File No. S360-25
Form GY28-6831-2

Program Logic

IBM System/360 Operating System

FORTRAN IV Syntax Checker

Program Number: 360S-FO-550

This publication describes the internal logic of the FORTRAN IV Syntax Checker, which is a component of the IBM System/360 Operating System Conversational Remote Job Entry (CRJE) and of the Time Sharing Option (TSO) of System/360 Operating System. Thus, the reader is required to have a knowledge of FORTRAN IV and an understanding of the concepts and facilities of CRJE and/or TSO. The publication identifies areas of the syntax checker that perform specific functions and relates those areas to the program listings.

The FORTRAN IV Syntax Checker, a processing program called via the terminal command language of CRJE or TSO scans input written for the E, G, G1, H, or Code and Go levels of the FORTRAN IV language for syntactical errors. The input is checked on a single-statement basis, i.e., no cross-checking between statements is performed. The syntax checker:

- receives FORTRAN source statements in a chain of buffers from CRJE or TSO.
- scans these statements for errors; multiple errors in one statement are diagnosed whenever possible.
- sends appropriate error messages to CRJE or TSO for printing at the user's terminal.

This program logic manual is directed to the IBM customer engineer who is responsible for program maintenance. Because program logic information is not necessary for program operation and use, distribution of this manual is restricted to persons with program maintenance responsibilities.

PREFACE

This publication provides customer engineers and other technical personnel with information describing the internal organization and logic of the FORTRAN IV Syntax Checker. Publications that are required for an understanding of the syntax checker are:

IBM System/360 and System/370 FORTRAN IV Language, Order No. GC28-6515

IBM System/360: Basic FORTRAN IV Language, Order No. GC28-6629

IBM System/360 Operating System: CRJE Concepts and Facilities, Order No. GC30-2012

Related information is found in:

IBM System/360 Operating System:

CRJE System Programmer's Guide, Order No. GC30-2016

CRJE Terminal User's Guide, Order No. GC30-2014

CRJE Program Logic Manual, Order No. GY30-2011

This manual consists of the following parts:

1. An Introduction, which describes the syntax checker as a whole, including its relationship to Conversational Remote Job Entry and to the Time Sharing Option of the System/360 Operating System. The major divisions of the program and the relationships among them are also described in this section.
2. A Method of Operation section which provides: (a) an overview of the logic of each of the major divisions and (b) detailed descriptions of specific operations and routines within these divisions.
3. A section describing the organization of the FORTRAN IV Syntax Checker. Program components (modules, control sections, and routines) are described both in terms of their operation and their relation to other components. Flowcharts are included at the end of this section.
4. A directory that helps the reader find named areas of code in the program listing, which is contained on microfiche cards.
5. A section illustrating the layouts of tables and work areas used by the syntax checker. These layouts may not be essential for an understanding of the basic logic of the program, but are essential for analysis of storage dumps.
6. A section containing diagnostic aids, including debugging aids and general register contents during execution of major routines.
7. Appendixes, which provide system generation information about the syntax checker, an error code message-originating routine cross reference table, and a sample FORTRAN language definition.

If more detailed information is required, the reader should refer to the comments and coding in the syntax checker program listings.

Third Edition (May 1971)

The specifications contained in this publication correspond to Release 20.1 of the IBM System/360 Operating System.

This edition is a reprint of GY28-6831-1, which was made available in a limited edition.

Changes are periodically made to these specifications; any such changes will be reported in subsequent revisions or Technical Newsletters. Before using this publication in connection with the operation of IBM systems, refer to the latest SRL Newsletter, Order No. GN20-0360, for editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM Branch Office serving your locality.

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, Programming Publications, 1271 Avenue of the Americas, New York, New York 10020.

Summary of Amendments Number 1

Date of Publication: May, 1971

Form of Publication: Revision GY28-6831-2

Metalanguage Descriptions Added

New: Documentation Only

Appendix D, describing the metalanguage for modules IPDTEE and IPDAGH, is included in this edition.

Syntax Table Description Expanded

Maintenance: Documentation Only

The explanation of the contents of a syntax table in the Method of Operation section has been updated for clarification.

Error Message Description Expanded

Maintenance: Documentation Only

The description of error messages in the Method of Operation section has been updated for clarification.

Contents of Communications Area Altered

New: Program and Documentation

The first four bits of the communications region indicate whether the call to the Syntax Checker is a first, intermediate, or final call.

Get-Character Routines Modified

New: Program and Documentation

The descriptions of the Get-Character routines in the Program Organization section have been changed to indicate that those routines obtain characters from a work area.

Executive Module (IPDSNEXC) Descriptions Modified

New: Program and Documentation

The following information has been included in the Program Organization section:

- The Executive builds a statement character string in a work area.
- The Executive passes control to the Checker to check statements.

The following information has been included in the Method of Operation section:

- The parameter list passed by the Executive to the Checker (IPDSNCKR) contains an additional entry.
- The description of the function of the Executive has been expanded for clarification.

Checker Module (IPDSNCKR) Description Modified

New: Program and Documentation

The description of the CKREXPON portion of the Checker has been modified to indicate the setting of a type switch.

Error Message Text Changed

New: Program and Documentation

The text of error messages 104 and 105 in Appendix B has been revised to apply to both free- and fixed-form source statements.

Miscellaneous

New: Program and Documentation

Throughout the book changes have been made to reflect the Syntax Checker's operation with TSO, FORTRAN G1, and Code and Go FORTRAN.

WKATINUE Table Replaced

New: Program and Documentation

The WKATINUE table has been replaced by the table WKATINU, which accommodates free-form source statements.

Work Area Tables Expanded

New: Documentation Only

The tables describing the Syntax Checker Work Area (IPDSNWKA) and the work areas which it comprises have been expanded to include the relative address of each field.

Editorial changes that have no technical significance are not noted here.

Specific changes to the text made as of this publishing date are indicated by a vertical bar to the left of the text. These bars will be deleted at any subsequent republication of the page affected.

TABLES

FIGURES

CHARTS

This section provides general information describing the purpose, structure, and configuration of the FORTRAN IV Syntax Checker, a component of OS/360 Conversational Remote Job Entry (CRJE) and of the Time Sharing Option (TSO) of OS/360.

PURPOSE OF THE SYNTAX CHECKER

At the request of a terminal user, the OS/360 FORTRAN IV Syntax Checker analyzes input written for the E, G, G1, H or Code and Go levels of the OS/360 FORTRAN IV language. Checking is requested via the EDIT command and the SCAN subcommand of the terminal command language. The FORT parameter of the EDIT command specifies the language definition level that is to be used. The user's source statements may be in a user library (CRJE) or OS data set (CRJE and TSO), or they may be entered from the terminal. These statements are checked for errors on a single-statement basis. Any error requiring cross-checking between different statements, such as a reference to a missing label, is not detected. Also, within one statement each syntactical unit is checked independently; e.g., function definition arguments that are not unique within a statement are not diagnosed. Multiple errors in a statement can be diagnosed only if the continued scan through the statement does not depend on any corrective action.

When an error is detected, a diagnostic message is produced. Generally, a diagnostic message produced when scanning a statement for FORTRAN E will be the same as the message produced when the same error is found in scanning that statement for FORTRAN G or H. However, there are messages unique to a particular language level. These messages diagnose, for example, the use of features not supported by a paticular level, such as the use of FORTRAN G features not allowed in FORTRAN E or H.

Since the user receives error diagnostics immediately and can take corrective action, much compilation cost and waiting time for a complete job turnaround can be eliminated.

Structure of the Syntax Checker

The FORTRAN IV Syntax Checker which consists of four modules -- IPDTEE, IPDAGH, IPDSN, and IPDER -- employs a table-driven, syntax-directed approach to checking FORTRAN statements. The syntax of the language is defined in two tables of constants: module IPDTEE for FORTRAN IV level E and module IPDAGH for FORTRAN IV levels G, G1, H and Code and Go.

Module IPDSN, executing under the direction of the syntax table selected by the user, checks the source statements entered for scanning. IPDSN contains two logical segments that are coded in separate control sections: IPDSNEXC, the executive, and IPDSNCKR, the checker. These two control sections use the same work area, IPDSNWKA. The executive interfaces with the environmental system to get source statements and to issue error messages. It does not read statements from or print messages at the terminal; all input/output is performed by the system. Statements read from the terminal are passed to the executive in a chain of buffers; diagnostics are returned to the system in the work area. Other necessary information is passed to the executive via a communications area and an options word. The executive builds a character string in the work area from an input source statement and passes control to the checker. The checker receives pointers to a syntax table and to the character string, checks the statement, and, regardless of whether an error is found, returns control to the executive.

When an error condition is detected, the executive passes control to the error processor module, IPDER, which constructs diagnostic messages.

The flow of program control and data between the FORTRAN Syntax Checker and CRJE or TSO is illustrated in Figure 1.

Syntax Checker and Configuration Considerations

The FORTRAN IV Syntax Checker operates as a separate component in the CRJE or TSO environment. The checker operates on all machine configurations supported by CRJE or TSO and is entirely I/O device independent since all device interface is handled by the environmental system.

At system generation time, an installation specifies which syntax checker modules, if any, are to be included on LINKLIB (see Appendix A). When system generation is completed and if the FORTRAN IV Syntax Checker modules are to be included, LINKLIB contains:

1. The IPDSNEXC load module, which results from linkage editing IPDSN and IPDER and which contains the syntax checker's executable code; and

2. either or both definition table modules, IPDTEE and IPDAGH.

At CRJE startup time, one of three configurations may be selected. Included in each of the configurations is the executable IPDSNEXC module, 10,240(10.0K) bytes, and a work area of 4096(4.0K) bytes plus:

1. only the definition table for FORTRAN E (IPDTEE), total size, 16,384(16.0K) bytes, or

2. only the definition table for FORTRAN G, G1, H and Code and Go (IPDAGH), total size, 19,456(19.0K) bytes, or

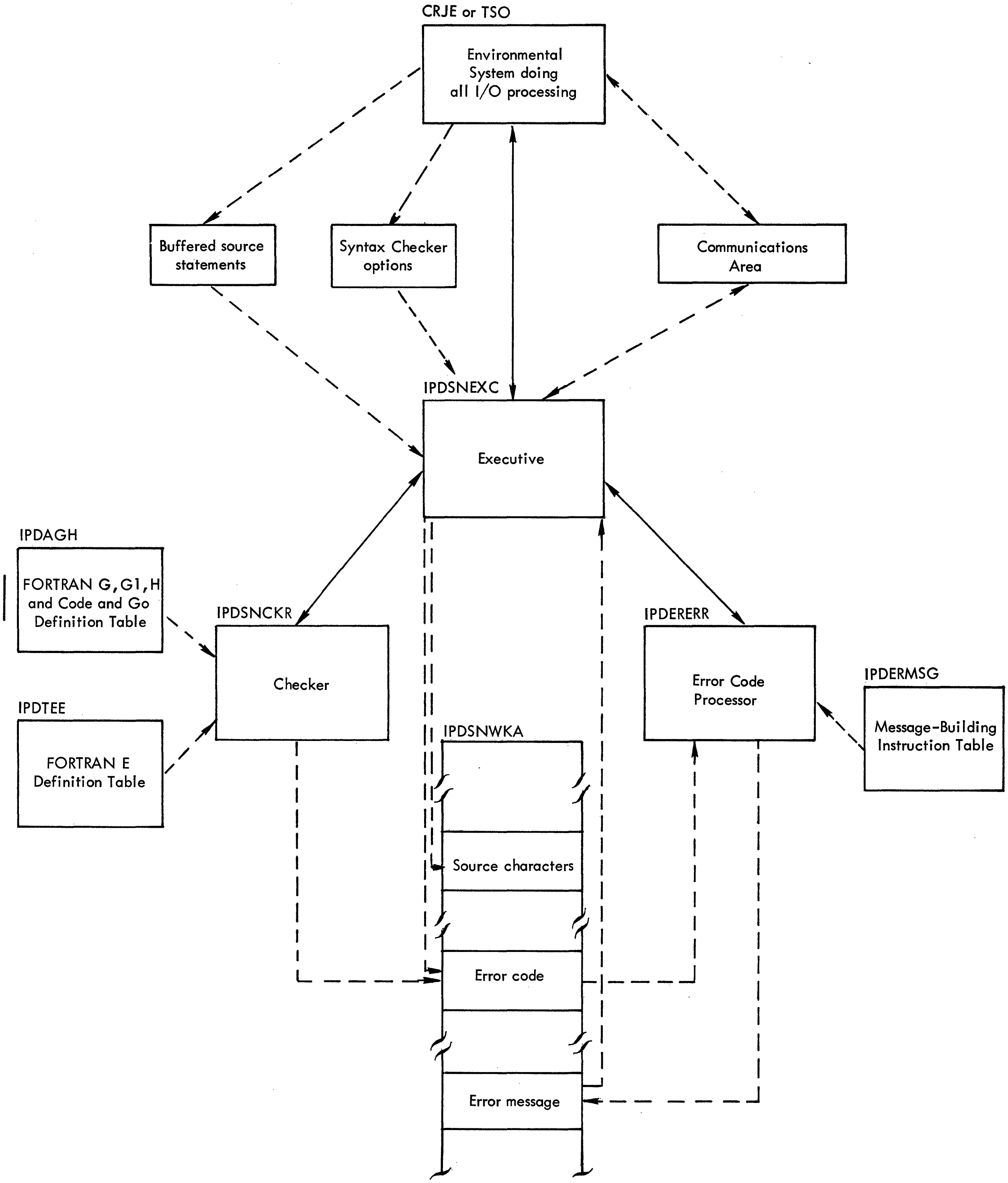
3. both definition tables, total size, 21,504(21.0K) bytes.

At TSO START time, the Link Pack Area may be set up to contain any, all, or none of:

syntax checker executable code
syntax table for FORTRAN level E
syntax table for FORTRAN levels G, G1, H, and Code and Go

For a given TSO SCAN session, the user requires a work area (in his region), the checker executable code (in his region or LPA), and one of the syntax tables (in his region or LPA). Thus the user region storage requirement depends upon the Link Pack Area contents selected at START time; the requirement could be 4.0, 6.0, 9.0, 14.0, 16.0 or 19.0K depending on the contents of LPA.

The modules necessary for each configuration must be resident on LINKLIB after system generation is completed.



Note: The CSECT's IPDSNEXC, IPDSNCKR, and IPDERERR are in the load module IPDSNEXC.

Broken lines represent passage of data; solid lines represent transfer of control.

Figure 1. FORTRAN Syntax Checker System Flow

This section contains:

- a description of the modules IPDTEE and IPDAGH, which contain the FORTRAN IV syntax definitions.
- a description of the syntax checker's interface with the environmental system.
- a detailed functional description of the syntax checker modules IPDSN and IPDER.

SYNTAX TABLES: A DESCRIPTION OF THE MODULES IPDTEE AND IPDAGH

The syntax checker operates on the source statements submitted for checking by comparing them with a "syntax table." The syntax table contains a description of all the valid statements in the language being checked. In the FORTRAN syntax checker, there are two syntax tables: the IPDTEE module is a syntax table for FORTRAN level E, and the IPDAGH module is a syntax table for the G, G1, H, and Code and Go levels of FORTRAN. This section explains how a syntax table (such as IPDAGH) is created, and how it is compared to the source statement being checked.

This explanation commences with "What a Syntax Table Describes." Then, "How a Syntax Table is Created" gives a very general description of syntax table creation, defines a metalanguage, and introduces the use of a metalanguage in creating syntax tables. "The Metalanguage" gives a general description of the metalanguage used in creating IPDTEE and IPDAGH. "The Form of Syntactic Lines" and "Active and Passive Syntactic Lines and Enclosing Lines" describe the type of statements in this metalanguage, and the form in which they are written. A detailed description of the elements of this metalanguage is given in Table 1. The remaining topics assume a general understanding of Table 1. "Elements in the Definition Portion of Syntactic Lines" divides the metalanguage elements in Table 1 into several classes, and explains how some of these elements effect the scanning of portions of the source statement. Then, "Scanning the Syntax Table" explains how the source-scanning metalanguage elements interact with the other metalanguage elements to control the scan of the syntax table. "Issuing Error Messages" describes using the metalanguage to determine which error message is issued if the source statement contains an error. "The Definition of a Simple Language" uses the metalanguage to define a small subset of FORTRAN, providing an example of the use of the metalanguage. Finally, "Translation from the Metalanguage to Assembler Language" describes how the metalanguage description is translated into a series of assembler language DC's to form a syntax table module.

Note: In this context, a metalanguage is defined as a language used to describe another language.

What a Syntax Table Describes

The complete syntactic description of a language would be a description of every character in every possible valid statement in the language. However, the syntax tables in the checker do not describe everything that could be present in the input stream of a FORTRAN compiler. Rather, they describe statement portions of the input lines (i.e., exclusive of statement label and continuation symbols, if present).

Another way to explain what the syntax tables describe is to say that they describe the forms permitted in any fields to which the "get-character" routines have access. These are the routines which obtain characters from the character string for comparison with a syntax table. If the get-character routines were changed, the checker could be used for input other than a contiguous character string. For example, the get-character routines could be tailored to obtain characters from text fields of the records in an input buffer chain. The checker is thus designed with a high degree of independence from source language definition.

How a Syntax Table Is Created

The syntax table for a language could be written directly as a series of DC's in assembler language. But it is also possible to describe the language with a metalanguage and then use the rules stated

in Table 1 and "Translation from the Metalanguage to Assembler Language" to convert the metalanguage description of the language into the DC's. This approach was used to create the IPDTEE and IPDAGH tables. The metalanguage descriptions are part of the comments in the IPDTEE and IPDAGH source modules. Listings of the metalanguage used to create the two tables are included in Appendix D.

The Metalanguage

Table 1 describes in detail all the elements of the metalanguage that was used in creating the IPDTEE and IPDAGH syntax tables. A broader view of using the metalanguage to describe a complete language is given here.

Note: For most of the elements in Table 1 the word "operator" can, and often does in this description, replace "element."

The description of a language is accomplished by writing a series of syntactic lines, using the elements of the metalanguage. The syntactic lines in the definition of a language perform two functions in addition to describing the syntax of the language:

- They determine the order in which the syntax checker will use the syntactic lines in examining a statement submitted for checking.
- They determine which error message the syntax checker will issue if the statement submitted for checking fails to match the definition of the language.

Strictly speaking, neither of these functions is necessary for the description of a language, so the operators and rules which provide the functions can be considered extensions to the concept of a metalanguage. These extensions are necessary when the definition is to be used to analyze statements and diagnose those found invalid. Throughout this manual, the term "metalanguage" refers to the extended metalanguage needed to provide ordering and error message capability in addition to language description.

The order in which syntactic lines are used is not determined by the order in which they are written, as it is in a language such as FORTRAN. Instead, a syntactic line is used when another syntactic line refers to it. The only ordering rule is that one particular syntactic line is used first. This first line will refer to other lines, and these may in turn refer to further lines, and so on. Theoretically, there is no limit to the number of such levels of reference. In addition, a syntactic line may refer to itself. The self-reference facility is very useful in describing expressions, for example. In a properly written set of syntactic lines, every line except the first will be referred to by another line, since no line (except the first) will be used unless another line refers to it.

The Form of Syntactic Lines

Since the metalanguage description need not be prepared in machine-processable form, there are no rules about spacing, continuing, or terminating syntactic lines, and there is no limitation on their length. (For legibility, conventions for these were established for the metalanguage portion of the comments in the source decks of IPDTEE and IPDAGH.)

Each syntactic line consists of three parts: a name, an equal sign, and a definition portion. A syntactic line's name is a word which identifies the line so that references can be made to it. The equal sign separates the name from the definition portion of a syntactic line. The definition portion is the part of the line that actually describes source characters. It uses the definition portion metalanguage elements (see Table 1) to state the syntactic rules for valid statements in the language being described.

Active and Passive Syntactic Lines and Enclosing Lines

There are two kinds of syntactic lines, active and passive. They differ in two ways: the elements that can appear in their definition portions are different, and they are referred to in a different manner. The first line of a language definition must be an active line. the IPDTEE and IPDAGH definitions consist mainly of active lines, with just a few passive lines.

An active line's definition portion may contain any of the definition portion metalanguage elements described in Table 1, except table definitions. Thus, the active line's definition portion can express alternatives, optional items, definite iteration, indefinite iteration, and literals; define error messages; use operators and action codes; refer to active and passive syntactic lines; and state that a statement type has been recognized and that a form is no longer optional. An active line is referred to simply by writing its name in the definition portion of a syntactic line.

A passive line's definition portion consists entirely of one table definition. Only literals, references to syntactic lines, action codes, and zeros may appear between the double quotes that enclose the definition portion. The part of the definition portion between the double quotes is a group of pairs. The first member of each pair is a literal, and the second member is a syntactic line reference, an action code, or zero, which determines what is done if the source characters match the literal. Passive lines are referred to by a plus or a minus followed by the name of the passive line. Since a passive line's definition portion is a table, references to a passive line are often called table references. (See the description of operators in Table 1 of this section.)

The entire group of active and passive syntactic lines defining a language is enclosed by two lines with a special format. These two lines are not considered to be syntactic lines. The one which precedes all the syntactic lines is written:

```
          SYNTAX active-line-name
```

where "active-line-name" is the name of the active line which is to be used first. This active line is called the first syntactic line of the definition, even if it is not physically the first syntactic line. The last, which follows all the syntactic lines, is written:

```
              SYNTAX END
```

Because of these special uses of the word SYNTAX it is not used as the label of any syntactic line.

Elements in the Definition Portion of Syntactic Lines

The elements used in writing the definition portion of both active and passive syntactic lines can be divided into two groups: those that describe source statement characters, and those that do not. The first group consists of the following elements: N, M, L, D, A, K, S, H, C, literal, not-literal, scan, scan-not, the literals in a table definition, and action codes. (Including all the action codes in this class is an arbitrary decision, since an action code amounts to a transfer out of the syntax table to the executable routine corresponding to the action code number. The executable routine could perform any function desired, including the description of source characters.) All the other definition portion elements described in Table 1 constitute the group of elements that do not describe source characters.

Except for the scan, the scan-not, and some of the action codes, all of the elements that describe source characters also can advance a source pointer. If the next available source characters satisfy the description expressed by the element, a pointer in the get-character routines is advanced. After the pointer is advanced, the next available source character becomes the one just to the right of the character(s) which satisfied the element's definition. The characters which matched the element's definition are now unavailable. The source pointer always proceeds from left to right in the string of characters accessible to the get-character routines. However, the syntax checker can save the source pointer and can reset it to the saved value if necessary. This ability to back up in the source is required in the processing of optional items and alternatives.

When an element that can advance the source pointer examines source characters, it produces a T (for true) or an F (for false). Except for the not-literal element, the T is produced when the source pointer is advanced, and F when it is not. The not-literal element produces F if it advances the source pointer (that is, if the literal is present in the source) and T if it does not advance the source pointer. Action codes could be written to work either way, but in the IPDTEE and IPDAGH definitions all action codes that can advance the source pointer produce T if they do advance the source pointer and F if they do not. The scan and scan-not elements never advance the source pointer, but they do produce a T or an F. The scan produces a T if the character it scans for exists anywhere in the available source, and an F if it does not. The scan-not produces an F if the character exists anywhere in the available source, and a T if it does not. The action codes that do not advance the source pointer also produce either a T or an F when they are executed.

The elements that do not describe source characters control the scan of the syntax checker through the syntax table and define error messages. These functions are described in "Scanning the Syntax Table" and "Issuing Error Messages".

Another grouping of elements that is used later in this explanation is T/F elements. The T/F elements are all the elements that produce a T or F when the syntax table scan encounters them. The T/F elements consist of all the source-describing elements, references to active lines, and references to passive lines. Active and passive line references do not describe source characters. However, because they are T/F elements, it is often convenient to think of references to active and passive lines as describing source characters, even though the actual description occurs in the definition portion of the line to which reference is made.

Scanning the Syntax Table

This description of the scan of a syntax table is intended to aid in reading metalanguage descriptions. The actual scan of the table of DC's by the syntax checker is not implemented exactly as described here. However, the effects of the scan described here and the actual scan are the same, provided that the metalanguage statements are properly translated into DC's as specified by Table 1 and "Translation from the Metalanguage to Assembler Language."

Each time the syntax checker begins checking a statement, it starts a scan of the syntactic lines. This scan always begins with the definition portion of the first syntactic line. The scan proceeds element by element, in left to right order, in the definition portion of the syntactic line, except when the scan encounters:

1. An "or" operator in a series of alternatives
2. A reference to a syntactic line
3. The end of an active syntactic line
4. A definite or indefinite iteration operator
5. An active line element which produces an F

The effect of each of these five cases on the scan will now be described. In case 1, the fact that the scan reached the "or" operator means that all the T/F elements in the alternative that precedes the "or" produced T's. Hence, one of the alternatives described was present in the source, and no further alternatives need be tried. The scan therefore skips to the first element after the right brace at the end of the series of alternatives. (If the last alternative is satisfied, the right brace will be encountered instead of an "or," and no skipping is necessary since the normal left-to-right scan then passes outside the right brace.)

In case 2 (reference to a syntactic line) the action that occurs depends on the kind of line that is referred to. If it is an active line, a process known as nesting occurs. This consists of saving the location of the reference to the active line, and transferring the scan to the start of the definition portion of the named active line. References to passive lines operate differently depending on whether a plus or a minus precedes the passive line name. In either a plus or a minus passive line reference, the scan passes from left to right through the literals in the passive line's table. This scan of the literals stops if one of the literals produces a T, or if all the literals are scanned, meaning that they all produced F's.

- A minus passive line reference produces an F if the scan stopped because a T literal was found, and that T literal advances the source pointer. The source pointer is not advanced, and a T is produced by the minus passive line reference, if all the literals produce F's. The element following a T literal is not used in a minus passive line reference.

- A plus passive line reference produces an F (and does not advance the source pointer) if all the literals produce F's. If the table scan is stopped by a T literal, the source pointer is advanced beyond the corresponding source characters, and the element just after the T literal is used. If this element is an action code, an active line reference, or a plus or a minus passive line reference, the effect is the same as if the T literal and the element after it were substituted for the plus passive line reference in the active line which made the reference. If the element after the T literal is a zero, the substitution would be just the T literal; hence the plus passive line reference just produces a T and advances the corresponding source pointer.

Note that if a literal occurred more than once in a table, the left-to-right scan through the table would always produce a T for the leftmost occurrence of the literal when the source matched the literal. Since only the first occurrence can have any effect, multiple occurrences of a literal are considered erroneous. A further consequence of the left-to-right scan of literals is that if any literal begins with the characters of another (shorter) literal, it must be placed to the left of the shorter literal. If this were not done, source characters which matched the longer literal would produce a T with the shorter literal, and the longer literal could never produce a T. Provided that these precautions are observed, any reference to a passive line could be replaced with a reference to an equivalent active line. The equivalent active line will be different for plus references than that for minus references. For example, assume there is a passive line:

```
TABLE = " '1' $1 '2' TWO '3' +THREE
  '4' :-FOUR '5' :0 "
```

where action code 1, active line TWO, and passive lines THREE and FOUR are defined elsewhere. Every reference to +TABLE could be replaced by a reference to PLUSTABLE, which would have to be defined:

```
PLUSTABLE = < '1' :$1 | '2' TWO | '3' :+THREE
  | '4' :-FOUR | '5' :>
```

and every reference to -TABLE could be replaced by a reference to MINUSTABLE which would be defined:

```
MINUSTABLE = ┐'1' ┐'2' ┐'3' ┐'4' ┐'5'
```

These two active lines advance the source pointer past the same source and produce the same T/F result as their passive equivalent. Their effect on the syntax table scan would be equivalent (but not identical) to that of the passive line. Passive lines, then, exist to increase the syntax checker's execution speed. They are usually faster than the equivalent active line when there are four or more literals. They can also reduce core storage requirements, since they can replace two active lines when both plus and minus references are needed to define the language.

Case 3 (the end of an active syntactic line) causes the syntax checker either to return to a "definition-satisfied" routine, or to "T-unnest" to the line that referred to the line that just ended. If the line that ended is the first line, or if the scan encountered a statement commit on that line, the definition-satisfied routine is invoked. If no messages have yet been issued, the definition-satisfied routine will issue an "invalid or excess source characters" message if any nonblank source characters remain available. If the line that ended was not the first line, and if no statement commit on that line has been scanned, T-unnesting occurs. T-unnesting consists of 1) producing a T for the reference which nested to the line just ended, and 2) transferring the scan back to the line containing that reference. The scan then continues with the first element to the right of that active line reference.

In case 4 (definite or indefinite iteration) the scan is restarted at the first iterated element, except when the iteration limit of a definite iteration is satisfied. The right parenthesis after the iteration operator and its matching left parenthesis enclose the iterated elements. The first iterated element is the one just to the right of the enclosing left parenthesis. The first time the scan passes into the iterated elements, it passes over the left parenthesis enclosing them. This sets an iteration count to zero. Then every time the iteration operator is scanned, one is added to this count. This count is kept for indefinite as well as definite iterations so that an action code can test it. For definite iteration, the scan is not restarted if the iteration count equals the iteration limit. In this case, the scan is transferred to the first element after the right parenthesis which encloses the iteration.

In case 5 (an active line element which produces an F) the effect on the scan depends on the location of the element producing the F, and on whether the checker is committed to finding source to match that element. If the syntax checker is committed, an error message will be produced. Discussion of this case is deferred to "Issuing Error Messages", which also explains how to commit the syntax checker. When the syntax checker is not committed, the effect on the scan depends on whether the F-producing element is or is not enclosed in parentheses or braces. In determining whether an F-producing element is enclosed in parentheses or in braces, the syntactic line containing that element is the only one considered. If the F-producing element is within nested braces and parentheses, the effect on the scan is determined by the innermost braces or parentheses enclosing that element. Thus, in the metalanguage line:

```
EXAMPLE = A ( B < C | D ( E ) > ) F
```

A and F are not enclosed by braces or parentheses, B and E are enclosed by parentheses, and C and D are enclosed by braces.

The rules given below describe, for each place it could occur, how an F-producing operator affects the scan when the syntax checker is not committed. Each rule describes only the immediate effect on the scan. To determine the total effect, these rules are applied as many times as necessary. An example of their repeated application is given after the rules are stated.

- If the F-producing element is not enclosed by either parentheses or braces, F-unnesting occurs. F-unnesting consists of 1) producing an F for the reference which nested to the line where the F was produced, and 2) transferring the scan back to the line containing that reference.

- If the F-producing element is enclosed in parentheses, any advances of the source pointer during the current scan of the parenthesized elements are canceled by restoring a saved source pointer, and the scan skips to the first element after the right parenthesis. In effect, the parentheses and the elements they enclose act as a single element which produced a T. Thus, in the current scan of the parenthesized elements, the source pointer is unchanged, and no failure occurs if the optional form is not present in the source characters.

- If the F-producing element is enclosed in braces, any advances of the source pointer in the alternative which contains the element are canceled. The scan then skips to the first element of the next alternative, if there is one. If the F occurred in the last alternative, the entire series (from left to right brace) acts as a single element that produced an F, which is treated as described by the preceding rules.

For the examples, assume that these two lines are written to describe a name, optionally followed by a plus or minus followed by a number:

```
LINEA = N ( LINEB )
LINEB = < '+' | '-' > K
```

Suppose now that these are the first available source characters when nesting to LINEA occurs:

```
Example 1:  A - B
Example 2:  SIX*3
```

In example 1, the N on LINEA advances the source pointer beyond the A and then the reference to LINEB causes nesting to LINEB from LINEA. The first alternative on LINEB produces an F, so the second alternative is tried, and succeeds, advancing the source pointer beyond the minus sign in the source. Since the first available source character is now the B, the K operator fails, and F-unnesting back to LINEA occurs. The reference to LINEB on LINEA has therefore produced an F, but since it is in parentheses, all that happens is that the source pointer advanced by LINEB is backed up, and the scan passes to the end of LINEA, which causes T-unnesting to the line which referred to LINEA. Thus, the reference to LINEA only advanced the source pointer beyond the A.

In example 2, the N on LINEA advances the source pointer beyond the SIX, and then nesting to LINEB occurs. On LINEB, the first alternative is tried and fails, so the second (and last) alternative is tried. This too fails, so the set of alternatives acts as a single F-producing element, producing F-unnesting to LINEA. The F in the parentheses on LINEA acts just as in example 1, with the result that the reference to LINEA advanced the source pointer beyond the characters SIX, and produced a T.

Issuing Error Messages

The error messages issued by the syntax checker can be divided into two classes: explicit and implicit. The difference is that message codes corresponding to the explicit messages are actually written in the syntactic lines, while the message codes for implicit messages are internal to action codes and operators and do not appear explicitly on the syntactic lines.

Explicit Error Messages

For an explicit error message to be issued, two conditions must be satisified:

- Some element must produce an F

- The syntax checker must be committed when the F is produced.

If these two conditions are satisfied, the "current message" is issued. The current message is the error message corresponding to the current error code on the line where a committed element produced an F. The current error code is either the last one scanned on that line, or, if none has been scanned, is determined as explained below.

When the syntax checker begins checking a statement, a default message, "System or Syntax Checker Failure", is the current message for the first syntactic line. This message, or the current message on any active syntactic line, is replaced by a new current message for the syntactic line whenever the scan encounters another error message (i.e., an asterisk followed by the code for that message) on that line.

When nesting to an active line occurs, the current message on that line is set from the current message on the line from which nesting occurred. This message may be appropriate for the entire line to which nesting occurred. If it is not, the definition portion of that line is written so that, as the scan proceeds through the line, the current message is redefined whenever another message would be more appropriate.

In the case of unnesting, the current message on the line to which unnesting occurs remains the same as it was before nesting. Thus, there is no need to reassign that message, even though the line from which unnesting occurred might have redefined its current message many times.

Note: Except for this "restoring of current error message on unnesting," any reference to an active line could be replaced by writing the definition portion of that active line in place of the reference to it. Because of the error message restoration, such a substitution would have to be followed by the error message to be restored if the substituted definition portion redefined the current error message. By rewriting any passive lines as their active equivalent(s), and by then making the substitution just described, any syntactic definition could be reduced to a single active syntactic line. However, if this were done with a definition in which any line refers either directly or indirectly to itself, the resulting active line would be of infinite length.

The second requirement for issuing the current message is that the syntax checker be committed. The checker becomes committed whenever the scan of the syntax definitions encounters a slash(/) or a colon(:). The appearance of either of these two operators signifies that the definition is no longer optional at this point and that any subsequent failure to satisfy the definition constitutes an error in the source statement. These operators are also called "local commit" (/) and "statement commit" (:) in the discussion that follows. The effects of the statement commit described here are in addition to its effects described in "Scanning the Syntax Table." However, braces or parentheses alter syntax checker commitment according to the following rules:

- When the scan encounters a left brace or a left parenthesis, any commit in effect is suspended until the scan passes outside the corresponding right brace or parenthesis, at which time that commit is again in effect. Thus, elements in braces or parentheses cannot be committed by a commit outside the braces or parentheses.

- When a local commit is in effect within braces or parentheses, its effect is terminated when the scan encounters an "or", a right brace, a definite or indefinite iteration operator, or a right parenthesis.

- When a statement commit is in effect within braces, or within parentheses without iteration, the statement commit remains in effect when the scan encounters an "or", a right brace, or a right parenthesis.

- When a statement commit is in effect within parentheses with iteration, it acts like a local commit within the parentheses. However, the statement commit remains in effect when an uncommitted F-producing element or a satisfied iteration count transfers the scan outside the parentheses.

- Any commit in effect when nesting occurs, causes a local commit to be in effect when the scan of the referenced line begins.

- If a local commit is in effect when unnesting occurs, it remains in effect on the line to which the scan returns in unnesting. (Unnesting cannot occur when a statement commit is in effect.)

These rules state that a commit in effect outside a series of alternatives does not commit any element within the braces. However, if none of the alternatives is found in the source, the braces and all the elements they enclose act as a single F-producing element. So if a commit preceded this F-producing element, the current error message is issued.

After it issues an error message, the syntax checker takes one of two actions: checking of the statement is terminated, or checking is continued. If the message code for the current message was an odd number, or, if no more nonblank source characters are available, termination occurs. If the message code was an even number, and if nonblank source characters remain available, checking is continued by changing the F which caused the message to a T. The syntax table scan then proceeds as though the F had not been produced. The source pointer is not changed when the F is changed to a T.

Each message is assigned two message codes: an even number and the next higher odd number. This allows a given message to be issued either with or without termination, whichever is more appropriate for a given situation.

Implicit Error Messages

Implicit error messages are the error messages issued by the C, H, K, N, and S syntactic elements and by the action codes which issue messages. The implicit error message codes are defined in the coding of the routines which interpret these elements, and therefore cannot be altered by anything written on a syntactic line. Some of the implicit messages will not be issued unless the element which can issue them is committed; others are issued even if the element is not committed. Some elements can issue more than one implicit message. Termination of checking can occur after an element issues its implicit message(s). When termination does not occur, the element which issued the implicit message(s) produces a T. None of the implicit-error-message elements described in this manual can cause the checker to become committed. The descriptions of these elements should be consulted for specific details about the implicit error messages associated with an element, and about its effect on the source pointer.

The Definition of a Simple Language

The syntactic description of a subset of FORTRAN is given in Figure 2. Appendix C provides the assembler language equivalent for this subset's description. The subset consists of these types of FORTRAN statements: DO, arithmetic assignment, CONTINUE, unformatted READ and unformatted WRITE, both with required I/O lists, STOP, and END. Logical constants and operators, complex constants, relational operators, subscripts, function references, implied DOs in I/O lists, variable names as DO parameters, variable names as data set references numbers, and digit strings after STOP are not permitted in the statements of this subset. The subset does permit mixed mode, parenthesized expressions. The FORTRAN subset was chosen because its metalanguage description, although short, illustrates many of the metalanguage's facilities. Thus, the subset described in Figure 2 is not intended to be a practical programming language. The action codes and error message codes of Figure 2 are the same as the ones described more fully elsewhere in this manual.

SUBSET, the first syntactic line of the subset definition, determines the overall strategy in scanning statements. It first defines "Unrecognizable Stmnt or Misspelled Keywd" as the message to be issued if all its alternatives fail, and then gives as alternatives the general classes of statements in the subset language. (In this discussion, "fail" is equivalent to "produce an F", and "succeed" is equivalent to "produce a T.") The ordering of these alternatives is important to the correct and efficient operation of the syntax checker, so the reasons for the ordering shown are described in some detail. The most commonly used statement in FORTRAN is the assignment statement. For efficiency, then, a source statement should first be checked as a possible assignment statement. However, FORTRAN permits assignment statements such as:

```
    DO3I=16
```

Therefore, if assignment statement were tried first, no commit could be written in the definition of assignment statement until the checker had determined that a comma did not follow the first operand after the statement's equals sign. To permit an early commit in assignment statement checking, the first alternative tests whether the statement begins with "DO", and nests to DO if it does. The reference to DO will fail if the statement is not a DO statement, and then assignment statement will be tried. If the statement does not begin with "DO", the literal in the first alternative will fail and the scan will skip to the second alternative without nesting to DO.

The second alternative of SUBSET describes assignment statements. If it is reached, the statement is not a DO statement, but could be either an assignment statement or one of the statements beginning with a keyword. Since some FORTRAN keywords are longer than six characters ("CONTINUE" being the only one in the subset), it is not appropriate to use the N operator in the second alternative. The N operator would issue a "name too long" message for any of the long keywords. Instead, the M operator, which issues no error messages, is used. This operator will fail if it encounters a long keyword (or a name too long) so that the third alternative of SUBSET, which refers to a table of the keywords, will be tried before an error message is issued. If the characters which caused M to fail actually are a long name (or a misspelled keyword), instead of a keyword, the reference to the table of keywords will also fail. Then the

```
SYNTAX SUBSET
SUBSET =     *3 < 'DO' DO | M ASSIGNMENT |
             +KEYWORD | N ASSIGNMENT>
DO = S N     '=' -OPERATOR K $100 ',' :
             *4 -OPERATOR K $100 ( ',' /
             -OPERATOR K $100 )
OPERATOR     =  " '+' 0 '-' 0 '**' 0 '*' 0 '/' 0 "
ASSIGNMENT =  '=' : *7 EXPRESSION
EXPRESSION =  ( < '+' | '-' > )
             OPERAND *55 ( +OPERATOR / OPERAND ... )
OPERAND =    < N | K | '(' / *7 EXPRESSION *12 ')' >
KEYWORD =    " 'CONTINUE' 0 'STOP' 0 'READ' INOUT
             'WRITE' INOUT 'END' $300 "
INOUT =   : *30 '(' *27 K $105 *12 ')' *58 IOLIST
IOLIST =  N ( ',' / N ...)
SYNTAX END
```

Error Messages Used in the Subset Definition

<u>Message Code</u>	<u>Message Text</u>
3	Unrecognizable Stmnt or Misspelled Keywd
4	Unsigned Integer Expected
7	Expression Expected
12	) Expected
27	Data Set Ref Number Expected
30	(Expected
55	Operand Expected in Arith Expression
58	I/O List Item Expected

Action Codes Used in the Subset Definition

<u>Code</u>	<u>Action</u>
100	produces T if the preceding K operator was satisfied by a nonzero integer constant. Otherwise, when a commit is in effect a message "nonzero integer expected" is issued, but when a commit is not in effect an F is produced.
105	similar to 100, but the constant must be in the range 1-99 inclusive and the message is "data set ref number expected".
300	produces T if label and continuation fields are blank. Otherwise, when a commit is in effect a message "END requires blank label & contin fields" is issued, but when a commit is not in effect an F is produced.

Figure 2. Metalanguage Definition of a FORTRAN Subset

fourth alternative, which again describes assignment statements, is tried. The fourth alternative uses the N operator, since the fourth alternative cannot be reached if the statement is a valid keyword statement.

The line labeled DO is nested to after the source pointer has been advanced past "DO" in the source, and therefore describes the syntax of a DO statement to the right of the "DO". The sequence

```
-OPERATOR K $100
```

is used to describe the DO parameters. The minus reference to OPERATOR fails if an operator precedes one of the parameters. The K advances the source pointer beyond the parameter, and then action code 100 fails unless the parameter was a nonzero integer. As explained previously, until the comma after the first parameter is found, the statement could be an assignment statement beginning with "DO". So the commit on DO must be after the literal which advances the source pointer past that comma. The third parameter of a DO statement is optional and is therefore enclosed in parentheses. If, however, a comma appears after the second parameter, the third parameter is required. The commit after the literal comma in the parentheses reflects this fact.

OPERATOR is a passive line describing all the arithmetic operators in FORTRAN. It illustrates the general principle that if the same source characters could match more than one of a table's literals, the longer literal must appear first in the table. In this instance, both the '*' and the '**' in the table would match two asterisks in the source. If the '*' appeared first in the table, it would advance the source pointer beyond one of the two asterisks in the source, leaving the other one available. It would never be possible for '**' to succeed if '*' appeared first.

The order of the alternatives on the first syntactic line permits a statement commit in ASSIGNMENT if an equals sign is the first available source character when the scan nests to ASSIGNMENT. The remaining source must satisfy EXPRESSION, or the checker will issue message 7. An EXPRESSION is defined as an optional plus or minus sign followed by at least one OPERAND. This may optionally be followed by any number of OPERATOR OPERAND pairs, with the presence of an operator requiring an operand. Message 55 is in effect all during this indefinite iteration to diagnose the absence of an OPERAND after an OPERATOR. An OPERAND is either a name, a numeric constant, or an expression enclosed in parentheses. In the last case, OPERAND nests to EXPRESSION, and then EXPRESSION nests again to OPERAND as many times as required to advance the source pointer beyond the expression in the parentheses, finally unnesting back to OPERAND to advance the source pointer beyond the matching right parenthesis in the source. At any of these nestings another left parenthesis may cause the process to be repeated. Even though EXPRESSION is, in this process, indirectly referring to itself, no difficulty arises. A line may directly or indirectly refer to itself because 1) the syntax table is not modified in any way as it is scanned, and 2) the processing required in these "self-nestings" is exactly the same as that for any nesting.

KEYWORD is a passive line with literals for all of the subset's keywords. If the statement is a STOP or CONTINUE, the plus reference to KEYWORD produces a T in the third alternative of SUBSET, causing checking to terminate with no message except, possibly, an "invalid or excess source characters" message. If the first source characters are READ or WRITE, nesting to INOUT occurs, reflecting the fact that the syntax for these statements is identical after the keyword. In the case of END, action code 300 fails if the END statement was labeled with a number, since labeled END statements are not permitted in FORTRAN.

INOUT describes input/output statements to the right of their keywords in a straightforward way. Action code 105 checks the data set reference number for proper range. Message 58 is in effect all during the scanning of IOLIST since no message is defined in IOLIST. The input/output lists consist of at least one name, optionally followed by any number of names separated by commas.

Translation from the Metalanguage to Assembler Language

With one exception, namely, the equals sign separating a line's name from its definition portion, every element in a metalanguage syntactic line has an equivalent in the assembler language coding of the syntax table. The name portion of a syntactic line is translated to:

```
symbol EQU *
```

The symbol naming conventions are explained in "Symbol Conventions." All the elements in the definition portion of the syntactic line are translated. Some elements are represented by just a one-byte code; other elements are represented by a one-byte code followed by one or more parameters. These parameters can be displacements (pointers to other elements in the current syntactic line or to other syntactic lines), length factors, literals, etc. The translation formula for each of the elements is contained in Table 1, Metalanguage Elements and Assembler Language Equivalents. Appendix C illustrates the translation of a sample metalanguage definition. The translation formulas follow symbol and displacement conventions, which should be understood before using Table 1 to perform a translation from the metalanguage to assembler language.

Symbol Conventions

Within the assembler language coding of the IPDTEE and IPDAGH modules, nine types of symbols are used. The first type of symbol is the label assigned to the control section. This label is the name specified as the active line name written after SYNTAX in the first statement in the meta-language definition. This active line name is also used as the operand of each module's END instruction.

The remaining eight types of symbols are distinguished by their first three characters, which may be any of the following: LIN, BRC, PAR, ALT, TAB, COD, ACT, and DEF. The symbols starting with LIN, BRC, PAR, ALT, and TAB have their three-character mnemonic followed by a five-digit numeric, which starts at 00001 and which is incremented by one when a unique symbol of the particular type is needed. For each syntactic line (both active and passive) in the definition, there is a LIN symbol, which is used in the coding every time that the syntactic line is referenced. The LIN symbol is created the first time a reference is made to a syntactic line, whether it be on the right or the left side of a syntactic line's equal sign. The label LIN00001 is assigned to the assembler coding for the first syntactic line to be used when source statement scanning begins.

There is a BRC symbol for each right brace, >, in the definition. Each time a left brace, <, is encountered, a BRC symbol is created for the corresponding right brace.

There is a PAR symbol for each right parenthesis,), in the definition. Each time a left parenthesis, (, is encountered, a PAR symbol is created for the corresponding right parenthesis.

There is an ALT symbol for each alternate operator, |, and for each right brace in the definition. Each time a left brace is encountered, an ALT symbol is created for the first alternate operator. Each time an alternate operator is encountered, an ALT symbol is created for the next alternate operator, or, if there is no next alternate operator, for the right brace which ends the series of alternatives.

There is a TAB symbol for each passive line (table line) in the definition. The TAB symbol is the label of the byte that contains the length of the longest literal in the passive line.

The COD and ACT mnemonics are followed by a three-digit numeric. There is a COD symbol for each unique error code, *n, and an ACT symbol for each unique action code, $n, used in the definition.

There is a DEF symbol defined for each of the metalanguage operators and for the end-of-syntactic line indicator. Following the characters DEF there are a maximum of five characters used to specify particular operators, for example, DEFITDEF specifies the definite iteration operator and DEFEND specifies the end of a syntactic line.

All the COD, ACT, and DEF symbols are defined in a series of EQU's following the last metalanguage statement, SYNTAX END.

Displacement Conventions

The syntax tables contain two types of displacements:

1. one-byte displacements, which are calculated from the start of a syntactic line.
2. two-byte displacements, which are calculated from the start of the syntax definition table.

The following are the conditions under which one-byte displacements occur:

1. Each time a left brace is encountered in a syntactic line, a "false" displacement is constructed to the first alternate operator in the series of alternatives and a "true" displacement is constructed to the associated right brace.
2. Each time an alternate operator is encountered, there is constructed a displacement to the next alternate operator or to the right brace which ends this series of alternatives.
3. Each time a left parenthesis is encountered, a displacement to the associated right parenthesis is constructed.

The following are the conditions under which two-byte displacements are constructed:

1. At the start of the table, there is a displacement to the first syntactic line to be used by the checker when scanning FORTRAN source statements.
2. Each time a reference within a syntactic line is made to another syntactic line, a displacement to the start of the definition portion of the referenced line is constructed.

Table 1. Metalanguage Elements and Assembler Language Equivalents (Part 1 of 6)

Metalanguage Element	System/360 Assembler Language Equivalent	DEF Symbol Value
I. Elements used in definition portion of syntactic lines.		
< Left brace; indicates the start of a series of alternatives.	DC AL1(DEFLBRCE) DC AL1 (ALTxxxxx-LINxxxxx) DC AL1 (BRCxxxxx-LINxxxxx)	DEFLBRCE= X'00'
> Right brace; indicates the end of a series of alternatives.	ALTxxxxx EQU * BRCxxxxx DC AL1(DEFRBRCE)	DEFRBRCE= X'02'
| Or; separates the alternatives in a series of alternatives within braces. The | can be used only within braces.	ALTxxxxx DC AL1(DEFOR) DC AL1 (ALTxxxxx-LINxxxxx)	DEFOR= X'04'
(Left parenthesis; indicates the start of a series of optional items.	DC AL1(DEFOPTST) DC AL1 (PARxxxxx-LINxxxxx)	DEFOPTST= X'06'
) Right parenthesis; indicates the end of a series of optional items.	PARxxxxx DC AL1(DEFOPTED)	DEFOPTED= X'08'
If the source satisfies the definition within parentheses, source statement scanning continues with the first available source character after the last source character used to fulfill the definition. Normally, once the source fails to satisfy the parenthetical definition, no error is recorded but source scanning backs up to the first source character tested for the parenthetical definition. However, the appearance of the / or : operator (see subsequent descriptions) within parentheses signifies that the definition is committed; i.e., the definition is no longer optional at this point, and, any subsequent failure to satisfy the parenthetical definition constitutes an error in the source statement.		
... Indefinite iteration; represents iteration that has no upper or lower limit. The iteration must be specified within parentheses and it starts with the element following the left parenthesis. The elements preceding the periods may appear one time, many times, or not at all.	DC AL1(DEFITIND)	DEFITIND= X'0E'
.n. Definite iteration; represents iteration with an upper limit of n. The iteration must be specified within parentheses and the iteration starts with the element following the left parenthesis. The set of elements preceding .n. may appear up to a maximum of n times or not at all. A maximum of 255 may be specified for n.	DC AL1(DEFITDEF) DC AL1(n)	DEFITDEF= X'10'

Table 1. Metalanguage Elements and Assembler Language Equivalents (Part 2 of 6)

Metalanguage Element	System/360 Assembler Language Equivalent	DEF Symbol Value
I. Elements used in definition portion of syntactic lines.		
/ Local commit; commits the checker to a particular alternative or optional definition during the time this operator is in effect. (A diagnostic is issued if a committed definition is not satisfied.) If the / operator is enclosed within < > or ( ), whether on its own line or at an earlier syntactic line, the commitment remains in effect from the time the / operator is encountered until the time its closing > or) is encountered.	DC AL1(DEFCOMIT)	DEFCOMIT= X'0A'
: Statement commit; essentially, this operator is a request to disregard alternatives. If the : operator is encountered, any future failure in any < > or ( ) enclosing the : on this line will cause a diagnostic condition regardless of alternatives at a higher level. In fact, alternatives at higher levels are never examined during the remainder of the statement's checking. If the : operator is encountered in an alternative, any subsequent alternatives in that series of alternatives are ignored.	DC AL1(DEFSTCMT)	DEFSTCMT= X'0C'
active line name to the right of a syntactic line's equal sign. Symbol; implies nesting to the named syntactic line. The effect, except for error messages, is the same as if the entire definition portion of the named line appeared where its name appears. (For error message handling with nested syntactic lines see "Explicit Messages".)	DC AL1(DEFSYMBL) DC AL2 (LINxxxxx-definition name)	DEFSYMBL= X'12'
M Maybe-name; defines a FORTRAN variable name, which consists of from one through six characters, the first of which is alphabetic. If the first character is not alphabetic, or if more than five successive alphameric characters follow the first alphabetic in the source, the M operator fails.	DC AL1(DEFMNAME)	DEFMNAME= X'14'
N Name; defines a FORTRAN variable name. The N operator differs from the M operator in that the N operator is satisfied by any length alphameric string whose first character is alphabetic; however, a "name" longer than six characters is diagnosed by an implicit error message in the N operator routine. (Source scanning resumes at the first non-alphameric character.)	DC AL1(DEFNAME)	DEFNAME= X'16'
L Letter; defines a single alphabetic character ('A' through 'Z' or '$').	DC AL1(DEFLETTR)	DEFLETTR= X'18'

Table 1. Metalanguage Elements and Assembler Language Equivalents (Part 3 of 6)

Metalanguage Element	System/360 Assembler Language Equivalent	DEF Symbol Value
I. Elements used in definition portion of syntactic lines.		
D Digit; defines a single decimal digit ('0' through '9').	DC AL1(DEFDIGIT)	DEFDIGIT= X'1A'
A Alphameric; defines a single alphameric character ('A' through 'Z', '$', or '0' through '9').	DC AL1(DEFALMER)	DEFALMER= X'1C'
K Numeric constant; defines either a real constant or an integer constant. The format of these constants is exactly the same as described in <u>IBM System/360 and System/370 FORTRAN IV Language</u>, Order No.GC28-6515.	DC AL1(DEFNUMBR)	DEFNUMBR= X'1E'
S Statement number; defines a number that consists of at least one non-zero digit, followed by a maximum of four more digits. Optionally, the statement number may have any number of leading zeros.	DC AL1(DEFSTNUM)	DEFSTNUM= X'20'
H wH-literal; defines a character string that is preceded by wH where w is the number of characters in the string (leading zeros are permitted in w). If the wH form is recognized, but w is not within the range of 1 through 255, an error message is printed.	DC AL1(DEFHOLLR)	DEFHOLLR= X'22'
C Character string; defines a character string enclosed by quotes. The string can have a maximum of 255 characters. Character string scanning is terminated when 256 source characters have been examined and a terminating quote is not found. An error message is printed, and statement checking is terminated.	DC AL1(DEFCSTRG)	DEFCSTRG= X'24'
'aa...a' Literal; defines a literal value expected in the source statement.	DC AL1(DEFQUOTE) DC AL1(length of aa...a) DC C'aa...a'	DEFQUOTE= X'26'
¬'aa...a' Not literal; defines a literal value that is not valid in the source statement, i.e., source unequal to literal satisfies the definition.	DC AL1(DEFNOTQT) DC AL1(length of aa...a) DC C'aa...a'	DEFNOTQT= X'28'
A quote and an ampersand within a literal must be represented by two single quotes and two ampersands, respectively. In determining the length of the literal, two quotes or two ampersands count only as a single character. A blank anywhere in a literal always causes a non-match with the source statement.		

Table 1. Metalanguage Elements and Assembler Language Equivalents (Part 4 of 6)

Metalanguage Element	System/360 Assembler Language Equivalent	DEF Symbol Value
I. Elements used in definition portion of syntactic lines.		
&a Scan a; represents a search of the source statement for the character represented by a. The search succeeds if the character is found.	DC AL1(DEFSCAN) DC C'a'	DEFSCAN= X'2A'
&¬a Scan not a; represents a search of the source statement for the character represented by a. The search succeeds if the character is not found.	DC AL1(DEFSCNOT) DC C'a'	DEFSCNOT= X'2C'
The search starts at the next source character to be examined and continues until the character sought is found or the end of the statement is reached. After the search is completed, source scanning resumes at the character at which the search was started. The character sought can not be a blank.		
$n Action; executes the action routine represented by the code n. (n is a maximum of 999, but no more than 128 different action codes may be defined.) The routine is executed immediately when the code is encountered. Action routines perform specialized checking that is not done by the operators.	DC AL1(DEFACTN) DC AL1(ACTn)	DEFACTN= X'2E'
*n Error code; defines an error message that is to be printed if an error occurs when checking subsequent fields in the syntactic line. If the checking of a subsequent field involves nesting to another syntactic line for which there is an error code, say nl, and an error occurs on a committed definition in this second line, the error message represented by code nl is printed. There may be more than one error code specified per syntactic line but only one code is in effect on that line at any one time, and that code is the last one encountered as the checker proceeds along the syntactic line. The error code n, which is 1, 2, or 3 digits, is specified as follows: odd number - error terminates source statement scanning; even number - error does not terminate source scanning. For each error message there are two codes; that is, 2 and 3 represent one message, 4 and 5 another message, etc. The allowable range for code numbering is 2 to 255, which provides a maximum of 127 unique messages.	DC AL1(DEFMESSG) DC AL1(CODn)	DEFMESSG= X'30'
+table-name (referred to as +passive line name in the previous metalanguage descriptions) Search table: causes a search on a series of literals.	DC AL1(DEFTABLP) DC AL2 (LINxxxxx-definition name)	DEFTABLP= X'32'

Table 1. Metalanguage Elements and Assembler Language Equivalents (Part 5 of 6)

Metalanguage Element	System/360 Assembler Language Equivalent	DEF Symbol Value
I. Elements used in definition portion of syntactic lines.		
-table-name (or -passive line name) Search not table; causes a search on a series of literals none of which should appear in the source. Passive line name is the name of the passive syntactic line that defines the table of alternatives.	DC AL1(DEFTABLM) DC AL2 (LINxxxxx-definition name)	DEFTABLM= X'34'
" 0 " $n 'aa...a' symbolic-name ... +table-name -table-name Table definition; surrounds the string of table arguments and functions. A table argument must be a literal of the form 'aa...a'. A table function can be one of the following: 0 representing a return to the syntactic line without further action or checking. $n representing an action code. The action specified is performed before returning to the syntactic line. symbolic-name representing nesting to another syntactic line. +table-name or -table-name representing a search of another table of alternative literals. If the table is being searched because of the -table-name operator, table functions will always be ignored.	DC AL1(DEFTABLE) DC AL2 (TABxxxxx-*+1) DC AL1(length of aa...a) DC C'aa...a' DC AL1(symbol 1) DC X'FF' for a func- DC C'T' tion of 0 or DC AL1(ACTn) for DC C'T' $n or DC AL2(symbol2) otherwise . repeat pre- . vious DC's . (except for . first two) . for each pair . of table ar- . guments and . functions TABxxxxx DC AL1 (length of longest literal) where symbol 1 =DEFACTN for 0 or $n =DEFSYMBL for symbolic-name =DEFTABLP for +table-name =DEFTABLM for -table-name symbol 2 =LINxxxxx- definition-name	DEFTABLE= X'40'

Table 1. Metalanguage Elements and Assembler Language Equivalents (Part 6 of 6)

Metalanguage Element	System/360 Assembler Language Equivalent	DEF Symbol Value
II. Other Metalanguage Elements		
symbolic name to the left of a syntactic line's equal sign Line name; indicates the start of a syntactic line, and provides a name which may be used in other lines to refer to it.	LINxxxxx EQU *	
End; indicates the end of a syntactic line. There is no corresponding operator in the meta-language. This DC is to be coded at the end of each syntactic line to indicate its end to the syntax checker.	DC AL1(DEFEND)	DEFEND= X'36'

SYNTAX CHECKER ENVIRONMENTAL INTERFACE

All communication between the syntax checker and its environment is handled by the executive segment (IPDSNEXC) of the IPDSN module. Input to IPDSNEXC from the environmental system consists of a pointer in register 1 to a parameter list with the format:

```
DC A(1st buffer in chain)
DC A(communications area)
DC X'80' (this byte is not tested by
          the syntax checker)
DC AL3(options word)
```

Buffer Chain

The first entry in the parameter list contains the address of the first buffer in a chain of one or more buffers.

1. Format: The format of the buffer chain varies according to the type of records it contains. There are two types of records: fixed and variable length.

   For fixed length records, a buffer in the chain has the following format:

```
┌─┬─────────────┬───────────────┬───────────────┬────
│ │             │    record1    │    record2    │ ...
│C│chain address├───────┬───────┼───────┬───────┼────
│ │             │       │line   │       │line   │
│ │             │text1  │no.1   │text2  │no.2   │
└─┴─────────────┴───────┴───────┴───────┴───────┴────
0 1             4               L+4             2L+4

                        L


      ┌─────────────────────────────┐
  ... │           record            │
      ├──────────────┬──────────────┤
      │    text      │   line no.   │
      └──────────────┴──────────────┘
      (C-1)  L+4                 CL+4
```

where:

C = low order seven bits specify number of records in the buffer; bit 0 is set to 1 by the syntax checker when the buffer has been processed and should be released by the calling program.

chain address = address of the next buffer; the last buffer in the chain has this field set to binary zeroes.

record = text and line no.

text = columns 1-72 (TSO) or columns 1-80 (CRJE) of a FORTRAN line. However, in both TSO and CRJE, only columns 1-72 are syntax checked, since columns 73-80 in CRJE are reserved for sequence numbers.

line no. = data set line number. There are always 8 bytes present for the line number field even though they are meaningless if the options word, byte 3, bit 1=1.

L = length of each record (80 or 88 bytes)

CL+4 = length of buffer (relative address of first byte beyond buffer.)

For variable length records, a buffer in the chain has the following format:

```
┌─┬─────────────┬─────────────────────┬────
│ │             │       record1       │...
│ │             ├────┬─────────┬──────┼────
│C│chain address│    │ binary  │      │
│ │             │ L1 │ zeros   │ data1│
└─┴─────────────┴────┴─────────┴──────┴────
 0 1             4    6         8
```

L_1

```
     ┌─────────────────────┐
     │       record        │
     ├────┬─────────┬──────┤
  ...│    │ binary  │      │
     │ L  │ zeros   │ data │
     └────┴─────────┴──────┘
```

L

where:

C and chain address are defined as for a buffer with fixed length records.

record = record length (L), two bytes of binary zeros, and data.

Li = the length of the _i_th record. This is a signed binary number. The sign (leftmost) bit must be positive (a binary zero).

data = (a) line-numbered data set (options word, byte 3, bit 1=0): 8-bytes for a line number followed by text consisting of the characters to be syntax checked.

(b) _not_ a line-numbered data set (options word, byte 3, bit 1=1): text consisting of the characters to be syntax checked.

2. _Text Format_: Buffer chains of fixed length records may contain either standard-form or free-form FORTRAN source text. Buffer chains of variable length records may contain only free-form FORTRAN source text.

Standard-form FORTRAN statements (specified by options word, byte 3, bit 5=0) are in EBCDIC and in the form required by the E, G, G1, and H level FORTRAN compilers. Any variations in the format permitted to the terminal user (e.g., the use of a hyphen for a continuation symbol or the absence of a continuation symbol in column 6) will have been translated into standard FORTRAN format by CRJE or TSO before control is given to the syntax checker. Standard FORTRAN statement format is described in _IBM System/360 and System/370 FORTRAN IV Language_, Order No. GC28-6515.

Free-form FORTRAN statements (as currently defined for Code and Go FORTRAN) are described in Appendix B. If free-form is specified (options word, byte 3, bit 5=1), no distinction is made between upper and lower-case alphabetic characters for the scan. However, with the exception of the character that indicates a line is continued, any character in the text portion of the record is considered part of the FORTRAN statement to be scanned. It is assumed that such characters as tab, backspace, and carriage-return will have been removed before the record reaches the checker for scanning.

The maximum length of a FORTRAN statement is 1320 characters, in both free-form and standard-form. The checker will scan up to 20 lines per statement in either form. (Statement length excludes the statement number and continuation fields.) All statements in a buffer, except perhaps the last, are checked without returning to the calling program unless an error is encountered. Whether or not the last statement (assuming no previous errors) is checked before returning depends upon the following considerations. If buffer chain span mode is not specified (options word, byte 3, bit 3=0), the last statement is considered a complete statement and is scanned before returning. For buffer chain span mode (options word, byte 3, bit 3=1) and if the input is standard-form, the last statement of the current buffer chain is assumed continued in the next buffer chain; the statement (or partial statement) is not scanned until the next call to the checker. For buffer chain span mode and free-form FORTRAN input, if the last line of the last statement is a continued line, the statement is not scanned until the next call to the checker; otherwise, the statement is checked before returning.

Communication Area

The second entry in the parameter list points to a communications area which contains four words. The first word of the communications area contains information to control the checker's working storage requests and statement scanning. The second word contains the address of the text of an error message sent by checker to the calling program. The third and fourth words are used as temporary storage areas.

Word 1

Byte 0 bits 0-3 (X means either a 0 or a 1)

0XXX initial entry. Obtain and initialize work area and load FORTRAN definition tables specified in byte 1 of the options word. After the work area has been initialized, set relative line number to zero and perform syntax checking of input. (However, if the buffer address in the parameter list is zero, as it may be on the call to the syntax checker by CRJE at CRJE startup time, perform no syntax checking.)

1X1X last entry. Do not syntax check but release the work area and return.

1000 intermediate entry after return code 0 or 4. Reinitialize relative line number to zero and perform syntax checking for the new input buffer(s). (Work area exists.)

110X intermediate entry after return code 8 (see error messages description). Continue checking the contents of the buffer(s).

1001 intermediate entry after return code 12. Register 1 points to the parameter list, the first word of which is:

DC A(0)- no further buffer chains; check last statement of previous buffer chain as is.

DC A(1st buffer in next chain)- (previous buffer should have been released.) Perform syntax checking on last statement of previous buffer chain (source has been saved in work area) and on statements of this next buffer chain.

bits 4-7 reserved for future use.

Bytes 1-3 address of the work area is stored here by the checker on first entry for use in subsequent entries.

Word 2

Address of the error message. The format of the error message is described in the section on output.

Words 3 and 4

Temporary storage area used by the checker when it issues a GETMAIN for working storage.

Options Word

The last entry in the parameter list contains the address of an options word, which is formatted as follows:

Byte 0 contains a code indicating the level of the FORTRAN language to be used in scanning.

Code	Level
X'00'	FORTRAN H
X'01'	FORTRAN E
X'02'	FORTRAN G
X'03'	Code and Go FORTRAN
X'04'	FORTRAN G1

Levels G, G1, H, and Code and Go accept the full FORTRAN IV language, while level E accepts only the basic subset of the FORTRAN IV language. Levels G, G1, and Code and Go accept DEBUG language statements. At the first release of TSO, list-directed I/O will be supported by the G1 level making it identical to the Code and Go FORTRAN level. In all the levels, FORTRAN keywords may be used as variable names and blanks may appear anywhere in the source statement. (Note: FORTRAN E compiler has this as an option.)

Byte 1 bits 0-5 reserved for future use.

bit 6=1 FORTRAN G/G1/H/Code and Go definition is loaded on an initial entry.

bit 6=0 FORTRAN G/G1/H/Code and Go definition is not loaded on an initial entry.

bit 7=1 FORTRAN E definition is loaded on an initial entry.

bit 7=0 FORTRAN E definition is not loaded on an initial entry.

Byte 2 record length: If fixed length records, this is 80 bytes for TSO and 88 bytes for CRJE. This field is not used with variable length records.

Byte 3 bit 0 reserved for future use

bit 1=0 line numbered data set (line number should appear in error message)

bit 1=1 no line numbers

bit 2 not used by FORTRAN syntax checker

bit 3=0 scan current buffer chain as containing complete statements; i.e., do not issue return code 12. (Buffer chain span bit off)

bit 3=1 return to calling program with return code 12 if last statement in input buffer chain is or may be incomplete. (Buffer chain span bit on)

bit 4=0 fixed length records

bit 4=1 variable length records

bit 5=0 standard-form source

bit 5=1 free-form source

bits 6-7 not used by FORTRAN syntax checker

Output

Error Messages

Each statement is classified according to type, either keyword or assignment, and then scanned for errors. Checking continues after an error is found if it is very probable that the error does not affect the scan of the remainder of the statement. For example, scanning would be resumed if an invalid statement label is found.

When an error is found, a return code in register 15 and a diagnostic message, whose address is in the communications area, are passed to the CRJE or TSO calling program. The return code indicates whether all statements in the input buffer chain have been completely checked. If the buffer has not been completely checked (return code=8), the environmental system will return to the syntax checker indicating this situation via the information in the communications area. If the buffer has been completely checked (return code=4), the next call to the syntax checker will be to scan a new input buffer, or if no statements remain to be scanned, to release working storage. The error message area is 72 bytes long. Error messages contain the following fields in left-to-right order:

1. The six-character message identification code followed by two blanks.

2. The line identifier of the line that contained the error, followed by one blank.

   a. Line Numbered Data Sets (options word, byte 3, bit 1=0): The line identifier is the last eight characters of the record that contained the error, except that if these characters contain leading blanks and/or zeros, the line identifier is shortened to exclude as many as seven of them.

   b. Data Sets with No Line Numbers (options word, byte 3, bit 1=1): The line identifier is one to eight digits, representing the relative position of the erroneous line within the buffer chains received from the calling programs since the last initial entry or intermediate entry after return code 0 or 4.

3. Optionally, six source statement characters followed by a blank. This field will not be present if the end of a source statement had been reached when the error was detected. Otherwise, the six source characters will be a nonblank character indicating the location of the error, and the five characters (including blank characters) immediately following it in the source statement line. If there are fewer than six

characters remaining in the statement/ field of the line, the six-character field is padded with trailing blanks.

4. The text of the diagnostic.

5. Sufficient blanks to fill the remainder of the message buffer.

Return Codes

When the syntax checker returns to the calling program, a return code is set in register 15 to indicate the status of the checker. The values of the return code are summarized in Table 2. The only case in which a return code has no significance is on return from a "last entry" call by the environmental system to free the work area. Return codes 4 and 8 are described above under "Error Messages."

On syntax checker return to the calling program, a return code of 0 indicates no error message and completed buffer checking. The next call to the checker will be for the purpose of scanning a new set of FORTRAN statements or to release the working storage area.

On syntax checker return to the calling program, a return code of 12 indicates the interface has specified buffer chain span mode (options word, byte 3, bit 3=1) and the last statement of the buffer chain is or may be incomplete. The next call to the checker will provide the next buffer chain for continuing the checker's processing, the scan to begin with the last statement of the previous buffer.

THE IPDSN MODULE

Module IPDSN, which is directed by the appropriate definition module, verifies the syntax of FORTRAN statements sent to it by the environmental system (CRJE or TSO). There are two segments, executive and checker. All entries and exits of the syntax checker from the environmental system go through the executive segment. Calls to the error message generator, IPDERERR, are also made only by the executive segment.

EXECUTIVE (IPDSNEXC)

The primary functions of the executive are to manage work areas and syntax definition tables, to scan through input buffers in order to construct a character string of FORTRAN statement text, to call the checker for syntax checking, to call the error processor for the construction of all error messages, and to set up proper returns as shown in Table 2, to the system calling the syntax checker.

Table 2. Return Code Summary

Return Code	Meaning
0	no error message to be sent, get more source input
4	error message to be sent to terminal user, get more source input
8	error message to be sent, call syntax checker module again with same buffer chain and 'continue checking buffer contents' indicated (communications area word 1, byte 0, bits 0-3 set to '110X')
12	no error message to be sent, last statement of buffer chain has been saved but not syntax-checked, it may be incomplete; call syntax checker module again with the address of the next buffer chain or a zero address indicating the statement saved is to be checked as is.
16	conditional GETMAIN failure, not enough core storage available
20	definition table requested by terminal user conflicts with the tables requested on initial entry (see the bit settings in bytes 0 and 1 of the options word)

The first word in the communications area indicates in bits 0-3 of the first byte whether the call to the syntax checker is a first, intermediate, or final call. On a first call, after obtaining the work area from subpool 1 via a GETMAIN, IPDSNEXC places the work area's address into bytes 1-3 of the communication area's first word (If the required amount of working storage is unavailable, the return code of 16 is passed back to the system.) The syntax definition table(s) requested by bits 6 and 7 in the second byte of the options word are loaded using the OS LOAD macro; whether this table is brought in from disk or resides in the Link Pack Area does not affect the checker. The LOAD will ABEND if not enough core is available or if the table is not in the Link Pack Area or in

LINKLIB. Since CRJE uses the first entry call at startup time, any ABEND resulting from a table not being in LINKLIB will occur only then and not during a CRJE session. This error should occur only if the set of tables requested by the second byte of the options word does not correspond to the options specified in the system generation CHECKER macro. Also, if there is insufficient storage for the table(s), the LOAD results in an ABEND at startup time.

In the TSO environment, any of these conditions will result in an ABEND during a user's terminal session since an initial call to the syntax checker is made at the time of user request for FORTRAN syntax checking; only that user's task will ABEND.

For intermediate entries in the CRJE or TSO system, the syntax checker is called to perform processing for various users, who may require different language definition tables. On the final entry to the syntax checker a DELETE is executed for the syntax definition table(s). The work area is released using a FREEMAIN. When the syntax checker returns from the final entry call, the return code setting has no significance.

The executive collects lines of a FORTRAN statement from the input buffers and builds a character string in the work area consisting of the statement label and the statement (continuation symbols, if present, are not retained). In addition to building the character string, the executive constructs a table (WKATINU) of displacements from the beginning of the character string and line numbers for the lines of the statement. The character string is used by the get-character routines. The WKATINU table is used by the executive in preparing input to IPDER.

The environmental system may supply a buffer chain of FORTRAN source lines with the initial call; it is expected to supply a new buffer chain upon intermediate entry after return code 0 or 4. For the above entries, the executive scans the buffer chain for the initial line of a source statement. During this scan, comment lines are ignored; continuation lines (which can only be recognized as such in standard-form source) cause the executive to send an error message to the system. When an initial line is found, the executive moves the source characters of that line into the character string, constructs an entry in the WKATINU table, and continues to collect continuation lines for the statement. The character string and WKATINU table are complete when the last line of the statement has been found. For standard-form source, the last line of a statement cannot be recognized as such until the initial line of the next statement is found. For free-form, the last line of a statement does not have the continuation symbol present in the last position of the line.

If buffer chain span mode is specified (options word, byte 3, bit 3=1), the executive does not assume that a buffer chain contains complete statements, i.e., that the end of the buffer chain coincides with the end of the last statement in the buffer chain. Rather, if the end of the buffer chain is reached before the statement's last line is found, the executive will return to the environment with return code 12. The environmental system is expected to use the executive's intermediate entry after return code 12 to supply a new buffer chain that will be treated as an extension of the previous buffer chain. The executive can then continue to collect lines of the last statement of the previous buffer chain. By specifying a buffer chain address of zero upon entry after return code 12, the environmental system can direct the executive to process the last statement of the previous buffer chain as complete.

The excutive performs special processing while collecting the lines of a statement. A continuation indicator is saved for later use by the checker: the contents of column 6 of the last line of a standard-form statement is saved; the continuation symbol of a free-form statement is saved, if present. The twenty-first line of a statement causes the executive to send a message to the system and to flush the remaining lines of the statement. For free-form source, no more than 1320 characters plus the number of digits in the label if present (maximum of five) are moved into the character string; a longer statement is diagnosed as a syntax error. For standard-form source, the first embedded comment line within a statement causes the executive to send an error message to the system.

When all the lines of a statement have been collected, the executive scans the statement label for errors; if a label is present, an indicator is set for later use in the checker.

At this point, the source statement is ready to be sent on to the checker, IPDSNCKR. The executive calls this segment of the module as a subroutine with the standard OS linkage and register 1 pointing to a list of the following five parameters: the beginning source pointer, the end source pointer, the address of the language definition table needed, the address of the work area obtained for checking, and the address of the options word.

The checker returns to the executive when the source statement has been completely checked or each time an error is found. If the statement is completely checked and if there is no error message to be issued, the executive determines if there is another record in the buffer chain. If there is not, the executive returns to the environmental system with a return code of zero, which is a request for a new buffer chain. If there is another statement or beginning of a statement in the buffer, the executive calls the checker to scan the new statement.

If an error is found, the checker sets up an error message code, turns on a checker-detected-error switch (WKACERSW), saves its own pointers for possible recall, and returns control to the executive. The executive calls the error processor, IPDER (entry point IPDERERR), to construct the appropriate error message. Upon return from IPDER, the executive places the address of the error message into the second word of the communications area for the environmental system. Furthermore, the executive determines whether it will need a new buffer chain of source input. If further checking can be performed on the same statement (error code indicated a non-terminating error), the executive will "save its place" and return to the environmental system with a return code of 8. In that case, it is expected that the environmental system will recall the executive to "continue checking buffer contents," and the executive, in turn, will recall the checker (WKACERSW still on) to continue checking the statement diagnosed. If the executive determines that the error should terminate checking of the statement (the error code is terminating), it resets WKACERSW so that only a new statement can be checked by IPDSNCKR. Then it must examine its current buffer chain to determine whether the chain may contain another statement to be checked. If there is another record in the current buffer chain, the executive again "saves its place" and returns to the environmental system with a return code of 8; if the erroneous statement is at the end of the buffer chain, a return code of 4, which requests a new buffer chain, is passed to the environmental system. In either of these two cases, when the executive is recalled, it provides IPDSNCKR with a new statement to check.

CHECKER (IPDSNCKR)

The checker edits a FORTRAN statement (exclusive of the statement label previously checked by the executive) for syntactic errors by matching the source statement against a table that defines the syntax for FORTRAN IV statements.

The metalanguage notation rather than its assembler language equivalent will be used to refer to the contents of the syntax definition table, e.g., syntactic line (rather than table entry), operator (rather than op-code), alternatives, optional items, etc.

Calling Sequence

```
LA          1,WKACKPRM

L           15,=V(IPDSNCKR)

BALR        14,15
```

Where, in the work area acquired by the executive, there is:

```
WKACKPRM   DS   0F

           DS   A(First Source Character)

           DS   A(Last Source Character)

           DS   A(Definition Table)

           DS   A(Work Area)

           DS   A(Options Word)
```

Operator Interpretation

The checker starts at the beginning of the syntax definition table, interprets the first operator, and branches to the routine associated with that operator. The path of checking to be followed for a particular statement is determined by this and subsequent operators. There are two types of operators, those that influence the path through the definition table as a function of the source, and those that do not. Among the operators in the first category are all those that test source characters and those that control the testing of optional or alternate items. In the second category are the operators that define a message code and cause nesting to another syntactic line.

Source Character Management

The get-character routines are called to get the next n characters (CKRGTANY) or the next n nonblank character(s) (CKRGTNB1,CKRGTNBS). On return, those routines provide the source location of the first character supplied and the source location beyond the last character supplied, in addition to the characters themselves. If the characters obtained satisfy an operator, the checker updates the source pointer beyond the last character supplied. That value of the source pointer is now used the next time a get-character routine is called. However, if the characters do not satisfy the operator, the checker will not update the source pointer; the next time a get-character routine is called that source will be supplied again to be tested against an alternative definition. Figures 3 and 4 illustrate scanning logic.

In general, an operator that fails does not advance the source pointer. An exception is the not-literal (¬'aa...a') operator which advances the source pointer beyond a character string that matches the literal and then fails. The name(N), numeric constant(K), and statement number (S) operators may be satisfied by, and advance the source pointer beyond an excessively long name or number, after diagnosing the error.

If a definition that is committed (no alternatives allowed) fails to be satisfied, the checker returns control to the executive directing it to issue an error message. When applicable, the printed message indicates the point of failure by supplying a string of a maximum of 6 source characters. If the error is not terminating, the executive recalls the checker to continue checking source at the next available source character. If the error is terminating, no further checking of the statement is performed. The executive will call the checker again only if there is another statement to be processed.

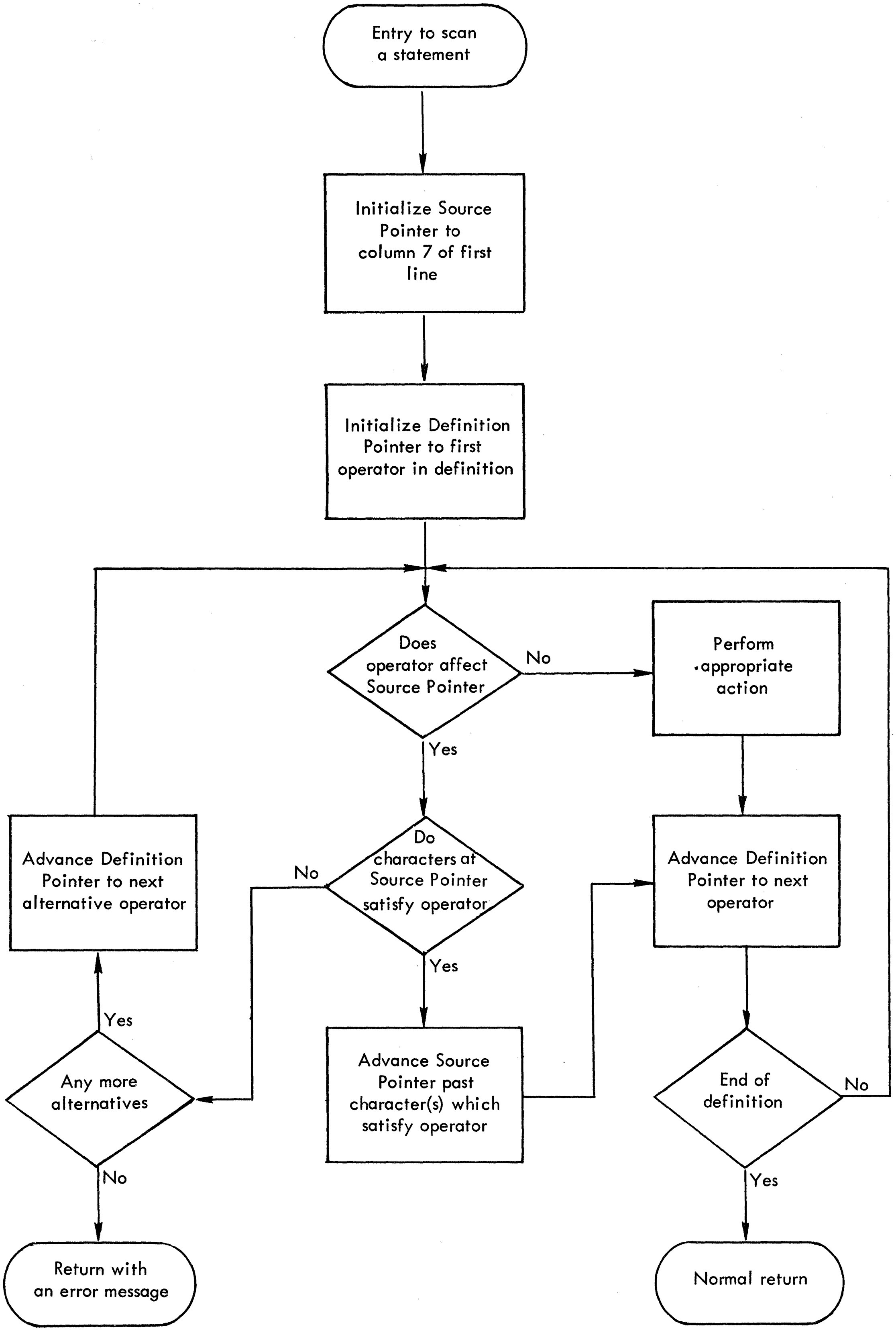
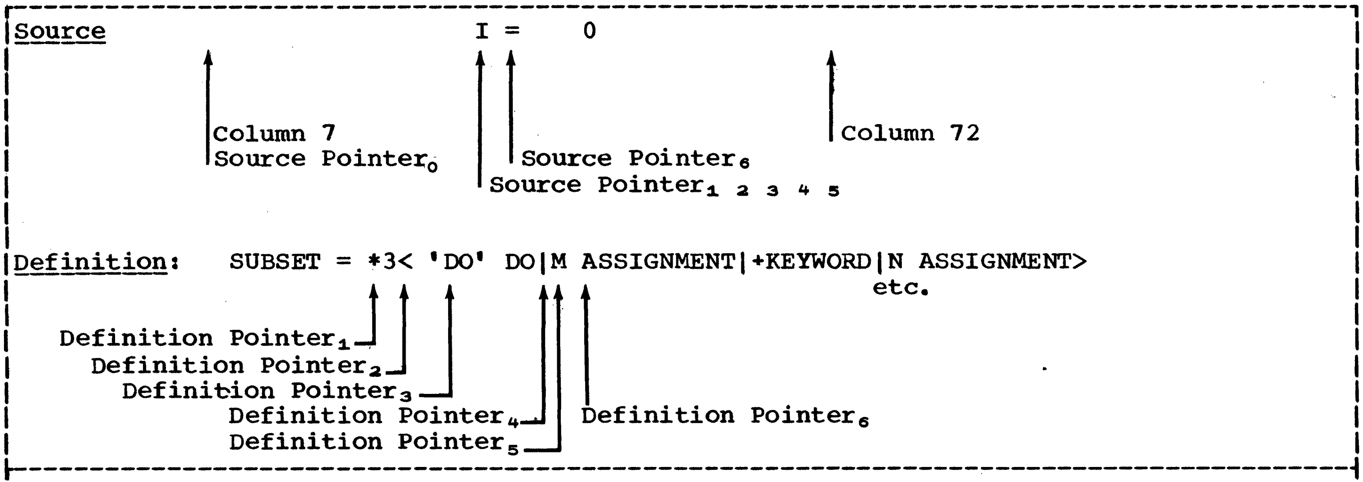

Entry to scan a statement
Initialize Source Pointer to column 7 of first line
Initialize Definition Pointer to first operator in definition
Does operator affect Source Pointer
No
Perform •appropriate action
Yes
Do characters at Source Pointer satisfy operator
No
Yes
Advance Definition Pointer to next alternative operator
Advance Definition Pointer to next operator
Yes
Any more alternatives
No
Advance Source Pointer past character(s) which satisfy operator
End of definition
No
Yes
Return with an error message
Normal return

Figure 3. Flow of Source-Definition Scan



Note: The pointer subscripts refer to the position of the pointer at the beginning of the corresponding step.

Step 0. Beginning of scan. The source pointer is advanced to the 1st nonblank character, "I".

Step 1. Message "3" is established as the current message; the source is not tested and the source pointer is unaffected; the definition pointer is advanced to the "<" start-of-alternatives.

Step 2. A qualification entry is built and the definition pointer is advanced to the 'DO' literal.

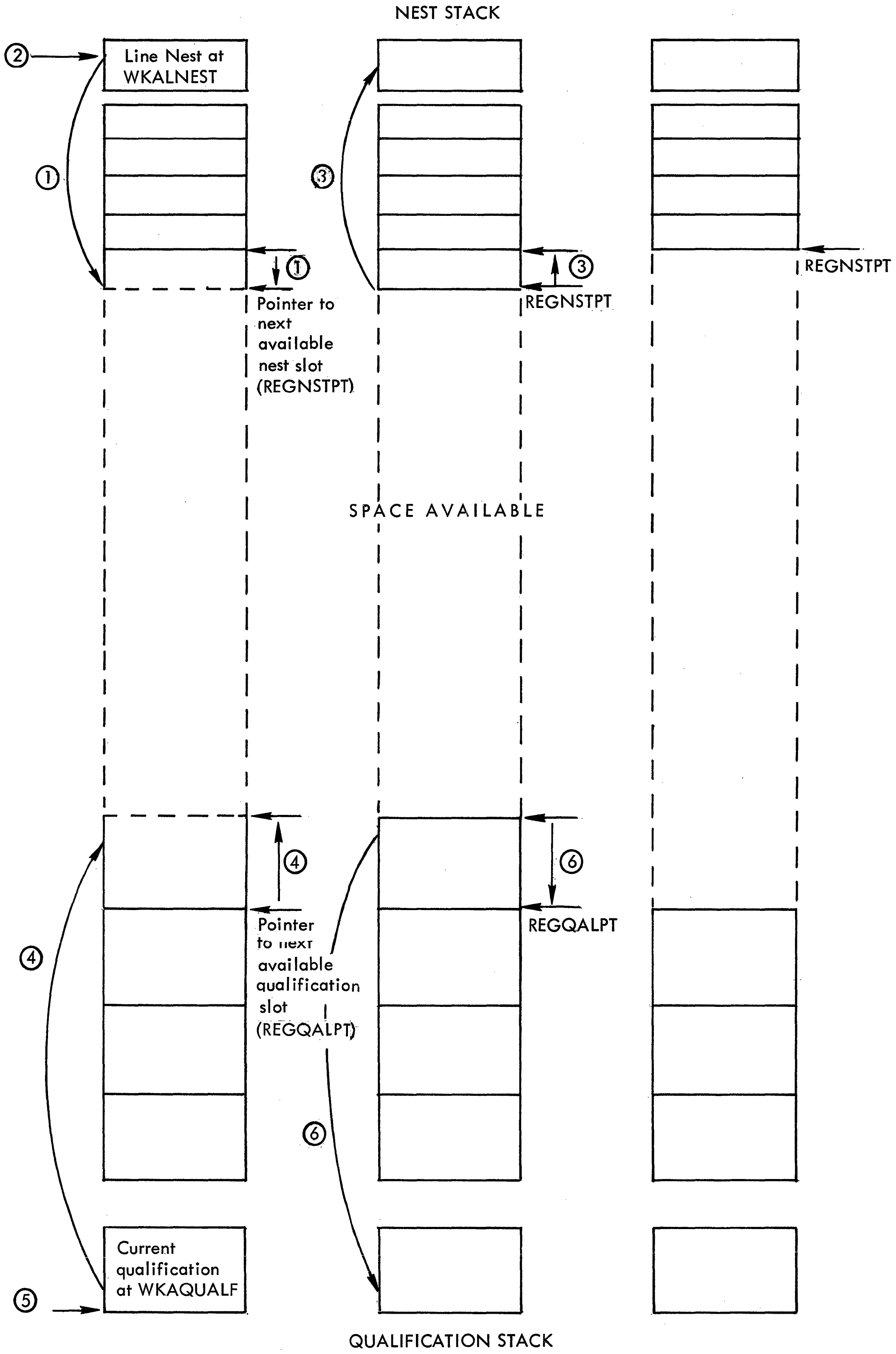
Step 3. Source "I" does not satisfy the 'DO' literal operator; the source pointer is not advanced but the definition pointer is skipped to the "|" operator.

Step 4. Because the literal operator in the first alternative was not satisfied, i.e., it produced an F, the definition pointer is advanced to the first operator of the second alternative, the "M" operator.

Step 5. Source "I" satisfies the "M" name operator; the source pointer is advanced to the "=" and the definition pointer is advanced to the "ASSIGNMENT" symbolic-line operator.

Step 6. Nesting to line ASSIGNMENT takes place: definition pointer is saved in the nest list, and the current definition pointer is reset to point to the first operator of the ASSIGNMENT line.

Figure 4. Example of Source-Definition Scan


NEST STACK
②
Line Nest at WKALNEST
①
③
①
③
Pointer to next available nest slot (REGNSTPT)
REGNSTPT
REGNSTPT
SPACE AVAILABLE
④
⑥
Pointer to next available qualification slot (REGQALPT)
REGQALPT
④
⑥
Current qualification at WKAQUALF
⑤
QUALIFICATION STACK

Nesting

When the symbolic name of another syntactic line is encountered in the definition, ① the current line nest is pushed down, and ② a new nest is built for the line named.

Unnesting

When the end-of-line operator is reached, ③ the previous line nest is popped up to become the current line nest again

When a ⟨ , left brace, or (, left parenthesis, is encountered in the definition, ④ the current qualification entry is pushed down, and ⑤ a new qualification entry is built in the current qualification slot.

When a ⟩ , right brace, or), right parenthesis, is reached. ⑥ the previous qualification entry is popped up to become the current qualification entry again.

Figure 5. Operation of the Pushdown Stacks

Line Nesting

The logic of the checker can be outlined in terms of definition lines and alternatives on those lines. A symbol encountered on the right side of a definition line that is itself defined on the left side of some other definition line causes "nesting" to that other line. When the operators on the line "nested to" have been satisfied, the checker "unnests" back up to the point immediately beyond the symbol that caused the nesting. Any number of levels of nesting may be required to match the source against a symbol: line "nested to" may cause nesting to another line, which in turn may also nest to some other line, etc. Recursive nesting, i.e., nesting to the same line, (whether directly or indirectly), is also a possibility. The nests are kept in a push down list in which the last entry made in the list is the first entry out of the list (LIFO). Refer to Figure 5 for a schematic diagram illustrating the nesting process.

Qualification

On any given line there may be alternative definitions and even definitions of items that are optional, but whose presence in the source statement must be recognized. For this discussion, a qualification is defined as all the operators enclosed in <> or (). A qualification enclosed in <> comprises a series of alternative definitions separated by the | operator and is therefore known as an alternative qualification. A qualification enclosed in () is an optional definition, and is known as a parenthetical qualification.

Qualifications of either type require information to be saved from the left qualification symbol to the corresponding right symbol. Since a qualification may be enclosed within any number of encompassing qualifications (on the same or on earlier lines), e.g., (<L|D>). Qualification information is also kept in a push down list (LIFO). The nest and qualification lists grow towards each other to optimize space availability. On encountering a < or (in the definition, the previous qualification entry is placed on the top of the qualification list ("pushed down") and a new qualification entry is built. When a > or) is reached, a qualification entry is removed from the top of the list ("popped up") to replace the most recently built qualification entry. So long as <>s and ()s are properly paired in the definition, a right qualification symbol will reference the information from its corresponding left qualification symbol. See Figure 5 for a schematic design illustrating how the qualification stack is used.

Procedure

When the checker gets control, it determines whether it is being called to start checking a new statement or to continue checking a statement in progress.

Upon starting a new statement, source and table definition pointers are obtained from the parameter list, and work areas and switches are initialized. The top line of the definition is automatically committed (since there are no more alternatives).

To start off the checking, the CKRSYNS, nest to symbolic line, operator routine (Chart 013) is given control to push down the nest information into its list, start the syntactic definition pointer at the top line, and set up nest information for the new line (level number = 1).

Once the syntactic definition pointer is set, control is transferred to CKRINTRP, the syntactic interpreter (Chart 009), to use the syntactic operator code from the definition to locate and branch to the corresponding operator routine. Errors are detected by operator routines and are normally handled by the CKRFAIL routine (Chart 035), which is detailed later. However, certain operators are complex enough to be a time-saving substitute for nesting to another syntactic line. The implied nesting may have implicit error message(s) associated with it. Detecting a particular error by an operator routine can cause a specific error message to be issued and allow processing to continue as if no error had occurred.

When the checker is recalled to continue checking a statement for which a diagnostic has been issued, the checker restores its pointers and continues processing just as if no error had been detected.

Operators

The syntactic operators and their corresponding routines are described below. Unless otherwise noted, the routine branches to CKRINTRP with an updated definition pointer at completion of its functions.

<, Start of a series of alternatives (CKRLBRCE, Chart 010)

The qualification information is pushed down into the qualification list (see Figure 5). The "false" and "true" displacements to the next | operator and to the corresponding > operator respectively, as well as the source pointer, the updated definition pointer, and the line nest level number, are saved as current qualification information. The qualification iteration count is set to zero and the commit switch is turned off.

>, End of a series of alternatives (CKRRBRCE, Chart 010)

If the last alternative failed, the source pointer is backed up, the qualification list is popped up (see Figure 5) and control is transferred to CKRFAIL, the checker's general error routine. If the last alternative attempted was successful, the qualification list is popped up, and the next syntactic operator will be interpreted at CKRINTRP.

|, Separator of alternatives (CKROR, Chart 010)

If the last alternative attempted was successful, the definition pointer is updated from the qualification "true" displacement to point to the > operator that terminates this series of alternatives. If the last alternative tested failed, the indication of failure is removed (since there is still another alternative to be tested which may be successful), the source pointer is backed up to try an alternative definition against the same source, and the qualification "false" displacement is updated to the next | operator or to the associated > operator if this is the last | operator within it.

(, Start of an optional definition (CKRLPARN, Chart 010)

Same as the < operator, except that only "false" displacement is meaningful and that is the displacement to the associated) operator.

), End of an optional definition (CKRRPARN, Chart 010)

If the optional definition was not satisfied, the indication of failure is removed and the source pointer is backed up to test the same source against the next operator.

Whether the definition was satisfied or not, the number of iterations performed (qualification iteration count) is saved for possible action routine use, and then the qualification list is popped up before the next operator code is interpreted.

/, Commit to this alternative or optional definition (CKRCOMIT, Chart 011)

The commit switch associated with the current qualification information is turned on.

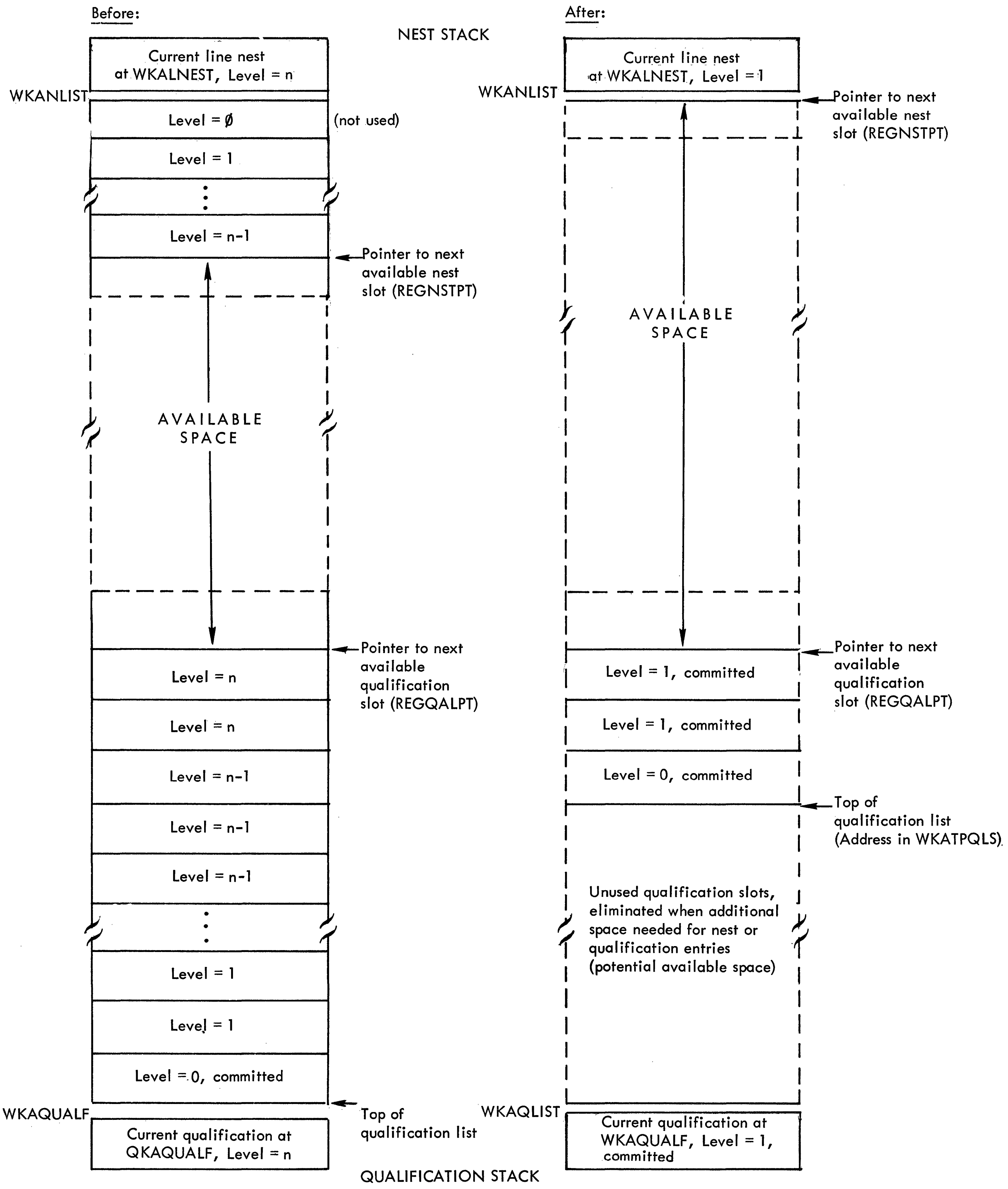

Before:
After:
NEST STACK
Current line nest at WKALNEST, Level = n
WKANLIST
Level = ∅
(not used)
Level = 1
Level = n-1
Pointer to next available nest slot (REGNSTPT)
AVAILABLE SPACE
Pointer to next available qualification slot (REGQALPT)
Level = n
Level = n
Level = n-1
Level = n-1
Level = n-1
Level = 1
Level = 1
Level = 0, committed
WKAQUALF
Top of qualification list
Current qualification at QKAQUALF, Level = n
Current line nest at WKALNEST, Level = 1
WKANLIST
Pointer to next available nest slot (REGNSTPT)
AVAILABLE SPACE
Pointer to next available qualification slot (REGQALPT)
Level = 1, committed
Level = 1, committed
Level = 0, committed
Top of qualification list (Address in WKATPQLS)
Unused qualification slots, eliminated when additional space needed for nest or qualification entries (potential available space)
WKAQLIST
Current qualification at WKAQUALF, Level = 1, committed
QUALIFICATION STACK

Figure 6. Effect of Statement Commit on Pushdown Stacks

:, Commit to this type of statement (CKRSTCMT, Chart 011)

The statement global commit switch is turned on. The commit switches (see / above) are turned on in all qualification entries associated with this line (all entries having the nest level number of the current line). The current line becomes level 1. Figure 6 illustrates the effect of : on the nest and qualification lists.

..., Indefinite Iteration (CKRITIND, Chart 012)

This operator routine can be reached only after a successful iteration. The qualification iteration count is incremented for possible action routine use. The new source pointer value is saved as qualification information, the qualification commit switch is turned off, and the syntactic definition pointer is set to point back to the operator immediately following the qualification left parenthesis.

.n., Definite Iteration (CKRITDEF, Chart 012)

The qualification iteration count is incremented and compared to n. If fewer than n iterations were checked, processing continues as for indefinite iteration (saving the new source pointer, etc.). The nth iteration gets us to the right parenthesis ending the iteration loop.

SYMBOLIC-NAME, nest to a syntactic line (CKRSYNS, CKRSYNST after a table, Chart 013)

The current line nest is placed at the top of the nest list (i.e., "pushed down" into the nest list; see Figure 5). The syntactic definition pointer, updated to point beyond the symbolic-name, is saved in the nest. The level of nesting is incremented for the new line, and a pointer to the new line is placed in the nest. The syntactic definition pointer is now set to point to the new line definition (first operator after = sign).

M, Check source for FORTRAN name that may or may not be present (CKRMNAME, Chart 014)

If the first nonblank source character examined is not alphabetic, or if six alphameric characters are found following the first alphabetic, this test fails and control is transferred to CKRFAIL.

When this test succeeds, the source pointer is updated beyond a satisfactory FORTRAN name, i.e., an alphabetic followed by from zero to five alphamerics.

N, Check source for expected FORTRAN name (CKRNAME, Chart 014)

If the first nonblank source character examined is not alphabetic, this test fails and control is transferred to CKRFAIL.

If the first character is alphabetic, the source is assumed to be a name, and the source pointer is advanced until a non-alphameric character is detected. If the number of characters (initial alphabetic plus subsequent alphamerics) exceeds six, control is returned to the executive with an error indication of name too long. When the executive recalls the checker after recording the error, processing continues with the syntactic operator following N and that first non-alphameric character detected, just as if the name were a valid length.

L, Check source for alphabetic (CKRLETTR, Chart 015)

If the next nonblank source character is an alphabetic character (A through Z) or the character $, the test is successful and the source pointer is updated beyond the letter obtained. Otherwise, the test has failed, and control is transferred to CKRFAIL.

D, Check source for digit (CKRDIGIT, Chart 015)

If the next nonblank source character is a digit (0 through 9), the test has succeeded and the source pointer is updated. Otherwise, control is transferred to CKRFAIL.

A, Check source for alphameric (CKRALMER, Chart 015)

If the next nonblank source character is alphameric (A through Z or $ or 0 through 9), the test has succeeded and the source pointer is updated. Otherwise, the test has failed, and control is transferred to CKRFAIL.

K, Check source for a number in FORTRAN real or integer form (CKRNUMBR, Chart 016)

Definition of terms

leading-zeros count - number of zeros before the first nonzero digit or number of zeros before the decimal point, whichever is less, i.e., "insignificant zeros."

digit count - number of digits including and after the first nonzero digit or number of digits after a decimal point, whichever is greater, i.e., number of significant digits.

zero count - number of zeros after the decimal point. Zero count is equal to zero in the case of no decimal point. It includes all zeros, not just leading zeros.

Notes:

1. The total number of digits equals the leading-zeros count plus digit count. FORTRAN uses this to determine length of basic real constants.
2. Any number is zero if the digit count equals the zero count.

ten's power - computed for real constants, ignoring any associated exponent; the smallest integer greater than or equal to the logarithm to the base 10 of a basic real constant or integer. The decimal position of the leftmost nonzero digit is used to compute ten's power, e.g., 50.38 or 74, ten's power equals 2; -.4, ten's power equals 0; 0.0001, ten's power equals -3.

type bit - indicates Real (0) or Integer (1).

length bit - indicates E (0) or D (1) length. (E for integer).

value bit - indicates zero (0) or nonzero(1).

Type bit, length bit, and value bit can occur in the following combinations with the meanings given in Table 3.

Table 3. K Operator Bit Combinations

Bit Combinations			
Type bit	Length bit	Value bit	Meaning
1	0	0	integer length 4 zero
1	0	1	integer length 4 nonzero
0	0	0	real length 4 zero
0	0	1	real length 4 nonzero
0	1	0	real length 8 zero
0	1	1	real length 8 nonzero

The combinations 110 and 111 will not occur if the K operator takes the "true" return. The combination 111 (nonzero integer of length 8) is therefore set when the K operator takes the "false" return (not a numeric constant).

Procedure

The leading-zeros count, digit count, and zero count are initialized to zero.

If the first nonblank character obtained from the source is neither a digit (0-9) nor a decimal point, the K switches are set to indicate length D integer (not a number) and the test fails with a branch to CKRFAIL.

If the first source character is a digit greater than zero, the digit count is incremented by one. If that first character is zero, the leading zero count is incremented by one. If the first character is a decimal point, control transfers to CKRDECPT.

After the first digit (zero or nonzero) is obtained (and if no decimal point has been found yet), subsequent nonblank characters are treated as follows:

1. 1 through 9: digit count is incremented by one, next character is obtained and examined;
2. zero: if digit count is still zero, leading-zero count is incremented by one; if digit count is greater than zero, it is incremented by one; in either case, the next character is examined;
3. decimal point: control transfers to CKRDECPT;
4. E or D: length switch is set appropriately, ten's power is set from digit count, and control is transferred to CKREXPON;
5. any other character: integer processing follows at CKRINTEG.

CKRINTEG. The number is assumed to be an integer. If its magnitude exceeds 2,147,483,647, control is returned to the executive with an error indication of integer too large. When the executive recalls the checker, processing continues as if the integer were a valid length. Type and length switches are set for a valid integer (length E), and the value switch is set according to whether the number has all zero digits or not. These switches are set for possible use by action code routines. The operator beyond K can now be interpreted (CKRINTRP).

CKRDECPT. Ten's power is set from digit count. If "OR," "AN," or a relational operator (e.g., "EQ," "GT") follows the decimal point, the point is taken to be part of the operator, and the source pointer is backed up to that point. If no digits (zero or greater) were encountered prior to the operator, the numeric test fails, the K switches are set to indicate length D integer (not a number) and control is transferred to CKRFAIL. If any digits did precede the operator, those digits are treated as an integer and control is transferred to CKRINTEG.

If the decimal point is not followed by a logical or relational operator, characters beyond the point are handled as follows:

For each consecutive digit, digit count is incremented by one; zero count is also incremented for each zero digit, and if ten's power was never greater than zero, it is decremented for each zero until a nonzero is encountered. When a character is encountered that is not a digit, a check is made that there are some digits before or after the decimal point. If no digits precede or follow the point, and the definition is not committed, the numeric test fails (K switches, not a number), and control is transferred to CKRFAIL; if there are no digits, but the definition is committed, control is returned to the executive with an error indication that a real constant must have at least one digit, and when the executive recalls the checker, processing continues at CKRRDORE, as if there were some digits.

CKRRDORE. If the source character is E (single-precision) or D (double-precision), the length switch is set appropriately and control is transferred to CKREXPON. If the source character is not E or D, the length switch is set to E, and processing continues at CKRCKSIZ.

CKREXPON. The type switch is set for read, and the exponent is evaluated as a signed number. Processing continues at CKRCKSIZ for range testing.

CKRCKSIZ. If the algebraic sum of ten's power and exponent value (exponent value alone if the number is zero) is less than -78 or greater than 76, control is returned to the executive with the appropriate error indication. When the executive recalls the checker, processing continues at CKRREAL, just as if the exponent were an acceptable value.

CKRREAL. The type switch is set for real, and the value switch setting is determined by whether all digits (excluding an exponent) were zero or not. Any invalid decimal points or extraneous exponents are diagnosed. The operator beyond K may now be interpreted (CKRINTRP).

S, Check source for a statement number (CKRSTATM, Chart 017)

If a sign (+ or -) is present, an error indication is saved and the sign is bypassed.

If the next nonblank source character is not a digit, this test fails with a transfer of control to CKRFAIL. Leading zeros are ignored and significant digits counted until a non-digit is encountered in the source. At that time, if the number of significant digits is between one and five inclusive and no sign was encountered, the test is successful. Otherwise, control is returned to the executive with an error indication of invalid statement number. When the executive recalls the checker, processing continues at CKRINTRP, just as if a valid statement number had been found.

H, Check source for the wH form of a literal constant or a literal format code (CKRHOLLR, Chart 018)

Starting with the first nonblank source character, consecutive digits, as they are encountered, are converted to a width value. If the first character is not a digit, or if an H is not obtained after the width, the test fails and control is transferred to CKRFAIL. However, once wH is found, the source is assumed to be a literal constant or a literal format code.

If w is zero or greater than 255, control is returned to the executive with an error indication to that effect. When the executive recalls the checker, if w was zero, processing continues with interpretation of the next operator; otherwise, processing continues as for a valid width.

If there are fewer than w characters remaining in the source statement, control returns to the executive with a terminating error indication of incomplete literal field. Otherwise, the source pointer is spaced w characters (including blanks) after the H.

C, Check source for a character string in single quotes (CKRCSTRG, Chart 019)

If the first nonblank source character obtained is a single quote, the source is assumed to be a character string. Otherwise, the test fails and control is immediately transferred to CKRFAIL.

In counting characters in the string, blanks are included. Two successive single quotes are counted as one character. The string is successfully ended when a single quote not followed by another quote is encountered. If the source ends or more than 255 characters are counted before the closing single quote is found, or if the character string was empty, control is returned to the executive with an appropriate terminating error indication.

'aa...a'. Check source for the presence of the literal quoted (CKRQUOTE, Chart 020) and

¬'aa...a', Check source for the absence of the literal quoted (CKRNOTQT, Chart 020)

The next length-of-literal nonblank characters in the source are gathered and compared to the literal given. If the literal is matched, the source pointer is updated beyond the last character obtained.

The test fails with a transfer of control to CKRFAIL (1) if there is a match and ¬ 'aa...a' is the operator or (2) if there is not a match and 'aa...a' is the operator.

The test is successful (1) if there is a match and 'aa...a' is the operator or (2) if there is not a match and ¬ 'aa...a' is the operator.

&a, Scan for the presence of the argument a (CKRSCAN, Chart 021) and

&¬a, Scan for the absence of the argument a (CKRSCANF, Chart 021)

The remainder of the source is scanned for the given character. The scan terminates when the character is found.

&a succeeds if the character is found before the end of the source; &¬a succeeds if the end of the source is reached without the character being found. The source pointer is unaffected by the scan. If the character is not found and &a is the operator, or if the character is found and &¬a is the operator, the test fails and control is transferred to CKRFAIL.

$n, Call action routine N (CKRACTN, CKRACTNT after a table, Chart 022)

The appropriate special-purpose subroutine is called to perform a specific test or task, identified by n. The individual action code routines and their returns to the checker are described in "Action Code Routines."

*n, Define an error message (CKRMESSG, Chart 033)

The error message code n is saved as part of the nest information, where it can be accessed by the CKRFAIL routine in processing an error.

+TABLE-NAME, Check source for one of the literals in the table, and

-TABLE-NAME, Verify that source does not appear in the table (TABL, TABLT after a table, Chart 033)

A number of nonblank characters equal to the maximum-argument-size is obtained from the source. The length-of-argument characters are compared to each literal argument in the table until either a match is found or the table is exhausted.

If there is no match and the operator is -TABLE-NAME, the test is successful. The syntactic definition pointer is incremented so that the operator following the first table reference in the chain will be interpreted next. If there is no match, but the operator is +TABLE-NAME, the test fails and control is transferred to CKRFAIL.

If there is a match, the source pointer is incremented beyond the matching source characters. If the operator is -TABLE-NAME, the test fails and control is transferred to CKRFAIL. If the operator is +TABLE-NAME, the test has succeeded and the function associated with the matched argument is examined. An operator routine invoked as a table function is entered at a special point (operator-routine-name suffixed by T) to account for the different definition pointer used. If the function is:

1. SYMBOLIC-NAME, the syntactic line named is nested to at CKRSYNST;

2. +TABLE-NAME, the table named is searched at CKRTABLT;

3. $n, the action code n is processed at CKRACTNT;

4. 0(null action) or if the last table function has been successfully processed, the syntactic definition pointer is incremented so that the operator following the first table reference is examined at CKRINTRP.

Once some argument in a table is matched, even if tests fail at the function level, no other arguments in the table are checked against the source.

END-OF-DEFINITION-LINE, Unnest syntactic line (CKRSYUNS, Chart 034)

If the current line nest is at level 1, the definition has been satisfied and control is to be returned to the executive. The presence of any excess source character(s) is diagnosed, unless a prior error was detected in the statement.

At any subsequent level of nesting, the syntactic definition pointer is restored from the nest to point immediately beyond the symbolic reference to this line on an earlier line. Then a line nest is removed from the top of the nest list ("popped up") to replace the current nest (see Figure 5).

All operators valid for interpretation have been discussed. Failure in an operator routine often causes control to be transferred to CKRFAIL, the failure routine, which operates as follows:

CKRFAIL, Failure routine, (Chart 035)

A switch is turned on to indicate failure in an operator. If we are committed to this path, i.e., if the commit switch is on for the current qualification, the error message code is set from the current line nest and the error source reference is set to the source location of the last character (or character string) obtained. At CKRREINT the branch register is set to return control to CKRINTRP in case the checker is recalled to continue checking beyond the failing element. At CKRTSTML, if the end-of-source has been reached, an error code bit is set on to indicate a terminating error. At CKRTMRET, the error message information is compared to that of any previous message issued for the current statement. If the new error information is unique, it is saved, and the error will be recorded (up to a maximum of five checker-detected errors per statement) via the executive; processing continues at CKRERRET. If identical error information was already issued, and the error is still not a terminating error, processing continues as though the executive had issued the message and recalled the checker; and the checker continues its scan of the same statement.

If an identical error message was issued, but the error is now a terminating error, control is returned to the executive without indication of error. At CKRERRET registers are saved for reloading after possible recall by the executive, the checker-detected error switch is turned on, and control is returned to the executive.

If we are not committed to this path, any necessary unnesting takes place to bring the level of line nesting back up to the level of the current qualification. The syntactic pointer is reloaded from the qualification "false" displacement so that the next alternative operator, | , or qualification end, > or), may be interpreted.

Action Code Routines (Charts 023-032)

The action code ($n) operator routine initializes for no errors and then gives control to the appropriate action code subroutine (an unrecognizable action code constitutes a system or syntax checker failure--WKASFAIL set).

Upon completion of the action subroutine, control is returned to the main body of the checker where error indicators are tested and control is transferred appropriately:

1. Failure switch (WKAFALSW) on -- Branch to the CKRFAIL routine to determine whether an error message is to be issued.

2. Error code set (WKAERRCD not zero) -- If committed, branch to CKRREINT (in the CKRFAIL routine) to issue an error message before proceeding to interpret the next operator (at CKRINTRP). If not committed, branch to CKRFAIL to try the next alternative.

3. Not-operational byte set (WKASFAIL, in work area IPDERWKA, not zero) -- branch to CKRERRET (in the CKRFAIL routine) to issue a system or syntax checker failure message and discontinue checking the statement.

4. None of the above -- branch to CKRINTRP to interpret the next operator.

100-106. K operator action routines

Action routines 100 through 106 perform tests on fields and switches set by the K operator. The K digit count contains the number of significant digits counted by the K operator. The K switches are: type, which may be integer or real; length, which may be D (double-precision) or E (single-precision); and value, which may be nonzero or zero. If type = integer and length = D, indicating that the absence of a numeric constant was previously diagnosed, control is immediately returned to the checker.

100. Nonzero integer (CKRAR100, Chart 023)

Switch settings are examined for type = integer and value = nonzero. If either of the above requirements is not met, control is returned to the checker with an error code indicating that a nonzero integer is required.

101. Nonzero number (CKRAR101, Chart 023)

The switch settings are checked for value = nonzero. If value = zero, control is returned to the checker with an error code indicating that a nonzero number is required.

102. Integer (CKRAR102, Chart 023)

Switch settings are checked for type = integer. If type = real, control is returned to the checker with an error code indicating integer required.

103. Save K operator switches (CKRAR103, Chart 024)

This routine is called only for complex numbers. The switches set by the K operator for the real portion of the complex number are saved for use in routine 104.

104. Complex number checker (CKRAR104, Chart 024)

The K operator switches that have just been set for the imaginary portion of a complex number and those switches saved by routine 103 are tested to be sure that type = real for both and that both have the same length attribute (E or D). If either requirement fails to be met, an error code indicating invalid complex number is returned to the checker.

105. Data set reference number (CKRAR105, Chart 024)

Switch settings are checked for type = integer and value = nonzero and the digit count is checked for not being greater than two. If any of the above requirements is not met, control is returned to the checker with an error code indicating an invalid data set reference number.

106. Real number (CKRAR106, Chart 025)

Switch settings are checked for type = real. If type = integer, control is returned to the checker with an error code indicating real number required.

200-202. Check number of subscripts

200. Possibly too many subscripts precede (CKRAR200, Chart 026)

This routine checks the iteration count saved after an iteration loop has been completed. If the saved iteration count exceeds 2 for FORTRAN E, or 6 for any other level of FORTRAN, control is returned to the checker with an error code warning that possibly too many subscripts precede the source characters (")") pointed to.

201. Too many subscripts (CKRAR201, Chart 026)

This routine checks the iteration count of the current iteration loop. If the current iteration count is exactly 2 for FORTRAN E, or 6 for any other level of FORTRAN, control is returned to the checker with an error code indicating too many subscripts at this point ("," following last valid subscript).

202. Too many subscripts precede (CKRAR202, Chart 026)

This routine is identical to 202 with the exception that the error code indicates that too many subscripts definitely precede the source characters (")") pointed to.

300. Check END statement (CKRAR300, Chart 027)

If the statement was labeled (with a statement number) or if there was a nonblank character in the continuation column or if, in free-form, the statement contains more than 66 characters (not including trailing blanks), control is returned with a code indicating a terminating error.

301. Check FORMAT statement for label (CKRAR301, Chart 027)

If the statement was not labeled (with a statement number), control is returned with an error code indicating that the statement number is missing.

400. Check for DEBUG facility (CKRAR400, Chart 028)

If the source is not being checked for FORTRAN G, G1, or Code and Go syntax, control is returned with a code indicating a terminating error. The code indicates that the Debug facility is not supported.

401. Check for List-Directed I/O support (CKRAR400, Chart 028)

If the source is not being checked for FORTRAN G1 or Code and Go syntax, control is returned with a code indicating a terminating error. The code indicates that List-Directed I/O is illegal.

500. Check range of IMPLICIT statement (CKRAR500, Chart 029)

The next nonblank characters in the source are checked for an alphabetic character not followed by a hyphen or for an alphabetic character followed by a hyphen followed by a higher alphabetic character, where the character $ is considered alphabetically greater than Z. If the source does not meet the requirements, control is returned with an error code indicating that an invalid range was specified in an IMPLICIT statement.

600-603. Check I/O lists

Action routines 600-603 are used in the checking of I/O lists:

600-602 to verify that the index variable of an implied DO is not subscripted;

603 to look ahead to verify that a right parenthesis closes an implied DO.

600. Initialize for no subscripting (CKRAR600, Chart 030)

The subscript switch is initialized off.

601. Indicate subscripting (CKRAR601, Chart 030)

The subscript switch is set on.

602. Test for subscripting (CKRAR602, Chart 030)

If the subscript switch is on, control is returned with an error code indicating that the variable may not be subscripted.

603. Test for following right parentheses (CKRAR603, Chart 030)

If the next nonblank character in the source is not a right parentheses, control is returned with an error code indicating that a right parenthesis is required. (If the right parenthesis is detected, it is not bypassed.)

700. Check format code width specification (CKRAR700, Chart 031)

Starting with the first nonblank source character, consecutive digits are converted to a "width" value. If the first character is not a digit, control is returned to the checker with the failure switch on.

Once the first non-digit is encountered, the width value is checked for size. If the width is not in the range 1-255, control is returned with an error code to that effect, and a width value of 0 or 255, as appropriate, is saved.

701. Check format code decimal places specification (CKRAR701, Chart 032)

Starting with the first nonblank source character, consecutive digits are converted to a "decimal places" value. If the first character is not a digit, control is immediately returned to the checker with an error code indicating that decimal places must be specified.

Once the first non-digit is encountered, the decimal places value is compared to the width value saved by routine 700. If the decimal places value is greater than the width, control is returned with an error code to that effect.

800. Check for end of source (CKRAR800, Chart 032)

If there are any further source characters, the failure switch is set on before control is returned to the checker.

801. Fail unconditionally (CKRAR801, Chart 032)

The failure switch is set on and control is returned to the checker.

Get-Character Routines (CKRGTANY, CKRSKANY, CKRGTNBS, CKRGTNB1, CKRSERCH), (Charts 036-039)

These are the routines in IPDSNCKR which are used to get or skip characters in the source statements. Their inputs are a supplied source pointer, an end-of-statement pointer, a count of characters to be moved, and a character to be searched for. Their outputs are current and update source pointers, a source-end switch setting, a result buffer, and the count of characters actually moved. The routines obtain characters from the character string in the work area.

The supplied source pointer contains the address of a character in the character string. A negative source pointer indicates no more source. The supplied source pointer may be the initial source pointer passed by the executive to IPDSNCKR or any current, previous, or update source pointer value for the current statement.

The end-of-statement pointer contains the address of the last character of the statement character string.

When characters are requested to be moved, they are placed in the result buffer. When fewer characters are moved than were requested, an end-of-source character is appended to those already placed in the buffer. A request for zero characters is not legitimate and causes undefined actions.

The get-character routines will fold lower case alphabetics to upper case if free-form is specified in the options word.

The CKRSKANY and CKRGTANY routines share common code.

Get Any Source Characters (CKRGTANY, Chart 037)

Starting with the one specified by the supplied source pointer, the requested number of characters is moved to the result buffer.

The current source pointer is set equal to the supplied source pointer.

The update source pointer is set to the character after the last one moved, or it is set negative if there is no next character.

If there are not enough source characters to satisfy the request, the source-end switch is set on; otherwise, it is set off.

The count of characters moved, not including a possible end of source character, is stored.

Skip N Source Characters (CKRSKANY, Chart 037)

This subroutine skips a specified number of characters in the lines of a source statement. Skipping begins at the character specified by the source pointer and continues until the count is satisfied or the end of source is found.

If the specified number of characters are available and skipped, an update pointer is set to the source byte following the last one skipped.

This subroutine has the same specifications as CKRGTANY except that the result buffer is destroyed and the number of characters requested may be more than the length of the result buffer.

Get N Nonblank Characters (CKRGTNBS, Chart 037)

The requested number of nonblank characters is moved to the result buffer. The scan begins with the character specified by the supplied source pointer.

Upon entering the routine the current source pointer is set to zero. If there are no nonblank characters, the current source pointer remains zero. Otherwise, the current source pointer specifies the first nonblank character moved.

The update source pointer is set to the next character, blank or nonblank, after the last one moved, or it is set negative if there is no next character.

If there are not enough nonblank characters to satisfy the request, the source-end switch is set on. Otherwise, it is set off.

The count of nonblank characters found, not including a possible end-of-source character, is stored.

Get One Nonblank Character (CKRGTNB1, Chart 036)

This subroutine has the same specification as CKRGTNBS except it is understood that the request is for one nonblank character, and the number of characters moved is not recorded.

This is a separate routine from CKRGTNBS in order to increase efficiency.

Search Source for Specific Character (CKRSERCH, Chart 039)

This subroutine searches the remainder of a source statement for a specified character. The search begins at the source character specified by the supplied source pointer and continues until a match is found or the source statement end is encountered.

If a source character matches the supplied character, the update source pointer is set to that source byte, and the source end switch is set off.

If no match is found, the source end switch is set on, and the update source pointer is set negative.

THE IPDER MODULE

The IPDER module constructs diagnostic messages in the message buffer, WKAERBFR. It is used to construct all the messages sent from the IPDSN module to the environmental system. The format of these error messages is described in Section II, "Error Messages." The messages are constructed from information passed to IPDER in the area IPDERWKA when it is called, and from information in two tables, MSGTABLE and MSG000, internal to the module. The IPDERWKA information is used to generate the message identification, line number, and source character fields of the message. Tables MSGTABLE and MSG000 are used to generate the message text field as follows: The error code with its rightmost bit zeroed is used as an index into MSGTABLE which is a table of halfwords. The halfword thus obtained from MSGTABLE is a displacement from the start of MSG000, the table containing the actual message texts. The address of the text of the desired message is this displacement added to the base address of MSG000. The length of the desired message is found by subtracting its displacement (in MSGTABLE) from the displacement of the next message, i.e., from the next two bytes of MSGTABLE. The text address and length are used to move the message to the message buffer. The remainder of the buffer is filled with blanks, completing the operation of the IPDER module.

The following text and the flowcharts at the end of this section describe the executable control sections and routines that accomplish the functions of the syntax checker modules. There are three control sections containing executable code: the executive, the checker, and the error code processor. Each control section and its routines are described in the order in which they are discussed in Section II. Figure 7 shows the organization of the syntax checker modules in core storage.

EXECUTIVE

Control Section Name: IPDSNEXC (Charts 001-005)

Entry Point

The executive is called by the environmental system at IPDSNEXC when FORTRAN IV syntax checking is requested by a terminal user.

Function

IPDSNEXC performs the following functions:

- Acquires and releases a work area for the syntax checker.

- Ensures that the requested language definition table is available and in core storage.

- Gets a source statement from the buffer chain passed to it by the system and builds a statement character string in the work area.

- Checks for valid statement number.

- Passes control to the checker to syntax check a source statement.

- Recalls checker to resume syntax checking on a statement that has had a non-terminating error.

- Calls error code processor which constructs error messages when the checker finds an error in a source statement.

- Requests more source input when needed.

- Returns to caller with appropriate error messages.

Routines Called

IPDSNCKR is called to begin syntax checking a source statement or to resume checking on a statement after the executive has passed its syntactical error message to the system.

IPDERERR is called to generate an error message from an error code supplied by the executive or returned to the executive by the checker.

Exits

The executive exits to the system with an error message to be sent to the user, with a request for more input, or after the final-entry cleanup has been accomplished.

Attributes

IPDSNEXC is re-enterable. Its work area must not be modified by the environmental system.

CHECKER

Control Section Name: IPDSNCKR (Charts 006-036)

Entry Point

IPDSNCKR is called by the executive (IPDSNEXC), either to process a new statement or to continue processing a statement after an error has been recorded.

Function

The function of the checker is to edit a single FORTRAN statement for syntactic errors by matching the source statement against a table that defines the syntax for FORTRAN IV statements.

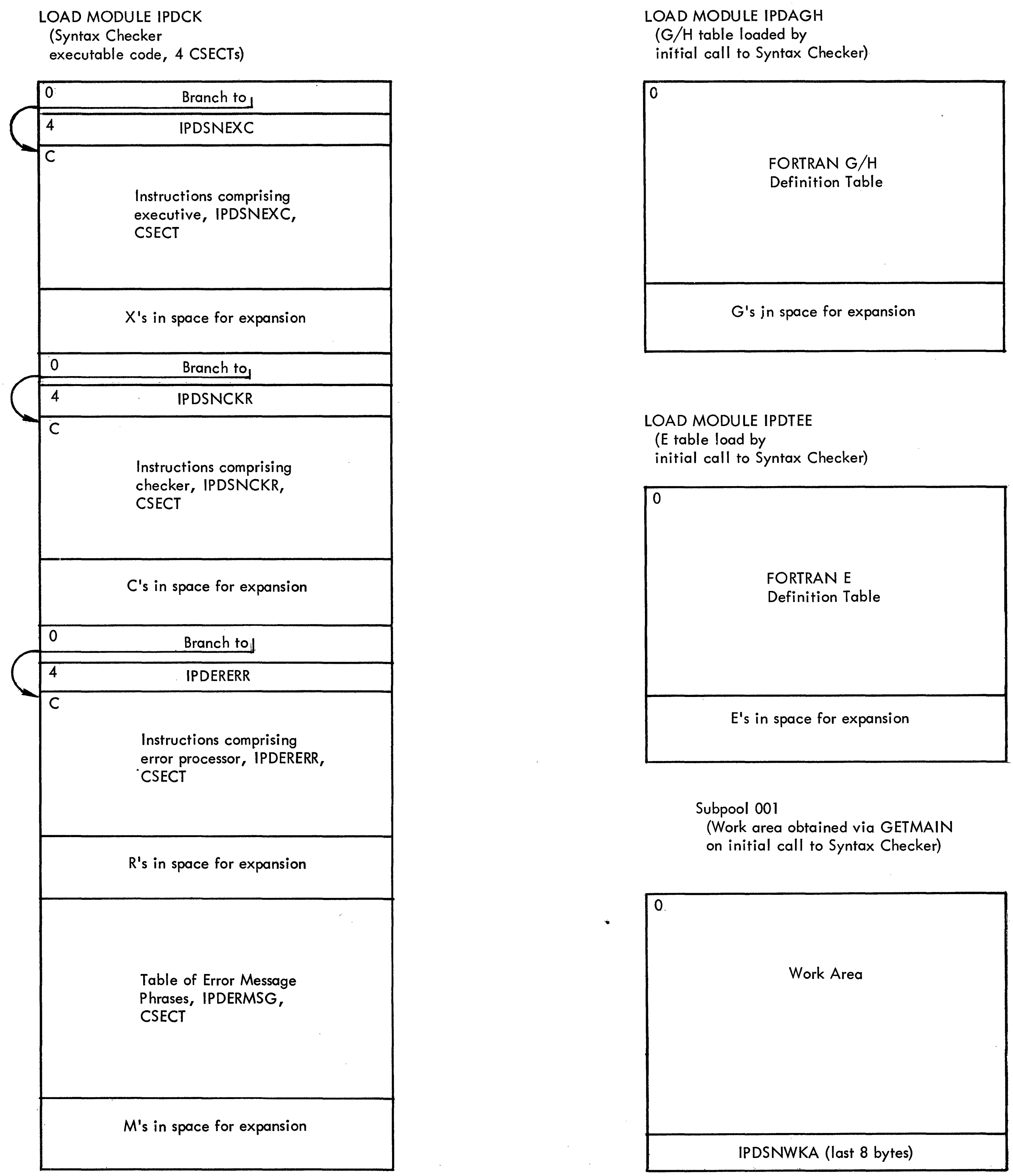


Figure 7. Map of Core Storage

Routines Called

The various get character routines are called to get the next n characters (CKRGTANY), the next nonblank character (CKRGTNB1), or the next n nonblank characters (CKRGTNBS). CKRSKANY is called to skip the next n characters. CKRSERCH is called to scan the remainder of the source for a particular character.

Exits

Upon completion of its processing, the checker returns control to the executive, IPDSNEXC.

Attributes

Re-enterable

GET ANY SOURCE CHARACTERS

Subroutine Name: CKRGTANY (Chart 037)

Entry Point

CKRGTANY is entered when a specified number of characters are to be moved from the source statement to a result buffer.

Functions

CKRGTANY performs the following functions:

- Moves a specified number of source characters to a result buffer.
- Updates a pointer to the source byte following the last one moved, or sets it negative if there is no next source byte.
- Sets on source-end switch and appends an end-of-source character to the last character moved into the result buffer if there were not enough source characters to satisfy the request.

The number of characters requested must fit in the result buffer.

Routines Called

None

Exits

When the number of requested source characters has been found and moved, or the source statement end is encountered, the subroutine CKRGTANY returns to the calling program.

Attributes

Re-enterable

SKIP N SOURCE CHARACTERS

Subroutine Name: CKRSKANY (Chart 037)

Entry Point

CKRSKANY is entered when a specified number of characters are to be skipped in the source statement.

Functions

CKRSKANY performs the following functions:

- Skips source characters until the count is satisfied.
- Updates a pointer to the source byte following the last one skipped, or sets it negative if there is no next source byte.
- Sets on source-end switch if the end of source is found before a specified number of characters are skipped.

Routines Called

None

Exits

The subroutine CKRSKANY returns to the calling program when the specified number of source characters or the end of source has been found.

Attributes

Re-enterable

GET NONBLANK CHARACTERS

Subroutine Name: CKRGTNBS (Chart 037)

Entry Point

CKRGTNBS is entered when a specified number of characters are to be moved from the source statement to a result buffer.

Functions

CKRGTNBS performs the following functions in getting source characters:

- Moves a specified number of nonblank source characters to a result buffer.

- Sets a pointer to the first source character found and moved.

- Sets the source-end switch when the specified number of characters cannot be satisfied and returns a count of those found and moved.

- Sets an update source pointer to the byte beyond the last source character found or sets it negative if there is no next source byte.

- Moves a special character into the result buffer after the last valid source character moved if the request was not completely satisfied.

Routines Called

None

Exits

When the number of requested source characters have been found and moved or the source statement end is encountered, the subroutine CKRGTNBS returns to the calling program.

Attributes

Re-enterable

GET ONE NONBLANK CHARACTER

Subroutine Name: CKRGTNB1 (Chart 036)

Entry Point

CKRGTNB1 is entered to get one nonblank character.

This subroutine has the same specifications as CKRGTNBS except that the number of characters to be found and the number actually moved is assumed to be one.

SEARCH SOURCE FOR SPECIFIC CHARACTER

Subroutine Name: CKRSERCH (Chart 039)

Entry Point

CKRSERCH is entered to search the remainder of a source statement for a user-specified character.

Function

CKRSERCH performs the following functions:

- Searches the statement character string for a specified character.

- Sets the update source pointer to the source character that matches the user-specified character.

- Sets on the source-end switch if the end of the source statement is reached without finding a match, otherwise off.

Routines Called

None

Exits

The subroutine CKRSERCH returns to the calling program when a matching character has been found in the source or the end of source is reached.

Attributes

Re-enterable

ERROR CODE PROCESSOR

Control Section Name: IPDERERR (Chart 040)

Entry Point

IPDERERR is called by the executive (IPDSNEXC) at location IPDERERR.

Function

IPDERERR is given a message code and, using the information in its message definition table, assembles the appropriate message in the message buffer.

Routines Called

None

Exits

When the error message has been assembled in the message buffer, IPDERERR returns control to its caller.

Attributes

Re-enterable

Chart 001. IPDSNEXC (Part 1 of 5)

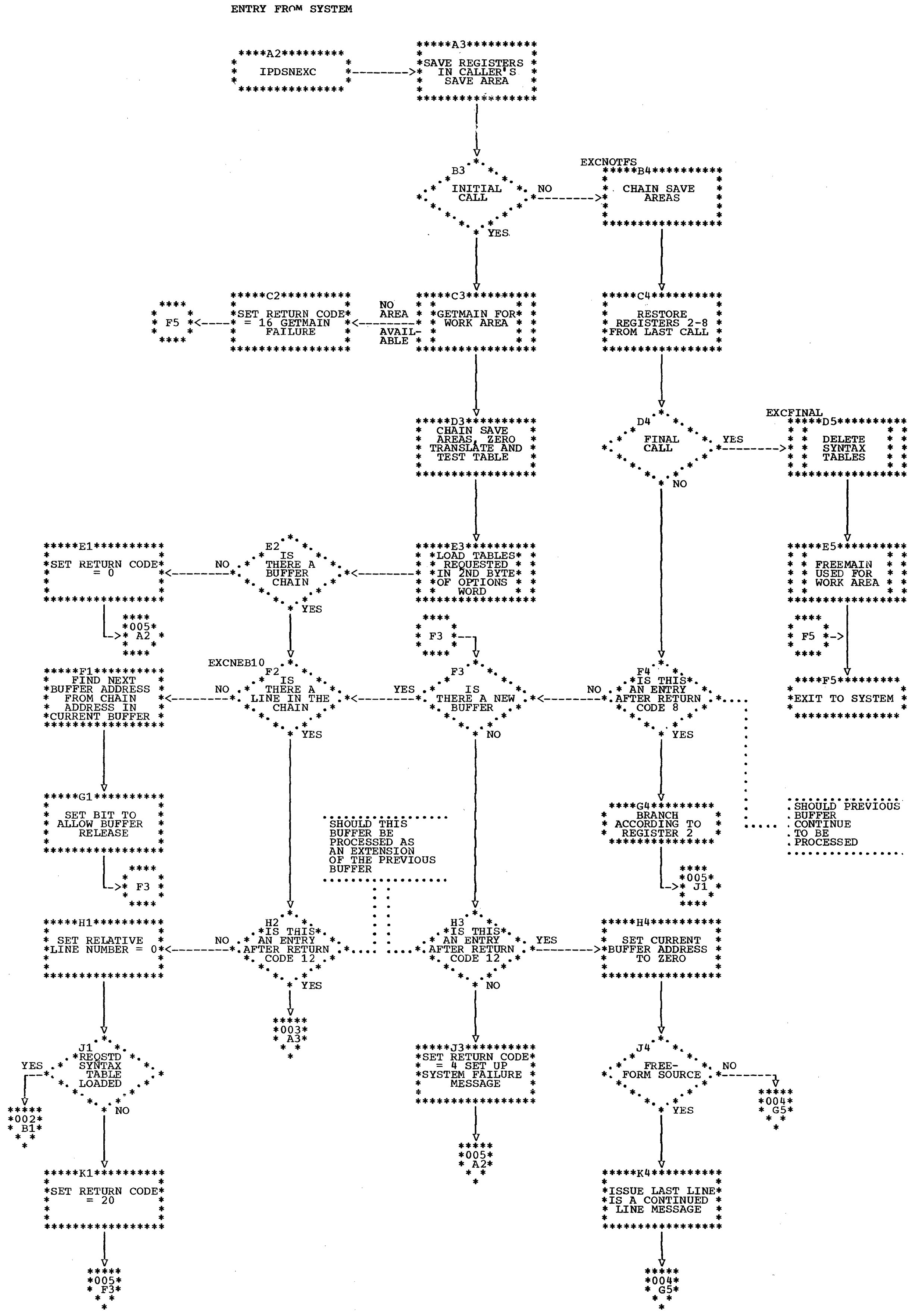

ENTRY FROM SYSTEM
A2 IPDSNEXC
A3 SAVE REGISTERS IN CALLER'S SAVE AREA
B3 INITIAL CALL
NO
YES
EXCNOTFS
B4 CHAIN SAVE AREAS
C2 SET RETURN CODE = 16 GETMAIN FAILURE
F5
NO AREA AVAIL- ABLE
C3 GETMAIN FOR WORK AREA
C4 RESTORE REGISTERS 2-8 FROM LAST CALL
D3 CHAIN SAVE AREAS, ZERO TRANSLATE AND TEST TABLE
D4 FINAL CALL
YES
NO
EXCFINAL
D5 DELETE SYNTAX TABLES
E1 SET RETURN CODE = 0
005 A2
E2 IS THERE A BUFFER CHAIN
NO
YES
E3 LOAD TABLES REQUESTED IN 2ND BYTE OF OPTIONS WORD
E5 FREEMAIN USED FOR WORK AREA
F5
F3
EXCNEB10
F1 FIND NEXT BUFFER ADDRESS FROM CHAIN ADDRESS IN CURRENT BUFFER
F2 IS THERE A LINE IN THE CHAIN
NO
YES
F3 IS THERE A NEW BUFFER
YES
NO
F4 IS THIS AN ENTRY AFTER RETURN CODE 8
NO
YES
F5 EXIT TO SYSTEM
G1 SET BIT TO ALLOW BUFFER RELEASE
F3
SHOULD THIS BUFFER BE PROCESSED AS AN EXTENSION OF THE PREVIOUS BUFFER
G4 BRANCH ACCORDING TO REGISTER 2
005 J1
SHOULD PREVIOUS BUFFER CONTINUE TO BE PROCESSED
H1 SET RELATIVE LINE NUMBER = 0
H2 IS THIS AN ENTRY AFTER RETURN CODE 12
NO
YES
003 A3
H3 IS THIS AN ENTRY AFTER RETURN CODE 12
YES
NO
H4 SET CURRENT BUFFER ADDRESS TO ZERO
J1 REQSTD SYNTAX TABLE LOADED
YES
NO
002 B1
J3 SET RETURN CODE = 4 SET UP SYSTEM FAILURE MESSAGE
005 A2
J4 FREE-FORM SOURCE
NO
YES
004 G5
K1 SET RETURN CODE = 20
005 F3
K4 ISSUE LAST LINE IS A CONTINUED LINE MESSAGE
004 G5

Chart 002. IPDSNEXC (Part 2 of 5)

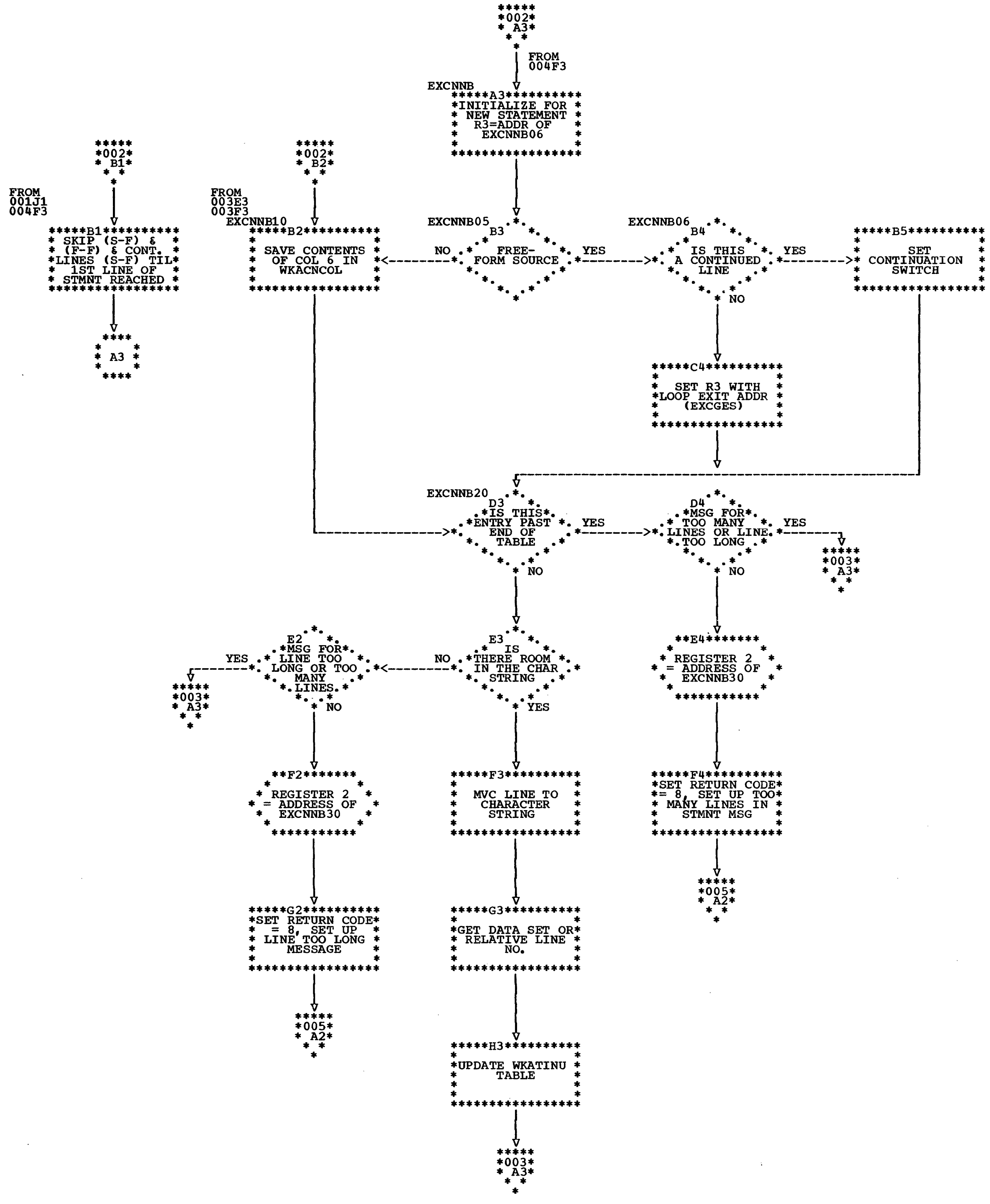
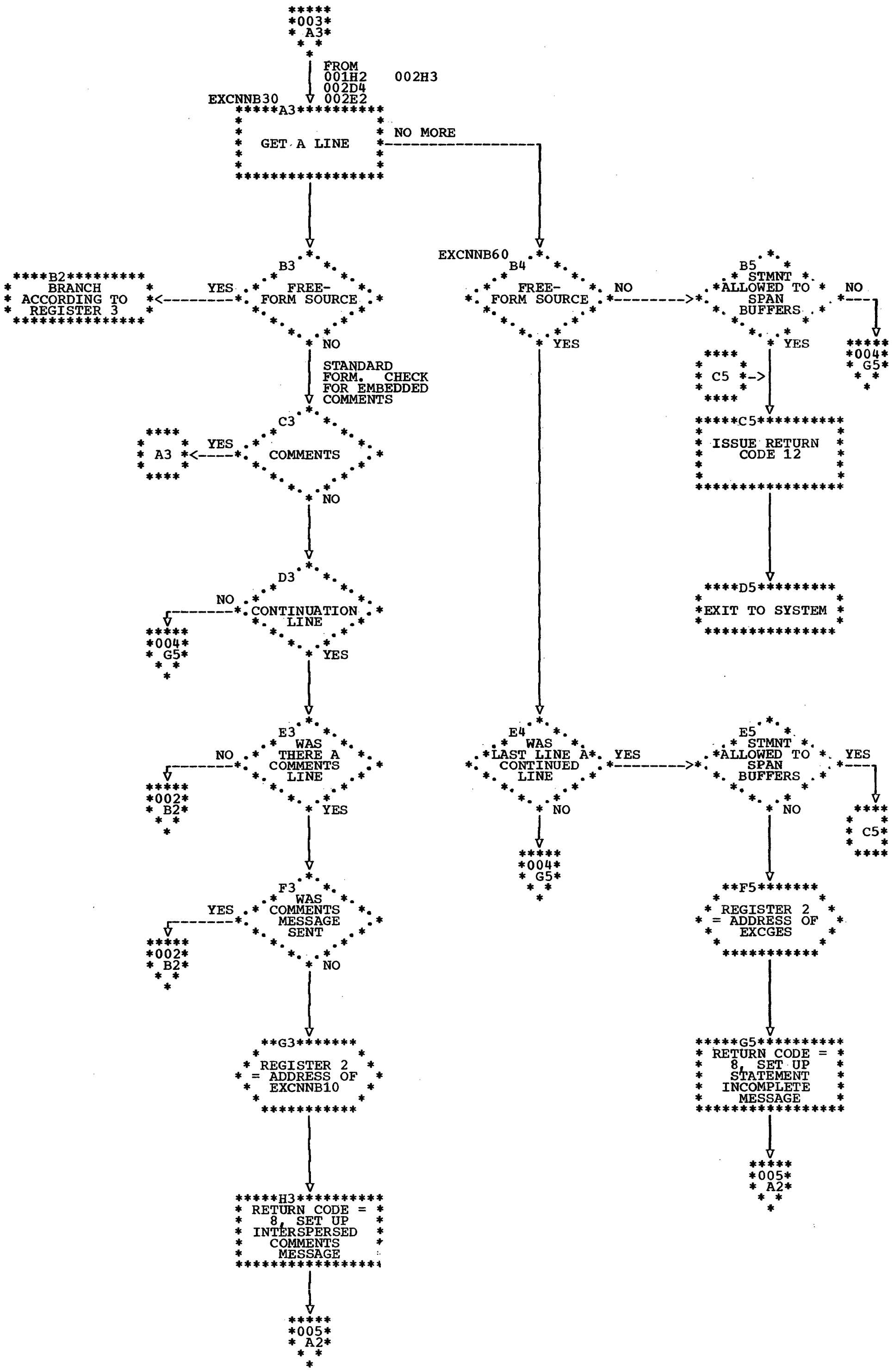
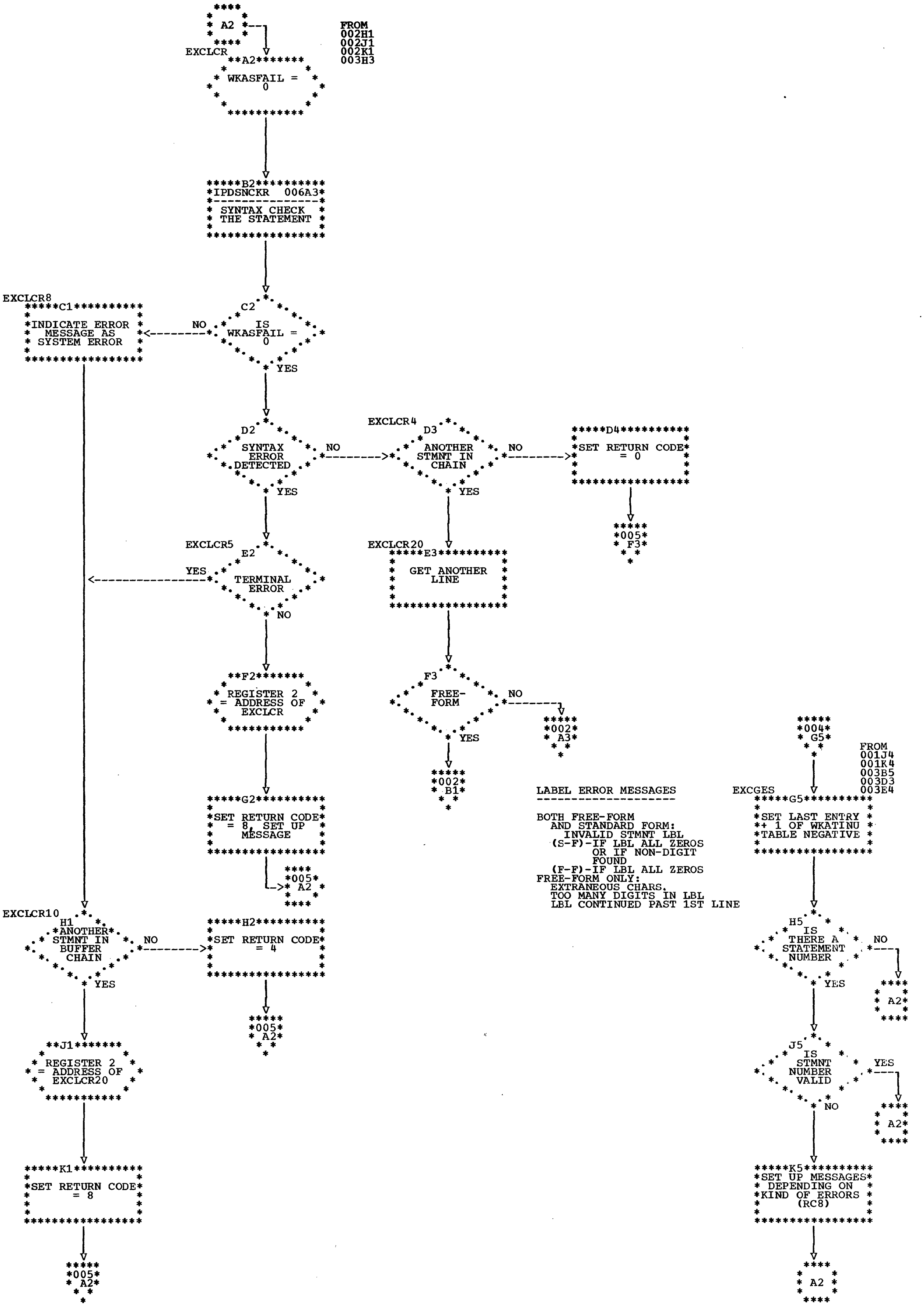

002 A3
FROM 004F3
EXCNNB
A3
INITIALIZE FOR NEW STATEMENT R3=ADDR OF EXCNNB06
002 B1
FROM 001J1 004F3
B1
SKIP (S-F) & (F-F) & CONT. LINES (S-F) TIL 1ST LINE OF STMNT REACHED
A3
002 B2
FROM 003E3 003F3
EXCNNB10
B2
SAVE CONTENTS OF COL 6 IN WKACNCOL
EXCNNB05
B3
FREE-FORM SOURCE
NO
YES
EXCNNB06
B4
IS THIS A CONTINUED LINE
YES
NO
B5
SET CONTINUATION SWITCH
C4
SET R3 WITH LOOP EXIT ADDR (EXCGES)
EXCNNB20
D3
IS THIS ENTRY PAST END OF TABLE
YES
NO
D4
MSG FOR TOO MANY LINES OR LINE TOO LONG
YES
NO
003 A3
E2
MSG FOR LINE TOO LONG OR TOO MANY LINES.
YES
NO
003 A3
E3
IS THERE ROOM IN THE CHAR STRING
NO
YES
E4
REGISTER 2 = ADDRESS OF EXCNNB30
F2
REGISTER 2 = ADDRESS OF EXCNNB30
F3
MVC LINE TO CHARACTER STRING
F4
SET RETURN CODE = 8, SET UP TOO MANY LINES IN STMNT MSG
005 A2
G2
SET RETURN CODE = 8, SET UP LINE TOO LONG MESSAGE
G3
GET DATA SET OR RELATIVE LINE NO.
005 A2
H3
UPDATE WKATINU TABLE
003 A3

Chart 003. IPDSNEXC (Part 3 of 5)


003
A3
FROM
001H2
002D4
002E2
002H3
EXCNNB30
A3
GET A LINE
NO MORE
B2
BRANCH ACCORDING TO REGISTER 3
YES
B3
FREE-FORM SOURCE
NO
STANDARD FORM. CHECK FOR EMBEDDED COMMENTS
A3
YES
C3
COMMENTS
NO
D3
NO
CONTINUATION LINE
004
G5
YES
E3
NO
WAS THERE A COMMENTS LINE
002
B2
YES
F3
YES
WAS COMMENTS MESSAGE SENT
002
B2
NO
G3
REGISTER 2 = ADDRESS OF EXCNNB10
H3
RETURN CODE = 8, SET UP INTERSPERSED COMMENTS MESSAGE
005
A2
EXCNNB60
B4
FREE-FORM SOURCE
NO
YES
E4
WAS LAST LINE A CONTINUED LINE
YES
NO
004
G5
B5
STMNT ALLOWED TO SPAN BUFFERS
NO
004
G5
YES
C5
C5
ISSUE RETURN CODE 12
D5
EXIT TO SYSTEM
E5
STMNT ALLOWED TO SPAN BUFFERS
YES
C5
NO
F5
REGISTER 2 = ADDRESS OF EXCGES
G5
RETURN CODE = 8, SET UP STATEMENT INCOMPLETE MESSAGE
005
A2



```
LABEL ERROR MESSAGES
--------------------

BOTH FREE-FORM
  AND STANDARD FORM:
    INVALID STMNT LBL
   (S-F)-IF LBL ALL ZEROS
          OR IF NON-DIGIT
          FOUND
   (F-F)-IF LBL ALL ZEROS
FREE-FORM ONLY:
  EXTRANEOUS CHARS.
  TOO MANY DIGITS IN LBL
  LBL CONTINUED PAST 1ST LINE
```

Chart 005. IPDSNEXC (Part 5 of 5)

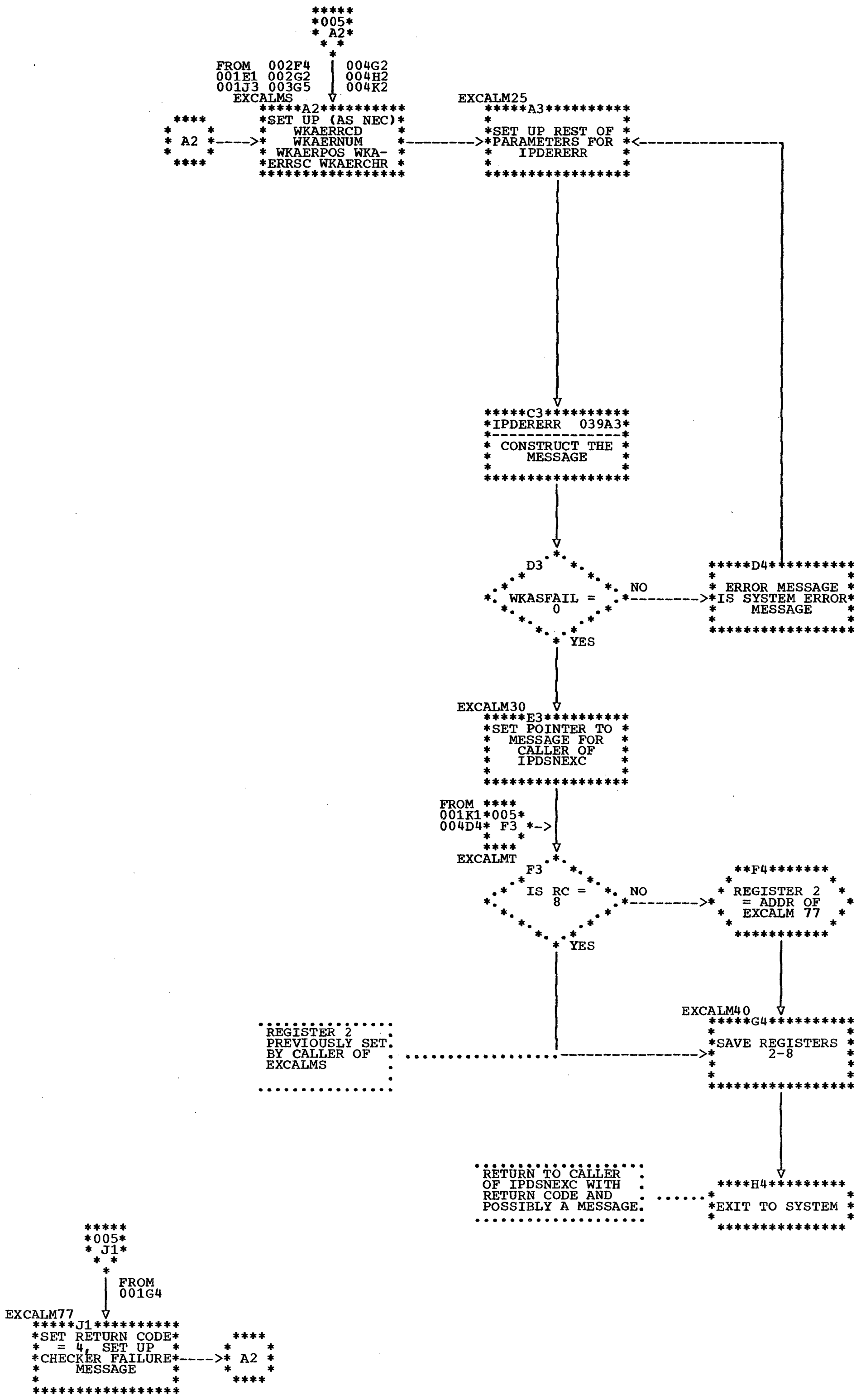
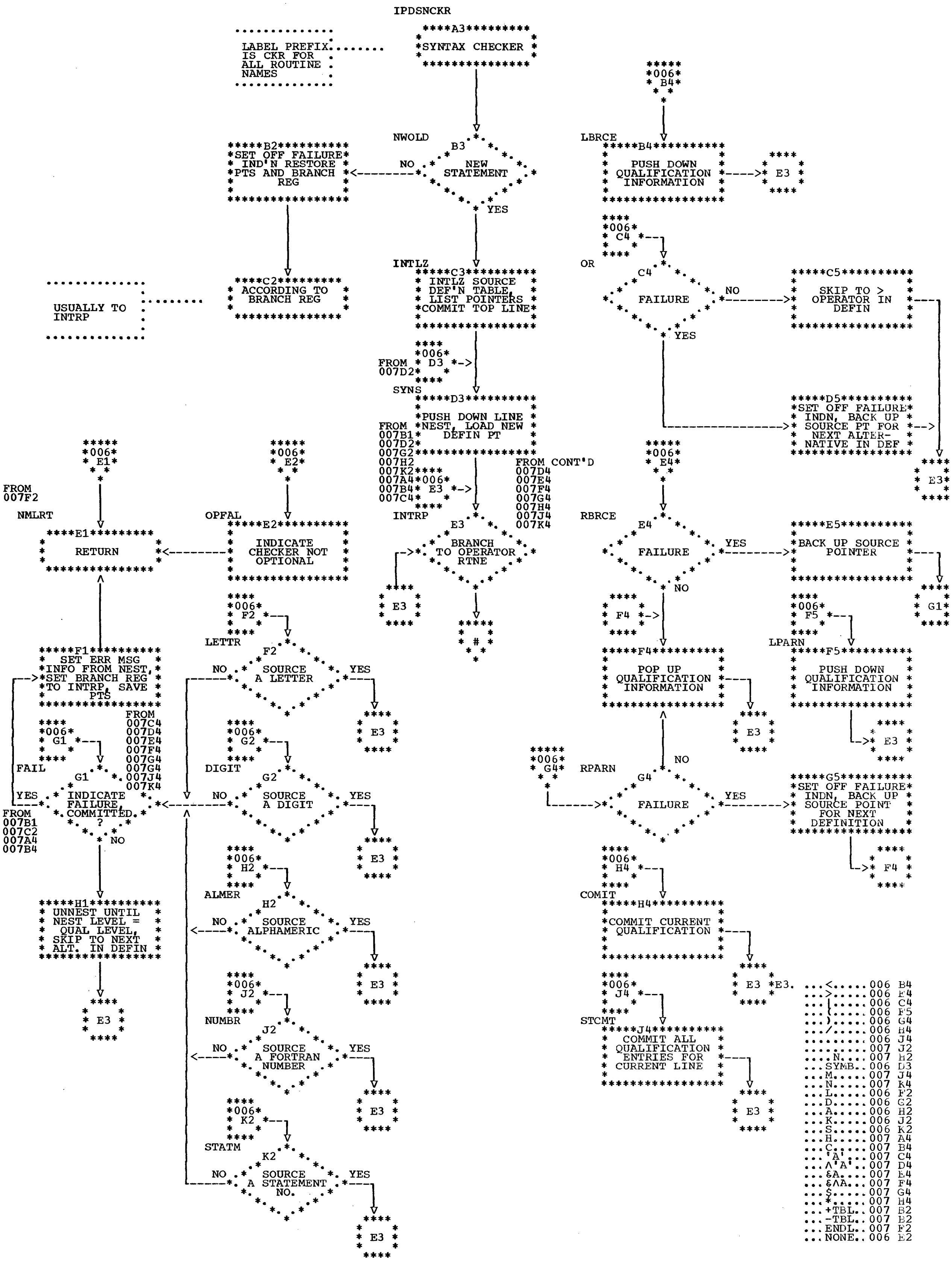

005 A2
FROM 002F4 001E1 002G2 001J3 003G5 004G2 004H2 004K2
EXCALMS
A2
A2 SET UP (AS NEC) WKAERRCD WKAERNUM WKAERPOS WKA-ERRSC WKAERCHR
EXCALM25
A3 SET UP REST OF PARAMETERS FOR IPDERERR
C3 IPDERERR 039A3 CONSTRUCT THE MESSAGE
D3 WKASFAIL = 0
NO
YES
D4 ERROR MESSAGE IS SYSTEM ERROR MESSAGE
EXCALM30
E3 SET POINTER TO MESSAGE FOR CALLER OF IPDSNEXC
FROM 001K1 004D4 005 F3
EXCALMT
F3 IS RC = 8
NO
YES
F4 REGISTER 2 = ADDR OF EXCALM 77
REGISTER 2 PREVIOUSLY SET BY CALLER OF EXCALMS
EXCALM40
G4 SAVE REGISTERS 2-8
RETURN TO CALLER OF IPDSNEXC WITH RETURN CODE AND POSSIBLY A MESSAGE.
H4 EXIT TO SYSTEM
005 J1
FROM 001G4
EXCALM77
J1 SET RETURN CODE = 4, SET UP CHECKER FAILURE MESSAGE
A2

Chart 006. **IPDSNCKR Overview**



IPDSNCKR
A3 SYNTAX CHECKER
LABEL PREFIX IS CKR FOR ALL ROUTINE NAMES
NWOLD
B3 NEW STATEMENT
NO
YES
B2 SET OFF FAILURE IND'N RESTORE PTS AND BRANCH REG
C2 ACCORDING TO BRANCH REG
USUALLY TO INTRP
INTLZ
C3 INTLZ SOURCE DEF'N TABLE, LIST POINTERS COMMIT TOP LINE
006 D3
FROM 007D2
SYNS
D3 PUSH DOWN LINE NEST, LOAD NEW DEFIN PT
FROM 007B1 007D2 007G2 007H2 007K2
006 B4
LBRCE
B4 PUSH DOWN QUALIFICATION INFORMATION
E3
006 C4
OR
C4 FAILURE
NO
YES
C5 SKIP TO > OPERATOR IN DEFIN
D5 SET OFF FAILURE INDN, BACK UP SOURCE PT FOR NEXT ALTER-NATIVE IN DEF
E3
006 E1
FROM 007F2
NMLRT
E1 RETURN
006 E2
OPFAL
E2 INDICATE CHECKER NOT OPTIONAL
006 E3
FROM 007A4 007B4 007C4
INTRP
E3 BRANCH TO OPERATOR RTNE
FROM CONT'D 007D4 007E4 007F4 007G4 007H4 007J4 007K4
E3
#
006 E4
RBRCE
E4 FAILURE
YES
NO
E5 BACK UP SOURCE POINTER
G1
006 F4
F4 POP UP QUALIFICATION INFORMATION
E3
006 F5
LPARN
F5 PUSH DOWN QUALIFICATION INFORMATION
E3
F1 SET ERR MSG INFO FROM NEST, SET BRANCH REG TO INTRP, SAVE PTS
FROM 007C4 007D4 007E4 007F4 007G4 007G4 007J4 007K4
006 G1
FAIL
G1 INDICATE FAILURE, COMMITTED ?
YES
NO
FROM 007B1 007C2 007A4 007B4
006 F2
LETTR
F2 SOURCE A LETTER
NO
YES
E3
006 G2
DIGIT
G2 SOURCE A DIGIT
NO
YES
E3
006 G4
RPARN
G4 FAILURE
NO
YES
G5 SET OFF FAILURE INDN, BACK UP SOURCE POINT FOR NEXT DEFINITION
F4
H1 UNNEST UNTIL NEST LEVEL = QUAL LEVEL, SKIP TO NEXT ALT. IN DEFIN
E3
006 H2
ALMER
H2 SOURCE ALPHAMERIC
NO
YES
E3
006 H4
COMIT
H4 COMMIT CURRENT QUALIFICATION
E3
006 J2
NUMBR
J2 SOURCE A FORTRAN NUMBER
NO
YES
E3
006 J4
STCMT
J4 COMMIT ALL QUALIFICATION ENTRIES FOR CURRENT LINE
E3
006 K2
STATM
K2 SOURCE A STATEMENT NO.
NO
YES
E3
E3. ...<......006 B4
...>......006 F4
...|......006 C4
...(......006 F5
...)......006 G4
.../......006 H4
..........006 J4
..........007 J2
....N.....007 H2
...SYMB..006 D3
...M......007 J4
...N......007 K4
...L......006 F2
...D......006 G2
...A......006 H2
...K......006 J2
...S......006 K2
...H......007 A4
...C......007 B4
...'A'....007 C4
...^'A'..007 D4
...&A.....007 E4
...&^A...007 F4
...$......007 G4
...*......007 H4
...+TBL..007 B2
...-TBL..007 B2
...ENDL..007 F2
...NONE..006 E2

Chart 007. IPDSNCKR Overview

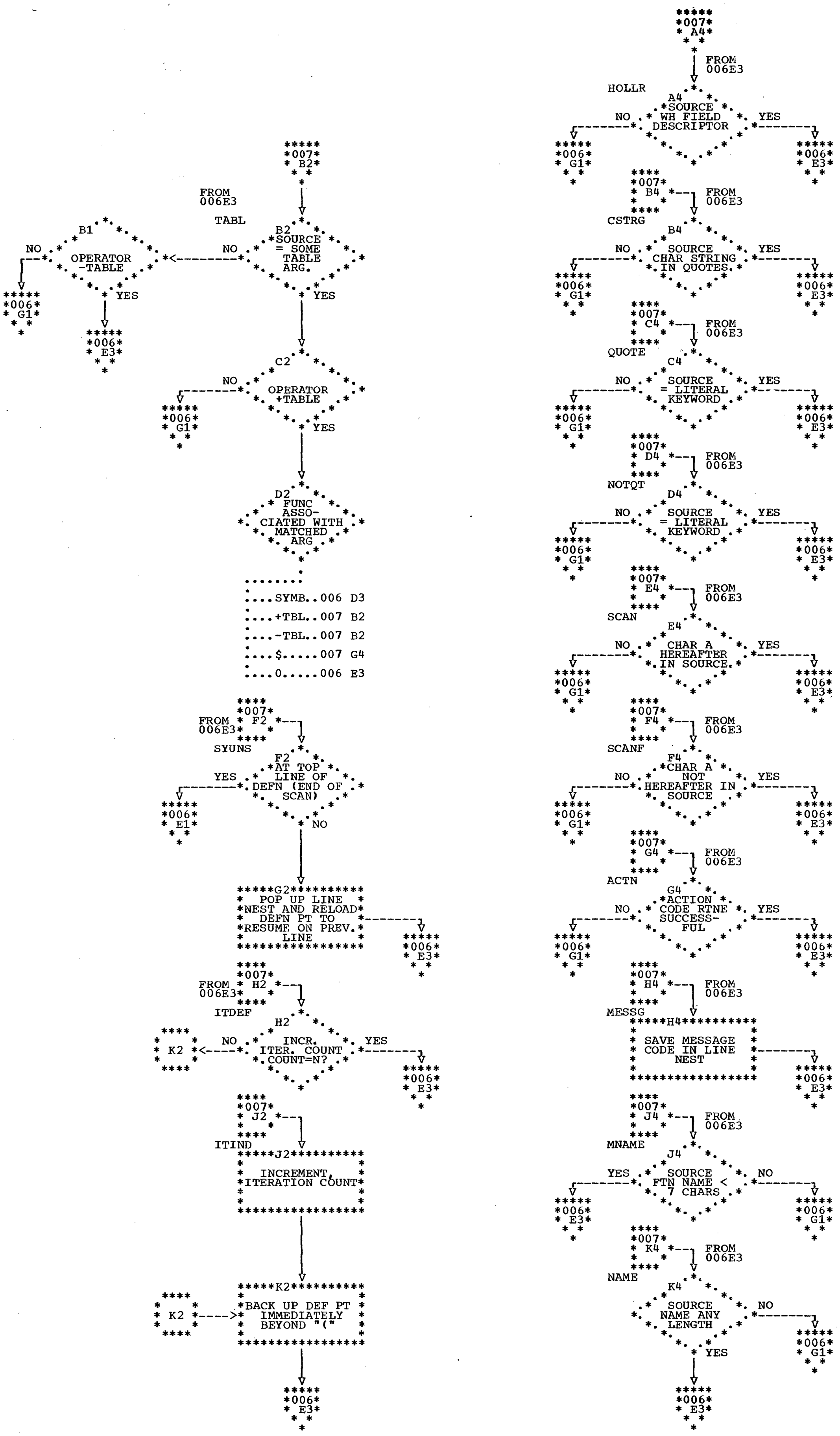


Chart 008. IPDSNCKR (CKRNWOLD, CKRINTLZ)

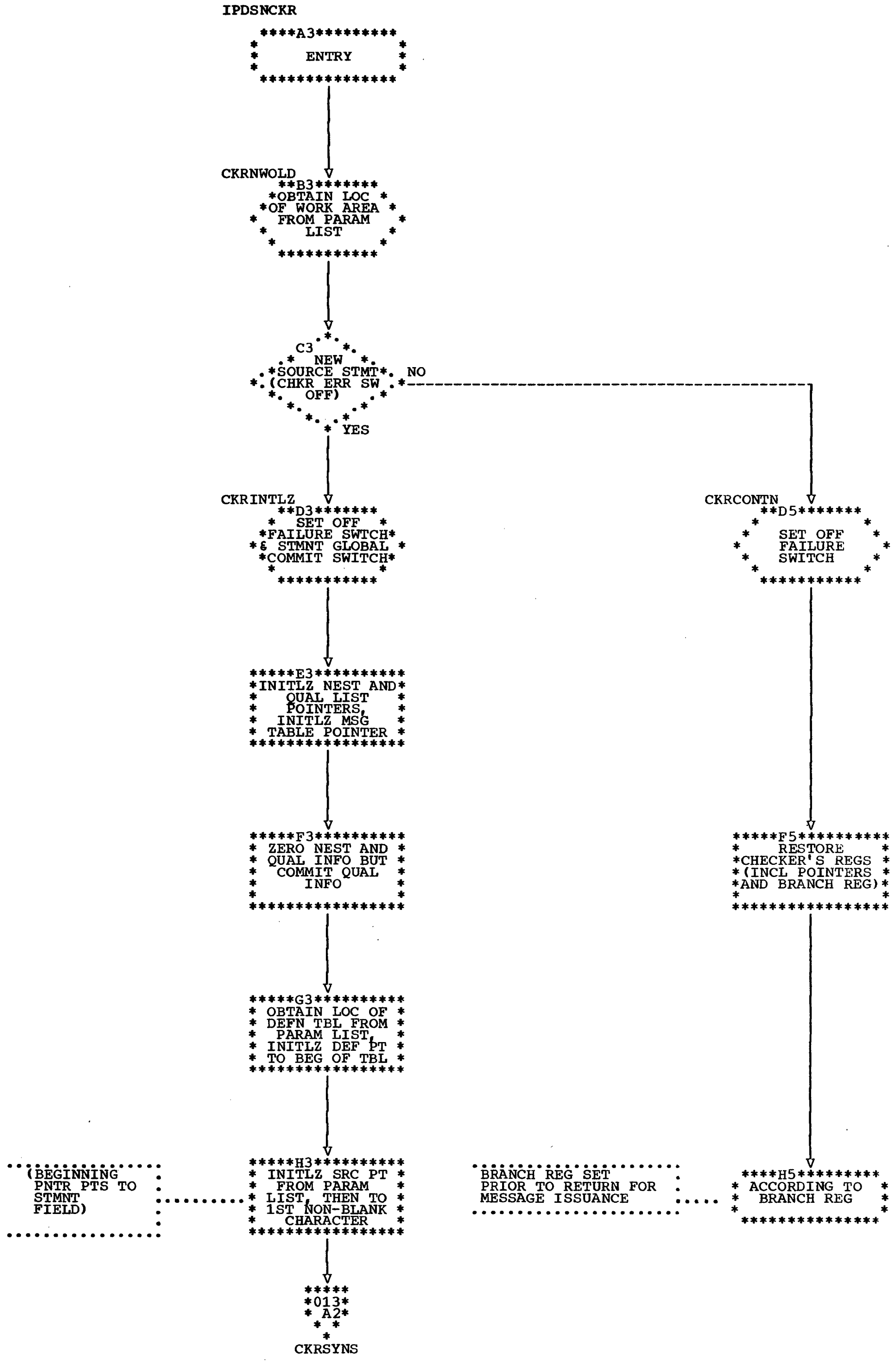
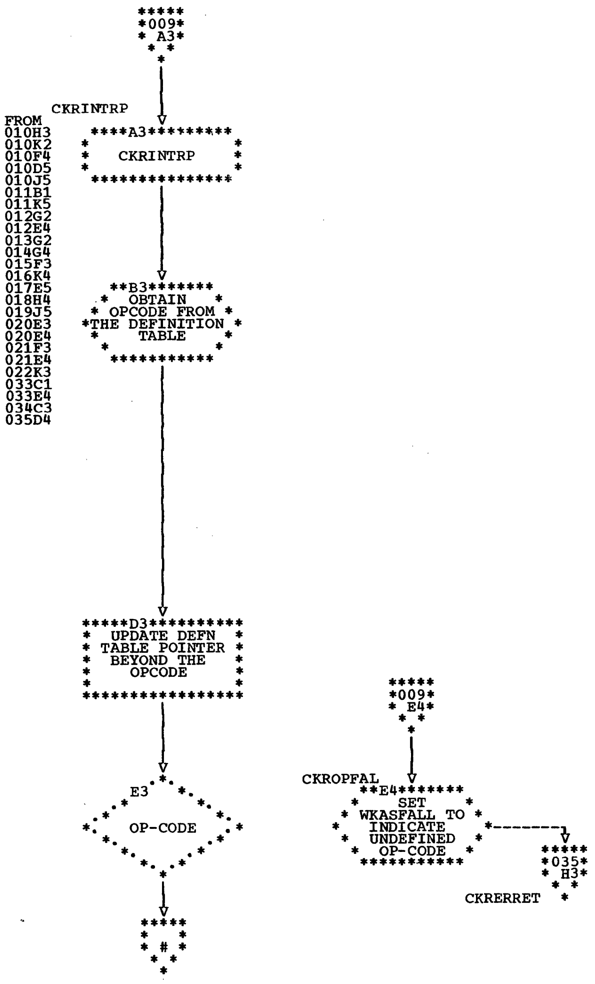

IPDSNCKR
A3
ENTRY
CKRNWOLD
B3
OBTAIN LOC OF WORK AREA FROM PARAM LIST
C3
NEW SOURCE STMT (CHKR ERR SW OFF)
NO
YES
CKRINTLZ
D3
SET OFF FAILURE SWTCH & STMNT GLOBAL COMMIT SWITCH
CKRCONTN
D5
SET OFF FAILURE SWITCH
E3
INITLZ NEST AND QUAL LIST POINTERS, INITLZ MSG TABLE POINTER
F3
ZERO NEST AND QUAL INFO BUT COMMIT QUAL INFO
F5
RESTORE CHECKER'S REGS (INCL POINTERS AND BRANCH REG)
G3
OBTAIN LOC OF DEFN TBL FROM PARAM LIST, INITLZ DEF PT TO BEG OF TBL
(BEGINNING PNTR PTS TO STMNT FIELD)
H3
INITLZ SRC PT FROM PARAM LIST, THEN TO 1ST NON-BLANK CHARACTER
BRANCH REG SET PRIOR TO RETURN FOR MESSAGE ISSUANCE
H5
ACCORDING TO BRANCH REG
013
A2
CKRSYNS

Chart 009. IPDSNCKR (CKRINTRP)



```
E3.  ...00....010 A1
     ...02....010 A5
     ...04....010 F3
     ...06....010 A2
     ...08....010 A4
     ...10....011 A1
     ...12....011 A3
     ...14....012 A2
     ...16....012 A4
     ...18....013 A2
     ...20....014 A2
     ...22....014 A4
     ...24....015 A2
     ...26....015 A1
     ...28....015 A3
     ...30....016 A2
     ...32....017 A1
     ...34....018 A3
     ...36....019 A3
     ...38....020 A2
     ...40....020 A4
     ...42....021 A3
     ...44....021 A4
     ...46....022 A3
     ...48....033 A1
     ...50....033 A3
     ...52....033 A3
     ...54....034 A3
     ...NONE..009 E4
```

Chart 010. IPDSNCKR (CKRLBRCE, CKRLPARN, CKROR, CKRRPARN, CKRRBRCE)

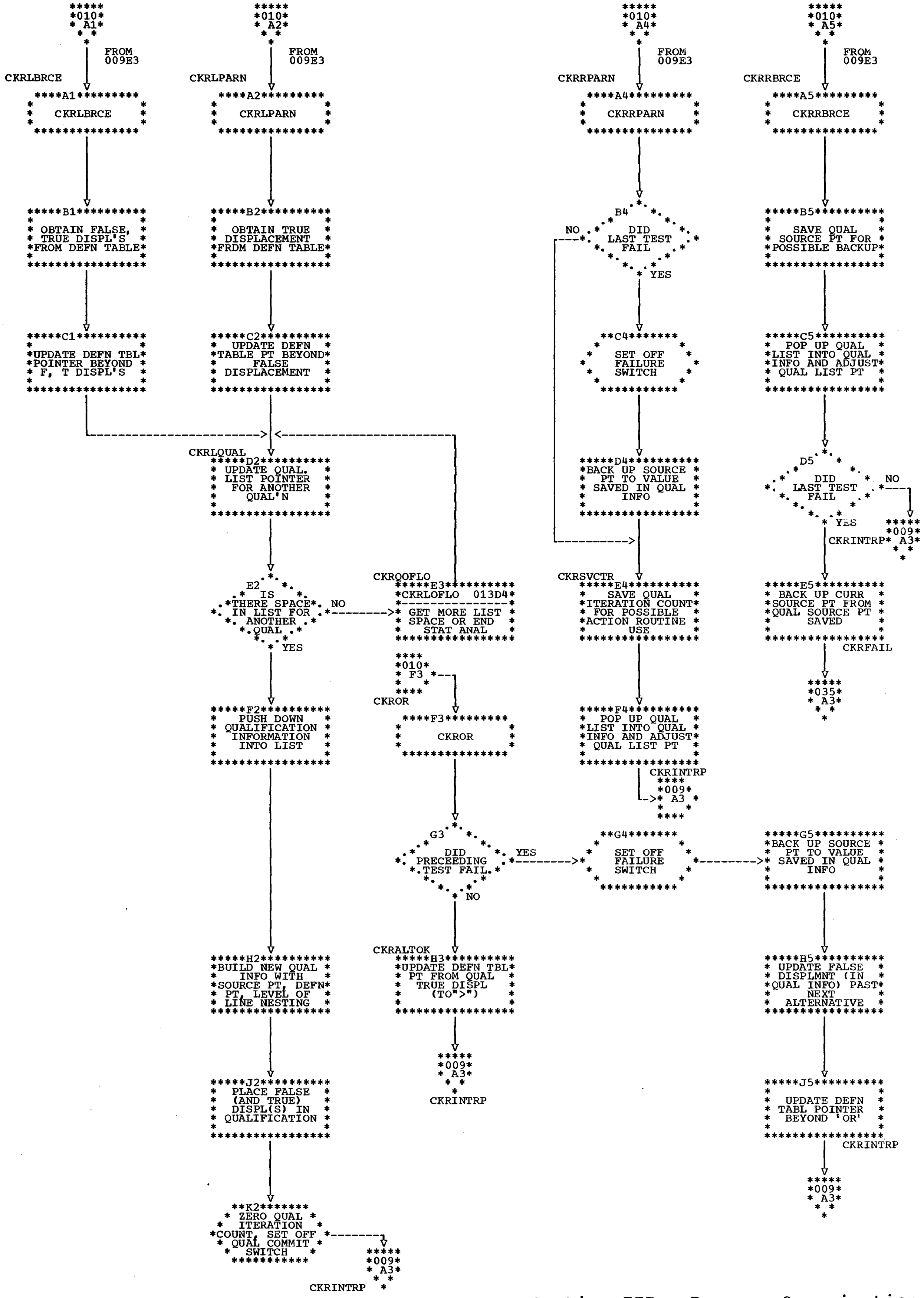

010 A1
FROM 009E3
CKRLBRCE
A1 CKRLBRCE
B1 OBTAIN FALSE, TRUE DISPL'S FROM DEFN TABLE
C1 UPDATE DEFN TBL POINTER BEYOND F, T DISPL'S
010 A2
FROM 009E3
CKRLPARN
A2 CKRLPARN
B2 OBTAIN TRUE DISPLACEMENT FRDM DEFN TABLE
C2 UPDATE DEFN TABLE PT BEYOND FALSE DISPLACEMENT
CKRLQUAL
D2 UPDATE QUAL. LIST POINTER FOR ANOTHER QUAL'N
E2 IS THERE SPACE IN LIST FOR ANOTHER QUAL
NO
YES
CKRQOFLO
E3 CKRLOFLO 013D4 GET MORE LIST SPACE OR END STAT ANAL
F2 PUSH DOWN QUALIFICATION INFORMATION INTO LIST
H2 BUILD NEW QUAL INFO WITH SOURCE PT, DEFN PT, LEVEL OF LINE NESTING
J2 PLACE FALSE (AND TRUE) DISPL(S) IN QUALIFICATION
K2 ZERO QUAL ITERATION COUNT, SET OFF QUAL COMMIT SWITCH
009 A3
CKRINTRP
010 F3
CKROR
F3 CKROR
G3 DID PRECEEDING TEST FAIL.
YES
NO
CKRALTOK
H3 UPDATE DEFN TBL PT FROM QUAL TRUE DISPL (TO"&gt;")
009 A3
CKRINTRP
010 A4
FROM 009E3
CKRRPARN
A4 CKRRPARN
B4 DID LAST TEST FAIL
NO
YES
C4 SET OFF FAILURE SWITCH
D4 BACK UP SOURCE PT TO VALUE SAVED IN QUAL INFO
CKRSVCTR
E4 SAVE QUAL ITERATION COUNT FOR POSSIBLE ACTION ROUTINE USE
F4 POP UP QUAL LIST INTO QUAL INFO AND ADJUST QUAL LIST PT
CKRINTRP
009 A3
G4 SET OFF FAILURE SWITCH
010 A5
FROM 009E3
CKRRBRCE
A5 CKRRBRCE
B5 SAVE QUAL SOURCE PT FOR POSSIBLE BACKUP
C5 POP UP QUAL LIST INTO QUAL INFO AND ADJUST QUAL LIST PT
D5 DID LAST TEST FAIL
NO
YES
009 A3
CKRINTRP
E5 BACK UP CURR SOURCE PT FROM QUAL SOURCE PT SAVED
CKRFAIL
035 A3
G5 BACK UP SOURCE PT TO VALUE SAVED IN QUAL INFO
H5 UPDATE FALSE DISPLMNT (IN QUAL INFO) PAST NEXT ALTERNATIVE
J5 UPDATE DEFN TABL POINTER BEYOND 'OR'
CKRINTRP
009 A3

Chart 011. IPDSNCKR (CKRCOMIT, CKRSTCMT)

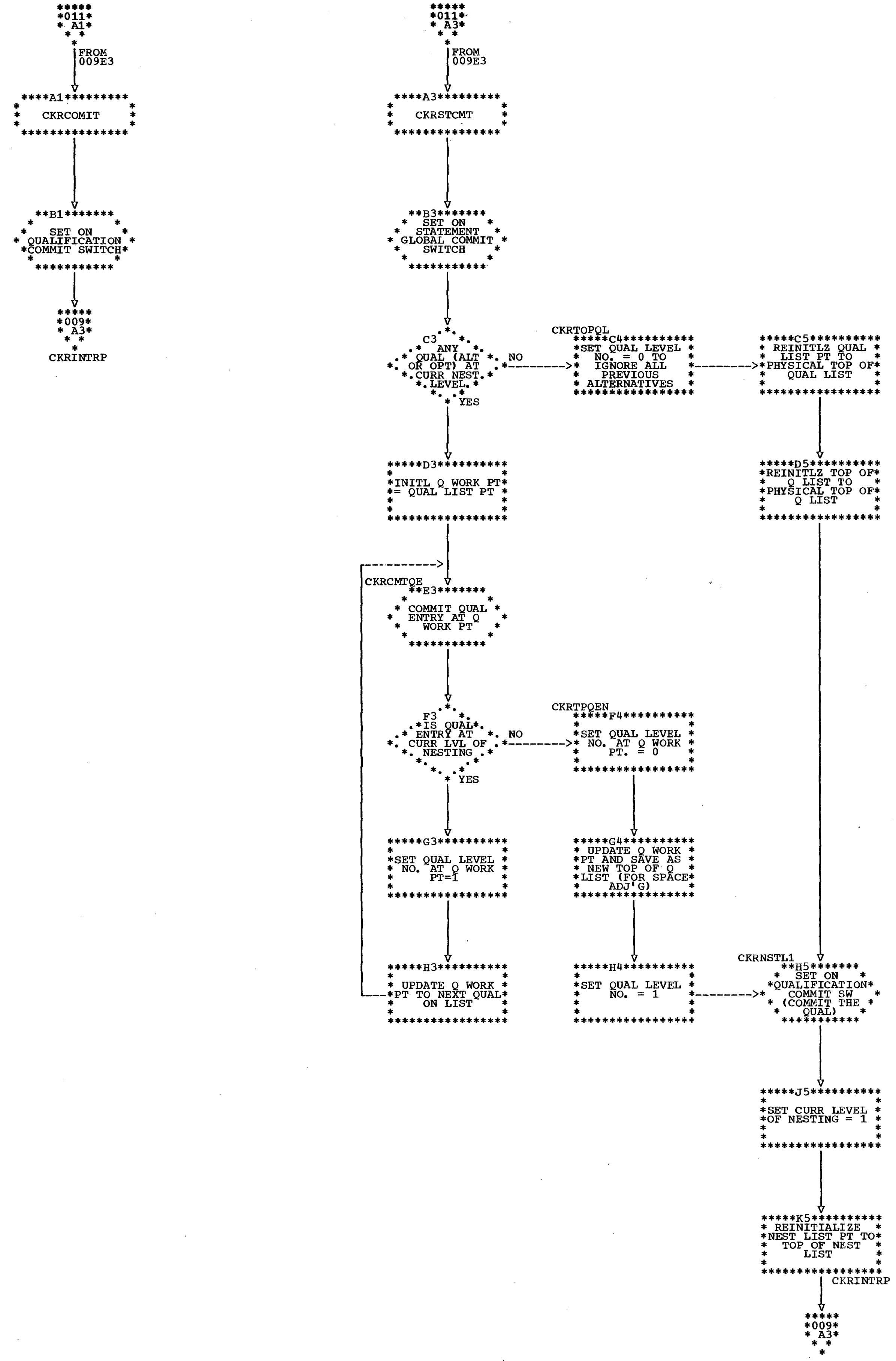

011 A1
FROM 009E3
A1 CKRCOMIT
B1 SET ON QUALIFICATION COMMIT SWITCH
009 A3
CKRINTRP
011 A3
FROM 009E3
A3 CKRSTCMT
B3 SET ON STATEMENT GLOBAL COMMIT SWITCH
C3 ANY QUAL (ALT OR OPT) AT CURR NEST. LEVEL.
NO
YES
CKRTOPQL
C4 SET QUAL LEVEL NO. = 0 TO IGNORE ALL PREVIOUS ALTERNATIVES
C5 REINITLZ QUAL LIST PT TO PHYSICAL TOP OF QUAL LIST
D3 INITL Q WORK PT = QUAL LIST PT
D5 REINITLZ TOP OF Q LIST TO PHYSICAL TOP OF Q LIST
CKRCMTQE
E3 COMMIT QUAL ENTRY AT Q WORK PT
F3 IS QUAL ENTRY AT CURR LVL OF NESTING
NO
YES
CKRTPQEN
F4 SET QUAL LEVEL NO. AT Q WORK PT. = 0
G3 SET QUAL LEVEL NO. AT Q WORK PT=1
G4 UPDATE Q WORK PT AND SAVE AS NEW TOP OF Q LIST (FOR SPACE ADJ'G)
H3 UPDATE Q WORK PT TO NEXT QUAL ON LIST
H4 SET QUAL LEVEL NO. = 1
CKRNSTL1
H5 SET ON QUALIFICATION COMMIT SW (COMMIT THE QUAL)
J5 SET CURR LEVEL OF NESTING = 1
K5 REINITIALIZE NEST LIST PT TO TOP OF NEST LIST
CKRINTRP
009 A3

Chart 012. IPDSNCKR (CKRITIND, CKRITDEF)

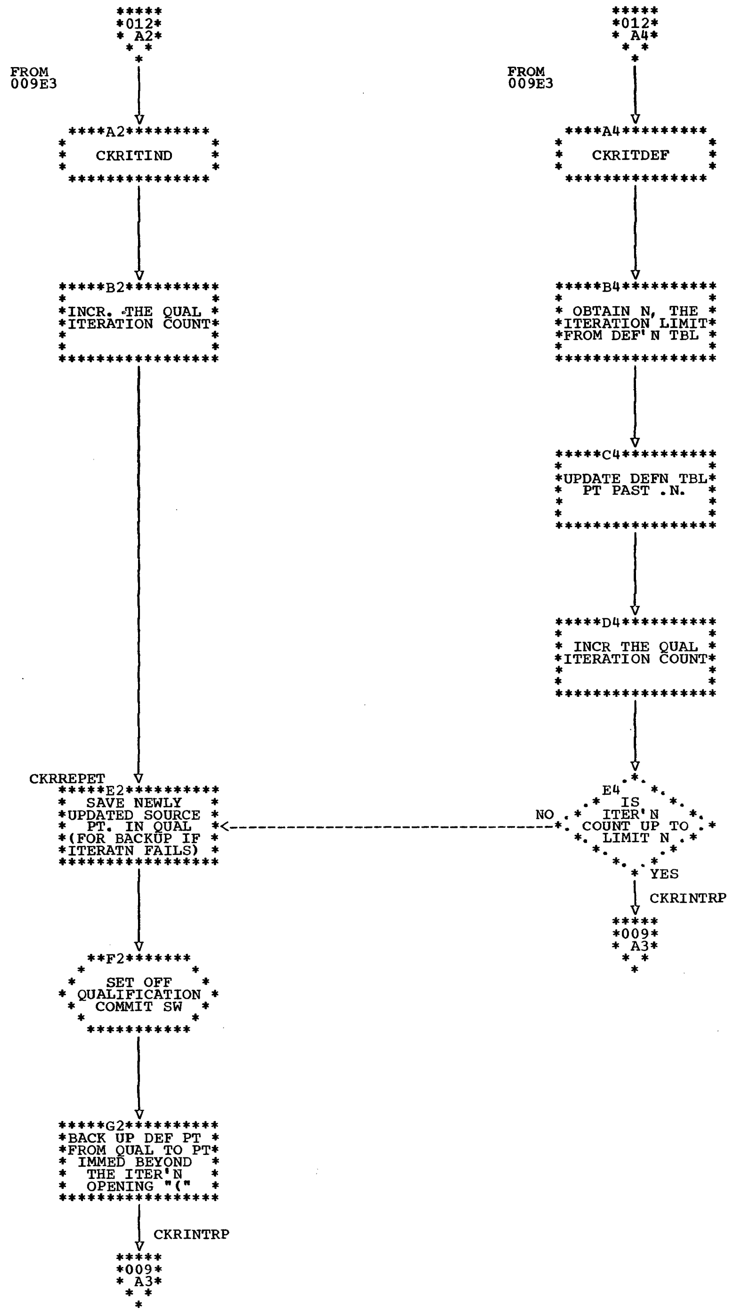

012
A2
FROM 009E3
A2
CKRITIND
B2
INCR. THE QUAL ITERATION COUNT
CKRREPET
E2
SAVE NEWLY UPDATED SOURCE PT. IN QUAL (FOR BACKUP IF ITERATN FAILS)
F2
SET OFF QUALIFICATION COMMIT SW
G2
BACK UP DEF PT FROM QUAL TO PT IMMED BEYOND THE ITER'N OPENING "("
CKRINTRP
009
A3
012
A4
FROM 009E3
A4
CKRITDEF
B4
OBTAIN N, THE ITERATION LIMIT FROM DEF'N TBL
C4
UPDATE DEFN TBL PT PAST .N.
D4
INCR THE QUAL ITERATION COUNT
E4
IS ITER'N COUNT UP TO LIMIT N
NO
YES
CKRINTRP
009
A3

Chart 013. IPDSNCKR (CKRSYNS, CKRLOFLO)

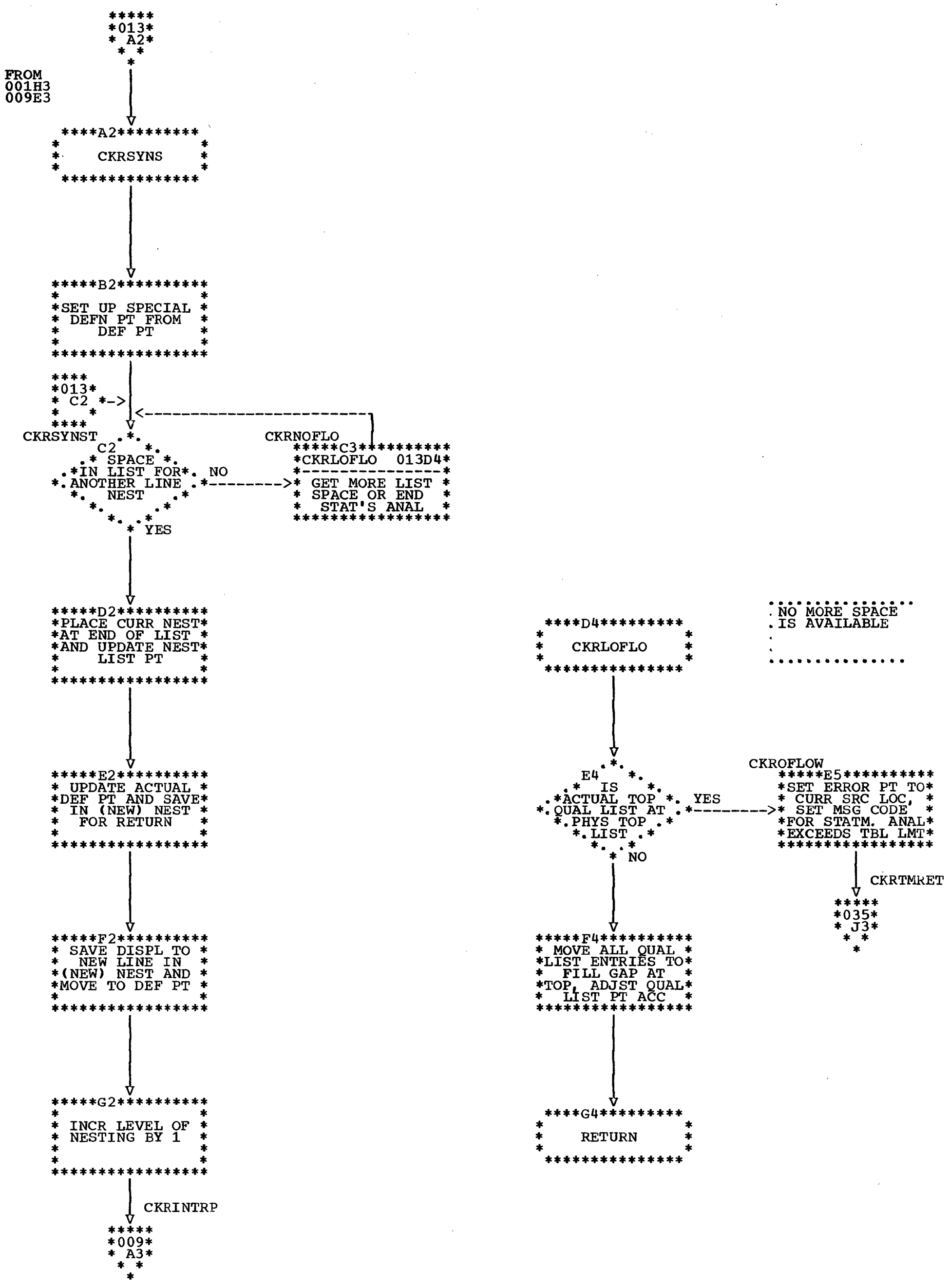

013 A2
FROM 001H3 009E3
A2 CKRSYNS
B2 SET UP SPECIAL DEFN PT FROM DEF PT
013 C2
CKRSYNST
C2 SPACE IN LIST FOR ANOTHER LINE NEST
NO
YES
CKRNOFLO
C3 CKRLOFLO 013D4
GET MORE LIST SPACE OR END STAT'S ANAL
D2 PLACE CURR NEST AT END OF LIST AND UPDATE NEST LIST PT
E2 UPDATE ACTUAL DEF PT AND SAVE IN (NEW) NEST FOR RETURN
F2 SAVE DISPL TO NEW LINE IN (NEW) NEST AND MOVE TO DEF PT
G2 INCR LEVEL OF NESTING BY 1
CKRINTRP
009 A3
D4 CKRLOFLO
NO MORE SPACE IS AVAILABLE
E4 IS ACTUAL TOP QUAL LIST AT PHYS TOP LIST
YES
NO
CKROFLOW
E5 SET ERROR PT TO CURR SRC LOC, SET MSG CODE FOR STATM. ANAL EXCEEDS TBL LMT
CKRTMRET
035 J3
F4 MOVE ALL QUAL LIST ENTRIES TO FILL GAP AT TOP, ADJST QUAL LIST PT ACC
G4 RETURN

Chart 014. IPDSNCKR (CKRMNAME, CKRNAME)

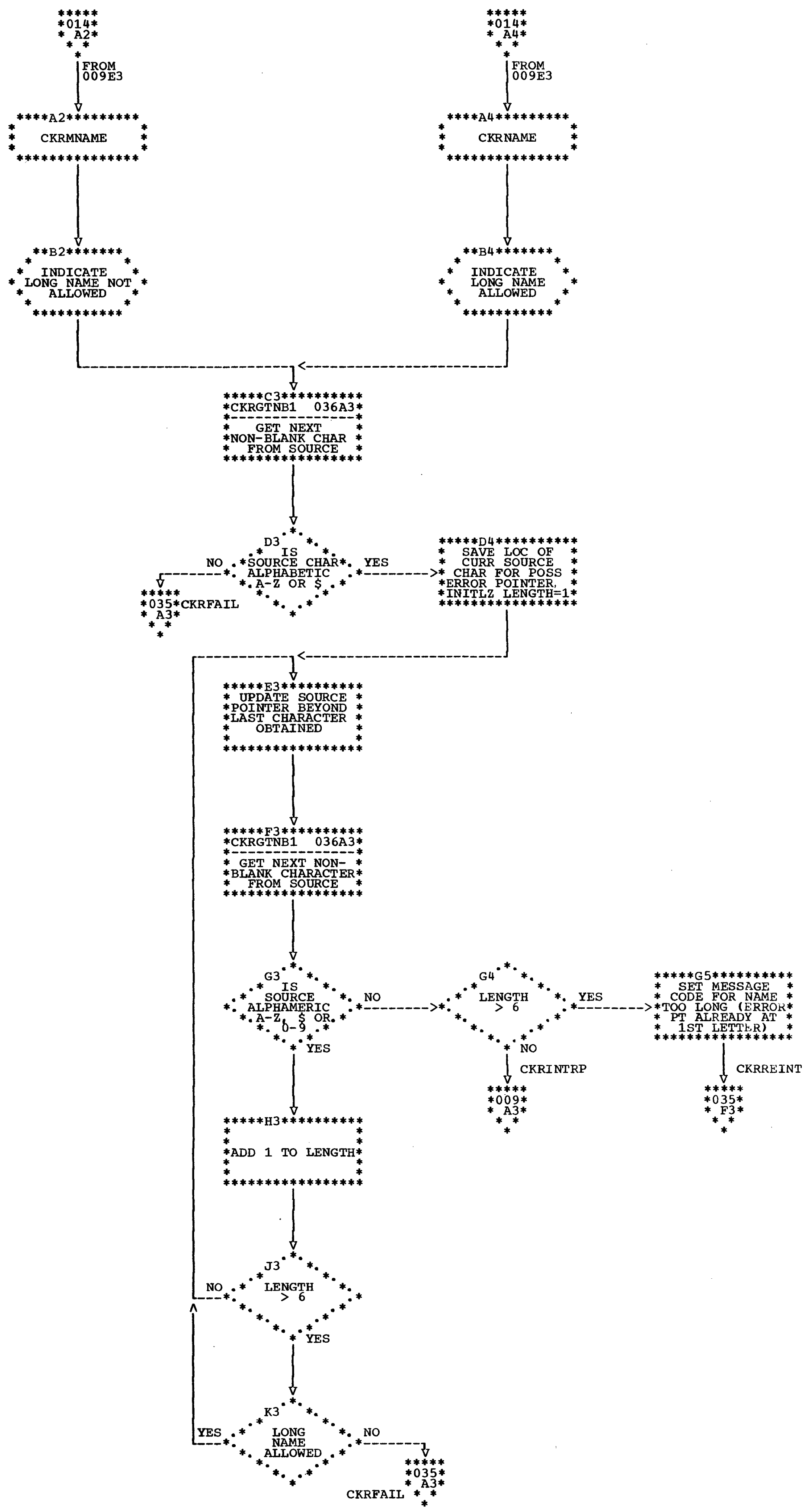

014 A2
FROM 009E3
A2 CKRMNAME
B2 INDICATE LONG NAME NOT ALLOWED
014 A4
FROM 009E3
A4 CKRNAME
B4 INDICATE LONG NAME ALLOWED
C3 CKRGTNB1 036A3
GET NEXT NON-BLANK CHAR FROM SOURCE
D3 IS SOURCE CHAR ALPHABETIC A-Z OR $
NO
035 A3 CKRFAIL
YES
D4 SAVE LOC OF CURR SOURCE CHAR FOR POSS ERROR POINTER, INITLZ LENGTH=1
E3 UPDATE SOURCE POINTER BEYOND LAST CHARACTER OBTAINED
F3 CKRGTNB1 036A3
GET NEXT NON-BLANK CHARACTER FROM SOURCE
G3 IS SOURCE ALPHAMERIC A-Z, $ OR 0-9
NO
G4 LENGTH > 6
YES
G5 SET MESSAGE CODE FOR NAME TOO LONG (ERROR PT ALREADY AT 1ST LETTER)
NO
CKRINTRP
009 A3
CKRREINT
035 F3
YES
H3 ADD 1 TO LENGTH
J3 LENGTH > 6
NO
YES
K3 LONG NAME ALLOWED
YES
NO
035 A3
CKRFAIL

Chart 015. IPDSNCKR (CKRDIGIT, CKRLETTR, CKRALMER)

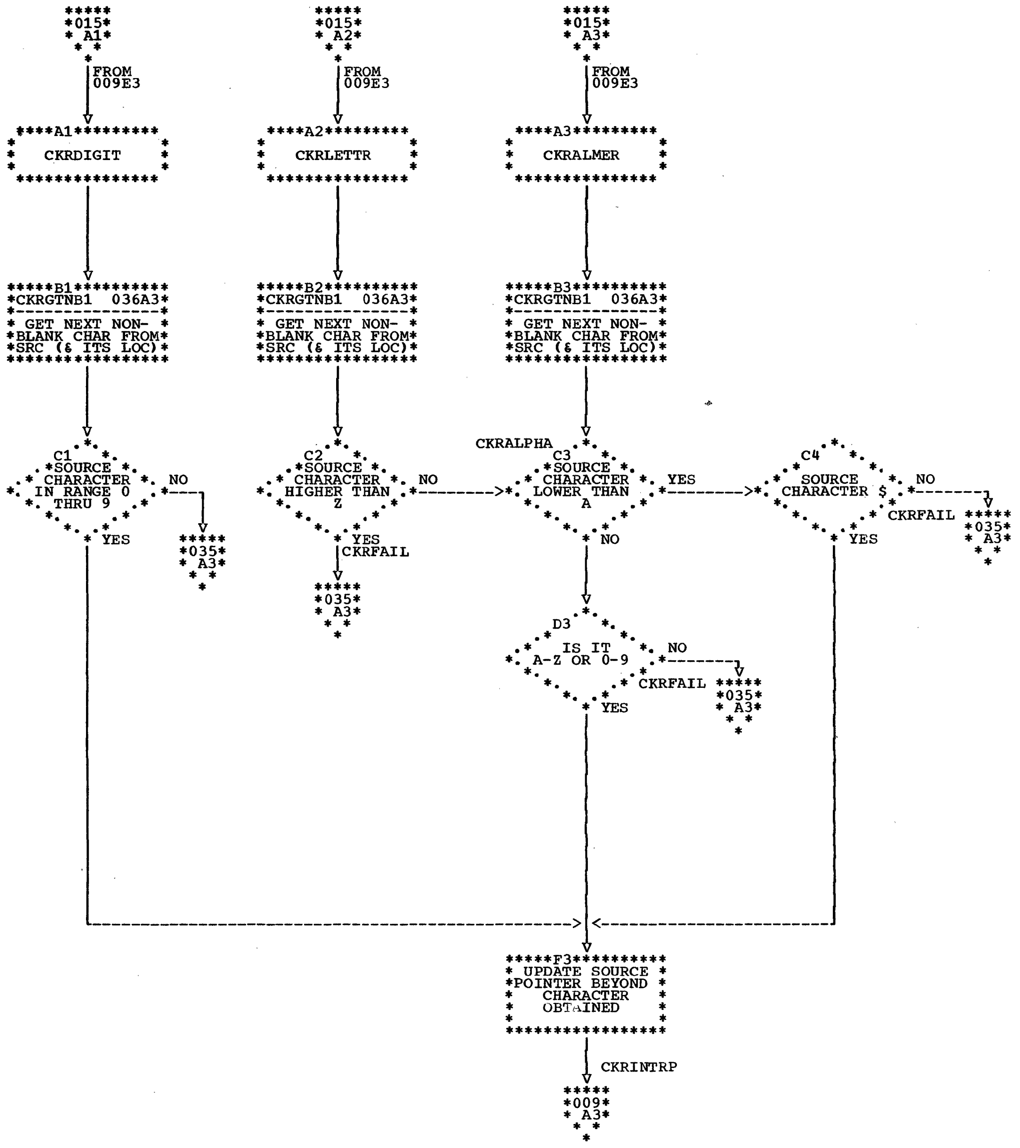

015 A1
FROM 009E3
A1
CKRDIGIT
B1
CKRGTNB1 036A3
GET NEXT NON-BLANK CHAR FROM SRC (& ITS LOC)
C1
SOURCE CHARACTER IN RANGE 0 THRU 9
NO
035 A3
YES
015 A2
FROM 009E3
A2
CKRLETTR
B2
CKRGTNB1 036A3
GET NEXT NON-BLANK CHAR FROM SRC (& ITS LOC)
C2
SOURCE CHARACTER HIGHER THAN Z
NO
YES
CKRFAIL
035 A3
015 A3
FROM 009E3
A3
CKRALMER
B3
CKRGTNB1 036A3
GET NEXT NON-BLANK CHAR FROM SRC (& ITS LOC)
CKRALPHA
C3
SOURCE CHARACTER LOWER THAN A
YES
NO
C4
SOURCE CHARACTER $
NO
CKRFAIL
035 A3
YES
D3
IS IT A-Z OR 0-9
NO
CKRFAIL
035 A3
YES
F3
UPDATE SOURCE POINTER BEYOND CHARACTER OBTAINED
CKRINTRP
009 A3

Chart 016. IPDSNCKR (CKRNUMBR)

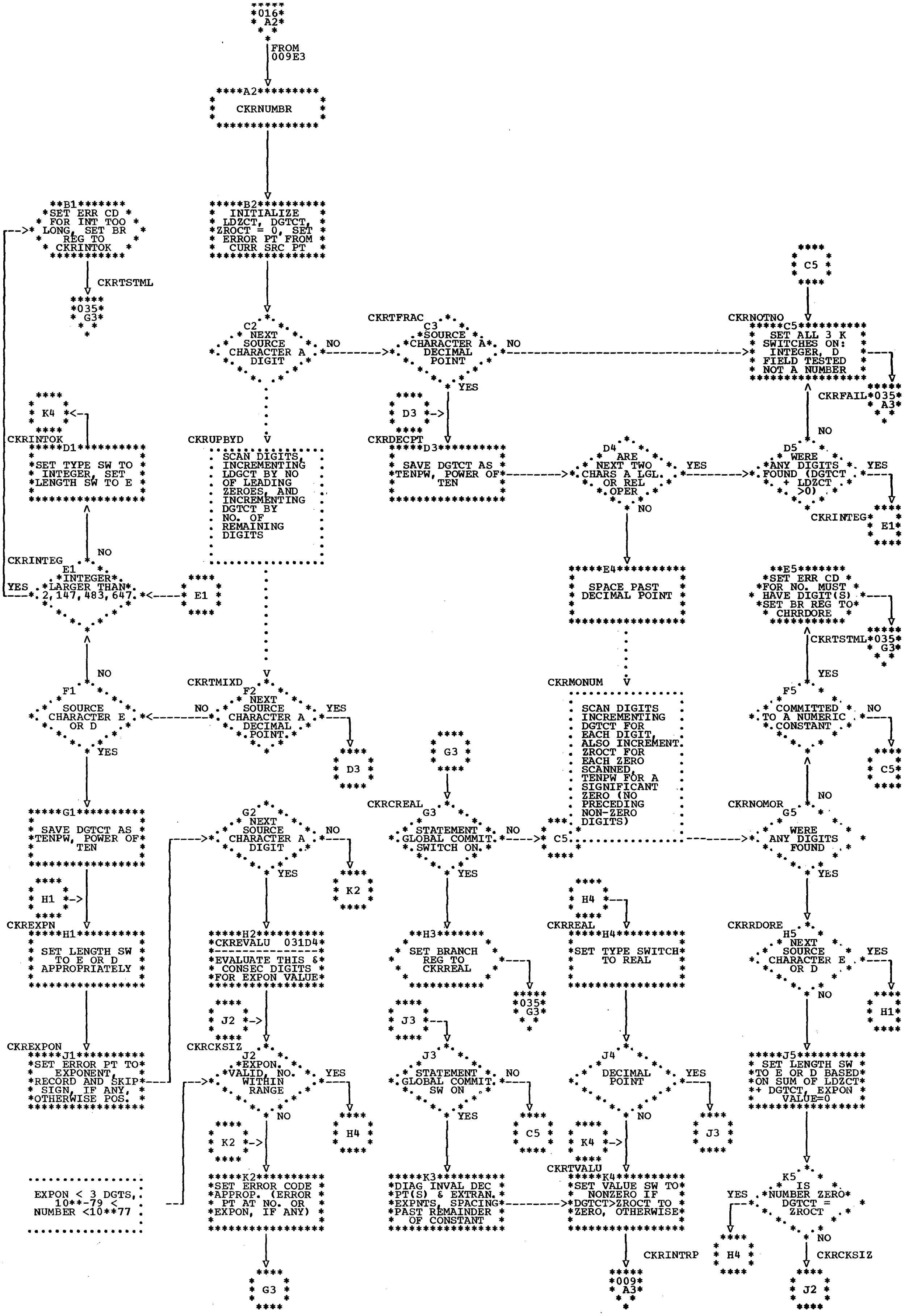

016
A2
FROM 009E3
A2
CKRNUMBR
B1
SET ERR CD FOR INT TOO LONG, SET BR REG TO CKRINTOK
CKRTSTML
035 G3
B2
INITIALIZE LDZCT, DGTCT, ZROCT = 0, SET ERROR PT FROM CURR SRC PT
C5
C2
NEXT SOURCE CHARACTER A DIGIT
NO
CKRTFRAC
C3
SOURCE CHARACTER A DECIMAL POINT
NO
YES
CKRNOTNO
C5
SET ALL 3 K SWITCHES ON: INTEGER, D FIELD TESTED NOT A NUMBER
CKRFAIL
035 A3
K4
CKRINTOK
D1
SET TYPE SW TO INTEGER, SET LENGTH SW TO E
CKRUPBYD
SCAN DIGITS, INCREMENTING LDGCT BY NO OF LEADING ZEROES, AND INCREMENTING DGTCT BY NO. OF REMAINING DIGITS
D3
CKRDECPT
D3
SAVE DGTCT AS TENPW, POWER OF TEN
D4
ARE NEXT TWO CHARS A LGL. OR REL OPER
YES
NO
D5
WERE ANY DIGITS FOUND (DGTCT + LDZCT >0)
YES
NO
CKRINTEG
E1
CKRINTEG
E1
INTEGER LARGER THAN 2,147,483,647
YES
NO
E1
E4
SPACE PAST DECIMAL POINT
E5
SET ERR CD FOR NO. MUST HAVE DIGIT(S) SET BR REG TO CHRRDORE
CKRTSTML
035 G3
YES
NO
F1
SOURCE CHARACTER E OR D
YES
CKRTMIXD
F2
NEXT SOURCE CHARACTER A DECIMAL POINT
NO
YES
D3
CKRMONUM
SCAN DIGITS INCREMENTING DGTCT FOR EACH DIGIT, ALSO INCREMENT ZROCT FOR EACH ZERO SCANNED, TENPW FOR A SIGNIFICANT ZERO (NO PRECEDING NON-ZERO DIGITS)
F5
COMMITTED TO A NUMERIC CONSTANT
NO
C5
NO
G3
G1
SAVE DGTCT AS TENPW, POWER OF TEN
G2
NEXT SOURCE CHARACTER A DIGIT
NO
YES
K2
CKRCREAL
G3
STATEMENT GLOBAL COMMIT. SWITCH ON.
NO
C5
YES
CKRNOMOR
G5
WERE ANY DIGITS FOUND
YES
H1
CKREXPN
H1
SET LENGTH SW TO E OR D APPROPRIATELY
H2
CKREVALU 031D4
EVALUATE THIS 6 CONSEC DIGITS FOR EXPON VALUE
H3
SET BRANCH REG TO CKRREAL
035 G3
H4
CKRREAL
H4
SET TYPE SWITCH TO REAL
CKRRDORE
H5
NEXT SOURCE CHARACTER E OR D
YES
NO
H1
J2
CKREXPON
J1
SET ERROR PT TO EXPONENT, RECORD AND SKIP SIGN, IF ANY, OTHERWISE POS.
CKRCKSIZ
J2
EXPON. VALID NO. WITHIN RANGE
YES
NO
H4
J3
J3
STATEMENT GLOBAL COMMIT. SW ON
NO
YES
C5
J4
DECIMAL POINT
YES
NO
J3
J5
SET LENGTH SW TO E OR D BASED ON SUM OF LDZCT + DGTCT, EXPON VALUE=0
K2
K4
CKRTVALU
EXPON < 3 DGTS, 10**-79 < NUMBER <10**77
K2
SET ERROR CODE APPROP. (ERROR PT AT NO. OR EXPON, IF ANY)
K3
DIAG INVAL DEC PT(S) &amp; EXTRAN. EXPNTS, SPACING PAST REMAINDER OF CONSTANT
K4
SET VALUE SW TO NONZERO IF DGTCT&gt;ZROCT TO ZERO, OTHERWISE
CKRINTRP
009 A3
K5
IS NUMBER ZERO DGTCT = ZROCT
YES
NO
H4
CKRCKSIZ
J2
G3

Chart 017. IPDSNCKR (CKRSTATM)

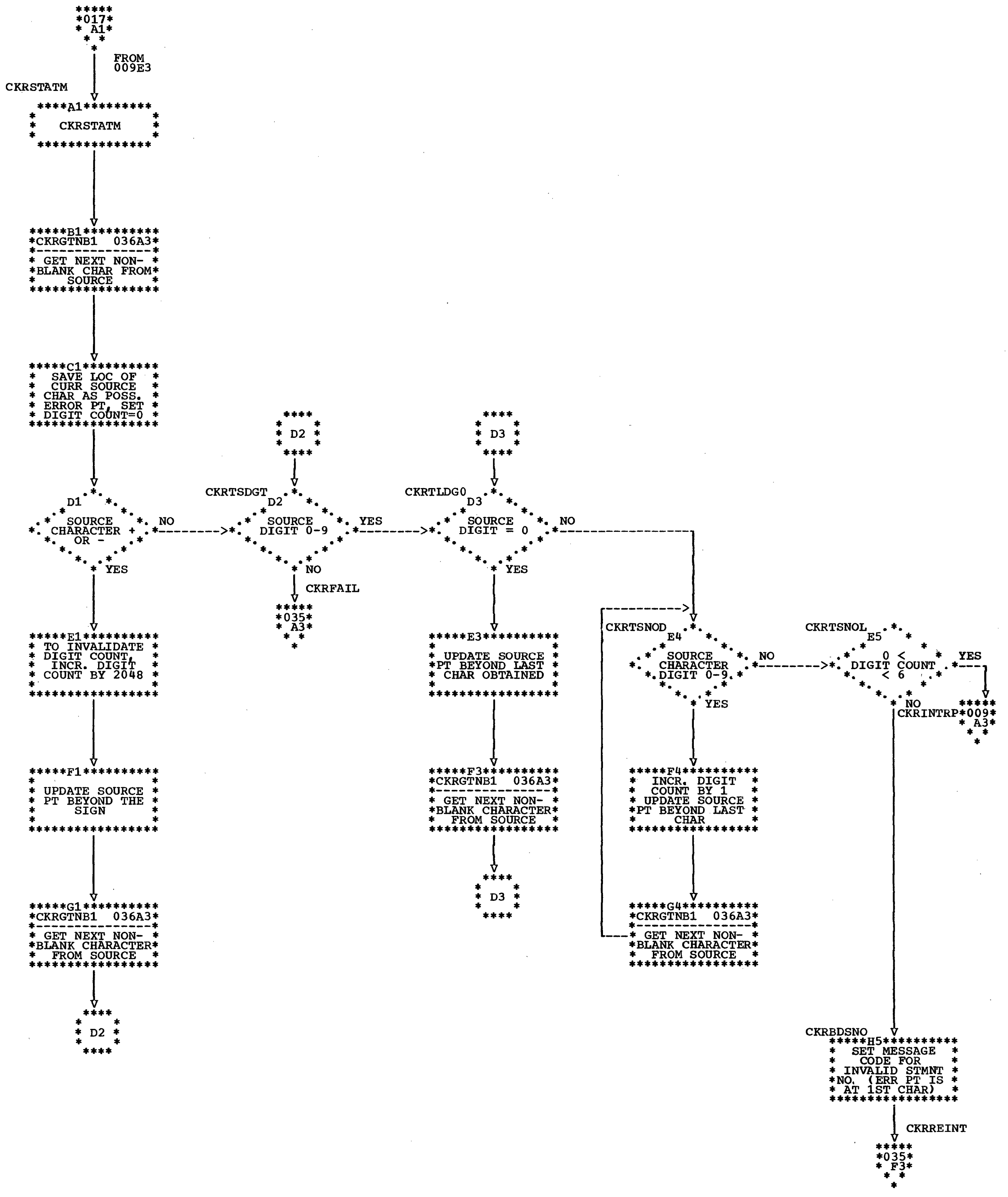

017
A1
FROM 009E3
CKRSTATM
A1
CKRSTATM
B1
CKRGTNB1 036A3
GET NEXT NON-BLANK CHAR FROM SOURCE
C1
SAVE LOC OF CURR SOURCE CHAR AS POSS. ERROR PT, SET DIGIT COUNT=0
D1
SOURCE CHARACTER + OR -
NO
YES
E1
TO INVALIDATE DIGIT COUNT, INCR. DIGIT COUNT BY 2048
F1
UPDATE SOURCE PT BEYOND THE SIGN
G1
CKRGTNB1 036A3
GET NEXT NON-BLANK CHARACTER FROM SOURCE
D2
D2
CKRTSDGT
D2
SOURCE DIGIT 0-9
YES
NO
CKRFAIL
035
A3
D3
CKRTLDG0
D3
SOURCE DIGIT = 0
NO
YES
E3
UPDATE SOURCE PT BEYOND LAST CHAR OBTAINED
F3
CKRGTNB1 036A3
GET NEXT NON-BLANK CHARACTER FROM SOURCE
D3
CKRTSNOD
E4
SOURCE CHARACTER DIGIT 0-9
NO
YES
F4
INCR. DIGIT COUNT BY 1 UPDATE SOURCE PT BEYOND LAST CHAR
G4
CKRGTNB1 036A3
GET NEXT NON-BLANK CHARACTER FROM SOURCE
CKRTSNOL
E5
0 < DIGIT COUNT < 6
YES
CKRINTRP
009
A3
NO
CKRBDSNO
H5
SET MESSAGE CODE FOR INVALID STMNT NO. (ERR PT IS AT 1ST CHAR)
CKRREINT
035
F3

Chart 018. IPDSNCKR (CKRHOLLR)

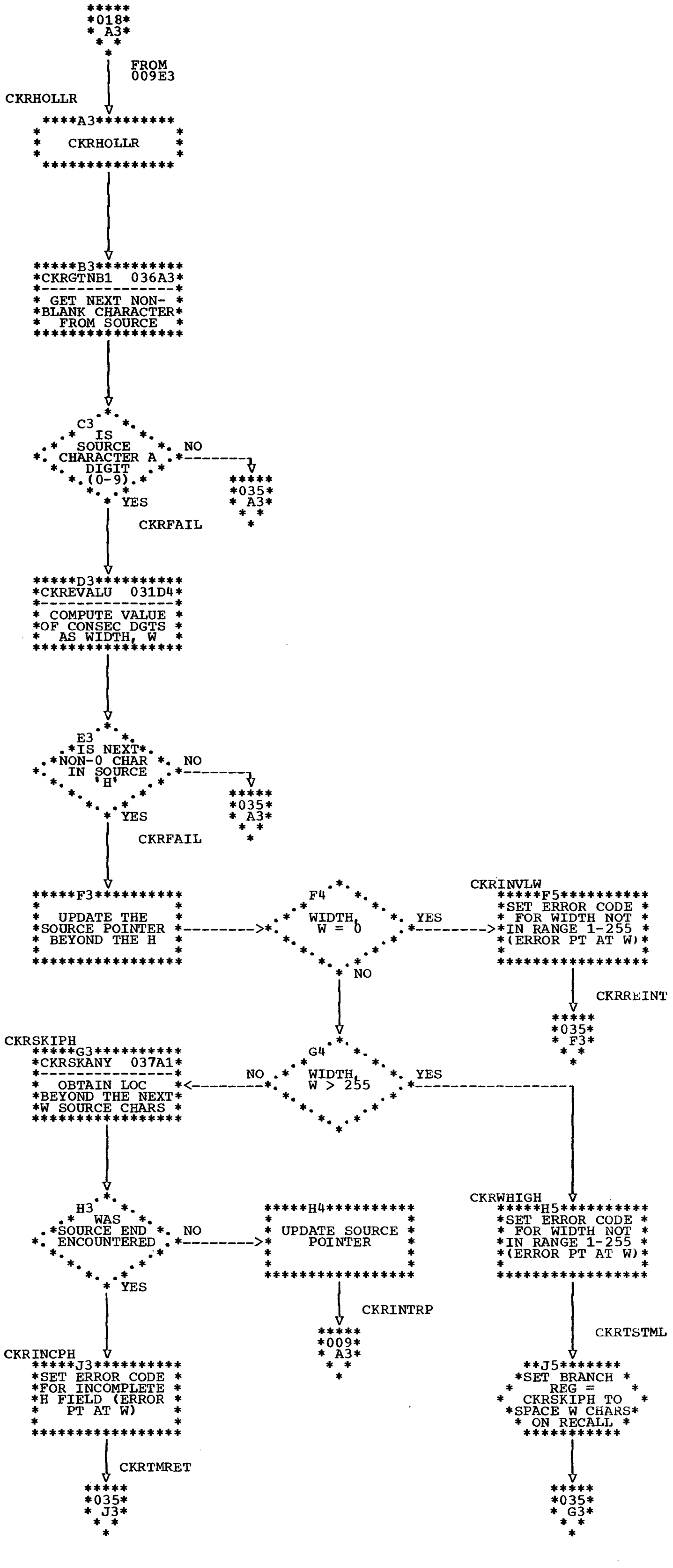
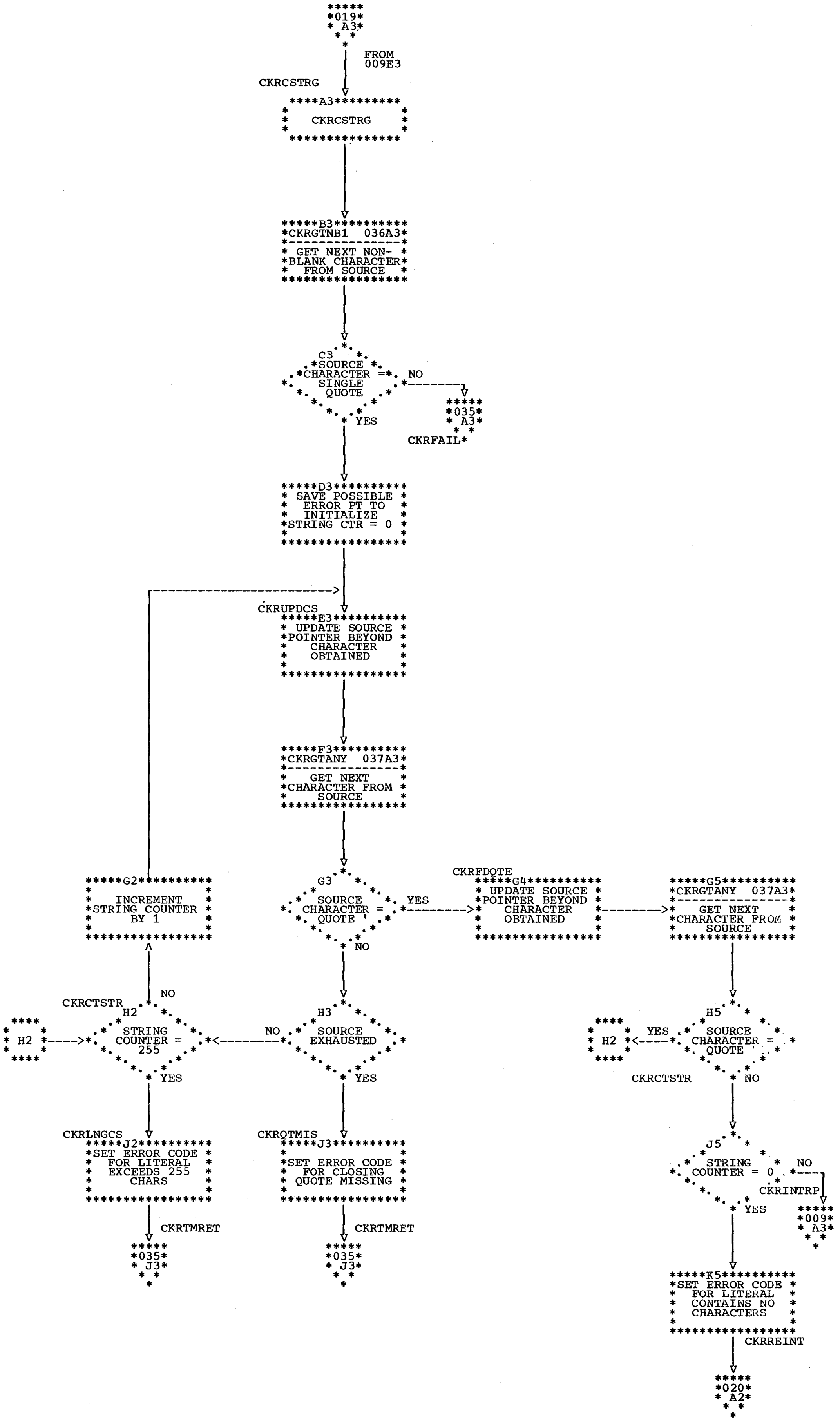

018 A3
FROM 009E3
CKRHOLLR
A3
CKRHOLLR
B3
CKRGTNB1 036A3
GET NEXT NON-BLANK CHARACTER FROM SOURCE
C3
IS SOURCE CHARACTER A DIGIT (0-9)
NO
035 A3
YES
CKRFAIL
D3
CKREVALU 031D4
COMPUTE VALUE OF CONSEC DGTS AS WIDTH, W
E3
IS NEXT NON-0 CHAR IN SOURCE 'H'
NO
035 A3
YES
CKRFAIL
F3
UPDATE THE SOURCE POINTER BEYOND THE H
F4
WIDTH, W = 0
YES
CKRINVLW
F5
SET ERROR CODE FOR WIDTH NOT IN RANGE 1-255 (ERROR PT AT W)
CKRREINT
035 F3
NO
CKRSKIPH
G3
CKRSKANY 037A1
OBTAIN LOC BEYOND THE NEXT W SOURCE CHARS
G4
NO
WIDTH, W > 255
YES
H3
WAS SOURCE END ENCOUNTERED
NO
H4
UPDATE SOURCE POINTER
CKRINTRP
009 A3
CKRWHIGH
H5
SET ERROR CODE FOR WIDTH NOT IN RANGE 1-255 (ERROR PT AT W)
CKRTSTML
YES
CKRINCPH
J3
SET ERROR CODE FOR INCOMPLETE H FIELD (ERROR PT AT W)
CKRTMRET
035 J3
J5
SET BRANCH REG = CKRSKIPH TO SPACE W CHARS ON RECALL
035 G3

Chart 019. IPDSNCKR (CKRCSTRG)



019
A3
FROM 009E3
CKRCSTRG
A3
CKRCSTRG
B3
CKRGTNB1 036A3
GET NEXT NON-BLANK CHARACTER FROM SOURCE
C3
SOURCE CHARACTER = SINGLE QUOTE
NO
035
A3
CKRFAIL*
YES
D3
SAVE POSSIBLE ERROR PT TO INITIALIZE STRING CTR = 0
CKRUPDCS
E3
UPDATE SOURCE POINTER BEYOND CHARACTER OBTAINED
F3
CKRGTANY 037A3
GET NEXT CHARACTER FROM SOURCE
G2
INCREMENT STRING COUNTER BY 1
G3
SOURCE CHARACTER = QUOTE '
YES
CKRFDQTE
G4
UPDATE SOURCE POINTER BEYOND CHARACTER OBTAINED
G5
CKRGTANY 037A3
GET NEXT CHARACTER FROM SOURCE
NO
NO
CKRCTSTR
H2
H2
STRING COUNTER = 255
NO
H3
SOURCE EXHAUSTED
H5
H2
YES
SOURCE CHARACTER = QUOTE '
CKRCTSTR
NO
YES
YES
CKRLNGCS
J2
SET ERROR CODE FOR LITERAL EXCEEDS 255 CHARS
CKRQTMIS
J3
SET ERROR CODE FOR CLOSING QUOTE MISSING
J5
STRING COUNTER = 0
NO
CKRINTRP
YES
009
A3
CKRTMRET
035
J3
CKRTMRET
035
J3
K5
SET ERROR CODE FOR LITERAL CONTAINS NO CHARACTERS
CKRREINT
020
A2

Chart 020. IPDSNCKR (CKRQUOTE, CKRNOTQT)

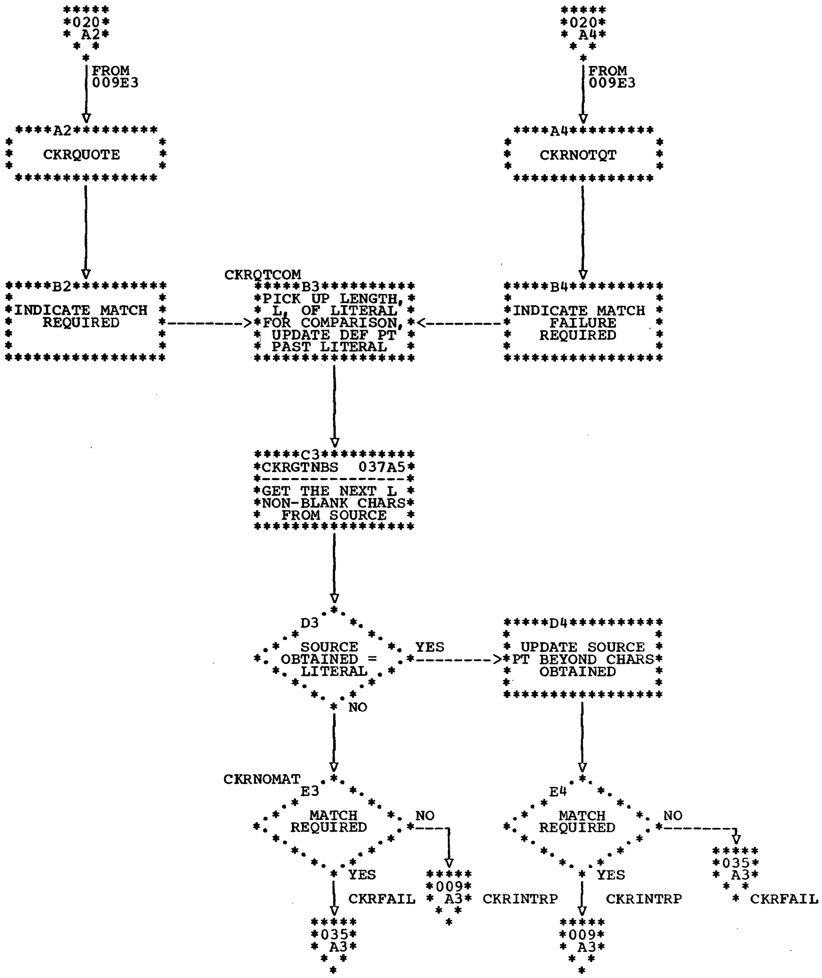

020 A2
FROM 009E3
A2 CKRQUOTE
B2 INDICATE MATCH REQUIRED
020 A4
FROM 009E3
A4 CKRNOTQT
B4 INDICATE MATCH FAILURE REQUIRED
CKRQTCOM
B3 PICK UP LENGTH, L, OF LITERAL FOR COMPARISON, UPDATE DEF PT PAST LITERAL
C3 CKRGTNBS 037A5
GET THE NEXT L NON-BLANK CHARS FROM SOURCE
D3 SOURCE OBTAINED = LITERAL
YES
NO
D4 UPDATE SOURCE PT BEYOND CHARS OBTAINED
CKRNOMAT
E3 MATCH REQUIRED
NO
YES
CKRFAIL
035 A3
009 A3
CKRINTRP
E4 MATCH REQUIRED
NO
YES
CKRINTRP
009 A3
035 A3
CKRFAIL

Chart 021. IPDSNCKR (CKRSCAN, CKRSCANF)

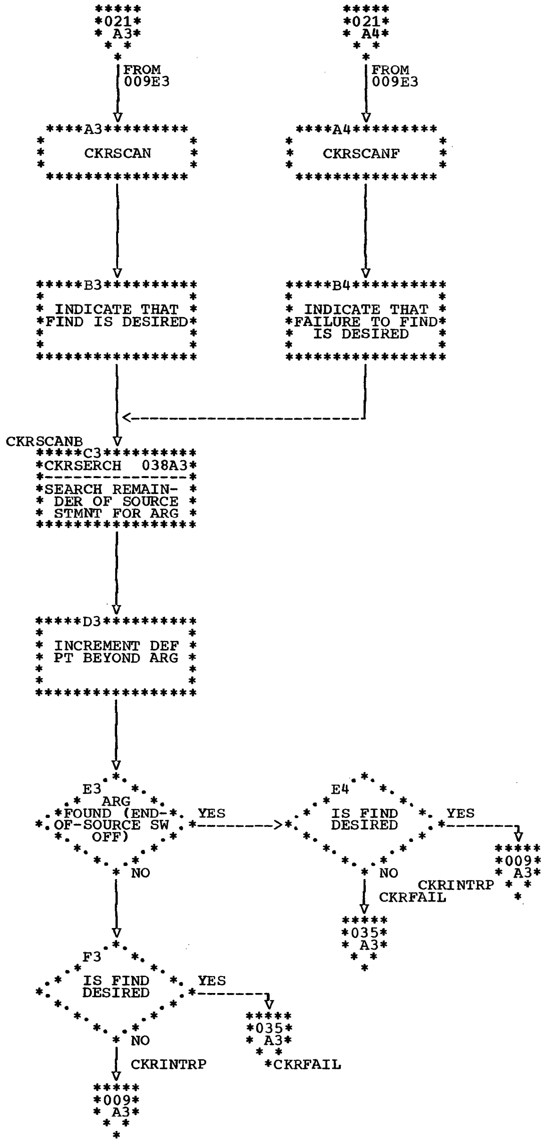
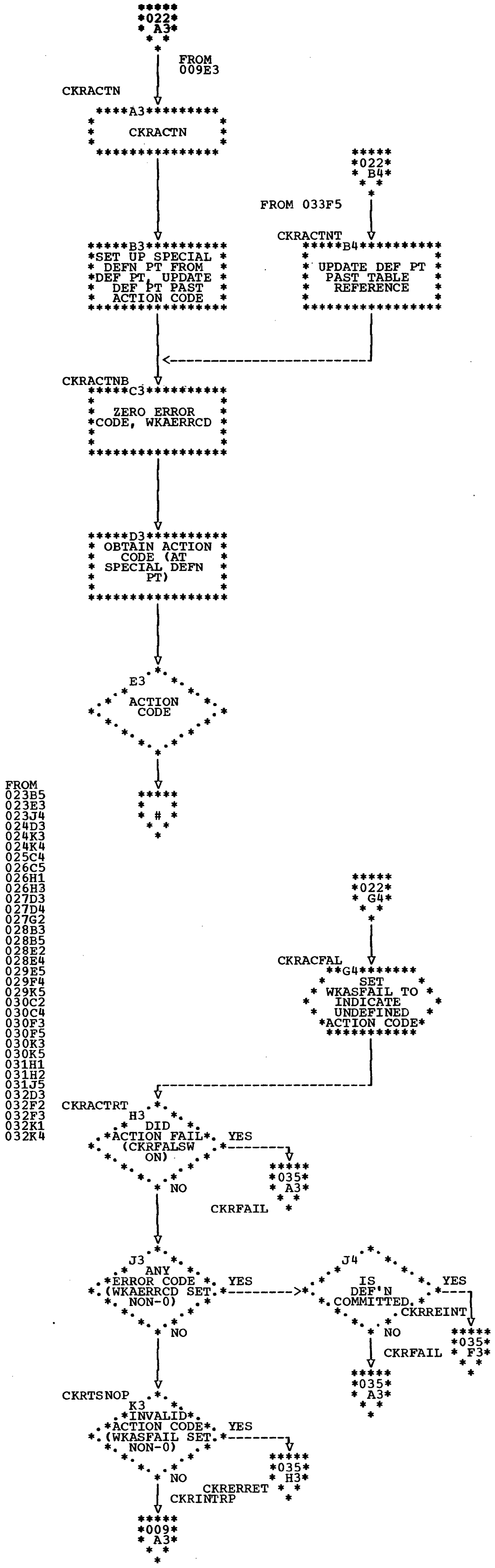


Chart 022. IPDSNCKR (CKRACTN)



E3.	...100...	023	A3
	...101...	023	D1
	...102...	023	G3
	...103...	024	A3
	...104...	024	E1
	...105...	024	E4
	...106...	025	A2
	...200...	026	A2
	...201...	026	E1
	...202...	026	E3
	...300...	027	A3
	...301...	027	F3
	...400...	028	A2
	...401...	028	A4
	...500...	029	A3
	...600...	030	A2
	...601...	030	A4
	...602...	030	D3
	...603...	030	H2
	...700...	031	A1
	...701...	032	A2
	...800...	032	H2
	...801...	032	J3
	...NONE..	022	G4

Chart 023. IPDSNCKR (CKRAR100, CKRAR101, CKRAR102)

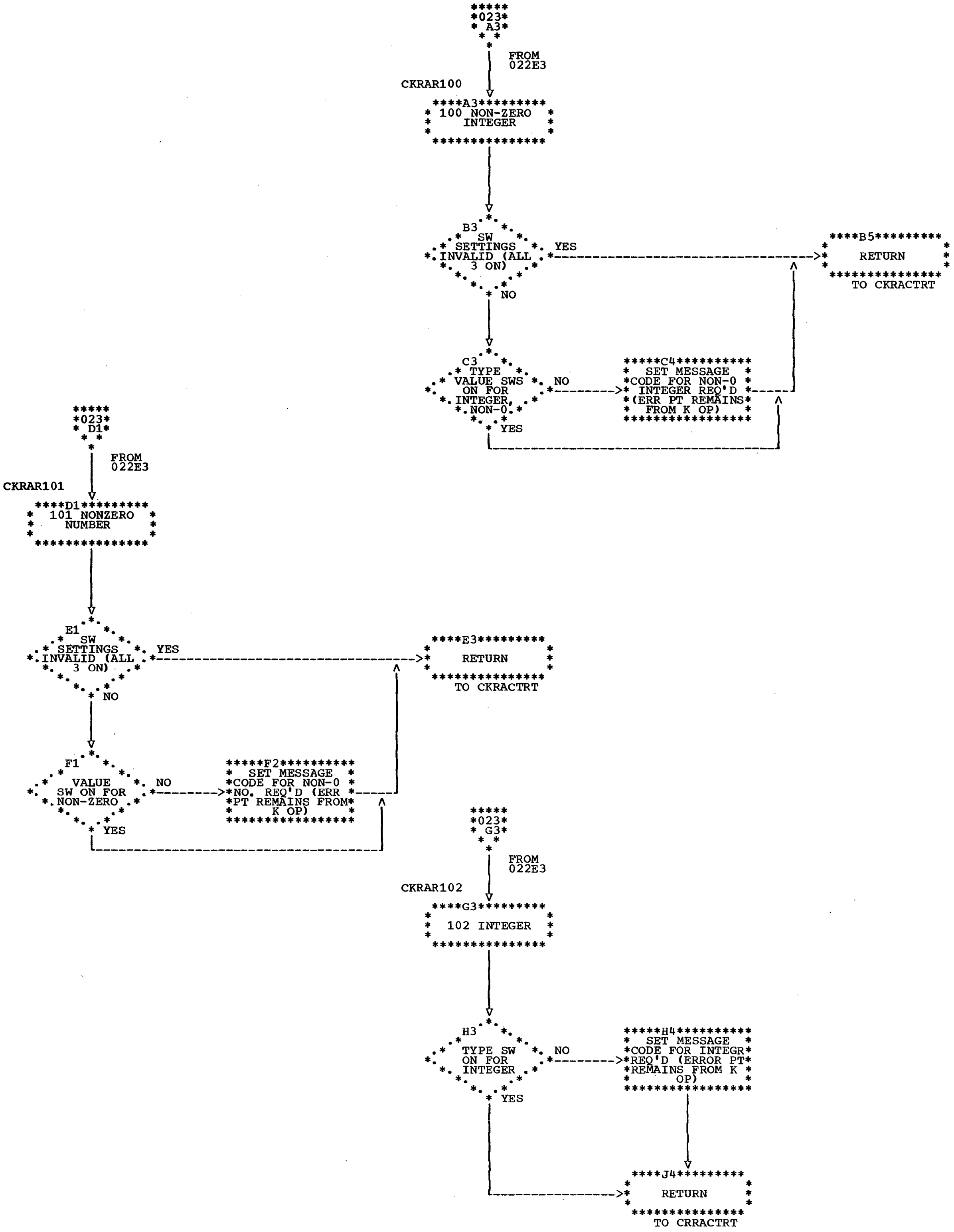

023 A3
FROM 022E3
CKRAR100
A3 100 NON-ZERO INTEGER
B3 SW SETTINGS INVALID (ALL 3 ON)
YES
NO
B5 RETURN
TO CKRACTRT
C3 TYPE VALUE SWS ON FOR INTEGER, NON-0
NO
YES
C4 SET MESSAGE CODE FOR NON-0 INTEGER REQ'D (ERR PT REMAINS FROM K OP)
023 D1
FROM 022E3
CKRAR101
D1 101 NONZERO NUMBER
E1 SW SETTINGS INVALID (ALL 3 ON)
YES
NO
E3 RETURN
TO CKRACTRT
F1 VALUE SW ON FOR NON-ZERO
NO
YES
F2 SET MESSAGE CODE FOR NON-0 NO. REQ'D (ERR PT REMAINS FROM K OP)
023 G3
FROM 022E3
CKRAR102
G3 102 INTEGER
H3 TYPE SW ON FOR INTEGER
NO
YES
H4 SET MESSAGE CODE FOR INTEGR REQ'D (ERROR PT REMAINS FROM K OP)
J4 RETURN
TO CRRACTRT

Chart 024. IPDSNCKR (CKRAR103, CKRAR104, CKRAR105)

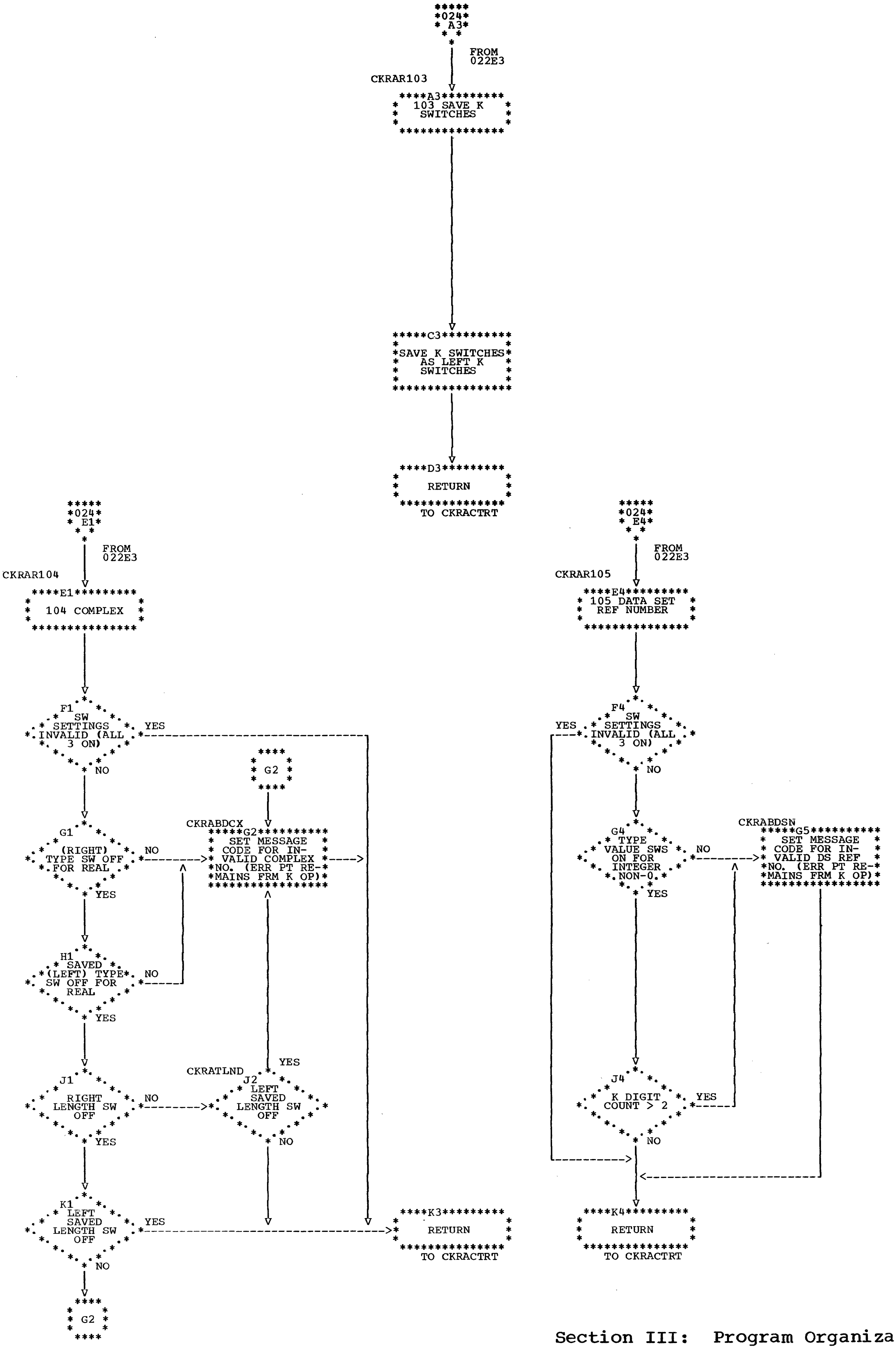


Chart 025. IPDSNCKR (CKRAR106)

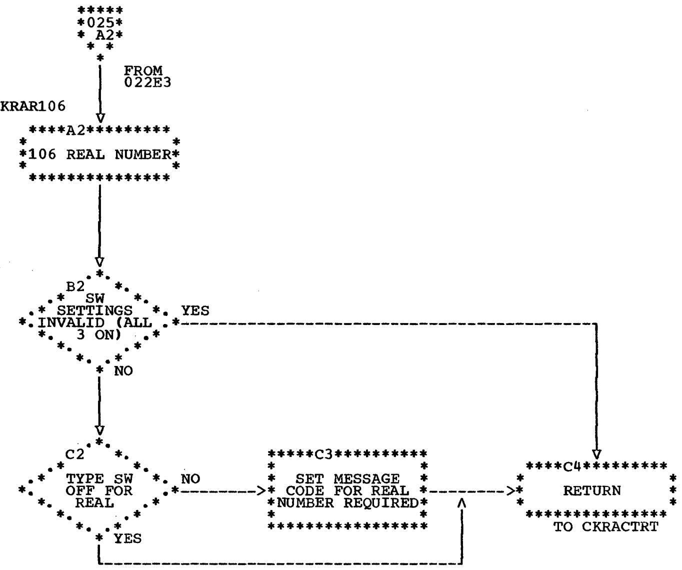


Chart 026. IPDSNCKR (CKRAR200, CKRAR201, CKRAR202)

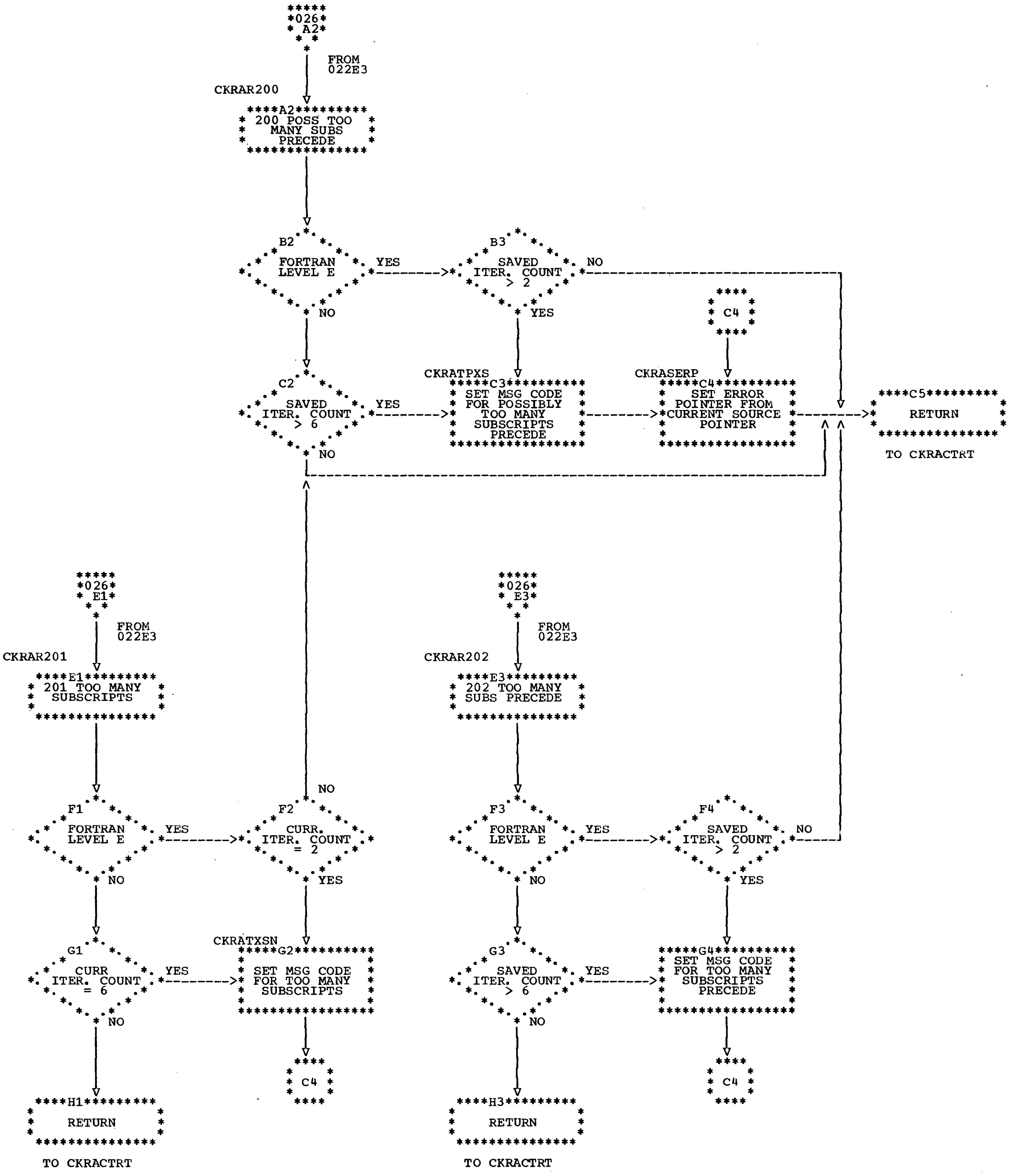


Chart 027. IPDSNCKR (CKRAR300, CKRAR301)

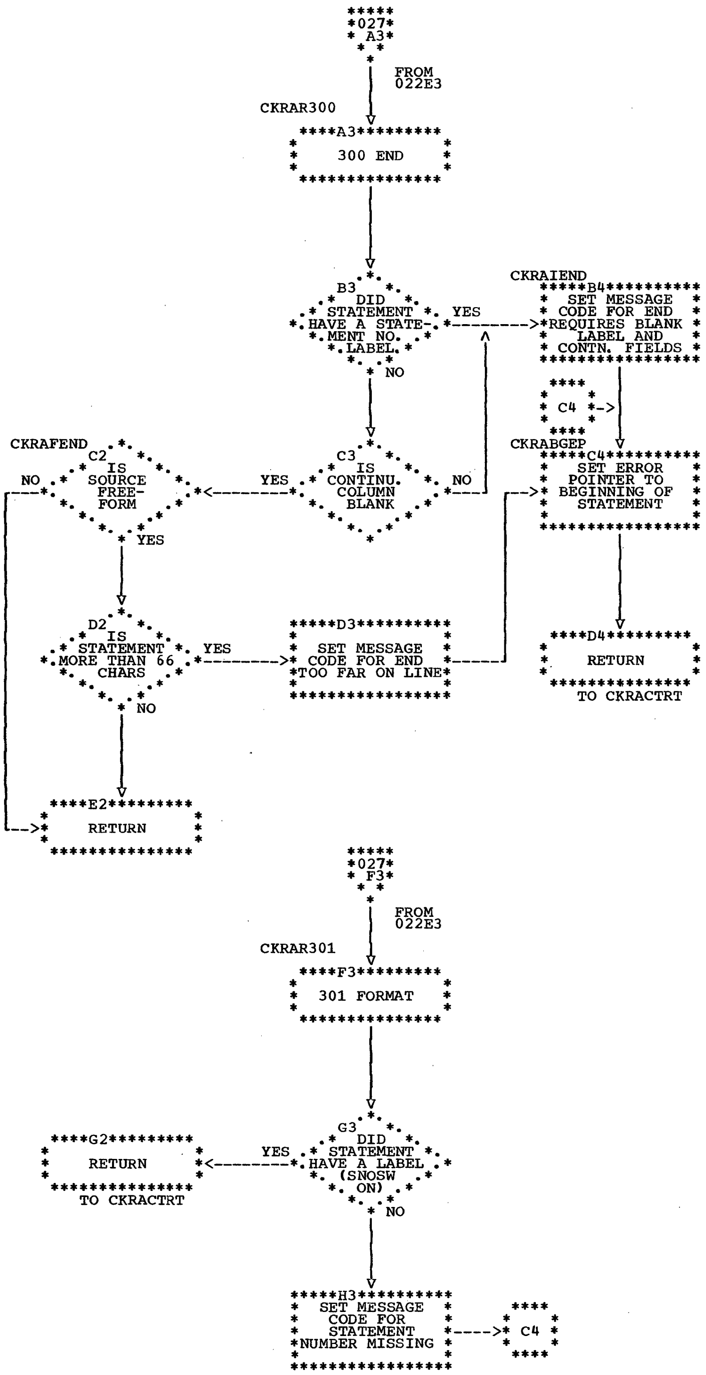

027 A3
FROM 022E3
CKRAR300
A3
300 END
B3
DID STATEMENT HAVE A STATEMENT NO. LABEL
YES
NO
CKRAIEND
B4
SET MESSAGE CODE FOR END REQUIRES BLANK LABEL AND CONTN. FIELDS
C4
CKRABGEP
C4
SET ERROR POINTER TO BEGINNING OF STATEMENT
D4
RETURN
TO CKRACTRT
CKRAFEND
C2
IS SOURCE FREE-FORM
NO
YES
C3
IS CONTINU. COLUMN BLANK
YES
NO
D2
IS STATEMENT MORE THAN 66 CHARS
YES
NO
D3
SET MESSAGE CODE FOR END TOO FAR ON LINE
E2
RETURN
027 F3
FROM 022E3
CKRAR301
F3
301 FORMAT
G3
DID STATEMENT HAVE A LABEL (SNOSW ON)
YES
NO
G2
RETURN
TO CKRACTRT
H3
SET MESSAGE CODE FOR STATEMENT NUMBER MISSING
C4

Chart 028. IPDSNCKR (CKRAR400, CKRAR401)

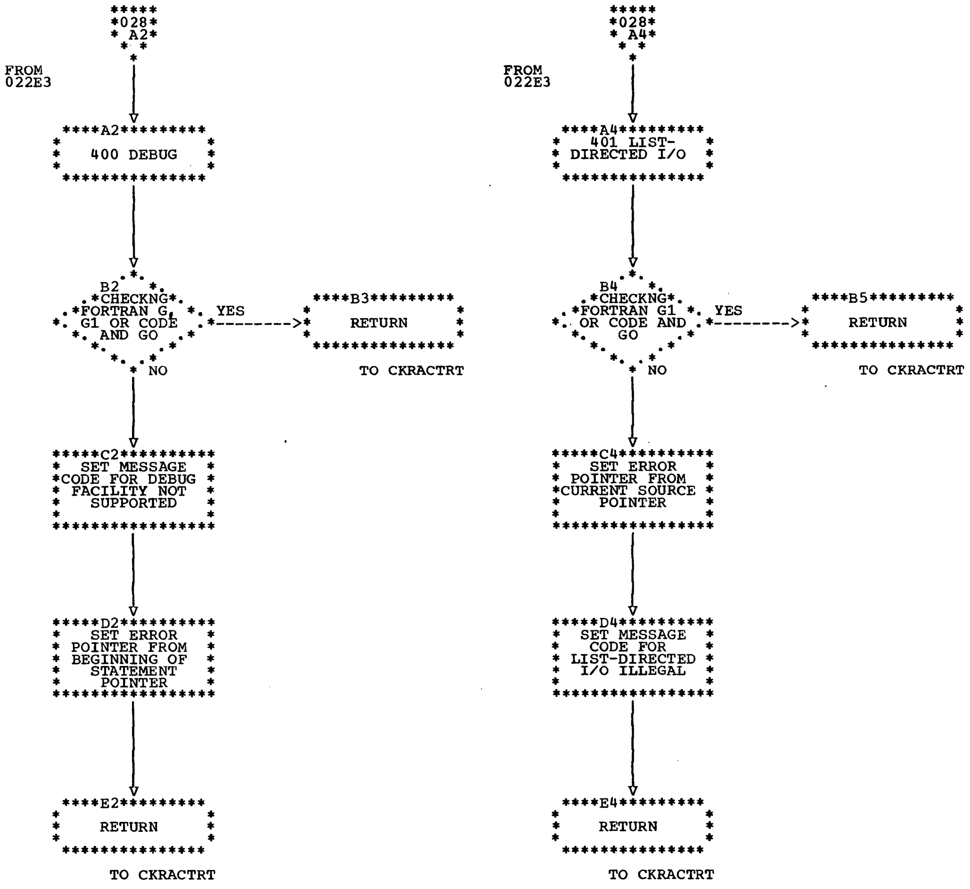

028 A2
FROM 022E3
A2
400 DEBUG
B2
CHECKNG FORTRAN G, G1 OR CODE AND GO
YES
B3
RETURN
TO CKRACTRT
NO
C2
SET MESSAGE CODE FOR DEBUG FACILITY NOT SUPPORTED
D2
SET ERROR POINTER FROM BEGINNING OF STATEMENT POINTER
E2
RETURN
TO CKRACTRT
028 A4
FROM 022E3
A4
401 LIST-DIRECTED I/O
B4
CHECKNG FORTRAN G1 OR CODE AND GO
YES
B5
RETURN
TO CKRACTRT
NO
C4
SET ERROR POINTER FROM CURRENT SOURCE POINTER
D4
SET MESSAGE CODE FOR LIST-DIRECTED I/O ILLEGAL
E4
RETURN
TO CKRACTRT

Chart 029. IPDSNCKR (CKRAR500)

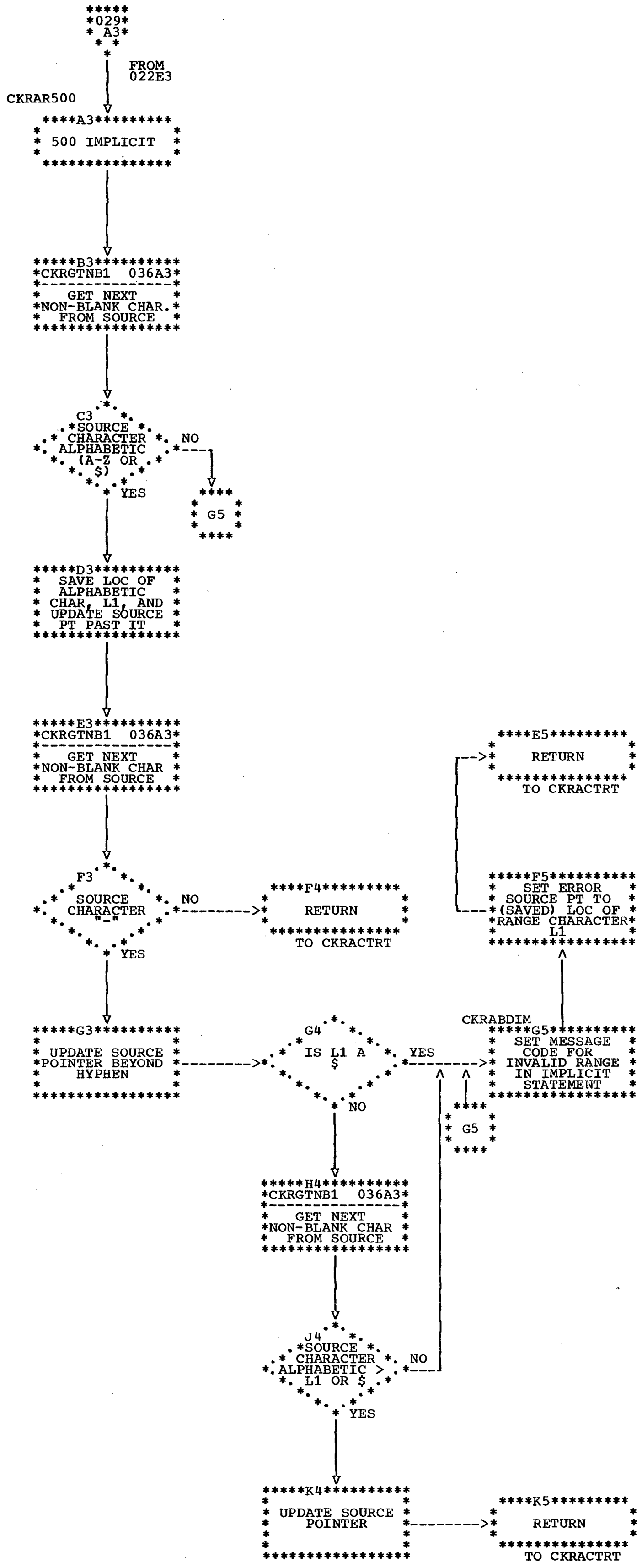

029
A3
FROM 022E3
CKRAR500
A3
500 IMPLICIT
B3
CKRGTNB1 036A3
GET NEXT NON-BLANK CHAR. FROM SOURCE
C3
SOURCE CHARACTER ALPHABETIC (A-Z OR $)
NO
G5
YES
D3
SAVE LOC OF ALPHABETIC CHAR, L1, AND UPDATE SOURCE PT PAST IT
E3
CKRGTNB1 036A3
GET NEXT NON-BLANK CHAR FROM SOURCE
E5
RETURN
TO CKRACTRT
F3
SOURCE CHARACTER "-"
NO
F4
RETURN
TO CKRACTRT
F5
SET ERROR SOURCE PT TO (SAVED) LOC OF RANGE CHARACTER L1
YES
G3
UPDATE SOURCE POINTER BEYOND HYPHEN
G4
IS L1 A $
YES
CKRABDIM
G5
SET MESSAGE CODE FOR INVALID RANGE IN IMPLICIT STATEMENT
G5
NO
H4
CKRGTNB1 036A3
GET NEXT NON-BLANK CHAR FROM SOURCE
J4
SOURCE CHARACTER ALPHABETIC > L1 OR $
NO
YES
K4
UPDATE SOURCE POINTER
K5
RETURN
TO CKRACTRT

Chart 030. IPDSNCKR (CKRAR600, CKRAR601, CKRAR602, CKRAR603)

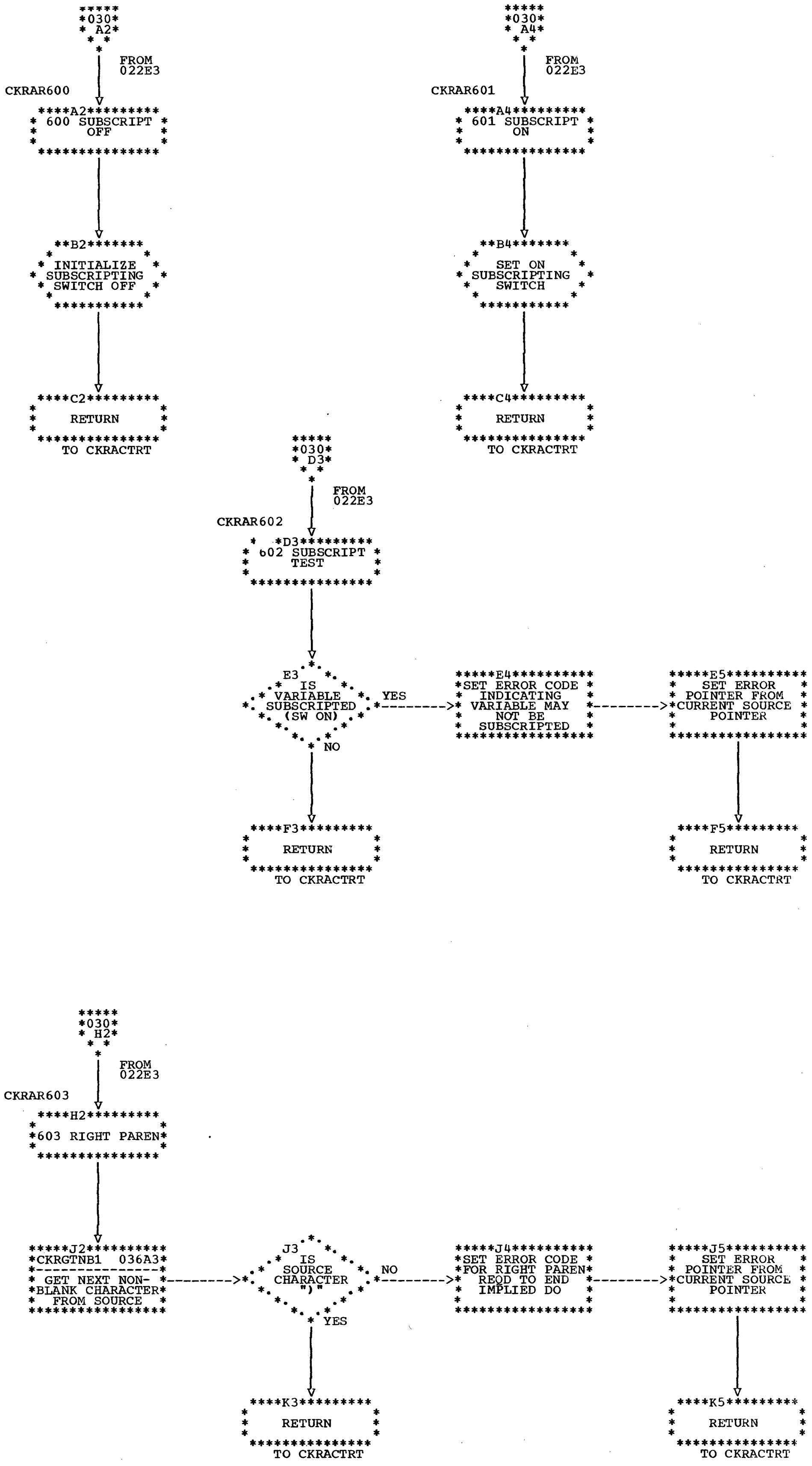

030 A2
FROM 022E3
CKRAR600
A2 600 SUBSCRIPT OFF
B2 INITIALIZE SUBSCRIPTING SWITCH OFF
C2 RETURN
TO CKRACTRT
030 A4
FROM 022E3
CKRAR601
A4 601 SUBSCRIPT ON
B4 SET ON SUBSCRIPTING SWITCH
C4 RETURN
TO CKRACTRT
030 D3
FROM 022E3
CKRAR602
D3 602 SUBSCRIPT TEST
E3 IS VARIABLE SUBSCRIPTED (SW ON)
YES
NO
E4 SET ERROR CODE INDICATING VARIABLE MAY NOT BE SUBSCRIPTED
E5 SET ERROR POINTER FROM CURRENT SOURCE POINTER
F3 RETURN
TO CKRACTRT
F5 RETURN
TO CKRACTRT
030 H2
FROM 022E3
CKRAR603
H2 603 RIGHT PAREN
J2 CKRGTNB1 036A3
GET NEXT NON-BLANK CHARACTER FROM SOURCE
J3 IS SOURCE CHARACTER ")"
NO
YES
J4 SET ERROR CODE FOR RIGHT PAREN REQD TO END IMPLIED DO
J5 SET ERROR POINTER FROM CURRENT SOURCE POINTER
K3 RETURN
TO CKRACTRT
K5 RETURN
TO CKRACTRT

Chart 031. IPDSNCKR (CKRAR700, CKREVALU)

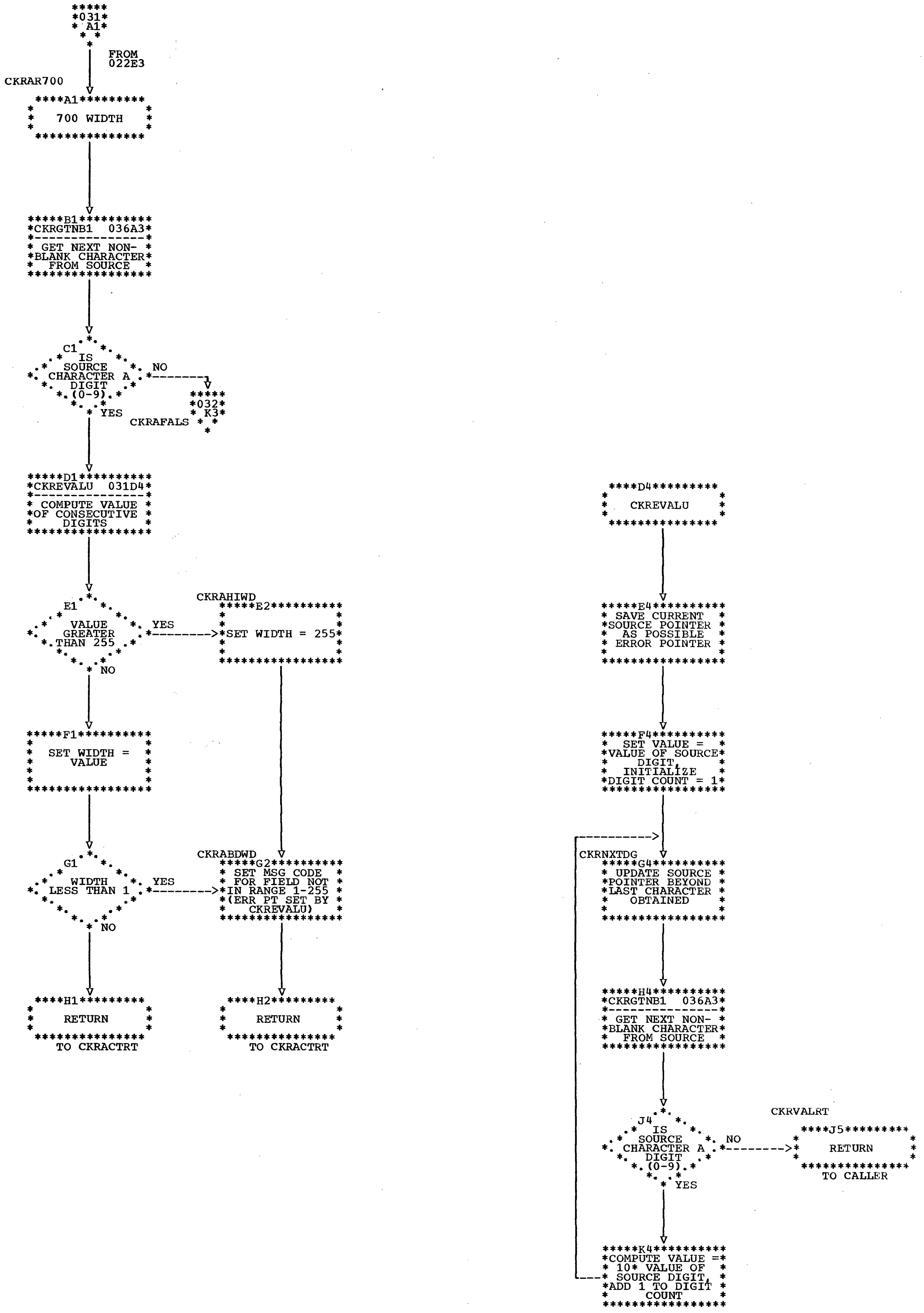

031 A1
FROM 022E3
CKRAR700
A1
700 WIDTH
B1
CKRGTNB1 036A3
GET NEXT NON-BLANK CHARACTER FROM SOURCE
C1
IS SOURCE CHARACTER A DIGIT (0-9)
NO
YES
032 K3
CKRAFALS
D1
CKREVALU 031D4
COMPUTE VALUE OF CONSECUTIVE DIGITS
E1
VALUE GREATER THAN 255
YES
NO
CKRAHIWD
E2
SET WIDTH = 255
F1
SET WIDTH = VALUE
G1
WIDTH LESS THAN 1
YES
NO
CKRABDWD
G2
SET MSG CODE FOR FIELD NOT IN RANGE 1-255 (ERR PT SET BY CKREVALU)
H1
RETURN
TO CKRACTRT
H2
RETURN
TO CKRACTRT
D4
CKREVALU
E4
SAVE CURRENT SOURCE POINTER AS POSSIBLE ERROR POINTER
F4
SET VALUE = VALUE OF SOURCE DIGIT, INITIALIZE DIGIT COUNT = 1
CKRNXTDG
G4
UPDATE SOURCE POINTER BEYOND LAST CHARACTER OBTAINED
H4
CKRGTNB1 036A3
GET NEXT NON-BLANK CHARACTER FROM SOURCE
J4
IS SOURCE CHARACTER A DIGIT (0-9)
NO
YES
CKRVALRT
J5
RETURN
TO CALLER
K4
COMPUTE VALUE = 10* VALUE OF SOURCE DIGIT, ADD 1 TO DIGIT COUNT

Chart 032. IPDSNCKR (CKRAR701, CKRAR800, CKRAR801)

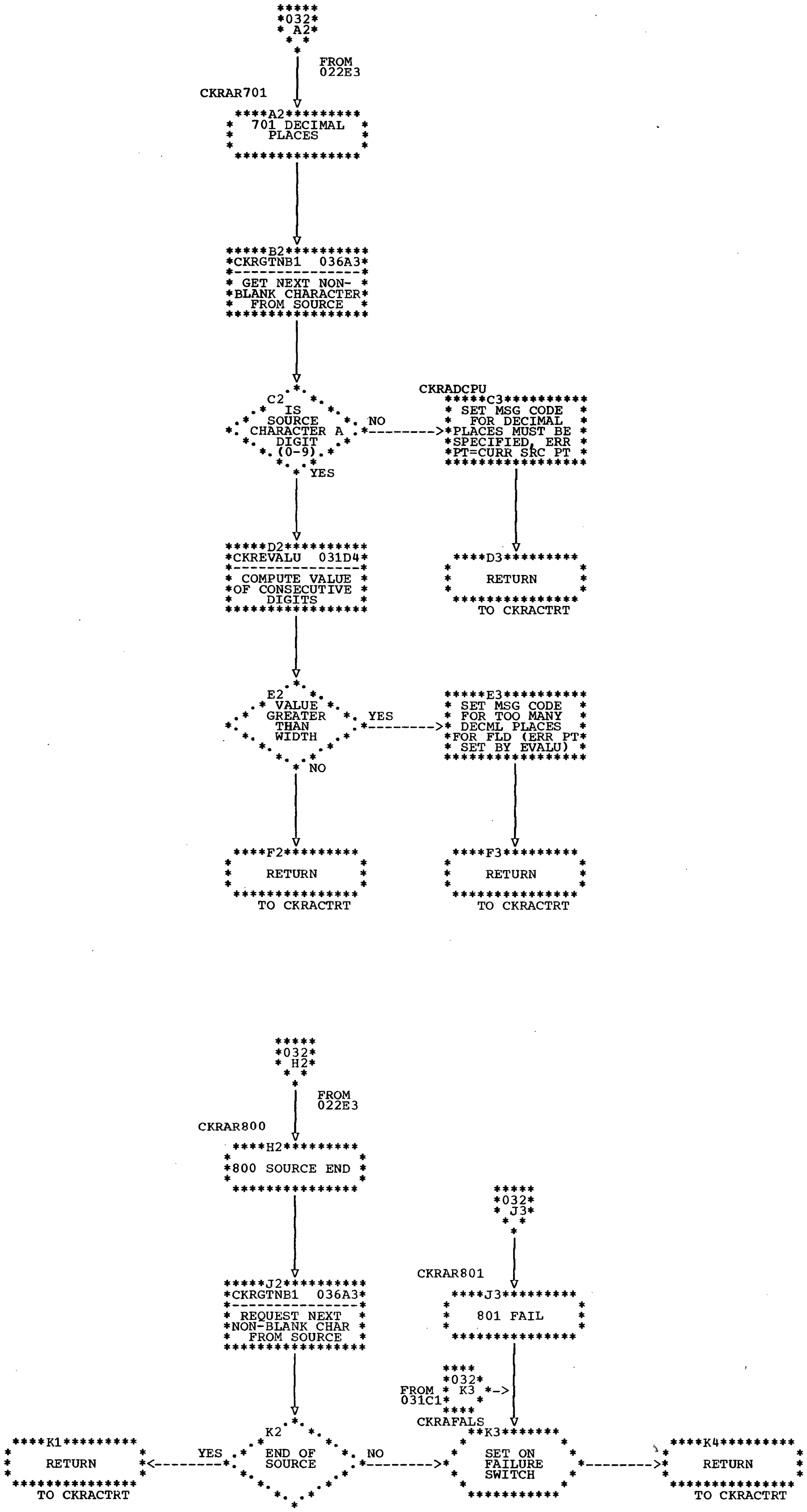


Chart 033. IPDSNCKR (CKRMESSG, CKRTABL)

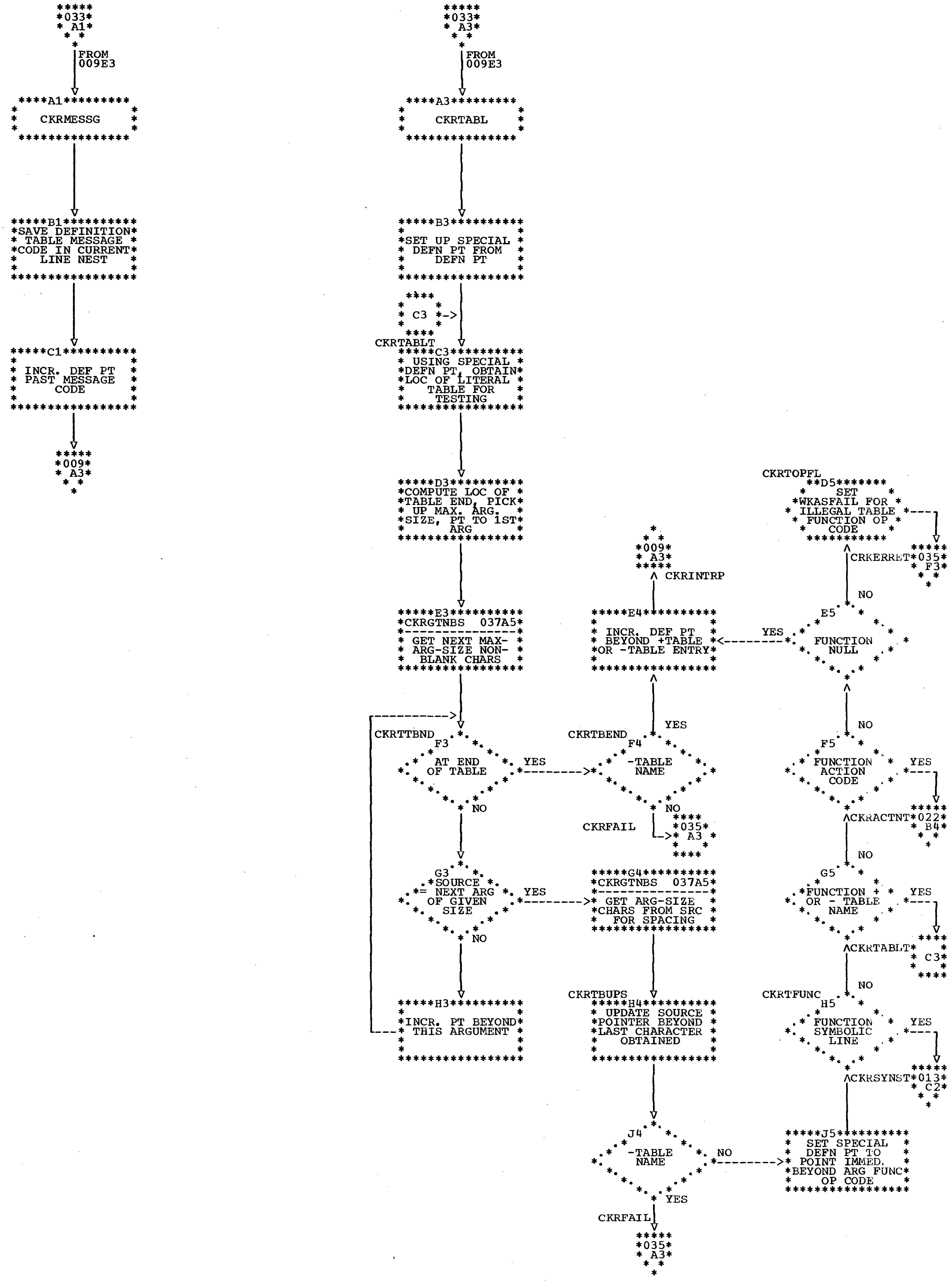

033 A1
FROM 009E3
A1 CKRMESSG
B1 SAVE DEFINITION TABLE MESSAGE CODE IN CURRENT LINE NEST
C1 INCR. DEF PT PAST MESSAGE CODE
009 A3
033 A3
FROM 009E3
A3 CKRTABL
B3 SET UP SPECIAL DEFN PT FROM DEFN PT
C3
CKRTABLT
C3 USING SPECIAL DEFN PT, OBTAIN LOC OF LITERAL TABLE FOR TESTING
D3 COMPUTE LOC OF TABLE END, PICK UP MAX. ARG. SIZE, PT TO 1ST ARG
E3 CKRGTNBS 037A5
GET NEXT MAX-ARG-SIZE NON-BLANK CHARS
CKRTTBND
F3 AT END OF TABLE
YES
NO
G3 SOURCE = NEXT ARG OF GIVEN SIZE
YES
NO
H3 INCR. PT BEYOND THIS ARGUMENT
009 A3
CKRINTRP
E4 INCR. DEF PT BEYOND +TABLE OR -TABLE ENTRY
YES
CKRTBEND
F4 -TABLE NAME
NO
CKRFAIL
035 A3
G4 CKRGTNBS 037A5
GET ARG-SIZE CHARS FROM SRC FOR SPACING
CKRTBUPS
H4 UPDATE SOURCE POINTER BEYOND LAST CHARACTER OBTAINED
J4 -TABLE NAME
NO
YES
CKRFAIL
035 A3
CKRTOPFL
D5 SET WKASFAIL FOR ILLEGAL TABLE FUNCTION OP CODE
CKRERRET 035 F3
NO
E5 FUNCTION NULL
YES
NO
F5 FUNCTION ACTION CODE
YES
CKRACTNT 022 B4
NO
G5 FUNCTION + OR - TABLE NAME
YES
CKRTABLT C3
NO
CKRTFUNC
H5 FUNCTION SYMBOLIC LINE
YES
CKRSYNST 013 C2
J5 SET SPECIAL DEFN PT TO POINT IMMED. BEYOND ARG FUNC OP CODE

Chart 034. IPDSNCKR (CKRSYUNS, CKRUNEST)

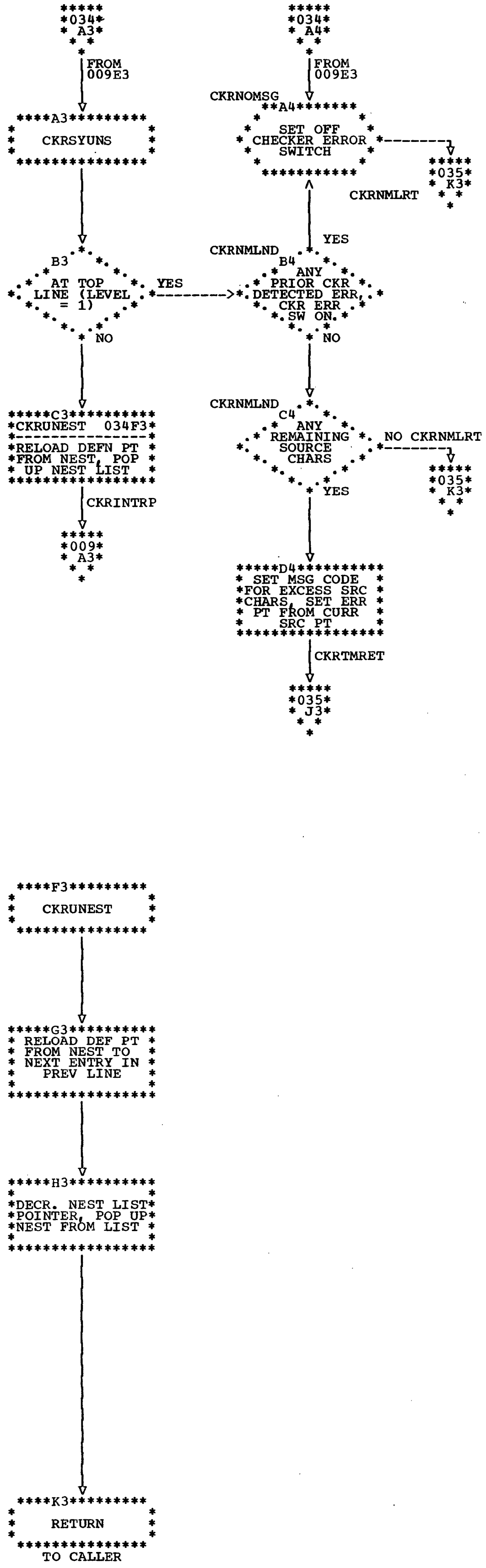

034 A3
FROM 009E3
A3
CKRSYUNS
B3
AT TOP LINE (LEVEL = 1)
YES
NO
C3
CKRUNEST 034F3
RELOAD DEFN PT FROM NEST, POP UP NEST LIST
CKRINTRP
009 A3
034 A4
FROM 009E3
CKRNOMSG
A4
SET OFF CHECKER ERROR SWITCH
035 K3
CKRNMLRT
YES
CKRNMLND
B4
ANY PRIOR CKR DETECTED ERR, CKR ERR SW ON.
NO
CKRNMLND
C4
ANY REMAINING SOURCE CHARS
NO CKRNMLRT
035 K3
YES
D4
SET MSG CODE FOR EXCESS SRC CHARS, SET ERR PT FROM CURR SRC PT
CKRTMRET
035 J3
F3
CKRUNEST
G3
RELOAD DEF PT FROM NEST TO NEXT ENTRY IN PREV LINE
H3
DECR. NEST LIST POINTER, POP UP NEST FROM LIST
K3
RETURN
TO CALLER

Chart 035. IPDSNCKR (CKRFAIL)

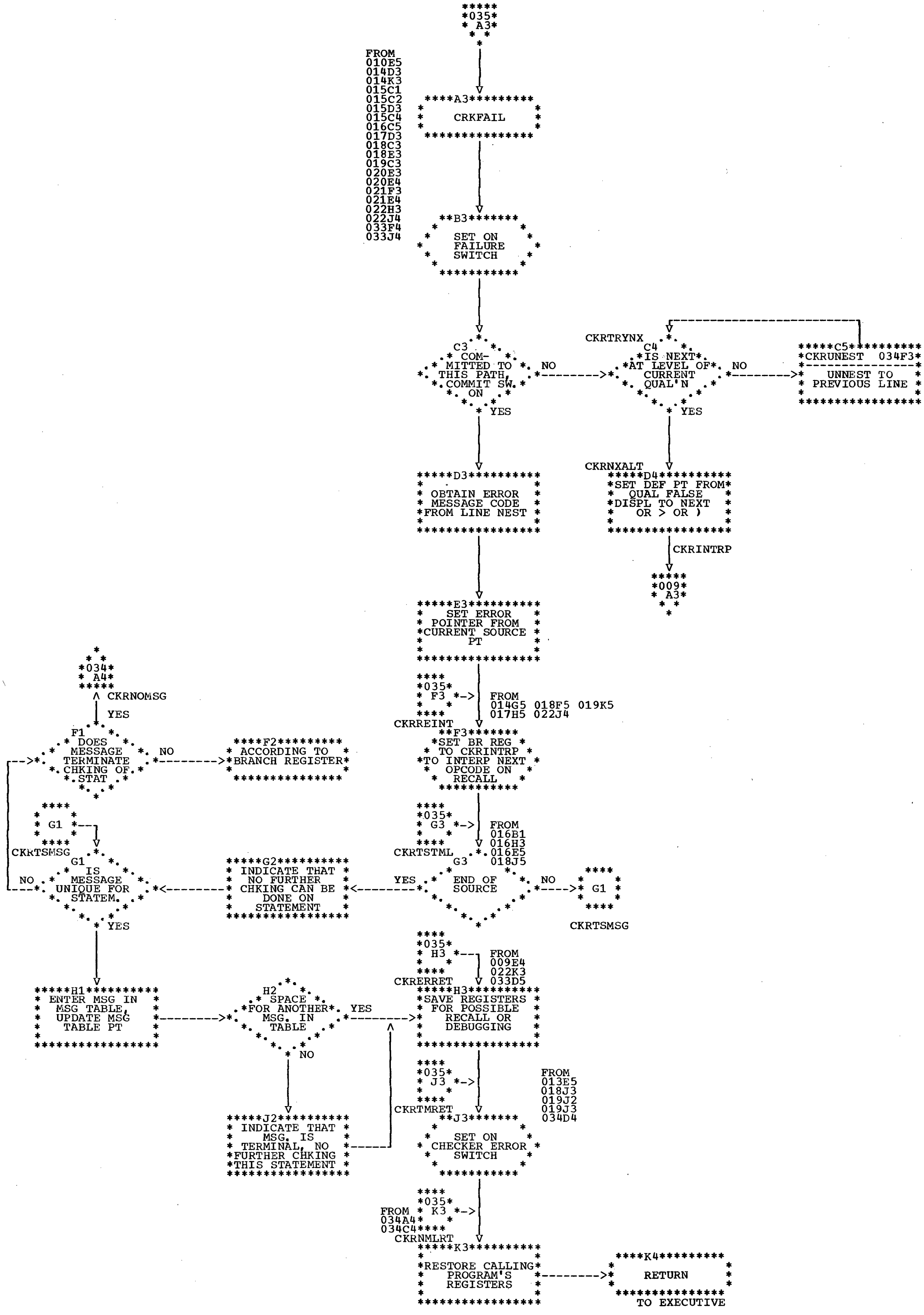

035 A3
FROM 010E5 014D3 014K3 015C1 015C2 015D3 015C4 016C5 017D3 018C3 018E3 019C3 020E3 020E4 021F3 021E4 022H3 022J4 033F4 033J4
A3 CRKFAIL
B3 SET ON FAILURE SWITCH
C3 COMMITTED TO THIS PATH, COMMIT SW. ON
NO
YES
CKRTRYNX
C4 IS NEXT AT LEVEL OF CURRENT QUAL'N
NO
YES
C5 CKRUNEST 034F3 UNNEST TO PREVIOUS LINE
D3 OBTAIN ERROR MESSAGE CODE FROM LINE NEST
CKRNXALT
D4 SET DEF PT FROM QUAL FALSE DISPL TO NEXT OR &gt; OR)
CKRINTRP
009 A3
E3 SET ERROR POINTER FROM CURRENT SOURCE PT
034 A4
CKRNOMSG
YES
035 F3
FROM 014G5 018F5 019K5 017H5 022J4
CKRREINT
F1 DOES MESSAGE TERMINATE CHKING OF STAT
NO
F2 ACCORDING TO BRANCH REGISTER
F3 SET BR REG TO CKRINTRP TO INTERP NEXT OPCODE ON RECALL
G1
CKRTSMSG
G1 IS MESSAGE UNIQUE FOR STATEM.
NO
YES
G2 INDICATE THAT NO FURTHER CHKING CAN BE DONE ON STATEMENT
035 G3
FROM 016B1 016H3 016E5 018J5
CKRTSTML
G3 END OF SOURCE
YES
NO
G1
CKRTSMSG
035 H3
FROM 009E4 022K3 033D5
CKRERRET
H1 ENTER MSG IN MSG TABLE, UPDATE MSG TABLE PT
H2 SPACE FOR ANOTHER MSG. IN TABLE
YES
NO
H3 SAVE REGISTERS FOR POSSIBLE RECALL OR DEBUGGING
035 J3
FROM 013E5 018J3 019J2 019J3 034D4
CKRTMRET
J2 INDICATE THAT MSG. IS TERMINAL, NO FURTHER CHKING THIS STATEMENT
J3 SET ON CHECKER ERROR SWITCH
035 K3
FROM 034A4 034C4
CKRNMLRT
K3 RESTORE CALLING PROGRAM'S REGISTERS
K4 RETURN
TO EXECUTIVE

Chart 036. IPDSNCKR (CKRGTNB1)

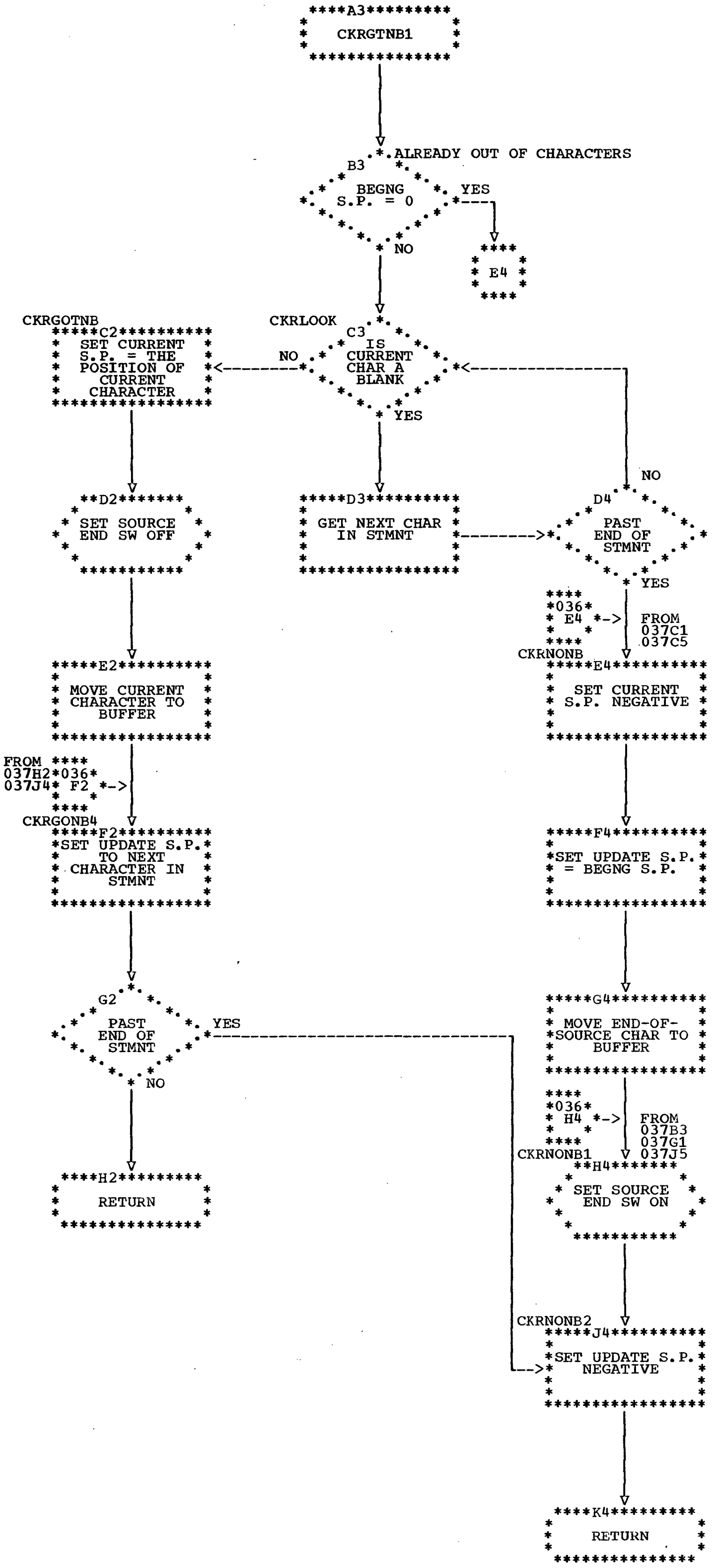


Chart 037. CKRSKANY, CKRGTANY, CKRGTNBS

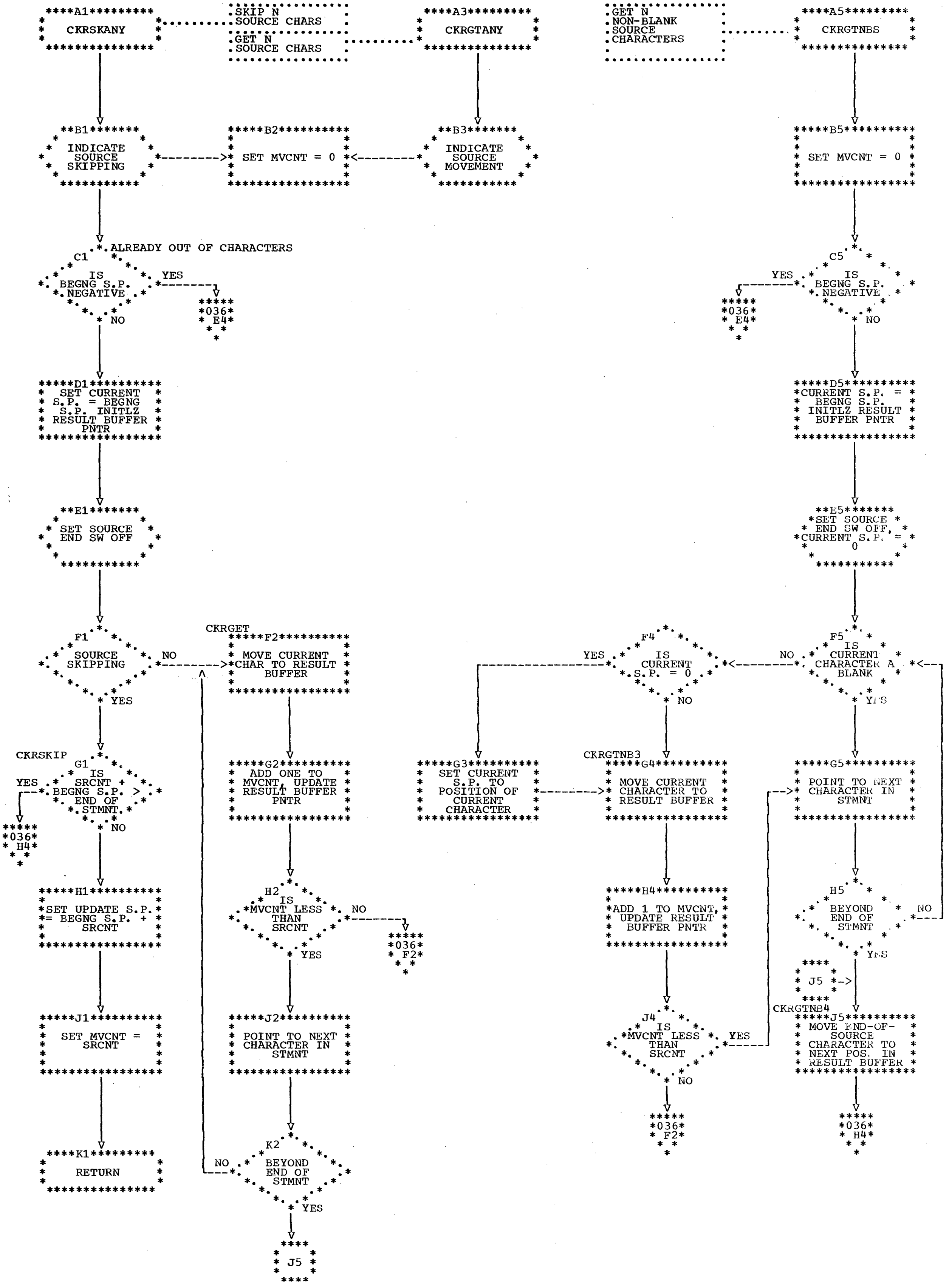

A1 CKRSKANY
SKIP N SOURCE CHARS
GET N SOURCE CHARS
A3 CKRGTANY
GET N NON-BLANK SOURCE CHARACTERS
A5 CKRGTNBS
B1 INDICATE SOURCE SKIPPING
B2 SET MVCNT = 0
B3 INDICATE SOURCE MOVEMENT
B5 SET MVCNT = 0
C1 ALREADY OUT OF CHARACTERS
IS BEGNG S.P. NEGATIVE
YES
NO
036 E4
C5 IS BEGNG S.P. NEGATIVE
YES
NO
036 E4
D1 SET CURRENT S.P. = BEGNG S.P. INITLZ RESULT BUFFER PNTR
D5 CURRENT S.P. = BEGNG S.P. INITLZ RESULT BUFFER PNTR
E1 SET SOURCE END SW OFF
E5 SET SOURCE END SW OFF, CURRENT S.P. = 0
F1 SOURCE SKIPPING
NO
YES
CKRGET
F2 MOVE CURRENT CHAR TO RESULT BUFFER
F4 IS CURRENT S.P. = 0
YES
NO
F5 IS CURRENT CHARACTER A BLANK
NO
YES
CKRSKIP
G1 IS SRCNT + BEGNG S.P. > END OF STMNT.
YES
NO
036 H4
G2 ADD ONE TO MVCNT, UPDATE RESULT BUFFER PNTR
G3 SET CURRENT S.P. TO POSITION OF CURRENT CHARACTER
CKRGTNB3
G4 MOVE CURRENT CHARACTER TO RESULT BUFFER
G5 POINT TO NEXT CHARACTER IN STMNT
H1 SET UPDATE S.P. = BEGNG S.P. + SRCNT
H2 IS MVCNT LESS THAN SRCNT
NO
YES
036 F2
H4 ADD 1 TO MVCNT, UPDATE RESULT BUFFER PNTR
H5 BEYOND END OF STMNT
NO
YES
J5
CKRGTNB4
J1 SET MVCNT = SRCNT
J2 POINT TO NEXT CHARACTER IN STMNT
J4 IS MVCNT LESS THAN SRCNT
YES
NO
036 F2
J5 MOVE END-OF-SOURCE CHARACTER TO NEXT POS. IN RESULT BUFFER
036 H4
K1 RETURN
K2 BEYOND END OF STMNT
NO
YES
J5

Chart 038. CKRSERCH

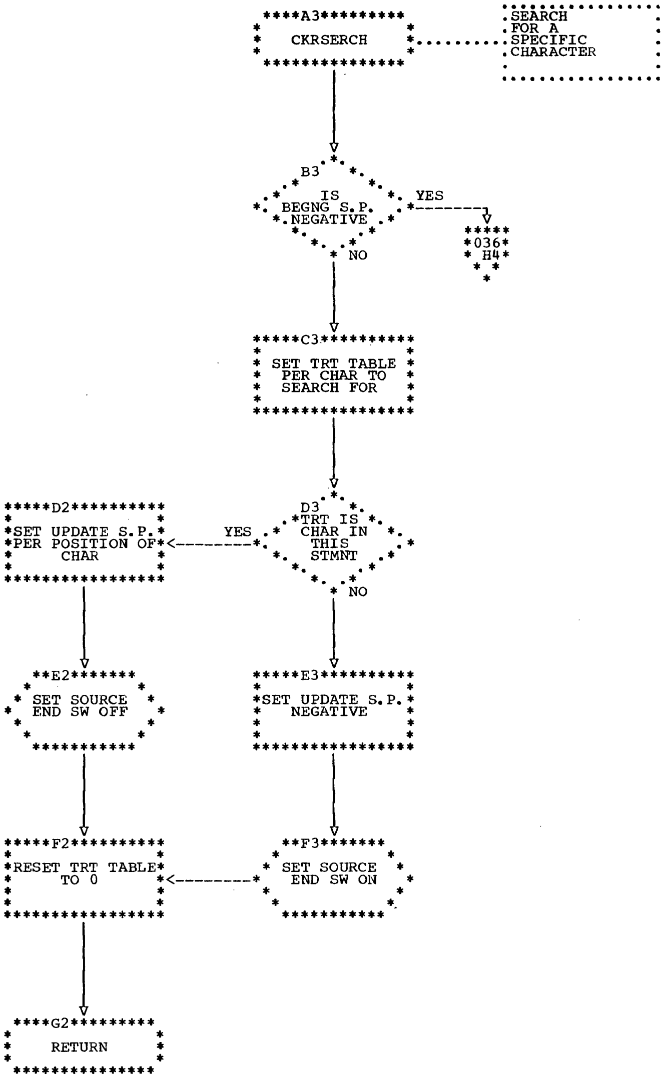
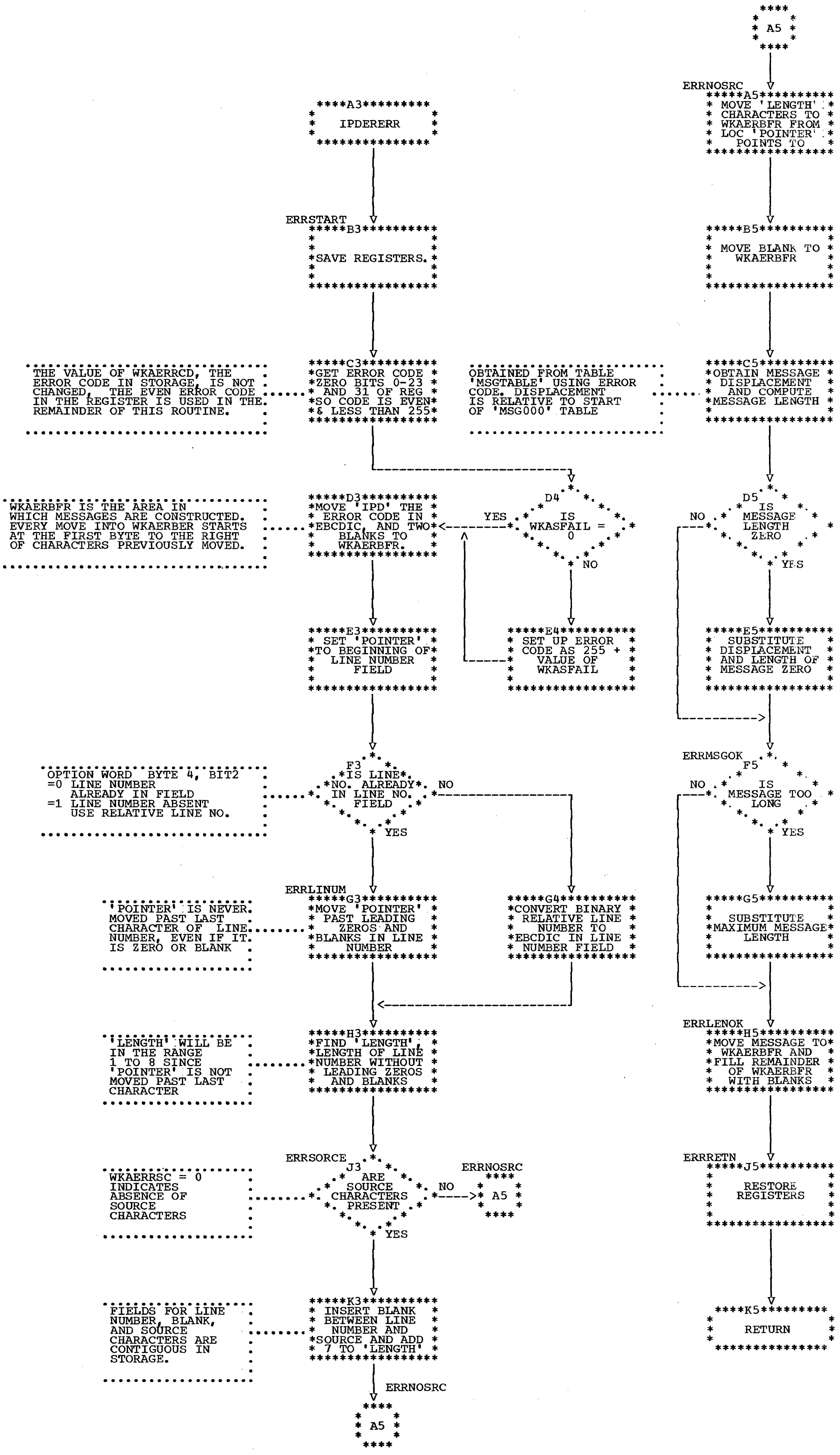


Chart 039. IPDERERR (Error Code Processor)



A3 IPDERERR
ERRSTART B3 SAVE REGISTERS.
C3 GET ERROR CODE ZERO BITS 0-23 AND 31 OF REG SO CODE IS EVEN & LESS THAN 255
THE VALUE OF WKAERRCD, THE ERROR CODE IN STORAGE, IS NOT CHANGED. THE EVEN ERROR CODE IN THE REGISTER IS USED IN THE REMAINDER OF THIS ROUTINE.
D3 MOVE 'IPD' THE ERROR CODE IN EBCDIC, AND TWO BLANKS TO WKAERBFR.
WKAERBFR IS THE AREA IN WHICH MESSAGES ARE CONSTRUCTED. EVERY MOVE INTO WKAERBER STARTS AT THE FIRST BYTE TO THE RIGHT OF CHARACTERS PREVIOUSLY MOVED.
D4 IS WKASFAIL = 0
YES
NO
E3 SET 'POINTER' TO BEGINNING OF LINE NUMBER FIELD
E4 SET UP ERROR CODE AS 255 + VALUE OF WKASFAIL
F3 IS LINE NO. ALREADY IN LINE NO. FIELD
NO
YES
OPTION WORD BYTE 4, BIT2 =0 LINE NUMBER ALREADY IN FIELD =1 LINE NUMBER ABSENT USE RELATIVE LINE NO.
ERRLINUM G3 MOVE 'POINTER' PAST LEADING ZEROS AND BLANKS IN LINE NUMBER
'POINTER' IS NEVER. MOVED PAST LAST CHARACTER OF LINE NUMBER, EVEN IF IT IS ZERO OR BLANK
G4 CONVERT BINARY RELATIVE LINE NUMBER TO EBCDIC IN LINE NUMBER FIELD
H3 FIND 'LENGTH', LENGTH OF LINE NUMBER WITHOUT LEADING ZEROS AND BLANKS
'LENGTH' WILL BE IN THE RANGE 1 TO 8 SINCE 'POINTER' IS NOT MOVED PAST LAST CHARACTER
ERRSORCE J3 ARE SOURCE CHARACTERS PRESENT
NO
ERRNOSRC A5
YES
WKAERRSC = 0 INDICATES ABSENCE OF SOURCE CHARACTERS
K3 INSERT BLANK BETWEEN LINE NUMBER AND SOURCE AND ADD 7 TO 'LENGTH'
FIELDS FOR LINE NUMBER, BLANK, AND SOURCE CHARACTERS ARE CONTIGUOUS IN STORAGE.
ERRNOSRC
A5
A5
ERRNOSRC A5 MOVE 'LENGTH' CHARACTERS TO WKAERBFR FROM LOC 'POINTER' POINTS TO
B5 MOVE BLANK TO WKAERBFR
C5 OBTAIN MESSAGE DISPLACEMENT AND COMPUTE MESSAGE LENGTH
OBTAINED FROM TABLE 'MSGTABLE' USING ERROR CODE. DISPLACEMENT IS RELATIVE TO START OF 'MSG000' TABLE
D5 IS MESSAGE LENGTH ZERO
NO
YES
E5 SUBSTITUTE DISPLACEMENT AND LENGTH OF MESSAGE ZERO
ERRMSGOK F5 IS MESSAGE TOO LONG
NO
YES
G5 SUBSTITUTE MAXIMUM MESSAGE LENGTH
ERRLENOK H5 MOVE MESSAGE TO WKAERBFR AND FILL REMAINDER OF WKAERBFR WITH BLANKS
ERRRETN J5 RESTORE REGISTERS
K5 RETURN

The microfiche directory is designed to help you find named areas of code in the program listing, which is contained on microfiche cards at your installation. Microfiche cards are filed in alphameric order by object module name. If you wish to locate a control section, subroutine, table, or work area on microfiche, find the name in column one and note the associated object module name. You can then find the item on microfiche, via the object module name; for example, the subroutine CKRGTANY is on card IPDSN. In the case where a work area is referenced by two object modules, the names of both modules appear in the module name column. The other columns provide a description of the item, its flowchart identification (if applicable), and a synopsis of its function (or its contents, if a table).

Table 4 below contains a module-CSECT cross-reference table.

Table 4. Module-CSECT Cross-Reference Table

Load Module Name	Object Module Name	CSECT Name
IPDAGH	IPDAGH	IPDAGH
IPDSNEXC	IPDER	IPDERERR
	IPDSN	IPDSNCKR, IPDSNEXC
IPDTEE	IPDTEE	IPDTEE

Name	Description	Object Module Name (Microfiche Name)	CSECT/ DSECT Name	Chart ID	Synopsis
CKRACNDX	table	IPDSN	IPDSNCKR	--	Displacements to action code routines
CKRAMTBL	table	IPDSN	IPDSNCKR	--	Translate and test table for A-Z and 0-9
CKRGTANY	subroutine name	IPDSN	IPDSNCKR	037	Get next n source characters, both blank and nonblank
CKRGTNBS	subroutine name	IPDSN	IPDSNCKR	037	Get next n nonblank source characters
CKRGTNB1	subroutine name	IPDSN	IPDSNCKR	036	Get next nonblank source character
CKROPNDX	table	IPDSN	IPDSNCKR	--	Displacements to syntactic operator routines
CKRSERCH	subroutine name	IPDSN	IPDSNCKR	039	Search source for a specific character
CKRSKANY	subroutine name	IPDSN	IPDSNCKR	037	Skip next n source characters, both blank and nonblank
EXCADRDF	table	IPDSN	IPDSNEXC	--	Displacements in EXCSYNXS to syntax table address associated with each FORTRAN level.
EXCLODTB	table	IPDSN	IPDSNEXC	--	Syntax definition table names. Used for loading and deleting purposes.
EXCSYNXS	table	IPDSN	IPDSNWKA	--	Locations of syntax definition tables in core storage.

Name	Description	Object Module Name (Microfiche Name)	CSECT/ DSECT Name	Chart ID	Synopsis
IPDCKWRK	work area	IPDSN	IPDSNWKA	--	Miscellaneous checker work areas
IPDERERR	CSECT	IPDER	IPDERERR	040	Error code processor, constructs error messages
IPDERWKA	work area	IPDSN and IPDER	IPDSNWKA and IPDERWKA	--	Communications area between executive, checker, and error code processor
IPDSNCKR	CSECT	IPDSN	IPDSNCKR	006-036	Checker checks source statement against a syntax definition table
IPDSNEXC	CSECT	IPDSN	IPDSNEXC	001-005	Executive interfaces with environmental system; calls checker to check source statements; calls error code processor to construct error messages
IPDSNWKA	DSECT	IPDSN	IPDSNWKA	--	Work area used by executive, checker and error code processor
MSGTABLE	table	IPDER	IPDERERR	--	Table of displacements of error message texts relative to MSG000.
MSG000	table	IPDER	IPDERERR	--	Table containing texts of all error messages
WKACHRST	work area	IPDSN	IPDSNWKA	--	Source statement character string
WKACKPRM	table	IPDSN	IPDSNWKA	--	Parameter list set up by the executive when it calls the checker
WKAGTCHR	table	IPDSN	IPDSNWKA	--	Supplied and returned parameters of get-character routines
WKALNEST	table	IPDSN	IPDSNWKA	--	Current syntactic line nest
WKANLIST	table	IPDSN	IPDSNWKA	--	Push down stack for syntactic line nests
WKAQLIST	table	IPDSN	IPDSNWKA	--	Push down stack for qualification information
WKAQUALF	table	IPDSN	IPDSNWKA	--	Current qualification information
WKASERTB	table	IPDSN	IPDSNWKA	--	Translate and test table for CKRSERCH routine
WKATINU	table	IPDSN	IPDSNWKA	--	Displacements within WKACHRST of lines of current source statement and associated line numbers.

This section provides detailed layouts of internal tables and work areas used during syntax checking.

Table 5 indicates the control sections in which the tables and work areas are referenced. The format and content of each of the areas listed follows Table 5.

Table 5. Table and Work Area Usage

Table/Work Area	Initialized, Used, and/or Modified by
CKRACNDX*	IPDSNCKR
CKRAMTBL*	IPDSNCKR
CKROPNDX *	IPDSNCKR
EXCADRDF *	IPDSNEXC
EXCLODTB *	IPDSNEXC
IPDSNWKA	IPDSNEXC, IPDSNCKR, IPDERERR
MSGTABLE*	IPDERERR
MSG000*	IPDERERR

*Table is created at assembly time and must not be modified during execution.

CKRACNDX: Action Routine Branch Table

```
0       2         4        6
┌───────┬─────────┬────────┬─────────   ─────────────────────────┬──────────┐
│       │         │        │          ...                         │          │
└───────┴─────────┴────────┴─────────   ─────────────────────────┴──────────┘
```

Each halfword contains a displacement from CKRACTN to an action code routine. There is one entry for each action code. The displacement of the entry from the beginning of the table (CKRACNDX) is equal to the hexadecimal action code.

CKRAMTBL: Translate and Test Table for A-Z and 0-9

```
0        1        2        C1     C2     C3                          FF
┌────────┬────────   ─────┬──────┬──────┬─────┬───   ────────┬─────┬────────┐
│ NU*    │ NU          NU │  00  │  00  │     │    ...        │     │  01    │
└────────┴────────   ─────┴──────┴──────┴─────┴───   ────────┴─────┴────────┘
*NU = not used
```

Only the last 63 bytes of this 256-byte table are used; they are used by TRT instructions to determine whether a character whose hex equivalent is at least C1 is alphameric. The table bytes corresponding to the letters A through Z and digits 0 through 9 are set to X'00'. The remaining bytes in the X'C1' through X'FF' range are nonzero.

CKROPNDX: Operator Routine Branch Table

```
0       2         4        6
┌───────┬─────────┬────────┬─────────    ─────────────────────┬──────────┐
│       │         │        │         ...                      │          │
└───────┴─────────┴────────┴─────────    ─────────────────────┴──────────┘
```

Each halfword contains a displacement from CKRINTRP to an operator routine. There is one entry for each syntactic operator. The displacement of the entry from the beginning of the table (CKROPNDX) is equal to the hexadecimal code for the operator.

EXCADRDF: Definition Address Displacement Table

```
0             1             2             3             4
┌─────────────┬─────────────┬─────────────┬─────────────┬─────────────┐
│             │             │             │             │             │
└─────────────┴─────────────┴─────────────┴─────────────┴─────────────┘
```

Each byte contains a displacement (X'00' or X'04') from EXCSYNXS to the address of the definition table to be used in scanning each FORTRAN level. The order of the entries is FORTRAN H, FORTRAN E, FORTRAN G, Code and Go FORTRAN, and FORTRAN G1.

EXCLODTB: Definition Load Table

```
┌───────────┬────────────┬────────────┬────────────┬────────────┬──────────┐
│   name    │    code    │    name    │    code    │    name    │   code   │
└┬──────────┴┬───────────┴────────────┴────────────┴────────────┴──────────┘
 │           │
 │           └--code (1 byte) for each syntax definition table.  Code
 │              values are the same as those specified in bits 6-7 of
 │              byte 1 of the options word.  The options word is passed
 │              by the environmental system to the checker.
 │
 └--name (8 bytes) of each definition table.
```

There are two entries in the table: IPDTEE, definition for FORTRAN E, and IPDAGH, definition for FORTRAN G, G1, H and Code and Go.

MSGTABLE: Table of Message Text Displacements

```
0       2         4        6                                  1
┌───────┬─────────┬────────┬─────────    ─────────────────────┬──────────┐
│       │         │        │         ...                      │          │
└───────┴─────────┴────────┴─────────    ─────────────────────┴──────────┘
```

Each halfword contains a displacement from MSG000, the start of message zero, to MSG<u>n</u>, the start of the text of error message <u>n</u>. The error code <u>n</u> (even value of an even-odd pair associated with a message, e.g., <u>0</u>, 1; <u>12</u>,13; etc.) is used as an index to MSGTABLE.

MSG000: Table of Texts of Error Messages

```
┌─────────────────┬─────────────────┬─────    ─────────┬────────────────┐
│                 │                 │     ...          │                │
└─────────────────┴─────────────────┴─────    ─────────┴────────────────┘
```

Entries take the form of a labeled (MSG<u>n</u>) character string defining the text of an error message. Entries are variable-length and are ordered by (even-numbered) error code <u>n</u>.

Table 6. IPDSNWKA: Syntax Checker Work Area (Part 1 of 2)

Field Name	Bytes	Relative Address		Field Description
		Dec	Hex	
EXCSVRGS	72	0	0	18-word register save area used by the executive.
WKACKPRM	20	72	48	5-word area containing parameter list passed to checker by executive.
EXCSYNXS	8	92	5C	Table containing the address (zero) of each definition table loaded (not loaded).
EXCCRCRD	12	100	64	Current line information.
Bytes 1-4				Location of current line.
Bytes 5-8				Location of buffer containing current line.
Bytes 9-10				Relative position of current line in buffer.
Bytes 11-12				Relative position of current line in buffer chains received since last initial entry or intermediate entry after return code 0 or 4 (i.e., the relative line number).
EXCNXCRD	12	112	70	Next line information. Format of information is the same as that for EXCCRCD.
EXCFSCRD	12	124	7C	First line of current statement information. Format of information is the same as that for EXCCRCRD.
EXCSVCRD	12	136	88	First line of next statement information. Format of information is the same as that for EXCCRCD.
WKALEVEL	1	148	94	Level of FORTRAN for scan (set from options word). Code Meaning 00 FORTRAN H 01 FORTRAN E 02 FORTRAN G 03 FORTRAN Code and Go 04 FORTRAN G1
WKACNCOL	1	149	95	For standard-form source, contents of the continuation column from the last line of a statement. For free-form source non-blank for a statement of more than one line, otherwise blank.
IPDERWKA	99	152	98	Error code processor communication area.
WKACERSW	1	251	FB	Checker error switch: Code Meaning 00 no error 01 statement error
WKASNOSW	1	252	FC	Statement number switch: Code Meaning 00 statement label field is blank. 01 a statement label (nonblank) is present in statement label field.
EXCFSCOM	1	253	FD	Comment line switch (standard-form source only): Code Meaning 00 No comment line since beginning of statement. FF Comment line encountered.
EXCOMSG	1	254	FE	Comment message switch (standard-form source only): Code Meaning 00 Intervening comment card; message has not been sent. FF Message has been sent

Table 6. IPDSNWKA: Syntax Checker Work Area (Part 2 of 2)

		Relative Address		
Field Name	Bytes	Dec	Hex	Field Description
EXCFSCON	1	255	FF	Continuation switcon (standard-form source only): Code — Meaning 00 — No continuation line has preceded first statement in buffer chain received on initial entry or intermediate entry after return code 0 or 4. FF — Continuation line encountered first.
EXCEXSLN	1	256	100	Extraneous lines switch: Code — Meaning 00 — No more than twenty lines have been found for statement. FF — Have encountered extraneous line of statement.
EXCSLERR	1	257	101	Statement label errors switch: Code — Meaning 80 — Label contains digit 1-9. 40 — Label contains 0. 20 — Label contains extraneous character. 10 — Extraneous character found. 08 — Message for statement too long already sent. 04 — Statement field missing (free-form source only). 02 — Message for invalid statement label issued.
IPDEXCWK	23	258	102	Miscellaneous executive (IPDSNEXC) work areas.
WKATINU	202	282	11A	Table of displacement and line number information for lines of statement.
WKACHRST	1325	484	1E4	Character string containing statement label and text of FORTRAN statement.
WKASERTB	256	1809	711	Translate and test table used by CKRSERCH (a get-character routine).
WKAGTCHR	34	2068	814	Information used and set by the get-character routines.
IPDCKWRK	173	2104	838	Miscellaneous checker (IPDSNCKR) work areas.
WKALNEST	6	2278	8E6	Current syntactic line nest information.
WKANLIST and WKAQLIST	Variable, but not less than 1560	variable	variable	This area is shared by the line nest list and the qualification information list. WKANLIST starts adjacent to WKALNEST and WKAQLIST starts adjacent to WKAQUALF and the two lists grow toward each other.
WKAQUALF	12	variable	variable	Qualification information.
WKADNAME	8	4088	FF8	IPDSNWKA - the name of the area.

IPDSNWKA, whose layout is described above, contains several small tables or work areas which require a more detailed explanation. These areas are listed in Table 7. A description of the content and format of each table or work area listed follows Table 7.

Table 7. Work Areas within IPDSNWKA

Work Area	Initialized, Used, and/or Modified by
EXCSYNXS	IPDSNEXC
IPDCKWRK	IPDSNCKR
IPDERWKA	IPDSNEXC, IPDSNCKR, IPDERERR
IPDEXCWK	IPDSNEXC
WKACKPRM	IPDSNEXC, IPDSNCKR
WKAGTCHR	IPDSNCKR
WKALNEST	IPDSNCKR
WKANLIST	IPDSNCKR
WKAQLIST	IPDSNCKR
WKAQUALF	IPDSNCKR
WKASERTB	IPDSNEXC, IPDSNCKR
WKATINU	IPDSNEXC, IPDSNCKR

EXCSYNXS: Definition Address Table

```
┌────────────────────────┬────────────────────────┬───────────────────────┐
│                        │                        │                       │
└─┬──────────────────────┴────────────────────────┴───────────────────────┘
  │
  └--Address (4 bytes) - For each FORTRAN definition table there is the address at which
     the table is located.  The order of the entries is the same as that of the EXCLODTB
     table.  If the definition table has not been loaded, the address is zero.
```

Table 8. IPDCKWRK: Miscellaneous Checker (IPDSNCKR) Work Areas (Part 1 of 2)

		Relative Address		
Field Name	Bytes	Dec	Hex	Field Description
WKASVR13	4	2104	838	Executive's register 13 saved by checker.
WKABEGSC	4	2108	83C	Pointer to first nonblank character in source statement.
WKALDZCT	4	2112	840	K operator leading zeros count.
WKADGTCT	4	2116	844	K operator digit count.
WKAZROCT	4	2120	848	K operator zero count.
WKATENPW	4	2124	84C	K operator power of ten. Used to test magnitude of a number.
WKAADNLS	4	2128	850	Address of the line nest list (WKANLIST).
WKATPQLS	4	2132	854	Address of actual top of qualification list (WKAQLIST unless changed by a statement commit).
WKAVALUE	4	2136	858	Numeric value (fixed point format) of consecutive source digits.
WKAWIDTH	4	2140	85C	Width, w, of format codes, e.g., wH or Iw.
WKACKRGS	64	2144	860	Registers 0-15 saved by the checker on return (to the executive); restored by the checker when recalled to continue checking the same statement.
WKASAVNQ	8	2208	8A0	Temporary save area for nest list register and qualification list register.
WKACSVSC	4	2216	8A8	Temporary save area for source pointer register, REGSRCPT.
WKASVRTN	4	2220	8AC	Subroutine return register save area.
WKAGTSV	16	2224	8B0	Register save area for get-character routines.
WKAMSGAD	4	2240	8C0	Address of next message information word in table WKAMSGTB.
WKAMSGTB	20			Table of error message information for current statement. Contains 5 full-word entries of the form:
Byte 0				Error code (even number value).
Bytes 1-3				Failing character address.
WKATEMPH	2	2264	8D8	Halfword aligned temporary storage.
WKACNTDG	2	2266	8DA	Number of continuous digits evaluated (by CKREVALU routine).
WKACKRSW	1	2268	8DC	IPDSNCKR switch byte containing bit switches:
WKAFALSW				False switch; if bit 0 (CKRFALSW) = 1, false condition exists in an operator or action code routine.
WKAGLCMT				Statement global commit switch : if bit 1 (CKRGLCMT) = 1, ":" was encountered in the definition for the current statement.
WKAKSWCH	1	2269	8DD	K operator switches: bit 7 (CKRKTYPI): 1/0 Type is integer/real bit 6 (CKRKLEND): 1/0 Length is D/E bit 5 (CKRKVALU): 1/0 Value is nonzero/zero bits 5,6,7 (CKRKFAIL): All ones, source is not numeric.
WKASVKSW	1	2270	8DE	K operator switches saved for complex test. These switch bits describe the left (real) portion of the complex number. Same form as WKAKSWCH.
WKANONZS	1	2271	8DF	K operator byte switch. It is set zero if no nonzero digits were encountered; otherwise it is set nonzero.
WKAEXPSN	1	2272	8E0	K operator sign of exponent.

Table 8. IPDCKWRK: Miscellaneous Checker (IPDSNCKR) Work Areas (Part 2 of 2)

Field Name	Bytes	Relative Address		Field Description
		Dec	Hex	
WKASWTCH	1	2273	8E1	Temporary processing switches, set and tested by various operator and action code routines.
WKAIMPL1	1	2274	8E2	First character of an IMPLICIT range.
WKASVICT	1	2275	8E3	Iteration count saved from qualification information for possible action routine use.
WKATBLOP	1	2276	8E4	Hexadecimal operator code for +Table or -Table operator.

Table 9. IPDERWKA: Error Code Processor Communications Area

Field Name	Bytes	Relative Address		Field Description
		Dec	Hex	
WKAEROPT	4	152	98	Location of fourth byte of options word.
WKAERBFR	72	156	9C	Error message buffer. This must be aligned to the middle of a doubleword.
WKAERRSC	4	288	E4	Address of WKAERCHR or zero if WKAERCHR is not to be used.
WKAERPOS	2	232	E8	Relative line number in binary format. Not used if source data set is line numbered.
WKAERRCD	1	234	EA	Error message code.
WKASFAIL	1	235	EB	Not operational error byte: Code Meaning 00 No internal errors 01-0F IPDSNEXC not operational 10-1F IPDSNCKR not operational 20-2F IPDERERR not operational
WKAERNUM	8	236	EC	Data set line number, right adjusted in EBCDIC. Used if source data set is line-numbered.
WKAERPAD	1	244	F4	Space for insertion of a blank.
WKAERCHR	6	245	F5	Source statement characters in error.

Note: IPDER depends for its operation on the sizes, order, and contiguity of WKAERNUM, WKAERPAD, and WKAERCHR.

Table 10. IPDEXCWK: Miscellaneous Executive (IPDSNEXC) Work Areas

Field Name	Bytes	Relative Address		Field Description
		Dec	Hex	
EXCSVLNN	8	258	102	Temporary storage for data set line number of line pointed to by EXCSVCRD.
WKATECHR	6	266	10A	Temporary storage for six-character error string.
EXCSVLBL	4	272	110	Address of last digit of statement label (free-form only).
WKATERSC	4	276	114	Temporary storage for address of error string.
EXCFCHAR	1	280	118	Temporary storage for folding lower case alphabetics to upper case.

Table 11. WKACKPRM: Checker Parameter List

0	(0)	Beginning source pointer	WKABEGST
4	(4)	End source pointer	WKAENDST
8	(8)	Address of definition table	WKADEEF
12	(C)	Address of work area	WKAWADDR
16	(10)	Address of options word	WKAOPTPT

		Relative Address		
Field Name	Bytes	Dec	Hex	Field Description
WKABEGST	4	72	48	Initially supplied beginning source pointer. This field points to the start of the statement field of a source statement in WKACHRST.
WKAENDST	4	76	46	Initially supplied end source pointer. This field points to the last character of the statement field of a source statement in WKACHRST.
WKADEEF	4	80	50	Address of the definition table to be used by the checker when scanning.
WKAWADDR	4	84	54	Address of the IPDSNWKA work area.
WKAOPTPT	4	88	58	Address of the options word passed by the environmental system to the syntax checker.

Table 12. WKAGTCHR: Areas for Communication with the Get Character Routines

		Relative Address		
Field Name	Bytes	Dec	Hex	Field Description
WKASRCCR source pointer format	4	2068	814	Current source pointer. This field points to the first character obtained by a get character routine.
Byte 0				Source end indicator Code Meaning 00 End of statement not reached 80 End of source statement reached
Bytes 1-3				Address of the source character in the character string.
WKASRCUP	4	2072	818	Update source pointer. CKRGTNB1, CKRGTNBS, CKRGTANY and CKRSKANY set this field to the next character beyond the last source character found or set it negative if there iis no next character. It is in source pointer format. (See WKASRCCR)
WKASCHRS	20	2076	81C	Result buffer to which source characters are moved. A special character (that is non alphameric) is moved to the buffer beyond the last character obtained if a CKRGTNB1, CKRGTNBS, or CKRGTANY request cannot be completely satisfied.
WKASRCNT	2	2096	830	The number of characters to be moved to WKASCHRS when CKRGTNBS or CKRGTANY is called, or the number to be skipped when CKRSKANY is called.
WKAMVCNT	2	2098	832	The number of source characters actually moved by CKRGTNBS or CKRGTANY, or skipped by CKRSKANY.
WKASRCHX	1	2100	834	The character to be searched for by CKRSERCH.
WKASNDSW	1	2101	835	Source end switch. Code Meaning 00 not end of source 01 end of source encountered before a get-character request was fulfilled.

Tables 13-16. Line Nest List and Qualification Information List Pushdown Stacks

The nest and qualification lists are pushdown stacks that grow toward each other in order to optimize storage usage. The nest list grows from lower to higher-numbered storage locations, adding a 6-byte entry, or "nesting", when the table scan begins, and thereafter, whenever the table scan encounters an active line reference. A table reference, + or -, does not cause nesting. On reaching the end of an active definition line, the nest most recently added is removed from the nest list ("unnested"), thereby adding six bytes to the space into which the nest list and the qualification list can expand.

The qualification list grows from higher to lower-numbered locations, adding a 12-byte entry, whenever a < or (is encountered to signal the beginning of a list of alternatives or an option in the definition. The qualification entry most recently added is removed from the list when its closing) or > is encountered.

All the nest entries and some of the qualification entries can be deleted when a statement commit (:) is encountered in the definition. The statement commit makes the current line (described in WKALNEST) the top line. Therefore, all entries in the nest list are deleted (by resetting a pointer) since they describe earlier lines that will not be "unnested to" again in checking the current statement. The statement commit also eliminates the need for saving any of the qualification entries associated with earlier lines. However, the remaining qualification entries are not shifted down to create growth space until space is actually needed for an additional nest or qualification entry.

Table 13. WKALNEST: Current Line Nest

0(0) WKANDFPT Displacement pointer to current definition line		2(2) WKANDFBK Displacement pointer back to earlier definition	3 halfwords
4(4) WKANLVLN Level of nesting	5(5) WKANFMSG Error message code in effect		

Field Name	Bytes	Relative Address		Field Description
		Dec	Hex	
WKANDFPT	2	2278	8E6	Displacement to current definition line from beginning of definition table.
WKANDFBK	2	2280	8E8	Displacement immediately beyond the reference to the current line on the earlier definition line.
WKANLVLN	1	2282	8EA	Level of nesting of the current line, 1 for top line of definition.
WKANFMSG	1	2283	8EB	Error message code in effect.

Table 14. WKANLIST: Line Nest List Pushdown Stack

```
                                  ┌────────────────────────────┐      ┐
                                  │6(6)                        │      │
                                  │                            │      │
                                  │                            │      │
┌─────────────────────────────────┘                            │      │
│8(8)                                                          │      │
│Earliest line nest                                            │   6 byte
│(Level 0 before statement commit)                             │ entries on
│                                                              │  halfword
│                                                              │ boundaries
│                                                              │      │
├──────────────────────────────────────────────────────────────┤      V
│12(C)                                                         │
│Next to earliest line nest                                    │
│                                                              │
│                                 ┌────────────────────────────┤
│                                 │18(10)                      │
│                                 │                            │
│                                 │                            │
├─────────────────────────────────┘                            │
│20(14)                     .                                  │
│                           .                                  │
│                           .                                  │
│                                                              │


│                                                              │
├──────────────────────────────────────────────────────────────┤
│Next to Current Line Nest                                     │
│(Immediately prior to current line nest)                      │
│                                 ┌────────────────────────────┘
│Last in, first out               │
└─────────────────────────────────┘
```

Table 15. WKAQLIST: Qualification Information List Pushdown Stack

```
┌──────────────────────────────────────────────────────────────┐
│Qualification information for next to current, that is        │
│next to innermost, < > or ( ) encountered in definition       │
│                                                              │
│Last in, first out                                            │
├──────────────────────────────────────────────────────────────┤
│                                   .                          │
│                                   .                          │
│                                   .                          │
│                                                              │


│                                                              │
│                                                              │
│                                                              │
├──────────────────────────────────────────────────────────────┤
│WKAQLIST -24 (WKAQLIST -18)                                   │
│Qualification information for outermost < > or ( )           │
├──────────────────────────────────────────────────────────────┤      ┐
│WKAQLIST -12 (WKAQLIST -C)                                    │      │
│Qualification information for requirement prior to first <   │   12-byte
│or ( (always committed)                                       │ entries on
└──────────────────────────────────────────────────────────────┘  fullword
 WKAQLIST                                                         boundaries
                                                                      │
                                                                      V
```

Table 16. WKAQUALF: Qualification Information

<table>
<tr><td colspan="4">0(0) WKAQSCPT
Pointer to source for retry</td></tr>
<tr><td colspan="2">4(4) WKAQDFBK
Displacement pointer to definition for iteration</td><td>6(6) WKAQNLVL
Level of nesting of corresponding line</td><td>7(7) WKAQFALS
Displacement to next), |, or ></td></tr>
<tr><td>8(8)
WKAQTRUE
Displ. to right brace</td><td>9(9)
WKAQICNT
Iteration count</td><td>10(A)
WKAQSWCH
Commit switch</td><td>11(B)
Not used</td></tr>
</table>

3 fullwords

Field Name	Bytes	Relative Address Dec	Relative Address Hex	Field Description
WKAQSCPT	4	3900	F3C	Source pointer for backup and retry if alternative or option fails.
WKAQDFBK	2	3904	F40	Displacement in table to beginning of optional definition for iteration.
WKAQNLVL	1	3906	F42	Level of nesting of corresponding line.
WKAQFALS	1	3907	F43	Displacement from beginning of current line to the next), |, or >, whichever comes first.
WKAQTRUE	1	3908	F44	Displacement from beginning of current line to end of these alternatives, >.
WKAQINCT	1	3909	F45	Count of successful iterations.
WKAQSWCH	1	3910	F46	Switch byte; bit 0 (CKRCMTSW)=1 if current alternative or option is committed.
	1			Not used.

WKASERTB: Translate and Test Table for CKRSERCH, a Get-Character Routine

0	1	2	3			FF
				...		

Every entry is a byte of binary zeros, except while one character is being tested for. At that time the character in question is used as an index to WKASERTB, and the byte pointed to is made nonzero for a TRT instruction. It is subsequently reset to zero.

WKATINU: Table of Displacement and Line Number Information for Lines of Statement.

```
0        2             A     C                             hex C8
┌────────┬─────────────┬─────┬──────────────────┬──────────────┬───────┐
│        │             │     │                  │ ...          │       │
└┬───────┴┬────────────┴─────┴──────────────────┴──────────────┴───────┘
 │        │
 │        └--line number (8 bytes).  For a line-numbered data set the
 │           data set line number right adjusted in EBCDIC; for a non
 │           line-numbered data set, the first two bytes of field
 │           contain the relative line number.
 │
 └--displacement (2 bytes) of last character of line from WKACHRST-1.
```

The *i*th ten-byte entry contains the displacement from the beginning of the character string of the last character of the *i*th line and its associated data set or relative line number. Bit 0 of the *n*th entry is set to 1 when the (n-1)th word points to the last line of the statement.

SECTION VI: DIAGNOSTIC AIDS

This section contains information that may be useful in diagnosing difficulties with the syntax checker. Included are:

Register Contents

Table 16 provides a description of general register contents for locating errors in Syntax Checker Operation.

Debugging Aids

The syntax checker has one macro, three global symbols, and a flag byte, WKASFAIL, to aid in getting and interpreting dumps. See Tables 17 and 18.

The flag byte, WKASFAIL, is set to a non-zero value if the csect IPDSNEXC or IPDSNCKR detects an operational error. IPDER currently does not use this diagnostic tool. A non-zero value for WKASFAIL causes IPDER to construct the error message "System or Syntax Checker Failure" with a message number IPDnnn where nnn is a decimal number equal to the value of WKASFAIL plus 255. The executive will then send this message to the environmental system. Values of WKASFAIL and associated causes of error are listed in Table 18. The message will also be issued if IPDER is called to construct a message for which no text is defined. In this case the message number will be an even number less than 255, as explained in Table 18.

Table 17. General Register Contents (Part 1 of 2)

CSECT	Register Contents
I. At entry:	
IPDERERR	1--IPDERWKA address 13--Save area address 14--Return address 15--Entry point address
IPDSNEXC and IPDSNCKR	1--Pointer to parameter list 13--Save area address 14--Return address 15--Entry point address

Table 17. General Register Contents (Part 2 of 2)

CSECT	Register Contents
II. During processing the registers usually have the following contents:	
(a) IPDERERR	1,13,14,15--same as at entry. Register 1 is used as the base register for IPDERWKA, and register 15 is used as the base register for IPDERERR 2--bits 0-23 and 31 zero. Error code bits 0-6 in bits 24-30. 3,4,5--work registers. When the MVC instruction labeled ERREXMVC is the subject of an EX instruction, 3 contains the address of the transmitting field, 4 contains one less than the number of characters being moved, and 5 contains the location of the receiving field.
(b) IPDSNEXC	0,1,4,14,15--Work registers 2--Linkage register for calling internal subroutines 3--Branch register used only in processing free-form source 5--Index to WKATINU table 6--Location of column 1 of a line 7--Pointer used in building character string 8--Length minus one of line of statement 9--Options word DSECT register 10--Communications word DSECT register 11--Contents of register 1 upon entry to IPDSNEXC 12--Base register 13--Address of syntax checker work area (IPDSNWKA)
(c) IPDSNCKR	0,1,2,3,10,14,15--Work registers 4--Line nest list pointer 5--Qualification list pointer 6--Source pointer 7--Base register for syntax definition table 8--Contains operator or action code currently being interpreted 9--Syntax definition table displacement pointer 11,12--Base registers for CSECT 13--Address of syntax checker work area (IPDSNWKA)
Note: At times, some registers are saved so they can be used as work registers.	

Table 18. Causes of Message "System or Syntax Checker Failure" (Part 1 of 2)

Originating CSECT	WKASFAIL setting (in hexadecimal)	Cause	Associated Debugging Aids
IPDSNEXC	01-0F		Executive's registers 2-8 are saved in EXCSVRGS prior to return to calling program. The checker, if called, has previously saved registers 9-12 (probably valid) in EXCSVRGS. The registers referred to below are those in EXCSVRGS.
	01	Intermediate entry but there are no input records in buffer chain	Buffer address specified in first word of parameter list to IPDSNEXC is zero, or every buffer in chain has a count of zero records or its high order bit set to one. Register 11 contains location of parameter list passed to IPDSNEXC.
	02	Standard-form variable-length records specified in options word	The syntax checker will not accept standard-form variable-length records. Register 9 contains location of options word.
	03	Zero-length text in variable-length record	The executive has calculated a zero-length FORTRAN text field within a variable-length input record. Register 6 contains the address of that field.
	04	IPDSNEXC recalled to continue checking buffer chain but IPDSNEXC's prior return code was not 8	Register 10 contains location of communications area.
	05	Mishandling of relative line numbers.	IPDSNEXC has calculated a very large (at least greater than 65,535) relative line number. Note: issuing of return code 12 does not reset relative line number to zero.
	06	Intermediate entry after return code 12 but calling program has changed options word specification from standard-form to free-form since last call to IPDSNEXC	This is a special check on the interface to avoid ABEND of IPDSNEXC.

Table 18. Causes of Message "System or Syntax Checker Failure" (Part 2 of 2)

Originating CSECT	WKASFAIL setting (in hexa-decimal)	Cause	Associated Debugging Aids
IPDSNCKR	11-1F		Checker's registers 0-15 are saved in WKACKRGS prior to restoration of executive's registers and BR 14. The registers referred to below are those saved in WKACKRGS.
	11	Unrecognizable operator code	Hexadecimal operator code--in low order byte of register 8; displacement to bad operator code in definition table--in register 9.
	12	Too many right braces, >s	Displacement 1 byte past > operator code in definition table--in register 9.
	13	Too many right parentheses,)s	Displacement 1 byte past) operator code in definition table--in register 9.
	14	Longest table argument length is zero or too large for buffer area	Length--in register 4; location of length in definition table--in register 5.
	15	Unrecognizable or illegal table function operator code	Location of operator code in definition table--in register 2.
	16	Too many unnestings	Displacement 1 byte past end-of-line operator--in register 6.
	17	Unrecognizable action code	Action code--in low order byte of register 8; location of action code in definition table--in register 10.
	18	Length of literal zero or too large for buffer area	Length--in register 10 and in WKASRCNT; displacement to length in definition table--in register 9.
IPDERERR	not set by IPDERERR	Error processor was called with a message code for which no message text was defined.	Actual message code passed to error code processor appears as nnn of IPDnnn field in message. (If actual code was an odd number, nnn is the next lower even number.)

Table 19. Debugging Aids Which Depend on Assembly Parameters

I. Macro:

Macro Name	CSECT in which Used	Meaning
BOMBR	IPDSNEXC	The macro can be used to invalidate an operator in an Assembler Language Instruction. Zeroes are placed into the fourth byte preceding the location at which the macro is specified. When the program tries to execute the zero operator, a dump is taken. (Not used in released version.)

II. Global Symbols:

Symbol Name	CSECT in which used	Meaning
&EXCALMS	IPDSNEXC	(1) If set to *+1, the program dumps at the point at which a syntax checker system failure is recognized. (2) If set to EXCALMS, the program returns to the environmental system with an indication (error message IPD000) that the syntax checker is not operational. (Set to EXCALMS in released version.)
&EXCALMT	IPDSNEXC	Same as &EXCALMS except set the symbol to EXCALMT. (Set to EXCALMT in released version.)
&ITNLDBG	IPDSNCKR	(1) If set to 'E' (extended) or 'L' (limited), conditionally assembled tests for errors in the checker, or its interface with the executive, or the language definition table, are assembled and executed. When such a test fails, WKASFAIL is set and control is returned to IPDSNEXC to record a system or syntax checker failure (IPD000). (Set to 'E' in released version.) (2) If &ITNLDBG is not set or is set null, ' ', the conditional debugging tests in (1) are not assembled into IPDSNCKR.

At system generation time, the CHECKER macro must be specified in order to include the FORTRAN IV Syntax Checker modules in SYS1.LINKLIB of the generated system. This macro, which may specify either the FORTRAN IV or the PL/1 syntax checker, is coded as follows for FORTRAN:

CHECKER Macro Instruction Format:

Name	Operation	Operand
[name]	CHECKER	TYPE=FORTRAN [DESIGN= (design[,design]...)]

where 'design' specifies the FORTRAN language level(s) to be included as E, G, and/or H. If this parameter is omitted, level G is assumed. Level G or H also allows syntax checking for levels G1 and Code and Go.

The relationship between the CHECKER parameter specifications and the contents of SYS1.LINKLIB is illustrated in Table 19.

Table 20. Contents of SYS1.LINKLIB

CHECKER Design Values	FORTRAN IV Syntax Checker Load Modules in SYS1.LINKLIB After System Generation
E	IPDTEE, IPDSNEXC
G and/or H or null	IPDAGH, IPDSNEXC
E and G and/or H	IPDTEE, IPDAGH, IPDSNEXC

SYSTEM GENERATION PROCESSING

System/360 Operating System is generated in two stages. In Stage I, user-supplied macro instructions that describe both the installation's machine configuration and the programming options desired are analyzed and used to generate a job stream. In Stage II, this job stream is processed to generate the libraries of modules that form the user's operating system.

During Stage I the CHECKER macro tests the validity of its specified parameters and sets the following global symbols pertaining to the FORTRAN IV Syntax Checker:

&SGMENTB(57) is set to one if the CHECKER macro is specified;

&SGCKFTB(1) is set to one if TYPE=FORTRAN is specified;

&SGCKFTB(2) is set to one if E is specified as one of the design levels;

&SGCKFTB(3) is set to one if G is specified as one of the design levels or if the DESIGN parameter is omitted;

&SGCKFTB(4) is set to one if H is specified as one of the design levels.

If an invalid 'design' parameter is specified or if the CHECKER macro is specified more than once for FORTRAN, an error message is printed and system generation terminates in Stage 1 before any job control language for subsequent stages is produced.

The GENERATE macro, which is specified as the last macro instruction in a user's system generation input deck, contains these inner macro instructions that pertain to the FORTRAN IV Syntax Checker:

1. SGGEN100 -- If the CHECKER macro is specified but, in the SCHEDULR macro, OPTIONS=CRJE or OPTIONS=TSO is not, and if GENTYPE=ALL in the GENERATE macro, SGGEN100 issues a warning diagnostic, which states that all syntax checker modules will be omitted from the generated system. All the &SGCKFTB symbols are set to zero. However, if GENTYPE=PROCESSOR, the checker modules will be included in the generated system whether or not CRJE or TSO is specified.

2. SGIPD400 -- Control cards are generated to link the IPDSN and IPDER modules into one load module, IPDSNEXC, in SYS1.LINKLIB during Stage II of system generation. Modules IPDSN and IPDER, along with IPDAGH and IPDTEE, are in the Syntax Checker component library, SYS1.FO550.

3. SGIPD500 -- Controls cards for the IEBCOPY utility program are generated so that the IPDTEE and/or IPDAGH modules will be copied from SYS1.FO550 to SYS1.LINKLIB during Stage II of system generation. If &SGCKFTB(2) is one, IPDTEE is copied; if &SGCKFTB(3) and/or &SGCKFTB(4) are one, IPDAGH is copied.

If &SGCKFTB(1) is zero, SGIPD400 and SGIPD500 are not called. No control cards will be generated and the FORTRAN checker modules will not be placed on SYS1.LINKLIB.

Error Code	Text	Originator					
				IPDTEE		IPDAGH	
		IPDSNEXC	IPDSNCKR	Action Code	Error Code	Action Code	Error Code
000, 001	System or syntax checker failure	X	X				
002, 003	Unrecognizable stmnt or misspelled keywd				X		X
004, 005	Unsigned integer expected				X		X
006, 007	Expression expected				X		X
008, 009	Possibly too many subscripts precede			X		X	
010, 011	Too many subscripts			X		X	
012, 013	) expected				X		X
014, 015	Arith IF requires statement number list						X
016, 017	Invalid expression in IF statement						X
018, 019	Unrecognizable stmnt after logical IF						X
020, 021	Non-zero integer expected			X		X	
022, 023	Illegal statement after logical IF						X
024, 025	Statement expected	X					
026, 027	Data set ref number expected			X	X	X	X
028, 029	Length specification invalid						X
030, 031	(expected				X		X
032, 033	Name expected				X		X
034, 035	Dummy argument expected				X		X
036, 037	Array dimensions expected				X		X
038, 039	/ expected						X
X - indicates the originator of the message							

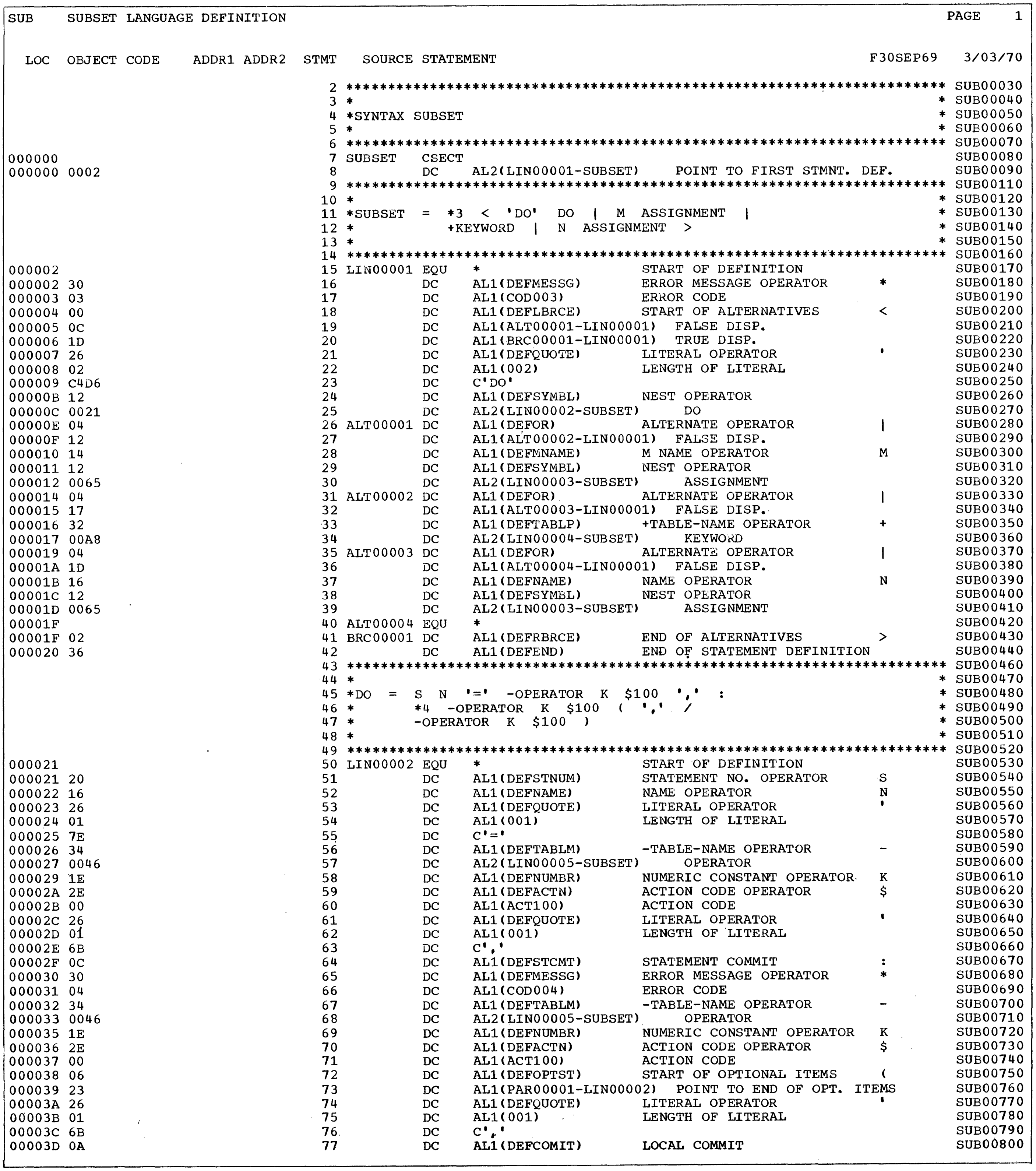
		Originator					
				IPDTEE		IPDAGH	
Error Code	Text	IPDSNEXC	IPDSNCKR	Action Code	Error Code	Action Code	Error Code
040, 041	Invalid data type						X
042, 043	Statement number expected			X	X	X	X
044, 045	'TO' expected						X
046, 047	Argument expected				X		X
048, 049	Data list expected						X
050, 051	Relational operator expected						X
052, 053	, expected				X		X
054, 055	Operand expected in arith expression				X		X
056, 057	Operand expected in logical expression						X
058, 059	I/O list item expected				X		X
060, 061	' expected				X		X
062, 063	Incorrect parameter - must be E, L, or U				X		X
064, 065	DEBUG parameter expected						X
066, 067	Subscript expected				X		
068, 069	Too many levels of parentheses				X		X
070, 071	Statement too long	X					
072, 073	Integer expected			X		X	
074, 075	Complex number invalid					X	
076, 077	Delimiter missing or invalid FORMAT code				X		X
078, 079	Variable list expected						X
080, 081	. expected in FORMAT code				X		X
082, 083	Name too long		X				
084, 085	Statement number invalid	X	X				
086, 087	H-literal incomplete		X				
088, 089	Field width not in range 1-255		X	X		X	

X - indicates the originator of the message

Error Code	Text	Originator					
				IPDTEE		IPDAGH	
		IPDSNEXC	IPDSNCKR	Action Code	Error Code	Action Code	Error Code
090, 091	Literal exceeds 255 characters		X				
092, 093	Statement analysis exceeds table limits		X				
094, 095	END requires blank label & contin fields			X		X	
096, 097	Invalid or excess source characters		X				
098, 099	Invalid range in IMPLICIT statement					X	
100, 101	First line is a continuation	X					
102, 103	Comment line within statement	X					
104, 105	Too many lines in statement	X					
106, 107	Too many decimal places for field width			X		X	
108, 109	Decimal places must be specified			X		X	
110, 111	) required for implied DO			X		X	
112, 113	DO variable cannot be subscripted			X		X	
114, 115	DEBUG facility not supported					X	
116, 117	Exponent missing or invalid		X				
118, 119	Real constant must have at least 1 digit		X				
120, 121	Integer too large		X				
122, 123	Closing ' expected		X				
124, 125	Data illegal for dummy array						X
126, 127	Real number expected					X	
128, 129	Invalid characters after STOP or PAUSE				X		X
130, 131	Real number outside of allowable range		X				
132, 133	FORMAT stmnt no. or array name expected						X

X - indicates the originator of the message

Error Code	Text	Originator					
				IPDTEE		IPDAGH	
		IPDSNEXC	IPDSNCKR	Action Code	Error Code	Action Code	Error Code
134, 135	Misplaced length specification precedes						X
136, 137	List-directed I/O illegal					X	
138,139	Arith exp expected after relational op						X
140,141	Invalid comma in DO				X		X
142,143	= expected				X		X
144,145	Literal contains no characters		X				
146, 147	Invalid IF after logical IF						X
148, 149	Invalid decimal point		X				
150, 151	Too many digits in statement number	X					
152, 153	Statement no. not complete on initial line	X					
154,155	Last line is a continued line	X					
156, 157	Invalid characters before statement	X					
158, 159	Too many subscripts precede			X		X	
160, 161	END too far on line			X		X	
X - indicates the originator of the message							

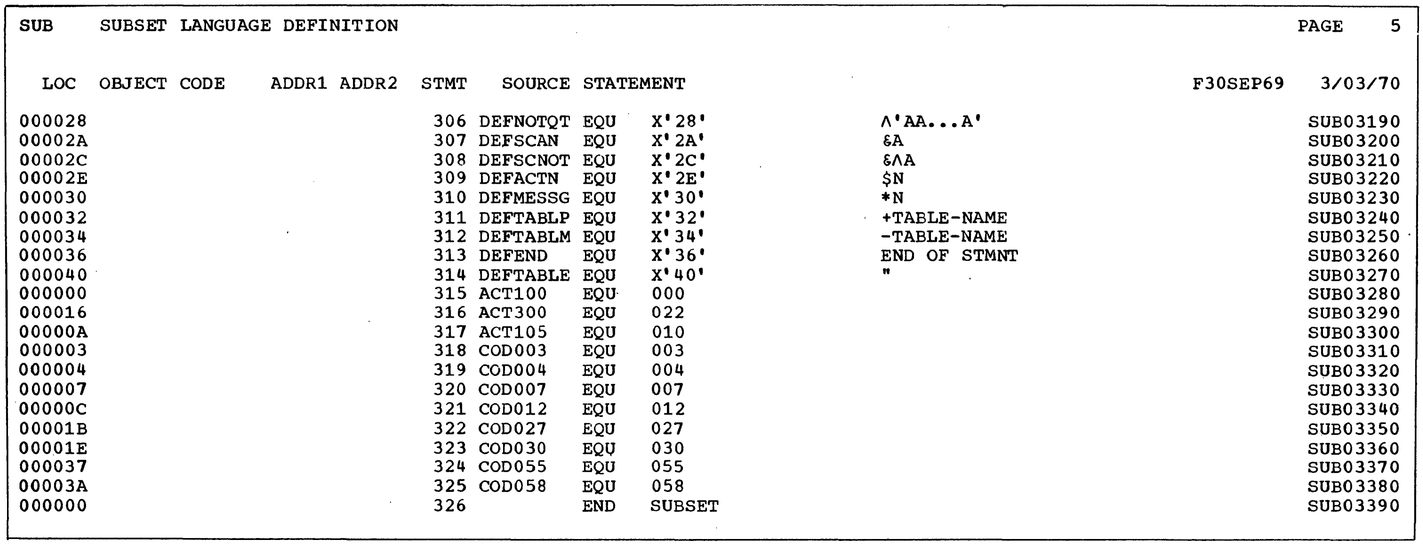


```
SUB    SUBSET LANGUAGE DEFINITION                                                                           PAGE    1

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                           F30SEP69   3/03/70

                                      2 *********************************************************************** SUB00030
                                      3 *                                                                     * SUB00040
                                      4 *SYNTAX SUBSET                                                        * SUB00050
                                      5 *                                                                     * SUB00060
                                      6 *********************************************************************** SUB00070
000000                                7 SUBSET   CSECT                                                          SUB00080
000000 0002                           8          DC    AL2(LIN00001-SUBSET)    POINT TO FIRST STMNT. DEF.       SUB00090
                                      9 *********************************************************************** SUB00110
                                     10 *                                                                     * SUB00120
                                     11 *SUBSET  =  *3  <  'DO'  DO  |  M  ASSIGNMENT  |                      * SUB00130
                                     12 *           +KEYWORD  |  N  ASSIGNMENT  >                             * SUB00140
                                     13 *                                                                     * SUB00150
                                     14 *********************************************************************** SUB00160
000002                               15 LIN00001 EQU   *                   START OF DEFINITION                  SUB00170
000002 30                            16          DC    AL1(DEFMESSG)       ERROR MESSAGE OPERATOR      *        SUB00180
000003 03                            17          DC    AL1(COD003)         ERROR CODE                           SUB00190
000004 00                            18          DC    AL1(DEFLBRCE)       START OF ALTERNATIVES       <        SUB00200
000005 0C                            19          DC    AL1(ALT00001-LIN00001)  FALSE DISP.                      SUB00210
000006 1D                            20          DC    AL1(BRC00001-LIN00001)  TRUE DISP.                       SUB00220
000007 26                            21          DC    AL1(DEFQUOTE)       LITERAL OPERATOR            '        SUB00230
000008 02                            22          DC    AL1(002)            LENGTH OF LITERAL                    SUB00240
000009 C4D6                          23          DC    C'DO'                                                    SUB00250
00000B 12                            24          DC    AL1(DEFSYMBL)       NEST OPERATOR                        SUB00260
00000C 0021                          25          DC    AL2(LIN00002-SUBSET)     DO                              SUB00270
00000E 04                            26 ALT00001 DC    AL1(DEFOR)          ALTERNATE OPERATOR          |        SUB00280
00000F 12                            27          DC    AL1(ALT00002-LIN00001)  FALSE DISP.                      SUB00290
000010 14                            28          DC    AL1(DEFMNAME)       M NAME OPERATOR             M        SUB00300
000011 12                            29          DC    AL1(DEFSYMBL)       NEST OPERATOR                        SUB00310
000012 0065                          30          DC    AL2(LIN00003-SUBSET)     ASSIGNMENT                      SUB00320
000014 04                            31 ALT00002 DC    AL1(DEFOR)          ALTERNATE OPERATOR          |        SUB00330
000015 17                            32          DC    AL1(ALT00003-LIN00001)  FALSE DISP.                      SUB00340
000016 32                            33          DC    AL1(DEFTABLP)       +TABLE-NAME OPERATOR        +        SUB00350
000017 00A8                          34          DC    AL2(LIN00004-SUBSET)     KEYWORD                         SUB00360
000019 04                            35 ALT00003 DC    AL1(DEFOR)          ALTERNATE OPERATOR          |        SUB00370
00001A 1D                            36          DC    AL1(ALT00004-LIN00001)  FALSE DISP.                      SUB00380
00001B 16                            37          DC    AL1(DEFNAME)        NAME OPERATOR               N        SUB00390
00001C 12                            38          DC    AL1(DEFSYMBL)       NEST OPERATOR                        SUB00400
00001D 0065                          39          DC    AL2(LIN00003-SUBSET)     ASSIGNMENT                      SUB00410
00001F                               40 ALT00004 EQU   *                                                        SUB00420
00001F 02                            41 BRC00001 DC    AL1(DEFRBRCE)       END OF ALTERNATIVES         >        SUB00430
000020 36                            42          DC    AL1(DEFEND)         END OF STATEMENT DEFINITION          SUB00440
                                     43 *********************************************************************** SUB00460
                                     44 *                                                                     * SUB00470
                                     45 *DO  =  S  N  '='  -OPERATOR  K  $100  ','  :                         * SUB00480
                                     46 *       *4  -OPERATOR  K  $100  (  ','  /                             * SUB00490
                                     47 *       -OPERATOR  K  $100  )                                         * SUB00500
                                     48 *                                                                     * SUB00510
                                     49 *********************************************************************** SUB00520
000021                               50 LIN00002 EQU   *                   START OF DEFINITION                  SUB00530
000021 20                            51          DC    AL1(DEFSTNUM)       STATEMENT NO. OPERATOR      S        SUB00540
000022 16                            52          DC    AL1(DEFNAME)        NAME OPERATOR               N        SUB00550
000023 26                            53          DC    AL1(DEFQUOTE)       LITERAL OPERATOR            '        SUB00560
000024 01                            54          DC    AL1(001)            LENGTH OF LITERAL                    SUB00570
000025 7E                            55          DC    C'='                                                     SUB00580
000026 34                            56          DC    AL1(DEFTABLM)       -TABLE-NAME OPERATOR        -        SUB00590
000027 0046                          57          DC    AL2(LIN00005-SUBSET)     OPERATOR                        SUB00600
000029 1E                            58          DC    AL1(DEFNUMBR)       NUMERIC CONSTANT OPERATOR   K        SUB00610
00002A 2E                            59          DC    AL1(DEFACTN)        ACTION CODE OPERATOR        $        SUB00620
00002B 00                            60          DC    AL1(ACT100)         ACTION CODE                          SUB00630
00002C 26                            61          DC    AL1(DEFQUOTE)       LITERAL OPERATOR            '        SUB00640
00002D 01                            62          DC    AL1(001)            LENGTH OF LITERAL                    SUB00650
00002E 6B                            63          DC    C','                                                     SUB00660
00002F 0C                            64          DC    AL1(DEFSTCMT)       STATEMENT COMMIT            :        SUB00670
000030 30                            65          DC    AL1(DEFMESSG)       ERROR MESSAGE OPERATOR      *        SUB00680
000031 04                            66          DC    AL1(COD004)         ERROR CODE                           SUB00690
000032 34                            67          DC    AL1(DEFTABLM)       -TABLE-NAME OPERATOR        -        SUB00700
000033 0046                          68          DC    AL2(LIN00005-SUBSET)     OPERATOR                        SUB00710
000035 1E                            69          DC    AL1(DEFNUMBR)       NUMERIC CONSTANT OPERATOR   K        SUB00720
000036 2E                            70          DC    AL1(DEFACTN)        ACTION CODE OPERATOR        $        SUB00730
000037 00                            71          DC    AL1(ACT100)         ACTION CODE                          SUB00740
000038 06                            72          DC    AL1(DEFOPTST)       START OF OPTIONAL ITEMS     (        SUB00750
000039 23                            73          DC    AL1(PAR00001-LIN00002)  POINT TO END OF OPT. ITEMS       SUB00760
00003A 26                            74          DC    AL1(DEFQUOTE)       LITERAL OPERATOR            '        SUB00770
00003B 01                            75          DC    AL1(001)            LENGTH OF LITERAL                    SUB00780
00003C 6B                            76          DC    C','                                                     SUB00790
00003D 0A                            77          DC    AL1(DEFCOMIT)       LOCAL COMMIT                         SUB00800
```

```
SUB    SUBSET LANGUAGE DEFINITION                                                                       PAGE    2

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          F30SEP69   3/03/70

00003E 34                            78          DC    AL1(DEFTABLM)       -TABLE-NAME OPERATOR        -        SUB00810
00003F 0046                          79          DC    AL2(LIN00005-SUBSET)     OPERATOR                        SUB00820
000041 1E                            80          DC    AL1(DEFNUMBR)       NUMERIC CONSTANT OPERATOR   K        SUB00830
000042 2E                            81          DC    AL1(DEFACTN)        ACTION CODE OPERATOR        $        SUB00840
000043 00                            82          DC    AL1(ACT100)         ACTION CODE                          SUB00850
000044 08                            83 PAR00001 DC    AL1(DEFOPTED)       END OF OPTIONAL ITEMS       )        SUB00860
000045 36                            84          DC    AL1(DEFEND)         END OF STATEMENT DEFINITION          SUB00870
                                     85 ********************************************************************** SUB00890
                                     86 *                                                                    * SUB00900
                                     87 *OPERATOR  =  "  '+'  0  '-'  0  '**'  0  '*'  0  '/'  0  "           * SUB00910
                                     88 *                                                                    * SUB00920
                                     89 ********************************************************************** SUB00930
000046                               90 LIN00005 EQU   *                   START OF DEFINITION                  SUB00940
000046 40                            91          DC    AL1(DEFTABLE)       START OF TABLE ENTRIES      "        SUB00950
000047 001D                          92          DC    AL2(TAB00001-*+1)    LENGTH OF TABLE                     SUB00960
000049 01                            93          DC    AL1(001)            LENGTH OF LITERAL                    SUB00970
00004A 4E                            94          DC    C'+'                                                     SUB00980
00004B 2E                            95          DC    AL1(DEFACTN)        ACTION CODE OPERATOR        0        SUB00990
00004C FF                            96          DC    X'FF'               NULL ACTION CODE                     SUB01000
00004D E3                            97          DC    C'T'                TABLE FUNCTION PAD CHARACTER         SUB01010
00004E 01                            98          DC    AL1(001)            LENGTH OF LITERAL                    SUB01020
00004F 60                            99          DC    C'-'                                                     SUB01030
000050 2E                           100          DC    AL1(DEFACTN)        ACTION CODE OPERATOR        0        SUB01040
000051 FF                           101          DC    X'FF'               NULL ACTION CODE                     SUB01050
000052 E3                           102          DC    C'T'                TABLE FUNCTION PAD CHARACTER         SUB01060
000053 02                           103          DC    AL1(002)            LENGTH OF LITERAL                    SUB01070
000054 5C5C                         104          DC    C'**'                                                    SUB01080
000056 2E                           105          DC    AL1(DEFACTN)        ACTION CODE OPERATOR        0        SUB01090
000057 FF                           106          DC    X'FF'               NULL ACTION CODE                     SUB01100
000058 E3                           107          DC    C'T'                TABLE FUNCTION PAD CHARACTER         SUB01110
000059 01                           108          DC    AL1(001)            LENGTH OF LITERAL                    SUB01120
00005A 5C                           109          DC    C'*'                                                     SUB01130
00005B 2E                           110          DC    AL1(DEFACTN)        ACTION CODE OPERATOR        0        SUB01140
00005C FF                           111          DC    X'FF'               NULL ACTION CODE                     SUB01150
00005D E3                           112          DC    C'T'                TABLE FUNCTION PAD CHARACTER         SUB01160
00005E 01                           113          DC    AL1(001)            LENGTH OF LITERAL                    SUB01170
00005F 61                           114          DC    C'/'                                                     SUB01180
000060 2E                           115          DC    AL1(DEFACTN)        ACTION CODE OPERATOR        0        SUB01190
000061 FF                           116          DC    X'FF'               NULL ACTION CODE                     SUB01200
000062 E3                           117          DC    C'T'                TABLE FUNCTION PAD CHARACTER         SUB01210
000063 02                           118 TAB00001 DC    AL1(002)            LENGTH OF LONGEST TABLE ARG          SUB01220
000064 36                           119          DC    AL1(DEFEND)         END OF STATEMENT DEFINITION          SUB01230
                                    120 ********************************************************************** SUB01250
                                    121 *                                                                    * SUB01260
                                    122 *ASSIGNMENT  =  '='  :  *7  EXPRESSION                               * SUB01270
                                    123 *                                                                    * SUB01280
                                    124 ********************************************************************** SUB01290
000065                              125 LIN00003 EQU   *                   START OF DEFINITION                  SUB01300
000065 26                           126          DC    AL1(DEFQUOTE)       LITERAL OPERATOR            '        SUB01310
000066 01                           127          DC    AL1(001)            LENGTH OF LITERAL                    SUB01320
000067 7E                           128          DC    C'='                                                     SUB01330
000068 0C                           129          DC    AL1(DEFSTCMT)       STATEMENT COMMIT            :        SUB01340
000069 30                           130          DC    AL1(DEFMESSG)       ERROR MESSAGE OPERATOR      *        SUB01350
00006A 07                           131          DC    AL1(COD007)         ERROR CODE                           SUB01360
00006B 12                           132          DC    AL1(DEFSYMBL)       NEST OPERATOR                        SUB01370
00006C 006F                         133          DC    AL2(LIN00006-SUBSET)     EXPRESSION                      SUB01380
00006E 36                           134          DC    AL1(DEFEND)         END OF STATEMENT DEFINITION          SUB01390
                                    135 ********************************************************************** SUB01410
                                    136 *                                                                    * SUB01420
                                    137 *EXPRESSION  =  (  <  '+'  |  '-'  >  )                              * SUB01430
                                    138 *               OPERAND  *55  (  +OPERATOR  /  OPERAND  ...  )       * SUB01440
                                    139 *                                                                    * SUB01450
                                    140 ********************************************************************** SUB01460
00006F                              141 LIN00006 EQU   *                   START OF DEFINITION                  SUB01470
00006F 06                           142          DC    AL1(DEFOPTST)       START OF OPTIONAL ITEMS     (        SUB01480
000070 0E                           143          DC    AL1(PAR00002-LIN00006)  POINT TO END OF OPT. ITEMS       SUB01490
000071 00                           144          DC    AL1(DEFLBRCE)       START OF ALTERNATIVES       <        SUB01500
000072 08                           145          DC    AL1(ALT00005-LIN00006)  FALSE DISP.                      SUB01510
000073 0D                           146          DC    AL1(BRC00002-LIN00006)  TRUE DISP.                       SUB01520
000074 26                           147          DC    AL1(DEFQUOTE)       LITERAL OPERATOR            '        SUB01530
000075 01                           148          DC    AL1(001)            LENGTH OF LITERAL                    SUB01540
000076 4E                           149          DC    C'+'                                                     SUB01550
000077 04                           150 ALT00005 DC    AL1(DEFOR)          ALTERNATE OPERATOR          |        SUB01560
000078 0D                           151          DC    AL1(ALT00006-LIN00006)  FALSE DISP.                      SUB01570
000079 26                           152          DC    AL1(DEFQUOTE)       LITERAL OPERATOR            '        SUB01580
00007A 01                           153          DC    AL1(001)            LENGTH OF LITERAL                    SUB01590
```

```
SUB    SUBSET LANGUAGE DEFINITION                                                                    PAGE    3


  LOC  OBJECT CODE     ADDR1 ADDR2  STMT    SOURCE STATEMENT                                    F30SEP69   3/03/70

00007B 60                            154          DC    C'-'                                                  SUB01600
00007C                               155 ALT00006 EQU   *                                                     SUB01610
00007C 02                            156 BRC00002 DC    AL1(DEFRBRCE)       END OF ALTERNATIVES          >       SUB01620
00007D 08                            157 PAR00002 DC    AL1(DEFOPTED)       END OF OPTIONAL ITEMS        )       SUB01630
00007E 12                            158          DC    AL1(DEFSYMBL)       NEST OPERATOR                        SUB01640
00007F 008F                          159          DC    AL2(LIN00007-SUBSET)     OPERAND                         SUB01650
000081 30                            160          DC    AL1(DEFMESSG)       ERROR MESSAGE OPERATOR       *       SUB01660
000082 37                            161          DC    AL1(COD055)         ERROR CODE                           SUB01670
000083 06                            162          DC    AL1(DEFOPTST)       START OF OPTIONAL ITEMS      (       SUB01680
000084 1E                            163          DC    AL1(PAR00003-LIN00006)  POINT TO END OF OPT. ITEMS       SUB01690
000085 32                            164          DC    AL1(DEFTABLP)       +TABLE-NAME OPERATOR         +       SUB01700
000086 0046                          165          DC    AL2(LIN00005-SUBSET)     OPERATOR                        SUB01710
000088 0A                            166          DC    AL1(DEFCOMIT)       LOCAL COMMIT                 /       SUB01720
000089 12                            167          DC    AL1(DEFSYMBL)       NEST OPFRATOR                        SUB01730
00008A 008F                          168          DC    AL2(LIN00007-SUBSET)     OPERAND                         SUB01740
00008C 0E                            169          DC    AL1(DEFITIND)       INDEFINITE ITERATION         ...     SUB01750
00008D 08                            170 PAR00003 DC    AL1(DEFOPTED)       END OF OPTIONAL ITEMS        )       SUB01760
00008E 36                            171          DC    AL1(DEFEND)         END OF STATEMENT DEFINITION          SUB01770
                                     172 ******************************************************************* SUB01790
                                     173 *                                                                 * SUB01800
                                     174 *OPERAND  =  <  N  |  K  |  '('  /  *7  EXPRESSION  *12  ')'  >   * SUB01810
                                     175 *                                                                 * SUB01820
                                     176 ******************************************************************* SUB01830
00008F                               177 LIN00007 EQU   *                   START OF DEFINITION                  SUB01840
00008F 00                            178          DC    AL1(DEFLBRCE)       START OF ALTERNATIVES        <       SUB01850
000090 04                            179          DC    AL1(ALT00007-LIN00007)  FALSE DISP.                      SUB01860
000091 17                            180          DC    AL1(BRC00003-LIN00007)  TRUE DISP.                       SUB01870
000092 16                            181          DC    AL1(DEFNAME)        NAME OPERATOR                N       SUB01880
000093 04                            182 ALT00007 DC    AL1(DEFOR)          ALTERNATE OPERATOR           |       SUB01890
000094 07                            183          DC    AL1(ALT00008-LIN00007)  FALSE DISP.                      SUB01900
000095 1E                            184          DC    AL1(DEFNUMBR)       NUMERIC CONSTANT OPERATOR    K       SUB01910
000096 04                            185 ALT00008 DC    AL1(DEFOR)          ALTERNATE OPERATOR           |       SUB01920
000097 17                            186          DC    AL1(ALT00009-LIN00007)  FALSE DISP.                      SUB01930
000098 26                            187          DC    AL1(DEFQUOTE)       LITERAL OPERATOR             '       SUB01940
000099 01                            188          DC    AL1(001)            LENGTH OF LITERAL                    SUB01950
00009A 4D                            189          DC    C'('                                                  SUB01960
00009B 0A                            190          DC    AL1(DEFCOMIT)       LOCAL COMMIT                 /       SUB01970
00009C 30                            191          DC    AL1(DEFMESSG)       ERROR MESSAGE OPERATOR       *       SUB01980
00009D 07                            192          DC    AL1(COD007)         ERROR CODE                           SUB01990
00009E 12                            193          DC    AL1(DEFSYMBL)       NEST OPERATOR                        SUB02000
00009F 006F                          194          DC    AL2(LIN00006-SUBSET)     EXPRESSION                      SUB02010
0000A1 30                            195          DC    AL1(DEFMESSG)       ERROR MESSAGE OPERATOR       *       SUB02020
0000A2 0C                            196          DC    AL1(COD012)         ERROR CODE                           SUB02030
0000A3 26                            197          DC    AL1(DEFQUOTE)       LITERAL OPERATOR             '       SUB02040
0000A4 01                            198          DC    AL1(001)            LENGTH OF LITERAL                    SUB02050
0000A5 5D                            199          DC    C')'                                                  SUB02060
0000A6                               200 ALT00009 EQU   *                                                     SUB02070
0000A6 02                            201 BRC00003 DC    AL1(DEFRBRCE)       END OF ALTERNATIVES          >       SUB02080
0000A7 36                            202          DC    AL1(DEFEND)         END OF STATEMENT DEFINITION          SUB02090
                                     203 ******************************************************************* SUB02110
                                     204 *                                                                 * SUB02120
                                     205 *KEYWORD  =  "  'CONTINUE'  0  'STOP'  0  'READ'  INOUT           * SUB02130
                                     206 *                'WRITE'  INOUT  'END'  $300  "                   * SUB02140
                                     207 *                                                                 * SUB02150
                                     208 ******************************************************************* SUB02160
0000A8                               209 LIN00004 EQU   *                   START OF DEFINITION                  SUB02170
0000A8 40                            210          DC    AL1(DEFTABLE)       START OF TABLE ENTRIES       "       SUB02180
0000A9 002F                          211          DC    AL2(TAB00002-*+1)    LENGTH OF TABLE                     SUB02190
0000AB 08                            212          DC    AL1(008)            LENGTH OF LITERAL                    SUB02200
0000AC C3D6D5E3C9D5E4C5              213          DC    C'CONTINUE'                                           SUB02210
0000B4 2E                            214          DC    AL1(DEFACTN)        ACTION CODE OPERATOR         0       SUB02220
0000B5 FF                            215          DC    X'FF'               NULL ACTION CODE                     SUB02230
0000B6 E3                            216          DC    C'T'                TABLE FUNCTION PAD CHARACTER         SUB02240
0000B7 04                            217          DC    AL1(004)            LENGTH OF LITERAL                    SUB02250
0000B8 E2E3D6D7                      218          DC    C'STOP'                                               SUB02260
0000BC 2E                            219          DC    AL1(DEFACTN)        ACTION CODE OPERATOR         0       SUB02270
0000BD FF                            220          DC    X'FF'               NULL ACTION CODE                     SUB02280
0000BE E3                            221          DC    C'T'                TABLE FUNCTION PAD CHARACTER         SUB02290
0000BF 04                            222          DC    AL1(004)            LENGTH OF LITERAL                    SUB02300
0000C0 D9C5C1C4                      223          DC    C'READ'                                               SUB02310
0000C4 12                            224          DC    AL1(DEFSYMBL)       NEST OPERATOR                        SUB02320
0000C5 00D9                          225          DC    AL2(LIN00008-SUBSET)     INOUT                           SUB02330
0000C7 05                            226          DC    AL1(005)            LENGTH OF LITERAL                    SUB02340
0000C8 E6D9C9E3C5                    227          DC    C'WRITE'                                              SUB02350
0000CD 12                            228          DC    AL1(DEFSYMBL)       NEST OPERATOR                        SUB02360
0000CE 00D9                          229          DC    AL2(LIN00008-SUBSET)     INOUT                           SUB02370
```

```
SUB     SUBSET LANGUAGE DEFINITION                                                                  PAGE    4


  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  F30SEP69   3/03/70

0000D0 03                            230            DC     AL1(003)           LENGTH OF LITERAL                   SUB02380
0000D1 C5D5C4                        231            DC     C'END'                                                 SUB02390
0000D4 2E                            232            DC     AL1(DEFACTN)       ACTION CODE OPERATOR        $       SUB02400
0000D5 16                            233            DC     AL1(ACT300)        ACTION CODE                         SUB02410
0000D6 E3                            234            DC     C'T'               TABLE FUNCTION PAD CHARACTER        SUB02420
0000D7 08                            235 TAB00002 DC       AL1(008)           LENGTH OF LONGEST TABLE ARG         SUB02430
0000D8 36                            236            DC     AL1(DEFEND)        END OF STATEMENT DEFINITION         SUB02440
                                     237 ********************************************************************** SUB02460
                                     238 *                                                                   * SUB02470
                                     239 *INOUT  =  :  *30  '('  *27  K  $105  *12  ')'  *58  IOLIST            * SUB02480
                                     240 *                                                                   * SUB02490
                                     241 ********************************************************************** SUB02500
0000D9                               242 LIN00008 EQU      *                  START OF DEFINITION                 SUB02510
0000D9 0C                            243            DC     AL1(DEFSTCMT)      STATEMENT COMMIT            :       SUB02520
0000DA 30                            244            DC     AL1(DEFMESSG)      ERROR MESSAGE OPERATOR      *       SUB02530
0000DB 1E                            245            DC     AL1(COD030)        ERROR CODE                          SUB02540
0000DC 26                            246            DC     AL1(DEFQUOTE)      LITERAL OPERATOR            '       SUB02550
0000DD 01                            247            DC     AL1(001)           LENGTH OF LITERAL                   SUB02560
0000DE 4D                            248            DC     C'('                                                   SUB02570
0000DF 30                            249            DC     AL1(DEFMESSG)      ERROR MESSAGE OPERATOR      *       SUB02580
0000E0 1B                            250            DC     AL1(COD027)        ERROR CODE                          SUB02590
0000E1 1E                            251            DC     AL1(DEFNUMBR)      NUMERIC CONSTANT OPERATOR   K       SUB02600
0000E2 2E                            252            DC     AL1(DEFACTN)       ACTION CODE OPERATOR        $       SUB02610
0000E3 0A                            253            DC     AL1(ACT105)        ACTION CODE                         SUB02620
0000E4 30                            254            DC     AL1(DEFMESSG)      ERROR MESSAGE OPERATOR      *       SUB02630
0000E5 0C                            255            DC     AL1(COD012)        ERROR CODE                          SUB02640
0000E6 26                            256            DC     AL1(DEFQUOTE)      LITERAL OPERATOR            '       SUB02650
0000E7 01                            257            DC     AL1(001)           LENGTH OF LITERAL                   SUB02660
0000E8 5D                            258            DC     C')'                                                   SUB02670
0000E9 30                            259            DC     AL1(DEFMESSG)      ERROR MESSAGE OPERATOR      *       SUB02680
0000EA 3A                            260            DC     AL1(COD058)        ERROR CODE                          SUB02690
0000EB 12                            261            DC     AL1(DEFSYMBL)      NEST OPERATOR                       SUB02700
0000EC 00EF                          262            DC     AL2(LIN00009-SUBSET)    IOLIST                         SUB02710
0000EE 36                            263            DC     AL1(DEFEND)        END OF STATEMENT DEFINITION         SUB02720
                                     264 ********************************************************************** SUB02740
                                     265 *                                                                   * SUB02750
                                     266 *IOLIST  =  N  (  ','  /  N  ...  )                                  * SUB02760
                                     267 *                                                                   * SUB02770
                                     268 ********************************************************************** SUB02780
0000EF                               269 LIN00009 EQU      *                  START OF DEFINITION                 SUB02790
0000EF 16                            270            DC     AL1(DEFNAME)       NAME OPERATOR               N       SUB02800
0000F0 06                            271            DC     AL1(DEFOPTST)      START OF OPTIONAL ITEMS     (       SUB02810
0000F1 09                            272            DC     AL1(PAR00004-LIN00009)  POINT TO END OF OPT. ITEMS     SUB02820
0000F2 26                            273            DC     AL1(DEFQUOTE)      LITERAL OPERATOR            '       SUB02830
0000F3 01                            274            DC     AL1(001)           LENGTH OF LITERAL                   SUB02840
0000F4 6B                            275            DC     C','                                                   SUB02850
0000F5 0A                            276            DC     AL1(DEFCOMIT)      LOCAL COMMIT                /       SUB02860
0000F6 16                            277            DC     AL1(DEFNAME)       NAME OPERATOR               N       SUB02870
0000F7 0E                            278            DC     AL1(DEFITIND)      INDEFINITE ITERATION        ...     SUB02880
0000F8 08                            279 PAR00004 DC       AL1(DEFOPTED)      END OF OPTIONAL ITEMS       )       SUB02890
0000F9 36                            280            DC     AL1(DEFEND)        END OF STATEMENT DEFINITION         SUB02900
                                     281 ********************************************************************** SUB02920
                                     282 *                                                                   * SUB02930
                                     283 *SYNTAX END                                                         * SUB02940
                                     284 *                                                                   * SUB02950
                                     285 ********************************************************************** SUB02960
000000                               286 DEFLBRCE EQU      X'00'              <                                   SUB02990
000002                               287 DEFRBRCE EQU      X'02'              >                                   SUB03000
000004                               288 DEFOR    EQU      X'04'              |                                   SUB03010
000006                               289 DEFOPTST EQU      X'06'              (                                   SUB03020
000008                               290 DEFOPTED EQU      X'08'              )                                   SUB03030
00000A                               291 DEFCOMIT EQU      X'0A'              /                                   SUB03040
00000C                               292 DEFSTCMT EQU      X'0C'              :                                   SUB03050
00000E                               293 DEFITIND EQU      X'0E'              ...                                 SUB03060
000010                               294 DEFITDEF EQU      X'10'              .N.                                 SUB03070
000012                               295 DEFSYMBL EQU      X'12'              SYMBOL                              SUB03080
000014                               296 DEFMNAME EQU      X'14'              M                                   SUB03090
000016                               297 DEFNAME  EQU      X'16'              N                                   SUB03100
000018                               298 DEFLETTR EQU      X'18'              L                                   SUB03110
00001A                               299 DEFDIGIT EQU      X'1A'              D                                   SUB03120
00001C                               300 DEFALMER EQU      X'1C'              A                                   SUB03130
00001E                               301 DEFNUMBR EQU      X'1E'              K                                   SUB03140
000020                               302 DEFSTNUM EQU      X'20'              S                                   SUB03150
000022                               303 DEFHOLLR EQU      X'22'              H                                   SUB03160
000024                               304 DEFCSTRG EQU      X'24'              C                                   SUB03170
000026                               305 DEFQUOTE EQU      X'26'              'AA...A'                            SUB03180
```



```
SUB     SUBSET LANGUAGE DEFINITION                                                                              PAGE    5

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT    SOURCE STATEMENT                                             F30SEP69    3/03/70

000028                              306 DEFNOTQT EQU    X'28'                ∧'AA...A'                           SUB03190
00002A                              307 DEFSCAN  EQU    X'2A'                &A                                  SUB03200
00002C                              308 DEFSCNOT EQU    X'2C'                &∧A                                 SUB03210
00002E                              309 DEFACTN  EQU    X'2E'                $N                                  SUB03220
000030                              310 DEFMESSG EQU    X'30'                *N                                  SUB03230
000032                              311 DEFTABLP EQU    X'32'                +TABLE-NAME                         SUB03240
000034                              312 DEFTABLM EQU    X'34'                -TABLE-NAME                         SUB03250
000036                              313 DEFEND   EQU    X'36'                END OF STMNT                        SUB03260
000040                              314 DEFTABLE EQU    X'40'                "                                   SUB03270
000000                              315 ACT100   EQU    000                                                      SUB03280
000016                              316 ACT300   EQU    022                                                      SUB03290
00000A                              317 ACT105   EQU    010                                                      SUB03300
000003                              318 COD003   EQU    003                                                      SUB03310
000004                              319 COD004   EQU    004                                                      SUB03320
000007                              320 COD007   EQU    007                                                      SUB03330
00000C                              321 COD012   EQU    012                                                      SUB03340
00001B                              322 COD027   EQU    027                                                      SUB03350
00001E                              323 COD030   EQU    030                                                      SUB03360
000037                              324 COD055   EQU    055                                                      SUB03370
00003A                              325 COD058   EQU    058                                                      SUB03380
000000                              326          END    SUBSET                                                   SUB03390
```

APPENDIX D: METALANGUAGE FOR IPDTEE AND IPDAGH

METALANGUAGE USED FOR IPDTEE SYNTAX DEFINITION TABLE

```
TEE 3  37IPDTEE, FORTRAN IV LEVEL E DEFINITION                          2048
SYNTAX IPDTEE                                                           IPDE0010
IPDTEE  =   *3   < 'DO' DO | M ASSIGNMENT | +KEYWORD | N ASSIGNMENT >   IPDE0030
*                                                                     * IPDE0040
*     THIS LINE DETERMINES THE OVERALL STRATEGY                       * IPDE0050
*     IN SCANNING STATEMENTS.  ERROR MESSAGE                          * IPDE0060
*     3 IS ISSUED IF THE STATEMENT IS NONE OF                         * IPDE0070
*     THE ALTERNATIVES, SINCE THIS IS THE FIRST                       * IPDE0080
*     LINE OF THE SYNTAX AND IS THEREFORE AUTOMATIC-                  * IPDE0090
*     ALLY COMMITTED.  ERROR MESSAGE 3 IS                             * IPDE0100
*     "UNRECOGNIZABLE STMNT OR MISSPELLED KEYWORD".                   * IPDE0110
*                                                                     * IPDE0120
*     AS THIS LINE INDICATES, EACH                                    * IPDE0130
*     STATEMENT IS FIRST EXAMINED TO SEE WHETHER                      * IPDE0140
*     IT IS A DO STATEMENT.  IF IT IS NOT,                            * IPDE0150
*     IT IS EXAMINED TO SEE WHETHER IT IS AN                          * IPDE0160
*     ASSIGNMENT STATEMENT, THEN A KEYWORD                            * IPDE0170
*     STATEMENT, AND FINALLY, IF IT IS NONE                           * IPDE0180
*     OF THESE, ASSIGNMENT STATEMENT IS ATTEMPTED                     * IPDE0190
*     ONCE MORE USING A SLIGHTLY DIFFERENT                            * IPDE0200
*     SYNTAX WHICH ALLOWS THE ASSIGNMENT                              * IPDE0210
*     STATEMENT TO BEGIN WITH A NAME THAT                             * IPDE0220
*     IS LONGER THAN SIX CHARACTERS.                                  * IPDE0230
*     IF THE N ASSIGNMENT FORM IS TRIED, THE N                        * IPDE0240
*     OPERATOR  WILL ISSUE A "NAME TOO LONG" MESSAGE                  * IPDE0250
*     FOR INITIAL NAMES OF MORE THAN SIX CHARACTERS                   * IPDE0260
*     EVEN THOUGH ASSIGNMENT MAY NEVER BECOME COMMITTED.              * IPDE0270
*                                                                     * IPDE0280
DO      =   (  '0'  ...  )  D  (  D  .4.  )                            *IPDE0285
            (  ','  :  *140  $801  *33  )  N  *143  '='  *5            *IPDE0290
            <  N  |  USNZINT  >  *53  ','  :  <  N  |  /  USNZINT  >   *IPDE0295
            (  ','  <  N  |  /  USNZINT  >  )                           IPDE0300
*                                                                     * IPDE0310
*     DEFINES THE SYNTAX OF A DO STATEMENT.                           * IPDE0320
*     THE N-OPERATOR IS USED HERE INSTEAD OF                          * IPDE0330
*     THE M-OPERATOR EVEN THOUGH N WILL REQUIRE                       * IPDE0340
*     AT LEAST ONE VALID NAME BEFORE THE STATEMENT IS                 * IPDE0350
*     COMMITTED TO BEING A DO STATEMENT.  THIS                        * IPDE0360
*     IS PERMISSIBLE BECAUSE THE INITIAL DIGITS REQUIRED              * IPDE0370
*     BY THIS DEFINITION RULE OUT THE POSSIBILITY THAT                * IPDE0380
*     A KEYWORD STATEMENT WILL SATISFY THIS DEFINITION.               * IPDE0390
*     EACH PARAMETER OF THE DO IS A NAME OR AN                        * IPDE0400
*     UNSIGNED, NON-ZERO INTEGER.                                     * IPDE0410
*                                                                     * IPDE0420
*     THIS DEFINITION WILL ALMOST ALWAYS FAIL                         * IPDE0430
*     AT THE INITIAL DIGITS, FOR STATEMENTS THAT                      * IPDE0440
*     ARE NOT DO STATEMENTS.  HOWEVER, UNTIL                          * IPDE0450
*     THE FIRST COMMA IN THE PARAMETER LIST IS                        * IPDE0460
*     FOUND, IT COULD BE AN ASSIGNMENT STATEMENT                      * IPDE0470
*     SUCH AS     "DO3I=N**2".  THEREFORE                             * IPDE0480
*     THE STATEMENT CANNOT BE COMMITTED TO BEING                      * IPDE0490
*     A DO STATEMENT UNTIL THE COMMA IS                               * IPDE0500
*     ENCOUNTERED.                                                    * IPDE0510
*                                                                     * IPDE0512
*     SHOULD THERE BE A COMMA AFTER THE STATEMENT NUMBER,             * IPDE0514
*     ACTION CODE 801 CAUSES MESSAGE 140 TO BE ISSUED,                * IPDE0516
*     AND THE STATEMENT IS COMMITTED TO THIS LINE.                    * IPDE0518
*                                                                     * IPDE0520
USNZINT   =  *4   ¬'+'   ¬'-'   K   $100                                IPDE0530
*                                                                     * IPDE0540
*     DEFINES UNSIGNED, NONZERO INTEGER.  ACTION CODE                 * IPDE0550
*     100 AFTER THE K OPERATOR CHECKS TO SEE THAT                     * IPDE0560
*     THE NUMERIC CONSTANT FOUND BY THE K OPERATOR                    * IPDE0570
*     WAS A NON-ZERO INTEGER.                                         * IPDE0580
*                                                                     * IPDE0590
ASSIGNMENT   =   <  '=' :   |   '('  &=  <  N  ( ',' N ... ) ')=' :    *IPDE0600
               $200 |    SUB ( ',' SUB  ... )    ')='  :  $202    >  >  *IPDE0610
               *7    EXP                                                IPDE0620
*                                                                     * IPDE0630
*     DEFINES TWO CLASSES OF STATEMENTS                               * IPDE0640
*                                                                     * IPDE0650
*          A.  ARITHMETIC ASSIGNMENT STATEMENTS                       * IPDE0660
*                                                                     * IPDE0670
*          B.  ARITHMETIC STATEMENT FUNCTION DEFINITIONS                IPDE0680
*                                                                     * IPDE0690
*     A VALID SYMBOLIC NAME HAS BEEN FOUND BEFORE                     * IPDE0700
*     THIS LINE IS INVOKED, SO THE SYNTAX OF THE                      * IPDE0710
*     PART OF THE ASSIGNMENT BEFORE THE EQUALS                        * IPDE0720
*     SIGN IS ONE OF:                                                 * IPDE0730
*                                                                     * IPDE0740
*          1.  A NAME                                                 * IPDE0750
*                                                                     * IPDE0760
*          2.  A NAME FOLLOWED BY A PARENTHESIZED LIST OF NAMES       * IPDE0770
*                                                                     * IPDE0780
*          3.  A NAME FOLLOWED BY A PARENTHESIZED LIST OF             * IPDE0790
*              EXPRESSIONS, AT LEAST ONE OF WHICH IS NOT              * IPDE0800
*               SIMPLY A NAME                                         * IPDE0810
*                                                                     * IPDE0820
*     IN CASES 1 AND 3, THE STATEMENT IS IN                           * IPDE0830
*     CLASS A, SINCE CLASS B STATEMENTS MUST                          * IPDE0840
*     HAVE AT LEAST ONE NAME IN PARENTHESES                           * IPDE0850
*     BEFORE THE EQUALS SIGN, AND NO EXPRESSION                       * IPDE0860
*     EXCEPT A NAME IS PERMITTED IN THE PARENTHESES                   * IPDE0870
*     IN CLASS B STATEMENTS.  THEREFORE, IN                           * IPDE0880
*     CASE 3, ACTION CODE 202 IS USED TO CHECK                        * IPDE0890
*     FOR MORE THAN THREE SUBSCRIPTS.  ACTION                         * IPDE0900
*     CODE 202 ISSUES A "TOO MANY SUBSCRIPTS PRECEDE"                 * IPDE0910
*     MESSAGE IF THERE WERE MORE THAN THREE                           * IPDE0920
*     SUBSCRIPT EXPRESSIONS.                                          * IPDE0930
*                                                                     * IPDE0940
*     IN CASE 2, THE STATEMENT COULD BE IN                            * IPDE0950
*     EITHER CLASS A OR CLASS B, AND SO, IF                           * IPDE0960
*     MORE THAN THREE NAMES ARE PRESENT,                              * IPDE0970
*     A "POSSIBLY TOO MANY SUBSCRIPTS PRECEDE" MESSAGE                * IPDE0980
*     IS ISSUED BY ACTION CODE 200.                                   * IPDE0990
*                                                                     * IPDE1000
*     IF THE STATEMENT IS NOT CASE 1, IT                              * IPDE1010
*     IS SCANNED TO SEE WHETHER IT CONTAINS                           * IPDE1020
*     AN EQUALS SIGN SOMEWHERE TO THE RIGHT                           * IPDE1030
*     OF THE INITIAL NAME.  ASSIGNMENT                                * IPDE1040
*     FAILS IF AN EQUAL SIGN IS NOT FOUND.                            * IPDE1050
*     UNLESS A HOLLERITH FIELD CONTAINS THE                           * IPDE1060
*     EQUAL SIGN THAT SATISFIES THE SCANNING                          * IPDE1070
*     OPERATION, THIS TEST AVOIDS ANALYSIS                            * IPDE1080
*     OF A PARENTHESIZED FORM ( IN SUCH                               * IPDE1090
*     STATEMENTS AS FORMAT AND READ ) BY THE                          * IPDE1100
*     ASSIGNMENT SYNTACTIC LINE, WHEN THERE IS                        * IPDE1110
*     NO POSSIBITITY THAT THE STATEMENT IS AN ASSIGNMENT.             * IPDE1120
*     WHEN AN EQUALS SIGN IS FOUND IN THE                             * IPDE1130
*     PROPER PLACE, THE STATEMENT IS COMMITTED.                       * IPDE1140
*                                                                     * IPDE1150
*     THE SYNTAX TO THE RIGHT OF THE EQUALS                           * IPDE1160
*     IS THE SAME FOR CLASSES A AND B, EVEN                           * IPDE1170
*     THOUGH CLASS B DOES NOT ALLOW REFERENCES                        * IPDE1180
*     TO SUBSCRIPTED VARIABLES IN THE EXPRESSION.                     * IPDE1190
*     THIS IS BECAUSE THE SYNTAX CHECKER DOES NOT                     * IPDE1200
*     HAVE THE INFORMATION THAT WOULD ENABLE IT TO                    * IPDE1210
*     DETERMINE THAT A NAME FOLLOWED BY A                             * IPDE1220
*     PARENTHESIZED LIST OF EXPRESSIONS                               * IPDE1230
*     WAS AN ARRAY ELEMENT REFERENCE AND                              * IPDE1240
*     NOT A FUNCTION REFERENCE.  THE                                  * IPDE1250
*     SYNTAX CHECKER WOULD HAVE TO SAVE                               * IPDE1260
*     INFORMATION FROM DIMENSION AND OTHER                            * IPDE1270
*     ARRAY-DECLARING STATEMENTS TO MAKE                              * IPDE1280
*     THE DISTINCTION, AND THE SYNTAX CHECKER                         * IPDE1290
*     DOES NOT SAVE SUCH INFORMATION.                                 * IPDE1300
*                                                                     * IPDE1310
SUB     =   *67   <   (  USNZINT  '*'  )  N   (   <  '+' | '-' >  /    *IPDE1320
            USNZINT   )   |   USNZINT   >                               IPDE1330
*                                                                     * IPDE1340
*     DEFINES SUBSCRIPT EXPRESSION.  THIS FORM IS                     * IPDE1350
*     USED FOR THE EXPRESSIONS WHEREVER IT IS                         * IPDE1360
*     CERTAIN THAT A NAME FOLLOWED BY A PARENTHESIZED                 * IPDE1370
*     LIST OF EXPRESSIONS IS AN ARRAY ELEMENT REFERENCE               * IPDE1380
*     AND NOT A FUNCTION REFERENCE, AS IN                             * IPDE1390
*     INPUT/OUTPUT LISTS.                                             * IPDE1400
*                                                                     * IPDE1410
EXP     =   (   <  '+' |   '-'  >  )  OPERAND  *55 (   +ARITHOP  /     *IPDE1420
            OPERAND   ...    )                                          IPDE1430
*                                                                     * IPDE1440
*     DEFINES ARITHMETIC EXPRESSION.                                  * IPDE1450
*                                                                     * IPDE1460
ARITHOP =   "  '+' 0    '-' 0    '/' 0    '**' 0    '*' 0    "          IPDE1470
*                                                                     * IPDE1480
*     TABLE OF THE ARITHMETIC OPERATORS.  THE                         * IPDE1490
*     DOUBLE ASTERISK MUST PRECEDE THE SINGLE                         * IPDE1500
*     ASTERISK SO THAT A SPURIOUS MATCH ON                            * IPDE1510
*     "SINGLE ASTERISK" WILL NOT OCCUR WHEN THE                       * IPDE1520
*     SOURCE STATEMENT CONTAINS A DOUBLE                              * IPDE1530
*     ASTERISK.                                                       * IPDE1540
*                                                                     * IPDE1550
OPERAND =   <    K    |    N    (   '('  /   *7   EXP  ( ','  /   EXP  *IPDE1560
            ... )   $200  *12  ')' ) |   '(' / *7  EXP   *12  ')'  >    IPDE1570
*                                                                     * IPDE1580
*     DEFINES ARITHMETIC OPERANDS FOR USE IN                          * IPDE1590
*     ARITHMETIC EXPRESSIONS.                                         * IPDE1600
*                                                                     * IPDE1610
KEYWORD =  " 'BACK' BACKSPACE   'CALL' CALL        'COMM' COMMON       *IPDE1620
             'CONT' CONTINUE    'DEFI' DEFINEFILE  'DIME' DIMENSION    *IPDE1630
             'DOUB' DOUBLE      'ENDF' ENDFILE     'END' END           *IPDE1640
             'EQUI' EQUIVALENCE 'EXTE' EXTERNAL    'FIND' FIND         *IPDE1650
             'FORM' FORMAT      'FUNC' FUNCTION    'GOTO' GOTO         *IPDE1660
             'IF' IF            'INTE' INTEGER    'PAUS' PAUSE        *IPDE1670
             'READ' READ        'REAL' REAL       'RETU' RETURN       *IPDE1680
             'REWI' REWIND      'SUBR' SUBROUTINE 'STOP' STOP         *IPDE1690
             'WRIT' WRITE       "                                      IPDE1700
*                                                                     * IPDE1710
*     TABLE OF ALL THE KEYWORDS THAT MAY APPEAR                       * IPDE1720
*     AT THE BEGINNING OF A STATEMENT.  FOR EACH OF THE ENTRIES       * IPDE1730
*     A MATCH WITH THE LITERAL RESULTS IN A                           * IPDE1740
*     TRANSFER TO THE APPROPRIATE SYNTACTIC LINE.                     * IPDE1750
*                                                                     * IPDE1860
BACKSPACE  =  'SPACE'  :  DSREFNO                                       IPDE1870
*                                                                     * IPDE1880
*     DEFINES THE BACKSPACE STATEMENT.                                * IPDE1890
*                                                                     * IPDE1900
DSREFNO    =  *27  <  N  |  K  /  $105  >                               IPDE1910
*                                                                     * IPDE1920
*     DEFINES DATA SET REFERENCE NUMBER.                              * IPDE1930
*     ACTION CODE 105 ISSUES AN APPROPRIATE                           * IPDE1940
*     MESSAGE IF THE K ALTERNATIVE ENCOUNTERS                         * IPDE1950
*     ANY NUMERIC CONSTANT OTHER THAN A NON-ZERO                      * IPDE1960
*     INTEGER LESS THAN OR EQUAL TO 99.                               * IPDE1970
*                                                                     * IPDE1980
CALL     =  :    *33    N    (    '('   /    *46    EXP     (  ','    / *IPDE1990
                 EXP  ...     )     *13    ')'    )                     IPDE2000
*                                                                     * IPDE2010
*     DEFINES THE CALL STATEMENT.                                     * IPDE2020
*                                                                     * IPDE2030
COMMON   =  'ON' : *33 N  ( DECLARATOR )     (  ','  /  N             *IPDE2040
            (  DECLARATOR  )  ...  )                                    IPDE2050
*                                                                     * IPDE2060
*     DEFINES THE COMMON STATEMENT.                                   * IPDE2070
*                                                                     * IPDE2080
DECLARATOR = *37 '('  / USNZINT  ( ',' / $201 USNZINT ... ) *12 ')'     IPDE2090
*                                                                     * IPDE2100
*     DEFINES ARRAY DECLARATORS WITH CONSTANT                         * IPDE2110
*     DECLARATORS.                                                    * IPDE2120
*                                                                     * IPDE2130
CONTINUE = 'INUE' :                                                     IPDE2140
*                                                                     * IPDE2150
*     DEFINES THE CONTINUE STATEMENT                                  * IPDE2160
*                                                                     * IPDE2170
DEFINEFILE =  'NEFILE' :  *27  K  $105  *31  '('  USNZINT             *IPDE2180
              *53 ','  USNZINT  ','  *63 < 'L' | 'E' | 'U' >          *IPDE2190
              *53 ','  *33 N  *13 ')'  (  ',' /  *27 K $105  *31      *IPDE2200
              '('  USNZINT  *53  ','  USNZINT  ','  *63               *IPDE2210
              < 'L' | 'E' | 'U' >  *53 ','  *33 N  *13 ')'  ...  )      IPDE2220
*                                                                     * IPDE2230
*     DEFINES THE DEFINE FILE STATEMENT.  IN                          * IPDE2240
*     THIS STATEMENT, THE DATA SET REFERENCE                          * IPDE2250
*     NUMBER CANNOT BE A SYMBOLIC NAME, SO                            * IPDE2260
*     THE K OPERATOR FOLLOWED BY ACTION CODE 105                      * IPDE2270
*     IS USED WHERE DATA SET REFERENCE NUMBERS ARE                    * IPDE2280
*     REQUIRED.  THE FORM OF THE BASIC ELEMENT                        * IPDE2290
*     OF THIS STATEMENT IS GIVEN ON THE FIRST                         * IPDE2300
*     TWO AND-A-HALF LINES.  THE LAST TWO                             * IPDE2310
*     AND-A-HALF LINES DESCRIBE THE OPTIONAL                          * IPDE2320
*     REPETITION OF THIS ELEMENT FOLLOWING A COMMA.                   * IPDE2330
*                                                                     * IPDE2340
DIMENSION = 'NSION' :  *33 N  DECLARATOR  ( ','  /  N  DECLARATOR     *IPDE2350
            ...   )                                                     IPDE2360
*                                                                     * IPDE2370
*     DEFINES THE DIMENSION STATEMENT.  SINCE                         * IPDE2380
*     THE LINE IS COMMITTED AFTER THE LITERAL IS                      * IPDE2390
*     MATCHED, THE "ARRAY DIMENSIONS EXPECTED" MESSAGE                * IPDE2400
*     ON THE DECLARATOR LINE WILL BE ISSUED IF                        * IPDE2410
*     A DECLARATOR IS MISSING.  THE "NAME EXPECTED"                   * IPDE2420
*     MESSAGE ON THIS LINE THEREFORE APPLIES TO THE                   * IPDE2430
*     ENTIRE LINE.                                                    * IPDE2440
*                                                                     * IPDE2450
DOUBLE  =       'LEPRECISION'  <  'FUNC'  FUNCTION  |  TYPE  >          IPDE2460
*                                                                     * IPDE2470
*     TABLE OF TRANSFERS FOR STATEMENTS BEGINNING                     * IPDE2480
*     WITH 'DOUBLE PRECISION'.                                        * IPDE2490
*                                                                     * IPDE2500
TYPE     =   :     *33  N    ( DECLARATOR )    (    ','   /    N      *IPDE2510
                 ( DECLARATOR )    ...    )                             IPDE2520
*                                                                     * IPDE2530
*     DEFINES ALL THE TYPE-STATEMENTS.  THIS                          * IPDE2540
*     DEFINITION IS USED AFTER THE KEYWORD AT                         * IPDE2550
*     THE BEGINNING OF THE TYPE-STATEMENT HAS                         * IPDE2560
*     BEEN MATCHED IN THE APPROPRIATE TABLE.                          * IPDE2570
*                                                                     * IPDE2580
ENDFILE =  'ILE'  :   DSREFNO                                           IPDE2590
*                                                                     * IPDE2600
*     DEFINES THE ENDFILE STATEMENT.                                  * IPDE2610
*                                                                     * IPDE2612
```

```
END    =  $800   :  $300                                                IPDE2614
*                                                                     * IPDE2616
*     DEFINES THE END LINE.  ACTION CODE 800 PRODUCES AN F            * IPDE2618
*     IF THERE ARE ANY CHARACTERS OTHER THAN BLANKS AFTER THE         * IPDE2620
*     CHARACTERS 'END' WHICH CAUSED NESTING TO THIS LINE.             * IPDE2622
*     IF THERE WERE NO NON-BLANK CHARACTERS AFTER 'END', ACTION       * IPDE2624
*     CODE 800 PRODUCES A T, CAUSING ACTION CODE 300 TO DETECT AND    * IPDE2626
*     DIAGNOSE ANY STATEMENT LABEL OR CONTINUATION FIELD ERRORS.      * IPDE2628
*                                                                     * IPDE2629
EQUIVALENCE = 'VALENCE'  :  *30  '('  *33  N  (  DECLARATOR  )         *IPDE2630
            *53  ','  *33  N  (  DECLARATOR  )   (  ','  /   N         *IPDE2640
            ( DECLARATOR )  ... ) *12 ')'  (  ',' / *30 '('   *33  N   *IPDE2650
            ( DECLARATOR )  *53  ','  *33  N  ( DECLARATOR )  (  ','   *IPDE2660
            /   N   ( DECLARATOR )  ... )   *13  ')'   ...  )           IPDE2670
*                                                                     * IPDE2680
*     DEFINES THE EQUIVALENCE STATEMENT.                              * IPDE2690
*     AS IN THE DEFINEFILE DEFINITION, THE FIRST                      * IPDE2700
*     TWO AND-A-HALF LINES OF THIS DEFINITION                         * IPDE2710
*     DESCRIBE THE BASIC FORM.                                        * IPDE2720
*                                                                     * IPDE2730
EXTERNAL  =  'RNAL'  :  *33  N  (  ','  /  N  ...  )                    IPDE2740
*                                                                     * IPDE2750
*     DEFINES THE EXTERNAL STATEMENT.                                 * IPDE2760
*                                                                     * IPDE2770
FIND     =  :  *30  '('  DSREFNO  *61  ''''  *7  INTEGEXP  *13 ')'      IPDTEE
*                                                                     * IPDE2790
*     DEFINES THE FIND STATEMENT.  THE FOUR                           * IPDE2800
*     QUOTATION MARKS REPRESENT A LITERAL CONSISTING                  * IPDE2810
*     OF ONE QUOTE IN THE SOURCE.                                     * IPDE2820
*                                                                     * IPDTEE
*     PTM2770 FIX (RELEASE 19) CHANGED      EXP  TO INTEGEXP SO ******* IPDTEE
*     THAT REAL CONSTANTS WILL BE FOUND IN ERROR.  ******************** IPDTEE
*                                                                     * IPDTEE
*                                                                     * IPDE2830
FORMAT  =  'AT'  :  $301   *30  '('  *77  (  '/'  ...  )  (  GROUP     *IPDE2840
          (  <  ','  /  GROUP  |  '/'  (  '/'  ...  )  GROUP  >        *IPDE2850
          ...     )     (  '/'  ...  )     )    ')'                     IPDE2860
*                                                                     * IPDE2870
*     DEFINES THE FORMAT STATEMENT.  ESSENTIALLY,                     * IPDE2880
*     THE DEFINITION IS A PARENTHESIZED LIST OF                       * IPDE2890
*     GROUPS.  (GROUP IS DEFINED ON ANOTHER LINE)                     * IPDE2900
*     EACH DELIMITER IN THE LIST IS EITHER A COMMA                    * IPDE2910
*     OR ANY NUMBER OF SLASHES.  OPTIONALLY, THERE                    * IPDE2920
*     MAY BE ANY NUMBER OF SLASHES BEFORE THE                         * IPDE2930
*     FIRST GROUP IN THE LIST, OR AFTER THE LAST                      * IPDE2940
*     GROUP IN THE LIST, OR BOTH.  THERE                              * IPDE2950
*     NEED NOT BE ANY GROUPS AT ALL.  THE                             * IPDE2960
*     LAST SET OF OPTIONAL SLASHES IS INCLUDED IN                     * IPDE2970
*     THE OPTIONAL PARENTHESES FOR THE LIST                           * IPDE2980
*     OF GROUPS BECAUSE, IF THERE ARE NO                              * IPDE2990
*     GROUPS, THE FIRST SET OF OPTIONAL SLASHES                       * IPDE3000
*     WILL HAVE MATCHED ALL THE VALID CHARACTERS                      * IPDE3010
*     WITHIN THE SOURCE'S PARENTHESES.  THE                           * IPDE3020
*     MESSAGE ISSUED WHEN A RIGHT PARENTHESIS IS                      * IPDE3030
*     NOT FOUND IS "DELIMITER MISSING OR INVALID                      * IPDE3040
*     FORMAT CODE" SINCE ANY FAILURE TO MATCH                         * IPDE3050
*     THE RIGHT PARENTHESIS LITERAL IS PROBABLY                       * IPDE3060
*     DUE TO ONE OF THESE CAUSES.                                     * IPDE3070
*                                                                     * IPDE3080
GROUP     =  <  FIELDESCR  |  (  $700  )  '('  /  (  <  '/'  |         *IPDE3280
             (  $700  )  '('  /  *69  $801  >  ...  )                  *IPDE3285
             (  FIELDESCR  (  <  ','  /  FIELDESCR  |  '/'  (          *IPDE3290
             <  '/'  |  (  $700  )  '('  /  *69  $801  >  ...  )       *IPDE3295
             FIELDESCR  >  ...  )  (  '/'  ...  )  )  ')'  >            IPDE3300
*                                                                     * IPDE3310
*     DEFINES GROUP FOR USE IN THE FORMAT DEFINITION.                 * IPDE3311
*     A GROUP IS EITHER A FIELD DESCRIPTOR OR                         * IPDE3312
*     ANOTHER FORM THAT IS ESSENTIALLY THE SAME AS A                  * IPDE3313
*     FORMAT.  THE DIFFERENCES BETWEEN FORMAT AND                     * IPDE3314
*     THE SECOND FORM ARE  1) THE SECOND FORM OF                      * IPDE3315
*     GROUP MAY HAVE A REPEAT COUNT BEFORE THE                        * IPDE3316
*     INITIAL LEFT PARENTHESIS, AND  2) THE ITEMS                     * IPDE3317
*     IN THE PARENTHESIZED LIST ARE EACH FIELDESCR                    * IPDE3318
*     INSTEAD OF GROUP.  THE SECOND DIFFERENCE                        * IPDE3319
*     IS NECESSARY TO AVOID ALLOWING AN INDEFINITE NUMBER             * IPDE3320
*     OF LEVELS OF NESTING OF PARENTHESES IN FORMAT                   * IPDE3321
*     STATEMENTS.  FORTRAN ALLOWS ONLY ONE LEVEL                      * IPDE3322
*     OF NESTING INSIDE THE PARENTHESES WHICH ENCLOSE                 * IPDE3323
*     THE ENTIRE FORMAT SPECIFICATION.                                * IPDE3324
*                                                                     * IPDE3325
*     ACTION CODE 700 ADVANCES THE SOURCE POINTER PAST                * IPDE3330
*     THE GROUP REPEAT COUNT WHEN ONE IS PRESENT.                     * IPDE3340
*                                                                     * IPDE3350
*     ACTION CODE 801 IS USED TO ISSUE A MESSAGE                      * IPDE3360
*     DIAGNOSING TOO MANY LEVELS OF PARENTHESES IF ANY                * IPDE3370
*     LEFT PARENTHESIS IS FOUND WITHIN THE PARENTHESES                * IPDE3380
*     WHICH ENCLOSE THE REST OF THE SECOND ALTERNATIVE.               * IPDE3390
*                                                                      * IPDE3400
FIELDESCR  =  <  C  |  $700  'X'  |  (  $700  )                         *IPDE3410
              <  <  'E'  |  'F'  |  'D'  >  /  $700  *80  '.'  $701     *IPDE3420
              |  <  'I'  |  'A'  >  /  $700  >  |  H  |  'T' / $700     *IPDE3430
              |  ( '-' ) < $700 | '0' ( '0' ... ) >  'P'  ( $700 )      *IPDE3440
              <  'E'  |  'F'  |  'D'  >  /  $700  *80  '.'  $701  >      IPDE3450
*                                                                      * IPDE3460
*     DEFINES ALL THE FIELD DESCRIPTORS WHICH MAY                      * IPDE3470
*     APPEAR IN A FORMAT STATEMENT.                                    * IPDE3480
*                                                                      * IPDE3490
FUNCTION  =   'TION' :    *32   N   DUMMYARGS                            IPDE3500
*                                                                      * IPDE3510
*     DEFINES FUNCTION STATEMENTS, INCLUDING                           * IPDE3520
*     THOSE WHICH BEGIN WITH ONE OF THE                                * IPDE3530
*     TYPE DECLARATORS REAL, INTEGER, OR DOUBLE                        * IPDE3540
*     PRECISION.   IF ONE OF THESE TYPE                                * IPDE3550
*     DECLARATORS PRECEDES 'FUNCTION', IT HAS                          * IPDE3560
*     BEEN MATCHED IN THE APPROPRIATE TABLE.                           * IPDE3570
*     THEREFORE, THIS LINE DOES NOT NEED TO MATCH                      * IPDE3580
*     ANY OF THE TYPE KEYWORDS.                                        * IPDE3590
*                                                                      * IPDE3600
DUMMYARGS =   *35  '(' /    N     (  ','  /   N   ...   )    *13  ')'    IPDE3610
*                                                                      * IPDE3620
*     DEFINES THE LIST OF DUMMY ARGUMENTS,                             * IPDE3630
*     INCLUDING THE PARENTHESES WHICH ENCLOSE                          * IPDE3640
*     THE LIST, IN A FUNCTION STATEMENT.                               * IPDE3650
*                                                                      * IPDE3660
GOTO     =  :  <  '('  /  *43  S  (  ','  /  S  ...  )  *13  ')'        *IPDE3670
               *52  ','  *33  N   |   /  *43  S  >                       IPDE3680
*                                                                      * IPDE3690
*     DEFINES THE TWO KINDS OF GOTO STATEMENT.                         * IPDE3700
*     THESE ARE DEFINED IN THE ORDER:                                  * IPDE3710
*     COMPUTED GOTO,  UNCONDITIONAL GOTO.                              * IPDE3720
*     THIS ORDERING ALLOWS A COMMIT                                    * IPDE3730
*     TO PRECEDE THE S OPERATOR IN THE DEFINITION OF                   * IPDE3740
*     THE UNCONDITIONAL GOTO.                                          * IPDE3750
*                                                                      * IPDE3760
IF        =   :   *31  '('  *7   EXP   *13  ')'   *43   S   *53  ','    *IPDE3770
              *43  S   *53  ','    *43   S                               IPDE3780
*                                                                      * IPDE3790
*     DEFINITION OF THE ARITHMETIC IF STATEMENT.                       * IPDE3800
*                                                                      * IPDE3810
INTEGER   =   'GER'  <  'FUNC'  FUNCTION  |  TYPE  >                     IPDE3820
*                                                                      * IPDE3830
*     TABLE OF TRANSFERS FOR STATEMENTS BEGINNING WITH                 * IPDE3840
*     'INTEGER'.                                                       * IPDE3850
*                                                                      * IPDE3860
PAUSE     =   'E'  :  (  D  .5.  )  *129  $800                           IPDE3870
*                                                                      * IPDE3880
*     DEFINES THE PAUSE STATEMENT.                                     * IPDE3890
*                                                                      * IPDE3950
READ      =   :   *30  '('   DSREFNO   (  ''''  /  *7   INTEGEXP   )    *IPDTEE
              (  ','  /  *42  S  )  *13  ')'  (  IOLIST  )               IPDE3970
*                                                                      * IPDE3980
*     DEFINES THE FORM OF EITHER READ                                  * IPDE3990
*     OR WRITE AFTER THE KEYWORD.  THIS DEFINITION                     * IPDE4000
*     ENCOMPASSES SEQUENTIAL OR DIRECT ACCESS,                         * IPDE4010
*     FORMATTED OR UNFORMATTED, READ AND WRITE                         * IPDE4020
*     STATEMENTS.                                                      * IPDE4030
*     THE IOLIST IS OPTIONAL IN ALL FORMS.                             * IPDE4040
*                                                                      * IPDTEE
*    PTM2770 FIX (RELEASE 19)  CHANGED        EXP TO INTEGEXP SO ******* IPDTEE
*    THAT REAL CONSTANTS WILL BE FOUND IN ERROR.  ********************** IPDTEE
*                                                                      * IPDE4050
IOLIST  =   *58  <   IOVAR |  PARENLIST    >     (   ','      /         *IPDE4060
            <     IOVAR   |    PARENLIST    >     ...        )           IPDE4070
*                                                                      * IPDE4080
*     DEFINES AN INPUT/OUTPUT LIST.                                    * IPDE4090
*                                                                      * IPDE4100
PARENLIST =   '(' / *58 <  IOVAR   |   PARENLIST  >   (   ','           *IPDE4110
            /  *32  <  IOVAR (  '='   /  $602  <  N  |  /               *IPDE4120
            USNZINT > *52 ','  <  N  |  /  USNZINT >   (   ','  <       *IPDE4130
            N  |   /    USNZINT   >    )    $603  )   |                 *IPDE4140
            PARENLIST >   ...       )      *12  ')'                      IPDE4150
*                                                                      * IPDE4160
*     DEFINES THE PARENTHESIZED LIST THAT MAY BE                       * IPDE4170
*     A MEMBER OF AN INPUT/OUTPUT LIST.  THIS                          * IPDE4180
*     COMPLICATED LOOKING DEFINITION IS BASICALLY                      * IPDE4190
*     JUST:                                                            * IPDE4200
*                                                                      * IPDE4210
*          PARENLIST = '('  <  IOVAR  |  PARENLIST   >                 * IPDE4220
*                      ( ',' / < IOVAR  |  PARENLIST > ... )   ')'     * IPDE4230
*                                                                      * IPDE4240
*     HOWEVER, THERE IS A LENGTHY OPTION AFTER                         * IPDE4250
*     THE SECOND OCCURRENCE OF IOVAR.  THE OPTION                      * IPDE4260
*     DESCRIBES THE SYNTAX FOUND WHEN THE SOURCE                        * IPDE4270
*     CONTAINS AN IMPLIED DO.  THIS OPTION                              * IPDE4280
*     BEGINS WITH THE LEFT PARENTHESIS ON THE SECOND                    * IPDE4290
*     LINE AND ENDS WITH THE LAST RIGHT                                 * IPDE4300
*     PARENTHESIS ON THE FOURTH LINE.                                   * IPDE4310
*                                                                       * IPDE4320
*     THE FIRST OCCURRENCE OF IOVAR DOES NOT                            * IPDE4330
*     HAVE THE OPTION, BECAUSE AN IMPLIED                               * IPDE4340
*     DO SPECIFICATION MAY NOT BE THE FIRST                             * IPDE4350
*     ITEM INSIDE A PARENTHESIS.  IF AN EQUAL                           * IPDE4360
*     SIGN IS ENCOUNTERED AFTER SOME INPUT/OUTPUT                       * IPDE4370
*     VARIABLE AFTER THE FIRST VARIABLE OR PARENTHESIZED                * IPDE4380
*     LIST, THE OPTION IS COMMITTED.  ACTION CODE                       * IPDE4390
*     602 IMMEDIATELY CHECKS THE FLAG SET BY                            * IPDE4400
*     ACTION CODES 600 AND 601 TO SEE WHETHER                           * IPDE4410
*     THE VARIABLE PRECEDING THE EQUAL SIGN                             * IPDE4420
*     WAS SUBSCRIPTED.  IF IT WAS, AN APPROPRIATE                       * IPDE4430
*     ERROR MESSAGE IS ISSUED.  THEN THE                                * IPDE4440
*     PARAMETERS OF THE IMPLIED DO ARE CHECKED.                         * IPDE4450
*     THERE MUST BE A PARENTHESIS IMMEDIATELY                           * IPDE4460
*     AFTER AN IMPLIED DO SPECIFICATION.  ACTION                        * IPDE4470
*     CODE 603 CHECKS FOR THIS PARENTHESIS AND ISSUES                   * IPDE4480
*     AN APPROPRIATE MESSAGE IF IT IS ABSENT, BUT                       * IPDE4490
*     DOES NOT ADVANCE THE SOURCE POINTER, ALLOWING                     * IPDE4500
*     THE RIGHT PARENTHESIS LITERAL AT THE                              * IPDE4510
*     END OF THE DEFINITION TO BE                                       * IPDE4520
*     MATCHED IF THE RIGHT PARENTHESIS IS                               * IPDE4530
*     PRESENT.  ANY OTHER METHOD OF CHECKING                            * IPDE4540
*     FOR THE RIGHT PARENTHESIS WOULD ADVANCE                           * IPDE4550
*     THE SOURCE POINTER AND CAUSE A FAILURE                            * IPDE4560
*     ON THE RIGHT PARENTHESIS LITERAL AT THE                           * IPDE4570
*     END OF THE DEFINITION.                                            * IPDE4580
*                                                                       * IPDE4590
IOVAR   =     N   $600   (    '('   /   $601      SUB      (    ','    /     *IPDE4600
              $201  SUB  ...  )  *12  ')'  )                               IPDE4610
*                                                                       * IPDE4620
*     DEFINES THE ITEMS WHICH MAKE UP INPUT/OUTPUT                      * IPDE4630
*     LISTS.  ACTION CODES 600 AND 601 ARE USED TO                      * IPDE4640
*     SET A FLAG THAT CAN BE TESTED LATER TO DETERMINE                  * IPDE4650
*     WHETHER THE LAST INPUT/OUTPUT VARIABLE                            * IPDE4660
*     WAS SUBSCRIPTED.  ACTION CODE 600 SETS                            * IPDE4670
*     THIS FLAG TO "UNSUBSCRIPTED", AND ACTION CODE                     * IPDE4680
*     601 SETS IT TO "SUBSCRIPTED".  ACTION CODE                        * IPDE4690
*     201 TESTS FOR TOO MANY SUBSCRIPTS.                                * IPDE4700
*                                                                       * IPDE4710
REAL    =     <  'FUNC'  FUNCTION  |  TYPE  >                              IPDE4720
*                                                                       * IPDE4730
*     TABLE OF TRANSFERS FOR STATEMENTS BEGINNING WITH                  * IPDE4740
*     'REAL'.                                                           * IPDE4750
*                                                                       * IPDE4760
RETURN  =     'RN'  :                                                      IPDE4770
*                                                                       * IPDE4780
*     DEFINES THE RETURN STATEMENT.                                     * IPDE4790
*                                                                       * IPDE4800
REWIND  =     'ND'  :   DSREFNO                                            IPDE4810
*                                                                       * IPDE4820
*     DEFINES THE REWIND STATEMENT.                                     * IPDE4830
*                                                                       * IPDE4840
SUBROUTINE  =  'OUTINE'  :   *32   N   (  DUMMYARGS  )                     IPDE4850
*                                                                       * IPDE4860
*     DEFINES THE SUBROUTINE STATEMENT.                                 * IPDE4870
*                                                                       * IPDE4880
STOP      =   :  (  D  .5.  )  *129  $800                                  IPDE4890
*                                                                       * IPDE4900
*     DEFINES THE STOP STATEMENT.                                       * IPDE4910
*                                                                       * IPDE4970
WRITE   =   'E'  READ                                                      IPDE4980
*                                                                       * IPDE4990
*     DEFINES THE WRITE STATEMENT.                                      * IPDE5000
*                                                                       * IPDE5010
INTEGEXP =     (  <  '+'  |  '-'  >  )   OPERANDI  *55                     *IPDTEE
               (    +ARITHOP    /  OPERANDI    ...   )  ¬'.'                IPDTEE
*                                                                       * IPDTEE
*    THIS STATEMENT IS THE SAME AS      EXP  EXCEPT THAT A REAL         * IPDTEE
*    CONSTANT WILL NOT SATISFY THE DEFINITION.                          * IPDTEE
*    THIS STATEMENT WAS ADDED FOR PTM2770 FIX (RELEASE 19).  ********** IPDTEE
*                                                                       * IPDTEE
OPERANDI =     <   K / $102   |    N   (   '('  /   *7   EXP              *IPDTEE
               ( ',' /       EXP       ...  )   $200  *12  ')' ) |  '('  *IPDTEE
               / *7    EXP      *12  ')'    >                              IPDTEE
*                                                                       * IPDTEE
*    THIS STATEMENT IS THE SAME AS OPERAND   (BUT DEFINES OPERANDS      * IPDTEE
*    FOR INTEGEXP) EXCEPT THAT THE K OPERATOR WILL FAIL IF A            * IPDTEE
*    CONSTANT IS NOT AN INTEGER.                                        * IPDTEE
*    THIS STATEMENT WAS ADDED FOR PTM2770 FIX (RELEASE 19).  ********** IPDTEE
SYNTAX END                                                                 IPDE5020
```

METALANGUAGE USED FOR IPDAGH SYNTAX DEFINITION TABLE

```
AGH 3  53IPDAGH, FORTRAN IV LEVELS G, G1, H AND TSO DEFINITION           5120
SYNTAX IPDAGH                                                            IPDG0010
IPDAGH  =   *3 < 'DO' DO | M ASSIGNMENT | KEYWORD | N ASSIGNMENT >       IPDG0030
*                                                                      * IPDG0035
*      THIS LINE DETERMINES THE OVERALL STRATEGY                       * IPDG0040
*      IN SCANNING STATEMENTS.  ERROR MESSAGE                          * IPDG0050
*      3 IS ISSUED IF THE STATEMENT IS NONE OF                         * IPDG0060
*      THE ALTERNATIVES, SINCE THIS IS THE FIRST                       * IPDG0070
*      LINE OF THE SYNTAX AND IS THEREFORE AUTOMATIC-                  * IPDG0080
*      ALLY COMMITTED.  ERROR MESSAGE 3 IS                             * IPDG0090
*      "UNRECOGNIZABLE STMNT OR MISSPELLED KEYWORD".                   * IPDG0100
*                                                                      * IPDG0110
*      AS THIS LINE INDICATES, EACH                                    * IPDG0120
*      STATEMENT IS FIRST EXAMINED TO SEE WHETHER                      * IPDG0130
*      IT IS A DO STATEMENT.  IF IT IS NOT,                            * IPDG0140
*      IT IS EXAMINED TO SEE WHETHER IT IS AN                          * IPDG0150
*      ASSIGNMENT STATEMENT, THEN A KEYWORD                            * IPDG0160
*      STATEMENT, AND FINALLY, IF IT IS NONE                           * IPDG0170
*      OF THESE, ASSIGNMENT STATEMENT IS ATTEMPTED                     * IPDG0180
*      ONCE MORE USING A SLIGHTLY DIFFERENT                            * IPDG0190
*      SYNTAX WHICH ALLOWS THE ASSIGNMENT                              * IPDG0200
*      STATEMENT TO BEGIN WITH A NAME THAT                             * IPDG0210
*      IS LONGER THAN SIX CHARACTERS.                                  * IPDG0220
*      IF THE N ASSIGNMENT FORM IS TRIED, THE N                        * IPDG0222
*      OPERATOR  WILL ISSUE A "NAME TOO LONG" MESSAGE                  * IPDG0224
*      FOR INITIAL NAMES OF MORE THAN SIX CHARACTERS                   * IPDG0226
*      EVEN THOUGH ASSIGNMENT MAY NEVER BECOME COMMITTED.              * IPDG0228
*                                                                      * IPDG0230
DO      =    (  '0'  ...  )  D  (  D  .4.  )                           *IPDG0235
             (  ','  :  *140  $801  *33  )  N  *143  '='  *5           *IPDG0240
             <  N  |  USNZINT  >  *53  ','  :  <  N  |  /  USNZINT  >  *IPDG0245
             (  ','  <  N  |  /  USNZINT  >  )                           IPDG0250
*                                                                      * IPDG0260
*      DEFINES THE SYNTAX OF A DO STATEMENT.                           * IPDG0270
*      THE N-OPERATOR IS USED HERE INSTEAD OF                          * IPDG0280
*      THE M-OPERATOR EVEN THOUGH N WILL REQUIRE                       * IPDG0290
*      AT LEAST ONE VALID NAME BEFORE THE STATEMENT IS                 * IPDG0300
*      COMMITTED TO BEING A DO STATEMENT.  THIS                        * IPDG0310
*      IS PERMISSIBLE BECAUSE THE INITIAL DIGITS REQUIRED              * IPDG0320
*      BY THIS DEFINITION RULE OUT THE POSSIBILITY THAT                * IPDG0330
*      A KEYWORD STATEMENT WILL SATISFY THIS DEFINITION.               * IPDG0340
*      EACH PARAMETER OF THE DO IS A NAME OR AN                        * IPDG0350
*      UNSIGNED, NON-ZERO INTEGER.                                     * IPDG0360
*                                                                      * IPDG0370
*      THIS DEFINITION WILL ALMOST ALWAYS FAIL                         * IPDG0380
*      AT THE INITIAL DIGITS, FOR STATEMENTS THAT                      * IPDG0390
*      ARE NOT DO STATEMENTS.  HOWEVER, UNTIL                          * IPDG0400
*      THE FIRST COMMA IN THE PARAMETER LIST IS                        * IPDG0410
*      FOUND, IT COULD BE AN ASSIGNMENT STATEMENT                      * IPDG0420
*      SUCH AS     "DO3I=N**2".  THEREFORE                             * IPDG0430
*      THE STATEMENT CANNOT BE COMMITTED TO BEING                      * IPDG0440
*      A DO STATEMENT UNTIL THE COMMA IS                               * IPDG0450
*      ENCOUNTERED.                                                    * IPDG0460
*                                                                      * IPDG0462
*      SHOULD THERE BE A COMMA AFTER THE STATEMENT NUMBER,             * IPDG0464
*      ACTION CODE 801 CAUSES MESSAGE 140 TO BE ISSUED,                * IPDG0466
*      AND THE STATEMENT IS COMMITTED TO THIS LINE.                    * IPDG0468
*                                                                      * IPDG0470
USNZINT   =  *4   ¬'+'   ¬'-'   K   $100                                 IPDG0480
*                                                                      * IPDG0490
*      DEFINES UNSIGNED, NONZERO INTEGER.  ACTION CODE                 * IPDG0500
*      100 AFTER THE K OPERATOR CHECKS TO SEE THAT                     * IPDG0510
*      THE NUMERIC CONSTANT FOUND BY THE K OPERATOR                    * IPDG0520
*      WAS A NON-ZERO INTEGER.                                         * IPDG0530
*                                                                      * IPDG0540
ASSIGNMENT  =  < '=' : | '(' &= < N ( ',' N ... ) ')=' : $200  |       *IPDG0550
               ARITHEXP2 ( ','    ARITHEXP2 ... )  ')=' : $202  > >    *IPDG0560
               *7   < ARITHEXP | LOGICEXP >                             IPDG0570
*                                                                      * IPDG0580
*      DEFINES TWO CLASSES OF STATEMENTS                               * IPDG0590
*                                                                      * IPDG0600
*           A.   ARITHMETIC AND LOGICAL ASSIGNMENT STATEMENTS          * IPDG0610
*                                                                      * IPDG0620
*           B.   ARITHMETIC AND LOGICAL STATEMENT FUNCTION DEFINITIONS * IPDG0630
*                                                                      * IPDG0640
*      A VALID SYMBOLIC NAME HAS BEEN FOUND BEFORE                     * IPDG0650
*      THIS LINE IS INVOKED, SO THE SYNTAX OF THE                      * IPDG0660
*      PART OF THE ASSIGNMENT BEFORE THE EQUALS                        * IPDG0670
*      SIGN IS ONE OF:                                                 * IPDG0680
*                                                                      * IPDG0690
*           1.   A NAME                                                * IPDG0700
*                                                                      * IPDG0710
*           2.   A NAME FOLLOWED BY A PARENTHESIZED LIST OF NAMES      * IPDG0720
*                                                                      * IPDG0730
*           3.   A NAME FOLLOWED BY A PARENTHESIZED LIST OF            * IPDG0740
*                EXPRESSIONS, AT LEAST ONE OF WHICH IS NOT             * IPDG0750
*              SIMPLY A NAME                                           * IPD 0760
*                                                                      * IPDG0770
*      IN CASES 1 AND 3, THE STATEMENT IS IN                           * IPDG0780
*      CLASS A, SINCE CLASS B STATEMENTS MUST                          * IPDG0790
*      HAVE AT LEAST ONE NAME IN PARENTHESES                           * IPDG0800
*      BEFORE THE EQUALS SIGN, AND NO EXPRESSION                       * IPDG0810
*      EXCEPT A NAME IS PERMITTED IN THE PARENTHESES                   * IPDG0820
*      IN CLASS B STATEMENTS.  THEREFORE, IN                           * IPDG0830
*      CASE 3, ACTION CODE 202 IS USED TO CHECK                        * IPDG0840
*      FOR MORE THAN SEVEN SUBSCRIPTS.  ACTION                         * IPDG0850
*      CODE 202 ISSUES A "TOO MANY SUBSCRIPTS PRECEDE"                 * IPDG0860
*      MESSAGE IF THERE WERE MORE THAN SEVEN                           * IPDG0870
*      EXPRESSIONS.                                                    * IPDG0880
*                                                                      * IPDG0890
*      IN CASE 2, THE STATEMENT COULD BE IN                            * IPDG0900
*      EITHER CLASS A OR CLASS B, AND SO, IF                           * IPDG0910
*      MORE THAN SEVEN NAMES ARE PRESENT,                              * IPDG0920
*      A "POSSIBLY TOO MANY SUBSCRIPTS PRECEDE"  MESSAGE               * IPDG0930
*      IS ISSUED BY ACTION CODE 200.                                   * IPDG0940
*                                                                      * IPDG0950
*      IF THE STATEMENT IS NOT CASE 1, IT                              * IPDG0960
*      IS SCANNED TO SEE WHETHER IT CONTAINS                           * IPDG0970
*      AN EQUALS SIGN SOMEWHERE TO THE RIGHT                           * IPDG0980
*      OF THE INITIAL NAME.  ASSIGNMENT                                * IPDG0990
*      FAILS IF AN EQUAL SIGN IS NOT FOUND.                            * IPDG1000
*      UNLESS A HOLLERITH FIELD CONTAINS THE                           * IPDG1010
*      EQUAL SIGN THAT SATISFIES THE SCANNING                          * IPDG1020
*      OPERATION, THIS TEST AVOIDS ANALYSIS                            * IPDG1030
*      OF A PARENTHESIZED FORM ( IN SUCH                               * IPDG1040
*      STATEMENTS AS FORMAT AND IF ) BY THE                            * IPDG1050
*      ASSIGNMENT SYNTACTIC LINE, WHEN THERE IS                        * IPDG1060
*      NO POSSIBITITY THAT THE STATEMENT IS AN ASSIGNMENT.             * IPDG1070
*      WHEN AN EQUALS SIGN IS FOUND IN THE                             * IPDG1080
*      PROPER PLACE, THE STATEMENT IS COMMITTED.                       * IPDG1090
*                                                                      * IPDG1100
*      THE SYNTAX TO THE RIGHT OF THE EQUALS                           * IPDG1110
*      IS THE SAME FOR CLASSES A AND B, EVEN                           * IPDG1120
*      THOUGH CLASS B DOES NOT ALLOW REFERENCES                        * IPDG1130
*      TO SUBSCRIPTED VARIABLES IN THE EXPRESSION.                     * IPDG1140
*      THIS IS BECAUSE THE SYNTAX CHECKER DOES NOT                     * IPDG1150
*      HAVE THE INFORMATION THAT WOULD ENABLE IT TO                    * IPDG1160
*      DETERMINE THAT A NAME FOLLOWED BY A                             * IPDG1170
*      PARENTHESIZED LIST OF EXPRESSIONS                               * IPDG1180
*      WAS AN ARRAY ELEMENT REFERENCE AND                              * IPDG1190
*      NOT A FUNCTION REFERENCE.  THE                                  * IPDG1200
*      SYNTAX CHECKER WOULD HAVE TO SAVE                               * IPDG1210
*      INFORMATION FROM DIMENSION AND OTHER                            * IPDG1220
*      ARRAY-DECLARING STATEMENTS TO MAKE                              * IPDG1230
*      THE DISTINCTION, AND THE SYNTAX CHECKER                         * IPDG1240
*      DOES NOT SAVE SUCH INFORMATION.                                 * IPDG1250
*                                                                      * IPDG1260
ARITHEXP2 =    (  <  '+'  |  '-'  >  )   OPERANDA2  *55                     *IPDG1270
               (   +ARITHOP  /  OPERANDA2   ...    ) ¬'.'                    IPDG1280
*                                                                      * IPDG1290
*      THIS STATEMENT DEFINES ARITHMETIC EXPRESSIONS                   * IPDG1300
*      OF TYPE REAL OR INTEGER, BUT NOT COMPLEX.                       * IPDG1310
*      ANY ARITHMETIC EXPRESSION WHICH DOES NOT                        * IPDG1320
*      CONTAIN A COMPLEX, LOGICAL, OR LITERAL                          * IPDG1330
*      CONSTANT (EXCEPT AS AN ARGUMENT OF A                            * IPDG1340
*      FUNCTION REFERENCE) WILL SATISFY THIS                           * IPDG1350
*      DEFINITION.  THE SYNTAX CHECKER ASSUMES                         * IPDG1360
*      THAT ANY SYMBOLIC NAME IS OF THE CORRECT                        * IPDG1370
*      TYPE, SINCE IT HAS NO WAY OF CHECKING                           * IPDG1380
*      THE TYPE OF A SYMBOLIC NAME.  ARITHEXP2                         * IPDG1390
*      IS USED WHERE AN EXPRESSION CANNOT BE COMPLEX                   * IPDG1400
*      AS IN SUBSCRIPTS OR IN ARITHMETIC IF STATEMENTS.                * IPDG1410
*                                                                      * IPDG1420
OPERANDA2 =    <   K   |     N   (   '(' /  *7   FUNCACTARG    (        *IPDG1430
               ','   / FUNCACTARG  ...  )  $200   *12 ')'    )         *IPDG1440
               |   '('   ARITHEXP2   /     *12 ')'  >                    IPDG1450
*                                                                      * IPDG1460
*      DEFINES NON-COMPLEX OPERANDS FOR ARITHEXP2.                     * IPDG1470
*      THE OPTIONAL PARENTHESIZED LIST AFTER A NAME                    * IPDG1480
*      MAY BE A LIST OF SUBSCRIPTS OR A LIST OF                        * IPDG1490
*      FUNCTION ACTUAL ARGUMENTS.  HOWEVER, SINCE                      * IPDG1500
*      THE SYNTAX CHECKER CANNOT DISTINGUISH                           * IPDG1510
*      BETWEEN ARRAY ELEMENT REFERENCES AND FUNCTION                   * IPDG1520
*      REFERENCES, THE LIST IS TREATED AS A                            * IPDG1530
*      LIST OF FUNCTION ACTUAL ARGUMENTS.  THE                         * IPDG1540
*      PERMISSIBLE FORMS FOR SUBSCRIPTS ARE                            * IPDG1550
*      A SUBSET OF THOSE FOR FUNCTION ACTUAL                           * IPDG1560
*      ARGUMENTS, SO THE SYNTAX CHECKER EXCLUDES                       * IPDG1570
*      NO PERMISSIBLE FORMS.                                           * IPDG1580
*                                                                      * IPDG1590
*      THE FORM "ARITHMETIC EXPRESSION IN PARENTHESES"                 * IPDG1600
*      CANNOT BE COMMITTED UNTIL AFTER THE ARITHMETIC                  * IPDG1610
*      EXPRESSION IS FOUND.  THIS IS BECAUSE                              * IPDG1620
*      THERE ARE CASES IN WHICH IT WOULD NOT                              * IPDG1630
*      BE AN ERROR IF THE EXPRESSION IN THE PARENTHESES                   * IPDG1640
*      IS A LOGICAL EXPRESSION.  THIS POSSIBILITY                         * IPDG1650
*      ARISES IN ANY PLACE WHERE EITHER AN                                * IPDG1660
*      ARITHMETIC EXPRESSION OR A LOGICAL EXPRESSION                      * IPDG1670
*      IS PERMITTED, FOR EXAMPLE, IN ACTUAL                               * IPDG1680
*      ARGUMENT LISTS AND IN IF STATEMENTS.  IN                           * IPDG1690
*      ALL THESE CASES, THE SOURCE STATEMENT IS                           * IPDG1700
*      CHECKED FOR THE ARITHMETIC FORM FIRST,                             * IPDG1710
*      THEN THE LOGICAL, SINCE ARITHMETIC                                 * IPDG1720
*      EXPRESSIONS ARE MORE COMMON THAN LOGICAL                           * IPDG1730
*      EXPRESSIONS.  IF, FOR EXAMPLE, AN EXPRESSION                       * IPDG1740
*      IN ONE OF THESE PLACES WERE OF THE FORM                            * IPDG1750
*                                                                         * IPDG1760
*           (A.GT.B)                                                      * IPDG1770
*                                                                         * IPDG1780
*      WHICH IS A VALID FORM, A COMMIT BEFORE ARITHEXP2                   * IPDG1790
*      ON THE THIRD LINE WOULD CAUSE A SPURIOUS                           * IPDG1800
*      ERROR MESSAGE TO BE ISSUED.                                        * IPDG1810
*                                                                         * IPDG1820
ARITHOP    =      "   '+' 0     '-' 0     '/' 0     '**' 0     '*' 0      "  IPDG1830
*                                                                         * IPDG1840
*      TABLE OF THE ARITHMETIC OPERATORS.  THE                            * IPDG1850
*      DOUBLE ASTERISK MUST PRECEDE THE SINGLE                            * IPDG1860
*      ASTERISK SO THAT A SPURIOUS MATCH ON                               * IPDG1870
*      "SINGLE ASTERISK" WILL NOT OCCUR WHEN THE                          * IPDG1880
*      SOURCE STATEMENT CONTAINS A DOUBLE                                 * IPDG1890
*      ASTERISK.                                                          * IPDG1900
*                                                                         * IPDG1910
FUNCACTARG  =  <  ARITHEXP  |  LOGICEXP  |  C  |  H  >                      IPDG1920
*                                                                         * IPDG1930
*      DEFINITION OF THE FORMS THAT MAY APPEAR                            * IPDG1940
*      AS ACTUAL ARGUMENTS IN A FUNCTION                                  * IPDG1950
*      REFERENCE.  THESE ARE VALID FORMS FOR                              * IPDG1960
*      FUNCTION ACTUAL ARGUMENTS REGARDLESS OF                            * IPDG1970
*      THE TYPE OF THE EXPRESSION IN WHICH THE                            * IPDG1980
*      FUNCTION REFERENCE OCCURS.                                         * IPDG1990
*                                                                         * IPDG2000
ARITHEXP   =      (  <  '+'  |  '-'  >  )  *55  OPERANDA  (  <             *IPDG2010
                  '**'  /   OPERANDA2   |   +ARITHOP  /   OPERANDA         *IPDG2020
                  >  ...  )  ¬'.'                                           IPDG2030
*                                                                         * IPDG2040
*      DEFINES THE MOST GENERAL FORM OF ARITHMETIC                        * IPDG2050
*      EXPRESSION.  THE OPERANDS WHICH ARE                                * IPDG2060
*      CONSTANTS MAY BE OF ANY NUMERIC TYPE,                              * IPDG2070
*      EXCEPT THAT OPERANDS WHICH FOLLOW THE                              * IPDG2080
*      EXPONENTIATION OPERATOR MUST                                       * IPDG2090
*      BE OF TYPE REAL OR INTEGER.  SINCE                                 * IPDG2100
*      THIS DEFINITION EXPLICITLY CHECKS FOR THE                          * IPDG2110
*      EXPONENTIATION OPERATOR BEFORE USING THE                           * IPDG2120
*      ARITHOP TABLE, A MATCH TO THE                                      * IPDG2130
*      EXPONENTIATION OPERATOR IN THE TABLE WILL                          * IPDG2140
*      NOT OCCUR.                                                         * IPDG2150
*                                                                         * IPDG2152
*      IF A PERIOD OCCURS AFTER AN ARITHMETIC                             * IPDG2154
*      EXPRESSION, THE EXPRESSION WAS PROBABLY                            * IPDG2156
*      THE FIRST PART OF A LOGICAL EXPRESSION.                            * IPDG2158
*      THE  ¬'.'  AT THE END OF THIS DEFINITION                           * IPDG2160
*      CAUSES IT TO FAIL IN SUCH CASES.                                   * IPDG2162
*                                                                         * IPDG2164
OPERANDA   =      <   K   |   N   (   '('  /   *7   FUNCACTARG   (         *IPDG2170
                  ','   /  FUNCACTARG  ...  )   $200   *12  ')'   )    |   *IPDG2180
                  '('  (  <  '+'  |  '-'  >  )  K  $103  ','  /            *IPDG2190
                  (  <  '+'  |  '-'  >  )  K  $104  *12  ')'   |           *IPDG2195
                  '('  ARITHEXP   /   *12   ')'   >                          IPDG2200
*                                                                         * IPDG2210
*      DEFINES OPERANDS OF ANY NUMERIC TYPE                               * IPDG2220
*      INCLUDING COMPLEX.  ACTION CODES 103                               * IPDG2230
*      AND 104 ARE USED TO CHECK THAT THE                                 * IPDG2240
*      TWO NUMERIC CONSTANTS WHICH FORM A                                 * IPDG2250
*      COMPLEX CONSTANT AGREE IN LENGTH.  THE                             * IPDG2260
*      FORM "ARITHMETIC EXPRESSION IN PARENTHESES"                        * IPDG2270
*      CANNOT BE COMMITTED UNTIL AFTER THE ARITHMETIC                     * IPDG2280
*      EXPRESSION IS FOUND, FOR THE REASON GIVEN IN                       * IPDG2290
*      THE DISCUSSION OF OPERANDA2.                                       * IPDG2300
*                                                                         * IPDG2310
LOGICEXP   =      (  '.NOT.'  )    OPERANDL    *57    (  +LOGOP    /       *IPDG2320
                  OPERANDL    ...     )                                     IPDG2330
*                                                                         * IPDG2340
*      DEFINES LOGICAL EXPRESSIONS.                                       * IPDG2350
*                                                                         * IPDG2360
OPERANDL    =  <  NAME  -ARITHOP  (  +RELOP  /  *139  ARITHEXP3  )         *IPDG2370
               |  ARITHEXP3  /  *51  +RELOP  *139  ARITHEXP3               *IPDG2380
               |  '('  /  LOGICEXP  *12  ')'                               *IPDG2390
               |  '.TRUE.'  |  '.FALSE.'   >                                IPDG2400
```

```
*                                                                     * IPDG2410
*     DEFINES THE OPERANDS THAT CAN APPEAR IN                         * IPDG2420
*     LOGICAL EXPRESSIONS.  NOTE THAT THE COMMIT IN                   * IPDG2430
*     THE FORM "LOGICAL EXPRESSION IN PARENTHESES"                    * IPDG2440
*     PRECEDES THE EXPRESSION.  THIS COMMIT IS                        * IPDG2450
*     POSSIBLE SINCE ANY SOURCE BEING TESTED                          * IPDG2460
*     AGAINST LOGICAL EXPRESSION HAS ALREADY                          * IPDG2470
*     BEEN TESTED AGAINST ARITHMETIC EXPRESSION                       * IPDG2480
*     IF ARITHMETIC EXPRESSION WAS A POSSIBLE                         * IPDG2490
*     ALTERNATIVE.                                                    * IPDG2500
*                                                                     * IPDG2502
*     WHEN A NAME IS FOUND, THIS DEFINITION TESTS FOR THE             * IPDG2504
*     ABSENCE OF AN ARITHMETIC OPERATOR FOLLOWING IT.  IF             * IPDG2506
*     NO ARITHMETIC OPERATOR FOLLOWS, THE NAME COULD STILL            * IPDG2508
*     BE AN ARITHMETIC EXPRESSION, SO THE DEFINITION TESTS            * IPDG2510
*     FOR A RELATIONAL OPERATOR FOLLOWED BY AN ARITHMETIC             * IPDG2512
*     EXPRESSION AS AN OPTION AFTER THE NAME.  IF AN                  * IPDG2514
*     ARITHMETIC OPERATOR DOES OCCUR AFTER THE NAME, THE              * IPDG2515
*     NAME ALTERNATIVE FAILS, AND THE EXPRESSION IS PROCESSED        * IPDG2516
*     BY ARITHEXP3 IN THE NEXT ALTERNATIVE.  THE RELATIONAL           * IPDG2517
*     OPERATOR IS NOT OPTIONAL IN THIS CASE.                          * IPDG2518
*                                                                     * IPDG2519
LOGOP     =    "  '.AND..NOT.' 0    '.AND.' 0                          *IPDG2520
                  '.OR..NOT.' 0   '.OR.' 0    "                         IPDG2530
*                                                                     * IPDG2540
*     TABLE OF THE LOGICAL OPERATORS.  THE OPERATOR                   * IPDG2550
*     ".NOT." IS NOT A MEMBER OF THIS TABLE BECAUSE                   * IPDG2560
*     ALL OF ITS VALID USES ARE ACCOUNTED FOR BY                      * IPDG2570
*     THE OPTIONAL ".NOT." IN LOGICEXP.                               * IPDG2580
*                                                                     * IPDG2590
RELOP     =   "  '.LT.' 0    '.LE.' 0    '.EQ.' 0    '.NE.' 0          *IPDG2600
                 '.GE.' 0    '.GT.' 0    "                              IPDG2610
*                                                                     * IPDG2620
*     TABLE OF THE RELATIONAL OPERATORS.                              * IPDG2630
*                                                                     * IPDG2632
ARITHEXP3 =    ( <  '+'  |  '-'  >  )      OPERANDA2  *55              *IPDG2634
               (   +ARITHOP    /  OPERANDA2   ...   )                   IPDG2636
*                                                                     * IPDG2638
*     DEFINES NON-COMPLEX ARITHMETIC EXPRESSIONS FOR                  * IPDG2640
*     USE IN LOGICAL EXPRESSIONS.  UNLIKE ARITHEXP2,                  * IPDG2642
*     THIS DEFINITION ALLOWS THE ARITHMETIC EXPRESSION                * IPDG2644
*     TO BE FOLLOWED BY A PERIOD.                                     * IPDG2646
*                                                                     * IPDG2648
KEYWORD  =  <  'IF'  /  *31  '('  :  <  ARITHIF  |  /  *17  LOGICEXP   *IPDG2650
           *13  ')'  LOGICIF  >  |  +AFTERIF  |  +OTHERKW  >            IPDG2655
*                                                                     * IPDG2660
*     DEFINES ALL THE STATEMENTS STARTING WITH A KEYWORD              * IPDG2670
*     EXCEPT FOR THE DO STATEMENT.  EXCEPT FOR THE IF                 * IPDG2672
*     KEYWORD, ALL THE KEYWORDS ARE IN ONE OF THE                     * IPDG2674
*     TWO TABLES AFTERIF AND OTHERKW.  THESE                          * IPDG2676
*     TABLES TRANSFER THE SYNTAX TABLE SCAN TO THE                    * IPDG2678
*     APPROPRIATE SYNTACTIC LINE OR ACTION CODE IF THE FIRST          * IPDG2680
*     AVAILABLE SOURCE CHARACTERS MATCH A KEYWORD.                    * IPDG2682
*                                                                     * IPDG2684
*     THE STATEMENTS STARTING WITH THE IF KEYWORD MUST BE             * IPDG2686
*     HANDLED SPECIALLY SINCE THERE ARE TWO                           * IPDG2690
*     SYNTACTIC FORMS STARTING WITH 'IF(',                            * IPDG2700
*     AND THE REQUIRED DISTINCTION BETWEEN                            * IPDG2710
*     THEM CANNOT BE MADE IF THE 'IF(' KEYWORD                        * IPDG2720
*     IS PLACED IN THE TABLE OF THE KEYWORDS                          * IPDG2730
*     PERMITTED AFTER ONE OF THEM, THE LOGICAL IF.                    * IPDG2740
*     THE 'IF(' STATEMENTS ARE COMMITTED AFTER THE                    * IPDG2742
*     'IF' IN ORDER TO DIAGNOSE A MISSING LEFT                        * IPDG2744
*     PARENTHESIS.                                                    * IPDG2746
*                                                                     * IPDG2750
ARITHIF   =   <  NAME  ')'  <  S  :  |  LOGICIF  >  |  ARITHEXP2  :     *IPDG2760
           *13  ')'  *15  S  >  *53  ','  *43  S  *53  ','  *43  S      IPDG2765
*                                                                     * IPDG2770
*     DEFINES THE ARITHMETIC IF STATEMENT AND A                       * IPDG2775
*     SPECIAL CASE OF THE LOGICAL IF STATEMENT.                       * IPDG2780
*                                                                     * IPDG2785
*     THE SPECIAL CASE OF THE LOGICAL IF STATEMENT CAN                * IPDG2790
*     OCCUR WHEN THE PARENTHESIZED EXPRESSION                         * IPDG2795
*     FOLLOWING THE IF KEYWORD CONSISTS SOLELY                        * IPDG2800
*     OF AN OPTIONALLY SUBSCRIPTED VARIABLE NAME                      * IPDG2805
*     ENCLOSED IN ANY NUMBER OF PARENTHESES.  SINCE                   * IPDG2810
*     SUCH AN EXPRESSION COULD EITHER BE AN ARITHMETIC                * IPDG2815
*     EXPRESSION OR A LOGICAL EXPRESSION, THIS LINE                   * IPDG2820
*     CANNOT BE COMMITTED UNTIL A STATEMENT LABEL IS                  * IPDG2825
*     FOUND AFTER THE PARENTHESIZED EXPRESSION.  IF NO                * IPDG2830
*     STATEMENT LABEL IS FOUND, THE IF IS ASSUMED                     * IPDG2835
*     TO BE LOGICAL, AND NESTING TO THE LOGICIF LINE                  * IPDG2840
*     OCCURS.  (LOGICIF BEGINS WITH A STATEMENT COMMIT.)              * IPDG2845
*                                                                     * IPDG2850
*     BECAUSE THE SPECIAL CASE IS TRIED FIRST, ANY                    * IPDG2855
*     EXPRESSION WHICH SATISFIES ARITHEXP2 WILL                       * IPDG2860
*     CONTAIN AT LEAST ONE NUMERIC CONSTANT OR ARITHMETIC             * IPDG2865
*     OPERATOR.  FURTHER, THE ¬'.' AT THE END OF ARITHEXP2            * IPDG2870
*     ASSURES THAT IT IS NOT FOLLOWED BY A RELATIONAL                 * IPDG2875
*     OPERATOR.  THUS, THE STATEMENT COMMIT CAN OCCUR                 * IPDG2880
*     IMMEDIATELY AFTER THE REFERENCE TO ARITHEXP2.                   * IPDG2881
*                                                                     * IPDG2882
NAME      =  <  N  (  '('  /  *7  FUNCACTARG  (  ','  /                *IPDG2884
              FUNCACTARG  ...  )  $200  *12  ')'  )  |                 *IPDG2886
              '('  NAME  ')'  >                                         IPDG2888
*                                                                     * IPDG2890
*     DEFINES AN OPTIONALLY SUBSCRIPTED VARIABLE NAME                 * IPDG2892
*     ENCLOSED BY ANY NUMBER OF PAIRS OF PARENTHESES.                 * IPDG2894
*                                                                     * IPDG2896
LOGICIF   =  :  <  'DO'  DO2  /  *23  $801  |  M  ARITHASG  |          *IPDG2900
            +AFTERIF  |  'IF'  /  *19  '('  *147  ARITHEXP2  ')'  S    *IPDG2910
            *53  ','  *43  S  *53  ','  *43  S  |  /  *23  -OTHERKW    *IPDG2915
            *19  N  ARITHASG  >                                         IPDG2920
*                                                                     * IPDG2930
*     DEFINES THE PART OF THE LOGICAL IF STATEMENT TO THE             * IPDG2940
*     RIGHT OF THE PARENTHESIZED EXPRESSION WHICH FOLLOWS             * IPDG2950
*     THE IF KEYWORD.  THIS LINE IS SIMILAR TO THE FIRST              * IPDG2960
*     SYNTACTIC LINE (IPDAGH) OF THE DEFINITION, BUT NOT              * IPDG2970
*     IDENTICAL, SINCE THERE ARE RESTRICTIONS ON THE                  * IPDG2980
*     TYPE OF STATEMENT AFTER THE  IF(EXPRESSION)  PART OF            * IPDG2990
*     A LOGICAL IF.                                                   * IPDG3000
*                                                                     * IPDG3010
*     THE STATEMENT IS FIRST EXAMINED TO SEE WHETHER IT IS            * IPDG3020
*     A DO STATEMENT, JUST AS ON LINE IPDAGH.  HOWEVER,               * IPDG3030
*     SINCE DO STATEMENTS ARE INVALID, ACTION CODE 801 IS             * IPDG3040
*     USED TO ISSUE MESSAGE 23 IF DO2 PRODUCES A T.                   * IPDG3050
*                                                                     * IPDG3060
*     THE NEXT ALTERNATIVE IS  M ARITHASG, JUST AS IN IPDAGH.         * IPDG3070
*     THE THIRD AND FOURTH ALTERNATIVES CORRESPOND TO                 * IPDG3080
*     THE THIRD ALTERNATIVE OF IPDAGH: THEY DEFINE ALL                * IPDG3090
*     THE KEYWORD STATEMENTS VALID AFTER A LOGICAL IF.                * IPDG3100
*     THE IF ALTERNATIVE DESCRIBES ONLY THE ARITHMETIC                * IPDG3110
*     IF, SINCE A LOGICAL IF CANNOT FOLLOW A LOGICAL IF.              * IPDG3120
*                                                                     * IPDG3130
*     FINALLY, THE FIFTH ALTERNATIVE CHECKS THE NEXT AVAILABLE        * IPDG3140
*     SOURCE AGAINST THE TABLE OF KEYWORDS WHICH CANNOT               * IPDG3150
*     FOLLOW A LOGICAL IF.  IF THE AVAILABLE SOURCE MATCHES           * IPDG3160
*     ONE OF THESE KEYWORDS, MESSAGE 23 IS ISSUED.                    * IPDG3170
*     OTHERWISE,  N ARITHASG  IS TRIED, AND MESSAGE 19                * IPDG3180
*     IS ISSUED IF THAT FAILS.                                        * IPDG3190
*                                                                     * IPDG3290
DO2   =  S  (  ','  )  N  '='  <  N  |  USNZINT  >  ','                 IPDG3291
*                                                                     * IPDG3292
*     DEFINES THE SYNTAX OF THE BEGINNING OF A DO                     * IPDG3293
*     STATEMENT FOR USE IN DIAGNOSING ITS PRESENCE                    * IPDG3294
*     AFTER A LOGICAL IF.  THERE IS NO STATEMENT                      * IPDG3295
*     COMMIT ON THIS LINE SO THAT IT CAN UNNEST                       * IPDG3296
*     BACK TO THE LOGICIF LINE WHICH WILL ISSUE A                     * IPDG3297
*     MESSAGE IF THIS LINE PRODUCES A 'T' .                           * IPDG3298
*                                                                     * IPDG3299
ARITHASG =  <  '=' : |  &=  '(' ARITHEXP2 ( ','  ARITHEXP2 ... )       *IPDG3300
              ')='  : $202 >   *7   <  ARITHEXP | LOGICEXP  >           IPDG3310
*                                                                     * IPDG3320
*     DEFINES THAT PORTION OF ASSIGNMENT THAT                         * IPDG3330
*     MAY APPEAR AFTER A LOGICAL IF.  SINCE                           * IPDG3340
*     STATEMENT FUNCTION DEFINITIONS CANNOT APPEAR                    * IPDG3350
*     AFTER A LOGICAL IF, THE SECOND ALTERNATIVE                      * IPDG3360
*     OF ASSIGNMENT IS NOT INCLUDED IN THIS                           * IPDG3370
*     DEFINITION, AND THE "TOO MANY SUBSCRIPTS PRECEDE"               * IPDG3380
*     ACTION CODE IS USED.                                            * IPDG3390
*                                                                     * IPDG3400
AFTERIF =  " 'ASSI' ASSIGN      'BACK' BACKSPACE   'CALL' CALL         *IPDG3410
             'CONT' CONTINUE    'ENDF' ENDFILE     'FIND' FIND         *IPDG3420
             'GOTO' GOTO        'PAUS' PAUSE       'PRIN' PRINT        *IPDG3430
             'PUNC' PUNCH       'READ' READ        'RETU' RETURN       *IPDG3440
             'REWI' REWIND      'STOP' STOP        'WRIT' WRITE     "   IPDG3450
*                                                                     * IPDG3460
*     TABLE OF ALL THE KEYWORDS (EXCEPT IF) THAT                      * IPDG3470
*     ARE PERMITTED AFTER A LOGICAL IF.  A REFERENCE                  * IPDG3480
*     TO THIS TABLE CAUSES TRANSFER TO THE APPROPRIATE                * IPDG3490
*     SYNTACTIC LINE IF A MATCH IS FOUND.  WHEN                       * IPDG3500
*     A TRANSFER OCCURS, THE LINE TO WHICH THE                        * IPDG3510
*     TRANSFER IS MADE BEGINS CHECKING WITH THE                       * IPDG3520
*     FIRST CHARACTER AFTER THE CHARACTERS THAT                       * IPDG3530
*     MATCHED THE TABLE ENTRY.                                        * IPDG3540
*                                                                     * IPDG3550
ASSIGN  = 'GN'  :  *42   S  *44  'TO'  *33 N                            IPDG3560
*                                                                     * IPDG3570
*     DEFINES THE ASSIGN STATEMENT.                                   * IPDG3580
*                                                                     * IPDG3590
BACKSPACE = 'SPACE'  :  DSREFNO                                         IPDG3600
*                                                                     * IPDG3610
*     DEFINES THE BACKSPACE STATEMENT.                                 * IPDG3620
*                                                                      * IPDG3630
DSREFNO   =  *27   <  N  |  K  /  $105  >                                IPDG3640
*                                                                      * IPDG3650
*     DEFINES DATA SET REFERENCE NUMBER.                               * IPDG3660
*     ACTION CODE 105 ISSUES AN APPROPRIATE                            * IPDG3670
*     MESSAGE IF THE K ALTERNATIVE ENCOUNTERS                          * IPDG3680
*     ANY NUMERIC CONSTANT OTHER THAN A NON-ZERO                       * IPDG3690
*     INTEGER LESS THAN OR EQUAL TO 99.                                * IPDG3700
*                                                                      * IPDG3710
CALL    =   :  *33  N   (  '('  /  *46  CALLARG     (  ',' /            *IPDG3720
           CALLARG   ...   )    *13 ')'     )                            IPDG3730
*                                                                      * IPDG3740
*     DEFINES THE CALL STATEMENT.                                      * IPDG3750
*                                                                      * IPDG3760
CALLARG   = < ARITHEXP |  LOGICEXP |  C |  H |  '&&' /  *42  S  >        IPDG3770
*                                                                      * IPDG3780
*     DEFINES THE FORMS PERMITTED FOR AN ACTUAL                        * IPDG3790
*     ARGUMENT IN A CALL STATEMENT.                                    * IPDG3800
*                                                                      * IPDG3810
CONTINUE  = 'INUE'   :                                                   IPDG3820
*                                                                      * IPDG3830
*     DEFINES THE CONTINUE STATEMENT                                   * IPDG3840
*                                                                      * IPDG3850
ENDFILE   = 'ILE'    :  DSREFNO                                          IPDG3860
*                                                                      * IPDG3870
*     DEFINES THE ENDFILE STATEMENT.                                   * IPDG3880
*                                                                      * IPDG3890
FIND    =   :  *30  '('  DSREFNO  *61  ''''  *7  INTEGEXP  *13 ')'       IPDAGH
*                                                                      * IPDG3910
*     DEFINES THE FIND STATEMENT.  THE FOUR                            * IPDG3920
*     QUOTATION MARKS REPRESENT A LITERAL CONSISTING                   * IPDG3930
*     OF ONE QUOTE IN THE SOURCE.                                      * IPDG3940
*                                                                      * IPDAGH
*     PTM2770 FIX (RELEASE 19) CHANGED ARITHEXP2 TO INTEGEXP SO ******* IPDAGH
*     THAT REAL CONSTANTS WILL BE FOUND IN ERROR.  ******************** IPDAGH
*                                                                      * IPDG3950
GOTO     =  :  <  '(' /   *43   S  (  ','  /  S  ...  )  *13  ')'       *IPDG3960
               *52 ','    *33 N   |    N  /  *52  ','  *30  '('         *IPDG3970
               *43   S   (  ','  /  S  ...  )  *13  ')'   |   /         *IPDG3980
               *43   S  >                                                IPDG3990
*                                                                      * IPDG4000
*     DEFINES THE THREE KINDS OF GOTO STATEMENT.                       * IPDG4010
*     THESE ARE DEFINED IN THE ORDER: COMPUTED                         * IPDG4020
*     GOTO, ASSIGNED GOTO, UNCONDITIONAL                               * IPDG4030
*     GOTO.  THIS ORDERING ALLOWS A COMMIT                             * IPDG4040
*     TO PRECEDE THE S OPERATOR IN THE DEFINITION OF                   * IPDG4050
*     THE UNCONDITIONAL GOTO.                                          * IPDG4060
*                                                                      * IPDG4070
PAUSE      =   'E'  :  <  C  |  (  D  .5.  )  >  *129  $800              IPDG4080
*                                                                      * IPDG4090
*     DEFINES THE PAUSE STATEMENT.                                     * IPDG4100
*                                                                      * IPDG4160
PRINT   =  'T'   OLDIO                                                   IPDG4170
*                                                                      * IPDG4180
*     DEFINES THE PRINT STATEMENT.                                     * IPDG4190
*                                                                      * IPDG4200
OLDIO    =  :  *133  <  S  |  '*'   /  $401  |  N  >                    *IPDG4210
           (  ',' /  IOLIST    )                                         IPDG4215
*                                                                      * IPDG4220
*     DEFINES THE SYNTAX TO THE RIGHT OF THE KEYWORD                   * IPDG4230
*     FOR PRINT, PUNCH, AND THE OLD FORM OF READ.                      * IPDG4240
*                                                                      * IPDG4250
IOLIST  =   *58    <   IOVAR  |  PARENLIST >       (   ','    /         *IPDG4260
            <     IOVAR  |     PARENLIST  >        ...        )          IPDG4270
*                                                                      * IPDG4280
*     DEFINES AN INPUT/OUTPUT LIST.                                    * IPDG4290
*                                                                      * IPDG4300
IOVAR   =   N   $600  (   '('  /   $601    *7   ARITHEXP2   (  ','      *IPDG4310
            /   $201  ARITHEXP2 ...    )     *12   ')'   )               IPDG4320
*                                                                      * IPDG4330
*     DEFINES THE ITEMS WHICH MAKE UP INPUT/OUTPUT                     * IPDG4340
*     LISTS.  ACTION CODES 600 AND 601 ARE USED TO                     * IPDG4350
*     SET A FLAG THAT CAN BE TESTED LATER TO DETERMINE                 * IPDG4360
*     WHETHER THE LAST INPUT/OUTPUT VARIABLE                           * IPDG4370
*     WAS SUBSCRIPTED.  ACTION CODE 600 SETS                           * IPDG4380
*     THIS FLAG TO "UNSUBSCRIPTED", AND ACTION CODE                    * IPDG4390
*     601 SETS IT TO "SUBSCRIPTED".  ACTION CODE                       * IPDG4400
*     201 TESTS FOR TOO MANY SUBSCRIPTS.                               * IPDG4410
*                                                                      * IPDG4420
PARENLIST =    '(' / *58 <  IOVAR   |   PARENLIST    >    (  ','        *IPDG4430
            /  *32  <  IOVAR (  '='    /  $602  <  N  |  /              *IPDG4440
            USNZINT > *52 ','  <  N   |  /  USNZINT  >  (  ','  <       *IPDG4450
            N   |    /   USNZINT   >    )   $603   )   |                *IPDG4460
            PARENLIST   >     ...    )      *12  ')'                     IPDG4470
*                                                                      * IPDG4480
*     DEFINES THE PARENTHESIZED LIST THAT MAY BE                     * IPDG4490
*     A MEMBER OF AN INPUT/OUTPUT LIST.  THIS                        * IPDG4500
*     COMPLICATED LOOKING DEFINITION IS BASICALLY                    * IPDG4510
*     JUST:                                                          * IPDG4520
*                                                                    * IPDG4530
*          PARENLIST = '('  <  IOVAR  .|   PARENLIST  >              * IPDG4540
*                       ( ',' / < IOVAR  |   PARENLIST > ... )  ')'  * IPDG4550
*                                                                    * IPDG4560
*     HOWEVER, THERE IS A LENGTHY OPTION AFTER                       * IPDG4570
*     THE SECOND OCCURRENCE OF IOVAR.  THE OPTION                    * IPDG4580
*     DESCRIBES THE SYNTAX FOUND WHEN THE SOURCE                     * IPDG4590
*     CONTAINS AN IMPLIED DO.  THIS OPTION                           * IPDG4600
*     BEGINS WITH THE LEFT PARENTHESIS ON THE SECOND                 * IPDG4610
*     LINE AND ENDS WITH THE LAST RIGHT                              * IPDG4620
*     PARENTHESIS ON THE FOURTH LINE.                                * IPDG4630
*                                                                    * IPDG4640
*     THE FIRST OCCURRENCE OF IOVAR DOES NOT                         * IPDG4650
*     HAVE THE OPTION, BECAUSE AN IMPLIED                            * IPDG4660
*     DO SPECIFICATION MAY NOT BE THE FIRST                          * IPDG4670
*     ITEM INSIDE A PARENTHESIS.  IF AN EQUAL                        * IPDG4680
*     SIGN IS ENCOUNTERED AFTER SOME INPUT/OUTPUT                    * IPDG4690
*     VARIABLE AFTER THE FIRST VARIABLE OR PARENTHESIZED             * IPDG4700
*     LIST, THE OPTION IS COMMITTED.  ACTION CODE                    * IPDG4710
*     602 IMMEDIATELY CHECKS THE FLAG SET BY                         * IPDG4720
*     ACTION CODES 600 AND 601 TO SEE WHETHER                        * IPDG4730
*     THE VARIABLE PRECEDING THE EQUAL SIGN                          * IPDG4740
*     WAS SUBSCRIPTED.  IF IT WAS, AN APPROPRIATE                    * IPDG4750
*     ERROR MESSAGE IS ISSUED.  THEN THE                             * IPDG4760
*     PARAMETERS OF THE IMPLIED DO ARE CHECKED.                      * IPDG4770
*     THERE MUST BE A PARENTHESIS IMMEDIATELY                        * IPDG4780
*     AFTER AN IMPLIED DO SPECIFICATION.  ACTION                     * IPDG4790
*     CODE 603 CHECKS FOR THIS PARENTHESIS AND ISSUES                * IPDG4800
*     AN APPROPRIATE MESSAGE IF IT IS ABSENT, BUT                    * IPDG4810
*     DOES NOT ADVANCE THE SOURCE POINTER, ALLOWING                  * IPDG4820
*     THE RIGHT PARENTHESIS LITERAL AT THE                           * IPDG4830
*     END OF THE DEFINITION TO BE                                    * IPDG4840
*     MATCHED IF THE RIGHT PARENTHESIS IS                            * IPDG4850
*     PRESENT.  ANY OTHER METHOD OF CHECKING                         * IPDG4860
*     FOR THE RIGHT PARENTHESIS WOULD ADVANCE                        * IPDG4870
*     THE SOURCE POINTER AND CAUSE A FAILURE                         * IPDG4880
*     ON THE RIGHT PARENTHESIS LITERAL AT THE                        * IPDG4890
*     END OF THE DEFINITION.                                         * IPDG4900
*                                                                    * IPDG4910
PUNCH    =   'H'  OLDIO                                                IPDG4920
*                                                                    * IPDG4930
*     DEFINES THE PUNCH STATEMENT.                                   * IPDG4940
*                                                                    * IPDG4950
READ     =        <   ¬'('  OLDIO    |     NEWIO     >                 IPDG4960
*                                                                    * IPDG4970
*     DEFINES READ STATEMENTS.  IF THERE IS NOT                      * IPDG4980
*     A LEFT PARENTHESIS AFTER THE READ, THE                         * IPDG4990
*     STATEMENT IS THE OLD FORM OF READ.                             * IPDG5000
*                                                                    * IPDG5010
NEWIO  =  :    *30  '('  DSREFNO    (  ''''    /  *7  INTEGEXP  )     *IPDAGH
           (   ',' <  S  |  '*'  /  $401  |  N  ¬'='  >  )            *IPDG5030
           (  <  *42  ',END='  /  S    (  ',ERR='  /  S  )            *IPDG5035
           |  ',ERR='  /  S    (  ',END='  /  S  )  >  )   *13  ')'   *IPDG5040
           (   IOLIST   )                                              IPDG5050
*                                                                    * IPDG5060
*     DEFINES THE FORM OF EITHER READ (NEW FORM)                     * IPDG5070
*     OR WRITE AFTER THE KEYWORD.  THIS DEFINITION                   * IPDG5080
*     ENCOMPASSES SEQUENTIAL OR DIRECT ACCESS,                       * IPDG5090
*     FORMATTED OR UNFORMATTED, READ AND WRITE                       * IPDG5100
*     STATEMENTS.  ANY OF THESE STATEMENTS MAY                       * IPDG5110
*     HAVE THE ERR= AND END= PARAMETERS,                             * IPDG5120
*     ALTHOUGH NO INTERPRETATION IS GIVEN EITHER                     * IPDG5130
*     PARAMETER IN ANY WRITE, AND THE END= PARAMETER                 * IPDG5140
*     HAS NO INTERPRETATION IN A DIRECT ACCESS READ.                 * IPDG5150
*     THE IOLIST IS OPTIONAL IN ALL FORMS.                           * IPDG5160
*     THE DEFINITION IS MADE COMPLICATED BY                          * IPDG5170
*     THE FACT THAT WHEN BOTH END= AND                               * IPDG5180
*     ERR= OCCUR, EITHER ONE MAY OCCUR FIRST.                        * IPDG5190
*                                                                    * IPDG5200
*                                                                    * IPDAGH
*    PTM2770 FIX (RELEASE 19)  CHANGED ARITHEXP2 TO INTEGEXP SO ******* IPDAGH
*    THAT REAL CONSTANTS WILL BE FOUND IN ERROR.  ******************** IPDAGH
RETURN  =  'RN'  :  (    <  N   |   USNZINT   >    )                   IPDG5210
*                                                                    * IPDG5220
*     DEFINES THE RETURN STATEMENT.  THE                             * IPDG5230
*     RETURN I FORM IS ALWAYS PERMITTED BECAUSE                      * IPDG5240
*     THE SYNTAX CHECKER HAS NO INFORMATION                          * IPDG5250
*     AVAILABLE REGARDING THE KIND OF PROGRAM                        * IPDG5260
*     UNIT THE RETURN OCCURS IN.                                     * IPDG5270
*                                                                    * IPDG5280
REWIND    = 'ND'    :  DSREFNO                                         IPDG5290
*                                                                    * IPDG5300
*     DEFINES THE REWIND STATEMENT.                                   * IPDG5310
*                                                                     * IPDG5320
STOP      =   :  (  D  .5.  )  *129  $800                               IPDG5330
*                                                                     * IPDG5340
*     DEFINES THE STOP STATEMENT.                                     * IPDG5350
*                                                                     * IPDG5410
WRITE     = 'E'  NEWIO                                                  IPDG5420
*                                                                     * IPDG5430
*     DEFINES THE WRITE STATEMENT.                                    * IPDG5440
*                                                                     * IPDG5450
OTHERKW =  " 'AT' AT              'BLOC' BLOCKDATA   'COMM' COMMON       *IPDG5460
             'COMP' COMPLEX      'DATA' DATA        'DEBU' DEBUG        *IPDG5470
             'DEFI' DEFINEFILE   'DIME' DIMENSION   'DISP' DISPLAY      *IPDG5480
             'DOUB' DOUBLE       'END'  END         'ENTR' ENTRY        *IPDG5490
             'EQUI' EQUIVALENCE 'EXTE' EXTERNAL    'FORM' FORMAT       *IPDG5500
             'FUNC' FUNCTION     'IMPL' IMPLICIT    'INTE' INTEGER      *IPDG5510
             'LOGI' LOGICAL      'NAME' NAMELIST    'REAL' REAL         *IPDG5520
             'SUBR' SUBROUTINE   'TRAC' TRACE       "                   IPDG5530
*                                                                     * IPDG5540
*     TABLE OF ALL KEYWORDS THAT CANNOT FOLLOW                        * IPDG5550
*     A LOGICAL IF.  FOR EACH OF THE ENTRIES,                         * IPDG5560
*     A MATCH WITH THE LITERAL RESULTS IN A                           * IPDG5570
*     TRANSFER TO THE APPROPRIATE SYNTACTIC LINE.                     * IPDG5580
*                                                                     * IPDG5690
AT      =  :   $400 *43   S                                             IPDG5700
*                                                                     * IPDG5710
*     DEFINES THE AT STATEMENT.                                       * IPDG5720
*     SINCE THE AT STATEMENT IS PART OF THE DEBUG                     * IPDG5730
*     FACILITY AVAILABLE ONLY IN FORTRAN G, ACTION                    * IPDG5740
*     CODE 400 IS USED TO CHECK THAT THE                              * IPDG5750
*     SYNTAX DESIRED IS THAT OF FORTRAN G, AND                        * IPDG5760
*     ISSUE AN APPROPRIATE MESSAGE IF THE                             * IPDG5770
*     SYNTAX DESIRED WAS THAT OF FORTRAN H.                           * IPDG5780
*     THE MESSAGE ISSUED IS "DEBUG FACILITY                           * IPDG5790
*     NOT SUPPORTED".                                                 * IPDG5800
*                                                                     * IPDG5810
BLOCKDATA = 'KDATA' :                                                   IPDG5820
*                                                                     * IPDG5830
*     DEFINES THE BLOCK DATA STATEMENT.                               * IPDG5840
*                                                                     * IPDG5850
COMMON  = 'ON' : ( COMMONLABEL ) *33 N  (  DECLARATOR2 )   (   ','      *IPDG5860
           /    N    ( DECLARATOR2 )   ...    )    (  COMMONLABEL      *IPDG5870
           /   N    ( DECLARATOR2 )     (  ','   /  N                  *IPDG5880
           (  DECLARATOR2   )  ...  )   ...  )                          IPDG5890
*                                                                     * IPDG5900
*     DEFINES THE COMMON STATEMENT.                                   * IPDG5910
*                                                                     * IPDG5920
COMMONLABEL =   *38   '/'   /   (  N  )    '/'                          IPDG5930
*                                                                     * IPDG5940
*     DEFINES THE FORM OF THE LABEL OF A COMMON                       * IPDG5950
*     IN A COMMON STATEMENT.  THE NAME WILL                           * IPDG5960
*     BE ABSENT WHEN THE SOURCE IS DESCRIBING                         * IPDG5970
*     BLANK COMMON.                                                   * IPDG5980
*                                                                     * IPDG5990
DECLARATOR2 =   '('  /  USNZINT     (  ','  /  $201  USNZINT           *IPDG6000
                ...  )  *12  ')'                                        IPDG6010
*                                                                     * IPDG6020
*     DEFINES ARRAY DECLARATORS WITH CONSTANT                         * IPDG6030
*     DIMENSIONS.  THIS KIND OF DECLARATOR IS                         * IPDG6040
*     USED IN STATEMENTS (SUCH AS COMMON                              * IPDG6050
*     AND EQUIVALENCE STATEMENTS) WHICH DO                            * IPDG6060
*     NOT PERMIT VARIABLY DIMENSIONED ARRAYS.                         * IPDG6070
*                                                                     * IPDG6080
COMPLEX    =  'LEX'  <  'FUNCTION'  :  *33  N  CLENGTH                 *IPDG6090
              FUNCTIONARGS |  '*'  (  D ...  )     'FUNCTION'     :    *IPDG6093
              *134 $801  *33   N  CLENGTH  FUNCTIONARGS  |             *IPDG6096
              :  CLENGTH *32   N  CLENGTH  ( <  (   DECLARATOR3   )    *IPDG6100
              CDATA  |  DECLARATOR  /  *125 ¬'/'  >  )  (  ','         *IPDG6110
              / *32   N  CLENGTH  (  <  (  DECLARATOR3  )  CDATA       *IPDG6115
              |  DECLARATOR  /  *125  ¬'/'  >  )  ...  )  >             IPDG6120
*                                                                     * IPDG6130
*     DEFINES THE COMPLEX FUNCTION STATEMENT AND                      * IPDG6140
*     THE COMPLEX TYPE-STATEMENT.                                     * IPDG6150
*                                                                     * IPDG6152
*     SINCE DECLARATOR IS TESTED AFTER DECLARATOR3,                   * IPDG6154
*     DECLARATOR WILL BE SATISFIED IF AND ONLY IF                     * IPDG6156
*     THE ARRAY HAS A DUMMY DIMENSION.  IN SUCH A                     * IPDG6158
*     CASE, NO DATA-VALUE-INITIALIZATION LIST IS                      * IPDG6160
*     ALLOWED, AND THE ¬'/' TESTS FOR AND DIAGNOSES                   * IPDG6162
*     THE PRESENCE OF THE START OF SUCH A LIST.                       * IPDG6164
*                                                                     * IPDG6166
CLENGTH   =  (  '*'  <  '16'  |  '8'  |  / *28 $801  ( D ... )  >  )    IPDG6170
*                                                                     * IPDG6180
*     DEFINES LENGTH SPECIFICATIONS VALID FOR COMPLEX TYPE.           * IPDG6190
*                                                                     * IPDG6192
DECLARATOR3 =  '('  USNZINT  (  ','  USNZINT  ...   )  ')'  /  $202     IPDG6194
```

```
*                                                                                    * IPDG6196
*      DEFINES ARRAY DECLARATORS WITH CONSTANT                                       * IPDG6198
*      DIMENSIONS.  THIS DEFINITION IS IDENTICAL                                     * IPDG6200
*      TO DECLARATOR2, EXCEPT THAT NO MESSAGE IS                                     * IPDG6202
*      ISSUED IF A FAILURE OCCURS BEFORE THE FINAL                                   * IPDG6204
*      RIGHT PARENTHESIS OF THE DECLARATOR.                                          * IPDG6206
*                                                                                    * IPDG6208
CDATA       =    '/'  /  (  K  '*'  /  $100  )  CCONSTANT  (  ','  /                 *IPDG6210
                 (  K  '*'  /  $100  )  CCONSTANT  ...  )  *38  '/'                   IPDG6220
*                                                                                    * IPDG6230
*      DEFINES A LIST OF COMPLEX CONSTANTS ENCLOSED IN SLASHES.                      * IPDG6240
*                                                                                    * IPDG6250
CCONSTANT =    *41  <  '('  /  (  <  '-'  |  '+'  >  )  K  $103                      *IPDG6260
               *52  ','  (  <  '-'  |  '+'  >  )  K  $104  *12  ')'                  *IPDG6270
               |  HCHEX  >                                                            IPDG6280
*                                                                                    * IPDG6290
*      DEFINES THE KINDS OF CONSTANTS THAT MAY APPEAR IN                             * IPDG6300
*      COMPLEX TYPE-STATEMENTS IN THE DATA LIST.  THESE                              * IPDG6310
*      ARE:  COMPLEX CONSTANTS, BOTH FORMS OF LITERAL                                * IPDG6320
*      CONSTANT, AND HEXADECIMAL CONSTANTS.                                          * IPDG6330
*                                                                                    * IPDG6340
FUNCTIONARGS   =  *35  '('  <  N  |  '/'  /  *33                                     *IPDG6370
               N   *38   '/'   >    (   ','   /   *35  <  N  |                       *IPDG6380
               '/'  /    *33  N  *38  '/'   >    ...   )   *13   ')'                  IPDG6390
*                                                                                    * IPDG6400
*      DEFINES THE LIST OF DUMMY ARGUMENTS,                                          * IPDG6410
*      INCLUDING THE PARENTHESES WHICH ENCLOSE                                       * IPDG6420
*      THE LIST, IN A FUNCTION STATEMENT.                                            * IPDG6430
*                                                                                    * IPDG6440
DECLARATOR =   *37  '('  /  <  USNZINT    |   N  >  (  ','  /  $201                  *IPDG6550
                 <  USNZINT  |  N  >  ...  )  *12  ')'                                IPDG6560
*                                                                                    * IPDG6570
*      DEFINITION OF ARRAY DECLARATOR.  THIS DEFINITION                              * IPDG6580
*      IS USED WHERE VARIABLY - DIMENSIONED ARRAYS                                   * IPDG6590
*      MAY BE DECLARED.                                                              * IPDG6600
*                                                                                    * IPDG6610
DATALIST =   '/'  /  (  K  '*'  /  $100  )  CONSTANT  (  ','  /                      *IPDG6620
             (  K  '*'  /  $100  )  CONSTANT  ...  )  *38  '/'                        IPDG6630
*                                                                                    * IPDG6640
*      DEFINES THE DATA LISTS THAT MAY APPEAR IN                                     * IPDG6650
*      DATA STATEMENTS.                                                              * IPDG6660
*                                                                                    * IPDG6670
CONSTANT =   *40  <  (  < '+'  |  '-'  >  )  K  |                                    *IPDG6680
             HCHEX  |  '.TRUE.'  |  '.FALSE.'                                        *IPDG6685
             |  'T'  |  'F'  |  '('  /  (  <  '+'  |  '-'  >  )                      *IPDG6690
             K  $103  *52  ','  (  <  '+'  |  '-'  >  )  K  $104                     *IPDG6695
             *12  ')'  >                                                              IPDG6700
*                                                                                    * IPDG6710
*      DEFINES ALL THE TYPES OF CONSTANT THAT ARE                                    * IPDG6720
*      PERMITTED BY FORTRAN.                                                         * IPDG6730
*                                                                                    * IPDG6740
HCHEX   =    <  H  |  C  |  'Z'  /  HEXDIG  (  HEXDIG  ...  )  >                      IPDG6741
*                                                                                      IPDG6745
*      DESCRIBES H-LITERALS, LITERALS, AND HEXADECIMAL                                 IPDG6746
*      CONSTANTS.  THIS LINE IS USED FOR DATA LISTS IN                                 IPDG6747
*      DATA AND TYPE STATEMENTS.                                                       IPDG6748
*                                                                                      IPDG6749
HEXDIG =  <  D  |  'A'  |  'B'  |  'C'  |  'D'  |  'E'  |  'F'  >                     IPDG6750
*                                                                                    * IPDG6760
*      DEFINES A HEXADECIMAL DIGIT.                                                  * IPDG6770
*                                                                                    * IPDG6780
DATA       =    :  *79  VARLIST  *49  DATALIST  (  ','  /  *79                       *IPDG6790
               VARLIST  *49  DATALIST  ...  )                                         IPDG6800
*                                                                                    * IPDG6810
*      DEFINES THE DATA STATEMENT.                                                   * IPDG6820
*                                                                                    * IPDG6830
VARLIST  =  N  ( DECLARATOR2 )  *33  (  ','  /  N  ( DECLARATOR2 )                   *IPDG6840
            ...  )                                                                    IPDG6850
*                                                                                    * IPDG6860
*      DEFINES A LIST OF VARIABLES OF THE KIND                                       * IPDG6870
*      THAT APPEARS IN A DATA STATEMENT.                                             * IPDG6880
*                                                                                    * IPDG6890
DEBUG   = 'G'  :  $400    (  OPTION   (  ','  /  *65  OPTION                         *IPDG6900
          .4.  )  )                                                                   IPDG6910
*                                                                                    * IPDG6920
*      DEFINES THE DEBUG STATEMENT.  ACTION CODE                                     * IPDG6930
*      400 ISSUES A "DEBUG FACILITY NOT SUPPORTED"                                   * IPDG6940
*      MESSAGE IF THE CHECKER IS CHECKING FORTRAN H.                                 * IPDG6950
*      ONLY FIVE OPTIONS ARE ALLOWED IN THE DEBUG                                    * IPDG6960
*      STATEMENT SINCE AT LEAST ONE OPTION WOULD                                     * IPDG6970
*      HAVE BEEN REPEATED IF MORE THAN FIVE                                          * IPDG6980
*      OPTIONS WERE PRESENT.  HOWEVER, NO CHECK                                      * IPDG6990
*      IS MADE FOR REPEATED OPTIONS IF THERE                                         * IPDG7000
*      ARE FIVE OR FEWER OPTIONS PRESENT.                                            * IPDG7010
*                                                                                    * IPDG7020
OPTION  =  <  'TRACE'  |  'SUBTRACE'  |  'UNIT'  /  *30  '('                         *IPDG7030
           DSREFNO  *12  ')'  |  <  'SUBCHK'  |  'INIT'  >   (         *IPDG7040
           '('  /  *32   N   (  ','  /  N  ...  )  *12  ')'  )   >      IPDG7050
*                                                                      * IPDG7060
*     DEFINES THE FIVE OPTIONS THAT MAY APPEAR                         * IPDG7070
*     IN A DEBUG STATEMENT.                                            * IPDG7080
*                                                                      * IPDG7090
DEFINEFILE =  'NEFILE' :  *27  K  $105  *31  '('  USNZINT              *IPDG7100
              *53 ','  USNZINT  ','  *63 < 'L' | 'E' | 'U' >           *IPDG7110
              *53 ','  *33 N  *13 ')'  (  ',' /  *27 K $105  *31       *IPDG7120
              '('  USNZINT  *53  ','  USNZINT  ','  *63                *IPDG7130
              < 'L' | 'E' | 'U' >  *53 ','  *33 N  *13 ')'  ...  )      IPDG7140
*                                                                      * IPDG7150
*     DEFINES THE DEFINE FILE STATEMENT.  IN                           * IPDG7160
*     THIS STATEMENT, THE DATA SET REFERENCE                           * IPDG7170
*     NUMBER CANNOT BE A SYMBOLIC NAME, SO                             * IPDG7180
*     THE K OPERATOR FOLLOWED BY ACTION CODE 105                       * IPDG7190
*     IS USED WHERE DATA SET REFERENCE NUMBERS ARE                     * IPDG7200
*     REQUIRED.  THE FORM OF THE BASIC ELEMENT                         * IPDG7210
*     OF THIS STATEMENT IS GIVEN ON THE FIRST                          * IPDG7220
*     TWO AND-A-HALF LINES.  THE LAST TWO                              * IPDG7230
*     AND-A-HALF LINES DESCRIBE THE OPTIONAL                           * IPDG7240
*     REPETITION OF THIS ELEMENT FOLLOWING A COMMA.                    * IPDG7250
*                                                                      * IPDG7260
DIMENSION = 'NSION' :  *33 N  DECLARATOR  ( ','  /  N  DECLARATOR      *IPDG7270
            ...   )                                                     IPDG7280
*                                                                      * IPDG7290
*     DEFINES THE DIMENSION STATEMENT.  SINCE                          * IPDG7300
*     THE LINE IS COMMITTED AFTER THE LITERAL IS                       * IPDG7310
*     MATCHED, THE "ARRAY DIMENSIONS EXPECTED" MESSAGE                 * IPDG7320
*     ON THE DECLARATOR LINE WILL BE ISSUED IF                         * IPDG7330
*     A DECLARATOR IS MISSING.  THE "NAME EXPECTED"                    * IPDG7340
*     MESSAGE ON THIS LINE THEREFORE APPLIES TO THE                    * IPDG7350
*     ENTIRE LINE.                                                     * IPDG7360
*                                                                      * IPDG7370
DISPLAY = 'LAY'  :  $400  *33  N  (  ','  /  N  ...  )                  IPDG7372
*                                                                      * IPDG7374
*     DEFINES THE DISPLAY STATEMENT.  THIS                             * IPDG7376
*     STATEMENT IS VALID ONLY IN LEVEL G OF                            * IPDG7380
*     FORTRAN.  ACTION CODE 400 DETERMINES                             * IPDG7390
*     WHETHER THE CHECKER IS CHECKING THE G LEVEL                      * IPDG7400
*     OF FORTRAN, AND ISSUES A MESSAGE IF NOT.                         * IPDG7410
*                                                                      * IPDG7420
DOUBLE    =   'LEPRECISION'  <  'FUNCTION'  :  *33  N  FUNCTIONARGS    *IPDG7430
              |  :  *32  N  (  DECLARATOR  )  (  ','  /  N             *IPDG7440
              (  DECLARATOR  )  ...  )  >                               IPDG7450
*                                                                      * IPDG7460
*     DEFINES THE DOUBLE PRECISION TYPE-STATEMENT                      * IPDG7470
*     AND THE DOUBLE PRECISION FUNCTION STATEMENT.                     * IPDG7480
*                                                                      * IPDG7482
END    =  $800   :  $300                                                IPDG7484
*                                                                      * IPDG7486
*     DEFINES THE END LINE.  ACTION CODE 800 PRODUCES AN F             * IPDG7488
*     IF THERE ARE ANY CHARACTERS OTHER THAN BLANKS AFTER THE          * IPDG7490
*     CHARACTERS 'END' WHICH CAUSED NESTING TO THIS LINE.              * IPDG7492
*     IF THERE WERE NO NON-BLANK CHARACTERS AFTER 'END', ACTION        * IPDG7494
*     CODE 800 PRODUCES A T, CAUSING ACTION CODE 300 TO DETECT AND     * IPDG7496
*     DIAGNOSE ANY STATEMENT LABEL OR CONTINUATION FIELD ERRORS.       * IPDG7498
*                                                                      * IPDG7499
ENTRY    =  'Y'   :   SUBORENTRY                                        IPDG7630
*                                                                      * IPDG7640
*     DEFINES THE ENTRY STATEMENT.  THE                                * IPDG7650
*     SYNTAX TO THE RIGHT OF THE KEYWORD IS THE                        * IPDG7660
*     SAME AS THAT OF A SUBROUTINE STATEMENT, SINCE                    * IPDG7670
*     IT IS NOT KNOWN WHETHER THE ENTRY STATEMENT APPEARS              * IPDG7672
*     IN A FUNCTION SUBPROGRAM OR IN A SUBROUTINE SUBPROGRAM.          * IPDG7674
*                                                                      * IPDG7680
SUBORENTRY   =  *33  N  (   '('  /  *35   DUMMYARG                     *IPDG7690
              (   ','  /  DUMMYARG    ...   )   *13   ')'  )            IPDG7700
*                                                                      * IPDG7710
*     DEFINES THE FORM TO THE RIGHT OF THE KEYWORD                     * IPDG7720
*     IN A SUBROUTINE OR AN ENTRY STATEMENT.                           * IPDG7730
*                                                                      * IPDG7740
DUMMYARG  = <  N  |   '/' / *33 N *38  '/'  |  '*'  >                   IPDG7750
*                                                                      * IPDG7760
*     DEFINES DUMMY ARGUMENTS.  DUMMY ARGUMENTS                        * IPDG7770
*     SATISFYING THIS DEFINITION MAY BE USED IN                        * IPDG7780
*     EITHER A SUBROUTINE OR AN ENTRY STATEMENT.                       * IPDG7790
*                                                                      * IPDG7800
EQUIVALENCE =  'VALENCE' :  *30  '('  *33  N  ( DECLARATOR2 )          *IPDG7810
           *53  ','  *33  N  ( DECLARATOR2 )  (  ','  /  N             *IPDG7820
           ( DECLARATOR2 )  ... ) *12 ')'  (  ','  /  *30  '('         *IPDG7830
           *33 N  ( DECLARATOR2 )  *53 ','  *33 N  ( DECLARATOR2 )     *IPDG7840
           (  ','  /  N  ( DECLARATOR2 )  ...  )  *13 ')'  ...  )       IPDG7850
*                                                                      * IPDG7860
*     DEFINES THE EQUIVALENCE STATEMENT.  NONE OF THE                  * IPDG7870
*     DECLARATORS IN THIS STATEMENT MAY CONTAIN A                      * IPDG7880
*     SYMBOLIC NAME INSTEAD OF AN INTEGER CONSTANT.                   * IPDG7890
*     AS IN THE DEFINEFILE DEFINITION, THE FIRST                      * IPDG7900
*     TWO AND-A-HALF LINES OF THIS DEFINITION                         * IPDG7910
*     DESCRIBE THE BASIC FORM.                                        * IPDG7920
*                                                                     * IPDG7930
EXTERNAL =  'RNAL'  :  *33  N   (  ','  /  N   ...  )                   IPDG7940
*                                                                     * IPDG7950
*     DEFINES THE EXTERNAL STATEMENT.                                 * IPDG7960
*                                                                     * IPDG7970
FORMAT  =  'AT'  :  $301   *30  '('  *77  (  '/'  ...  )  (  GROUP     *IPDG7980
           (  <  ','  /  GROUP  |  '/'  (  '/'  ...  )  GROUP  >       *IPDG7990
           ...    )    (  '/'  ...  )    )    ')'                       IPDG8000
*                                                                     * IPDG8010
*     DEFINES THE FORMAT STATEMENT.  ESSENTIALLY,                     * IPDG8020
*     THE DEFINITION IS A PARENTHESIZED LIST OF                       * IPDG8030
*     GROUPS.  (GROUP IS DEFINED ON ANOTHER LINE)                     * IPDG8040
*     EACH DELIMITER IN THE LIST IS EITHER A COMMA                    * IPDG8050
*     OR ANY NUMBER OF SLASHES.  OPTIONALLY, THERE                    * IPDG8060
*     MAY BE ANY NUMBER OF SLASHES BEFORE THE                         * IPDG8070
*     FIRST GROUP IN THE LIST, OR AFTER THE LAST                      * IPDG8080
*     GROUP IN THE LIST, OR BOTH.  THERE                              * IPDG8090
*     NEED NOT BE ANY GROUPS AT ALL.  THE                             * IPDG8100
*     LAST SET OF OPTIONAL SLASHES IS INCLUDED IN                     * IPDG8110
*     THE OPTIONAL PARENTHESES FOR THE LIST                           * IPDG8120
*     OF GROUPS BECAUSE, IF THERE ARE NO                              * IPDG8130
*     GROUPS, THE FIRST SET OF OPTIONAL SLASHES                       * IPDG8140
*     WILL HAVE MATCHED ALL THE VALID CHARACTERS                      * IPDG8150
*     WITHIN THE SOURCE'S PARENTHESES.  THE                           * IPDG8160
*     MESSAGE ISSUED WHEN A RIGHT PARENTHESIS IS                      * IPDG8170
*     NOT FOUND IS "DELIMITER MISSING OR INVALID                      * IPDG8180
*     FORMAT CODE" SINCE ANY FAILURE TO MATCH                         * IPDG8190
*     THE RIGHT PARENTHESIS LITERAL IS PROBABLY                       * IPDG8200
*     DUE TO ONE OF THESE CAUSES.                                     * IPDG8210
*                                                                     * IPDG8220
GROUP   =   <  FIELDESCR  |  (  $700  )  '('  /  (  '/'  ...  )        *IPDG8230
           (  GROUP2  (  <  ','  /  GROUP2  |  '/'  (  '/'  ...  )     *IPDG8240
           GROUP2  >  ...  )  (  '/'  ...  )  )  ')'  >                 IPDG8250
*                                                                     * IPDG8260
*     DEFINES GROUP FOR USE IN THE FORMAT DEFINITION.                 * IPDG8270
*     A GROUP IS EITHER A FIELD DESCRIPTOR OR                         * IPDG8280
*     ANOTHER FORM THAT IS ESSENTIALLY THE SAME AS A                  * IPDG8290
*     FORMAT.  THE DIFFERENCES BETWEEN FORMAT AND                     * IPDG8300
*     THE SECOND FORM ARE  1) THE SECOND FORM OF                      * IPDG8310
*     GROUP MAY HAVE A REPEAT COUNT BEFORE THE                        * IPDG8320
*     INITIAL LEFT PARENTHESIS (ACTION CODE 700                       * IPDG8330
*     ADVANCES THE SOURCE POINTER PAST THIS COUNT                     * IPDG8332
*     IF IT IS PRESENT), AND  2) THE ITEMS                            * IPDG8334
*     IN THE PARENTHESIZED LIST ARE EACH GROUP2                       * IPDG8340
*     INSTEAD OF GROUP.  THE SECOND DIFFERENCE                        * IPDG8350
*     IS NECESSARY TO AVOID ALLOWING AN INDEFINITE NUMBER             * IPDG8360
*     OF LEVELS OF NESTING OF PARENTHESES IN FORMAT                   * IPDG8370
*     STATEMENTS.  FORTRAN ALLOWS ONLY TWO LEVELS                     * IPDG8380
*     OF NESTING INSIDE THE PARENTHESES WHICH ENCLOSE                 * IPDG8390
*     THE ENTIRE FORMAT SPECIFICATION.                                * IPDG8400
*                                                                     * IPDG8410
GROUP2   =  <  FIELDESCR  |  (  $700  )  '('  /  (  <  '/'  |          *IPDG8420
           (  $700  )  '('  /  *69  $801  >  ...  )                    *IPDG8425
           (  FIELDESCR  (  <  ','  /  FIELDESCR  |  '/'  (            *IPDG8430
           <  '/'  |  (  $700  )  '('  /  *69  $801  >  ...  )         *IPD78435
            FIELDESCR  >  ...  )  (  '/'  ...  )  )  ')'  >             IPD78440
*                                                                     * IPDG8450
*     DEFINES GROUP2 FOR USE IN GROUP.  AGAIN,                        * IPDG8460
*     THE SECOND FORM IS ESSENTIALLY THE SAME AS A                    * IPDG8470
*     FORMAT WITH AN OPTIONAL REPEAT SPECIFICATION.                   * IPDG8480
*     HOWEVER, IF THE SECOND ALTERNATIVE IS REACHED, THE              * IPDG8490
*     SOURCE IS ON THE SECOND LEVEL OF PARENTHESIS                    * IPDG8500
*     NESTING, SO ONLY FIELD DESCRIPTORS, AND                         * IPDG8510
*     NOT PARENTHESIZED LISTS, MAY BE MEMBERS                         * IPDG8520
*     OF THE PARENTHESIZED LIST.                                      * IPDG8530
*                                                                     * IPDG8531
*     ACTION CODE 801 IS USED TO ISSUE A MESSAGE                      * IPDG8532
*     DIAGNOSING TOO MANY LEVELS OF PARENTHESES IF ANY                * IPDG8533
*     LEFT PARENTHESIS IS FOUND WITHIN THE PARENTHESES                * IPDG8534
*     WHICH ENCLOSE THE REST OF THE SECOND ALTERNATIVE.               * IPDG8535
*                                                                     * IPDG8540
FIELDESCR  =  <  C  |  $700  'X'  |  (  $700  )                        *IPDG8550
              <  <  'E'  |  'F'  |  'D'  >  /  $700  *80  '.'  $701    *IPDG8555
              |  'G'  /  $700  (  '.'  /  *80  $701  )                 *IPDG8560
              |  <  'I'  |  'A'  |  'L'  |  'Z'  >  /  $700  >         *IPDG8565
              |  H  |  'T'  /  $700                                    *IPDG8570
              |  ( '-' )  < $700 | '0' ( '0' ... ) >  'P'  ( $700 )    *IPDG8575
              <  <  'E'  |  'F'  |  'D'  >  /  $700  *80  '.'  $701    *IPDG8580
              |  'G'  /  $700  (  '.'  /  *80  $701  )  >  >            IPDG8585
*                                                                     * IPDG8600
*     DEFINES ALL THE FIELD DESCRIPTORS WHICH MAY                     * IPDG8610
*     APPEAR IN A FORMAT STATEMENT.                                   * IPDG8620
*                                                                        * IPDG8630
FUNCTION    =  'TION'  :  *33  N  FUNCTIONARGS                             IPDG8631
*                                                                        * IPDG8632
*      DEFINITION OF THE FUNCTION STATEMENT WITH NO                      * IPDG8633
*      LENGTH SPECIFICATION PERMITTED.  USED FOR                         * IPDG8634
*      FUNCTION STATEMENTS NOT PRECEDED BY A TYPE.                       * IPDG8635
*                                                                        * IPDG8636
IMPLICIT  =  'ICIT'  :  *41 +TYPE  *31 '('  $500  (  ','  /  $500        *IPDG8640
             ... )  *13 ')'  (  ','  /  *41 +TYPE  *31 '('  $500         *IPDG8650
             (  ','  /  $500  ...  )  *13 ')'  ...  )                      IPDG8660
*                                                                        * IPDG8670
*      DEFINES THE IMPLICIT STATEMENT.  ACTION CODE                      * IPDG8680
*      500 IS USED FOR THE ELEMENTS OF THE LISTS                         * IPDG8690
*      THAT MAY APPEAR IN IMPLICIT STATEMENTS.  THE                      * IPDG8700
*      ACTION CODE CHECKS FOR THE SYNTAX                                 * IPDG8710
*                                                                        * IPDG8720
*      L ( '-' / L )                                                     * IPDG8730
*                                                                        * IPDG8740
*      THIS COULD BE DONE BY AN ORDINARY SYNTACTIC                       * IPDG8750
*      DEFINITION, BUT THE ACTION CODE PERFORMS AN                       * IPDG8760
*      ADDITIONAL TEST OF THE FORM WITH TWO LETTERS                      * IPDG8770
*      WHICH COULD NOT BE DONE IN ORDINARY SYNTAX.                       * IPDG8780
*      IF THE SECOND LETTER IS NOT LATER IN THE ALPHABETIC               * IPDG8790
*      SEQUENCE THAN THE FIRST, ACTION CODE 500                          * IPDG8800
*      ISSUES AN ERROR MESSAGE.                                          * IPDG8810
*                                                                        * IPDG8820
TYPE    =  "  'REAL'  RLENGTH        'INTEGER'  ILENGTH                  *IPDG8830
              'COMPLEX'  CLENGTH    'LOGICAL'  LLENGTH  "                  IPDG8840
*                                                                        * IPDG8870
*      TABLE DEFINING THE TYPE AND LENGTH SPECIFICATIONS                 * IPDG8880
*      THAT CAN APPEAR IN THE IMPLICIT STATEMENT.                        * IPDG8890
*                                                                        * IPDG8900
INTEGER     =  'GER'  <  'FUNCTION'  :  *33  N  ILENGTH                  *IPDG8905
               FUNCTIONARGS  |   '*'  (  D  ...  )  'FUNCTION'  :        *IPDG8910
               *134  $801  *33  N  ILENGTH  FUNCTIONARGS  |              *IPDG8911
               :  ILENGTH  *32  N  ILENGTH  (  <  (  DECLARATOR3  )      *IPDG8912
               IDATA  |  DECLARATOR  /  *125  ¬'/'  >  )  (  ','         *IPDG8913
               /  *32  N  ILENGTH  (  <  (  DECLARATOR3  )  IDATA        *IPDG8914
               |  DECLARATOR  /  *125  ¬'/'  >  )  ...  )  >               IPDG8915
*                                                                        * IPDG8918
*      DEFINES THE INTEGER FUNCTION STATEMENT AND                        * IPDG8920
*      THE INTEGER TYPE-STATEMENT.                                       * IPDG8922
*                                                                        * IPDG8924
*      SINCE DECLARATOR IS TESTED AFTER DECLARATOR3,                     * IPDG
*      DECLARATOR WILL BE SATISFIED IF AND ONLY IF                       * IPDG
*      THE ARRAY HAS A DUMMY DIMENSION.  IN SUCH A                       * IPDG
*      CASE, NO DATA-VALUE-INITIALIZATION LIST IS                        * IPDG
*      ALLOWED, AND THE ¬'/' TESTS FOR AND DIAGNOSES                     * IPDG
*      THE PRESENCE OF THE START OF SUCH A LIST.                         * IPDG
*                                                                        * IPDG
ILENGTH    =  (  '*'  <  '2'   |  '4'  |  / *28 $801 ( D ... )  >  )       IPDG8926
*                                                                        * IPDG8928
*      DEFINES LENGTH SPECIFICATIONS VALID FOR INTEGER TYPE.             * IPDG8930
*                                                                        * IPDG8932
IDATA      =    '/'  /  (  K  '*'  /  $100  )  ICONSTANT  (  ','  /      *IPDG8934
                (  K  '*'  /  $100  )  ICONSTANT  ...  )  *38  '/'         IPDG8936
*                                                                        * IPDG8938
*      DEFINES A LIST OF INTEGER CONSTANTS ENCLOSED IN SLASHES.          * IPDG8940
*                                                                        * IPDG8942
ICONSTANT = *41  <  HCHEX  |  (  <  '-'  |  '+'  >  )  K  /  $102  >       IPDG8944
*                                                                        * IPDG8948
*      DEFINES THE FORMS OF CONSTANT THAT ARE VALID IN THE               * IPDG8950
*      DATA LIST OF AN INTEGER TYPE-STATEMENT.  THESE ARE:               * IPDG8952
*      INTEGER CONSTANTS, BOTH FORMS OF LITERAL CONSTANT,                * IPDG8954
*      AND HEXADECIMAL CONSTANTS.                                        * IPDG8956
*                                                                        * IPDG8958
LOGICAL     =  'CAL'  <  'FUNCTION'  :  *33  N  LLENGTH                  *IPDG8959
               FUNCTIONARGS  |   '*'  (  D  ...  )  'FUNCTION'  :        *IPDG8960
               *134  $801  *33  N  LLENGTH  FUNCTIONARGS  |              *IPDG8961
               :  LLENGTH  *32  N  LLENGTH  (  <  (  DECLARATOR3  )      *IPDG8962
               LDATA  |  DECLARATOR  /  *125  ¬'/'  >  )  (  ','         *IPDG8963
               /  *32  N  LLENGTH  (  <  (  DECLARATOR3  )  LDATA        *IPDG8964
               |  DECLARATOR  /  *125  ¬'/'  >  )  ...  )  >               IPDG8966
*                                                                        * IPDG8968
*      DEFINES THE LOGICAL FUNCTION STATEMENT AND                        * IPDG8970
*      THE LOGICAL TYPE-STATEMENT.                                       * IPDG8972
*                                                                        * IPDG8974
*      SINCE DECLARATOR IS TESTED AFTER DECLARATOR3,                     * IPDG
*      DECLARATOR WILL BE SATISFIED IF AND ONLY IF                       * IPDG
*      THE ARRAY HAS A DUMMY DIMENSION.  IN SUCH A                       * IPDG
*      CASE, NO DATA-VALUE-INITIALIZATION LIST IS                        * IPDG
*      ALLOWED, AND THE ¬'/' TESTS FOR AND DIAGNOSES                     * IPDG
*      THE PRESENCE OF THE START OF SUCH A LIST.                         * IPDG
*                                                                        * IPDG
LLENGTH    =  (  '*'  <  '1'   |  '4'  |  / *28 $801 ( D ... )  >  )       IPDG8976
*                                                                        * IPDG8978
*     DEFINES LENGTH SPECIFICATIONS VALID FOR LOGICAL TYPE.            * IPDG8980
*                                                                      * IPDG8982
LDATA     =    '/'  /  (   K   '*'   /  $100   )  LCONSTANT   (  ',' /  *IPDG8984
               (  K  '*'  /   $100   )   LCONSTANT  ...   )   *38  '/'   IPDG8986
*                                                                      * IPDG8988
*     DEFINES A LIST OF LOGICAL CONSTANTS ENCLOSED IN SLASHES.         * IPDG8990
*                                                                      * IPDG8992
LCONSTANT =    *41  <  '.TRUE.'  |  '.FALSE.'  |  'T'  |  'F'  |       *IPDG8994
               HCHEX  >                                                  IPDG8996
*                                                                      * IPDG8998
*     DEFINES THE FORMS OF CONSTANT WHICH ARE VALID IN THE             * IPDG9000
*     DATA LIST OF A LOGICAL TYPE-STATEMENT.  THESE ARE:               * IPDG9002
*     LOGICAL CONSTANTS, ABBREVIATED LOGICAL CONSTANTS, BOTH           * IPDG9004
*     FORMS OF LITERAL CONSTANT, AND HEXADECIMAL CONSTANTS.            * IPDG9006
*                                                                      * IPDG9008
NAMELIST =  'LIST'  :  *39  '/'  *33  N   *39  '/'  *32  N             *IPDG9010
            (  ','  /  N  ...  )  (  '/'  /   *33 N  *39  '/'          *IPDG9020
            *32 N  (  ','  /  N  ...  )   ...   )                        IPDG9030
*                                                                      * IPDG9040
*     DEFINES THE NAMELIST STATEMENT.                                  * IPDG9050
*                                                                      * IPDG9060
REAL        =  <  'FUNCTION'  :  *33  N  RLENGTH                       *IPDG9065
               FUNCTIONARGS  |     '*'  (  D  ...  )   'FUNCTION'  :   *IPDG9070
               *134  $801  *33  N  RLENGTH  FUNCTIONARGS  |            *IPDG9071
               :  RLENGTH  *32  N  RLENGTH  (  <  (  DECLARATOR3  )    *IPDG9072
               RDATA  |  DECLARATOR  /  *125  ¬'/'  >  )  (  ','       *IPDG9073
               /  *32  N  RLENGTH  (  <  (  DECLARATOR3  )  RDATA      *IPDG9074
               |  DECLARATOR  /  *125  ¬'/'  >  )  ...  )  >             IPDG9075
*                                                                      * IPDG9078
*     DEFINES THE REAL FUNCTION STATEMENT AND                          * IPDG9080
*     THE REAL TYPE-STATEMENT.                                         * IPDG9082
*                                                                      * IPDG9084
*     SINCE DECLARATOR IS TESTED AFTER DECLARATOR3,                    * IPDG
*     DECLARATOR WILL BE SATISFIED IF AND ONLY IF                      * IPDG
*     THE ARRAY HAS A DUMMY DIMENSION.  IN SUCH A                      * IPDG
*     CASE, NO DATA-VALUE-INITIALIZATION LIST IS                       * IPDG
*     ALLOWED, AND THE ¬'/' TESTS FOR AND DIAGNOSES                    * IPDG
*     THE PRESENCE OF THE START OF SUCH A LIST.                        * IPDG
RLENGTH   =  (  '*'  <  '8'   |  '4'  |  / *28 $801 ( D ... )  >  )      IPDG9086
*                                                                      * IPDG9088
*     DEFINES LENGTH SPECIFICATIONS VALID FOR REAL TYPE.               * IPDG9090
*                                                                      * IPDG9092
RDATA     =    '/'  /  (   K   '*'   /  $100   )  RCONSTANT   (  ',' /  *IPDG9094
               (  K  '*'  /   $100   )   RCONSTANT  ...   )   *38  '/'   IPDG9096
*                                                                      * IPDG9098
*     DEFINES A LIST OF REAL CONSTANTS ENCLOSED IN SLASHES.            * IPDG9100
*                                                                      * IPDG9102
RCONSTANT = *41  <  HCHEX  |  (  <  '-'  |  '+'  >  )  K  /  $106  >     IPDG9104
*                                                                      * IPDG9108
*     DEFINES THE FORMS OF CONSTANT THAT ARE VALID IN THE              * IPDG9110
*     DATA LIST OF A REAL TYPE STATEMENT.  THESE ARE: REAL             * IPDG9112
*     CONSTANTS OF EITHER LENGTH, BOTH FORMS OF LITERAL                * IPDG9114
*     CONSTANT, AND HEXADECIMAL CONSTANTS.                             * IPDG9116
*                                                                      * IPDG9118
SUBROUTINE =  'OUTINE'  :   SUBORENTRY                                   IPDG9130
*                                                                      * IPDG9140
*     DEFINES THE SUBROUTINE STATEMENT.  THE                           * IPDG9150
*     SYNTAX TO THE RIGHT OF THE KEYWORD IS THE                        * IPDG9160
*     SAME AS THAT OF THE ENTRY STATEMENT.                             * IPDG9170
*                                                                      * IPDG9180
TRACE    =  <  'EON'  :  $400  |  'EOFF'  :  $400  >                     IPDG9190
*                                                                      * IPDG9200
*     DEFINES THE TRACEON AND TRACEOFF STATEMENTS.                     * IPDG9210
*     THESE ARE NOT VALID STATEMENTS UNLESS THE SYNTAX                 * IPDG9220
*     CHECKER IS CHECKING AGAINST FORTRAN LEVEL G.                     * IPDG9230
*     ACTION CODE 400 ISSUES A "DEBUG FACILITY NOT                     * IPDG9240
*     SUPPORTED" MESSAGE IF FORTRAN LEVEL H HAS                        * IPDG9250
*     BEEN SPECIFIED.                                                  * IPDG9260
*                                                                      * IPDG9270
INTEGEXP =     (  <  '+'  |  '-'  >  )   OPERANDI  *55                 *IPDAGH
               (    +ARITHOP   /  OPERANDI   ...  )  ¬'.'                IPDAGH
*                                                                      * IPDAGH
*    THIS STATEMENT IS THE SAME AS ARITHEXP2 EXCEPT THAT A REAL        * IPDAGH
*    CONSTANT WILL NOT SATISFY THE DEFINITION.                         * IPDAGH
*    THIS STATEMENT WAS ADDED FOR PTM2770 FIX (RELEASE 19).  ********** IPDAGH
*                                                                      * IPDAGH
OPERANDI =     <  K / $102   |    N   (   '('  /   *7   FUNCACTARG   (  *IPDAGH
               ','   /  FUNCACTARG  ...  )   $200   *12  ')'  )        *IPDAGH
               |  '('  ARITHEXP2   /    *12  ')'    >                    IPDAGH
*                                                                      * IPDAGH
*    THIS STATEMENT IS THE SAME AS OPERANDA2 (BUT DEFINES OPERANDS     * IPDAGH
*    FOR INTEGEXP) EXCEPT THAT THE K OPERATOR WILL FAIL IF A           * IPDAGH
*    CONSTANT IS NOT AN INTEGER.                                       * IPDAGH
*    THIS STATEMENT WAS ADDED FOR PTM2770 FIX (RELEASE 19).  ********** IPDAGH
*                                                                      * IPDAGH


SYNTAX END                                                               IPDG9280
```

INDEX

GY28-6831-2

IBM S/360 FORT. IV Syntax Checker Printed in U.S.A. GY28-6831-2

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)