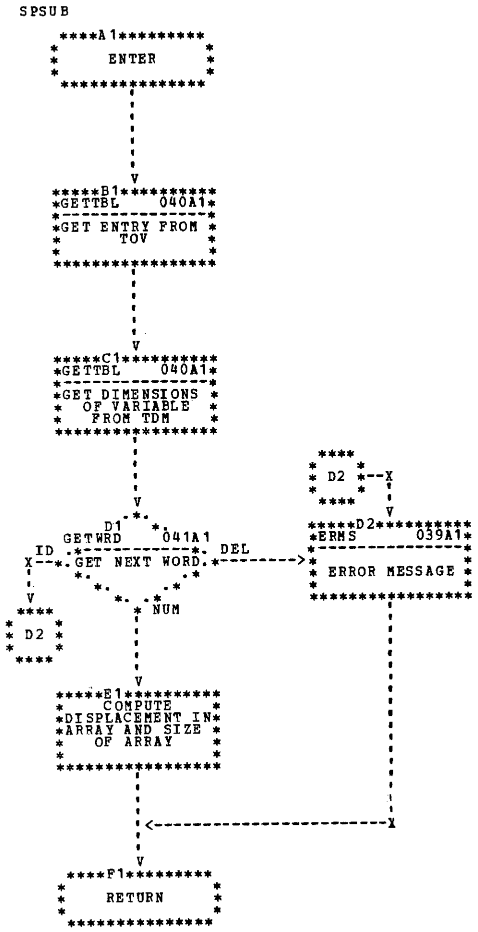
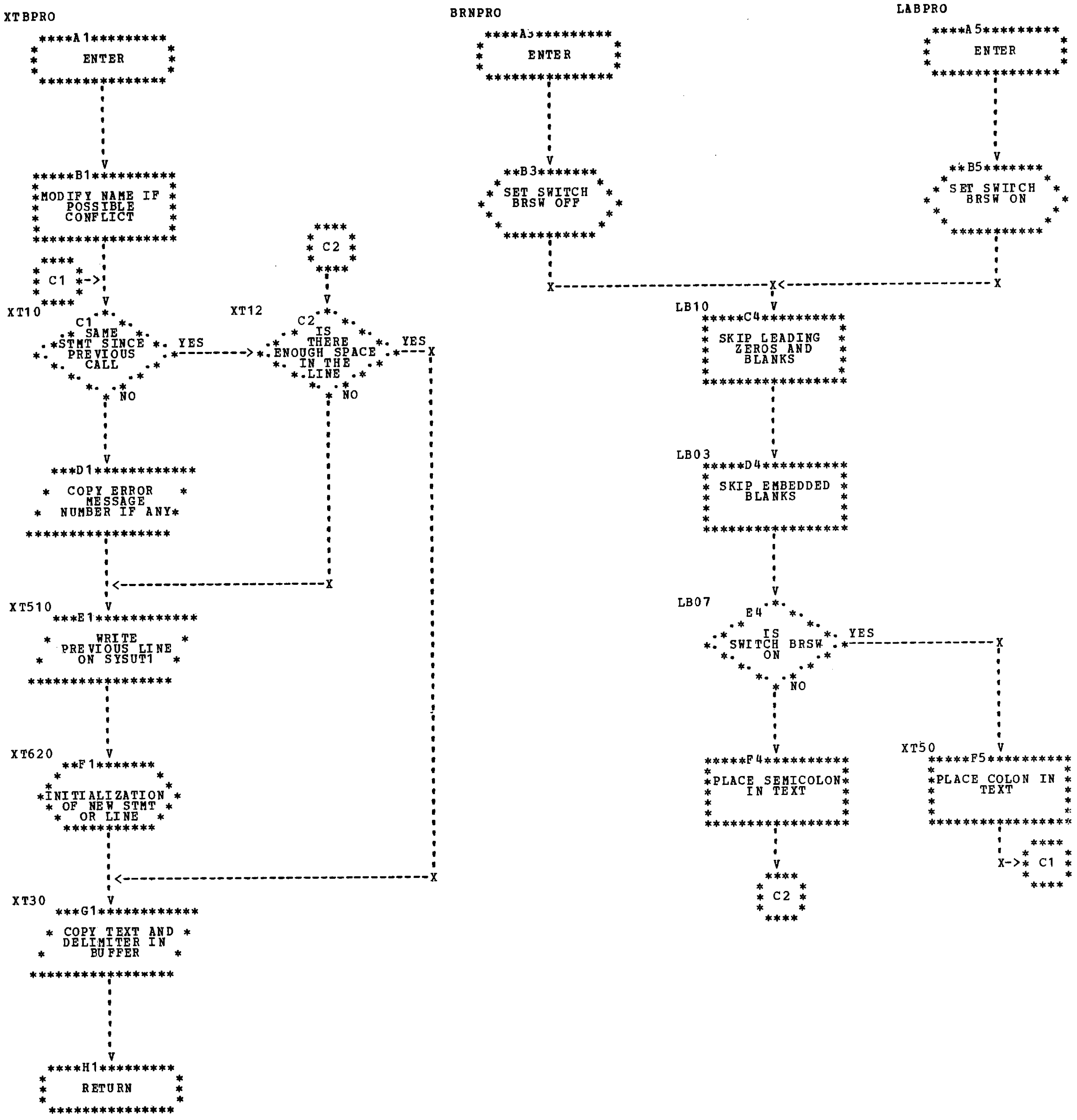


Chart 049. XTBPRO Routine



PHASE 20

Phase 20 (see Chart 050) is called if the source program includes either of the two following types of statements:

1. EQUIVALENCE statements

2. Statements containing initial values

that is, if there are any entries in tables TEQ or TDA.

Phase 20 then rearranges the tables created during Phase 10 that are associated with these statements, in order to prepare the generation of the PL/I target program in Phase 30.

The input to Phase 20 consists of the dictionary (TDI), and its associated tables: TEQ, TDT, and TDA.

Names occurring in EQUIVALENCE statements are processed first. An EQUBLKj block is created,

Embracing all the elements of the equivalenced names (see the section "EQUIVALENCE Statement" in the language conversion manual). The size, type, and length information associated with each element of the block is saved in table TEH, and a reference to this table for each of the equivalenced names is entered into the overflow table (TOV). To this reference is added the relative position, within the embracing block, of the first element of the name. This procedure is repeated for each equivalence chain in the FORTRAN program.

Phase 20 then places the initial values in ascending numerical order corresponding to the order of the elements. If any elements within the sequence have not been assigned, Phase 20 takes into account the number of elements between two consecutively initialized elements of an array.

If the initial value is a string of characters, a flag is set on in the TDT entry which contains the string and the type of dictionary entry for this name is modified to CHARACTER.

Where initialized variables or arrays occur in an EQUIVALENCE statement, the converted variables cannot have the INITIAL attribute because of the method of conversion used by the LCP.

The output from Phase 20 is a modified dictionary (TDI, TDT, TEH). On completion of this phase, control returns the Control Phase.

PHASE 20 ROUTINES

This section describes the two processing routines used by Phase 20: DTPRO and EVPRO. The two utility routines, GETTBL and PUTTBL, used by this phase have already been described in the section "Utility Routines."

DTPRO ______________________________ Chart 051

Purpose: To assign initial values to variables or to elements of arrays

Called by: PH20

Processing: The input to this routine consists of tables TDA and TDT. Variables included in DATA or TYPE statements are assigned a value determined by the literals appearing in these statements. The literals corresponding to each variable are chained together.

Routines called: GETTBL, PUTTBL.

Exit: Calling routine

EVPRO ______________________________ Chart 052

Purpose: To create EQUBLKs.

Called by: PH20

Processing: The input to this routine consists of tables TDI, TEQ, and TOV. All equivalenced variables are collected, and those which are included in the same EQUBLK are chained together. In addition, the size of each EQUBLK is computed, and the table TEH entry associated with each block is created. The chained variables are then collected and the overflow table (TOV) for each variable is updated.

Routines called: GETTBL, PUTTBL.

Exit: Calling routine.

Chart 050. Overall Logic of Phase 20

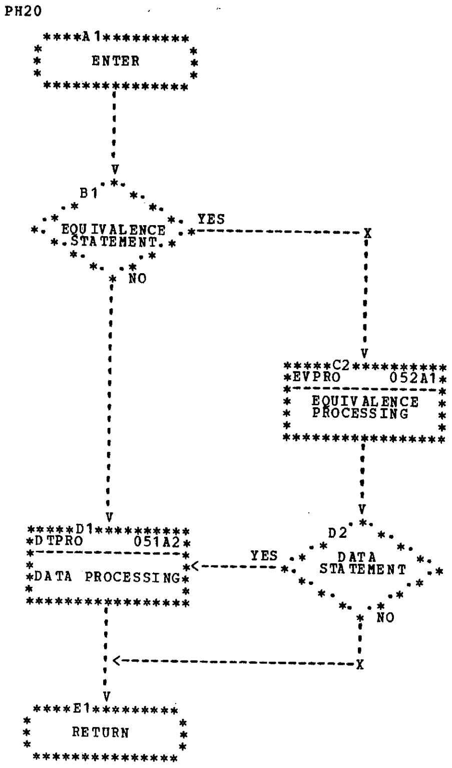
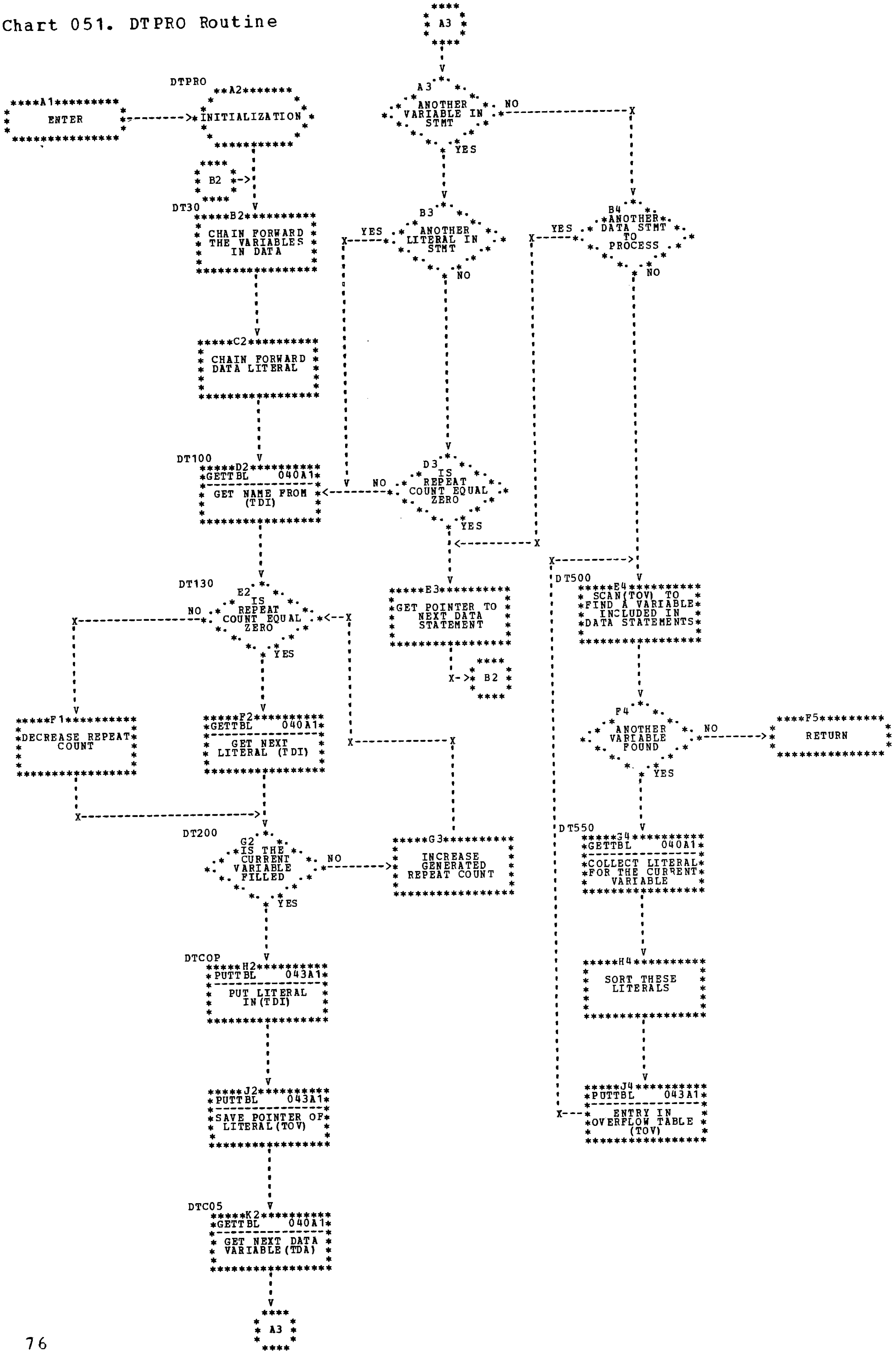
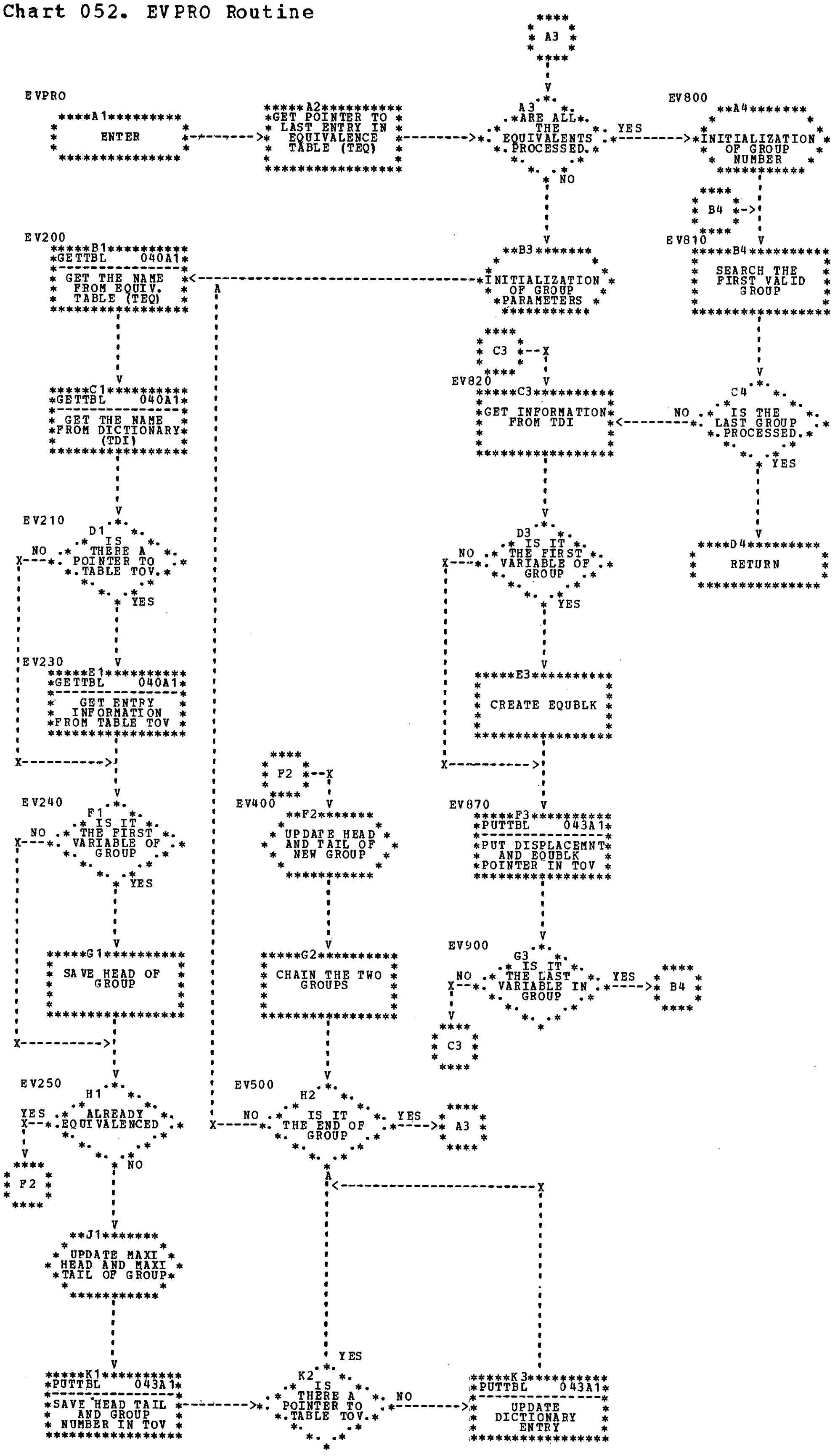


Chart 051. DTPRO Routine



A3
A1 ENTER
DTPRO A2 INITIALIZATION
B2
DT30 B2 CHAIN FORWARD THE VARIABLES IN DATA
C2 CHAIN FORWARD DATA LITERAL
DT100 D2 GETTBL 040A1 GET NAME FROM (TDI)
DT130 E2 IS REPEAT COUNT EQUAL ZERO
NO
YES
F1 DECREASE REPEAT COUNT
F2 GETTBL 040A1 GET NEXT LITERAL (TDI)
DT200 G2 IS THE CURRENT VARIABLE FILLED
NO
YES
G3 INCREASE GENERATED REPEAT COUNT
DTCOP H2 PUTTBL 043A1 PUT LITERAL IN(TDI)
J2 PUTTBL 043A1 SAVE POINTER OF LITERAL(TOV)
DTC05 K2 GETTBL 040A1 GET NEXT DATA VARIABLE (TDA)
A3
A3 ANOTHER VARIABLE IN STMT
NO
YES
B3 ANOTHER LITERAL IN STMT
YES
NO
D3 IS REPEAT COUNT EQUAL ZERO
NO
YES
E3 GET POINTER TO NEXT DATA STATEMENT
B2
B4 ANOTHER DATA STMT TO PROCESS
YES
NO
DT500 E4 SCAN(TOV) TO FIND A VARIABLE INCLUDED IN DATA STATEMENTS
F4 ANOTHER VARIABLE FOUND
NO
YES
F5 RETURN
DT550 G4 GETTBL 040A1 COLLECT LITERAL FOR THE CURRENT VARIABLE
H4 SORT THESE LITERALS
J4 PUTTBL 043A1 ENTRY IN OVERFLOW TABLE (TOV)

Chart 052. EVPRO Routine


A3
EVPRO
A1 ENTER
A2 GET POINTER TO LAST ENTRY IN EQUIVALENCE TABLE (TEQ)
A3 ARE ALL THE EQUIVALENTS PROCESSED.
YES
NO
EV800
A4 INITIALIZATION OF GROUP NUMBER
B4
EV200
B1 GETTBL 040A1
GET THE NAME FROM EQUIV. TABLE (TEQ)
B3 INITIALIZATION OF GROUP PARAMETERS
EV810
B4 SEARCH THE FIRST VALID GROUP
C3
EV820
C1 GETTBL 040A1
GET THE NAME FROM DICTIONARY (TDI)
C3 GET INFORMATION FROM TDI
C4 IS THE LAST GROUP PROCESSED.
NO
YES
EV210
D1 IS THERE A POINTER TO TABLE TOV.
NO
YES
D3 IS IT THE FIRST VARIABLE OF GROUP
NO
YES
D4 RETURN
EV230
E1 GETTBL 040A1
GET ENTRY INFORMATION FROM TABLE TOV
E3 CREATE EQUBLK
F2
EV240
F1 IS IT THE FIRST VARIABLE OF GROUP
NO
YES
EV400
F2 UPDATE HEAD AND TAIL OF NEW GROUP
EV870
F3 PUTTBL 043A1
PUT DISPLACEMNT AND EQUBLK POINTER IN TOV
G1 SAVE HEAD OF GROUP
G2 CHAIN THE TWO GROUPS
EV900
G3 IS IT THE LAST VARIABLE IN GROUP
NO
YES
B4
C3
EV250
H1 ALREADY EQUIVALENCED
YES
NO
F2
EV500
H2 IS IT THE END OF GROUP
NO
YES
A3
J1 UPDATE MAXI HEAD AND MAXI TAIL OF GROUP
K1 PUTTBL 043A1
SAVE HEAD TAIL AND GROUP NUMBER IN TOV
K2 IS THERE A POINTER TO TABLE TOV.
YES
NO
K3 PUTTBL 043A1
UPDATE DICTIONARY ENTRY

Phase 30 generates the PL/I target program, and all comments and messages. It uses two utility routines: UPRNT, which is described in detail later in this section, and GETTBL.

This phase draws its input from SYSUT1 and from the dictionary (TDI) and the tables associated with it.

After initialization of all phase parameters , Phase 30 generates a PL/I PROCEDURE statement. The label of this statement is either that of a FORTRAN subprogram or the LCP name MAINPRO in the case of a main program.

Phase 30 then processes identifiers that appear in COMMON statements. The elements of a labeled or unlabeled block, together with their attributes, are declared as minor structures at level 2. The major structure is declared EXTERNAL; it is identified by the COMMON block name (TBK) if the block was labeled by the programmer; if not, it is identified by the LCP name UNLABCM.

All blocks created in Phase 20 are then placed in a DECLARE statement; these embrace EQUIVALENCE blocks when they do not include a variable which is part of a COMMON block.

Next, Phase 30 examines the dictionary, generating DECLARE statements for all identifiers that have not appeared in COMMON statements and which must be explicitly declared: variables of a specified type, array names, procedure names, labels, equivalenced items, and initialized elements.

The equivalenced items are declared in two-level structures, where the first element of level-2 is a dummy variable giving the displacement of the variable in front of the EQUIVALENCE block (or the COMMON block if this item is equivalenced with a variable appearing in COMMON) and the second element of level-2 is the equivalenced item.

- When the type of a variable is found to be CHARACTER in the dictionary (TDI) entry, the variable is declared with the CHARACTER attribute.
- If a TDU table exists, each LCP-created variable is declared with the CHARACTER attribute. Then, the string of characters saved in table TDT and associated with it is placed after the INITIAL attribute.
- Print files are declared according to the information saved by the LCP in the initialization procedure.

Executable statements are generated next; they are checked for being a PL/I internal procedure (converted statement function). When one is encountered, the attributes of each parameter are generated in a DECLARE statement, using pointers in PL/I text and dictionary.

Finally, the error messages, if any, are printed on SYSPRINT.

The output from Phase 30 is in the form specified by the user in his EXEC control card, either in printed form on SYSPRNT or in punched-card form on SYSPCH.

On completion of Phase 30, control returns to the Control Phase.

The overall flow of Phase 30 is illustrated in Charts 053 and 054.

PHASE 30 ROUTINES

This section describes the utility routine UPRNT, which is used by Phase 30 only. Phase 30 also makes use of the utility routine GETTBL, which is described in the section " Utility Routines."

UPRNT Chart 055

Purpose: To generate declarative statements.

Called by: PH30

Processing: The input to this routine is a character-string in the area WORD. A line of buffer is initialized. Each time a line in the buffer is full, this line is placed on SYSPRINT and the next line is initialized. The output from this routine is the PL/I statement in the buffer.

Routines called: None.

Exit: Calling routine.

Chart 053. Overall Logic of Phase 30, Part 1 of 2

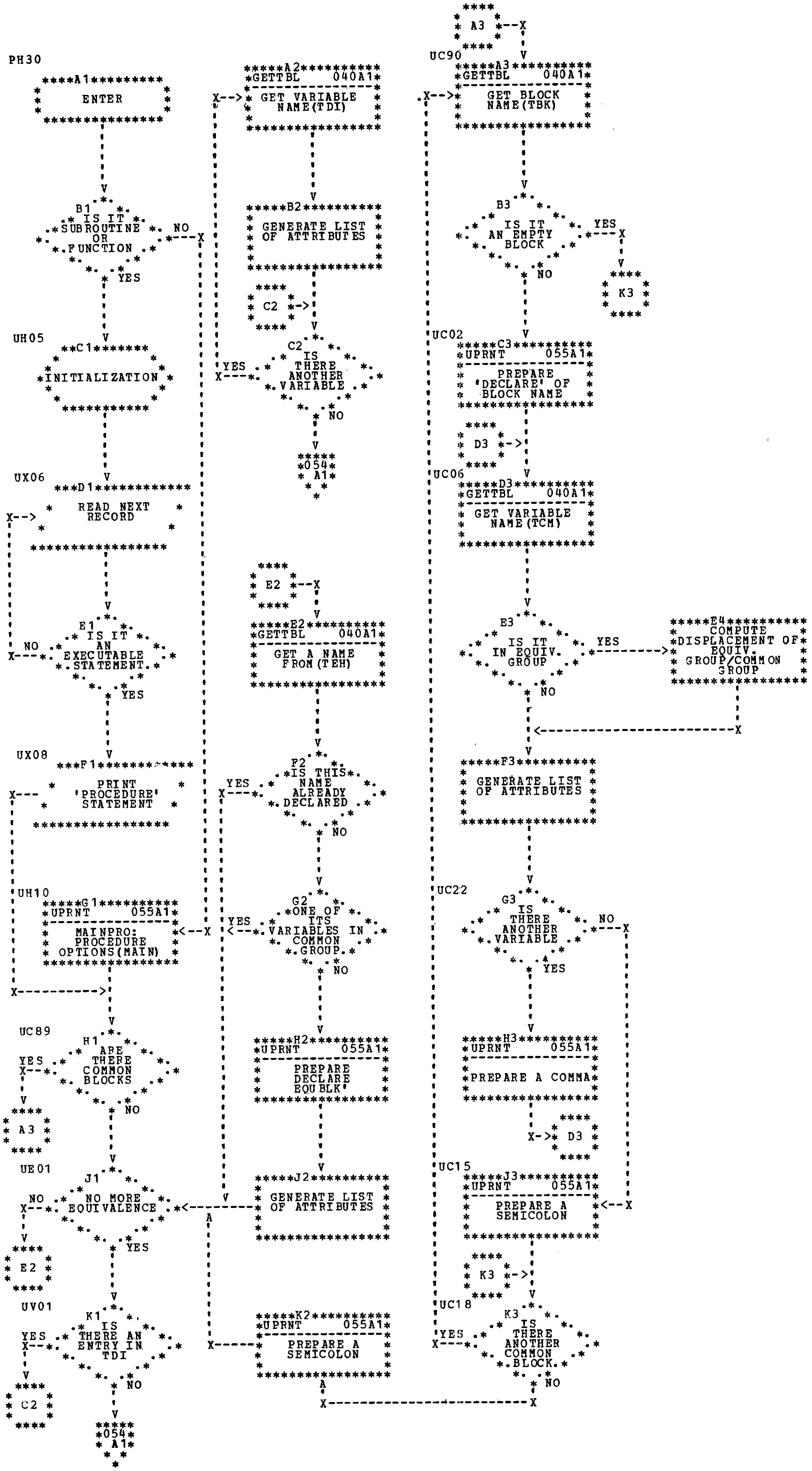


Chart 054. Overall Logic of Phase 30, Part 2 of 2

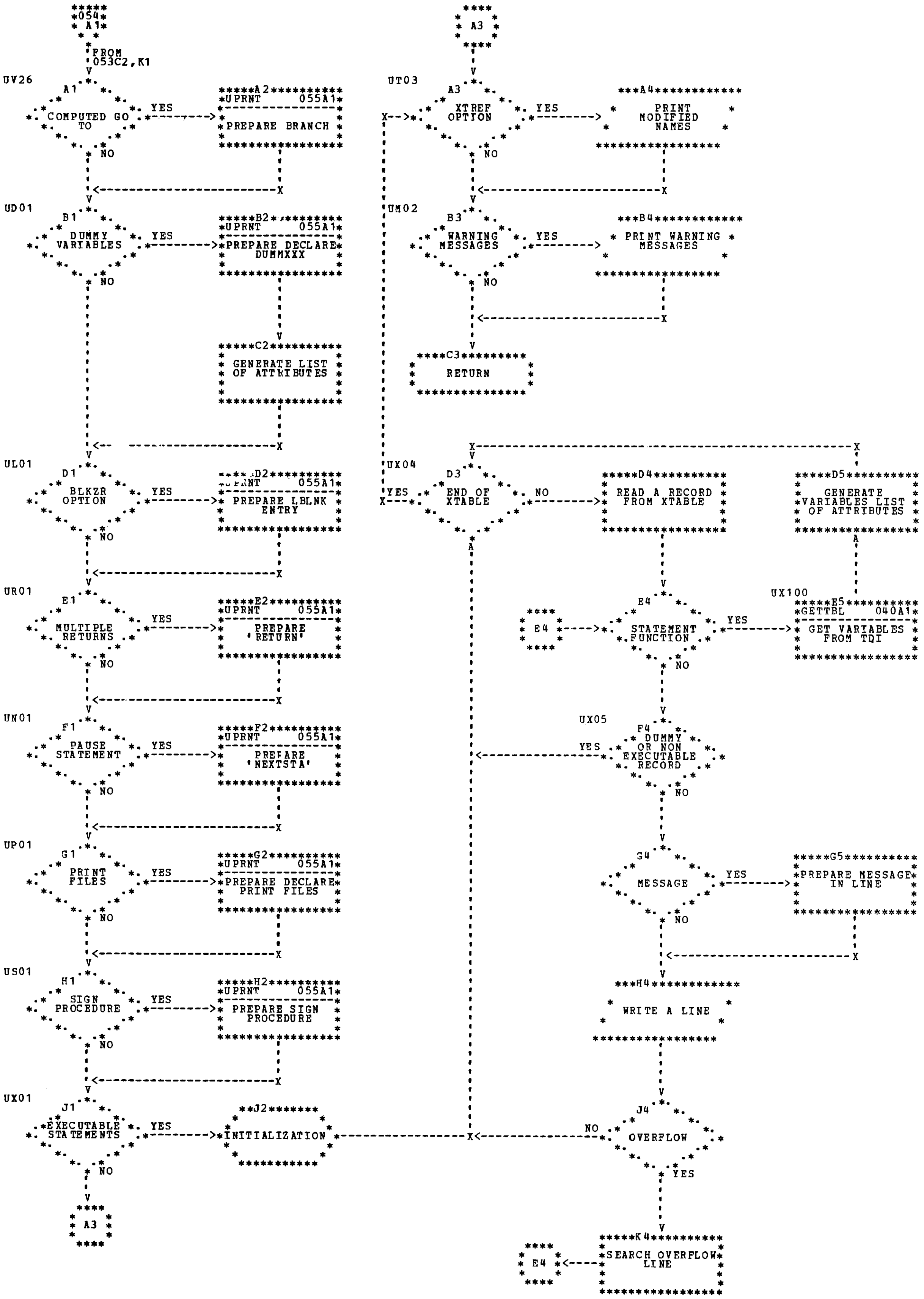
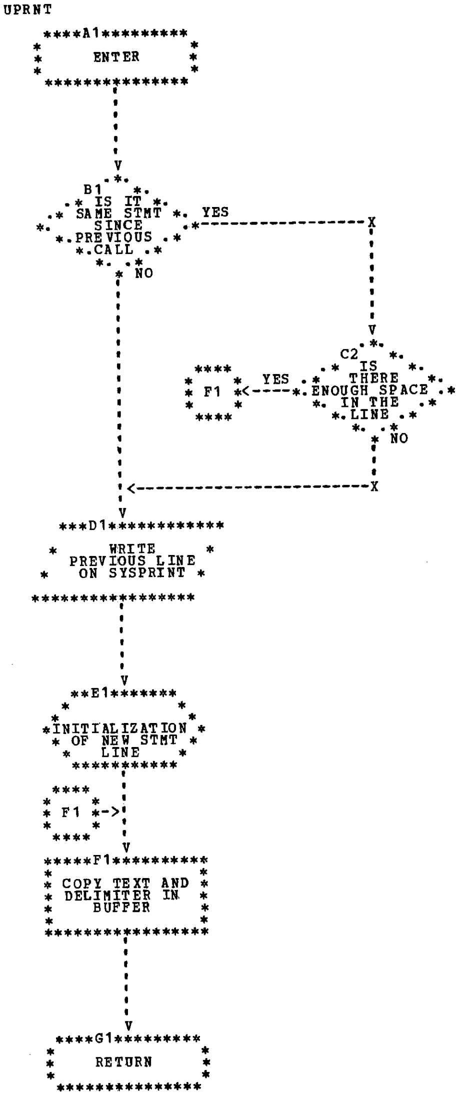


Chart 055. UPRNT Routine


UPRNT
A1
ENTER
B1
IS IT SAME STMT SINCE PREVIOUS CALL
YES
NO
C2
IS THERE ENOUGH SPACE IN THE LINE
YES
NO
F1
D1
WRITE PREVIOUS LINE ON SYSPRINT
E1
INITIALIZATION OF NEW STMT LINE
F1
F1
COPY TEXT AND DELIMITER IN BUFFER
G1
RETURN

APPENDIX A. TABLES

This appendix lists and describes all the tables used by the LCP.

GENERAL CONSIDERATIONS

Required information is fetched from FORTRAN declarative statements and stored in various tables. This information is then used in the conversion of executable statements. Certain elements of the executable statements are also stored in tables for the same purpose. In Phase 30, the PL/I DECLARF statements are generated using these tables. The length of each element of a table is shown in the listing.

All entries for the tables are placed in an area of main storage; entries within the same table are chained. When the area of main storage assigned to these entries is full, the contents of that area are placed on SYSUT2, and the area is then used for new entries. There is therefore no limitation in the size of these tables.

The tables used by the LCP fall into one of two categories:

1. The dictionary (Table TDI).
2. Specification tables. This category includes all the other tables used by the program, namely: Tables TBK, TCM, TDA, TDM, TDT, TDU, TEH, TEQ, TFM, TIM, TNL, TNV, TOV, TPD.

THE DICTIONARY (TABLE TDI)

The dictionary contains all the names used in the LCP and the class, type, and number of dimensions associated with each name. Names with the same number of characters are chained together.

Table Overflow: If the name has dimensions, or initial values, or appears in an EQUIVALENCE statement, the entry in the dictionary is extended by creating an entry in the overflow table (TOV). A pointer to this entry is then placed with the name in the dictionary. A description of Table TOV appears in the section "Specification Tables."

The layout of each entry in the dictionary is as follows:

DINAM	BCD Name
DILT	Number of Characters
DICL	Class
DITY	Type
DIET	Explicit Type
DILG	Length
DIAU	Storage Class
DIMO	Modify
DIAR	Array
DIOV	Overflow

DINAM
contains the symbol given to the name, or the sequence number if the name is generated.

DILT
contains the number of characters in the name.

DICL
contains one of the following constants:

LCVAR if DINAM is a variable
LCLAB if it is a label
LCENT if it is an entry name

DITY
contains one of the following constants:

LCTI if the type of DINAM is integer
LCTR if it is real
LCTC if it is complex
LCTL if it is logical
LCTA if it is a character string

DIET
contains the constant LCET when DINAM is explicitly declared.

DILG
contains, according to the length in bytes of DINAM:

LCLO for a length of 1 byte
LCLT for a length of 2 bytes
LCLF for a length of 4 bytes
LCLE for a length of 8 bytes
LCLS for a length of 16 bytes

DIMO
contains one of the following constants:

LCSUF if DINAM is variable or array name to which a suffix must be added.
LCEQU if DINAM is a reserved word which requires a PL/I equivalent.
Zero Otherwise

DIAU
contains constant LCAU if DINAM is used as a parameter.

DIAR
contains the number of dimensions if DINAM is an array name.

DIOV
contains the overflow table pointer if DINAM appears in a DIMENSION, DATA, or EQUIVALENCE statement.

SPECIFICATION TABLES

This section describes the layout of the specification tables used by the LCP, which appear in alphabetical order.

Table TBK (Common Block Table)

TBKN	Name
TBKP	Pointer to TCM

TBKN
contains the common block name.

TBKP
contains the pointer to the entry in table TCM.

Table TCM (Common Variable Table)

TCMC	Next Entry
TCMP	Pointer to TDI

TCMC
contains the pointer to the next entry for this block.

TCMP
contains the pointer to the dictionary entry for the variable.

Table TDA (Table of Variables with Initial Values)

TDAP	Pointer to TDI
TDAH	Head Value
TDAN	Next Entry

TDAP
contains the pointer to the dictionary entry.

TDAH
contains the head value in the EQUBLK including the variable.

TDAN
contains the pointer to the next entry in table TDA.

Table TDM (Dimension Table)

TDMN	Number of dimensions
TDMT	Constant or Pointer
TDMV	Value of Pointer

TDMN
number of subscripts in the array.

TDMT
contains one of the following constants:

TDMC if TDMV is a constant
TDMP if TDMV is a pointer to a dictionary entry

TDMV
contains the value of the pointer or of the constant.

Table TDU (Dummy Variable Table)

TDUL	Length of the literal
TDUI	Pointer to TDT table
TDUN	Dummy variable number

TDUL
contains the length of the literal which is part of a FORMAT statement in table TDT

TDUI
contains the pointer to table TDT

TDUN
contains the number (3 characters) of the dummy variable created for the literal (DUMMxxx)

Table TDT (Table of Initial Values)

TDTN	Next Entry
TDTL	Length of Literal
TDTR	Repetition Factor
TDTH	Position of Element
TDTT	Type of Data
TDTC	Character-String

TDTN
contains the pointer to the next entry in table TDT.

TDTL
contains the number of characters in the string.

TDTR
contains the repetition factor.

TDTH
contains the value of the position, within the array, of the element to be initialized.

TDTT
indicates whether TDTC is numeric (set to 0) or alphameric (set to 1).

TDTC
contains the character-string collected.

Table TEH (Table of EQUBLK's)

TEHN	EQUBLK Number
TEHT	Type
TEHE	Explicit Type
TEHL	Length
TEHM	Size
TEHA	Displacement EQUBLK/COMMON block (if necessary)

TEHN
contains the sequence number of the EQUBLK.

TEHT
contains one of the following constants:

LCTI if the type of the EQUBLK is
    integer
LCTR if it is real
LCTC if it is complex
LCTL if it is logical

TEHE
contains the constant LCET if the length or type of the EQUBLK must be explicitly declared.

TEHL
contains, according to the length in bytes of each element of the EQUBLK:

LCLO for a length of 1 byte
LCLT for a length of 2 bytes
LCLF for a length of 4 bytes
LCLE for a length of 8 bytes
LCLS for a length of 16 bytes

TEHM
contains the size of the EQUBLK.

TEHA
contains the displacement of EQUBLK from the COMMON block if an item of the equivalence chain appears in COMMON.

Table TEQ (Equivalence Table)

TEQP	Pointer to TDI
TEQH	Head
TEQT	Tail
TEQN	Next Element

TEQP
contains the pointer to the dictionary entry.

TEQH
contains the head value in EQUBLK.

TEQT
contains the tail value in EQUBLK.

TEQN
contains the chain pointer to the next element in the equivalence group.

Table TFM (Format table)

TFMF	Pointer to TDT table
TFML	Format label
TFMR	Print File indicator

TFMF
    contains the pointer to table TDT (packed FORMAT)

TFML
    contains the label of the FORTRAN FORMAT statement

TFMR
    indicates if the format is used with a PRINT or non-PRINT file or both.

Table TIM (Implicit Statement Table)

This table contains 256 entries, one for each of the possible bit configurations accepted by the System/360.

TIMT	Type of Symbol
TIML	Length of Type
TIME	Standard or not

TIMT
    contains one of the following constants:

    LCTI if the type of symbol is integer
    LCTR if it is real
    LCTC if it is complex
    LCTL if it is logical
    Zero if the first character of the symbol is not alphabetical

TIML
    contains one of the following constants:

    LCLO for a length of 1 byte
    LCLT for a length of 2 bytes
    LCLF for a length of 4 bytes
    LCLE for a length of 8 bytes
    LCLS for a length of 16 bytes
    Zero if the first character of the symbol is not alphabetical

TIME
    contains one of the following constants:

    LCET if either TIMT or TIML are not standard
    Zero if they are

Table TNL (NAMELIST Table)

TNLN	Name
TNLP	Pointer to TNV

TNLN
    contains the NAMELIST name.

TNLP
    contains the pointer to table TNV.

Table TNV (List of Variable Table)

TNVN	Name
TNVP	Next Entry

TNVN
    contains the variable name pointer.

TNVP
    contains the pointer to the next entry under this NAMELIST name.

Table TOV (Overflow Table)

TOVD	Pointer to TDM
TOVE	Pointer to TEH
TOVP	Displacement
TOVT	Indicator
TOVI	Pointer to TDT

TOVD
    contains the pointer to the entry in table TDM (array).

TOVE
    contains the pointer to the entry in table TEH for the EQUBLK containing the variable (equivalenced item).

TOVP
    contains the displacement.

TOVT
is used in Phase 20 to indicate whether or not the variable has already been placed in an EQUBLK.

TOVI
contains the pointer to the entry in table TDT.

Table TPD (DO Loop Label Table)

This is a push-down table which is built up by the GETCRD routine. When GETCRD returns control to Phase 10, this table is empty and no longer used.

TPDL	End of DO Loop Label

TPDL
contains the label of the end of the DO loop.

The LCP provides a subroutine (LBLNK) which is invoked at PL/I target program execution time to process the external form of numeric input data.

If the option BLKZR has been specified at conversion time, LBLNK is generated to convert E-, F-, and I-format items which refer to at least one non-PRINT file. The lenqth of the data field is passed as an arqument. LBLNK makes the following alterations during the transmission of external data:

- plus sign in BCDIC (&) is replaced by +
- D is changed to E
- An all blank field is replaced by 0
- If a blank appears after E, it is replaced by +
- other embedded and trailing blanks are replaced by zeroes.

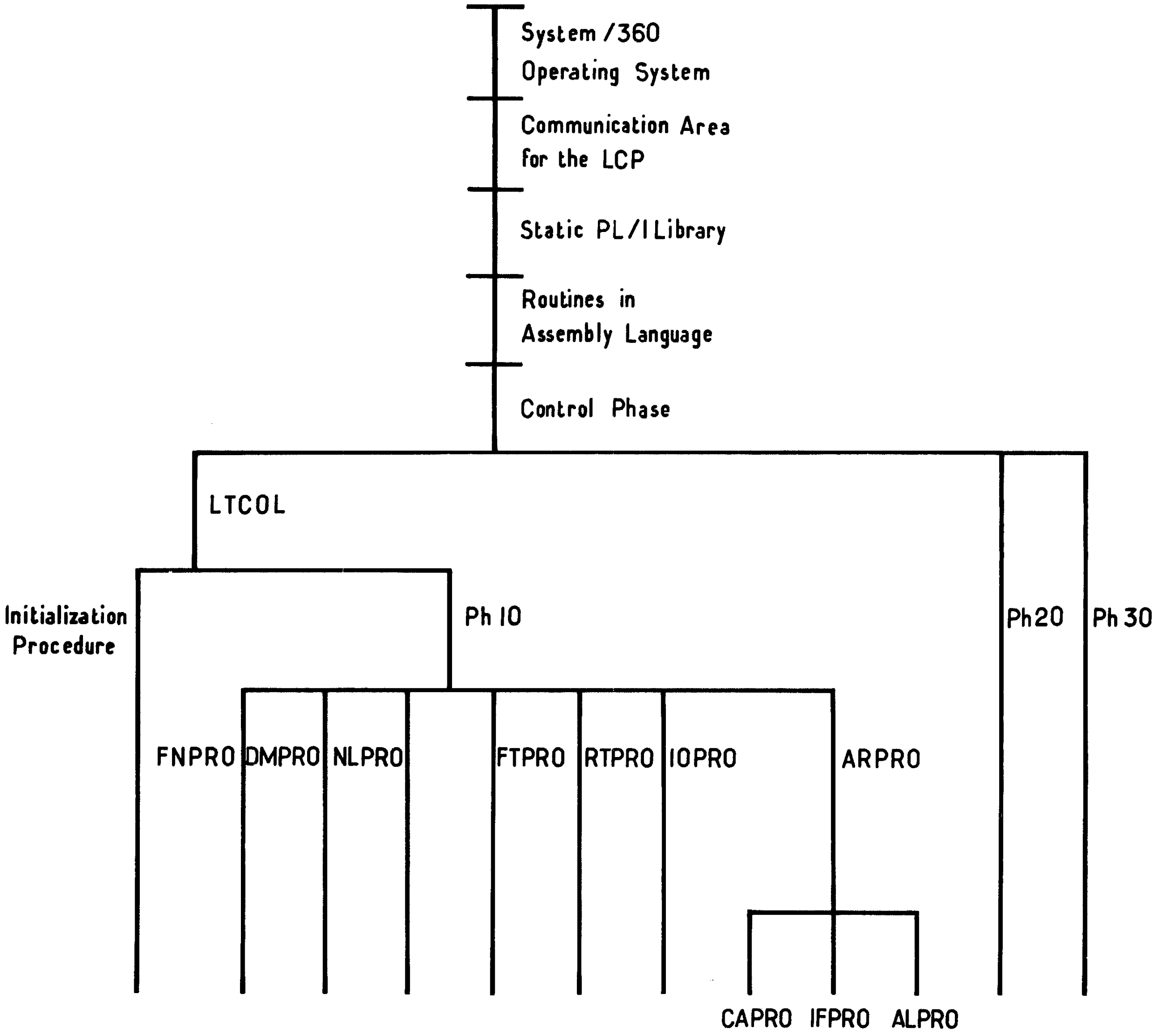
APPENDIX C. STORAGE MAP

The following diagram illustrates the way in which the LCP uses main storage.

The routines PUTTBL, GETTBL, ENTTDI and CONVER are written in Assembly Language.

The various tables are in the Communication Area.

The exact sizes of the components of the program are not shown since they depend on the options used to compile the LCP.



INDEX

Where more than one reference is given, the first page number indicates the major reference.

READER'S COMMENT FORM

Y33-7000-0

IBM System/360 Conversion Aids:
FORTRAN IV-to-PL/I Language Conversion Program
for IBM System/360 Operating System

- How did you use this publication?

  As a reference source .... □
  As a classroom text .... □
  As a self-study text .... □

- Based on your own experience, rate this publication . . .

As a reference source:	....	....	....	....	....
	Very Good	Good	Fair	Poor	Very Poor
As a text:	....	....	....	....	....
	Very Good	Good	Fair	Poor	Very Poor

- What is your occupation? ....

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . .

This SRL bulletin is one of a series which serves as reference sources for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold

Fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Department 813

Fold

Fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Y33-7000-0

Printed in U.S.A. Y33-7000-0

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]